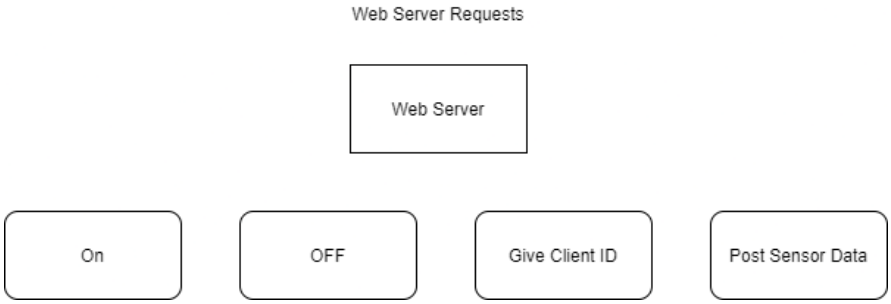
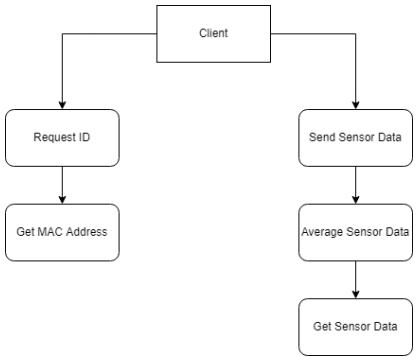
Guide for ESP32 Code	
Overview:	This guide will show how to set up the code to get the Server ESP32 set up with running its own webserver and the code to set up for the Client ESP32. The Server ESP32 will create its own wifi access point and the Client ESP32 will connect to the server and send over the sensor data. Any device that could connect to a wifi point will be able to connect to the Server ESP32 and manually turn on and off the pump along with see the fuel levels.
Requirement:	Server Microcontroller - ESP32-C3-DevKitM-1 Client Microcontroller - 356-ESP32-DEVKTC32VE IDE - Arduino IDE Manual Controller - Any device that could connect to a WI-FI
Server Code:	To get started the guide that is provided below was used to get a basic web server set up. The guide shows how to turn on and off an LED manually. The LED could easily be replaced by the pump for the torch fuel. It also allows a separate HTML file to be stored inside the board to have a dedicated HTML file rather than a weird Arduino method.  The figure above shows the required requests to make a basic torch communication system work. The web server and client utilize either just text or JSON to format the information. The on command turns on the pump while the off command turns off the pump manually. The Give Client ID command assigns an ID to the client. The Post Sensor Data command receives sensor data from the client and updates the information to the webserver. It also handles if the pump should be turned on or off. https://randomnerdtutorials.com/esp32-web-server-spiffs-spi-flash-file-system/
Client Code:	The client code was based on the guide below. The important part of the guide is mainly figuring out how to connect to a specific Wi-Fi so we can send over POST and GET requests to communicate sensor data. The client's goal is very simple: 1) Get an assigned ID from the server to establish a valid connection 2) Send sensor data over to the server. Below

	is a picture with the required functions.  As stated above, there are 2 main tasks for the client and that's getting an ID from the server and sending sensor data as shown in the flow chart. Going over requesting an ID first, requires 1 helper function which is Get MAC Address which sends over the ESP32's chip ID which is unique to the specific ESP32. The reason for this is so the webserver can keep track of which clients have connected in the past and if the client were to restart then it's able to find and match the client's ID again. For the sending sensor data, it requires 2 helper functions which are Average Sensor Data and Get Sensor Data. The Get Sensor Data is very straightforward and gets the sensor data through a built-in Arduino command "TouchRead(pin #)" which is used with a capacitive sensor. The Average Sensor Data function averages 5 sensor data that way whatever data is being sent over to the server is reliable and we can cancel out any outliers. https://randomnerdtutorials.com/esp32-client-server-wi-fi/
Testing Method:	The testing method for the code was fairly simple and it was just a manual test of just seeing how the communication between the server and the clients was handled. Making sure the connection between the server and the client was a big one and being able to control the LED/Pump manually through the web app and finally making sure the fuel level gets updated correctly and the pump turns on and turns off automatically depending on the fuel level.
Issues/Solution:	We did face some issues when starting. Initially, we planned on using a Nordic microcontroller rather than the ESP32 but after doing a little more research and with the help of Arduino commands we decided that going with the ESP32 was an easier choice. We also faced an issue with Bluetooth mesh which was what the team initially wanted to implement. When digging into it a little bit, we found out that it was actually more complicated than we thought so we decided to switch to a point-to-point Wi-Fi system which was an easier route to approach and proved our concept of using a wireless system.

Future Areas of Exploration:	Going with a bluetooth mesh or Wi-Fi mesh system or even the ESP-Now protocol which expands the range of the communication but it will also add more complications as now it's switched to a mesh network rather than point-to-point network. Add a feature where it only sends data over every x minutes to decrease the amount of necessary communication.