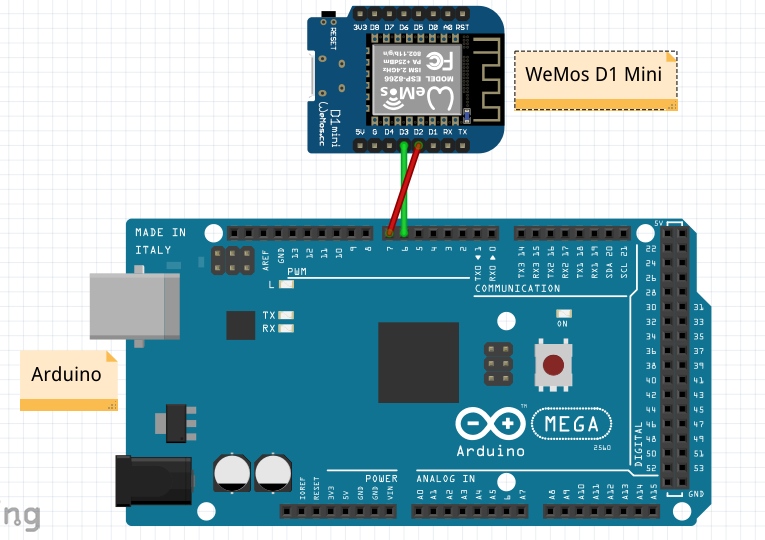
**Procedure**

**Arduino**

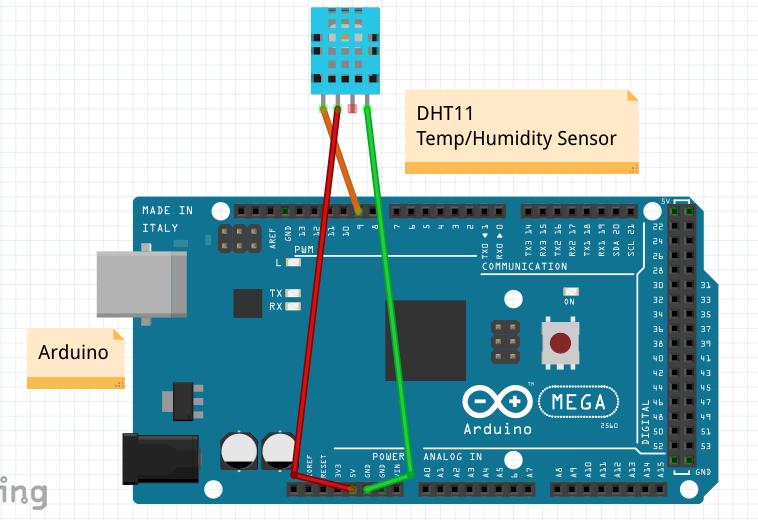
The hardware device was developed incrementally to ensure that each stage was successful and allow for appropriate testing and modification were required. By building the device in small stages, it allowed the team to track progress as well as served as a troubleshooting guide should an error occur. For example, if the device was behaving normally before a sensor was added, only to misbehave upon integration, it would seem the cause of the error lay with incorrect installation of the new sensor. This section will look at how each stage was carried out and how the overall solution was reached. Connection Diagrams will be provided for each stage, with a complete Diagram incorporating the whole hardware system included at the end of the section.

**Connect arduino and wemos**

****

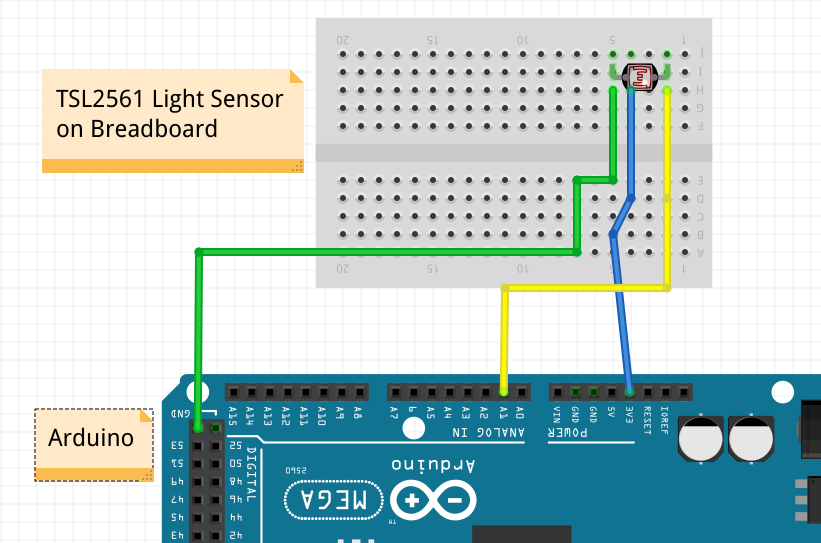
Before we could gather the required data from the sensors to send to the database, we had to give the Arduino the capability to connect to the Internet. This was achieved using a WeMos D1 Mini board. The WeMos is powered by a microUSB connection and uses two pins to send and receive data from and to the Arduino, making it relatively simple to connect and setup. The code running on the WeMos board consists of establishing a connection with the database, listening for the data sent from the sensors via the Arduino, placing the sensor values within a JSON string and finally querying the database and inserting the sensor values in the appropriate table. Once the WeMos has established a connection with the database, it will construct and send a JSON string every thirty minutes. Before any sensors were connected, dummy data was used to ensure that a WiFi connection could be set up between the device and the database server, as well as to ensure that the data was being stored in the database correctly. Once this stage had been complete, the next step was to connect a sensor and obtain useful data.

**Connect temp/humidity sensor**

****

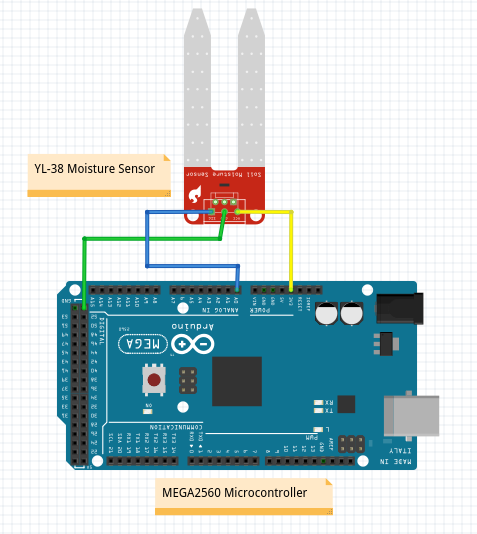
The DHT11 sensor was selected as it enabled the team to take readings on the temperature and humidity in one unit. This sensor requires three pins to operate. It is powered via a 5V pin, requires use of a GND pin and transfers data via a digital pin. A DHT library is also required for successful operation and enables the use of functions specific to the DHT11. The dht.begin() function starts up the sensor, whilst the dht.readHumidity() and dht.readTemperature() functions translate the data received from the sensor into a human readable format. These values are read as floats before being converted to String objects for use with the JSON string.

**Connect light sensor**

****

The TSL2561 was chosen for use as the light sensor as it is very simple and provides easy to use data. The sensor reads the amount of light falling on it in a given environment and outputs this data on a scale from 0-100. This allowed the team to save the data values as an int and display it to the user as a percentage with 0% meaning fully dark and 100% fully bright. The sensor requires three pins to operate: 3.3V to supply power, GND to ground the circuit and an analog pin to provide the data, in this case the A1 pin. A breadboard had to be used as the sensor is very basic and is not attached to a board to provide the pin out functionality. The breadboard can easily be replaced with female-to-male connection wires for a full production model, but the breadboard was kept in place as it allowed the team to daisy chain power connections for other sensors.

**Connect moisture sensor**

****

The YL-38 moisture sensor was chosen as it as again easy to connect to the system and provides data that is easy to interpret. The sensor is powered from a 3.3V pin, grounded via a GND pin and sends data through an analog pin, in this instance A0. The sensor works on the basis of a short circuit. The two prongs are connected to the power pin and the input pin and the difference in the returned value is used to calculate how much moisture is present in the given medium. Should the sensor be placed in wet soil, a connection between the two prongs will be established and the value returned will be lower than if the sensor is placed in dry soil where the current finds it harder to travel through the medium. Using this data, we can inform the user whether their plant requires watering or not by using a simple if-else statement to determine where the returned value falls on a scale, as well as collect the specific data value and use it to track how the soil dries out over the course of a day or week.

**Take readings and place in json string**

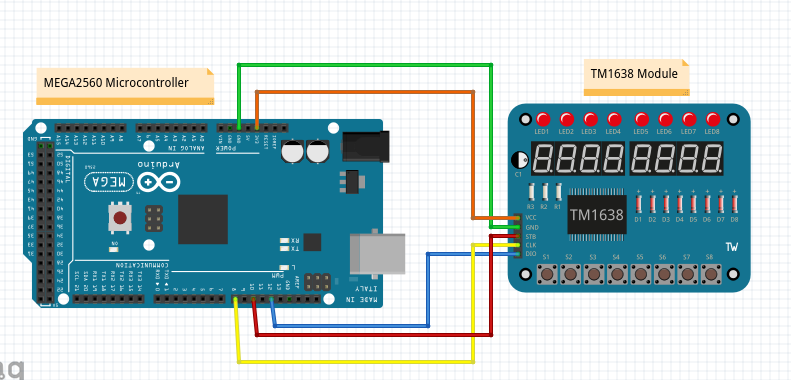
Once all the sensors were connected and providing useful data to the Arduino/WeMos sub-system, the next step was to place these data readings in a database. There was a number of options the team could have chosen, but the decided method was to insert all the data values into a JSON string and store this string within a database. The advantage of this method was that the database table only required two columns: one which held the unique identifier, in this case the MAC address of the WeMos, and a column which contained the JSON string itself. The team could then parse this JSON string using PHP and use the data to power the graphs on the website. The disadvantage of this method is that the JSON string won’t be sent to the database until it contains four separate readings [one for each sensor], meaning that database entries could be missed if a sensor became inoperational. The advantages here outweighed the disadvantage, which was deemed unlikely to occur unless the device or sensors were deliberately tampered with.

**Send json string to dB**

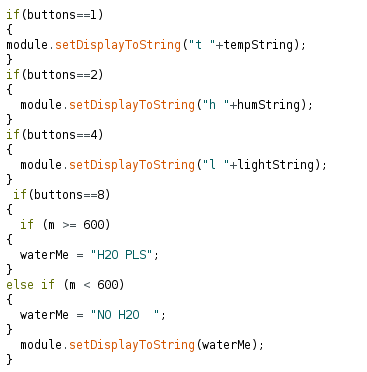
****

The WeMos is used to construct the JSON string before transferring it over WiFi to the database. The Arduino collects all the sensor data and passes it onto the WeMos via a digital pin. The WeMos listens for the sensor values and once it has collected all four required readings, a JSON string is created with the values being appended in the appropriate location. The WeMos then constructs a MySQL query consisting of the MAC address of the WeMos which is used as a unique identifier, the current time and the JSON string itself. Once the MySQL query has been executed the JSON buffer is cleared so that it can be used to construct the next JSON string. A delay of thirty minutes is set after which the process begins anew.

**Connect LED display**



Whilst the data collected from the sensors has more use in powering graphs in place on the website, a simple display was implemented so that the user can see the real time data collected from the sensors. A TM1638 module was selected as it contains eight seven-segment displays that can be used to display basic letters and numbers and eight buttons which can be used to make selections. A TM1638 library was required to access the functionality of the buttons, via the module.getButtons() function, and allow data output to the seven-segment displays, via the module.setDisplayToString() function. The team decided to use the first four buttons to switch between which sensor reading was currently displayed, with the default display and button showing the temperature reading. The second button was set to display the humidity reading, third to display the light level and the fourth to display whether the plant required watering or not. First all of the sensor values had to be converted to String objects as this is what the seven-segment displays required. A series of if statements were then used to interpret which button was pressed and the appropriate reading would be displayed to the user [see below code snippet]. The module used also benefits from requiring only 3.3V to run, lessening the overall power draw of the system. More complex displays could have been used, but the data displayed would remain much the same but the power draw of the device would be increased.



**Complete system circuit diagram**

