Groundstation: Design

Team #25
High-Altitude Rocketry Challenge
Natasha Anisimova
Terrance Lee
Albert Morgan

Abstract

The *Groundstation* software will collect telemetry from a rocket while it is in flight and graphically display the telemetry in real-time. Groundstation is made up several different components: collection of data, storage of data, interpolation of data, and display of data. This document will describe in detail the design of Groundstation.

CONTENTS

I Introduction 3
I-A Identification . . . 3
I-B Date of Issue . . . 3
I-C Scope . . . 3
I-D Stakeholders . . . 3

II Structure View 3

III Interaction View 9

IV Interface View 11

V Resource View 14

VI Revisions 14

VII Glossary 17

References 19

I. Introduction

A. Identification

This document will provide and in-depth software design description of the Groundstation software package.

B. Date of Issue

This document is issued on December 2, 2016. At this time, the design of the software is complete, but no work has begun on the implementation.

C. Scope

This document will provide details about the Groundstation software package. The concerns of the software will begin when the telemetry is received via the serial port on the Raspberry Pi. Collecting the data from the rocket and converting it into a protocol readable through the serial port is the concern of the Oregon State University (OSU) American Institute of Aeronautics and Astronautics (AIAA) High-Alititude Rocketry Team avionics section.

D. Stakeholders

The stakeholders of the Groundstation software package are members of the OSU AIAA High-Altitude Rocketry Team. The OSU AIAA High-altitude rocketry team is a group of multidisciplinary engineering students who are working to together to build the rocket. The stakeholders require the ability to:

- View the telemetry in real time in order to track the altitude and aid in recovery
- View the data graphically so that it is easy to understand
- Log the data so that it may be analyzed at a later date

II. Structure View

PC
Entity
See *Interaction View - PC*

Package Manager
Entity
Author: Albert Morgan
Type: Subprogram
Purpose: The package manager will install, track, and update software dependencies on the server.
Contents: Because Groundstation will be using Node and JavaScript for both the frontend and backend, Node Package Manager (NPM) will be used [1]. NPM has a large repository of both server-side and client-side JavaScript packages.

Frontend
Entity
Author: Natasha Anisimova
Type: Component
Purpose: User Interface
Contents: The frontend of the web server will create and display the information that it was given by the backend. Given that the information has to do with location (longitude, latitude, and altitude), a 3D display is preferred. The frontend will consist of programs using 3.js and WebGL to create the visualization necessary to quickly understand the location of the rocket.

Backend
Entity
Author: Albert Morgan
Type: Component
Purpose: The purpose of the backend is the facilitate communication between the rocket and the user.
Contents: The backend takes care of all data collection, transformation, logging, and serving that data to the user. A Node server will handle reading the data from the serial port, transforming the data into JSON, storing the data, and making the data available to the user over the wireless network.

Node
Entity
Author: Albert Morgan
Type: Subprogram
Purpose: Node is the software that runs the backend.
Contents: The backend will run on Node. Several choices were considered to run the backend, particularly Node [2], PHP [3], and Ruby [4]. The backend was chosen two criteria:

- Speed. The backend will run on a Raspberry Pi, so speed is important to minimize the system resources used.
- Interoperability. The backend needs to work with multiple components, including the logging software, the data coming in from the serial port, and serving the frontend to the user.

Node uses an event driven architecture, which is ideal for reading data from the serial port. PHP and Ruby on Rails are both HTML preprocessors, so they don't get activated until the web site is requested. Node, on the other hand, runs continuously in the background and uses an event driven architecture. The event driven architecture is ideal for reading the data from the serial port. Additionally, Node much faster than either ruby or PHP.

Serialport
Entity
Author: Terrance Lee
Type: Library
Purpose: Node serialport library
Contents: The Node serialport library allows us to transfer data from one device to another.

Log
Entity
Author: Albert Morgan
Type: Data store
Purpose: Store the data in non-volatile storage for later retrieval and update new connecting clients
Contents: Groundstation will store telemetry in JSON format. The log should limit the possibility of the corruption of the data due to a programming error and make the data easily accessible for newly connecting clients. Several choices were considered for the logging, including relational databases, JSON documents, and logging the raw telemetry.

A relational database would be unnecessary because the data does not have any relations that need to be tracked; each telemetry packet is an independent piece of information. Logging the telemetry as it comes in from the rocket would limit the possibility of corruption. A programming error in the JSON parsing routines could cause all of the data to be unusable. However, storing the data in JSON format would make it very easy for new clients that connect to the server in the middle of the rocket flight to be updated with all of the past data. Additionally, a JSON document would be easy to expand into a NoSQL database such as MongoDB [5]. For this reason, the telemetry will be stored in a JSON format.

JQuery
Entity
Author: Terrance lee
Type: Library
Purpose: Make it easier to use JavaScript on our website.
Contents: Our main concern with JQuery is that it may have limited functionality. JQuery has an extensive library, but if your website has a lot of customization in it, there is a chance of running into limited functionality with it. There are two ways to counter this issue. One is keep the website simple. Our main focus is to collect data from the rocket and display it. Keeping a simple website with that in mind should not be a problem. The second way if we need to customize more than it can handle is by using raw JavaScript for the parts that are causing the issue.

3.js
Entity
Author: Natasha Anisimova
Type: Library
Purpose: 3.js is a JavaScript 3D library used to create and display animated 3D computer graphics using WebGL.
Contents: Since the information about the rocket will be displayed by using a web browser through a local Wi-Fi network, making sure the information is displayed on time and correctly is crucial. 3.js allows for easy and rapid development of WebGL applications, which means we can make the most of the specialized graphics hardware on the users' PCs.

Rocket
Entity
Author: Natasha Anisimova
Type: Component
Purpose: Physical manifestation of the object being tracked.
Contents: The rocket will have sensors attached to it that will transmit information about its altitude, GPS coordinates, and tilt. While the rocket is on its journey, the GPS signal will cut out due to the COCOM limit. Because of the COCOM limit, that the longitude and latitude will have to be calculated based upon the other information that is received.

Web server
Entity
Author: Albert Morgan
Type: Part
Purpose: The web server will serve three primary functions:

- Serve web pages to the clients
- Receive telemetry from the serial port and convert it into JSON
- Make the JSON data available to the clients

Contents: The web server needs to be stable so there is no data loss during the flight. Additionally, the web server should be as lightweight as possible to minimize the resource consumption on the Raspberry Pi. Groundstation will use the Apache [6] web server. This web server was chosen over other options primarily for its stability. Apache uses a one-thread-per-connection model [7], so if one thread crashes, it will not affect the rest of the connections. NGINX [8] was also considered. On large-scale servers, it is much more lightweight because it uses a many-connections-per-thread model. This allows NGINX to stay responsive even when there are thousands of connections. However, the number of connections will be small and the reuse of threads could reduce stability. For example, the thread serving the connection could crash and disconnect everyone using the software. Lighttpd [9] was also considered. Although Lighttpd is much more lightweight than Apache, it is not nearly as mature. Because stability is a primary concern, Apache was selected.

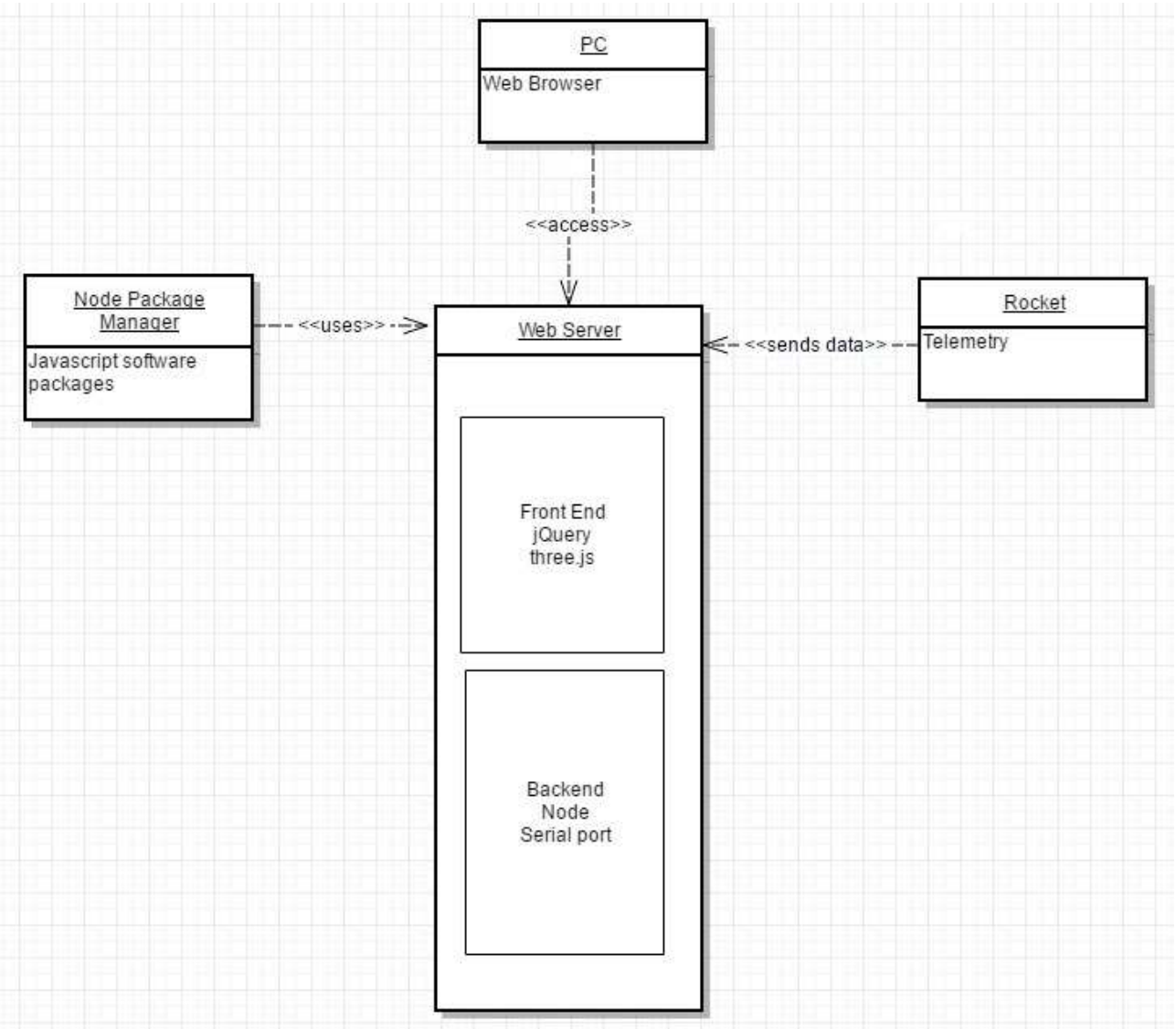

Fig. 1. A UML diagram of what the web server is and how it is connected to everything else.

Web browser
Entity
Author: Albert Morgan
Type: Process
Purpose: The web server
Contents: The client will use a web browser to connect to the Groundstation web server and access the content. The web browser may be any of:

- Chrome version 54 or higher
- Edge version 14 or higher
- Firefox version 49 or higher
- Safari version 10 or higher

These browsers were selected to be supported because they represent the most recent versions of the most popular web browsers in use today.

Web browser / Web server
Relationship
Author: Albert Morgan
Type: Connected
Contents: The web browser connects to the web server over a WiFi network via the HTTP protocol.

Frontend / Backend
Relationship
Author: Natasha Anisimova
Type: Connected
Contents: The Frontend will take what the Backend of the software gives it and use 3.js to create and display information to the user.

JQuery composition
Relationship
Author: Terrance Lee
Type: Composition
Contents: JQuery will be used to make our PC Controls on the website

3.js
Relationship
Author: Natasha Anisimova
Type: Part of
Contents: The JavaScript Library 3.js will be used with WebGL to create the frontend of Groundstation. Figure 2 goes through how 3.js is a part of the creation of the frontend

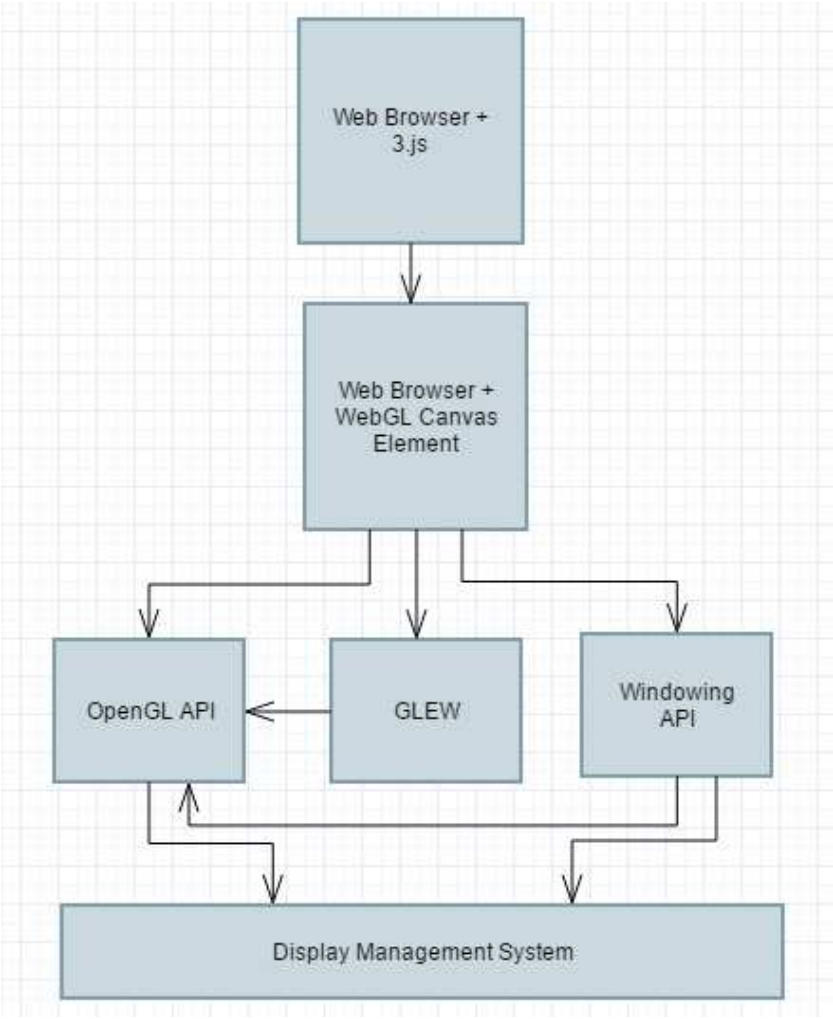

Fig. 2. A diagram of how 3.js is connected to WebGL and the frontend.

Node / Backend
Relationship
Author: Albert Morgan
Type: Part of
Contents: Node is used by the backend to do the majority of the work. Node is responsible for receiving, transforming, logging, and sending the telemetry to the user.

Serialport use
Relationship
Author: Albert Morgan
Type: Use
Contents: Node uses the serialport library to get data from the serial port.

Log / Backend
Relationship
Author: Albert Morgan
Type: Part of
Contents: The log is part of the backend, and will be written to by the backend and exist on the same device that runs the backend.

Backend / Rocket
Relationship
Author: Albert Morgan
Type: Connected
Contents: The backend will retrieve data from the rocket through the serial port. The exact protocol used will be determined at a later date by the OSU AIAA High-Altitude Rocketry avionics team.

NPM / Frontend
Relationship
Author: Albert Morgan
Type: Connected
Contents: NPM will be used to install all frontend software dependencies.

NPM / Backend
Relationship
Author: Albert Morgan
Type: Part of
Contents: NPM is part of the backend, and will be used to update and install all backend software dependencies.

III. Interaction View

PC
Entity
Author: Terrance Lee
Type: Component
Purpose: To be a tool to interact with the Raspberry Pi and the user.
Contents: With the interaction of the PC and the Raspberry Pi our concern is the loss of connection due to the loss of the Wi-Fi signal. The Raspberry Pi is just like any other Wi-Fi router and has the same issues that can drop signals. The major issue we have to be concerned about is overheating. We are going to be in New Mexico, out in the desert. In Black Rock, NM we may experience temperatures up to 101 degrees. We may be in a hotter area. To counter this, we will have the ground station in an open area so that it can get plenty of air flow. The second is that we will put in shade. We will have to bring an umbrella or a tent. This will allow the ground station to breath and stay cool. The umbrella or tent will bring the temperature down as well.

PC

Relationship
Author: Terrance Lee
Type: Component
Contents: With the interaction between the website and the user we still need the PC as a tool. Our concern in between these two is usability. With that concern we want to make sure that everyone can understand how to navigate the website so that they do not get loss. There are two things we want to focus on to counter this. The first thing is to use Krugs first law of usability, the web-page should be obvious and self-explanatory. When we make our website we need to make it so that the High-altitude Rocket team can get from point A to point B easily. That way if they need to navigate multiple pages they wont get lost. The second is the keep it simple principle (KIS). Doing this will help the Rocket team when they have to use anything on the page that they are on. If it is simple to use, then there is less of a chance of mistakes or misunderstandings.

Rocket
Entity
See *Structure View - Rocket*

Web Server
Entity
See *Structure View - Web Server*

Web Browser
Entity
See *Structure View - Web Browser*

Telemetry Protocol
Relationship
Author: Albert Morgan
Type: Signal
Contents: Once every second, the rocket will send a telemetry packet to the groundstation (the groundstation hardware, not the Groundstation software packagex). This packet will be received and transformed into a protocol that will be transmitted to the Groundstation software package via the serial port. The specification of this protocol will be determined by the avionics team at a later date.

JSON Protocol
Relationship
Author: Albert Morgan
Type: Signal
Contents: Data will be stored in the log file and transmitted to the user as a list of JSON objects in the following format:

```
{
    sensor: ...,
    value: ...,
    timestamp: ...
}
```

The JSON object has the following attributes:

- **Sensor**: This is the name of the sensor that the data is recorded from. This is an arbitrary string that uniquely identifies a specific sensor on the rocket. This string may be anything, and the user interface must understand which sensors provide which data. The specification of which strings correspond to which sensors will be done at a later date, after the protocol is defined.
- **Value**: This is the value that has been read off the sensor. The type of this data will depend on the specific sensor.
- **Timestamp**: This is the time and date that the data was received. This value is an integer representing the number of milliseconds that have elapsed since the epoch (January 1, 1970).

JSON objects will be transmitted to the client as a single object with a list of smaller objects inside. For example:

```
{
    {
```

```
        sensor: altimeter,
        value: 900,
        timestamp: 00584747458945
    },
    {
        sensor: tilt,
        value: 4,
        timestamp: 00584747458945
    }
}
```

IV. Interface View

3D View
Entity
Author: Natasha Anisimova
Function: Displays the current location of the rocket.
Interface: The user can interact with the view by using a mouse or track pad.
Contents: The view will be a 3D environment that will resize as the user resizes the window of the web browser. If the user clicks and drags in a particular direction the 3D environment will move in that direction. If the user just moves the mouse without clicking, the mouse will move freely on the screen. To make orientation easier in this environment, there will be a set of buttons, shown in the figure below, that will have preset views readily available for the user. The top-most button will show a bird's eye view. The left button will show a three-fourths view facing the left side of the launch site. The right button will do the same as the left button except facing the right side of the launch site. Finally, the bottom button in this set will give a view that is directly aligned with the ground, or XZ plane, looking out into the horizon. With the help of these buttons, the user should never feel like they are unable to get back to looking at the journey of the rocket. The scroll button will zoom in and out, with the point of origin being the launch location of the rocket. Figure 3 shows camera view controls that will be located in the top right corner of the browser.

Fig. 3. A diagram of the camera view controls.

3D Environment
Entity
Author: Natasha Anisimova
Function: Displays the current location of the rocket in the launch environment.
Interface: The user can interact with the view by using a mouse or track pad.
Contents: The 3D environment will consist of the topological map of the launch site (Spaceport America in New Mexico), with satellite images from NASA of the area mapped to it. If the user zooms past the point that the topological map covers, the ground will be displayed as a flat gray plane that goes on forever in every which way the ground would be. Since the rocket's goal is to reach 100,000 feet in height, labeling the different levels of the atmosphere that it enters would be helpful. Each level will be a transparent shade of blue, going from light to dark. Figure 4 shows the layers of the Earth's atmosphere.

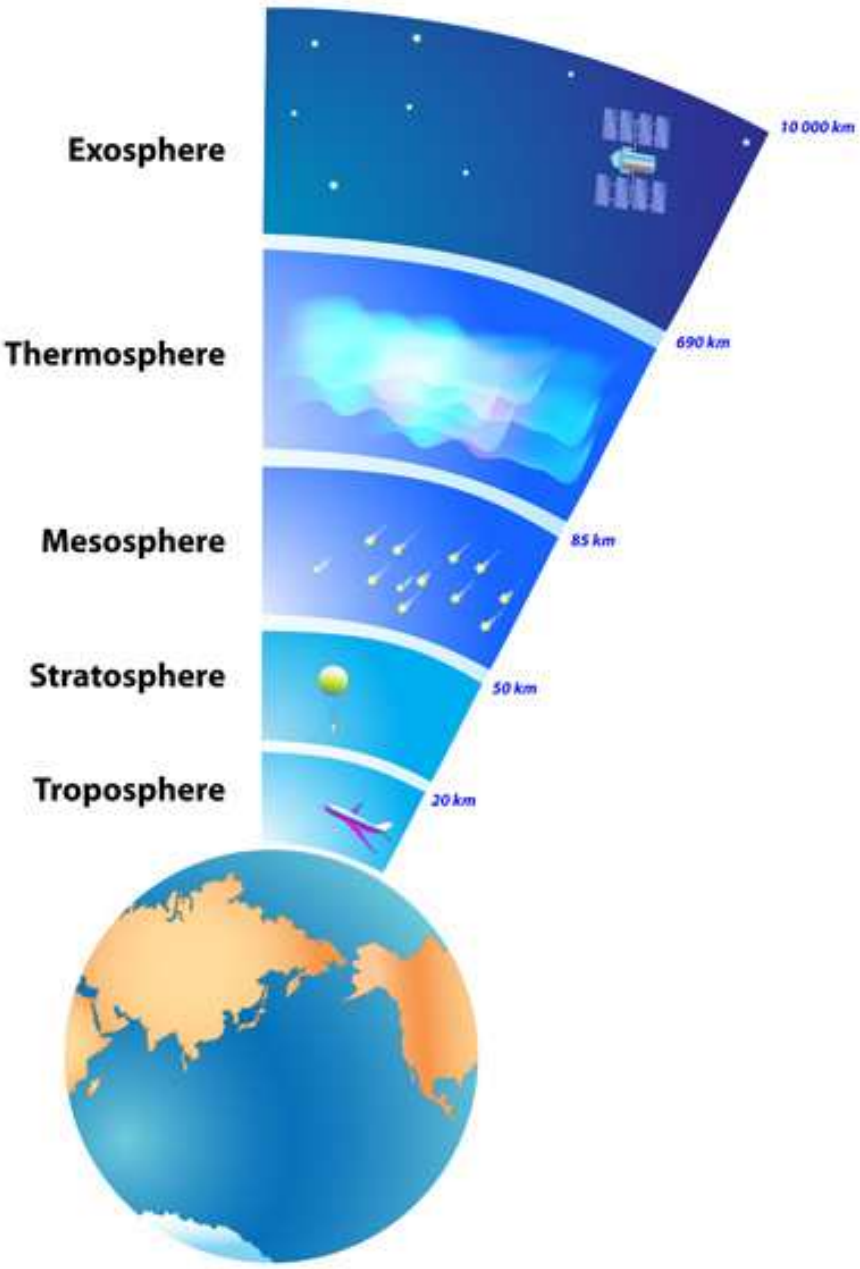

Fig. 4. A diagram of the Earth's atmosphere.

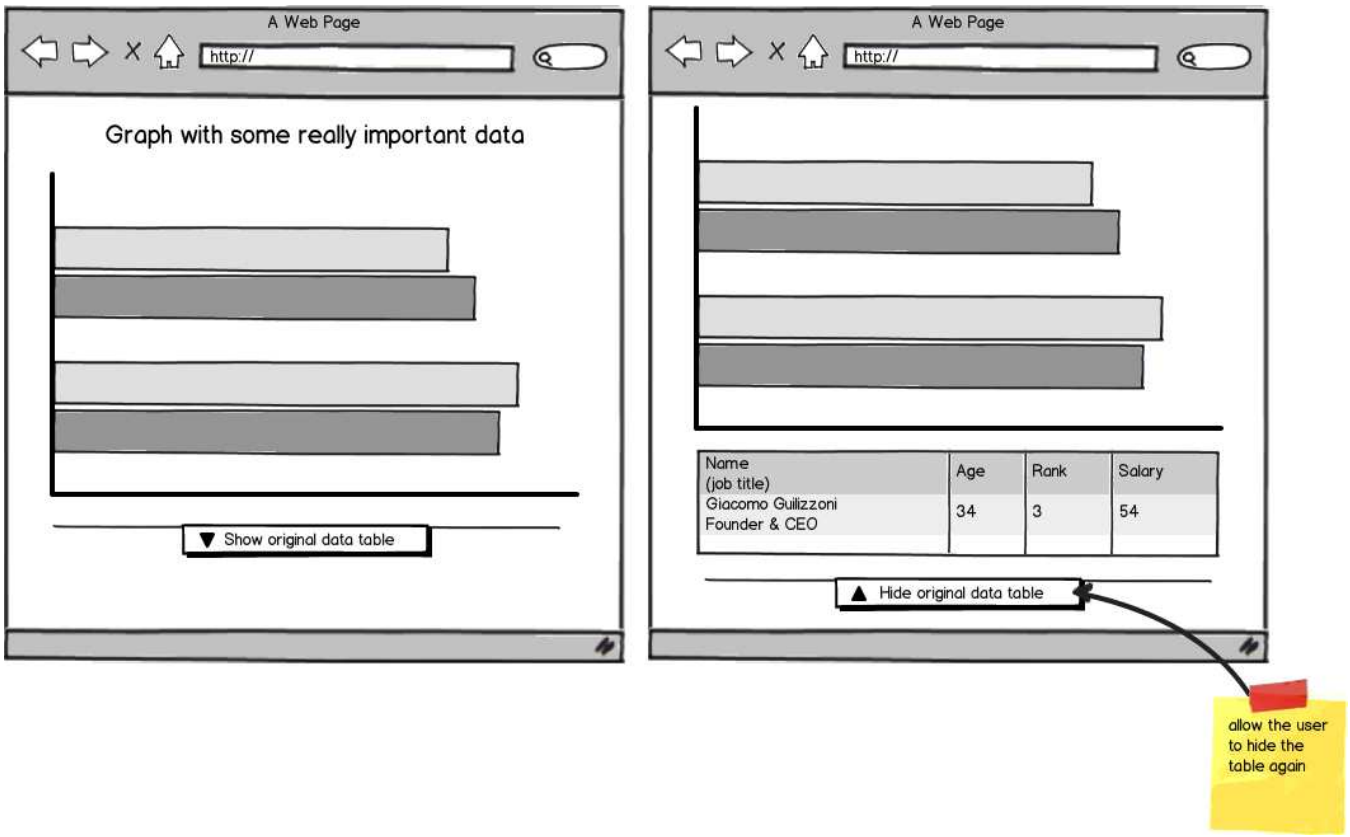

Fig. 5. A diagram of how to show or hide a table.

Numerical Data
Entity
Author: Natasha Anisimova
Function: Shows numerical data for the location of the rocket.
Interface: The user can view or hide the table.
Contents: Visual data can be hard to interpret without any numerical data to put it into perspective. Due to this, a table with the altitude, longitude, and latitude will be a part of the user interface. This table will be updated every time that a telemetry packet is received and given to the frontend of the web server. Figure 5 shows an example of how the table could be visible or hidden based off of what the user wanted.

JQuery
Entity
See *Structure View - JQuery*

JQuery interface
Relationship
Author: Terrance Lee
Type: interface
Contents: Our other concern with JQuery is the interface with the user. We want to build our dials, sliders or buttons using JQuery. The concern is not whether or not it will work, but if it is right for the sub-teams. We have at least five sub teams plus their members using them. JQuery has a vast selection of PC controls. If we select the wrong setup for the team it may make it difficult for the teams to use. To counter this, we have to think about usability. The first thing we should do is find data each team needs from the rocket. Next find out if they are going to need PC controls. If so, then find out which ones would suit them best through user stories.

V. Resource View

Math.Js
Entity
Author: Terrance Lee
Type: Library
Purpose: Node Math library
Contents: Our concern with Retrieval is loss of certain sensors. Do to CoCom limits we lose the GPS sensor until it is under 1200 mph and under 60,000 feet. This will happen after apogee. Also we lose accurate reading from the altimeter after 100,000 feet due to restrictions of the sensor itself. It cannot read accurately after that height; it will just read a constant pressure until the rocket gets below that 100,000 feet. To counter act these loses we have other sensors like accelerometer and gyroscope which we can use physics and math formulas to get a location of the rocket. With Math.js this allows us to use formulas and any other math equations to achieve this.

Node
Entity
See *Structure View - Node*

Node
Relationship
Author: Terrance Lee
Type: Part of
Contents: Node is will allow us to store to the server while we are collecting the data. The data will stay on the server until we power it down. This should allow us plenty of time to save it to our computers and hard drives. This allows us time for resets due to software issues as well.

VI. Revisions

Log Revised
Entity
Author: Albert Morgan
Type: Data store
Purpose: Store the data in non-volatile storage for later retrieval and update new connecting clients
Contents: Groundstation will store telemetry in CSV. The log should limit the possibility of the corruption of the data due to a programming error and make the data easily accessible for newly connecting clients. Several choices were considered for the logging, including relational databases, JSON documents, CSV, and logging the raw telemetry.

A relational database would be unnecessary because the data does not have any relations that need to be tracked; each telemetry packet is an independent piece of information. Logging the telemetry as it comes in from the rocket would limit the possibility of corruption. JSON format would make it easy to import the data into a running Node application, but may become corrupted if the program halts unexpectedly. Additionally, programming error in the JSON parsing routines could easily cause all of the data to be unusable. CSV format will be used for two reasons. First, there are no special ending characters, such as curly braces, that need to be inserted at the end of objects. The data will be easily readable even if the process halts unexpectedly and isn't able to finish formatting the log. Second, CSV files are easy to import into applications such as Microsoft Excel. Our teammates will be able to easily load up the data into an application of their choice for analysis after the launch.

Web Server Revised
Entity
Author: Albert Morgan
Type: Part
Purpose: The web server will serve three primary functions:

- Serve web pages to the clients
- Receive telemetry from the serial port and convert it into JSON
- Make the JSON data available to the clients

Contents: The web server needs to be stable so there is no data loss during the flight. Additionally, the web server should be as lightweight as possible to minimize the resource consumption on the Raspberry Pi. Groundstation will use the Node's [2]

built-in web server. This web server was chosen over other options primarily for its stability. Apache uses a one-thread-per-connection model [7], so if one thread crashes, it will not affect the rest of the connections. NGINX [8] was also considered. On large-scale servers, it is much more lightweight because it uses a many-connections-per-thread model. This allows NGINX to stay responsive even when there are thousands of connections. However, the number of connections will be small and the reuse of threads could reduce stability. For example, the thread serving the connection could crash and disconnect everyone using the software. Lighttpd [9] was also considered. Although Lighttpd is much more lightweight than Apache, it is not nearly as mature. Finally, we considered Node's built-in web server. Node's built-in web server would allow us integrate the rest of our code (which is written in Node) with the web server. Because we are already writing a lot of code based on the Node framework, writing the web server in the same language (JavaScript) will lower the time taken to develop. The extra time and ease of use will allow us to devote more time to testing and stability, effectively negating the benefit of using a more mature platform such as Apache. Finally, not using another piece of software for the web server will lower the time and effort it takes to install the software. This may be important if the software needs to be installed quickly on launch day.

For all of the reasons listed, the built-in Node web server was selected.

JSON Protocol Revised
Relationship
Author: Albert Morgan
Type: Signal
Contents: Data will be stored in the log file and transmitted to the user as a list of JSON objects in the following format:

```
{
    id: ...,
    latitude: ...,
    longitude: ...,
    accelerometer_x: ...,
    accelerometer_y: ...,
    accelerometer_z: ...,
    yaw: ...,
    pitch: ...,
    roll: ...,
    timestamp: ...
}
```

The JSON object has the following attributes:

- **id**: This is the id of the rocket component that the packet comes from. There are two stages to the rocket: the booster and the sustainer, and each of them will be sending packets. The id field will inform client which part of the rocket this packet comes from, '0' for the booster, and '1' for the sustainer.
- **timestamp**: This is the time and date that the data was received. This value is an integer representing the number of seconds that have elapsed since the epoch (January 1, 1970).
- **other**: All other fields represent individual sensor readings. The units of these fields will be metric, except for angles, which will be encoded as degrees.

Frontend
Entity
Author: Natasha Anisimova
Type: Component
Purpose: User Interface
Contents: The frontend of the web server will create and display the information that it was given by the backend. Given that the information has to do with location (longitude, latitude, and altitude), a 3D display is preferred. The frontend will consist of programs using 3.js and WebGL to create the visualization necessary to quickly understand the location of the rocket. There is a problem with the data being sent and having a 3D display. The Avonics team has a 3-axis gyroscope and the GPS will only be turning on at apogee. Gyroscopes tend to drift on their own so the direction of the rocket's flight will not be one-hundred percent accurate until the GPS. Having a simple 2D graph of the rocket's flight path to fall back on would be helpful in case the GPS does not turn on.

REVISION HISTORY

Name	Section Title	Date
Albert Morgan	Log Revised	2017-02-15
Albert Morgan	Web Server Revised	2017-02-15
Albert Morgan	JSON Protocol Revised	2017-02-15
Natasha Anisimova	User Interface Revised	2017-02-17

VII. Glossary

- **Accuracy:** The absence of errors in the telemetry GS receives, logs, and displays.
- **Binary data:** Telemetry that represents one of two states, for example, "stage 2 activated" and "stage 2 not activated."
- **COCOM limits:** The Coordinating Committee for Multilateral Export Controls (COCOM) limits apply hard limits to the speed and altitude of a GPS device. If a device exceeds these limits, it will stop providing GPS data. This is done to ensure that the device cannot be used as a missile.
- **Corruption:** The process by which data is altered or made unreadable.
- **Crash:** A software crash; the event in which a piece of software ceases operation unexpectedly.
- **Die:** A process dies when it ceases operation and is removed by the operating system. The difference between dying and crashing is that a crash may cause the program to become non-responsive, whereas dying causes the process to end.
- **GS:** Groundstation, the name of our software.
- **Graphical display:** Data that is displayed using a visualization.
- **Live:** Updated in real-time.
- **Non-volatile storage:** Storage that will not be erased when the system is powered down. For example, a hard drive or flash storage.
- **Page:** A web page that users of GS may connect to in order to view the telemetry.
- **Process:** A running program on a computer.
- **Raspberry Pi:** A small, inexpensive computing platform.
- **Real-time:** Each telemetry datum received from the rocket must be processed and displayed in under one second.
- **Redundant sensors:** Two or more sensors that provide the same type of data.
- **Reliability:** In the event of a software crash, the Groundstation software should automatically start and begin all normal functions in under five seconds.
- **Robustness:** In the event that GS receives data that is garbled or otherwise does not adhere to the protocol, it must continue to receive and display data and not break the real-time requirement.
- **Storage:** A device where data is logged.
- **Telemetry:** Data received from the rocket while the rocket is in flight.
- **Telemetry packet:** The rocket will send a telemetry update once per second. Each one of these updates is a "telemetry packet."
- **Visualization:** Information or data, transformed into an visual context.
- **X, Y, Z coordinates:** A three coordinate system that pinpoints a point in 3D space.
- **XZ plane:** A plane extending in the XZ directions. I.e., parallel to the ground.

INDEX

3.js, 3, 15

Accuracy, 17
Altitude, 5

Backend, 3, 9
Binary data, 17

Chrome, 7
COCOM limits, 17
Corruption, 17
Crash, 17
CSV, 14

Die, 17

Edge, 7

Firefox, 7
Frontend, 3, 15

Graphical display, 17
Groundstation, 17
GS, 17

Interaction view, 9
Interface view, 11

JavaScript, 3, 4, 15
JSON, 4, 5, 14

Latitude, 5
Live, 17
Log, 4, 9, 14
Longitude, 5

Node, 3, 9, 14
Non-volatile storage, 17
NPM, 9

Packet, 17
Page, 17
PHP, 3
Process, 17

Raspberry Pi, 3, 5, 14, 17
Real-time, 17
Redundant sensors, 17
Reliability, 17
Resource view, 14
Robustness, 17
Rocket, 5
Ruby, 3

Safari, 7
Storage, 17
Structure view, 3
Telemetry, 5, 14, 17
Telemetry packet, 17

User Interface, 4

Visualization, 3, 4, 15
Vizualation, 17

Web browser, 4, 7
Web Server, 14
Web server, 5, 7
WebGL, 3, 4, 15

X, Y, Z coordinates, 17
XZ plane, 17

REFERENCES

[1] Node package manager. [Online]. Available: https://www.npmjs.com
[2] Node.js. [Online]. Available: https://nodejs.org/en/
[3] Php. [Online]. Available: php.net
[4] Ruby. [Online]. Available: https://www.ruby-lang.org/en/
[5] Mongodb. [Online]. Available: https://mongodb.com
[6] Apache. [Online]. Available: https://www.apache.org
[7] O. Garrett. (2015) Nginx vs. apache: Our view of a decade-old question. [Online]. Available: https://www.nginx.com/blog/nginx-vs-apache-our-view/
[8] Nginx. [Online]. Available: https://nginx.com
[9] Lighttpd. [Online]. Available: https://www.lighttpd.net/

Nancy Squires	Date
Natasha Anisimova	Date
Terrance Lee	Date
Albert Morgan	Date