Text

Description automatically generated

This is an updated version of the previous code that utilizes a task queue to process uploaded images in the background with an AI model and return the distance of the object in frame. Here, we are using Celery as the task queue and Redis as the message broker and result backend.

**from** flask **import** Flask, jsonify, request

**import** os

**from** celery **import** Celery

app **=** Flask(\_\_name\_\_)

app.config['CELERY\_BROKER\_URL'] **=** os.environ.get('CELERY\_BROKER\_URL', 'redis://localhost:6379/0')

app.config['CELERY\_RESULT\_BACKEND'] **=** os.environ.get('CELERY\_RESULT\_BACKEND', 'redis://localhost:6379/0')

celery **=** Celery(app.name, broker**=**app.config['CELERY\_BROKER\_URL'], backend**=**app.config['CELERY\_RESULT\_BACKEND'])

celery.conf.update(app.config)

# Set allowed image extensions

ALLOWED\_EXTENSIONS **=** {'png', 'jpg', 'jpeg'}

# Set secret key for CSRF protection

app.secret\_key **=** os.environ.get('SECRET\_KEY', 'my-secret-key')

**def** allowed\_file(filename):

    """Check if file has allowed extension"""

**return** '.' **in** filename **and** \

           filename.rsplit('.', 1)[1].lower() **in** ALLOWED\_EXTENSIONS

@celery.task

**def** process\_image(img\_bytes):

    """Process uploaded image with AI model and return distance of object in frame"""

    # **TODO**: add AI model code to process image and return distance of object

    distance **=** 10  # Placeholder value for demonstration purposes only

**return** distance

@app.route('/upload', methods**=**['POST'])

**def** upload():

    # Check CSRF token

**if** request.headers.get('X-CSRF-Token') **!=** app.secret\_key:

**return** jsonify({'error': 'Invalid CSRF token.'}), 403

    # Check if image was provided

**if** 'image' **not** **in** request.files:

**return** jsonify({'error': 'No image provided.'}), 400

    image\_file **=** request.files['image']

    # Check if file has allowed extension

**if** **not** allowed\_file(image\_file.filename):

**return** jsonify({'error': 'File must be in PNG, JPG, or JPEG format.'}), 400

    # Read image bytes and process image in the background with AI model to get distance of object in frame

    img\_bytes **=** image\_file.read()

    task **=** process\_image.delay(img\_bytes)

    distance **=** task.get()

    # Return distance of object in frame

**return** jsonify({'distance': distance}), 200

**if** \_\_name\_\_ **==** '\_\_main\_\_':

    app.run()

This code defines a Celery task called **process\_image** that takes the uploaded image bytes as input, processes it with an AI model to get the distance of the object in frame, and returns the result. The **upload** function in the Flask app reads the image bytes from the request, submits a task to the Celery task queue to process the image in the background, and waits for the result. Once the task is completed, the result (i.e., the distance of the object in frame) is returned to the client in a JSON response. This approach allows the server to serve multiple clients simultaneously without being blocked by the processing time of the AI model.

app.config['CELERY\_BROKER\_URL'] **=** os.environ.get('CELERY\_BROKER\_URL', 'redis://localhost:6379/0')

app.config['CELERY\_RESULT\_BACKEND'] **=** os.environ.get('CELERY\_RESULT\_BACKEND', 'redis://localhost:6379/0')

celery **=** Celery(app.name, broker**=**app.config['CELERY\_BROKER\_URL'], backend**=**app.config['CELERY\_RESULT\_BACKEND'])

celery.conf.update(app.config)

These lines of code configure the Celery task queue system to be used with the Flask application.

The **CELERY\_BROKER\_URL** configuration variable is set to the value of the **CELERY\_BROKER\_URL** environment variable if it is defined, or to **'redis://localhost:6379/0'** if it is not defined. This variable specifies the URL of the message broker used by Celery to communicate between the Flask application and the Celery worker processes. In this case, Redis is used as the message broker, which is running on the same machine as the Flask application on the default port 6379 and the default database number 0.

The **CELERY\_RESULT\_BACKEND** configuration variable is set to the value of the **CELERY\_RESULT\_BACKEND** environment variable if it is defined, or to **'redis://localhost:6379/0'** if it is not defined. This variable specifies the URL of the backend used by Celery to store the results of completed tasks. In this case, Redis is also used as the backend.

The **Celery** class is used to create a new Celery application instance, which is named after the Flask application. The broker and backend URLs are passed to the constructor using the **app.config** dictionary.

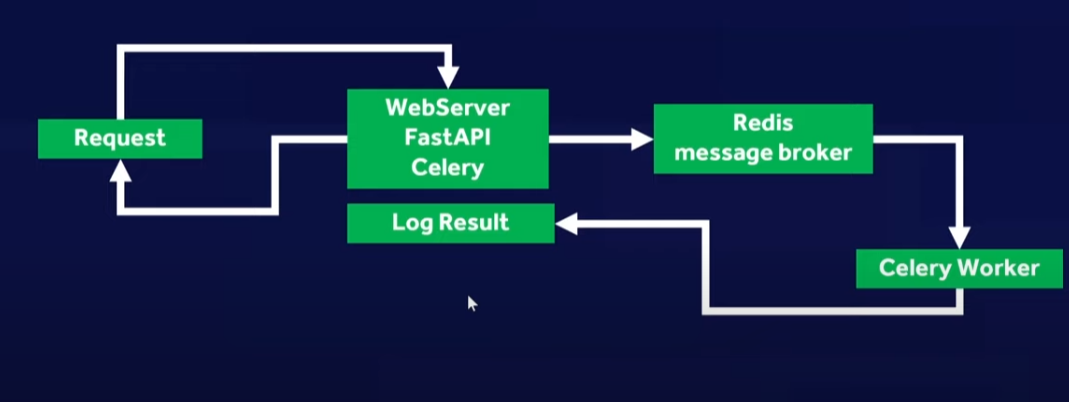
Finally, the **celery.conf.update(app.config)** line updates the Celery configuration with the Flask application's configuration. This ensures that any changes to the Flask application's configuration will also be reflected in the Celery configuration.

Graphical user interface, application

Description automatically generated

Graphical user interface, text

Description automatically generated



Graphical user interface, text

Description automatically generated

Text

Description automatically generated

4/4/2023

Text

Description automatically generated

I build a mobile application to take an image then return distance of object in frame to the users. The Flask server will handle HTTP POST request from users and process image. With celery and redisthe server can serve multi-clients simutaneously right? and this distributed task queue is useful in case of long-running task like processing images. So should I try another technique for the server like WSGI, what is the difference, compare 2 ways

Text

Description automatically generated