Energy Management System

A modular solution for power monitoring and management for homes and small businesses

The Web Application – An investigation into how to create the best looking and most responsive web application for users to monitor and manage their power usage.

Date:
Tuesday, April 4th 2015
Author:
Jacob Lauzon (jfl4577@rit.edu)
Other Team Members:
Andrew Cope (ajc4630@rit.edu)
Donald MacIntyre (djm4912@rit.edu)
Ryan McLaughlin (rpm6651@rit.edu)

Table of Contents

Overview

The web application is an integral part of the Energy Management System. The key idea behind the Energy Management System is for the user to be able to see and analyze their energy usage and be able to control their usage from a central interface. The web application is what will allow the user to do both of these tasks. The application will offer a way for users to view their usage data in a clean and informative way. There will be many dynamic charts that will show various statistics about the energy consumption of individual outlet modules, groups of outlet modules, or the entire home. These charts will be able to be viewed over many different date ranges and update in real time, or as close to it as possible. In addition to the charts, there will be an easy-to-use interface that will provide the user the option to name, group, and control, via a toggle or a schedule, their outlet modules. The application will also provide security in the form of user accounts. The user will be able to set up multiple user accounts that will ensure that only certain people can access the application. Overall, the web application will be the way for the user to interact with and control their Energy Management System.

Risk Specification

Marketing Requirements

1. The system shall allow for control of individual outlets.
2. The system shall provide intuitive visual representations of usage data.
3. The system shall have low cost in comparison to competitive products.
4. The system shall be easy to install by a professional.
5. The system shall have an easy to use interface.
6. The system shall be of reasonable size in comparison to existing systems.
7. The system shall consume minimal power.

The above marketing requirements are the subset of the project's marketing requirements that pertain to the web application. Some of them pertain more than others but they all must be taken into consideration when designing the web application.

Engineering Specifications

Marketing requirements	Engineering Requirements	Justification
3	A. Production cost should not exceed $200 for the main unit and $30 for the outlet modules.	This is based upon analysis of a competitive market and current design requirements. The cost of the main unit will be affected by the embedded platform chosen to server the application.
4	B. Installation time should not exceed two hours in a typical single family residence.	Using a professional electrician, the main unit and outlets can be installed within this time frame. The web application installation/setup time will affect this.
1,2,5	C. A web interface or web application should allow the monitoring and	This will allow for a user to be able to manage the system and perform various tasks associated with the system. This specification is directly related to the web application

	management of the system.	
5	D. The user shall be able to understand the complete system functionality within an hour.	Analysis shows that an intuitive interface should require minimal time to operate. The web application is directly involved because the user will need to be able to learn the application quickly
7	E. The system shall have greater than 95% efficiency at maximum rated load.	To achieve energy savings and to avoid excessive heating of the wall units. The platform that is chosen for the application will affect this.
1,5	F. Wall units shall be identifiable.	This allows the system to know what information is coming from what wall unit and to provide individual control. The web application is how the user will be able to accomplish this.

Risk Investigation

There were two main concepts that needed to be considered when thinking about building the web application. The first was which embedded platform to use and the other was how to server (or host) the application and which, if any, framework to use to aid in the building of the application. Both of these elements were important and would affect the overall design and functionality of the application. When deciding which embedded platform to use and how to serve the application, each element was looked at separately and a separate decision was made for each.

Embedded Platform

When considering the embedded platform it was first decided that the project required a platform that was very easy to set up and get connected to the network, had enough computing power to serve a fluid application and host a database, was easy to develop on and make changes to, and was relatively small and low power. We decided to go with an embedded platform that runs a variation of Linux to allow for the ease of setup, computing power, and flexibility. This narrowed the search down to two competing products, the BeagleBone Black and the Raspberry Pi 2. Both of these platforms are very powerful and run a Linux distribution. Both also come with some trade-offs that we decided to be necessary. Because these devices are full computers they both are much larger than a typical microcontroller, both consume more power than a typical microcontroller, and both have a large amount of overhead to run the Linux distribution. Despite all this, we decided that one of these devices would be best for what was needed. The two choices were then examined closely to decide on the best one. Table 1 shows a comparison of the specifications of both of these devices.

Product	Raaspberry Pi 2 Model B+	BeagleBone Black Rev. C
Price	$35	$55
Size	3.370" x 2.224"	3.4" x 2.15"
Processor	Broadcom BCM2836	AM3358BZCZ100
Cores	4	1
Clock Speed	900 MHz	1 GHz
RAM	1 GB	512 MB
Onbaord Flash	None	4 GB
External Storage	microSD	microSD
Operating Voltage	5V	5V
Power	230 - 800 mA	210 - 460 mA
Digital GPIO	40	65
Analog Inputs	None	7
I2C	Yes	Yes
SPI	Yes	Yes
Ethernet	10/100 RJ45	10/100 RJ45
USB	4	1
Video Out	HDMI	micro HDMI

After looking at the comparison of the two units, the choice was clear. The Raspberry Pi 2 was chosen as the embedded platform. The most significant reason for the choice was the cost. Both devices perform similarly and have similar specifications in most areas so a $20 price difference made the Pi the obvious choice. The Pi also has double the RAM of the BeagleBone which made it an even better fit. The device will have to host a web application, a database, and the Linux distribution so more RAM will definitely be beneficial. The Pi does lack in Digital IO and analog inputs when compared to the BeagleBone but the main unit will not require many inputs so the Pi will perform just fine. Both devices have Ethernet, USB and some communication buses so, again, the cost of the Pi was the deciding factor. In addition to the specifications, both devices were tested for their ease of use. The simple task of loading the Linux distribution and installing some basic software on both devices was performed. From this subjective test, the Pi was easier to setup and work with than the BeagleBone. The BeagleBone's internal storage proved more difficult to work with than the Pi's external SD card storage.

Application Framework

The other important element to investigate for the web application was the application framework that is going to be used and how the web application will be served. There were many aspects that were looked into when choosing the desired method; the ease of development, the flexibility, the language, the overhead (in terms of memory usage), the documentation, and many other things. There are dozens and dozens of different combinations of frameworks and application servers that each offer a slightly different twist. There were way too many to compare them all so a few were picked to be compared. The three methods that were picked were picked based on the team's prior knowledge, the popularity of these setups, and some discussion with some senior developers. The three framework/server combinations that were chosen to be compared were a Java web application hosted

by Apache Tomcat, a Django application being served by Apache, and an implementation using HTML and JavaScript for the front-end and PHP/Python for the server-side database requests that would also be served by Apache. After choosing these three methods, the three main concerns were memory usage, ease of use and setup, and the team's familiarity with the language that the framework used.

To look at memory consumption a very rough test of memory usage was performed by installing Apache Tomcat and Django on a Windows machine and observing the memory usage. The memory was observed at the default configuration and with no applications actually being served so the results must be taken with a grain of salt. The HTML/ JavaScript method was not tested because of the very large amount of variation depending on which JavaScript libraries were used, how the Apache server was configured, how heavy the traffic to the website would be, and many other factors. It would be very difficult to even get an estimate of the memory usage, however, since an HTML/JavaScript solution would not have the overhead of a framework, theoretically the memory usage could be lower than the other implementations proposed. Table 2 shows the results from the memory test as well as some other components of each setup.

Table 1 Framework/Server Comparisons

Setup	Apache Tomcat	Django on Apache	HTML/JavaScript on Apache
Idle Memory Consumption	~150 MB	20-30 MB	?
Language	Java	Python	HTML/JavaScript/PHP/Python
Ease of setup (5 = easiest)	3	2	5

Table 2 provided some information about each of the framework/server configurations however everything in the table (besides language) were very rough quick looks into the framework. Despite this, the Java Web Application was not chosen to be used because of the memory consumption and the fact that Java was decided to be too powerful for what the application needed to do. While the Raspberry Pi 2 has enough RAM to support a Java application we decided that the overhead was not worth it because the application does not require a very advanced interface that could be achieved with Java. The decision between Django and HTML/JavaScript was made solely on the group's familiarity with the language. Since the memory usage will be fairly similar between these two solutions the decision was made by which language the team wanted to work with. All team members are more familiar with Python than HTNL/JavaScript so we decided to go with the Django Framework.

Risk Mitigation Design

The design that was chosen was to use the Django Python framework and serve it on an Apache Web Server on the Raspberry Pi 2. This design was chosen to be the best solution to meet the marketing requirements and engineering specifications while maintaining a certain level of simplicity for the developers. The platform and framework chosen will help to provide a simple and clean interface for the user, reduce the cost of the main unit, make it easier for the developers to develop the necessary application, and make the system easy to install. To take advantage of the chosen framework and server and simple yet powerful design needed to be created for the web application. The application needs to be clean and simple while still providing all the required functionality in an intuitive and easy-to-learn way.

The design will contain three main parts that will work together to achieve the overall goal. There will be the application component and the database component. The application will provide the user interface where the user can interact with the data and the system. The application will also provide a way to communicate with the database component and interface with the other hardware in the main unit. The database component will be where the measurement data is actually going to be stored and how it is going to be organized. Both of these components together will provide the full functionality of the web application. The design will focus on meeting the provided engineering specifications and mitigating the risk that these requirements will not be met. Table 3 shows the engineering specifications for the project and how the web application design has to help to achieve these specifications.

Table 2 Engineering Specifications for Web Application

Engineering Requirement	How Web Application Will Help
A. Production cost should not exceed $200 for the main unit and $30 for the outlet modules.	The cost of the main unit will be dominated by the embedded platform that was chosen. By choosing the Raspberry Pi 2 over the BeagleBone Black the cost was significantly decreased.
B. Installation time should not exceed two hours in a typical single family residence.	Part of the installation of the system will be setting up the web application. This involves getting the main unit connected to the network and setting up the application. By choosing an embedded platform that runs Linux and has Ethernet built-in the installation of the web application will be straight forward and leave more time for the rest of the application to be installed.
C. A web interface or web application should allow the monitoring and management of the system.	This specifications is directly tied to the web application and therefore the specification can be met simply by designing a strong web application.
D. The user shall be able to understand the complete system functionality within an hour.	This specification is also heavily tied with the web application. Most of the interaction with system will done through the web application. The design must be intuitive and simple so the user can learn how to use it and be able to perform all tasks quickly and easily.
E. The system shall have greater than 95% efficiency at maximum rated load.	The design of the web application will affect the efficiency. Choosing the Raspberry Pi 2 increases the power consumption of the main unit slightly but choosing to go with a light framework such as Django will help to reduce this impact.
F. Wall units shall be identifiable.	The web application is how this is going to be achieved. The user will be able to name and group outlet modules through the application.

The Application

The application will be constructed using the standard web application layer design where each part of the application is separated into layers that interact with each other. This reduces coupling

within the application and allows for easy maintainability and upgradability. Figure 1 shows an example of this layering from Microsoft.

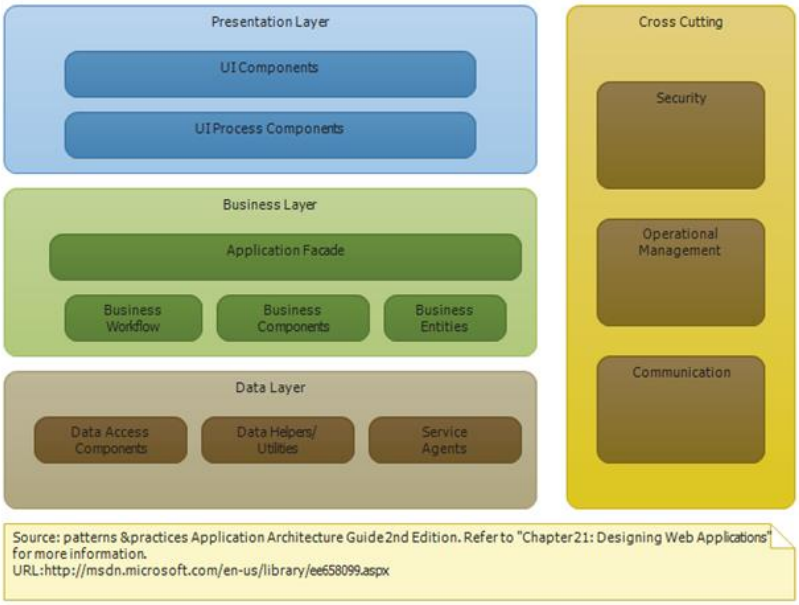

Figure 1 Web Application Layering

Figure 1 shows that there are three main layers and other components that interact with all layers. This is the typical design of a web application that is used by most in industry. The presentation layer (or UI layer) is where the user interface is implemented. This means that this is everything the user will see and interact with. So all the power consumption charts, the naming and grouping interface, and the scheduling or control interface will be implemented in this layer. The biasness layer (or Service layer) is where the UI will communicate to to get information or perform tasks the user requested. In this layer the users' requests from the UI layer will be processed and passed on to the Data layer. Also, in this layer, information from the Data layer will be processed and passed on to the UI layer to be displayed. In the data layer is where the actual requests and queries to the database are performed. There is usually a request from the Service layer that requires data to be fetched or modified. The Data layer handles parsing these requests into queries, running the queries against the database, and returning the results. The right-hand box shows some components that integrate with each layer such as security and communication. This design makes the application as loosely coupled as possible which is beneficial. A loosely coupled design is easier to maintain, upgrade, and allows for the development work to be divided easily. This design will also ensure the user interface is as simple as it can be because the UI will not be tied to the Data in any direct way.

The user interface is a crucial part of the design of the web application. The interface will be the sole interaction between the user and the system and therefore must be simple and intuitive. A very quick mock-up of the general idea behind the interface was done. Figures 2-5 show this mock interface.

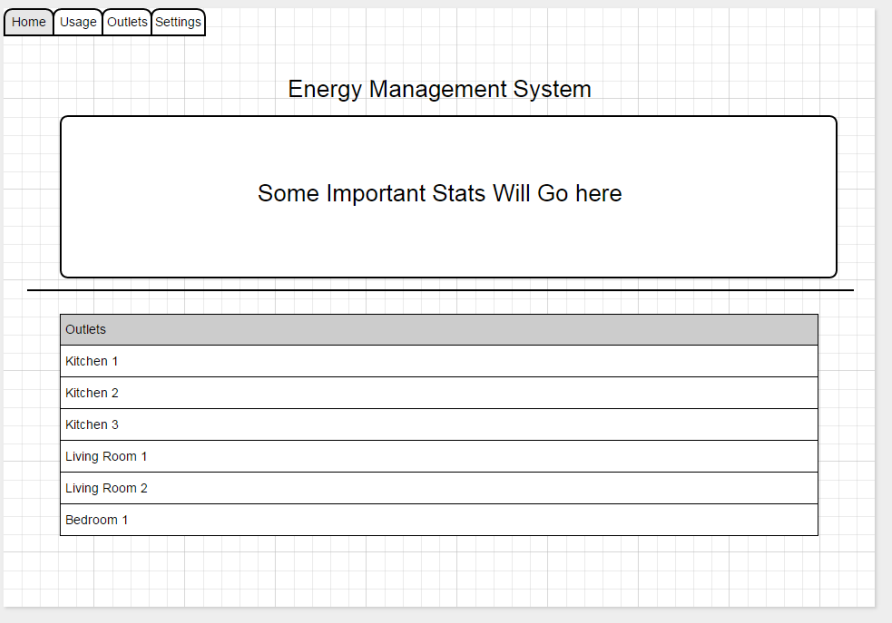

Figure 2 Home Tab of UI

Figure 2 shows the home tab of the application. This will be the screen that the user will see after they login to the application. Here, there will be some sort title on the top and then some basic statistics about that the user will want to see quickly. These statistics can include current power consumption, current energy use this month, current estimated cost this month, or other related statistics. The bottom half of this screen will be a list of all the outlet modules (or groups). From here you will be able to change name of the modules and toggle the state of the modules.

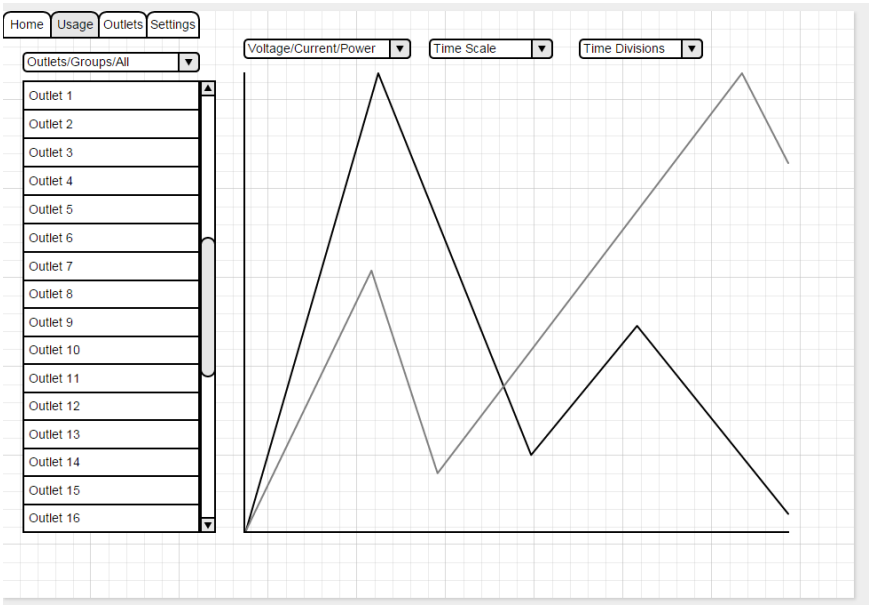

Figure 3 Usage Tab of UI

The next tab will be the usage tab (shown in Figure 3). This is the tab where the user will be able to view all the charts for their energy usage. The tab is just a general overview of what the screen could

look like. The user will have some way to select the outlet or group that they will want to view. There will then be one or more graphs where the user can select what they want to view (voltage, current, power, kWh, cost), the time scale of the graph (real time, day, week, month, year, etc.), and the time divisions of the graph that will be dependent on the time scale selected. There may be multiple graphs on this screen that can display multiple data at once or there is a possibility of displaying multiple lines on a single graph. This tab will be the most important tab and require the most development effort. The need to display so much information in many different ways will cause this tab to require an extensive coding effort.

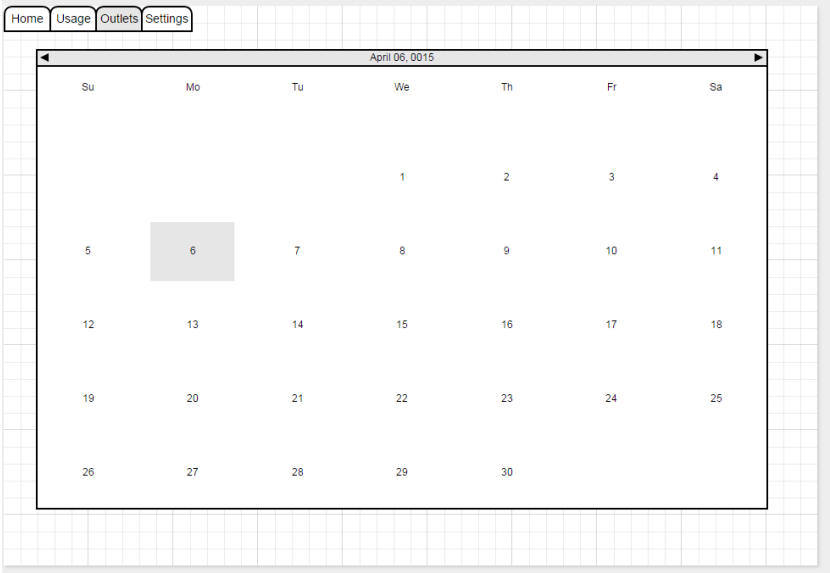

Figure 4 Outlet/Scheduling Tab of UI

The next tab (shown in Figure 4) will be the location where the user can setup the schedule for various modules and groups. The exact implementation of this functionality is not known yet but the vision is to have a calendar interface where the user can add events on any date and time, edit events, add repeated events, and other related tasks. The tab will act much like Google Calendar or a similar application.

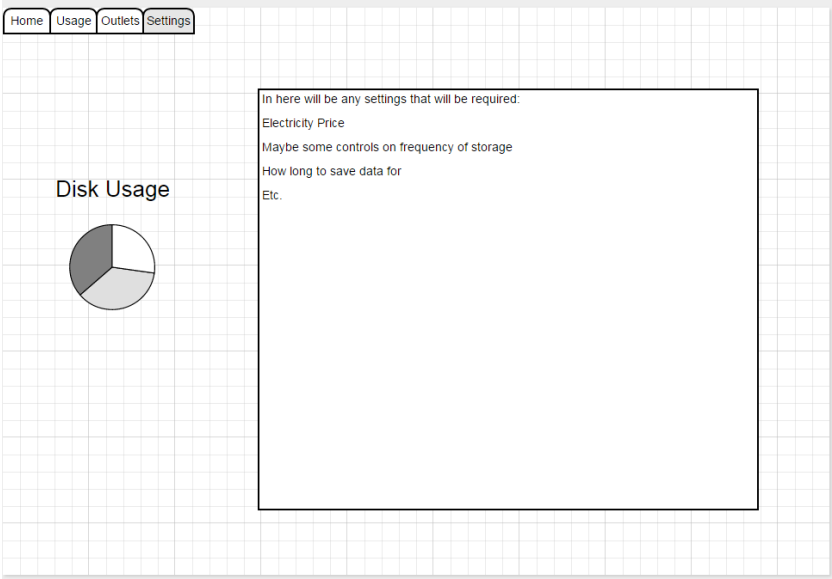

Figure 5 Settings Tab of UI

The final tab (shown in Figure 5) will be the setting tab. This is where the user will be able to configure various aspects of the application. There will also be a way to show the user how much disk space is remaining on their unit so they will know if their unit is close to running out of space. The exact settings are not known at this time but there will probably be settings to configure the cost of electricity for the user, the frequency at which the application stores data, and other server-related tasks.

The Database

The second part of the web application is the database. The actual type of database that is going to be used is being looked into by another team member but whatever the choice the application will need to interact with the database to save and fetch information. Database design is a difficult but important part of any web application. Creating the right table structure in a database is key to the performance of the application. The table structure can affect access times and how complicated the logic or queries must be within the application to access the database. Creating multiple smaller tables with the appropriate relationships can be better than one larger table but too many tables can lead to a more complicated design that requires more complicated queries. Fortunately, the database for the Energy Management System does not involve too many components and can be designed fairly simply. Figure 6 shows the entity-relationship diagram for the database structure that will be designed for the web application. This database will be where all of the data for the entire system will be stored.

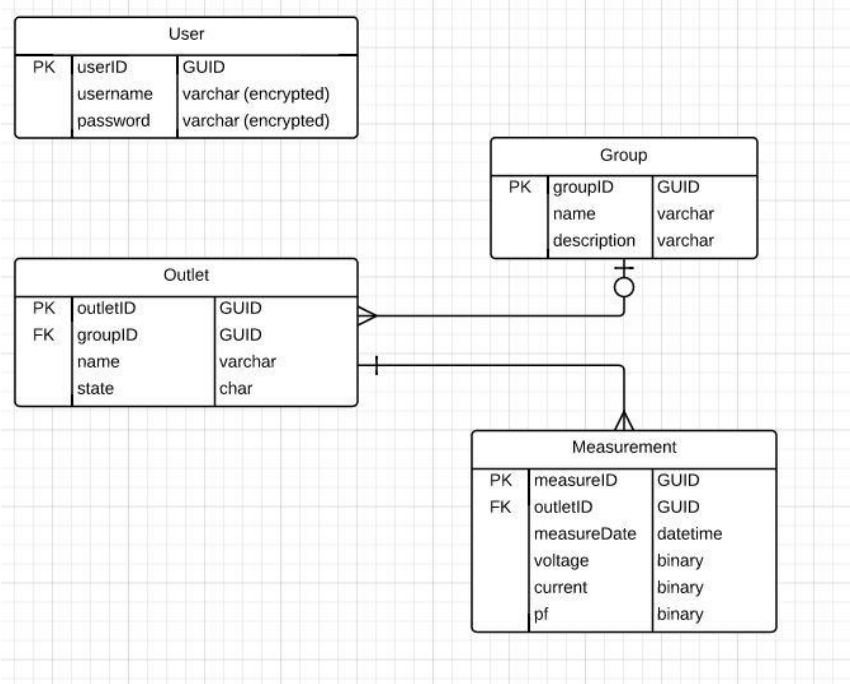

Figure 6 ER Diagram for Energy Management System Database

The database design shown in Figure 6 is a fairly simple design that should be a simple and effective structure for the application to use. There is one table to store the users of the application. This will be used to authenticate users that attempt to log into the application. The table contains a username and password field for each user, both of which will be stored as encrypted values for added security. The main design consists of an Outlet entity, a Group entity, and a Measurement entity. The Outlet entity represents an individual outlet module that has a name and an on/off state. There is a one-to-many relationship from the Outlet entity to the Measurement Entity. This is because there will be many measurements per outlet module. The Measurement entity is where the actual energy data will be stored. The date of the entry, a binary voltage, value, a binary current value, and a binary power factor (pf) value will be stored per measurement. There is also a many-to-one (or zero) relationship from Outlets to Groups. Since outlets can be grouped from within the interface to be controlled together, many outlets can belong to a group. A group simply has a name and a description. This database structure is very simple but provides for all the required functionality. This will allow for simplicity when creating queries from within the application and as fast of access times as possible.

Intellectual Property

The specific web application design for this project cannot really receive a patent so a patent search was not needed. In addition, all of the applications and frameworks that are being used are licensed under an open-source license and therefore are free for any and all use. This includes Django, Apache, and the Linux distribution loaded on the Raspberry Pi 2.

Parts List

Part	Cost	Team Cost	Availability
Raspberry Pi 2 Model B	$35 + shipping	$0 (Already Owned	In stock on Amazon
MicroSD Card (32 GB)	$16.80 (Sandisk)	$0 (Already Owned)	In stock on Amazon
USB Wireless Adapter	$9.99 (Edimax)	$0 (Already Owned)	In stock on Amazon

Testing Strategy

The application will obviously need to be extensively tested to ensure that there are no bugs and that the interface looks and performs as desired. Elaborate test plans have not yet been drawn up but will be in the future. These test plans will illustrate what needs to be tested and the steps to test each component. Unit tests may be set up to do some basic testing at the component level. Testing will initially be done with fake data but later move to real test data before release. A rotation will have to be set up where each team member is not testing their own code. In addition to internal testing, we will try and have a third party test the application for both usability and for bugs. A bug tacking system may be utilized if the need is great enough. In addition, version control software will definitely be used to allow for easy tracking of what has been done and the ability to rollback to an older version of the code.

The following a general list of the components that will need to be tested. This is a very basic level description of possible places for error. Most categories are very broad and will be broken down into specific test plans later.

- Login
    - Logging into the application with a default account
    - Setting up new user accounts
    - Ensuring there is no way to bypass the login screen
    - Testing encryption at the database level for user account information
- Outlet Module Naming and Grouping
    - The interface for naming the outlet modules
    - Names being saved correctly in the database
    - Names being displayed correctly throughout the application
    - Ability to create groups
    - Operations/Charts on groups
- Charts
    - All combinations of type, time scales, and time divisions will need to be verified to be showing the correct data
    - Ability to navigate through chart interface and select various charts
    - Ability to select different outlets and groups to see
    - Charts are showing accurate data
- Scheduling
    - The interface to create the schedule
    - The schedule being stored correctly
    - Schedule is actually being executed

- Settings
    - Accurate disk usage stats
    - Settings actually changing and saving
- Database
    - The database is storing the data correctly
    - Load testing on the database to ensure the Pi can handle all the queries
    - Accuracy of data being stored
    - Data is being queried correctly into web application
- Hardware
    - Application is actually able to communicate with the hardware API to turn them on and off

Uncertainties

There are few major uncertainties with this design. The first, and biggest, uncertainty is whether or not the Raspberry Pi 2 will be able to handle serving the web application and the database. The Django server was chosen to reduce memory usage however the usage is definitely not small. The application's usage along with the Linux distribution and hosting the database may be too much for the Pi. More extensive testing may need to be done to ensure that the memory usage will not be an issue. In addition to the memory usage, the processing power is a concern. There is some uncertainty as to whether the processor on the Pi will be able to handle all of the requests from the database, the hardware, and the application. If there is too much going on, the performance of the system could be affected.

The next uncertainty is if Django will have enough functionality to implement the desired application. The application seems simple but there will be lots of dynamic elements and many different charts that will involve very complicated logic. None of the team members have worked with Django before so we are unsure if the functionality we need is really there. If this becomes an issue a more powerful framework may need to be selected, like Java or more powerful JavaScript libraries. To go along with this uncertainty is also the time concern. This application is going to require an extensive coding effort. While the team is more than capable, the time constraint on the project is a concern because of all of the other components that are required for the Energy Management System. Some elements of the application may have to be dialed back to allow the application to be completed on time.

Appendix

BeagleBone Black
Product Page - http://beagleboard.org/BLACK
Processor Data Sheet - http://www.ti.com/lit/ds/symlink/am3358.pdf
Adafruit Page - http://www.adafruit.com/product/1876

Raspberry Pi 2 Model B
Product Page - http://www.raspberrypi.org/products/raspberry-pi-2-model-b/
Datasheet - http://www.adafruit.com/pdfs/raspberrypi2modelb.pdf

Django
Homepage - https://www.djangoproject.com/
Install Guide - https://docs.djangoproject.com/en/1.8/intro/install/
Django on Pi - http://www.hackedexistence.com/project/raspi/django-on-raspberry-pi.html
Django on Pi - http://www.hackedexistence.com/project/raspi/django-on-raspberry-pi.html

Apache Web Server
Homepage - http://httpd.apache.org/
Apache on Pi - http://www.raspberrypi.org/documentation/remote-access/web-server/apache.md