

---

Go School(Ngee Ann Polytechnic)

# **Project**

## **MPASS-M**

(Messaging-Platform-As-A-Service)

By: Chan Jiunn Hwa

**11<sup>th</sup> September 2021**

Version 0.999

---

## Credits:

Thanks to the lecturers and mentor for their sincere effort and guidance.

Thanks to GoSchool and Chloe.

Thanks to SGUP/Skills Future Singapore.

Mentor: Owen Yuwono

Lecturers: Lee Ching Yun, Low Kheng Hian

*Learning Golang*

*has enabled me to paint a new landscape.*

Jiunn@2021.

**Any omissions or errors are solely mine.**

## Table of Contents

<b>TABLE OF CONTENTS.....</b>	<b>1</b>
<b>1 EXECUTIVE SUMMARY.....</b>	<b>4</b>
<b>2 OVERVIEW.....</b>	<b>5</b>
2.1 Objective.....	7
2.2 Scope.....	7
<b>3 PROJECT PLAN.....</b>	<b>8</b>
3.1 Schedule.....	9
3.2 Table of Activities.....	10
3.3 Deliverables.....	13
<b>4 SYSTEM ANALYSIS.....</b>	<b>14</b>
<b>4.1 Requirements Analysis.....</b>	<b>15</b>
4.1.1 Functional and Quality Requirements Matrix™.....	16
4.1.2 Message Type Analysis.....	18
4.1.3 Process Flow Analysis.....	19
4.1.4 Service Pipelining.....	20
<b>4.2 Functional Features.....</b>	<b>21</b>
4.2.1 Create A Job.....	21
4.2.2 Check A Job.....	21
4.2.3 Job Assignment.....	21
4.2.4 Send A Job.....	21
4.2.5 Job Logging.....	21
4.2.6 Job Report.....	21
<b>4.3 System Architecture Design.....</b>	<b>22</b>
4.3.1 Database Schema.....	22
<b>5 SYSTEM DESIGN &amp; DEVELOPMENT.....</b>	<b>23</b>
<b>5.1 Design Philosophy.....</b>	<b>24</b>
5.1.1 Distributed Actor Model.....	26
5.1.2 OOP.....	26
5.1.3 Code Intent - Fluent Syntax.....	26
5.1.4 Parallelism vs Concurrency.....	28
5.1.5 Resiliency.....	28

<b>5.2 Schema.....</b>	<b>29</b>
5.2.1 Application Logical Schema.....	30
5.2.2 Project Layout.....	31
5.2.3 Code Layering Convention.....	32
<b>5.3 Use of Templates.....</b>	<b>33</b>
<b>5.4 Functions Implementation.....</b>	<b>34</b>
5.4.1 Create A Job.....	35
5.4.2 Check A Job.....	36
5.4.3 Job Assignment.....	37
5.4.4 Send A Job.....	38
5.4.5 Job Logging.....	40
5.4.6 Job Report.....	41
<b>6 SYSTEM TESTING.....</b>	<b>42</b>
<b>6.1 Test Strategy.....</b>	<b>42</b>
<b>6.2 Test Constraints.....</b>	<b>42</b>
<b>6.3 Unit Test.....</b>	<b>42</b>
<b>6.4 Integrated Test Cases.....</b>	<b>43</b>
6.4.1 T001A - Verify Create Message - OTP.....	44
6.4.2 T001B - Verify Send Message - OTP.....	46
6.4.3 T001C - Verify Send Message - OTP Not Blocked By Active DNC.....	47
6.4.4 T002A - Verify Create Message - OUN.....	48
6.4.5 T002B - Verify Send Message - OUN.....	50
6.4.6 T003A - Verify Create Message - MM.....	51
6.4.7 T003B - Verify Send SMS Message - MM.....	53
6.4.8 T003C - Verify Send WhatsApp Message - MM.....	54
6.4.9 T004 - Verify Gateway Routing.....	55
6.4.10 T005 - Verify Authorized Use.....	56
6.4.11 T006 - Verify Legitimate Recipients.....	57
6.4.12 T007 - Verify Bad Content Message Cannot Be Created.....	58
6.4.13 T008A - Verify DNC Auto Expiration Delete.....	60
6.4.14 T008B - Verify DNC Sending - MM Message Timeframe Throttle.....	61
6.4.15 T009 - Verify Daily Report.....	62
<b>7 PRODUCT DEPLOYMENT STRATEGY.....</b>	<b>63</b>
<b>7.1 PDCA Kanban Cycle - Continuous Improvement and Deployment.....</b>	<b>64</b>

<b>8 PRODUCT ROADMAP.....</b>	<b>65</b>
8.1 Worker Model.....	66
8.2 Message Model.....	66
8.3 Component Upgrade Model.....	66
<b>9 DATA TEMPLATES - MESSAGES.....</b>	<b>67</b>
9.1 OTP.....	67
9.2 OUN.....	67
9.3 MM.....	67
<b>10 DATA DICTIONARY.....</b>	<b>68</b>
10.1 Message.....	68
10.2 DNC.....	69
10.3 Report.....	70
<b>11 REFERENCES.....</b>	<b>71</b>

# 1 Executive Summary

This project MPASS-M(hereafter MPASS) is a http-based API microservice providing messaging services.

MPASS's mission is to provide a centralised messaging service. It's charter is to provide a new communications platform to centralise all outbound communications to customers. The goal is to have a single platform that the Client's developers can use to programmatically send messages from their service to customers.

This report covers the technical aspect of the SDLC, as well as some subjective observations and opinions for learning purpose.

## 2 Overview

This documents provides a record of the various phases and activities - a closure of the 'current state', while it is still live.

MPASS uses a Planned Approach to build out the Product. So far, this disciplined 'overall waterfall' type of process has yielded consistent, on time, and above minimum-stated-requirements deliverable.

Brief states: "The client has requested your team to design and build a working prototype of this system within 4 weeks." However, this 'Han-Solo' team was able to build a genesis proof-of-concept(POC) in the 1<sup>st</sup> week(3 days thinking, 3 days coding), which was reviewed to have met baseline requirements.

Then, a version was branched to further deepen some ideas once POC has passed the initial review. Therefore, the -M modified version is meant for further prototyping, whereas the genesis version is kept as a worse-case rollback option.



This document covers MPASS, whereas the decoupled logging MyLog-M(logging service) is covered in its own documentation. While the first-cut code was done in 1 week, this documentation is taking more than a week of iterations.

As both are the children of the parent project. So MPASS(main) and MyLog(subsidiary) have shared parts of documentation that are common or reused, but each is also documented in a self-contained manner - that is, each is sufficient unto its own, there is no need to jump to the other document.

MPASS's given charter is to build a Messaging System, hence the focus is to build a good Messaging platform, and will not be approached as a Messaging+Analytics+Whatever system. This direction will scope some of the design decisions.

As mentioned, the improvement sweep identified logging as a potential area for exploratory enhancement, given that grepping logs amongst lines interleaved within thousands of concurrent requests can be painful.

*Project documentation turns out to be longer in effort and duration(feels like forever, and even more onerous) than code writing - it just grew in length and depth even as I detailed it in bulleted forms for many parts.*

*This is perhaps a reflection of the fact that this project  
has rich dimensions to innovation,  
enough scope to become a single project, and  
perhaps also reflecting that Documentation is as important as Code.*

***In some ways, this report is more important than the code.***

*Each iterative proof-read results in novel points, or refined distillation of points, tending towards completeness.*

*External content was used in this document, they are linked in Reference Section.*

*Lastly, though this is the capstone project, but as it is post-curricula, it requires ginormous self-motivation and effort to stay focused, primarily due to a) Attention Drift, b) remains pretty much a self-conjectured project without interaction with real users, c) as well as the need to run another project concurrently - Project Job Hunt ;) - thus handicapping somewhat the product output.*



## 2.1 Objective

The objective of MPASS is to prototype a performant messaging service that can scale.

## 2.2 Scope

The scope is to develop a usable application focused on messaging.

### 3 Project Plan

A Project Plan helps ensure that work is expended with a disciplined approach and effort is planned and budgeted for each lifecycle activity.

### 3.1 Schedule

The sheet below shows the proposed schedule.

TimeLine 09 AUG - 30 SEP 2021									
Project MPASS-M	WORK PLAN								
MONTH	AUG					SEP			
WEEK STARTING	9	16	23	30	13	6	13	20	27
WEEK INDEX	0	1	2	3	4	5	6	7	8
<b>CONCEPTUALISATION/RESEARCH/SELECTION(C)</b>									
Project Planning									
Literature Review & Research									
<b>ANALYSIS &amp; DESIGN(AD)</b>									
System Analysis & Design									
<b>CODING &amp; DEVELOPMENT</b>									
A. Message Delivery									
B. Concurrent Requests									
C. Appropriate Delivery									
D. Graceful Failures									
E. Authorised Use									
F. Legitimate Recipients									
G. Content Abuse									
H. Logging									
I. Accounting									
<b>TESTING</b>									
Integration Test									
<b>REVIEW</b>									
Final changes/bug fix									
<b>DOCUMENTATION</b>									
Documentation/Demo Preparation									
<b>SUBMISSION</b>									

### 3.2 Table of Activities

CONCEPTUALISATION/RESEARCH/SELECTION(C)	
Project Planning	<p><b>A. <u>Creation of Project Plan</u></b></p> <p>Mapping and time-boxing the activities, as well as listing out the desired features are the first activities. The intention is to ensure proportionate and balanced effort.</p> <p>From experience, consistent output comes from proportionate budgeting, and coding usually does not occupy more than 50% of the lifecycle for a balanced plan.</p>
Literature Review & Research	<p><b>B. <u>Background Research</u></b></p> <p>Reading and research on the specific technologies to be used, and whether it is feasible.</p>
ANALYSIS & DESIGN(AD)	
Functions	<p><b>C. <u>Functional Analysis</u></b></p> <p>Based on the background research, features are listed, though it is estimated not all can be implemented. The outputs are database design, requirements matrix.</p> <p><b>D. <u>Feasibility Analysis</u></b></p> <p>Scoping and feasibility has to be estimated.</p>
CODING	
Sprint	<p><b>E. <u>Code Methodology</u></b></p> <p>The project uses a 'sprint' type of methodology for the coding sub-phase, but overall planning is still a 'waterfall' design. Each major feature will have a budgeted sprint cycle of up to 1 manday, while minor features will take 0.25 manday.</p>

### **F. Risk Management**

To manage timeline risk, each feature( $f_1, \dots$ ) is allowed a sprint cycle of about 1 day. Any deepening of features can be revisited in later sweeps, or put up as future features. Bottlenecks are required to be resolved within the day, as buffered up issues takes an exponential risk to panicking.

<b>TESTING</b>	
Planning & Design	<b><u>G. Test Case Design and Execution</u></b> Unit, Regression and Integration are included.
Integration Test	
<b>REVIEW</b>	
Final review	<b><u>H. Review &amp; Fix</u></b> Code freeze, and bug fixes only.
Final changes and fix	
<b>DOCUMENTATION</b>	
Planning	<b><u>I. Documentation</u></b> Documentation is continuous, but the main cycle starts some time after coding, after coding has peaked.  Final rounds will be done alongside the Review and Fix cycle, collated before submission.
Document Structure Setup	
Record	
Final Review	

**SUBMISSION****J. Zip and Upload**

By 2359 30<sup>th</sup> Sep 2021(THU), consolidate and put into submission folder:

- Writeup
  - Add this document in doc format.
  - Add pdf version as backup, in case the word doc format runs.
- Presentation slides
- Video of demo
  - test screen shots
  - videos
- Source code and all necessary files to setup and run the prototype
  - source files, executable, test data, scripts and any other artifacts.
- Zipped and upload<YourName\_ProjectName.zip>

### 3.3 Deliverables

The following are the deliverables:

- Application - a messaging service prototype.
- Documentation - this report.

## 4 System Analysis

Functional analysis is to box in an overall coherent release. Planning also allows systematic roadmapping of future features.

The following sections captures only briefly the main design factors as scope, time, and space does not allow a full elucidation.



## 4.1 Requirements Analysis

Functional features can be thought of as explicit, contractual requirements, whereas quality requirements can be considered as implicit requirements, typically performance related that is better than contractual acceptance; or improved features in terms of functionalities or usabilities that delights customer.

The following describes analysis based on the client's brief.

#### 4.1.1 Functional and Quality Requirements Matrix™

Understanding and Decomposing into the following Matrix allows the mapping of fuzzy requirements into well defined specifications,

with clear tabulated pathways for Additions and Enhancements.

The importance cannot be under-estimated because without the Matrix,

the Future will be a 'Tangled Mess'

as features can Overlap, Over-Arch, Under-Reach

in time if the Roots are not well articulated.

The Matrix is trademarked to mean:

- Rows as functional requirements
- Enhancement Column are quality requirements

This matrix™ represents very clearly on how Features can be added, and Enhancements injected, in a managed way.

#	Item	Current Method	Enhancement Idea
A	The system should deliver messages as soon as possible, but not earlier or later than instructed.	✓-Scheduler	
B	The system should be able to handle many concurrent requests.	◆-Sharding,	
C	The system should use the appropriate delivery method and gateway based on business rules.	✓-SmartRouting	
D	The system should be built to handle failures gracefully.	◆-Sharding, X Retry-Fallback	custom http client with retries, etc
E	The system should only be used by authorized source systems, and only for permitted use cases.	✓-RulesValidation	
F	The system should only send messages to legitimate recipients.	✓-RulesValidation ◆-DNC	
G	The system should have mechanisms to prevent abuse/inappropriate content.	✓-Templates ✓-StopWords ✓-Audit Trail	to use bloom filters
H	The system should log and track every message that has been sent for audit and compliance purposes.	✓-Logging	custom logging mechanism
I	The system should account for every message sent and provide accurate reports.	✓-Reporting	

**4.1.1.1 #A - Timely Message Delivery**

The system should deliver messages as soon as possible, but not earlier or later than instructed.

**4.1.1.2 #B - Provide Concurrent Requests**

The system should be able to handle many concurrent requests.

**4.1.1.3 #C - Use Appropriate Delivery Method**

The system should use the appropriate delivery method and gateway based on business rules.

**4.1.1.4 #D - Handle Failures**

The system should be built to handle failures gracefully.

**4.1.1.5 #E - Authorized Use**

The system should only be used by authorized source systems, and only for permitted use cases.

**4.1.1.6 #F - Legitimate Recipients**

The system should only send messages to legitimate recipients.

**4.1.1.7 #G - Prevent Content Abuse**

The system should have mechanisms to prevent abuse/inappropriate content.

**4.1.1.8 #H - Message Log**

The system should log and track every message that has been sent for audit and compliance purposes.

**4.1.1.9 #I - Report**

The system should account for every message sent and provide accurate reports.

#### 4.1.2 Message Type Analysis

As per the description, we can classify the messages into the following types below.

They are differentiated by these properties of recipient, time, and DNC(do not call) rule.

In future, further types can be added.

	Short Message	Notification	Long Message
Recipient	Single	Single	Multi
Time	Instant	Deferred	Scheduled
DNC	No	No	Yes
Examples	OTP	Order Update Notification Payment Notification Order Delivery Alert	Marketing Messages

### 4.1.3 Process Flow Analysis

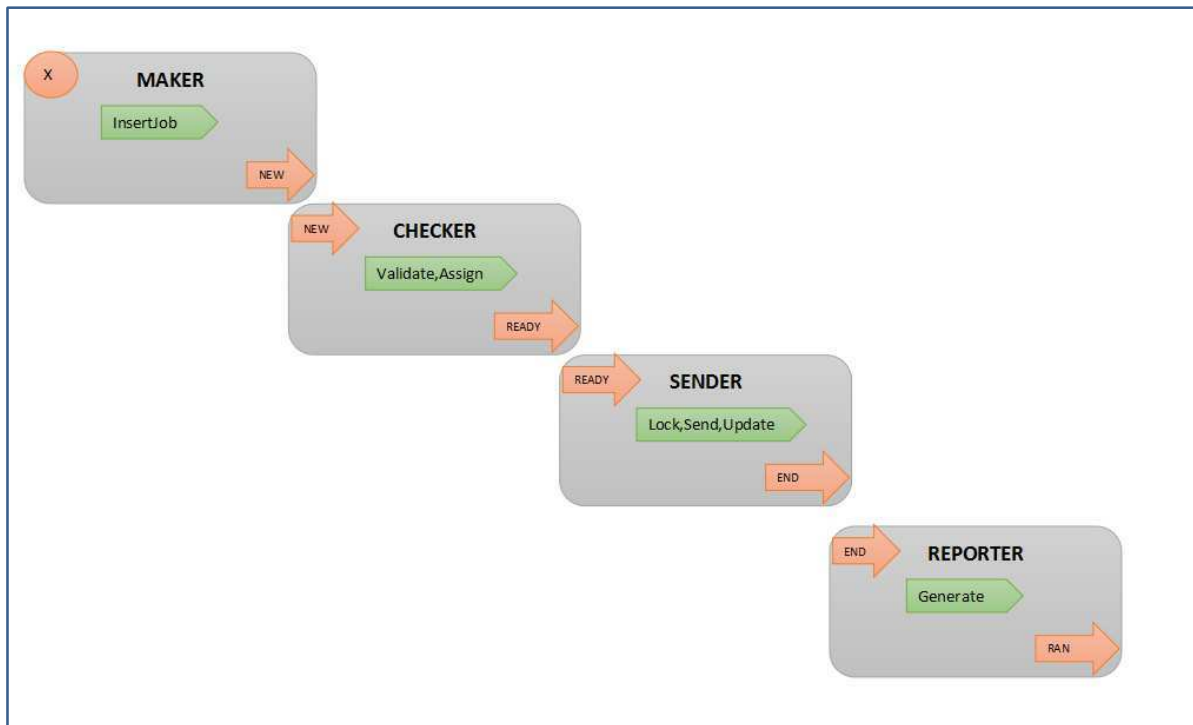
Message sending at the process level is a sequence of actions from different actors.

We can describe the process in the following 4 Steps, namely:

- Maker - where an User makes a job.
- Checker - an intermediate quality check.
- Sender - where a worker process pulls and sends the job.
- Report - capture of state and metrics for closure and downstream purpose.

For our purpose, a Step has a Start & End State, and is triggered by an User(Actor) or a Timer(Event).

The steps are 'sequence-path-dependent'.



#### 4.1.4 Service Pipelining

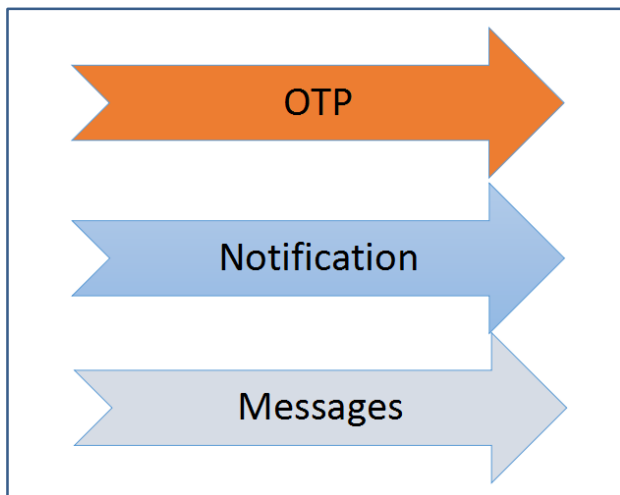
One benefit of sub-classing messages into types is that besides identifying different properties, we can also pipeline or slice the process into sub-requirements in terms of demands, cost and performance.

At the start, when the request volume is low, a single server servicing all the messages types can be sufficient. However, when volume rises significantly, and also when the business footprint expands into more geographic locations, this service can be multi-deployed or sliced according to load requirements, or location requirements, etc.

In other words, dedicated pipes or service pipelines can be setup to serve the different mix of message types under different sub-requirements.

We see that the different message types have different demands:

- OTP: single recipient, time-sensitive, low volume.
- Notification: single recipient, non-immediate, moderate load.
- Messages: one or many recipients, non-immediate, high volume/load. May be cost-sensitive, and may be subjected country's data protection laws.



This approach enables us to naturally have the option to either setup a shared common server to service all 3 types; or to set up 3 dedicated servers to serve each type; or an in between mix. By server, we can also mean a cluster.

Additionally, though not explicitly defined as a requirement, this allows the service to be split into different geo-locations, if required. For instance, it may be cheaper to send bulk messages to Indonesian customers from a server in Indonesia, whereas all other customers are served from a central service in HQ location; and so on.

## 4.2 Functional Features

This version will provide the following primary features in accordance to the requirements matrix.

### 4.2.1 Create A Job

The system shall provide a function for User to create a job of different message types(OTP, OUN, MM).

### 4.2.2 Check A Job

The system shall provide a function to check and validate for:

- #E - Authorized Use
- #F - Legitimate Recipients
- #G - Prevent Content Abuse

### 4.2.3 Job Assignment

The system shall provide a function to assign an appropriate provider:

- #C - Use Appropriate Delivery Method

### 4.2.4 Send A Job

The system shall provide a function to send a job in a timely and graceful manner:

- #A - Timely Message Delivery
- #B - Provide Concurrent Requests
- #D - Handle Failures

### 4.2.5 Job Logging

The system shall provide adequate logging:

- #H - Message Log

### 4.2.6 Job Report

The system shall provide transactional reporting:

- #I - Report

## 4.3 System Architecture Design

### 4.3.1 Database Schema

Description of the fields below is in the Data Dictionary.

Message	
ID	varchar
Name	varchar
MsgType	varchar
Providers	varchar
From	varchar
To	varchar
Body	varchar
CreateTime	timestamp
CreateBy	varchar
OriginSystem	varchar
SendTime	timestamp
WorkerName	varchar
StartTime	timestamp
EndTime	timestamp
Status	varchar
Result	varchar

DNC	
ID	varchar
ReasonCode	varchar
ExpireTime	timestamp

Report	
ID	varchar
EndTime	timestamp
Details	json



## 5 System Design & Development

This section describes the actual development of the features, walking through the design and the key code sections.

## 5.1 Design Philosophy

This section was added subsequently to blot out and give shape to some of the evolving and under-current inner thinkings whilst doing this project. This is not a comparison of languages and models, but more for self-learning by re-hashing through the design process and letting it derive its logical conclusion.

One can begin by asking, as is also embedded in the client's brief; 'What makes a good messaging platform'? Thence, as we sojourn through, we shall see - 'The Start determines The Ending' ...

If we define the problem space as of the 'messaging world', then a background and history trace would lead us to seeing 'tele-communications' as the ancestor, as it was the early precursor of modern messaging, and driving further, inevitably we would run into the often benchmarked Erlang/OTP platform.

Erlang was designed for creating applications (such as telephone switches) that must run nonstop. It is a functional, dynamically typed language invented in 1986 at Ericsson. It is used for code that has to be hot-swappable, distributed, and robust in the face of errors. Erlang was open-sourced in 1998 and has gained prominence as concurrency has become a modern computing concern.

At Ericsson, the AXD 301 switch is the flagship example of Erlang's scalability and reliability. This switch handles millions of calls per week and has a reported 99.9999999 percent uptime, which means the system only has less than one second of downtime during the period of 20 years - this is impossibly high even for many today's systems. Though there has been some contention over the nine 9's figure, but generally Erlang's model is the benchmark model for claimants of high-performance, highly-reliability system.

Solutions that are world scale and are using Erlang includes WeChat, WhatsApp, RabbitMQ, etc. RabbitMQ is a high-performance AMQP server written in Erlang that can reportedly handle 400,000 messages per second.

Erlang boast a number of high-profile successes, and these users consistently tout Erlang's strengths for writing distributed, concurrent, fault-tolerant software. It's clear that the actor model is a valid alternative for addressing concurrency without shared state and locks.

Therefore Erlang's system model, is a proven, mature, concrete production definition of a high-performance, highly-reliability system. It rest on these 4 Pillars:

- Concurrency
- Fault-Tolerance
- Distributed Model
- Actor Model

At the start, I did not embark by 'borrowing' Erlang's model as a framework, but in the process of documenting this project, I realize of the convergence in terms of design thinking, and bringing it in has helped to elide this section with a firmer theoretical basis.

In addition, if the macro level rests upon the something similar to the Erlang's model; I would add at the micro level, highly performant system should also follow the Unix's philosophy that systems should be simple and single focused(single responsibility).

- Simplicity Model(do not chose a more layered architecture, if a simpler one exists)
- Single Focused Model(do one thing, well)

Having not began from Erlang(as this is a Golang's course project), we derive the 6 pillars as the design backbone.

On the other hand, Golang serves as a good language to implement the pillars in comparison to C, C++, C#, Java, Python, etc.

### 5.1.1 Distributed Actor Model

Hence, by beginning with the abstract question (yes, an occupational hazard as the author's original degree is B.A. Philosophy) of 'What is a good (messaging) system (in the modern context)', we have converged to a possible design path, and from the concrete success of Erlang, then it seems that the Actor Model is key.

Also, though this is not mentioned in the brief, as it is a high level business requirements document, it's obvious that a single node solution will be constrained in scaling to the eventual billions of requests - no matter how well you can write go-routines. This is because a single node solution can only scale vertically through hardware upgrades, and this way has a diminishing returns in terms of performance-cost, as well as a hard upper limit in which no increase in cost can return another iota of performance.

### 5.1.2 OOP

Next, taking it further, the idea of asynchronous concurrency and communicating through message passing allows high availability and reliability. And by message passing, we would implicitly mean that at a high level of abstraction, we are talking about the concept of having Objects, or Actors. Now, there is some debate on OOP by the community of Golang, but that is not my interest here.

End of day, from this thread of analysis, it seems clear that a Distributed Actor Model, with some lightweight OOP concepts(Object/Actors), could result in cleaner outcomes in terms of performance, design and maintenance.

### 5.1.3 Code Intent - Fluent Syntax

Next, it is also clear that the message sending process is a sequence of actions, and therefore if we want to map them one-to-one to the code, so as to achieve higher Intent Transparency, then it is better to use a Fluent syntax. The use of Fluent syntax easily allows the Design Intent to be clearly seen, and can lead to better Quality outcomes

Though not all Gophers embrace Fluent Syntax, but a non-fluent syntax can mean superfluous lengthy code. In all honesty, overly long codes cannot be cleaner and simpler than what can be 'ichimoku'(一目了然). Golang, in trying to rid of the ternary operator 'evil', it has not really made things simpler in all cases - instead, what it has done is that it *has shifted horizontal complexity(terseness) to vertical complexity(overly lengthy superfluous lines crowding out the main Line-Of-Interest or intention)*. It is un-deniable that length creates its own complexity, and worse, it also forces the coder to do 'line scan' instead of doing 'binary search' of interesting section of codes.

In closing, this long detailing is to walk through together with the audience the design intent; and that this project is experimenting and trying out the different approaches; as well as to avoid a mark down due to some 'hard-coded' convention.

Though this section has been iterated a few times, but it is nevertheless not final, and some further revisions could refine some of the learning points further.

Having said that, the MyLog sub-project practices The Clean Architecture(thanks to the guidance by mentor) as it is more Resume Augmenting due to Industry Fashion.

### 5.1.4 Parallelism vs Concurrency

One of Golang's key strength is to enable concurrency very easily through the usage of goroutines.

Because this solution allows multiple deploys, it is actually a Parallel solution as each compute or deploy instance can run a slice of jobs independently, whereas Concurrent solutions(if single core) are essentially using OS's time-slicing to multi-task the work.

Additionally, it is found that even at low volume of logging, concurrent logging with each request itself generating multiple log lines makes the logs even more inter-leaved, and harder to catch offending lines as they are dispersed out.

### 5.1.5 Resilency

The distributed actor model allows better resiliency as each compute node processes a slice of jobs, and even if a node were to crash, the overall uptime is guaranteed as we increase nodes.

....

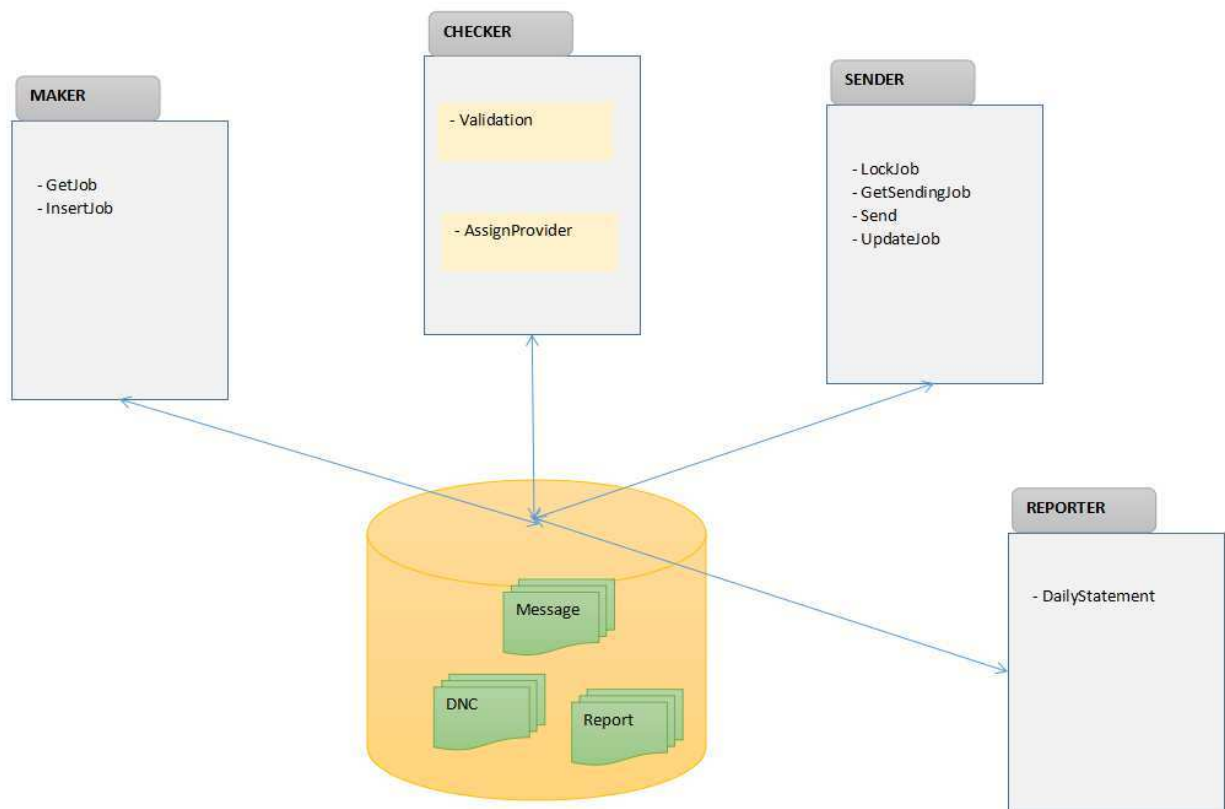
## 5.2 Schema

The following is to show the organization of the project.

### 5.2.1 Application Logical Schema

The system has 4 Parts:

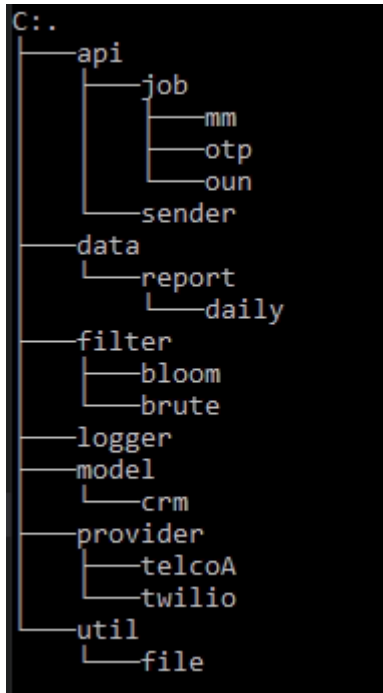
- Maker handles job creation(CRUD)
- Checker handles Validation and Assignment
- Sender handles the sending by Provider
- Reporter handles generating post transaction data for other applications.





## 5.2.2 Project Layout

The system files are organized accordingly:



### 5.2.3 Code Layering Convention

The following explains some of the ideas used for coding convention.

In this project, the coding structure is generally divided into 3 layers - top, supervisory, worker functions.

The 'division of labour' is as follows:

- Top. This is the 'boss' level that interfaces to external callers, and are contractually bounded to return an output when given an input, thus it should behave more like a Pure function. And at this level, the Fluent syntax seems more appropriate.
- Middle These are the supervisory functions that mediates between the Top and Lower levels. Generally, all errors should be resolved at this level, and not passed further up. Analogically it makes sense that if you pass Errors all the way to 'The Boss', it is Panic time.
- Worker. These are the low level functions that may interface more directly to the machine level, and thus adhere to the 'Return Value, Error' style.

Next, internal values are passed via pointers/structs, whereas message passing to external systems are via JSON.

Overall, the objective is to to allow 'ichimoku' of the Program/Section Intent - simply by visual inspection we can already tell whether it is aligned to the Matrix. Reducing cognitive load by making the intent intuitive allows for lower defects - a 'kanban' type of philosophy.

## 5.3 Use of Templates

The system will use Golang's templating system to implement message templates. The default templates for the different message types are listed in Section 9: 9Data Templates - Messages.

## 5.4 Functions Implementation

### 5.4.1 Create A Job

The function lets User to create a job of the different analyzed message types(OTP, OUN, MM) via JSON inputs, and respond back with JSON.

```
113 //myJob is the base implentation for Read and Insert a new job
114 func myJob(w http.ResponseWriter, r *http.Request) {
115     j := job.NewJob(DB)
116     if r.Method == http.MethodGet {
117         response.AsJSON(w, j.StatusCode, j.GetJobByID(r).Result)
118         return
119     }
120     if r.Method == http.MethodPost {
121         response.AsJSON(w, j.StatusCode, j.Insert(r.Body).Result)
122         return
123     }
124     response.AsJSONError(w, http.StatusBadRequest, "Invalid action")
125 }
```

server.go

### 5.4.2 Check A Job

The function check and validates for:

- #E - Authorized Use
- #F - Legitimate Recipients
- #G - Prevent Content Abuse
- Syntax, Validation errors

```
37 //Insert a new job
38 func (j *OTP) Insert(rdr io.Reader) *OTP {
39     if j.Parse(rdr).HasErr {
40         return j
41     }
42
43     //Set defaults,
44     now := time.Now().Local()
45     j.SetMsgDefaults("otp/sms", now, now, "READY")
46
47     //Check basic errors - empty fields, validity
48     if err := j.HasCommonErrorsV1(); err != nil {
49         j.SetResultErr(err.Error())
50         return j
51     }
52
53     //Check specific errors
54     if len(j.Recipients) != 1 {
55         j.SetResultErr("Recipient can only be one")
56         return j
57     }
58
59     //Check Rules
60     if strings.ToUpper(strings.TrimSpace(j.OriginSystem)) != "FINANCE" {
61         j.SetResultErr("Invalid OriginSystem")
62         return j
63     }
64
65     //Check Bad Content
66     if bloom.HasBadWord(j.Message) {
67         j.SetResultErr("Has bad content")
68         return j
69     }
70
71     //Assign Provider, default to twilio if no provider for OTP
72     provider.AssignProviderOTP(j.Job)
73
74     //Run
75     j.RunSQLInsert()
76     return j
77 }
```

otp.go

### 5.4.3 Job Assignment

The function assigns an appropriate provider:

- #C - Use Appropriate Delivery Method

```
14 //AssignProvider sets the gateway
15 //AssignProvider defaults to Twilio if no provider for sms,
16 //AssignProvider will also set the provider based on recipient number at this stage for single recipient, otherwise late bind u
17 func AssignProvider(j *job.Job) {
18     if strings.ToUpper(strings.TrimSpace(j.MessageType)) == "SMS" {
19         prov := strings.TrimSpace(j.Provider)
20         if len(prov) == 0 {
21             j.Provider = "twilio"
22             return
23         }
24
25         if len(j.Recipients) == 1 {
26             j.Provider = GetTelco(j.Provider, j.Recipients[0])
27         }
28     }
29 }
30 }
```

provider.go

#### 5.4.4 Send A Job

The function sends a job in a timely and graceful manner:

- #A - Timely Message Delivery

```
39 | //LockJobs locks the job based on Status and SendTime
40 | func (s *Sender) LockJobs(workerID string, limit int) *Sender {
41 |     newStatus, oldStatus := "LOCKED", "READY"
42 |     sql := "UPDATE pegasus.message "
43 |     sql += "SET WorkerName = '" + workerID + "', Status = '" + newStatus + "'" //LOCKED
44 |     sql += "WHERE Status = '" + oldStatus + "'" //READY
45 |     sql += "AND NOW() >= SendTime "
46 |     sql += "ORDER BY RID Desc "
47 |     if limit > 0 {
48 |         sql += fmt.Sprintf("LIMIT %d ", limit)
49 |     }
50 |     sql += ";"
51 |     s.ExecSQL(sql)
52 |     return s
53 | }
```

job.go

- #B - Provide Concurrent Requests

```
18 | //JobManager handles concurrent job sending
19 | func JobManager() {
20 |     s := sender.NewSender(DB)
21 |     jobs, _ := s.LockJobs(WorkerID, JobCount).GetSendingJob(WorkerID, "LOCKED", JobCount)
22 |     for _, v := range *jobs {
23 |         go worker(&v, s) //async send
24 |     }
25 | }
```

job.go



- #D - Handle Failures

```
118 //TrySendSMS post the sms to provider
119 //with Retries, client timeout, exponential backoff
120 func (tw *Twilio) TrySendSMS(from, to, body string, MaxTries int) *Twilio {
121     timeout := time.Duration(10) * time.Second
122     backoff := GetNextNum(0)
123     for i := 0; i < MaxTries; i++ {
124         tw.Post(from, to, body, timeout)
125         if tw.StatusCode == 200 {
126             break
127         }
128         //handle generic non-retry error
129         if tw.StatusCode == 404 {
130             logger.Log("DEBUG", 0, fmt.Sprintf("Status: %s\tBody : %s\n", tw.Status, tw.Bytes))
131             return tw
132         }
133         if tw.ErrCode > 0 {
134             logger.Log("DEBUG", 0, fmt.Sprintf("Status: %s\tBody : %s\n", tw.Status, tw.Bytes))
135             //handle any specific non-retry error
136
137             //do retry
138             if tw.ErrCode == 1000 && strings.Contains(tw.Status, "Timeout") {
139                 logger.Log("DEBUG", 0, fmt.Sprintf("Status: %s\tBody : %s\n", tw.Status, tw.Bytes))
140                 timeout = timeout * 2
141                 time.Sleep(time.Duration(backoff()+RandMinMax(0, 3)) * time.Second)
142                 continue
143             }
144         }
145     }
146     logger.Log("DEBUG", 0, fmt.Sprintf("Status: %s\tBody : %s\n", tw.Status, tw.Bytes))
147     return tw
148 }
```

twilio.go

### 5.4.5 Job Logging

The function provides logging:

- #H - Message Log

```
25 //Log provides logging similar to a Syslog format
26 func Log(LogType string, LogSeverity int, LogText string) {
27     rec := &Record{LogText: LogText}
28     rec.LogType = LogType
29     rec.LogSeverity = LogSeverity
30     rec.LogText = LogText
31     bytes, err := json.Marshal(rec)
32     if err != nil {
33         //fallback logging options, default to filelog
34         log.Println(err)
35         go Post(err.Error())
36         return
37     }
38     //multi-write, can post to multiple destination if needed for redundancy
39     log.Printf("%#v %#v %#v\n", LogType, LogSeverity, LogText)
40     go Post(string(bytes))
41 }
```

logger.go

### 5.4.6 Job Report

The function generates transactional reporting:

- #l - Report

```
66 //myView provides a report view of the list of transactions by parameter date
67 func myDaily(w http.ResponseWriter, r *http.Request) {
68     tmpl := html.LoadTemplate(tplDir, "view.gohtml")
69     if r.Method == http.MethodGet {
70         recs := *Get(r)
71         viewData := &ViewData{PageTitle: "View - Daily Transactions", ReportDate: r.URL.Query().Get("date"),
72         tmpl.Execute(w, viewData)
73         return
74     }
75     response.AsJSONError(w, http.StatusMethodNotAllowed, "Invalid action")
76 }
77
```

server.go

## 6 SYSTEM TESTING

### 6.1 Test Strategy

The approach is to design test cases of scenarios that are representative of the requirements, but not exhaustively due to length and constraints.

Not all tests done are documented, only the major ones, otherwise this documentation would be at least 10 more pages. Also, some of the tests can be even more elaborate, but a simpler test that sufficiently covers the scenario was done.

Testing is manual and uses any REST Client(eg. PostMan/Curl).

### 6.2 Test Constraints

Even though the test cases are not entire, it still covers about 30+ individual sub-items.

Other limitations include:

- Gateway - All outbound communications default to Twilio, as there were no other real gateways provided in this period. Without access to a real gateway from a Provider, it is not wise to test upon 'simulated' gateways, as they could give a sense of false correctness. This is because each telco provider can have very specific or proprietary low level error mechanisms, as well as protocol. Though we are prototyping with http, in high volume scenarios, native GSM(sms) protocols coupled with leased lines may be the choice for performance. In other words, http over internet, even on business subscription is not QoS enabled - leaving your message traffic and other traffic commingled.
- Phone numbers - Without access to a list of test phone numbers, as well as Usage Credits, it is not possible to do massive, multi messaging at a real-world level. Hence, concurrency and load testing is difficult without budget.

### 6.3 Unit Test

Unit test resides together with the code, and are performed with the built-in test runner. Respective unit tests done can be re-run and the results outputted ad-hoc.

## 6.4 Integrated Test Cases

For this project, the testing strategy will be an 'end-to-end' test of the various API calls to verify results are as expected, so as to verify scenarios as per the Matrix. The test sets represent key tests to show a baseline, viable product.

Testing itself has scope to be a project by itself, and that will include other tests like Load Testing, Security Testing, Parallel/Concurrency testing for multi-node configuration, etc. In production, this will mean having dedicated QA team with automated tools; as well as having Testing Strategy defined in parallel during requirements gathering phase.

Hence, in a way, the tests here are more for documentary evidence, than for actual audit objectives - essentially demonstrating if not for anything, that this student have been working hard :). Nonetheless, the test constructed are good enough, even though I have no formal QA qualifications. In actual production test cases I would write, if time permits, the language used would be even more formal and tighter. Professional Test Sentencing has to be specific, and non-ambiguous. Having said that, vendors many times scope test more to indemnify liabilities than to actually assert true correctness.

### **Highlights:**

Most of other cases were quite run-of-the-mill, but the following 2 took a bit of 'head-scratching'

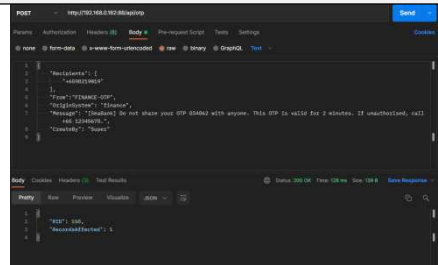
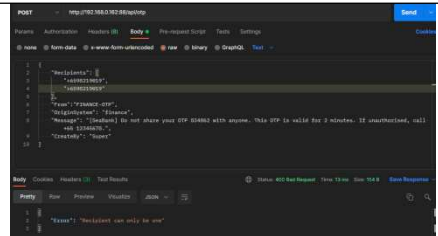
- Test Case T004 - Verify Gateway Routing.

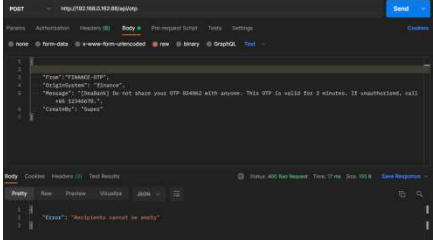
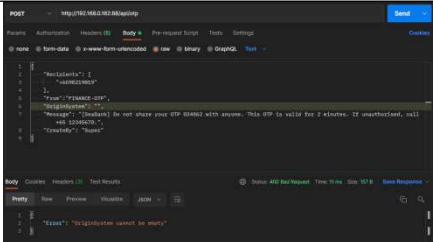
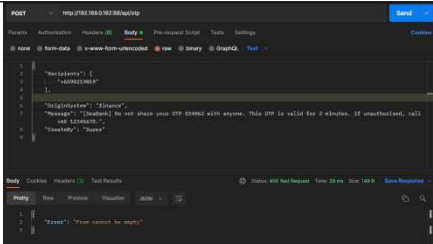
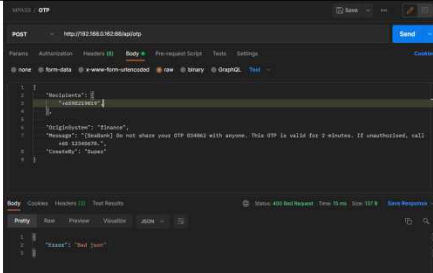
Finally, it was decided that this would be deferred as there is no concrete way to test until having access to real multiple gateways.

- Test Case T008B - Verify DNC Sending - Message Timeframe Throttle

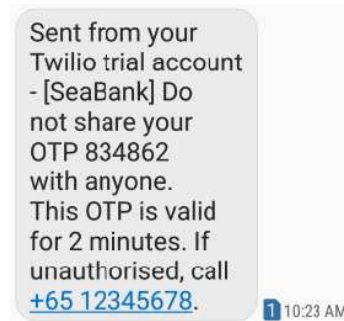
To test the correctness of this feature can require some creativity, but after some stumbling, it was decided that a simple and sufficient test at this stage can be done just by sending 3 messages, at T+0s, T+30s, T+61s. For more realness, other methods can be mixed in, and simultaneous sending from multiple users can be done, and results accounted for accordingly.

### 6.4.1 T001A - Verify Create Message - OTP

<b>Date</b>	2021-09-11 11:32:11 AM		
<b>Test Case ID</b>	T001A		
<b>Test Scenario</b>	Verify system is able to create the message type OTP.		
<b>Test Steps</b>	Using any REST Client, execute with the test data		
<b>Test Data</b>	JSON data as per screenshot.		
<b>Expected Results</b>	<b>Actual Results</b>	<b>Pass/Fail</b>	<b>Screen Log</b>
Record ID(RID) is seen.	As Expected.	Pass	
Multiple Recipients not allowed for OTP.	As Expected.	Pass	

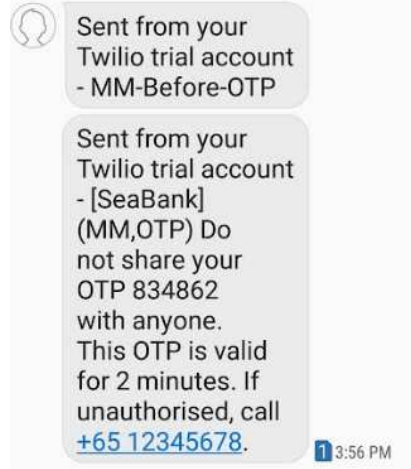
Recipient cannot be empty.	As Expected.	Pass	
Origin System cannot be empty.	As Expected.	Pass	
From cannot be empty.	As Expected.	Pass	
Bad json.	As Expected.	Pass	

**6.4.2 T001B - Verify Send Message - OTP**

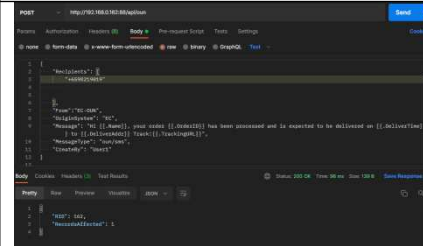
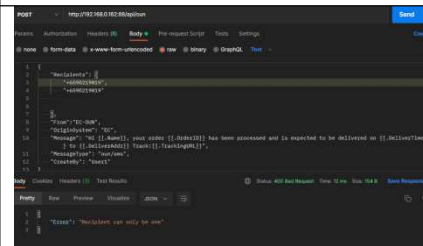
<b>Date</b>	2021-09-11 11:42:15 AM		
<b>Test Case ID</b>	T001B		
<b>Test Scenario</b>	Verify system is able to send the message type.		
<b>Test Steps</b>	Using any REST Client, execute with the test data.		
<b>Test Data</b>	JSON data as per screenshot.		
<b>Expected Results</b>	<b>Actual Results</b>	<b>Pass/Fail</b>	<b>Screen Log</b>
OTP message is received immediately	As Expected.	Pass	

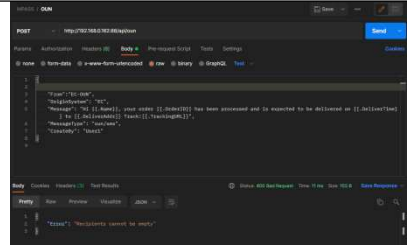
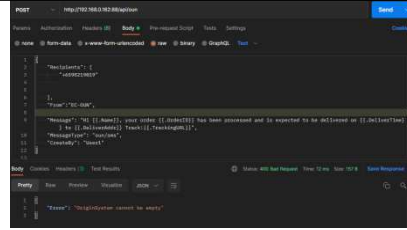
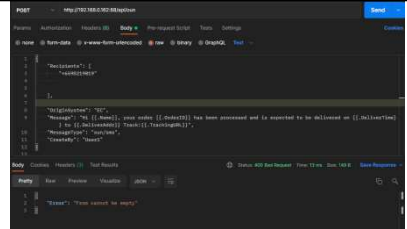
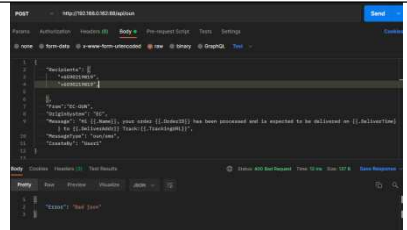


### 6.4.3 T001C - Verify Send Message - OTP Not Blocked By Active DNC

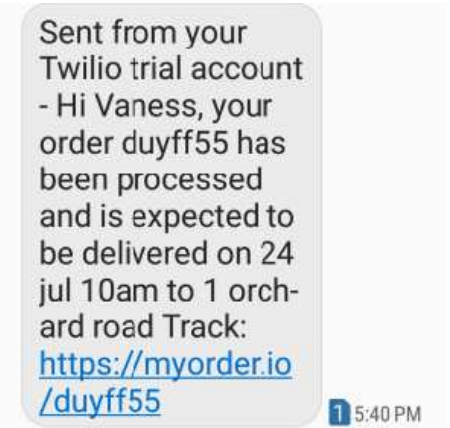
<b>Date</b>	2021-09-11 12:32:11 PM		
<b>Test Case ID</b>	T001C		
<b>Test Scenario</b>	Verify system is able to send OTP even if it is in active DNC		
<b>Test Steps</b>	Using any REST Client, execute with the test data.  A) Send MM B) Send OTP immediately after MM message is received.		
<b>Test Data</b>	JSON data as per OTP, MM template.		
<b>Expected Results</b>	<b>Actual Results</b>	<b>Pass/Fail</b>	<b>Screen Log</b>
MM message is received  OTP message is also received immediately	As Expected.	Pass	

#### 6.4.4 T002A - Verify Create Message - OUN

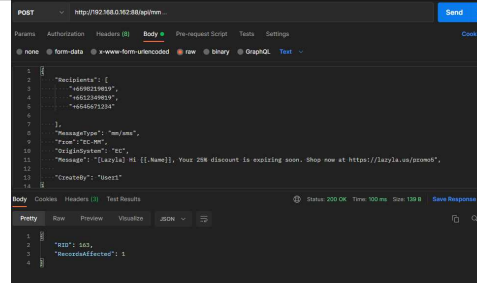
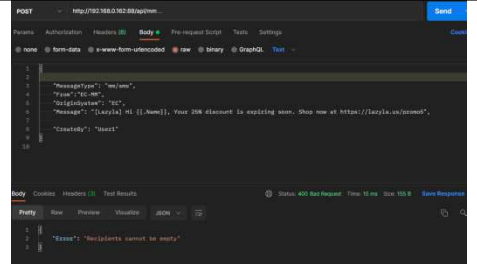
<b>Date</b>	2021-09-11 12:12:11 PM		
<b>Test Case ID</b>	T002A		
<b>Test Scenario</b>	Verify system is able to create the message type - OUN.		
<b>Test Steps</b>	Using any REST Client, execute with the test data		
<b>Test Data</b>	JSON data as per screenshot.		
<b>Expected Results</b>	<b>Actual Results</b>	<b>Pass/Fail</b>	<b>Screen Log</b>
Record ID(RID) is seen.	As Expected.	Pass	
Multiple Recipients not allowed for OTP.	As Expected.	Pass	

Recipient cannot be empty.	As Expected.	Pass	
Origin System cannot be empty.	As Expected.	Pass	
From cannot be empty.	As Expected.	Pass	
Bad json.	As Expected.	Pass	

**6.4.5 T002B - Verify Send Message - OUN**

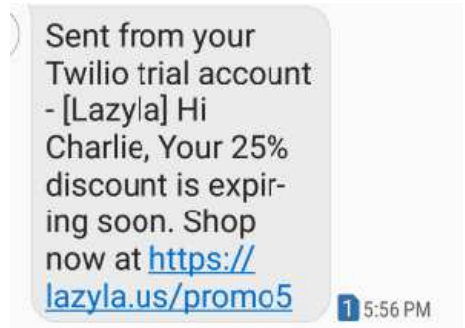
<b>Date</b>	2021-09-11 13:02:11 PM		
<b>Test Case ID</b>	T002B		
<b>Test Scenario</b>	Verify system is able to send the message type.		
<b>Test Steps</b>	Using any REST Client, execute with the test data.		
<b>Test Data</b>	JSON data as per screenshot.		
<b>Expected Results</b>	<b>Actual Results</b>	<b>Pass/Fail</b>	<b>Screen Log</b>
OUN message is received.	As Expected.	Pass	

### 6.4.6 T003A - Verify Create Message - MM

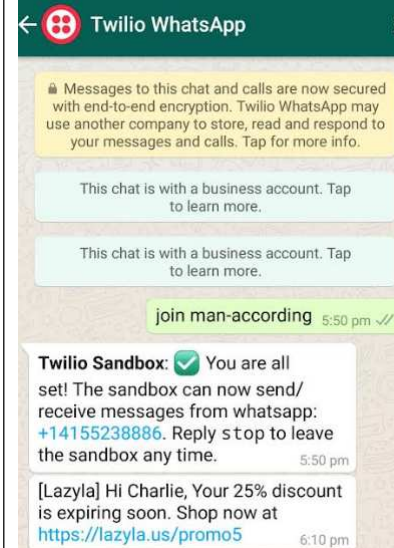
<b>Date</b>	2021-09-11 13:32:11 PM		
<b>Test Case ID</b>	T003A		
<b>Test Scenario</b>	Verify system is able to create the message type		
<b>Test Steps</b>	Using any REST Client, execute with the test data		
<b>Test Data</b>	JSON data as per screenshot.		
<b>Expected Results</b>	<b>Actual Results</b>	<b>Pass/Fail</b>	<b>Screen Log</b>
Record ID(RID) is seen.	As Expected.	Pass	
Recipient cannot be empty.	As Expected.	Pass	

Origin System cannot be empty.	As Expected.	Pass	
From cannot be empty.	As Expected.	Pass	
Bad json.	As Expected.	Pass	

**6.4.7 T003B - Verify Send SMS Message - MM**

<b>Date</b>	2021-09-11 13:52:11 PM		
<b>Test Case ID</b>	T003B		
<b>Test Scenario</b>	Verify system is able to send the message type.		
<b>Test Steps</b>	Using any REST Client, execute with the test data.		
<b>Test Data</b>	JSON data as per screenshot.		
<b>Expected Results</b>	<b>Actual Results</b>	<b>Pass/Fail</b>	<b>Screen Log</b>
OTP message is received.	As Expected.	Pass	

#### 6.4.8 T003C - Verify Send WhatsApp Message - MM

<b>Date</b>	2021-09-11 14:32:11 PM		
<b>Test Case ID</b>	T003C		
<b>Test Scenario</b>	Verify system is able to send the message type.		
<b>Test Steps</b>	Using any REST Client, execute with the test data.		
<b>Test Data</b>	JSON data as per screenshot.		
<b>Expected Results</b>	<b>Actual Results</b>	<b>Pass/Fail</b>	<b>Screen Log</b>
MM message is received.	As Expected.	Pass	 <p>The screenshot shows a Twilio WhatsApp chat interface. At the top, there's a header with the Twilio logo and 'Twilio WhatsApp'. Below the header, there are several system messages in yellow and light blue bubbles, including one about end-to-end encryption and another about business accounts. A green bubble contains the text 'join man-according' with a timestamp of 5:50 pm and a double-checkmark. Below that, a white bubble from 'Twilio Sandbox' with a green checkmark icon says 'You are all set! The sandbox can now send/receive messages from whatsapp: +14155238886. Reply s top to leave the sandbox any time.' with a timestamp of 5:50 pm. At the bottom, another white bubble from '[Lazyla]' says 'Hi Charlie, Your 25% discount is expiring soon. Shop now at https://lazyla.us/promo5' with a timestamp of 6:10 pm.</p>

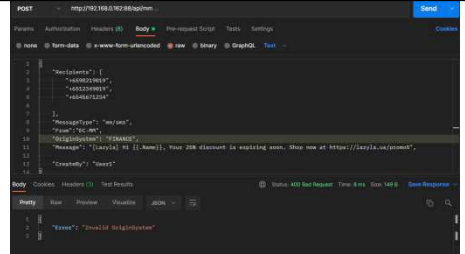


**6.4.9 T004 - Verify Gateway Routing**

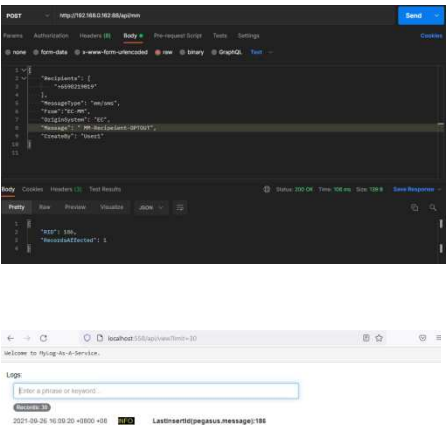
<b>Date</b>	2021-09-12 10:00:00 AM		
<b>Test Case ID</b>	T004		
<b>Test Scenario</b>	Verify system able to route message delivery based on business rules.		
<b>Test Steps</b>	Using any REST Client, execute with the test data A) numbers starting with +65123 are sent to Telco A B) numbers starting with +65456 are sent to Telco B c) System A may only send messages through Gateway X		
<b>Test Data</b>	JSON data as per screenshot.		
<b>Expected Results</b>	<b>Actual Results</b>	<b>Pass/Fail</b>	<b>Screen Log</b>
		Pass	

**DEFERRED**

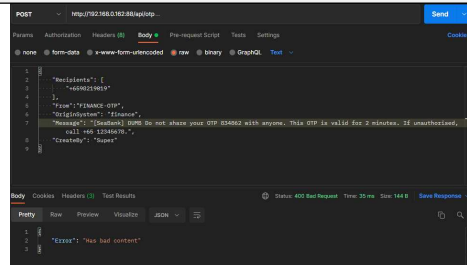
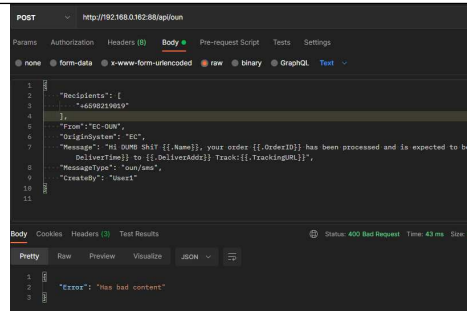
#### 6.4.10 T005 - Verify Authorized Use

<b>Date</b>	2021-09-12 10:00:00 AM		
<b>Test Case ID</b>	T005		
<b>Test Scenario</b>	Verify system allows only authorized source systems.		
<b>Test Steps</b>	Using any REST Client, execute with the test data		
<b>Test Data</b>	JSON data as per screenshot.		
<b>Expected Results</b>	<b>Actual Results</b>	<b>Pass/Fail</b>	<b>Screen Log</b>
Origin System cannot be FINANCE.	As Expected.	Pass	 <pre> 1 POST http://192.168.1.102:8080/api/v1 2 { 3   "sourceSystem": "FINANCE", 4   "targetSystem": "FINANCE", 5   "message": "Test message" 6 } 7 8 Status: 200 OK 9 Content-Type: application/json 10 { 11   "message": "Test message" 12 } 13 </pre>

### 6.4.11 T006 - Verify Legitimate Recipients

Date	2021-09-12 10:15:00 AM								
Test Case ID	T006								
Test Scenario	Verify system to send messages to legitimate recipients.								
Test Steps	Using any REST Client, execute with the test data A) DNC - opted out of marketing communications.								
Test Data	Data as per screenshot. ReasonCode is OPTOUT <table border="1"><thead><tr><th>RID</th><th>ReasonCode</th><th>ExpireTime</th></tr></thead><tbody><tr><td>+6598219019</td><td>OPTOUT</td><td>0001-01-01 00:00:00</td></tr></tbody></table>		RID	ReasonCode	ExpireTime	+6598219019	OPTOUT	0001-01-01 00:00:00	
RID	ReasonCode	ExpireTime							
+6598219019	OPTOUT	0001-01-01 00:00:00							
Expected Results	Actual Results	Pass/Fail	Screen Log						
A). No MM messages is received. From logs, MM message record is seen to be inserted.	As Expected.	Pass	<div></div>						

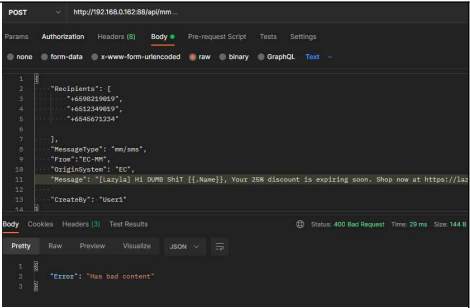
### 6.4.12 T007 - Verify Bad Content Message Cannot Be Created

<b>Date</b>	2021-09-12 10:30:00 AM		
<b>Test Case ID</b>	T007		
<b>Test Scenario</b>	Verify system does not allow bad content		
<b>Test Steps</b>	Using any REST client, execute with the test data		
<b>Test Data</b>	JSON data as per screenshot.		
<b>Expected Results</b>	<b>Actual Results</b>	<b>Pass/Fail</b>	<b>Screen Log</b>
OTP message returns 'has bad content'.	As Expected.	Pass	
OUN message returns 'has bad content'.	As Expected.	Pass	

MM message  
returns 'has bad  
content'.

As Expected.

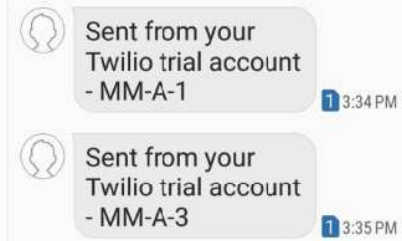
Pass



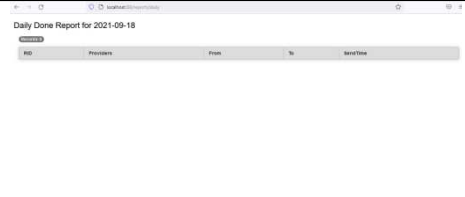
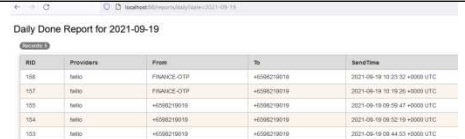
### 6.4.13 T008A - Verify DNC Auto Expiration Delete

Date	2021-09-12 10:45:00 AM											
Test Case ID	T008A											
Test Scenario	Verify system auto deletes temporary DNC on expiration, and do not delete permanent DNC.											
Test Steps	A) Insert an Expired DNC B) Insert an Expire in 1 minute DNC C) Insert an Expire in 5 minute DNC D) Insert an non-Expiration DNC  Verify logs now, 1 minute, 5 minutes, 1 hour later. Verify database that non-expired DNC record still in database											
Test Data	A) ID=past,ReasonCode=Expire1,ExpireTime=Current-1 hour B) ID=0001,ReasonCode=Expire5,ExpireTime=Current+1 minute C) ID=0005,ReasonCode=Expire5,ExpireTime=Current+5 minute D) ID=9999,ReasonCode=OPTOUT,ExpireTime=None											
Expected Results	Actual Results	Pass/Fail	Screen Log									
Temporary DNC is deleted on time, and permanent DNC is not deleted.	As Expected.	Pass	<div><div>localhost:5588/api/view/limit=30</div><div>Welcome to MyLog-As-A-Service.</div><div>Logs</div><div><div>Enter a phrase or keyword</div><div>Records: 30</div><div><div>2021-09-26 03:31:04 +0800 +08</div><div>2021-09-26 03:27:05 +0800 +08</div><div>2021-09-26 03:26:04 +0800 +08</div><div>2021-09-26 03:26:04 +0800 +08</div><div>2021-09-26 03:26:03 +0800 +08</div><div>2021-09-26 03:26:03 +0800 +08</div><div>2021-09-26 03:26:03 +0800 +08</div><div>2021-09-26 03:26:03 +0800 +08</div></div><div><div>DeleteExpiredDNC. Rows affected = 1</div><div>DeleteExpiredDNC. Rows affected = 1</div><div>DeleteExpiredDNC. Rows affected = 1</div><div>InsertDNC. Values = +000, OPTOUT, 0001-01-01 00:00:00</div><div>InsertDNC. Values = +999, Expire5, 2021-09-26 03:31:03</div><div>InsertDNC. Values = +991, Expire1, 2021-09-26 03:27:03</div><div>InsertDNC. Values = +001, Expire1, 2021-09-26 02:26:03</div><div>MPASS start ...</div></div></div><div><table><tr><th>RID</th><th>ReasonCode</th><th>ExpireTime</th></tr><tr><td>+000</td><td>OPTOUT</td><td>0001-01-01 00:00:00</td></tr><tr><td>NULL</td><td>NULL</td><td>NULL</td></tr></table></div></div>	RID	ReasonCode	ExpireTime	+000	OPTOUT	0001-01-01 00:00:00	NULL	NULL	NULL
RID	ReasonCode	ExpireTime										
+000	OPTOUT	0001-01-01 00:00:00										
NULL	NULL	NULL										

**6.4.14 T008B - Verify DNC Sending - MM Message Timeframe Throttle**

<b>Date</b>	2021-09-12 11:00:00 AM		
<b>Test Case ID</b>	T008B		
<b>Test Scenario</b>	Verify system time throttle of 1 minute is working		
<b>Test Steps</b>	Using any REST client, execute the following : Send 3 MM messages, with the 3 <sup>d</sup> message after 1 minute. A) Send message "MM-A-1" B) After 30 seconds or so, send message "MM-A-2" c) After 60 seconds, send message "MM-A-3"		
<b>Test Data</b>	<pre>{   "Recipients": ["+65XXXXXXX"],   "MessageType": "mm/sms",   "From": "EC-MM",   "OriginSystem": "EC",   "Message": " MM-B-?",   "CreateBy": "User1" }</pre> Where Message is MM-B-1,MM-B-2,MM-B-3,		
<b>Expected Results</b>	<b>Actual Results</b>	<b>Pass/Fail</b>	<b>Screen Log</b>
Only message "MM-A-1", and "MM-A-3" is received.	As Expected.	Pass	

### 6.4.15 T009 - Verify Daily Report

<b>Date</b>	2021-09-12 11:15:00 AM		
<b>Test Case ID</b>	T009		
<b>Test Scenario</b>	Verify report displays correctly the records.		
<b>Test Steps</b>	Using any REST Client, execute with the test data		
<b>Test Data</b>	A) Default, no date parameter B) Set date parameter		
<b>Expected Results</b>	<b>Actual Results</b>	<b>Pass/Fail</b>	<b>Screen Log</b>
A). Default without date parameter is display yesterday's records.	As Expected.	Pass	
B). Set date parameter displays records for transacted for that date.	As Expected.	Pass	



## 7 Product Deployment Strategy

This section describes a possible continuous deployment methodology.

## 7.1 PDCA Kanban Cycle - Continuous Improvement and Deployment

The road to a billion requests does not happen overnight, but only through planned scaling.

This project if accepted, will iterate into further cycles.

An example approach - all modern methodologies including Agile, has origins from PDCA:

Revise, Fix, Add  
Features

Release Testing



Monitor Performance,  
Resiliency, and  
Correctness

Deploy/Add a Node

## 8 Product Roadmap

This section describes possible enhancements.

## 8.1 Worker Model

The current work-flow model is based on a 'pull' model, it will not scale to the billions of requests. That will require a change to a distributed, push model.

## 8.2 Message Model

The Actor model uses message passing. Jobs information can be more formally modeled as (JSON, Binary, etc) messages and passed between the Actors(executables). Messages can be even be more standardized formally towards 'Mail' type of messages. Standardization of message model accrues payoff in the eco-system - eg. Allows talking to external systems 'naturally'.

## 8.3 Component Upgrade Model

The current design can be upgraded to higher performance in a number of ways. Exact upgrades can be done after real world performance analysis. These upgrades can include:

- Database choice
- Network upgrade(eg. Leased lines or managed pipes)
- Server upgrade
- Architectural changes
- Coding Tricks(eg. Bulk inserts)
- etc

A point for future purpose is that this project uses mySQL as a database, but it might be a potential bottleneck at very high scale, so some other databases or datastore may have to be investigated.

Nevertheless, between the schools of 'eventual consistency' and 'single source of truth', this project still prefers SQL over NoSQL or distributed databases at this point of time.

## 9 Data Templates - Messages

Messages sent are all derived from templates. New message templates can be added. The fields can be modified to match customer's requirements.

### 9.1 OTP

```
{
  "Recipients": [ "+6598XXXXXX" ],
  "From": "FINANCE-OTP",
  "OriginSystem": "finance",
  "Message": "[SeaBank] (MM,OTP) Do not share your OTP 834862 with anyone. This OTP is valid for 2 minutes. If unauthorised, call +65 12345678.",
  "CreateBy": "Super"
}
```

### 9.2 OUN

```
{
  "Recipients": [ "+6598XXXXXX" ],
  "From": "EC-OUN",
  "OriginSystem": "EC",
  "Message": "Hi {{.Name}}, your order {{.OrderID}} has been processed and is expected to be delivered on {{.DeliverTime}} to {{.DeliverAddr}} Track:{{.TrackingURL}}",
  "MessageType": "oun/sms",
  "CreateBy": "User1"
}
```

### 9.3 MM

```
{
  "Recipients": [ "+6598XXXXXX" ],
  "MessageType": "mm/sms",
  "From": "EC-MM",
  "OriginSystem": "EC",
  "Message": "MM-Recipeient-OPTOUT",
  "CreateBy": "User1"
}
```

## 10 Data Dictionary

### 10.1 Message

This table holds the information regarding the job.

ID	Type	Description
RID	String	Primary Key. Record ID.
Name	string	Message name.
MsgType	string	sms, email,etc.
Providers	string	Ordered list of preferences(eg: TelcoA, TelcoB)
From	string	Message name.
To	string	Message name.
Body	string	Message name.
Name	string	Message name.
CreateTime	time	Message name.
CreateBy	string	Message name.
OriginSystem	string	Message name.
SendTime	time	Time to send the message.
WorkerName	string	Machine name or id of worker service.
StartTime	time	Actual begin send time.
Endtime	time	Time completed.
Status	string	Job status: new,approved,validated,queued,locked,finished
Result	string	Collection of response text from job

## 10.2 DNC

This table holds data regarding DNC(Do-Not-Call).

ID	Type	Description
RID	String	Primary Key. Record ID.
ReasonCode	string	Reason of DNC.
ExpireTime	time	Time expiration of this record.

## 10.3 Report

Report table holds transacted data for downstream applications such as Accounting, Analytics, etc.

ID	Type	Description
RID	String	Primary Key. Report ID.
Details	string	A json representation of the report.



## 11 REFERENCES

1. 8 Companies That Use Elixir in Production <https://serokell.io/blog/elixir-companies>
2. PDCA Cycle. <https://kanbanize.com/lean-management/improvement/what-is-pdca-cycle>
3. Understanding actor concurrency, Part 1: Actors in Erlang.  
<https://www.infoworld.com/article/2077999/understanding-actor-concurrency--part-1--actors-in-erlang.html>
4. Share Memory By Communicating <https://go.dev/blog/codelab-share>