Technical Design Doc for *Multiple Email Providers Service*

Tech Design Author: Junyong Suh

Last updated: September 1st, 2014

Feature Summary

The Multiple Email Providers Service is to send out emails with fail-overs. The service will try to send a mail with the highest priority email service provider set in configuration and if it fails due to any reasons, it will try to send it again with next priority email service provider. This service will provide fall-back mechanisms to serve mail services without downtime during any email service provider outages.

Figure 1 - Multiple Email Providers Service Architectural Diagram

<To be added>

Components View

1. Multiple Email Providers Service

Any HTTP POST requests coming to '/emails/' endpoint will be processed.

2. 3rd party email providers

3rd party email providers (ex. Sendgrid, Mailgun, ManDrill, …) will receive requests on their endpoints and response back to Multiple Email Providers Service

Process View

1. HTTP POST Request coming in with payload from client
2. Payload validation
3. Compose HTTP Post Request for email providers
4. Request to email providers
5. Get response from email providers
6. Response to the client

Use Cases

1. Client send requests with payload
2. Client gets response from the server

3. If succeed, the mail should be sent via one of email providers

Data Model View

No database or storage needed. May change log system from dumping to a file to insert to databases. Log files are in JSON format to ease the ETL process that may need in the future.

Sample log: {“timestamp”: “[TIME_STAMP]”, “user_ip”: “[USER_IP]”, “user-agent”: “[USER_AGENT]”, “type”: “[response|request]”, “status_code”: “[STATUS_CODE]”, “payload”: “[PAYLOAD_FROM_REQUEST]”, “response”: “[RESPONSE_BODY]”, “version”: “[VERSION_OF_THE_SYSTEM]”, “provider”: “[NAME_OF_THE_PROVIDER]”}

Sample Log Table Schema

Parameter Name	Description
timestamp	timestamp of the request / response
user_ip	IP address of the request (client) / response (server)
user_agent	Browser and version (of the request)
type	request or response
status_code	status code (of the response)
payload	JSON format request payload
response	body of response
provider	name of the provider (response)
version	version of the system

API View

1. /

We don’t do anything for requests coming for the root

Response:

400 Bad Request with JSON format response

{“message”: “Please check API documentation”}

2. /email/

Accepts POST request only. Validate the payload and try email providers once each until succeed and return the result.

Request:

```
curl -X POST http://127.0.0.1:5000/email/ -H "Content-Type:application/json" -d
@./test/sample.json -v
```

./test/sample.json

```
{ "to": "junyongsuh@gmail.com", "to_name": "Junyong Suh", "from": "no-reply@uber.com",
"from_name": "Uber", "subject": "Your Monday evening trip with Uber", "body": "<h1> Hello </h1>
<p> Sample </p>" }
```

Responses:

on success - 200 OK with JSON format response

{“message”: “Mail has sent successfully.”, “response”: “[RESPONSE FROM EMAIL PROVIDER]”}

on failure:

if payload validation failed,

400 Bad Request with JSON format response

{“message”: “Error while processing the request”, “response”: “payload invalid”}

other failures,

if all providers response timeouted,

408 Request Timeout with JSON format response

{“message”: “Error while processing the request”, “response”: “[PROPER ERROR MESSAGES]”}
500 Internal Server Error with JSON format response
{“message”: “Error while processing the request”, “response”: “[PROPER ERROR MESSAGES]”}

<To be implemented>
3. /email/open/<email provider name>/
Email open webhooks for Mailgun and Mandrill
Request:
Response:

<To be implemented>
4. /email/click/<email provider name>
Email click webhooks for Mailgun and Mandrill
Request:
Response:

<To be implemented>
Test Plan

1. Requirements test
    a. Python v2.7.x
    b. Flask v0.10.0 or higher
    c. requests v2.3.0 or higher
1. Configuration validity test
    a. JSON Malformed
    b. JSON missing each key
    c. JSON missing each value
    d. JSON wrong key
    e. JSON wrong value
2. API calls test (mock responses from servers)
    a. time out test
    b. service unavailable test

c. sample positive API call to success

<u>Deployment Plan</u>

Manual code updates upon all test are completed.

<u>High Level Schedule</u>

Task	**Time**
Requirements and email providers API documentation checking	4h
Get working curl calls to email providers	4h
Initial documentation (README / Tech Design Doc)	4h
Setup Python, Flask and requests and be familiar with them	8h
Input payload validation	4h
Integrate Sendgrid	8h
Integrate Mailgun	4h
Integrate Mandrill	4h
Testing	8h
Further implementations (if time allows) - Fallback - Keeping records - Webhooks for opens and clicks - Delayed delivery	N/A

Buffer time - finalize the code and documentations	8 h
TOTAL:	**7 days (56 hours)**