Documentation of Software Components

1.0 Overview of the System

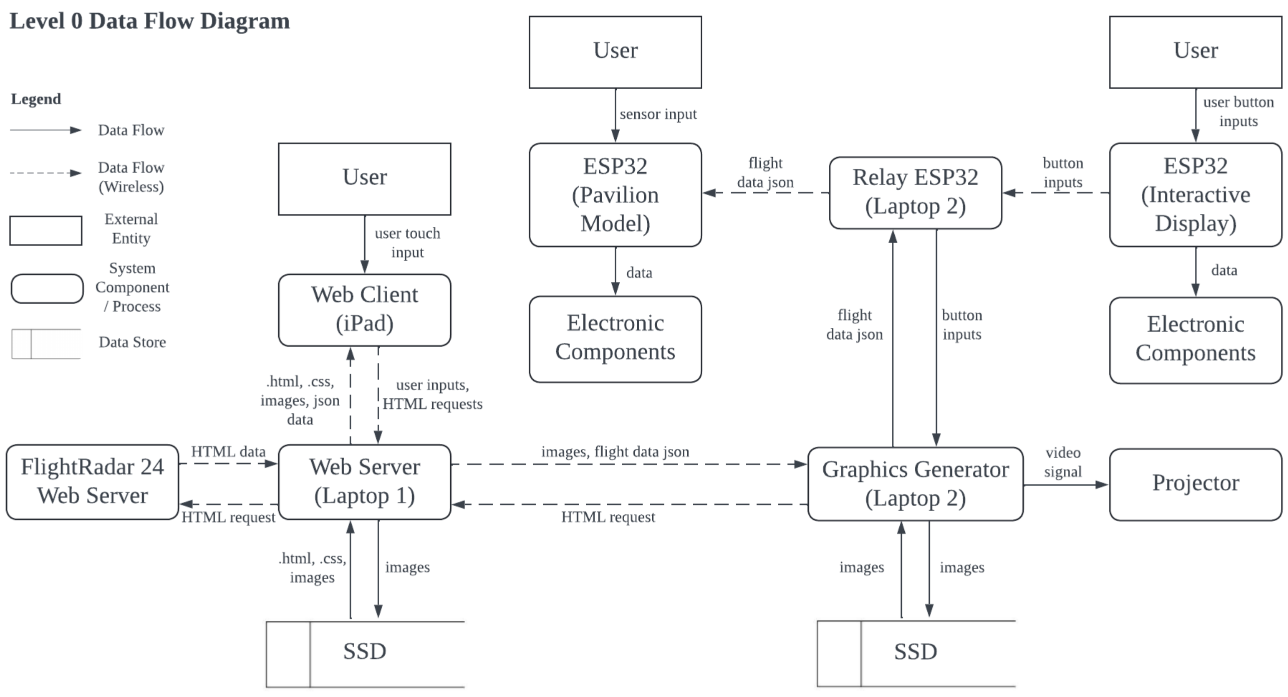


Figure 1.0.1: Data Flow Diagram

The above Data Flow Diagram (DFD) shows an overview of the components of our software system and the flow of data between them.

2.0 Web Server Code

2.1 Overview and File Structure

The Web Server obtains flight data from FlightRadar24, allows the webpage for the message designer terminal to be displayed at the Web Clients, generates the message image which is then sent to the Graphics generator.

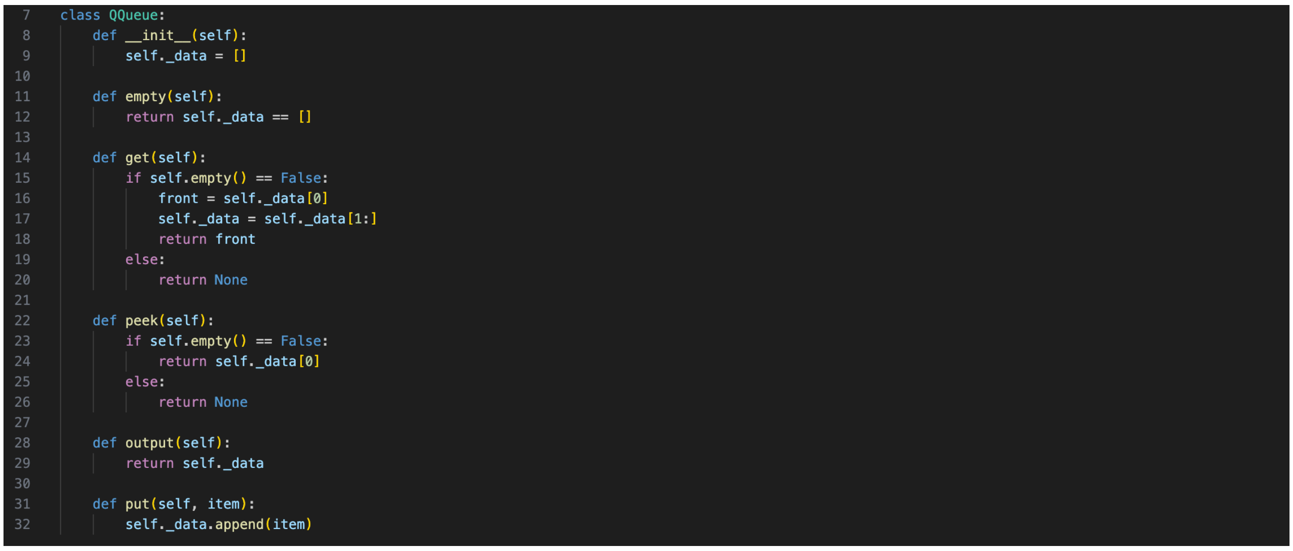
The languages used are: Python for the backend; html, css and javascript for the frontend.

File Structure:

```
Parent Directory
> static
    > generated
        - <images that are generated will be stored in this folder>
    - style.css
    - <images for website, image generation are stored in this folder>
> templates
    - designer.html
    - index.html
    - preview.html
    - publish.html
    - queue.html
- QuickQueue.py
- SkyScraper.py
- WebServer.py
```

2.2 QuickQueue.py

QuickQueue.py defines a queue data structure for use in WebServer.py. A queue is a data structure that allows elements to be inserted or removed in a first-in-first-out order.



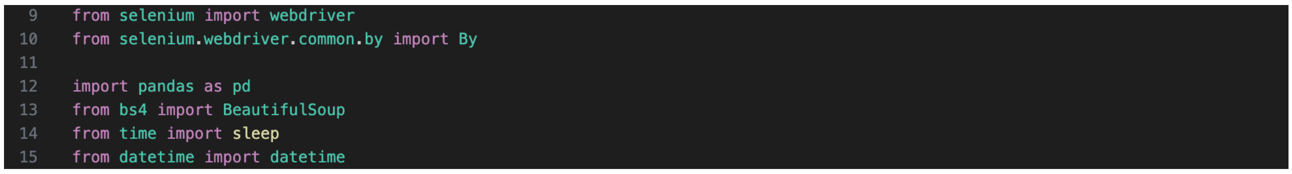
```python
class QQueue:
    def __init__(self):
        self._data = []

    def empty(self):
        return self._data == []

    def get(self):
        if self.empty() == False:
            front = self._data[0]
            self._data = self._data[1:]
            return front
        else:
            return None

    def peek(self):
        if self.empty() == False:
            return self._data[0]
        else:
            return None

    def output(self):
        return self._data

    def put(self, item):
        self._data.append(item)
```

Figure 2.2.1: Code for QuickQueue.py

2.3 SkyScraper.py

SkyScraper.py contains code that scrapes plane arrival data from the Flightradar24 website.

2.3.1 Module Imports



```python
from selenium import webdriver
from selenium.webdriver.common.by import By

import pandas as pd
from bs4 import BeautifulSoup
from time import sleep
from datetime import datetime
```

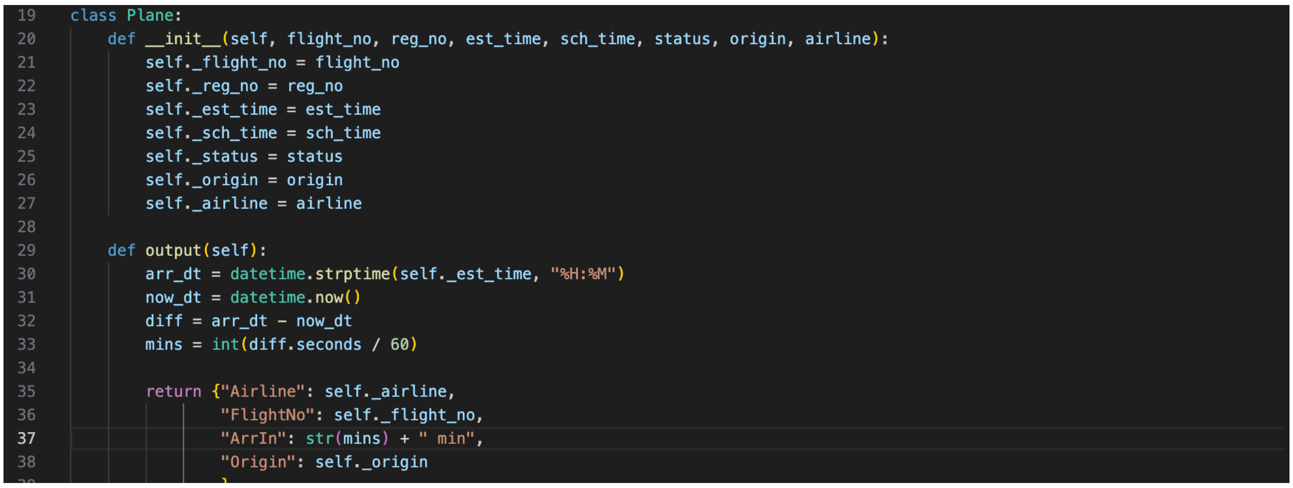
Figure 2.3.1: Import statements for SkyScraper.py

The following modules are used for SkyScraper.py:

Module Name	**Purpose**
selenium	Web scrape flight arrival data
pandas	Store data in tables
BeautifulSoup	Parse HTML data
time	sleep() is used to add delays to the code execution
datetime	Work with dates, time in code

2.3.2 Plane class

The Plane class defines all relevant attributes and methods of a Plane object. output() returns relevant data of a Plane object.

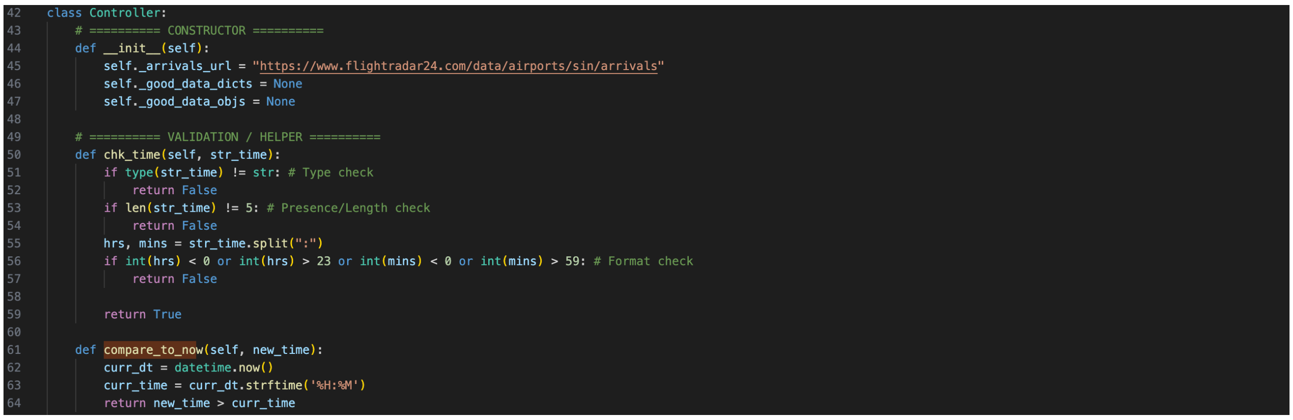


```python
class Plane:
    def __init__(self, flight_no, reg_no, est_time, sch_time, status, origin, airline):
        self._flight_no = flight_no
        self._reg_no = reg_no
        self._est_time = est_time
        self._sch_time = sch_time
        self._status = status
        self._origin = origin
        self._airline = airline

    def output(self):
        arr_dt = datetime.strptime(self._est_time, "%H:%M")
        now_dt = datetime.now()
        diff = arr_dt - now_dt
        mins = int(diff.seconds / 60)

        return {"Airline": self._airline,
                "FlightNo": self._flight_no,
                "ArrIn": str(mins) + " min",
                "Origin": self._origin
```

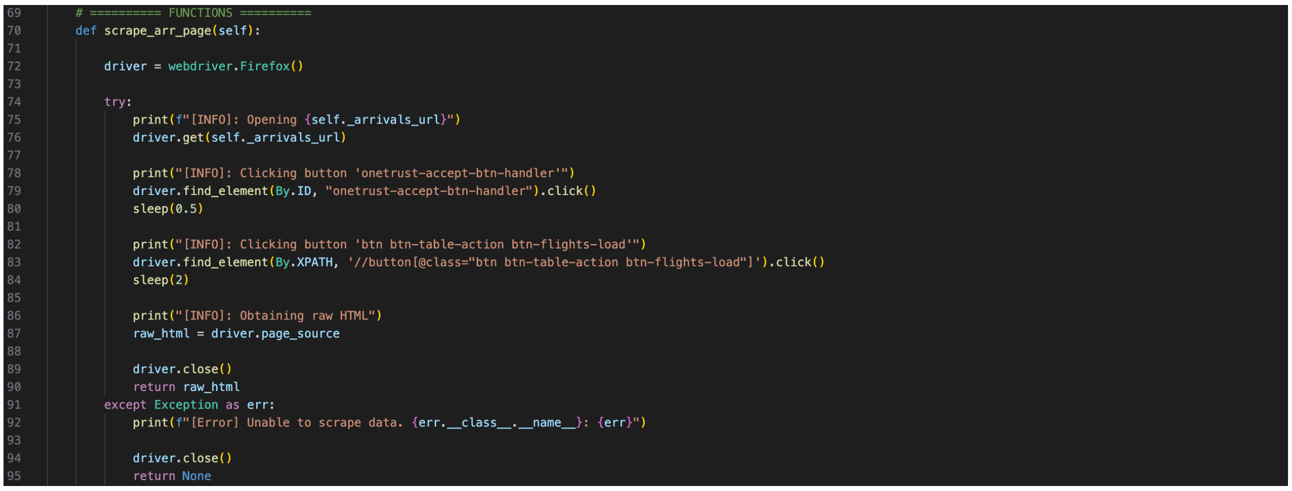
Figure 2.3.2: Code for Plane class in SkyScraper.py

2.3.3 Controller class

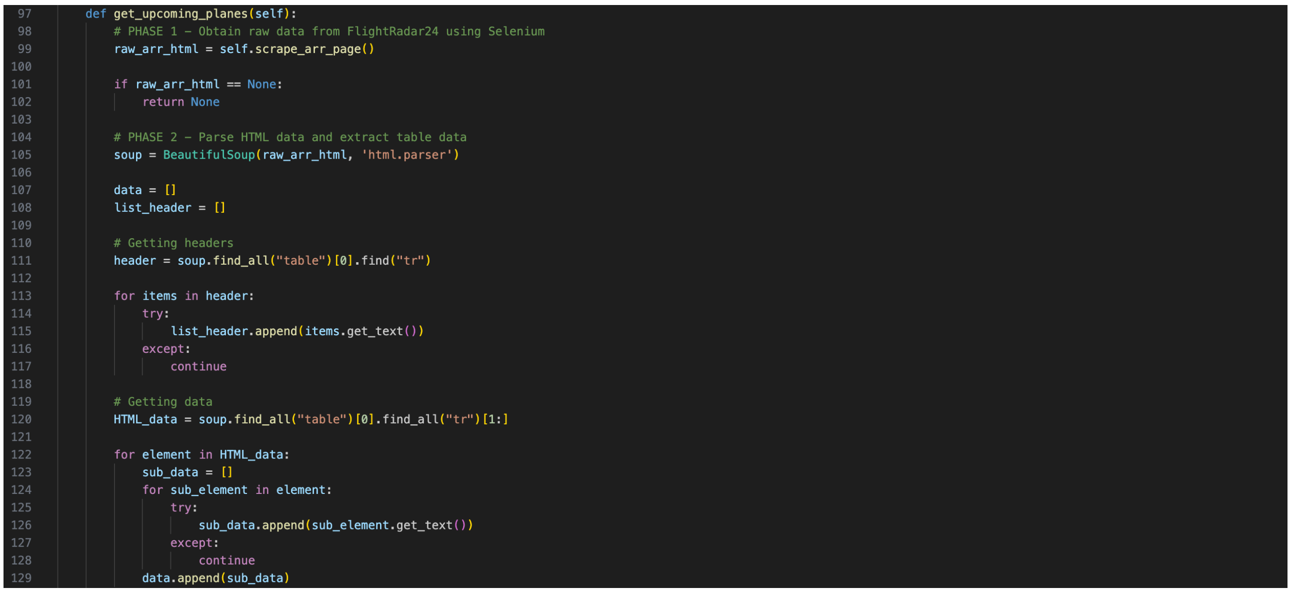
The Controller class defines all relevant attribute and methods for a Controller object, involved in the process of scraping data from the Flightradar24 website.



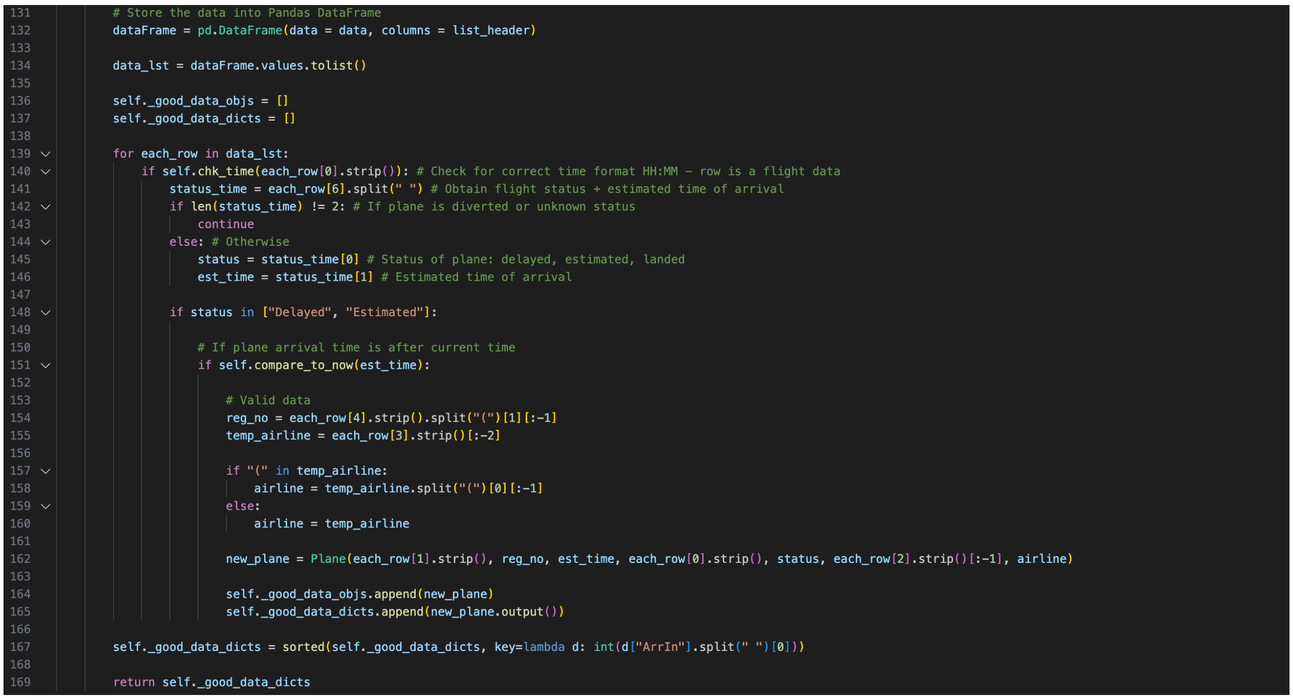
```python
class Controller:
    # ========== CONSTRUCTOR ==========
    def __init__(self):
        self._arrivals_url = "https://www.flightradar24.com/data/airports/sin/arrivals"
        self._good_data_dicts = None
        self._good_data_objs = None

    # ========== VALIDATION / HELPER ==========
    def chk_time(self, str_time):
        if type(str_time) != str: # Type check
            return False
        if len(str_time) != 5: # Presence/Length check
            return False
        hrs, mins = str_time.split(":")
        if int(hrs) < 0 or int(hrs) > 23 or int(mins) < 0 or int(mins) > 59: # Format check
            return False

        return True

    def compare_to_now(self, new_time):
        curr_dt = datetime.now()
        curr_time = curr_dt.strftime('%H:%M')
        return new_time > curr_time
```



```python
    # ========== FUNCTIONS ==========
    def scrape_arr_page(self):

        driver = webdriver.Firefox()

        try:
            print(f"[INFO]: Opening {self._arrivals_url}")
            driver.get(self._arrivals_url)

            print("[INFO]: Clicking button 'onetrust-accept-btn-handler'")
            driver.find_element(By.ID, "onetrust-accept-btn-handler").click()
            sleep(0.5)

            print("[INFO]: Clicking button 'btn btn-table-action btn-flights-load'")
            driver.find_element(By.XPATH, '//button[@class="btn btn-table-action btn-flights-load"]').click()
            sleep(2)

            print("[INFO]: Obtaining raw HTML")
            raw_html = driver.page_source

            driver.close()
            return raw_html
        except Exception as err:
            print(f"[Error] Unable to scrape data. {err.__class__.__name__}: {err}")

            driver.close()
            return None
```



```python
    def get_upcoming_planes(self):
        # PHASE 1 - Obtain raw data from FlightRadar24 using Selenium
        raw_arr_html = self.scrape_arr_page()

        if raw_arr_html == None:
            return None

        # PHASE 2 - Parse HTML data and extract table data
        soup = BeautifulSoup(raw_arr_html, 'html.parser')

        data = []
        list_header = []

        # Getting headers
        header = soup.find_all("table")[0].find("tr")

        for items in header:
            try:
                list_header.append(items.get_text())
            except:
                continue

        # Getting data
        HTML_data = soup.find_all("table")[0].find_all("tr")[1:]

        for element in HTML_data:
            sub_data = []
            for sub_element in element:
                try:
                    sub_data.append(sub_element.get_text())
                except:
                    continue
            data.append(sub_data)
```



```python
        # Store the data into Pandas DataFrame
        dataFrame = pd.DataFrame(data = data, columns = list_header)

        data_lst = dataFrame.values.tolist()

        self._good_data_objs = []
        self._good_data_dicts = []

        for each_row in data_lst:
            if self.chk_time(each_row[0].strip()): # Check for correct time format HH:MM - row is a flight data
                status_time = each_row[6].split(" ") # Obtain flight status + estimated time of arrival
                if len(status_time) != 2: # If plane is diverted or unknown status
                    continue
                else: # Otherwise
                    status = status_time[0] # Status of plane: delayed, estimated, landed
                    est_time = status_time[1] # Estimated time of arrival

                if status in ["Delayed", "Estimated"]:

                    # If plane arrival time is after current time
                    if self.compare_to_now(est_time):

                        # Valid data
                        reg_no = each_row[4].strip().split("(")[1][:-1]
                        temp_airline = each_row[3].strip()[:-2]

                        if "(" in temp_airline:
                            airline = temp_airline.split("(")[0][:-1]
                        else:
                            airline = temp_airline

                        new_plane = Plane(each_row[1].strip(), reg_no, est_time, each_row[0].strip(), status, each_row[2].strip()[:-1], airline)

                        self._good_data_objs.append(new_plane)
                        self._good_data_dicts.append(new_plane.output())

        self._good_data_dicts = sorted(self._good_data_dicts, key=lambda d: int(d["ArrIn"].split(" ")[0]))

        return self._good_data_dicts
```

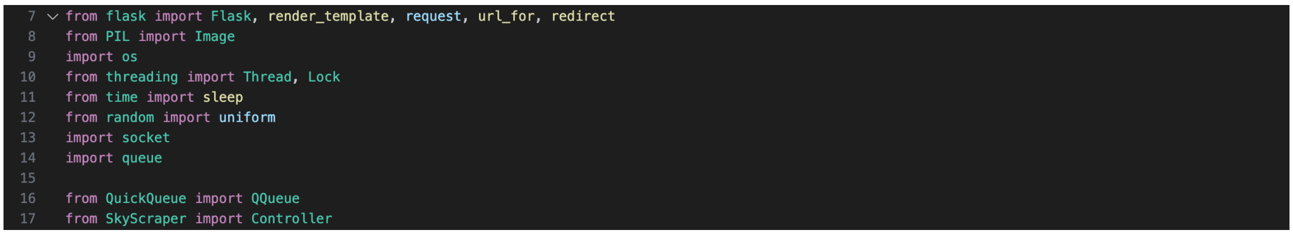
Figure 2.3.3: Code for Controller class in SkyScraper.py

1. scrape_arr_page() sends a HTML request to the FlightRadar24 web server and obtains the HTML data of the webpage from the server.
2. get_upcoming_planes() processes the HTML data to obtain the flight arrival data.

2.4 WebServer.py

WebServer.py contains code that runs the web server for the project and communicates between different devices for sending of crucial information.

2.4.1 Module Imports



```python
from flask import Flask, render_template, request, url_for, redirect
from PIL import Image
import os
from threading import Thread, Lock
from time import sleep
from random import uniform
import socket
import queue

from QuickQueue import QQueue
from SkyScraper import Controller
```

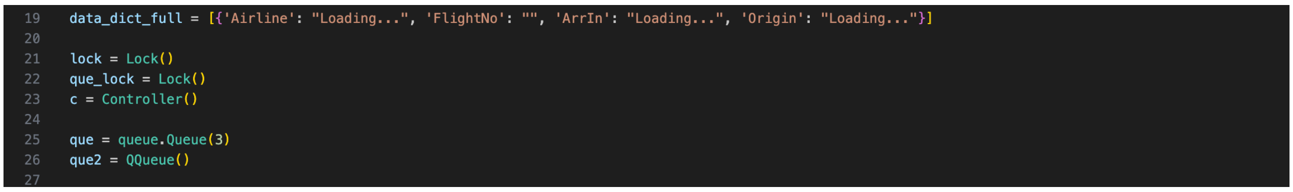
Figure 2.4.1: Module imports for WebServer.py

The following modules are used for WebServer.py:

Module Name	Purpose
flask	Runs the web server
PIL	Image processing
os	Operating System operations
threading	Allows multiple modules of code to be run simultaneously
time	sleep() is used to add delays to the code execution
random	Generate random values
socket	For communication between different devices in the network
queue	Contains a basic Python queue data structure
QuickQueue (self-defined)	Contains a queue data structure with more features
SkyScraper (self-defined)	Scrapes plane data from Flightradar24 website

2.4.2 Variables

Variables used in the global scope.

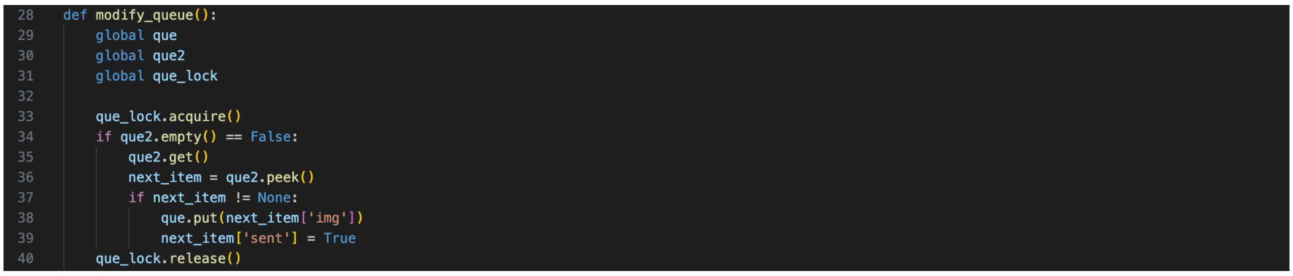


```python
data_dict_full = [{'Airline': "Loading...", 'FlightNo': "", 'ArrIn': "Loading...", 'Origin': "Loading..."}]

lock = Lock()
que_lock = Lock()
c = Controller()

que = queue.Queue(3)
que2 = QQueue()

```

Figure 2.4.2: Variables used in WebServer.py

2.4.3 modify_queue()

Obtains data from the image queue and triggers the process that sends the image to the Graphics Generator.

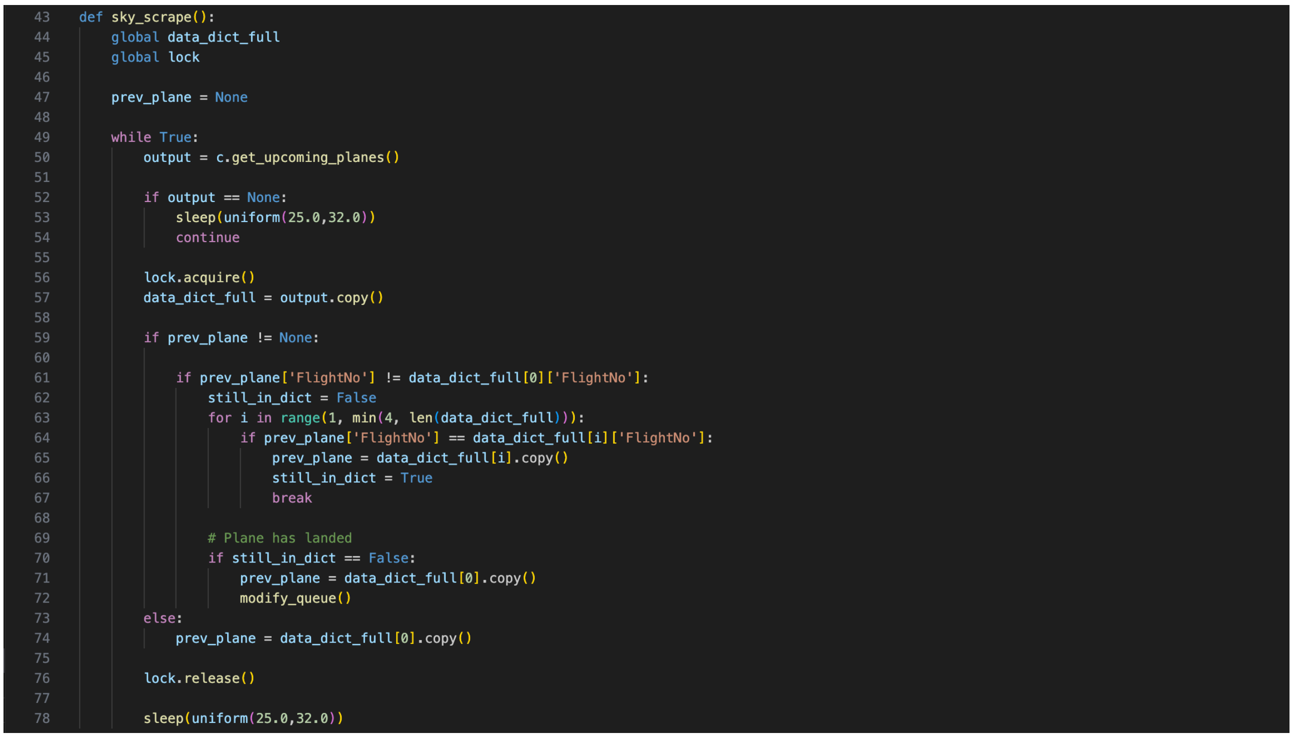


```python
def modify_queue():
    global que
    global que2
    global que_lock

    que_lock.acquire()
    if que2.empty() == False:
        que2.get()
        next_item = que2.peek()
        if next_item != None:
            que.put(next_item['img'])
            next_item['sent'] = True
    que_lock.release()
```

Figure 2.4.3: Code for modify_queue() in WebServer.py

2.4.4 sky_scrape()

Runs code to scrape data from Flightradar24 website, and detects when a plane has landed which triggers modify_queue() and sends the next image.

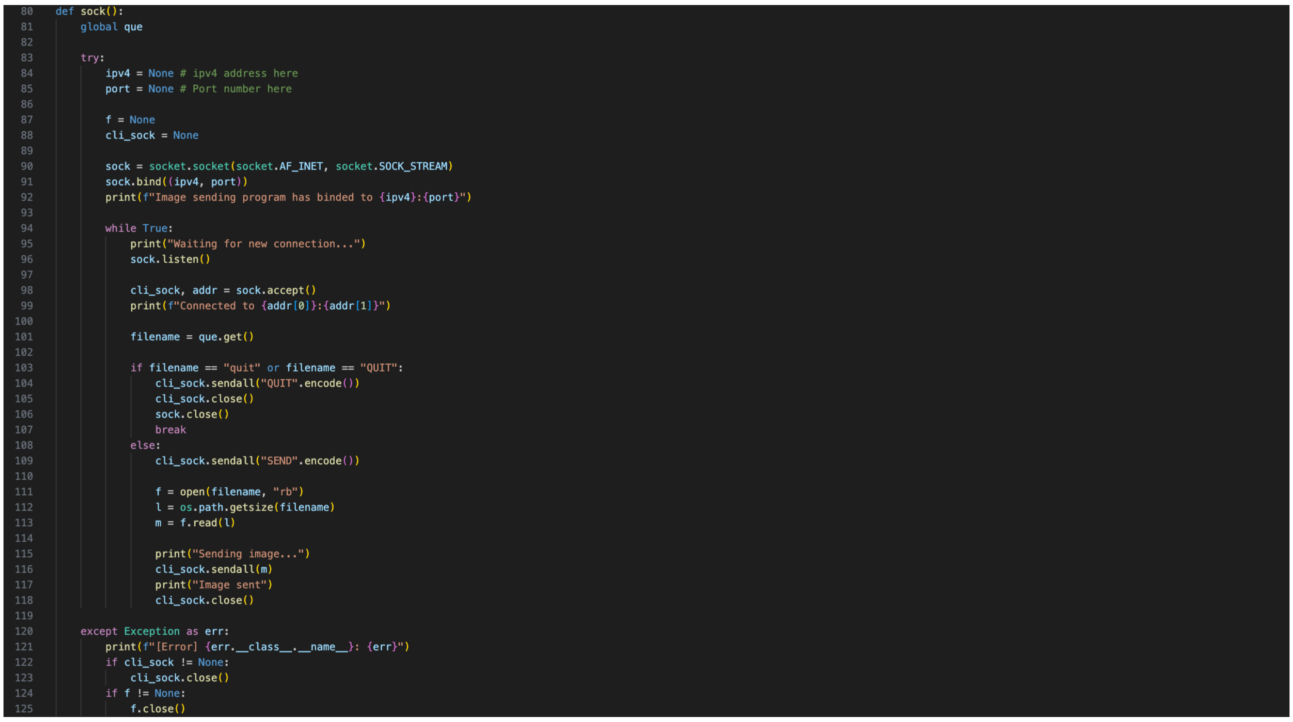


```python
def sky_scrape():
    global data_dict_full
    global lock

    prev_plane = None

    while True:
        output = c.get_upcoming_planes()

        if output == None:
            sleep(uniform(25.0,32.0))
            continue

        lock.acquire()
        data_dict_full = output.copy()

        if prev_plane != None:

            if prev_plane['FlightNo'] != data_dict_full[0]['FlightNo']:
                still_in_dict = False
                for i in range(1, min(4, len(data_dict_full))):
                    if prev_plane['FlightNo'] == data_dict_full[i]['FlightNo']:
                        prev_plane = data_dict_full[i].copy()
                        still_in_dict = True
                        break

                # Plane has landed
                if still_in_dict == False:
                    prev_plane = data_dict_full[0].copy()
                    modify_queue()
        else:
            prev_plane = data_dict_full[0].copy()

        lock.release()

        sleep(uniform(25.0,32.0))
```

Figure 2.4.4: Code for sky_scrape() in WebServer.py

2.4.5 sock()

Code that deals with sockets and sends image data to the Graphics Generator.

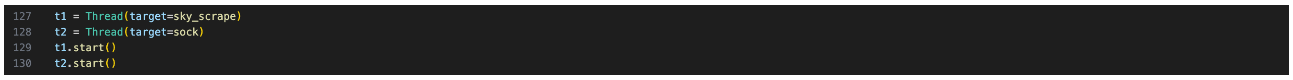


```python
def sock():
    global que

    try:
        ipv4 = None # ipv4 address here
        port = None # Port number here

        f = None
        cli_sock = None

        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sock.bind((ipv4, port))
        print(f"Image sending program has binded to {ipv4}:{port}")

        while True:
            print("Waiting for new connection...")
            sock.listen()

            cli_sock, addr = sock.accept()
            print(f"Connected to {addr[0]}:{addr[1]}")

            filename = que.get()

            if filename == "quit" or filename == "QUIT":
                cli_sock.sendall("QUIT".encode())
                cli_sock.close()
                sock.close()
                break
            else:
                cli_sock.sendall("SEND".encode())

                f = open(filename, "rb")
                l = os.path.getsize(filename)
                m = f.read(l)

                print("Sending image...")
                cli_sock.sendall(m)
                print("Image sent")
                cli_sock.close()

    except Exception as err:
        print(f"[Error] {err.__class__.__name__}: {err}")
        if cli_sock != None:
            cli_sock.close()
        if f != None:
            f.close()
```

Figure 2.4.5: Code for sock() in WebServer.py

2.4.6 Threads

Code that starts the threads for sky_scrape() and sock().

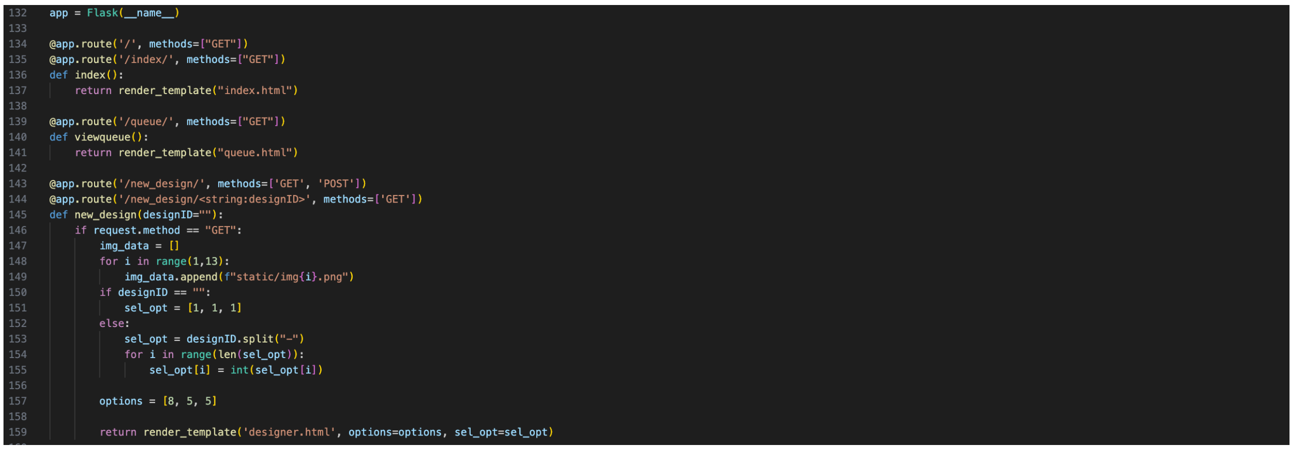


```python
t1 = Thread(target=sky_scrape)
t2 = Thread(target=sock)
t1.start()
t2.start()
```

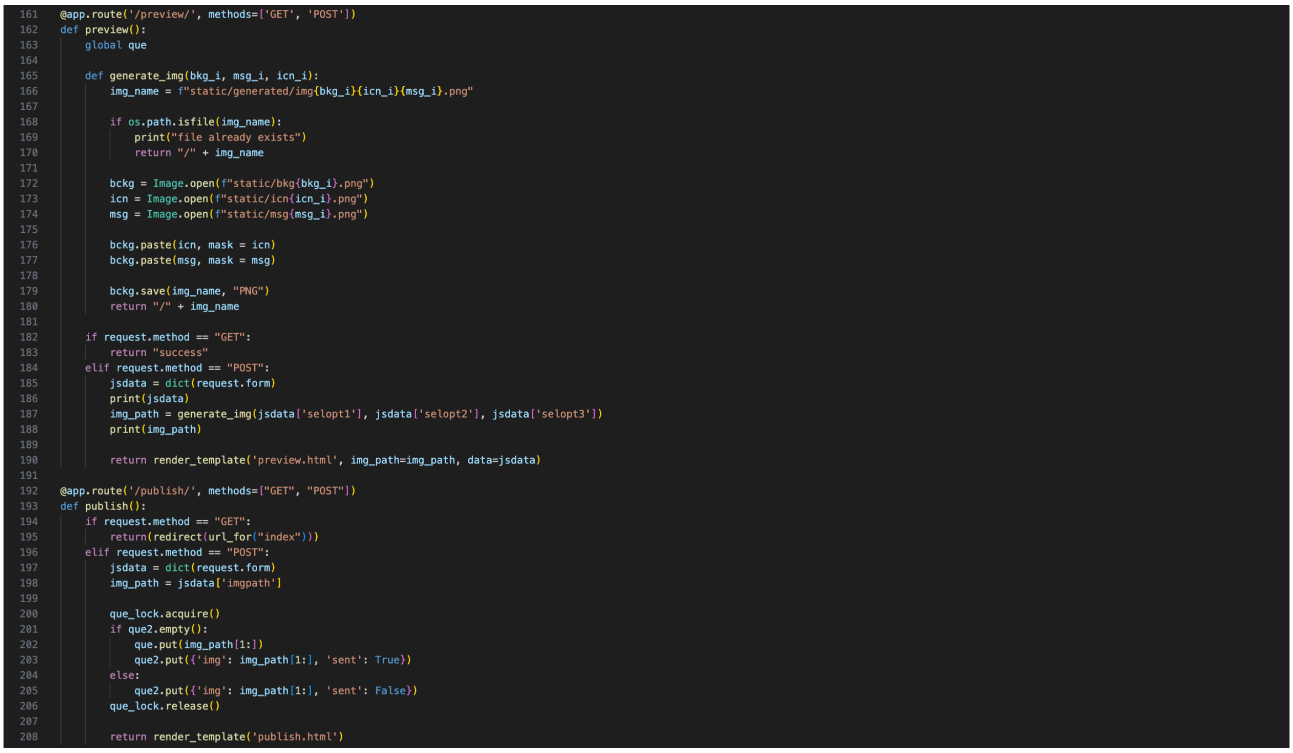
Figure 2.4.6: Code to start threads in WebServer.py

2.4.7 Flask

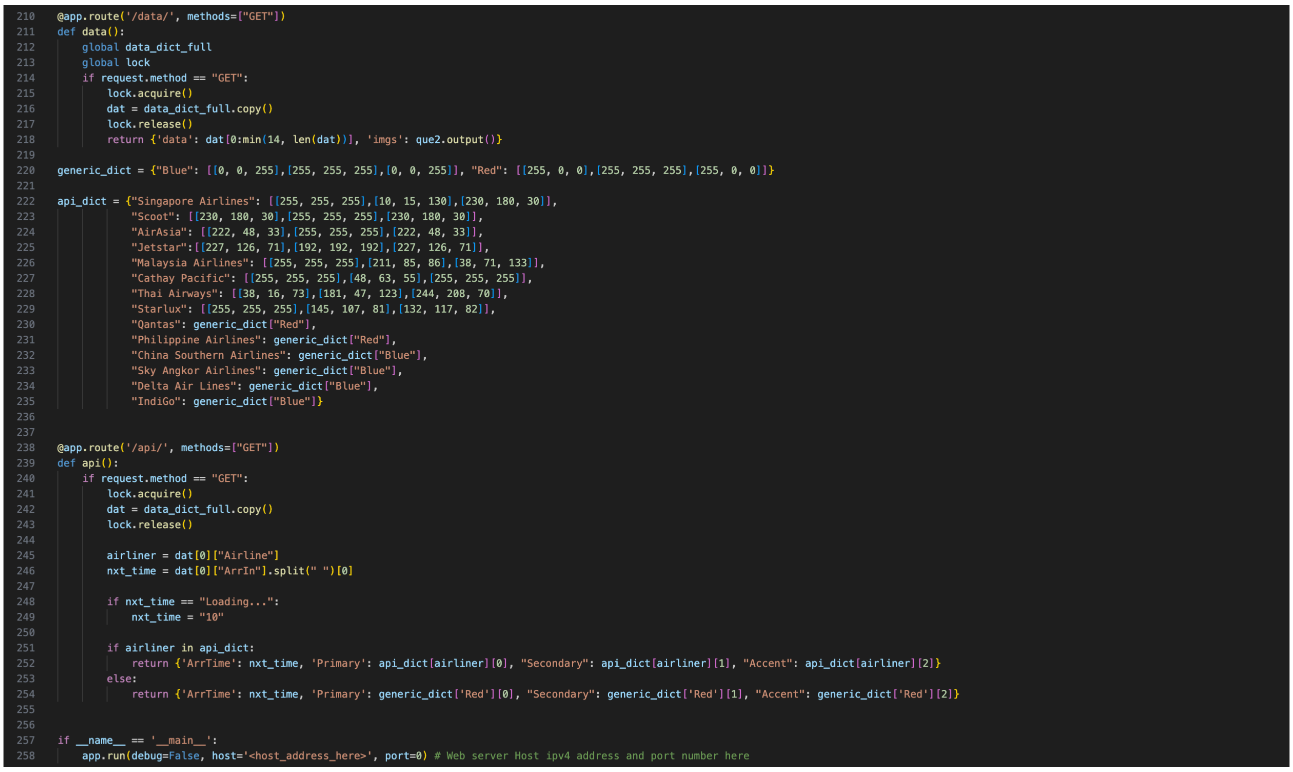
Code that runs the web server. Sends data to web browsers on the Web Clients.



```python
app = Flask(__name__)

@app.route('/', methods=["GET"])
@app.route('/index/', methods=["GET"])
def index():
    return render_template("index.html")

@app.route('/queue/', methods=["GET"])
def viewqueue():
    return render_template("queue.html")

@app.route('/new_design/', methods=['GET', 'POST'])
@app.route('/new_design/<string:designID>', methods=['GET'])
def new_design(designID=""):
    if request.method == "GET":
        img_data = []
        for i in range(1,13):
            img_data.append(f"static/img{i}.png")
        if designID == "":
            sel_opt = [1, 1, 1]
        else:
            sel_opt = designID.split("-")
            for i in range(len(sel_opt)):
                sel_opt[i] = int(sel_opt[i])

        options = [8, 5, 5]

        return render_template('designer.html', options=options, sel_opt=sel_opt)
```



```python
@app.route('/preview/', methods=['GET', 'POST'])
def preview():
    global que

    def generate_img(bkg_i, msg_i, icn_i):
        img_name = f"static/generated/img{bkg_i}{icn_i}{msg_i}.png"

        if os.path.isfile(img_name):
            print("file already exists")
            return "/" + img_name

        bckg = Image.open(f"static/bkg{bkg_i}.png")
        icn = Image.open(f"static/icn{icn_i}.png")
        msg = Image.open(f"static/msg{msg_i}.png")

        bckg.paste(icn, mask = icn)
        bckg.paste(msg, mask = msg)

        bckg.save(img_name, "PNG")
        return "/" + img_name

    if request.method == "GET":
        return "success"
    elif request.method == "POST":
        jsdata = dict(request.form)
        print(jsdata)
        img_path = generate_img(jsdata['selopt1'], jsdata['selopt2'], jsdata['selopt3'])
        print(img_path)

        return render_template('preview.html', img_path=img_path, data=jsdata)

@app.route('/publish/', methods=["GET", "POST"])
def publish():
    if request.method == "GET":
        return(redirect(url_for("index")))
    elif request.method == "POST":
        jsdata = dict(request.form)
        img_path = jsdata['imgpath']

        que_lock.acquire()
        if que2.empty():
            que.put(img_path[1:])
            que2.put({'img': img_path[1:], 'sent': True})
        else:
            que2.put({'img': img_path[1:], 'sent': False})
        que_lock.release()

        return render_template('publish.html')
```



```python
@app.route('/data/', methods=["GET"])
def data():
    global data_dict_full
    global lock
    if request.method == "GET":
        lock.acquire()
        dat = data_dict_full.copy()
        lock.release()
        return {'data': dat[0:min(14, len(dat))], 'imgs': que2.output()}

generic_dict = {"Blue": [[0, 0, 255],[255, 255, 255],[0, 0, 255]], "Red": [[255, 0, 0],[255, 255, 255],[255, 0, 0]]}

api_dict = {"Singapore Airlines": [[255, 255, 255],[10, 15, 130],[230, 180, 30]],
            "Scoot": [[230, 180, 30],[255, 255, 255],[230, 180, 30]],
            "AirAsia": [[222, 48, 33],[255, 255, 255],[222, 48, 33]],
            "Jetstar":[[227, 126, 71],[192, 192, 192],[227, 126, 71]],
            "Malaysia Airlines": [[255, 255, 255],[211, 85, 86],[38, 71, 133]],
            "Cathay Pacific": [[255, 255, 255],[48, 63, 55],[255, 255, 255]],
            "Thai Airways": [[38, 16, 73],[181, 47, 123],[244, 208, 70]],
            "Starlux": [[255, 255, 255],[145, 107, 81],[132, 117, 82]],
            "Qantas": generic_dict["Red"],
            "Philippine Airlines": generic_dict["Red"],
            "China Southern Airlines": generic_dict["Blue"],
            "Sky Angkor Airlines": generic_dict["Blue"],
            "Delta Air Lines": generic_dict["Blue"],
            "IndiGo": generic_dict["Blue"]}


@app.route('/api/', methods=["GET"])
def api():
    if request.method == "GET":
        lock.acquire()
        dat = data_dict_full.copy()
        lock.release()

        airliner = dat[0]["Airline"]
        nxt_time = dat[0]["ArrIn"].split(" ")[0]

        if nxt_time == "Loading...":
            nxt_time = "10"

        if airliner in api_dict:
            return {'ArrTime': nxt_time, 'Primary': api_dict[airliner][0], "Secondary": api_dict[airliner][1], "Accent": api_dict[airliner][2]}
        else:
            return {'ArrTime': nxt_time, 'Primary': generic_dict['Red'][0], "Secondary": generic_dict['Red'][1], "Accent": generic_dict['Red'][2]}


if __name__ == '__main__':
    app.run(debug=False, host='<host_address_here>', port=0) # Web server Host ipv4 address and port number here
```

Figure 2.4.7: Code to run Flask web server in WebServer.py

1. index() sends the home page of the website.
2. queue() shows the message queue.
3. new_design() sends the webpage for users to customise their message.
4. preview() obtains the user's options and sends the generated message image for preview by the user.
5. publish() puts the generated image in the message queue to be sent to the Graphics Generator.
6. data() allows webpages to fetch real-time plane data to be displayed on the webpages.
7. api() allows the Graphics Generator to fetch required plane data and relays them to the ESP32 at the architectural model.

3.0 Web Client Code

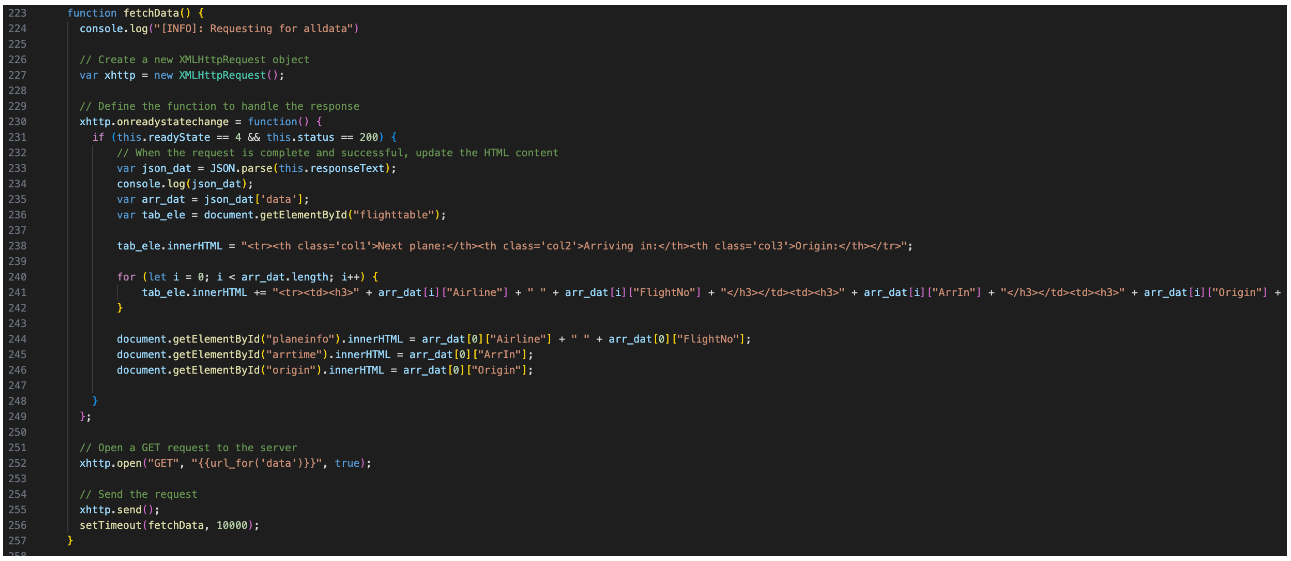
3.1 Overview

The web browser in the Web Client laptop sends requests to the Web Server to obtain the website. Web Server sends the relevant html, css, data and image files to the web browser which are then displayed to the user. The Web Client then sends user inputs to the Web Server for processing.

The coding language used in the webpages is Javascript.

3.2 Fetching data from server

Fetches plane arrival info and message display images from the Web Server.

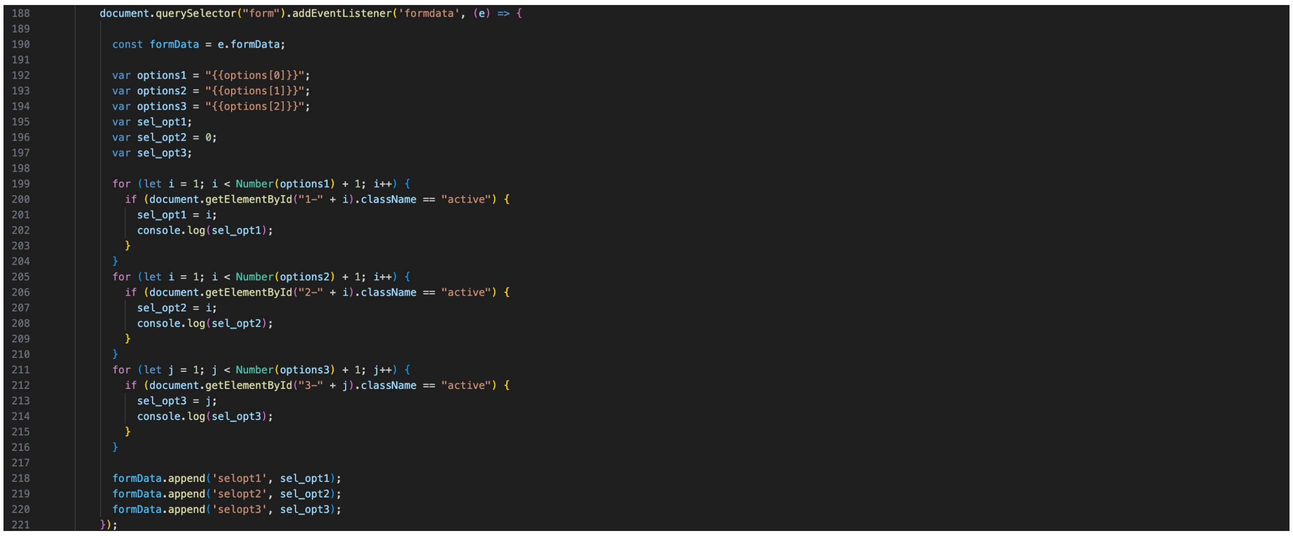


```javascript
function fetchData() {
  console.log("[INFO]: Requesting for alldata")

  // Create a new XMLHttpRequest object
  var xhttp = new XMLHttpRequest();

  // Define the function to handle the response
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
        // When the request is complete and successful, update the HTML content
        var json_dat = JSON.parse(this.responseText);
        console.log(json_dat);
        var arr_dat = json_dat['data'];
        var tab_ele = document.getElementById("flighttable");

        tab_ele.innerHTML = "<tr><th class='col1'>Next plane:</th><th class='col2'>Arriving in:</th><th class='col3'>Origin:</th></tr>";

        for (let i = 0; i < arr_dat.length; i++) {
            tab_ele.innerHTML += "<tr><td><h3>" + arr_dat[i]["Airline"] + " " + arr_dat[i]["FlightNo"] + "</h3></td><td><h3>" + arr_dat[i]["ArrIn"] + "</h3></td><td><h3>" + arr_dat[i]["Origin"] +
        }

        document.getElementById("planeinfo").innerHTML = arr_dat[0]["Airline"] + " " + arr_dat[0]["FlightNo"];
        document.getElementById("arrtime").innerHTML = arr_dat[0]["ArrIn"];
        document.getElementById("origin").innerHTML = arr_dat[0]["Origin"];

    }
  };

  // Open a GET request to the server
  xhttp.open("GET", "{{url_for('data')}}", true);

  // Send the request
  xhttp.send();
  setTimeout(fetchData, 10000);
}
```

Figure 3.2.1: Code to obtain data from Web Server

3.3 Detecting user inputs

Detects the user's options in the webpage for creating the message to be displayed in the interactive screen.

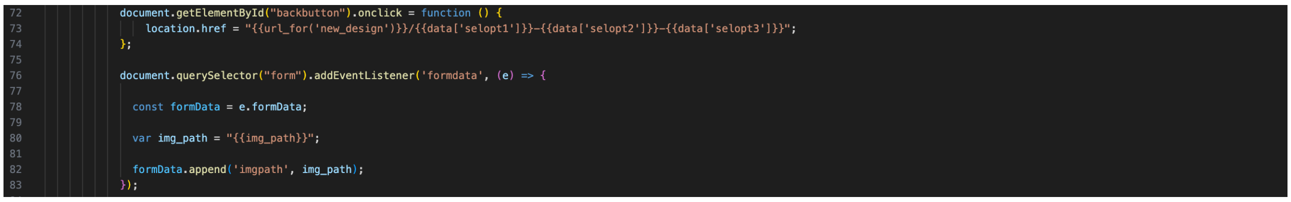


```javascript
document.querySelector("form").addEventListener('formdata', (e) => {

  const formData = e.formData;

  var options1 = "{{options[0]}}";
  var options2 = "{{options[1]}}";
  var options3 = "{{options[2]}}";
  var sel_opt1;
  var sel_opt2 = 0;
  var sel_opt3;

  for (let i = 1; i < Number(options1) + 1; i++) {
    if (document.getElementById("1-" + i).className == "active") {
      sel_opt1 = i;
      console.log(sel_opt1);
    }
  }
  for (let i = 1; i < Number(options2) + 1; i++) {
    if (document.getElementById("2-" + i).className == "active") {
      sel_opt2 = i;
      console.log(sel_opt2);
    }
  }
  for (let j = 1; j < Number(options3) + 1; j++) {
    if (document.getElementById("3-" + j).className == "active") {
      sel_opt3 = j;
      console.log(sel_opt3);
    }
  }

  formData.append('selopt1', sel_opt1);
  formData.append('selopt2', sel_opt2);
  formData.append('selopt3', sel_opt3);
});
```

Figure 3.3.1: Code to obtain user's choice of message

3.4 Sending user inputs

Sends the user’s inputs to the Web Server.



```
document.getElementById("backbutton").onclick = function () {
    location.href = "{{url_for('new_design')}}/{{data['selopt1']}}-{{data['selopt2']}}-{{data['selopt3']}}";
};

document.querySelector("form").addEventListener('formdata', (e) => {

  const formData = e.formData;

  var img_path = "{{img_path}}";

  formData.append('imgpath', img_path);
});
```

Figure 3.4.1: Code to send user’s inputs to the Web Server

4.0 ESP32 Code

4.1 Overview and File Structure

The ESP32s handles the electronic elements of the project. The ESP32s used also converts analog data from sensors of the project into digital data to be sent to the Graphics Generator, and vice versa.

File Structure:

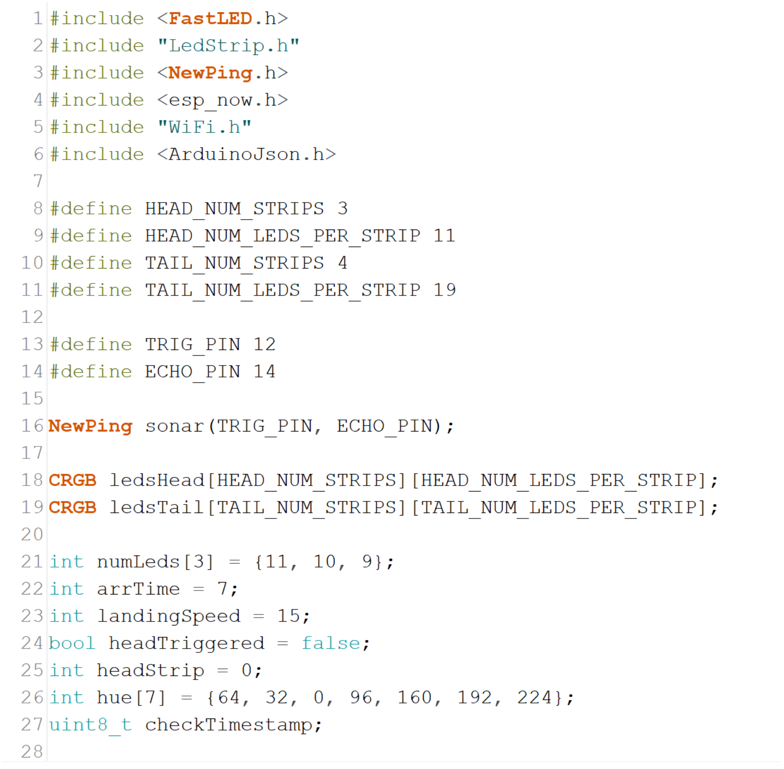
```
/ESP32
        /Pavilion
                Pavilion.ino
                LedStrip.h
        /Interactive_Display
                Interactive_Display.ino
                LedStrip.h
                PressurePlate.h
        /Relay_ESP
                Relay_ESP.ino
```

4.2 Pavilion ESP32 Code

4.2.1 Overview

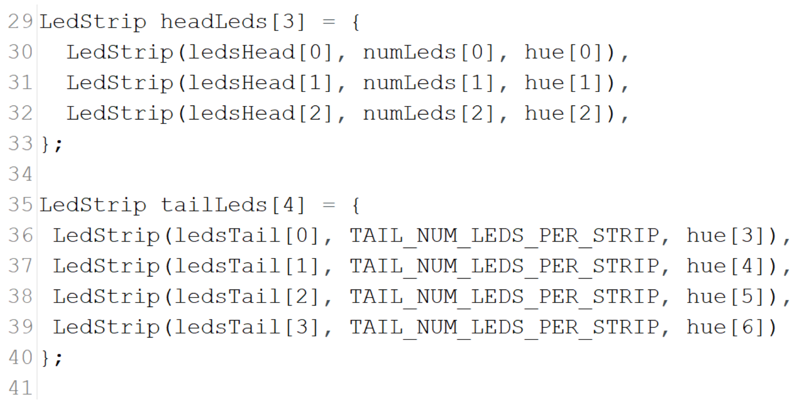
Pavilion.ino: Receives flight data from the Relay ESP and control the tail pavilion and head pavilion lights based on the data
LedStrip.h (Complete Version): Contains LedStrip class that contains the logic to turn on the led strip with different patterns

4.2.2 Pavilion.ino



```
#include <FastLED.h>
#include "LedStrip.h"
#include <NewPing.h>
#include <esp_now.h>
#include "WiFi.h"
#include <ArduinoJson.h>

#define HEAD_NUM_STRIPS 3
#define HEAD_NUM_LEDS_PER_STRIP 11
#define TAIL_NUM_STRIPS 4
#define TAIL_NUM_LEDS_PER_STRIP 19

#define TRIG_PIN 12
#define ECHO_PIN 14

NewPing sonar(TRIG_PIN, ECHO_PIN);

CRGB ledsHead[HEAD_NUM_STRIPS][HEAD_NUM_LEDS_PER_STRIP];
CRGB ledsTail[TAIL_NUM_STRIPS][TAIL_NUM_LEDS_PER_STRIP];

int numLeds[3] = {11, 10, 9};
int arrTime = 7;
int landingSpeed = 15;
bool headTriggered = false;
int headStrip = 0;
int hue[7] = {64, 32, 0, 96, 160, 192, 224};
uint8_t checkTimestamp;

```

Figure 4.2.1: Libraries, declaration of constants and variables



```
LedStrip headLeds[3] = {
  LedStrip(ledsHead[0], numLeds[0], hue[0]),
  LedStrip(ledsHead[1], numLeds[1], hue[1]),
  LedStrip(ledsHead[2], numLeds[2], hue[2]),
};

LedStrip tailLeds[4] = {
 LedStrip(ledsTail[0], TAIL_NUM_LEDS_PER_STRIP, hue[3]),
 LedStrip(ledsTail[1], TAIL_NUM_LEDS_PER_STRIP, hue[4]),
 LedStrip(ledsTail[2], TAIL_NUM_LEDS_PER_STRIP, hue[5]),
 LedStrip(ledsTail[3], TAIL_NUM_LEDS_PER_STRIP, hue[6])
};

```

Figure 4.2.2: Creation of array containing LedStrips objects

```
// Receiving Data
JsonDocument doc;
const int BUFFER_SIZE = 100;

typedef struct struct_message {
  char json[BUFFER_SIZE];
} struct_message;

struct_message myData;

void setupESPNow() {
  WiFi.mode(WIFI_STA);

  if (esp_now_init() != ESP_OK) {
    Serial.println("Error initializing ESP-NOW");
    return;
  }

  esp_now_register_recv_cb(OnDataRecv);
}
```

Figure 4.2.3: Setup to allow ESP32 to receive data from the relay ESP32

```

void OnDataRecv(const uint8_t * mac, const uint8_t *incomingData, int len) {
  memcpy(&myData, incomingData, sizeof(myData));
  deserializeJson(doc, myData.json);

  arrTime = doc["ArrTime"];
  headLeds[0].updateRGB(
      doc["Primary"][0],
      doc["Primary"][1],
      doc["Primary"][2]
  );
  headLeds[1].updateRGB(
      doc["Secondary"][0],
      doc["Secondary"][1],
      doc["Secondary"][2]
  );
  headLeds[2].updateRGB(
      doc["Accent"][0],
      doc["Accent"][1],
      doc["Accent"][2]
  );
}

```

Figure 4.2.4: Function to unpack JSON data received and update the respective variables and object attributes

```
void setup() {
  Serial.begin(115200);
  pinMode(TRIG_PIN, OUTPUT);
  pinMode(ECHO_PIN, INPUT);

  FastLED.addLeds<WS2812B, 27>(ledsHead[0], numLeds[0]);
  FastLED.addLeds<WS2812B, 26>(ledsHead[1], numLeds[1]);
  FastLED.addLeds<WS2812B, 25>(ledsHead[2], numLeds[2]);

  FastLED.addLeds<WS2812B, 21>(ledsTail[0], TAIL_NUM_LEDS_PER_STRIP);
  FastLED.addLeds<WS2812B, 19>(ledsTail[1], TAIL_NUM_LEDS_PER_STRIP);
  FastLED.addLeds<WS2812B, 18>(ledsTail[2], TAIL_NUM_LEDS_PER_STRIP);
  FastLED.addLeds<WS2812B, 5>(ledsTail[2], TAIL_NUM_LEDS_PER_STRIP);

  setupESPNow();

  checkLEDs();
}
```

Figure 4.2.5: Setup function; will run once when ESP32 is first powered on

```
void loop() {
  // Uncomment the below to simulate the flight data changing
  /*
  EVERY_N_SECONDS(5) {
    if (arrTime <= 0) {
      arrTime = 5;
      planeCounter++;
      testChangePlane();
    } else {
      arrTime--;
    }
  }
  */
  pulseTail();
  glowHead();
}
```

Figure 4.2.6: Loop function; will continuously call the pulseTail() and glowHead() functions after the setup function is executed

```

void checkLEDs() {
  // Pulse to check LED Strips
  checkTimestamp = millis();

  for(int i = 0; i < HEAD_NUM_STRIPS; i++) {
    headLeds[i].startPulse();
  }
  for(int i = 0; i < TAIL_NUM_STRIPS; i++) {
    tailLeds[i].startPulse();
  }

  while (millis() - checkTimestamp < 2000) {
    for(int i = 0; i < HEAD_NUM_STRIPS; i++) {
      if (!headLeds[i].pulseIsFinished()) {
        headLeds[i].pulseDuration(1000);
        fadeToBlackBy(ledsHead[i], numLeds[i], 2);
      }
    }
    for(int i = 0; i < TAIL_NUM_STRIPS; i++) {
      if (!tailLeds[i].pulseIsFinished()) {
        tailLeds[i].pulseDuration(1000);
        fadeToBlackBy(ledsTail[i], TAIL_NUM_LEDS_PER_STRIP, 2);
      }
    }
    FastLED.show();
  }

  // Turn off all LEDS
  for(int i = 0; i < HEAD_NUM_STRIPS; i++) {
    headLeds[i].off();
  }
  for(int i = 0; i < TAIL_NUM_STRIPS; i++) {
    tailLeds[i].off();
  }
  FastLED.show();
}
```

Figure 4.2.7: Causes all 7 pavilion led strips to pulse once in different colours, used as a test to ensure all is in working condition

```

void pulseTail() {
  switch(arrTime) {
    case 0:
      for(int i = 0; i < TAIL_NUM_STRIPS; i++) {
        tailLeds[i].updateRGB(27, 141, 252);
      }
      break;
    case 1:
    case 2:
      for(int i = 0; i < TAIL_NUM_STRIPS; i++) {
        tailLeds[i].updateRGB(71, 134, 229);
      }
      break;
    case 3:
    case 4:
      for(int i = 0; i < TAIL_NUM_STRIPS; i++) {
        tailLeds[i].updateRGB(120, 129, 213);
      }
      break;
    default:
      for(int i = 0; i < TAIL_NUM_STRIPS; i++) {
        tailLeds[i].updateRGB(124, 124, 140);
      }
  }

  for(int i = 0; i < TAIL_NUM_STRIPS; i++) {
    if(tailLeds[i].pulseIsFinished()) {
      tailLeds[i].startPulse();
    } else {
      tailLeds[i].pulseSpeed(landingSpeed + 15 * arrTime);
      fadeToBlackBy(ledsTail[i], TAIL_NUM_LEDS_PER_STRIP, 2);
    }
  }
  FastLED.show();
}
```

Figure 4.2.8: Causes the infinity mirror led strips to pulse with increasing speed as the plane gets closer to the site and also change colours in a white to blue gradient

```
void glowHead() {
  if (sonar.ping_cm() <= 3 && !headTriggered) {
    headTriggered = true;
    headStrip = 0;
    headLeds[headStrip].glowStart();
  }

  if (headStrip >= HEAD_NUM_STRIPS) {
    headTriggered = false;
  }

  if (headTriggered) {
    if(!headLeds[headStrip].glowIsFinished()) {
      headLeds[headStrip].glowDuration();
    } else {
      headLeds[headStrip].off();
      headStrip++;
      headLeds[headStrip].glowStart();
    }
  }
  FastLED.show();
}

```

Figure 4.2.9: Causes the head led strips to successively glow in the colours of the landing plane's livery when the ultrasonic sensor is triggered

4.2.3 LedStrip.h

```
#include <FastLED.h>

class LedStrip {
  private:
    CRGB* _leds;
    int _numLeds;
    uint8_t _hue;
    int _lastLed;
    unsigned long _lastOn = 0;
    int _red = 255;
    int _green = 255;
    int _blue = 255;
    float _brightness = 0;
    int _glowDuration = 1000;


  public:
    LedStrip(CRGB* leds, int numLeds, int hue) {
      _leds = leds;
      _numLeds = numLeds;
      _hue = hue;
    }

```

Figure 4.2.10: Declaration of LedStrip class attributes and object constructor

```

  void updateRGB(int red, int green, int blue) {
    _red = red;
    _green = green;
    _blue = blue;
  }

  void on() {
    for(int i = 0; i < _numLeds; i++) {
      _leds[i].setRGB(_red, _green, _blue);
    }
  }

  void singleColour(int colour) {
    for(int i = 0; i < _numLeds; i++) {
      _leds[i] = CHSV(colour, 255, 255);
    }
  }

  void off() {
    for(int i = 0; i <_numLeds; i++) {
      _leds[i] = CRGB::Black;
    }
  }

```

Figure 4.2.11: Functions to i) update the stored RGB values, ii) turn on the entire strip in the stored RGB values, iii) turn on the entire strip in a specified hue, iv) turn off the entire strip

```
  void startPulse() {
    _lastLed = 0;
  }

  void pulseDuration(int duration) {
    if ((millis() - _lastOn) >= (duration / _numLeds)) {
      _leds[_lastLed] = CHSV(_hue, 255, 255);
      _lastLed++;
      _lastOn = millis();
    }
  }

  void pulseSpeed(int timeDelay) {
    if ((millis() - _lastOn) >= timeDelay) {
      _leds[_lastLed].setRGB(_red, _green, _blue);
      _lastLed++;
      _lastOn = millis();
    }
  }

  bool pulseIsFinished() {
    if (_lastLed >= _numLeds) {
      return true;
    } else {
      return false;
    }
  }
```

Figure 4.2.12: Functions to i) reset the led strip to start a new pulse, ii) cause the strip to pulse with a specified duration per pulse, iii) cause the strip to pulse with a specified time delay between successive led lights turning on, iv) check whether the pulse has reached the end of the led strip

```
  
  void glowStart(int duration = 1500) {
    _lastOn = millis();
    _glowDuration = duration;
  }

  void glowDuration() {
    _brightness = (cos((millis() - float(_lastOn)) / float(_glowDuration) * 6.28) + 1.0) * 127.5;
    for(int i = 0; i < _numLeds; i++) {
       _leds[i].setRGB(_red, _green, _blue);
       _leds[i].maximizeBrightness();
       _leds[i].fadeLightBy(_brightness);
    }
  }

  bool glowIsFinished() {
    if (millis() - _lastOn >= _glowDuration) {
      _brightness = 0;
      return true;
    } else {
      return false;
    }
  }
```

Figure 4.2.13: Functions to i) reset the led strip to start to glow and specify the duration, ii) cause the led strip to gently glow, with the brightness following a negative cosine wave, iii) check whether the glow sequence has finished

4.3 Interactive Display ESP32 Code

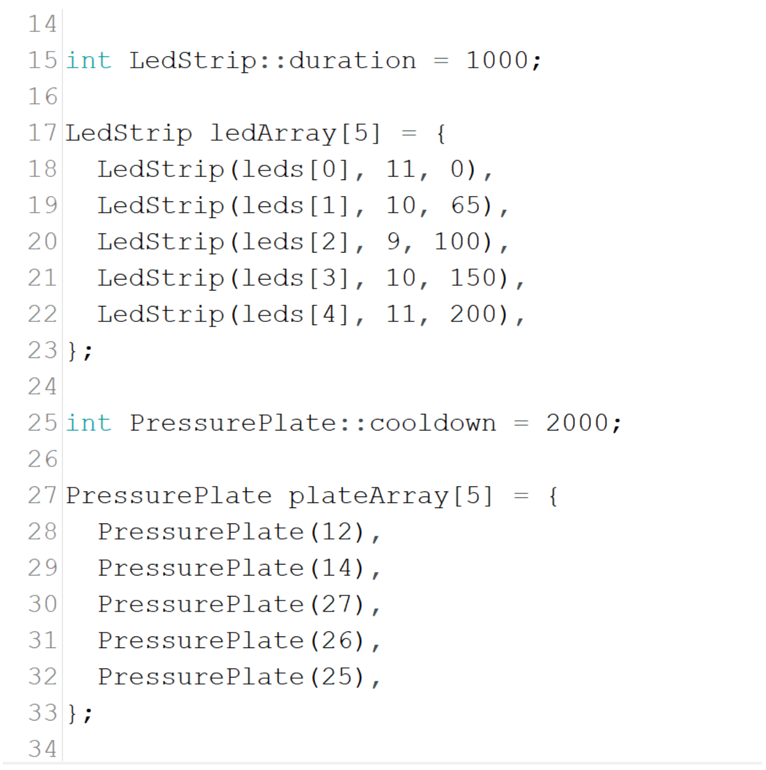
4.3.1 Overview

Interactive_Display.ino: Detects the button pressed and lights up the corresponding led strip and send button number to the Relay ESP
LedStrip.h (Simplified Version): Contains the LedStrip class that contains the logic to turn on the led strip with different patterns
PressurePlate.h: Contains the PressurePlate class that contains the logic to detect the button press

4.3.2 Interactive_Display.ino

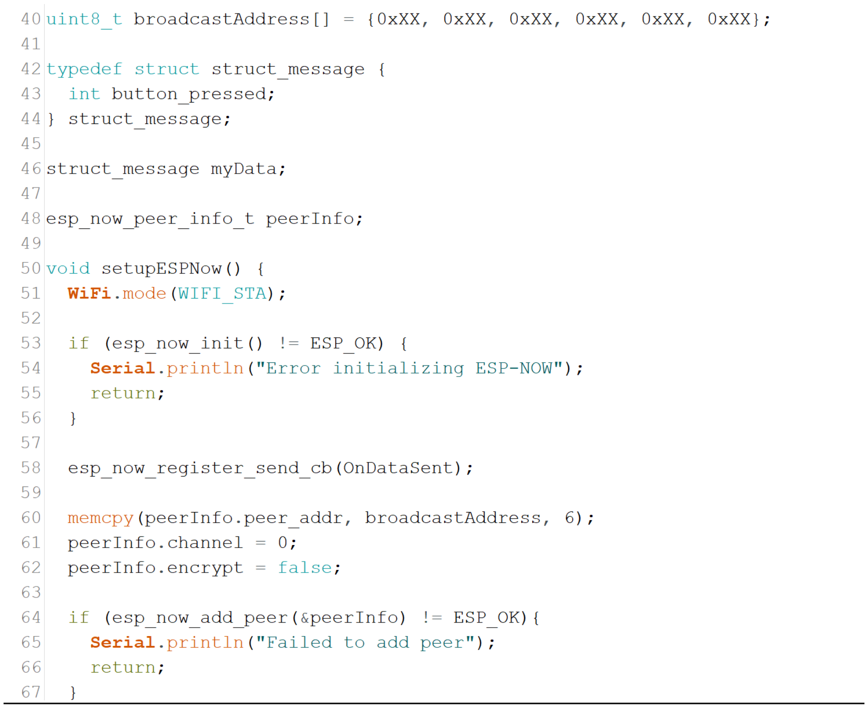
```
#include <FastLED.h>
#include <esp_now.h>
#include <WiFi.h>
#include "PressurePlate.h"
#include "LedStrip.h"

#define NUM_STRIPS 5
#define NUM_LEDS_PER_STRIP 11

CRGB leds[NUM_STRIPS][NUM_LEDS_PER_STRIP];

int numLeds[5] = {11, 10, 9, 10, 11};
int fadeSpeed[5] = {4, 2, 2, 3, 3};
```

Figure 4.3.1: Libraries, declaration of constants and variables



```cpp

int LedStrip::duration = 1000;

LedStrip ledArray[5] = {
  LedStrip(leds[0], 11, 0),
  LedStrip(leds[1], 10, 65),
  LedStrip(leds[2], 9, 100),
  LedStrip(leds[3], 10, 150),
  LedStrip(leds[4], 11, 200),
};

int PressurePlate::cooldown = 2000;

PressurePlate plateArray[5] = {
  PressurePlate(12),
  PressurePlate(14),
  PressurePlate(27),
  PressurePlate(26),
  PressurePlate(25),
};

```

Figure 4.3.2: Declaration of LedStrip and PressurePlate class constants and creation of arrays containing LedStrip objects and PressurePlate objects respectively



```cpp
uint8_t broadcastAddress[] = {0xXX, 0xXX, 0xXX, 0xXX, 0xXX, 0xXX};

typedef struct struct_message {
  int button_pressed;
} struct_message;

struct_message myData;

esp_now_peer_info_t peerInfo;

void setupESPNow() {
  WiFi.mode(WIFI_STA);

  if (esp_now_init() != ESP_OK) {
    Serial.println("Error initializing ESP-NOW");
    return;
  }

  esp_now_register_send_cb(OnDataSent);

  memcpy(peerInfo.peer_addr, broadcastAddress, 6);
  peerInfo.channel = 0;
  peerInfo.encrypt = false;

  if (esp_now_add_peer(&peerInfo) != ESP_OK){
    Serial.println("Failed to add peer");
    return;
  }
```

Figure 4.3.3: Setup to allow ESP32 to send button press data from the relay ESP32

```

void setup() {
  Serial.begin(115200);

  FastLED.addLeds<WS2812B, 4>(leds[0], numLeds[0]);
  FastLED.addLeds<WS2812B, 5>(leds[1], numLeds[1]);
  FastLED.addLeds<WS2812B, 18>(leds[2], numLeds[2]);
  FastLED.addLeds<WS2812B, 19>(leds[3], numLeds[3]);
  FastLED.addLeds<WS2812B, 21>(leds[4], numLeds[4]);

  setupESPNow();
}

```

Figure 4.3.4: Setup function; will run once when ESP32 is first powered on

```
void loop() {
  for(int i = 0; i < NUM_STRIPS; i++) {
    if (plateArray[i].isPressed()) {
      // Send message via ESP-NOW
      myData.button_pressed = i + 1;
      esp_err_t result = esp_now_send(broadcastAddress, (uint8_t *) &myData, sizeof(myData));
      if (result == ESP_OK) {
        Serial.println("Sent with success");
      }
      else {
        Serial.println("Error sending the data");
      }

      ledArray[i].start();
    }

    if (!ledArray[i].isFinished()) {
      ledArray[i].pulse();
    }
    fadeToBlackBy(leds[i], numLeds[i], fadeSpeed[i]);
  }
  FastLED.show();
}
```

Figure 4.3.5: Loop function; will continuously check whether a button is pressed, and if so, will send the data to the relay ESP32 and cause the corresponding led strip to pulse once

4.3.3 LedStrip.h (Simplified Version)

```
#include <FastLED.h>

class LedStrip {
  private:
    CRGB* _leds;
    int _numLeds;
    uint8_t _hue;
    int _lastLed;
    unsigned long _lastOn = 0;

  public:
    static int duration;
    LedStrip(CRGB* leds, int numLeds, int hue) {
      _leds = leds;
      _numLeds = numLeds;
      _hue = hue;
    }

```

Figure 4.3.6: Declaration of LedStrip class attributes and object constructor

```

  void start() {
    _lastLed = 0;
  }

  void pulse() {
    if ((millis() - _lastOn) >= (duration / _numLeds)) {
      _leds[_lastLed] = CHSV(_hue, 255, 255);
      _lastLed++;
      _lastOn = millis();
    }
  }

  bool isFinished() {
    if (_lastLed >= _numLeds) {
      return true;
    } else {
      return false;
    }
  }
```

Figure 4.3.7: Functions to i) reset the led strip to start a new pulse, ii) cause the strip to pulse with the hue specified during the initialisation of the object, iii) check whether the pulse has reached the end of the led strip

4.3.4 PressurePlate.h

```
class PressurePlate {
  private:
    int _pin;
    unsigned long _lastPressed = 0;

  public:
    static int cooldown;
    PressurePlate(int pin) {
      _pin = pin;
      init();
    }

    void init() {
      pinMode(_pin, INPUT_PULLUP);
    }

    bool isPressed() {
      if (digitalRead(_pin) == LOW) {
        if ((millis() - _lastPressed) >= cooldown) {
          _lastPressed = millis();
          Serial.print("Button ");
          Serial.print(_pin);
          Serial.println(" Pressed!");
          return true;
        }
      }
      return false;
    }
```

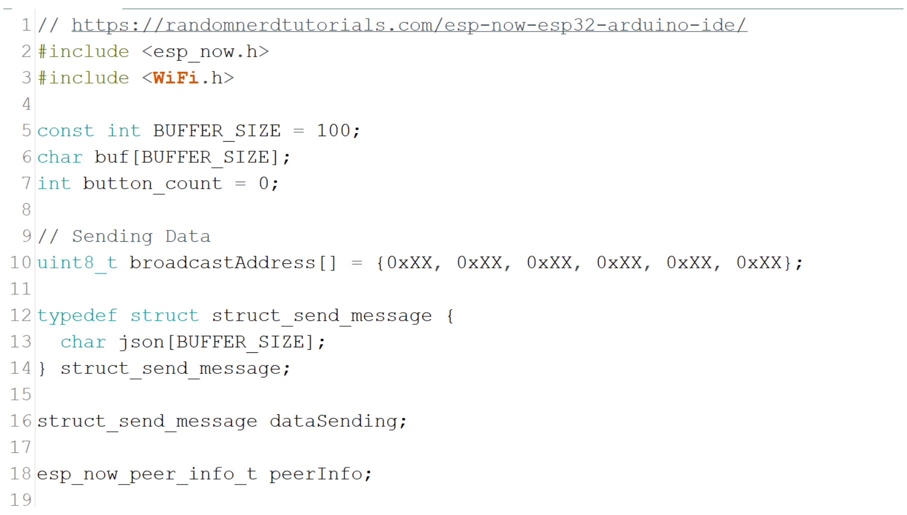
Figure 4.3.8: Declaration of LedStrip class attributes and object constructor and function to check whether the button is pressed with a cooldown system

4.4 Relay ESP32 Code

4.4.1 Overview

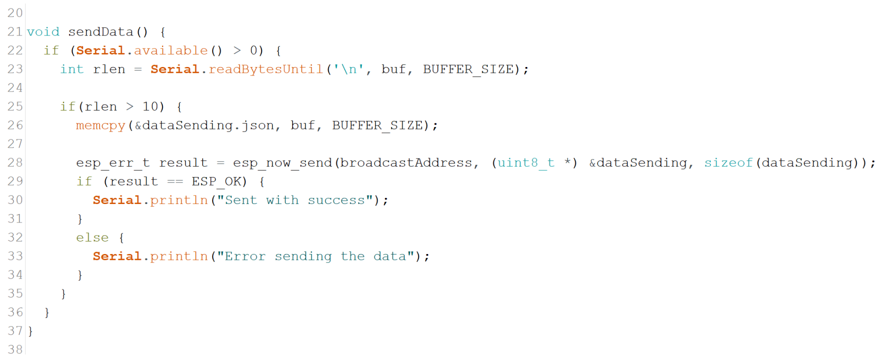
Used as a relay to send data between the Pavilion ESP and Interactive Display ESP and the Graphics Generator laptop used to project the display and request data from the web server.

4.4.2 Relay_ESP.ino



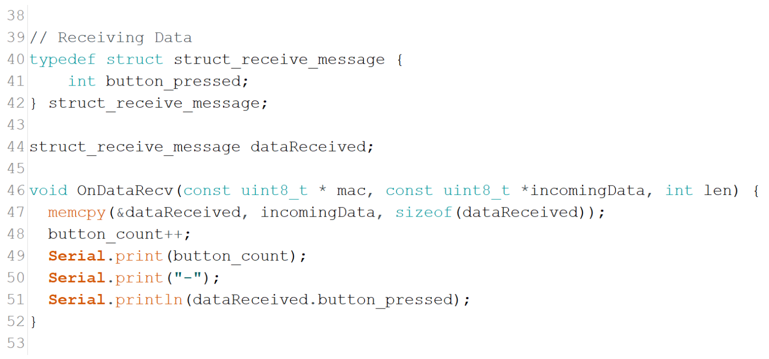
```
// https://randomnerdtutorials.com/esp-now-esp32-arduino-ide/
#include <esp_now.h>
#include <WiFi.h>

const int BUFFER_SIZE = 100;
char buf[BUFFER_SIZE];
int button_count = 0;

// Sending Data
uint8_t broadcastAddress[] = {0xXX, 0xXX, 0xXX, 0xXX, 0xXX, 0xXX};

typedef struct struct_send_message {
  char json[BUFFER_SIZE];
} struct_send_message;

struct_send_message dataSending;

esp_now_peer_info_t peerInfo;

```

Figure 4.4.1: Libraries and variable used when sending data to the Pavilion ESP32



```

void sendData() {
  if (Serial.available() > 0) {
    int rlen = Serial.readBytesUntil('\n', buf, BUFFER_SIZE);

    if(rlen > 10) {
      memcpy(&dataSending.json, buf, BUFFER_SIZE);

      esp_err_t result = esp_now_send(broadcastAddress, (uint8_t *) &dataSending, sizeof(dataSending));
      if (result == ESP_OK) {
        Serial.println("Sent with success");
      }
      else {
        Serial.println("Error sending the data");
      }
    }
  }
}

```

Figure 4.4.2: Function to read the flight data from the serial port and send the JSON data to the Pavilion ESP32



```

// Receiving Data
typedef struct struct_receive_message {
    int button_pressed;
} struct_receive_message;

struct_receive_message dataReceived;

void OnDataRecv(const uint8_t * mac, const uint8_t *incomingData, int len) {
  memcpy(&dataReceived, incomingData, sizeof(dataReceived));
  button_count++;
  Serial.print(button_count);
  Serial.print("-");
  Serial.println(dataReceived.button_pressed);
}

```

Figure 4.4.3: Allows the ESP32 to receive data and prints out the pressed button's number to the serial port to be read by the connected computer

```

void setupESPNow() {
  WiFi.mode(WIFI_STA);

  if (esp_now_init() != ESP_OK) {
    Serial.println("Error initializing ESP-NOW");
    return;
  }

  memcpy(peerInfo.peer_addr, broadcastAddress, 6);
  peerInfo.channel = 0;
  peerInfo.encrypt = false;

  if (esp_now_add_peer(&peerInfo) != ESP_OK){
    Serial.println("Failed to add peer");
    return;
  }

  esp_now_register_recv_cb(OnDataRecv);
}

```

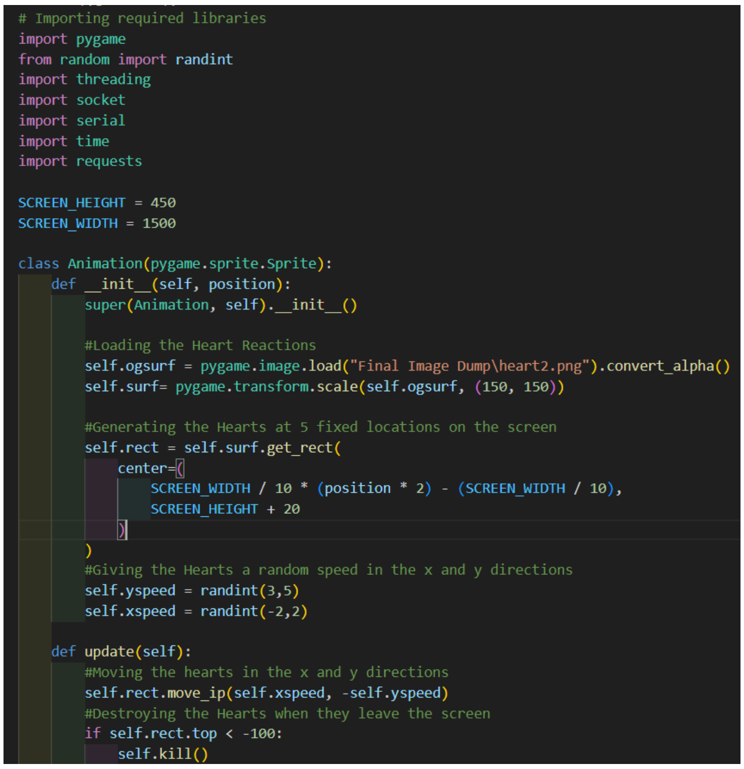
Figure 4.4.4: Setup to allow the ESP32 to send and receive data

5.0 Graphics Generator Code

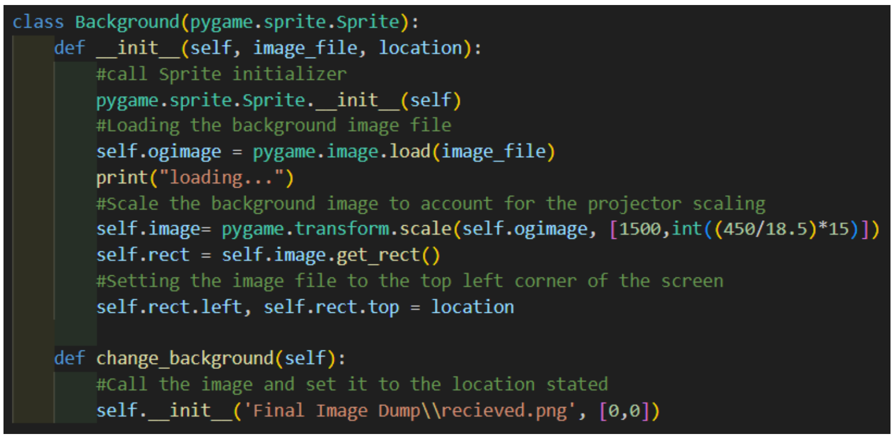
5.1 Overview

The Graphics Generator obtains the message image from the Web Server and displays it on the projector screen, together with reactions generated from input data from the Interactive Display ESP32. The Graphics Generator also acts as a relay to transfer flight data from the Web Server to the Pavilion ESP32.

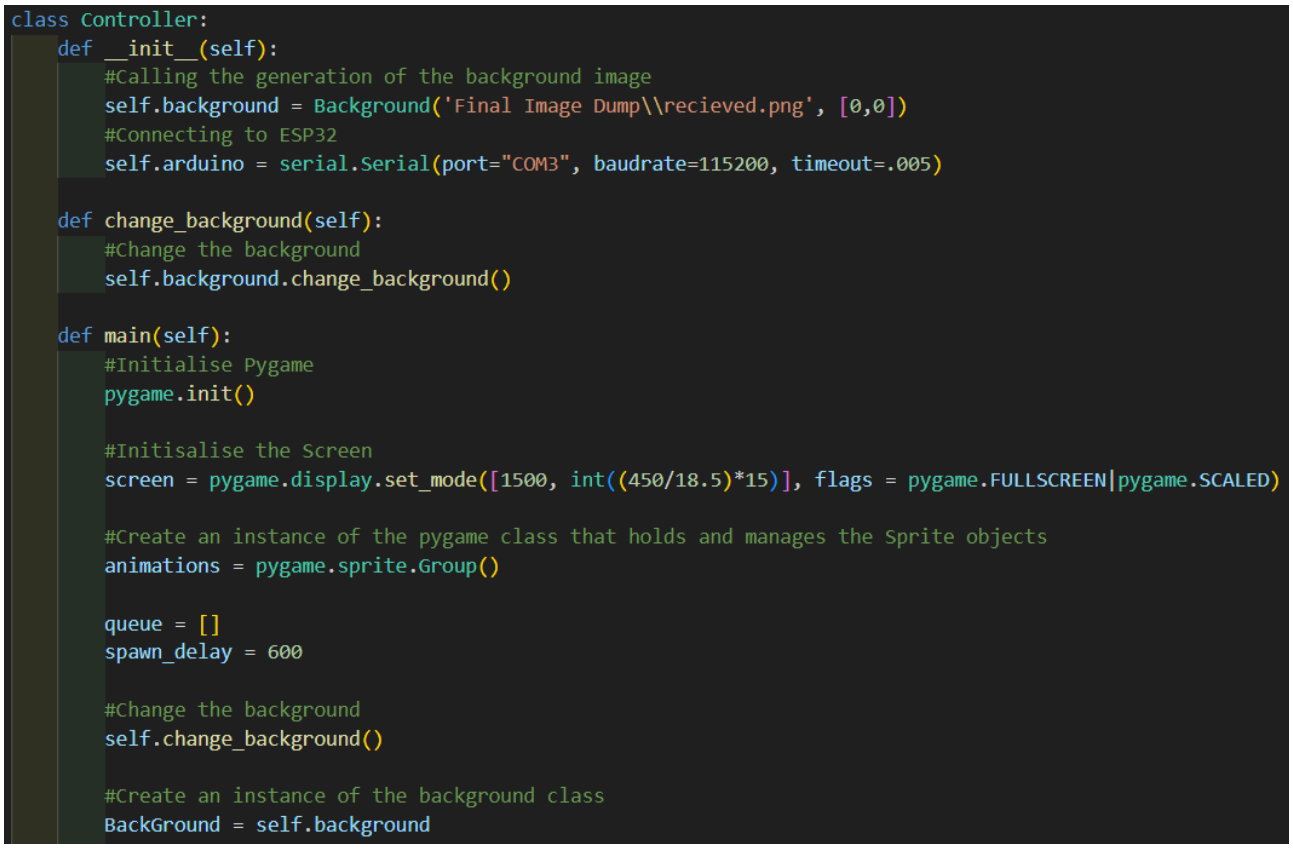
5.2 Code



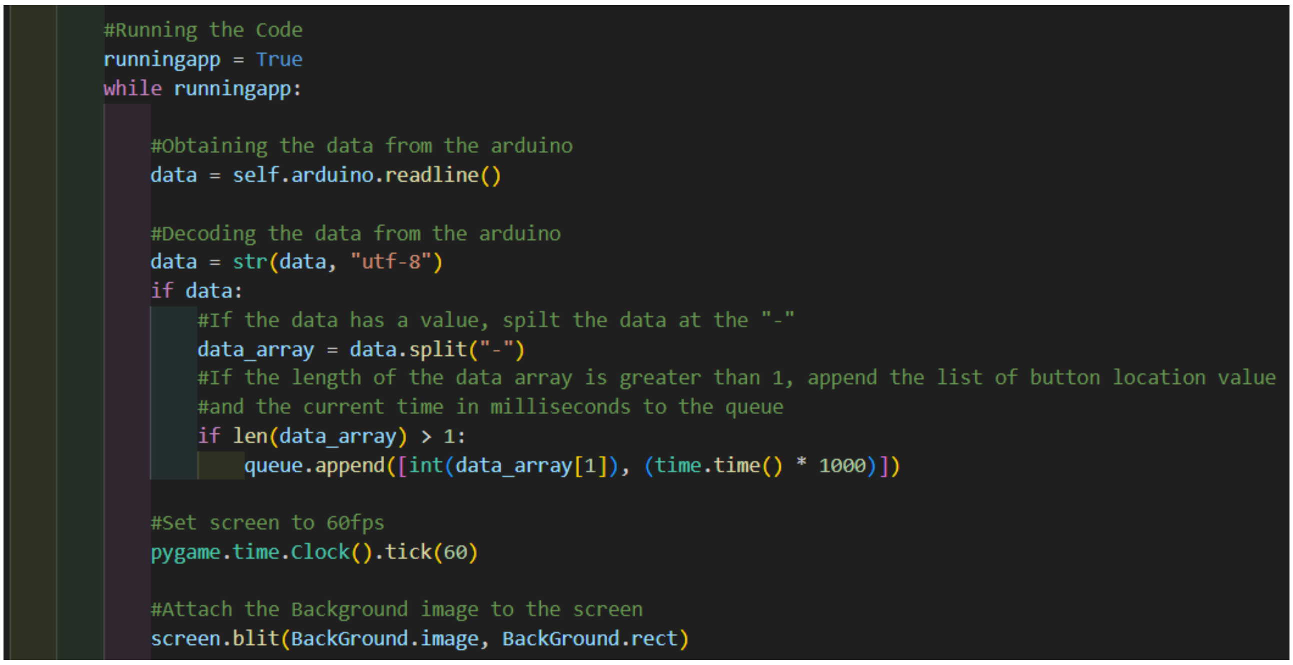
```python
# Importing required libraries
import pygame
from random import randint
import threading
import socket
import serial
import time
import requests

SCREEN_HEIGHT = 450
SCREEN_WIDTH = 1500

class Animation(pygame.sprite.Sprite):
    def __init__(self, position):
        super(Animation, self).__init__()

        #Loading the Heart Reactions
        self.ogsurf = pygame.image.load("Final Image Dump\heart2.png").convert_alpha()
        self.surf= pygame.transform.scale(self.ogsurf, (150, 150))

        #Generating the Hearts at 5 fixed locations on the screen
        self.rect = self.surf.get_rect(
            center=(
                SCREEN_WIDTH / 10 * (position * 2) - (SCREEN_WIDTH / 10),
                SCREEN_HEIGHT + 20
            )
        )
        #Giving the Hearts a random speed in the x and y directions
        self.yspeed = randint(3,5)
        self.xspeed = randint(-2,2)

    def update(self):
        #Moving the hearts in the x and y directions
        self.rect.move_ip(self.xspeed, -self.yspeed)
        #Destroying the Hearts when they leave the screen
        if self.rect.top < -100:
            self.kill()
```



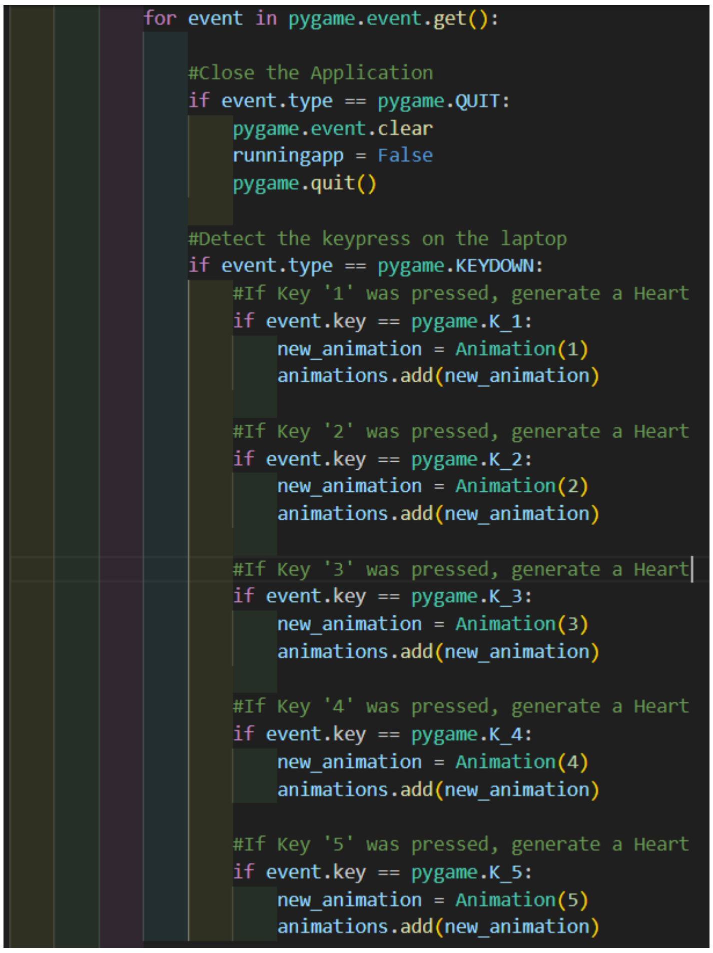
```python
class Background(pygame.sprite.Sprite):
    def __init__(self, image_file, location):
        #call Sprite initializer
        pygame.sprite.Sprite.__init__(self)
        #Loading the background image file
        self.ogimage = pygame.image.load(image_file)
        print("loading...")
        #Scale the background image to account for the projector scaling
        self.image= pygame.transform.scale(self.ogimage, [1500,int((450/18.5)*15)])
        self.rect = self.image.get_rect()
        #Setting the image file to the top left corner of the screen
        self.rect.left, self.rect.top = location

    def change_background(self):
        #Call the image and set it to the location stated
        self.__init__('Final Image Dump\\recieved.png', [0,0])
```



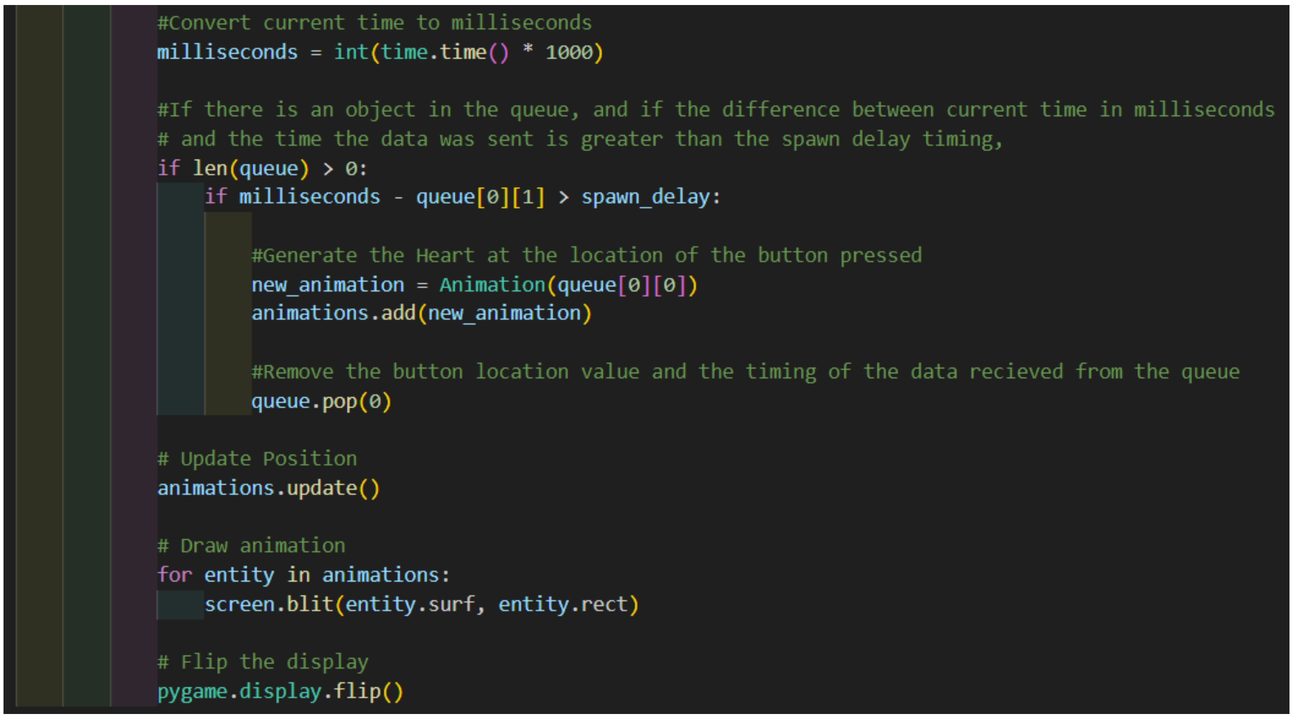
```python
class Controller:
    def __init__(self):
        #Calling the generation of the background image
        self.background = Background('Final Image Dump\\recieved.png', [0,0])
        #Connecting to ESP32
        self.arduino = serial.Serial(port="COM3", baudrate=115200, timeout=.005)

    def change_background(self):
        #Change the background
        self.background.change_background()

    def main(self):
        #Initialise Pygame
        pygame.init()

        #Initisalise the Screen
        screen = pygame.display.set_mode([1500, int((450/18.5)*15)], flags = pygame.FULLSCREEN|pygame.SCALED)

        #Create an instance of the pygame class that holds and manages the Sprite objects
        animations = pygame.sprite.Group()

        queue = []
        spawn_delay = 600

        #Change the background
        self.change_background()

        #Create an instance of the background class
        BackGround = self.background
```



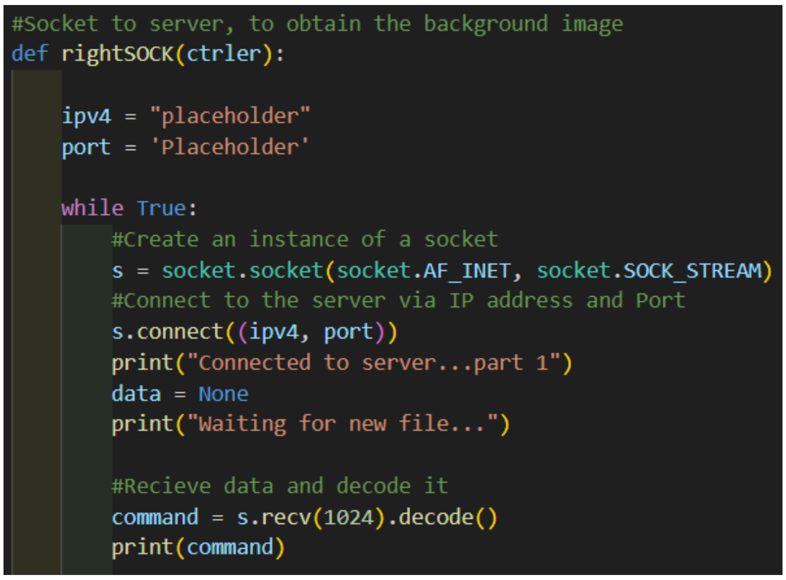
```python
        #Running the Code
        runningapp = True
        while runningapp:

            #Obtaining the data from the arduino
            data = self.arduino.readline()

            #Decoding the data from the arduino
            data = str(data, "utf-8")
            if data:
                #If the data has a value, spilt the data at the "-"
                data_array = data.split("-")
                #If the length of the data array is greater than 1, append the list of button location value
                #and the current time in milliseconds to the queue
                if len(data_array) > 1:
                    queue.append([int(data_array[1]), (time.time() * 1000)])

            #Set screen to 60fps
            pygame.time.Clock().tick(60)

            #Attach the Background image to the screen
            screen.blit(BackGround.image, BackGround.rect)
```



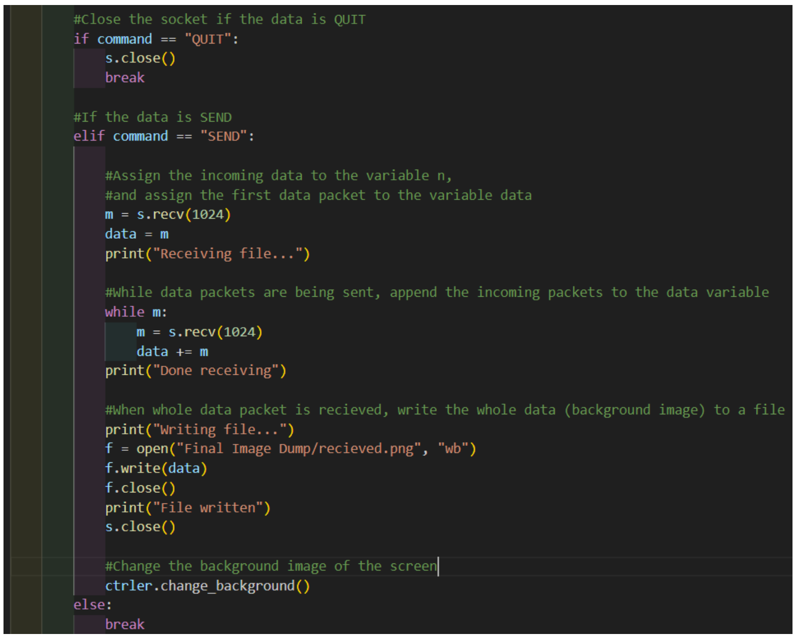
```python
            for event in pygame.event.get():

                #Close the Application
                if event.type == pygame.QUIT:
                    pygame.event.clear
                    runningapp = False
                    pygame.quit()

                #Detect the keypress on the laptop
                if event.type == pygame.KEYDOWN:
                    #If Key '1' was pressed, generate a Heart
                    if event.key == pygame.K_1:
                        new_animation = Animation(1)
                        animations.add(new_animation)

                    #If Key '2' was pressed, generate a Heart
                    if event.key == pygame.K_2:
                        new_animation = Animation(2)
                        animations.add(new_animation)

                    #If Key '3' was pressed, generate a Heart
                    if event.key == pygame.K_3:
                        new_animation = Animation(3)
                        animations.add(new_animation)

                    #If Key '4' was pressed, generate a Heart
                    if event.key == pygame.K_4:
                        new_animation = Animation(4)
                        animations.add(new_animation)

                    #If Key '5' was pressed, generate a Heart
                    if event.key == pygame.K_5:
                        new_animation = Animation(5)
                        animations.add(new_animation)
```



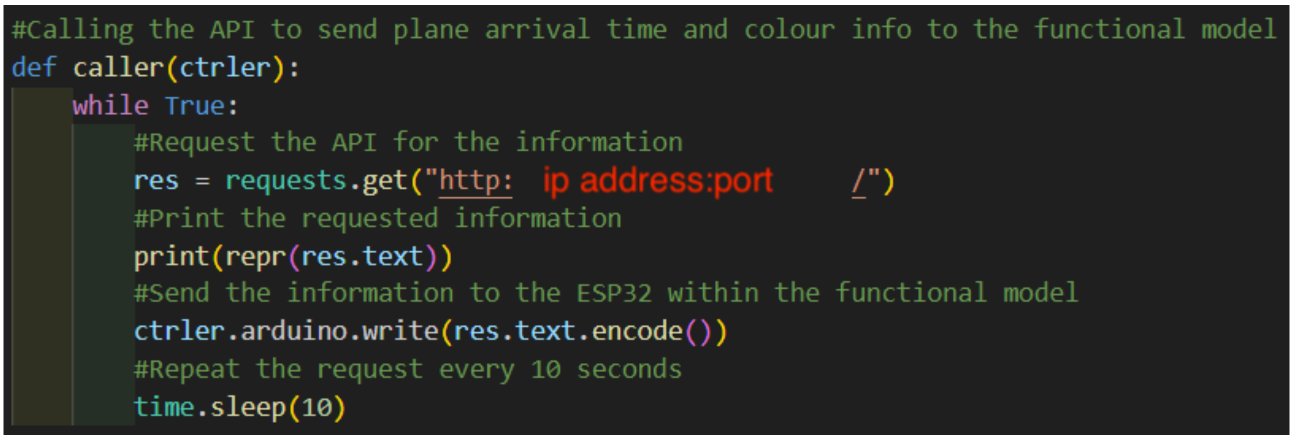
```python
            #Convert current time to milliseconds
            milliseconds = int(time.time() * 1000)

            #If there is an object in the queue, and if the difference between current time in milliseconds
            # and the time the data was sent is greater than the spawn delay timing,
            if len(queue) > 0:
                if milliseconds - queue[0][1] > spawn_delay:

                    #Generate the Heart at the location of the button pressed
                    new_animation = Animation(queue[0][0])
                    animations.add(new_animation)

                    #Remove the button location value and the timing of the data recieved from the queue
                    queue.pop(0)

            # Update Position
            animations.update()

            # Draw animation
            for entity in animations:
                screen.blit(entity.surf, entity.rect)

            # Flip the display
            pygame.display.flip()
```



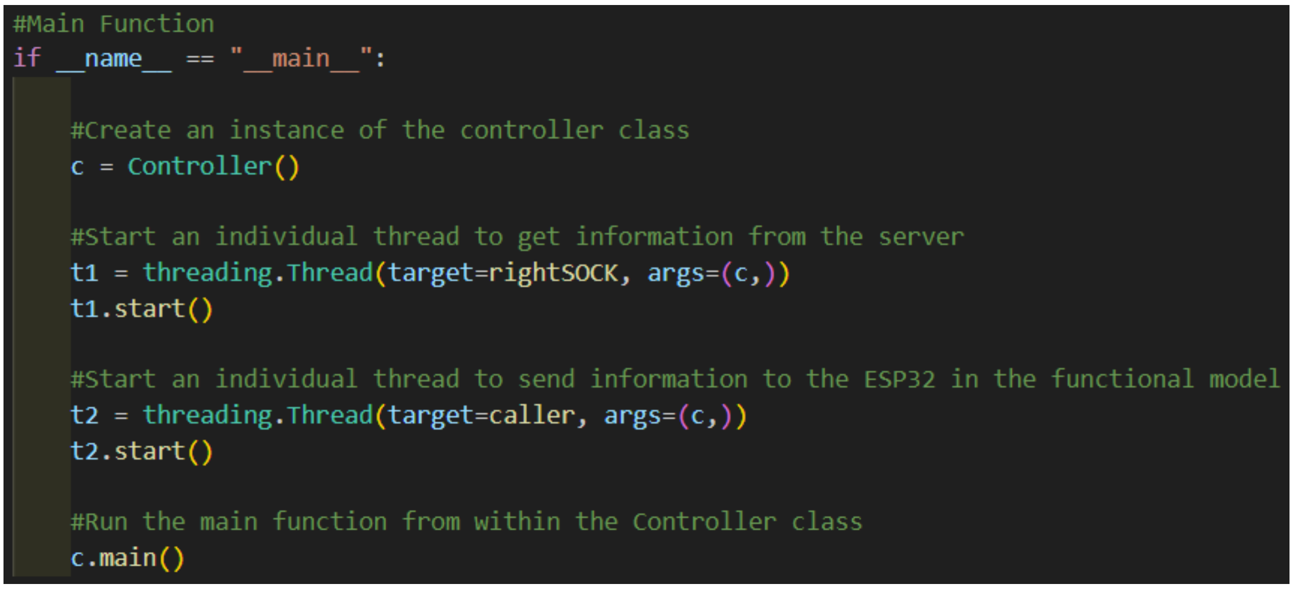
```python
#Socket to server, to obtain the background image
def rightSOCK(ctrler):

    ipv4 = "placeholder"
    port = 'Placeholder'

    while True:
        #Create an instance of a socket
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        #Connect to the server via IP address and Port
        s.connect((ipv4, port))
        print("Connected to server...part 1")
        data = None
        print("Waiting for new file...")

        #Recieve data and decode it
        command = s.recv(1024).decode()
        print(command)
```



```python
        #Close the socket if the data is QUIT
        if command == "QUIT":
            s.close()
            break

        #If the data is SEND
        elif command == "SEND":

            #Assign the incoming data to the variable n,
            #and assign the first data packet to the variable data
            m = s.recv(1024)
            data = m
            print("Receiving file...")

            #While data packets are being sent, append the incoming packets to the data variable
            while m:
                m = s.recv(1024)
                data += m
            print("Done receiving")

            #When whole data packet is recieved, write the whole data (background image) to a file
            print("Writing file...")
            f = open("Final Image Dump/recieved.png", "wb")
            f.write(data)
            f.close()
            print("File written")
            s.close()

            #Change the background image of the screen
            ctrler.change_background()
        else:
            break
```



```python
#Calling the API to send plane arrival time and colour info to the functional model
def caller(ctrler):
    while True:
        #Request the API for the information
        res = requests.get("http:  ip address:port     /")
        #Print the requested information
        print(repr(res.text))
        #Send the information to the ESP32 within the functional model
        ctrler.arduino.write(res.text.encode())
        #Repeat the request every 10 seconds
        time.sleep(10)
```



```python
#Main Function
if __name__ == "__main__":

    #Create an instance of the controller class
    c = Controller()

    #Start an individual thread to get information from the server
    t1 = threading.Thread(target=rightSOCK, args=(c,))
    t1.start()

    #Start an individual thread to send information to the ESP32 in the functional model
    t2 = threading.Thread(target=caller, args=(c,))
    t2.start()

    #Run the main function from within the Controller class
    c.main()
```