Makin' The Makeline – Pizza Locker

ECE4873 Senior Design Project

The Neatest ECE Team
Sd22p10
Dr. Xiaoli Ma
Papa John's International

Annie Liu	Damian Huerta	David Wechsler	Dennis Crawford	Leah Jackson
CmpE	CmpE	EE	CmpE	EE
aliu321	dhuertaortega3	dwechsler6	dcrawford41	ljackson74
aliu321@gatech.edu	damhue039@gatech.edu	davidwechsler@gatech.edu	dennis.crawford@gatech.edu	ljackson74@gatech.edu

Submitted

2022 May 3

Table of Contents

Executive Summary **3**

1. **Introduction** **4**

   1.1 Objective 5
   1.2 Motivation 5
   1.3 Background 6

2. Project Description and Goals 7

3. Technical Specification & Verification 10

4. Design Approach and Details

   4.1 Design Approach 11
   4.2 Codes and Standards 31
   4.3 Constraints, Alternatives, and Tradeoffs 12

5. Project Demonstration 31

6. Schedule, Tasks, and Milestones 33

7. Final Project Demonstration **32**

8. Marketing and Cost Analysis **35**

   7.1 Marketing Analysis 35
   7.2 Cost Analysis 36

9. Conclusion **37**

10. Leadership Roles **39**

11. References **39**

Appendices **40**

Executive Summary

The main objective for this project is to design a product that allows customers to efficiently pick up warm pizza orders in less time and more conveniently than by staying in line and waiting for an employee. Workers in Papa John's International have stated their intentions of getting people in and out of the store within 3 minutes. On average, it currently takes at least 6 minutes to do so. In order to allow customers to have a faster and better experience with ordering their pizza, we proposed an automated heated lockers that are connected to an application. A more efficient system would highly benefit Papa John's by allowing employees to work on other tasks besides handing orders to customers who have already paid.

This application will allow customers to order their pizza, then retrieve a barcode that will act as a key to opening a locker to their fresh pizza. On the employee side, the application automatically selects and unlocks an empty locker for the worker to place the corresponding pizza order inside. Main performance specifications include delivering a power efficient device that is user-friendly and successfully maintains pizzas within a desired temperature range.

The total cost of designing, developing, and building the initial prototype was $910. The project was demonstrated live at the Spring 2022 Capstone Design Expo, where over a hundred volunteers tested the locker's functionality. Next steps for this project include simplification of design to allow for potential mass production of the device; size expansion of individual lockers to enable more orders of more than three pizzas to be stored in the same locker; and addition of more lockers in the device.

Nomenclature

API – Application programming interface

GUI – Graphical user interface

GPIO – General-purpose input/output

IO – Input/output

OOP – Object-oriented programming

PWM – Pulse width modulation

SPI – Serial peripheral interface

Automated Pizza Locker

1. Introduction

The Neatest ECE Team has requested funding of approximately $910 to design, develop, and build an automated pizza locker to improve customer's experience at Papa John's International stores. A detailed breakdown of each design cost is also available under *Marketing and Cost Analysis*.

Motivation

In February 2022, our team visited multiple Papa John's locations in Atlanta. A common occurrence across multiple stores was the staff's desire to better optimize its pickup order system. At one location, it was mentioned that the employees aimed to decrease customer time in the store from the current average time of 6 minutes to less than 3 minutes – a reduction of at least 50%. Given this scenario, our team has decided that the use of an automated order pick-up system could vastly help Papa John's reach this goal.

Objective

In order to minimize customers' waiting times when picking up orders at Papa John's, the Neatest ECE Team proposed the adoption of a locker device to enable instant and autonomous collection of orders by the customer. Similarly, the creation and development of a smartphone application to coordinate such lockers is also within the scope of this project.

The intended users of the proposed locker are customers picking up pizza orders in-store, as well as employees placing orders in the locker. The device will contain several trays that will keep the pizza warm, which can only be unlocked by their corresponding customers, or an employee placing an order. Our team has created one application exclusively for the employees –

to coordinate orders being placed in empty lockers –, and one application where the customer will place an order, and consequently retrieve it once it has been placed in the locker. Such applications interact with our physical locker and its circuitry to deliver an all-in-one solution for Papa John's International. We anticipate that our device could be placed near the entrance or counter area of the stores to facilitate a fast and simple experience for the all intended users. Additionally, we would like to note that it would be possible to use the locker device to store other types of orders such as cookies, if the temperature requirements for that specific item are met. This project, however, describes the design of a prototype that focuses primarily on pizza orders.

The Neatest ECE Team is committed to diversity of thought, determination, and effort. This project was closely guided by such principles. We are devoted to delivering a project that will help both Papa John's International as well as its customers by improving the overall pizza pickup experience. We understand the difficulty of this task and always anticipated challenges such as limitations on technology given our budget constraints. However, we firmly believe that abiding by our core values will facilitate us to meet our end goals.

Background

The pizza locker device here proposed presents a solution to the problem statement described that entails well founded knowledge of software/hardware creation and development, as well as its integration. With that in mind, our team has split up tasks in order to fully optimize every group member's skills and abilities based on previous experience and existing knowledge. However, this project require understanding of new concepts, something that could only be achieved with continual learning. This document lists all external material used under its *References* section.

While our device has undergone through various stages of trials, its final demonstration was conducted in the official Spring 2022 Capstone Design Expo organized by the Georgia Institute of Technology, held on April 26th, 2022 at the McCamish Pavillion. We anticipate that over one hundred volunteers tested our device through prototype application. More information about the event can be found in *Project Demonstration* section.

This document provides an in-depth analysis of the project proposed, including customer requirements, technical specifications, design approach, project schedule, cost analysis, and more.

2. Project Description, Customer Requirements, and Goals

The team will be designing a locker and application that enables customers to directly access their order in a fast and efficient manner. This locker will have a scanner, where the customers will have to scan a unique barcode that they will retrieve once they are done ordering through the application. Once this code is scanned, the locker is opened, and the customer can retrieve their pizza. The order will contain information such as approximately what time their pizza will be ready and notify you when your pizza is hot and ready to be picked up in the locker.

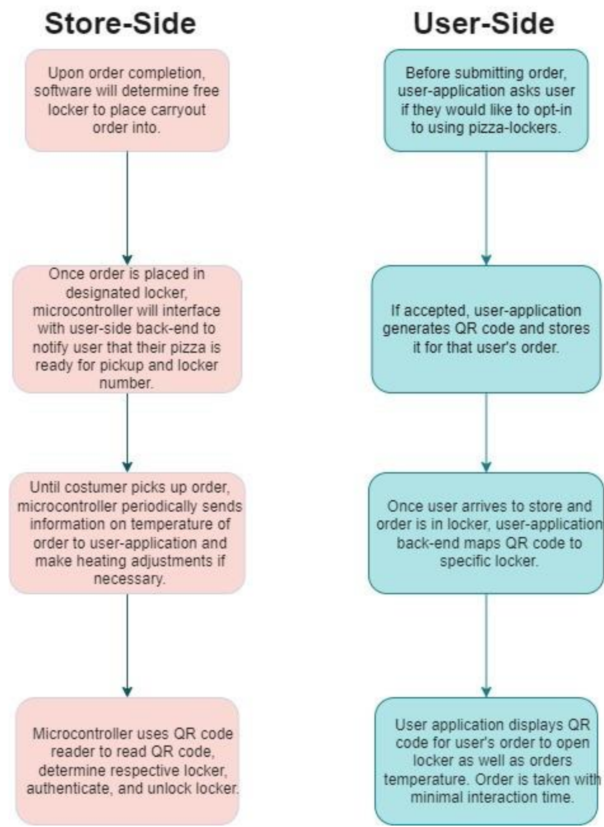


Figure 1. User-Side & Store-Side flow chart of locker/app functionality

Customers who are frequent pizza consumers and who enjoy a convenient way of accessing fresh pizza will be using this product. Customer needs for this project includes a free highly assessable application that is easy to use and straight forward. The application and locker access should be free and tell you at what location to pick up your pizza. At a typical Papa Johns, it is ideal for customers to be in and out within 3 minutes, however, because of many delays the process usually takes 7 to 8 mins. Our goal for this project is to have customers in and out in less time. The targeted price for the product will be close to $1000.

The design of the locker should allow the pizza to continually stay warm at a consistent temperature from 140 degrees Fahrenheit to 150 degrees Fahrenheit. To ensure quality of ingredients, once

a pizza is in a locker, the customer will be given 40 minutes to retrieve their pizza before a worker takes it back.

Keep Satisfied • Pizza Customers • Corporations	**Manage Closely** • Store workers • Managers
Monitor (Minimal Effort) • Help Desk • Engineers	**Keep informed** • Marketing

Figure 2. Stake Holders Chart

A user can expect that the locker mechanism will keep their order safe through the use of strong 9V solenoid locks and restrict access to a user's order to the user themselves and employees. The user can also rely on the touch-free barcode scanning mechanism to limit the interaction with high traffic surfaces that could rapidly spread COVID-19. Finally, our user should expect to interact with a GUI on the pizza locker notifying the user of which locker their order is placed in along with audio cues once their barcode is scanned and locker is unlocked.

Some of the constraints that we may have are getting hold of high-quality hardware components such as a temperature sensor that measures the temperature of the locker accurately. Another user-dependent constraint is that our barcode scanner must be able to read barcodes on phones with lower brightness and varying resolutions. This requires careful attention when selecting a barcode scanner to fit within this constraint.

	Engineering Specifications	Cost of material	Efficiency	Tempurature range	Power	Volume of box	Time to install		
Customer Importance:									
Efficiency		1	3	1	1	2	1		1 = Low Relationship
Cost of Pizza		1	1	1	1	1	1		2 = Medium Relationship
Availability		1	3	1	1	2	1		3 = High Relationship
Quality		3	1	3	2	1	1		
Fresh and Hot		2	3	3	1	1	1		
Good texture		2	1	3	1	1	1		
Appetizing appearance		2	1	3	1	1	1		

Figure 3. QFD chart for Customer Importance and Engineering Specifications

3. Technical Specifications & Verification

Requirements	Requirement #	Sub Requirement	Sub Requirement #
System follows power constraints	1	System must consume less than 1.5KW	1.1
		System must consume less than 120VAC	1.2
System software requirements	2	System must generate a barcode/QR code for customer to access locker	2.1
		System must generate a barcode/QR code for employee to access locker	2.2
		Employee must be able to update order status	2.3
		Customer must be able to see their order status & locker number	2.4
		User application can read and display data from the locker	2.5
System hardware requirements	3	Each locker can fit at least 3 pizzas	3.1
		Locker doors unlock when correct QR barcode is scanned	3.2
		Locker doors automatically lock when the door is closed	3.3
		Maintains pizza at a food-safe temperature	3.4
		System must be able to scan QR/barcode	3.5
		Locker doors have appropriate venetilation	3.6
System is efficient	4	Customers are able to pick up their pizza within 5 minutes	4.1
		Employees are able to drop off a pizza in less than a minute	4.2
System is user friendly	5	System has indicators and instructions to aid customer	5.1
		System displays order status (currently in locker, in progress)	5.2
		Pizza locker has a sign/words indicating what it is	5.3

Figure 4. Initial Technical Specification chart

Figure 4 clearly indicates the various technical specifications that were created during the design proposal portion of the semester. The table below, **Figure 5,** displays how various technical specifications changed as we progressed in building our prototype. Since our final product is a prototype, there may be some specifications that are not fully met.

Sub Requirement #	Updated Specification	Measured Specification
1.1	System must consume less than 1.5 KW	Specification achieved
1.2	System must consume less than 120VAC	Specification achieved
2.1	System must generate only a barcode for customer	Generates barcode
2.2	System must generate only a barcode for employee	Generates barcode
2.3	System must have a way to update order status	Specification achieved
2.4	Customer must be able to see their order status & locker number	Specification achieved
2.5	User application can read and display temperature data from locker	Specification not achieved
3.1	Each locker can fit at least 3 pizzas	2 pizzas fit comfortably, 3 fit tightly
3.2	Locker doors unlock when correct QR barcode is scanned	Locker door unlocks if barcode read is valid for an order in locker
3.3	Locker doors automatically lock	Locker door locks 5 seconds after being unlocked
3.4	Maintains pizza at food-safe temperature	Temperature ranges from 150F – 180F
3.5	System must be able to scan barcode	Scanner beeps upon scanning
3.6	Removed	Removed
4.1	Customers must be able to pick up a pizza within 5 minutes	If no wait, pickup time < 45 seconds
4.2	Employees can drop off a pizza in less than a minute	Drop off time < 45 seconds
5.1	System has indicators and instructions to aid customer	GUI directs customer
5.2	System displays order status	User application has order status
5.3	Pizza locker is Identifiable to user	Vinyl stickers to indicate purpsoe

Figure 5. Updated specification chart

4. Design Approach and Details

Our design approach can be broken up into four sub-categories: the physical lockers, micro controller, solenoid lock control and the user application. The physical lockers will be designed with the minimization of heat dissipation in mind to ensure a pizza that is ready for pickup can remain at an appropriate temperature. The last piece of major hardware in this category would be temperature sensors to periodically update the temperature data on the user-application.

In order for us to create a modern device, we will have a need for a microcontroller sub-system that will interface with the back end of our user application. This sub-system will oversee the locker's side of authentication and opening a specified locker based on a barcode, providing data to the user application about pizza temperature, and the physical locking/unlocking of each locker. The microcontroller we chose to go with is the Raspberry Pi 4 8GB version. For the application aspect of this project, we need an application that is account based, which can be used from either a customer standpoint or employee standpoint. An employee needs to be able to update order status and access lockers, while a customer needs to be able to see information regarding order status, pizza temperature, and authentication information through use of barcodes that allows them to scan at the locker and retrieve appropriate order. For this, we decided to use React Native, a JavaScript framework for cross platform mobile development, as the frontend for the applications, and Amazon Web Services for the backend, utilizing API Gateway for the interfacing between the employee, user, and locker, and DynamoDB, a NoSQL database service for holding the user information, employee information, and current order information.

The solenoid control system consists of a one-channel relay module and a solenoid lock, which is a computer controllable lock, for each tray in the locker. The current protype contains three trays,

however, the possibility of developing larger-sized lockers is investigated in later sections of this document.

The physical locker subsystem design approach was to adapt a preexisting, three-tiered pizza warmer into a pizza locker. The benefit to using a preexisting pizza warmer rather than building it from the ground up is that we were able to focus more on the sensor system and less on the heating elements and mechanics (as we are a team of electrical and computer engineers). From the pizza warmer, many adaptations still had to be made.

4.1 Design Concept Ideation, Constraints, Alternatives, and Tradeoffs

This section will look at the functions the design needs to fulfill and our solutions to each. Below is a list of fundamental functions that the design must fulfill according to our current specifications. They will be broken up into our four main sub-categories listed above.

1. Physical Locker
    a. The physical locker must maintain the pizza temperature between 150 – 200 Fahrenheit. This temperature is subject to change based on further testing of ideal pizza temperature within a cardboard box.
    b. Each locker must have some sort of ventilation to ensure appropriate moisture is maintained so that the food does not dry up or become soggy. Further specification on this will be collected via experiments.
    c. Each locker must be able to fit a reasonable order size of 2-5 pizzas or have varying sizes for different sized orders. This size can be obtained by viewing data of the average pizza order as well as the dimension of pizza boxes, breadstick boxes, and dessert containers.

d. There should be no access available to an individual locker unless you are the end consumer or an employee placing the order into the locker.

e. The Physical locker must be able to be powered by a regular 120V 15A outlet.

2. Micro-controller

a. Our Raspberry Pi must be able to execute code in a threaded manner to fulfil the GUI and functional requirements within our system.

i. This is accomplished using Python as our programming language since it allows for easy Object-Oriented programming that is incredibly helpful for containment and partitioning of our code.

b. Our micro controller must be able to interface with some user interface to provide UI and make order pickup as simple as possible.

i. Various libraries were considered for the UI interface such as Kivy, PyQT, and Tkinter [12]

ii. These libraries follow a trend of trade-offs that with increasing complexity in the programmability, you gain many features. In the end, our GUI did not require many advanced features, so PyQT5 was utilized.

iii. In addition, Tkinter has since been deprecated. The development of the code had started with Tkinter but limited documentation forced the shift to PyQT5.

iv. Finally, a huge decision was needed to be made on how to structure the GUI code. Many systems using microcontrollers structure their code in a single executing thread and periodically update the GUI based on events that occur. This structure is very similar to a state-machine and can be very easy to understand and program. However, in this scenario, it could lead to large delays/unresponsive GUI due to the nature of the external inputs (barcode scanner, temp sensor reading, network API calls). The trade-off is similar as

before, if we structure our code in a multi-threaded fashion, we add complexity in exchange for a far more responsive GUI and the ability to have concurrency. Ultimately, the gained performance is too great to overlook.

c. The micro controller must be able to communicate via the web to our user-application back end to send/receive data.

   i. There are various Raspberry Pi models that allow for both wired and wireless internet connection, but current chip shortages play a great role in the stock of both. In this scenario, we didn't have much of an option and had to use a wired version provided to us by the ECE parts shop. However, for our design exposition, we did not have access to ethernet connection so had to purchase a Wi-Fi dongle.

d. The micro controller must be able to interface with hardware such as a temperature sensor, locking mechanism (solenoid), potential infrared lighting system, and barcode reader. The main reasoning behind using a barcode for authentication is to have a unique barcode per transaction. Furthermore, in the times of COVID-19, we can use barcode readers to have contact-less transactions that could not be possible with a PIN code.

   i. The biggest constraint affecting the decision of which external hardware/sensors to use is that we are limited by the functionality and number of GPIO pins on the Raspberry PI. Therefore, we can only use sensors that function with the interfaces provided by the Raspberry PI. Such interfaces include serial communication, PWM pins, digital IO pins, analog IO pins, and an SPI interface. We would've loved to create a prototype with many more lockers than 3, but this would require allotting further pins and utilizing the limited power supplied by the Pi's 5V GPIO pin.

        ii. Another trade-off we made was with determining which barcode scanner to utilize. A potential option was to use a Bluetooth Barcode scanner shown in a demo online [10]. This implementation would require the use of a python library named evdev to handle events in a Linux environment such as the Raspbian OS of our microcontroller. While the wireless scanner could make the physical design of our locker simpler, it would be a bit over-kill on the microcontroller side of things. An alternative is a simpler USB wired scanner that can be treated as a file in python.

3. Solenoid Control
    a. The solenoid control system must maintain the pizza locker doors locked in its off state to prevent overheating of the solenoids, as well as to optimize device energy efficiency.
    b. The corresponding relay should be actuated upon correct barcode being scanned by scanner, whether on employee's or customer's side.
    c. The solenoids should promptly unlock the locker door corresponding to the order scanned when its relay is actuated.

4. User application
    a. The application must be able to read and display data from the actual locker unit to the end user (information from microcontroller), such as temperature, status of order, locker number to retrieve order from.
        i. The user application and employee applications have API calls using Amazon Web Services API Gateway to retrieve information stored in a database about their current order or list of current orders at their location in the case of the employee, and the Raspberry Pi updates order status in the database once the employee puts it in, periodically updates with temperature updates, and removes

the order from the database when the customer picks it up. One tradeoff here was that the employee application was modeled as a mobile application, which in an actual use case, may not be feasible, and therefore would most likely need to be integrated into the existing order system in store.

b. The application must be able to be used from a customer perspective and employee perspective to store/retrieve pizza in lockers.

    i. The application successfully interfaced with the locker to place and pickup orders, but one tradeoff due to time constraints had to do with security concerns. The application did not have a serious authentication system for user or employee accounts, instead tying the input username or employee id to the entry in the database without verifying credentials, but this format was fine for demonstration and proof of concept. In future iterations, Amazon Cognito and Amazon Secrets Manager could be used for authentication and providing login tokens that could be used from an employee or user perspective to verify credentials.

Within each of these design approaches there were various trade-offs to consider which will be discussed below.

1. Selecting to incorporate some sort of ventilation system does complicate the design of the code running on the micro-controller but adds more control to the quality of each consumable. In addition, further power will be consumed that needs to be considered. However, as the development of our project progressed, we deemed that the amount of time a user would have an order in a locker would be rather small and deem implementing a ventilation system as unnecessary. Some of our reasoning behind this is that a user is notified when their order is placed in a locker along with the already existing order estimation time. This means, ideally, a

user's order would not persist in a locker for much longer than 10 minutes. On top of that, we are restricted by our budget and use of a single Raspberry Pi.

2. The design of different size lockers poses an issue of potentially underutilizing space that could be used for a larger number of small orders. In our final iteration of our design, we will have to consider the dynamics of the store. If the pizza locker will be a separate entity, the pizza locker will fully utilize space by remaining the same width, but having more lockers vertically to about 5 feet, which is roughly shoulder height. The current design has the heating element above the top locker, serving as the heating element for all three tiers. If this is adapted to have maybe nine or so tiers, two more heating elements will have to be added. This design would be efficient, because while it scales the number of sensors and heating elements needed, it can still operate with one microcontroller, one barcode scanner, and one touch screen.
3. One very simple trade off to consider is the authentication method to use to access one's order. Utilizing a pin is a very viable option with simplicity but could hinder the user experience. There are simpler options that have adequate security for our design, such as using barcodes to unlock the lockers. Although this will add some complexity to the user application and micro-controller software, it improves the user experience with added ease and less time used inputting a pin. In addition, transferring a barcode from one phone to another is very simple and can have more uniqueness compared to say a 4-digit code. Once again, we refrain from more complex authentication methods due to the relatively low value of pizza.
4. The final big trade-off we faced was the selection of a micro-controller. There are infinite possibilities such as using an ARM mbed. Although there are various libraries housed on their online compiler, this microcontroller would limit our compute power that we wish to have. In addition, there would be limited screen options available to achieve our desired user interface. There are multiple other microcontrollers that have tradeoffs between price, computing power, IO ports, etc. We chose to use a Raspberry Pie since it has adequate computing power, display

functionality, various GPIO ports with a reasonable price increase. Furthermore, another trade off to consider is the programming languages supported by these micro controllers. The ARM mbed supports C style programming which can run very efficiently but lacks some abstractions and high-level libraries that would make programming for our design a lot easier. On the contrary, the Raspberry Pi can run multiple programming languages such as python which has high level libraries with a wider range of displays. Although python programming can be orders of magnitudes slower, it will not make much of a difference in our scenario and is a worthwhile trade-off. [11]

Similarly, there are other factors that can contribute to any design such as economic, cultural, sustainability, and environmental factors. The table below will look at some of the factors that could affect our design.

Factor	Affect
Sustainability	Ensuring that we create a sustainable design is imperative to the ethical standa of engineering. This factor can be echoed in selecting sustainable materials tha can ensure longevity of our design. A great example of this is selecting a meta chassis for our pizza lockers. This can easily result in a more sustainable desig without producing excess weight. Furthermore, creating a design that is modu is very important. This can ease the maintenance of our design so that smaller components can be very easily replaced. We also chose to go with electronic components that are actively manufactured to ensure replacements are easy to obtain.
Economic	Since our design is targeted at the pizza market, we would like the implementation of our design to not have economic effects on the price of the pizzas. Our design should only enhance the user experience and hopefully increase traffic. We would like to eliminate hesitation of ordering due to fear o time spent picking up a pizza or online ordering process. Therefore, we wish t have our budget reflect that and a sustainable design also ensures we have littl to no impact on the economics of pizza making.
Environmental	As explained in the sustainability portion, we chose parts that would result in ease of maintenance and try to minimize the environmental footprint by being able to reuse as much of our design if a single component breaks.

Figure 6. Global impacts of project

4.2 Engineering Analyses and Experiment

This section will cover the topics discussed in the Proposal and how each was resolved or implemented with notes on what was learned during the process. Once implementation details have been established, we will discuss the various tests used to ensure proper functionality. This will be broken down in the same sub-system sections throughout the document.

Locker

1. Locker doors and wiring

   a. The single door needed to be replaced with three doors, one per tier of the locker. Then, the sensors were attached to the doors. Wiring was routed across the door and up the side of the locker to the microcontroller subsystem.

2. Temperature management

   a. Insulation was added between the top of the pizza warmer and below the microcontroller subsystem to avoid overheating the Raspberry Pi.

3. Integrating the other subsystems

   a. The touch screen was adhered to the top of the locker using L brackets and JB Weld, a two-part epoxy adhesive specifically for metal. The Raspberry Pi was attached to the touch screen

4. Aesthetic features

a. This included a wire cover painted silver on the side of the pizza locker to cover the wires, vinyl stickers cut using the Silhouette Cameo saying “Pizza 1, 2, and 3” stuck on the respective locker, red duct tape along the edges of the locker doors that smoothed rough edges and matched the vinyl stickers, an electrical box to hold the solenoid control system, and a box that rested on top of the touch screen and barcode scanner to cover wires from the power source and barcode scanner.

Microcontroller

1. Multi-threaded approach

   a. Perhaps the biggest factor that influenced the overall structure of the code executing on the Raspberry Pi code is whether we create a thread for the GUI itself or run it all as a single process. Upon beginning to code test functions to test things like the temperature sensor, barcode scanner, and solenoid locks, we came to the realization that we lose performance utilizing a single process. For instance, our main function oversees sending temperature information to the user application back-end every 5 minutes. If we were to couple this with the GUI, then sending the data would be on the critical path of updating our GUI and would have negative affects on the user experience. Therefore, the decision was made to create an independent thread to deal with the GUI and display messaged based on user/employee actions.

   b. Directly after the main thread has initialized all hardware being used, it launches a GUI thread that is represented through an OOP approach with function calls to facilitate the GUI behavior as shown below.

      i. UiButtons

1. In charge of setting up start button to start looking for user/employee barcodes

2. Sets up textbox to notify user/employee of instructions

ii. Start_app

1. Starts process of waiting for actions with a call to scan

iii. Scan

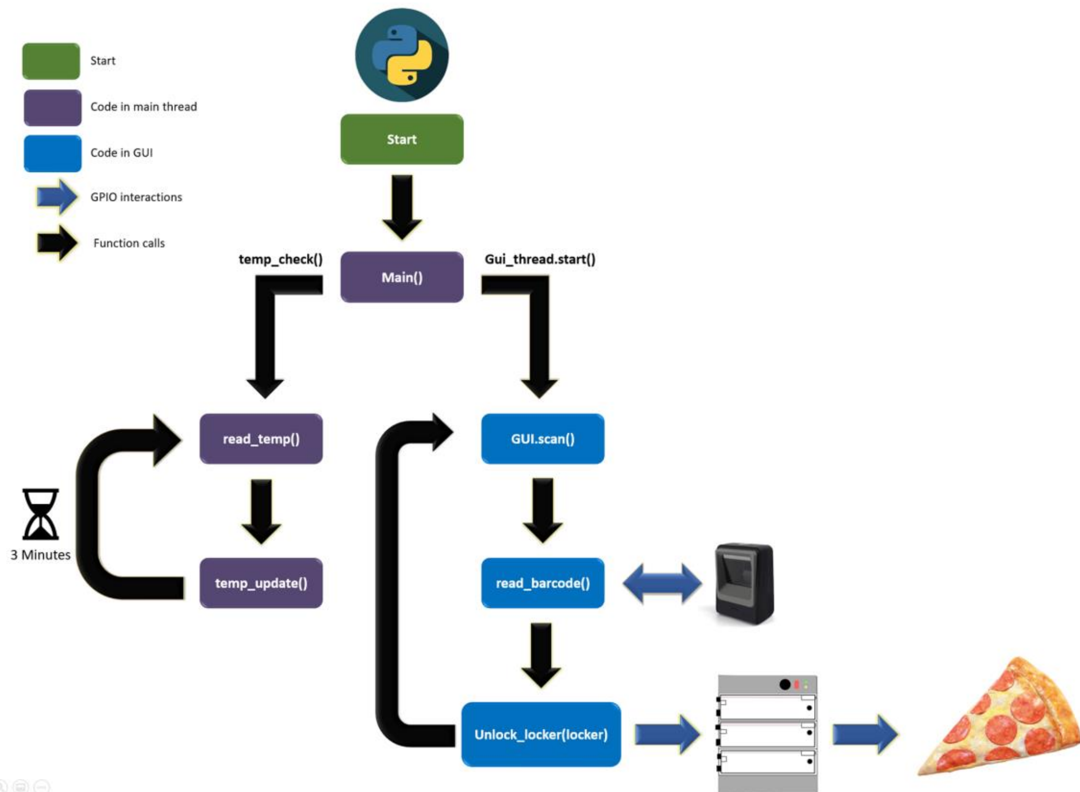
1. Scans barcode, determines if it is a user barcode or employee barcode

    a. Distinction is made by parsing the string that is read by barcode scanner. Employee barcode follows following format: "[storeID]$orderID" whereas the user barcode string will just be the orderID

2. If user barcode read, look up order_ID to locker number mapping and unlocks locker if such a mapping exists. Once unlocked, wait 5 seconds before locking and make API call to remove entry from database

3. If employee barcode read, look for available locker, unlock locker, lock locker, make API call to notify user with order_ID that their order is ready, and update data structures to indicate that locker is no longer free

4. Recursively calls scan operation again to prepare for next action



c.

Figure 7 displays the overall structure of the code and the diverging path of the two threads as well as function calls and interactions with external hardware. Note that during the execution of barcode read operations, the GUI is unresponsive which is acceptable since it just displays a message telling the user to scan a barcode. Once any operation that changes the GUI is performed, we make a call to the processEvents() function provided by PyQT5 to update the GUI.

Figure 7. Hierarchical view of code executing on Raspberry Pi.

d. The testing done for this portion of the code development followed a four-step process discussed below:

i. Initial tests ran code that verified that a secondary thread was executing and that the main thread did not finish execution until the GUI thread terminated. This was tested using the .join() method from the Threading library.

ii. The second phase of testing involved adding the read_barcode functionality to the GUI thread and ensuring that the ascii string the barcode was converted to was correct. This testing phase involved creating arbitrary barcodes using a barcode generator[3]. This was a critical verification role to ensure that a Raspberry PI scanning an OrderID would interpret correct information to use when communicating with the backend of the user app.

iii. Our next phase utilized the starting point of the pervious phase but added calls to unlock_locker(locker) to test that reading a given OrderID associated with a locker would result in correct solenoid behavior and ultimately unlock the locker. Since our system was not fully operational at this point in time, simulating this involved hardcoding an orderID to locker mapping in our data structures.

iv. Finally, we began testing the recursive nature of our scan by iteratively executing the pervious phase using recursion as is needed to process multiple user order pickups. This could've been implemented using an infinite while loop, but it seemed intuitive to use recursion. Furthermore, this testing phase verified that the GUI reset to its default state at the end of each interaction with the user. Thankfully, all testing phases were successful.

2. Barcode Scanner Solution

   a. As stated earlier, from the point of view of the python code, the barcode scanner is nothing more than a file named '/dev/hidraw0' which is opened to be read in binary format. **Appendix A** will have the python code for the entire project present. As seen in the appendix, every iteration within the read_barcode() function loops infinitely until a

non-zero byte is read from the file. This is done utilizing a read() operation of 8 bits and comparing the value to 0 to ensure we only read valid barcodes. Once we have obtained a non-zero value, the value is used to index into two dictionaries that hold decimal value to ASCII char mappings to convert the data to a string. The function continuously adds one character at a time to a buffer until we read a carriage return character indicating the end of the barcode. Finally, the string is printed for debugging purposes.

b. The main issue faced during this portion of code development is varying characters allowed on different barcode generation platforms. For our use-case, we only required alphabetical characters, numerical characters, and the "$" character as a delimiter. This is reflected in our code strictly prohibiting byte values that do not fit within the range of the characters mentioned.

c. The testing of our barcode.py module was done in two parts. The first was to verify that our barcode code would be able to successfully read barcode strings that corresponded to the format for a user and employee. More specifically an employee would have a barcode string "atlanta2$OrderID". Once this was verified, we implemented the function within the scan() GUI operation and were able to successfully perform appropriate functions (locking/unlocking, API calls, etc) based on if the barcode was for a user or employee. This led to the second phase which was focused on testing the functionality of the barcode scanner under various phone brightness levels. As you can see from **Figure 8**, there is a positive correlation of higher brightness with successful barcode reads. Given this, we advise users to use our system with high brightness levels (>80%).

d. Implementing the barcode functionality taught us a great deal about the abstractions that an OS and programming language like python make to interface with rather complex hardware such as a barcode scanner. Furthermore, it gave us real insight into catering your design for your specific use case and not for scenarios that will not be encountered. It was also critical to consider factors outside of the barcode scanner itself such as phone brightness to ensure correct functionality in very real-world scenarios.

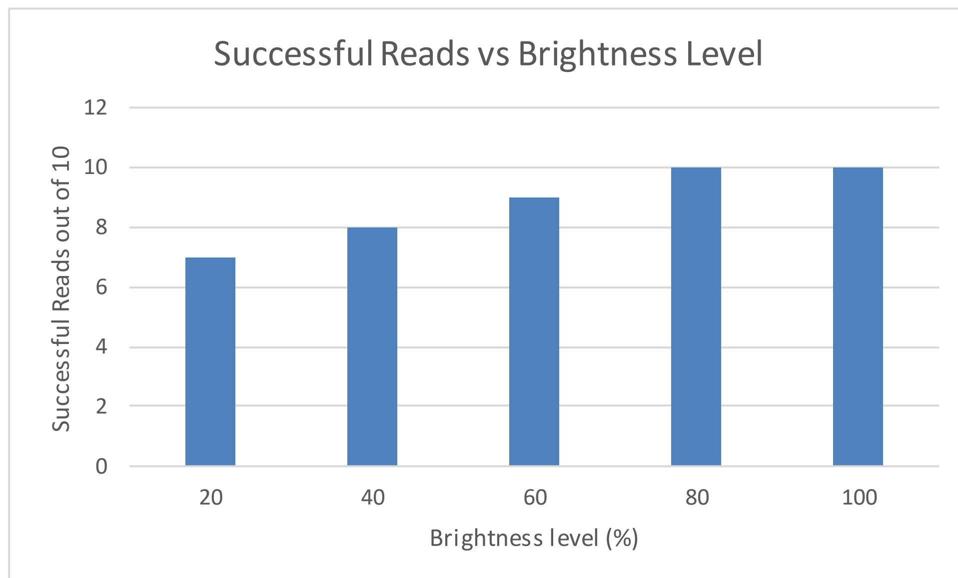


Figure 8. Table showing positive correlation of brightness level with successful reads out of 10.

3. API calls

a. In order for our system to update the user when their order has been placed within a locker and periodically updating them on temperature, we had to implement API function calls to the back-end of the user application. In python, this translates to using a library named requests which is a versatile framework to make API calls and allow for displaying response messages, send data, etc. [5]. Originally, we believed the best way to implement communication from the Raspberry Pi to the user application was using four API calls. The four would oversee notifying the user on their order status, updating temperature data, converting a barcode to an orderID, and deleting an order from the data base. However, some thinking led to the conclusion that encoding the orderID in the barcode would get rid of one API call and therefore reduce budget since each API call costs money. Similarly, when creating the API function call to update temperature data for the user application, a realization was made that we could also pass the locker number for a given order. This means that when the back-end receives this data and sees that a specific order ID does not have a locker associated to it within the database, it can now notify the user that their order has been placed inside a locker. We can guarantee this since our scan() operation sends the initial temperature update only after the pizza has been placed into the locker.

b. As a result, we noticed the need for only two API calls: one to update temperature data while notifying a user of their order and one to delete an order from the database once it has been retrieved. Limiting the number of API calls greatly reduces the code complexity on the back-end front, increases performance by reducing network traffic on the microcontroller, and ultimately saves money.

c. The implementation of API calls emphasized the important lesson of the drawbacks of over-engineering a solution. Although the four API functions make sense intuitively and promote clear semantics, they were not fit for our use-case. This stressed the importance of considering factors beyond technical aspects such as budget. Even though each API call costs a minute amount, our system is intended to be used during incredibly high traffic times and the decrease in API calls can greatly reduce the cost of maintaining our system in the long-term. Finally, there is great benefit to thinking critically about your solution idea before implementation. Fleshing out how we wanted our system to function really highlighted the lack of need for all four API functions and our ability to adapt accordingly lead to a far better implementation of the API portion of the microcontroller code. All API related code can be found in **Appendix B** in apy.py.

d. The testing portion for the API calls happened in two phases starting with just calling each API call with dummy data. In order for this the API to respond accordingly, we had to manually populate entries in the database for random orderIDs and update temperature data for this order as well as remove the order. The next step was to utilize the two API calls within our scan() operation depending on if we have read a user or employee barcode. Testing that the correct information was sent and displayed on the user application was critical to ensuring correctness in our API functions. After testing in both phases, the user was able to receive a notification when their order was placed in a locker, the temperature of their order, and the locker number. Similarly, once the user picks up the order our delete API call successfully deleted entries in the data base to reduce storage overhead.

e. In addition to API calls interfacing with the Raspberry PI, there were 3 API calls designed for interfacing with the user and employee applications. One of these calls was from the customer standpoint, to place an order, where the inputs were the username, the order info, and store location, which was then submitted to the database through the api call. After the order was placed, there was a second call to fetch information for this order periodically to check for updates, where once the order was complete, the application could display a barcode, locker number, ready status, and temperature to the user, based on information submitted by the Pi to the database. The final API call comes from the employee perspective, periodically scanning the database for new orders matching the employee location and updating the employee application with the list of current orders, where when tapped on, the employee could view the order as well as the barcode to use for placing the order in the locker.

Solenoid Control

Regarding the electrical circuitry of our solenoid control system, perhaps the biggest decision to be made was the component selection. More specifically, the components to be used to interact with the Raspberry Pi and control the solenoids. Relays deliver fairly simple and low-power electric switches that act in a fast way to convert small electrical signals into larger currents, therefore presenting a reliable solution to our requirements.

User application

The User application first takes you to a screen where you are asked to create a username. Once you create a username, you are shown an order screen where you can select what you would like to order, as well as be asked to choose a location. Once you verify your order, you are given a barcode that indicates that your order is ready.

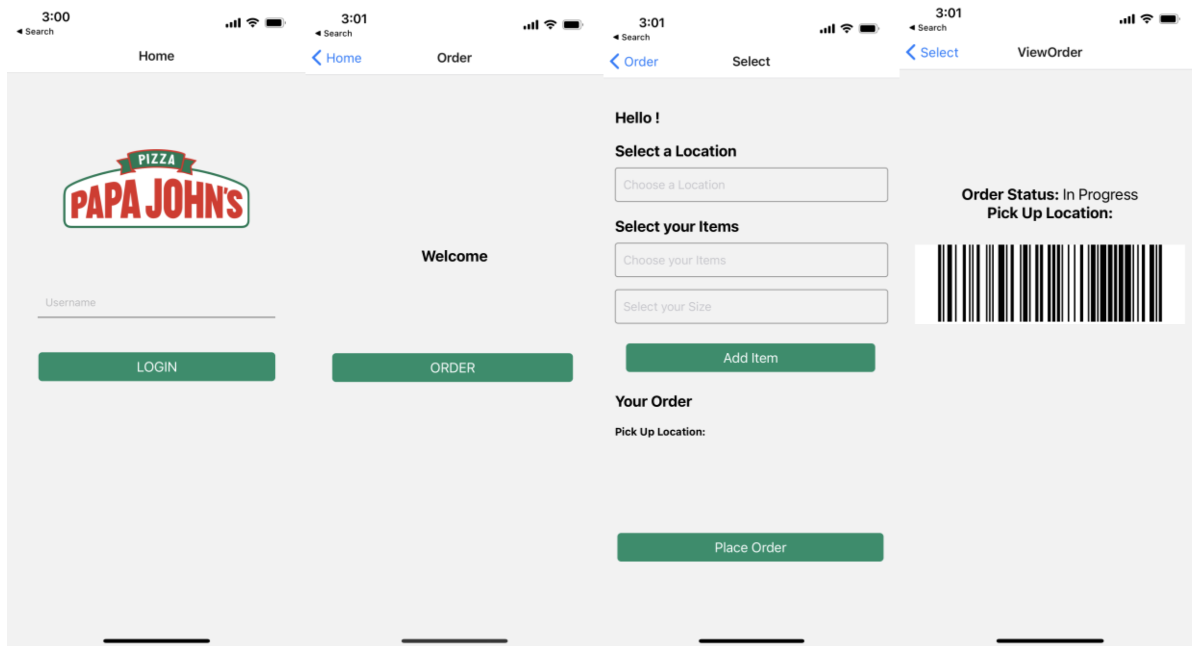


Figure 9. Flow of user application from login to view order status

Employee Application

The employee application first expects an employee id to be input for login purposes, which is tied to a store location through the employee database, where it then transitions to showing a list of active orders at the employee's location, and when the employee taps on an order, being able to view the order information and barcode used to place the order in the locker.

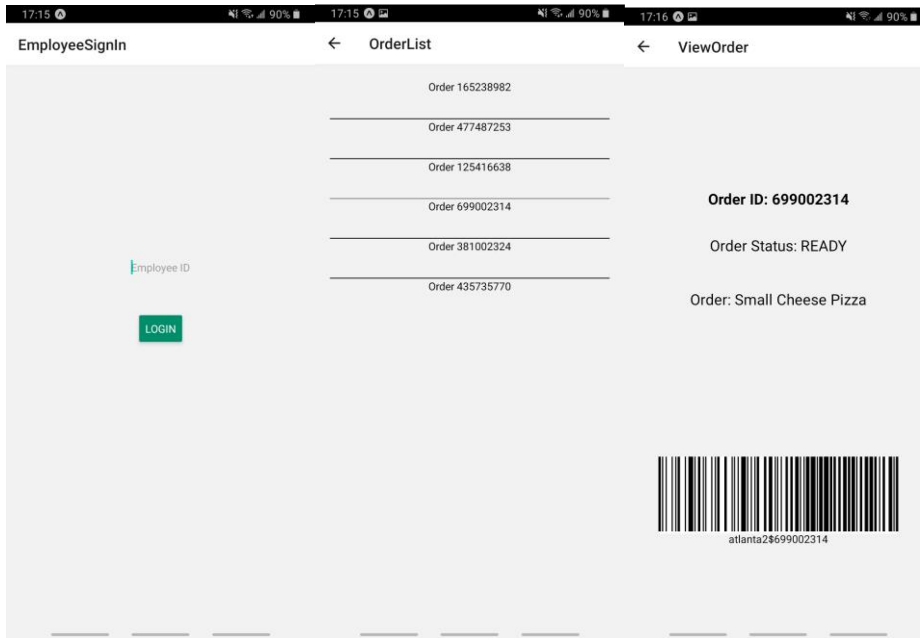


Figure 10. Flow of employee application from login to view an order status in list

4.3 Codes and Standards

Standard Body	Description	Impact
NFPA	Standards relating to heat producing appliances [8]	Heating apparatus/ power supply needs to follow the fire safety guidelines.
IEEE	Standards relating to electronics usage [7] and network traffic usage [6]	Need to follow rules regarding network traffic for application design and use of sensors in our product.
OSHA	Standards relating to workplace safety/health [1]	Need to make sure our locker design does not break any workplace safety requirements based on footprint or heating safety.
FDA	Standards relating to proper storage and handling of food	Need to make sure our locker is up to code relating to safety of food storage.

5. Project Demonstration

To demonstrate that our project was working we presented our project to Dr. Ma, our Advisor, on April 20th. During the demonstration we showed that the pizza locker was able to open using a barcode that we generated with a specific order ID and lock on both the user and employee side. We ordered all the different sized pizzas that were available, as well as appetizer and dessert to make sure that all the order sizes were able to fit into the locker. The locker was able to successfully warm the locker at a specific temperature and keep all the ordered items warm.

List of items that were tested:

- Barcode scanner was able to take the scanned barcode and process it to open the solenoid lock.
- Made sure that the raspberry pi code was working with the hardware components.
- Made sure that the solenoid locks were in place
- Made sure that the temperature sensor was in place.

Some of the specification parameters that we were unable to meet were getting the temperature sensor to display an accurate reading of the temperature of the locker and display it. However, we believe that the locker will enable customers to pick up their pizza in less than 45 seconds.

To test that the solenoid locks were working, we made sure that we were able to turn them on using the raspberry, Pi. Then we made sure that the barcode scanner was able to take in the specific barcode to unlock the solenoid lock. For the UI component we made sure that the user data was being sent through to the API call, which includes username, order ID, order list and order location.

Video of Project Demonstration:

https://drive.google.com/file/d/1Vm-YJi-cu71RZeS5AlVfTPmoqzvXmB3m/view?usp=sharing

6. Schedule, Tasks, and Milestones:

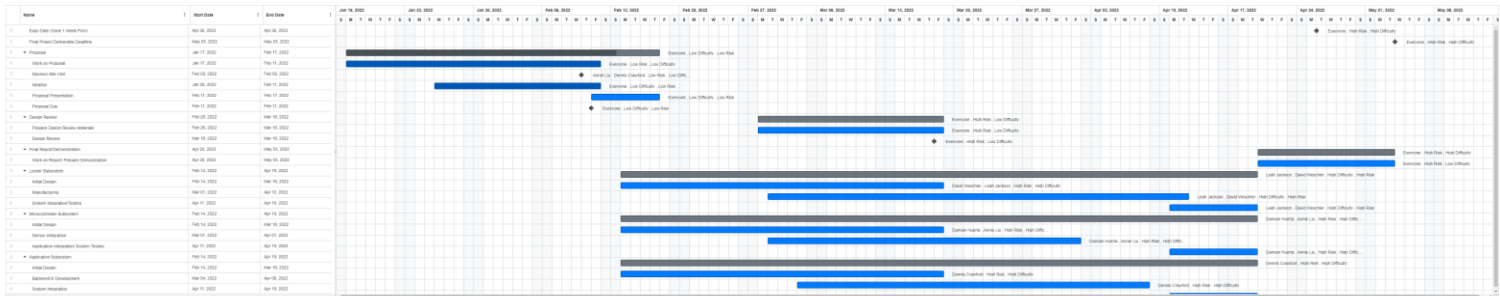


Figure 11. Final Gantt Chart. Higher resolution pdf inside of Final Documents folder and on website under Gantt Chart link

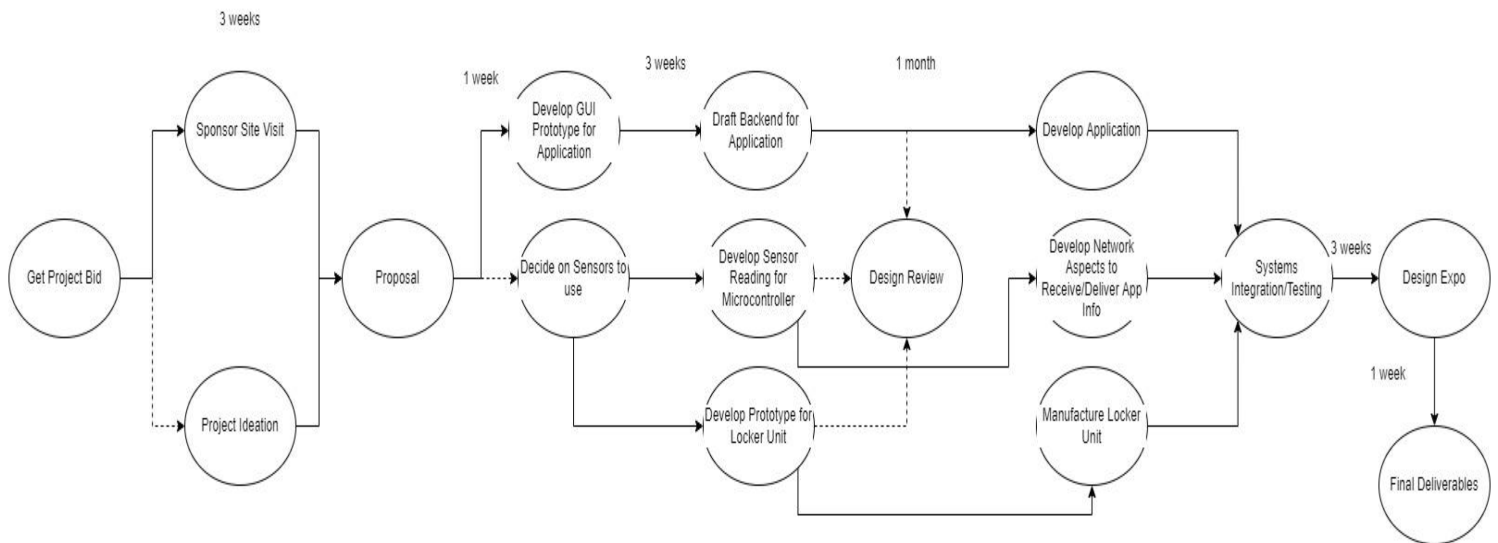


Figure 12. Final Pert Chart. Higher resolution image inside of Final Documents folder

7. Final Project Demonstration

For our final project demonstration, we were able to demonstrate our finished product at the expo. Attendees were able to volunteer to test the product and in return get a free slice of pizza. At the expo, we asked the attendees to make an order on our application and once their order was ready, they were given a barcode on the application that allowed them to scan and receive their order.

Figure 13. Expo attendees listen to explanations of locker's use and functionality

Figure 14. One of our team members demonstrates the use of locker while scanning an order's barcode

8. Marketing and Cost Analysis

8.1 Marketing Analysis

Existing products

1. Order pickup on a shelf
    a. Current competitors, like Chipotle, utilize in-store order pickup by fulfilling orders, attaching a sticker receipt to the order, and placing the order on a shelf. Three problems arise from this. The first is the security of the food. Anyone can grab the wrong order, or even worse, start stealing orders, creating a loss for the company. The second is potential contamination. While a seal is placed on the bag of the order, there's not much barrier to prevent the order from being opened, contaminated, and then reclosed. Third, there is no temperature management on the order. From the time the order is made to when the order is picked-up, it is constantly cooling.
2. Little Ceasars pizza locker
    a. One similar product is Little Ceasar's pizza locker [12]. While our product is similar conceptually, we are making it for our sponsor, their competitor, Papa John's. One fundamental distinction between our product and Little Ceasar's is Little Ceasar's focus is on always available, basic pizzas, whereas Papa John's focuses on order specific pizzas. This means that Little Ceasar's design has to be able to focus on keeping pizzas in there longer, whereas we have to focus more on employee interaction with the pizza locker.

Key marketable features of our product

1. Heated lockers
    a. Our pizza locker has a heating element and a temperature sensor that alerts employees when to adjust the temperature of the oven. While pickup time is expected to be minimal, it allows the pizzas to be at an optimal temperature upon when the customer picks it up.
2. Security
    a. Individual pizza lockers with computer-controllable locks prevent other people from grabbing the wrong order, stealing orders, and contaminating food. This will reduce customer frustration of their order being missing and instill a peace of mind that their order is noncontaminated.

3. Easy employee interface
    a. Since Papa John's focus is on order-specific pizzas, our system for the employees is just as easy to use as the customer interface. Once implemented in the real world, our app will integrate with the store's current order management system.

8.2 Cost Analysis (Budget)

Supplies:

Item	Cost
Pizza Warmer (1)	$500 [1]
Barcode Scanner (1)	$50 [2]
5" Monitor (1)	$70 [3]
Raspberry Pi (1)	$45 [4]
Computer controllable latches (3)	$100
Screws, metal, and accessories to make the doors	$100
Paint	$30
Sticker vinyl	$15
Total	**$910**

Design Labor Costs

Assuming that the average starting salary for an engineer is $75,000, this results in an hourly salary of $38/hr. The average course at Georgia Tech that is three credit hours requires nine hours of weekly time, and six of those hours are spent outside of class, which we will consider the billable hours of the project. There are 13 weeks, out of the 16 total in a semester, that this team of five engineers will each spend six hours on the project. This results in a total labor cost of $14,820.

$$38\text{hr} \cdot \text{engineer} \cdot 6\ \text{hrweek} \cdot 13\ \text{weekssemester} \cdot 5\ \text{engineers} = 14{,}820$$

Total Cost

"n" indicates that this is a cost per pizza locker

Item	Cost
Supplies	$910n
Design Labor	$14820
Assembly	$160(n-1)
Delivery	$200n

The recurring cost per pizza locker is the cost to assemble, supplies, and delivery fees. Assuming that after the initial design plans are made it takes a moderately skilled worker, paid $20/hr., one workday to assemble a pizza locker, this costs $160 for assembly. In the table above, it is assumed that the first pizza locker is assembled by the engineering design team. Delivery is hard to approximate, so for the sake of this discussion it will be approximated to $200, the cost of a day of work from someone at $20/hr, plus gas.

Overall Cost Analysis

Price: $2,000 per Pizza Locker					
#/Yr	Total Cost	Gross Profit	Net Profit	% Profit	Profit/# over 5 yrs
1	$15,930	$2,000	-$13,930	-696.50%	-$69,650
10	$27,360	$20,000	-$7,360	-36.80%	-$36,800
15	$33,710	$30,000	-$3,710	-12.37%	-$18,550
30	$52,760	$60,000	$7,240	12.07%	$36,200

Price: $5,000 per Pizza Locker					
#/Yr	Total Cost	Gross Profit	Net Profit	% Profit	Profit/# over 5 yrs
1	$15,930	$5,000	-$10,930	-218.60%	-$54,650
10	$27,360	$50,000	$22,640	45.28%	$113,200
15	$33,710	$75,000	$41,290	55.05%	$206,450
30	$52,760	$150,000	$97,240	64.83%	$486,200

Price: $10,000 per Pizza Locker					
#/Yr	Total Cost	Gross Profit	Net Profit	% Profit	Profit/# over 5 yrs
1	$15,930	$10,000	-$5,930	-59.30%	-$29,650
10	$27,360	$100,000	$72,640	72.64%	$363,200
15	$33,710	$150,000	$116,290	77.53%	$581,450
30	$52,760	$300,000	$247,240	82.41%	$1,236,200

The above table analyzes three price points for our product when selling different numbers of products. The first price point, $2,000, is unreasonable because it would take selling at least 30 products a year to make a profit. According to this analysis, it would be recommended to sell the pizza locker for between $5-10,000 if 10 units can be sold per year.

9. Conclusion & Current Status

The current status of this project is that the physical locker prototype is functional with the solenoid locks functioning, connected to the microcontroller system, and communicating with the app. The software functions excellently and is ready to be implemented with the real system. However, the physical locker is a protype, and further research and discussion with Papa John's will have to be conducted to determine what setup is optimal with their limited floor space. Overall, the complete system proved that this concept has the potential to be successful, as demonstrated at the design expo, where 50 transactions were completed.

There is not much we would do differently. We are happy with the physical design, given that we are electrical and computer engineers and mechanical systems are outside the scope of our specialty. On the software side, it functions as intended. The only thing we might consider adding is the option to type in a numerical code, as opposed to only scanning a barcode, in case of system failure, screen glare, or other difficulties.

The lessons learned from working on the locker subsystem were taking calculated risks, overcoming physical limitations, creative problem solving, and learning by doing. An individual team member would take on the physical system not due to being qualified to do it, but due to the possibility of learning how to do it. We knew we had the foundation of knowledge and access to resources that would make this possible. She mentions that "Being responsible for the physical system was a risk, but by calculating what the outcomes could be, I was able to decide it was one worth taking". From this, we've learned how important it is to push ourselves to take on new things and make calculated risks. In regard to physical limitations, it's possible to have a great plan but then to be thwarted by the physical limitations of the system and have to adapt it. One example of this was when one of our team members used a jig saw to widen a hole made by a drill for the lock of the solenoids. The jig saw was able to fit for the top two holes and was successful in this. However, for the bottom one, the handle of the jigsaw wouldn't fit because of the bottom of the pizza oven. To overcome this, she put the blade in backwards, which allowed her to hold the jigsaw upside down, even though this was not the intended operation of the jigsaw.

Like the above point, we had to further our problem-solving abilities. One example of this is wire routing. Another team member had to route nine wires from the sensors, across the oven, and up the side of the pizza locker to where the Raspberry Pi connection would be. We wanted to do the wire routing safely, but also not be an eyesore. The solution to this entailed using a small wire cover, originally white but painted silver to further blend in, on the side to hide the wires, and then behind it drilling a hole that connected to the layer between the oven and the top of it, and then the wires were able to come out on top. In learning by doing, we've had to learn several new skills to complete this project, such as using a bandsaw, jigsaw, drilling metal, and using a vinyl cutter. For the vinyl cutter, we didn't have anyone to teach us how to use it, and all the resources online were bare boned. They told you what to do but left out a lot of the practical tips that make it successful. Nevertheless, we

figured out how to do it, and learned that sometimes, we have to learn by doing rather than knowing how to do it going into it.

Future work can continue research into sustainability and contemporary issues. This would include how long the product would last based on the life of the pizza warmer, the solenoid locks, and the temperature sensors. Contemporary issues include how the pizza locker affects the flow of the restaurant and how much space it takes up. All in all, this is a functional prototype with functional software, but further research and development is required for implementation.

10. Leadership Roles

1. Webmaster: Dennis Crawford
2. Expo Coordinator: David Wechsler
3. Sponsor Liaison: Annie Liu
4. Financial Manager: Damian Huerta
5. Team Lead: Leah Jackson

11. References

[1] “Department of Labor Logo United States department of Labor,” *Regulations (Standards - 29 CFR) | Occupational Safety and Health Administration*. [Online]. Available: https://www.osha.gov/laws-regs/regulations/standardnumber. [Accessed: 03-May-2022].

[2] “ECFR :: 21 CFR Part 110 -- current good manufacturing practice in ...” [Online]. Available: https://www.ecfr.gov/current/title-21/chapter-I/subchapter-B/part-110. [Accessed: 03-May-2022].

[3] “Free online barcode generator,” *BarcodesInc*. [Online]. Available: https://www.barcodesinc.com/generator/index.php. [Accessed: 03-May-2022].

[4] “How to control a solenoid with a raspberry pi using a Relay.” [Online]. Available: https://www.youtube.com/watch?v=BVMeVGET_Ak. [Accessed: 03-May-2022].

[5]“HTTP for humans™¶,” *Requests*. [Online]. Available: https://docs.python-requests.org/en/latest/. [Accessed: 03-May-2022].

[6] “IEEE Communications Standards,” *SA Main Site*, 31-Jan-2022. [Online]. Available: https://standards.ieee.org/search/?q=Communications&type=Standard. [Accessed: 03-May-2022].

[7] “IEEE Electrical Safety Standards,” *SA Main Site*, 31-Jan-2022. [Online]. Available: https://standards.ieee.org/search/?q=National+Electrical+Safety+Code+NESC&type=Standard. [Accessed: 03-May-2022].

[8] “List of NFPA Codes & Standards,” *List of NFPA Codes and Standards*. [Online]. Available: https://www.nfpa.org/Codes-and-Standards/All-Codes-and-Standards/List-of-Codes-and-Standards. [Accessed: 03-May-2022].

[9] “Little Caesars just launched a pizza locker that lets you avoid human interaction,” *Little Caesars Just Launched A Pizza Locker That Lets You Avoid Human Interaction*. [Online]. Available: https://www.foodbeast.com/news/little-caesars-pizza-portal/. [Accessed: 03-May-2022].

[10] Piddler, “Wireless Barcode Scanner,” *piddlerintheroot*, 11-Dec-2020. [Online]. Available: https://www.piddlerintheroot.com/wireless-barcode-scanner/. [Accessed: 03-May-2022].

[11] Raspberry Pi, “Buy A raspberry pi 4 model B,” *Raspberry Pi*. [Online]. Available: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/. [Accessed: 03-May-2022].

[12] ResellerClub, “The 6 best python GUI frameworks for developers,” *Medium*, 18-Oct-2019. [Online]. Available: https://medium.com/teamresellerclub/the-6-best-python-gui-frameworks-for-developers-7a3f1a41ac73. [Accessed: 03-May-2022].

Appendices

Appendix A – Raspberry Pi Python Code

Main.py

```
from hardware import *
from locker_threads import*


#set of all lockers
lockers = {1,2,3}




def locker_temp_check():
    print("checking locker temp")
    temp = checkTemp(1)
    for locker in locker_to_order_number:
        print(locker)
        order_temp_update(locker_to_order_number[locker], temp[0], locker)

    return




def main():
```

```
    #initialize hardware
    initialize_hardware(free_lockers)

    #launching GUI thread
    print("Launching GUI Thread")
    gui_thread = myThread(1, "GUI-Thread")
    gui_thread.start()


    #start periodically sending temp information to lockers that are open
    while 1:
        locker_temp_check() #might just use the one sensor since all lockers should be same temp
        print("sent temp info to back end")
        time.sleep(600) #sending temp every 5 minutes

    #waiting for gui thread to finish
    gui_thread.join()






if __name__ == "__main__":
    main()
```

Locker_threads.py

```
#regular imports
import time
import threading
from gui import *
from api import *




class myThread (threading.Thread):
   def __init__(self, threadID, name):
      threading.Thread.__init__(self)
      self.threadID = threadID
      self.name = name
   def run(self):
      print ("Starting " + self.name)
      gui_thread()
      print ("Exiting " + self.name)
```

```
def gui_thread():
    print("Inside GUI thread")
    setup_initial_gui()
    return
```

Gui.py

```
#imports
from PyQt5.QtWidgets import QApplication, QMainWindow, QWidget, QPushButton, QLabel
import sys
import time
from api import order_temp_update,  delete_order_number
from bacode import *
from hardware import *



class Window(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setGeometry(0, -10, 1950, 1100)
        self.setWindowTitle("Pizza Locker GUI")
        self.UiButtons()
        self.show()

    #method to setup initial buttons
    def UiButtons(self):
        self.start_btn = QPushButton("Start", self)
        self.start_btn.move(250,150)
        self.start_btn.pressed.connect(self.start_app)
        self.text_label = QLabel(self)
        self.text_label.setGeometry(220,0,300,50)


    #start application method
    def start_app(self):
        self.start_btn.deleteLater()
        self.show()
        self.scan()

    def scan(self):
        print("start scanning")
        barcode = read_barcode()
        split_barcode = barcode.split("$")

        if split_barcode[0] == store_barcode:
            #barcode read was store barcode so now read for user barcode
            print("Dealing with employee  barcode")
            user_barcode = split_barcode[1]
            locker = get_free_locker()
```

```
        if locker == -1:
            print("No locker available")
        else:
            #unlock locker
            print("going to unlock {}".format(locker))
            self.text_label.setText("Employee, place order in locker {}".format(locker))
            QApplication.processEvents()
            unlock_locker(locker)
            time.sleep(5)
            lock_locker(locker)

            #send API call now
            (temp, humidty) = checkTemp(1)
            order_number = user_barcode #might need to change this to just part of what was read
            order_temp_update(order_number, temp, locker)
            self.text_label.setText("User has been notified that their order is ready for pickup")
            QApplication.processEvents()
            #update dictionaries
            locker_to_order_number[locker] = order_number

    else :
        #read user barcode
        order_number = barcode
        print("dealing with user barcode")
        locker_number = get_locker_number(order_number)
        if (locker_number == -1):
            print("order not found")
            self.text_label.clear()
            QApplication.processEvents()
            self.scan()
        else:
            print("found order")

        #notify user where their order is placed and unlock
        self.text_label.setText("User, locker {} has been unlocked. Please close after retreiving order".format(locker_number))
        QApplication.processEvents()
        unlock_locker(locker_number)
        time.sleep(5)
        lock_locker(locker_number)
        delete_order_number(barcode)

        #update dictionaries
        locker_to_order_number.pop(locker_number)

    time.sleep(5)
    self.text_label.clear()
    QApplication.processEvents()
    self.scan()

def setup_initial_gui():
```

```
    app = QApplication(sys.argv)
    window = Window()
    app.exec_()
```

Hardware.py

```
#imports
import time
import board
import adafruit_dht
import RPi.GPIO as GPIO

#dictionaries
locker_to_gpio = {1:16, 2:20, 3:21}
locker_status = {} #1 is available 0 is busy
free_lockers = {}
locker_to_order_number = {}


#initialize harware. Might not actually need this
def initialize_hardware(free_lockers):
    GPIO.setwarnings(False)
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(16, GPIO.OUT)
    GPIO.setup(20, GPIO.OUT)
    GPIO.setup(21, GPIO.OUT)
    locker_status[1] = 1
    locker_status[2] = 1
    locker_status[3] = 1
    return

def unlock_locker(locker):
    GPIO.output(locker_to_gpio[locker], 0)

def lock_locker(locker):
    GPIO.output(locker_to_gpio[locker], 1)

def get_free_locker():
    for locker in locker_status:
        if locker_status[locker] == 1:
            locker_status[locker] = 0
            return locker
    return -1



def get_locker_number(order_number):
    for locker in locker_to_order_number:
        if locker_to_order_number[locker] == order_number:
            return locker
```

```
    return -1

def checkTemp(locker_number):
    dhtDevice = adafruit_dht.DHT22(board.D18)
    while True:
        try:
            temperature_c = dhtDevice.temperature
            temperature_f = temperature_c * (9/5) + 32
            humidity = dhtDevice.humidity
            print("Temp: {:.1f} F / {:.1f} C Humidity: {}% ".format(temperature_f, temperature_c, humidity))
            tup = (temperature_f, humidity)
            return tup
        except RuntimeError as error:
            # Errors happen fairly often, DHT's are hard to read, just keep going
            time.sleep(2.0)
            continue
        except Exception as error:
            dhtDevice.exit()
            raise error
```

<u>Barcode.py</u>

```
#File in charge of reading barcode scanner and returing string might use
import sys
import json

store_barcode = "atlanta2"

def read_barcode():
    hid = {4: 'a', 5: 'b', 6: 'c', 7: 'd', 8: 'e', 9:
           'f', 10: 'g', 11: 'h', 12: 'i', 13: 'j', 14: 'k', 15: 'l', 16: 'm',
           17: 'n', 18: 'o', 19: 'p', 20: 'q', 21: 'r', 22: 's', 23:
           't', 24: 'u', 25: 'v', 26: 'w', 27: 'x', 28: 'y', 29: 'z',
           30: '1', 31: '2', 32: '3', 33: '4', 34: '5', 35: '6', 36:
           '7', 37: '8', 38: '9', 39: '0', 44: ' ', 45: '-', 46: '=',
           47: '[', 48: ']', 49: '\\', 51: ';', 52: '\'', 53: '~', 54:
           ',', 55: '.', 56: '/'}

    hid2 = {4: 'A', 5: 'B', 6: 'C', 7: 'D', 8: 'E', 9: 'F', 10: 'G', 11:
            'H', 12: 'I', 13: 'J', 14: 'K', 15: 'L', 16: 'M', 17: 'N',
            18: 'O', 19: 'P', 20: 'Q', 21: 'R', 22: 'S', 23: 'T', 24:
            'U', 25: 'V', 26: 'W', 27: 'X', 28: 'Y', 29: 'Z', 30: '!',
            31: '@', 32: '#', 33: '$', 34: '%', 35: '^', 36: '&', 37:
            '*', 38: '(', 39: ')', 44: ' ', 45: '_', 46: '+', 47: '{',
            48: '}', 49: '|', 51: ':', 52: '"', 53: '~', 54: '<', 55:
            '>', 56: '?'}
```

```
fp = open('/dev/hidraw0', 'rb')
#print("opened")
#print(fp)
ss = ""
shift = False

done = False

while not done:
    #print("looping")
    ## Get the character from the HID
    buffer = fp.read(8)
    #print(buffer)
    for c in buffer:
        #print(c)
        if ord(chr(c)) > 0 and ord(chr(c)) <= 56:
            ##  40 is carriage return which signifies we are done
            ##  looking for characters
            if int(ord(chr(c))) == 40:
                done = True
                break

            ##  If we are shifted then we have to use the hid2
            ##  characters.
            if shift:

                ## If it is a '2' then it is the shift key
                if int(ord(chr(c))) == 2:
                    shift = True

                ## if not a 2 then lookup the mapping
                else:
                    ss += hid2[int(ord(chr(c)))]
                    shift = False

            ##  If we are not shifted then use the hid characters

            else:

                ## If it is a '2' then it is the shift key
                if int(ord(chr(c))) == 2:
                    shift = True

                ## if not a 2 then lookup the mapping
                else:
                    ss += hid[int(ord(chr(c)))]
print("READ BARCODE: {}".format(ss))
return ss
```

Api.py

```
#file in charge of all API requests
import json
from wsgiref.util import request_uri
import requests

#Functions
api_url = "https://xsb8acpx2a.execute-api.us-east-1.amazonaws.com/dev/barcode"
base_url = "https://xsb8acpx2a.execute-api.us-east-1.amazonaws.com/prod/locker/"


#tells API that order_number has new temp. Will tell user updated temp
def order_temp_update(order_number,  temp, locker_number):
    print(order_number)
    print(temp)
    print(locker_number)
    url = base_url + order_number
    print(url)

    r = requests.put(url, json = {"temperature": str(temp), "locker" : locker_number})
    print(r.status_code)
    print(r.content)
    return

#tells API to remove entry for order_number since it has been picked up
def delete_order_number(order_number):
    print("deleting order")
    url = base_url + order_number
    print
    r = requests.delete(url)
    print(r.status_code)
    print(r.content)
    return
```

Appendix B – API Calls Handler Python Code

pizzalocker_placeorder

lambda_function × Execution results ×

```python
import json
import boto3
import random

REGION="us-east-1"
dynamodb = boto3.resource('dynamodb', region_name=REGION)
orderTable = dynamodb.Table('PizzaLocker_OrderItems')
userTable = dynamodb.Table('PizzaLocker_Customer')

def lambda_handler(event, context):
    username = event['body-json']['username']
    storeID = event['body-json']['storeid']
    order = event['body-json']['order']
    orderID = str(abs(hash(username+storeID+str(random.randrange(1,100)))))[0:9];
    employeeBarcode = storeID + "$" + orderID
    response = userTable.get_item(Key={'username':username})
    if 'Item' in response :
        if response['Item']['orderID'] is not '':
            return {
                'statusCode': 403,
                'body': "There is already an active order"
            }
    orderTable.put_item(
    Item={
            "OrderID":orderID,
            "StoreID":storeID,
            "Order":order,
            "OrderStatus":"PLACED",
            "Temperature":None,
            "LockerNumber":None,
            "EmployeeBarcode":employeeBarcode
        }
    )
    response = userTable.update_item(
        Key={
            'username': username
        },
        UpdateExpression="set orderID=:o",
        ExpressionAttributeValues={
            ':o':orderID
        },
        ReturnValues="UPDATED_NEW"
    )

    return {
        'statusCode': 200,
        'body': orderID
    }

```

pizzalocker_customer_getorder

lambda_function

```python
import json
import boto3
from boto3.dynamodb.conditions import Key, Attr

REGION="us-east-1"
dynamodb = boto3.resource('dynamodb', region_name=REGION)
orderTable = dynamodb.Table('PizzaLocker_OrderItems')
userTable = dynamodb.Table('PizzaLocker_Customer')

def lambda_handler(event, context):
    username = event['params']['path']['username']

    response = userTable.get_item(Key={'username':username})
    if 'Item' not in response:
        return {
            'statusCode': 403,
            'body': 'This user does not have an active order!'
        }
    orderID = response['Item']['orderID']

    orderResponse = orderTable.scan(
        FilterExpression=Attr('OrderID').eq(orderID)
    )
    if 'Items' not in orderResponse:
        return {
            'statusCode': 403,
            'body': 'There is no active order matching this id!'
        }
    item = orderResponse['Items'][0]
    item = {
        'OrderID': item['OrderID'],
        'Order': item['Order'],
        'LockerNumber': item['LockerNumber'],
        'Temperature': item['Temperature'],
        'OrderStatus': item['OrderStatus'],
        'StoreID': item['StoreID']
    }
    return {
        'statusCode': 200,
        'body': item
    }

```

pizzalocker_employee_getorders

```python
import json
import boto3
from boto3.dynamodb.conditions import Key, Attr

REGION="us-east-1"
dynamodb = boto3.resource('dynamodb', region_name=REGION)
orderTable = dynamodb.Table('PizzaLocker_OrderItems')
employeeTable = dynamodb.Table('PizzaLocker_Employees')

def lambda_handler(event, context):
    employeeId = event['params']['path']['id']

    response = employeeTable.scan(
        FilterExpression=Attr('EmployeeID').eq(employeeId)
    )
    storeID = response['Items'][0]['StoreID']

    orderResponse = orderTable.scan(
        FilterExpression=Attr('StoreID').eq(storeID)
    )
    items = orderResponse['Items']
    return {
        'statusCode': 200,
        'body': items
    }

```

pizzalocker_updateorder

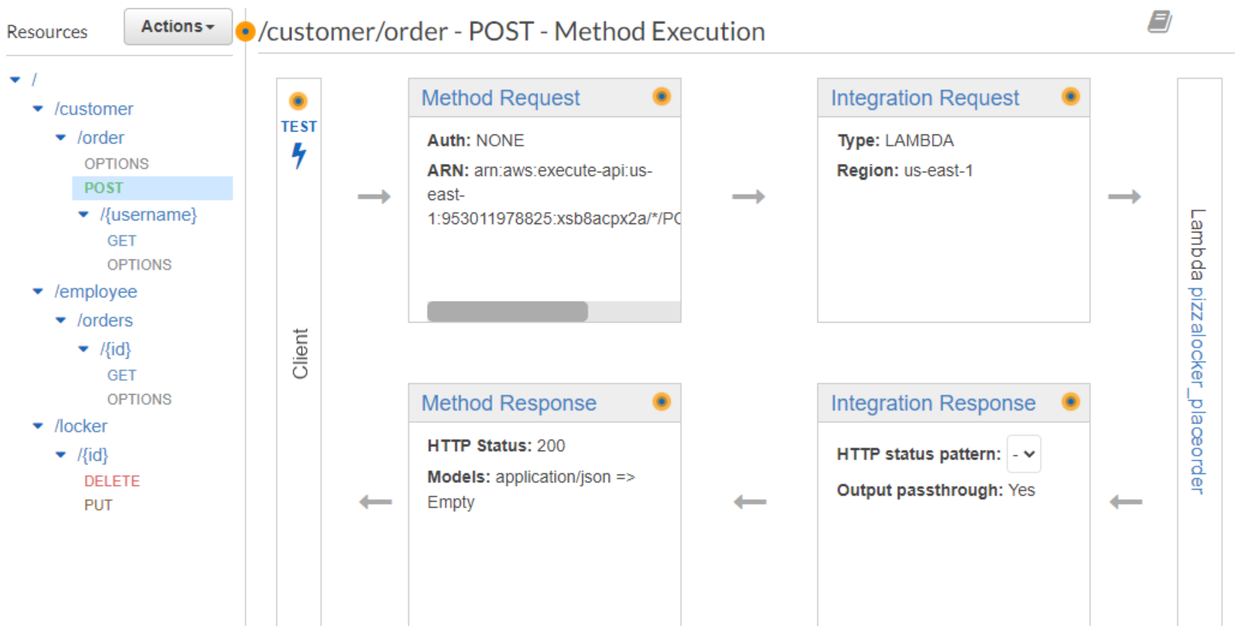
```python
import json
import boto3
from boto3.dynamodb.conditions import Key, Attr

REGION="us-east-1"
dynamodb = boto3.resource('dynamodb', region_name=REGION)
orderTable = dynamodb.Table('PizzaLocker_OrderItems')

def lambda_handler(event, context):
    locker = event['body-json']['locker']
    temperature = event['body-json']['temperature']
    orderID = event['params']['path']['id']
    orderResponse = orderTable.scan(
        FilterExpression=Attr('OrderID').eq(orderID)
    )
    storeID = orderResponse['Items'][0]['StoreID']
    orderUpdateResponse = orderTable.update_item(
        Key={
            'OrderID':orderID,
            'StoreID':storeID
        },
        UpdateExpression="set Temperature=:t, LockerNumber=:l, OrderStatus=:o",
        ExpressionAttributeValues={
            ':t': temperature,
            ':l': locker,
            ':o': "READY"
        },
        ReturnValues = "ALL_NEW"
    )

    return {
        'statusCode': 200,
        'body': json.dumps(str(orderUpdateResponse))
    }

```

pizzalocker_pickup

```python
import json
import boto3
from boto3.dynamodb.conditions import Key, Attr

REGION="us-east-1"
dynamodb = boto3.resource('dynamodb', region_name=REGION)
orderTable = dynamodb.Table('PizzaLocker_OrderItems')
userTable = dynamodb.Table('PizzaLocker_Customer')

def lambda_handler(event, context):
    orderID = event['params']['path']['id']
    orderResponse = orderTable.scan(
        FilterExpression=Attr('OrderID').eq(orderID)
    )
    storeID = orderResponse['Items'][0]['StoreID']
    deleteOrderResponse = orderTable.delete_item(
        Key={
            'OrderID':orderID,
            'StoreID':storeID
        }
    )
    userResponse = userTable.scan(
        FilterExpression=Attr('orderID').eq(orderID)
    )
    userID = userResponse['Items'][0]['username']
    deleteUserResponse = userTable.update_item(
        Key={
            'username': userID
        },
        UpdateExpression="set orderID=:o",
        ExpressionAttributeValues={
            ':o': None
        }
    )

    return {
        'statusCode': 200,
        'body': json.dumps('Order Deleted')
    }

```

Appendix C – Sample AWS API Setup and Database Setup

API Gateway



Order DynamoDB Table

Tables (3)

Any table tag

Find tables by table name

- PizzaLocker_Customer
- PizzaLocker_Employees
- PizzaLocker_OrderItems

PizzaLocker_OrderItems

Autopreview | Actions | Create item | Update table settings

Scan/Query items

Scan | Query

Table or index: PizzaLocker_OrderItems

Filters

Run | Reset

Completed Read capacity units consumed: 0.5

Items returned (6)

OrderID	StoreID	Employe...	LockerN...	Order	OrderSta...	Temperature
165238982	atlanta2	atlanta2$1...	null	[{"M":{"t...	PLACED	null
477487253	atlanta2	atlanta2$4...	3	[{"M":{"t...	READY	165
125416638	atlanta2	atlanta2$1...	1	[{"M":{"t...	READY	155.7
699002314	atlanta2	atlanta2$6...	1	[{"M":{"t...	READY	155.7
381002324	atlanta2	atlanta2$3...	2	[{"M":{"t...	READY	165
435735770	atlanta2	atlanta2$4...	1	[{"M":{"t...	READY	155.7

Example Order Item

Item editor

Form | JSON

Attributes

Add new attribute

Attribute name	Value	Type
OrderID - *Partition key*	165238982	String
StoreID - *Sort key*	atlanta2	String
EmployeeBarcode	atlanta2$165238982	String
LockerNumber	Null	Null
Order	Insert a field	List
0	Insert a field	Map
size	Small	String
type	Pepperoni	String
OrderStatus	PLACED	String
Temperature	Null	Null

Cancel | Save changes