POW-R

Power Outlet Wireless Reporter

Grace De Geus
Charles Hathaway
Forest Immel
Nate Pickett
Niloc Quimby

April 24^{th}, 2013

Requirements

Product Requirements

- Complete Requirements Document
- Scope of Product versus scope of Project
- Notable Product Requirements:
  - Satellite power requirement of less than 1 Watt
  - Server power requirement of less than 10 Watts
  - Power Bill Guesstimator
  - Production cost of Satellite to be under $7.50 (USD)

Functional Requirements - Satellites

- The Satellite shall have the ability to plug into standard National Electrical Manufacturers Association (NEMA) 5-15 mains electrical outlets.
- The Satellite shall report with less than 5% error on current and voltage readings.
- The Satellite shall transmit information every 60 seconds.
- Losing readings shall be considered an error.
- The Satellite shall have the ability to connect to the Server wirelessly.

Functional Requirements - Server

- The physical Server shall reside inside the monitored building.
- The Server shall host the web server for the Display.
- The Server shall have the ability to sync with Satellites.
- The Server shall have the ability to connect to the user's network.
- The Server shall have the ability to connect to Satellites wirelessly.

Functional Requirements - Display

- The Display must provide Device management. This includes the ability to:
  - Add Devices, Disable Devices (does not include deleting Devices)
  - Modify Devices
  - Rename Devices
  - Change outlet association
- The Display must provide Satellite management. This includes the ability to:
  - Add Satellites
  - Remove Satellites

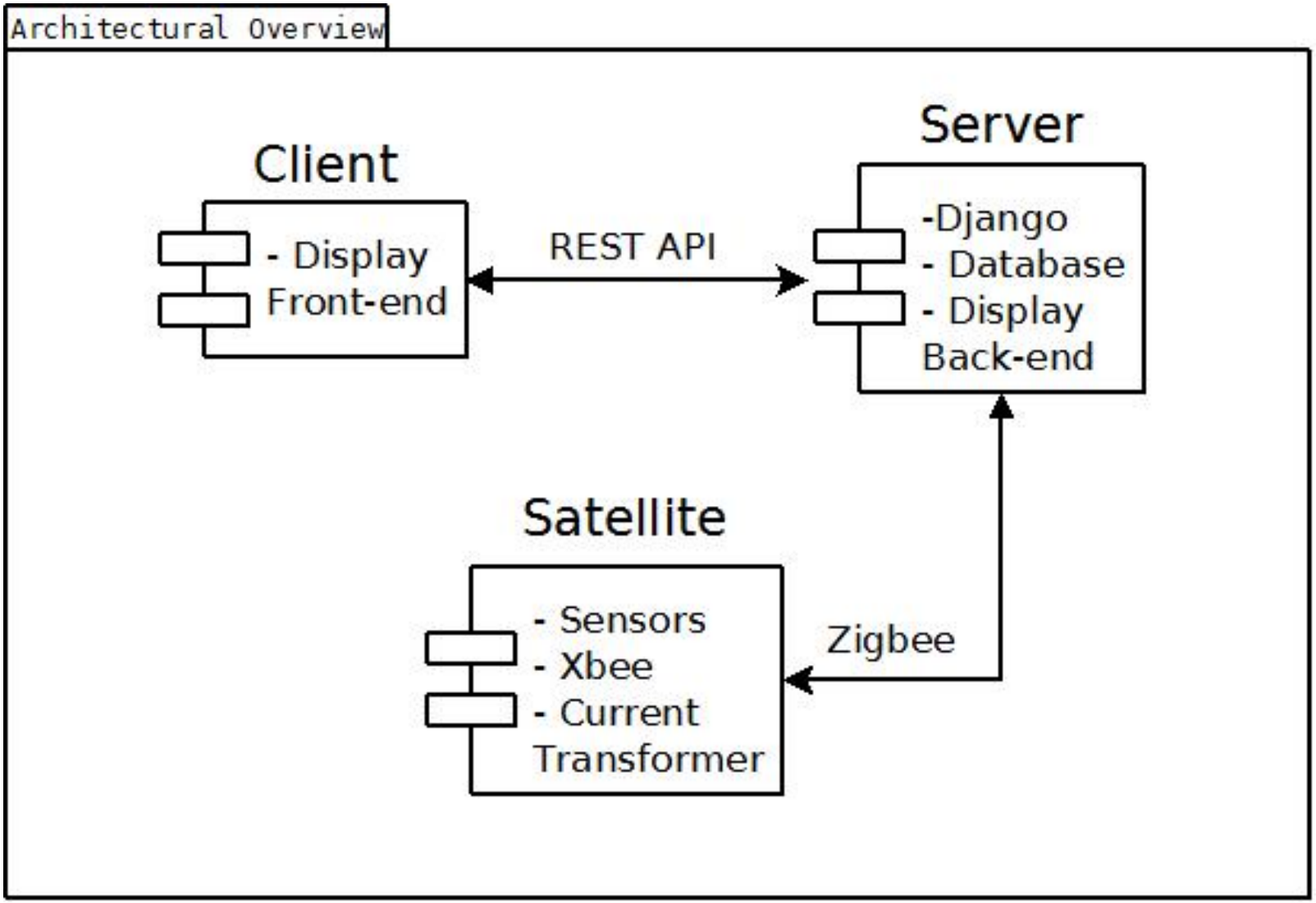
Functional Requirements - Display cont.

- The Display shall provide the ability to view data in graphs, including:
  - View Device Power Consumption (Showing power consumption on a per-device basis)
  - View Power Consumption Over Time (Showing power consumption over a specified range of time including total and per-device power consumption)
- The Display shall have the ability to compare power usage of devices. The Display shall provide the ability to view data in graphs:
  - View Device Cost Over Time
  - Showing the cost to run a device over a specified time range
  - Showing a comparison of device costs over a specified range with a given interval.

Non-Functional Requirements

- The Satellite shall be conveniently sized and shaped.
- The Satellite shall have a small LED on it to indicate it is powered.
- The Display shall have a an intuitive interface that is easy to learn.
- The Display shall provide the user with feedback within 300 ms (responsiveness).
- The Display shall retain all data pertaining to a Device when it is associated with a new Satellite.

Overview


Architectural Overview
Client
- Display
Front-end
REST API
Server
-Django
- Database
- Display
Back-end
Satellite
- Sensors
- Xbee
- Current
Transformer
Zigbee

Why XBee Radios?

- Small form factor (just larger than U.S. quarter)
- Low power consumption (˜.1 W)
- Talk over ZigBee 802.15.4 standard

ZigBee Specification

- High level communications protocol
- Designed for low power digital radios
- Mesh network topology
- Network can expand on the fly
- 2.4GHz operating spectrum

ZigBee Mesh and POW-R

- One Coordinator per mesh
  - Maintains mesh
  - Receives transmissions from all router XBees
  - Attached to POW-R server via Arduino
- All Satellites have router XBees
- Router XBees "bounce" transmissions to Coordinator

Coordinator Design

Must:

- Host Coordinator XBee
- Host LCD to display IP
- Listen for readings
- Send readings to Server

Coordinator Implementation

Arduino:

- Talks to LCD using SoftwareSerial
- Listens for readings from XBee
- Sends readings over USB cable

Server Design

Server must:

- Be small
- Consume as little power as possible
- Listen for readings from Coordinator
- Store readings
- Host all of the Display

Server Implementation

Raspberry Pi:

- Small form factor (˜8.5 x 5.6 cm)
- Low power consumption (˜3.5 W)
- Acts as data center and web server for Display

Software Architecture

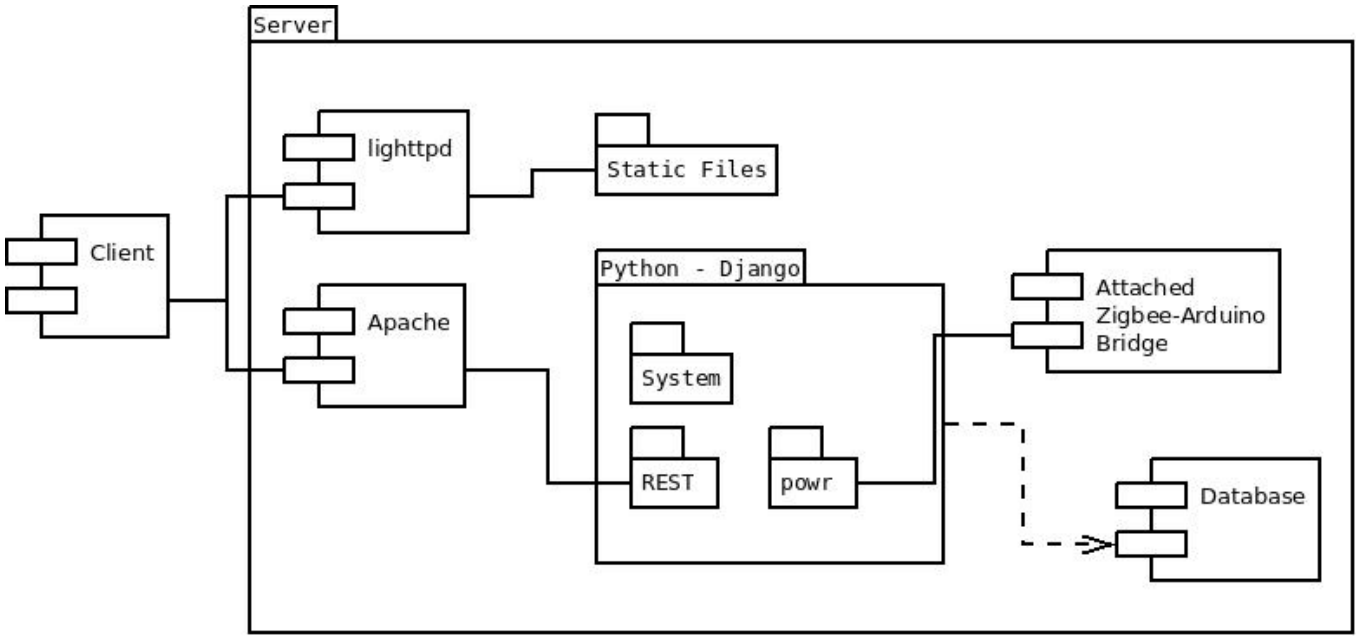
Software Overview

- Python on the backend
  A. Django for the REST, HTTP stuff [**?**]
  B. Custom python to interact with Arduino interface
- Heavy Javascript on the frontend
  A. jqplot for creating graphs (jQuery included) [**?**]
  B. django-compress to reduce the Javascript files to a manageable size.
  C. AngularJS for a MVC architecture that consumes the REST backend

Backend Overview

- Django will be used to handle database
- Tastypie will be used to prototype the REST API
- Each functional area of the project is a Django module
  A. System
  B. POW-R
  C. REST API

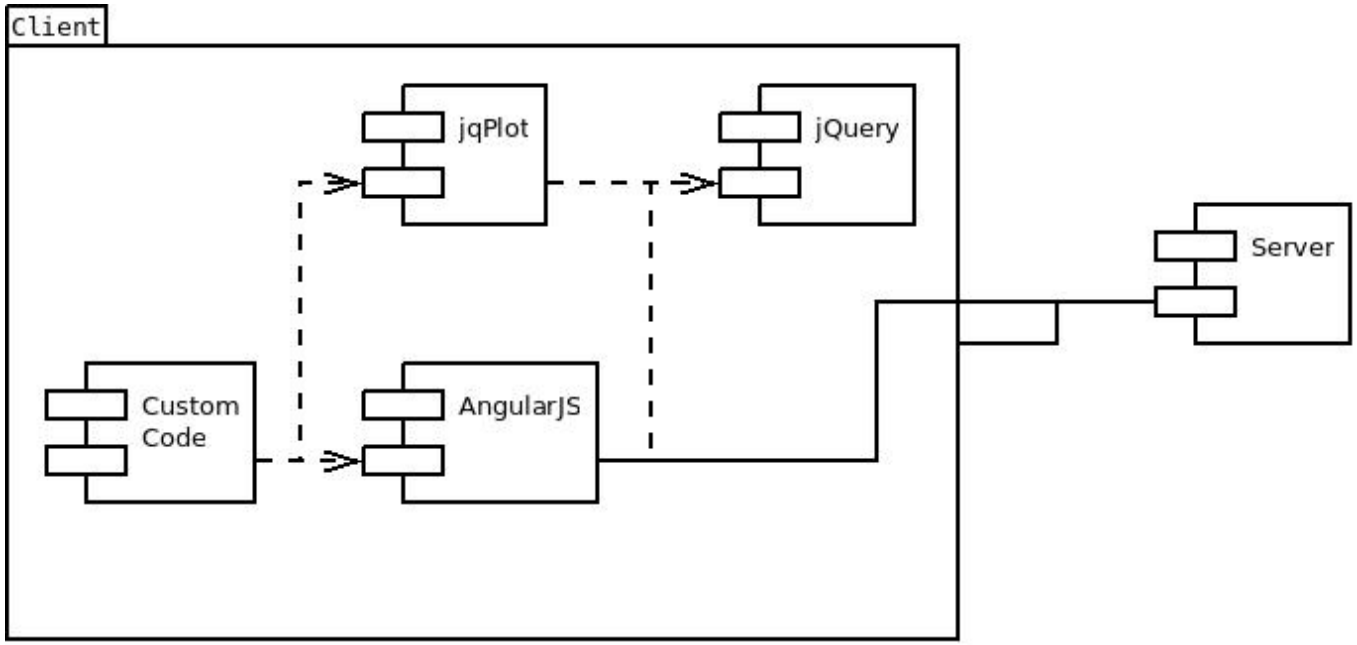
Backend Architecture



Frontend Architecture

- django-compress
  A. Minifies the JavaScript code so clients load faster
  B. Easy to use, can be tested
- AngularJS - MVC Architecture on the client-side
- jqplot - Renders charts and graphs
- jQuery - Deals with all the behind-the-scenes AJAX stuff

Frontend Architecture



Software Implementation

Software Implementation

- Went more-or-less according to plan
  1. We switched away from Backbone.js and iCanHaz because AngularJS covered both domains
  2. We didn't use Asynchronous Module Definition (AMD) because very few libraries supported it
  3. There were some small modifications to the REST API to make it more compatible with the world
- Biggest software problem was the complexity of the setup
  1. Because of the number of libraries and frameworks, it was difficult to do development in Windows
  2. We developed two solutions; a completely isolated Python development environment, and a VirtualBox for people with Windows

Software Implementation

- Some things didn't make it to the final product for a variety of reasons
  - Adding satellites was ditched because it was too confusing for an end user
  - The power-bill-guestimater was ditched because it would be impossible to accurately track all power consumption, thus leading to incorrect guestimates
  - The user-management stuff was simplified because we don't need a complex permission system
- Some things were added
  - We used Intro.JS to create a "help" feature in our website
  - We added a JS compressor to keep the codebase small when delivered to the client

Testing

Test Cases - Unit

- REST API
- Unicode and Backend
- Python Automated Tests
- Zigbee Mesh Test
- Voltage Circut Test
- Current Circuit Test

Test Cases - Integration

- Create a Graph
- View Graphs
- Data Retention Across Satellites
- Zigbee Data Transfer
- Satellite-to-Server Protocol

Test Cases - System

- Log In
- Log Out
- Add Device
- Disable Device
- Add User
- Remove User
- Rename Device
- Reassign Device

Test Cases - Acceptance

- Ease of Learning
- Examining the Satellite
- Examining the Server
- Data Loss Error
- Display Responsiveness

Test Results

- Failed Tests:
  - Ease of Learning
  - Remove User
- Corrective Action Taken
  - Edited four tests to accomodate late changes
  - Adjusted User Management page to make "Remove User" test passable
  - Made plans for UI changes that would allow "Ease of Learning" test to pass

Lessons Learned

- Order parts ASAP
- Understanding new material
- Team dynamics
- Time management

How would we do it all over again?

- Start development sooner
- Stick to schedule

Future plans, potential improvements

- Satellite functionality
- Home Automation Framework

Demonstration time!

Check it out!

Questions and Closing

Questions?

Presentation made using LaTeX

Our website: `http://powr.logrit.com/`