POW-R Low Level Design Document

Grace De Geus
gdegeus@vtc.edu

Charles Hathaway
chathaway2@vtc.edu

Nate Pickett
npickett@vtc.edu

Niloc Quimby
nquimby@vtc.edu

Forest Immel
fimmel@vtc.edu

November 29, 2012

Table 1: Revision History

Date	Version	Description	Author
11/09/12	<1.0>	Original version of this document	
11/13/12	<1.1>	Added a bibliography	chathaway
11/19/12	<1.2>	Added and formatted all use cases	gdegeus
11/20/12	<1.3>	Added a script to generate tex from our codes, and found a way to include those in the document.	chathaway
11/28/12	<1.4>	Added a Revision History table.	gdegeus
11/29/12	<1.5>	Added Non-functional Requirements section, added glossary section, re-arranged document layout, updated Introduction, index and Acronyms section.	gdegeus

Contents

List of Figures

List of Tables

1 Introduction

1.1 Purpose of this Document

This document describes the low level design of all components of the Power Outlet Wireless Reporter (POW-R) system. The purpose of the Design Document is to provide a description of the design of a system fully enough to allow for software development to proceed with an understanding of what is to be built and how it is expected to built. This document provides information necessary to provide description of the details for the software and system to be built.

1.2 Scope

This Design Document is for a base level system which will work as a proof of concept for the use of building a system the provides a base level of functionality to show feasibility for large scale production use. This Design is focused on the base level system and critical parts of the system.

2 | Acronyms and Definitions

2.1 Acronyms

A Amps

AJAX Asynchronous JavaScript and XML

API Application programming interface

CSS Cascading Style Sheets

GPIO General Purpose Input Output

HTML HyperText Markup Language

HTTP Hypertext Transfer Protocol

IEEE Institute of Electrical and Electronics Engineers

IP Internet Protocol

JSON JavaScript Object Notation

LCD Liquid Crystal Display

M Meters

NEMA National Electrical Manufacturers Association

PIP Python package installer

POST Request Method

POW-R Power Outlet Wireless Reporter

REST REpresentational State Transfer

SATA Serial ATA

SQL Structured Query Language

SSD Solid State Srive

TLS Transport Layer Security

UI User Interface

URL Uniform Resource Locator

USB Universal Serial Bus

VDC Volts DC (Direct Current)

W Watts

XML Extensible Markup Language

XSS Cross-site scripting

YAML YAML Ain't Markup Language

2.2 Definitions

Amperes (Amps) The SI base unit of electric current, equal to a flow of one coulomb per second.

API An application programming interface (API) is a specification intended to be used as an interface by software components to communicate with each other.

AJAX AJAX is a way of executing asynchronous calls to the server by using Javascript. The benefit of this is that we don't need to refresh the page, or send extra data over the network.

HTTP The Hypertext Transfer Protocol (HTTP) is an application protocol for distributed, collaborative, hypermedia information systems. HTTP is the foundation of data communication for the World Wide Web.

JSON Javsascript Object Notation. JSON is a way of representing objects in a hash-table that is easy to use in the Javascript scripting language. This is basically a hashmap, and can be looped through using for-each loops, for loops, and while loops.

REST REpresentational State Transfer is a software architecture that makes use of the existing HTTP protocol to implement a well defined, practical, and efficient means of representing resources on the web. REST uses URL's to denote different resources, with the query string being used to filter the resources. Although some believe that using a "clean url" instead of the query string is

proper REST, for our purposes there is no reason to have especially "human-readable" URL's that result from using clean URL's. In short, this is how we would denote a device in the REST API:

/api/raw/device/5

/api/raw is simple the entry path to the API. The next part, /device, is the table we are accessing. The last part, /5, is the primary key of the object we are looking at.

Another way to represent the same device, using the query string, would be:

/api/raw/device/?id=5

If you simple open these URL's in a web browser, you will be initiating a GET request. A GET request returns data already on the server. The verbs in the HTTP vocabulary are:

- GET - Get's information about the device
- POST - Creates a new entry, if directed at the resource collection
- PUT - Updates the specified resource with the given information
- DELETE - Deletes the specified resource or everything in the specified collection

Combing these verbs with the search query and the URL, we can effectively create an uniform API for accessing the database over the web. In addition, we can control access and authorization via standard methods handled by the web.

Volt (Volts) The SI unit of electromotive force, the difference of potential that would carry one ampere of current against one ohm resistance.

3 | Glossary

Glossary is unused in current document due to Section 2 Acronyms and Definitions providing terms and definitions for internal use of the document.

4 | Use Cases

The following tasks are broken down into several simple steps. Although they do not account for things like "find the button to click", it is clear when a user is expected to click a button.

The "Actors" section describes the things that will play a role in this scenario. These can include:

- Users
- The display
- The Satellite

The "Assumptions" section describes assumptions that are made for this scenario. These include things such as "User is logged in", or "Several Satellites are connected".

The "Procedure" describes the procedure that is taken to by the user to accomplish the goal.

The "Expected Outcome" is the ultimate goal. It is what the user expects to happen after they execute the procedure. These can be visual outputs, stored data, or even changes to the way the system behaves.

Listed below are all following Use Cases in order.

- Logging In
- Logging Out
- View Monthly Power Consumption
- View Largest Power Consumer
- View Power Consumption Over Time

- Adjust Power Cost
- Add User
- Delete User
- Add Device
- Rename Device
- Disable Device
- Re-enable Device
- Remap Device
- Adding a Satellite
- Removing a Satellite

4.1 Logging In

In this scenario the user logs into the system.

Actors

- User
- System

Assumptions

The user has valid login information
The user has the login page open in their browser

Procedure

The system prompts the user for a username and password
The user enters their username and password
The user clicks a "OK" or "Submit" button
The system logs the user in

Expected Outcome

The user is now logged in.

4.2 Logging Out

In this scenario the user logs out of the system.

Actors

- User
- System

Assumptions

The user is logged in
The user is on any page

Procedure

The user clicks the logout button
The system logs the user out
The display informs the user that they have been logged out
The system returns the user to the login page

Expected Outcome

The user is now logged out.

4.3 View Monthly Power Consumption

In this scenario the user is intending to view the power consumption of the entire house for a series of specified consecutive months.

Actors

- User
- Display
- System

Assumptions

The user is logged into the display site
The user is at the home page
The system has logged data from at least one month ago

Procedure

The user clicks the "View Data" button in the navigation menu
The display opens a drop-down menu with several more links

- One of which is "View Device Power Consumption"

The user selects "View Device Power Consumption"
The user specifies the date range that will viewed
The user clicks a "OK" or "Submit" button
The display shows the user a table containing:

- The name of the month
- The total power consumed by the entire system for that month

The site displays the sum of the power consumption for each Satellite on each day of the month

Expected Outcome

The site will display the power consumption from the specified month or a warning if the data does not exist.

4.4 View Largest Power Consumer

This use case describes the process that a user would employ to display a list of the largest power consumers being recorded by the system.

Actors

- User
- Display

Assumptions

The user is logged in and on the home page
There are several devices associated with the server

Procedure

The user clicks the "View Data" button from the navigation menu
The display opens a drop-down menu with several more links

- One of which is "Device Power Consumption"

The user selects "Device Power Consumption"
The display shows the user a table containing:

- The name of each device
- The power consumed by each device
- The device's power consumption per hour
- Sorted alphabetically by name

The user clicks on the "Power consumed" column to sort it from greatest to least
The display sorts the list from greatest power consumption to least

Expected Outcome

The user sees that the first item in the list is using the most power.
The user sees the next few largest power-consuming devices.

4.5 View Power Consumption Over Time

This use case describes the process that a user would employ to display graphs indicating the power consumption of devices over a specified time range.

Actors

- User
- Display

Assumptions

The user is logged in and on the home page
There are several devices associated with the server
The server has several months worth of data

Procedure

The user selected the "View Data" button from the navigation menu
The display shows the user a list of charts they can view

- One of which is "View Power Use over Time"

The display shows a page that prompts the user to select:

- What devices or device groups they want to compare
  - Defaults the group "All devices"
- What time period they would like to compare them over
  - Defaults to the last six months

The user supplies the requested information The user clicks "Display" The display generates one chart for each device/device group over the specified range Displays them in the order the user selected them

Expected Outcome

The user sees the power consumption trends of the selected devices over the selected time period

4.6 Adjust Power Cost

This use case describes the process that a user would employ to change the cost-per-kilowatt-hour that the system uses to guesstimate the monthly power bill.

Actors

- User
- Display

Assumptions

The user is logged in and on the home page
The user has permission to change the kilowatthr cost

Procedure

The user clicks the "Settings" link from the navigation menu
The display presents the user with the settings panel, which includes a link to the "Adjust Power Cost" page
The user clicks the "Adjust Power Cost" link
The display presents the user with a page consisting of two columns of text boxes. The left column indicates the range for the cost, which is specified in the right column

- The goal is to allow the user to say "My first 500 kW/hrs cost $0.08, my next 1000 kW/hrs cost $0.10, and everything after that costs $0.12 per kWhr"

The user adjusts the cost by changing the ranges or changing the costs
The user clicks a "Submit" button
The display shows the changed power costs

Expected Outcome

The display applies the changed power costs to all new calculations. Previous calculations shall remain unchanged, with assumption that costs just increased.

4.7 Add User

This use case documents the procedure that a user would use to add a new user.

Actors

- User
- Display

Assumptions

The user is logged in and on the home page
The user has permission to modify user permissions

Procedure

The user clicks the "Settings" link from the navigation menu
The display presents the user with the settings panel, which includes a link to the "User Management" page
The user clicks the "User Management" link
The display shows the user the "User Management" page

- This includes a button to add a user

The user clicks "Add user"
The display presents the user with a page that contains the necessary fields to create a new user:

- First Name
- Last Name
- Username
- Password
- A check-box requiring the user to change their password at login
- A multi-select to put them into groups
- A list of permissions to give the user

The user supplies the requested information
The user clicks a "Submit" button
The display informs the user that the user has been added
The display shows the user the "User Management" page

Expected Outcome

A new user was added with the requested username, password, and permissions.

4.8 Delete User

This use case documents the procedure that a user would use to delete a user.

Actors

- User
- Display

Assumptions

The user is logged in and on the home page
The user has permission to modify user permissions

Procedure

The user clicks the "Settings" link from the navigation menu
The display presents the user with the settings panel, which includes a link to the "User Management" page
The user clicks the "User Management" link
The display shows the user the "User Management" page

- This includes a button to delete a user

The display presents the user with a list of users with check-boxes next to their usernames
The user then clicks on the check-box next to appropriate username
The user clicks "Delete user"
The display shows a dialog box asking to confirm the delete
The user clicks on "Apply action"
The display informs the user that the user has been deleted
The display shows the user the "User Management" page

Expected Outcome

A user was deleted from the system by another authorized user.

4.9 Add Device

This use case documents the procedure that a user would use to add a new device to the system.

Actors

- User
- Display
- Device

Assumptions

The user is logged in and on the home page
The user has permission to modify devices

Procedure

The user clicks the "Settings" link from the navigation menu
The display presents the user with the settings panel, which includes a link to the "Device Management" page
The user clicks the "Device Management" link
The display shows the user the "Device Management" page

- This includes an "Add device" button

The user clicks the "Add device" button
The display prompts the user for information about the device, including:

- Name
  - Optional: room, location in room, owner. These will be used to guide the user in coming up with a unique name
- What Satellite it is plugged into
  - Optional, may be not be plugged in

The user fills in the requested information
The user clicks an "OK" or "Submit" button
The display informs the user that the device has been added
The display shows the user the "Device Management" page

Expected Outcome

The display has the device available for history and mapping to Satellites.

4.10 Rename Device

This use case documents the procedure that a user would use to rename a device associated with an outlet.

Actors

- User
- Display
- Device

Assumptions

The user is logged in and on the home page
The user has permission to modify outlet-device mappings

Procedure

The user clicks the "Settings" link from the navigation menu
The display presents the user with the settings panel, which includes a link to the "Device Management" page
The user clicks the "Device Management" link
The display shows the user the "Device Management" page

- This includes a list of all existing devices

The user clicks on the row of the device they want to edit
The display presents a page with a form that allows the user to edit the name of the device
The user changes the name of the device
The user clicks "Submit" or "Save"
The display informs the user that the device has been renamed
The display shows the user the "User Management" page

Expected Outcome

The device is now know by a new name.

4.11 Disable Device

This use case documents the procedure that a user would use to disable a device.

Actors

- User
- Display
- Device

Assumptions

The user is logged in and on the home page
The user has permission to modify devices

Procedure

The user clicks the "Settings" link from the navigation menu
The display presents the user with the settings panel, which includes a link to the "Device Management" page
The user clicks the "Device Management" link

- This includes a drop down with an action to disable a device
- This includes a table listing all the devices, with check-boxes next to each row

The user selects the 'Disable Device" action
The user then clicks on the check-boxes next to appropriate devices
The user clicks an "OK" or "Submit" or "Apply" button
The display shows a dialog box asking to confirm applying the action
The user clicks on "Apply action"
The display informs the user that the device has been disabled
The display shows the user the "Device Management" page

Expected Outcome

A device was disabled and will no longer appear in any menus.

4.12 Re-enable Device

This use case documents the procedure that a user would use to re-enable a device.

Actors

- User
- Display
- Device

Assumptions

The user is logged in and on the home page
The user has permission to modify devices

Procedure

The user clicks the "Settings" link from the navigation menu
The display presents the user with the settings panel, which includes a link to the "Device Management" page
The user clicks the "Device Management" link
The display shows the user the "Device Management" page

- This includes a link to a "Disabled device Management" page

The user clicks on the "Disabled device Management" link
The display shows the user the "Disabled device Management" page

- This includes a drop down with an action to re-enable a device
- This includes a table listing all the devices, with check-boxes next to each row

The user selects the "Re-enable Device" action
The user then clicks on the check-boxes next to appropriate devices
The user clicks an "OK" or "Submit" or "Apply" button
The display shows a dialog box asking to confirm applying the action
The user clicks on "Apply action"
The display informs the user that the device has been re-enabled
The display shows the user the "Device Management" page

Expected Outcome

A disabled device was re-enabled and will appear in all menus.

4.13 Remap Device

This use case documents the procedure that a user would use to associate a device with a Satellite.

Actors

- User
- Display
- Satellite

Assumptions

The user is logged in and on the home page
The user has permission to modify Satellites and modify devices
The device has been created
The Satellite has been associated with the display

Procedure

The user clicks the "Settings" link from the navigation menu
The display presents the user with the settings panel, which includes a link to the "Device Management" page
The user clicks the " Device Management" link
The display shows the user the " Device Management" page

- This includes a table listing all the Satellites

The user clicks on the row of the device they wish to associate with a Satellite
The display shows the user a page that contains information about the selected device

- Including a drop down that allows the user to select the associated Satellite

The user selects the Satellite the device is plugged into from the drop down
The user clicks a "OK" or "Submit" button
The display informs the user that device has been updated
The display shows the user the "Device Management" page

Expected Outcome

The device has been associated with a new Satellite, and all data from that Satellite will be attributed to the specified device.

4.14 Adding a Satellite

This use case documents the procedure that a user would use to associate a new Satellite with the display.

Actors

- User
- Display
- Satellite

Assumptions

The user is logged in and on the home page
The user has permission to modify Satellites
The user has plugged the Satellite into the wall
Everything goes according to plan

Procedure

The user clicks the "Settings" link from the navigation menu
The display presents the user with the settings panel, which includes a link to the "Satellite Management" page
The user clicks the "Satellite Management" link
The display shows the user the "Satellite Management" page

- This includes an "Add Satellite" button

The user clicks the "Add Satellite" button
The display prompts the user to press the "Connect" button on the Satellite and then click "OK"
The user presses the "Connect" button on the Satellite
The display informs the user that it's searching
The display then displays the serial number (xxx-xxx) of the connected Satellite
The user confirms that this is the correct Satellite
The display shall informs the user that a new Satellite was connected
The display shows the user the "Satellite Management" page

Expected Outcome

A new Satellite was added to the display.

4.15 Removing a Satellite

This use case documents the procedure that a user would use to remove a Satellite with the display.

Actors

- User
- Display
- Satellite

Assumptions

The user is logged in and on the home page
The user has permission to modify Satellites
The Satellites being removed do not have any devices associated with them

Procedure

The user clicks the "Settings" link from the navigation menu
The display presents the user with the settings panel, which includes a link to the "Satellite Management" page
The user clicks the "Satellite Management" link
The display shows the user the "Satellite Management" page

- This includes a drop down with an action to remove Satellites
- This includes a table listing all the Satellites, with check-boxes next to each row

The user clicks on the appropriate check-boxes next to the Satellites
The user selects the "Remove Satellite" action from the drop down menu
The user clicks "Apply" or "OK"
The display prompts user to confirm the removal
The user clicks "Submit" or "OK"
The display informs the user that the Satellites were removed
The display shows the user the "Satellite Management" page

Expected Outcome

The specified Satellites have been removed from the display. No historical data should have been lost due to this procedure.

5 | Hardware Design Overview

5.1 Description

The POW-R project is comprised of two main hardware components, the Server and the Satellites. The Server refers to the physical hardware from which the Display shall be served. It also acts as the data center for all Satellites associated to it, collecting data and storing it in a hard drive. The Satellites will talk over ZigBee specification to the one Zigbee module connected to the Server. That one Zigbee module can talk to the Server over Universal Serial Bus (USB).

Figure 5.1 shows the layout of the hardware system.

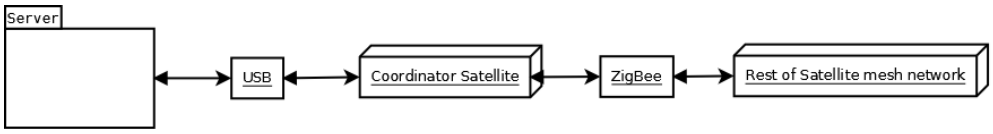


Figure 5.1: System Overview

5.2 The Server

5.2.1 PC Hardware

Mainboard

The Server's main hardware component is the mainboard, a SYS9400-ECX Developer-Ready Reference Platform. It's a small form-factor, low-power machine with the following specs:

- 1.6 GHz Intel Atom E6XX Series Processor
- 1 GB DDR2 RAM

- Roughly 6" by 4"
- Various connection interfaces:
  - 2x Serial ATA (SATA) ports
  - Header for SATA power
  - Ethernet port
  - 5x USB 2.0 ports
  - General Purpose Input Output (GPIO) pin header

Potential Problems

If for whatever reason using this mainboard falls through, it should be noted that the requirements for the Server hardware concern not just specifications, but interfaces as well. In particular, this project requires at least Ethernet, 2 USB ports, a SATA port, and accessible GPIO pins.

Storage

The Server's mainboard is connected via SATA to a 40GB Solid State Srive (SSD).

Power Supply

An adapter rated for 12Volts DC (Direct Current) (VDC) @ 3Amps (A) is used to connect the Server's board to mains electricity.

5.2.2 Add-Ons

The Server provides for the computational needs of the POW-R project; more hardware shall be added before the utility needs of the project are met.

5.2.3 Server Hardware IO

IP Display

The IP of the Server shall be displayed somewhere on it's casing. The user will enter this IP into their browser to access the Display.

This will be achieved by connecting an Arduino via USB to the Server, and housing it inside the Server's casing. The Arduino will be connected to an

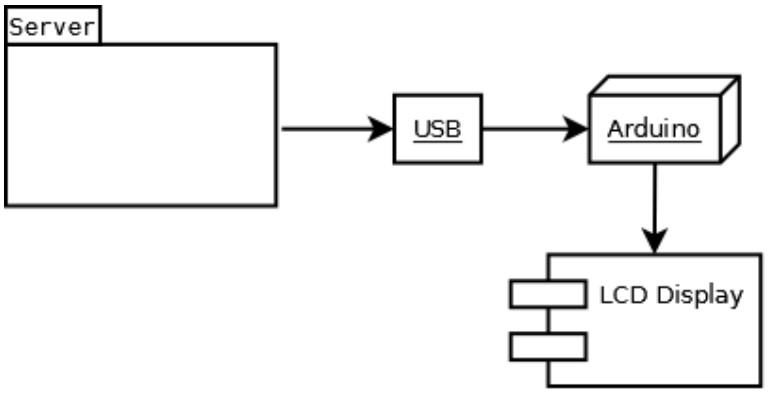


Figure 5.2: IP Display Diagram

Liquid Crystal Display (LCD) display which will output the network IP of the Server. Figure 5.2 shows the interaction between Server and Arduino.

The IP address of the system can be obtained from the operating system, and sent to the Arduino one byte at a time. This works well, since the Arduino connects over serial and thusly takes a byte at a time.

Power Switch

A standard rocker switch will be added to the case to provide the user with a way to turn the Server on and off. The style of the switch will clearly signify "On" or "Off".

Factory Reset[??]

This button should intentionally be placed somewhere inconvenient: Sunken into the case far enough that you need a skinny rod (such as a paperclip) to push it, and in a spot that chaotic forces (children, mean people, God's divine will) will not notice it.

5.2.4 Server Hull

The Server needs a protective casing for several reasons. On the physical level, a durable casing will protect the hardware. The casing also serves to render unnecessary ports inaccessible to users. Lastly, the casing shall meet client expectations of what a server looks like.

As far as prototyping goes, the casing can be as simple as a folded piece of aluminum with holes cut in it to fit the IP display, the buttons, and any ports

that must be exposed. As an end-game product, the casing would probably be more professional and streamlined.

5.2.5 Potential Problems

As of right now, the Server has not been explored, and as with the rest of this project, is unfamiliar territory.

5.3 The Satellites

The Satellites are made up of two sets of hardware: Instruments for measuring current and voltage, and radios for communicating that data to the Server. The end product shall have a hard casing around it, National Electrical Manufacturers Association (NEMA) 5-15 sockets on one side, and prongs for a NEMA 5-15 male end on the other. However, it should be noted that prototyping will most likely involve all of the hardware being spread out on breadboards.

5.3.1 Measurement Hardware

Current and voltage running through the mains outlet to the Satellite's associated device must be measured and sent to the Server.

5.3.2 Communications Hardware

For the communication between the Server and the Satellites, we will be using the Xbee Series 2 Radio Modules.

XBee Series 2 Radio Modules

The XBee Series 2 Radio Module is configured to be running the ZigBee mesh firmware. We chose this module because of it's low current draw. When configured as an end device, it draws at most 40 mA. In addition to it's low power draw, it also has a considerable indoor transmit range of 133 ft.

Interface Board

The Digi Interface board is the interface board that we will be using to connect to the Xbee Series 2 Radio Modules. We chose this due to it's simplicity in communicating with the Modules.

5.3.3 ZigBee Standard

ZigBee is a standard that uses Institute of Electrical and Electronics Engineers (IEEE) 802.15.4 wireless protocol as it's foundation [2]. It's particularly useful for setting up ad-hoc mesh networks. Much like how any Bluetooth devices can talk to each other, any ZigBee devices nearby can talk to each other (so long as they have the proper permissions).

Why ZigBee?

The advantage of using ZigBee is that it lines up with the requirements of the POW-R project. It's a specification targeted for low-power, low-data applications that need to be secure.

Another advantage of using ZigBee is that XBee radios, which also fit the requirements of the POW-R project, are ZigBee-compliant, so the forming of a mesh network can happen quickly and easily.

Device Types

The nodes of a ZigBee network are obviously ZigBee devices, but from the network's perspective, there are three distinct types of ZigBee devices:

- Coordinator
- Router
- End Device

The coordinator runs the network, keeps it healthy, makes sure everybody is talking when they should be, and overall, just maintains the Mesh. It is also the device that is typically plugged into a computer via USB or some other serial interface, meaning that this is the node you want to send data to. There cannot be a ZigBee network without exactly one coordinator.

Routers are fully functional ZigBee modules, capable of taking care of themselves and routing messages across the Mesh. Routers are capable of being "parent" devices.

End devices are low-power ZigBee devices that don't do any routing. They can only talk to one other device, which is referred to as it's "parent" device. Only a router or a coordinator may be a parent device.

In the POW-R project, the coordinator will be attached to the Server so that it may also act as a data bridge between the Server and the Satellites. The rest of the Satellites are routers; There will be no end devices in the POW-R's mesh network.

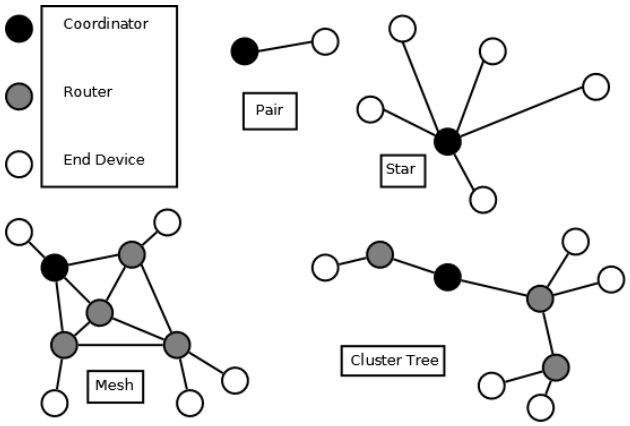


Figure 5.3: ZigBee Network Topologies

Network Topology

Zigbee supports four network topologies, shown in Figure 5.3. A ZigBee network is defined by two rules:

- A ZigBee network must have two or more ZigBee devices
- One of those devices *must* be a coordinator

As mentioned before, the POW-R project will use the Mesh configuration, but without end devices. This means that the Mesh will consist entirely of routers and one coordinator.

5.3.4 Potential Problems

XBee Series 2 radios and their interface boards have as of yet been unexplored. This means that as with the rest of this project, there will be about as much learning as there will be doing. To counteract this somewhat, the engineers in charge of weaving the Mesh have learning materials that are perfect for this project.

6 | Software Design Overview

6.1 Description

The POW-R project uses various pieces of software to control the Satellites, provide an API for displays, and generate pages for the primary display. The POW-R project will consist of the following pieces of software:

- The microprocessor code
- The server code to communicate with the Satellites
- The server code to present the API
- The client code to render the display for the user

The microprocessor code is documented in the hardware section.

6.1.1 The Display Backend

The general plan for the backend is to write modules for a Django [5] web application. These modules will either be running in the background and reporting the data given to the server from the Satellites to the database through the REST API functions, or responding to user requests sent through the REST API.

The logic for using Django as the base, and write modules around Django, is one of necessity. We do not have enough time or manpower to make everything we need from scratch. Therefore, we will be using several open source utilities to assist us.

Django will give us an easy-to-use database interface. It will deal with sanitizing all Structured Query Language (SQL) queries, and make sure all output is escaped (i.e. no Cross-site scripting (XSS)). Using Django gives us almost all of the security benefits listed in our non-functional requirements.

To make our REST API, we will utilize an open-source Django library called tastypie. Tastypie gives us both model resources (that is, API elements that are basically the database tables) and the ability to create custom resources that may, or may not, map to the database.

Another advantage of Django is that it comes with a very extensive, expandable, and adaptable authentication system. This will help accomplish several optional and required features almost instantly.

6.1.2 The Display Frontend

The general plan for the frontend is to serve up some fairly static pages, and use extensive amounts of Javascript to fill in the content and animate things. The frontend will communicate with the backend through the REST API. This will be fairly simple since the backend API will support multiple formats, including JavaScript Object Notation (JSON).

To assist us on the frontend, we will be using several open source libraries. They will all serve a very clear function.

- Backbone.js - A library to provide basic functions dealing with fetching and syncing data with the server [4]
  - Backbone.js will deal with tracking the "model" portion of our Javascript code
  - Backbone.js will be made compatible with our REST API by using a small extension called backbone-tastypie [7]
- iCanHaz - A library that provides advanced templating functionality This is client-side templating, and relies on several "fake" HyperText Markup Language (HTML) tags [1]
- RequireJS - A library that deals with tracking and loading assets asynchronously This includes loading libraries that we will be using [6]
- jQuery - jQuery will be used for both parts of the front end and for the AJAX support [3]
  - We will use jqplot for dealing with graphs
  - We will use the jquery UI to deal with most of our widgets and such

6.2 Program Flow

This program will have three main "threads". One thread would be responsible for listening to messages from the Satellites, and another would be responsible

for listening and responding to requests on the API. The last thread would be responsible for serving up static files for the main display.

The Satellite thread will be responsible for taking the logging information provided by the Satellites and storing it in the database. Each Satellite will report it's information to the server once every second. Therefore, all this thread needs to do is react to incoming connections.

The API thread would provide the Representation State Transfer API (REST) that will be used by all future displays to get data in a standardized, well defined way. This thread will NOT be serving up images or page templates. This is done to make best use of the caching functionality included in the HTTP server software that will be used.

The last thread will be running a "dumb" web page that servers up static HTML and Javascript files. This thread will also retrieve images and Cascading Style Sheets (CSS) files from the server.

6.3 Data Flow

This application will use a SQL database to store all data. Each module will interact with the REST API, which interacts with the database as needed.

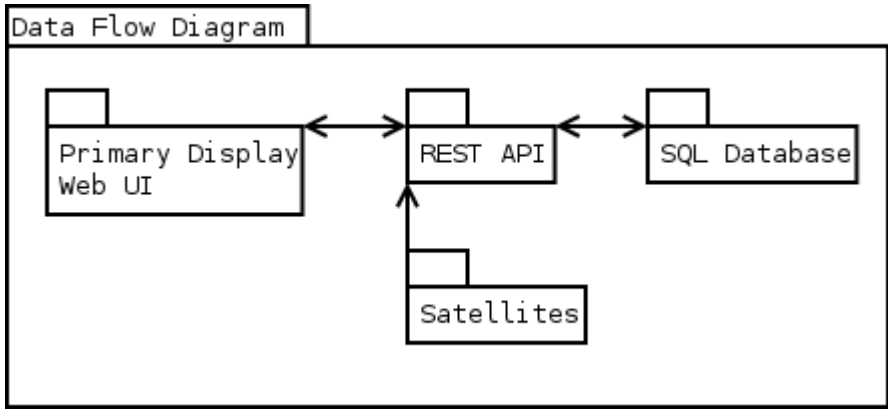


Figure 6.1: Data Flow Diagram

Figure 6.1 shows how various components will be interacting. Notice that the database is not directly accessed by any components, but goes through the REST API instead. This allows us to provide a standard interface that should be more-or-less accessible from any programming language or system. This will also allow us to use some well-known security features, such as Transport Layer Security (TLS).

6.4 Description of Software Modules

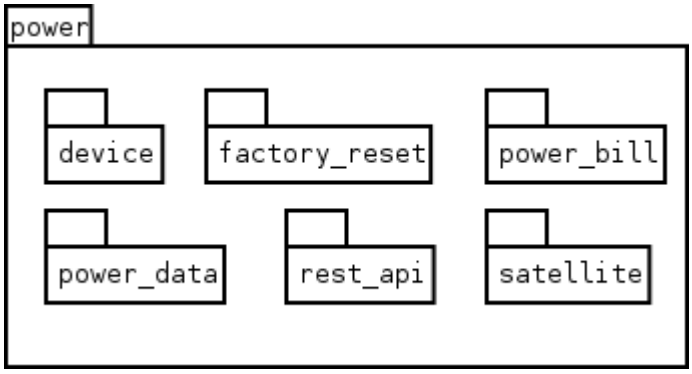


Figure 6.2: Listing of software modules

Figure 6.2 contains all the Django modules that will be created inside the actual Django "website" itself. This is basically the folder hierarchy of our web application source code. The "power" folder consists of folders for each module (not including the various open-source modules we are using, since those are being controlled via Python package installer (PIP)). Each of these contains a `models.py` file, which specifies what the database for that module should look like, an `__init__.py` file which will register the module with the REST API, and whatever other files that module needs.

6.5 Potential Problems

Using open-source libraries requires us to learn them. This is a big problem because of the time constraints on this project, and will require us to spend almost as much time learning as we will be in development (possibly more). This can lead to frustration, especially in light of the fact that we will need to learn the languages in addition to the libraries.

However, using well developed and designed libraries allows us to skip over a good part of the development and planning. We will be able to use extensive API's without taking responsibility for maintaining the software. This will allow us to focus on our product instead of the all the small things that make it up.

Another potential problem with using the suggested architecture will be clients that don't like using Javascript in their browser. Using heavy javascript allows us to offset a good bit of the load from our low-powered server to the clients computers, which will have a good deal more power. If we wanted to support clients that don't like using javascript (E.g. security nuts), we would be looking

at adding a significant load to our server (text-processing can consume a good bit of resources, especially string manipulation). Although we may be able to do it, for the first version we will simply add "Javascript support" as a requirement for the clients.

6.6 The Backend

6.6.1 Authentication

Description

This module describes the architecture of how the backend will manage features related to users and groups (authentication features).

The backend authentication management provided by Django is very robust. It includes user authentication, access control (including groups and individual permissions), administrative panels, and logging of who does what.

Program Flow

The authentication module will be called whenever a user tries to access something. It will verify that the user is logged in, has permissions, and even write a short log message if it is an administrative function.

Data Flow

Because this module is not being written by us, it will not be following the standards we are using. Therefore, it will be using the standard Django classes to access database objects instead of going through the REST API.

While that is not ideal, it should be OK anyway. This part of the Django framework has been tested extensively, in many production environments. There is a low risk that something will not work, and if there is a problem there is a very high chance that upon being reported to the Django project, it will be fixed quickly.

Something worth noting is the way passwords will be handled. Django hashes all passwords in a variety of algorithms, as specified in the password field. It also salts the passwords. This is much better than the lazy way of hashing it (maybe) and storing it in a database column. For more details, please reference `https://docs.djangoproject.com/en/dev/topics/auth/`.

Potential Problems

Like I stated above, the largest potentional problems will be bugs in this piece of the code. Because we did not create, and we will not be supporting it, any bugs that we find will need to be reported to the Django framework community and we will have to wait for them to roll out a fix.

Another problem that might be common is the difficulty in expanding on the user profiles. This will not be a problem for us because Django did such a good job in keeping their framework extensible. All we need to do is declare a OneToOneField in our model, and add a few panels to the administrative interface and we will have an instant user profile. Not only that, but we will have successfully separated user information from the authentication information, which is considered good practice as it is faster and more secure.

Objects

The objects for the authentication section are completely managed by Django. Therefore, we will not be documenting these. If you are curious, please see the Django project for more information (`https://www.djangoproject.com/`).

6.6.2 The REST API

Description

This module describes the architecture of how the backend will manage features related to providing access to the REST API.

The REST API will provide a `urls.py` file, which will provide a handler for all REST methods. These could be achieved by simply using the tastypie Uniform Resource Locator (URL)'s, which can be generated by calling `include(api_v1.urls)` in the urlpatterns object.

Program Flow

The REST API will be activated by the Django chain when a request comes in for the access to the API, or when another module directly calls functions in the API during their response to a request from the user.

Having each module register itself when the system launches should work, but there may be a problem with things getting registered twice.

Data Flow

All data will be returned to the calling client either as a Python dictionary (if called directly from another module), or as JSON, Extensible Markup Language (XML), or YAML Ain't Markup Language (YAML) through a web request (as specified in the query filter using ?format=json).

The REST API will directly access the database. Initially, tastypie will be dealing with converting the Hypertext Transfer Protocol (HTTP) request to a method on the Django objects. Django will be generating the SQL that actually gets run, and encapsulating objects in Python classes.

Potential Problems

At first, we will be using a prepackaged Django module to help provide the REST API. This could be problematic because we don't know a lot about the implementation of the API, and if there is a problem, tracking it down could be difficult.

The good news is that django-tastypie is well supported by the community, and source code is liberally commented. It follows Python commenting standards, and should be fairly easy to navigate. In addition, members of our team have already used tastypie for projects in both work and recreation.

The most problematic bit about using tastypie for the initial run-through is the compatibility problems it has with Backbone.js. However, the community has developed an addition to Backbone to help deal with this. It is called Backbone-tastypie. It is a simple modification to the Backbone sync operations that allows it to communicate with a standard tastypie REST API.

When the web server is launched, it initialized all modules by calling their `__init__.py` file. In this file, each module can register themselves with the API. This will allow all modules to be self-contained, which is ideal in that it will allow us to upgrade one module without breaking everything else.

Early experiments indicate that the `__init__.py` file is called twice during the loading of the system. This could be solved a number of ways, or it might not even be a problem. It's very possible the first time the module is loaded it is just being checked for syntax and compiled, then the program is restarting with the compiled files.

Data Objects

The REST API will have each module extend the tastypie.resources.ModelResource class. They will, as needed, override some methods to allow custom resource handling. In a later version, we will create a custom pager and implement a way of "expanding" data if we need it.

Table 6.1: device/init.py

```
1  from tastypie.resources import ModelResource
2
3  from rest_api.apis import raw
4  from device.models import Device
5
6  print "Initing the Device module..."
7
8  class DeviceResource(ModelResource):
9      class Meta:
10         queryset = Device.objects.all()
11
12 raw.api.register(DeviceResource())
```

Table 6.1 shows the code that is used by the Device module to init itself and register that module with the raw API. The raw API will be a direct database mapping, and will be used primarily as a means to test the efficiency of the system in the first prototype. Currently, there is no authentication or authorization in effect. These features will be added at a later time. Please see the tastypie documentation (`http://django-tastypie.readthedocs.org/en/latest/`) for information on how this will be done.

Table 6.2: power data/init.py

```
from tastypie.resources import ModelResource
from tastypie.authentication import Authentication
from tastypie.authorization import Authorization

from rest_api.apis import raw
from power_data.models import Data

print "Initing the Data module..."

class DataResource(ModelResource):
    class Meta:
        queryset = Data.objects.all()
        authentication = Authentication()
        authorization = Authorization()

raw.api.register(DataResource())
```

Table 6.2 demonstrates what kind of code would be used to restrict access to the API. This module does NOT permit POSTing of data to the REST API, since this data will be supplied internally by the power_data module.

6.6.3 Satellite

Description

This module describes the architecture of how the back end will manage features related to managing Satellites.

The Satellite Management back end is in control of maintaining the information in the database pertaining to Satellites. The back end will also generate the framework for the web pages. It will pass the basic form of the Satellite Management page to the front end to be filled in with additional formatting.

Program Flow

The back end Satellite Management module will be activated by a request from the front end Satellite Management. This module will complete the request by

adding information to the SQL database or querying said database for specific data.

Data Flow

The Data Flow will behave in much the same manner as the Program Flow. The back end Satellite Management module will receive data from the Satellite Management front end via the REST API. This includes data that needs to be changed in the database, and requests for data. This module will then modify or query the database as needed, returning the requested data or a confirmation of modification to the front end through the REST API.

Sub-modules

Communication with the Satellites:
This modules serves as an interface and a form listener. It will receive Request Method (POST)ed commands for dispatch to the ZigBee network via a serial-port connected ZigBee. It will also receive the incoming data from the ZigBee network and store it in the Database. The data will consist of the current and voltage readings from each Satellite on the network.

Objects

The Satellite module consists of the following objects:

Table 6.3: satellite/models.py

```
1 from django.db import models
2
3 from device.models import Device
4
5 class Satellite(models.Model):
6         serial_number = models.CharField(max_length=9)
7         outlet = models.CharField(max_length=9)
8         unique_together = ("serial_number", "outlet")
9         device_id = models.ForeignKey(Device)
```

Database Description

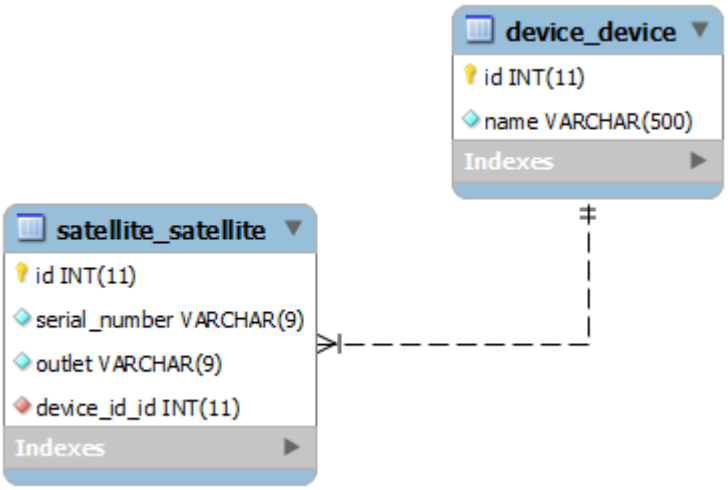
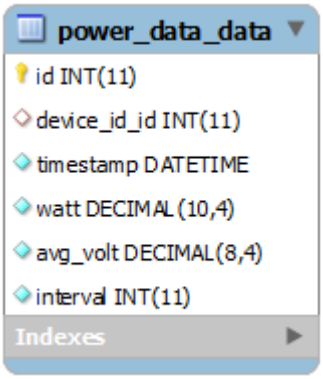


Figure 6.3: Satellite Module Database Schema

Figure 6.3 describes the database for the power_data module. This schema is here because it is so closely related to the Satellite module. The fields are expanded upon below.

satellite

- id - This is the primary key for this device. It will be indexed, and will be the primary means of updating and deleting objects.
- serial_number - This, combined the the outlet, form a unique key. This field will be used to identify the outlet to the user.
- outlet - This field represents the outlet on the satellite.
- device_id - This field is a foreign key to the device that is associated to this outlet. It will be used to log the data to the appropriate devices when it is inserted into the power_data module.



Data.png

Figure 6.4: Power Data Module Database Schema

Figure 6.4 describes the database for the Power_Data module. The fields are expanded upon below.

data

- id - This is the primary key for this device. It will be indexed, and will be the primary means of updating and deleting objects.
- device_id - This is a foreign key relating this piece of data to a device. This will be index, and used to aggregate appropriate data.
- timestamp - This field represents the time that this data was put into the database.
- watt - This field represents the total wattage being drawn by the device over interval. It is a product of the voltage and amperage, and amperage can be calculated from it.
- avg_volt - This is the average voltage pulled by the device over interval.
- interval - This is the length of time that was measured. It will be used to average the voltage and amperage

6.6.4 Devices

Description

This module describes the architecture of how the back end will manage features related to Device Management.

The Device Management back end will be in charge of maintaining the database with information pertaining to the Devices. The back end will also generate the framework for the web pages. It will send the basic format of the Device Management page to the front end to be filled in with additional formatting.

Program Flow

The back end Device Management module will be activated by a request from the front end Device Management. This module will complete the request by adding information to the SQL database or querying said database for specific data.

Data Flow

The Data Flow will behave in much the same manner as the Program Flow. The back end Device Management module will receive data from the Device Management front end via the REST API. This includes data that needs to be changed in the database, and requests for data. This module will then modify or query the database as needed, returning the requested data or a confirmation of modification to the front end through the REST API. The data for the basic format of the web pages will also be sent from the back end to the front end through the REST API.

Sub-modules

The Device-Satellite Association:
Devices will be associated to a Satellite at all times, unless the Device is disabled. That way we can associate the power consumption data from the Satellite with the correct Device. This association will be made in the database. These associations not be static, Devices will be able to be moved from one location to another. By associating the Device with each new Satellite it is physically connected to we can track that Device regardless of its location.

This module will implement the associations between the Devices and the Satellites.

Objects

The Device module consists of the following objects:

Table 6.4: device/models.py

```
1 from django.db import models
2
3 class Device(models.Model):
4         name = models.CharField(max_length=500)
5
```

Database Description

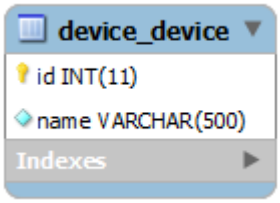


Figure 6.5: Device Module Database Schema

Figure 6.5 describes the database for the Device module. The fields are expanded upon below.

device

- id - This is the primary key for this device. It will be indexed, and will be the primary means of updating and deleting objects.
- name - This is the "name" of the device. This should be human readable and unique.

6.6.5 Power Bill Guestimator

Description

The Power Bill Guestimator backend will be in charge of maintaing a record of utility power costs for different times and usage rates. These rates are set by the user on the front end and stored in the database. It will be able to calculate the cost of the power used between certain time ranges and give that data to the front end to be formatted.

Program Flow

Data flows through this module when the front end requests the cost of power used for Satelites. This triggers a query of data from the database of the power usage history for that Satelite (or combination of Satelites) and a query of the rates that applied to the time period. With that data the module will calculate how much power was used at each rate and give that data back to the front end.

Potential Problems

Since rates can vary based on time of day, total power used, or other unknown factors, keeping track of rates may be tricky. Research must be done to find out the most common ways utilities charge for power and be able to store rates accordingly.

6.6.6 Factory Reset

Description

This module describes the architecture of how the backend will manage features related to the factory reset. The factory reset will restore everything to the exact same way it was when the system came out of the "factory". This means that all data will be wiped, and custom modifications will be wiped, and any software upgrade will be wiped.

Program Flow

This function will be invoked either through a call passed down from the web interface (it will be handled by the REST API, and the call will be intercepted and result in an action) or by a program picking up on the push of the physical factory-reset button.

Data Flow

Doing the factory restore will involve a good bit of trickery. First, we will need to put a partition on the disk that stores all the files that make up the "live" running system. This might be best done by simply putting a tarball and a script on a small partition at the end of the disk. The script will be responsible for wiping the primary partition, re-creating it, and extracting the tarball.

Getting the system to boot into the "restore" mode will involve modifying grub settings on the fly. We can do this, and it shouldn't be too hard, but it feels like a really bad idea.

After the machine is in recovery mode, something will need to launch the script to do the restore. This shouldn't be too difficult if just use an old-school `rc.conf` script or something of the sort. The script will also restore grub, so we don't have to worry about returning that to it's old good state.

Potential Problems

One of the biggest risks here is failing to boot into the restore partition and ruining our grub configuration. This would cause the customer to need to send us back the unit, we would have to flash it, and send it back. If we can get this feature working however, we will not have to worry so much about breaking the web UI by accident.

6.7 The Frontend

6.7.1 Authentication

Description

This module describes the architecture of how the frontend will create pages that allow the user to manage user and groups.

The authentication front end will consist of both the "login" page, and the user management pages. These pages either post directly to the server using standard form-submission methods (ie, not Asynchronous JavaScript and XML (AJAX)) or by using AJAX (if we have time to recreate the entire management system).

Program Flow

The user will first be presented with a login screen, which will take two pieces of information:

- Username - The user's username
- Password - The user's password

After these two pieces of information have been verified (ie, that password belongs to that user), the user will be logged in.

From here, the user has all kinds of options. The only ones that are interesting to this module is the settings panel, which includes a link to the user administration panel. The user administration panel will, initially, be a themed Django-admin view. Time permitting, this may be recreated to be a AJAX type interface.

Data Flow

All data will be sent to the server through POST requests, not through the REST API. This will allow us to skip over this module almost entirely. This is not consistent with everything else, and will need to be remade later on if there is time.

Potential Problems

Themeing the admin interface to match our site will be tricky. At best, it will involve telling Django to include some extra stylesheets or using a different template. At worst, it will involve modifying the Django stylesheets directly. Either option works, but due to version control it would greatly preferred to be able to tell Django what styles to use.

6.7.2 Satellite

Description

This module describes the architecture of how the frontend will create pages that allow the user to add and delete Satellites associated with the system.

It will receive a basic framework for the web pages in HTML from the back end module and use Javascript to render the rest of the page. The pages will be created with a mix of HTML and CSS from the back end and heavy Javascript from this module. These pages include the Satellite Management page and the Add Satellite wizard.

Program Flow

The user will be presented with the option to add or delete a Satellite. Adding a Satellite allows the user to then associate it with a Device. Deleting a Satellite will disassociate the Device from the Satellite, leaving it unassociated to anything. The Satellite will then no longer be available to be associated with a Device.

Data Flow

The Satellite Management front end sends a request to the back end through the REST API. The back end returns either a confirmation of action or the requested data to the front end via the API. This module receives the basic format for the web pages from the back end Satellite module. It then fills in the additional and page-specific content.

6.7.3 Devices

Description

This module describes the architecture of how the frontend will create pages that allow the user to manage devices in the system.

It will receive a basic framework for the web pages in HTML from the back end module and use Javascript to render the rest of the page. The pages will be created with a mix of HTML and CSS from the back end and heavy Javascript from this module. The Device Management front end includes the Device Management page, Add Devices page, and the Confirm Disable Device dialog.

Program Flow

The user will be presented with the option to add, edit, or disable a Device. Adding a Device allows the user to name the new Device and associate it with a specific Satellite. Editing a Device allows the user to change the Satellite associated with the Device. The user will not have the option to edit the name of the Device once it has been created in order to maintain consistant data. Disabling a Device will remove it from all graphs and tables and data will no longer be collected on the Device. Devices cannot be deleted in order to maintain past data.

Data Flow

The data will be sent to the Device Management back end module through the REST API. The back end returns either a confirmation of action or the requested data to the front end via the API. This module receives the basic format for the web pages from the back end Device module. It then fills in the additional and page-specific content.

6.7.4 Frontend - View Data

Description

This module describes the architecture of how the frontend will create pages that allow the user to view various pieces of data in various formats.

The View Data front end is responsible for providing the user with data regarding each Device and Device Group. It will provide three separate graphs and tables with data specified by the user; Device Power Consumption, Monthly Power Consumption, and Power Consumption Over Time.

This module will be using jqPlot to render the graphs on the web page. This will allow the back end to provide the data for the content and the front end to plot the graphs as needed.

Program Flow

From the Home page the user will be able to select which graph or table they would like to view. These graphs and tables provide the user with information pertaining to the Devices specified and within the time-frame selected.

Data Flow

This module requests data for the graphs and tables from the back end through the REST API. It receives a response and renders the graphs and tables with the data provided.

Potential Problems

The main sources of concern with this module have been stated previously in Section 3.5. We will be facing a large learning curve as we must familiarize ourselves with the software we will be using, such as jqPlot. There is also the problem where some users will not have Javascript supported in their browsers. This will keep them from being able to view any of the information presented by this module using Javascript or jqPlot. These problems have been addressed previously.

Sub-modules

Device Power Consumption:
This is a table that will contain the name, current power consumption, and average power consumption rate of each selected Device or Device Group. The content will be sortable by each field.

Monthly Power Consumption:
This is a graph with power in watts on the vertical axis and time in months on the horizontal axis. The user will specify which Devices or Device Groups and the range of months they would like to view data for. The graph will then display a line graph with a separate lines for each separate Device, showing the power usage over time.

Power Consumption Over Time:
This is a graph that allows the user to specify not only the Devices and Device Groups to view, but the time interval as well. The user will specify whether time, on the horizontal axis, will be measured in days, weeks, months, quarters

or years. The units of the vertical axis will be power in watts. The graph will then display a line graph with separate lines for each separate Device, showing the power usage over time.

6.7.5 Power Bill Guestimator

Description

This module describes the architecture of how the frontend will create pages that allow the user to view guestimates of their monthly power bill, in various scenarios.

It will receive a basic framework for the web pages in HTML from the back end module and use Javascript to render the rest of the page. The pages will be created with a mix of HTML and CSS from the back end and heavy Javascript from this module.

Program Flow

When the user would like to estimate a power bill they first have to select what devices they want to view and for what time period. With that information selected, this module asks the backend for the power cost data for said devices.

When the user is modifiying the utility company rates they will be presented with a wizard that ill guide them through inputting them. This wizard will ask them several questions about how the utility company charges for power and will give data to the backend accordingly.

Potential Problems

One potential problem is having a utility company have rates change in a way we did not anticipate. One other issue is not having rates defined before trying to calculate the bill.

6.7.6 Factory Reset

Description

This module describes the architecture of how the frontend will create pages that allow the user to reset the system back to factory default settings. It will receive a basic framework for the web pages in HTML from the back end module and use Javascript to render the rest of the page. This front end module will be a page that will allow the user to initiate a Factory Reset.

Program Flow

When the user loads this page they will be presented with an option to perform a factory reset. If they choose that option, this module will ask the user if they are sure and list the consiquences of doing so. If the user still agrees, it will request a factory reset to be performed from the back end module.

Potential Problems

Making sure only administrators can reset the device to reduce the risk of accidentally breaking it. Also may want to implement a method of backing up and restoring all of the data.

7 | The Server

7.0.7 Description

The server will be responsible for collecting and recording data given to it by the Satellites, storing the display, and responding to user requests. This section reiterates the hardware of the server, which is also documented in the hardware section because so many parts of it belong specifically to the Zigbee things.

7.0.8 The Hardware

Mainboard

The Server's main hardware component is the mainboard, a SYS9400-ECX Developer-Ready Reference Platform. It's a small form-factor, low-power machine with the following specs:

- 1.6 GHz Intel Atom E6XX Series Processor
- 1 GB DDR2 RAM
- Roughly 6" by 4"
- Various connection interfaces:
  - 2x SATA ports
  - Header for SATA power
  - Ethernet port
  - 5x USB 2.0 ports
  - GPIO pin header

Potential Problems

If for whatever reason using this mainboard falls through: It should be noted that the requirements for the Server hardware concern not just specifications, but interfaces as well. In particular, this project requires at least Ethernet, 2 USB ports, a SATA port, and accessible GPIO pins.

Storage

The Server's mainboard is connected via SATA to a 40GB SSD.

Power Supply

An adapter rated for 12VDC @ 3A is used to connect the Server's board to mains electricity.

7.0.9 Software

In addition to storing our custom written software, which is documented in the software section, the server will utilize a good number of other programs. This section will document what software will be used, and for what purpose.

Ubuntu Server 12.04

The base system running on the server will a Ubuntu server base install. We will be installing additional packages on top of this, using the aptitude package manager.

The filesystem will contain 3 partitions.

- /boot - This partition will store the files needed to boot the system, including the Linux image and grub configuration files
- / - This partition will store all the files used during run time, including configuration files, software, and static data
- <Restore> - This partition will contain a minimal boot system, and a tarball will all the factory default software. It will be used to restore the /boot and / systems when requested, but will not normally be mounted

Apache2

We will use an Apache2 server with `mod_wsgi` to deal with responding to requests on the REST API or executing parts of the web application. This server

will not be server static files, as that would be a waste of it's features (which includes caching of recently accessed pages to avoid re-executing the same code several times over).

lighttpd

The lighttpd is also a web server. This web server is much less robust than Apache, but much faster at serving up static files. It will be responsible for giving images, javascript, css, html, and other static content to the user.

MySQL

Initially, we will use a MySQL database to store data from the Satellites. This may be changed to posgresql if performance becomes an issue, but MySQL is much easier to install and configure.

Python

To run our application, we will need Python 2.7 installed. We will have a whole bunch of modules, which will be discovered as we are doing the lower level design and implementation, installed along with Python. To track packages we need for Python, we will use pip. In addition to pip, we will use virtualenv to simply the maintance and portability issues of using Python + pip in place of the aptitude package manager. We are not using the aptitude package manager because it does not have the most recent PyPy packages.

IP Tables

IP tables will be used to prevent traffic from bridging the server, and to prevent the server from sending out requests anywhere. This will prevent a wanna-be-hacker from using the machine to attack the client's network. Note that this will not help if a hacker got root-access to the Server, but it is better than nothing.

8 | Non-functional Requirements

8.1 Summary of Requirements

The system is made up of the following components:

- Satellites
- The Server
- The Display

Each of these components have their own requirements. The basic summary of the non-functional requirements of each one would be:

8.1.1 Satellite Requirements

- Able to work in groups of up to 255 for any one Server within a 500 meter radius
- Conveniently sized and shaped
  - Unobtrusive
  - As small as possible given the hardware
- A small LED near each socked to indicate power
- A button for turning On or Off
- A button to initiate connection to the Server
- Will be an unobtrusive color other than white
- Data sent from Satellite to Server shall be encrypted

8.1.2 Server Requirements

- Enough hard drive space to hold 5 years of data

- Supports up to 255 Satellites
- Receives a reading from any Satellite at any time
  - Losing readings is considered an error
- Hardware specifications:
  - 1.2 GHz CPU, i386 architecture
  - 512 MB Ram
  - 40G SSD Drive
- Small and unobtrusive case
- Color shall not be white
- User can add Satellites at any time
- Encryption used on all communication to Satellites
- User cannot log in to the Server and get a shell

8.1.3 Display Requirements

- Interface is easy to learn
- Maximum of 500 milliseconds to load a page
- Data is not lost when a device is moved to a new Satellite
- Not vulnerable to SQL injection
- Not vulnerable to XSS
- No unauthorized access
- Data transmitted to the Display is encrypted
- User has a modern web browser:
  - Internet Explorer 9
  - Chrome 22
  - Firefox 15
- User's PC can render a page in their web browser that contains:
  - Javascript
  - Images
  - Extensive mark up

Bibliography

[1] andyet. icanhaz. http://icanhazjs.com/.

[2] Robert Faludi. *The Chicago Manual of Style*. Building Wireless Sensor Networks: with ZigBee, XBee, Arduino, and Processing, fourth edition, 2011.

[3] jQuery Team. jquey. http://jquery.com/.

[4] Backbone.js Team. Backbone.js. http://backbonejs.org/.

[5] Django Web Framework Team. Django web framework. https://www.djangoproject.com/.

[6] RequireJS Team. Requirejs. http://requirejs.org/.

[7] Paul Uithol. Backbone-tastypie. https://github.com/PaulUithol/backbone-tastypie.