# Solar Picnic Table

Azzan Al Busaidi, Thach Vo, Amos Luo, Faisal Al Ghafri

Group 18

University of Central Florida School of Electrical Engineering and Computer Science, Orlando, FL

***Abstract* — A project that combines high-level concepts from the fields of Electrical Engineering and Computer Engineering. Our Solar Picnic Table is dedicated to make any outdoor experience more convenient while maintaining the global standards of an eco-friendly product. Using a monocrystalline solar panel, this is a project that powers the needed amenities demanded on any outdoor trip utilizing energy obtained from solar arrays. The solar panel delivers power to two different circuits, an alternating current circuit, and a direct current circuit. The AC circuit accompanies features like weather information gathering and motion triggered LED lights, controlled by a microcontroller embedded on a printed circuit board. The printed circuit board is designed to manage the sensors on the table and deliver gathered information to the user via a bluetooth phone application. The DC circuit on our table provides on-demand power delivery to the consumers via a power outlet and a USB outlet. Furthermore, a lead acid battery is utilized to store excess energy and supply it to the table for night time usage.**

***Index Terms* — Bluetooth low energy, solar power generation, solar panels, microcontrollers, motion sensors**

## I. Introduction

Made to withstand weather conditions and help make the most out of any outdoor trip, our Solar Picnic Table incorporates modern technologies to deliver the most demanded amenities on a picnic table according to the market. Our design is made in a way that minimizes the effort and cost of maintaining the product, while picking efficient components, which gives our project excellent durability. Since it is our duty to care for the environment, we chose to power our table with a highly efficient monocrystalline solar panel. The panel then delivers the energy it gathers to the charge controller that dissipates it to the two circuits on the table, while also protecting our battery cell from overcharging/discharging. The first circuit mainly feeds our printed circuit board and sensors. The printed circuit board has two voltage regulating circuits, the first regulates the voltage inputted by the panel and feeds the microcontroller chip and the temperature sensor, while the second circuit regulates the input voltage and supplies the motion sensors and LED lights with power. In addition to that, we incorporated a voltage sensor on the printed circuit board that indicates the remaining voltage on our battery. The microcontroller embedded on our printed circuit board manages the information gathered by all the sensors on the table, and sends that information to a phone application that we designed to make the table easy to use. This phone application then shows all information gathered by the sensors to the user in a user-friendly interface, and enables the user to manually control the LED lights. As for our second circuit on the table, its duty is to deliver power to an inverter that inverts the direct current received from the solar panel to an alternating current that is then available for use either via a power outlet, or a USB outlet. Since our table uses a solar panel as a source of power, we accompanied our design with a lead acid battery that stores excess energy from the solar panel, and supplies that energy to the loads as needed.

## II. System Components

The solar picnic table is composed of different components. Each is essential to the functionality of the project. In this section, these components will be briefly introduced.

### *A. Microcontroller*

The microcontroller is one of the most important components of the picnic table. The MCU in use is the Espressif ESP32-WROOM-32E-N16. Equipped with 38 pins, 4 MB of flash memory, low power consumption, and a clock speed of up to 240 MHz allowing quick computations, this microcontroller is the best fit for the project.

### *B. PCB*

The printed circuit board lies at the core of our project, serving as the pivotal component responsible for regulating the 12VDC input from the charge controller. Its primary function is the precise conversion of this input into 5VDC and 3.3VDC, tailored to accommodate the distinct power needs of both sensors and the microcontroller unit. This precision is ensured through the inclusion of two dedicated voltage regulators, the TPS5410DR and TPS54331, which not only guarantee efficient power transformation but also maintain stable and reliable power supplies across the entire system.

Central to the PCB's architecture is the strategically positioned ESP32 MCU, functioning as the project's command center. The MCU controls data processing, executes commands, and ensures seamless coordination among the interconnected components. Augmenting these capabilities, the PCB incorporates a specialized voltage reading circuit that provides real-time feedback on power levels of the battery, enhancing the system's adaptability to dynamic operational conditions. Furthermore, connector pins on the PCB facilitate

the integration of temperature and motion sensors, enabling precise data collection. The design of the PCB not only ensures efficient power distribution to all connected components but also facilitates communication channels between the MCU and the array of sensors.

### *C. Solar Panel*

One of our goals as a team for this project is for it to be self-sufficient and sustainable. This is why the solar panel is the only power source in our project that powers all connected components.

In the process of choosing one we had various constraints and requirements that had to be taken under consideration. The panel had to be able to power the whole system while simultaneously charging the battery storage system. In order to achieve this the correct power rating and efficiency of the panel was important to calculate. Also we had various solar panel technologies options that all affect the functionality of our project. Our final decision was to go with the 50 Watt 12 Volt Renogy Monocrystalline Solar Panel.

### *D. Battery Storage System*

The battery storage system in our project plays a crucial part in providing the necessary sustainability. During the hours when there is no sunlight our battery is able to power all of the components in the system simultaneously. The voltage rating of our battery matches the voltage rating of the solar panel which is 12 Volts. The battery that we are using in our system is 12V 8AmpHour EXP1280. So the battery's rating is $Energy\ Rating = 8\,Ah\ *\ 12\,V\ =\ 96\,Watt\ Hour.$ If we assume that the total consumption of all loads in our system is 20 Watts which is an overestimation, then the battery will ideally last for 4.8 hours. The following formula is the calculation.

$Usage\ time\ =\ (96\,Watt\ hours)/(20\,Watts)\ =\ 4.8\,hours$ .

### *E. Charge Controller*

The charge controller is responsible for regulating the charging of the battery storage system by the solar panel. Since all batteries have a charge capacity limit, charge controllers are needed to ensure that that limit is not crossed, nor that the battery is over-drained. To do that, two voltage levels are determined: the first is a High Voltage Disconnect point (HVD), at which the battery is full, and the charge controller is signaled to stop the current supplied by the source from reaching the battery. The second one is a Low Voltage Disconnect point (LVD), at which the battery needs to be charged.

The two main choices of technologies of the charge controller was either the Pulse width Modulation (PWM) charge controller or the Maximum Power Point Tracking (MPPT) charge controller. Both are good choices but after careful consideration we decided to use the PWM charge controller.

### *F. Motion Sensor*

The motion sensor used in this project is the AM312 motion sensor module. This sensor utilizes passive infrared technology to detect motion up to 5 meters in a 100 degree cone angle. In addition, it has low power consumption, an operating voltage range of 2.7-12 volts, and is small in size. Taking these features into account, it is the perfect choice for the table. The main purpose of these sensors is to keep track of who is approaching or occupying the table and control the LEDs based on these conditions.

### *G. Temperature Sensor*

Another sensor that will be used is the DHT22 temperature and humidity sensor. Like the motion sensor, the DHT22 has low power consumption and an operating voltage of 3-5 volts. It comes with a temperature reading range of -40 to 80 degrees celsius and a +/- 0.5 degree celsius accuracy. On top of this it is able to read in the surrounding humidity and output it as a percentage with an accuracy of around +/-2 percent.

### *H. Battery Sensor*

One functionality of the picnic table is the battery sensor. Essentially, the sensor is a voltage divider circuit built within the PCB that will cut down the 12 volt battery into 3.3 volts for the ESP32 to read and output to the mobile app.

## III. SYSTEM DESIGN

The solar picnic table is made up of various parts that work together to ensure the functionality of the table. In section II, we highlighted the main components that were a part of the table. In this section, we will be going into depth about those components and their design and how it interacts with the other pieces to make the solar picnic table function properly.

### *A. Microcontroller and Sensors*

The ESP32 and sensors are interfaced with each other using GPIO pins. In total, five GPIO pins are used. Note that this number only accounts for the data pins not the others. Furthermore, the sensors and MCU use the 1-Wire communication protocol. This protocol consists of a controller device that communicates with a peripheral device through one wire.

Regarding the motion sensors, the solar picnic table will be fitted with four. Each will be placed at each corner of the table. Since the max coverage of a single motion sensor is 100 degrees, having four of them ensures that the table has 360

degree detection coverage. This means that the table will be able to sense motion from any angle. These sensors come with 3 pins: power, data, and ground. They will be connected to the ESP32 using jumper wires. Doing this will also allow more freedom to place the sensors in the right location.

Next, the ESP32 will be connected to a single DHT22 temperature and humidity sensor. The connection is similar to AM312 motion sensors in that it requires the same 3 pins. The sensor will also be connected to the ESP32 using jumper wires. This sensor will be placed further underneath the table as it can measure temperature from anywhere.

Finally, the battery sensor, unlike the others, is built integrated within the PCB. The voltage divider circuit will be connected to the 12 volt power source and run into the ESP32. For this design, it requires specific pins to be connected. To properly read in voltage levels, an analog-to-digital converter (ADC) must be used. There are many ADC pins available on the ESP32 module, for this, GPIO33 was used. By using this, the ESP32 will be able to read in the battery level as a digital value allowing this value to be more easily manipulated when coding.

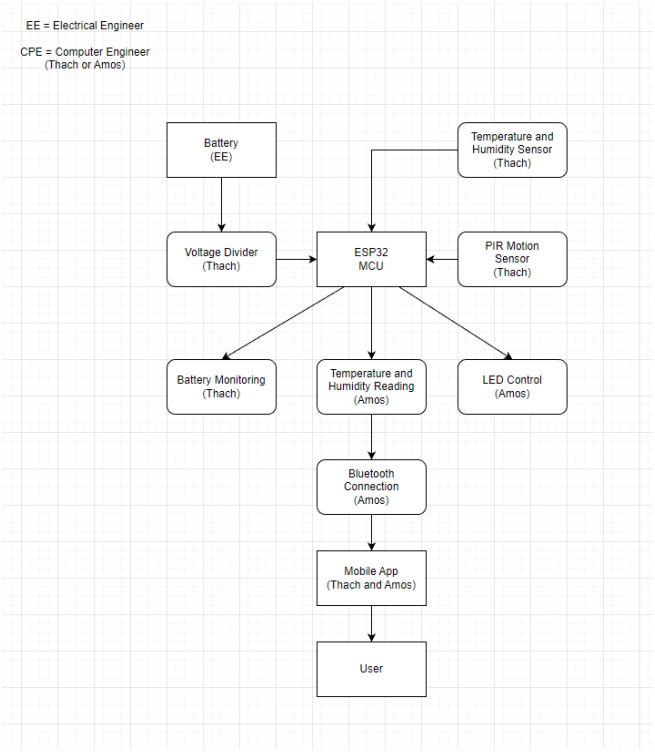



Fig 1. Block diagram of the system design regarding the MCU, sensors, and App.

*B. PCB*

As we discussed, the PCB is one of the main components in our system. In this section we will provide a detailed explanation of its design. The explanation is going to be divided into three parts. First part being the voltage regulation, then the output connectors, and on PCB peripherals.

For voltage regulation we have two circuits. One common input voltage and two separate output voltages. The overall design idea for voltage regulation and distribution is shown in the figure below.

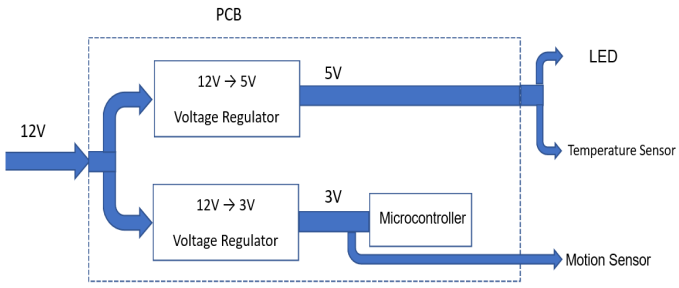



Fig 2. PCB Voltage regulation

The first circuit uses the voltage regulator from TI TPS54331, this voltage regulator has an input range of 3.5-28 volts, 80-90 % efficiency, and an output voltage of 5 volts using the equation $Vout = Vref * ((R1 + R2) / R2)$

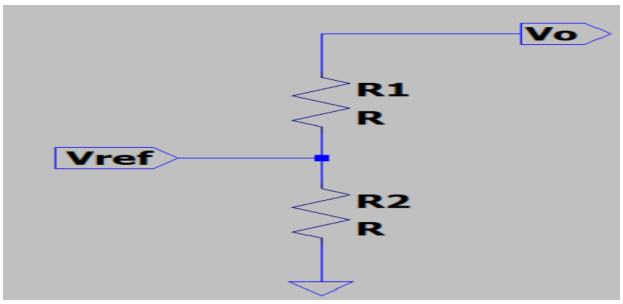



Fig 3. Regulation voltage divider

The typical value of the reference voltage for this regulator is 0.8V, so in order to obtain an output of 5V we used these resistor values, R1=10k ohms and R2=1.96k ohms. It was recommended to use a 10.2k resistor instead of a 10k one which would have resulted in a more accurate output voltage. But we used the 10k resistor because of its abundance.

$$Vout = 0.8 * ((10k + 1.96k)/1.96k = 4.88V .$$

The second voltage regulation circuit also uses a TI voltage regulator. But in this circuit we are using the TPS5410 to obtain the 3.3V output. Using the same regulation voltage divider circuit in Figure 2, The equation

to obtain the 3.3v is the following $Vout = (R1 * Vref + R2 * Vref) / R2$. For this regulator the datasheet mentions that the typical Vref value is equal to 1.221. So using resistors R1=10k and R2=5.9k ohms we have an output of 3.3 volts.

$$Vout = (10k * 1.221 + 5.9k * 1.221)/5.9k = 3.29V.$$

Now that we have achieved the output voltages desired, We will use connector pins for connecting the sensors to the power supply. We have 4 motion sensors and 1 temperature sensor so we are using 5 3-pin male connectors as shown in the figure below.

Fig 3. A 3-pin connector

These connectors have a pin for power supply (5v or 3.3v) , a ground pin, and a communication pin with the MCU. A similar connector is also used on the PCB for connecting the LED which uses 4 pins. To connect the LED 1 pin is for input voltage (5v) and the other three are all connected to the mcu. In our design, we've also incorporated three 1-pin connectors, each dedicated to the 3.3V output, 5V output, and ground, respectively. This addition serves a specific purpose: simplifying the post-soldering testing process. By introducing these connectors, we aim to streamline the testing phase, making it more accessible and efficient.

The individual 1-pin connectors provide a convenient interface for connecting jumper wires directly to the PCB and, subsequently, into a breadboard during the testing phase. This design choice allows for an easier and modular approach to testing, facilitating the quick and secure attachment of testing equipment. The 3.3V and 5V connectors cater to the specific voltage outputs of our PCB, while the ground connector establishes a stable reference point for the testing setup.

This user-friendly testing methodology not only expedites the debugging and verification process but also enhances the overall ease of handling and troubleshooting during the initial stages of the project. It underscores a commitment to practicality and efficiency, ensuring that testing and validation procedures are seamlessly integrated into the project workflow. As a result, the 1-pin connectors contribute to a more user-friendly and accessible experience, allowing for straightforward and reliable testing of the soldered PCB before full deployment in the project environment.

One feature on the PCB is a voltage reading circuit that will provide real time data on how much percentage of charge we got in our battery storage system. This is achieved by having a simple voltage divider circuit which will take the input voltage and divide it by 3.636. The circuit is shown in the figure below.

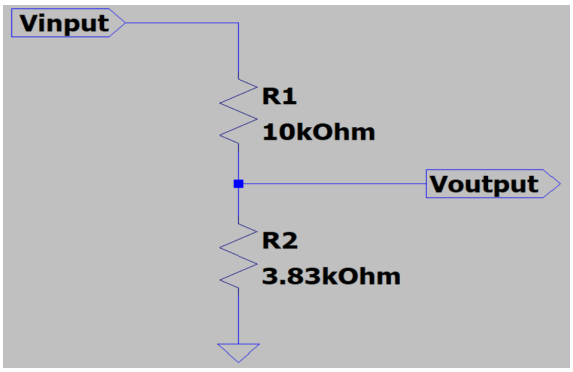



Fig 4. Voltage divider

So if the battery is fully charged and the input is 12 volts then the circuit will divide it and result in 3.3 volts which will be read by the mcu and converted into a percentage value. The same goes for any input value between 0-12 volts. If the Vout exceeds 3.3 volts we have a zener diode that will protect the mcu from that.

On the PCB we are using two push buttons, one is for the mcu to go into reset mode and the other is for it to go into boot mode. These two are needed when uploading the software into the mcu.

To go into reset mode, the enable pin which should always be pulled high for the mcu to be turned on must be grounded, this is done by pushing and holding the button in the following circuit. The 10k Ohm resistor functions as a pull-up resistor, this is essential to the functionality of the MCU as it ensures that the enable pin is pulled high unless the button is pressed.

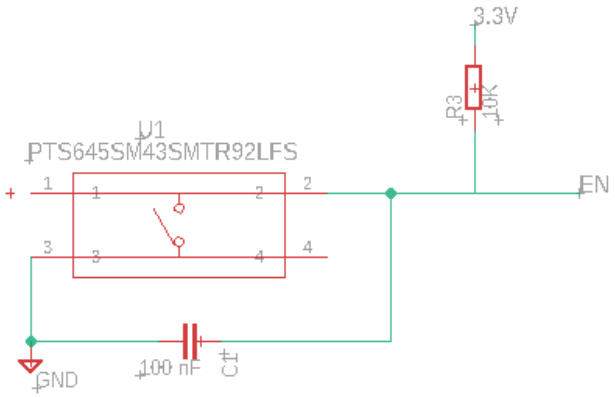


Fig 5. Reset button schematic

Then, to go into boot mode a similar circuit is used for the IO0 pin. The mcu goes into boot mode by grounding this pin, which should always be pulled high otherwise. Similarly, IO0 should be kept at high when working with the MCU, IO0 should only be grounded when the user wants to boot the MCU in download mode.

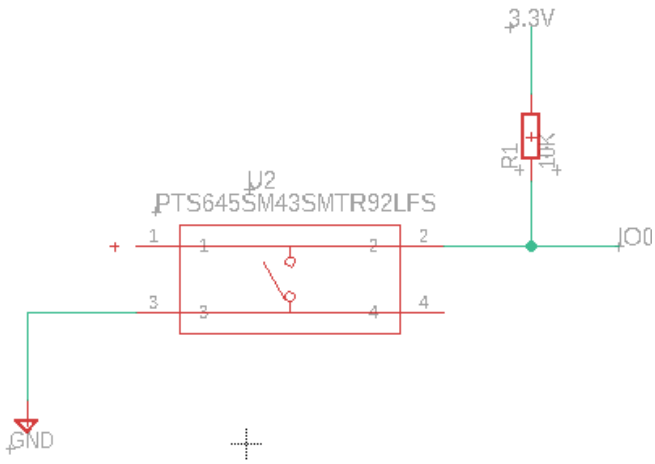



Fig 6. Boot button schematic

As for our board design we approached it by tracing the two circuits on our schematic. By using this approach, the connected components are the closest to each other on the board, making the traces between them shorter. All components connected to external devices out of the printed circuit board are placed on the sides and edges of the board, making our connections easier and clearer to avoid any chances of error. Furthermore, we added one pin connectors to our printed circuit board for the purpose of testing. Each of the one pin connectors is connected to an output voltage that the printed circuit board produces, in addition to a ground pin.

Our microcontroller chip is a vital part for our design, so we took protection measures to ensure that it works perfectly. We placed our microcontroller at the top of our board design with the bluetooth chip sticking out of the board, this prevents chances of the corruption of the signals sent by the microcontroller to the phone application. In addition, we added a TI *MMSZ4684-TP* zener diode that protects the microcontroller from getting any voltage above 3.3V, preventing any chances of the microcontroller getting fried.

Another feature we added on our board design is via stitching. Since both copper pour planes on our printed circuit board are ground planes, via stitching was an important feature to add. Via stitching enables us to ensure the connection of both ground planes so that we have one common ground on our PCB. The figure below shows the placement of the microcontroller on the PCB, in addition to showing a part of the via stitching.

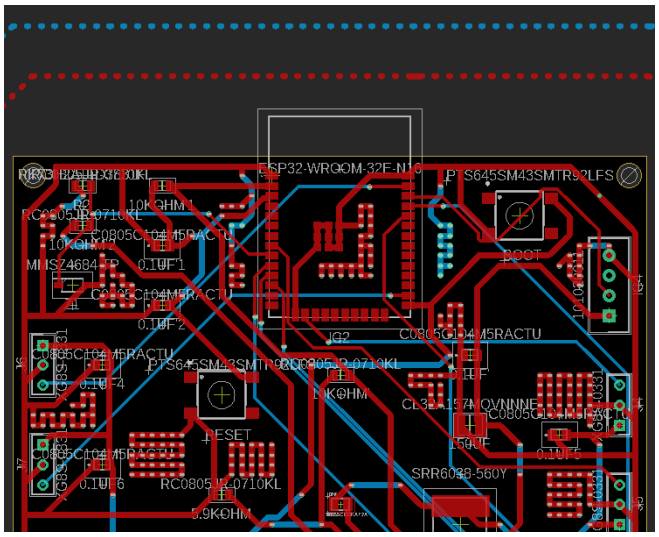



Fig 7. MCU location on the board

## *C. Table Design*

The table design can be separated into a few sections, the electronics box, the table, and the outlet box on top of the table. A visual example of this is shown in figure 7.

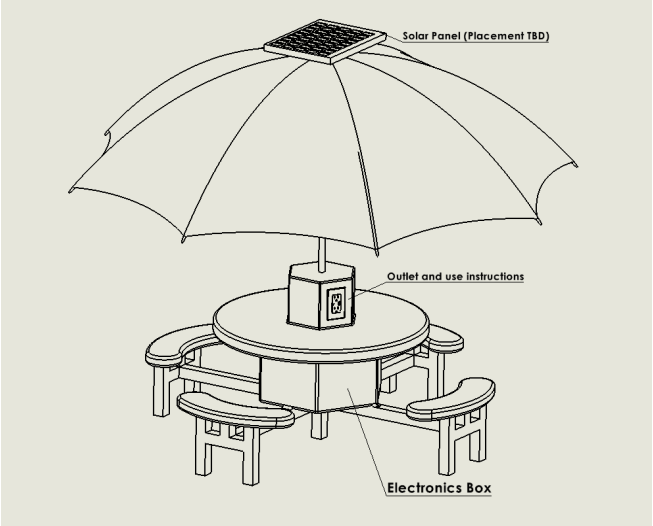



Fig 8. Table Prototype

The top of the table will consist of a 3D printed enclosure that holds the AC/DC inverter which also provides the user with an outlet and two USB ports. The bottom electronics box will be a plastic container that is bolted to the bottom of the table and will consist of the PCB, charge controller, and

the battery. This plastic container will have a rubber seal around the surface where it mates with the table to ensure a waterproof seal. The solar panel will be bolted onto a metal frame that is attached to a wooden pole that drives through the table to the ground. This metal frame will have the most optimal angle for the most efficient solar energy production.

The table is a table bought off of Amazon and required a few modifications. As shown in figure 8, we have replaced the plastic pole in the middle to provide more stability for the solar panel.

Figure 9. Image of purchased table

With the replaced pole, we can also run wires through the wooden pole to hide as many cables as possible. On the four corners of the table, the motion sensors are placed to provide the most coverage.

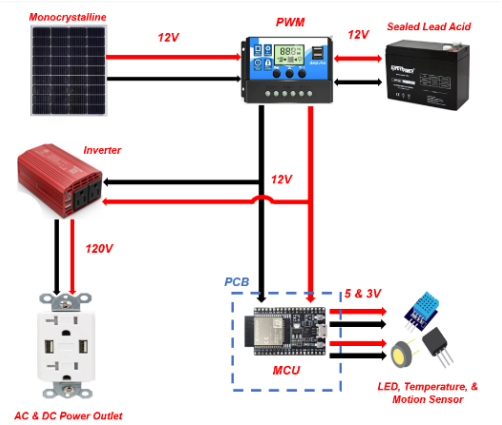



Fig 10. Overall system design and connections of the components

## IV. SOFTWARE DESIGN

In this section, the software aspect of this project will be explained. The picnic table's software functionalities can be split into two parts: the microcontroller's firmware and the mobile app.

### *A. Microcontroller Firmware*

The firmware for the ESP32 is written in the C++ programming language and was created and compiled within the Arduino IDE. This aspect of the software serves one main purpose: to implement the different sensors' functionalities onto the ESP32 and to allow communication to a mobile app via bluetooth. To start off, the code utilizes several libraries including the DHT (used for the DHT22 sensor), BLEDevice, BLEServer, BLEUtils, and BLE2902 (all used for bluetooth low energy connection) [1], [3]. There are a multitude of variables used for pin definitions, BLE characteristics, and BLE server setup. These pin definitions are used to set up the sensors and assign them to certain pins on the MCU.

The software is able to manage motion events efficiently and accurately. To do this, the code utilizes an interrupt service routine (ISR). Whenever motion is detected by the sensors, the interrupt calls upon the motionISR() function. This function is fairly straightforward ,however, it is accurate in determining motion and is able to filter out false positives fairly well. Essentially, the function subtracts the current time that the interrupt is triggered with the last time that motion was detected and compares it with a delay value. In this case, it is 1 second. If the value is smaller than the delay, it keeps the LEDs turned on else it will turn them off.

Another element of the table that the software controls is the temperature sensor. The DHT22 is implemented within the code using its own specialized library. This makes the usage of the sensor fairly simple as within this library, it

contains all of the necessary functions required to take in readings of the environment and output them to the user.

It is important for the user to know how much battery charge is available within the solar picnic table. This is accomplished through the code. Physically, the voltage divider circuit within the PCB allows for the ESP32 to be able to read in the voltage levels. However, the code is able to convert this value into something readable. To do this, the map() function is utilized to convert the raw value into a percentage. To put it into more detail, the analogRead() function is used to read in the value of the ADC pin. This value can range from 0 to 4095 as it is a 12-bit ADC. From there, the ESP32 can use this value and convert it to a percentage of 0 to 100 using the map() function.

Inorder for the mobile app to be compatible and able to communicate with the ESP32 is for there to be a bluetooth connection. This is accomplished through bluetooth low energy (BLE). As mentioned previously, the firmware contains various different BLE libraries that contain a multitude of functions that help establish a bluetooth server. The bluetooth characteristics and services are defined using Universally Unique Identifiers (UUIDs). Within the setup() function, the BLE device is initialized, the server is created, and the service is set up with the bluetooth characteristics all which notifies them allowing the ESP32 to send updates to the mobile app.

### *B. Mobile App Software*

The mobile application was written in JavaScript and utilized Expo, which is an open-source platform for making universal native apps that run on Android and iOS with JavaScript and React. Expo plays an important part in the software development process because it allows us to program an application for both Android and iOS without having to switch languages. Expo also simplifies the testing process for simulating the application because it has pre-made packages and dependencies for both Android and iOS platforms. A visual example of how Expo works is shown in Figure 11.

Expo also creates a local server that runs the application and allows for real-time testing on simulators such as Xcode, which is Apple's built in platform for creating apps, or allows the user to download the application to their phone which allows for furthering testing such as testing the phone's bluetooth connection. In the demo, we will be utilizing this feature to save time and money. Uploading an app to Apple's app store requires an annual fee of $99 and takes a few weeks for it to go through an approval system.
The mobile application uses a few libraries from React/React Native that allow it to utilize phone capabilities. One of the most important libraries is "BleManager" which allows us to use the bluetooth features in both Android and iOS. This bluetooth manager library allows us to connect our phones to the ESP32 and allows for communication between the two. Another library that is very important includes a function called "atob" which decodes the messages sent from the ESP32. This allows the application to decode the message encoded in Base64 from the ESP32 and convert it to a character variable which allows the app to display it.

The mobile application is split into two main groups, the functions and the interface. One of the most important parts of the application is the function which includes connecting to the ESP32.

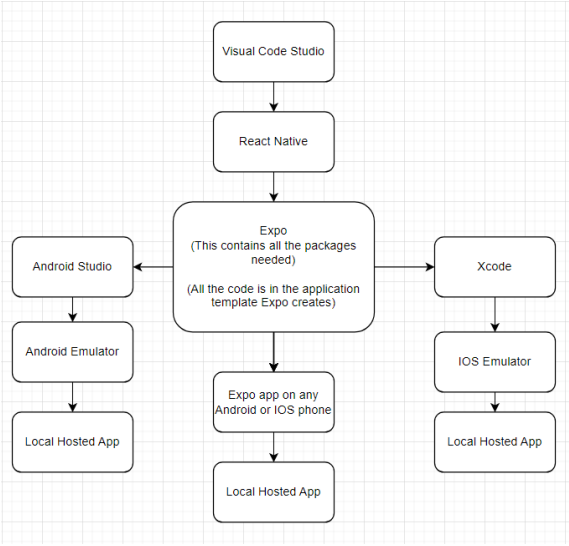



Fig 11. Structure of the App

The function to connect to the ESP32 starts off with starting a "startDeviceScan" function from the "BleManager" library. This function uses the phone's bluetooth capabilities to search for a specific bluetooth device name, in this case, we labeled it as "ESP32 SD" in the Arduino code for the ESP32. Once connected, the function closes the "DeviceScan" and checks if the connection is completed, if not an error is thrown out and retries another time. Once connected, the app uses the "BleManager" again to read in all the characteristics. Characteristics is how data is sent over bluetooth as shown in figure 12. Each characteristic has an unique ID that is generated using a website and was set within the code uploaded on the ESP32.

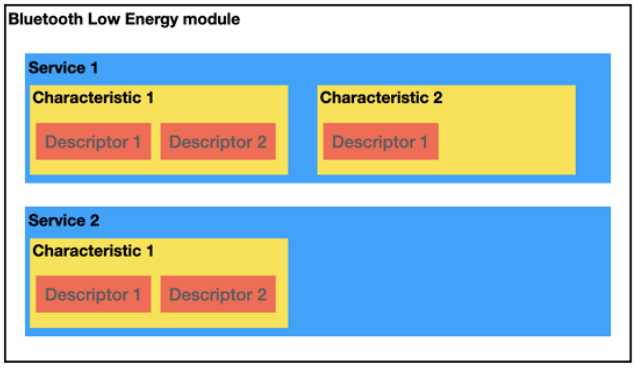



Fig 12. Bluetooth Data Structure

The ID's are set as constant variables in the code and the "BleManager" looks for those ID's and reads in their value and decodes it into a character variable. This step is necessary because React Native cannot display the raw value that is received from the ESP32. Once that is done, the code is displaying all the values. To disconnect the device, there is a function that is activated whenever the button is pressed. This just tells the "BleManager" to disconnect all

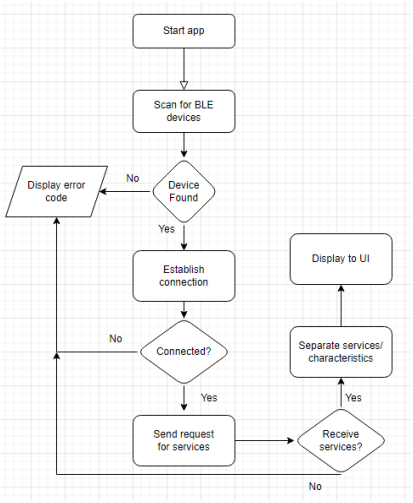


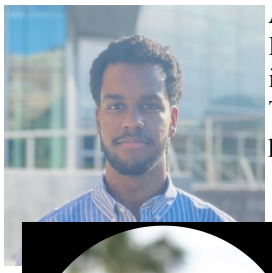

Fig 13. Basic Functions of the App

devices and checks for any errors and then updates the connection status. The flowchart of how these functions work is shown in figure 13.

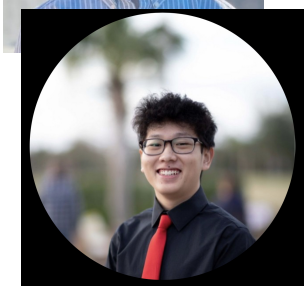
The UI portion of the mobile app uses some HTML/JavaScript and some built in features of React/React Native. Most of the functions are synced to a button so once the button is clicked, the function is activated.

## The Engineers

**Thach Vo** is a 23 year old that will be graduating from the University of Central Florida in Computer Engineering. Thach plans to pursue a software focused career and work for tech companies such as Apple or Microsoft.



**Azzan Al Busaidi** is a 23 year old Electrical Engineering student enrolled in the Power & Renewable Energy Track. After graduation he plans to pursue a career in the power industry.



**Amos Luo** is a 22 year old Computer Engineering Student at the University of Central Florida. He is planning on pursuing a software engineering job at companies like Amazon or Dropbox.

**Faisal Al Ghafri** is a 22 year old Omani student who achieved a scholarship to study Electrical Engineering at the University of Central Florida. Faisal is graduating this semester and he is already on the look for a job to apply the knowledge he earned in the field of power and renewable energy.

## Acknowledgement

The authors wish to acknowledge the assistance and support of Dr.s Arthur Weeks, Chung Yong Chan, Samuel Richie, and Lei Wei; University of Central Florida.

## References

[1] "adafruit/DHT-sensor-library," *GitHub*, May 04, 2020. https://github.com/adafruit/DHT-sensor-library

[2] "Bluetooth technology overview," Bluetooth® Technology Website. [Online]. Available: https://www.bluetooth.com/learn-about-bluetooth/tech-overview/.

[3] N. Kolban, “ESP32 BLE for Arduino (This repository is kept for archive. BLE code is now included in Arduino directly.),” *GitHub*, Mar. 13, 2023. https://github.com/nkolban/ESP32_BLE_Arduino