Sall, Djibril (salldl)

salldl@mail.uc.edu

voice/web home monitoring and seCurity system

Final Project for EECE6038 Advanced Microsystem Design

SPRING 2018

Advisor: Purva Bhatnagar

Table of Contents

[A. Problem Statement: 3](#_Toc512636417)

[B. Design Overview: 5](#_Toc512636418)

[I. Hardware: 5](#_Toc512636419)

[1) The Mikromedia Board: 5](#_Toc512636420)

[2) The ESP8266: 6](#_Toc512636421)

[3) Sensors and Actuators: 6](#_Toc512636422)

[4) Ubuntu-server: 7](#_Toc512636423)

[II. Software: 7](#_Toc512636424)

[1) Mplab X IDE: 8](#_Toc512636425)

[2) Arduino IDE: 8](#_Toc512636426)

[3) Ubuntu Server on AWS: 8](#_Toc512636427)

[4) Node Red: 8](#_Toc512636428)

[5) Mosquitto MQTT: 8](#_Toc512636429)

[https://photos.app.goo.gl/cXo4PpHQFjHh4SnZ7 9](#_Toc512636430)

[C. Design Implementation: 10](#_Toc512636431)

[I. Webserver: 11](#_Toc512636432)

[1. Mosquitto Server: 11](#_Toc512636433)

[2. Node-Red: 12](#_Toc512636434)

[a) Dashboard: 12](#_Toc512636435)

[b) Redbot: 14](#_Toc512636436)

[c) MQTT and Adafruit.IO: 14](#_Toc512636437)

[II. PIC24 and ESP8266 Firmware: 15](#_Toc512636438)

[D. ESP8266: 15](#_Toc512636439)

[1. Initialization: 15](#_Toc512636440)

[2. MQTT Message Received: 16](#_Toc512636441)

[3. MQTT Sending Message: 16](#_Toc512636442)

[D. PIC24EP Firmware: 16](#_Toc512636443)

[1. UART and Overall System: 17](#_Toc512636444)

[2. User Interface and Touch Inputs: 19](#_Toc512636445)

[D. Results and Testing 22](#_Toc512636446)

[D. Conclusion: 23](#_Toc512636447)

# Problem Statement:

Throughout the past years, we have seen a huge rise of iot devices, from connected cars, healthcare devices, industrial control systems etc… One of the field we have seen a huge grow in is home connected devices and monitoring systems. People like these systems as they make it easy to control devices from anywhere in the world as well as having control over their air conditioning or heating systems for convenience and energy saving.

But the biggest problem for this industry is that most of these devices are expensive, and there is not a system to control all of them easily from one interface or system.

For example, if one person wanted to change their regular thermostat into a smarter one like a Nest thermostat, they will have to spend $250 just to acquire the nest thermostat. Then getting an amazon echo or any other system as well as additional WIFI connected plugs can bring the cost of the system to at least $300*. (figure 1)*

The other issue that comes with this is privacy. Most of these systems requires having all the data and information handled by the manufacturers servers. For example, when someone uses a Google Home Assistant or Alexa Echo most of their voice requests are stored in google servers and requests and controls are issued from those servers. The issue with this approach is that the consumer doesn’t know what is recorded or which data are kept by the manufacturer, but also, they don’t have any power over their data as the company can use it anyway they want to. Recently Google launched a smaller product the Home mini, and costumers discovered that the device was continuously listening and uploading all the contents in Google’s servers. Luckily the company pushed an update to stop the device from recording and removing the wake word functionality that was causing that definitely.

Once we come around the price and privacy issue, the other problem is that all the systems that integrate those different devices are complex or hard to use for the average person.

For example, when you buy an amazon Echo, you have to download a separate app that works with that, then you need to download a separate app for your Nest Thermostat and another one for your smart plugs. In order to integrate all these separate devices together, you will need strong programming skills to build your own costume solution or other systems that are not user friendly or requires some technical skills to accomplish.

All these different problems make it imperative to find a new cheaper, more secure and user-friendly system. That’s how the idea of this project came to life: building a cheap and easy to use system accessible to everyone and that’s also secure.

The scope of this project was to primarily use all the different resources available in the PIC microcontroller to show that any actuator or sensors can be integrated with the system.

Also use different modes of interacting with the system using voice (Google Assistant), a smartwatch, text(Chatbot) and a web interface on a website accessible from anywhere.

The system has to be controllable using the different modes of communication listed above but also send feedbacks via text and on the website.

The implementation of this project was simple given that the PIC24EP512GU810 Mikromedia board used in this class had all the different modules we would need for it, but also throughout the semester, we had labs that were directly connected to every feature required in this project.

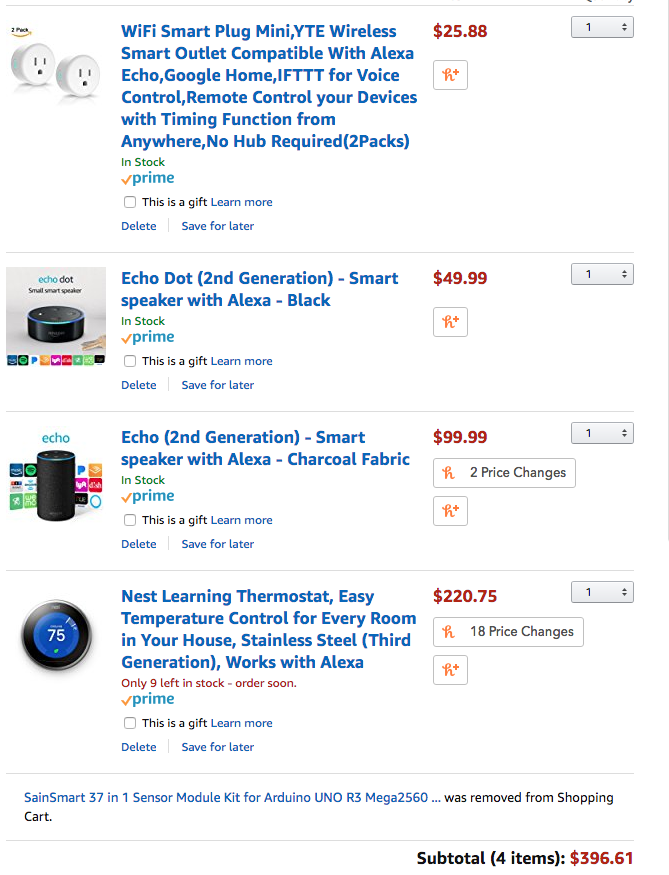


Figure Estimated price for smart thermostat,

two smart plugs and voice assistant on amazon.

# Design Overview:

The design of this system required combining both hardware and software systems running on them to coordinate simultaneously in order to work in accordance. The two major groups hardware and software included different subsystems that were coordinating to achieve the end goal.

## Hardware:

For the hardware part we have:

1. The Mikromedia Board: containing the PIC24EP512GU810 microcontroller from microchip which also included a touchscreen LCD panel as well as different GPIO’s that support different communication protocols (Analogue to Digital, UART, I2C etc). *(fig. 2)*

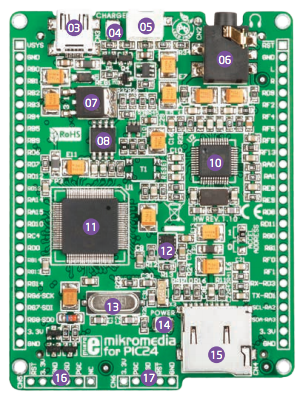
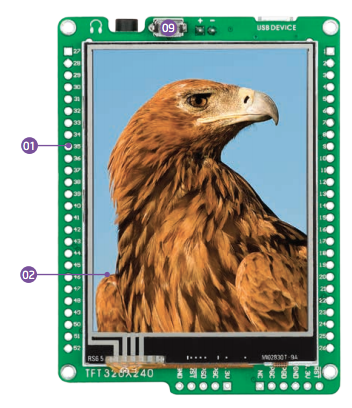
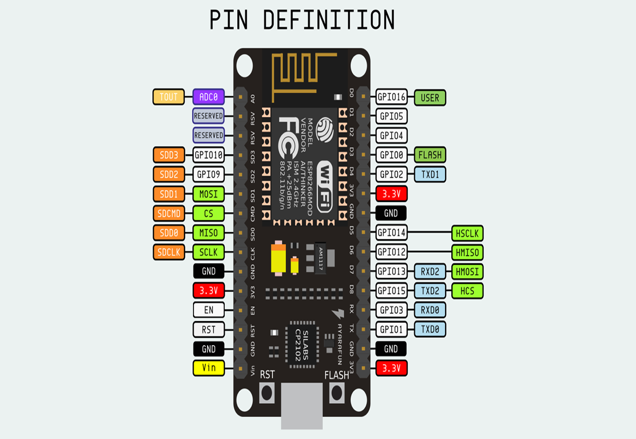
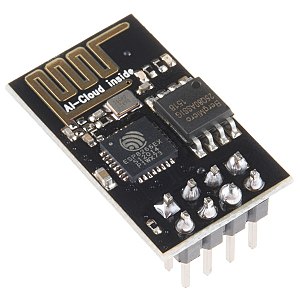


Figure Mikromedia Pic Board

1. The ESP8266: A WIFI MCU that supports MQTT protocol and that also contains UART for programming purposes and also to communicate with other devices.



Figure

1. Sensors and Actuators: The sensors and actuators could be used to get data from the environment or act on it.

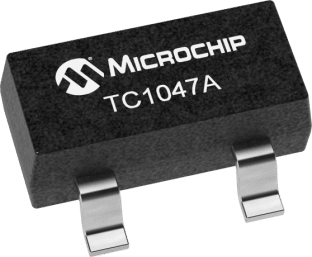
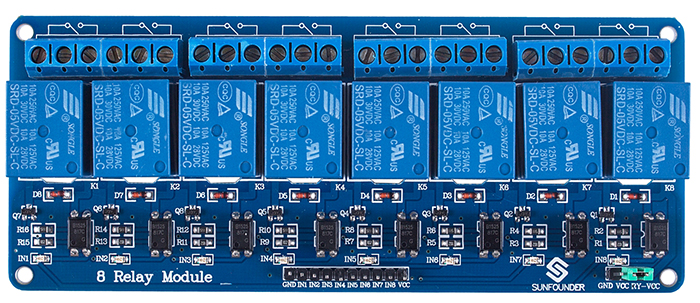
 

Figure Analog Temperature Sensor Figure Servo Motor

Figure 8 Channel Relay



1. Ubuntu-server: We also had an instance running on an amazon EC2 instance to act as our main server, but this could have been easily replaced with a Raspberry Pi Zero W or any SOC that can run Linux.



Figure Amazon EC2

## Software:

1. Mplab X IDE: Mplab is used to program the microchip microcontroller with the firmware written in C to control all the actuators with commands coming from the webserver, but also get the data from the sensors and transfer it to the web-server also to provide the person with a user interface.
2. Arduino IDE: The Arduino IDE was used to write a simple firmware for the ESP8266 on the NodeMCU in order to get messages from the server and transfer it via UART to the PIC since the PIC didn’t WIFI on it.
3. Ubuntu Server on AWS: the Ubuntu Server contain several servers that runs for the connection between the different interaction modes and the PIC: Node Red and Mosquitto.
4. Node Red: Node Red is a graphical programming interface based on Javascript and that has nodes for different communication protocols and other components to make Web UI’s. It’s owned by IBM and it helps make simple web UI’s for automation projects
5. Mosquitto MQTT: it’s a light weight server that help send messages via TCP or any other networking transport available. It’s organized as follow:
   * + you have a broker: which is the Ubuntu server in this instance that runs the mosquito server.
     + Clients: devices that can connect to the server to publish or subscribe to a topic.

The client can publish an information to the server and on the other hand, if another client is subscribed to that topic and have the right to access it, the broker transmits the information to the clients subscribed to that topic.

The PIC board can get data from the sensors using ADC or any GPIO process it, and send the information to the ESP8266 via UART.

The UART just relays that message using the internet connection provided by the WIFI connection to the MQTT server running on the amazon instance.

The Node Red server running on the server get the MQTT message and reflects the required action on the web UI or via a text to the Telegram Chatbot.

The second communication way is when we send a command using Google Assistant, a smartwatch, Chatbot or the website. This information is processed by the Node Red server, then sent to the ESP8266 via the MQTT server, which relays the message to the PIC using UART. The PIC board then processes the message and act on the actuators or give a feedback on the LCD screen on the board.

This is a block diagram showing the design of the system:

The way the overall system is designed can be resumed in the following blog diagram:



This is a short video explaining how the devices are connected:

# <https://photos.app.goo.gl/cXo4PpHQFjHh4SnZ7>

# Design Implementation:

When it comes to hardware, there was no design involve as most of the required features were embedded in the board and embedded shield. The only thing that we had to do was connect the different hardware together using jumper wires. No soldering or PCB design was done due to time constraints.

Different diagram about how the different components on the board are on the Mikromedia’s website and here is a [link.](https://download.mikroe.com/documents/smart-displays/mikromedia/3/pic24ep/mikromedia-pic24ep-manual-v100b.pdf.)

We have the servo motor with the PWM input on it connected to the RF8 Pin of the PIC24EP512GU810. The servo motor is the SG90 micro servo, it’s a small servo that runs on 5V and this is a link to its [datasheet](http://akizukidenshi.com/download/ds/towerpro/SG90.pdf).

The temperature sensor [TC4017A](http://ww1.microchip.com/downloads/en/DeviceDoc/21498D.pdf) an analogue temperature sensor is connected to RB9 through ADC.

I also have an 8 Channel [relay](http://wiki.sunfounder.cc/index.php?title=8_Channel_5V_Relay_Module) connected to different pots in the PIC.

Here is a table with the different pins:

|  |  |  |
| --- | --- | --- |
| Device | Pin Connected | Type of GPIO |
| Relay 1-Routeur | F1 | Digital |
| Relay 2-Living\_Room | F0 | Digital |
| Relay 3-Kitchen | G1 | Digital |
| Relay 4-Room1 | G0 | Digital |
| Relay 5-Room2 | D11 | Digital |
| Relay 6-Fridge | D9 | Digital |
| TC4017A | B9 | Analogue |
| Servo SG9 | F8 | PWM |
| UART TX ESP8266 | RD3 | UART RX |
| UART RX ESP8266 | RD1 | UART TX |
| Switch 1 | RD13 | Digital |

A detailed diagram showing where the different LCD and TFT connections are linked to the Mikromedia board can be found in this [link](https://download.mikroe.com/documents/smart-displays/mikromedia/3/pic24ep/mikromedia-pic24ep-manual-v100b.pdf.). There you will find the type of LCD screen used as well as different details of the driver used for the board, which in this case is the [ILI9341](https://cdn-shop.adafruit.com/datasheets/ILI9341.pdf) LCD driver.

The majority of the implementation of this project was mainly software based and it can be divided in two major parts: the firmware on the PIC24EP512 and the webserver and how it connects to the other devices.

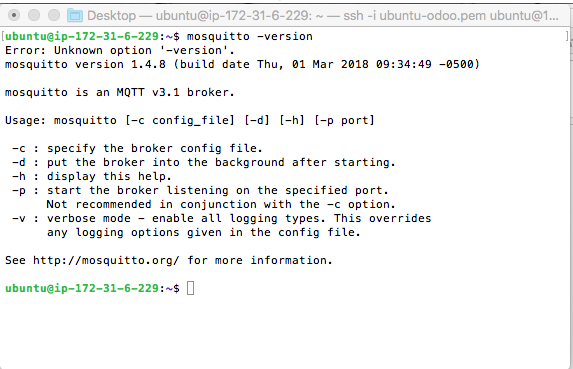
We will first start with the webserver and how it interacts with any other phone or service.

## Webserver:

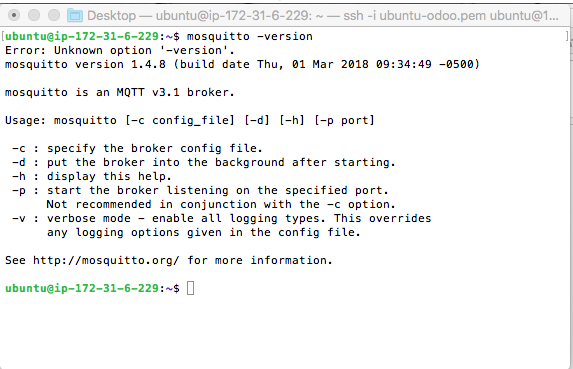
The webserver can be divided in 3 parts the Node-red server which is the main components that links the peripherals and the PIC24EP which is the main control unit, the mosquito server which acts as the broker and IFTT to get messages from the phone or smartwatch and transfer it via MQTT to the node-red server.

### Mosquitto Server:

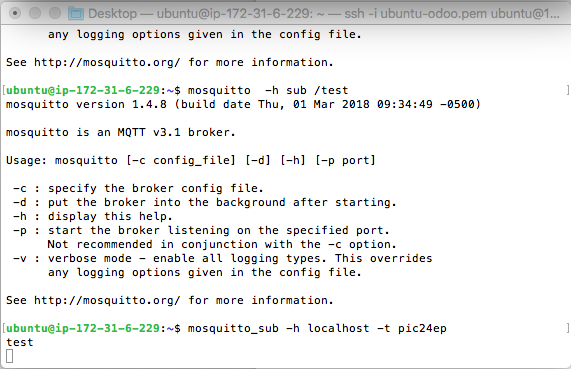
The mosquito server was the first sever I installed in the Ubuntu server, it’s only 532 Kb which makes it extremely light weight for the job it performs. More information about Mosquitto, the installation process and how it works can be found on their [website](https://mosquitto.org/).

Once installed you can check if it’s working and test it by publishing and subscribing: 

Figure



Figure

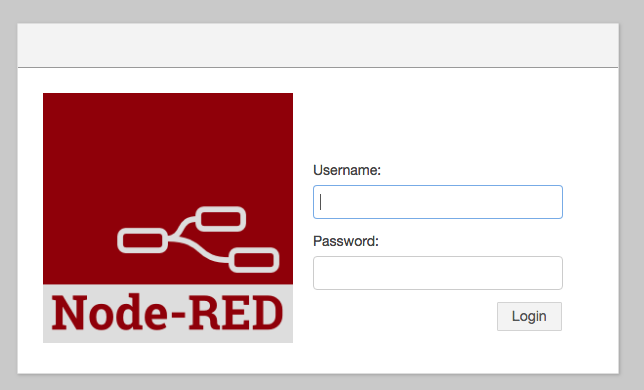


Figure

## Node-Red:

The node red server, is a server based on Javascritpt created using Nodejs. It provides you with a graphical interface for programming which has access to thousands of API for different communication protocols. Multiple documents can be found online on how to install Node-red in any given computer.

I also added security to it so that a username and password are asked every time someone tries to connect to then server.



Figure

Once we have node-red up and running we need several to install several Nodes that are required for this project:

1. Dashboard:

Which helps create the web user interface that can be accessed from anywhere using the website and the credentials of the node-red client.

These are the different nodes for the dashboard for creating diagrams, buttons, graphs and all the different UI needed for the website.

Here is a screenshot of the nodes in node-red and some of the Javascript code to change the UI.

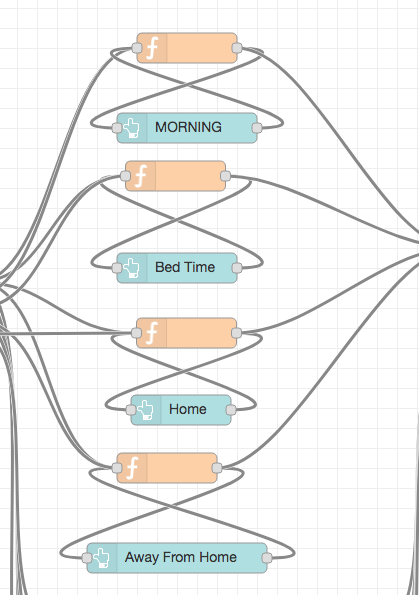
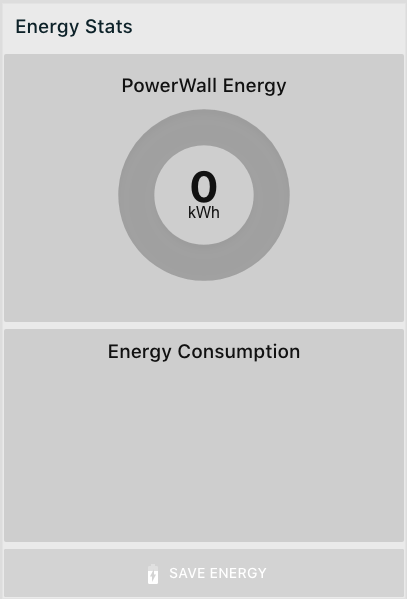
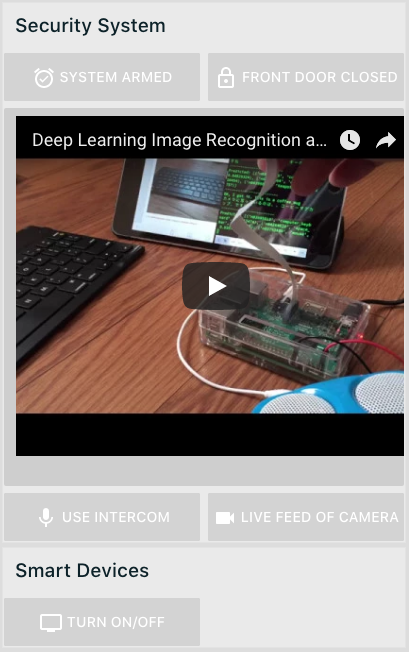
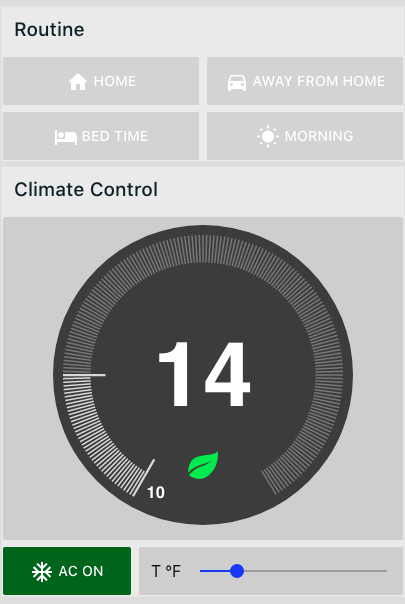


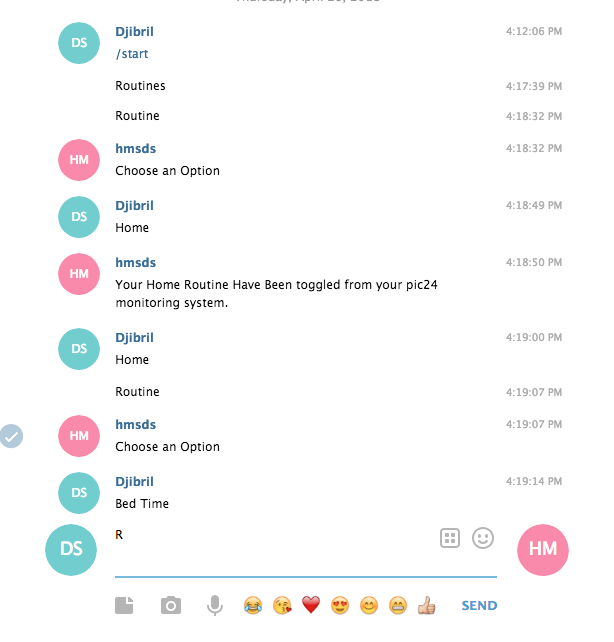
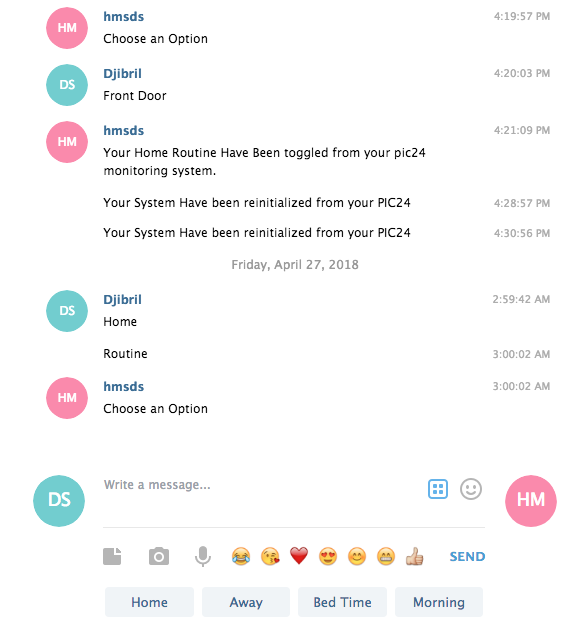
Figure Figure



Figure

### Redbot:

This node helps us create a chatbot on telegram that we can use in order to send messages to the node server that can be relayed via MQTT to the PIC24EP.



Figure

All the information needed can be found in this [link](https://docs.google.com/document/d/15Wf4tTncwQVpS-baSAgFcQ5rR4oeZtpqb5mAYAjjzNc/edit?usp=sharing) that contain my node-red flow and it can be easily copied to any node-red server.

### MQTT and Adafruit.IO:

The MQTT component and the Adafruit.io is what we use to get information from the phone when data is sent from our phone when we use the Google Assistant or IFTT on the smartwatch it sends a string to the Node-red Server using MQTT.

I set-it up for different options for instance when you say “Open the front Door”, IFTT sends to the Node-red server a string of 4 characters that the node-red server will then transmit to the PIC board.

On the other hand, the content of our speech can also be transmitted to the pic. When I say “Ok Google set the Thermostat to #value” the value is then sent to the server.

I will include links to videos showing the different use cases in the result and testing section.

I could have given a detailed explanation of how everything is implemented on the server side, but it would just take tens of pages, but there is thousands of resources online showing how to implement this kind of solution.

## PIC24 and ESP8266 Firmware:

We have too firmware one for the PIC24 and another one for the ESP8266 to connect the PIC to our webserver.

## ESP8266:

For the ESP8266, I used the Arduino IDE in order to write a simple program that will connect the ESP8266 to our webserver on Amazon AWS. The ESP8266 will only perform too simple tasks, subscribe to certain topics on the Mosquitto server.

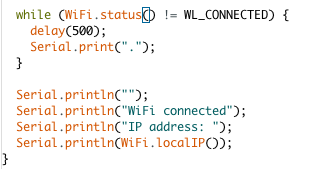
Every time a message is published on the Mosquitto server from the Node-Red Server when someone pushes a button on the website or give a voice or text command from their phone; The ESP8266 will automatically get that message and send it via UART to the PIC.

On the Other hand, whenever the ESP8266 receives a message from the PIC, it will automatically publish it on the MQTT server and the Node-red server can use it to display the information on the website or send it via text to the chatbot.

We can divide the program into three parts:

### Initialization:

This part is just done to connect the ESP8266 to the WIFI network, as well as set-up the connection to the mosquito server and UART pins and Baud Rate for communicating with the PIC.



### MQTT Message Received:

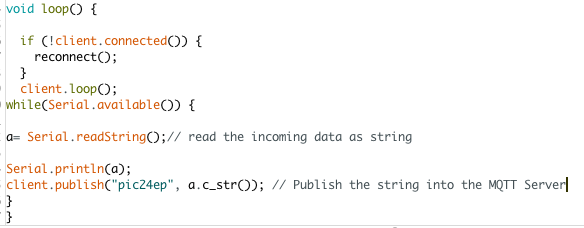
When a message is received, it comes per character which we store into a string variable a that we send via UART to the PIC24.

It’s done via the reconnect() function that runs in the main loop and checks continuously if a new message is published in the MQTT server.



### MQTT Sending Message:

Lastly, we have in the main loop a function that will store every time a character is received via the UART from the pic into a string. The string is then published into the MQTT server on the webserver so that the Node-Red server can access it.



## PIC24EP Firmware:

The firmware on the PIC24 help use as much as possible of the different modules that came with this board. We have parts of the firmware that help get the message from our webserver and analyze it in order to act on the actuators. The second part provide a visual feedback as well as a user interface that the user can use in order to interact with the system.

We will divide this into two sections, the first one being receiving the messages via UART and analyzing them in order to act on them. The second part will discuss how the user interface was designed and how users can use the device to interact with the system.

### UART and Overall System:

The overall system works like the following.

We have different routines that can be set and based on those routines different devices will be turned OFF and others turned OFF.

We had 4 Routines: Home, Bed Time, Morning and Away. The Home Routine turns on all the devices in this case the relays which represents different devices or rooms.

There is also another section that we can use to set our home security system, open the front door or close it.

This part of the program project required using several modules in the PIC and organizing them in a way, so that they won’t be any interference with their functionalities.

The most important feature of this project was to get the information from and to the server and his was done using the UART module on the PIC. I decided to send strings of 4 characters to the PIC for each different task that need to be performed by the PIC. For example, if we won’t to turn on a given routine or open or close the door(servo).

When someone selects a routine on the website or it’s automatically triggered using geolocation of your phone, the Node Red server sends a string of 4 characters to the PIC.

When someone presses HOME on the website, the Node-Red Server send “HMON” to the PIC.

I have set-up the UART Interrupt on the PIC and gave it the highest priority, so that whenever data is received via UART the program runs to the interrupt subroutine to get the data and analyze it.

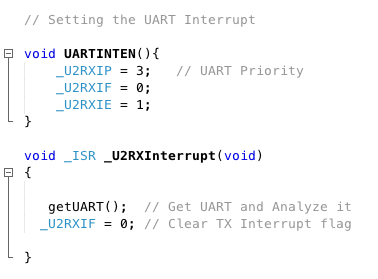
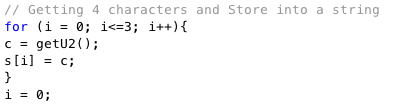
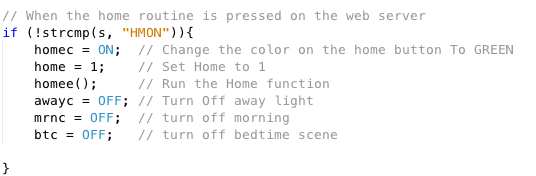


Figure UART Interrupt Setup

In the Get UART function, we have a for loop that get the 4 characters and store them into a string s.

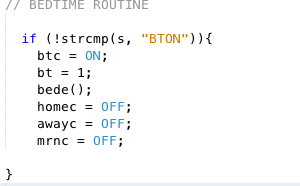


Once the string is stored in s, we compare the string with the different options we have using the stcmp command in c. If the content of s matches one of the different options like “HMON” it can change some colors on the user interface and or run a function that correspond to that command sent from the webserver.



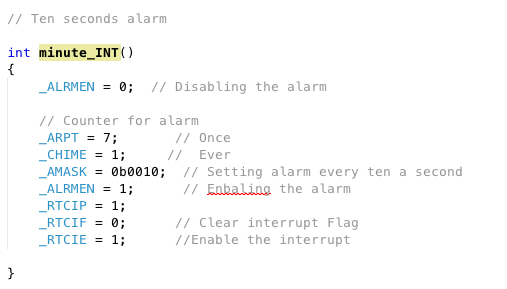
Figure

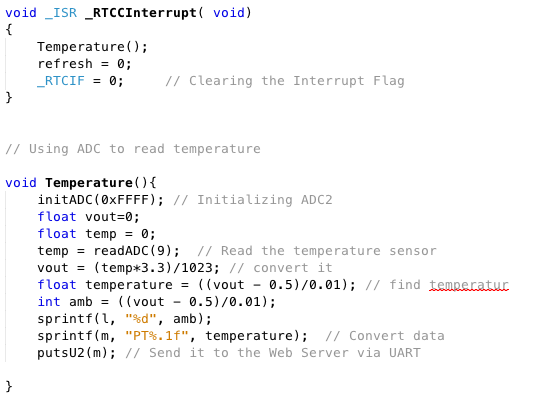
For instance, if the PIC receives another message, like “BTON” it will turn the bedtime mode On.



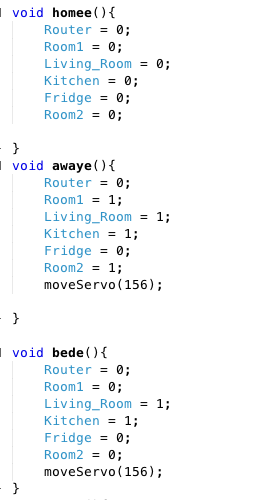
I also used the RTC alarm in order to read the temperature every 10 seconds.

Every ten second, the value of the temperature sensor is read and then after conversion and calculation it’s sent via UART to the web server which displays it on the website.





Each routine or scenario is defined on a function to set different conditions for the actuators.

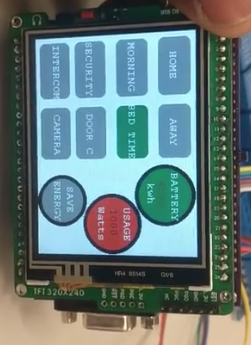
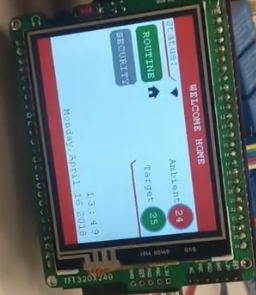


Figure

The code is accessible at this [link](https://drive.google.com/drive/folders/1v_iyOmxbfIOBzsJj8jSsTePv59lgBMXj?usp=sharing).

### User Interface and Touch Inputs:

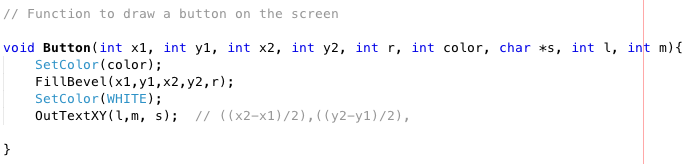
For the Graphical interface, I drew buttons using the bevel function and other LCD functions.

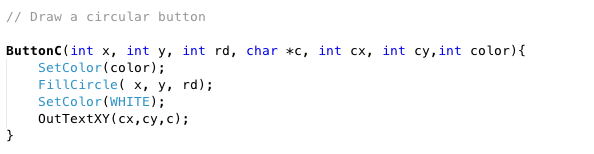


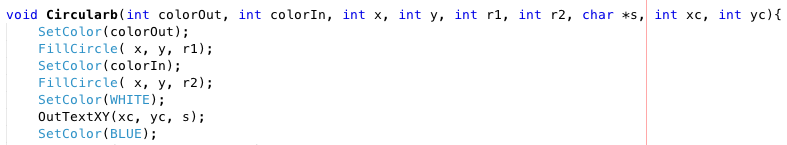


Figure

These are snipsets of the functions that I made to easily draw the buttons:







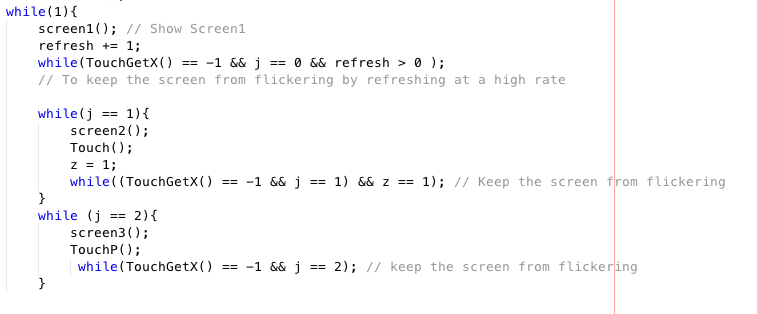
Figure

Once I got all the screens ready I used the external interrupt through which I can use the switch or a photo sensor connected to it. When one waives their hand in front of the photo sensor it makes the device go to the next screen. A video showcasing this feature working will be included in the result section.

Every time the switch is pressed or someone waves in front of the sensor the interrupt is triggered and we have the next screen shown in the LCD by incrementing the value of j.



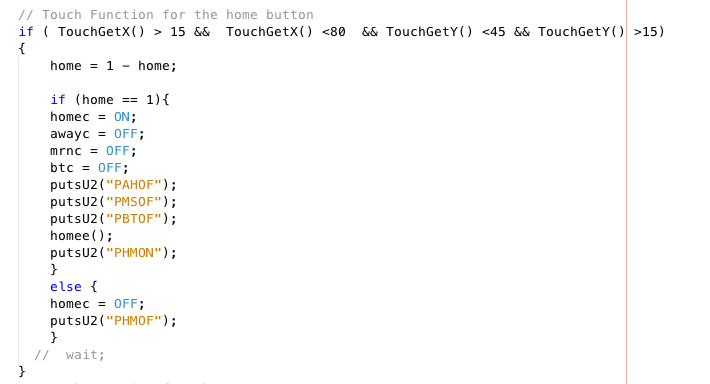
In the main function, we display a certain screen depending on the value of j.



For the Touch Screen, I used the TouchGetX and TouchGetY functions with the borders of the buttons in order to trigger actions.

This is an example for the Home button:





I replicated the same for the additional buttons in each screen.

# Results and Testing

Overall the system worked as expected, we were able to send commands from the watch, text and using the website. The website was also getting information from the PIC24 and showing it on the website. Here are several videos showing different test cases and the result we got.

List of test cases:

Web Interface:

<https://photos.app.goo.gl/daoiBJQeYgt6PnEr8>

-Routine and Security:

<https://photos.app.goo.gl/TbCJvrPXzbj2iVN12>

<https://photos.app.goo.gl/yW96nFCpk8lJiaaf1>

-Energy Stat:

<https://photos.app.goo.gl/o4EyCAJR1hbwYLmo6>

-Temperature:

<https://photos.app.goo.gl/daoiBJQeYgt6PnEr8>

Interacting with the system from the PIC touch Screen:

<https://photos.app.goo.gl/E4OYZFkTM648wZI43>

When we sent a command using voice, the system reacted to it accordingly.

<https://photos.app.goo.gl/ojobMvY5MUzjMUci8>

When we also used the watch to do something it reacted to it.

<https://photos.app.goo.gl/RlsnXdB8FHzjVyGh1>

<https://photos.app.goo.gl/hSJC9psyHGus5aS3A>

The chatbot was also able to control the devices from telegram.

<https://photos.app.goo.gl/bxQcGFRfbuPshQvY9>

This video shows the ability to change the screens using hand gestures.

<https://photos.app.goo.gl/JzdiPUaxWMejCREn9>

Link to all codes for this project:

<https://drive.google.com/drive/folders/1v_iyOmxbfIOBzsJj8jSsTePv59lgBMXj?usp=sharing>

Cost Analysis:

Price of the components used:

|  |  |
| --- | --- |
| Device | Price |
| PIC24 Board with PIC kit | $140.00 |
| Relays | $8.00 |
| Servo Motor | $4.00 |
| Jumper Wires | $8.00 |
| Embedded Shield | $45.00 |
| Node MCU ESP8266 | $8.00 |
| Total | $213.00 |

Potential Cost for Commercialization (This price is only for the components that will be needed no labor included):

|  |  |
| --- | --- |
| Device | Price |
| SOC running Linux | $10.00 |
| Electronic Components | $8.00 |
| Temperature Sensor | $2.00 |
| Casing and Manufacturing | $5.00 |
| Relay with WIFI | $5.00 |
| Total | $30.00 |

# Conclusion:

Overall the project was successful as it helped us explore all the potential of this board and use everything we learned during this semester on this class. It also helped us learn more about project development and how many unexpected issues can occur and how to deal with them.

I also had a chance to explore other domains that I wasn’t familiar with specially the software side. I think the biggest problem I faced doing this project was time constraint and not having a teammate. Having a teammate would have had relived some of the weight from my shoulders and I could have concentrated most of my time on integrating more sensors and actuators to have a fully home automation and security systems. Also trying to add new features in the middle of the semester made it harder to finish on time. But overall it was a good experience and I learned a lot about firmware development as well as product development as a whole.

Ressources:

[www.mosquitto.org](http://www.mosquitto.org)

[www.node-red.org](http://www.node-red.org)

[www.amazon.com](http://www.amazon.com)

[www.wiki.org](http://www.wiki.org)

[www.microchip.com](http://www.microchip.com)

[Figure 1 Estimated price for smart thermostat, 5](#_Toc512636448)

[Figure 2 Mikromedia Pic Board 6](#_Toc512636449)

[Figure 3 6](#_Toc512636450)

[Figure 4 Analog Temperature Sensor Figure 5 Servo Motor 7](#_Toc512636451)

[Figure 6 8 Channel Relay 7](#_Toc512636452)

[Figure 7 Amazon EC2 7](#_Toc512636453)

[Figure 8 11](#_Toc512636454)

[Figure 9 11](#_Toc512636455)

[Figure 10 12](#_Toc512636456)

[Figure 11 12](#_Toc512636457)

[Figure 12 Figure 13 13](#_Toc512636458)

[Figure 14 13](#_Toc512636459)

[Figure 15 14](#_Toc512636460)

[Figure 16 UART Interrupt Setup 17](#_Toc512636461)

[Figure 17 18](#_Toc512636462)

[Figure 18 19](#_Toc512636463)

[Figure 19 20](#_Toc512636464)

[Figure 20 20](#_Toc512636465)