

Session 09 Security Report

Group J (noname)

April 2020

1 Introduction

This report contains information about the exercises from session 09. The content is about security of our own and an opposing group K's systems including monitoring of their system based on their SLA.

2 Start monitoring each others SLA

Our target group was group K aka "DevOps drengene".

SLA

The following document describes the service-level agreement (SLA) for the [simulator API](#).

Monthly Uptime Percentage

We strive to deliver a monthly **99,99% uptime**, meaning that the API is able to serve valid responses to its client in this timeframe.

Mean Time To Recover

In case of outage, the recovery time is expected to be around **12 hours**.

Average Response Times

The expected average response time on API calls on the following API routes:

- GET `/latest` : **100ms**
- POST `/msgs/{username}` : **100ms**
- GET `/fills/{username}` : **100ms**
- POST `/fills/{username}` : **150ms**
- GET `/msgs` : **5s**
- GET `/msgs/{username}` : **700ms**

This is targeted clients located in Europe only, and is measured on a monthly basis.

Error Rates

The average monthly number of valid requests from client that results in response with HTTP status code ≥ 500 is **0,1%**.

2.1 Our monitoring approach

We did this by writing a python script. The purpose of this script is to send requests to confirm that their reported response times, uptime and error rates are upheld.

We have decided to setup and run the script from our backup server, which is running on one of our personal servers. This is to ensure that work done on our own project/system will not affect the work of monitoring their system.

For the test script, see appendix A — *monitor.py*

A short explanation of the script: For each of the endpoints in the SLA, we measure the request-response time, calculate the average (based on all our previous requests to that endpoint) and log the time. If the average exceeds the SLA, we log it in red. In order to not flood their system with requests, we make the script sleep for 60 seconds before making new requests.

Since this monitoring is not hooked up to any third party visualization or monitoring tool, we will need to extract the numbers and visualize them manually. This process of course is prone to errors as we will have to code and setup everything ourselves. Getting to a graphical state with the data takes a lot of effort even using libraries and wiring everything up. We chose a reasonably formatted output from the monitoring script, but still the final log containing thousands of lines needs to be post-processed and turned into eg. a .csv file that again needs to be handled. To cut it short we chose to feed the .csv file to Google Spreadsheets which will easily parse such a format and turn it into readable graphs.

The part of the script that creates a .csv using awk is shown below.
The rest is shown in appendix B.

```
awk '
{date=$1;datetime=$2}
{gsub("\\[", "", date); gsub("\\]", "", datetime)}
{endpoint=$3}
{print date, "datetime ", " endpoint ", " $5 ", " $7}
' monitor.log | grep -v 'errors' >> monitor.csv
```

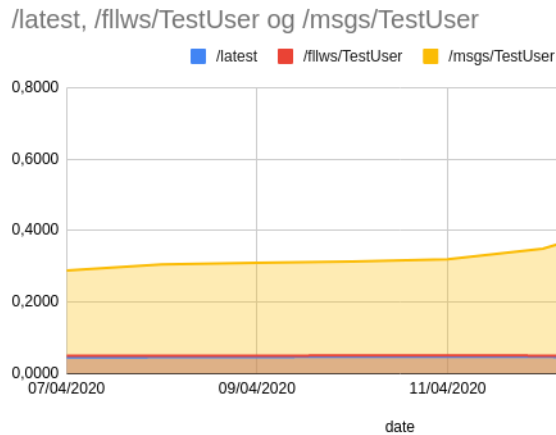
The post-processing is made using a BASH script that prints some basic statistics and turns the logged entries into the more usable .csv file format. This code could potentially be connected to other scripts to automate the process of getting the data from the backup server to e.g. somewhere on Github for the whole team to scrutinize. This is however not setup.

2.2 Results and conclusion

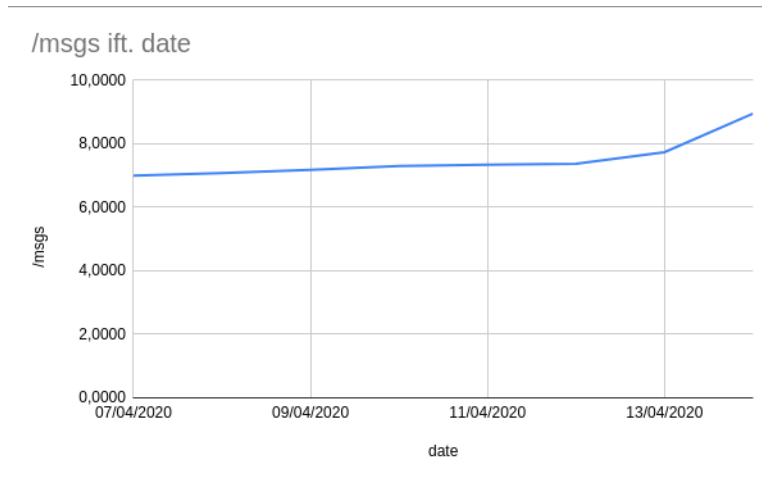
The following shows the average response time of the monitored API for each endpoint.

GET	/latest	0.0467
POST	/msgs/{username}	0.0511
GET	/flws/{username}	0.0504
POST	/flws/{username}	0.0509
GET	/msgs	7.42
GET	/msgs/{username}	0.666

The API was monitored for just under a week. Group K state that their specified average response time are on a monthly basis so we cannot conclude whether the SLA is upheld. However, based on our monitoring the expected average response time is not upheld for the endpoint GET /msgs. It is stated in the SLA that we should expect an average response time of 5s but our monitoring shows an average response time of 7.42s. The other endpoints respond within the expected time.



As seen the response time has a high average, we speculate that this is because they return too many messages on this particular endpoint.



3 Security assessment of our system

3.1 Assets of our system

Below is a list of the assets of our system and some vulnerabilities associated with the assets.

Physical assets

- Digital ocean server
 1. Natural causes: The physical server(s) gets destroyed, happens on rare occasions and has a catastrophic impact.
 2. External errors: Power/network outage compromising availability, it is unlikely to happen and has a critical impact.

Logical assets

- App and API
 1. Human error: Update/deploy error is possible but has a marginal impact.
 2. SQL injection: can corrupt or leak data, unlikely but has a critical impact.
 3. DOS attacks: Compromises the availability and is possible and has a critical impact.
 4. Cross-site scripting: Compromises the integrity and is possible and has a marginal impact.

- Database
 1. Deletion of the database: All business data will be lost
 2. Compromised data: If adversaries successfully launch an attack that leaks information about private user information.

External logical Assets

- Circle CI / Docker Hub / GitHub
 1. Service down: The service can go down / offline.

Damage to our Circle CI, Docker hub and GitHub services will not affect the App and API directly but it will impact productivity of development of the application. These are all examples of external services, that are out of our hands to control, if a service goes down. For example if Docker Hub goes down, we will not be able fix it, we have to accept the risk, and the consequence of deployments getting paused.

3.2 Create a risk matrix for your project

	Negligible	Marginal	Critical	Catastrophic
Certain	<i>Medium</i>	<i>High</i>	<i>High</i>	<i>High</i>
Likely	<i>Low</i>	<i>Medium</i>	<i>High</i>	<i>High</i>
Possible	<i>Low</i>	<i>Medium</i>	<i>Medium</i>	<i>High</i>
Unlikely	<i>Low</i>	<i>Low</i>	<i>Medium</i>	<i>High</i>
Rare	<i>Low</i>	<i>Low</i>	<i>Low</i>	<i>Medium</i>

Table 1: Risk matrix

Definitions of Severity

Negligible: Minor inconvenience - e.g. a few seconds extra loading time

Marginal: Greater annoyance

Critical: System unavailable

Catastrophic: Permanent damage

Definitions of likelihood

Rare: Not expected to happen

Unlikely: Might happen

Possible: Happens sometimes

Likely: Happens often

Certain: Happens almost all the time

The above matrix shows the amount of risk involved with how likely or frequent the event of a vulnerability being exploited is and how severe the consequences are if they are exploited. The matrix helps prioritizing protection of the assets.

To help us we can ask the questions: What can we do to reduce the chance of this happening? And what can we do to reduce the severity if the event occurs? An example could be the human error of deleting the database:

What can we do to reduce the likelihood of the database being accidentally deleted?

- Restricting access to certain operations, and minimising the amount of people interacting directly with the database.

What can we do to reduce the severity if the database is deleted?

- By creating backup data.

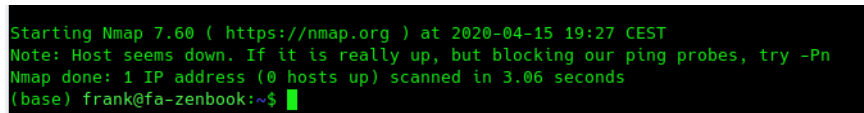
These are simple ways to address security risks whether they are vulnerable to hackers or human errors.

3.3 Pen testing our system

We have taken inspiration from the The Open Web Application Security Project (OWASP) top ten document that describes some vital security risks in web app development.

Some of the things we will be looking at are SQL injection attacks and security configurations in terms of vulnerable open ports. To test our own system with the view of an outsider we chose to begin with a basic port-scan using the tool `nmap`.

`nmap 46.101.215.40`

A terminal window with a black background and green text. The text shows the output of an nmap command: 'Starting Nmap 7.60 (https://nmap.org) at 2020-04-15 19:27 CEST', 'Note: Host seems down. If it is really up, but blocking our ping probes, try -Pn', 'Nmap done: 1 IP address (0 hosts up) scanned in 3.06 seconds', and the prompt '(base) frank@fa-zenbook:~\$' followed by a green cursor.

```
Starting Nmap 7.60 ( https://nmap.org ) at 2020-04-15 19:27 CEST
Note: Host seems down. If it is really up, but blocking our ping probes, try -Pn
Nmap done: 1 IP address (0 hosts up) scanned in 3.06 seconds
(base) frank@fa-zenbook:~$
```

A straight forward scan did not yield any results, as our server does not respond on `ping-probes` command.

`nmap -Pn 46.101.215.40`

```
frank: bash
File Edit View Bookmarks Settings Help
(base) frank@fa-zenbook:~$
(base) frank@fa-zenbook:~$
(base) frank@fa-zenbook:~$ nmap -Pn 46.101.215.40

Starting Nmap 7.60 ( https://nmap.org ) at 2020-04-15 19:19 CE
Nmap scan report for 46.101.215.40
Host is up (0.027s latency).
Not shown: 993 filtered ports
PORT      STATE SERVICE
22/tcp    open  ssh
3000/tcp   open  ppp
5000/tcp   open  upnp
5001/tcp   open  complex-link
5100/tcp   closed admd
9090/tcp   open  zeus-admin
9200/tcp   open  wap-wsp

Nmap done: 1 IP address (1 host up) scanned in 6.24 seconds
(base) frank@fa-zenbook:~$
```

This gave us a list of expected accessible ports, namely 22 for our ssh management, this could be hidden on some other port, but hiding port is not the same as making it secure

3000 is Grafana for monitoring

5000 is our Minitwit UI

5001 is our Minitwit Api

5100 is closed, but was at a point used for Kibana

9090 is the Prometheus web

9200 is ElasticSearch for logdata

This is as expected based on the services we want to make public accessible and according to our firewall rules.

```
root@minitwit-group-noname-server:~# ufw status
Status: active

To Action From
--
22/tcp LIMIT Anywhere
2375/tcp ALLOW Anywhere
2376/tcp ALLOW Anywhere
5000 ALLOW Anywhere
5001 ALLOW Anywhere
22/tcp (v6) LIMIT Anywhere (v6)
2375/tcp (v6) ALLOW Anywhere (v6)
2376/tcp (v6) ALLOW Anywhere (v6)
5000 (v6) ALLOW Anywhere (v6)
5001 (v6) ALLOW Anywhere (v6)
root@minitwit-group-noname-server:~#
```

The two remaining ports 2375 and 2376 are for docker.

3.4 Vulnerabilities and potential fix

One issue is the amount of ports we have open to the public. By exposing multiple ports to the public we expose multiple pathways for potential hackers to gain access to our system, by exploiting a vulnerability of one of the services.

A way to fix this issue is to either:

- 1: Create a whitelist of IP addresses authorized to access these ports via the internet (not the minitwit app itself, which should be accessible to all).
- 2: Limit the amount of open ports to the public and making the port only accessible by people who can establish an SSH connection to the server.

Either way this can limit the exposure to the public.

The simulator that runs queries at our system, currently uses a simple authorization header with a password hardcoded into the source code. This is bad practice, and the system could be hardened by excluding the password from the source code, and include a whitelist to the simulator that runs queries on our site.

Another thing we tried was to use simple SQL attacks. However, these attacks were not possible due to the fact that we use an ORM that prevents regular SQL injection attacks. We do not sanitize the input in any way, meaning there might be a vulnerability somewhere.

4 Test the Security of Your Monitored Team

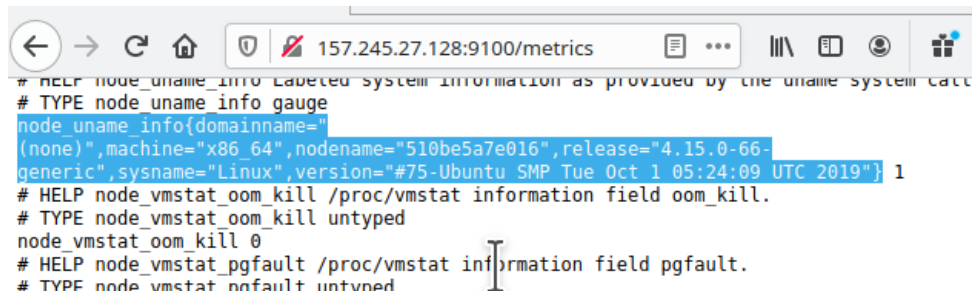
4.1 Putting on the white hat, pen testing group K

As with the testing of the security of our own system, we began with a basic port scan using nmap.

```
Starting Nmap 7.60 ( https://nmap.org ) at 2020-04-15 20:08 CEST
Nmap scan report for 157.245.27.128
Host is up (0.028s latency).
Not shown: 993 closed ports
PORT      STATE SERVICE
22/tcp    open  ssh
3000/tcp   open  ppp
3001/tcp   open  nessus
5001/tcp   open  complex-link
5002/tcp   open  rfe
9090/tcp   open  zeus-admin
9100/tcp   open  jetdirect

Nmap done: 1 IP address (1 host up) scanned in 0.41 seconds
(base) frank@fa-zenbook:~$ nmap -Pn 46.101.215.40
```

A risk is found on port 9100, where we are able to gain somewhat secret information about their setup, including memory and file-system information, through their metrics logs.



```
# HELP node_uname_info Labeled system information as provided by the uname system call
# TYPE node_uname_info gauge
node_uname_info{domainname="(none)",machine="x86_64",nodename="510be5a7e016",release="4.15.0-66-generic",sysname="Linux",version="#75-Ubuntu SMP Tue Oct 1 05:24:09 UTC 2019"} 1
# HELP node_vmstat_oom_kill /proc/vmstat information field oom_kill.
# TYPE node_vmstat_oom_kill untyped
node_vmstat_oom_kill 0
# HELP node_vmstat_pgfault /proc/vmstat information field pgfault.
# TYPE node_vmstat_pgfault untyped
```

The information exposed tells us something about the architecture, OS type and version, things an experienced hacker could find useful.

Poking around in the git repository of group k, we found postgres database credentials in the docker-compose.yaml file.

This might not be useful from the outside, but if we get in we now have easy access.

```
https://github.com/DevOps-Drengene/DevOps-Project/blob/master/docker-compose.yml

8      db:
9          image: postgres
10         networks:
11             - main
12         expose:
13             - "5432"
14         # Upon startup, it will create a Postgress user 'docker
15         # and a database called 'minitwit'.
16         environment:
17             - POSTGRES_USER=docker
18             - POSTGRES_PASSWORD=docker
19             - POSTGRES_DB=minitwit
20
21     simulator-api:
22         build: ./backend
23         networks:
24             - main
25         ports:
26             - "5001:5001"
27         command: bash -c "/wait && npm run simulator"
28         depends_on:
29             - db
30             - filebeat
31         environment:
32             - PSQL_DB_NAME=minitwit # Must be the same as 'POSTGRES_DB'
33             - PSQL_DB_USER_NAME=docker # Must be the same as 'POSTGRES_USER'
34             - PSQL_DB_USER_PASSWORD=docker # Must be the same as 'POSTGRES_PASSWORD'
35             - PSQL_HOST_NAME=db # Must be the same as the name of the db service
36             - WAIT_HOSTS=db:5432
37
```

<https://github.com/DevOps-Drengene/DevOps-Project/blob/master/docker-compose.yml>

Grafana Credentials found, this gives us admin access to Grafana UI.

```
25     grafana:
26         image: dagandersen/grafana
27         environment:
28             - GF_SECURITY_ADMIN_USER=Helge&friends
29             - GF_SECURITY_ADMIN_PASSWORD=uiUIui # Super safe
```

<https://github.com/DevOps-Drengene/DevOps-Project/blob/master/docker-compose.prod.yml>

Besides the information found on their port 9100, we did not find any other vulnerabilities we could exploit.

4.2 Searching for attacks on our system

We were not able to find any signs of attacks on our web system directly. We have looked at our log and monitoring data. We have only sent a very small number of 401 response codes and there is nothing unusual in the amount and type of requests and responses.






To see whether our host system have been under attack we check our authentication log files found in `/var/log/auth.log`

As it can be seen using the `grep` command, we are receiving a lot of unwelcome login attempts from a wide range of IP's with different credentials

`grep -m 10 "Invalid" /var/log/auth.log`

```
root@minitwit-group-noname-server:~# grep -m 10 "Invalid" /var/log/auth.log
Apr 12 06:44:44 minitwit-group-noname-server sshd[14885]: Invalid user backuppc from 90.89.38.205 port 47247
Apr 12 07:15:39 minitwit-group-noname-server sshd[15005]: Invalid user pi from 82.65.11.173 port 43722
Apr 12 07:18:09 minitwit-group-noname-server sshd[15018]: Invalid user testuser from 109.115.187.31 port 4286
Apr 12 07:22:17 minitwit-group-noname-server sshd[15036]: Invalid user chiuan from 106.12.5.190 port 57264
Apr 12 07:48:21 minitwit-group-noname-server sshd[15144]: Invalid user upload from 89.91.209.87 port 43327
Apr 12 07:59:05 minitwit-group-noname-server sshd[15185]: Invalid user gabriel from 134.122.121.110 port 3611
Apr 12 08:03:57 minitwit-group-noname-server sshd[15206]: Invalid user admin from 101.89.201.250 port 33806
Apr 12 08:19:58 minitwit-group-noname-server sshd[15381]: Invalid user ghost from 92.239.176.230 port 38734
Apr 12 08:43:36 minitwit-group-noname-server sshd[15468]: Invalid user terri from 188.226.167.212 port 33792
Apr 12 08:50:22 minitwit-group-noname-server sshd[15579]: Invalid user shizoom from 54.233.72.136 port 51488
root@minitwit-group-noname-server:~# grep -m 10 "Invalid" /var/log/auth.log.1
Apr 5 06:30:23 minitwit-group-noname-server sshd[2052]: Invalid user chat from 160.16.226.158 port 38710
Apr 5 06:35:12 minitwit-group-noname-server sshd[2072]: Invalid user chat from 160.16.226.158 port 54442
Apr 5 06:39:57 minitwit-group-noname-server sshd[2092]: Invalid user site from 160.16.226.158 port 41856
Apr 5 06:44:43 minitwit-group-noname-server sshd[2110]: Invalid user site from 160.16.226.158 port 57408
Apr 5 06:45:20 minitwit-group-noname-server sshd[2116]: Invalid user oracle from 150.95.115.145 port 50180
Apr 5 06:47:43 minitwit-group-noname-server sshd[2136]: Invalid user john from 150.95.115.145 port 43418
Apr 5 06:49:32 minitwit-group-noname-server sshd[2147]: Invalid user opyu from 160.16.226.158 port 44732
Apr 5 06:50:02 minitwit-group-noname-server sshd[2150]: Invalid user postgres from 150.95.115.145 port 36628
Apr 5 06:54:19 minitwit-group-noname-server sshd[2165]: Invalid user pico from 160.16.226.158 port 60380
Apr 5 06:55:08 minitwit-group-noname-server sshd[2171]: Invalid user castis from 157.230.188.53 port 44532
root@minitwit-group-noname-server:~#
```

Extracting the invalid data logs and making a count of attempt's by IP, we see that one in particular have been hitting us hard with over 400 invalid attempts.

<div>  <div>Session9_aut_log_invalid</div> <div>☆</div> <div></div> </div> <div> Fil Rediger Se Indsæt Formatér Data Værktøjer Tilføj </div>																																																																										
<div> <div>     </div> <div> 100% kr % .0 .00 123 Standard (...) </div> </div>																																																																										
<div> <div>fx</div> <table> <tr> <th></th><th>A</th><th>B</th><th>C</th><th>D</th></tr> <tr> <td>1</td><td>Pos.</td><td>IP</td><td>Count invalid logs pr.</td><td></td></tr> <tr> <td>2</td><td>1045</td><td>35.229.179.45</td><td>407</td><td></td></tr> <tr> <td>3</td><td>1251</td><td>52.174.50.120</td><td>131</td><td></td></tr> <tr> <td>4</td><td>565</td><td>157.92.24.249</td><td>121</td><td></td></tr> <tr> <td>5</td><td>1112</td><td>46.101.136.110</td><td>114</td><td></td></tr> <tr> <td>6</td><td>1312</td><td>62.171.142.113</td><td>105</td><td></td></tr> <tr> <td>7</td><td>1316</td><td>62.210.73.82</td><td>88</td><td></td></tr> <tr> <td>8</td><td>644</td><td>173.212.202.169</td><td>78</td><td></td></tr> <tr> <td>9</td><td>66</td><td>104.200.134.250</td><td>46</td><td></td></tr> <tr> <td>10</td><td>1467</td><td>92.63.194.47</td><td>36</td><td></td></tr> <tr> <td>11</td><td>1470</td><td>92.63.194.91</td><td>36</td><td></td></tr> <tr> <td>12</td><td>1472</td><td>92.63.194.93</td><td>36</td><td></td></tr> <tr> <td>13</td><td>1473</td><td>92.63.194.94</td><td>36</td><td></td></tr> </table> </div>						A	B	C	D	1	Pos.	IP	Count invalid logs pr.		2	1045	35.229.179.45	407		3	1251	52.174.50.120	131		4	565	157.92.24.249	121		5	1112	46.101.136.110	114		6	1312	62.171.142.113	105		7	1316	62.210.73.82	88		8	644	173.212.202.169	78		9	66	104.200.134.250	46		10	1467	92.63.194.47	36		11	1470	92.63.194.91	36		12	1472	92.63.194.93	36		13	1473	92.63.194.94	36	
	A	B	C	D																																																																						
1	Pos.	IP	Count invalid logs pr.																																																																							
2	1045	35.229.179.45	407																																																																							
3	1251	52.174.50.120	131																																																																							
4	565	157.92.24.249	121																																																																							
5	1112	46.101.136.110	114																																																																							
6	1312	62.171.142.113	105																																																																							
7	1316	62.210.73.82	88																																																																							
8	644	173.212.202.169	78																																																																							
9	66	104.200.134.250	46																																																																							
10	1467	92.63.194.47	36																																																																							
11	1470	92.63.194.91	36																																																																							
12	1472	92.63.194.93	36																																																																							
13	1473	92.63.194.94	36																																																																							

This indicates that we should most likely setup a prevention system, banning / blacklisting IP's after some amount of failed attempts. A way to maybe see if a breach has been successful is to run through the entire auth logs matching the IP's with multiple invalid attempts with a successful one.

Another way to spot intruders is that we could match a selected portion of our file-system with a initial backup, comparing hash values of each files to the "original" searching for any alterations.

5 Appendices

5.1 A — monitor.py

```
import json
import base64
import time
import requests
from colorama import Fore, Back, Style
from datetime import datetime

# print(Fore.RED + 'some red text')
# print(Back.GREEN + 'and with a green background')
# print(Style.DIM + 'and in dim text')
# print(Style.RESET_ALL)
# print('back to normal now')
# today = datetime.date.today()

BASE_URL = 'http://157.245.27.128:5001'
USERNAME = 'simulator'
PWD = 'super_safe!'
CREDENTIALS = ':'.join([USERNAME, PWD]).encode('ascii')
ENCODED_CREDENTIALS = base64.b64encode(CREDENTIALS).decode()
HEADERS = {'Connection': 'close',
           'Content-Type': 'application/json',
           f'Authorization': f'Basic_{ENCODED_CREDENTIALS}'}

LATEST_TIME = 0
LATEST_REQUESTS = 0
POST_MSGS_TIME = 0
POST_MSGS_REQUESTS = 0
GET_MSGS_TIME = 0
GET_MSGS_REQUESTS = 0
GET_USER_MSGS_TIME = 0
GET_USER_MSGS_REQUESTS = 0
GET_FLLWS_TIME = 0
GET_FLLWS_REQUESTS = 0
POST_FLLWS_TIME = 0
POST_FLLWS_REQUESTS = 0

FAULTS = 0

def sendGetRequest(endpoint):
    global FAULTS
    url = f'{BASE_URL}/{endpoint}'
    response = requests.get(url, headers=HEADERS)
    if not response.ok:
        FAULTS += 1

def testEndpoint(endpoint, time_counter, request_counter, sla_req,
getRequest=True, data=None):
    start = time.time()
    if getRequest:
        sendGetRequest(endpoint)
    else:
        sendPostRequest(endpoint, data)
    total = time.time() - start
```

```

time_counter += total
request_counter += 1

time_avg = time_counter / request_counter

print(datetime.now().strftime("[%d/%m/%Y %H:%M:%S]"), end='_')

if (time_avg > sla_req):
    print("/%-*s_time: %.3gs \t t_avg: %.3gs \t \t ALERT" % (15,
        endpoint, total, time_avg))
else:
    print("/%-*s_time: %.3gs \t t_avg: %.3gs" % (15, endpoint,
        total, time_avg))

return time_counter, request_counter

def sendPostRequest(endpoint, data):
    global FAULTS
    url = f"{BASE_URL}/{endpoint}"
    response = requests.post(url, data=json.dumps(data), headers=
        HEADERS)
    if not response.ok:
        FAULTS += 1

#Creating users used for monitoring
username = 'TestUser'
data = {'username': username, 'email': f'{username}@test', 'pwd': '
    foo'}
sendPostRequest('register', data)

username2 = 'TestUser2'
data = {'username': username2, 'email': f'{username2}@test', 'pwd': '
    foo'}
sendPostRequest('register', data)

count = 0

while (True):
    count += 1

    #GET /latest
    LATEST_TIME, LATEST_REQUESTS = testEndpoint('latest',
        LATEST_TIME, LATEST_REQUESTS, 0.1)

    #GET /msgs
    GET_MSGS_TIME, GET_MSGS_REQUESTS = testEndpoint('msgs',
        GET_MSGS_TIME, GET_MSGS_REQUESTS, 5)

    #GET /flws
    GET_FLLWS_TIME, GET_FLLWS_REQUESTS = testEndpoint(f'flws/{
        username}', GET_FLLWS_TIME, GET_FLLWS_REQUESTS, 0.1)

    #GET /msgs/user
    GET_USER_MSGS_TIME, GET_USER_MSGS_REQUESTS = testEndpoint(f'
        msgs/{username}', GET_USER_MSGS_TIME,
        GET_USER_MSGS_REQUESTS, 0.7)

```

```

# POST /msgs/user
data = {'content': 'test_message123'}
POST_MSGS.TIME, POST_MSGS.REQUESTS = testEndpoint(f'msgs/{
    username}', POST_MSGS.TIME, POST_MSGS.REQUESTS, 0.1, False,
    data)

#POST /fllws/user
data = {'follow': f'{username2}'}
POST_FLLWS.TIME, POST_FLLWS.REQUESTS = testEndpoint(f'fllws/{
    username}', POST_FLLWS.TIME, POST_FLLWS.REQUESTS, 0.15,
    False, data)

print(f'No_response_errors:{FAULTS}/{count*6}')

time.sleep(60)

```

5.2 B — clean.py

Script that cleans up the log.

```

#!/usr/bin/env bash

## Print meaningful stats from monitor.log

set -eo pipefail

# echo "Probes pr minute: 1"
# echo "Request pr minute: 6"

# echo -n "Total no probes against server: "
# echo $(awk '/error/{print}' monitor.log | wc -l)

# echo -n "Total no requests "
# echo $(awk '/error/{print}' monitor.log | awk '{print $6}' | tail
    -1)

# echo "Average end point response times: "
# awk '{print $3 " " $7}' monitor.log | tail -6 | column -t -s ' '

# echo -n "Slowest end-point response: "
# echo $(awk -v max=0
    '{if($5>max){want=$1"$2"$3"$4"$5;_max=$5}}
    _____END{print _want}' monitor.log)

# echo -n "Fastest end-point response: "
# echo $(cat monitor.log | grep -v 'No response errors' \
    | awk -v min=666 '{if($5<min){want=$1"$2"$3"$4"$5;_min=
        $5}}END{print _want}')

echo "Creating_monitor.csv_from_logged_data..."
echo "date,datetime,endpoint,resp-time,avg_resp-time" > monitor
    .csv
echo ""

awk '
    _____{date=$1;datetime=$2}
    _____{gsub("\[",",",date);_gsub("\]",",",datetime)}
    _____{endpoint=$3}

```

```
#####{print_date",,"datetime_",,"_endpoint_",,"_$5_",,"_$7}  
#####' monitor.log | grep -v 'errors' >> monitor.csv
```