UNIVERSITY OF CALGARY

ENGO 500: GIS and Land Tenure #2

Progress Report

By: Jeremy Steward
Kathleen Ang
Ben Trodd
Harshini Nanduri
Alexandra Cummins
Supervisor: Dr. Steve Liang

DEPARTMENT OF GEOMATICS ENGINEERING

SCHULICH SCHOOL OF ENGINEERING

02/14/2014

ENGO 500 – Group GIS & Land Tenure 2
Jeremy Steward
Alexandra Cummins
Harshini Nanduri
Kathleen Ang
Ben Trodd

February 14, 2014

Dr. Steve Liang
Schulich School of Engineering
University of Calgary
2500 University Drive NW
Calgary, Alberta T2N 1N4

Dear Dr. Steve Liang:

Attached you will find a Progress Report detailing the efforts of the GIS & Land Tenure 2 group to fulfill the ENGO 500 course requirements. The report details the team's work towards the goal of creating an Internet of Things (IoT) application that interfaces with the Open Geospatial Consortium IoT specification, as well as provides an evaluation of the team's progress, and estimates remaining work.

Within the first month and a half of the semester, the team has managed to post observations from a sensor-microcontroller combination to the OGC SensorThings API, create the framework for a web-server, build an interactive webpage for users to configure the solution in physical space, and finally receive and display observations from the server.

Some of the tasks that were intended to be finished at this time have crept passed their projected deadline, and evaluation of the reasons behind this are discussed within the report. Overall, the team has progressed well and a clear path to completion has been identified.

Thank you, Dr. Liang, for taking the time to oversee our project. Your efforts to uphold weekly meetings with the team have greatly increased our productivity. If you have any questions about the contents of the report, or concerns of any other nature, please feel free to contact one of the team members at your convenience.

Sincerely,

Group GIS & Land Tenure 2

Table of Contents

Index of Figures

Index of Tables

Introduction

Purpose & Scope

The purpose of this document is to summarize and describe the progress of group “GIS & Land Tenure #2” to date. This report describes in detail the work done by the group with regards to the planning process of the project, the development of a prototype smart-shelf as part of the project design, and the development of a user-friendly web-based interface with which to interact with our shelf prototype [1].

Upon completion of describing the work done thus far in the project lifetime, the project schedule and timeline is evaluated. Milestones accomplished and problems encountered are discussed and evaluated, and finally a risk-assessment for the future of the project lifespan is considered. A final summary of the project status is stated, followed by a brief conclusion on the work done and goals to strive for in the remaining life of the project [1].

Project Overview

As mentioned in previous reports, the main focus of this project is about the concept of “Internet of Things”. The “Internet of Things (IoT)” is a scenario in which objects, animals or people are provided with unique identifiers and the ability to automatically transfer data over a network without requiring human-to-human or human-to-computer interaction [2].

This project, in particular, will be focusing on developing a Location Aware Smart Shelf (LASS), that can track customer interests or purchases throughout a grocery store. This will be developed as a Free, Open Source Software (FOSS) for developers who are looking to develop an “Internet of Things” using the Open Geospatial Consortium (OGC) Standard. The OGC Standards are technical documents that detail interfaces or encodings. Software developers often use these documents in order to build open interfaces and encodings into their product [3]. Thus, the project comprises two major components:

1. The Sensor Prototype component: comprises the development of a prototype smart shelf that can send sensor readings to the OGC IoT SensorThings data service.
2. Software / Web Component: Allows a user to interact / acquire readings from one or more of these prototype shelves that are mentioned above.

Location Aware Shelf System (LASS) Prototype

This section describes all of the progress made on the LASS prototype since mid-December. It begins with an overview of the major electronic components we have acquired and have been using so far. Following that, a description of the OGC SensorThings API Data Model for IoT applications is given, specifically highlighting the relevant components for our application. The next sub-section breaks down the different use cases that have been identified and how they have been implemented so far.

Status of Hardware Inventory

At this point, the hardware necessary for preliminary testing has been acquired. All of the programming and electronic assembly has been primarily accomplished with the following equipment:

- 1 Raspberry Pi and peripherals (monitor, keyboard, mouse)

- 1 breadboard and Pi cobbler
- 1 PIR motion sensor designed by seeed

The equipment being used is shown in Figure 1:

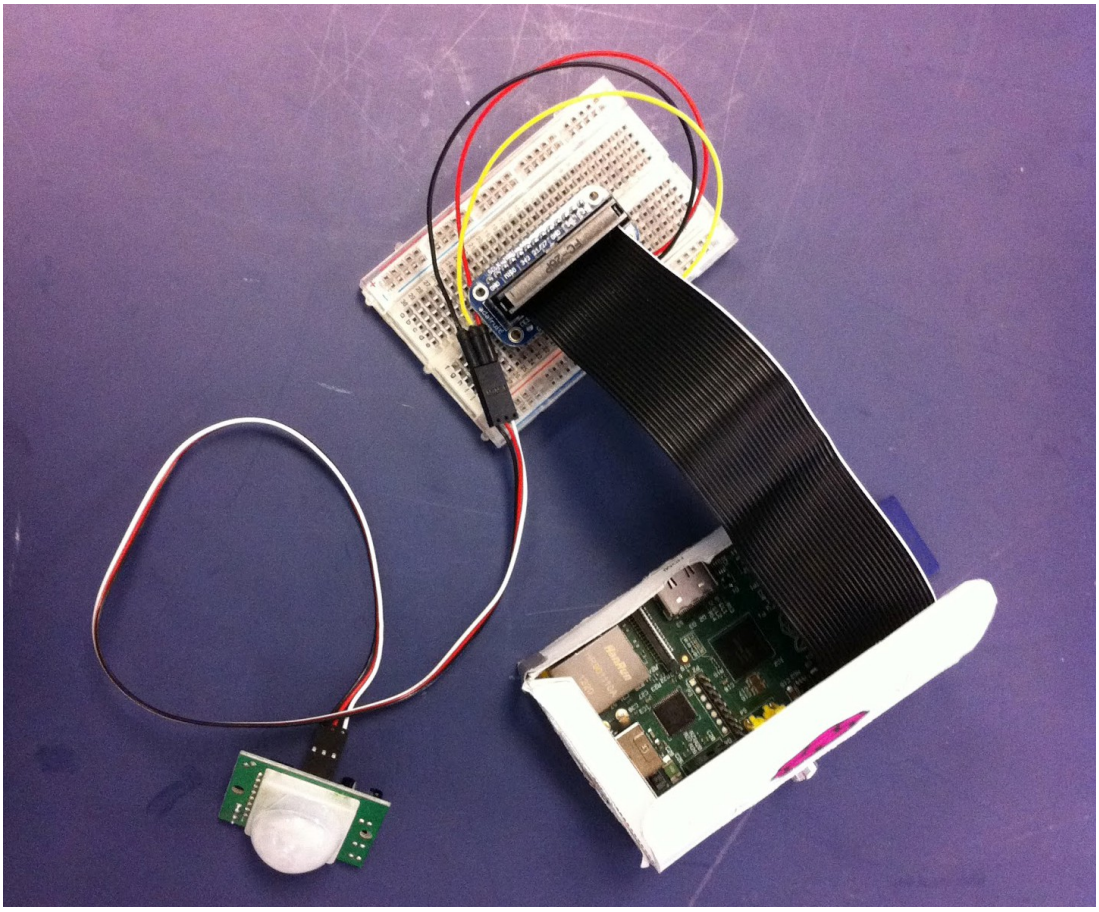

Figure 1: Current hardware being used, including a pseudo-infrared motion sensor, breadboard, pi-cobbler accessory and breakout cable, and Raspberry Pi

While the equipment lacks the physical shelf component, it has enough functionality to facilitate implementation of the key use cases. The current electronics setup was established based on following a tutorial found in the Adafruit Learning System [4]. This was used mainly to determine which GPIO pins should be used for the different PIR sensor wires.

Finally, additional sensors have been ordered, including more PIR motion sensors as well as photo interrupters, which will be used for detecting shelf stock. In this way, we have replaced our previous order list of magnetometers (which can measure any magnetic field) to photo interrupters, which will only pick up the electromagnetic signal of nearby elements through induction.

OGC SensorThings API – Data Model

One of the major requirements for this project is compliance with the OGC SensorThings API draft standard. As such, it is necessary for the prototype to fit into the defined IoT data model [5]. The entire IoT data model acts as a general umbrella which can encompass a large range of different IoT applications. Aside from the two core elements (a Thing and a Location), there are two main parts: the Tasking Profile and the Sensing Profile. Because the purpose of the Tasking Profile is to outline the data structure of a “Thing” which can be controlled by an application, this part of the data model

is not relevant to the LASS prototype. On the other hand, the Sensing Profile defines the components of an IoT application (both in terms of physical device and attributes) which can be acted on by four main functions – Create, Read, Update and Delete.

The general Sensing Profile, which is shown in Figure 2, has several entities. The entities which have been identified as primarily relevant and utilized in this stage of our project are highlighted in blue. In the context of our application, the Thing is the Raspberry Pi, the Datastream identifies/groups the motion sensor observations together and each Observation corresponds to a reading from the sensor. At this point, since only one sensor is being used, the Sensor entity has not yet been included. There is also no Observed Property entity either, because a description of the observed property is included in the Datastream and the units are not crucial in this application. Neither of the GeoJSON_Geometry objects (Location and Feature of Interest) were included either, since the location identification will be achieved via the web interface.

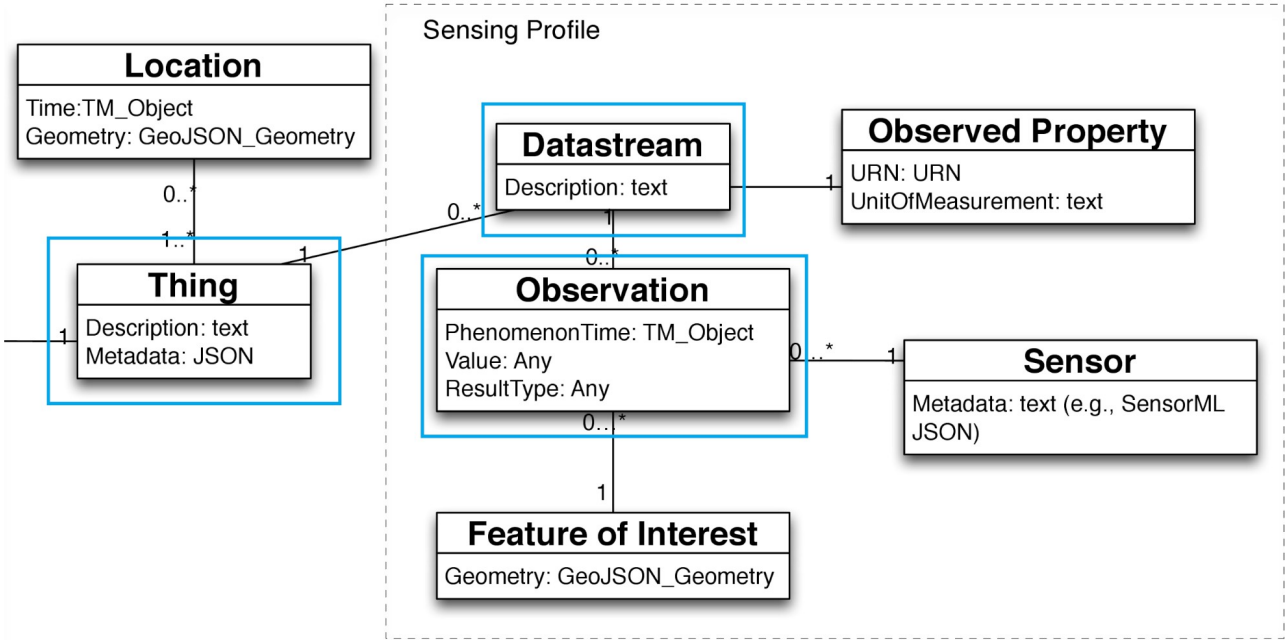

Figure 2: Core elements of OGC IoT data model and the Sensing Profile aspects, with the relevant attributes outlined in blue.

Use Case Implementation Status

There were several use cases identified and explained in the previous Technical Deliverables report [1], which have been implemented since then. As a reminder, the use cases are listed in Table 1 below, followed by an explanation of how they were implemented.

Table 1:Use cases identified and defined in [1]

1.1	Update database with formatted data
1.2	Format data according to OGC IoT standard
1.3	Uniquely identify sensors
1.4	Send data to Raspberry Pi

Use Case 1.1

The first use case identified for the LASS prototype was to post data to the data service, which is hosted at http://demo.student.geocens.ca:8080/SensorThings_V1.0/. Based on the components of the data model which were identified as relevant, this procedure was done in two main steps – initialization (corresponding to the CREATE function, or HTTP POST) and updating observations (corresponding to the UPDATE function). The first step creates the Thing with its related Datastream(s) and the second step creates Observations as they are read.

In the initialization step, a request is made to the data service to HTTP POST a "Thing", effectively creating a new Thing to the database. Because of the data service implementation, the new Thing is automatically indexed based on how many Things are existing already. Once the Thing is created, a response is returned to the program which can be used to determine its location (URL). Before posting, the new Thing must include information regarding how many Datastreams belong to that Thing.

In the updating step, the Datastream of interest must be identified (which can be done using the URL returned from the initial request). Once it is identified, Observations that are assigned to the Datastream can be posted at regular intervals. All of the above steps were implemented using a free Python library called *requests*.

Use Case 1.2

The data format defined by the OGC SensorThings API for each entity follows the structure of JSON objects. Each of the different entity types have some attributes which are required and others which are optional. For the purposes of the project at this stage, this use case implementation was mostly hard coded. This can be considered as a first iteration of the use case solution.

At present, the Time and ResultValue properties are dynamically updated. The Time property is updated using the built-in time class and its isoformat() function and the ResultValue is updated based on the sensor reading. In order to test, sensor observations were simulated (as explained in the Status of Code section).

Use Case 1.3

Although the PIR sensor can be identified by the functions implemented in use case 1.1, we have yet to assign it an official identity. This will be done by consciously designing the electronic layout such that a sensor can be identified according to the GPIO pin it is connected to.

At the moment, this use case is not a great concern, since we are only working with one sensor and one datastream. Our goal is to create a data streaming process so that it can be scaled; in other words, if the process is fully functional with one sensor, it can easily be expanded across several more. We aim to expand our sensor inventory over the next month, and once we do so, this use case will be of more importance and will be completed.

Use Case 1.4

The fundamentals of progress in this area were easily completed with the help of studying several online projects posted by others, such as Matt Hawkins, [6] as a way to observe and understand the GPIO classes and functions. These libraries were previously unknown to the group members and as such had to be researched.

Several different methods of relating data from the PIR sensor to the Raspberry Pi via GPIO pins were tested and the most applicable libraries were chosen. These allow the sensor to receive data from it's surroundings and relay it to the Raspberry Pi, where it can then be recorded, formatted and

posted to the network as per the other use cases. Currently we are able to make use of all the basic functions of the sensor.

Status of Code

t this point in time the group has created an overall python class that encompasses all of the use case goals (Table 2). Using this class, referred to as class “data”, the group has been able to:

- Record and format the datastream as required by use case 1.2
- Update the formatted data according to use case 1.1
- Send data to the raspberry Pi as specified by use case 1.4
- Use case 1.3 is included in 1.1 at the moment.

Table 2: UML Representation of “data” class

Class: data
ID: string
rootURI(): root URI *localtime():* time in local format *string():* string of time in required format *timeisoformat():* time in ISO 8601 format *initRequest():* creates and posts Thing to data service *sendObs(observation):* sends observation and time to data service

Although the use cases and the class “data” are functional, when seen from the perspective of actual implementation, they are still considered to be in a skeleton format. There are many checks and design modifications that must be added to make them robust which will be discussed in the risk section below.

Simulation File

When working with sensors and data streams, real-time testing is vital and yet can also be problematic when running the code. The sensors may not be ideally sensitive and real world conditions can interfere. It is for this reason we have created an additional python file that generates random values and ultimately simulates the PIR sensor, as well as accessing the “datetime” library in real time as the sensor would. This facilitated testing when posting data to the network as needed by use case 1.1. This simulation file will likely not be used after this point, now that use case 1.1 has been fully tested and is considered complete.

Website (Interface) Development

The website interface development has also made progress in implementing the use cases defined in the technical deliverables report. Table 3 outlines these use cases, with the details available on the project wiki [7].

Table 3: Use cases identified and defined for website

2.1	Assign location to a micro-controller
2.2	Configure a store layout
2.3	Get aisle traffic observations
2.4	Get facing observations
2.5	Get aisle traffic
2.6	See if products are faced
2.7	Show map UI
2.8	Log In/Authenticate
2.9	Become System Configurator

To facilitate the implementation of these use cases, the first necessary step was to construct a web-server to suit the needs of the project. Due to the web-server being behind the scenes, with actors not interacting with it directly, the work done cannot be assigned to a single use case. Rather, it forms the framework for all of them.

Work on specific use cases has begun in the form of the Store Layout Creator (use case 2.2). The Store Layout Creator is a web page that allows users to make a digital representation of a physical store. Along with use case 2.2, this work also contributes to the following use cases:

> **2.1** – With the store layout configured, a microcontroller can be assigned to a shelf, which has a location. The ability to edit shelf attributes in the store layout accomplishes this use case, as this allows the microcontroller to be mapped to a shelf. At the time of writing, this is done by supplying the URL of the microcontroller on the OGC SensorThings API.
>
> **2.5 & 2.6** – Similar to assigning a location to a microcontroller, sensors can be assigned to sections within shelves. This gives them a location on the map UI, which accounts for part of these use cases. The remaining work will be to visualize observations on the map.
>
> **2.7** – Since the store layout creator visualizes the map as it is being built, the majority of work in this use case is done. The remaining work will be to render the visualization on a separate page, which can be accomplished once the store layout is saved to a database.

Work has also began on use cases 2.3 & 2.4, in the form of a demo application. The purpose of the demo is to get and display data from the OGC SensorThings API server. While the demo does not specifically provide any functionality for the final product of our website, the code can be easily tweaked and re-used after demonstration. The details of the work completed on the website interface is explained in the following sections.

Web-server Development

In order to effectively serve and store content for the proposed website, a small server needed to be created. The purpose of such a server was twofold: we needed something that could serve dynamic pages in a clear-cut fashion, and likewise needed a way to respond to user requests from the server (such as in the case of adding sensors or shelves to profile). Thus, development of a simple web-server began. The requirements of such a server were:

1. Must be able to load content dynamically – for this reason, a templating language or specification for HTML content was desired
2. The web-server should ideally be capable of connecting to a database of some kind (Redis, MongoDB, or PostgreSQL) in order to store user data and handle log in / register requests.
3. The code for the server should be split up in a modular fashion. In particular, a model-view-controller or MVC approach is preferred, to make development and deployment rapid and easily iterable.

Working off of the requirements above, a simple web-server was implemented using the Node.js runtime [8] while using the Express.js framework [9]. Using Express.js made development of the server incredibly easy, and allowed the group to maximize the amount of time spent on the functionality of the website itself, while minimizing time wasted trying to serve content. To satisfy the first requirement above, Mustache templating [10] was likewise used, for its simplicity and the ease of which it could be integrated into Express.js. Done in this way, the potential learning curve of various web technologies was reduced.

Of particular note is that there is not yet any specific support for a database, nor is there currently any way to store user data. A few approaches were researched, such as implementing a PostgreSQL database to return JSON fields, or even running a Redis server (NoSQL) to store user data. The primary roadblock was that neither were easy to set up while the extent of the user data required to be stored was unknown. While this could have been implemented multiple different ways, this particular subject has been avoided for the time being in favour of placing more effort towards developing the basic functionality of the website. Given more time in the future, this avenue will be explored more thoroughly. Overall this was determined to be a minor risk given that the primary functionality of the website will not differ significantly as a single-user system vs. as a multi-user system.

Lastly, in an attempt to prevent clutter of code within the ENGO500 repository [11] on Github, the group has decided to split the website and server code from the main repository. All of the website functionality and code can now be found in the ENGO500-web-server repository [12], likewise available on Github.

Store Layout Creator

Due to differences in physical layouts between stores, the web application required a means to create a digital representation of a specific physical space. Not only does this representation need to make sense in terms of data, but also requires a visual component. Once the store layout is configured, it will be used to display observations such as an activated PIR motion sensor or a shelf that is low on stock. To accomplish this, a combination of JavaScript, jQuery, and D3.js were used. JavaScript and jQuery were used to create and manipulate the objects that define the store layout, as well as organize them into an easy to configure manner. D3.js was used to visualize these objects in a way that reflects the physical world. Together they make a coherent system which will facilitate the initial set-up of the LASS in a real-world space. The creation of such a system went through 2 major iterations to reach the state it is at the time of writing. The following section details the two iterations by providing background information, design choices, and how the systems work.

Technologies Used

jQuery

jQuery is a JavaScript library that simplifies HTML document manipulation, event handling,

animation and Ajax. This allows for the creation of dynamic web pages in a much simpler manner than that of using straight JavaScript. Used by 57.7% of all websites, it has the largest market share of JavaScript libraries at 93.1% [13]. It was chosen due to the library being free, open source and very prevalent. The API is well documented and there are many tutorials and help references available.

jQuery UI

jQuery UI is a set of user interface elements and themes that build on top of jQuery's feature set. For the store layout creator, the accordion widget was used to house an editable representation of the underlying objects. This widget is a collapsible menu which can hold a lot of content while minimizing scrolling.

D3.js

D3.js is also a JavaScript library, but it exists for a different reason than jQuery. D3's main use is for visualizing data within web pages. It does this by generating and styling SVG graphics which can include transitions and other dynamic effects. It was chosen for the project due to it's ability to portray GIS data while utilizing web standards all while being free and open source.

Jeditable

Jeditable is a jQuery plugin that allows in-line text editing. It was chosen due to it providing the exact behaviour needed for the store layout creator. It is provided with an MIT license.

Underlying Data Structure

While the exact data structure has yet to be determined, it has been implemented in such a way that editing or adding to it is trivial. As it exists at the time of writing it is as follows:

```
Shelf:{
      shelfName: ,
      notes: ,
      rpUUID: ,
      sections:[{
            sectionName: ,
            displayId: ,
            displayColor: ,
            pirURL: ,
            pintURL: ,
      }],
}
```

An array of shelves is initialized when the store layout creator is opened, and shelves & sections are added to it as needed. Shelves serve as a container for sections, and also have a number of attributes such as a name, and an attribute used to match the shelf to a raspberry pi. Sections are for organizing products within the shelf. An example of this organization could be a shelf holding snack food, with sections for chips, nuts, cookies, etc.

Iteration 1: Using Solely D3.js

When the task was initially taken on, an attempt was made to provide a solution using only D3.js and JavaScript. This choice was mainly made due to unfamiliarity with D3.js. It was known that D3.js would be able to provide the level of manipulation that the team wanted, but the complexity

of providing such a solution was not known. The details of this implementation follow.

Figure 3: Greeting screen

Figure 3 above shows the initial look of the first iteration layout creator. Users were greeted with a mostly blank screen with a single enabled button in the bottom left. Clicking this button would add a shelf to the screen, as seen in Figure 4:

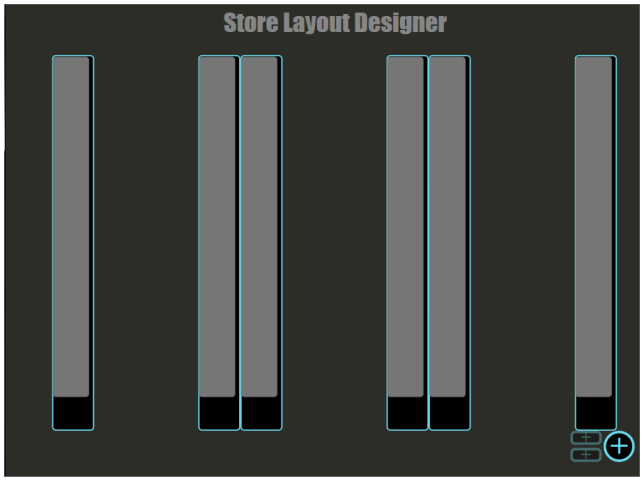

Figure 4: Shelves Added

The black rectangles with the blue outline represent shelves, and the grey rectangles represent a section within that shelf. The spacing of these rectangles is not correct for a finished product, but served as a prototype due to the design being abandoned shortly after. At this point users were intended to click on a shelf, which would highlight it as shown in Figure 5.

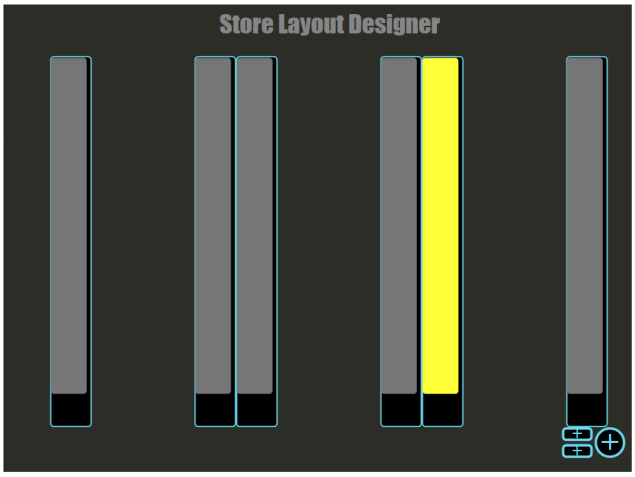

Figure 5: Section selected

Additional controls become activated as seen in the bottom right hand corner. These two stacked buttons allow the user to add additional sections to the shelf. Selecting a shelf was also meant to bring up controls for configuring that section's attributes. This step of the implementation is ultimately what changed the course of the development to include a way of manipulating the layout outside of an SVG interface.

The last piece of functionality implemented was that the existing section would resize to half of the available space within a shelf when one of the add section buttons was clicked (see Figure 6). The next step would have been to generate a new section and display it in the empty space.

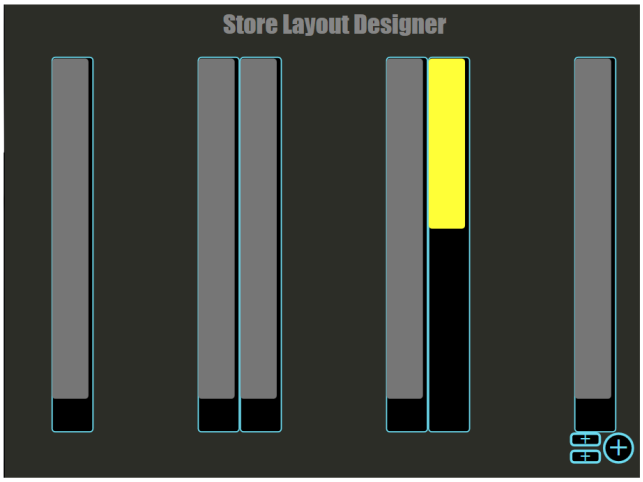

Figure 6: Section Split

Along with the aforementioned issues with this iteration, another design issue is that there is very little guidance given to the user. These qualms prompted the second iteration design.

Iteration 2: jQuery and D3.js

One of the first things tackled by the design of the second iteration was the lack of direction given to the user. This was done by simply designing a better HTML layout. Figure Error: Reference source not found shows the store layout screen as it appears before the user interacts with it. Already there is improvement in that there are some text instructions and the '+ Shelf' button is labeled rather than purely graphical as seen in iteration 1. The page was structured into a 30/70 split to accommodate both the manipulation and visualization elements.

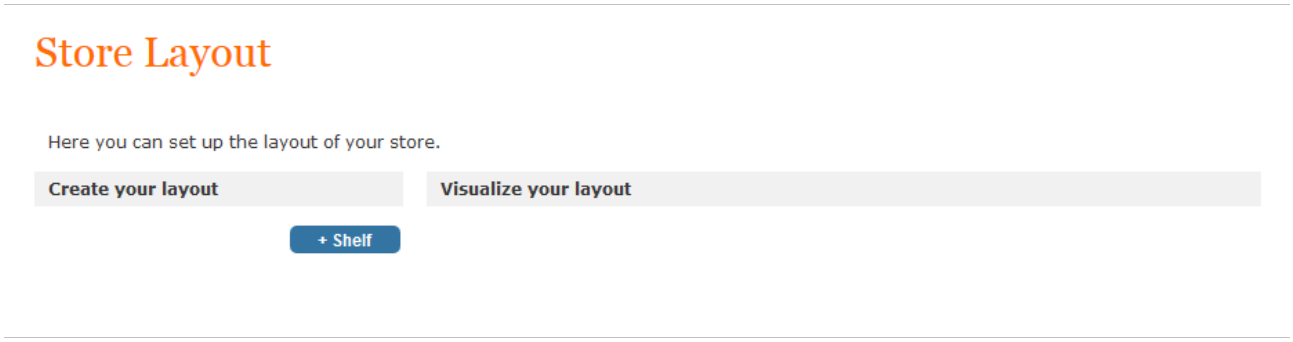

Figure 7: Second greeting screen, using jQuery and D3.jjs

Clicking on the '+ Shelf' button allows users to add a blank shelf element to the screen. This is represented both by an accordion element in the left pane, and an SVG element in the right pane, seen in Figure 7. Multiple shelves can be added, and the spacing of the shelves will adjust automatically.

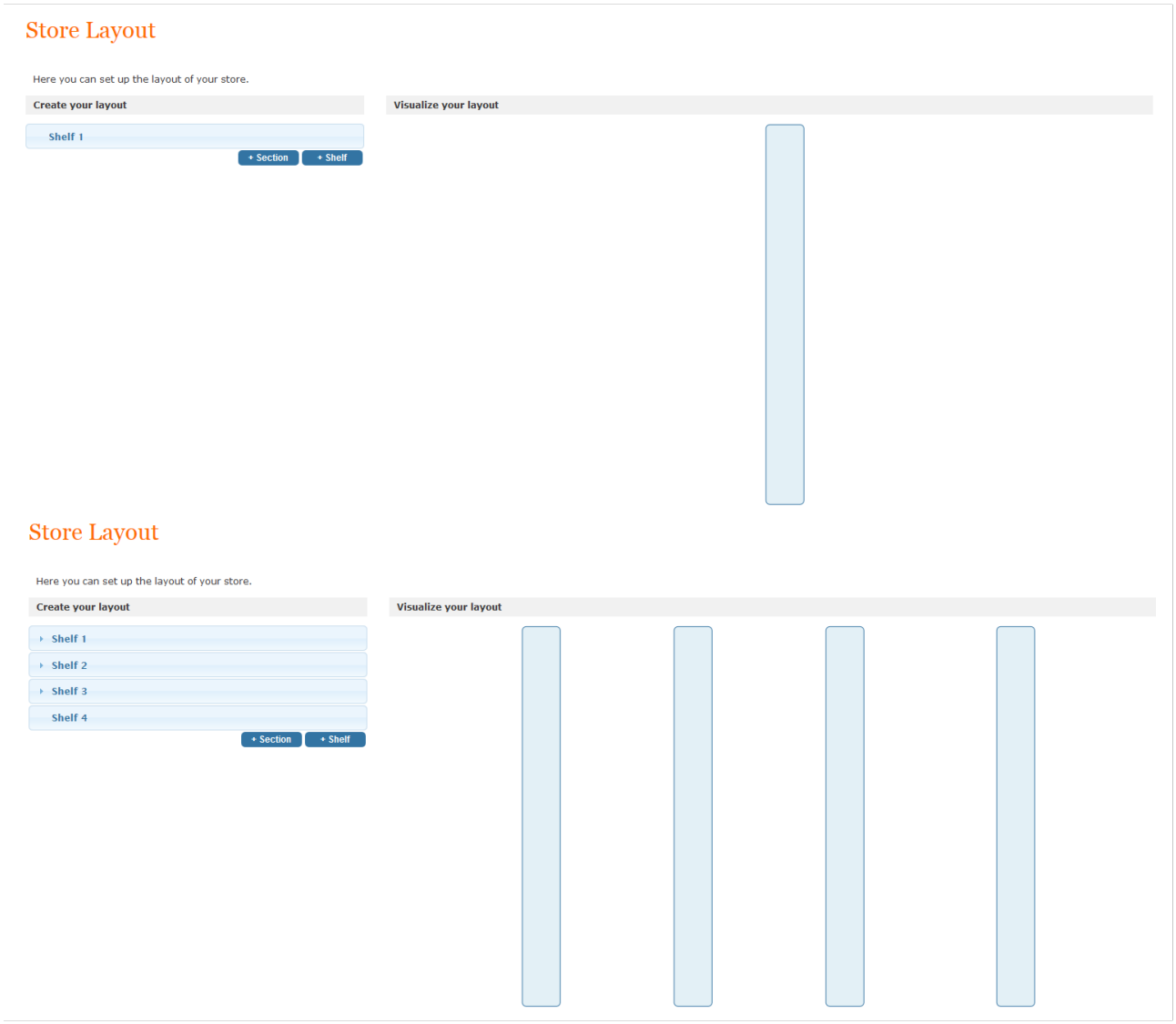

Figure 7: Single and multiple shelves added

Opening a shelf accordion element in the left pane shows the existence of the shelf attributes (Figure 8). When the shelf is first created, this attribute section is automatically generated. With a shelf selected (accordion is open) a user can then add sections to the shelf using the '+ Section' button. Similar to adding shelves, both an accordion element and an SVG element will be generated which can also be seen in Figure 8:

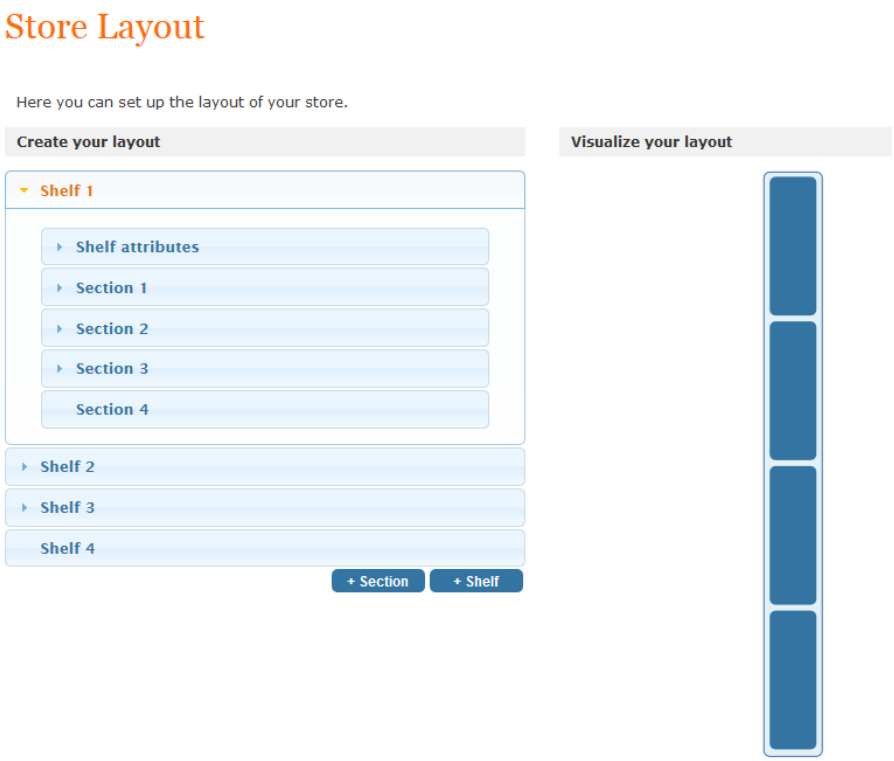

Figure 8: Shelf with attributes and four sections

If a user clicks to open one of these sections, they are shown the attributes belonging to either a section or a shelf. When the object is first created, these variables are initialized but do not hold a value. They are displayed with the message 'Click to edit' which informs the user how they should interact with the element. This can be seen in Figure 9, below:

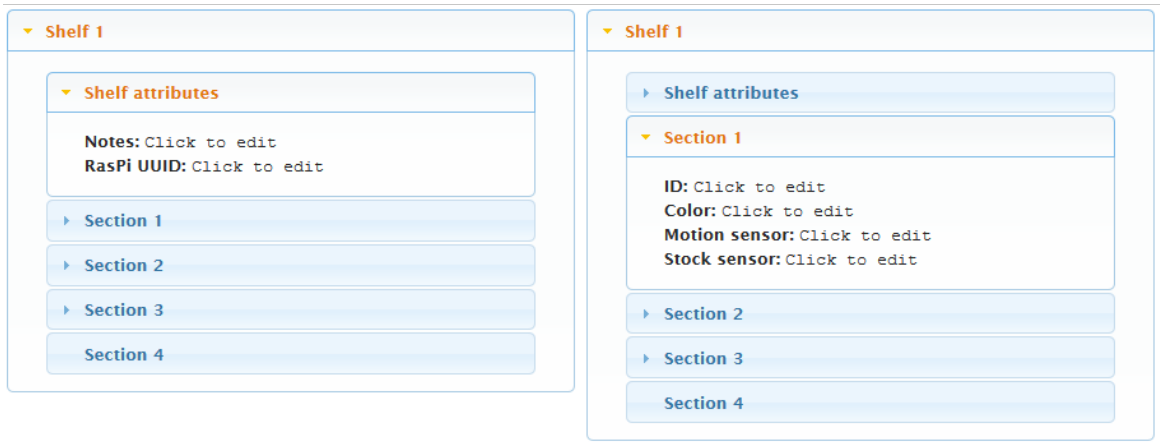

Figure 9:Editable attributes within accordian

Clicking on one of them will enter an editing mode. The user can save their input by pressing enter, or cancel by pressing escape or clicking outside of the text box. Once saved, the underlying object

will be updated to retain the input value. Figure 10 shows this process.

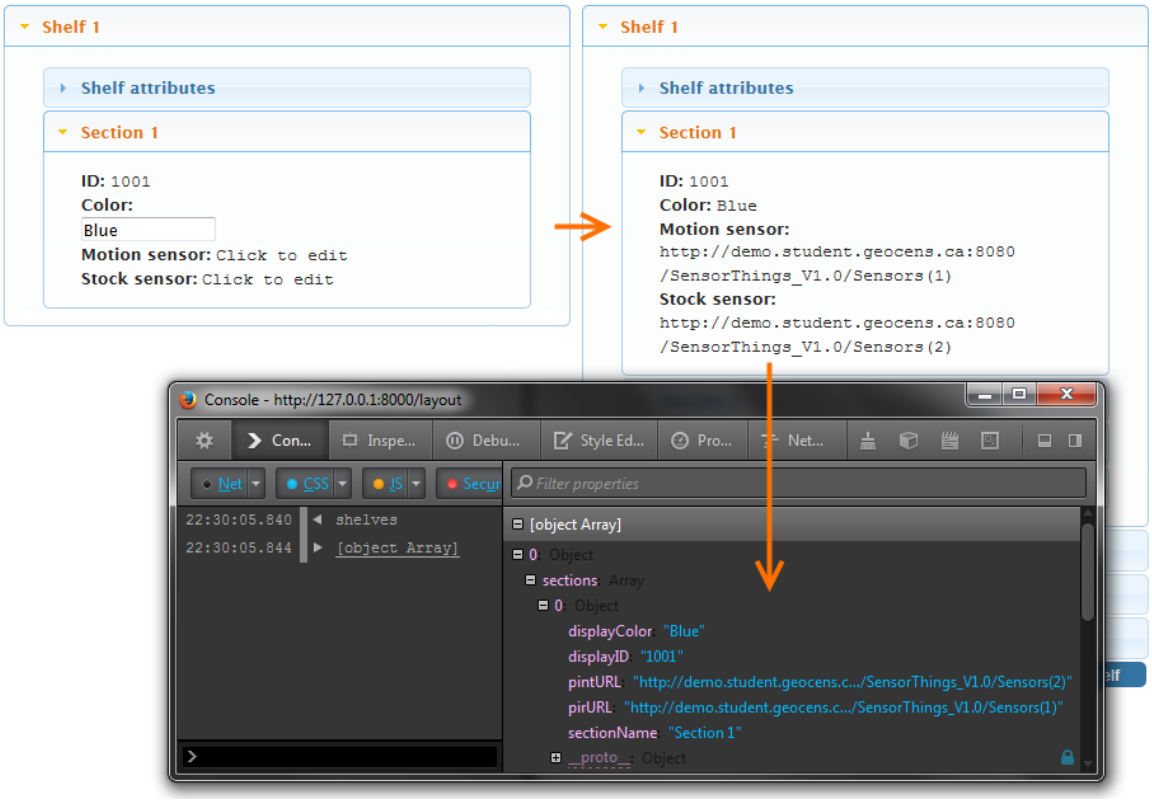

Figure 10: Editing attributes and reflected change within data structure

When fully configured, the user is left with a layout like the one shown in Figure 11. While this system does not yet contain the full feature set, the core functionality exists. Additional enhancements will be detailed in the schedule section.

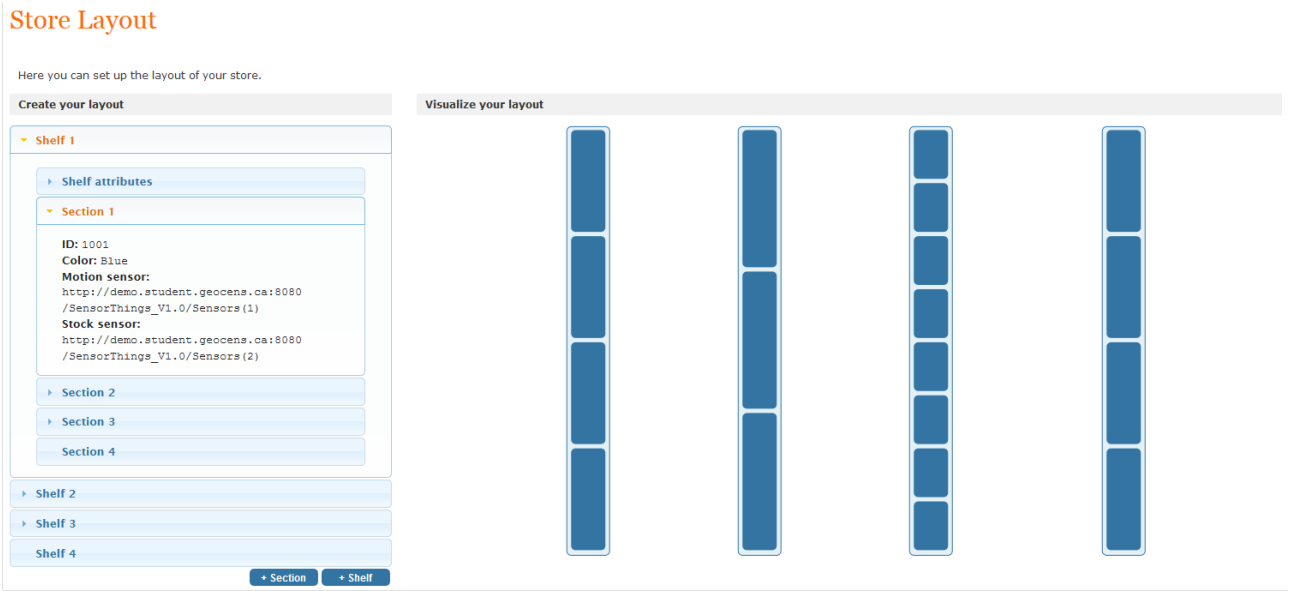

Figure 11: Example of a store layout

Demo – Data Relay

As noted in Use Cases 2.3 and 2.4, we need to be capable of pulling data from the OGC IoT SensorThings API dynamically in real time. As a partial exercise to show off this functionality, and likewise explore the data on the server in more detail, a small demo was created. This demo pulls all of the data for each 'Thing' on the server, and organizes it into an interactive force-graph using D3.js. An example screenshot of this demo can be seen in Figure 12 below:

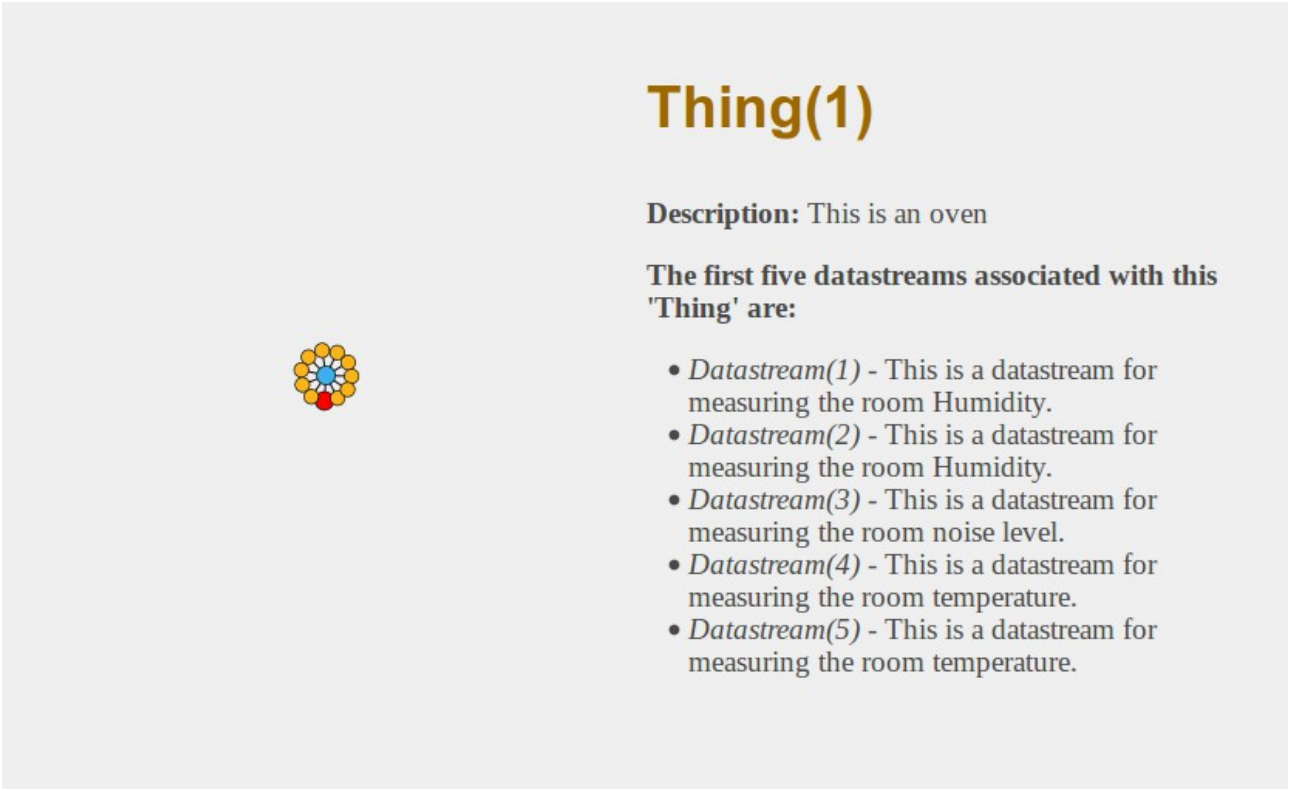

Figure 12: Example of force-graph demo pulling data dynamically from server and displaying it alongside the graph

While the above demo code doesn't specify the exact way that such code will be used in a production setting, it does provide an interesting visualization of the uploaded objects into the API. What's more, since this demo loads dynamically when the page is loaded, the visualization changes as objects and observations are added to the server. The group expects the transition from the above code to production code to be trivial in nature, once the specific functionality is required by other pages / modules.

Schedules, Risks, and Status

Schedule

Since the previous technical deliverables report, much effort has been put into finalizing our use-case definitions, and likewise working towards implementation of our individual use-cases. Overall, we have almost entirely (~70% - 80%) implemented most of our use-cases with regards to our prototype development, and have for the most part implemented four of our nine (~40%) proposed use-cases with regards to the website.

Major roadblocks regarding use-case implementations can be attributed to two primary factors: delays with the shipping of our hardware / sensors, and difficulties in determining the simplest way to implement basic authentication and database storage within the web-server. While it appears that

few of the use-cases have been implemented on the web-server side, it is important to note the distinction that many of the use-cases depend on the primary functionality provided by use-cases 2.2 and 2.7, which have currently received the most attention to date. Moreover, much of the code regarding many of our use-cases can be tweaked and re-used in other use-cases as well (e.g. displaying the map UI is necessary in both 2.2 and 2.7, and largely uses the exact same data elements). To see a full layout of how we expect these delays and roadblocks affecting the overall progress of our final project deliverables, see the modified Gantt chart shown in Figure 13.

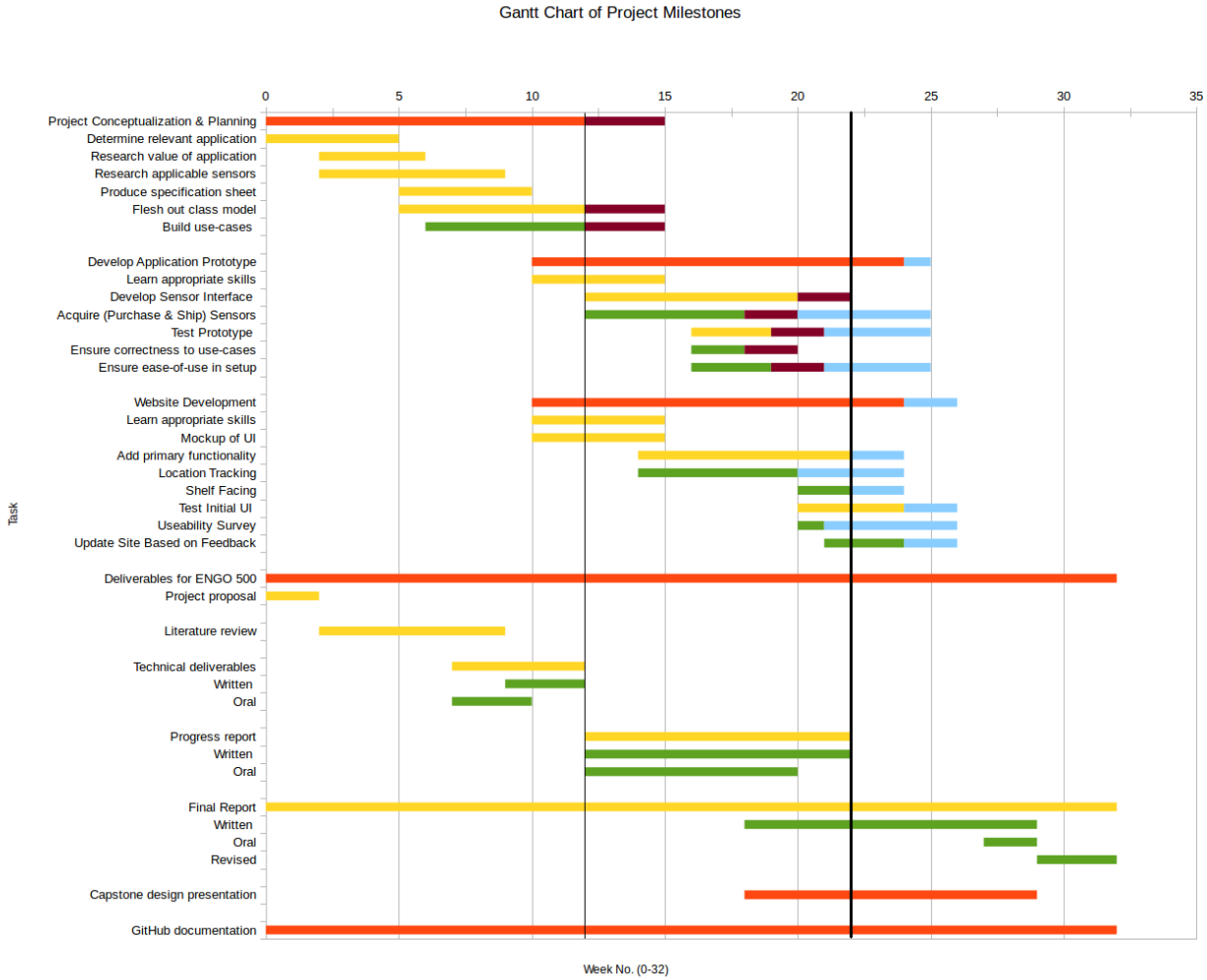

Figure 13: Modified Gantt chart. Thin bar represents progress as of Technical Deliverables report, thick bar represents progress as of this report. Blue areas are potential delays in elements of the project as discussed in the risks section below

As seen in the Gantt chart above, the largest effects relate to post-development and usability testing of the prototype and website. Given the iterative approach to our development, bugs have been dealt with as they appear, and most of our completed functionality is bug-free. A more detailed outline of risks can be found in the subsequent Risks section.

Future Work

Given the schedules and progress outlined above, the prototype development will aim to accomplish the following goals in the coming weeks/months:

LASS Prototype Development

- Add user-friendly features, such as having an on/off switch and status indicator LEDs
- Modify data collection program to take real data from a PIR sensor once new sensors arrive. The group also has the option of attempting to correct the spastic-nature of our current sensor
- Generalize class “data” for more than one datastream or sensor type
- Include ID tag specific to each Raspberry Pi. This will most likely be hardcoded
- Finish implementing use-cases completely (final 20%-30%)
- Develop and finalize physical prototype (requires that sensors be shipped)

Website (Interface) Development

- Enhance interactiveness of shelf layout (tooltips, colours and IDs, click section to open appropriate accordion)
- Make editing easier (if you select a raspberry pi for the shelf, maybe make the sensors belonging to the Raspberry Pi that appear in a dropdown for the sections, dropdown for colors, etc).
- Change spacing of shelves to reflect real world (2 grouped)
- Fix known bugs
- Save configuration to database
- Add ability to load configuration (user data) from database
- Ability to delete in layout editor so we adhere to CRUD design
- Finish implementing use-cases

Risks

Several risks that could become a threat to the proposed timeline for the remainder of our project are considered below. Primarily, there is always a chance that the sensors we have ordered through our advisor may arrive later than expected. This risk is possibly the most threatening since is it not completely within our control to prevent. However, our hardware is currently shipping and is expected to arrive within the next week or two, which still allows for plenty of time to test and work with the hardware provided that they do not arrive late.

Furthermore, there is a potential risk that the ordered sensors may not function as described upon time of purchase. This has already caused some concern among the group, since the PIR sensor we are currently working with is proving to be over sensitive, which we did not expect. Several solutions are being considered to counteract its oversensitivity. One possibility to mitigate is a redesign of our sensor positioning, such as placing it on the ceiling to remove vibrations caused by the environment of the shelf face.

Another risk to consider lies in the robustness of the connectivity. Connection to the network is necessary to pass the datastream to the website, and if a connection cannot be made for any reason, the user should be notified of the problem. Currently there is no alert for this sort of error. The group will create checks that will report such an error, although it may not be able to identify and account for the vast variety of potential problems. However, the priority is not fix or identify the connection error, but merely to alert the user of a problem. With this in mind, this risk is

approachable in our remaining time frame, and does not pose much of a threat. Moreover, given that in production the system would likewise be under a wired ethernet connection or a closed wi-fi network, network stability becomes less and less of a problem.

With regards to the website development, the biggest risk involved is in not fully implementing all of the functionality of the website. Specifically, with regards to a multi-user system, this may not be possible within the time-frame. More research currently needs to be put in to explore this topic in more depth, and find the optimal solution that can be implemented within time. On the grand scale of the project, this risk isn't too important, as a single user system with the remainder of the basic functionality implemented would be sufficient as a first iteration of the project requirements.

As mentioned previously in the Schedules section, because of delays in other areas of development, some of our usability testing may not be completed, or may be delayed significantly. Given the simple nature of the website functionality and our iterative development process, however, we believe that this will not present significant hurdles in terms of usability for our final site, as we are performing iterative testing and filing bug reports in code as it is being developed.

Our final concern regards downloading and displaying massive bouts of data from the server. Because the server has refreshed itself numerous times since we have started the project, we have not had any chance to download and test the speed of the server given a large data set. This may or may not present problems when we aim to show a map of observations over a store layout, as we do not know how reliable or easy it will be to retrieve and handle large quantities of data from the server.

Risk Mitigation

In order to mitigate the risks outlined above, some changes to the group focus and our methods have been considered:

1. Much of the prototype development is ahead of schedule, despite the sensors shipping late. For this reason, we believe it to be in our best interests to move some of the man-power off of the prototype development and move them to assist with building the basic functionality for the website.
2. As previously mentioned, changing the sensor configuration from the shelf to perhaps on the roof or another area has the potential to mitigate some of the issues experienced with the PIR sensor sensitivity.
3. Future focus will be more oriented towards developing the basic functionality of the website. With #1 above, the group believes that we can mitigate any fears or risks that we may encounter regarding not being able to implement certain features, such as a proper login / authentication system.

Conclusion

The project has progressed considerably since the Technical Deliverables Report was written and submitted. Both components of the project (shelf prototype and web interface) have been significantly developed, with various sample applications/programs having been created to demonstrate functionality.

In terms of shelf prototype development, all four use cases previously identified have been implemented to usable, albeit not robust, functionality. Use case 1.1, which addressed uploading data to the data service, was a key link between the two sides of the project. The Raspberry Pi can now successfully register itself as a Thing in the data service, and can post observations to a designated datastream. Knowledge from this process can be extended to more sensors when they

arrive. The other three use cases (1.2-1.4) primarily address functionality which occurs on the Raspberry Pi locally, and have been achieved with some manual effort. For example, the Raspberry Pi can identify a sensor based on which GPIO pin it is attached to, but this should be established by the device creator (and not modified by the client). Reading the data from the sensor was achieved by using the GPIO pin library, and formatting the data according to the OGC IoT SensorThings API was accomplished based on examples in the OGC IoT Interactive SDK. From here, the work will be extended to include different sensors and to increase the robustness in implementation.

As for the web interface side of the project, progress had been made in both creating a web-server and also implementing use cases. The web-server technologies chosen were a node.js web-server using an express.js framework. These were chosen for ease of implementation as well as flexibility between client side and server side code. The remaining work in this area will be to implement a database to store user settings and content. In terms of implementing use cases, significant progress has been made on a Store Layout Creator (use case 2.2) and the use cases it supports (2.1, 2.5 – 2.7). At this time, it is possible for a user to create a representation of their physical store in a dynamic environment that allows editing of attributes and also provides a visualization of the underlying objects. The team has also implemented as much of use cases 2.1, 2.5 – 2.7 as possible, with most of the remaining work waiting on implementing a database to save settings. Additionally, use cases relating to getting data from the server (2.3 & 2.4) have progressed as the team is able to get and display information from the OGC SensorThings API. Outstanding use cases on the web interface include those dealing with user management, such as being able to log in with different users. These have been deemed low priority in favor of getting a single user system working with full functionality.

Based on the progress made on both the sensor prototype and the web interface, the team is confident that we will be able to produce a working implementation of our design concept which is compliant with the original project requirements.

References:

[1] Jeremy Steward, Alexandra Cummins, Kathleen Ang, Ben Trodd, and Harshini Nanduri, “ENGO 500: GIS & Land Tenure #2 - Technical Deliverables Report,” University of Calgary, Technical Deliverables Report, Dec. 2013.

[2] Margaret Rouse, “What is the Internet of Things (IoT)?,” *whatis.com*, Jul-2013. [Online]. Available: http://whatis.techtarget.com/definition/Internet-of-Things. [Accessed: 05-Dec-2013].

[3] “OGC Standards and Supporting Documents,” *OGC Standards and Supporting Documents*. [Online]. Available: http://www.opengeospatial.org/standards. [Accessed: 06-Dec-2013].

[4] Adafruit, “Hardware | Adafruit’s Raspberry Pi Lesson 12. Sensing Movement | Adafruit Learning System,” *Hardware | Adafruit’s Raspberry Pi Lesson 12. Sensing Movement | Adafruit Learning System*. [Online]. Available: http://learn.adafruit.com/adafruits-raspberry-pi-lesson-12-sensing-movement/hardware. [Accessed: 13-Feb-2014].

[5] “OGC SensorThings API,” *OGC SensorThings API*, 2013. [Online]. Available: http://ogc-iot.github.io/ogc-iot-api/. [Accessed: 05-Dec-2013].

[6] Matt Hawkins, “Cheap PIR Sensors and the Raspberry Pi - Part 1 | Raspberry Pi Spy,” *Cheap PIR Sensors and the Raspberry Pi - Part 1 | Raspberry Pi Spy*, 23-Jan-2013. [Online]. Available: http://www.raspberrypi-spy.co.uk/2013/01/cheap-pir-sensors-and-the-raspberry-pi-part-1/. [Accessed: 13-Feb-2014].

[7] Ben Trodd, Alexandra Cummins, Jeremy Steward, Kathleen Ang, and Harshini Nanduri, “Use Cases - ThatGeoGuy/ENGO500 Wiki,” *Use Cases - ThatGeoGuy/ENGO500 Wiki*. [Online]. Available: https://github.com/ThatGeoGuy/ENGO500/wiki/Use-Cases. [Accessed: 13-Feb-2014].

[8] “node.js,” *node.js*. [Online]. Available: http://nodejs.org. [Accessed: 13-Feb-2014].

[9] “Express - node.js web application framework,” *Express - node.js web application framework*. [Online]. Available: http://expressjs.com/. [Accessed: 13-Feb-2014].

[10] “{{ mustache }},” *{{ mustache }}*. [Online]. Available: http://mustache.github.io. [Accessed: 13-Feb-2014].

[11] Jeremy Steward, “ThatGeoGuy/ENGO500,” *ThatGeoGuy/ENGO500*. [Online]. Available: https://github.com/ThatGeoGuy/ENGO500. [Accessed: 05-Dec-2013].

[12] Jeremy Steward, “ThatGeoGuy/ENGO500-web-server,” *ThatGeoGuy/ENGO500-web-server*. [Online]. Available: https://github.com/ThatGeoGuy/ENGO500-web-server. [Accessed: 13-Feb-2014].

[13] “Usage Statistics and Market Share of JavaScript Libraries for Websites, February 2014,” *Usage Statistics and Market Share of JavaScript Libraries for Websites, February 2014*. [Online]. Available: http://w3techs.com/technologies/overview/javascript_library/all. [Accessed: 13-Feb-2014].