IoT: Automated Inventory Management & Route Optimization

Routing through Sensor Networks

Final Report

Team Number: sdmay19-29
Client: Jimmy Paul (Crafty)
Adviser: Dr. Goce Trajcevski

Team Members:
David Bis — Meeting Facilitator, Back-End Developer
Hanna Moser — Meeting Scribe, Front-End Developer
Adam Hauge — Report Manager, Computer Network Architect
Ben Gruman — Resource Acquisition, Hardware Architect
Sam Guenette — Public Relations, Back-End Developer
Noah Bix — Documentation Manager, Hardware Architect

Team Email: sdmay19-29@iastate.edu
Team Website: http://sdmay19-29.sd.ece.iastate.edu/

Revised: April 29, 2019

Table of Contents

1 Introductory Material **1**
1.1 Acknowledgment 1
1.2 Problem and Project Statement 1
1.3 Operational Environment 1
1.4 Intended Users and Intended Uses 2
1.5 Assumptions and Limitations 2
2 Requirements Specification **3**
2.1 Functional Requirements 3
2.1.1 Sensor Network 3
2.1.2 Analytics Software 3
2.2 Constraints Considerations 3
2.2.1 Non-Functional Requirements 3
3 System Design & Development **5**
3.1 Proposed Design 5
3.1.1 Sensor Modules 5
3.1.2 Sensor Network 6
3.1.3 Database 8
3.1.4 Project Back-End 9
3.1.5 Front-End User Interface 10
3.2 Development Process 11
4 Implementation **13**
4.1 Implementation Diagram, Technologies, Software Used 13
4.2 Rationale for Technology/Software Choices 14
4.3 Applicable Standards and Best Practices 16
5 Testing, Validation, and Evaluation **18**
5.1 Test Plan 18
5.1.1 Interface Specifications 18
5.1.2 Hardware and Software 18
5.1.3 Functional Testing 19
5.1.4 Non-Functional Testing 20
5.2 Evaluation 21
5.2.1 Functional Testing Evaluation 21

5.2.2 Non-Functional Testing Evaluation 23
6 Project and Risk Management 25
6.1 Task Decomposition & Roles and Responsibilities 25
6.1.2 Task Decomposition by Team Member 25
6.2 Project Schedule 25
6.3 Risks and Mitigation 27
6.4 Lessons Learned 28
7 Conclusion 30
7.1 Closing Remarks 30
7.2 Future Work 30
References 31
Appendix A: Datasheets 32
Appendix B: Screen Sketches 34

List of Figures

Figure 1: Weight Sensor Circuit Diagram
Figure 2: Sonar and Weight Sensor Wiring Diagram via ESP8266
Figure 3: Sonar Sensor Wiring Diagram via ESP8266
Figure 4: 3-Way-Handshake Algorithm for Network Communication
Figure 5: Deployment Diagram of Solution Components
Figure 6: Implementation Diagram
Figure 7: Relevant Technologies, Frameworks, and Libraries
Figure 8: LCD Setup for Raspberry Pi
Figure 9: Process Diagram
Figure 10: Raspberry Pi Model Zero-W Pinout
Figure 11: Database Schema
Figure 12a-h: Front-end Screenshots

List of Tables

Table 1: Raspberry Pi Connection Test Results
Table 2: Stress Testing Results
Table 3a-d: Gantt Charts

List of Definitions

Master-slave System: A type of software design where multiple smaller *slave* components carry out work, communicate, and are controlled by a single *master* component.
Sensory Device: A collection of sensors assigned to a specific product for the sensor network.
ADC: Analog to Digital Converter
SYN: Synchronize Sequence Number. A type of networking packet used to request or establish a connection.
ACK: A type of networking packet used to acknowledge a request for connection.
RFID: Radio Frequency Identification. Uses electromagnetic waves to identify and monitor the location of tags attached to certain objects.
UPC: Universal Product Code

1 Introductory Material

The objective of this project is to implement a system that can effectively monitor goods in office pantries and create efficient orders based off their current status. Using these orders, an optimal delivery route is to be calculated to each office location.

1.1 Acknowledgment

The Inventory Automation Team would like to thank the Iowa State University Department of Electrical and Computer Engineering for providing this team with a professional experience, quality resources, and consultation with experts. The team appreciates the willingness of the Electronics and Technology Group (ETG) to help provide hardware and server components for the project. Special thanks also go to Iowa State's Dr. Goce Trajcevski for weekly consultation with regards to dealing with technical issues and moving forward throughout the development process. Appreciation also goes towards our client, Crafty, LLC., for the given project. Special thanks goes to CTO and co-founder of Crafty, Jimmy Paul, for making time to meet with the development team to gain more information and feedback throughout the project's development.

1.2 Problem and Project Statement

Crafty, LLC is a warehouse company that delivers food to office pantries. Their current infrastructure is based on having an employee at each of their client offices handling shipment orders and physically monitoring when the company's pantry needs to reorder certain products, which is prone to human error. Furthermore, warehouse truck routing becomes severely inefficient and expensive from restocking at individual offices based on separate orders.

Our objective is to provide an integrated solution that enables more effective management of office pantries. Specifically, we aim to balance local (i.e. at the level of individual offices) and global (i.e. at the level of a geographic region) management in terms of both customer satisfaction and overall efficiency for Crafty. Towards that, our proposed solution consists of two main collaborative modules that we've developed: (1) a microcontroller device that monitors the quantities of different items in an individual office pantry, coupled with other existing monitoring tools (e.g. weight scales, sonar sensors); (2) software tools that combine the multiple restocking notifications from different offices, and plan effective shipment (i.e. routes) and delivery.

1.3 Operational Environment

The microcontroller used to monitor office shipment levels is expected to be stored in a dry pantry area with a reliable power supply and WiFi connection. The current device consists of a microcontroller-connected "Smart-Shelf" that is expected to fit on shelves. Any software setup with the device can be done through a web application. The web application is available for Crafty employees stationed at office locations for setting up the physical device and in the warehouse for overseeing and handling shipment orders.

1.4 Intended Users and Intended Uses

There are three main users of our sensor network. Each user interacts with the sensor network through use of the web application component. The following are brief overviews of the three users:

1. *Pantry Employee*: Actor based in a given company office in charge of handling shipments sent from the warehouse. This user is in charge of setting up the microcontroller device and registering items for the device to monitor.

2. *Warehouse Employee*: Actor based in the company warehouse in charge of overseeing shipment orders and filling trucks with the proper items. This user goes into the application to view orders and routes. This user is expected to possess moderate technical experience.

3. *Warehouse Truck Driver*: Actor in charge of the physical delivery of items between the Crafty warehouse and client office pantries. This user employs the online application to view their assigned shipping route for the day.

1.5 Assumptions and Limitations

Assumptions:

- Pantry sensor network has access to a reliable power source
- Pantry sensor network can connect to internet
- Pantry sensor network is not at risk of water damage
- Pantry Employee properly sets up sensor network and associates each monitoring device with the corresponding product being stored through the online application
- Trucks stop at destinations for a constant amount of time
- Traffic variance is global across all roads
- All units of product are the same size, thus the truck loads can be normalized to a constant number of units
- Traffic variance is assumed to be provided by various external sources
- Only complete orders will be added to a truck
- Monitoring devices are registered to a company

Limitations:

- Accuracy of automatic orders is completely reliant on the accuracy of the sensors used in the sensor network
- Sensor network monitors product availability in bulk, not individually
- Sensor network can only make correct orders if the Crafty user properly establishes what type of products are being monitored

2 Requirements Specification

In this section we discuss both functional and non-functional requirements and their use cases.

2.1 Functional Requirements

The project solution can be broken into two components: a sensor network and analytics software. The functional requirements for both components are shown below.

2.1.1 Sensor Network

FR.1: The system can keep track of the levels of various product in a customer's pantry stockroom automatically. This inventory should be sent over to Crafty to determine whether a product needs to be restocked.

FR.2: The user should be able to register/unregister sensor arrays to fit their unique stockroom.

2.1.2 Analytics Software

FR.3: The system can calculate an optimal route for deliveries for restocking client pantry stockrooms daily.

FR.4: The system can display the stockroom inventory data and filter it according to refined searches.

2.2 Constraints Considerations

In order to ensure that the project satisfies the necessary requirements, the team has taken into consideration the following constraints. These constraints include relevant non-functional requirements and standards to follow, such as IEEE standards.

2.2.1 Non-Functional Requirements

Scalability - The sensor network should be able to be replicated and integrated into multiple pantry stockrooms. The sensor network should also be able to easily support a variety of different products that can be found in stockrooms. This requirement is accomplished through designing the sensor network to be modular and adaptable.

Data Integrity - The sensors should be accurate and consistent throughout their lifetimes in order to ensure the inventory is correctly monitored. This has been accomplished by employing a variety of sensors to audit each other's the readings. The sensors have also be tested to ensure accuracy.

Availability - The system should be available to update the inventory at least once per work day. It should also be available to determine delivery routes once per work day.

Deployment - The sensor network should be easily deployed and installed in pantry stockrooms. This has been be accomplished by designing the sensor network architecture to be modular and adaptable to a variety of environments.

Usability - The initial setup of the sensor network by the Crafty employee should be intuitive. Since the amount of sensors and the product they are detecting differs from stockroom to stockroom, and can be changed in a stockroom, the assigned Crafty employee should be easily able to select which sensor is monitoring a specific product.

Resilience - The sensor network is intended to be deployed in an environment of constant change due to the movement of boxes in and out of the stock rooms. This can run risk of damaging the system if a box comes into unwanted contact with the sensors. The sensor arrays should be designed to be constantly protected by the expected motions of boxes in the environment.

3 System Design & Development

In this section we will discuss our design decisions and approaches in detail.

3.1 Proposed Design

The design for this solution consists of five major components:

1. Sensor Modules
2. Sensor Network
3. Database
4. Project Back-End
5. Front-End User Interface

The sensor modules are used to approximate the quantity of any given item in the stockroom. The sensor network is controlled by a master Raspberry Pi using socket communication with the sensor modules to continuously collect inventory data throughout the day to be sent to the database. The front-end of the web application allows users to login, view inventory levels, and edit monitoring device information. The back-end handles automatic reordering of products once they have reached their threshold level. Sections 2.1.1 through 2.1.5 describes the functionality of each component in more detail.

3.1.1 Sensor Modules

The sensor modules consist of a combination of weight sensors and sonar sensors . These sensors work together to calculate the weight and depth of the inventory used to output an estimated calculation on the total inventory. Each monitoring device has a weight sensor as a primary component to monitor product levels. The sonar sensor is intended to act as a backup to monitor when a package has been placed on or removed from the device.

The weight sensor (Figure 1) utilizes a bridge circuit which alters in total resistance as the load cell bends under the weight of items placed on top of it. The total resistance of the bridge circuit directly relates to a given output voltage. This voltage is converted to a digital value through an ADC and is then communicated back to the Raspberry Pi via an ESP8266 chip with the wiring specifications below (Figure 2). These values can then be quickly calibrated for each specific product on the weight scale. Calibration is done by taking a voltage reading for one item on a single scale, and equating that value as one unit. As items are added and removed from the sensor, the value output by the sensor automatically adjusts with respect to the unit value. Once a reference exists for a given product, future sensors can be calibrated by leaving the sensor empty, assigning the resulting value to zero, and shifting the reference accordingly.

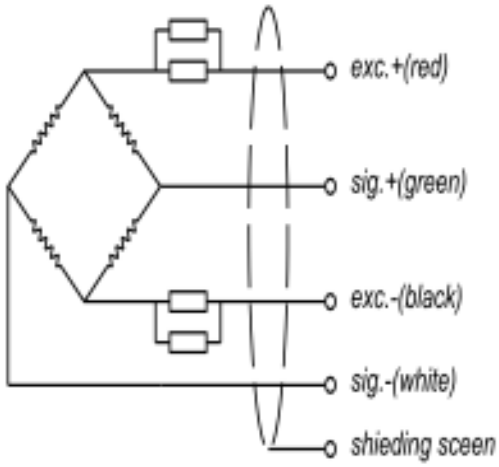


Figure 1: Weight Sensor Circuit Diagram

The sonar sensor sends a pulse signal out and detects the time it takes for the signal to reach an object and bounce back. The amount of time required for the pulse signal to return to the sensor is then reported as a value to the master Raspberry Pi. This sensor uses an ESP chip as well with the wiring specification in Figure 3. To make sense of these values, the sensors are calibrated to display distance using regression analysis. Once the sensor is calibrated, it then is able to track the depth of each stack of products.

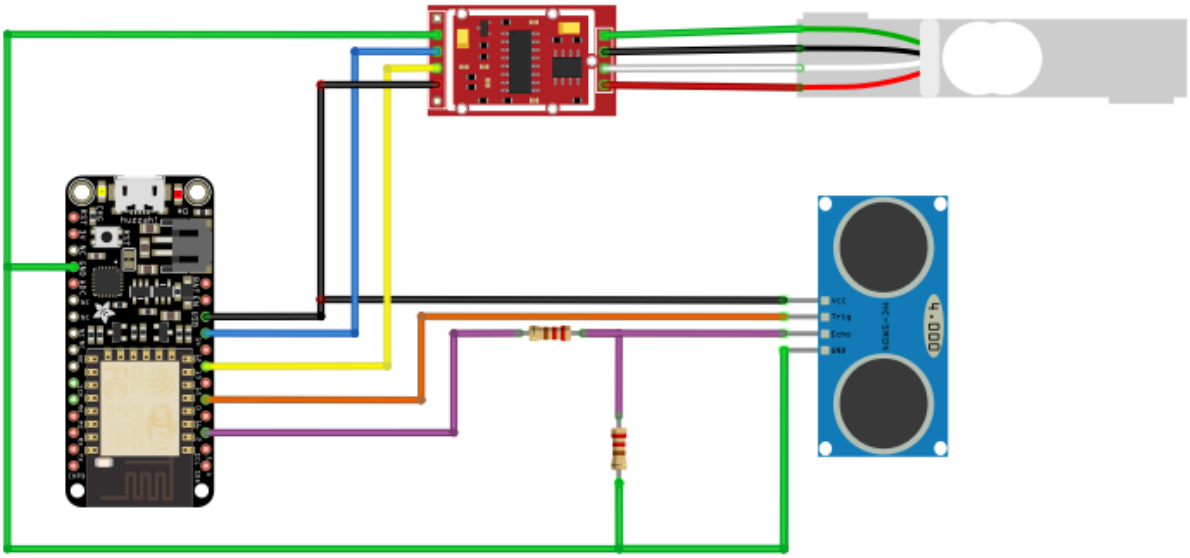


Figure 2: Sonar and Weight Sensor Wiring Diagram via ESP8266

3.1.2 Sensor Network

The sensor network, controlled by a Raspberry Pi, is responsible for gathering all inventory data throughout the day. The Raspberry Pi is programmed to act as a server continuously waiting for sensor module clients to request a connection and transmit data. Connections are established

through the use of a 3-way-handshake algorithm (as shown below and through figure 2) using the Raspberry Pi as the server and the sensor modules as clients. Upon receiving information from any of the stockroom's sensor modules, the Raspberry Pi packages the data to be sent over to the database before closing the connection.

The 3-way-handshake employs the following algorithm:

1. Sensor Module sends a SYN packet to Raspberry Pi to initiate connection.
    - SYN packet contains all necessary sensor module identification data.
2. Raspberry Pi receives SYN packet and sends a SYN-ACK packet to Sensor Module.
    - SYN-ACK packet acknowledges that the Raspberry Pi is ready to receive data.
3. Sensor Module receives SYN-ACK packet and sends ACK packet to Raspberry Pi.
    - ACK packet contains all necessary sensor data needed to calculate total inventory data.

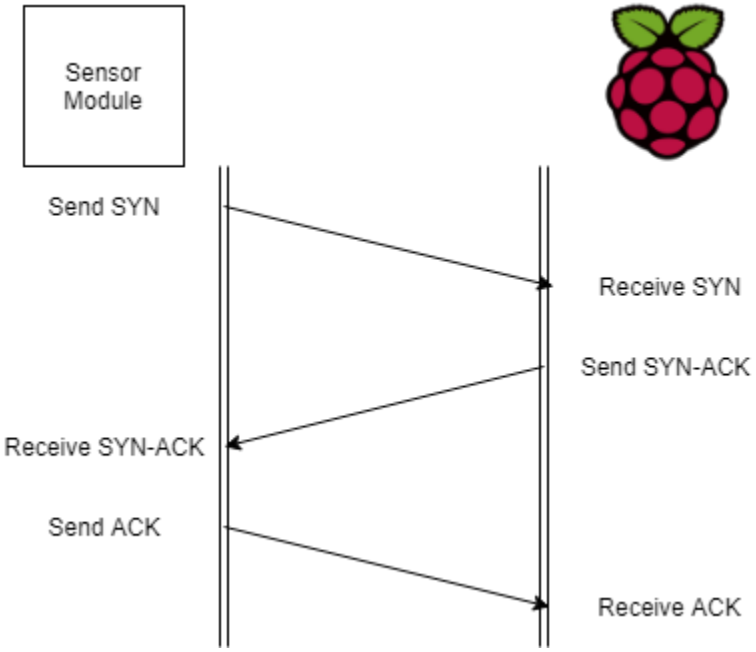


Figure 3: 3-Way-Handshake Algorithm for Network Communication

All communication between the Raspberry Pi and the Sensor Modules is handled via socket communication. Due to this, it is important to ensure that the server program is working consistently. Furthermore, ensuring a consistent server for handling raw data is required to meet the non-functional requirement for data integrity. One possible issue that could arise is that a client could request to connect to the server by sending a SYN packet, but never sends the proceeding ACK packet. This could be caused by a faulty sensor module or possibly as an attack on the sensor network. In order to mitigate this, all connections are given a time constraint. All transactions that are not completed within the given time constraint are discarded. This ensures that the server is never busy for too long and that all sensor modules have an opportunity to connect to the server. An issue that arises with the 3-way-handshake connection is the possibility of a SYN flood attack on the server. SYN flood attacks occur when a client repeatedly sends SYN packets in an attempt to occupy all the memory on the server and prevent legitimate

clients from making a connection. In order to mitigate this, the server only accepts packets from registered sensors and does not send any ACK packets to clients not identified in the sensor network. Any SYN packets from clients not recognized by the Raspberry Pi are discarded immediately.

The sensor network is set up through a series of preloading all necessary programs onto the ESP8266 wireless chips and the Raspberry Pi before connecting to power. All WLAN data needs to be loaded onto the Raspberry Pi's micro SD card in the file "/etc/wpa_supplicant/wpa_supplicant.conf". The Raspberry Pi is also pre-assigned a static IP address which the ESP8266 wireless chips are programmed to connect with. Because of this, the sensor network is automatically setup upon each module powering on.

In order to preload WLAN connection information onto the Raspberry Pi, the following must be added to /etc/wpa_supplicant/wpa_supplicant.conf

```
network={ ssid="my-network-name"
          psk="my-network-pass"
          key_mgmt=WPA-PSK
        }
```

When it comes to setting up the sensor network, the Raspberry Pi can automatically detect what sensors modules are connected to the local network. Since the ESP8266 chips on each sensor module have an open port, they are be discoverable to the Raspberry Pi, which stores each module's identification locally after a successful connection. A unique ID is then assigned to each sensor module and forwarded to the database. If a sensor module loses connection with the Raspberry Pi, the database does not update during the regularly scheduled inventory measurement. In this case both the Raspberry Pi and the back-end flag the sensor network as disconnected.

3.1.3 Database

The database is implemented as a relational database with MySQL connected with primary and foreign keys between tables. The database schema is shown in Appendix A. The database contains a single repository of certified devices as a security precaution. Whenever a new monitoring device is registered, the device must send an ID matching a unique ID in the *ActiveDevice* table that is not currently in use.

The *company* table used to keep track of *clients*, including their name and address. All rows in the *users* database, which contains login information, links to an instance in the company table. This is used for both login security and as a key reference when storing orders. The *product* table stores information on every product offered by Crafty that clients can order from. This stores information like name, weight, serving size, etc.

The *monitoringDevice* table stores every monitoring device that can send information to the database. Each monitoring device contains foreign key reference to the client that owns the device (*clients foreign key*), the current product being monitored (*product foreign key*), the

product's minimum threshold, maximum threshold, current inventory level, and a boolean to tell whether or not the device is registered. If a device is not registered, then values for threshold, product monitored, and current level default to null. These values change when the user registers a product to monitor on the web component. Since the devices are pre-assigned to a company; the company ID will never be null .When a threshold for one product changes, the threshold changes for all other monitoring device instances that share the same product and company values.

The *orderLog* table stores shipping orders and historical data of shipping orders. There is only one active shipping order at all time, which is specified by by a boolean value "newOrder". Each entry also contains a shipping ID, and the date the shipping order was made.

The *orderItem* table represents the contents of each order from the *orderLog* table. Whenever a client is running low on an item, an entry containing the company id, the product being ordered, the quantity of the, and the order ID corresponding to the most active order ID in the *orderLog* table.

The *Routing* table stores a single instance the truck routes for the most recent order. This stores JSON data that indicates information such as the number of trucks needed and the delivery routes for each truck

3.1.4 Project Back-End

The back-end is implemented in a NodeJS environment using the ExpressJS framework. This component interfaces directly with the database and processes data for further analysis. The Express API is designed to manage data flow between the front-end and the back-end. There are be four types of HTTPS requests used: GET, POST, PUT, and DELETE. GET calls retrieve data from the database. POST calls are intended for only uploading new data. PUT is be used for updating existing entries in the database. DELETE is used for removing entries from the database. Each endpoint accesses the database using the Sequelize library. This is to take advantage of the object-relational mapping (ORM) properties of the database. Sequelize pulls information from the database and converts it into classes with the proper inheritance of information between datatables. This allows developers to make queries and access database information more efficiently. The Sequelize library is capable of being paired with the *sequelize-auto* library, which includes a command line interface that can take an existing database schema and generate the necessary data access files for each table. It is important to add the additional field *timestamps* with value *false* into each table data access file because the default value is true and would throw an exception otherwise.

The back-end is required to process the data on a regular basis. To do this, the *node-scheduler* library is used to run a series of processes at of end of each weekday to determine and create orders for each client based on their inventory levels. This is done by comparing the current levels of each product with their specified threshold. Monitoring devices that monitor the same product have their current levels summed and then compared with the product threshold. If the current level is below the threshold, it is added to the order list. The items ordered are stored the

orderItem table the entire order tracking information in the *orderLog* table. The orderLog table is used to track of Crafty's order history. One orderLog contains multiple orderItems.

After the orders are determined, an optimal delivery route is created. An optimal route is determined by the shortest transit time with the least amount of trucks. Additional parameters include limiting truck routes to no more than 8 hours, and ensuring that each truck has a limited capacity. It is also assumed that every stop along the route takes a constant amount of time. The *products* table stores information regarding packaging information, such as weight and size. Order contents are assumed to be entirely on the same truck in order to reduce the number of stops by the drivers.

Based on the contents of each truck, an optimal order of locations to stop must be determined. However, trying to find the most efficient path is similar to the Travelling Salesman Problem, which is an NP-Hard problem. To resolve this, an improved version of the Clarke-Wright Savings Algorithm is employed to calculate the most optimal route in an acceptable amount of time.

3.1.5 Front-End User Interface

The front-end of the web application is implemented using the ReactJS framework. There are two types of views: admin and customer. The purpose of the UI is to provide the user with visualization of the pantry data, as well as provide them with an interface necessary for configuring the sensor network. ReactJS allows for breaking down each web component into independent exportable objects, leading to easier implementation and testing. The client view has three main pages: Inventory, Devices, and Recent Order. The admin view has four main pages: Inventory, Devices, OrderLog, and Routing. There is also a login page which allows for authorization of different users. All front-end screen sketches can be found for reference in Appendix B.

When a client logs in, they are immediately taken to the Inventory page. On this page, the client is allowed to view each of the products in the pantry associated with the client's company ID as well as update both the minimum and maximum thresholds associated with each product by selecting the row of the product they wish to update. On the Devices page, the client is able to see all of the monitoring devices registered for their company along with the product each device is monitoring alongside that product's corresponding minimum and maximum threshold. By selecting a row associated with a device, the client may update the product, minimum threshold, and maximum threshold. The client may also register any unregistered devices associated with their company on this page. On the Recent Order page, the client can view each of the products and corresponding amounts of the products of the most recent order associated with the client's company.

Similar to a client view, when an admin logs in, they are immediately taken to the Inventory page. On the Inventory, Devices, and OrderLog pages, the client can view the same data and perform the same actions as a client can. The only difference is that the admin can perform these actions for each of the companies in the database, which is why there is a select dropdown

of the companies in the database at the top of each of these pages. Admins are provided with one more page than clients: the Routing page. On this page, admins can view each of the current routes, including each of the waypoints along the route. Admins can also select the row corresponding to the route they are interested in, which will pull up a modal with a Google map visual of that route as well as a printout below the map with the details of each of the waypoints along the route.

Communication with the backend is handled via HTTP protocol. GET, PUT, and POST requests are used to query and mutate objects in the database. Axios is used to provide promised-based HTTP requests, where a promise is provided for the eventual completion of a request, whether it be a success or failure.

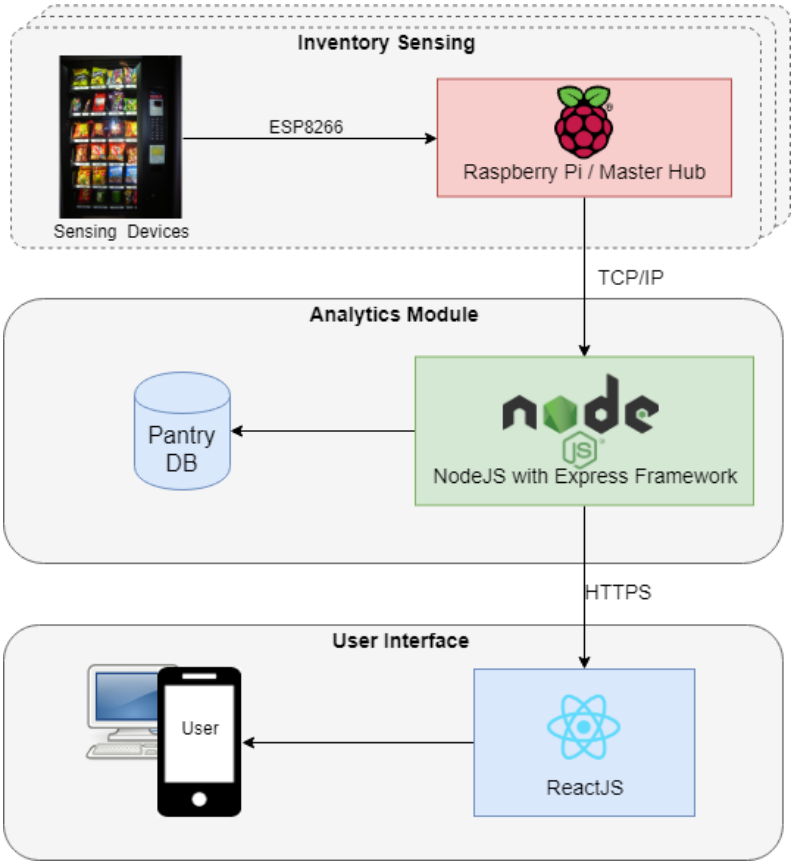


Figure 4: Deployment Diagram of Solution Components

3.2 Development Process

The project is broken into two major components: the sensor network and the analytics software. The two components are developed simultaneously, having three members assigned to the sensor network and three members assigned to the analytics software. However, design plans and decisions were all worked on as team. This ensures all possible designs have been considered in order to maximize the overall quality of our plans. The project has been divided

into three phases, each advancing the development of a prototype to final product. The goals for each phase are as follows:

Phase 1

- Sensors can collect data
- Sensor data can be sent to remote database
- Front-end can visualize data from the database

Phase 2

- Integrate system for multiple sensors to be registered to a specific product (assemble sensor array)
- Orders can be generated based on measured inventory data

Phase 3

- Integrate multiple sensor arrays for variety of products (integrate master-slave system)
- Route can be determined from multiple order requests

Each phase was treated as an Agile sprint to ensure continuous improvement upon the planned architecture. During each phase, our work went through testing and validation requirements for further refinement and approval. The rationale behind the Agile process is due to the team's familiarity with the framework. Due to the necessity to balance this project among other concurrent projects for classes, the Agile process helped to keep us to our deadlines, so that we could continuously deliver satisfactory progress to our client. Likewise, the process acted as a good tool to help break down the project into easily digestible phases, ensuring that we were staying on track with the project and not skewing away from the originally identified requirements.

4 Implementation

In this section we will elaborate on the types of software and hardware technologies used throughout the development of this project.

4.1 Implementation Diagram, Technologies, Software Used

An implementation diagram of the project solution is shown below.

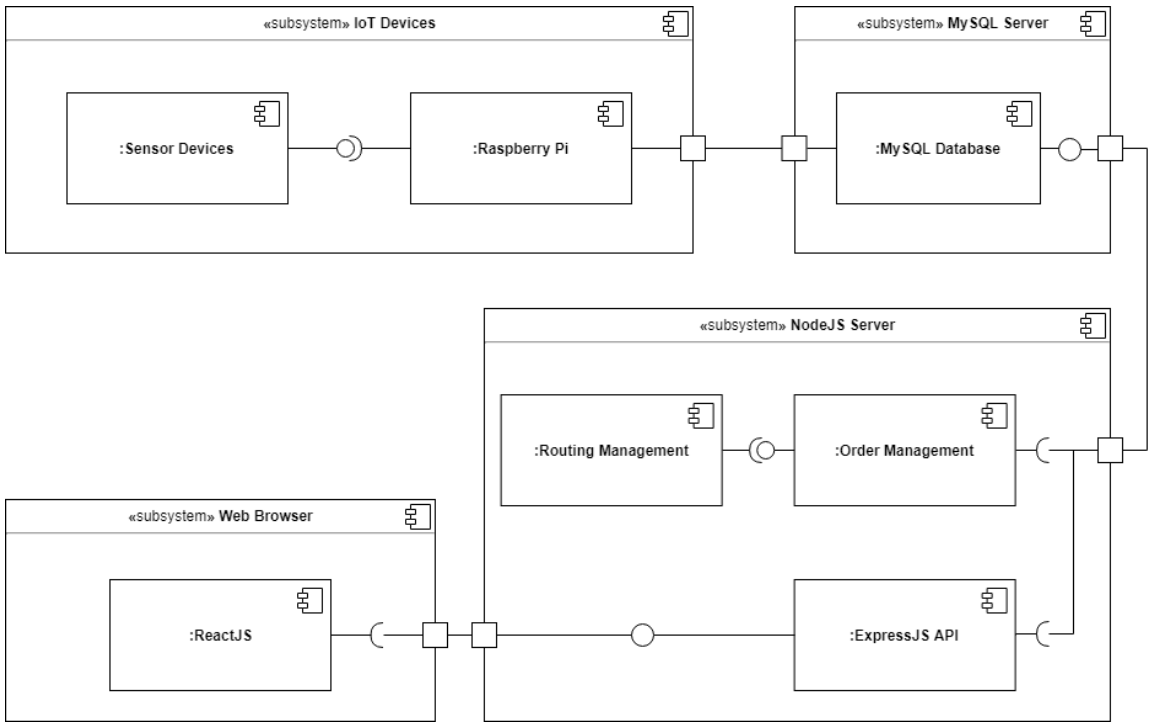


Figure 5: Implementation Diagram

The technologies, frameworks, and libraries used are shown in the diagram below.

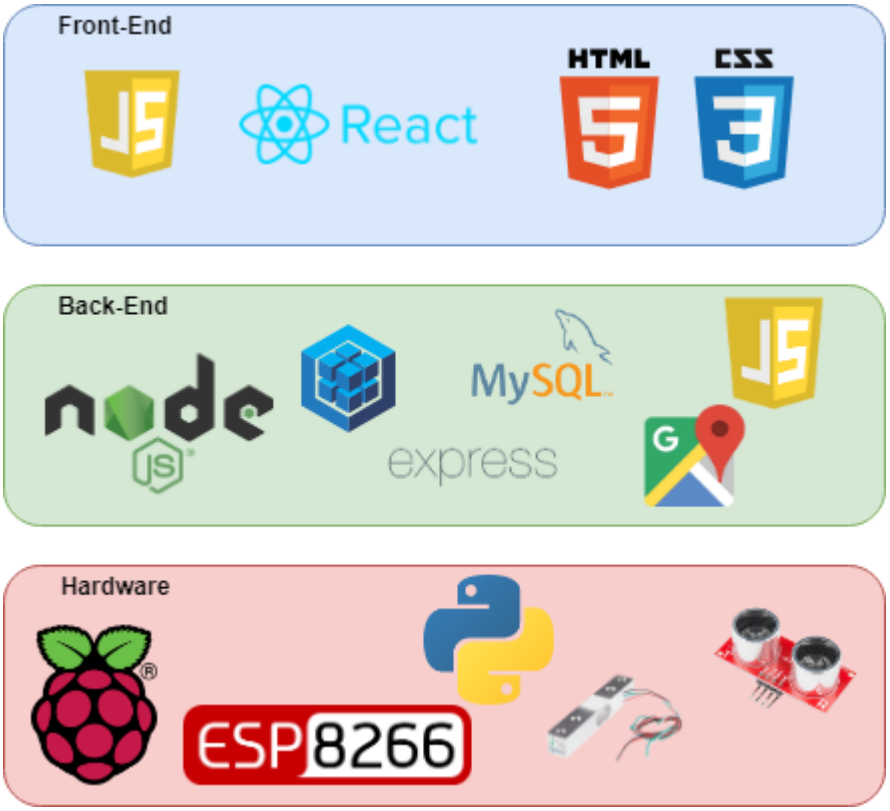


Figure 6: Relevant Technologies, Frameworks, and Libraries

4.2 Rationale for Technology/Software Choices

There are many pros and cons to this architectural design. The three-tier setup allows for easy maintenance and updates to software components without negatively affecting the rest of the project. This also presents an ease-of-use for the user since they simply have to plug in their device, and then finish the setup through a simple form on the web-application.

In terms of physical sensors, RFID sensors were thought of as an alternative solution to the sensor network. However, the other sensors were selected for their simplicity and cost factor. While RFID sensors would compliment our design well, the initial cost of RFID sensors and the capitalized cost of RFID sensor tags were concluded to be too expensive for our project. Labor is also a factor with the RFID sensor, Crafty employees would have to spend time and labor placing tags on every item that is shipped in.

Weight sensors are much more effective as the primary sensor in our network. The weight sensor eliminates the capital cost that comes with an RFID sensor. Weight sensors also reduce the labor factor of applying RFID tags to each Crafty product. With this solution the error factor should be minimized as well. This is due to the fact that the weight of each product has to be within a certain range instead of an exact value. Sonar sensors are effective as a secondary,

auditing component. By monitoring depth along with information collected from weight sensors, inaccuracies and errors can be detected through outlier combinations of sensor data.

An alternative considered for the master computer for the sensor network was an Arduino microcontroller, as opposed to the Raspberry Pi. While both options provide sufficient features to satisfy the requirements, the Raspberry Pi was selected due to its ease of use for development purposes. An important difference is the Raspberry Pi's on-board operating system (OS). A non-functional requirement for the project was scalability. Since Crafty services multiple pantries, it is important that the sensor network is modularly configurable. Since the OS is on-board the Raspberry Pi, it enables the user to easily replace and alter sensor array configurations. The OS enables changes at run-time through a program written for the device. One last difference is that the Raspberry Pi has an on-board wireless capability, whereas the Arduino does not. Since the master computer must make calls to upload data to the database, the feature is convenient.

The use of wireless socket communication throughout the sensor network allows for easy sensor module organization: the sensor modules are able to be placed anywhere in the stockroom in any possible configuration. This means that our solution is able to be implemented in any possible stockroom given that the assumptions of constant power and network connection hold true. Unfortunately, using the socket communication also means that each sensor module and the Raspberry Pi must always be connected to the WLAN in the stockroom. However, connecting each sensor module and the Raspberry Pi to the network requires an extra step of connecting each piece of hardware to the Wi-Fi network.

By using one-way communication from monitoring device to the database, the accuracy of the web-component analysis relies completely upon the data being sent. Accuracy is also completely reliant on the weight sensors' accuracy.

Using an SQL-Database can also make things a little more difficult for long-term analysis. Unlike databases like MongoDB, which has its data nested to easily obtain association in information; SQL adds a layer of difficulty through its key-value architecture, making it difficult to obtain and group information in the proper format to analyze. However, since our primary objective is inventory and order tracking, a row-column database setup was an ideal choice for storing and retrieving information easily and optimally.

The NodeJS back-end is beneficial in that it is asynchronous by nature. This makes it easy to manage multiple simultaneous processes. Although we do not expect our back-end to experience heavy traffic, the environment is able to help mitigate any potential concurrency issues. The NodeJS environment also offers up a number of easily integrated external libraries, or node modules. This reduces the amount of necessary code to write, and simplify new code. However, this runs risk due to the increasing number of dependencies. If any of the libraries used stops being maintained or has changes that affect our application's performance, we would have to be reactive to the changes of our dependencies.

The route optimization is challenging to optimize as it is equivalent to solving the "travelling salesman problem", which is an NP-hard problem. To solve this, a modified version of the

Clarke-Wright Savings Algorithm has been used to simplify the computation. The algorithm takes input as a distance matrix, where the first row and column correspond to the warehouse, and a list of quantities required by each destination. The distance matrix is obtained through the use of the Google Maps API, where distance is considered as time to travel between locations in seconds. Assuming that there are n stops to make in a given day, the algorithm initializes n routes, each one from the warehouse to one unique stop and back. From here, the algorithm continuously calculations the savings of each link addition to a route. The savings of each link is calculated with the following equation:

$$Savings(i,j) = [dist(1,i) + dist(j,1) - dist(i,j)] * trafficFactor(mergedRouteDistance)$$

Where i and j are the stops considered, and the trafficFactor is a factor between 0 and 1 of the expected traffic over the course of the potentially merged route. A 1 would mean nominal traffic (traveling at the speed limit), and a 0 would mean a complete stop. Each iteration, the savings are saved into a list, and sorted in increasing order. The link with the greatest savings is removed from the list, and considering for merging. The link checks whether two routes in the tour can be merge. The merge conditions are as follows:

1. Both i and j are adjacent to the warehouse in the route
2. The merge operation will not make the route exceed the maximum truck capacity or the maximum route duration

If both conditions are true, the routes are merge. Once all links are removed from the savings list, the algorithm terminates and the outputs a list of routes for the trucks to make that day.

In the case of a traffic abnormality, each route is recalculated to consider the change in transit time. Since each truck has the products for certain destinations, other trucks cannot be rerouted mid-tour to satisfy new destinations. Only existing destinations are able to be removed from routes if the time constraints rise above the allocated time.

ReactJS helps break down the front-end into separate components, allowing for each to be independently implemented and tested. This helps ensure that the front-end appropriately satisfies each use-case.

4.3 Applicable Standards and Best Practices

The project abides to the following IEEE standards:

- **IEEE 802.11**: Wi-Fi between ESP8266 and Raspberry Pi. Being that our solution involves communication of information between a local Wi-Fi and our own and database, development has to follow appropriate networking standards. Likewise, it is important that each of sensor devices are abide to the standard Wi-Fi protocol so that they can properly connect to the network and be discovered by the master Raspberry Pi.
- **IEEE 1012-2016 - IEEE Standard for System, Software, and Hardware Verification and Validation:** The project contains a rigorous testing suite to ensure

the solution satisfies all the necessary requirements. This helps verify and validate the practicality of our solution once it is implemented.

- **IEEE 1532 In-System Configuration of Programmable Devices** is essential since since our product is an IoT technology involving a massive network of monitoring devices that record and communicate information to each other as well as a web component. This involves a significant amount of hardware configuration and embedded software that needs to be implemented effectively and securely.

5 Testing, Validation, and Evaluation

This section serves as an overview of our planned process for the testing, validation, and evaluation stages of the project.

5.1 Test Plan

In this section, we will discuss our step by step testing process for each individual component in detail.

5.1.1 Interface Specifications

In any project of this size, testing is necessary in order to ensure that all components work properly both individually and together as a unit. Our sensor modules are tested using scripts designed to continuously output data from the ADCs connected to the sensors. Different amounts of inventory are used with the sensor modules in order to ensure that the values returned by the ADCs are making realistic approximations to the true inventory values. The master Raspberry Pi runs test scripts that takes manual input through a socket connection and pushes the data into the database. This ensures that the Raspberry Pi is able to accept data and send it to the database seamlessly.

5.1.2 Hardware and Software

The sensor modules are tested through the use of python scripts designed to continuously output readings from the connected ADCs to be displayed on a monitor. These scripts are used to ensure that the data output from the sensor modules accurately depicts the amount of inventory being measured. Furthermore, when the amount of inventory is changed, these scripts show whether or not the sensor modules are able to detect the change.

In order to test the sensor network, a team laptop acts as a client by sending premade packets to the Raspberry Pi via a socket connection. This ensures that the Raspberry Pi is able to make viable connections with the sensor modules. Furthermore, incorrectly formatted data has been sent to the Raspberry Pi to ensure that bad packets are discarded from the network to ensure that the network is not be kept busy by misformatted packets. Upon successful testing of the communication sockets, the Raspberry Pi sends test data to the database in order to ensure a successful database connection.

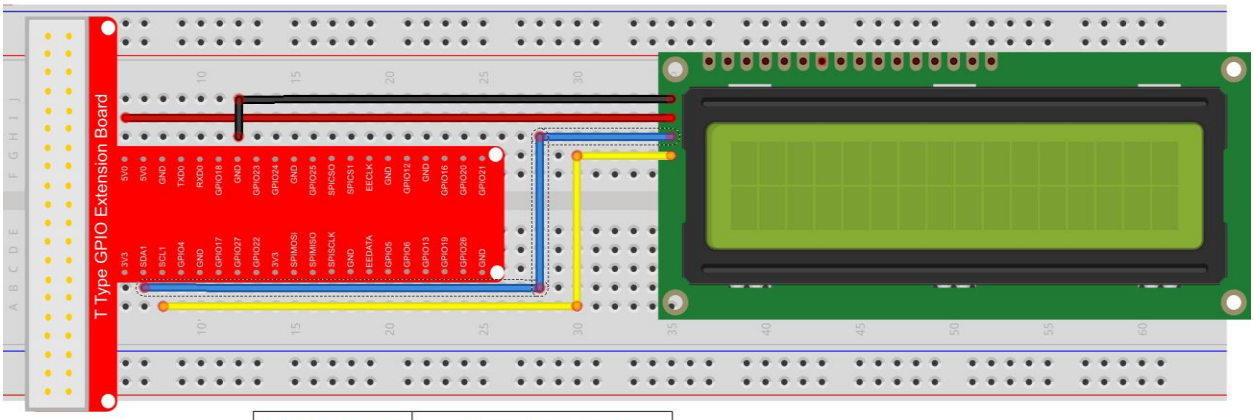
To automate the testing on the back-end, the Mocha and Chai testing libraries are used. These libraries are responsible for testing all data processing modules, such as the truck loading algorithm and route planning algorithm. Postman, a GUI software used to simulate REST API calls, is used to test the API and requested endpoints. Postman provides various features that help automate the test process, such as preparing test suites for calling the different endpoints, as well as feeding test data into the system.

5.1.3 Functional Testing

To test the validity of the sensors, various products of different shapes and weight are be placed on the scales. A snapshot of the data is taken before and after a product is added or removed. The two snapshots are then compared to determine the accuracy of the sensors. This applies to both the load cell and the sonar sensor.

The act of adding or removing new sensor arrays in the system is tested by a simple metric of success. Success is measured by the master Raspberry Pi's ability to detect and receive accurate data from any sensor array. If the master Raspberry Pi is able to detect the new sensor array and receive accurate data from it, the test is deemed successful, and a failure if otherwise. The same applies for removing a sensor array.

In order to assist in testing the Raspberry Pi, an LCD screen has been attached as shown in figure 8 below. This screen displays the IP address of the Raspberry Pi alongside its current network connection status.



1602LCD	Raspberry Pi
VCC	5V
GND	GND
SDA	SDA1
SCL	SCL1

Figure 7: LCD Setup for Raspberry Pi

A testing suite is used to test all of the written project software. This testing suite includes test cases using:

- Expected Data
- Fringe Data
- Failure Data

The tests are run in a variety of stages starting with unit tests. The following stage consists of testing after integration with the system. The purpose of this stage is to test not only the newly

integrated module, but to test existing modules to ensure they are still satisfy their requirements. The final stage consists of user testing, which focuses on validation of the user experience. The purpose of this stage is to test and verify the practicality of the solution. This stage includes testing front-end usage and sensor network deployment.

The objective of these tests are to verify that the solution saves the user time in the inventory management process. Through the tests, both functional and non-functional, sufficient evidence has been collected to prove a time save.

5.1.4 Non-Functional Testing

The project has been tested for satisfaction of the following non-functional requirements: scalability, data integrity, availability, deployment, usability, resilience. Procedures and success and failure criteria are be provided in section 5.2.

Scalability

Procedure

1. Begin with *k* sensory devices connected to the sensor network
2. Set-up a new sensor devices to connect to the sensor network
3. After completion, open front-end and check if *k+1* sensory devices are connected

Success Criteria: *k+1* sensory devices are identified on the sensor network
Failure Criteria: < *k+1* sensory devices are identified on the sensor network

Data Integrity

Procedure

1. Begin monitoring sensor data from Raspberry Pi. Save sensor data in a *.csv* file.
2. Transmit sensor data to database. Store data into *.csv* on back-end.
3. At the end of the observation period, compare the two *.csv* files to determine equality

Success Criteria: The two files are equivalent
Failure Criteria: The two files are not equivalent

Availability

Procedure

1. Schedule using the *node-scheduler* library an inventory process to happen at the end of every weekday
2. At the start of a new day, check if a new order and delivery route has been generated

Success Criteria: A new order and delivery route has been generated
Failure Criteria: A new order and delivery route has not been generated

Deployment

Procedure
1. Set-up sensory devices in desired location
2. Run Raspberry Pi sensory device discovery protocol
3. Check if three-way handshake is successful for each new sensory device

Success Criteria: Each new sensory device is discovered by the master Raspberry Pi
Failure Criteria: A sensory device is not discovered by the master Raspberry Pi

Usability

Procedure
1. Track the time it takes for one individual to manually count the inventory
2. Based on the inventory counts, determine an order for the day, end timing of process
3. Track the time it takes for the sensor network to log inventory and produce order
4. Compare manual and automatic times

Success Criteria: Automatic time is less than manual time
Failure Criteria: Manual time is less than automatic time

Resilience

Procedure
1. Once per day, add or remove product from the sensory device
2. Repeat process for a month
3. After a month of moving product, check to make sure the sensory device is not damaged

Success Criteria: The sensory device is not damaged
Failure Criteria: The sensory device is damaged

5.2 Evaluation

The following section contains testing results of our project solution.

5.2.1 Functional Testing Evaluation

The following tables display the testing results of the Raspberry Pi. A successful connection signifies the Raspberry Pi's ability to fully receive a complete packet and send it to the database. An LCD screen was attached to the Raspberry Pi in order to help with testing as shown previously in figure 8.

Raspberry Pi Connection Test

Sensor Module Test #	Packet Format	Connection Status
1	ID + Weight + Sonar	Successful
2	ID + Weight + Sonar Sent at the same time as module 1	Successful
3	ID + Weight + Sonar No connection termination	Connection successful. Timed out after 10 seconds.
4	ID + Sonar	Rejected
5	ID + Weight + Sonar + Random Number	Rejected
6	ID + Weight + Sonar Module unplugged before termination connection	Connection timed out after 10 seconds.

Table 1: Raspberry Pi Connection Test Results

As can be seen by table 1, only correctly formatted packets were accepted by the Raspberry Pi. Furthermore, all connections were successfully terminated. This shows that the Raspberry Pi functions correctly as a sensor network control unit.

In order to easily test our code, the Mocha testing framework was used automate unit testing. Each method was paired with a set of unit tests to verify expected results. Automation was also used for the integration testing by running groups of methods and ensuring the stream of processing still expected results.

Some features needed to be manually tested to ensure accuracy and practicality. One of these manual tests was for the routing algorithm. Based on feedback from our client, we do not expect to handle more than 100 stops per day for to be considered for the routing algorithm. To ensure that the algorithm would be able to produce a result in a reasonable amount of time, distance matrices of dimension n x n were randomly generated and timed for their practical runtime. The average of ten trials were collected for matrices of size n=10, 100, and 200. The results are shown below.

Routing Algorithm Stress Testing

n	Time (s)
10	0.003
100	23.1
200	543.9

Table 2: Stress Testing Results

As shown, the algorithm is able to produce results within a number of minutes, if not faster, for all expected inputs of size n. However, a trial of size $n = 1000$ was run, but did not conclude a single trial over the course of 8 hours. This should not be a problem for this project, as that is far above the expected range of input values.

5.2.2 Non-Functional Testing Evaluation

Scalability
In testing scalability, one ESP8266 was initially set up in the sensor network. From there, more ESP8266 chips were added to the sensor network. As each chip made an initial connection to the Raspberry Pi, the amount of recognized sensor modules continued to increase.

Data Integrity
To test data integrity, test data was gathered and correctly packaged on the Raspberry Pi and forwarded to the database. Upon the database receiving the testing data, it was observed that the data points remained the same as those gathered by the Raspberry Pi. This observation shows that the system upholds our requirement for data integrity.

Availability
The *node-scheduler* library is used to initiate the daily route generation algorithm. Since the team used this frequently during development to test the algorithm performance, we have the confidence that the library will work in production. Likewise, once the routine is run, the event is consistently resulting in favorable output after running the required processes and algorithms.

Deployment
In order to test deployment, several ESP8266 chips were programmed to request connection with the Raspberry Pi to ensure that they could all successfully create connections. Success was measured by an ESP8266 chip's ability to request a connection, successfully transmit and receive data to and from the Raspberry Pi, and then close the connection. The process started with one ESP8266 chip and later more were added. In all cases, each ESP8266 chip was able to successfully connect with the Raspberry Pi.

Usability
This system has been utilized multiple times throughout the semester. In all cases, data is collected by the sensors nearly instantaneously. In every test in pushing new data to the database, the system consistently takes significantly less than a minute to complete a transaction. This proves that our system is more usable than manually counting inventory.

Resilience
A sensor module has been used by the team throughout the past three months as part of the development of this project. At minimum, this reflects the normal usage a sensor module could

experience in a real stockroom. Upon the completion of this project, the sensor device was still operating as expected meaning that the sensor module design is proven to be resilient.

6 Project and Risk Management

6.1 Task Decomposition & Roles and Responsibilities

Throughout the project, tasks were divided between three categories: hardware, the sensor network, and software systems. Hardware was divided between the weight sensor and sonar sensor, while the software systems were divided into front-end and back-end systems. Hardware was handled by the team's two electrical engineering students, the sensor network was handled by one of the computer engineering students, and the software system was handled by the team's two software engineering majors and remaining computer engineering major. This approach allowed each of the teammates to work at their optimal level of comfort allowing for the fastest progress to be made. Oftentimes, team members from each division would meet together for product integration in order to ensure that all components of the project worked together as a unit.

Furthermore, non-technical tasks were divided between team members based on a set of roles determined at the beginning of the project. Non-technical tasks include handling documentation, ordering parts, and managing team members. All technical and non-technical roles were divided by team members as enumerated in section 6.1.2.

6.1.2 Task Decomposition by Team Member

- David Bis - Backend Developer, Meeting Facilitator
- Noah Bix - Hardware Architect, Documentation Manager
- Ben Gruman - Hardware Architect, Resource Acquisition
- Sam Guenette - Backend Developer, Public Relations
- Adam Hauge - Computer Network Architect, Report Manager
- Hanna Moser - Frontend Developer, Meeting Scribe

6.2 Project Schedule

This project followed a series of cycles of brainstorming, research, prototyping, testing, refining, and demonstrating. Figure 7 details the team's design process, inspired by the Agile development process.

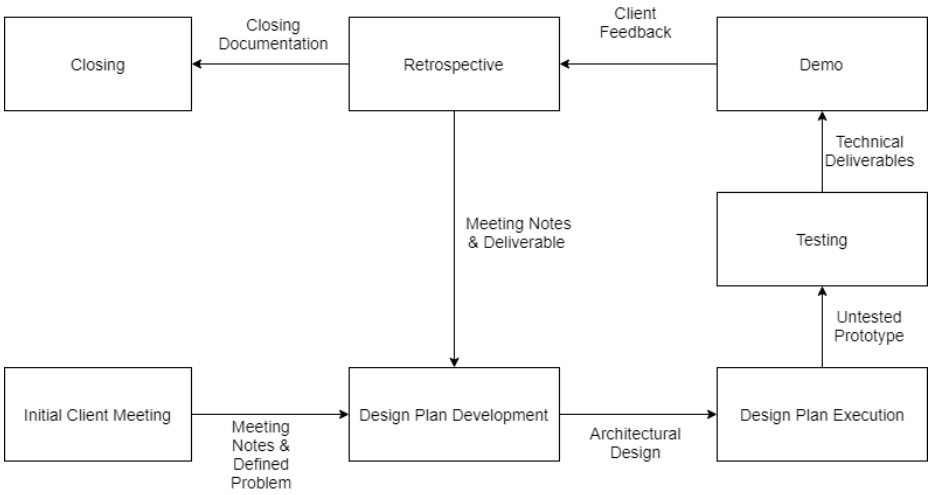


Figure 8: Process Diagram

This process entails a series of sprints where at the end of each sprint, a prototype or technical deliverable was expected to be completed. The process begins with an initial meeting with the client to define the project requirements, after which the first sprint commences. Each sprint follows a similar process, beginning with creating a design plan for the projected deliverable. Once the design architecture is determined, the plan is executed and then tested to ensure that it satisfies all of the necessary requirements. Once the deliverable is complete, it may be demonstrated to the client to gather feedback on its progress, which then guides and focuses future development. A retrospective meeting is then held by the team to reflect on previous work and to encourage continuous improvement upon the project architecture and design process. Once the project is complete, the team compiles the documentation to ensure that no technical knowledge regarding design decisions is lost.

The project has been broken into four different phases, where each phase operates as an Agile sprint. The following Gantt charts (Figures 8a-d) were used throughout the project to keep the team on track with consistent progress. These Gantt charts are broken down into the four separate phases of the project spanning the entire 2018-2019 academic year: two initial prototype phases, Minimal Viable Product, and Final Product.

	Task Title	9/3/18	9/10/18	9/17/18	9/24/18	10/1/18	10/8/18	10/15/18	10/22/18
Prototype 1	Determine Team Values and Norms								
	Gather Requirements								
	Design Initial Solution Architecture								
	Implement Initial Solution Architecture								
	Complete Prototype								
	Testing								
	Documentation								

Table 3a: Gantt Chart for Prototype 1 Phase

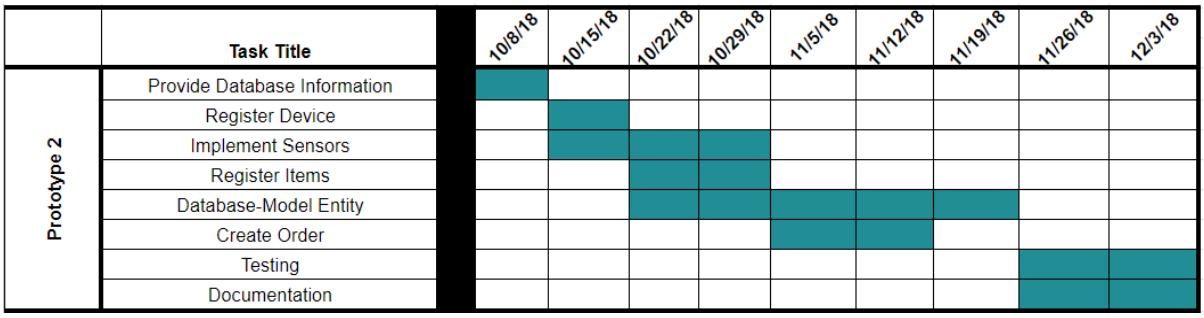


Table 3b: Gantt Chart for Prototype 2 Phase

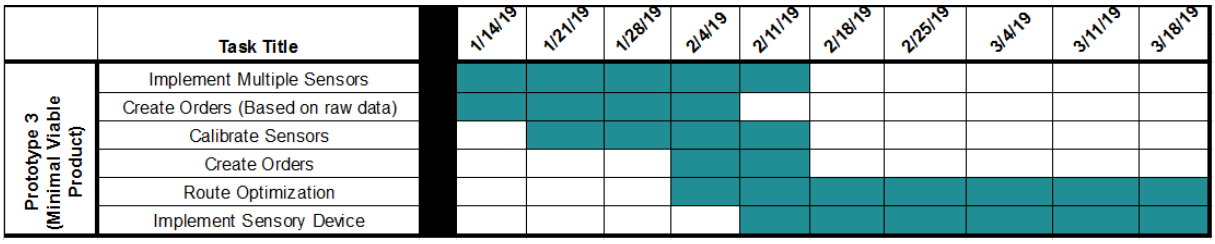


Table 3c: Gantt Chart for Prototype 3 (Minimal Viable Product) Phase

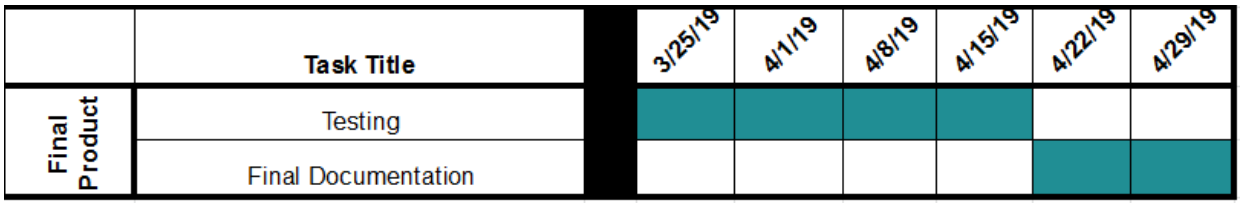


Table 3d: Gantt Chart for Final Product Phase

6.3 Risks and Mitigation

There are a number of risks involved with the project. Below is an enumerated list of identified risks and its accompanied mitigation plan.

Risk 1: Monitoring Device Goes Down
Information: The monitoring device stops communicating to the master Raspberry Pi.
Mitigation Plan: The front-end alerts the user when the back-end detects that it has not received sufficient data from any sensor in the network. By doing so the Crafty employee is notified when a sensor module needs to be fixed or replaced.

Risk 2: Sensor Degradation
Information: Sensors do not stay calibrated forever and so over time they become less accurate. Sensors may also start to degrade or break down. Both of these issues can lead to inaccurate reading of data and thus inability to automatically reorder products with precision.
Mitigation Plan: In order to consistently track data regarding the inventory of products, sensors need to be calibrated on a regular basis, and there needs to be catches to track whether the sensor itself is degrading or is broken.

Risk 3: Routing Inaccuracy Due to Traffic Abnormality
Information: If an accident or bad weather needs to be avoided, then inaccuracy in the routing may instead lead the delivery vehicle into a terrible backup of traffic that may cause them to be unable to make it to all of the delivery locations that they were supposed to reach that day. These events are referred to as traffic abnormalities.
Mitigation Plan: The algorithm considers traffic abnormalities and can reroute trucks when they occur. Trucks are not be able to handle other truck's routes because each truck is only supplied with the products for their originally assigned routes. As a result, the algorithm recalculates the optimal route for the remaining tour, and if the new time estimation is above the constraint, destinations are removed starting with the removal of least impact until the route satisfies the time constraints.

Risk 4: Sensor Ambiguity
Information: The load cell and the sonar sensor can provide contradictory readings of the inventory, causing ambiguity of the current inventory levels.
Mitigation Plan: A "sanity check" is developed to check the sensor readings, verifying that the data is logical. For example, if the load cell reading goes down, we'd expect the sonar reading to go up, and vice versa. It is also acceptable if both remain static. If both sensors move in the same direction, we flag the data as erroneous and refuse to update the database. This way, we request the user to manually check the sensor before updating the database as to prevent incorrect readings.

6.4 Lessons Learned

This project has given the team significant experience in developing and managing a large-scale engineering project. While working on this project, we all learned how to work effectively with a group of engineers with different knowledge backgrounds. We all gained experience working with electrical engineers, computer engineers, and software engineers and learned how to use each team member's strengths in order to help the team make the most progress possible. We've also experienced first-hand the development process of a large-scale project while working with a client to get all of the requirements and conditions that the project must meet in order to be completed.

We learned a great deal on how implementation must be adjusted when trying to get certain APIs and libraries to work together. When deciding to use ReactJS as our front-end, we did not realize the axios library, which makes requests to the server, does not respond to common programming conventions with ReactJS as opposed to other frontend languages. Similar issues occurred when trying to implement a part of the Google Maps API. There were multiple libraries and methodologies to implement the Google Maps API. Since the web component backend is developed in NodeJS, the API was originally installed and implemented using conventions documented by the npm library. However, the npm library does not return information synchronously. This forced a reimplementation of the API through the GoogleMaps general library, which works, but forced a reprogramming of our routing component.

Furthermore, we learned a lot of new skills in implementing hardware-software integrated systems. For this project, we needed to incorporate a lot of hardware and software aspects and the experience of implementing a hardware device to work alongside a highly-technical software system such that data can be transmitted from a sensor module up to a web component was new to everyone on the team.

7 Conclusion

The conclusion section will reflect upon the current status of the solution and future goals for the team.

7.1 Closing Remarks

Crafty, LLC's current infrastructure is based on a middleman creating an unnecessary expense that is prone to human error. This causes warehouse truck routing to become severely inefficient and expensive from restocking at individual offices based on separate orders.
Our solution consists of a microcontroller device that automatically monitors the amount of items in an office pantry and places orders to the Crafty warehouse when the office has reached its minimum threshold value for certain pantry items. All current orders are then processed and then organized into optimized shipment routes for delivery.

We believe that the presented design will be able to assist Crafty and its clients to a more effective and efficient inventory management system. This product is capable of tracking the inventory of a pantry and using the inventory data from multiple pantries to determine an efficient route to resupply pantries with products that are at a low quantity. Through this solution, we have created an automated inventory management system that suits Crafty's needs.

7.2 Future Work

Upon the completion of this project, all portions of the solution will be handed over to Crafty, LLC. This includes code, sensor module schematics, and documentation. Crafty has expressed major interest in continuing to develop and implement our solution into their warehouses and stockrooms. Communication will be maintained with the client during this time so that they may continually update the team on their implementation progress. Our team looks forward to cooperating with Crafty further in order to implement this project into real stockrooms. When working in large-scale integration, more constraints can be considered to create a more holistic solution. For instance, there will be variance in the unit size of products and traffic variance will be dependent on the serviced area.

Another improvement to the application could be through customer order analytics. We are storing the history of orders that each client makes, but are not taking advantage of it. A possible feature that could be made with the data would be predicting when a customer needs to resupply a product, and preemptively order to further optimize the resupplying process.

References

Intellectual Reference

1. Impinj.com. (2018). Automated Inventory Management with RAIN RFID | Impinj. [online] Available at: https://www.impinj.com/solutions/healthcare/inventory-management/.
2. Route4me.com. (2018). Dynamic Route Optimization™ On-Demand Deliveries & Pickups. [online] Available at: https://www.route4me.com/platform/routing.
3. Barcodesinc.com. (2018). Inventory Control System - Automate Your Tracking. [online] Available at: https://www.barcodesinc.com/articles/inventory-control-system.htm.
4. Wind.cs.purdue.edu. (2018). [online] Available at: http://wind.cs.purdue.edu/doc/adhoc.html.
5. GeeksforGeeks. (2018). Computer Network | TCP 3-Way Handshake Process - GeeksforGeeks. [online] Available at: https://www.geeksforgeeks.org/computer-network-tcp-3-way-handshake-process/.
6. Interserver Tips. (2018). What is SYN Flood attack and how to prevent it? - Interserver Tips. [online] Available at: https://www.interserver.net/tips/kb/syn-flood-attack-prevent/.
7. Python, R. (2018). Socket Programming in Python (Guide) – Real Python. [online] Realpython.com. Available at: https://realpython.com/python-sockets/.
8. Raspberrypi.org. (2018). Networking Lessons | Raspberry Pi Learning Resources. [online] Available at: https://www.raspberrypi.org/learning/networking-lessons/rpi-static-ip-address/.
9. "Raspberry Pi Starter Kit Lesson 13: I2C 1602 LCD," *Raspberry Pi Starter Kit Lesson 13: I2C 1602 LCD « osoyoo.com*. [Online]. Available: http://osoyoo.com/2017/07/raspbery-pi3-drive-i2c-1602-lcd/.

Pricing Reference Websites:

1. Electronics, S. (2018). SEN-13329 SparkFun Electronics | Sensors, Transducers | DigiKey. [online] Digikey.com. Available at: https://www.digikey.com/product-detail/en/SEN-13329/1568-1852-ND/7393715/?itemSeq=272691527.
2. Electronics, S. (2018). SEN-13879 SparkFun Electronics | Sensors, Transducers | DigiKey. [online] Digikey.com. Available at: https://www.digikey.com/product-detail/en/SEN-13879/1568-1436-ND/6202732/?itemSeq=272691544.
3. Industries, A. (2018). ADS1015 12-Bit ADC - 4 Channel with Programmable Gain Amplifier. [online] Adafruit.com. Available at: https://www.adafruit.com/product/1083.
4. Industries, A. (2018) HC-SR04 Ultrasonic Sonar Distance Sensor. [online] Adafruit.com. Available at: https://www.adafruit.com/product/3942.
5. Sparkfun.com. (2018). Raspberry Pi 3 B+ - DEV-14643 - SparkFun Electronics. [online] Available at: https://www.sparkfun.com/products/14643.
6. Sparkfun.com. (2018). WiFi Module - ESP8266 - SparkFun Electronics. [online] Available at: https://www.sparkfun.com/products/13678.

Appendix A: Datasheets

Raspberry Pi Model zero-w Datasheet

- Industries, A. (2018). ADS1015 12-Bit ADC - 4 Channel with Programmable Gain Amplifier. [online] Adafruit.com. Available at: https://www.adafruit.com/product/1083.

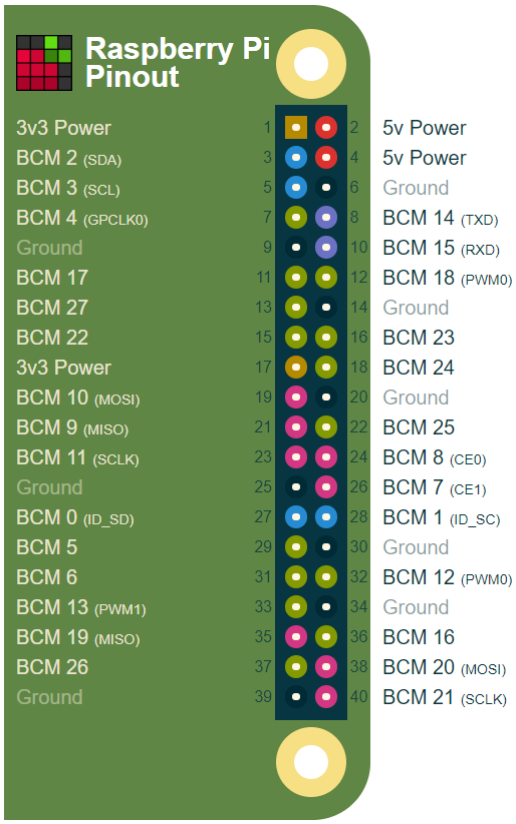


Figure 10: Raspberry Pi Model Zero-W Pinout

Pinout.xyz. (2018). Raspberry Pi GPIO Pinout. [online] Available at: https://pinout.xyz/.

Database Schema

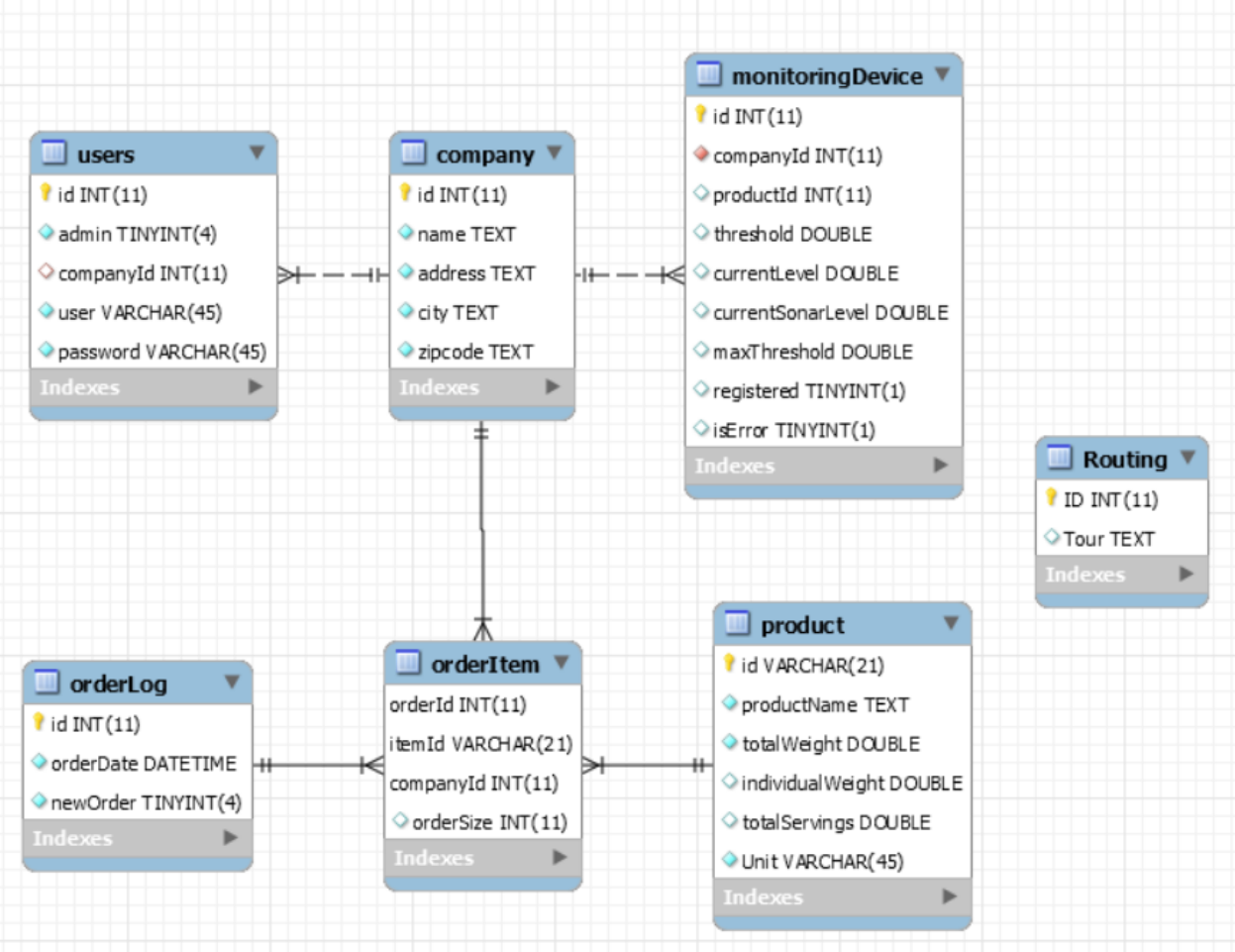


Figure 11: Database Schema

TAL220 Load Cell Datasheet:

Cdn.sparkfun.com. (2018). [online] Available at: https://cdn.sparkfun.com/datasheets/Sensors/ForceFlex/TAL220M4M5Update.pdf [Accessed 26 Nov. 2018].

Appendix B: Screen Sketches

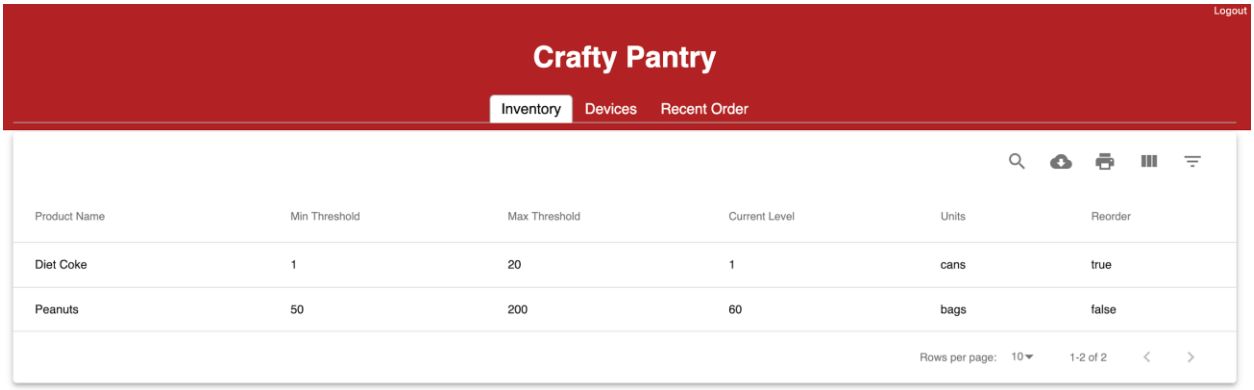


Figure 12a: Inventory Page (client)

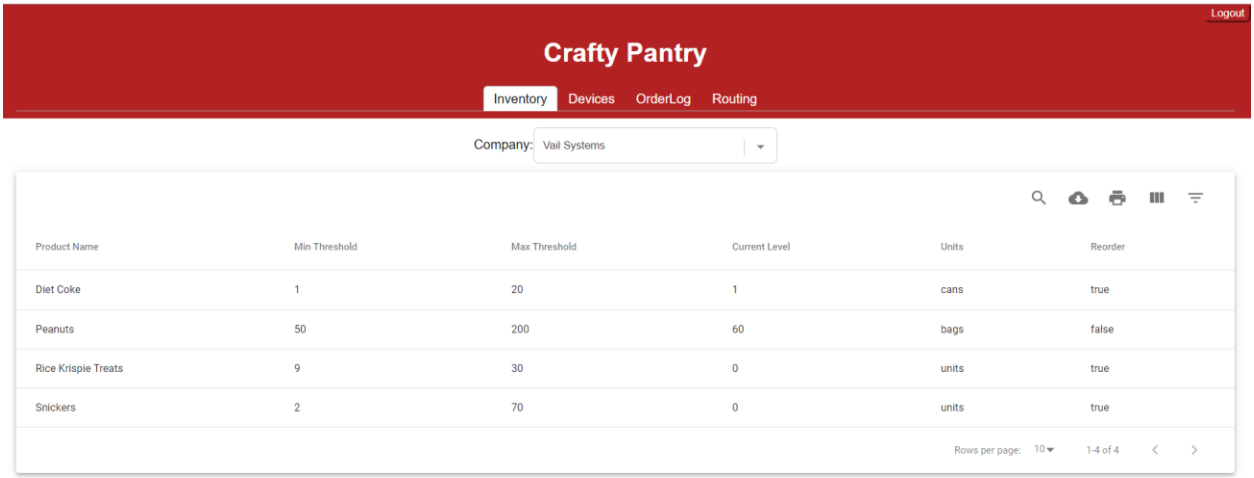


Figure 12b: Inventory Page (admin)

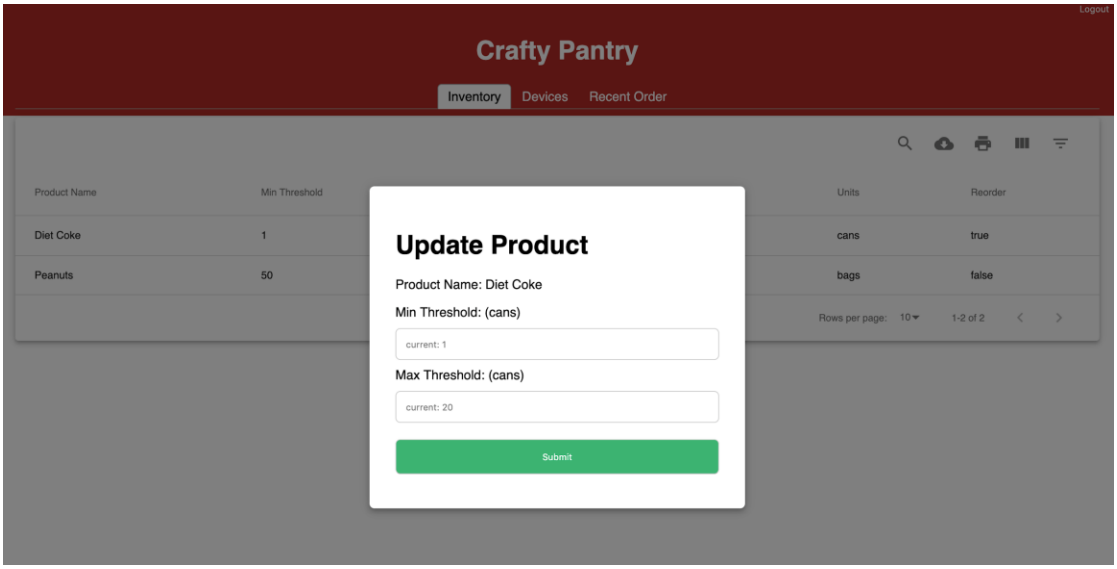


Figure 12c: Update Product Min/Max Threshold Modal (admin & client)

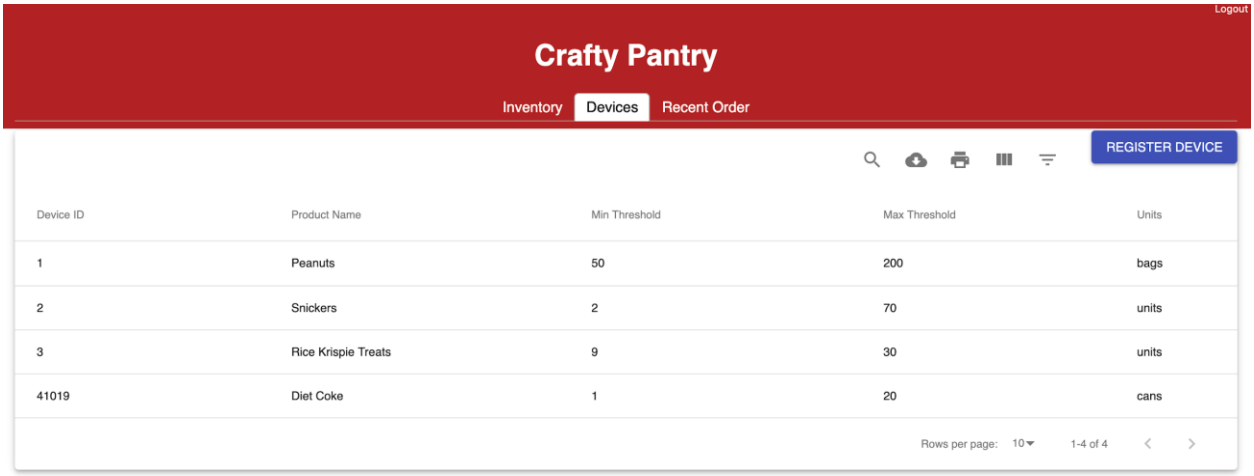


Figure 12d: Devices Page (client)

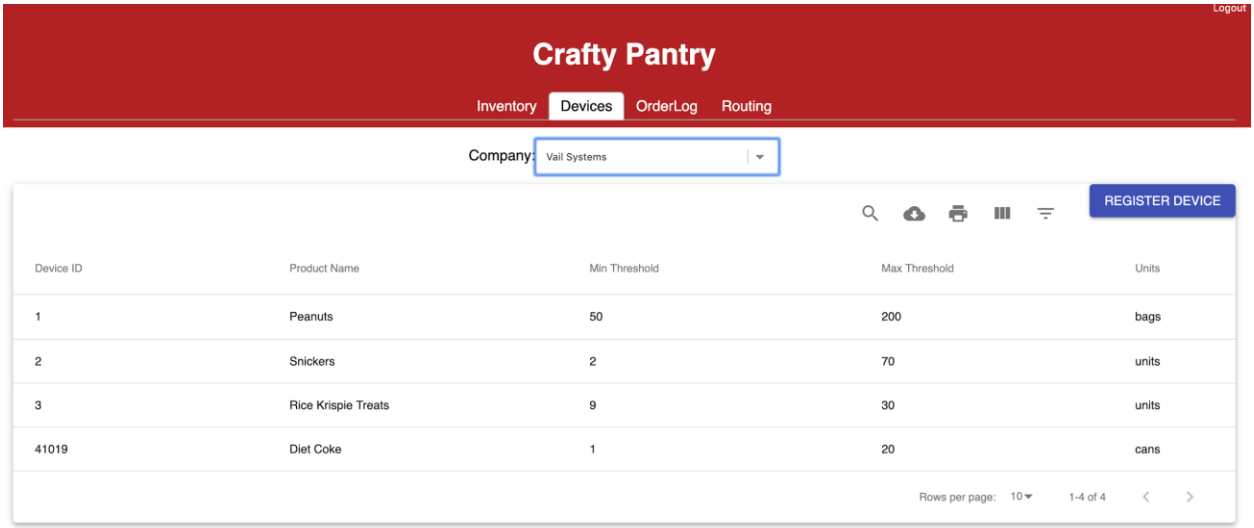


Figure 12e: Devices Page (admin)

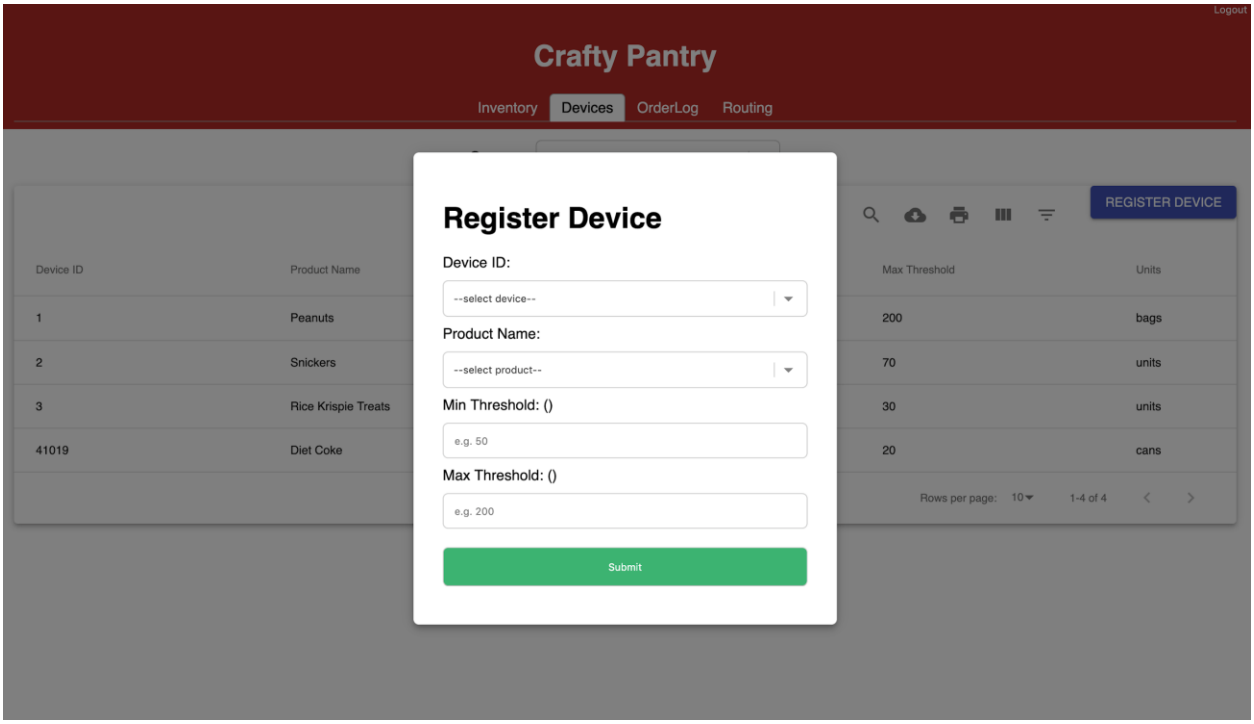


Figure 12f: Device Registration Modal (admin & client)

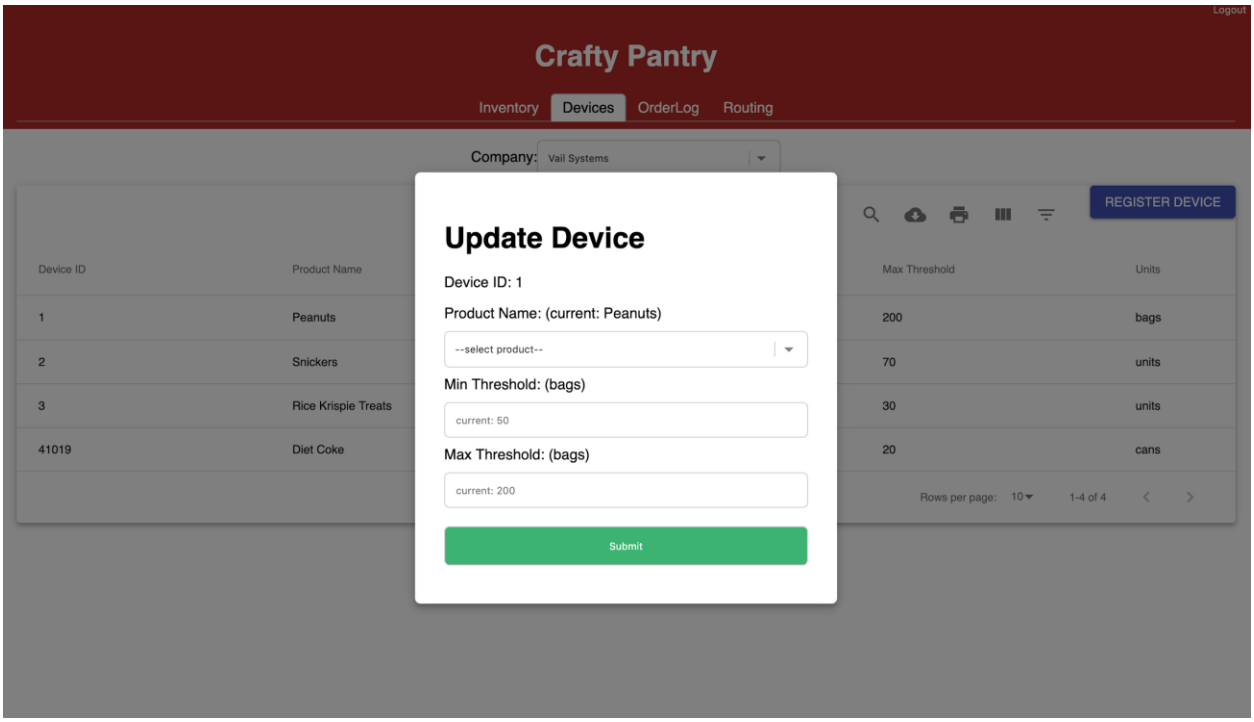


Figure 12g: Update Device Modal (admin & client)

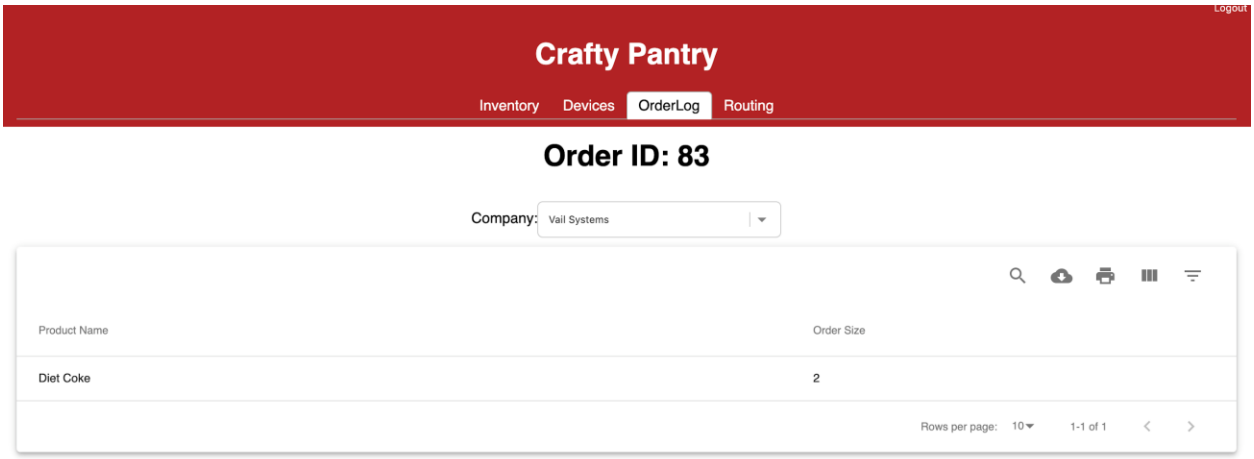


Figure 12h: OrderLog Page (admin)

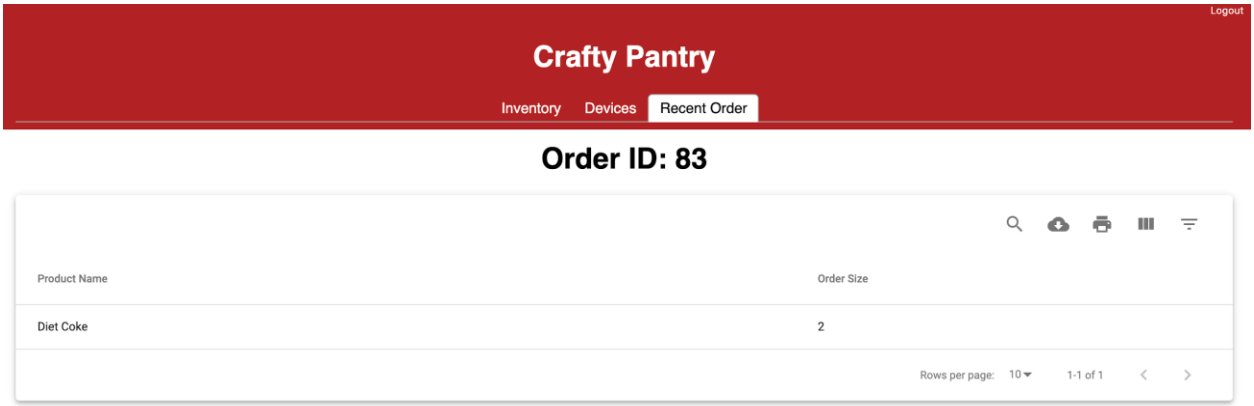


Figure 12i: Recent Order Page (client)

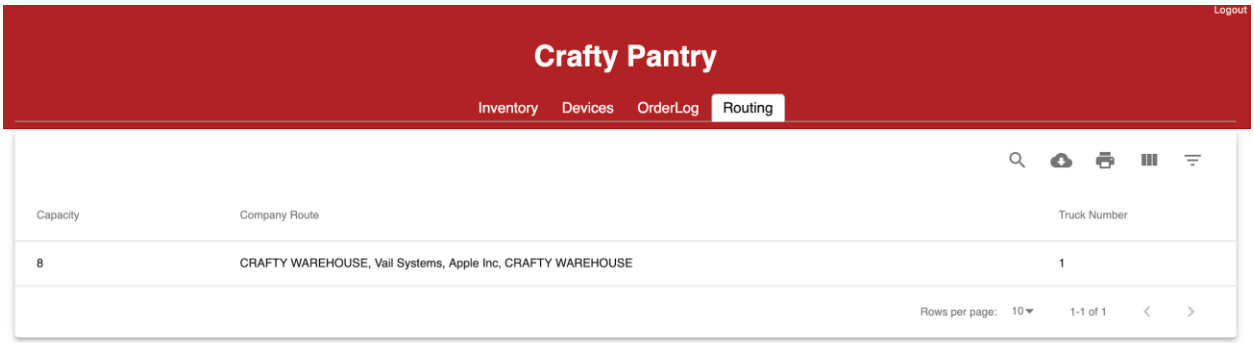


Figure 12j: Routing Page (admin)

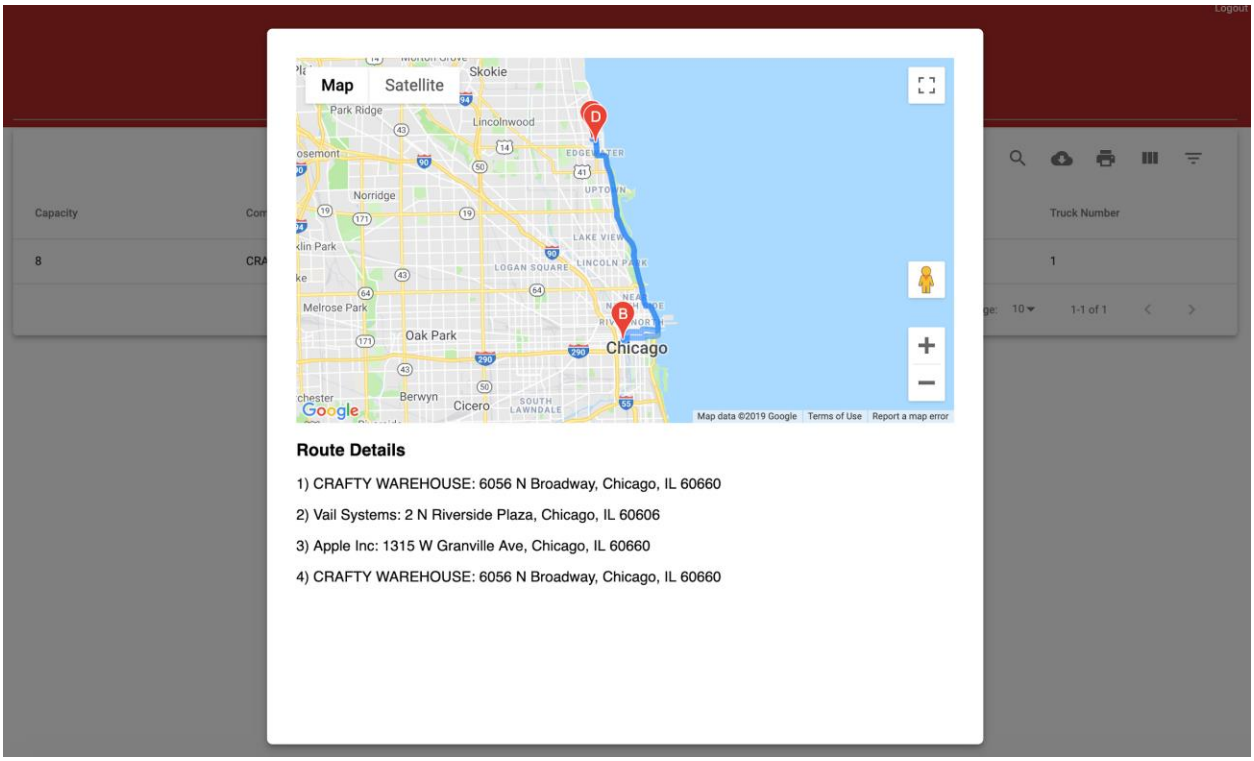


Figure 12k: Google Maps Modal (admin)