

Home Safety and Automation System

Matthew Allen, Guilherme Costa, Felix Henriquez, Avery Stevenson

Dept. of Electrical and Computer Engineering, University of Central Florida, Orlando, Florida, 32816-2450

Abstract — As technologies advance, our world is becoming increasingly more efficient. We are in the era of hands-free multitasking and automation. Our objective for this project is to create a Home Automation System that allows consumers to control various appliances in their home wirelessly, manage the energy consumption of these appliances through either a smart phone application or web application, and provide an extra safety net of sensors and surveillance devices. This will be accomplished with a smart outlet, smart wall switch, a thermostat for the HVAC System, a motion sensor, and a fire hazard detector. This paper will explain our motivations as well as the components we have chosen and the methodology behind our software programming.

I. INTRODUCTION

Today, most people want their day-to-day interactions to occur in a fast, simple and convenient manner. From driving to work without having to step on the accelerator pedal to coming home and having their bedroom lights turning on as we step into the room. The aim of this project will be to take care of this last stage. We prefer our users not to worry about manually turning their lights on, remembering from turning the stove off, and forgetting to change the temperature of their air conditioning system. With our Home Safety and Automation System (HSAS) it will be all integrated for easy control and management of most of the devices in a home. This system is meant to provide the end-user with a one-stop interface where they will be able to control, monitor, and manage their home lights, outlets, and air conditioning system from a simple and easy to use interface, from anywhere they like. We hope to provide a home automation system that is convenient for the average user to use and requires minimal installation and maintenance.

Infrared motion detectors will be able to detect when someone enters or leaves a room, or if someone is at the front door. This functionality will be part of the core system, as it will help determine if the lights of a specific room should be on or off, or if the stove should be on when there is no one at home, for example. Near Infrared photodiodes will also be used to detect the temperature in a specific room. Its main purpose will be to alert the homeowner of a possible fire in their premise. Since this

sensor permits a much faster response time than common smoke detectors, it will allow the user to react in a timely manner. The system will also facilitate the control of outlets spread across a house. With our system acting as a middleman for the device and the power grid of the user's electrical system, they will be able to control when a specific outlet should receive power and for how long. This will improve power usage and help reduce the overall power efficiency of a household. Our smart switch will also help us with this purpose. The HVAC System is one of the more costly appliances in a home in terms of energy consumption. To manage this system we will use our smart thermostat to control the HVAC System's relay control board.

To integrate the features described previously, the system will also provide a simple user interface that can be accessed from anywhere through a web browser where the user will be able to control and manage all the sensors and devices connected to the system. The interface, will also allow the user to monitor the power usage of the devices, therefore reducing the power cost of the system and the household.

II. GOALS AND OBJECTIVES

One goal of this project is to make the daily life of our end-user easier. We want our users, to be able to control most of the devices in their household and to receive alerts about them in an efficient and simple manner. Another important goal to build a low power system. In this way, we are able to provide the customer with a low-cost system and at the same time we can monitor the household power usage to further reduce energy costs. We also tried to minimize component costs as much as possible. One hurdle we see with current home automation systems is that their price points are still out of reach for low income consumers. An unfortunate pattern with new technologies is that wealthy consumers benefit first and low income consumers are left out of the loop for quite a while. Our goal is to help bridge this divide with our low cost solutions.

In order to convey this information in a comfortable way, we need to develop a user interface that is friendly and not cluttered. This interface needs to be simple and the user should be presented with all the information and it shouldn't be cumbersome to get it. We made the interface system portable because we want users to be able to access it from anywhere they desire. With this feature, we are ultimately able to reduce the overall cost and provide the user with more flexibility to manage the system. Since we have opted to a modular system, this modularity also needs to convey simplicity. We do not want our end-user to have a cumbersome installation of the system. Our objective is to make the installation as simple as possible, where a user with no electronic expertise can perform it easily.

III. HARDWARE – COMPONENTS

The system is comprised of 5 peripheral devices: The Thermostat, The Outlet, The Switch, a Motion Sensor, and a Fire Hazard Detector. Our goal for the devices was to be able to make them cheap and reliable. To achieve this, we set out to use cheap, readily available, extensively tested components. By ensuring that the components were massively produced parts we decreased our liability on any special order component malfunctioning and the gained the ability to have numerous spares for quick replacement in case any had defects

A. MCU ESP-12f

The peripheral devices are managed by an ESP-12F MCU, a model of the esp8266 by Expressif Systems. We chose this model for its cost, the available gpio pins, and the vast number of available software libraries.

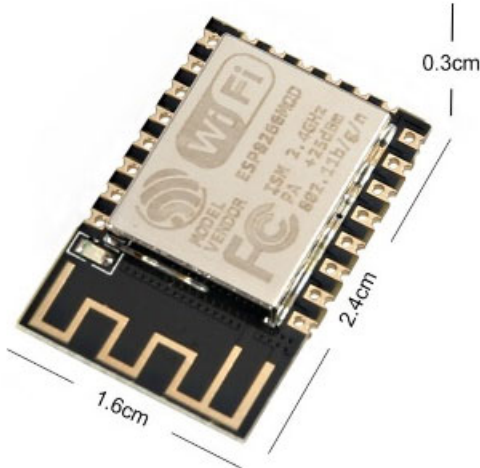


Fig. 1. ESP-12F MCU

The ESPs are the smallest 802.11b/g/n Wi-Fi SOC modules available. They are available with 16 and 32bit processors with up to 160MHz clock speed. They include a 10bit ADC and have UART/IIC/SPI interfaces, sleep modes, and this particular model comes with 4MB flash memory.

B. Power Relays

We used 5 pin magnetic power relays as our workhorses in the switching circuits. We chose mechanical relays or solid state because the currents our devices can see at the output terminals can be a lot greater than what the compact SSRs can handle. We also did not require very fast switching. For the Thermostat we used an array of 4 3V/10A Relays and 12V/20A relays in the Outlet and Switch.

C. DHT22 Temperature Sensor

A vital component to the HVAC system the DHT22 uses a Polymer Capacitor Sensor to measure ambient air temperature and humidity. This is a low power sensor that only uses 1.5mA when measuring. It uses a 3.3V supply. The sensor communicates via a simplex data line that is active low. To receive data the MCU pulls the line to ground for 1us and waits for up to 40us for the sensor to send its readings. The DHT22 is rated to be 95% accurate in humidity and ± 0.5 C temperature error rate. For home applications we found this accuracy to be sufficient.

D. ACS712 Current Sensor

The ACS712 is a Hall Effect current sensor capable of measuring AC and DC currents up to 30A. The ACS712 operates using 5VCC at 10mA making it a very low power sensor. The sensor uses an analog voltage output to describe how much current is flowing through the current terminals. We will be using the following formula to find how much power the Outlet and Switch is consuming.

$$P = 120RMS * (SensorVoltage)/(66mA/V)$$

The 66mA/V is the conversion ration based on the sensor's datasheet.

E. Power Supplies

The Thermostat, Outlet, and Switch use plugged-in power wires. The Motion and Heat sensors are battery operated. The thermostat is supplied by the HVAC relay system 24VAC why the Switch and Outlet are supplied by the home's 120VAC. The thermostat uses a full wave bridge rectifier circuit to an LM2596T voltage regulator to supply the 3.3VCC. The Outlet and Switch use an HLK-PM12 to convert the 120VAC to 12VDC and from that a pair of LM2596Ts for 3.3V and 5V supplies for the MCU and sensor component.

F. OLED 1.8" IIC Display

To display current temperature readings and settings in the Thermostat we have a small IIC display from Adafruit. The display uses the 3.3VCC and is install as a separate module with an array of button inputs.

IV. FIRE HAZARD DETECTION UNIT

The fire hazard detection unit we designed consists of a near-infrared photodiode, a lens, and a casing. It can detect fire as well as materials over 250 degrees Fahrenheit and it has a range of up to 2 feet for a typical lighter flame. This fire hazard detection unit is designed to be placed over a stove to detect fire as well as potential fire hazards such as a burner which has been left on after cooking.

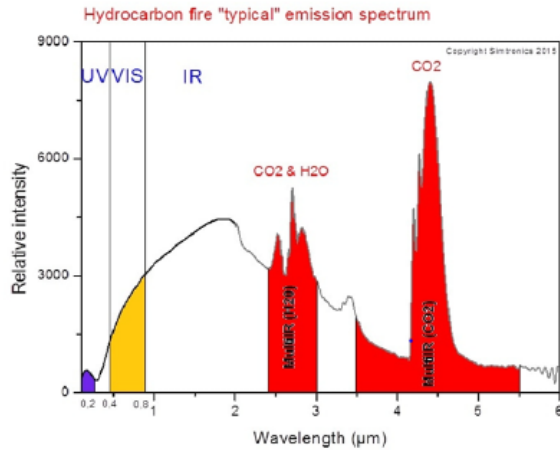


Fig. 2. Emission Spectrum

The near-infrared photodiode is made of silicon and it has a detection spectrum between 0.7-1.1μm. Figure 2 shows the blackbody spectrum of a typical hydrocarbon fire and we can see there is emission within the sensor's detection spectrum. The sensor's detection spectrum is also useful for stopping false detections because there are only a few other sources which emit in this spectrum naturally, and actions such as pointing the fire hazard detection unit downward and away from the sun or lighting can easily mitigate false detections. The detection angle is approximately 16 degrees so most of a stovetop is within the area of detection if it's placed 2 feet above.

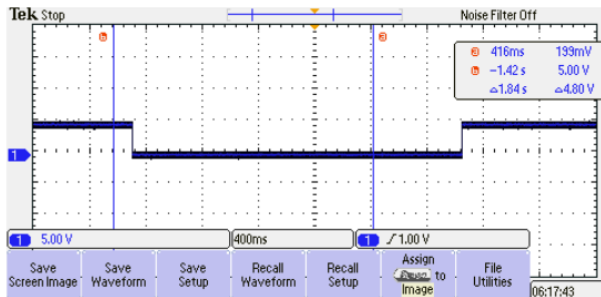


Fig. 3. DC Output Signal

The near-infrared photodiode has a 5V input voltage and has options for both an AC output and a DC output. We chose to use the DC output because it was much more consistent and allowed us to take a simpler route for programming the detection software. Figure 3 shows the DC output signal of the photodiode and we see that there is a constant steady-state voltage of 5V and when there is a fire hazard detection we see that the voltage drops to 199mV.

The near-infrared photodiode has a steady-state input current of about 90μA. This makes the steady-state power usage low at about 450μW. We utilized an N-

BK7 plano-convex lens to focus incoming radiation onto the sensor. We chose this material because it transmits radiation in the near-infrared spectrum and it is cost effective costing only \$35.

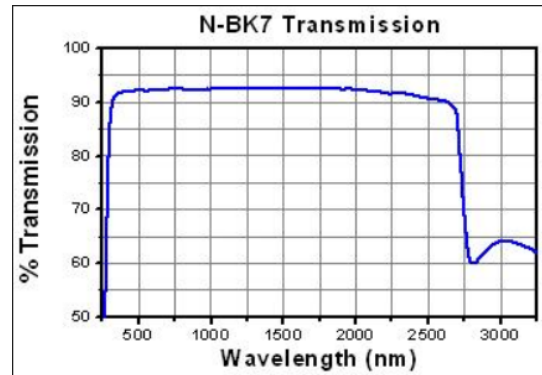


Fig. 4. N-BK7 Transmission by Wavelength

Figure 4 shows the transmission spectrum of N-BK7 and we can see that the near-infrared spectrum between 0.7 and 1.1μm has a transmission of about 95% which makes this lens viable for focusing the correct radiation onto the photodiode. The lens has a focal length of 1 inch and we chose this length so that we could have the maximum detection angle for our fire hazard detection unit. We chose a 1 inch diameter lens because it has a large surface area in comparison to the photodiode so more incident radiation will be focused onto it.

The near-infrared fire hazard detection sensors are encased in a rectangular acrylic casing. The lens is mounted onto an opening that is 1 inch above the base of the casing that the near-infrared photodiode is mounted onto. This case is easily attachable to stovetops and walls because it is only 2.5"x1.5"x1.5" and it weighs only 200 grams.

V. MOTION SENSORS

We utilized passive-infrared sensors for our motion sensor modules. The sensors use two inversely polarized sensors that passively receive infrared radiation from the surrounding environment. They are able to detect a differential in the incident radiation so it is able to detect when objects move into and out of the active areas.

The range of the motion sensor is up to 5 meters and the angle of detection is 120 degrees. This sensor is very sensitive to any change in infrared radiation, so even reflections of human blackbody radiation can set it off. This does not change the viability of this sensor for the system because any human presence should continue to trip the motion sensor. The sensor is covered by a Fresnel lens which allows the angle of detection to be so large.

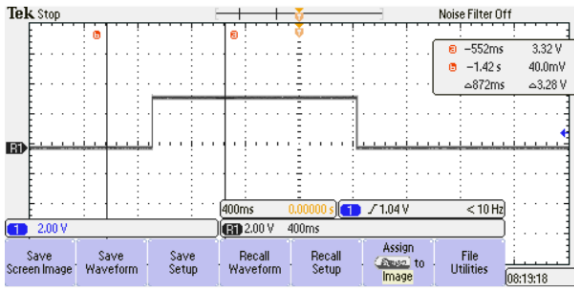


Fig. 5. PIR Output Voltage

The PIR sensors use a DC input voltage of 5V and a steady-state ‘off’ DC output voltage of 40mV. When the sensor is tripped we see a short jump in voltage to a DC ‘on’ voltage of 3.32V for 1.5 seconds. The sensor then has a 5 second reset time until it can turn on again. Figure Z shows the output voltage of the PIR sensor and we can see that there is a constant DC ‘off’ and ‘on’ voltages at the formerly mentioned voltages. The steady-state input current of the PIR sensor is equal to about 5mA. This makes the steady-state power usage very low at 16.5mW.

VI. HARDWARE – DEVICE CIRCUITS

Our system of 5 wifi enabled devices will use very similar circuit setups. The Outlet and the Switch circuit boards are almost identical. This uniformity allowed us to cut down on design time and testing since the components and design concepts are being reused and only slightly modified to fit the application. These boards were designed using EagleCAD and manufactured by Advanced Circuits (www.4pcb.com)

A. Thermostat

As mentioned, our smart Thermostat uses an array of 4 relays for its switching circuit. These relays control the function of the HVAC System by routing 24VAC power to different terminals on the HVAC’s relay controller. Since there are many different HVAC system and no ONE wiring standard we cannot accommodate for every kind of heating and cooling system. However, we believe that our configuration should work in the majority of homes. The four relays will allow us to support up to 5 output wires. These are the FAN (G), HEATER (W), COMPRESSOR (Y), and REVERSE-VALVE ON/OFF (O/B).

B. Wall Outlet and Switch

The Outlet and Switch circuits are nearly identical. The outlet operates by having a relay break the circuit between the LINE IN wire and the receptacle. The system is normally closed to ensure that the receptacle is operational regardless of the functionality of the MCU. In the Switch, the NO and NC terminals of the relay are wired in a 2-way

circuit with a standard sized paddle switch. This configuration allows for operation by both the MCU relay control and the physical switch; Both will act as a toggle. The current sensor will measure current going into the COMMON terminal of the relay, this measurement will also allow us to determine when the connected appliance is actually on.

C. Sensors

The sensors do not have specialized circuits. To accommodate the possibility of various sensor configurations this circuit was generalized to be powered by any DC voltage up to 40V. The GPIO pins are accessible through header pins and cable connections can be made between the pins and the sensor outputs.

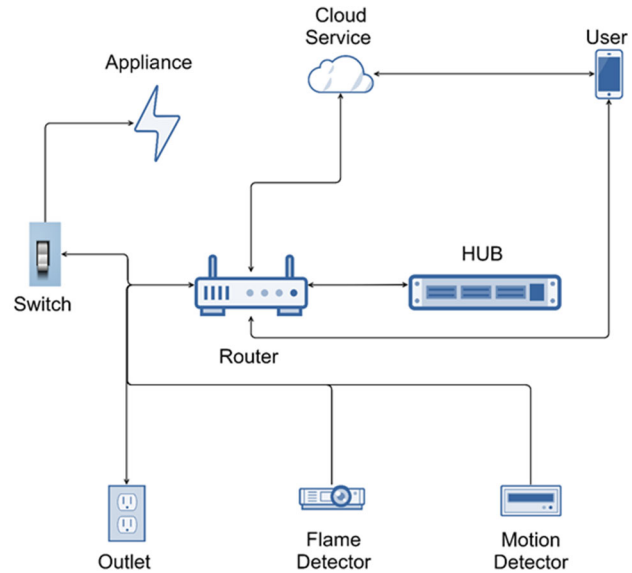


Fig. 6. System Diagram

VII. SOFTWARE- DEVICES

The system overview can be seen in figure 6. We used the Arduino IDE to flash the MCU. We chose this because of the many libraries available for the sensors and components in these circuits. The Arduino format allows us to have class based code structure which makes the code compartmentalized and easier to read and debug.

A. Connection and Communication

The devices will communicate with the System Controller using WIFI and the MQTT protocol. The system controller will broadcast a dedicated WIFI network that only our devices have access to. MQTT allows us to publish and subscribe to various data streams. We use these data streams to differentiate between the type of devices and the commands being passed. This allows us to make sure devices that are not

supposed to interact together, can't because they have no knowledge of the other devices data topics. Only the system controller is aware of all the data topics and we can minimize the risk of miscommunication through the system controller's software rather than debugging the code in the devices as well. Compartmentalism was crucial to the simplification of the project code.

B. Control Loop

To provide a seamless set up will we are connecting our devices to the system hub creating our own private network. When first powering up, the devices will be on stand-by mode awaiting a successful connection to the hub. The device will attempt to connect to the hub indefinitely. When connected, the devices will operate as normal sending out data and awaiting HUB commands. Configuring the device will be left up to the user, through the Web UI.

VIII. SOFTWARE- SYSTEM CONTROLLER

For the purpose of coordinating the myriad devices which are to be used as part of our project's system, we opted for the creation of a 'central hub' device which will act as an intermediary between the web-based sections of the system, and the 'modules', devices within the user's home which are part of the system. To manage this, we chose to make use of the Raspberry Pi 3 Model B+, as it provides the necessary functionality in terms of wireless communication capabilities and is simultaneously a powerful platform capable of running the scripts developed for the purpose of coordinating connected modules and reporting state back to the web-based

component of the project. The hardware specifications of the Raspberry Pi 3 Model B+ can be found in Table 1 on the following page.

The operating system chosen to be used in tandem with the Raspberry Pi 3 Model B+ was Raspbian, a Debian-Based Linux distribution compiled for ARM processors with the Raspberry Pi in mind. Although alternative operating systems can be run on the Raspberry Pi, Raspbian provided all of the necessary functionality for the completion of the project, and as the official operating system for the Raspberry Pi, it boasts a robust user-base which will be a significant help in regards to troubleshooting any technical issues encountered during the project development process. This is particularly helpful in the case of our project, as Raspberry Pi devices used with the Raspbian operating system are a popular choice for Internet-Of-Things applications, and thus a great deal of information related to those types of projects in particular is readily available.

A. Network Overview

For in-home networking, we have chosen Wi-Fi as the primary means of establishing connections between the Raspberry Pi, the modules which make up the smart-home system, and the web server via the Internet. It can be assumed that most clients interested in making use of such a device would have a local area network of some kind, but for those that choose not to broadcast a Wi-Fi signal, it is also possible to connect the Raspberry Pi directly to the network via the built-in Ethernet port, maintaining

TABLE 1

RASPBERRY PI 3 MODEL B+ SPECIFICATIONS

Component	Specifications
CPU	Broadcom BCM2836B0 Cortex-A53 (ARMv8) 64-bit SoC @ 1.4 GHz
Memory	1GB LPDDR2 SDRAM
Wireless Networking	2.4GHz and 5GHz IEEE 802.11.b/g/n/ac wireless LAN, Bluetooth 4.2, BLE
Wired Networking	Gigabit Ethernet over USB2.0 (maximum throughput 300 Mbps)
Pin I/O	Extended 40-pin GPIO header
USB	4 USB 2.0 Ports
Digital Video	Full-size HDMI
Analog AV	4-pole stereo output and composite video port
Storage	Micro SD port (loaded with 32GB SanDisk microSD with Raspbian installed for the purposes of this project)
Power	5V/2.5A DC Power input

the same functionality as if it were connected via Wi-Fi. The Raspberry Pi is, in turn configured to broadcast an access-point which the modules are configured to connect to. This architecture simplifies the process of establishing a connection between the hub and the modules, as the modules do not need to be configured to connect to the user's Wi-Fi network specifically and the hub device itself can be expected to have a static IP address from the point of view of the modules which are connecting to it as an access point.

To avoid potential technical pitfalls and simplify the process of configuring the Raspberry Pi to run as both a Wi-Fi access-point and as a client to a Wi-Fi LAN, the Edimax EW-7811Un USB Wi-Fi adapter was chosen and configured to serve the role of facilitating Wi-Fi client communications, with the onboard Wi-Fi module built into the Raspberry Pi 3 Model B+ is configured to run in access point mode and broadcast a network to which the client modules can connect.

The MQTT protocol was chosen as a means of organizing communications carried out by the devices and the hub, with the Raspberry Pi hub running an MQTT broker (in this case, the open-source Mosquitto MQTT Broker [1]) which the modules establish a connection to in order to publish messages to, and subscribe to, various topics. On the hub-side, the open-source Paho MQTT client module for Python is used to publish and receive messages from the broker running locally.

B. Hub Software Design

Python was chosen as the primary language for the development of the software designed to be run on the Raspberry Pi itself, as the benefits of clear, readable high-level code and relative ease of development as compared to lower-level languages such as C or C++ far outweigh any need for speed or low system memory footprint in this case. The hub software is automatically started following system boot and keeps track of changes of state reported by other devices networked to the hub, while sending commands via published MQTT statements which reflect changes made to the database entries for the various connected devices via the web interface, i.e. publishing a message for the HVAC system to turn on the 'cool' mode on the air-conditioner when the user selects said option in the web interface.

The software running on the hub locally stores JSON representations of the information stored in the

MongoDB database running on the web server, as this format closely matches the format of the data which is being manipulated remotely, and can in turn be read, modified, and used to update the database entries of for the relevant modules on the server with new information when necessary. By checking for discrepancies between the state of key fields of a module on the database as compared to their state in the local JSON listing, the software can determine whether a command should be published to change the state of the relevant module via the MQTT topic.

Local storage of JSON representations of the JSON objects stored in the MongoDB database is an important factor in reducing latency for certain triggered actions, such as having a light turn on when a particular motion sensor is triggered, and for reducing networking overhead by ensuring that messages to change the state of connected modules are only sent when such a message represents a change of state in the relevant database field. These JSON objects are only kept in memory and are automatically re-populated in the event that the system is powered down and restarted.

Devices connected to the hub are automatically registered in the database upon first registered communication, provided that a document has not already been created for them in the database prior to the first contact during this runtime. Modules are differentiated via a unique serial number with which they are hard-coded before deployment and which is designed to remain static for the entirety of their lifetime. This serial number is used both as a primary key in the database, and in differentiating modules from one another in MQTT communications carried out via the broker, i.e. both for determining which status-update messages belong to which modules, and also for the modules to determine whether a command sent from the hub is meant for another device and should be ignored, or meant for the device in question and should be used to trigger a state change of some kind.

IX. SOFTWARE- CLIENT AND SERVER

In order to support the functionalities previously discussed in the other sections, we needed a strong backend. We decided to use the popular web framework Django, written in Python. Django allowed us to quickly build a functioning website so that the devices could be tested. It uses the model-view-controller design, as shown in Fig. 6. Model is directly connected to the database, the view component is what the user sees and interacts with, and the controller component is where all the logic resides to

manage all the actions the user can perform. Our backend consists of many functionalities to improve our system and to better serve our users. Such functionalities include: user registration, dependent registration, password reset, profile editing, notifications (when the sensors are triggered), and management of devices (adding, removing, and updating). Other features that are also implemented in our system, is the ability to schedule tasks for each device. For example, it will be possible for our users to set a time when they want a smart outlet to turn off. Another scenario is the ability to turn a light on when the motion sensor is triggered.

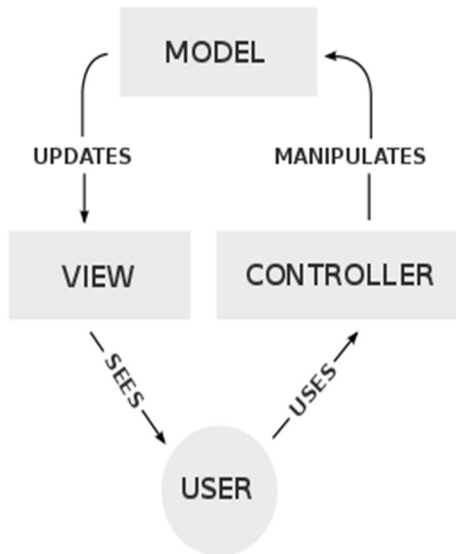


Fig. 7. Django Architecture Design

A. The Database

We decided to implement MongoDB as our database of choice. Because of that, we were also able to host such database with the cloud service MongoDB Atlas, which allowed us to reduce the load from our other cloud services and hub controller, as we will discuss further in this section. By choosing MongoDB, this also allowed us to easily manipulate the data from our devices from both the website and the controller, which is discussed in a further section.

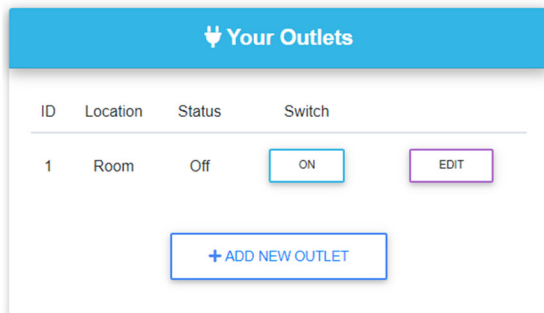
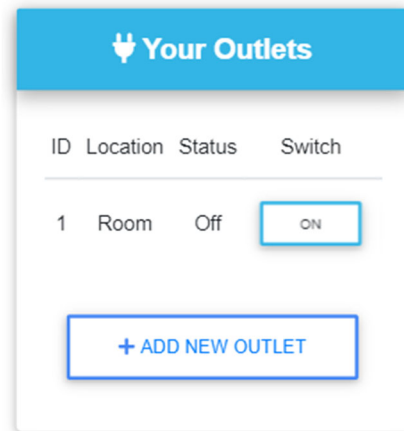


Fig. 9. Smart Outlets Management Example

Our goal with our system, is to allow the user to control the devices installed in their homes from anywhere they would like. Our website allows just that. The user is able to access their system from any supported web browser and from there, they are able to check which devices are being used, if the sensors had any activity recently, and also manage their HVAC system. An example of the management of the smart outlets is shown in Fig. 7.

B. Web Services

To achieve this, we decided to utilize cloud services to allow remote access to the system. In our project, we are using a combination of two cloud service providers. Amazon Web Services and DigitalOcean. From AWS, we are utilizing their Work Mail service, to implement a password reset functionality in case our users need to change their password. This can also be used to send any communication to our users. From DigitalOcean, we are using their compute services to host our website, which also manages our domain. Combining both of these services, we were able to keep a low cost, reducing the investment of our users. As seen in Fig. 8, we have opted to a very simple design. We opted for this design, because we want the user to be able to navigate with ease through our system, and to be presented with a familiar design language. That is why we decided to utilize Material Design Bootstrap as our frontend design.



C. Backend Framework

Fig. 8. Smart Outlets Mobile UI

For our project, we decided to utilize Django to handle both the backend and the frontend of the system. We decided to go against utilizing a standalone frontend framework because we knew Django was capable of handling both aspects. This also allowed us to keep a similar interface for the website in case the user chooses to access the system through their mobile device. As shown

in Fig. 8, the layout stays consistent no matter which device the website is being accessed from. Note: the “Edit” button can still be accessed by scrolling to the left.

D. Dashboard

Our dashboard presents the user with a list of all of their devices. The Motion Sensor, Fire Detector, HVAC system, smart lights, and smart outlets. For the Motion Sensor, the user has information about the current status (if it was recently triggered or not), as well with the date and time when it last detected motion. For the Fire Detector, similar information is displayed. For the HVAC system, the user receives information about the current temperature, humidity, and mode of operation of the device.

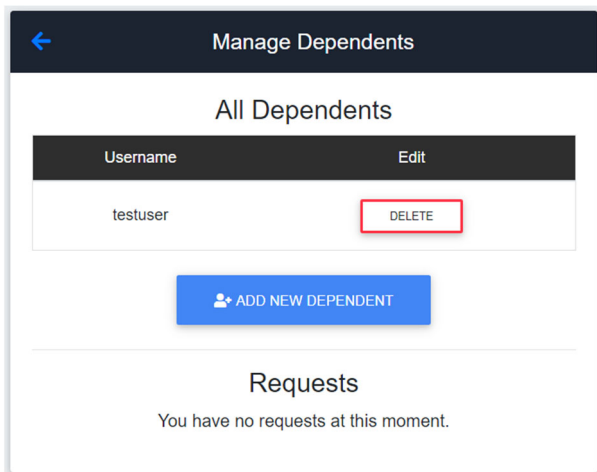


Fig. 10. Management of Dependents

The user is also able to change the temperature and the mode to cooling or heating. If the user chooses to do so, they can also see the power usage of the system. Finally, for the smart lights and outlets, the user has information about the location and status of both devices. They are also given a switch to change the status of such devices can also edit them, to change their location for example. In addition to these features, the user has the option to delete such devices, after a confirmation prompt is shown. Another feature of our system is the ability to add dependents to the ‘main’ user. A common scenario would be for the parent of a family choose to give access to a child to view the devices installed in their household.

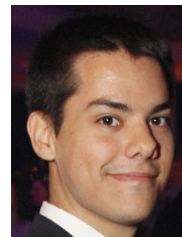
To access such feature, the main user would navigate to their profile page where they can add any number of dependents to their account. Each dependent, needs to have a account already registered in the system in order to benefit from such feature. This measurement also improves the security of the system, to avoid unwanted access to the devices in a household. Fig. 10 shows the page to add new dependents. When the request for a dependent is created, such user receives a notification where they can choose to accept or not. Upon acceptance,

the dashboard of the user will be updated to show the same devices as the master user. The dependent however, cannot add or delete devices. Their ability is limited to editing the location of the devices and changing the temperature of the HVAC system.

REFERENCES

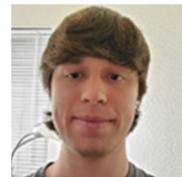
- [1] Particle. (n.d.). Retrieved April 06, 2016, from <https://docs.particle.io/datasheets/photon-datasheet/>
- [2] Split Core Current Transformer ECS1030-L72. (n.d.). Retrieved April 06, 2016, from <http://cdn.sparkfun.com/datasheets/Sensors/Current/ECS1030-L72-SPEC.pdf>
- [3] LCM-S01602DSF/A Datasheet. (n.d.). Retrieved April 06, 2016, from <http://www.mouser.com/ds/2/244/LCM-S01602DSFA-108827.pdf>
- [4] R. A. Light, "Mosquito: server and client implementation of the MQTT protocol," The Journal of Open Source Software, vol. 2, no. 13, May 2017, DOI: 10.21105/joss.00265

THE TEAM



home country.

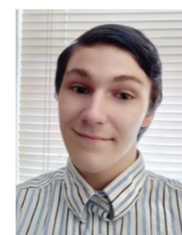
Guilherme Costa is a student from Brazil, currently in the United States pursuing his bachelor’s degree in Computer Engineering with expectations to graduate in the Fall of 2019. After graduation he plans to use the experience acquired through UCF to work as a network engineer back in his



Matthew Allen is a Computer Engineering student at the University of Central Florida, plans to pursue a career in the software development following graduation.



Felix Henriquez is a Computer Engineering student from the Dominican Republic. After graduation Felix plans to work a few years as a software contractor before starting his own software engineering firm.



Avery Stevenson is a Senior at UCF graduating with a Photonic Science and Engineering Degree from the Center of Research Education of Optics and Lasers. He plans to work with optical sensors and lasers in both production and research.