NAU Energy Dashboard

Technological Feasibility Analysis

Team: Save Watt

November 9, 2018

Sponsored by:

Jonathan Heitzinger & Dr. Truong Ngheim

Faculty Mentor:

Isaac Shaffer

Team members:

Madison Boman, Hyungi Choi,

Ian Dale, Brandon Thomas

Introduction

Project Description

For the last several years, Northern Arizona University (NAU) has been collecting a trove of raw data related to the operation and energy consumption of its buildings. Until now, NAU has not had a proper tool to display and analyze the data efficiently. Current methods are cumbersome, time consuming, and require a great deal of technical expertise. There are currently three major issues with their business process:

- An interface which combines historical and live data does not exist
- A graphical representation of data (Charts, Graphs, etc.) does not exist
- An intuitive way to export data into a mutable file type does not exist

In searching for a solution, Jon Heitzinger and Truong Nghiem have commissioned our team, Save Watt, to build the NAU Energy Dashboard. The NAU Energy Dashboard is a comprehensive web-based application which is able to retrieve, graphically display, export, and run basic analytics on NAU's collected data. If successful, this software will have a huge impact on campus operation, leading to significant improvements in energy usage, cost reduction, waste minimization, campus sustainability, and research. Not only will the dashboard make the existing NAU building data useful, it will also aid NAU Facility Services in monitoring, understanding, and making well-informed decisions regarding energy efficiency. In the following sections, we will be discussing the feasibility of several aspects of this project and what challenges we predict in this early stage of development.

Document Outline

The goal of this document is to analyze the feasibility of each of the major design aspects of our envisioned solution. This analysis will serve as the basis of our design decisions moving forward and is an early idea of our design rational. We will explore our analysis in the following three sections.

In **section 1**, we will outline the major technological needs and challenges we foresee encountering. Here we will describe the rationale behind each requirement and give a brief insight into where the challenges may lie. In **section 2**, we will discuss each challenge, introduce various solutions, and prove the feasibility of our chosen solution by discussing our research. Finally, in **section 3**, we will combine the solutions proposed in section 2 into an integrated solution. Our cohesive solution will be depicted and discussed by way of a system diagram. We will close our document with a brief overview of our findings.

Section 1: Technological Challenges

Introduction

Here we will introduce the major requirements of the NAU Energy Dashboard and briefly discuss the challenges associated with each requirement. We will do this by first listing each of the requirements and secondly exploring the rationale behind each requirement and listing the major challenges we foresee. These challenges will be further analyzed in Section 2.

1.1 Technical Requirements

As we have met with our sponsors, we have collected a series of requirements that will form the basis of our system. We have selected five major requirements to discuss in this document. These requirements are listed below:

- Retrieve historical data
- Retrieve live data
- Clean Data and Perform Statistical Analysis
- Present Data Graphically

Implementing these requirements is the basis of our project. Our energy dashboard will need to efficiently and effectively perform each of these tasks in a way that is satisfactory to our sponsors. In the future, we will discuss the metrics by which each of these requirements will be measured (see Requirements Specification Document). For the purposes of this document, we will simply be evaluating whether or not each requirement is feasible and how we plan to demonstrate feasibility. In the following subsections, we will discuss the rationale and challenges behind each of these requirements.

1.2 Retrieving Historical Data

Our clients' business process begins with data retrieval. Historical data, in the form of trend-logs, is currently stored in a large SQL database. Although there is a wealth of information to be gathered from this data, the data stored here is disorganized, inconsistent, and error prone. High level skills in SQL are required to manipulate and extract any information from the data. Our system must be able to query this data in a way that is intuitive to the average user. The major challenge here will be finding a tool that is powerful enough to search through this large database, is flexible enough to support NAU's growth, and is compatible with NAU's current architecture.

1.3 Retrieving Live Data

This second requirement is much like the first. Along with historical data, we will need to retrieve live data from building sensors. This process involves sending requests to the sensors via the appropriate protocol. The challenges here lie in granting access to our system and finding a tool which can communicate with the building servers. This technology is older and not uniformly configured, so we will need to use specialized languages and/or frameworks to retrieve this data.

1.4 Presenting Graphical Data

Here we are focusing strictly on technology that will graph and chart the data we will be retrieving. Navigation, user experience, and design aesthetics of our web application is not foreseen as a major challenge. Our sponsors would like to be able to represent trends in the form of time series graphs, histograms, and comparative line graphs. The major challenge here is finding a tool that is able to handle large amounts of data, and is able to present it interactively.

1.5 Cleaning Data and Performing Statistical Analysis

The final major requirement is cleaning the data and facilitating internal analysis as well as external analysis. This entails accounting for missing data points, mitigating errors, and logically aligning the data. After cleaning the data, our sponsors would like for the dashboard to run minor statistical tests. These tests may include correlations, five point summaries, and regressions. Our sponsors would also like to easily search for data, export it, and use outside tools to perform analysis. The major challenge here is finding a tool fit for data handling and statistical analysis.

1.6 Conclusion

Now that we have introduced the requirements and associated challenges, we can begin our analysis of possible solutions. In the following section, Section 2, we will discuss the tools and methods we will implement in our overall solution.

Section 2: Technology Analysis

Introduction

In this section we will discuss possible solutions to the challenges introduced in the previous section. We will discuss the methods and reasoning behind our approach to each solution. To reiterate, our challenges are the following:

- Retrieve historical data
- Retrieve live data
- Clean Data and Perform Statistical Analysis
- Present Data Graphically

Next, we will explore our proposed solutions to the listed challenges.

2.1 Retrieving Historical Data

Retrieving historical data is key to providing a comprehensive view of NAU's energy consumption. Our system must be able to query this data and use it for analysis. A major challenge associated with this will be running real-time queries to this database when performing searches for information. Our sponsors would like to search for specific data sets using familiar keywords and domain-level nomenclature. In order to do this, we will need to use a tool compatible with both SQL and the rest of our system. After comparing several database technologies, we narrowed the decision down to two alternatives

Alternatives

- *PYODBC*
  Pyodbc is an open source Python module that makes accessing databases simple. This module implements Python's database API 2.0 in a way that is easy to use with knowledge of Python. It can connect with various databases including Microsoft SQL.

- *PYMSSQL*
  Pymssql is a simple database interface compatible with Microsoft SQL. This package is very specific to Microsoft SQL, so it is not compatible to any other database technology. This fact will limit our dashboard in the event that NAU migrates to another technology.

Package	Support for Microsoft SQL	Flexible Support	Maintainability
pyodbc	x	x	x
pymssql	x		x

Figure 1. Shows an evaluation of both pyodbc and pymssql based on relevant features.

Chosen approach

Figure 1 shows a comparison between pyodbc and pymssql. Using these metrics we were able to select a tool for our system. Our chosen tool for retrieving historical data is pyodbc. This option provides the simplest way to connect to the database within our system. Pyodbc will also allow for our dashboard to continue to grow with NAU's needs.

Proving Feasibility

To prove feasibility, we implemented a test environment with a custom SQL database. In this test environment, we were able to select, update, import, and export data. At the moment connecting to the database requires windows authentication from a campus computer. We are not able to connect using a VPN. In order for our code to work is must be run in a remote desktop. We are unable to download new python packages on a remote desktop without administrative access. We know that this tool is compatible with a similar database, but lack the authority to connect the the provided database.

2.2 Retrieving Live Data

All of NAU's building operation data is retrieved via meters which can be called upon periodically to report the current status of any given utility. Each building has dozens of sensors which are controlled by central building controllers. These building controllers are essentially servers which make requests to the individual sensor IP addresses. The challenge here lies in getting authorization to access these IP addresses and implementing the proper protocol for our system to communicate with the building controllers.

Alternatives

- Niagara AX

  Niagara AX is the native language of the building controllers. In order to retrieve live data we will have to encounter this language in some way. The benefits of using this language solely is that our system will be completely in tune with the building controllers. Maintenance will be easy, assuming technicians who are familiar with he building controllers are also familiar with Niagara AX. Although maintenance and compatibility are high priorities, these are all that Niagara has to offer. This language does not work intuitively with any other language we have researched and does not offer any other tools. Familiarity is a key factor here as well. We are not familiar with this language at all, so the learning curve for our team will be extremely challenging. Overall, we would like to minimize our use of this language.

- DGLUX

  DGLUX is a framework used to build energy dashboards. This framework has been used to build an older iteration of a dashboard at NAU. DGLUX provides a built in way to communicate with building controllers running on Niagara AX. DGLUX is also compatible with Javascript in the form of a widget. This tool is already designed for the construction of a web-based energy dashboard, so it may fit perfectly in our needs. We also have a sample implementation where we can base some of our work. This sample will allow for a better basis to learn how to use the framework. The only drawbacks of working with DGLUX are that the framework seems a bit outdated, and the widget for javascript is very limited in its functionality. In order for this tool to work we will have to make some adjustments.

Chosen Approach

The tool that will best suit our system is DGLUX. Niagara AX on its own is not robust enough to be compatible with the rest of our system in a way that is intuitive. Using this language on its own would require extensive learning and unorthodox methods to force compatibility. DGLUX is more viable, although it comes with its own challenges. It is proven to have worked in the past and offers tools for compatibility between our system and the building controllers. We will need to invest more time in building modules to improve DGLUX, but, despite the need for extra customization, it seems to be the best tool for the job.

Proving Feasibility

We know this approach is feasible, because it has been done in the past. We have been using the files from the previous iteration of the dashboard to decode the functionality of DGLUX. We have not been able to run any tests, because the building controllers are heavily guarded by NAU ITS. We are currently in the process of gaining access and meeting with individuals in ITS that may have more information than our clients.

2.3 Cleaning Data and Performing Statistical Analysis

Our sponsors would like for the dashboard to run minor statistical tests for them. They would like to be able to perform commonly used analyses automatically in order to save time and energy for larger analyses. In the past, attempts at statistical analysis have resulted in errors which rendered the data useless. We will need a tool that can accurately run the desired tests, account for data anomalies, pass the results to our presenting module, and export the data to a desirable file type such as CSV. The tools we are considering are Python packages, numpy and pandas.

Alternatives

- *NUMPY*

  Numpy is a general purpose array-processing package. Numpy offers many beneficial tools such as linear algebra calculations, transformations, and random number generators. These tools make it powerful enough to run numerical analysis on the data we gather. Numpy is very powerful, but integration with other technologies is cumbersome. Integration requires is C/C++ and Fortran code. These barriers to integration make this tool very difficult to include in our system.

- *PANDAS*

  Pandas offers many data structures and operations for manipulating numerical tables and time series. These tools will allow us to align data points, sort data within our own structures, and account for missing data. It includes many of the same tools for numerical analysis as numpy, but is better suited to integrate with web-based technologies. Analyses performed with Pandas is easily compatible with SQL, Javascript, and R. This compatibility is an enormous benefit as it allows us and our clients to use familiar tools when working with the data.

Package	Integration?	Features
Numpy	C/C++, Fortran, Python	Array processed packaging, Linear algebra calculations, transformations, and random number generations.
Pandas	Javascript, SQL, R, etc.	Many datastructures, many operations for data manipulation on numerical tables and time series, data alignment, NA friendly stats, and similar features to SQL calls

Figure 2. Shows an evaluation of both numpy and Pandas based on relevant features.

Chosen Approach

Figure 2 compares the features of numpy and Pandas. After comparing the two, the only advantage numpy has is the ability to use linear algebra for analysis. Despite this, we have chosen to use pandas as the technology for our statistical analysis. Numpy alone is a powerful tool, similar to using technologies as MATLAB, but does not allow for straightforward integration. Pandas will allow us to create rich time series functionality for sorting logs, data alignment for our data comparisons, NA-friendly statistics for instances were data may be missing, data groupings, merging, and joining methods.

Proving Feasibility

To ensure that pandas will work, we tested this library alongside our chosen graphical interface in order to ensure compatibility. Since our historical data will be pulled from an SQL database, we were able to pull data and work with it using pandas. The next phase here is to develop a set of analyses tests alongside our client that would best fit their business process.

2.4 Presenting Data Graphically

Presenting interactive graphical data is the main result our sponsors would like to see. This project is driven by the need for a graphical user interface that abstracts away the technical details and focuses on the data. A major challenge here is choosing a tool optimized for reading and presenting large amounts of data. There are three alternatives that we are exploring: matplotlib, Charts.js, and amCharts. We will discuss the benefits and drawbacks of each in the following subsections.

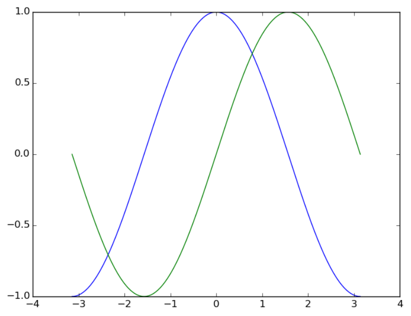
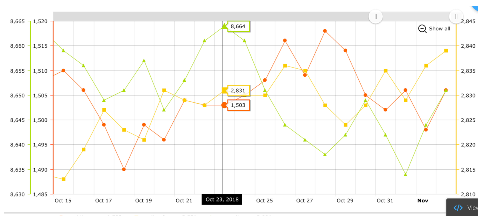
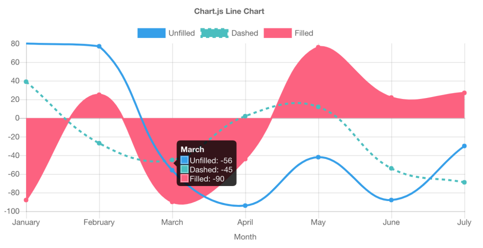
Language/Package	Graphical Examples	Interactivity?
PYTHON Matplotlib/Seaborn		None.
JAVASCRIPT amCharts		Hover data descriptions, Vertical line for pinpointing data points. Multilayered graphs. Graph color customization.
JAVASCRIPT Charts.js		Hover data descriptions, Horizontal line for pinpointing data points. Multilayered graphs. Graph color customization. Animations on load

Figure 3. Shows an evaluation of matplotlib, amCharts, and charts.js.

Alternatives

Python Django offers graphing tools such as Matplotlib/Seaborn to better graphically present the data. On the other hand, Javascript has two options, amCharts and, Charts.js. Matplotlib/Seaborn would be suitable to use because they interact well with Python. However, the graphs are mediocre when considering a modern graphical interface. Figure 3 shows a comparison of graphics between each of the alternatives and describes some of their features. We can see from Figure 3 that Chart.js provides a rich and descriptive view of the data it is given.

Chosen Approach

The approach we have chosen is Charts.js. Matplotlib/Seaborn cannot compete with Charts.js because it is not as appealing nor does it allow for any animations. We chose Charts.js over amCharts because Charts.js includes the animations on load. This is helpful when providing engaging animation from the very beginning for our users. With Charts.js we are able to hover over data points to reveal in depth information in a clean and intuitive way. AmCharts did include a similar feature, but implementation is messy and unappealing. The clients have also shown us various websites that use Charts.js and have expressed a desire for a similar design. Overall, Charts.js provides the compatibility and design aspects that best fit our system.

Proving Feasibility

To choose the best approach to graphically represent the data, we downloaded numerous examples from the their websites to run and, test with in our own environment. Using Javascript's Charts.js will work with our system as a whole seeing as there are numerous instances that Chart.js has been used to display data from Python's pandas that we will be using on our data. We were able to use our own environment to create a set of test values in Pandas and pass them to Charts.js. The scripts we used to modify our data are straightforward. They simply require a file to pass several values and allow for flexible theme changing.

2.5 Conclusion

Now that we have introduced the solutions to the major challenges of our system, we will discuss the system as a whole. In the next section, we will map the system in terms of its modules and discuss how each of our solutions will come together as a cohesive unit.

Section 3: Technology Integration

Introduction

In this section we will discuss how each of the requirements will be fulfilled and will be made to work together. Figure 2 shows a diagram of our perceived system. The back end consists of data retrieval, cleaning, and statistical analysis. The front end is concerned with presenting the data to the user. Each of the packages chosen are depicted in the diagram.

3.1 Integration

Python provides a cohesive ecosystem that allows us to retrieve data, run statistical analysis, and develop powerful graphical user interface. Our overall framework with run on Python Django. After speaking with ITS we have found that Django is compatible with NAU's systems. Using this python based framework, we will be able to include the packages previously mentioned, since these packages are written in a common language.

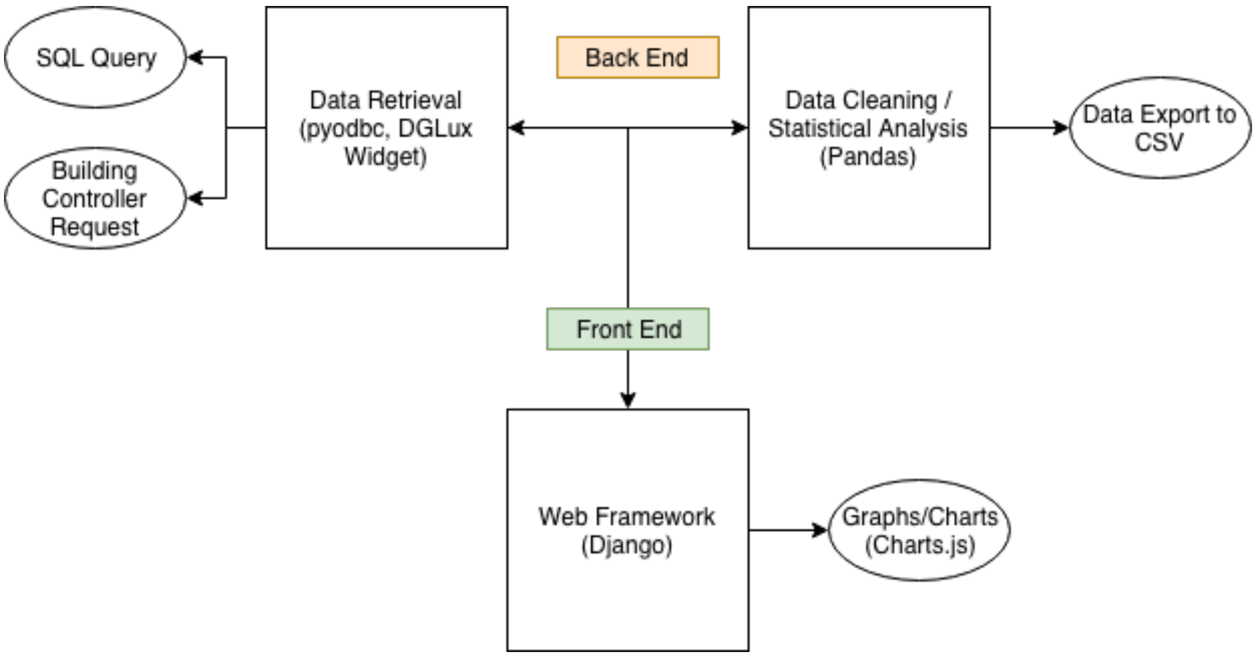

SQL Query
Building Controller Request
Data Retrieval (pyodbc, DGLux Widget)
Back End
Data Cleaning / Statistical Analysis (Pandas)
Data Export to CSV
Front End
Web Framework (Django)
Graphs/Charts (Charts.js)

Figure 4. A System Diagram showing how modules will be organized, how they will interact, and what tools will be used in implementation.

Our system as a whole will be constructed as shown in Figure 4. The back end will consist of collecting and cleaning the data. We will be using Pyodbc to query NAU's SQL database and an abstraction of Niagara AX to request updates from the building controllers. Historical data will be cleaned using Pandas. Here we will wipe away detected anomalies, account for missing/null data, and align data points as necessary. Using

Pandas, we will also be able to perform general statistical analysis. Clean data will be presented to the user using Charts.js. The user will be able to search for and export data using user interface designed with Django. These functions will allow the user to select data sets and manipulate the date within a CSV outside of our system.

3.2 Integration Challenges

The major integration challenge we foresee will lie in communicating with NAU's building controllers. This will require a separate networking package which will be able to send and receive proper requests using the protocol that the controllers use. DGLux is compatible with javascript which works closely with Django, so this is our foreseen path to solving this problem. There are 3 major issues associated with communication between our system and NAU's building controllers, these issues include: Access Granting, Tool Creation, and Compatibility. We will discuss these in the following subsections.

Access Granting

In order to even begin testing this integration issue, we need access to NAU's closed network of building controllers. Our sponsors are unsure of how this technology is implemented, so we must direct our communications to ITS. Due to processing delays and our limited access privileges as students, we have run into several delays in gaining access to these systems. Once access is granted, we will be able to investigate the architectural setup of the building controllers. An insight into the architectural set up of the building controllers will allow us to run tests using the necessary protocols and IP addresses. We expect access to be granted shortly.

Tool Creation

As we have delved into this issue we have discovered technologies that have been used to access the building controllers in the past (i.e. ResHall Dashboard, Lucid Dashboard). These technologies have built their own libraries which implement the protocols necessary. ResHall Dashboard, for example, uses a custom widget for DGLux to Javascript as mentioned before. This proves that accessing the meters directly is possible, but requires implementation of a package that is not publicly available. With these older technologies as reference, we will need to create a custom tool that may communicate with the necessary meters.

Compatibility

The custom tool will need to be compatible with our pythonic system in order to properly integrate with our system. This could mean that the tool is written using python libraries or simply allowing for communication between the system and the tool using an intermediary. We plan to create a similar javascript widget that will be able to communicate with our system.

3.3 Conclusion

In this section we have discussed the planned structure of our system and how we foresee it working together. In this early stage we require future investigation to prove a conclusive design. Major issues are largely due to a lack of authorization at this point. Moving forward we will be implementing a testing environment to show a small scale version of our system. In the next section we will close our analysis with some final thoughts.

Analysis Conclusion

Our mission is to create an interactive energy dashboard for NAU's facility services. With the help of our sponsors, Jonathan Heitzinger and Truong Nghiem, we will to create a system which allows for historical data retrieval, live data retrieval, statistical analysis, graphical representation, and clean exportation of NAU's building operation data. In the sections above, we explored the feasibility of each of these features. We concluded that the python language provides the most comprehensive and development friendly tools to accomplish each of these tasks. Python will provide packages for web programming (Django), data analysis (Pandas), and data retrieval (pyodbc). We will also include compatible javascript packages to present our data (Charts.js) and retrieve live data (DGLux Widget). With these tools we plan to build the NAU Energy Dashboard with a view to a maintainable, scalable, and energy efficient future.

As we discussed in previous sections, this analysis has taken place at an early stage of the project in order to begin to mitigate future issues. There are still open questions with regards to communicating with NAU's building controllers. We plan to resolve these questions with further investigation alongside our sponsors and ITS. We also plan to use previous solutions as a guideline when developing a tool to accomplish our clients' goals. We are confident that we are on our way to a robust, interactive, user-friendly system that will change the way that NAU views consumption.