Electrical & Computer Engineering

EC464 Senior Design Project

Final 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: 04/10/2020

Table of Contents

Executive Summary

EC464 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 simplifies grocery store managing and shopping. There are two parties that can greatly benefit from using Blitz: store managers and customers. Store management does not have an easy way to track grocery store item inventory; currently, either employees are walking through the store manually updating item inventory or item inventory is being 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 made 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 will know exactly where to go in the store to find exactly what they need in the shortest time possible.

By employing simple shelving 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 load cells 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. System Description

The smart shelving units will allow for easy and efficient inventory management. The smart shelving units 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 converts 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 the Appendix, Figures 1, 2, 3, and 4. The ESP32 utilizes the HX711 library provided by the Arduino IDE to detect the current weight of the shelf. The ESP32 communicates the detected weight, once the weight has stabilized, to the store CPU via a Linksys router. The shelf's ESP32 determines that the weight of the shelf has stabilized and sends this weight to the store CPU when the current weight reading is within 0.05 lbs of the previous weight reading. This delay in communication assures that the shelf's ESP32 does not send weight readings that are a byproduct of a store employee or customer touching the shelf, or a byproduct of the load cells taking a few seconds to stabilize once a grocery store item has been placed on or removed from the shelf.

Our system currently includes three smart shelving units that communicate to the store CPU simultaneously. The weight readings communicated to the store CPU are organized by the IP address of the ESP32 of each shelf. Therefore, our current store CPU expects three weight readings from three different shelves with three different IP addresses. Shelf 1 has IP address 192.168.1.138, Shelf 2 has IP address 192.168.1.100, and Shelf 3 has IP address 192.168.1.119. The CPU receives the weight readings from these three shelves and detects item placement on and removal from these shelves.

To detect item placement on and removal from the shelves, we populated the shelves with their respective items in the database. We did this by associating an item to a shelf using the shelf's IP address. The item name, price, and weight was recorded and was associated with its respective shelf. Therefore, the database was able to detect item placement on and removal from the shelves by weight differences. For example, if Shelf 1 was responsible for the inventory of peanut butter (1 lb) and apple juice (4.15 lbs) and increased by 1 lb, the inventory of peanut butter on that shelf would increase by 1.

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 tables: one that tracks the total weight on each shelf (shelves) and one that tracks the store inventory (products). The shelves table has rows that are formatted as follows:

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

The products table has rows that are formatted as follows:

id VARCHAR(255) - name of product
weight FLOAT - weight of product (lbs.)
count INT - count of product
Price FLOAT - price of product
ip VARCHAR(255) - the IP associated with the shelf that the product is on

To facilitate communication between the CPU and the application, we decided to use two functions, an update function and a ping function. The update function would execute on a timer (i.e. once per minute) and would send the current store inventory to the database in OpenShift. The ping function would also execute on a timer but would be slightly more complicated. In this function, the CPU would send a ping to OpenShift. OpenShift would respond by sending any updates to the store layout back to the CPU. For example, if a product is removed from one shelf and added to another, the CPU would receive this update and fix the local database.

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 a store manager or employee and allows the store manager or employee to view store inventory and to create a map of their store. To authenticate users, the application uses Okta single-sign-on. Customers will be able to sign in, view store inventory, and create a shopping list. The customer will receive a shortest path through the grocery store based on their shopping list.

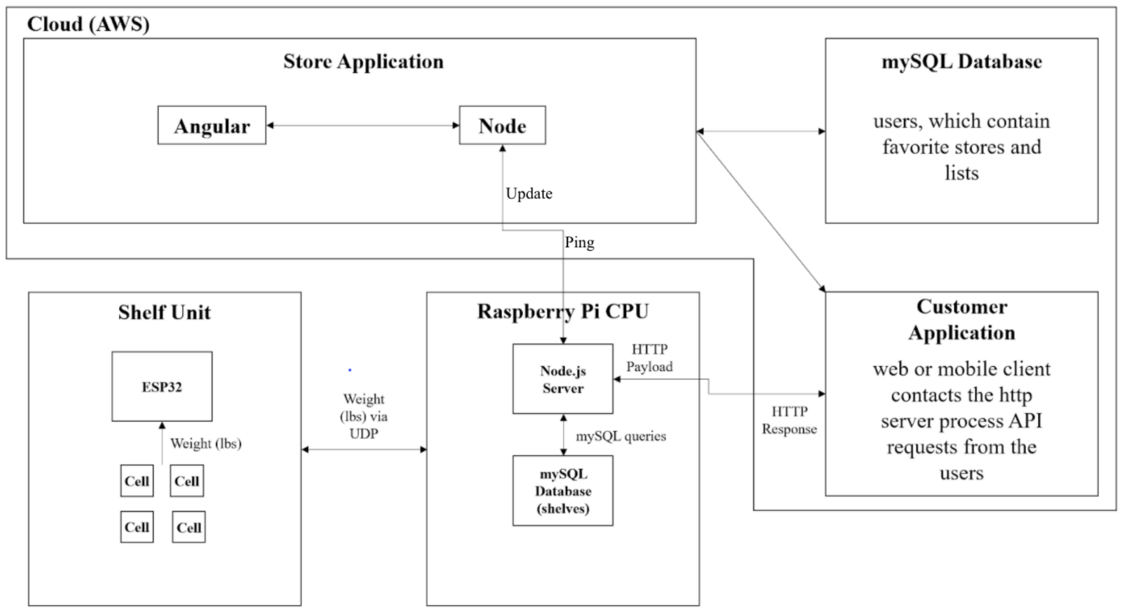


Figure 1. System Block Diagram

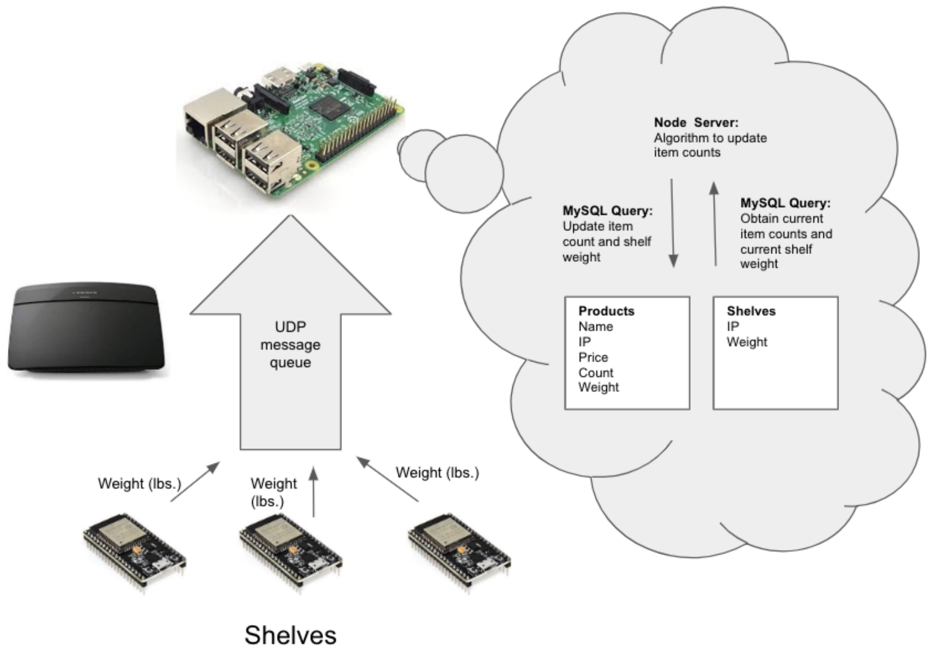


Figure 2. CPU Block Diagram

3. Second Semester Progress

This past semester, our team made some significant progress with the hardware and software aspects of our project. For hardware, we created two more smart shelving units that are functional and reliable. Therefore, we have three complete smart shelving units. Each of these units is made up of two plywood shelving surfaces, four load cells, a HX711 module, and an ESP32 module. These modules are wired according to Appendix, Figures 1, 2, 3, and 4. Over the course of the semester, these shelving units were debugged and rewired multiple times to assure that the units produced stable and accurate weight readings.

Furthermore, the two new shelving units were programmed into the database using the unique IP address of the ESP32 module associated with each shelf. Once the shelving units were programmed into the database, grocery store items were programmed into the database based on which shelf they were on. The item name, price, and weight was recorded in the database. The shelving units were able to communicate weight readings to the database via WIFI and the database was able to detect item placement or removal based on the IP address of the shelf, the weight reading being received from the shelf, and the known weight of the items on the shelf.

For our second semester testing, we utilized 12 products: 4 jars of peanut butter, 2 gallons of apple juice, 2 boxes of cheerios, 2 bags of goldfish, and 2 containers of oatmeal. Shelf 1 (IP .138) was responsible for the inventory of peanut butter and apple juice, Shelf 2 (IP .100) was responsible for the inventory of cheerios, and Shelf 3 (IP .119) was responsible for the inventory of goldfish and oatmeal. The shelving units were able to communicate stable and accurate weight readings to the database through WIFI; therefore, the database was able to detect the correct inventory of each shelving unit as items were placed or removed.

This semester, we also finalized the database schema for the CPU. The mySQL database, hosted on the Raspberry Pi, used to hold one table for shelf weights and one table for each of the shelves in the store in order to track inventory. This schema was deemed inefficient and was changed to a two table format. Instead of having one table per shelf, we implemented a products table. In addition to the name, weight, count, and price of a particular product, the table stored the ip of the shelf that each product was located on. Figures 15 and 16 in the appendix show the products and shelves tables in the mySQL terminal.

We also successfully connected all three shelves to the CPU via UDP simultaneously. Each shelf sent weight readings into a UDP queue to be processed by the inventory update algorithm. Ensuring that all shelves were updated within three seconds of an inventory change without any server crash required debugging the server. To implement this feature successfully, we changed the declaration of a variable that stored the current shelf weight reading of interest to be local to the socket function, which allowed each function instance to run independently. With this change, the connection between our shelves and our CPU was complete.

We also created our “shortest path” algorithm that was going to guide users through stores more efficiently based on their shopping list. Our algorithm, given a map of a store and an item list, can generate a shortest path between each item in the list and the entrances and exits of the store. The algorithm begins with the store entrance as the source; it then checks the Manhattan distance (or taxicab distance) to all items in the list and picks the item that is closest. It then sets that item as the new source and reruns the algorithm, checking the remaining items until the shopping list is empty and the shortest path has been created. Figure 17 shows our shortest path algorithm.

For the application, we managed to deploy it in the MOC cloud environment using Openshift. This was running a MySQL server, Node.JS backend, and Angular frontend running on NGINX. These containers were configured in the same namespace with only the Node and NGINX server being exposed through a route. The progress of the frontend and backend code was such that most major components were either written or soon to be written and combined. The infrastructure to connect the shelves to this architecture was also completed.

4. Technical Plan For Completion

Task 1. Implement an LCD display. The LCD display should display item name and price. This should take **one week**.

Task 2. Implement a PCB to improve shelving unit aesthetics and to assure good connection. This should take **one week**.

Task 3. Add borders to the shelving units to improve shelving unit aesthetics. This should take **one week**.

Task 4. Full system testing and debugging. Allow for **one week** of testing and debugging.

Task 5. Develop store-side app: view inventory, create store map. Finish Full Stack development in Angularjs and node.js with a MySQL Database. This should take **one to two weeks**.

Task 6. Test and verify REST endpoints on RPi with application. We will integrate the CPU with the application, showing that the update function pushes the inventory of the CPU to OpenShift and that the ping function signals the application to send any shelf layout updates back to the CPU. This should take no more than **five days**.

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. This should take no more than **one day**.

Task 8. Manufacture CPU. We will manufacture the CPU container in EPIC and insert the RPi. This should take no more than **three days**.

Task 9: Integrate algorithm for shortest path. We will integrate the completed shortest path algorithm into the front end system. This should take no more than **four days**.

Overall, a subsequent team would probably need one or two weeks to complete our project with all requirements met.

5. Budget

Item	Description	Cost
1	Shelving Surface x 6	$90.00
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
	Total Cost	$198.50

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. 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 two functions, update and ping, to communicate with OpenShift
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)

Hardware requirements 1, 2, 3, 4, 9, 10, and 11 were tested and met. Our smart shelving unit is the correct size to be employed on a grocery store shelf: smaller than 1m by 3.5m. Our smart shelving unit is adjustable because it is made up of two plywood sheets and can easily be produced at any desirable size. We utilized four load cells and these load cells are accurate to the nearest 0.025kg. This accuracy was tested with a 1 lb jar of peanut butter. The weight reading of this 1 lb jar of peanut butter is shown in Appendix, Figure 14. Each smart shelving unit also contains an ESP32 module that communicates the weight of the shelf to a CPU. The ESP32 also only communicates the current weight of the shelf when the weight reading is stable. This was tested by placing items on and removing items from the shelving units and watching when the weight readings were sent and assuring that that weight reading were stabilized when sent.

Hardware requirements 5 and 7 were not tested. Requirement 5 states that the shelving unit should be able to hold 150kg. We did not test this requirement; however, according to the load cell specifications, the load cells should be able to hold and should produce accurate readings for loads up to 200kg. Requirement 7 states that the shelving unit should be able to operate normally at 0 degrees celsius. We did not test this requirement; however, according to the load cell specifications, the load cells do not have any limitations due to temperature. Therefore, even though we weren't able to test these functionalities, these requirements would likely have been met if tested.

Hardware requirements 6 and 8 were also not tested. Requirement 6 states that the shelving unit must be primarily powered by an outlet but should last for at least 18 hours per battery charge. We were not able to test that our units could last for at least 18 hours per battery charge, as we did not have a finished product, and we were only utilizing an outlet for other testing. However, we believe that this requirement is feasible and would likely have been met if tested. Requirement 8 states that the shelving unit should communicate item name and price through a LCD display. We were unable to test this functionality, as we were in the process of implementing the LCD display before the semester came to a close.

Requirement 12 was not only met but exceeded, as the database schema only required two tables instead of one for each shelf. In addition to the name, count, and weight of each product, we also stored product price. However, requirement 13 was not met, as we were not able to get the weight reading on the shelf as accurate as we planned. Limiting the error to +/-0.01 caused some item placements to be missed.

Requirements 14 through 16 were in the design phase but were not completed. For requirements 14 and 16, we designed an update function that sent the CPU database to OpenShift and a ping function that recieved shelf layout updates from OpenShift. For requirement 15, we were planning to mount the cpu to the wall and use a microUSB to Type A plug supplied by Belkin to connect to power.

Requirement 21 was partially completed. Our shortest path algorithm works as a standalone function; however, it was not integrated into the front end so it cannot pull store information, item locations, etc.

Requirements 17 and 18 were met. Requirement 22 was changed to a more general cloud deployment and is mostly completed. Requirement 19 and 20 were partially completed.

7. Appendix

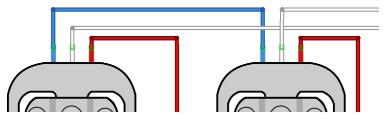


Figure 1: Load Cell Wiring Schematic

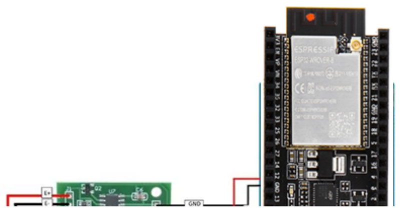


Figure 2: Load Cell, HX711, and ESP32 Wiring Schematic

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

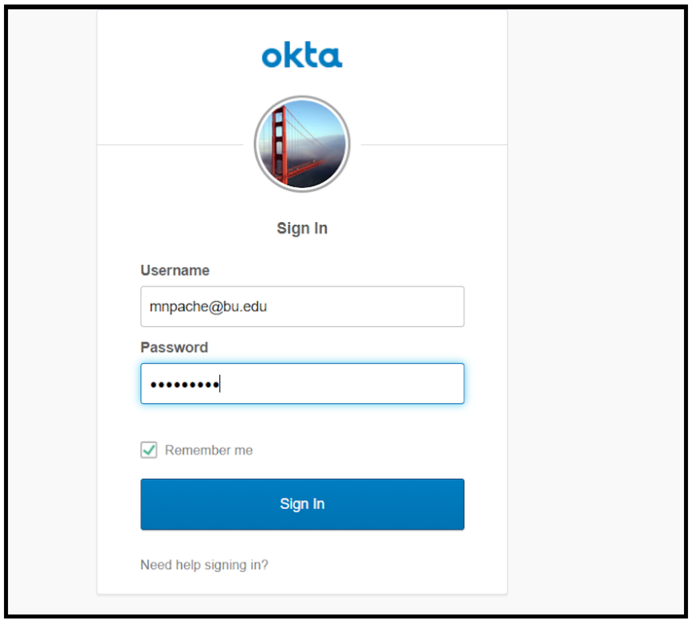


Figure 10. Authentication

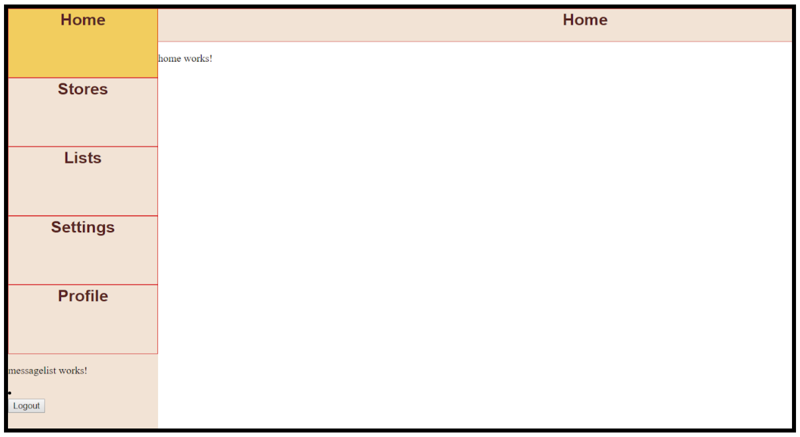
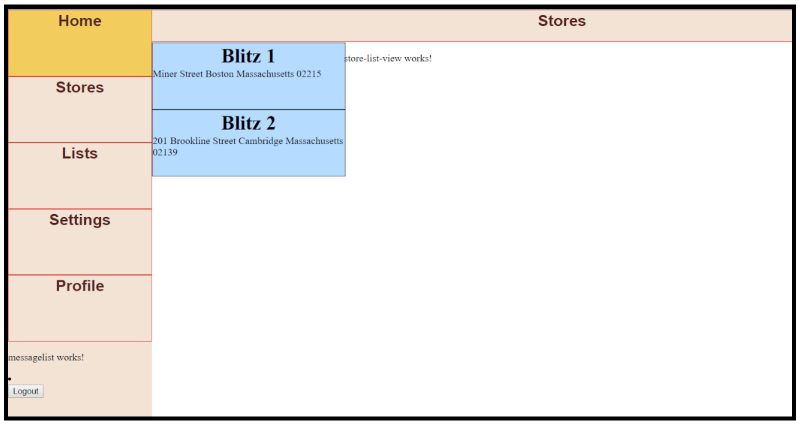


Figure 11. Angularjs Frontend



Stores

Blitz 1
Miner Street Boston Massachusetts 02215

Blitz 2
201 Brookline Street Cambridge Massachusetts 02139

store-list-view works!

messagelist works!

Figure 12. Data Fetching From Backend

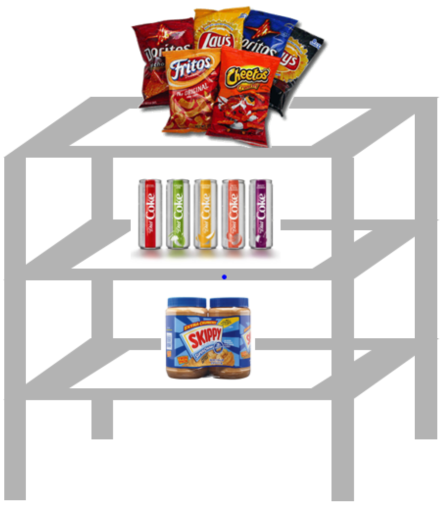
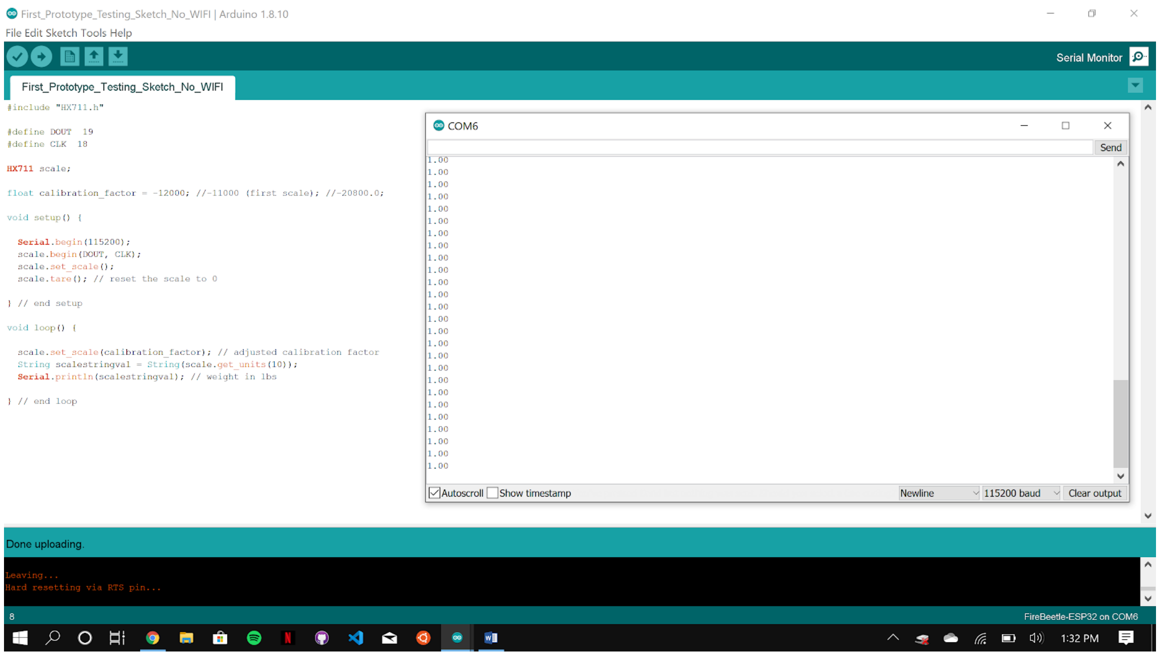


Figure 13. Stacked Smart Shelving Unit Visualization


First_Prototype_Testing_Sketch_No_WIFI | Arduino 1.8.10
File Edit Sketch Tools Help
Serial Monitor
First_Prototype_Testing_Sketch_No_WIFI
COM6
Send
Autoscroll
Show timestamp
Newline
115200 baud
Clear output
1:32 PM

```
#include "HX711.h"

#define DOUT  19
#define CLK  18

HX711 scale;

float calibration_factor = -12000; //-11000 (first scale); //-20800.0;

void setup() {

  Serial.begin(115200);
  scale.begin(DOUT, CLK);
  scale.set_scale();
  scale.tare(); // reset the scale to 0

} // end setup

void loop() {

  scale.set_scale(calibration_factor); // adjusted calibration factor
  String scalestringval = String(scale.get_units(10));
  Serial.println(scalestringval); // weight in lbs

} // end loop
```

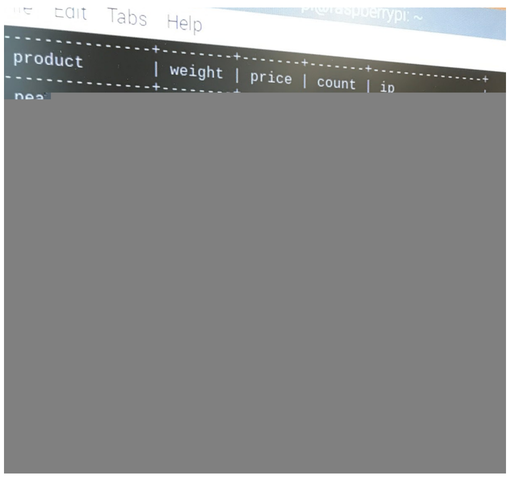
```
1.00
1.00
1.00
1.00
1.00
1.00
1.00
1.00
1.00
1.00
1.00
1.00
1.00
1.00
1.00
1.00
1.00
1.00
1.00
1.00
1.00
1.00
1.00
1.00
1.00
1.00
```

Done uploading.

```
Leaving...
Hard resetting via RTS pin...
```

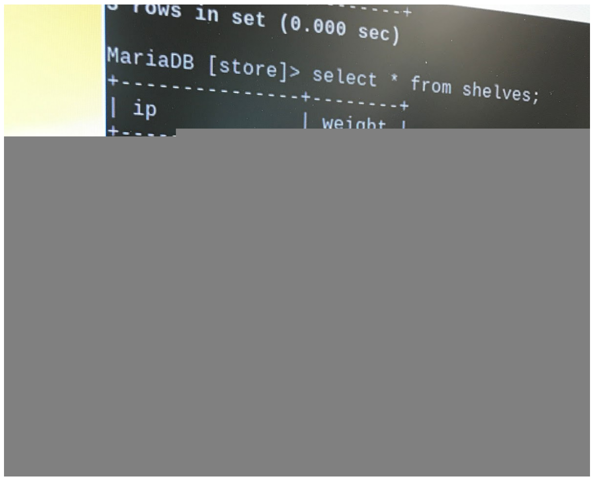
8 FireBeetle-ESP32 on COM6

Figure 14. Shelf Weight Reading of a 1 lb Jar of Peanut Butter



```
product          | weight | price | count | id
pea
```

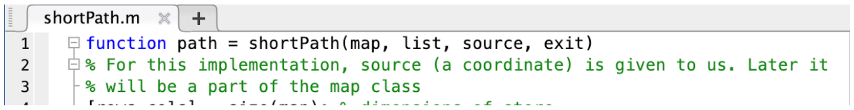
Figure 15. Products table, CPU database



```
 rows in set (0.000 sec)

MariaDB [store]> select * from shelves;
+---------------+--------+
| ip            | weight |
```

Figure 16. Shelves table, CPU database



```
function path = shortPath(map, list, source, exit)
% For this implementation, source (a coordinate) is given to us. Later it
% will be a part of the map class
```

Figure 17. Shortest Path Algorithm