Multi-room Space Heater System

Your names and affiliation: Eric Menq, Jie Sun, Rong Feng Ye

Department of Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*— Many students who live off campus in old Pittsburgh houses struggle with a viable and flexible heating system that can cater to the preferences of multiple tenants. In our project, we provide a solution for these students by creating a remote controlled space heating system that is managed by a web application. Our system dynamically controls the temperature across different areas in a house via space heaters located throughout multiple rooms. Our system uses infrared and motion sensors to monitor occupancy and regulate temperature levels to effectively limit energy waste and circuit overload. The listed functionalities and preferences can be easily controlled by the user through our web application that is hosted on a HTTP Apache Web Server on an AWS EC2 instance.**

Index Terms

Personal space - A room or a chunk of occupied space around 100 square feet.

Infrared Motion Sensors - Motion detectors that detect moving humans using thermal heat.

Smart Plugs - Wifi smart plugs that can remotely turn the power on or off.

Temperature sensor - A smart thermometer capable of sending temperature data through IOT.

I. Introduction

As students who have lived in old off campus houses with multiple roommates throughout the past three years, we have noticed many challenges and problems that arise from this common living situation. One of the major issues in these houses during the frigid Pittsburgh winter time is the presence of a safe, reliable, and flexible heating system that can cater the varying temperature preferences of multiple tenants. Many of these Pittsburgh houses are very old, therefore lack proper insulation along with a dependable thermostat system. As a result, students have resorted to using space heaters, but their combined power wattages often lead to very expensive energy bills and breaker shut downs. Although this seems like a troubling issue for college tenants, rental demand for these off campus houses is relatively inelastic, allowing for landlords to continuously profit year to year without renovating these houses. Therefore, we hope to solve this problem for college students by creating a multi-room space heater system that is financially conscious, safe, and flexible to the varying temperature preferences of tenants. This space heater system will be regulated with a Web Application that will be hosted by an Apache HTTP server on an EC2 instance. The web application will allow for multiple users to regulate the temperature of their personal space via the application interface.

Our multi-room space heater system is primarily targeted towards an audience of technologically literate college students who live in old Pittsburgh houses. However, since the design report, we did include additional automatic integration of our devices with a Wifi hotspot to simplify the setup process and aid less technologically savvy users. With our application, college students who live off campus in old Pittsburgh houses will be able to safely regulate the temperature within their personal space without disrupting the temperature of their roommates'. They will be able to do this on the web application by scheduling a specific preferred temperature (active and idle) for the space heater in their personal space. Additionally, our application uses motion sensors to keep count of the number of people within a defined personal space, and automatically drop to a predefined idle temperature when unoccupied to lower energy consumption and decrease the risk of electrical caused incidents. This will lead to decrease in electrical breaker issues, cut down financial and electrical consumption, improve overall tenant satisfaction during the colder months. Although we do not have direct competing products, there exists similar technologies. Our product mimics the function of a modern app controlled centralized heating system. However, it is costly and highly unlikely that the landlords of these off-campus houses will go through costly renovation to install a system. Additionally, Shelly is a company that offers wireless cloud controlled electronic gadgets. Although they do not have a product as complex as a wireless heating system, they offer many tools that can be used together to create a more complex system. Specifically, we will be using the Shelly Smart Plug to regulate and keep track of electrical usage in our system.

II. Use-Case Requirements

Our goal is to help students living in standard dorms with reasonable accommodations. When coming up with the design requirement numbers, we did research on how an average cmu student lives, how much energy they consume, what they need most from a thermostat system, etc. As everyone has different housing and circuit layouts, our web application allows the users to customize their own setups. We measured the rooms in our own house, and the average area of our rooms are around 100 square feet. Hence, that will be the area of operation for each room during our testing. For bigger rooms, the users can set up two heaters in the same room and register them both on the same circuit with separate temperature sensors to ensure the room has consistent temperature throughout. The application will have a multi-user set up with login, registration, home and heater specification pages. The entire household will be able to manage their home system under one account, with each user being able to manage their individual space heaters. The first time a user accesses the page they will be redirected to the registration page of the website as they won't be allowed to access the application without an account. Once the user is registered, all of their login and heater preference information will be stored confidentially inside our database.

In the specification page, the users have the option to set their heater schedule down to the hour as well as setting their preferred temperature. The system will keep the temperature within 2 degrees Fahrenheit of the user's set temperature number. The space heaters we bought have the power to heat up the room temperature by 1 degree every 5 minutes, given that the room's around 100 square feet. We feel like these temperature requirements are more than enough for the average cmu students. For each registered user, we will keep track of their total energy consumption in watts for students who are on a budget for their electricity bill or care about their carbon footprint. From our personal experience, we also have had the case where space heaters shut down the breakers from the space heater using too much energy. Therefore, everytime the user adds a space heater, we will have an option for the heater to be added into a circuit system. Each circuit will keep track of how many space heaters are turned on. From our research, the average standard circuit breaker has a 1600 Watts capacity. Taking into account the average cmu students will have other electrical appliances running at the same time, we will have our maximum wattage from space heaters running at 800 Watts, half of the maximum circuit breaker capacity. If the current wattage is over the set limit, the system will not turn on a scheduled heater or a heater under the set temperature. Similarly, it will not allow the users to manually turn on a heater. Instead, the heaters due to be turned on will enter a queue, and once a heater gets turned on in the circuit, the heater next in queue will get turned on. This allows a fair order on which heater gets turned on in a first come first serve basis. Additionally, since the design report, we have come up with a solution for fair space heater usage for users in the same room. We simply set a 10 minute time limit for any running heater if there are other heaters on the queue, and when this expires, the running heater will be shut off and added to the end of the queue.

III. Architecture and/or Principle of Operation

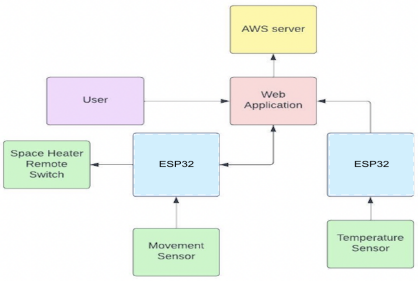

Fig. 1. Original High level Block Diagram demonstrating the design of the structure of our entire Multi-Room Space Heating System. Since then, we have automated the Heater Temperature control algorithm and logic to a separate python script apart from the Web Application that sits in its own EC2 instance. We changed it such that the ESP32 microcontroller communicates with DynamoDB tables.

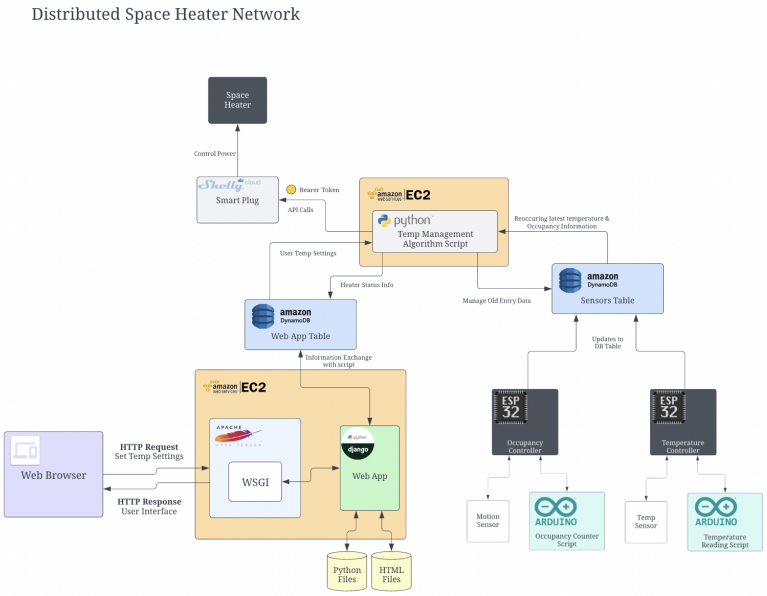

Fig. 2. Updated Block Diagram of our overall system

The user adds their temperature preferences and scheduling options to their profile by interacting with the GUI of our WebApp through a computer browser. The WebApp then uploads these temperature settings onto a DynamoDB table that feeds into our Python Script that implements the algorithm for our heating logic.

The right side showcases our hardware subsystem. Our ESP32 micro controllers are programmed with Arduino code and are connected to an AWS IOT endpoint. DynamoDB The ESP for the temperature sensor will query the sensor every 10

seconds and send the temperature reading to the endpoint via MTTQ requests. Similarly, when the ESP reads a change in occupancy from the motion sensors it sends an update in the same fashion. The AWS endpoint is configured with a routing protocol that extracts the fields in the MTTQ request and adds them to the corresponding row in the DynamoDB table. The Python script is deployed on an AWS EC2 instance and continuously reads and updates the two DynamoDB tables which represent user temperature preferences and the latest updates to the temperature and occupancy counter within the physical space. The script then makes API calls to regulate the Shelly Smart Plug that controls the power to turn on/off the space heater when needed.

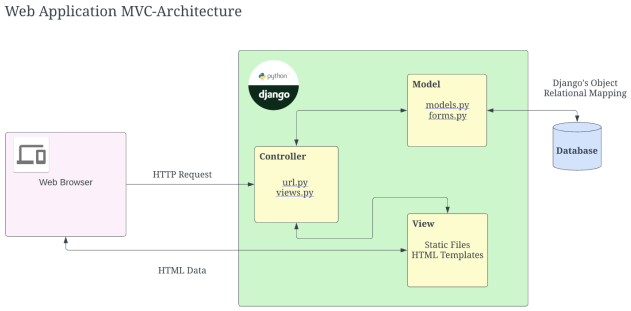

Fig. 3. High Level Block Diagram demonstrating the design of the Model-View-Controller high level architecture behind our functioning Web Application

First of all we have models, which define the fields we are using in the application. For our models we have heater which has heater id, heater power status, heater circuit number, the user it belongs to, and the location; circuits which includes the user group and the heaters in that circuit; and user profiles. The models created are stored in the django database.

Then there are templates, which are html files used to outline what each webpage looks like. We have the login and registration pages, which simply have the login and registration forms respectively. Once the user is logged in, they will have access to the homepage, the adding heater page, the heater customization page, and circuits page. All pages have headers with user ids and links back to the homepage. The homepage will have a list of all the heaters the user are linked to. The heater customization page will allow the user to change the settings for each heater and the circuits page displays all the circuits created.

Lastly we have the controllers, which interact with both the views and the models. Everytime a web page is accessed or a button is pressed, it will send a request to the controller, which then processes the data and redirects the user to their destination. For example, when a user is registers a heater on the add heater page, the controller will save the heater data in the Django database by creating a new heater model; then it will redirect the user to the homepage template where it loads all the heaters belonging to the user within the Django database; allowing users to check that the heater was created successfully. At the same time, it can access the database when the user clicks on customize for one of the heaters. The controller will consequently access the model database to find the heater information corresponding to the one user clicked on and sends it to the templates for displaying in the user's web browser.

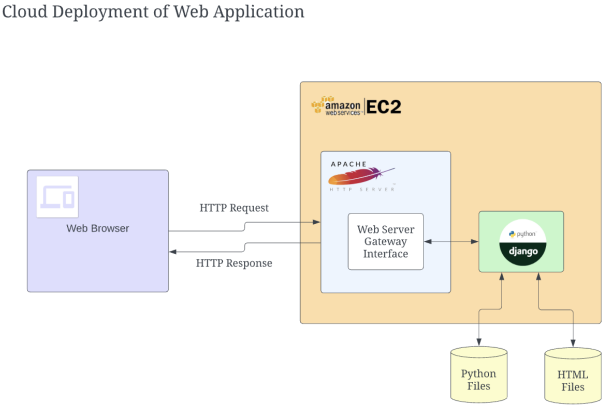

Fig. 4. A closer look at the specifics of how a user can interact with our web application, which is deployed on an HTTP Apache Web Server on an AWS EC2 instance.

The EC2 server acts as the host for the django server. When the user interacts with our website, it will send a HTTP request to the web server gateway interface, which in turn accesses the django web application hosted on the AWS EC2 server. Once django receives this request, it will act as described in the previous figure.

III. Design Requirements

Our Design requires all communication between devices and the Web Application to be 100% successful, meaning nothing is dropped and all HTTP requests are successful. These communications must occur within 100ms such that we can meet our use case requirements. Our occupancy counter must be 95% accurate, meaning it never fails to count someone walking through the door, or overcounts. However, realistically, we have to add a limitation to this requirement that assumes people walk into the room one at a time and are all adults, as the sensors will be placed at torso level. To meet our use case requirement of heating up the room by 1 degree every 5 minutes, it is also a design requirement to find a space heater that can achieve this while balancing power usage so that our power consumption requirements can be met. This being said, it will be a requirement for our heating system to never break our 800 Watt limit, sacrificing our temperature requirements for this. To meet our requirement of maintaining a +- 2 degree temperature range from the user setting, we need a temperature sensor that is accurate to 1 degree, and the temperature sensor will turn the heater on or off within 1 second of exceeding 1 degree of the set temperature range.

For our web application, we will upload the system onto an AWS EC2 instance. We will require it to provide robust computing power capable of handling up to 50 different user accounts on all major web browsers and 200 distinct space heaters.

Once we have all the basic design requirements reached, there are a few more requirements to implement. First, we need to design algorithms that will hit our goal to reduce energy consumption. We believe, with our consistent temperature and maintenance, scheduling, and occupancy detectors, we can reach a requirement of decreasing average energy consumption by 15%. We also have a requirement for competing space heaters. For example, if two people are in the same room with vastly different temperature settings, it will be impossible to perfectly achieve their temperature settings due to the thermal flow of heat from one side of the room to the other. Since space heaters can only increase temperature, this means the person who wants the room warmer will have a temperature much closer to their preference compared to the other. In this situation and similar ones, such as open doorways and poorly insulated walls, we will write algorithms such that there is a compromise between users that are adjacent to each other in the house. More specifically, if two users are next to each other, and the temperature preference cannot be met, our software will recognize this and achieve a temperature for both users that is equally far from their setting.

IV. Design Trade Studies

Table I. Microcontroller Requirements

Feature	*ESP32*	*RSPi*	*Necessary?*
Built in Wifi connection	yes	yes	yes
Size	Fits on one column of breadboard	~ twice as large	NA
GPIO pinouts with ADC	Yes	Yes	Yes
OS	No	Yes	No
CPU capacity	lower	High	No

a.

Our first design choice was to decide which microcontroller we should use to control our sensors. Although initially we considered the Arduino Mini Wifi and the Raspberry Pi Nano, we eventually decided to use the ESP32. While the Arduinos and Raspberry Pis provide greater capabilities, all three of these microcontrollers satisfy our specific design requirements (shown in the table above). However, the ESP32 is the cheapest and most available, giving us more room in our budget and less time waiting for materials to arrive. One helpful advantage of the RSPi is its higher CPU, which would be helpful when connected to multiple sensors. However, with some research, we were confident the ESP32 could handle 2 sensors and our occupancy detector code. As a result, our tradeoff for a smaller, simpler, and easier to obtain microcontroller is better than using a microcontroller that has more features we can live without in our design.

Another tradeoff we made was whether infrared detectors or infrared motion sensors. The drawback for the motion sensors is that if it ever failed, our occupancy counter would be incorrect, which would be an inconvenience for the user to fix, even if we implemented a way for them to do that. However, the reason we chose the motion sensors is that the detectors may miss blind spots in rooms that are not rectangular. Additionally, we could not find commercially available detectors that are built for automated use, only detector guns that have to be manually operated by humans. In a real world use without our resource constraints, we would choose to design our own infrared detector, but it was not feasible for our project.

We also had to decide how we implemented our motion sensors. One option was to have a motion sensor that could sense the whole room. The other option was to build an occupancy counter at the door using two weaker sensors. The issue with using one, more powerful sensor, is that we would run into the same problem with the blind spots. Additionally it would not be able to detect someone if they are sleeping or if the person is not moving within their room (similar to how motion detector lights will go off if you stay still in a room). This means they would have to move around in order to maintain their temperature setting they want when they are in the room.

Another decision we had to make within the system of our occupancy sensor was how to place our sensors. Our first option, which we saw in other implementations of occupancy counters, was to place the sensors far apart on the breadboard. However, after testing, this would struggle to meet our requirement of near-perfect occupancy detection, as both sensors may sense movement while someone is walking through. As a result, we chose to take the sensors off the breadboard using three long ribbon cables and tape them on the walls inside and outside of the door frame. Although this is less convenient, we found it worth it, because we could physically separate the sensors so that we can guarantee the sensor inside the room only goes off when someone is moving inside the room, and vice versa. Finally, we chose to do this because it allows us to put the sensors at the torso level of most people as they walk through the door, which we found to be much more reliable than placing it on the floor.

$$Watts = \text{Amps}*\text{Volts} \quad (1)$$

One key design choice for us is what our threshold is for the maximum electrical power our space heaters can use on one circuit. We first researched the average maximum wattage for most circuits in houses, and found a range of 1500-2000 Watts. We then looked at the specs for breaker boxes in older houses and found they have a max electrical load of 25 Amps. Given the standard electrical supply to houses of 120V, this is 3,000 Watts. A large appliance, such as a washer, will use 100V and 15 Amps. Using equation (1), this is 1500 Watts. Similarly, two medium devices, like a hair dryer or microwave, will be around 700-1000 Watts. As a result, we felt 800 Watts would be a good threshold for our space heaters, giving about 200 extra Watts if two medium devices are being used, or one large appliance and a smaller device.

Onto the software side of things, there were also many key design choices that we made which streamline the cohesiveness of our system and optimize the cost and software which we use. The first big choice was picking between a website and a web application. Firstly, a website is a collection of interlinked web pages with the same domain name. But a web application is a program or software that a user can access through a web browser. We selected the web application because Web applications have complemented the ever increasing sales of eCommerce and Retail growth within the United States in the past two decades.

The design of web applications are generally based off of the high level architecture of Model, View, Controller, which can be seen in Figure 2. To construct our web application, we leveraged the Django framework to do so. Django has been a rapidly growing user-centric framework with detailed documentation that simplifies development for Web Application engineers. It simplifies the routing of different pages within the application to easily link the Model and Views to the controller. It's extremely hackable in a sense that it provides many optional settings and extensions for the developer to specify.

After the development of our web application, we deployed our software service onto an Apache HTTP server. The reason which we selected Apache is because it has been one of the major leading choices as a server within the industry throughout the past few decades. Its continued presence demonstrates that it is a solid reliable choice for the deployment and scalability of our product.

Lastly, to host our Apache HTTP server, we selected an Amazon Web Server EC2 instance. Working with cloud based tools shifts the large capital expenditure costs of hosting into a more streamlined experience of discounted cash flows (subscription model) and hedges the costs risk for an independent developing team like us. With that in mind, there were other additional cloud options like Heroku and Google AppEngine in the market. The reason we chose AWS is because it provides cheap and reliable service costing us 10 cents per day. AWS also provides their customers with a one year free trial. Costs aside, it can be configured with many software tools and frameworks, and also has incredible security backed by cryptographic keys like SSH and SSL. We also have experience working with EC2 during our Web application class as well as our internships which gave us additional comfort with selecting AWS.

V. System Implementation

A. Subsystem A

The first system is the hardware system, as mentioned before, we have the temperature sensor, infrared motion sensor, and the wifi remote plug. Both the temperature sensor and the infrared motion sensor will be connected to a breadboard to the ESP32. There will be 2 infrared sensors for each system, which will be placed on the doorframe of the user's room and connected to the breadboard via extended jumper wires.. a room, we will be using the infrared sensors to One sensor will be placed on the doorframe facing the corridor, while the other will be placed on the doorframe facing the inside of the room. With this setup, we are able to tell whether someone is entering or exiting the room by the order these sensors go off. For ex, if the outside sensor goes off first and the inside one afterwards, we can be certain that the person is entering the room, by which we increase the occupancy by 1. On the other hand, if the inside sensor goes off before the outside sensor, we know that the person is exiting the room, hence we decrease the occupancy by 1. We are assuming that only 1 person will be entering or exiting the room at the same time. Additionally, we added aluminum foil on the outside of the sensors to limit their sensing range such that people moving in close proximity to the door would not set off the motion sensors. This information will then be processed on the ESP32, keeping track of the number of occupants in that room. When the occupancy status changes, the ESP will send a MTTQ request containing a time stamp, the HeaterID, and the occupancy status to the AWS endpoint. In order to make this request, we defined a IOT "thing" that represents the ESP in AWS and configured its certificates and policies that allow secure data requests. We then wrote a message routing rule that parses the MTTQ requests and creates a DynamoDB action that uses the parsed fields. This will create a new entry in the DynamoDB table, using the timestamp as the primary key and the other fields as columns.

We will also be using a smart temperature sensor which is also connected to the ESP32. The temperature sensor is used to measure the temperature in the user's living space, such as a desk or a bed. After testing, we added a 5V voltage module to the temperature sensor, as it requires more than the 3.3V provided by the ESP to operate. A pull-up resistor was also added so that the initial input voltage when serial communication begins is "High," which is necessary based on the code in the temperature sensor library. The temperature sensor then sends temperature data to the ESP32 every 10 seconds. The ESP will then forward this information to AWS using the same process as described previously, and the request will contain the timestamp, HeaterID, and the current temperature. This creates a new entry that has a null column for occupancy, but contains a value in the temperature column.

We are also using the smart shelly plugs that we connect the space heater with. The plugs can be remotely controlled via the internet. Shelly provides a list of API calls that we are using to control and monitor the space heaters. This entails powering the plug on or off, and wattage monitoring. If the python script finds that the temperature is out of range from the temperature sensor data, it will execute the API call that turns off the remote plug, turning off the space heater, and vice versa with turning the heater back on.

Finally, both the ESP and Shelly will be preconfigured to connect to a Wifi Hotspot rather than connecting directly to the home wifi. This will be provided for each room. This allows for a consistent internet connection that can mitigate connectivity issues and makes the setup for the user much easier.

B. Subsystem B

For the software subsystem, we have the AWS EC2 instance and the Django Web Application. For our web application, we are first implementing a multi user structure in order to accommodate a bigger household. For ex., the house we are currently living in has 12 occupants, with each occupant having their own preference of temperature setting. Hence allowing everyone to have their own accounts which leads to less arguments about thermostat settings and happier students. We are using Django to set up our web application since we have experience with this during our time in the web application class.

As mentioned in the design report from earlier in the semester, we completed the web application as the first step. It was capable of supporting up to 50 users at the same time. Within the application, we have register and sign in pages for users visiting the website for the first time; add heater page for users to register their heaters; edit heater page for users to edit their heating configurations; and circuit page for the users to add their heater in a circuit. This all worked, and users were able to input their information into the Django database easily, and from our user surveys it was easy to understand even for non technical users. However, we started running into issues when we tried to implement the algorithm to control the heaters according to the user's preferences. Our original idea was to for the web application to access the DynamoDB table directly, and storing all the user data in Django. However, there were a couple of complications related to that implementation. First of all, we need to run a while true loop that checks the sensor data updated onto DynamoDB, checks the data with user's settings, and in turn powers the heaters on or off using the shelly smart plugs. And doing so inside the web application will be very inefficient since the web application is already handling the users requests. Secondly, we found that we need two dynamoDB tables. The ESP32 uses a different API call than python, and it can only add entries into the DynamoDB table, and unable to edit or delete any entries. Therefore, we need one table for the ESP32, and one table for the web application. The ESP 32 table is for all the sensor's data: Temperature and occupancy readings; and the web application table is for the user's preferences, such as their preferred temperature, settings, circuits they are connected to etc. In order to connect both tables, we decided to run a separate python script on an EC2 server. The script will be responsible for checking both DynamoDB tables, updating each other, and turning the shelly smart plugs on or off. It runs in a while true loop and sleeps for 2 seconds between each iteration. During each iteration, it will check the web application table, if there's new heater information, it will be stored in the script locally. After that, it will check the sensors table, if the temperature exceeds the user's preference, or the occupancy sensor indicates the room is empty, it will turn the heater off using the shelly smart plug API call. The script also handles the circuit implementation. If more than two heaters are turned on within the same circuit ID on the heater, it will enter a queue in the script, and the next time a heater gets turned off in that circuit, it will pop a heater off the queue and turn it on.

While writing the script and web application, we ran into some issues connecting the subsystems. First of all, there were some race conditions between the script and the web applicaiton. In the web application, when the user selects manual on in their heater preference, the web application side would automatically toggle the heater as on in the dyanmoDB table, disregarding the circuit system. To counteract this, we disabled the manual turning on option when adding the heater in the web application. As a result the user can only turn it on manually when editing their heater preference, and the script will turn it on accordingly based on the proper circuit system. Also, if the heater was previously on automatic and is then turned on manually while the script is running, sometimes it would lead to a race condition. The web application will switch the power to on, but since the script hasn't been updated, it will power the heater off because it was still on the automatic setting when computing the sensors data. This lead to the heater being turned off even though it should be manually turned on. To counteract this, we added another field to the dynamoDB table which indicates the mode of the heater: 1 being automatic, 2 being manual on, and 3 being manual off. This will give users proper control over the heater's power.

VI. Test, Verification and Validation

The testing of our software was broken down into unit testing the software, unit testing the hardware, before finally unit and integration testing the combination of hardware and software into our entire Multi-room space heater system. For each of these unit and integration tests, we verified the fulfillment of user-case requirements which we have theoretically defined back from the beginning of the semester. To also properly gauge the user-experience of our product, we were able to successfully poll 15 college students that live in off campus housing to gauge the user satisfaction of our product. We collected feedback with surveys to not only gauge the demand and need for our product but also to tweak commonly mentioned complaints and issues that may have slipped through the cracks because of our lack of diversity and lack of foresight. Receiving alternative perspectives was critical for the development of our project to properly package the final creation of our service for the demo.

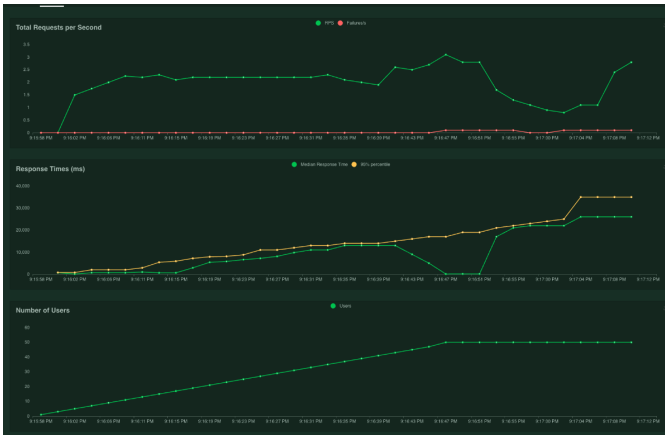

Fig. 5. Using the Python Locust library, these are the Load-testing results of our WebApp that is hosted on an EC2 instance on the AWS free tier. Because of such, we can see that HTTP failures and server response times increase as user requests approach our defined capacity limit of 50 users.

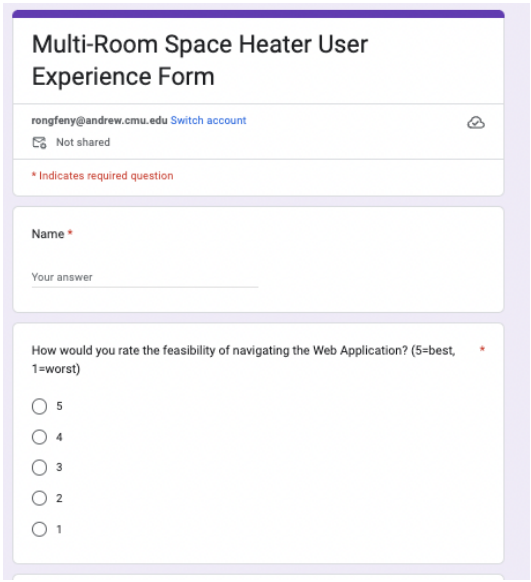

Fig. 6. An user-experience form was given to our friends who volunteered to test out our product. Since the form is long, only the first question is shown in this picture

A. Results for the load bandwidth of our WebApp

As shown in Figure 5, we tested the load-carrying capacity of the application. We conducted a load test using the Python Locust library. Our web application is hosted on an EC2 instance on the AWS free tier, and we measured the server response time and HTTP failures as we increased the number of virtual users accessing the website.

Our load testing results showed that our web application can robustly handle up to 50 users simultaneously. However, as we increased the number of virtual users beyond this limit, the number of HTTP failures increased, and the server response time also increased.

To further analyze the limitations that prevent us from exceeding this user-case specification of the load bandwidth of our web application, we analyzed the plot (generated by the Locust graphical user interface) comparing the value we analyzed in the design to the measurements we made after implementing our web application. The plot clearly shows that our web application can handle up to 50 users simultaneously, as per the design specification. However, as the load increases beyond this limit, the performance of the web application starts to degrade, leading to HTTP failures and increased server response time.

In our previous section IV, we discussed the significance of our load testing results and how they help us improve the performance of our web application. We also compared our measurements to the theoretical predictions we made earlier and found that they successfully match closely.

B. Results for User satisfaction

To ensure that our product meets or exceeds Specification of user experience satisfaction, we conducted a survey with 15 of our friends. The survey consisted of 5 questions on a scale of 1-5 (5 being the best), and we used the Google Form platform to collect the responses.

Our survey results showed that we achieved an average total satisfaction score of 4.6, which indicates that our product is meeting the user's expectations and providing a satisfactory experience.

To analyze the limitations that might prevent us from meeting or exceeding this specification, we created a plot comparing the value we analyzed in the design to the measurements we made after implementing our product. However, since the measurement of user experience satisfaction is subjective, we cannot represent it numerically in a plot. Instead, we can represent the survey results using a graph that displays the distribution of responses across different questions.

Our survey results indicate that most of our users rated our product highly in terms of user experience satisfaction. However, we also received some feedback that can help us identify areas where we can improve the product.

In section IV, we discussed the significance of our survey results and how they help us understand the user's perspective and improve the product accordingly. We also compared our measurements to the theoretical predictions, and since the measurement of user experience satisfaction is subjective, there is no theoretical prediction to compare.

C. Results for Hardware Requirments

asdf

Our design requirement for our motion sensors is 95% accuracy in detecting occupancy. This is closely related to our use case of bringing the personal space to a lower temperature

when the user is not in the room. We tested this by having all 3 of us continuously walking in and out of the room while monitoring the occupancy outputs on the Arduino serial monitor. Then, we simply counted how many times the occupancy sensor failed to register someone entering or leaving the room. In general, we were able to consistently detect exiting and entering with 90% success rate, which is below our requirement of 95%. However, the occupancy sensor is sometimes able to perform almost flawlessly for many minutes, but starts failing more. Now that Eric's finals are complete, he will be doing some last minute testing to try to make the sensors reach our requirement consistently.

The temperature sensor is a simpler setup, so we were hoping for a 100% success rate in measuring temperature with a precision within half a degree, which is crucial for our use case requirement to maintain temperature within 2 degrees fahrenheit. This was successfully achieved when we tested the temperature sensor against a handgun thermometer while constantly changing the temperature with a space heater and opening the window. There was a small delay, as it took the sensors around 15 seconds to respond to temperature changes, but this should be a minimal issue when it comes to user experience, considering the temperature usually won't change more than a degree within a minute. The sensor also has a precision to 2 decimal places in Fahrenheit, but it was difficult to measure exactly how accurate it was as the handgun thermometer was likely not exactly accurate. But we found it measured within 0.1 degrees of the thermometer >95% when the 15 second delay is accounted for.

We also needed the Shelly to turn on and off the power for the space heaters with 100% success, which was achieved by manually sending API calls to the Shelly.

D. Results for Software Requirements

For software requirements, our design requirements state that all communication between the devices and web application has to be 100% successful. We tested that by configuring the user settings on the web application, and see if all the changes are reflected correctly on the dynamoDB. This was successful 100% of the time. In turn, we also tested if the script received all the data in the dynamoDB, and after printing out all the entires on the script side, this was also 100% successful. The requirements also indicate that we need to ensure 100% data transfer between the hardware and the script. That was tested by checking if all the data that was supposed to be on the dynamoDB table were sent and received. After testing this continuously, it was also 100% successful.

The communication between the subsystems is done through amazon web servers, theoretically is should be sent within 100ms. However, depending on the server load or the free tier bandwidth, it could take longer. In any case it should not affect normal usage.

The circuit queue system was also thoroughly tested. As mentioned earlier in the same circuit there should never be more than 2 heaters turned on at the same time. We tested this by adding all three heaters in the same circuit, and turning them all on either by setting a high temperature setting, or turning it on manually on user's preference. Upon doing so, the heater that was supposed turned on last did not turn on. It is also stated on the website that the last heater was entered into a queue. Then, we turned one of the heaters off, and this in turn turned on the last heater. We also added some bogey heaters to test if the system works in a circuit with more than 3 heaters, and it is working as intended.

E. Results for Overall Use Cases

Overall, our system was able to meet our use case requirements of maintaining a user set temperature within 2 degrees and follow a schedule. Additionally, it was able to increase the temperature by more than 1 degree every 5 minutes in a personal space. We were able to test this during integration testing by running our entire project and measuring the temperature with the handgun thermometer. The only times it would fail is once again, in the case the occupancy counter failed.

We were also able to test the responsiveness of the system to turn on or turn off the heaters within 5 seconds of the temperature leaving the threshold boundaries. We tested this by watching the temperature outputs from the temperature sensors, and making sure the space heaters were turned on and off within 5 seconds.

VII. Project Management

A. Schedule

The Gantt Chart at the end of our report demonstrates scheduling of tasks for our project throughout the course of the semester. As we can see, the green indicates tasks that were completed earlier than scheduled, whereas red indicates a task that was completed later than expected.

B. Team Member Responsibilities

Eric focused more on the hardware and hands-on tasks relevant to the setup and construction of the system consisting of temperature & motion sensors, smart plugs, and space heaters. Due to his great meticulous nature and internship experience on Amazon's Device Team, Eric is also responsible for researching and choosing hardware & electronic gadgets that meet constraint requirements and fit the tradeoff between technical specifications and product cost.

Jay focused on the software elements of the project. He was be responsible for the design and development of the web application. Leveraging his knowledge from his coursework in 17437 and 15440, as well as his experience during his time interning at Amazon Web Services Jay designed the high-level architecture of the Model, View, and Control that would allow for a safe, scalable, and modular application. At the same time, he worked on the python script that acted as the middle man between the web application and hardware sensors. He was be also responsible for the Unit and Integration testing of different software components throughout the application.

Rong focused more so on a product managerial role, coordinating between the Professors, TA, and the team to

ensure smooth communication of roadblocks, design choices, and project progress. Additionally, he worked alongside to help Jay and Eric bridge the gap between the software and hardware. This means Rong also tested Unit test the functionality of the electronic hardware gadgets before helping Eric implement it within a larger system. Finally, after Jay has completely developed the application, Rong was responsible for deploying it onto an Apache HTTP Server that runs on an EC2 instance.

Fig. 7. Gantt chart showing schedule example with milestones and team responsibilities

C. *Bill of Materials and Budget*

Product	Manufacturer	Quantity	Unit Cost	Total Cost W Shipping
Space Heater	Riomor	3	19.99	63.57
Temperature Sensors	Aideepen	3	9.99	31.27
ESP32 Microcontroller	HiLetgo	6	10.99	69.9
Motion Sensors	Onyehn	1	10.99	11.65
Breadboards	Ambberdr	1	9.99	10.88
Jumper Wires	Bestlus	2	15.55	32.78
Camping Tent	Wakeman	1	18.39	19.68
Power Supply Adapter	Arkare	3	7.19	22.86
Unlocked Hotspot	Inseego	2	80.28	170.16
			Grand Total	432.75

D. *AWS Usage [if credits requested/used]*

Originally, we planned on requesting credits, but since the operations we wanted to perform (occasional reading and writing from DynamoDB, receiving MTTQ requests) were small, we realized we didn't need to, as it would only cost us a few cents per month with the AWS free tier. When we load tested, we used considerably more resources, but since it was for a short period of time, the cost also came out to be less than a dollar.

E. *Risk Management (used to be Risk Mitigation Plans in Design Document)*

Our risk management strategy when faced with roadblocks was to proactively replan our schedule to give ourselves time to address them.

Another risk we faced was the variability of housing. Every house has very different attributes, whether it's different room sizes, break load capacities, insulation, etc. We were able to mitigate these potential issues in two ways. First, we made the assumption that the user's house would be old with a weak circuit system. This way, we could ensure that our requirement to prevent circuit breakages would be met for all houses. Second, when testing, we spent significant effort setting up our project in multiple friend's houses, taught them how to use it, and asked for feedback to make sure our project worked with different sized rooms and insulation. While this is not a guarantee our project works for all variabilities, as we could not test every house, it is a good indication our methodology generalizes well to most student housing near CMU.

Midway through our project, we realized that our use case requirement for an easy user setup might be challenged as in order to connect to the Wifi, the user would have to manually put in their Wifi credentials in the Arduino code and upload it to the ESP. Additionally, they would have to configure the Shelly plug. When we were facing issues with connecting the Shelly with CMU-Devices, we realized we could fix both these issues by connecting our devices to a Wifi hotspot, therefore mitigating a networking risk during the demo and making the user setup experience much more convenient.

We also faced the issue of weather, as our project is intended to be used in the winter, when the outside of the house is significantly colder. This is due to the fact that the space heaters can only increase temperature, so we needed that outside factor to lower the temperature when our space heaters were turned off. We mitigated this risk by having a backup plan of using our air conditioning units instead of space heaters if necessary. Luckily, there were plenty of cool days this Spring, and if we needed to, we could also just run the air conditioner units simultaneously while testing the space heaters. so we did not need to use the backup plan.

We recognized a race condition risk, as mentioned in the software implemntation section.

Overall, we did a very good job of responding to issues that came up and mitigating their risks to our project. However, in the future, we hope to do a better job of mitigating potential risks that may come up later in our schedule. For example, we wished we had spent more time and effort doing detailed research on our hardware devices to have made a better selection, as the specific temperature sensor we chose has historically had timing issues with the ESP. One risk we did proactively mitigate is that we knew our project would face significant integration challenges due to the complexity of our wireless communications. We allocated a large chunk of our schedule for integration, which turned out to be very successful for giving us time to deal with integration issues, as well as the sensor roadblocks we were facing.

VIII. Ethical Issues

One of the worst case scenarios that we discussed as a team that could happen with our product is if someone intentionally (or unintentionally) set the space heater preference temperature to the highest setting and pointed it at an extremely combustible surface. Additionally, let's say there were two doors to this room/personal space such that the motion sensor responsible for maintaining the head count within the personal space does not drop to zero when the person leaves — therefore the space heater stays at the functioning temperature rather than the lower idle temperature.

This worse case scenario is different from our usual case scenario in that we assume that our users are performing rationally. Our product is also directed towards an audience that isn't necessarily financially well off, therefore they should be conscious of leveraging our product to create a cost-friendly solution to maintaining a warm household temperature.

In terms of who gets harmed in this scenario in this very

specific case, the space heater is running a very high wattage and using up a lot of electricity. Additionally, it may start an electrical fire on the combustible surface and lead to a larger accident. In this worst case scenario, there are hazardous results and expensive monetary consequences as well.

Therefore, the homeowner and/or the tenants are directly harmed in this situation. Additionally, by taking up first responder resources (firefighting, ambulatory) you may be unintentionally harming other people in society from this unnecessary situation.

For this case, the ethical concepts that we see being violated are trust and responsibility. As the creators of the product, we are designing the use case of our product to enhance the living experience of our intended audience at a low cost. However, by providing this open ended trust and placing the responsibility in the hands of the user, we leave our product to be misused in ways that may be intentional or unintentional.

After thorough class discussion, we were able to learn about other perspectives about the possible ethical and societal impacts of our project and its use case. One additional scenario discussed was about the usage of our occupancy data that is recorded by our motion sensors. Our motion sensor setup continuously records and updates the number of people within a defined personal space. This is continuously updated such that our algorithm for temperature maintenance logic can be properly managed on our software side. However, this data can be used by malicious actors or hackers to stalk or monitor the location of the users of our Multi-room space heater application.

This worse case scenario is different from our usual case scenario in that the occupancy data is now not used for regulating one's electrical and utility bill consumption. Our product is directed towards college students who live off campus in living situations where there are often multiple students living within one house. This can result in the valuable belongings of multiple students placed at risk. Therefore, the tenants are directly harmed from this situation.

For this case, the ethical concepts that we see being violated are also trust and responsibility. As the creators of the product, we are responsible for defining user-case requirements as well as the potential risks that our product presents for users. Because of this, we re-evaluated our system design following this classroom discussion. We decided to utilize AWS DynamoDB tables to ensure a more robust and secure management of our user's sensitive data that can be easily manipulated.

IX. Related Work

As briefly mentioned earlier in the Introduction section, the market currently does not have directly competing products, but there exists similar technologies. This can be attributed to the very unique user population of college tenants and the inelastic demand on rental property of off-campus houses near private universities.

First off, our product mimics the function of a modern app controlled centralized heating system. Both products provide the service of specifying and regulating temperature in a dynamic and safe manner that can cater towards the needs of multiple tenants across different personal spaces within a house. However, it is costly and highly unlikely that the landlords of these off-campus houses will go through costly renovation to install a system. Our project can be seen as a jank band aid for an issue whose root cause has corresponding low incentives to be addressed by landlords.

Additionally, Shelly is a company that offers wireless cloud controlled electronic gadgets. Although they do not have a product as complex as a wireless heating system, they offer many tools that can be used together to create a more complex system. Because of such, we used Shelly products as a complementary gadget for our Multi-room space heating system. Specifically we used the Shelly Smart Plug to regulate and keep track of electrical usage in our system.

X. Summary

As students who lived off campus for 3 years, inadequate and uncomfortable heating was an issue that we faced. We heavily resonated with that and wanted to create a solution to this issue. Overall, the course of development, final results of our project have pleasantly reached our expectations for our defined goals of the user-case requirements. Our classroom discussions have also provided us with additional perspectives to unforeseen ethical and societal dilemmas of our design that allowed us to reassess and carefully retool the details of our project. Additionally, through our personal struggles debugging the hardware and software, we have become more comfortable with these ECE tools to be used beyond the scope of neatly defined classroom assignments. Lastly, through the course of our project, our major lessons learned as a team: Research resources & equipment before purchasing them, thoroughly understanding devices beforehand instead of jumping in and dealing with trial and error, expect errors & have mitigation plans.

References

[1] Ardumotive_com, and Instructables. "Arduino IOT: Temperature and Humidity (with ESP8266 WIFI)." *Instructables*, Instructables, 30 Sept. 2017, https://www.instructables.com/Arduino-IOT-Temperature-and-Humidity-With-ESP8266-/.

[2] Beginners, ESP for. "What Is the Difference between an Arduino/ESP and a Raspberry Pi?" *ESP for Beginners Atom*, https://www.espforbeginners.com/guides/differences-between-arduino-raspberry-pi/.

[3] Bennett, Jessica. "How to Calculate Your Home's Electrical Load and What It Means for Your Power Needs." *Better Homes & Gardens*, Better Homes & Gardens, 26 Oct. 2022, https://www.bhg.com/home-improvement/electrical/how-to-check-your-homes-electrical-capacity/#:~:text=But%20how%20do%20you%20tell,individual%20breakers%20in%20the%20box.

[4] Harrrry, and Instructables. “Ultrasonic Occupancy Counter (2-Way).” *Instructables*, Instructables, 29 Apr. 2021, https://www.instructables.com/Occupancy-Counter-2-way/.

[5] “HTTP.” *Shelly Technical Documentation*, https://shelly-api-docs.shelly.cloud/gen2/ComponentsAndServices/HTTP/.

[6] ProjectPro. “AWS vs Azure-Who Is the Big Winner in the Cloud War?” *ProjectPro*, ProjectPro, 6 June 2022, https://www.projectpro.io/article/aws-vs-azure-who-is-the-big-winner-in-the-cloud-war/401.

TABLE II. TEAM SCHEDULE