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 in Pittsburgh struggle with heating system from older house designs. In our project, we will provide a solution for these students by creating a remote-controlled space heating system that is managed by a web application. The goal of this system is to provide temperature control via space heaters across multiple rooms. Our system will limit energy waste and overloaded circuits by using motion sensors to track whether each room is occupied or not. All of this will be easily controlled by the user through a web application, allowing custom temperature schedules and settings for each room.**

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. Many 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 through the application interface.

Our multi-room space heater system will be primarily targeted towards an audience of technologically literate college students who live in old Pittsburgh houses. 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 in 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 comfort and 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 as we can test the implementation in our own rooms. 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.

III. Architecture and/or Principle of Operation

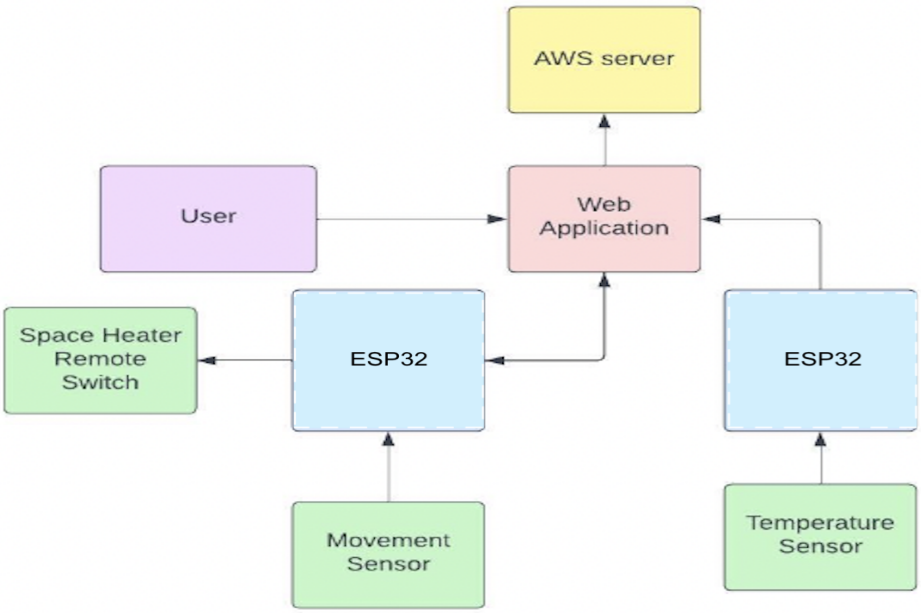


Fig. 1. High level Block Diagram demonstrating the design of the structure of our entire Multi-Room Space Heating System

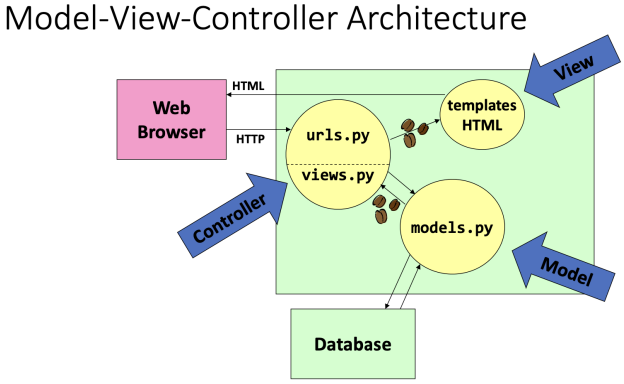


Fig. 2. High Level Block Diagram demonstrating the design of the High level architecture behind our functioning Web Application

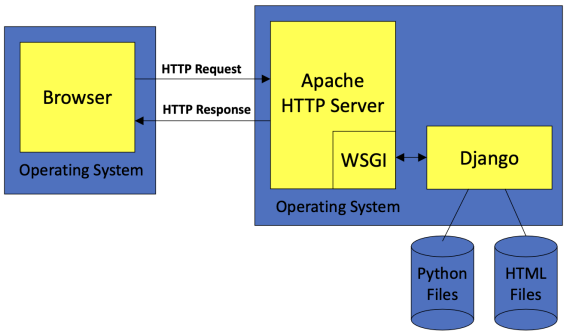


Fig. 3. High Level Block Diagram demonstrating how the deployment of our Web Application on an Apache HTTP Server within an EC2 instance will function to be accessible by users on a web browser

IV. 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 100% 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.

V. 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.

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

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 plan to deploy 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.

VI. System Implementation

Within our design, we have 2 different subsystems. One is the hardware subsystem, where we have the temperature sensor, infrared motion sensor, the wifi remote plug. The other subsystem is the software system, which consists of the AWS EC2 instance and the web application. We will have ESP32 act as the middleware to interact with both systems.

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 via long wires to the ESP32. The reason we are using ESP32 over other microcontrollers is that it is much cheaper than the other options and we don't need much processing power in our proposed system. There will be 2 infrared sensors for each system, and they will both be placed above the doorframe of the user's room. As we are counting the number of occupants in a room, we will be using the infrared sensors to help achieve that. 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. This information will then be processed on the ESP32, keeping track of the number of occupants in that room. 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. We will run a long wire from that space to the breadboard. The temperature sensor then sends temperature data to the ESP32 periodically. 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 the control and monitor the space heaters. This entails powering the plug on or off, and wattage monitoring. If the web application found 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. We will also be utilizing the energy monitoring functionality, which records the wattage usage for that specific heater. This data will be uploaded to the web app via dynamoDB The ESP32 will be our middleware interacting with both the software and the hardware system. After receiving the data from the sensors, it will then send the data to the AWS server. We will be using the dynamoDB database to store the information. For each system, we will create a new entry, and each entry will have the current temperature, the number of occupants in the room, if the heater is powered on or not, and the wattage used this session by the space heater. Every Time of the data points are updated in the ESP, it will be reflected in the dynamoDB. This data is then interpreted on the web application, which in turn powers on or off the space heater.

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 shown in figure 2, we are following the model view controller setup within our 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. Firstly 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 and the heater customization page. Both pages have headers with user ids, links back to the homepage, and the button back to the homepage. The homepage will have a list of all the circuits the user created, and the heaters will be displayed under each respective circuit. The heater customization page will allow the user to change the settings for each heater. First of all, the user can adjust the preferred temperature for that heater. The user can also turn the heater to either manual mode or auto mode. The manual mode will keep the heater on regardless of the room temperature or occupancy status, while the auto mode will let the space heater system take control depending on the thermostat or the occupancy. The user can also customize a schedule for the heater to turn on or off within the page. Finally, the customization page gives the user some statistics on the heater. The amount of wattage the space heater is currently using as well as the past data will be displayed in the customization page. It will also show the number of people in the room at the moment.

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 send it to the templates for displaying in the user's web browser. The controller is the most important part of the model view controller system, as it acts as the master node controlling multiple parts of the system and redirects the user to their desired target. This Django application will communicate with the hardware part by AWS DynamoDB. We will write a javascript program that periodically checks for updates in the dynamoDB table. Then it checks the new data with the user's settings in the django database. If a space heater's power status changed as a result of that data, it will communicate with the shell smart plugs via API calls to turn it on or off. After we complete the Django application, we will upload the system onto an AWS EC2 instance using Apache.

VII. Test, Verification and Validation

The testing of our software will be 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 must verify the fulfillment of user-case requirements which we have theoretically defined, however it is important that we must match the theory with the real life application. To do so, we plan to poll 15 college students that live in off campus housing to gauge the user satisfaction of our product. We will collect 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. This will be critical for us to package the final creation of our service.

A. *Tests for Quantitative performance metrics*

We have defined many quantitative user-requirement metrics for the performance of our product using extrapolations based on gadget specifications and intuitive knowledge of electronics.

The tests we can perform are:

- Meet safe and reliable power output (Wattage) consumption of electricity
- Smart thermostat accuracy by comparing the measured temperature on the smart thermostat and another purchased thermostat
- The rate of the room temperature increasing from the space heater by checking how long it takes for the room to heat up by one degree on the thermostat
- Reduce average energy consumption by at least 15% through maintaining a constant temperature, temperature scheduling, and dealing with inefficiencies from users with different preferences

B. *Tests for Use-Case Specification Software*

As we have learned in our computer science courses, as well as our brief stints within the industry, it is critical that we ensure the verification of software products.

It all first starts off with the individual unit testing of our software services, which will include

- Ability to create/login users and corresponding space heater units on the web application
- If the server runs while in a local development setting

Then afterwards, it will be important test the integration of

- Our developed software within a cloud-based deployed setting
- The integration of our software services with our physical hardware gadgets

Eventually for our entire web application, we will need to round out more rigorous testing such as

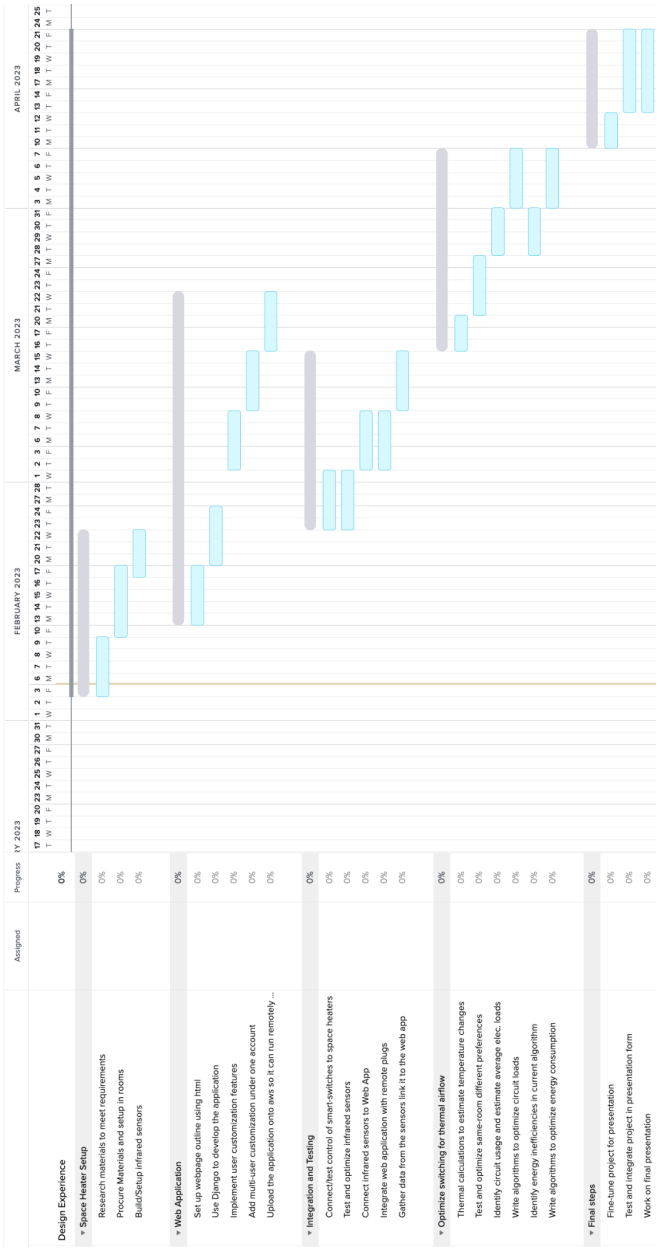
- Load testing our domain with services such as Artillery.io to simulate synthetic load

C. *Tests for Use-Case Specification Hardware*

- We will test our sensors to make sure they meet the requirements we stated earlier. We will test the accuracy and sensitivity of our temperature sensors, and test our motion sensors to see if they can reliably detect motion.
- We will test to make sure our smart plug can reliably turn on and off electricity when receiving HTTP requests.
- We will test to make sure every device can reliably communicate with our web app.
- We will test to make sure all of hardware is reliable and easy to set up for the user.

VIII. Project Management

A. *Schedule*



B. *Team Member Responsibilities*

Eric will focus 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 will focus on the software elements of the project. He will 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. He will be also responsible for the Unit and Integration testing of different software components throughout the application.

Rong will focus 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 will work alongside to help Jay and Eric bridge the gap between the software and hardware. This will mean Rong will need to 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 will be responsible for deploying it onto an Apache HTTP Server that runs on an EC2 instance.

Fig. 4. 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
			Grand Total	220.05

D. *Risk Mitigation Plans*

One huge risk factor is our lack of diversity of our expertise in our ECE Area courses. All three of us took 15440 and 17437, and aside from that we took 18310 and 18270. This may be a huge issue when debugging and dealing with foreign issues that may lie outside our narrow spectrum of courses.

Another risk factor will be the linking of hardware to software. None of us had worked with connecting microcontrollers to a corresponding software system, so this project will be a foreign experience in that aspect.

Lastly, a significant risk factor is the proper testing of our space heater system. The space heater system is intended to be used within poorly insulated rooms in old Pittsburgh houses during the winter time. Given that the expected completion of our project is to be in April, which is far removed from the cold winter time, we will not be able to test the natural diffusion effect of temperature difference without the use of an artificial cold air source like an air conditioner. The testing of this may cause breaker issues due to the wattage load on the electrical outlets. It would also be an extremely costly testing operation as well.

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 will be using Shelly products as a complementary gadget for our Multi-room space heating system. Specifically we will be using the Shelly Smart Plug to regulate and keep track of electrical usage in our system.

X. Summary

Our proposed project is a robust space heater controller system that can help college students with their heating situation at home. By allowing each user to customize their heating preferences in their living space using space heaters, it is a very cost effective solution for students living on a budget. We considered the challenges we personally encountered throughout our college life and tackled them with the knowledge we learned in our ECE classes. Some challenges we might face are the infrared occupancy sensors which might not be 100% accurate, and we might have to run some long wires across the room to complete our intended setup. However, we can overcome these challenges by testing and perfecting our wiring strategies.

Glossary of Acronyms

MQTT – Message Queuing Telemetry Transport
OBD – On-Board Diagnostics
RPi – Raspberry Pi

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.