University of Arkansas – CSCE Department

Capstone I – Final Proposal– Fall 2020

Home Re:Stock

Brandon Russell, Cody Sturgeon, Samuel Hudson,
Daniel Rowett, Alex Kalmes, Jackson Calton

Abstract

Home Re:Stock is a kickstarter project that is in need of software developers to create a robust and scalable backend server, as well as a visually appealing mobile application to interface with its current prototype device. The goal of Home Re:Stock is to provide users a more calculated and efficient means to keep their home stocked with necessary goods. The user will place a variety of affordable sensors around their homes, then register those sensors with the mobile application developed by this group. This application will communicate with the sensors through a backend server and allow the user to monitor and replenish goods in their home based on their specified preferences. Not only will this allow consumers more time to spend in meaningful ways, this could also provide a solution for those with disabilities to obtain their necessities in a more independent manner.

Although Home Re:Stock has provided us with a prototype sensor device, it will be our job as developers to try and improve the physical hardware, as well as write all of the software required to make the Home Re:Stock ecosystem fully functional. We will strive to utilize paradigms that offer the user consistent service, smooth and logical functionality, and persistence across multiple users and devices.. With these points in mind, our overall goal is to deliver all of the required software that will help move Home Re:Stock out of its development phase and into production.

1.0 Problem

Before the United States received its first wave of COVID-19 cases, the average American household made 2.7 trips to the grocery store weekly. Since COVID, the percentage of online grocery sales increased from 14.5% to above 29% [JC1]. Regardless of the pandemic, it is clear that as more convenient options are provided to consumers, they will pursue the route of convenience. In 2019, Forbes Magazine reported a prediction of $35 billion in revenue from Walmart curbside pickup services, of which 40-60% of customers were to be new Walmart shoppers [JC2]. Pre-pandemic projections like this are what gives Home Re:Stock tremendous confidence that they are tapping into an explosive market with the pandemic seemingly exaggerating the speed and trajectory of moving to an online grocery shopping platform.

The applications of a product like Home Re:Stock lies not only in consumer convenience, but also as a potential solution to giving those living with disabilities more autonomy and independence. Crippling anxiety, addiction, physical disabilities, and even old age are all factors that could have considerable influence on one's ability to go to a store and properly obtain the items one needs throughout their daily lives. A product like Home Re:Stock could give those living under these circumstances a better opportunity to obtain their necessities within their ability.

At the time of Home Re:Stock's conception and our joining its development, there were no other reported products quite like Home Re:Stock. This means that Home Re:Stock has the potential to widen the percentage of online shoppers considerably as something like this hasn't hit the market yet. Our goal is to provide a software application that pursues this theme of convenience in the digital shopping trade. With the help of our solution, Home Re:Stock will soon be able to provide the services it needs to be successful.

2.0 Objective

The objective of this project is to not only improve Home Re:Stock's existing prototypes, but also to provide them with a mobile application, backend server, and sensor firmware that all interact in unison to offer the user consistent service, smooth and logical functionality, and persistence across multiple uses and devices.

3.0 Background

3.1 Key Concepts

Home Re:Stock plans on creating a variety of different sensors that can keep track of products using alternative methods in the future, but for the scope of our project we will only be responsible for the 'scale' variant of the planned sensors. The scale sensor, which we refer to simply as the 'sensor' throughout the proposal, requires a product to be placed on top of it and is able to use the weight of the product in conjunction with some product information to determine its remaining stock.

The overall project will involve some hardware development, but the majority of the project will be software orientated. The software side will include the creation of a backend server, the main user interface in the form of a mobile app, and the firmware for Home Re:Stock's scale variant of its sensors. As for the hardware side of the project, it will only consist of refining the existing scale sensor design or branching it off into different form factors.

The backend server will function as a bridge between the user interface and all of the Home Re:Stock sensor variants. It will be responsible for interpreting all of the commands or queries issued by a user interface, and when necessary it may relay a command or request data from a relevant sensor. The server will also manage a database so that user preferences and sensor data can be stored and accessed securely.

In terms of hardware, Home Re:Stock provided us with a prototype scale sensor which we will need to physically refine and develop firmware for. The sensor's primary function is to periodically send data about the product it is monitoring to the server, but it will also need to be able to interpret commands relayed by the server and communicate directly with the user interface during its initial setup process.

The user interface will allow the user to communicate with the backend server and in turn communicate with the sensor indirectly. It will also be able to access information stored on the server's database and store a short time period's worth on the user's device so that recent data can be viewed even when offline. In the long term Home Re:Stock aims to constantly build upon the user interface so that it can increase user convenience by suggesting alternative products and even automatically reorder products based on user preferences.

3.2 Related Work

Currently the most direct competitor to Home Re:Stock is Amazon. Amazon had a product called the 'Dash Button' which has been discontinued, but this product was limited in that it was tied to a single product listing on Amazon, and it did not directly monitor a product's levels. Home Re:Stock differs from the Dash Button as it will be able to monitor the actual amount left of any desired consumable product and provide reordering suggestions before the product runs out completely, and it is also not tied to a specific store.

Another competitor produced by Amazon is Amazon's 'Dash Services'. At first glance it may seem like a similar product to Home Re:Stock as it is able to track the levels of consumables in certain products, however it is only available when that product has been built specifically to support the feature. Dash Services is a software that is built into certain technologies that use consumables, like printers (ink), smoke detectors (batteries), and other smart devices. The devices themselves are able to monitor the levels of something they utilize and automatically notify you or reorder it when it is needed. This is great for monitoring certain products used by a said device, but we do not consider it a direct competitor to Home Re:Stock as it is not intended to monitor consumables that are not utilized within another device. On top of this, by design it is

technically tied into one specific product and is not able to be swapped to measure something else. [CS1]

Home Re:Stock will provide a similar product to Amazon Dash services, but it will focus far less on items specifically used by a separate device. We will make it easier to use, and compatible with almost any product that you can "run low" on. This will mainly consist of everyday-use items like food, dog treats, trash bags, laundry detergent, and other things of that nature. It will be easy to use with no intense set up. Consumers can choose whatever product they want to measure, and the sensor will never be locked to one particular product, allowing it to be more versatile than both Dash Services and the Dash Button. Home Re:Stock will make everyday life at home easier by doing the grocery shopping for you!

4.0 Design

4.1 Requirements and/or Use Cases and/or Design Goals

In the short-term, Home Re:Stock aims to provide users with means to an affordable network of sensors that not only monitors the levels of various products throughout your house, but that can also provide the user with fast access to the information as long as they have an internet connection. In the long-term, Home Re:Stock aims to drastically increase a user's convenience by utilizing sensor data and user preferences in order to potentially re-order essential products before they are needed but not available. A list of specific design goals and requirements can be found below:

Sensor goals:

- Affordable physical design. The total retail cost should be no more than $8 per sensor. (The prototype provided by Home Re:Stock was $5.30, so we will try and stick to that)
- It should be quick, easy, and intuitive for the user to set up and configure a sensor using the Home Re:Stock mobile app.
- The sensor should only be configurable via direct connection (its own temporary network), or through the server by an authorized user.
- The sensor should only communicate with the Home Re:Stock server once it is set up through the app. This connection should be secure.
- The sensor should be very power efficient.
- Function normally at fridge-like temperatures~.
- Be able to communicate with a home's wifi network at a reasonably long range (from the wifi access point)
- The sensor should automatically send values such as weight and battery level to the server at periodic intervals. The sensor should receive a confirmation from the server, or a resend request depending on whether or not the server thinks the data is "good".
- Be able to receive commands from the server at any time.

- The sensor should be rechargeable OR allow for easy battery replacement. (This depends on the sensor model.)
- The sensor's "0" measurement should be able to be recalibrated via the user interface.
- Be able to be physically reset so it can be freshly re-set up in odd circumstances.

User interface goals:

- Be able to communicate with the server.
- Be able to communicate directly with a sensor during initial setup. There should be an easy way to set up a new sensor with a good step by step walkthrough outlining the specific details.
- The user interface should be the primary way of issuing commands to the server, so that the server can then issue them to the sensor.
- Be able to create an account or login to an existing account. Be able to save login info for quick access in future.
- The user interface should support adding a large number of sensors, and having a large number of sensors should not impact any functionality.
- Be able to assign a name, description, and product to a specific sensor. (and reassign)
- Be able to unassign a sensor without deleting it from the network.
- Be able to delete a sensor. (This will issue a reset command to the sensor)
- Be able to view sensor data. This will include battery, signal strength, timeline, etc, data.
- Be able to set user preferences for the app / account. This will include things like disabling data sensor data collection or allowing predictive reorder suggestions.
- Recent data should be stored on the user's device so that if the user does not have an internet connection they can still view the data from when they did.
- The user interface should closely resemble the mockups created by the champion. All actions should be intuitive.

Server goals:

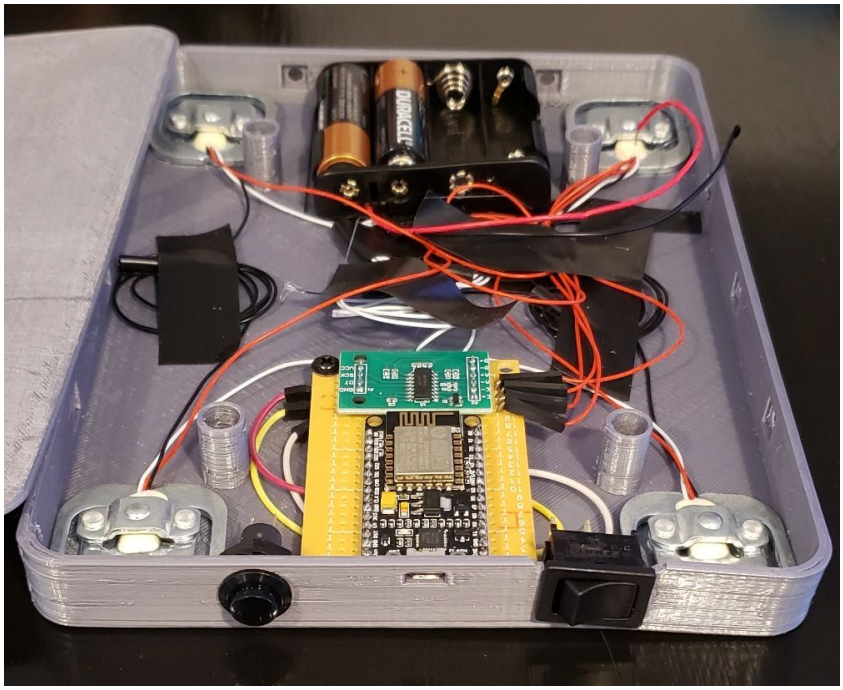
- Communicate efficiently with a large number of sensors and user interfaces simultaneously and securely.
- Be heavily threadable for future expandability.
- Utilize a database to store all data.
- Store user related information securely. All personal information needs to be secured using the most currently accepted standard security procedures.
- Store sensor related information efficiently. There will be a lot of data over time so this is important. Plan to store all sensor data for at least 6 months.
- The server should have measures in place to easily delete all user and / or sensor related data upon request.
- Individual sensors should be associated with individual users. Only associated users will be able to access any given sensor's data.

- Be able to integrate with company APIs such as Amazon or Walmart. This will be used for product lookup and potentially reordering purposes.
- On top of what is listed in this section, the server should be able to handle all communication related functions listed in the “Sensor goals” and “User interface goals” sections.

4.2 High Level Architecture

Sensor Design:
The sensor design is primarily broken into two main parts: The sensor hardware, and the sensor software.

Sensor Hardware:
In terms of sensor hardware, the sensor requires a bare minimum of 5 components to function. This includes a casing / housing for all of the parts to be secured inside, 1 to 4 load cells, a load cell amplifier, a microcontroller with a built in wireless chipset, and finally a power supply. As mentioned above, all of these parts need to be cheap (but reliable) so the total retail cost of the sensor is no more than $8. The design itself is quite straightforward. The load cells produce a minute change in resistance based on the weight on them, which is then input into a load cell amplifier. This produces a measurable output which is then fed to the wireless chipset. Finally the data is processed and sent over a network to the Home Re:Stock server.



The above image is the initial prototype provided by Home Re:Stock. As you can see, it is truly in its “prototype” phase, and will need various improvements.

As far as refining our design, we decided that the casing needs to be as thin as possible so that the device does not take up much vertical room. It also needs to be made of a sturdy material, that does not have much flexibility. Since there are a large variety of products with varying sizes, we decided that we will need to make varying sizes of sensors to ensure that a user can pick one that is right for their needs. The actual hardware inside the casings will not need to be altered per sensor, just the positioning of some components. This was an important decision as if a sensor is too small for a given product it could cause inconsistent data to be collected. In terms of load cells, these are a bit tricky to refine.

The next, and arguably most important thing that we will refine is the type of load cell and load cell amplifier we are going to use. These are the backbone of the sensor, as they are what actually produce weight outputs. Because of this, they need to be accurate, reliable, and consistent between different sensors. The load cells used in the initial Home Re:Stock prototype we were provided with were built for weights up to 50kg, and as a result had poor low weight outputs. This was clearly not ideal for our application, so we decided to look into higher precision load cells instead. While the exact specification is not yet final, we are planning to utilize “Thin Film” load cells instead. These types of load cells are much higher precision at lower weights, but are also more expensive. As a result, we are going to reduce the number of load cells per sensor from 4, to 1. This will introduce new challenges regarding weight balancing on the sensor, however we are confident that we can solve this problem with further development.

In terms of the wireless chipset, we are not going to stray from the initial prototype’s NodeMCU hardware. This hardware is extremely cost effective, and perfect for our desired use.

The final piece of hardware that will need further attention is the battery powering the sensor. Whilst we can get away with a simple battery for our prototype, the final product needs to have a well thought out solution so that users are not constantly having to replace or recharge the battery. The best thing we can currently do is make sure that the hardware itself is extremely power efficient. Otherwise, until we are able to begin long term testing we will not be able to accurately predict an expected battery life for the product, and as a result we will not make a final decision on the batteries accessibility or serviceability either.

Sensor Software:

The sensor software provided with the initial prototype is extremely minimal. It currently simply allows a client to connect to a network broadcasted by the sensor, and request a measurement, which is then sent by the sensor to the client. This will be completely overhauled to meet the following high level design:

The sensor will need to have two main modes of function. When the sensor turns on without any network credentials in memory, it will need to enter setup mode. This mode will cause the sensor to create a temporary WiFi network, where it will then wait for a user to connect via the User Interface. Once a user is connected, it will receive a configuration from the user interface including the user’s home network, credentials to connect to said network, and the user’s account

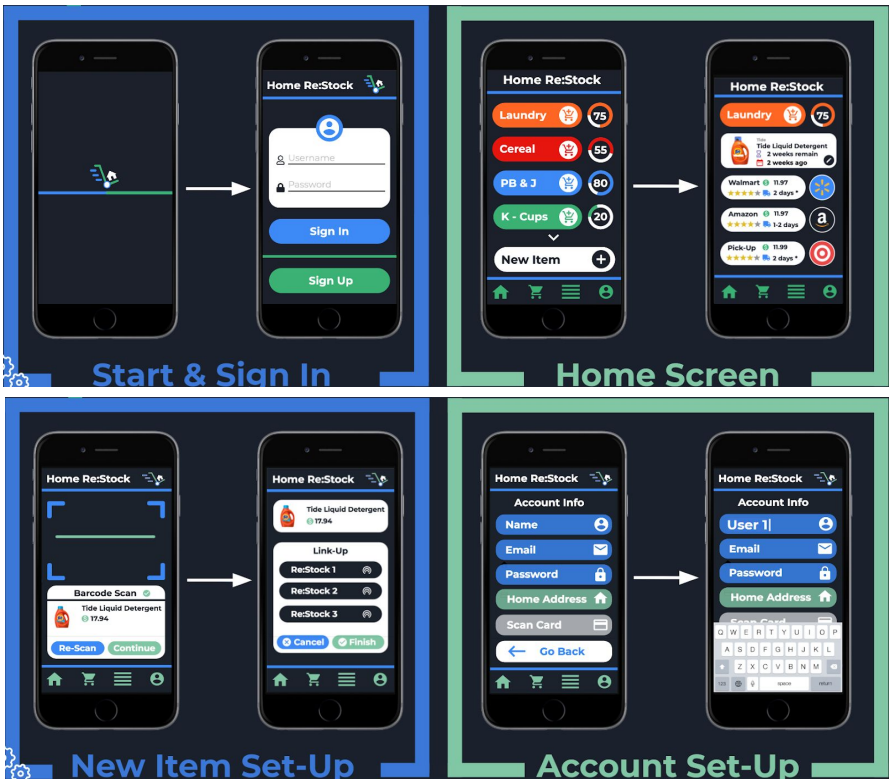
details for authorization. After the sensor verifies that it can connect to the Home Re:Stock server, it will reboot itself into standard operating mode.

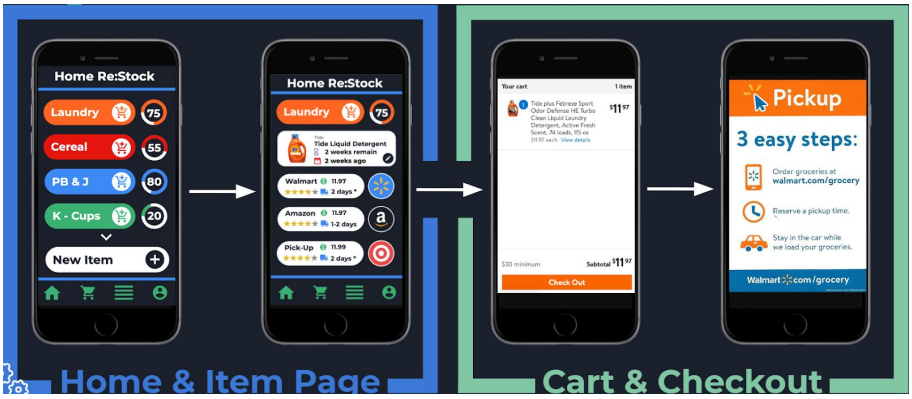
In standard operation mode the sensor should only accept commands from the Home Re:Stock server if they are issued by an authorized user. As such, all incoming communications will need to be checked for credentials, and all outgoing communications will need to provide sensor identifying information. A list of commands the sensor will need to interpret include:

- Set sensor credentials
- Set periodic auto-send interval for status updates
- Tare / Zero
- Force send a status update including battery level, ~signal strength, and current measurement
- Clear sensor memory (factory reset)

As you can see, the sensor software will not be very complex as its purpose is solely to send information to the server.

User Interface:
The user interface will be the second largest part of the project after the server / backend. If we have time to create both an Android and Apple iOS app that would be ideal, but we would rather fully implement one or the other than partially implement both. We were provided with the following mockups from Home Re:Stock to base the user interface off of:





Whilst we do have some creative liberty, we need to keep the design aesthetic consistent with the mockups, and should certainly try to match them as closely as possible.

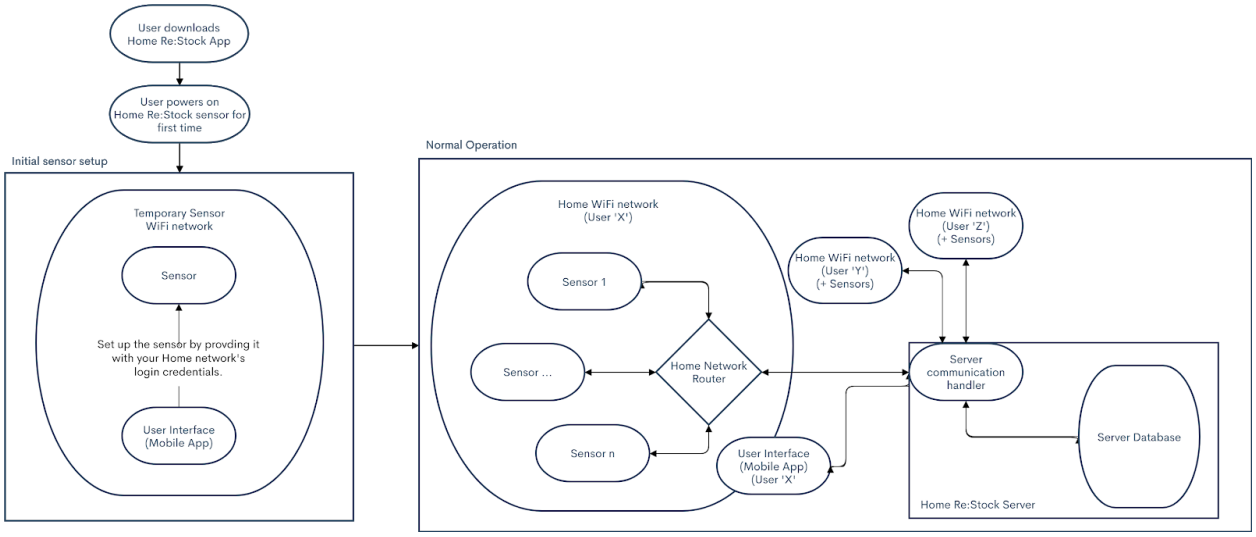
As for how the user interface actually functions, we will need to make it communicate seamlessly with the server so that it can perform a variety of actions. All of these actions will interact with the server by sending information regarding the command the user interface wants the server to execute, as well as any relevant data to accompany it. The client will then wait for a response from the server, and process any data if it is returned. As mentioned many times throughout this document, the user interface will rely on the server for the majority of its operations.

In the event that the user does not have an internet connection, the user interface should be able to at least view the data that it received the last time it was connected to the internet. This will mainly include things like viewing a sensor's last reported levels and information regarding the product it was monitoring.

Server / Backend:
This will be the largest part of the project. There is no initial prototype so it will be completely developed from scratch. One of the biggest parts of the server will be the communication handler. This will allow the server to send and receive data from both the user interface and the individual sensors. We will separate each possible command into its own function / class, and we will take particular care to clearly define the exact input and output data formats for each and every command. This will be essential in getting everything to communicate properly, and will allow for easier development of user interfaces on different platforms.

Another large part of the backend will be the database. The layout of this will need to be thought out in great detail before writing any code, as it will be very difficult to alter later down the line. The database will store all information regarding user accounts, which users own which sensors, which products are assigned to which sensors, and of course the data that all sensors collect. The server will access all of this information by performing queries based on the data relevant to each command. We will need to take precautions to ensure that the server can not be tricked into querying for and sending data to unauthorized users. This will likely be done through data sanitization and strict command input regulation.

General communication overview:



The above diagram should make it very clear that the user interface does not communicate directly with the sensors during standard operation. All sensor and user interface communication is handled by the Home Re:Stock server. This is to ensure that the user's device is not responsible for collecting data from the sensors, and subsequently that the sensor's are still able to collect data when the user's device is not nearby.

4.3 Risks

Risk	Risk Reduction
Wasting a large amount of development time learning and debugging	Use familiar technologies, development methods, and implementations, and assign people to tasks they are best suited for.
Wasting time trying to get the sensor, user interface, and server to communicate.	All *general* communication related standards will be carefully defined before starting any coding. The server inputs / outputs will have the highest priority. As such, they will be fully defined before the command itself is implemented. All user interface and sensor inputs / outputs will conform to the server's requirements. If the server does not have a command in place to accept a desired action from a sensor or user interface, a placeholder function with detailed input / output requirements should be

	implemented on the server before continuing with the development of the sensor or user interface.
Trouble getting access to retailer APIs (Walmart, Amazon)	Use existing API documentation and company provided sample queries and responses as the basis for related designs. This way even if we can not directly get an API account it will not halt development.
Difficulty performing comprehensive tests with a physical device that not all members have access to	Since the entire sensor is designed to be very cheap, the main hardware component of the device can be obtained off of Amazon for a reasonable price. It is called the 'Node MCU' (The cheapest ones are found on aliexpress.com, but have ridiculous shipping times). As long as you have a USB Micro B cable, you will be able to power the device, upload software to it from your computer, and perform simulations with 'fake' data. Since the prototype needs to be upgraded regardless, once final parts are decided upon we can order enough to make an entire sensor for every group member who wants a physical copy.

4.4 Tasks

Hardware related tasks:

- Look at various cheap "thin film" load cells and determine which one will be the best suited for our sensors. Consider price, min weight, max weight, overload %, etc.
- Determine a method of balancing the weight on scale better. Perhaps one spring in each corner would work decently. Need to be careful not to affect the reading though.
- Figure out a way to keep the sensor pressure plate attached firmly even when upside down.
- Make the sensor as thin as possible. Will likely require removing the vertical pins from the Node MCU board.
- Come up with a small, medium, and large form factor sensor. The large scale should be able to support at least 10kgs of weight, and should be a good size for large containers of laundry products. The medium should be a good fit for a gallon of milk, and the small should be a good fit for a litre of milk. These sizes are fairly well known and should work with a variety of other household items. All form factors should have the same components with the exception of the large form factor one allowing for higher capacity load cells.

Server related tasks:

- Create a comprehensive list of the commands that are expected to be sent to the server from the user interface. Define detailed input and output formats for each of these commands. Make a note of which of these commands is forwarded to the sensors.
- Create a comprehensive list of the commands that will be sent to the server from the sensors. Define detailed input and output formats for each of these commands.
- Discuss what database table structure would be ideal for the project. Figure out all of the expected table names and column names per table. Include data types.
- Figure out how to host a test server and integrate it with the database. Heroku is a good option.
- Begin writing the NodeJS server code. Start by creating a structural outline with empty functions representing the major components that will need to be implemented at some point.
- Write code that allows the server to store information in the database. Make sure that sensitive information is protected and not in plain text
- Write code that allows the server to query and retrieve information from the database.
- Begin implementing the comprehensive list of server commands that are essential for communication between the 3 softwares.
- Look at how to use Walmart / Amazon API for product information queries. Begin to implement skeleton code using response examples.
    - Sensor communication and user interface communication is highest priority.

Firmware related tasks:

- When the sensor first boots it needs to check to see if it has any existing credentials. If it does not, it needs to enter into setup mode. If it does, it needs to enter into normal operation mode.
- Make the sensor create a temporary network during setup mode. Since there is no server interaction for this step, clearly define what information the sensor is expecting to receive from the user interface in order to complete the setup process.
- Write the part of the program that handles the setup process.
- Figure out how to determine how much battery is left on the sensor
- Figure out how to determine the signal strength of the sensor to the home network
- Write the part of the program that handles normal automatic operations such as sending a status update to the server containing battery info, signal info, and current sensor information.
- Write the part of the program that handles incoming commands issued by the server. Use the detailed input and outputs outlined by the server as reference.

User Interface related tasks: (See mockups above for all screen names)

- View Android UI mockups provided by Home Re:Stock. Figure out what is missing and create some more to fill in the gaps. Suggest changes to existing ones if you feel it can be better.

- Begin implementing skeletons for the various screens. Make sure that all of the transition buttons work properly. (Even without communication to the server, 99% of screens should be reachable as they will just display the last data, so transitions should never break.)
- Fully implement the account creation screen
- Fully implement the account login screen
- Implement the “Home screen”
- Implement the new sensor setup process with detailed instructions on the setup process. The Wyze camera setup process is an excellent example of what the app should strive for. This screen is accessed via “Home screen”
- Implement the screen that appears when you select an individual sensor. This screen should automatically issue a sensor status update command upon being selected if the user has internet connection. Until a response is received, it will show the last data. This screen should allow the user to primarily view the sensor status, name, currently monitored product, and relative weight. It should also have a button that enters another screen which allows you to rename the sensor, edit the product / remove the product, issue a tare command, or delete the sensor (by issuing factory reset command). This screen will likely look different than the mockups provided, as it will contain less product info and more sensor info instead.
    - Probably should create a new mockup for this first. Current one is too focused on where you can buy the monitored product and should be more focused on sensor information.
- Implement the sensor modification screen, (accessed from the screen described above by the edit button)
- Implement the user preferences screen. This should allow you to opt out of information collection, request data deletion, change account details, etc.
- Implement the remaining screens. Prioritize the ones that enable interaction with the sensor and server.
- Constant: Bug testing, code clean up, visual interface inspection / clean up. Ask yourself “What is this missing?” or “What is this not able to do that I want it to do?”

4.5 Schedule

Tasks	Dates
Hardware	
Determine which load cell is the most suitable for the 3 sensors	1/15 - 1/18
Figure out out to better balance a load on the sensor	1/19 - 1/23

Design a better sensor plate mount	1/28 - 2/1
Make the sensor as thin as possible	2/1 - 2/4
Design a small, medium, and large sensor	2/5 - 2/12
Server	
Document commands that will be sent to the server from the app and sensors, including detailed input and output	10/21 - 1/2
Determine and design database schema	1/3 - 1/11
Decide where to host a test server	1/12 - 1/13
Begin creating the server with a skeleton app that defines the routes needed	1/13 - 1/24
Write code for interacting with the database	1/24 - 2/6
Implement server route handlers for all of the server <-> sensor commands	2/7 - 2/21
Implement server route handlers for all of the server <-> user interface commands	2/22 - 3/14
Research how to use retailer APIs	3/15 - 3/21
Implement retailer APIs	3/22 - 4/3
Final testing and debugging	4/4 - 4/11
Firmware	
Implement boot into setup vs standard operation mode	1/13 - 1/20
Create code for temporary network in setup mode	1/21 - 1/28
Write the part of the program that handles the setup process	2/1 - 2/8
Figure out how to determine battery life and signal strength	2/9 - 2/16

Write the part of the program that handles normal automatic operations like sending status updates	2/17 - 2/23
Write the part of the program that handles incoming commands issued by the server	2/24 - 3/2
Final testing and debugging	4/4 - 4/11
User Interface	
Review existing mockups and create any missing views. Improve existing views where needed	12/13 - 12/19
Implements a skeleton of the app with functioning view transitions	1/3 - 1/13
Implement account creation	1/14 - 1/30
Implement login	1/31 - 2/7
Implement user preferences view	2/8 - 2/17
Implement home view	2/8 - 2/17
Implement new sensor setup	2/18 - 3/5
Implement sensor view	3/6 - 3/13
Implement sensor modification view	3/14 - 3/23
Implement the remaining views	3/14 - 4/30
Final testing and debugging	4/4 - 4/11

4.6 Deliverables

- **Hardware design overview document:** Contains a listing of each sensor design that made it to the development stage. Each design will contain further details outlining its intended use case, component cost analysis, and any information specific to that sensor design alone.
- **Hardware design CAD files:** The 3D CAD files relevant to each sensor design. The designs will mainly be focused on the sensor chassis, but individual component layout

within the chassis will also be included. These will likely be Autodesk Inventor or Sketchup designs.

- **Software design overview document:** Contains in depth design details regarding the software for the sensor specific software, user interface, backend server, and database scheme. Each individual section will also contain documentation and a generalized description for particularly important or unique sections of code.
- **Sensor software:** The C++ code that will run on all of the sensors. If any sensor requires specially modified C++ code, that will be included separately.
- **User interface mockups:** A comprehensive set of mockups that the final user interface software will closely resemble.
- **User interface:** The source code for the Home Re:Stock Android application.
    - Potentially the Swift code for the Apple iOS application. Whether or not we create this will depend on how much time we have left after fully developing the android application.
- **Backend server code:** The complete NodeJS code for the Home Re:Stock backend server.
- **Database scheme and initial data:** We are unsure of which particular database
- **Final Report**

5.0 Key Personnel

Brandon Russell – A senior Computer Science major in the CSCE Department at the University of Arkansas. He has experience developing and managing complex websites and applications. The most notable of these are “Bots on Discord” and “Mirai Bot for Discord”. His experience from these will help with the planning and development of the backend for Home Re:Stock. He will mainly be responsible for the server and helping with the user interface when needed.

Cody Sturgeon - Sturgeon is a senior Computer Science major with a minor in Mathematics in the CSCE department at the University of Arkansas. He has completed, or currently enrolled in, all of the CSCE core degree classes. This includes learning C++, Java, Python, NodeJS, and SQL. He has also completed the electives Cryptography and Artificial Intelligence with A’s, and is currently enrolled in Computer Graphics and Programming Challenges. Cody will work primarily on the user interface and server, as well as the database.

Daniel Rowett - Rowett is a senior Computer Science major in the CSCE department at the University of Arkansas. He is currently enrolled in, or has completed all of the CSCE core degree classes, as well as some relevant electives including ‘Big Data Analytics & Management,’ ‘System Synthesis & Modelling,’ and ‘Computer Networks’. He will be primarily responsible for the hardware and server related tasks, but he will also be helping out with other parts as needed as the project’s general coordinator / quality control.

Alex Kalmes - A senior Computer Science major in the CSCE department at the University of Arkansas. Alex has various experience with coding through the classes he has taken during his

time at the university. This will help with wherever he is needed for the project, whether it's working on a certain task or helping another member with their task. He will primarily work on the user interface, but is interested in working on the server aspect of the project as well.

Sam Hudson - Sam is a senior computer science major at the University of Arkansas. He has experience in web and software development, and professional experience in database management. He will assist in developing the relevant software for the Home Re:Stock website and the physical product. He plans to primarily contribute to database related tasks and on the user interface, but is also interested in some firmware development.

Jackson Carlton - Jackson is a senior at the University of Arkansas studying Computer Science. He has professional experience in data engineering and data analytics and educational experience in software development. His experience with mobile application development, database management systems, and networking will be necessary to contribute meaningfully to Home Re:Stock. He plans to contribute to this project primarily by working on the user interface, but is also interested in working on the server and database.

Nauman Malik - (Project Champion) As the person who originally came up with the idea for the product and has kept it alive through all of its ups and downs, Nauman has steadily worked towards laying the groundwork to make Home Re:Stock a reality. Whilst he is currently in his senior year as a Biomedical Engineering undergrad, he has always been interested in entrepreneuristic ideas and has a strong talent for producing eye-catching ideas and designs.

6.0 Facilities and Equipment

6.1 Facilities

The only facility we are likely to use is the university's FAY Fabrication Laboratories. We will likely request to use the CNC lab as well as their 3D Lab when we get close to finalizing our sensors hardware designs.

6.2 Equipment

- **Computer**: Anyone working on the software side needs a computer strong enough to run android development software or a programming IDE of their choice.
- **Autodesk Inventor (Student)**: AutoCAD software is free for students. This software requires a strong enough computer to run as well.
- **Arduino IDE**: Arduino IDE is an easy to use and open source software that enables coding for, and uploading to arduino hardware.
- **3D printer / CNC**: Only one group member has access to a 3D printer regularly, so we potentially utilize 3D printers or CNC machines from the facility discussed above.
- **Electrical tools**: A soldering iron and other electrical tools are required for the hardware side of the project to modify the prototype.
- **Thin film load cell**: A load cell is a small sensor that can be used to measure force by outputting a varying resistance in Ohms. They can be bought for very cheap depending

on the specification. 1-4 of these will be connected to a load cell amplifier to properly read a load.

- **Load cell amplifier**: The change in resistance generated by load cells is usually far too small for standard devices to read. A load cell amplifier is able to amplify the change in resistance to a value that is more readable by a standard device.
- **NodeMCU Lua**: NodeMCU is a lua based firmware designed for the ESP8266 Wifi development board [SH1]. The ESP8266 microchip built into the board can connect to a wifi access point, run a simple http server, and do a variety of other network related tasks. This board serves as the main computing unit in the initial prototype.
- **Battery**: The type of battery is yet to be decided, but the battery used for the prototype can be something trivial.

References

[JC1] *FMI: 2020 U.S. Grocery Shopper Trends. (2020). Retrieved November 12, 2020, from https://www.fmi.org/our-research/research-reports/u-s-grocery-shopper-trends*

[JC2] *Danziger, P. (2019, April 08). Walmart Leads The Soon-To-Be $35 Billion Curbside Pickup Market. Retrieved November 12, 2020, from https://www.forbes.com/sites/pamdanziger/2019/04/07/walmart-is-in-the-lead-in-the-soon-to-be-35-billion-curbside-pickup-market/?sh=7a0481b7199e*

[CS1] *Amazon Dash Services Information Website https://developer.amazon.com/en-US/alexa/dash-services?pf_rd_m=ATVPDKIKX0DER&pf_rd_s=merchandised-search-7&pf_rd_r=T3QZE2P6RFTZEP5ZH45S&pf_rd_t=101&pf_rd_p=d9277661-081a-440c-8a1a-f834dcab6c6a&pf_rd_i=15426532011*

[SH1] *https://nodemcu.readthedocs.io/en/release/*