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

Daniel Schroeder, Aubrey Thenell, Parker Bruni

Abstract

CONTENTS

1 INTRODUCTION

1.1 Purpose

Identify the purpose of this SDD and its intended audience. (e.g. This software design document describes the architecture and system design of XX. .).

1.2 Scope

Provide a description and scope of the software and explain the goals, objectives and benefits of your project. This will provide the basis for the brief description of your product.

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 0. 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 that has 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 show general data use for all the buildings on campus that have AcquiSuites as well as 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 their pages to create a unique dashboard for their own unique interests and subsets of buildings. An example of this would be a page 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.

1.4 Reference Material

List any documents, if any, which were used as sources of information for the test plan.

1.5 Definitions and Acronyms

Provide definitions of all terms, acronyms, and abbreviations that might exist to properly interpret the SDD. These definitions should be items used in the SDD that are most likely not known to the audience.

- **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.
- **Page:** A dashboard of different blocks.
- **Story:** A user generated collection of pages.
- **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.

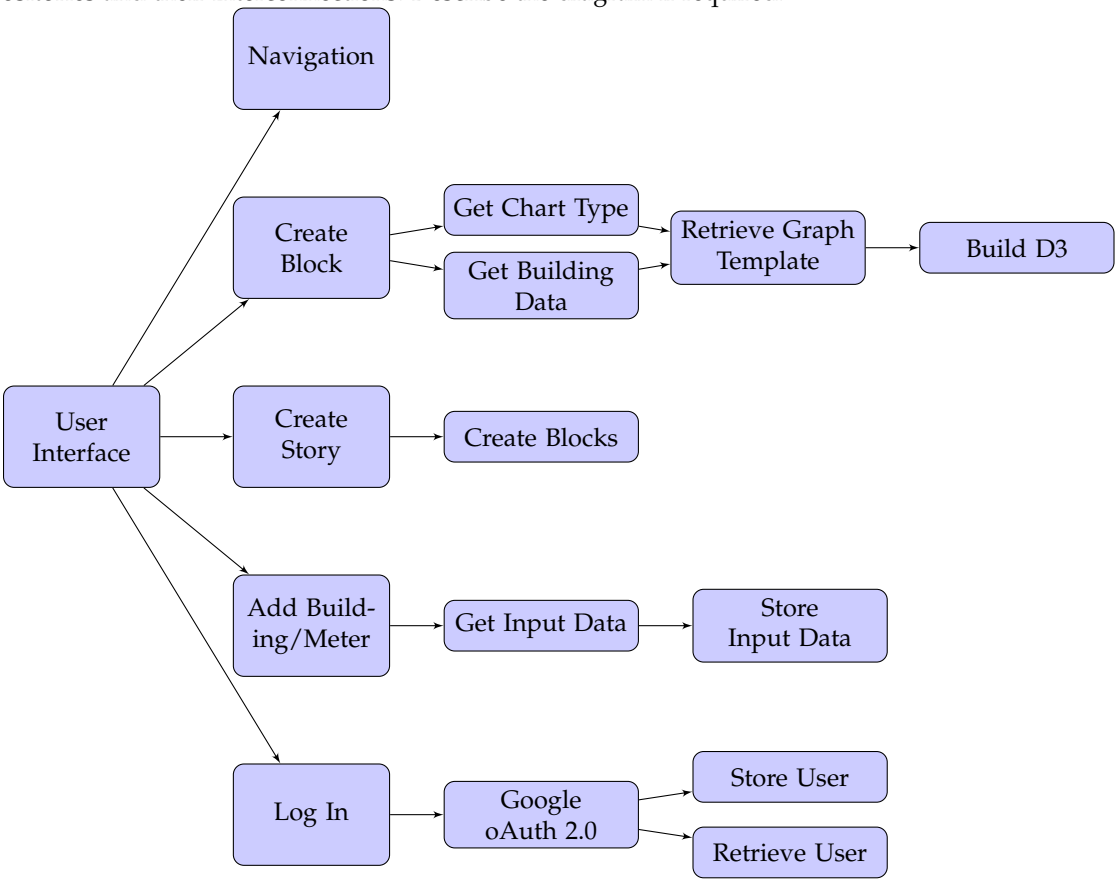
2 System Overview

Give a general description of the functionality, context and design of your project. Provide any background information if necessary.
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 that has 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 show general data use for all the buildings on campus that have AcquiSuites as well as 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 their pages to create a unique dashboard for their own unique interests and subsets of buildings. An example of this would be a page 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 System Architecture

3.1 Architectural Design

Develop a modular program structure and explain the relationships between the modules to achieve the complete functionality of the system. This is a high level overview of how responsibilities of the system were partitioned and then assigned to subsystems. Identify each high level subsystem and the roles or responsibilities assigned to it. Describe how these subsystems collaborate with each other in order to achieve the desired functionality. Dont go into too much detail about the individual subsystems. The main purpose is to gain a general understanding of how and why the system was decomposed, and how the individual parts work together. Provide a diagram showing the major subsystems and data repositories and their interconnections. Describe the diagram if required.



3.2 Decomposition Description

Provide a decomposition of the subsystems in the architectural design. Supplement with text as needed. You may choose to give a functional description or an object oriented description. For a functional description, put top level data flow diagram (DFD) and structural decomposition diagrams. For an OO description, put subsystem model, object diagrams, generalization hierarchy diagram(s) (if any), aggregation hierarchy diagram(s) (if any), interface specifications, and sequence diagrams here.

3.2.1 Log In

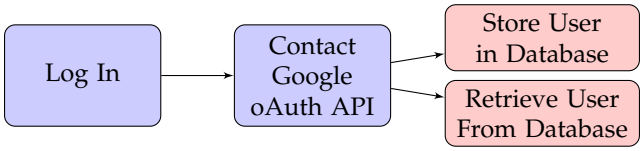


Figure 1: A functional representation of the log in subsystem.

3.2.2 Navigation

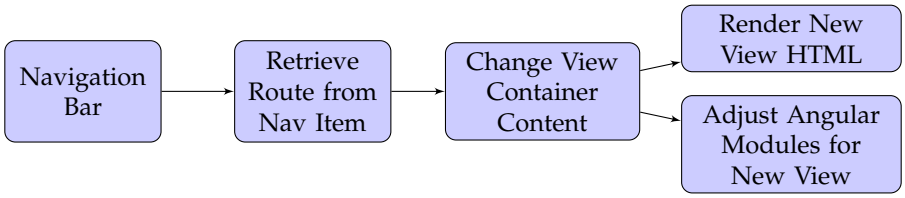


Figure 2: A functional representation of the navigation subsystem.

3.2.3 Create Block

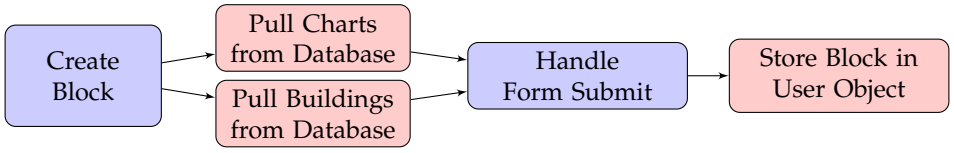


Figure 3: A functional representation of the block creation subsystem.

All database operations (retrieving charts, retrieving buildings, storing block) will be a “Service” of the block view.

3.2.4 View Blocks

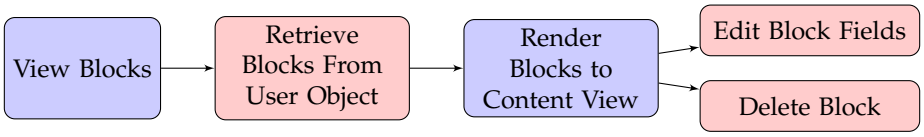


Figure 4: A functional representation of the block view subsystem.

Blocks are stored in the User object who created them in an array “Blocks.” This way a user can keep track of the specific blocks they create and reuse them to create stories.

3.2.5 Create Story

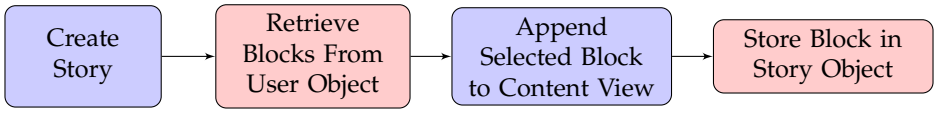


Figure 5: A functional representation of the story creation subsystem.

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.

3.2.6 Add Building/Meter

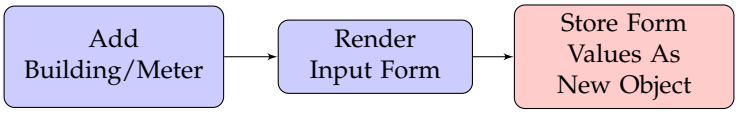


Figure 6: A functional representation for storing new building and meter objects to the database.

3.2.7 Get AcquiSuite Data

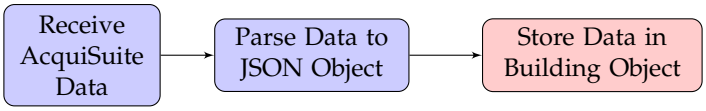


Figure 7: A functional representation for storing AcquiSuite data to a building object.

Timestamped data entries from Acquisuite Servers will live in the building objects the data came from. 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.

3.2.8 Create Graph

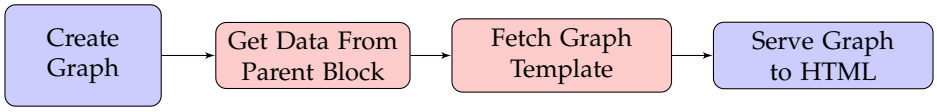


Figure 8: A functional representation for building a graph in a block.

To render a chart or graph to the page, 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). Then the graph component will fetch a D3 template for its specific chart type, input the data from the block, and serve it to the page.

3.3 Design Rationale

Discuss the rationale for selecting the architecture described in 3.1 including critical issues and trade/offs that was considered. You may discuss other architectures that were considered, provided that you explain why you didnt choose them.

4 Component Design

In this section, we take a closer look at what each component does in a more systematic way. If you gave a functional description in section 3.2, provide a summary of your algorithm for each function listed in 3.2 in procedural description language (PDL) or pseudo code. If you gave an OO description, summarize each object member function for all the objects listed in 3.2 in PDL or pseudocode. Describe any local data when necessary.

4.1 Block

A block is a component in our application that contains a graph and information about the data in the graph. For Example:

- A title
- Building(s)
- Date Range
- Variable being plotted

4.1.1 Element

4.1.2 Functionality

4.1.3 Rationale

5 Human Interface Design

5.1 Overview of User Interface

Describe the functionality of the system from the users perspective. Explain how the user will be able to use your system to complete all the expected features and the feedback information that will be displayed for the user.

5.2 Screen Images

Display screenshots showing the interface from the users perspective. These can be hand drawn or you can use an automated drawing tool. Just make them as accurate as possible. (Graph paper works well.)

6 Requirements Matrix

Provide a cross–reference that traces components and data structures to the requirements in your SRS document. Use a tabular format to show which system components satisfy each of the functional requirements from the SRS. Refer to the functional requirements by the numbers/codes that you gave them in the SRS.

7 Appendixes

Appendices may be included, either directly or by reference, to provide supporting details that could aid in the understanding of the Software Design Document.