**Smart Home Energy Management System**

**(SHEMS)**

*Ryan Mears*

*Faculty Advisor: Dr. Kourosh Sedghisigarchi*

*California State University Northridge*

*January 2, 2018*

# ***Abstract— The Shems project is a complete design with improvements from a previous senior design. Shems consists of four subsystems: the database, implemented with the use of xampp and aws; mobile app, made with android studio ide; shems circuit; and microcontroller, controlled with an Arduino nano and esp8266 module. the integration of the 4 subsystems defines the Shems project, which provides users the ability to view power consumption of devices in their home with a user-friendly mobile app.***

# ***Keywords— Smart home, Android App, Arduino, Energy Management***

# I. Introduction

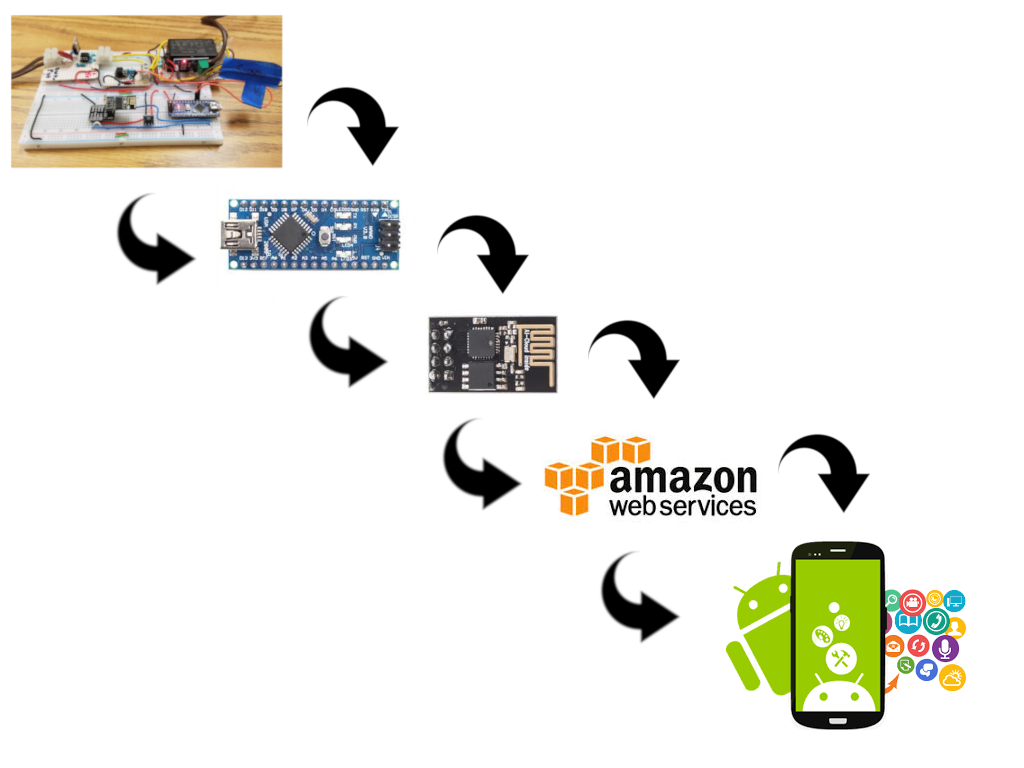
This semester’s goals were to take the existing design, model, and programming from Spring 2018, and to improve upon these to create a refined project. This paper will be focusing on the modification on the code of the Arduino microcontroller and the database with the goal of fulfilling two primary goals.

1. Database - Move the database from the local XAMMP server to an Amazon Web Services (AWS) server. The use of an AWS server allows for online accessibility of the database making the project far more portable, as well as gaining the benefits of increased security that AWS offers in comparison to a local server.
2. Communications- The microcontroller must be able to send data collected from SHEMS circuit to AWS database and the mobile app must be able to retrieve the same data in order to provide an accurate cost analysis and visual displays for the user.

# II. Project Overview

The Smart Home Energy Management System (SHEMS) project hardware is constructed in four primary sections. These individual sections each do a specific part, and the data retrieved from the individual sections, processed, and send to a user-friendly application to be viewed. We utilized these main components overall while changing and optimizing parts within these main groups. Three of the four main groups are the Current Sensor, Voltage Sensor, and the Microcontroller. The fourth component enables switching, which gives the user the ability to turn on and off the load device if desired. This is done through a Triac.

The basis of this device is centered around the voltage sensor and the current sensor. with these two devices, we can measure the phase angle, Vrms, and Irms, which in turn can all be used to calculate the Apparent Power, and the Power Factor of whatever load may be applied to the device. This data is sent to the Microcontroller, where it is calculated based on embedded code. Once this has been calculated, it is sent to the database, where the data can be viewed and the device controlled via a user-friendly cell phone application.



*Figure 1. Block diagram of the system*

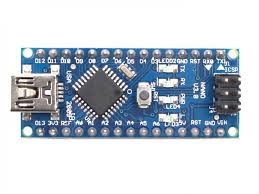
# III. Project Overview

## Arduino Nano

The SHEMS unit was implemented with an Arduino Nano microcontroller. The benefits that the come with choosing the Nano include small size, highly compatible programming language, as well as the relatively low cost. The Nano is equipped with the ATmega328 microcontroller with 32KB flash memory, of which 2 KB is used by the bootloader.

The Arduino Nano was selected instead of the NodeMCU ESP8266-12E in part to the Nano’s more affordable pricing ranging from $10 to $20 in addition to the more compact size with a PCB size of dimensions 18mm x 45mm. However, the primary benefit of using the Arduino Nano is that the analog input pins are capable of measuring the voltage and current readings required for power calculations including real power apparent power, and power factor without the need to include external hardware such as an analog to digital converter (ADS1115) as required in past SHEMS designs.

A drawback to using the Nano is the lack of integrated WiFi capabilities on the device. To rectify this problem, it is necessary to include an external WiFi module in order to transmit the power calculations to a network database. To this end, the ESP8266-01 is the WiFi module used to transmit the data to the database.



*Figure 2. Arduino Nano Microcontroller*

* 1. ESP8266-01 WiFi Module

The wifi module on this project in itself was not changed. What was modified about the wifi module was how it was connected. The wifi module requires a 3.3V input. The voltage on our board provided by the power supply was 5V. A voltage divider was designed and implemented to provide a clean 3.3V to the wifi module base that connected the module itself to the board, where connections such as power and data could be made. We were able to find a different base for the wifi module, directly replacing the old one, but it included a voltage divider in it. This wifi module base accepted a 5V input then provided the wifi module itself with the 3.3V it needs to operate. Making this change directly replaced an existing part, while also eliminating another component, the voltage divider, making the overall size and cost less.



*Figure 3. ESP8266-01 WiFi Module*

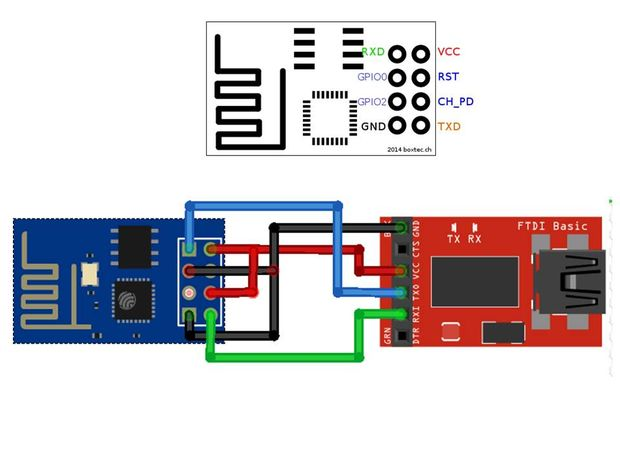
## ESP 8266-01S0 WIFI Shield Pros/Cons

The ESP8266 is an inexpensive WiFi module that will meet the requirements needed for most projects that include the need to connect to the internet. The module comes equipped with a range of capabilities such as 2 pins used for GPIO (General Purpose Input Output) and support various functions such as I2C, SPI, and UART. Fortunately, this module boasts high compatibilities with programming IDEs (Integrated Development Environment), including the Arduino IDE. Incidentally, the ESP8266 has since gathered a large community of users who have then published a large quantity of third-party documentation on the ESP8266 and its various functions. One consideration to keep in mind when implementing the ESP8266 is that the input voltage should not exceed 3.3 volts for risk of damaging the module. This requirement necessitates the use of either a voltage divider to reduce the incoming voltage to the module from the standard 5V input or alternatively include the use of an additional power supply. For this project, the benefits of implementing the ESP8266 far exceed the drawbacks that the module possesses.

## ESP8266-01 Setup

Before the ESP8266-01 WiFi module can be implemented, it must first be set up by downloading the appropriate firmware by following the steps below:

1. Connect the ESP8266 to FTDI module as shown in figure 4.



*Figure 4. ESP8266-01 WiFi Module to FTDI module*

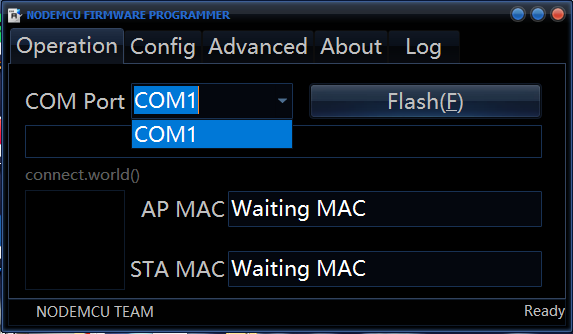
1. Download NODEMCU Firmware Flasher (hit clone or download button on website) <https://github.com/nodemcu/nodemcu-flasher>
2. Download firmware from Github.

<https://github.com/jayraj4021/Personal-Weather-Station-14/blob/master/v0.9.2.2%20AT%20Firmware.bin>

1. Connect ESP8266 to FTDI module and plug into USB port of a computer.

*\*note ESP8266 Rx to FTDI Tx and ESP8266 Tx to FTDI Rx*

1. Open NODEMCU flasher and select COM Port the ESP is connected to.

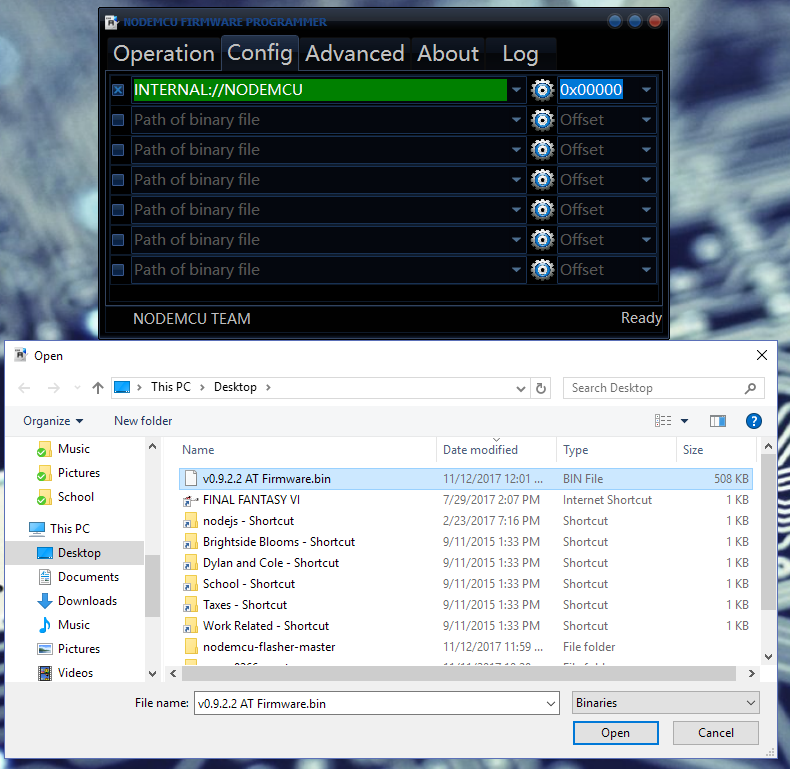


*Figure 5. NODEMCU Flasher*

1. If COM Port is not showing automatically, a driver must be installed for FTDI Basic.

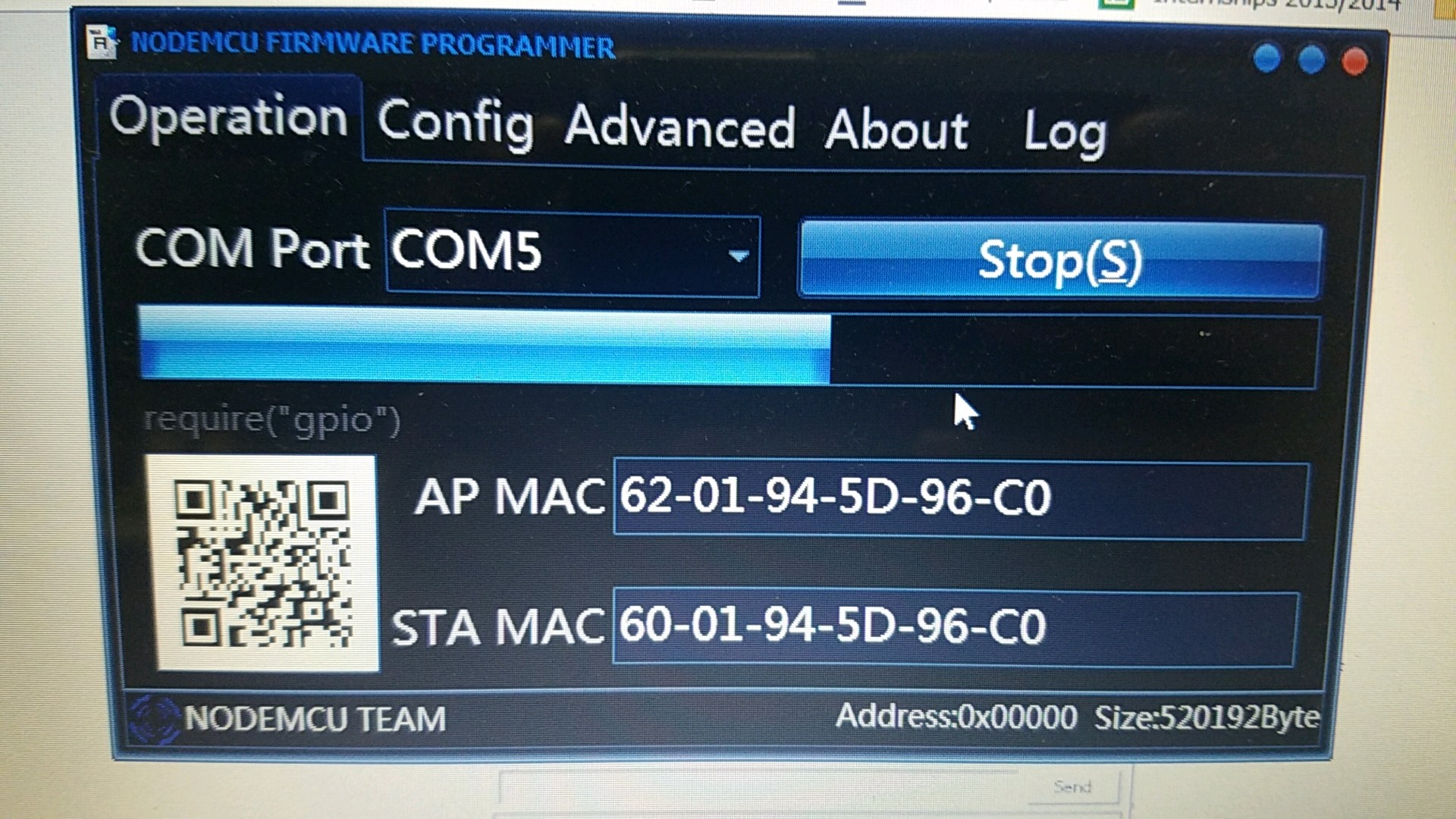
<https://learn.sparkfun.com/tutorials/how-to-install-ftdi-drivers/windows---quick-and-easy>

1. Double check that GPIO0 is connected to ground so that ESP is in programming mode
2. Open the Config tab, select first gear button and choose the firmware specified above. (v0.9.2.2 AT Firmware.bin)



*Figure 5. Configuration Tab*

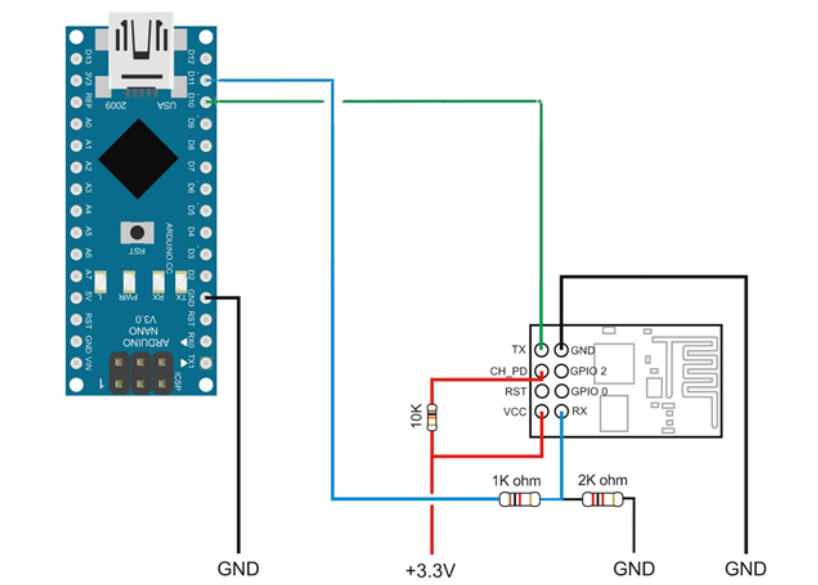
1. Open Operations tab and hit Flash button. Wait until green circle shows.



*Figure 6. Operations Tab*

## Pin Connection

The connection of the various components to the microcontroller was crucial in this project. In order for the Arduino nano to send or receive data to and from the microcontroller, certain connects were to be made. The CH\_PD and VCC ports were powered with 3.3 V which was reduced by the voltage divider, from the 5V line voltage from the power supply. The RST, GP0, GP2 pins on the microcontroller were not used; the GND port was connected directly to ground. The TXD is directly connected to the nano’s GPIO port, (pin 10). Finally, the ESP’s RXD port is connected to the Arduino nano’s GPIO port (pin 11) after going through a 1k ohm resistor as well as being connected to ground after going through a 2k ohm resistor.



*Figure 7 Circuit diagram of Arduino and ESP 8266 WiFi module pin connections.*

* 1. Arduino IDE

After configuring and updating the firmware onto the ESP8266-01, the Arduino Integrated Development Environment (IDE) is used to upload programming to the device. A program that is written using the Arduino IDE is called a sketch. An Arduino program sketch consists of two functions: the setup() function, and the loop() function. The setup function is called after the program is first powered up or when a reset to the microcontroller has occurred. The function is used to initialize variables, set modes for the necessary pins, as well as to include the libraries that are required. This function will only be called once after which all included commands have been executed will it then call the loop function. The loop function has the property of continuously executing the commands it contains until the board is either powered down or is reset.

The code used to program the ESP8266 first requires the inclusion of the Software Serial library which is needed for the Arduino Nano and ESP8266 to communicate with each other. The baud rate is the rate at which the two devices communicate; the baud rate for both devices are set at a rate of 9600. In the setup function, the ESP8266 is reset and is set to client mode. The modes are set using a type of Hayes command set known as AT commands which consist of a parsed string read one character at a time. The AT commands follow a strict set of rules requiring the use of a carriage symbol at its end in addition to being issued serially. Once the AT command has been received successfully, the ESP8266 will respond with a serial message of an affirmative ‘OK’ or a negative ‘not OK’. If an affirmative response is not received, the program should then reset and either try again or cease communications.

In the loop function, the ESP8266 is issued more AT commands that will instruct the module to connect with the available network using the credentials provided to it. The Host, User, Password, and Database Name are passed through communications in order to properly connect to the database that will be storing the given calculations.

## Database

The XAMMP server, an open source web server developed by Apache Friends, was locally run from a PC laptop. The slow server load handling speed based off the PC specifications and database information being inaccessible when the machine was turned off or being only available if user’s device was on the same network would become problematic. Thus, the transfer of a local server to a cloud computing platform where server requests handling and database usage being accessible all the time throughout the internet would need to be implemented.

The Amazon Web Services was tested with the transfer of the MySql/MariaDB database. The AWS had multiple cloud services that had compatibility with the existing XAMMP server and phpMyAdmin administration tool used to build the database. The newly public AWS server was created through Amazon EC2, virtual servers on the cloud instance service, which had to be manually configured into a LAMP(Linux, Apache, MySql, Php) server whereas XAMMP created the server automatically. The Amazon documentation on how to create the LAMP server was straightforward and easy to follow. The configuration settings needed to be setup to allow HTTP requests to grab data from database and security rules to open ports on the public IP address. Once the AWS server was configured, the transfer of files from the XAMMP was needed which was transfer through SSH, a cryptographic network protocol which can remotely send files and login to the AWS server. The SHEMS project would now be able to handle an increased of users load on the server and the server can be accessed anywhere where Amazon Web Services is available.

# IV. Database Connection Code

The microcontroller Arduino Nano was used in correlation with the ESP8266 module. The purpose was to test the connection between the SHEMS circuit and the AWS server with the database. In order to send the power calculations to the MySql database through the use of a webpage containing HTML, PHP, and MySql commands. SQL (Structured Query Language) code is responsible for managing values within the database.

The Arduino code used to connect to the AWS database is as follows:

*#include <SoftwareSerial.h>*

*#include <WiFi.h>*

*//Pins used on Arduino for tranmit(TX) and recieve (RX)*

*#define RX 10*

*#define TX 11*

*String WIFINAME = "WIFI NAME"; //CHANGE*

*String PASS = "PASSWORD"; // CHANGE*

*//Server used to used where data is sent*

*String HOST = "54.164.68.96"; // Host Server*

*String PORT = "80";*

*//Variable used for troubleshooting, counting commands*

*int countTrueCommand;*

*int countTimeCommand;*

*boolean found = false;*

*//Hard coded data used for testing purposes*

*//int power = 500;*

*//double powerfactor = 0.90;*

*String Device = "Device1" ;*

*SoftwareSerial esp8266(RX,TX);*

*void setup() {*

*Serial.begin(9600);*

*esp8266.begin(9600);*

*// Used to make sure data is being understood by ESP properly*

*sendCommand("AT",5,"OK");*

*//Ensures that ESP is in client mode, will respond*

*//with a fail message if already in client mode so don’t be alarmed.*

*sendCommand("AT+CWMODE=1",5,"OK");*

*//Sends credentials (Wifi name and password)*

*sendCommand("AT+CWJAP=\""+ WIFINAME +"\",\""+ PASS +"\"",10,"OK");*

*}*

*void loop() {*

*//Get command for the data that is sent*

*//Make sure there is a space after “Host:” followed by the server address, \r and \n are used for*

*//carriage return, “device”, “power”, “powerfactor” are column names in database*

*String getData = "GET /write\_data.php?device="+Device+"&&power="+String(power)+"&&powerfactor="+String(powerfactor)+" HTTP/1.1\r\nHost: 54.164.68.96\r\n\r\n";*

*//Opens chennels*

*sendCommand("AT+CIPMUX=1",5,"OK");*

*//Starts channel 0 for sending data*

*sendCommand("AT+CIPSTART=0,\"TCP\",\""+ HOST +"\","+ PORT,15,"OK");*

*//Tells that channel ‘0’ will be sent a total of (length of getData) number of characters*

*sendCommand("AT+CIPSEND=0," +String(getData.length()),4,">");*

*//Sends data from getData variable*

*esp8266.println(getData);*

*delay(1500);*

*countTrueCommand++;*

*//Closes channel 0*

*sendCommand("AT+CIPCLOSE=0",5,"OK");*

*//delay(5000);*

*}*

*//Method used for troubleshooting data in Arduino serial monitor*

*void sendCommand(String command, int maxTime, char readReplay[]) {*

*//Prints number of commands sent*

*Serial.print(countTrueCommand);*

*Serial.print(". at command => ");*

*Serial.print(command);*

*Serial.print(" ");*

*//Amount of time given for task*

*while(countTimeCommand < (maxTime\*1))*

*{*

*esp8266.println(command);//at+cipsend*

*if(esp8266.find(readReplay))//ok*

*{*

*found = true;*

*break;*

*}*

*countTimeCommand++;*

*}*

*if(found == true)*

*{*

*Serial.println("Pass");*

*countTrueCommand++;*

*countTimeCommand = 0;*

*}*

*if(found == false)*

*{*

*Serial.println("Fail");*

*countTrueCommand = 0;*

*countTimeCommand = 0;*

*}*

*found = false;*

*}*

The SQL code used to send data to the database is as follows:

Filename: write\_data.php

*<?php  
$con = mysqli\_connect("54.164.68.96","CSUN","User","Password");  
  
// Check connection  
if (mysqli\_connect\_errno())  
 {  
 echo "Failed to connect to MySQL: " . mysqli\_connect\_error();  
 }  
  
mysqli\_query($con,"SELECT \* FROM devicedata ");  
mysqli\_query($con,"DELETE FROM devicedata WHERE device = '".$\_GET["device"]."'");  
mysqli\_query($con,"INSERT INTO devicedata (device,power,powerfactor,OnOff) VALUES ('".$\_GET["device"]."','".$\_GET["power"]."','".$\_GET["powerfactor"]."','".$\_GET["OnOff"]."')");   
?>*

Below is an addition to the code to connect to the database as an add-on to control the device’s power state that determines if the device should be powered on or powered off. However, it should be noted that this code has not been thoroughly tested to ensure that it is completely compatible with the already established code.

*char result [1];*  
*int i =0;*

*int OnOff = 1;*

*"GET /powerstatus.php?device="+Device+" HTTP/1.1\r\nHost: 54.164.68.96\r\n\r\n";*

*sendCommand("AT+CIPMUX=1",5,"OK");*

*sendCommand("AT+CIPSTART=0,\"TCP\",\""+ HOST +"\","+ PORT,15,"OK");*

*sendCommand("AT+CIPSEND=0," +String(getData.length()),4,">");*

*esp8266.println(getData);*

*delay(1500);*

*countTrueCommand++;*

*sendCommand("AT+CIPCLOSE=0",5,"OK");*

*delay(1000); //This delay is really important to read client response  
 i=0;  
 while (client.available())  
 {  
 result[i] = client.read();  
 //Serial.print(result[i]); //Displays client response  
 i++;  
 } //End of while  
 Serial.println();*

*//If database shows a '0' turn the device off. Otherwise turn device on.  
 if (result[i-1]=='0')*

*{  
 //Serial.print("TURN OFF");*

*OnOff = 0;  
 }  
 else  
 {  
 //Serial.print("TURN ON");*

*OnOff = 1;  
 } //End of if*

*} //end of powerstatus  
 delay(1000);*

The SQL code used to receive the data from the database to determine if the device should be powered on or powered off is as follows:

Filename: powerstatus.php

*<?php  
$con = mysqli\_connect("54.164.68.96","CSUN","User","Password");  
  
if (mysqli\_connect\_errno()) {  
 echo "Failed to connect to MySQL: " . mysqli\_connect\_error();  
}  
$result = mysqli\_query($con,"SELECT OnOff FROM devicedata WHERE device = '".$\_GET["device"]."'");  
  
while($row = mysqli\_fetch\_array($result)) {  
 echo $row['OnOff'];  
}  
?>*

# Conclusion

In conclusion, the communication between the Arduino Nano microprocessor and database was left incomplete following the end of the last semester. Currently, the Nano can communicate with a private database including the ability to both write and read data from the said database as well as record the timestamp at which the database was accessed. The values stored in the database can then be updated in real time and displayed in a user-friendly manner.

The database may have moved successfully to a public server rather than a local server however the network security has now become an important factor, safeguarding the user's privacy and information when using SHEMS.

References

[1]Docs.aws.amazon.com. (2018). *Tutorial: Install a LAMP Web Server on Amazon Linux 2 - Amazon Elastic Compute Cloud*. [online] Available at: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-lamp-amazon-linux-2.html [Accessed 25 Aug. 2018].

[2]W3schools.com. (2018). *PHP Select Data From MySQL*. [online] Available at: https://www.w3schools.com/php/php\_mysql\_select.asp [Accessed 15 Oct. 2018].

[3]Android Developers. (2018). *Schedule repeating alarms | Android Developers*. [online] Available at: https://developer.android.com/training/scheduling/alarms [Accessed 10 Sep. 2018].

[4] *Preparation of Papers for IEEE Transactions and Journals*, 2007.

[5]Android Developers. (2018). *Creating a Fragment | Android Developers*. [online] Available at: https://developer.android.com/training/basics/fragments/creating [Accessed 2 May. 2018].

[6] Vishay IL300 Data. (2017) https://www.vishay.com/docs/83708/appnote50.pdf [Accessed 28 September. 2018]