
 Queensland University of Technology	QUT Systems Engineering UAVPayload^{TAQ}-G2	Doc No: TR-UAVPayload ^{TAQ} -G2-WEB-01 Issue: 1 Page: 1 of 22 Date: 19 October 2020
--	--	---

UAVPayloadTAQ

Subsystem Unit Testing – Web Visualisation

Project: UAVPayloadTAQ-G2 WP Name: Subsystem Integration Testing WP Number: WP-WEB-5	Type of Test: Unit Test	
Test Article: Web Application	Part Number: N/A	Serial Number: N/A
System Requirements: REQ-M-3, REQ-M-4, REQ-M-5, REQ-M-6, REQ-M-7, REQ-M-12, REQ-M-15	Test Equipment: Lenovo X1 Carbon, Macbook Pro, Raspberry Pi 3B+	
Test Operators: Brian Sivertsen	Test Engineers: Brian Sivertsen	
Project Manager: Alexander Iftene	Project Supervisor: Dr Felipe Gonzalez	

 Queensland University of Technology	QUT Systems Engineering UAVPayload^{TAQ}-G2	Doc No: TR-UAVPayload ^{TAQ} -G2-WEB-01 Issue: 1 Page: 2 of 22 Date: 19 October 2020
--	--	---

Revision Record

Document Issue/Revision Status	Description of Change	Date	Approved
1.0	Initial Document	9 October 2020	Brian Sivertsen
1.1	Adding Post Integration Tests	19 October 2020	Brian Sivertsen

Table of Contents

Paragraph	Page No.
1 Introduction	7
1.1 Scope	7
1.2 Background	7
2 Reference Documents.....	8
2.1 QUT Avionics Documents	8
2.2 Non-QUT Documents	8
3 Test Objectives	9
4 Test Setup and Equipment.....	10
4.1 Pre-Integration Testing Setup and Equipment	10
4.2 Post-Integration Testing Setup and Equipment.....	10
5 Test Procedures	11
5.1 Pre-Integration Test Procedures	11
Test 1: REQ-M-7	11
Test 2: REQ-M-4.....	11
5.2 Post-Integration Test Procedures.....	12
Test 3: REQ-M-3.....	12
Test 4: REQ-M-5	12
Test 5: REQ-M-6.....	12
Test 6: REQ-M-15	12
6 Test Results	13
6.1 Pre-Integration Test Results	13
Test 1: REQ-M-7	13
Test 2: REQ-M-4.....	15
6.2 Post-Integration Test Results.....	15
Test 3: REQ-M-3.....	15
Test 4: REQ-M-5	17
Test 5: REQ-M-6.....	17


Test 6: REQ-M-15.....	17
7 Conclusion.....	18
Through the test procedures outlined in section 5 and results in section 6 it is clear that WEB subsystem is capable of satisfying all of the relevant requirements. The requirements taken from RD/2 include various image and data output challenges that the WEB subsystem has proven to satisfy.	18
8 Appendices	19
8.1 Home Page	19
8.2 Sensor Data JSON object	20
8.3 Flask Server.....	20
8.4 Front End API call.....	21
8.5 Front End Image Render	21
8.6 Test Flask Server	22

List of Figures

Figure	Page No.
Figure 3: Home Page	19

Definitions

UAV	Unmanned Aerial Vehicle
UAVPayload	Unmanned Aerial Vehicle Payload
GCS	Ground Control Station
WEB	Web Visualization Subsystem
FTP	File Transfer Protocol
HLO	High Level Objective
SR	System Requirements
POC	Proof of Concept
LAN	Local Area Network
TBD	To Be Determined
API	Application Programming Interface

 Queensland University of Technology	QUT Systems Engineering UAVPayload^{TAQ}-G2	Doc No: TR-UAVPayload ^{TAQ} -G2-WEB-01 Issue: 1 Page: 7 of 22 Date: 19 October 2020
--	--	---

1 Introduction

In order to ensure proper intended functionality for each subsystem it is important to conduct adequate testing to ensure they each operate correctly prior to integration.

1.1 Scope

This document outlines the Unit testing for the WEB subsystem. Testing is conducted using a Raspberry Pi serving dummy data to a Windows computer which is outputting the web server that is to be accessed locally by other devices on the network.

1.2 Background

The Queensland University of Technology (QUT) Airborne Sensing Lab have appointed Group 2 of the EGH455 (Advanced Systems Design) class to design a UAV Payload for indoor air quality to be installed on a S500 UAV designed for navigating in GPS denied environments. The UAVPayloadTAQ is required to conduct constant air quality sampling in a simulated underground mine. During monitoring, it must find and identify multiple markers placed by miners around the mine. Additionally, QUT Airborne Sensing Systems requires that the UAVPayloadTAQ is designed and developed using Systems Engineering to ensure QUT Airborne Sensing Systems requirements are met. Taken from RD/3.

2 Reference Documents

2.1 QUT Avionics Documents

RD/1	CN-UAVPayloadTAQ-01	UAV Payload Customer Needs
RD/2	SR-UAVPayloadTAQ-01	UAV Payload System Requirements
RD/3	PMP-UAVPayloadTAQ-01	UAV Payload Project Management Plan

2.2 Non-QUT Documents


RD/8	https://flask.palletsprojects.com/en/1.1.x/	Flask User Guide
RD/9	https://expressjs.com/	ExpressJS Documentation
RD/10	https://reactjs.org/	React Documentation
RD/11	https://chakra-ui.com/getting-started	Chakra UI Documentation
RD/12	https://github.com/jerairrest/react-chartjs-2	Chart.js React Integration

3 Test Objectives

Unit testing has been conducted on the WEB subsystem to ensure complete satisfaction of the relevant system requirements. From RD/2 the relevant system requirements for the WEB sub system are:

- **REQ-M-3:** The UAVPayload^{TAQ} shall communicate with a ground station computer to transmit video, target detection and air quality data.
- **REQ-M-4:** The target identification system shall be capable of alerting the GCS of a targets type.
- **REQ-M-5:** The Web Interface is required to display real time air sampling data that is recorded directly from the UAVPayload^{TAQ} and updated dynamically throughout the duration of the flight.
- **REQ-M-6:** The Web Interface is required to display the images of the targets that are taken directly from the UAVPayload^{TAQ} and updated every time a new picture is taken.
- **REQ-M-7:** The Web Interface shall be designed and run as a web server, which is to be accessible by any computers on the local network.
- **REQ-M-15:** Live data from the UAV must be made available through the web server within 10 seconds of capture.

Through the testing below these requirements are showed to be verified.

 Queensland University of Technology	QUT Systems Engineering UAVPayload ^{TAQ} -G2	Doc No: TR-UAVPayload ^{TAQ} -G2-WEB-01 Issue: 1 Page: 10 of 22 Date: 19 October 2020
--	---	--

4 Test Setup and Equipment

Testing for the WEB subsystem was completed over 2 sessions as some of the requirements can be verified prior to integration, such as REQ-M-4 and REQ-M-7 while the others, REQ-M-3, REQ-M-5, REQ-M-6 and REQ-M-15 can only be accurately tested after the WEB subsystem has been integrated with the other subsystems.

4.1 Pre-Integration Testing Setup and Equipment

The Pre-Integration test will use a test Flask server served from the same laptop that is hosting the final web server, the test server will serve a static image and random valued for the sensor data and detection ids. The script for the test server can be seen in appendix 6. the web server laptop will also run the React application and display any test data served to it from the test server.

The Equipment used in this initial testing are:

1. Lenovo Thinkpad X1 Carbon: This computer is used as the GCS and will host the webserver in the final demonstration
2. TP-LINK Archer A10 WiFi Router: This is a generic router just used to test the ability to serve the application over the LAN, can be swapped out with any router
3. Macbook Pro: This laptop is just used to test serving the application over LAN, any Wi-Fi compatible device could be used instead, even a mobile phone.

4.2 Post-Integration Testing Setup and Equipment

The Post-Integration test will serve the final Flask server from a raspberry Pi with both the Image Processing and Sensor Data subsystems running. This will provide verification for requirements that are also reliant on successful output and completion of these subsystems. The final Flask server script is attached in Appendix 6.

This set of tests included the same equipment listed above as well as:

1. Raspberry Pi 3B+: This is the micro controller which is used to process and serve the information
2. Raspberry Pi Camera Module v2: This is the camera used in the final demo to take the input images for the Image Processing subsystem
3. Pimoroni Enviro+ Sensor Hat: this sensor hat is used in the final demo to gain various pieces of info in the surrounding area.

5 Test Procedures

As stated above, the overall testing was conducted over 2 sessions, this was to make sure that all the requirements could be properly verified. Testing of REQ-M-4 and REQ-M-7 was able to be verified in the pre-integration test as they were not reliant on the other subsystems. However, REQ-M-3, REQ-M-5, REQ-M-6 and REQ-M-15 could only be accurately tested after the WEB subsystem has been integrated with the other subsystems and a post-integration test would have to be conducted.

5.1 Pre-Integration Test Procedures

Test 1: REQ-M-7

REQ-M-7 states that: The Web Interface shall be designed and run as a web server, which is to be accessible by any computers on the local network. In order to test and verify this requirement, several steps are required:

1. Run test Flask server (Appendix x)
2. Run React web server
3. Verify React server is served on Local Host
4. Verify separate device can access web server over LAN

Test 2: REQ-M-4

REQ-M-4 requires that: The target identification system shall be capable of alerting the GCS of a targets type. This can be validated by running the prior steps in test 1 and validating that the web application outputs the various audio outputs for each detection type:

Detection	Audio Output
None	No output
Corrosive	Corrosive Output
Dangerous Goods	Dangerous Goods
Aruco Marker	Aruco

5.2 Post-Integration Test Procedures

Test 3: REQ-M-3

REQ-M-3 has the requirement: The UAVPayload^{TAQ} shall communicate with a ground station computer to transmit video, target detection and air quality data. A few small tests need to succeed for this requirement to be satisfied, these include:

1. Run final webserver on Raspberry Pi (appendix x)
2. Run Image Detection on Raspberry Pi
3. Verify Image output on GCS computer
4. Verify sensor data and detection Id output on GCS computer

Test 4: REQ-M-5

REQ-M-5 states that: The Web Interface is required to display real time air sampling data that is recorded directly from the UAVPayload^{TAQ} and updated dynamically throughout the duration of the flight. To satisfy this requirement only 2 small tests are needed:

1. Run React Application on GCS
2. Verify sensor data is populating and updating

Test 5: REQ-M-6


To satisfy REQ-M-6: The Web Interface is required to display the images of the targets that are taken directly from the UAVPayload^{TAQ} and updated every time a new picture is taken. To verify this requirement all that needs to be done after the previous test is to:

1. Verify images are populating and updating

Test 6: REQ-M-15

For REQ-M-15: Live data from the UAV must be made available through the web server within 10 seconds of capture. All that needs to be done after the previous tests is to:

1. Verify sensor data and images are taking less than 10 seconds to populate

 Queensland University of Technology	QUT Systems Engineering UAVPayload^{TAQ}-G2	Doc No: TR-UAVPayload ^{TAQ} -G2-WEB-01 Issue: 1 Page: 13 of 22 Date: 19 October 2020
--	--	--

6 Test Results

These test results apply to the tests outlined above in section 5

6.1 Pre-Integration Test Results

Test 1: REQ-M-7

1. Run test Flask server:

```
brian@DESKTOP-11DSJEN MINGW64 /c/2020/SEM2/EGH455/flask-and-react/react-flask-app (master)
$ yarn start-api
yarn run v1.19.1
$ cd api && flask run --no-debugger
* Serving Flask app "api.py" (lazy loading)
* Environment: development
* Debug mode: on
* Restarting with stat
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

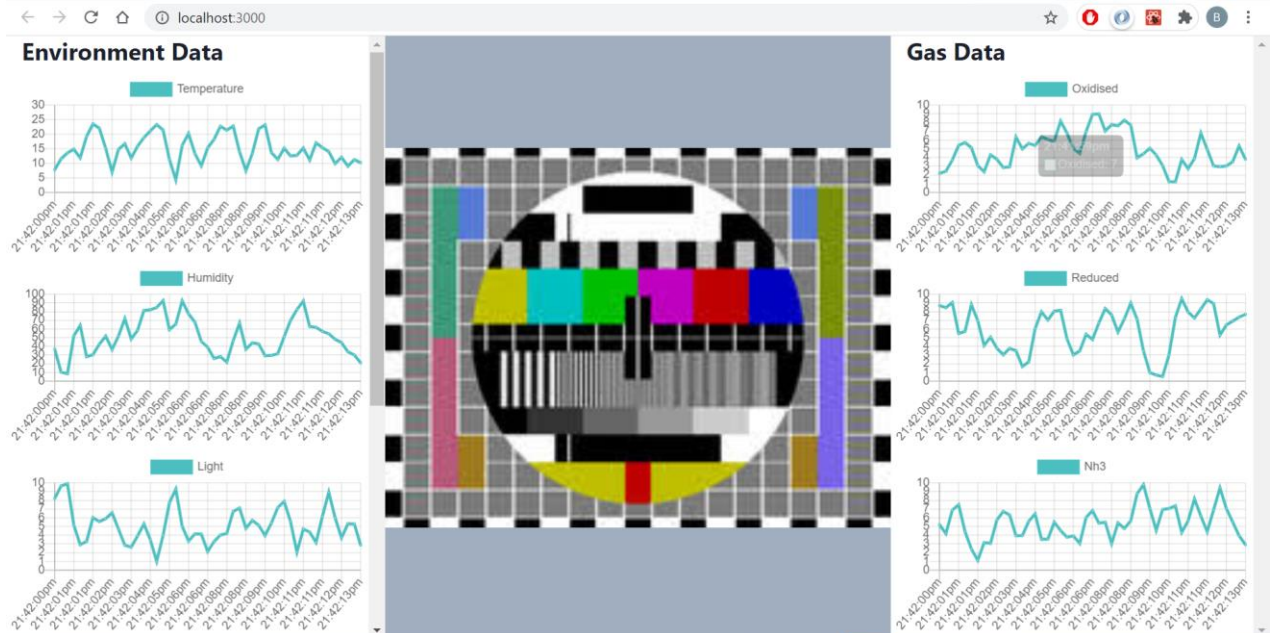
yarn start-api is called and the test server starts running on <http://127.0.0.1:5000/> or <http://localhost:5000/>.

2. Run React web server

```
brian@DESKTOP-11DSJEN MINGW64 /c/2020/SEM2/EGH455/flask-and-react/react-flask-app (master)
$ yarn start
yarn run v1.19.1
$ react-scripts start
i [wds]: Project is running at http://0.0.0.0:3000/
i [wds]: webpack output is served from
i [wds]: Content not from webpack is served from C:\2020\SEM2\EGH455\flask-and-react\react-flask-app\public
i [wds]: 404s will fallback to /
Starting the development server...
Compiled with warnings.
```

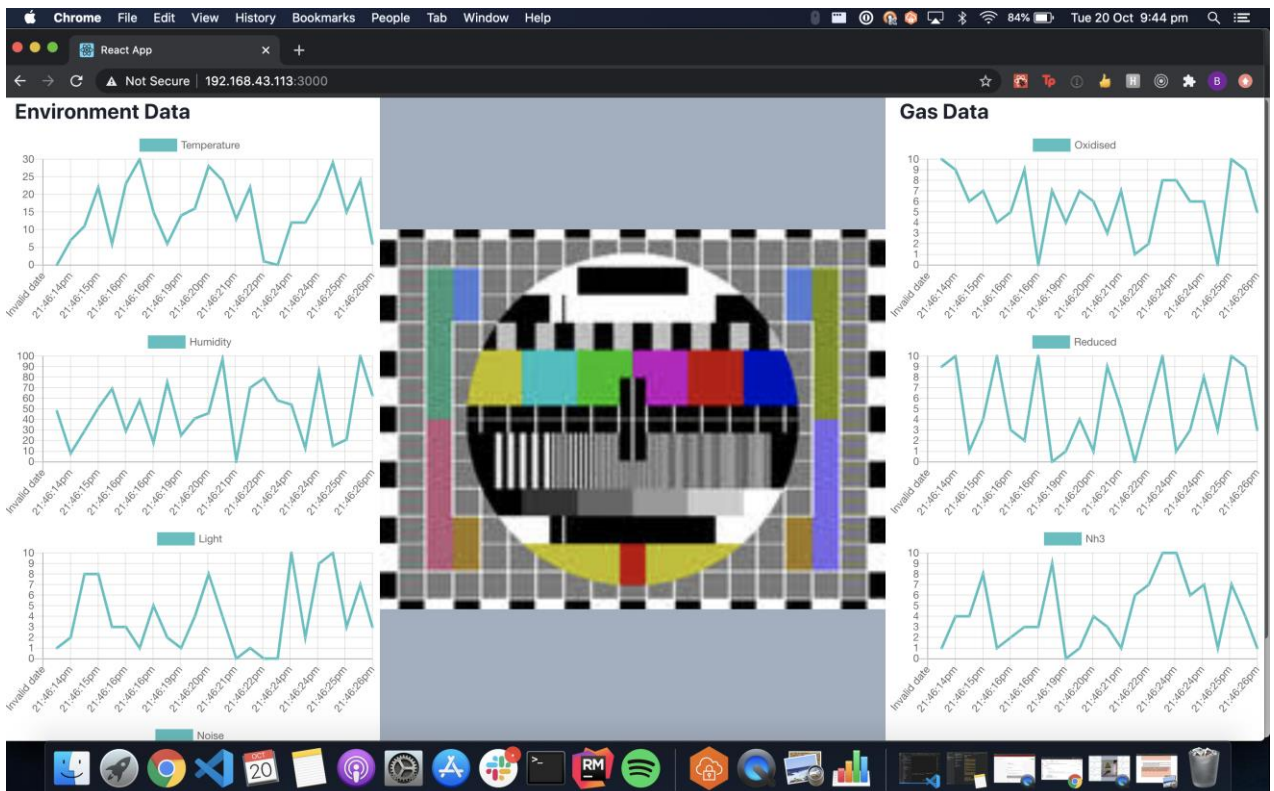
yarn start is called and the webserver is running on <http://0.0.0.0:3000/> or <http://localhost:3000/>.

3. Verify React server is served on Local Host



Navigating to <http://localhost:3000/> shows that the application is successfully running and getting served on Local Host

4. Verify separate device can access web server over LAN



Navigating to <http://192.168.43.113:3000/> on the Macbook Pro shows that the application is successfully running on the GCS and can be served over LAN.

Test 2: REQ-M-4

By listening to the audio output of the MacBook Pro all the relevant audio outputs can be heard, as listed in the table below

Detection	Audio Output	Verified?
None	No output	Yes
Corrosive	Corrosive Output	Yes
Dangerous Goods	Dangerous Goods	Yes
Aruco Marker	Aruco	Yes

This verifies that REQ-M-4 has been satisfied.

6.2 Post-Integration Test Results

Test 3: REQ-M-3

1. Run final webserver on Raspberry Pi

```

pi@Group2:~ $ cd server
pi@Group2:~/server $ ls
api.py api.py.save api.py.save.1 api.py.save.2 __pycache__
pi@Group2:~/server $ export FLASK_APP=api.py
pi@Group2:~/server $ flask run --host=0.0.0.0
* Serving Flask app "api.py"
* Environment: production
  WARNING: Do not use the development server in a production environment.
  Use a production WSGI server instead.
* Debug mode: off
2020-10-20 13:01:03.321 INFO      DataSensing455.py - Displays readings from all
of Enviro plus' sensors
Press Ctrl+C to exit!

2020-10-20 13:01:03.589 INFO      * Running on http://0.0.0.0:5000/ (Press CTRL+
C to quit)

```

By navigating to the *server* directory, setting the `FLASK_APP` variable to `api.py`, the name of the server file **Appendix x**, and calling `flask run --host=0.0.0.0` The Flask server is run on the Raspberry Pi's localhost.

2. Run Image Detection on Raspberry Pi

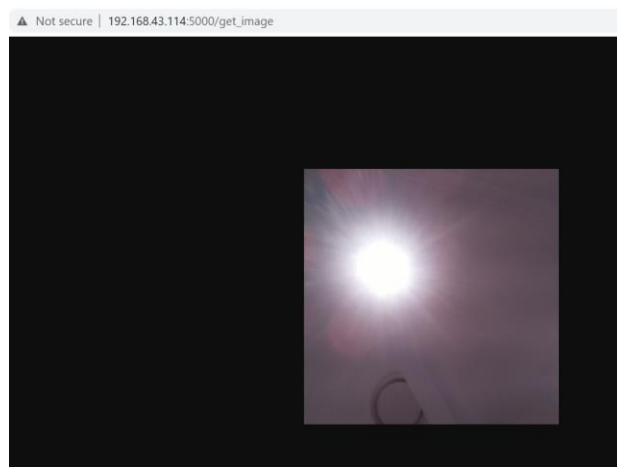
```

pi@Group2:~ $ cd pythonscripts
pi@Group2:~/pythonscripts $ python Projecttest.py
imports loaded
beginning iteration
()
()
targ_id at end of while: 0
0
no classifiers detected
beginning iteration
()
()
targ_id at end of while: 0
0
no classifiers detected

```

Navigating to the *pythonscripts* directory and calling *python Projecttest.py* starts the Image Processing application, as can be seen there are no detections in the first 2 photo's taken by the camera.

3. Verify Image output on GCS computer



By navigating to http://192.168.43.114:5000/get_image (the IP of the Raspberry Pi and port of the server and route for image output) we can verify that the Raspberry Pi is outputting images to the GCS


4. Verify sensor data and detection Id output on GCS computer

```

{
  humidity: 31.15409702043967,
  id: "0",
  light: 151.569,
  nh3: 0.03799725668962668,
  noise: 20.71233749213445,
  oxidised: 0.13984436048032392,
  pressure: 1013.3606801926179,
  random: 17,
  reduced: 1.532726482042194,
  temperature: 4.516666210254726,
  time: 1603195515.031669
}

```

By navigating to <http://192.168.43.114:5000/data> (the IP of the Raspberry Pi and port of the

 Queensland University of Technology	QUT Systems Engineering UAVPayload^{TAQ}-G2	Doc No: TR-UAVPayload ^{TAQ} -G2-WEB-01 Issue: 1 Page: 17 of 22 Date: 19 October 2020
--	--	--

server and route for sensor data output) we can verify that the Raspberry Pi is outputting its sensor data and image detection id to the GCS.

Test 4: REQ-M-5

1. Run React Application on GCS

Same as Test 1 step 2.

2. Verify sensor data is populating and updating

https://drive.google.com/file/d/1nsv1o0OBgQNMR02Z_W8j3ZtDdRH7DGrN/view?usp=sharing

A screen recording of the GCS output has been taken and uploaded to the URL above, the sensor data is populating and updating via the Raspberry Pi sensor data.

Test 5: REQ-M-6

1. Verify images are populating and updating

https://drive.google.com/file/d/1nsv1o0OBgQNMR02Z_W8j3ZtDdRH7DGrN/view?usp=sharing


By viewing the same screen recording as above it is verified that the images are populating and updating.

Test 6: REQ-M-15

1. Verify sensor data and images are taking less than 10 seconds to populate

https://drive.google.com/file/d/1nsv1o0OBgQNMR02Z_W8j3ZtDdRH7DGrN/view?usp=sharing

By viewing the screen recording and seeing the time difference between events happening in the images (e.g. lights turning off) I can verify that the sensor data and image outputs were updated within 2 seconds of the event occurring, well within the needed 10 second delay.

 Queensland University of Technology	QUT Systems Engineering UAVPayload^{TAQ}-G2	Doc No: TR-UAVPayload ^{TAQ} -G2-WEB-01 Issue: 1 Page: 18 of 22 Date: 19 October 2020
--	--	--

7 Conclusion

Through the test procedures outlined in section 5 and results in section 6 it is clear that WEB subsystem is capable of satisfying all of the relevant requirements. The requirements taken from RD/2 include various image and data output challenges that the WEB subsystem has proven to satisfy.

8 Appendices

8.1 Home Page



Figure 1: Home Page



8.2 Sensor Data JSON object

```
data = {  
    humidity: 36.312575027605625,  
    id: "0",  
    light: 45.2082,  
    nh3: 0.023396810545597963,  
    noise: 15.996364231394265,  
    oxidised: 0.24942645135306174,  
    pressure: 1013.8826293833631,  
    reduced: 1.0039478116889995,  
    temperature: 6.132050825639343,  
    time: 1603106876.655459  
}
```

8.3 Flask Server

```
import time  
from flask import Flask, send_file  
from random import seed  
from random import randint  
from flask import jsonify  
import sys  
# insert at 1, 0 is the script path (or '' in REPL)  
sys.path.insert(1, '/home/pi/enviropius/examples')  
sys.path.insert(2, '/home/pi/pythonscripts')  
  
import dataSense  
  
app = Flask(__name__)  
  
@app.route('/data')  
def get_current_data():  
    id = open("/home/pi/pythonscripts/detectionFile.txt", "r")  
    d = dataSense.returnSensors()  
    d["random"] = randint(0, 100)  
    d["id"] = id.read()  
    return jsonify(d)  
  
@app.route('/get_image')  
def get_image():  
    return send_file('/home/pi/photos/output/outputimage.jpg')
```



8.4 Front End API call

```
useEffect(() => {  
  fetch('/data').then(res => res.json()).then(data => {  
    data.Time=Date.now();  
    setData(data);  
    let newGraphs = graphs;  
  
    if (graphs.length < 50)  
      newGraphs.push(data);  
    else {  
      newGraphs.shift()  
      newGraphs.push(data);  
    }  
  
    if (data.id == "1"){  
      let toxic = new Audio("./toxic.ogg")  
      toxic.play();  
    } else if ( data.id == "2"){  
      let dangerous = new Audio("./dangerous.ogg")  
      dangerous.play();  
    } else if ( data.id == "3"){  
      let aruco = new Audio("./aruco.ogg")  
      aruco.play();  
    }  
  
    setGraphs(newGraphs);  
  });  
});
```

8.5 Front End Image Render

```
<Image width="40%" height="100vh" src={`/${get_image?${data.Time}`} ` />
```



8.6 Test Flask Server

```
import time
from flask import Flask, send_file
from random import seed
from random import randint
# seed random number generator
seed(1)

app = Flask(__name__)

@app.route('/data')
def get_current_data():
    return {'time': time.time(),
            'temperature': randint(0, 30),
            'pressure': randint(0,10),
            'humidity': randint(0, 100),
            'light': randint(0, 10),
            'noise': randint(0, 10),
            'oxidised': randint(0, 10),
            'reduced': randint(0, 10),
            'nh3': randint(0, 10),
            'id': str(randint(0,3))}

@app.route('/get_image')
def get_image():
    return send_file('test.jpg')
```