Multi-Context Shopping Optimization

Design Document

Team Number: sdMay20-23

Clients/Advisors: Goce Trajcevski

Team Members:

Max Garton
Ethan Shoemaker
Jesrik Gomez
Karla Montoya
Nate Wernimont
Arnoldo Montoya-Gamez

Team Email: sdmay20-23@iastate.edu

Website: http://sdmay20-23.sd.ece.iastate.edu/

Revised: December 8, 2019 / v3

Executive Summary

For a majority of shoppers, a considerable amount of time and effort is spent on activities related to shopping. We have identified three activities that can be optimized: maintaining shopping lists, deciding which stores to shop at, and deciding the order or route to take while shopping. Using microservices that combine inventory data from users' homes provided by sensors with nearby stores and maps services, we aim to create a system that saves shoppers time, money and effort.

Development Standards & Practices Used

The following development standards and practices apply to this project:
Electronic Circuits:

- Safety
- Reliability
- Durability
- Efficiency

Software Practices:

- Agile development
- Lean development - fail-fast

Engineering Standards:

- Modularity
- Reliability
- Scalability
- Performance

Summary of Requirements

Functional Requirements

- A device that measures and reports inventory status of household goods - such as milk - to a remote server
- The remote server tracks the status of inventory for each user
- A remote server computes the optimal destinations and routes for shopping given a variety of constraints - price, time, location, convenience, delivery speed
- An Android application allows users to retrieve optimal shopping routes from the remote server
    - The Android application allows users to add or remove - and adjust quantity - of additional items to their personal shopping list

Environmental Requirements

- A device measuring and reporting inventory status would require wifi in the user's home - and potentially in their fridge or other storage place

- The device requires a power source
- The device must be able to operate in a refrigerator - down to 0 degrees Celsius

Economic Requirements

- Microcontrollers, sensors, power sources and housings for the inventory measuring/reporting device
- Two central servers, one for data collection and one for recommendations, will be needed
- Smart phones - Android devices - will be needed to use the mobile app.

Applicable Courses from Iowa State University Curriculum

- COM S 227: Object-Oriented Design
- COM S 228 & COM S 311: Data Structures and Algorithm Efficiency
- COM S 309: Software Development Practices
- S E 339: Software Architecture
- S E 319: User Interfaces
- CPR E 288: Sensor Applications and Embedded Systems
- CPR E 388: Android Application Development
- ENGL 314: Reporting, Documenting, and Technical Communication
- STAT 330: Introduction to Statistics
- CPR E 308: Operating System basics, Inter-Process Communication, and File Management
- COM S 363: Database Management
- S E 329: Software Project Management
- CPR E 430: Network Protocols and Security
- CPR E 489: Data Communication and Client-Server Paradigms

New Skills & Knowledge Acquired From the Project

We expect to learn and research the following knowledge areas in order to be successful in this project:

- System architecture design
- Cloud computing
- Full-stack development
- Hardware design - enclosing a circuit or microcontroller into a complete package
- Requirements development

Table of Contents

List of Figures

List of Tables

1 Introduction

Below we will acknowledge external contributions and introduce the main idea of the project.

1.1 Acknowledgement

The team thanks the Iowa State University department of Electrical and Computer Engineering for giving us a resources, guidance and expert consultation. We appreciate the Electronic Technology Group for providing us with our team website server and hardware components for the project. Thank you to Goce Trajcevksi for meeting with us weekly to give us guidance and advice.

1.2 Problem Statement

Consumers are presented with many different ways to shop - going to a favorite store, going to the closest store, or going to the store where an item's price is lowest, taking the most efficient route to reach several stores. How can a consumer know that the choice they have made is the best choice? How can we provide an optimal shopping recommendation to a consumer using a combination of contextual information such as the user's current inventory, the user's current location, the user's shopping list, stores in proximity to the user, hours of the stores, and the stores' availability and prices for items?

Our solution is to develop a system that utilizes microcontrollers to automatically monitor the quantity of items that a user has in their household. The microcontrollers would communicate the data to a remote server that tracks the user's inventory of several items. This inventory data would be accessible and modifiable via a smartphone application. The user would also be able to utilize this application to maintain a shopping list - which is also automatically curated based on detected low inventory values in their household. The app could then provide the user with recommendations based on the user's current location, inventory data, the supply of items at nearby stores, and other contextual information to suggest an optimal way for the user to shop. Figure 1 in section 1.5 illustrates these identified use cases.

1.3 Operational Environment

The microcontroller device that measures the user's inventory is expected to be stored within range of a WiFi connection - with internet service. If the device is stored inside a refrigerator, this may block or severely limit the connection. The device will also need a power source - either via a powered USB port or a power outlet - to function.

We will rely on the user to have an Android device that is WiFi enabled. For initial setup, the Android device must be in range for a WiFi connection to the sensor module, and there must be

a WiFi network for the setup process. After completing setup, we assume that the user's Android device has a reliable internet connection through either WiFi or mobile data.

We plan to deploy our remote server code to cloud resources from Microsoft Azure. We will simply create a virtual machine (VM) and run our various services from it. We intend to use a Linux VM from Azure due to the friendlier development environment of Linux. Additionally, a number of members on our team are more accustomed to developing in UNIX environments. For the databases, we will be using Azure's SQL Database implementation.

1.4 Requirements

Functional Requirements

- A microcontroller measures and reports inventory status of household goods - such as milk - to a remote server
- The remote server tracks the status of inventory for each user
- A remote server computes the optimal destinations and routes for shopping given a variety of constraints - price, time, location, convenience, delivery speed
- An Android application allows users to retrieve optimal shopping routes from the remote server
    - The Android application allows users to add or remove - and adjust quantity - of additional items to their personal shopping list

Environmental Requirements

- A device measuring and reporting inventory status would require wifi in the user's home - and potentially in their fridge or other storage place
- The device requires a power source
- The device must be able to operate in a refrigerator - down to 0 degrees Celsius

Economic Requirements

- Microcontrollers, sensors, power sources and housings for the inventory measuring/reporting device
- Smart phones - Android devices - will be needed to use the mobile app.

1.5 Intended Users and Uses

Intended User

This solution is intended to be used by any consumer that shops regularly. This could include:

- Families in an urban and suburban setting within 20 miles of a store
- Members of apartments who share kitchen and other household supplies

Intended Uses

The diagram below illustrates the core use cases for our project. These are included in the minimum viable product. If time allows, we will add additional features such as extending the

shopping recommendations to families with multiple shoppers. We plan to update this document to reflect any such changes.

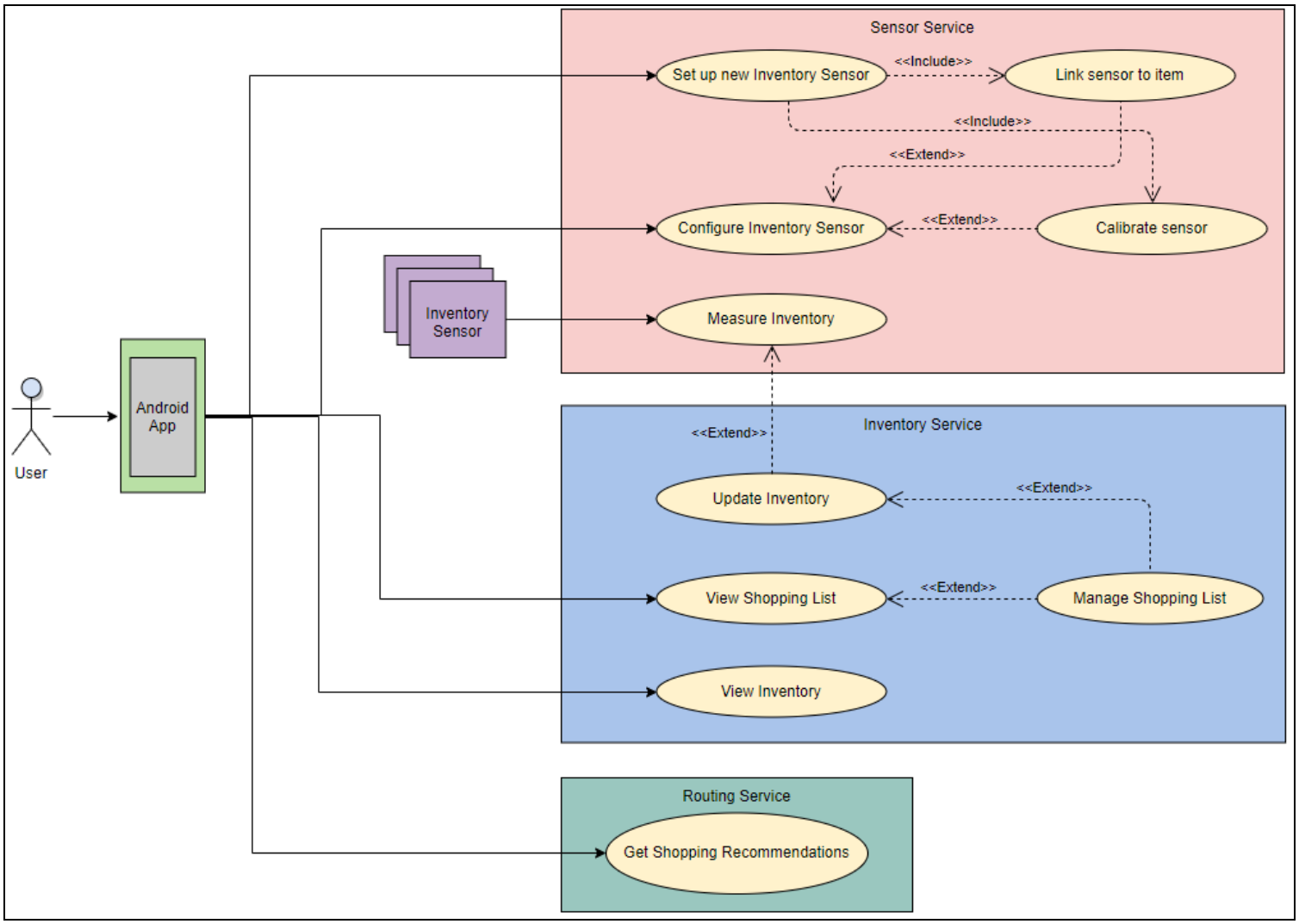

Figure 1: Use Case Diagram

The use cases are further described below from the perspective of our user persona, Mr. Smith:

1. Mr. Smith sets up a sensor module to add it to his account
    a. Mr. Smith opens the Android application and opens the “add a sensor” menu
    b. The Android application automatically locates the new sensor module and prompts Mr. Smith to enter the Wifi SSID and passkey to join
    c. After joining the network, Mr. Smith is prompted to configure a high, low and empty weight setting on the sensor module from the application
2. Mr. Smith views the status of his inventory
    a. He opens the Android application and logs into his account
    b. He can see his current inventory amounts, such as:

        i. Milk - full
        ii. Eggs - 50% remaining
        iii. Detergent - 10% remaining
3. Mr. Smith views and modifies his shopping list
    a. He opens the Android application and views his current shopping list
    b. He adds another item to the shopping list
    c. He updates the quantity needed of an item on the shopping list
    d. He removes an item from the shopping list
4. The inventory sensing module tracks Mr. Smith's inventory of items
    a. A sensor module reading is sent to the remote inventory server
    b. The inventory server identifies this as a low amount
    c. The server updates the stored inventory value for that item
    d. The server adds the item to Mr. Smith's shopping list if the low value is below Mr. Smith's set threshold
5. Mr. Smith gets shopping recommendations
    a. He opens the Android application and opens the "start shopping" menu
    b. He establishes his shopping criteria within the app:
        i. Store location - maximum distance away
        ii. Prioritization of price and time - via a slider
    c. He submits the request for shopping recommendations

1.6 Assumptions and Limitations

In order to narrow the scope of the problem statement, we introduce the following assumptions and limitations:

Assumptions

The design decisions described in this document are built upon the following assumptions:

The user's home has wifi

For the project, we are assuming the refrigerator scale has a suitable WiFi connection. Without it being connected to WiFi, the inventory sensor's data cannot be used to make shopping recommendations. All of the use cases in section 1.5 build from this assumption.

The user owns a GPS and Data enabled Android device

Because our whole project wraps around the idea of sending notifications to users, we will not be able to do this unless our user owns an Android device with the required technology. All of our use cases rely on this assumption because an Android device would be required to set up a sensor device - use case 1.

Availability of external data:

Our proposed solution to the problem statement relies on the availability of third party data - more specifically, the inventory data of stores. This can be mitigated by doing initial research on the APIs available for public use and looking into those of popular grocery stores. If we are unable to find public APIs that give us the data we need, we may need to mock it in order to further our proof of concept. Use case 5, receiving shopping recommendations, depends heavily on this assumption; a store is not easily recommended without insight into the products that are available at the store.

Limitations

We have established the following limitations for the project:

The product shall not be too expensive.

The sensor devices - in use case 1 - should have a reasonable cost of less than $25 per sensor for the finalized product. The servers used for the project - for all use cases - should be of minimal cost.

The product should have minimal impact on WiFi throughput.

The sensor devices should only send data over WiFi and the internet when necessary - ie. when the inventory has changed or during initial setup, in use cases 1 through 4.

The product should not overuse the user's Android device's hardware - battery, RAM, CPU, storage, WiFi.

Any algorithmic computation - such as recommending shopping routes in use case 5 - should be done on a server and not on the user's device. This is to ensure that the user interface on the Android application is responsive for the user.

The only user interfaces of the project will be within the Android application.

Because Android is the most common mobile operating system, we will develop our mobile application on Android first. If time allows, a web application or iPhone application with the user interface may be developed. The Android application will be involved in use cases 1, 2, 3, and 5 in section 1.5.

1.7 Expected End Product and Deliverables

The final product delivery will be split into a proof-of-concept, minimum viable product, and a final product deliverable. The proof-of-concept and minimum viable product (MVP) will be

delivered by the end of the fall 2019 semester. The finalized product will be delivered at the end of the spring 2020 semester. An overall design manual of the product and its architecture will also be provided for any future development.

1. Finalized System Design - December 1, 2019
    a. The proof of concept prototype will have the majority of our design decisions finalized. It will also demonstrate the feasibility of such a system to measure and track the supply of household items, communicate the data to a server, show the data to a user on a mobile device, and recommend an optimal way to shop based on the given data and user inputs. At this point in the project, the system architecture as well as the tools and technologies used to construct it have been researched and selected.
2. Minimum Viable Product - March 15, 2020
    a. The minimum viable product is the overall system connected, where each component listed in the proof of concept is completed and integrated so that the system functions. Functionality may be limited at this point - the recommendation may not be based on all information or the user's input.
3. Finalized Product - April 15, 2020
    a. The finalized product would have complete functionality, supporting all use cases described. The recommendation feature is extended to take into account other user's - family members - locations and shopping lists, optimizing the shopping experience for a whole household instead of one user.

2. Specifications and Analysis

Below we will discuss how we will solve the problem that we have described above.

2.1 Proposed Design

- Our current approach includes using simple analog sensors that can be interfaced to a widely available microcontroller in order to monitor the current contents in a user's home
    - The current plan is to use a Raspberry Pi as our microcontroller, and weight sensors to weigh products in the user's home.
    - Since we are using an analog pressure sensor, this will require the user to calibrate an "empty" and "full" weight for the item being measured by the system.
- The microcontroller will connect to wifi, and will interface directly with our server, sending it the weight of tracked items a user has in their home.
    - The Raspberry Pi will communicate with the remote server via HTTPS requests to guarantee confidentiality, integrity, and authenticity
- After our server receives product weight data, if the quantity is below a certain threshold we will notify our users they are running low.

- There will be an Android application that allows the user to manually input products that they need to shop for
    - The application displays the current shopping list, which may or may not include items that were automatically added based on the sensor data
    - The user can add or remove items to the list, or update quantities needed.
- Our database will be a simple SQL server instance and our data will be stored in simple key-value pair format.
- The Android application will allow users to view optimal routes for what to purchase.
    - The optimal routes are computed by remote server and displayed on the application's user interface.
    - The user can choose the extent to which time and money savings should be prioritized when browsing the nearby stores.
    - These optimal routes should be "perfect", meaning that they give the ideal answer, and they should have adjustments for the tradeoff between time and money
        - For example, it might be more expensive to buy groceries from the same store, but it saves a lot of time.

2.2 Design Analysis

Our proposed solution with the plan for each component explained in section 2.4, will address our problem statement. At this point in the project, everything is based on research, hence the details of each component or interfaces within the system are subject to change as we prototype and improve on the current design. The current design's strengths are its simplicity and low cost - minimal hardware costs and virtually no software costs. However, the design's two biggest weaknesses are that it may not be user friendly - especially the setup process when a user needs to register a new sensor module - and that it might have limited functionality for recommendations - which will be built more in the second semester of the project. We plan to iterate on these issues and further address them during the implementation phase of our project. This means that the finalized designs may be completely different than the prototype designs that we have documented as of yet. For example, we may use a smaller, lower power microcontroller - the current design uses a full-sized raspberry pi - in a later design.

Our proposed solution will take advantage of existing security protocols, concepts, and paradigms to ensure the security of our system. This includes encrypted communication; encrypting customer sensitive information, both in transit and at rest; preventing people from using a man in the middle attack with our devices; preventing tampering with devices that collect data from within user's homes; and securing our databases. Part of our focus during the second semester will be on securing the communication between components in the system. The scope of our project does not include the invention of any new security protocols or concepts

2.3 Development Process

In the fall semester, we have been using an Agile development process. We have continually made small adjustments to our schedule and weekly plans in order to reflect the decisions we are making and the research we are conducting. We also plan to follow an Agile approach in the spring semester, although it may be more rigid - we will already have a fairly solid idea of the specifications and details of the implementation by then. However, we anticipate that there will still be changes and adjustments to the scope and schedule, like any large software development project.

2.4 Design Plan

The team has designed the system illustrated by Figure 2 below which consists of 3 major components: the inventory sensor, the remote server and the Android application.

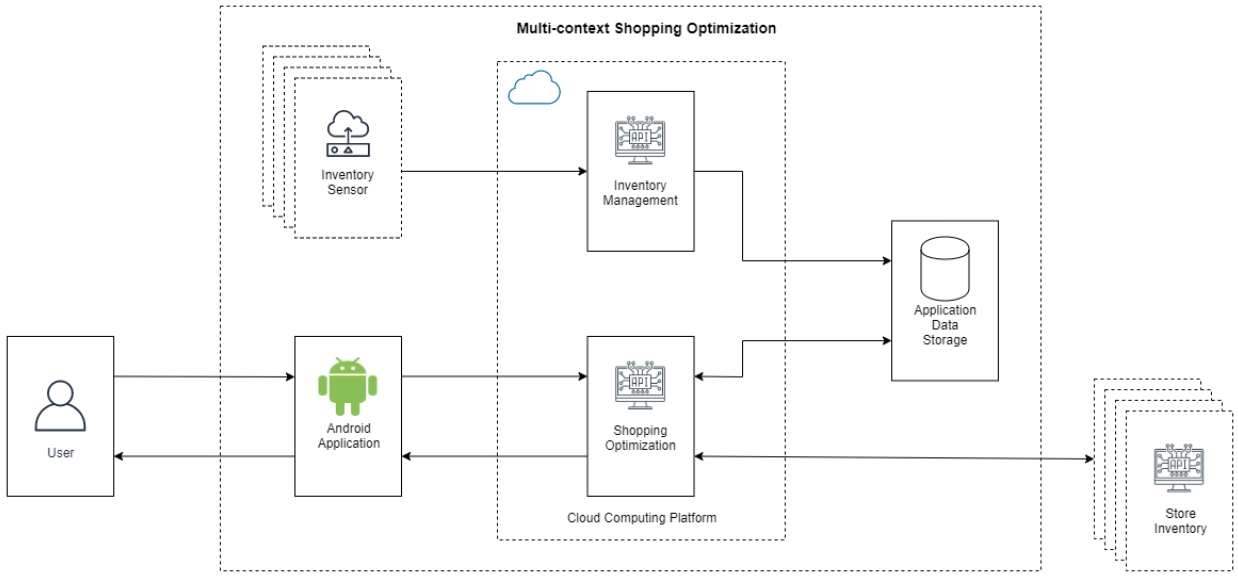

Figure 2: System Component Diagram

Following will be an elaboration on the design and operation of the different components of the system as well as the main use cases for each of them.

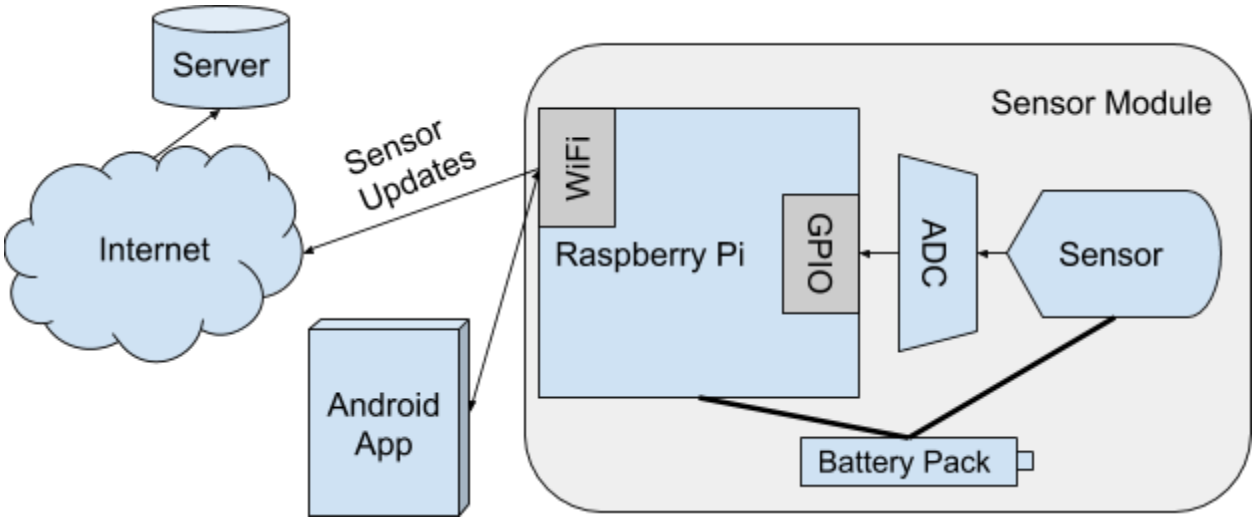

Figure 3: Sensor Data Collection Architecture

The use cases involving the inventory sensor are described below:

- Inventory data collection
    - The weight sensor's value is an analog value - just a voltage that varies with weight, so it must be converted to a signal that can be interpreted by a microcontroller
    - The weight sensor is connected in series with the battery pack and the ADC to be converted to a digital value that the microcontroller can process
    - The Raspberry Pi then sends these values to be stored in the remote database
- Setup of a sensor
    - The user will power on their sensor module
    - The user connects their Android phone to the WiFi network that the sensor will operate on
    - The user uses WiFi to connect with their new sensor module's private network and this allows the user to connect their sensor module with the home WiFi
    - The user then assigns an item name that this sensor will be used to track
- Data transmission method
    - Every 60 minutes - regardless of any change detected, the sensor module will send a sensor value update to the server
    - If the sensor module detects a change, it will send an instantaneous update to the server

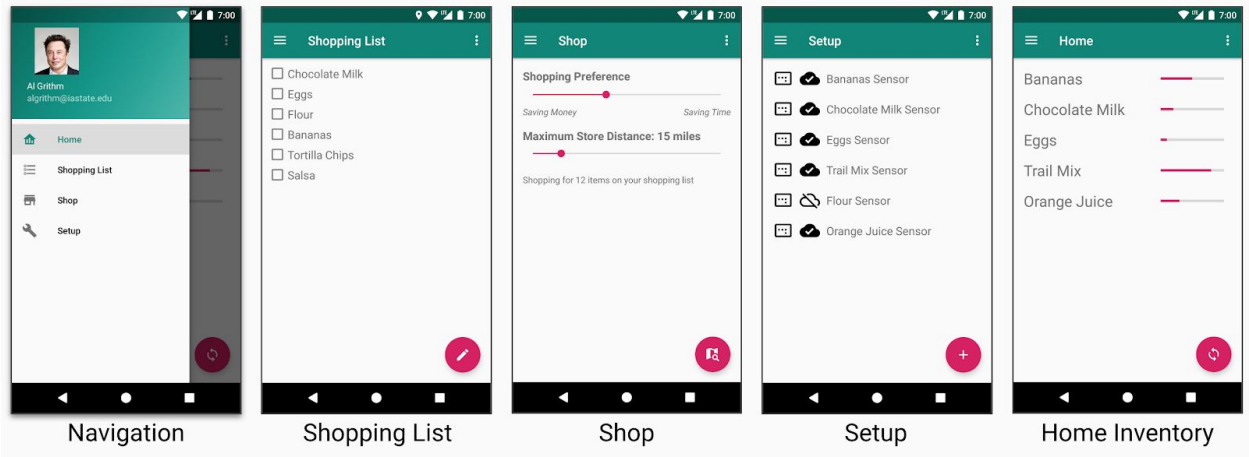

Figure 4: Android Application mockups

- Login Activity (not pictured)
    - The user will first need to login before using the application by entering a username and password
    - If the user has logged in in the past, the device will store a Cookie, which it will use to automatically login every time.
    - optional - The user can manually log out to prevent automatic login on opening the app.
- Home Inventory Activity
    - The user can view their inventory of items, indicated by measurements from inventory sensors tied to their account
- Shopping List Activity
    - Fetches and displays the user's shopping list from the remote server.
    - The shopping list can be edited:
        - Adding items
        - Removing items
        - Updating the quantity of items
    - Once the user exits this activity, the remote server will be updated with the latest list.
    - If the user wants to start shopping, they will select the "start shopping" button from here.
- Setup Activity
    - The user can view all of the sensors tied to their account, along with their status (online or offline).
    - They can modify a sensor, either calibrating it or changing the item it measures.
    - They can add a new sensor to their account, which includes linking it to their account, tying it to an item, and calibrating it (as shown in Figure 1).

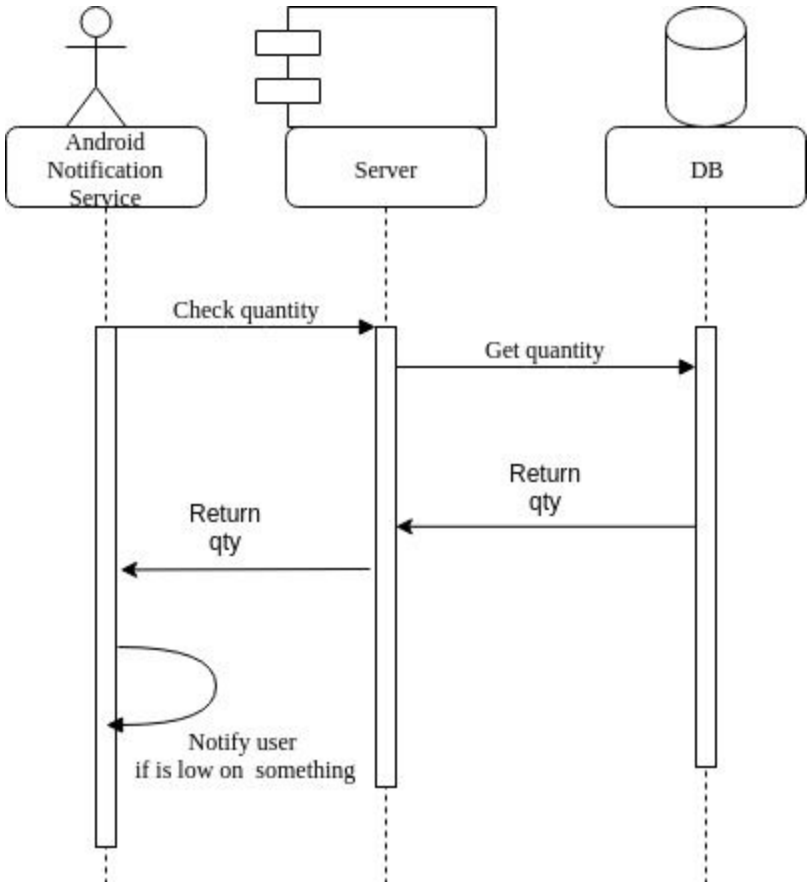

Figure 5: Notification Service Sequence Diagram

As part of the use case for viewing the user's inventory, we have designed a notification that the user will receive when an inventory sensor detects that they are running low on a given item. The "Running Low" Android notification is described below; it is an extension of use case 4d described in section 1.5.

- The Android App will notify the user if the measured inventory amount of an item falls below a set threshold.
- If the user is notified, the user can simply click on the notification, which will then open the app.
- The process handling the notification will be a background process, or service, which will periodically - about every 30 minutes - check the quantity of all items. Then, if an item that was not already low runs low, the user will be notified.
- This process of doing a fast, simple check periodically is lightweight, so it will not utilize more device resources than necessary. It should result in minimal battery use.

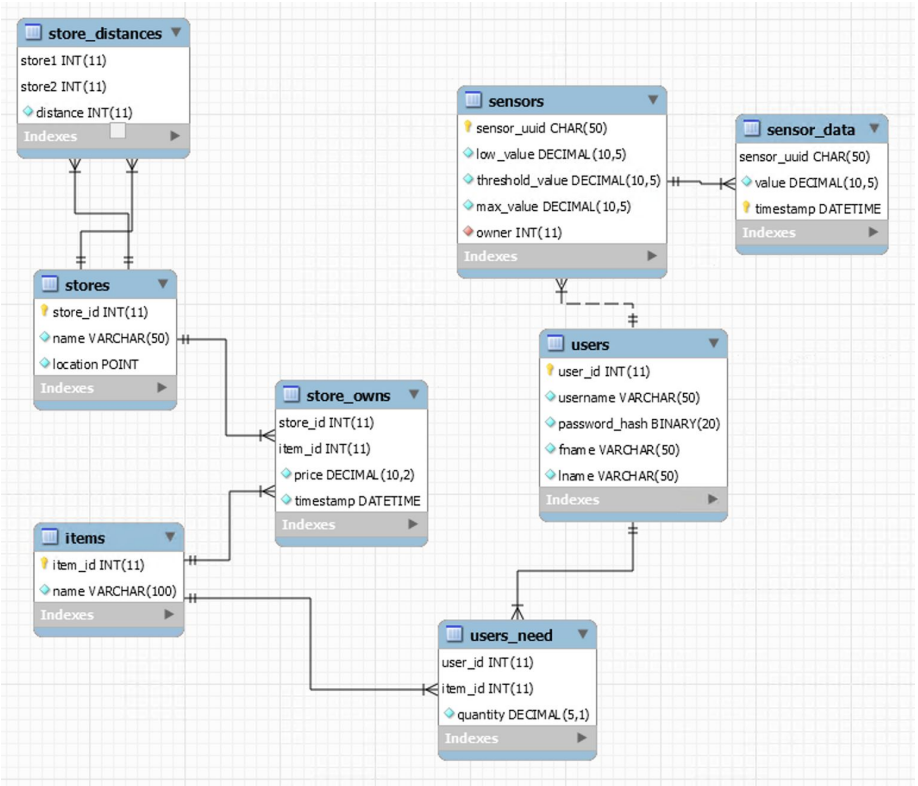

Figure 6: Database Schema Diagram

Each database table in Figure 6 is described below:

- Store_distances
  - Database: `routing`
  - Purpose: Serve as a cache for distances between stores. Stores do not move, so there is no reason to go to the third party API every time the routing algorithm needs this information.
- Stores
  - Database: `mock_stores`
  - Purpose: Mock store information, so we can verify functionality of other components before integrating with third party APIs.
- Items
  - Database: `items`
  - Purpose: Store items with their name. This will allow relations between sensors and what item they are tracking, as well as users and the items they need. It standardizes items across users.
- Store_owns
  - Database: `mock_stores`
  - Purpose: Relate stores to what items they sell and for what price
- Users_need
  - Database: `shopping`
  - Purpose: Relate users to what items they need and in what quantity

- Users
    - Database: `users`
    - Purpose: Store user information and metadata. This will also enable authentication because it stores user passwords.
- Sensors
    - Database: `sensors`
    - Purpose: Stores configuration information for a given sensor UUID, such as what is considered a “max” or full weight for an item, what is considered a “low” or empty weight for the item, and what is considered a “threshold” weight, which is the weight that we add the item to the user’s shopping list. Additionally, this table tracks who owns the sensor.
- Sensor_data
    - Database: `sensors`
    - Purpose: Stores the sensor data reports and timestamps them.

If possible, we will obtain inventory data from APIs provided by grocery stores. We have found that both Walmart and Target have APIs that provide the necessary data, but they do not allow public use. We plan to mock stores’ inventory data in the likely scenario that we are unable to utilize existing APIs.

We plan to route users to store locations using directions provided by the Bing Maps API. We also considered using the Google Directions API, but the service requires a paid subscription. Bing Maps allows our service to make 50,000 transactions per day in the free tier, so long as we are considered an Education use case. Due to this free tier, we will rely on the Bing Maps API. Additionally, Bing Maps is capable of routing from an initial start location to a final destination through multiple waypoints while factoring in traffic conditions. These exact capabilities align with the needs of the project, so Bing Maps will be sufficient for our project.

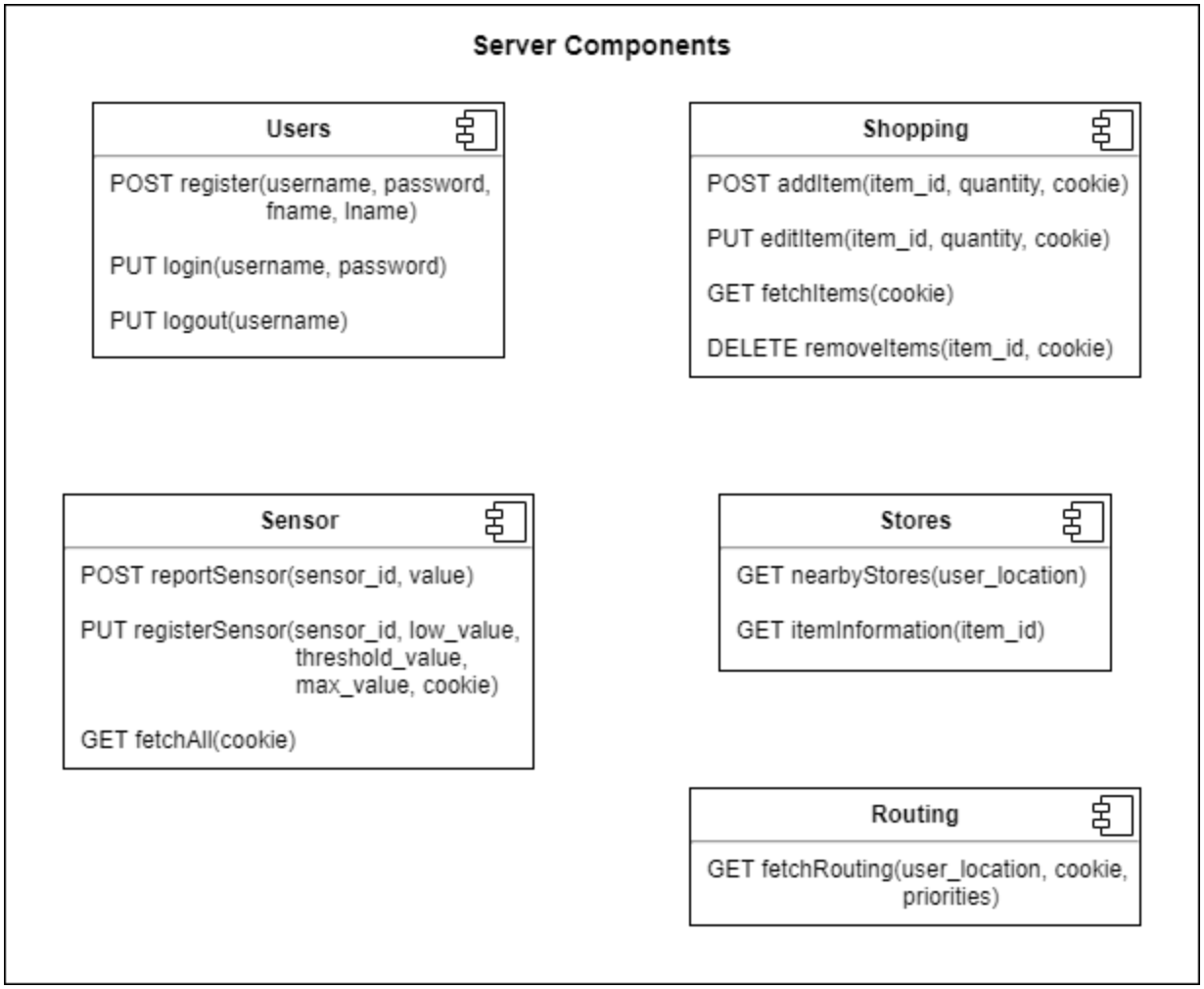

Figure 7: Backend Services

3. Statement of Work

Below we will discuss our process, goals, and tasks for the undertaking of our project.

3.1 Previous Work And Literature

A previous senior design team at Iowa State University worked on creating a “smart bin” device that measures the quantity of an item based on the measured weight, combined with software that helps businesses restock products more efficiently [1]. While this project aims to solve a similar problem, the user that we are targeting is a general consumer or shopper, rather than a business. The project had a lower cost than our anticipated cost - their project required $80 minimum purchase of microcontrollers and sensor modules, while our anticipated cost is under $250. In addition, the Multi-Context Shopping Optimization project includes the possibility of extending the optimization to multiple shoppers, which differentiates it from past projects. A

document detailing Sequence-Aware Recommender Systems states that it will discuss some relevant topics to any Recommender Systems; however, these are typically geared towards more general scenarios where the scope of the project or system is a bit larger than what our system is designed to do. Such things include user-item pairing matrices to predict other user-item ratings and session-based recommendations. These do not apply to our project since we are not strictly speaking developing recommendations for users based on their habits. We are providing recommendations based on factual information for which an objectively optimal selection can be chosen - ie the store that is closest and has the cheapest price [2].

3.2 Technology Considerations

Strengths and weaknesses different technologies that may be used for the project are listed below. Bolded technologies indicate that the team chose to utilize them.

- Microcontroller
  We considered the following options:
  - **Raspberry Pi 4 and Raspberry Pi Zero W**
    - Strengths: The team has experience with projects using it. It is well documented and supported. It allows flexibility in programming languages. The board has all of the
    - Weaknesses: It is expensive compared to other microcontroller options. It has high power consumption and features that would not be used by this solution.
  - Other Linux-based microcontrollers
    - Strengths: Most raspberry pi alternatives are cheaper than the raspberry pi. They also allow flexibility in programming languages.
    - Weaknesses: These boards are not documented or supported as much as the raspberry pi or arduino.
  - Arduino Nano IOT
    - Strengths: The team has experience with projects using it. It is well supported and documented. It costs less than most other microcontrollers
    - Weaknesses: It requires us to write programs in C. The board may be tied to Arduino's IOT platform - it may not allow us to easily connect to our own services.

  We chose to use the Raspberry Pi 4 and Raspberry Pi Zero W due to their flexibility in development. We plan to build sensor module prototypes using the larger Raspberry Pi 4, and then we will build later prototypes - closer to the finalized design - using the smaller, more power efficient Raspberry Pi Zero W.

- User Inventory Sensing
  The following were considered as options for sensing the quantity of the user's items

- **Weight Sensor**
  - Strengths: Applies to most items a user would want to track since the item's weight is most often proportional to the amount remaining
  - Weaknesses: Objects with a low product weight to container weight ratio may not be able to be sensed accurately or correctly
- Infrared or SONAR distance sensor
  - Strengths: Can be used on objects with a large or irregular base platform since it does not need to be underneath the object
  - Weaknesses: Distance sensors often require calibration specific to each instance. This would most likely require the user to compromise the container of the object in order for it to sense the "level" of the product. Only applies to products that are fluid-like and have a "level" such as milk, juice, or even sugar and salt.
- Digital Image Processing
  - Strengths: One camera or sensor can be used to track multiple items
  - Weaknesses: Algorithms are incredibly difficult to fine tune. The camera needs to be placed somewhere distant enough from the product to observe the level. Only applies to products that are fluid-like and have a "level" such as milk, juice, or even sugar and salt.

We elected to use the weight sensor as it can be used for a wide variety of products and in an effective, predictable manner with minimal calibration or software sophistication required. It is also the cheapest of the 3 options considered. This sensor is a proof of concept and is subject to revision as we implement the project.

- Web Server
  - **NodeJS**
    - Strengths: Ideal for efficient data-intensive real-time applications that run across distributed devices. The runtime environment is lightweight and efficient, which will allow us to satisfy our scalability requirements. The team has experience developing web services with NodeJS. It supports open source libraries.
    - Weaknesses: NodeJS is not ideal for CPU-intensive operations due to its single-thread non-blocking I/O design. Heavy package dependencies.
  - Flask - Python
    - Strengths: Can handle heavy server-side CPU-intensive operations. Mature and stable open source libraries.
    - Weaknesses: None of the team members have developed web services with Flask, and are not as experienced with Python.

We decided to build the application's REST API with NodeJS because the vast majority of the operations will be small and fast; in other words, we expect a large volume of simple constant time operations, as opposed to less frequent complex operations. It is

also highly scalable, and our team has experience developing web services with this runtime environment.

- Mobile Application
  We considered two mobile platforms to develop a mobile application for:
  - **Android**
    - Strengths: The team has experience developing Android applications. Android is well-documented as a development platform and supports many open source libraries. As far as Android libraries being used, we are currently only planning on using internal Android Libraries.
    - Weaknesses: Supporting so many devices can be difficult, due to them having different hardware. Many Android devices are running older versions, which may require us to use old development libraries to ensure that a majority of users would have access to our application.
  - iOS
    - Strengths: It would allow more users to utilize this project's product.
    - Weaknesses: No one on the team has experience developing iOS applications.

  For our application, we have decided to use **Android**. The reasons are listed below.
  - Development tools: Android Studio is available for all major OSs - Linux, Mac, and Windows. Unlike Android studio, Xcode can only be executed by the Mac operating system, which would be needed in order to develop in IOS.
  - Test Devices: Most people in our team own Android devices, which makes it easier for us to test our application.
  - Experience: Everyone in the team has at least done some Android development, while others have even had internships related to Android Dev. Also, Android is written in Java, which everyone has experience with.

- Data Storage
  - **mySQL**
    - Strengths: Most of the team is familiar with relational databases and querying. It is also incredibly well supported.
    - Weaknesses: Constrains us to using relational data models.
  - Data Warehouse
    - Strengths: Could efficiently analyze large amounts of data for trends and performance
    - Weaknesses: Not geared towards the individual user; typically used to create performance evaluations
  - GraphDatabase
    - Strengths: Useful in finding relationships in your data or getting patterns that are geared towards your customer
    - Weaknesses: Not useful when your data is key-valued and tabular

For our application, we have decided to use mySQL. mySQL will satisfy our data storage requirements with its relational model architecture because we anticipate a large volume of small transactions. Furthermore, our team is familiar with mySQL database management system and it is heavily used in the industry. The database engine is incredibly mature, and there are many libraries available for interfacing with the databases. Additionally, the other storage methods considered created too much overhead for our data model. mySQL should allow straightforward and performant development for our backend services.

- Cloud Computing & Modularization
    - **Microsoft Azure**
        - Strengths: The team has experience using Azure to deploy microservices and web applications. Azure also has a free tier for services that will satisfy our needs for this project. It has many options for app hosting features. Azure is well documented with many tutorials for creating and deploying various web applications and services.
        - Weaknesses: Azure is not as compatible with open-source software as other cloud computing options.
    - Amazon Web Services - AWS
        - Strengths: AWS is more open source friendly than Microsoft Azure. AWS has many services and is a well documented platform.
        - Weaknesses: The team is not as familiar with AWS. We have previously encountered bugs in Aurora.
    - Docker
        - Strengths: The team has experience using Docker. There are also readily available container images.
        - Weaknesses: Can be somewhat confusing for anyone who has not used Docker.

We chose Azure because our team is more familiar with Azure. Additionally, we will be able to stay within the free tier. We chose to seek out a cloud based approach because other aspects of our project are already forced to be cloud-based (e.g. Bing Maps and store data); as such, we are already paying the penalty for network latency elsewhere. Additionally, we deemed the additional complexity of hosting our own solutions to be not worth the small performance benefit of making things ourselves.

3.3 Task Decomposition

The project is broken down into the following tasks:

- Sensor Module
    - Test sensor for precision and sensitivity and determine if calibration is needed

        - Ethan
    - Develop software to allow Raspberry Pi to connect to the user's WiFi
        - Ethan
        - Max
    - Develop program to read values from the sensor and send this data to the remote server
        - Max
    - Optional: create a housing module to hold the microcontroller, power source, sensor and any additional hardware
- Android application
    - Design app layout and navigation scheme
        - Karla
    - Create notification service
        - Arnoldo
    - Create network libraries specific to our application
        - Arnoldo
    - Create login activity
        - Karla
    - Create configuration activity
        - Karla
    - Create inventory activity
        - Karla
    - Create shopping list activity
        - Karla
    - Create shopping activity
        - Arnoldo
    - Fetch data from server, update each activity depending on server
        - Arnoldo
    - Update server with data from our app
        - Arnoldo
- Servers
    - Determine API that allows the Android app and sensor modules to acquire data
        - Jesrik
    - Develop functions for mySQL
        - Jesrik
        - Nate
    - Develop Users Service which will allow users to register, log in, and log out
        - Jesrik
    - Develop Shopping Service which will allow users to fetch, add, edit, and remove items from both, the shopping and inventory list
        - Jesrik
        - Nate
    - Develop Sensor Service to register, report, and fetch data from sensors

        - Jesrik
        - Max
        - Ethan
    - Develop Stores Service to find nearby stores and item information
        - Jesrik
        - Nate
    - Develop Routing Service according to user and store location, and also shopping priorities
        - Jesrik
        - Nate
- Databases
    - Construct relational model, tables, and fields for each table
        - Nate
    - Analyze necessary queries to establish needed indices
        - Jesrik
- Testing
    - Unit testing
    - Integration testing
        - Integrate database/servers
            - User Database / User Service
            - Shopping Database / Shopping service
            - Sensor Database / Sensor service
        - Integrate third party APIs and servers
            - Stores service / stores APIs (or mocked data)
            - Routing Service / mapping API, shopping service, stores service
        - Integrate servers/Android application
            - Users service / app
            - Shopping service / app
            - Routing Service / app
        - Integrate Raspberry Pi / Sensor Service
    - Functional testing
        - Functional tests for each core use case are described in Table 2 in section 5.3.

High-level testing tasks were outlined in this section. For more information about the testing plans for the project, refer to section 5.

3.4 Possible Risks And Risk Management

Possible risks and mitigation methods for this project are listed below:

- Security: sending user's data over the internet - and over WiFi - can present a risk that the user's data could be exposed to others.

    - This is mitigated by encrypting the data before sending it over the internet. This prevents the likelihood of someone sniffing the data and seeing sensitive user information.
- Authenticity: if attackers know the data format of the recommended stores, they can spoof the identity of the server and recommend non-existent or invalid stores to the user
    - This can be mitigated by encrypting the data and by using digital signatures to verify the integrity of the message as well as the identity of the source
- Integrity: Attackers may be able to determine the format of our requests and send fake requests to the server, potentially causing harm
    - This is mitigated by encrypting data so that 1) an attacker cannot easily determine the destination of the requests, 2) an attacker cannot easily reverse engineer the format of our requests, and 3) an attacker cannot send invalid requests to our server
- Availability: We expect our uptime to always be above 99.99%, which is the service level agreement for Microsoft Azure
    - This can be mitigated somewhat natively by backbone internet service provider routers, but additionally by blacklisting IP addresses that appear to spam our servers - and blocking unused ports.
- Scalability: We expect our system to be scalable for more than 100,000 concurrent users. A user is considered anyone who has at least one of our sensor modules installed or uses the app
    - This can be ensured by acquiring sufficient resources on the server and optimizing our use of the resources. When applicable, our algorithms for recommendation will be analyzed for optimum performance.

3.5 Project Proposed Milestones and Evaluation Criteria

Key milestones for the project will be based on the use cases or features listed below. Each milestone is completed by the verification of the use-case functionality, specified in section 5.3. After we have reached all of these milestones, we will focus on refining and improving non-functional requirements, such as security, reliability, performance, and scalability.

1. Installing and Setting up an Inventory Sensor Device
2. Viewing the status of a User's Inventory
3. Viewing and Modifying a User's Shopping List
4. Automatically Tracking a User's Inventory
5. Receiving Shopping Recommendations Upon the User's Request

3.6 Project Tracking Procedures

Our team will follow an Agile development process, with 14-day sprints. Gitlab will be heavily used to track the sprint progress via the issues board. Project milestones, such as the completion of the remote server APIs, will be included in the Gitlab repository to ensure that the

progress aligns with the planned schedule. This serves as a common place to view the status of work being done - and work that will be started soon. As tasks - Gitlab issues - are completed, team members are expected to document the work within the issue page so that the work can be revisited and understood in the future. Furthermore, we will also be adding more specific issues for bugs, features, and functions that need to be implemented when necessary.

Slack, the instant messaging platform, will be used frequently for communication between team members. This platform will provide the team with an efficient way to discuss concerns, ideas, logistics and clarifications amongst all members of the team or specific individuals.

Status reports will be completed and posted at regular intervals (every 10 days during the fall semester and every 14 days during the spring semester) on our team Senior Design website. Notes will be taken during team and client meetings and then stored in our shared Google Drive folder. This will ensure that decisions and discussions that take place during meetings are documented for future reference.

3.7 Expected Results and Validation

Expected Results

The desired outcome is that a user is able to download the Android application created during this project and install the inventory sensors in their home. The user will be informed of the measured inventory in their home, items will be automatically added to their shopping list, and they will be able to make better shopping decisions using shopping recommendations from the application. Additionally, the desired outcome includes a full user guide that allows a layperson to setup and use the system.

Validation

We plan to validate that each functional requirement and use case is met by conducting testing for each use case. The test plans and acceptance criteria are defined in section 5.3.

4. Project Timeline, Estimated Resources, and Challenges

We now provide a detailed timeline in the form of a gantt chart for the work pertaining to our project. One chart outlines the fall semester timeline while the other shows the spring semester timeline.

4.1 Project Timeline

The below diagram demonstrates the time frames associated with the various tasks as well as the person that will lead the effort for the given task. October is spent primarily planning for how and what data needs to be communicated. Starting in November, work begins on iterative design tasks, such as testing the sensors, the software on the Raspberry Pi, the inventory service, and the store service. We plan to connect the Raspberry Pi to the inventory service once both services have been sufficiently developed independently to be able to support being hooked up. Come second semester, the bulk of development will be underway including more thoroughly implementing the remote services, data sensing and logging. Android user interfaces will be realized by the minimum viable product deadline of March 15th. Each subsystem should ideally be fully functioning independently by early March so that the integration can be more completely tested. We expect to have all tasks done by late April, so that we can focus on preparing presentation documents prior to the end of the spring semester.

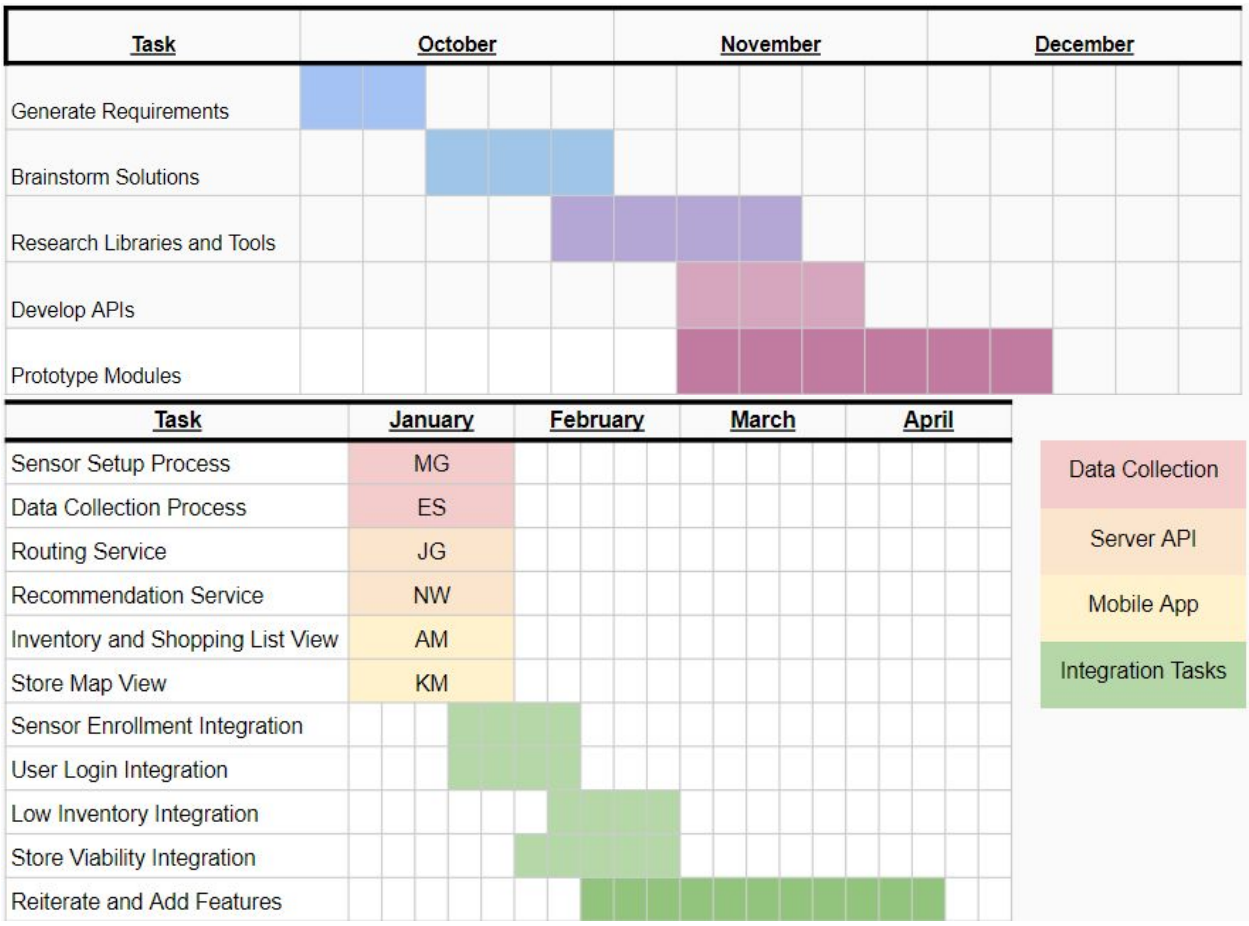

Figure 8: Project Timeline Gantt Chart

4.2 Feasibility Assessment

We expect that the core use cases outlined in section 1.5 will be possible to implement. We anticipate that some sources of data, such as inventory data from grocery stores, will not be available to us within the timeline of the project. To overcome this limitation, we will mock the store inventory data if necessary.

One challenge that may be encountered with the sensor subsystem is developing a program which can communicate with our Android app directly via a WiFi network broadcast directly from the Raspberry Pi such that it can connect to the user's home WiFi network. Other smart home devices use this method to connect to a user's network, namely the Google Home series and the Amazon Echo series.

The routing algorithm is seen as a significant challenge to our project. It has to route users, which could be viewed as a traveling salesperson problem, but then it also needs to conditionally go to certain stores. We can simplify the problem by narrowing the range of possibilities so that the algorithm does not need to consider every store. In order to choose which stores, it has to weigh the additional travel times of going to a store against the possible savings by going to that store. In order to view the possible savings of a store, there can be some subset of items that are cheaper at a given store, compared to another store. There are many variables involved. Once a "perfect" algorithm - that always gives the correct, exact answer - is developed, it will be possible to develop an approximate certain parts of the algorithm in order to lower the complexity of the problem from NP-hard to P. An additional option would be to make some sort of naive local hill climbing AI agent that could be expanded to have a higher chance of giving the user a "good" selection.

4.3 Personnel Effort Requirements

Task	**Description**	**Person-Hours and main contributor**
Testing the sensors	Observe the behavior of the sensors and determine if calibration is required per sensor	15 Ethan Shoemaker
Develop software to allow Raspberry Pi to connect to user's WiFi	Write a program which can programmatically connect to a user's WiFi at runtime	35 Max Garton
Develop program to read sensor values and send to remote server	Write a program on the Raspberry Pi which can interface with the sensors and send the data to the remote server for processing	40 Ethan Shoemaker (& Max Garton if needed)

Create Notification Service	Create a background service on our android app that will periodically check for changes in the quantities of our inventory. If we are below a certain threshold, we will notify the user. This part will require services, bound services, and notifications.	30 Arnoldo Montoya-Gamez
Create Network libraries on Android	Create utility libraries that will be used to communicate with the server. Whether that is to send data, receive data, or update data. All server communication should be done within this Util library.	5 Arnoldo Montoya-Gamex
Create Login Activity	Create initial activity that will be used to login. This activity will check if the user has logged in in the past. If not, we will ask user to login. We will possibly use “Login with Google” feature, depending on complexity.	25 Karla Montoya
Create Main Activity	This will be the main activity, consisting of three buttons, with each representing a different activity. The user can switch to the Inventory, Shopping List, or “Start Shopping” activities by clicking the corresponding buttons.	10 Karla Montoya
Create Inventory Activity	This activity will be the screen showing the current quantity of the tracked inventory. This activity will also give users the option to add new sensors.	20 Karla Montoya
Create Shopping List Activity	This activity will show the user’s current shopping list, and will have the option to “check” items off. This activity will depend heavily on network connections, because it may update, add, delete items from our checklist, which should be updated in the server as well.	20 Karla Montoya
Create Store and Price	This activity will show stores where	30

Activity	items on the user's shopping list can be purchased from, with the ability to sort by price or distance.	Arnoldo Montoya-Gamez
Fetch Data from Server to Android app	Ensure that every activity that needs server data fetches it correctly using the network util library.	10 Arnoldo Montoya-Gamez
Update server with data from our app	Ensure that every activity that updates the server does it correctly using the network util libraries.	10 Arnoldo Montoya-Gamez
Determine API that allows endpoints to communicate with server	Determine a high-level API design for the users, shopping, sensor, stores, and routing services	15 Jesrik Gomez
Make Users Service	Service to register new users and handle logins with cookies	10 Jesrik Gomez
Make Shopping Service	Main inventory management service which will handle inventory and shopping lists for users	40 Nate Wernimont
Make Sensor Service	Service to handle communication between sensors and server	15 Jesrik Gomez
Make Stores Service	Main store inventory management service to gather item information from nearby stores	40 Nate Wernimont
Make Routing Service	Service to create route based on shopping priorities	40 Nate Wernimont
Construct relational model	Create the schema and the databases in MySQL	10 Nate Wernimont
Analyze queries for needed indices	Look at the needed queries and optimize them to use indices	20 Jesrik Gomez

Table 1: Personnel Effort Requirements

4.4 Other Resource Requirements

In order to complete this project, we will need two Raspberry Pi microcontrollers and two Raspberry Pi zero “W” microcontrollers. A portable power source will be required to power the sensor modules to allow them to be placed remotely around a home - in areas from which electrical outlets are not reachable. Aside from the microcontrollers, we will need weight-dependent resistors - with built in analog to digital converters. We will be using the free service tier for Microsoft Azure to host our backend services. We do not expect to need to upgrade to a paid tier, but this issue may need to be revisited in the spring when we are testing the recommendation service with large amounts of data.

4.5 Financial Requirements

We need to purchase some raspberry pis for developing and testing prototypes. 2 Raspberry Pi 4 boards and 2 Raspberry Pi Zero W boards should be sufficient for our purposes. Additional circuitry parts or plastic housings may be required for the microcontrollers and weight sensors. We need to buy the weight sensor, which is typically available for around $10. The overall estimated hardware cost of this project is $250.

5. Testing and Implementation

Testing and test results are a critical indicator of the success of this project.

Testing of this project will be conducted as follows:

1. Types of Tests
    a. User acceptance testing based on functional requirements will be conducted to validate that requirements are met. Test plans and acceptance criteria for each use case are listed in section 5.3.
    b. System testing will be conducted to ensure that all of the components integrate with each other properly. Non-functional requirements such as performance, reliability and scalability will be tested.
2. Items to be Tested
    a. Raspberry pi & weight sensor device
    b. Android application
    c. Remote server
        i. Inventory service
        ii. Store service
        iii. Recommendation service
3. Test Cases

    a. Specific test cases will be created during the second semester of the project. General test cases aligned with our core use cases are described in sections 5.3 and 5.4.
4. Test Documentation
    a. The testing process and procedures will be standardized. Once finalized, the testing process will be documented.
    b. Results of tests will be documented - problems or areas of improvement will be noted so that the next revision can address them.

5.1 Interface Specifications

Sensor to Raspberry Pi

The weight or pressure sensor and the raspberry pi will be connected via the microcontroller GPIO.

Raspberry Pi to Remote Server

Communicate via an encrypted TCP/IP port and exchanging HTTPS requests. The format of the data sent could be formatted using the widely accepted JSON or XML schemes.

Remote Server to Android App

The connection between the remote server and the Android app is similar to the connection between the raspberry Pi and the remote server - using HTTPS for encrypted communication.

5.2 Hardware and Software

We will use the following hardware to test and develop the project:

- Android phone to test the Android application.
- Raspberry pi to connect to the weight or pressure sensor and relay data to our remote server
- Oscilloscopes to observe the behavior of the weight sensors - voltage gain, offset, delay, accuracy
- Function generators to provide regular, sophisticated waveforms to stimulate the weight sensors
- DC voltage sources to provide regulated power to the weight sensors
- Digital multimeters to observe the value - current, voltage, resistance - of the weight sensors with differing loads

The following software will be used:

- Android Studio provides an emulator for any team members without an Android phone

- Postman allows testing of the REST API calls on our server
- MySQL Workbench will be used to test storage within our databases

The end product of this project will result in one instance of the remote server - running multiple services - and many instances of sensor devices - many users having many sensors. This also includes many mobile devices interacting with the system.

5.3 Functional Testing

Functional testing will be conducted by user acceptance testing for each use case. A test plan with acceptance criteria is outlined for each of our use cases in the table below.

Use Case	Testing Plan	Acceptance Criteria
1. Install and Set up an Inventory Sensor Device	The device is powered on for the first time by the user - it has previously been programmed by us. The user will be given instructions via the Android application. The application will walk them through the setup process - connecting the device to their wifi network, calibrating the weight thresholds, and naming the item that the sensor is measuring - and account creation if they do not already have an account.	The sensor device should be successfully registered to the user's account - and the user should have an account, if they did not have one previously. The inventory service - remote server - should now be aware of the sensor device and relate it to the user.
2. View inventory status	The user opens the Android application on their device and goes to the inventory menu. They view the status - relative inventory amount - of the items that are being tracked by sensor devices associated with their account.	The user should be able to view the inventory status of all sensor devices associated with their account. The sensor reading should be timely - updated within an hour of viewing the status - and accurate - based on the user's calibration.
3. Viewing and modifying the shopping list	The user opens the shopping list menu in the Android application. They modify their shopping list by adding, removing, or updating the quantity of items on the list.	The user should be able to modify their shopping list - remove, add, and update quantity - and have the changes reflected in the application the next time they open the application - in other

		words, the remote server needs to save the changes.
4. Automatic inventory tracking	The user decreases the quantity - the weight decreases - of an item that is being tracked by a sensor device - the device is already set up and associated with the user's account. Both small decreases and full depletions of inventory are simulated.	After a change in the inventory - weight - that a sensor measures, the inventory server updates the stored inventory value in the database. If the value is below the user's set "low" value - established during the sensor setup process, then the item - whatever item the sensor is measuring - is added to the user's shopping list.
5. Receiving shopping recommendations	A user with an established shopping list - with multiple items that can be found at multiple different stores - opens the Android application. They open the "start shopping" menu to receive shopping recommendations.	The user should receive shopping recommendations based on their current location, their shopping list, and their input optimization criteria - saving the most time, saving the most money, or somewhere in the middle. The recommendations are also based on store proximity and availability - operating hours, available items in stock, sufficient quantity in stock at the specific store.

Table 2: Functional Testing Plan & Acceptance Criteria

5.4 Non-Functional Testing

After the functional requirements of the project are satisfied and validated by the testing defined above in section 5.3, we will conduct non-functional tests in order to ensure the quality of the product beyond the visible result to the user. System testing, stress tests, and monitoring will be conducted as described for each area below:

Performance Testing

- Monitor the network use while the inventory sensors are active with the user's home
- Monitor the network, battery, RAM, CPU, and storage usage on the smartphone when the app is running and when the app is in the background

Security Testing

- Use a packet sniffer to try to gain access to user information
- Use SQL injection and buffer overflow attacks to try and break both client and server code

Usability Testing

- User testing on outsiders who are unfamiliar with our project can indicate whether the project and the user interfaces are usable enough or if they need additional explanation.

Scalability Testing

- We are initially focusing on smaller data sets to simplify the scope of the recommendation problem - otherwise we are dealing with an NP-complete algorithm problem. Later on, we can introduce more data and possibilities for recommendations and test the algorithms that we have designed. We can also simulate a large number of sensors or users to see how our system scales.

5.5 Design & Development Process

During the fall semester, the team followed the process illustrated in Figure 9 below while designing the solution. First, we researched different options for hardware and software technologies that could be used to solve the problem at hand. After comparing the options, we narrowed it down and made selections based on a list of pros and cons for each. Currently, we are in the prototyping phase, where we will create rough prototypes of the system components. For example, we currently have prototypes of the sensor devices using Raspberry Pi microcontrollers and weight sensors. In the beginning of the spring semester, we will still iteratively prototype, test, and improve the designs of each component, making small improvements or changes with each iteration.

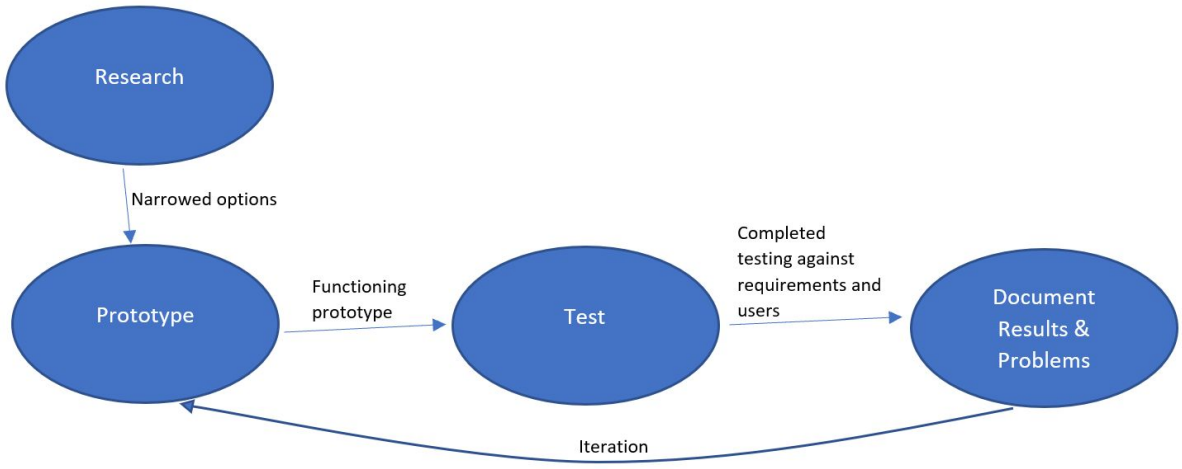

Figure 9: Project Testing & Iterative Development Process Diagram

5.6 Results

During the fall semester, we prototyped many things on the Android side. The three main prototypes focused on encrypted communication, background tasks, and notifications. We created a prototype Android App that communicates with a Node.JS server using HTTPS (encrypted network), and determine if there was a change in the server's inventory quantities. If the measured quantity of an item falls below a specific threshold, it will create an Android notification on our device. The prototype demonstrates our ability to communicate through encrypted networks, helping with our user privacy and security. We plan to incorporate it into the final Android application next semester.

Additionally, a prototype inventory sensor was created using a raspberry pi microcontroller and a force-sensing resistor to measure the weight of inventory items. We found that the force-sensing resistors were not sensitive enough to provide a sufficient inventory measurement because the resistance did not change in small enough amounts. We plan to use more sensitive weight-sensors next semester to ensure that we can accurately measure inventory amounts by weight.

6. Closing Material

This concludes the planning and designing of our system. Following will be the takeaways from our work thus far.

6.1 Conclusion

Up to this point we have done extensive research on the tools and technologies that we can use to help us develop a solution to our problem statement. After heavy ideation, we've landed on design decisions based on given constraints and criteria created by the team. The current draft summarizes our findings, decisions and vision for subsequent versions of the project.

From here, we plan to begin some rudimentary prototyping to approach our goal of a Minimum Viable Product by March 15th. The smaller details of the designs will inevitably evolve but solid foundations have been created.

6.2 References

[1] A. Hauge, D. Bis, S. Guenette, H. Moser, N. Bix and B. Gruman. *Automating Inventory Management: Routing through Sensor Networks*. Iowa State University: 2019. http://sdmay19-29.sd.ece.iastate.edu/

[2] M. Quadrana, P. Cremonesi, and D. Jannach, "Sequence-Aware Recommender Systems," Sequence-Aware Recommender Systems, Feb-2018. https://arxiv.org/pdf/1802.08452.pdf.