IoT Passive Monitoring for Assisted Living Homes

Final Report

Client
Andrew Guillemette

Advisor
Goce Trajcevski

Team 14 Members/Roles
Austin Sudtelgte: Sensor Team & Co-Lead Engineer
Joshua Blanck: Sensor Team & Report Manager
Austin Kerr: App Team & Co-Lead Engineer
Ryan McCullough: Sensor Team & Meeting Facilitator
Nick Schneider: App Team & Meeting Scribe
Trevor Lee Henderson: Backend Team & Test Engineer

Email:
sddec18-14@iastate.edu

Website:
http://sddec18-14.sd.ece.iastate.edu/

List of Figures

List of Tables

1 Introduction

1.1 Problem Statement

The families of elderly people can often have a hard time keeping track of their elderly family members' health and ensuring that they are given the care they need. Often times, however, elderly people may forget to follow up the prescribed regimen (in both nutritional and therapeutic sense) - and sometimes, elderly people want their family to think that they are in good health, so they may even mislead their family about following the prescribed regimen, in order to keep them from worrying. As an extreme case, if the an elderly (or other frail individual) is not doing well and needs medical attention, it is imperative that the family is able to get the tenant that attention in a timely manner. Towards that, a system for effective monitoring of the nutritional and therapeutic habits of an elderly person is a paramount.

1.2 Purpose

We are proposing a new product designed to solve this problem. Essentially we want to use passive, non-invasive sensors to collect and store data about the subjects habits. In order to help the family know if their elderly relative is doing well, we propose a system that will enable collecting data about eating/drinking habits, sleeping habits, and personal hygiene. This data will be analyzed to see if the elderly relative is staying within normal ranges - and the family will be notified if there are deviations from the expected behavior; for example, the elderly relative stops eating. This, in turn, will enable proper actions to be taken to ensure that the elderly person can be taken care of in order to maintain/improve their health status.

1.3 Goals

We focused on designing a novel prototype system that will enable a collection of data from multiple sensors. The objective is that the sensors used to gather data must be passive and non-invasive. Data of interest would be eating/drinking habits, sleeping habits, and personal hygiene. This data must be securely stored and needs to analyzed to ensure that behavior of the elderly member is within acceptable ranges. Whenever abnormalities are detected, family members and/or caretakers must be notified to take the appropriate action.

2 Deliverables

The deliverables for this project can be broken into three parts: Data Collection, Cloud Server, and Mobile Application.

The first part is the lowest level and deals with data collection. One of the deliverables is that the sensors researched and selected to perform the data collection function together and that more sensors are able to be added when necessary with little overhead. The sensors will need to report their data to something while remaining mostly “unintelligent” meaning they don’t communicate with the outside internet. The sensors will report their data to a local hub, which is responsible for forwarding the data to the cloud server. The different types of sensors are required to perform differently. The magnetic door sensors are required to report an open/close event and the duration of the event. Flowmeters are required to report the duration of the flow event, and the flow rate of the event. Additionally, we were tasked with researching a sensor to monitor sleep quality and duration, as well as load cells capable of measuring minute changes in a person’s weight while remaining out of the way.

The cloud server must be capable of storing the collected data and running analysis on any date that is already stored in the database. The cloud server must also be able to push data to mobile and or a webapp.

The mobile application must be capable of displaying collected sensor data and events as well as their duration and date of occurrence.

Finally, as with any project, documentation of the work done and system as-is is necessary.

3 Design

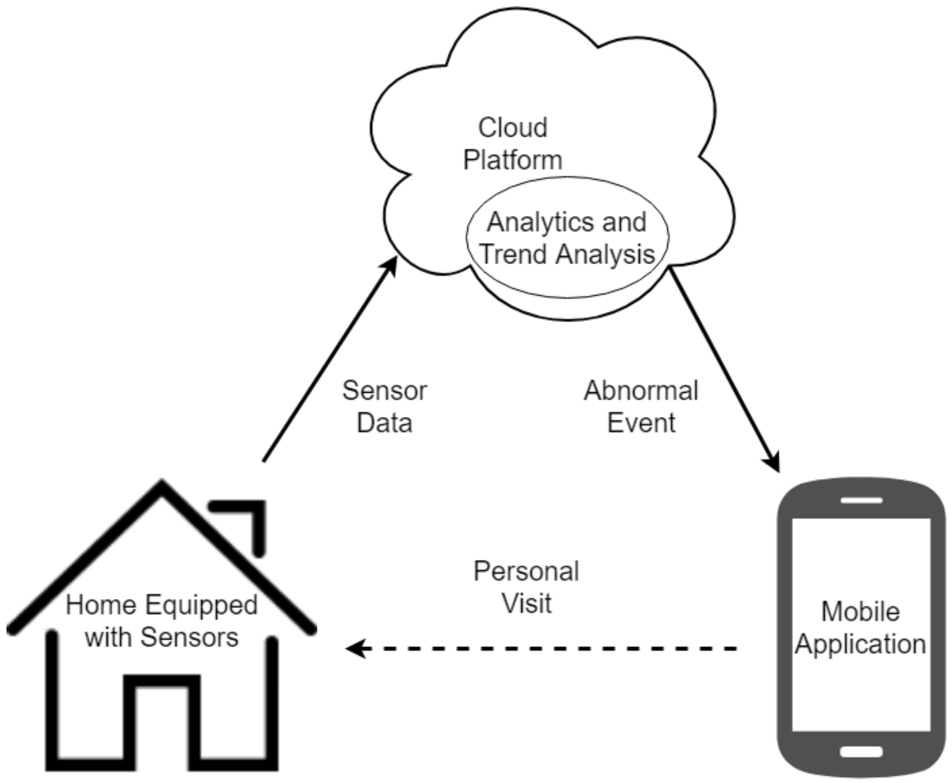

Fig 3.1: High-Level Overview of the Main Design Components

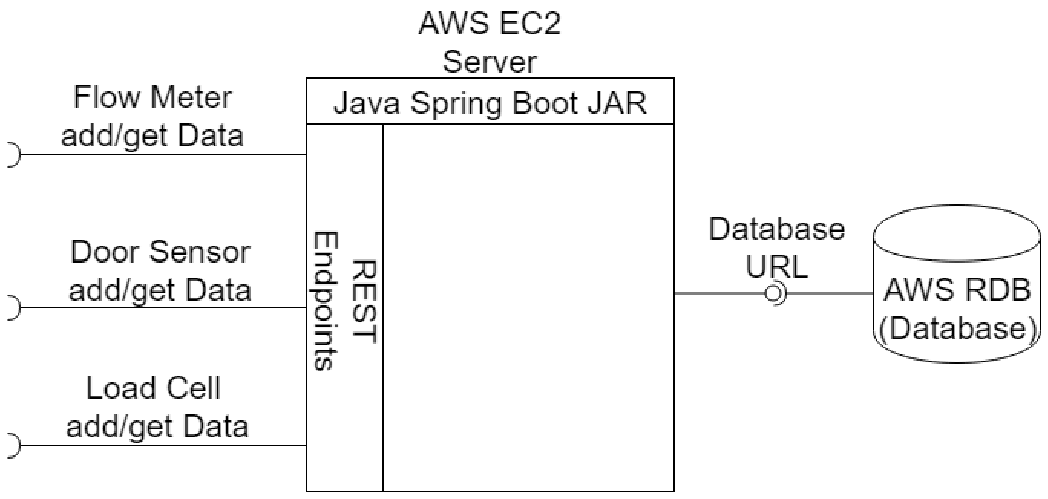

Fig 3.2: Server Design

3.1 Previous Work and Literature

Many of the solutions we found during our research [Torres, Roberto L. Shinmoto, et al.][IEEE] contained wearables, which the client did not want to include for this project.

3.2 System Block Diagram

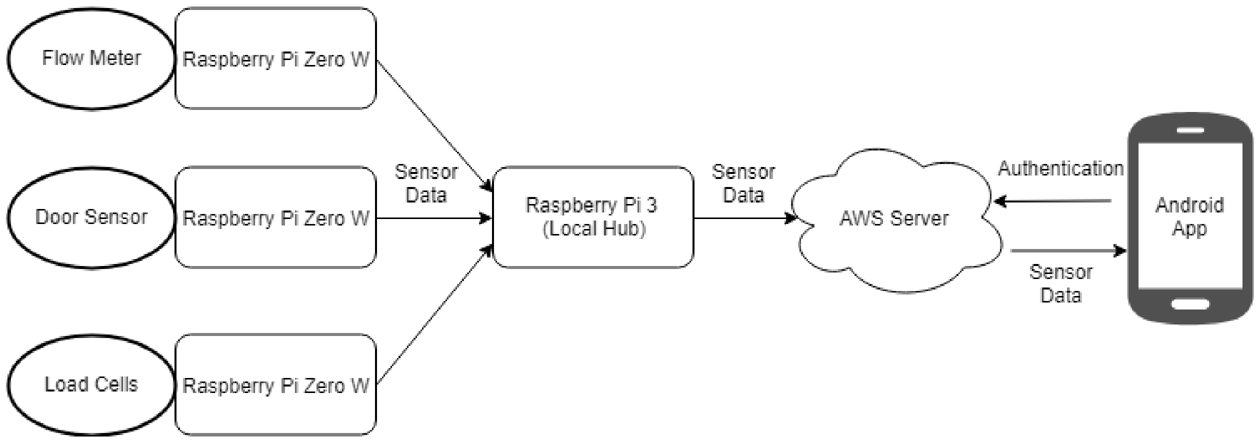

Fig 3.3: Block Diagram of Designed System

3.3 Operating Environment

The system was developed with the intent to be used in an independent living facility with an elderly or disabled occupant. As such, the sensors are required to be non-intrusive and non-invasive so as to not get noticed or tampered with. Keeping these things in mind, sensors were selected based on a “low profile” and low maintenance needs. Sensors were also selected such that they could be installed out of sight of the resident, to mitigate distress and the residents’ desire to tamper with the system.

The mobile application environments was not given any specifications from the client, as such, specifications were created as the project evolved. The mobile application was selected to reach the largest user base - and for the purpose of the proof-of-concept implementation, Android environment was chosen.

The web interface, having been given no particular specification (or other detailed requirements), was chosen and designed based on the prior knowledge that the team members already had.

3.4 Intended Users and Use

The project has three categories of users, they are: the resident, the caregiver or healthcare provider, and the system admin. These roles are elaborated on below.

The residents that the sensors were designed to monitor are elderly people living in independent households. The goal when implementing our sensors was that the residents could continue living their life as normal, without being disturbed by the sensors collecting the data. During our second semester, the resident filled out logs of their actual meal details to help us determine if our data collection was accurate.

Healthcare providers and residents' loved ones fall under the second category of users. The role that these users play is to monitor the event logs in the app and determine if the resident has missed any meals. This way the resident is able to maintain their independence instead of living in a retirement home and the residents' loved ones are able to feel comfortable about their safety.

Admins are needed to add new users to the database and new residents to user accounts. Once the accounts are set up the healthcare providers and residents' family members will be able to login with the respective credentials to view the sensor logs. They are also able to install and set up new sensors in the database for a particular resident.

3.5 Assessment of methods

An assessment of the local server is that it is both modular and simple to implement, however it does place an upper limit on extensibility. That limit is unlikely to be reached, however.

The sensor choices were made for their low profile, low cost, and low maintenance factor. The disadvantage of this is that the data received from the sensors may not always be the most accurate or desirable.

3.6 Validation

For our system to be considered valid, we must satisfy several criteria. Firstly, this system is intended to monitor the subject's health via eating/drinking, sleeping, and restroom usage. Thus, the system must record instances of these actions. The sensors on cabinets and flowmeters, combined with the event identification in the backend satisfy this criterium.

These sensors must also be installed in such a manner so as to be either unnoticeable, or at least not disruptive to the resident. Further, these sensors and the physical components of the system must be low-maintenance and reliable, such that technicians would seldom need to repair the system, and thus prevent undue interruption to the resident.

The data collected from the sensors will need to be transmitted to a central location. Otherwise, technicians would need to continually visit the residence to pull data from the sensors. Along with transmitting the data, the data must be readily available to the app. As health issues can be time-sensitive, application users should be able to view events and data relatively soon after they occur.

4 Project Requirements and Specifications

4.1 Functional Requirements

- The system must record:
  - an instance of someone eating/drinking
  - an instance of someone sleeping
  - bathroom activity
- The app is required to:
  - be responsive
  - enable data visualization by showing a log of events
  - be secure in both data and lower level system access
- System can report:
  - loss of a sensor
  - loss of power or a low battery

4.2 Non-Functional Requirements

- System should:
  - function passively and non in-intrusively
  - collect data to some central location
- Data should be:
  - analyzed to determine if an alert should be sent to users
- Mobile application should:
  - allow users to view sensor events
  - alert users based on event criteria and data collected

4.3 Standards

Restful API

A REST API is used to facilitate communication between the server, IoT Devices, and the android application. The statelessness of the REST API comes as a great advantage for our project as data is constantly being sent and retrieved by a large number of devices.

The REST API also enforces a separation of concerns between all components of the project. If anything on the application changes the functionality of the server will remain the same as long as the protocol is adhered to. Likewise changes to the servers code base or even the sensors code base will not change the functionality of the project.

HTTP

HTTP is a core component of our REST API. HTTP, along with JSON, enforces a standard for communicating with the server. The local hub that communicates with the sensors utilizes HTTP to post sensor data to the server while the android application uses HTTP to fetch data from the server. So long as these protocols are adhered to each component of the project exists as an independent entity

Relational DB

Due to the nature of our project, there are many relations between various sets of data that while it wouldn't be prudent to group them together need to be unified in some way. This is why we chose to use a relational database. With a relational database we are able to store information in separate tables and then perform a relational operations to find pertinent data. For example, a given resident with in a set of residents have multiple sensors of multiple types that pertain to them. Rather than lumping all the sensors together we can set them into separate tables and have the residents ID field be a secondary key to the sensor.

Wi-Fi

Wi-Fi is used by the Raspberry Pi's to communicate with each other as well as with the server. After the local Pi's receive an event from a sensor the data is formatted properly and sent over Wi-Fi to the Pi acting as a local hub. The hub keeps the events in a queue and sends them to the server for storage. The Pi's all need to be constantly connected to Wifi to send the events they recieve.

JSON

The JSON format is used between all of our devices in order to communicate with each other. When the sensors are triggered and collect new data, that data is combined with the sensor id and timestamp before being sent to the local hub in JSON format. The local hub is able to process this JSON data and forward it on to the server. The AWS endpoints are set up in a way that allows JSON data to be easily read and processed by the back end. The Android app is set up in a way that allows it to communicate with the server through JSON as well.

5 Testing/Development

5.1 Interface Specifications

The sensors are wired into the raspberry pis. Interfacing between the raspberry pis and the remote server is done by putting the data into a json string and sending the string via http request to the cloud server.
The remote server is a REST API that has endpoints to either add or retrieve data from the relational database. All requests and responses from the remote server are formatted into JSON objects allowing for better separation of concerns between the remote server and the other components of our project and to make parsing easier from the app and to be able to use standard libraries for JSON string creation.

The database receives queries from the remote server through a JDBC driver. This is done by creating a class that inherits the JpaPersistence interface. The JpaPersistence interface prevents SQL injection by sanitizing of data going to the database.

5.2 Hardware

Our prototype used several types of sensors that we wired into raspberry pis in order to collect data about the subject's habits.

- **Door Sensor:** We track when a person eats by using door sensors. The door sensors is meant to be placed on refrigerator and pantry doors or even cabinets doors for cabinets containing dishes for eating. The data of these doors opening and closing can then be interpreted to show when a person is eating. We chose this sensor because it operated as desired and integrates well with the rest of the sensors.
- **Flowmeter:** We planned to use a flow meter for the purpose of tracking drinking,bathing, and restroom habits. Based on how long there is flow, we can determine whether or not an event occured. This sensor was chosen because it performed the desired task, required only a basic setup, and worked well with the other sensors.
- **Load Cell:** We planned to use load cell sensors to keep track of toilet usage. The load cells would be placed under the toilet seat and will be able to detect when a person sits on the toilet seat, in theory they will also detect how much waste has left the person's body. We also considered attaching a flowmeter to the toilet to help account for the case of a male urinating standing up.

We created python scripts on the raspberry pis to detect the data coming from the sensors and add timer functionality before sending it on to the local hub. We used threading and interrupts to allow us to send data coming in from multiple sensors at the same time. For links to the sensors that we chose after researching, see Appendix 1.

5.3 Functional Testing Process

We used the following processes to test the functionality of each part of the project.

5.3.1 Sensor Testing

The door sensors we used function in a very simple manner. Testing them involved ensuring that the system recognize an event created by connecting and disconnecting the two magnetic pieces. The system was tested to ensure it was able to accurately detect an open/close event, as well as record the time at which the event occured. The only other functionality that needed to be tested was that the close time was recorded as well as the open time, so that the duration of the event could be calculated. This testing process involved including print statements in the door sensor python script inside event recognition blocks. Whenever the door sensors were brought together the system would immediately print off the change in state.

In order to test the flowmeter, an apparatus was constructed [Fig. 5.1]. This apparatus allowed us to accurately measure the amount of water flowing through the flowmeter as well as gain experience in the best way to install this device in a live environment. Measuring the amount of water flowing through the flowmeter was important as the manufacturer noted that each flowmeter would need to be calibrated, also the viscosity of different substances isn’t standard.

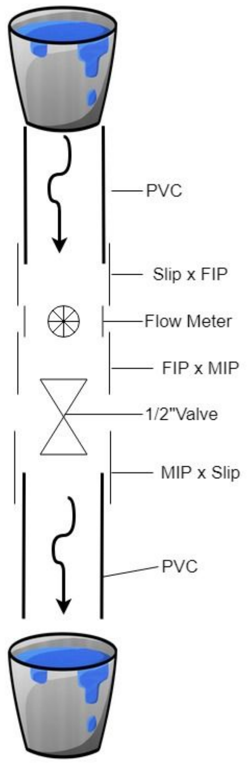

Figure 5.1: Flowmeter Testing Environment

5.3.2 Communications Testing

The local hub needed to successfully receive every communication from the other raspberry pis hosting the sensors. To do this we compiled a list of likely scenarios for the prototype to encounter, and then ran through them. An example is the local server needed to correctly receive data when multiple sensors on different pis sent data all at the same time. We set up a small environment for that scenario and ran through it many times in slightly different configurations, until we were satisfied that the prototype would perform correctly in this area in the larger testing environment we were going to move into.
The remote server also needed to receive all of the communications from the local server with now failures. To validate that this communication was also working as expected after each session of running scenarios we checked the database against the print statements on the console of the local hub raspberry pi. We quickly achieved nearly 100% accuracy in communication between the local hub and the remote server, but continued to validate throughout the testing process.

5.3.3 Application Testing

The final goals of the Android application was to allow authentication of a user and display information related to said user. Authentication testing required testing multiple users in the database while also submitting erroneous usernames and passwords with special characters. Once past the login screen user info must display correctly. The first test was to make sure all data matched what was stored in our database. Second the app had to be tested on several devices to ensure that information was readable and displayed in the manner expected.

5.3.4 Integration System Testing

After individual component testing was finished we tested the integration of the entire system by turning different sensors on and off and checking the app to see if data that matched up appeared. This tests every component, if the data shows up in the app, that means the remote server correctly put it into the database, the local server received and sent it correctly, and the sensor pi's collected and formatted the data correctly.
There is an additional form of integration system testing that we wish we had done before installing in a live environment, because it would have made it much easier to catch the minor bugs we encountered there. This test would involve setting up sensors in a small scale environment we could easily get to and debug in for longer periods of time. If everything was left on at all times when we could have debugged any problems that came up before we installed the sensors in the live testing environment.

5.3.5 Live Testing Environment

Our live testing environment was set up at Green Hills Retirement Community in Ames, IA. The client for this project was able to get permission by the community and one of their residents to allow for set up and monitoring of the teams passive sensors. Figure 5.2 below shows where in the resident's kitchen the sensors were placed. Sensors 10-15 were connected to Pi 1, sensors 16-19 were connected to Pi 3, and sensors 20 and 21 were connected to Pi 3. All of the sensors were installed on the last weekend of October 2018 and the team began data collection on Tuesday, October 30th, 2018.

The team made note of what items were stored in what cabinets and did an initial questionnaire with the resident to determine baselines for what meals he ate and when. Throughout the testing period the resident filled out a one page log over what meals he ate, when he ate them, and what appliances/utensils he used for each meal. These logs were then used to cross reference the test data received by the team to determine if the sensors were recording data correctly. These logs are also beneficial when working on event identifiers or in other words determining whether or not the resident had a meal.

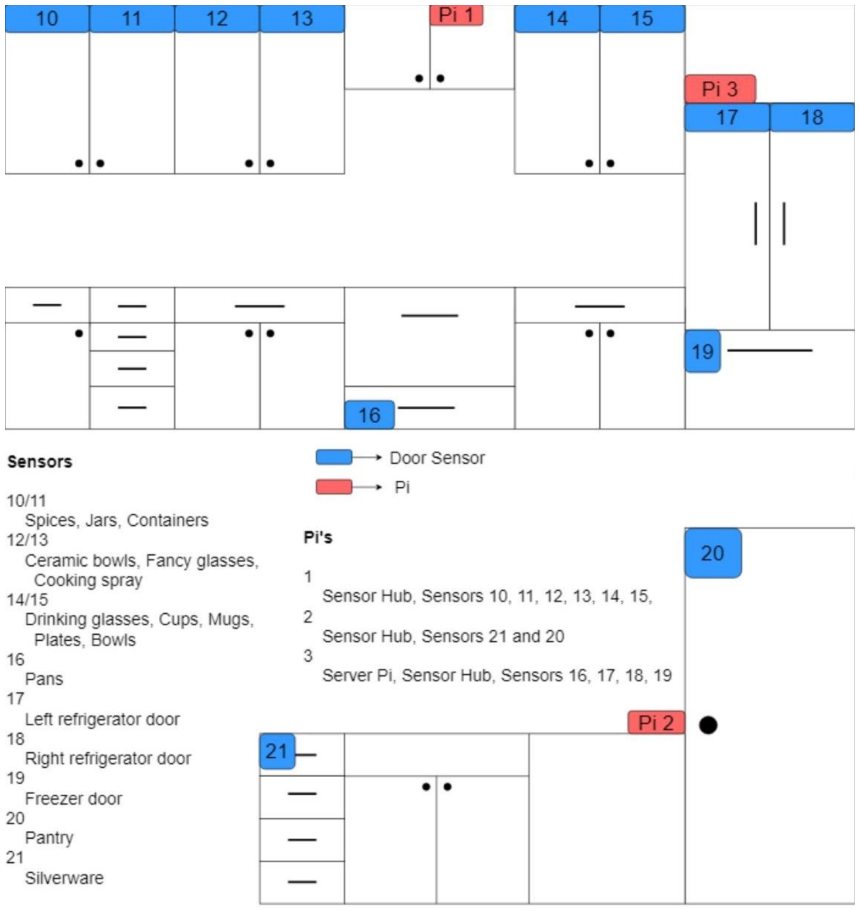
10
11
12
13
Pi 1
14
15
Pi 3
17
18
19
16
Door Sensor
Pi
20
Pi 2
21
Sensors
10/11
Spices, Jars, Containers
12/13
Ceramic bowls, Fancy glasses, Cooking spray
14/15
Drinking glasses, Cups, Mugs, Plates, Bowls
16
Pans
17
Left refrigerator door
18
Right refrigerator door
19
Freezer door
20
Pantry
21
Silverware
Pi's
1
Sensor Hub, Sensors 10, 11, 12, 13, 14, 15,
2
Sensor Hub, Sensors 21 and 20
3
Server Pi, Sensor Hub, Sensors 16, 17, 18, 19

Fig. 5.2: Sensor Placement in Live Testing Environment

6 Design Testing/Implementation Results

The initial design encompased a much wider range of operating environments. The original intent with the mobile application was to create it using the Xamarin framework. The Xamarin framework allows an application to be written in C# then compiles into both and .apk file for installation on android devices, and an .ipa file for installation on an iOS device. Upon attempting to use Xamarin, the app team was constantly running into roadblocks or having to reinstall the Xamarin framework on their development device. To speed production, the call was made to move the existing app to android in java.

In conjunction with Xamarin, the server hosting platform initially selected was Azure. This platform was selected because it integrated well with Xamarin and was advertised to have better machine learning capabilities than the other cloud hosting platforms on the market, something that would be necessary in the future. In the end, the switch from Xamarin and the increased cost of Azure pushed a change. Amazon Web Services(AWS) was switched to with almost no extra work and a much smoother working environment as previous knowledge of how to use the platform existed.

An unrelated change that was made to the initial plan was the load cells. Initially, some load cells were purchased and installed but they proved not to offer the functionality desired. This was likely a result of miscommunication and not enough thorough research into the capabilities of the load cells selected. Ultimately, an alternative was selected and implemented over the summer. This semester the discovery was made that those load cells wouldn’t work either as they started to adjust themselves to the weight being applied to them after a short time.

The final change to the initial plan was the removal of one of the sensors. The initial plan was to use a Ballistocardiogram to monitor sleep habits and a variety of other sleep data such as heart rate, breathing, quality of sleep, and duration of sleep. This option, however, was eliminated from further consideration because a single unit with the targeted functionality turned out to cost more than the client was willing to spend on the prototype.

7 Timeline

7.1 First Semester

January	February	March	April
-Determine what data will be useful to health monitoring **Hardware** -Brainstorm various sensor use cases -Generate and justify list of sensors to collect data	-Work on hardware and software flowcharts **Hardware** -Brainstorm ways to bypass selected sensors **Server** -Research cloud/ local server solutions **Application** -Begin work on data visualization solution	**Hardware** -Present and finalize sensors and use cases -Design and build tests for sensors **Server** -Implement interface between sensors and storage **Application** -Work on storage system (cloud and local)	-Begin collecting test data -Prep for in depth testing over the summer (to be done by Client) **Server** -Refine storage solution **Application** -Finish MVP of app prototype

Table 7.1 First Semester Schedule, created at end of First Semester

7.2 Second Semester

September	October	November	December
Hardware -Finalize pin configuration -Implement multithreading on Pi's **Application** -Create Heatmap visualization **Server** -Create sensor ID API	**Hardware** -Plan and install into test environment **Application** **-**Improve application UI **Server** -Implement Login and Authentication API **Testing** -Create and finalize Daily Log and Questionnaire for test resident	**Hardware** -Debug and solve issues in test environment **Application** -Improve data readability via string mapping **Server** -Create APIs to facilitate the revisions to the application **Testing** -Collect and analyze daily logs	Finalize Project documentation.

Table 7.2 Second Semester Schedule, created concurrently during Second Semester

8 Conclusions

The demographics of the United States are changing rapidly. According to the Population Reference Bureau nearly 25% of the population is projected to be over the age of 65 by 2060, from 15% in 2016.
The overall idea of this project is to help families keep track of their elderly relatives' health which will be in increasing demand over the coming years.
Towards this end we constructed a prototype as an initial effort towards this goal. We believe this prototype is a good model of what technology like this should look like. Our prototype is able to both collect important health data about the subject, and respect the subjects privacy because of the passive nature of the sensors we used. We have also chose what we believe is the best model of delivering information to the subjects loved ones in our password protected smartphone app.
The next section will be devoted to the project moving forward into the future.

9 Future Project Extensions

This project has been designed to be extensible, thus there are some features that can be added in later.

Further sensors or data collection devices can be added to the system: prototype support for a smart outlet is being developed, as well as a wearable identification/authentication solution. Previous research has been conducted into obtaining a sensor to monitor sleep and sleeping habits.

Once sufficient data has been collected, a machine learning algorithm can be applied to identify the subject's regular behavior, and then can be extended to identify behavior deviating from the subject's normal. This can be applied to all users.

Additionally, circuits can be made for the individual sensors so they don't have to be wired across the kitchen/bathroom areas.

Finally, the system can be made more user-friendly in the areas of adding new or additional sensors, creating new users, adding new caregivers or assigning caregivers to a subject, and the app can be made more robust in its data visualization.

10 References

Context-Aware Wireless Sensor Networks for Assisted Living and Residential Monitoring - IEEE Journals &; Magazine, ieeexplore.ieee.org/document/4579768/?reload=true.

Schietse, Elly. “Smart Homes for Seniors: How the IoT Can Help Aging Parents Live at Home Longer.” Qorvo, 9 Aug. 2017, www.qorvo.com/design-hub/blog/smart-homes-for-seniors-how-iot-helps-aging-parents.

Torres, Roberto L. Shinmoto, et al. “A Battery-Less and Wireless Wearable Sensor System for Identifying Bed and Chair Exits in a Pilot Trial in Hospitalized Older People.” *PLOS ONE*, Public Library of Science, journals.plos.org/plosone/article?id=10.1371/journal.pone.0185670.

Mather, Mark. “Fact Sheet: Aging in the United States.” *Population Reference Bureau*, Population Reference Bureau, 13 Jan. 2016, www.prb.org/aging-unitedstates-fact-sheet/.

Appendix 1

- The Door Sensor used in this project: https://www.adafruit.com/product/375.
- The Flow Meter used in this project: http://www.hobbytronics.co.uk/yf-s201-water-flow-meter.
- The Load Cell used in this project: https://www.sparkfun.com/products/10245.
- The Raspberry Pi Zero used in this project: https://www.adafruit.com/product/3708.
- The Raspberry Pi used in this project: http://a.co/9qI22gA.

Appendix 2 - Operation Manual

Hardware Setup

1. Download the latest version of Raspbian and flash it to a Micro SD card
2. Run *sudo apt-get update* and *sudo apt-get upgrade* to make sure the OS is the latest version
3. Follow the steps for the desired sensor type found below

Running the code
Each Raspberry Pi can support multiple of the same sensors; all that is required is to update the GPIO_Info file. The format is such that the Sensor ID (used on the server) is the value on the left, and the GPIO number is the value on the right; a single space is the delimiter and the file must end on an empty line. Otherwise, the start script won't parse the last line. GPIO pins are Board pins not GPIO number. This can be changed if desired.

Each sensor type has a start script that starts a new instance of the program that watches the designated GPIO pin. The name of this file varies but the functionality is the same. To run it type: *bash StartScript.bash GPIO_Info*.

General idea for moving forward

Startup
The thought to remove this setup process is to write a bash script to install the files to the proper location such that they would run on startup.

Adding new sensors
An idea for adding new sensors is to install the sensor and connect it to the Raspberry pi, then have a script that watches all GPIO pins not already being watched and that are suitable to collecting generic data, and prompts the user to open and close the door sensor, or run the flowmeter, etc. In effect, this looks for any GPIO pins that are receiving data that were not previously. The script then asks the user for the sensor type and, with this information, asks the server to assign a new ID for the device. It then adds the ID and GPIO pin to the GPIO_Input file.

DoorPi Setup

1. Follow steps prescribed in the ReadMe from the parent directory if not already complete
2. Put doorSensors.py and DoorStart.bash into the home directory of the pi
3. Create a GPIO_Info file (The name can be whatever you want)
4. Install Door sensors and connect them to the pi. Update the GPIO_Info file with the sensor ID used on the server and the GPIO Board pin number
5. Type `bash DoorStart.bash GPIO_Info` to start monitoring door sensors

FlowPi Setup

1. Follow steps prescribed in the ReadMe from the parent directory if not already complete
2. Put flowSensors.py and FlowStart.bash into the home directory of the pi
3. Create a GPIO_Info file (The name can be whatever you want)
4. Install flow sensors and connect them to the pi. Update the GPIO_Info file with the sensor ID used on the server and the GPIO Board pin number
5. Type `bash FlowStart.bash GPIO_Info` to start monitoring door sensors

Note:
Using multiple Flow meters from one pi has not been tested and may result in too much power being drawn from the pi. To mitigate this risk, consider using an external power source rather than the two 5V pins on the pi.

Additionally, the voltage will need to be stepped down so the signal coming back from the flowmeter doesn't fry the board. Here is the image of the voltage divider. Where the black wire is ground and the yellow wire is the signal coming in. The connectors with the round ends are coming from the flowmeter and the square-ended-connectors are going back to the pi.

Local Server Pi Setup

1. Put httpServer.py somewhere on the pi you want to use as the local server
2. Make sure this pi has a dedicated IP address or will never be turning off

3. Update the door sensor and flowmeter scripts with the appropriate IP address
4. Either open the file and run it from the IDE or use a terminal and type py httpServer.py

Note:
The server can be run on a pi that is also monitoring sensors. However, we recommend running the server on a separate pi, should available resources allow.

AWS Server Setup

Building
To build the project, navigate to the project directory where the mvn file is stored and use Maven to build the project. This would look like *mvn package* while in the correct directory.

Deploying
After building the project, you should have a stand alone .jar that acts as an executable. You will want to use FileZilla or another SFTP utility to add your .jar to the server.

In order to access the server, you need to have a private key file. To get a private key file, login to AWS and navigate to the running EC2 instance. Click the connect button and then click on the link that will navigate you to a connection wizard. This will generate a private key file for you.

Once you obtain the key, follow the wizard’s steps on how to SSH into, and then transfer files to the server.

Running the server
After you have transferred your .jar to the server and you have SSH'd in you need to run *ps* on the linux server find the process number of the previous jar and kill it through the kill command. Then run your jar using the command *nohup java -jar "insert name of jar here".jar &* This will prevent the server from closing when you close the SSH client, as well as run it as a background process.

Android App Setup

1. Download the app to the desired smartphone
2. Login to a user account with the appropriate credentials