**IoT System Project**

**Mini Project 1[E2]**

(Smart Cabinet)

**Group 10 Members**

Armando Mak [223175Z]

R Sajeev [221325C]

1. Set up

### **. Power On and Connect ESP32 Devices**

* **ESP32 Smartwatch**: Power on the device and ensure it's connected to Wi-Fi.
* **ESP32 CAM**: Power on the camera module and verify the camera feed is accessible.
* **ESP32 in Cabinet**: Ensure the device is powered and connected to the network.
* **Router:** Ensure the router is connected to the pwer

### **2. Set Up and Start the Flask Web Server**

* **Navigate to Your Project Directory**:  
  bash  
  Copy code  
  cd path\_to\_your\_project\_directory
* **Activate Virtual Environment (if used)**:  
  bash  
  Copy code  
  source venv/bin/activate # Linux/Mac
* .\venv\Scripts\activate # Windows
* **Start the Flask Server**:  
  bash  
  Copy code  
  python app.py
* **Check Server Availability**: Open your browser and go to http://localhost:5000 to see if your server is running correctly.

### **3. Connect ESP32 Devices to the Server**

* **Verify Wi-Fi Connection**: Ensure all your ESP32 devices are connected to the same Wi-Fi network as the server. Use Serial.print(WiFi.localIP()); on ESP32 devices to check their IP addresses.
* **Update ESP32 Code (if needed)**: Ensure that the server IP in your ESP32 code points to the correct IP address where the Flask server is hosted.  
  cpp  
  Copy code  
  http.begin("http://IP-address"); // Replace with your actual server IP

### **4. Test Communication Between ESP32 Devices and Web Server**

* **Send Data from ESP32 Devices**: Trigger actions on your ESP32 devices that send data to the Flask server (e.g., pressing a button on the cabinet).
* **Monitor Server Logs**: Check your server's console to verify that data is being received from the ESP32 devices.
* **Access the Web Interface**: Open the web interface in your browser to verify that data is being displayed correctly and that the controls work.

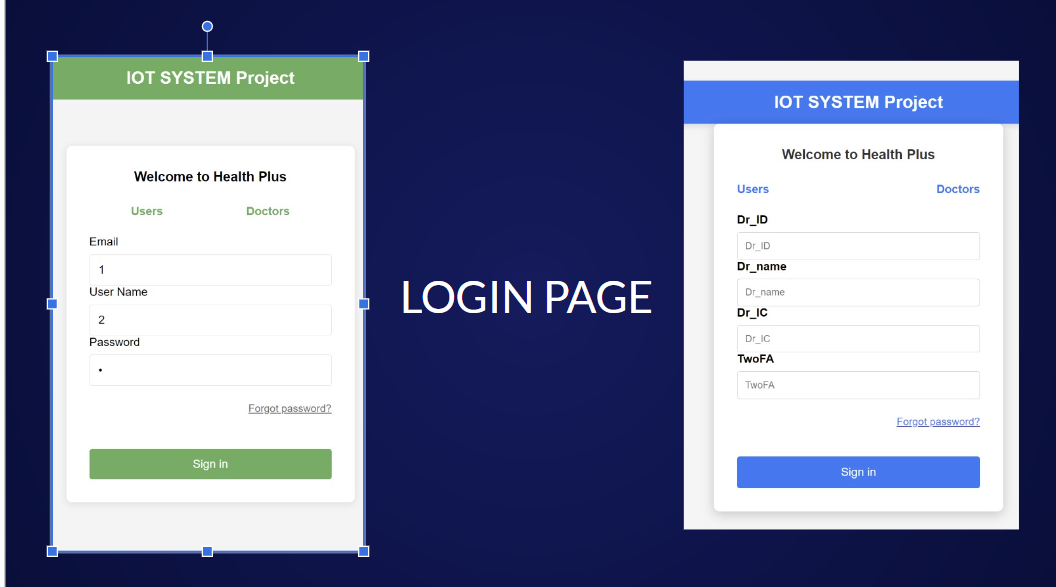
### **5. Network Configuration**

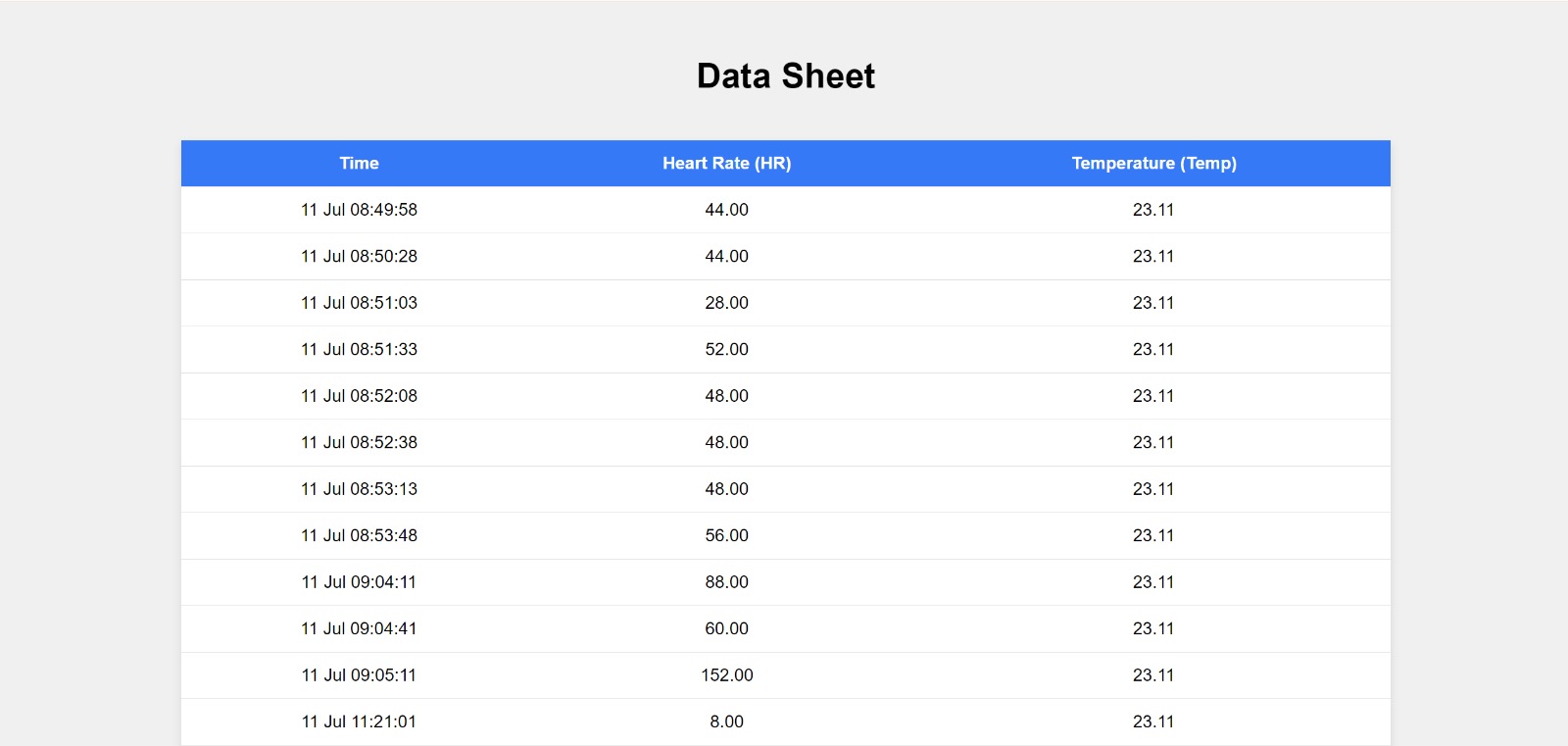
* **Static IP for ESP32 Devices**: Assign static IP addresses to your ESP32 devices in your router settings to avoid reconfiguring IPs after a restart.

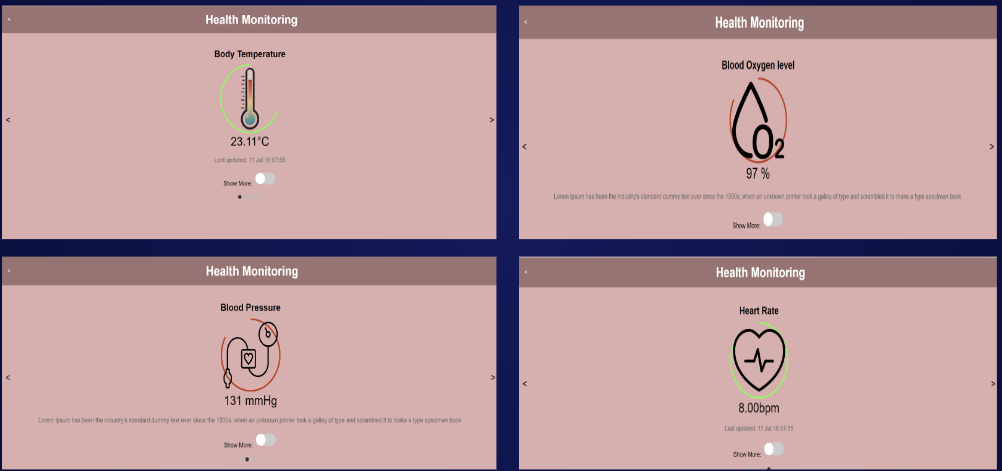
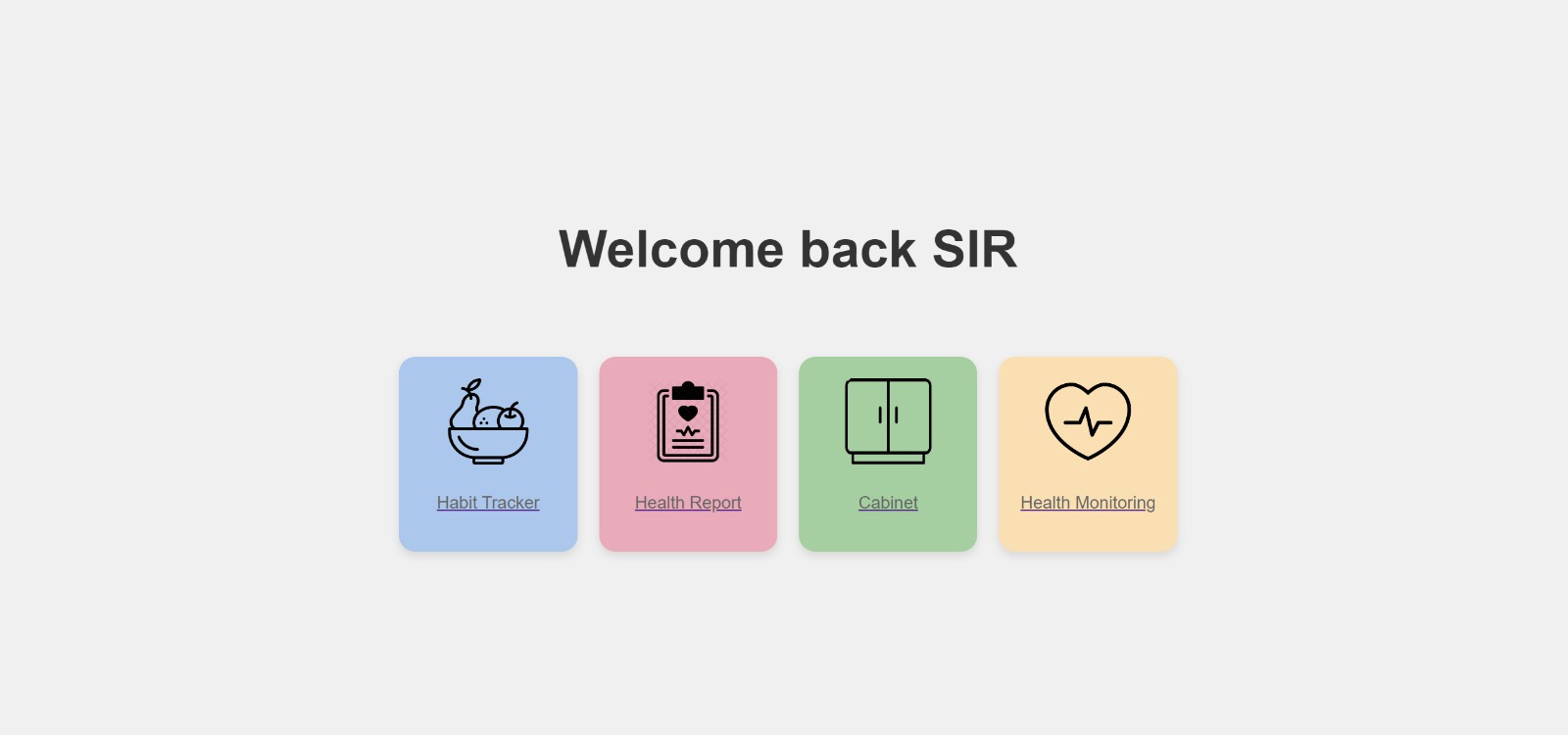
### **6. Secure Your Setup**

* **Use HTTPS**: Implement HTTPS on your Flask server using SSL certificates (e.g., Let's Encrypt) for secure communication.
* **Authentication**: Implement basic authentication for your web interface to prevent unauthorized access.

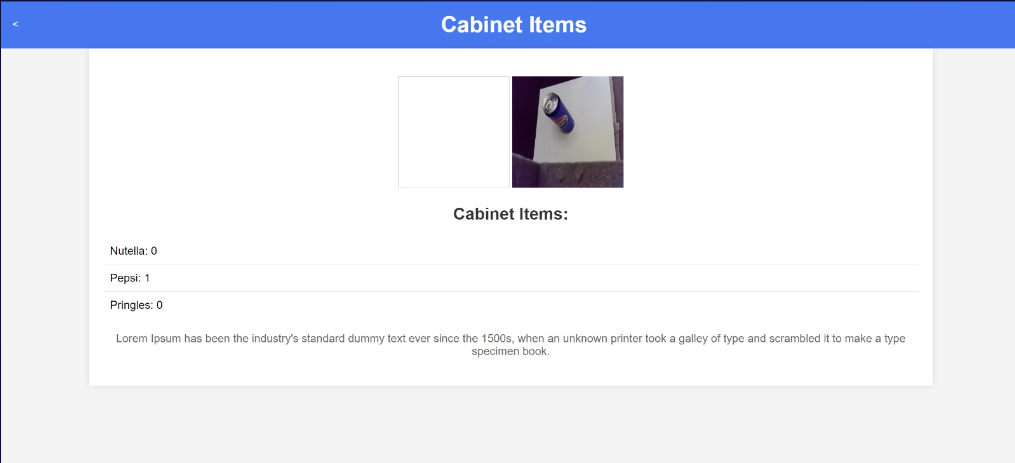
2. Dashboard graphics

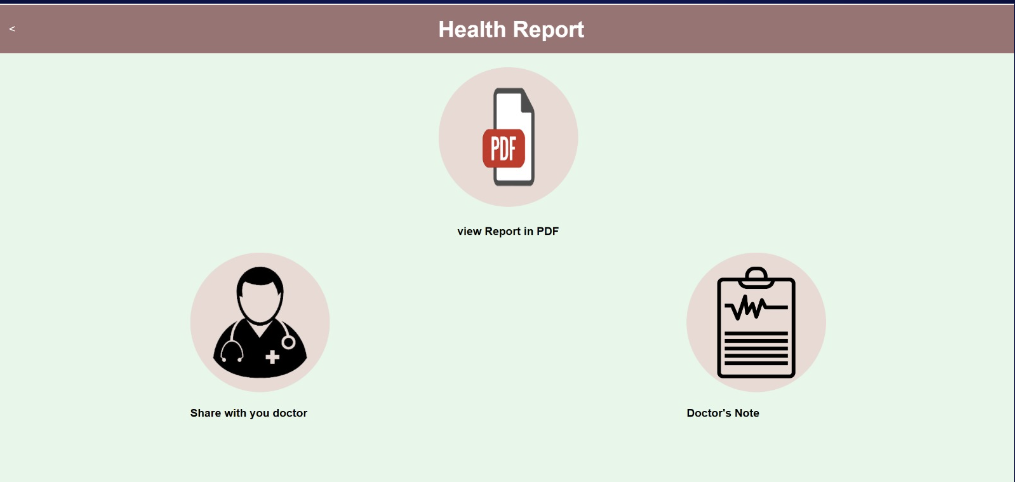


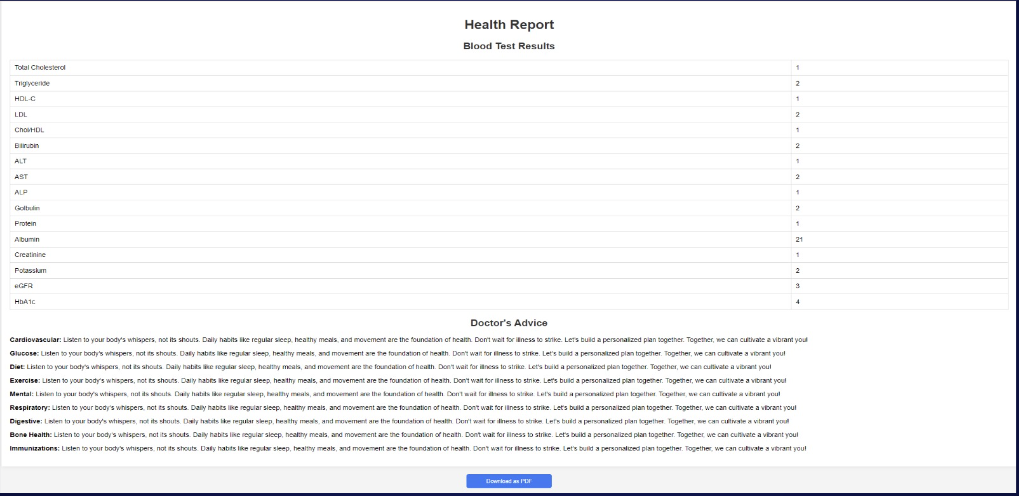


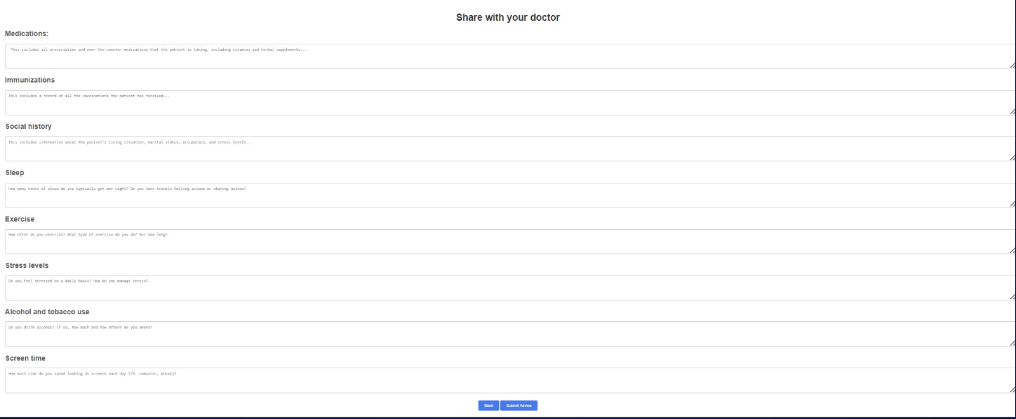


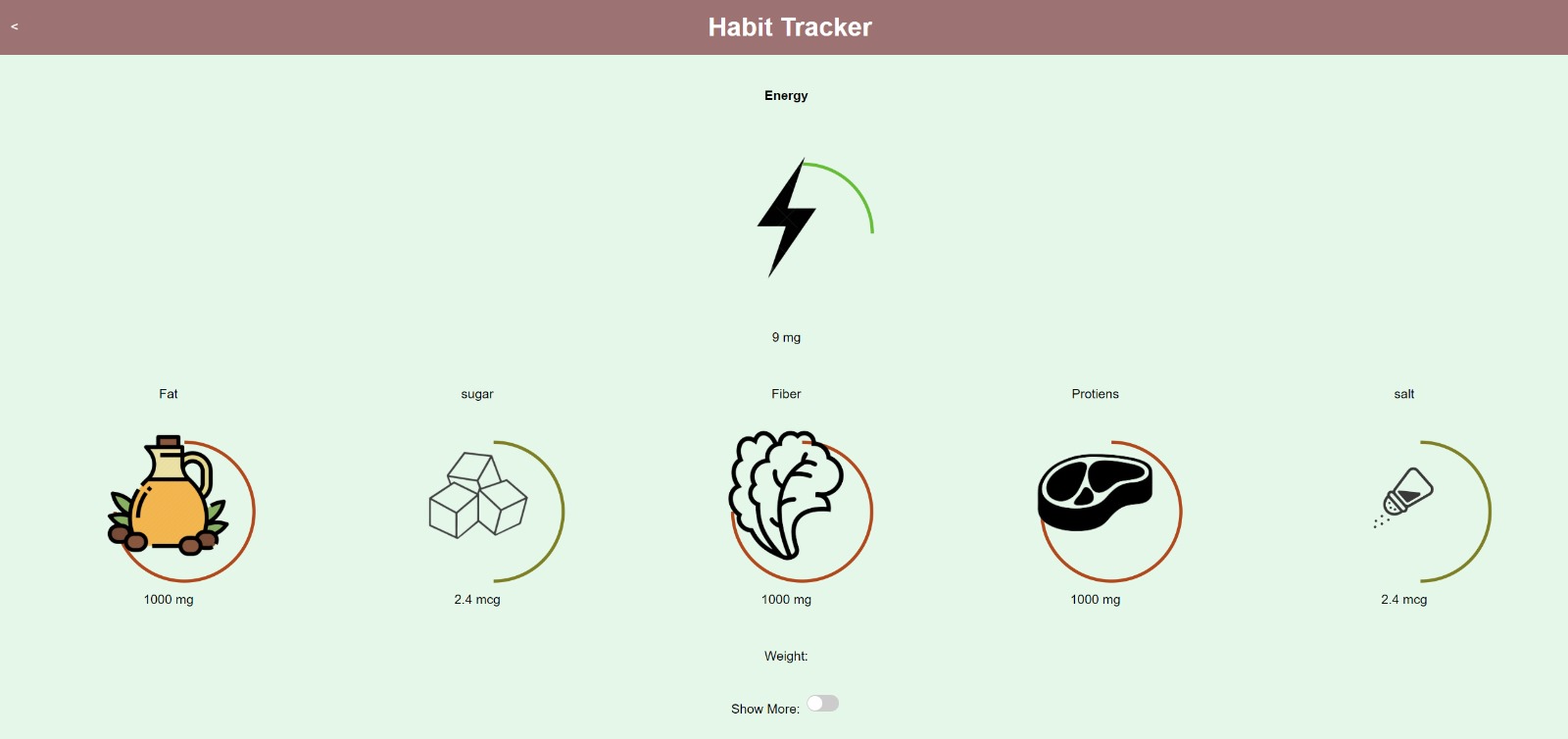










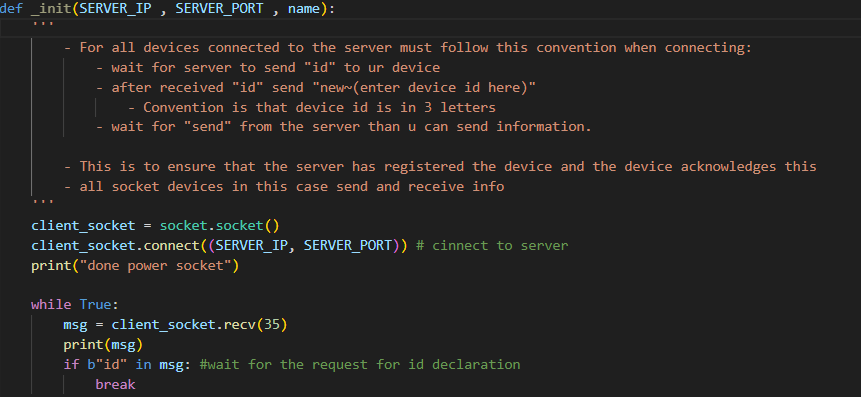


3. Setup for of sockets

# Register New ESP client

When the esp client connects to the server

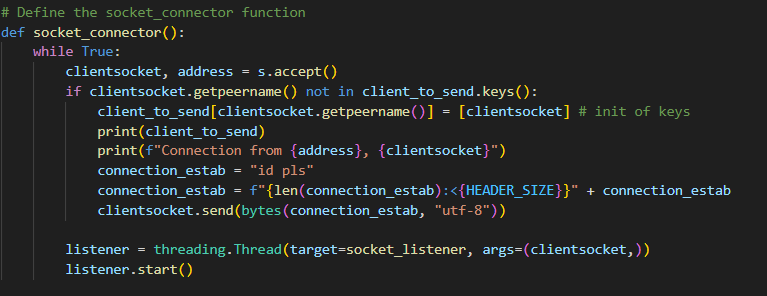
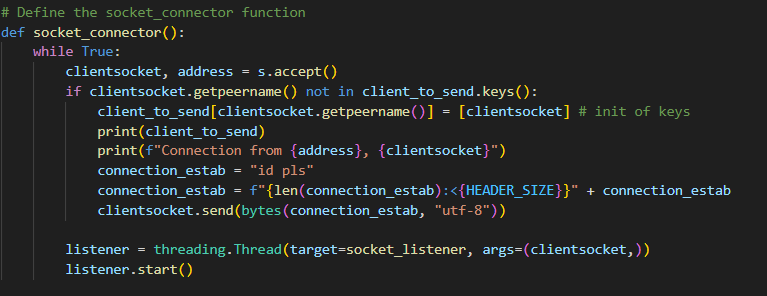
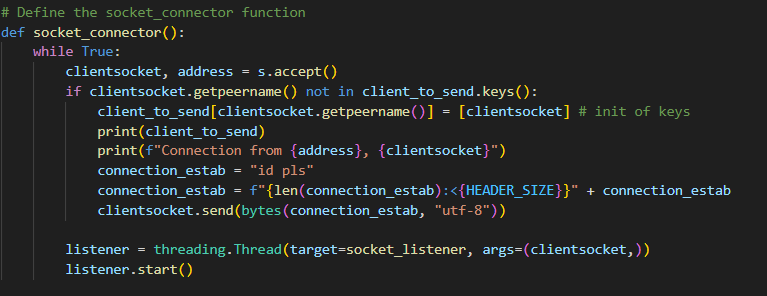
ESP-Client

* 
* The ESP client on power up will try to initialize the socket object and connect to the server
  + 
* Then it will wait for the server to request for the client for its id
  + 

|

V

Server

* + 
* Run the above code in a thread
  + In this code it will accept any socket that tries to connect to itself
    - 
  + Then check in the client to send if the socket was registered before
    - If it does not it will request id from the ESP client
    - 
  + Then start the message listener “socket\_listener”
    - 

|

V

# Initialize relevant info

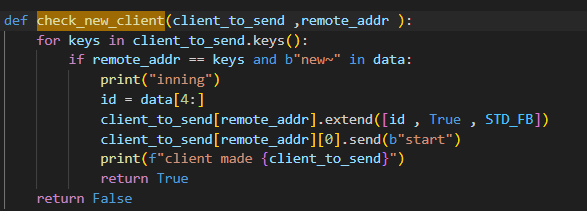
ESP-Client

* Client then sends “new~”
  + 
  + Which is to initialize all the other relevant info in client\_to\_send
  + Then wait for server to send “start”
    - 

|

V

ESP Server

* Receives the “new~”
  + 
* Checks for new client
  + 
* If it is a new client:
  + Check if the remote address is inside the client\_to\_send dictionary and has the “new~” message
    - 
  + Put the necessary information
    - 
    - id
      * for convenience on coding side as we can just call “cab” instead of its remote address to find its information
    - True
      * (unused)
    - STD\_FB
      * To set the standard frame buffer size = 2048 bytes 
  + Send the “start”
    - 

|

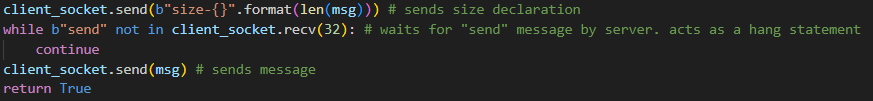
V

ESP-Client

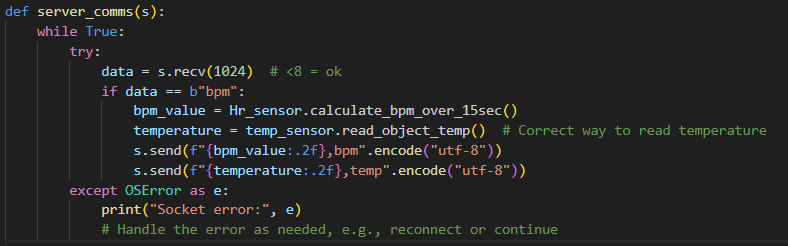
* Then allow the rest of the code to proceed
  + 

4 & 5. Sending info using sockets

* sending can only be in utf-8 or base64
* The convention to send information from the esp client looks like: <https://www.canva.com/design/DAGJTyrvH_w/cjLpcchwf4Gd9K_r6uydag/edit?utm_content=DAGJTyrvH_w&utm_campaign=designshare&utm_medium=link2&utm_source=sharebutton>
* Convention 1 is to ensure there is always space for data packets to come into the server



* Convention 2 just sends and does not declare the size:



* Just to add on, all the esp32 can do threading except the ESP32 CAM. As it will mess with its firmware, which results in error messages.

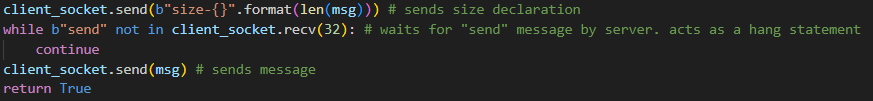
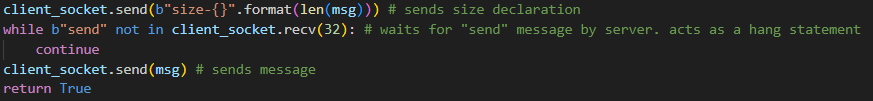
# Convention 1

This method only applies from ESP client to server

* This is to ensure no matter what the frame buffer will always be bigger
* Can be seen in the camera and the cabinet

Steps:

**For the ESP client:**

1. For the esp-client it first has to send the size declaration
   * 
   * Then it waits for the send
     + 

**On the server side:**

1. It will first search for the highest Frame buffer size from all the known users(default is 2048 bytes )
2. Then it will receive any information at a frame buffer of the highest known frame buffer from all its users
   * 
3. Receive the data at the highest frame buffer size

* 

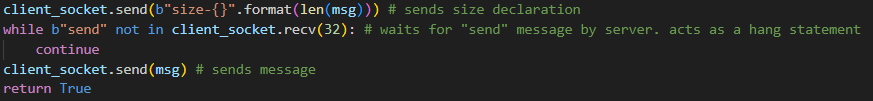
1. Then find the id from the remote address
   * 
2. See if it is a new client

* 

1. If it is not a new client, check the message and see if it is a size declaration or a message:
2. Size Declaration

* If it is a size declaration:
  + Only take the size of the frame buffer and assign it to the esp-client’s frame buffer size
  + 
  + As typically the send is sent with “size-FrameBufferValue”. Where FrameBufferValue is the size of the message to be sent
  + Then send the acknowledgement “send”

**For the ESP client:**

* When the ESP client receives the “send” acknowledgement, then it can send the information
  + 

1. Message:

* Repeat steps 2 to 4
* Find the id of the ESP client sender
  + 

# Convention 2

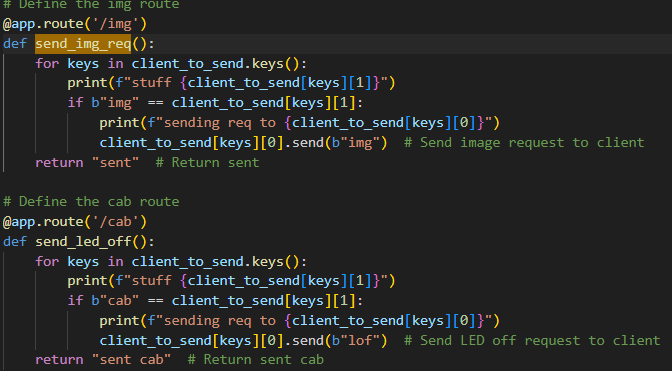
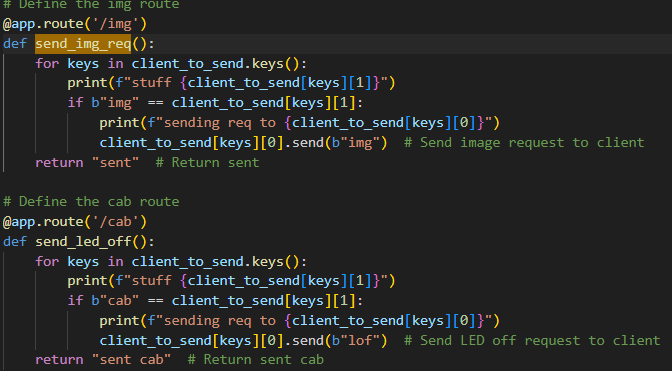
It can be done for

* Server to ESP client
  + The messages sent from the server to the clients are typically 3-4 letters/characters long
* ESP client to Server
* The frame buffer size will always stay the same no matter what
* Can be seen in the camera and the cabinet
  + Partly seen in smart watch

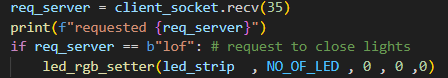
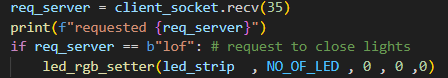
**Server to ESP client**

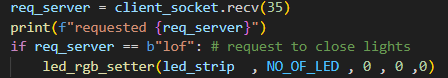
Steps:

Server

* Server first has to find the id to send
  + 
* Send the message
  + 

ESP client:

* Waits for the message to be received
  + 
  + In this case the frame buffer is 35
    - I chose 35 bytes because it should be more than enough to accommodate 3-4 letter messages
* Use the message
  + In this case if the message is lof 
    - close all the lights

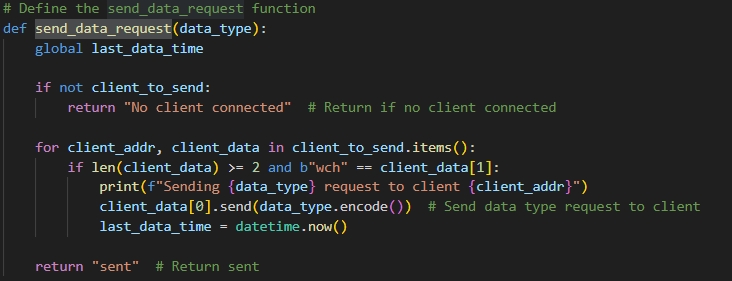


**Server to ESP client to Server**

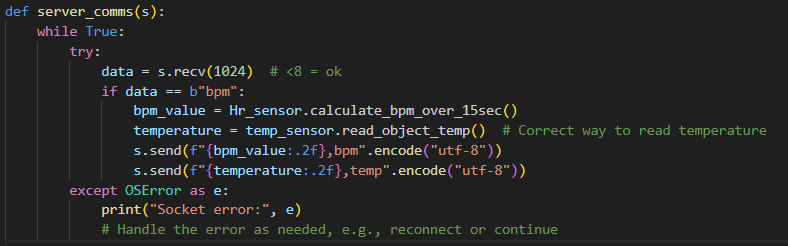
* This is only seen in the smartwatch
  + As the server will keep requesting for the smart watch every 30 seconds
  + This is so to ensure the server will not be overloaded with all the messages
    - As it will be ready to accept the smart watch
* It is similar to **Server to ESP client**

Steps:

Server

* It is first initialized by a scheduler
  + 
* First check if 30s has elapsed since the last sent time or initialized time
  + 
* **Server to ESP client (same process)**
  + Code used:

ESP Client

* Data is then sent
  + 

Server

* It will first search for the highest Frame buffer size from all the known users(default is 2048 bytes )
  + Since we **never** declared the size, the highest will be whatever the highest frame buffer from other users in that time
  + However, no matter what, the smallest size will still be 2048 bytes because the size declaration never changed as we never initialized it
* Then it will receive any information at a frame buffer of the highest known frame buffer from all its users
  + 
* Receive the data at the highest frame buffer size
* 
* Then find the id from the remote address
  + 
* See if it is a new client
* 
* If it is not a new client find the id of the ESP client sender
  + 