Cal Poly Pier Remote Monitoring

A Senior Project by Anthony Annuzzi
Advised by Dr. Bridget Benson
Sponsored by Thom Maughan

Computer Engineering Department
California Polytechnic State University, San Luis Obispo
June 2016

Table of Contents

List of Figures 3
List of Tables 4
Acknowledgements 5
Abstract 6
Introduction 7
Project Requirements 8
  Data Collection 8
  Web Server 8
  Power Consumption 8
Project Design 9
  Hardware 9
    Circuit Design 10
  Software 11
    Microcontroller Overview 11
      Cell Modem 12
      Current Monitor 12
      RS232 Interface 13
    Web Server Overview 14
      API Design 14
      Front End Design 15
Testing 19
  Hardware 19
  Web Server 19
Conclusion and Future Work 21
References 22
Appendix 23
  Analysis of Senior Project Design 23
  Bill of Materials 25
  Eagle Schematic 26
  Source Code 27
    Microcontroller Software – Energia 27
    Web Server API (Node) – Server.js 30
    Controller – App.js 38
    Index.html 48
    Livedata.html 50

List of Figures

Figure 1: System Block Diagram 10
Figure 2: Remote Monitoring System with the Battery and Charger 11
Figure 3: Example Request for Pushing Data 15
Figure 4: Example Request for Fetching Data 15
Figure 5: Data Mapping Dialog 16
Figure 6: Dialog for Managing Charts 17
Figure 7: Charting Dashboard 18

List of Tables

Table 1: Calculations for Calibration 13
Table 2: Test Results for Hardware 19
Table 3: Test Results for Web Server 20
Table 4: Bill of Materials 25

Acknowledgements

This project was made possible through the help of many people. I would like to thank Thom Maughan of the Monterey Bay Aquarium Research Institute for sponsoring this project. I would also like to thank Dr. Bridget Benson from the Computer Engineering Department for advising me throughout this project. I would also like to thank Tom Moylan and Jason Felton for letting me test my project at the pier. Last but not least, thank you all of my professors I've had during my time here at Cal Poly, I've learned so much from each and every one of you.

Abstract

Marine scientists have deployed experiments at the Cal Poly Pier to study various parameters. A major problem with having these remote experiments is that scientists do not know if the experiment is functioning properly or if sensors stop working. The only way to check is to physically visit the deployment site. The goal of this project is to design and implement a remote monitoring system for equipment that is located at the Cal Poly Pier. This system will focus on monitoring the vital components of the system, specifically the voltage and current being supplied by the solar panel, the voltage of the battery, and the current that is being drawn by the experiments. This data will be sent to a web server and will be accessible through a website that will graph the data and send notifications should the data be outside its typical values.

Introduction

Thom Maughan, an Engineer from the Monterey Bay Aquarium Research Institute (MBARI), has solar powered marine experiments set up at the Cal Poly Pier near Avila Beach. Currently, the only way to check the status of the experiments is to physically visit the pier and check them. The remote monitoring system will collect vital data that ensures the experiments are operating. The scope of this project is to collect voltage and current to ensure that the battery has enough charge to keep the system running. If the system loses power, the server can send notifications that data hasn't been received so that someone can check on the status. A future goal is to read sensor data from the experiments and push that data to the server as well.

Having the ability to remotely monitor experiments ensures that the experiments are completed as efficiently as possible. Without a system like this, experiments could be deployed without any data being collected if there is a problem with the experiment setup. This system will allow scientists to ensure that data is collected so that they can continue to enhance their knowledge.

Project Requirements

Data Collection

For this project, the preliminary requirements are to measure the voltage and current of both the solar panel and the battery. It must be able to handle a voltage range between ±26V and current up to 8A. The voltage shall be accurate to within 50mV of the actual value. The current measurement shall be accurate to 10mA. This is because at night, when the battery is not charging, the only load will be the experiments, which will draw much less current than when the battery is charging. Having a large range of potential values may cause this to be a difficult requirement to meet. Once the data is read from the sensors, it must be sent to a remote computer using a cell modem every 15 minutes.

Web Server

Once the server receives the data, it must be kept in persistent storage. The server will analyze the data to ensure it is within tolerance and send alerts if it determines something is wrong. The data is to be graphed on a web page so that it can be monitored. The graphs shall be user customizable for the date range and fields that are to be graphed. This website must also function on mobile browsers so that data can be monitored from any location.

Power Consumption

This device is to be powered from the same battery as the experiments. Because of this, power consumption must be kept to a minimum. The goal for the system is to consume less than 2.5W of power on average. This includes any power dissipated by any measuring devices that are included in the system.

In addition to these requirements, there are secondary requirements to read sensor data from RS232 devices. This data shall be sent to the server in the same way as the voltages and currents.

In order to be deployed at the pier, the system must be waterproofed. It is to be housed inside a plastic toolbox that is already at the pier and currently houses the battery and solar charger.

Project Design

Hardware

When choosing hardware for this project, a few criteria were used to determine what would work best for this system. The main focus was to optimize accuracy and minimize power consumption.

I chose a high-side current monitor, the INA219 from Texas Instruments. This sensor works by measuring the voltage drop with a precision op-amp across a shunt resistor. In order to determine the resistance value for the shunt resistor, I had to factor in accuracy of the system and the power losses through the resistor. Since this resistor is a passive component that is always wired into the system, the losses through it will be constant. This was a major factor for determining what resistance to use. The short-circuit current of the solar panel is rated at 6 Amps so I used that for all of my calculations. I also tried to optimized the full scale voltage of the the op-amp, which is 80mV. These two factors led me to choose a shunt resistance of 0.01Ω. The constant power dissipation through this resistor is 0.36W, only $^1/_9$ of the required average power dissipation.

The additional goal for the project is to read data from RS232 devices that are attached to the experiment being run. I did not have access to any of these devices to prototype with so I designed the interface for the microcontroller 3.3V logic to the ±12V logic of RS232. For this, I chose a MAX2323 to handle this voltage differential. This device is a 2-channel line-driver, line-receiver, and charge-pump circuit in order to boost the voltage from 3.3V to 12V and handle the voltage drop from 12V to 3.3V. On the PCB I designed, I put the RX and TX lines to header pins so that they can easily be added in the future.

In order to send the data to the web server, I used a FONA800 cell modem from Adafruit. It is based off the SIMCom 800 and interfaces through UART. In order to communicate with the microcontroller, it uses AT commands.

When choosing a microcontroller, I focused primarily on power consumption and interfacing ports. The Texas Instruments MSP432 Launchpad was chosen because it offered 4 UART and an I2C port. This allowed me to interface the INA219, MAX3232, and FONA800 to the Launchpad.

Circuit Design

For the circuit design, I first focused on how the entire system would be powered. The deep-cycle battery at the pier ranges between 12V and 15V, the microcontroller and other circuits operate at 3.3V, and the cell modem operates at 3.7V. In order to power the cell modem, the manufacturer suggests it use a 3.7V lithium polymer battery in order to handle current spikes that occur during transmission of data. The battery can be charged either through USB or 5V. From here I chose a switching regulator that would drop the voltage from the battery level to 5V. This would provide the power for the cell modem to charge the battery. In addition to the switching regulator, I chose a linear regulator that would drop the 5V to 3.3V for the other components.

The input for the solar panel is only so that the shunt resistor can be added in series so that the current and voltage can be measured. The battery input is also measured but the low side of the input is used as the power for the monitoring system. The cell modem is also interfaced so that the microcontroller can check the power status and toggle it as needed to save power when not sending data. The block diagram for the system is shown below in Figure 1.

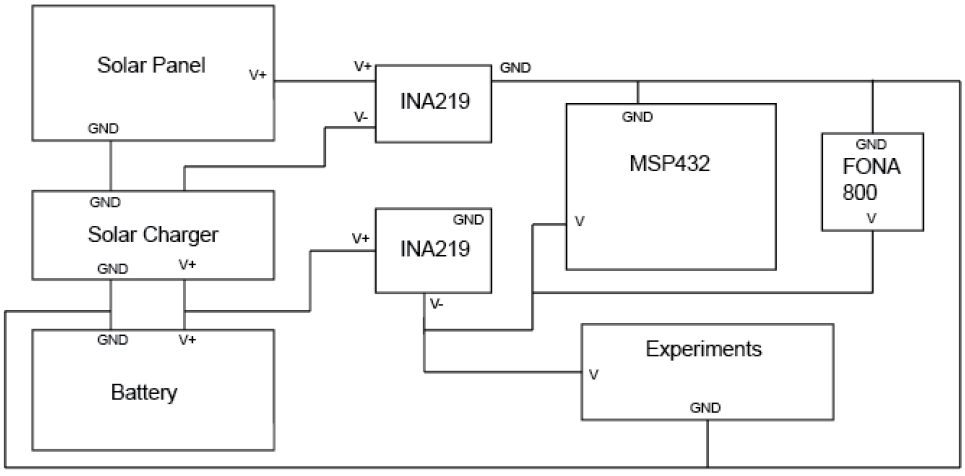


Figure 1: System Block Diagram

In order to interface these components, I designed a PCB that would interface with the header pins on the microcontroller. I needed to optimize the size of the PCB and match the exact pinouts of the microcontroller and cell modem. These requirements took up about half the size of the entire PCB. The other half of the PCB contained the two power management ICs, input terminals, RS232 inputs, and power regulators. Once this was built, header pins were used to connect it to both the microcontroller and the cell modem. The completed device, solar charger, and battery are shown below in Figure 2.

Figure 2: Remote Monitoring System with the Battery and Charger

Software

Microcontroller Overview

Having prior experience with the MSP430, I expected no problems interfacing UART and I2C. With the MSP432 being a new platform, there were some changes that I could not figure out. Instead of spending more time using Code Composer Studio, I chose to use a program called Energia. This allows you to program the microcontroller in a way that is very similar to Arduino. It contains many libraries that allow for interfacing with both UART and I2C with just a few lines of code.

Cell Modem

The Adafruit FONA800 communicates with the microcontroller via UART. In order to first test the device, I used a USB to TTL Serial Cable so that I could send commands and view the responses in a serial console. This allowed me to test the functionality and allowed me to find a sequence of commands that need to be sent in order to send a web request to the server. I then programmed the microcontroller to perform those same steps to ping the server. By default, the UART controller will echo any input. This is extremely helpful when using the serial console, but not when trying to parse responses on the microcontroller. The command ATE0 allowed me to disable the echo so responses could be more easily parsed. I also needed to find a way to read the cellular network time. To get this to work, I first needed to enable the setting so that when the cell modem registered to the cellular network, it would sync the network time and allow me to access it.

Current Monitor

The INA219 communicates with the microcontroller via I2C. When first prototyping I purchased a breakout board from Adafruit. I then downloaded a library[1] for it which allowed me to test it with a small circuit. Once I verified that worked properly, I determined that this would work for the final circuit design. I modified the library I downloaded so that I could properly calibrate the sensor for the range of values expected for the load. In order to calibrate the sensor, I needed to define the voltage range for the bus. The sensor had two values for this range, either 16V or 32V. The solar panel has a maximum open circuit voltage of 25V so I needed to use the upper range of 32V. I set the maximum shunt voltage to the lowest range of 80mV and calculated the maximum current without overflow to be 8A. This provided a good range considering the solar panel should produce a current around 5-6A. Continuing the calculations, I chose a value for the current LSB (250µA) and verified it would work with the other values. This gave me a calibration value of 16834 which is written to the calibration register of the INA219. Each of the calibration steps are shown in Table 1.

[1] Adafruit INA219 Library: https://github.com/adafruit/Adafruit_INA219

BUS_MAX	32
SHUNT_MAX	0.08
R_SHUNT	0.01
MAX_POSS_I	8
MAX_EX_I	6
MIN_LSB	1.83E-04
MAX_LSB	1.46E-03
CUR_LSB	2.50E-04
CAL	16384.00
PW_LSB	0.0050
MAX_CUR	8.19E+00
MAX_SHUNT	0.082
MAX_PWR	262.14

Table 1: Calculations for Calibration

RS232 Interface

Being the stretch goal of the project, I didn't spend too much time trying to implement this feature. The MSP432 has 4 UART ports, but only two of them are accessible from Energia. Given more time, I would've worked more to get everything working in Code Composer Studio. Since each RS232 device is different, and I never had access to one, there are 2 RS232 headers on the PCB, along with the IC needed to convert the logic levels to the microcontroller. There are also header pins that jumpers can be connected to when this is implemented in the future.

Web Server Overview

When designing the web server, I decided to try out a software stack that I have not used before but had some experience looking through code. I chose to develop the web server using Node.js, Express, Angular, and a Mongo database. Node.js is a Javascript based server that became popular a few years ago. Express and Angular are used together to render the pages and handle data binding. In addition, a CSS library, Bootstrap, is used to make the user interface more polished. MongoDB is a NoSQL database, so there is not a fixed schema like many other popular databases. This is perfect for the remote monitoring equipment. When new measurements are added in the future, nothing needs to be modified to accommodate the additional data.

I then focused on finding a good way to graph each of the data points over time. There are many Javascript charting libraries but I had a very hard time finding one that met all the requirements I was looking for. The most difficult requirement to find support for was an inconsistent time-series graph. With the remote monitoring system, there is no guarantee that data will be sent every 15 minutes so this is the major requirement. After lots of research and trial and error, I determined that Highcharts provided the best set of features for this situation. The data format for this library uses an array of points in a series with the format [timestamp, value]. In order to fetch the data points for the graph, I designed an API that will fetch data for any respective column in the database. This allows the end user to get the properly formatted data for any data that is saved.

In addition to this, the incoming data is monitored to ensure that the experiments are functioning as expected. The range of expected values is user configurable so that any type of data can be monitored. In the case that a data point is out of range, an email will be sent to the configured address. Since the remote monitoring system is powered from the battery, if that runs out of power, nothing will be sent to the server. In order to detect this failure, a job periodically runs to ensure that data has recently been sent to the server.

API Design

There are two main endpoints used for pushing and fetching data. The endpoint for pushing data is /data. When designing the API for reporting measurements, I chose to use a GET request to

make it easier for the microcontroller to send the data. Each value is appended as a query parameter to the URL. When the server receives this data, it converts the time received from the cell modem, to UNIX time so that the data can be used later and won't need to be converted when accessed. In addition, it stores a Mongo date object so that averages can be calculated. It then stores each query parameter in the database, using the key as the column and the value as the data. An example request is shown below in Figure 2. The time string is captured from the cell modem and each of the data fields is added on as a query parameter.

$$/data?\,time = yy/MM/dd,hh{:}mm{:}ss\&k1 = v1\&k2 = v2\ldots$$

Figure 3: Example Request for Pushing Data

The endpoint for fetching data is /fetchData/column/time. This is the endpoint that is used to get the data for Highcharts to plot. There are two parameters that are passed two this endpoint which determine the data and the time range of the data. The column parameter specifies the database column that is to be returned. The time parameter can be one of the following: live, today, yesterday, hourly average, daily average, weekly average. Each of these different times will return their respective data. If the live time is used, a third parameter can be passed to specify the number of data points that should be returned, if no third parameter is passed, it will return the most recent 50. For each request, the maximum number of data points that will be returned is 300. There is a noticeable degradation of responsiveness in the charts if there are too many points. An example of fetching the last 100 data points for k1 is shown below in Figure 4.

$$/fetchData/k1/live/100$$

Figure 4: Example Request for Fetching Data

Front End Design

In order to make the system easily expansible, I designed the user interface to that the user can dynamically generate the charts they want to see. The first step is optional, but allows a mapping between the column in the database and a display name. For example, the battery voltage might be stored in the database as *battVolt* but can be mapped to the display name of *Battery Voltage*. The user interface for this mapping is shown in Figure 5. This is helpful when creating the chart, especially if there are many database columns.

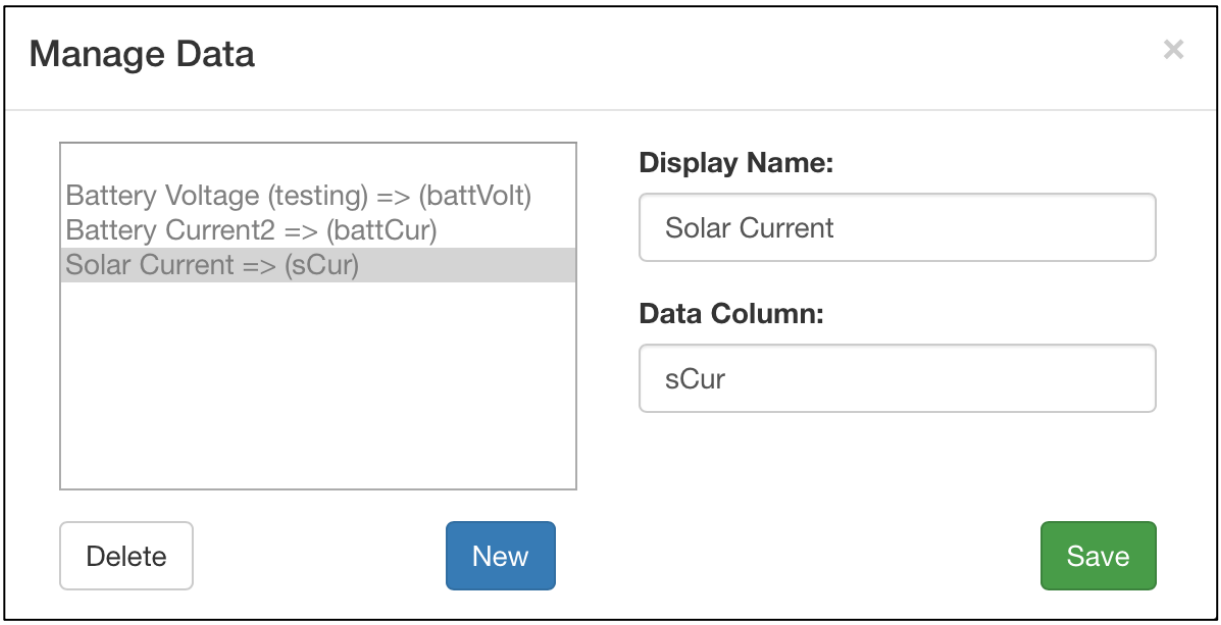


Figure 5: Data Mapping Dialog

In order to create the chart, a few parameters must be defined. They are the: chart title, database column, and time range. In order to pick the database column, one of the mappings can be chosen from a list, or the column can be directly entered. The possible values for time range are listed in the previous section about fetching data. A few more parameters will be added to allow users to set the data units and control the Y-axis range. An image of the user interface for this is shown below in Figure 6.

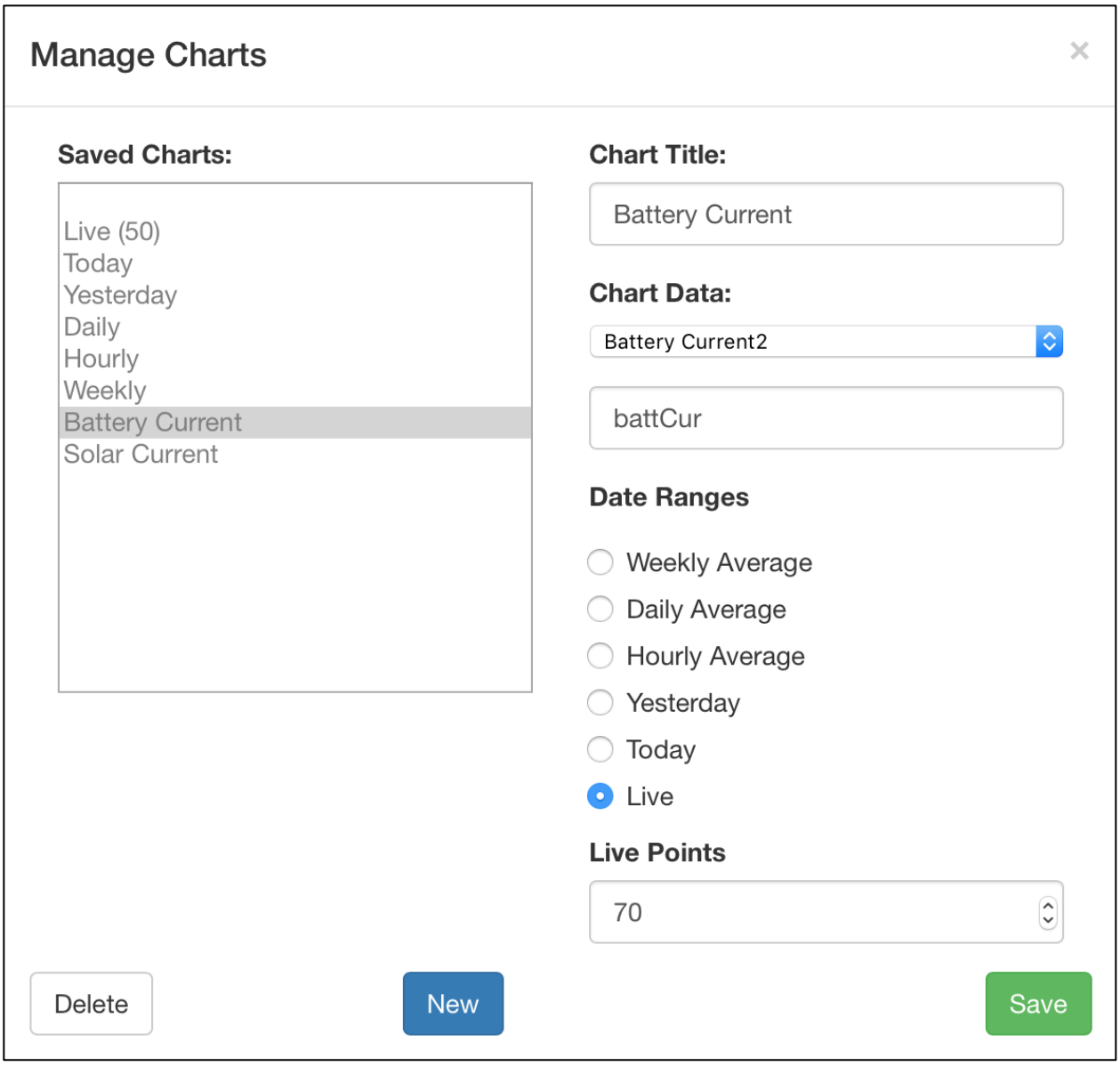


Figure 6: Dialog for Managing Charts

Once the charts are defined, they are stored as JSON objects in a separate collection in MongoDB. While the page is loaded, these objects are fetched through another API call. For each of these objects, the associated data that is required for each chart is also loaded. Once all of this is loaded, the Highcharts object is created and each chart container is dynamically added to the page so the chart can be rendered. Once rendered on the page, each chart will display its data along with any labels that have been provided through the data column mapping.

The main graph page shows a dashboard of all of the configured graphs. There are two ways to view the data, either in a list or in a 2x2 grid. At this time, there is not a way to configure the order of the graphs other than the order that they are added, a sample dashboard is shown in Figure 7. Any charts that are configured as 'live' will automatically update when new data is pushed to the server.

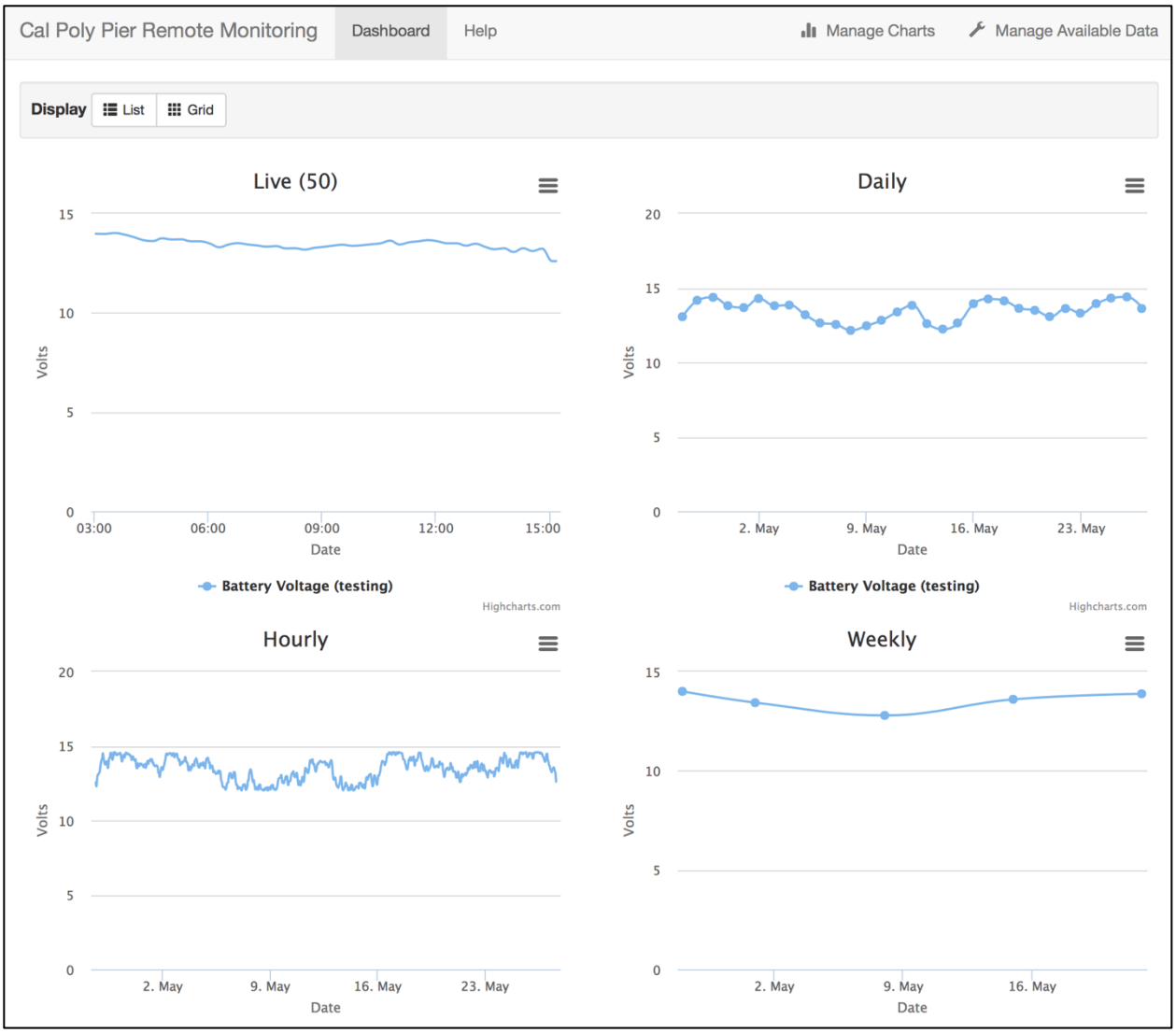


Figure 7: Charting Dashboard

Testing

Hardware

To test the system, I first tested the current monitor for the solar panel. By measuring the current of a small test circuit, I was able to verify that the circuit operated as expected. I then needed to load test the circuit to verify that it could handle the range of currents that the solar panel is expected to produce. I used a LED strip that I had at home and measured the current up to almost 5 Amps. After inspecting the device, I determined that it would be able to handle the additional load that the solar panel would produce. I then tested the other input for the battery and experiments in the same way. Table 2 shows the results of each test.

Test Case	Result	Notes
Power MSP432 from battery	PASS	
Charge FONA battery from 12V	PASS	
Solar panel voltage measurement within range	PASS	Verified with test circuit and multimeter
Solar panel current measurement within range	PASS	Verified with test circuit and multimeter
Battery voltage measurement within range	PASS	Verified with test circuit and multimeter
Experiment current measurement within range	PASS	Verified with test circuit and multimeter
Handles negative currents correctly	PASS	Verified with test circuit and multimeter
Read and send data to server	PASS	

Table 2: Test Results for Hardware

Web Server

In order to test the web server, I first opened the site on various different platforms and web browsers. I then ran a python script to insert data points for a specific time range. The complete results of testing the web server are shown below in Table 3. Some of the tests passed most of the required functionality but had some small issues. The most noticeable issue is that the Mange Data and Manage Charts buttons do not appear when the size of the browser window is too small. This is mostly an issue on mobile browsers, but may appear if the window is resized on a computer. When the button is visible on mobile browsers, the rest of the functionality is as expected.

Test Case	Result	Notes
Store data in database, verify correct	PASS	
Create data mapping user interface and operation	PASS*	Button does not appear on small browsers
Create chart user interface and operation	PASS*	Button does not appear on small browsers
Web site is operational on mobile browsers	PASS*	Charts cannot be created/modified
Create charts from user defined charts	PASS	Must manually refresh after changes
Live chart updates when new data points are pushed	PASS	
Days are handled as expected	PASS*	Time zone not handled properly
Alerts are sent when expected	PASS	

Table 3: Test Results for Web Server

Conclusion and Future Work

The core requirements of the project were completed to meet the requirements. The system successfully measures voltage and current and sends the data to a remote server. The original plan was to run the server from a Raspberry Pi but due to time constraints that is incomplete. I deployed the server software to Heroku, a cloud based hosting platform for testing. It is free to use, but must be offline for 6 hours every 24 hours. This could also be hosted on any existing servers that MBARI has or any other hosting services that support Node.js and MongoDB.

In my testing at the pier, I was unable to fully integrate it due to waterproofing concerns. In order for me to connect the system, some waterproofing material would have to be removed while I connected everything and I was unsure if I would be able to replace the waterproofing material. I was able to wire the system to the battery and saw data being pushed to the server. I was also unable to wire in the experiment that was being run, because I did not want to interfere with any data that was being collected.

This project can be extended to send data of any kind to the remote server. Additional sensors can be interfaced and the software can be extended so that additional data can be monitored as it becomes available. This could lead into fully autonomous operation of experiments run at the Cal Poly Pier. There would only be two times that would be necessary to visit the pier, once for setup and once for teardown. Everything else could monitored completely remotely. This would ensure that the experiments are working properly and allow for more time to design new experiments instead of making trips to the pier to monitor and collect data.

References

Texas Instruments. “Zero-Drift, Bi-Directional CURRENT/POWER MONITOR with I2C Interface,” INA219 datasheet, Aug. 2008 [Revised Sept. 2011].

Texas Instruments. “MAX3232 3-V to 5.5-V Multichannel RS-232 Line Driver/Receiver With ±15-kV ESD Protection,” MAX3232 datasheet, Jan. 2000 [Revised Jan. 2015].

SIMCom. “SIM800 Series_AT Command Manual_V1.01,” Jul. 2013.

Appendix

Analysis of Senior Project Design

Summary of Functional Requirements

The main goals of this project are to measure the voltage and current of the solar panel, measure the voltage of battery, measure the current drawn by the experiments, push the data to a remote server, visualize the data on server, and send alerts if any of the value are outside of the typical range set by the end user.

Primary Constraints

The most difficult part of this project was being able to handle the large range of current that may be produced from the solar panel. I needed to be able to precisely read up to 6A and as small as 0.01A. I needed to be able to measure this efficiently, dissipating the least amount of power. I was able to find an integrated circuit that contained a precision op-amp that would measure the voltage drop across a very small resistor. Another difficult part of this project was that I had to design, build, and test the entire system before I saw the existing hardware at the pier.

Economic

Estimated Cost: $150
Final Cost: $145.05

The bill of materials is included in the Appendix

Estimated development time: 348 hours

- Initial Research: 24 hours
- Prototyping: 36 hours
- PCB Design: 36 hours
- Microcontroller Software: 96 hours
- Web server software: 108 hours
- Testing/Setup: 48 hours

Actual development time: 370 hours

- Initial Research: 40 hours
- Prototyping: 40 hours
- PCB Design: 50 hours
- Microcontroller Software: 50 hours
- Web Server Software: 120 hours
- Testing/Setup: 70 hours

Environmental

The main environmental concern with the project would be if the device and related equipment fall into the ocean.

Manufacturability

If this device were to be manufactured, different soldering methods could be used for the PCB. Complete integration of the microcontroller would only be possible if a port to program it was required. The PCB would need to be redesigned as well, currently there is an IC on the bottom of the PCB. If it were commercially manufactured, this would need to be moved to the top in order to save manufacturing costs.

Sustainability

The main sustainability issue with the project is the manufacturing of the PCB and other integrated circuits. The completed system is solar powered so it is sustainable.

Ethical

The misuse of this project could be used to record measurements of anything, possibly without the permission of the equipment owner. This could potentially be used to steal data from competing companies

Health and Safety

This project has a few safety issues. The system can handle up to 8 Amps of current. If an end-user puts too much current through the board, there is a chance it could catch fire. The second safety issue is that there is a lithium polymer battery that is powering the cell modem. The user would need to be careful that nothing is able to puncture or short the battery.

Social and Political

If the charting site is public, people might become more interested in the results of these experiments and care more about the environment.

Development

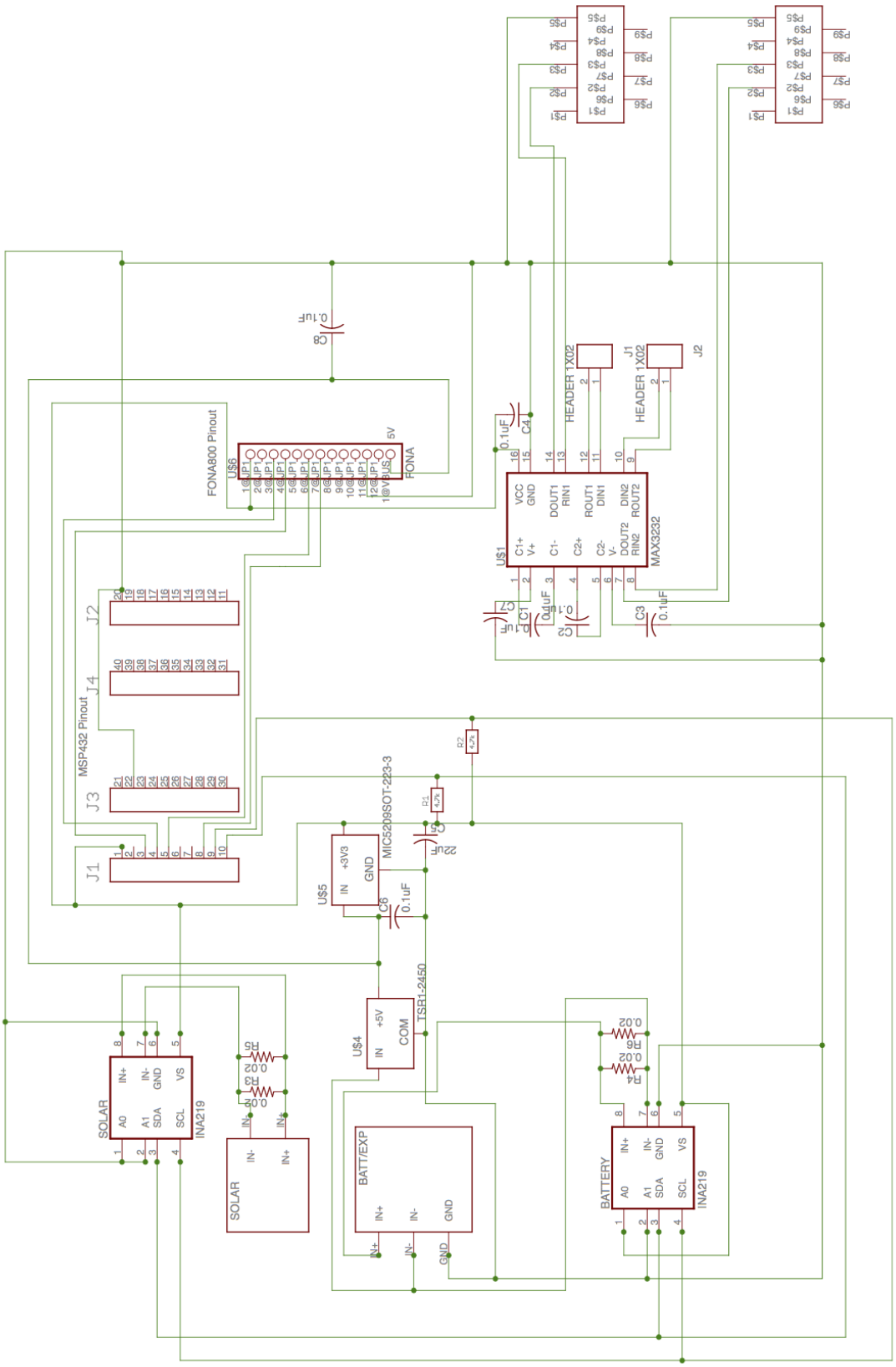
This project provided quite a few challenges that I had not experienced before. The first major challenge was fabricating a PCB. I also learned how to use Node.js and MongoDB in order to create a web application.

Bill of Materials

Item	Description	Item Cost	Quantity	Total Cost
Capacitor	0.1µF	$0.04	7	$0.27
Capacitor	22µF	$0.99	1	$0.99
Resistor	0.02Ω	$0.39	4	$1.55
Resistor	4.7kΩ	$0.15	2	$0.30
MSP432		$12.99	1	$12.99
TRACO-TSR	2450	$6.20	1	$6.20
MIC-5209	3.3YS	$1.40	2	$2.80
1x10 Female Socket		$0.99	2	$1.98
12/16-Pin Header Kit		$2.00	1	$2.00
2 Pin Male Header		$0.10	2	$0.20
2 Pin Terminal Block		$0.99	1	$0.99
3 Pin Terminal Block		$2.18	1	$2.18
RS232 Terminal		$1.75	2	$3.50
INA219	BIDR	$2.39	2	$4.78
MAX3232	CDR	$1.76	1	$1.76
FONA 800		$39.99	1	$39.99
3.7V LiPo Battery	1200mAh	$9.95	1	$9.95
PCB		$12.67	1	$12.67
Raspberry Pi 2	Model B	$39.95	1	$39.95
			Total	$145.05

Table 4: Bill of Materials

Eagle Schematic



Source Code

Microcontroller Software – Energia

```
#include <Adafruit_INA219.h>

#define FONA_PS 8
#define FONA_KEY 5

HardwareSerial *fonaSerial = &Serial1;

Adafruit_INA219 solar;
Adafruit_INA219 battery;
String fonaResponse = "";

typedef struct FONAResult {
  String response;
  boolean success;
} FONAResult;

void setup()
{
  // put your setup code here, to run once:
  Serial.begin(9600);
  Serial1.begin(9600);

  pinMode(FONA_PS, INPUT);
  pinMode(FONA_KEY, OUTPUT);
  digitalWrite(FONA_KEY, HIGH);

  solar.begin();
  solar.setCalibration_SolarPanel();

  battery.begin(0x44);
  battery.setCalibration_SolarPanel();
}

void loop()
{
  Serial.println("Started waiting...");

  turnFONAOn();
  delay(10000);
  flushFONA();

  sendCommand("AT+CSQ");

  float solarVoltage = 0;
  solarVoltage = solar.getBusVoltage_V();
  Serial.println("READ SOLAR waiting...");
  Serial.print("Measured bus voltage: ");
  Serial.println(solarVoltage);

  float solarCurrent = 0;
  solarCurrent = solar.getCurrent_mA();
  solarCurrent /= 1000.0;
  Serial.print("Measured current (A): ");
  Serial.println(solarCurrent);

  float batteryVoltage = 0;
  batteryVoltage = battery.getBusVoltage_V();
  Serial.println("\nREAD BATTERY waiting...");
  Serial.print("Measured bus voltage: ");
  Serial.println(batteryVoltage);

  float batteryCurrent = 0;
  batteryCurrent = battery.getCurrent_mA();
  batteryCurrent /= 1000.0;
  Serial.print("Measured current (A): ");
```

```
  Serial.println(batteryCurrent);
  Serial.println("");

  sendCommand("ATE0");

  // get the current time
  sendCommand("AT+CCLK?");
  int firstNdx = fonaResponse.indexOf('\"') + 1;
  int lastNdx = fonaResponse.lastIndexOf(':') + 3; // get the last : and include the seconds but
substring is exclusive
  String urlSafeTime = fonaResponse.substring(firstNdx, lastNdx);
  Serial.print("Unsafe TIME: ");
  Serial.println(urlSafeTime);
  urlSafeTime.replace("/", "%2F");
  urlSafeTime.replace(",", "%2C");
  urlSafeTime.replace(":", "%3A");
  Serial.print("  Safe TIME ");
  Serial.println(urlSafeTime);


  sendCommand("AT+CGATT=1");
  sendCommand("AT+SAPBR=3,1,\"CONTYPE\",\"GPRS\"");
  sendCommand("AT+SAPBR=3,1,\"APN\",\"epc.tmobile.com\"");
  sendCommand("AT+SAPBR=1,1");
  sendCommand("AT+HTTPINIT");
  sendCommand("AT+HTTPPARA=\"CID\",1");

  String httpCommand = "AT+HTTPPARA=\"URL\",\"http://annuzzi-monitor.herokuapp.com/data?time=";
  httpCommand.concat(urlSafeTime);
  httpCommand.concat("&battVolt=");
  httpCommand.concat(batteryVoltage);
  httpCommand.concat("&battCur=");
  httpCommand.concat(batteryCurrent);
  httpCommand.concat("&solarVolt=");
  httpCommand.concat(solarVoltage);
  httpCommand.concat("&solarCur=");
  httpCommand.concat(solarCurrent);
  httpCommand.concat("\"");

  sendCommand(httpCommand);
  sendCommand("AT+HTTPACTION=0", 2);
  sendCommand("AT+HTTPREAD", 1);
  sendCommand("AT+HTTPTERM");
  sendCommand("AT+SAPBR=0,1");

  turnFONAOff();

  delay(30000); // wait 30s

}

void sendCommand(String s) {
  sendCommand(s, 1);
}

void sendCommand(String s, int numResponse) {
  FONAResult result;
  flushFONA();
  Serial.print("Sending request: ");
  Serial.println(s);

  fonaSerial->println(s);

  while (numResponse > 0) {
    // check every 100ms if there is data
    int x = 0;
    while(!fonaSerial->available() && x <= 100) {
      x++;
      delay(100);
    }
```

```
    fonaResponse = "";
    Serial.print("Response: ");
    while(fonaSerial->available()) {
      char c = fonaSerial->read();
      if (c != 0x0A && c != 0x0D) {
        fonaResponse.concat(c);
        Serial.print(c);
      }
    }
    Serial.println("");
    Serial.println(fonaResponse);
    numResponse--;
  }
  delay(1000);
}

void flushFONA() {
  fonaSerial->flush();
}

void turnFONAOn() {
  if (digitalRead(FONA_PS)) {
    Serial.println("FONA is ON");
  } else {
      Serial.print("Turning on...");
      digitalWrite(FONA_KEY, LOW);
      delay(2000);
      digitalWrite(FONA_KEY, HIGH);
      Serial.println("success");
  }
}

void turnFONAOff() {
  if (!digitalRead(FONA_PS)) {
    Serial.println("FONA is OFF");
  } else {
      Serial.print("Turning off...");
      digitalWrite(FONA_KEY, LOW);
      delay(2000);
      digitalWrite(FONA_KEY, HIGH);
      Serial.println("success");
  }
}
```

Web Server API (Node) – Server.js

```
var express = require("express");
var path = require("path");
var bodyParser = require("body-parser");
var mongodb = require("mongodb");
var url = require("url");
var ObjectID = mongodb.ObjectID;

var CONTACTS_COLLECTION = "contacts";
var DATA_COLLECTION = "demoCollection";
var CHART_COLLECTION = "charts";
var NAMES_COLLECTION = "names";

var app = express();
app.use(express.static(__dirname + "/public"));
app.use(bodyParser.json());

// Create a database variable outside of the database connection callback to reuse the connection
pool in your app.
var db;

var HEROKU_MONGO = "mongodb://localhost"
var MAX_RECORDS = 250;

mongodb.MongoClient.connect(process.env.MONGOLAB_URI || HEROKU_MONGO, function (err, database) {
  if (err) {
    console.log(err);
    process.exit(1);
  }

  // Save database object from the callback for reuse.
  db = database;
  console.log("Database connection ready");
});

  // Initialize the app.
var server = app.listen(process.env.PORT || 8080, function () {
  var port = server.address().port;
  console.log("App now running on port", port);
});
var io = require("socket.io")(server);


// Generic error handler used by all endpoints.
function handleError(reason, message, res) {
  console.log("ERROR: " + reason);
  res.status(500).json({"error": message});
}


// Helper function to take {time: 12345678, dataCol : 12.32}
// and transform it to [12345678, 12.32] for charting purposes
function handleDataPointCallback(err, dataPoints, dataCol, live, numPoints, chartId, res) {
  if (err) {
    handleError(err.message, "Failed to get data", res);
  } else {
    var respJSON = {};
    var data = [];
    for (i = (dataPoints.length - 1); i >= 0; i--){
        data.push([dataPoints[i]['time'], dataPoints[i][dataCol]]);
    }
    respJSON['data'] = data;
    respJSON['dataCol'] = dataCol;
    respJSON['live'] = live;
    respJSON['livePoints'] = numPoints;
    respJSON['chart_id'] = chartId;
    res.status(200).json(respJSON);
  }
}
```

```
function handleAverageDataPointCallback(err, dataPoints, dataCol, chartId, res) {
  if (err) {
    handleError(err.message, "Failed to get data", res);
  } else {
    var data = [];

    for (i = 0; i < dataPoints.length; i++) {
        dataPoint = dataPoints[i];
        hours = 0;
        if (dataPoint.date.hour != undefined) {
          hours = dataPoint.date.hour;
        }
        millis = Date.UTC(dataPoint.date.year, dataPoint.date.month - 1, dataPoint.date.day,
hours, 0, 0);
        data.push([millis, Math.round(dataPoint[dataCol] * 100)/100]);
    }

    respJSON = {};
    respJSON['data'] = data;
    respJSON['dataCol'] = dataCol;
    respJSON['live'] = false;
    respJSON['livePoints'] = -1;
    respJSON['chart_id'] = chartId;
    res.status(200).json(respJSON);
  }
}

function handleDBCallback(err, data, avg, res) {
    if (err) {
      handleError(err.message, "Failed to get data", res);
    } else {
      var respJSON = {}
      respJSON['avg'] = avg;
      respJSON['data'] = data;
      res.status(200).json(respJSON);
    }
}

function getTodayMillis() {
  var date = new Date();
  date.setHours(0,0,0,0)
  return Date.UTC(date.getFullYear(), date.getMonth(), date.getDate(),
                date.getHours(), date.getMinutes(), date.getSeconds());
}

function getYesterdayMillis() {
  var date = new Date();
  date.setHours(0,0,0,0);
  date.setDate(date.getDate() - 1)
  return Date.UTC(date.getFullYear(), date.getMonth(), date.getDate(),
                date.getHours(), date.getMinutes(), date.getSeconds());
}

function isNumber(num) {
  return !isNaN(num)
}

function doMongoQueryToday(dataColumn, res, chartId) {
  var query = {time: {}}
  query["time"]["$exists"] = 1
  query["time"]["$gte"] = getTodayMillis();
  query[dataColumn] = {$exists: 1}

  var mongoRequest = createMongoQuery(dataColumn, query).limit(MAX_RECORDS);
  mongoRequest.toArray(function(err, docs) {
      handleDataPointCallback(err, docs, dataColumn, false, -1, chartId, res);
  });
}

function doMongoQueryYesterday(dataColumn, res, chartId) {
```

```
  var query = {time: {}}
  query["time"]["$exists"] = 1
  query["time"]["$gte"] = getYesterdayMillis();
  query["time"]["$lt"] = getTodayMillis();
  query[dataColumn] = {$exists: 1}

  var mongoRequest = createMongoQuery(dataColumn, query).limit(MAX_RECORDS);
  mongoRequest.toArray(function(err, docs) {
      handleDataPointCallback(err, docs, dataColumn, false, -1, chartId, res);
  });
}

// Helper function to get most recent data
function doMongoQueryLive(dataColumn, numPoints, res, chartId) {
  var query = {time: {}}
  query["time"]["$exists"] = 1
  query[dataColumn] = {$exists: 1}

  var mongoRequest = createMongoQuery(dataColumn, query).limit(numPoints);
  mongoRequest.toArray(function(err, docs) {
      handleDataPointCallback(err, docs, dataColumn, true, numPoints, chartId, res);
  });
}


function createMongoQuery(dataColumn, query) {
  // if there's no time, we cant do anything about it
  var include = {_id:0, time:1}
  include[dataColumn] = 1

  return db.collection(DATA_COLLECTION).find(query, include).sort({time : -1})
};

function doMongoQueryDailyAvg(dataColumn, res, chartId) {

  var group = {"$group" : {}}
  group["$group"]["_id"] =  {year : {"$year" : "$isodate"}, month:
      {"$month": "$isodate"}, day: {"$dayOfMonth": "$isodate"}};

  group["$group"][dataColumn] = {}
  group["$group"][dataColumn]["$avg"] = "$" + dataColumn;

  var sort = {"$sort": {'_id.year': 1, '_id.month':1, '_id.day':1}}

  var project = {"$project": {}}
  project["$project"]["date"] = "$_id";
  project["$project"][dataColumn] = 1;
  project["$project"]["_id"] = 0;

  var stages = [group, sort, project];

  return db.collection(DATA_COLLECTION).aggregate(stages).toArray(function(err, docs) {
    handleAverageDataPointCallback(err, docs, dataColumn, chartId, res);
  });
}

function doMongoQueryHourlyAvg(dataColumn, res, chartId) {

  var group = {"$group" : {}}
  group["$group"]["_id"] =  {year : {"$year" : "$isodate"}, month:
      {"$month": "$isodate"}, day: {"$dayOfMonth": "$isodate"}, hour: {"$hour": "$isodate"}};

  group["$group"][dataColumn] = {}
  group["$group"][dataColumn]["$avg"] = "$" + dataColumn;

  var sort = {"$sort": {'date.year': 1, 'date.month':1, 'date.day':1, 'date.hour':1}}

  var project = {"$project": {}}
  project["$project"]["date"] = "$_id";
  project["$project"][dataColumn] = 1;
  project["$project"]["_id"] = 0;
```

```
  var stages = [group, project, sort];

  return db.collection(DATA_COLLECTION).aggregate(stages).toArray(function(err, docs) {
    handleAverageDataPointCallback(err, docs, dataColumn, chartId, res);
  });
}

function doMongoQueryWeeklyAvg(dataColumn, res, chartId) {

  var initialSort = {"$sort": {'isodate': 1}}

  var group = {"$group" : {}}
  group["$group"]["_id"] =  {year : {"$year" : "$isodate"}, month:
      {"$month": "$isodate"}, /*day: {"$first" :{"$dayOfMonth": "$isodate"}},*/ week: {"$week":
"$isodate"}};

  group["$group"][dataColumn] = {}
  group["$group"][dataColumn]["$avg"] = "$" + dataColumn;
  group["$group"]["day"]= {"$first" : {"$dayOfMonth": "$isodate"}}

  var sort = {"$sort": {'date.year': 1, 'date.month':1, 'date.day':1}}

  var project = {"$project": {}}
  project["$project"]["date"] = {};
  project["$project"]["date"]["year"] = "$_id.year"
  project["$project"]["date"]["month"] = "$_id.month"
  project["$project"]["date"]["day"] = "$day"
  project["$project"][dataColumn] = 1;
  project["$project"]["_id"] = 0;

  var stages = [initialSort, group, project, sort];
  console.log(stages);

  return db.collection(DATA_COLLECTION).aggregate(stages).toArray(function(err, docs) {
    handleAverageDataPointCallback(err, docs, dataColumn, chartId, res);
  });
}

// this is backwards get doing a post but is much simpler
// for the client
app.get("/data", function(req, res) {
  //var reqJson = "{"
  var query = url.parse(req.url, true).query;

  for (var key in query) {
    var value = query[key]
    if (isNumber(value)) {
      query[key] = +value;
    }
  }

  // if there are parameters, then we're adding stuff
  if (query['time'] != null){

    // time in format yy/MM/dd,hh:mm:ss
    var datetime = query['time'];

    var date = datetime.split(",")[0]
    var time = datetime.split(",")[1]

    var dateSplit = date.split("/")
    var timeSplit = time.split(":")

    //for the year we only have a 2 digit year so it must be converted
    var yr = +dateSplit[0]
    var year = yr + (yr < 16 ? 2100 : 2000)

    var d = Date.UTC(year, dateSplit[1] - 1, dateSplit[2], timeSplit[0], timeSplit[1],
timeSplit[2])
    var isodate = new Date(d);
```

```
    console.log(d)

    query['time'] = d
    query['isodate'] = isodate

    db.collection(DATA_COLLECTION).insertOne(query, function(err, doc) {
      if (err) {
        handleError(err.message, "Failed to add data", res);
      } else {
        io.sockets.emit("newData", query)
        res.status(200).end("OK");
      }
    });
  } else {
    res.status(400).end("ERROR");
  }
});

app.delete("/alldata", function(req, res) {
  db.collection(DATA_COLLECTION).deleteMany({}, function(err, doc) {
    if (err) {
      handleError(err.message, "Failed to delete all", res);
    } else {
      res.status(200).end();
    }
  });
});

app.get("/alldata", function(req, res){
  db.collection(DATA_COLLECTION).find({}).toArray(function(err, docs) {
      handleDBCallback(err, docs, false, res)
    });
});

app.get("/voltage", function(req, res) {
  var query = url.parse(req.url, true).query;
  var time = query['time']
  if (time == "today" || time == "yesterday") {
    console.log(query['time']);
      var date = new Date();
      date.setHours(0, 0, 0, 0)
      console.log(date);
      date = Date.UTC(date.getFullYear(), date.getMonth(), date.getDate(), date.getHours(),
date.getMinutes(), date.getSeconds());
      console.log(date);
      if (time == "yesterday") {
        var yday = date - 1000*60*60*24;
        console.log(date)
        console.log(yday)
        db.collection(DATA_COLLECTION).find(
        {battVolt:{$exists:true}, time:{$gte: yday, $lt: date}},
        {_id:0, battVolt:1, time:1}).sort({time: 1}).toArray(function(err, docs) {
          if (err) {
            handleError(err.message, "Failed to get voltages", res);
          } else {
            res.status(200).json(docs);
          }
        });
      } else {
        db.collection(DATA_COLLECTION).find(
          {battVolt:{$exists:true}, time:{$gte: date}},
          {_id:0, battVolt:1, time:1}).sort({time: 1}).toArray(function(err, docs) {
            if (err) {
              handleError(err.message, "Failed to get voltages", res);
            } else {
              res.status(200).json(docs);
            }
          });
      }
  } else {
      db.collection(DATA_COLLECTION).find({}).sort({time : 1}).limit(150).toArray(function(err,
```

```
docs) {
          if (err) {
            handleError(err.message, "Failed to get voltages", res);
          } else {
            res.status(200).json(docs);
          }
        });
  }
});

/* "/fetchData/:data/:time"
 *  GET: return "data" for the specified "time"
 *  Acceptable values for time: live, today, yesterday, hourly, daily, weekly
 *  NOTE: Each of these endpoints will return AT MOST MAX_RECORDS entries
 */
 app.get("/fetchData/:data/:time", function(req, res) {
    var queryData = req.params.data;
    var queryTime = req.params.time;

    var chartId = req.query.chartId;

    if (queryTime === "today") {
      doMongoQueryToday(queryData, res, chartId);
    } else if (queryTime === "yesterday") {
      doMongoQueryYesterday(queryData, res, chartId);
    } else if (queryTime === "hourly") {
      doMongoQueryHourlyAvg(queryData, res, chartId);
    } else if (queryTime === "daily") {
      doMongoQueryDailyAvg(queryData, res, chartId);
    } else if (queryTime === "weekly") {
      doMongoQueryWeeklyAvg(queryData, res, chartId);
    } else if (queryTime === "live") {
      doMongoQueryLive(queryData, 50, res, chartId);
    } else {
      res.status(404).end();
    }
 });

 /* "/fetchData/:data/live/:numPoints"
  *  GET: return most recent data up to numPoints
  */
 app.get("/fetchData/:data/live/:numPoints", function(req, res) {
    if (isNaN(req.params.numPoints)) {
      res.status(400).end();
    }

    var queryData = req.params.data;
    var queryPoints = +req.params.numPoints;

    var chartId = req.query.chartId;

    console.log("here")

    if (queryPoints > MAX_RECORDS) {
      res.status(400).end("Number of points cannot be greater than " + MAX_RECORDS);
    } else if (queryPoints < 10) {
      res.status(400).end("Must request at least 10 points");
    } else {
      doMongoQueryLive(queryData, queryPoints, res, chartId);
    }
 });

app.get("/charts", function(req, res) {
  db.collection(CHART_COLLECTION).find({}).toArray(function(err, docs) {
    if (err) {
      handleError(err.message, "Failed to get charts.", res);
    } else {
      res.status(200).json(docs);
    }
  });
});
```

```
app.post("/charts", function(req, res) {
  db.collection(CHART_COLLECTION).insertOne(req.body, function(err, docs) {
    if (err) {
      handleError(err.message, "Failed to post NAME", res);
    } else {
      res.status(201).json(docs.ops[0])
    }
  });
});

app.patch("/charts/:id", function(req, res) {
  db.collection(CHART_COLLECTION).update({_id: new ObjectID(req.params.id)}, {$set : req.body},
function(err, docs) {
    if (err) {
        handleError(err.message, "Failed to modify item", res);
    } else {
      res.status(200).end("OK");
    }
  })
});

app.delete("/charts/:id", function(req, res) {
  db.collection(CHART_COLLECTION).deleteOne({_id: new ObjectID(req.params.id)}, function(err,
docs) {
    if (err) {
      handleError(err.message, "Failed to delete item", res);
    } else {
      res.status(200).end("OK");
    }
  })
});

app.get("/dataNames", function(req, res) {
  db.collection(NAMES_COLLECTION).find({}).toArray(function(err, docs) {
    if (err) {
      handleError(err.message, "Failed to get names.", res);
    } else {
      res.status(200).json(docs);
    }
  });
});

app.post("/dataNames", function(req, res) {
  db.collection(NAMES_COLLECTION).insertOne(req.body, function(err, docs) {
    if (err) {
      if (err.code === 11000) {
        res.status(400).json({})
      } else {
        handleError(err.message, "Failed to post NAME", res);
      }
    } else {
      res.status(201).json(docs.ops[0])
    }
  });
});

app.delete("/dataNames/:id", function(req, res) {
  db.collection(NAMES_COLLECTION).deleteOne({_id: new ObjectID(req.params.id)}, function(err,
docs) {
    if (err) {
      handleError(err.message, "Failed to delete item", res);
    } else {
      res.status(200).end("OK");
    }
  })
})

app.patch("/dataNames/:id", function(req, res) {
  db.collection(NAMES_COLLECTION).update({_id: new ObjectID(req.params.id)}, {$set : req.body},
function(err, docs) {
```

```
    if (err) {
      if (err.code === 11000) {
        res.status(400).json({})
      } else {
        handleError(err.message, "Failed to modify item", res);
      }
    } else {
      res.status(200).end("OK");
    }
  })
})
```

Controller – App.js

```
angular.module("contactsApp", ['ngRoute'])
    .config(function($routeProvider) {
        $routeProvider
            .when("/dashboard", {
                templateUrl: "livedata.html",
                controller: "ChartController",
                resolve: {
                    charts: function(Monitor) {
                        return Monitor.getCharts();
                    },
                    names: function(Monitor) {
                        return Monitor.getNames();
                    }
                }
            })
            .when("/help", {
                controller: "HelpController",
                templateUrl: "help.html"
            })
            .otherwise({
                redirectTo: "/dashboard"
            })
    })
    .factory('socket', ['$rootScope', function($rootScope) {
        var socket = io.connect();

        return {
            on: function(eventName, callback) {
                socket.on(eventName, callback);
            }
        };
    }])
    .service("Monitor", function($http) {

        this.getYesterdayVoltage = function() {
            return $http.get("/fetchData/battVolt/all").
                then(function(response) {
                    return response;
                }, function(response) {
                    alert("Error getting todays voltages.")
            });
        }

        this.getCharts = function() {
            return $http.get("/charts").
                then(function(response) {
                    return response;
                }, function(response) {
                    alert("Error getting charts json.")
                });
        }

        this.getNames = function() {
            return $http.get("/dataNames").
                then(function(response) {
                    return response;
                }, function(response) {
                    alert("Error getting data names.")
                });
        }
        this.saveDataEntry = function(entry) {
            var url = "/dataNames";
            console.log(entry);
            return $http.post(url, entry).
            then(function(response) {
                    return response;
                }, function(response) {
                    if (response.status === 400) {
                        return false;
```

```
            }
            alert("Error posting response.");
        });
}

this.editDataEntry = function(entry) {
    var url = "/dataNames/" + entry._id;
    console.log(entry);

    var patch = {dataCol: entry.dataCol, displayName: entry.displayName};

    return $http.patch(url, patch).
    then(function(response) {
            return response;
        }, function(response) {
            if (response.status === 400) {
                return false;
            }
            alert("Error editing data.")
        });
}

this.deleteName = function(id) {
    var url = "/dataNames/" + id;
    return $http.delete(url).
        then(function(response) {
            return response;
        }, function(response) {
            alert("Error deleting this id.");
            console.log(response);
        });
}

this.deleteChart = function(id) {
    var url = "/charts/" + id;
    return $http.delete(url).
        then(function(response) {
            return response;
        }, function(response) {
            alert("Error deleting this chart.");
            console.log(response);
        });
}

this.saveChartEntry = function(entry) {
    var url = "/charts";
    return $http.post(url, entry).
    then(function(response) {
            return response;
        }, function(response) {
            alert("Error posting response.");
        });
}

this.editChartEntry = function(entry) {
    var url = "/charts/" + entry._id;
    delete entry._id;
    return $http.patch(url, entry).
        then(function(response) {
            return response
        }, function(response) {
            alert("Error editing chart.")
        })
}

this.getChartData = function(dataCol, time, livePoints) {
    var url = "/fetchData/" + dataCol + "/" + time;
    if (time === 'live') {
        url += "/" + livePoints;
    }
    console.log(url);
```

```
            return $http.get(url).
                then(function(response) {
                    return response;
                }, function(response) {
                    alert("Error getting data for " + url);
                })
        }

        this.getChartUrl = function(dataCol, time, livePoints, id) {
            var url = "/fetchData/" + dataCol + "/" + time;
            if (time === 'live') {
                url += "/" + livePoints;
            }

            url += "?chartId=" + id;
            console.log(url);

            return $http.get(url);
        }
    })
    .controller("ChartController", function(charts, names, $scope, $q, Monitor, socket) {
        $scope.charts = charts.data;
        $scope.names = names.data;

        /* Scope Variables for DisplayName Modal */
        $scope.selectedName = null;
        $scope.isNewEntry = false;
        $scope.isEditing = false;
        $scope.errors = {name: false, col: false, duplicate: false, unsaved: false};
        $scope.newItem = null;

        /* Scope Variables for Charts Modal */
        $scope.selectedChart = null;
        $scope.isNewChart = false;
        $scope.isEditingChart = false;
        $scope.newChartItem = null;
        $scope.chartErrors = {title: false, data: false, time: false, livePoints: false};

        /* Scope Variable for Chart Data */
        $scope.chartData = [];
        var highchartObjects = {};

        socket.on('newData', function(result) {
            $scope.$apply(function(data) {
                console.log(result);
                console.log("-----------------------------");
                $scope.chartData.forEach(function(chart) {
                    console.log(chart);
                    if (chart.live && result[chart.dataCol] != undefined) {
                        console.log("Live: " + chart.chart_id + " " + chart.live);
                        console.log("Data: " + chart.dataCol);
                        console.log(result[chart.dataCol]);


                        var chartElement = angular.element(document.querySelector("#c" +
chart.chart_id)).highcharts();
                        console.log("Data Size " + chartElement.series[0].data.length);
                        console.log("Num expected " + chart.livePoints);
                        if (chartElement.series[0].data.length < chart.livePoints) {
                            chartElement.series[0].addPoint([result.time, result[chart.dataCol]],
true, false)
                        } else {
                            chartElement.series[0].addPoint([result.time, result[chart.dataCol]],
true, true)
                        }
                    }

                    /*if (chart.live && data[chart.dataCol] !== undefined) {
                        console.log("Update " + chart.chart_id);
                    }*/
```

```
            console.log(chart);
        });
    })
    //$scope.apply(function)
});

$scope.getTodayVoltage = function() {
    Monitor.getTodayBatteryVoltage().then(function(response) {
        $scope.voltage = response.data;
    }, function(reponse) {
        alert(response);
    });
}

$scope.getAllVoltage = function() {
    Monitor.getBatteryVoltage().then(function(response) {
        $scope.voltage = response.data;
    }, function(reponse) {
        alert(response);
    });
}

$scope.getYesterdayVoltage = function() {
    Monitor.getYesterdayVoltage().then(function(response) {
        $scope.voltage = response.data;
    }, function(reponse) {
        alert(response);
    });
}

$scope.setSelection = function(data) {
    console.log(data);
    $scope.selectedName = data;
    $scope.isEditing = false;
    $scope.isNewEntry = false;
    $scope.errors = {name: false, col: false, duplicate: false, unsaved: false};
}

$scope.deleteSelection = function() {
    console.log("Deleting " + $scope.selectedName.dataCol);
    if ($scope.selectedName._id === undefined) {
        $scope.names.pop();
        $scope.selectedName = null;
        $scope.isNewEntry = false;
        $scope.isEditing =  false;
        $scope.isDuplicateEntry = false;
    } else {
        Monitor.deleteName($scope.selectedName._id).then(function(response) {
            $scope.selectedName = null;
            $scope.isEditing = false;
            $scope.isNewEntry = false;
            $scope.isDuplicateEntry = false;
            Monitor.getNames().then(function(response){
                $scope.names = response.data;
            });
        });
    }
}

$scope.editEntry = function() {
    $scope.isEditing = true;
}

$scope.saveEntry = function() {
    $scope.errors.unsaved = false;
    if ($scope.isNewEntry) {
        doSaveOrError();
    } else if ($scope.isEditing) {
        doEditOrError();
    }
}
```

```
$scope.newEntry = function() {
    if ($scope.isNewEntry || $scope.isEditing) {
        $scope.errors.unsaved = true;
    } else {
        $scope.newItem = {dataCol: "", displayName: ""};
        $scope.names.push($scope.newItem);
        $scope.selectedName = $scope.newItem;
        $scope.isNewEntry = true;
    }
}

$scope.$watch('selectedName.displayName', function() {
    $scope.errors.name = false;
})

$scope.$watch('selectedName.dataCol', function() {
    $scope.errors.col = false;
    $scope.errors.duplicate = false;
})

// this is triggered whenever a new item in the list is selected
$scope.$watch('selectedName.$$hashKey', function(oldValue, newValue) {
    if ($scope.isNewEntry && $scope.selectedName === $scope.newItem) {
        console.log("same");
    } else if ($scope.isEditing) {
        console.log(oldValue);
        console.log(newValue);
        oldValue = $scope.editItem;
    } else {
        if ($scope.isNewEntry) {
            $scope.names.pop();
        }
        $scope.isEditing = false;
        $scope.isNewEntry = false;
        $scope.errors = {name: false, col: false, duplicate: false, unsaved: false};
    }
})

$scope.$watch('selectedChart.dataCol', function() {
    $scope.chartErrors.dataCol = false;
})

$scope.$watch('selectedChart.title', function() {
    $scope.chartErrors.title = false;
})

$scope.$watch('selectedChart.time', function() {
    $scope.chartErrors.time = false;
})

$scope.$watch('selectedChart.livePoints', function() {
    $scope.chartErrors.livePoints = false;
})


function verifyFields() {
    var result = true;
    if ($scope.selectedName == null || $scope.selectedName.displayName == undefined
            || $scope.selectedName.displayName === "") {

        $scope.errors.name = true;
        result = false;
    }
    if ($scope.selectedName == null || $scope.selectedName.dataCol == undefined
            || $scope.selectedName.dataCol === "") {

        $scope.errors.col = true;
        result = false;
    }
    return result;
```

```
        }

        function verifyChartFields() {
            var result = true;
            if ($scope.selectedChart == null || $scope.selectedChart.title == undefined
                    || $scope.selectedChart.title === "") {

                    $scope.chartErrors.title = true;
                    result = false;
            }
            if ($scope.selectedChart == null || $scope.selectedChart.dataCol == undefined
                    || $scope.selectedChart.dataCol === "") {

                    $scope.chartErrors.dataCol = true;
                    result = false;
            }
            if ($scope.selectedChart == null || $scope.selectedChart.time == undefined
                    || $scope.selectedChart.time === "") {

                    $scope.chartErrors.time = true;
                    result = false;
            }
            if ($scope.selectedChart == null || $scope.selectedChart.livePoints === undefined
                    || $scope.selectedChart.livePoints === "" ) {

                    $scope.chartErrors.livePoints = true;
                    result = false;
            } else if (+$scope.selectedChart.livePoints > 250 || +$scope.selectedChart.livePoints
< 10) {
                $scope.chartErrors.livePoints = true;
                result = false;
            }
            return result;
        }

        function doEditOrError() {
            if (verifyFields()) {
                Monitor.editDataEntry($scope.selectedName).then(function(response) {
                    if (response === false) {
                        $scope.errors.duplicate = true;
                    } else {
                        Monitor.getNames().then(function(response){
                            $scope.names = response.data;
                            resetScope();
                        });
                    }
                });
            }
        }

        function doSaveOrError() {
            if (verifyFields()) {
                Monitor.saveDataEntry($scope.selectedName).then(function(response) {
                    if (response === false) {
                        $scope.errors.duplicate = true;
                    } else {
                        Monitor.getNames().then(function(response){
                            $scope.names = response.data;
                            resetScope();
                        });
                    }
                });
            }
        }

        function resetScope() {
            $scope.isEditing = false;
            $scope.isNewEntry = false;
            $scope.errors = {name: false, col: false, duplicate: false};
            $scope.selectedName = null;
            $scope.newItem = null;
```

```
}

function resetChartScope() {
    $scope.selectedChart = null;
    $scope.isNewChart = false;
    $scope.isEditingChart = false;
    $scope.newChartItem = null;
    $scope.chartErrors = {title: false, data: false, time: false, livePoints: false};
}

/* Functions for chart management */
$scope.deleteChart = function() {
    Monitor.deleteChart($scope.selectedChart._id).then(function(response) {
            $scope.selectedChart = null;
            Monitor.getCharts().then(function(response){
                $scope.charts = response.data;
            });
        });
}

$scope.newChart = function() {
    $scope.newChartItem = {dataCol: "", title: "", livePoints: 50, time:""};
    $scope.charts.push($scope.newChartItem);
    $scope.selectedChart = $scope.newChartItem;
    $scope.isNewChart = true;
}

$scope.editChart = function() {
    $scope.isEditingChart = true;
}

$scope.saveChart = function() {
    if ($scope.isEditingChart) {
        doSaveEditChart();
    } else if ($scope.isNewChart) {
        doSaveNewChart();
    }
}

function doSaveNewChart() {
    if (verifyChartFields()) {
        Monitor.saveChartEntry($scope.selectedChart).then(function (response) {
            Monitor.getCharts().then(function (response) {
                $scope.charts = response.data;
                resetChartScope();
            });
        });
    }
}

function doSaveEditChart() {
    if (verifyChartFields()) {
        Monitor.editChartEntry($scope.selectedChart).then(function(response) {
            Monitor.getCharts().then(function (response) {
                $scope.charts = response.data;
                resetChartScope();
            })
        })
    }
}

angular.element(document).ready(function() {
    //console.log(JSON.stringify($scope))
    loadAllChartData();
})

function loadAllChartData(callback) {
    var chartDataUrls = [];
    $scope.charts.forEach(function(entry) {
        chartDataUrls.push(generateChartRequest(entry));
        /*loadChartDataForEntry(entry, function(data){
```

```
            $scope.chartData.push(data);
            highchartObjects.push(generateHighChartObject(data));
        });*/
    })
    $q.all(chartDataUrls).then(function(results) {
        results.forEach(function(result) {
            var result = result.data;
            result.dataDisplayName = getDataDisplayName(result.dataCol);
            result.title = getChartTitle(result.chart_id);

            if (result.dataCol === "battVolt") {
                result.y_min = 0;
            }

            if (result.dataCol.includes("Volt")) {
                result.y_axis = "Volts"
            } else {
                result.y_axis = "Amps"
            }

            $scope.chartData.push(result);
            highchartObjects[result.chart_id] = generateHighChartObject(result);
            console.log(result)
        })
        console.log(highchartObjects);

        //draw each chart
        var mainChartsContainer = angular.element(document.querySelector("#charts"));
        $scope.chartData.forEach(function(chartData) {
            //var chart = highchartObjects[0]
            var chartGrid = document.createElement("div");
            chartGrid.className = "chart col-xs-6 col-lg-6 grid-group-item"

            var chart = highchartObjects[chartData.chart_id];

            var chartElement = document.createElement("div");
            chartElement.id = "c" + chart.chart.id;

            chartGrid.appendChild(chartElement);
            mainChartsContainer.append(chartGrid);
            Highcharts.chart(chartElement, chart);
        })
    });
    }


function getDataDisplayName(dataCol) {
    if ($scope.names != null) {
        for (var i = 0; i < $scope.names.length; i++) {
            if ($scope.names[i].dataCol === dataCol) {
                console.log("match")
                return $scope.names[i].displayName;
            }
        }
    }
    console.log("none")
    return dataCol;
}

function getChartTitle(chartId) {
    if ($scope.charts != null) {
        for (var i = 0; i < $scope.charts.length; i++) {
            if ($scope.charts[i]._id === chartId) {
                console.log("match")
                return $scope.charts[i].title;
            }
        }
    }
    console.log("none")
    return "";
}
```

```
        function generateHighChartObject(chartData) {
            return chartObj =
                {
                    chart: {
                        type: "spline",
                        id: chartData.chart_id
                    },
                    title: {
                        text: chartData.title
                    },
                    xAxis: {
                        type: 'datetime',
                        dateTimeLabelFormats: {
                          /*second: '%l:%M:%S %P',
                          minute: '%l:%M %P',
                          hour: '%l:%M %P',
                          day: '%b %e',
                          week: '%b %e',
                          month: '%b',
                          year: '%Y'*/
                          month: '%b %#d',
                          year: '%G'
                        },
                        title: {
                            text: 'Date'
                        }
                    },
                    yAxis: {
                        title: {
                            text: chartData.y_axis
                        },
                        min: chartData.y_min
                    },/*
                    tooltip: {
                      dateTimeLabelFormats: {
                          second: '%l:%M:%S %P',
                          minute: '%l:%M %P',
                          hour: '%l:%M %P',
                          day: '%b %e',
                          week: '%b %e',
                          month: '%b',
                          year: '%Y'
                      }
                    },*/
                    series: [{
                      name: chartData.dataDisplayName,
                        data: chartData.data
                    }]
                };
        }

        function loadChartDataForEntry(entry, callback) {
            Monitor.getChartData(entry.dataCol, entry.time,
entry.livePoints).then(function(response) {
                var result = response.data;
                result["title"] = entry.title;
                result["dataDisplayName"] = getDataDisplayName(entry.dataCol);
                console.log(result);
                callback(result);
            });
        }

        function generateChartRequest(entry) {
            return Monitor.getChartUrl(entry.dataCol, entry.time, entry.livePoints, entry._id);
        }

        function resizeAllCharts() {
            $("[data-highcharts-chart]").each(function () {
                var highChart = Highcharts.charts[$(this).data('highchartsChart')];
                highChart.reflow();
```

```
        });
    }

    $scope.listClicked = function() {
        console.log("list");
        var chartElements = $('#charts .grid-group-item')
        chartElements.addClass('list-group-item');

        if (chartElements.length > 0) {
            resizeAllCharts();
        }
    }

    $scope.gridClicked = function() {
        console.log("grid");
        var gridElements = $('#charts .list-group-item')
        gridElements.removeClass('list-group-item');
        //$('#charts .chart').addClass('grid-group-item');
        if (gridElements.length > 0) {
            resizeAllCharts();
        }
    }
});
```

Index.html

```
<!DOCTYPE html>
<html lang="en" ng-app="contactsApp">
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <!-- The above 3 meta tags *must* come first in the head; any other head content must come
*after* these tags -->
    <!-- jQuery (necessary for Bootstrap's JavaScript plugins) -->
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/2.1.4/jquery.min.js"></script>
    <script src="https://ajax.googleapis.com/ajax/libs/jqueryui/1.11.4/jquery-
ui.min.js"></script>

    <!-- Latest compiled and minified JavaScript -->
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/js/bootstrap.min.js"></script>
    <!-- Angular lib -->
    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.6/angular.min.js"></script>
    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.6/angular-
route.min.js"></script>

    <script src="https://cdnjs.cloudflare.com/ajax/libs/socket.io/1.4.6/socket.io.js"></script>


    <!-- testing for highcharts -->
    <script src="https://code.highcharts.com/highcharts.js"></script>
    <script src="https://code.highcharts.com/modules/exporting.js"></script>

    <!-- Latest compiled and minified CSS -->
    <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css">
    <link rel="stylesheet"
href="https://ajax.googleapis.com/ajax/libs/jqueryui/1.11.4/themes/smoothness/jquery-ui.css">

    <!-- Create some padding for body -->
    <style>
      body {
        padding-top: 70px;
      }

      #id {
        background-color: #444333;
      }
      select {
        width: 100%;
      }
      .error {
        color: red;
      }
      .modal-open {
        position: fixed;
      }
      .glyphicon {
          margin-right:5px;
        }

      .chart.list-group-item
      {
          float: none;
          width: 100%;
          background-color: #eeeeee;
      }

      .chart.list-group-item:before, .chart.list-group-item:after
      {
          display: table;
          content: " ";
      }

      .chart.list-group-item:after
```

```
        {
            clear: both;
        }

        .highcharts-container
        {
            width:100% !important;
            height:100% !important;
        }
    </style>

    <script src="/js/app.js"></script>
  </head>

  <body>
    <nav class="navbar navbar-default navbar-fixed-top">
      <div class="container-fluid">
        <div class="navbar-header">
          <div class="navbar-brand" href="#/">Cal Poly Pier Remote Monitoring</div>
        </div>
        <div id="navbar" class="navbar-collapse collapse">
          <ul class="nav navbar-nav">
            <li class="active"><a href="#/dashboard">Dashboard</a></li>
            <li><a href="#/help">Help</a></li>
          </ul>
          <ul class="nav navbar-nav navbar-right">
            <li><a style="cursor:pointer" onclick=$('#mChartModal').modal('show');><span
class="glyphicon glyphicon-stats"></span> Manage Charts</a></li>
            <li><a style="cursor:pointer" onclick=$('#mDataModal').modal('show');><span
class="glyphicon glyphicon-wrench"></span> Manage Available Data</a></li>
          </ul>
        </div>
      </div>
    </nav>
    <!-- Dynamically render templates -->
    <div class="container-fluid" ng-view>
    </div>
  </body>
</html>
```

Livedata.html

```
<div class="stuff">
  <div class="well well-sm">
    <strong>Display</strong>
    <div class="btn-group">
      <a ng-click="listClicked()" id="list" class="btn btn-default btn-sm"><span class="glyphicon
glyphicon-th-list"></span>List</a>
      <a ng-click="gridClicked()" id="grid" class="btn btn-default btn-sm"><span class="glyphicon
glyphicon-th"></span>Grid</a>
    </div>
  </div>

<!-- Modal for chart management -->
<div class="modal fade" id="mChartModal" tabindex="-1" role="dialog" aria-
labelledby="mChartModal" aria-hidden="true">
    <div class="modal-dialog">
        <div class="modal-content">
            <div class="modal-header">
            <button type="button" class="close" data-dismiss="modal" aria-
hidden="true">&times;</button>
            <h4 class="modal-title" id="myModalLabel">Manage Charts</h4>
            </div>
            <div class="modal-body">
              <div class="container-fluid">
                <div class="row">
                  <div class="col-sm-6">
                    <div class="form-group">
                      <label for="charts-select">Saved Charts:</label>
                      <select name="reg_email" id="charts-select" size="16" width="100%" ng-
model="selectedChart" ng-options="chart as chart.title for chart in charts" ng-
disabled="isNewChart || isEditingChart">
                        <option ng-hide="true" value=""></option>
                      </select>
                    </div>


                  </div>
                  <div class="col-sm-6" id="test">
                    <form role="form">
                      <div class="form-group">
                        <label for="chart-title">Chart Title:</label>
                        <input type="text" class="form-control" id="chart-title" ng-
model="selectedChart.title" ng-disabled="!isNewChart && !isEditingChart">

                        <p ng-hide="!chartErrors.title" class="error">This field is required</p>
                      </div>
                      <div class="form-group">
                        <label for="chart-data">Chart Data:</label>
                        <select name="chart-data" ng-model="selectedChart.dataCol" ng-
options="data.dataCol as data.displayName for data in names" ng-disabled="!isNewChart &&
!isEditingChart">
                        </select>
                      </div>
                        <div class="form-group">
                          <input type="text" class="form-control" id="chart-data-col" ng-
model="selectedChart.dataCol" ng-disabled="!isNewChart && !isEditingChart">
                          <p ng-hide="!chartErrors.dataCol" class="error">This field is
required</p>
                        </div>

                      <div class="form-group">
                        <label for="chart-time">Date Ranges</label>
                        <div class="radio">
                          <label><input type="radio" name="chart-time" ng-
model="selectedChart.time" value="weekly" ng-disabled="!isNewChart && !isEditingChart">Weekly
Average</label>
                        </div>
                        <div class="radio">
```

```
                        <label><input type="radio" name="chart-time" ng-
model="selectedChart.time" value="daily" ng-disabled="!isNewChart && !isEditingChart">Daily
Average</label>
                      </div>
                      <div class="radio">
                        <label><input type="radio" name="chart-time" ng-
model="selectedChart.time" value="hourly" ng-disabled="!isNewChart && !isEditingChart">Hourly
Average</label>
                      </div>
                      <div class="radio">
                        <label><input type="radio" name="chart-time" ng-
model="selectedChart.time" value="yesterday" ng-disabled="!isNewChart &&
!isEditingChart">Yesterday</label>
                      </div>
                      <div class="radio">
                        <label><input type="radio" name="chart-time" ng-
model="selectedChart.time" value="today" ng-disabled="!isNewChart &&
!isEditingChart">Today</label>
                      </div>
                      <div class="radio">
                        <label><input type="radio" name="chart-time" ng-
model="selectedChart.time" value="live" ng-disabled="!isNewChart && !isEditingChart">Live</label>
                      </div>
                      <div class="form-group" ng-hide="selectedChart.time !== 'live'">
                        <label for="live-points">Live Points</label>
                        <input type="number" class="form-control" id="live-points" ng-
model="selectedChart.livePoints" ng-disabled="!isEditingChart && !isNewChart">
                        <p ng-hide="!chartErrors.livePoints" class="error">Please enter a value
between 10-250.</p>
                      </div>
                      <p ng-hide="!chartErrors.time" class="error">This field is required</p>
                    </div>
                  </form>
                </div>
              </div>
            </div>
            <div class="row">
                    <div class="form-group">

                      <div class="col-sm-4">
                        <button type="button" class="btn btn-default" ng-
click="deleteChart()">Delete</button>
                      </div>
                      <div class="col-sm-4">
                        <button type="button" class="btn btn-primary pull-center" ng-
click="newChart()">New</button>
                      </div>
                      <div class="col-sm-4">
                        <button type="button" class="btn btn-primary pull-right" ng-
click="editChart()" ng-hide="isEditingChart || isNewChart" ng-disabled="selectedChart ==
null">Edit</button>

                        <button type="button" class="btn btn-success pull-right" ng-
click="saveChart()" ng-hide="!isEditingChart && !isNewChart">Save</button>
                      </div>
                    </div>

          </div>
        </div>

    </div>
  </div>
</div>


<!-- Modal for choosing data columns to Display Names -->
<div class="modal fade" id="mDataModal" tabindex="-1" role="dialog" aria-labelledby="mDataModal"
aria-hidden="true">
    <div class="modal-dialog">
        <div class="modal-content">
            <div class="modal-header">
```

```
            <button type="button" class="close" data-dismiss="modal" aria-
hidden="true">&times;</button>
            <h4 class="modal-title" id="myModalLabel">Manage Data</h4>
            </div>
            <div class="modal-body">
              <div class="container-fluid">
                <p class="error" ng-hide="!errors.unsaved">There is unsaved data, please save and
try again</p>
                <div class="row">
                  <div class="col-sm-6">
                    <div class="form-group">
                      <select name="data" id="chart-data" size="10" ng-model="selectedName" ng-
options='data as (data.displayName + " => (" + (data.dataCol) + ")") for data in names' ng-
selected='selectedName == data'ng-disabled="isEditing || isNewEntry">
                        <option value="" ng-hide="true"></option>
                      </select>
                    </div>
                  </div>
                  <div class="col-sm-6">
                    <form role="form">
                      <div class="form-group">
                        <label for="display-name">Display Name:</label>
                        <input type="text" class="form-control" id="display-name" ng-
model="selectedName.displayName" ng-disabled="!isEditing && !isNewEntry" ng-required="required">
                        <p ng-hide="!errors.name" class="error">This field is required</p>
                      </div>
                      <div class="form-group">
                        <label for="data-col">Data Column:</label>
                        <input type="text" class="form-control" id="data-col" ng-
model="selectedName.dataCol" ng-disabled="!isEditing && !isNewEntry" ng-required="required">
                        <p ng-hide="!errors.col" class="error">This field is required</p>
                        <p ng-hide="!errors.duplicate" class="error">There is already an entry
for {{selectedName.dataCol}}. Please select a different column or edit the existing entry.</p>
                      </div>
                    </form>
                  </div>
                </div>
                  <div class="row">
                     <div class="form-group">

                      <div class="col-sm-4">
                        <button type="button" class="btn btn-default" ng-
click="deleteSelection()">Delete</button>
                      </div>
                      <div class="col-sm-4">
                        <button type="button" class="btn btn-primary pull-center" ng-
click="newEntry()">New</button>
                      </div>
                      <div class="col-sm-4">
                        <button type="button" class="btn btn-primary pull-right" ng-
click="editEntry()" ng-hide="isEditing || isNewEntry" ng-disabled="selectedName ==
null">Edit</button>

                        <button type="button" class="btn btn-success pull-right" ng-
click="saveEntry()" ng-hide="!isEditing && !isNewEntry">Save</button>
                      </div>
                    </div>

                  </div>
                </div>
              </div>
            </div>
        </div>
      </div>
    </div>
    <!--End Modal DIV-->

    <div id="charts" class="row list-group">
    </div>

    <div id="chart-container">
    </div>
```

```
</div>
```