WASHINGTON STATE UNIVERSITY
VANCOUVER

CS 320 Course Project Final Report

for

Calculation Unitility

Prepared by

Group Name: **"Segmentation Fault: 11"**

Daniel Rehman	**11458334**	**daniel.rehman@wsu.edu**
Jake Palmer	**11634568**	**jake.palmer@wsu.edu**
Andrew Oakes	**11630284**	**andrew.oakes@wsu.edu**

Date: **12/13/18**

Contents

1 Introduction

The Calculation Utility is a web app that allows the user to perform basic arithmetic, define basic and complex units, and perform unit conversions. By storing the units in a database it allows users to view and edit the units they've defined.

1.1 Project Overview

The project has two main features. The first is a unit converter, which converts units of the same type (e.g. length or mass) from one 'unit system' to another. Users define base units by entering information about the unit, the unit type, and the unit system. Once the user has multiple units of the same type, they can define a conversion factor between the two units. For example, the user could define 'meter' as a length unit in the metric system, and 'foot' as a length unit in the imperial system. They can then define a conversion between the two, in this case 1 meter = 3.28084 feet. When the user enters '1m' and switches the system to imperial, the app will display '3.28084ft'. Users can also define complex units in terms of existing units. The definition is just a string that the program parses. For example, Newtons would be defined as 'kg*m*s^-2'. Assuming all required base units and conversions are defined in each system, the user can then convert Newtons to the Imperial equivalent.

The second feature is a calculator. The UI follows the format of a basic calculator, and allows all standard arithmetic operations. In its current stage, the UI also presents a variety of more complex functions, such as sine, cosine, and tangent, but these are not functional. The calculator is integrated with the unit converter, and can perform calculations with units. For example, entering '1m + 1ft' would produce '1.3048m'. This also works with complex units. The calculator also includes buttons for adding new units, systems, and conversions, and for displaying all defined units.

1.2 Definitions, Acronyms and Abbreviations

No definitions, acronyms, or abbreviations are used in this report.

1.3 References and Acknowledgments

This document references the SRS and SDD previously submitted as part of this project.

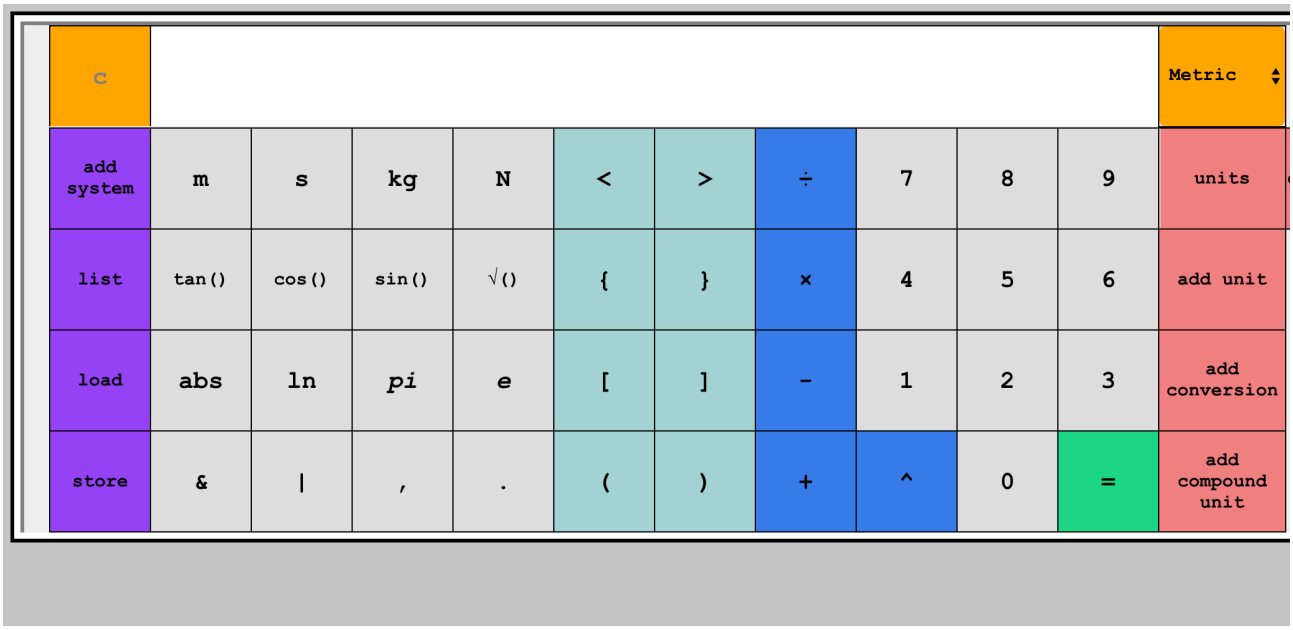
2 Design

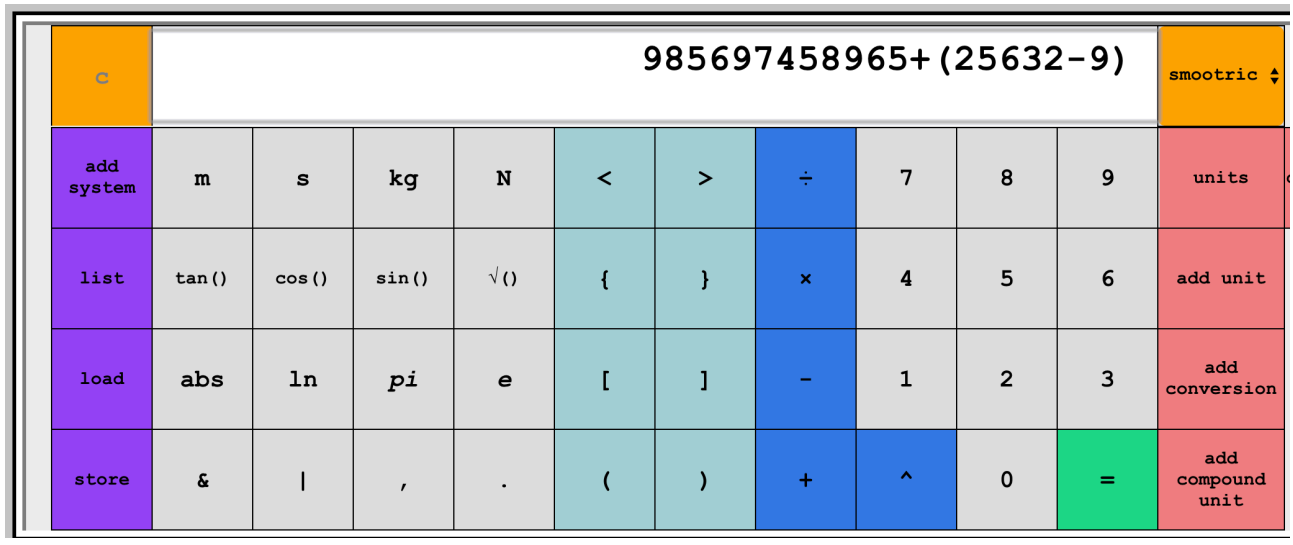
2.1 System Modeling

Our implementation strictly follows the design document (milestone 2), except regarding the functions and variables, which we have not yet implemented.

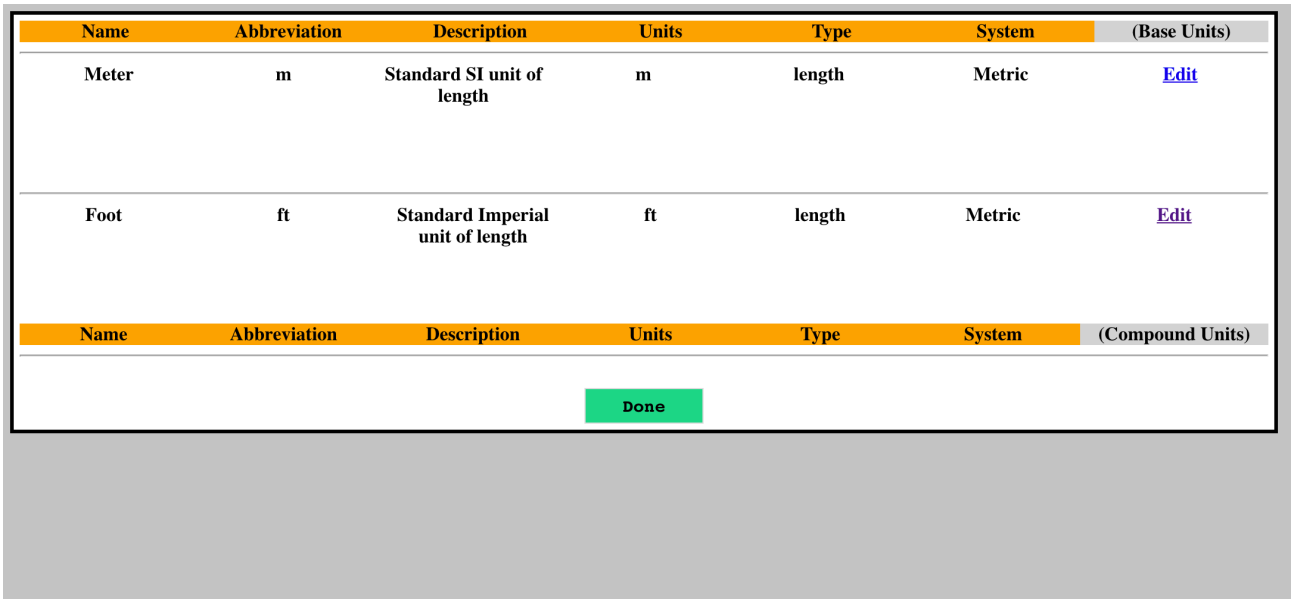
2.2 Interface Design

The Calculator

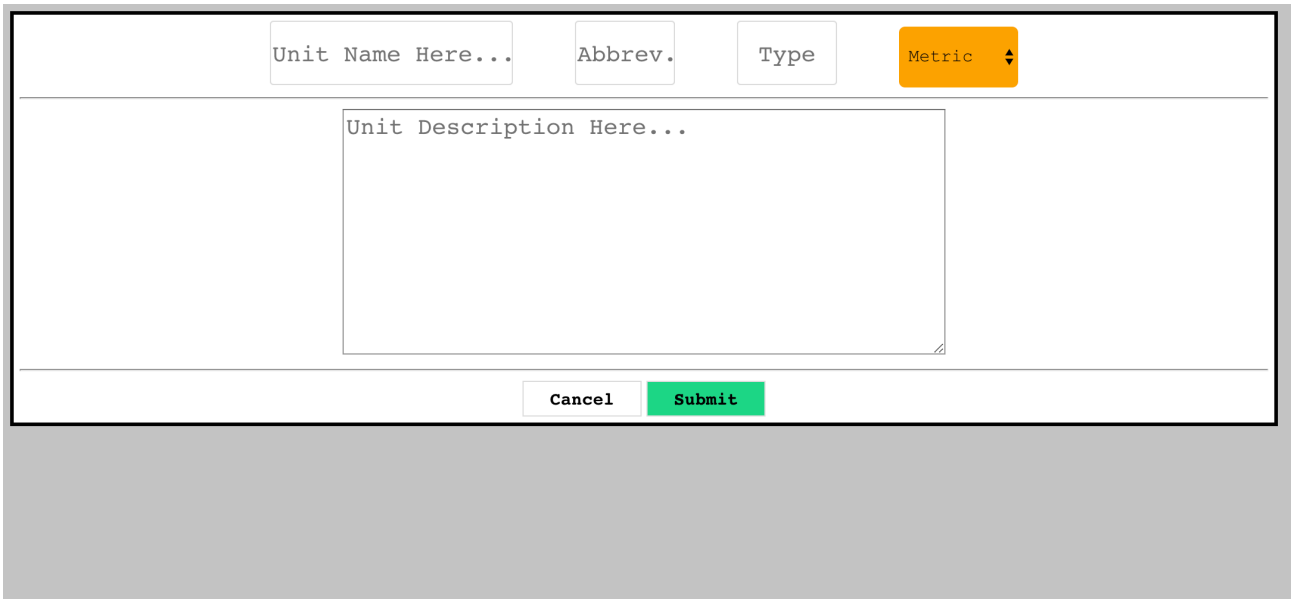




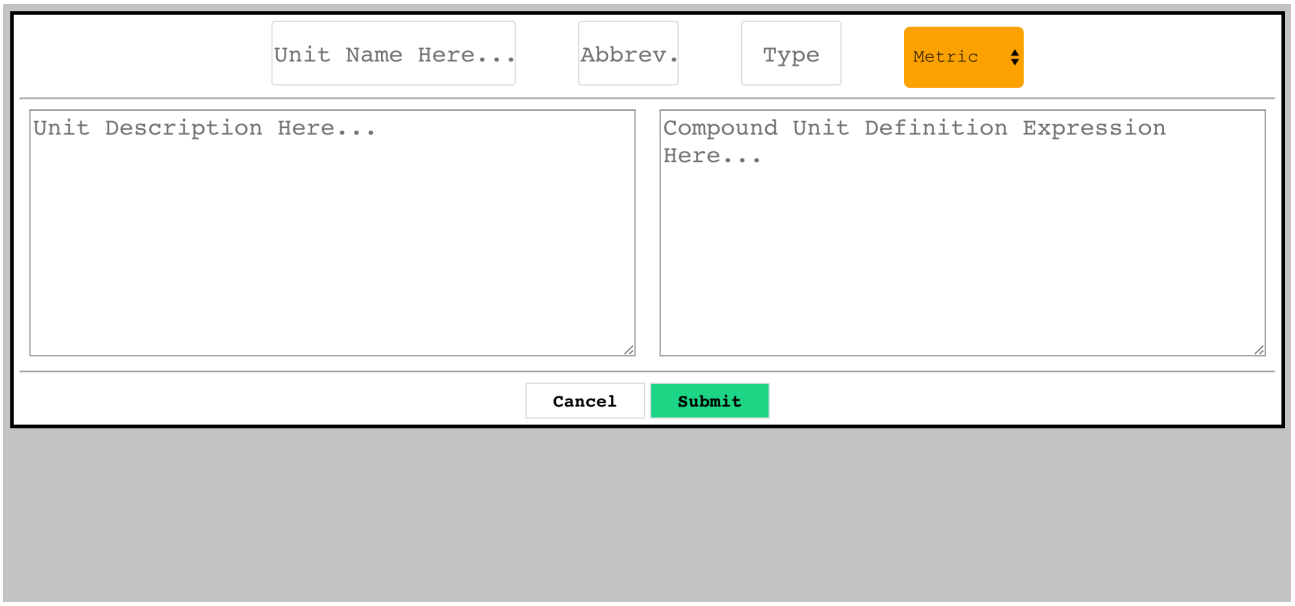
Listing Units



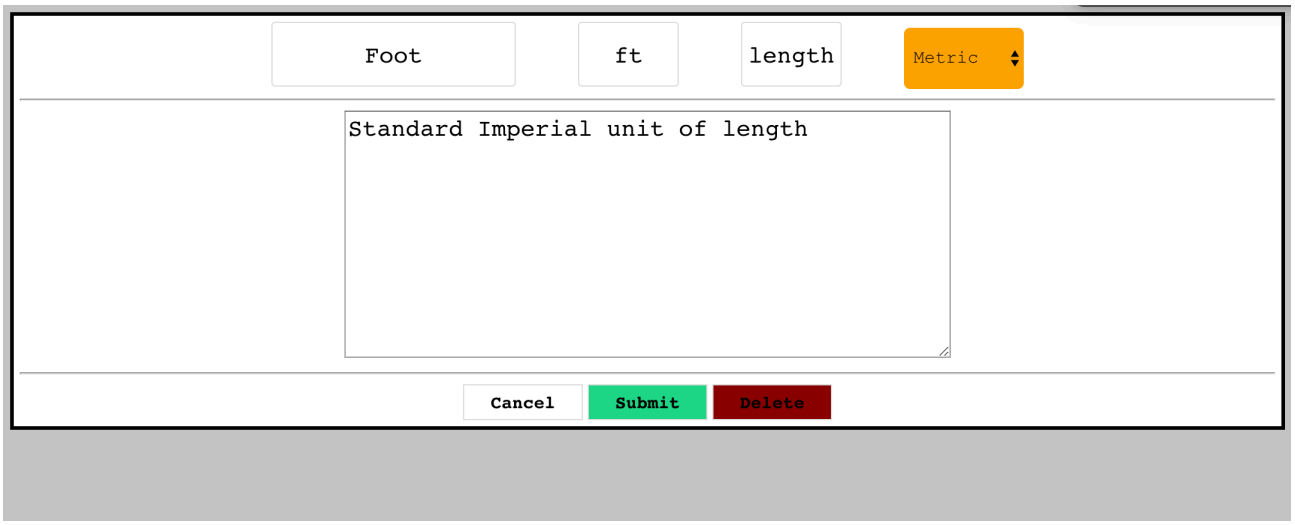
Adding New Base Units



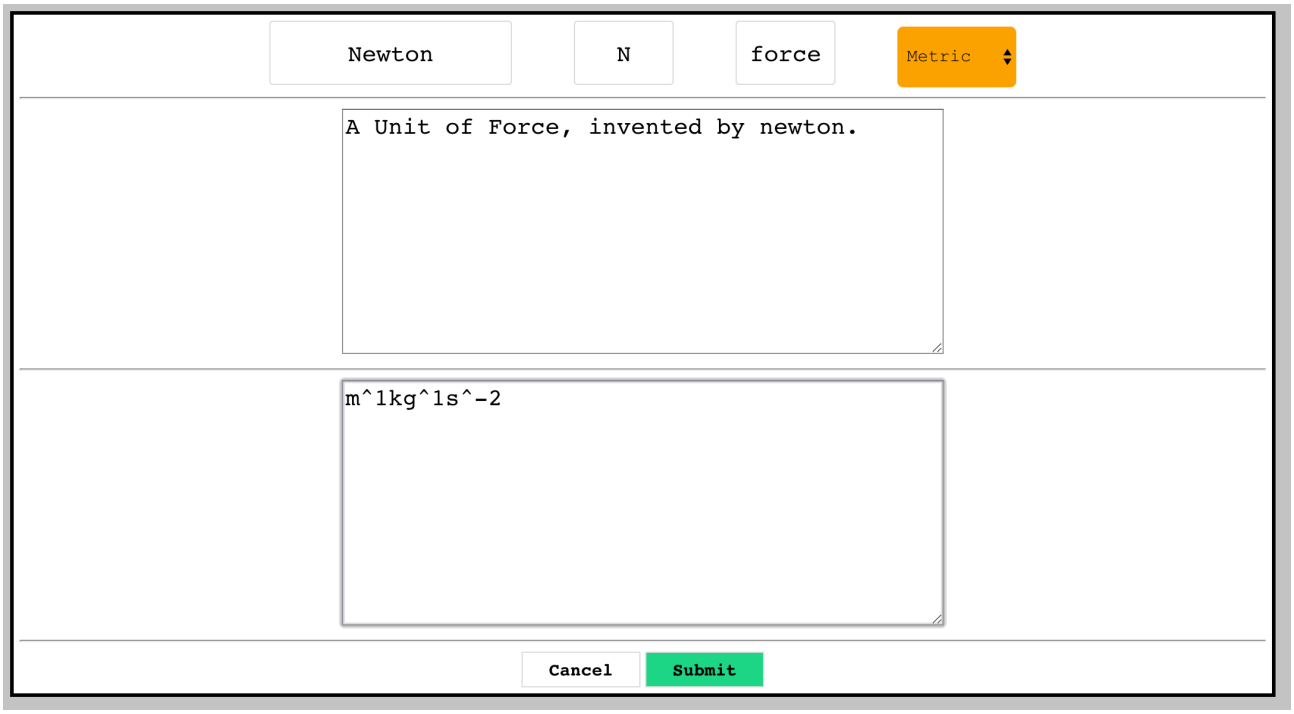
Adding Compoud Units



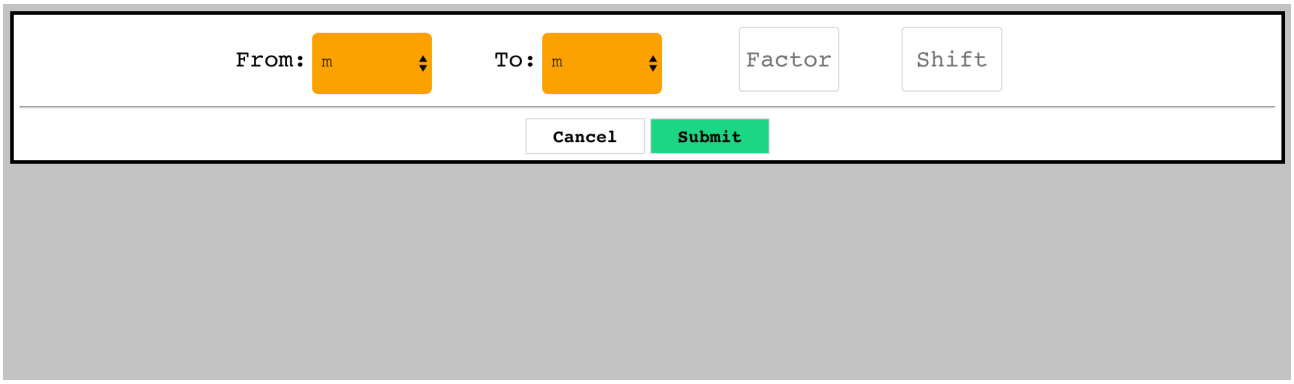
Editing Existing Base Units



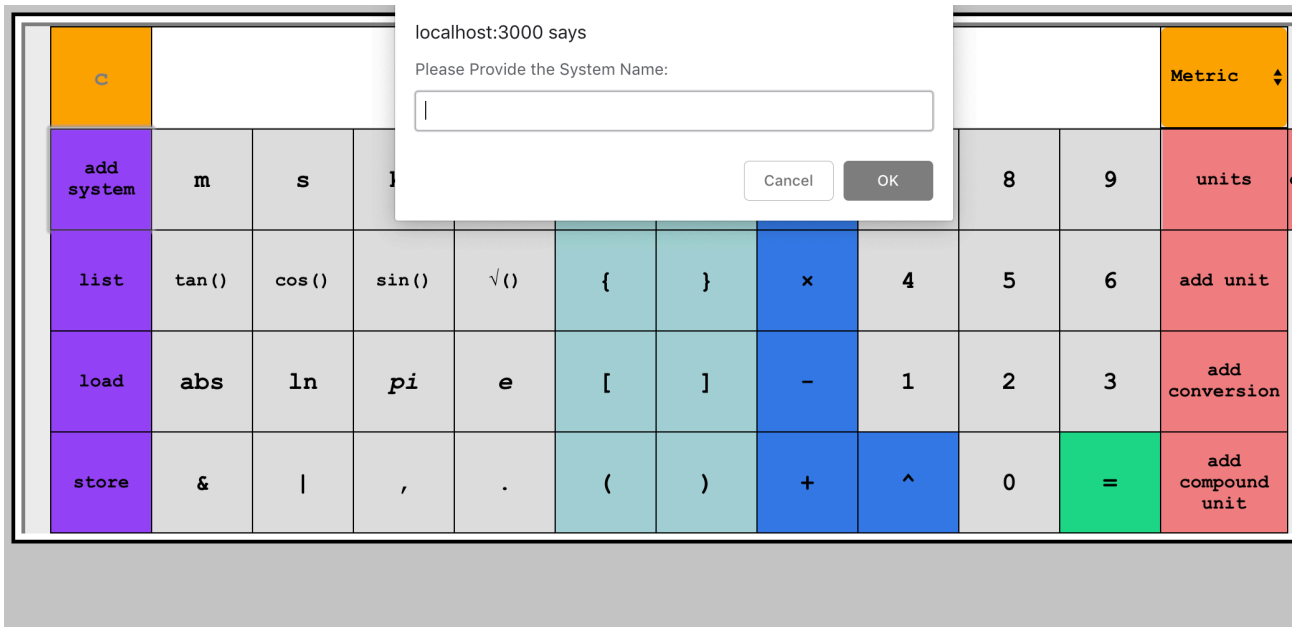
Editing Existing Compound Units



Adding Conversion



Adding New System



3 Implementation

3.1 Development Environment

Initial mockups of the UI components were developed with pure HTML and CSS, and the unit conversion algorithms were written in Javascript. To develop this into a functional application we used Meteor, a web app framework. Meteor uses Javascript, and integrates with MongoDB and Blaze for database and rendering support. Development took place in JetBrain's IntelliJ IDEA IDE.

3.2 Task Distribution

We didn't outline and follow strict responsibilities during development, so many aspects of the project likely have contributions from all members. In general the work was divided as follows

Jake Palmer: Created the Meteor project, and integrated the UI mockup with Meteor, Blaze, and FlowRouter. Created databases with schemas for storing units. Implemented functionality to add and edit units in the database.

Andrew Oakes: Created the complex expression parser/solver. Translated the database information into javascript objects for use in parser/solver and unit conversion algorithms. Made an abstract compound conversion algorithm so unnamed unit combinations could be converted.

Daniel Rehman: Created the the UI design and app layout, the CSS for the UI, and helped connect the UI to the functionality. Also helped develop the algorithms and datastructures for the functionality.

3.3 Challenges

One challenge we faced was designing an easy to use user interface. We overcame this challenge by discussing ideas and solutions at each group meeting, and testing out different approaches.

Another challenge was integrating the unit conversion algorithms we had developed with Meteor and MongoDB. Our current solution is messy, and we are still working on completely solving this issue. The most important step in overcoming this challenge is simply learning more about how to use Meteor and MongoDB.

The development of the base unit conversion algorithm was also very challenging. Our intent was to make it figure out transitions by following paths through a "transition graph" of sorts. This turned out to be pretty tricky, but after a few meetings discussing the best approach, we managed to construct a working algorithm that could figure out transitions that were not explicitly defined by the user. However, the edge case of unit conversions with shifts still caused problems until a while later(i.e. Fahrenheit to Celsius).

4 Testing

4.1 Testing Plan

There are three main parts of the project which require the majority of testing: The database, the user interface, and the conversion utilities. The conversion utilities, and the database code should be tested thoroughly while they are being developed (TDD or something similar), because each component tends to rely upon previous components to function. If something is found to not work for some edge case mid development, and it isn't due to the current code being developed, it could be a pain to debug. However, the interface can be tested and debugged fairly easily at any point (or so I think).

Each of these larger components break into many small components. The database code allows the adding of base units, compound units, conversions between units, and systems. The database code also allows for editing of units that already exist. All of these should be tested. The conversion utilities include code for converting between basic units defined by the user, converting between compound units, converting from one system of units to another system of units, and parsing and solving mathematical expressions which contain unit abbreviations and powers. The user interface should be tested for possible user inputs on each page, making sure the interface responds in the expected way.

4.2 Tests for Functional Requirements

Completed Tests:

- Database:
    - adding units to the database -> SUCCESS
    - adding conversions to the database -> SUCCESS
    - adding systems to the database -> SUCCESS
    - editing existing units -> SUCCESS
    - deleting units from the database -> FAILED

- UI:
    - Submitting new base units -> SUCCESS
    - compound units -> SUCCESS
    - conversions with invalid,duplicate, or valid data and verifying that the page responds correctly -> SUCCESS
    - editing a unit through the interface -> SUCCESS
    - deleting a unit through the interface -> SUCCESS
    - Not displaying deleted database entries -> FAILED

- Conversion Utilities:
    - Converting to a base unit with a defined path -> SUCCESS
    - Converting to a base unit with an undefined but reachable path -> SUCCESS
    - Converting between compound units with a defined path -> SUCCESS
    - Converting between compound units without a defined path, but each base unit has a defined or reachable conversion -> SUCCESS
    - Converting from a compound to an abstract compound when there is no parallel in the desired system -> SUCCESS
    - Translating database information into local objects -> SUCCESS

4.3 Tests for Non-functional Requirements

At the moment our Non-functional requirements are a mess. For the most part we didn't have time to worry about performance since our primary concern was getting it functional before the deadline. The one real Non-functional requirement test we used was ES-lint inspections, but we were not super consistent about editors, so that test would completely fail currently.

4.4 Hardware and Software Requirements

The hardware requirements for performing these set of tests, were simply to have a running computer or laptop, with a screen and keyboard and mouse, and with sufficient power.

However, the software requirements, were for the computer to have access to a unix terminal, have git, a sufficiently modern web browser with javascript, have meteor installed, have a series of required node module packages installed, as well as a text editor to make changes to the source code.

5 Analysis

Milestone 1: (Project's Conception ... SRS Document)

Approx. Number of Hours:
- Andrew: ~3h
- Jake: ~10h
- Daniel: ~16h

Milestone 2: (SRS Document ... SDD Document)

Approx. Number of Hours:
- Andrew: ~10h
- Jake: ~5h
- Daniel: ~10h

Milestone 3: (SDD document ... Final Report Document)

Approx. Number of Hours:
- Andrew: ~40h
- Jake: ~30h
- Daniel: ~20h

Judging from the amount of time spent on each milestone, the milestone that took the most effort would probably be the 3rd milestone, from the time of submitting the SDD document, to the time of writing the final report for the project. This milestone was the most packed with functionality implementations, UI CSS tweaks, and meteor debugging sessions.

6 Conclusion

Through the design and development process of this project, we have came across a plethora of different types of challenges, ranging from algorithmic challenges, to UI design challenges, to even third-party compatibility code architecture challenges (namely, integrating meteor into our project). In all of these challenges, we worked as a team to figure out the root of the problem, and then collaboratively developed a solution that pulled from each of our experiences as developers.

One of the most important things we learned while completing this project, despite learning an immense amount about the meteor API and web application design in general, was the ability to accurately convey a complex implementation idea or solution to another developer. This skill alone, maybe, allowed us to retain the cohesiveness in our application development, because we all knew exactly where the application was, and where it was going.

In addition, we also learned about how development processes, such as Agile development, are actually done in practice. This gave us real experience of what development may be like in the work place, thus preparing us for any other complex application development we may do in the future.

Appendix A - Group Log

We formed our group in September, and begun discussing ideas for the project. In early October we decided to create a customizable unit converter, and began discussing how the system would work and look. Through October and November we met up about once a week and worked on the unit conversion system. Meetings became less frequent in November, but after Thanksgiving break we refocused and worked together several times to eventually finish the application to the specifications.

Group meetings were always productive, and we communicated well with one another.