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*—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 will help students optimize the use of their time in the gym by providing real-time occupancy and availability of gym cardio machines. The system will use 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 will aid 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 will provide 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? aims to utilize different technologies like physical 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 our goal is to surpass these metrics in order to improve upon existing solutions and make Can U Cardio? a viable and useful tool for busy students and fitness enthusiasts.

2 USE-CASE REQUIREMENTS

With our use case in mind, we have created requirements that will ensure 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 want to have an overall detection accuracy greater than or equal to 90%. There are two factors that will determine this metric. First, we must properly and consistently identify a single machine as occupied. Therefore, we can establish that we must fulfill a detection accuracy metric on a per machine basis. In order to fit the overall accuracy, this metric needs 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 must be 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 should 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 how invasive the system should be. Users want as little of any sort of interference as possible 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 should 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 should 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 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 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 interpret 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 Arduino code and libraries that have 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 customize and modify its operation to meet our requirements and operational metrics.

The final component of our sensor module was the power supply. Given that both the IR sensor and the NMCU work at similar voltage ranges, a 5V power supply is best in this case. With modularity and size in mind, we decided to draw power from batteries. The type and capacity of these batteries was not decided at this stage. It needs to be carefully considered. The design requirements will aid this decision and more details will be provided in the System Implementation portion of this report.

3.2 Software Stack

With our sensor module already deigned, 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.

First, 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 will need to contain 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 will be updated and the data will be stored in the MySQL database. Although the default Django database of SQLite would work with EC2, MySQL is more scalable under load and provides more robustness compared to SQLite.

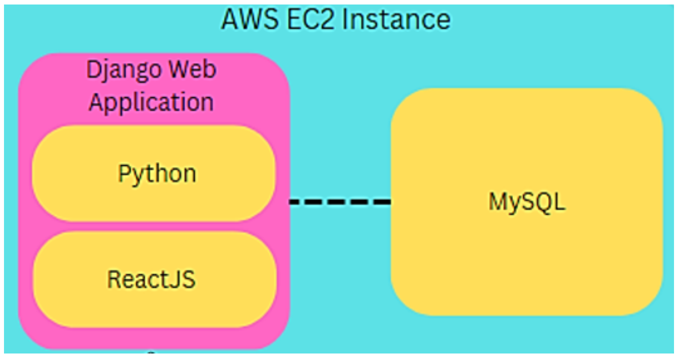


Figure 1: The Software Stack, consisting of an EC2 instance hosting the Django web app and MySQL

4 DESIGN REQUIREMENTS

Given the architecture of our system and the chosen mechanism for occupancy detection, a couple design considerations come into play. We must 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 documen-

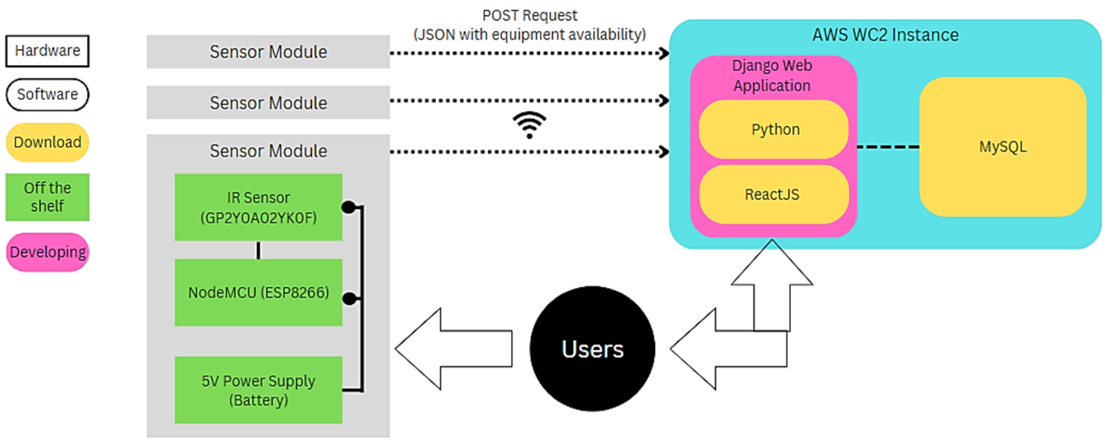


Figure 2: Block diagram depicting connections and data flow of the system.

tation for the machines we will base our design on, we know that the farthest detection range we will be dealing with is the treadmill. 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 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 1ft (= 12" ≈ 31cm) to 5ft (= 60" ≈ 152cm). The upper bound on this range will help with the detection accuracy on treadmills, since we do not want to mark the machine as occupied if someone passes in front of it farther than 5ft from the control panel where the sensor will be mounted.

With detection accuracy in mind, we will need a programmable microcontroller that allows 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). 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 1ft to 2ft from the sensor module mounting points. As a result, we need to be able to establish an artificial range within the one our sensor provides in order to avoid the same problem described above in the case of the treadmill. We do not want mark a bike as occupied unless someone is actually sitting on it rather than passing along.

The detection delay use case requirement translates into a latency requirement for our system. As a result, we want our overall system to have a latency of less than 30 seconds. This overall latency will be affected 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 will be 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 will reduce the invasiveness of our sensor module and make it and easily mountable to the gym machines selected. With this design choice we must also factor in the usage time use-case requirement. Ee have determined that our sensor module composed of the IR sensor and the NMCU needs to have a battery life of at least 16.5 hours to last through the whole day. From inspecting the documentation on each component we plan on using, we know that the IR Sensor can take between 4.5 V to 5.5V of supply voltage and operates at 33mA. The NMCU can take anywhere from 4.5V to 5.5V and typically operates at around 80mA. The combined current drawn is 113mA. Therefore, we need a 5V battery with a capacity of roughly 1,864.5mAh.

5 DESIGN TRADE STUDIES

While envisioning our system, we identified three key areas our design will be centered around 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. With our detection accuracy requirement of 90% in mind, 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 are needed to implement our system. However, this adds a layer of cost and complexity that is 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 addresses the issue of privacy, keeping the user anonymous in terms of occupancy metrics. These sensors allow for contactless sensing, making the system relatively non-invasive. When it comes to accuracy, such sensors allow for more accurate readings given that the idea is 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. 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. Another aspect to consider is the range, which in this case exceeds our requirements. In addition its robustness to vibration is low which is a concern given that these sensors will be physically mounted on treadmills and bicycles under use, hence the sensors will have to deal with an element of vibration and minor motion.

- IR Sensor: Although this sensor is more sensitive to its environment and object detection is affected by color and transparency, given the nature of our application, these factors are not a hindrance The temperature and conditions inside a gym are stable which eliminates the need for a sensor that works in harsh environments. The people typically using the machines wear clothing with a range of colors therefore we expect some variation in sensing, yet it will not be an issue since users will typically be within 4ft or less of the sensor, compensating for this color sensitivity. As discussed, this sensor's range is smaller and more suitable for our needs and its robustness for vibration suggests that it will provide stable readings even when the machines are in use.

Given the information compiled and its subsequent analysis, we decided to utilize IR proximity sensors since, despite some drawbacks compared to ultrasonic sensors like cost and power consumption, the levels of 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 it would slow down with *many* requests. Clearly, this is not a viable option.

5.2.2 Amazon EC2

Instead, hosting will be 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.

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 only has 3 three connecting wires: a wire for Vcc or supply voltage, one for GND, and one for Vo or the output voltage. Recall that this sensor is analog so we will be using ADC pin of the NMCU to connect it. The NMCU has internal regulator circuitry that will process this input in order to interpret it and program the microcontroller to send the data.

Given our battery life and modularity requirement from section 4, we opted to power the whole module using rechargeable AA batteries in a 6V battery holder (4 battery spaces) that will supply 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 intend to use 4 batteries per module totaling 4.8V which is in the range of operation of both components. Each module will have a total of approximately 11,200mAh, meaning that a single module will most likely be operational for 5 to 6 days if interrupted (turned off at the end of the 16.5 hour schedule of the gym) or approximately 4 days uninterrupted.

All these components will be housed inside a house-fabricated (3D-printed or handcrafted) encasing that can be attached to the dashboard panel or other points of the gym equipment. Possible mounting points for this module will be determined based on the results of our testing. Another factor in deciding this will also be the comfort element for the gym user since we want invasivness and interference to remain low. Nonetheless, initial design considerations suggest that our case should be adaptable and fit multiple mounting mechanisms like, clips, Velcro, or straps since we will be monitoring two types of equipment (bikes and treadmills). As a result, separate studies with each machine will be conducted on each type of equipment to determine the distance and orientation that is most optimal to safely secure the module in a manner that provides accurate and consistent readings.

6.2 Software Stack

The Software Stack contains the EC2 instance that the web application and MySQL will be 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 would specify the type of equipment (bicycle or treadmill) in the form of a string, the equipment id in the form of an integer to serve as an identifier, the status of the equipment in the form of a string ”free” or ”busy”. Moreover, models can be used to store the times the POST requests are made, such as a change from ”free” to ”busy” and vice-versa, so that the average use time can be calculated and displayed on the web-app.

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 determining the average time per machine or typical business 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. However, if the t2.micro is not suitable for our needs, we have space allocated in our budget to purchase credits for a stronger instance.

Finally, the way the Sensor Module contacts the web application is via a POST request made. The microcontroller would send a POST request to the web application in the form of a JSON, which would contain the following information:

- id: an integer identifier that corresponds to each machine. This allows us to identify which machine is sending the request.
- key: a UUID (32 chars) that is hidden to public in order to allow us to verify that the POST request is coming from the machine and not a malicious actor. The key would be checked in views.py to a list of preset keys, and if $key \notin keys$, the web app will not be updated.
- time: the time the POST request was sent, used in order to calculate average times and busiest hours.
- status: an integer identifier $\in [0, 1]$ that determines whether the equipment is busy ($status = 1$) or busy ($status = 0$)

Once the POST request is received, views.py verifies that it was sent from one of the sensor modules by verifying the key. Next, it would update the corresponding equipment model with its new status, and incorporate the new use time into the average use time. Then, the next time a GET request is made to the web app, the updated information will be shown.

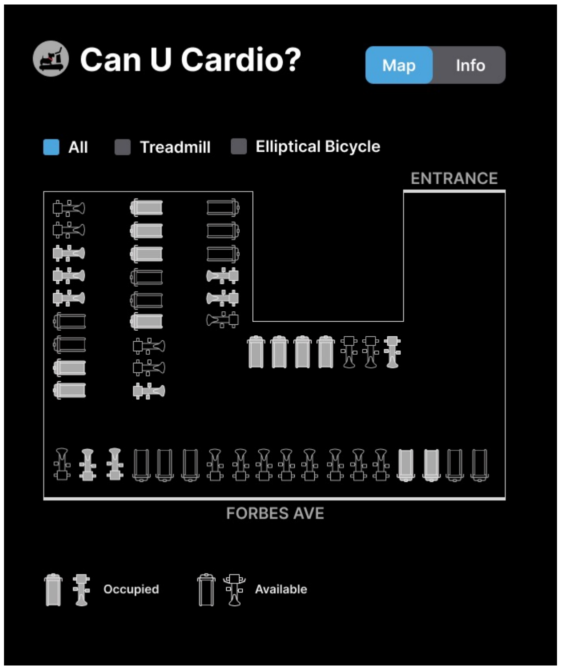
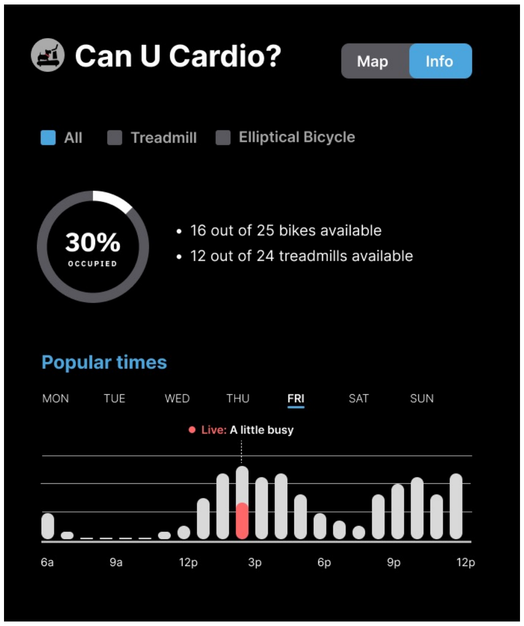


Figure 3: The web application's map, which displays a map of the UC Gym with equipment marked as free or busy



- 16 out of 25 bikes available
- 12 out of 24 treadmills available

Figure 4: The web application's info tab, which displays how number of equipment is currently in use and average business per hour

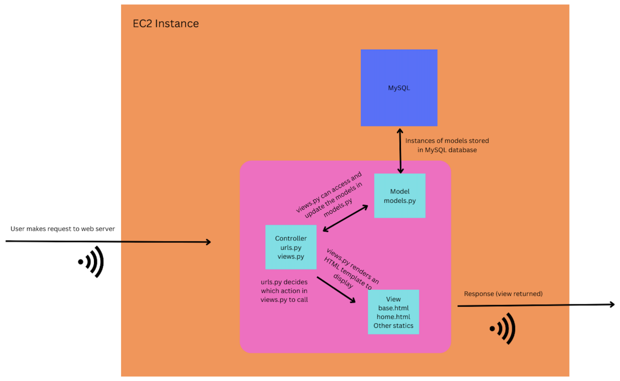

EC2 Instance
MySQL
Instances of models stored in MySQL database
views.py can access and update the models in models.py
Model models.py
User makes request to web server
Controller urls.py views.py
views.py renders an HTML template to display
urls.py decides which action in views.py to call
View base.html home.html Other statics
Response (view returned)

Figure 5: The MVC view of the Django web application

7 TEST & VALIDATION

With modularity in mind we have developed a series of tests that can were specifically designed and divided by subsystem. As a we have mentioned before the overall solution consists of two susbsytems: the sensor module and the software stack Thus we have testing plans for each seperate component.

7.1 Tests for Sensor module

As discussed before there are 3 elements for our sensor module: the IR sensor, the NMCU, and the power supply (batteries). Thus some tests will be done individually while others will be performed with assembled components.

The first thing we wanted to test was the operational metrics of our sensor. We want to obtain local performance of this component and cross-check it with the documentation from the manufacturer. This way we can adjust our system based on practical measurements. To do this we want to measure inputs and outputs of our IR sensor in terms of voltage and current. We must meet a 0V to 3V output measurement depending on the distance to ensure that our sensor does not damage the NMCU and to validate the manufacturer's claims and specifications.

To accomplish this we will utilize the ECE lab and its resources to wire up the sensor with a local power source and experiment with different orientations, colors, and materials and observe how the sensor behaves via a voltmeter and ammeter probes attached to the sensor or breadboard testing equipment. The main focus with this test will be distance and color sensitivity while the secondary yet quite important priority is determining the detection angle to understand how to best orient our sensor with detection accuracy in mind. With this test we will verify the design requirement of range and contribute data that can be used as a starting point to validate the use-case requirement of accuracy, effectively ensuring that the chosen sensor is the proper one for the application.

After local testing for the sensor we will focus on powering the NMCU and interfacing it with the development environment in order to program it. Once we establish this local connection we can move on to hook up the IR senor to the NCMU and start senidng data and see how it needs to be dealt with. All of this will be done with local power supply adjusted to the expected voltage from our batteries. Nonetheless for the purposes of local testing we need stable environment to ensure our component works as intended.

After this we will focus on power consumption and perform timed studies of power consumption with our batteries. We plan on measuring the consumption over 16.5 hours with nothing in front of the sensor (to mimic an idle voltage output) and alternatively with something directly in front of it (to mimic constant measurement). Our aim is to have the module last the full 16.5 hours even with constant measurement and transfer in order to ensure that in real-world scenarios of idle and measuring times the module will last the full day. From this data we should validate the use-case and verify and design requirements of 16.5 hours of usage time and battery life. However, we want to extend the testing to the 4 to 6 days mentioned in the System Implementation section if possible to see how the module behaves, with aims of exceeding our requirements for battery life making the overall system more usable and convenient for the administrative staff of the gym.

With these tests out of the way, the aim will shift to the module encasing and the method of mounting the module onto the gym machines. We will iterate with different designs and study these by testing their durability via stress tests involving vibration and other typical motions related to the usage of the machines. Given that the UC gym will most likely not be available for testing, we must replicate the dashboard configuration of each machine or get our hands on similar models that allow us to do this local testing. For this test our measure of success is uninterrupted and continuous reflection of occupancy on our software stack.

7.2 Tests for Software Stack

The first test for the Software Stack is to check that the Django app is successfully deployed on EC2. Next, we need to test that transmitting data from the sensor to the web app via the POST request works. To do this, we will simply test sending sample POST requests as described in Section 6.2, and checking if they are successfully received by the web app.

Both of these tests can be *verified*, or confirmed to meet design requirements, if there is a **latency** of less than 30 seconds as described in Section 4. Both of these tests can be *validated*, or confirmed to meet the use-case requirements, if the detection delay is subsequently less than 30 seconds, because our use-case requirement is that users want real-time data as to when an equipment is used.

Additionally, we want to test the following:

- Accurate data is displayed – we would test this by sending test JSONs and verifying that the correct data is being displayed on the web application.
- Filters – Given that our web application allows users to filter between treadmills and bicycles, we want to make sure that this filtering option is correct
- Fluidity – We want to determine that sending many requests at once still results in accurately displayed information. This test would pass if all the data is accurately displayed given we send 50 simultaneous POST requests with different ids.

These tests would all satisfy our accuracy use-case requirement (validation), as we aim for detection accuracy $\geq 90\%$.

8 PROJECT MANAGEMENT

8.1 Schedule

Our schedule is split up into three general areas: the Software Stack, the microcontroller, and the IR sensor. Ian Brito will work on the Software Stack, Nataniel Arocho-Nieves will work with the NodeMCU, and Ian Falcon will work with the IR Sensor.The schedule is shown in Fig. 7.

8.2 Team Member Responsibilities

Ian Brito: Software Stack Nataniel Arocho-Nieves: NodeMCU Ian Falcon: IR Sensor All: Slack, Integration, Testing

8.3 Bill of Materials and Budget

Table 1 lists the Bill of Materials for our project. Most of our costs incurred will be on the components for our sensor modules with one item for AWS credits to facilitate our web application. The total cost for the materials in our project is $101.35. However, more costs could be added based on testing and any redesign. Note that our bill of material covers our minimum viable product of 3 working modules, since adapting a solution for the entirety of the UC gym cardio floor would exceed our budget. In addition we wanted to save a significant portion of the budget in case of any major roadblocks or inconveniences that might happen along the way. Furthermore, we might decide to upgrade from the free tier of EC2.

8.4 Risk Mitigation Plans

If the t2.micro EC2 instance is not powerful enough for the Django web app, one risk mitigation plan is to allocate our budget to pay for an upgraded tier. Next, if for some reason EC2 deployment does not work, Heroku would be the next option to deploy the Django web app.

Table 1: Bill of materials

Description	Model #	Manufacturer	Quantity	Cost @	Total
AA Rechargable Batteries		HiQuick	16 (16-pack)	$2.31	$36.99
AA 6V Battery Holder		QTEATAK	4 (4-pack)	$1.61	$6.44
NodeMCU	ESP8266	NodeMCU	3-pack	$5.46	$13.68
Infared Proximity Sensor	GP2Y0A02YK0F	Sharp	4	$11.06	$44.24
					$101.35

9 RELATED WORK

10 SUMMARY

Glossary of Acronyms

- AWS - Amazon Web Services
- EC2 - Amazon Elastic Compute Cloud
- GND - Ground
- HTML - HyperText Markup Language
- IoT - Internet of Things
- IR - Infrared
- JSON - JavaScript Object Notation
- NMCU - NodeMCU (Microcontroller Unit)

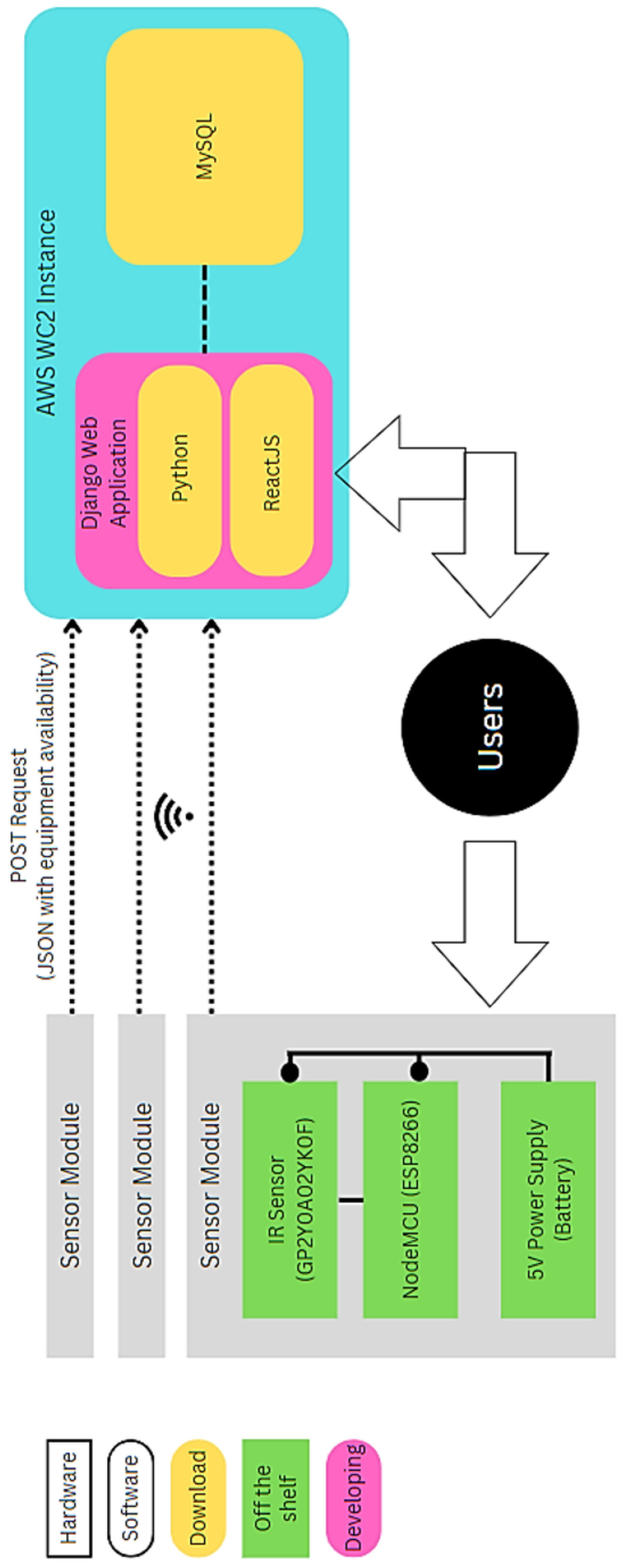
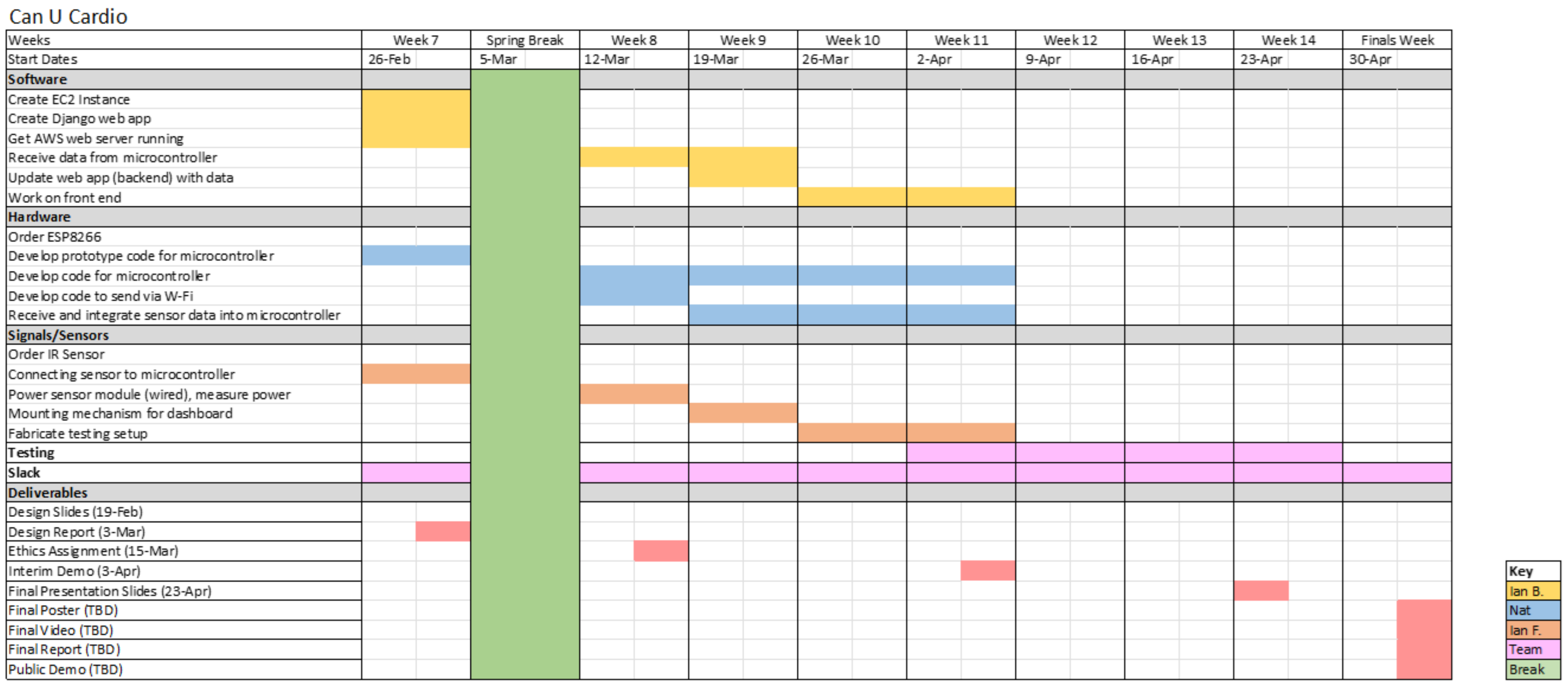


Figure 6: A full-page version of the same system block diagram as depicted earlier.



Can U Cardio

Weeks	Week 7	Spring Break	Week 8	Week 9	Week 10	Week 11	Week 12	Week 13	Week 14	Finals Week
Start Dates	26-Feb	5-Mar	12-Mar	19-Mar	26-Mar	2-Apr	9-Apr	16-Apr	23-Apr	30-Apr
Software		🟩								
Create EC2 Instance	🟨	🟩								
Create Django web app	🟨	🟩								
Get AWS web server running	🟨	🟩								
Receive data from microcontroller		🟩	🟨	🟨						
Update web app (backend) with data		🟩		🟨						
Work on front end		🟩			🟨	🟨				
Hardware		🟩								
Order ESP8266		🟩								
Develop prototype code for microcontroller	🟦	🟩								
Develop code for microcontroller		🟩	🟦	🟦	🟦	🟦				
Develop code to send via W-Fi		🟩	🟦							
Receive and integrate sensor data into microcontroller		🟩		🟦	🟦	🟦				
Signals/Sensors		🟩								
Order IR Sensor		🟩								
Connecting sensor to microcontroller	🟧	🟩								
Power sensor module (wired), measure power		🟩	🟧							
Mounting mechanism for dashboard		🟩		🟧						
Fabricate testing setup		🟩			🟧	🟧				
Testing		🟩				🟪	🟪	🟪	🟪	
Slack	🟪	🟩	🟪	🟪	🟪	🟪	🟪	🟪	🟪	🟪
Deliverables		🟩								
Design Slides (19-Feb)		🟩								
Design Report (3-Mar)	🟥	🟩								
Ethics Assignment (15-Mar)		🟩	🟥							
Interim Demo (3-Apr)		🟩				🟥				
Final Presentation Slides (23-Apr)		🟩							🟥	
Final Poster (TBD)		🟩								🟥
Final Video (TBD)		🟩								🟥
Final Report (TBD)		🟩								🟥
Public Demo (TBD)		🟩								🟥

Key
🟨 Ian B.
🟦 Nat
🟧 Ian F.
🟪 Team
🟩 Break

Figure 7: Gantt Chart