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
  - Will control LED driver circuit with a serial connection through GPIO pins
  - Supply power for the Pi and LED driver circuit should come from the PowerGistics racks
- Support for at least 20 RGB LEDs per rack module
  - Due to current draw constraints on the Raspberry Pi GPIO, the LEDs will need to be driven by a separate power source
  - May require full-spectrum control of the LEDs, depending on the color-coding requirements of the library

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 Server that communicates with other Raspberry Pis
    - Master/Slave 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. 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 minute so they could show an old status for up to a minute.
- Our design can only display can only display 8 colors (red, green, blue, cyan, yellow, magenta, white, off/no light)

1.7 Expected End Product and Deliverables

We will deliver a system of computers and LEDs that link to software. A user should be able to modify the status of the LEDs, corresponding to the status of a loanable item. This item will be delivered to the customer on December 16th, 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 December 16th, 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 Raspberry Pi Zero will operate about 20 LEDs). For the bagged equipment on racks our team's tentative plan is to set up a board of LEDs with corresponding device IDs as reference. This provides a more static but still expandable way to track the bagged devices as their positions are more prone to change. 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. As far as software, we could have had each Raspberry Pi run independently, however, this would cause far too many client requests to the LibCal API and overcomplicate the entire process. Thus we decided on a Master server communication with API and each rack Pi.

2.2 Design Analysis

In terms of hardware design, we have created a breadboard prototype of an LED control module using shift registers which store the state of a given LED and control transistors that direct the current flow to the RGB LEDs. This approach shows promise as far as a method of controlling the various LEDs through a serial connection on the Raspberry Pi, but, due to the binary nature of the shift registers, also creates a limitation in the range of colors that we may display in the LEDs. We have discussed this limitation with the clients, who believe that the limited color range would still be sufficient for the purposes of the project. We have also considered solutions that remove the color range limitation, namely using LED driver circuits with built-in PWM (pulse-width modulation) to control the LEDs with a full color spectrum. These driver circuits work very well for controlling the LEDs using the same sort of serial connection, but are more expensive than the simple shift register driver circuit and also come in a surface mount integrated circuit package, which is much more difficult for us, as well as library workers in the future, to solder by hand.

For the software design, we have connected a very basic client to the LibCal API. We use two http servers to communicate between the rack pis and the master server. The master server makes http requests to the raspberry pi to update the leds and do health checks. While the rack pis make http requests to register themselves and initialize their leds.

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

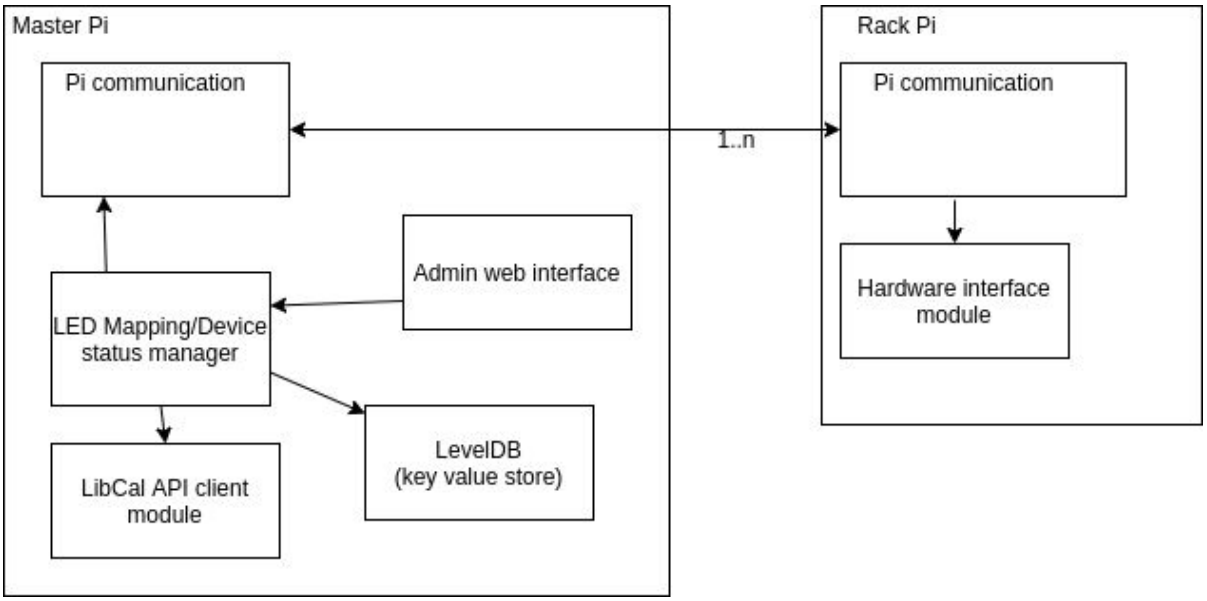


Figure 2.1: Software Architecture Block Diagram

Machines

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

RackPi - The Raspberry Pis running in the individual racks.

Modules

Master server

- LibCal API client modules - Communicates with the libcal api
- API client cache - Stores data to reduce the number of requests sent to libcal
- Led Mapping/Device status manager - Manages status information about the laptops from libcal and maps that to leds on the Rack Pis
- Admin Web Interface - Allows the library to manage the mappings from a libcal id to an led and manage the Rack Pis
- Rack Pi manager handles communicating with the individual Rack Pis

Rack pi

- Rack Pi Management - Communicates with the Master server
- Hardware Interface Module - Controls the individual leds

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 a useful project we found though 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 slaves 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 a key value store 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 Task Decomposition

1. Elicit requirements from clients
   a. Schedule appointment
   b. Ask clients about requirements and record them.
2. Research
   a. Research hardware options
   b. Research software options
   c. Learn how to use software & hardware options chosen.
3. Planning
   a. Create architecture for hardware
   b. Create architecture for software
   c. Create an integration plan for both.
4. Create LibCal API connection software
5. Create LED Mapping Module
6. Create raspberry pi management software
7. Create Web Interface
8. Order needed hardware parts
9. Create LED Device Driver Hardware
10. Integrate Software and Test
11. Create hardware prototype
12. Test hardware
13. Integrate software and hardware

14. Prepare for final demonstration

3.4 Possible Risks And Risk Management

With the recent move to online our biggest concern is working on the hardware component of our project. Since our team is spread across several cities with no access to labs it will make developing our hardware a lot more difficult.

3.5 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.

3.6 Project Tracking Procedures

Using gitlab issues and milestones.

3.7 Expected Results and Validation

Our desired outcome is a fully functional lending status system that can easily be used, configured and expanded.

We will confirm it works at a high level by testing with multiple pis.

4. Project Timeline, Estimated Resources, and Challenges

4.1 Project Timeline

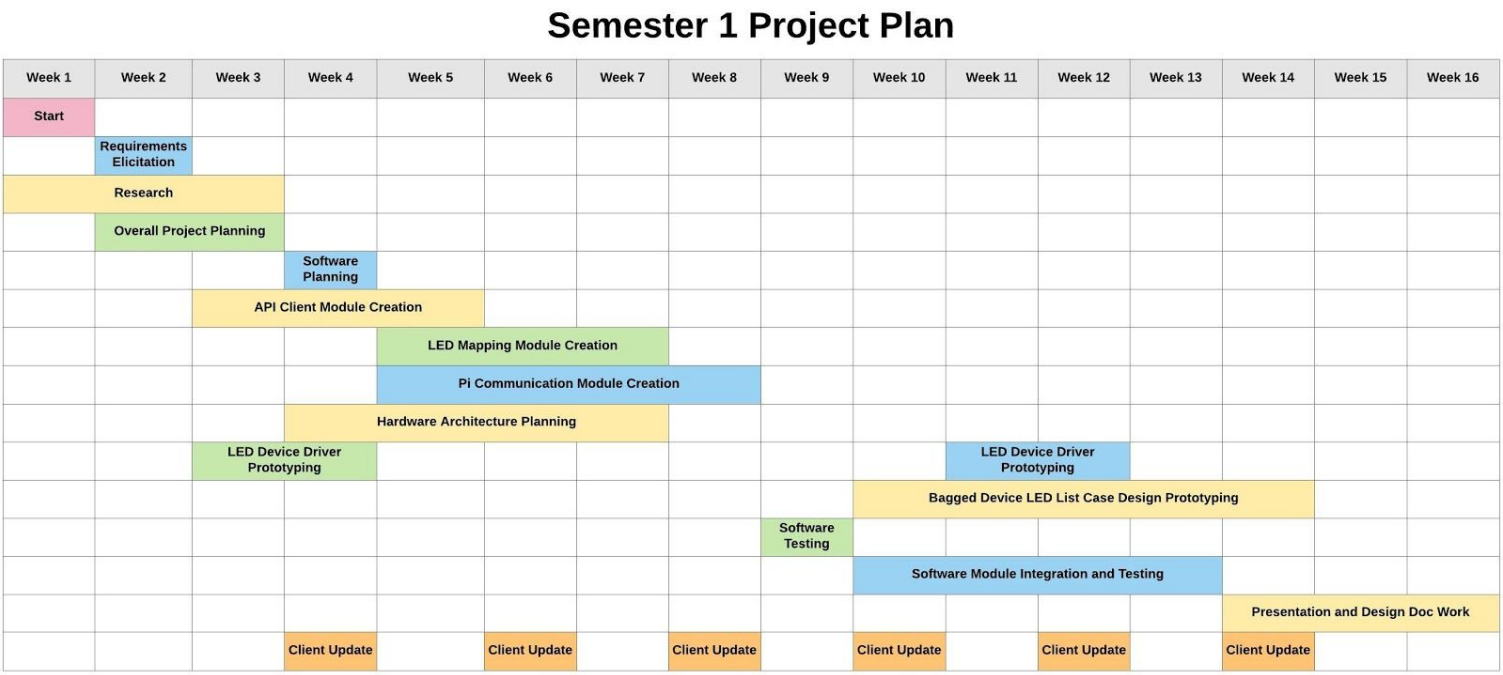


Figure 4.1: Timeline of Semester 1

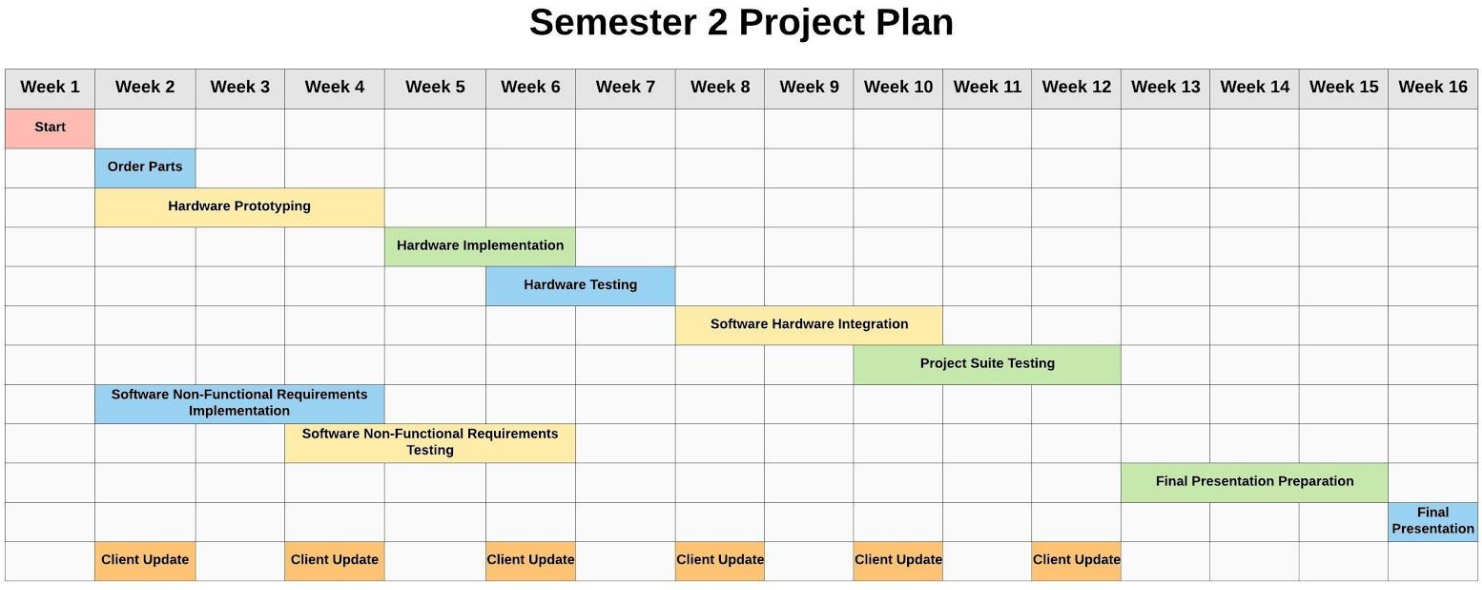


Figure 4.2: Timeline of Semester 2

The Software team has completed the largest functional requirements, and many non-functional requirements by the end of Semester 1, (May 8th, 2020). Due to the COVID-19 outbreak and the transition to online classes, we have yet to order parts for our hardware and coordinate our hardware design. For Semester 2 our team hopes to order all the relevant parts, finish hardware prototyping by integrating it with received parts. By week 7 of Semester 2 we hope to have all Software and Hardware testing completed in order to begin the integration process of the two. After that we hope to finish overall testing of the system by week 13 so our team can prepare for the final presentation.

This timeline is being used because it makes realistic but erring on the side of long time-estimates for each project piece. It also allows plenty of time for testing, especially on the software side as a large amount of the software work has been completed already. Semester 2 being more hardware heavy is also a choice made due to the switch to online courses during Semester 1 which makes hardware development far more difficult

4.2 Feasibility Assessment

By the end of next semester we will have units for the laptop racks and the software to power them. In addition we will have the resources and documentation so that the library can build more units as they add to their inventory.

I expect we will have issues with the individual pis getting out of sync with the libcal resources because any system that relies on making partial changes to its state is prone to issues. To mitigate that we have tried to limit the amount of the code that handles the state to make a smaller surface for bugs.

4.3 Personnel Effort Requirements

Task	Work required
Elicit requirements from clients	This will require several meetings (~3 hours total)
Research hardware options	3 team members. ~4 to 6 hours each.
Research software options	3 team members. ~4 to 6 hours each.
Learn how to use software & hardware options chosen.	Depends on experience (~4 to 12 hours per member)
Create architecture for hardware	Requires various meetings for the duration of the project.
Create architecture for software	Requires various meetings for the duration of the project.

Create an integration plan for both.	Requires various meetings for the duration of the project.
Create LibCal API connection software	Requires implementing and optimizing a libcal api client.
Create LED Mapping Module	Requires implementing database handling logic and logic for send led change events.
Create raspberry pi management software	Requires implementing an http server and making api calls to it.
Create Web Interface	Requires creating frontend and backend for the control interface.
Create LED Device Driver Hardware	Requires making I2C requests to the hardware.
Integrate Software and Test	Will happen continuously over the semester
Create hardware prototype	Requires creating a pcb and soldering the components to it.
Test hardware	Requires verifying that the hardware acts correctly and does not draw too much power.
Integrate software and hardware	Most of this should be handled while creating the driver but there will need to be some debugging of the final product.
Prepare for final demonstration	Fix any loose ends and make it look pretty.

Figure 4.3: Table of personnel effort requirements

4.4 Other Resource Requirements

- A virtual machine
- An ssl certificate

4.5 Financial Requirements

Amount	Item	cost per individual	cost per item
20	RaspberryPi-ZeroW	20 x 10$	200$
20	Micro-USB cables 6ft	20 x 1$	20$
200	RGB LEDs	200 x 0.09$	18$
1,000ft	Electrical Wire	1ft x 0.10$	100$
5kg	3D plastic	1kg x 30$	150$
		Total	488$

Figure 4.4: Breakdown of financial costs

5. Testing and Implementation

Testing is an **extremely** important component of most projects, whether it involves a circuit, a process, or a software library

Although the tooling is usually significantly different, the testing process is typically quite similar regardless of CprE, EE, or SE themed project:

1. Define the needed types of tests (unit testing for modules, integrity testing for interfaces, user-study for functional and non-functional requirements)
2. Define the individual items to be tested
3. Define, design, and develop the actual test cases
4. Determine the anticipated test results for each test case 5. Perform the actual tests
6. Evaluate the actual test results
7. Make the necessary changes to the product being tested 8. Perform any necessary retesting
9. Document the entire testing process and its results

Include Functional and Non-Functional Testing, Modeling and Simulations, challenges you've determined.

5.5 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.

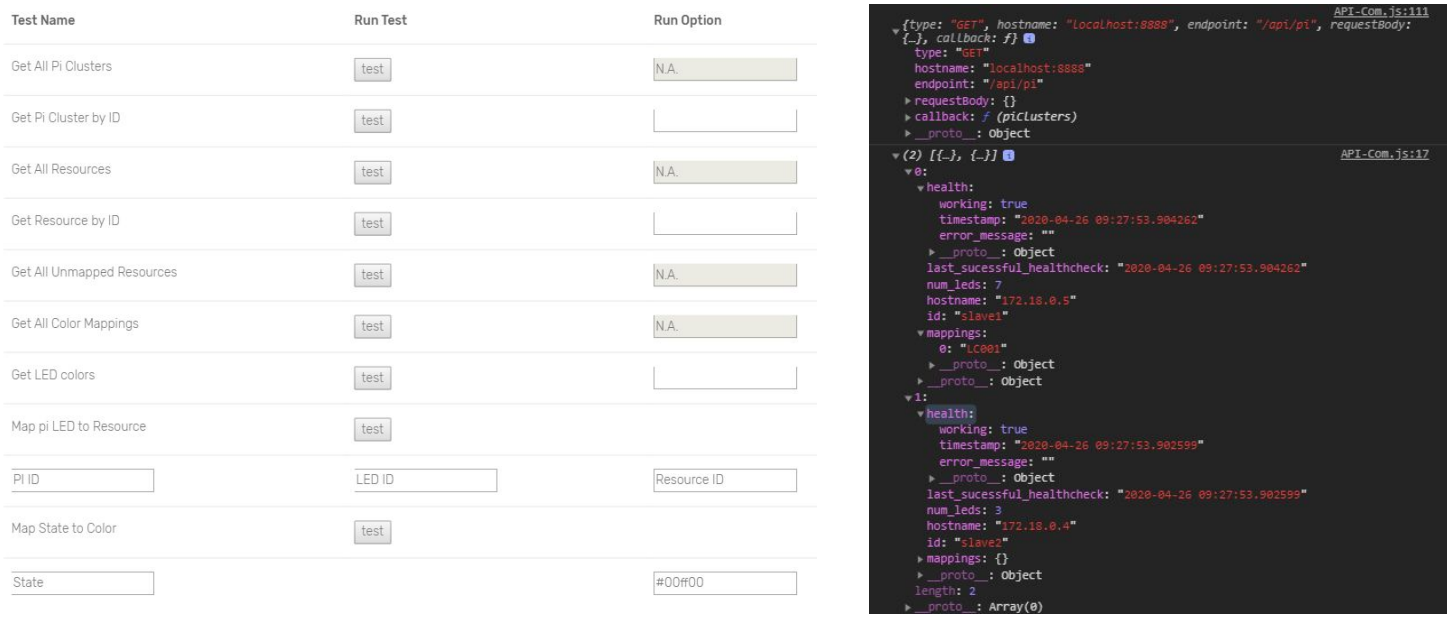


Figure 5.1: Debug Page on Web Interface

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.

Hardware and Software Testing

A breadboard prototype of the LED driving module was developed early on to test options for powering many LEDs with few Raspberry Pi GPIO pins. The breadboard has proved valuable for testing things like daisy-chaining shift registers, powering LEDs via separate power, and as a testbed for the development of our software PWM library in lieu of our final printed circuit board construction.

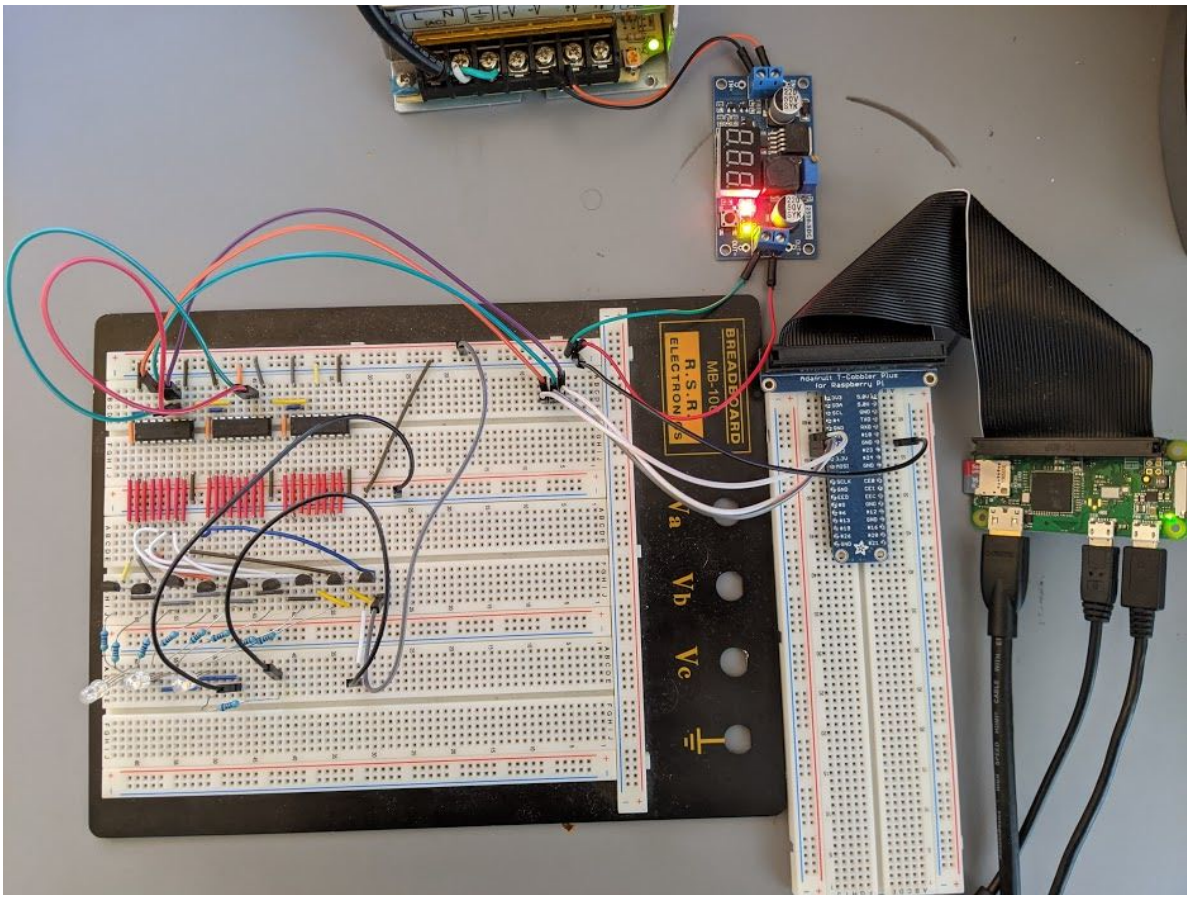


Figure 5.2: Breadboard testing of driving circuit for rack-mounted LEDs

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.

5.3 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.
    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.

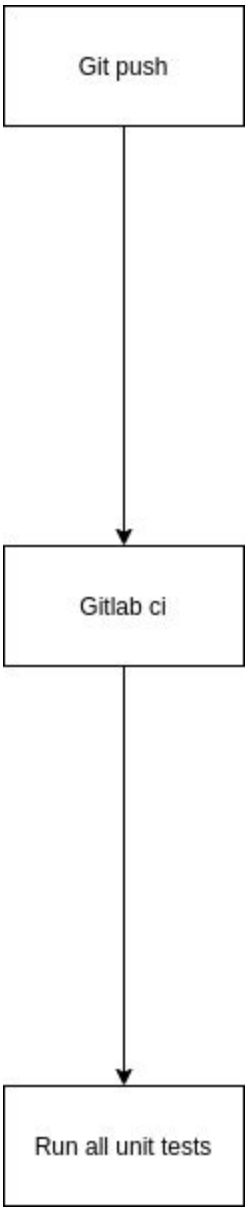
5.4 Non-Functional Testing

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.

5.5 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.



5.6 Results

So far we only have unit tests for the Software side of things. 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.

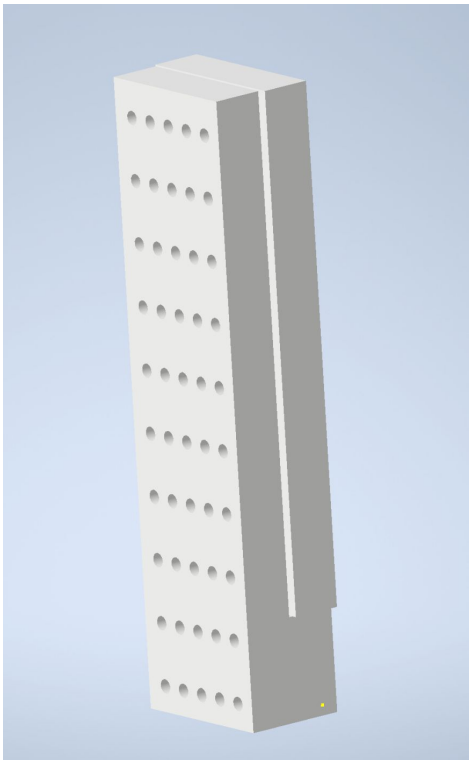


Figure 5.3: Case for LED board - Design 1

This was the initial design for how the LEDs would be set up with the hangers. The major issue with this design was that there was not enough room to label each of the LEDs.

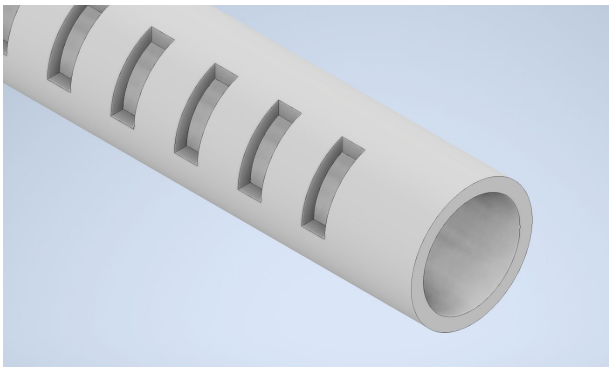


Figure 5.4: PVC pipe case for LED - Design 2

This is a newer design we came up with. It is a pvc pipe that extends across the hanger bar. This should have enough room for each of the LED labels, solving the major issue. From this we learned a lot about how the LEDs will need to be oriented, and displayed to a user.

6. Closing Material

6.1 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 set 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 are debating between two solutions, a masterlist of LEDs with corresponding device ID labels, *or* a PVC pipe fitted onto the bar with slots to hold the LEDs.

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 Master-Slave architecture in which we have a Master-Server with each Pi equipped with a Sub Pi software module. The Master Server houses the following modules, LED Mappings to Devices module, LevelDB database for storing LED mappings, LibCal API Client, Sub Pi Manager, and Admin Web Interface. We went with this solution because it has an intuitive architecture and involves low coupling between modules. This made it somewhat easier for each module creator to work separately.

Overall, our team has made significant progress on our project this semester and we are on schedule to have the project finished by December 2020. Next semester our software team plans to add non-functional requirements to our software and provide more documentation for the software modules. Our hardware team plans to order the needed parts to construct several Sub-Pi LED modules for the racked devices, as well as constructing casing for the bagged device designs.

6.2 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.

6.3 Appendices

Appendix A: 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 B: Schematics & Design Documents

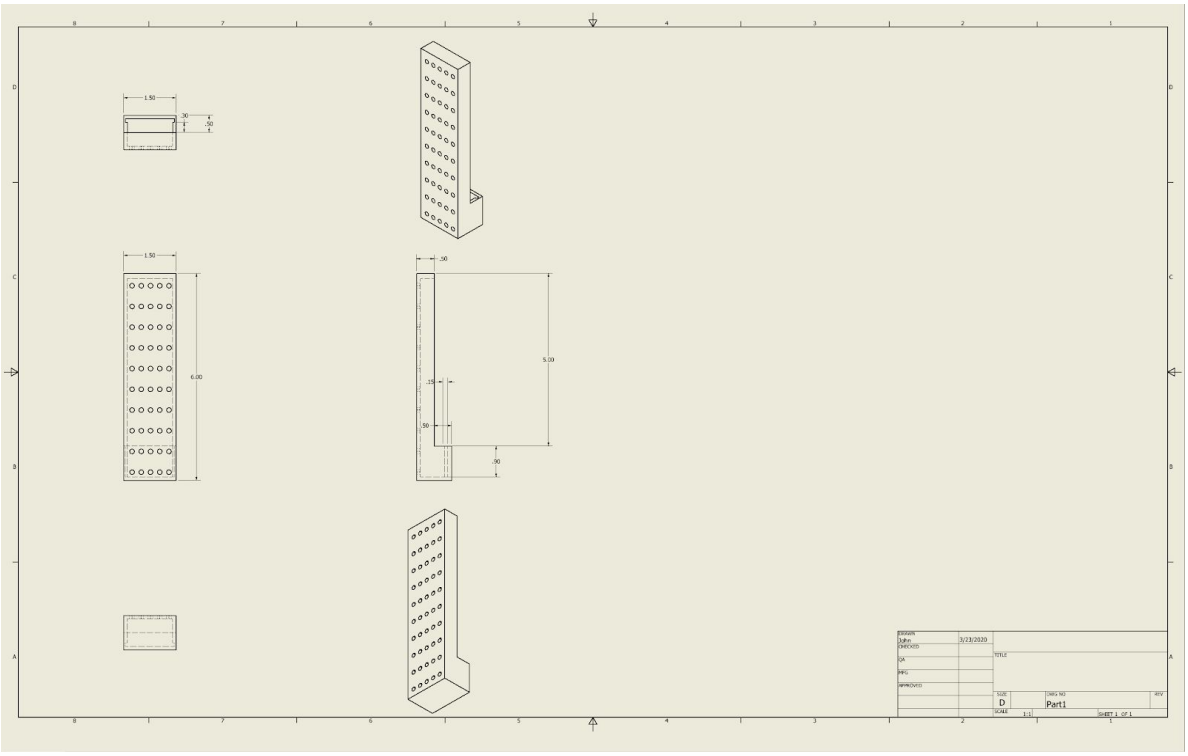


Figure 6.1: LED Master list Case Schematic

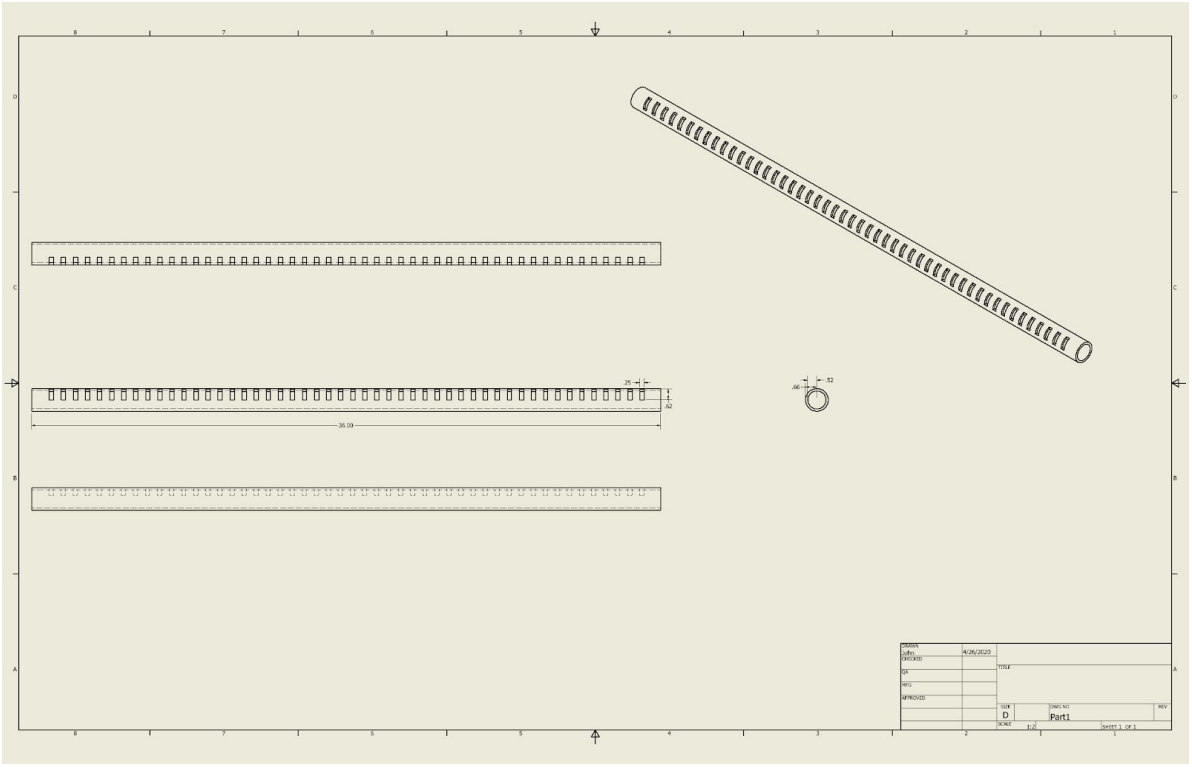


Figure 6.2: LED PVC Pipe Case Schematic