Open Energy:
Platform for Energy Transactions

Final Report

Team #41

Arun Sondhi: Software Engineer

Alec Dorenkamp: Software Engineer

Noah Eigenfeld: Software Engineer

Brendon Geils: Founder/Software Engineer

Jack Myers: Hardware Engineer

Joe Staudacher: Hardware Engineer

Client: Open Energy

Adviser: Goce Trajcevski

sdmay18-41@iastate.edu

sdmay18-41.sd.ece.iastate.edu

Revised: 4/22/18

Table of Contents

1 Introduction

In this section, we will discuss the problem the foundation of our project is based around and our proposed solution. In the following section, we will discuss our approach to the project, the reasoning behind any decisions that led to that approach, and the context in which our solution lies. In section three, we will introduce the high level design for our project and how we approached various tradeoffs and technical challenges. Section four introduces how we went about completing the project in terms of planning our two semesters and what resources we needed to draw from. In section five, we dive into the detailed design in terms of the functional modules that make up our final implementation. Finally, in section six, we detail how we tested our solution and verified that it was complete and robust.

1.1 Problem Statement

The primary aim of our project is to incentivize renewable energy generation from individuals and small businesses by facilitating peer to peer trading of surplus energy. By creating a free market environment for energy trading, individuals will think more about how they produce and consume energy, and will be more inclined to generate energy of their own. With this new understanding and market accessibility, energy prices will fluctuate to be at parity with their true value, not solely what the utility company dictates. A more detailed description of our implementation of this free market solution is covered in the design portions of this document.

Our secondary goal is aiding in the decentralization of the power generation market. The interconnectedness of grids has already contributed to the reduction of blackouts, as one individual power plant or utility company is not solely responsible for all energy generation. At the time of writing, a Chicago grid can pull from a Toronto plant if they approach their capacity curve. Our system can aid this interconnectedness a degree further, as the power loss and cost to transfer energy a mile up the road would be less than the power loss and cost to move that energy from Toronto to Chicago. A more decentralized grid would help the overall grid to be robust to fluctuations, as the sources of energy would be widespread and independent. Figure 1 displays how this distributed energy approach allows the energy generation to more accurately track the demand.

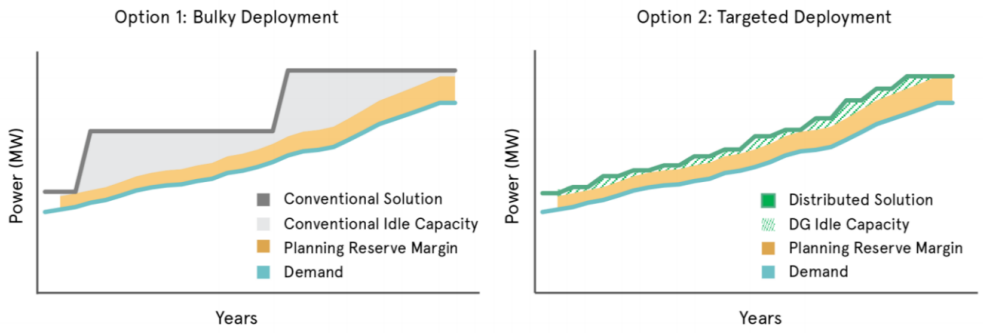


Figure 1: Bulky vs. Targeted Deployment of Energy Resources [7]

The connection between decentralized energy generation and our solution is clear. We hope that an open market will allow individuals to operate their own renewable energy sources like solar panels or wind turbines with increased economic feasibility. If this goal is realized, energy production will be less reliant on large plants, giving consumers a more diverse array of energy sources than those currently available.

An additional side effect of incentivizing renewable energy generation is that it will also contribute to reducing climate change. Renewable energy usage is seen as one of the key ways to target this problem, but a large portion of worldwide energy does not come from this "clean" energy generation. To change this, the generation of renewable energy needs to be more accessible to individuals and businesses, rather than only those who have the resources and capabilities of a large energy company.

Our project can be broken into two high-level components: a simple IoT (Internet of Things) capable power meter and software to facilitate peer to peer trading of surplus energy. Individuals using our hardware and software will be able to buy or sell surplus energy at rates favorable to those offered by the utility company.

1.2 Operating Environment

The operating environment for our solution was a relevant factor when designing the final implementation. The smart meter was designed with the assumption that the consumption and production lines are in the same location within a weather-safe area.

On the software side, the operating environment will be the economic and political climate in which our solution is being used. There are many legal factors that could come into play with this kind of trading. Use of the utility company's infrastructure or trading between different cities, states, countries is one example. These are all factors that would need to be explored in further detail if this project were to be expanded beyond the proof of concept.

As we pass our progress to our client, Open Energy, more focus can be put into refining the robustness of the hardware and making sure our software implementation integrates effectively with the economy and politics of the location where our solution is being used.

1.3 Intended Users and Intended Uses

The users can be split into two groups, which we refer to as "producers" and "consumers." The producers are the users who will supply excess energy that they produce into the system. Producers look to maximize the profit that they can create from producing energy, and our open energy market will enable them to do just that. In order to best serve these users, we are minimizing the transaction costs and maximizing the ease with which they can find buyers for their energy. The consumers are the users who will be consuming the excess energy that producers create. We can best serve them by minimizing the transaction costs and making it as easy as possible for them to find producers to provide them energy.

The intended use is to incentivize generation of energy by creating an accessible market for both producers and consumers. The intended users will expand to include anyone with an electrical service connected to their home or business. Our intent is simply to attract as many users that already have direct generation installations as possible. More producers in our user base will allow us to obtain more consumers by making the market more competitive and attractive. In due course, the flow of consumers to the marketplace will

encourage more people to become producers and install their own direct generation setups. This circular growth will eventually lead to us reaching our goal of widespread incentivisation of direct generation.

1.4 Assumptions and Limitations

Because our goal is to make an impact on large scale issues like the energy market and climate change, it is important that we clarify what restrictions exist with our solution, and what aspects we are leaving for future work. The following list gives the most important of these assumptions and limitations regarding the scope of our project.

Assumptions:

- Enacting the distribution of power after our transaction is completed is outside of the scope of our project, including any new power equipment that could be required on the distribution side
- For a full implementation of our project, an agreement will have to be completed with the utility company owning the power infrastructure so they will allow these transactions to take place
- Failing to reach an agreement with a regular utility, an agreement will have to be completed with a developer of a subdivision, whom may own the power infrastructure of said subdivision
- The level of testing that we will complete will be within an individual municipality, so interstate/international trading laws will not be applicable

Limitations:

- The cost of the IoT smart power meter must not exceed that of the average power meter used in Ames, IA (the area of testing)
- The purchase of all hardware components and software licenses must be approved by our client and must not exceed the amount of funds they have allocated for the project
- The cost for user operation of our smart power meter must be minimal to make the implementation worthwhile for the customer measured by the cost benefit of the system

1.5 End Product and Other Deliverables

Smart Meter

For the implementation of our marketplace, an Ethernet or WiFi-enabled smart meter will need to be installed at a user's property to read the flow of energy on the production and consumption lines into their property. The smart meter connects to the internet, providing the necessary data to the software component of our project. It accomplishes this by reading the current and voltage values on both the input and output lines of a user's system and reporting this to our databases. Along with the basic data acquisition and communication functions, the computing module of the smart meter is capable of displaying vital information to the user via a display screen directly on the meter.

Energy Marketplace Implementation

Power transactions, both consumption and production, are made and recorded using a non-relational database which is manipulated by a marketplace controller on our backend system, which is a Ubuntu Linux environment hosted on AWS. This system coordinates with the users marketplace settings which live in the web interface for seamless configuration.

Web/Mobile Application

Users manage their power transactions through a web application, which interacts with the energy marketplace through REST API calls, receiving and mutating information through a secure MongoDB connection. This application allows the user to monitor their energy use and production through viewing different energy charts. The user may also buy and sell energy through a marketplace interface. Users are also able to download personal usage and production statistics, which are actively aggregated within the MongoDB. The web application includes login and account creation capabilities.

2 Approach and Statement of Work

In this section, we present our project requirements, options we considered for meeting those requirements, and our proposed approach.

2.1 Functional Requirements:

The requirements that had to be completed for our project to have an operational implementation are:

1. *An IoT Smart Meter device*
    a. *Device to read power at the meter and send this information to the cloud at second granularity*
2. *Application on web and mobile for visualizing data and configuring marketplace settings*
    a. *The ability to see real-time consumption and production data*
    b. *Access to configure settings to which the marketplace will trade on*
    c. *The ability to download data for offline analysis*
3. *Backend processing block for clearing market transactions and managing user data*

2.2 Non-Functional Requirements:

The main qualitative elements of our project that we strived to achieve are as follows.

1. *Ease of setup*: Any user must be able to easily install and configure our hardware/software.

2. *Portability*: The web application must be usable on various platforms.

3. *Robustness*: The hardware must be able to withstand the conditions in which it is installed and be able to respond to signal loss and power outages. The software must be tested to handle edge cases and avoid fatal errors.

4. *Scalability*: The hardware and software must designed in such a way that it could handle a large network of homes that would be required for a full implementation of our design.

5. *Security:* The anonymity of the users must be protected, and all transactions must be secure.

2.3 Overview of the State of the Art

We now present an overview of the previous work and literature that we have reviewed regarding comparable technologies.

2.3.1 Literature review:

Our literature review was focused on understanding the current industry practices for distributed energy related work. Our review was on both the software and hardware that the market had available. We go into detail in the Home energy consumption solutions section below on the specific companies that are working in this space.

For the software perspective there are a number of solutions that are available in separate applications. Commonwealth Edison Company (ComEd) for example has a visualization tool for their customers. They also provide a capability to download data. The broader community has begun endorses the Green Button, which is a initiative for people to have the ability to download their energy data in a standardized format.

From the hardware side companies such as Sense provide hardware add-ons that can be used to track energy consumption and production data. The Sense team also has a application for monitoring this data. The following section provides a literature review with a focus on the marketplace and trading aspects.

2.3.2 Home energy consumption solutions:

TransActive Grid + LO3 Energy

TransActive Grid is the IP holding company of LO3 Energy. The Consensys team previously teamed up with LO3 Energy to test the viability of energy blockchain. In 2016 there was a blockchain energy implementation in New York City, specifically the Brooklyn borough. The pair wired up two Brooklyn residences and traded energy on the blockchain. The details of this transaction were left out due to IP implications from the organization [7].

Besides the 2016 article LO3 energy has been relatively quiet about their progress. The gist from their end is that energy blockchain is viable from a technical perspective. The Brooklyn Microgrid is now being brought to other countries, most recently Germany.

Grid+

Grid plus is an Austin, TX based energy retailer. The parent company is New York City based ConsenSys. Grid+ provides a smart argent and blockchain implementation. The smart agent is used to buy and sell the GRID token. They also have another coin, BOLT. This coin is known as a stable coin. The advantage to stable coins is while a cryptocurrency will readily fluctuate in price a stable coin reflects a more stable currency, in this case the USD. A single BOLT is equivalent to a single USD. The end user will have the advantage of near real time service and security of the blockchain while not having a wildly fluctuating cost of energy.

Grid+ however, does not have a peer to peer energy model. The business model they use is to cut administrative costs from the distributor and retailer in the energy supply chain. From a technical perspective the Grid+ energy blockchain implementation is using the ethereum blockchain [5]. The ERC20 token standard is their building block for the smart contract. The focus of their contracts is on the token and ICO, while they have long term development outlines for their energy business.

SolarCity

SolarCity wrote a report [11] in which they explain the distributed energy environment and its subsequent advantages. They specify that the distributed energy model is a net benefit for society through benefits related to voltage and power quality, conservation voltage reduction, grid reliability and resiliency, equipment life extension, and reduced energy prices. The hurdle they present is that the current utility incentive model does not coincide with a distributed energy focus. To convert to their model they proposed legislative changes.

Power Ledger

From the Power Ledger white paper [12], the Australian based energy blockchain startup looks to provide peer to peer energy trading. They will facilitate these energy transactions with the ethereum blockchain. Power Ledger provides very little technical documentation of their work, but boast a trading matching algorithm, meter reading device and token sale.

2.4 Possible Risks and Risk Management

As we began the development of our project, there were a few concerns that we were aware of that had a chance to hinder our progress. Though our team members have a wide array of backgrounds to serve as a knowledge base for our project, a project of this magnitude inevitably has the possibility of unexpected roadblocks or safety concerns. Our main concern for the hardware was safety in relation to interacting with 120/240VAC. We tried to minimize our interactions with high voltage lines, as well as making sure we had physically secure and insulated testing environments when we had to interact with these voltages. For an installation of our system in a home or business, a licensed electrician would most likely be required. We also worked diligently to ensure that the software we wrote is error-free and secure, and we identified security vulnerabilities that we were not able to fix, so that they can be addressed in future work.

Information security is also a very large concern with our project. The steps listed below were taken to enhance the security of our project. Because of the large scope of the project, and the depth of some of the security concerns, a portion of the security implementation will be left for future work.

2.4.1 Implemented work addressing risks

1. Smart Meter Information Transfer
    a. API keys are used to ensure data is only received from registered smart meters
    b. Server public-private key implementation used to prevent sniffing of posted data
2. Web
    a. SSL used to secure web application API requests
    b. Backend route authentication
        i. AuthToken passed with context in React to only allow verified users to access API
    c. Prevent injection/XSS by using input validation
3. User Security
    a. Okta is used for user validation - same standard as Iowa State University

2.4.2 Future work addressing risks

1. Limit server brute force attacks by banning IP addresses with too many illegitimate ssh attempts
2. Restrict server access to certain IPs (was not feasible on the ISU campus since IPs change)
3. Information privacy
    a. Mask user information to a certain extent when displaying graphs with smart meter locations, using pinpoint markers only for administrators, and a general heatmap for all users
4. Market
    a. One of the largest concerns is preventing users from spoofing production data. This is a large concern and will take a lot of work. Some options include:
        i. Document user's max production capacity, though this does not prevent them from overstating production while staying below their max production capabilities

ii. Compare production relative to other geographically close producers with similar/identical production sources, while also factoring outside conditions

iii. Implement anti-tampering monitoring on the meter

3 High-Level Design

In this section, we discuss the various tradeoffs that we investigated, followed by the description of the overal design.

3.1 Design Tradeoffs/Technology Considerations

We researched various ways to go about solving our problem. Those that received the most notable consideration are detailed below.

3.1.1 Software

The main technologies that we considered using for our marketplace are as follows.

Ethereum Approach

This approach entails using the open-source ethereum (blockchain) platform to develop smart contracts for buying and selling energy. Using ethereum, we could utilize their stable Solidity language to implement our smart contracts. The advantage here is there are many projects built with this stack allowing for more resources and support.

Hyperledger Approach

This approach consists of using the open-source hyperledger (blockchain) platform, which is newer compared to the more established ethereum approach. Hyperledger is supported by larger organizations such as IBM. This allows the technology to stabilize long term with the backing of a large company, compared to the burn-out many open-source projects that lack a large backing organization have seen.

Traditional Marketplace Approach

Another considered alternative was to use a traditional marketplace instead of a blockchain implementation, allowing us to provide cheaper transaction rates to our customers. This is because paying for each blockchain transaction is a fairly expensive overhead ($0.20-0.40 in the last 6 months) to selling energy. Implementing a traditional marketplace also has the advantage of being simpler and easier to implement.

Despite its increased security and transparency, we chose to move away from the use of blockchain technologies due to its bandwidth restrictions. Instead, we choose to pursue the traditional marketplace approach for it's cheaper overhead and simplified design.

3.1.2 Hardware

The platforms which we considered for the smart meter are as follows.

Arduino

Advantages:

- Entire team has experience
- Multiple libraries and shields for Wi-Fi and other processes
- I/O system is the easiest

Disadvantages:

- Little learning in terms of team's intellectual growth
- Does not provide a strong IP story

Arduino would be best used if we encounter high I/O in our system while not requiring heavy web based protocols. The shields and libraries are cohesive for simple Wi-Fi connectivity, but lack libraries for some of the networking functionality that we require.

Raspberry Pi

Advantages:

- Easiest Wi-Fi connectivity process
- Experienced members on our team
- Many available built in libraries and add-on modules
- Allows web server code/processing
- Rolling new updates is simpler

Disadvantages:

- Does not provide a strong IP story
- Single point of failure on the system
- Takes a significant amount of time to boot up

A Raspberry Pi system would allow us the greatest extensibility and code reusability. Many of the Arduino advantages are also present with the Raspberry Pi. By connecting direct, we can remove a node that would be required for computational logic (cloud or local server) that both the Arduino and PCB require.

PCB/Embedded

Advantages:

- Improve our knowledge of fabrication and low level hardware/software
- Capability to be fastest processing
- Least amount of resources used - power/computation

Disadvantages:

- Learning curve is highest
- Development cycle is the longest
- Documentation for our application is minimal

In going the PCB/Embedded route we had the opportunity to become more adept in a technology our team is not familiar with. This would have also provided us the best intellectual property scenario as the technology is not easily reproducible in code. The drawbacks are large in terms of development cycles and resources that we can tap to work through any issues. Many otherwise seamless processes such as Wi-Fi connectivity would be more difficult on this platform. However, the finalized product has the potential to be faster while consuming less resources.

We also explored options that would have allowed the device to operate indepently of the user's home internet service, such as local connections like Bluetooth or Zigbee, or cellular connectivity. Ultimately, we decided that this method would come at too great a financial cost to the user for it to be attractive to them. Furthermore, internet connectivity is the implementation strategy with which our team is the most familiar.

The approach we eventually chose to implement utilizes a Raspberry Pi for the hardware computing module that communicates with the circuit boards through a mounted ADC. The printed circuit boards continuously read energy input and output and pass that data through the ADC to the Raspberry Pi. The data is then passed to the online servers to be used for buying and selling. The Raspberry Pi has built in capabilities which provide the needed support for hardware communication via SPI transactions and software communication via server posting. In the prototype form, the Raspberry Pi and PCBs non-intrusively connect to the existing power buses, serving as a smart meter that can coexist with the existing traditional meter. Power usage is tracked and data is transmitted to the server.

In the end, we chose the design approach that would be the most intuitive to implement rather than one that was focused on being as cost-effective or market-ready as possible.

3.2 Overall Design

After considering the above approaches, we determined that the best plan of action was to implement our marketplace through a traditional marketplace approach and use a Raspberry Pi approach for our smart meter, while adding PCBs for some additional data filtering and processing. There is a PCB dedicated to converting the mains voltage into a source useable for the sensitive components of our smart meter and another PCB dedicated to measuring and filtering the data in the mains into a form readable by our Raspberry Pi. In Figure 2, we visualize an overview of our system. Each user/property uses the smart power meter to monitor the flow of power to the property and from any power-generating devices, such as solar panels or wind turbines, to their power company. Each smart meter will be linked to a user's account via the

web application. A user will create an account via the web application and link their device ID through the settings tab. A user for any given property may access the marketplace and view analytics in real-time about their energy consumption through the web or mobile application.

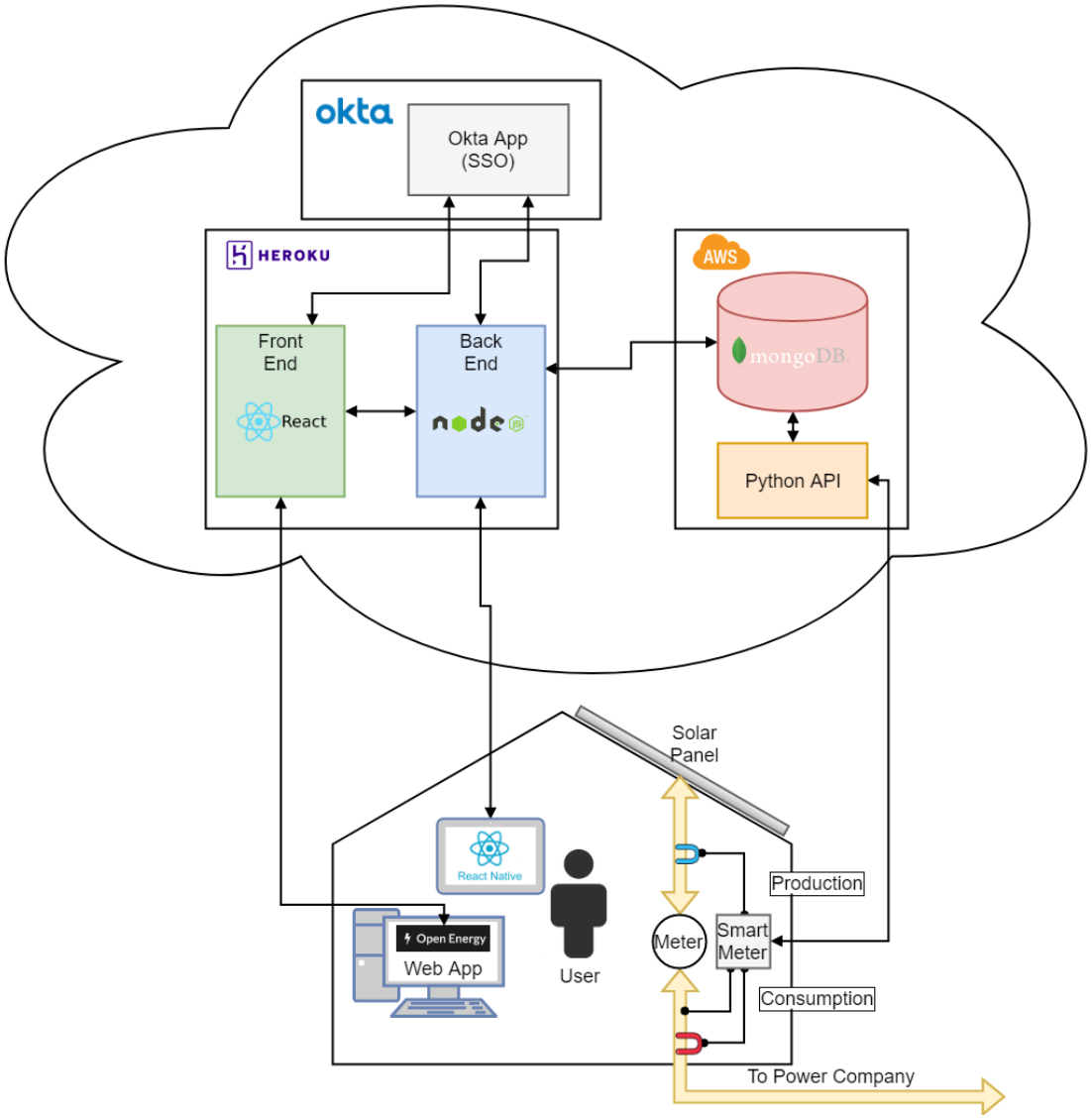


Figure 2: High level system diagram

Figure 3 shows our DevOps strategy. We have a production environment of the application that all users will see and use, individual environments for each of the software developers on the team to work in, and a staging environment for testing changes before pushing out to the production environment. The user information will be stored in MongoDB databases in Amazon Web Services (AWS), and the web application is deployed on Heroku.

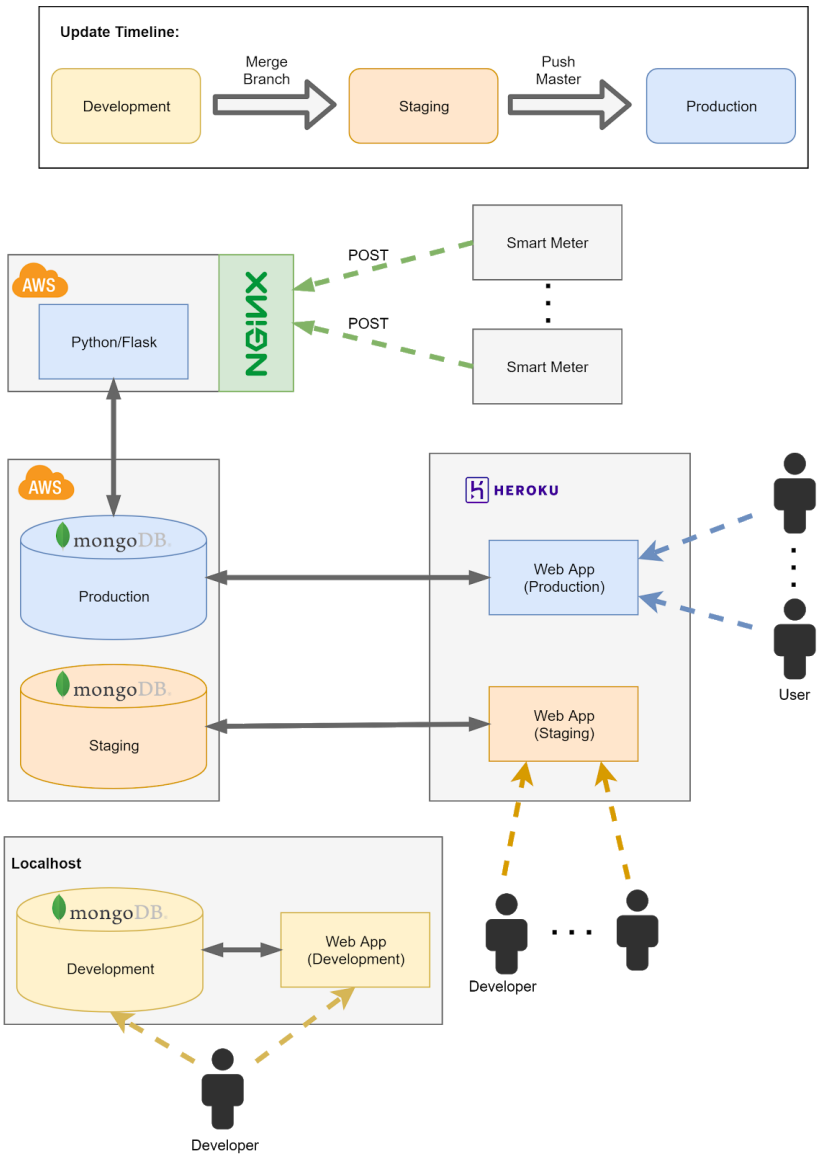


Figure 3: DevOps diagram illustrating the design and development process

IoT Smart Meter

Based on research about products that are readily available for a reasonable cost, we determined that it was in our best interest to begin our development of the smart meter on a module that already has basic functionality like a Raspberry Pi. The basic block diagram describing the major components and functionality of the IoT smart meter is shown in Figure 4.

The Raspberry Pi was chosen over a standard Arduino because the Raspberry Pi is a more powerful device and it is easier to program because it does not require as deep of a knowledge of embedded system programming as an Arduino. Despite the price and power consumption tradeoffs, we see Raspberry Pi as the best option because it is easier to debug and troubleshoot in the early stages of prototyping. The Raspberry Pi also has a built-in Ethernet port which can make networking simpler.

For the current sensor, we are using the MASTECH MS3302 AC current clamps (with a voltage and current range within our required operating conditions) feeding into a signal processing circuit and then passing that information to the Raspberry Pi for networking. We decided on this method of implementation because of its relative simplicity and the fact that it allowed us to compartmentalize the work for different groups of team members. We acknowledge the fact that a long-term solution would probably have an integrated current reading mechanism with higher accuracy, but at this stage of design, we are aiming for a functional proof of concept.

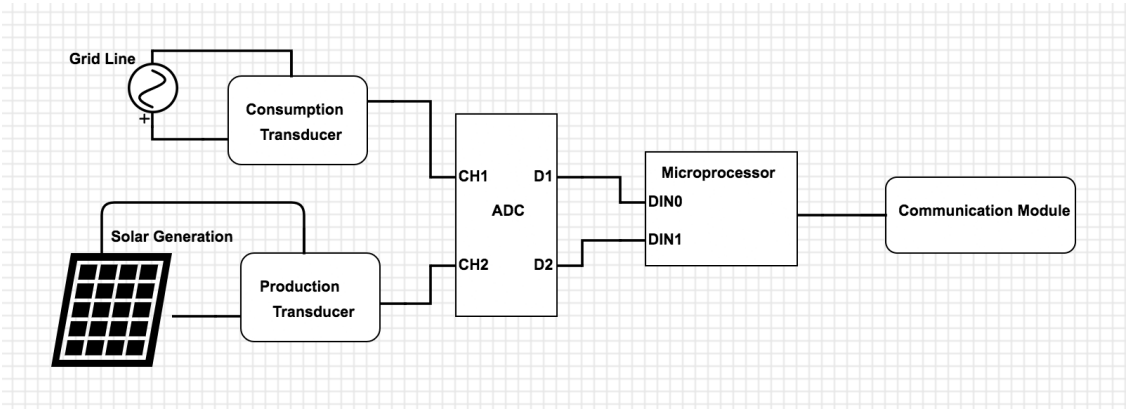


Figure 4: A top-level block diagram of the smart meter basic functionality

Web Application and API

All relevant user data, including account information, energy usage statistics, and transaction details, is stored in a MongoDB database on AWS. Our web application interacts with our MongoDB database via an API. The app also interacts with Okta for account creation and creating user tokens on login. A component diagram of the web application can be seen below in Figure 5.

For our software design, we are implementing a web application using the MERN (Mongo, Express, React, Node) stack, an open source end-to-end dynamic application framework. This MERN stack includes a MongoDB database for data storage and manipulation. Express is a back end web application framework that runs on top of NodeJS. React is a JavaScript library that allows us to build a dynamic user interface for the web application and React Native for the mobile application. Both interfaces render based on the current state of the application. The use of React and React Native allows for sharing logic across platforms as well as the back-end. Finally, NodeJS is an open source server framework. The app is deployed using Heroku, which makes running our NodeJS app fairly straightforward.

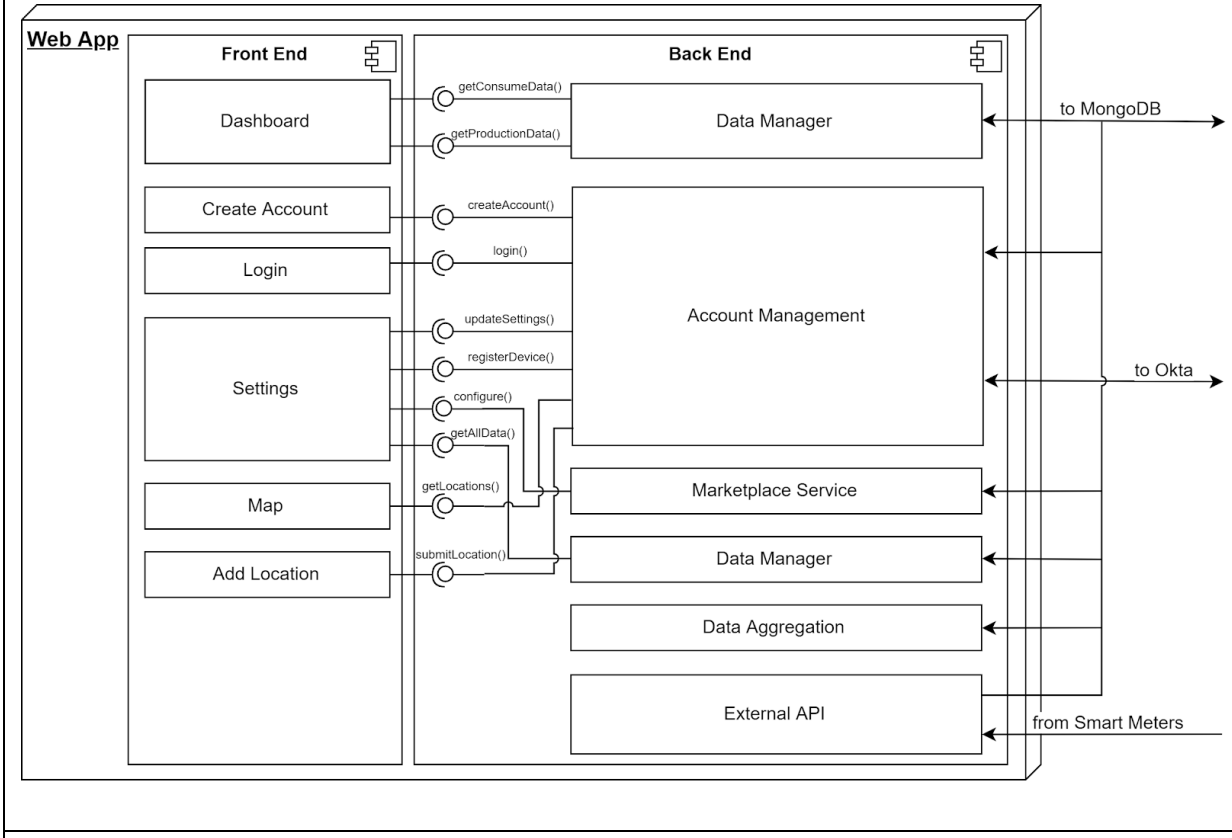


Figure 5: A component diagram of the web application

3.3 Challenges

1. Figuring out how to correctly price the energy to have consumers and producers enter and exit the market for a steady-state goal.

2. Although the MERN stack is robust, there is a relatively small amount of documentation for this software stack, since it is fairly new. This means that debugging issues takes more effort than if we were using less modern, but more tested, technologies.

3. One of the largest concerns is preventing users from spoofing energy production data. This is a large concern and will take a lot of future work. Some options include:

a. Document user's max production capacity, though this does not prevent them from overstating production while staying below their max production capabilities

b. Compare production relative to other geographically close producers with similar/identical production sources, while also factoring outside conditions

c. Implement some sort of anti-tampering monitoring on the meter

4 Estimated Resources and Project Timeline

In this section, we discuss the monetary, time, and personnel resources required to complete the project, including our timeline of task completion.

4.1 Personnel Effort Requirements

Task	Team	Effort Required	Reference/Explanation
Program Smart Meter Display	Hardware	Low	The libraries available for Raspberry Pi devices should make it fairly simple to take in input and display output to the user on the meter.
Coordinate Interfacing with Web App	Hardware	Medium	This step will require effective communication with the software team on what data they need to receive and what form it will come in.
Test Initial Proof of Concept Prototype	Hardware	Medium	The focus of this step will be to ensure that this version of the system has all of the basic desired capabilities that are needed for an early implementation of the project. Essentially we will need to make sure that we are able to reliably acquire and transmit data.
Design PCB/Embedded Board	Hardware	High	To implement the hardware required for this project from scratch, we would need to acquire various individual components and connect them on a PCB. In addition, we would have to write our own drivers for this board.

Order Board	Hardware	Low	Once we have the design completed, ordering the board will just be a matter of determining the best option to have it fabricated
Test and Compare with Raspberry Pi Version	Hardware	Medium	This step should just be a matter of determining which aspects of performance we will give the most weight and comparing the two implementations, including the effort and cost that went into creating them.
Marketplace: Init Transaction	Software	Low	Transactions are a standard procedure for applications to perform, and initializing a request and posting it to the marketplace should be straightforward
Marketplace: Accept Transaction	Software	Medium	We will need to do some extra programming to deal with the timing of when accounts receive payment, such as using a holding pot until the transaction is complete
Smart Meter API: Power Transaction Signal	Software	High	Requires blockchain/smart meter interaction, which will require collaboration between teams and probably more testing, since this is essential to our project
Web App: Create Account/Register Smart Meter	Software	Medium	We will tie each account to an Ethereum account, so account creation will be handled elsewhere; however, we will need to handle linking an account to a smart meter
Web App: Login	Software	Low	We will just need to pass login through Ethereum, so there should be little programming required for this step
Web App: View Transaction History	Software	Low	This will simply require some queries to the MongoDB database, followed by displaying that data in a meaningful way

Web App: User Analytics	Software	Medium	Retrieving the information that we need will be easy, as will displaying the raw information to the dashboard. Depending on how complex we want our analytics to be, there is potential for more intensive algorithms to write and program
Web App: Automated Transaction Matching	Software	High	This task will probably involve some form of artificial intelligence, or a very efficient matching algorithm. Optimizing our strategy will be key, since this process may run for hundreds of users at a time, if our market caught on
Perform End-to-End Acceptance Tests	Software	Medium	We will want to be thorough with our tests, and will probably need to do quite a bit of debugging and refining, but if our designs are well thought out, and each previous task goes well, we should not have any major changes to make

4.2 Other Resource Requirements

The main parts required for this project are on the hardware (smart meter) side, as this is the physically tangible part of the project. For the early prototype, we needed a Raspberry Pi module and set of current clamps to acquire data. Along with an analog to digital converter and a few passive components (wires, resistors, capacitors), these are the baseline components required to obtain the fundamental data. This relies on the assumption that the voltage is reasonably steady, so a current reading can give us accurate power consumption data.

For the full final implementation of the smart meter, there is a large number of components required. Links to our two parts orders are provided below.

First parts order: 2/28/18

Second parts order: 4/27/18

While the number of components is large, most of them are inexpensive so the total price of the meter can compete with existing smart meter solutions (see *4.3 Financial Requirements*)

In addition to the components, we also needed to have PCBs fabricated. This required the use of the Eagle PCB design software so we could send the needed files to the PCB fabricator.

The software team will require the React, Node, and Express libraries to run local unit tests and generate local development environments for the web application, as well as an AWS instance to run the software backend.

4.3 Financial Requirements

Because the purchasing of hardware for this project was done through the Electronics Technology Group, we do not have exact numbers regarding the prices of our various components, so some of the following data is based on estimation. The main components that had to be purchased for our project are the following.

MASTECH MS3302 AC Current 0.1A-400A Clamp Meter Transducer True RMS

at $17 x 2 = **$34**

Justification: To measure the current (and therefore power) on the production and consumption lines

Raspberry Pi 3

at $35 x 1 = **$35**

Justification: To be able to process and post our acquired data to the server. . Import our logic libraries and subsequent code base.

PCB + miscellaneous components

Power board	at $15 x 1 =	**$15**
Data acquisition board	at $50 x 1=	**$50**
Components	at $40 x 1 =	**$40**
Total:		**$105**

Justification: To carry out our desired functionality of being able to power the module from the line and amplifying and filtering the data

Enclosure

At $10 x 1 = **$10**

Justification: To contain all of the hardware components of the meter and keep them protected from damage, tampering, or general wear and tear which could cause connections to go bad

Total Cost: $184

It is worth noting that these price estimations are based on the quantities in which we bought the listed items. For larger-scale production of the meter, high-voluming purchasing could cut down the per-unit cost drastically.

4.4 Project Timeline

A large portion of the success of our project was centered around adhering to our timeline. Although at the time of creation of the timeline many of our tasks were in the preliminary stage, it was vital for us to use this tool to maintain steady progress. An overview of our major tasks and milestones is shown below, while the more detailed version is shown in the attached Gantt charts for both first (EE/CprE/SE 491) and second semester (EE/CprE/SE 492) (Figures 6 and 7, respectively).

4.4.1 First Semester

Class Deliverables

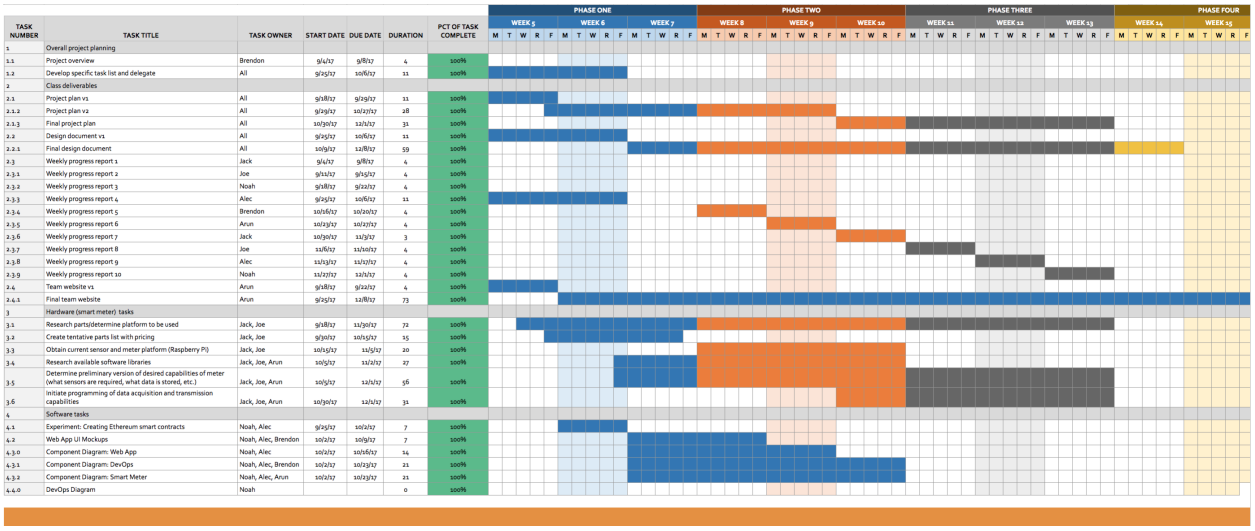
1. Project Plan
2. Design Document
3. Team Website

Hardware Team (Jack, Joe, Arun)

1. Research parts and determine platform to be used
2. Create tentative parts list with pricing
3. Obtain current sensor and meter platform (Raspberry Pi)
4. Research available software libraries for Raspberry Pi module we are using
5. Determine preliminary version of desired capabilities of meter (sensors required, data stored, etc.)
6. Initiate programming of data acquisition and transmission capabilities

Software Team (Brendon, Noah, Alec, Arun)

1. Experiment with Ethereum and creating smart contracts
2. Experiment with MERN stack
3. Web app user interface mockups
4. Component diagram for web app
5. Component diagram for smart meter API
6. Basic web app development environment setup



TASK NUMBER	TASK TITLE	TASK OWNER	START DATE	DUE DATE	DURATION	PCT OF TASK COMPLETE
1	Overall project planning					
1.1	Project overview	Brendon	9/4/17	9/8/17	4	100%
1.2	Develop specific task list and delegate	All	9/25/17	10/6/17	11	100%
2	Class deliverables					
2.1	Project plan v1	All	9/18/17	9/29/17	11	100%
2.1.2	Project plan v2	All	9/29/17	10/27/17	28	100%
2.1.3	Final project plan	All	10/30/17	12/1/17	31	100%
2.2	Design document v1	All	9/25/17	10/6/17	11	100%
2.2.1	Final design document	All	10/9/17	12/8/17	59	100%
2.3	Weekly progress report 1	Jack	9/4/17	9/8/17	4	100%
2.3.1	Weekly progress report 2	Joe	9/11/17	9/15/17	4	100%
2.3.2	Weekly progress report 3	Noah	9/18/17	9/22/17	4	100%
2.3.3	Weekly progress report 4	Alec	9/25/17	10/6/17	11	100%
2.3.4	Weekly progress report 5	Brendon	10/16/17	10/20/17	4	100%
2.3.5	Weekly progress report 6	Arun	10/23/17	10/27/17	4	100%
2.3.6	Weekly progress report 7	Jack	10/30/17	11/3/17	3	100%
2.3.7	Weekly progress report 8	Joe	11/6/17	11/10/17	4	100%
2.3.8	Weekly progress report 9	Alec	11/13/17	11/17/17	4	100%
2.3.9	Weekly progress report 10	Noah	11/27/17	12/1/17	4	100%
2.4	Team website v1	Arun	9/18/17	9/22/17	4	100%
2.4.1	Final team website	Arun	9/25/17	12/8/17	73	100%
3	Hardware (smart meter) tasks					
3.1	Research parts/determine platform to be used	Jack, Joe	9/18/17	11/30/17	72	100%
3.2	Create tentative parts list with pricing	Jack, Joe	9/30/17	10/15/17	15	100%
3.3	Obtain current sensor and meter platform (Raspberry Pi)	Jack, Joe	10/15/17	11/3/17	20	100%
3.4	Research available software libraries	Jack, Joe, Arun	10/3/17	11/2/17	27	100%
3.5	Determine preliminary version of desired capabilities of meter (what sensors are required, what data is stored, etc.)	Jack, Joe, Arun	10/3/17	12/1/17	56	100%
3.6	Initiate programming of data acquisition and transmission capabilities	Jack, Joe, Arun	10/30/17	12/1/17	31	100%
4	Software tasks					
4.1	Experiment: Creating Ethereum smart contracts	Noah, Alec	9/25/17	10/2/17	7	100%
4.2	Web App UI Mockups	Noah, Alec, Brendon	10/2/17	10/9/17	7	100%
4.3.0	Component Diagram: Web App	Noah, Alec	10/2/17	10/16/17	14	100%
4.3.1	Component Diagram: DevOps	Noah, Alec, Brendon	10/2/17	10/23/17	21	100%
4.3.2	Component Diagram: Smart Meter	Noah, Alec, Arun	10/2/17	10/23/17	21	100%
4.4.0	DevOps Diagram	Noah			0	100%

Figure 6: Snapshot of the semester 1 Gantt chart

4.4.2 Second Semester

Class Deliverables

1. Working prototype smart meter
2. Basic functional web application
3. Working energy marketplace implementation
4. All components successfully interacting

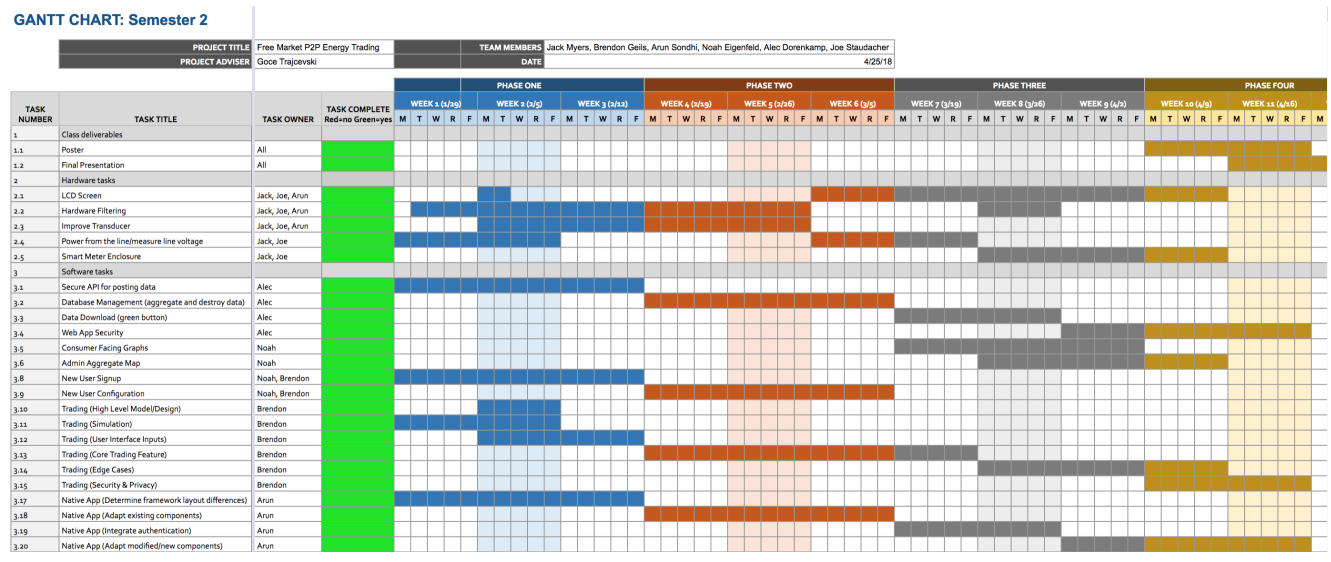
Hardware Team (Jack, Joe, Arun)

1. Program user interface
2. Coordinate interfacing with web app
3. Test initial proof of concept prototype
4. Design PCB/embedded board
5. Order board
6. Test and compare with Raspberry Pi version
7. Program data acquisition and transmission capabilities

Software Team (Brendon, Noah, Alec, Arun)

1. Base web app functional
2. API components for communicating between web app and smart meter
3. Create account functionality through web app
4. Login functionality through web app
5. View transaction history functionality through web app
6. Initiate and accept transaction functionality through web app
7. View energy consumption analytics

8. User smart meter data is automatically aggregated
9. (Stretch goal) Automated transaction matching



GANTT CHART: Semester 2

PROJECT TITLE	Free Market P2P Energy Trading			TEAM MEMBERS	Jack Myers, Brendon Geils, Arun Sondhi, Noah Eigenfeld, Alec Dorenkamp, Joe Staudacher		
PROJECT ADVISER	Goce Trajcevski			DATE	4/25/18		

				PHASE ONE													PHASE TWO													PHASE THREE													PHASE FOUR												
TASK NUMBER	TASK TITLE	TASK OWNER	TASK COMPLETE Red=no Green=yes	WEEK 1 (1/29)					WEEK 2 (2/5)					WEEK 3 (2/12)					WEEK 4 (2/19)					WEEK 5 (2/26)					WEEK 6 (3/5)					WEEK 7 (3/19)					WEEK 8 (3/26)					WEEK 9 (4/2)					WEEK 10 (4/9)					WEEK 11 (4/16)				
				M	T	W	R	F	M	T	W	R	F	M	T	W	R	F	M	T	W	R	F	M	T	W	R	F	M	T	W	R	F	M	T	W	R	F	M	T	W	R	F	M	T	W	R	F	M	T	W	R	F	M	T	W	R	F	M
1	Class deliverables																																																					
1.1	Poster	All																																																			
1.2	Final Presentation	All																																																		
2	Hardware tasks																																																				
2.1	LCD Screen	Jack, Joe, Arun																																																			
2.2	Hardware Filtering	Jack, Joe, Arun																																																		
2.3	Improve Transducer	Jack, Joe, Arun																																																		
2.4	Power from the line/measure line voltage	Jack, Joe																																																		
2.5	Smart Meter Enclosure	Jack, Joe																																																		
3	Software tasks																																																				
3.1	Secure API for posting data	Alec																																																		
3.2	Database Management (aggregate and destroy data)	Alec																																																		
3.3	Data Download (green button)	Alec																																																		
3.4	Web App Security	Alec																																																		
3.5	Consumer Facing Graphs	Noah																																																		
3.6	Admin Aggregate Map	Noah																																																		
3.8	New User Signup	Noah, Brendon																																																		
3.9	New User Configuration	Noah, Brendon																																																		
3.10	Trading (High Level Model/Design)	Brendon																																																		
3.11	Trading (Simulation)	Brendon																																																		
3.12	Trading (User Interface Inputs)	Brendon																																																		
3.13	Trading (Core Trading Feature)	Brendon																																																		
3.14	Trading (Edge Cases)	Brendon																																																		
3.15	Trading (Security & Privacy)	Brendon																																																		
3.17	Native App (Determine framework layout differences)	Arun																																																		
3.18	Native App (Adapt existing components)	Arun																																																		
3.19	Native App (Integrate authentication)	Arun																																																		
3.20	Native App (Adapt modified/new components)	Arun																																																		

Figure 7: Snapshot of semester 2 Gantt chart

[Full Project Timeline]

5 Detailed System Design

We now present a detailed design of our software and hardware systems.

5.1 Software

Figure 8, below, displays a detailed component diagram for all of the software operating within our system. The diagram is divided into several large components: the Heroku deployment (which is divided into a sub-component for the front-end client and another for the back-end application), mobile app, AWS server, and the smart meter's logic module. Within each of these components can be seen the sub-components, and the methods in which the sub-components interact with each other.

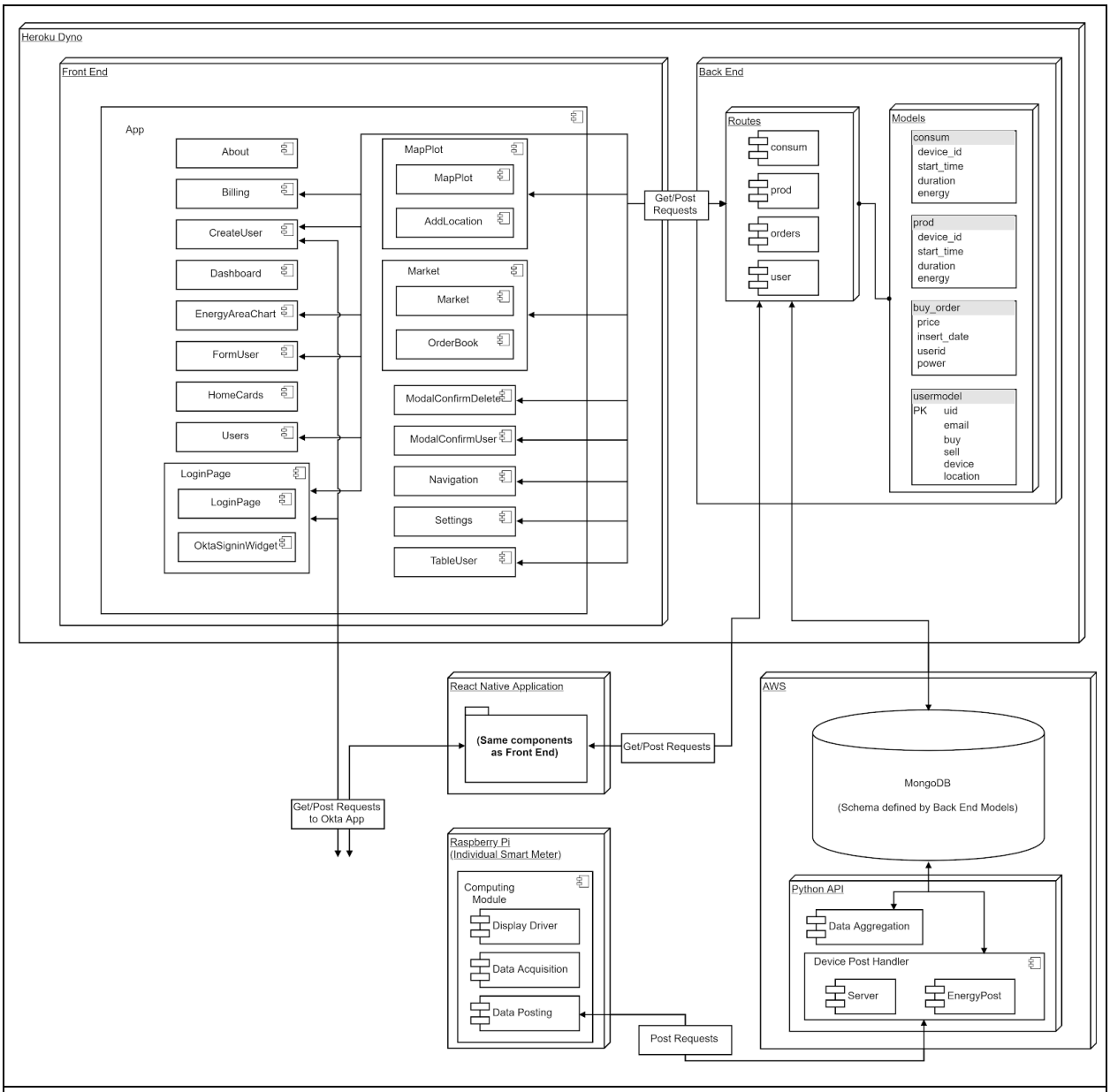


Figure 8: Detailed diagram of software components on the smart meter, mobile application, and web application

5.1.1 Web App

I/O Specifications

Web application output and input, categorized as user input and non-user input.

User input:

- User account information, including login credentials, smart meter information, buying and selling price thresholds, location information

Non-User input:

- Smart meter power production and consumption data

Output:

- Energy consumption and production charts, smart meter location map, downloadable data in csv, feedback on change of user settings

User Interface

The interface was designed with accessibility and cleanliness in mind. All information can be reached within one to two clicks and is easily seen and understood without the need for large sections of explanatory text. As shown in figure 9 below, the UI is minimalistic in style without lacking in form or function. The live web application can be viewed at: www.myopenenergy.com

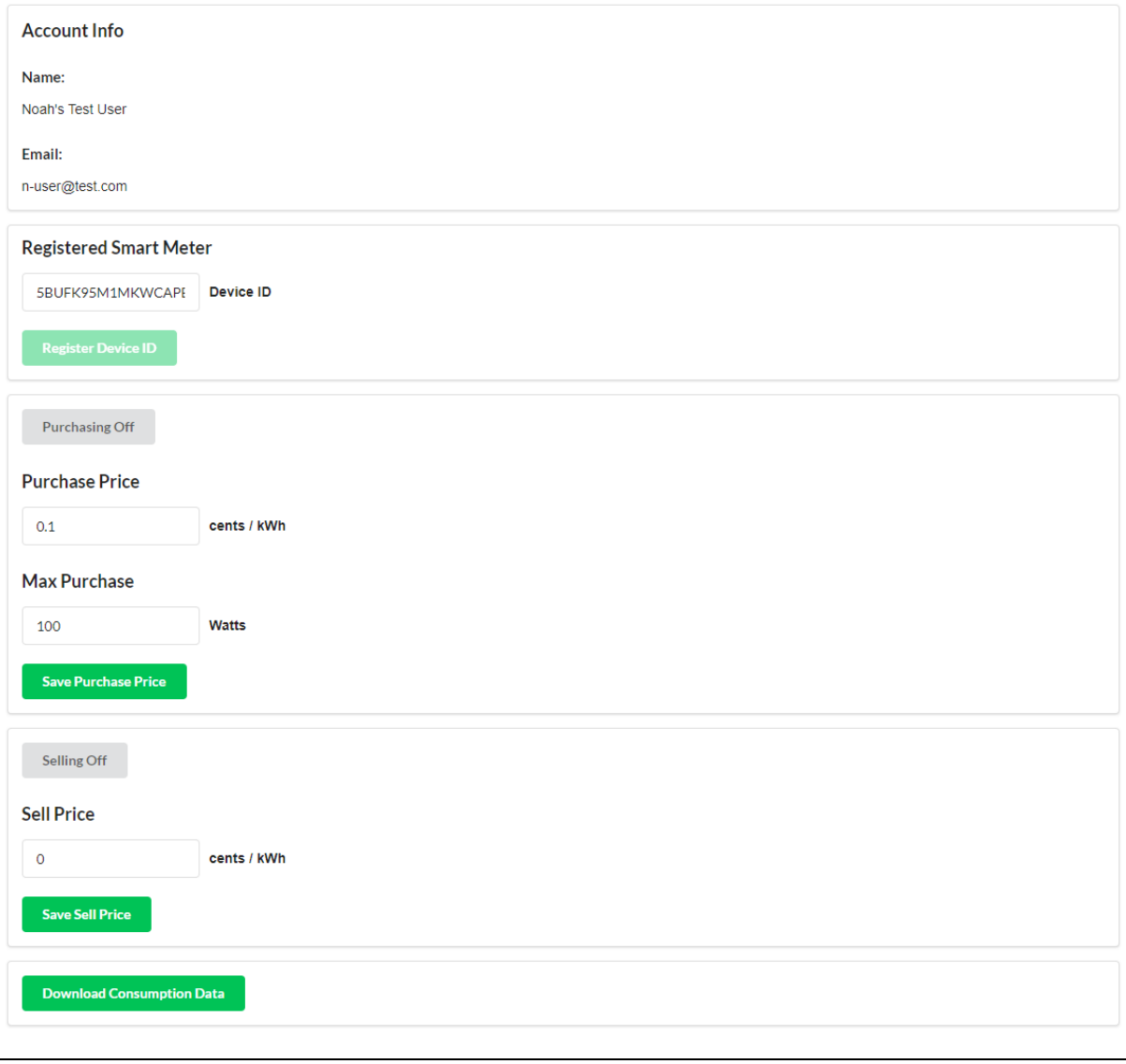


Figure 9: A view of the settings page, from which users can configure their accounts

Account Management

The web application facilitates a user's account registration and configuration, so that the user may view the data associated with their device. To aid in account management, the web application interacts with Okta, which stores a user's name, email address, and unique identifier, and creates an authentication token upon sign-in to keep track of a user during the current session. Upon successful account creation, the unique identifier from Okta is stored in a MongoDB document for referencing device data and account settings associated with that user.

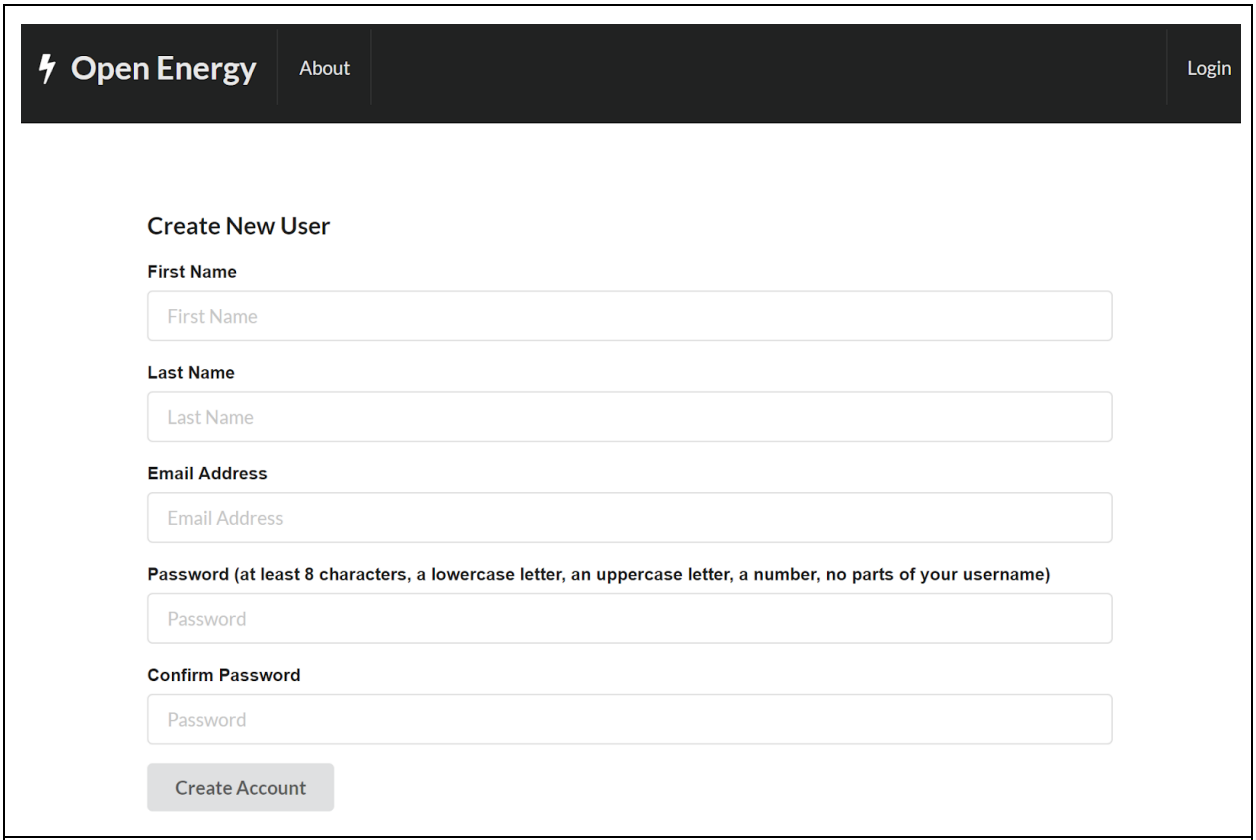


Figure 10: The account creation page on the web application

Once a user has created an account, they can register a device via the device ID. Though we only created one prototype for the smart meter, our goal is that each smart meter would be configured with its own device ID, either shown to the user through the smart meter's display, or through accompanying literature. To register a device and start viewing one's data, a user need only register the device ID through the settings page. If a device ID is already registered to an account, it cannot be registered again; however, registered devices can be updated at any point. At this time, only one device ID may be registered to an account, though we expect this to be an expansion point in the future.

The settings page also allows users to modify their purchasing and selling configurations. The usage of these values is described further down, within the *Marketplace* section.

Rather than navigating our web application to view all of their data, some users may wish to format and analyze their usage data directly. For this purpose, users may download their power consumption data from the settings page. A button at the bottom of the page will download a user's consumption data as a .csv file.

Data Display

Users may enter their device's location and view the locations of other users' devices through the map page. At this time, one's own device is displayed and labeled, and all other users' devices are listed as anonymous. In the future, once an admin construct is added, specific device locations will be limited to admin views only, with standard users receiving a masked map, grouping locations by city or region.

Location functionality interacts with two APIs: Mapbox GL JS and Google Maps JavaScript API. Mapbox GL JS implements the overall device location map found in the map page, and Google Maps JavaScript API implements the searchable map found in the add-location page (reachable through the map page). The Google Maps library is accessed through the google-maps-react module, which supplies a React JS wrapper for accessing Google Maps interfaces as React components.

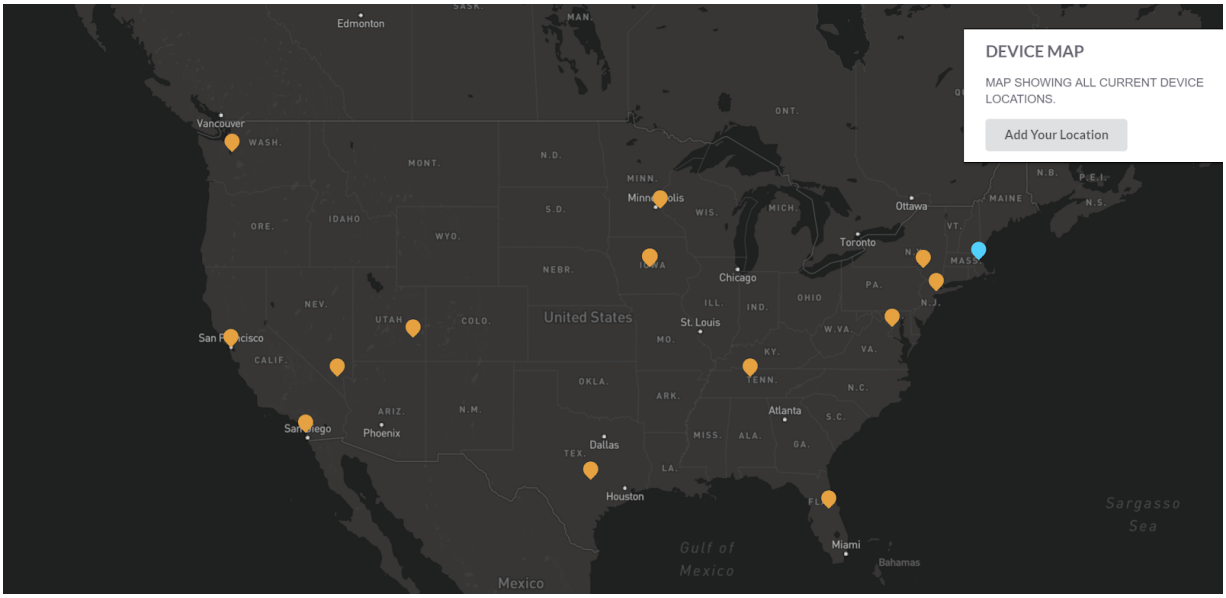


Figure 11: A view of the web application map of device locations

For users who wish to view current information about their consumption usage, we implemented a simple graph on the dashboard which displays a real-time readout of their power consumption. This graph updates every three seconds, as do the energy statistics displayed above it. These statistics show the chart average, real-time usage, seven-day, month, and year averages. At the time of this report, energy production does not have a real-time graph implemented; however, this is a simple matter to expand due to the similar storage of production data.

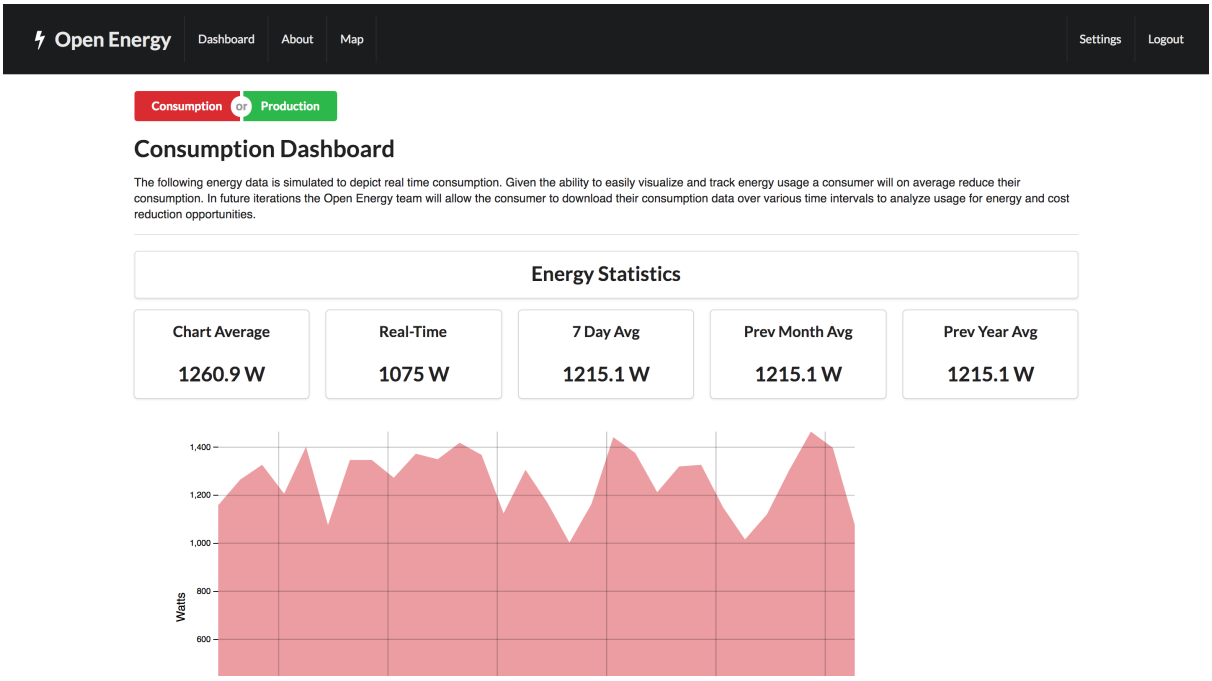


Figure 12: A view of the web application user interface

5.1.2 Mobile App

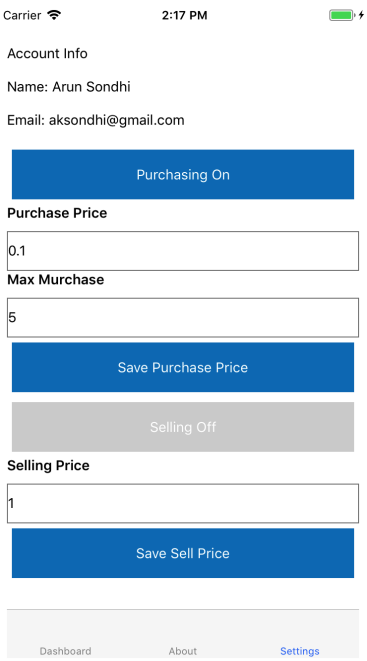


Figure 13: A view of the mobile application user interface

Account Management

The mobile application is very similar to the web application. It also interacts with Okta to verify user interactions and authenticate upon sign-in. A user is not able to sign up or add a device via the mobile application. They are able to manage their purchasing and selling configurations through the settings tab.

Data Display

Users are able to view their consumption and usage via the dashboard in real-time. The data updates every three seconds and shows real-time usage, seven-day, month and year averages.

5.1.3 Backend

Data Management

The backend of the software system is Ubuntu 16 virtual machine on AWS (Amazon Web Services). This VM includes a MongoDB instance that holds all the important client data, including basic account information (not including user credentials, which is managed using Okta), user settings, and device data. The web application sends and retrieves information from this database using get and post requests secured via SSL.

Clients' smart meters use secure post requests to send information to the database. This posted information is authenticated from both directions using an API key to authenticate the smart meter and a public-private key combination to ensure that only the server can read posted data.

Data aggregation is performed every three minutes by a python CronJob to reduce smart meter consumption and production posts (3 second intervals) into a minimum of 1 hour and maximum of 2 hours of 3 min chunks of data, between 7-8 days of hourly data, and older data into daily chunks, while always maintaining at least 1 minute of 3 second data to allow the user to monitor real time statistics. This scheduling is fairly arbitrary, and the code was made in a very modular way to allow easy manipulation of these time boundaries in the future. Currently, the size of the data stored for each smart meter is ~250 records to hold data from the last 8 days + 1 record for every day the smart meter was in use before then.

5.1.4 Marketplace

Key terms

Producer – entity that puts energy into the grid and is signed up with Open Energy to sell at the OE (open energy) price

Consumer – entity that agrees to purchase energy from the grid at the OE (open energy) price

OE – Open Energy

OEIP – Open Energy Internal Price

OEP – Open Energy Price

High Level Overview

Producers and consumers enter and exit the market purchasing and selling at the OE price. The OE price fluctuates given the market forces.

Transaction Cost

The cost for each transaction will vary based on the average distance between the producers and consumers. In our MVP design we will have a fixed transaction cost. The code design should accommodate a future where the transaction cost fluctuates.

Open Energy Internal Price

The open energy internal price (OIEP) is set by OE to ensure the market is stable given the number of producers and consumers in the market. The OEP is a value used internally to calculate OEP. The following scenarios effects the OEIP:

1. # of producers rises
    a. OEP decreases
2. # of producers falls

a. OEP increases

3. # of consumers rises

a. OEP increases

4. # of consumers falls

a. OEP decreases

Open Energy Price – OEP

The following formula defines the OEP.

OEP = OEIP + Transaction Cost / 2 + OE Fee

Purchase Energy

To purchase energy, the user must do the following:

1. Open an account with OE at http://www.myopenenergy.com/

a. Have an admin sign into Okta and create the account

2. Enable purchasing in the settings page of OE www.myopenenergy.com/settings

Marketplace System Function

Every five seconds the system should take a poll of all the users that are willing to purchase energy at or below the OEP and all users that are willing to sell energy at or above the OEP. These users are the active set. The active producer set is aggregated to find the total energy provided during the five second interval.

The active consumer set is then aggregated to find the amount of maximum consumed energy. The total energy from the active producer set is then distributed among the active consumer set (see distributeConsumer).

Functions

distributeConsumer(float total_energy, array active_consumer_set)
The use of the distribute_consumer() function is to take in the total amount of energy from the active producer set and the active consumer set, process the set and then return any left-over energy (see handleExtraEnergy).

The processing will first find the average distributed power: total_energy / length(active_consumer_set). This will be distributed by charging each active consumer up to the maximum they have allocated at the current OEP. If the maximum is reached the next consumer with a high maximum will have the energy rolled over.

handleExtraEnergy(float extra_energy)
This function will sell all extra energy back to the grid at the net-metering rate. Documenting accordingly.

5.2 Hardware

The high level design of the hardware system is shown in Figure 14, composed of three main modules. These modules are the computing module, the data acquisition circuit board, and the circuit board used to power the meter. The data acquisition board and the device power board were the most design-intensive modules of the hardware system as they were fabricated printed circuit boards (PCBs) designed in Autodesk Eagle. Each PCB went through two iterations of testing and design. The computing module was the bridge between the hardware and software design processes.

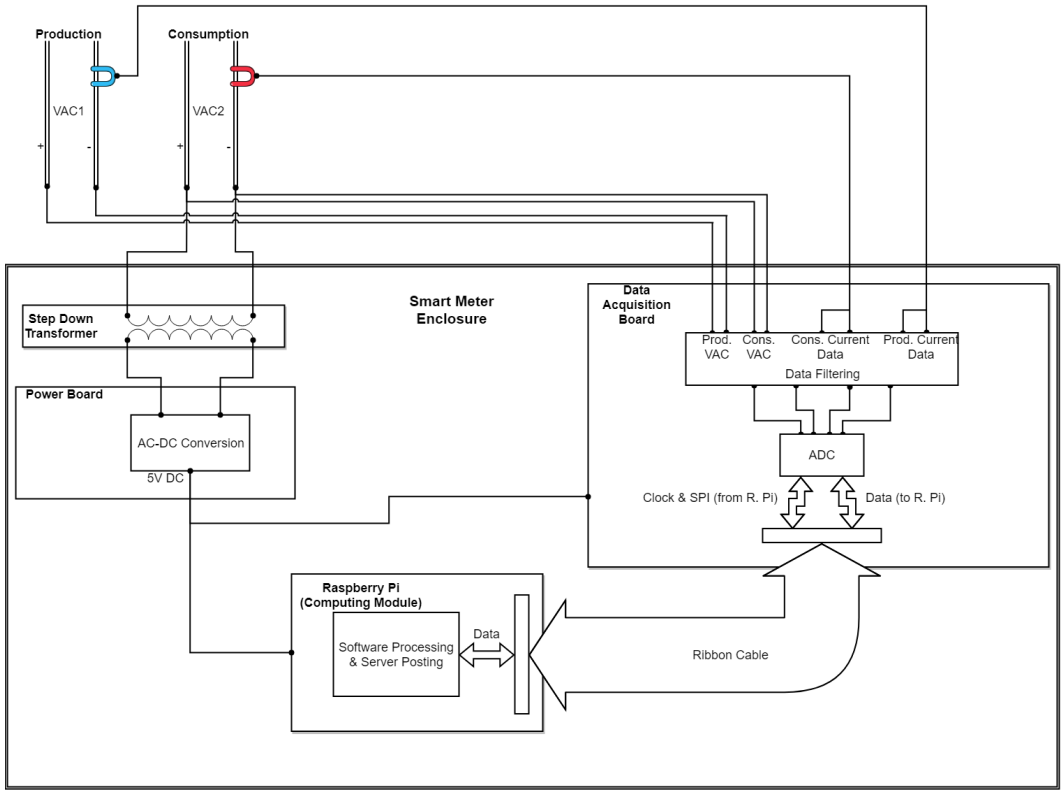


Figure 14: A high-level overview of the components within the smart meter

5.2.1 Data Acquisition Board

The purpose of the data acquisition circuit board is to take in and filter data from two sources: the output from the current clamps used to read the total current draw, and the mains voltage itself. The output from the current clamps is a small AC voltage (on the order of roughly 10-500 mV) that corresponds directly to the current running through the mains. This voltage signal is then filtered and smoothed to form that could be read by an ADC and transferred to the Raspberry Pi for computation. The schematic of this filtering circuit is shown below:

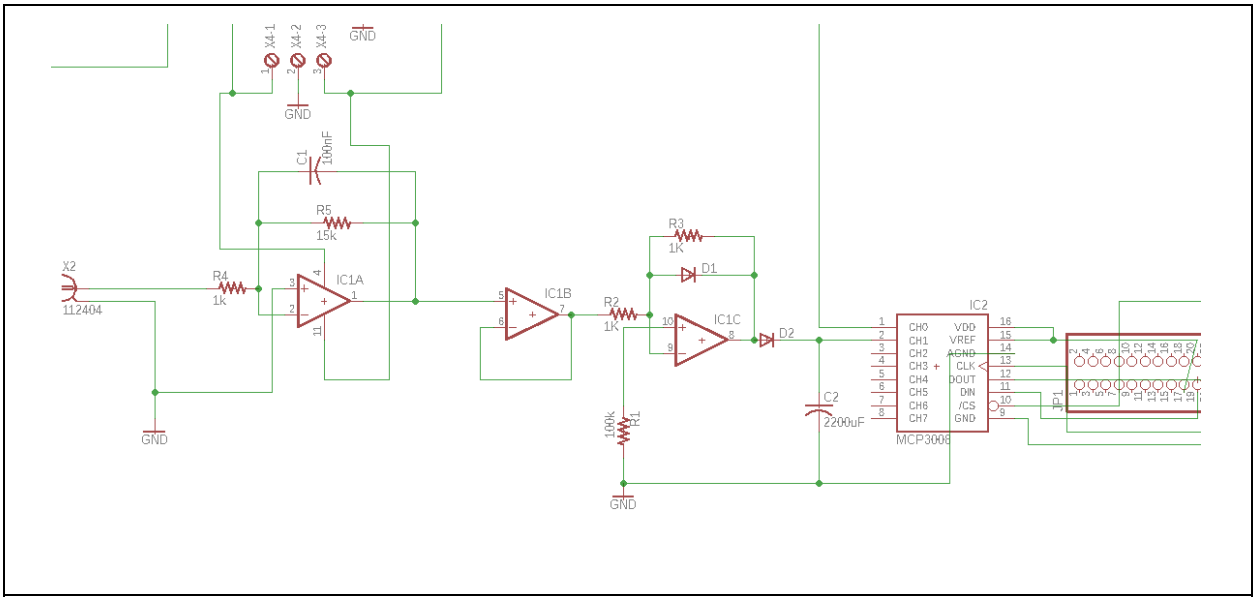


Figure 15: Schematic of the current reading circuit on the acquisition board

This schematic shows the input of the current clamps at the terminal on the far left and and the input from the power board with the terminals at the top. For the output of the circuit, it is connected to terminal 2 of the ADC, which leads to header pins connected to a ribbon cable that transmits the data over SPI to the Raspberry Pi (computing module).

The first stage of the circuit is an active low-pass filter, with a gain of 15 for signals in the passband. The cutoff frequency is 106.1 Hz, which allows us to filter high-frequency noise that is well above the expected signal output frequency of 60 Hz. The goal of this stage is to filter out any noise present in the signals from the current clamps, as well as amplify this small signal to a range that can be more accurately read by the ADC, which has a reference voltage of 3.3V. The second stage is a buffer that isolates the first stage from the third stage. The final stage is a precision rectifier with a smoothing capacitor that takes the AC signal and converts it to DC. The goal of this is to provide a consistent signal to the ADC so reading the value at any given time would be indicative of the present current flow.

As referenced earlier, the second part of the data acquisition board is dedicated to reading and filtering the mains voltage. The main component of this circuit is the isolation amplifier. It is used to create a barrier between the mains voltage and all of the other components so that they are protected from high voltages. The schematic diagram of this circuit is shown below.

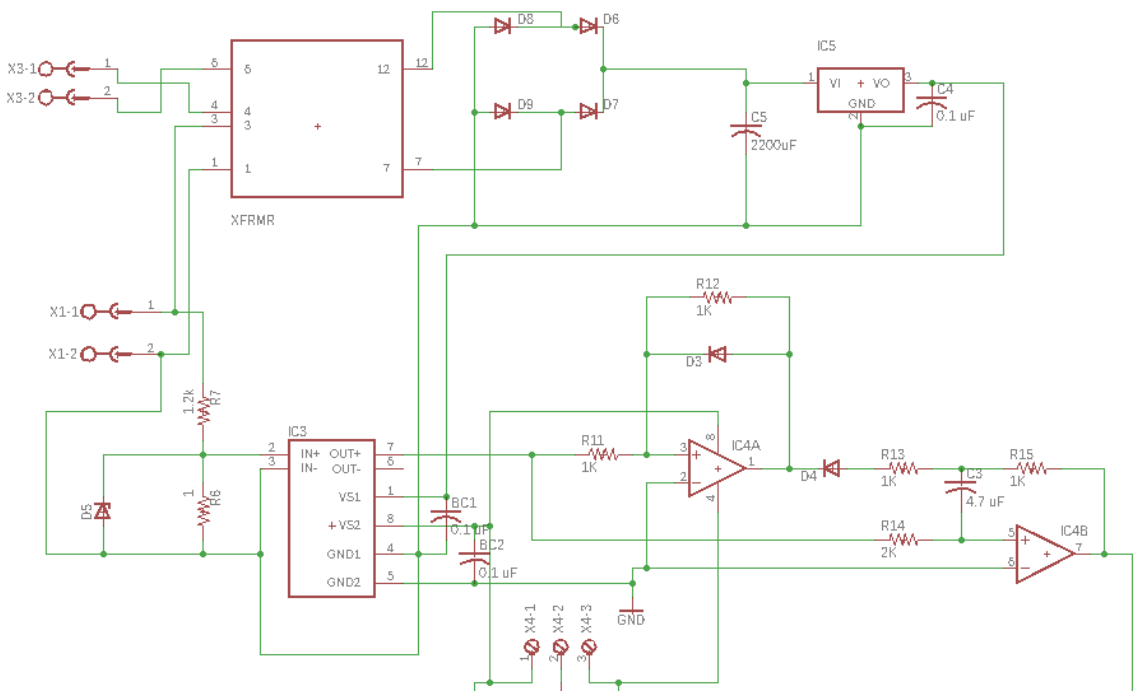


Figure 16: Schematic of the voltage reading circuit on the acquisition board

The op-amps and the input side of the isolation amplifier in this circuit are powered off of the same power source as the other circuit, the power board, but the isolation amp presented a problem in that it required two separate power supplies. While the supply from the power board is usable for the input side of the amp, we had to figure out a way to get another isolated power supply for the output. This required the implementation of another transformer and rectifier circuit separate from the one on the power board. The two sets of terminals on the left side of the schematic represent the voltage input from the mains with the top set going to the aforementioned transformer, and the bottom set going into a voltage divider before being used as the input for the isolation amplifier. The output of the isolation amplifier leads to a filtering circuit and then goes to the input terminal 1 of the ADC. The final layout of the entire data acquisition board is shown below.

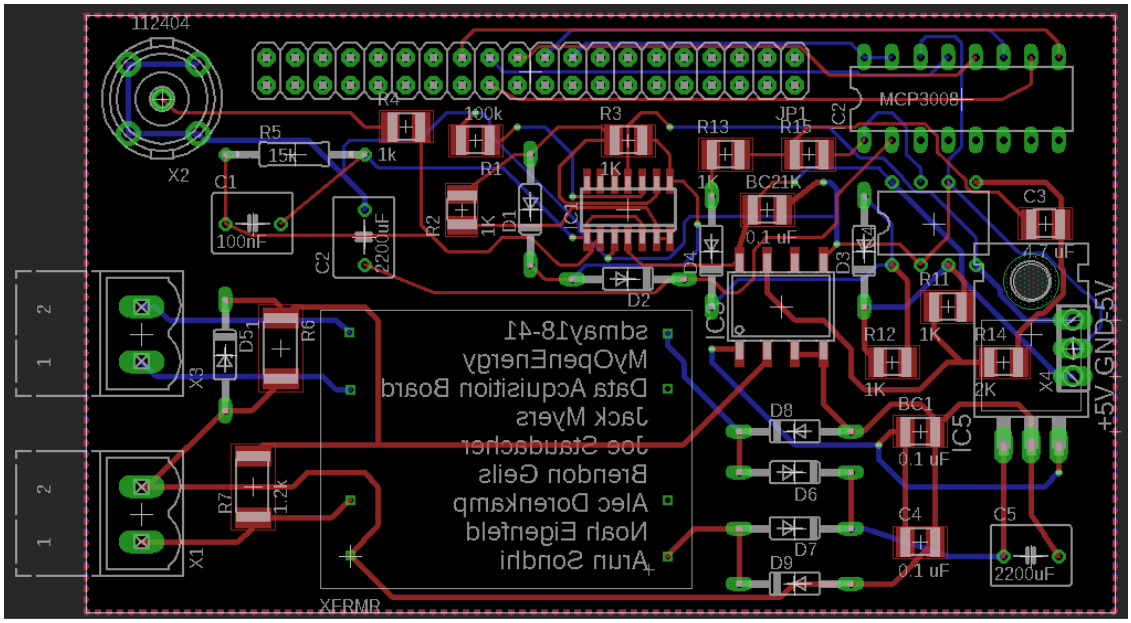


Figure 17: Layout view of the data acquisition board

5.2.2 Power board

The power circuit board is much smaller compared to the data acquisition board and contains fewer components. Its only purpose is to convert 12V AC from the main power transformer to a DC voltage that can be used to power the Raspberry Pi and the active components of the data acquisition board. This required the implementation of a dual-output full-wave rectifier and the use of positive and negative voltage regulators. The following figure shows the schematic of the power PCB:

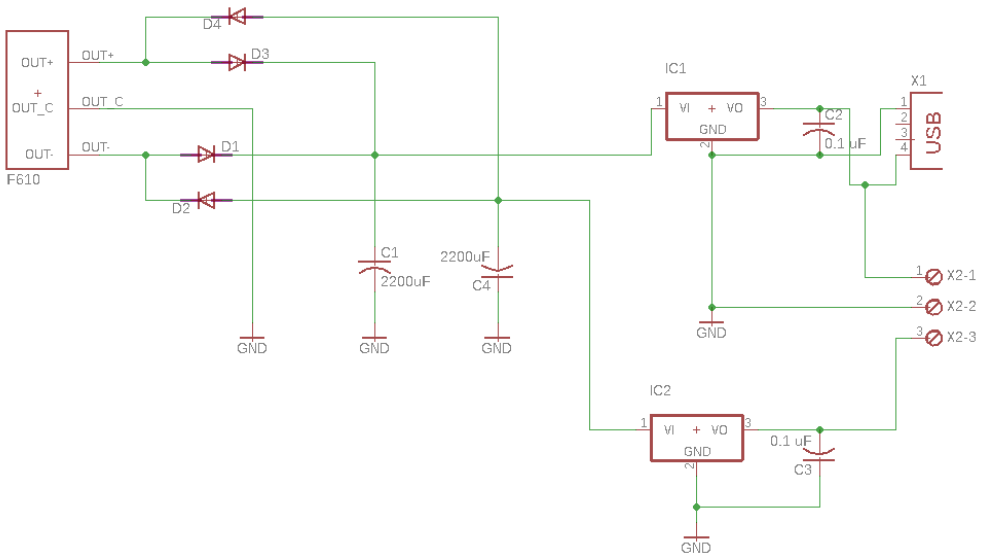


Figure 18: Schematic of the circuit on the power board

The input on the left is from the secondary of the main transformer and leads to the input of the full-wave rectifier. The output of the rectifier leads to the two aforementioned voltage regulators: one that outputs a +5v DC supply, and one that outputs a -5V DC supply. Both supplies lead to a screw terminal used to power the necessary components on the data acquisition board and the +5V supply also leads to a USB port from which the Raspberry Pi can be powered. The final layout of the power supply PCB is pictured below:

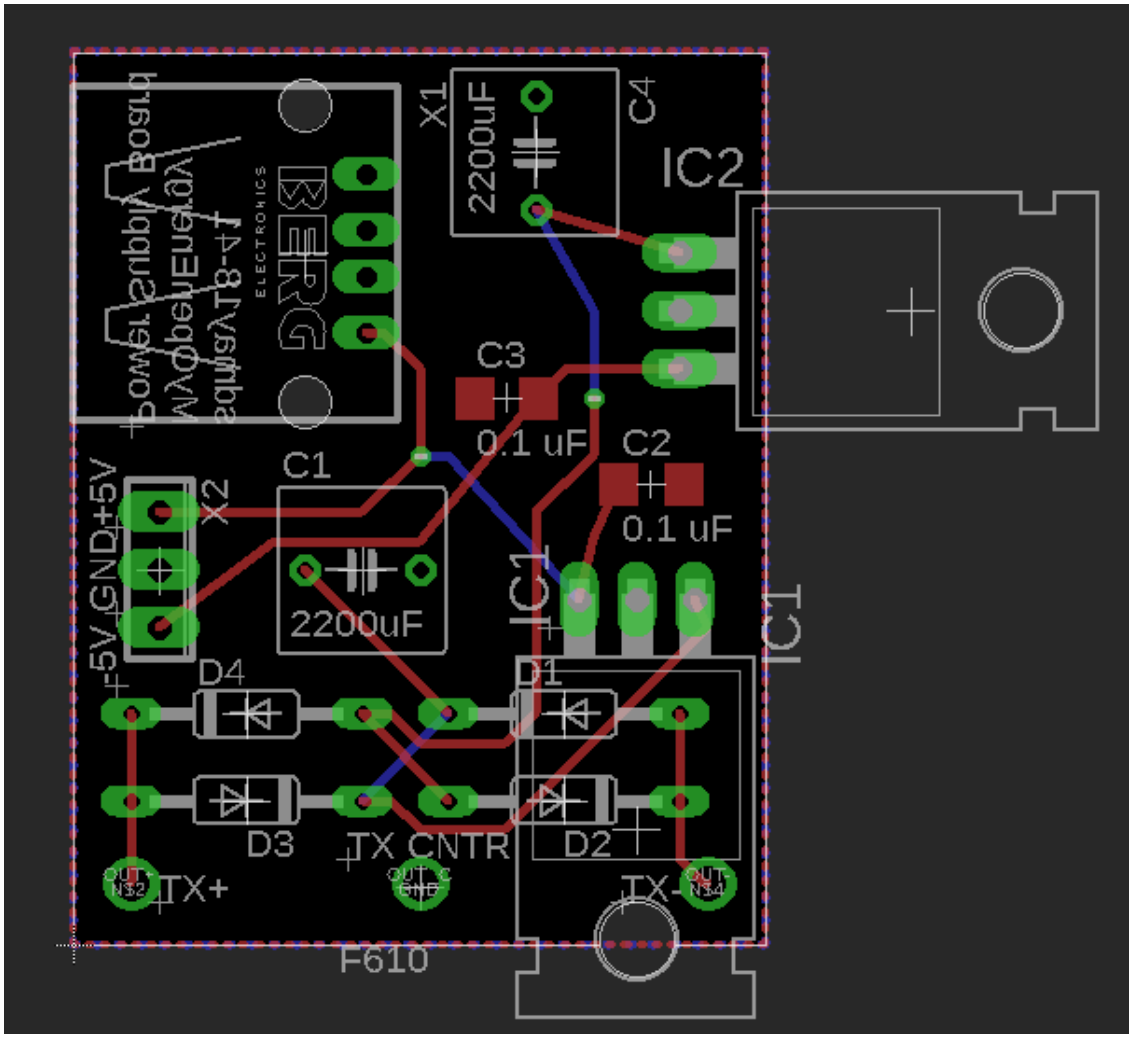


Figure 19: Layout view of the power board

5.2.3 Computing Module

The computing module of the hardware system consists of a Raspberry Pi that takes in the data and power from the PCBs, and transmits current consumption data to the platform developed by the software team. The software on the Raspberry Pi had three functions to carry out: acquire data from the ADC, display basic information to the user, and post data to the server. Once the raw data was acquired from the ADC and PCBs, the Pi was responsible for calculating the line current and voltage values represented by that data. These values are then displayed on a screen and posted to the servers through a SPI transaction. Once the transaction is completed, the computing module has fulfilled its purpose.

6 Testing and Implementation

Successful and thorough testing is necessary to implementing our solution correctly. The methods we used to test our design both in terms of individual units and integration testing is detailed in the following section.

- Manual testing
    - Frontend Web/Mobile
    - Backend
    - Hardware simulation and testing boards with various inputs
- Integration testing
    - Using frontend to test integration between front and back end code
    - Results of test actions confirmed in database
    - IoT uploading testing through front end data views and database updates

6.1 Unit Testing

The first main tests that a successful implementation has to pass are unit tests.

6.1.1 Software

There are three software components that needed to be tested. These include the MongoDB database and functions to add and change data in it, the web application, and the mobile application. The API methods written to interact with the MongoDB database from the web and mobile applications were unit tested. Each of the web app's pages functionality, usability, and performance was tested to ensure that everything works as intended and that there were not any simple errors preventing users from functioning properly. Finally, the mobile application was unit tested in a manner similar to the web application.

Web Application

Account Management

Account creation, login, and device and location registration were tested to ensure that correct inputs would be accepted, and that incorrect inputs would be rejected or handled accordingly. For each required entry, we checked that invalid strings could not be submitted or prompted the user before attempting to submit. Within the add-location page, the latitude and longitude will only accept legitimate coordinates; any coordinates outside the valid range are automatically dropped or raised to the upper or lower limits, respectively, when the focus moves away from those text boxes. For device registration, we make sure that no duplicate device IDs can be entered (functionality that we also tested). Similarly, no two accounts may share the same email address; this functionality is implemented through our Okta app, but we made sure to test for it as well.

Data Display

The data download functionality was robustly developed. It is actively disabled if the user information required for data access is not available, or correct, and was tested successfully on a variety of users and data volumes.

Marketplace

We did functional testing of the marketplace, with aspirations to have specified unit tests for the system. Our proof of concept did not require these tests so we have noted them for the client for further development.

Web Server

Backend Data Management

Data aggregation was tested thoroughly, and aggregates data based on strict time boundaries. Testing was done with a variety of input sizes and volumes, multiple time ranges, and devices. The max volume of records that we aggregated was just over 2 million unaggregated records from one smart meter, which took 10 min to aggregate, which was then reduced down to ~250 records. At this rate, we could continuously aggregate the data from 8000 smart meters in 10 min.

Data posting was tested first by validating that information posted and received. We ensured the database was always complete and accurate, and the security of the data is proven by the methods and algorithms used to secure the data.

Mobile App

Login and all user input was tested to ensure that valid requests were made and that invalid inputs are prevented. This was done by using authentication libraries provided by Okta as well as using type specific inputs to prevent malicious data. Retrieval of the most recent statistical data and marketplace data was verified and crossed with the web application.

6.1.2 Hardware

For our project to be successful, it is imperative that we are able to acquire power consumption measurements in the same way that existing power meters do. Given this, most of the goals and tests associated with the hardware are functional--they must work for our project to be complete. The main components are as follows:

Data acquisition board

A fundamental requirement for our project is to be able to get accurate and consistent current and voltage readings in order to have a power meter that has a similar level of precision to options that are currently on the market. After doing some preliminary research, we were able to come up with an initial design for amplifying, filtering, and processing the signal generated by our current and voltage measurement mechanisms. The first step to determine if our design would properly function was a simulation of the schematic in a basic form of SPICE software.

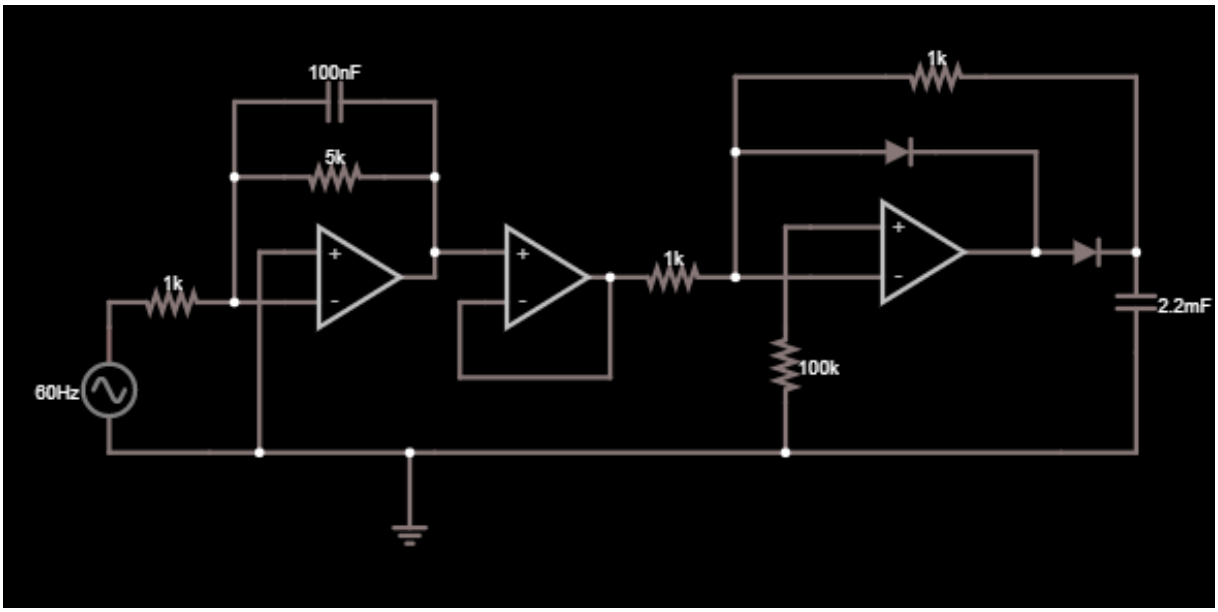


Figure 20: A schematic of our circuit design that we simulated. This portion of the circuit is for filtering and amplifying the signal from the current transducer.

We adjusted some of the resistor and capacitor values to determine which would give us the optimal desired behavior in terms of amplification, smooth filtering, and speed of response. This simulation also helped us catch a problem in our initial design, as we needed to add a buffer between the two main stages of the circuit to avoid undesired feedback effects. Once we were confident with the fundamental design, we moved onto testing the circuit with real components on a breadboard.

We also did similar testing on the voltage reading part of the board, with the circuit simulation shown below. Before actually hooking up live 120V or 240V to our circuit, we directly hooked up the small signal that would be produced by the voltage divider to the input of our isolation amplifier which fed into the filtering circuit. From here, we were able to confirm that the isolation amplifier and filtering components of the circuit worked correctly, so as long as the voltage divider worked correctly, we could know that this subcomponent of the circuit would work properly for reading the line voltage.

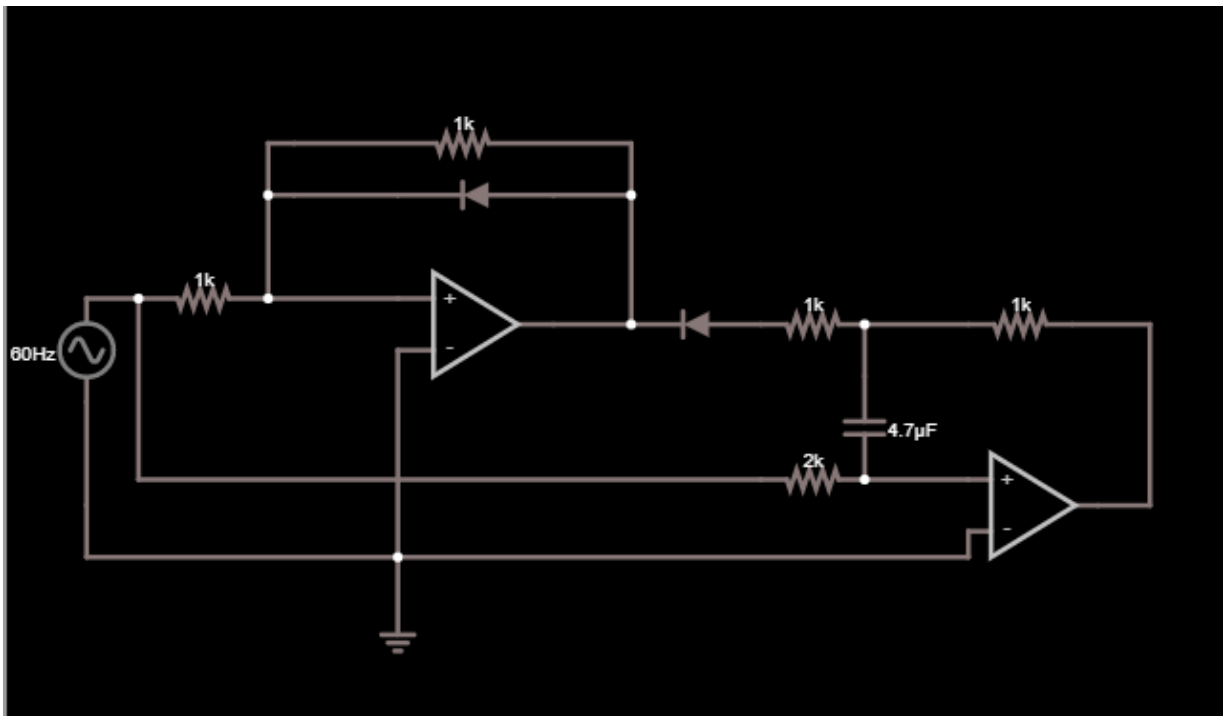


Figure 21: A schematic of the simulated voltage reading circuit from the output of the isolation amplifier and prior to the ADC input. This portion of the circuit is for smoothing out the signal from the isolation amp to a signal that is appropriate for the ADC.

Breadboard testing was fairly straightforward, but it helped us to confirm that any assumptions or approximations made by the simulator did not cause a discrepancy with the real behavior of the system. We built the circuit on a breadboard and made sure that it behaved the same way as the simulation. This was helpful in two ways. First, it allowed us to know that any issues that we faced with the PCB were likely because of bad soldering or bad components because we had already checked that the circuit would behave as desired with real components. Second, it allowed us to gain insight on what various signals in the circuit would look like if there were bad connections. For example, we were able to see what symptoms a bad connection on one of the op amps would have, which helped us to identify this behavior in the real board.

Before sending the board out, we used the built in Electrical Rule Check (ERC) and Design Rule Check (DRC) in the Eagle software that we used to design the board to make sure that we were creating a usable PCB. The main testing came once we actually had the physical board to work with. To test the final board, we used an oscilloscope to measure the signal at various points in the circuit to make sure that things were behaving as expected. This was especially important because this circuit had a mix of small AC signals and large DC signals, so reading with a multimeter would only give us a limited amount of information. We tested various nodes on the circuit to check that the signal there was expected. This was crucial in the debugging process, as it allowed us to determine points of failure in the circuit. This helped to find and fix issues like floating solder pads and cold solder joints. We were confident that the final board was working as expected when we tested the voltage at various important nodes and confirmed that it matched the simulations.

In addition to the analog data processing part of the circuit, it was also key for us to test that the header pins of the circuit were wired properly such that we could connect a ribbon cable to our board and the Raspberry Pi and be able to communicate via the SPI interface. The first way that we tested that these connections were wired properly was connecting the ribbon cable between the two boards and using a multimeter to run a continuity test between the ADC pins and the header pins on the Pi that we needed to be connected. After we confirmed that the connections were as expected, we checked to see if we could read data. When we were able to print out the readings in software, this confirmed that things were connected properly to communicate via SPI between the Pi and our ADC.

Power Board

The power board had a significantly smaller number of components, so in some ways it was easier to test than the data acquisition board. However, we were dealing with high voltage signals in some parts of the circuit, so we were at a much greater risk of either hurting ourselves or damaging the board. Because of this, it was especially important that we were confident in our design before trying to build it. Similar to the data acquisition board, we started off with a SPICE simulation.

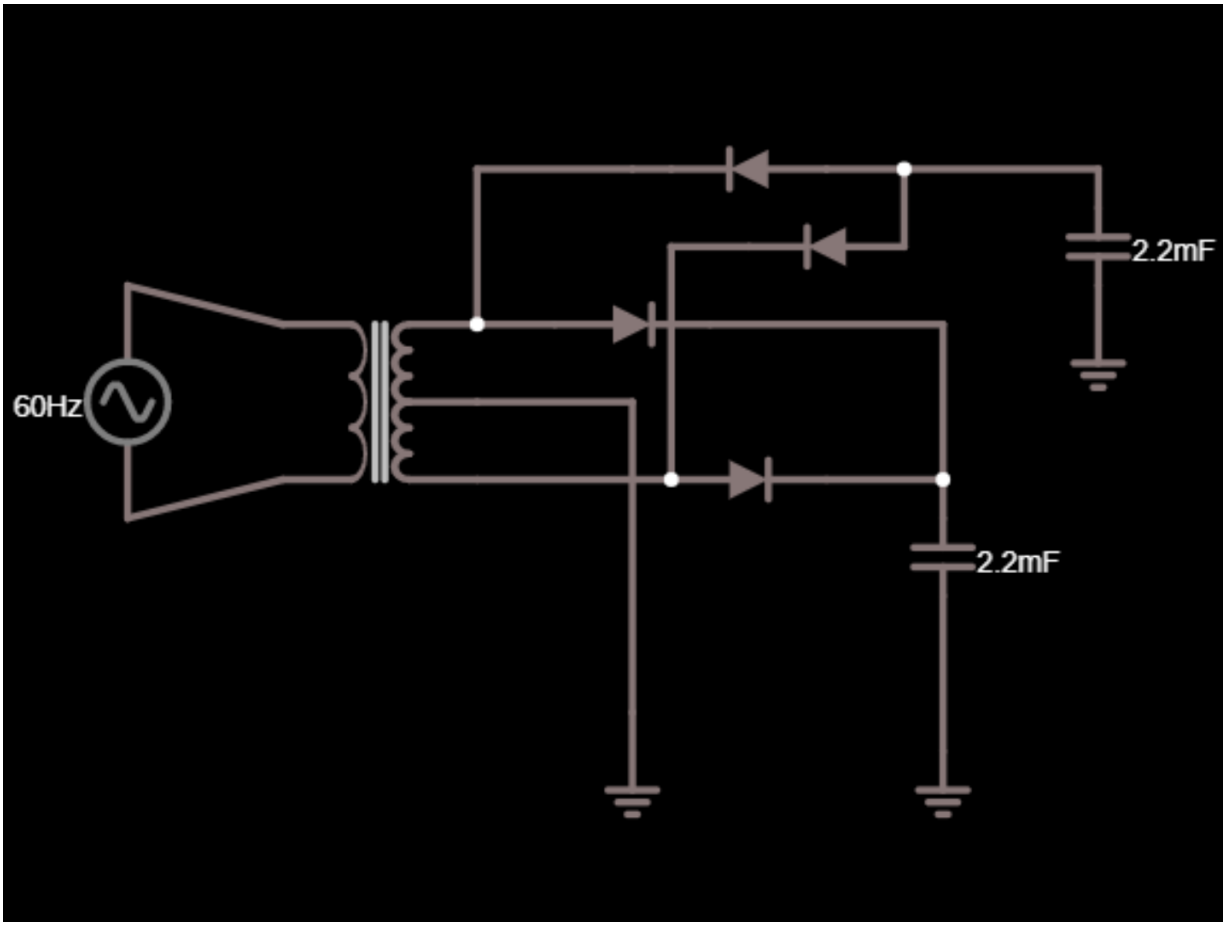


Figure 22: A schematic of our initial circuit design for acquiring power from the line. This part of the circuit produced a +5V source and a -5V source for powering the other board as well as the Pi.

This circuit did not require too much testing of the high-level design, because AC-DC converters are very well defined with ample resources available for working with them since they are so ubiquitous. Our main concern was making sure that we could implement the circuit in a safe and reliably way. The most important part that we had to make sure we were doing safely was on the live 120V side. To do this, we created the temporary transformer setup shown in Figure 23. This gave us a secure platform where the high voltage connections were insulated, only leaving the secondary side of the transformer (at a safe voltage of 12VAC) exposed.

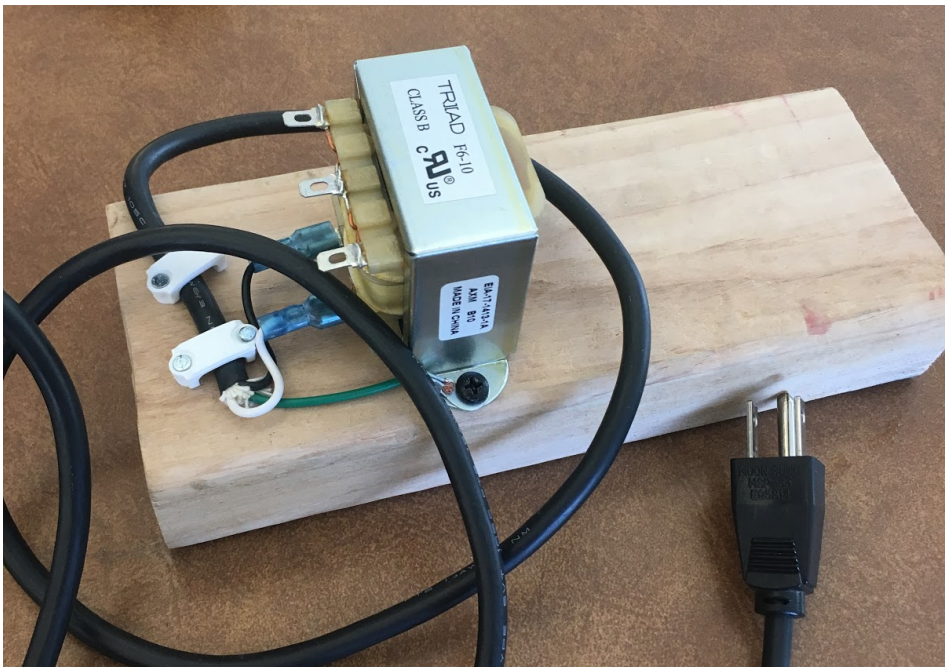


Figure 23: Main 240-12VAC transformer on testing platform

We ordered the board after making sure that our design met the Eagle DRC and ERC tests. Using our transformer platform, we connected the secondary of the transformer to our board. Once we had this setup, this board was very easy to test functionality--if we had 5V and -5V at the desired outputs of the circuit, then it worked. We ran into a few issues with this circuit, but they all seemed to revolve around bad components. Probing the nodes in the rectifier component of the circuit helped us to determine where things were failing. After replacing some diodes, we were eventually to get our +/- 5V outputs, as well as 5V on our USB connector that would be used to connect power from this board to the computing module.

Computing module

As our computing module was a Raspberry Pi, a pre-existing device, this simplified much of the testing that we had to complete. There were two main things that we needed to test.

First is the ability to accurately get a reading of the analog voltages from our data acquisition board. We tested this by comparing the values read in software from the ADC channel over the SPI interface with the values that we measured with a multimeter. We printed out the values that were read in software (converted

to analog) to the screen and compared them to the multimeter. After running tests on each channel with various voltages, we determined that we were able to get an accurate reading of analog voltages. Because we had already tested the analog data filtering on the board, at this point we were able to know that the data that we were reading in software directly corresponded to the current and voltage that we were reading with our transducer.

The other main thing that we needed to test was our ability to reliably post this data to our server. We were able to test this using a unit testing scheme. Each unit test ensured a portion of the API was running and responding as expected. We ran our unit tests locally to start and then on our staging instance of the API. After the staging instance was deployed we tested the functionality through a staging environment API request. This verified our server was accepting at the port we had the staging instance listening on.

6.4 Integration testing

While it was important that we had thoroughly tested the various modules of our project, it is crucial that we are able to integrate these components cohesively for a successful final deliverable.

6.4.1 Software

Software integration testing was simple because the way in which we setup our software development lifecycle dictated that it be performed continuously throughout the semester. Once unit testing was complete, each software component was tested locally and within the staging environment to ensure that it integrated nicely with the existing software components. When error-free, consistent behavior was established, the software component was integrated into the production codebase, and its functionality was further validated through testing with multiple users. Because the new software components were tested on multiple MongoDB instances, as well as both local and remote (AWS) servers, integration was consistently performed effectively and without incident.

6.4.2 Hardware

Integration testing for the hardware was straightforward. We had to make sure that the power board could provide enough current to power the Raspberry Pi as well as the active circuitry on the data acquisition board. We hooked all three of our hardware modules together with the key interfaces as follows:

- Power via the USB connector/cable to the computing module
- +/- 5V via screw terminals to the data acquisition board for powering amplifier circuitry
- Ribbon cable from the data acquisition board header pins to the Raspberry Pi header pins
- Display hooked up to the I2C pins of the Pi

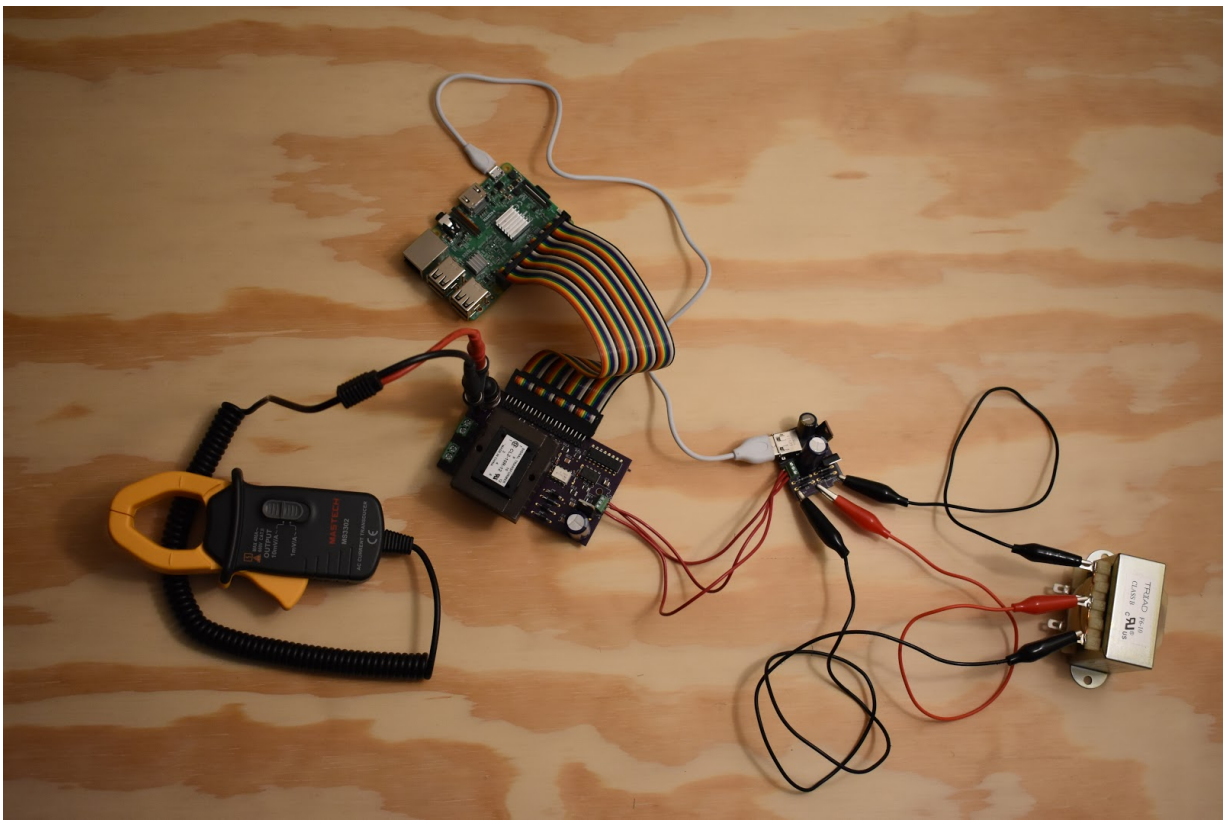


Figure 24: The overall components that make up the smart meter. The main components are the power transformer, power board, data acquisition board, computing module, and current clamps.

We had initially written the software for reading data, posting to the server, and driving the display as separate modules, so we had to combine them for this integration testing. Similar to the previously described testing, this testing was by and large functional. After some small modifications to the software and the hardware, we were able to use our hardware to power directly from the line voltage, read data, and post to the server. From there, we had to decide on an appropriate enclosure for the PCBs to rest in. We chose an enclosure made out of ABS plastic for a lightweight and durable combination. The box utilizes six screw-points to fasten shut securely and protect the circuits inside. The plastic of the enclosure is flexible enough for holes to be drilled through it, allowing for the connections leading to and from the components inside.

7 Conclusion

Money is the primary motivator for many making energy decisions, and as of right now, there is not enough of an economic incentive to prioritize the generation of renewable energy. Our goal is to develop methodologies and tools that would enable an economic benefit in order to incentivize personal installations of renewable energy. We aim to produce this economic benefit by providing a way to make or save money from renewable energy via a marketplace implementation of peer to peer energy transactions.

We determined that our solution would require three main components:

1. IoT smart meter: for reading energy usage and verifying transactions
2. Marketplace functionality: for facilitating secure transactions
3. Web applications: for allowing users to interface with our system

We have created prototypes of our three main components and put them through heavy testing.

As of April 2018, we have a functional prototype that we hope to use as a proof of concept in order to gain the traction to apply our solution at a large scale. Between the smart meter, user analytics platform, and trading software, we have the vital components that are needed to show the benefits of this approach to distributing surplus energy has over the existing methodology.

While there are groups like Grid+, LO3, and ConsenSys that have already made strides towards a similar solution, we feel that our team is working at just the right time. We are able to learn from the mistakes of our predecessors by taking what they would have done differently and applied those insights to our project. While we are not the first to work on this type of project, we are early enough that we are not fighting against any other groups that are dominating or monopolizing the market. We hope that our solution meets the needs of the users at hand and takes strides towards increasing the worldwide consumption and generation of renewable energy.

7.2 Future Work

As the project is handed off to Open Energy we are optimistic for the future work to come.

From a hardware perspective we would like the ability to send consumption and production data to our system in a more fault tolerant manner. With the current implementation network errors would introduce issues with clearing transactions for a client. Also, the communication protocol would be better suited as agnostic to our board processing. This would allow us to use cellular, ZigBee or other communication protocols in constrained environments. For example, a farmer in rural Iowa might not have WiFi connectivity. In this case a cellular connection may be more relevant.

From a software perspective we would like to allow more data visualizations and reports for the user. We believe with more data visualizations our users could reduce their energy usage. Another point we would like to add is dynamic transactional costs for the marketplace. This would allow us greater granularity in pricing the cost of energy. The overall costs would thereby be reduced because the individual user would feel the effects of further distance from the consumer, because they would pay higher transactional costs. The current implementation divides the transactional cost evenly among the users, thereby taking the load off distant producers or consumers. The final software improvement we are optimistic for is the ability to add

an automation system to the software client. This would allow the laymen user the capability to easily install this system and have it optimize the purchasing and usage of energy. Having a simpler solution would drive adoption as well.

7.3 Acknowledgement

We would like to thank those who have made contributions to the project, outside of the members of our team. First and foremost, we would like to thank our adviser, Dr. Goce Trajcevski, for his technical assistance and comprehensive guidance throughout the project. We would also like to thank Open Energy for providing the funding for many of our hardware components of the project, as well as providing the overall idea of the project. The Electronics Technology Groups was also a vital resource in the hardware design and component specifications stages of the project. Lastly, we would like to express our gratitude towards the faculty of Iowa State University for their support in giving us the technical background and knowledge for us to handle a project of this scale. Without the support of these individuals and organizations, our project's success would not have been possible.

8 Appendices

Appendix 1: Operating Manual

8.1.1 Software

Setup/Test

Below are directions to run and test the back-end, front-end and mobile app.

Back-End

1. Execute `npm install` from openenergy/open-energy
2. Execute `npm run dev` (if using windows then run `npm run dev-windows`) to run the main server on port 3000 and view it from the browser.

Front-End

1. Execute `npm install` from openenergy/open-energy/react-src
2. Execute `npm run dev`(if using windows then run `npm run dev-windows`) to run the development server for React on port 4200.

Mobile

1. Execute `npm install` from openenergy/mobile/okta-rn.
2. Execute `npm run ios` to run the application in the iOS simulator.

8.1.2 Hardware

To set up the hardware, the connections pictured in Figure 19 need to be made.

<u>Connections internal to the enclosure:</u>

- Use a USB cable to connect the USB port on the power board to the micro-USB power in port of the Raspberry Pi
- Use a ribbon cable connecting the header pins of the Raspberry Pi to the header pins of the data acquisition board
- Connect the display (hanging towards the center of the Pi) to the six header pins on the opposite corner of the ethernet port
- Connect the +/- 5V and GND from the power board to the data acquisition board
- Connect the secondary (top three pins) of the power transformer to their marked corresponding pins on the power board

For a market-ready version of the smart meter, these connections would already be made and would not require any additional work from the user.

Connections external to the enclosure:

These connections entail the steps that someone who purchases our smart meter would have to follow. Most likely, this installation would require a licensed electrician. For connecting to the mains voltage, one possible solution would be to have a unit that fits in the slot of a circuit breaker and has wires that extend to our meter.

- Connect the +120V, Neutral, and -120V into the corresponding port on the enclosure leading to the voltage measurement and powering circuitry
- Plug the black and red connectors on the current clamps to the black and red ports extending from the BNC connector on the data acquisition board
- Clamp the current transducers around the main production and consumption lines

Other setup:

To initialize the smart meter functionality, run the *simpledemo.py* script on the Desktop of the Raspberry Pi to begin acquiring data from the ADC, displaying the current power consumption information on the display, and posting this data to the server. In a commercial application of our meter, this program would be configured to run upon startup after the user had connected the meter to their Wi-Fi network.

Appendix 2: References

[1] "Bitfinex." *BitFinex*. N.p., n.d. Web.

[2] "ERC20 Token Standard." *ERC20 Token Standard - The Ethereum Wiki*. N.p., n.d. Web.

[3] Geils, Brendon. "Bgeils/pwr-blockchain." *GitHub*. N.p., n.d. Web.

[4] Geils, Brendon. "Pwr.company." *Pwr.company*. Blockchain News, n.d. Web.

[5] "Grid+." *GridPlus.io*. N.p., n.d. Web.

[6] "Gridsingularity.com." *Gridsingularity.com*. N.p., n.d. Web.

[7] Lacey, Stephen. "Our Picks for the Most Compelling Stories and Themes of 2016." *Greentech Media,* 19 December 2016. Web. 22 November 2017.

[8] Lumb, David. "This New York Project Fuses Energy Microgrids With Blockchain Technology." *Fast Company*. Fast Company, 06 May 2016. Web.

[9] Morgen E. Peck and David Wagman. "Blockchains Will Allow Rooftop Solar Energy Trading for Fun And Profit." *IEEE Spectrum: Technology, Engineering, and Science News*. N.p., 01 Oct. 2017. Web.

[10] "Project Sunroof - Solar Calculator." *Project Sunroof*. Google, n.d. Web.

[11] Sebnem. "Token Model for Energy - Part 1: Review of the Power Ledger Token Model." *Medium*. Medium, 12 July 2017. Web.

[12] Solar City, comp. "A Pathway to the Distributed Grid." (2016): n. pag. Print.

[13] "PowerLedger Token Generation Event." PowerLedger Token Generation Event, powerledger.io/.

[14] Ji, Ling. "Global Electricity Trade Network: Structures and Implications." *Public Library of Science,* 9 August 2018. Web. 22 November 2017.