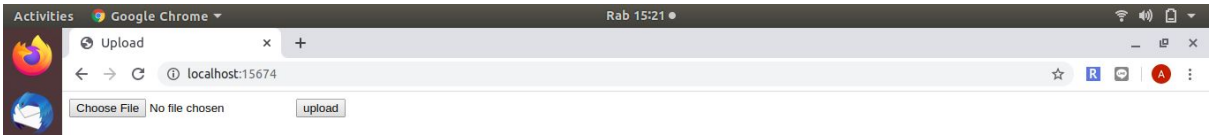
Alif Ahsanil Satria
1606882540

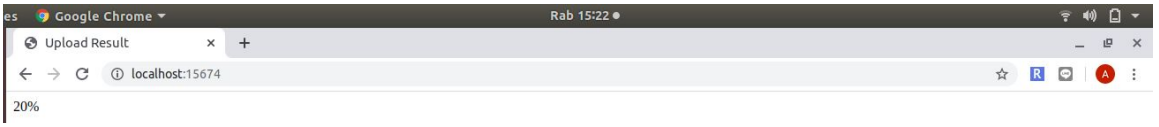
Service : Flask
Websocket : Flask-SocketIO (server) dan Socket.io.js (client)
Background Process : Celery

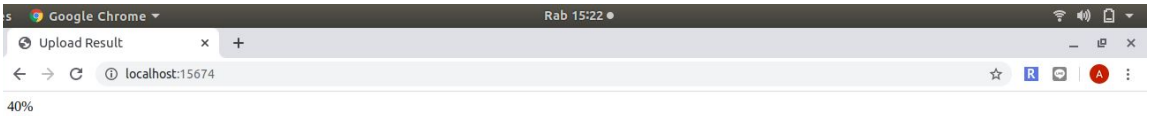
Berikut ini adalah skenario 1,2,3 dan code yang bersesuaian :




```python
server1.py > upload
from flask import Flask, render_template, request, redirect, url_for
from flask_socketio import SocketIO, emit, send
from celery_example import make_celery
import requests, uuid, json, pika, time

server1_app = Flask(__name__)
server1_app.config['SECRET_KEY'] = 'secret server 1'
socketio = SocketIO(server1_app)

celery_flask_app = make_celery(server1_app)

@server1_app.route('/', methods=['GET','POST'])
def upload():
    if request.method == "POST":
        print("masuk server 1")

        file = request.files['file']
        unique_id = str(uuid.uuid4())

        consume.delay(unique_id)
        # do something
        r = requests.post('http://127.0.0.1:5000/',
                          files={ 'file' : file },
                          headers = {"X-ROUTING-KEY": unique_id})

        result = json.loads(r.text)

        return render_template('upload_result.html')
    else:
        return render_template('upload.html')
```

Skenario 6 dan C dan code bersesuaian :





server1.py > upload

```python
from flask import Flask, render_template, request, redirect, url_for
from flask_socketio import SocketIO, emit, send
from celery_example import make_celery
import requests, uuid, json, pika, time

server1_app = Flask(__name__)
server1_app.config['SECRET_KEY'] = 'secret server 1'
socketio = SocketIO(server1_app)

celery_flask_app = make_celery(server1_app)

@server1_app.route('/', methods=['GET','POST'])
def upload():
    if request.method == "POST":
        print("masuk server 1")

        file = request.files['file']
        unique_id = str(uuid.uuid4())

        consume.delay(unique_id)
        # do something
        r = requests.post('http://127.0.0.1:5000/',
                          files={ 'file' : file },
                          headers = {"X-ROUTING-KEY": unique_id})

        result = json.loads(r.text)

        return render_template('upload_result.html')
    else:
        return render_template('upload.html')
```

upload_result.html yang menampilkan halaman HTML + JavaScript (lengkap dengan koneksi WebSocket & StompJS ke RabbitMQ). Dalam hal ini, saya menggunakan socket.io.js yang secara fungsionalitas sama dengan StompJS :

```
<title>Upload Result</title>
<script src="//code.jquery.com/jquery-1.12.4.min.js" integrity="sha256-ZosEbRLbNQzLpnKIkEdrPv7lOy9C27hHQ+Xp8a4MxAQ=" crossorigin="
<script src="//cdnjs.cloudflare.com/ajax/libs/socket.io/2.2.0/socket.io.js" integrity="sha256-yr4fRk/GU1ehYJPAs8P4JlTgu0Hdsp4ZKrx8
<script type="text/javascript" charset="utf-8">
    $(document).ready(function() {
        // Use a "/test" namespace.
        // An application can open a connection on multiple namespaces, and
        // Socket.IO will multiplex all those connections on a single
        // physical channel. If you don't care about multiple channels, you
        // can set the namespace to an empty string.
        // namespace = '/upload';
        namespace = '/'

        // Connect to the Socket.IO server.
        // The connection URL has the following format, relative to the current page:
        //     http[s]://<domain>:<port>[/<namespace>]
        var socket = io(namespace);

         // Event handler for server sent data.
        // The callback function is invoked whenever the server emits data
        // to the client. The data is then displayed in the "Received"
        // section of the page.
        socket.on('display_progress', function(msg, cb) {
            if (msg <= 100) {
                $("#progress").html(msg + "%");
                socket.emit('upgrade_progress', msg+10);
            }
        });

        socket.on('connect', function() {
            socket.emit('display_progress', {data: 'Start to display progress'});
        });
    });
</script>
```

compress file menggunakan background process pada file **server2.py** :

```
@celery_flask_app.task(name='server2.compress_file')
def compress_file(file_path, x_routing_key):
    connection, channel = produce()

    file = open(file_path, 'r')

    stat_info = os.stat(file_path)
    file_size = stat_info.st_size

    progress = 0

    file_compress = gzip.open(os.getcwd() + '/file_compress/' + os.path.basename(file.name) + '.gz', 'wb')
    while progress < 1:
        content = file.read(int(0.1*file_size))
        file_compress.write(str.encode(content))
        progress += 0.1

        channel.basic_publish(exchange='1606882540',
                              routing_key=x_routing_key,
                              body=str(progress*100) + '%')
        time.sleep(1)
```

server2.py sebagai producer yang akan mengirimkan persentase ke rabbitmq secara asyncrhonous :

```
def produce():
    credentials = pika.PlainCredentials('0806444524', '0806444524')
    connection = pika.BlockingConnection(
        pika.ConnectionParameters(host='152.118.148.95',
                                  port='5672',
                                  virtual_host='/0806444524',
                                  credentials=credentials
                                  ))
    channel = connection.channel()
    channel.exchange_declare(exchange='1606882540', exchange_type='direct')

    return connection, channel
```

server1.py sebagai consumer yang akan menerima data persentase dari rabbitmq secara asynchronous :

```
@celery_flask_app.task(name='server1.consume')
def consume(routing_key):
    credentials = pika.PlainCredentials('0806444524', '0806444524')
    connection = pika.BlockingConnection(
                    pika.ConnectionParameters(host='152.118.148.95',
                                              port='5672',
                                              virtual_host='/0806444524',
                                              credentials=credentials
                                              ))
    channel = connection.channel()

    channel.queue_declare(queue=routing_key, exclusive=True)
    channel.queue_bind(exchange='1606882540',
                       queue=routing_key,
                       routing_key=routing_key)

    channel.basic_consume(queue=routing_key, on_message_callback=receive_progress, auto_ack=True)
    print(' [*] Waiting for messages')

    channel.start_consuming()

def receive_progress(ch, method, properties, body):
    print("masuk receive progress")

    print(" [x] %r" % body)
```