

Design Document for: Scalable Web Application Framework for Monitoring Energy Usage on Campus

Daniel Schroeder, Aubrey Thenell, Parker Bruni



Abstract

The purpose of this document is to outline the architecture and component design for a Scalable Web Application Framework for Monitoring Energy Usage on Campus. The designs for each modular component are discussed in detail, outlining the data residing in each object and the functionality it holds in the application. Additionally, this document aims to discuss the rationale for each design decision and the interactions between individual components.

CONTENTS

1	Introduction to Project	5
1.1	Who requested it?	6
1.2	Why was it requested?	6
1.3	What is its importance?	6
1.4	Who was/were your client(s)?	6
1.5	Who are the members of your team?	6
1.6	What were their roles?	6
1.7	What was the role of the client(s)?	6
2	Requirements Document	6
2.1	Introduction	6
2.1.1	Purpose	6
2.1.2	Scope	6
2.1.3	Definitions, acronyms, and abbreviations	6
2.1.4	References	7
2.1.5	Document Overview	7
2.2	Overall description	7
2.2.1	Product perspective	7
2.2.2	Product functions	7
2.2.3	User characteristics	8
2.2.4	Constraints	8
2.2.5	Assumptions and Dependencies	8
2.2.6	Apportioning of Requirements	8
2.3	Specific Requirements	8
2.3.1	External Interfaces	8
2.3.2	Functions	9
2.3.3	Performance Requirements	9
2.3.3.1	Users	9
2.3.3.2	Data Management	9

		2
2.3.3.3	Visualization	10
2.3.3.4	Objects	10
2.3.4	Logical Database Requirements	10
2.3.5	Design Constraints	10
2.3.5.1	Standards Compliance	10
2.3.6	Software System Attributes	10
2.3.6.1	Reliability	10
2.3.6.2	Availability	10
2.3.6.3	Security	10
2.3.6.4	Maintainability	11
2.3.6.5	Portability	11
3	Design Document (original)	11
3.1	Introduction	11
3.1.1	Purpose	11
3.1.2	Scope	11
3.1.3	Overview	11
3.1.4	Reference Material	12
3.1.5	Definitions and Acronyms	12
3.2	System Overview	12
3.3	Design Viewpoints and Concerns	13
3.3.1	Stakeholders and Concerns	13
3.3.2	Design Viewpoints	13
3.3.2.1	Context Viewpoint	13
3.3.2.2	Composition Viewpoint	13
3.3.2.3	Dependency Viewpoint	13
3.3.2.4	Information Viewpoint	13
3.3.2.5	Logical Viewpoint	14
3.3.2.6	Interface Viewpoint	14
3.3.2.7	Structure Viewpoint	14

		3
	3.3.2.8 Interaction Viewpoint	14
3.4	System Architecture	15
	3.4.1 Architectural Design	15
	3.4.2 Decomposition Description	15
	3.4.2.1 Log In	15
	3.4.2.2 Navigation	16
	3.4.2.3 Create Block	16
	3.4.2.4 View Blocks	16
	3.4.2.5 Create Dashboard	16
	3.4.2.6 Create Story	17
	3.4.2.7 Add Building/Meter	17
	3.4.2.8 Create Graph	17
	3.4.2.9 Get AcquiSuite Data	17
	3.4.3 Design Rationale	18
3.5	Component Design	18
	3.5.1 Block	18
	3.5.1.1 Element	18
	3.5.1.2 Functionality	18
	3.5.1.3 Rationale	18
	3.5.2 Dashboard	18
	3.5.2.1 Element	19
	3.5.2.2 Functionality	19
	3.5.2.3 Rationale	19
	3.5.3 Login	19
	3.5.3.1 Element	19
	3.5.3.2 Functionality	19
	3.5.3.3 Rationale	19
	3.5.4 Navigation Bar	19
	3.5.4.1 Element	20
	3.5.4.2 Functionality	20

		4
3.5.4.3	Rationale	20
3.5.5	Graph Component	20
3.5.5.1	Element	20
3.5.5.2	Functionality	20
3.5.5.3	Rationale	20
3.5.6	Story	21
3.5.6.1	Element	21
3.5.6.2	Functionality	21
3.5.7	Content View	21
3.5.7.1	Element	21
3.5.7.2	Functionality	21
3.5.7.3	Rationale	21
3.6	Human Interface Design	21
3.6.1	Overview of User Interface	21
3.6.1.1	Public User	21
3.6.1.2	Authorized User	22
3.6.1.3	Administrative User	22
3.6.2	Screen Images	22
3.6.2.1	Home Webpage (Public Access)	22
3.6.2.2	Home Webpage (Logged in Access)	22
3.6.2.3	Buildings Webpage	22
3.6.2.4	Selected Building Page	23
3.6.2.5	Dashboards Page	23
3.6.2.6	Selected Dashboard Webpage	23
3.6.2.7	Story Webage	23
4	Tech Review (original)	23
5	Weekly Blog Posts (all team members' posts)	23
6	Final Poster	24

		5
7	Project Documentation	24
7.1	How does your project work? (Could include the following...)	24
7.2	How does one install your software, if any?	24
7.3	How does one run it?	24
7.4	Are there any special hardware, OS, or runtime requirements to run your software?	24
7.5	Any user guides, API documentation, etc.	24
8	Recommended Technical Resources for Learning More	24
9	Conclusions and Reflections (each team member answers all questions individually)	24
10	Appendix 1: Essential Code Listings.	24
11	Appendix 2: Anything else you want to include.	24
11.1	Design Document Images	25
11.1.1	Home Webpage (Public Access)	25
11.1.2	Home Webpage (Logged in Access)	25
11.1.3	Buildings Webpage	26
11.1.4	Selected Building Page	26
11.1.5	Dashboards Page	27
11.1.6	Selected Dashboard Webpage	27
11.1.7	Stories Webpage	28
11.2	Technology Review Images	28

1 INTRODUCTION TO PROJECT

(Hint, refer to project proposal and problem statement)

- 1.1 Who requested it?
- 1.2 Why was it requested?
- 1.3 What is its importance?
- 1.4 Who was/were your client(s)?
- 1.5 Who are the members of your team?
- 1.6 What were their roles?
- 1.7 What was the role of the client(s)?

(I.e., did they supervise only, or did they participate in doing development)

2 REQUIREMENTS DOCUMENT

2.1 Introduction

2.1.1 Purpose

The purpose of this document is to outline the project and all associated information. Furthermore, it will outline how the end product will be used, how it was developed, and all resources and documentation referenced and used during development. This document will also describe the applications target audience, user interface, and any hardware/software requirements. Finally, it will define how our client, team, and users will see the final product and related functionality.

2.1.2 Scope

This project will provide a web application that displays different collections of energy use reports for campus buildings. Each of these collections will be customizable, with the ability to add, remove, and filter building data according to user preference. The web application will allow users to visually comprehend energy data in a way that can raise awareness about consumption and reduce the campus-wide carbon footprint. The web application will also have three different permission levels: Guest, Registered Users, and Admin Users. Each one will provide permissions to what can be viewed and edited.

2.1.3 Definitions, acronyms, and abbreviations

Term	Definition
OSU	Oregon State University
MEAN stack	MongoDB, Express, AngularJS, Node.js
JSON	JavaScript Object Notation
AcquiSuite	Data Acquisition Servers responsible for metering energy usage
Collections	User Interface dashboard that displays assortment of blocks
Blocks	Charts/graphs of building data
Stories	Collections of dashboards

2.1.4 References

The references are:

- [1] "AcquiSuite-A8812", AcquiSuite — English, 2017. [Online]. Available: <http://www.obvius.com/Products/A8812>. [Accessed: 02Nov2017].
- [2] "Data-Driven Documents", D3 — English, 2017. [Online]. Available: <https://d3js.org>. [Accessed: 02Nov2017].
- [3] "Vis.js", Vis.js — English, 2017. [Online]. Available: <http://visjs.org/>. [Accessed: 02- Nov- 2017].
- [4] "MongoDB, Express, AngularJS, Node.js", MEAN.IO — English, 2017. [Online]. Available: <http://mean.io>. [Accessed: 02- Nov- 2017].
- [5] "Introducing JSON", JSON. — English, 2017. [Online]. Available: <http://json.org> [Accessed: 02- Nov- 2017].
- [6] "BuildingOS", BuildingOS — English, 2017. [Online]. Available: <https://buildingos.com/>. [Accessed: 02- Nov- 2017].
- [7] "Modbus How Does Modbus work Modbus Tutorial", Learning Engineering — English, 2016. [Online]. Available: <https://www.youtube.com/watch?v=jfKY928Ozrw/>. [Accessed: 02- Nov- 2017].
- [8] "AcquiSuite EMB", AcquiSuite — English, 2017. [Online]. Available: <http://www.obvius.com/Products/A8810-0/>. [Accessed: 02- Nov- 2017].
- [9] "Modbus Addressing & Wiring Best Practices", obviusenergy — English, 2017. [Online]. Available: <https://www.youtube.com/watch?v=7x0tQfIOsDY>. [Accessed: 02- Nov- 2017].
- [10] "Mocha", Mocha — English, 2017. [Online]. Available: <https://mochajs.org/>. [Accessed: 02- Nov- 2017].

2.1.5 Document Overview

The remaining sections of this document provide a more elaborate description of product functionality, assumptions, and specific requirements. Section 2 provides information about functional requirements, data requirements, and assumptions made for designing the *Scalable Web Application Framework for Monitoring Energy Usage on Campus*. Section 3 outlines the specific requirements for the final product, the external interfaces that communicate with the software, and the functional requirements of the system.

2.2 Overall description

This section will provide an overview of the web application as a whole, including:

- 1) Details about the user interface and expected user interactions.
- 2) An outline of specific constraints and dependences included in development.
- 3) Background information about the specific software requirements.

2.2.1 Product perspective

This product will replace the current web application that is used by The Oregon State Sustainability Office. The product will display energy usage information about Oregon State University buildings through an intuitive user interface. The application will gather data from energy meters by connecting to AcquiSuiteTM data acquisition servers. The energy data from the meters will be transferred to a database that the product will access directly. Users will interact with the application interface from an internet browser application.

2.2.2 Product functions

The product will allow users to create accounts that may be given either administrative or user permissions. Users have the ability to personalize their accounts. Administrators accounts will have special capabilities that user accounts will not.

The application will allow users to create customizable dashboards that will contain easily adjustable blocks of campus building data that will contain the campus building energy usage data. These blocks of data will be the basic building blocks for the dashboard and will provide an intuitive view of the data. They will feature various graph types, building energy efficiency rankings, and data trends.

Each OSU building that contains the energy monitoring meter(s) will have a specific, non-customizable page that will display general information. The product will also have a public interface and a private administrative interface. Administrators of the application will have the ability to add, remove, or edit entire buildings profiles, building subspaces, or individual meters.

2.2.3 User characteristics

A user that will be using the general public UI will not need to know any specific information about the application to navigate the various energy data presentations. A public user will be able to intuitively navigate the UI at their discretion.

An administrative level user will need a basic understanding of the tools of the application because they will be allowed the freedom to control parts of the website as well as have access to more specific energy data within the application. An administrator will likely not need extensive training to use this application for more specific purposes as the administrator UI will be designed to be intuitive to navigate.

2.2.4 Constraints

Data updates will be limited to a granularity of 15 minute intervals. The data acquisition server is capable of providing a granularity of up to 15 second intervals but it is not necessary for the purposes of the application. Any other constraints on the product are subject to the internet browsing interface that is accessing our product. The UI will be intuitive and able to navigate data presentations, but will not allow manipulation of the data or underlying structure of the application.

2.2.5 Assumptions and Dependencies

The application will be dependant on the data acquisition server. If the data acquisition server is drastically changed or removed, the application will not be functional. Meter data within the application will also be limited by the functionality of the meters themselves. Should a meter malfunction, the energy data will not be gathered which may cause some of the data presentations to deviate from expected data. The use of the application will require a compatible web browsing interface, which may be limited to browsers such as Firefox, Chrome, and Internet Explorer. The functionality of the product will depend on those internet browsers performing as expected with regards to their intended functionality and internet access.

2.2.6 Apportioning of Requirements

Future versions of the application may include features such as cost tables, automated electronic invoice generation, energy billing analysis capabilities, budget analysis capabilities, and mobile energy data entry.

2.3 Specific Requirements

2.3.1 External Interfaces

AcquiSuite data acquisition servers [meters] made by Obvious

- Used for collecting electric, water, gas, steam, and other energy parameters over the web.
- Data is received through IP-based connection.
- Data can be reached anywhere with an internet connection, as long as the AcquiSuite is online.
- Data will be collected every 15 minutes.
- Data from the AcquiSuite servers will be stored into a database.

2.3.2 Functions

This section defines how the software system should behave with regards to input and output.

- The system shall receive and store data from AcquiSuite data acquisition servers into the database.
- The system shall have input validation measures in place to monitor incoming data and protect from malicious injections.
- The system shall retrieve data from the database and populate webpages with filtered datasets when prompted.
- The system shall have permission based restrictions for accessing certain data.
- The system shall generate alerts for offline buildings and high energy usage.
- The system shall generate emails for users to access a sign-up form and create an account.
- The system shall be able to create arbitrary combinations of datasets given user filters.
- The system shall be able to calculate rankings for all the buildings based on certain metrics like “kilowatt-hour consumption”.

2.3.3 Performance Requirements

This section describes the functionality requirements for the software as well as what a user should be able to accomplish when using the web application.

2.3.3.1 Users

The system will have 3 types of users: admin level users, registered users, and generic users. An administrative user will be able to add new meters, buildings, and other objects into the database. A registered user, for example the head of a department, can create their own stories with information and collections that are unique to their own interests or departments. Lastly, a generic user can simply view the publically facing web application and see public dashboards and browse public stories.

- A registered user should be able to customize dashboard layouts in a grid-based orientation. A *story* page should have a customizable layout where a user can add different *blocks* with information relevant to their personal needs.
- An administrative user should be able to add buildings to the database through a web form.
- An administrative user should be able to add data acquisitions servers.
- An administrative user should be able to download specific datasets as a .csv file.
- An administrative user should be able to customize public stories.

2.3.3.2 Data Management

- The web application should update data blocks every 15 minutes as new data is received into the database.
- The web application should allow users to create which are collections of dashboards. These *stories* are meant to bring related buildings and datasets together into intuitive groups, for instance “Residence Halls” or “Engineering Buildings.”
- The application should be able to filter building data by date range specifications.
- The web application should be able to scale up to as many buildings are on campus.
- The web application should properly create and store new entities (buildings, users, meters) into the database.

2.3.3.3 Visualization

- The web application should be able to generate different types of graphs for different *blocks*. For example load-profile charts, comparative line charts, and heat maps.
- The web application should have generic pages for each building in the database. This page will display a series of graphs and charts that outline energy usage for a particular building.
- The *blocks* on each page should automatically update as new data is received by the database.

2.3.3.4 Objects

The system should have three main entities:

- **Building** - A building on campus that is connected to a data acquisition server.
- **Meter** - A specific data acquisition server.
- **User** - A user with a specific role.

2.3.4 Logical Database Requirements

Most of the calls to the database will be requests for meter data from a specific building or group of buildings. Access to the database should be hidden from the user and only accessed from the back-end of the application itself. Any input into the database should be validated and encrypted, if applicable.

2.3.5 Design Constraints

Design constraints may include server availability which could harm scalability.

2.3.5.1 Standards Compliance

There may be standards when storing energy data based on The Office of Sustainability Standards. [Check with Client]

2.3.6 Software System Attributes

2.3.6.1 Reliability

The system will be reliable at the time of deployment if all data displayed in graphs and charts is correct.

2.3.6.2 Availability

The system should be consistently available as long as the servers are up and running. As a web application, the system will be available via URL.

2.3.6.3 Security

The system will validate input for duplicate and malicious content. The system will also have back-end functionality to protect user passwords through proper hashing. The web application will restrict access to certain pages and content based on user roles and permissions.

2.3.6.4 Maintainability

Maintainability for this application should be simple for anyone with web development experience. The application code will be organized into logical directories and sub directories that should mimic a full stack web application. There will be documentation that explains how to connect to the AcquiSuite servers and how to use our API for data collection.

2.3.6.5 Portability

Our web application should be visible and accessible from most web browsers. There are not any host-dependent constraints since it will be hosted from a central server and available to the public via the internet. The source code should stay on Github so modifications can be made easily.

include the original document, showing what you thought, at the time, was the project definition with the original Gantt chart Add (your client should have okay'd): What new requirements were added? What existing requirements were changed? What existing requirements were deleted? Why? Add: Final Gantt Chart as a record of what happened when.

3 DESIGN DOCUMENT (ORIGINAL)

3.1 Introduction

3.1.1 Purpose

The purpose of this Software Design Document (SDD) is to provide details about the architecture of the web application as well as details about each component. It will describe the underlying design of each component and their purpose within the context of the application that will allow Oregon State affiliates to effectively monitor energy usage on campus.

3.1.2 Scope

The software outlined in this document will function as an interface by which Oregon State students, faculty, and affiliates may monitor energy usage of various buildings located on the Oregon State Campus. It will act as a tool for users to make informed infrastructure decisions and adjustments or act as a display piece to be presented within buildings on campus. It will serve the OSU Office of Sustainability as a replacement to an outdated and costly implementation.

3.1.3 Overview

Oregon State University is constantly making strides to reduce its carbon footprint and reduce its energy consumption. There is a carbon neutrality goal for 2025 where the university is trying to generate as much energy as it uses and have a net carbon footprint of zero. Our web application aims to monitor the energy use of buildings on campus in order to create a visual representation of each building's consumption and incentivize people to monitor their consumption habits and reduce the campus's overall consumption.

Our web application will resemble an administrative dashboard with charts and graphs of energy use over time for individual buildings and subgroups of buildings on Oregon State's campus. Our application will have a series of public facing pages that will show energy usage for all the buildings on campus that have AcquiSuite meters as well as display user generated stories of personalized dashboards. Each page will be a grid-based dashboard with personalized blocks for displaying data through time-series charts or graphs. A user will be able to add different blocks to create unique dashboards for their own unique interests and subsets of buildings. An example of this would be a dashboard designated to only residence halls where each block shows a usage over time graph for energy consumption of each

residence hall. Users will be able to apply date filters to their blocks to generate different data sets and more explicit visualizations.

Our web application will be constructed using a MEAN Stack framework hosted on AWS. Our application will have a MongoDB database server and a Node.js application server hosted on a single virtual EC2 instance to make deployment easy and reliable.

3.1.4 Reference Material

- [1] “Dashboard”, v4-alpha.getbootstrap.com, 2017. [Online]. Available: <https://v4-alpha.getbootstrap.com/examples/dashboard/>. [Accessed 30- Nov- 2017].
- [2] “Using OAuth 2.0 to Access Google APIs”, developers.google.com, 2017. [Online]. Available: <https://developers.google.com/identity/>. [Accessed 30- Nov- 2017].
- [3] “Data-Driven Documents”, d3js.org, 2017. [Online]. Available: <https://d3js.org/>. [Accessed 30- Nov- 2017].
- [4] “ng”, angularjs.org, 2017. [Online]. Available: <https://docs.angularjs.org/api/ng>. [Accessed 30- Nov- 2017].
- [5] “AcquiSuite”, obvius.com, 2017. [Online]. Available: <http://www.obvius.com/Products/A8812>. [Accessed 30- Nov- 2017].

3.1.5 Definitions and Acronyms

This subsection will define terms and acronyms that are specific to the application that may otherwise be misunderstood or poorly interpreted.

- **Dashboard:** A grid-based information management tool for visually tracking and displaying metrics and data through graphs and charts.
- **Block:** An individual graph or chart depicting time-series data. More specifically, it is an HTML element containing one or more buildings, with filter buttons, informative text, and graphs.
- **Story:** A user generated collection of dashboards. What each story displays is based solely on user preferences. With multiple stories, the user can select between “presets” that can show different dashboards.
- **MEAN Stack:** An acronym used to define a full stack application engineered from the MongoDB, Express.js, Angular.js, and Node.js frameworks.
- **Bootstrap:** Bootstrap CSS is a front-end framework that uses component templates to easily generate different HTML elements like buttons, navigation, or forms.
- **OAuth 2.0:** OAuth 2.0 is an authorization protocol that grants authentication through tokens rather than credentials.
- **Filter:** Filters are essentially parameters for fetching data from that constrain the subset of data being received to the specifications of the filter parameters (i.e. a date range).
- **D3:** D3.js is a visualization framework that appends charts and graphs to DOM elements on a webpage.
- **AcquiSuite:** AcquiSuites are data acquisition servers made by a company called Obvius that post building meter data to a designated IP address.
- **AWS:** AWS is an acronym for Amazon Web Services which offers reliable cloud computing services for building and hosting web applications.
- **Passport:** Passport is authentication middleware for Node.js. Extremely flexible and modular, Passport can be unobtrusively dropped in to any Express-based web application. A comprehensive set of strategies support authentication using a username and password, Facebook, Twitter, etc.

3.2 System Overview

Our web application will provide users with the functionality to create unique dashboard-style collections of charts and graphs to analyze energy use from Oregon State’s campus buildings. Our system will be highly modular, containing multiple smaller components that provide their own individual functionality while sharing information across other components. Designing a modular architecture will simplify the code base and allow the developers to encapsulate specific data models as their own entities and keep abstractions in the system differentiable.

Our design also stems from the main constructs behind AngularJS and data injection. Most of our components and subsystems are, at large, data models and objects that hold all the necessary information to create graph and charts. With this design, AngularJS services will extract data from the session user and rendered components and generate render the desired graphs and views to the content container. This allows the application to break up back-end services as well as UI components to simplify how data is gathered and processed.

3.3 Design Viewpoints and Concerns

3.3.1 Stakeholders and Concerns

Developers concerns consist of simplifying complex model functions, replicable components, modular subsystems, and simple data access functionality. Addressing and capitalizing on these concerns will create an application design that will enhanced understanding, simplify implementation, and produce a high-end product. From a developer's standpoint, it is also essential to design an application that is ideal for both implementation and usability.

Client concerns revolve around satisfying the requirements of the application and providing all the necessary user functionality. The implementation should adhere to the client's requests and specifications while creating a user-friendly interface to perform all the necessary actions.

3.3.2 Design Viewpoints

3.3.2.1 Context Viewpoint

The design of this web application will greatly depend upon satisfying the user actions outlined in the requirements document. With that in mind, all components and subsystems should cater towards simplifying the user's experience while performing actions and navigating through the application. This viewpoint provides design constraints for individual component functionality, as well as the overall layout and structure of the application as a whole. Our design implementations should cater to a wide range of users while creating an elegant way for the user to create, edit, and remove content.

3.3.2.2 Composition Viewpoint

This application contains a number of components that are recurrently used throughout the application and across users. These include the basic data blocks, dashboards, and stories that a user can create. The design of this application should embrace the modularity of these components and provide a consistent model for individual functionality while maintaining the ability to communicate and pass data across different components or objects.

3.3.2.3 Dependency Viewpoint

With a design focused around modularity and individualized components, there needs to be designs for services and dependencies so each component can interact. Some examples of these are: how a block component depends on multiple entities contained inside the block object like the building objects stored within the block, the filter's current values, the graph template, and the graphing service. The design of this application should encapsulate the relationships between different modular components and provide intuitive means for inter-component communication.

3.3.2.4 Information Viewpoint

Our applications design should provide logical ways of storing data and receiving data from building meters. This involves an explicit declaration of individual data models and how data is created and stored. Our design will attempt to create a modular data structure where data specific to certain entities reside in the entity objects themselves.

This allows the application to access data from an individual component as it gets created or moved throughout the application.

3.3.2.5 Logical Viewpoint

The application should maintain a consistent model for each component which can be used in different ways for different purposes. An entity should contain and provide the necessary information as it is needed throughout various areas of the application. The design of the application as a whole should consider where certain data is needed, and what data models must be present to satisfy the system's requests.

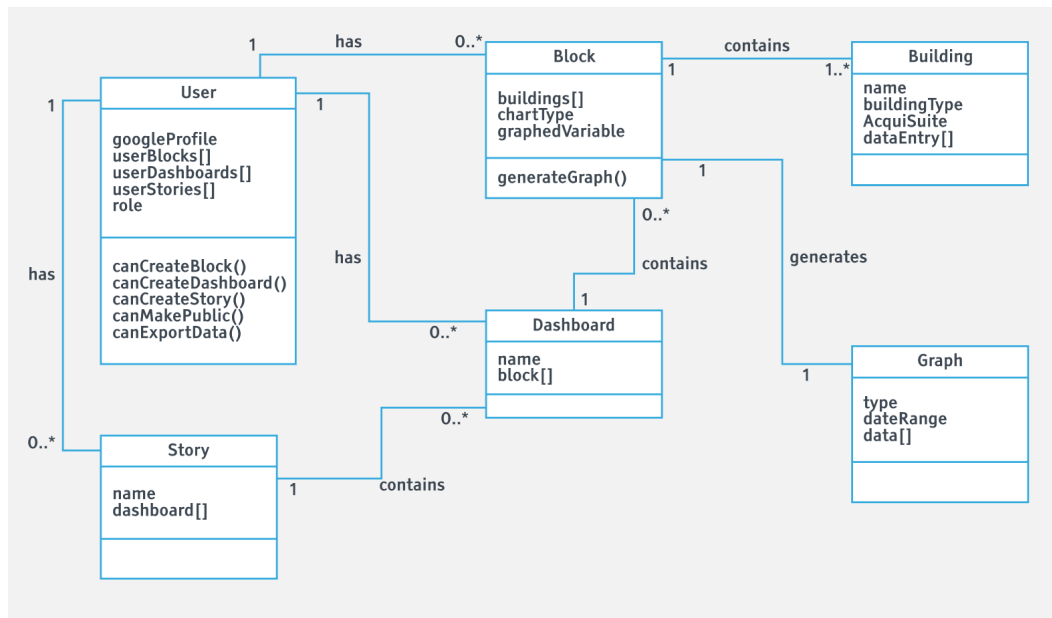


Figure 1: A UML diagram to show component relationships.

3.3.2.6 Interface Viewpoint

Users will require a front end interface to interact with the application and utilize its functionality. The design of these interfaces should focus on satisfying all user actions in a simple and appealing manner. Users should be able to intuitively navigate the application and perform essential tasks without confusion or error.

3.3.2.7 Structure Viewpoint

The design of this application describes the structure of independent modular components and their relationships to other entities. From a developer's standpoint, organizing the structure and interdependence of different components is essential when implementing a complex system. This application's design highlights key individual components along with their individual functionality and relationships with other entities.

3.3.2.8 Interaction Viewpoint

Each object component in the application should not need any logic to handle its interactions with other components. This functionality should be performed by services and model functions implemented in the back-end of the application. The design of this application aims to keep data and entities separate from the logical functionality required to extract and share data across different components.

3.4 System Architecture

3.4.1 Architectural Design

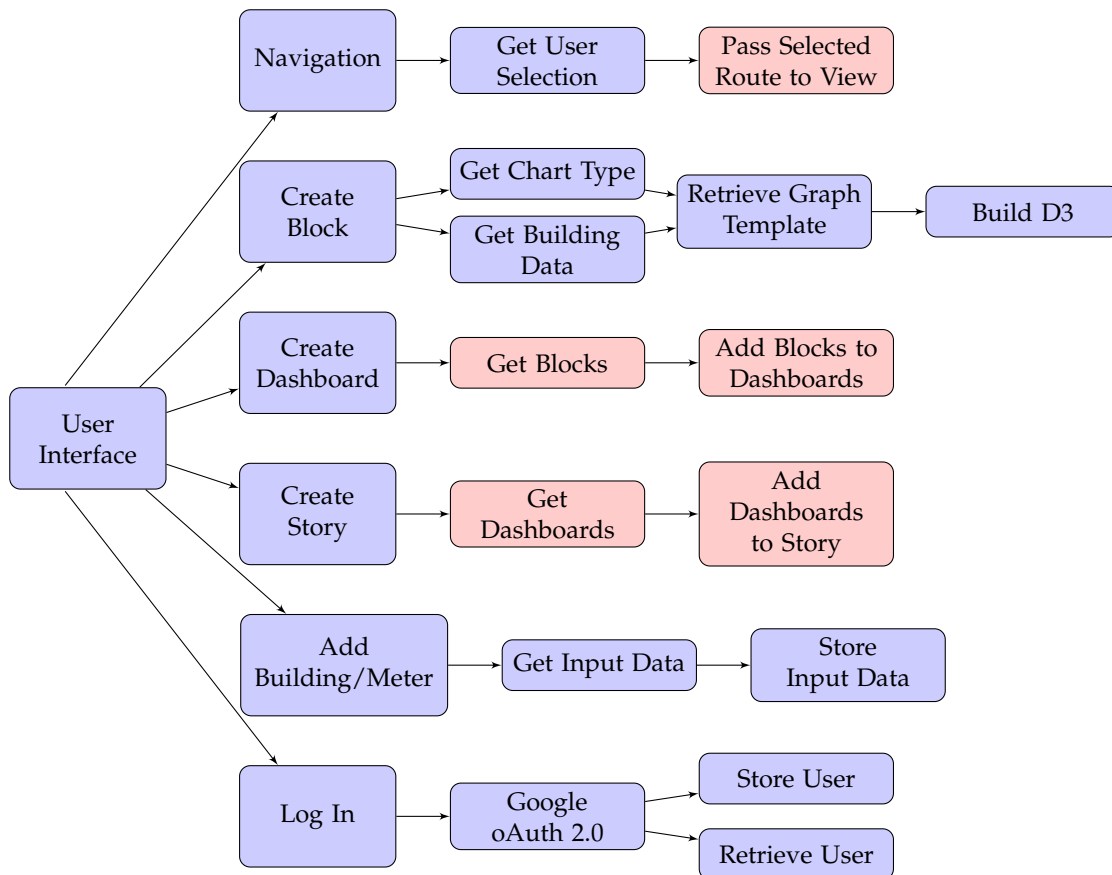


Figure 2: An overview of the major application systems.

3.4.2 Decomposition Description

All database operations (retrieving data and storing data) are denoted with red fill and will be carried out by an AngularJS “Service” within the component’s controller.

3.4.2.1 Log In



Figure 3: A functional representation of the log in subsystem.

3.4.2.2 Navigation

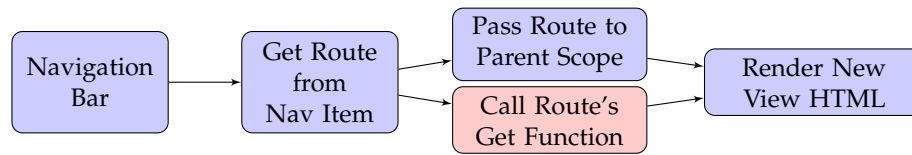


Figure 4: A functional representation of the navigation subsystem.

3.4.2.3 Create Block

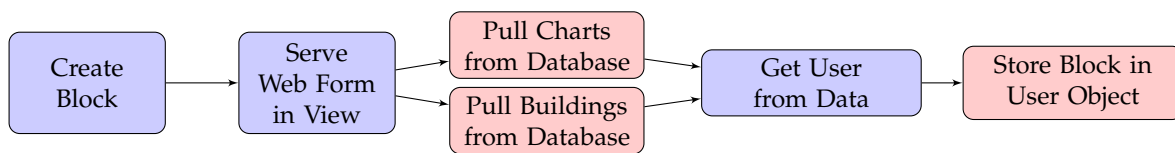


Figure 5: A functional representation of the block creation subsystem.

3.4.2.4 View Blocks

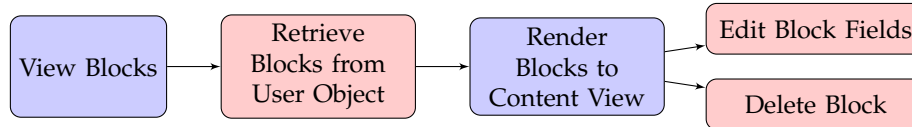


Figure 6: A functional representation of the block view subsystem.

User created blocks will be stored in their respective “Blocks” array. This allows a user to keep track of the specific blocks they create and reuse them to create stories.

3.4.2.5 Create Dashboard

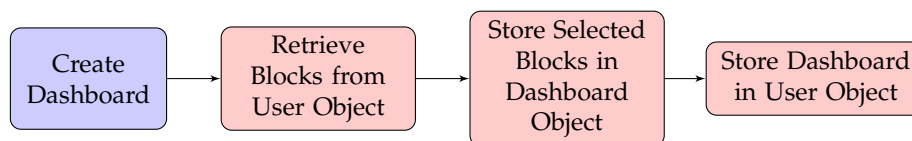


Figure 7: A functional representation of creating a dashboard object from user created block.

A dashboard is a collection of blocks. The dashboard component provides a way to group blocks together based on user preference and display them in the content view by simply rendering each block stored.

3.4.2.6 Create Story

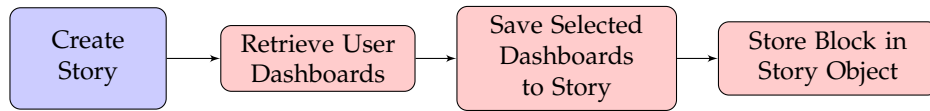


Figure 8: A functional representation of the story creation subsystem.

Stories are stored in the User object who created them in an array “Stories.” This allows the application to render stories created by the session user easily and will eliminate the use of SQL style querying to retrieve stories based on a user key.

3.4.2.7 Add Building/Meter

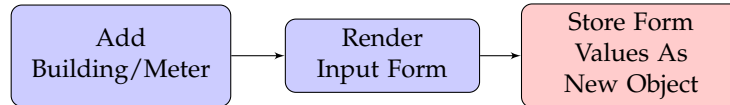


Figure 9: A functional representation for storing new building and meter objects to the database.

3.4.2.8 Create Graph

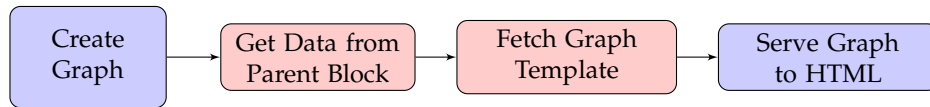


Figure 10: A functional representation for building a graph in a block.

To render a chart or graph to the webpage, our application will gather all the parameters from the block component that contains the graph (the building(s), the chart type, and the date range). The the graph component will then fetch a D3 template for its specific chart type, input the data from the block, and serve it to the dashboard.

3.4.2.9 Get AcquiSuite Data

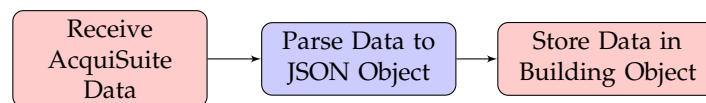


Figure 11: A functional representation for storing AcquiSuite data to a building object.

Timestamped data entries from AcquiSuite Servers will be contained in their respective building objects from which the data was sent. Similar to the rational behind storing blocks in the user object that created them, each building will have

an array of meter data sorted by timestamp. This will allow our application to use a service to retrieve the data for any graphs or charts directly from the buildings objects in the relative block. A graph service can then fetch the building data, apply filters (like date ranges), and pass it to a D3 graph to render. This minimizes the amount of total data to be parsed and collected by narrowing the data to the relative block.

[**Note:** Subject to change in later revisions. The methodology of how to receive data from AcquiSuite data acquisition servers is still being designed.]

3.4.3 *Design Rationale*

The system's architecture revolves around a heavily modular system where individual components have their own functionality and are able to work cohesively with other components. The modular architecture also breaks up the back-end components into smaller services which simplifies functionality across the entire application.

3.5 **Component Design**

This subsection is designated to describing individual components in greater detail, including the type of element, designated functionality, and the reasoning behind their design. Overall, this subsection aims to provide an in-depth view of the application components and describe their individual roles in the application as a whole.

3.5.1 *Block*

A Block is a component in our application that will act as a container for a graph, filter options, and provide information about the data in the graph.

3.5.1.1 *Element*

A Block is an HTML element that uses the Bootstrap 4 "card" component for its design. It will have a title bar with the Block's name, then a content container split into a grid. On the left, the Block will display the building(s) being graphed, contain a column of filter buttons, and display the D3 graph in the right column.

3.5.1.2 *Functionality*

The Block component neatly organizes data for the user by encapsulating the data, filters, and graph components into one container. A Block needs to pull data from the "building" objects that it contains, gather data from any filters, and pass all this data into the correct graph template to generate a D3 graph. There will be a service within the Block's AngularJS controller that will gather this data and pass the desired output to the dashboard.

3.5.1.3 *Rationale*

The Block component is an efficient way to create a reusable HTML template that can be reproduced throughout the application. This provides a simple and consistent process for displaying graphs and data. Additionally, the block component yields a compact solution for organizing data from multiple sources.

3.5.2 *Dashboard*

The dashboard component acts a way to store collections of block objects based on user preference.

3.5.2.1 *Element*

A dashboard is an object in the database that holds a collection of block objects.

3.5.2.2 *Functionality*

The dashboard object is a way to conveniently package blocks together into groups so users can reuse the same groups or create unique collections of different blocks. When “viewing” a dashboard, the application will simply iterate through the block objects and render them to the view container in a vertical list.

3.5.2.3 *Rationale*

The dashboard components provides a simple way to organize data keep track of user generated collections. The dashboard component will work nicely with AngularJS to render an HTML element for every object in an array, which in this case are the blocks in the dashboard.

3.5.3 *Login*

The Login component is an HTML button that will allow users to login with a Gmail account by redirecting the user to Google’s oAuth 2.0 API.

3.5.3.1 *Element*

The Login function will be a button in the corner of the screen or somewhere in the navbar. The button will change to “logout” if the user is already logged in.

3.5.3.2 *Functionality*

Clicking the Login component will redirect the user to our authentication service. The application will use “PassportJS” as an authentication middleware which authenticates the user with Google oAuth 2.0 tokens and sets the req.user variable upon redirecting to the specified route. Once a user is logged in, they will have access to additional application functionality and navigation items based on role.

3.5.3.3 *Rationale*

From a developer standpoint, requiring users to login for certain features will make separation of privileges much easier. With authentication, users will also be able to have the convenience of Google SSO.

3.5.4 *Navigation Bar*

The Navigation bar is a commonly used UI feature that provides users the ability to navigate through different webpages of the application.

3.5.4.1 *Element*

The navigation bar will be an HTML element with differentiable “nav-items” that render different views to the content view space. There will be a fixed-top navigation bar for generic item and a side bar navigation for more explicit functionalities such as admin controls.

3.5.4.2 *Functionality*

Clicking on a nav-item has one of two functionalities:

- Redirects the user to a new webpage by having the route’s get function perform a redirect.
- Renders a new view to the webpage by having the route’s get function render a new HTML view.

The navigation bar shows which item is currently being served by changing the style of the active nav-item to be different than the rest.

3.5.4.3 *Rationale*

From a user perspective, having a navigation bar to navigate around the main features is an extremely useful UI design decision. From a design perspective, having a dedicated area to insert useful links is much more streamline than listing them. The navigation bar is a very useful tool for serving different content to the view space statically.

3.5.5 *Graph Component*

Our application will contain a graph component that accepts a graph type and data as input and constructs a graph using the D3 visualization framework.

3.5.5.1 *Element*

A graph component will be an SVG element, created with D3, that is served to the webpage inside of a block.

3.5.5.2 *Functionality*

The graph component’s functionality involves gathering different input parameters from the parent block component, fetching a template based on the graph type, and creating a unique graph. The graph component should have its own service to handle all the data organization and create a list of uniform parameters than can be passed to a D3 graph template.

3.5.5.3 *Rationale*

Having a structure like this simplifies the act of generating graphs by having a series of templates that can be fed data in a specific format. The graph service can take all the data and different filter parameters from the block object and format it into the right structure. All the computationally heavy functionality is black-boxed in the graph service and keeps the functionality handled by the user and front-end at a minimal. Another benefit provided by the graph component is that it can change the graph dynamically. Our application may include filters in the block component like a date range or a filter to change the graphed variable, which can trigger the graph service to reconstruct the data and create a new graph to serve based on the applied filters.

3.5.6 *Story*

A story is a container for user dashboards to be stored and retrieved. A story is an abstract object that allows the user to create unique collections of dashboards with general theme or purpose.

3.5.6.1 *Element*

A story will be an database object that contains collections of dashboards.

3.5.6.2 *Functionality*

Each story will have a collections of dashboards. The number of dashboards in the collection will vary from story to story based on user preference.

3.5.7 *Content View*

The content view is a portion of the webpage where different views can be rendered to provide different content.

3.5.7.1 *Element*

The content view will be an HTML container that resides within the context of the webpage. The user will be able to change the content of the container by selecting different navigation items.

3.5.7.2 *Functionality*

When the user clicks on a button or list on the side bar, it will pass a route to the scope variable in the view container. Once that route is called, the Angular function “ng-include” will inject the rendered HTML into the context of the webpage.

3.5.7.3 *Rationale*

From a developer standpoint, having a content view will dramatically reduce clutter both in the code and in the UI. The content view allows the application to serve dynamic content without reloading the webpage and creates a simple way to perform navigation services.

3.6 **Human Interface Design**

3.6.1 *Overview of User Interface*

3.6.1.1 *Public User*

A public user is a user without a registered account or a user that is not logged in to their registered account. The authentication of a user is based on if they have a registered Oregon State Gmail domain. Public users will have limited access to features and data on the website. For example, the left side navigation bar will be hidden from a public user. Public users will not be able to create personalized dashboards or stories. Public users will be allowed to view public user generated dashboards as well as general building dashboards. Essentially, public users will not be able to create personalized data sets or modify elements of the site but will be allowed to view general public data.

3.6.1.2 *Authorized User*

An authorized user is a user with a registered Oregon State Gmail account but does not have administrative privileges. When an authorized user is logged into their account, they have greater privileges than a public user as well as all privileges granted to a public user. They are allowed access to a left side navigation bar that has special functionality. Unique functionality granted by the vertical navigation bar include: viewing of personal and public dashboards, viewing of personal and public stories, and access to personal settings. Authorized users may create personalized dashboards and stories for unique purposes.

3.6.1.3 *Administrative User*

Administrative users will have the highest privileges and control of the website. They have all privileges as a public or authorized user. Administrative users will be able to add or remove buildings, meters, public dashboards, and public stories as well as perform general administration tasks.

3.6.2 *Screen Images*

3.6.2.1 *Home Webpage (Public Access)*

link: See Appendix B for image

If a general public user does not have an account but wishes to access energy monitoring data, they will have a unique experience with the site that is different than those who are logged in to authorized accounts. For instance, they will not have the ability to create personal dashboards or stories, but will be allowed to view public stories, public dashboards, and general building dashboards. They will not have access to the same navigation bar on the left side of the screen as a logged in user.

3.6.2.2 *Home Webpage (Logged in Access)*

link: See Appendix B for image

If an authorized user wishes to access energy monitoring data, they will be able to have special privileges. For instance, they will have the ability to create personal dashboards or stories which they may specify as private or public. They will have access to a navigation bar on the left side of the screen which allows for unique data viewing and privileges.

3.6.2.3 *Buildings Webpage*

link: See Appendix B for image

Users may select the “Buildings” tab in the navigation bar to bring up an array of building cards to choose from. Every building on campus that is gathering metering data associated with this application will have a card in this array. A user may select a building card to bring up a generalized dashboard of information for each specific building. The user will be able to search for buildings by name in a search bar. Administrators will be able to add buildings as the metering system expands to new buildings.

3.6.2.4 Selected Building Page

link: See Appendix B for image

Once a user has selected a building card, a generalized dashboard will be displayed to them. All buildings associated with this application will contain the same generalized dashboard to be viewed. These dashboards will display various cards of data detailing the most general information about their respective buildings. A user may full screen the application to remove menu clutter and allow for a more presentable display of the data. Users will have the option to modify personal dashboards or public dashboards by adding a card, adjusting the organization of cards, or editing the information on the cards.

3.6.2.5 Dashboards Page

link: See Appendix B for image

Users may select the “Dashboards” tab in the navigation bar to bring up a list of user-generated dashboards. The dashboards will contain user specified arrangements of data cards. The list of dashboards will detail the name of each dashboard, the user that created each dashboard, and the date and time of when each was created. The user will be able to search for dashboards by name in a search bar. A user may select a dashboard from the list to view or use the options menu for more specific tasks.

3.6.2.6 Selected Dashboard Webpage

link: See Appendix B for image

Once a user has selected a dashboard, a user-generated dashboard will be displayed to them. The dashboard will display various cards of data presenting information about various buildings energy usage. A user may full screen the application to remove menu clutter and allow for a more presentable display of the data.

3.6.2.7 Story Webpage

link: See Appendix B for image

Users may select the “My Stories” tab in the navigation bar to bring up a drop down list of their personal stories in the left side navigation bar. Each story will contain a list of dashboards that was generated by the current user to their preference. The list of stories will include both their private and public stories. A user may select a story from the list to view or use the options menu for more specific tasks. include the original document, showing what you thought, at the time, was the project design Add (your client should have okay’d): What design aspects were changed, deleted or added? Why?

4 TECH REVIEW (ORIGINAL)

5 WEEKLY BLOG POSTS (ALL TEAM MEMBERS’ POSTS)

formatted nicely and clearly distinct from one another include all members posts, clearly distinct, clearly organized

6 FINAL POSTER

scaled to fit on a color-printed on a single 8.5"x11" paper.

7 PROJECT DOCUMENTATION

7.1 How does your project work? (Could include the following...)

- What is its structure?
- What is its Theory of Operation?
- Block and flow diagrams are good here.

7.2 How does one install your software, if any?

7.3 How does one run it?

7.4 Are there any special hardware, OS, or runtime requirements to run your software?

7.5 Any user guides, API documentation, etc.

This needs to be detailed enough to recreate and/or use your project!

8 RECOMMENDED TECHNICAL RESOURCES FOR LEARNING MORE

What web sites were helpful? (Listed in order of helpfulness.) What, if any, reference books really helped? Were there any people on campus that were really helpful?

9 CONCLUSIONS AND REFLECTIONS (EACH TEAM MEMBER ANSWERS ALL QUESTIONS INDIVIDUALLY)

What technical information did you learn? What non-technical information did you learn? What have you learned about project work? What have you learned about project management? What have you learned about working in teams? If you could do it all over, what would you do differently? Be honest here – no B.S.

10 APPENDIX 1: ESSENTIAL CODE LISTINGS.

You don't have to include absolutely everything, but if someone wants to understand your project, there should be enough here to learn from. If you worked within a larger project, something like a patch file might be a good way to go.

11 APPENDIX 2: ANYTHING ELSE YOU WANT TO INCLUDE.

Photos, etc.

11.1 Design Document Images

11.1.1 Home Webpage (Public Access)

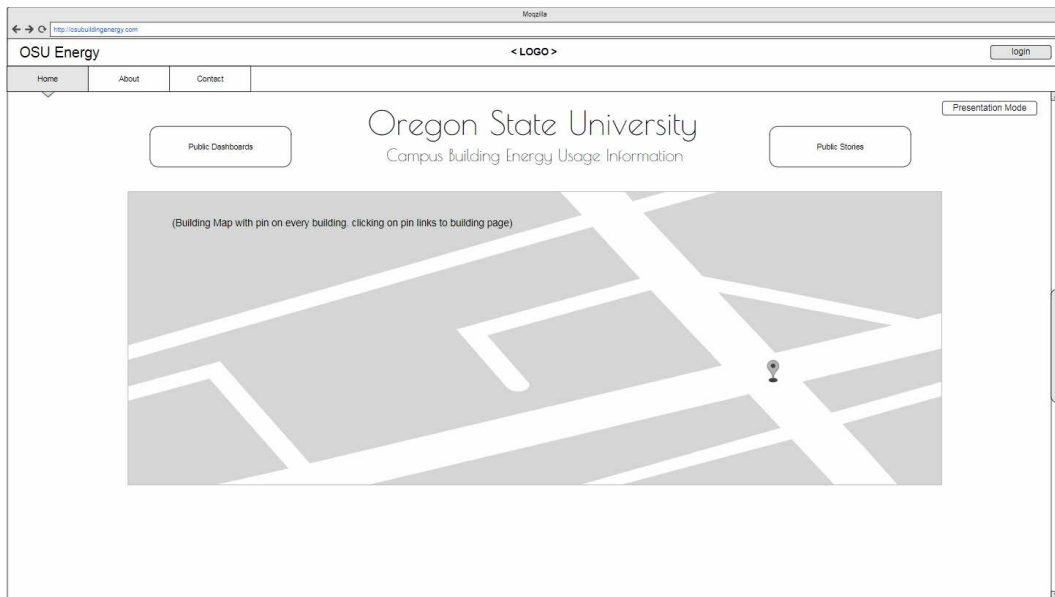


Figure 12: A mock-up of a home page from a general public user access perspective (not logged in to an account).

11.1.2 Home Webpage (Logged in Access)

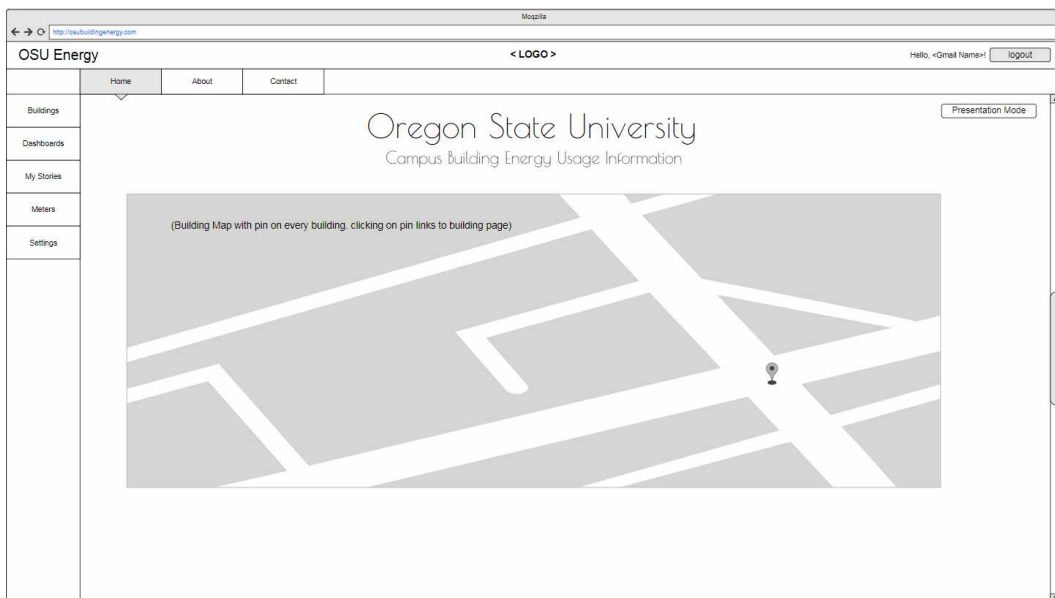


Figure 13: A mock-up of a home page from an authorized user access perspective (logged in to an account).

11.1.3 Buildings Webpage

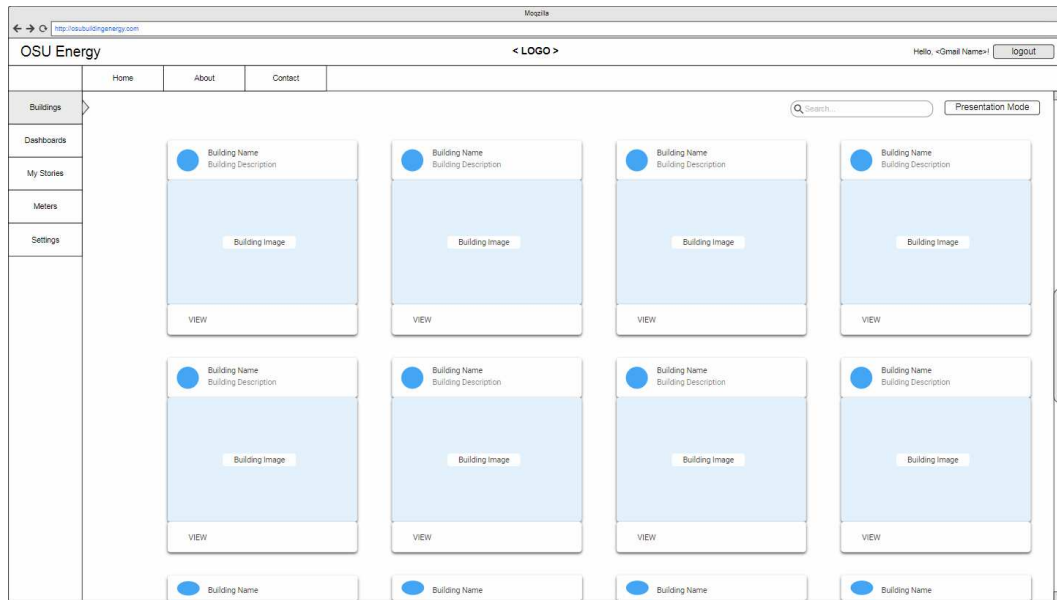


Figure 14: A mock-up of the webpage that shows all buildings.

11.1.4 Selected Building Page

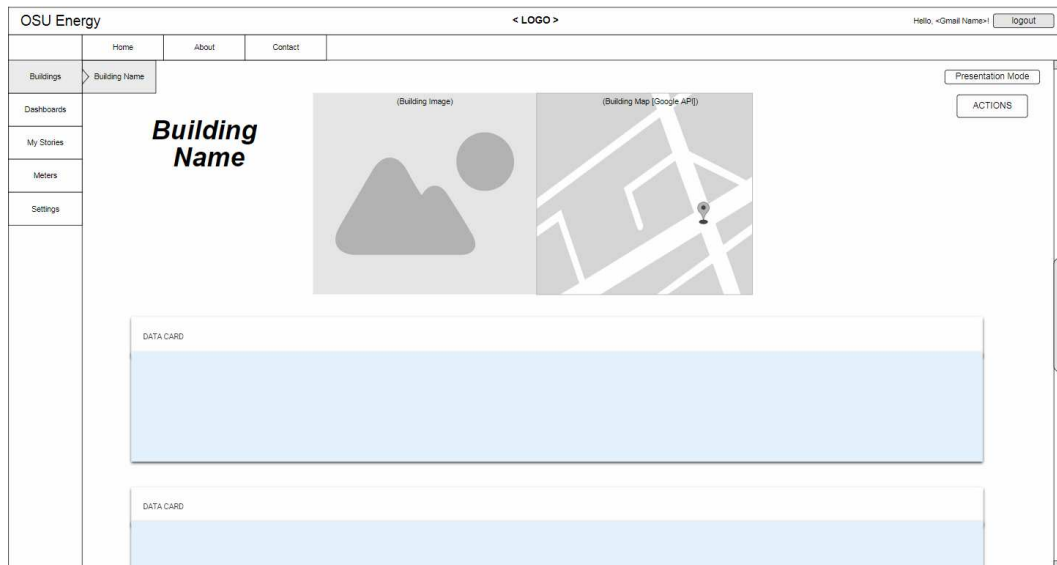


Figure 15: A mock-up of a selected building page.

11.1.5 Dashboards Page

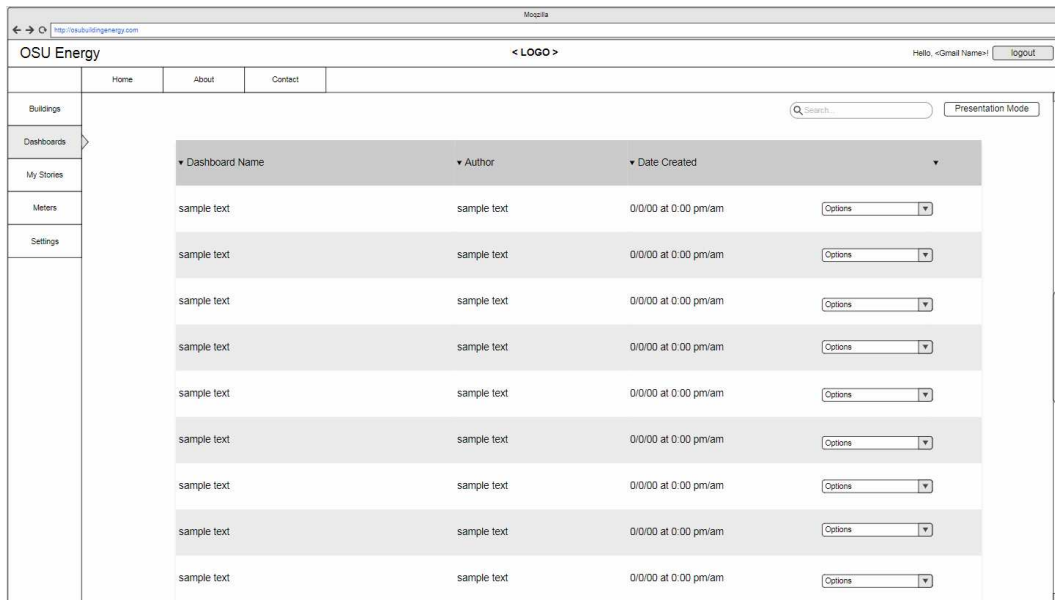


Figure 16: A mock-up page of a list of user generated dashboards.

11.1.6 Selected Dashboard Webpage

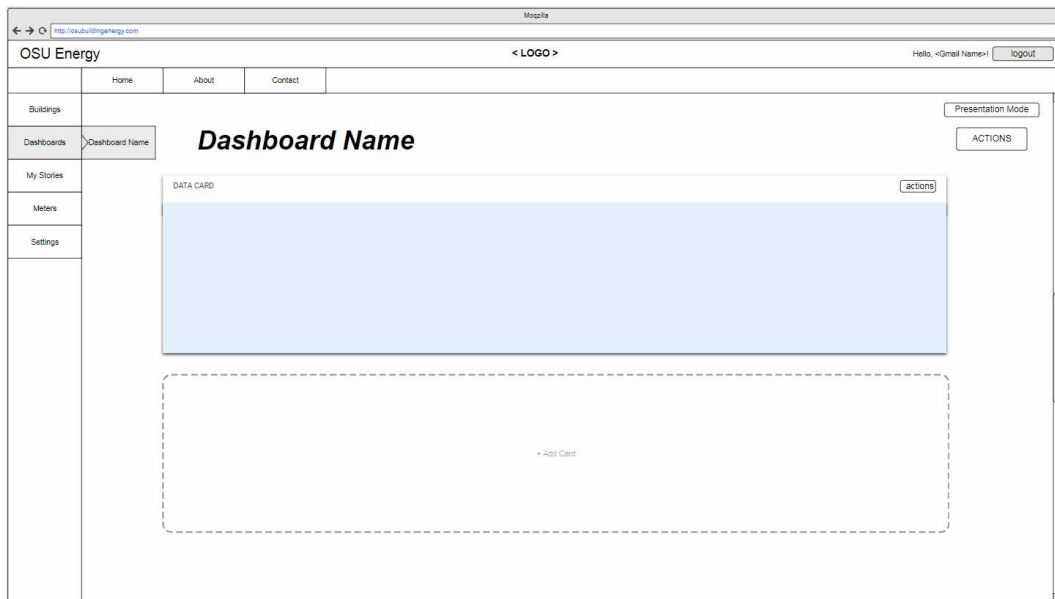


Figure 17: A mock-up of a selected dashboard webpage.

11.1.7 Stories Webpage

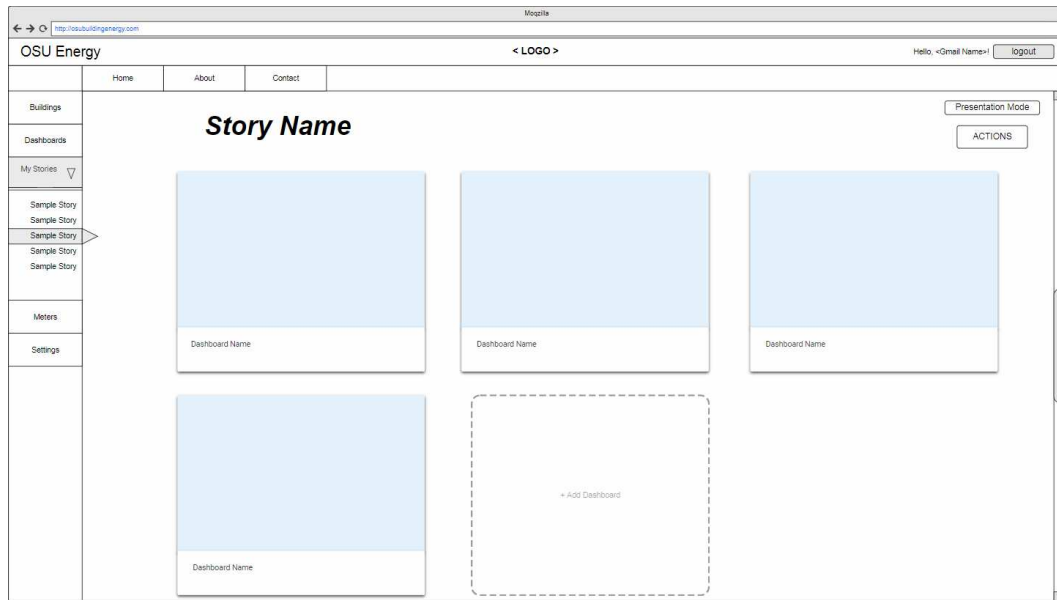


Figure 18: A mock-up of a list of user generated story Webpages.

11.2 Technology Review Images