Boston University
Electrical & Computer Engineering
EC463 Senior Design Project

First Semester Report

Blitz

Submitted to

Michael Hirsch

by

Team 7
Blitz

Team Members

Cooper Salmon cosalmon@bu.edu
John Gillig jgillig@bu.edu
Kyle Martin ksmartin@bu.edu
Lindsey Volk volkl@bu.edu
Mariano Pache mnpache@bu.edu

Submitted: 12/10/2019

Table of Contents

Executive Summary

EC463 Senior Design Project
Team 7 Blitz Store Mapper

Shoppers are often unaware of the locations of their shopping list items. Consequently, customers often wander through grocery store aisles in search of their needed items. This is a waste of time and is extremely inefficient.

Moreover, store employees must be conscious of item inventory and must restock items throughout the day. Store employees are also required to keep track of sold items. Performing these tasks manually is tedious, inefficient, and could easily lead to erroneous data.

Our team is determined to make the shopping experience more efficient and enjoyable for both customers and store employees. We plan to create a shelving unit that displays item name and price, tracks item inventory with load sensors, and reports item inventory to a database that is accessible by both customers and store employees. We will develop a web application so that stores can track item inventory and can identify which items are most popular. We will also design a mobile application so that customers can create shopping lists based on their selected grocery store's inventory. The mobile application will provide the customer's shortest path through the grocery store based on their shopping list.

1. Introduction

Blitz Store Mapper is designed to simplify managing and shopping at a grocery store. There are two parties that can greatly benefit from using Blitz: store managers and customers. Store management does not have an easy way of tracking inventory for items on all of their store's shelves; either employees walk through the store or the inventory is updated when items are checked out. Through these methods, inventory is not updated in real-time. Another issue for management is when items change location or there are promotions: both require employees to manually change the item stickers on the front of each shelf (the pieces of paper that show item name, price, promotional info, etc.). Shoppers can also waste a lot of time walking through stores, looking for many items. Overall, we think that shopping can be more efficient in terms of time and effort.

The purpose of Blitz is to create a total shopping platform for management and consumers. Both parties can greatly benefit from automated processes to aid in everyday tasks. For management, tracking inventory is extremely important and with Blitz, inventory is updated when an item is taken off or put on a shelf. Management can know immediately, before too much time has passed waiting for customers to check out, if an item is low or out of stock. Also, when items are moved or there are promotions, employees will not have to change the item stickers on the dedicated shelves themselves. To help the customer be more efficient, they can use our mobile application to create a shopping list and receive a populated, shortest-path route through their selected grocery store. Instead of wasting time, customers can know exactly where to go in the store to find exactly what they need in the shortest time possible.

By employing simple devices, Blitz is able to solve all of the issues management and consumers face. It is able to track inventory, give real-time alerts and updates, and generate optimized routes for shoppers. Through the use of four loads and an ESP32 module, each shelf can keep track of inventory by telling when an item of known weight is taken off or put back on the shelf. The ESP device will process changes in weight data to determine any new products that have been. Updates to shelf inventory will be communicated to a central processing unit. The CPU will utilize a Raspberry Pi to process shelf inventory updates coming from around the store. The Raspberry Pi will update the store's SQL database to reflect the new shelf inventories; this way, managers can see immediate changes in inventory and then act upon the information. Since every shelf has an ESP device, we can easily tap into a store's database to find the location of items (which shelves they are on) to use the information to make the specific shopping routes for customers. Each shelf will have its own LED display so that the item name, pricing, and discounts can be changed remotely through the store's web application.

Our system will be designed to be scalable so that the user will be able to use the application at any Blitz-enabled grocery store. The mobile application will be aesthetically appealing, displaying product visuals and a comprehensive store map. We want to create a customizable system that benefits the shopper and the store. We want the

shopper to be able to customize their shopping experience in the mobile application, and we want the store to be able to customize the shelving unit based on their specific needs and space requirements. We want this system to allow for efficient and enjoyable shopping and simplified store management.

2. Concept Development

The problems solved by *Blitz* are twofold. Grocery shoppers' trips to the store often take longer than necessary because shoppers are unsure of the in-store locations of one or more items on their shopping list. At the moment, most shoppers' desire to visually evaluate groceries in-person outweighs a simultaneous desire for the greater convenience offered by grocery delivery platforms. Brick-and-mortar grocery store are in clear need of a systemic improvement to their convenience so that ever-busier shoppers can complete their trip in the shortest period of time.

Grocery store managers and employees face a similar dilemma of inefficiency. The shelf restocking process relies on employees walking around the store and visually identifying products in need of restocking, an inefficient process which consumes far more employee energy than necessary. Grocery stores and chains would be able to derive significant savings from a system which would increase the efficiency of the restocking process.

When considering the exact requirements of a system which would satisfy the needs of customers, the following list of pruned objectives was considered:

0. Shopping optimization system for grocery shoppers
- **1. System should reduce duration of shopping trips**
  - 1.1. Provides fastest path around a store
  - 1.2. Allows users to upload their shopping list and view item's stock status ahead of time
- **2. System should be marketable**
  - 2.1. System should be easily accessible by users
    - 2.1.1. System should be available as a mobile application
      - 2.1.1.1. Available for Apple devices
      - 2.1.1.2. Available for Android devices
    - 2.1.2. System should be free to users

When considering the exact requirements of a system which would satisfy the needs of store managers, the following list of pruned objectives was considered:

0. Inventory management system for store managers and employees
- **3. System should reduce the time needed to restock shelves**
  - 3.1. Notifies managers which shelves are in need of restocking
    - 3.1.1. Provides number of items needed to restock
    - 3.1.2. Provides in-store location of shelves in need of restocking
      - 3.1.2.1. Provides shortest travel route between shelves in need of restocking
  - 3.2. System should update shelf item count on a real-time basis
  - 3.3. System should allow for the switching of a single grocery item's location on a shelf
- **4. System should be marketable**

4.1. System should be easily accessible by users
  4.1.1. System should be available as a mobile application
    4.1.1.1. Available for Apple devices
    4.1.1.2. Available for Android devices
  4.1.2. System should be available as a desktop application
4.2. System should be affordable relative to existing store fixed costs

In considering the best implementation of the *Blitz* system to ensure the satisfaction of the above customer- and manager-facing solutions, a number of solutions were considered, with the conceptual approach taken to developing a solution focused around ease of implementation to grocery stores, the single most important factor a store would consider when determining whether to install the *Blitz* system.

One concept considered was a visual system which would utilize artificial intelligence to recognize the identity of products on the shelf and upload that data to the cloud. This system could have been operated by means of a camera-holding store employee traversing the store, or a camera-equipped autonomous vehicle driving around the store. This concept was abandoned due to the fact that the portable camera would have been unable to detect the depth of the stocked items on the shelf, a necessary ability in order to determine which items will be out-of-stock with sufficient notice. Another concept considered involved mounting cameras on each shelf individually, but this concept was abandoned because cameras are cost-prohibitive compared to load cells. In the end, a system centered around a weight-sensitive shelf was deemed easiest for grocery stores to implement. The weight-sensitive shelf would have four weight sensors, one at each corner, such that the shelf could detect both scalar changes in weight and changes in the shelf's center of mass.

The following function-means trees outline the details of the chosen *Blitz* solution:

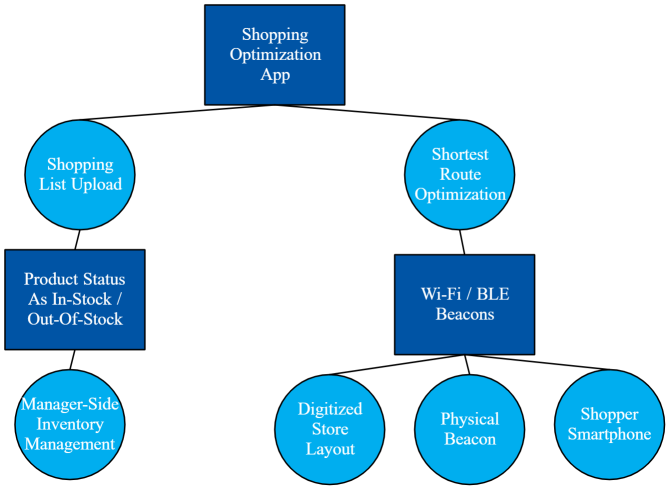

Figure 1. Function-means tree for customer-facing solution

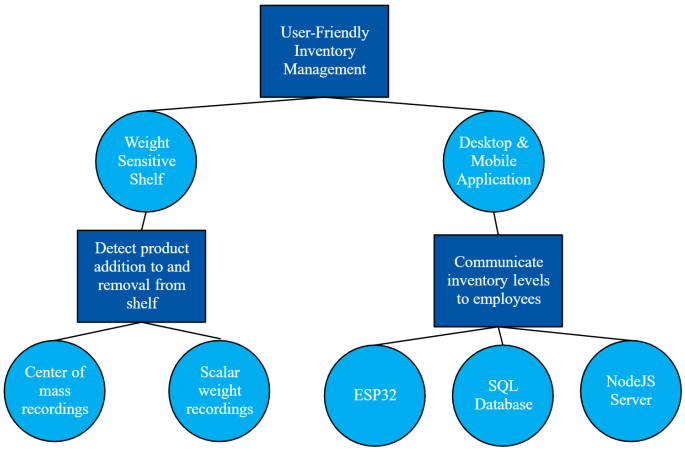

Figure 2. Function-means tree for manager-facing solution

3. System Description

The smart shelving unit will allow for easy and efficient inventory management. The smart shelving unit will lay atop the current grocery store shelves. This allows for easy installation of our system. Each smart shelving unit utilizes 4 load cells (1 load cell at each corner of the shelving surface), a HX711 module, and an ESP32 module. The 4 load cells detect item weight and interface with the HX711 module. The HX711 module is a precision analog to digital converter that converters a detected analog voltage to a digital value. The HX711 module interfaces with the ESP32. Diagrams and pinouts for the load cells, HX711, and ESP32 can be found in Appendix 3, Figures 1, 2, 3, and 4. The ESP32 is loaded with calibration and tested sketches written in the Arduino IDE. The ESP32 communicates the detected weights to the store CPU via a Linksys router.

We will use UDP protocol to facilitate communication between shelves in the store and the CPU. This allows for multiple ESP devices to send packets to the same port of the Pi server. These packets contain the current weight on a particular shelf. Upon reception of UDP messages, the CPU will compare the incoming shelf weight with the previous shelf weight stored in the "all_shelves" database. We compare the change in weight to the weight of each item known to the shelf to decide whether or not a known item was added (see section 4 for algorithm draft). The server then performs any required update statements to the database.

Our CPU uses a Raspberry Pi to host a node.js server and a mySQL database that stores two types of tables: one that tracks the total weight on each shelf (all_shelves) and one that tracks the inventory of each shelf. There will only be one table in the database that tracks the total weight on each shelf and its rows are formatted as follows:

shelf_id VARCHAR(255) - IP address of ESP32 on shelf
weight FLOAT - total weight on shelf (lbs.)

There will be as many tables to track shelf inventory as there are shelves in the store. The table name of any given shelf will be the IP address of the ESP32 device on the shelf. The format of these table rows is as follows:

id VARCHAR(255) - name of product
weight FLOAT - weight of product (lbs.)
count INT - count of product

To facilitate communication between the CPU and the application, we decided to use three REST API endpoints, one to add or remove a product type to or from a shelf (POST), one to get all shelf ids (GET), and one to get the inventory of a particular shelf (GET). The purpose of the endpoints is to provide a simple way for the application to obtain all of the information it needs from a store. The requests will use the following payloads and trigger the corresponding mySQL statements:

- POST new product type: {shelf_id, product_id, product_weight}
  - "INSERT INTO shelf_id (id, weight, count) VALUES (product_id, product_weight, 0)”
- GET shelf ids: no payload
  - "SELECT * FROM all_shelves”
- GET shelf inventory: {shelf_id}
  - "SELECT * FROM shelf_id”

The Blitz app, written in Angularjs, works as a single page application that forwards all API calls to the node.js backend. The node.js backend is used to communicate with the database, to execute shortest-path analysis, and to perform transaction handling. The mySQL database for the application will store users and their favorite stores and shopping lists as well as the maps of each store. Store maps will be stored as a list of grid coordinates, where each coordinate represents the location of a shelf. All application databases will be hosted in AWS.

The Angularjs app contains 5 main views: Home, Stores, Lists, Settings, and Admin. The Admin view only exists when the user is stored as a store manager or employee and allows the user to view store inventory and to create a store map. To authenticate users, the application uses Okta single-sign-on.

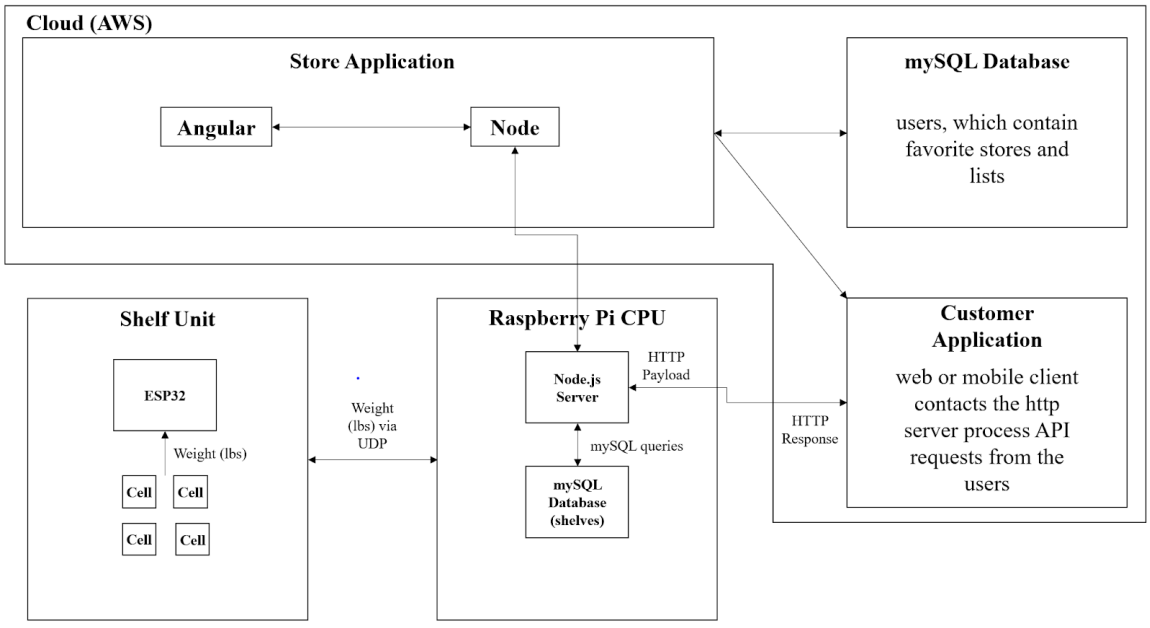

Figure 1. System Block Diagram

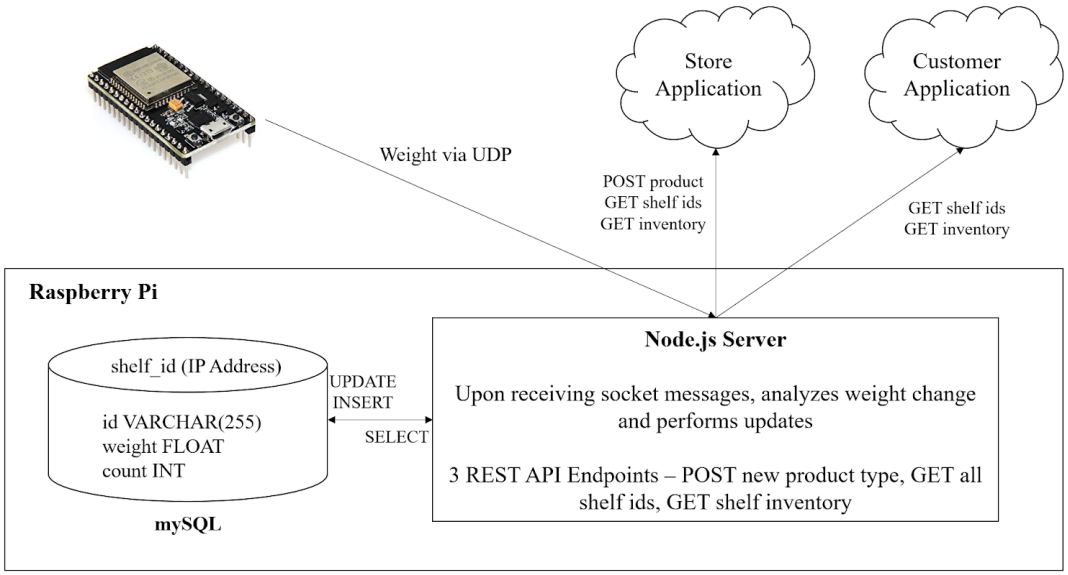

Figure 2. CPU Block Diagram

4. First Semester Progress

This semester, our team has made significant progress with the hardware and software aspects of our project. For hardware, we have created a smart shelving unit that is functional and reliable. To do so, we started by flushing out our weight sensing options. We decided to utilize 4 50kg load cells, which can support a weight of 200kg when used together. Once these load cells arrived, we started our testing. We first wired 1 load cell to the HX711 module, which is a precision analog to digital converter. We wired the HX711 to the ESP32. We wired these modules according to Appendix 3, Figures 1, 2, 3, and 4.

We then ran a calibration sketch in the Arduino IDE using a HX711 library provided by the Arduino IDE. We started by testing 1 load cell. To calibrate the load cell, we added a 1lb. weight to the load cell and determined a calibration factor that yielded the correct weight. Then, we ran a test sketch in the Arduino IDE that allowed us to test the load cell with multiple known weights. Once the load cell was functioning correctly, we started testing with all 4 load cells.

Before testing with all 4 load cells, we 3D printed load cell mounts that are pictured in Appendix 3, Figure 5. These mounts assured that the load cells were static and sturdy during testing. To test with all 4 load cells, we determined the calibration factor that yielded the correct weight when the 1lb. weight was placed upon the 4 load cells and shelving surface. This factor was slightly different than the calibration factor that we found for 1 load cell.

Once the 4 load cells were yielding accurate results with the shelf placed atop of them, we started testing sequences of item placement and removal to assure that our shelving unit could recognize inventory changes based on weight detection. Then, we started to communicate the weight detected by the load cells to a node.js server via UDP. This server ran an algorithm to determine the item that was added to or removed from the shelf. For the first prototype testing, we demonstrated the functionality of our smart shelving unit, the functionality of the communication between our smart shelving unit and the node.js server, and the functionality of our algorithm by displaying the correct inventory in the mySQL database at all times. The sequences of item placement and removal that were tested can be found in Appendix 3, Figures 7, 8, and 9. We were 100% successful in our first prototype testing.

Following the first prototype testing, we continued to develop our smart shelving unit and software applications. For the smart shelving unit, we first screwed the load cell mounts to the bottom shelving surface in the corners of the bottom shelving surface. Then, we soldered and heat shrunk the load cell wires. Then, we labeled the load cells and the load cell wires. The bottom shelving surface is displayed in Appendix 3, Figure 6.

We have implemented some, but not all, of the CPU design features. So far, we have established a socket connection between the node.js server and one ESP32. Throughout testing, the node.js server has been locally hosted on a laptop. We have also

implemented a first draft of the algorithm that updates shelf inventory. We used this algorithm in first prototype testing. It executes upon receiving a UDP message from the ESP32:

1. Calculate the absolute value of the difference between old shelf weight and new shelf weight upon reception of socket message (assign to delta)
2. Store the sign of delta
3. Calculate the absolute value of the difference between the weight of each item on the shelf and delta and assign the result to the item's error
4. If the minimum error is greater than 0.10lbs, then mark the item prediction as unknown
5. If the minimum error is greater than |0-delta|, then assume no item was added
6. If steps 4 and 5 are both false, then predict that the item with the minimum error was added or removed, depending on the sign of delta.
7. Update the count of the predicted item in the database unless the item was deemed to be removed and the count of that item is already at zero (NOTE: this execution condition is meant as a backup to ensure that no item count is ever negative).

In this implementation, we stored the weight on the shelf as a variable within the server. However, this task will require an additional database when we have more than one shelf on the network.

For the application, we developed the frontend in Angularjs, developed the backend using node.js and used a mySQL database for storage. We have fully implemented Okta authentication, written and tested data getters and setters from the Database, sent data to AWS, and built our mySQL database schema, where the users table links to a table of stores and a table of shopping lists. We have started, but have not completed, the page for store selection and the page for building shopping lists. They are present in the app but are not functional.

5. Technical Plan

Task 1. Develop RPi server with one ESP32 device

- We will develop simulation code on the ESP32 that writes weight data to the RPi server via UDP. We will show that the server can correlate the IP of the shelf to the shelf's table in the database and that the inventory updates according to the simulation. **Task lead:** Kyle Martin

Task 2. Debug RPi server to accept readings from multiple ESP32 modules

- We will flash the simulation code developed in task 1 to three ESP32 devices. We will show that the RPi server can receive UDP messages from three different devices, differentiating them by their IP addresses. We will then show that the server can maintain an accurate inventory for each in the mySQL database. **Task lead:** Kyle Martin

Task 3. Debug RPi server to accept HTTP requests from an HTML file

- Write an HTML file that simulates HTTP requests from the application. Test each endpoint - GET shelf ids, POST new product type, GET shelf inventory; and display the results in browser. **Task lead:** Kyle Martin

Task 4. Develop store-side app: view inventory, create store map

- Continue Full Stack development in Angularjs and node.js with a MySQL Database. **Task Lead:** Mariano Pache

Task 5. Test and verify RPi server with actual shelf units

- We will repeat Tasks 1 and 2 with actual shelf units and actual products, demonstrating the true function of the CPU-shelf system. **Task leads:** Kyle Martin and Lindsey Volk

Task 6. Test and verify REST endpoints on RPi with application

- We will integrate the CPU with the application, showing that the application can obtain all shelf ids, post new product types to a shelf table in the database, and obtain the current inventory of a shelf. **Task leads:** Kyle Martin and Mariano Pache

Task 7. Design CPU

- We will design the CPU container in PTC Creo. This container will be no larger than 6"x6"x3" and will feature a removable cover and a Type A plug. The container will be mountable to a wall. **Task lead:** Kyle Martin

Task 8. Manufacture CPU

- We will manufacture the CPU container in EPIC and insert the RPi. **Task lead:** Kye Martin

Task 9. Complete smart shelving unit construction

- We will complete the smart shelving unit by adding wooden borders to the current shelving unit that is comprised of a shelving base and shelving surface. **Task lead:** Lindsey Volk

Task 10. Stacked shelving unit construction

- We will purchase a stacked shelving unit and will implement three smart shelving units on the stacked shelving unit. We will construct two more smart shelving units in the same manner that we constructed the first smart shelving unit. **Task lead:** Lindsey Volk

Task 11. Implement an LCD display

- We plan to purchase LCD displays and implement them on each of our smart shelving units. The LCD displays should communicate item name and price. **Task lead:** Lindsey Volk

Task 12. Testing between three smart shelving units on the stacked shelving unit

- We will assure that three smart shelving units can communicate with a database synchronously. **Task lead:** Lindsey Volk and Kyle Martin

Task 13: Implement Algorithm for shortest path

- **Task lead:** Cooper Salmon

Task 14: Implement algorithm for load cells

- We plan to implement an algorithm that can detect from where an item was placed or removed from the shelf. **Task lead:** John Gillig

Task 15: Full system testing

- As a team, we will complete full system testing to assure our system's functionality. **Task:** Team

6. Budget Estimate

Item	Description	Cost
1	Shelving Surface x 3	$45.00
2	Load Cell (Pack of 4) x 3	$27.00
3	ESP32 x 3	$19.50
4	LCD Display x 3	$27.00
5	Raspberry Pi	$35.00
6	Stacked Shelving Unit	$90.00
	Total Cost	$243.50

Fortunately, the most expensive component of this project will be the shelving unit display, which we will purchase to display to our audience on ECE Day that multiple smart shelving units can communicate item weight and inventory to a database.

We have purposely paid close attention to the prices of our items because, in order for our project to be scalable to grocery stores, it must be low in cost.

6. Attachments

6.1 Appendix 1 – Engineering Requirements

Requirement	Value, range, tolerance, units
1	Shelving unit should be no larger than 1m by 3.5m
2	Shelving unit should be adjustable; the minimum length should be no larger than 1m by 1.5m
3	Load cells located at each corner of the shelving unit
4	Load cells must be accurate to the nearest 0.025kg
5	Each shelving unit should be able to hold 150kg
6	Shelving unit must be primarily powered by an outlet but should last for at least 18 hours per battery charge if there is no outlet accessible
7	Shelving unit should be able to operate normally at 0 degrees celsius
8	Shelving unit should communicate item name and price through LCD display
9	Each smart shelving unit should contain an ESP32 module
10	The ESP32 sends the current weight in lbs through a UDP socket to a Raspberry Pi based CPU
11	The ESP32 only sends the current weight when it has stabilized within +/- 0.05 lbs
12	The Raspberry Pi stores a mySQL database for each shelf, where each database stores the id, count, and weight of each product
13	The shelf database only updates counts if the change in shelf weight matches the stored item weight within +/- 0.01 lbs
14	Employees should be able to update the products that each shelf contains from the application
15	The CPU should be powered by an outlet
16	The Pi server has REST API endpoints to get a list of shelves in the store, to post a product type to a shelf, and to get the inventory of a shelf
17	The application will run on node.js and Angularjs
18	The application will use Okta to authenticate users
19	Manager/employee accounts will have access to a live store inventory and a store map editor that allows them to place shelves and assign items
20	Customers will be able to store their favorite stores and shopping lists
21	Customers will be guided through the store based on Dijkstra's algorithm for shortest path

22	All mySQL databases will be hosted in AWS (users, shopping lists, maps)

6.2 Appendix 2 – Gantt Chart

MEMBER	TASKS	12/02-12/08	12/09-12/15	12/16-12/22	BREAK	01/20-01/26	01/27-02/02	02/03-02/09	02/10-02/16	02/17-03/22	03/23-03/29
Kyle	1. Develop RPi server with one ESP32 device										
Kyle	2. Debug RPi server to accept readings from multiple ESP32s										
Kyle	3. Debug RPi server to accept HTTP requests from an HTML file										
Nico	4. Develop store-side app: view inventory, create store map										
Kyle	5. Test and verify RPi server with actual shelf unit										
Kyle	6. Test and verify REST endpoints on RPi with application										
Kyle	7. Design CPU (RPi and some container)										
Kyle	8. Manufacture CPU										
Lindsey	9. Complete the implementation of the singular smart shelving unit										
Lindsey	10. Purchase a stacked shelving unit and implement three smart shelving unit				Major Milestone						
Lindsey	11. Implement LCD or LED display on the smart shelving units										
Lindsey	12. Assure communication between the three smart shelving units and a database								Major Milestone		
Cooper	13. Shortest path algorithm										
John	14. Algorithm for load cells										
All	15. Full system testing and debugging										End of project

6.3 Appendix 3

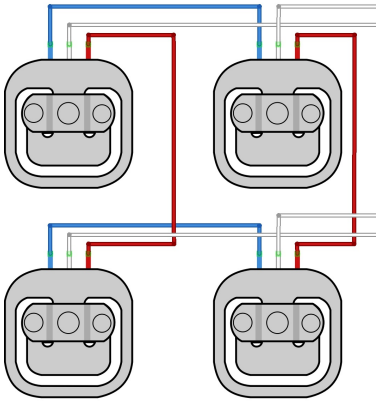

Figure 1: Load Cell Wiring Schematic

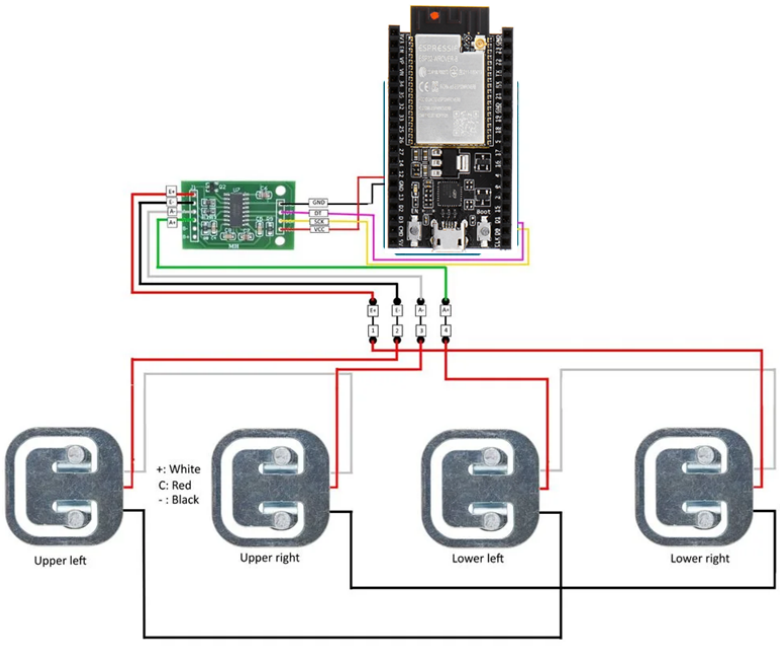

Figure 2: Load Cell, HX711, and ESP32 Wiring Schematic

Signal	Pin
VCC (from HX711)	3V
DT (from HX711)	Digital Pin 6
SCK (from HX711)	Digital Pin 5
GND (from HX711)	GND

Figure 3. ESP32 Pinout Table

Signal	Pin
Lower Right Load Cell Data	E+
Upper Left Load Cell Data	E-
Upper Right Load Cell Data	A-
Lower Left Load Cell Data	A+

Figure 4. HX711 Pinout Table

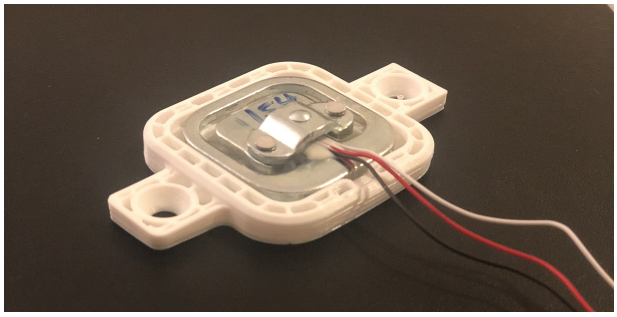

Figure 5. 3D Printed Load Cell Mount

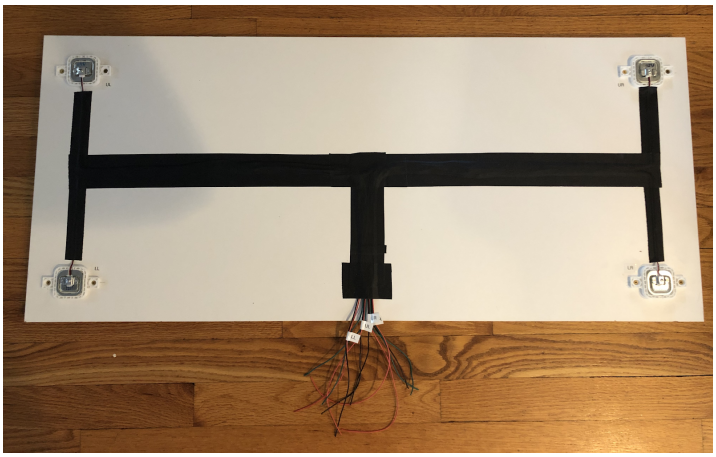

Figure 6. Bottom Shelving Surface With Mounted Load Cells and Organized and Labeled Wires

Item Name	On/Off Shelf	Expected Inventory of Apple Juice	Expected Inventory of Peanut Butter	Expected Inventory of Chips	Accurate Inventory (Yes/No)
Apple Juice	On	1	0	0	
Apple Juice	Off	0	0	0	
Peanut Butter 1	On	0	1	0	
Peanut Butter 2	On	0	2	0	
Peanut Butter 1	Off	0	1	0	
Peanut Butter 2	Off	0	0	0	
Chips	On	0	0	1	
Chips	Off	0	0	0	

Figure 7. First Prototype Testing Sequence 1

Item Name	On/Off Shelf	Expected Inventory of Apple Juice	Expected Inventory of Peanut Butter	Expected Inventory of Chips	Accurate Inventory (Yes/No)
Apple Juice	On	1	0	0	
Peanut Butter 1	On	1	1	0	
Peanut Butter 2	On	1	2	0	
Peanut Butter 1	Off	1	1	0	
Peanut Butter 2	Off	1	0	0	
Chips	On	1	0	1	
Apple Juice	Off	0	0	1	
Chips	Off	0	0	0	

Figure 8. First Prototype Testing Sequence 2

Item Name	On/Off Shelf	Expected Inventory of Apple Juice	Expected Inventory of Peanut Butter	Expected Inventory of Chips	Accurate Inventory (Yes/No)
All items	On	1	2	1	
All items	Off	0	0	0	

Figure 9. First Prototype Testing Sequence 3

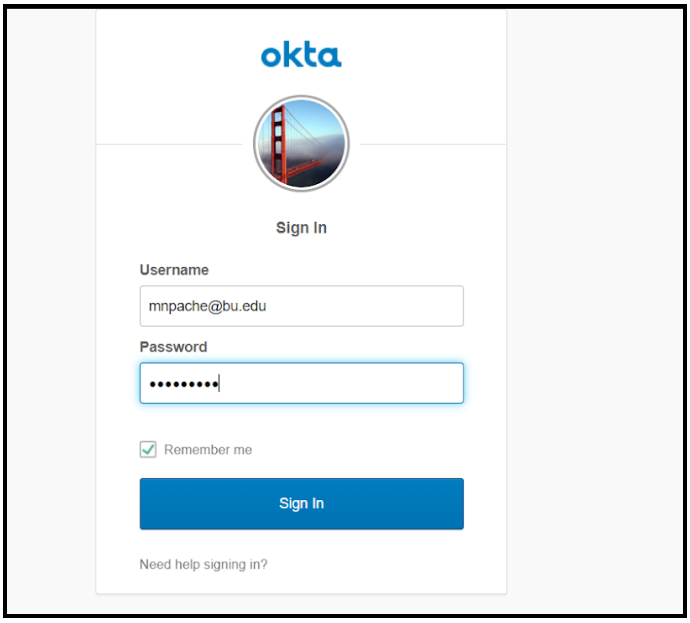

Figure 10. Authentication

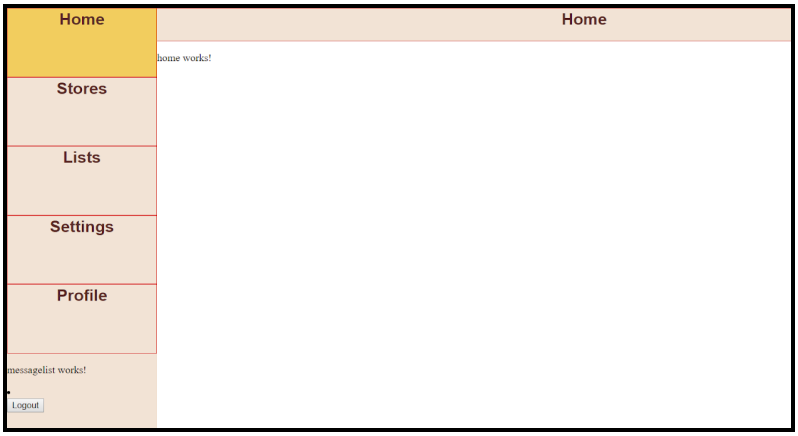

Figure 11. Angularjs Frontend

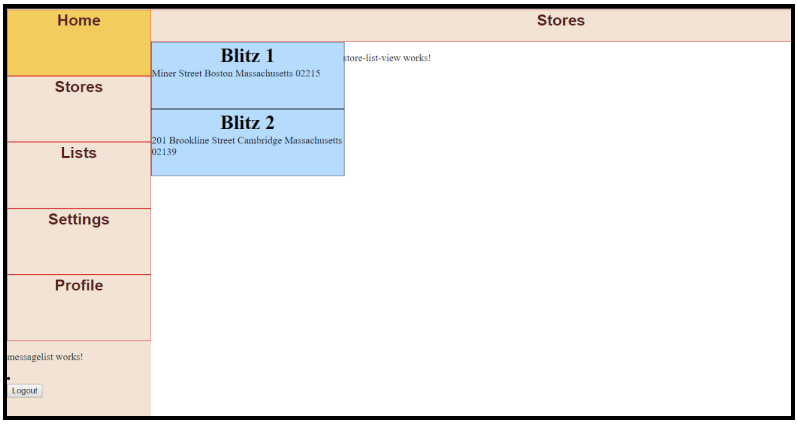

Figure 12. Data Fetching From Backend

Figure 13. Stacked Smart Shelving Unit Visualization