


Department of Electrical Engineering and Computer Science (EECS)

CLU: Connected Locking Unit

Team CE16:

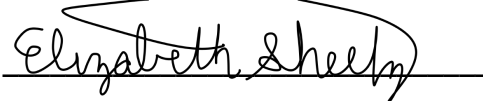
Jonathan Kenney

Submitted in Partial Fulfillment of the degree of Bachelor of Science in Computer Engineering

Team Member Signature: 

Elizabeth Sheetz

Submitted in Partial Fulfillment of the degree of Bachelor of Science in Electrical Engineering

Team Member Signature: 

Technical Advisor's Signature: _____

Dedication

We dedicate this project to those who have seen us through to the successful end of our undergraduate careers.

To our families: thank you for offering your unwavering love as we have ventured on this professional journey.

To the friends we have made on the way: thank you for enriching our lives and pushing us to be the best students and people we can be.

To our co-op mentors: thank you for your patience and allowing us to gain priceless experience applying our skills to real-world scenarios.

To the remarkable EECS faculty and staff: thank you for your guidance, expertise, and care that has kept us on the path and allowed us to realize our academic goals.

To the 1819 Innovation Hub: thank you for your financial and facility support for our project. UC is incredibly fortunate to have your services under its umbrella.

To Dr. Carla Purdy: thank you for your good humored and instructive leadership throughout this senior design experience, especially in light of COVID-19.

To Dr. Joni Torsella: thank you for your encouragement and feedback as we have iterated on this project. We hope we can one day send you a fully functioning CLU unit of your own.

Table of Contents

Dedication	2
Table of Contents	3
Abstract	4
Introduction	4
Problem / Need	4
Solution	5
Credibility	5
Project Goals / Brief Methodology	6
Discussion	6
Project Concept	6
Design Objectives	7
Methodology/Technical Approach	10
Standards	15
Budget	16
Timeline	18
Problems Encountered / Analysis of Problems Solved	20
Future Recommendations	20
Conclusion	21
References	22
List of Figures	23

Abstract

Connected Locking Unit (CLU) is a smart-lock container one accesses from a phone. It can be used by public institutions to provide secure storage on-site, or it can be used in private settings for package delivery. Additionally CLU is scalable, so UC could buy 500 units to supply secure on-campus fridges for student lunches, or someone could buy one for their porch for worry-free package delivery.

Introduction

Problem / Need

The problem statement is:

People need access to scalable, modular locking systems for...

- *Public storage*
- *Handoff solutions*
- *Private delivery*

There are many situations where having access to a modular, connected locking system could benefit people both in public and private environments. From mail delivery to medicine to a place to keep lunch cold, there are many needs that could be met or improved by having a convenient and secure storage unit.

In the case of college campuses, there are many students living off-campus that could benefit from a secured refrigeration unit on campus to encourage packing a lunch. For these students, there can be steep financial and health consequences for eating out too frequently, and many do not have any access to a space to store a lunch to keep it fresh during the day.

There are many products which perhaps solve parts of the problem—lockers on campuses, shared fridges, and even personal fridges that someone could carry around with them. However, these products are neither scalable nor convenient.

Of course, the possibilities of having a system of connected locking units extend beyond refrigeration. Storage of medicine and valuables, package delivery, and item handoff (think key handoff for an AirBnb) all offer common use cases that could benefit from making secure containers accessible and connected.

Solution

CLU is a smart lock. It is special because of its “collective awareness” – users and administrators can locate and control different locking units, or CLUs, on the app. It is connected via a mobile app and is controlled by users and monitored by admins. It is modular so that it can be easily scaled up as a collection of multiple units by the same customer.

In order to address the problem, our team prototyped Connected Locking Unit or CLU. This perhaps seems vague, after all are we not really making a *container*? While the proof of concept involves separate container prototypes, the long term goal would be to offer a modular lock that can be attached to existing containers, with recommended storage options for those not retrofitting their own container.

For context, the team originally set out to solve the specific problem of refrigeration, however, it became clear that the innovation is in the smart lock and software aspects. We do not want to spend copious hours focusing on engineering a fridge—something that appliance companies are already quite skilled at. Instead, we want to enable organizations like universities to provide secure and scalable access to refrigeration. Enter CLU.

Our approach to designing CLU is to view requirements from an objective perspective (*what do we need to accomplish?*) and a technical perspective (*how should we accomplish it?*). To these ends, we have laid out different artifacts highlighting our process and results.

Credibility

With the team members' combined experience in embedded systems, software development, and prototyping, the team is prepared for this project. We have

consulted with Dr. Joni Torsella for guidance in software development and in engineering project management. We did not have previous experience in locking mechanism technologies; but we were able to research options compatible with our design. Finally, we have utilized programs available through the University of Cincinnati 1819 Innovation Hub for funding and prototyping resources.

Project Goals / Brief Methodology

The goals of this project for the term were to successfully demonstrate a proof of concept for our CLU design. This included a minimum value product development for the mobile app, server, and physical device.

We began this process by iterating on our design and adopting an agile workflow. Each subsystem had its own progression, and we would have integration phases where the parts would be put together.

Discussion

Project Concept

In a nutshell, CLU is meant to be the next generation of modular storage. The concept started as an idea to bring accessible refrigeration to college students, but it grew to encompass myriad other possibilities. A driving question became: “where is the innovation?” We came to realize the novelty was not in attempting to redesign a fridge, but the connected technology. While IoT appliances are becoming commonplace, they are often expensive and not scalable for use cases like our college campus.

With all of this in mind, we switched to focusing all our efforts on building a strong design for the software and embedded system, while leaving the container as general purpose as possible. In other words, CLU should work for refrigerators on campuses, but also for package delivery solutions. This extends to other public uses as well, such as escrow and handoff services (e.g., picking up pharmaceuticals), or temporary storage at stadiums and concert venues for non admissible items (e.g., water bottles).

Design Objectives

The design of CLU had multiple desired objects. We used a design tree to narrow down what our guiding objectives would be:

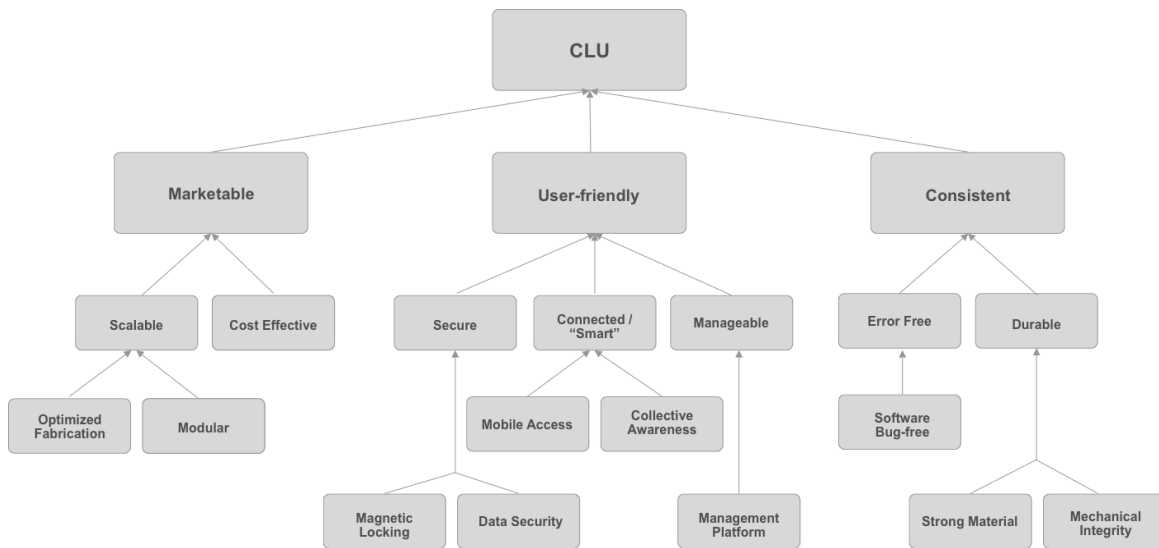


Figure 1: Design Objective Tree

From this, we did a pairwise comparison to rank our key objectives:

	Scalable	Cost	Secure	Connected	Manageable	Consistent	Durable	Total
Scalable	-	0	0	0	0	0	0	0
Cost	1	-	0	0	0	0	0	1
Secure	1	1	-	0	1	1	1	5
Connected	1	1	1	-	1	1	1	6
Manageable	1	1	0	0	-	0	1	3
Consistent	1	1	0	0	1	-	1	4
Durable	1	1	0	0	0	0	-	2

Figure 2: Objective Pairwise Comparison

Ultimately, our core design objectives ranked as follows:

1. Connected
2. Secure
3. Consistent
4. Manageable
5. Durable
6. Cost
7. Scalable

KEY OBJECTIVES



Connected

Connect with the smart CLU network to allow easy public or private access.



Secure

Protect user's contents with hardware and information security and privacy with software.



Consistent

Work seamlessly in the network and repeatedly without error.



Manageable

Allow for easy management for deployments small and large, with care for use.



Durable

Design hardware and software with robust longevity in mind.



Cost / Scale

Sustainable cost with the ability to scale as a given deployment requires.

Figure 3: Key Objectives Overview

These attributes are all important for the development for CLU, but the top four objectives were the focus of the proof of concept. Durability, cost efficiency, and scalability are all objectives we seek to optimize in the next phase of development, where we would like to start scaling up manufacturing of units.

Connectedness was our primary objective. This is the greatest selling point for CLU: its ability to have collective awareness and easy access by the user. Security was a close second. Especially in today's environment of high data privacy sensitivity, we want our

product to have airtight security. Of course, being a locker, guaranteeing physical security is just as important. However until we have more funds to create a more durable container, security concerns are practically limited to the novel software design.

Consistency ranked highly as this product would be difficult to update once deployed. The mobile app may be simple to ship updates for, but deployed units would be a challenge to debug. Therefore, ensuring consistent quality is essential. Manageability also made the top four objectives for this proof of concept, as many of the use cases involve a significant array of units being deployed.

In addition to our key objectives, there are constraints for the POC:

- Cost
 - Cost was listed as an objective and constraint. Firstly, cost matters as it relates to marketability, and this is the meaning when discussing it as an objective. It relates as a constraint as the product, being hardware, will have a significant cost overhead, even in the POC phase.
- Error-free
 - We had error-free as a constraint for the same reason that consistency is an objective; because CLU is deployed into the field, it is essential that it operates in a completely bug-free manner, as deploying firmware updates would be incredibly costly and inconvenient.
- Public Integration
 - A large hurdle for deployment, integrating with the public (largest use cases) was a constraint on development. For instance, striking up contracts with universities in the use case of student access to refrigeration, or with Amazon/UPS/USPS/FedEx/etc. for package delivery.
- System Power
 - While added appliances (e.g. fridge) will be given an opening to plug in, the device is not expected to always have access to an AC power source. Therefore, CLU is limited by its ability to power itself. This is why we have power efficiency as an important development constraint.

Methodology/Technical Approach

Before getting into the details of implementation, we first analyzed the high level interaction of users with the product, and the internal components with each other. To this end, we first created a user diagram:

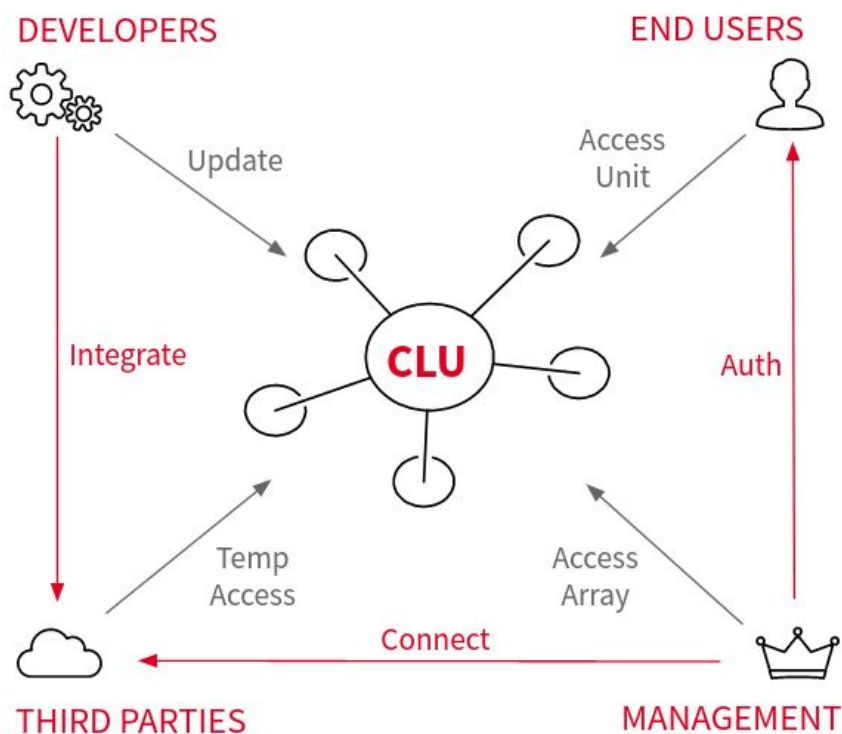


Figure 4: CLU User Diagram

Here there are four main users summarized: the developers, third parties, management, and end users. Each user has their own unique interactions with the CLU array that are to be considered. Additionally, there are interactions between each user with other users. These interactions include:

- **Developers** build **third party** integrations
- **Managers** enable **third parties (e.g., Amazon)** on their platform

- **Managers** grant **users** permissions for use (e.g., UC grants all UC students access)

Understanding how these interactions need to be supported, we next started designing the implementation. At the high level, we designed a black box diagram to show the main systems foci:

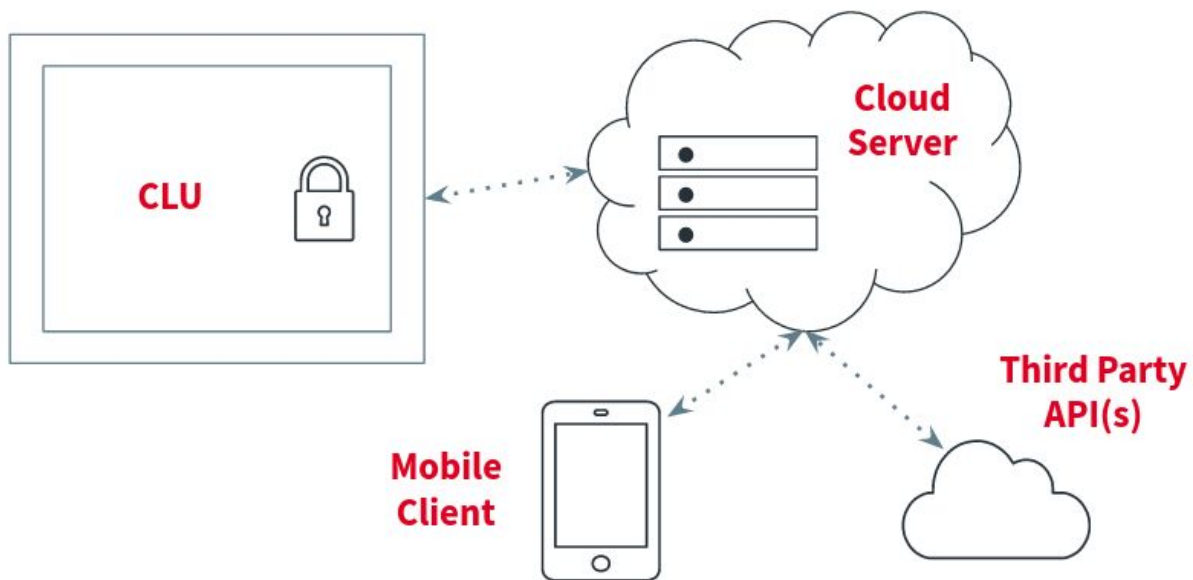


Figure 5: Black Box Diagram

Separating the systems into these categories allowed us to decide how we would allocate resources and make sure each system was ready at various phases of integration. More specifically, Elizabeth was able to take charge of building the physical systems for the CLU device, while Jonathan was able to focus on the mobile and server-side components.

The next step of the design was to detail what components would be required in each system, as well as how they would communicate. For this, we constructed a white box diagram that kept the high level structure of the black box:

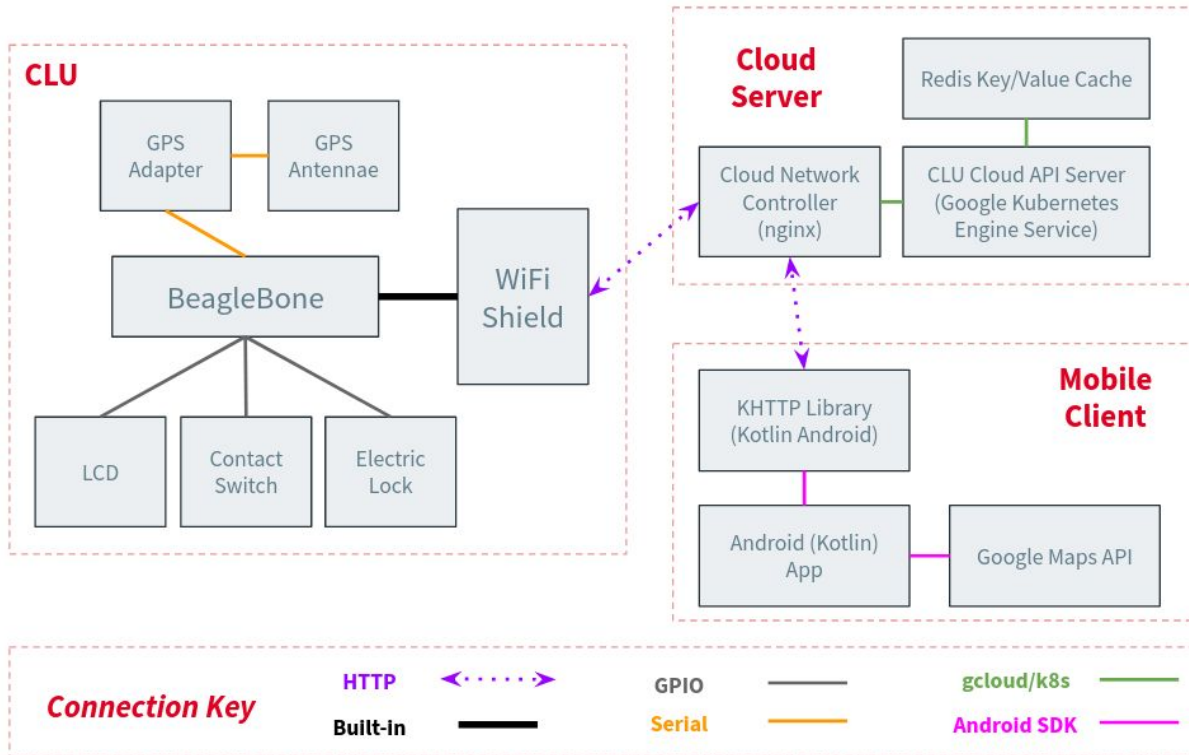


Figure 6: White Box Diagram

This diagram shown here was updated as we were able to get more precise with our design. For example, we were able to update MCU to BeagleBone after we trade studied various options for our MCU.

With the specifics drawn up and an approved budget (as seen in Figure 11 later on), we began working on implementing our vision. This started mostly with software, as parts needed to be ordered. As we will discuss in challenges, many parts never came due to the coronavirus. Nonetheless, we were still able to show POC on the app and server-side implementations.

The app is designed in Kotlin, and uses the Google Maps API to build a map overlay. The functionality of the app is not fully realized yet, but currently it can fetch active units from the server's redis cache. This is shown with the following API call:

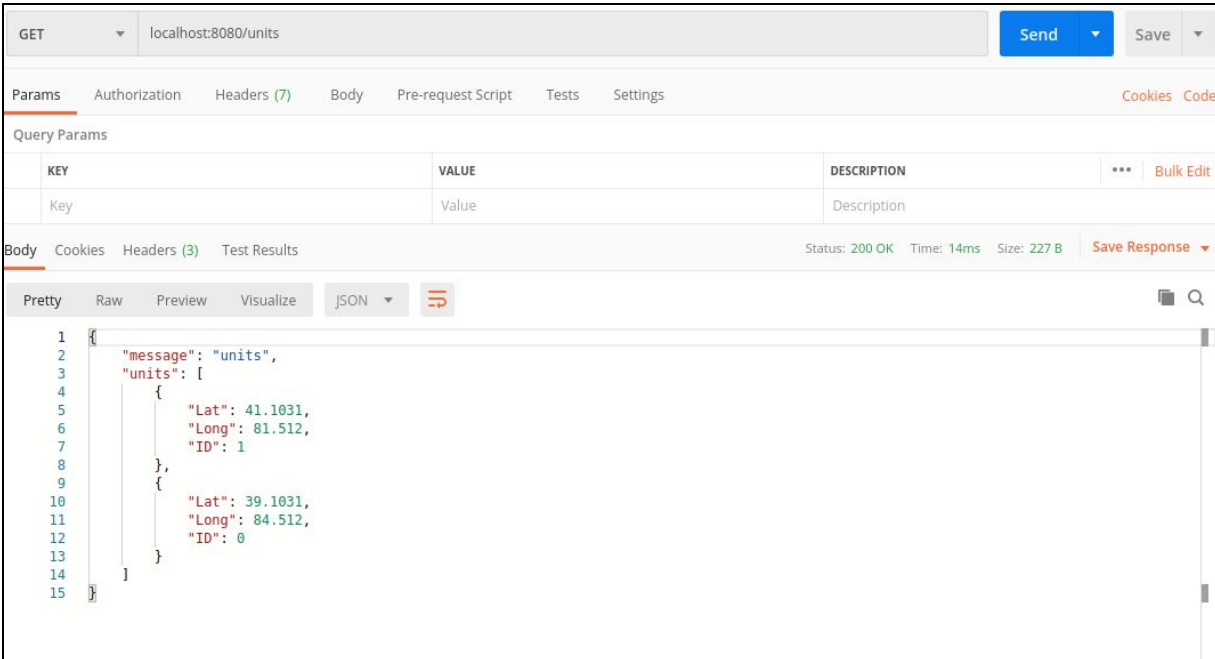


Figure 7: API Call To List Units (Demoed In Postman Here)

After the app makes this call, it can overlay the nodes on the Maps frame:

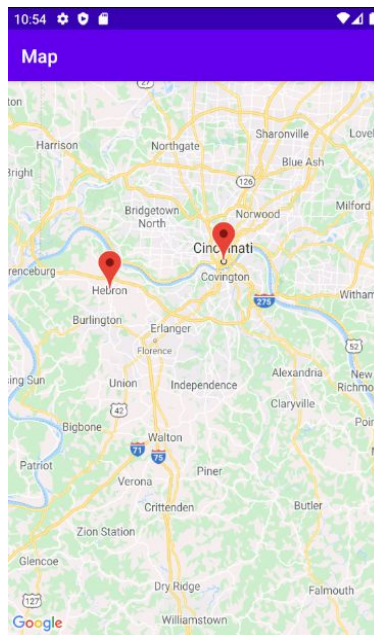


Figure 8: CLU Units on Google Map (Demoed In Android Studio Emulator Here)

Additionally, the CLU units need the ability to create and update their records in the redis cache. For this, there is another API call:

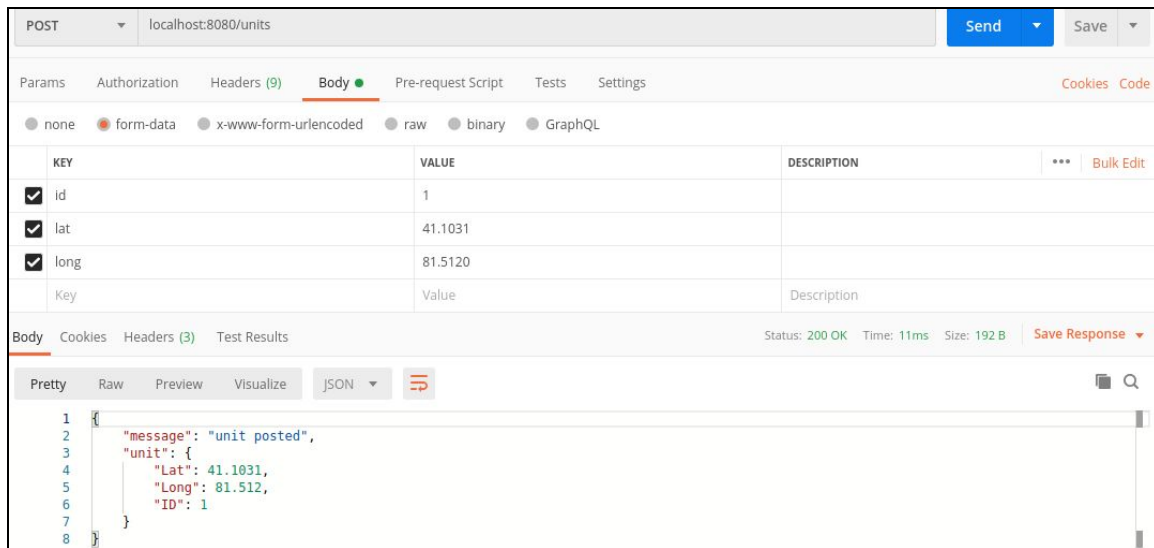


Figure 9: API Call To Create Unit Records (Demoed In Postman Here)

Currently, this API call only allows for record creation. One next step already in progress is to create a call that a node would run relatively frequently to update its record (in case the location changes).

For the POC, we had originally planned to run these tests on a Google Kubernetes Engine cluster, however the credits we had to pay for the cluster expired. Therefore, we instead ran the server locally for the POC:

```

jonathan@clu-backend$ ./clu-backend
[GIN-debug] [WARNING] Creating an Engine instance with the Logger and Recovery middleware already attached.
[GIN-debug] [WARNING] Running in "debug" mode. Switch to "release" mode in production.
- using env:   export GIN_MODE=release
- using code:  gin.SetMode(gin.ReleaseMode)

PONG <nil>
[GIN-debug] GET    /ping          --> main.main.func1 (3 handlers)
[1 0]
[GIN-debug] GET    /units        --> main.getUnitsHandler.func1 (3 handlers)
[GIN-debug] POST   /units        --> main.setUnitsHandler.func1 (3 handlers)
[GIN-debug] Listening and serving HTTP on localhost:8080
[GIN] 2020/04/07 - 22:51:14 | 200 | 320.911µs | 127.0.0.1 | GET    "/units"
[GIN] 2020/04/07 - 22:52:33 | 200 | 545.573µs | 127.0.0.1 | POST   "/units"

```

Figure 10: Live API Server (Running Locally Here)

Standards

Considering the design objectives and requirements, the team developed the following list of standards to which CLU should adhere:

- RoHS
- Data encryption
- SSL Verification
- UL
- IP67 Waterproofing.

These requirements are essential to the basic safety and functionality of the product. The product must work, its users must be safe, and their privacy must be protected.

RoHS standards protect the environment and the people who are near the product by restricting the use of certain hazardous materials. Hazardous materials pollute the environment and landfills. They also harm employees involved in manufacturing products containing them. Currently RoHS standards have been met because all components purchased are RoHS compliant (“RoHS Compliance FAQ”, 2020).

Data encryption and SSL verification are both necessary for our goals of security and user privacy. This means that any user info in transit or at rest needs to be properly encrypted. Additionally, all messages should be properly signed to avoid any potential man-in-the-middle attacks on the networking. Finally, additional measures need to be taken to ensure that the IoT devices are secure against malicious hardware attacks and device accessible network attacks.

In the future, CLU should be UL Listed. According to C3Controls.com,

“In a nutshell, UL is a safety organization that sets industry-wide standards on new products. They continually check these products to ensure they’re up to these standards. UL testing makes sure that wire sizes are correct or devices can handle the amount of current they claim to be able to. They also ensure that products are constructed correctly for the highest safety” (2019).

UL Listed requires the entire product to be evaluated and approved. Achieving UL Listed is an important step in ensuring the physical safety of CLU users.

IP67 standards are essential to the functionality of CLU. If it is labeled IP67, the dust resistance rating is 6, which is the highest rating, and the water resistance rating is 7, which means the device can survive 30 minutes in less than 1 meter of water. (Illini Gadget Garage, 2017). If the product is used outdoors and it rains, or if a drink is spilled inside a refrigerator locked by CLU, then CLU needs to continue operating. Therefore, it must be water resistant.

Budget

Funding was received from two separate entities at the University of Cincinnati. The EECS Department contributed approximately \$175 to the project. The project was also awarded \$300 in credit at the Ground Floor Makerspace + Microfactory at the 1819 Innovation Hub (iHub) via the iHub's NEXT STEP program.

The team budgeted for building two CLU unit prototypes. The original estimated cost for the project was \$1,225. After specific parts were selected, the new projected cost for two units became \$593. Significant factors in the cost savings were in the cost of the locks and sensors. In addition, most hardware overhead is covered for free by the NEXT STEP program in this stage of the project. The original budget and new projection are detailed in Figure 11. To date, the team has spent \$250. Actual spending is detailed in Figure 12.

The budget for the project is categorized into software costs, hardware costs, and container costs.

Software costs include cloud service and app registration. Software costs do not increase with the number of CLU units created; however, they would increase with the amount of data storage and server traffic that was needed. Costs are estimated for the low, basic level needed for prototyping and demonstration. If the project was deployed to market, software costs would increase. At this time, no costs have been incurred in the software category.

Hardware costs include the control system, locks, sensors, and any other hardware needs for the project. Hardware costs are a variable cost; they increase with the number of units built. At this time, two beaglebones and two locks have been purchased, and one GPS system was purchased. Specific products for other items have been priced but not purchased.

Finally, container costs are for the purchase of pre-fabricated products onto which CLU would be installed. Two primary applications for CLU are 1) a refrigerated locker and 2) a package handoff locker. Therefore, the team budgeted for a purchase of a mini refrigerator and a locker in order to demonstrate each of these applications. The products have been priced but have not been purchased.

Category	Component	Budgeted	Projected
Software Costs			
App	Cloud Server (est.)	\$ 100.00	\$ 100.00
App	Open Source Software	\$ -	\$ -
App	Google Play Store Registration (est.)	\$ 25.00	\$ 25.00
Total Software Costs		\$ 125.00	\$ 125.00
Hardware Costs			
Control System	Beaglebone Black Wireless	\$ 100.00	\$ 81.50
Locking Mechanism	Electric Door Lock	\$ 200.00	\$ 12.11
UI	LCD Screen	\$ -	\$ 18.95
Hardware Overhead	Tools, Wires, Materials	\$ 100.00	\$ -
Sensors		Sensors	
Sensors - GPS	GPS Antenna		\$ 16.00
Sensors - GPS	GPS Breakout		\$ 42.75
Sensors - GPS	GPS Connector Adapter		\$ 4.23
Sensors - Door Switch	Reed Switch		\$ 1.45
Sensors - Door Switch	Magnetic Ring		\$ 0.95
Sensors - Total	Sensors - Total	\$ 150.00	\$ 65.37
Total Hardware Costs per Unit - Electronics		\$ 550.00	\$ 177.93
Total Hardware Costs for 2 Units		\$ 1,100.00	\$ 355.86
Container Options (Purchase 1 each)			
Containers	Iron 16"x12"x23" Storage Locker	\$ -	\$ 63.89
Containers	12V 4L Mini Fridge	\$ -	\$ 47.91
Total Container Costs		\$ -	\$ 111.80
Total Project Cost for 2 Locks and 2 Containers		\$ 1,225.00	\$ 592.66

Figure 11: Original Budget and Projected Budget

Category	Component	Qty Purchased	Cost Per 1	Total Spend
Software Costs				
App	Cloud Server (est.)	0	\$ 100.00	\$ -
App	Open Source Software	0	\$ -	\$ -
App	Google Play Store Registration (est.)	0	\$ 25.00	\$ -
Total Software Spending				\$ -
Hardware Costs				
Control System	Beaglebone Black Wireless	2	\$ 81.50	\$ 163.00
Locking Mechanism	Electric Door Lock	2	\$ 12.11	\$ 24.22
UI	LCD Screen	0	\$ 18.95	\$ -
Hardware Overhead	Tools, Wires, Materials	0	\$ -	\$ -
Sensors	GPS Antenna	1	\$ 16.00	\$ 16.00
Sensors	GPS Breakout	1	\$ 42.75	\$ 42.75
Sensors	GPS Connector Adapter	1	\$ 4.23	\$ 4.23
Sensors	Reed Switch	0	\$ 1.45	\$ -
Sensors	Magnetic Ring	0	\$ 0.95	\$ -
Total Hardware Spending				\$ 250.19
Container Options (Purchase 1 each)				
Containers	Iron 16"x12"x23" Storage Locker	0	\$ 63.89	\$ -
Containers	12V 4L Mini Fridge	0	\$ 47.91	\$ -
Total Container Spending				\$ -
Total To-Date Spending				\$ 250.19

Figure 12: To-Date Spending

Timeline

A gantt chart was used to manage the project timeline. The gantt chart is organized into three major sections: Project Management, Software Engineering, and Hardware & Controls Engineering.

The Project Management section, as shown in Figure 13, is broken down into Fundraising, Documentation, Poster, and Design. Each of these sub-sections were conducted in parallel. The milestone dates for Documentation, Poster, and Design activities were imposed by classroom activities, while the Fundraising timeline was imposed by the Next Step program.

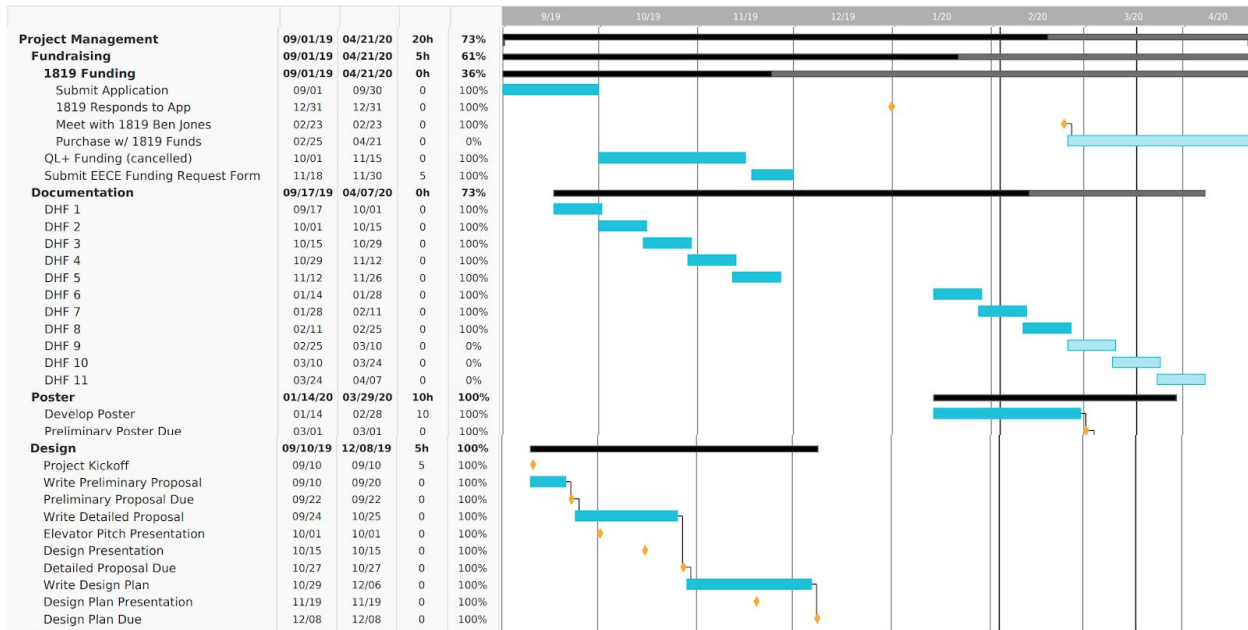


Figure 13: Gantt Chart of Project Management Activities

Once design was completed, development could begin. The development schedule is represented in the two Engineering sections, as shown in Figure 14. Each team member was assigned one of the sections, which is why they are represented separately. The team decided to follow an agile-style process, with four sprints. The goals for each sprint were: Rough & Rapid Prototyping, Minimum Value Product, UI/UX, and Feature Enrichment. At the end of each sprint, an integration period was scheduled to ensure the software and hardware “departments” continued to communicate and create compatible products.

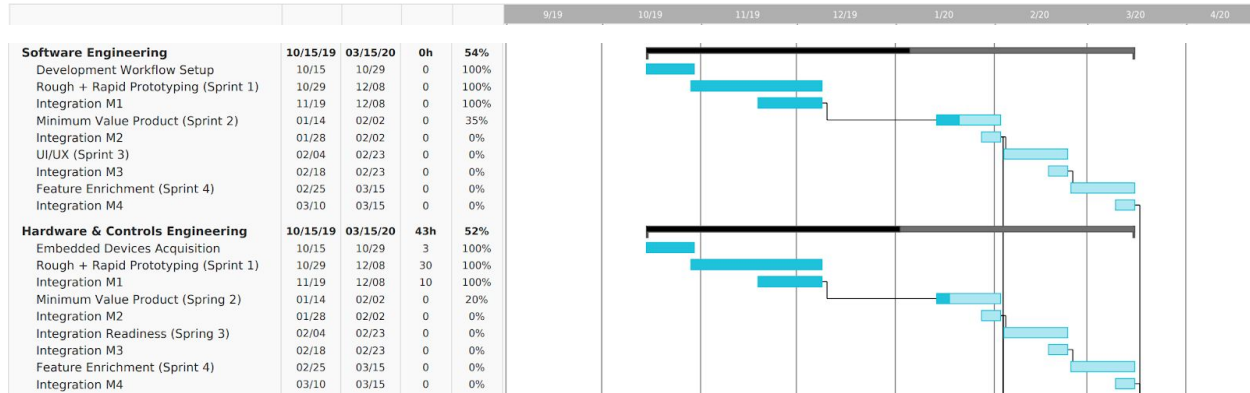


Figure 14: Gantt Chart of Engineering Activities

Problems Encountered / Analysis of Problems Solved

For our design, we encountered a problem of scope. We were trying to solve multiple technical challenges to achieve a specific goal. By simplifying the design and focusing on the major points of innovation, we actually managed to expand our use cases far beyond just modular refrigeration on college campuses.

With hardware, we had a few hurdles. Many of the parts we ordered in our first shipment came in defective. However thanks to the ability to test the components at the iHub, we were able to discern what needed replacements. Unfortunately, COVID-19 has put a hold on actually acquiring these parts and our ability to access the fabrication tools of the iHub.

The software side was most hindered by complexity. Due to the changed nature of the project and the desire to create a POC without all the parts, the software was simplified. This means that some features like security standards and third-party integrations are delayed. However, we still managed to achieve a demonstration that our concept is strong.

Future Recommendations

Due to the complications from COVID-19, the team can provide several future recommendations. First, once the remaining required parts are procured, individual parts should be tested for functionality, and then the hardware should be fully

integrated and tested. The software should be moved from a test stack and onto a paid GCP cloud service, which would enable Redis and cluster networking. In parallel, the containers need retrofitting via 3D printed parts, then the CLU locks and sensors can be affixed to the containers. The minimum safety standards should be achieved. Then, the Android app can be deployed on the Google Play Store. This would constitute a working prototype. Future work after this would include feature enrichment, UI/UX improvements, 3rd party integration, and progress toward further safety and functionality standards. The working prototype can be used to demonstrate the product to incubator programs and other sources of mentorship and funding, where the product could be further developed and brought to market.

Conclusion

- The goal of designing a connected, modular, scalable locking system was achieved.
- The goal of prototyping the system was not achieved due to the COVID-19 crisis.

References

- c3controls. "What Is UL Certification? The Difference Between UL Recognized and..."
c3controls.com, c3controls, 31 July 2019,
www.c3controls.com/blog/the-difference-between-ul-recognized-and-ul-listed/.
- Illini Gadget Garage. "Splish Splash, the IP Ratings Bath." Illini Gadget Garage,
University of Illinois at Urbana-Champaign, 30 May 2017,
illini-gadget-garage.istc.illinois.edu/tag/ip-rating/.
- "RoHS Compliance FAQ ." RoHS Guide, Rohsguide.com, 28 Apr. 2020,
www.rohsguide.com/rohs-faq.htm.

List of Figures

Figure 1: Design Objective Tree	7.0
Figure 2: Objective Pairwise Comparison	7.0
Figure 3: Key Objectives Overview	8.0
Figure 4: CLU User Diagram	10.0
Figure 5: Black Box Diagram	11.0
Figure 6: White Box Diagram	12.0
Figure 7: API Call To List Units (Demoed In Postman Here)	13.0
Figure 8: CLU Units on Google Map (Demoed In Android Studio Emulator Here)	13.0
Figure 9: API Call To Create Unit Records (Demoed In Postman Here)	14.0
Figure 10: Live API Server (Running Locally Here)	14.0
Figure 11: Original Budget and Projected Budget	17.0
Figure 12: To-Date Spending	18.0
Figure 13: Gantt Chart of Project Management Activities	19.0
Figure 14: Gantt Chart of Engineering Activities	20.0