Final Design UAVPayload^{TAQ} Web Visualization

Revision Record

Prepared by	R.B Ryan Brooker, Web Visualization lead	Date	28/10/2022
Checked by	A.G Alexander Gray, Project Manager	Date	28/10/2022
Approved by	M.B Marissa Bowen, Project Manager	Date	28/10/2022
Authorised for use by	______________________________ Dr. Felipe Gonzalez, Project Coordinator	Date	_______________

Queensland University of Technology
Gardens Point Campus
Brisbane, Australia, 4001.

Document Issue/Revision Status	Description of Change	Date	Approved
1.0	Initial	21/10/2022	R.B
2.0	Updates to Software Design and document formatting	28/10/2022	R.B

Table of Contents

List of Figures

List of Tables

Definitions

Table 1: Final document definitions

PMP	Project Management Plan
USB	Universal Serial Bus
AQS	Air Quality Sensor
TAIP	Target Acquisition and Image Processing
ED	Enclosure Design
ST	Sampling Tube
WVI	Web Visualization and Interface
QUT	Queensland University of Technology
GPS	Global Positioning System
ASP	Advance Sensor Payload
UAV	Unmanned Aerial Vehicle
HLO	High Level Objective

1 Introduction

This document is a final design of UAVPAYG19 for the subsystem of Web Visualisation and Interfaces (WVI). A final design report is an essential part of the System Engineering Design Approach as it provides an overview of the subsystem and supplies a seamless transition to developing the final solution. This document outlines the requirements provided in RD/1 and how these requirements are to be achieved. A Subsystem Architecture is shown which describes the interfaces this subsystem has with the others. The final design of the Web Interface with modelling images and shown with justifications to the design choices.

1.1 Scope

This final design document details information about the WVI subsystem and where it fits into the ASP for UAV. The WVI system architecture final design documents the operation, functionality, and details of each interface.

1.2 Background

The Queensland University of Technology's Airborne System Lab (ASL) has commissioned the group UAVPAYG19 to design and develop a payload capable in detecting specific objects, recording air quality data to be displayed on a web interface and to pierce a ground sample. This payload is to be attached to a S500 UAV which completed an automated flight path. The payload is mounted on the bottom of the UAV using a proved bracket. This payload must contain all components to complete its required tasks. These components are:

- Raspberry Pi 3b+
- Raspberry Pi Camera
- Pimoroni enviro sensor
- DF15RSMG 360 Degree Motor

The project is required to identify three targets, a valve (In open or closed position), a fire extinguisher and an ArUCO marker. The Pimorino sensor is to be used to record air temperature, pressure humidity, light and potentially hazardous gas level data. This data along with a live feed of the Raspberry Pi Camera is to be visualized on a Web Interface. Lastly a soil sample must be obtained using a sampling mechanism.

2 Reference Documents

2.1 QUT Avionics Documents

Table 2: QUT Avionics documents

RD/1	UA–System Requirements	UAVPayloadTAQ System Requirements
RD/2	UA –Customer Needs	Advanced Sensor Payload for UAV Target Detection and Air Quality Monitoring in GPS Denied Environments
RD/10	UAVPAYG19-WVI-PD-02	WVI Preliminary Design
RD/25	UAVPAYG-19-WVI-TR-02	Air Quality Sensor, Target Acquisition and Image Processing, Web Vision Interface Integration Report
RD/26	UAVPAYG-19-AQS-TAIP-WVI-TR-01	Enclosure, Air Quality Sensor, Target Acquisition and Image Processing, Sampling Tube Test Report
RD/27	UAVPAYG-19-ED-AQS-TAIP-WVI-ST-TR-01	Enclosure, Air Quality Sensor, Target Acquisition and Image Processing, Sampling Tube Integration Report
RD/31	UAVPAYG19-AQS-FD-02	AQS Final Design Report
RD/32	UAVPAYG19-TAIP-FD-02	TAIP Final Design Report
RD/33	UAVPAYG19-ST-FD-02	ST Final Design Report

3 Subsystem Introduction

The Web Visualization and Interfaces subsystem purpose is to collect data sent from the ASP to save in a database and output the data sent onto a Web Interface. The WVI subsystem will implement a Database, Web Interface, and Server. The Web Interface will display data sent from the ASP to the client within 10 seconds of capture. The data sent includes a live video stream and images from the TAIP subsystem (RD/32) and air quality data from the AQS subsystem (RD/31). The web interface is designed to show this data in a readable and easily accessible way whilst constantly displaying new data sent from the ASP in real time to meet the subsystem requirements (RD/31) . The Web Interface will have to vocalise targets detected by the TAIP system and give the users of the interface access to the logged data collected. Without the web interface, the project will not have met the project requirements.

3.1 Subsystem Requirement

Table 3: Subsystem requirements and design considerations

Requirements	Description	Verification
REQ-M-03	The UAVPayloadTAQ shall communicate with a ground station computer to transmit video, target detection and air quality data.	Demonstration
REQ-M-04	The target identification system shall be capable of alerting the GCS of a target's type.	Demonstration
REQ-M-05	The Web Interface is required to display real time air sampling data that is recorded directly from the UAVPayloadTAQ and updated dynamically throughout the duration of the flight.	Demonstration
REQ-M-06	The Web Interface is required to display the images of the targets that are taken directly from the UAVPayloadTAQ and updated every time a new picture is taken.	Demonstrated
REQ-M-07	The Web Interface shall be designed and run as a web server, which is to be accessible by any computers on the local network. This shall store logged sensor data and target detections with corresponding timestamps.	Demonstrated
REQ-M-10	Preliminary designs shall be completed by week 7	Submitted Document
REQ-M-15	The system shall have logged functioning operation for a minimal period of 10 minutes prior to acceptance test.	Demonstrated
REQ-M-19	Live data from the UAV must be made available through the web server within 10 seconds of capture.	Demonstrated

4 Subsystem Architecture

The WVI subsystem interfaces with the Raspberry Pi 3B+, however, there are no physical hardware components. System architecture diagram of the WVI subsystem can be seen displayed in the Figure 1with the WVI subsystem highlighted in the green box. The subsystem receives data from the Raspberry Pi which then is stored in a database and made available through a Web Interface that can be accessed using a web browser.

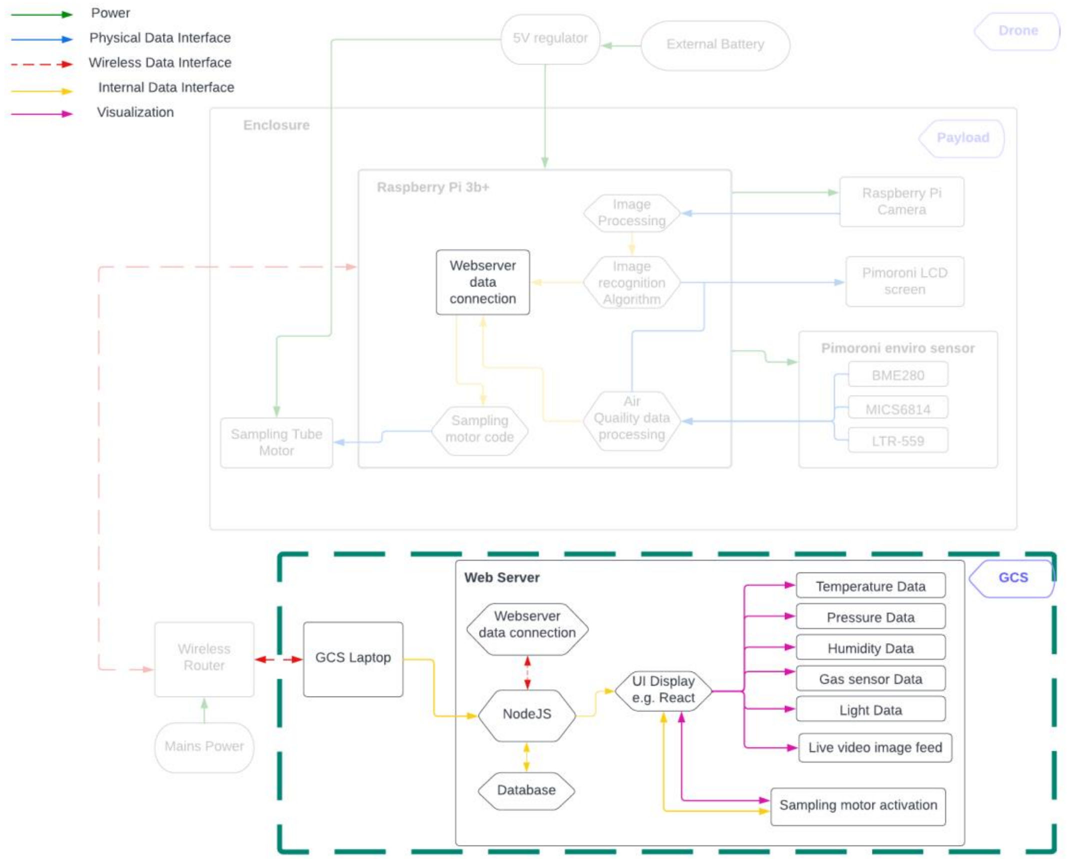


Figure 1:Web Visualization and Interface Subsystem Overview

4.1 Interfaces

For all the requirements to be met for the project the WVI will need to interface with multiple subsystems. Firstly, the Raspberry Pi sends a request to the personal computer which is then interfaced with the software stack. The AQS and TAIP subsystems will need to interface with the WVI subsystem to send the data that is going to be displayed on the Web Interface. The ST subsystem will need to interface with the WVI subsystem to see if the manual deploy of the sampling tube button has been checked. Details of the interfaces are shown in the table and figure below.

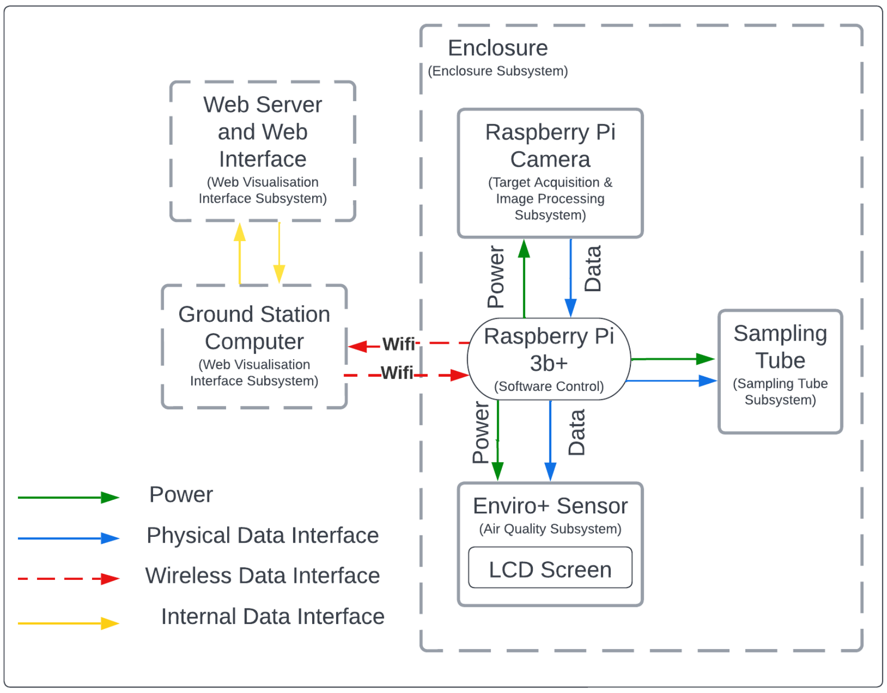


Figure 2: WVI Interfaces Diagram

Table 4: WVI Interfaces and Interface Description

Interface	Component	Interface Description
Interface 1	Raspberry Pi 3B+	Data is sent from the Raspberry Pi to the Ground Station Computer.
Interface 2	Group Station Computer	The Server is hosted on the Ground Station Computer
Interface 3	Air Quality Sensor	The WVI subsystem will interface with the Air Quality Server Subsystem as the WVI subsystem will need to receive data from the AQS to display on the Web Interface. Type: Wireless Connection Data Type: JSON

Interface 4	Target Analysis and Image processing	The WVI subsystem will need to interface with the TAIP subsystem to receive processed images, live video stream and other data. The data sent from the TAIP subsystem will be displayed on the Web Interface. Type: Wireless Data Type: Images, Video Stream, JSON
Interface 5	Sampling Tube	The WVI subsystem will need to interface with the ST subsystem. The ST subsystem will query the server and check to see if the manual deploy button has been enabled. The server will respond to see if the sampling tube should be deployed. Type: Wireless Data Type: JSON

5 Design

After finalising the system requirements for the Web Interface research was conducted as a section of the Systems Engineering Process. It was decided that the Web Server component of the WVI subsystem would use the Node.Js Server Framework. The Node.Js Server Framework is a Modern JavaScript server framework that has the capability to meet the project requirements. The framework allows for easy integration with a MySQL Database and React.Js. React.Js is a front-end JavaScript library that will allow the development of a modern web interface that can meet the requirements of receiving and displaying the Data within 10 seconds without the need of manually requesting the data (For example, refreshing the web page). The Final design of the WVI subsystem is based on this Web Server and Frontend stack.

5.1 Hardware Design

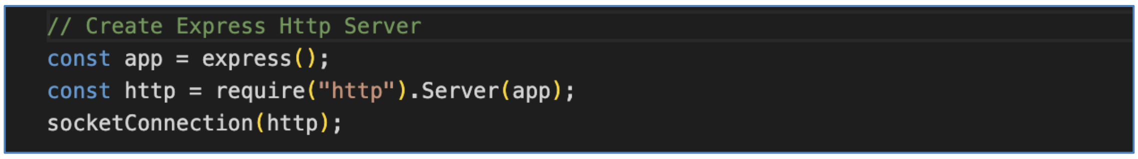
The WVI subsystem can be run on any PC regardless of the PC operating system. This means that there is no physical hardware needed to complete this subsystem. Therefore, there is no Hardware design for this subsystem.

5.2 Software Design

To complete the WVI subsystem and meet client requirements, the WVI relies on a web software-based environment. The Software Design will implement industry software design principles. These principles include documented software for an easily maintainable subsystem, correct software formatting for readability, refactorization where needed and testing. This ensures that if Group 19 needs to make future adjustments, the code can be easily changed and/or if the member working on the WVI is unable to complete the subsystem due to personal impact, another member can continue working.

5.2.1 Web Interface Server

The Web Interface is hosted on a server framework called Node.js. Node.js is a server framework that is developed in the language JavaScript. Node.Js allows for easy integration between the Database and the Web User Interface. The server will allow request from traffic on the same local network, allowing the Payload to communicate with the server. The server allows for communication on specified access application programming interface (API) endpoints and rejects request that fall outside of these specified endpoints. The Web Interface connects to the Web User Interface and Database, and acts as the middle-point between the user interface and database.



```
// Create Express Http Server
const app = express();
const http = require("http").Server(app);
socketConnection(http);
```

Figure 3: Node.Js Initializing Server Code

5.2.2 Web User Interface

The Web User Interface is hosted on a client-side framework called React. React is a user interface framework that is developed in the language JavaScript. React allows for easy integration with a server whilst supporting a separation of concerns. The framework allows for integration with other frameworks the are needed to meet the project requirements. The web user interface can be used on any browser type that allows JavaScript, including mobile. The Web User interface has been designed with ease of use in mind, such as the main page when opening the user interface will show all the up-to-date information that has been sent from the payload. The web user interface has 4 interactive graphs displaying AQS data and 3 image sections displaying still images and an image stream sent from the TAIP subsystem. This is shown in figure 4.

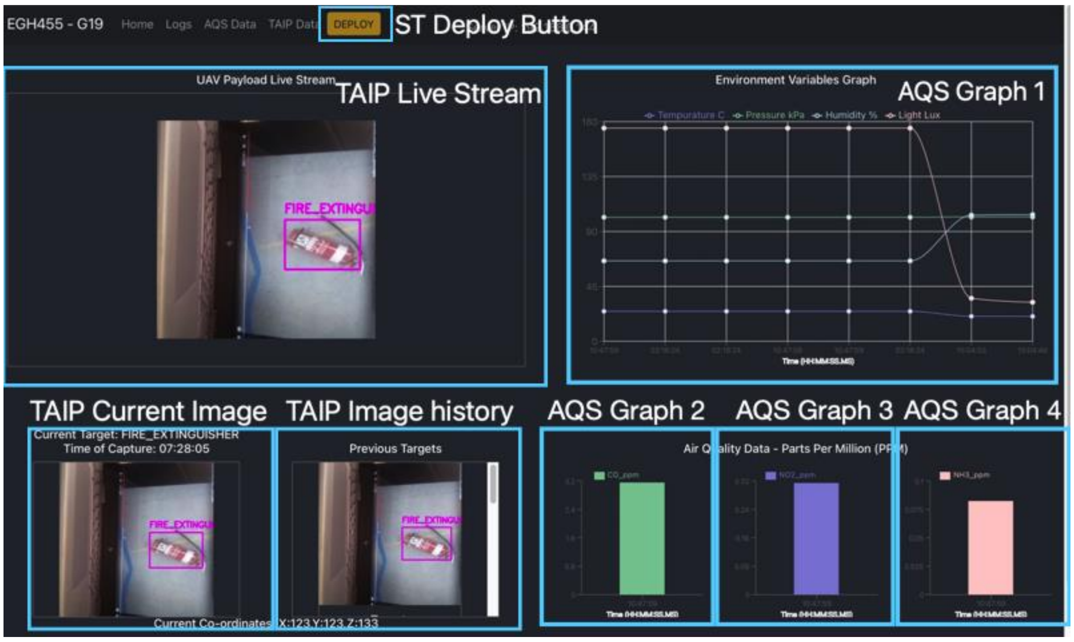


Figure 4: Annotated Web Interface

Web Interface Database

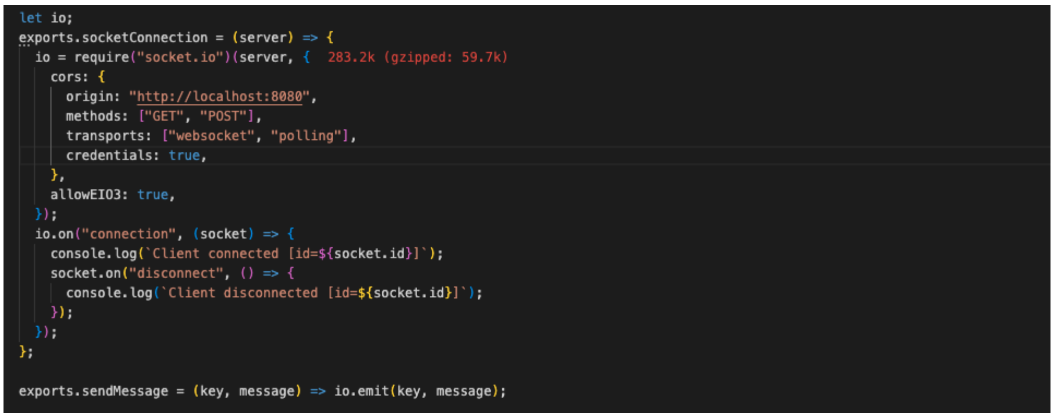
The Database the Web Interface uses is MySQL. MySQL is an open-source database management system. The system allows for users to create tables that hold data sent from the ASP. MySQL uses Structured Query Language (SQL) to make requests to the database to Update, Delete or Select Data. The Database will be used store data sent from the ASP and therefore, be accessed by the front end at any time. The database has been designed to separate the incoming data into multiple tables to ensure the data is easily obtainable. The Database design is shown in Appendix A. The Code to initialise the Database is shown in Appendix B.

5.2.3 JavaScript Library

Opensource JavaScript libraries have been used to help with integration between the Server, Database, and frontend. These JavaScript libraries are crucial to the WVI.

JavaScript Library: Socket.io

The WVI subsystem heavily relies on Web Sockets for constant communication between the server and the user interface without the need of refreshing the page. When the server receives new data from the payload, a broadcast from the server is made over the Web Socket framework which is then received by the user interface. The user interface will then request fresh data from the server and display it to the user. Socket.io is a JavaScript framework that creates a web socket server that allows for this communication. Socket.io is open source and widely adapted in professional production. The Web Interface uses Socket.io to notify the user interface to request new data and is also used to transfer a live image stream to the Web Interface when the payload is sending an image stream from the TAIP subsystem. Web Socket code used for the server is shown in Figure 5.



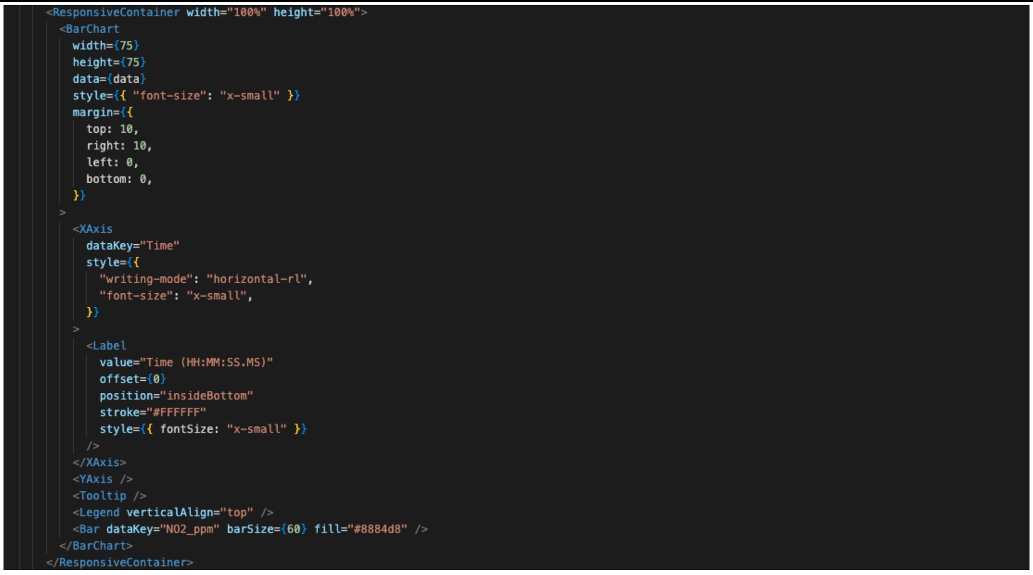
```
let io;
exports.socketConnection = (server) => {
  io = require("socket.io")(server, {  283.2k (gzipped: 59.7k)
    cors: {
      origin: "http://localhost:8080",
      methods: ["GET", "POST"],
      transports: ["websocket", "polling"],
      credentials: true,
    },
    allowEIO3: true,
  });
  io.on("connection", (socket) => {
    console.log(`Client connected [id=${socket.id}]`);
    socket.on("disconnect", () => {
      console.log(`Client disconnected [id=${socket.id}]`);
    });
  });
};

exports.sendMessage = (key, message) => io.emit(key, message);
```

Figure 5: Web Socket Component for Web Server

JavaScript Library: ReCharts

The user interface displays AQS data in interactive graphs that allow the user to easily read the data. The graphs are created using a free JavaScript library called ReCharts. ReCharts is a library specifically developed for graphing data. The user interface uses 3 bar graphs and 1 line graph to display the AQS data. This graphing library allows for significant customization of the graphs, such as changing the colours, size, axis labels, titles, and legends on the graph. ReCharts also support real-time data change, this means when the user interface receives new data the graph handles the added information without needing to refresh the page. Code to create a graph is shown in Figure 6.

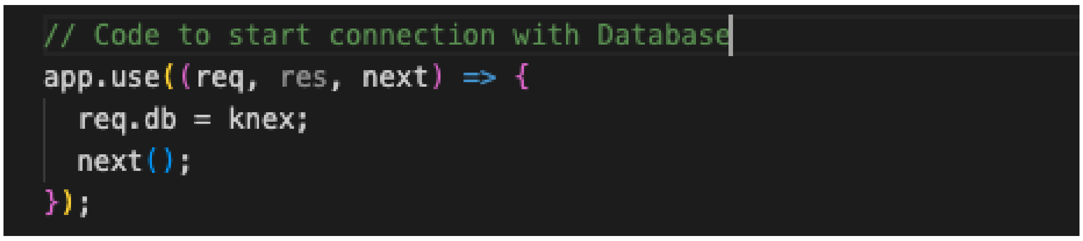


```
<ResponsiveContainer width="100%" height="100%">
  <BarChart
    width={75}
    height={75}
    data={data}
    style={{ "font-size": "x-small" }}
    margin={{
      top: 10,
      right: 10,
      left: 0,
      bottom: 0,
    }}
  >
    <XAxis
      dataKey="Time"
      style={{
        "writing-mode": "horizontal-rl",
        "font-size": "x-small",
      }}
    >
      <Label
        value="Time (HH:MM:SS.MS)"
        offset={0}
        position="insideBottom"
        stroke="#FFFFFF"
        style={{ fontSize: "x-small" }}
      />
    </XAxis>
    <YAxis />
    <Tooltip />
    <Legend verticalAlign="top" />
    <Bar dataKey="NO2_ppm" barSize={60} fill="#8884d8" />
  </BarChart>
</ResponsiveContainer>
```

Figure 6: ReCharts code to generate bar graph

JavaScript Library: Knux

The WVI subsystem uses a MySQL Database to store the data sent from the payload. The payload will send the data to the server and the server saves that data in the Database. Knux is an open-source JavaScript library that allows the server to interface with the Database, thus enabling the server to save the data in the data base. Code to initialise connection with the Database is shown in Figure 7.

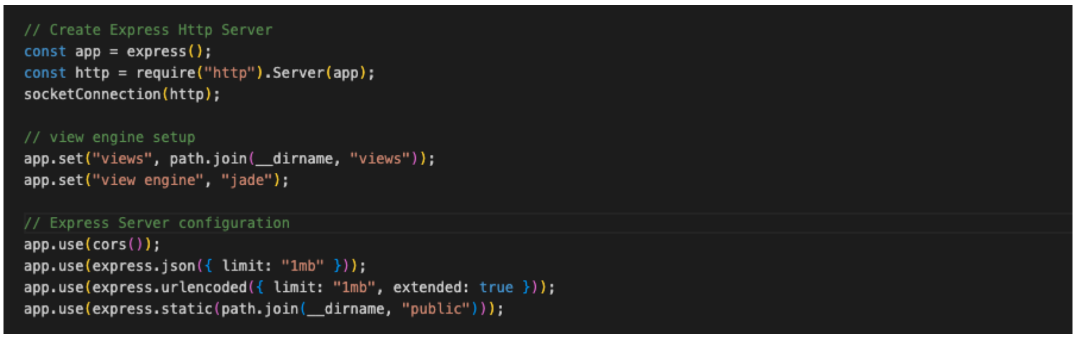


```
// Code to start connection with Database
app.use((req, res, next) => {
  req.db = knex;
  next();
});
```

Figure 7: Knex Database connection code

JavaScript Library: Express

The Web Interface server handles the requests from the payload. For the server to handle the request, the server has interfaces with a JavaScript library called Express. Express allows for the creation of a HTTP server which enables the server to handle HTTP request. HTTP request and used by the payload and the user interface to send and receive data. Express also allows for further server customisation, such as maximum data transfer limit, which was set to 1MB per request as shown in Figure 8.



```
// Create Express Http Server
const app = express();
const http = require("http").Server(app);
socketConnection(http);

// view engine setup
app.set("views", path.join(__dirname, "views"));
app.set("view engine", "jade");

// Express Server configuration
app.use(cors());
app.use(express.json({ limit: "1mb" }));
app.use(express.urlencoded({ limit: "1mb", extended: true }));
app.use(express.static(path.join(__dirname, "public")));
```

Figure 8: Express Server Configuration Code

JavaScript Library: ReactStrap
The user interface uses a JavaScript library called ReactStrap to handle the functionality of generic components such as the navigation bar, buttons, footers, and tables. ReactStrap is based off a popular framework called Bootstrap, however, it is specifically made to integrate with React. ReactStrap has been used in the Web Interface to develop the navigation bar and button components in the user interface.

JavaScript Library: Axios
The user interface and the server communicate using the HTTP request framework. Axios allows for robust HTTP request to get data for the user interface from the server. Axios has been implemented in the user interface to get AQS and TAIP data from the database. Example Axios code used in the Web Interface is shown in Appendix C.

5.2.4 Web User Interface Images

The user interface uses images to display the data sent from the TAIP subsystem. The data sent from the TAIP subsystem includes an image that has been base64 encoded, thus allowing for easy transfer over the HTTP protocol. The user interface then displays this data but decoding the image using an image tag. Image tags are support by all browsers, thus allowing the user interface to displaying the TAIP data as an image.

5.2.5 Web User Interface Audio

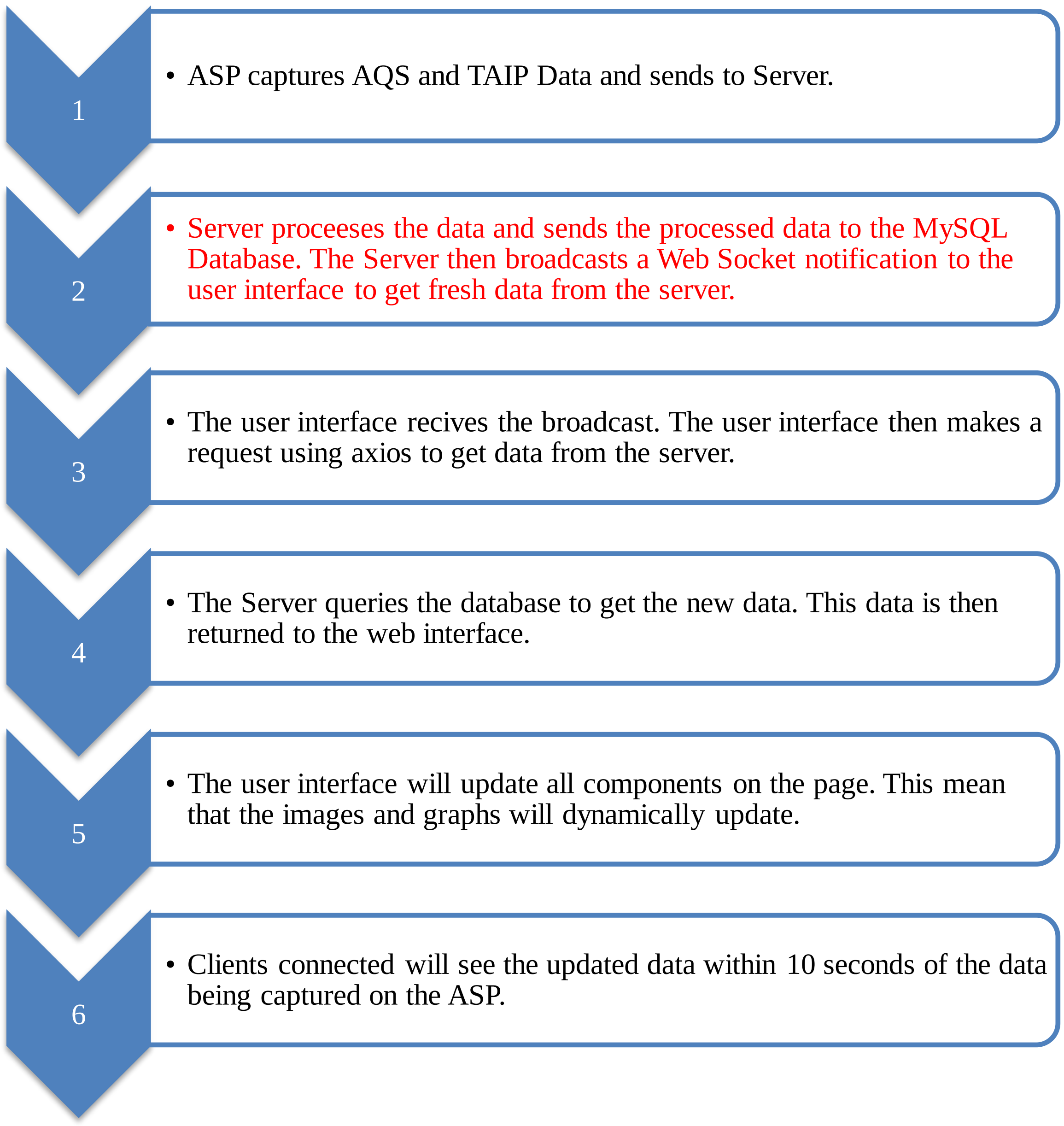
The user interface uses audio to vocalise any targets that have been detected by the TAIP system. This is implemented using a .wav recording of a text-2-speech online tool and JavaScript to play the audio once the TAIP system has sent data regarding the target.

5.2.6 Web User Interface Deployment system

The web user interface includes a button in the navigation bar that is disabled. The button will set a flag once pushed notifying the ST subsystem that it can deploy the sampling tube mechanism. The button will become available once the TAIP system send a request to the server to enable the button. The user interface can visually distinct between the displayed button and an enabled button by colour. The displayed button will be greyed out, this is implemented to ensure that the user knows when the button become available to use.

5.2.7 Software Flow Diagram

The software flow diagram details the smaller subsystems of the WVI and how the software will flow from one of these smaller subsystems to the next for the WVI to be affective. This is displayed in Figure 9.



1
- ASP captures AQS and TAIP Data and sends to Server.

2
- Server proceeses the data and sends the processed data to the MySQL Database. The Server then broadcasts a Web Socket notification to the user interface to get fresh data from the server.

3
- The user interface recives the broadcast. The user interface then makes a request using axios to get data from the server.

4
- The Server queries the database to get the new data. This data is then returned to the web interface.

5
- The user interface will update all components on the page. This mean that the images and graphs will dynamically update.

6
- Clients connected will see the updated data within 10 seconds of the data being captured on the ASP.

Figure 9: Software Flow Diagram

5.3 Web interface Design

The Web Interface is designed to show the data collected on an online application that is accessible from the browser. The Web Interface is also designed to display the information in a readable manner for the client viewing the webpage. The Web Interface will host 4 webpages: The Home Page, Logs page, AQS page and TAIP page.

5.3.1 Home Page Design

The home page will display 4 main components: 3 bar graphs for air quality gas data sent from the AQS (RD/31), a graph component to display the air quality environmental data sent from the AQS subsystem (RD/31), a component to hold the images sent from the TAIP subsystem (RD/32), a live video component to display the live video stream from the TAIP (RD/32). The homepage will also display 3 subcomponents to add extra functionality to the Web Interface, these include a navigation component for navigation to and from the Logs webpage, a server status component to show that the React.Js webpage is connected to the Node.Js server and a manual deploy button for the ST subsystem (RD/33). The Home page is shown in Figure 10 below with an annotated Home Page shown in Figure 11.

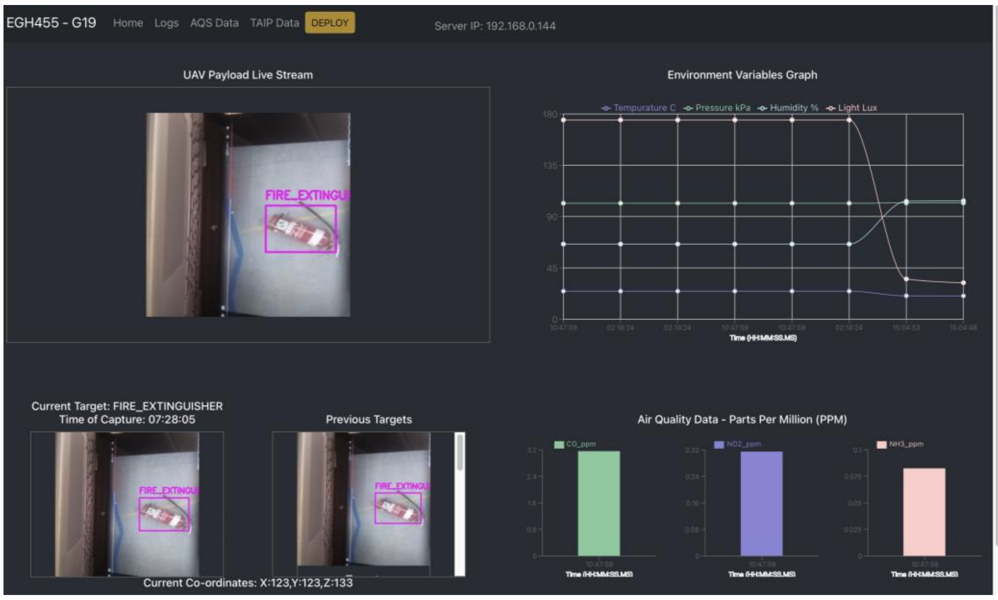


Figure 10: Non-Annotated Visual of Home Page

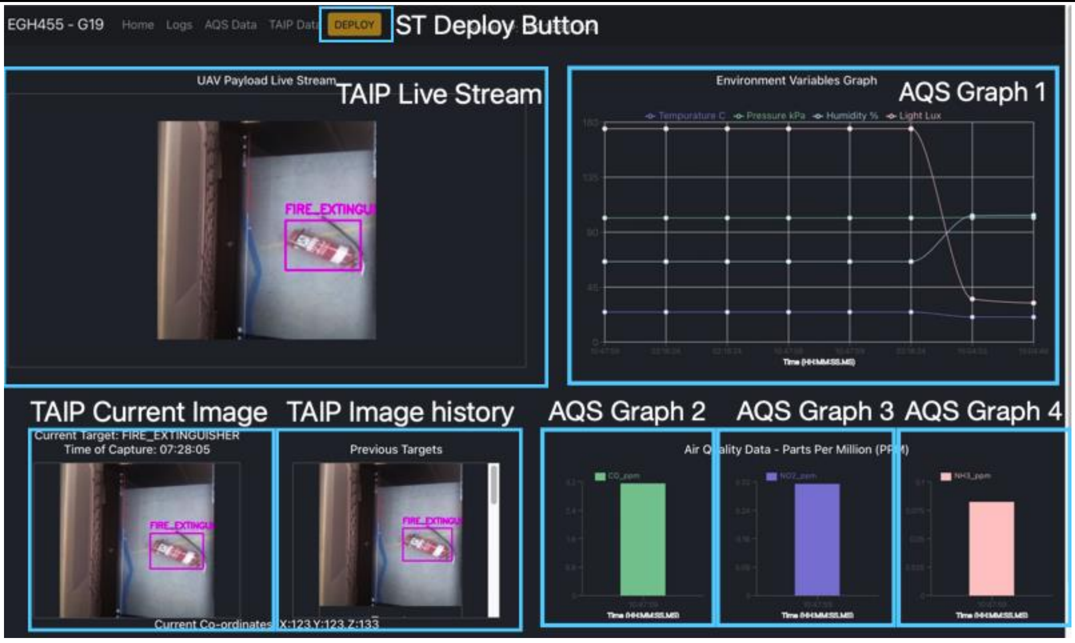
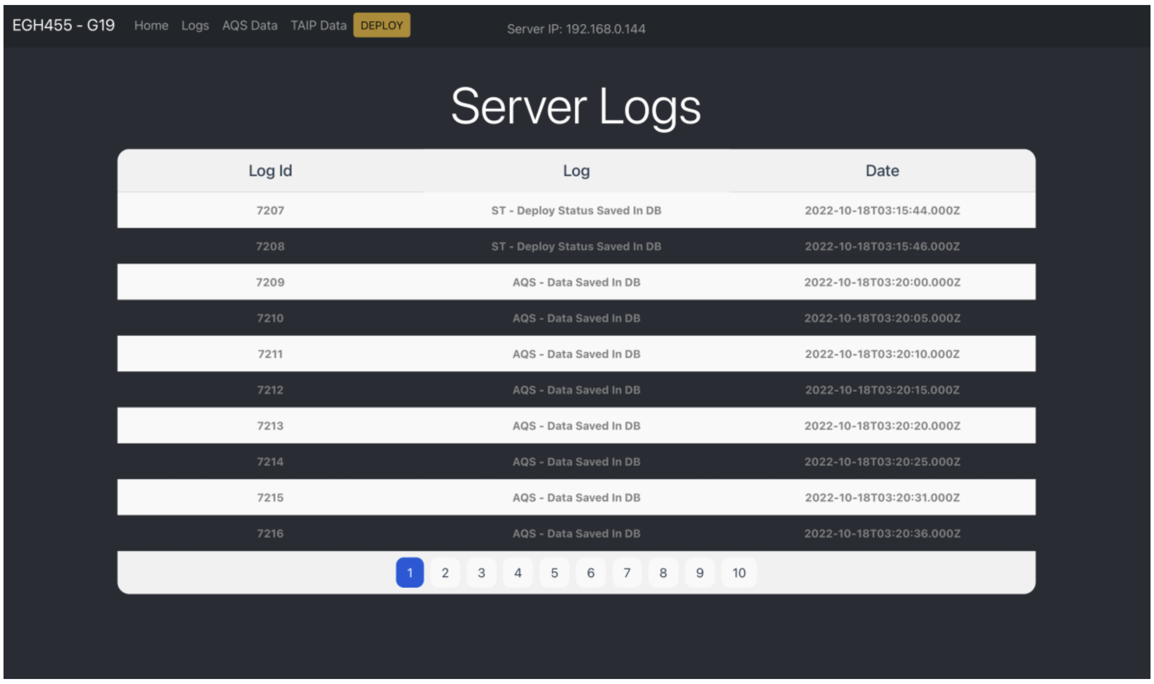


Figure 11: Annotated Visual of Home Page

5.3.2 Logs Page Design

The Logs page will display 1 main component. This is the logs recorded by the server. The server will record a log in the database when the server receives a request from sent from the ASP. The logs will include information regarding the request and a timestamp of the request. This can be used to verify the requirement REQ-M-15. The Logs page also included a navigation subcomponent for navigation to and from the home webpage. The Logs page is shown in Figure 12 below.



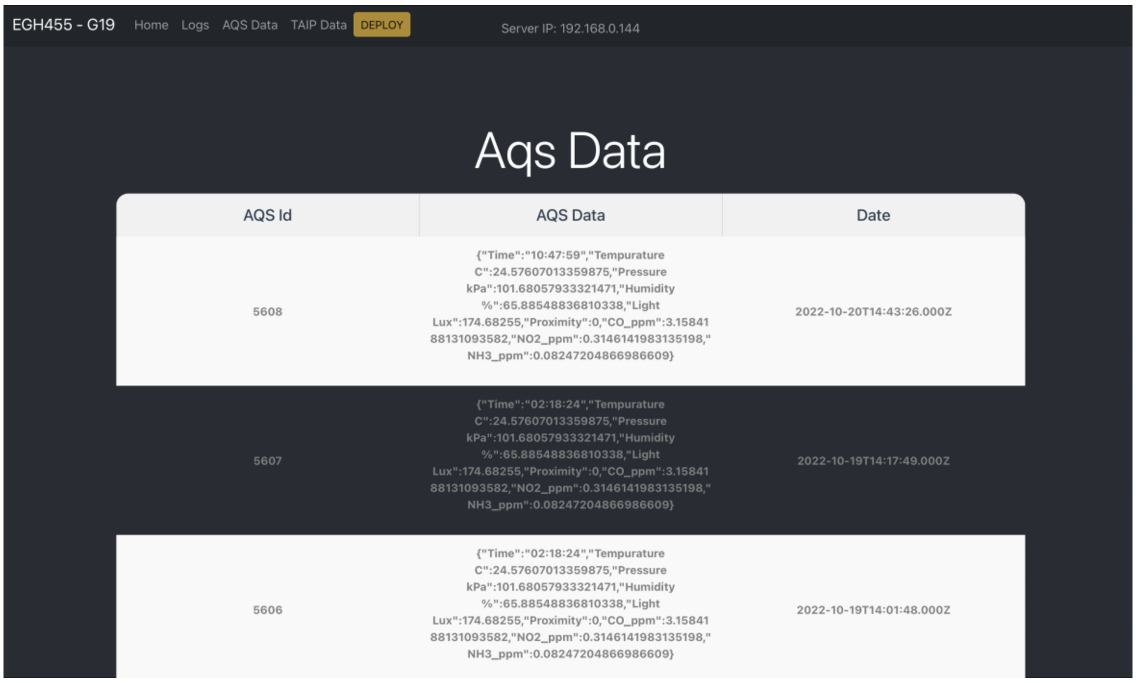
Server Logs

Log Id	Log	Date
7207	ST - Deploy Status Saved In DB	2022-10-18T03:15:44.000Z
7208	ST - Deploy Status Saved In DB	2022-10-18T03:15:46.000Z
7209	AQS - Data Saved In DB	2022-10-18T03:20:00.000Z
7210	AQS - Data Saved In DB	2022-10-18T03:20:05.000Z
7211	AQS - Data Saved In DB	2022-10-18T03:20:10.000Z
7212	AQS - Data Saved In DB	2022-10-18T03:20:15.000Z
7213	AQS - Data Saved In DB	2022-10-18T03:20:20.000Z
7214	AQS - Data Saved In DB	2022-10-18T03:20:25.000Z
7215	AQS - Data Saved In DB	2022-10-18T03:20:31.000Z
7216	AQS - Data Saved In DB	2022-10-18T03:20:36.000Z

Figure 12: Visual of Logs Page

5.3.3 AQS Page Design

The AQS page will display 1 main component. This is the AQS logged data recorded by the server. The server will record a log of AQS data in the database when the server receives a request from sent from the ASP. The AQS will include information regarding air quality information such as gasses, temperature, pressure, and humidity. This can be used to verify the requirement REQ-M-15. The AQS page also included a navigation subcomponent for navigation to and from the home webpage. The AQS page is shown in figure 13 below.



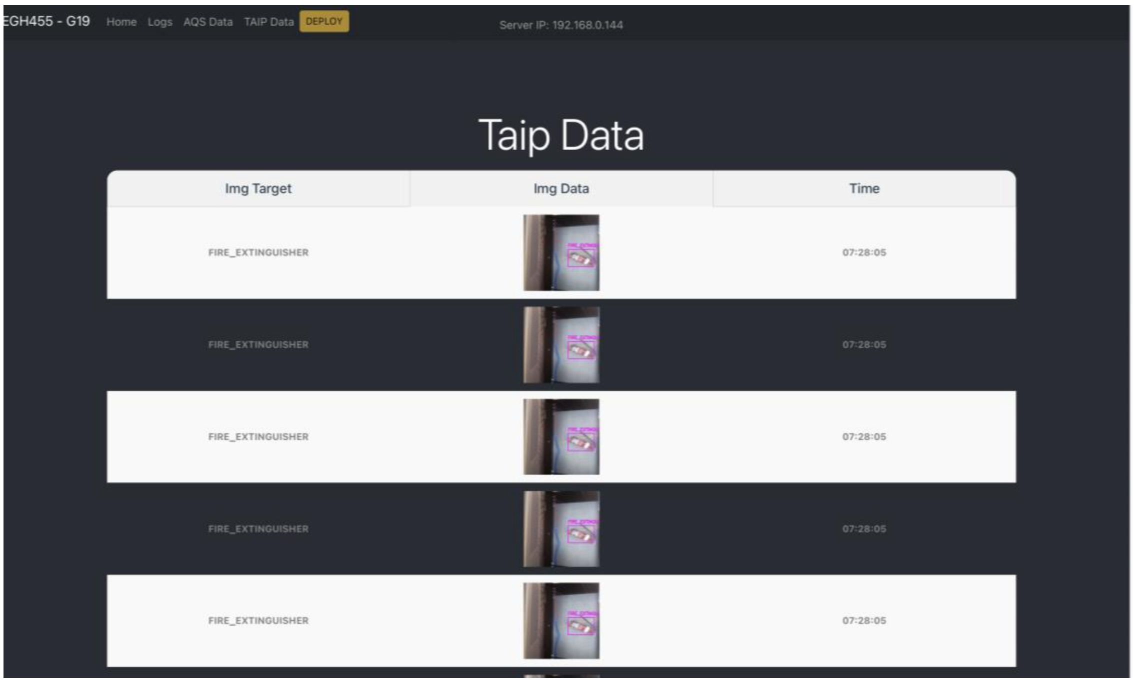
Aqs Data

AQS Id	AQS Data	Date
5608	{"Time":"10:47:59","Tempurature C":24.57607013359875,"Pressure kPa":101.68057933321471,"Humidity %":65.88548836810338,"Light Lux":174.68255,"Proximity":0,"CO_ppm":3.1584188131093582,"NO2_ppm":0.3146141983135198,"NH3_ppm":0.08247204866986609}	2022-10-20T14:43:26.000Z
5607	{"Time":"02:18:24","Tempurature C":24.57607013359875,"Pressure kPa":101.68057933321471,"Humidity %":65.88548836810338,"Light Lux":174.68255,"Proximity":0,"CO_ppm":3.1584188131093582,"NO2_ppm":0.3146141983135198,"NH3_ppm":0.08247204866986609}	2022-10-19T14:17:49.000Z
5606	{"Time":"02:18:24","Tempurature C":24.57607013359875,"Pressure kPa":101.68057933321471,"Humidity %":65.88548836810338,"Light Lux":174.68255,"Proximity":0,"CO_ppm":3.1584188131093582,"NO2_ppm":0.3146141983135198,"NH3_ppm":0.08247204866986609}	2022-10-19T14:01:48.000Z

Figure 13: Visual of AQS Page

5.3.4 TAIP Page Design

The TAIP page will display 1 main component. This is the TAIP logged data recorded by the server. The server will record a log of TAIP data in the database when the server receives a request from sent from the ASP. The TAIP will include data such as target information, image data and image time. This can be used to verify the requirement REQ-M-15. The TAIP page also included a navigation subcomponent for navigation to and from the home webpage. The TAIP page is shown in Figure 14 below.



Taip Data

Img Target	Img Data	Time
FIRE_EXTINGUISHER		07:28:05
FIRE_EXTINGUISHER		07:28:05
FIRE_EXTINGUISHER		07:28:05
FIRE_EXTINGUISHER		07:28:05
FIRE_EXTINGUISHER		07:28:05

Figure 14: Visual of TAIP Page

6 Conclusion

The WVI subsystem has been designed to satisfy all requirements supplied in the system requirements (RD/2). As this subsystem contains no hardware components, only software components have been designed. The developed design meets the requirements of REQ-M-03 for the ASP to communicate to transmit video, images, and data. The requirement REQ-M-04 and REQ-M-06, sending target alerts that are captured from the TAIP subsystem and display images captured from the TAIP subsystem is complete as the WVI handles the alerts along with the images and will display then on the Web Interface. The requirements REQ-M-05 and REQ-M-19, displaying real time data and dynamically updating the page within 10 seconds of capture, is met as the server forces the clients to update the data on the page once the server stores the data. Lastly the requirements REQ-M-07 and REQ-M-15, the WVI must be hosted on a local machine and be accessible over a local network with 10 minutes of logged data prior to acceptance testing, the Web Server is accessible to anyone on the same network and due to prior testing, the server will have logged more then 10 minutes of data before acceptance testing. Moreover, following the systems engineering approach, the WVI subsystem design has met all the requirements. The software architecture in this final design may be changed due to unforeseen results during the testing phase of the project.

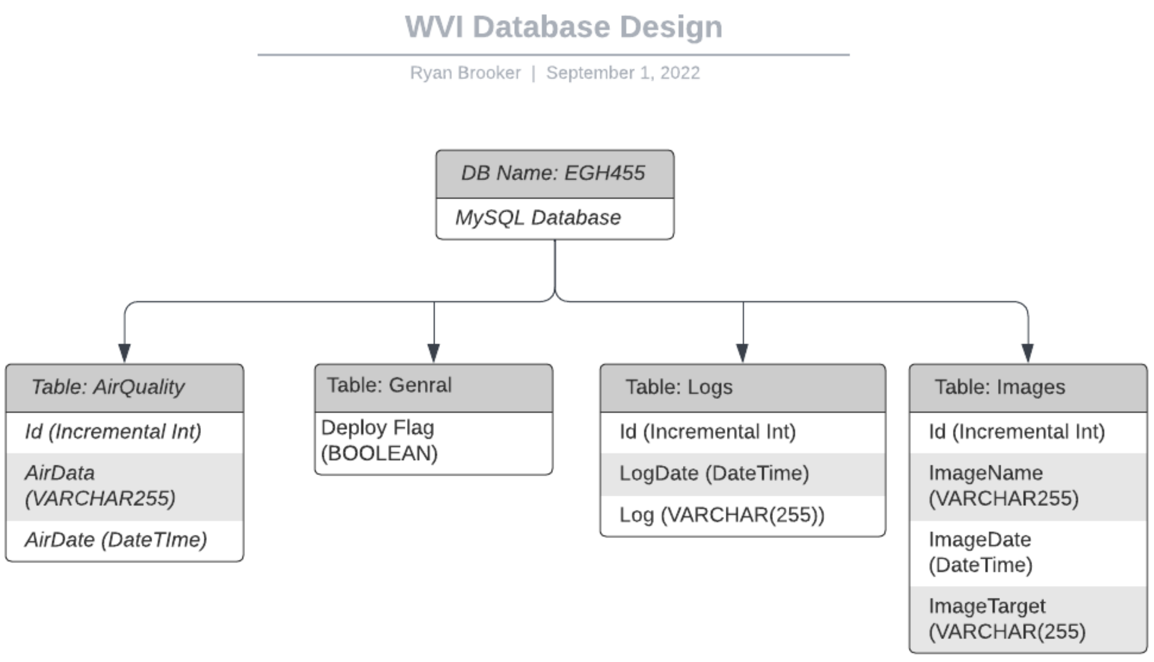
Table 5: Requirements Met by Final Design

Requirements	Description	Verification
REQ-M-03	The UAVPayloadTAQ shall communicate with a ground station computer to transmit video, target detection and air quality data.	Met: -This requirement has been met as seen by the results of Software Test 1 and further displayed in Software Test 4 in RD/25.
REQ-M-04	The target identification system shall be capable of alerting the GCS of a target's type.	Met: -This requirement has been met as seen by the results of Software Test 8 in RD/25.
REQ-M-05	The Web Interface is required to display real time air sampling data that is recorded directly from the UAVPayloadTAQ and updated dynamically throughout the duration of the flight.	Met: -This requirement has been met as seen by the results of Software Test 4 in RD/25.
REQ-M-06	The Web Interface is required to display the images of the targets that are taken directly from the UAVPayloadTAQ and updated every time a new picture is taken.	Met: -This requirement has been met as seen by the results of Software Test 4 in RD/25.
REQ-M-07	The Web Interface shall be designed and run as a web server, which is to be accessible by any computers on the local network. This shall store logged sensor data and target detections	Met: -This requirement has been met as seen by the results of Software Test 6 in RD/25.

	with corresponding timestamps.	
REQ-M-10	Preliminary designs shall be completed by week 7	Met: -Submitted documentation
REQ-M-15	The system shall have logged functioning operation for a minimal period of 10 minutes prior to acceptance test.	Met: -This requirement has been met as seen by the results of Software Test 7 in RD/25.
REQ-M-19	Live data from the UAV must be made available through the web server within 10 seconds of capture.	Met: -This requirement has been met as seen by the results of Software Test 2 in RD/25.

7 Appendix

Appendix A Database Design Diagram



Appendix B Database Initialization Code

```sql
CREATE DATABASE IF NOT EXISTS EGH455;
USE EGH455;

CREATE TABLE IF NOT EXISTS `aqs` (
  `id` int NOT NULL AUTO_INCREMENT,
  `airQualityData` varchar(500) DEFAULT NULL,
  `airQualityDate` datetime DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=4975 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
;
CREATE TABLE IF NOT EXISTS `general` (
  `deployFlag` BOOLEAN DEFAULT 0
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

CREATE TABLE IF NOT EXISTS `taip` (
  `id` int unsigned NOT NULL AUTO_INCREMENT,
  `imageName` varchar(255) DEFAULT NULL,
  `imageData` longtext,
  `imageDate` datetime DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=75 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

CREATE TABLE IF NOT EXISTS `logs` (
  `id` int NOT NULL AUTO_INCREMENT,
  `logData` varchar(255) DEFAULT NULL,
  `logDate` datetime DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=11 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

Appendix C Axios Example Code

```
  // Example Axios Request to GET TAIP Logged Data
  axios("/taip/getTaipLogData")
    .then((res) =>
      setDataTable(() => {
        return res.data;
      })
    )
    .catch((err) => console.log(err));
```

```
//Axios example to POST the deploy flag to the server
axios.post("/st/deployFlag", { deploy: "true" }).then((res) => {
  console.log(res.data);
});
```