Go School(Ngee Ann Polytechnic)

Project

MyLog-M

(Logging-As-A-Service)

By: Chan Jiunn Hwa

Reviewers: Owen Yuwono, Lee Ching Yun, Low Kheng Hian

11th September 2021

Version 0.999

Table of Contents

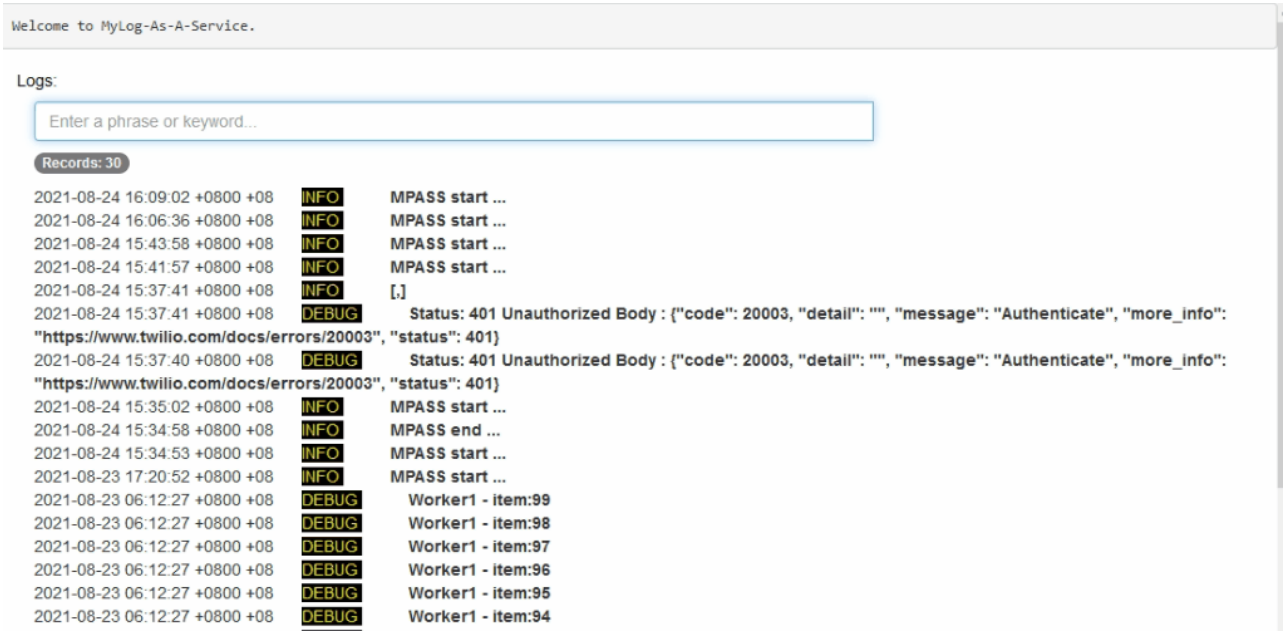
1 Executive Summary

This project MyLog-M(hereafter MyLog) is a http-based API microservice providing logging services. MyLog is an extracted module out of the main project MPASS-M(Messaging-Platform-As-A-Service).

The -M suffix in both system denotes a modified version that is slated for Prototyping, with the original version kept as a rollback option. This document covers only MyLog, whereas MPASS is covered in its own documentation.

MyLog's mission is to to provide a logging as a service. It has the following notable features:

- Logs to database, instead of traditional file
- Easy search on items of interest
- Logging as a separate service, meaning it is decoupled from the Producer Application.
- Extensible



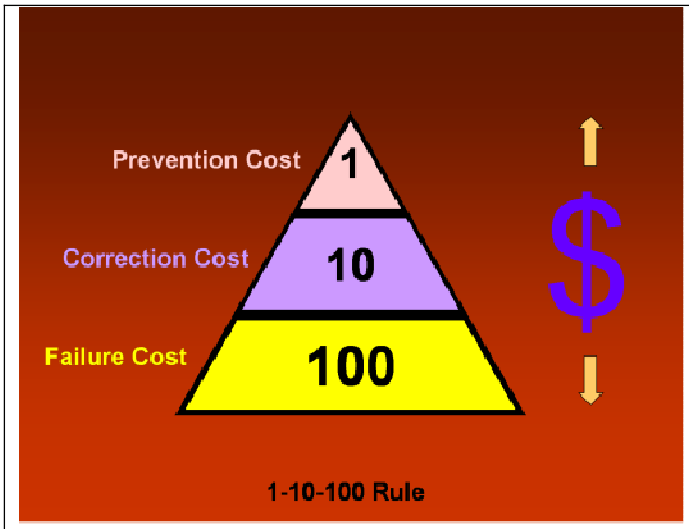
In a nutshell, instead of file logging, the user will log using a JSON Post to the carved out MyLog service which saves it into a database. The decoupling enables better handling of the quality requirement of 'billion requests a day', assuming each request can generate a number of log lines.

Also, a web log viewer allows easier grepping and faster fault tracing time, especially in production, thus increasing troubleshooting productivity. Additionally, it has rich enhancement capabilities(eg - native mini-splunk allowing log montioring, alert, dashboard, etc.)

Looking back, the point of motivation is that given the expectation that the messaging may scale eventually to billions of requests per day, file logging could prove to be a potential bottleneck. At the very minimal, if each request were to generate a few log lines, then we are looking at potentially daily log files of many of billion lines, as well mega/gigabytes in size.

Although the client company may have availability to systems like Splunk that can ease the 'log grepping' pain point, but it is still not the most efficient, and adds further layers and costs.

Further more, there is the issue of the 1-10-100 rule.



https://totalqualitymanagement.wordpress.com/2009/02/25/what-is-1-10-100-rule/

Generally, if error(s) or traces are propagated through time/space, the 'source of truth' fragmentates further, and becomes harder to assemble back the root causal chain. This means if errors or traces are not grouped near source, the risk of the 1-10-100 rule increases, and returns multiples in terms of cost, time loss, and faulty diagnosis.

Fragmentation occurs even more when we have multiple concurrent process all racing towards writing to the file, thus interleaving the traces - scattering an Event/Request across time and space. Hence, the further in time/space you are away from the source, the more the source Event/Error fades in context.

Hence it is hoped that this sub-project provides a better alternative.

2 Overview

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

MyLog 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.

MyLog will code using Robert C. Martin’s Clean Architecture, which is widely used by companies that use Golang. This is further elaborated in the Design Section.

2.1 Objective

Having experienced common issues and difficulties of traditional file logging, the objective of MyLog is to deliver a time and cost effective logging service.

2.2 Scope

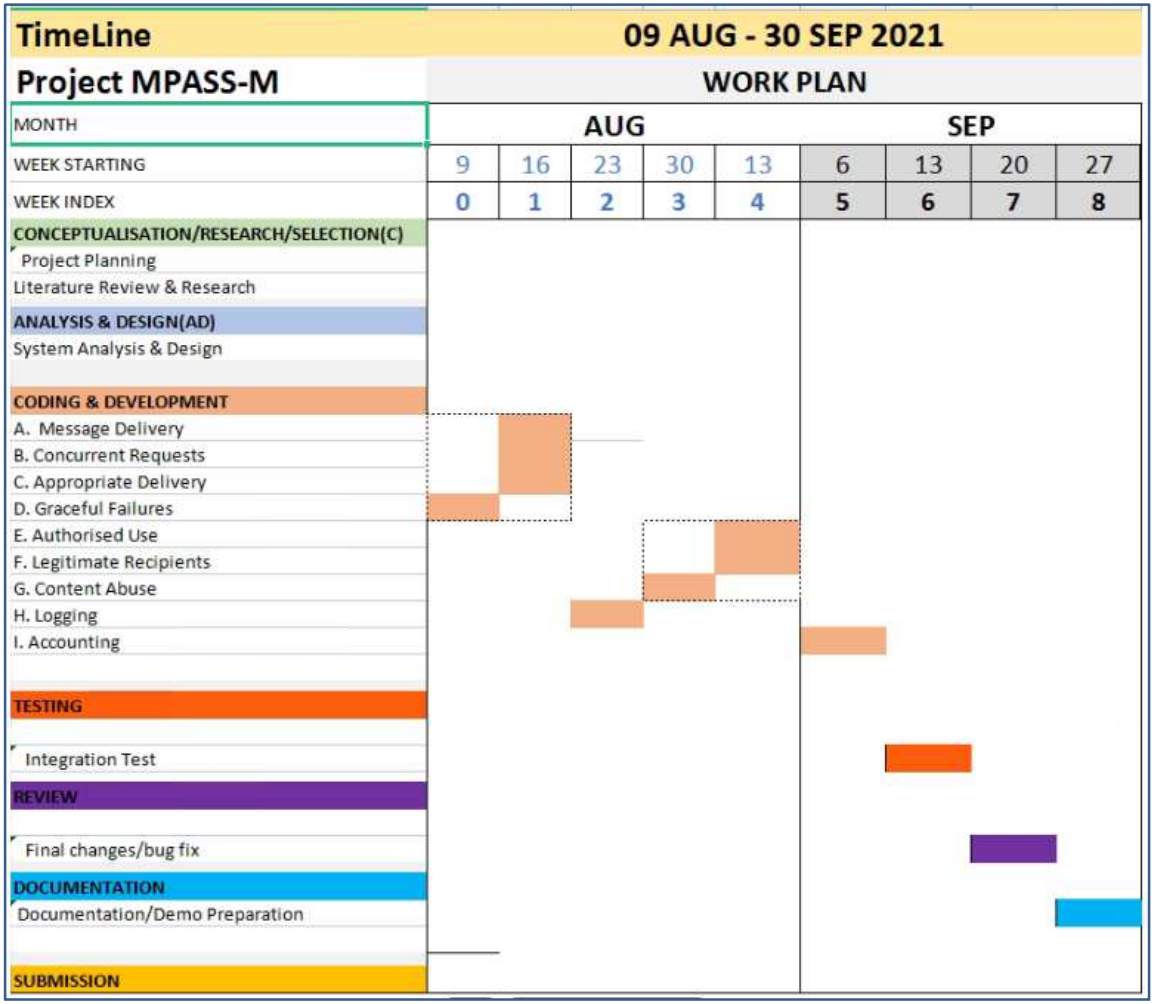
Due to time constraints, and given the fact that the developer does not come from a DevOps background, the challenge and scope is to develop a simple, yet complete, useful and usable application.

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. MyLog has been extracted as a sub-project occurring on Week Index 2.



3.2 Table of Activities

CONCEPTUALISATION/RESEARCH/SELECTION(C)	
Project Planning	**A. Creation of Project Plan** 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. From experience, consistent output comes from proportionate budgeting, and coding usually does not occupy more than 50% of the lifecycle for a balanced plan.
Literature Review & Research	**B. Background Research** Reading and research on the specific technologies to be used, and whether it is feasible.
ANALYSIS & DESIGN(AD)	
Functions	**C. Functional Analysis** Based on the background research, features are listed, though it is estimated not all can be implemented. The outputs are database design, requirements matrix. **D. Feasibility Analysis** Scoping and feasibility has to be estimated.
CODING	
Sprint	**E. Code Methodology** 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.

	F. Risk Management To manage timeline risk, each feature(f1,…) 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.
TESTING	
Planning & Design Integration Test	**G. Test Case Design and Execution** Unit, Regression and Integration are included. .
REVIEW	
Final review Final changes and fix	**H. Review & Fix** Code freeze, and bug fixes only.
DOCUMENTATION	
Planning Document Structure Setup Record Final Review	**I. Documentation** 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.

SUBMISSION	**J. <u>Zip and Upload</u>** By 2359 30th 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 http logging service..
- Documentation - this report.

4 System Analysis

Functional analysis is to box in an overall coherent release, and yet allows roadmap planning of future features.

4.1 Functional Features

This version will provide the following primary features:

4.1.1 Log Insert

The system shall provide an interface for user to insert a record.

4.1.2 Log Read

The system shall provide an interface for user to retrieve an existing record.

4.1.3 Retrieval Of Tail Lines of the Log

The system shall provide an interface for user to receive a response comprising of the tail slice of a log

4.1.4 Viewing the Log

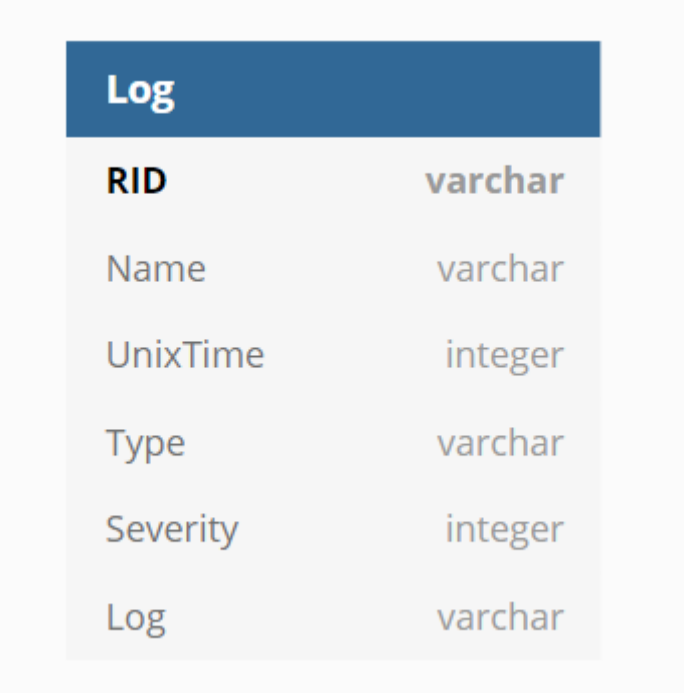
The system shall provide an interface for user to view a slice of the log.

4.1.5 Filtering the Log

The system shall provide an interface for user to further filter a subset from a retrieved slice of the log.

4.2 System Architecture Design

4.2.1 Database Schema

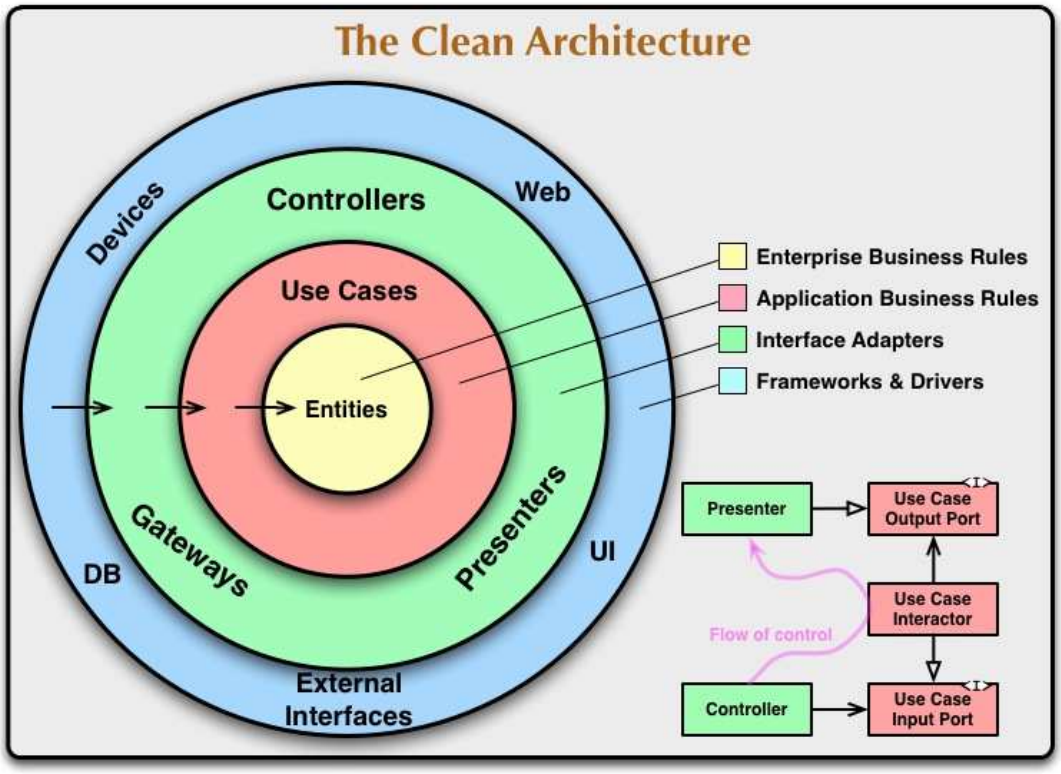


5 System Design & Development

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

5.1 Clean Architecture

As per guided by Owen, this project is written following the concepts of Clean Architecture.

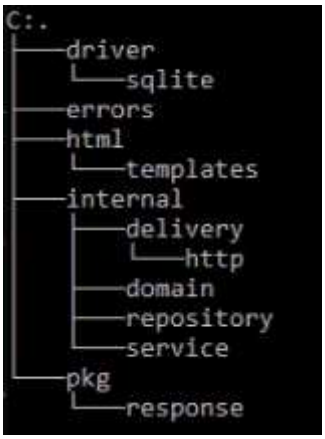


The objective, is to achieve better maintainability via the separation of concerns. This separation is achieved by dividing the software into layers. In this project, a 3 layer approach is taken:

- Delivery
- UseCase
- Repository

5.2 Project Schema

The following shows the organization of the project.



5.3 Functions

The system interacts primarily through JSON data exchange but also provides a web view for human interface.

5.3.1 Log Insert (myLog)

MyLog insert a record when a user does a http Post of a JSON record.

```go
//MyLog gets a record by id, or insert a posted record
func (h Handler) MyLog(w http.ResponseWriter, r *http.Request) {
    store := repository.NewLogRepo(h.db)
    if r.Method == http.MethodGet {
        ID := -1
        if n, err := strconv.Atoi(strings.ToUpper(r.URL.Query().Get("id"))); err == nil {
            ID = n
        }
        if ID < 0 {
            response.AsJSONError(w, http.StatusMethodNotAllowed, "Invalid id")
            return
        }
        resp, err := store.Get(int64(ID))
        log.Println(resp)
        if err != nil {
            response.AsJSONError(w, http.StatusMethodNotAllowed, err.Error())
            return
        }
        response.AsJSON(w, 200, resp)
        return
    }
    if r.Method == http.MethodPost {
        var data repository.Data
        err := json.NewDecoder(r.Body).Decode(&data)
        if err != nil {
            response.AsJSONError(w, 200, err.Error())
            return
        }
        LastID, err := store.Insert(data)
        if err != nil {
            response.AsJSONError(w, 200, err.Error())
            return
        }
        resp, err := store.Get(LastID)
        if err != nil {
            response.AsJSONError(w, 200, err.Error())
            return
        }
        log.Println(resp)
        response.AsJSON(w, 200, resp)
```

handler.go

5.3.2 Log Read(myLog)

MyLog retrieve an existing record by id provided by the user.

```go
//MyLog gets a record by id, or insert a posted record
func (h Handler) MyLog(w http.ResponseWriter, r *http.Request) {
    store := repository.NewLogRepo(h.db)
    if r.Method == http.MethodGet {
        ID := -1
        if n, err := strconv.Atoi(strings.ToUpper(r.URL.Query().Get("id"))); err == nil {
            ID = n
        }
        if ID < 0 {
            response.AsJSONError(w, http.StatusMethodNotAllowed, "Invalid id")
            return
        }
        resp, err := store.Get(int64(ID))
        log.Println(resp)
        if err != nil {
            response.AsJSONError(w, http.StatusMethodNotAllowed, err.Error())
            return
        }
        response.AsJSON(w, 200, resp)
        return
    }
    if r.Method == http.MethodPost {
        var data repository.Data
        err := json.NewDecoder(r.Body).Decode(&data)
        if err != nil {
            response.AsJSONError(w, 200, err.Error())
            return
        }
        LastID, err := store.Insert(data)
        if err != nil {
            response.AsJSONError(w, 200, err.Error())
            return
        }
        resp, err := store.Get(LastID)
        if err != nil {
            response.AsJSONError(w, 200, err.Error())
            return
        }
        log.Println(resp)
        response.AsJSON(w, 200, resp)
```

handler.go

5.3.3 Retrieval Of Tail Lines of the Log(myTail)

MyTail returns as JSON the last limit number of records, with default limit=1

```go
//MyTail returns as json the last limit number of records, with default limit=1
func (h Handler) MyTail(w http.ResponseWriter, r *http.Request) {
    store := repository.NewLogRepo(h.db)
    if r.Method == http.MethodGet {
        limit := 1 //default show lastest 1
        if n, err := strconv.Atoi(strings.ToUpper(r.URL.Query().Get("limit"))); err == nil {
            limit = n
            log.Println("limit:", limit)
        }
        resp, err := store.Tail(int64(limit))
        if err != nil {
            response.AsJSONError(w, 200, err.Error())
            return
        }
        log.Println(resp)
        response.AsJSON(w, 200, resp)
        return
    }
    response.AsJSONError(w, http.StatusMethodNotAllowed, "Invalid action")
}
```

handler.go

5.3.4 Viewing the Log(myView)

MyView is a web view that shows the last limit number of records, with default limit=1.

```go
//MyView shows the last limit number of records, with default limit=1
func (h Handler) MyView(w http.ResponseWriter, r *http.Request) {
    tmpl := LoadTemplate(tplDir, "view.gohtml")
    store := repository.NewLogRepo(h.db)
    if r.Method == http.MethodGet {
        limit := 1 //default show lastest 1
        if n, err := strconv.Atoi(strings.ToUpper(r.URL.Query().Get("limit"))); err == nil {
            limit = n
        }
        recs, err := store.Tail(int64(limit))
        if err != nil {
            fmt.Fprintf(w, err.Error())
            return
        }
        viewData := &ViewData{PageTitle: "VIEW", Records: *recs, RowCount: len(*recs)}
        tmpl.Execute(w, viewData)
        return

    }
    response.AsJSONError(w, http.StatusMethodNotAllowed, "Invalid action")
}
```

handler.go

5.3.5 Filtering the Log(myView)

User can further filter('grep') for keywords within the displayed lines, implemented in javascript.This script is injected upon Template.Execute()

```go
//MyView shows the last limit number of records, with default limit=1
func (h Handler) MyView(w http.ResponseWriter, r *http.Request) {
    tmpl := LoadTemplate(tplDir, "view.gohtml")
    store := repository.NewLogRepo(h.db)
    if r.Method == http.MethodGet {
        limit := 1 //default show lastest 1
        if n, err := strconv.Atoi(strings.ToUpper(r.URL.Query().Get("limit"))); err == nil {
            limit = n
        }
        recs, err := store.Tail(int64(limit))
        if err != nil {
            fmt.Fprintf(w, err.Error())
            return
        }
        viewData := &ViewData{PageTitle: "VIEW", Records: *recs, RowCount: len(*recs)}
        tmpl.Execute(w, viewData)
        return

    }
    response.AsJSONError(w, http.StatusMethodNotAllowed, "Invalid action")
}
```

handler.go

```
        function myFuncB(ulID,filter){
            ul = document.getElementById( ulID);
            console.log(ulID, ul);
            li = ul.getElementsByTagName('li');
            for (i = 0; i < li.length; i++) {
                txtValue = li[i].innerText;
                if (txtValue.toUpperCase().indexOf(filter) > -1) {
                    li[i].style.display = "";
                } else {
                    li[i].style.display = "none";
                }
            }
        }
```

view.gohtml

6 SYSTEM TESTING

6.1 Unit Test

By Golang convention, unit test are performed with the built-in test runner. Respective unit tests were done and can be re-run and the results outputted ad-hoc.

6.2 Integrated Test Cases

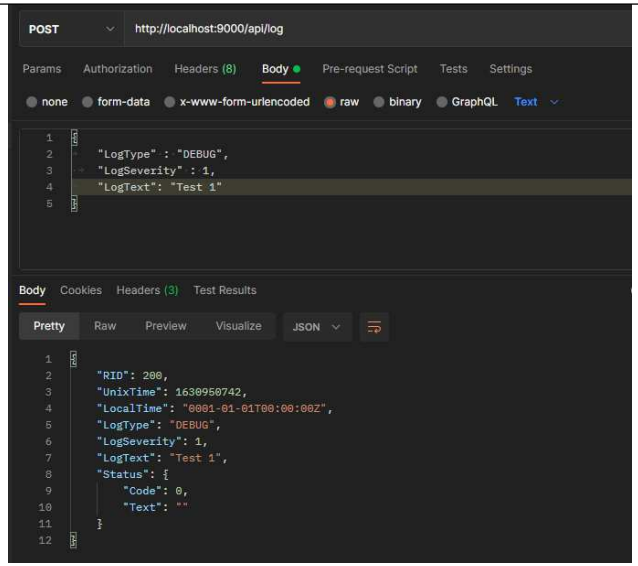
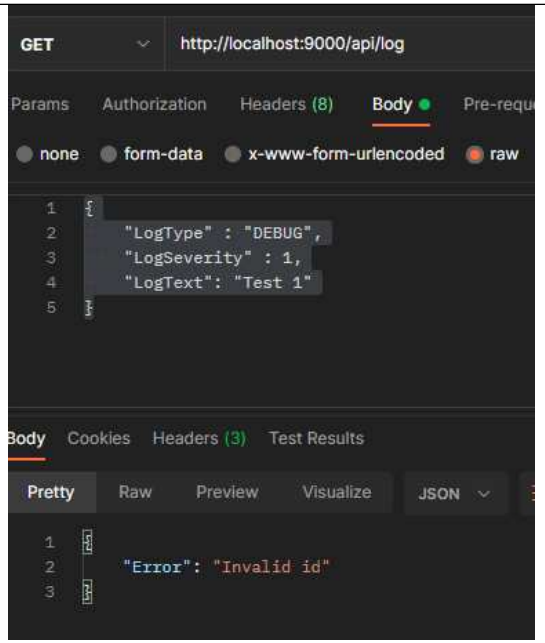
For this project, the testing strategy will be an ‘and-to-end’ test of the various API calls to verify results are as expected.

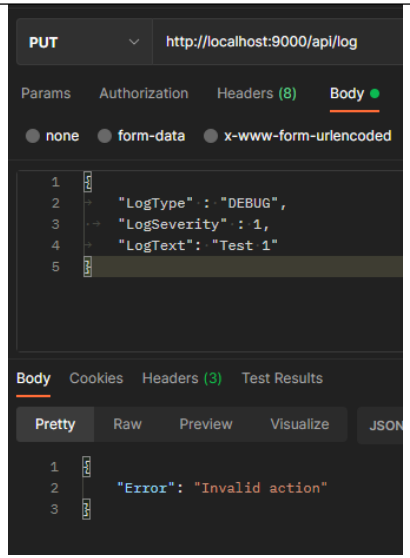
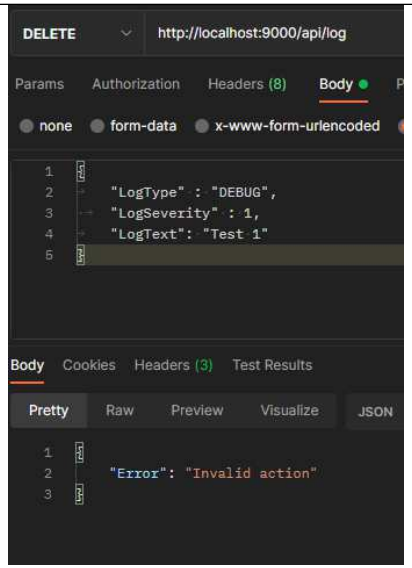
6.2.1 T001 Verify myLog/Post Can Insert A Record

Date	2021-09-11 09:32:11 AM		
Test Case ID	T001		
Test Scenario	Verify API is able to insert a record.		
Test Steps	Using any REST client, execute with the test data, using the following http methods: A. POST B. GET C. PUT D. DELETE		

Test Data	{
	"LogType" : "DEBUG",
	"LogSeverity" : 1,
	"LogText": "Test 1"
	}		

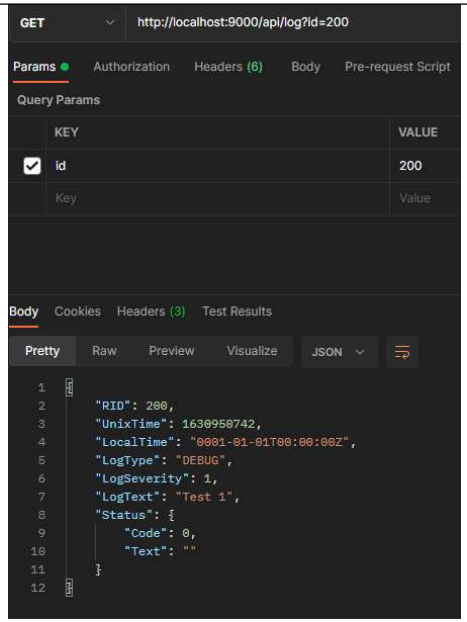
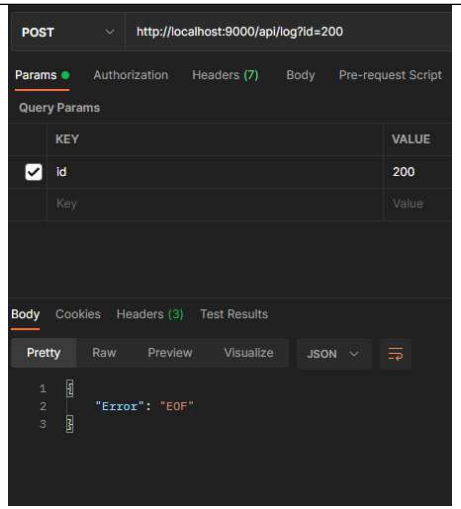
Expected Results	**Actual Results**	**Pass/Fail**	**Screen Log**

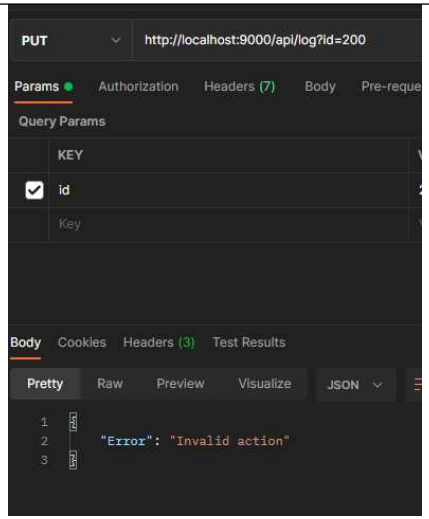
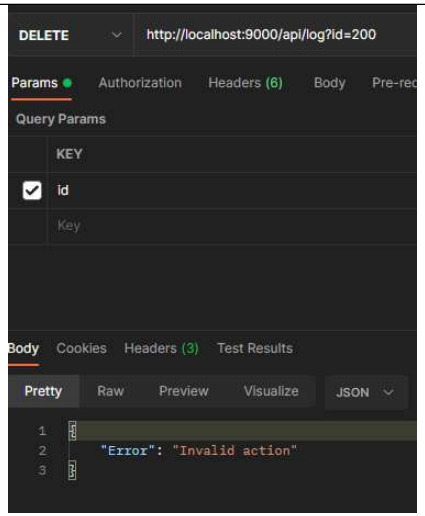
A). Response with RID = 200	As Expected.	Pass	
B). Response with “Invalid ID”	As Expected.	Pass	

C). Response with “Invalid Action”	As Expected.	Pass	
D). Response with “Invalid Action”	As Expected.	Pass	

6.2.2 T002 Verify myLog/Get Can Fetch A Record

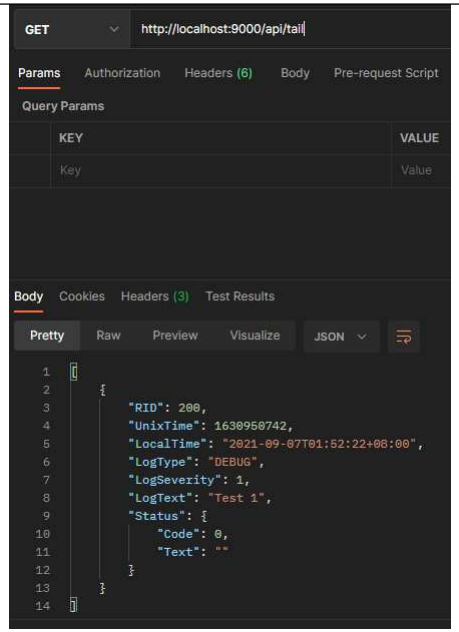
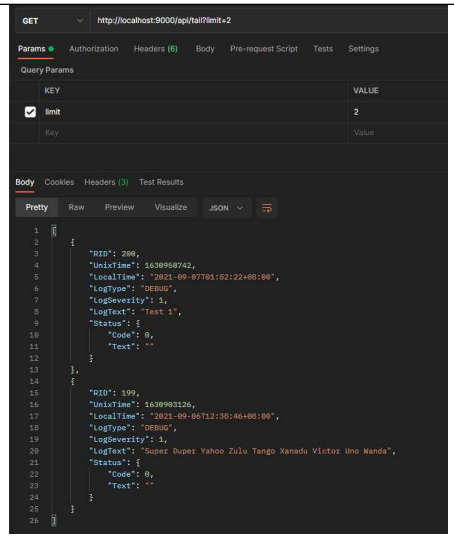
Date	2021-09-11 09:45:11 AM		
Test Case ID	T002		
Test Scenario	Verify API is able to read a record by ID		
Test Steps	Using any REST client, execute with the test data, using the following http methods: A) GET B) POST C) PUT D) DELETE		
Test Data	ID=200		
Expected Results	**Actual Results**	**Pass/Fail**	**Screen Log**

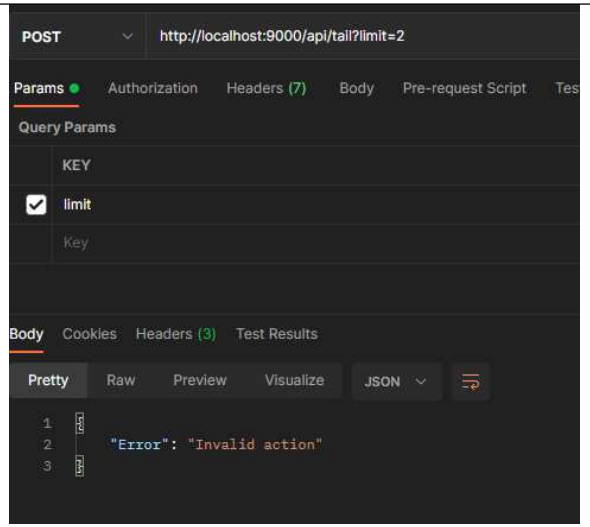
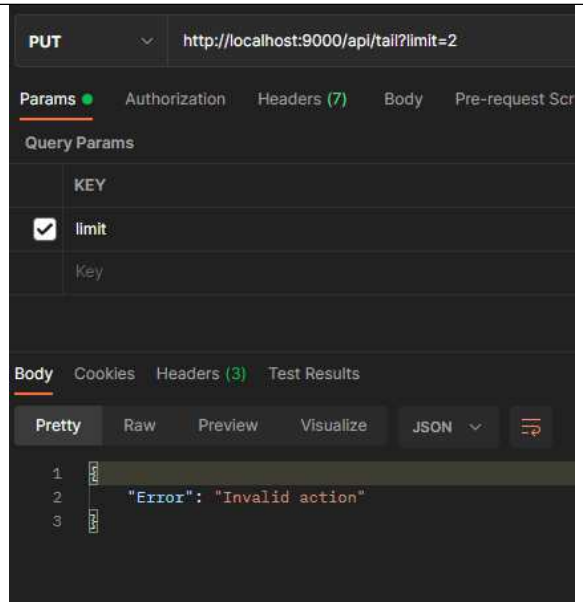
A). Response with RID = 200	As Expected.	Pass	
B). Response with “EOF	As Expected.	Pass	

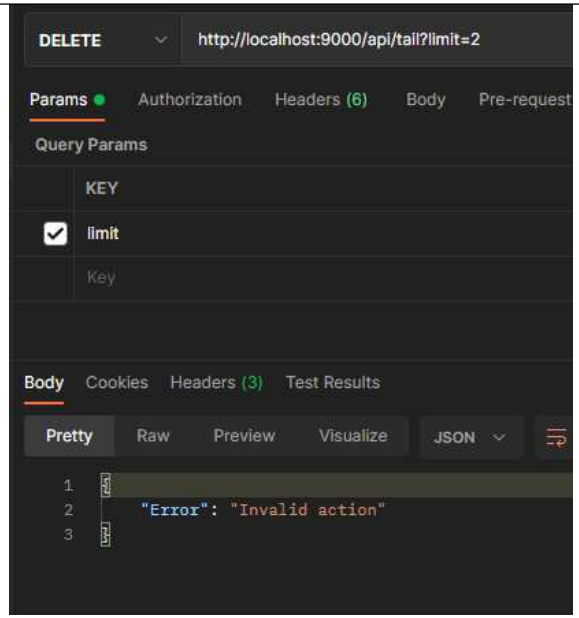
C). Response with “Invalid Action”	As Expected.	Pass	
D). Response with “Invalid Action”	As Expected.	Pass	

6.2.3 T003 Verify myTail Can Retrieve A Set Of Records

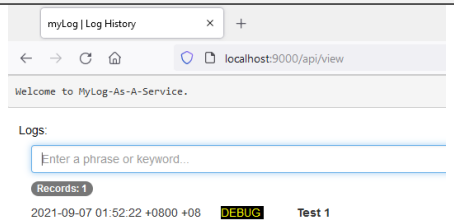
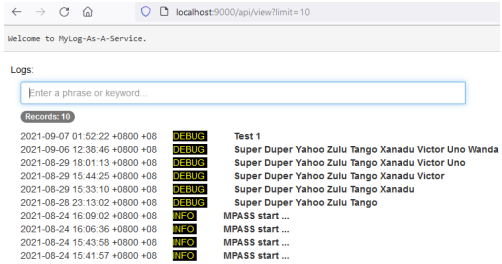
Date	2021-09-11 10:02:11 AM		
Test Case ID	T003		
Test Scenario	Verify API is able to read the tail slice of records		
Test Steps	Using any REST client, execute with the test data, using the following http methods: A) GET (do not specify limit) B) GET (limit=2) C) POST D) PUT E) DELETE		
Test Data	Limit=N		
Expected Results	**Actual Results**	**Pass/Fail**	**Screen Log**

A). Response with RID = 200	As Expected. Last record is returned.	Pass	
A). Response with RID = 200,199	As Expected. Limit records are returned.	Pass	

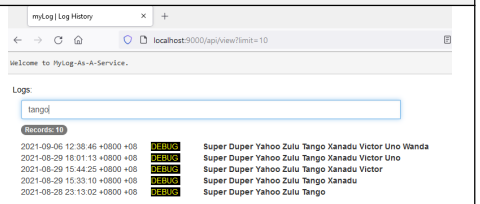
B). Response with “Invalid Action”	As Expected.	Pass	
C). Response with “Invalid Action”	As Expected.	Pass	

D). Response with “Invalid Action”	As Expected.	Pass	

6.2.4 T004 Verify myView Can Display A Set Of Records

Date	2021-09-11 10:32:11 AM		
Test Case ID	T004		
Test Scenario	Verify API is able to fetch a number of records		
Test Steps	Using any web browser, execute with the test data A) No limit specified B) Limit=10		
Test Data	Limit=N		
Expected Results	**Actual Results**	**Pass/Fail**	**Screen Log**
A). Last record is displayed.	As Expected.	Pass	
B). Last records as per Limit are displayed.	As Expected.	Pass	

6.2.5 T005 Verify myView Can Filter A Set Of Records

Date	2021-09-11 11:32:11 AM		
Test Case ID	T002		
Test Scenario	Verify API is able to filter down by keyword		
Test Steps	Using any web browser, execute with the test data A) Limit=10, filter=tango		
Test Data	Limit=N		
Expected Results	**Actual Results**	**Pass/Fail**	**Screen Log**
A). Last records as per Limit are displayed. And records are filtered	As Expected.	Pass	

7 Product Roadmap

This prototype can be extended. The below is documented to allow future research.

7.1 UI

7.1.1 Dashboard

A dashboard can be added to house all the aggregated information flows.

7.1.2 Alarms And Alerts

Alarms and Alerts can be added to provide automatic monitoring and alerts of certain key events, thresholds, etc.

7.2 Distribution Model

7.2.1 Logging Redundancy, Rotation,

The service can be deployed on multiple servers for logging redundancy or rotation

7.2.2 Log Topic

Specific servers can be set up to log by topic.

7.3 Transport Model

7.3.1 Connections

A simple load test shows a benchmark of 10 inserts/sec for the un-optimised code on a low-end machine(ie. Notebook)

If higher connection performance is required, the system can be upgraded to have

- Connection pooling
- Websockets
- Protobuf
- Message queuing
- Database upgrade or change
- Hardware, network upgrade
- Use of faster http lib like fasthttp
- etc.

The upgrade choice will depend on further analysis on the actual physical bottleneck.

7.4 Data Model

7.4.1 Custom Tagging

The benefit of database logging is that we can directly tag and index fields of interest. File logging with tags suffer from further layers of ingestion into another system which can add ambiguities, noise, time and cost.

8 Data Dictionary

8.1 Log

This data schema is similar to SysLog structure.

ID	Type	Description
RID	String	Primary Key. Record ID.
Name	string	Event name.
Description	string	A short summary of the course.
UnixTime	Int	Unix timestamp
Type	string	Log type.
Severity	Int	Log severity level
Log	String	Log text content.

9 REFERENCES

1. The Clean Architecture. https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html

2. Clean Architecture in Android. https://sanaebadi97.medium.com/clean-architecture-in-android-1026f66661fb

3. The True Cost of “Search-First” Problem-solving on Your Production Systems. https://www.honeycomb.io/blog/the-true-cost-of-search-first-problem-solving-on-your-production-systems/

4. What is 1-10-100 Rule? https://totalqualitymanagement.wordpress.com/2009/02/25/what-is-1-10-100-rule/