Reducing Campus Electricity Consumption Through Interactive Data Visualization

Topic 2 - Software Engineering

Christopher Maree - 1101946

Executive Summary

This paper presents the conceptualization, design and implementation of a toolset aiming to provide comparative energy usage visualizations to unlock the power of energy data. The goals of the project is to shed light on Wits' energy usage to gain insight into behaviour, identify trends and in the long run, reduce campus consumption. The project aims to help Wits save money and become a greener campus. The most effective visualization techniques to convey the underlying information embedded within energy usage data are explored. A generic, technology stack agnostic solution is proposed along with a working minimum viable product that can be found online at witsecop.co.za and documented repository can be found here.

WITS
UNIVERSITY

School of Electrical and Information Engineering
ELEN4000 Electrical Engineering Design II - Software Engineering
Supervised by Prof. K. Nixon
23 October 2018

CONTENTS

I. Introduction

The cost of electricity in South Africa has increased by almost 900% in the last fifteen years[1]. This increase places direct financial pressure on universities who have little to no financial headroom to deal with increasing costs. Moreover, South Africa's national utility (Eskom) has often fallen short of meeting the national grid demand, resorting to rolling blackouts (load shedding)[2]. The intermittent electricity supply is very disruptive to normal university procedures and results in further incurred costs. Looking to the future, it will become increasingly important to become a "green" institution with the implementation of carbon taxes in South Africa in 2020[3]. Changing the way we consume energy on campus could make a real positive impact on our carbon emissions and help combat climate change. It is therefore apparent that something needs to be done to drastically reduce the electricity consumption within South African universities to address these pressing problems.

Reducing an institution's electricity footprint requires both technological and behavioural intervention[4]. It does not matter how technologically advanced an intervention is, if there is not the associated required behavioural change to support it. This means that in order to achieve electricity reduction within universities, a multidisciplinary approach must be taken. A key component in reducing electricity consumption is the ability to analyze usage patterns in order to identify inefficiencies, enable optimization and promote a behavioural change toward energy usage. The most powerful tool to achieve this is data visualization of live and historic energy consumption on campus.

This paper presents the conceptualization, design and implementation of an interactive toolset used to visualize, compare and contrast electricity consumption on campus to support green initiatives and to help universities transition towards a more sustainable future. The paper is broken down into three broad sections: firstly, the problem statement is fully analyzed, considering the South African and Wits context. The role of visualization in addressing and reducing energy consumption use is also explored. The social, economic and environmental impacts are considered as well as the stakeholders who will use the final system. Then, both technical and non-technical implementation details are discussed such as technology architecture and a suggested implementation. An accompanying MVP design is then discussed. An online version of this can be found here: witsecop.co.za Finally, the solution is critically analyzed considering the sustainability, maintainability and effectiveness as an engineering solution.

II. Problem Statement Analysis

"*If You Can't Measure It, You Can't Improve It*" - Peter Drucker

If you don't know how much, where and when energy is being used then it is impossible to reduce consumption in a meaningful way. Visualization provides the feedback necessary to quantify any increase or decrease in energy consumption such that appropriate action can be taken. If a green initiative is proposed on campus, such as to reduce energy consumption through the installation of LED light bulbs, a mechanism is required to quantify any reduction in energy consumption to prove the successfulness of the initiative. Without adequate energy usage visualization the university is potentially overspending on electricity due to being in the dark about potential inefficiencies and wastage.

The role that visualization can play in energy consumption reduction on campus can be broken up into three main sections. Firstly, it can be used as a tool to quantify reduction in energy resulting from green initiatives. Secondly, it can be used to identify inefficiencies within campus by comparing current consumption to past trends to enable proactive response to consumption trend changes. Lastly, energy visualization can be used to positively change the mindset towards energy usage on campus. Each of these key points is explored in turn.

A. Visualization Advantage: Quantifying Consumption Reduction

Without data driven insights into energy consumption there is no way to rationalise the savings brought about by green energy initiatives. The traditionally hidden nature of electrical energy makes it hard to visualize where and how energy is being used. Through the visualization of electrical energy invisible usage patterns and trends can become clearly visible. This visualization can be used to quantify reduction in consumption resulting from green energy initiatives. As a result, green initiatives can be more easily motivated by having a quantitative backing to prove energy usage reduction.

B. Visualization Advantage: Identify Campus Inefficiencies

By visualizing the holistic consumption of all buildings on campus one can identify the largest consumers and thus potential inefficiencies. Building load profiles can show insight into the consumption patterns of a building over the course of the day. This can highlight potential saving opportunities such as high loads during the night indicating that lights or machinary are being potentially left on. Moreover, inefficiencies during weekends and holidays can

be identified where the building consumption should be much lower than during university term time. Comparative visualizations are key in highlighting these potential savings.

C. Visualization Advantage: Invoking a Mindset Change towards Energy Usage

There is a powerful link between emotions and visual imagery[5]. Research has shown that this link can influence both motivation and goals[5]. By providing a public, open platform to visualize energy consumption on campus, people can see for themselves the impact that they are having on campus energy consumption and as a result begin to change their mindsets towards how energy is used. A public platform available to students and staff alike can instill a sense of social responsibility toward energy usage reduction by making it clear to people how they personally contribute to the consumption on campus. More broadly, visualization provides a way to convey the previously invisible usage patterns to campus management and administration therein altering the mindset towards energy. Energy is no longer an invisible thing that can't be controlled or reduced because it is impossible to quantify or visailize. Visualization makes energy usage something that can be understood, controlled and reduced with visual feedback at every step of the process.

III. Literature Review

There is a myriad of relevant literature to draw upon when looking at how to reduce energy consumption within universities and what role energy visualization plays in reducing usage. Three key papers and projects have been identified that provide invaluable insight for this project.

The Journal of Energy in Southern Africa[4] looked at how to promote energy efficiency in a South African university by performing a case study on the University of Johannesburg in 2016. The report identified a number of key factors that influence the ability for an academic institution to reduce energy consumption. The relevant takeaways from this paper were the hurdles that had to be overcome in facilitating a radical reduction in energy usage within a South African context. The paper emphasised the importance of an institutional behavioural change towards energy usage within the university in order to realise any real world reduction in consumption. This takeaway is of direct relevance to this project wherein visualizations are being used to try and instill a behavioural change towards how energy is consumed. The paper also highlighted the importance of quantitative feedback in any energy efficiency program, further adding validity to the importance of relative comparisons to verify energy usage reduction.

The Energy visualization for Carbon Reduction (eViz) project[5] aimed to show people how energy is being lost in their homes and how different energy behaviours impact on their energy usage. Their papers highlighted different visualizations techniques and how they could be used to address energy usage. Of particular relevance and usefulness was their discussion into the psychological response to different visualization types and how visualizations can be used to grab attention and evoke emotions. Primarily, they speak about the cognitive and psychological effect that imagery has and the important motivational role that it can play in influencing decisions. The authors identified that a powerful visualization technique is heat maps using thermal imaging as they have been found to evoke an emotional and memorable response resulting in a more impactful long term behavioural change. These findings were used to inform decisions regarding what visualizations should be included to instil the maximum positive behavioural impact.

Lastly, Harvard University has created an energy visualization website[6] that represents energy and emissions for the university over time. Their website contains a number of informative visualizations aiming to show the footprint of the university. Their implementation provides a valuable point of reference to different visualization techniques to compare and contrast buildings. An alternative version of the website [7] shows the correlation between building consumption and other properties like temperature or humidity. These websites show how powerful different visualization techniques can be to identify large consumers on campus as well as how energy reduction initiatives can have a quantitative reduction in consumption over time.

IV. South Africa and Wits Context Context

Most South African university campuses were not designed to be energy efficient[4]. South African universities tend to have multiple large, high load buildings that were constructed when energy efficiency was not considered. Coupled with this is that any proposed new solutions will require to operate within the current environment as a brown field solution. It is not financially feasible to do campus wide overhauls and hardware replacement and resultantly any proposed technological system must be able to be retrofitted within the existing architecture.

More broadly, South Africans in the past have not been overly concerned with their energy consumption as there was little to no financial incentive to be green due to an abundant supply of cheap electricity[2]. However, the last

15 years have seen a drastic increase in electricity prices in South Africa resulting a forced behavioural change towards electricity consumption[8]. Universities are not exempt from this price increase and so finding a way to reduce energy costs is becoming an ever pressing requirement.

Wits University currently has over 260 energy sensors installed by PIMD[8]. These sensors have been installed for each main supply, substation and building on campus and are used primarily for the billing and recoveries for internal and external tenants such as companies operating out of the Matrix[8]. Information captured by these sensors is stored on the ecWIN platform, a system created by IST[8]. The goal of the ecWIN platform is to create a mechanism to cut energy and utility costs, to identify energy intensive activities and to assist in creating awareness of energy consumption. The ecWIN platform is however not accessible to the public and proves inadequate at providing insight into usage patterns and trends due to its cumbersome and confusing user interface. There is no ability to generate relative metrics to compare one building against another so inferring behavioural changes becomes very difficult.

An important part in defined system needs to be stakeholder engagements. The tools provided need to be appropriate for variable members of the university community, including management, staff and students. Detailed stakeholder interactions can be seen in Appendix B-A.

V. Effective Use of Visualization

The types of visualizations used will influence the effectiveness of the solution as a whole. Emotive, powerful messages need to be conveyed through the visualizations while aiming to positively influence people's behaviour. Energy visualization in and of itself does not save energy but can be used as a tool to influence people's perceptions. A number of different visualizations have been selected to best convey the complex information hidden within energy usage patterns. Each visualization technique and the associated intended message is outlined below.

A. Campus Heat Map

Harvard University very effectively used a colour coded map to represent different buildings across campus in their energy visualization website. Combining this idea with the learnings from the eViz project, wherein in heatmaps were shown to be the most effective form of visualization, a heat map superimposed on a campus map is thought to best convey campus wide energy consumption. In this visualization the colour of a building should correspond to its consumption, relative to the rest of the buildings wherein red is maximum usage and blue is the minimum as with standard heatmaps. This is a powerful visualization technique as it can show at a glance who the largest consumers on campus are, while providing a high level overview of all other relative consumptions. Examples and further discussion for this visualization can be found in Appendix C-A.

B. Building Bar Chart

Comparing a buildings consumption against itself in the past is a powerful technique to identify if the building is acting as usual or if there has been some alteration to its behaviour pattern. A bar graph is useful as it lends itself well to resampled time series data. For example, if a week average graph is to be shown then each bar represents a 24 hour time period. You could then compare the past week to all previous weeks in the same chart to provide relative comparisons. Examples and further discussion for this visualization can be found in Appendix C-B.

C. Comparison Line and Area Charts

While a line and bar chart can be used to convey the same information, a line chart is preferable to represent more dense data clearly, such as comparing the loads of 5 different buildings over time. The key inference from a line chart is the ability to identify when one load is higher than the other over time, thus identifying the load profile and footprint of a building and comparing how they perform. Likewise, an area chart is similar to a line chart but stacked and is a effective way to show the general cumulative trend of a set of data. Examples and further discussion for this visualization can be found in Appendix C-C.

D. Sankey Diagram

A sankey diagram is used to represent the flow of electrical energy within the system. Energy is shown to flow from one node (such as a sub station) to the next node (such as a building) where the thickness of the connections between nodes corresponds to the magnitude of the flow. This visualization is powerful in gauging a high level overview of all the interconnections within the system. Examples and further discussion for this visualization can be found in Appendix C-D.

E. Sunburst Diagram

A sunburst diagram is a powerful, visually appealing way to represent nested data, wherein each datum is a subset of other categories. In the context of the university, this could be used to show what campuses consume the most relative to each other and what category of building uses more than others. Nested within this outer layer is the building names and below this the sensor information. Examples and further discussion for this visualization can be found in Appendix C-E.

VI. System Implementation

The implementation is now broken down by first analyzing the high level design overview such as the system goals, design ethos, system specifications, sprint planning and user stories. After that, a technical implementation is proposed that is technology stack agnostic. Then, a potential system architecture is discussed in the context of the created MVP that can be found online here. All source code discussed can be found here. A justification for the development of a custom system can be found in appendix F-A.

A. High Level System Goals

In broad terms, the goal of the system is to provide a framework to easily visualize campus energy data in an intuitive and simple way. The user should be able to view detailed information about the current usage of each and every building on campus as well as view historic information. Emphasis needs to be made on relative consumptions, such as comparing a building against it own averages to detect anomalies/behavioral changes and comparing one building against another to detect other patterns and relative changes. All visuals used should paint a clear picture depicting the situation on campus and should emphasize anomalous behaviour that requires attention. This high level goal is fleshed out fully in the application epics discussed in section VI-C.

B. Development Methodology And Planning

An agile software development mentality has been chosen as the framework to define project timelines, sprints and user stories. Agile software solutions evolve through the project life as an iterative, collaborative effort between software developers and the customer. Due to the explorative nature of this particular project (there is no clear cut best solution or ideal visualization set), an iterative approach is ideal as it allows for continuous feedback and refinement of the core idea. This means that at every step along the development process feedback from stakeholders can be used to improve the product.

The defining success metric in this project is whether it can actually result in a reduction of energy consumption over time. This can only be achieved through an effective and powerful visualization tool set that facilitates a behavioural change. This outcome is most likely to be achieved if the project owner is involved throughout the process and stakeholders/users are constantly engaged to improve the product during the development lifecycle. Lastly, as agile software development focuses heavily on achieving value add at all stages of the development life cycle, a working product will be produced sooner than with other development mentalities.

C. Application Agile Epics

Six major epics have been identified that outline all desired core functionality of the project and have been simplified in the sections that follow below. Full user stories can be found in the appendix D-A. The development of the MVP achieved a vertical slice through each epic to provide some insight into the core functionality that each aims to deliver. The application users are first identified to flesh out the epic roles. There are two key envisioned users within the system: a *user* and an *administrator*.

1) *Users* include students, staff, management and any member of the public. They are able to view the website online and interact with all retrieval functionality. They do not need a login as this information should be in the public domain. They can view all website functionality.

2) *Administrators* include relevant university staff. They have all the functionality of a normal user and they are able to login to the platform and modify the underlying data sources used to generate visualizations. This functionality includes the ability to add new sensors and update existing sensors. Their login should operate with the existing Wits login systems to minimize the barrier to entry.

Next, the application epics are outlined.

1) *High level campus energy consumption comparison*
   As a *user*, I want a high level view of energy consumption for the whole university in a number of graphical formats. I want to gain a holistic insight into campus wide consumption and clearly identify outliers.

2) *Individual building information and comparison*
   As a *user*, I want to be able to find out more detailed information on a particular building by clicking on the map or a list. I want to be able to view live and historic information for this building and compare it against its own averages.

3) *Campus wide detailed building comparison*
   As a *user*, I want to be able to compare and contrast different buildings around campus over time. I want to be able to add as many buildings as I want to a comparison framework and view them in the same graphs over different time periods to gain insight into trends.

4) *Multiple campus support*
   As a *user*, I want to be able to select buildings at different Wits campuses, such as medical and main and compare and contrast them with all aforementioned functionality.

5) *Administration access* As an *administrator*, I want the ability to login with my existing Wits account without having to use any custom credentials. I want to be able to view the current sensor configuration and view the data sources used in visualization generation.

6) *Administration update functionality* As an *administrator*, I want to be able to access the current data entry points and edit the data sources. I want to be able to update building names and locations when new constructions occur and new sensors are added.

D. Sprints and Other Planning

The development process has been broken up into six separate sprints to tackle all defined user stories. Each sprint has been configured to deliver the maximum value at the end of the sprint. All activities undertaken by the developer must result in value add at the end of the sprint to maximize return on developer time. The first two sprints were conducted in the MVP development, as well as all underlying backend and architectural requirments to facilitate future development. The status of each development sprint can be found in appendix D-A.

Github was used extensively during the development process as a version control platform. Outstanding developer stories were also added to Github as issues and it is recommended that future development of the platform utilizes the Github toolset as far as possible as a mechanism to orchestrate the development process. Further Github usage discussion can be found in appendix E-A.

E. Design Ethos

All design decisions have been made around achieving a predefined set of system properties and behaviour traits. These properties govern how the system was designed and what technologies went into the creation of the MVP. Each of these properties and the rational for their usage are now outlined to lay the foundation upon which the technical discussion is based.

Modularity and separation of concerns has been included at all levels of the design. This ethos is embodied from the frontend application, to the backend system and associated devops. The design has been constructed to be generic enough to be implemented with any number of technology stacks to create both the front and backend systems. This enables any number of the many components that make up the final design to be swapped out without affecting others. Additionally, it enables a developer to use the tool set and languages of their preference when constructing components of the system.

Usability of the final system is of high importance. An appealing frontend with intuitive user experience is paramount. To this end, design decisions in the backend must facilitate scalability without hindering application latency to deliver a fast, interactive user experience. Additionally, the frontend must be highly interactive to enable the user to find specific information about different buildings and compare and contrast them. Lastly, the frontend should be accessible to all kinds of users and not only provide value to skilled technical users.

Scalability of the backend system needs to be sufficient to accommodate thousands of different sensors across many buildings and continue to function well into the future. This means that new data points added to the system should not negatively affect the system performance. The scalability of the system should not sacrifice the responsiveness; low latency must be achieved to provide a fluid user experience.

Interoperability with the current architecture is paramount. This means that the final solution should be able to integrate with the current ecWIN technology without needing any modification.

All software used within the solution has been kept to a strict *opensource* policy. Likewise, the final source code falls under an MIT licence to continue the ethos of sharing, software community and to providing an open framework upon which future work can be built.

VII. High-Level Technical Overview

Extensively, the proposed system consists of two distinct layers: an interactive frontend application to display data visualization in accordance with the defined specifications and a backend system that utilizes live and historic data to provide computed analysis, used in the generation of visualizations in the frontend. The layers should communicate by means of a documented API to provide the notion of modularity in the design to facilitate separation of concerns.

A. Proposed Theoretical System

A theoretical system capable of generating the desired graphs and visualizations is now proposed. This system is technology stack agnostic and shows a theoretical design that is capable of achieving the goals. Modularity and separation of concerns was key in this design to enable system upgradability and interoperability.

The proposed system is intended to operate within the current ecWIN architecture stack without requiring the addition of any new software or hardware to be installed. It can also accommodate the addition of new sensor hardware in the future that uses a different platform to ecWIN.

B. Technology Agnostic Backend Solution

First, the backend system used to capture, store and aggregate data is outlined. The goals of this backend system are to provide a framework to:

- Store a large amount of historic time series data while allowing addition of many sensors into the future
- Perform calculations and aggregations over time series data while having high availability to computed results
- Provide live/real time information from sensor data with a maximum of 30 minutes delay
- Incorporate a clearly defined API interface to facilitate modularity for the frontend
- Integration with the current ecWIN architecture without requiring any rebuild of their technology stack
- Ability to onboard new measurement systems in the future without much complexity via the frontend
- Flexibility in metrics produced thereby enabling the creation of new visualizations in the future
- Require minimum hardware and software to run and follow a simple design to allow for the design ethos

Analysis of these requirements show contradictory properties; the notion of scalability (the capability to store a large amount of data) while maintaining high system availability (low latency access to computed results) requires special consideration in the design. If a single database solution was implemented to achieve this, one of the two properties would need to be compromised. If scalability was chosen over availability then each and every query to generate a visualization would require computation over the historic time series data to extract relevant visualization information. For example, if a graph for an average week was required then all historic data points for a particular building would be needed to generate this visualization. This problem becomes even more apparent if one wants campus wide comparisons, such as a heat map that contains summated and aggregate information over hundreds of thousands if not millions of data points. Alternatively, if high availability is required, then pre-computed results are mandatory. However, this prevents the storage of historic data as once aggregation occurs the original datums can't be retrieved.

To achieve both properties of scalability and availability, two different databases are used in conjunction to leverage properties of each. A time series database, optimized for massive datasets, is used to store historic and live consumption data. A computational framework is used to calculate aggregate metrics that are used in the formation of visualizations. These computed results are stored within a seperate database where the emphasis is on availability.

This implementation can be seen as a multiple step process. First, datums are added to the time series database. This either occurs via a periodic web scraper to accommodate ecWIN or directly from a custom sensor configuration that pushes sensor results directly. Next, a computational framework retrieves information from the time series database and performs the required agrigations to generate visualizations. These results are then added to a traditional database. Information from this database should be accessible via an API. Figure 1 outlines a potential implementation of this system.

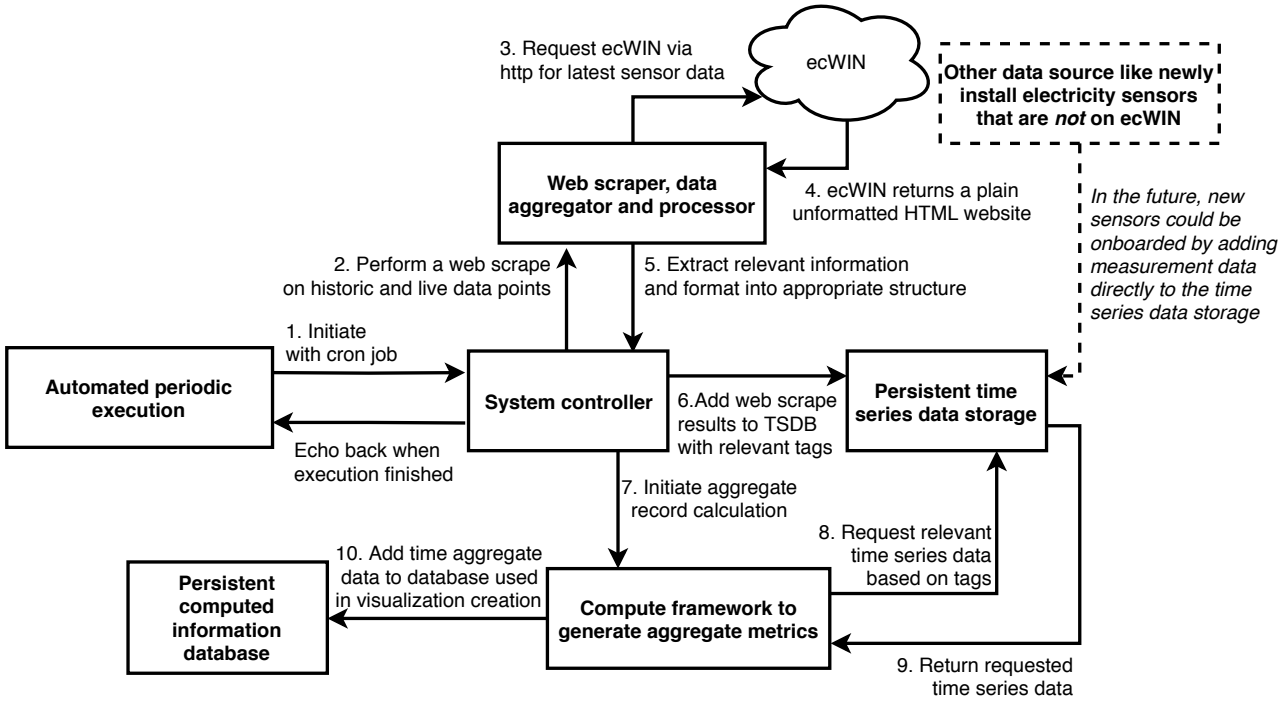

Fig. 1: Diagram depicting the envisioned backend system used to retrieve, store and calculate aggregate data used in the generation of visualizations. Numbers within the labels represent the action's step in the execution process

The notion of data aggregation is important when visualizing time series data as often the time period of information requested in a visualization does not correspond to that of the original sensor data. EcWIN data, for example, is recorded every 30 minutes. If a bar graph is wanted to show an average month wherein each bar corresponds to a day period, the data set will need to be resampled and aggregated to accommodate this. This is where the computation framework comes in. This is used in the process of resampling data into different time buckets to accommodate different requirements for different visualizations.

C. Technology Agnostic frontend Solution

A system is required to visualize the captured and aggregate energy data. This could take the form of a standard desktop application, website or mobile app. As visualization data is retrievable via an API, any configuration is possible. In order to provide a framework that is as accessible as possible a website is proposed as the most universale option. The diagram 2 below outlines a simplified version how this system would connect. The goals of the frontend application are as follows:

- Simple, intuitive user interface with clear instructions removing technical knowledge from reading energy data
- Responsive graphical visualizations that make effective use use of colour in comparing building consumption
- Interactive user experience with fast load times for all user
- Modular, component design enabling upgradability and maintainability
- Mobile optimization to make the platform as accessible as possible to a wide range of users

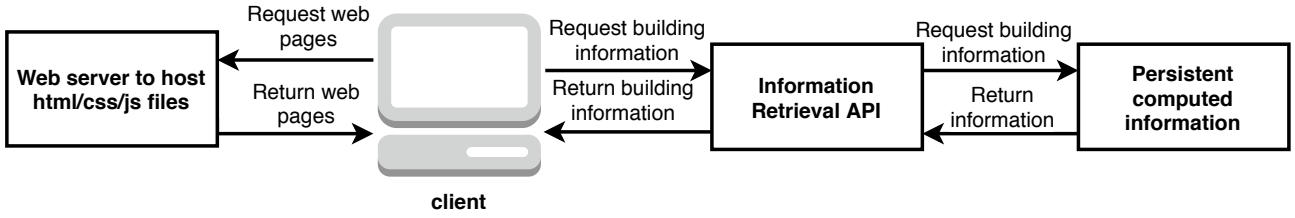

Fig. 2: Diagram depicting the envisioned simplified frontend system showing how the frontend retrieves data from the backend system

A javascript framework is recommended to be used in conjunction with a charting visualization tool set. The javascript framework is used to provide the required interconnection between the API and the charting tool set.

Multiple frameworks and charting tools are possible in this implementation and could be acceptable.

A technology stack agnostic high level depiction of the proposed system has now been outlined. Next, a working system architecture configuration is proposed that implements each piece of the software solution. This culminates in an operational Minimum Viable Product (MVP) that containers all key logic defined in the proposed high level design. This MVP can be found online at witsecop.co.za and source code can be found on Github here. All core architecture in the high level was implemented at some level to provide a vertical slice through all components. This means that every component discussed in the design that follows has been implemented at some level. As with the high level overview, the backend system is followed by the frontend. Throughout this discussion, any open source software used will be linked to the respective repositories.

VIII. Minimum Viable Product Implementation: Backend System

The backend implementation (source code can be found here) utilizes a number of different technologies and languages to achieve the desired system properties. Extensively, the backend acts to collect, aggregate, process and store time series data in a way that can then be visualized by the frontend.

A. Docker Configuration

As with the development of all other layers of the technology stack, the ethos of separation of concerns was embodied in how different services and backend components communicate through the use of Docker containers. Every discrete service discussed in the following sections reside within their own container. These containers communicate via a Docker network configuration and specified ports are open on specific containers to allow for outside communication. In total, six Docker containers were used to host all key architecture. Figure 3 shows the implementation detail and each section that follows refers to the respective container for each service. For a full rational into why Docker was used at such an intimate level of the project, see appendix F-C.

B. Time Series Data Collection

In order to integrate with the existing ecWIN configuration, without needing to modify the underlying technology, a web scraper is required to request the most recent time series data periodically. This can be implemented using Python in conjunction with a Selenium web scraper to retrieve sensor data. Scraping automation can then be initiated automatically by either using a crontab or a Unix Daemon. This process is outlined in Container 6 within diagram 3 and represents steps 1 through 5 in figure 1.

The MVP implementation did not fully build out this scraping component but rather utilized a series of historic time series data points captured within three excel spreadsheets dating back to 2015. These spreadsheets were generated using the same process that would in future be automated to capture live data on a recurring basis. Code used to retrieve, format and store the historic data from the aforementioned spreadsheets can be found within a Jupyter Notebook that can be found here. This notebook contains all data storage, formatting and aggregation code used in the calculation of aggregate metrics.

Future implementations would require the ability to integrate with new sensors that are not part of the ecWIN platform. These sensors could directly store their data within the time series data storage solution discussed next.

C. Time Series Data Storage

The time series data is then required to be stored within a time series optimized database to be used in computations to generate aggregate data for visualizations. There are a number of choices when it comes to this database design configuration. Firstly, a traditional SQL or noSQL database could be used. However, these configurations are not optimized for storing large amounts of time series data and do not perform well over scale as has been shown in a number of benchmarks [9], [10]. Additionally, neither a traditional SQL or noSQL lend themselves well to being queried over long time frames wherein a large data set is returned. This was shown in [9] wherein a time series database was shown to perform large time based queries 1400x faster than MongoDB. Additionally, time series databases like InfluxDB have been shown to use 64x less disk space and deliver 3x faster insertions compared to MongoDB architectures[9]. The selection then needs to be made for an appropriate time series database. The three major contenders are OpenTSDB, InfluxDB and TimescaleDB. All offer scalability, high querability and are all open source with a large developer base. InfluxDB was chosen as the most suitable time series database and a full rational into this choice can be found in Appendix F-B.

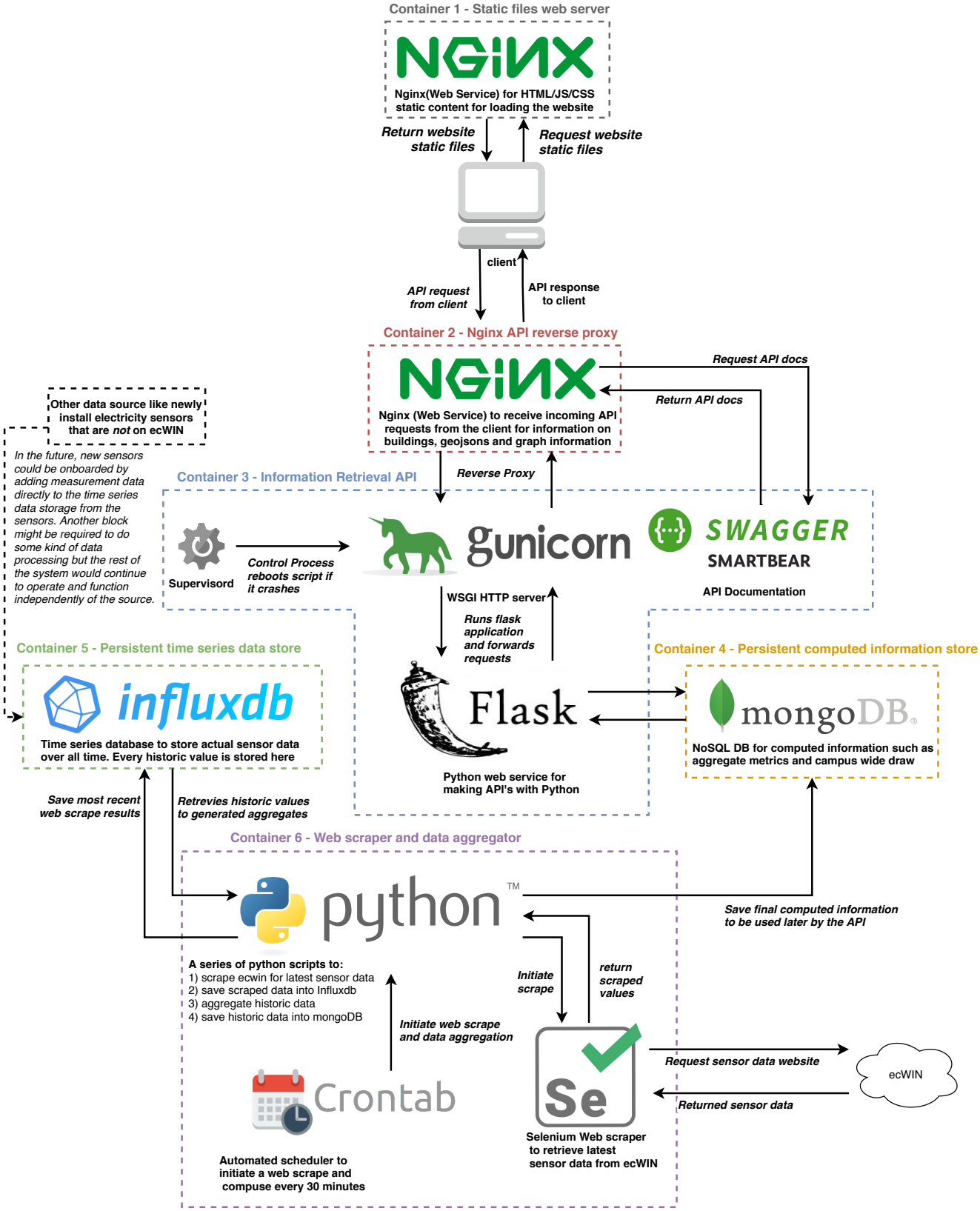

Fig. 3: Diagram depicting a number of Docker containers used to server the website content. All containers reside on the same server

The MVP implementation, as outlined in the Jupyter notebook, simulates the retrieval of data from ecWIN and then writes data into the InfluxDB data store. Data is stored using the building name, building number and sensor name as tags enabling querability by each property in the future. This process is represented by the retrieval of data from container 6 and the storage of data in container 5 in diagram 3. Step 6 from Figure 1 is compleated here.

D. Time Series Aggregation and Computation

At this point, a system to retrieve and store time series data has been defined. The next step is to perform aggregation and computation over the datasets to generate the required information for all visualizations and the

campus map. This process is performed within Python and has been fully implemented within the Jupyter notebook. This aggregation process occurs within container 6 and data is stored into container 4 from diagram 3. Steps 8 and 9 are represented from Figure 1.

Each visualization requires a particular kind of metric to be generated in a particular format. For example, to generate a graph that depicts the average week of a building over all time requires resampling of time series data into 24 hour bins that are then averaged over all days. This requires a high degree of data processing and resampling. To achieve this, the Python Numpy and Pandas libraries proved very useful. Information from the time series database is read in based on a query over a particular duration for each given metric required to be computed. This information is then loaded into a Pandas dataframe to facilitate computation over the time series data such as resampling and aggregation.

There are two main kinds of aggregations and computations required for the frontend. Firstly, the calculation of colours, based off relative magnitudes, used in the formation of the heat map. These colours need to correspond to the consumption of one building, relative to all other buildings. These are generated by iterating over all buildings in the dataframe to identify key values, such as maximum draw for a particular period of time. These values are then later converted to an RGB colour value, wherein the maximum value of all buildings is used as the upper bound (represented by a red colour). The second type of aggregation required is used in the formation of relative line and bar graphs. This process is done by resampling, and then averaging over different time periods.

As each visualization requires its own data source (some can be re-used but a week graph requires different data than a year graph, for example) different computation is required to be performed for each data set type. This provides the ability to generate new visualizations as and when needed by simply adding the computation type to the Python script used to generate metrics and then adding the access to the API.

E. Fuzzy Joining Building Names to Sensor Names

Geojson information for all buildings is defined by specifying GPS coordinates on a map. Each building has an associated set of properties, such as the building name and building id. The computed data discussed in the previous section is generated by performing a fuzzy join using difflib between user specified campus names within the geojson and the most similar sensor name from the ecWIN dataset. This removes the manual element of linking different buildings to sensor names.

F. Computed Data Storage

Once each computed metric has been generated it is stored within a MongoDB. Different Mongo documents are defined for each key visualization type. MongoDB was chosen over a SQL database due to its simplicity of usage and querability. Complex joins and links are not required at any point for the visualization and ease of use and consistent high availability are of higher importance. Additionally, MongoDB's json stores closely mimic the data structure used within Python's json notation enabling easy writing to the database. This json returned from the MongoDB also easily integrates into the format required by the frontend visualization requirements. Lastly, as MongoDB stores json documents, adding the Geojson required to generate interactive maps becomes trivial. The full data structure used for the backend computed data store can be found in appendix F-D This computed data is stored within container 5 from Figure 3 and represents step 10 from Figure 1.

G. API Implementation and Documentation

In order to make the computed results accessible to the frontend web application a Flask API was created to facilitate restricted and predictable access to the MongoDB. Swagger was used to generate API documentation to ensure standardization across all endpoints and to provide a framework upon which future developers can build. This swagger interface provides a publicly accessible UI that defines the specifications for all API endpoints. Futher discussion into the API can be found in appendix F-E.

Effectively the Flask API provides a standard interface that any application can use to gain access to computed values through a predictable, document format. This API could have been replaced by simply leaving the MongoDB accessible to the frontend application but this removes any notion of data sanitization and does not provide a consistent way to interface with the backend data. The server configuration used to provide this API is discussed in next section.

H. API Nginx Reverse Proxy

Using a Flask API all on its own is not appropriate for a production system. At a high level, Flask allows you to build out the logic that defines how the server responds to a certain HTTP request. Flask provides a development server with useful features but it is only intended to be used in a development context as it can't handle a large number of simultaneous connections in a robust way.

In order to make the Flask API scale to a production appropriate level, a reverse proxy was created using Nginx. The Flask process was run using a combination of Gunicorn and Supervisord. Flask is build upon the Web Service Gateway Interface (WSGI) which defines the interaction between application logic and a web server. Gunicorn provides an execution environment for the Flask logic to operate in, providing a mechanism to handle concurrent connections using the Flask WSGI. Gunicorn accepts HTTP requests and then calls the application logic to get a response. Supervisord is used to monitor the state of the Gunicorn process and reboot it if it fails.

The Nginx reverse proxy is used to provide scalability and API isolation. Nginx has the effect of sheltering the Python execution environment by handling all HTTP requests before forwarding them to the Gunicorn process. Additionally, Nginx allows for dynamic load balancing, enabling traffic to be sent to multiple servers from one entry point. Nginx also provides a shield from DDOS attacks by preventing a burst of traffic to the Flask container.

The Gunicorn, Supervisord and Flask API should reside within the same system and have been Dockerised as one service as depicted in container 3 in Figure 3. They communicate with the Nginx service via a predefined port, enabling separation between the two components. The Nginx service can then run within its own container, depicted in container 2 in Figure 3.

I. Python Virtual Environment

The Python Flask API requires a number of libraries to operate correctly, such as Numpy and Pandas. In order to make the development process as simple as possible and to create a reproducible production environment, a Python virtual environment was created with virtualenv. This virtual environment is used to store all the application packages and associated configurations. This means that when the Docker container running the Python Flask is initially set up it will first create a virtual environment and then install all packages required in the same way that was done in the development environment.

J. Encryption and SSL

Even if a website does not have any sensitive information stored on it, it should still use HTTPS and SSL to provide critical security and data integrity. Additionally, HTTPS is a requirement for many modern browsers and is needed for a progressive web application. Added to this, when the future implementation of administration login is added SSL will become mandatory. Having SSL also improves website search engine rankings. If this project aims to gain any real traction, this will become important.

In order to implement SSL, given the proposed backend technology stack, the Nginx server should be used to serve SSL certificates to the client to provide an encrypted HTTPS communication channel. The basic premise of this was implemented in the Docker containers by mounting a volume containing the signing keys for the SSL configuration. Full SSL and thus HTTPS was not implemented but it is not overly complex to do from the current implementation point. A more detailed discussion on SSL implementation can be found in Appendix F-F. The frontend application will also require to serve SSL certificates as both the API endpoints and website should be encrypted. The process to achieve this is the same via mounting the certificates into the Nginx Docker container then serving them to the client on request.

K. Administration Login

To provide the system with administration login Security Assertion Markup Language (SAML) used in conjunction with Oauth provided by Auth0 and leveraging Json Web Tokens(JWT) to authenticate API calls is proposed. While this stack might seem complex, it provides some benefits that can't be achieved in any other configuration, such as utilizing the existing Wits login system via Oauth. Additionally, JWT's provide a very secure API verification framework that does not require additional authentication or overhead. A full detailed discussion on how this stack operates as well as a user authentication sequence diagram is shown in appendix F-F. This authentication functionality was not fully implemented in the MVP. The basic store component to provide access to user session information was created as a starting point for future development.

L. Updating Backend Sensor Endpoints

All building information is defined by the campus geojson files. Only if a building has been included within a geojson file is it included on the map and subsequently its consumption can be plotted. If a new building is added to the geojson file it will be fuzzy joined to an associated sensor. Clearly, this assumes that there is an associated sensor for the building but this is a strong assumption at this point wherein the building sensor coverage is more extensive than that the mapped buildings within the current geojson files. This means that there are a number of unmapped buildings that once mapped will be joined by the compute script.

As and when new sensors are added to either the Wits ecWIN system or if information is added directly by sensors to the time series database, they will enter the pool capable of being joined onto building names from the geojson files. This means that the only step needed to add new buildings to the system is to update the underlying geojson files with new buildings and new building names. In order to achieve this update process, once a system administrator has logged in they should be able to view the current geojson files via a geojson file editor, add new buildings to the map and label them accordingly.

The update functionality has not been fully implemented but rather requires a manual update of the geojson file. Extending this to work automatically as previously discussed will required the creation of a new restful endpoint within the Flask API to update the geojson files.

IX. Minimum Viable Product Implementation: Frontend System

At this point, a full backend system has been described and implemented, capable of analyzing time series data in order to extract relevant and required information for visualization. Next, the recommended frontend technology stack is proposed. A full breakdown of all technologies used how how they interconnect can be seen in Appendix G-A in. A series of screenshots outlining all implemented functionality can be found in appendix I. All front end source code can be found on Github in a documented reposotory here.

A. Javascript Framework Rational

Vuejs was selected as the javascript framework of choice to provide the mechanism to interconnect the application frontend components to the API backend. For a full rational into why Vue was select over other frameworks like Angular or React, see appendix G-A. While a javascript framework is not strictly required to build a reactive, modern website it makes the underlying javascript complexities a lot simpler, such as state management, routing and separation of concerns through component based design. Additionally, the integration with third party libraries becomes easier. Moreover, the use of a framework makes synchronization between the state and the user interface posible. This synchronization is hard to achieve without the use of a framework. Lastly, a framework allows for the creation of complex yet efficient and easy to maintain user interfaces that would not be realisable with vanilla javascript.

B. Javascript State Management

Effectively, the frontend acts to pull information from an API and display it. In order to provide modularity and upgradability, a state management pattern was embodied to act as a centralized store for all component information wherein rules are used to ensure that the state can only be mutated in a particular fashion. This means that the mechanism used to retrieve information is decoupled from how information is displayed on the frontend. A full explanation into how this was achived can be found in appendix G-B.

C. Component Based Design

In order to make the frontend as modular as possible to facilitate upgradability, easy addition and modification, every element within the frontend is a series of nested components. This means that the frontend can programmatically render selective elements based on the users interaction. For example, if the user selects to include a particular building within a chart a new component is rendered within the frontend Document Object model (DOM). A full analysis into this component design and a diagram depicting how all key components fit together can be found in appendix G-C.

D. Mapping Framework

Interactive maps are required as a key part of the user frontend to show relative building comparison. To achieve this, the open source graphing framework Leaflet was used. In order to integrate leaflet with Vue within the component oriented design paradigm the Vue2Leaflet package is recommended to provide flexibility of leaflet within the Vue context. Building and street names are retrieved from openstreetmap.

E. Visualization Framework

There are a number of very powerful javascript plotting and graph frameworks. Chart.js, D3.js and plotly.js were all considered for the design. Plotly was chosen over charts.js due to its wider range of charts allowing for future extension to more complex configurations. Plotly was chosen over D3 due to its relative simplicity, offering the ability to make complex, mixed plots without requiring the customization required for a D3 plot.

X. DevOps and MVP Deployment

The last component of the implementation detail, following the backend and frontend implementation, is the devops and system deployment. Where the server used to run the Docker containers for the front and backend is hosted will define the system latency and resultant system responsiveness. Vultr was chosen as the cloud deployment infrastructure to host a Virtual Private Server (VPS) to run the Docker containers. While this server is not in South Africa, their Amsterdam based servers have quick enough response times to not warrant the additional cost of hosing the application within South Africa. There is no personal information stored on any of the databases used in this project and so Popi compliance is not important[11]. A further discussion on how Vultr was configured and how Portainer was used to manage the Docker containers can be found in Appendix H-A.

The domain witsecop.co.za was registered using Hostking for cheap .co.za domain names. Cloudflare was chosen to act as a Content Delivery Network (CDN) to increase system responsiveness and decrease page load times. A more detailed discussion on the networking configuration can be found in appendix H-B.

XI. Implementation Critical Analysis

The proposed solution has now been rationalised and implementation details discussed. The solution is now critically analysed in the context of an engineering solution to identify its social, economical and technical applicability in each respective problem domain.

A. Social Analysis: System Applicability and Stakeholder Satisfaction

The created MVP serves the key functionality outlined in the system goals as to provide a platform to visualize, compare and contrast energy consumption on campus. The final envisioned system, once all user stories have been fully realised, will represent an extensive comparison platform that will hopefully provide valuable insight into campus usage and behavioural trends. With that said, it is impossible at this point to gauge the successfulness of the project unless future stakeholder engagement is performed. All relevant users of the system, from general university students to administration need to be approached and their feedback needs to incorporated into the design. Additionally, a pilot needs to be run for a period of time wherein usage metrics are captured and analyzed to gain feedback into how the platform is being used to hopefully drive further user adoption.

B. Social Analysis: Ongoing Evaluation

The true success of the platform can only be gauged after it has been in operation for a number of years. Evaluation needs to verify if the creation of the platform resulted in a reduction in campus energy consumption, provided a positive behavioural change or if the platform has been useful to university administration. Clearly, one can't quantitatively say that if the platform has been in operation for five years and the energy usage on campus has reduced over those five years that the energy reduction has been as a result of the platform. This is confusing correlation and causation. It is therefore important that an ongoing stakeholder engagement is performed periodically to consistently gauge the usefulness of the platform from multiple perspectives. It is therefore recommended that for the first two years of the platform operation meetings are conducted with key stakeholders every six months. These stakeholders include university administration, ICTS and a random selection of university students. Specific predefined questions should be asked to gauge the usefulness of the solution and to identify possible refinements. After the first two years of operation, stakeholder evaluation can be reduced to once per year for the remainder of the projects life span. This ongoing evaluation process is critical to maintaining the relevance and usefulness of the application. This ongoing evaluation process will however incur a cost as someone will be required to run the stakeholder engagements and a developer will be required to implement the required changes, if need be. Ideally though these changes will be small as the final product at the end of the development process will be close to ideal due to constant feedback and stakeholder engagement during the development process.

C. Economic Analysis: Implementation and Operational Costs

Implementation cost will depend on the developers chosen to create the final product. With that said, a cost projection can be created given the average developer cost in South Africa, coupled with the outstanding sprints and user stories. Effectively, one third of the development process has been completed in two one week sprint. Assuming that the rest of the development can occur in the remaining 4 developer sprints, an intermediate full stack developer should be able to complete the project in a month of full time development work. This could be done inhouse by the university as to not incur any direct costs or it could be outsourced to a development house to conclude the development process. Either way, the effective cost to complete the project is expected to be between 30 and 40 thousand rands based off average developer salaries within South Africa[12].

Ongoing operational costs fall under two main sections: technical infrastructure costs, including servers and domain costs and personal costs, such as a qualified person responsible for the ongoing maintenance of the platform. Infrastructure costs are very low due to the architectural decisions made enabling the website to be hosted on a very barebone web server. The web server used to run the platform could be hosted internally within the university but this means that a dedicated machine would be required and the additional administration, electricity and internet costs then need to be considered. It is easier and cheaper therefore to host the web server in the cloud wherein these costs can be mitigated. Cloud infrastructure hosting at Vultr works out to a total of R1728.36 per year for a machine with 2GB of ram, an intel Skylake CPU and 40GB of solid state hard drive. Given the technology stack, this machine can accommodate a realistic number of users, comfortably accommodating hundreds of concurrent connections. If the machine is found wanting and higher end specifications are required, the cloud infrastructure can be scaled vertically as far as required, an impossibility if hosted internally. The witsecop.co.za domain was registered with hostking with a yearly cost of R80. The personnel cost is harder to calculate as it will depend on the work required to maintain the system. With that said, due to how the architecture was configured using Docker containers, system maintenance should be close to zero and so the associated cost should be kept to a minimum.

D. Economic Analysis: Potential Electricity and Money Savings

It is hard to accurately calculate the potential savings from a hypothetical system. With that said, some basic calculations can be done to calculate the potential monetary savings for the university. Harvard University can be used as a case study to find expected possible reductions after introducing an energy saving intuitive. In 2008, Harvard established an aggressive initiative to drastically reduce their greenhouse gas emissions 30% by 2016, from a 2006 baseline[6]. After 8 years into the program Harvard was able to drop their consumption by 23%. If one then assume that Wits can reduce its consumption by half of what was attained at Harvard, a total of 16 million Rand could be saved per year. These calculations are based off the 2017 annual report claiming that Wits spent 141 million Rand last year on electricity[13].

E. Technical Analysis: System Maintainability, Upgradability and Scalability

The use of modularity in system design through Docker containers and separation of concerns makes system maintainability and upgradability easier. There is however a trade off in using Docker and other newer technologies in that future maintenance and upgradability will require technical knowledge within these technologies resulting in potentially higher personnel costs. System upgradability should also be achievable as individual components can be upgraded independently of the rest of the system. System scalability was achieved through appropriate selection of database configurations and architectural design decisions. This stack was able to easily handel one years worth of sensor data for 240 sensors while providing high availability and reliability. Future benchmarking of the solution is required to ensure that the solution is future proof, capable of handling the addition of any number of buildings and sensors while remaining preformant.

F. Technical Analysis: Software Stack Future Support and Applicability

A number of newer technologies were proposed to be used within the core architectures technology stack, such as Vuejs and Docker. Newer technologies always run the risk of being unsupported in a few years resulting in a lack of developer resources to facilitate continued maintenance on the platform. To this end, the packages chosen when constructing the technology stack were picked to have a high developer following at time of writing with no signal component within the stack showing a glaring lack of support into the future.

G. Technical Analysis: System Interoperability

The ability for the proposed solution to integrate with existing infrastructure as well as the infrastructure of the future will be a defining characteristic of its success (or failure). The modularity of the database design provides

the ability for different data sources to be used in the computation of metrics used for visualizations. This offers the ability to add in new data sources in the future, such as sensors that are not within the ecWIN ecosystem. The use of an open, documented API will hopefully provide future projects with valuable information to aid in their analysis.

The modular design approach used for data capture and storage will also hopefully facilitate the ability for the system to integrate with different time series data sources, such as water consumption for different buildings. The underlying logic used in defining how data flows within the system is not only limited to electrical energy data and should in theory be able to accommodate any time series data.

H. Technical Analysis: System Licence Considerations

A requirement was set for all technologies used in the software stack to be free and open source. This was achieved at all layers of the design and the final MVP source code follows the same licencing, falling under an MIT licence. This prevents vendor lock in, such is the case with the current ecWIN system. A potential risk is that one of the key architecture components used in the final solution will become closed source in the future. This would pose a problem as that component would then need to be replaced. The likelihood of this occurring however is low as all components chosen have a large following in the open source community meaning that even if a components core architecture became closed source the open source development community would most likely continue the development of the open sourced components.

I. Technical Analysis: Data Integrity

One key downside to the proposed platform is that there is no way to validate the integrity of sensor data and no way to infer consumption values for missed data points. If there is an incorrectly accounted for consumption value from ecWIN, it will be included in the database and sampled accordingly. There is nothing that can be done to prevent this as the compute framework will never be able to detect an invalid measurement.

XII. Future Improvements

There are a number of interesting future extensions to this solution, such as plotting water usage on the same map as energy, as discussed in the interoperability analysis. If this idea is taken to its logical extreme, all relevant campus data could be stored and shown on the same platform. This could potentially provide even further insights into usage and behavioural patterns on campus.

A fault logging platform wherein students and staff alike can logon and report system maintenance needs, such as a burst water pipe or leaking tap would also be extremely useful to provide quicker response times to resource wastage.

Gamification of certain application elements could also be effectively used. This would involve creating monetary incentives for a changed behaviour.

XIII. Conclusion

Energy visualization has been shown to be a powerful tool in identifying trends and patterns that are normally obscured within energy data. Additionally, energy visualization is useful to instill a positive behavioural change towards energy usage. A visualization tool set to aid the university in a transition to a greener future was conceptualized, designed and partially implemented. This visualization tool set proved to be effective in conveying consumption information and identifying usage behaviours around campus. To truly define its success, future stakeholder engagement is required.

APPENDIX A

The solution presented in this project has the potential to have a long term positive impact on social, economic and environmental fronts. That being said, as with every engineered solution, the possibility of negative impacts must be considered. This short report compares and contrasts the advantages and disadvantages of the proposed system from a non-technical perspective, reflecting on the impact of an engineering solution within society and discusses how negative impact has been ameliorated.

The design and implementation of the proposed solution has been structured around utilizing the most valuable data to achieve the largest impact, resulting in the maximum possible reduction in energy consumption around campus. This project alone cannot address the inefficiencies within our university and in and of itself will not result in a drastic reduction of energy consumption. To realise real world change, a social paradigm shift needs to occur wherein a conservative mentality is taken towards energy usage in all aspects of campus life. Staff and students alike need to become accountable for the energy they use.

Any real world reduction in energy usage on campus requires a campus wide initiative with support from multiple faculties and departments. The energy usage visualization tools presented in this project can aid in this green initiative by providing visual feedback into usage trends and patterns. Additionally, with the aid of this tool set, green initiatives can be empirically shown to reduce energy consumption on campus over time thereby providing justification for future initiatives. Lastly, this tool set can help mitigate unnecessary energy wastage around campus by providing a campus monitoring framework to identify unusual consumption patterns.

A. Social Positives Impacts

The presented solution can hopefully instill a greener mentality amongst staff and students. Through the visualization of energy consumption, accountability to one's own role in energy consumption can become posibible.

Additionally, the solution presented could be used to provide some degree of accountability for future green initiatives. If the university commits to reducing consumption by X amount in the next Y period of time, it is currently impossible for students and others to hold the university accountable if they do not meet these goals as there is no public platform to monitor campus wide energy usage. Through the implementation of the proposed solution, students, staff and interested third parties can view accurate live and historic energy usage and can thus hold the university accountable for its consumption.

B. Economic Positives Impacts

As a result of potential reductions in energy consumption, the total monetary expenditure on energy can be reduced, saving money for the university in the long run. The knock on effect of reduced spend can be felt all the way down to the university fees spent by students at the end of the year. This economic incentive proves to be powerful in changing people opinions on consumption; reducing your usage is no longer just a good thing to do because it is environmentally friendly but rather can actually save you money. Additionally, South Africa will soon be subject to carbon taxes from January 2020 wherein "Organisations who don't take care ro reduce their carbon footprint will face punitive measures" [2].

C. Environmental Positives Impacts

Any reduction in energy consumption will result in the university as a whole becoming greener. It is important that Wits as an institution takes the lead moving towards a more sustainable future. Green initiatives require substantial leadership and the university is well positioned to instill a more environmentally friendly mentality in our immediate community.

D. Potential Negative Impact

The most likely and impactful negative risk is a lack of user adoption. If no campus wide initiative is taken towards a greener future, the implementation of the system proposed in this project will have a drastically reduced impact. That being said, even if no campus wide initiative is undertaken, the system proposed will result in public awareness towards the inefficiencies within campus and can thus hopefully begin to change people's mindsets towards energy usage.

If the usage of the platform is drastically less than what is expected, it is possible that the cost of the development of the platform will exceed the monetary savings from the its implementation. While this is a risk, the design of

the solution has been structured around simplicity and maintainability resulting in the development cost being as low as possible while ensuring system maintainability into the future.

Consumers highly value the social and ethical leadership of tertiary institutions. If Wits implements the system proposed and does not change their consumption pattern accordingly, the created exception will not have been meet resulting in social dissatisfaction towards the institution.

APPENDIX B

A number of stakeholders were identified through the course of the project as key individuals that will be defining in the success (or failure) of the project. Tailoring the Final system to be congruent with their requirements is vital in the adoption of the system. These stakeholders should be continuously engaged into the future along the development process to ensure that the final product meets their requirements.

A. Stakeholder Engagement: Mr Jason Huang

Jason Huang is part of the planning and development team for the university. This department is responsible for the development of the university, such as rolling out new projects. All major infrastructure development done at the university are implemented by this department. This makes Mr Huang a key stakeholder as he is part of a department than will be instrumental in any kind of institutional consumption change.

A meeting was conducted with Mr. Huang to identify his requirements. Mr. Huang notes the importance of energy visualization and identified a number of key requirements that will help him and his department make informed decisions with regards reducing energy consumption within the university. These requirements are as follows:

1) A strong emphasis was placed on the need for a comparative tool, that allows a user to compare two buildings energy consumption together. This information was critical in choosing visualizations for the MVP.
2) A system to enable buildings of different sizes to be compared to identify load profiles.
3) A tool set to detect anomalies that deviate from normal consumption behaviour.

Other stakeholder engagements were conducted, such as an interview with Business Intelligence Services (BIS). However, their did not display much interest into a visualization platform and so it was decided that perhaps they were not part of the target intended audience. As a result, the tools created should be more heavily focused towards those that would use it, such as Mr Huang and his department who showed much enthusiasm for the possibility of energy visualization.

Stakeholder engagement should not be seen as a once off process and should continue into the future as an ongoing process. Additional stakeholders should be identified to gauge the success of the project. This idea is outlined in Critical analysis section XI-B.

APPENDIX C

Example visualizations from those discussed within the Effective Use of Visualization section of the report are discussed below. Each section shows screenshot or diagram to convey the point of that visualization. All figures were either created new or taken from MIT licensed websites.

A. *Effective Use of Visualization: Campus Heat Map*

Heatmaps are very powerful at conveying information that relates to geographical locations, such as building relative consumption. Harvard effectively used this vizulization in their two websites, as in Figure 4 and 5 where buildings and building groups are shown on a map. Insparation was taken from these implementations to create a custom heatmap wherein the building colours represent relative consumption. This can be seen in Figure 28.

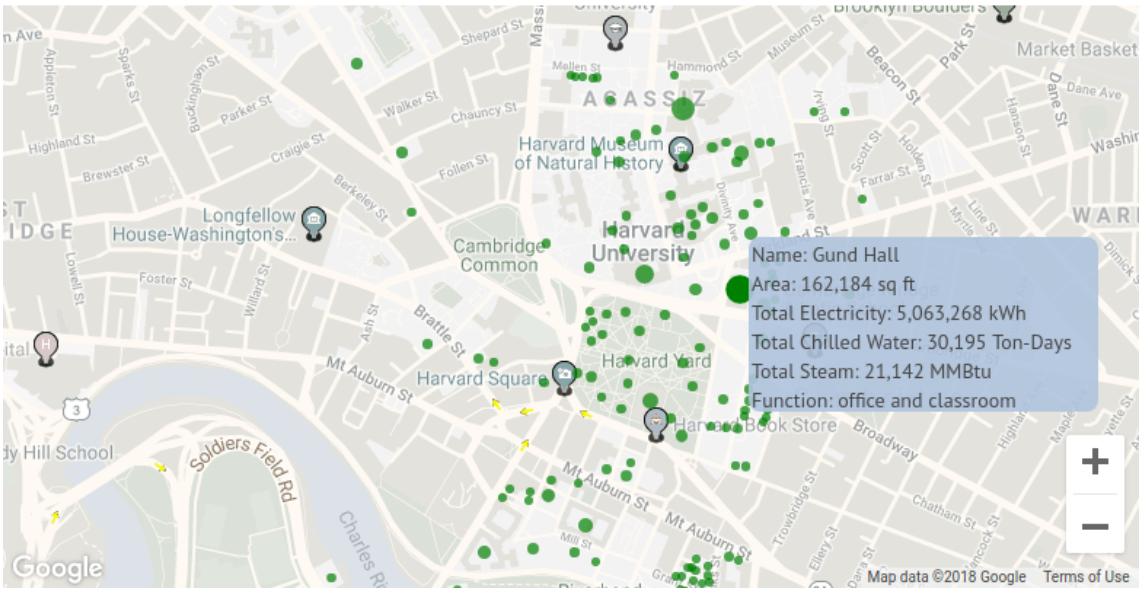

Fig. 4: Screenshot of Harvard website showing different building locations

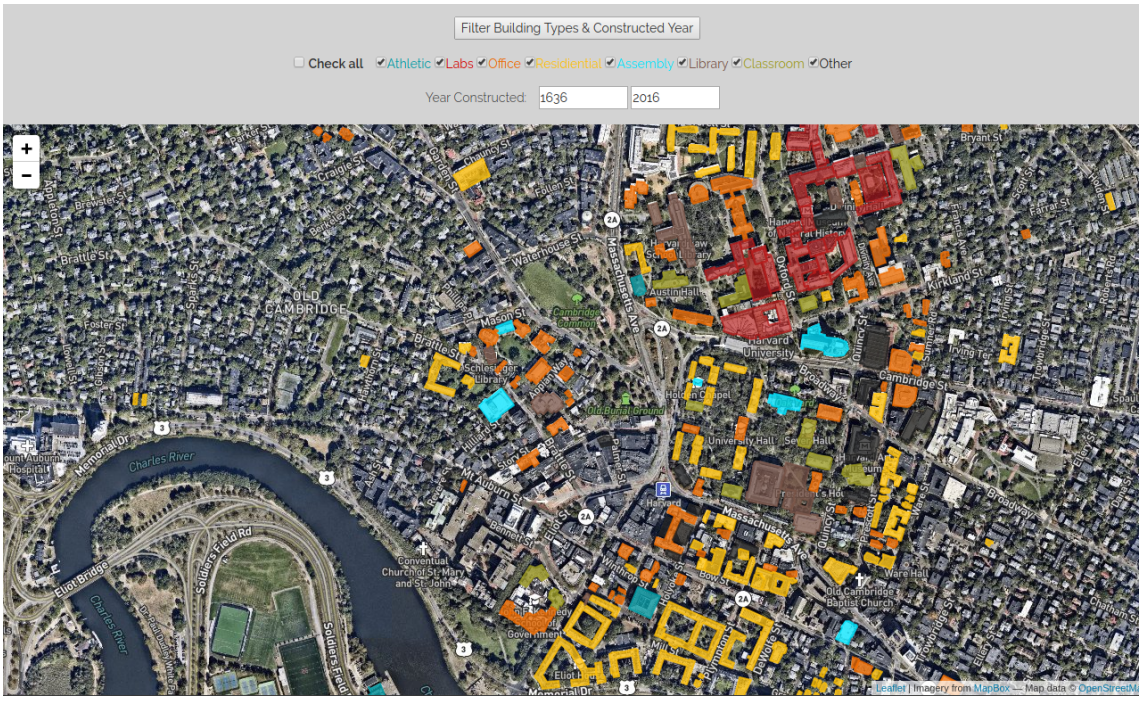

Fig. 5: Screenshot of Harvard website showing different building groupings

B. Effective Use of Visualization: Building Bar Chart

Bar charts are very effective at communicating re-sampled data wherein the bar width corresponds to the re-sampling duration. They can clearly emphasis a change in trend over time. Figure 6 and 7 both show clear patterns that can be used to identify a particular behaviour. Figure 6 shows the Chemistry Building over the course of the last day of available readings. You can clearly see that this last day shows a different consumption pattern the normal day pattern. The last day shown here is the 18th of august 2018. This was a Saturday so the much flatter consumption pattern is expected. Likewise Figure 7 shows the William Cullen Library over the course of a week. Clear drops in the weekend consumption are visible.

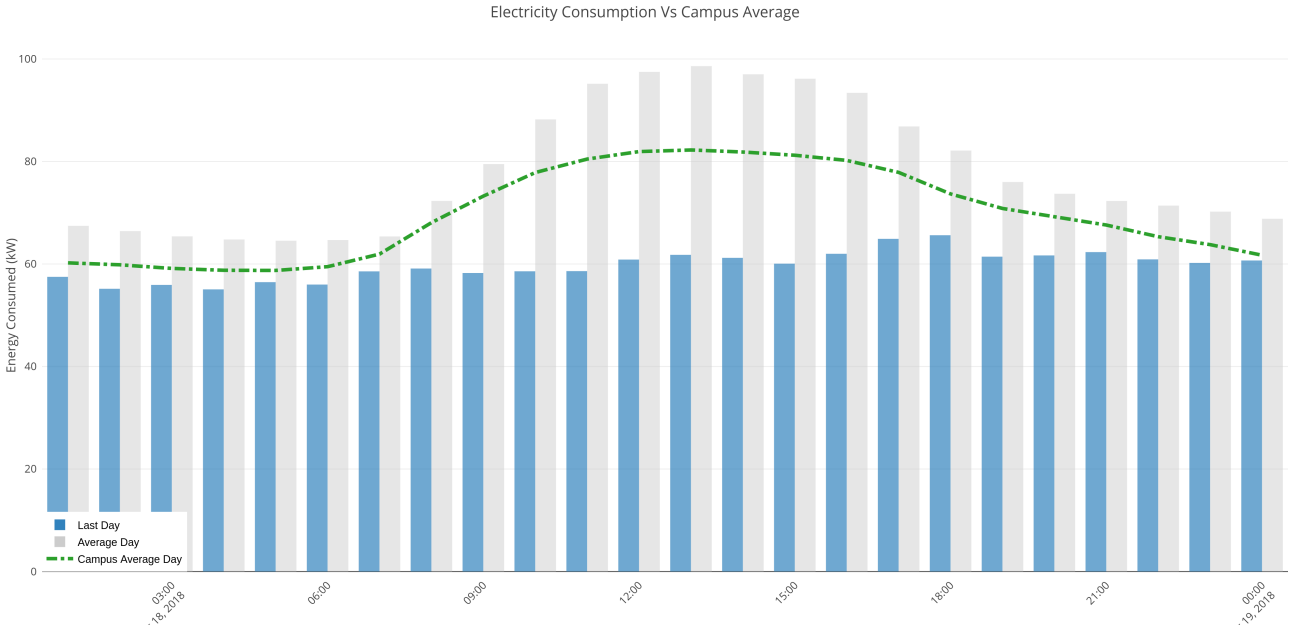

Fig. 6: Plot of the Chemistry Building clearly showing a consumption pattern different during the least and average day

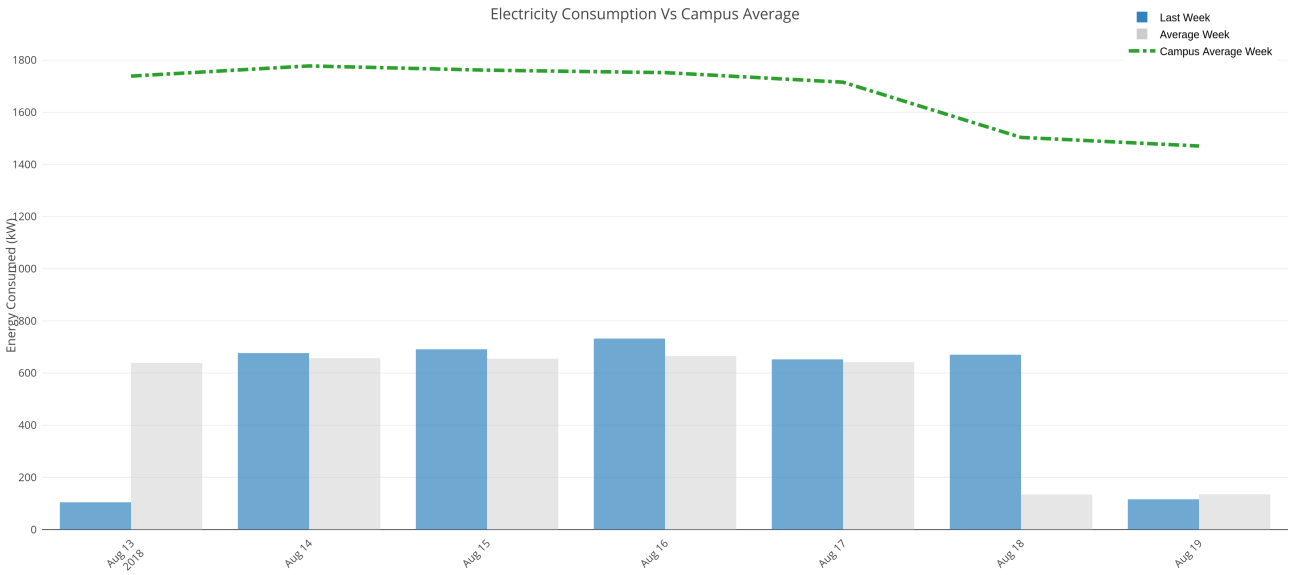

Fig. 7: Plot of the William Cullen Library showing a drop in consumption over the weekend

C. *Effective Use of Visualization: Comparison Line and Area Plot*

To compare multiple buildings against each other, a line chart is more appropriate as it can represent dense data. Figure 8 shows the average consumption of a number of buildings. This figure makes it clear when one building is consuming more energy than another as the lines cross. Figure 9 shows a area plot. This is the same information as the line plot 8 but information is now stacked. This makes it easier to identify the cumulative behavioural pattern of all the loads as well as to identify the maximum total draw of the loads.

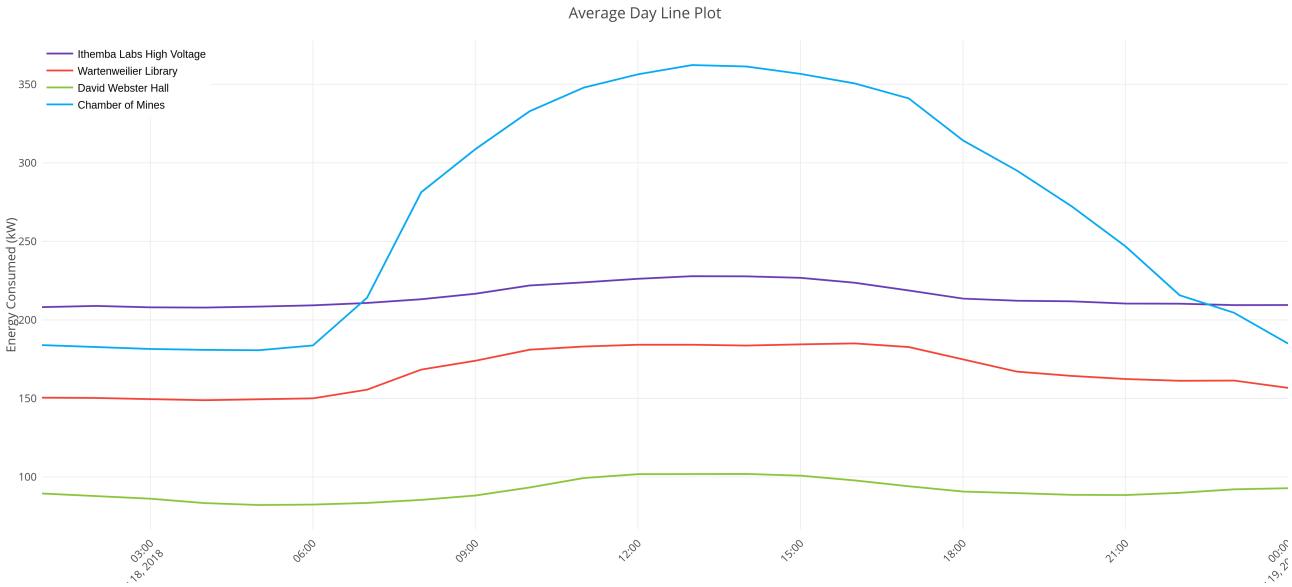

Fig. 8: Line Plot of a 4 buildings consumption clearly showing the relative behavioural changes of different buildings. Can compare the magnitude of different buildings and identify when one uses more than another

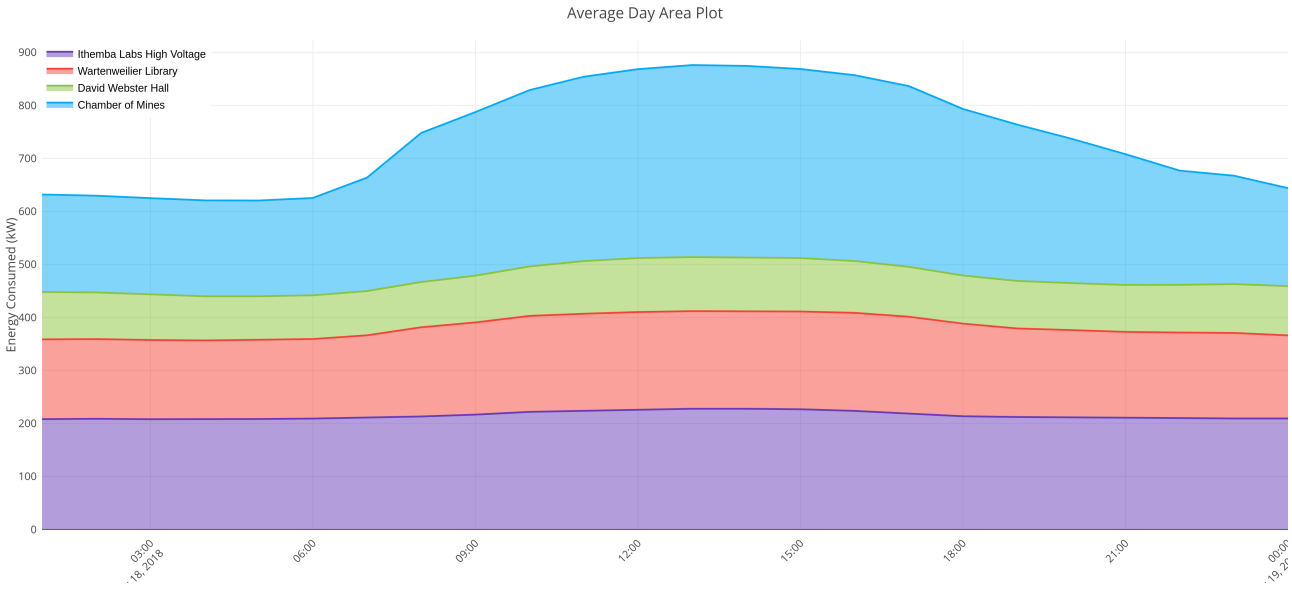

Fig. 9: Area Plot of the same 4 buildings consumption showing the cumulative draw of all selected. Can identify the maximum points and see the holistic draw.

D. Effective Use of Visualization: Sankey Diagram

A sankey diagram is a powerful visualization tool to show the flow within a system. Harvard very effectively used this on their website to show their campus energy flow. If you hover or click on a section a popup tells you about where the energy comes from and how sustainable it is. A screenshot of a potentual implmentation can be found in Figure 10.

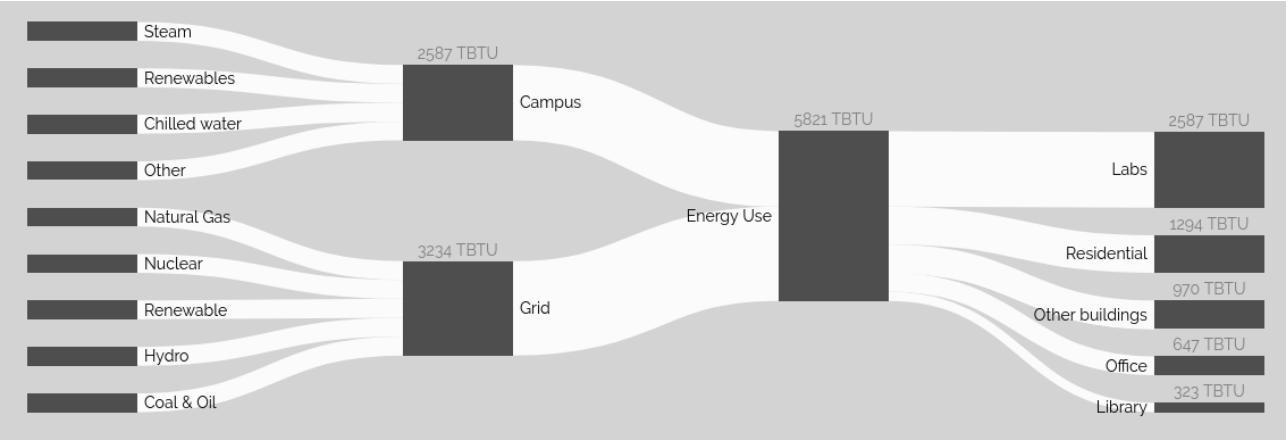

Fig. 10: Screenshot of a Sankey diagram used to show energy flow within Harvard university

E. Effective Use of Visualization: Sunburst Diagram

A sunburst diagram is a powerful way to show the relative sizes of entities within common groups. A sunburst chart shows the full hierarchy of the the predefined groups to provide deeper analysis capabilities. With a Sunburst chart, it's easy to see the largest contributing segments within a hierarchy of multiple levels. Figure 11 shows an example sunburst plot taken from a Microsoft discussion on sunburst charts that can be found here. This example shows the number of books in diffrent categories. For this project, the catecories could be building type, the next layer out could be building names and the final layer is the particular sensor. This diagram could be integrated with the comparative platform provided for the heatmap. An opensource javascript library recommended to implement this sunburst chart is sunburst-chart.

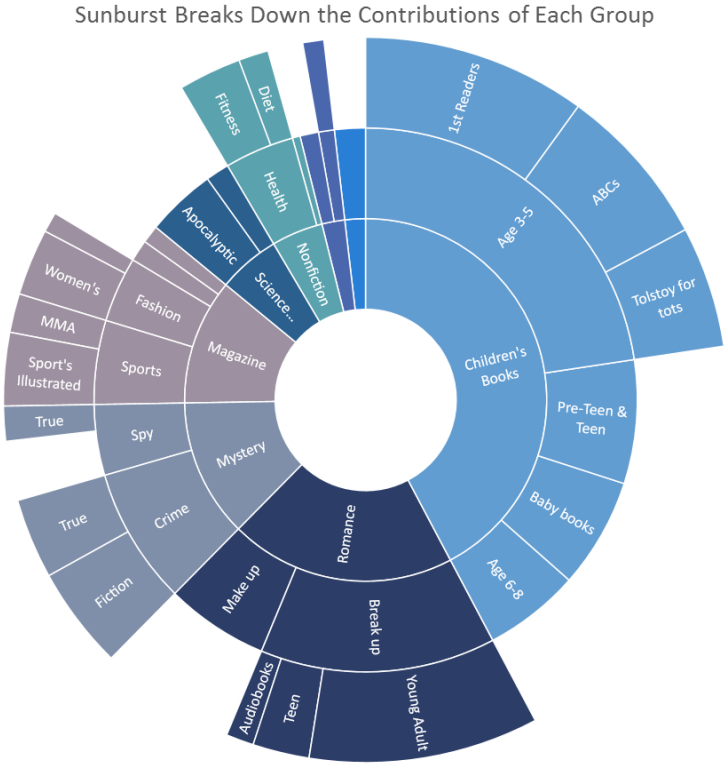

Fig. 11: Example of a sunburst chart to convey the basic idea

APPENDIX D

A. User Stories

The application user stores are shown table I below. These stories identify all core functionality desired within the application. With each section of user stories, a difficulty and priority calculation has been done and the story status has been included. As all development aimed to implement a vertical slice through the user stories, most epics have some degree of development done within them. The deadline indicates when the sprint epic should be done by to finish in 6 sprints.

Epic/Story	Difficulty	Importance	Status	Deadline
Epic 1: High level campus energy consumption comparison				
As a user, I want all buildings on campus on a heat map.	3	5	Complete	1
As a user, I want to see what building is using the most to least energy on the map.	4	5	Complete	1
As a user, I want to see all buildings on campus in a list ordered by consumption.	2	4	Complete	1
As a user, I want a Sankey diagram outlining energy flow on campus.	4	2	Incomplete	3
As a user, I want a Sunburst chart showing different consumers usage on campus.	3	2	Incomplete	3
Epic 2: Individual building information and comparison				
As a user, I want to find out information on a particular building by clicking on it.	3	5	Complete	1
As a user, I want to see a day/week/year graph of the building.	2	5	Complete	1
As a user, I want to see the average day/week/year graph of the building.	2	5	Complete	1
As a user, I want to see the average campus draw along with the building of interest.	4	4	Complete	1
Epic 3: Campus wide detailed building comparison				
As a user, I want to be able to compare different building consumption over time.	4	5	Complete	2
As a user, I want to be able to select building from the map to add to comparison.	4	5	Complete	2
As a user, I want to be able to select building from a list and add to comparison.	3	5	Complete	2
As a user, I want to compare as many buildings as I want to this comparison.	2	3	Complete	2
As a user, I want to be able to plot a line and area plot over day/week/year to compare.	3	3	Complete	2
As a user, I want to be able to plot the last or average charts of each time period.	3	3	Complete	2
Epic 4: Multiple campus support				
As a user, I want to be able to view maps for different campus.	4	3	Partial	4
As a user, I want to be able to change my map location to another campus.	2	3	Incomplete	4
As a user, I want to be able to add buildings from different campuses to my comparion.	3	3	Incomplete	4
As a user, I want to be able to compare whole campuses consumption and patterns.	3	3	Incomplete	4
Epic 5: Administration access				5
As an administrator, I want to be able to log in with my normal wits credentials.	5	3	Partial	
As an administrator, I want to be able to view a overview of all data sources used.	4	2	Incomplete	5
As an administrator, I want to see the geojson information for different buildings.	2	2	Incomplete	5
As an administrator, I want to be able to see the different visualizations generated.	2	2	Incomplete	5
Epic 6: Administration update functionality				
As an administrator, I want to be able to edit the underlying data sources.	5	3	Partial	6
As an administrator, I want to update the geojson when new buildings are added.	3	2	Partial	6
As an administrator, I want to be able to be able to update visulzations configurations.	4	2	Incomplete	6

TABLE I: Application epics and user stories

APPENDIX E

A. *Development Version Control and Planning Using Github*

Github is a powerful tool for individual and collaborative projects and is an integral part of the opensource software community. Github was used extensively as a version control management system, providing a framework for multiple developers to work together in the future. The application repository can be found here. Exensive documentation was also added to each section of the repository to show how to run each component. A number of issues were created to track the completion of the next stage of development stories. Screenshot 12 shows some of these outstanding issues. These issues were automatically added by a Github automation to the project board, as seen in screenshot 13. This project board is used during sprint planning. When an issue is closed on Github the card will automatically move to the ”Done” section. Issues associated with an open pull requiest are automatically put into the ”Needs Review” section.

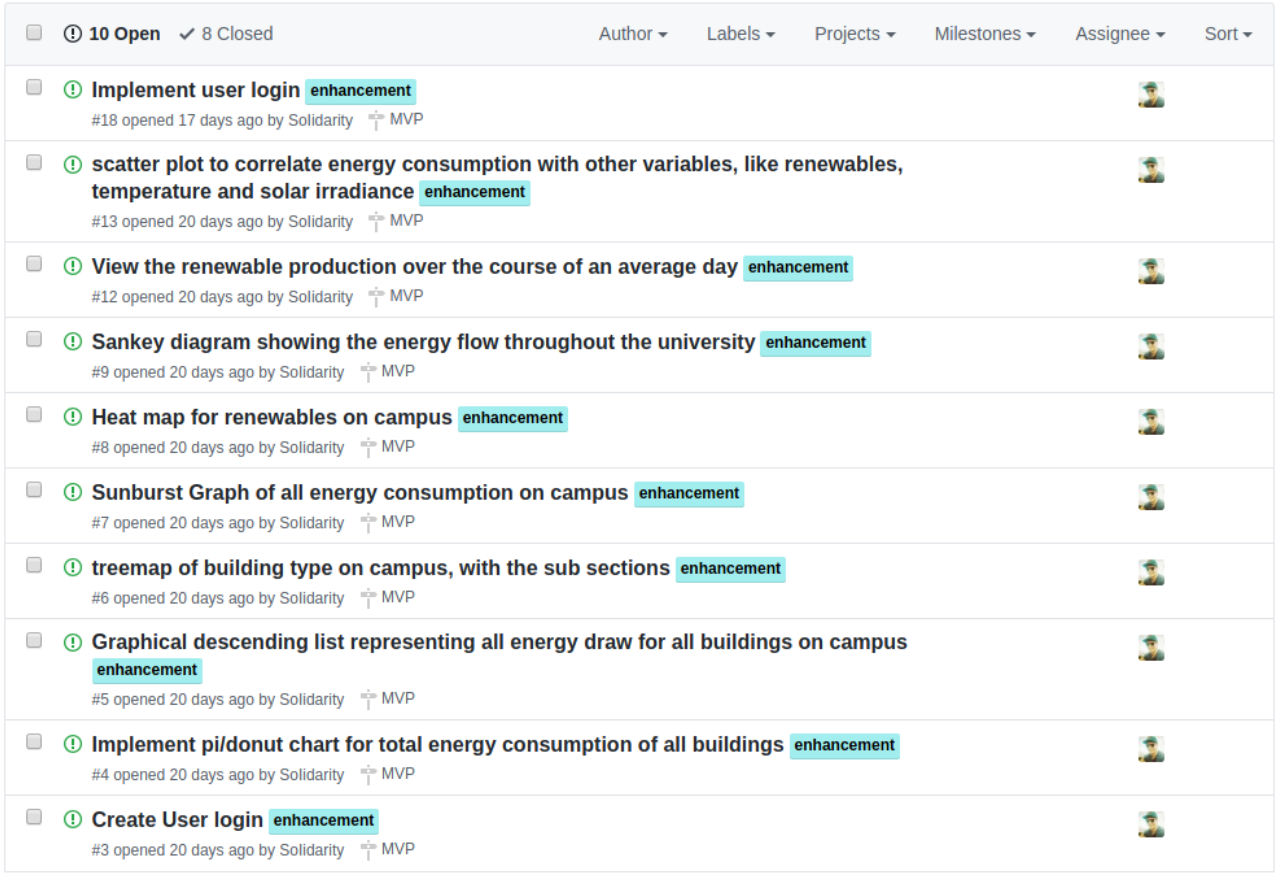

Fig. 12: Screenshot of the github showing a number of outstanding issues.

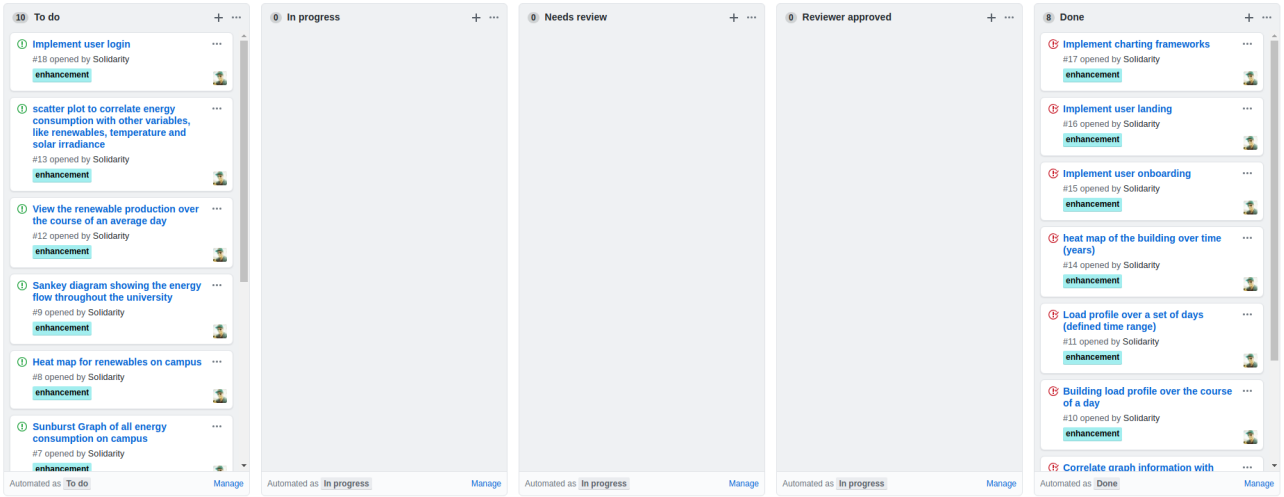

Fig. 13: Screenshot of the github showing the project board with automatic issue sections.

APPENDIX F

Additional backend design and implementation details are outlined within this appendix.

A. Backend Design: Justification for a Custom Built System

A number of turn key solutions currently exist on the market that are capable of visualizing energy consumption data such as Panasonics KW series, datapine or Qlik to name a few. However, the majority are not free/open source and none of these platforms offer the ability to integrate with the existing Wits ecWIN architecture. As a result, a suite of custom middleware would be required to integrate the systems together to provide a communication layer. This would incur very high development costs in order to integrate the two systems. Added to this, an ongoing subscription fee for premium solutions would make the total system operation cost exceed its worth.

More broadly, there is a mirriod of systems used to store, process and visualize time series data that could be used for this project. Two notable ones are Prometheus coupled with Grafana and the Telegraph, InfluxDB, Chronograph and Kapacitor called the TICK stack. Prometheus offers a time series data monitoring solution that is capable of performing complex computations and queries, provides a visualization platform and utilizes sharing and federation to archive data storage scaling. Prometheus is often coupled with Grafana, an open source platform used to analyze and monitor data sets, to provide a full suite of analytics and graphing options. Both projects are open source and free and offer a good stack to achieve the desired goals of the system. That being said, the potential interactivity available within the Grafana platform is heavily limited meaning that any kind of relative comparisons requiring user interaction and/or selection becomes impossible.

The TICK stack does seem promising at face value, offering an open source time series platform that can accumulate, analyze and act on time series data. The Chronograf administrative user interface provides the ability to make extensive visualizations and dashboards. The TICK stack falls short when it comes to dashboard customization. While the platform does come with a number of beautiful pre-made templates and does offer a simple ability to customize them, the level of interactivity again becomes an issue.

In order to achieve the desired characteristics of scalability, usability, interactivity, modularity, interoperability and accessibility a custom solution is the only possible option wherein specific design decisions are needed to be made to achieve specific properties. The rest of this paper will outline this proposed solution as well as evaluate its successfulness as an engineering solution.

B. Backend Design: Selection of an Appropriate Backend Time Series Database

A time series database is a key component of this projects design. The selection of the most appropriate time series database will be based off the design ethos. This ethos can be seen as a checklist of properties that the design must fulfill. A comparison is now drawn between the three major contenders OpenTSDB, InfluxDB and TimescaleDB. All offer scalability, high querability and are all open source with a large developer base.

Firstly, OpenTSDB offers high availability and scalability with the ability to store and serve a massive amount of time series data. OpenTSDB runs on the Hadoop and Hbase architecture making it highly scalable. That being said, OpenTSDB has been shown to require a lot of complex configuration and requires an order of magnitude more administration and overhead to run in production over the other configurations [14].

The choice then needs to be made between InfluxDB and TimescaleDB. Both are easy to configure and have large developer bases. With that said, InfluxDB has been shown to outperform TimescaleDB when executing a query over one single metric tag[14]. This is the configuration required for this projects application and so InfluxDB seems like a clearer choice. Added to this is that InfluxDB is older with a more mature developer base with almost three times the number of stars on github and 14x more commits indicating that it might be a better choice to ensure long term sustained support and developer backing. For these reasons, InfluxDB has been chosen as the backend database storage solution to store time series data. All five discussed databases were experimented with and the experiences with each were sumarized in the table II below.

Property	traditional SQL	noSQL (MongoDB)	OpenTSDB	InfluxDB	TimescaleDB
High scalability for time series data	2	2	5	5	5
Easy queryability for time series data	2	3	5	5	4
Easy deployment and configuration	5	5	3	5	4
Administration ease	5	5	3	5	4
Community support (future proof)	5	5	4	5	3
Development ease	5	5	3	4	3
Available documentation	5	5	5	5	4
Ease of integration with Python	5	5	3	5	4

TABLE II: Rating of different database types to store time series data

C. Backend Design: Microserver Containerization Using Docker

Docker is considered one of the fastest growing technologies in software history, providing a framework to pack, distribute and manage scalable applications[1]. Docker helps automate the development and deployment process of software. In the context of this project, it provides a number of clear benefits. The first and largest is the reduction in infrastructure and maintenance costs. Once the development of the production environment is fully specified the Docker container can be deployed within any Linux environment without any additional configuration. Migrating to new cloud server architecture becomes far easier as no custom configurations are required to port the server to a new architecture. From a maintenance perspective, by isolating each discrete component of a software system they can fail without breaking the rest of the system. Moreover, each component of the technology stack can be scaled both vertically and horizontally independent of one another. Management of each part of the technology stack can be individually configured and then automated resulting in reproducible and predictable deployment.

Isolation and separation of application components enhances system security by preventing access from one container to the next. Docker containers can't access the contents of other containers and if networking between containers is configured correctly, complete isolation of databases and other key architecture is achievable. This notion of isolation was implemented for the MongoDB, InfluxDB and Python script containers wherein each is completely isolated from the outside world and can only be accessed through the restricted and controlled API.

D. Backend Design: Entity Relational Diagram and Database Structure

The backend system consists of two distinct databases: a time series optimized InfluxDB used to historic consumption data. and MongoDB is used to store computed results. Live data is added to InfluxDB by a web scraper. In the future, data can be added directly to InfluxDB. Diagram 14 shows how the data structures defined within the MongoDB and InfluxDB interconnect connect in an entity relational diagram.

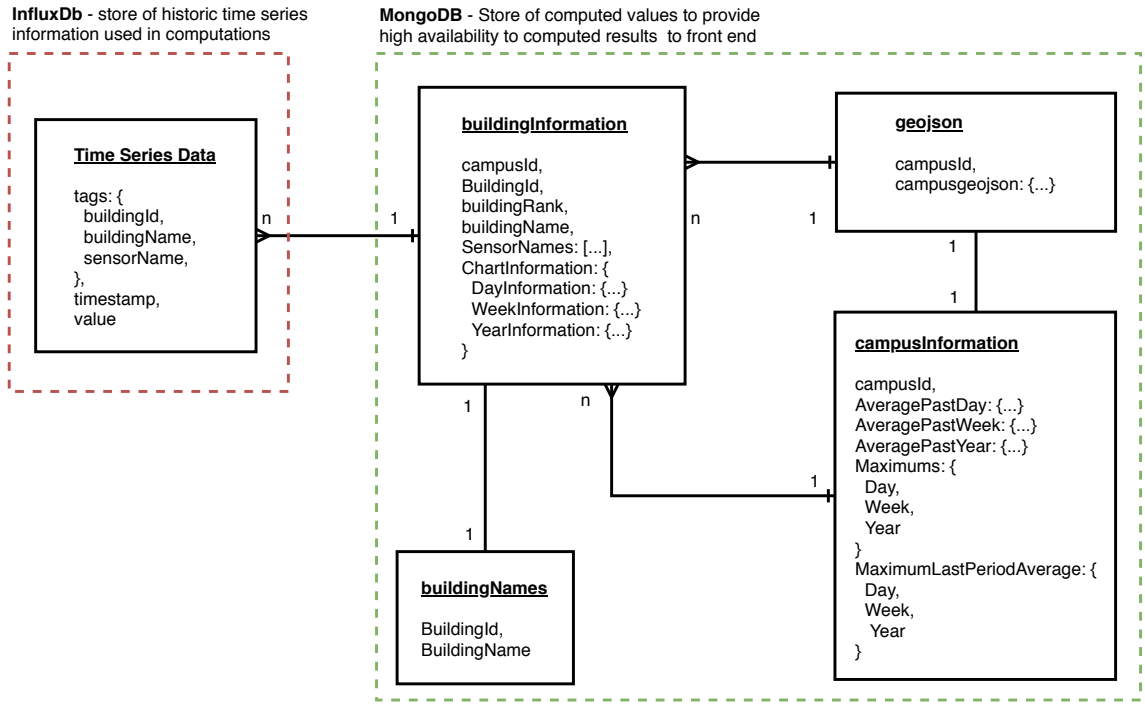

Fig. 14: Diagram depicting a simplification of entity relational diagram used to relate data object together within the noSQL architecture

The `buildingInformation` document store is used to store information about each and every building. Each building can relate to a number of sensors stored within the time series InfluxDB. Each building information

document also stores the information required to generate each of the charts for the frontend. These are represented as json blobs within the document wherein a unix timestamp is related to a value over the period of the chart. For example, within the `ChartInformation` section, the `WeekInformation` blob is used to store an array of the last weeks draw, an average weeks draw, the last average week and the maximum week. This is all the information required to generate plots for a particular building over all possible time periods. If additional charts are required, extra information can be added to this document store.

The `geojson` document store contains information for each campus wherein a standard geojson object is defined to store building information for all buildings on a particular campus. This information is used in the generation of interactive heat maps. This information includes the GPS locations of each building as well as the heat map values for that particular building[5].

The `campusInformation` document store is used to store campus wide computations used in relative graphs and metrics to compare one building against the campus averages. Other information, like campus wide maximums used in heatmap generation, are also defined.

E. Backend Design: Interactive Documented Application Interface With Swagger

Swagger was used to create an interface to view all API end points. This interface also conveys the data type that each end point is expecting as well as the format of data returned by the endpoint. This provides future projects the ability to integrate easily into this system and re-use the calculated data. The production version of this system will server the endpoints over HTTPS.

Figure 15 shows the endpoints defined for the API. Each section can be expanded to view the underlying API information, such as expected input and output formats. The data formats expected for each endpoint is defined in 16.

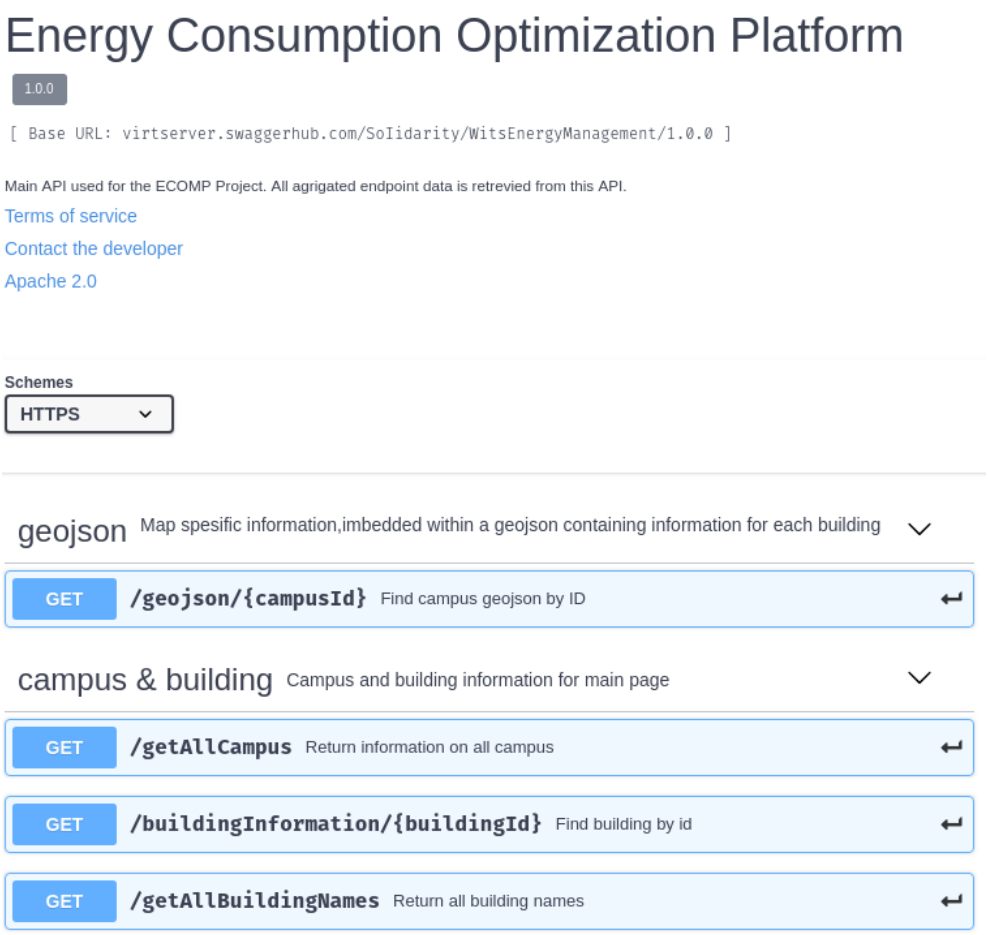

Fig. 15: Screenshot of the swagger UI. This shows the API end points generated for the MVP. More end points would be added as and when additional visualizations and functionality are needed.

Fig. 16: Screenshot of the swagger UI. This shows all data models defined. These are used to generate the frontend visualizations

F. Backend Design: SSL and User Authentication

User authentication is a key component of the administration of the platform. Only authorized user should be able to log in and alter the underlying data structure. To enable this functionality while still enabling users to use their traditional wits user names and passwords oAuth can be used to enable delegated authentication. This means that Wits will authenticate the users against their backend authentication system (such as active directory) and then provide an authentication token this projects backend server. This token is then used to authenticate the user. To facilitate this token process, SAML is used to integrate Wits' authentication system with 0Auth, an authentication as a service platform. Auth0 then generates a json web token that is passed to the client to authenticate API requires. All privileged API requests will then use this token to authenticate. Flask will be able to verify the legitimacy of this token to validate the user.

All this added complexity is there to leverage Wit's existing architecture such that people will not require new credentials. If a user changes their credentials within the Wits system they will change on the new platform as well meaning that users have a unified, universal login. This prevents the need for custom credentials and makes the on boarding of new users far easier. Figure 17 below outlines the authentication process for the client in the form of a sequence diagram.

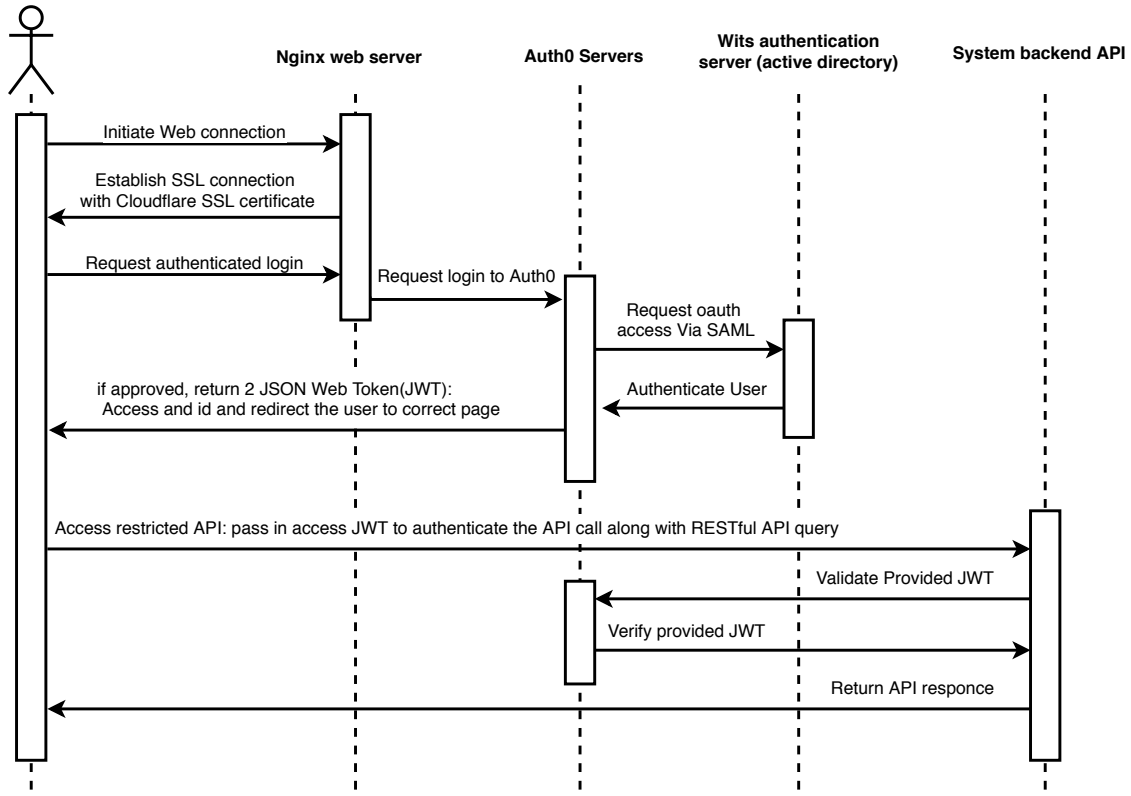

Fig. 17: Sequence diagram depicting how a user receives an access token and then uses it to query the API

APPENDIX G

Additional frontend design and implementation details are outlined within this appendix.

A. *Frontend Design: Technology stack configuration*

A number of diffrent javascript libraries were used in the design of the the frontend application. Primarily, Vuejs was used as a frontend framework, with Material Vue as a styling framework. A basic theme was used from Creative Tim. Plotly was used to generate graphs, wraped within the vue-plotly package to gain access to the plotly as a vue component. Leaflet and openstreetmap was used to generate interactive maps. Leaflet and openstreetmaps were chosen over Google maps as they have unlimited free usage. Npm was used as the package manager, using a webpack configuration generated by vue-cli to build the application. Eslint was used to lint the source code to ensure it conformed to the Airbnb styling guide. Figure 18 shows all the diffrent technology stacks used and how they interconnect.

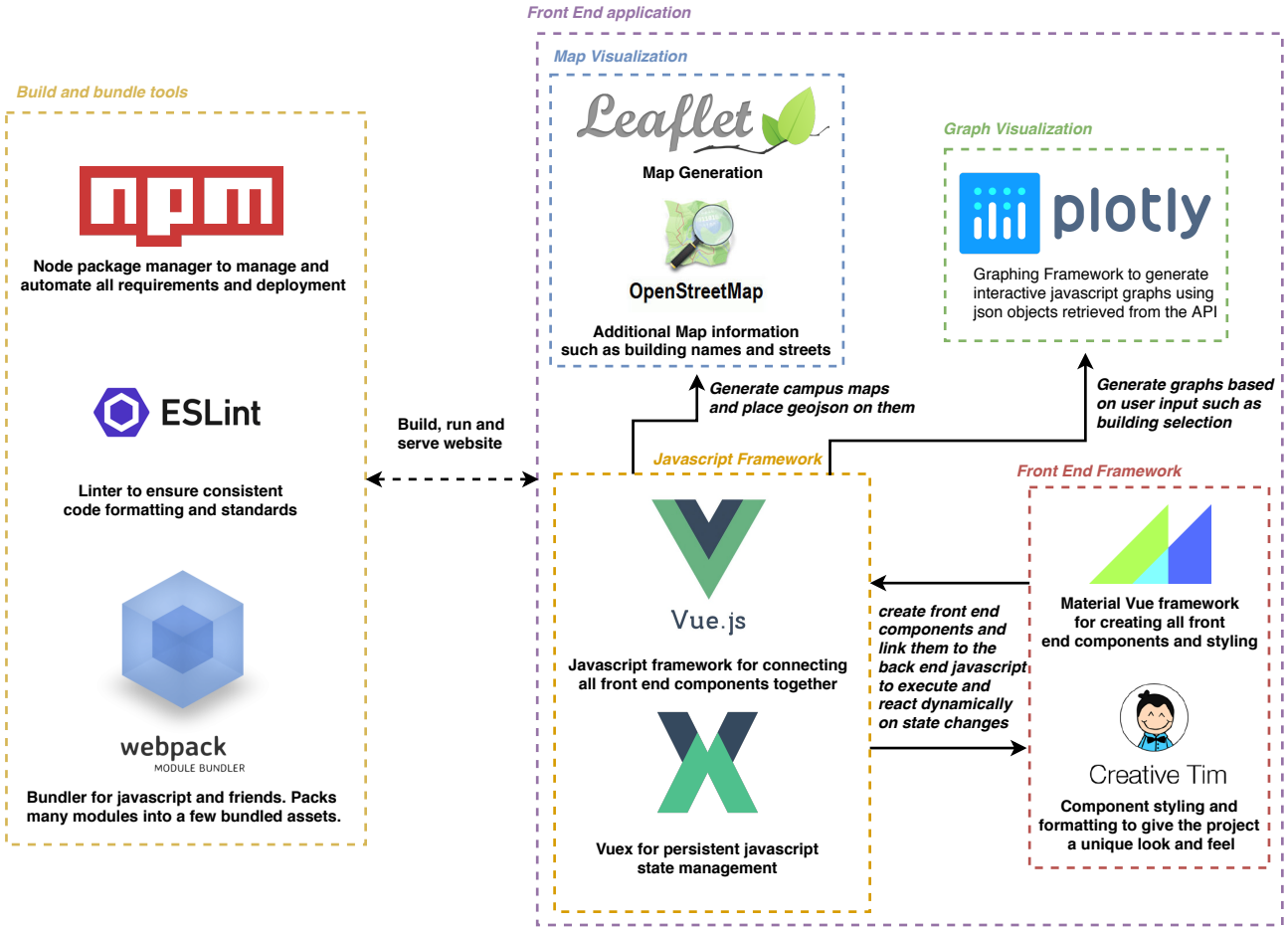

Fig. 18: Diagram depicting the implemented frontend application technologies usages

Vuejs was chosen as the javascript framework of choice. It was chosen over Angular and React for a number of reasons. Primarily, its ease of use, small learning curve and active developer community made it ideal for this project. Additionally Vue has a ligher browser payload than the other two frameworks. Table III outlines the differences. This information was adapted from reference [15].

	Angular	React	Vue
Type	A Framework	Library to build UI	A Framework
Why Choose	If you want to use TypeScript	If you want to go for “everything-is-JavaScript” approach	Easy JavaScript and HTML
Founders	Powered by Google	Maintained by Facebook	Created by Former Google Employee
Initial Release	September 2016	March 2013	February 2014
Application Types	If you want to develop Native apps, hybrid apps, and web apps	If you want to develop SPA and mobile apps	Advanced SPA and started supporting Native apps
Ideal for	If you want to focus on large-scale, feature-rich applications	Suitable for modern web development and native-rendered apps for iOS and Android	Ideal for web development and single-page applications
Learning Curve	A steep learning curve	A little bit easier than Angular	A small learning curve
Developer-friendly	If you want to use the structure-based framework	If you want to have flexibility in the development environment	If you want to have separation of concerns
Model	Based on Model-View-Controller	Based on Virtual Document Object Model	Based on Virtual Document Object Model
Written in	TypeScript	JavaScript	JavaScript
Community Support	A large community of developers and supporters	Facebook developers community	open-source project sponsored through crowd-sourcing
Language Preference	Recommends the use of TypeScript	Recommends the use of JSX – JavaScript XML	HTML templates and JavaScript
Popularity	Widely popular among developers	More than 27,000 stars added over the year	More than 40,000 stars added on GitHub during the year
Companies Using	Used by Google, Forbes, Wix, and weather.com	Used by Facebook, Uber, Netflix, Twitter, Reddit, Paypal, Walmart, and others	Used by Alibaba, Baidu, GitLab, and other

TABLE III: Table comparing diffrent properties of Angular, React and Vue

B. Frontend Design: Javascript State Management

Persistent state management was utilized to enforce predictable access control to the Javascript state. Figure 26 shows the communication call stack used by state management from an API call.

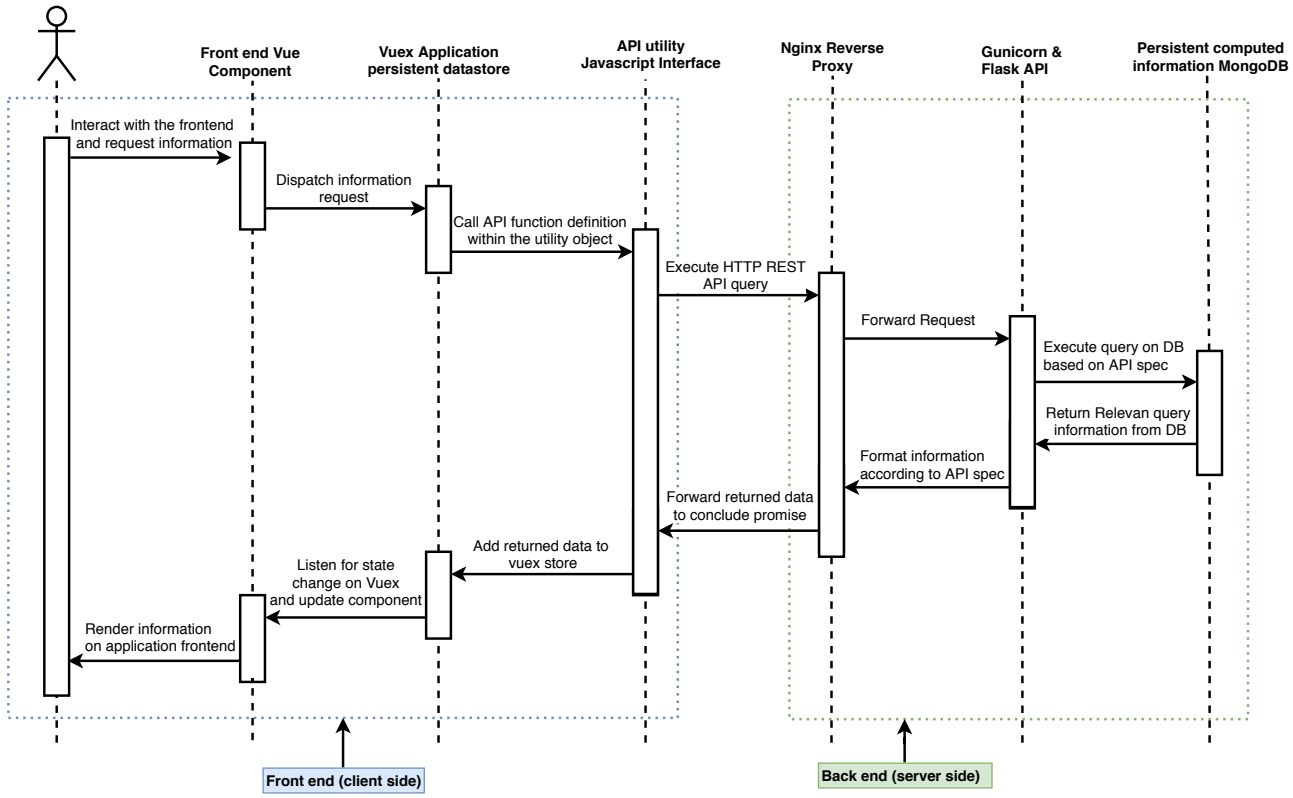

Fig. 19: Diagram depicting the frontend API request call stack including state management modification

In effect, this operates by the user interacting with a frontend component that then dispatches a request to the state management store (called Vuex in Vuejs). Vuex then performs an API call by calling the API interface object defined by a utility script and then adding the returned result to the Vuex store. The Utility scrip makes use of Axios to execute the query. The frontend component watches the state of the store through a computed function that reacts asynchronously to state changes. When the component changes, the computed function updates. These computed functions are then used to update the state of the user interface. This means that there are effectively three layers in the frontend between a user interacting with a component and the interface updating with each layer having its own domain of responsibility.

Firstly, the frontend components do not know anything about how the state is managed nor how the API interface operates. They just need to know how to display data when the state changes. Secondly, the Vuex store is only

concerned with listening for dispatched requests, calling the utility API object and then storing the results. It does not know how information is used or how it is retrieved. Lastly, the API object interface is only concerned with how to call the API interfaces and what those definitions look like. It has no notion of what the underlying API logic looks like nor how the information is used by the store. This separation and modularity makes iterative development and upgradability simple and natural.

Traditionally, Vuex stores are not persistent. This means that after a page reload or if you have the website open in two tabs, there is no retention or synchronization of the state. As a result, there is high overhead for returning users who then need to re-query the API to update the Vuex state. In order to minimize the number of API requests required on page reload and to keep user configurations between reloads (such as what charts they have plotted against other charts) Vuex-persistedstate was used to persist and rehydrate the Vuex state between page reloads. This means that the Vuex state is stored within local storage within the users browser between reloads.

C. Frontend Design: Component Based Frontend Modelling

A component based design has been used at all levels of the frontend architecture within the vuejs application. Every element rendered on the frontend is comprised of a number of sub elements, making updating one section of the frontend simpler. The diagram 20 below shows the diffrent components within the frontend.

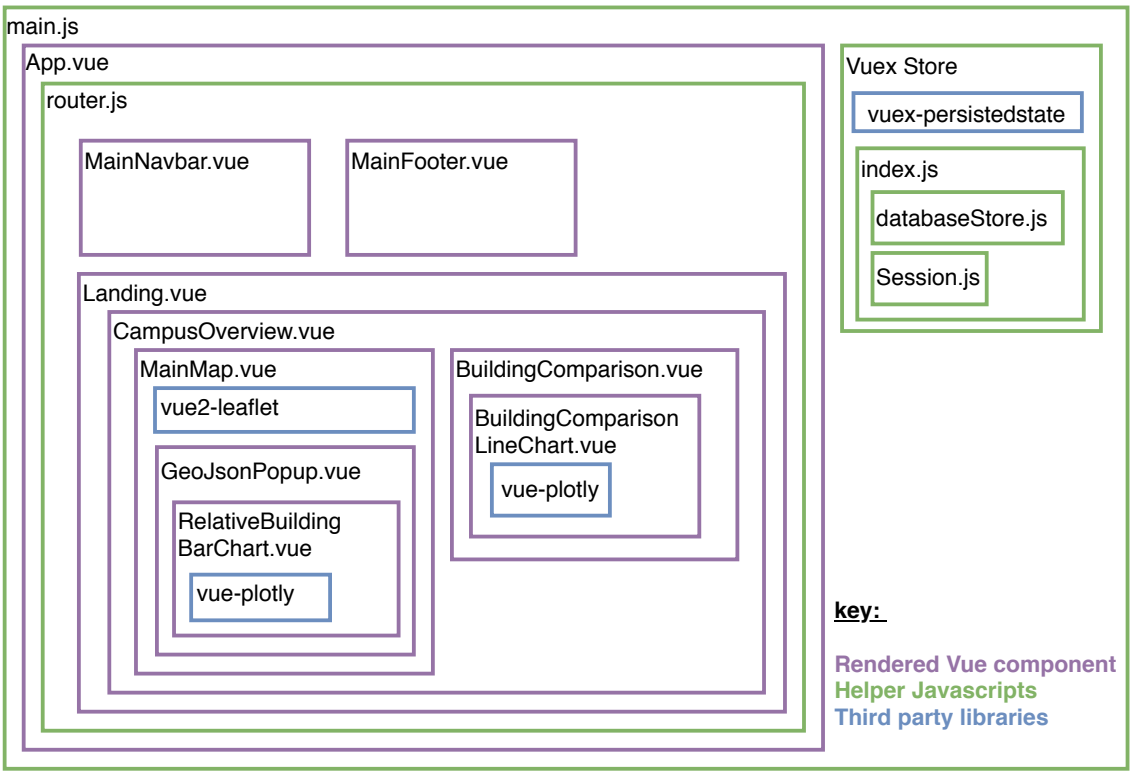

Fig. 20: Diffrent layers within the Vue component design

APPENDIX H

A. *Devops Design: Virtual Private Server and Content Delivery Network*

The Vultr was used to host all Docker containers that run all system architecture. This server specs a single core Intel Skylake CPU, 2GB or ram and 40GB of solid state hardware. All development work should be done on the server using SSH or Portainer discussed in Appendix F-C. Image 21 shows a screenshot from the main Vultr dashboard. This dashboard would be used to update the server configuration if need be in future.

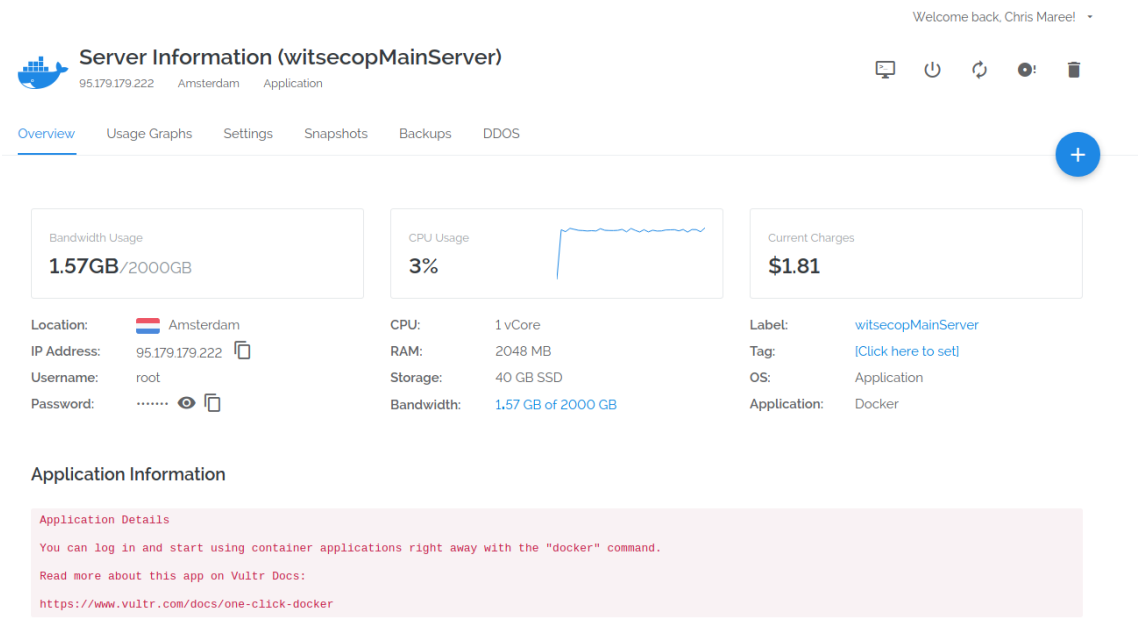

Fig. 21: Screenshot of the Vultr server dashboard

Portainer was used to manage the Docker containers used to run the technology stack. Portainer allows you to create new containers using a GUI as well as to mange running containers and compose stacks. This means that you can update the containers without needing to use a terminal. This project has not implemented any notion of a continuous integration(CI) pipeline as it was not felt to add enough value to the development process given the size of the project. Portainer reduces the complexity in updating making deployments almost to the ease of CI without the complexity and rigidity of a CI pipeline. Figure 22 and 23 show screenshots of the Portainer interface used to control Docker container deployments.

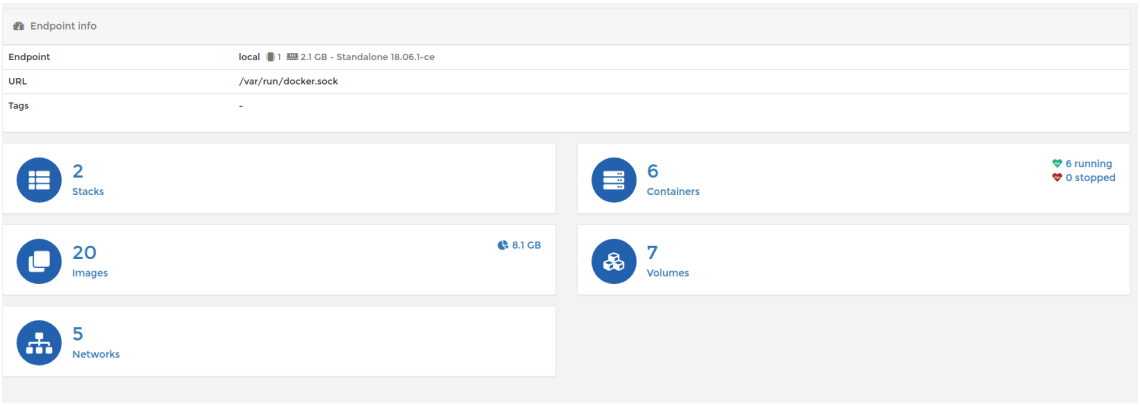

Fig. 22: Screenshot of Portainer showing the stacks, containers and images used to run the web server and API

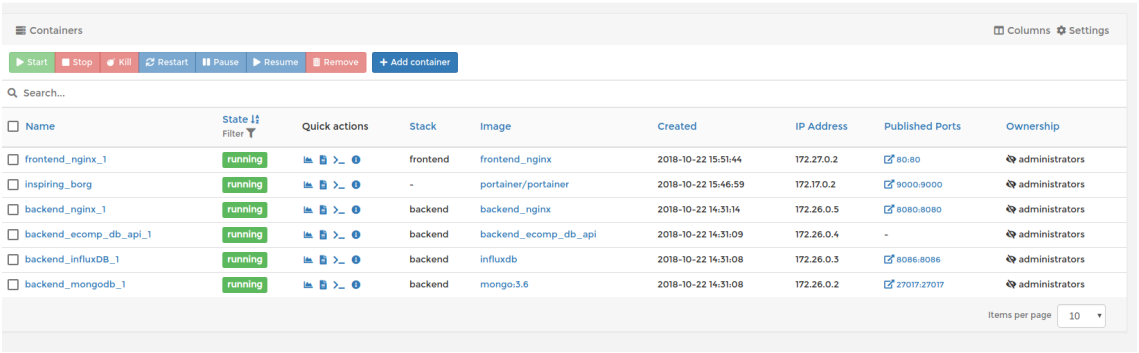

Fig. 23: Screenshot of Portainer container management allowing you to start/stop containers and manage deployments

B. Devops Design: Networking Topography

A network configuration was set up to handle the traffic of a high number of concurrent users. The configuration was not benchmarks to identify concurrency limitations but implementation did follow the standard requirements to achieve a highly scalable system.

On a client request, DNS records point to Cloudflares DNS management system wherein the Cloudflare Content Delivery Network is positioned to server pre-cached information, such as static page files. Cloudflare also minifys the pages to reduce file overheads. The Cloudflare CDN then requests information from the Vultr instance, wherein the Docker containers hosting the files servers reside. A diagram depicting this network communication can be seen in Figure 24 below.

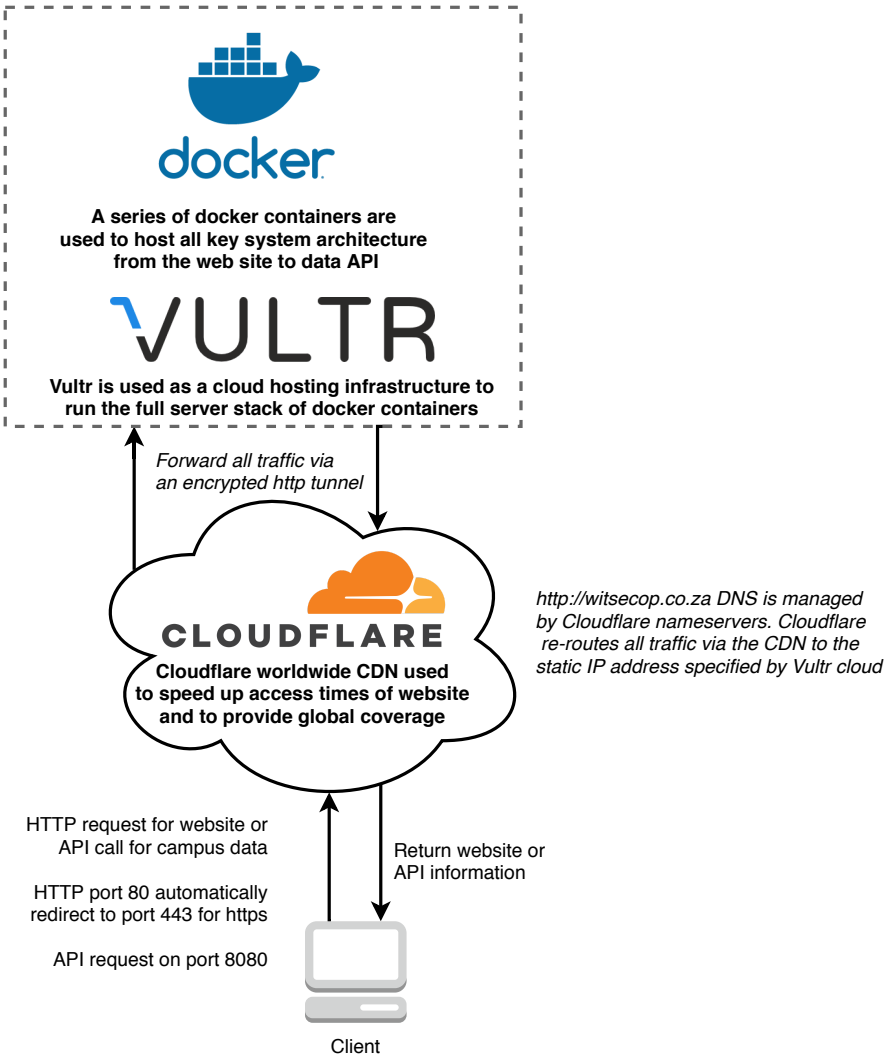

Fig. 24: Network topology depicting how the frontend communicates with the backend system via the Cloudflare CDN

APPENDIX I

An MVP was created to showcase the core functionality of the envisioned system. Vertical slices through the the application epics have been completed such that a fully working produce was created, as can be seen witsecop.co.za. Emphasis was placed on achieving full functionality for a small number of core features rather than more incomplete features.

A. MVP Application Screenshots: Application Landing Page

A graphical landing page was created to make the web interface visually appealing and to draw the user in.

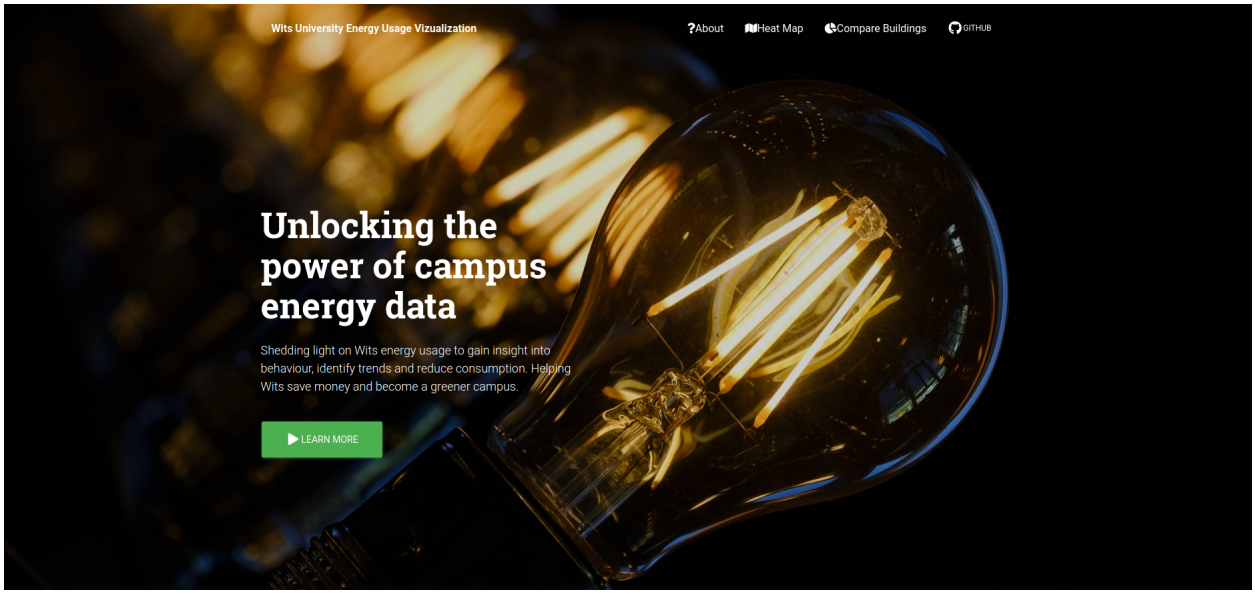

Fig. 25: Screenshot of the website landing page

B. MVP Application Screenshots: Application Infograph Section

Information surrounding why the application was created, what problems it aims to solve and how it aims to do this are included next. this information aims to show the user in a short amount of text *why* the system is important.

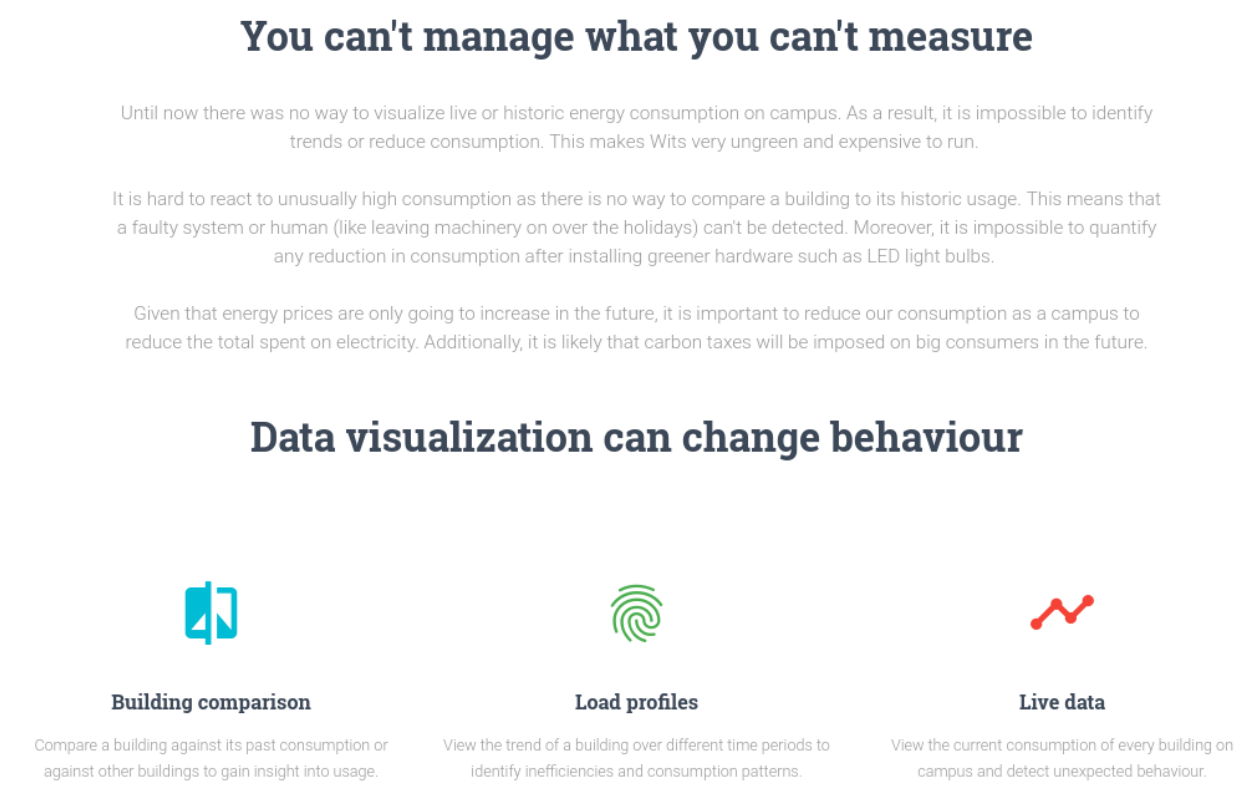

Fig. 26: Screenshot of the website information section, used to explain why the system is important

C. MVP Application Screenshots: Campus Heatmap

An interactive heatmap was created that shows detailed information on all campus buildings. Figure 27 below shows what heading and start of the campus map and Figure 28 shows the full Wits main campus along with relative building heats.

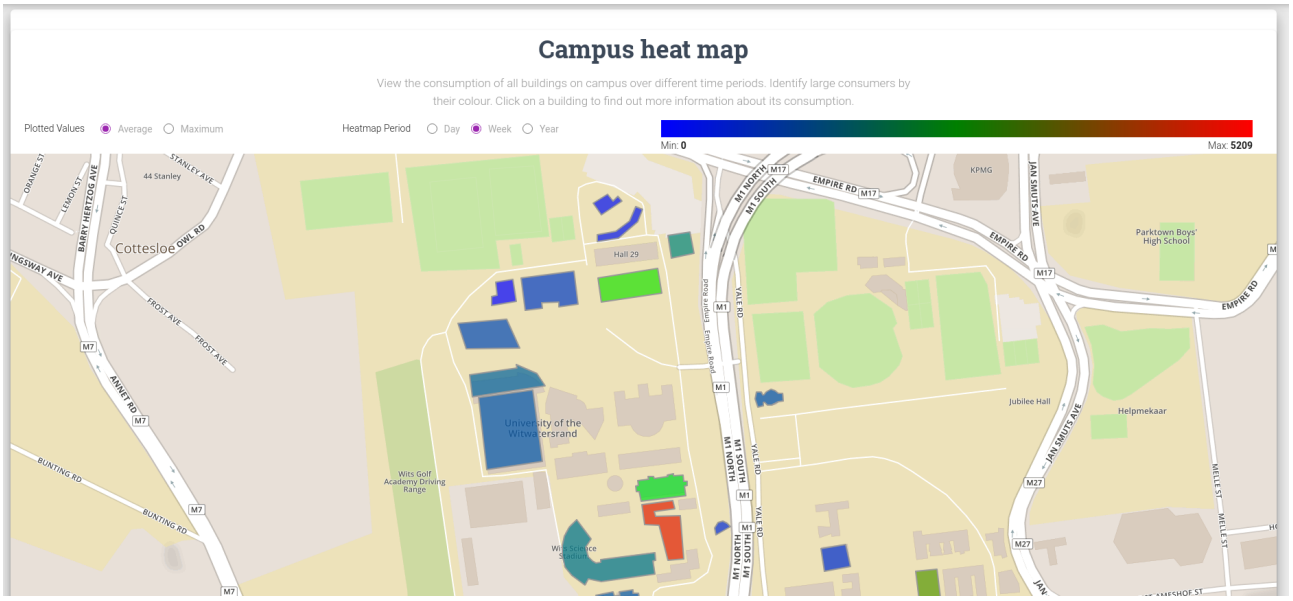

Fig. 27: Screenshot of the website heatmap section, including the plotted values and heatmap period. The ploted values change between showing the average or maximum ranges over a day, week or year.

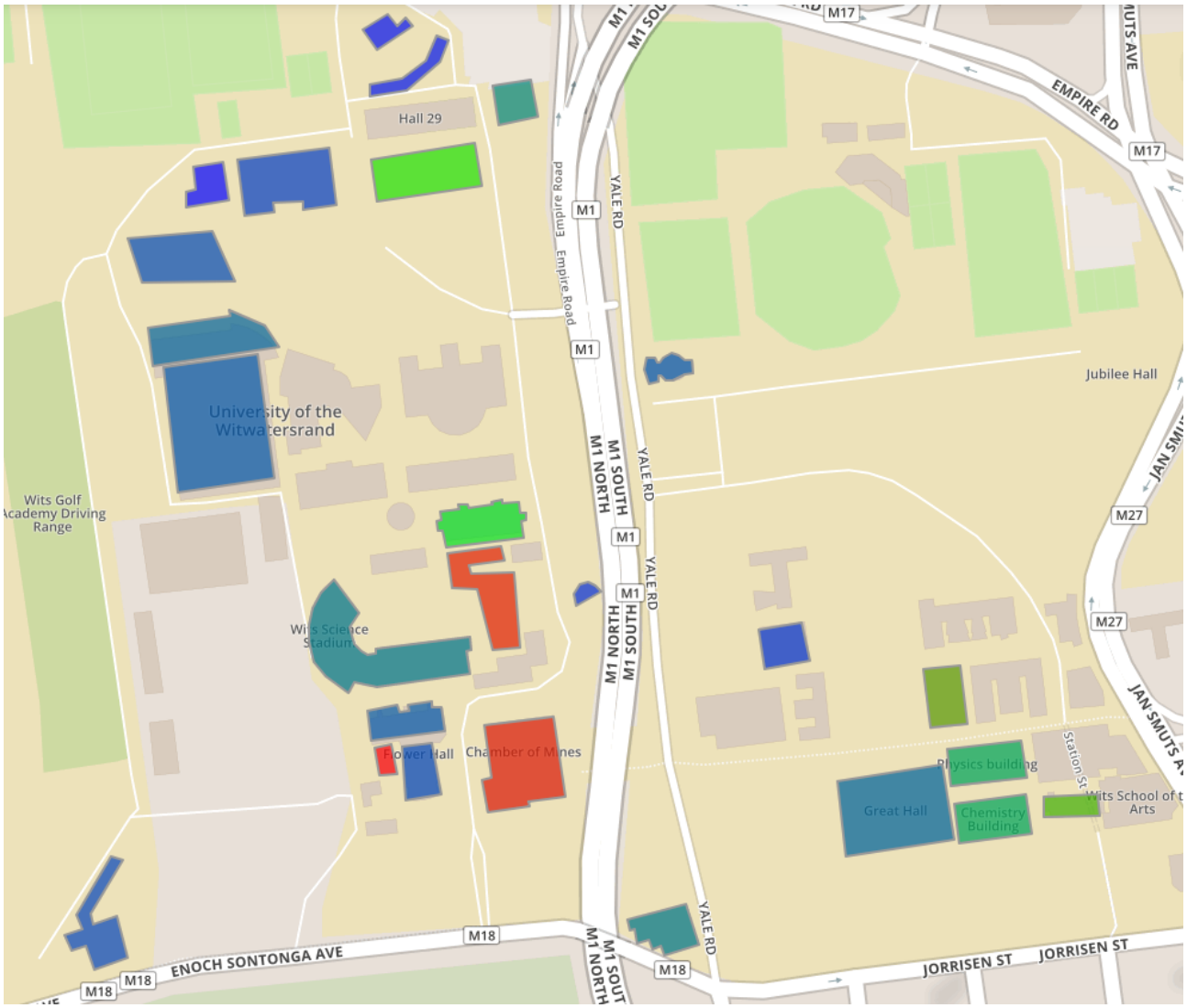

Fig. 28: Screenshot of the website campus heatmap, showing all buildings and their relative temperatures

D. MVP Application Screenshots: Building Information Popup

If the user clicks on a building within the heatmap, an informative popup opens to show them information about that particular building. Information is only retrieved from the API when the user clicks on the building to minimize the server overhead when the page loads. Figure 29 shows the Origins Centre popup in one week mode and 30 shows chamber of mains in day mode. This figure can show the power of relative visualizations wherein you can clearly see a difference in consumption for the last week as apposed to an average week.

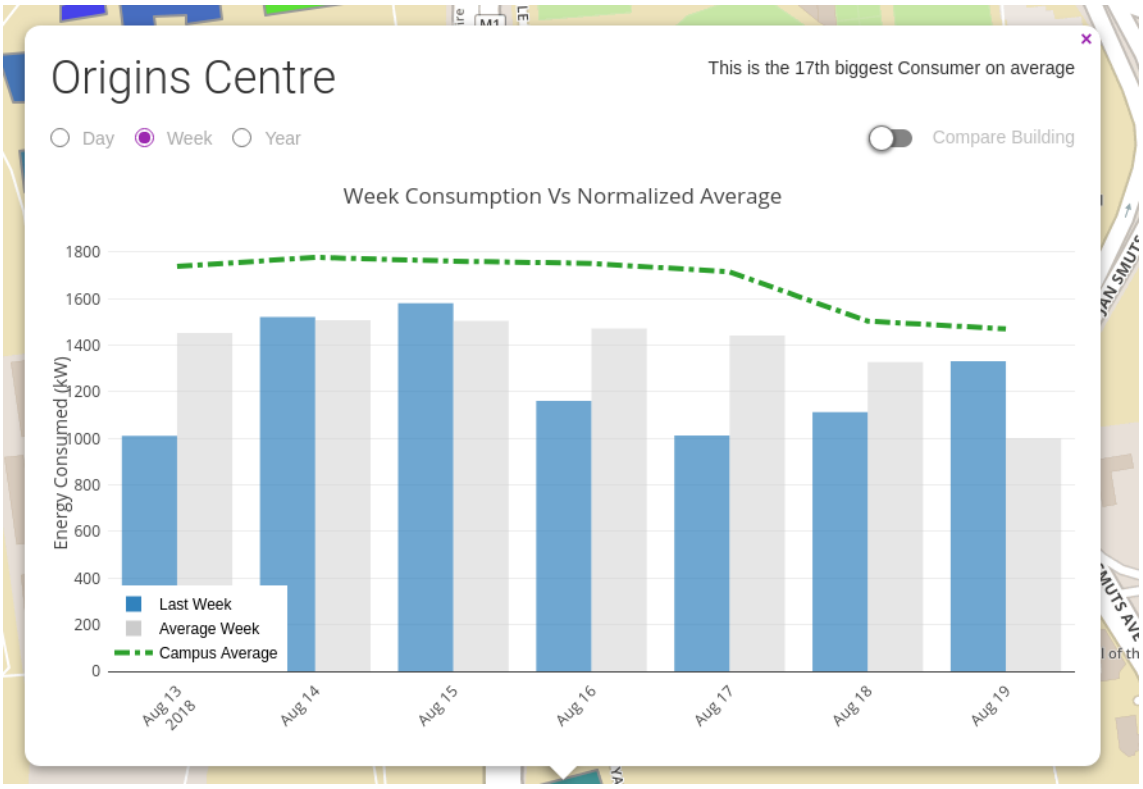

Fig. 29: Screenshot of the website building Information over the course of one week. Can see that this building is less than the average draw and can compare this past week to an average week

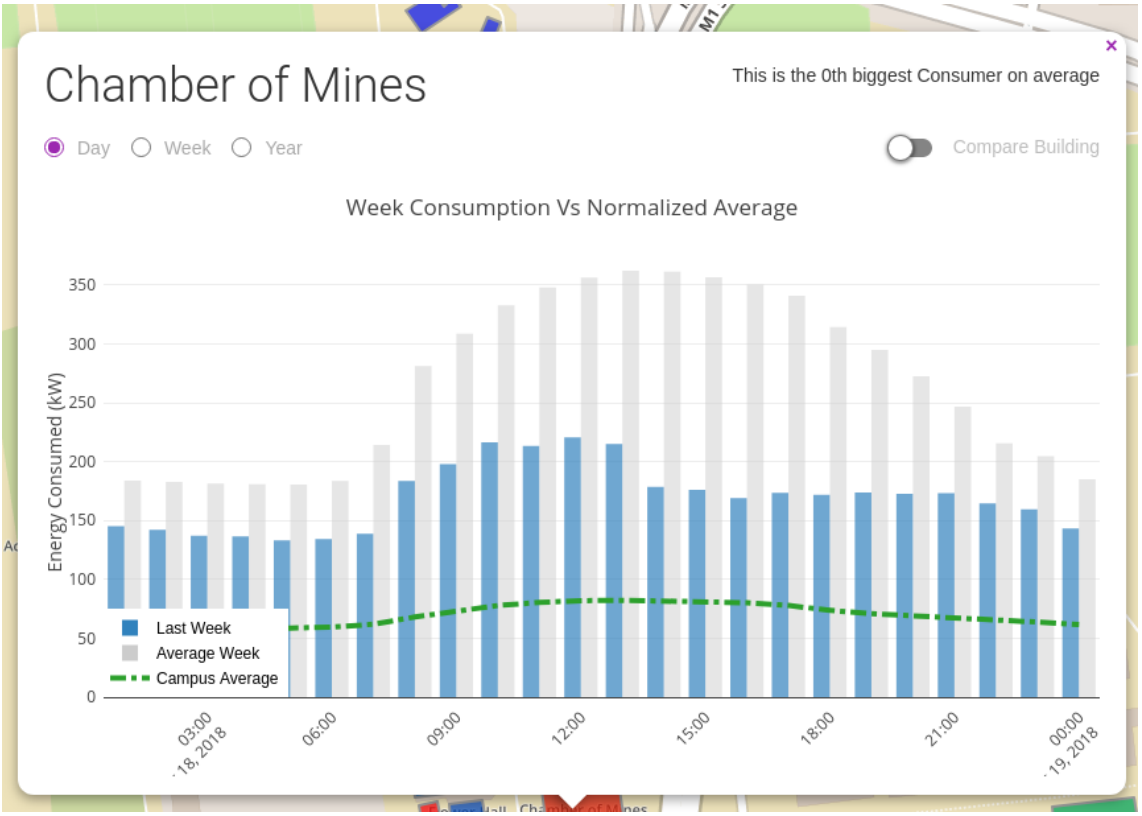

Fig. 30: Screenshot of the website building Information over the course of one day. Can view that this building is displaying a diffrent pattern to the normal behaviour for a day due to a difference in the plot shape

E. MVP Application Screenshots: Relative Building Comparison

The next section show relative building comparisons, capable of comparing a number of diff rent buildings against each other. Buildings can be chosen from a drop down list of by clicking *compare building* within the building popup. Figure 31 shows the empty state of the building comparison section. Figure 32 shows a number of buildings compared using a stacked area plot to get the cumulative consumption profiles. This plot is in average mode to show the standard trend of these loads over time. Figure 33 takes the same data set but plotted using a line graph. Different patterns emerge in this mode, such as when one load uses more than another. Figure 34 shows the same data set, but over the last period of time rather than an average. Lastly, 35 shows the value in being able to switch between a area and line plots to identify when one building consumes more than another.

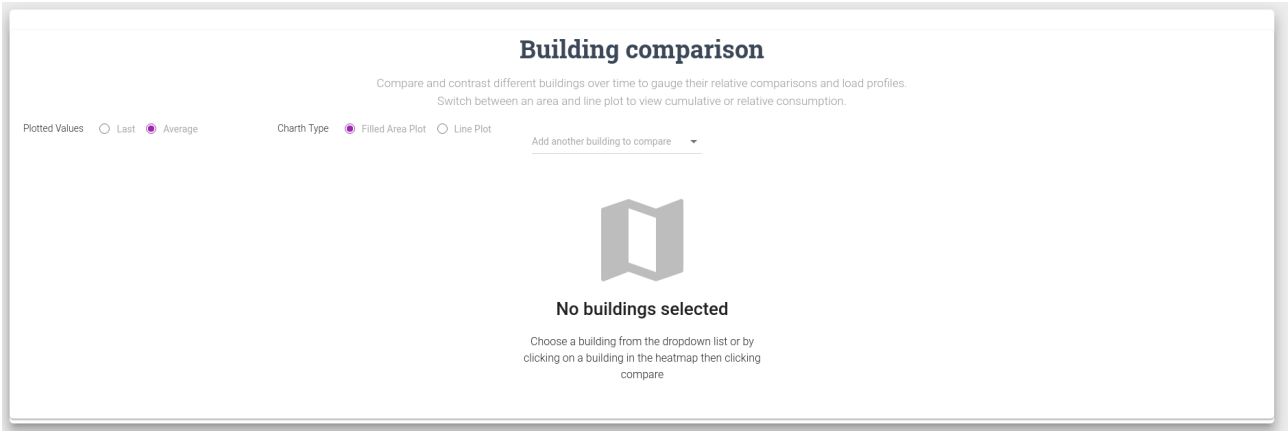

Fig. 31: Screenshot of the website building comparison section in its empty state

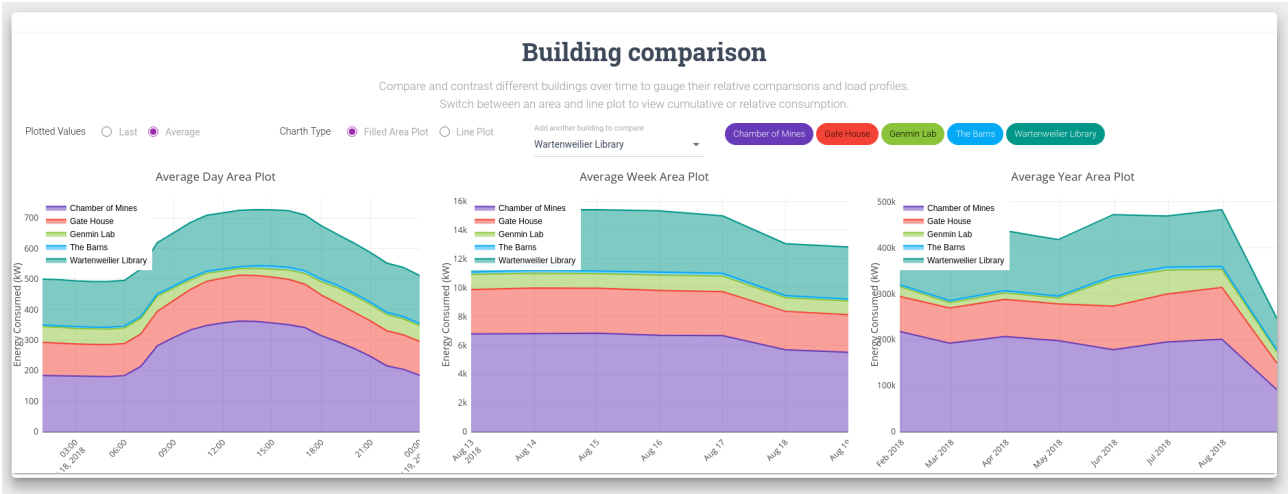

Fig. 32: Screenshot of the website building comparison section with 5 buildings selected, in area plot mode. The cumulative trends can be identified

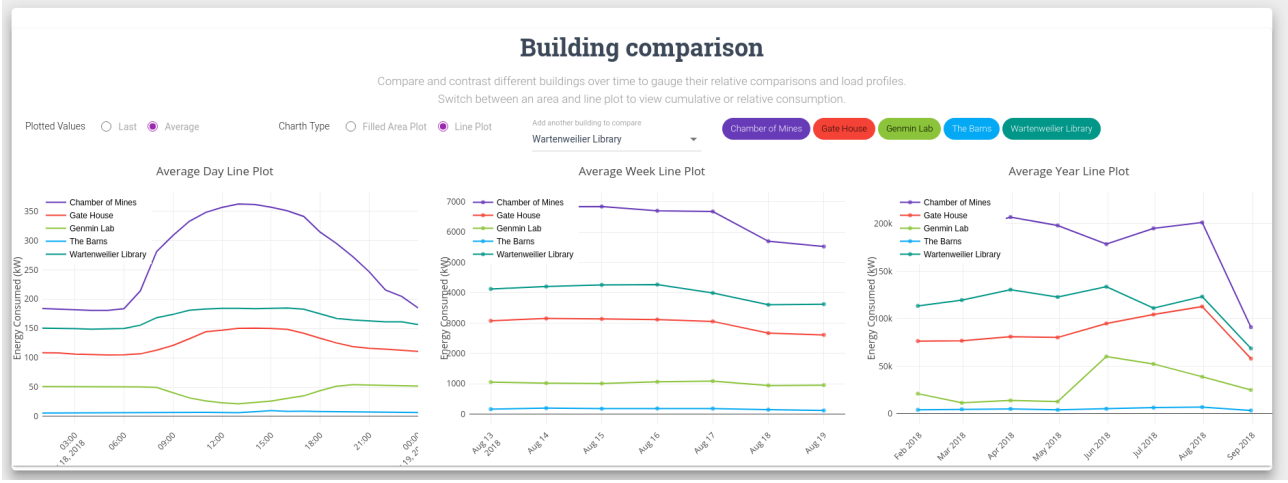

Fig. 33: Screenshot of the website building comparison section with 5 buildings selected, in line plot mode. The individual consumption trends can be identified

Fig. 34: Screenshot of the website building comparison section with 5 buildings selected, in line plot mode over the last period of time. The individual consumption trends can be identified and load fluctuations can be clearly seen. This information was hidden within the average plot

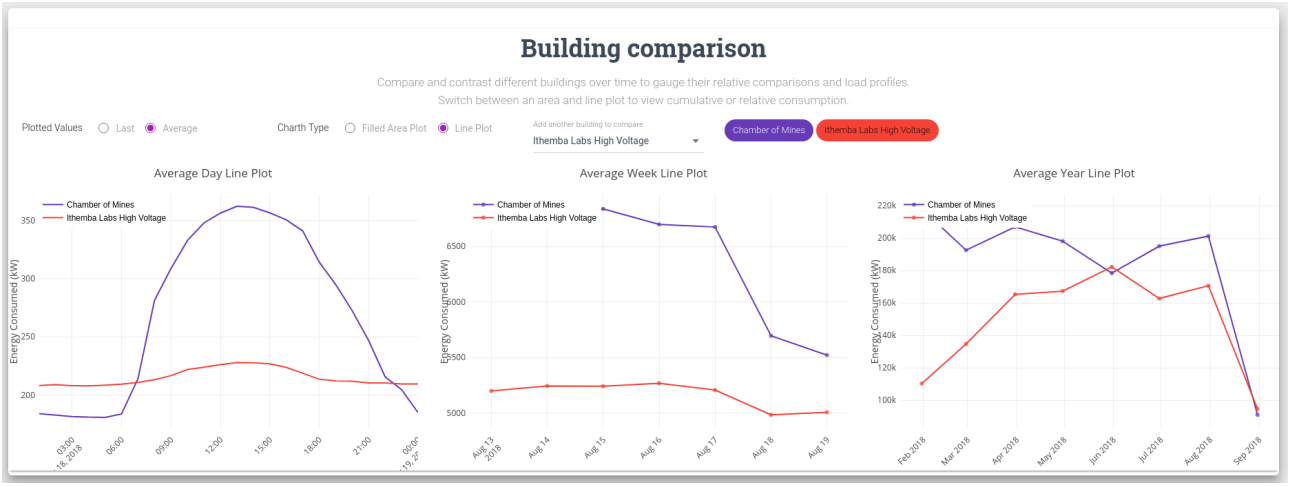

Fig. 35: Screenshot of the website building comparison section with 2 buildings selected, in line plot mode showing averages of two buildings with very different consumption patterns. Can identify the point in time where one building is consuming more energy than the other

REFERENCES

[1] IOL Staff Reporter. "Electricity will become unaffordable... unless we go green.", 2018. URL https://www.iol.co.za/ios/behindthenews/electricity-will-become-unaffordable-unless-we-go-green-16939263.

[2] E. Anton and L. Amory. "South Africa's Electricity Choice (Part 2): Renewable Energy is a win-win — Daily Maverick.", 2018. URL https://www.dailymaverick.co.za/article/2018-02-02-south-africas-electricity-choice-part-2-renewable-energy-is-a-win-win/.

[3] Minister Of Finance. "DRAFT CARBON TAX BILL." Tech. rep., 2017. URL http://www.treasury.gov.za/publiccomments/CarbonTaxBll2017/DraftCarbonTaxBillDecember2017.pdf.

[4] N. Maistry and T. M. Mckay. "Promoting energy efficiency in a South African university." Tech. Rep. 3, 2016. URL http://www.scielo.org.za/pdf/jesa/v27n3/01.pdf.

[5] S. Pahl, J. Goodhew, C. Boomsma, and S. R. J. Sheppard. "The Role of Energy Visualization in Addressing Energy Use: Insights from the eViz Project." *Frontiers in psychology*, vol. 7, p. 92, 2016. URL http://www.ncbi.nlm.nih.gov/pubmed/26903900.

[6] Harvardmetalab. "Student Visualization of Harvard Energy & Emissions Data.", 2016. URL https://green.harvard.edu/student-vizualization/.

[7] Harvard. "Harvard University Energy Consumption.", 2013. URL http://bin-yan.github.io/visualization-energy/.

[8] H. J. "Final year project working on campus energy measurements and visualization." Tech. rep., 2018.

[9] C. CHURILO. "InfluxDB Markedly Outperforms OpenTSDB in Time Series Data & Metrics Benchmark.", 2018. URL https://www.influxdata.com/blog/influxdb-markedly-outperforms-opentsdb-in-time-series-data-metrics-benchmark/.

[10] Mike Freedman. "TimescaleDB vs. InfluxDB: purpose built differently for time-series data.", 2018. URL https://blog.timescale.com/timescaledb-vs-influxdb-for-time-series-data-timescale-influx-sql-nosql-36489299877.

[11] Information Regulator South Africa. "Protection of Personal Information Act, 2013 Ensuring protection of your personal information and effective access to information." Tech. rep., 2013. URL http://www.justice.gov.za/inforeg/docs/InfoRegSA-POPIA-act2013-004.pdf.

[12] Payscale. "Software Developer Salary (South Africa).", 2018. URL https://www.payscale.com/research/ZA/Job=Software{_}Developer/Salary.

[13] Nishi Sooful. "2017 ANNUAL REPORT OF THE UNIVERSITY OF THE WITWATERSRAND, JOHANNESBURG INCORPORATING REPORTS OF SENATE AND COUNCIL." Tech. rep., 2017. URL https://www.wits.ac.za/media/wits-university/giving-to-wits/documents/WitsAnnualReport2017.pdf.

[14] C. CHURILO. "MongoDB vs InfluxDB for time series data workloads.", 2018. URL https://www.influxdata.com/blog/influxdb-is-27x-faster-vs-mongodb-for-time-series-workloads/.

[15] Spec India. "React vs Angular vs Vue.js: A Complete Comparison Guide.", 2018. URL https://www.spec-india.com/blog/react-vs-angular-vs-vue-js-a-complete-comparison-guide/.