Laptop Lending Status System

Design Document

sddec20-02
Client: Eric Schares
Advisor: Maruf Ahamed
Team Members
Zoe Sanders — Software
Farouk Al Obaidi — Software
Camden Thomas — Software
John Wagner — Hardware
Aaron Thune — Hardware
Ryan Ray — Software
Team email: sddec20-02@iastate.edu
Team Website: http://sddec20-02.sd.ece.iastate.edu/

Revised: Date/Version

Executive Summary

Development Standards & Practices Used

List all standard circuit, hardware, software practices used in this project. List all the engineering standards that apply to this project that were considered.

Hardware and Standard circuit:

- Use of Raspberry Pi Zero W microcomputer to control each rack module
    - Will communicate with a more powerful Raspberry Pi acting as a central server to receive status updates for its rack
    - Using a pre-fabricated strip of WS2811 LEDs which support a wide range of colors and have optimized driver library support on the Pi
- Support for at least 50 RGB LEDs per rack module
    - Be cognizant of the power-draw requirements for the modules, to ensure reliability of the circuitry in the worst-case scenario
    - Rack modules are designed to be as easy-to-assemble as possible, allowing additional racks to be created by library staff in the future
    - Provide sufficient power for both the Pi and the LED strip in a combined circuit, to simplify construction and reduce use of wall outlet real estate

Software practices:

- Documentation
    - Code that is at all ambiguous should have block or inline comments preceding it.
    - All modules and programs should have README files detailing usage, system requirements and dependencies
- Coding Standards
    - Package and Module names should be all lowercase
    - Class names should follow CapWords convention
    - Numerical constants should be made and used as variables. (i.e., no magic numbers)
- Version Control
    - Consent is obtained from both branch owners before resolving a merge conflict.
    - Must obtain consent from entire team before merging a branch, or committing into Master

Summary of Requirements

List all requirements as bullet points in brief.

Requirements:

- RGB LEDs that display the status of laptops, tablets and equipments
- Central Raspberry Pi that communicates with other Raspberry Pis
    - Master/Follower Technology
- Each device rack should have its own Raspberry Pi.
- Each bagged device should have a corresponding LED
- All Raspberry Pi code should be in Python.
- The system should be integrated with ISU LibCal API.
- System Scalability for future equipements.

Applicable Courses from Iowa State University Curriculum

Software:

- ComS 339 (Architecture)
- ComS 319 (APIs)
- ComS 309 (GIT)

Hardware:

- CprE 288 (Embedded Systems)
- EE 201/220 (Circuits)

New Skills/Knowledge acquired that was not taught in courses

List all new skills/knowledge that your team acquired which was not part of your Iowa State curriculum in order to complete this project.

Software:

- Python
- LibCal

Hardware:

- Soldering
- Micro-Controllers

Table of Contents

1 Introduction

1.1 Acknowledgement

On behalf of the sddec20-02 team, we would like to acknowledge and thank the individuals that have provided support and assistance throughout the project development.

Library staff: David Harborth, Eric Schares, Lisa Smith, Mitch Steimel.
Team Advisor: Md Maruf Ahamed.

1.2 Problem and Project Statement

Problem statement

Parks Library lends over 200 individual laptops and tablets to ISU's student body through its Tech Lending office. The status of each individual machine, such as Available, Checked Out, Overdue, No Longer Lending, etc, is kept in the booking software LibCal, and also maintained on printed out sheets that are placed in each vacant slot in the charging rack. This current system makes maintaining at-a-glance information about the devices tedious as one must replace the sheets of paper with updated ones.

Solution approach

This project aims to replace the printed sheets of paper with RGB LEDs. Each RGB LED will display a color that will correlate to the status of each laptop. To do this, a Raspberry Pi will be used to communicate with the booking system API and pull the current status of each machine. These updates will then be sent to Raspberry Pis on each device rack or bar which will update the LED associated with the changed device. This will allow for a quick and easy way to identify the status of various devices at a glance.

1.3 Operational Environment

The final implemented system will be located in the Technology Lending office in Park's Library. The system is not expected to be exposed to any extreme temperatures or weather. The Technology Lending office was kept clean by the Iowa State University cleaning staff, and the room the implemented system will reside in is air conditioned.

1.4 Requirements

The requirements for the project are as follows.

- The system must be able to support more than 500 unique devices.
- The status of each device must be updated autonomously from the LibCal booking API.
- The rack modules should be simple enough that the library could make more if it's inventory were to expand.
- Each rack module must be able to support at least 20 RGB LEDs.

1.5 Intended Users and Uses

Once implemented, this project is intended to help the Park's Library staff, and student employees working in the Tech Lending office quickly and reliably determine the status of any device available for loaning. It is expected that the primary form of interaction between the intended users and

the system implemented will be purely visual, as the color status of the LEDs will be automated. However, if new devices are acquired by the Tech Lending office, a member of the Park's Library staff will be required to register the device in the system.

1.6 Assumptions and Limitations

Assumptions

- The status of a device will not change very often (no more than 30 laptop status changes per hour on average)
- IDs mostly be assigned consecutively i.e. l001, l002, l003, etc.
- An ID might be reused if an old device is replaced
- New devices will be added several times a year so our system should be able to scale

Limitations

- The status LEDs will only update every 30 seconds so they could show an old status for up to 30 seconds.
- The hanger case design does have the drawback of the LEDs staying in fixed positions, while the loaner items will likely not.

1.7 Expected End Product and Deliverables

We will deliver a system of micro-controllers and LEDs that link to a central software. A user should be able to modify the status of the LEDs, corresponding to the status of a loanable item. A bill of materials and instructions for the assembly of circuitry for the project will be provided, but actual installation of modules will not be completed this semester. The items mentioned will be delivered to the customer on Nov 19th, 2020.

We will deliver software that will use a Raspberry Pi, to communicate to other Raspberry Pi-Zeros, to modify the status of the LEDs. The program is to be used by the check-out desk staff, so it will be stable and easily understandable. The software will also have a procedure to add new loanable items to the system. This will be delivered to the customer on November 19th, 2020.

2. Specifications and Analysis

2.1 Proposed Approach

Our team's approach to the project is to designate a Raspberry Pi Zero for each computer rack to operate the LEDs (there are about 20 computers per rack so each rack-module Raspberry Pi Zero will operate about 20 LEDs). For the bagged equipment on racks our team's plan is to set up a PVC pipe with LEDs poked through, where the LEDs correspond to device IDs. This should provide enough space for the labels needed for each of the loaner items. It does have the drawback of the LEDs staying in fixed positions, while the items will likely not. Connected to each Pi will be our Master server which will manage each rack Pi. It will control the flow of device data from and to each rack Pi.

From the beginning we knew we would need to use microcontrollers to connect the data from the LibCal API to operating the LEDs. Although other microcontrollers may work, our team decided on a Raspberry Pi as it is affordable and comes in many different levels of complexity and functionality. Each rack and bar will have a lower-complexity raspberry pi, each of which connects to a central, higher-complexity Raspberry Pi which communicates with the LibCal API and sends out device updates to all other Pis.

2.2 Design Analysis

Hardware

We have decided to use an off-the-shelf strand of individually addressable RGB LEDS which can be controlled by the rack pis. Using off the shelf pre-fabbed parts makes it easier for our clients to assemble new rack Pis as they expand their inventory down the road. Each module of a Each module is powered via a 5-volt, 5-amp power supply, which covers our worst-case power draw scenario.

Software

For the software design, the architecture is divided into two main modules, Master Pi and Rack Pis, which are controlled by higher-complexity and a lower-complexity Raspberry Pis respectively. The lower complexity Pis which are the Rack Pis, handle the direct control of the LEDs and also receive updates from the higher-complexity Master Pi. The Master Pi is central to the entire process as it retrieves and sends updates from the LibCal API to the Rack Pis, controls the Administrative Web portal, and maintains and queries a key-value LevelDB database.

2.3 Development Process

Due to our team's varying experience with some of the technologies required for the project, we have decided it is best to follow the Agile development process. This allows us to continually improve our design as we acquire new information regarding the project. Additionally, by using this development process, we are able to react and adapt to changes in requirements from the client.

2.4 Conceptual Sketch

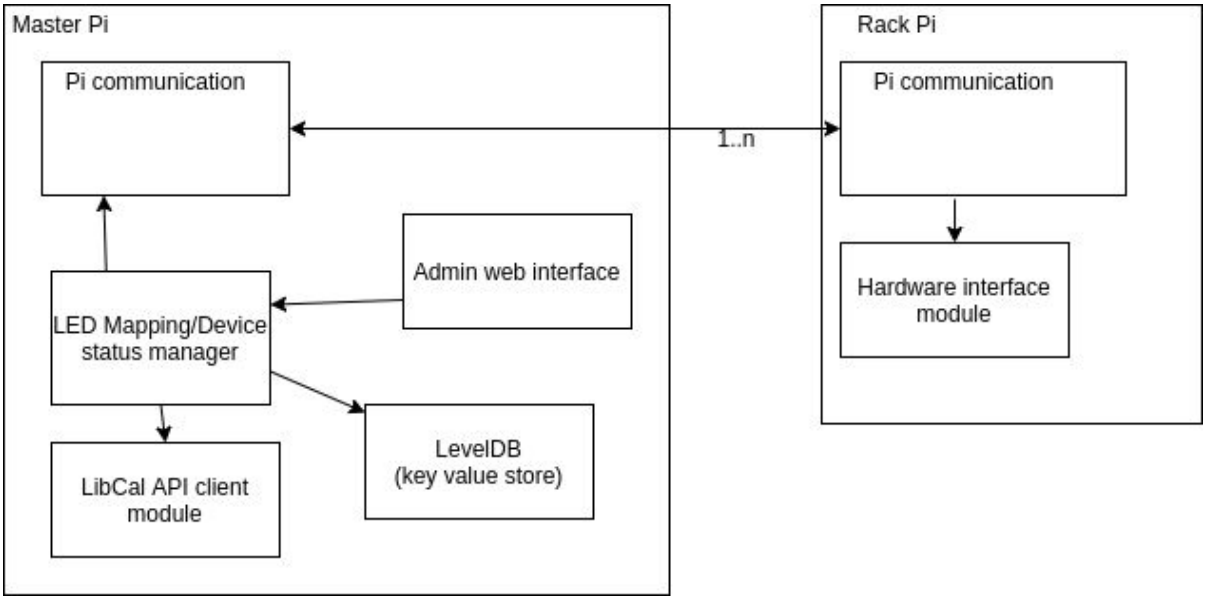


Figure 2.1: Software Architecture Block Diagram

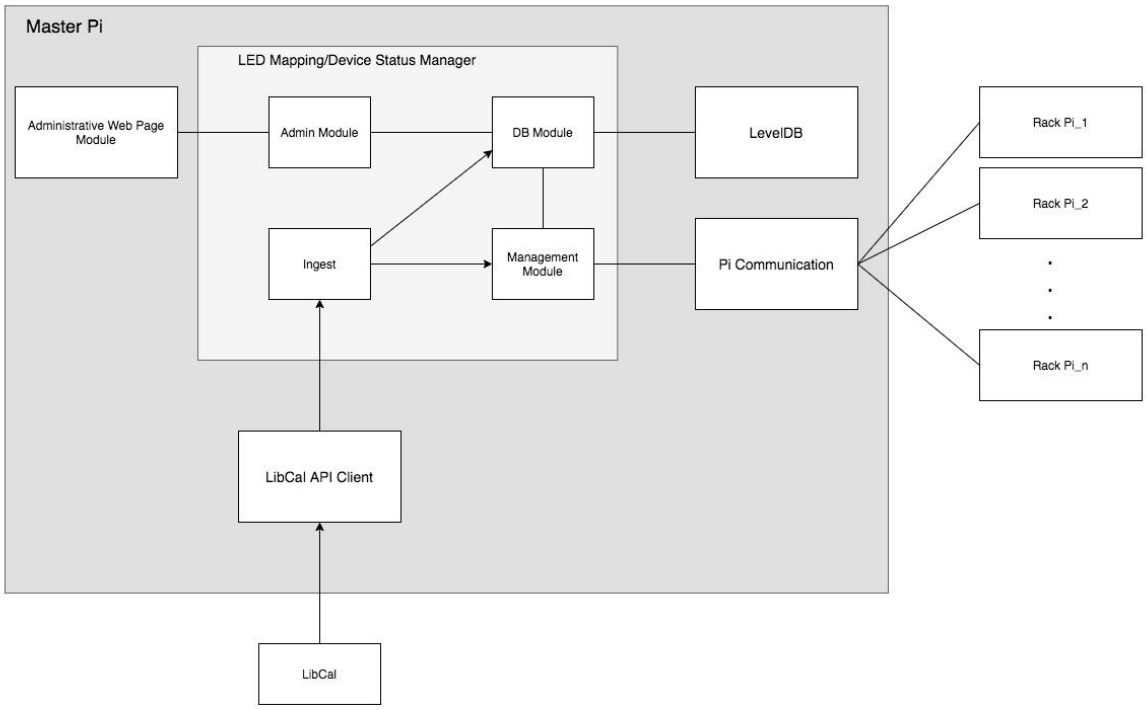


Figure 2.2: More detailed look at LED Mapping/Device Status Manager and its interactions internally and externally with other modules.

Machines

MasterPi - The central Raspberry Pi which manages the Pis, runs the web interface server and communicates with the libcal api.

RackPi - The Raspberry Pis running in the individual racks.

Modules

Rack Pi

- SubPi module: The SubPi module is installed on a Rack Pi and handles registering the Rack Pi with Master Pi as well as receiving LED updates, and forwarding them to the LED Driver Module. *(Python)*
- LED Driver Module: The LED Driver Module is responsible for controlling and updating the physical LED colors of the LEDs connected to its Rack Pi. *(Python)*

Master Pi

- Pi Communication Module: This manages all of the Rack Pis. It receives and forwards registrations from Rack Pis to the LED Mapping Status Manager (LMSM) module, and it forwards status updates from the MDSM to the Rack Pis. *(Python)*
- LED Mapping Status Manager: The LED Mapping Status Manager is central to the overall communication within the Master Pi module. It has the following modules.
    - *Management*: The Management module interfaces with the Rack Pis. It is responsible for receiving registration requests from new Rack Pis and registering them in the database, it is also responsible for checking for status changes and sending status updates to the Rack Pis. *(Python)*
    - *Ingest:* The ingest pulls updates from the LibCall API Client Module and compares them with the the current data in the LevelDB database, updates database accordingly. *(Python)*
    - *Admin Module*: The Admin module provides a useful interface for other modules to access the LevelDB database, and is responsible for notifying the *Management* module when changes are made to mappings. *(Python)*
    - *Database Module:* Handles setting and retrieving values from LevelDB database.
- LevelDB Database Module: This module is responsible for storing data about the Rack Pis (such as ID, LEDs and Health Status) and the Status-To-Color mappings currently supported.
- LibCal API Client: Responsible for interfacing with the LibCal API. Periodically retrieves device status data.
- Admin Web Interface: Responsible for providing an Administrative Web Page where users can check Pi and Device statuses, review system logs, update/add/delete status-to-color mappings and register new devices. Contains a Backend RestAPI written in Python which interfaces with the *LED Mapping Status Manager,* and a frontend UI written with HTML, CSS, Javascript which uses Ajax to interact with the backend. See Section 4.1.

The entire Master Pi module is booted up via Docker.

3. Statement of Work

3.1 Previous Work And Literature

There were little to none, exact existing projects, and there was no previous work done on this project, by another team. There was, however, a useful project we found in Robinson's, Raspberry Pi Projects, called Disco Lights. The project was powering LEDs to the frequencies of music/sound. It gave valuable python code and circuit design

"Chapter 11 Disco Lights." Raspberry Pi Projects, by Andrew Robinson et al., John Wiley & Sons Ltd, 2014, pp. 251–274.

3.2 Technology Considerations

1. Our project uses Http Polling for libcal which means the leds could have stale data while we are waiting to make our next set of requests.
   Alternatives: None, libcal does not support anything else
2. Our project uses Http servers on the Master and Rack Pis so that either end can initiate a connection. This means we don't have long lived connections (avoids issues like TCP hangups)
   Alternative: Sustained TCP connections
3. Our project uses key value storage instead of a traditional database which means if we need to make more complex queries they take a little longer to run.
   Alternative: Traditional database

3.3 Project Proposed Milestones and Evaluation Criteria

1. A single Raspberry Pi with an led that can be mapped to a libcal resource's state by the admin interface.
2. Multiple Raspberry Pis that receive live updates everytime we poll libcal and the admin interface is secured.
3. Logs and health checks from the raspberry pis can be viewed in the admin interface.
4. We have a custom Raspbian image for the Raspberry Pis.

4 Testing and Implementation

4.1 Interface Specifications

Software

In order to interface with the API, an admin web portal was created. The following image shows a simple debug page used to test communications with the back-end API.

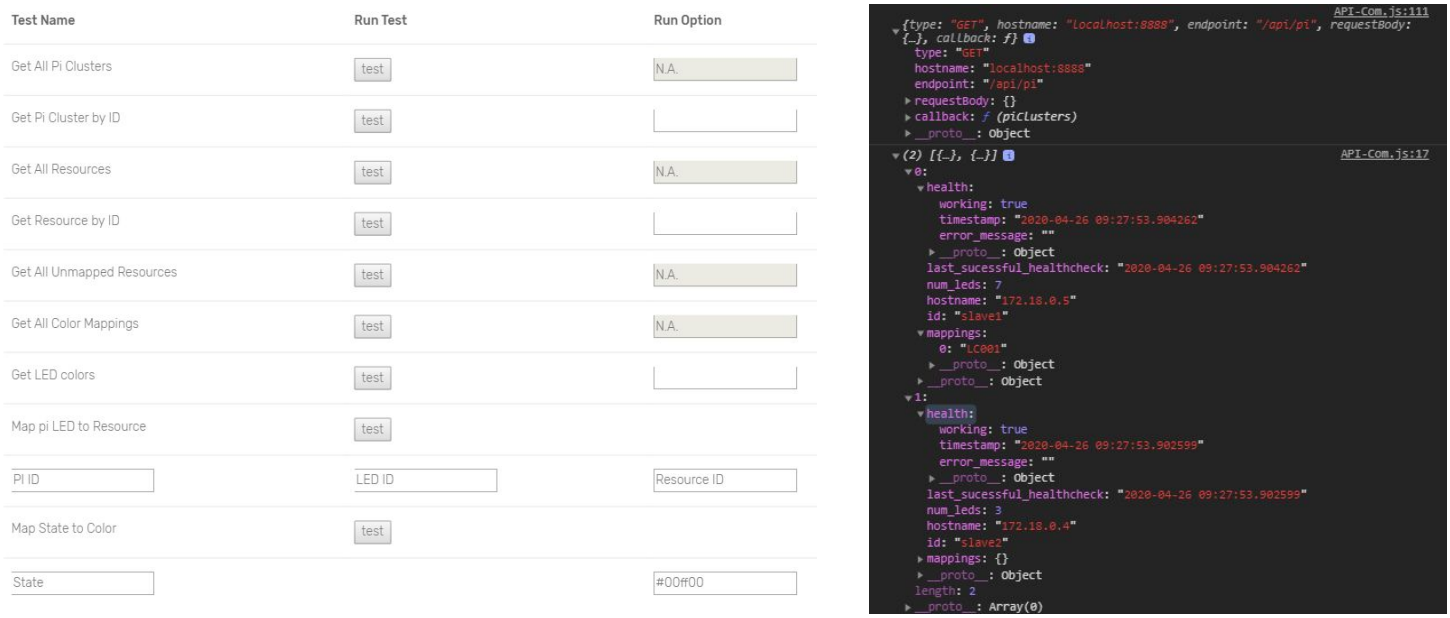


Figure 5.1: Debug Page on Web Interface

The console output on the right is the result of running the “Get All Pi Clusters” API test. The console output shows the AJAX request, followed by the API’s response. Currently there are only 2 pi clusters in our test database, however other devices will show here as more are added.

4.2 Hardware and Software Testing

For software testing we are using a combination of unit tests, integration tests and system tests. For each module we created unit tests to test the most important methods. Then we create integration tests at the module level. And then we have a system that tests all of the modules together.

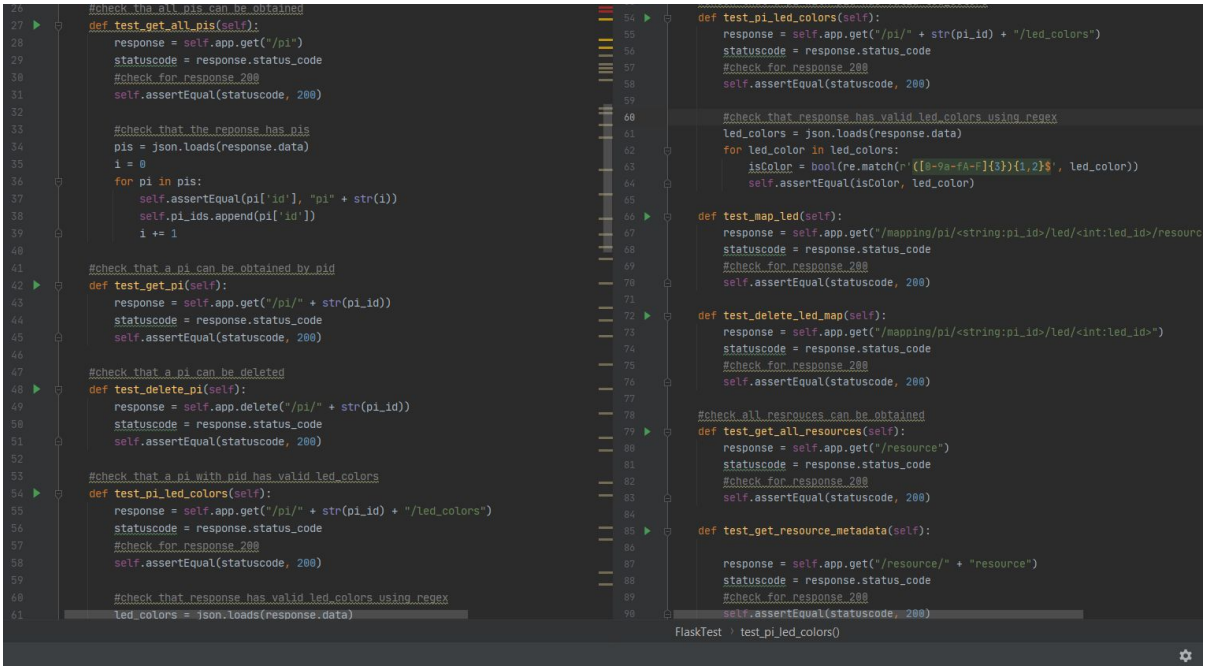
For development we use containers to simulate the master and rack pis and we have a mock api server which simulates the libcal api. The mock libcal api changes the statuses of resources frequently so we can see how the system responds to updates. This allows us to test in a very realistic environment while not having to each have our own raspberry pi setup.

4.3 Functional Testing

Unit testing:

1. Libcal AP
    i. Test that all the Libcal items are pulled.
2. Rest API
    i. Test that all the endpoints can be called.

The unit test code below shows the webapp RestAPI.



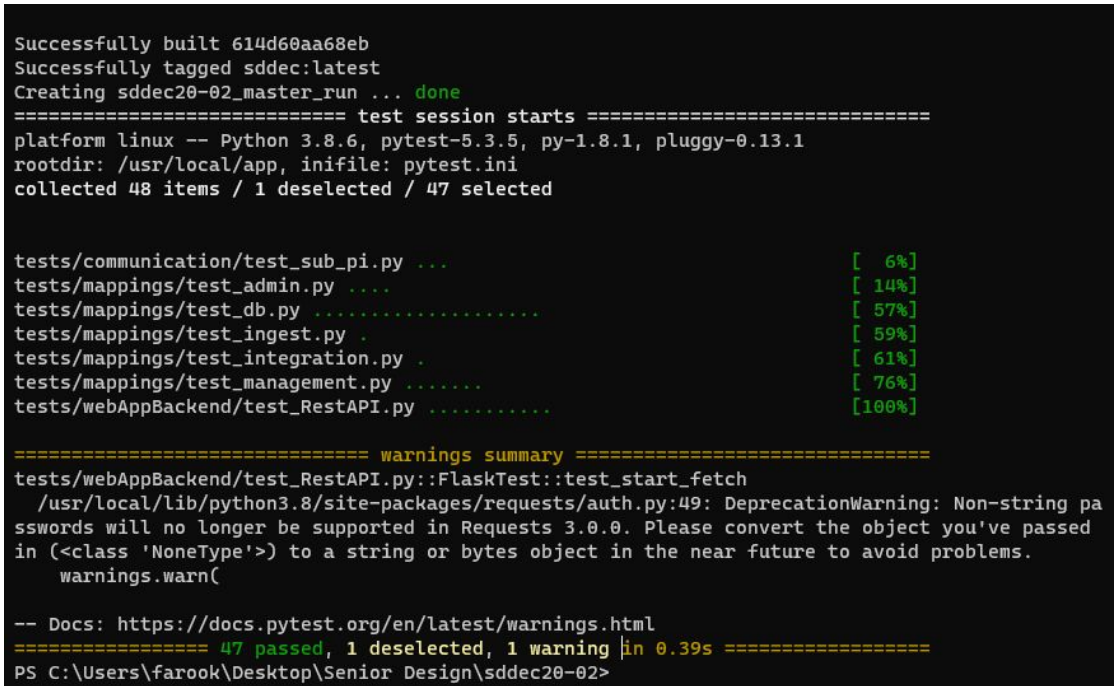
```
    #check tha all pis can be obtained
    def test_get_all_pis(self):
        response = self.app.get("/pi")
        statuscode = response.status_code
        #check for response 200
        self.assertEqual(statuscode, 200)

        #check that the reponse has pis
        pis = json.loads(response.data)
        i = 0
        for pi in pis:
            self.assertEqual(pi['id'], "pi" + str(i))
            self.pi_ids.append(pi['id'])
            i += 1

    #check that a pi can be obtained by pid
    def test_get_pi(self):
        response = self.app.get("/pi/" + str(pi_id))
        statuscode = response.status_code
        self.assertEqual(statuscode, 200)

    #check that a pi can be deleted
    def test_delete_pi(self):
        response = self.app.delete("/pi/" + str(pi_id))
        statuscode = response.status_code
        self.assertEqual(statuscode, 200)

    #check that a pi with pid has valid led_colors
    def test_pi_led_colors(self):
        response = self.app.get("/pi/" + str(pi_id) + "/led_colors")
        statuscode = response.status_code
        #check for response 200
        self.assertEqual(statuscode, 200)

        #check that response has valid led_colors using regex
        led_colors = json.loads(response.data)
```

```
    def test_pi_led_colors(self):
        response = self.app.get("/pi/" + str(pi_id) + "/led_colors")
        statuscode = response.status_code
        #check for response 200
        self.assertEqual(statuscode, 200)

        #check that response has valid led_colors using regex
        led_colors = json.loads(response.data)
        for led_color in led_colors:
            isColor = bool(re.match(r'([0-9a-fA-F]{3}){1,2}$', led_color))
            self.assertEqual(isColor, led_color)

    def test_map_led(self):
        response = self.app.get("/mapping/pi/<string:pi_id>/led/<int:led_id>/resourc
        statuscode = response.status_code
        #check for response 200
        self.assertEqual(statuscode, 200)

    def test_delete_led_map(self):
        response = self.app.get("/mapping/pi/<string:pi_id>/led/<int:led_id>")
        statuscode = response.status_code
        #check for response 200
        self.assertEqual(statuscode, 200)

    #check all resrouces can be obtained
    def test_get_all_resources(self):
        response = self.app.get("/resource")
        statuscode = response.status_code
        #check for response 200
        self.assertEqual(statuscode, 200)

    def test_get_resource_metadata(self):

        response = self.app.get("/resource/" + "resource")
        statuscode = response.status_code
        #check for response 200
        self.assertEqual(statuscode, 200)
```

FlaskTest › test_pi_led_colors()

The code tests various functionalities such as getting/deleting a pi, getting a pi led colors and making sure they are valid, mapping/unmapping a pi led to a resource, getting resources… etc.

The image below shows the console output of the unit test case above.



ii. Test that each endpoint calls its level DB method.

3. Level DB

i. Test that the database can be created.
ii. Test all methods that modify/insert/delete data from the database.

4. Pi communication

i. Test that slave pis register themselves with master pi.
ii. Test that that pis are always listening to updates from master pi.

5. LEDS

i. Test that LED can display any selected status color.

6. Admin GUI

i. Test that all the Rest API endpoints can be called using the Admin GUI.

Integration Testing:

· Test that when the Libcal API is called all the items are pulled and the database is updated. Additionally, the slave pis receive the new information from the updated database and the Pi LEDs are updated.

· Test that when the Admin GUI calls an endpoint, the Rest API calls the Level DB method for that endpoint and the right values are returned to the Admin GUI.

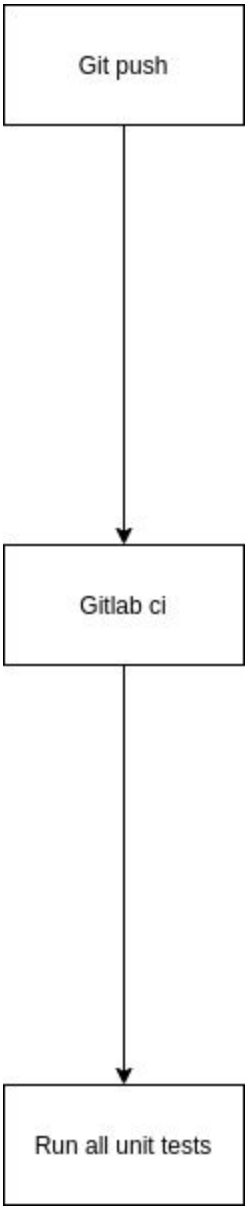
· Test that when the Admin GUI calls an endpoint to update mapping colors, the Rest API calls the Level DB method to update the mappings, the mappings are updated in the database, Pis receive the updates and pis update the LED colors.

4.4 Non-Functional Testing

Our main focus for non functional requirements are user experience and stability. For user experience we are going to pull in a volunteer and have them try to assemble our project to make sure we have enough documentation. For usability we are going to run the system for a day or so and see if it encounters any errors.

4.5 Process

Our software testing process is fairly simple. We use gitlab ci to run all of our tests (in no particular order) every time we push.



4.6 Results

Our unit tests almost always pass and on the few occasions where they fail the person who broke them quickly fixes the code or updates the test to match the new state of the code.

5. Closing Material

5.1 Conclusion

The project goal for our team was to build a functioning and scalable device loan status display for the Tech Lending center at Parks Library. This display would link each loanable device to an LED which will display a color corresponding to the device's current status (i.e., Booked, Reserved, etc.).

It was necessary to handle two cases when it came to the display of the LEDs. First there are racked items that are placed out on sets of 20 shelves. This case is simple to handle as we will be able to easily attach one LED on each shelf for the device on said shelf. The second case is more difficult as the devices are in bags and hung on bars. The difficulty arises due to the fact that the bags are checked out with the item (negating the possibility of attaching LEDs to the bags), and each bar doesn't have a set number of devices, they can range from 20-50 devices per bar. Thus, we decided to create a PVC enclosure to fit on the bars with holes for LEDs. See Appendix II figure 6.2.

For our software side, our goal was to connect to the LibCal API which housed all of the live device data including ID numbers and statuses, and then map that data to a Pi, and from that Pi to an LED. We achieved this by constructing a Leader-Follower architecture in which there is a central Raspberry Pi that runs the Web Interface, retrieves data from the LibCal API and forwards updates from both to the Rack Pis.

Overall, our team has made significant progress on our project but due to the unexpectedly shortened semester due to Covid-19, we were unable to complete the last phase of the project; installation. However even with this being the case, we are confident that we have designed our project in such a way that anyone should be able to follow our installation instructions and set up the system.

5.2 References

- Andrew Robinson et al., "Chapter 11 Disco Lights." in *Raspberry Pi Projects*, by Andrew Robinson et al., John Wiley & Sons Ltd, 2014, pp. 251–274.

5.3 Appendices

Appendix 0: Definitions

Term	Definition
GPIO	General-Purpose Input/Output
LevelDB	Key/value store API used to store device mappings to LEDs
LibCal	Library Scheduling API used by Parks Library and Tech Lending
MasterServer	Central server managing RackPis and LibCal connection.

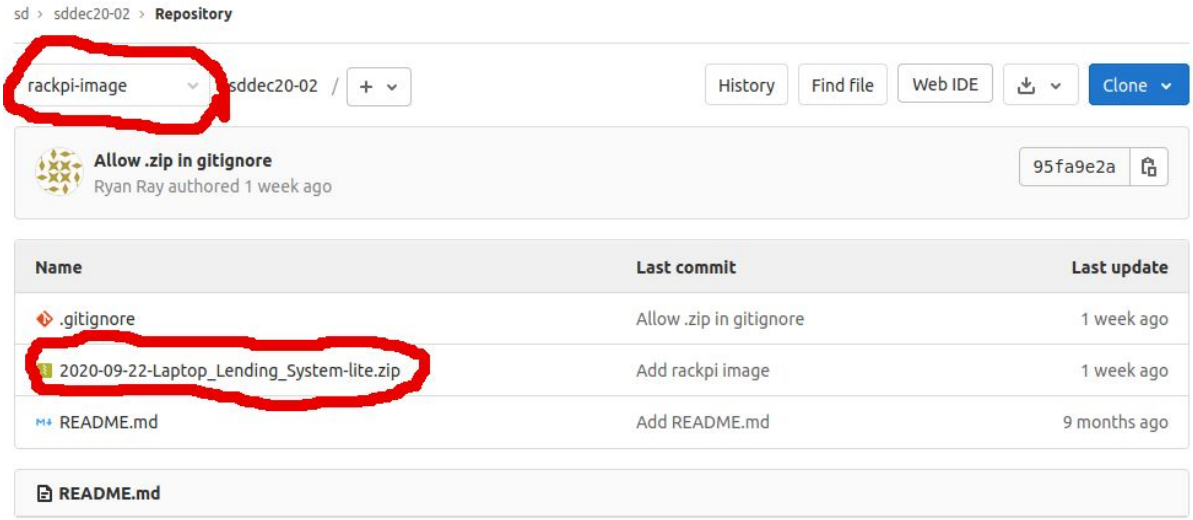
Pi Zero	A smaller and more affordable Raspberry Pi Model
PWM	Pulse-Width Modulation
RackPi / Sub Pi	Pi Zero assigned to a rack of devices

Appendix I: Manual

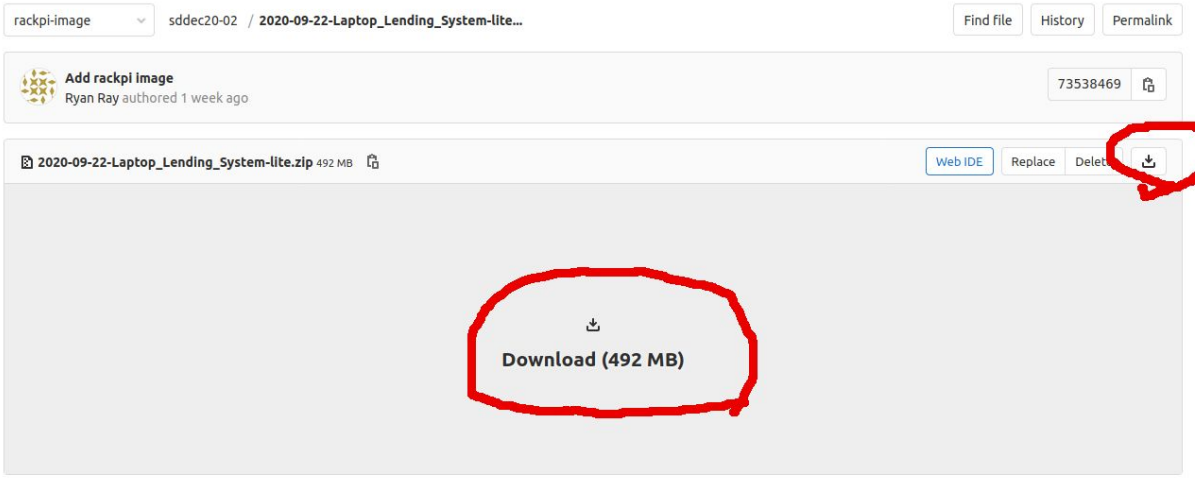
Installing the rackpi

This is basically the same as installing regular raspbian except you have to use our custom image.

To get the custom image go to https://git.ece.iastate.edu/sd/sddec20-02/-/tree/rackpi-image and make sure that rack-pi image is selected in the drop down. Next click on the zip file (the date may be different).



Click the either of the two download buttons.

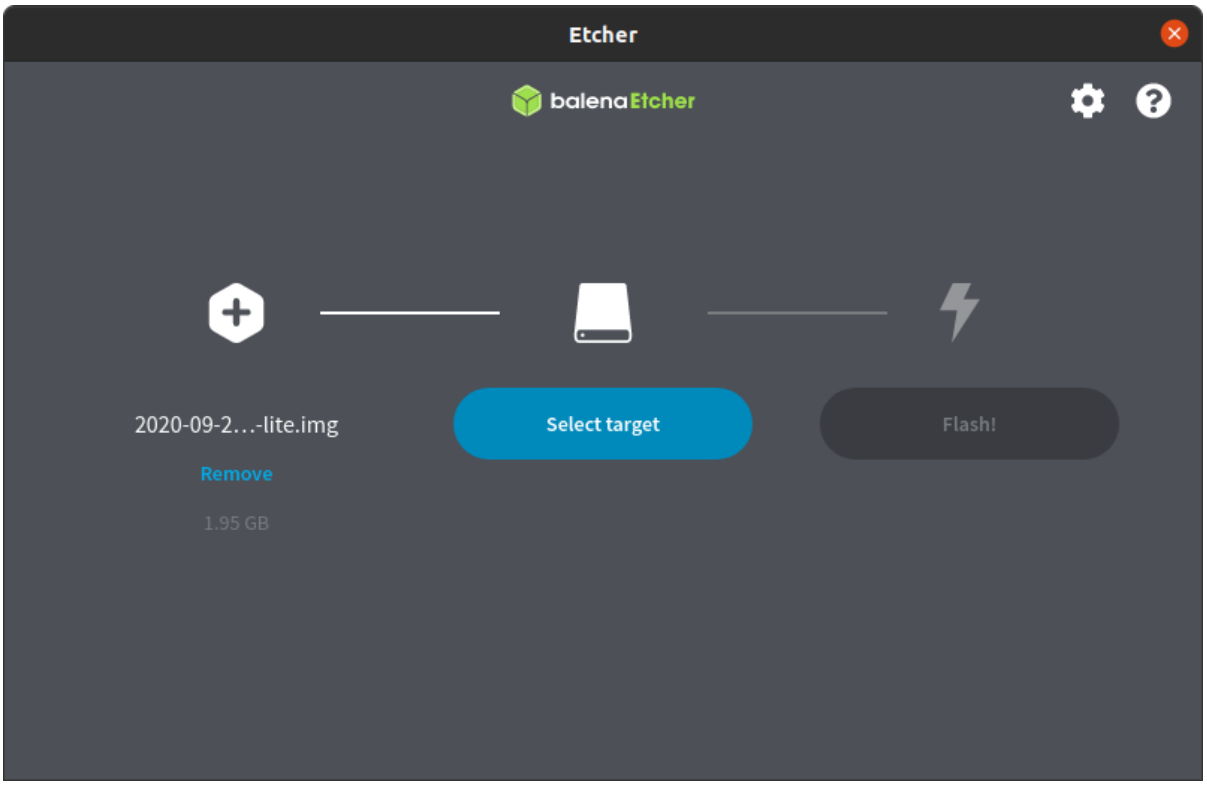


Unzip the file raspberry pi image. You should see a .img file (it probably has the known file type icon). Download etcher from https://www.balena.io/etcher/ and run it.

Insert the sd card for your pi into your computer.

Select the custom img file which you extracted from the zip archive, select the sd card you want to use for the pi, and click flash.

WARNING: flashing a drive will erase all data on it so make sure you select the correct device.



Configuring the rackpi

After booting the rack pi you should see its mac address printed in cyan.

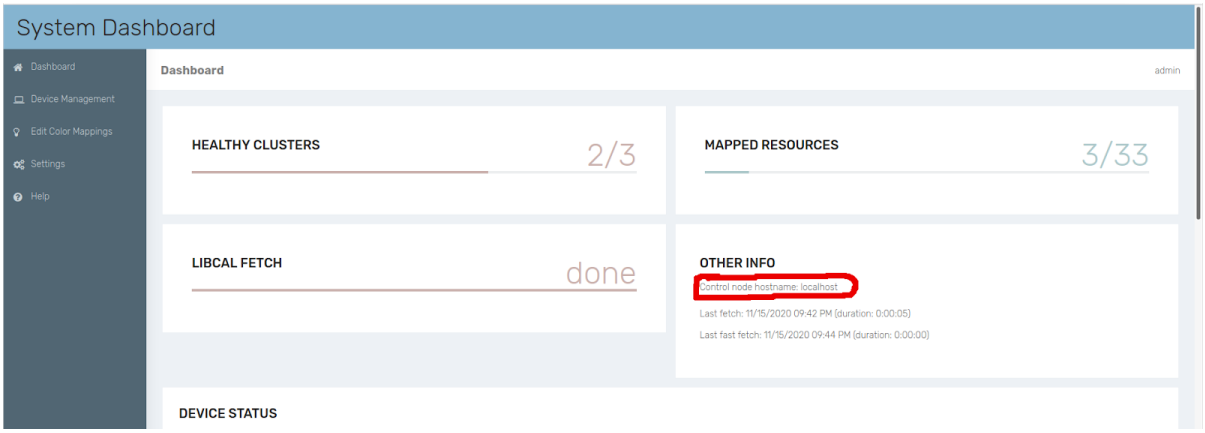
Unplug the rackpi and register the rackpi with netreg using its mac address.

Plug the rack pi back in and when you see this menu type C.

```
1) (C)onfigure the rack pi
2) check the rack pi service (S)tatus
3) show rackpi (L)ogs
4) restart rackpi service
5) run (B)ash
6) Disable Ctrl+c/exit/press any key messages
7) run (R)aspi-config, configures wifi, keyboard configuration and other os things about the os
8) (U)pdate rackpi
9) reboot
0) (E)xit

Rack pi service has not been configured yet (option 1/C) this must be done to connect it to the control/master node.

Choice:
```

When prompted enter the number of leds attached to the raspberry pi.

After that you will be asked for the control node hostname this can be found in the admin interface.



After that the rackpi will check that it can reach the control node if it succeeds it will install the latest rackpi module and start it. If it fails to configure see troubleshooting.

Assuming all goes well you see the rackpi under device status section in the admin interface.

Troubleshooting

Could not ping 1.1.1.1 you are probably not connected to the internet.

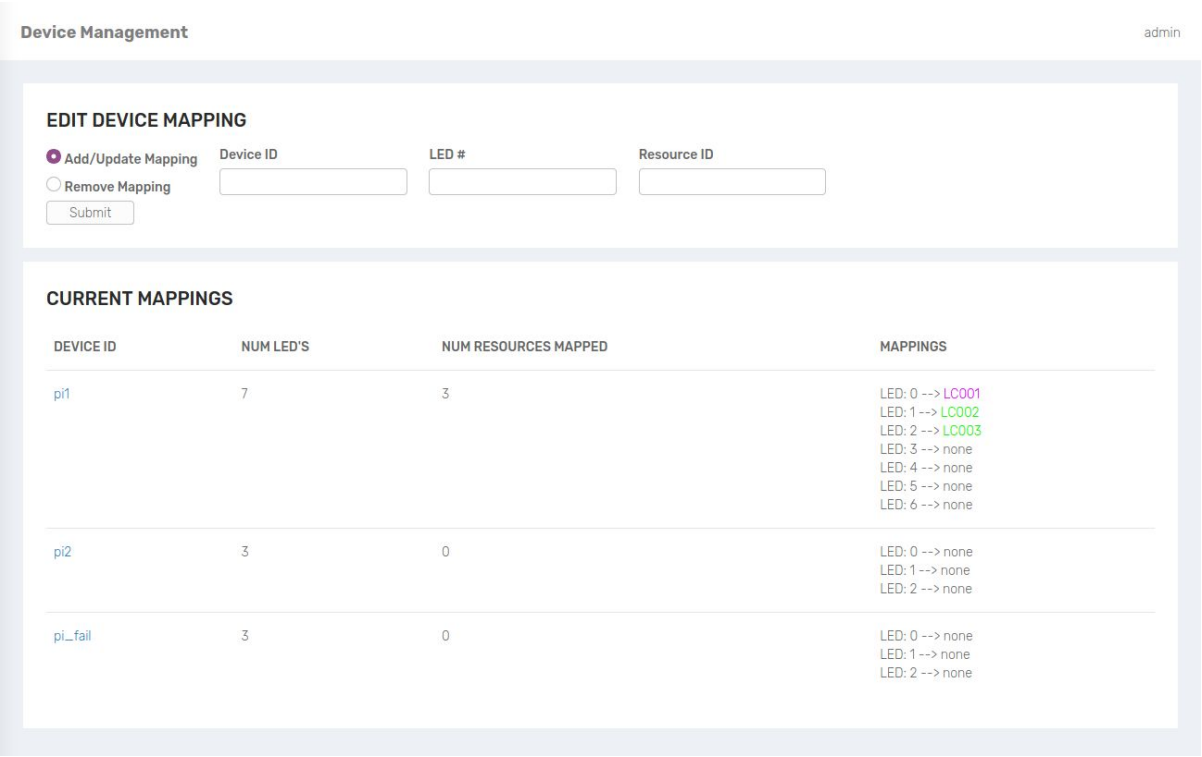
If you get this power off the pi and wait a few minutes it is possible its registration on Iowa States's network is still going through. If the issue persists type R in the main menu and use the raspi-config tool to connect to wifi.

Could not ping control node. Check that the machine is running and you have the correct hostname.

This probably means you entered the wrong hostname. Take the url you use to visit the admin interface and take the part between the http:// (or https://) and the next / and use that. So for http://ll.iastate.edu/bla you would use ll.iastate.edu as the hostname.

Configuring the admin interface

To map an LED to a libcal resource go to device management and enter a device id, led id and resource id from the table of devices. If resource or led is already mapped the existing mapping will be removed.



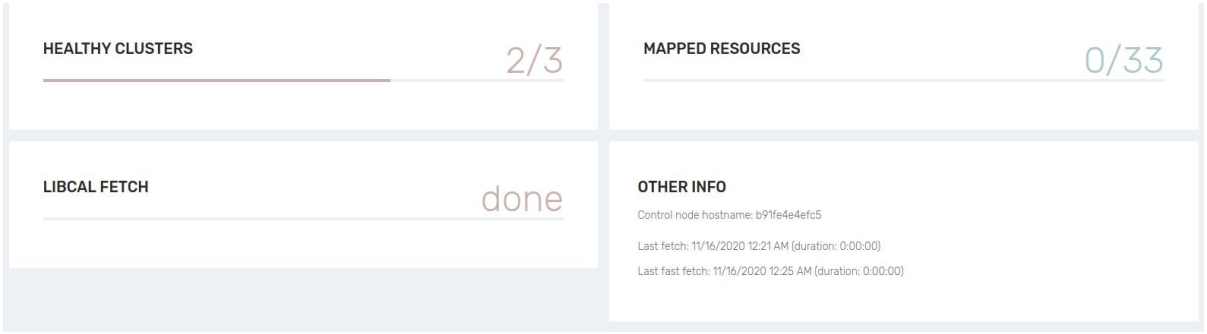
To rename a raspberry pi or change the number of leds attached to it click on the name in the admin interface to get to the pi page. Then just change the appropriate field and click save.

To delete a pi click the delete button. If the pi is still running it will register itself again in a few minutes.

To assign a color to a libcal state go to the color mappings page select the color with the color picker and click the save button next to it. As the libcal api detects new states they will be added to this page.

The unknown state is a fallback any unmapped led or color that has not been set will default to the unknown color.

Data will be fetched automatically from libcal every 5 minutes and the status of the current resources will be fetched every 30 seconds. The fast fetch, which happens every 30 seconds, should be good enough for most of the time but it will not detect if resources have been added or deleted. You can track the progress of fetches on the dashboard and see when the last fetch took place.



Appendix II: Schematics & Design Documents

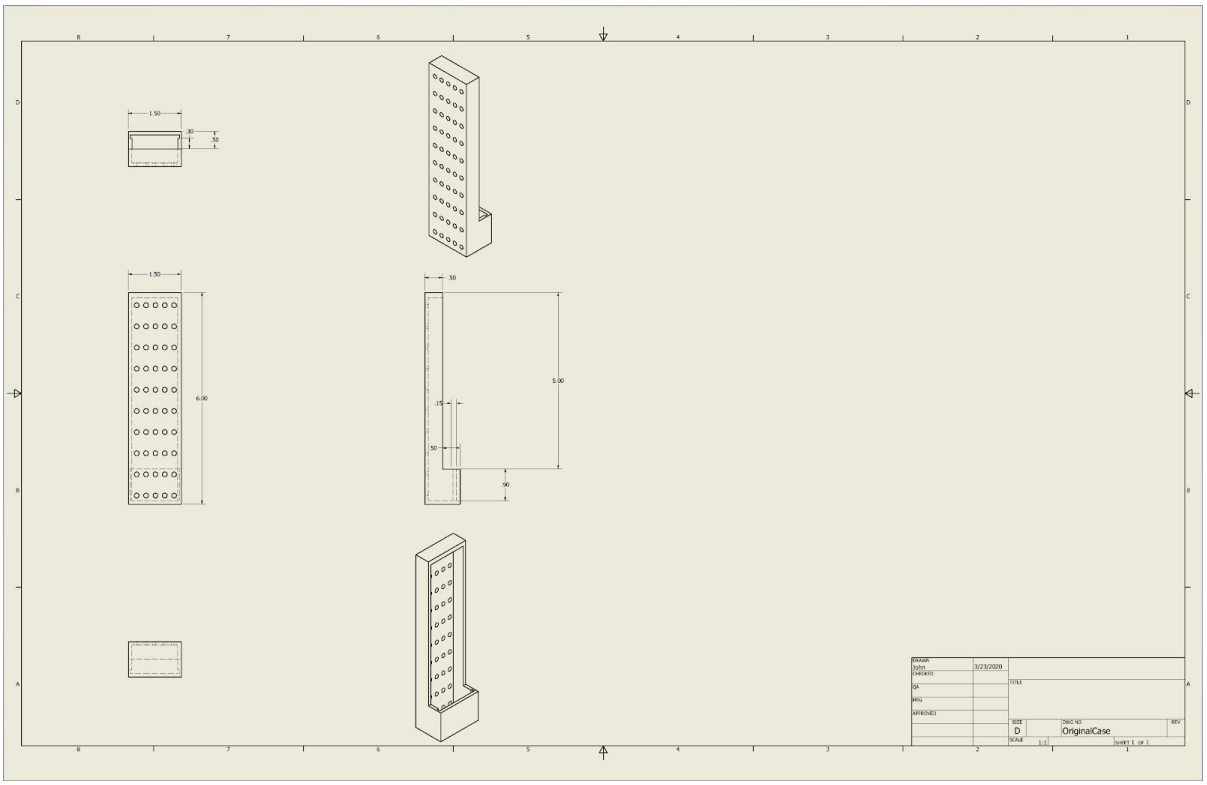


Figure 6.1: LED Master list Case Schematic

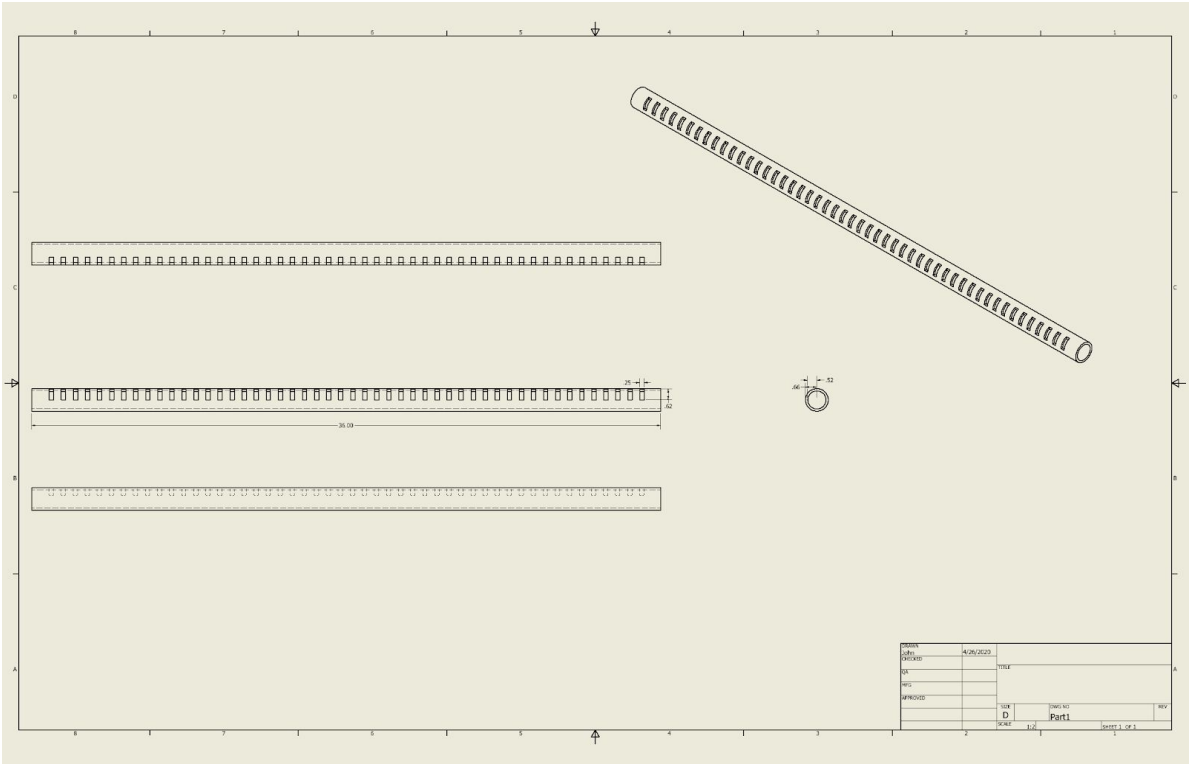


Figure 6.2: LED PVC Pipe Case Schematic

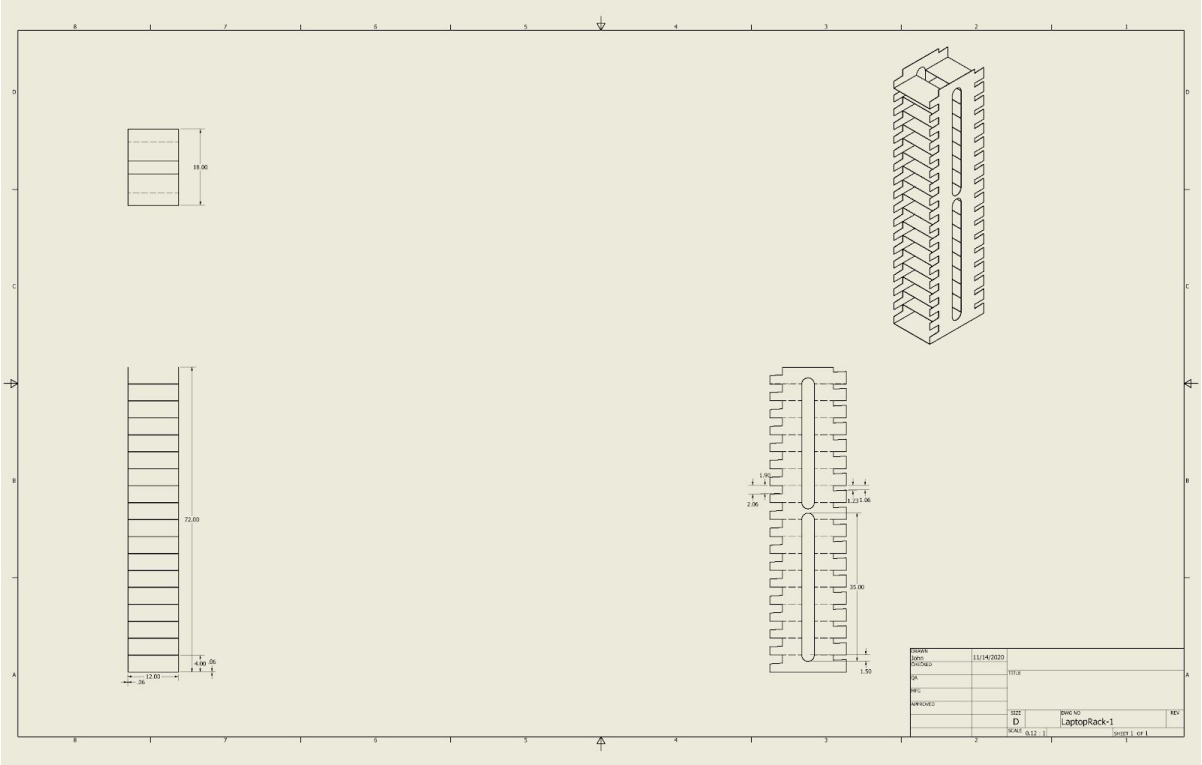


Figure 6.3 Laptop Rack reference model

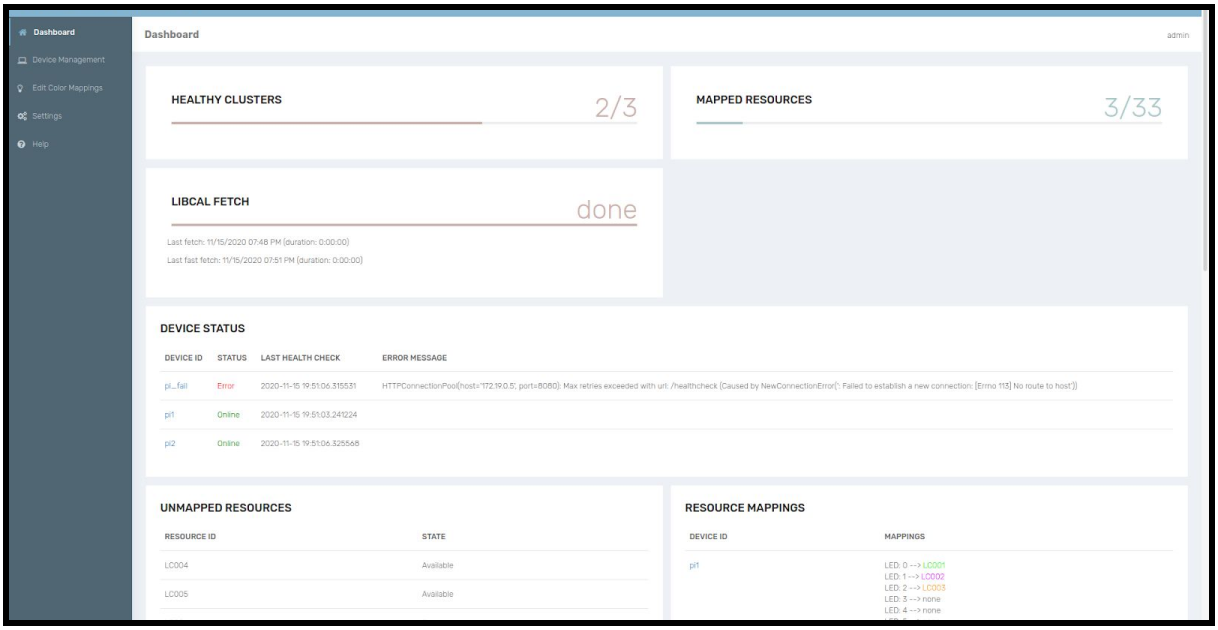


Figure 6.4: Administrative Interface Main Page