University of Arkansas – CSCE Department

Capstone II – Final Report – Spring 2021

Home Re:Stock

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

Abstract

Home Re:Stock is a kickstarter project that was in need of software developers to create a robust and scalable backend server, create a visually appealing mobile application, and improve their existing prototype device. Home Re:Stock's goal is to provide users a more calculated and efficient means to keep their home stocked with necessary goods. In teaming up with Home Re:Stock, we aimed to deliver all of the required software that would help move Home Re:Stock out of its development phase and into production.

After our contributions to Home Re:Stock, the user can now place a variety of affordable sensors around their homes, then manage those sensors with the Home Re:Stock Mobile Application. This application communicates with the sensors through a backend server and gives the user the ability to utilize their sensors in a variety of ways. The user can create and manage a personal account, register new sensors, and monitor usage data of products associated with each of their sensors. Our solution provides users the ability to gain meaningful insight into their resource consumption. Home Re:Stock has the potential to be a massive data analytics solution in more domains than just the consumer level. In the future, the Home Re:Stock Mobile App should be extended to incorporate retailer APIs, tested with various testing scripts, and deployed to the app store.

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 lie 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. Home Re:Stock also has the potential to be a massive solution in the data analytics domain. Giving users profound insight into their resource consumption has applications in consumer, retailer, and even manufacturing analytics.

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 now has the potential to widen the percentage of online shoppers considerably as something like this hasn't hit the market yet. Our goal was to provide a software application that pursues this theme of convenience in the digital shopping trade as well as the potential in data analytics. With the help of our solution, Home Re:Stock can now provide the services it needs to be successful.

2.0 Objective

The objective of this project was not only to improve Home Re:Stock's existing prototypes, but also to provide them with a mobile application, backend server, and sensor firmware. All of these components were designed to work in unison and 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 were responsible for the 'scale' variant of the planned sensors. The scale sensor, which we refer to simply as the 'sensor' throughout the report, 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 involved some hardware development, but the majority of the project was software orientated. The software side included 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 consisted of refining the existing scale sensor design and branching it off into different form factors.

The backend server functions as a bridge between the user interface and all of the Home Re:Stock sensor variants. It is responsible for interpreting all of the commands or queries issued by a user interface, and handles all of the non-setup related communication with the sensor. The server also manages a MongoDB database so that all user and sensor data can be securely stored and accessed.

In terms of hardware, Home Re:Stock provided us with a prototype scale sensor. Over the project we were able to refine its physical design and create a completely custom firmware for it. The sensor's primary function is to periodically send data about the product it is monitoring to the server, but it is also able to interpret commands relayed by the server and communicate directly with the user interface during its initial setup process.

The user interface allows the user to communicate with the backend server and in turn communicate with the sensor indirectly. Its primary function is to access information stored on the server's database, but it has the optional ability to store data directly on the user's device so that it 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 particular device, but we do not consider it a direct competitor to Home Re:Stock as it is not intended to monitor generic consumables. On top of this, by design it is tied into one specific product and is not able to be swapped to measure something else. [CS1]

In the long term Home Re:Stock aims to provide a similar product to Amazon Dash services, but it will focus far less on items specifically used by a separate device. Over time, they 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. Consumers can choose whatever product they want to measure, and the sensor will always be reassignable. This flexibility alone makes it more versatile than both Dash Services and the Dash Button. Couple this with the fact that Home Re:Stock will utilize real sensor data rather than just predicted values, and it becomes clear that Home Re:Stock has a strong plan in place to make sure their product stands out.

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 for our project’s scope can be found below:

Sensor goals:

- Affordable physical design. The total retail cost should be no more than $8 per sensor.
- 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 initially configurable via direct connection
- 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.
- Have an acceptable WiFi range for operation in a large home
- The sensor should automatically receive and send relevant data to the backend server
- The sensor should be rechargeable OR allow for easy battery replacement.
- The sensor should be able to interpret and act on user and server issued commands.
- 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 the sensor’s initial setup.
- Be able to create an account or login to an existing account.
- Be able to save login info for quick access in future.
- Be able to add any number of sensors to a single user account.

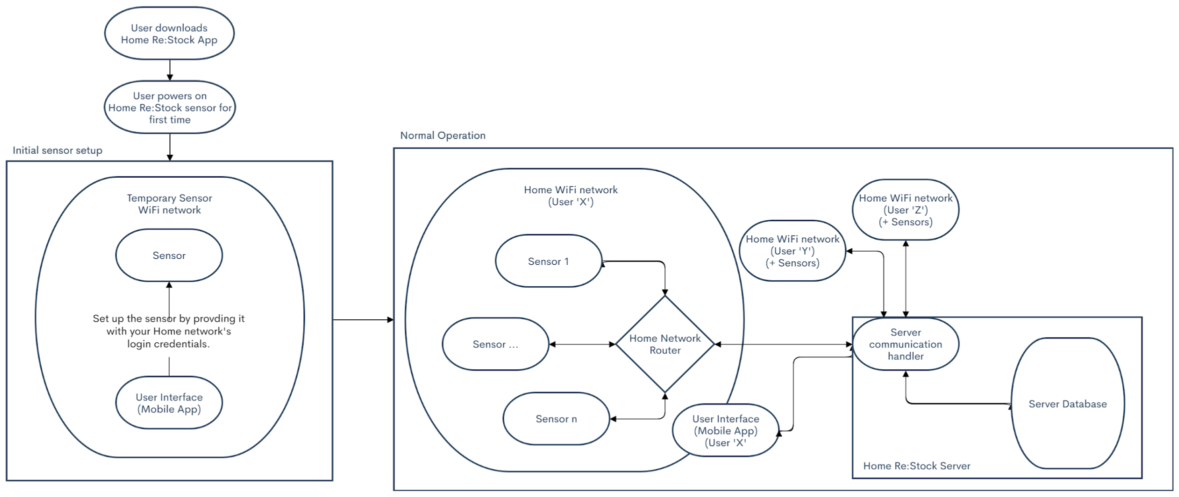
- Be able to issue command(s) to a selected sensor.
- Be able to configure or view the information of a selected sensor.
    - Includes: Battery, Signal Strength, Data over time, Firmware version, Status Interval
- Be able to set user preferences for the installed app or logged in account.
- Recent data should be able to 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 their last connection.
- The user interface should closely resemble the mockups created by the champion.
- All actions should be intuitive.
- Gesture based actions should be considered where applicable.

Server goals:

- Communicate efficiently with a large number of sensors and user interfaces.
- Be designed with future expandability in mind.
- Utilize a database to store all sensor and user related 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.
    - Sensor data should support being stored for at least 6 months.
- The server should have measures in place to easily delete all user and / or sensor data.
- Sensor data should only be retrievable through the server by the sensor's owner.
- Be able to integrate with company APIs such as Amazon or Walmart. This will be used for product lookup and potentially reordering purposes in the future.
- The server should be able to handle all communication related functions listed in the "Sensor goals" and "User interface goals" sections.

4.2 Detailed Architecture

This project is broken into 4 main parts: The sensor physical design, the sensor firmware, the user interface, and the backend.


User downloads Home Re:Stock App
User powers on Home Re:Stock sensor for first time
Initial sensor setup
Temporary Sensor WiFi network
Sensor
Set up the sensor by provding it with your Home network's login credentials.
User Interface (Mobile App)
Normal Operation
Home WiFi network (User 'X')
Sensor 1
Sensor ...
Sensor n
Home Network Router
User Interface (Mobile App) (User 'X'
Home WiFi network (User 'Y') (+ Sensors)
Home WiFi network (User 'Z') (+ Sensors)
Server communication handler
Server Database
Home Re:Stock Server

Ecosystem communication overview

4.2A Sensor physical design:

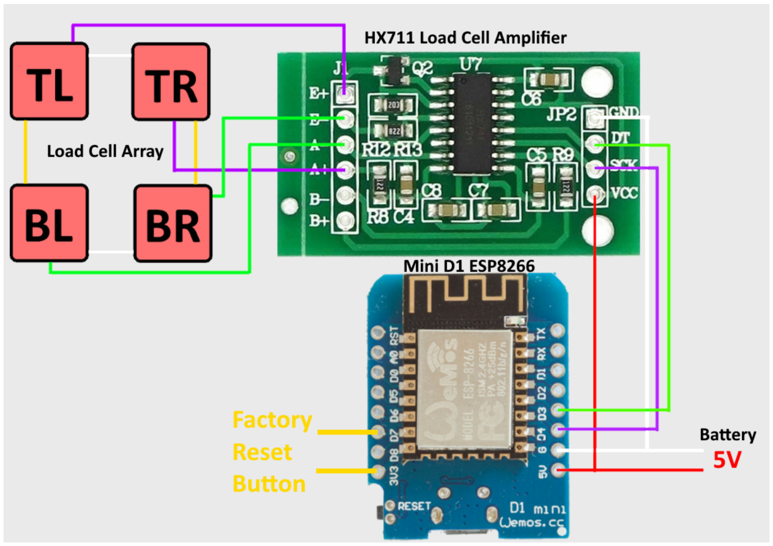
Architecture and technology:

The sensor requires a bare minimum of 5 physical 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. These parts needed to be cheap, but still reliable, so the total retail cost of the sensor is as low as possible.

The concept behind the design itself is quite straightforward. The load cell(s) produce a tiny change in resistance based on the force exerted on them, which is then input into a load cell amplifier. The load cell amplifier uses this change in resistance to produce a measurable output which is fed to the chipset for processing. This value is finally sent to the Home Re:Stock server.

Implementation:

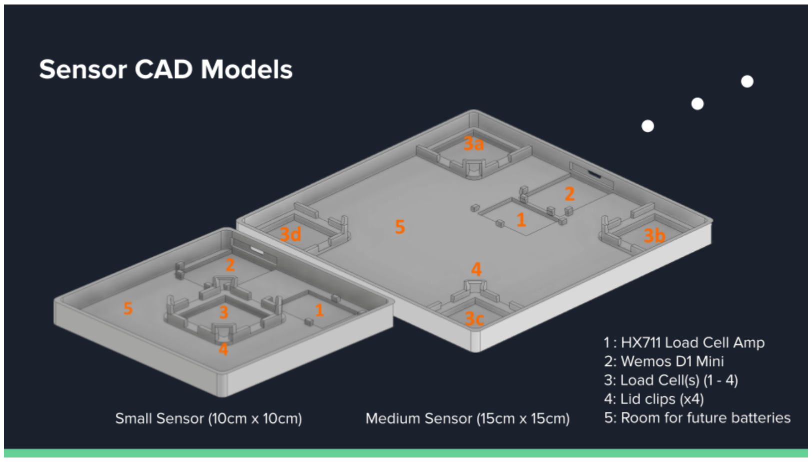
Our electrical implementation did not differ very much from Home Re:Stock's provided prototype. We used the same HX711 Load Cell Amplifier, the same type of 50Kg capable load cell, and the same ESP8266 based chipset for our microcontroller. One of the largest changes we made was moving from the provided Node MCU board to a Wemos D1 Mini board. This particular board was not only cheaper, but it also had a smaller form factor, which allowed us to make sensor chassis changes. Despite the board being different, it was still equivalent in its functionality compared to the Home Re:Stock provided prototype.



Wiring Schematic for all x4 load cell sensor variants. TL = Top Left, BR = Bottom Right, etc.

Due to the change from a Node MCU to Wemos D1 Mini board, we were able to shrink the height of the sensor chassis from 23mm to 14mm. Another change we made to the chassis was including insets for all of the components. These eliminated the need for adhesive or screws for mounting all of the components, as they could instead be mounted via push-fit instead. The largest chassis change in terms of a noticeable performance gain we made was creating proper mounting spots for the load cells. This solved the problem of inconsistent readings observed in the provided prototype. A final change we made to the chassis design was the incorporation of lid clips. This enabled the sensor lid to be easily popped on or off, but would not allow the lid to fall off if accidentally turned upside down.

For this project we decided to make 2 sensor chassis variants. The first was a ‘small’ variant, which had a size of 100 x 100 x 14 mm. The small sensor was designed with things like a liter of milk in mind, and could easily fit into spots such as a fridge shelf. The small sensor design is unique because it had very tight space constraints, and as such we had to go with a single load cell design instead of the typical 4. The second design was a ‘medium’ variant which had a size of 150 x 150 x 14mm. This design was based on the size of a gallon of milk, and is the one we suspect will be most applicable for measuring a variety of items. We originally intended to design a 3rd ‘large’ variant, but we decided to spend our time on other parts of the project as anything larger than the medium proved to be the same design as the medium but larger.



CAD Model overview for our small and medium chassis sensor designs

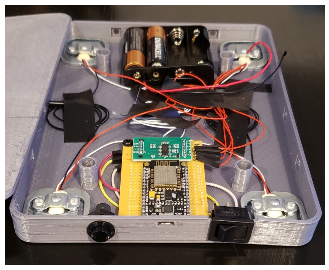
Component Price Breakdown:

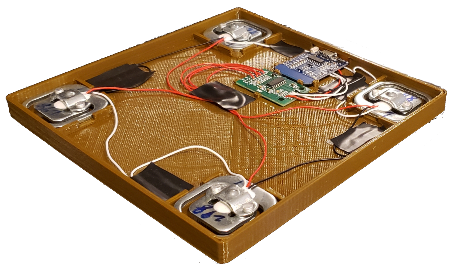
All prices are based on listings from Aliexpress, which is as close as you can get to wholesale prediction as an individual consumer. Links to specific products pages are not included as Aliexpress sellers come and go.

Component	Price	Prelim Price
HX711 Load Cell Amplifier	$0.50	$0.75
50Kg Load Cell (x1)	$0.25 (x4)	$0.30 (x4)
Mini D1 ESP8266 Based Development Board	$1.80	$2.75
Molded Chassis*	Assume =>	$0.50
Lithium Battery	Assume =>	$0.11
Total	$3.91	$5.31

* Prelim Price refers to the cost of products on the prototype provided by Home Re:Stock
* We are unsure of how Home Re:Stock calculated their Molded Chassis cost. Injection molding can have a large upfront cost for the initial mold design, so we will use the price they referenced.

The difference in our prototype's prices mostly comes down to swapping out the Node MCU development board to the Wemos D1 Mini development board. This allowed us to save nearly a dollar per unit, which is quite substantial considering the overall device price. As for the other price changes, we believe they stem from increased part availability over the last couple years, as the components themselves are the same.


Prototype - Provided by Home Re:Stock


Prototype (Medium) V2 - Created by our group

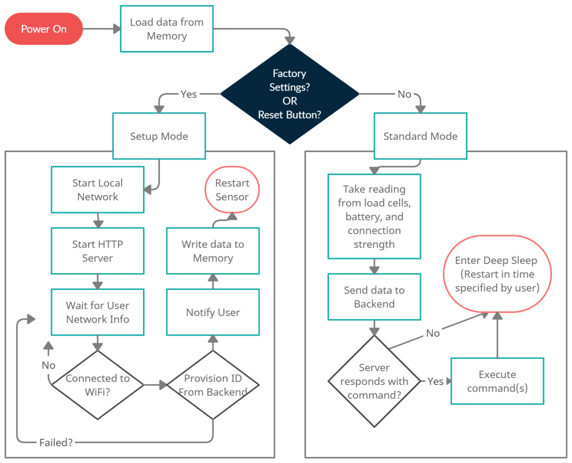
Future Work:

The next step in development would be to analyze the power efficiency of the sensor and try to make improvements where possible to decrease the power draw. This is a time consuming process, but would allow Home Re:Stock to choose which battery capacity is suitable for their desired use cases.

4.2B Sensor Firmware:

Architecture and technology:

The sensor firmware was written with two major modes of operation in mind. The first mode, called 'setup mode', enabled the sensor to be configured from a factory state via the user interface so that it could connect to the user's home network. After being set up for the first time, the sensor would from then on out enter the second mode upon startup, called 'standard mode'. Standard mode enables the sensor to send data regarding its current status to the Home Re:Stock backend server, as well as receive queued commands from the user that cause the sensor to perform a specific action based on the command.


Power On
Load data from Memory
Factory Settings? OR Reset Button?
Yes
No
Setup Mode
Standard Mode
Start Local Network
Restart Sensor
Start HTTP Server
Write data to Memory
Wait for User Network Info
Notify User
No
Connected to WiFi?
Provision ID From Backend
Failed?
Take reading from load cells, battery, and connection strength
Send data to Backend
Enter Deep Sleep (Restart in time specified by user)
No
Server responds with command?
Yes
Execute command(s)

Sensor Firmware operation overview

Implementation:

As a preface, all of the firmware code was written in an Arduino wrapped version of C++. The code can be compiled to the correct binary format and uploaded to the physical development board using the free Arduino IDE.

The Memory management part of the firmware involves a custom structure containing all of the variables that need to be persistent between restarts. This includes things like a boolean representing if the device should enter setup or not, a calibration factor and zero offset for the scale readout, and then user related information like the network login and sensor owner. During the sensor startup process the persistent memory is checked for two specific strings, one at the start of the memory and one at the end. If these strings are not equal to a specific phrase then the sensor considers its memory corrupt, and will factory reset itself so it is not rendered useless. The factory reset process can also be started manually via a physical button, or through a command in the user interface. These decisions were made so that it is extremely difficult, if not impossible to 'brick' the sensor through software related problems alone.

When the sensor boots into setup mode it first broadcasts its own local WiFi network named 'Home ReStock Scale'. This allows the user to connect directly to the device so that they can send it their own network's details. Once the sensor receives details from the user it will attempt to connect to the Home Re:Stock backend, and if successful, will provision itself an ID, notify the user of a successful setup, calibrate the scale's zero value, and finally write all of this information to the persistent memory so it can reboot into standard mode. If the sensor is not able to provision itself an ID, it will inform the user of exactly what went wrong. All of the design choices in setup mode were made with ease of use in mind. We did not want the process to take a long time, and we did not want the device to be able to fail silently. This led to us developing a setup process that can be completed in as little as 30 seconds in conjunction with the user interface, and is very responsive when dealing with unexpected scenarios.

Standard mode is what the sensor will be performing for the majority of its lifetime. It starts off by retrieving an up-to-date HTTPS certificate from the backend, which it can then use to secure all of its subsequent transmissions. The sensor then collects a reading from the load cell amplifier, as well as from its on board sensors that determine connection strength and battery level. If any of these numbers are erroneous the firmware sets the weight reading to a special 'hardware failure' number which lets the backend and user know that the data is bad. Once data collection is complete, the sensor sends this information to the backend, which will respond with a success value, as well as a list of queued commands that the user has issued. If any commands exist, they are then iterated through and executed. Finally, the sensor will read the value from the backend response representing the user's defined 'status interval' time, and put itself into a deep sleep power saving mode until that amount of time passes, at which point the sensor reboots, restarting the entire process again.

The currently supported user commands are: Factory reset, Update server root, tare, and update firmware. As discussed previously, the factory reset command allows the user to reset the device in the event something seems irrecoverably wrong. The update server root command enables the device to retrieve a new backend base URL in the event that the server needs to be migrated. The tare command allows the user to change the scale readout zero offset, which can be useful if they want to put something on the scale but not have it factored into the weight reading. The update firmware command is particularly important, as it allows the user to update the sensor's firmware to a newer version in the event that more commands or functionality is added in the future. These commands are all issuable through the user interface and are designed with sensor lifetime in mind, as we did not want the sensor to ever be completely unusable or unable to be updated.

4.2C User Interface:

Architecture and technology:

The most important consideration for the Home Re:Stock Mobile Application was portability. Because of this, we chose React Native, a hybrid mobile technology that utilizes the React and Babel libraries to compile React Native components into Native components of the user's operating system. This means our app can be run seamlessly on both Android and Apple mobile devices. We also used a framework/bundling platform called Expo to handle the building, networking, and deployment of the mobile application.

Implementation:

The Home Re:Stock Mobile Application was built using React Native conventional design patterns. The App is rendered as a tree of components, having data being passed down from parents to children. In a manner similar to the Model-View-Provider design pattern, global data in the form of a Context was built at the Application component level (highest level in the component tree). This Context object is accessible to all children of the App component. A reducer function was built out for the Context object so it could make changes to the data wrapped by the Context object. This meant that based on certain actions taken by the user or application, the data encapsulated in the global Context object could be updated, removed, or retrieved from any screen across the whole application.

This application also utilized a global navigation structure that partitioned the screen navigation into two separate navigation flows. The initial navigation flow is the first half of the navigation structure and is the path taken when the user first opens the application. This flow includes the screens associated with logging in and account creation. The other half of the navigation structure includes the flow of users transitioning between all the other screens in the app. A global navigation reference was built to provide app specific navigation methods as well as ways to pass data between screens in an app specific way.

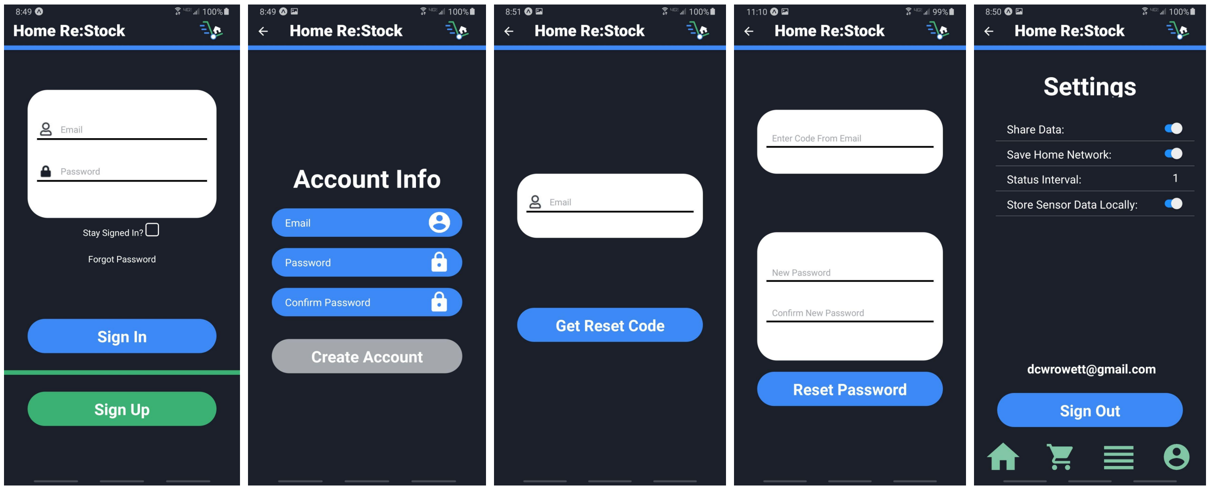
All of the communication with the backend server is handled asynchronously to avoid feeling any load in lag while new data is being retrieved. In conjunction with React's event based nature, this allows the user experience to feel smooth at all times, regardless of how many sensors and how much data is associated with your account.

Despite being heavily backend reliant, we wanted to make sure that a user could still view their sensor's data even when offline. If a user enables the 'store data locally' setting, found on the preferences screen, the app will store the most recently retrieved sensor data on the local device so that it may be viewed even when offline. This locally stored data has an added benefit of making the app feel more responsive, as oftentimes retrieving data from a network can be much slower than retrieving data from local storage. This means that any screen reliant on sensor data for UI elements can be rendered with old data, rather than being blank or replaced with a placeholder, while the more recent data loads in.

Account related screens:

The login screen is the initial screen that the user sees when the app is pulled up. This is where the user will enter in their email and password to login, and can choose to save this information for quicker access in the future. If the user doesn't have an account, they can hit the "Sign Up" button and create one. Also accessible from the login screen is the reset password screen, which is where the user can initiate the 2-step password reset process.

Once the user logs in, they will be able to access the settings screen. This screen has options such as sharing data with the server, saving the home network information for ease of repeated sensor setup, changing the default sensor interval, and the option to save sensor data locally. The user can also see what account is currently logged in, as well as log out. The settings screen will certainly have more options the user can configure as more features are added.

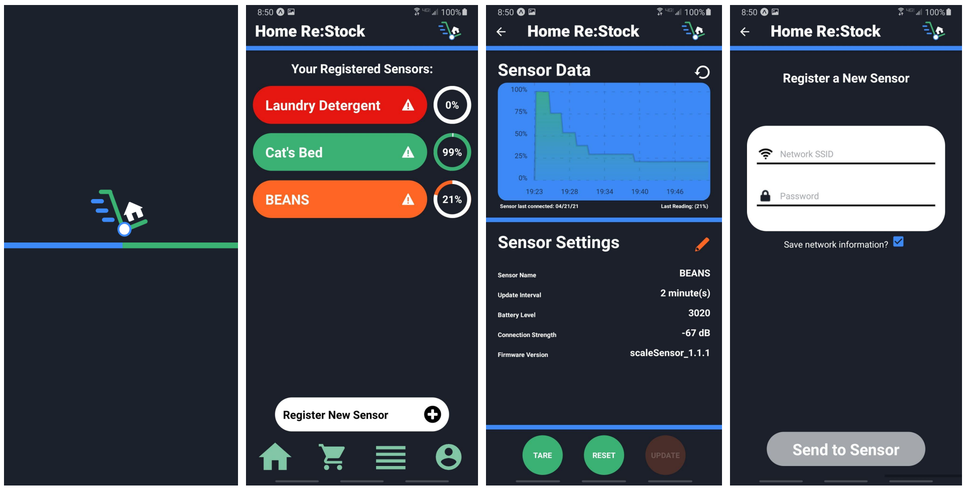


In order: Login, register account, initiate reset password, finalize reset password, and settings screens

Sensor related screens:

The Home screen is where a user can see all of their registered sensors as well as register a new one. Each sensor on this screen changes color dynamically based on its current readout, as well as displays an informative icon if any notable events are occurring with the sensor, such as low battery or a firmware update being available. If a user clicks on a sensor it will open that sensor's 'detailed view' screen, which lets them see a graph over time of its data, edit or view its name or preferences, and send commands to the sensor. If the user clicks on the 'Register New

Sensor' button on the Home Screen, it will bring them to a screen which displays the steps involved in setting up the sensor. From there they can send their home network's information to the sensor. The sensor will then notify the user if it was set up correctly, and bring them back to the Home screen where they can open the newly registered sensor for configuration.



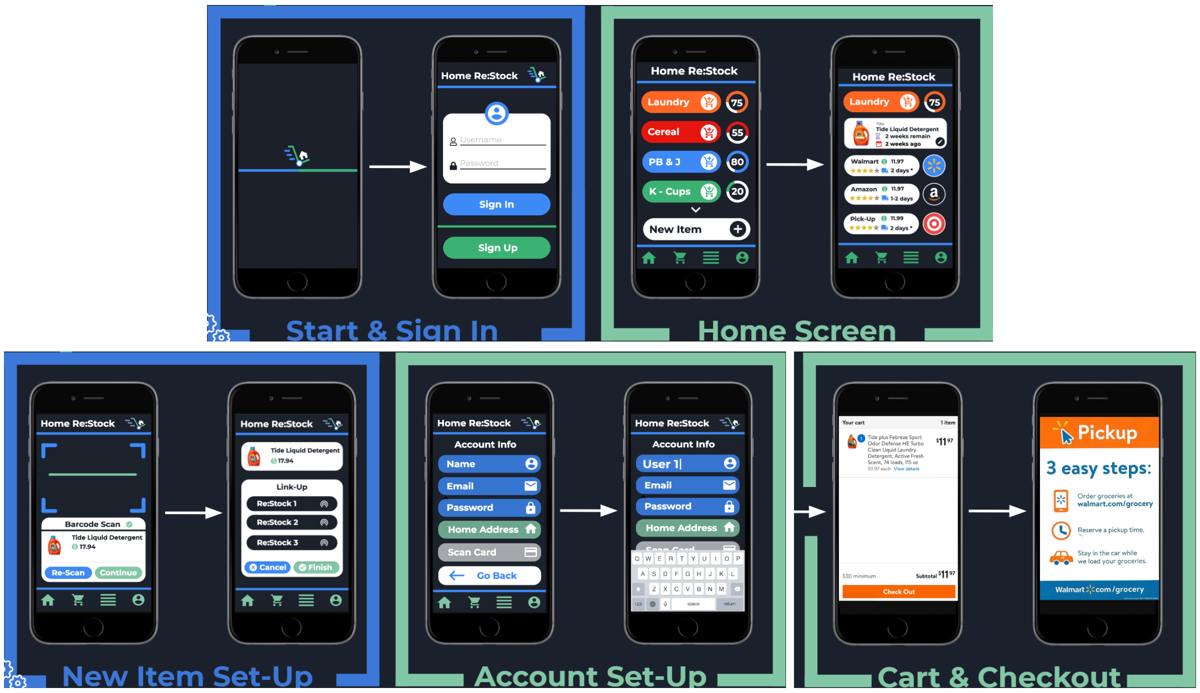
Sensor Name	BEANS
Update Interval	2 minute(s)
Battery Level	3020
Connection Strength	-67 dB
Firmware Version	scaleSensor_1.1.1

In Order: Splash, Home, detailed view, and register new sensor screens

Future Work:

The next steps in development for Home Re:Stock would be to begin implementing product and retailer integration. When this is complete, they can begin analyzing the best way to utilize sensor data for achieving their overall goal of increasing user convenience.

Home Re:Stock provided mockups for reference:



4.2D Backend and database:

Architecture and technology:

We decided to use NodeJS for the backend server. Not only is Javascript easy to learn and use, but some of our members already had experience in it. This worked well with our frontend app also being written in Javascript. For our database we decided to use MongoDB due to familiarity and ease of integration. The API itself is implemented using Express, which is widely recognized, simple to use, stable, and performant enough for our needs.

For hosting we currently use Heroku and MongoDB Atlas. Heroku has been very convenient for development, but we may run into issues with pricing when scaling up into a release. MongoDB Atlas should make scaling very easy for us, but we will have to look into our options more in the future as our usage and data grows.

Implementation:

The main purpose of the API is to act as a bridge between the physical sensors and the app. Sensors send status updates to the server, which are then added to the database. The app can pull sensor data from the server for display, and in-between we can process the data as needed for efficiency and security. In turn, the app can update settings and issue commands, which the sensors will receive upon doing their regular status updates.

The Home Re:Stock API is currently implemented in a single process. As API usage increases we could easily run multiple processes behind a load balancer to handle demand. No data is cached and all database modifications use versioned updates to prevent race conditions.

Home Re:Stock requires a user account for use. User accounts are created using an email and password combo. Only one account can be made with an email. Passwords are encrypted using a salted SHA-512 hash, which is still considered to be very secure. To authenticate after logging in the client receives the hash of its password. We do not generate a token for temporary use, which would provide an extra layer of security if the token/hashed password were exposed. When creating an account or changing a password an email confirmation is needed. We currently send emails using the main Home Re:Stock gmail account, but recognize that a gmail account is only a temporary solution during development.

Future Work:

To fully implement the Home Re:Stock idea we must integrate with third-party APIs from stores and marketplaces like Walmart and Amazon. These will allow price quotes and automatic reordering from within the app. Each company has their own API with its own challenges, so this would require a decent amount of work. On top of this, we would need to get in contact with these companies and get approved to use their APIs.

Additional work must also be done to ensure that the platform can scale as needed. This could potentially involve redesigns and/or changing hosting services/providers.

4.3 Risks

Risk	Risk Reduction
Wasting 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.	Have a clear communication ecosystem outline in place. Document expected inputs and outputs before writing any code. Use constants for any permanent route names, and share these constants between the software.
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	Create a clear wiring schematic for the sensor. Provide a detailed part lists outlining exactly what you will need to rebuild it. Provide instructions for uploading the firmware to the device.

4.4 Tasks

Hardware related tasks:

- ✓ Determine the best load cell type from price, performance, and form factor.
- ✓ Determine the best method of weight balancing on a single load cell.
- ✓ Figure out a way to keep the sensor pressure plate attached even when upside down.
- ✓ Make the sensor as thin as possible.
- ✓ Increase the reliability of the measurement output.
- ✓ Come up with a small, medium, and large form factor sensor. Incorporate well known household product sizes.

Server related tasks:

- ✓ Create a comprehensive list of all of the routes that will be needed for communication between the sensor, user interface, and server.
- ✓ Determine the best overall database structure for the project.
- ✓ Determine the best method for storing large amounts of individual sensor data.
- ✓ Setup a 'live' testing environment. (Heroku integration)
- ✓ Create Node JS server skeleton. Include all routes needed using the lists outlined above.
- ✓ Write server code that handles database interactions including storage and retrieval.
- ✓ Implement the comprehensive list of commands outlined above.
- ➢ Look at how to use Walmart / Amazon API for product information queries.
  - ○ Begin to implement skeleton code using response examples.

Firmware related tasks:

- ✓ Implement the check to see if the sensor should boot into setup mode or standard mode.

- ✓ Implement the temporary HTTP server used during setup mode. Define expected inputs.
- ✓ Implement the setup process.
- ✓ Implement the standard operation process.
- ✓ Figure out how to determine how much battery and signal strength.
- ✓ Implement the handling of commands received from the server.
- ✓ Implement persistent storage for use with certain variables.
- ✓ Implement firmware updating over a network.
- ✓ Implement failsafe to allow the user to factory reset the device in weird circumstances.

User Interface related tasks:

- ✓ Analyze the mockups provided by Home Re:Stock. Create more if needed
- ✓ Begin implementing skeletons for the various screens.
- ✓ Implement the account creation screen(s)
- ✓ Implement the account login screen
- ✓ Implement the forgot password screen
- ✓ Implement the 'Home screen' (Sensor list view screen)
- ✓ Implement the register new sensor screen(s)
- ✓ Implement the screen that appears when you select an individual sensor.
  - ✓ Be able to view data, issue commands, and edit preferences from this screen
- ✓ Implement the user preferences screen.
- ➢ Begin Product and Retailer integration

4.5 Schedule

Tasks	Started on	Completed on
User Interface Tasks		
Set up React development environment	01/20/21	01/25/21
Implement session variables (authContext)	03/10/21	03/15/21
Review / Create / Improve mockups	01/26/21	Continuous
Implement app Header	01/26/21	01/31/21
Implement standard styling sheet	01/26/21	Continuous
Implement account creation screen	01/26/21	02/26/21
Implement login screen	01/26/21	02/26/21
Implement "Home" screen	01/31/21	04/24/21
Obtain Sensor Data from Backend	04/01/21	04/24/21
Implement single sensor view	04/17/21	04/20/21
Implement reset password process	01/26/21	02/26/21
Implement new sensor setup	03/02/21	03/31/21
Implement navigation tree	03/02/21	03/05/21
Implement Systems/Preferences screen	03/08/21	04/19/21
Implement app Footer and assign to relevant screens	03/08/21	04/19/21
Implement local storage	04/20/21	04/21/21
Implement quality of life features	04/20/21	04/22/21

Implement 'Cart' screen (Retailer related)	Future Work	
Implement 'Quick Glance' screen (Retailer related)	Future Work	
Backend / Server Tasks		
Set up development environment	01/17/21	01/28/21
Document and create skeletons for all routes	01/20/21	01/28/21
Figure out DB storage of sensor data	02/04/21	02/04/21
Implement sentry error logging	02/23/21	02/23/21
Implement input validation for all routes	03/12/21	03/14/21
Implement rolling data logging for sensors	04/18/21	04/19/21
Implement send email functionality	02/03/21	02/04/21
Implement create new account + email verification process	01/20/21	01/20/21
Implement account login	01/20/21	01/20/21
Implement account deletion	01/21/21	01/23/21
Implement sync user preferences (get)	02/15/21	02/16/21
Implement change user preferences (post)	02/15/21	02/16/21
Implement the ability to command a sensor	02/03/21	02/04/21
Implement password reset process	02/12/21	02/14/21
Implement retrieve sensor data	02/05/21	02/06/21
Implement retrieve all sensor data	04/14/21	04/14/21
Implement change sensor preferences	04/20/21	04/20/21
Implement retrieve HTTPS fingerprint	01/25/21	01/27/21
Implement provision sensor ID	02/05/21	02/06/21
Implement sensor update status	02/06/21	02/06/21
Implement retrieve issued commands	02/04/21	02/14/21
Retailer Integration	Future Work	
Sensor Hardware Tasks		
Make the sensor as thin as possible	03/09/21	03/09/21
Incorporate lid clips into the design	03/10/21	03/11/21
Design a small sensor variant	03/23/21	03/24/21
Design a medium sensor variant	03/09/21	03/11/21
Create a working prototype of one of the variants	03/12/21	03/13/21
Design a large sensor variant	Future Work	
Power draw analysis	Future Work	
Sensor Firmware Tasks		
Implement boot into setup vs standard operation mode	01/28/21	01/30/21
Implement temporary network creation during setup mode	01/28/21	01/30/21
Implement setup process	01/30/21	02/02/21
Figure out how to determine battery life and signal strength	03/08/21	03/08/21
Implement automatic normal operation functions	02/26/21	02/28/21
Implement forced operation functions	03/11/21	03/13/21

Here is a more detailed breakdown of the tasks. Includes comments and contributors

4.6 Deliverables

- **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.
- **Sensor software:** The C++ source code that will run on all of the sensors.
- **User interface:** The source code for the Home Re:Stock React-Native application
- **Backend server code:** The Node JS code for the Home Re:Stock backend server.
- **Database scheme and initial data:** MongoDB Schema & Data from testing
- **Final Report:** This document. (Also incorporates the Software and Hardware design overview documents mentioned in the 'Capstone 1 - Final Proposal' document.)

5.0 Key Personnel

Brandon Russell – A senior Computer Science major in the CSCE Department at the University of Arkansas. They have experience developing and managing complex websites and server-side applications. The most notable of these are the "Bots on Discord" bot list and "Mirai Bot" Discord bot. Their experience from these will help with the planning and development of the backend for Home Re:Stock. Brandon focused on designing and building the backend API. They also worked on designing and integrating the database.

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. Cody worked on various parts of the user interface, namely the 'log in,' and 'account creation,' screens. He also designed the app header and fixed visual bugs throughout the app.

Daniel Rowett - Rowett is a senior Computer Science major in the CSCE department at the University of Arkansas. Some of the relevant electives he has taken include: 'Big Data Analytics & Management,' 'System Synthesis & Modelling,' and 'Computer Networks'. Daniel worked primarily on improving the provided prototype sensor's physical design, as well as writing the sensor's firmware from scratch. He also worked on the user interface and backend where needed, acting as the project's general quality control or coordinator.

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. Alex worked on the forgot password section which was a part of the login, as well as the settings screen, and the footer buttons that appear at the bottom of most of the screens.

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. Sam worked on the 'Register Sensor' screen in the user interface.

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 assisted him in contributing to Home Re:Stock. Jackson's main contributions were identifying and setting up a technology solution for the Mobile Application. Once development was underway, he built an efficient navigation tree so the other developers had access to methods to navigate the user between different screens, as well as simple means of getting data back and forth between screens. Ultimately, Jackson was in charge of getting the Home Screen implemented.

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. He has always been interested in entrepreneuristic ideas and has a strong talent for producing eye-catching designs.

6.0 Facilities and Equipment

- **Computer**: A computer is necessary for running development IDEs, working on reports, and participating in the project as a whole.
- **Autodesk Inventor (Student)**: This software enables the design of 3D models and subsequent conversion to printable formats such as STL.
- **Arduino IDE**: Arduino IDE is an easy to use and open source software that enables coding for, and uploading to arduino based hardware.
- **3D printer**: Used to print the physical chassis of any prototype sensors.
- **Electrical tools**: A soldering iron and other electrical tools are required for the hardware parts of the project to modify and analyze the prototype.
- **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.
- **WeMos Mini D1**: A NodeMcu Lua based device utilizing the ESP8266 Wifi development board. 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 our end of semester prototype.

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*