Can U Cardio?

Authors: Ian Brito, Nataniel Arocho-Nieves, Ian Falcon
Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*—A system capable of providing occupancy and availability of gym cardio machines in real-time. This system is specifically designed for stationary bikes and treadmills. It aims to reduce the time wasted in the gym as well as help students plan cardio workout sessions in a time efficient manner. Incorporating sensors and wireless communications, this system will reflect this data via a web application available to students and staff.**

***Index Terms*—Arduino, Django, IoT, MySQL, NodeMMCU, Occupancy, ReactJS, Sensors, Web Application, Wi-Fi**

1 INTRODUCTION

The life of a CMU student is busy to say the least. As a result, time management is a big priority for many students. However, there are some spaces of their day-to-day lives they cannot really control. One of theses spaces is the gym. Often times gym machines are occupied and a lot of time can be wasted waiting around for machines to become available. Can U Cardio? is a system that helps students optimize the use of their time in the gym by providing real-time occupancy and availability of gym cardio machines. The system employs proximity sensors to detect occupancy which is reflected on a web application available to students. This app could be used on-site during busy gym times, or outside the gym prior to a workout. Thus, our system aids students in planning out their workouts in a time efficient manner that maximizes productivity.

Many occupancy solutions have been developed in the past. However, none of the past projects tackle gym occupancy. They focused on different spaces like dining locations and study areas, among others. Therefore our project provides students occupancy data and information about another specific space. In addition, many projects have used computer vision as the method of occupancy detection. Can U Cardio? utilizes different technologies like physical distance and proximity sensors to ensure a greater degree of accuracy and speed than computer vision techniques. Other solutions have used physical sensors with good results. From what we've gathered, many of these aimed for 1 minute detection latency and over 70% detection accuracy. Thus, we designed our system with the goal of surpassing these metrics and successfully improved upon existing solutions. Overall, Can U Cardio? is a viable and useful tool for busy students and fitness enthusiasts.

2 USE-CASE REQUIREMENTS

With our use case in mind, we established requirements that ensured our system is reliable and efficient for our users. The first requirement is detection accuracy. A system that does not reflect the true occupancy of a gym is not useful. Thus, we set out to have an overall detection accuracy greater than or equal to 90%. First, we needed to properly and consistently identify a single machine as occupied. Therefore, we wanted to fulfill a detection accuracy metric on a per machine basis. In order to fit the overall accuracy, this metric needed to be greater than or equal to 90%. Another component of overall accuracy is the accuracy of the mapping/layout. Our system must be able to display the correct amount of machines available and occupied as well as the correct mapping of each machine based on the gym layout. Thus once more this metric required an accuracy greater than or equal to 90% to satisfy the overall goal.

The second use-case requirement is detection delay. Since this is a real-time system, we must minimize the time it takes to inform a user that a machine is currently occupied. As a result, to justify the on-site use of our system, the time it takes from detecting to updating the occupancy information must be less than or equal to 30 seconds, since this is typically the minimum amount of time it would take a user in a busy gym to roughly scope out the layout and determine which machines are available.

The third use-case pertains to usage time. The University Center gym is open from 6:30am to 11:00pm on weekdays and from 10:00am to 9:00pm on weekends. This means that our system must work continuously for at least 16.5 hours. Ideally, it would also function every day for the specified amount of time with as little intervention as possible. Yet the minimum requirement remains the time the gym is open during the day.

The fourth requirement deals with the invasiveness of the system. Users want no sort of interference throughout their workouts. This interference can take different forms depending on the method for occupancy detection that is employed. Gym members do not want anything interfering with their movements while using the machines. In addition, a gym that is cluttered with wires, possibly affecting walkways and common areas, is not appealing or convenient. Hence our solution needed to be small and compact or employ methods that are not physically invasive on or around the machines. However, with these alternate methods (like cameras), the issue of privacy also comes into play. So, overall our solution needed to be minimally invasive, small, compact, and private.

3 ARCHITECTURE AND/OR PRINCIPLE OF OPERATION

With our use case and use case requirements as centerpieces, we proceeded to design our system with accuracy, speed, durability, and comfortableness for the user as our priorities. As a result, we wanted to have a system that was simple yet efficient to accomplish our goals. For this reason we opted to divide the architecture into two main components: the sensor module, and the software framework.

3.1 Sensor Module

The main question that shaped our design was centered around which method we would use to detect occupancy of the machines. In order to prioritize speed and accuracy our team opted to go for physical distance and proximity sensors mounted on individual machines. Specifically we decided to use a range detection IR sensor that provides reliable and easily manipulable readings to send to a software framework. The chosen sensor was the Sharp GP2Y0A02YK0F IR distance measuring sensor unit with an analog output. Since our sensor is a range detector, it works around the principle of outputting a range of voltages depending on the distance between the sensor and an object within its line of sight.

Now came the question of how to interpret and translate the output of the sensor into data that could be sent and communicated to our user. Since our sensor is analog and outputs a range of voltages, we realized we needed a programmable microcontroller that could process and manipulate this data before sending and transmitting it. We also had to consider which communication protocol we were going to use. We decided to utilize existing frameworks and communicate this data via Wi-Fi in order to simplify the process. Thus, we needed a microcontroller with analog compatibility and Wi-Fi capabilities. As a result, we decided to use the NodeMCU ESP8266 which has an analog port and a built-in Wi-Fi module. This piece is also incredibly cheap and easy to program since it is completely compatible with the Arduino IDE and its libraries, which have a variety of applications and uses. The NMCU is also open source which is convenient if we encounter problems or roadblocks as this would allow us to obtain technical support and customize and modify its operation to meet our requirements and operational metrics.

The final component of our sensor module was the power supply. Both the IR sensor and the NMCU work at similar voltage ranges. Nonetheless, the deciding component for our power supply is the IR sensor which has a more restricted range of voltages it operates under than the NMCU. The sensor operates anywhere between 4.5V to 5.5V. Thus, we decided that a 5V power supply is best in this case. With modularity and size in mind, we decided to draw power from batteries. We opted to utilized high capacity rechargeable Ni-MH batteries.

3.2 Software Stack

With the architecture of sensor module in place, we shifted our focus on developing a way to receive the data via Wi-Fi, store it, and display it for the users. Therefore we needed some form of database framework to sort out the data from each sensor module. In addition we needed an appealing visual display on the front end to display the occupancy data.

So, we conceptualized a software stack that would fit these needs. The stack is comprised of an AWS EC2 instance running a Django web application using Python and ReactJS, as well as MySQL.

The web application is a Django web application created with Python hosted on an Apache web server using EC2. Django provides a secure framework for creating web applications, and EC2 is a secure and highly configurable method of deploying the Django web app. The front-end of the web app is designed using ReactJS, and AJAX is used in order to automatically update the web page to display up-to-date information.

In order to receive information, the web application receives a JSON from the sensor module via a POST request. The JSON contains information about which equipment the POST request is coming from, such as an ID, whether the equipment is free or busy, the time the request is sent, and a way to determine that the post request was indeed made by one of the sensors. Once the data is received and verified, the web application is updated and the data is stored in the MySQL database. Although the default Django database of SQLite works with EC2, MySQL is more scalable under load and provides more robustness compared to SQLite.

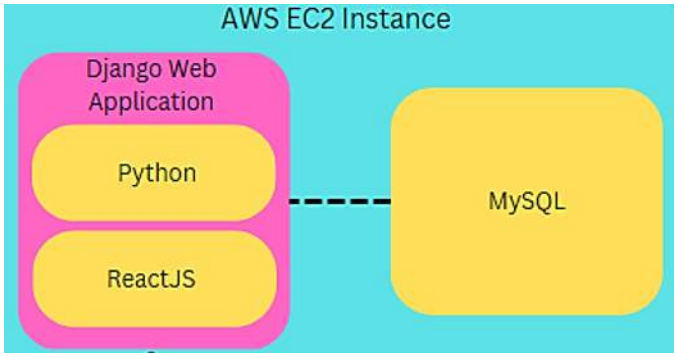

Figure 1: The Software Stack, consisting of an EC2 instance hosting the Django web app and MySQL

4 DESIGN REQUIREMENTS

Given the architecture of our system and the chosen mechanism for occupancy detection, a couple design considerations come into play. It was required to establish requirements for our sensor module as well as the overall system including the web app and AWS framework.

In terms of the sensor module, based on the machines

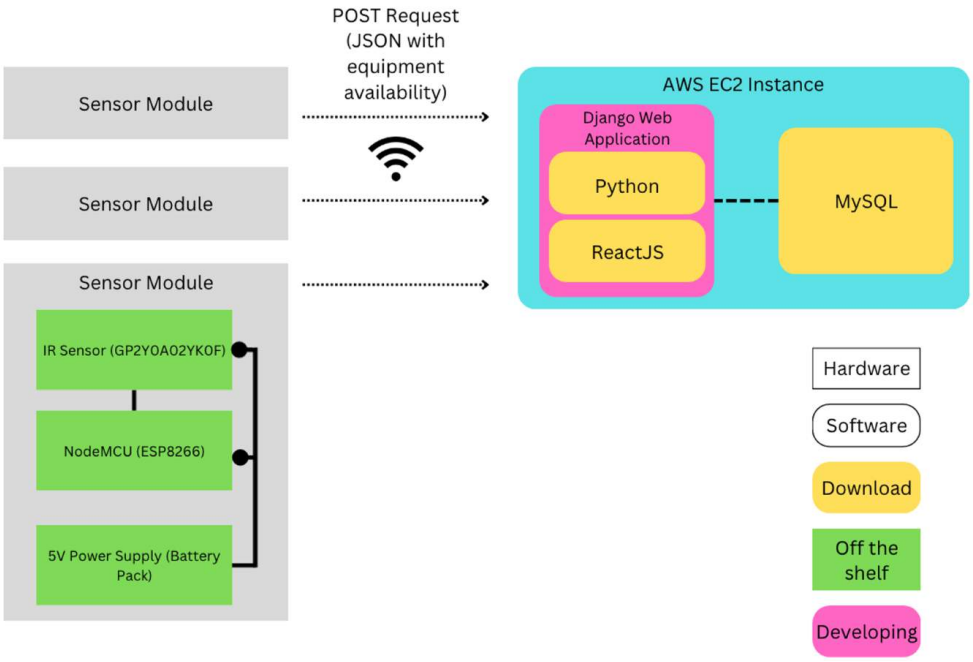

Figure 2: Block diagram depicting connections and data flow of the system.

currently in use at the UC gym, we know that the farthest detection range we will be dealing with comes from treadmills. In other words the largest distance between sensor mounting points on the control panel of the machine and the user will be on the treadmills. It has a track with a total running length of 5ft. Given that its dashboard console is directly above the starting point of this track, it is safe to assume that our users will be running anywhere from 1ft to 5ft from our module. Thus, our sensor must have a detection range between 0.5ft (= 6" ≈ 15cm) to 5ft (= 60" ≈ 152cm). The upper bound of this range was instrumental for the detection accuracy on treadmills, since we did not want to mark the machine as occupied if someone passed in front of it farther than 5ft from the control panel where the sensor will be mounted.

With detection accuracy in mind, we needed a programmable microcontroller that allowed us to manipulate the sensor data and fine tune it based on the type of equipment the sensor module is installed on. This is especially important when detecting occupancy in stationary bicycles. Given that our sensor will have a maximum range of 5ft, more detailed and meticulous control will be needed when working with bike sensor modules since the bike itself is no longer than 105cm (≈ 41.5" ≈ 3.5ft) according to the manufacturer (LifeFitness). The distance between the control panel of the bike or the handles (which are all possible mounting points for the module) and where a user will typically be seated is anywhere between 0.5ft to 2ft from the sensor module mounting points. As a result, we needed to establish a specific range within the one our sensor provides in order to avoid the same problem described above in the case of the treadmill. We did not want mark a bike as occupied unless someone is actually sitting on it rather than passing by.

The detection delay use case requirement translates into a latency requirement for our system. As a result, our overall system needed to have a latency of less than 30 seconds. This overall latency is composed by sensor latency, NMCU latency, and AWS-MySQL-web app latency. Given that the latency for our chosen sensor is around the millisecond scale, the overall latency was primarily influenced by the efficiency of the wireless (Wi-Fi) communication between the modules and the server as well as the efficiency of the data sorting program within the web application.

Opting for batteries reduced the invasiveness of our sensor module reducing wires and connections, making it easily mountable to the gym machines selected. With this design choice, we also factored in the usage time use-case requirement. In consequence, we determined that our sensor module composed of the IR sensor and the NMCU needed to have a battery life of at least 16.5 hours to last through the a whole day of gym operation. From inspecting the documentation on each component we used, we knew that the IR Sensor can take between 4.5 V to 5.5V of supply voltage and was expected to operate at 33mA. The NMCU can take anywhere from 4.5V up to 10V with a recommended operating voltage of 5V and was expected to operate at around 80mA. The combined current drawn is 113mA. Therefore, we needed a 5V battery with a capacity of roughly 1,864.5mAh (113mA times 16.5 hours).

5 DESIGN TRADE STUDIES

While envisioning our system, we identified two key areas that influenced our design and decided to perform some

research in order to determine which approach and which technologies would best fulfil our requirements.

5.1 Method of Detecting Occupancy

There are a variety of methods utilized to detect occupancy. Based on our detection accuracy requirement of 90% and our detection delay requirement of 30 seconds or less, we explored two main options of detecting whether the gym machines were being used or not.

5.1.1 Computer Vision

Our initial thought was to use cameras, paired with computer vision algorithms to detect occupancy. Nonetheless, we noted significant concerns with this approach. One of them is the accuracy of the readings. With a single camera, depth, perspective, and resolution become issues. The more muddled the picture becomes, especially during peak hours of gym usage, the lower the accuracy of the readings would be, which in turn would defeat the whole purpose of our system. These concerns paired with the fact that, depending on the size of the gym, there might not be an optimal position for a single camera to record the whole room in a single frame, indicated that multiple cameras were needed to implement our system. As a result, we were facing an added a layer of cost and complexity that in our view was avoidable.

In addition, this also brings up the issue of the privacy of the users. Many gym members might not be comfortable with more cameras recording their workouts aside from the ones already employed by the establishment for security purposes. While it would be ideal to access these existing feeds, its safe to say that the process is complicated and the gym owners/managers might not be content with sharing recordings, which makes the task of retrofitting existing cameras in the gym quite bothersome and unclear. For this reason we decided to explore other simpler, more accurate, less invasive, yet easily scalable options.

5.1.2 Physical Proximity Sensors

Our other line of thought was to use proximity sensors in order to detect occupancy. This approach certainly addressed the issue of privacy, keeping the user anonymous in terms of occupancy metrics. These sensors allowed for contactless sensing, making the system relatively non-invasive. When it comes to accuracy, such sensors allowed for more accurate readings given that the idea was to individually mount them on each cardio machine. However, the mechanism and working principle of such sensor was also something to be considered and analyzed. For the scope of our application we considered two types of proximity sensors.

- Ultrasonic Sensor: This type of sensor has advantages like low current consumption which is good for battery life. In addition, the object detection is not affected by color or transparency. Nonetheless, when detecting objects that are soft or have extreme textures this sensor is not suitable. The sensor's readings are sensitive to sound, which could be an issue in a crowded and busy gym. Another aspect to consider is the range, which in this case exceeds our requirements by a great margin (typical max of 400cm $\approx$ 13.2ft). A lower detection range helps with detection accuracy since the sensor itself has a detection limit closer to our standard. In addition, ultrasonic sensors usually have wide angles of detection, which is not ideal for spaces like our gym. Given that the machines are placed consecutively besides each other, dealing with a sensor that has wide angle of detection is cumbersome and could lower detection accuracy.

- IR Sensor: This sensor draws more current which reduces battery life. In addition, it is more sensitive to its environment and object detection is affected by color and transparency. However, the UC gym is well lit at night and not too bright by day so light sensitivity is not a concern. In addition, IR sensors detect a larger variety of textures with ease. Both IR and ultrasonic sensors are usually distance sensors. However, IR sensors have less accurate distance readings than ultrasonic ones. Nonetheless, our system does not necessitate an accurate and precise distance measurement. We merely need to detect if something is there or not, which IR sensors are more than adequate to do. Moreover, due to the physical principle that drives IR sensors (light vs sound), IR sensors are faster than ultrasonic ones, which is good for our detection delay. IR sensor outputs are easily condensed to a single output voltage pin, which is easier to manipulate than the 2 output trig and echo pins typically associated with ultrasonic sensors.

Given the information compiled and its subsequent analysis, we decided to utilize IR proximity sensors since, despite some drawbacks like cost and power consumption, the levels of detection accuracy and reliability they provide justify its selection over ultrasonic sensors and most definitely over computer vision occupancy detection mechanisms.

5.2 Hosting Web Server

5.2.1 Raspberry Pi 4B

Initially, we envisioned using a Raspberry Pi 4B to host the web application. The main benefit of using an RPi is the lower cost due to not having to pay recurring fees for AWS. However, hosting the web application on an RPi would mean we would need access to CMU's router in order to allow port-forwarding. Additionally, the RPi is more vulnerable to attacks such as DDoS, as well as physical attacks that could damage the RPi. Finally, a single RPi is not too scalable, as the more requests the RPi would receive, the more it would slow down. Clearly, this is not a viable option.

5.2.2 Amazon EC2

Instead, hosting was done with an Apache web server utilizing Amazon EC2. The main benefits of EC2 compared to hosting on the RPi are security and accessibility. In terms of security, EC2 is far more secure than hosting on an RPi given that AWS has various different security tools compared to an out-of-the-box RPi. Furthermore, unlike EC2, the RPi is prone to being physically damaged or suffering from a power outage, which would shut down the website. On the other hand, EC2 is far less likely to experience these risks and therefore much more reliable. Additionally, the aforementioned cost for EC2 is mitigated due to use of the free tier, which allows 750 monthly hours of a t2.micro instance which is sufficient for this project. Therefore, the use-case requirement of usage time is met as this system will continuously run as opposed to the RPi which is more likely to experience interruptions.

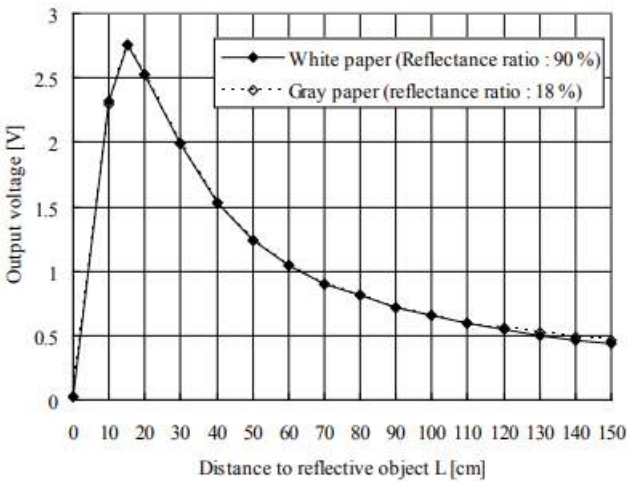

Figure 3: Output Voltage vs Distance graph from Sharp IR GP2Y0A02YK0F datasheet

6 SYSTEM IMPLEMENTATION

6.1 Sensor Module

The main component of our sensor module is, of course, the IR sensor. The chosen sensor is the Sharp GP2Y0A02YK0F IR distance measuring sensor unit. It has three connecting wires: a wire for Vcc or supply voltage, one for GND or ground, and one for Vo or the output voltage. The next component of the module is the NMCU. We are using the NMCU 1.0 version. This board comes with an ESP8266 Wi-Fi microchip, a CP2102 serial chip, and is programmable using the Arduino IDE. To setup the NMCU we downloaded some drivers to detect the port connection of the board (CP2102 serial chip) to a computer via MicroUSB and installed the ESP8266 board manager in order to upload our programming to the board.

Recall that our sensor is analog, so we connected its output to the A0 analog pin of the NMCU. Once connected to the NMCU, this input is processed by a built-in 10-bit resolution analog to digital converter. The converter transforms our input into an ADC reading with a resolution of 5V/1024 or 4.88mV. In other words, the ADC outputs a number from 0 to 1024 that is linear to the voltage from the IR sensor, where 5V is the maximum voltage corresponding to the max ADC reading. Using this number, we manipulated our NMCU loop to set detection thresholds based on voltage, ADC reading, and distance. While the relationship between voltage and ADC is linear, the relationship between voltage and distance was more complex and obtained from a manufacturer supplied graph in the IR sensor datasheet. So the data flow is distance → voltage → ADC reading, and that last one is use to calibrate detection depending on the machine (bike or treadmill).

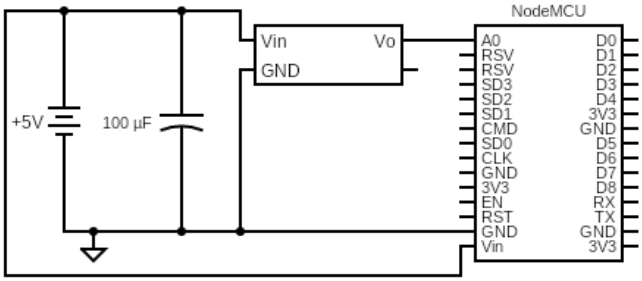

Figure 4: Sensor, NMCU, and battery wiring diagram

Given our battery life and modularity requirement from section 4, we opted to power the whole module using rechargeable high capacity Ni-MH AA batteries in a 4-cell switchable battery holder that supplies both the NMCU and the sensor. The chosen batteries are manufactured by Hi-Quick and each supply 1.2V and carry 2,800mAh. Thus we utilized 4 batteries per module totaling 4.8V which is in the range of operation of both components The IR sensor and the NMCU. Each module has a total of approximately 11,200mAh, meaning that a single module will most likely be operational for 6 days if interrupted (turned off at the end of the 16.5 hour schedule of the gym) or 4 days uninterrupted. All these components are wired and soldered to a small breadboard PCB. In parallel to the battery back and the IR sensor a 100μF capacitor was added to stabilize the power supply line coming into the sensor. The fully assembled module is housed inside a 3D-printed encasing that was designed using Solidworks, exported as an STL file, and printed utilizing the UltiMaker Cura software compatible with the Creality Ender-3 printers available at the Techspark spaces on campus. The 3D printer utilizes standard PLA for fused deposition molding set with a layer height of 0.2mm and an infill density of 20%.

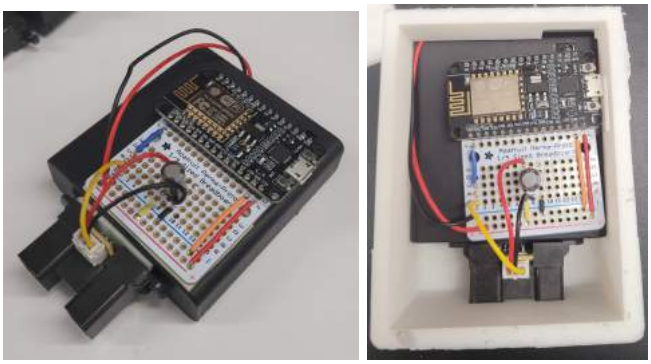

Figure 5: A single sensor module

Figure 6: Treadmill sensor module encasing

Figure 7: Bike sensor module encasing

Figure 8: Machine dashboards with highlighted sensor module mounting positions

Even though the same 3D template was used for all modules, specific changes were made depending on whether the sensor was intended to be mounted on a bike or a treadmill. The bike module encasing is lifted up at an angle (approximately 35.6° with respect to the inclined bike dashboard and around 25° with respect to ground level) to account for the fact that the dashboard mounting position is directly in front and lower than the bike handlebars. Thus we needed to raise the sensor to a range over the handlebar to detect a user.

This alteration also helped with detection accuracy since the angle was suitable enough to detect shorter users and it reduced the error that would be present in a completely level (parallel to ground) measurement. These encased modules are placed on top of the gym equipment dashboard parallel and below the central console of the machine. Given that the people in charge of gym operations would need to access the battery pack, we decided to not fix the module in place and make it removable, essentially sacrificing the safety of the module in order to obtain better readings and make recharging more convenient. Thus our mounting mechanism is not efficient nor realistically sustainable. A proper solution would require more secure fasteners that would directly tamper with the dashboard and console, but we are not allowed to do this. The optimal solution is to have the sensor powered by the machine and integrated into the console so that we eliminate the recharging hassle and reduce the risk of people tampering with the sensor while also ensuring its continual use.

6.2 Software Stack

The Software Stack contains the EC2 instance that the web application and MySQL is running on, the Django web application (Python, React), and MySQL.

First, the Django web app follows the MVC (Model, View, Controller) convention. The *Models* are in models.py and contain the data stored in the database. In our case, the primary model we will be using is an Equipment model, where each equipment would have once instance of the model stored in the MySQL database. The fields of this model include:

- status: The equipment's current status, which can be "free", "busy", or "down"
- machine: The type of equipment the machine is, "bicycle" or "treadmill". Defaults to "treadmill". This field can be changed on the admin page.
- mac_addr: The MAC address of the nodeMCU that was used to register the device
- id: Django's built-in, automatic field which serves as the primary key
- internal_id: An identifier used for mapping equipment on the home page. This can be changed on the admin page.

Other models include Django's built-in User model, which is used to log in to the admin page, and an Hour model, which is used to compute the average occupancy per hour.

Next, the *View* is comprised of the static HTML templates presented. The View is what the user sees displayed on their device when visiting the web application. Moreover, the View is updated by the Controller to display updated information.

Finally, the *Controller* processes each request made to the web application. For instance, Django's urls.py specifies which action in views.py to display given the url visited, which is how www.canucardio.com/home knows how to display the home page and www.canucardio.com/info knows how to display the information page. Django's aforementioned views.py specifies which template to render. Actions in views.py, when directed from urls.py, are passed in the request made, so different actions can be done depending on if a "GET" or "POST" request has been made. Additionally, the *Controller* is responsible for any calculations or processing required, such as calculating the average occupancy per hour.

Therefore, our Django web app implements the following:

- urls.py – Specifies which action to run depending on the URL a request is made to
- views.py – Contains the actions that render the static HTML web pages with the render() function, providing a context dictionary
- models.py – Contains the model objects used to map to database (MySQL) via Object Relational Mapping (ORM)
- Static HTML files – static HTML files contain what is actually displayed to the user
- Static JS files – the JavaScript files will be required to implement both the front-end with ReactJS as well as AJAX to allow for automatic refreshing of the data

Next, the Software Stack runs on an Apache web server on an EC2 instance. For this project, the t2.micro is being used as it is part of the free tier and provides 750 hours of monthly use, which we have determined to be enough for our use case.

Finally, the way the Sensor Module contacts the web application is via a POST request made. There are two phases for the microcontroller - "setup" and "update". In the setup phase, the NodeMCU makes a POST request to 'http://canucardio.com/register', sending a JSON with the following information:

- key: A secret token used to verify that the POST request is coming from the nodeMCU and not an outside source
- mac_addr: The NodeMCU's MAC address

The 'register' endpoint checks if the supplied key is valid. If not, a 404 Response is returned. If the key is valid, the web app checks if there is an existing Equipment object with the supplied MAC address. If so, the response encodes the existing object's id as a JSON. If not, the web app creates a new instance of the Equipment object, and the response encodes the new object's id as a JSON.

AJAX is used to asynchronously update the data presented on the home page. If a machine's status is updated, the web app will automatically reflect the change without the user needing to refresh the page.

The 'home' page illustrates a map of the Cohon University Center's gym, with icons indicating whether a given machine is free or busy. The 'info' page presents the average occupancy per hour at the gym in order for users to determine when the optimal time to go to the gym may be. The 'report' page allows users to report a machine as down or broken, which is reflected on the main page. Finally, the 'admin' page allows a gym administrator to edit machines' internal ids, which allows them to edit the machines' locations on the home page, switch a machine between 'treadmill' and 'bicycle', and delete machines.

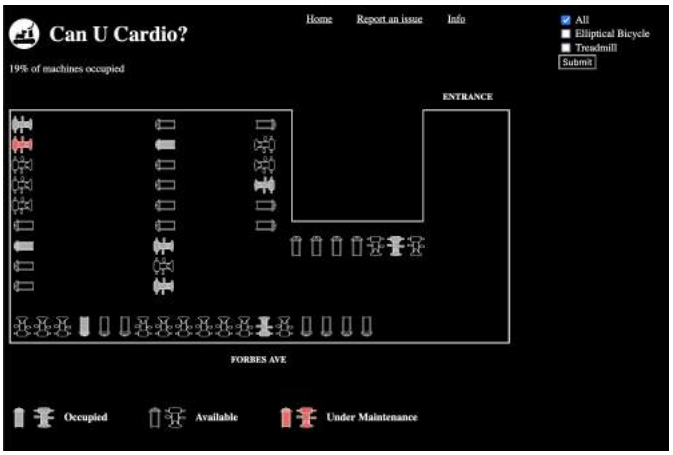

Figure 9: The home page for the Can U Cardio? website

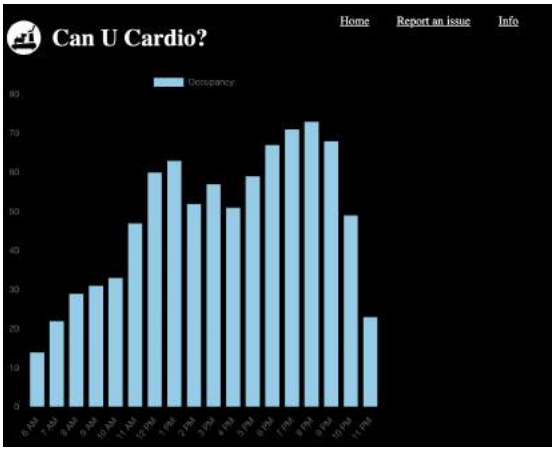

Figure 10: The info page, which displays average occupancy per hour

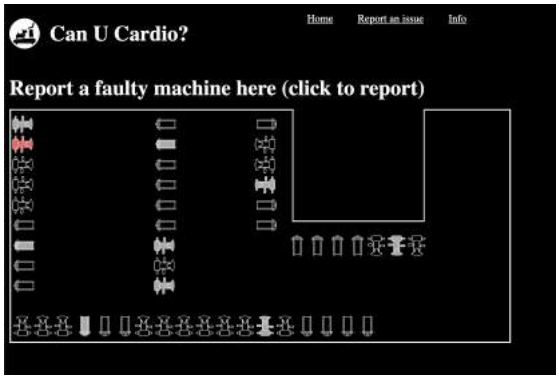

Figure 11: The report page, where users can click a machine and submit a report that marks the machine as "down"

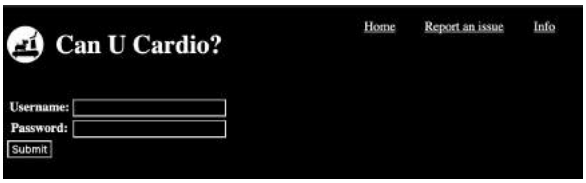

Figure 12: The login page for admin

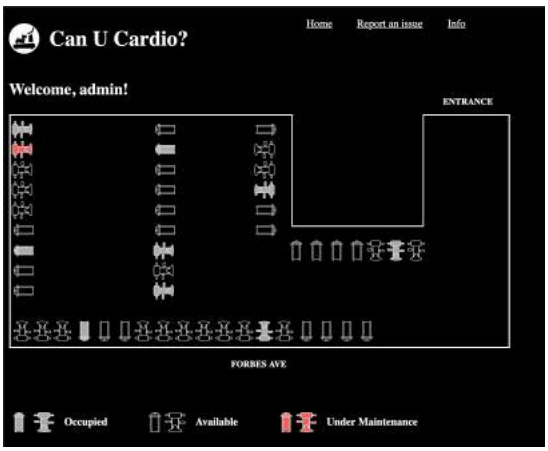

Figure 13: The admin page, where the administrator can click on machines for advanced options

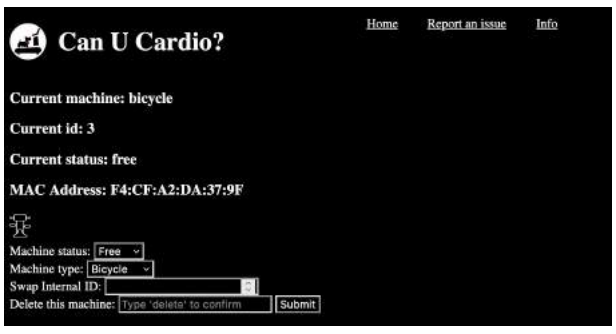

Figure 14: Admin's advanced machine settings

7 TEST & VALIDATION

To test and validate our design we created separate testing phases for each subsystem. Each phase aims to verify and validate our 4 use-case and design requirements: detection accuracy, detection delay, battery life, and invasiveness.

7.1 Results for IR Sensor Thresholds

Before thinking about our requirements we first had to verify that our chosen components worked as intended. As a result, the first set of test and trials we performed were centered around the IR sensor. Our aim with this phase was to understand how the sensor behaved and verify if the manufacturer's claims about distance versus output voltage were true, which would in turn determine if these were viable options for our design. Thus we first tested the sensors in the lab before integrating them with the NMCU. The setup for the initial round of testing was informal and mostly qualitative. We powered our sensor with a DC power supply set to 4.8V and connected a 100μF capacitor in parallel to stabilize the power supply line. We connected an LED to the output voltage and ground, and observed the voltage drop across it with a voltmeter. The sensor was functional as evidenced by the fluctuating voltage as we varied the distance as well as by the dimming of the LED.

Knowing that our sensor was qualitatively behaving as intended, we proceeded with the next phase and gathered quantitative data. For this round, the setup was a circuit very similar to the one depicted in Figure 4, with the exception that a lab power supply set at 4.8V was used rather than a battery pack. In addition, we connected a voltmeter in parallel between Vo (and A0 pin) and ground (GND) to record the output voltage of the sensor. On the NMCU front, a very simple Arduino script was developed using the built-in ADC function to obtain a converted number from the output voltage and display it in the serial monitor and plotter from the IDE. For each trial, we stood at a specific distance from the sensor according to a measuring tape we had configured on the floor. We tabulated the distance, the output voltage observed on the meter, and the ADC reading displayed by the monitor to produce the graphs depicted in Figures 15-17. Due to reading fluctuations, we recorded the highest and lowest bound of voltage and ADC that was observed.

Although not identical, the graph we produced from our measurements is very similar to the one provided by the manufacturer. The shape is the same yet the numbers varied quite a bit. Regardless, we were happy with our results since the ADC provided sufficient granularity for the purpose of our application. In addition, we correlated the ADC reading to the voltage and saw that it behaved pretty much linearly which was also something that we expected. Thus, this phase of testing served as the basis for our detection scheme and our results were used throughout the calibration process of the sensor module.

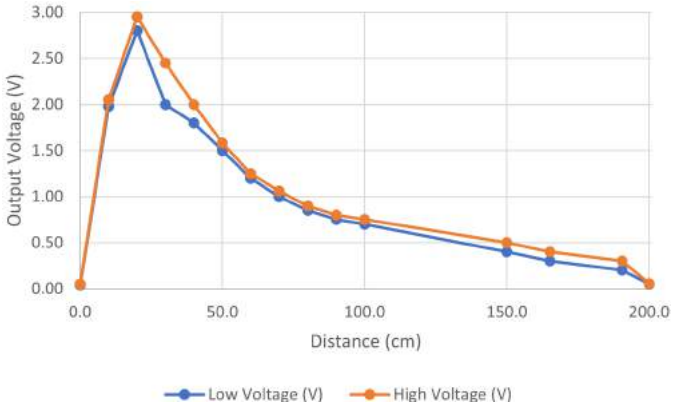

Figure 15: Output Voltage (V) vs Distance(cm) from IR Testing

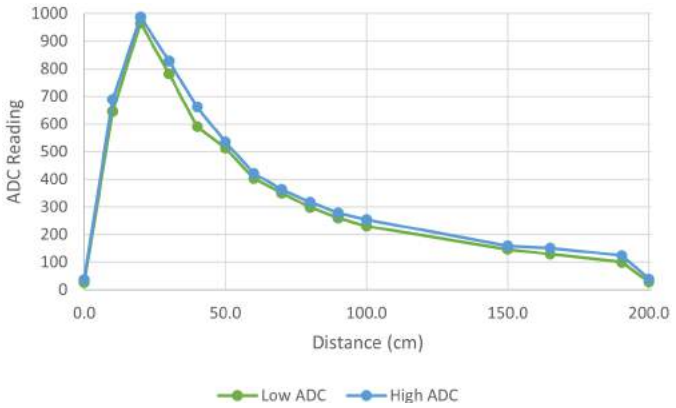

Figure 16: ADC Reading vs Distance(cm) from IR Testing

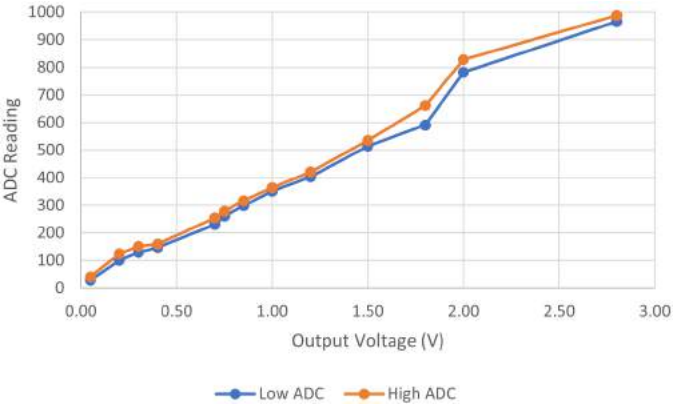

Figure 17: ADC Reading vs Output Voltage (V)) from IR Testing

7.2 Results for Detection Accuracy

With our gathered sensor data and after we were able to create a mobile module like the one depicted in Figure 5, we went to the gym and started our next phase of testing which was divided into two parts. First, we wanted to see if the module was working properly independent of the web app. So we tweaked our IR sensor testing script to employ the built-in LED of the NMCU as a visual reference for detection. With our measurements at hand we set initial ADC thresholds based on the physical dimensions of the gym equipment. Thus, this set of trails consisted of using the LED as a detection and occupancy indicator. The modules were placed in the mounting spots shown in Figure 8 and the testing outcomes were very simple: if the person was within range then the LED would light up, if not the LED would stay off. From this first part we tweaked the ADC thresholds and calibrated our sensor.

The next set of testing involved a fully integrated system with module and web-app both working. We took the script from the previous phase and modified it to incorporate the communication protocol between the NMCU and the web-app. This time we ran rigorous trials with a final product in mind. Here, passing test results were observed when the built-in LED lit up as expected and the same was reflected in the web-app. So, the first set of tests involved on-range usage of the machines. We wanted to validate that the a person using the machine was actually detected and the occupancy matched it on the web-app. The second set of trials was out of range stand ins but within the line of sight of our sensor. Here, we wanted to simulate a person standing in front of the machine but not actually using it. The third set, involved pass-bys out of range of the machine that mimicked people walking by through the main corridors and walkways of the floor plan. The fourth and final set of test involved standing in between machines and its surroundings to see if there was no wide angle detection.

Each set consisted of 10 roughly minute long trials. So, in total we ran 160 trials: 4 of each type with each of our 4 sensor modules. We utilize the data from these trials to determine the accuracy of each sensor and furthermore of the entire system. The first set of trials was successful in all modules. We observed detection in all cases we anted to observe.

For the second set of trials, we only saw a failed case (false positive) for a single bike module. This failure could be explained by the fact that we were standing quite close to what we considered to be the detection boundary near the seat of the bike. Contrary to the treadmill, people can stand at this boundary where as in a treadmill the boundary is much more defined by the actual physical architecture of the machine. In the treadmill the boundary is quite literally a line, whereas in the bike you could stand at the boundary but still lean in causing the sensor to detect you. However we must acknowledge that some boundary tests we performed were unrealistic or at least very uncommon postures or usages of the bikes. We leaned back on the bike at unreasonable usage angles since we wanted to see how much overlap between a leaned user and an out of range stand-in there was. There was indeed overlap. However based on the common use of gym machines nobody typically stands in so close to the machine as we did, while almost nobody leans back as far as we did. Hence the testing was not ideal at boundary condition, but still

acceptable for the purpose of our system.

For the third set, we saw no problem with pass-bys across all sensors. However, much like with the previous set, unreasonable pass-bys were conducted very close if not nearly on top of the machines near the boundary and some tests were faulty. Nonetheless, once again with reasonable usage in mind, pass-bys were also a success since we did not detect users when we did not wanted to (no false positives). For the fourth case, treadmills were not an issue due to the handles they have at each side. It was practically impossible to be detected at a wide angle if you were not standing on the running surface. Thus, standing and passing in between machines revealed no false positives. Once again, in the case of the bike reasonable stand-ins and pass-bys at a wide angle produced no false positives. We did however push the boundary, yet we observed that you needed to be directly in the line of sight (unreasonably leaning in from the side) in order to be detected. Thus this set of trials were also successful and also validated our design choice for IR sensors on the basis of its narrow detection angle.

After individual module trials, we tested different configurations of busy and free across sensors and saw that thanks to our initial setup, the web-app marked the proper icon in the layout based on the sensors that was detecting a user. In other word our detection mapping was successful and behaved as configured with maximum accuracy.

Thus with the results tabulated, we calculated that sensors 1 and 2 (treadmill sensors) displayed a detection accuracy of 100% each while sensors 3 and 4 (bike sensors) were 97.5% accurate. Overall our 4 module system ended up with a detection accuracy of 98.8% which exceeded our detection accuracy requirement of an accuracy larger than 90%

7.3 Results for Detection Delay

The delay we set out to test in this phase was the overall system delay. which is simply the delay from when the sensor detects a user to when the updated data is displayed on the website. For these trials, we utilized the built-in LED of the NMCU and manually timed how long each update took. In other words, we measured the time from light up of the LED to light up of the icon in the web-app.

For each of the sensor modules, we tested setting the sensor from "free" to "busy" and recorded the time it took the web app to reflect the updated status. Next, we did the same, except setting the sensor from "busy" to "free". The "overall" column is simply an average of the "Free to Busy" column and the "Busy to Free" column. 10 trials were run on each sensor module for this test, for a total of 40 trials.

The max average detection delay for one sensor was 3.62 seconds. The max single delay recorded by one of the sensors was 6.75 seconds. Taking the average of all these delays we would calculated that typical and overall delay is 3.19 seconds. Given that our use-case requirement was less than 30 seconds, these results met our expectations.

	# of successful trials out of 10			
Sensor Module #	On-range stand-in	Out-of-range stand-in	Pass-by	Surroundings (wide angle)
1 (treadmill)	10	10	10	10
2 (treadmill)	10	10	10	10
3 (bike)	10	10	10	10
4 (bike)	10	9	10	10

Figure 18: Detection Accuracy Results

	Avg detection delays out of 10 trials (s)		
Sensor Module #	Free to Busy	Busy to Free	Overall (avg of BF and FB)
1	2.37	4.77	3.57
2	4.30	2.93	3.62
3	1.61	3.46	2.55
4	3.92	2.12	3.02

Figure 19: Detection Delay Results

7.4 Results for Battery Life

For this portion of our testing, no extensive and prolonged trials observing battery life were conducted due to how inconvenient they were in terms of schedule and logistics. Given these circumstances we wrote down measurements from which we estimated our battery life. These measurements were obtained in the lab using our components including IR sensor, NMCU, and battery pack wired in the manner depicted in Figure 4. Nonetheless, given that we needed to measure the current drawn by each component, we utilized a breadboard that would allow us to connect an ammeter in series with our components and manipulate the circuit layout. With a fully soldered module, connecting in series was not really possible.

As a result, we used spare components for this. Before measuring we validated that the components were working as intended runnning similar tetst as the ones described in previous sections while. Thus, we calibrated the modules in the same way our mobile modules were. With this out of the way, we measured the current drawn by the IR sensor to be quite stable at 27mA. In the case of the NMCU, we saw 80mA drawn while booting up. However, during operation the reading fluctuated from 25mA to 80mA with occasional readings of 40mA. In light of this, we decided to calculate our battery life based on the worse case sce-

nario measurements. So, given that our components are connected in parallel effectively dividing the current drawn from the battery pack, the total current consumption of our module is the sum of the individual currents. Hence the total consumption is 107mA.

Each cell in our battery pack has a capacity of 2,800mAh, so the whole pack contains 11,200mAh. Dividing the overall capacity by the current drawn by our module we obtain around 104.67 hours of use which translates to approximately 4 days, 8 hours, and 40 minutes of use. This calculation assumes that the sensor is constantly drawing current, meaning that it is operating continuously. However, our battery pack has a switch that allows you to turn off the module. Thus, if the module was turned off during off hours then we could extend the battery life of the module. The UC gym operating periods are typically 16.5 hours long during weekdays. If we round this up to 17 hours to account for opening and closing times, we have that 107mA drawn for 17 hours equates to a consumption of 1,819mAh per day. In consequence, if we divide our total capacity by the per day usage which we assume is the same for every day (despite this not being the case on the weekends) we get that our module could be operational for 6.157 days or approximately 6 days, 3 hours, and 46 minutes. Thus our battery life surpasses our requirement of 16.5 hours of use, yet we believe it would have been a great milestone to reach 7 days of continuous use without recharge to eliminate as much hassle as we can for the administrative staff. Nonetheless, our results exceeded what we planned and designed for

8 PROJECT MANAGEMENT

8.1 Schedule

In terms of the schedule, Ian Brito was responsible for the software stack, while Ian Falcon and Nataniel were responsible for the sensor module. The main changes in schedule arose from not having planned enough tasks originally. Comparing the original schedule from early in the semester to the final schedule, each individual has double the tasks that we originally envisioned. Additionally, there was a design shift from using a Raspberry Pi to EC2, which shifted the schedule. We experienced delivery delays with the IR sensor, which pushed our schedule back. Finally, certain tasks simply took longer than expected, such as work on the front end of the web application. The schedule is shown in Fig. 21.

8.2 Team Member Responsibilities

Ian Brito worked on the software stack, so his primary responsibilities were everything involving the web app portion of the project. Responsibilities included creating the Django web app, working on both the frontend (visuals to display map of UC gym) and backend (store models, receive data from nodeMCU), creating the EC2 instance and deploying Apache web server. Ian Falcon and Nat worked on the sensor module. Ian Falcon worked with the signal and sensor portion of the sensor module, such as connecting the sensor to the microcontroller, writing code to test the sensor's distance, power consumption, and encasing for the sensor module. Nat worked on the nodeMCU code to send POST requests from the sensor module to the web app.

8.3 Bill of Materials and Budget

Please refer to Table 1 to obtained a detailed breakdown of parts and costs.

8.4 Risk Management

First, our initial plan was to upgrade from the t2.micro EC2 instance to a paid tier if the free tier did not meet our needs for the web app. Fortunately, we did not need to use this, but we had space allocated in our budget if needed. Another alternative was Heroku, but EC2 deployment worked fine. Next, we had researched other sensors in case our desired Sharp IR sensors did not ship in time. Although they were back ordered for a while, we eventually received them without needing to use our backup plan. Thankfully, everything went according to plan, but in the event that something went astray, we had backup plans.

9 ETHICAL ISSUES

In terms of physical harm, our system does not really pose a threat or risk since the main physically accessible part to our users is the sensor module, which we envision will not be invasive and will be small in size. On the other hand, the main vulnerabilities of our system are of a virtual kind (communication between sensor module and network, general safety risks of the internet, among others).

If somebody can get access to at least one of the nodeMCU that will be connected to the gym's WiFi, they could potentially get the credentials of that WiFi. From there, they could perform malicious actions while connected to the WiFi network that they could not perform otherwise without the credentials taken from our product. Suppose the gym has a private WiFi only for staff, which the nodeMCU are also connected on. A malicious actor can gain access to this WiFi by stealing the nodeMCU and performing nefarious activities that they are not meant to.

As mentioned above, if a malicious agent were to gain access to a network they're not supposed to have access to, a worse scenario could unfold if the gym facilities use a single Wi-Fi network throughout. Thus the perpetrator could obtain data from the gym and the staff as well as gain access to the customers' devices that are connected to said network, essentially creating vulnerabilities akin to those you encounter with public Wi-Fi networks.

The person who gets harmed in this worst case scenario would have to be the individual user. While no physical harm is done, a malicious agent getting access to a user's

Table 1: Bill of materials (* denotes items added since Design Report)

Description	Model #	Manufacturer	Quantity	Cost @	Total
AA Rechargable Batteries	HQAA2800	HiQuick	16 (one 16-pack)	$2.31	$36.99
* Quarter-Sized Breadboard PCB		Adafruit	6 (two 3-packs)	$4.00	$23.98
NodeMCU Board	1.0 ESP8266	NodeMCU	6 (two 3-packs)	$4.56	$27.36
* 4-cell AA Battery Holder with Switch	4AA-Switch-2P-A	LAMPVPATH	4 (two 2-packs)	$3.75	$15.00
Infared Proximity Sensor	GP2Y0A02YK0F	Sharp	4 (two 2-packs)	$9.44	$37.76
* 16-Bay AA Ni-MH Battery Charger	MY-C006	Bonai	1	$30.99	$30.99
* 140 piece Jumper Wire Kit	WK1	QSU	1	$6.49	$6.49
* AWS Credits (out-of-pocket)		Amazon	1	$13.00	$13.00
					$191.57

personal devices can lead to breaches that compromise the mobile devices themselves as well as personal data like login credentials and other sensitive data. This could potentially result in serious consequences ranging from identity theft to users losing access to other critical personal resources like bank accounts, among other such sensitive things.

In reality, anyone that uses our system and has devices (capable of connecting to the internet) with connections to personal information is at risk in this case. Thus, there is not a particular kind of person that is vulnerable above any other since basically all individual users/customers are vulnerable as well as the gym facilities and staff.

The ethical concepts of privacy and trust would absolutely be violated in the above scenario. Users losing access to their personal data would absolutely be a breach of privacy. Knowing that this loss of personal data came from our web app would feel like a breach of trust in addition to this. In addition, the responsibility of making the system secure and impervious to different scenarios would fall on us and many people would hold us accountable for damages like the ones we described above.

10 RELATED WORK

There have been many occupancy projects in the past for a variety of spaces. Even during this semester a fellow section B team is developing a version that calculates wait times in queues for food establishments. Past projects include one from 2020 that detected occupancy in a library using cameras. It aimed for a usage time of of 72 hours (3 days) and an object detection accuracy of 76.5%. Another project from 2022 attempted to use cameras as well to detect occupancy in study spaces. This project aimed for 45 seconds detection delay and a margin of error of 20%.

A third project from 2021 is similar to our approach. It utilized chair sensor modules to detect occupancy at the library. This project achieved between 17.6 and 27 seconds of delay and over 4.2 days of battery life with continuous use as well as possibly 1.5 months with sleep mode configurations.

Thus our project in some ways is better than others but lacks in other areas. For instance, in terms of detection accuracy and latency, we surpassed the figures of previous projects utilizing computer vision and was quite on par with the 2021 project. However, our implementation was lacking in terms of battery life compared to the max achieved by the 2021 project. The project that used computer vision did not have to worry about battery life and the 2020 projects battery life was on-par with ours.

Thus for usage time our implementation is not the most ideal yet for accuracy and delay it works much better than computer vision, which validates the thought process we went through while designing the system.

11 SUMMARY

Overall, our system met our expectations based on the minimum viable product we wished to deliver of 4 sensor modules and a web app, and the use-case requirements we determined. Detection accuracy and delay requirements were met with over 90% accuracy and less than 30 seconds of delay. When it comes to usage and battery life our requirements were conservative yet met. We recognize that although the 16.5 hour requirement was met, a solution that allows for a full week or even longer of continuous operation would be ideal and optimal. More so, an implementation that does not require charging would be stellar.

In terms of the invasiveness of the system it is fairly noninvasive, yet our mounting spots and encasings could be a bit cumbersome for some users since they occupy some dashboard space that is used by gym goers for other purposes like holding their phones or other belongings. As mentioned in section 6.1, our module is not fixed and securely mounted on the gym equipment. Thus, are module is not safe from tampering, theft, damage, and other consequences stemming from human interactions. Moreover, this can have serious repercussions for our design since a stolen, broken, poorly positioned, or missing sensor module greatly diminishes the effectiveness, accuracy, and usability of our system.

In consequence our mounting mechanism is not efficient nor realistically sustainable. A proper solution would require secure fasteners. The optimal solution is to have the sensor powered by the machine and integrated into the console so that we eliminate the recharging hassle and reduce the risk of people tampering with the sensor while also en-

suring its continual use.

So, a future project could incorporate a better and more secure mounting mechanism that reduces dashboard space used and increases module stability or could be developed around a completely different implementation that eliminates this problem. With this in mind, further expansions could also be done to include more and other machines in the UC gym, such as other cardio machines and weight machines on the first floor. Aside from the mounting problem, another key point to focus on and improve is battery life.

Glossary of Acronyms

- ADC - Analog to Digital Converter
- AWS - Amazon Web Services
- DDoS - Distributed Denial-of-Service
- EC2 - Amazon Elastic Compute Cloud
- GND - Ground
- HTML - HyperText Markup Language
- IDE - Integrated Development Environment
- IoT - Internet of Things
- IR - Infrared
- JSON - JavaScript Object Notation
- JS - JavaScript
- LED - Light Emitting Diode
- NMCU - Node Microcontroller Unit
- RPi - Raspberry Pi
- SQL - Search Query Language
- UC - University Center

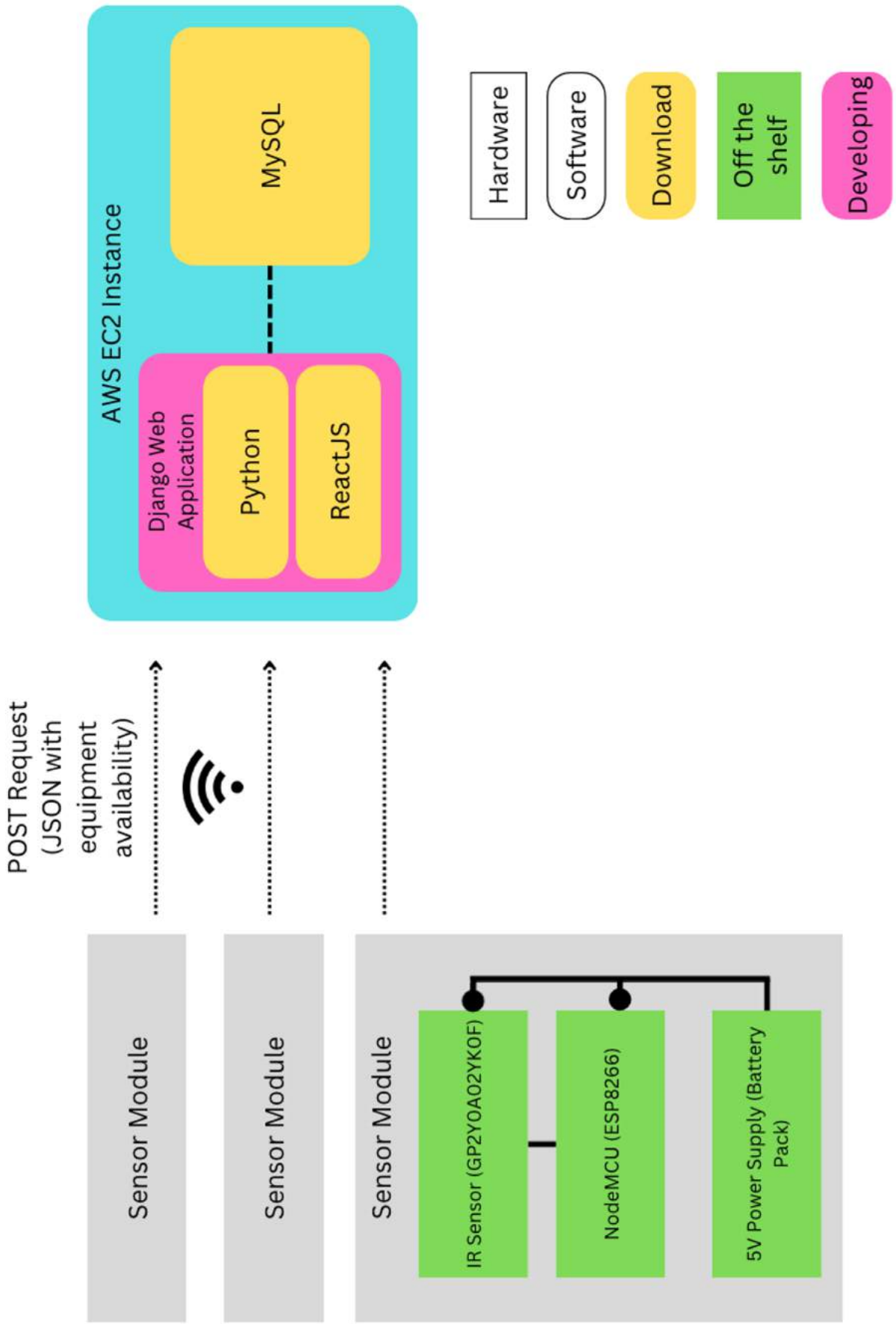

Figure 20: A full-page version of the same system block diagram as depicted earlier.

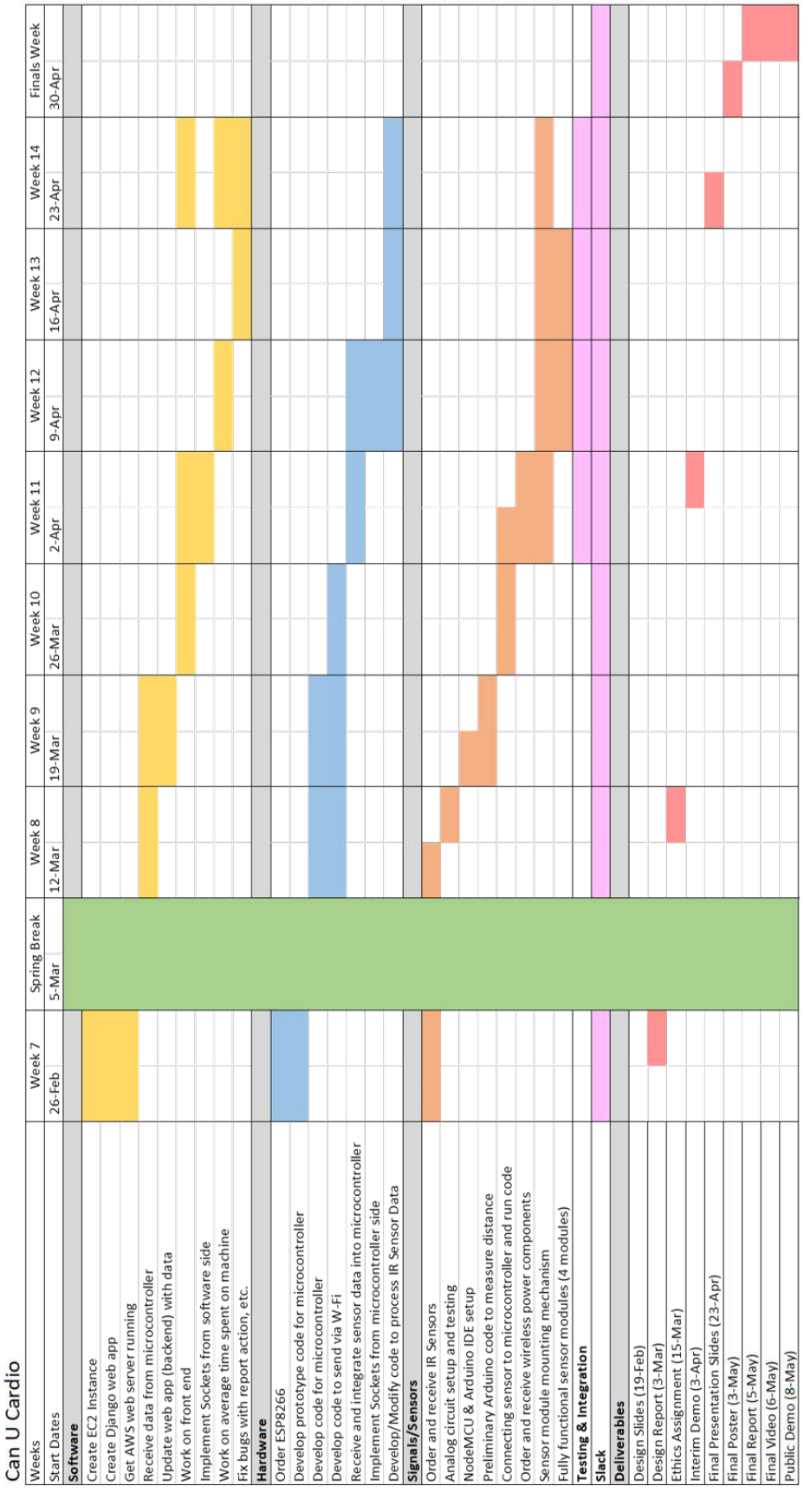

Figure 21: Gantt Chart