

# PROJECT 3

## Satisfaction Measurements



### Authors:

**Lara Turunc, Justyna Kluska, Severin Nyffenegger, Thomas M. Joensen**

**Date of Submission: 16/12-2022**

**Characters with Spaces: 53014**

# 1. Summary

The following report documents the work of group 4 on the Satisfaction Measurements Console. The survey is done by buttons on a console which operates with the ESP32. The data will be stored in a database, which is hosted on our Raspberry Pi. To communicate with the database, we send our data over MQTT to our MQTT-broker. In our product the Raspberry Pi works as the MQTT-broker, the host of our database and the host of our webpage. The webpage will be used as the visualizer from the collected satisfaction measurements data. Furthermore, we want to give our customers a secure console by building our own private network with integrated firewalls.

## Table of content

<b>1. Summary</b>	<b>2</b>
<b>2. Introduction</b>	<b>5</b>
<b>3. Requirements</b>	<b>5</b>
<b>4. The Business</b>	<b>6</b>
<b>5. Modules</b>	<b>6</b>
5.1. <i>Network Setup</i>	6
5.1.1. Module design	6
5.1.2. Module implementation	7
5.1.3. Formal Module Test	9
5.2. <i>Raspberry Pi</i>	10
5.2.1. Module design	10
5.2.2. Module implementation	11
5.2.2.1. SQL / MariaDB	11
5.2.2.2. MQTT	14
5.2.2.3. SQL Handler Module	17
5.2.2.3.1. SQL handler - intro	17
5.2.2.3.2. SQL handler - initialiser	18
5.2.2.3.3. SQL handler - inserter	18
5.2.2.3.4. SQL handler - fetcher	19
5.2.2.3.5. SQL handler - algorithms	20
5.2.2.3.6. SQL handler – title and location fetcher	23
5.3. <i>Console</i>	29
5.3.1. Module design	29
5.3.2. Module implementation	31
5.3.2.1. MQTT and buttons	31
5.3.2.2. Offline-mode	34
5.3.2.3. Deep-sleep mode	35
5.3.3. Formal module test	36
5.4. <i>Webpage</i>	37
5.4.1. Module design	37
5.4.2. Module implementation	39
5.4.3. Formal module test	40
<b>5.5. System Integration test</b>	<b>40</b>

<b>5.6. Project management</b>	<b>41</b>
<b>6. Prove</b>	<b>44</b>
<b>7. Conclusion</b>	<b>44</b>
<b>8. Reflection</b>	<b>44</b>
<b>9. Appendices</b>	<b>45</b>
9.1. <i>MQTT setup</i>	45
9.2. <i>Python</i>	46
9.3. <i>Console software</i>	58
9.4. <i>Webpage</i>	65
9.4.1. Index.HTML	65
9.4.2. ALL365DaysPie.html	70
9.4.3. AllLine30Days.html	74
9.4.4. SH-A2.09-365DaysPie-html	78
9.4.5. SH-A2.09-Line30Days.html	82
9.4.6. About.html	87
9.5. death_valley_highs_lows.py by Kristian Pontoppidan	93

## 2. Introduction

Satisfaction surveys are questionnaires that are used to help owners of businesses to understand their customers' points of view and get feedback from clients about their products or services. Nowadays it is very important to know the expectations of customers and develop the business according to their needs, otherwise, the competitors might be faster. Checking the level of customer satisfaction is also important for public services to improve and simply make the life of residents better. The satisfaction surveys can be designed in different forms for example online questionnaires, telephone surveys, paper polls, in-person interviews, etc. The modern solution to check customer satisfaction right very quickly after the service, is setting up a measurement console at the exit of the buildings. Typically, the customers can choose between one of 3 emojis, sad, mediocre, or happy on a console to indicate their level of satisfaction with the service.

## 3. Requirements

The general requirements for working on the project are the same for the whole class and can be found in the project description. However, the requirements differ according to the chosen project case. As our group chose case 3, the specific requirements for our solution are listed below:

- A wireless solution is preferred
- For Hardware: use existing old raspberries, compare to ESP32
- Install and configure a new wireless infrastructure with a suitable security level for the project.
- Automate deployment of new capture devices.
- Use DHCP or similar process to assign the ID to the capture device.
- Communication to the administrative system must be over MQTT or VPN
- Position of the console must be saved along with the "satisfaction data".
- Measuring and saving values for field strengths of known wireless access points, may be used as position data

An administrative system:

- Will save the entered data on a Central Linux server. Preferably in a database
- Will be able to display the "satisfaction data" in real time.
- When a device becomes active, it should start capturing data. And hold data to serve to the database once connected.

## 4. The Business

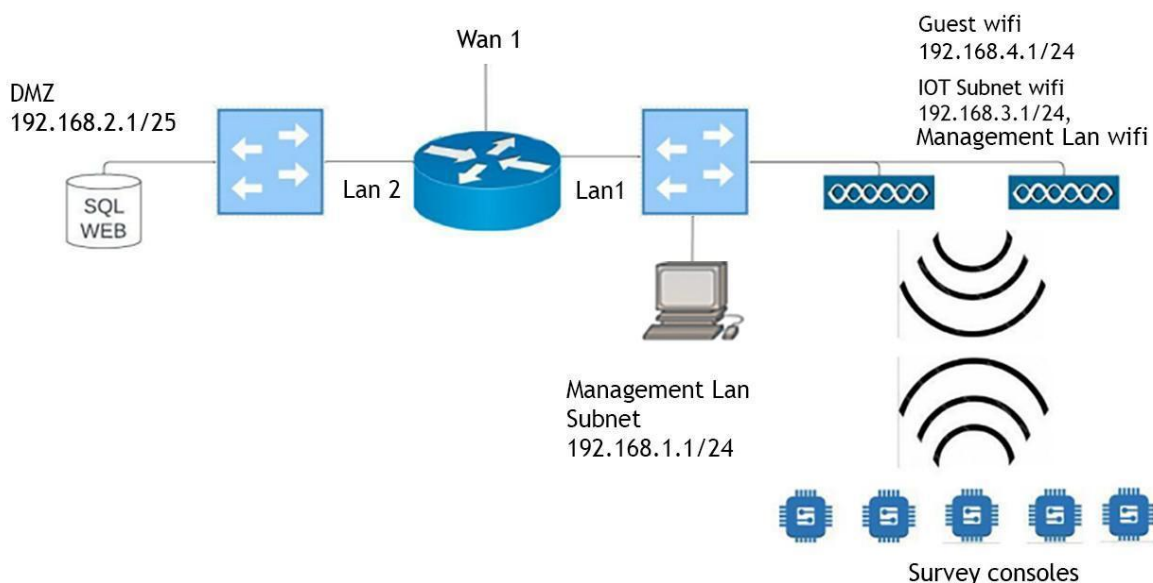
Satisfaction measurement tools are used in many different businesses, especially by big corporations. It is very important for business owners to know if their customers are satisfied right after the visit. The satisfaction measurement surveys can be seen for example at the exit of the big stores, restaurants, or service providers for example in the Ikea store and restaurant. They can be in the form of a console with colorful buttons, a touch screen, or a paper survey, however it is very important for the solution to be as easy and fast for a client as possible.

## 5. Modules

For best possible performance we divided the task into different parts. With different parts available every group member could choose the field they were most comfortable in.

### 5.1. Network Setup

#### 5.1.1. Module design



Our network begins with the router, a Ubiquiti security gateway, which is connected to a switch on each LAN port. On port LAN2, we're hosting our DMZ, where our server is connected. This is done because it's running a webserver, which is accessible from anywhere, as can be seen in the port forwarding table, on the next page. This subnet is also with mask /25, simply to demonstrate that it needn't be as large as the other two.

On LAN1 we're hosting the management Lan subnet, which can be reached through an ethernet port on the switch or through its associated WIFI, hosted by the access points. As the name suggests, this subnet is intended for administration only, as will also be apparent

with the port forwarding. The access points also host a secondary and tertiary subnets, the IoT subnet for the consoles, and guest wifi which are exclusively accessible through WLAN.

As it proved necessary, we are also hosting a guest wifi, to allow access to the webserver, since we were unable to figure out the settings for WAN access in time, as we hadn't discovered it as an issue until at the very end.

Furthermore, Unifi also allows for hosting specific WIFI's on specific access points, should certain subnets need to be sectioned off.

### 5.1.2. Module implementation

#### Networks

NAME	ROUTER	SUBNET	IP LEASES	INTERNET
Default	group4_gateway	192.168.1.0/24	0 (249)	Default (WAN1)
group4_dmz	group4_gateway	192.168.2.0/25	0 (125)	Default (WAN1)
group4IoT	group4_gateway	192.168.3.0/24	0 (249)	Default (WAN1)
Group4_us...	group4_gateway	192.168.4.0/24	0 (249)	Default (WAN1)

Figure displays the subnets; LAN, DMZ, IoT, and user LAN

Figure displays the ports LAN and DMZ are mapped to

#### Ports

NAME	STATUS
WAN - WAN	Auto Negotiation >
LAN - LAN	100 Mbps HDX >
dmz	Auto Negotiation >

[Configure Interfaces](#)

Figure displays the wireless interfaces for LAN and IoT\_LAN

#### WiFi

NAME	NETWORK	AP GROUPS	CLIENTS (PEAK)	AUTHENTICATION	WIFI EXPERIENCE
group4_guest	Group4_user...	All APs	0 (1)	WPA Personal	N/A
group4IoT	group4IoT	All APs	0 (2)	WPA Personal	N/A
group4_lan	Default	All APs	1 (3)	WPA Personal	93%

[+ Create New WiFi Network](#)

DMZ has no wireless interface as it isn't needed. Strictly speaking, the 'LAN' doesn't require wireless either, it has an ethernet interface, but we've kept wireless access for debugging, and starting our python scripts as they don't run automatically when the server is started. In an industrial setting, this level of interaction with the machine, would of course be unnecessary, as such services would be accounted for in the boot sequence since they don't benefit from being launched by a user. For specific VLANs used, we went quite plainly with 1, 2 and 3.

The thought process of our physical infrastructure is as follows:

We wanted a setup for our demo model that, while not realistic in its dimensions, clearly displays the components and their connections, and illustrates its capacity to be scaled up and down. It's no big deal to expand the network by attaching additional switches or access points or cutting it down to a minimum of one switch and one AP, or one end device and one access point.

Another benefit of this construction is its portability, the literal one, which also comes in handy for a demo model. Scaling a flight of stairs with it would be a small endeavor.



Port Forwarding

[Create New Forwarding Rule](#)

NAME ^	FROM	PORT	DEST PORT/IP	ENABLED
dmz http	*	80	192.168.2.5:80	✓
dmz https	*	443	192.168.2.5:443	✓
lot mqtt	192.168.3.1/24	1883	192.168.2.5:1883	✓
SSH	192.168.1.1/24	22	192.168.2.5:22	✓

The port forwarding is quite straightforward. As any good DMZ, ours allows access to our web server from everywhere on port 80 and 443, and restricts other protocols to their respective subnet. SSH is a management tool, so it's only usable from the management LAN, and the MQTT publishing is only possible from the IoT subnet.

That said, we've remained unable to reach the DMZ from ITEK 2nd in the few days we've been aware of the issue. This could perhaps be an issue with the unifi firewall.



### 5.1.3. Formal Module Test

Testing a network with clearly defined purposes like ours is a matter of testing if the protocols work from their respective subnet.

Can an IoT device connect to the broker, hosted on the server, and publish messages?

Does a secure shell work?

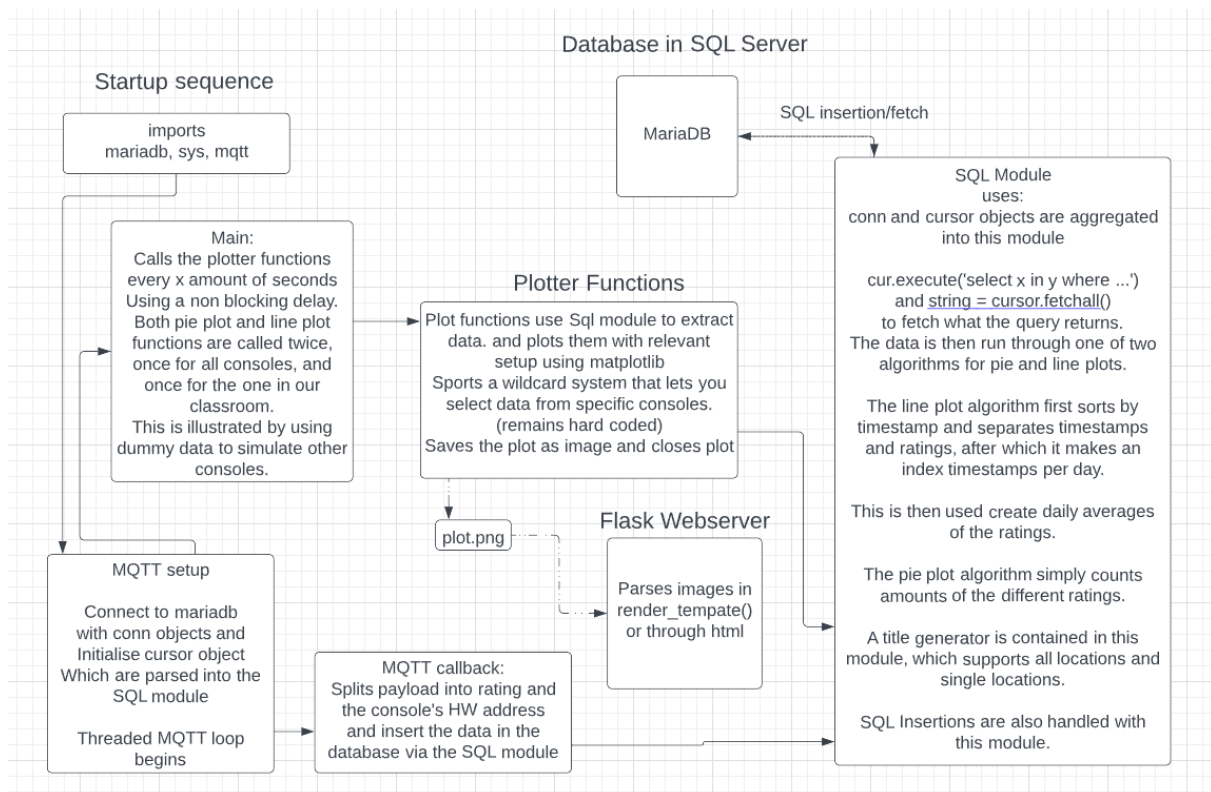
The initial answer was *no*.

The server wasn't showing up on the network, after setting up static network config in ubuntu's netplan. This was, naturally, because it wasn't done right. Ubuntu 22.04 has a new and slightly more difficult way of configuring the netplan than 20.04 and 21.04, with the deprecation of gateway4 and its complete replacement with static routes. After this hitch was loosened, the rest was more or less on rails, MQTT worked as it should, same with secure shell. We were also unable to get access through the WAN firewall to our dmz on port 80, so we implemented a guest wifi for guests to see the web page.

## 5.2. Raspberry Pi

The Raspberry-Pi (RPI) is the workhorse of our project and therefore it is important to set it up flawlessly. We use the RPI to work as the MQTT-broker, host the database and host the webpage. To make these parts work together, we needed to create an API in order to make the different programs communicate together.

### 5.2.1. Module design



The software can be broken into two parts. The program we use, Mosquitto's MQTT Broker, MariaDB, flask webserver and the API we write to make them work together.

Effectively the chain of function is that a survey console connects to the broker and begins publishing methods, which the broker distributes, this distribution is received by the python program which is subscribed to it. In the python program there is a callback function that converts the received message into different variables and inserts them into an SQL table, hosted on our MariaDB SQL server. After this, data from relevant time periods is then extracted to be used for plotting. The plot types we settled on were two pie plots, one with ratings from all consoles, and one with data from a single one with a time frame of the previous 365 days. The same goes with the line plot, except the time frame is only 30 days and the ratings of each day are averaged. The plots are then saved as png images to be used by the webserver.

## 5.2.2. Module implementation

### 5.2.2.1. SQL / MariaDB

For the second part we needed to install and set up the SQL server MariaDB, a fork of MySQL.

We used the following commands.

Installation:

```
$ sudo mysql_secure_installation
```

Login:

```
$ sudo mysql -uroot -p
```

For additional security we removed anonymous users and blocked remote root access.

Then we created a database and a user to access it remotely

```
MariaDB [(none)]> CREATE DATABASE group4;
```

```
Query OK, 1 row affected (0.003 sec)
```

```
MariaDB [(none)]> CREATE USER group4@'localhost' IDENTIFIED BY 'group4four';
```

```
Query OK, 0 rows affected (0.009 sec)
```

```
MariaDB [(none)]> GRANT ALL PRIVILEGES ON group4.* TO group4@'localhost';
```

```
Query OK, 0 rows affected (0.005 sec)
```

Logically we created a database called “group4” to house the console table where we store the console data and locations table for locations. To access the database from python we had to create a user and grant it the necessary privileges.

Afterwards we created the table “console”. We first start with a number that is automatically incrementing with each data entry. Although we didn’t end with this, it could be used for sorting dates, this will be discussed later. The rating is in our third column using float as the type to allow averaging in queries, again something to discuss later. 1 stands for bad service, 2 for mediocre and 3 for good service. To identify which console sent which rating we save the MAC-address of the message in the fourth column. In the last column we save the date and time of the data that was entered.

```
MariaDB [(group4)]> CREATE TABLE console(
id INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
rating FLOAT,
macAddress VARCHAR(17),
ts TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP
);
```

id	rating	macAddress	ts
1	1	NULL	2022-11-29 10:28:07
2	2	NULL	2022-11-29 10:28:30
3	3	NULL	2022-11-29 10:28:35
8	2	NULL	2022-11-29 10:31:27
9	1	NULL	2022-11-29 10:34:45
10	3	NULL	2022-11-29 10:34:49
11	1	90:38:0C:5A:44:2C	2022-11-29 11:57:42
12	1	90:38:0C:5A:44:2C	2022-11-29 11:59:43
13	2	90:38:0C:5A:44:2C	2022-11-29 11:59:47
14	3	90:38:0C:5A:44:2C	2022-11-29 11:59:51
15	1	90:38:0C:5A:44:2C	2022-12-06 13:30:33
16	1	90:38:0C:5A:44:2C	2022-12-06 13:30:56
17	2	90:38:0C:5A:44:2C	2022-12-06 13:31:00

Additionally a table containing locations was created

```
MariaDB [(group4)]> CREATE TABLE locations(
id INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
location VARCHAR(30),
macAddress VARCHAR(17),
ts TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP
);
```

```
MariaDB [group4]> select * from locations
-> ;
```

id	location	macAddress	CreationTime
6	SH-A2.09	90:38:0C:5A:44:2C	2022-12-14 11:47:15
7	N/A	22:82:22:22:22:22	2022-12-15 13:13:35
8	N/A	32:82:22:22:22:22	2022-12-15 13:19:50
9	N/A	32:82:22:22:22:22	2022-12-15 13:20:06
10	N/A	32:82:22:22:22:33	2022-12-15 13:22:01

Interacting with MariaDB from python is done using their own library, the Connector/Python, which has proven an usually great challenge to install on ubuntu based systems as python's package manager pip throws an obscure exception almost regardless of which version of mariadb you try to install, regardless of the system. The solution was simple, manual installation.

To use it in python you must firstly you import it along with sys

```
import mariadb, sys
```

Then you initialise the connect object filling in the necessary details and later, its cursor object

```
#mariadb

try:
    conn = mariadb.connect(

        user="user",
        password="password",
        host="127.0.0.1",
        port=3306,
        database="db-name",
        autocommit=True
    )
    print("mariadb connected with code 0")
except mariadb.Error as e:
    print(f"Connection error, code: {e}")
    sys.exit(1)

# Get Cursor

cur = conn.cursor()
```

Afterwards, you can use most if not all SQL queries from python using:

```
cursor.execute("query")
```

And fetching its result using

```
cursor.fetchall(), cursor.fetchmany(), cursor.fetchone()
```

The fetch methods all return a list of sets, which typically look as follows:

```
[(x0,y0,z0),(x1,y1,z1),]
```

That said, we've exclusively used the fetchall methods and handled single items with the 'SELECT DISTINCT' statement in SQL queries.

#### 5.2.2.2. MQTT

The first thing that happens after you turn on our survey console is that it tries to connect to an MQTT broker, so that is the origin point on the raspberry. Mosquitto makes an adequate broker for this purpose. Especially as it allows an additional layer of security, that of a username and password. In our case we used a generic set of credentials which we also use for everything else. Again, not something that would fly in the industry, but it would be overkill to use proper credentials when the worst malicious actors are classmates, fast at work with their own projects. (12.1 MQTT setup)

Firstly, the following lines must be added to the mosquitto config file: `/etc/mosquitto/conf.d/`

```
per_listener_settings true
allow_anonymous false
password_file /etc/mosquitto/passwd
```

This enforces listeners and publishers to have their own settings, and to be identified. It also specifies the file that contains the passwords

The following line is used for defining the port for MQTT to run over TCP.

```
listener 1883
```

Then the following commands are issued:

```
$ sudo mosquitto_passwd -c /etc/mosquitto/passwd [username]
$ sudo mosquitto_passwd /etc/mosquitto/passwd [username2]
```

These prompt you to enter a password and they will create entries in the password file. Note that the `-c` flag denotes creating a new file or overwriting a previous file. The second command instead appends to the end of the file.

Hashed passwords can then be seen using

```
$ cat /etc/mosquitto/passwd
group4:$7$101$SC6BmI4cIMmUGROb$ckfVxSD/76huae5wjcTEq2bx5AU9ya5cKKL9wWCV7r2fMQN+FLLn4pRAE
cMW3Ttu3ZUjVYZa0aK84yrypDVqww==
```

The service will need to be restarted afterwards.

Then it can be tested using mosquito's pub and sub commands

```
term 1: $ mosquitto_sub -h 10.120.0.57 -t dimple/# -u uname -P passwd
```

```
term 2: $ mosquitto_pub -h 10.120.0.57 -m "test" -t dimple/johnnie/barnacle/jr/ -u uname -P passwd
```

```
term 1: test
```

Interacting with MQTT on the python side can be done using the Paho MQTT client.

Firstly, you must import the library and establish a connection by initializing the Client object and using its methods.

```
#MQTT
```

```
import paho.mqtt.client as mqtt
```

```
try:
```

```
    client = mqtt.Client()
```

```
    client.on_connect = on_connect
```

```
    client.on_message = on_message
```

```
    client.username_pw_set("user", "password")
```

```
    client.connect("127.0.0.1")
```

```
except:
```

```
    print("mqtt connection error")
```

Then you define a function for when a connection has been established

```
def on_connect(client, userdata, flags, rc):
```

```
    if rc==0:
```

```
        print(f"MQTT connected with code {str(rc)}")
```

```
        client.subscribe("Project3/console")
```

```
    else:
```

```
        print("MQTT connection Error: Returned code=",rc)
```

After attempting to establish a connection, this one's on\_connect function simply checks if the rc flag is in order and subscribes to a topic.

The following function to define, is the `on_message` function, also called callback. The way we use ours is to strip the payload of unnecessary characters and split it into usable parts. These parts, the rating and mac address, are then inserted into the SQL table that contains the ratings, using a method in our own SQL handler module, which will be covered later. It also does a check as to whether the mac address is recognized. If it isn't, an empty location and the mac address are inserted into the table that contains the locations, also using our SQL handler.

```
#Callback for msg

def on_message(client,userdata,msg):
    try:
        load = str(msg.payload)
        load = load.strip("b'")
        x = load.split("&")
        rating = float(x[0])
        address = str(x[1])
        print("rating received: ",rating," from: ", address)
        Sql.inserter(rating, address,True)
        if address not in Sql.macList:
            Sql.inserter("N/A",address, False)

    except Exception as e:
        print("Error in payload: ",e)
        print(f'{msg.topic} {msg.payload}')
```

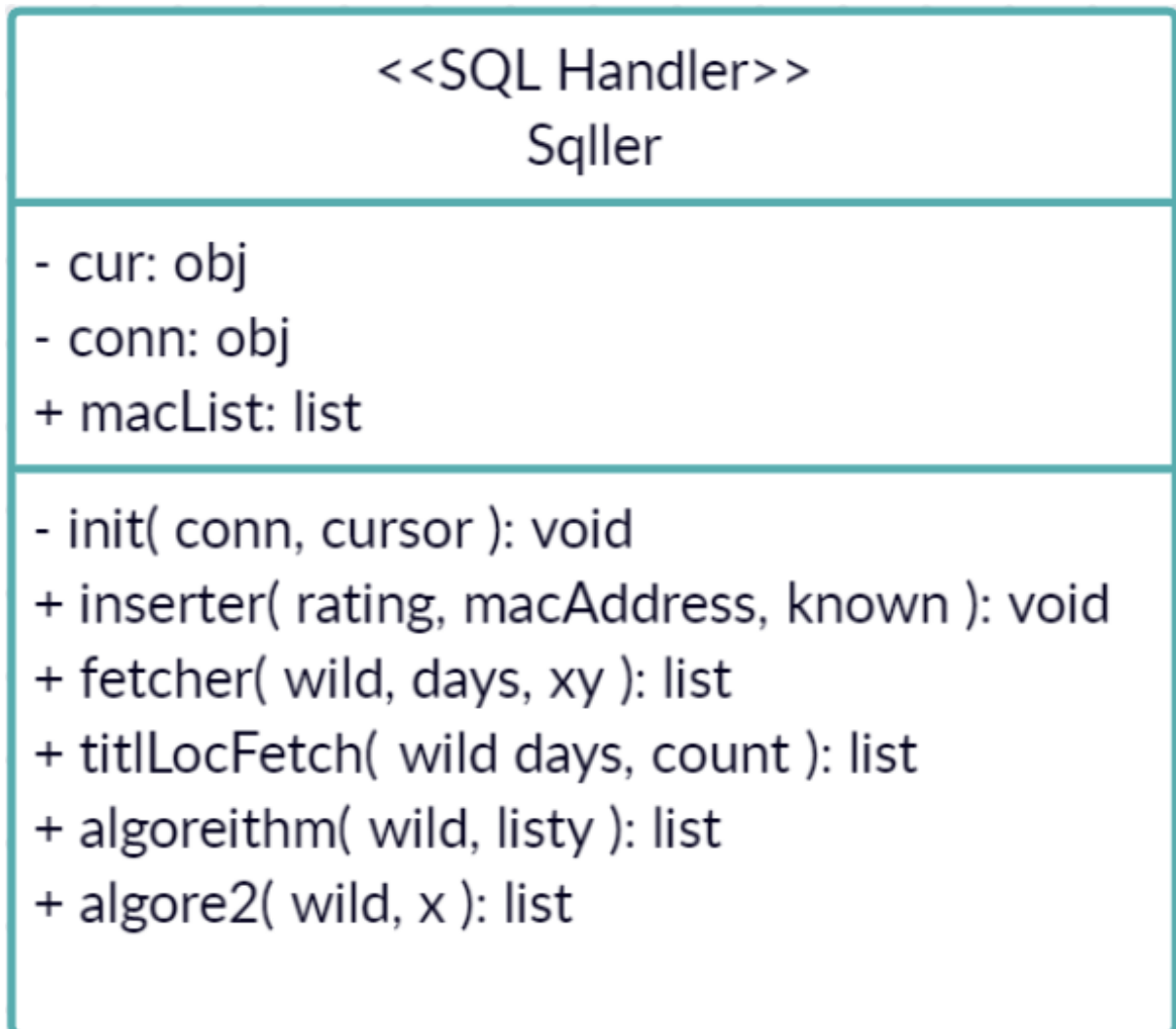
On the topic of the paho client, if you simply want to publish messages from python without receiving, you can forego setting the `on_connect()` and `on_message()`, and you can simply use

```
publish("topic", payload=None, qos=0, retain=False)
```



## 5.2.2.3. SQL Handler Module

## 5.2.2.3.1. SQL handler - intro



The SQL Handler module handles everything there is to be done with SQL. It's responsible for extracting data from the SQL server and converting it to a usable format. The conn and cursor objects from earlier are aggregated into it by parsing them into the SQL handler's initialiser. In future iterations they could potentially be composited inside it instead, as they aren't used externally.

The initializer also creates a list of known mac addresses.

The *fetcher* and the title-location fetcher, *titlLocFetch* methods, and the two algorithms, *algoreithm* and *algore2*, take a wildcard (*wild*) which determines which console's data is to be used, or if all data is to be use if *wild* is set to 0.

The xy variable in *fetcher* determines which type data is to be extracted whether it's only ratings for a pie chart, or both ratings and dates for a line chart. This variable also determines which algorithm is to be called to treat the raw data.

Ideally though you'd handle as much of the computing as you can within your SQL queries, as SQL is vastly faster than python.

#### 5.2.2.3.2. SQL handler - initialiser

The init defines the parsed objects within the class to be used as its methods.

It also fetches a query of known mac addresses, which it converts to a usable list.

```
class Sqlier:
    def __init__(self,conn,cur):
        self.__cur = cur
        self.__conn = conn                #Defining the aggregated objects
        self.__cur.execute("SELECT macAddress from group4.locations ORDER by id")
        self.__macList0 = self.__cur.fetchall() #Fetching list of Mac addresses
        self.macList = []
        for mac in self.__macList0:         #Cycling through the list
            self.macList.append(mac[0])     # to append values to new list
        del(self.__macList0)
        print("known mac addresses: ", self.macList)
```

#### 5.2.2.3.3. SQL handler - inserter

The inserter method does a check of the 'known' boolean to determine which submethod is to be used. This bool is decided in the on\_message function. If known is true, it will insert the rating and MAC into the table with ratings. If known is false, it will insert the new MAC and an empty location into the locations table, which has an automatic time stamper, like the table for the ratings. The time stamper is there so that you can note the time down yourself and compare the time of setting up the device with the time of registration in the SQL table.

It works as follows; you set up a console and enter the first rating and note down the time. Then you can set up several over consoles, noting down the time of entering a rating on each.

Then, you can compare the timestamps to the entries on the database and add names of the locations, which could be done manually on the database, or through a helper program.

This is the closest we could get to complete automation.

```
def inserter(self,rating,macAddress, known):
    if known == True:
        self.__cur.execute("INSERT INTO group4.console( rating, macAddress) VALUES(?,?)",
(rating, macAddress))
    if known == False:
        print("unknown mac: ",macAddress, " inserting in group4.locations")
        self.__cur.execute("INSERT INTO group4.locations(location, macAddress) VALUES(?,?)",
(rating, macAddress))
        print("updated mac addresses: ", self.macList)
```

#### 5.2.2.3.4. SQL handler - *fetcher*

The *fetcher* firstly does a check of the *xy* variable. This is currently a string for the sake of readability, but for an industry model, it would be handled with a less memory intensive type. Whether *xy* is “y” or “xy” determines whether it returns ratings alone (only y axis), for pie plots, or both timestamps and ratings (x and y axis) for line plots.

Afterwards it checks *wild*, which corresponds with the survey console’s index number in the list of known mac addresses (*self.macList/sqler.macList*). If it’s greater than zero, it’ll select ratings from a specific index number, and if it isn’t, it’ll select data from all variables. We have only built one console, but we can simulate other consoles by either sending an MQTT message in the right format while running the program or inserting them directly in the database.

Important to note is that *days* correspond with the number of desired days back to be fetched from the table. This is done using the *DATE\_SUB* function in sql which in this case, subtracts an interval of days from current day, found via the *CURDATE* function.

```
def fetcher(self, wild, days, xy):
    if xy == "y":
        if wild >= 1:
            wild += -1

            self.__cur.execute(f"SELECT rating FROM group4.console WHERE macAddress='{self.macList[wild]}' and ts >= DATE_SUB(CURDATE(), INTERVAL {days} DAY) ")
        else:
            self.__cur.execute(f"SELECT rating FROM group4.console WHERE ts >= DATE_SUB(CURDATE(), INTERVAL {days} DAY)")

    self.__ratings = self.__cur.fetchall()
    self.__ratings2 = self.algore2(self.__ratings)

    return self.__ratings2
```

It then calls the algorithm method *algore2* and parses the returned query in it, to ultimately return what’s returned from *algore2*.

The second half is similar, but with the difference that it selects timestamps along with the ratings. The list is then sorted before being parsed into the method *algoreithm*.

```

elif xy == "xy":

    if wild >= 1:
        wild += -1

        self.__cur.execute(f"SELECT ts, rating FROM group4.console WHERE macAddress= '{self.macList[wild]}' and ts >=
DATE_SUB(CURDATE(), INTERVAL {days} DAY) ")
    else:
        self.__cur.execute(f"SELECT ts, rating FROM group4.console WHERE ts >= DATE_SUB(CURDATE(), INTERVAL {days} DAY)
")#ORDER BY idx")

    self.__raw = self.__cur.fetchall()

    self.__raw.sort()
    listy = self.algoreithm( self.__raw)
    return listy

```

#### 5.2.2.3.5. SQL handler - algorithms

The first algorithm we'll cover, handles data for pie charts.

Its task is to return three usable sizes for the pie charts, which can consist of sizes like these: (3,5,9) or (3,6,1,8,4,7).

```

def algore2(self, x):
    self.__bad = 0
    self.__mid = 0
    self.__good = 0

    for i in x:
        if i[0] == 1:
            self.__bad += 1           #Counts the ratings
        if i[0] == 2:
            self.__mid += 1
        if i[0] == 3:
            self.__good += 1

    self.__returnable = [self.__bad, self.__mid, self.__good]
    return self.__returnable

```

Effectively what it does, is to sum up the number of ones, twos, and threes and returns them as a list.

The second algorithm, *algoreithm* handles the data for line charts. This one is a fair bit more complex, as the line charts aren't simply charts of ratings, but an average rating for each day. The algorithm is separated in two, the first part separates time stamps and ratings from each other..

```
def algoreithm(self, listy):
    #Getting X *****
    ...
    #Some code is omitted here, see appendix for details.
    for row in listy:
        self.current_Ts = row[0].strftime('%Y-%m-%d-%H-%M') #Creates timestamp
        self.current_day = row[0].strftime('%Y-%m-%d') #Creates date
        self.rating = row[1] #Creates float of rating
        self.TsWithTIME.append(self.current_Ts) #Appends timestamp to list
        self.TsWithoutTIME.append(self.current_day) #Appends date to list
        self.ratings.append(self.rating) #Appends rating to list

    del(self.rating, self.current_day)
    self.ratingCount = len(self.TsWithTIME) #Count is used for titles
    self.x = list(set(self.TsWithoutTIME)) # Elimination of duplicates
```

This part serves to create a list of timestamps, dates, and ratings.

The second part is for making the average scores

```
#Getting Y *****
#Creating empty list
self.ytemp3 = []
for i in self.x: #Cycles through the days
    self.ytemp1 = [] #Temporary list used for creating avg
    for u in self.TsWithTIME: #Cycles timestamps
        if i in u: #Checks if date is in timestamp
            self.idx = self.TsWithTIME.index(u) #Fetches current index of the timestamp
            # list item.
            self.ytemp0 = self.ratings[self.idx] #Specific rating is taken via index num
            self.ytemp1.append(self.ytemp0) #Rating is added to list of ratings for
            # that day

    self.ytemp1sum = sum(self.ytemp1)
    self.ytemp1len = float(len(self.ytemp1)) + 1
    self.ytemp2 = self.ytemp1sum/self.ytemp1len #This creates an average for that day

    self.ytemp3.append(self.ytemp2) #The avg is then appended to list of avgs
self.x, self.y = zip(*sorted(zip(self.x, self.ytemp3))) #Lists are then sorted again,
# according to timestamp using
# a zip method
self.lister = [self.x, self.ytemp3, self.ratingCount] #Returnable list
```

```
return self.lister
```

The zip method aggregates the list entries to each other in tuples, like how there were, when fetched from SQL. The aggregated list is then sorted and unzipped. We've tested with the first and second sorting alone and both produce inconsistent date order in the plots. In a future iteration this would have to be ordered by the index number in the SQL select query for more reliable results.

A prudent question here would be why we don't just do all of this in SQL. And we did ask ourselves that, and as it turns out, it's totally possible, in SQL. We had a type of line plot planned with averaging per hour. Using the query:

```
SELECT avg(rating) FROM console WHERE creation_time >= DATE_SUB(CURDATE, INTERVAL 14 DAY) GROUP BY hour(creation_time);
```

Note: this example groups by minutes, but it illustrates the point just the same.

```
MariaDB [group4]> SELECT avg(rating) as hourlyDaily FROM console
-> WHERE creation_time >= DATE_SUB(CURDATE(), INTERVAL 14 DAY)
-> GROUP BY minute(creation_time)
-> ;
```

hourlyDaily
2
1
1.9
1.8666666666666667
1.8571428571428572
1.9666666666666666
2.024390243902439
2.1666666666666665
2
2.375
1.8636363636363635
1.8666666666666667

```
12 rows in set (0.002 sec)

MariaDB [group4]> |
```

But when we do this in python, cursor.fetch method does the most curious thing;

```
mDB returns : [(2.0,), (1.0,), (1.0,), (1.0,), (1.0,), (1.0,), (2.0,), (2.0,), (2.0,), (2.0,), (1.0,), (1.0,)]
```

It truncates the floats. And as it happens, this behavior is not described in MariaDB's documentation.

### 5.2.2.3.6. SQL handler – title and location fetcher

Something that makes sense to include with a plot is a descriptive title, which is why it's baked into the images used on the webpages.

As such a method had to be devised to handle just that.

This method fetches a title and location using a mac address, and like all the other methods, it uses the wildcard for that. It takes a day count and a rating count.

```
def titlLocFetch(self, wild, days, count):

    if wild >= 1:                                #Wildcard check
        wild += -1                                #Corrects position for macList, as python lists
                                                # start with zero

        self.__cur.execute(f"SELECT location FROM group4.locations where macAddress = '{self.macList[wild]}'")
        self.__loc = self.__cur.fetchall()        #Query fetches location

        self.__loc1 = ""
        self.__loc1 = str(self.__loc[0]).strip("'{}"), #Conversion to usable string

        self.__title = f"Ratings from classroom: {self.__loc1} for the past {days} day(s)\n Total rating(s): {count}" #*****
                                                #Creates title with location
                                                # and number of ratings

        #From all terminals
    else:
        self.__title = f"Ratings from all locations for the past {days} day(s)\n Total rating(s): {count}" #*****
        self.__loc1 = ""                                #Generic title for plots
                                                # with all consoles

    setty = []
    setty.append(self.__title)
    setty.append(self.__loc1)

    return setty
```

#### 5.2.2.4. *Plotting program*

##### 5.2.2.4.1. *Plotting program - setup*

Aside from the imports for Mariadb and paho MQTT, datetime and timedelta from datetime are also imported. This is to do a simple non-blocking delay.

The imports take place within

```
if __name__ == "__main__":  
    from sqller import Sqller  
    import paho.mqtt.client as mqtt  
...
```

which is an obvious developmental oversight and should not be the case in a production model.

This effectively nullifies the `__name__` check, as the purpose of that is to prevent imported code from running automatically. The `__name__` check is often used as a safeguard against potential bugs and in worst cases, malware.

After creating the mqtt client object and mariadb connect and its cursor object, our SQL handler module is created with the connect and cursor parsed in its parameters.

The rest of the setup looks as follows:

```
# Get Cursor  
cur = conn.cursor()  
  
Sql = Sqller(conn,cur)  
  
then = dt.now()  
now = dt.now()  
client.loop_start()  
main(now,then)
```

The threaded MQTT client loop is started here. Refer to [5.2.2.2](#) for the mqtt functions



#### 5.2.2.4.2. *Plotting program - main*

The main loop is almost as simple as it gets

We utilise an infinite loop within which we have a datetime comparison to act as a non-blocking delay which may be a bit of overengineering when the mqtt client loop is running in a separate thread. Its function boils down to a timestamp taken right before the check, being compared to an old timestamp. So if the difference is greater than 10 or seconds, the plotting cycle is activated and afterwards, the *then* timestamp is updated.

Two plotting functions are called twice, each are called once for ratings from all survey consoles, and once for the console with index = 1 in the sql tablem, or index=0 in sqler.macList in python.

This is done by specifying the id in the plot functions' first argument. The second argument is the number of days in the desired time frame. This is hardcoded for now but could easily be made interactive by taking user input during the startup phase or for each cycle, but while the latter could make it more dynamic in a sense, the consistent required user interaction would kill the automation.

It could also run through a cycle of consoles or have random console selected for each cycle, should glamour be an interest.

Try-except clauses is to catch potential errors, occurred during testing, where matplotlib's `plt.savefig('savename')` method would raise an exception from being unable to open a non-existent file instead of creating a new file. This could be due to using relative paths and perhaps not testing it from the correct directory, as the subdirectory used for the plot images would not exist in this case. The flask program would do something similar when run from the wrong directory, ex: `$ python3 project3/main.py`. It seems python files can use whatever directory the cursor is in, in the terminal, rather than the file's own location, which can cause some confusion with relative paths such as "static/plot.png", so the simple solution could be making sure to run it from the correct directory. Another solution could be using absolute paths.

```
def main(now, then):
    while 1:
        now = dt.now()
        if (now - then).total_seconds() >= 10:
            print("plot cycle started")
            try:
                then = dt.now()
                dayLinePlotter(0,30)
                dayLinePlotter(1,30)
                piePlotter    (0,365)
                piePlotter    (1,365)
            except:
                print("error in plotting")
            now = dt.now()
            sleep(0.01)
            print("plot cycle ended")
            then = now
```

#### 5.2.2.4.3. Plotting program – Line plot

The dayLinePlotter calls the sqller's Fetcher method with the keyword xy for returning both an x and y value (time and rating) and with a number of days.

The data will then be plotted by matplotlib's line/axis plot function.

A title is then fetched which will depend on the *wild* number and *days* number

Axes are given labels, x for dates, y for ratings. Plot undergoes a bit more formatting, before it is saved as a png, where the name depends on the previously fetched location and number of days sampled.

```
def dayLinePlotter(wild, days): #Produces a nice line graph
    listy = Sql.fetcher(wild,days,"xy")           #Calls the data fetcher method
    # Plot the Ratings.                           # from sqller class

    fig, ax = plt.subplots()
    ax.plot(listy[0], listy[1], c='red', alpha=0.5) #Plots x and y lists

    # Format plot.
    #fetches the specified title and location from the argument via sql
    titlelocation = Sql.titlLocFetch(wild, days, listy[2])
    plt.title(titlelocation[0], fontsize=13.5)      #Fetches and creates location

    plt.xlabel('Day', fontsize=16)                 #Matplot x axis label
    fig.autofmt_xdate()                           #Angles x axis text to avoid text overlap

    plt.ylabel('Rating', fontsize = 16)            #Matplot y axis label
    plt.tick_params(axis='both', which='major', labelsize=12)

    plt.grid(True)
    plt.tight_layout()
    #savename
    dayer = str(days)
    if wild >= 1:                                  #Creating a savename
        savename = titlelocation[1] + f"-Line{dayer}Days.png" # from days, lacoation,
        plt.savefig('static/'+savename)             # and console and save it
    else:
        plt.savefig(f'static/AllLine{dayer}Days.png') #Same, but for for all
    #plt.show()                                     # consoles
    plt.close()                                     #File is closed
```

#### 5.2.2.4.4. *Plotting program – Pie plot*

Like the previous function, this one simply calls the sql fetcher method with the keyword y for pie plot, and plots the returned data by first defining the sizes. Now that this is a pie plot, it plots slices of a pie, and you define the size of each percentage. In the case of the program the sizes are determined by the number of ratings per type: ones, twos, and threes.

As with the line plotter, this one is also formatted, the slices are labelled, and we've used a bit of the "explode" to highlight the high ratings, which pulls it further out of the centre of the pie than the other ones.

Like the line plot, this plot is also given a title and saved under a name based on the location and number of days.

```
def piePlotter(wild, days):

    x = Sql.fetcher(wild,days, "y")           #Calls the data fetcher method
                                           # from sqller class

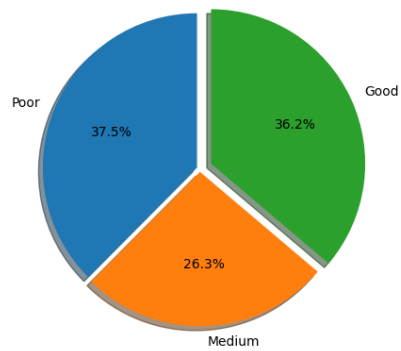
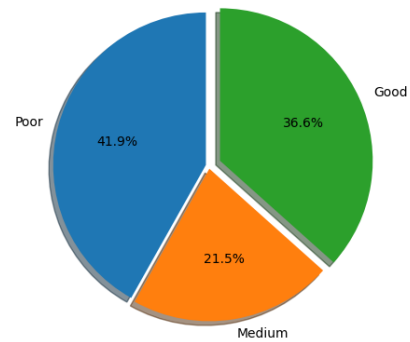
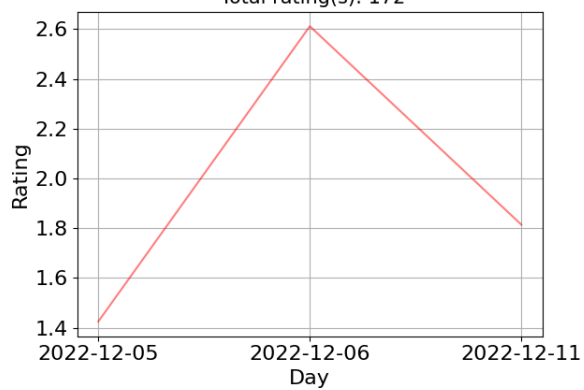
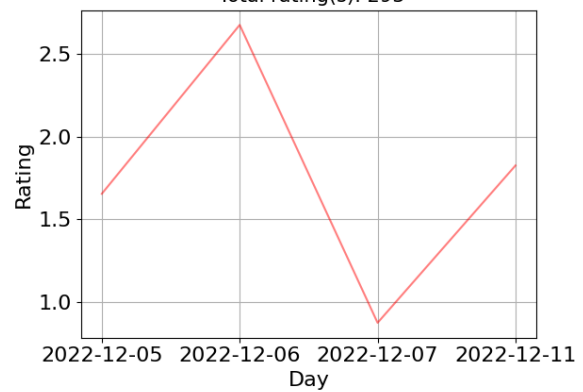
    count = x[0] + x[1] + x[2]               #Counting total ratings
    labels = 'Poor', 'Medium', 'Good'
    poor = x[0]                             #Separating list for readability
    medium = x[1]
    good = x[2]

    sizes = [poor, medium, good]             #Plotting the sizes.
    explode = (0.02, 0.02, 0.08)

    fig1, ax1 = plt.subplots()              #Matplotlib settings/magic
    ax1.pie(sizes, explode=explode, labels=labels, autopct='%1.1f%%',
            shadow=True, startangle=90)
    ax1.axis('equal')
    #fetches the specified terminal from the argument via sql
    titlelocation=Sql.titlLocFetch(wild,days,count)
    plt.title(titlelocation[0], fontsize=15) #Fetching and creating title from module
    plt.subplots_adjust()
    #savename
    dayer = str(days)

    if wild >= 1:                            #If a console is specified, a specific name
        savename = titlelocation[1] + f"-{dayer}DaysPie.png" # is created
        plt.savefig('static/'+savename)
    else:
        plt.savefig(f'static/All{dayer}DaysPie.png')         #Saved as All consoles name
    #plt.show()
    plt.close()                               #File is closed
```

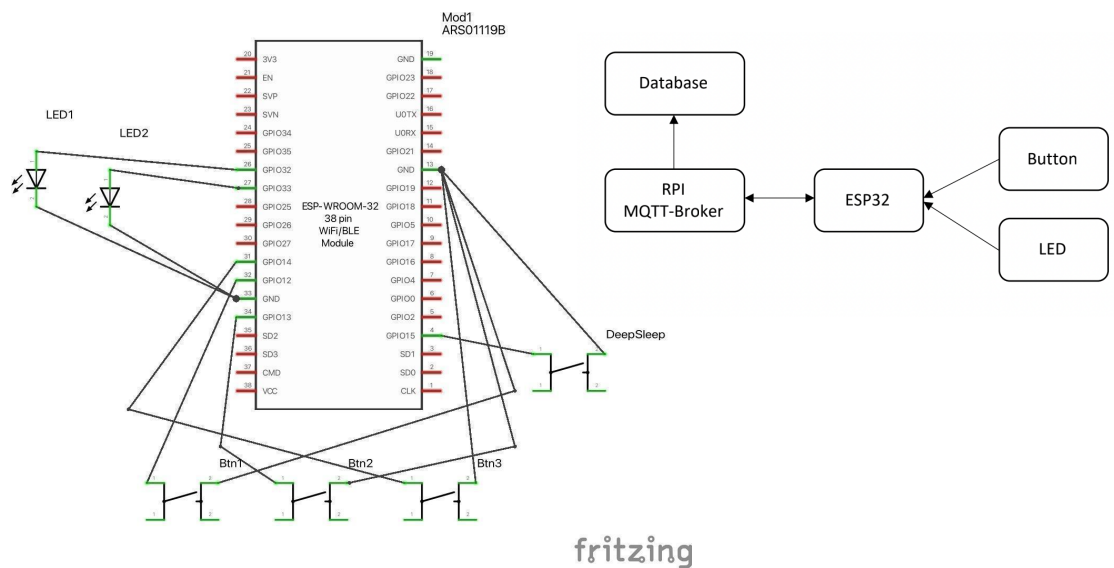
## OUTPUT :

Ratings from all locations for the past 365 day(s)  
Total rating(s): 293Ratings from classroom: SH-A2.09 for the past 365 day(s)  
Total rating(s): 172Ratings from classroom: SH-A2.09 for the past 30 day(s)  
Total rating(s): 172Ratings from all locations for the past 30 day(s)  
Total rating(s): 293

## 5.3. Console

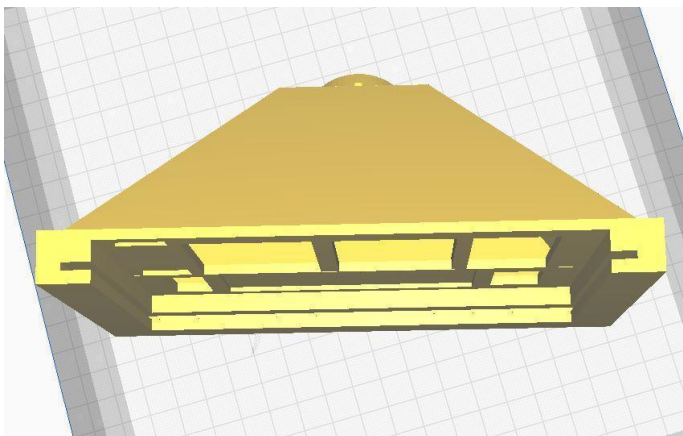
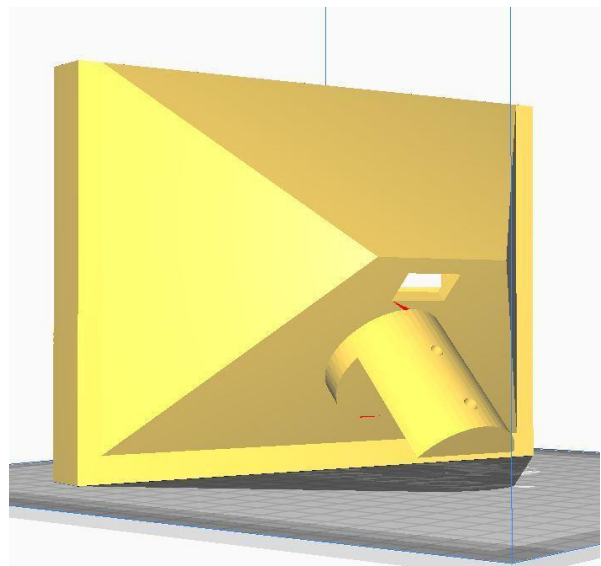
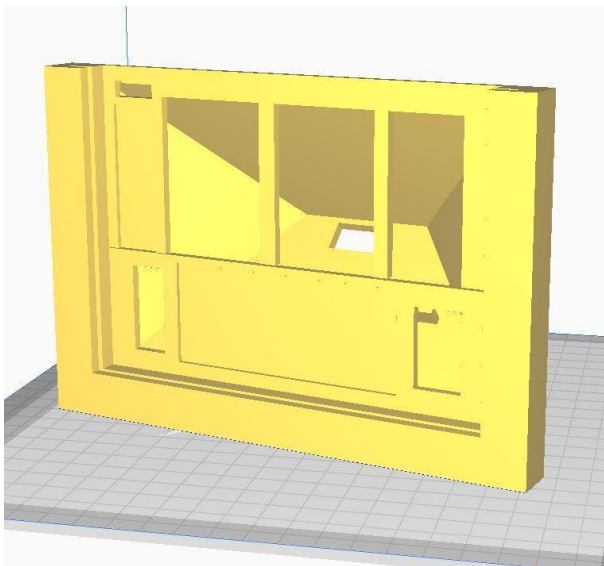
The console is our product and therefore the heart of this project. In order to get the product with the best performance and a modern look, we had to test different software and many different designs. In this part, we take you through the whole process.

### 5.3.1. Module design



### 5.3.1.1. Physical console

The physical console was designed in Autodesk Fusion 360 with the paper holder as the primary focus. The case behind it had to have enough space for wires, cables and the power bank. A small hole was also made in the back for easier access to a charging cable. On our prototype this is also used to house the sleep button and an extension cable that allows to quickly shut off the esp32, as the sleep mode is set to a brief period for demonstration, instead of the realistic off period that would keep the device in a low power state from the building closing until it opens. There is also a plate that sits behind the buttons to act as support, so the buttons can be pressed properly. Two screw holes are part of the design on the neck, if you will, so that the holder could be attached to a pole with a base.



### 5.3.2. Module implementation

Before we started writing the software, creating the console itself was our priority. We opted for a 3D printed design, because of the high flexibility and modification choices. Our vision was a 45° angled design that we could mount on a wooden podium. Our first design was surprisingly functional and well printed, so we had no reason to print a second one. Afterwards we focused on writing and testing our console software.

Our console is button triggered and sends the data via MQTT to our Raspberry-Pi. It was important that the console is autonomous and works with very low power consumption. Furthermore, we had to set up the console so that it works without WIFI connection.

#### 5.3.2.1. MQTT and buttons

Firstly, we made sure that our ESP32 works flawlessly with the MQTT and that the buttons send the correct rating.

```
const char* ssid = "group4_IoT";
const char* password = "group4four";
const char* mqtt_server = "10.120.0.57";
```

```
WiFiClient espClient;
PubSubClient client(espClient);
unsigned long lastMsg = 0;
#define MSG_BUFFER_SIZE (50)
char msg[MSG_BUFFER_SIZE];

const char* mqtt_user = "group4";
const char* mqtt_passwd = "group4four";
```

In this part we set up the MQTT. We connect to our private network "group4\_IoT" and to our MQTT-broker "10.120.0.57". Because the MQTT-broker is password protected we had to define the MQTT-user – password.

```
#define button1 12
#define button2 13
#define button3 14
#define led1 32
#define led2 33
```

The buttons 1 – 3 we are using to capture the rating and the two LED's we are using to show on which step the customer is on.

```
pinMode(button1, INPUT_PULLUP);
pinMode(button2, INPUT_PULLUP);
pinMode(button3, INPUT_PULLUP);
pinMode(led1, OUTPUT);
pinMode(led2, OUTPUT);

Serial.begin(115200);
setup_wifi();
client.setServer(mqtt_server, 1883);
client.setCallback(callback);
```

In the setup() we use the pinMode method to initialize the buttons and LED's. Furthermore, we use the given code to initialize the MQTT.

```
void loop() {
  digitalWrite(led1,HIGH);
  if (digitalRead(button1)==LOW){
    digitalWrite(led1,LOW);
    digitalWrite(led2,HIGH);
    String status1 = "1";
    status1.toCharArray(msg, MSG_BUFFER_SIZE);
    client.publish("Topic", msg);
    delay(3200);
    digitalWrite(led2,LOW);
    digitalWrite(led1,HIGH);
  }
```

We started our program that if the buttons get pressed, we send the rating over MQTT to our database. The status had to be converted into a char array, because MQTT only allows chars as data.

Once we got it working, we wanted to define the topic globally to modify it with ease. Additionally, we wanted to send a globally defined device number to identify which console sent which rating.

```
String topic = "Project3/Console";
```



```
String deviceNumber = "1";
```

```
#define bufferSize (50)
```

```
char topicArray[bufferSize];
```

```
char macArray[bufferSize];
```

Due to the fact that we need to convert the topic into a char array, we created a second array to store our topic.

```
String status1 = "1&";
```

```
status1 += (String) WiFi.macAddress();
```

```
status1 = status1 + "&" + deviceNumber;    //Our message
```

```
status1.toCharArray(msg, MSG_BUFFER_SIZE);
```

```
topic.toCharArray(topicArray, bufferSize); //Topic -> char array
```

```
client.publish(topicArray, msg);
```

For even more security we added the MAC-address to the message. We use “&” in-between the parts for the python program so it can split the parts easily. Now our message has this format:

The message: *Buttondata&MAC-Address*

We publish this message on our globally defined topic. The customer can communicate with us, which logical topic he wants and we only need to change it once. It gives us and the customer high flexibility.

### 5.3.2.2. *Offline-mode*

After we successfully made our console to work with MQTT our focus was on the offline-mode. We had to make sure that our console works even without the MQTT connection. For that reason, we implemented another if-statement into the loop-method.

```
void loop() {
  digitalWrite(led1,HIGH);
  if (!client.connected()) {
    //code
  }else {
    //code
  }
}
```

We check inside the loop method if the client is connected to MQTT. If it is connected, we execute the code as stated in the chapter above. But if the console is not connected, we store the data offline.

```
#define sizeDisc (1000)
String disconnectArray[sizeDisc];
int arrayIndex = 0;      //Index for the array
int index2 = 0;          //Index for the for-loop
```

Having said that we need to store the data somewhere, we created an array. This array gets filled if the console couldn't make a connection with the MQTT.

```
void loop() {
  digitalWrite(led1,HIGH);

  if (!client.connected()) {
    if (digitalRead(button)==LOW){
      String status1 = "1&";
      status1 += (String) WiFi.macAddress();
      status1 = status1 + "&" + deviceNumber; //Our message
      disconnectArray[arrayIndex] = status1; //Adds the message to array
      arrayIndex++; //Counts the number of array entries
      index2 = 1; //Info that there is data in the array
      reconnect(); //Checks connection after every button press
    }
  }
}
```

This part will be executed if there couldn't be a MQTT connection established. The program checks each time the button is pressed if there is still no MQTT connection.

```
void loop() {
    digitalWrite(led1,HIGH);
    if (!client.connected()) {
        //code
    }else {
        for(int i = 0; i < index2; i++){//With index2 it checks if there is data
            Offline_Handler();           stored in the array.
        }    //index2 = 1 -> calls Offline_Handler()
    //code           index2 = 0 -> ignores this loop
}
```

Once there is a connection it will reach the for loop. If there is no data in the array, index2 will be 0 and the loop will be ignored. If we have data in the array, index2 will be 1 and the loop will be executed and the Offline\_Handler() method gets called.

```
void Offline_Handler(){
    for(int i = 0; i <= arrayIndex; i++){ //loops through array
        String message = disconnectArray[i]; //takes out the message
        message.toCharArray(msg, MSG_BUFFER_SIZE); //Message -> char array
        deviceTopic.toCharArray(topicArray, bufferSize); //Topic -> char array
        client.publish(topicArray, msg); //Publishes the message
    }
    memset(disconnectArray, 0, sizeDisc); //clears the array
    arrayIndex = 0;           //to know that there is no more data in the array
    index2 = 0;    //makes sure that we don't execute the Offline_Handler
                  every time
}
```

The purpose of the Offline\_Handler() is to take each data entry out of the and publish it with MQTT. In order to do so, we loop thorough the array and publish each entry one by one. After it has published the data, we need to clear the array for the future. For that reason we use the memset() method and set both indexes to 0.

### 5.3.2.3. Deep-sleep mode

Because we want a product that is autonomous for a longer period and work with low power consumption, we implemented a deep-sleep mode. This deep-sleep mode can be triggered by a 4<sup>th</sup> button. It allows the customer to turn off the console during closing hours. The amount of time is predefined in the software but can be changed easily.

```
int numbOfHours = 5;
int numbOfHoursConv = Hours * 3600;
int conversionHours = 1000000;

#define button4 15
    //code
void loop() {
    //code
    else if (digitalRead(button4)==LOW){
        Serial.println("sleep");
        esp_sleep_enable_timer_wakeup(numbOfHoursConv * conversionHours);
        esp_deep_sleep_start();
    }
```

Once the button is pressed the console will shut off all everything besides the UltraLowPower (ULP) Coprocessor. This allows the console to save power, but still be able to turn on automatically after the hours have passed. (11.3 Console software)

### 5.3.3. Formal module test

We started by testing the core functions of the ESP32, such as the MQTT connection and the buttons / LEDs. For that purpose, we used the MQTT-box and checked if there were messages on the specific topic.

After the successful integration of the core function, we focused on testing the offline-mode. By that time, we already started using our RPI as the broker. To test it, we switched off the RPI and checked on the serial-monitor if the data gets stored offline. In our experience the easiest way to check if there were any blocking functions, we used many print statements at different stages of the software.

## 5.4. Webpage

### 5.4.1. Module design

The design choices are made with the consideration of easy navigation and clear web design. The main purpose of the web page is to make the user reach the data in the most effortless way. We tried a few design options such as buttons with hover effect and color changes when the cursor is on the buttons. Additionally, we used color palettes for the combinations for aesthetic purposes.

webpage-project-tree

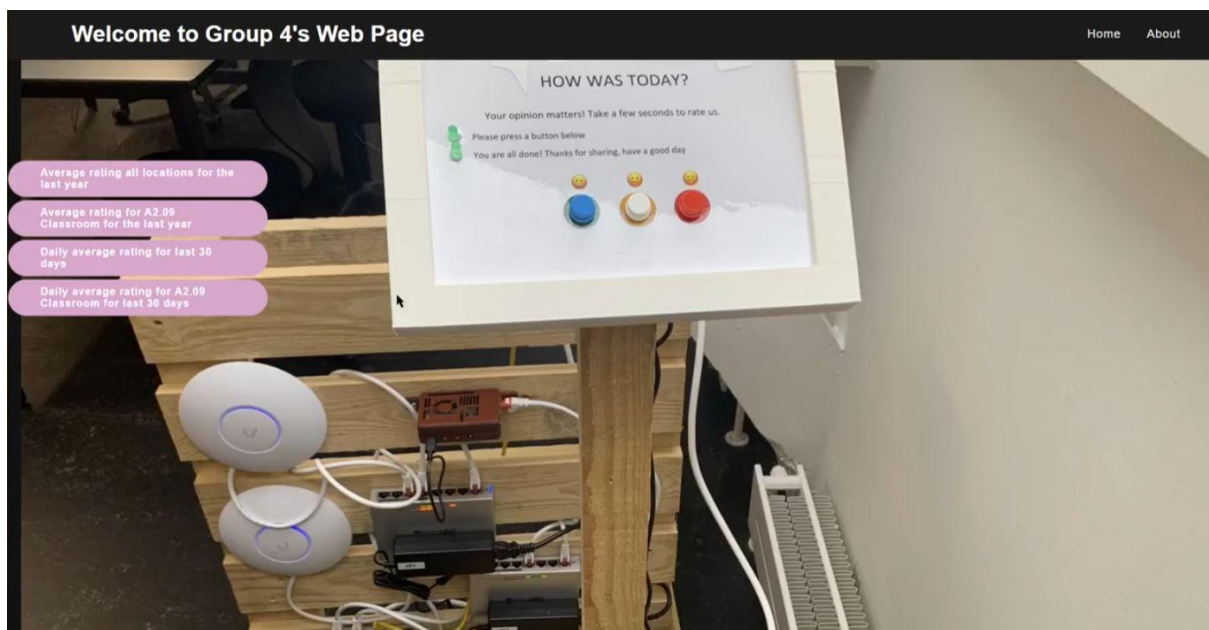
```

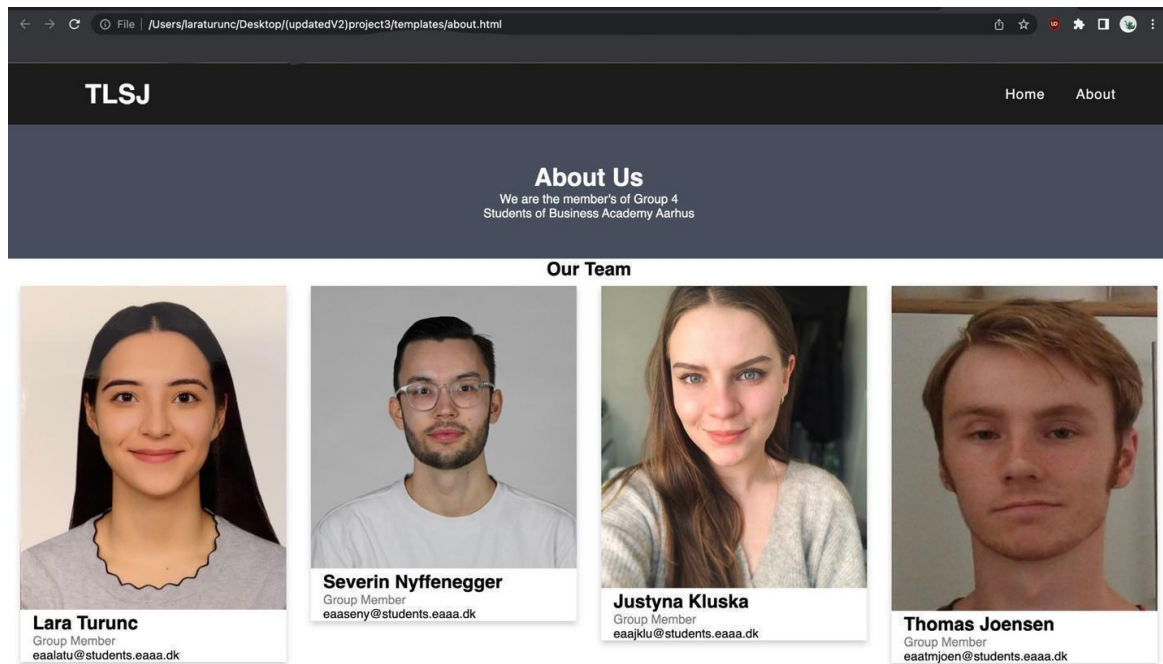
├─ app.py
├─ static

|   └─ Graphics
|   └─ Plots
├─ templates
    └─ index.html
    └─ All365DaysPie.html
    └─ AllLine30Days.html
    └─ SH-A2.09-365DaysPie.html
    └─ SH-A2.09-Line30Days.html
    └─ about.html
  
```

Page designs:

*Front page*





About Page



Plot page template

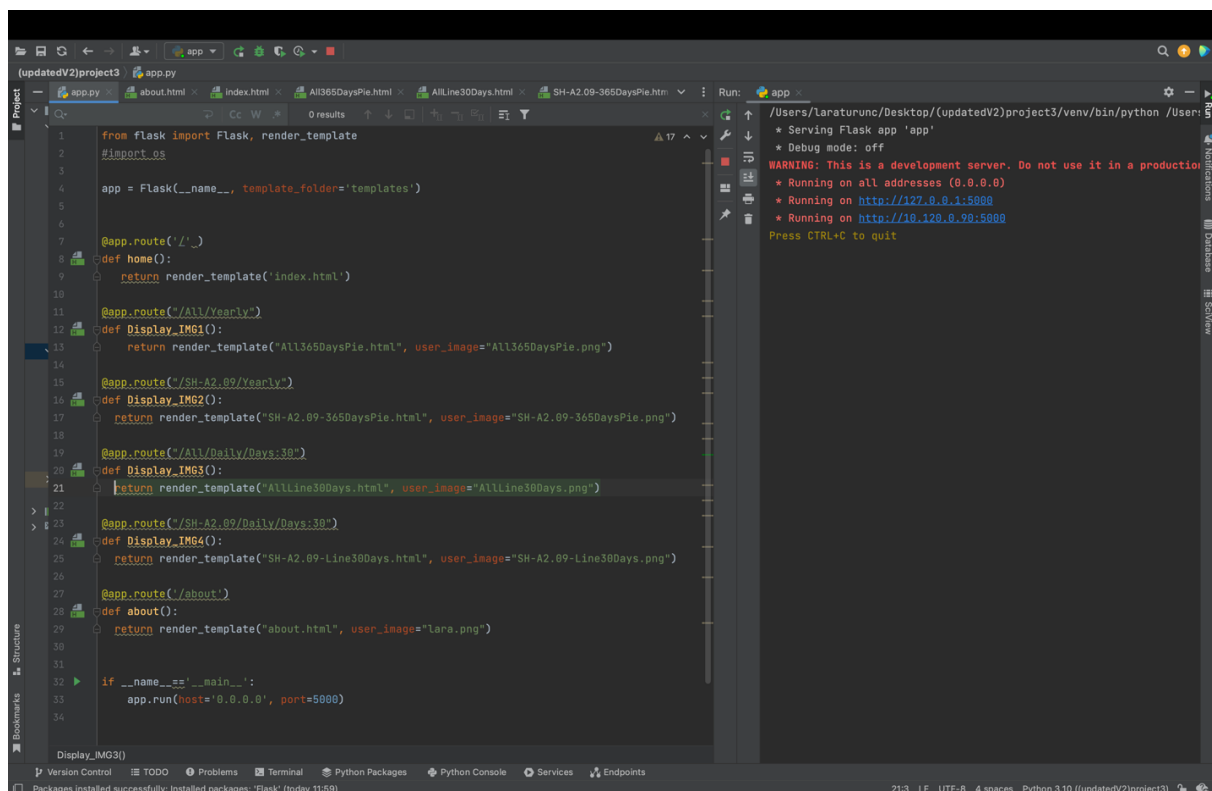
## 5.4.2. Module implementation

As most other web pages, our pages were created by using HTML and CSS, and out of convenience, we wrote the CSS at the of the HTML files rather using .CSS files to be imported as headers.

We made the web page to work with buttons for four different survey results. First two buttons link to pages which display the results of the last year's average ratings for all locations and one specific location, SH-02.09, our classroom. Third and fourth buttons link to pages that display the results of the daily average ratings for the last 30 days, again for all consoles and our classroom specifically.

As the flask web server uses images that are dumped in the 'static' directory, which is a bit mislabeled as we also use the directory dynamically, the webserver has no direct link to the database or data visualization software.

We installed the flask package to be used with the system's python interpreter on our server. Flask is bundled with an interpreter named Jinja2 which gives us the opportunity to include variables, create loops and statements right in the html templates. With that in mind, Flask also provides its hallmark `render_template()` helper function that allows use of Jinja template engine. For flask, we created two new folders inside our project folder: static and templates. The 'static' folder stores our images while the 'templates' folder holds our HTML templates, as they are considered with the flask framework. This is a crucial detail since the directory tree and meticulous names are important to Flask.



```
1 from flask import Flask, render_template
2 #import os
3
4 app = Flask(__name__, template_folder='templates')
5
6
7 @app.route('/')
8 def home():
9     return render_template('index.html')
10
11 @app.route('/All/Yearly')
12 def Display_IMG1():
13     return render_template("All365DaysPie.html", user_image="All365DaysPie.png")
14
15 @app.route("/SH-A2.09/Yearly")
16 def Display_IMG2():
17     return render_template("SH-A2.09-365DaysPie.html", user_image="SH-A2.09-365DaysPie.png")
18
19 @app.route("/All/Daily/Days:30")
20 def Display_IMG3():
21     return render_template("AllLine30Days.html", user_image="AllLine30Days.png")
22
23 @app.route("/SH-A2.09/Daily/Days:30")
24 def Display_IMG4():
25     return render_template("SH-A2.09-Line30Days.html", user_image="SH-A2.09-Line30Days.png")
26
27 @app.route('/about')
28 def about():
29     return render_template("about.html", user_image="lara.png")
30
31
32 if __name__ == '__main__':
33     app.run(host='0.0.0.0', port=5000)
34
35 Display_IMG3()
```

Run: app

- \* Serving Flask app 'app'
- \* Debug mode: off
- WARNING: This is a development server. Do not use it in a production
- \* Running on all addresses (0.0.0.0)
- \* Running on <http://127.0.0.1:5000>
- \* Running on <http://10.120.0.90:5000>

Press CTRL+C to quit

21:3 LF UTF-8 4 spaces Python 3.10 (updatedV2|project3)

### 5.4.3. Formal module test

Testing a simple web server isn't a terribly complex matter. We started testing HTML files by running them one by one to ensure the CSS styling and html markup was looking as we intended and ensure that the buttons had the jinja function calls in their hyperlinks and that static images were displaying correctly. After that, we ran the actual webserver and tested the function of all hyperlinks and dynamic images. This is done simply by clicking on the hyperlinks and renaming the images to see if they disappear upon reload and reappear upon backtracking the steps.

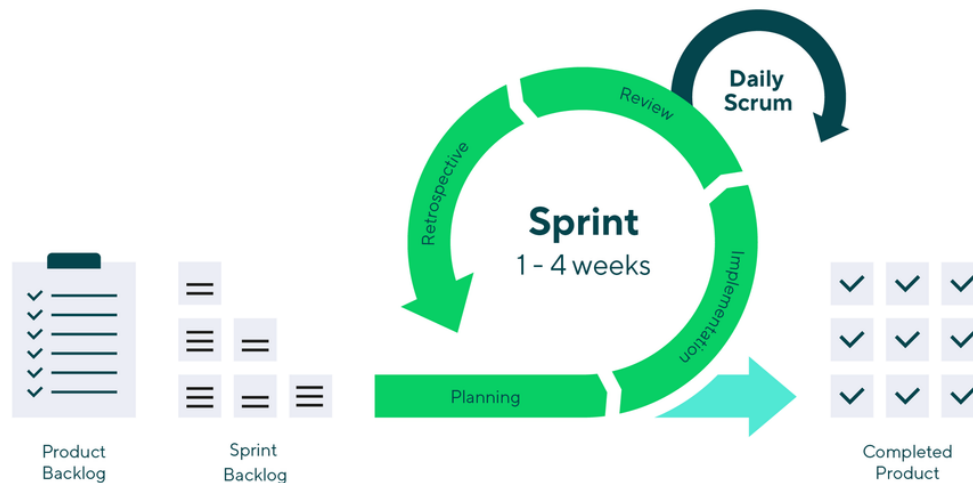
## 5.5. System Integration test

Throughout the two sprints, we tested each component individually. After we got the console and the database working, we connected them and tested if they work together. With the implementation of the webpage, our product was almost finished. The focus now laid on testing the product on our own private network. With a few changes to our console software and the python program, we performed the system integration test.

Our network was set up correctly, however we had some trouble with the plotter creating images correctly on the RPI. At first the RPI couldn't find the necessary directories, but with changing the cursor's location, it was able to access them. After switching flask to port 80, we were forced to run it as a super user. The last check was if the webpage updated the plots as it should and it did so.



## 5.6. Project management



In order to efficiently manage the process of working on our project, we relate to the Scrum framework for our software development part and for the management of the project we used a classic framework. The scrum method works perfectly for small teams like ours. Every project day the members working on the software part started with a short 5 - 10 minutes long scrum meeting about our progress, challenges and future development. Carrying out a sprint helped us to achieve our goals in a shorter time frame. During the whole project work we carried out 2 sprints lasting 2 weeks each.

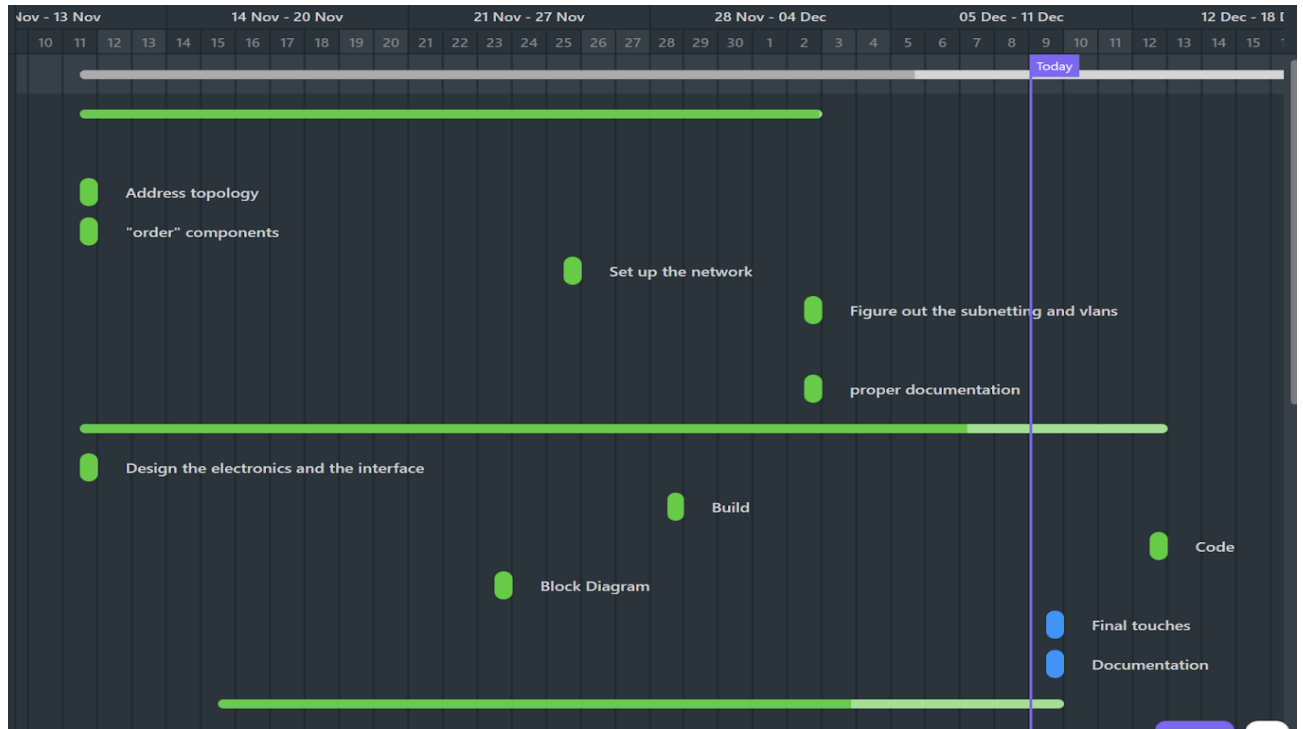
In our first sprint we focused on writing the software for our console and the MariaDB. Severin wrote the working code for the console and the original “MQTT receiver → SQL inserter” program. Thomas rigged up the network and set up the port rules, subnets and wifis, and designed, printed and built the console holder. Justina and Lara made the physical design of the face of the console. The implementation and setting up of the MariaDB was done by Lara. It was important to us that we test the two components together in our first sprint. We had to make sure that the data from a button click gets saved on our MariaDB, and with it working our first sprint ended.

The second sprint was focused on writing the code for our webpage and all the plotting programs used with it. Thomas focused on the plotter part and Lara focused on the webpage. We got a little bit overwhelmed with the communication between the database and our plotter function. After some intense testing and rewriting of the code, the bugs in the plotting program were ironed out, and the plots would look right, and work as intended with our webpage. With that, our second print ended.

To organize our tasks in the timeframe, we used Clickup application. We used it to divide tasks between our team members, organize them in the time and keep checking if we are on time. The tool helped us to get a good overview of the project, the big help were dependencies, so we could see which tasks should be done first before we continue with

other ones. Moreover, we made personal log books that we updated after every project day. Each of us wrote down what we did on a given day, and what we plan to do next time.

Titles such as product owner and scrum master have been rather fluid given the small team, with each member in part being the owner of their own products, though with Thomas inching towards both on all fronts of development. Often taking on the role of grease in dry gears, trying to help out where it was needed, regardless of area.



We mostly used the list view, so our gantt is not the prettiest chart.

Network

+ NEW TASK

HIDE CLOSED

COMPLETE 7 TASKS

	ASSIGNEE	DUE DATE	COMMENT	PRIORITY	STATUS
✓ "order" components	T	Nov 11	-	High	Review
✓ Address topology	TS	Nov 11	-	High	Review
✓ Make a rig for the network that works with the server	T		-	Medium	Review
✓ Documentation	-		-	Medium	-
✓ Figure out the subnetting and vlans	-	Dec 2	-	Medium	-
✓ proper documentation	T	Dec 2	-	High	-
✓ Set up the network	TJ	Nov 25	Hard dependency	Medium	Progressing

+ New task

TO DO 1 TASK

HIDE CLOSED

	ASSIGNEE	DUE DATE	COMMENT	PRIORITY	STATUS
✓ make website reachable from itek	JT		-	Medium	-

+ New task

Console

+ NEW TASK

HIDE CLOSED

COMPLETE 0 TASKS

	ASSIGNEE	DUE DATE	COMMENT	PRIORITY	STATUS
✓ Design the electronics and the interface	all	Nov 11	-	Medium	Review
✓ Block Diagram	JK	Nov 23	-	Medium	Review
✓ Build	T	Nov 28	-	Medium	Review
✓ Code	-	4 days ago	-	Medium	Review
✓ Final touches	ST	Dec 9	-	Medium	Review
✓ Documentation	-	Dec 9	-	High	-

+ New task

COMPLETE 8 TASKS

HIDE CLOSED

	ASSIGNEE	DUE DATE	COMMENT	PRIORITY	STATUS
✓ Make a design for an rpi server case	T		-	Medium	Review
✓ Build the box	T		-	Medium	Review
✓ Install MariaDB and LAMP on rpi	L	Nov 15	-	Medium	Review
✓ set up database table	-	Nov 25	-	Medium	-
✓ Uninstall and replace LAMP with flask	-	Nov 25	-	Medium	-
✓ correct network setup	-		-	Medium	-
✓ Final touches	-	Dec 9	-	Medium	Review
✓ Documentation	-	Dec 9	-	High	-

+ New task

Program for server

+ NEW TASK

ADD DESCRIPTION

ADD COMMENT

HIDE CLOSED

COMPLETE 5 TASKS

	ASSIGNEE	DUE DATE	COMMENT	PRIORITY	STATUS
✓ MQTT receiver // SQL inserter	-		will depend on progress in ...	Medium	-
✓ CSS template	LJ		-	Medium	-
✓ SQL extractor // HTML png updater	T	4 days ago	-	Medium	-
✓ Invent modules	-		-	Medium	-
✓ Documentation of both programs	-		-	High	-

+ New task

Planning

+ NEW TASK

HIDE CLOSED

COMPLETE 1 TASK

	ASSIGNEE	DUE DATE	COMMENT	PRIORITY	STATUS
✓ Initial plan	T	Nov 11	-	High	Progressing

+ New task

Logging

+ NEW TASK

HIDE CLOSED

TO DO 2 TASKS

	ASSIGNEE	DUE DATE	COMMENT	PRIORITY	STATUS
✓ Running : [ 1 ]	-	Today	-	Medium	-
✓ Note down what we do	all	3 days ago	-	Medium	-
✓ Report	all	Today	-	Medium	-

+ New task

This view has unsaved changes

To save, press **Ctrl + Enter** or click the **Save** button. Saving updates the view for everyone.

Revert Autosave view **Save Ctrl+Enter**

## 6. Prove

<https://youtu.be/vhpy-jEkaEM>

<https://youtu.be/t3Lsml3XJe4>

## 7. Conclusion

To sum up, we have managed to finish the project with a functional survey console that we consider satisfactory under the given criteria. We can save and update the data in a database through the MQTT broker when a user clicks a button on the console. You can keep track of the overall data in four different ways from the web page. Although we had some challenges during the project, we were able to complete it before the deadline. We worked on the project during and after the scheduled school hours. Both workload and environment around the group were always positive since every member of the group was communicative.

## 8. Reflection

During working on this project, we used SQL, html, RPI, and ESP32, access points and router. We built the console on our own. Our group consisted of 4 people. Each of us has a bit of a different experience to share with others. The cooperation in the group went very well and we tried our best to be sure that everyone knew what was going on at all times. For managing the project in time, we used the Clickup tool, which helped us with tasks division and time estimations. We also wrote our personal logbooks to have the overview of how the work is going and wisely plan the next steps. At the end we manage to provide the final working product which is the satisfaction measurement console.

## 9. Appendices

### 9.1. MQTT setup

```
$ cat /etc/mosquitto/mosquitto.conf
# Place your local configuration in /etc/mosquitto/conf.d/
#c
# A full description of the configuration file is at
# /usr/share/doc/mosquitto/examples/mosquitto.conf.example

per_listener_settings true

pid_file /run/mosquitto/mosquitto.pid

persistence true
persistence_location /var/lib/mosquitto/

log_dest file /var/log/mosquitto/mosquitto.log
include_dir /etc/mosquitto/conf.d

allow_anonymous false
listener 1883
password_file /etc/mosquitto/passwd

$ sudo mosquitto_passwd -c /etc/mosquitto/passwd [username]
$ sudo mosquitto_passwd /etc/mosquitto/passwd [username2]

$ cat /etc/mosquitto/passwd
thomas:$7$101$1eFPOBxN0ZB4HpIR$8ip+FQUhaVyCzWHyJws2CXBVCTRAH5wlcRO378Uuk1wdNW0KZwbzpmvxyypJQ4GgfmouXOLIo7YISzM
8PcQYcNQ==
group4:$7$101$SC6Bml4clMmUGROb$cKfVxSD/76huae5wjcteQ2bx5AU9ya5cKKL9wWCV7r2fMQN+FLLn4pRAEcMW3Ttu3ZUjVYZa0aK84yr
ypDVqww==

term 1: $ mosquitto_sub -d -t dimple/# -u group4 -P group4four
term 2: $ mosquitto_pub -d -m "foiyuoooo" -t dimple/johnnie/barnacle/jr/ -u group4 -P group4four
term 1: Client (null) received PUBLISH (d0, q0, r0, m0, 'dimple/johnnie/barnacle/jr/', ... (9 bytes))
foiyuoooo

term 3: $ mosquitto_pub -h 10.120.0.57 -m "foiyuoooo" -t dimple/johnnie/barnacle/jr/ -u group4 -P group4four
term 4: $ mosquitto_sub -h 10.120.0.57 -t dimple/# -u group4 -P group4four
term 4: foiyuoooo
```

## 9.2. Python

### 9.2.1. SQL Module

class Sqller:

```
def __init__(self,conn,cur):
    self.__cur = cur
    self.__conn = conn
    self.__cur.execute("SELECT macAddress from group4.locations ORDER by id")
    self.__macList0 = self.__cur.fetchall()
    self.macList = []
    for mac in self.__macList0:
        self.macList.append(mac[0])
    del(self.__macList0)
    print("known mac addresses: ", self.macList)
```

```
def inserter(self, rating, macAddress, known):
    if known == True:
        self.__cur.execute("INSERT INTO group4.console( rating, macAddress)
VALUES(?,?)", (rating, macAddress))
    if known == False:
        print("unknown mac: ", macAddress, " inserting in group4.locations")
        self.__cur.execute("INSERT INTO group4.locations(location, macAddress)
VALUES(?,?)", (rating, macAddress))
        print("updated mac addresses: ", self.macList)
```

```
def fetcher(self, wild, days, xy): #wildcard for console number, 0 is all, day count, x
or y for plot
```

```
if xy == "y":
```

```
if wild >= 1:
    wild += -1
```

```
        self.__cur.execute(f"SELECT rating FROM group4.console WHERE  
macAddress='{self.macList[wild]}' and ts >= DATE_SUB(CURDATE(), INTERVAL {days}  
DAY) ")
```

```
    else:
```

```
        self.__cur.execute(f"SELECT rating FROM group4.console WHERE ts >=  
DATE_SUB(CURDATE(), INTERVAL {days} DAY)")
```

```
    self.__ratings = self.__cur.fetchall()
```

```
    self.__ratings2 = self.algore2(self.__ratings)
```

```
    return self.__ratings2
```

```
elif xy == "xy":
```

```
    if wild >= 1:
```

```
        wild += -1
```

```
        self.__cur.execute(f"SELECT ts, rating FROM group4.console WHERE  
macAddress= '{self.macList[wild]}' and ts >= DATE_SUB(CURDATE(), INTERVAL {days}  
DAY) ")
```

```
        # COULD ORDER BY INDEX NUMBER ON RPI, DATETIME IS OUT OF ORDER
```

```
    else:
```

```
        self.__cur.execute(f"SELECT ts, rating FROM group4.console WHERE ts >=  
DATE_SUB(CURDATE(), INTERVAL {days} DAY) ")#ORDER BY idx")
```

```
        self.__raw = self.__cur.fetchall()
```

```
        self.__raw.sort()
```

```
        listy = self.algoreithm( self.__raw)
```

```
        return listy
```

```

def titlLocFetch(self, wild, days, count):

    if wild >= 1:
        wild += -1
        self.__cur.execute(f"SELECT location FROM group4.locations where
macAddress = '{self.macList[wild]}'") # rating from group4.console topic like [insert
topic] and mac like [insert mac])
        self.__loc = self.__cur.fetchall()

        self.__loc1 = ""
        self.__loc1 = str(self.__loc[0]).strip("{}{,}")

        self.__title = f"Ratings from classroom: {self.__loc1} for the past {days} day(s)\n
Total rating(s): {count}" #*****

        #From all terminals
    else:
        self.__title = f"Ratings from all locations for the past {days} day(s)\n Total
rating(s): {count}" #*****
        self.__loc1 = ""

    setty = []
    setty.append(self.__title)
    setty.append(self.__loc1)

    return setty

```

```

def algoreithm(self, listy):
    #Getting X *****
    self.x = []
    self.y = []

```



```
self.TsWithTIME = []
self.TsWithoutTIME = []
self.ratings = []
self.rating = 0.0

for row in listy:

    self.current_Ts = row[0].strftime('%Y-%m-%d-%H-%M')
    self.current_day = row[0].strftime('%Y-%m-%d')
    self.rating = row[1]
    self.TsWithTIME.append(self.current_Ts)
    self.TsWithoutTIME.append(self.current_day)
    self.ratings.append(self.rating)

del(self.rating, self.current_day)

self.ratingCount = len(self.TsWithTIME)

#making a version of TsWithoutTIME without duplicates
self.x = list(set(self.TsWithoutTIME))

#Getting Y *****
#Creating empty list

self.ytemp3 = []

for i in self.x:
    self.ytemp1 = []
    for u in self.TsWithTIME:
        if i in u:
```

```
        self.idx = self.TsWithTIME.index(u)

        self.ytemp0 = self.ratings[self.idx]
        self.ytemp1.append(self.ytemp0)

        self.ytemp1sum = sum(self.ytemp1)
        self.ytemp1len = float(len(self.ytemp1)) + 1

        self.ytemp2 = self.ytemp1sum/self.ytemp1len

        self.ytemp3.append(self.ytemp2)

    self.y = self.ytemp3

    self.x, self.y = zip(*sorted(zip(self.x,self.y)))
    self.lister = [self.x,self.y, self.ratingCount]

    return self.lister

def algore2(self, x):

    self.__bad = 0
    self.__mid = 0
    self.__good = 0

    # For this one I simply need the amount of 1's 2's and 3's,
    # so I simply use booleans as the type, in hope of cutting processing time
    for i in x:

        if i[0] == 1:
            self.__bad += 1
```

```

    if i[0] == 2:
        self.__mid += 1
    if i[0] == 3:
        self.__good += 1

    self.__returnable = [self.__bad, self.__mid, self.__good]
    return self.__returnable

```

### 9.2.2. Plot program

```

def main(now,then):
    while 1:
        now = dt.now()

        if (now - then).total_seconds() >= 10:
            print("plot cycle started")
            try:
                then = dt.now()
                #Fetch lists of time stamps and ratings and plot them
                dayLinePlotter(0,30)

                #Fetch lists of time stamps and ratings for console1 and plot them
                dayLinePlotter(1,30)

                #Fetch list of ratings and plot them
                piePlotter (0,365)

                #Fetch list of ratings for console1 and plot them
                piePlotter (1,365)
                #^^ all of these take arguments: wildcard for the console, amount of days
            back
        except:
            print("error in plotting")
            now = dt.now()

            sleep(0.01)

```

```
        print("plot cycle ended")
        then = now

def dayLinePlotter(wild, days): #Produces a nice line graph

    listy = Sql.fetcher(wild,days,"xy")

    # Plot the Ratings.
    fig, ax = plt.subplots()

    ax.plot(listy[0], listy[1], c='red', alpha=0.5)

    # Format plot.

    #fetches the specified title and location from the argument via sql
    titlelocation = Sql.titLocFetch(wild, days, listy[2])

    plt.title(titlelocation[0], fontsize=13.5)

    plt.xlabel('Day', fontsize=16)
    fig.autofmt_xdate()

    plt.ylabel('Rating', fontsize = 16)
    plt.tick_params(axis='both', which='major', labelsize=12)

    plt.grid(True)
    plt.tight_layout()
    #savename
```

```
dayer = str(days)
if wild >= 1:
    savename = titlelocation[1] + f"-Line{dayer}Days.png"
    plt.savefig('static/'+savename)
else:
    plt.savefig(f'static/AllLine{dayer}Days.png')
#plt.show()
plt.close()

def piePlotter(wild, days):

    x = Sql.fetcher(wild,days, "y")

    count = x[0] + x[1] + x[2]

    labels = 'Poor', 'Medium', 'Good'
    poor = x[0]
    medium = x[1]
    good = x[2]

    sizes = [poor, medium, good]
    explode = (0.02, 0.02, 0.08) # only "explode" the 3rd slice ('good')

    fig1, ax1 = plt.subplots()
    ax1.pie(sizes, explode=explode, labels=labels, autopct='%1.1f%%',
            shadow=True, startangle=90)
    ax1.axis('equal') # Equal aspect ratio ensures that pie is drawn as a circle.
```

```
#fetches the specified terminal from the argument via sql
titlelocation=Sql.titLocFetch(wild,days,count)
plt.title(titlelocation[0], fontsize=15)

plt.subplots_adjust()
#savename
dayer = str(days)
#print("day count and titles: ",dayer, titlelocation, "\n")

if wild >= 1:
    savename = titlelocation[1] + f"-{dayer}DaysPie.png"
    plt.savefig('static/'+savename)
else:
    plt.savefig(f'static/All{dayer}DaysPie.png')
plt.show()

plt.close()

def on_connect(client, userdata, flags, rc):
    if rc==0:
        print(f"MQTT connected with code {str(rc)}")
        client.subscribe("Project3/#")
    else:
        print("MQTT connection Error: Returned code=",rc)

#Callback for msg
def on_message(client,userdata,msg):

    try:
```

```
if (msg.topic == "Project3/Console"):
    load = str(msg.payload)
    load = load.strip("b")
    x = load.split("&")
    rating = float(x[0])
    address = str(x[1])
    print ("rating received: ",rating," from: ", address)
    Sql.inserter(rating, address,True)
    if address not in Sql.macList:
        Sql.inserter("N/A",address, False)

except Exception as e:
    print("Error in payload: ",e)
    print(f"{msg.topic} {msg.payload}")
```

```
if __name__ == "__main__":

    #Importing classesSELECT User, Host FROM mysql.user
    from sqller import Sqller

    #MQTT
    import paho.mqtt.client as mqtt

    #MariaDB
    import mariadb, sys
```

```
#Datetime
from datetime import timedelta as td
from datetime import datetime as dt

#Sleep
from time import sleep

#Matplotlib
from matplotlib import pyplot as plt
from matplotlib.ticker import PercentFormatter
from matplotlib import colors

try:
    client = mqtt.Client()
    client.on_connect = on_connect
    client.on_message = on_message
    client.username_pw_set("group4", "group4four")
    client.connect("127.0.0.1")

except:
    print("mqtt connection error")

#mariadb
try:
    conn = mariadb.connect(

        user="group4",
        password="group4four",
        host="127.0.0.1",
        port=3306,
        database="group4",
```



```

        autocommit=True
    )
    print("mariadb connected with code 0")
except mariadb.Error as e:
    print(f"Connection error, code: {e}")
    sys.exit(1)

# Get Cursor
cur = conn.cursor()

```

```
Sql = Sqller(conn,cur)
```

```

then = dt.now()
now = dt.now()
client.loop_start()
main(now,then)

```

### 9.2.3. Flask program

```
from flask import Flask, render_template
```

```
#import os
```

```
app = Flask(__name__, template_folder='templates')
```

```
@app.route('/')

```

```
def home():

```

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

```
@app.route("/All/Yearly")

```

```
def Display_IMG1():

```

```
    return render_template("All365DaysPie.html", user_image="All365DaysPie.png")
```

```
@app.route("/SH-A2.09/Yearly")

```

```
def Display_IMG2():

```

```

        return render_template("SH-A2.09-365DaysPie.html",
user_image="SH-A2.09-365DaysPie.png")

@app.route("/All/Daily/Days:30")
def Display_IMG3():
    return render_template("AllLine30Days.html", user_image="AllLine30Days.png")

@app.route("/SH-A2.09/Daily/Days:30")
def Display_IMG4():
    return render_template("SH-A2.09-Line30Days.html",
user_image="SH-A2.09-Line30Days.png")

@app.route('/about')
def about():
    return render_template("about.html", user_image="lara.png")

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000, debug=True, use_reloader=True)

```

### 9.3. Console software

```

String topic = "Project3/Console";

String deviceNumber = "1";

int numbOfHours = 5;

//int numbOfHoursConv = Hours * 3600;

int conversionHours = 1000000;

#include <WiFi.h>

#include <PubSubClient.h>

const char* ssid = "group4_IoT";
const char* password = "group4four";
const char* mqtt_server = "10.120.0.57";

```

```
const char* mqtt_user = "group4";
const char* mqtt_passwd = "group4four";

WiFiClient espClient;
PubSubClient client(espClient);
unsigned long lastMsg = 0;
#define MSG_BUFFER_SIZE (50)
char msg[MSG_BUFFER_SIZE];

#define bufferSize (50)
char topicArray[bufferSize];
char macArray[bufferSize];

#define button1 12
#define button2 13
#define button3 14
#define button4 15
#define led1 32
#define led2 33

#define sizeDisc (1000)
String disconnectArray[sizeDisc];
int arrayIndex = 0;
int index2 = 0;

void setup() {
  pinMode(button1, INPUT_PULLUP);
  pinMode(button2, INPUT_PULLUP);
  pinMode(button3, INPUT_PULLUP);
  pinMode(button4, INPUT_PULLUP);
  pinMode(led1, OUTPUT);
  pinMode(led2, OUTPUT);

  Serial.begin(115200);
```

```
setup_wifi();

client.setServer(mqtt_server, 1883);

client.setCallback(callback);

}

void loop() {

  digitalWrite(led1,HIGH);

  if (!client.connected()) {
    if (digitalRead(button1)==LOW){
      digitalWrite(led1,LOW);
      digitalWrite(led2,HIGH);
      String status1 = "3&";
      status1 += (String) WiFi.macAddress();
      status1 = status1 + "&" + deviceNumber;
      disconnectArray[arrayIndex] = status1;
      arrayIndex++;
      index2 = 1;
      delay(2200);
      reconnect();          //tweaked this
      digitalWrite(led2,LOW);
      digitalWrite(led1,HIGH);

    }else if (digitalRead(button2)==LOW){
      digitalWrite(led1,LOW);
      digitalWrite(led2,HIGH);
      String status2 = "2&";
      status2 += (String) WiFi.macAddress();
      status2 = status2 + "&" + deviceNumber;
      disconnectArray[arrayIndex] = status2;
      arrayIndex++;
      delay(2200);
      reconnect();
      index2 = 1;
    }
  }
}
```

```
digitalWrite(led2,LOW);
digitalWrite(led1,HIGH);

}else if (digitalRead(button3)==LOW){
digitalWrite(led1,LOW);
digitalWrite(led2,HIGH);

String status3 = "1&";
status3 += (String) WiFi.macAddress();
status3 = status3 + "&" + deviceNumber;
disconnectArray[arrayIndex] = status3;
arrayIndex++;
delay(2200);
reconnect();
index2 = 1;
digitalWrite(led2,LOW);
digitalWrite(led1,HIGH);

} else if (digitalRead(button4)==LOW){
Serial.println("sleep");
esp_sleep_enable_timer_wakeup(numbOfHours * conversionHours);
esp_deep_sleep_start();
}

}else {
for(int i = 0; i < index2; i++){
Offline_Handler();
}
if (digitalRead(button1)==LOW){
digitalWrite(led1,LOW);
digitalWrite(led2,HIGH);

String status1 = "3&";
status1 += (String) WiFi.macAddress();
status1 = status1 + "&" + deviceNumber;
```

```
status1.toCharArray(msg, MSG_BUFFER_SIZE);
Serial.println(status1);
topic.toCharArray(topicArray, bufferSize);
client.publish(topicArray, msg);
delay(3200);           //tweaked this
digitalWrite(led2,LOW);
digitalWrite(led1,HIGH);

} else if (digitalRead(button2)==LOW){
    digitalWrite(led1,LOW);
    digitalWrite(led2,HIGH);

    String status2 = "2&";
    status2 += (String) WiFi.macAddress();
    status2 = status2 + "&" + deviceNumber;
    status2.toCharArray(msg, MSG_BUFFER_SIZE);
    Serial.println(status2);
    topic.toCharArray(topicArray, bufferSize);
    client.publish(topicArray, msg);
    delay(3200);
    digitalWrite(led2,LOW);
    digitalWrite(led1,HIGH);

} else if (digitalRead(button3)==LOW){
    digitalWrite(led1,LOW);
    digitalWrite(led2,HIGH);

    String status3 = "1&";
    status3 += (String) WiFi.macAddress();
    status3 = status3 + "&" + deviceNumber;
    status3.toCharArray(msg, MSG_BUFFER_SIZE);
    Serial.println(status3);
    topic.toCharArray(topicArray, bufferSize);
    client.publish(topicArray, msg);
    delay(3200);
```

```
    digitalWrite(led2,LOW);
    digitalWrite(led1,HIGH);
}else if (digitalRead(button4)==LOW){
    Serial.println("sleep");
    esp_sleep_enable_timer_wakeup(numbOfHours * conversionHours);
    esp_deep_sleep_start();
}
}
client.loop();

}

void Offline_Handler(){
    for(int i = 0; i <= arrayIndex; i++){
        String message = disconnectArray[i];
        Serial.println(message);
        message.toCharArray(msg, MSG_BUFFER_SIZE);
        topic.toCharArray(topicArray, bufferSize);
        client.publish(topicArray, msg);
    }
    memset(disconnectArray, 0, sizeDisc);
    arrayIndex = 0;
    index2 = 0;
}

void DeepSleepHandler(){
    Serial.println("sleep");
    esp_sleep_enable_timer_wakeup(3000);
    esp_deep_sleep_start();
}

void setup_wifi() {

    delay(10);
    Serial.println();
```

```
Serial.print("Connecting to ");
Serial.println(ssid);

WiFi.mode(WIFI_STA);
WiFi.begin(ssid, password);

while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
}

randomSeed(micros());

Serial.println("");
Serial.println("WiFi connected");
Serial.println("IP address: ");
Serial.println(WiFi.localIP());
}

void callback(char* topic, byte* payload, unsigned int length) {
    Serial.print("Message arrived [");
    Serial.print(topic);
    Serial.print("] ");

    for (int i = 0; i < length; i++) {
        if((char)payload[i] == 's'){
            Serial.println("Data received");
            DeepSleepHandler();
        }
    }
    Serial.println();
}

void reconnect() {
```



```
// Loop until we're reconnected
//for (int i = 0; i < 1; i++) {
  Serial.println("Attempting MQTT connection...");
  // Create a random client ID
  String clientId = "ESP8266Client-";
  clientId += String(random(0xffff), HEX);
  // Attempt to connect
  if (client.connect(clientId.c_str(),mqtt_user,mqtt_passwd)) { //the real magic
    Serial.println("connected");
    // Once connected, publish an announcement...
    client.publish("Project3/Setup", "console connected");
    // ... and resubscribe
    client.subscribe("Project3/Power");
    Serial.println("subscribed");
    //i = 1;
  } else {
    Serial.print("failed, rc=");
    Serial.print(client.state());
    Serial.println(" try again in 5 seconds");
    Serial.println("Storing data offline");
    // Wait 5 seconds before retrying
  }
}
```

## 9.4. Webpage

### 9.4.1. Index.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Satisfaction Survey</title>
  <link rel="shortcut icon" href>

  <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
<style>
```

```
/*nav bar*/
```

```
{
```

```
margin: 0;
```

```
padding: 0;
```

```
box-sizing: border-box;
```

```
font-family: 'Poppins', sans-serif;
```

```
}
```

```
nav{
```

```
display: flex;
```

```
height: 80px;
```

```
width: 100%;
```

```
background: #1b1b1b;
```

```
align-items: center;
```

```
justify-content: space-between;
```

```
padding: 0 50px 0 100px;
```

```
flex-wrap: wrap;
```

```
}
```

```
nav .logo{
```

```
color: #fff;
```

```
font-size: 35px;
```

```
font-weight: 600;
```

```
}
```

```
nav ul{
```

```
display: flex;
```

```
flex-wrap: wrap;
```

```
list-style: none;
```

```
}
```

```
nav ul li{
```

```
margin: 0 5px;
```

```
}
```

```
nav ul li a{
```

```
color: #f2f2f2;
```

```
text-decoration: none;
```

```
font-size: 18px;
```

```
font-weight: 500;
```

```
padding: 8px 15px;
```

```
border-radius: 5px;
letter-spacing: 1px;
transition: all 0.3s ease;
}
nav ul li a.active,
nav ul li a:hover{
color: #111;
background: #fff;
}
nav .menu-btn i{
color: #fff;
font-size: 22px;
cursor: pointer;
display: none;
}
input[type="checkbox"]{
display: none;
}

/* navbar ends */
/*buttons*/
.btn {
font-family: Helvetica, Arial, sans-serif;
font-size: inherit; /*it was 10px */
text-align: left;
cursor: pointer;
padding: 10px 50px; /*button thickness*/
display: inline-block;
margin: 5px 2px; /*space between buttons*/
letter-spacing: 1px;
font-weight: bold;
outline: none;
position: relative;
top: 150px; /*buttons position vertically*/
-webkit-transition: all 0.3s;
-moz-transition: all 0.3s;
transition: all 0.3s;
}
```

```
.btn:after {  
  content: "";  
  position: absolute;  
  z-index: -1;  
  -webkit-transition: all 0.3s;  
  -moz-transition: all 0.3s;  
  transition: all 0.3s;  
}  
  
.btn-cta {  
  border-radius: 50px; /*corner sharpness*/  
  display: block;  
  width: 400px;  
  border: none;  
  background: #E2A5CD;  
  color: #E2A5CD;  
  overflow: hidden;  
}  
  
.btn-cta:active {  
  color: #F5D5ED;  
}  
  
.btn-cta:hover {  
  background: #F5D5ED;  
  color: black;  
}  
  
.btn-cta svg { /*arrow*/  
  position: absolute;  
  left: 200%;  
  top: 0;  
  height: 100%;  
  font-size: 125%;  
  line-height: 3.5;  
  color: #333;  
  -webkit-transition: all 0.3s;
```

```
-moz-transition: all 0.3s;
transition: all 0.3s;
}

.btn-cta:active:before {
  color: #545454;
}

.btn-cta:hover svg {
  left: 90%; /*arrow placement horizontally */
}

/*buttons end*/

body,html {
  margin: 0;
  padding: 0;
  height: 100%;
  background-image: url("static/background.jpeg");
  background-position: center;
  background-repeat: no-repeat;
  background-size: contain;
  font-family: sans-serif;
}

a {
  text-decoration: none;
  color: black;
}
</style>
</head>

<body>

<nav>
<div class="logo">Welcome to Group 4's Web Page</div>
<ul>
```

```

<li><a href={{ url_for( 'home' ) }}>Home</a></li>
<li><a href={{ url_for( 'about' ) }}>About</a></li>
</ul>
</nav>

```

```

<button class="btn btn-cta" >
<p><a href= {{ url_for( 'Display_IMG1' ) }}> Average rating all locations for the last year</a></p>
<svg xmlns="" version="1.1" id="arrow" x="0px" y="0px" width="11.121px" height="19.414px" viewBox="0
0 11.121 19.414" enable-background="new 0 0 11.121 19.414" xml:space="preserve">
<polygon fill="#fff" points="1.414,19.414 0,18 8.293,9.707 0,1.414 1.414,0 11.121,9.707 "/>
</svg> </button>

```

```

<button class="btn btn-cta">
<p><a href= {{ url_for( 'Display_IMG2' ) }}> Average rating for A2.09 Classroom for the last year</a></p>
<svg xmlns="" version="1.1" id="arrow" x="0px" y="0px" width="11.121px" height="19.414px" viewBox="0
0 11.121 19.414" enable-background="new 0 0 11.121 19.414" xml:space="preserve">
<polygon fill="#fff" points="1.414,19.414 0,18 8.293,9.707 0,1.414 1.414,0 11.121,9.707 "/>
</svg> </button>

```

```

<button class="btn btn-cta">
<p><a href= {{ url_for( 'Display_IMG3' ) }}> Daily average rating for last 30 days</a></p>
<svg xmlns="" version="1.1" id="arrow" x="0px" y="0px" width="11.121px" height="19.414px"
viewBox="0 0 11.121 19.414" enable-background="new 0 0 11.121 19.414" xml:space="preserve">
<polygon fill="#fff" points="1.414,19.414 0,18 8.293,9.707 0,1.414 1.414,0 11.121,9.707 "/>
</svg> </button>

```

```

<button class="btn btn-cta">
<p><a href= {{ url_for( 'Display_IMG4' ) }}> Daily average rating for A2.09 Classroom for last 30 days</a></p>
<svg xmlns="" version="1.1" id="arrow" x="0px" y="0px" width="11.121px" height="19.414px" viewBox="0
0 11.121 19.414" enable-background="new 0 0 11.121 19.414" xml:space="preserve">
<polygon fill="#fff" points="1.414,19.414 0,18 8.293,9.707 0,1.414 1.414,0 11.121,9.707 "/>
</svg> </button>

```

```

</body>
<div class="container">
<div class="box">

</div>

```

```
</div>
```

### 9.4.2. ALL365DaysPie.html

```
<!DOCTYPE html>
```

```
<html lang="en">
```

```
<head>
```

```
<title>Average rating all time</title>
```

```
<link rel="icon" type="image/x-icon" href="../favicon.ico">
```

```
<meta charset="utf-8">
```

```
<meta name="viewport" content="width=device-width, initial-scale=1">
```

```
<style>
```

```
/*nav bar*/
```

```
{
```

```
margin: 0;
```

```
padding: 0;
```

```
box-sizing: border-box;
```

```
font-family: 'Poppins', sans-serif;
```

```
}
```

```
nav{
```

```
display: flex;
```

```
height: 80px;
```

```
width: 100%;
```

```
background: #1b1b1b;
```

```
align-items: center;
```

```
justify-content: space-between;
```

```
padding: 0 50px 0 100px;
```

```
flex-wrap: wrap;
```

```
}
```

```
nav .logo{
```

```
color: #fff;
```

```
font-size: 35px;
```

```
font-weight: 600;
```

```
}
```

```
nav ul{
```

```
display: flex;
```

```
flex-wrap: wrap;
list-style: none;
}
nav ul li{
margin: 0 5px;
}
nav ul li a{
color: #f2f2f2;
text-decoration: none;
font-size: 18px;
font-weight: 500;
padding: 8px 15px;
border-radius: 5px;
letter-spacing: 1px;
transition: all 0.3s ease;
}
nav ul li a.active,
nav ul li a:hover{
color: #111;
background: #fff;
}
nav .menu-btn i{
color: #fff;
font-size: 22px;
cursor: pointer;
display: none;
}
input[type="checkbox"]{
display: none;
}

body {
font-family: Arial, Helvetica, sans-serif;
}

/* Style the header */
header {
background-color: #F7C1BB;
```



```
padding: 30px;
text-align: center;
font-size: 35px;
color: black;
}
```

```
article {
float: left;
padding: 20px;
width: 100%;
background-color: #885A5a;

}
```

```
/* Clear floats after the columns */
```

```
section::after {
content: "";
display: table;
clear: both;
}
```

```
/* Responsive layout - makes the two columns/boxes stack on top of each other instead of next to each other, on
small screens */
```

```
@media (max-width: 600px) {
nav, article {
width: 100%;
height: auto;
}
}
```

```
footer{
bottom: 10px;
position: fixed;
width: 100%;
}
```

```
.footer {
```

```

height: 50px;
margin: auto;
width: 700px;
text-align: center;
padding: 20px;
color: black;
}
</style>
</head>
<body>
<nav>
<div class="logo">TLSJ</div>
<ul>
<li><a href={{ url_for( 'home' ) }}>Home</a></li>
<li><a href={{ url_for( 'about' ) }}>About</a></li>

</ul>
</nav>

<header>
Average rating all time
</header>

<div><h3>Pie chart</h3></center>
</div>

<footer>
<div class="footer">
copyright &copy; 2022 all rights reserved because Group 4 Ltd. is a serious company <br>
contact: badboyZ4eva@gmail.com
</div>
</footer>

</body>
</html>

```

### 9.4.3. AllLine30Days.html

```
<!DOCTYPE html>
<html lang="en">
<head>

<title>Average rating all time</title>
<link rel="icon" type="image/x-icon" href="../favicon.ico">

<meta charset="utf-8">
<meta name="viewport" content="width=device-width, initial-scale=1">
<style>
/*nav bar*/
*{
margin: 0;
padding: 0;
box-sizing: border-box;
font-family: 'Poppins', sans-serif;
}
nav{
display: flex;
height: 80px;
width: 100%;
background: #1b1b1b;
align-items: center;
justify-content: space-between;
padding: 0 50px 0 100px;
flex-wrap: wrap;
}
nav .logo{
color: #fff;
font-size: 35px;
font-weight: 600;
}
nav ul{
display: flex;
flex-wrap: wrap;
list-style: none;
}
nav ul li{
```

```
margin: 0 5px;
}
nav ul li a{
color: #f2f2f2;
text-decoration: none;
font-size: 18px;
font-weight: 500;
padding: 8px 15px;
border-radius: 5px;
letter-spacing: 1px;
transition: all 0.3s ease;
}
nav ul li a.active,
nav ul li a:hover{
color: #111;
background: #fff;
}
nav .menu-btn i{
color: #fff;
font-size: 22px;
cursor: pointer;
display: none;
}
input[type="checkbox"]{
display: none;
}

body {
font-family: Arial, Helvetica, sans-serif;
}

/* Style the header */
header {
background-color: #F7C1BB;
padding: 30px;
text-align: center;
font-size: 35px;
color: black;
```

```
}
```

```
article {  
float: left;  
padding: 20px;  
width: 100%;  
background-color: #885A5a;  
  
}
```

```
/* Clear floats after the columns */
```

```
section::after {  
content: "";  
display: table;  
clear: both;  
}
```

```
/* Responsive layout - makes the two columns/boxes stack on top of each other instead of next to each other, on  
small screens */
```

```
@media (max-width: 600px) {  
nav, article {  
width: 100%;  
height: auto;  
}  
}
```

```
footer{  
bottom: 10px;  
position: fixed;  
width: 100%;  
}
```

```
.footer {  
height: 50px;  
margin: auto;  
width: 700px;  
text-align:center;
```

```
padding:20px;
color:black;
}
</style>
</head>
<body>
<nav>
<div class="logo">TLSJ</div>
<ul>
<li><a href={{ url_for( 'home' ) }}>Home</a></li>
<li><a href={{ url_for( 'about' ) }}>About</a></li>

</ul>
</nav>

<header>
Average rating last 30 days
</header>

<div><h3>Pie chart</h3></center>
</div>

<footer>
<div class="footer">
copyright &copy; 2022 all rights reserved because Group 4 Ltd. is a serious company <br>
contact: badboyZ4eva@gmail.com
</div>
</footer>

</body>
</html>
```

#### 9.4.4. SH-A2.09-365DaysPie-html

```
<!DOCTYPE html>
<html lang="en">
<head>
```

```
<title>Average rating all time</title>
<link rel="icon" type="image/x-icon" href="../favicon.ico">

<meta charset="utf-8">
<meta name="viewport" content="width=device-width, initial-scale=1">
<style>
/*nav bar*/
*{
margin: 0;
padding: 0;
box-sizing: border-box;
font-family: 'Poppins', sans-serif;
}
nav{
display: flex;
height: 80px;
width: 100%;
background: #1b1b1b;
align-items: center;
justify-content: space-between;
padding: 0 50px 0 100px;
flex-wrap: wrap;
}
nav .logo{
color: #fff;
font-size: 35px;
font-weight: 600;
}
nav ul{
display: flex;
flex-wrap: wrap;
list-style: none;
}
nav ul li{
margin: 0 5px;
}
nav ul li a{
color: #f2f2f2;
```

```
text-decoration: none;
font-size: 18px;
font-weight: 500;
padding: 8px 15px;
border-radius: 5px;
letter-spacing: 1px;
transition: all 0.3s ease;
}

nav ul li a.active,
nav ul li a:hover{
color: #111;
background: #fff;
}

nav .menu-btn i{
color: #fff;
font-size: 22px;
cursor: pointer;
display: none;
}

input[type="checkbox"]{
display: none;
}


body {
font-family: Arial, Helvetica, sans-serif;
}


/* Style the header */
header {
background-color: #F7C1BB;
padding: 30px;
text-align: center;
font-size: 35px;
color: black;
}


article {
float: left;
```



```
padding: 20px;
width: 100%;
background-color: #885A5a;

}

/* Clear floats after the columns */
section::after {
content: "";
display: table;
clear: both;
}

/* Responsive layout - makes the two columns/boxes stack on top of each other instead of next to each other, on
small screens */
@media (max-width: 600px) {
nav, article {
width: 100%;
height: auto;
}
}

footer{
bottom: 10px;
position: fixed;
width: 100%;
}

.footer {
height: 50px;
margin: auto;
width: 700px;
text-align: center;
padding: 20px;
color: black;
}
</style>
</head>
```

```

<body>
<nav>
<div class="logo">TLSJ</div>
<ul>
<li><a href={{ url_for('home') }}>Home</a></li>
<li><a href={{ url_for('about') }}>About</a></li>

</ul>
</nav>

<header>
Average rating for SH-A2.09 All Time
</header>

<div><h3>Pie chart</h3></center>
</div>

<footer>
<div class="footer">
copyright &copy; 2022 all rights reserved because Group 4 Ltd. is a serious company <br>
contact: badboyZ4eva@gmail.com
</div>
</footer>

</body>
</html>

```

#### 9.4.5. SH-A2.09-Line30Days.html

```

<!DOCTYPE html>
<html lang="en">
<head>

<title>Average rating all time</title>
<link rel="icon" type="image/x-icon" href="../favicon.ico">

<meta charset="utf-8">
<meta name="viewport" content="width=device-width, initial-scale=1">

```

```
<style>
/*nav bar*/
*{
margin: 0;
padding: 0;
box-sizing: border-box;
font-family: 'Poppins', sans-serif;
}
nav{
display: flex;
height: 80px;
width: 100%;
background: #1b1b1b;
align-items: center;
justify-content: space-between;
padding: 0 50px 0 100px;
flex-wrap: wrap;
}
nav .logo{
color: #fff;
font-size: 35px;
font-weight: 600;
}
nav ul{
display: flex;
flex-wrap: wrap;
list-style: none;
}
nav ul li{
margin: 0 5px;
}
nav ul li a{
color: #f2f2f2;
text-decoration: none;
font-size: 18px;
font-weight: 500;
padding: 8px 15px;
border-radius: 5px;
letter-spacing: 1px;
```

```
transition: all 0.3s ease;
}

nav ul li a.active,
nav ul li a:hover{
color: #111;
background: #fff;
}

nav .menu-btn i{
color: #fff;
font-size: 22px;
cursor: pointer;
display: none;
}

input[type="checkbox"]{
display: none;
}


body {
font-family: Arial, Helvetica, sans-serif;
}


/* Style the header */
header {
background-color: #F7C1BB;
padding: 30px;
text-align: center;
font-size: 35px;
color: black;
}


article {
float: left;
padding: 20px;
width: 100%;
background-color: #885A5a;
}
}
```

```
/* Clear floats after the columns */
```

```
section::after {  
  content: "";  
  display: table;  
  clear: both;  
}
```

```
/* Responsive layout - makes the two columns/boxes stack on top of each other instead of next to each other, on  
small screens */
```

```
@media (max-width: 600px) {  
  nav, article {  
    width: 100%;  
    height: auto;  
  }  
}
```

```
footer{  
  bottom: 10px;  
  position: fixed;  
  width: 100%;  
}
```

```
.footer {  
  height: 50px;  
  margin: auto;  
  width: 700px;  
  text-align:center;  
  padding:20px;  
  color:black;  
}
```

```
</style>
```

```
</head>
```

```
<body>
```

```
<nav>
```

```
<div class="logo">TLSJ</div>
```

```
<ul>
```

```
<li><a href={{ url_for('home') }}>Home</a></li>
```

```
<li><a href="{{ url_for('about') }}">About</a></li>
```

```
</ul>
```

```
</nav>
```

```
<header>
```

Average rating this week

```
</header>
```

```
<div><h3>Pie chart</h3></center>
```

```
</div>
```

```
<footer>
```

```
<div class="footer">
```

copyright &copy; 2022 all rights reserved because Group 4 Ltd. is a serious company <br>

contact: badboyZ4eva@gmail.com

```
</div>
```

```
</footer>
```

```
</body>
```

```
</html>
```

### 9.4.6. About.html

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>About</title>
<style>
  /*nav bar*/
  *{
margin: 0;
padding: 0;
box-sizing: border-box;
font-family: 'Poppins', sans-serif;
}
nav{
display: flex;
height: 80px;
width: 100%;
background: #1b1b1b;
align-items: center;
justify-content: space-between;
padding: 0 50px 0 100px;
flex-wrap: wrap;
}
nav .logo{
color: #fff;
font-size: 35px;
font-weight: 600;
}
nav ul{
display: flex;
flex-wrap: wrap;
list-style: none;
}
nav ul li{
margin: 0 5px;
}
```

```
nav ul li a{
  color: #f2f2f2;
  text-decoration: none;
  font-size: 18px;
  font-weight: 500;
  padding: 8px 15px;
  border-radius: 5px;
  letter-spacing: 1px;
  transition: all 0.3s ease;
}
nav ul li a.active,
nav ul li a:hover{
  color: #111;
  background: #fff;
}
nav .menu-btn i{
  color: #fff;
  font-size: 22px;
  cursor: pointer;
  display: none;
}
input[type="checkbox"]{
  display: none;
}
/*nav bar ends*/

body {
  font-family: Arial, Helvetica, sans-serif;
  margin: 0;
}

html {
  box-sizing: border-box;
}

*, *:before, *:after {
  box-sizing: inherit;
}
```



```
.column {  
  float: left;  
  width: 25%;  
  margin-bottom: 20px;  
  padding: 0 8px;  
}  
  
.card {  
  box-shadow: 0 4px 8px 0 rgba(0, 0, 0, 0.2);  
  margin: 8px;  
}  
  
.about-section {  
  padding: 50px;  
  text-align: center;  
  background-color: #474e5d;  
  color: white;  
}  
  
.container {  
  padding: 0 16px;  
}  
  
.container::after, .row::after {  
  content: "";  
  clear: both;  
  display: table;  
}  
  
.title {  
  color: grey;  
}  
  
.button {  
  border: none;  
  outline: 0;  
  display: inline-block;
```

```
padding: 8px;
color: white;
background-color: #000;
text-align: center;
cursor: pointer;
width: 100%;
}

.button:hover {
background-color: #555;
}

@media screen and (max-width: 650px) {
.column {
width: 100%;
display: block;
}
}
</style>
</head>
<body>
<nav>
<div class="logo">TLSJ</div>
<ul>
<li><a href={{ url_for('home') }}>Home</a></li>
<li><a href={{ url_for('about') }}>About</a></li>

</ul>
</nav>
<div class="about-section">
<h1>About Us Page</h1>
<p>Some text about who we are and what we do.</p>
<p>Resize the browser window to see that this page is responsive by the way.</p>
</div>

<h2 style="text-align:center">Our Team</h2>
<div class="row">
<div class="column">
```

```
<div class="card">
  
  <div class="container">
    <h2>Lara Turunc</h2>
    <p class="title">Group Member</p>
    <p>eaalatu@students.eaaa.dk</p>
  </div>
</div>

<div class="column">
  <div class="card">
    
    <div class="container">
      <h2>Severin Nyffenegger</h2>
      <p class="title">Group Member</p>
      <p>eaaseny@students.eaaa.dk</p>
    </div>
  </div>
</div>

<div class="column">
  <div class="card">
    
    <div class="container">
      <h2>Justyna Kluska</h2>
      <p class="title">Group Member</p>
      <p>eaajklu@students.eaaa.dk</p>
    </div>
  </div>
</div>

<div class="column">
  <div class="card">
    
    <div class="container">
      <h2>Thomas Joensen</h2>
      <p class="title">Group Member</p>
```

```
<p>eaatmjoen@students.eaaa.dk</p>
</div>
</div>
</div>
</div>
</body>
</html>
```

### 9.5.death\_valley\_highs\_lows.py by Kristian Pontoppidan

```
import csv
from datetime import datetime
from matplotlib import pyplot as plt

filename = 'death_valley_2018_simple.csv'
with open(filename) as f:
    reader = csv.reader(f)
    header_row = next(reader)
    # Get dates, high and low temperatures from this file.
    dates, highs, lows = [], [], []

    for row in reader:
        current_date = datetime.strptime(row[2], '%Y-%m-%d')
        try:
            high = int(row[4])
            low = int(row[5])
        except ValueError:
            print(f"Missing data for {current_date}")
        else:
            dates.append(current_date)
            highs.append(high)
            lows.append(low)

print(dates)
print(highs)
print(lows)

# Plot the high and low temperatures.
plt.style.use('seaborn')
fig, ax = plt.subplots()
ax.plot(dates, highs, c='red', alpha=0.5)
ax.plot(dates, lows, c='blue', alpha=0.5)
```

```
plt.fill_between(dates, highs, lows, facecolor='blue', alpha=0.1)

# Format plot.
title = "Daily high and low temperatures - 2018\nDeath Valley, CA"
plt.title(title, fontsize=20)
plt.xlabel("", fontsize=16)
fig.autofmt_xdate()
plt.ylabel("Temperature (F)", fontsize=16)
plt.tick_params(axis='both', which='major', labelsize=16)

plt.show()
```