Design and Implementation of a Secure Web Platform For a Building Energy Management Open Source Software

Kruthika Rathinavel

Thesis submitted to the faculty of the
Virginia Polytechnic Institute and State University
In partial fulfillment of the requirements for the degree of

Master of Science
In
Computer Engineering

Saifur Rahman, Chair
Manisa Pipattanasomporn
Chang-Tien Lu

July 1, 2015
Arlington, Virginia

Keywords: building energy management system web platform, open source, data modeling, user interface and security

Design and Implementation of a Secure Web Platform For a Building Energy Management Open Source Software

Kruthika Rathinavel

ABSTRACT

Commercial buildings consume more than 40% of the total energy consumption in the United States. Almost 90% of these buildings are small- and medium-sized buildings that do not have a Building Energy Management (BEM) system. The reasons behind this are – lack of awareness, unavailability of inexpensive packaged solutions, and disincentive to invest in a BEM system if the tenant is not the owner.

Several open source tools and technologies have emerged recently that can be used for building automation and energy management. However, none of these systems is turnkey and deployment ready. They also lack consistent and intuitive navigation, security, and performance required for a BEM system.

The overall project - of which this thesis research is a part - addresses the design and implementation of an open source secure web based user platform to monitor, schedule, control, and perform functions needed for a BEM system serving small and medium-size buildings. The focus of this work are: principles of intuitive graphical user interface design, abstracting device functions into a comprehensive data model, identifying threats and vulnerabilities, and implementing a security framework for the web platform.

Monitor and control solutions for devices such as load controllers and sensors are abstracted and their decentralized control strategies are proposed and implemented using an open source robust scalable user platform accessible locally and remotely. The user platform is open-source, scalable, provides role-based access, dynamic, and modular in design. The comprehensive data model includes a user management model, device model, session model, and a scheduling model. The data model is designed to be flexible, robust and can be extended for any new device type. Security risks are analyzed using a threat model to identify security goals. The proposed security framework includes user authentication, device approval, role-based access, secure information exchange protocols, and web platform security. Performance of the user interface platform is evaluated for responsiveness in different screen sizes, page response times, throughput, and the performance of client side entities.

This work was supported in part by the U.S. Department of Energy under Grant# DE-EE-0006352.

Acknowledgements

I would like to thank my advisor, Dr. Saifur Rahman for his guidance and support throughout my graduate studies at Virginia Tech and my research at Advanced Research Institute (ARI). You provided the needed clarity throughout my research. Your spirit and hard-working nature encouraged to work harder and stay focused on my goals.

I thank Dr. Manisa Pipattanasomporn for her invaluable guidance throughput my graduate studies. You were my mentor and guide. You helped me organize all of my research into quality documents and clarified ideas whenever I needed it. You taught me to keep my calm when I was under stress. I sincerely thank you for all the support you have provided me and being a good friend whenever I needed companionship.

I am grateful to Dr. Chang Tien-Lu who served as a member of my committee. I thank you for your guidance when I first joined Virginia Tech and throughout my graduate studies. You helped me undertake research at the Advanced Research Institute. You have always been my mentor guiding me through my Information Retrieval research and guided me toward publishing my first paper. Thank you for your support and guidance.

I thank Dr. Murat Kuzlu for his unwavering support throughout my research at ARI. I thank you for your guidance and for boosting my spirits to perform better. I also thank Yonael Teklu and Jinghe Zhang for being good friends and cheering me up.

I am grateful to the BEMOSS Team and all of my friends at ARI. They have been a good support and have helped me learn and conceptualize ideas throughout my research. Every day here was a challenge and made me stronger because of you all. Our discussions inspired a lot of valuable research ideas.

Finally, I would like to dedicate this thesis to my husband, Shankar Natarajan. Without your encouragement and support, this would not have been possible. Your love has been the key to my success. I would like to thank my parents, my brother and my in-laws for their constant support and encouragement.

Table of Contents

List of Tables

List of Figures

List of Acronyms and Abbreviations

BEM	Building Energy Management
BEMOSS	Building Energy Management Open Source Software
REST	Representational State Transfer
MQ	Message Queue
URL	Uniform Resource Locator
URI	Uniform Resource Identifier
sMAP	Simple Measurement and Actuation Profile
HTTP	Hyper Text Transfer Protocol
HTTPS	Hyper Text Transfer Protocol Secure
WS	Web Socket
WSS	Web Socket Secure
VAV	Variable Air Volume
RTU	Roof Top Unit
UI	User Interface
NIST	National Institute of Standards and Technology
HMAC	Hash-based Message Authentication Code
SHA	Secure Hash Algorithm
PBKDF	Password Based Key Derivation Function
AJAX	Asynchronous JavaScript and XML
DOM	Document Object Model
PNNL	Pacific Northwest National Laboratory
IEB	Information Exchange Bus
SMS	Short Message Service
SSH	Secure Shell
FTP	File Transfer Protocol
SFTP	Secure File Transfer Protocol
JSON	JavaScript Object Notation
RDBMS	Relational Database Management System
SMTP	Simple Mail Transfer Protocol
HMI	Human Machine Interface
HVAC	Heating, Ventilation and Air Conditioning
IEEE	Institute for Electrical and Electronics Engineers
XSS	Cross-Site Scripting
SQL	Structured Query Language
CSRF	Cross Site Request Forgery
SSL	Secure Sockets Layer
XML	Extensible Markup Language
CSS	Cascading Style Sheet
TCP	Transmission Control Protocol
TLS	Transport Layer Security
HTML	Hyper Text Markup Language
DRY	Don’t Repeat Yourself

API	Application Programming Interface
MVC	Model View Controller
IFTTT	If This Then That
SE	Smart Energy
IDS	Intrusion Detection Systems
NBD	Non-BEMOSS Device
MAC	Media Access Control
RGB	Red Green Blue
HEX	Hexadecimal
SaaS	Software as a Service
OLTP	Online Transaction Processing
DoS	Denial of Service
DDoS	Distributed Denial of Service
OWASP	Open Web Application Security Project
ACL	Access Control List
CAPTCHA	Completely Automated Public Turing test to tell Computers and Humans Apart
PKC	Public Key Certificate
TPM	Trusted Platform Module
USB	Universal Serial Bus
Nmap	Network Mapper
ORM	Object Relational Mapping
GCM	Galois/Counter Mode
CA	Certificate Authority
AES	Advanced Encryption Standard
IPC	Inter-Process Communication
PUB SUB	Publish Subscribe
CPU	Central Processing Unit
LAN	Local Area Network

1. Introduction

1.1 Background

Commercial buildings consume over 40% of the total energy consumption in the United States [1]. A significant portion of the energy consumed in these buildings is wasted due to inadequate building controls and automation systems.

More than 90% of the commercial buildings in the United States are small-sized (less than 5,000 sqft.) or medium-sized (between 5,000 and 50,000 sqft.).These buildings often do not have a Building Energy Management (BEM) systems. The reasons behind this are – lack of awareness, unavailability of inexpensive packaged solutions, and disincentive to invest in a BEM system if the tenant is not the owner. Energy is primarily consumed for in these buildings for heating, followed by lighting, plug loads and cooling.

The major requirements of a BEM system for small- and medium-sized buildings are:

- Interoperability
- Ease of deployment
- Scalability
- Open architecture
- Plug and play capabilities, and
- Local and remote monitoring capabilities.

Although many systems exist that satisfy one or more of the above requirements, none are available as turnkey packaged solutions. Commercial automation systems often use proprietary architectures which require specific control equipment from specialized vendors, leaving customers with no flexibility to choose control equipment.

With the advent of open source tools and communication technologies, owners and consumers have a choice given these protocols support connectivity with the Internet. Several open communication technologies are now currently available and vendors are also opening up their interfaces to consumers.

An open source BEM system that meets the major requirements for small- and medium-sized buildings will attract many consumers to adopt the systems.

For realizing the requirements of such open source BEM systems, it is important that end users be able to connect and communicate seamlessly with all hardware components prevalent in small- and medium-sized buildings, including stand-alone input devices such as utility power meter, temperature sensors, occupancy sensors and others. Communications with these end units should be possible from a graphical end user interface that is available both locally and remotely from

smart devices. The end-user graphical interface for the above needs should have an open architecture and intuitive navigation.

Although many BEM systems have surfaced in the past few years, they lack consistent and intuitive navigation, secure implementation that can prevent malicious intrusion into the systems and other system and network level security guidance as the application is installed into the building for control operations. Even commercial systems are not secure enough as understood from the recent attacks on a few commercial widely used BEM systems. An intuitive web based user platform that is multi-user facing and can allow monitoring and control of devices in a building are not highly available today. Commercial and open source systems that are available today require training before use, and require an expert to set up the automation application in the building.

1.2 Objective and Scope of the Thesis

The objective of this thesis is to propose, design and implement an open source secure web based user platform to monitor, control, schedule, and perform functions needed for a BEM system serving small and medium-sized commercial buildings. The prototype implementation is part of a larger platform, the Building Energy Management Open Source Software (BEMOSS).

The proposed design incorporates the principles of intuitive graphical user interface design and key components of a secure implementation for such an interface. A graphical interface for a BEM system differs from a traditional application interface in its ease of use, navigation, and consistency of control across multiple devices.

To achieve the objective, monitor and control solutions for load controllers and sensors (including thermostats, VAVs, RTUs, lighting and plug load controllers as well as sensors and power meters) are abstracted and their decentralized control strategies are proposed and implemented.

A security framework for the web-based automation systems is developed and implemented in the prototype.

The following tasks are performed as part of this thesis:

i) Develop a web-based open source, robust and scalable user platform accessible locally and remotely using a web browser from computers, smart phones, laptops, tablets, etc. to monitor and control devices and perform other building operations like scheduling, alarms, and notifications for a BEM system serving small and medium-sized buildings. The proposed user platform has been designed to have the following characteristics:

   - Control system setup – this tool enables the user to make various inputs to control three major loads in buildings (i.e., HVAC, lighting and plug loads) including schedules and set points.
   - System status display – this tool provides dynamic display of the status of assessable loads and sensors throughout the building.
   - Report generation – this tool generates dynamic user reports as required by the user.

- Scheduling – this tool provides an interactive interface for the administrators to schedule device operations using the rich graphical elements. Different types of schedulers are implemented including workdays and holidays.

ii) Develop a comprehensive meta database to store system related information and an Object Relational Mapper that allows efficient and secure communication with the database using an abstract data model. Subtasks include definition of data models for:
   - User management
   - Alarms and notifications management
   - Session and administrator log management
   - Device management

iii) Model threats and vulnerabilities, and implement a security framework for the web platform. The sub tasks include:
   - Analyze the current system model for security
   - Develop a threat model for the BEM system and enumerate possible threats and vulnerabilities in the application
   - Design the system with a 'security first' objective thus implementing security along with the system design.
   - Implement the security framework to mitigate these vulnerabilities

1.3 Contributions

1.3.1 User Platform Design Principles Identification

This thesis identifies the principles of graphical user interface design for BEM systems in small- and medium-sized commercial buildings. The cost effectiveness of such a system has been factored in the design requirements of a lightweight and robust interface that needs little or no training and minimum setup for end-user deployment. Specific user platform principles to minimize costs and maximize performance have been identified and discussed. User management strategies for developing a multi-user platform to allow role-based access have also been identified. The contribution in this part is the BEM system-specific user platform design to cater to small- and medium-sized building needs using the rich graphical interface tools.

1.3.2 Prototype Implementation of the User Platform

The work conducted as part of this thesis also includes the development of a prototype that implements the proposed design principles. This prototype is subject to a wide range of use cases and tests to assess the intuitiveness and user-friendliness of the interface. The contribution in this part is the demonstration of the effectiveness of the user platform design that is proposed above in Section 1.3.1. It also includes the comprehensive architecture that accommodates several existing open source technologies and a medium to integrate future open source technologies as desired.

1.3.3 Data Storage Models

This thesis identifies various data storage requirements in a BEM system for small- and medium-sized commercial buildings. The data models accommodate a flexible, dynamic, and expanding data platform for the open source BEM system. The contribution in this area is the development of a data storage model that accommodates dynamic device types in an automation system while continually providing tight integration with the system. The data storage model is based on the Online Transaction Processing (OLTP) that is separate from the time-series data store.

1.3.4 Security Framework for a Web-Based BEM System

This thesis analyses the security of the prototype implementation and formulates a threat model that can serve as the base case for similar web-based BEM systems. The threat model identifies common threats and vulnerabilities in a BEM system in all levels of software implementation. Mitigation strategies are proposed for common vulnerabilities and implemented in the prototype to demonstrate its effectiveness. The contributions in this area include security vulnerability, threat identification, and mitigation strategies utilizing open source tools and algorithms.

2. Literature Review

This chapter summarizes the literature search categorized into the following sections: overview of BEM systems, state-of-the-art, and user feedback with web technologies, dynamic visualizations, user interface design, data management, and security requirements. Each section covers the current literature relevant to this thesis.

2.1 Building Energy Management Systems

Commercial BEM systems have evolved continuously over the past 50 years to a point now where they can be completely controlled from a central location – thus being referred to as 'smart buildings'. These buildings have relied on proven and robust technologies like BACnet [2]. Most of these smart buildings are huge (over 50000 square feet of building space). Such huge buildings/business can afford to monitor and control the building energy usage using the commercial 'smart building' technologies. Such commercial smart building technologies are quite mature and have been in use for past few decades in most large-sized commercial buildings. The BEM systems that are commercially available enable energy savings and cost reduction in all of these large commercial buildings.

2.1.1 Commercial BEM Systems

Siemens has a commercial BEM system available for all building types, sizes and for every use. This system uses a wide variety of communication standards and interfaces that allows integration of a diverse set of products in disciplines like ventilation, heating, air conditioning, lights and blinds and other energy saving product technologies. A centralized energy management option is provided to achieve cost and energy savings. The European product is called Desigo [3] while their solution for small and medium sized commercial and multi-purpose buildings is called Synco [4]. For the Asia/Pacific and America, the BEM system from Siemens is termed APOGEE [5] – all of them offering a clean single-point solution for energy savings and cost reduction.

Desigo allows checking and operating the BEM system from a central location, has modular, user friendly software, makes operation easy and intuitive, and prevents errors. The system has fully integrated processing of historic and real-time data and generation of regular reports. A central alarm management displayed on the management station is integrated into the system. Desigo provides user specific displays that are tailored to the specific access rights, layout, and type of information displayed to the requirements and responsibilities of individual users and remote access over the Internet.

Synco is a HVAC automation tool targeting small and medium sized commercial buildings developed by Siemens. Their controllers offer maximum energy efficiency and reliability in a modular design. The product is highlighted by Siemens as well prepared for any requirement with a comprehensive product range that provides remote plant access, with reliability, versatility, and expandability for future use.

Honeywell Building Controls [6] offers a variety of their products integrated into one BEM system framework called Tridium Niagara. It integrates all product control onto a single platform, and individual room control based on specific schedules. It optimizes energy savings utilizing existing components wherever possible.

Johnson Controls developed the Metasys [7] BEM system, which is been adopted by many different commercial buildings. It provides essential instrumentation and control, which saves energy, lowers operational costs, and enables productive secure environments. Metasys connects HVAC, lighting, security and protection systems, connects these devices together using the software platform. The user interface presents information in a simple, intuitive manner to identify potential problems. Their system allows quick filtering to navigate to the information that the user is looking for. It is also mobile-optimized and allows remote access using the Internet.

The above-mentioned BEM systems are some of the best available commercial products in the industry. These buildings usually have a dedicated building operations team/manager available to monitor and control building operations using these products.

2.1.2 Building Size Distribution in the US

These products are mostly not affordable by small and medium sized buildings today in the US. Buildings consume over 40% of the total energy consumption in the US. According to U.S. Energy Information Administration (EIA), vast majorities of commercial buildings are relatively small [8]. As shown in Figure 2-1, about half (49.9%) of all commercial buildings are 5,000 square feet or smaller in size. Medium-sized commercial buildings (between 5,001 to 50,000 square feet) constitute almost the other half (44.0%). These buildings typically do not use BEM systems to monitor and control their buildings from a central location. This is primarily due to the unavailability of low-cost and simple BEM system that can function without the presence of a building engineer on-site.

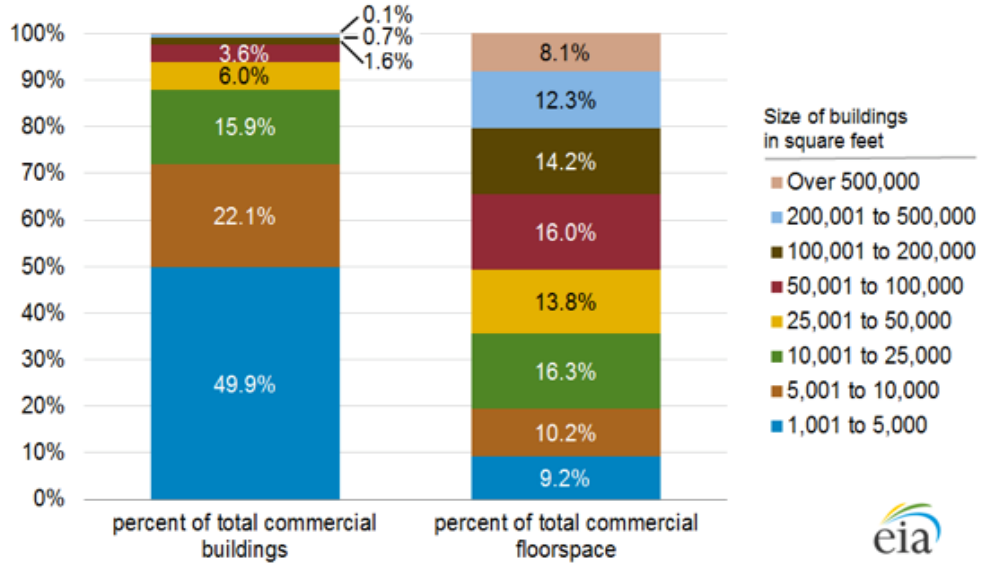


Figure 2-1 Percent of total commercial buildings and floor space by building size [2013, Public Domain]
Source: http://www.eia.gov/consumption/commercial/reports/2012/buildstock/images/fig2_bg.png

2.1.3 Open Source Solutions to BEM System

To provide the small and medium sized buildings with low cost/open source solutions to building energy management, several solutions are developed and available today. When it comes to open source, it is not only the building technologies but also device technology. Several devices today have openly available APIs to connect to them using several different software platforms.

Project Haystack [9] is an open source initiative to streamline working with data from the Internet of Things. It standardizes semantic data models and web services with the goal of making it easier to unlock value from the vast quantity of data being generated from smart devices permeating homes, buildings, cities, etc. Applications developed using Haystack include automation, control, HVAC, lighting, etc.

Honeywell acquired Tridium [10] but let it continue open sourcing its software. Tridium's Niagara is a Java software framework and infrastructure that focus on building tools and applications for BEM systems. Tridium leverages the Java Virtual Machine to run its application.

Freedomotic [11] is a multi-platform open source flexible secure Internet of Things development framework to build and manage modern smart spaces. It targets private individuals for home automation as well as business users (small retail environments, monitoring and analytics, etc.). It can be run on embedded devices with low processing power like the Raspberry Pi.

Open Remote [12] is a software integration platform for residential and commercial building energy management. OpenRemote platform is protocol agnostic and operates off the shelf hardware. Like Freedomotic, it can be run on multiple platforms. It provides a system of controls that can be designed using their integrated development platform.

SmartThings [13] from Samsung provides an open platform for smart homes and the Internet of Things consumers. SmartThings enables consumers to connect, manage, and monitor their homes via their smartphones using SmartThings mobile application. It connects using multiple communication technologies and can add hundreds of connected devices to its platform. They provide a hub to which smart things can be connected and controlled from.

Volttron [14] is an open source agent-based platform developed by Pacific Northwest National Labs (PNNL) to provide distributed control and sensing software platform. It is a platform to deploy intelligent agents that can utilize its resources to do decentralized cooperative decision making, enabling customers, building owners, utilities, etc. to realize better energy efficiency and reliability.

The above open source technologies are available today for developers to use them and build tools of their choice. These solutions are not readily available for the small and medium-sized building owners/building operators to use in their building.

A turn-key application that is completely open source, ready to be deployed, with plug and play capabilities integrating several devices with multiple open source communication technologies and APIs together into one single platform is still unavailable today. A building operator should be

able to download the software, buy smart devices, and install them without much effort or the need of a support engineer onsite/online.

2.2 Components in a Small- and Medium-Sized BEM system

A study by PNNL [15] showed that although there are minor differences, heating consumption is the dominant end use, followed by lighting, plug loads and cooling. Such an energy management system for small and medium-sized buildings should have API that allow interconnection of different device types from different vendors.

A self-sustaining platform should have an intelligent platform for managing and controlling devices automatically using algorithms as well as through user input. Such a platform should have strong communication capabilities to interact amongst each other and with devices. Volttron is a good example of an intelligent core platform that can act as a perfect middleware tool in a BEM system. Volttron is a modular system to add all user agents that can communicate intelligently using a common message exchange bus.

A robust, intuitive, role-based user platform that can let users navigate through, monitor and control different devices in the building from anywhere including a smartphone, tablet, or laptops with Internet. Most of the open source platforms have some form of a user platform that is currently available but not a complete system that incorporates all needs of a small- and medium-sized building operation. A small and medium-sized building can be a single tenant building or multi-tenant. If the building is multi-tenant, it should allow controlled access to the BEM system.

A time-series data store to allow storage of time-series data collected at frequent intervals from devices is an important requirement in a BEM system to allow data analysis and improve the control systems. A metadata store is also necessary to keep track of devices currently available in the building, and manage their general properties. Users can also be managed using such metadata stores.

2.3 Human Machine Interface (HMI) Design

The human machine interface is becoming a primary selling point in the commercial systems with the addition of technologies that allow remote communication with the BEM system, and real time interfacing. The user interface for a BEM system is the ultimate tool that will be used by all building tenants (single/multi-tenant). An intuitive navigation interface becomes one of the essential components of efficient interfacing.

The HMI in Johnson Control's Metasys allows real time navigation to access devices. It allows quick filtering of data to see information that is important to the end user. Metasys is mobile-optimized and can be managed from anywhere, on any device. Johnson Controls states 'Training time is also reduced, since you don't have to learn a different way to use the system for each separate device. The result is improved productivity'. While reducing training time might be a useful feature, it still requires the end users to pay the exorbitant price for their product. With open source technologies, in addition to completely eliminating training with a simple to understand user guide and intuitive navigation, a customizable and user-friendly HMI will be of great benefit.

Freedomotic, [11] which is open source provides the building map onto which devices are added and can be viewed or controlled from that location. This is also web based and allows monitoring and control remotely. Freedomotic uses the open source ActiveMQ [16] for its message exchanging requirements.

The HMI for a BEM system that is open source should use existing technologies and build on top of it. Connecting to existing platforms for automation using their common message exchange bus is a fundamental requirement to allow the HMI to meet with the control platform in the server. This requires good connectivity capabilities with the existing control solutions available in the open source space.

2.4 User Interface Design Principles

Design principles define usability of a product and are fairly constant based on cognition and human behavior. Some of the usability principles are discussed in this section.

- Principle of Proximity: This is one of many principles defined in the Gestalt principles of perception [17]. The principle states that we perceive relationships between objects that are closer together.

- Hick's Law discussed in [18] helps calculate the time it takes for user to make a decision as a result of the number of choices they have. It is also known as reaction time. Having a complex navigation pattern is an example of this scenario.

- Fitt's Law: Fitt's Law discussed in [19] determines of the size of elements such as buttons, menus, and text in an application within a page based on the distance a user pointing device must travel. This distance determines the location of the popups and modals in addition to regular positioning of the GUI elements.

The following design principles are discussed in [20] to ensure good and easy-to-use user interface design.

- Visibility, Visual Feedback and Prominence: Visibility is anything that brings visual focus to an element or action in the graphical user interface. This can be accomplished by Typeface (different styles and sizes of text that can draw a user's attention), opacity (adjusting an item's opaqueness helps reduce or enhance its visibility), prominence (those elements larger than others will be more visible), status (indicating the processing of a request or when a user has received a new message, etc.) and color/contrast (items with varied colors and contrasts drive more attention).
- Hierarchy: In applications with complex functionalities, organizing the layout that still shows all the features that the user has might be difficult. This can be made better using the hierarchy principle that should provide visual indicators of the structure of the application.

- Progressive Disclosure: Hiding options that are not possible can reduce the users' cognitive load and guide them more effectively through their tasks in the application. This is useful in BEM systems which has different devices with many different features and functionalities.

- Consistency: Developing a consistent layout and interface for users will ease the learning process. It is also helpful to use the most commonly used GUI tools for every possible requirement.

- Confirmation: Whenever an important event occurs in the application, it is imperative to verify that the user wants this operation to occur. This is especially required when the user is trying to delete some information which cannot be retrieved.

2.5 Web Technologies

Several open source web-based solutions are available today, which allow integration of other open source tools in order to build a complex system. For a BEM system, it needs to be modular and quick for setup and use. Inviting developers to contribute in an open source platform needs developers to be available in the technology that the development occurs in. Several programming languages that provide web-based frameworks and technologies are available although only a few of them are suited for a BEM system.

2.5.1 Languages

Java [21] is a concurrent, class-based programming language that is object-oriented. It is intended to let application developers write code once and run it everywhere. J2EE is combination of web technologies developed using the Java programming language. Being open source, it can incorporate several new web frameworks into building a user platform for BAS.

Python [22] is a widely used general purpose programming language that emphasizes code readability and allows developers to express the concept using fewer lines of code. It is widely used in open source applications. It is a multi-paradigm programming language with dynamic typing which makes it easier for developers to program the concept. Several applications and platforms are built using Python today - one such example is Volttron [14]. A large number of open source libraries and frameworks built using Python with a good developer support make it a language of choice for many open source programmers.

2.5.2 Web Frameworks

Web frameworks are plenty in Python with a notable few being widely used. For an open source BEM system, it is important that the most flexible web framework with powerful features and a good developer set is available to enable continuous support for the libraries and active developer engagement. Django [23] is one such web framework with a variety of features that comes with it. The platform is already modular if it uses Django. Clean separation of middleware from code enablers allow new developers to add applications to a platform without having to deal with the details of how the process occurs in the background.

Flask [24] is a python micro-framework to create custom tools and utilities for any web application. It does not support a database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions.

2.5.3 Messaging/Web Service Technologies

Representational State Transfer (REST) [25] is a software architecture style to create scalable web services. It is a stateless protocol and also cacheable. The core platform and the user platform can create web services using REST technology that can be consumed by the other. RESTful web services allow session based authentication and consumption by its clients. Considering that the web application and the core platform rest in the same core, RESTful web services may serve as an efficient option. With web services, if a server fails, the client must take responsibility to handle the error. When the server is working, again the client is responsible for resending it. If the server gives a response to the call and the client fails the operation is lost. If millions of clients call a web service on one server in a single second, the load handling might not be feasible.

Message queues are components used for inter-process/over the web communication or for inter-thread communication within the same process. A queue is used for messaging. Many different architecture patterns like publish-subscribe, client-server, request-reply, parallel pipeline, and others are available in different open source message queue platforms. The most popular ones for python are ZeroMQ [26], RabbitMQ [27], and Celery [28]. A message queue is more fault tolerant as compared to the web service technology. If the server fails, the queue persists the message (sometimes, even if the server is powered off). When the server is working again, it receives the pending message. If the server gives a response to the call and the client fails, if the client did not acknowledge the response the message is persisted. Since a socket connection is established, the application has a fair idea of the number of the message requests or responses that will pass through the socket, and therefore the developer can anticipate and decide the number of requests that will pass through and handle contention.

Volttron platform discussed in [14] chosen for the core platform uses ZeroMQ for inter-agent communication. It would eliminate the need for developing new web services to connect with the user platform and reduce the load by reusing existing technology in the platform than to rebuild something and consume more memory. Operating on low power embedded devices requires that memory usage is kept to a minimum. Adding another communication layer in terms of web services is therefore eliminated. ZeroMQ is inbuilt into Volttron and eliminates the need for a new setup while developing a user platform. Volttron can be used as a dependency for the user platform and all messaging APIs can be reused. This also reduces the need for increased developer time.

2.5.4 Front-End Frameworks

A web-based platform should have a mature and flexible front end framework to create dynamic web pages. Several front end frameworks are gaining importance in the recent years. At the start of this project, only a few front end frameworks were mature enough to be utilized for a full-fledged web front end.

Bootstrap [29] is a front end HTML, CSS, and JS framework for developing responsive, mobile first projects. The framework size is about 145 KB which is quite huge considering that most of this will be loaded into the web pages every time a page loads. It was released in 2011 and quite mature with a large user community and quick support. More features can be added to the framework using Less and Sass preprocessors. It is compatible with all the major browsers like

Firefox, Chrome, Safari, and IE8+. Although emergent technologies like Pure.css are taking preference today with its tiny file size, the prototyping started in early 2013 at which point, these new CSS frameworks does not exist. Bootstrap is the standard for most websites today and a typical administrator template is easy to develop and expand using this framework.

2.6 Web Application Protocols

In [30] web pages are referred to as online documents that can be accessed using a unique document address, the Universal Resource Locator (URL). Web browsers access single pages using the unique URLs which were then combined to make a simple website which is a collection of related websites with a common URL. These websites usually shared static information that never changed until the page itself was modified. The last 15 years has seen enormous growth of web applications which are dynamic and can show content specific to a country, user, platform, time of the day, month, etc.

The three basic components that form the essence of web technology are:

- A markup language to format hypertext documents
- A uniform notation scheme for addressing accessible resources over the network
- A protocol for transporting messages over the network

Designing a web application involves multiple core technologies and protocols such as HTTP (Hyper Text Transfer Protocol) and HTML (Hyper Text Markup Language) which are fundamental to the creation and transmission of web pages. The URL has the following generalized notation associated with it:

scheme://host[:port#]/path/…/[;url-params][?query-string][#anchor]

scheme: designated the underlying protocol to be used (e.g., 'http' or 'ws').
host: is either the name or the IP address for the web server from where the application is being served.
port#: is an optional portion of the URL designating the port number that the target web server listens to.
path: is the file system path from the 'root' directory of the web application in the server for the desired document. However, this path may vary based on the application semantics and the configuration available for the application.
url-params: this is a rarely used portion and used mostly for session identifiers in web servers supporting the Java Servlet API. It is followed by a semi-colon to specify the end.
query-string: this is an optional portion of the URL that contains dynamic user-entered variables sent to the server using the HTML forms.
anchor: is a reference to a positional marker within the requested document, like a bookmark.

The HTTP Web Protocol: This is the foundation protocol of the World Wide Web. It uses a request/response paradigm, meaning that an HTTP client program sends a HTTP request message to a HTTP server, which returns a HTTP response message. The structure of request and response messages are similar to that of email messages and consist of message headers, followed by a blank line, followed by a message body. It is a stateless protocol meaning that it has no explicit support

for the notion of a state. An HTTP transaction consists of a single request from a client to the server, followed by a single response from the server back to the client.

The Web Socket Protocol: As stated in [31,32], web socket enables two-way communication between a client running untrusted code in a controlled environment to a remote host that has opted-in to communications from that code. An origin-checker security model is used by web browsers to enable security. The protocol consists of an opening handshake followed by basic message framing, layered over TCP, to establish two-way communication with servers that do not reply on opening multiple HTTP connections like *XMLHTTPRequest*, iframe, or long polling the server. There are two URI schemes for using web sockets.

ws-URI = "ws:" "//" host [":" port] path ["?" query]
wss-URI = "wss:" "//" host [":" port] path ["?" query]

While the definitions are similar to the URL schema discussed earlier, *ws* are used for default communication on port 80, while *wss* is the secure counterpart of the *ws* protocol.

In [33], the authors compare latency in web sockets with HTTP long polling. When the packets must travel a long distance, or via a congested network, resulting in a network underlying latency that exceeds half the data measurement rate, then the Web socket protocol is a better choice. The stateful approach that the Web socket protocol uses does provide better latency (on average) for real-time Internet communication.

In [34], the authors suggest that Web Sockets provides an enormous step forward in the scalability of the real-time web. As you have seen in this article, HTML5 Web Sockets can provide a 500:1 or—depending on the size of the HTTP headers—even a 1000:1 reduction in unnecessary HTTP header traffic and 3:1 reduction in latency.

In [35] the authors show that a Web sockets server consumes 50% less network bandwidth than an AJAX server; a Web sockets client consumes memory at constant rate, not at an increasing rate; and Web sockets can send up to 215.44% more data samples when consuming the same amount network bandwidth as AJAX.

HTTP Request Types: The two most used HTTP methods are GET and POST.

- GET: Requests data from a specified resource
- POST: Submits data to be processed to a specified resource

GET: GET requests can be cached; they remain in the browser history; it can be bookmarked; they should never be used when dealing with sensitive data; they have length restrictions; should be used only to retrieve data.

/test/demo_form.asp?name1=value1&name2=value2

POST: Query strings are sent in the body of the POST request. POST requests are never cached and they do not remain in the browser history. They don't have restrictions on data length.

```
POST /test/demo_form.asp HTTP/1.1
Host: w3schools.com
name1=value1&name2=value2
```

Status Codes – HTTP: The first line of a response is the status line, consisting of the protocol and its version number, followed by a three-digit status code and an explanation of that status code. Status codes are grouped into categories. There are 5 categories defined:

- **1xx:** Classified as *informational.*
- **2xx:** Indicate *successful* responses.
- **3xx:** For purposes of *redirection.*
- **4xx:** Represent *client request errors.*
- **5xx:** Represent *server errors.*

HTTP Session Management: HTTP is a stateless protocol – this requires some form of maintenance of state between HTTP requests. This is established through sessions. HTTP/1.1 establishes this connection through Set-Cookie and Cookie headers. Set-Cookie is a response header sent by the server to the browser, settings attributes that establish the state within the browser. Sessions can be maintained also by using Cookies stored in Databases. This ensures more security than the browser cookies, where cookies can be replayed.

2.7 Web Application Architecture

Architecting a web platform is particularly important in a scenario when there is a fully operational control platform since it involves communicating with an existing platform. The web platform designed for a BEM system should be modular and manageable. Clearly segregating application modules and allowing a modular addition of new components that ensures reusability of older components and enhancing its features using other modules is important for such an application.

Several architectural patterns exist today, out of which the broad architecture design for the entire BEM system should preferably be layered. Layered architecture allows separation of concerns among components. Components within a specific layer deal only with logic that pertains to that layer. While the overall architecture is conceptualized as layered, the web platform architecture needs to be event driven partially.

2.7.1 Event Driven Architecture

This pattern is a distributed asynchronous architecture pattern for producing highly scalable applications. It is highly adaptable and can be used for small applications and as well as large complex ones. The event-driven architecture is made up of highly decoupled, single-purpose event processing components that asynchronously send/receive events. There is a need for event driven architecture in the BEM system that uses Volttron for the control platform, since Volttron uses the ZeroMQ, a messaging socket for communication.

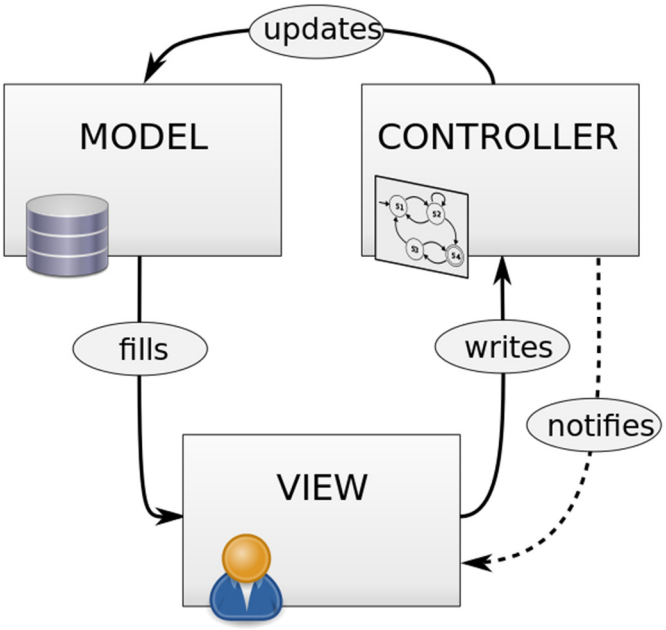


Figure 2-2 Model-View-Controller pattern © [Used under fair use, 2015]
Source: http://commons.wikimedia.org/wiki/File:MVC_Diagram (Model-View-Controller).svg

Although model-view-controller segregates the components of the web application, it does not offer a communication pathway to the messaging socket. A modification of this pattern to include the messaging socket communication and also allow push notifications to the web page would be applicable to the BEM system web platform.

Model-View-Controller [36] is a software architectural pattern associated with user interface implementation. It divides a given software application into three interconnected parts, so as to separate internal representations of information from the ways that information is presented to or accepted from the user. The central component of this pattern is the model that captures the behavior of the application in terms of its problem domain, independent of the graphical user interface. A view is an output of all the information processed. The controller accepts input and converts it to commands for the model/view. A representative diagram of the model-view-controller pattern is shown in Figure 2-2.

2.7.2 Review of Existing BEM System Platforms

Architectures such as the above have been used sparingly in the BEM systems today. Authors in [37] discuss a framework for designing decentralized control using object orientation. Their approach follows the physical architecture of a building as a starting point for their automation software.

The authors in [38] use a layered architecture using web based technologies like REST. Their application also discusses device identification function. Their architecture clearly separates every layer and allows communication between different layers using REST protocol. REST is a synchronous protocol and might not be relevant to provide lightweight push updates to the user. Their architecture defines the authentication manager in the center and interconnects all other

gateways to connect to this central manager. The application allows OAuth authentication with Facebook and other OAuth providers to facilitate authentication. However, this cannot be used for a closed group system like the one we intend to design. The RESTful web service API uses a topic based publishing format that uses topics as the Web service URI to be queried for information.

The paper [39] also discusses the use of REST architecture as the primary communication mode between the different control modules in their architecture. The authors suggest the need for stateful communication since REST is stateless.

Authors of [40] discuss a service-oriented architecture for BEM system. Their approach uses the modular Open Services Gateway Interface (OSGi) which is a dynamic Java-based application platform that supports modularization of software by use of a component model and a service registry as well as by using restrictive implementation requirements. The authors clearly segregate the different activities in a BEM system into service categories further dividing each category into smaller control components. Their user interface is also built as a service with different workflows.

In [41] the authors discuss about providing a control and management interface in addition to several other interfaces like network interface, point-to-point interface, process interface, etc. These control interfaces are human machine interfaces that allow customization of various system parameters.

In [42], the authors introduce a modular solution that separates domain knowledge, interaction design, and presentation heuristics into multiple collaborating models. Each model contains knowledge about a particular aspect of interface design, and uses this knowledge to create each user interface that is needed to support the users of a control system dynamically.

Opto 22 in [43] discusses the requirements for a Human Machine Interface that is clean and provides appropriate information rather than data being thrown in the midst of graphics. 3D type data display is confusing according to the white paper, and a simple design in 2D with the related data grouped together.

In [44], the application is divided into several layers, each linked to the other. The style rule layer defines the presentation and behavior of a family of interaction techniques.

2.8 Security Framework in BEM Systems

Several BEM products exist in the market today and most of them are commercial with built-in security. However, recent attacks on BEM systems, like Tridium Niagara Framework [45] hacking into Google's office in Australia, increase concern about the security framework in these systems. In one instance of hacking, hackers gained control of building locks, electricity, elevators, etc. [46] as reported by Wired. In similar systems intended for home security, several vulnerabilities were found [47], as warned by HP. Stuxnet [48] worm was designed to attack industrial programmable logic controllers (PLCs), which demonstrated the wide-ranging havoc that could be caused by malicious software infecting plant controllers. Similar attacks on systems, like a BEM system connected to a smart grid, could shut down the regional electricity grid network.

Security in BEM systems range from device security to platform security to front end security. Front end security for web based systems should be emphasized since that is an easy target for botnets and other malicious software crawling the internet.

The authors of [49] analyze the standard security available in these systems, and suggest methods to make the system more secure. Their approach concentrates on the underlying software security techniques like code signing, software watermarking, proof-carrying code, etc. They also discuss the use of Intrusion Detection Systems (IDS), like signature-based IDS, anomaly-based IDS, software monitoring techniques (SMTs), sandboxing, and other attack specific countermeasures for various attacks in the system. According to the authors, manual inspection and configuration prevents a lot of attacks. Physical partitioning is also an efficient mechanism.

In [50], the authors propose a common approach for the pre-design phase of integrated safety and security systems. Techniques such as the risk analysis common in both areas are synchronized to figure out overall hazards that endanger safety or security of a BEM System.

Authors in [51] discuss a top-down multi-abstraction layer approach for embedded security design that reduces the risk of security flaws, letting designers maximize security while limiting area, energy, and computation costs.

The paper [52] presents a novel, generic approach on using client puzzles to prevent Denial of Service attacks. According to the authors, using this technique, if prevention is not possible, can be detected at least.

SANS Network Security Digest [53] quote: 'Buffer overflows appear to be the most common problems reported in May, with degradation-of-service problems a distant second. Many of the buffer overflow problems are probably the result of careless programming, and could have been found and corrected by the vendors, before releasing the software, if the vendors had performed elementary testing or code reviews along the way.' The authors in [54] discuss StackGuard: a simple compiler technique that virtually eliminates buffer overflow vulnerabilities with only modest performance penalties.

The authors in [55] discuss a static analysis for finding XSS vulnerabilities that directly addresses weak or absent input validation. Cross Site Request Forgery (CSRF) has emerged as a potent threat to Web 2.0 applications. Because of the stateless nature of the HTTP protocol, a malicious website can force the user's browser to send unauthorized requests to a trusted site. [56Reference [51] discusses ways in which some popular web applications are exploited using CSRF. It also demonstrates techniques by which CSRF signatures can be detected and attacks effectively resisted even before initiation.

The authors of [57] enumerate the various SQL Injection attack methods and defenses. The following are the SQL Injection attack avenues according to the authors:

- Cookies
- Server variables
- Second-order injection
- Physical user input

Several prevention mechanisms are discussed:

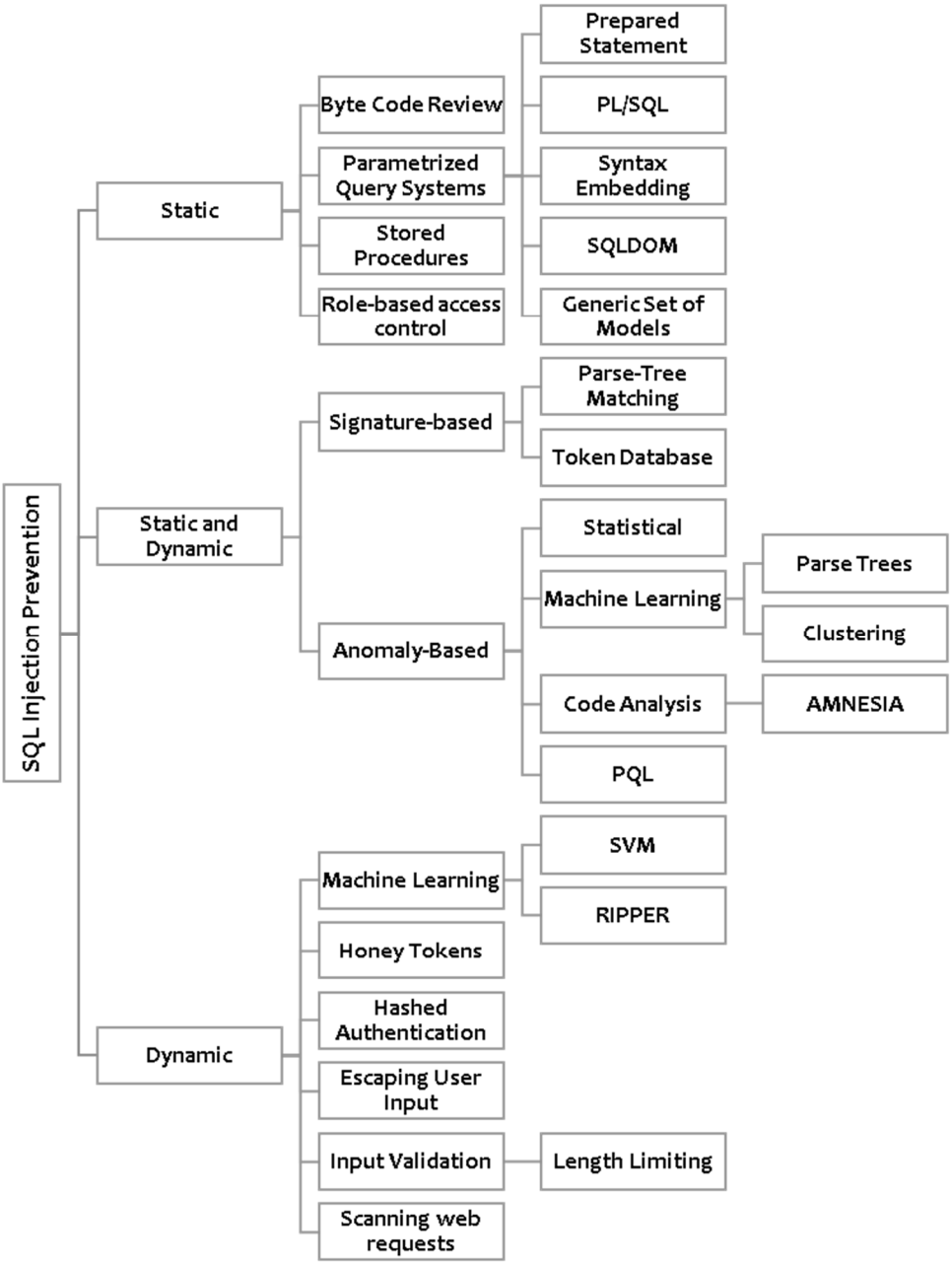


Figure 2-3 SQL injection prevention techniques [57] [Used under fair user, 2015]
Chandrashekhar, R., Mardithaya, M., Thilagam, S. & Saha, D. 2012, "SQL Injection Attack Mechanisms and Prevention Techniques" in Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 524-533.

In [58] the authors discuss a lightweight static analysis method to detect common security vulnerabilities (including buffer overflows and format string vulnerabilities). Authors in [59] discuss the need for integrating security throughout the life cycle of application development. The authors discuss a detailed review of the steps involved in applying security-specific activities

throughout the software development life cycle using effective, efficient application design, development, and testing.

Security threat modeling is important to ensure security is built into the web applications. Authors in [59] investigate ways to extend the computational approach to security assessment in cyber-physical systems, by modeling potential threats to obtain probabilities of state changes in a Markov chain description of security violations.

The security development lifecycle discussed in [60] provides a process for modeling threat while developing applications. It discusses the use of STRIDE method for identifying threats in an application. The method looks at each part of the application and determine whether any threats that fall into the S (Spoofing identity), T (Tampering with data), R (Repudiation), I (Information disclosure), D (Denial of service), or E (Elevation of privilege) categories exist for that component or process.

Authors in [61] have published a lightweight security framework called TinySec to secure the link layer of wireless sensor networks. Implementation of security options for most wireless communication protocols, including IEEE 802.15.4 are very resource consuming. TinySec balances among security, packet overhead, and resource requirements.

The article in [62] discusses using SSL/TLS security for Web socket security similar to HTTPS. Web socket uses the origin security model, as defined by [63]. If the Web socket gateway can be configured for origin-based access control then cross-origin Web socket connections can be made securely.

Curve-based security discussed in [64] provides security for ZeroMQ. CurveZMQ is a protocol for secure messaging across the Internet that closely follows the CurveCP security handshake. This protocol provides security against Eavesdropping, Fraudulent Data, Altering/Replaying data, Amplification attacks, Man-in-the-middle attacks, Key theft attacks, identifying the client attacks and Denial of Service attacks.

2.9 Conclusions and Knowledge Gaps

This chapter summarizes the literature search and identifies the knowledge gaps in three areas: web platform for open source BEM systems, data modeling, and security framework of BEM system.

2.9.1 Web platform for Open Source BEM Systems

Much work has been done on developing control platforms for BAS. However not many open source tools are available today. The open source BEM systems available today are not turn-key and require extensive configuration and in some cases building the interface components after setup of the application in the local environment. Such toolkits are not necessarily helpful for small- and medium-sized businesses that may lack the technical knowledge to set up the BEM system in their buildings.

Some of these applications like Volttron and a few others do not provide a user platform for their control platform. Applications discussed in Section 2.4 do not discuss in detail about the user interface that is available and their ease of use.

Selected publications on BAS/control systems do not discuss the user interface portion of the application in detail. Research in the area of providing an easy-to-use, informative user interface that follows appropriate design principles for producing a rich platform is not exclusive. Most of the automation system discussions are restricted to overall architecture, device communications, and control algorithms. They do not discuss user interface or the streamlining of control mechanisms.

The mechanism used for inter-process communication or inter-layer communication in these automation systems are not detailed in most discussions. Communication methods that use RESTful technologies or message queues or AJAX polling or plain client/server models have not been made clear. In most applications, appropriate use of any of these techniques alone will result in a highly efficient application. Application performance and throughput are important factors to consider for a remotely controlled application and the web platform performance plays a critical role in the overall performance. This calls for laying out methodologies for communicating the control/monitoring information with the user. Web browser updates in real time has also not been discussed in any of the selected prior publications discussed in this chapter. Controlling access to several parts of the application is not discussed as well given the web platform is not elaborated in the discussion.

Web platform modeling is important considering the amount of information that needs to be provided a control access. Role based access control plays an important application in providing feedback and enable a collaborative monitoring/control effort in a modern small- and medium-sized BEM systems. These details have not been discussed in prior publications.

2.9.2 Data Modeling

Designing a good data model is essential for bringing up an efficient web platform and the core platform. A proper data model resolves most of the technical needs of a system. A data model defines the user management and the role management in the BEM systems. Users can be grouped into different roles or they can be allocated to the default role the system provides. These users based on the role will be provided different levels of access within the same page. In other words, dynamic pages can be generated on the basis of the user management data model.

Every device needs to be modeled in the system to enable efficient monitoring and control capabilities that reflect real world physical monitoring and control of devices. For this reason, devices need to be modeled to reflect every control/monitor parameter available into a single model that provides common metadata information about all devices in the system. Such data modeling are not discussed in detail in selected publications described here.

Selected prior publications discussed in this chapter do not discuss the data model used in the application. Control platforms do not discuss how data models are utilized in monitoring/control functionalities.

Selected publications mentioned above also do not discuss the alarms and notifications configuration or enablement in the automation systems. Although several commercial systems allow alarm configuration and notification customization, the open source tools do not discuss in detail about this. Neither do the commercial systems discuss the level of configuration allowed in their systems. A comprehensive model for managing alarm creation, activation, and managing notifications are not discussed in the publications listed in this chapter.

2.9.3 Security

Security in BEM system is not only concerning device security and the communication protocols that devices use, but also the security of the different layers in the system architecture. Communication between layers is also an important factor to consider while ensuring security of the system. Some systems may use RESTful web services and some others might use message queues and others just simple polling. All of these need to be secure and authorized communications.

Selected prior publications do not discuss the security of the different layers in the application. The web platform security is not discussed in depth in these publications. Web platform being the most vulnerable component that is accessible by millions in the world need to be secured from all the web application vulnerabilities.

Deriving a threat model for such an application is not the focus of security related publications discussed in this chapter. Selected publications discussed in this chapter talk about security of device communication protocols. This is only a part of the entire automation system security and needs many more security components integrated into the system from the design phase to ensure security.

For open source tools discussed in select references, there is not much detail about how security is implemented or how it is secure against the cyber-attacks that occur in the open source security frameworks on an everyday basis. The open-source nature of a technology makes it more vulnerable [65] to cyber-attacks like the Heartbleed bug that was found recently in the popular open source tool, OpenSSL that went unnoticed for months.

Security in the communications interface between different layers (control, user platform, etc.) have not been discussed, nor a clear picture of how the communication occurs been discussed. These discussions are critical to the development of BEM systems and to avoid any breach in security in these systems and in the buildings they operate in.

3. Objective

BEM systems currently available in the market today are expensive as a packaged solution making it unaffordable to small and medium sized buildings. This prompts the development of a less expensive, open source BEM system.

Any BEM system should include the following:

1. The core energy management processes.
2. A visual of the current system status and graphical user interface for monitoring and controlling the building operations/devices.
3. A database to store system information including device metadata and application settings.
4. Secure interaction with the system from anywhere (local/remote). Insecure systems are vulnerable to a host of attacks through the application and the network.

The objective of this thesis is as follows:

1. To develop a web-based open source, developer- and user-friendly responsive graphical user interface that is accessible both locally and remotely, from any type of device connected to the web, that allows monitoring, sensing, scheduling and control, and perform functions needed for a BEM system serving small- and medium-sized buildings.
2. To develop a comprehensive data model for the BEM system to abstract device and system information.
3. To analyze the security risks, threats and vulnerabilities and to design, and implement a security framework for the BEM system.

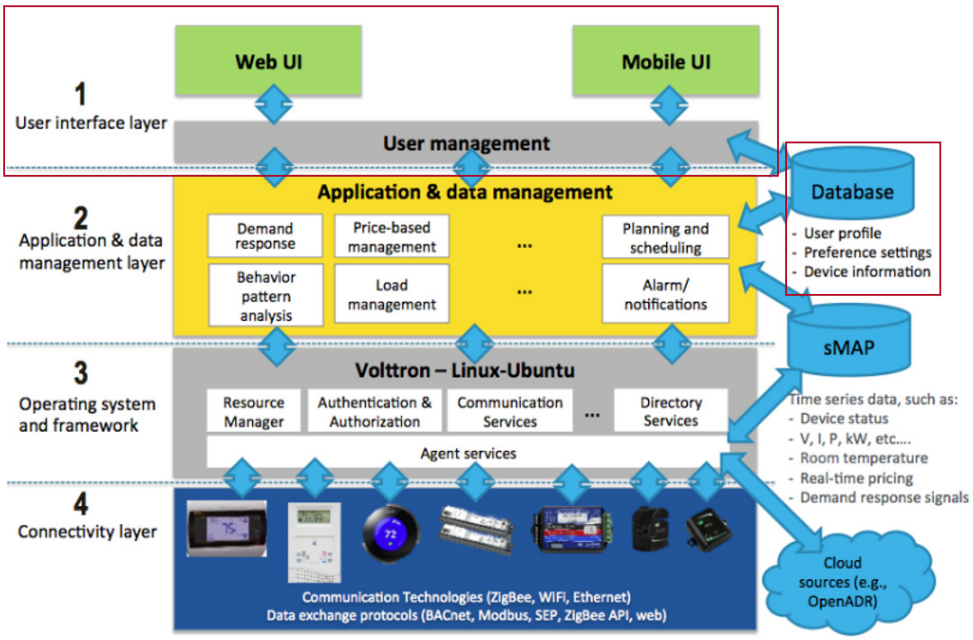


Figure 3-1 BEMOSS architecture: red boxes indicates the objective of this thesis

3.1 Remote Controllable Responsive User Platform Design

Objective: To develop a web-based open source, developer- and user-friendly responsive graphical user interface that is accessible both locally and remotely, from any type of device connected to the web, that allows monitoring, sensing, scheduling and control, and perform functions needed for a BEM system serving small- and medium-sized buildings.

The developed UI has the following key features:

- Scalability: The user interface is built to be highly scalable that can cater to commercial small and medium- sized BEM system. The user interface is built to be responsive and provides a seamless experience from devices of any size and type through the web browser. The UI dynamically adapts to user inputs, roles, and system status.

- Role-based access control: The high-resolution graphical interface is tailored to the roles of individual users and facilities; it is designed to accommodate both novice and experienced users. It is built to cater to multiple user roles including tenant, zone manager, and administrator. These roles can be allocated dynamically and modified as per the needs of the building. The user interface (UI) components are chosen from standardized modern components to allow dynamic views of current system status and control.

- User management: The system has a unique user management system, consisting of tenant, admin, and zone manager roles. Administrator level access is provided to approve device registration, and zone allocation process. The administrator also approves and registers new

user to the system, and allocates roles to them. The administrator can modify the user role at any time using the guided graphical interface.

- Dynamic views: The user interface provides a dynamic view of the status of all devices connected to the system. Whenever a new device is discovered, the messaging protocol immediately makes it visible in the user interface.

- Open source: Given the need for an open source system, the user interface uses only open source components. The user interface is designed to allow future developers to jump in and quickly develop more applications without spending much time analyzing previously written code. The modularity of the source code allows for quick understanding, debugging, and development of the user interface and integrates it with the core system.

- Modular design: Both the source code and the user interface are designed as modular components. This will allow seamless access to the different device/zone dashboards in the graphical user interface and quick development and deployment of modules in the source code level where new applications can be developed.

3.2 Comprehensive Data Model Design

Objective: To develop a comprehensive data model for the BEM system to abstract device and system information.

The following tasks are performed to achieve this objective.

- Implement a user management model: A user management system based on access control lists is implemented. This allows dynamic customization of web pages based on user roles and feature restriction. A central database for both the core system and the user interface is developed. The database is normalized to store metadata of devices connected to BEMOSS, and operating characteristics. An optimized naming scheme is provided for the unique device id based on device type and MAC address.

- Develop a device model: A central devices table holds the details of all devices connected to the system. Normalization based on device type allows adding new smart devices and remain connected to the central devices table. A careful choice of device type identifier enables easy use of metadata in the user interface layer and the core platform.

- Develop a session model: In addition to maintaining smart device information in the database, the user interface also stores user session information. This allows timeout to be automated on session expiry. The database is tightly integrated with the web framework models allowing quick insert/update/delete operations. The database is connected to the core system and nodes (embedded devices) by hosting it in a central server. Replication of the database allows for high availability.

- Implement password storage in the database is in the form of hash, based on NIST standards. Data format is standardized to prevent data duplication. Clear naming conventions allow quick access to the tables from the application making it easier for developers to access database.

- Develop a scheduling model: Apart from using a relational database management system for storing metadata, some applications like scheduling require a dynamically growing data. A Java Script Object Notation (JSON) data format is therefore used, instead of traditional data tables. The JSON format can capture device schedule changes quickly and communicate it to the core system.

3.3 BEM Security Framework

Objective: To analyze the system for security risks and to design and implement a security framework for the BEM system at the application layer.

The following tasks are performed to achieve this objective.

- Analyze a threat model: The security risks are analyzed at the platform level (user platform, control platform and applications layer) and a threat model is realized. The threat model provides an overview of the system use cases, the possible security risks, threats from various sources and some generic way to eliminate the threats.

- Identify BEM system security goals: The threat model is based on the security goals identified for a BEM system. The system security goals are confidentiality, availability, and integrity. Privacy is another major requirement in BEM systems especially when Demand Response operations are performed using BEM systems. All of these goals are achieved by using appropriate security measures.

- Implement authentication: Certificate based client authentication is implemented to secure communications over the web. User authentication and authorization for BEM systems is implemented using the BEMOSS web framework. SHA256 hash with a reasonable number of iterations is used for password storage.

- Implement device approval process: Device approval process inbuilt in the administrator console averts any fake/unauthorized device addition during the plug and play discovery process. The core system invokes an agent corresponding to a device only after administrator approval. The approval process follows triggering device identification from the web interface, followed by manual approval.

- Develop a user management system: A comprehensive role-based user management system is implemented in a BEM system for three roles – administrator, zone manager, and tenant. These roles also function as groups that can be assigned to each user. Group- and permission-based authorization system is implemented. Every request to the server is authenticated before processing.

- Secure Information Exchange: Using a message exchange bus between different nodes connected to BEMOSS enables a multi-node system in BEMOSS. The nodes communicate using the TCP protocol in the message bus. Authorizing all TCP-based communications on the message bus and encrypting the data that is being transferred control unencrypted and unauthorized traffic on this message bus. An agent authorization process is also implemented that verifies if the agent is registered in the system.

- Implement web platform security: Web application security is one of the primary areas to be considered for enabling BEM system platform security. Web platform security is analyzed and security measures have been implemented at every possible point in the system. The Linux platform security is also discussed and guidelines for hardening the Linux platform security have been presented.

4. Architectural Overview of the BEM System

This thesis is part of a much bigger research project – building an open source building energy management platform, BEMOSS. This chapter discusses the architecture and implementation technologies of this open source BEM system implementation – BEMOSS. The chapter is organized as follows. Section 4.1 discusses the system architecture; Section 4.2 discusses the software architecture; and Section 4.3 discusses the user platform architecture, followed by an overview of the user platform and implementation.

4.1 System Architecture

In the BEMOSS architecture for a small commercial building with a few loads, a single board computer (e.g., Beagleboard) with BEMOSS installed can be used to monitor and control of all load controllers in the buildings. This embedded system can communicate with different types of load controllers, i.e., HVAC controllers, lighting load controllers, plug load controllers, sensors/power meters via wireless signals. The UI platform also runs on this embedded device.

For buildings with larger number of devices, the BEMOSS operating system can be set up to deploy its multi-layer architecture feature. In this architecture, several BEMOSS nodes[1] communicate with each other and also communicate with a BEMOSS core[2]. Each BEMOSS node is responsible for monitoring and controlling a zone in which a selected set of load controllers reside, while the BEMOSS core is responsible for supervising the overall system operation and allow local and remote access for monitoring and control of all devices in buildings. The BEMOSS core alone hosts the user platform, and the end user can access the UI using a smartphone, tablet, or a PC. The OS platform residing in the BEMOSS core relays information about the other nodes and their devices to the user platform.

For even larger buildings, a number of BEMOSS nodes can be deployed on different floors/zones in the building depending on the number of zones and devices to be monitored and controlled. In this case, the user platform is again available on the BEMOSS core, and the information from the nodes is relayed to the user platform using the Information Exchange Bus and the Relational Database.

BEMOSS also supports BACnet [2] and Modbus [66] devices, such as RTU and VAV controllers, lighting and plug load controllers, and power meters. For these devices to communicate with BEMOSS, their signals are converted to wireless signals. The architecture is explained in detail in the [67]. The system architecture defines the overall architecture of the BEMOSS system including the other entities in the application like OS layer and API layer. Further discussion on system architecture is beyond the scope of this thesis.

[1] **BEMOSS Node** is the BEMOSS operating system with limited functionalities. It supports connections with hardware devices, hosts selected agents, can run limited applications/algorithms and supports minimal database. It does not support user interface.

[2] **BEMOSS Core** is a full version of the BEMOSS operating system that supports user interface.

4.2 Software Architecture

BEMOSS is designed as a robust open source BEM system that is built completely using open source software tools. The entire BEMOSS system comprises of four layers: User Interface (UI) layer, Application and Data Management Layer, Operating System and Framework layer, and the Connectivity Layer. In addition to these, there are also parallel BEMOSS databases that help with the storage of all information pertaining to BEMOSS and help smooth its functioning. Figure 4-1 shows the overall system architecture.

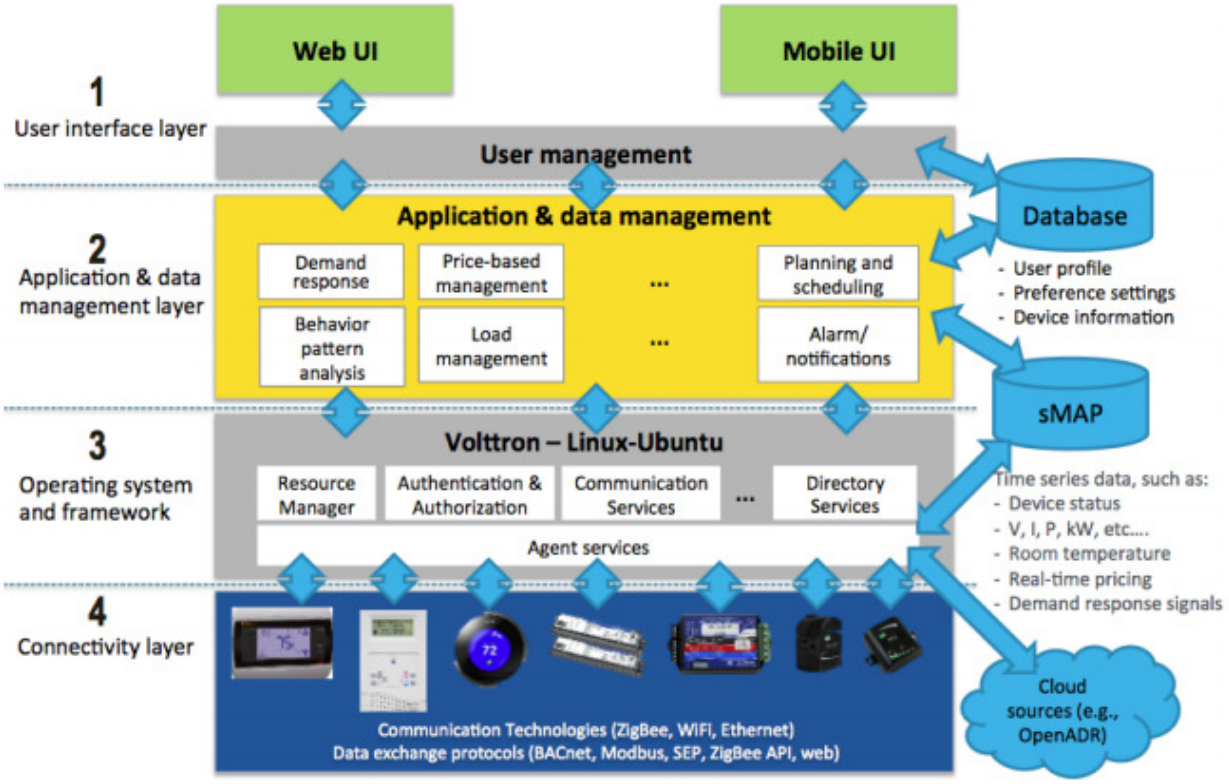


Figure 4-1 BEMOSS software architecture

Layer 1: User Interface (UI) – The research work in this thesis partly corresponds to this layer. The BEMOSS UI layer has two components: UI (i.e., web browser interface and mobile interface) and user management. BEMOSS web UI is a dashboard type interface with visuals and graphs to show current settings of devices in each zone. Authenticated users can also control these devices through an on-site interface.

BEMOSS can also be accessed from smart phones and tablets to monitor and control selected loads in the building remotely. The mobile interface follows the same standards as the web interface. Regarding user management in BEMOSS, role-based access control is implemented to allow different levels of access to different individuals. For example, building engineers will have full authority to adjust set points and schedules of loads in buildings, while tenants and zone managers will have limited access to view current status and historical load data, or control selected loads in specific zones. In BEMOSS, this role-based access control is achieved using access control lists.

Layer 2: Application and Data Management **-** This layer embeds algorithms to allow monitoring and control of hardware devices interfaced with BEMOSS. Examples of possible applications include demand response, price-based management, planning and scheduling, behavior pattern analysis, load management, as well as alarm/notifications. sMAP (Simple Measurement and Actuation Profile) is selected for storing BEMOSS time-series data. A relational database management system (PostgreSQL) stores metadata that identifying users, devices, and process controls. The relational data models are implemented as part of this thesis.

Layer 3: Operating System and Framework **-** In this layer, VOLTTRON™, a distributed core platform developed by Pacific Northwest National Laboratory (PNNL), is chosen as the software platform for BEMOSS. Several agents have been developed to support BEMOSS, including device discovery agent, control agents (i.e., thermostat agent, lighting load agent and plug load agent) as well as monitoring agent (i.e., sensor agent). All agents communicate over an Information Exchange Bus (IEB). The entire BEMOSS system is also designed to allow email/SMS notifications using IFTTT (If This Then That).

Layer 4: Connectivity Layer - This layer takes care of the communication between the Operating System and Framework layer and all physical hardware devices. To allow BEMOSS to communicate with hardware devices that use different communication technologies, data exchange protocols and have device functionalities (different device API's), the BEMOSS team created several API interfaces. Each API interface allows BEMOSS agents to communicate with a group of devices based on their unique APIs. API interfaces provide a translation service for BEMOSS agents so that agents can get readings and send control commands to devices (without knowing their API's) using simple function calls: getDeviceStatus and setDeviceStatus.

4.3 BEMOSS Deployment on a Single-Board Computer

The entire BEMOSS operating system including the VOLTTRON™ package and PostgreSQL database have been installed on a PC and then migrated to single board computers, like BeagleBone and PandaBoard. Specifications of these devices are shown in
Figure 4-4.

CPU: 1GHz ARM Cortex-A8
RAM: 512MB SD
Ethernet: 10/100 RJ45
USB 2.0: Available
Price: $55
Size: 3.4'' x 2.1''

CPU: Dual Core 1.2GHz ARM Cortex-A9
RAM: 1GB SD
Ethernet: 10/100 RJ45
USB 2.0: Available
Price: $220
Size: 4.5'' x 4.0''

CPU: Quad Core 1.7GHz processor
RAM: 2GB SD
Ethernet: 10/100 RJ45
USB 2.0: Available
Price: $59
Size: 3.2'' x 1.8''

Figure 4-2 Specifications of BeagleBone, PandaBoard and Odroid
Source: Beagleboard:http://www.beagleboard.org; Pandaboard: http://pandaboard.org;
Odroid: http://www.hardkernel.com ; [Used under fair use, 2015]

Installation times for entities including Volttron package, BEMOSS user platform, and PostgreSql database are compared in Table 4-1.

Table 4-1 Installation times for BEMOSS operating system

BEMOSS host	BeagleBone	PandaBoard	Odroid	PC
BEMOSS package installation time	2 hours 15 minutes	1 hour 20 minutes	1 hour 20 minutes	20 minutes

4.4 List of Devices Currently Supported by the Platform

Table 4-2 shows the list of devices currently supported by BEMOSS as of May 2015 [68]. All devices in this list can be monitored and controlled from the UI platform.

Table 4-2 List of devices currently supported by BEMOSS

Device Model	Vendor	Protocol
Thermostat		
CT30 w/ WiFi USNAP module	RadioThermostat	WiFi
CT50 w/ WiFi USNAP module	RadioThermostat	WiFi
CT80 w/ ZigBee SE USNAP module	RadioThermostat	ZigBee SE
CT80 w/ WiFi USNAP module	RadioThermostat	WiFi
Nest	Google	WiFi
BACnet Thermostat EXL-01610	Exact Logic	BACnet MS/TP
VAV/RTU		

Device Model	Vendor	Protocol
PL-M1000RTU	Prolon	Modbus RTU
VC1000	Prolon	Modbus RTU
Lighting controller		
WeMo light switch	Belkin	WiFi
Philips Hue	Philips	WiFi/Ethernet
Step-dimmed ballast	Douglas	ZigBee (customized)
LMRC-212-U	Wattstopper	BACnet MS/TP
Plug load controller		
WeMo smart plug	Belkin	WiFi
Digi XBee smart plug	Digi	ZigBee API
LMPL-201	Wattstopper	BACnet MS/TP
VT load controller	VT	ZigBee API
Sensor		
Digi XBee sensor	Digi	ZigBee API
NetAtmo Weather Station	NetAtmo	WiFi
Power meter		
Dent PowerScout 3+, Ethernet	Dent	BACnet IP
	Dent	Modbus TCP
Dent PowerScout 3+, Serial	Dent	BACnet MS/TP
	Dent	Modbus RTU
Wattnode WNC-3Y-208-MB	Wattnode	Modbus RTU

(Source: http://www.bemoss.org)

4.5 User Platform Architecture

The user platform for BEMOSS is designed and implemented as part of this research. The user platform is the end user platform for accessing BEMOSS applications. The user platform architecture is scalable, robust, and secure. The BEMOSS user platform has an open architecture and it is completely modular. It can be easily scaled and adapted to accommodate more devices and functionalities. It is built to encourage iterative development and deployment. The platform uses open source tools to build its applications. The application resides on a web server and usually resides on the most powerful nodes in a BEMOSS connected architecture. The BEMOSS user platform architecture is shown in Figure 4-3.

The user platform is a distributed platform using a variety of different technologies to implement its various features. The overall user platform architecture can be defined as a model-message-view-template architecture.

The user platform has two top-level components: the web handler and the web socket handler.

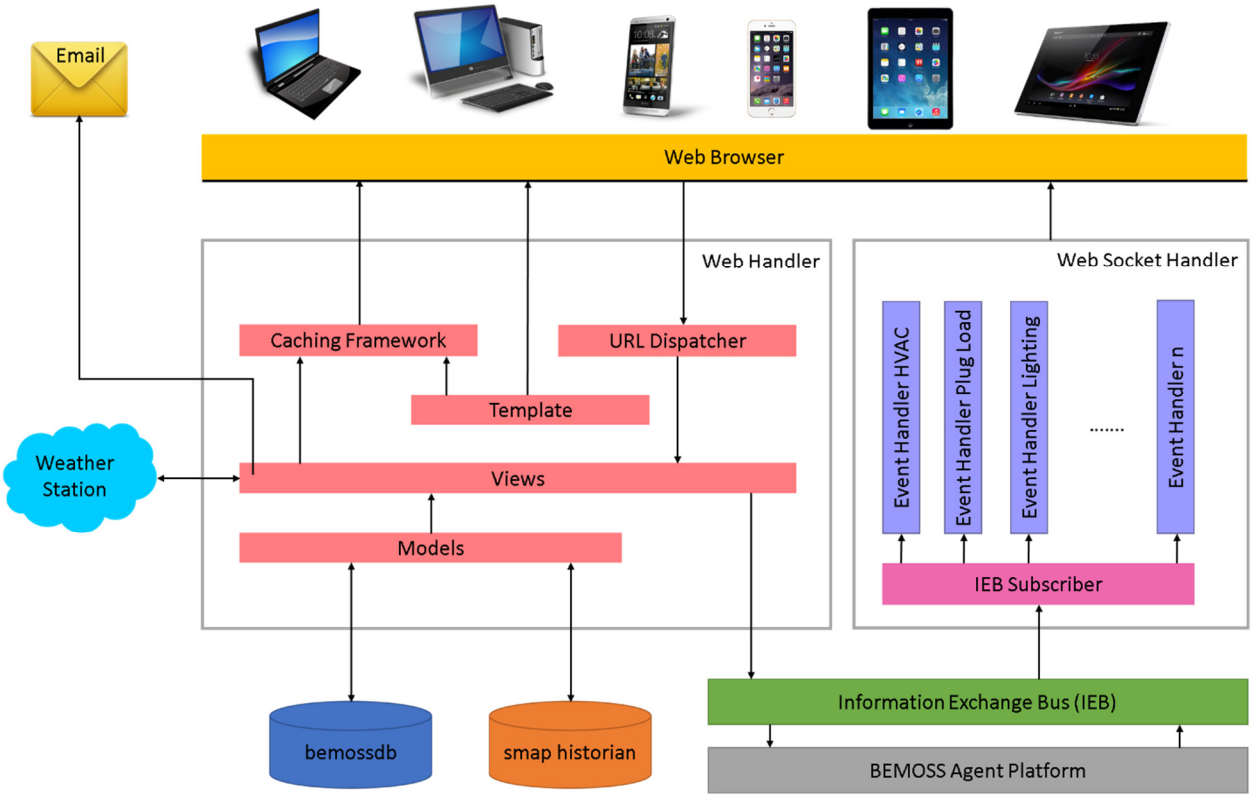


Figure 4-3 BEMOSS user platform architecture
Source: http://www.htc.com; http://www.apple.com/;
http://www.sonymobile.com/us/products/tablets/xperia-tablet-z/ [Used under fair use, 2015]

4.5.1 Web Handler

The web handler is the web server that allows the HTTP communication with the application. It is developed as a model-view-template architecture as opposed to the standard model view controller architecture. The user accesses a BEM system UI via a web browser. The request is processed using the Model-Message-View-Template architecture outlined in Table 4-3.

Table 4-3 UI web handler architecture

Element	**Purpose**
Models	Describes data
Messages	Describes communication with the core platform.
Views	Controls what users see
Templates	How the user sees it

Models: This layer defines the data and interacts with it. It wraps the data access to/from the database using an Object Relational Mapper. Data validation, data behavior, and data relationships can be modeled in this layer. Models designed in this layer interact with the sMAP historian and the bemossdb.

Messages: The messages layer is required to send messages to the control platform for every control input from the user. It acts as a communications platform to the control platform. This layer is also used to receive messages from the control platform and update the web pages. This layer connects to the Information Exchange Bus (IEB) of the Volttron platform to send and receive messages to the BEMOSS OS.

Views: The view function performs the requested action, which typically involves writing/reading to the database. Most of the control inputs from the user are redirected to the core at this point. Certain views communicate with the weather station to retrieve current weather information. The view also takes care of sending emails to users when necessary.

Templates: Templates typically return HTML pages. This layer makes presentation-related decisions – how an entity should be displayed on the web page or other document type. The output is typically HTML but may include other document types.

URL Dispatcher: The URL dispatcher maps the requested URL to a view function and calls it. If caching is enabled, the view function can check to see if a cached version of the page exists and bypass all further steps, thus returning the cached version.

After performing any requested tasks, the view returns a HTTP response object to the web browser via the template. The view can store the HTTP response object in the caching system for a specified length of time. This web handler is analogous to the standard MVC architecture.

4.5.2 Web Socket Handler

The web socket handler is important to allow push notifications to the user and real-time updates. Typical web applications use the AJAX request/response architecture for asynchronous real time updates. The AJAX architecture might be well suited for a regular web application that is based on a simple transactional design.

A BEM system shows real-time updates. This means that a user should see updates as and when they happen without having to refresh the web page. To achieve this, modern web application architectures use asynchronous update features.

The web socket handler serves this purpose. The web socket is the primary entry point for asynchronous real-time updates to the web browser. While the initial update and the control requests are navigated through the web handler, the response from the core platform for every control request is navigated to the UI through the web socket handler.

The web socket handler has two components: Event handler and the IEB subscriber.

Event Handlers: Event handlers are designed to respond to specific events from specific device types. Multiple event handlers exist based on the system load. In most cases, the event handlers are based on device type. For example, all thermostats have a single event handler, which dispatch events to the web sockets listening in on the browser. All the plug loads have a single event handler

dispatching information to the web sockets about the plug loads as and when they receive information. The event handlers dispatch information to the web browser asynchronously.

IEB Subscriber: Whenever a control message is sent to the core platform, it processes the control, and responds to the user platform using a control message. The control message may contain the control status information (success/failure). Device status change information is also relayed to the web browser using the IEB Subscriber. The IEB Subscriber receives all messages targeting the user platform, and relays the information to the Event Handlers that are listening in to specific events.

The web sockets present in the browser are actively listening to these events. Once they receive the events, they process the message in the events and take appropriate action. At no point does the web socket send control information to the platform. It is designed to simply receive information and display it to the end user.

4.6 Implementation Technologies

The BEMOSS user platform is built using several open source technologies that are widely used in modern commercial applications and is built on the open source Linux distribution. The user platform is completely customizable and convenient for developer to focus on application development without the need to spend time on understanding the system. Figure 4-4 shows the different technologies used in different parts of the user platform architecture.

For security and monitoring purposes, other open source tools are also used. These tools are discussed in Chapter 8.

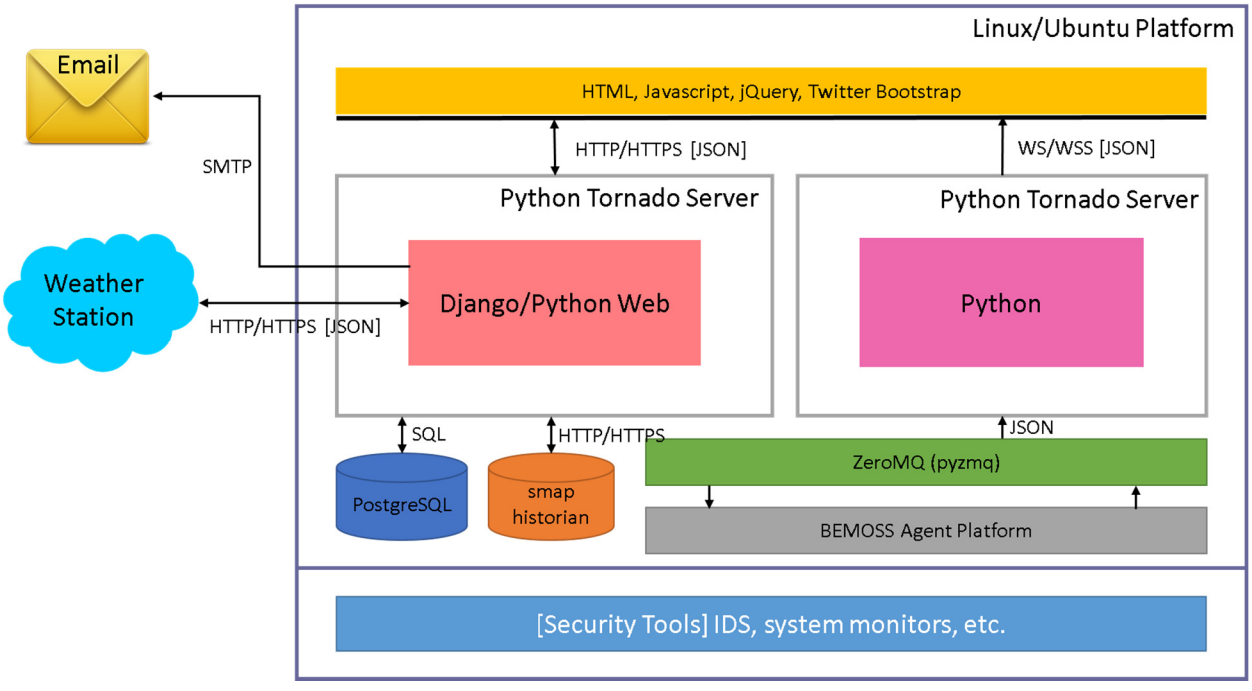


Figure 4-4 Technologies used in the user platform

The entire platform resides in the Linux Platform (Ubuntu, desktop/server). Several security tools including Intrusion Detection Systems (IDS) and System Monitors constantly check for the system and network security are installed in the Linux Platform. These details are discussed in Chapter 9.

4.6.1 The Platform Technologies

Web Pages: The web pages use several technologies like HTML, HTML5, JavaScript, jQuery, jQuery Mobile, Twitter Bootstrap, Font Awesome, etc. A combination of these technologies allows dynamic UI that changes every time devices are updated or device status changes.

Server: The user platform includes a web handler and web socket handler. Tornado framework written in Python is utilized for this purpose. Using two different handlers allows two purposes from a single point of origin. Having two different handlers allows options for load balancing and other future modifications as necessary.

Web Framework: The web framework, Django, is leveraged in most places for the user platform. The powerful ORM features, template tags and the flexibility to add new components to the framework that is loosely coupled with the original framework allows the developer to widen the web platform to build more variety of components.

Database: The metadata for this prototype system is stored in the Relational Database Management System (RDBMS), i.e., PostgreSQL. This was the database of choice because of the powerful capabilities it has and the various tools that can be added to make it work as per the platform necessities.

Python: Python is the base language for the entire BEMOSS application. Python was a choice to identify and work quicker with the underlying Volttron platform. Volttron platform is built using Python.

4.6.2 Communication Technologies

HTTP/HTTPS**:** HTTP or the secure protocol HTTPs is used for communicating with the web browser, and the sMAP historian. It is also used for communicating with the Wunderground [69] weather station from where information about current weather conditions is retrieved.

WS/WSS: This is the web socket protocol similar to HTTP/HTTPS that allows communication between the web sockets in the browser and the Web Socket Handlers.

SMTP: SMTP is the Simple Message Transfer Protocol that is used for sending email communication to the user. In the regular scenario, the user platform sends a user registration request confirmation email to the user.

SQL: SQL is used by the model to talk to the BEMOSS database. The model uses an Object Relational Mapper to encapsulate the SQL messages. However, the actual communications with the database are SQL queries.

JSON: JSON is the preferred messaging format for the messages sent between two entities in the BEMOSS application. JSON is also used for communicating with the weather station. The user platform stores the schedule information in the form of JSON files.

5. User Interface (UI) Design

A UI for BEM system must be easy to configure. Not only should the BEM system suit specific building and business needs, it should also have an intuitive graphical user interface with a user feedback mechanism. The BEM system needs to flexible to be able to tailor to user needs, and be accessible using a web portal from anywhere in the world. Users should be able to lean the tool with little or no training and it should be deployed with ease.

Such systems should have a web portal available and tenant user accessibility to deliver targeted views of data to building occupants. This will in turn allow enhanced energy management. Local and remote access of the web-portal is necessary for the building engineer to accommodate comfort in the building from virtually anywhere.

Graphical user interfacing should be scalable to suit buildings of different sizes. The interface may be accessed via a computer workstation, laptop, smartphones, and tablet using the internet. Each individual user will only access their individual screen, which can be tailored to the user and accessible via password protection.

This chapter discusses the graphical user interface design requirements for accomplishing the above-mentioned needs in the BEMOSS implementation. In the rest of this chapter, the need for a smart UI and the requirements and characteristics of a comprehensive BEM user platform are discussed.

5.1 Requirements in a BEM System

To achieve energy savings in a building, all systems in the building including heating, lighting, plug loads, and security tools should function together as one cohesive unit. Bringing individual systems together leads to better analysis and more intelligent energy use and targeted energy savings.

A wireless system, with highly efficient software capabilities can ensure that these systems operate together. The information obtained from each of these subsystems is intelligently transformed into targeted energy savings using comprehensive algorithms in the system. This often needs feedback from the user/building engineer who has had previous experience working in the environment. Inputs about user's comfort settings and tenant's entry-exit behavior help optimize the energy usage in the building. A quick reference of the current alarms in the system is essential to convey the building engineer of any anomalies in the system and quickly put the system back in order. A smart UI that provides dynamic views of the system conditions and provides logs of the system's status also plays an important role in targeted energy savings.

Today, controllers for major building loads, like HVAC, lighting and plug Loads, etc., can be actively connected to the internet using individual Application Programming Interfaces (APIs). While most of these systems are available individually, connecting them into a single comprehensive system is vital. An open source platform that developers can quickly customize to

their needs will enable a bigger community of users to allow more customization and applications eventually.

A BEM system should bring all types of building sub-systems together. This means connecting even the smallest light bulb in the building to the system. All new devices that are added to the building should be discoverable and manageable from the central BEM system. The UI architecture should allow addition of more devices and controlling them without having to do a lot of modification to the code. A developer should be able easily add new applications to the web portal without having to worry about affecting the existing system functionality. The system architecture should incorporate these functionalities.

In the proposed design methodology of the UI, all of the above key elements are incorporated making it a highly robust, attractive, and easy-to-use interface. The proposed design provides an intuitive interface that allows navigation using standard end-user interface elements with minimum user input requirements. The following sections describe the key characteristics of the proposed BEM system's end-user interface.

5.1.1 Scalability

Focusing on small and medium sized buildings, include services like the doctor's office, lawyer's office, small financial consultancies, or restaurants like McDonalds, Burger King, etc., the challenge is to build a system that can be accommodated in a tiny embedded system like Beaglebone or Panda board and does not need a dedicated desktop computer to run the energy management platform. Such a system should also cater to larger buildings with 2 or 3 floors but within 50,000 square feet. Two different needs for scalability are thus evident here – the system should scale up or scale down and still function efficiently. The proposed design characteristics address all the above mentioned needs.

5.1.2 Modular Design

Modularity in the platform design is as important as scalability considering the variety and number of devices that will become part of the system with time. Dividing an application into logical modules makes using a module for another application in the future or modifying existing applications without disturbing other parts of the application. Modular content will scale better and it is more maintainable and reusable.

5.1.3 Loose Coupling

A good application design should incorporate loose coupling and tight cohesion. The various layers of the framework should not know about the other layers unless absolutely necessary. For example, the template system should not know anything about the web requests, the database layer should not know anything about data display, and the view system should not need to know about which template system a programmer uses.

5.1.4 The DRY principle of development

Every distinct concept and data should live in one, and only one place. Redundancy is bad and normalization is good. Duplication [70] can lead to maintenance problems, poor factoring, and logical contradictions. Duplication, and the strong possibility of eventual contradiction, can arise anywhere: in architecture, requirements, code, or documentation. The effects can range from badly implemented code and developer confusion to complete system failure.

From a developer perspective, organizing source code and following proper coding conventions prove to be useful for long-term development and maintenance. The prototype development follows the 'DRY' principle. Not repeating a common line of code in every file that is being coded is important to maintain consistency in the output and for maintenance. Being a web application developed using HTML and CSS with complex interface elements, the HTML code tends to run over 1000 lines for every single page.

5.1.5 Data Model Design

Fields should not assume certain behaviors based solely on the name of the field. This requires too much knowledge of the system and is prone to errors. Instead, behaviors should be based on key word arguments.

Models should encapsulate the object. Following the Martin Fowler Active Record design pattern, an object that wraps a row in a database table or view, encapsulates the database access, and adds domain logic on that data. An object carries both data and behavior. Much of this data is persistent and needs to be stored in a database. Active Record uses the most obvious approach, putting data access logic in the domain object. This way all people know how to read and write their data to and from the database.

5.1.6 URL Design

URLs should not be coupled with the underlying code. Along these lines, it should be possible that the URLs for the same app can be different in different contexts. For example, one site may use /devices/ for devices, while another may use /smartthings/ for devices. Both should be acceptable. URLs should be as flexible as possible. File extensions in web page URLs should be avoided.

5.1.7 Template System Design

Template system should be a tool that allows control of presentation and presentation logic alone. The template system should not support any further functionality that includes business logic or data modeling.

The template system should follow the inheritance principle. This will allow a common site-wide design – a common header, footer, and navigation bar.

The template system should allow non-HTML formats such as plain text. Most template systems ignore whitespace and trim any whitespace content. While this may be appropriate in certain

situations, a white space should be treated as whitespace in the template tags. It should also account for security by not allowing insertion of malicious code.

5.1.8 Views Design

Writing a view should not involve writing a class, a functions within a class and having to instantiating them on every run. It should be possible to run a view function without having to instantiate a class. Views should access the request object using the framework's APIs rather than reinvent the wheel. The object corresponding to a view should be available in a view function, even if it involves dynamically populating the view.

5.2 User Platform Design

5.2.1 Model-Message-View-Template (MMVT) Architecture

The architectural design of a user platform with the above requirements should be easily configurable and navigable. It should allow new components to be added to the system intuitively as and when required. For an intelligent agent-based energy management system, it is important to segregate the user platform from the core platform. However, a need to pass messages from/to the core platform is necessary.

The general MVC (Model-View-Controller) architecture becomes obsolete in this situation since there is a need for something more than just accessing the model relevant to the application. The smart device in the building should be controlled using input from the user. A message should be sent to a device and to the core platform therefore for every input, a message is essential. A modified MVC architecture – the MMVT architecture is devised to realize the needs of the user platform.

The details of this architecture are discussed in Section 5.5.

This design is completely modular and well-suited for a BEM user platform design. Adding new devices to the platform and devising new functionality gets quicker and more organized.

5.2.2 Quick Setup and Deployment

An open source system involves a variety of open source components and plugins. This makes it difficult to maintain these components because of frequent updates made to the supporting plugins. The user platform works on a very specific version of the web framework ensuring that this error is nullified. The user platform can be installed with minimum commands, and conveniently using a script file. Using a setup script ensures that there is minimum errors and the developers / users do not have to go through a long process to get setup.

The proposed design involves starting up the web server and the web socket server as separate elements. This is merged together by using a server script that runs the whole user platform as one entity. Running the core platform subscriber is a single command on the terminal. Greatly simplifying setup and deployment will ensure that more developers are willing to contribute to the platform – one of the primary needs of an open source platform.

5.2.3 Data Flow

Data flow in such energy management systems should follow a defined process that allows replication of similar scenarios when building new applications in the same platform. A successful implementation should not overload the system with multiple messages for a single control instruction and also convey information appropriately. The system should be able to define a general information flow for every specific instruction possible. For example, for a general control message the data flow could be defined as follows:

i) Collect information from the web page.
ii) Validate information in the user platform.
iii) Send information to the core platform.
iv) Core agent processes information, updates device using the setDeviceStatus API.
v) Core platform sends a success/failure message to the user platform.
vi) User platform receives the message using a web socket event handler.
vii) The web page validates, processes the information, and displays the result to the user.

This data flow is common for all device control inputs that the user submits to the application. Similarly, defined processes should exist for every activity in the application.

5.2.4 Communication

Communication between the different entities of the user platform and with the control platform should be defined before the application is developed. A proper communication methodology will ease the development process and clarify the data flow methodology. The application might communicate with the control platform using message queues, or network sockets, or using web services. The protocol for communication should also be laid out. The inter-user platform communication can be using the web framework protocols and in some cases, third party web services may also be involved. Ajax asynchronous calls may be involved and the appropriate protocol to handle the message validations and processing should be laid out. GET messages should not reveal excess information about the application or have some vulnerability that the user can use to navigate to unauthorized places.

For example, the communication for all real-time asynchronous updates could be using web sockets, and the asynchronous user control inputs could be performed using Ajax calls. The communication to the control platform could occur using the message queues and the responses can be handled using event handlers listening in on specific messages. An appropriate communication mechanism needs to be laid out for successful platform development.

5.2.5 Deploying Application to the Server

In a BEM system, multiple layers exist to separate platform functionality. The control platform can operate separately from the user platform. In this case, the control platform need not be deployed to a server. However, the user platform is mostly a web based interface which needs to be running on a centralized server for users to access it using a web browser. Such a deployment should include setting up the environment and ensuring that the server is configured with all requirements appropriately.

In general, when deploying to the server, the domain and port information on which the application is deployed should be specified. If the IP address is mapped to a domain, the mapping should be appropriately configured in the server configuration files. The static files or the media files of the application should be configured in the server with the right path for the server to retrieve them as required. Aside from all of this, the server IP whitelists and blacklists should also be made to go live as the server is started up. The server should be configured to perform these operations appropriately to ensure availability of the web application. If multiple types of application handlers are used, all of them should be started up with the single server start-up mode. The server should be secured behind a firewall.

5.2.6 Serving Static Assets

Static assets/media files in a web application are essential for the client side functionality in an application. For applications which are real-time and asynchronous, important functions usually occurs using the files served through the static or media folder. Files including JavaScript, jQuery, images, CSS, font templates and other visual feature rendering form part of static assets in a web application. These static assets should be mentioned with the appropriate folder location in the server. The server uses the static root to render any static files.

To improve performance, it is generally a good idea for browsers to cache static resources aggressively so browsers won't send unnecessary requests that might block the rendering of the page. Using servers that allow static content versioning should take care of such scenarios. If the application is heavy and required a more optimized performance, it might be a wise decision to serve static files from a separate static file server like nginx.

Within the static folder, it is important to organize the static assets appropriately. Some applications might throw in files of all kinds in the root static folder. This might work for tiny applications, but as applications grow in size and become more complex, a modular system might be appropriate.

For example, the static folder can have subfolders like application_javascript, base_javascript, css, images, required_javascript, font_templates, etc. There could be subfolders as necessary. Making modularity a base requirement at every level will avoid any future application debugging and platform-wide reorganization issues.

5.3 UI Design

UI is an important component in a BEM system that allows human-machine interaction. A single uniform graphical interface can be used to monitor and interact with all connected devices. Graphical design, semantics, and typography should be used to interact with the user. A good interface design must balance technical functionality and visual and mental models to create a platform that are operational and adaptive to changing user interaction and needs.

Most BEM systems require focused training for the end users. A neat and comprehensive UI design will reduce the need for training and allow the users to self-learn. It is important for such an interface to adapt dynamically to different user types. Local and remote monitoring systems are

on the rise today and allowing web-based interaction is necessary today more than ever. An administrator type interface will be the most appropriate for the needs of an Energy Management System. Minimal use of text fields and areas where the user input from a keyboard is required can be achieved in an administrator-type interface.

UI design should follow certain established design principles to immediately appeal to the end-users and make it more readable. A specific process to follow for the UI design will ensure the usefulness and intuitiveness of the interface.

The primary features of a good UI design for BEM systems are listed below:

i) Intuitive
ii) Dynamic
iii) Minimal
iv) Usable
v) Real-time interaction
vi) Clear and concise

5.3.1 UI Design Process

The processes to deliver a final UI design [71] to end-users is shown in Figure 5-1 and described below.

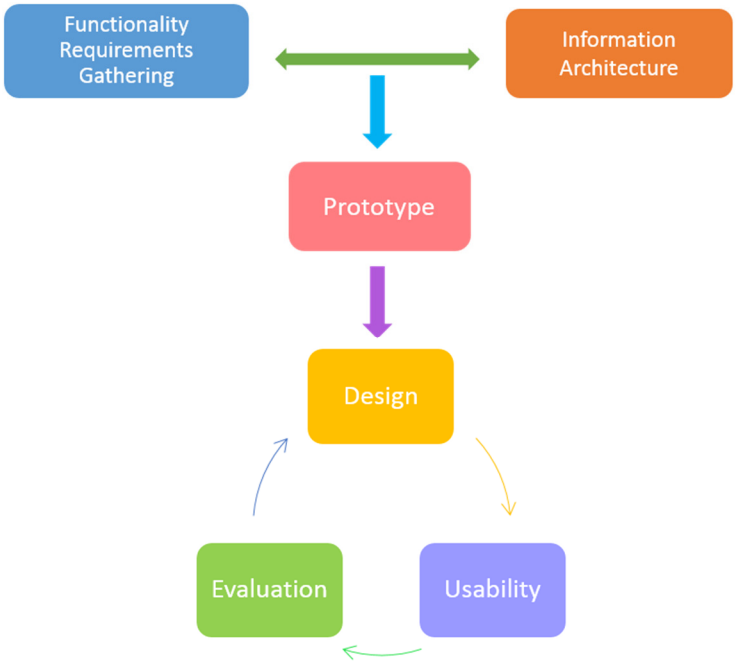


Figure 5-1 UI design cycle [71]
Lauesen, S. 2005, User interface design: a software engineering perspective, Pearson/Addison-Wesley, New York; Harlow, England [Used under fair use, 2015]

- Functionality requirements gathering collects the requirements of the graphical UI.
- User analysis addresses the following questions:

    - What is the purpose of the system for the user?
    - How technically oriented is the user?
    - Has the user worked on similar systems before?

- Information architecture develops the process to guide the user through different areas of the interface, and the intuitiveness of the process for the end-user.
- Prototyping – A non-functional and wireframe type prototype of the proposed UI design for every page or every view in the system should be prepared as part of the design process. This can be hand drawn or simple wireframes of the end-user interface.
- Usability Testing – A designer's goals might not meet with the ultimate user's need if the prototype is not tested with an actual user.
- Graphical Interface Design – Final part of the design process, when the actual fully functional model of the UI is design and ready for use.

5.3.2 Minimal and Responsive Design

A lightweight design that is intuitive and responsive is important for a heavy system managing multiple components and handling huge data sets. A lightweight UI design focuses on anticipating what users might needs to do and interface elements that are easy and intuitive to access and interact with.

The proposed UI design brings together the concepts from interaction design, visual design, and information architecture. In today's internet-driven world, users have become familiar with interface elements acting in a certain way, and the proposed design makes sure that it is consistent with the user expectations and predictable in terms of choices and layout. This ensures the efficiency, satisfaction, and minimum requirement for user training.

Minimal use of images, and substituting images for vector icons wherever possible, using shades of grey as opposed to image based UI elements, and custom backgrounds to highlight the shades of grey together contribute to the lightweight and intuitive design of the UI.

<u>Minimal use of images</u>

Images are a primary feature to enable intuition in navigation for the end user. However, more images also mean additional server load that might slow down the interface. The proposed design uses very few images keeping in mind the complexity of the interface. Image scaling is a frequent problem in web development – today there are numerous devices of a variety of sizes, which makes it difficult to resize images not knowing the screen size the page is being rendered in. Icon fonts have been used in place of images wherever possible. Icon fonts offer scalable images that look the same irrespective of the size of the font.

Images are being used only in places where intuitiveness is lacking when font icons are used. For example, the dashboard page has images of smart plugs, HVAC controllers, etc., which is intuitively identified by placing the actual device images in place of icons. This feature is made available in the dashboard page where the user normally visits to get an overview of devices or identify devices/modify device nicknames.

Vector Icons

When you design a website, images take up a lot of web space due to their file sizes. They have to call every time a web page using those images is rendered. They have a variety of problems including image scaling, losing definition of increased size, etc.

Icon images [72] can replace regular images for some common characteristics, such as temperature, vehicles, social networking, etc. The scaling is similar to that of letters and numbers irrespective of any size that is used. By using Icon Images, basic icons can be added to sites, and scaled to the required size, limited only by the dimensions of their documents, and without losing definition. Colors can be changed and transparency can be controlled by using these images. An example of the icon images and their names are shown in Figure 5-2 below.

Figure 5-2 Vector icons
Source: http://www.webiconset.com/minimal-vector-mini-icon-set/
[Used under fair use, 2015]

Utilizing Shades

A minimalist design shows data in the most efficient way as possible. What is important for BEM system User platform is an efficient UI. Using boxes and other image elements to compose a widget or element on the UI is similar to adding images to a website. Those many number of image elements have to be requested from the server every time a page loads.

To minimize such complexity, the proposed design utilizes shades of grey to represent boxes and widgets as opposed to using complex image backgrounds. A simple background-scaled image serves as a background. The elements in the UI are spaced on top of this background using varying shades of grey. This makes it easier to scale the design over multiple viewer sizes (tablets, smartphones, and desktops of different sizes). Having shades of grey dictate the widget elements makes the web pages behave appropriately corresponding to the different device sizes. A 12-column responsive grid with plenty of components and widgets, JavaScript plugins, typography, and a customizer is added in the proposed design.

5.3.3 Custom Backgrounds

The idea of lightweight design includes minimum use of images. This brought in the use of shades of grey for widgets and creating user-friendly modals and tools. To cater to the needs of all the requirements in the web design, and to keep it minimal and catchy, the idea of custom backgrounds is used in the proposed design.

Different users have different preferences, and they can customize views. Each custom background is an image of size 25 pixels by 25 pixels, which is a very small size to cover the entire web page but enough to create unique views. The image - scaled to web page size - acts a background cover. All the UI elements are placed over it. A sample of custom backgrounds of the same size is available to choose from.

5.4 Designed For Open Source

The design under discussion is for an open source platform. An open source platform is built using a combination of open source tools that are fairly maintained. The proposed user platform is completely open source and allows the user to jump in and quickly build pages and applications.

5.4.1 Custom Built Device Application and Data Model

In the user platform, every device type has a specific app that takes care of controlling the web page for that device type. For example, a thermostat is a HVAC controller, and has a device application. Each application means that there is a specific type of data model that has to be built to communicate with the database and send data to the web page. The thermostat model would comprise all essential data points and platform operating characteristics programmed in the application.

Whenever a user requests a page, the URL Configuration file is referred to land on a custom views script that collects the information necessary for that particular request. It communicates with the core platform if required and with the database using the data model, gets the necessary information to be displayed. The request context is then updated and the control is transferred to the HTTP page where the corresponding JavaScript is executed.

This is achieved with the URLs script, which defines the path to an application, and the script to execute for any request.

5.5 UI Design Features

UI makes use of lightweight elements that make page load faster and less error-prone.
Interface elements include but are not limited to:

i) Input Controls: buttons, text fields, checkboxes, radio buttons, dropdown lists, list boxes, toggles, date field
ii) Navigational Components: breadcrumb, slider, search field, pagination, slider, tags, icons
iii) Informational Components: tooltips, icons, progress bar, notifications, message boxes, modal windows

iv) Containers: accordion

There are three fundamental principles involved in the use of the visible language.

i) Organize: provide the user with a clear and consistent conceptual structure
ii) Economize: do the most with the least amount of cues
iii) Communicate: match the presentation to the capabilities of the user.

All of the above principles have been incorporated in the UI platform design.

5.5.1 Consistency

Best practices in the industry have been employed to design the UI. The interface is clean and simple, and avoids unnecessary elements. Clear usage of labels and messaging ensures that the interface is intuitive. By using common elements in the UI, a comfortable experience is created and this helps a user to navigate the system quickly. The UI design ensures that there are definitive patterns created in language, layout, and design throughout the application. This helps the user navigate through the application pages and perform the intended activity without confusion.

For example, in all device dashboard pages, there is device nickname and zone information at the top left corner. The following lines add device control and monitor elements. The submit button, set schedule button, and the view charts and statistics button is consistently available in the bottom after all elements allowing the user's intuition to take charge while navigating to different pages. This type of design is ensured in all of the web pages.

5.5.2 Spatial Layout

Spatial layout of information has been given importance in the interface design. Careful placement of items on the page can help draw attention to the most important information and aid readability. Color, contrast, and texture have been used to draw the attention of users towards information that should not be missed. Different sizes, fonts, and arrangement of the text enable improved readability, legibility thus providing clean view of data as opposed to cluttered.

In the end, the UI design is meant to provide information about the underlying system that is being controlled. Appropriate information and fast notification access is provided.

5.5.3 Wireframes

Wireframes are used in the design of every page. Wireframe is a basic visual interface guide that suggests the structure of an interface and the relationships between its pages. They serve as a blueprint that defines page structure, content, and functionality. Wireframes undergo several changes as they are developed into a fully functional web pages. A wireframe for one of the pages is shown in Figure 5-3.

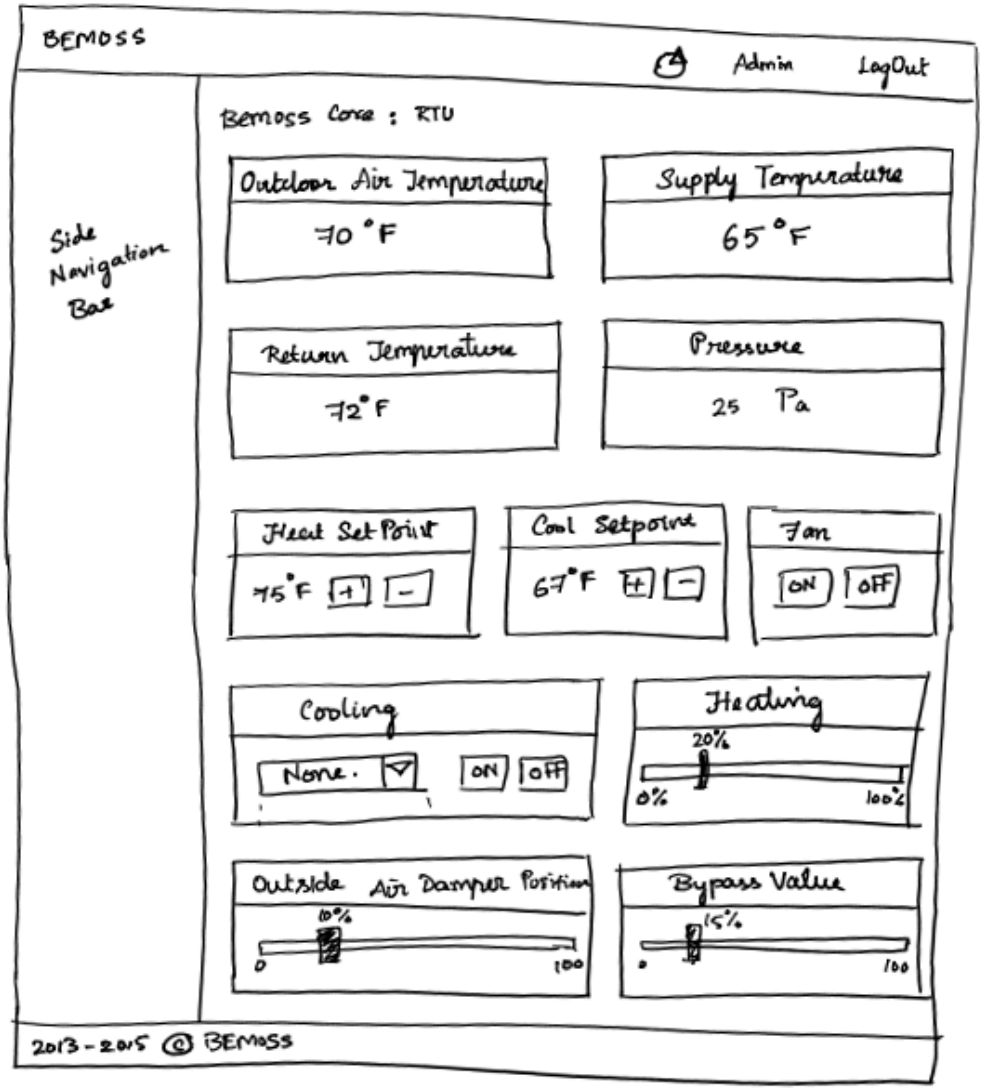


Figure 5-3 Example wireframe

5.5.4 Principle of Proximity

Several principles and laws should be considered for the design of every web page to bring about a clean and driven UI design. Principle of Proximity, one of Gestalt principles [17, 20] of perception is said to have the greatest potential impact for an application (web or otherwise). The principle states that we perceive relationships between objects that are closer together. Conversely, objects that are further apart would, seemingly, have less relation.

Items that are placed closer together are usually perceived as a group. For example, in Figure 5-4, we perceive three separate groups:

Figure 5-4 Objects grouped together
Source: http://architectingusability.com/2011/05/26/using-the-gestalt-laws-of-perception-in-ui-design/
[Used under fair use, 2015]

On the other hand, in Figure 5-5, objects appear to be arranged in rows and columns rather than grouped together. This is due to their proximity to each other. The distance between the dots making up the rows and columns less than a distance between a dot in one row or column and the nearest dot in the next row or column.

Figure 5-5 Objects farther apart - appear as arranged in rows and columns
Source: http://architectingusability.com/2011/05/26/using-the-gestalt-laws-of-perception-in-ui-design/
[Used under fair use, 2015]

Figure 5-6 demonstrates the application of this concept to the user registration page. The image on the left is what the registration form would look like if it was designed without the proximity principle. The labels are placed so far away from the text fields and looks like they form separate groups. The connection is not instantly clear. The image on the right fixes this issue by reducing the distance between the labels and their corresponding text fields.

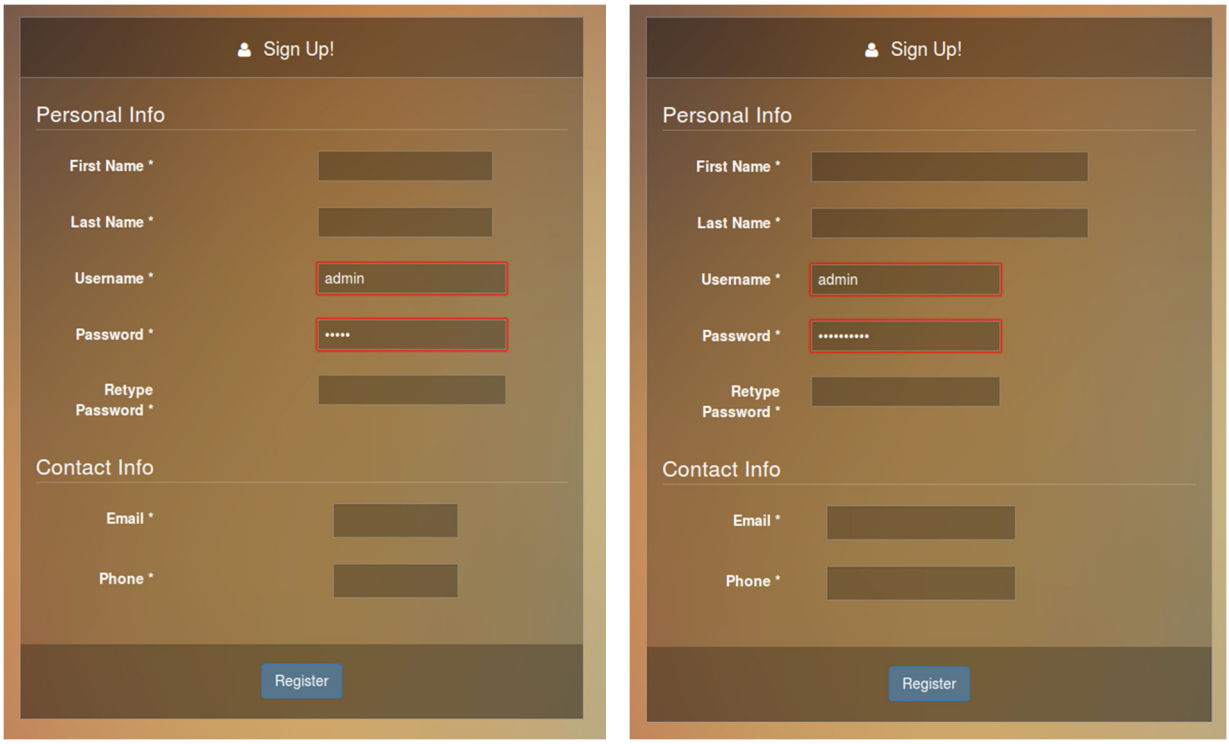


Figure 5-6 User Registration page demonstrating principle of proximity
Source: http://www.bemoss.org. Used with permission, 2015.

5.5.5 Progressive Disclosure

Wherever possible, elements should have a hierarchy for easy navigation through the system. When a user cannot perform a particular function, it should be shown as disabled. In some cases, the functions that are not allowed should be hidden.

For example, in the thermostat page, the administrator can change the thermostat mode while a tenant cannot. The tenant still needs to know the thermostat mode. Therefore, buttons corresponding to the thermostat mode are disabled but visible to the tenant.

6. Implementation of the UI platform in BEMOSS

Based on the UI design principles discussed in Chapter 5, a user platform is implemented in BEMOSS. The modular architecture layout of BEMOSS enables the implementation of a complete layer for the user platform. Several open source technologies were incorporated in building the user platform based on standard UI design principles. This chapter discusses some of the key features of the prototype implementation.

6.1 Scalability

The prototype is designed for small- and medium-sized buildings such as small offices, fast food restaurants etc. Most of these can use the BEMOSS for building energy management. A large building (over 10,000 sqft. floor area) can still use BEMOSS using the multi-node feature and operate the UI from any device such as a tablet, a smart phone, or a laptop. The UI is light and responsive to be operated from any smart device.

6.1.1 Large Buildings Monitor and Control

The design caters to large buildings by providing a distributed platform that is seamlessly interconnected to a core system. A multi-node system consisting of smaller embedded systems connected to a larger centralized core system can manage large buildings. The embedded systems communicate with each other through a messaging queue and transfer data into a visual UI.

The initial setup begins with all devices connected to the core system as they are discovered. The administrator can then move devices to zones. These zones can correspond to different floors, different systems, or any other logical group. Once grouped, all devices in a zone are further combined by function internally. With every grouping, messages are sent between the system core, the UI, and grouping to synchronize information.

6.1.2 Communicate locally or via the cloud

The user platform communicates with the core using the ‘inter-process communication’ technique. This way, communications are simplified and happen within the same machine or embedded system.

If the user platform is hosted through an internet server (the cloud), the UI can switch to ‘TCP’ communication mode to communicate over the Internet with the core.

6.1.3 Scale up or Scale down

The UI can be hosted in any device from a powerful server computer to a small embedded system. The UI elements like buttons, boxes are created using color shades instead of complex elements. Images used are also lightweight and measure only 25px x 25px.

Smaller devices hosting the UI can therefore function with ease and provide a seamless experience to the end users. However, since these systems have processor limitations, not more than 50 devices can be hosted.

In large server machines, the lightweight design allows more devices to be hosted and accessed from the same interface without any system lags. Not all device information is loaded in the page at once. By logically separating devices based on function, the page-loading time is reduced significantly.

6.1.4 Seamless Access

The proposed UI is designed as a web portal that is accessible using a simple web browser that is connected to the Internet. Users of the BEMOSS application therefore can have seamless access to the UI from virtually anywhere.

The interface is also mobile-optimized access using any type of smart device. This was achieved by using a lightweight web container for information display. The UI pages are highly responsive and can dynamically change the alignment and view behavior based on the size of the screen from which the application is accessed.

6.1.5 Modular Design

While a small building has limited number of devices and loads, a large building can have many devices, sub-systems, and loads. A modular design of the UI allows easy access and monitoring/control operations for the user. Appropriate naming conventions with easy-to-identify device images make the interface more readable and user-friendly.

In the proposed design, the user can navigate to a zone or a device intuitively. A comprehensive list of devices is available to provide a quick overview of the system.

To control these device subsystems, the administrator can use to the device page and, if available, schedule the device for a specific setting. Designing a modular view is easier to navigate intuitively and gives a quick perspective of the system status from the comprehensive view page.

6.2 Role-based Access Control

The proposed system provides access to the building engineer as well as its tenants. Allowing tenants to monitor their energy consumption and assess factors affecting consumption such as building occupancy. The system is also at a risk if it is open to all users of the building. Therefore, certain features are restricted only for the administrators. This is implemented using a role-based access control.

6.2.1 Authenticated Sessions

The design allows access only to registered users. Authentication is performed with the login screen, which is the first screen of the BEMOSS page. The user is prompted to enter username and password provided during registration. On providing the correct credentials, the user is logged into

the system. The system maintains a list of registered users and the session information. The user is authenticated using this information during the login process. For every page the user navigates to, the session is checked to see if the user is authorized to access the page. On log out, the session information is invalidated and the user is logged out.

6.2.2 Login, Log Out

Figure 6-1 shows the login screen, the primary landing page to the BEMOSS system. The login page prompts a user name and a password from the user. If the credentials are verified, the user is redirected to the home page. If the credentials are incorrect, the user redirected back to the login page with an error message.

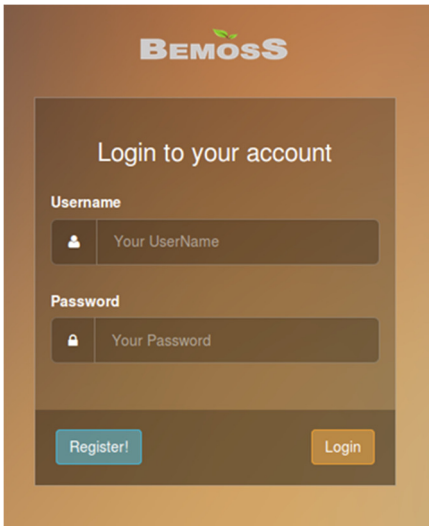


Figure 6-1 User login screen
Source: http://www.bemoss.org. Used with permission, 2015.

Figure 6-2 shows the logout buttons in the system. When the user logs out, the user session expires and will require re-login to use the system.

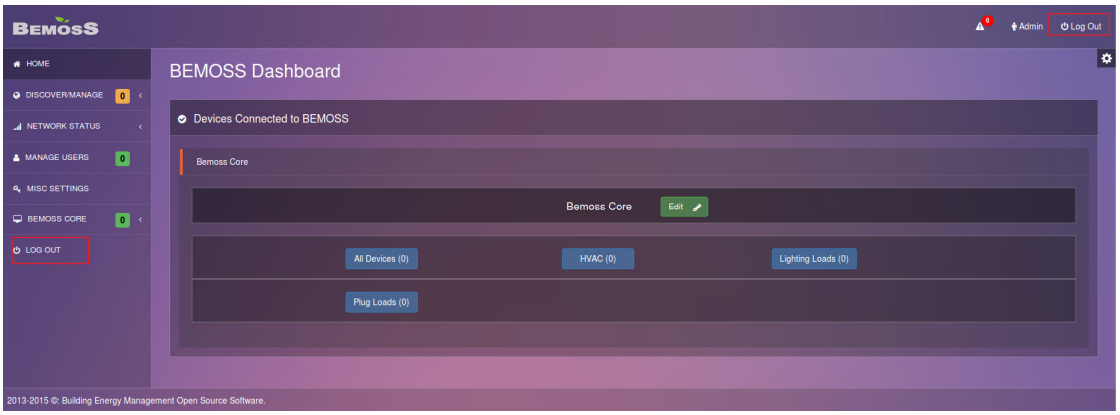


Figure 6-2 Use log out feature to end a user session
Source: http://www.bemoss.org. Used with permission, 2015.

6.2.3 User Roles

Figure 6-3 summarizes the role-based access control design.

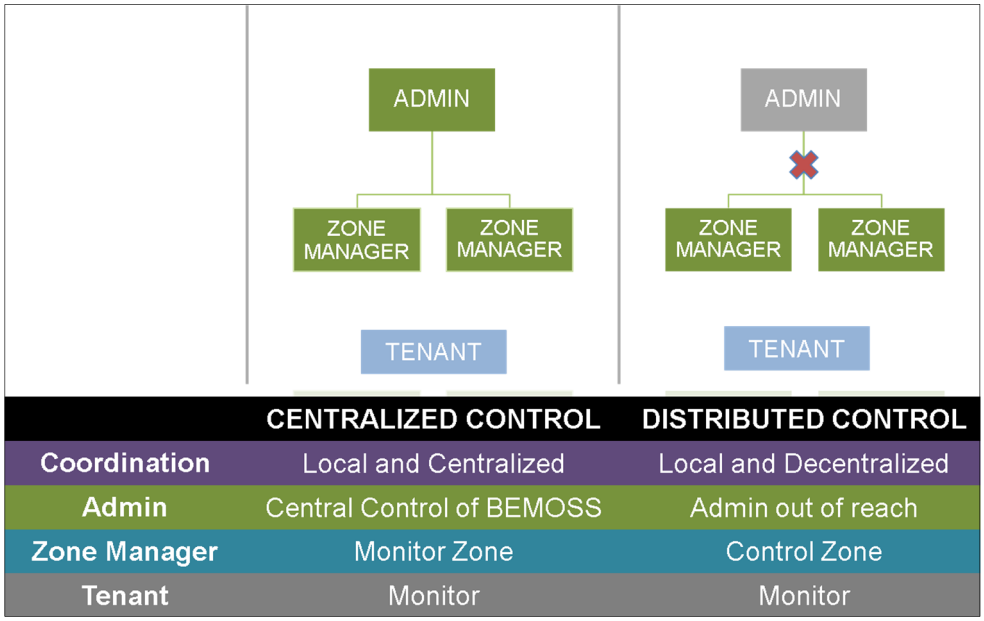


Figure 6-3 User roles

The system provides three user roles as described below:

i) Administrator

The administrator has the highest privileges and complete access to the system and can monitor and control the system without any restrictions. The administrator approves any new user registrations, and assigns zone managers to particular zones. The administrator authorizes new devices. The administrator can schedule devices, setup alarms, and notifications for any registered events in the system. The administrator can also generate reports for further analysis.

ii) Zone Manager

The administrator can assign a user to manage a particular zone in the building (in case of a large building). The zone manager can then control devices within this zone. The zone manager effectively acts as an administrator to the zone. However, the zone manager does not have control over device name modification. This restriction preserves system naming conventions. The zone manager role can be assigned and revoked as per the needs of the building engineer. Once the role is revoked, the zone manager is assigned the tenant role or the administrator role.

iii) Tenant/User

The tenant is a general user in the building. A tenant can monitor devices but cannot control them. A tenant can request for any change in system or device settings, if needed. This is

allowed using a 'Request Change' button in device pages. The zone manager or administrator can service this request by either allowing it or rejecting it based on system conditions. Change requests are a way to get user feedback, and registered in the system and processed during analysis.

The role of a zone manager allows multiple people to be involved in building operations rather than only the administrator. Having decentralized control using the zone managers makes it possible for controlling the building facility even in the event of the administrator being unable to access the system.

i) Centralized Control

In everyday scenarios, the administrator/building operator is available in the premises or connected to the building network to perform routine operations of building (including zones). Even though zone managers are available in the building, the administrator(s) can take control of the building operations bearing a few regular and general cases.

ii) Distributed Control

There might be scenarios when a building operator may be out of reach /without Internet access. In such cases, the building operator can delegate control operations to the zone managers. In the absence of a building operator, a zone manager's role becomes active, thus allowing distributed control.

6.2.4 Role-oriented Dynamic Page Views

A role based system provides the flexibility of feedback from tenants as well as decentralized control. A role-oriented system also means that multiple people must be able to login at the same time. Ease of use and flexibility based on user roles should be achieved at the same time.

The proposed design checks for access restrictions for a user at every critical point, like a 'submit' button or a specific page link. The user is authenticated and authorized for every page load. The page view is modified dynamically based on user permissions. For example, a 'Request Change' button is all that is available to a tenant, while a 'Submit Changes' button is available to the zone manager or an administrator. The administrator has access to general settings pages, user manager pages, and system status pages while the zone manager and tenants do not get access to these pages. The administrator can also access the alarms and notifications pages, create new alarms, read notification logs, etc. while the zone manager or tenants only receive notifications, if authorized, in the form of a text message or email as per system settings.

The design using the access control API that is developed for this purpose. The database is populated with the user registration and permission information and connected to the UI using controlled database models. Functions like 'is-active' and 'is-authenticated' are developed to verify if the user that is logging in is an active user and has provided the correct credentials.

6.2.5 Customized Home Screen based on User Role

The BEMOSS home screen is customized based to the user role. For the administrator, all features are available. Figure 6-4 shows the administrator home screen.

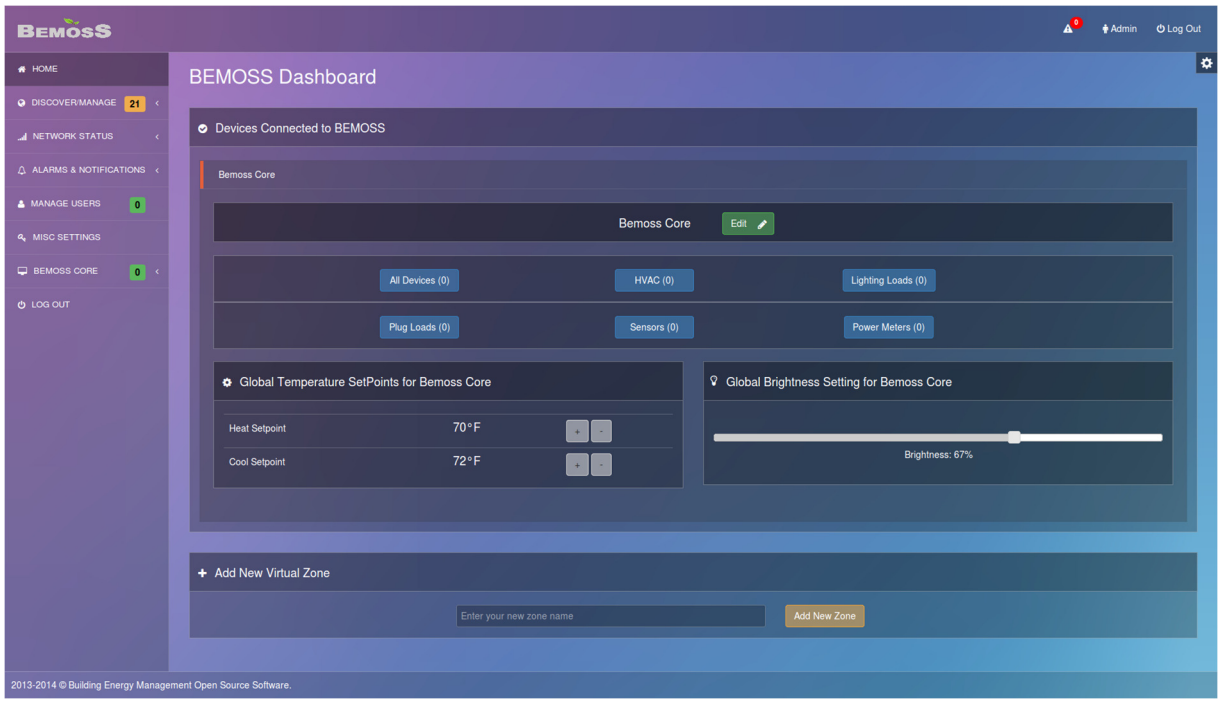


Figure 6-4 Administrator home screen
Source: http://www.bemoss.org. Used with permission, 2015.

For the tenant, only device information and current status features are available. The tenant has no control access to system features. The zone manager gets control access to devices in the assigned zone, and monitor access for all other zones, effectively playing the role of a tenant for the rest of the zones. Figure 6-5 shows the home screen for tenant and zone manager.

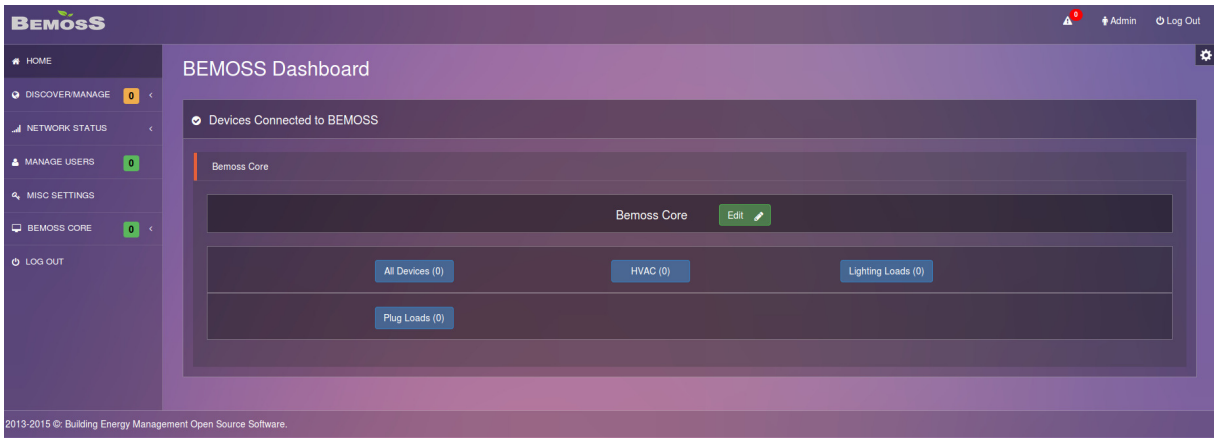


Figure 6-5 Tenant and zone manager home screen
Source: http://www.bemoss.org. Used with permission, 2015.

The home screen features a rich accordion of display for each of the zones in the system. On opening each accordion tab, the tab opens a list device types, and devices in each type.

6.3 Device Discovery and Acknowledgement

In BEMOSS, all discoverable devices in the network should be identified by the system and instantly displayed in the device discovery interface. Information about the newly discovered devices such as device type, name, and address should also be displayed.

If the discovered device is not to be monitored or controlled, it should be removed from the system. This calls for an approval process. This approval process is built in to the UI. Once plugged into the network, a smart device is discovered by BEMOSS and displayed on the system, pending approval. After approval, it is assigned to a specific zone and acknowledged into the system.

This process of device approval makes sure that only authorized devices are entered in to the BEMOSS system. An unapproved device could be a cause for cyber-attacks if left unchecked. Figure 6-6 shows the process flow from device discovery to approval. The user interface for the administrator to manually approve the devices and move them from status 'Pending' to 'Approved' or mark them as 'Non-BEMOSS device' is implemented in the user platform. This process is discussed in detail in Section 8.4.13 Device Approval Process).

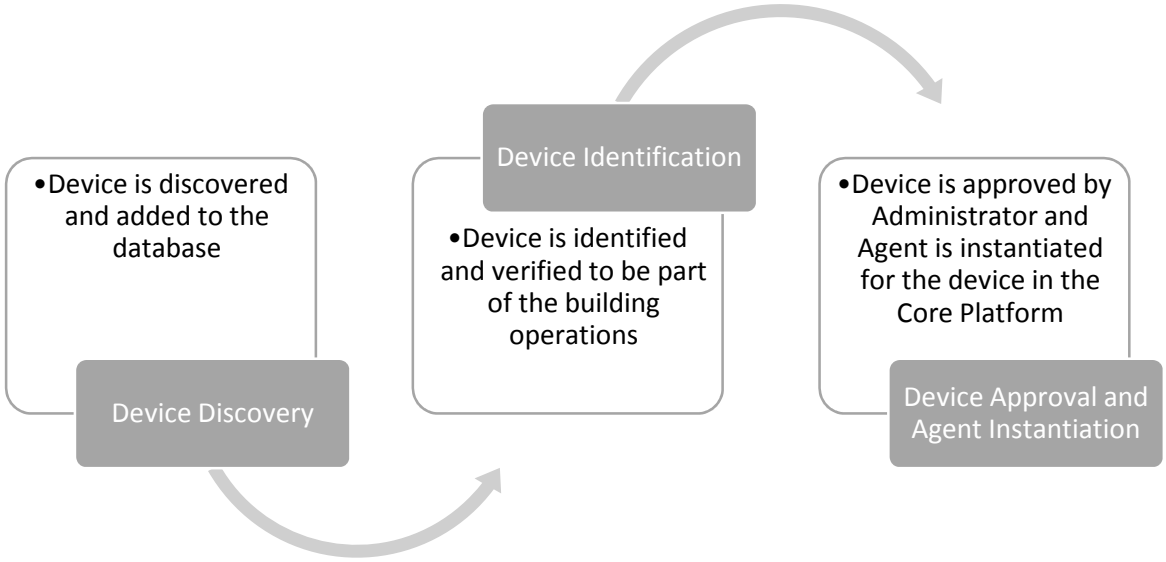


Figure 6-6 Device discovery and approval process

6.3.1 Administrator Control Panel

Control panel refers to specific pages in the UI for device and user management. These pages are accessible only by administrators and are collectively termed as control panel. The control panel includes the following:

1) Discovered devices dashboard

When devices are first discovered, devices are populated in the discovered devices dashboard. The administrator uses the dashboard to approve or reject devices. Figure 6-7 shows the discovered devices dashboard page. Intuitive graphical elements such as numbered tiles have been used to focus an administrator's attention to new devices discovered in the system. Dropdowns, tab controls, paginated tables, and a search form to allow search function in case of more than one page of devices available is available in the interface. All these collectively allow intuitive and quick navigation between the different system functions.

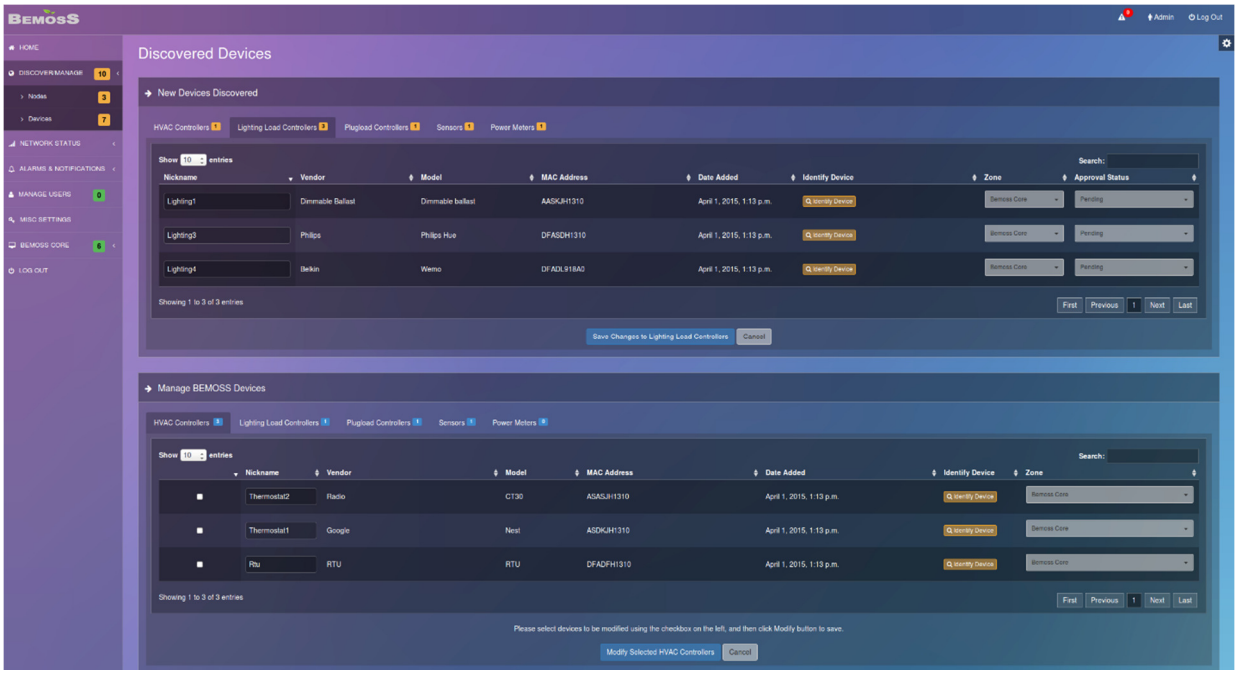


Figure 6-7 Discovered devices dashboard
Source: http://www.bemoss.org. Used with permission, 2015.

2) Discovered nodes dashboard

The Discovered nodes dashboard is similar to the discovered devices dashboard and shows all of the discovered nodes connected to the BEMOSS core. Figure 6-8 shows the node dashboard page. This dashboard also uses the same intuitive features as the Discovered Devices dashboard to ensure consistency and seamless interaction. Use of proximity and progressive disclosure principles aid the ease of use of this interface.

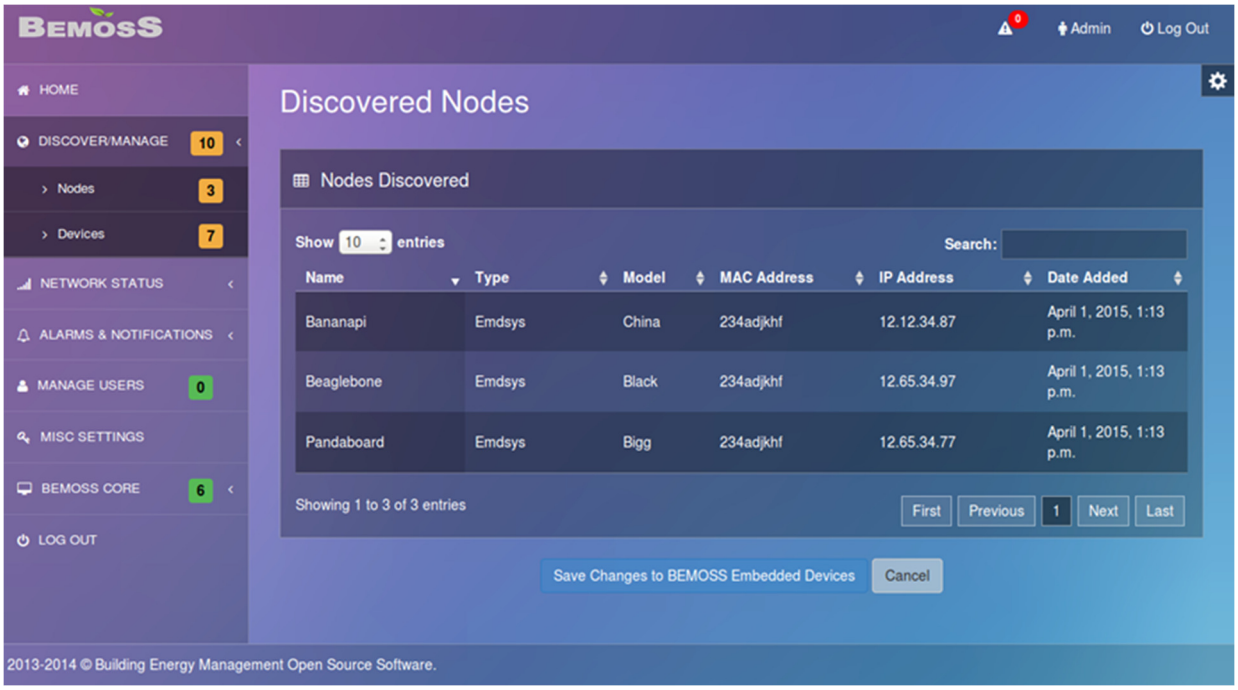


Figure 6-8 Discovered nodes dashboard
Source: http://www.bemoss.org. Used with permission, 2015.

6.3.2 BEMOSS Devices and Non-BEMOSS Devices

When a device is initially discovered, it is automatically marked as a BEMOSS device. If a device is not approved, it is marked as a Non-BEMOSS Device (NBD and removed from the system. The administrator-approved 'BEMOSS devices' can be monitored and controlled using BEMOSS.

6.4 User Management

User management is an important part of the UI platform. The user management is restricted to the UI platform to simplify the core platform processes. As discussed in Section 0, there are three user roles defined in the BEMOSS UI platform:

i) Administrator
ii) Zone Manager
iii) Tenant

6.4.1 Managing Users

When new users register to the system, their registration is pending approval from the administrator. These requests are reviewed in the 'Manage Users' page of the control panel. This page has a list of all active users and the users with pending registration requests. The administrator can approve or deny a registration request, view the roles of registered users, and can assign or modified their roles. Figure 6-9 shows the manager user page.

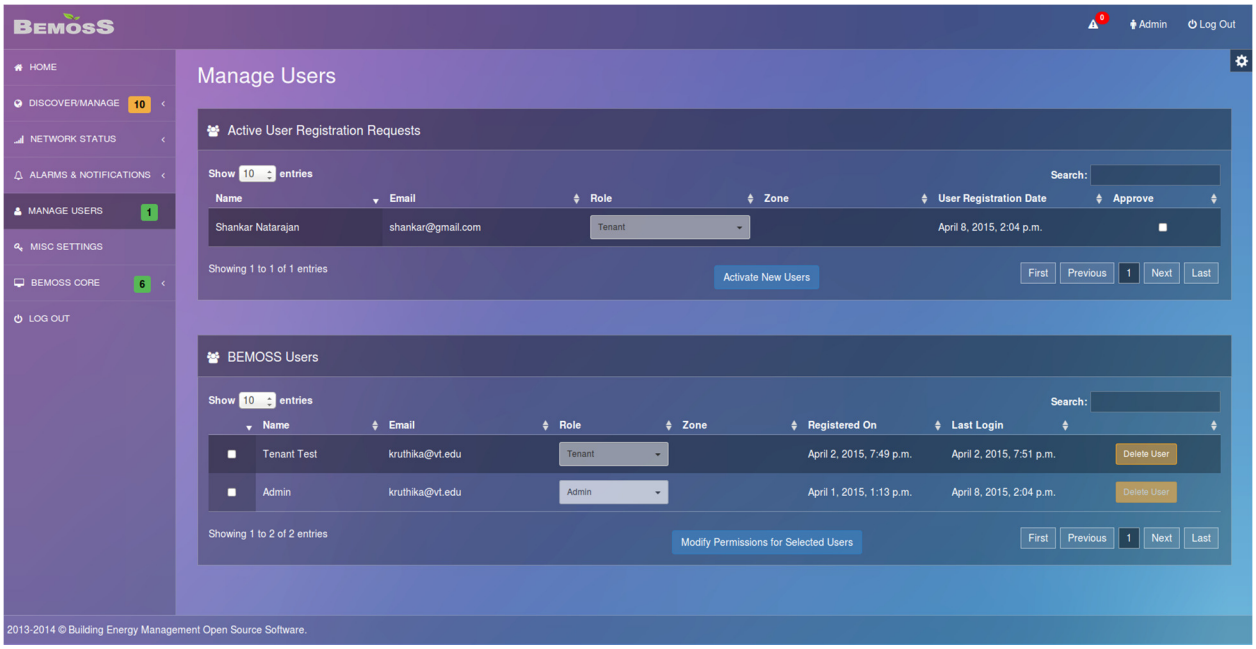


Figure 6-9 Manage users page
Source: http://www.bemoss.org. Used with permission, 2015.

6.4.2 User Registration

The user registration is a semi-automatic process in the proposed design, considering the closed user group that any specific installation of this platform is intended for. The user registration process creates an inactive user account. The registration process requires a username, password preference, an email to associate with the user account, and the person's name. Other information is optional. Figure 6-10 shows the user registration form.

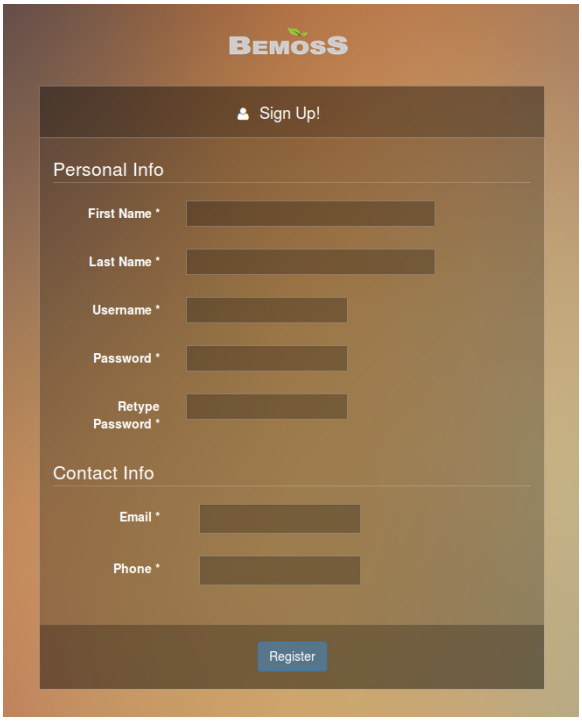


Figure 6-10 User registration form
Source: http://www.bemoss.org. Used with permission, 2015.

6.4.3 User Authorization

The UI platform ensures that the user is authenticated and authorized to access a page even before it begins processing the page load functions. After authentication and at every step in the page load, the platform ensures that only authorized pages are viewed.

6.5 Dynamic Views

The UI should be dynamic and responsive to show live device status. To achieve this, web sockets with push notifications are used. Web sockets are discussed in Section 6.8.4. Dynamic views also imply that pages should change based on user roles.

6.5.1 Unified Resource Locator (URL) Dispatcher

The UI platform configures URLs to map the page patterns to server callback functions. For every URL that is designed, a corresponding source function exists, which is mapped and run when the URL is typed into the browser address bar. The source function authenticates the user requesting the browser page and then collects necessary information for page display, and forwards it to the HTML page. All query information is gathered as POST information. This allows the URLs to be clean and consistent over different views and different users.

A clean URL as used in BEMOSS:

http://<host>/thermostats

An unclean URL that provides more information than necessary to the user:

http://<host>/thermostats?user=abcd&role=tenant&zone=999

For the unclean URL mentioned above, the user can easily change the request parameters passed to the application. The system becomes vulnerable with the user being exposed to such information.

BEMOSS UI platform internally handles all of this information without it being exposed to the user using session information. Figure 6-11 shows the flowchart describing how the UI platform processes the request from the user.

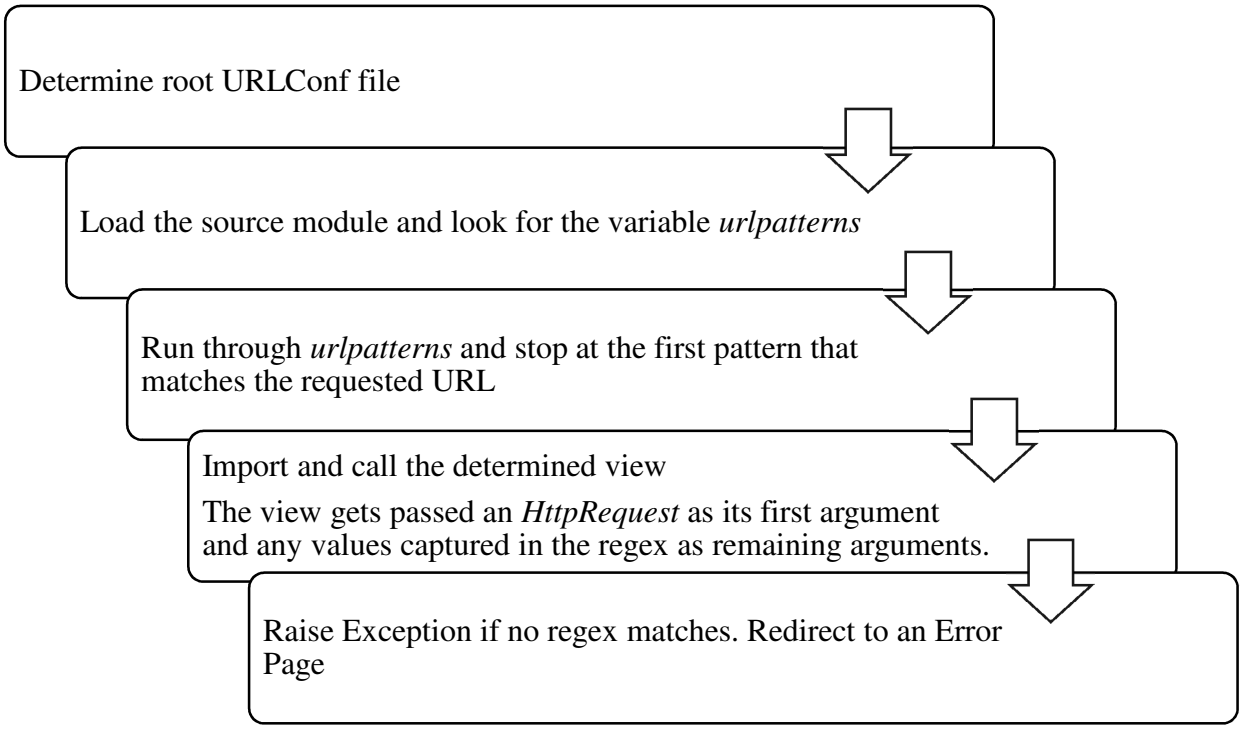


Figure 6-11 Request processing flowchart

6.5.2 Role-based Dynamic Views

Role-based dynamic views are created by implementing checker functions that are inserted as a template tag wherever required in the web page source code. In addition, at every page load, the user authentication was verified and redirected to the login page if the user was not authenticated. The user was redirected to the home page in case the user was not authorized to view that page. Implementing HTML template tags as pure server functions kept the web pages lightweight.

6.5.3 Error Abstraction

Error Abstraction is built for two kinds of errors. A 404 error, which is an error in the system functionality, and a 500 error, when the user tries to reach a page that, does not exist. Not providing a 404- and a 500- page view will expose critical information to the user such as error codes, types, session information, etc.

The UI platform has a separate view for a 404- and 500- errors. Any error in the system run, or an unavailable page redirects the user appropriately to the error abstraction pages. This shields the user from viewing unnecessary information on the web page.

6.5.4 Template Tags

Template tags are ensure that the load on the web server is minimized. Custom filters and template tags are Python functions that take one or two arguments:

i) The value of the variable (input) – usually a string, but not necessarily
ii) The value of the argument – usually a default value, can be empty.

Since template tags are comprehensive python functions and accessed directly using tags on the HTML pages, it is not necessary to transfer data through the Ajax request/response model. Any exception raised from a template filter is exposed as a server error, which is handled using a reasonable fall back value without throwing an error. Once filters are written, they are registered in the server and loaded into any HTML page that intends to use the filter. The template tags provide the developer with the flexibility to add more presentation level features without having to depend on the request/response view models.

6.6 Zone-based Control

In the proposed system, when a node is discovered by the core platform, it is assigned a zone and instantly populated in the UI platform. All background functions between the core and the node are encapsulated. Devices when first discovered are initially assigned to the core. They can then be moved to the appropriate zone, as required, by the administrator.

A new virtual zone can be added in the system using the ‘Add New Zone’ widget. It is called a ‘virtual zone’ because it is a logical zone to the user and may not physically exist. Once the zone is created, the administrator can go to the ‘Manage Devices’ page, and assign devices to the newly created virtual zone. Once devices are moved from one zone to another, the corresponding device agents are moved to that particular zone and restarted. Figure 6-12 shows the virtual zone creation window in the BEMOSS home page.

Figure 6-12 Virtual zone creation
Source: http://www.bemoss.org. Used with permission, 2015.

Once a new virtual zone is created, it is added to the system. A message is sent to the core platform indicating the new zone addition allowing the core to start the zone-related. The newly created virtual zone appears on the home page along with the core, and has the same options as the core. Figure 6-13 shows the new virtual zone in the home page.

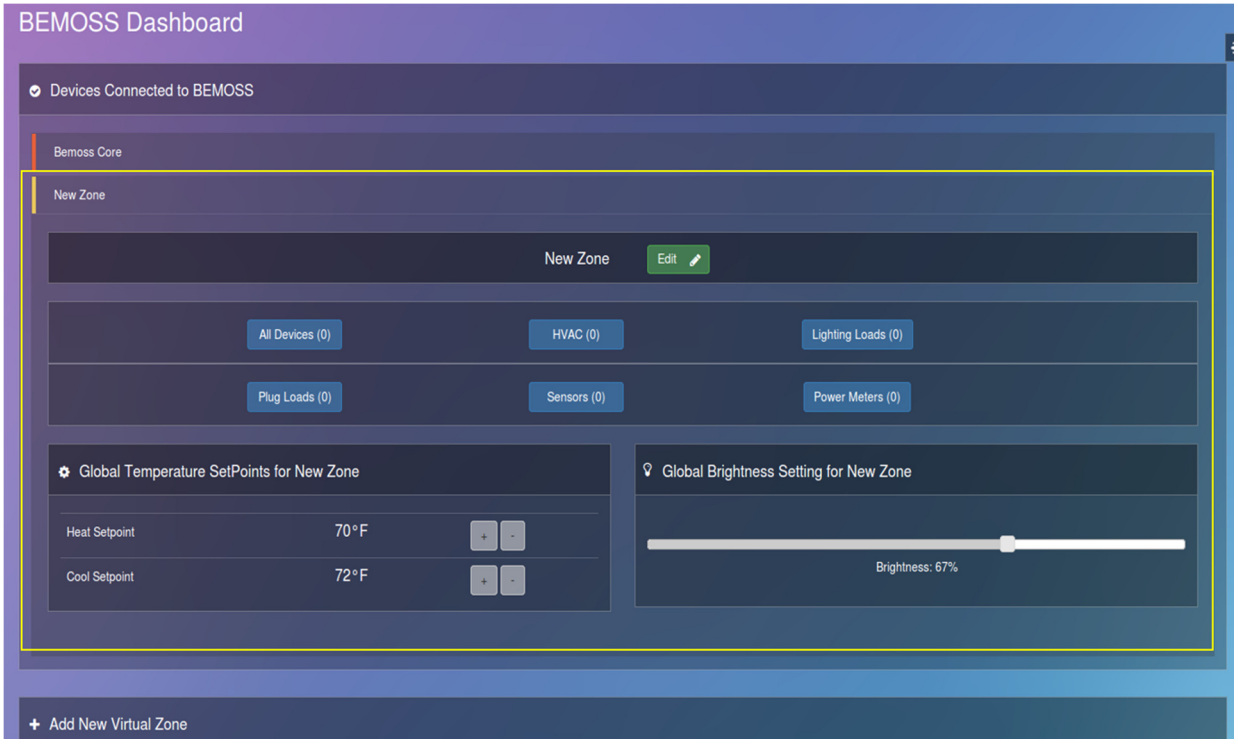


Figure 6-13 Virtual zone in the home screen
Source: http://www.bemoss.org. Used with permission, 2015.

6.7 Communication

The UI platform is a standalone interface and communicates with the core using the inter-process communication protocol. This protocol is implemented using an open source communication framework called ZeroMQ and is referred to as the Information Exchange Bus (IEB). The UI web pages also use web sockets for communicating with the core for push notifications.

6.7.1 Messaging Protocol

ZeroMQ is the basis of interaction between the core and the UI platform. A standard protocol for sending and receiving messages is established using named *topics*. The core and the UI can communicate with each other and with other platform services over a common message bus. ZeroMQ provides scalable multi-language communication that allows agents to post topics to, and allows other agents to subscribe to receive events they are interested in.

To enable the UI-core communication, an agent called *IEBSubscriber* is developed and UI-specific *topics* are designed for communication. The general message topic format is as follows:

sender/receiver/bemoss/zone_id/device_type/device_id/function/response

Table 6-1 shows a sample set of message exchanges between the core and the UI via the *IEBSubscriber*.

Table 6-1 Messaging formats for the UI-core communication

Message format	Message type
ui/agent/bemoss/991/thermostat/1TH234SFDGFS3/update	Request from the UI to the core platform to update a device status
agent/ui/bemoss/991/thermostat/1TH234SFDGFS3/update/response	Response from the core platform to the UI
ui/web/bemoss/thermostat/991/1TH234SFDGFS3/update/response	A topic from the UI platform to the webpage

The user requests for an update for any device by clicking the 'Submit' button. The submit action triggers a message to the core platform using the above discussed topics. The response message contains the current status and is received and displayed in the device page.

6.7.2 Message Queue Subscription

Communications in the BEMOSS platform is custom built to improve efficiency. Although ZeroMQ was used for this purpose in the core system, the customization makes it essential to communicate using the data exchange formats and topics so that all devices do not have to read all the messages.

As mentioned previously the *IEBSubscriber* manages the UI-core communication. Specific topics were created to exchange messages between the core and the UI Platform on the IEB. The messages are then exchanged using the appropriate topics and processed as per the algorithm shown below:

i) *IEBSubscriber* startup
ii) Subscribe to the topics specified in the decorators.
iii) Receives message
    a. Parse message
    b. Send it to the web sockets (or)
    c. Save information as necessary
    d. Exit process
iv) Listen to further messages

The *IEBSubscriber* agent is configured with the address of the message queue and the parsing algorithm before the message is fed into the IEB. The address configured is usually 'inter-process' if the core platform and the UI platform reside in the same server.

6.7.3 Web Sockets for HTML

Web sockets are used to push updates on device status and changes from the core to the UI as and when they occur, without constantly requesting the server for any change. With web sockets, a single request is used to establish a web socket connection and reuse the same connection from the client to the server, and the server to the client.

Web sockets [73] significantly reduce latency. The server does not need to wait for a request from the client. Similarly, the client can send messages to the server at any time. This single request significantly reduces latency over polling, which sends a request at intervals, regardless of whether messages are available. This real time saves communication bandwidth, CPU power, and latency. Web sockets follow the protocol shown in Figure 6-14.

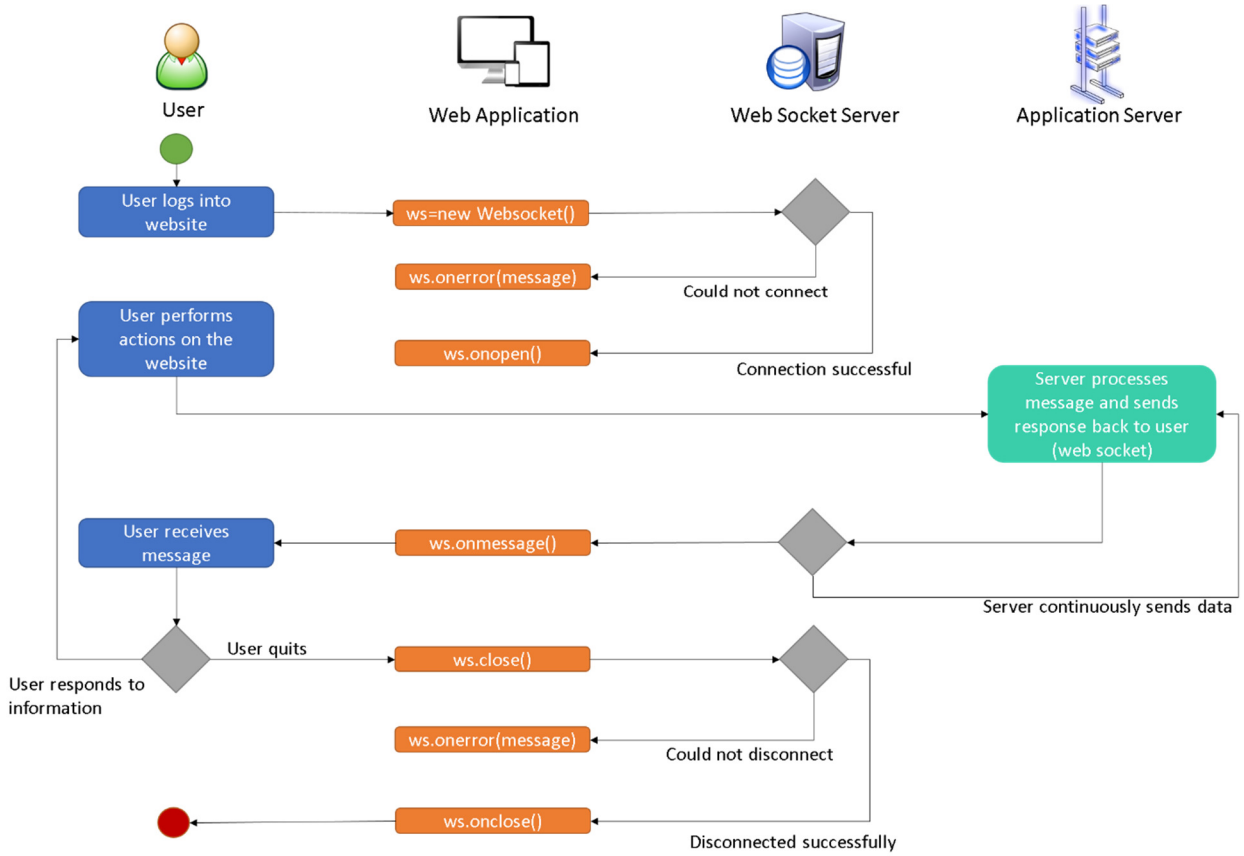


Figure 6-14 Web socket communication in BEMOSS

The browser sends a request to the server to switch protocols from HTTP to Web socket. The client replies through the upgrade header:

GET ws://www.bemoss.com/thermostat HTTP/1.1
Origin: http://www.bemoss.com
Cookie: __utma=99as
Connection: Upgrade
Host: bemoss.com
Sec-WebSocket-Key: uRovscZjNol/umbTt5uKmw==
Upgrade: websocket
Sec-WebSocket-Version: 13

If the server understands the web socket protocol, it agrees to switch through the upgrade header.

HTTP/1.1 101 WebSocket Protocol Handshake
Date: Fri, 10 Feb 2015 17:38:18 GMT
Connection: Upgrade
Server: Tornado
Upgrade: WebSocket

Access-Control-Allow-Origin: http://www.bemoss.com
Access-Control-Allow-Credentials: true
Sec-WebSocket-Accept: rLHCkw/SKsO9GAH/ZSFhBATDKrU=
Access-Control-Allow-Headers: content-type

At this point, the HTTP connection is replaced by the web socket connection over the same underlying TCP/IP connection. The web socket connection uses the same ports as HTTP (80) and HTTPS (443), by default.

Once established, Web socket data frames are sent back and forth between the client and the server in full-duplex mode. Both text and binary frames can be sent in either direction at the same time. The data is minimally framed with just two bytes. In the case of text frames, each frame starts with a 0x00 byte, ends with a 0xFF byte, and contains UTF-8 data in between. Web socket text frames use a terminator, while binary frames use a length prefix.

A new Web socket instance is created, providing the new object with a URL that represents the end-point for every device type. A *ws://* (for a Web socket) or a *wss://* (for a secure Web socket) connection is used.

A web socket connection is established by upgrading from the HTTP protocol to the web sockets protocol during the initial handshake between the client and the server. The connection is exposed via the "onmessage" and "send" functions defined by the web socket interface.

The web sockets are connected to the ZeroMQ message bus to ensure push notifications to the web pages. A web socket is created for every device type. Every device page has a web socket instance created establishing connection between the client (web page) and the server (user platform). Since the ZeroMQ message bus in this case does not follow the topic protocol established for data-exchange between the user platform and the core platform, a slightly different data-exchange mechanism is established.
Any message that is sent as a device status update message or an acknowledgement of a successful device status update is published to the IEB using the IEB Subscriber. The IEBSubscriber re-publishes the message on the message bus using a specific topic to communicate with the web sockets. The listeners on the web socket handlers are much broader to ensure simplicity. The listeners listen to topics of the following format.

sender/receiver/device_type

All of the messages corresponding to a device type are addressed with a more simplified listener, and the client side filters the messages as necessary. The topics used to communicate with the web sockets take the following definition:

sender/receiver/device_type/bemoss/zone_id/device_id/function/response

As an example,

ui/web/thermostat/bemoss/991/XXXX99999/update/response

Device type takes priority in this topic. The messages that come to IEB Subscriber are untouched, but sent on a different topic that is listened to by the web sockets. On receiving a message, the message is forwarded to the client (HTML page) using the 'send_message' function. The client side receives the message, filters the message that is intended for the corresponding device alone, discarding the rest.

Using web sockets reduces server load on both the user platform and the core platform. The latency is reduced considerably, and the system is comparably simpler. This feature is developed by connecting multiple technologies, including Tornado, ZeroMQ, Python, and JavaScript.

6.8 Live Streaming Data Visualization

BEMOSS cannot be complete without the ability to view the past data point information on a holistic graph. Data visualization for a building is the display of a rich set of variables and parameters that managers can use to verify the energy savings and identify malfunctions of building equipment or problems with operating strategies. Effective data visualization depends on having graphic presentation formats that reveal the data relevant to the building's performance [74].

BEMOSS achieves this visualization real-time by using several open source tools. An algorithm is developed to fetch data periodically from the database. In addition to projecting live streaming data on the user platform, the platform can also display graphs and charts for a specific time intervals.

6.8.1 Interface with the Time-Series Database

While showing all data since the beginning might be an intensive task for any processor, it is important that the administrator or a user be given the option to view the past usage information to get some insight and for decision making in certain demand response situations. For this purpose, a time-series database was used.

Every data point relevant to a device is periodically queried by the UI platform to update the graphs in real-time. Data points are stored in the time-series database in a tree-like structure using the sMAP driver functionality by the core platform as shown in Figure 6-15. The structure format is stored as metadata.

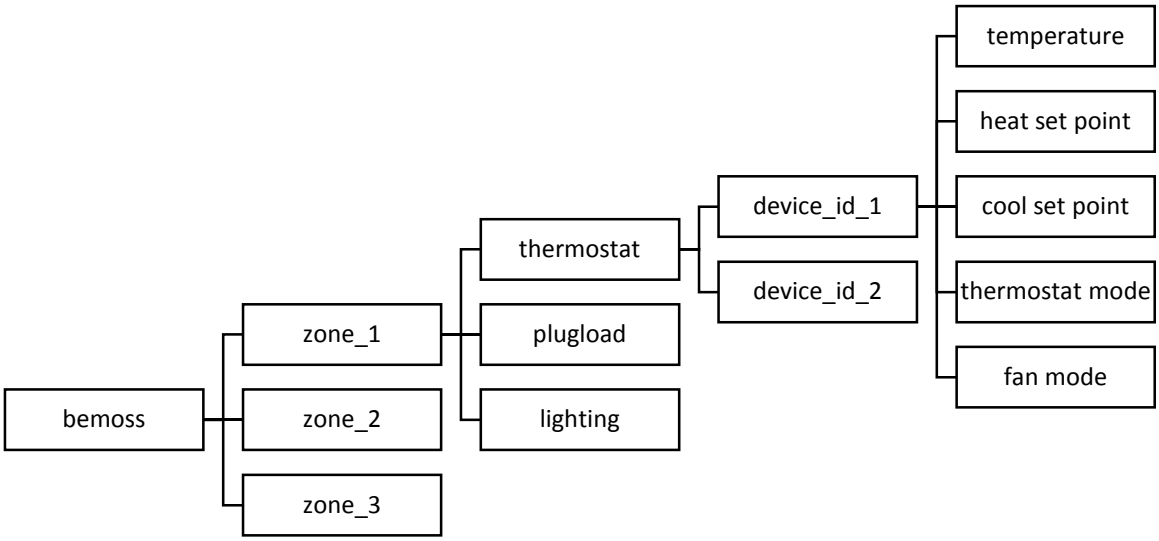


Figure 6-15 Data Storage Structure

The UI platform queries the metadata to obtain the root node for the specific device, obtains the tree structure, and periodically queries the time-series database to update the visualizations on the user platform.

Live Data Streaming

Data is sanitized and displayed on a rich graphical interface with a variety of options. All data points correspond to a particular device on the visualizations page, where the user can compare and analyze data points over different times. The live data stream updates every minute showing the user the latest information about a device, graphically. The following process is implemented to generate live visualization in the forms of charts:

- The metadata required for generating the live stream is stored in the sMAP metadata storage, which is a relational database. This database is queried to obtain information about the particular device for which data visualization is required.
- The HTTP URI is generated.
- This URI is queried to obtain the time-series data.

The process is simplified to obtain data directly using the HTTP URI using periodic queries.

6.8.2 Customized Visualizations

In the UI platform, users can filter the data based on date and time. This feature presents concise information about the system status at any point in time. Normally, data for the last 24 hours is visible and continuously updated. However, this setting can be customized by the user to show a longer time period.

For example, a thermostat may have data points, like temperature, heat set point, cool set point, thermostat mode, fan mode, etc., and a user might want to view certain data points among these at the same time in the same graph to get a perspective on device usage statistics. The visualizations interface shows all data points plotted on the graph, and the user can disable some of these data points to get a singular picture of one data points or more.

6.8.3 Charts and Statistics

Visualizations are the most important factor that will ultimately help in energy savings and reduced power consumption. Visualization is provided using charts and statistics for each device. A user can navigate to this page using the side navigation bar or device dashboard page. The UI design organizes the interface elements in the charts and statistics page consistently for every device type.

Charts are placed in the center of the page, and the data points to choose from appear on the right in a widget. Just below the data points widgets, are the 'Auto Update' and 'Export Data' widgets. Clicking on 'Auto Update' enables live streaming and data is retrieved and updated every minute.

The charts also show a color-coded legend to identify the data points. 'Export Data' widget exports the time-series data in the time and value format for all the data points mentioned in the Data Points widget. Figure 6-16 shows an example chart from the thermostat device page.

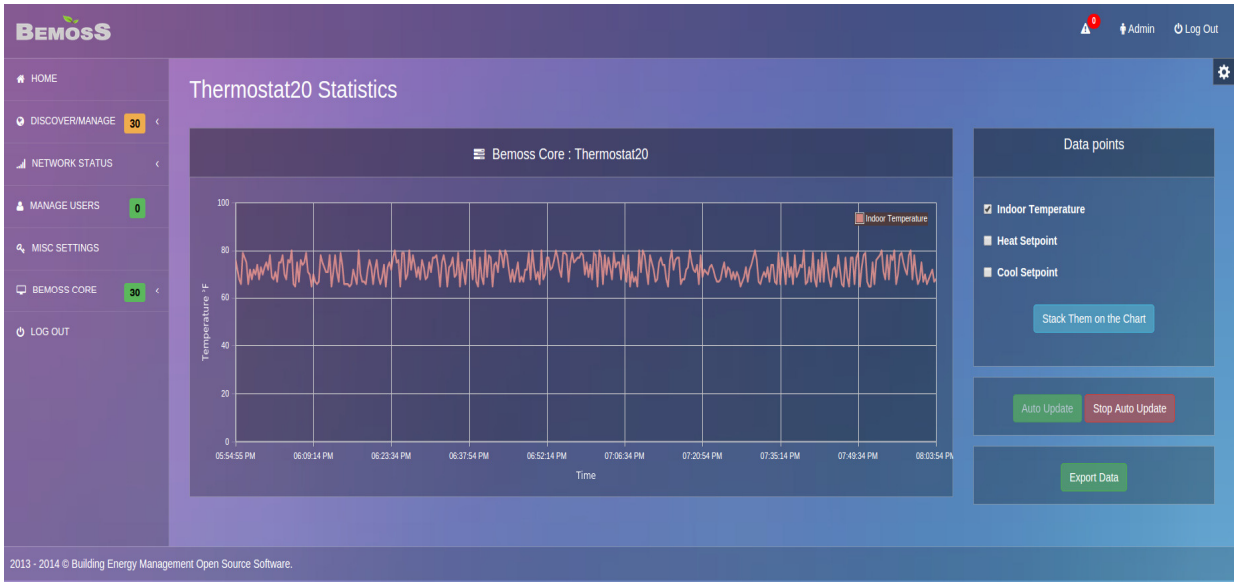


Figure 6-16 Chart for thermostat - indoor temperature
Source: http://www.bemoss.org. Used with permission, 2015.

It is also possible to stack more than one data point on the chart for comparison, as shown in Figure 6-17.

Figure 6-17 Stacked chart for thermostat - indoor temperature and heat set point
Source: http://www.bemoss.org. Used with permission, 2015.

6.9 Monitor and Control Interface

The unique features of the UI monitor and control interface are elaborated below.

6.9.1 Device Identification

The BEMOSS device approval process is unique in identifying devices to ensure the right device being added to the system. As part of device approval process, device identification is necessary. Device Identification is defined as the process of uniquely identifying a device in the building using specific control features of the device. For example, identifying a Philips Hue Bridge and all the light loads connected to it by turning each of the lights on and perform a color change loop for each light.

In the proposed design, the 'Identify Device' button is made available on the UI dashboard for every device that can identified, and in the Discovery Devices page beside every newly discovered device. Typically, these are the two areas where a building engineer would prefer to identify a device. On clicking the 'Identify Device' button, a message is sent to the core requesting the identification of the particular device. The core platform takes care of delegating the message to the device thus providing the necessary Identification functionality.

6.9.2 Global Set Points

A BEM system can have devices from a few tens to hundreds. Setting control parameters for every single device is time consuming and inefficient. To overcome this and also ensure that settings are fairly uniform throughout the building, the concept of global set points is introduced.

Using global set points, an administrator can set device set points for all devices in a building or a particular zone. The global settings can be overridden at a device level, if needed.

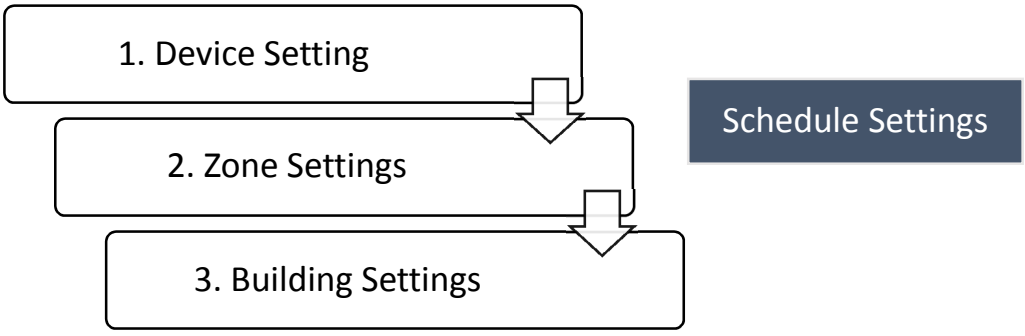


Figure 6-18 Priority of settings for devices

The priority levels of the settings are shown in Figure 6-18. Device settings get the highest priority followed by zone settings and building settings. Apart from these settings, device parameters can be modified using scheduling. A schedule parameter for a device takes the highest priority with respect to time. If a schedule parameter is activated, and no other control is in effect, the schedule settings take effect. However, after the schedule is operational, if a device setting is modified, device settings override the schedule.

Global set points functionality is available in the home page and shown in Figure 6-19. Currently, global set points are available for certain basic functionality –heat and cool set points and brightness levels for lighting controllers. Everytime the global set points are enabled, a message is sent to the core platform to enable the required global settings. The core platform ensures that the device settings are altered as per the global setting priority.

As per design principles described earlier, colors and fonts and used to capture the user's attention. Heat and cool set point values, following by '+' and '-' buttons are based on the proximity and spatial layout principles. Relationships between these items are intuitively identified since they are close together with each other.

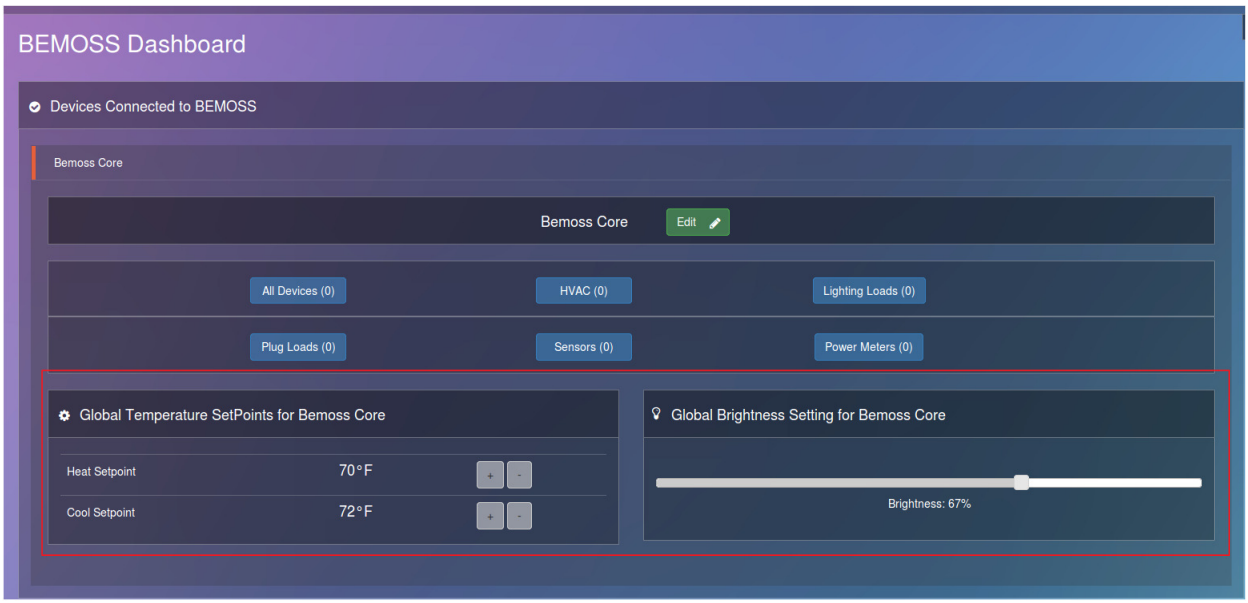


Figure 6-19 Global set points for zone - available to the administrator
Source: http://www.bemoss.org. Used with permission, 2015.

6.9.3 Monitor

An active monitoring system is essential for efficient building operations. To enable active monitoring, the system should display status of devices and controllers in real time.

In the proposed design, device status is updated on the user interface every time its status changes or is updated by user. Monitoring the system can be at a device level or at the zone level showing the overall system settings and key device monitoring parameters.

A thermostat device dashboard, for example, will show the outdoor temperature and humidity, indoor temperature, heat and cool set points, current thermostat mode and fan mode, and an option to access the scheduler from device page. The same page can be also control the device. Parameters are displayed in individual widgets in clear and bold fonts and in some cases, using graphics and meters (current consumption, power consumption, etc.). The status information for the monitor dashboard is picked up from either the database (on page load) or a live update (every time after page load) from the web sockets.

A zone-level dashboard page shows all online devices in individual widgets, grouped by device type. This widget also shows key device parameters to get a quick overview of the status. For example, a thermostat widget on the zone dashboard shows the current indoor temperature, the current thermostat mode, and the set point.

6.9.4 Control

Besides monitoring, the dashboard can also be used to control devices. Every device has a dashboard for monitoring and control. Devices can be monitored with all parameters displayed,

and can be controlled using the control buttons that are interspersed in context with the monitor elements of the dashboard.

For example, a thermostat has a + and – button beside the heat and cool set points, where the user can modify the set point. If the thermostat mode is set to ‘heat’, then the heat set points are enabled for modification. If the thermostat mode is set to ‘cool’ then the cool set points are enabled. If the thermostat mode is set to ‘auto’ or ‘off’, then both heat and cool set points are disabled. There is no need for a user to manually type in any command.

Once the user clicks the required buttons and clicks submit, the UI sends a JSON formatted message to the core platform, requesting a change in device status. The message is processed, and using the communications protocol, device status is updated based on the user inputs. Once an acknowledgement from the device is received, a notification is displayed. This notification is pushed from the server and displayed as a floating notification in the right bottom corner of the user interface. Such a notification perspective saves space on the screen and attracts the user’s attention to read the notification.

The control interface for different device types varies based on the need of the system. For sensors and power meters, it is more of a read dashboard than write. The power meter dashboard provides the current power consumption readings to the user, whereas a light sensor could provide information about the current light intensity in the room. All dashboard provide the unit of measurement for every data point displayed.

Figure 6-20 shows the process between the user submitting the new values to the server and server notifying the user of the change. The complete process takes about less than a second to update a device with the new input and may vary based on network speed and device latency.

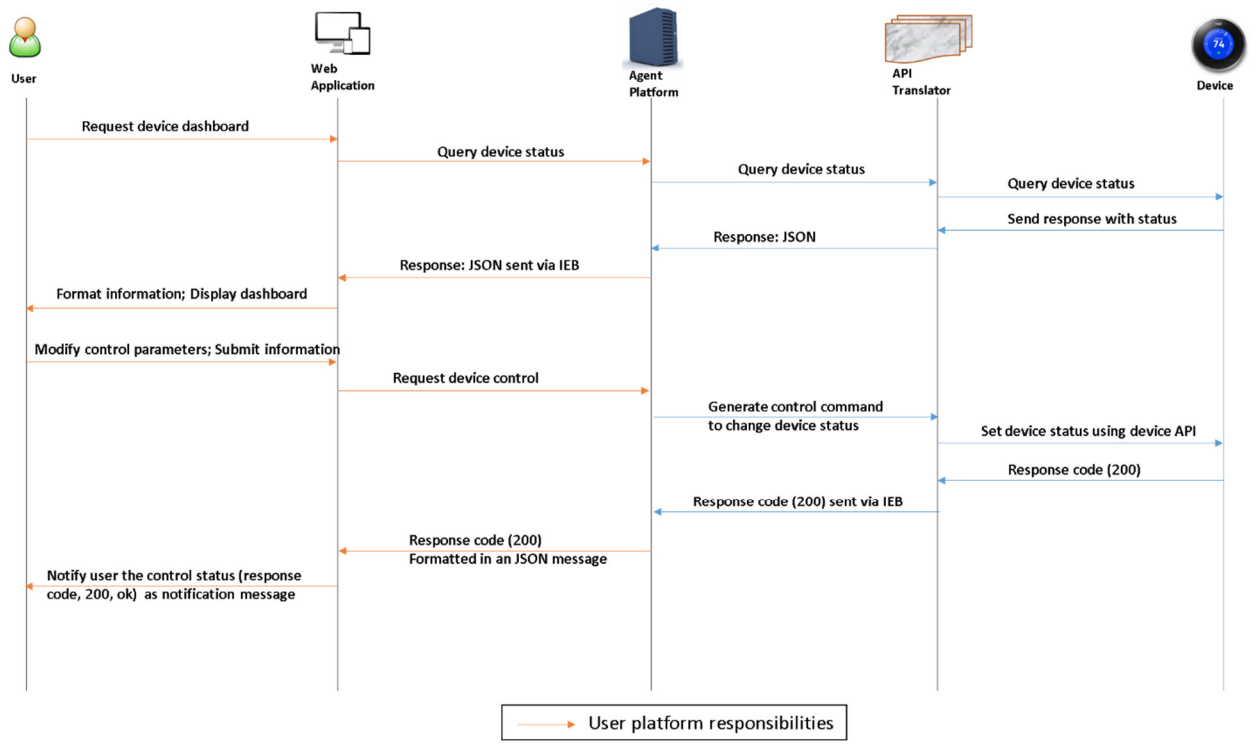


Figure 6-20 Device control process

6.9.5 Device Information

Multiple devices from different manufacturers are connected to BEMOSS. It is imperative for the building operator to get a quick overview of the device manufacturer, model, MAC address, and other basic information.

In the proposed design, the zone/building dashboard has a 'View/Edit Information' bar in every device widget. This bar opens a modal, which provides metadata, including manufacturer, model number, MAC address, commercial name, etc. It also shows a nickname, which can be modified as needed.

A device information modal is shown in Figure 6-21. This modal opens up as a translucent dialog on top of the existing page, showing all information in a table. The nickname can be modified only after clicking the edit button doing which opens a text box to edit the nickname. The edit button is available only to the administrator who has the permission to modify nicknames.

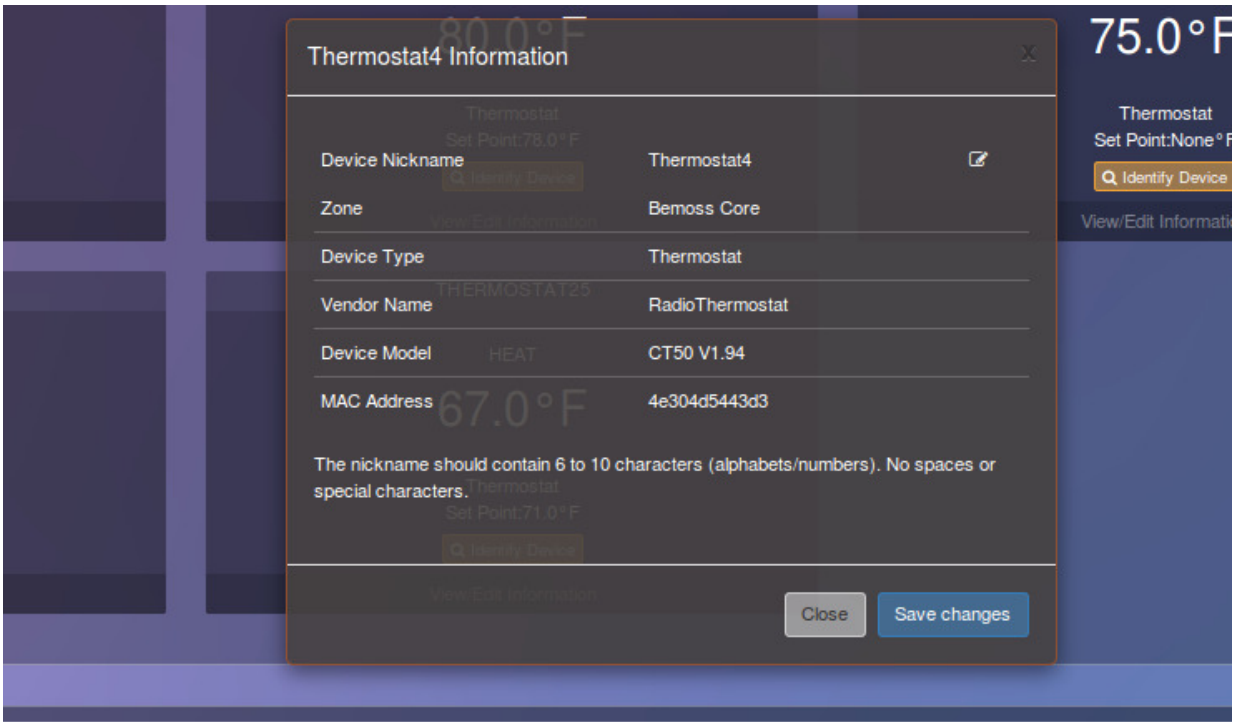


Figure 6-21 Device information modal
Source: http://www.bemoss.org. Used with permission, 2015.

6.9.6 Device Nicknames

In a BEM system, it may be a common practice to have multiple devices of the same type from the same manufacturer. In such cases, the MAC address is usually the most definitive way to differentiate devices. However, MAC address is not human-friendly involving alphanumeric characters that are hard to remember and identify a device by.

In the proposed design, the concept of Device Nicknames is introduced, which involves naming a device, an easy human-friendly name that the administrator or a user alike, can use to identify a device uniquely. This nickname is derived from the device type and the order in which it was first discovered in the building. For example, a Google Nest Thermostat has 'NestThermostat10' as the machine-assigned device name where '10' stands for the 10th Nest thermostat discovered in the building. The nickname can be changed to something more sensible and human-readable by the administrator.

In the modal 'View/Edit Information', an option to modify the nickname is provided. It is not immediately available to the administrator. The administrator has to make available the text field to edit the nickname by clicking on a button in the modal. This simple feature of not having the option to immediately change the nickname has two benefits – security, and discouraging the administrator from frequently changing the nickname. Nicknames are a definitive way to identify a particular device in the zone or a building, but frequently changing the nickname may dilute this very benefit.

From a developer point of view, the nickname is a name tag associated with the device. It is not used for any other machine-level interaction with the device. No communication or business logic utilizes nicknames. They are solely for the benefit of the user to interact with the system easily.

6.9.7 Customized Monitor and Control Interface

Monitor and control interfaces are customized for every device type, based on specific device features. If the data points for two thermostats are different, they are treated as two separate device types, thus creating two different but closely related interfaces. For example, the Google Nest thermostat (shown in Figure 6-22) features a battery level option in its API, and calls for an additional widget on the Google Nest dashboard, thus differentiating it from a regular thermostat dashboard (shown in Figure 6-23).

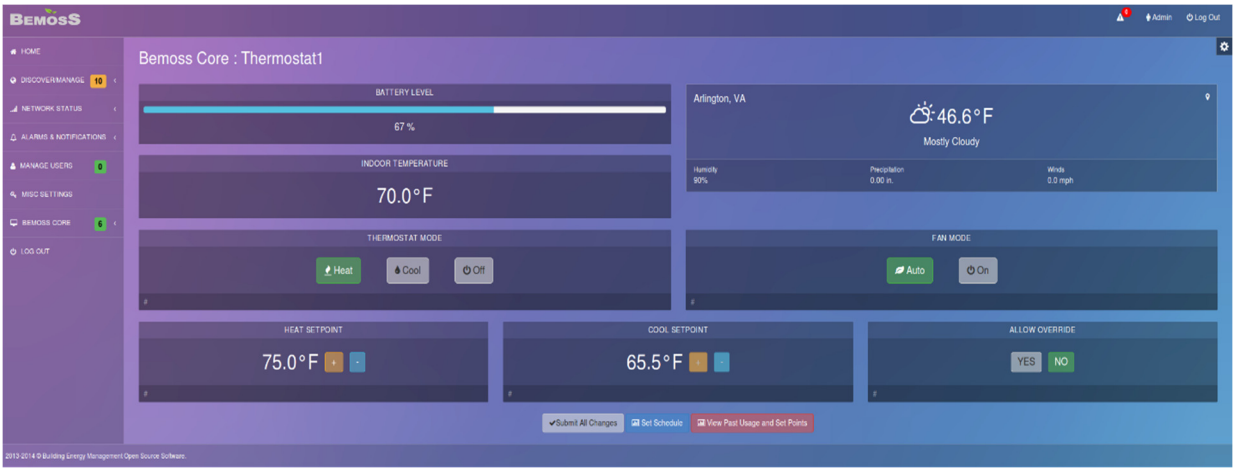


Figure 6-22 Google nest dashboard
Source: http://www.bemoss.org. Used with permission, 2015.

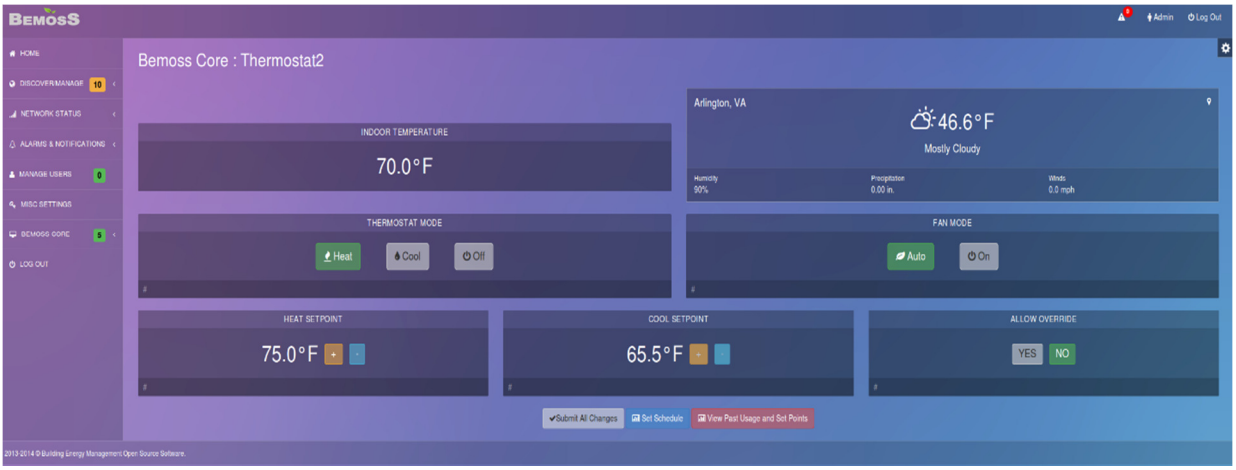


Figure 6-23 Radio thermostat dashboard
Source: http://www.bemoss.org. Used with permission, 2015.

For role-based access, submit changes and set schedule links have been removed for tenants and zone-managers who do not manage a particular zone. Any control buttons, and text fields that require administrator inputs are either disabled or removed from the tenant view.

In addition to having the regular thermostat elements in the dashboard page, the outdoor temperature and humidity information is also included, since this information is needed for changing settings in a thermostat. Additional functionalities, like 'Allow Override' in this case, are also included, when the core platform requires some user input for providing certain functionality.

Figure 6-24 shows the tenant view of a thermostat dashboard. The 'Submit Changes' and 'Set Schedule' options are removed and other control buttons are disabled although the user can view them to get device status information.

Monitor and control interfaces are developed for all devices listed in Table 4-2 of Chapter 5.

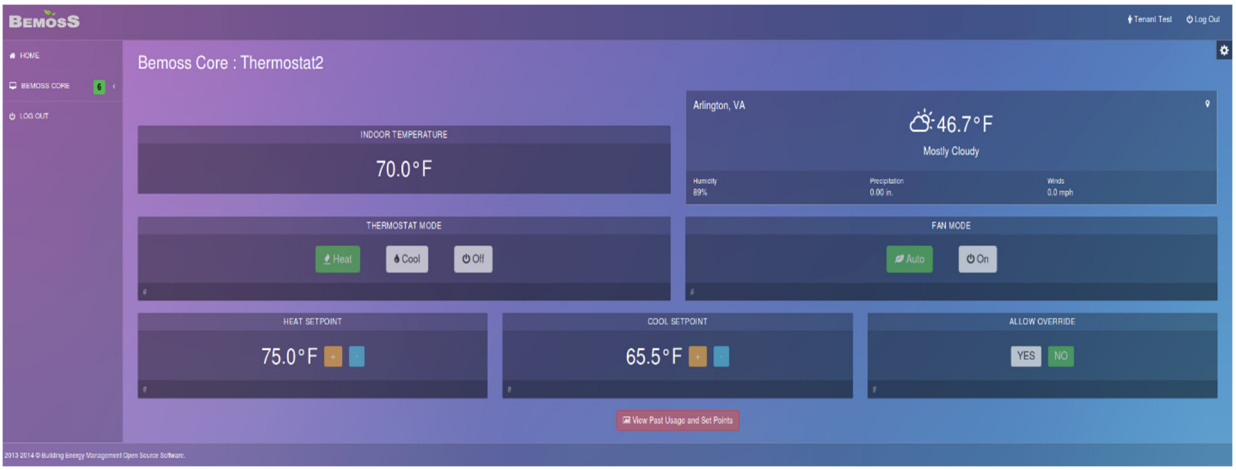


Figure 6-24 Tenant view of a regular thermostat dashboard
Source: http://www.bemoss.org. Used with permission, 2015.

6.9.8 Design of Monitor and Control Interface Elements

The monitor and control interface is designed from the user perspective. Several user interface guidelines were referred to and design principles were verified before implementing the interface. For example, in the thermostat interface, the proximity principle is verified by ensuring that the thermostat mode and the fan mode options are beside each other. The thermostat set point options for heat and cool are available right below the thermostat mode option to give the user the visual continuation and correlation impact while viewing the dashboard, thus making it faster to navigate through the interface

Usage of common elements like buttons reduces the need for user to understand any new interface element. Use of reasonable colors like green to signify selection is quite common in any web interface. Similarly, orange is used to signify increase in temperature to convey that it is warmer, and blue is used to specify that it is cooler. This is intuitive and aligns with the user thought process.

The outdoor temperature widget gives information similar to that of a regular weather information widget.

6.10 Control Scheduling

Any BEM system should support scheduling devices for certain operation. Setting temperature points and on/off programs every day may not be feasible. This brings about the concept of pre-scheduling, where the administrator can schedule the time and the setting for a particular device.

The UI platform can automatically modify device settings as per the set schedule. An important aspect of scheduling is differentiating the concept of weekdays and holidays. A regular weekday in the United States is Monday to Friday and the regular working hours are from 8 AM to 5 PM. This implies regular weekday temperature settings should be activated by around 7 AM to get the temperature stabilized in the building by 8 AM. Similar weeknight temperatures can be set to start functioning at about 6 PM in the evening.

In the proposed UI platform design, scheduling is available as an inbuilt application that is first activated when the administrator requests the schedule page. Two different schedule sets, one for weekdays and one for holidays, are automatically generated the first time a device schedule page is activated. The schedule is stored in the system and loads every time the user requests the schedule.

Schedules can be enabled at a device level or at the zone level. Device level schedules are more relevant in the current context, since different locations in the building would have different requirements. These two schedules are activated by default and it is left to the administrator to decide further action.

The default schedule is set for a day and evening period. Period 1 starts at 7 AM in the morning and ends at 6:59 PM in the evening. Period 2 covers the remaining 12-hour period. This set up is basic and gives the administrator a quick insight into scheduling features in the system. Basic scheduling is the same for both holidays and weekdays. The holiday dates can be set in the General Settings page. The holiday settings will be enabled for all holidays listed in the holiday settings.

6.10.1 Timing and Control

The scheduler runs continuously once enabled. If the scheduler settings are activated, and device status is overridden by the administrator, the latest setting will take priority. Settings vary based on device type. The holiday scheduler should also be maintained similar to the weekday scheduler, inserting whatever period the administrator intends to have the settings for.

The settings depend on device type. For example, a thermostat will have heat and cool set point schedules. Heat and cool set points can be set by altering temperature ranges in the page. By default, both heat and cool set points can be set in the schedule page, and based on the current setting, the corresponding setting will take effect. For example, if the thermostat is set to HEAT mode, the heat set point provided in the schedule for that time period will take effect.

Similarly, for a lighting load/plug load controller with simple ON/OFF function, the corresponding ON/OFF behavior will be available in the device page for scheduling. If a device has color changing functions or dimming control, appropriate controllers are available on the schedule page.

Implementation of the scheduler is discussed in subsequent sections.

6.10.2 Schedule Nicknames

Having multiple timing and control settings for each day might not make it easy for an administrator to be intuitively guided. A nickname, say 'Morning' for a time period between 8:00 AM to 12:00 PM might be appropriate and helps easy identification.

Similar to device nickname, the concept of nicknames is also available for the Schedule periods based on the time settings for control. For every period that is added a default period nickname is added, which can be modified by the administrator. This concept of nicknaming each period helps quick identification of the time and settings. The administrator can choose to leave it to the default value.

6.10.3 Scheduler

Scheduling a device for a certain operating is one of the most important advantages of a control system for energy management. Although learning algorithms greatly eliminate the need for scheduling, scheduling is a definitive way of knowing that building functions as per the user set predefined building environment.

Schedulers in the proposed design are predefined, and are activated the first time a user navigates to the scheduler page using the 'Set Schedule' navigation element. The schedule is designed for every device type that can accept a schedule, and comes predefined with a general schedule. The page loads with a predefined schedule.

Figure 6-25 shows the set schedule page for a thermostat.

i) General Schedule Interface Elements

The 'Activate Schedules' widget allows the administrator to disable one or more schedules if necessary. By default, both schedules are enabled. Disabling a schedule might be necessary in case the building requires manual intervention. Intuitively styled 'Radio buttons' are used for the user to select the schedules.

The 'Currently Active Schedules' widget shows the currently active schedules in the list.

The main schedule activity occurs in the 'Set Schedule' widget. The elements in the widget are highly customizable and dynamic. The interactive interfacing with customization capabilities make it easy and flexible for a user to set schedule without need for any prior training. For a thermostat, heat and cool set point schedules are enabled. The set point based

on the current thermostat mode is picked up and enabled. For the daily scheduler, all days from Sunday to Saturday are listed and are allowed scheduling.

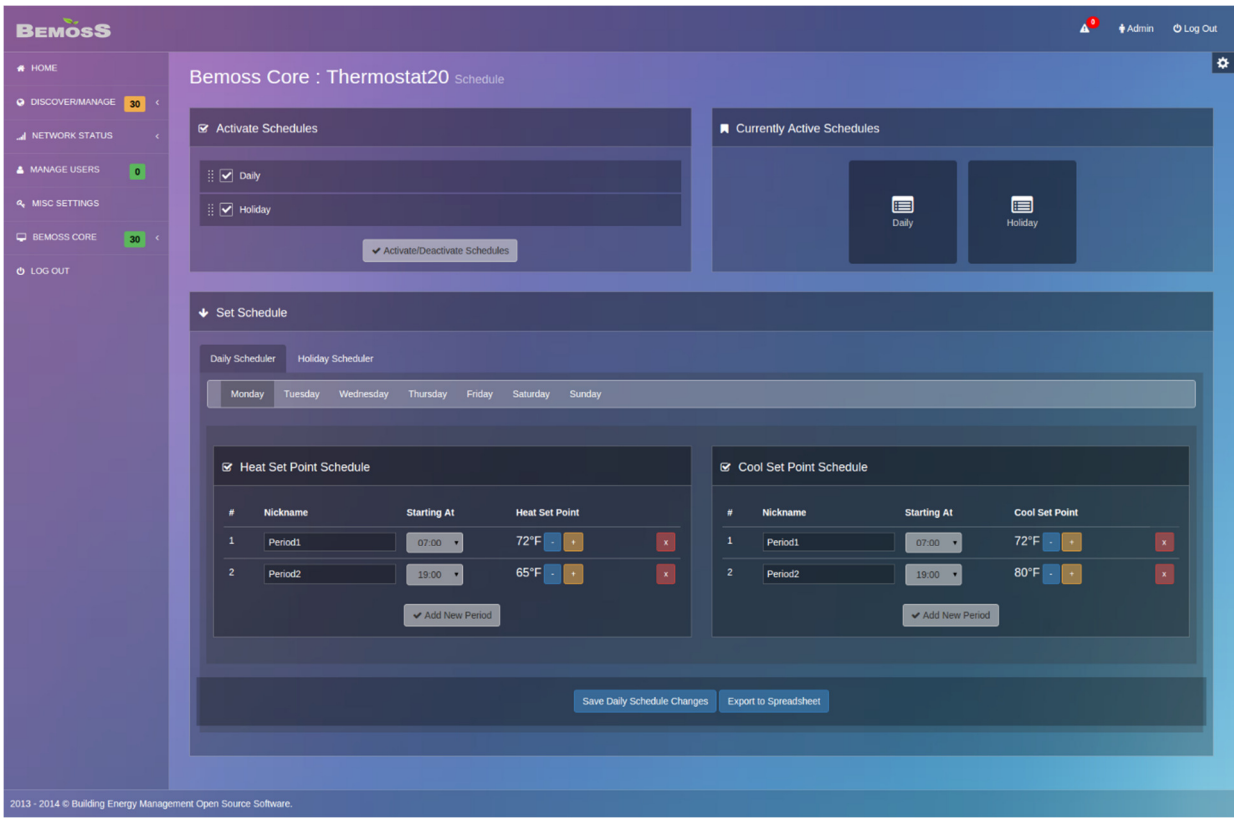


Figure 6-25 Set Schedule page – thermostat
Source: http://www.bemoss.org. Used with permission, 2015.

The ‘Starting At’ column specifies the start time for the beginning of the period. The ‘Starting At’ time chosen in any one of the periods is disabled from being selected in any new additional periods. In Figure 6-26, the dropdown is of this column element is shown, and 07:00 is disabled, because it is already chosen. This feature avoids duplication in schedules.

In the proposed design, a user can add up to seven periods. More than seven periods is disallowed because of its unrealistic nature.

A sequential number is displayed for every row, followed by the nickname for the schedule period. The ‘X’ button is used to remove a period. The ‘Add New Period’ button adds a new row to the schedule period list. Each of the schedulers, e.g., ‘Daily’ and ‘Holiday’ have a separate ‘Save’ button and updates the schedule file separately.

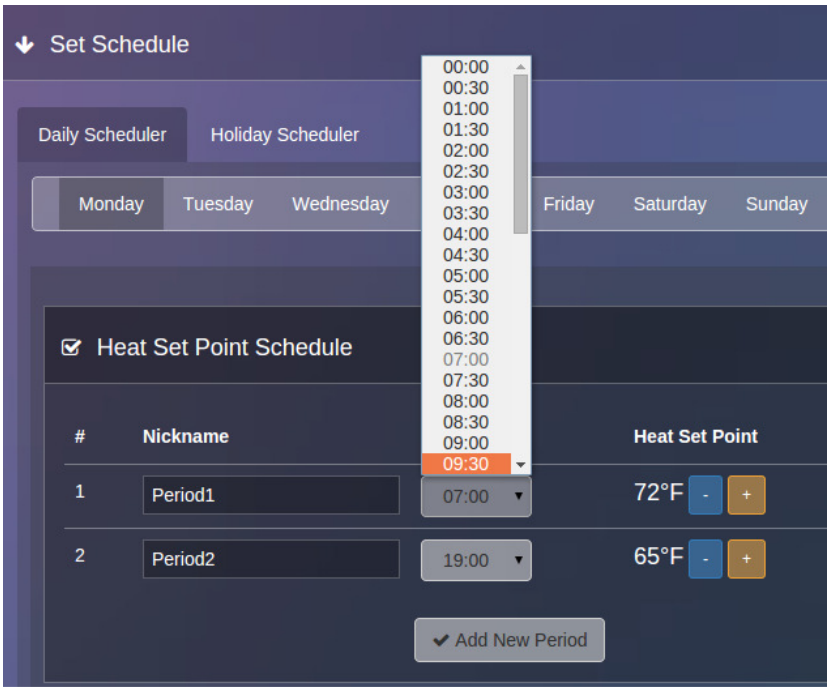


Figure 6-26 Drop down for choosing the 'Starting At' time
Source: http://www.bemoss.org. Used with permission, 2015.

ii) Thermostat Scheduler Interface Elements

The 'Set Point' column is similar to the thermostat dashboard element, and goes from 35 °F to 95 °F.

iii) Lighting Scheduler Interface Elements

The lighting scheduler has elements corresponding to the functions available in the lighting load controller that is being scheduled. Figure 6-27 shows the potential controls for a lighting load controller.

A regular step dim ballast will have On/Off function and step dim function with values 0, 50, and 100. The brightness function can be modified using a slider, which has been provided the step option with 0, 50, and 100. In a fully functional lighting controller like the Philips Hue, which can change colors, a color chooser is used. The slider is disabled if the user chooses OFF mode. When the ON mode is selected, the slider is at a 100% and the user can modify the brightness level. This feature is intuitive to the user as the mode is changed from ON to OFF or vice versa.

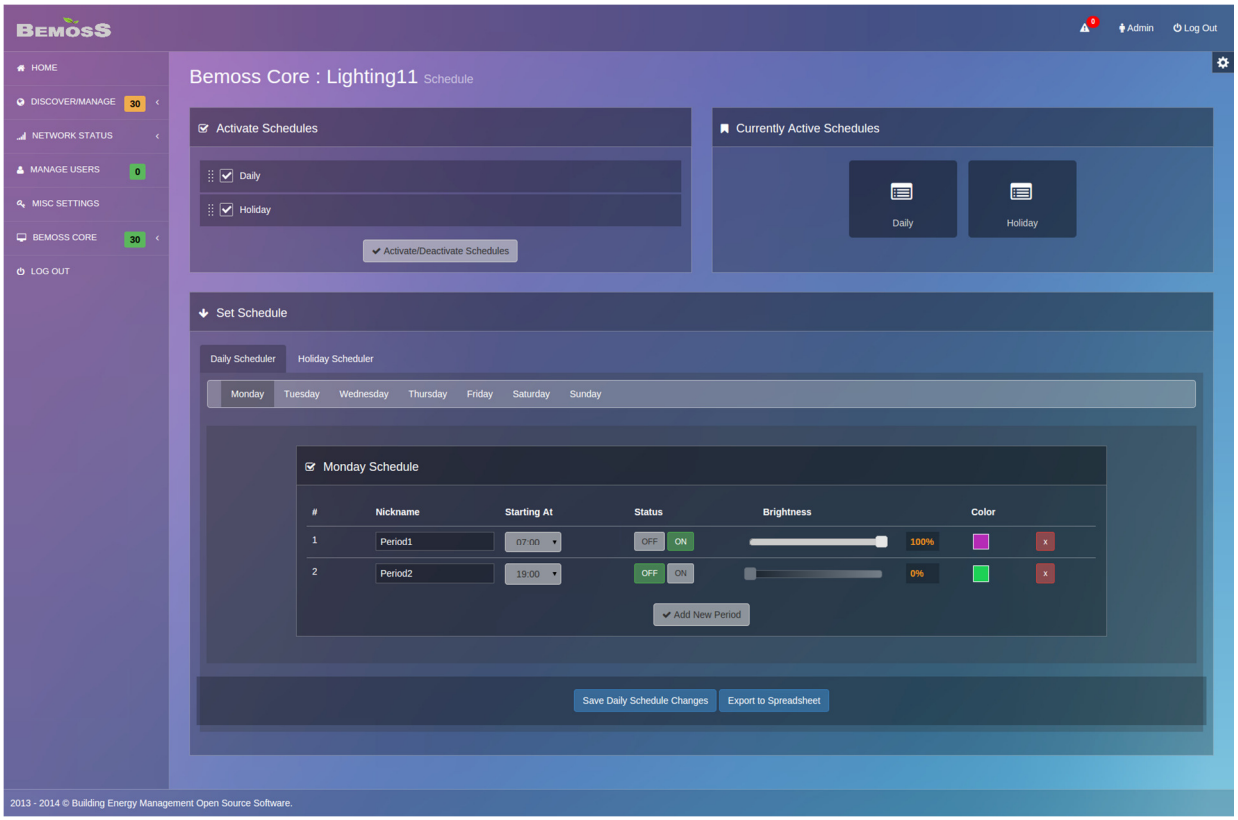


Figure 6-27 Set Schedule page - lighting load controller
Source: http://www.bemoss.org. Used with permission, 2015.

The color chooser, shown in Figure 6-28 provides RGB values which are used to activate the settings. An option to allow manual entry of the RGB value or the HEX value is available. In the server, RGB values are retrieved and communicated to the core platform. The color chooser is script based, and all colors with the RGB range of 0 to 255 can be chosen.

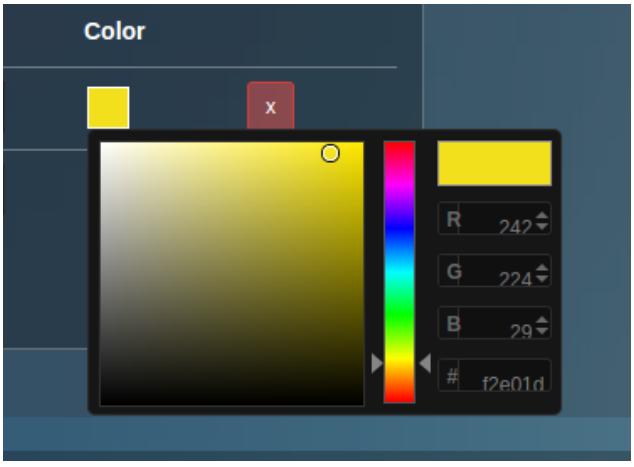


Figure 6-28 Color chooser
Source: http://www.bemoss.org. Used with permission, 2015.

iv) Plug Load Scheduler Interface Elements

The plug load scheduler has just the extra ON/OFF functionality as shown in Figure 6-29.

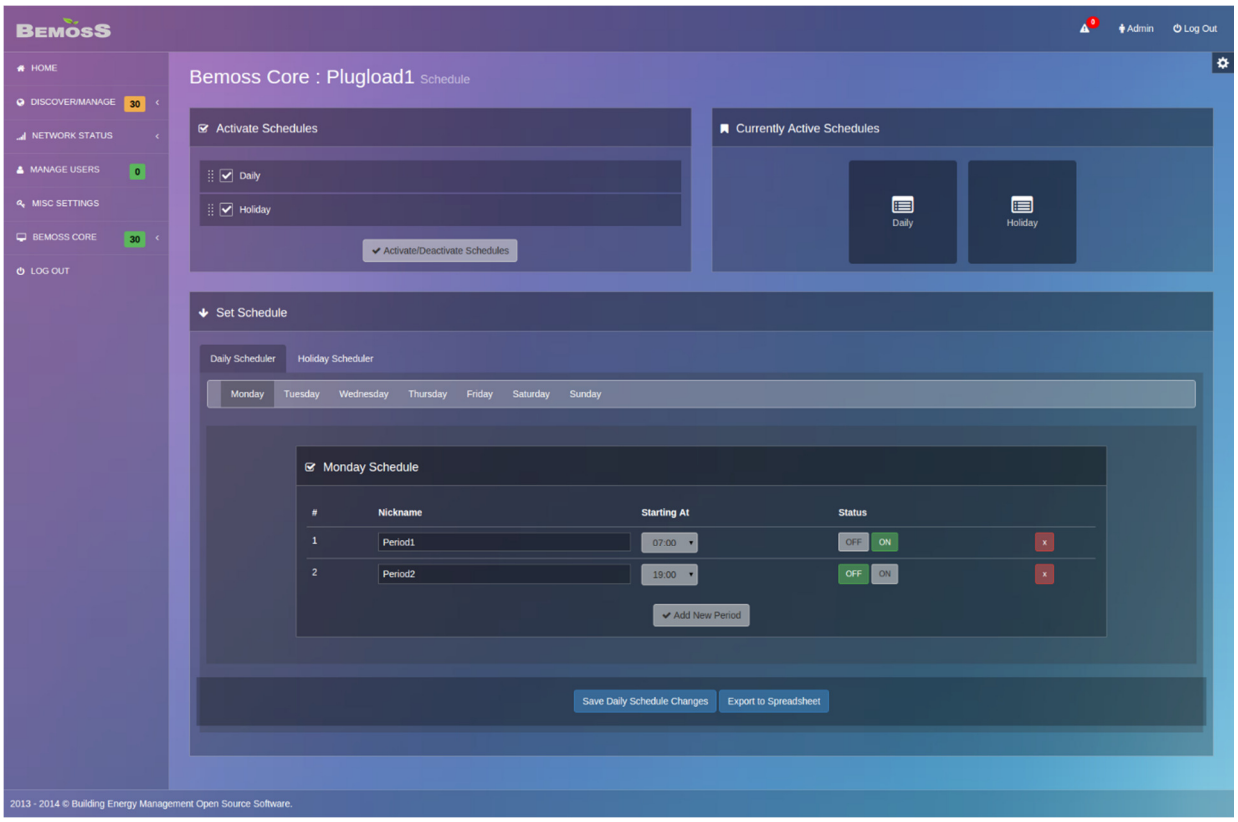


Figure 6-29 Set schedule page – plug load controller
Source: http://www.bemoss.org. Used with permission, 2015.

6.11 Alarms and Notifications

Alarms and notifications are an important feature of any BEM system. Configurable alarms enable the administrator to prioritize the most important events in a building requiring action. In addition to notifying the user currently logged in to the system, the administrator or the assigned person should be intimated by registered means of communication based on the event priority.

Notifications should be available through multiple means instead of just through the system. It is also essential to log the events for later review. A good alarm system should neither notify every small little event that occurs in the system nor ignore critical alarms. A logical balance is to be sought to maintain a functional BEM system. By customizing to the building needs, the building engineer can focus on the urgent and impactful interventions from the perspective of occupant comfort, energy consumption, cost, and business impact.

6.11.1 Alarms

In the proposed design, the building engineer (administrator) is allowed to configure alarms. Some alarms like temperature alarms, smoke are essential and common, while some others are building-specific. The proposed design provides a list of basic configurable alarms with priority, and mode of contact when activated. There are three priority levels available – Low (Info), Warning, Critical.

While adding notification method (email or text), the administrator also specifies the intended text/email address. More than one user can be registered to receive such alarms. If nothing is specified, the email is sent to the members in the administrator category.

Additional customization is also available depending on the type of alarm. For example, if an alarm is being set when the temperature exceeds a certain value, say 80 °F, the administrator can specify the value and unit and a customized alarm will be generated for this scenario.

All alarms currently set in the system are available for the administrator to view. This feature provides details about what type of alarm is set, customizations, if any, the alarm priority, the notification mode and address information, and the time this alarm was registered in the system.

Alarms being an important feature in BEMOSS, are allowed to be completely customizable from the list of alarms possible in the system. The proposed design does not provide any default alarm settings, but expects the user to create their alarms using a list of alarms. This way the users can set their own alarms and their priority levels.

Once an alarm is registered in the system, it is displayed in the Alarms page under Registered Alerts as shown in Figure 6-30.

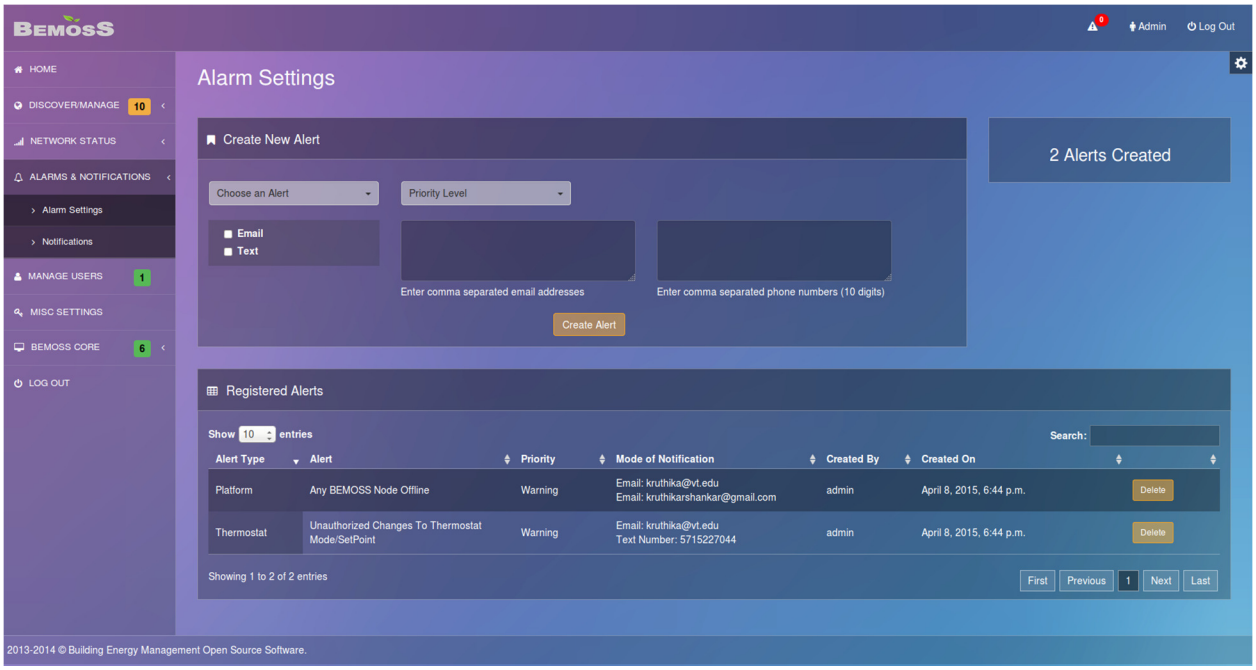


Figure 6-30 Registered alerts in the alarms page
Source: http://www.bemoss.org. Used with permission, 2015.

Figure 6-31 shows the custom alarm creator widget. The user can choose a custom property like Temperature, Low Battery etc. using this widget. The user can also provide a comparator such as greater than (<), less than (>) etc. and enter an integer value to trigger the alarm. The user can then set the priority level thus activating the alarm.

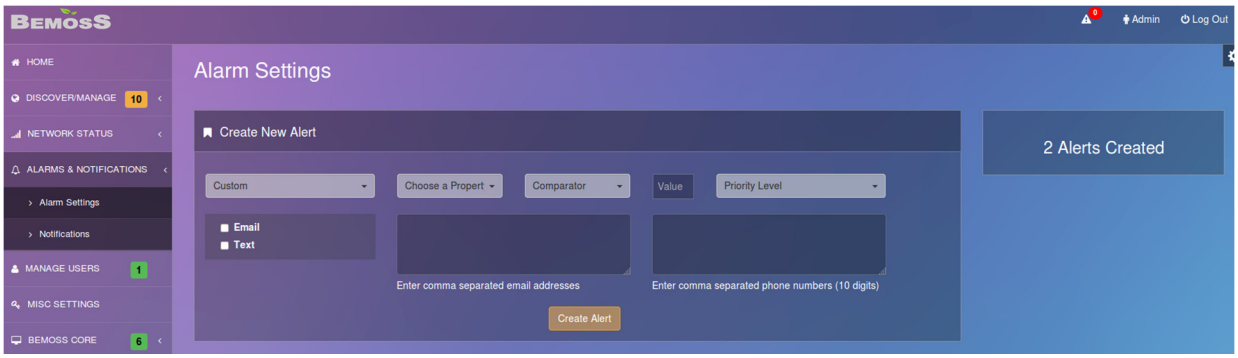


Figure 6-31 Custom alarm generator in alarms page
Source: http://www.bemoss.org. Used with permission, 2015.

6.11.2 Notifications Tool Bar

A clearly visible notification bar ensures that the user reads notification. In the proposed design, a notification bar is placed in the top right corner of the screen. Total number of notifications is available in intuitive colors to grab attention. The notifications bar is available only to the administrator of the system.

Alarms also appear in the notification bar. Low-priority alarms are removed once viewed. A warning message is treated similarly, unless the administrator specifically requires that the intended people be notified via text or email.

In addition to the alarms, important system activities are also notified. A pending user registration request for example, is notified to ensure that user registration requests are not ignored.

6.11.3 Notifications Log

A log of all notifications is maintained by the system for diagnostics and analysis. The proposed design has a separate page to review all notifications that occurred in the system in reverse chronological order. Since there is a possibility that the log may contain hundreds of notifications, they are displayed on demand. The latest set of the notifications are displayed first, and further logs are retrieved as needed. This ensures that the page is not overloaded, and responds faster to requests.

6.11.4 Notifications Floating Bar

Notifications are not only the events that are registered in the system, but also the status updates for all changes requested by the user in the system. Every time a user requests a change in device status, a command is triggered as per the control algorithm, and the success/failure message is notified to the user. These notifications are displayed in the form of floating notifications.

Floating notifications can be made available in any of the four corners of a web page. These floating notifications bar popup shows the message for a few seconds and disappear. It is intended for a user who sends a control message and expecting to see a system response about the success/failure of the control command submitted. These messages provide a quick and lightweight means of updating the user about the system status.

The floating notifications mechanism is completely asynchronous and built using web sockets. Therefore, it ensures that there is little load on the server and still provides required information to the user. The message provides information about the time an update sent to the system, and if the update is a success or a failure, and in certain cases the user requested the update. If there is an error at any level, a generic error message is reported.

Notifications are based on a variety of events that occur in the system. They can be because of an alarm activation, new user registration request, user registration request approval, and system update status.

Notifications are available in the notifications log section in the side navigation bar. Most notifications are also available in the top navigation bar where a maximum of five new notifications are displayed. The user can either click on those notifications to take action or just view them to mark them read. If the notification is critical, the notification stays on the notification alerts bar until the error is rectified.

Another form of notifications is the system status update in the form of a pop up bar in the right bottom corner of the page intimating the user of the status of the recent updates requested. The

notifications popup bar is floating, and disappears in five seconds. This timing is adjustable by a developer. Figure 6-32 shows some sample notifications.

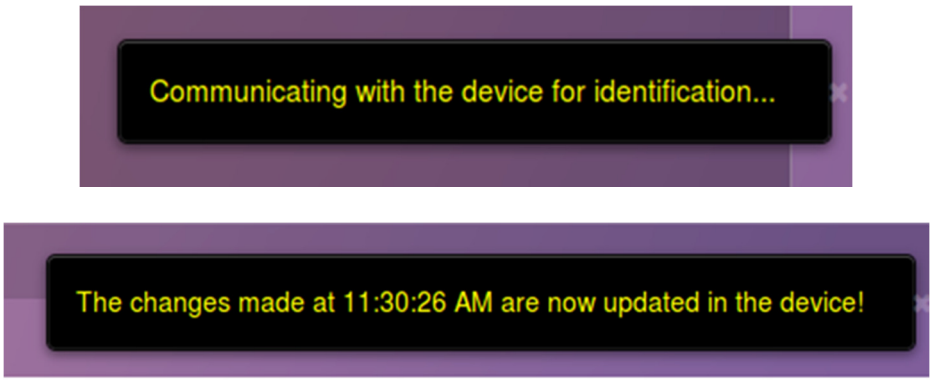


Figure 6-32 Sample notifications
Source: http://www.bemoss.org. Used with permission, 2015.

6.12 Network Status Dashboard

The network status dashboard has two primary components – Node Status, and Device Status.

6.12.1 Device Status

The proposed design features a device status dashboard for the entire system grouped by device type. All thermostats and HVAC controllers are grouped together, lighting load controllers grouped together, and so on. The status viewer scans the system for all approved devices and determines their online / offline status. The last scanned time is displayed, since the scans are conducted periodically every few minutes.

The last offline time, if any, is displayed since a device could have gone offline the last time since it was scanned. The zone information, device nickname and other identification level details.

The status viewer is shown in
Figure 6-33.

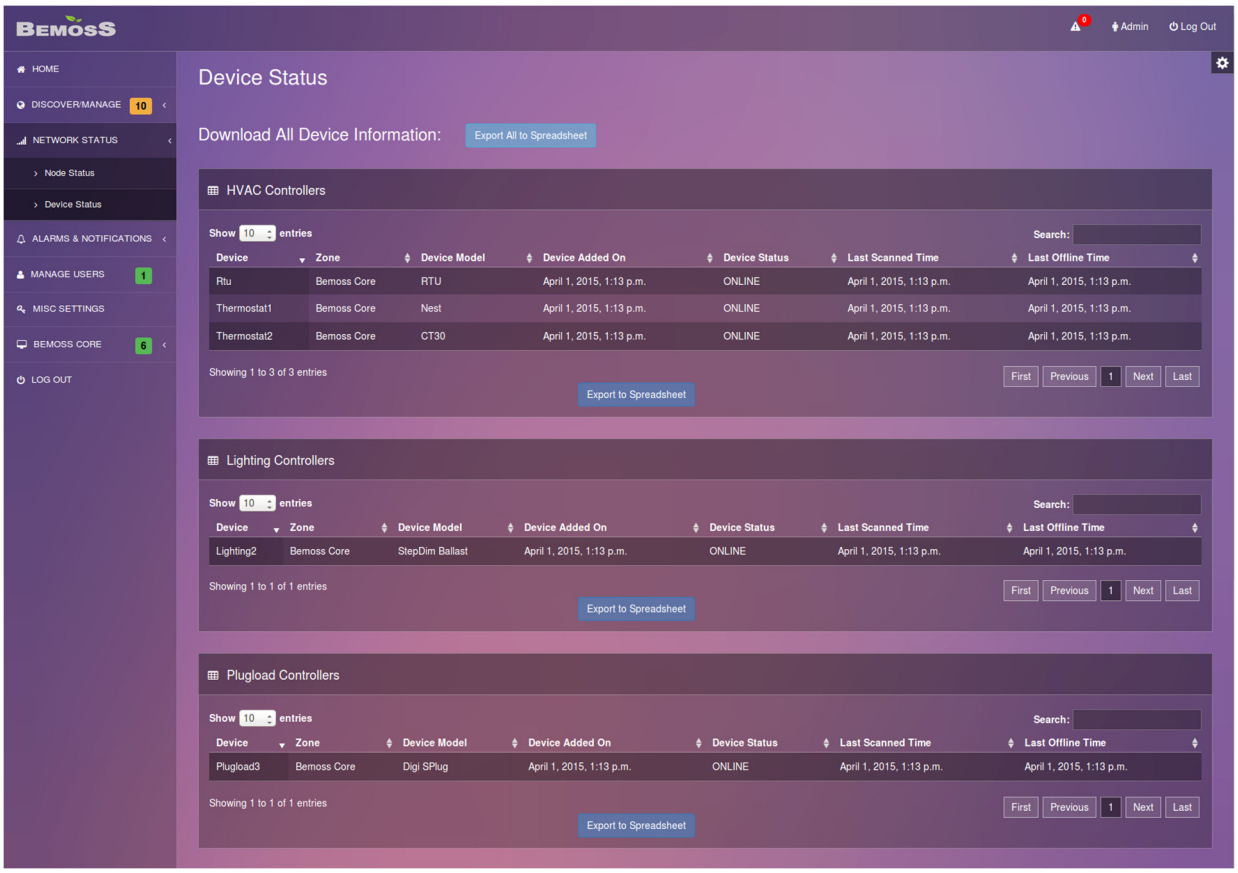


Figure 6-33 Device status dashboard
Source: http://www.bemoss.org. Used with permission, 2015.

6.12.2 Node Status

The proposed design features a Node Status dashboard which provides information similar to that of device status dashboard. The network is scanned for nodes that are registered in the system, and communicated with nodes to infer their status. This health check is performed periodically and the dashboard is updated. The node nickname, node type (Beaglebone, Pandaboard), node status (online/offline), last scanned time, last offline time, etc. are available in the Node Status dashboard. The core system is also included in this dashboard, thus ensuring that the health of the core system is also scanned.

The node status viewer is shown in
Figure 6-34.

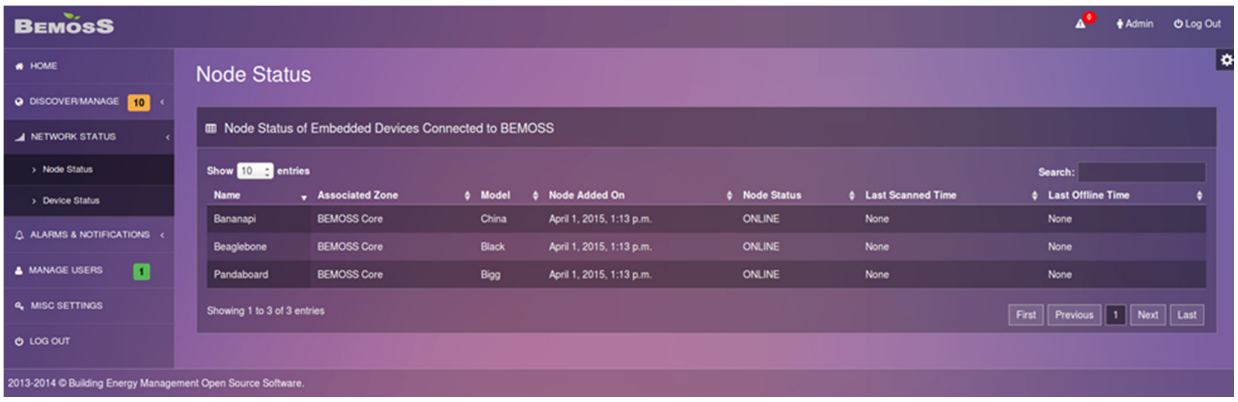


Figure 6-34 Node status dashboard
Source: http://www.bemoss.org. Used with permission, 2015.

6.13 Zone Management

Zone management in large buildings significantly reduces the load on the system. For buildings with a large number of devices, BEMOSS can be deployed as a multi-layer system. In a multi-layer architecture, the building can be divided into several small zones, each having its own BEMOSS node running in an embedded system. These BEMOSS nodes are in turn connected to a larger embedded system or server, called the BEMOSS core. Each BEMOSS node is responsible for monitoring and control the zone in which the selected set of load controllers reside, while the BEMOSS core is responsible for supervising the overall system operation and allow local and remote access for monitoring and control of all devices in buildings.

For even larger buildings, the BEMOSS architecture can be expanded to accommodate more devices. A number of BEMOSS nodes can be deployed on different floors or zones in a building depending on the number of zones and devices to be monitored and controlled. This zone management is built for scalability, reliability, and redundancy.

6.13.1 Automatic Zone Generation

In the proposed design, the user platform ensures device controls change as more zones are added. For this purpose, the user platform adds zones automatically when a new node is discovered. Once a new zone is generated, the user can manage the zone by adding or removing devices in the system. Devices corresponding to these zones are monitored and controlled by the associated zone controls. The information from the zone to the user platform is communicated via the common message bus, where the message is re-routed from the zone to the core to the UI platform.

6.13.2 Virtual Zones

In addition to generated zones in the user platform, the user (administrator) can create virtual zones, which can accommodate logical zones in a building. These virtual zones can be created from the dashboard page. Once a new virtual zone is created, it is added to the system, the core platform is intimated about the new zone. The virtual zone will be managed by the BEMOSS core,

but will appear as a zone to the user. For the user platform once a virtual zone is created, there is technically not much difference between the virtual zone and the generated zones.

6.13.3 Zone Nicknames

All zones can be nicknamed for user convenience similar to the nicknames for devices. When a new virtual zone is created, the user enters a nickname. The zone management happens in the user platform abstracting the details from the user. A generated zone has a name assigned to it based on the embedded system name corresponding to the zone. The administrator can modify any of these nicknames anytime from the dashboard. Nicknames are expected to follow certain rules for security purposes.

6.13.4 Zone Dashboards

The core and every zone has a centralized intuitive dashboard which to view all devices and the current status at any point of time. This is accessible from the side navigation bar. The zone dashboard (Figure 6-35) lists all devices that are currently online and their current status. It also has an ‘Identify’ button used during device approval and management.

Each of the widgets in the Zone dashboard has the following features:

- Current Status
- A visual image of the device (optional)
- Identify button (available to the administrator only)
- View/Edit Information (opens a modal window)

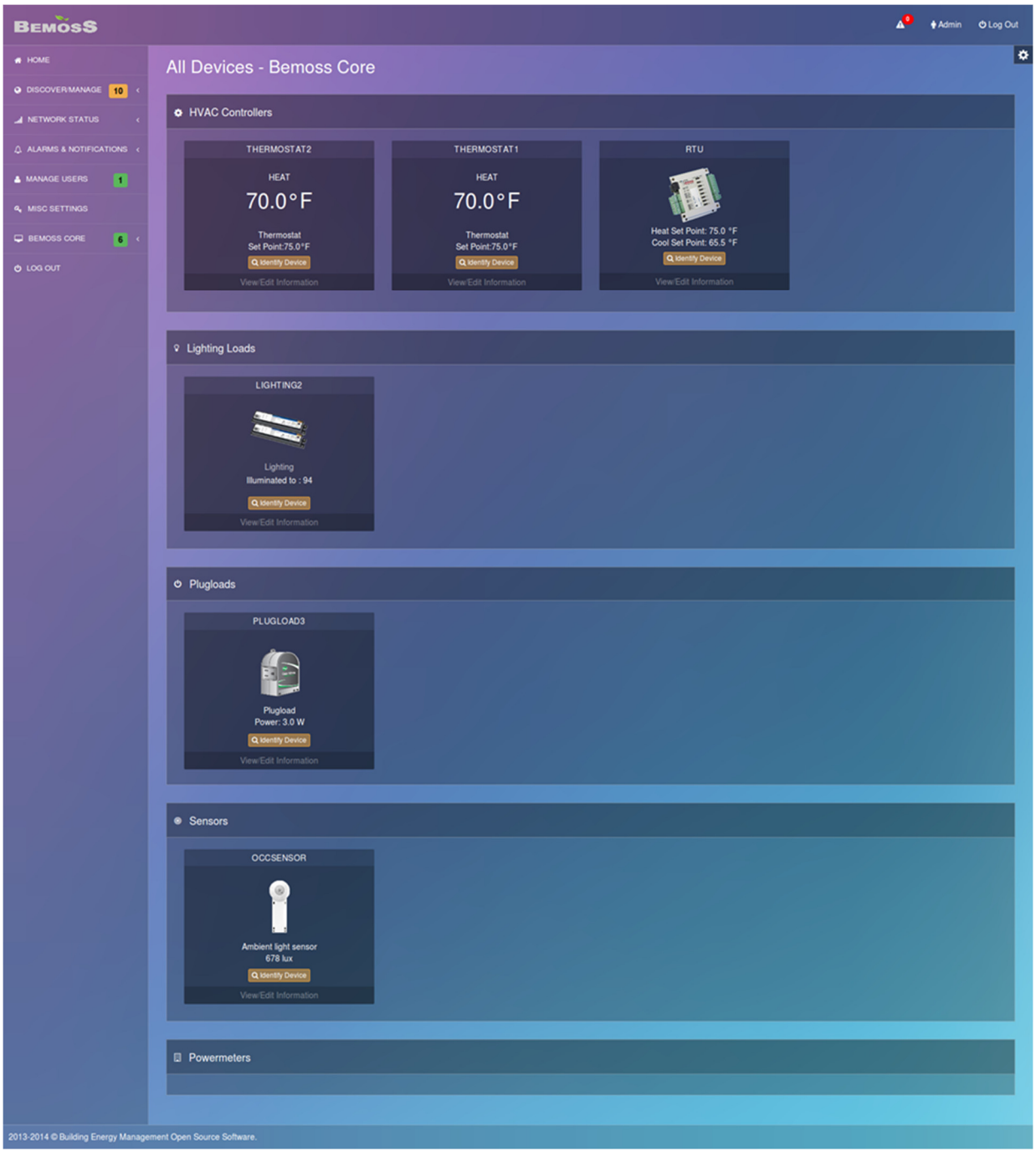


Figure 6-35 Zone dashboard
Source: http://www.bemoss.org. Used with permission, 2015.

A few examples of how these elements are laid out to provide the necessary information with a visual impact is discussed below.

The thermostat widget (Figure 6-36) has necessary information that a user will look for while looking at a comprehensive dashboard.

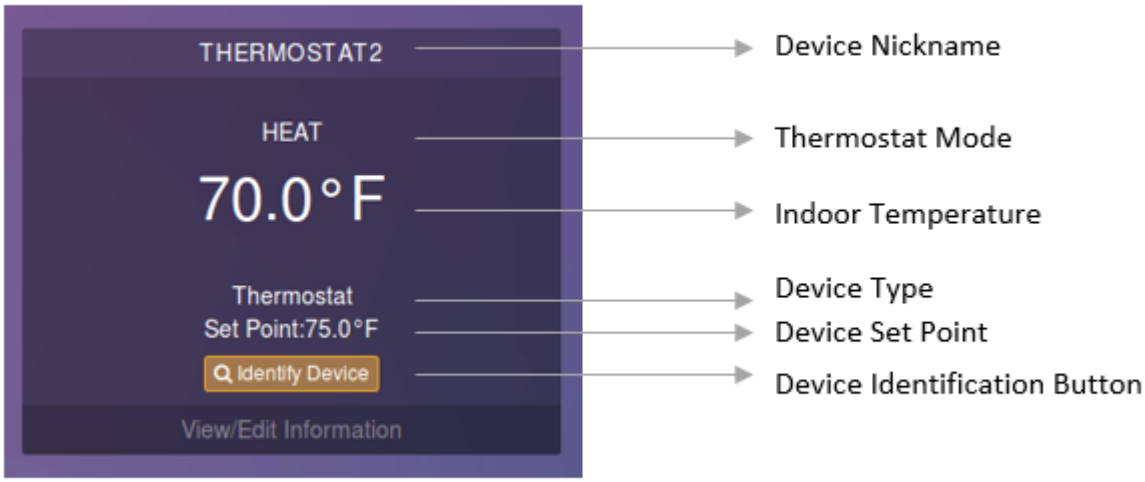


Figure 6-36 Thermostat widget in the dashboard
Source: http://www.bemoss.org. Used with permission, 2015.

Similarly, Figure 6-37 shows a device which has the image shown.

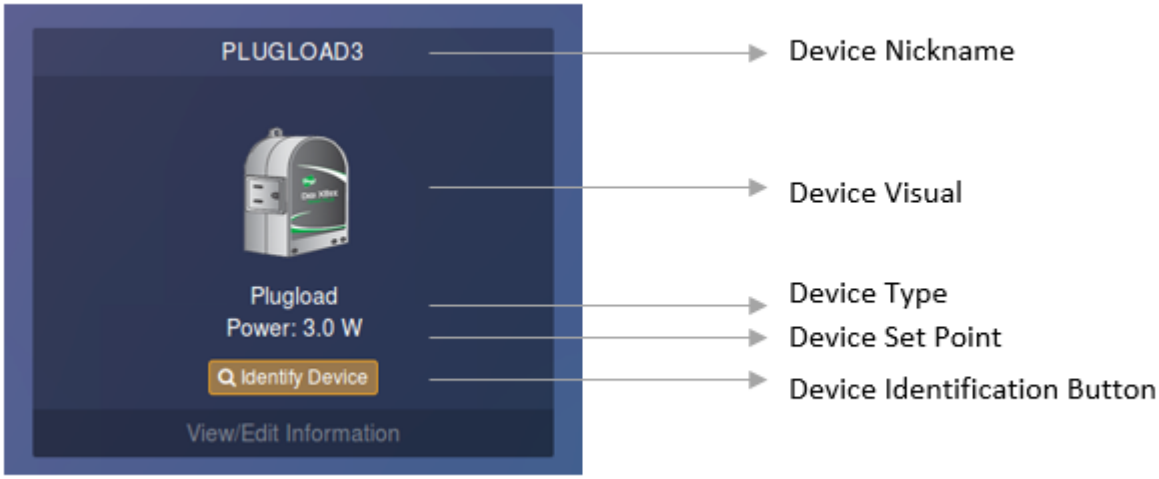


Figure 6-37 Digi smart plug dashboard widget
Source: http://www.bemoss.org. Used with permission, 2015.

6.13.5 Zone Dashboard based on Device Type

Device type is a deciding factor for allowing a logical separation between different devices. Device types are of a broad form – HVAC controllers, plug load controllers, lighting load controllers, sensors, power meters, etc.

Grouping devices by their broad device types allows sensible navigation throughout the user interface. A dashboard is available for every device type similar to the zone dashboard.

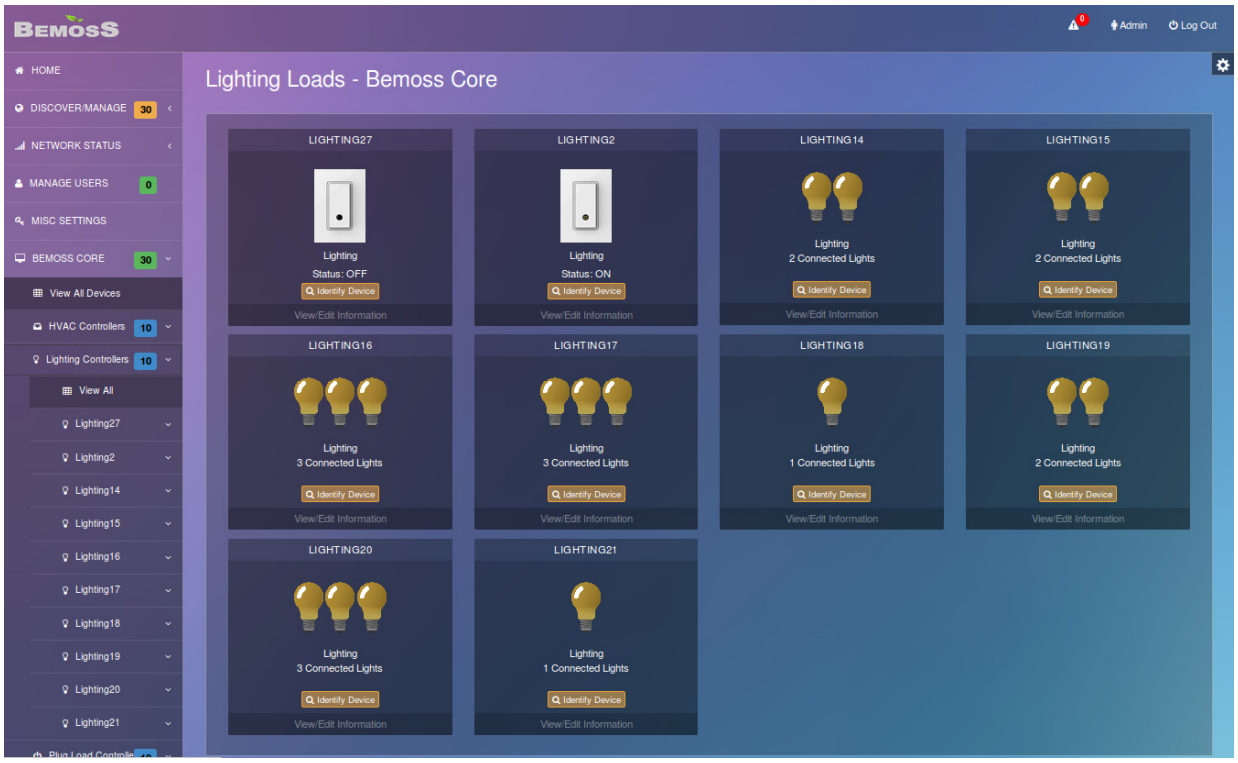


Figure 6-38 Zone dashboard for the lighting load device type
Source: http://www.bemoss.org. Used with permission, 2015.

From the side navigation bar, under lighting controllers, the 'View All' button takes the user to the above page, where the user can see a list of all lighting load controllers in the BEMOSS core.

6.14 Reports

Report generation involves collecting information from a variety of sources, consolidating into appropriate tabs and sheets, and generating a report document.

Report generation from BEMOSS eliminates the need for a user to be logged on to the system to analyze data. Data collected by BEMOSS every day is enormous and this data can be valuable if reports can be automatically generated by the system at the click of a button.

In the proposed design, reports can be generated for multiple scenarios. The generated reports are stored in a spreadsheet organized sheet-wise for different data points. Reports about the current system status, time-series data collected over time, and schedules can be generated from various locations in the system.

For example, from the Network Status pages, the current device status information can be collected and downloaded in the form of a spreadsheet. From the visualizations page, the reports about the time-series data collected for different data points for a device can be downloaded. This can also be filtered by date and time. Device Schedule reports can also be generated from devices.

The following standard report templates are available in the proposed design:

- Reports to record alarms and notification logs
- Reports to record system status
- Reports to record system schedules
- Reports to record time-series data about the system status over different periods of time.

Report printing is scattered in relevant places in the BEMOSS application to ensure that the user understands the reports needed and how they are customized. Having a dedicated report printing panel with a variety of reports will only provide a huge log of events and system information and may not be customizable.

Reports are available for the following:

i) Device Status Log
    a. All Device Types
    b. Every Device Type
ii) Schedule Log
    a. For Each Device
        i. For Each Schedule Type
iii) Past Usage Statistics
    a. For Each Device

Figure 6-39 shows the device status page. On clicking on ‘Export All to Spreadsheet’ button, a file open dialog opens, with a default file name asking for user instructions.

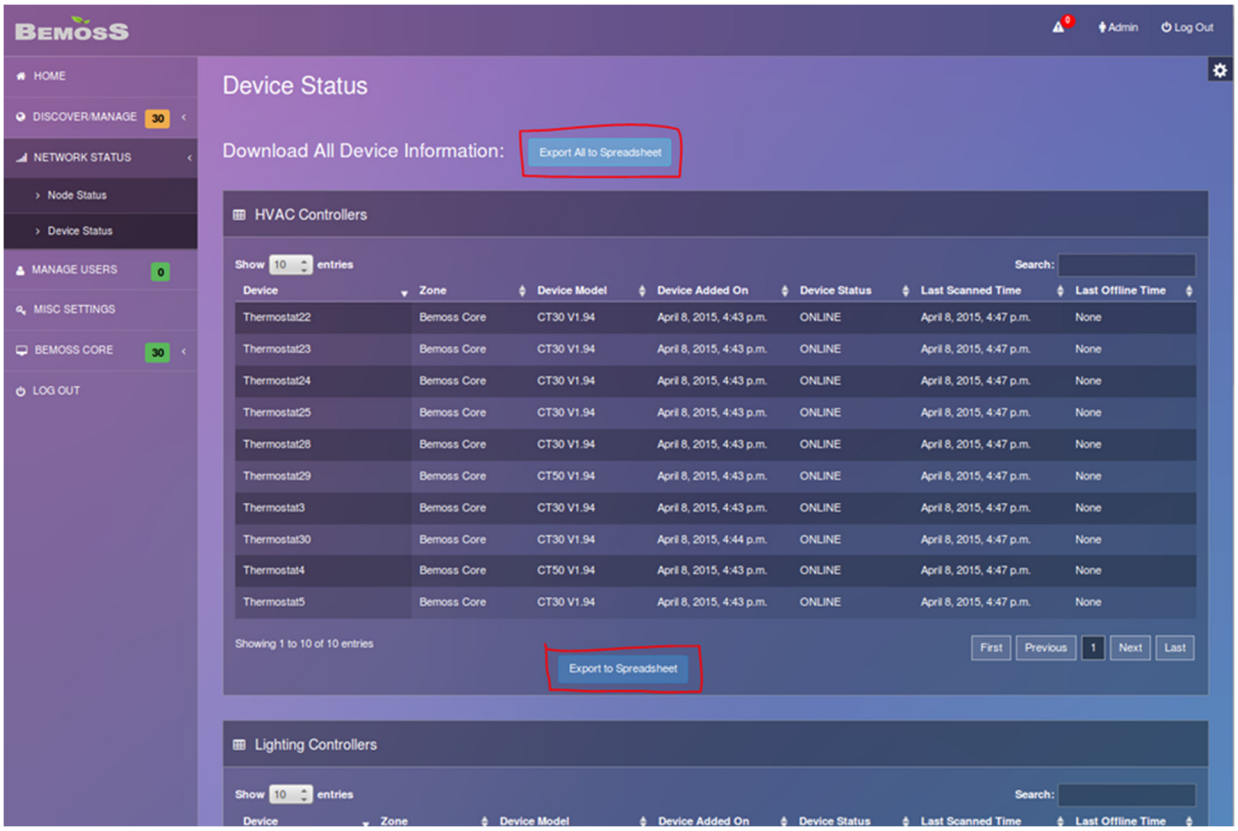


Figure 6-39 Export to spreadsheet options on device status page
Source: http://www.bemoss.org. Used with permission, 2015.

This is the general application behavior that is replicated to make navigation intuitive. Figure 6-40 shows an example of device status log file that was exported using the report printing application. Devices are sorted by device type as shown in the sheet name; each new device type is assigned a different sheet thus allowing ease of navigation.

	A	B	C	D	E	F	G
1	Device Nickname	Zone	Device Mode	Device Added On	Network Sta	Last Scanned Time	Last Offline Time
2	Thermostat28	Bemoss Core	CT30 V1.94	2015-04-08 16:43:48.584716	Online	2015-04-08 16:51:09.427847	None
3	Thermostat29	Bemoss Core	CT50 V1.94	2015-04-08 16:43:59.519484	Online	2015-04-08 16:51:20.409443	None
4	Thermostat30	Bemoss Core	CT30 V1.94	2015-04-08 16:44:00.443066	Online	2015-04-08 16:51:21.259009	None
5	Thermostat3	Bemoss Core	CT30 V1.94	2015-04-08 16:43:07.745801	Online	2015-04-08 16:51:08.741381	None
6	Thermostat4	Bemoss Core	CT50 V1.94	2015-04-08 16:43:08.737962	Online	2015-04-08 16:51:09.510798	None
7	Thermostat5	Bemoss Core	CT30 V1.94	2015-04-08 16:43:09.532526	Online	2015-04-08 16:51:10.423930	None
8	Thermostat22	Bemoss Core	CT30 V1.94	2015-04-08 16:43:33.511656	Online	2015-04-08 16:51:14.409866	None
9	Thermostat23	Bemoss Core	CT30 V1.94	2015-04-08 16:43:34.347537	Online	2015-04-08 16:51:15.246643	None
10	Thermostat24	Bemoss Core	CT30 V1.94	2015-04-08 16:43:35.162042	Online	2015-04-08 16:51:16.088241	None
11	Thermostat25	Bemoss Core	CT30 V1.94	2015-04-08 16:43:35.971573	Online	2015-04-08 16:51:16.897310	None
12							
13							
14							
15							
16							

Figure 6-40 Exported device data shown as a spreadsheet
Source: http://www.bemoss.org. Used with permission, 2015.

6.15 Navigation

Navigating through the BEMOSS system is in itself a challenge given numerous devices and device-related functions available in the system. Intuitive navigation and fast access to important information related to the system is important for the user to navigate to the desired location in the system.

BEMOSS is a host of multiple devices, each of them having their own dashboard, scheduler, and time-series data visualization tools. In the proposed design, an intuitive side bar navigation is provided that gives primary access to all important pages in the system. The navigation bar is organized as follows.

- Home
- Discover/ Manage
    - Nodes
    - Devices
- Network Status
    - Node Status
    - Device Status
- Manage Users
- Miscellaneous Settings
- BEMOSS Core
    - View All Devices
    - HVAC Controllers
        - Thermostat1
            - Monitor and Control
            - Set Schedule
            - Charts and Statistics
        - …
    - Lighting Controllers
        - Lighting Controller 1
            - Monitor and Control
            - Set Schedule
            - Charts and Statistics
        - …
    - Plug Load Controllers
        - …
    - Sensors
        - …
    - Power Meters
        - ...
- Log Out

The home page is the landing page for the users (administrators and tenants, alike, but for hidden features for the tenant) on a successful login. The home page has the option to create a new virtual zone available to the administrator. The Discover/ Manage pages for nodes and devices are visible only to the administrator, and hidden from the zone manager and the tenant. These pages provide the necessary tools to manage devices and nodes once they are discovered.

The 'Manage Users' side bar tab is available to the administrator who has permissions to approve or deny a user registration request, modify user permissions and roles. The 'Miscellaneous settings' available on the side bar is a quick way to navigate to the system settings, where general settings including managing zip code of the building location, holidays, and username/passwords for devices such as Google Nest [75], Ecobee [76] thermostats is available.

Following these nodes on the side bar, different nodes are available in the system starting with the core. Within these node headers, the 'View all' device takes the user to the dashboard that consists of all devices that the zone is equipped to manage. The different devices are also listed with the corresponding pages in child nodes.

The side bar thus serves as the one-stop location for accessing all important system features and navigate intuitively, without having to jump from parent to child pages to navigate. This intuitive navigation system also provides appropriate and intuitive icons to help a user navigate through the system.

6.15.1 Side Navigation Bar

Side navigation bar is designed such that the most important functions of the system are available from any page for the user to navigate to. The side navigation bar is role-centric, and only if the user is an administrator, all features are visible. Otherwise, only a limited number of features and functionalities are made available to the user.

A screen capture of the side navigation bar is shown in Figure 6-41. This is the side bar view when all sub nodes are closed.

The nodes 'Discover/Manage', on clicking, collapses to show 'Nodes' and 'Devices' sub-nodes. The number beside the 'Discover/Manage' gives the total number of devices waiting for approval from the administrator and all of the nodes. The number beside the sub-nodes, 'Nodes' shows all nodes currently online in the system. The number beside the sub-nodes 'Devices' shows the newly discovered devices awaiting approval from the administrator. They are not incorporated in the system until the administrator approves it.

The 'Network Status' node, on clicking, collapses to show 'Node Status' and 'Device Status' sub-nodes. These nodes show the status information of the nodes and devices approved in the system. The 'Alarms and Notifications' node, on clicking, collapses to show 'Alarm Settings' and 'Notifications' sub-nodes.

The 'BEMOSS Core' is available for any deployment of this prototype design. This node collapses to show all device types and devices that are part of the Core system. The number beside 'BEMOSS Core' shows the total number of devices that are managed by BEMOSS Core.

The last option in the side bar is the ‘Log Out’ node, clicking on which closes the current user session and performs logout.

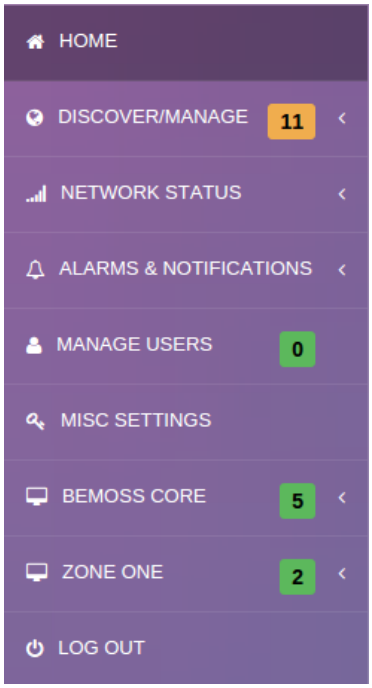


Figure 6-41 Side navigation bar
Source: http://www.bemoss.org. Used with permission, 2015.

The BEMOSS core opens to show the sub-nodes as shown in Figure 6-42. When the ‘BEMOSS core’ is clicked, it collapses to a structure presented in the left. Clicking ‘View all Devices’ opens up the dashboard page for this core, displaying all devices present in the core grouped by device type. The total number of devices controlled by the core is shown at the top, and the number next to each device type shows the number of that type in the core.

For every device type, the collapsed node shows the list of all devices listed using device nicknames. Each of these devices has three sub-nodes – ‘Monitor and Control’, ‘Schedule’ and ‘Past Usage Statistics’. While monitor and statistics features are available for all device types, control and schedule features are unavailable for sensors and power meters. This is reflected in the sidebar as well.

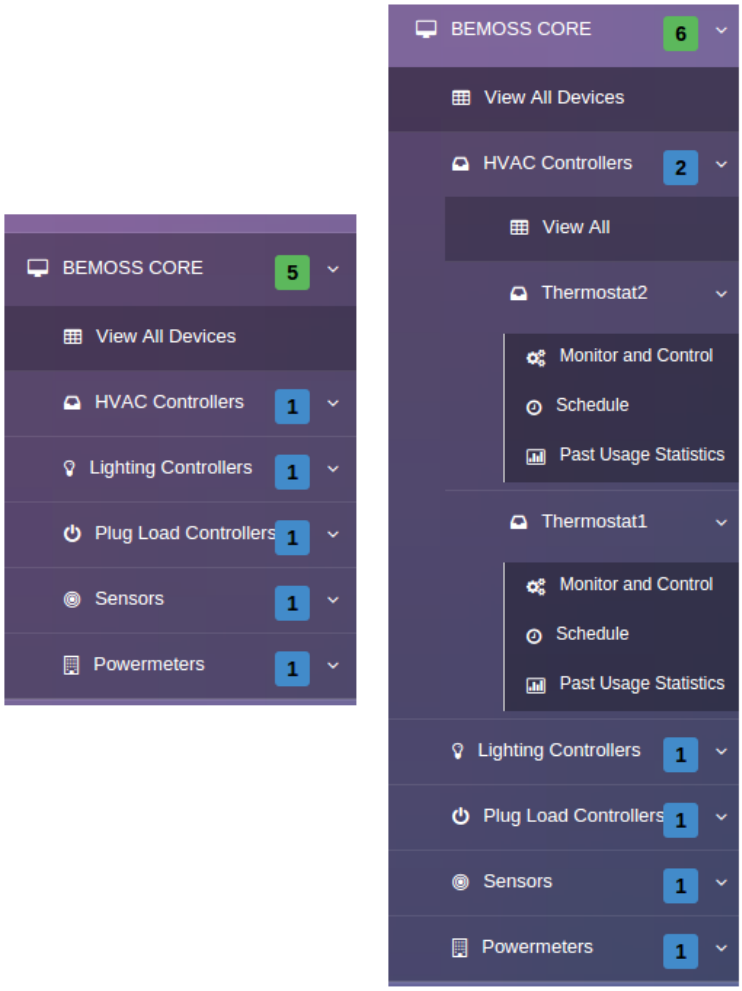


Figure 6-42 Sub-nodes for each zone or BEMOSS core
Source: http://www.bemoss.org. Used with permission, 2015.

Similarly, every node in the system contains sub-nodes with all features mentioned above. For example, in this case, Zone One is an additional zone, which collapses to show all features like those shown under BEMOSS core. This comprehensive side bar is available from all the pages in the system.

6.16 Settings and Preferences

While most of the settings are available in the corresponding device dashboards, some settings are specific to a set of devices or to the entire platform. Such settings are available in the 'Miscellaneous Settings' page. Currently, there are options to add 'Holidays' and 'Zip Code' in the settings. This page can be used to add any additional settings, like custom user account information for devices like Google Nest, Ecobee, which have their own customized authentication mechanisms.

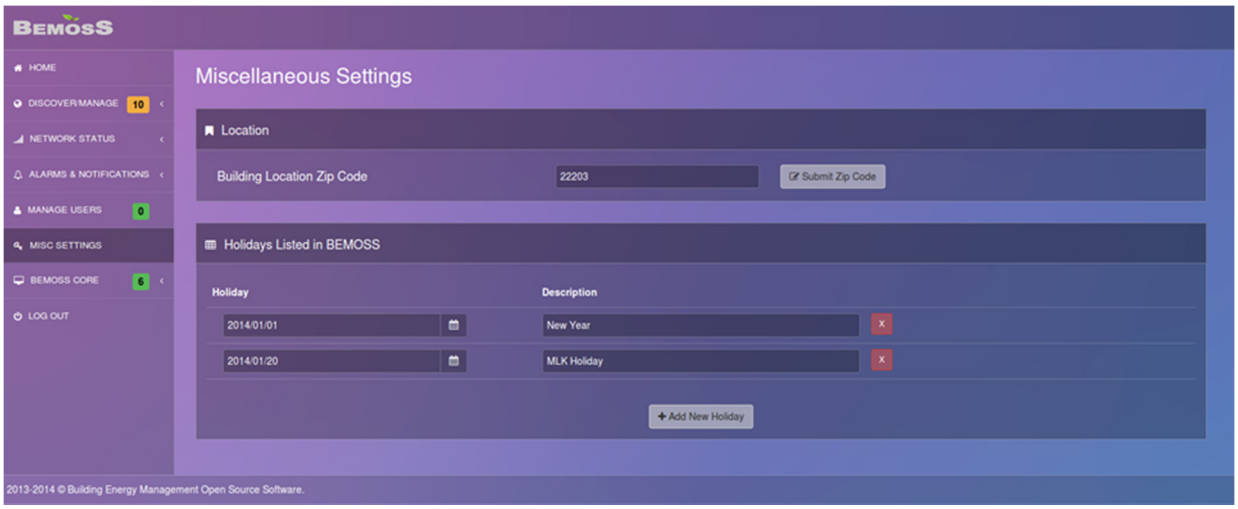


Figure 6-43 BEMOSS miscellaneous settings
Source: http://www.bemoss.org. Used with permission, 2015.

In terms of preferences, a general user has access to color preferences, where the user can choose from a list of color preferences from the widget on the right top corner of any page bearing login and logout and error pages.

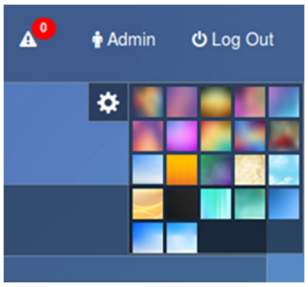


Figure 6-44 Dashboard color preference widget
Source: http://www.bemoss.org. Used with permission, 2015.

This option is provided to users since the dashboard is a thin interface with minimum design elements, and focuses on using shades of grey for its design elements. The only preference the user has is the background color. This improves the spatial layout preference of an individual user.

7. Relational Data Model Design

This chapter discusses the relational database that was designed and implemented to store metadata as a part of the energy management system. This metadata contains all important information about a device and helps uniquely identify all devices.

7.1 Objective

The relational database should encapsulate several views of the system: device information, device relationship, user information, alarms and notification, configuration information, user session information. This model should allow a comprehensive representation of the entire BEM system that can be updated and maintained periodically. For this metadata information storage, a relational data model has to be available that allows OLTP (Online Transaction Processing) model. This database will not be used to perform data analysis but rather used for basic functionality of the entire BEM system.

The relational data model should have a central table that lists all devices and connects to tables that are based on individual device types. It should cater to the user management and session management of the user platform. It should also handle all alarms, notifications, and important system logs. The user platform depends on the relational data model described here for its important functions. Therefore, it should be designed as a fast, reliable data model that is also secure from outside influence.

Some of the questions/queries that require answers from this relational database are as follows:

- Is the user “abcd” currently in session? Can you provide session information?
- How many devices are currently marked as ‘NBD’ (Non-BEMOSS Device)?
- A list of all devices that are ‘Approved’ and ‘Online’ currently.
- What zone is the zone manager ‘xyz’ assigned to manage?
- How many user sessions are currently active?
- Who is the manufacturer of device ‘abc’?
- What are the notification logs in the system in the year 2013?
- How many alarms of ‘critical’ type are currently registered in the system?

7.2 Database Design

The database design is progressive. The system needed a strong data store for metadata management of all devices, users, and system information. This was achieved by using the following data models:

i) User Model
ii) Session Information Model
iii) Alarms and Notifications Model

iv) Device Model

This section discusses the need and design of these data models.

7.2.1 User Model

As part of the metadata modeling, the user model was developed first. It is designed to authenticate and authorize users of the system. Within this model, there are Users, Permissions, and Groups.

Figure 7-1 describes the user model for the BEM system. The model is normalized suit system needs. The *django_content_type* table is system-generated to log information about all applications and corresponding data models that are part of the system. Although this table is not used in practice, it is useful for the analysis of system, applications, and their relationships.

The user model also assigns roles to users, and in case of a zone manager, links the system to a *building_zone* table. This table in turn links device tables to the user models.

The data models use one-to-one relationships and many-to-one relationships to capture relationships between tables and encapsulating device details within the tables.

The user permissions can be used to verify if the user is allowed to perform certain activities at the system level, and/or access devices. The *auth_user* table provides the general information about the user, while the *accounts_userprofile* table provides the role information about users. Since roles meant redundant information, they are further normalized into the *auth_group* table.

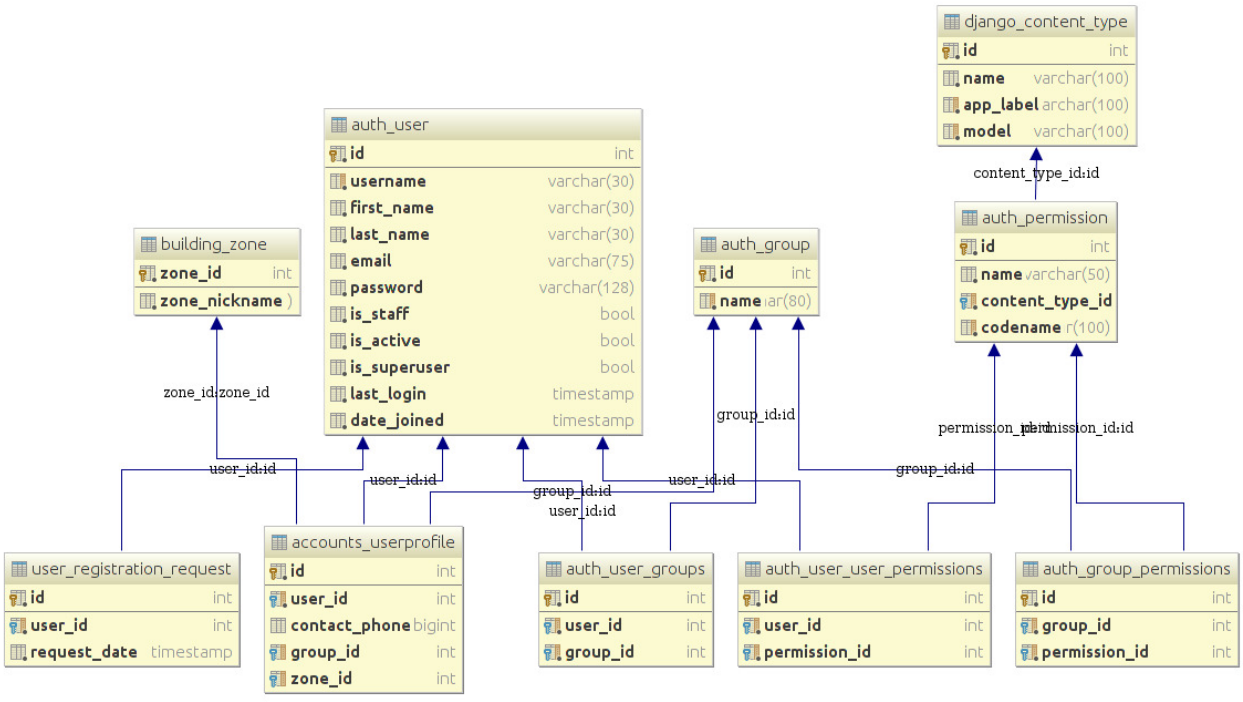


Figure 7-1 User model in BEMOSS

7.2.2 Device Model

Device model stores device manufacturer, model information and their status in the BEM system. It should satisfy the following needs of the application:

- Device approval
- Device information (manufacturer information, MAC address, approval status, etc.)
- Device status information (nickname, current status, network)

Device model was initially developed as a single *device_info* table and was then normalized to accommodate all device types without repetitive information.

The challenge in designing the device model for a BEM system lies in the capacity of the model to accommodate any new device type that is introduced into the system at a later stage in the development process. The data model should be capable of accommodating any new device type into the system. The device model design is shown in Figure 7-2.

Tables *device_info, node_info, building_zone* form the heart of the model. All of the generic constant information about every device is stored in *device_info* table. This table has the list of all devices, identified by their unique device id. The unique *device_id* is a combination of *device_model_id* and MAC address, so it never changes for a particular physical device.

Each device type, like thermostat, lighting, plugload, etc., has a separate table that holds device specific frequently updated information. These tables are mapped to *device_info* table using *device_id* as the foreign key. The model shown below shows three different device types – thermostat, plugload, and lighting. More device type tables (power meters, motion sensor, occupancy sensor, etc.) can be added and linked to *device_info* table without modifying the existing tables.

The *global_zone_setting* table is used to set global heat and cool set points, and brightness at the zone level. All of the above described tables are linked together with the *building_zone* table. The device model is updated by the core every time a device is discovered or the device status is modified.

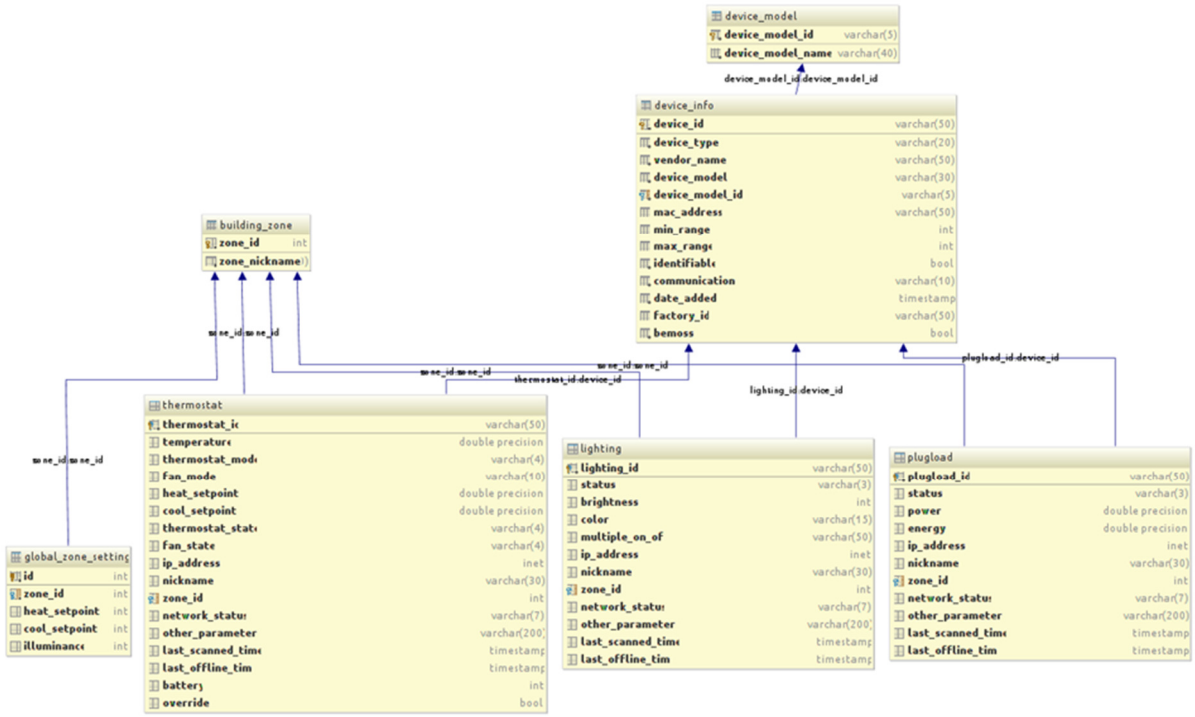


Figure 7-2 Device model in BEMOSS

7.2.3 Session Information and Administrator Logs

Although a slow process, it is more reliable and secure to depend on a database-backed session for the BEM system. The session information is stored into the *django_session* table. The session id and the session expiration date and time are logged into the table for currently in-session users.

The administrator logs are stored in the *django_admin_logs* table which is mapped to the *auth_user* table to fetch the user id and the *django_content_type* table to fetch the content type information.

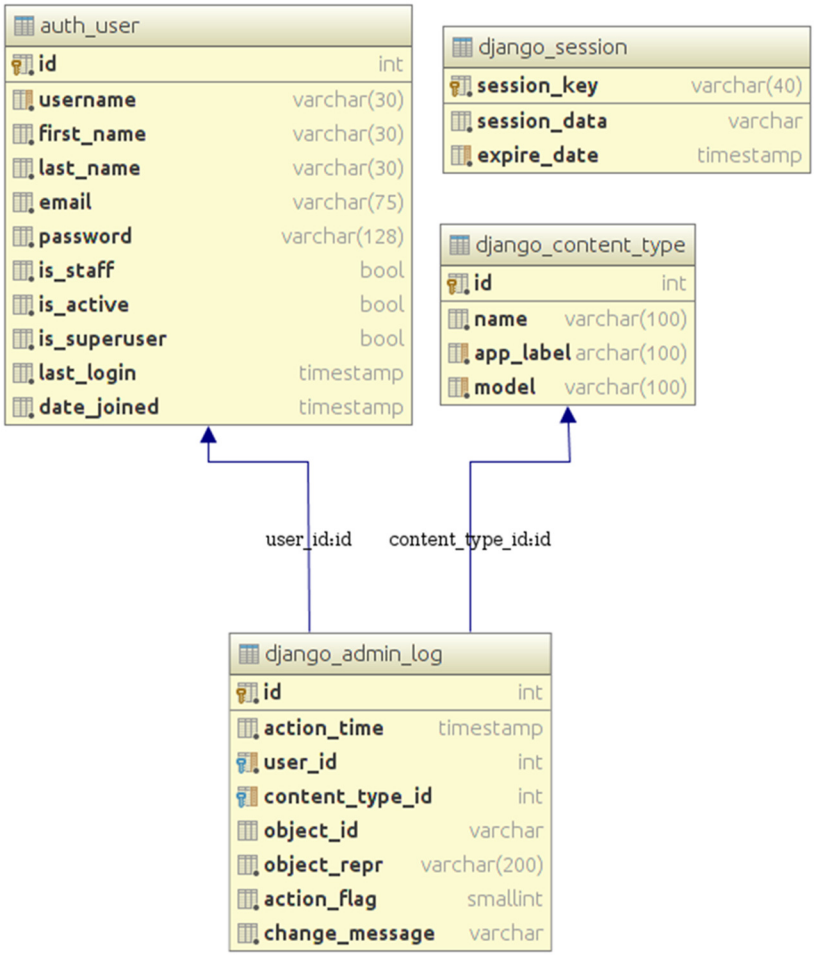


Figure 7-3 Session and administrator logs

7.2.4 Alarms and Notifications

Alarms and notifications being a critical component in a BEM system, need a comprehensive data model to accommodate the variety of alarms that can be registered into the system. The prototype design (shown in Figure 7-4) allows the user to generate new alarms with the custom alarm registration feature. This is discussed in Section 0. To allow such custom designs, the database should allow the flexibility to add or remove custom parameters.

For example, the administrator can set an alarm to send a notification when the temperature in a specific thermostat exceeds, say, 78 °F. In this case, the device id, the parameter temperature (it could be brightness in case of lighting load controllers), and the value of 78 ° F are all dynamically set. When such an alarm is registered in to the system, the control platform can have custom algorithms written that can trigger the notification as per the alarm registered.

The *active_alert* table stores all alerts registered in the user platform. The alarm priority is mapped to the priority table which allows low, warning, and critical as the three levels of priority for any registered alarm. Notification channels are of three types:

i) BEMOSS notification, which is the in-application notification
ii) Email notification
iii) Text notification.

The application allows multiple email addresses and text numbers to be added for a single alarm registration. When a notification is triggered, the notifications are logged in the notification table. All customizable parameters are available in the *active_alert* table.

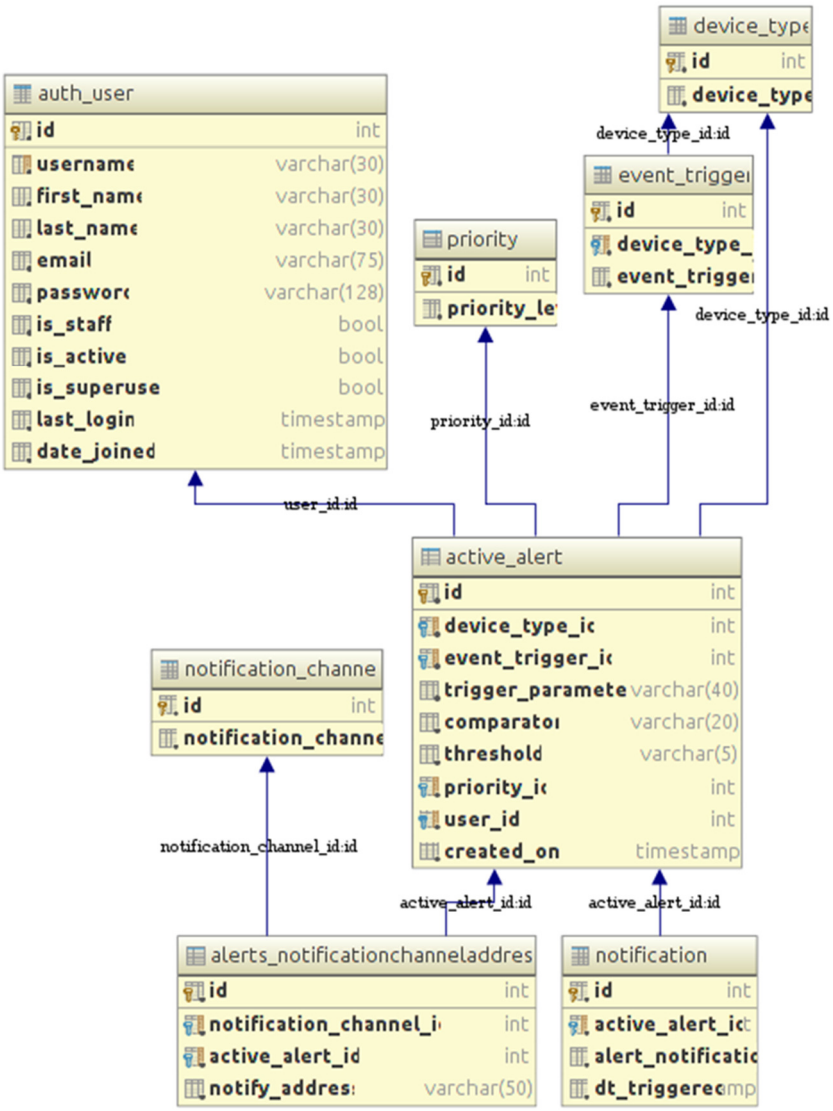


Figure 7-4 Alarms and notifications model

8. Security in the User Interface

Security becomes an important factor in the design and development of these systems, especially if the system is designed as a distributed system depending on different technologies and protocols. When some of the components of this distributed system rely on the public Internet for its services, the need for security becomes even higher. Such systems are deployed in a building, with devices scattered all around a building. Physical security also becomes important in such cases. In addition to providing DR integration, BEMOSS is designed to serve as a web application that can be accessed from anywhere. This brings in a host of security threats associated with the web application.

Several attacks are happening today in such systems as discussed in Chapter 2. This chapter summarizes the security goals of a web-based BEM system and the possible vulnerabilities and threats that need to be addressed. Following this, the threat modeling for the prototype BEMOSS implementation and also enumerates the security implementation for the prototype BEMOSS is discussed.

This chapter also discusses the security goals of the user interface layer of BEMOSS, possible mitigation strategies that can be implemented to prevent attacks, implementation of these mitigation strategies in the prototype implementation and evaluation of the implementation.

8.1 Security Goals

In a BEM system, the user interface layer is the end user facing layer. Modern end user implementations are web-based and are accessible from anywhere in the world using the Internet. This flexibility comes with a host of security risks and threats that are widespread in the Internet space today. The user interface layer is considered a web-based platform in the following discussion.

A web-based application is vulnerable to a host of infections, Trojans, malware, malicious scripts, etc. It is essential that a web-based interface be built with the ‘security-first’ design model; where security is enforced as part of the system design itself and not as a mere afterthought.

Security goals are high-level objectives, one or more of which are achieved by implementing a security system or technique. The major security goals for a BEM system can be represented in the CIA (Confidentiality, Integrity, and Availability) triad [77], shown in Figure 8-1. CIA is a widely used benchmark for evaluation of information systems security, focusing on the three core goals of confidentiality, integrity, and availability of information.

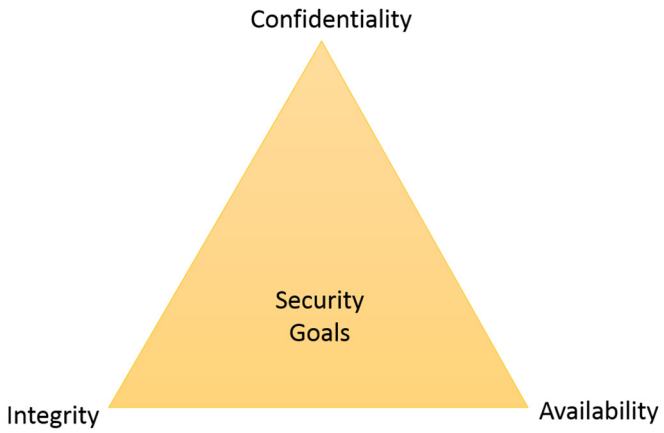


Figure 8-1 CIA triad [77]
Source: http://www.cyber-51.com/cyber51-blog/145-cia-triad [Used under fair use, 2015]

8.1.1 Confidentiality

Confidentiality refers to limiting information access and disclosure to authorized users -- "the right people" -- and preventing access by or disclosure to unauthorized ones -- "the wrong people." It is the property of content of a communication happening between two parties being unable to be retrieved by an interceptor or an eavesdropper.

Encryption is one of the key components in protecting information. Encryption ensures that only the right people can read the information. Access Control Lists (ACLs) are also used to ensure confidentiality.

In the context of a BEM system, confidentiality is one of the most important of all security goals. BEMOSS can be accessed within the network or wirelessly using the external public network. BEMOSS itself has multiple nodes that need to be connected to each other, and confidentiality in information transfer needs to be ensured. It is easier to tap into a wireless network, and read messages transferred between wireless devices. BEMOSS is designed and developed as a wireless platform, which can be attacked if it is not immune to such passive attacks.

8.1.2 Integrity

Integrity is the guarantee that data is protected from accidental or deliberate (malicious) modification. Integrity refers to the trustworthiness of information resources.

Integrity is a key concern, particularly for data passed across networks. It includes the concept of "data integrity" -- namely, that data have not been changed inappropriately, whether by accident or deliberately malign activity. It also includes "origin" or "source integrity" -- that is, that the data actually came from the person or entity you think it did, rather than an imposter.

In the case of multi-hop communications where data passes through intermediate routing devices, data can be altered. Another way of conducting such an attack would be to record a legitimate transmission and replay it later from an illegitimate device. This is known as the ‘replay attack’.

Wireless networks are susceptible to replay attacks. An external device can easily inject data into a network by making a transmission on the shared wireless medium. This can lead to the recipient being duped about the legitimacy of the message received.

Integrity for data in transit is typically provided by using hashing techniques and message authentication codes.

8.1.3 Availability

Availability of information refers to ensuring that authorized parties are able to access the information when needed.

Availability, like other aspects of security, may be affected by purely technical issues (e.g., a malfunctioning part of a computer or a communications device), natural phenomena (e.g., wind or water), or human causes (accidental or deliberate). The most common attack is the Denial of Service (DoS) attacks. The primary aim of DoS attacks is to deny users of the website access to resources of a website. Such downtime can be very costly.

In a system like BEMOSS, the Information Exchange Bus (IEB) can be bombarded with messages that can cause the message queue to overflow and the agents / user platform to lose messages that are intended to be received and acted upon.

8.2 Vulnerabilities

An end-user facing web-application that is part of a BEM system is vulnerable to a host of possible vulnerabilities and other security threats. The OWASP (Open Web Application Security Project) [78] has identified the top web application security risks.

Such applications usually communicate with the smart devices (that need to be monitored and controlled) using a communication interface – usually a RESTful service or a message queue depending on the application requirements. The communication happens using a messaging queue or web services in most cases.

Vulnerabilities are attributed to not only the web application and platform implementation mistakes, but also the network on which the system resides. If the system is installed on the mainstream network in a building, instead of a Virtual segment, it can be a security risk. A secure and isolated network which other users cannot access is important for such an implementation.

This section discusses possible security vulnerabilities and outlines countermeasures in a BEM system communications platforms. It also discusses the broad outline of possible security measures that can be implemented to prevent attackers from exploiting these vulnerabilities.

It is to be noted that this section does not discuss the physical infrastructure requirements/threats that should be fixed to enhance security. This is discussed in the Section 8.2.11.

8.2.1 SQL Injection

SQL Injection [79] is caused by unchecked user inputs being passed to a server database for execution. The attacker might embed SQL queries in the input fields of the application. When exploited, a SQL injection may cause unauthorized access to sensitive data, updates, or deletions from the database, and even shell command executions.

An attacker alters a web page parameters (GET/POST data or URLs) to insert arbitrary SQL snippets that an immature web application executes in its database directly. This vulnerability mostly occurs when constructing SQL queries from a user input without sanitizing the input. For example, in function to filter the contact information, the user-provided the username is queried to retrieve contact information as shown below:

```
user_name = request.GET['username']
query = "SELECT * FROM user_contacts WHERE username = '%s';" % user_name
#execute query here.
```

A malicious user can obtain more information than intended by cleverly crafting the username input. If the user inputs *"' OR 'a'='a'"* in the input field and submits it, the above query would look as follows:

```
SELECT * FROM user_contacts WHERE username = ' ' OR 'a' = 'a'
```

This query would return every single row of the table is returned, which is too much of information leak, because of the failing to check the user input. If the user input was modified as *" ';DELETE FROM user_contacts WHERE 'a' = 'a'"*, this would delete the entire contact list from the table.

```
SELECT * FROM user_contacts WHERE username = ' ' ; DELETE FROM user_contacts
WHERE 'a' = 'a'
```

Countermeasures

To prevent SQL injection attacks, one can use a safe API that provides a parameterized interface. Abstracting the query using a parameter-based model API will eliminate the risk of Injection. Another countermeasure would be to sanitize user inputs, by escaping special characters using a specific escape syntax for that interpreter.

8.2.2 Cross Site Scripting (XSS)

Cross Site Scripting (XSS) [80] attacks are a type of injection in which malicious scripts are injected into otherwise harmless and trusted webpages. XSS allows an attacker to embed malicious JavaScript, VBScript, ActiveX, HTML or Flash into a vulnerable dynamic page to fool the user, executing the script on a host machine in order to gather data. The use of XSS might compromise private information, manipulate/steal cookies, create requests that can be mistaken for those of a valid user, or execute malicious code on end-user systems.

The data are usually formatted as a hyperlink containing malicious contents that are distributed over any possible means on the internet. It allows the attacker to inject client side scripts into the browser of the victim. This is usually achieved by storing malicious scripts in the database where it will be retrieved and displayed to other users, or by getting the user to click a link, which will cause the malicious JavaScript code to be executed by the browser. XSS attacks can be originated from any untrusted source of data (e.g., cookies, web services, etc.) whenever the data are not validated before being included in a page.

XSS attacks occur in the following common scenarios:

i) Data enter a web application through an untrusted source, say, a web request.
ii) The output from an application to a browser is not HTML encoded.

Such attacks if successful can gain access to account credentials. By exploiting XSS vulnerabilities in an application, an attacker can perform attacks, like hijacking the account, spread web malware, Trojans, etc., access browser history and clipboard contents, control the browser remotely without the victim's knowledge, scan and exploit intranet appliances and applications.

For example, consider the following URL that contains a script that is delivered to the user through clicking on a malicious link. The XSS request is initiated from the victims browser, sent to the vulnerable web application, and then reflected back to execute in the user session.

http://www.unssafebank.com.../search.jsp?q=<script>x=new Image;x.src=http://evildomain.com…/sessionhijack.jsp?session-cookie=+document.cookie;</script>

Countermeasures

Such attacks can be mitigated by identifying and preventing XSS errors in web applications.

i) Validate all user inputs from the browser to the web application in the server.
ii) Encode all output sent from the server to the browser. (Convert special characters to their HTML or URL encoded equivalents; URL encodes user input returned as part of URLs.)
iii) Allow users to disable client side scripts. Although this is a development tradeoff, using appropriate plugins, their restrictions can be nullified for specific applications.)

8.2.3 Cross-Site Request Forgery (CSRF)

Cross Site Request Forgery (CSRF) [81] occurs when a malicious site can cause a victims browser to make a request to your server that causes a change on the server. The server thinks that because the request uses cookies, the user intended to submit the form. For example, if an application allows a user to submit a change request that does not include anything secret.

http://bemossinsecure.com/app/changethsetpt?thmode=HEAT&deviceId=1THands132ASJ

The attacker constructs a request that will perform the above operation from the victims account, and then embeds this attack in an image request or a frame stored on various sites under the attacker's control.

<img
src="http://bemossinsecure.com/app/changethsetpt?thmode=HEAT&deviceId=1THands132ASJ" width="0" height="0" />

If the victim visits any of the attackers sites while already authenticated to bemossinsecure.com, these forged requests will automatically include the user session information, authorizing the attacker's request.

Countermeasures

Preventing CSRF attacks require a special unpredictable token in each HTTP request. Such tokens should be unique for every user session and should be cryptographically strong. This token can be included in a hidden field. This causes the value of the token to be sent in the HTTP request, and avoiding its inclusion in a URL.

For GET requests, the user can be asked to re-authenticate using some special formats, like CAPTCHA, thus preventing CSRF attacks, although inserting this token in a GET request in the URL as a parameter has the risk of token exposure.

One means of ensuring CSRF protection is by using double-submit cookies, which send a random value in both a cookie and as a request parameter. The server verifies if both these values are equal.

8.2.4 Exposed Error Messages

A stack trace is an information leak, which reveals information about the application implementation. Failure to filter sensitive information when propagating exceptions often results in information leaks that can aid the attacker in further exploitation of the application. It might reveal too much information about the source code, and the request information. An attacker may create inputs to expose the internal code structure and mechanism. This might seem harmless initially, but it can lead to the attacker gaining access to the server and may become the root to higher security issues like denial of service attacks.

Countermeasures

The stack trace of an error can be altogether avoided, by either catching the error at the application level and implementing remedial measures or by redirecting the user to an error page based on the particular error that occurred (404, 500, 302, etc.).

8.2.5 Username Enumeration

A BEM system often authenticates a user before providing access to its applications. With the rise of web applications, applications increasingly incorporate these techniques. When a user enters username and password for authentication, the server processes the authentication request, and if the authentication fails, responds with an error message. If this error message indicates 'The

username entered is incorrect', the attacker has the option to enumerate usernames in the system. Exploiting this vulnerability helps the attacker to experiment with different usernames and determine the ones registered in the system with the help of error messages. The attacker can in turn use these usernames and different password combinations to hack into the application.

Countermeasures

Display consistent error messages to prevent disclosure of valid usernames. Deleting usernames created for testing purposes will greatly minimize the hacking due to commonly used username and password hijacking. Displaying 'Incorrect username/password combination' instead of specifying which of the two is incorrect will eliminate such attacks.

8.2.6 Broken Authentication and Session Management

Authentication and session management [82] is essential and critical to web application security. Attacks are possible when the application fails to protect credentials and session tokens throughout their lifecycle. Such flaws can lead to hijacking of user accounts, and cause other privacy violations.

Custom built authentication and session management schemes, which is flawed, can lead to this attack. It may include login, logout and password management, session timeout implementation, secret question, etc.

For example, let us assume that application session timeouts are not implemented. If a user uses a public computer to access the application, and chooses to close the browser instead of logging out of the application, an attacker can open the browser and gain access to the application. Another scenario could be when the passwords are not hashed when stored in the database. In such cases, if the attacker who gains access to the database can steal all passwords that are used in the application.

Countermeasures

XSS flaws should be fixed in order to prevent session id leakage. Passwords, when stored in a database should be hashed instead of plain storage. The user login information should be transmitted over Transport Layer Security (TLS) for additional security. Authentication error messages must be generic to avoid revealing excessive information to the attacker. Brute force attacks must be prevented by limiting login attempts. Session timeout should be implemented, with respect to time and on browser close.

8.2.7 Missing Function Level Access Control

Applications back-ends usually verify function level access rights before making that functionality visible in the user interface. However, applications need to perform access control and authorization checks [83] on the server as well when any function is being requested. If this authorization is not verified, the user must be redirected to a page corresponding to the user's role. An attacker who might be an authorized user can change the URL or some system parameters trying to gain access to the controlled modules of the application. This scenario is common when

appropriate verification is not completed in a role-based access control system. Administrative functions are key targets of such activity.
For example, a page can provide an 'action' parameter to specify a function being invoked, and different actions are invoked based on different roles. If the roles are not verified again then there is a flaw.

Countermeasures

The application should have authorization verification module invoked in each and every function being invoked. The user interface should hide any possible navigation to the unauthorized pages than denying the request after the page has been requested. Frequently checking the functionality of the application with different user roles during development will control attacks in these scenarios.

8.2.8 Unencrypted Communication

Communication over HTTP with sensitive information might not be appropriate for a system that contains sensitive information about a building, and for a system that connects to the grid for the demand-response operations. An attacker can be eavesdropping and listening to all the network traffic in an insecure network, obtaining as much as information as possible.

Countermeasures

HTTPS client authentication is a secure method of authentication. It uses HTTP over Secure Sockets Layer - SSL (i.e., HTTPS), in which the server authenticates the client using Public Key Certificate (PKC). SSL technology provides data encryption, server authentication, message integrity, and optional client authentication for a TCP/IP connection. A private CA can be created to issue certificates in open source per-building (for every unique organization or building using BEMOSS, a private CA can be generated) scenarios.

8.2.9 Unacknowledged Devices in the System

A BEM system has multiple devices from different vendors connected together. Since these devices are in most cases auto-discovered by the system, they get added to the database without the administrator intervention. However, if a device is not intended for building operations, then it is considered harmful to add such a device to the system. This might lead to unforeseen security leaks. Attackers can use this scenario to send control messages to the system, and also gain access to the system maliciously.

Countermeasures

Mitigation strategies involve having a semi-manual discovery process, where the administrator can approve/reject a device when it is discovered by the system. This ensures that only devices that are approved by the user are added to the system creating control parameters. This will ensure that any unauthorized communication is eliminated.

8.2.10 Insecure Communication Sockets

Communication between various layers in a BEM system happens using web services or sockets. Socket communication unless secured, is usually unencrypted and can be easily sniffed by an attacker listening passively over the network. Such communication is not only vulnerable to passive attacks, but also active attacks where the attacker can insert malicious data over communication sockets, or bombard the channel with multiple messages, thus causing denial of service level attacks.

Countermeasures

Such attacks can be prevented by using a secure encrypted connection for communication. TLS encryption will provide encrypted communication similar to HTTPS. It includes server authentication, data encryption, and message integrity.

8.2.11 Linux Platform Security

Several attackers are looking to gain access to the physical machine that runs the application. It is crucial to safeguard the physical machine. Compromising the machine itself will result in total system failure. The physical infrastructure [84] should be protected along with other software level modifications in the machine. The following measures are essential:

- Limited physical access to systems and to the networks they are attached to is essential in a system like BEMOSS. USB ports should be removed and any other accessible media drives should be removed/disabled.
- Usage of tools like John the Ripper, a password cracking tool can help identify any weak passwords for users on the machine. A periodic scan of the machine using such a tool is imperative to prevent unauthorized access to the system. Similar to application authentication, CAPTCHA and other tools used for login can strengthen the security.
- Drive encryption could be used to prevent access via alternate-media-booting (using USB disk or CD/DVDs). A Trusted Platform Module (TPM) may be used.
- The firmware of devices and the BIOS itself should be constantly updated to make sure new security updates are enabled. External device drives should be made inaccessible. Boot device should be restricted. Network boot options should be disabled.
- Configure the Linux machine to auto-download security updates.
- Use of SSH for console access and FTP/SFTP for file access will greatly avoid information leaks via the network.
- Remote login for ‘root’ users of the Ubuntu machines should be disabled. Even for a regular user, disable the permissions to access the system as a ‘root’ user when logged in remotely. Using ‘sudo’ will allow viewing system logs of sudo operations later. Limit the number of user accounts.
- Using the concept of ‘who you are’, ‘what you know’, ‘what you have’ for authentication to the physical machine will greatly avoid misuse of user credentials. Strong passwords and usernames are useful when it comes to preventing guessing attacks.

- Regular network scans using tools like nmap [85] to identify network services offered and any unauthorized services.
- Controlling incoming and outgoing network traffic using a built-in host based firewall to control who/what can connect to the system can limit attack surface. This can help prevent DDoS attacks. Limit SSH (Secure shell) and SCP (Secure copy) logins only from specific IP addresses.
- Tools like *tripwire* to check system integrity can be used to monitor abnormal file system changes. File system scanning tools like chkrootkit, etc., can be used to check for known exploits on a periodic basis.

8.3 Threat Modeling

Hackers are using new techniques to gain access to sensitive data, disable applications, deny services to legitimate users, and administer other malicious activities. Developing secure applications must come with the 'secure by design' model. If security is designed as an afterthought, it might be a burdening task and may overlook some of the nasty security loopholes in the application. For this reason, threat modeling is necessary. Threat modeling is a repeatable process that helps find and mitigate all of the threats in the application. Threat modeling is not a one-time process but iterative, and should be updated for evolving threats and newly identified application loopholes. There are three approaches to threat modeling:

i) Asset-centric
ii) Attacker-centric
iii) Software-centric

Some of the common vulnerabilities in a BEM system that is internet-facing are discussed in Section 8.2. A complete set of web application vulnerabilities is discussed in [86]. Based on the Microsoft's list for threat modeling [87] web applications, the following are the steps involved in modeling the threats in the prototype BEMOSS implementation.

- Identify security objectives: The first step in threat modeling is to identify the security goals and objectives for an application. This includes identifying stakeholders, possible attackers, threats, and the environmental factors that contribute to the above.
- Application Overview: Following the identification of security objectives, an application overview should be generated, that can provide a list of all the important characteristics and entry/exit point in the application to easily identification of threats in the application.
- Decompose Application: From the overview, a detailed application flow listing provides an understanding of the application workflow/dataflow. Once the workflow is identified, it is easy to enumerate threats in the different layers.
- Identify Threats: At this point, the threat identification process begins, and the threats are identified at every point detailed in the application overview. These threats align with the security goals and objectives of the application.
- Identify Vulnerabilities: The different threats are used to identify the vulnerabilities in the application based on the implementation.

This process is cyclic [87] as shown in Figure 8-2 and it is repeated to make sure that the application remains secure to newer attacks and threats.

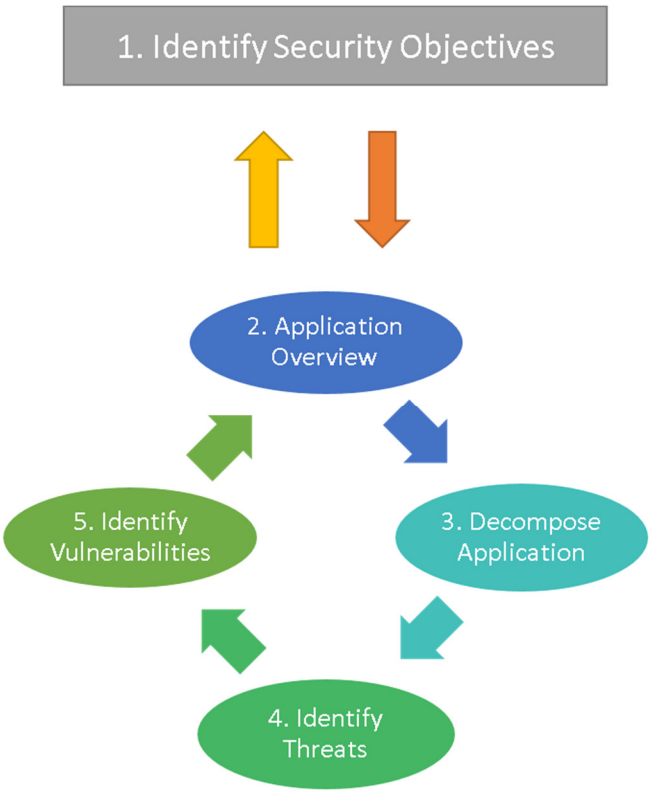


Figure 8-2 Iterative threat modeling
Source: https://msdn.microsoft.com/en-us/library/ff648644.aspx [Used under fair use, 2015]

8.3.1 Security Objectives

Security objectives are goals and constraints related to the confidentiality, integrity, and availability of the data and application. The followings are the security objectives identified for BEMOSS at its first iteration of development

- Ensure application availability
- Ensure reliability of information available to a user in the application
- Prevent unauthorized users from modifying device information or device status
- Prevent modification/leakage of sensitive building related information

8.3.2 Application Overview

An architecture diagram that helps with identification of design level security requirements is shown in Figure 8-3.

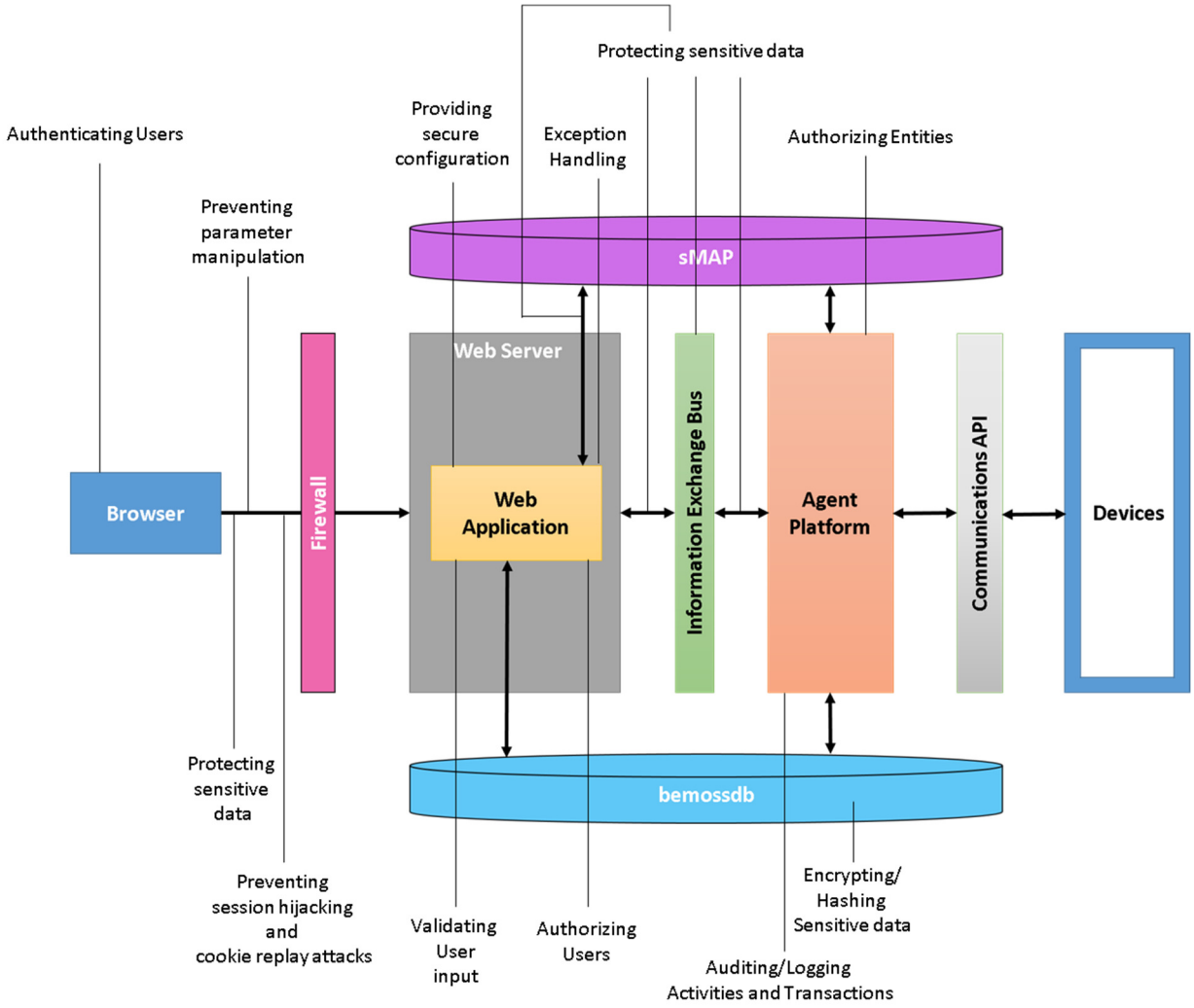


Figure 8-3 Application architecture depicting the possible security design issues

The above diagram gives a complete overview of the areas being considering for security implementation in this thesis. The link between communications API and devices is beyond the scope of this thesis

Authenticating Users: At the browser, a user is authenticated using login parameters. This is the first entry point into the application. This is an area of a security vulnerability that needs to be addressed.

Protecting Sensitive Data: It is important that all the communication between the client and the server is encrypted. Securing this communication layer is important to protect sensitive data including username, password, and other important system information.

Preventing parameter manipulation: If the sensitive information / form data is not protected on transit, they can be manipulated by the adversary and used to attack/gain access to the application.

Preventing session hijacking and cookie replay attacks: An adversary could hijack/steal the session cookies and replay them later posing to be a legitimate user. The application should have definitive strategies to prevent such attacks.

Providing secure configuration: The application connects to the user using a web client (browser) and to the control platform using a message queue. All of these should be configured to ensure that only legitimate users can access/modify information. URL configuration information should also be protected.

Validating User Input: All form data are user input information and should be validated at the server before processing the data.

Authorizing users/entities: All the users should be verified for authorization at every point in the application. This is important to prevent misuse of data or unauthorized amendment of data. For the core platform, only authorized agents should be allowed to publish/subscribe to messages in the application.

Exception Handling: Being a web platform the biggest source of information for a user is the exception dump being thrown on the screen. It leaks sensitive information about the application implementation which an adversary can use to hack the application. All exceptions have to be handled in the implementation and the errors should be abstracted at the user level.

Auditing/ Logging Activities/Transactions: All of the activities in the application (web and control platform) need to be logged for auditing and debugging purposes. This will also provide detailed information in case of an attack on the system.

Encrypting Sensitive Information: Sensitive information like passwords, and other session data need to be stored in encrypted form to prevent loss/misuse of information. Such data is usually hashed or encrypted depending on the needs of the data.

As described in Chapter 6, the application has three primary roles – Administrator, Zone manager and Tenant. Access boundaries are clearly defined for each role.

Key Usage Scenarios Important application scenarios are:

- A new user registers into the system.
- Any of the three role types login to the system
- A tenant views device status information and historical trend in energy consumption of each of these devices.
- A zone manager/administrator controls devices (modify status) using the web application
- An administrator approves new user registration requests
- An administrator approves new devices and adds/marks as ‘NBD’/denies each of them.
- Users log out of the system or close the browser to terminate their application activity.
- An administrator sets the schedule of particular devices.
- An administrator manages devices and their zone allocation in the building

- An administrator modifies nicknames or zones and devices.
- An administrator generates reports from the application.
- An administrator manages user roles from time to time.
- An administrator adds new alarms to the applications
- An administrator views notification logs.
- Other similar scenarios.

Technologies Used The following are the technologies used to build this application:

- Web Server: Tornado
- Web page updates: Web Sockets, JavaScript, jQuery
- Templates: Template tags, HTML, CSS, Bootstrap
- Server platform: Python, Django
- Database: PostgreSQL
- Secure Sockets Layer: To encrypt HTTP traffic

8.3.3 Application Decomposition

Identified trust boundaries are:

- The perimeter firewall
- The database server trusts calls from the Web application.
- The database server trusts calls from the core platform.
- The data access components (ORM) trusts the core platform and user platform models to pass fully validated data.

Some of the identified data flows are:

- User Registration: User registers himself to the application. The registration information is accepted by the web page and validated by regular expression at the user platform server. Information like user full name, preferred username, email address, phone number, and user preferred password is required.

- User Registration Request Approval: User registration request is approved by the administrator. This information is updated in the database. The application sends an automated email providing the login URL for the user.

- User login: User logs in to the web application by entering the username and password. This information is handled by the login web page and passed on to accounts components (app) for authentication. These components pass the data to the models to verify the information. The data access component verifies the information to determine the validity. On successful authentication, a session id is created and the user is redirected to the application home page.

- Monitor/Visualize Historical Data: The user navigates to a device page to monitor the status of a particular device. The information is pulled from the database by the data access layer, and then passed on to the monitor view for the particular device which is then pushed to the template for the user to view. The user does not modify any information in a 'monitor' activity.

- Control: If a user decides to control a device from the web page (administrator role or zone manager role), the user modifies the status, and sends submit. The page pulls the modified information and sends the information to the control component. The control component validates the information and forwards the information in the form of a ZeroMQ message to the core platform. When the core platform responds with a ZeroMQ message, the message is read by the web sockets corresponding to the control page, and the user is notified (success/failure) using a pop up in screen. (The database is updated by the core platform in this case.)

- Schedule: The scheduling information is stored as a JSON file in the user platform. Whenever a user requests the schedule page, the JSON file is parsed and the information is displayed in user readable format to the user. The user can modify the information or add new schedules. The web page validates this information and sends this to the schedule component. The schedule component validates the information and saves the information to the JSON file, and sends a ZeroMQ message to the core platform confirming a change in the schedule and requesting activation of the new schedule.

- Generate Reports: Reports for a particular component can be generated using the 'export to spreadsheet' button. The reports component then communicates with the sMAP historian to retrieve information. Once the information is retrieved, the information is composed into a spreadsheet, and the user is provided the option to download the report from the web page.

- Manage Devices: When the administrator requests some modification of device contents and submits information, the web page collects the modified information and sends it to the manage_devices components after validating the information. The information is sent to the data access component, which updates the database with the new information. In required situations, ZeroMQ messages are sent to the core platform to take appropriate action.

Entry points are:

- Port 8000 for web requests.
- Port 443 for SSL.
- All other ports are restricted by firewall.
- The logon page in the web application, accessible to all users in the internet. (Home page is only accessible to registered and approved users).
- All other pages like zone dashboard, device dashboard, device schedule, device statistics, manage devices, user management, alarms and notifications, network status, etc.

Exit points are:

- Device dashboard, zone dashboard, view information modal, home page, notifications page, statistics page, etc.

8.3.4 Threats

The following threats could affect the BEMOSS application web platform:

- Brute force attacks occur against the data store.
- Network eavesdropping occurs between the browser and web server to capture client credentials.
- An attacker captures an authentication cookie to spoof identity.
- SQL injection occurs, enabling the attacker to exploit an input validation vulnerability to execute commands in the database, and thereby accessing and/or modifying the data.
- Cross-site scripting occurs when an attacker succeeds in injecting scripts.
- Cookie replay attacks, allowing an attacker to spoof identity and access the application as a different user.
- Information disclosure and sensitive exception information is revealed to the client because of an error.
- An attacker manages to take control of the web server, gain unauthorized access to the database.
- An attacker obtains the encryption keys used to encrypt sensitive data (usernames, passwords, device information, etc.)
- An attacker/client obtains unauthorized access to the web server resources and static information.
- An attacker exploits the function level relaxation in user role verification.
- An attacker can modify the messages sent in the message bus or overwhelm the message bus by adding unauthorized information to the message bus.
- An attacker can add an unauthorized agent to the core platform thus confusing the original application flow.
- An attacker could gain access to the sMAP historian because of the vulnerability in the http server of the sMAP historian application.
- An attacker could perform a CSRF attack against the application.
- An attacker can exploit the cross-origin vulnerability in the web sockets.
- An attacker can overwhelm the server with connections thus denying access to the application for legitimate users.

8.4.4 Vulnerabilities

Application vulnerabilities are:

- User password storage.
- Lack of password complexity enforcement.
- Lack of password-retry logic.
- Missing or weak input validation at the server.
- Failure to validate cookie input.
- Failure to sanitize data read from a shared database.
- Failure to encode output leading to potential cross-site scripting issues.
- Exposing an administration function through the Web application.
- Exposing exception details to the client.
- Exposed username/password exception information to the client, unfiltered.
- Not securing the HTTP server using encryption
- Allowing cross-origin headers in the web sockets.
- Failure to segregate role-related web page access at every possible point.

8.4 Security Implementation in the Prototype – BEMOSS

This section discussed the security framework implemented for the prototype implementation discussed in this thesis. Although the user platform alone was designed and developed as part of this thesis, the core platform was also analyzed for security risks and security mechanisms were implemented. Physical infrastructure modifications were also incorporated.

8.4.1 User Authentication

The first step to creating a secure web-application user experience is to enable the user name and password based authentication to the system. A building tenant or engineer needs to first obtain access to the system in order to interact with it. Any user accessing the application is redirected to the login screen where login credentials are requested to get access to the system. This is the first important step towards a secure platform. User registration should be semi-automatic considering the small fraction of users who would be using the system as opposed to all of the building occupants.

This semi-automatic process ensures that only legitimate users are authorized to use the system. Third party logins using Google, Facebook, and similar others were avoided for this scenario since the system was implemented with the interest of local users, and a local network in case of zero internet connectivity. Users cannot access the system from outside of the local network, but some small and medium sized buildings could use this as an option. Since adding on to login with Google, Facebook, etc. is not a necessity this is not implemented in the prototype design. However, they could be implemented as simple applications in the user platform.

Figure 8-4 shows the user registration process as discussed in Chapter 6. The user registration is a two-step process:

- Authenticate a user by checking a username and password against a database of users.
- Verify that the user is authorized to perform some given operation.

Authentication also provides the user with a session for that login period. The session verification and management is discussed later in this Chapter.

8.4.2 Authorization

An authenticated user is authorized to access parts of the application or the entire application based on role. Role management is based on user groups. A general BEM system may define three roles in the system – tenant, administrator, and zone manager. The assumption here is that the building operator and building engineer assume the role of an administrator in building operations. The zone manager is given authorization to control devices of the zone assigned to him. The tenant has read-only access to parts of the system

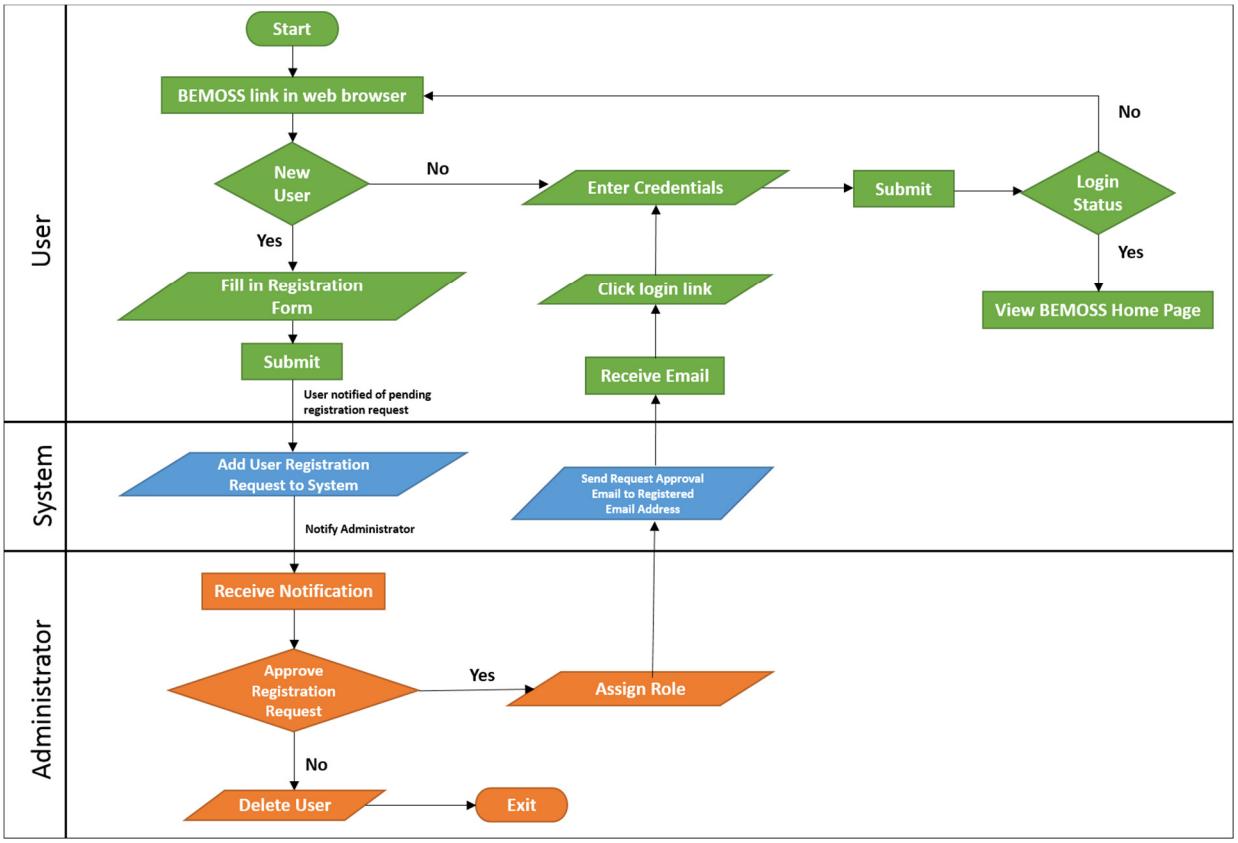


Figure 8-4 User registration process

Static Role

The administrator has maximum privileges in the system and has access to all of the pages in the web application. The tenant has the least privileges – only read only access, and does not have access to any of the administrator control panel pages or general system settings. The only customization allowed is the background color change from the list of shades available.

Dynamic Role

The zone manager role is dynamic. It is created based on the need and can be added or removed anytime based on the requirements for building operation. The administrator chooses to assign a user with the role of a zone manager, and also specifies the zone which needs to be managed. The zone manager role then gets created and allocated dynamically. This process is described in the algorithm shown in Figure 8-5.

```
username = 'a'
zone_id = n
n_role = 'zone_manager'
x = determine need for managing a zone.
if (x):
      current_role = get_role(username).
      role[name] = n_role.
      role[zone] = zone_id.
      current_role = role.

# Authorization in web pages
in zone_id:
      if (username[role][name] == n_role
                  && username[role][zone] == zone_id):
            enable n_role control access.
      else:
            treat as 'tenant'.
```

Figure 8-5 Dynamically assign zone manager – flowchart

In general, the user is authorized for every page access control elements access. However, this leads to security risks as discussed in Section 8.2.7. To address this security risk, every function ensures that a user is authenticated, and also ensures that a user has the right to access the page. A user is redirected to the home page if the system denies authorization to a particular feature. Checking for user authentication and authorization at every function ensures that there is no security risk of an attacker gaining access into the system maliciously.

Aside from this authorization factor, the database also stores permissions assigned to different users and roles and their permissions in the system to perform certain operations. Although the system does not need to use this level of indirection to ensure authorization, this feature is available to allow administrators to have further control over users who have control/insert/update/delete access to the system database as described in Section 7.2.1.

8.4.3 User Operational Log

A log of user operations such as login, logout, and session information is necessary to make sure only authorized personnel access the system. The prototype design has a data table that logs the users' session information including session id, login date and time and logout date and time.

8.4.4 Session Management

The session framework allows retrieval of necessary information based on the user that is logged in. It stores data on the server side and abstracts the sending and receiving of cookies. Cookies contain a session ID – not the data itself. Sessions are implemented using the session middleware. In the prototype implementation, the session is backed by a data table in the server, which is used to manage sessions. Every request from a web page contains the session information. For example, a GET request from a localhost server installation of the prototype implementation has the following header information.

GET http://10.0.2.15:8000/ **Status:** HTTP/1.1 200 OK	
Request Headers	
Accept	text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Encoding	gzip, deflate, sdch
Accept-Language	en-US,en;q=0.8
Cookie	sessionid=1a01679167c41e193370401b1dc5ec7c; csrftoken=Sqr3PsFwHFPS4phqt5jK0lHyeLdqO0eC; style=e
User-Agent	Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/41.0.2272.118 Safari/537.36
Response Headers	
Content-Length	49440
Content-Type	text/html; charset=utf-8
Server	TornadoServer/4.1
Set-Cookie	csrftoken=Sqr3PsFwHFPS4phqt5jK0lHyeLdqO0eC; expires=Mon, 11-Apr-2016 21:58:05 GMT; Max-Age=31449(
Vary	Cookie

Figure 8-6 GET request and response headers

The session ID in the Request headers enables the session related information to be saved and accessed. For the prototype implementation, the session is set to expire at close of browser. This makes sure that an attacker does not misuse the session ID in case the user does not log out of the session, but closes the browser.

A session is cookie-based – the session IDs are stored in cookies. It does not fall back to putting the session IDs in URLs. This method is secure as using the session IDs in URLs makes it vulnerable to session theft using 'Referrer' header.

Saving Session ID

A session is stored as a data model in the metadata storage system. Each session is identified by a pseudo-random 32 character hash stored in a cookie. Since it is a model, it can be accessed just

like any other database table. The prototype system using the model API to do this access. Decoding the session ID gives the user ID in the JSON format, which can be used to perform required operations.

Database-backed sessions vs. Cookie-based sessions

Using database-based cookies is safer since it defines a next level of indirection for an attacker trying to misuse the session. A cookie-backed server session on the other hand might be susceptible to replay attacks, since the cookies are not freshness guaranteed and do not usually have an expiration date. This means that an attacker who has access to your session id while you were still logged in can store cookies and access your account even after you are logged out.

Accessing the database to read the cookie expiration date is heavy on the server compared to completely session based cookies, but it makes the session more secure for the legitimate user.

Table 8-1 Data table fields for sessions table

Field	Description
session_key	The encoded session key value
session data	Session related data
expire_date	Session expiry date and time information

Figure 8-7 shows the session information for three sessions stored in the database.

	session_key [PK] character varying(40)	session_data text	expire_date timestamp with time zone
1	1a01679167c41e193370401b1dc5ec7c	MDM4ODE3ZGYwZjgyOGMxNzVkMjk0Y2JmOGZkYWE3NTQwN2IwODYwNjqAAn1xAShVEl9hdXRoX3Vz	2015-04-27 11:55:50.036239-04
2	cfa41e08fe410bf61009ab1a8369781a	MDMzMThmOWUyY2ZiZGRlNzk0OWRlOGM4ZWNhNjk1NTU1OTc5MzMyNDqAAn1xAS4=	2015-04-27 11:48:21.126212-04
3	fbe8f28222ee5f226be76baf37eb2969	MDM4ODE3ZGYwZjgyOGMxNzVkMjk0Y2JmOGZkYWE3NTQwN2IwODYwNjqAAn1xAShVEl9hdXRoX3Vz	2015-04-23 11:05:03.527156-04

Figure 8-7 Session information stored in a data table

8.4.5 Cross Site Request Forgery (CSRF) Protection

CSRF was discussed in length in Section 8.2.3. A CSRF attack relies on the fact that the browser manages cookies, and will include cookies associated with a target domain to the forged HTTP request. The user platform uses a Double Submit Cookie. This is considered enough protection from a CSRF attack since it is impossible for an attacker to control the cookie filed in a CSRF attack.

A Double Submit cookie is defined as sending a random value in both a cookie and as a request parameter, with the server verifying the cookie value and the request value. If the verification fails, the page is redirected to a 403 error screen as shown in Figure 8-8.

Figure 8-8 Unmasked CSRF error screen

In the prototype implementation, a CSRF cookie is set in the HTML page using a 'hidden' field.

<input type="hidden" name="csrfmiddlewaretoken" value="{{ csrf_token }}">

This ensures that the server receives the CSRF token without any malicious code that modifies the token value. This process is implemented in every page that is rendered to the user.

While the above method covers any POST requests using a form, it is not possible to use this when sending an AJAX request to the server. For this reason, custom JavaScript is written that acquires the CSRF token and sends it to the server for every AJAX request sent.

The mechanism used to ensure CSRF protection is as follows. A CSRF cookie is generated as a session independent nonce value. Other sites do not have access to this cookie. The safety of this technique also relies on how random the CSRF token is. The following algorithm was used to ensure that CSRF token is not guessable and cannot be obtained by brute force attacks. The current state of the Pseudo Random Number Generator (PRNG) is captured, and combined along with current time and SECRET_KEY value for the application and hashed using the SHA256 hashing algorithm. The hashed value is used as the seed for the random number generator. The random number generated as a result is used as the CSRF token.

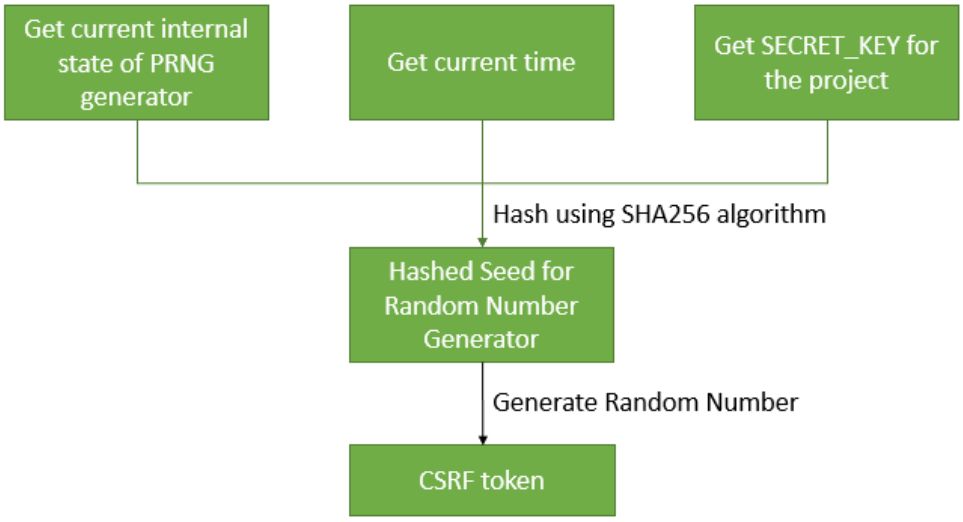


Figure 8-9 CSRF token generation algorithm

The random number generator used in this context is the Mersenne Twister Algorithm [88]. This cookie is set by the Middleware in the User platform.

i) Hidden form fields are present in the HTML pages that contain this CSRF token. For every AJAX request sent, a CSRF token is included in the POST request.

ii) For all the incoming requests that are not HTTP GET, HEAD, OPTIONS or TRACE, a CSRF token is required, and a field in the HTML page 'csrfmiddlewaretoken' must be present. Failure to have one of these keys sent to the server will result in a 403 error.

Secure Request (HTTPS)

For HTTPS request, the Referrer Header is verified. When using a session independent nonce, a man-in-the-middle attack is possible. The attacker will need to provide a CSRF cookie and token. The attacker can circumvent the CSRF protection while attacking as a man-in-the-middle, because even a HTTP Set Cookie header is accepted by clients that are talking to an application using HTTPS. If there is no Referrer header, the request is rejected in case of HTTPS. Under HTTPS, the Referrer header is missing for same-domain requests in only about 0.2% of cases or less [89], so using Referrer header for checking a HTTPS request ensures CSRF protection.

This ensures that only forms and requests that originated from the actual web application in question can be used to POST data back.

GET requests are ignored for CSRF checks, because GET requests never have any potentially dangerous side effects. POST, PUT, DELETE is considered 'unsafe' and all other methods are unsafe too, except GET.

8.4.6 Cross Site Scripting (XSS) Protection

Cross Site Scripting is defined as one of the primary security loopholes in many web-based applications where user input is taken into the system un-sanitized. When generating HTML from templates, there is always a risk that a variable will include characters that affect the resulting HTML.

In the prototype design, each untrusted variable is run through an escape filter that converts potentially harmful HTML characters to safe elements. Table 8-2Table 8-2 shows the XSS protection variable conversion map for XSS protection.

Table 8-2 XSS Protection variable conversion map

Variable	Converted symbol
<	<
>	>
‘	'
“	"
&	&

Additional elements are also implemented to prevent this attack. In most cases, a user input field (a text field, text area) has been avoided. In an energy management system, user inputs are required and standardized. These standardized inputs have minimum use of text fields where users are required to type in a value. This way, most of the causes for XSS attack can be avoided. In other cases, a client-side and server-side verification of the user input is performed. Even if one of them fails, the user is notified of the error in his input.

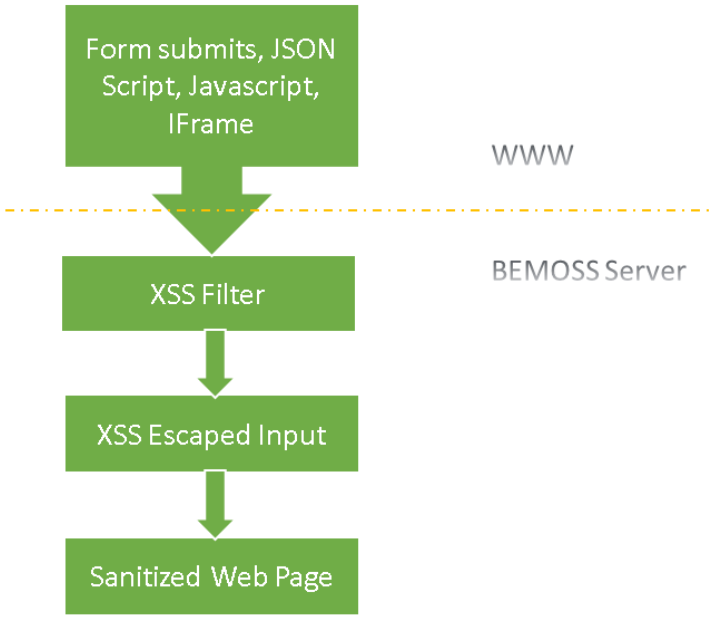


Figure 8-10 XSS prevention flowchart

A regular expression (a sequence of characters that form a search pattern) is used to perform such a test at both client and server side.

```
^[A-Za-z0-9_]*[A-Za-z0-9][A-Za-z0-
```

The only area in the web application that involves user input validation is for nicknames (device nicknames, zone nicknames, schedule nicknames). All other inputs from users are curated and do not involve the user entering any form of text. The above regex (regular expression) is the only one in the entire application that is required to perform user input validation. In other cases, the user input is obtained in the form of button clicks which alter the value of a field and similar other techniques.

In most cases that involve a device status to be modified, the user input is sent to the core platform for processing the data. The data is sanitized at the user platform level before sending it to the core platform. The data is not stored in the metadata storage unless it is something that is specific to the user needs (like nicknames, etc.). The sanitized data is sent to the core platform and device status is updated.

8.4.7 Password Management

One of the direct ways to attack a web-based application is password storage. Forcing strong passwords are always a good way to ensure application security, but storing the password is also important. To begin with, during the registration process, the user should be forced to pick a strong password, and strong password choosing mechanism should be elaborated.

Once a user picks a strong password, it is essential to channel this password to the password storage system without the password getting leaked anywhere in between via JavaScript or HTML or server side code. One of the preferred mechanisms to store password is hashing. Encrypted store can also be used, but it can always be broken. Because hashing is a one-way operation and cannot be reverted, it is best to use hashing to store passwords.

In the prototype design, password is hashed and the hash is stored in the database. The password attribute of a user model is a string in the following format:

<algorithm>$<iterations>$<salt>$<hash>

The above string indicates the components used to store a password in the database. There are varieties of algorithms that can be used for password hashing, but the prototype design uses the PBKDF2 algorithm with a SHA256 hash which will be explained shortly. Iterations describe the number of times the algorithm runs over the hash. The random seed that is used is the salt and the hash is the result of the one-way function. A salt is random data that is used as an additional input to a one way cryptographic function that hashes a password. Using salts prevents dictionary attacks using a list of password hashes, and other rainbow table attacks.

NIST, a standards body recommends a password stretching mechanism using the PBKDF2 algorithm with a SHA256 hash. This mechanism is secure and would involve massive amount of computing time before it can be broken. For the purposes of a BEM system, this should be sufficient. The PBKDF2 algorithm [90] uses a number of iterations of hashing. This slows down attackers making brute force attacks against hashed passwords. The salt used is a cryptographically secure nonce in ASCII. The prototype implementation uses 24000 iterations, and uses HMAC with SHA256 for the digest.

PBKDF stands for Password-Based Key Derivation Function. Key derivation functions are deterministic algorithms used to derive cryptographic keying material from a secret value (password). The algorithm can be described as follows:

The input to the algorithm includes the user-given password (P), a salt (S) and key length information (kL). The master key (MK) is obtained as follows:

$$\mathrm{MK} = \mathrm{PBKDF}_{(Pseudorandon\ function, Iteration\ count)}(\mathrm{P}, \mathrm{S}, \mathrm{L})$$

The algorithm [90] is shown in Figure 8-11. When the application receives a password and username at login, the hashing operation is performed and the resultant hash value is compared with the password hash that is stored in the database. If the two hashes match, the user is logged in to the system. With hashing, the application never needs to store the password in clear text in any part of the application. Although hashing is irreversible, password based hashes are vulnerable to brute force attacks and dictionary attacks. To combat this the salt is used. Since the salt is a pseudo random seed used in the password hash generation, the attacker can realistically not be able to break the password.

Algorithm for Password Hashing

Input:

P → Password from user
S → Salt
C → Iteration count
kL → Length of Master Key

Output:

MK → Master Key

Parameters:

Prf → Pseudo Random Function
hL → Digest size of the hash function
l → number of octet blocks in the derived key
r → number of octets in the last block
T<0,1,..,r-1> → Truncation of binary string T that retains its first r bits
Int(i) → 32 bit encoding of Integer i with the most significant bit on the left

Algorithm:

If $(kL > (2^{32} - 1) * hL)$:
Output error: "Derived key too long"
Else:
$l = \left\lceil \frac{kL}{hL} \right\rceil$
$r = kL - (l - 1) * hL$
For $i = 1$ *to* l*:*
$T_i = 0$
$U_i = S \,||Int(i)$
For $j = 1$ *to* C:
$U_j = HMAC(P, U_{j-1})$
$T_i = T_i \oplus U_j$
Return MK $= T_1 \,||\, T_2 \,||\, \ldots ||T_l < 0 \ldots r - 1 >$

Figure 8-11 PBKDF2 algorithm

8.4.8 Error Handling

With web applications or general desktop applications, the error dump on the screen is detailed enough for a developer to examine the cause of the error. Such detailed information is a good development tool, but it reveals too much information to the user while in production. For this reason, it is important to abstract away error information from the end user.

The prototype implementation abstracts away all error messages to secure away the source code and implementation details from the end user or an attacker. In general, all errors, like 404, 500, 302 etc. were captured in the server, and the user was redirected in such cases to the corresponding error pages. For example, when a page that is not part of the regular URL configuration is accessed,

it is redirected to a 404 error page (See Figure 8-12) abstracting the server process. Figure 8-13 shows the error page when error abstraction is not available.

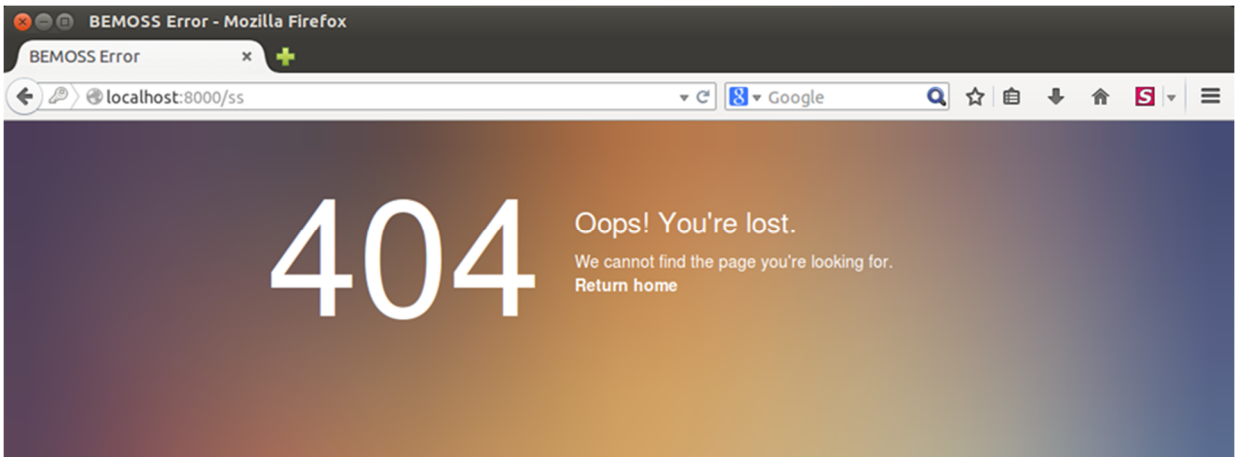


Figure 8-12 404 Error page with link to home page
Source: http://www.bemoss.org. Used with permission, 2015.

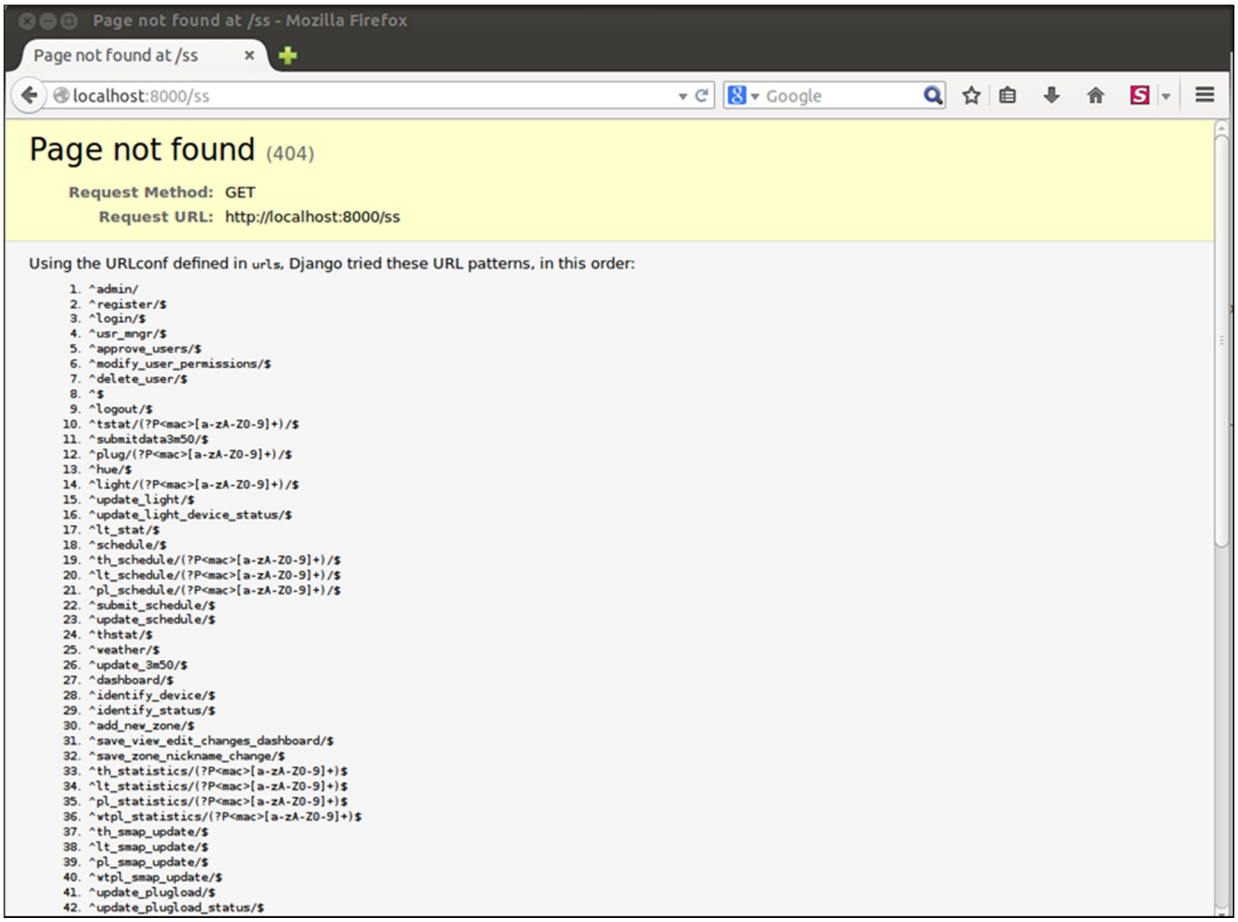


Figure 8-13 A 404 detailed error page

The above figure gives implementation information about the type of URL configuration the application searched through to find out that the entered URL does not exist. Some of these are potentially dangerous resources to be available to the end user or an attacker, alike.

In some scenarios, the user may have just the 'tenant' role, but may know that the URL http://<hostname>:<portnumber>/usr_mngr can lead to user manager page. The prototype implementation redirects the user to the home page, if such requests are made.

In other cases, like the user typing incorrect password or username, the implementation carefully avoids giving away exact details about the username, as discussed in Section 8.2.5. In case of an incorrect user name or password, the user is given an uninformative message like – 'Incorrect username/password combination'. This message does not specifically say if the username was incorrect or the password was incorrect but provides a general message. This acts towards abstracting away implementation information from the user or an attacker.

8.4.9 Securing the web application using TLS

To allow no compromise of user credentials and sessions, SSL encryption is necessary. An SSL certificate installed on the server encrypts traffic end-to-end between client and server. Although use of SSL slows down the application a little, it is important to have a secure communication between various layers of the site. Using HTTPS protocol instead of insecure HTTP will keep the data safe from eavesdroppers.

In the prototype implementation, OpenSSL was used keeping in mind the completely open source nature of the application. While generating a self-signed certificate using a private Certificate Authority generated using OpenSSL may serve the purposes of demonstration and small businesses in some cases, obtaining certificates from an actual CA will ease the problem of browser issues on using an unknown CA.

Authentication and Authorization are the first steps towards a secure web application. Certification requires a trusted third party that holds information about each subscriber. The certificate affirms that the holder is who it says it is and holds several pieces of information – holder name, issuer, and expiration date, and so on.

This certificate is signed in a process similar to watermarking checks, which demonstrates authenticity. The SSL uses certificates as part of the handshake between the client and the server. When the client connects to the server, they exchange a series of messages that do much more than just authenticating each other. During the handshake [91], the following information is exchanged:

- Agree on the version of the protocol to use.
- Select cryptographic algorithms
- Authenticate each other by exchanging and validating digital certificates.
- Use asymmetric encryption techniques to generate a shared secret key. SSL/TLS use shared key for symmetric encryption of messages, which is faster than asymmetric encryption.

The protocol is shown in Figure 8-14.

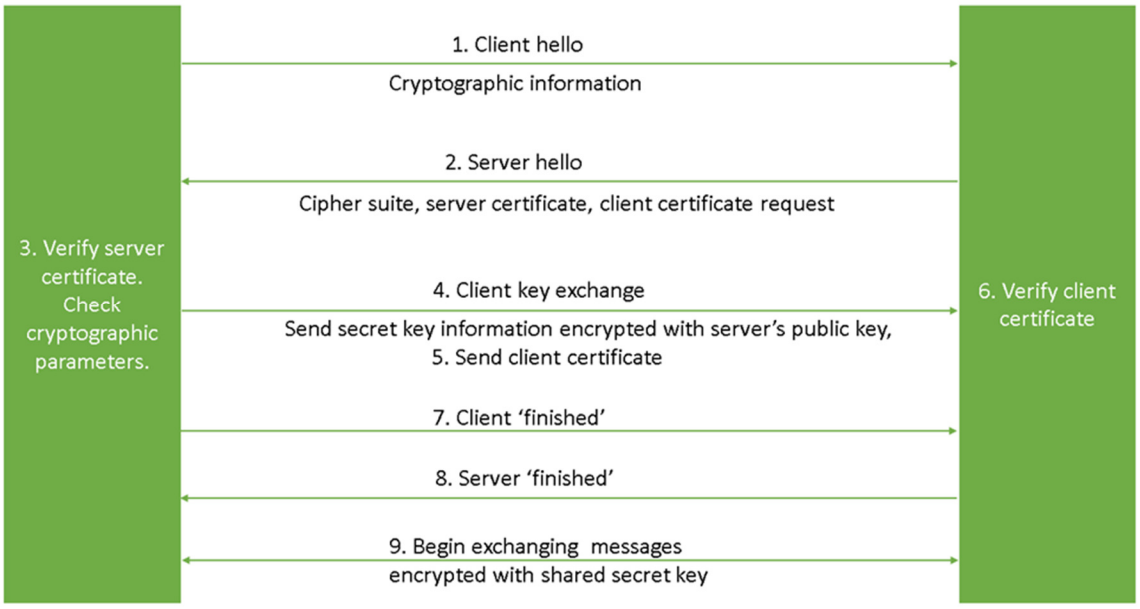


Figure 8-14 SSL/TLS handshake

In the prototype implementation, BEMOSS acts as its own certificate authority (CA). Since BEMOSS is mostly internal to a building, issuing its own certificates is a reasonable way to ensure security, being open source. Therefore, BEMOSS is its own CA. Being a CA means dealing with cryptographic pairs of private keys and public certificates.

BEMOSS being its own CA needs to be registered with the browser the first time a user logs into the site. Connections between the client and server are encrypted using AES. The key size used for encryption is 128 bits. This can be modified to 256 to make it more secure. It operates on the Galois/Counter Mode (GCM) [92], which is used for authenticated encryption. Figure 8-15 shows the site encryption for BEMOSS installation.

The certificates and keys are added to the server and the server is configured to start up and function in the encrypted mode. The same function can also be performed with a verified certificate authority like Verisign, GlobalSign, if required.

The BEMOSS application when accessed in the insecure mode is redirected to the secure client, thus preventing accidental use of unencrypted connection.

8.4.10 Web Sockets Security

A web application in general might not be a combination of so many different technologies. However, with the evolution of web sockets, more and more web applications are adopting web sockets to enable push for web page updates in their framework. The prototype implementation is all about live updates, and showing up to date information to the user.

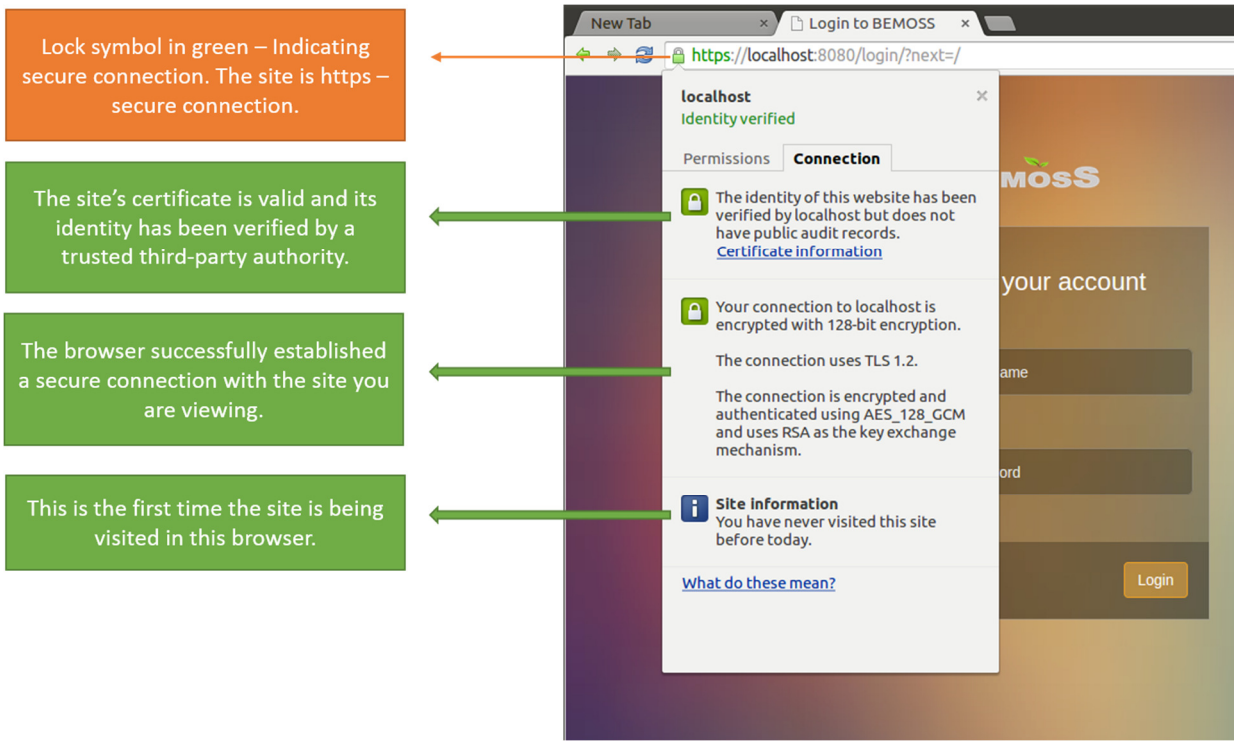


Figure 8-15 Anatomy of the HTTPS site encryption and authentication specification
Source: http://www.bemoss.org. Used with permission, 2015.

Whenever a user requests updates, the request is sent via the server, and the response is picked up by web sockets in the web page. The page is then updated for the user. The page updates are asynchronous and achieved using web sockets and AJAX request response modules.

Similar to encrypting HTTP over SSL/TLS, web sockets can also be encrypted using SSL/TLS. This protects the system again the man-in-the-middle attacks. Tunneling any important requests through web sockets has been avoided. Tunneling requests like database connection can lead to an XSS attack thereby leading to the breach of the entire application.

Client inputs through a web socket are completely avoided. All client inputs are validated at the client side and the server side, and the requests are redirected to the core platform via the server. No client requests are sent through the web sockets – thus the application cannot be infected via a client side web socket request. The same-origin policy that rejects any requests or pings from cross-origin servers is implemented.

8.4.11 Preventing Brute Force Attacks

Brute force attacks refer to exhaustive key search method is a cryptanalytic attack that can, in theory be used against encrypted information. Unlike regular hacking that focuses on vulnerabilities in software, a Brute Force Attack aims at being the simplest kind of method to gain access to a site: trying usernames, passwords, over and over again, until the password is cracked.

Although this is inelegant, when unsafe passwords are used, these attacks are easily possible. Due to the nature of these attacks, the server memory is used up often causing performance issues. This is because the server receives far more number of http requests than anticipated, resulting in the server running out of memory. In the prototype implementation, the following steps have been taken to prevent brute force attacks:

Enforce Strong Passwords

The users are forced to set strong passwords. The following specification has been used for ensuring password strength:

- Minimum password length: 6
- Maximum password length: 120 (Can be set to none.)
- Password dictionary: This is a simple text file that contains a list of commonly used passwords that have to be avoided. The password that the user enters is verified against this password to make sure that commonly used passwords are avoided.
- Password Match: Repeat password feature is implemented where the fuzzy match threshold is used (0 if the password does not match; 1 if the password matches and other values between 0 and 1 for in between values.)
- Password complexity: The definition of a strong password in BEMOSS is determined by the following factors:
    - At least one upper case letter
    - At least one lower case letter
    - At least one number
    - At least one punctuation character
    - At least one special character
    - No use of first name or last name in the password

Lock after failed login attempts

Locking the system after x number of failed login attempts for n minutes is one of the most used techniques to prevent brute force and Denial of Service attacks. If a user attempts a wrong password for more than x times, the account is locked for n minutes before the user can attempt login again.

In the prototype implementation, the number of login attempts before an account lockout is set to 3. The cool off period after which the account will be re-enabled is set to 10 minutes. This is achieved with a ‘signal dispatcher’ which allows decoupled functions to be notified when actions occur elsewhere in the framework. The components of a signal are shown in Figure 8-16:

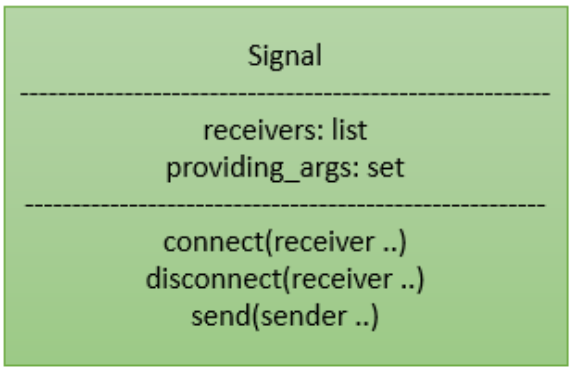


Figure 8-16 Signals pattern

- receivers: A List that has references to all receiving functions or methods
- providing_args: A set of all arguments the signal provides (in this case, cool off, number of attempts)
- If a function needs to receive messages from the signal, it can subscribe or connect to the signal by invoking connect () method of the signal.
- If the function wants to stop receiving, it can unsubscribe or disconnect by invoking disconnect () method of the signal.
- If the function wants to send a message to a signal, it can invoke the send () method of the signal. This message will be received by all receivers that are connected to it.

Automated Public Turing Tests

A captcha is a program that protects websites against computer bots by generating and grading tests that humans can pass but current computer programs cannot. For example, humans can read distorted text similar to the one shown in Figure 8-17, but current computer programs cannot.

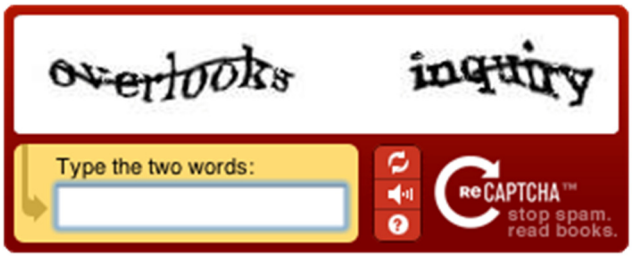


Figure 8-17 CAPTCHA example
Source: http://www.captcha.net/images/recaptcha-example.gif [Used under fair use, 2015]

CAPTCHAs of the above form are the primary CATPCHAs available today and widely used, but the prototype implementation uses simple math questions (as shown in Figure 8-18) to prove that the user is human. Such questions usually comprise of simple addition, multiplication, and subtraction that are easy for a general user to answer. A CAPTCHA is used only in the second

login attempt to prove that the user is human. The first failed attempt is not CAPTCHA enabled considering the chance of a human error.

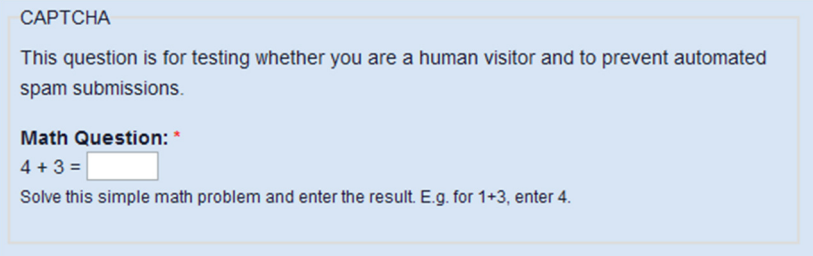


Figure 8-18 Math CAPTCHA to distinguish humans from computer bots
Source: https://www.gachisites.com/files/imce-shared/features/CaptchaMath.gif
[Used under fair use, 2015]

8.4.12 Preventing Denial of Service Attacks

Denial of Service attack is a malicious attempt to make a server or a network resource unavailable to users, usually by temporarily interrupting or suspending the services of a host connected to the Internet. The most common way to attempt this attack involves flooding the target resource with external communication requests. This overload prevents the target from rendering service to legitimate users, or slows the response time. One of the easiest attacks, if left unchecked because of low probability attack ratio for a BEM system may lead to serious consequences.

Algorithm for IP address based DOS prevention:

```
On server startup:
        enumerate banned IPs
        enumerate whitelisted IPs
        ban the IPs in the list
        function process_incoming_request:
                ip_addr = get IP address from request
                if ip_addr is whitelisted:
                        return None
                else if ip_addr is banned or monitor_abuse(ip_addr):
                        return http_response_forbidden

        function monitor_abuse(ip_addr):
                n = number of hits per second for a given IP.
                if n >= IP abuse threshold:
                        ban IP
                else
                        n++
                return ip_abuse_limit_exceeded as True
```

Figure 8-19 Algorithm for IP address based DOS prevention

The prototype takes measures in the software implementation to prevent denial of service attacks. A middleware is implemented in the user platform layer to lock out or banish users by IP address. Users are automatically banned if they exceed a certain number of requests per minute, which is considered a likely form of denial of service attack. The algorithm used in this middleware is shown in Figure 8-19. This algorithm describes the methodology used to ban IP addresses from attacking the site with the DOS motive.
The Section 8.4.11 discusses the implementation against brute force password attacks, which can also become a denial of service attack in case of unlimited tries using a bot.

8.4.13 Device Approval Process

In a BEM system, adding the right and legitimate devices to the building system is of utmost importance. Several devices maybe part of a building, and it is difficult to determine which one belongs to the building operations and which ones were added by an attacker. For this reason, a device approval process is necessary.

In a huge building with several devices, it is not possible to add devices to a building manually. An automatic device discovery process runs in the background discovering new devices as they become available in the network.

Device Approval Algorithm

Device approval flowchart is shown in Figure 8-20 and listed below.

i) Device discovery process adds the newly discovered devices to devices in the database with the status 'Pending'.
ii) The user platform gets these devices and makes them available in the 'Discovered Devices' page in the dashboard.
iii) The administrator manually verifies these devices.
iv) If it is a device that is approved for building operation, the administrator marks the status as 'Approved'. This change sends a signal (message) to the core platform and the platform starts an agent for this device. It is then moved to the 'Manage Devices' section in the user dashboard.
v) If a device does not belong to the building operations, the administrator can mark the device as a 'Non-BEMOSS Device' after which it never appears in the user dashboard.
vi) If the administrator is undecided on further action, the device status will be left unchanged (Pending).

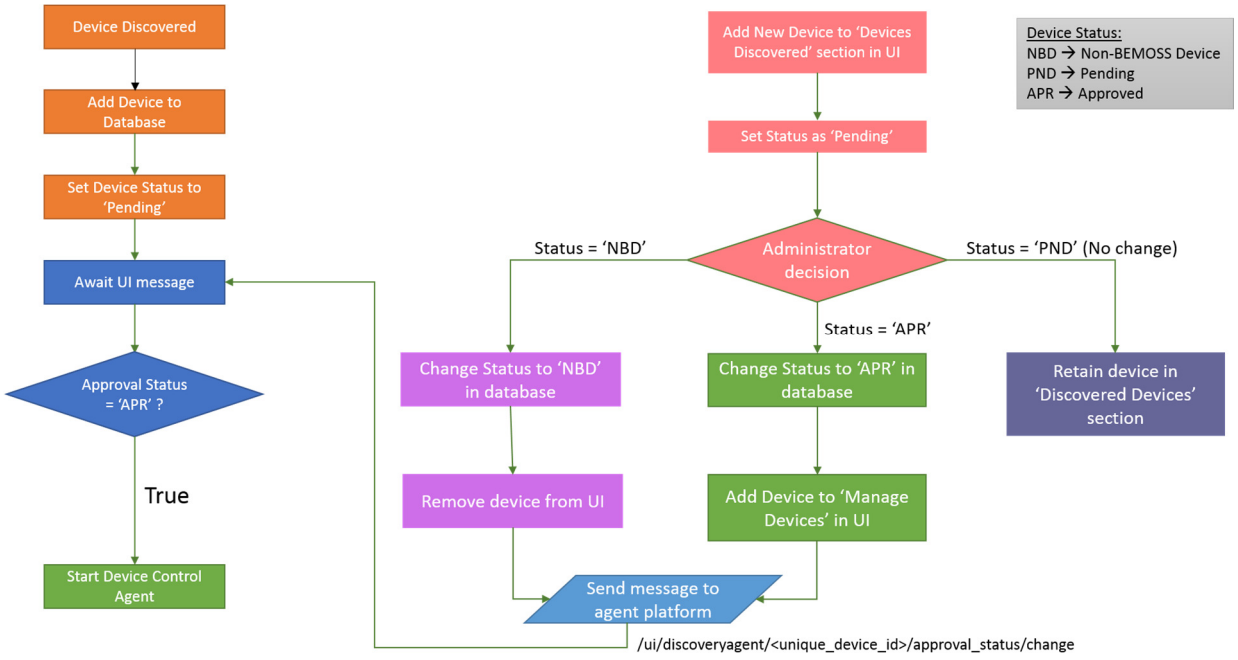


Figure 8-20 Device approval process

8.4.14 Securing the Message Queue

A BEM system often exchanges messages using web services or message queues. In the prototype implementation, a message queue is used. The message queue is implemented as publish - subscribe mechanism. There are two types of communication protocols used: inter process communication (ipc) and transmission control protocol (tcp).

Inter-Process Communication (IPC)

In ipc, there is no need of any IPv4 address to be used, but ipc endpoint names. This is a local name, and both processes need to refer to the same file name and permissions should be set appropriately. The ipc endpoint name does not require file creation. The queuing service created the resource to allow both processes to communicate with each other. This process is almost completely secure considering the machine is not compromised. These ipc endpoints cannot be published to the Internet.

Transmission Control Protocol (TCP)

TCP is a ubiquitous, reliable, unicast transport. When connecting distributed applications over a network with ZeroMQ, TCP is used for transport. Since messages are transported over a network, this protocol is vulnerable to a host of network-based attacks. The TCP sockets need to be encrypted to allow communications using this channel.

The security architecture for TCP connection is broken into several layers:

i) A ZeroMQ Message Transport Protocol adds a security handshake to all message queue connections.
ii) The security protocol, CurveZMQ implements a perfect forward secrecy between two peers over a TCP connection.
iii) A set of security mechanisms, like NULL, PLAIN, CURVE, is available, and the developer can choose to use any of the three.
    a. NULL: No protection
    b. PLAIN: Simple username and password-based authentication
    c. CURVE: Implements CurveZMQ security protocol
iv) An extended API allows configuration of a security mechanism to be used for this communication type (TCP).

Before discussing the security implementation, the Publish-Subscribe (PUB-SUB) mechanism is described below:

PUB-SUB is a pattern where publishers instead of programming messages to be directly sent to subscribers, but publish messages without the knowledge of any subscribers existing and listening to it. [93]

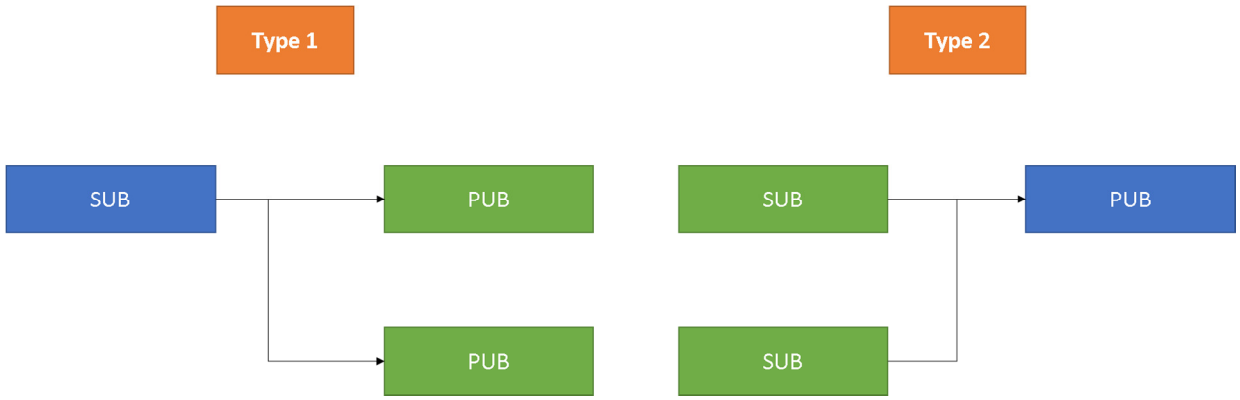


Figure 8-21 Types of PUB/SUB mechanism [93]
Source: http://learning-0mq-with-pyzmq.readthedocs.org/en/latest/pyzmq/patterns/pubsub.html
[Used under fair use, 2015]

In Type 1, ZMQ.SUB can connect to multiple ZMQ.PUB (publishers). No single publisher overwhelms the subscriber. The messages from both publishers are interleaved. Type 2 is a general pattern where multiple subscribers subscribe to messages/topics being published by a publisher.

Publishers are created with ZMQ.PUB socket type. Data is published on a particular topic. A subscribe can choose to subscribe to these topics. Subscribers are created using ZMQ.SUB socket type. A ZMQ subscriber connects with many publishers. Messages can be filtered based on topics. Pub/Sub communication is asynchronous. If a 'publish' service has been started and a 'subscriber' subscribes to a topic already published, those messages would not be received by the subscriber.

Messages published after a subscription is enabled will be received. If the subscriber connects to more than one publisher, the data arrive interleaved thus not allowing a single publisher to drown the queue with its messages. Filtering happens at the subscriber and not at the publisher end. In case of slow message transmissions, the messages are queued up at the subscriber.

Message Queue Security Mechanism:

The above discussed PUB-SUB mechanism and the security architecture is used to generate a secure queuing mechanism. Multiple publishers and multiple subscribers exist on the platform.

For every publisher:
Generate a public-private key pair.
Store it in a file in a safe location.
Manually distribute the public key to the subscribers.
For every subscriber:
Generate the key pair.
Send the public key to the publisher with which this subscriber needs to authenticate to subscribe to.

Create PUB socket. ("tcp://0.0.0.0:9161")
Configure this socket with the long-term key pair.
Create SUB socket. ("tcp://0.0.0.0:9162")
Configure each SUB socket with its own key pair and the server's public key.
Bind PUB socket.
Connect SUB sockets.

Send and receive messages over secure channel.

Multiple publishers can publish on the publish socket, and multiple subscribers can subscribe to the subscribe socket.

The cryptographic protocol used in this mechanism is Curve25519, the basis for the libsodium cryptographic library. Curve25519 provides encryption and authentication with 256-bit keys, equivalent to 3072-bit RSA keys [94].

Perfect Forward Secrecy

Although this is not enough to guard the message, perfect forward secrecy can be used to create temporary session keys. These temporary session keys can be exchanged using the long term keys. On session completion, these short term keys are discarded thus making it impossible for the attacker to obtain these keys. Clients and servers (publishers and subscribers) have long term key pairs, and each of them create a transient key pair for each connection.

The CurveZMQ protocol uses a synchronous handshake. After this, messages are exchanged asynchronously. A connection is started from client to server when the client sends the HELLO

message to the server. The server responds with a WELCOME message. In the next two messages, INITIATE and READY, each side is authenticated by the other. Metadata information is also exchanged. After this, the connection is authenticated and messages are exchanged securely and asynchronously.

At any point, either the client or server can close the connection. It can also happen because of network issues. These errors are soft errors and the disconnected entity may try to reconnect after x seconds. Sometimes a client may reject a connection explicitly, and at this time, the server sends an ERROR command (hard error).

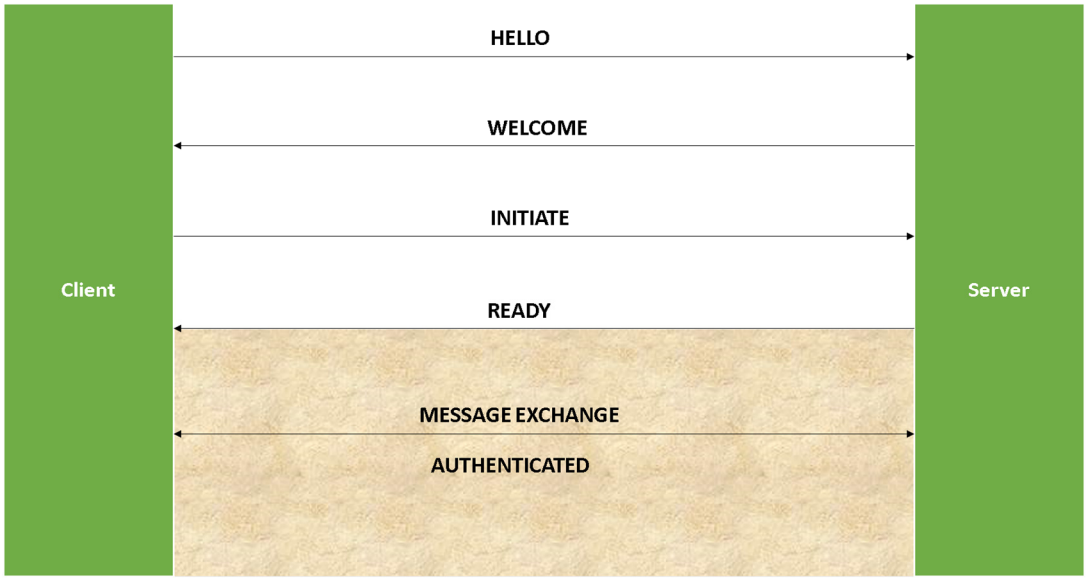


Figure 8-22 ZMQ authentication protocol using CurveZMQ

In the defined model, the client must know the server public key to attempt the above authentication. All clients share the same public key, that the server checks. In this case, access to the server is restricted to authorized clients. For the prototype implementation where private communication needs to happen over public infrastructure in multi-node communication, this model fits perfectly.

8.4.15 Isolating Sensitive Information

In an open source implementation where plenty of developers access a repository, check out and use the same code it is important that sensitive information be protected and kept unique. Securing sensitive information by placing them in areas that cannot be reached can prevent any security flaws in the system. However, in case of a system compromise most of this information is leaked and it is not secure anymore.

In the prototype implementation, in several areas, there is the usage of a SECRET_KEY, which is used for cryptographic signing within the application. It is important to safeguard this key at all times. The secret key is used for the following functions in the application, directly or indirectly:

- JSON object signing
- Password reset token
- Form security (to create a secure hash for the form instances)
- Protect against message tampering
- Protect session data and create random session keys
- Create random salt for most password hashers
- Create random passwords, as required
- Create CSRF tokens

The settings.py file contains all the configuration information required to run the web platform.

There is other sensitive information in the settings.py file, like the database information and URL root, etc. These configuration entities should be separated from the main settings.py file. The INSTALLED_APPS entity is referred to by the application management functions. This is also an entity frequently edited by developers as and when new apps are added to the root. Such frequently accessed entities can be separated from the otherwise constant entities in the settings file.

All sensitive settings are added to the configuration file and are retrieved using fundamental techniques for purposes of security. The frequently used entities are added to a different configuration file. The main settings file is configured to refer to these entities as and when required.

8.4.16 Linux Platform Security and System Monitoring

The prototype implementation is developed in the Linux Ubuntu environment. Relevant security modules are implemented and being tested on a constant basis for security purposes. Security of such a system is ongoing during the development stages. Penetration testing using some of the tools mentioned in Section 8.2.11 is performed periodically as the system goes through new iterations of development. For small embedded systems, since there is a constraint on the system resources that can be used for these purposes, the testing is completed on the bigger machines, and similar security rules can be incorporated in smaller machines for those possible.

System log utilities like systemd [95] are installed to obtain system logs. The core platform and the user platform constantly add logs to files to allow monitoring of application activities and for debugging purposes. Intrusion detection tool ‘Snort’ is installed and a set of rules are implemented to monitor the system intrusion.

9. Performance Evaluation

The prototype implementation, BEMOSS may interact with hundreds of devices in a building. At the same time, many users can access the application, reading or modifying device settings, and other parameters. As the number of requests to the application increases, the system experiences an increased load, which may affect the performance and user experience. Performance testing the application therefore becomes necessary. Performance and scalability are two key attributes to consider for evaluating the prototype. This chapter discusses the performance evaluation and the results for the web platform.

9.1 Performance Evaluation Requirements

SWEBOK [96] defines Software testing as follows: Software testing consists of the dynamic verification that a program provides expected behaviors on a finite set of test cases, suitably selected from the usually infinite execution domain.

Performance evaluation verifies that the software meets the specified performance requirements. It assesses the application for performance parameters like: capacity, response time. It is one of the forms of Software Testing. The test techniques used in this thesis are as follows:

Usage-based Techniques

This technique is used for operational testing. The environment reproduced for the purposes of testing is very close to the real-time operational environment of the application. The goal here is to ensure the reliability of the application in the real environment. Inputs are assigned in the form of profiles that depict real world scenarios.

Model-based Techniques

A model is an abstract representation of the software that is being testing, in this case, BEMOSS. This technique is used to validate the requirements, look for consistency, and develop test cases that focus on the operational behavior of the application. Specifically, workflow-based models are generated for testing. Workflow models are a sequence of activities performed by humans/software tools. Each workflow consists of a sequence of actions that represent activities corresponding to a real-world scenario. Different user roles also form part of the workflow testing, where each role is evaluated with the same workflow.

9.1.1 Response Time

Web page response time [97] is a common metric to quantifying user satisfaction. When an application is local to a building, web page response times might be faster, since it is cut off from the complex routing in the Internet. However, BEMOSS is a web application and response times might be delayed due to network traffic. The ‘Login’ page and the ‘Registration’ page have the

potential to have many page hits compared to other pages. Another key requirement is to ensure that response times for web pages are uniform.

Overhead of Dynamic Content and Web Page Complexity

There is an overhead caused by dynamic content loading for different users based on their roles. Web page complexity is increased as more number of objects is embedded into the web page increasing its total size. The number of objects in a web page determines the number of requests required from a server to the client. This can affect the response times. BEMOSS is still under development, which implies that script files, style sheets, etc. are split into several files for developer convenience. This directly affects response times due to the number of requests that need to be made every time the page is requested.

Lines of Code

The number of lines of code can also affect the response time of the web pages. This can be reduced by using HTTP compression techniques. This can reduce the effects of textual data by 70 to 75% [98]. HTTP compression or content encoding is a way to compress textual content transferred over the over from servers to clients. It uses compression algorithms, like gzip and compress, to compress HTML objects, JavaScript, Cascading Style Sheets, and other text files at the server. HTTP Compression along with caching and persistent connections can greatly improve web performance.

9.1.2 Workload Characterization

The testing scenarios mirror the way the system typically works in a real-world scenarios. This requires careful production workload characterization. The key here is to evaluate the think times, network strength and traffic, etc. to analyze the real world scenario, and characterize the same in the testing workload. Central to this activity is the identification of the variables /operations that characterize the operational features of the system. After modeling the application and the key parameters, the test scenario is constructed. Typically, the workload characterization involves collecting data over a period of time to understand user and system behaviors. In this case, the workload is characterized based on the system usage during the iterative testing phases by a variety of users.

BEMOSS installed in a building faces three different user roles at any point in time one of which is dynamic. The workload is described and recorded as scenarios that users are likely to execute. Think times are also registered. Workloads are characterized here by recording particular scenarios in the application that are commonly used. These scenarios are tested for response time and scalability.

Loads can be varied in four different forms:

i) Constant: Generates a fixed number of virtual users for the workload scenario.

ii) Ramp-up: Generates a number of virtual users that increases throughout the test. This can be used to test the server behavior in scenarios of increasing loads.

iii) Peak: Generates virtual users with periodically heavy loads. This load can be used to test the server condition in reverting to normal condition after the offset of peak loads.

iv) Custom: Here the individual tests can be assigned different types of loads by programming it.

9.1.3 Scalability and Platform

Scalability is limited in BEMOSS since number of users is restricted. Scalability is tightly linked to the platform on which the application is installed. For example, a Beaglebone is not capable of controlling more than 50 devices with a complete installation of BEMOSS. Scalability for BEMOSS is defined as the number of users and devices that can connect simultaneously to the system at any point of time.

9.1.4 Evaluation Metrics

Performance metrics for BEMOSS are based on application domain, requirements, and technology. Key metrics considered for performance testing are as follows:

- Response time – defines how fast the application performs
- Throughput – defines the number of megabits of data per second returned by the server
- CPU utilization – defines the percentage of CPU used by the server when a page request is processed.
- Memory utilization – defines the amount of memory utilized to serve a page request
- User load – defines the number of concurrent users BEMOSS can withstand
- Network Utilization – defines the network traffic characterized by using BEMOSS over the network / utilization of network bandwidth

The above-mentioned metrics are a combination of client-side (browser) and server-side (web platform) metrics.

9.1.5 Designing Test Scenarios

The workload modeling enumerates key scenarios of application usage by generating scripts and by recording the scenario using the load testing tools. These tools support transactions recording (workflow) and can run test scenarios in the live application. To ensure that tests reproduce real world scenarios, the following parameters need to be modeled in the recorded model:

- Think time – time taken by the end user between navigation of web pages or screens.
- Data Parametrization/Actions – the parameters involved in a particular page in the application. It can be entering form data or clicking button through the process to submit information.
- Speed/Browser simulation – the amount of time a browser takes to send the data to the server should be simulated since the test scenario does not usually run on browsers.

- Error handling – involves automating redirection or system exit in case of a system error.

9.1.6 Critical Scenarios

The critical scenarios are recorded as a single workload and a collective analysis needs to perform. Selection of the right performance scenarios is a critical task in a performance test. Performances test scenarios for BEMOSS are identified based on their importance, frequency, and performance impact.

9.2 Scope

The scope of testing in this thesis is limited to the web platform, focusing on the effectiveness of the user platform and the robustness of the web platform. For the purposes of testing the web platform, a core platform simulator is developed.

9.2.1 Core platform Simulator

Whenever there is a control input from the user, the user platform performs input validation, groups it into appropriate control elements, and relays the control input from the user to the core platform. The core platform processes this message and updates the device. Once updated, the core platform sends a confirmation message to the user. The web platform needs a simulator to mimic this agent functionality and to enable user platform developers to test the application without having devices beside them.

The agent simulator simulates the functionality of the agents to a much smaller scale to test the functionality of the user platform. The simulator listens to topics published by the user platform just like an agent would. When it receives the message corresponding to a topic it is listening to, it processes the message and publishes a message on the message bus for the user platform to receive it. The response message follows the data-exchange format rules and sends any data required in the JSON format. The agent simulator runs on the underlying Volttron platform and needs the platform to be running for it to function.

9.3 Tools

This section introduces the tools used in the performance evaluation of the web platform. Performance testing employs a server to act as injectors that emulate the presence of number of users and each running an automated sequence of interactions with the BEMOSS instance. The following tools are used in various testing processes:

- Neoload [99]
- Firefox/Chrome Developer Tools
- Firebug [100]
- YSlow [101]

9.3.1 Neoload

Neoload is a tool used for measuring and analyzing the performance of a web application. This tool helps with performance optimization of the web platform. In addition to simulating response times, it also simulates end user interaction and think times.

9.3.2 Firefox/Chrome Developer Tools

Firefox and Chrome Browsers contain tools for inspecting pages, executing arbitrary JavaScript code and viewing HTTP requests and other messages live. These tools were used throughout the development phase of the web platform to analyze HTML, debug JavaScript, and modify page styles. The Responsive Design View is verified for every page using these tools instead of having to port the application into devices of different sizes.

9.3.3 Firebug

Firebug is a free open-source browser extension for Firefox and Chrome. It allows live debugging and script monitoring including HTML, CSS, JavaScript, XHR, etc. for the web pages. It is also used to perform web page security testing and web page performance analysis. Firebug's network monitoring facilities allow the developer/tester to track response times per HTTP request (i.e. per resource), monitor browser cache usage, inspect HTTP request and response headers, and visualize the entire loading of a web page (from the root HTML document to any referred resources) on a timeline. Firebug's JS profiling features allow diagnosing script induced performance issues. Today, similar features are also available in browsers like Chrome and Firefox, which have also been leveraged for purposes of the prototype development and testing.

9.3.4 YSlow

YSlow is a browser plugin available in Firefox and Chrome browsers to analyze web page performance. It grades web pages based on a predefined rule set [101] as discussed by Yahoo. It also provides a summary of the web page components, page statistics and provides additional tools for performance analysis.

9.4 Performance Evaluation Process

This section discusses the overall application performance evaluation. Figure 9-1provides a broad overview of the process each of which is discussed in this section.

BEMOSS is designed to run on single board embedded systems like Raspberry Pi2, Odroid, Pandaboard and also on powerful computers and desktop servers. Such flexibility requires a highly efficient application. BEMOSS is run on Odroid embedded system and on VirtualBox in a regular computer for purposes of this testing. The performance of these two instances of BEMOSS is tested and the results are discussed in this section.

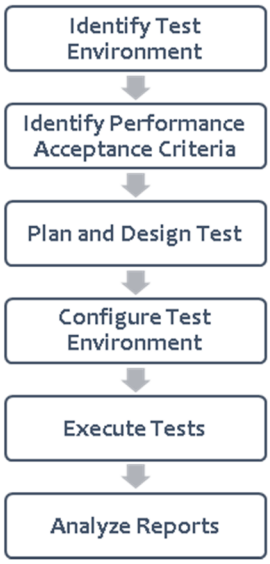


Figure 9-1 Performance evaluation process
Source: https://msdn.microsoft.com/en-us/library/bb924356.aspx
[Used under fair use, 2015]

9.4.1 Test Environment

For testing BEMOSS, the testing environment can be set at the lower end of resource availability to simulate the real world scenario. The real world scenario may compromise of a single board computer with a local area network within a building that may restrict access to outside the building premises. A single board computer is simulated in the virtual test environment. Also simulated is the lowest possible setting in a fully functional computer in the same testing environment. The following is the system configuration for the test environment in the computer.

- VirtualBox version 4.3 on Windows 8.1
- Base Memory: 2 GB
- Processor: 2 CPU
- Network: Bridged Adapter (Intel PRO/1000 MT Desktop)
- 30 GB disk space
- Linux Ubuntu 14.04 LTS

For testing on embedded systems, an Odroid U3 board is used. The following is the system configuration of the board:

- Processor : 4 CPU (1.7 GHz Quad-Core)
- Base Memory: 2 GB
- 32 GB disk space
- Lubuntu 14.04 OS

9.4.2 Performance Acceptance Criteria

Performance tests are executed against quantitative goals and test results are verified against these goals. For BEMOSS, certain realistic goals are set up considering the web platform alone.

- Response Time: The maximum page load time for the heaviest page is set at less than 7 seconds. This is normally the zone dashboard with about 50 to 70 device widgets.
- Throughput: The throughput is the expected number of megabits transferred per second by the server. This is set at a minimum of 3 Mb per second.
- Resource Utilization: CPU usage is usually about 80% for BEMOSS while the memory usage should be about 5%.
- Maximum User Load: As opposed to user load, the number of page hits is a better measure for BEMOSS considering that the user population is smaller but the page hits might be about 5 pages per minute per user.

9.4.3 Plan and Design Test

Only critical scenarios are considered for testing. They are combined into one recording and executed for a period of 30 minutes continuously with varying user load. Table 9-1 lists the test scenarios.

Table 9-1 Test scenarios

Scenario Definition	Metrics
Login	Response time should be less than 2 seconds for about 300 page hits
View Zone Dashboard	Response time should be less than 4 seconds for a page that loads about 30 device widgets for about 300 page hits
View Device Dashboard	Response time should be less than 3 seconds for a device control dashboard page for about 300 page hits
Control Device	Response time is defined as the time between submitting a control request and receiving success/failure notification on the screen for any device that is controlled. The response time for such a scenario is about 1 to 2 seconds.
Schedule Page	Response time should be less than 3 seconds for a device schedule page for a 200 page hits approximately.

Here page hits are defined as the number of concurrent hits on the web page.

9.4.4 Test Environment Configuration

After identifying the requirements, the environment is set up and implemented. In this phase, the test environment is set up as discussed in Section 9.4.1.

9.4.5 Implementing the Test Design

Once a test environment is established, performance scenarios were scripted and recorded using Neoload. A population of five virtual users is considered all with the administrator role. This role

is considered because of the highest level of authorization available to this role. User actions are simulated and the real time parameters are added to the test system.

A think time of about 5 milliseconds is provided. The username, password, and other information required to navigate through the page are recorded in the scenario.

9.4.6 Test Execution

The test scripts recorded are run against different user sets with different resource configuration and test results are monitored live on Neoload to observe the trend. Three types of user loads are simulated in the execution.

- Constant: Generates 5 virtual users for the workload scenario.
- Ramp-up: Generates 10 virtual users in total starting with 2 users and incrementing by 2 users for every 2 minutes.
- Peak: Generates 10 virtual users with periodic heavy loads. This adds ten virtual users to the system every 2 minutes.

9.4.7 Test Results

The test results are collected in the form of graphs and statistics for analysis. The evaluation results are discussed in Section 9.5.

9.5 Performance Evaluation Results

The user platform server is monitored and a core platform running 40 simulated devices (10 HVAC controllers, 10 lighting load controllers, 10 plug load controllers, and 10 sensors). The statistical performance summary for the two configurations is discussed in Section 9.4.1.

9.5.1 Performance Testing on Odroid Embedded System

An Odroid board has limited resources and therefore the maximum number of devices that can be run on BEMOSS is tested. Table 9-2 shows the statistical performance summary on an Odroid board.

Table 9-2 Statistical performance summary for Odroid

Total pages	579	Average pages/s	4.8
Total hits	1250	Average hits/s	10.3
Total users launched	40	Average Request response time	1.2s
Total iterations completed	61	Average Page response time	2.25s
Total throughput	0.54MB	Average throughput	0.04MB
Total hit errors	0	Error rate	0 %
Total action errors	0	Total duration alerts	0 %

A total of 40 users were launched with BEMOSS running 25 devices. In an Odroid board, with the time-series data collection, 25 devices could be successfully simulated.

Figure 9-2 highlights the highest response time to be ~6s for 40 virtual users with no errors and no critical alerts.

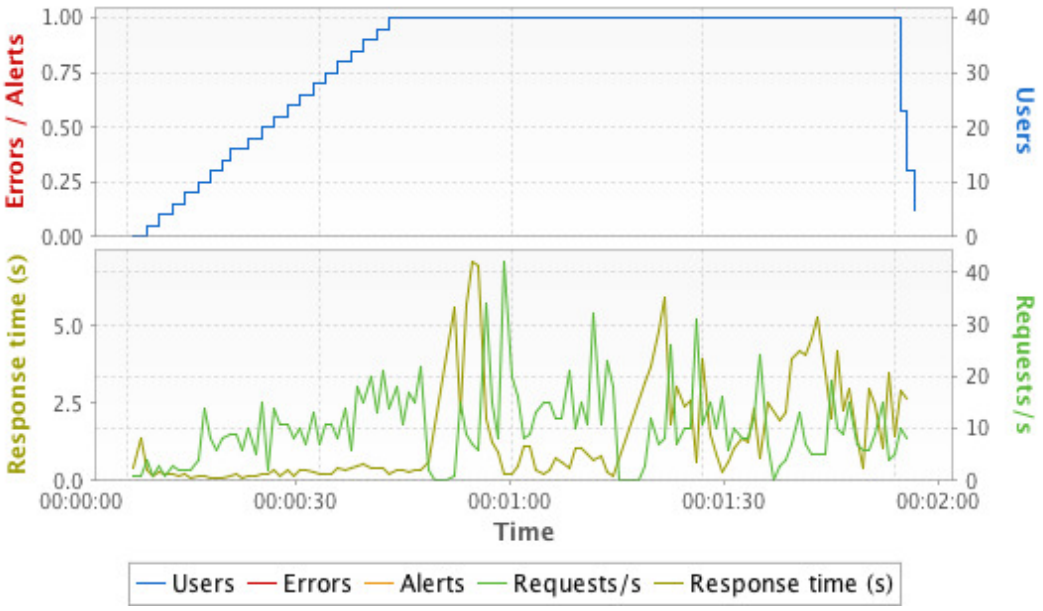


Figure 9-2 Overall performance summary of Odroid

Response Time

Response time is the total time taken from the time a page /control is requested to the time when a page is displayed or a notification is received in case of control input. Figure 9-3 shows the request response time for the first two minutes of the test run.

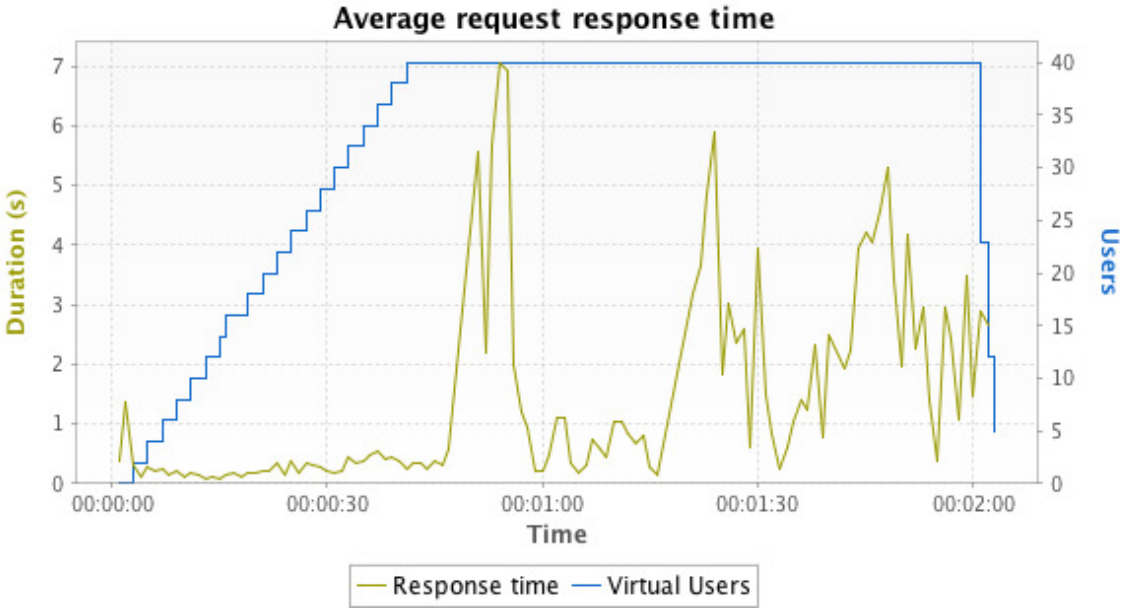


Figure 9-3 Request response times for Odroid

One set of population involving 40 virtual users are tested in this scenario and an average page response time of 2.247 seconds is observed with 0% error.

The average page response time in seconds of all pages during the test is shown in Figure 9-4. The blue line indicates the number of virtual users and the page response time is shown in green. The page response varies between 0.124 seconds and 11.234 seconds approximately for most cases. This includes pages with high number of objects as well as simple pages like the home page. The control input to a thermostat is also considered for this scenario.

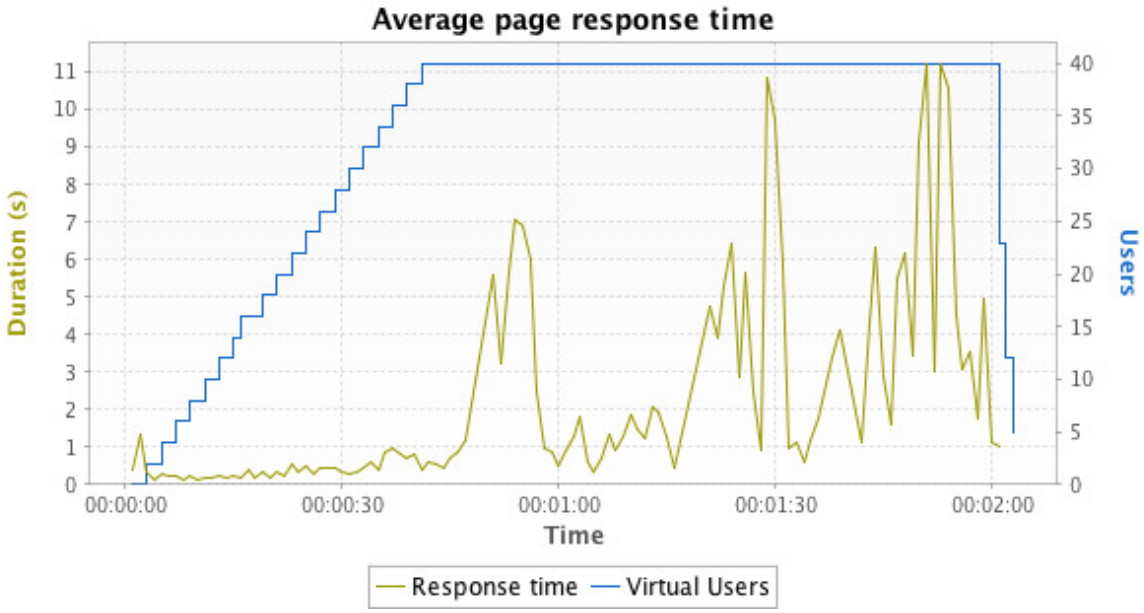


Figure 9-4 Page response times for Odroid

The statistics for the above graph is also shown in Table 9-3.

Table 9-3 Page response time statistics for Odroid

Minimum (seconds)	Average (seconds)	Maximum (seconds)	Median (seconds)	Standard Deviation
0.124	2.247	11.234	0.946	2.683

The average request response time is the key indicator of the server request processing speed. Request Response time is the time taken by the server to process a single request with the current execution environment. This response time will increase with more users and more load. From the test results, the average request response time is about 2.247 seconds with a maximum of 7.061 seconds and a minimum of 0.085 seconds. Average request response times are shown in Figure 9-5.

Throughput

Throughput is the number of megabits per second that is returned by the server to the client. It reflects the capacity of the server to respond to the request from the user. There is an initial peak when new pages are requested and there is no browser caching. All pages are continuously re-requested by the client. After a page is loaded for the first time, the cached version of the page reduces the load on the server thus reducing the number of megabits transferred from ~0.55 Megabits/second to ~0 megabits/second. Throughput results for the first two minutes of the run are shown in Figure 9-5.

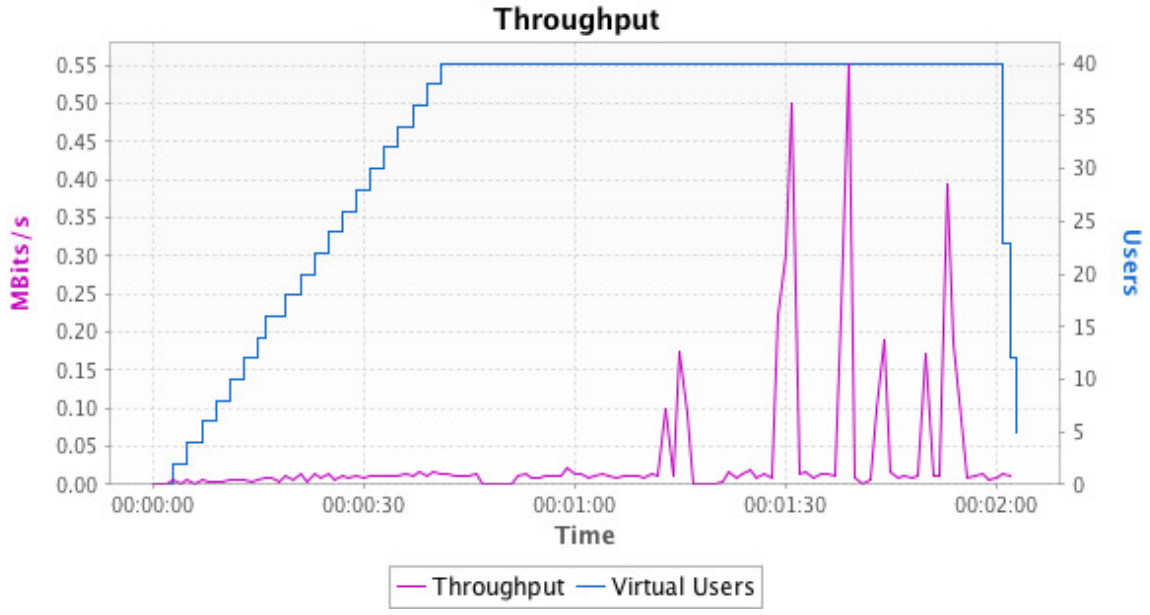


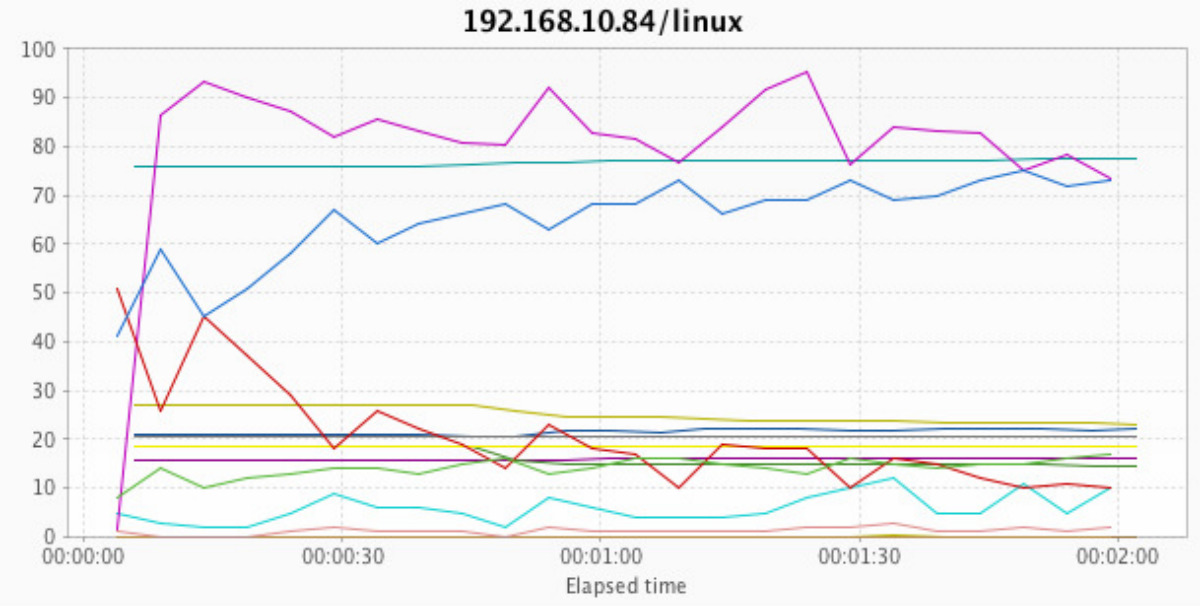
Figure 9-5 Throughput results for Odroid

Resource Utilization

CPU, memory, user load, and network usage results are discussed in this section. An average of 70% CPU utilization is observed with peaks of about 83% utilization when the number of users is at a peak. Memory utilization of about 15% average is observed with a maximum of 18%, indicating system resources necessary for the platform to operate without errors in the highest load setting. Resource utilization results are summarized in Figure 9-6. CPU utilization is the percentage of time the processor is not idle. Memory utilization is a direct indicator of the amount of total memory used for the application activity.

Network/segments sent/sec: This is the rate at which the TCP segments are sent through the network. It defines the amount of data (information) sent out for TCP/IP transmissions.

Network/segments retransmitted/sec: This is the rate at which re-transmissions of the segments occur. Retransmissions are measured based on bytes in the data that are recognized as being transmitted before. Excessive retransmissions indicate a distinct reduction in application bandwidth. These details are furnished in the graph shown in Figure 9-6.



Test 12:48 - 13 Jul 2015 **Statistic** -

Color	Scale	Name	Unit
	1	192.168.10.84/linux/System/CPU User	%
	1	192.168.10.84/linux/System/CPU System	%
	1	192.168.10.84/linux/System/CPU Idle	%
	0.01	192.168.10.84/linux/System/Context switch count per CPU	
	1	192.168.10.84/linux/System/Process Runnable	
	1	192.168.10.84/linux/System/Process Runnable per CPU	
	1	192.168.10.84/linux/System/Swap in	
	1E-5	192.168.10.84/linux/Memory/Memory Total	
	1E-5	192.168.10.84/linux/Memory/Memory Used	
	0.0001	192.168.10.84/linux/Memory/Memory Free	
	0.001	192.168.10.84/linux/Memory/Memory Buffered	
	0.0001	192.168.10.84/linux/Memory/Memory Cache	
	1E-5	192.168.10.84/linux/Memory/User Memory	
	1	192.168.10.84/linux/Memory/% User Memory	%
	1	192.168.10.84/linux/Memory/Swap Used	
	1	192.168.10.84/linux/TCP/% Segments retransmitted	%
	1	192.168.10.84/linux/Network (per interface)/% Incoming packet errors(wl...	%
	1	192.168.10.84/linux/Network (per interface)/% Outgoing packet errors(wl...	%

Figure 9-6 Resource utilization results for Odroid

9.5.2 Performance Testing on BEMOSS running on a VirtualBox in Windows PC

VirtualBox is installed on a Windows 8.1 PC as described in Section 9.4.1. The VirtualBox is limited to 2GB memory and 30GB disk space with 2 CPU cores allocated for processing. The following is the statistical performance summary on BEMOSS installed in a Virtual environment in a PC. The virtual population is ramped up from 2 users adding 2 users every 12.0 seconds, to a maximum of 40 users.

Table 9-4 Statistical performance summary

Total pages	719	Average pages/s	3.0
Total hits	2034	Average hits/s	8.4
Total users launched	40	Average Request response time	0.495s
Total iterations completed	83	Average Page response time	1.26s
Total throughput	49.47MB	Average throughput	1.64MB/s
Total hit errors	0	Error rate	0 %
Total action errors	0	Total duration alerts	83.1%

A total of 40 users were launched with BEMOSS running 40 devices.

Figure 9-7 highlights the highest response time to be ~4s for 40 virtual users with no errors and no critical alerts.

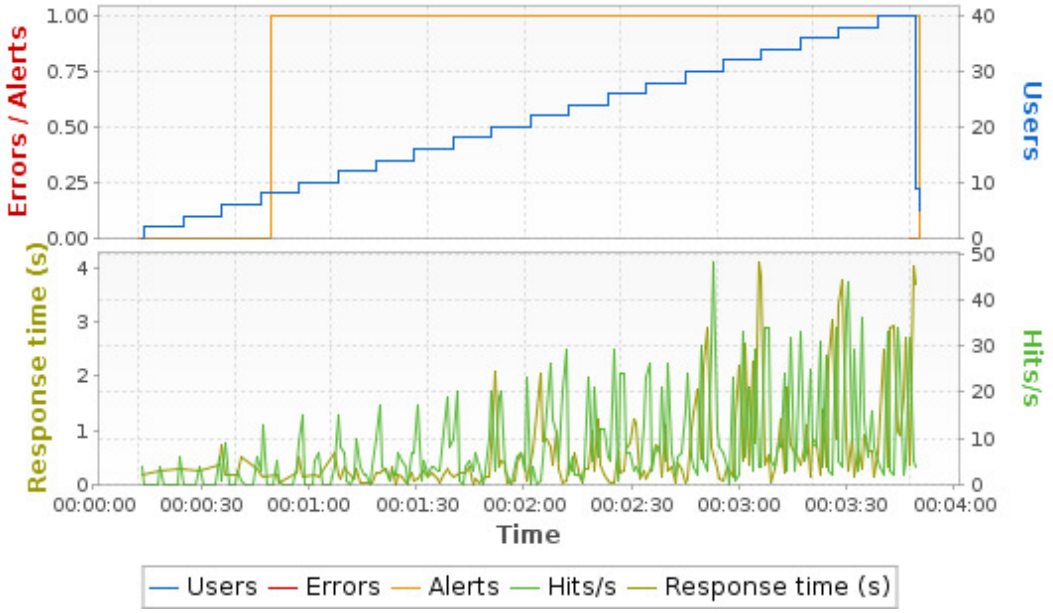


Figure 9-7 Overall performance summary for PC

Response Time

Response time is the total time taken from the time a page /control is requested to the time when a page is displayed or a notification is received in case of control input. Figure 9-8 shows the response time for the first four minutes of the test run.

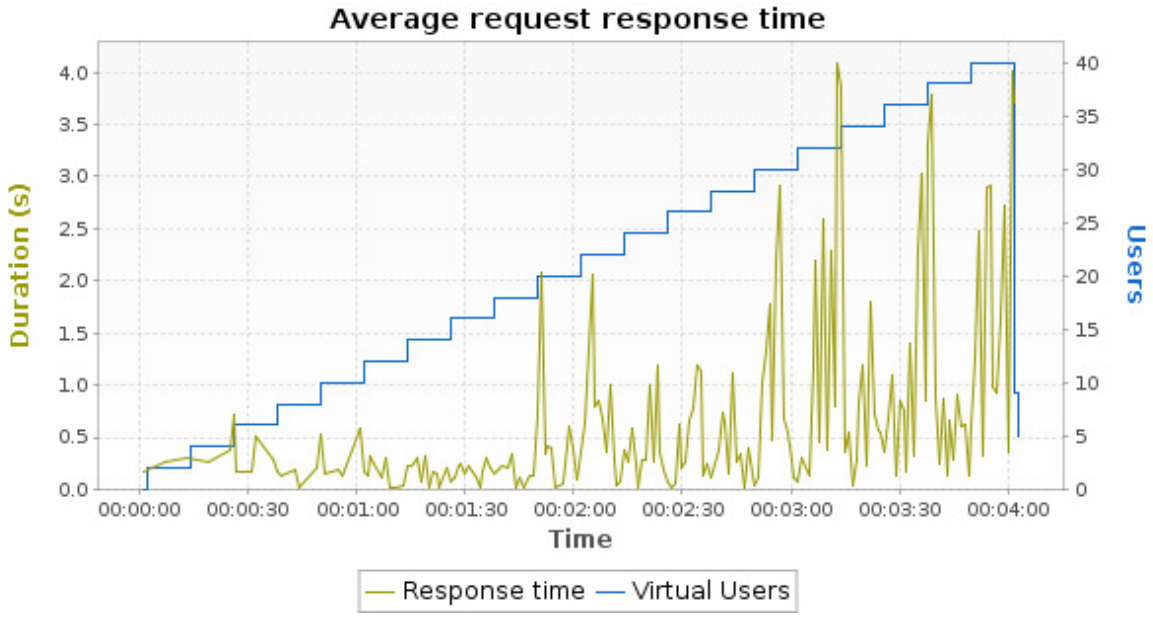


Figure 9-8 Request response times for PC

One set of population involving 40 virtual users are tested in this scenario and an average page response time of 1.255 seconds is observed with 0% error.

The average page response time in seconds of all pages during the test is shown in Figure 9-9. The blue line indicates the number of virtual users and the page response time is shown in green. The page response varies between 0.015 seconds and 6.813 seconds approximately for most cases. This includes pages with high number of objects as well as simple pages like the home page. The control input to a thermostat is also considered for this scenario.

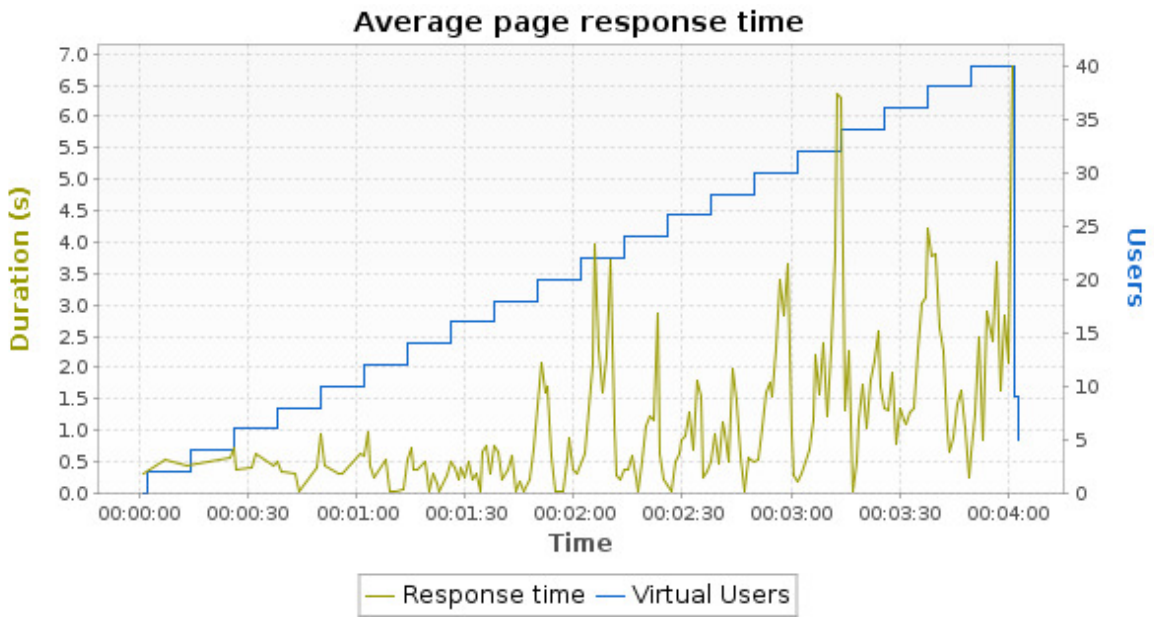


Figure 9-9 Page response times for PC

The statistics for the above graph is also discussed in the Table 9-5.

Table 9-5 Page response time statistics for PC

Minimum (seconds)	Average (seconds)	Maximum (seconds)	Median (seconds)	Standard Deviation
0.015	1.255	6.813	0.942	1.149

The average request response time is the key indicator of the server request processing speed. Request Response time is the time taken by the server to process a single request with the current execution environment. This response time will increase with more users and more load. From the test results, the average request response time is about 0.486 seconds with a maximum of 4.096 seconds and a minimum of 0.01 seconds.

Throughput

Throughput is the number of megabits per second that is returned by the server to the client. It reflects the capacity of the server to respond to the request from the user. All pages are continuously re-requested by the client. After a page is loaded for the first time, the cached version of the page reduces the load on the server thus reducing the number of megabits transferred. Throughput results for the first four minutes of the run are shown in Figure 9-10.

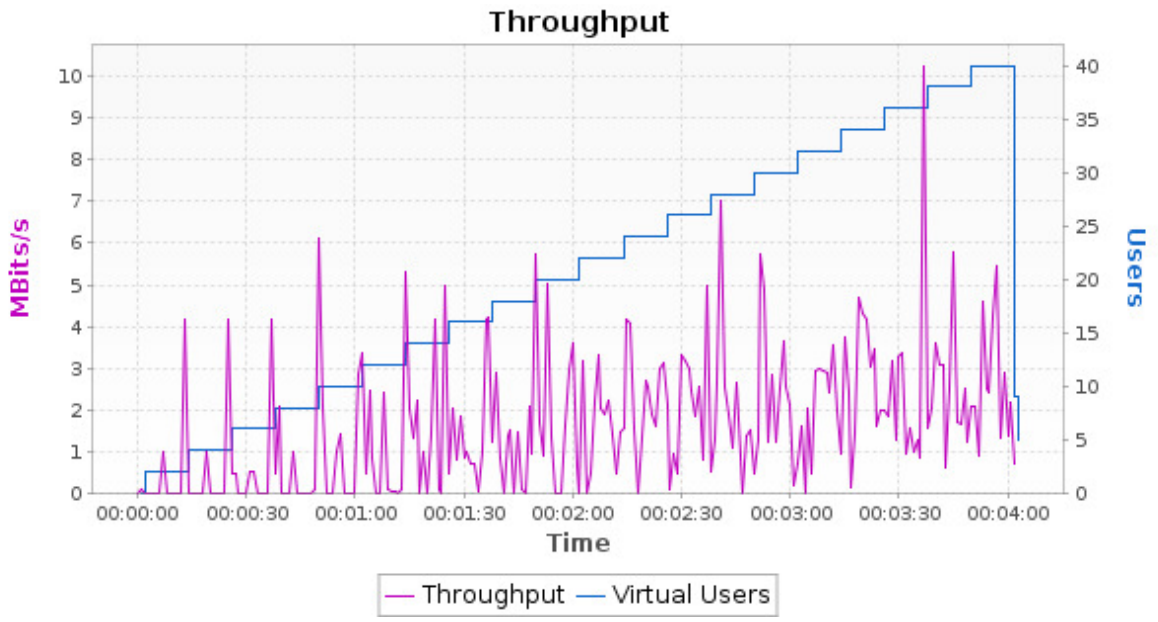


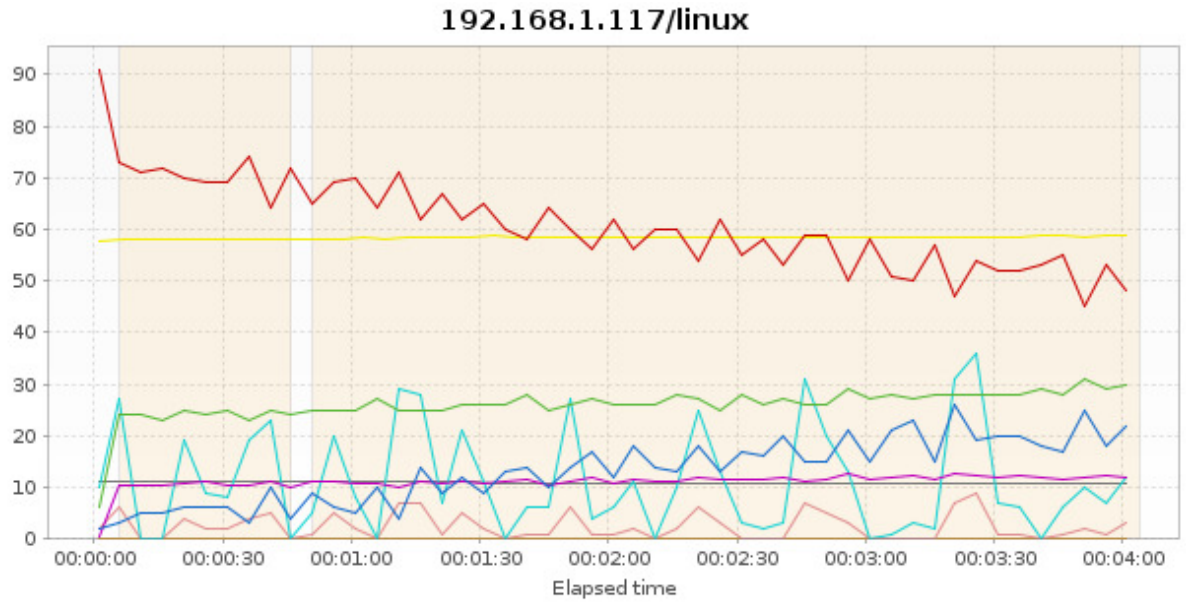
Figure 9-10 Throughput results for PC

Resource Utilization

CPU, memory, user load, and network usage results are discussed in this section. An average of 30% CPU utilization is observed with peaks of about 35% utilization when the number of users is at a peak. Memory utilization of about 58% average is observed. This is indicative of the amount of system resources necessary for the platform to operate without errors in the highest load setting. Resource utilization results are summarized in Figure 9-11. CPU utilization is the percentage of time the processor is not idle. Memory utilization is a direct indicator of the amount of total memory used for the application activity.

Network/segments sent/sec: This is the rate at which the TCP segments are sent through the network. It defines the amount of data (information) sent out for TCP/IP transmissions.

Network/segments retransmitted/sec: This is the rate at which re-transmissions of the segments occur. Retransmissions are measured based on bytes in the data that are recognized as being transmitted before. Excessive retransmissions indicate a distinct reduction in application bandwidth. These details are furnished in the graph shown in Figure 9-11.



Test 22:46 - 13 Jul 2015 **Statistic** -

Color	Scale	Name	Unit
	1	192.168.1.117/linux/System/CPU User	%
	1	192.168.1.117/linux/System/CPU System	%
	1	192.168.1.117/linux/System/CPU Idle	%
	0.001	192.168.1.117/linux/System/Context switch count per CPU	
	1	192.168.1.117/linux/System/Process Runnable	
	1	192.168.1.117/linux/System/Process Runnable per CPU	
	1	192.168.1.117/linux/System/Swap in	
	1E-5	192.168.1.117/linux/Memory/Memory Free	
	1	192.168.1.117/linux/Memory/% User Memory	%
	1	192.168.1.117/linux/Memory/Swap Used	
	1	192.168.1.117/linux/TCP/% Segments retransmitted	%
	1	192.168.1.117/linux/Network (per interface)/% Incoming packet erro...	%
	1	192.168.1.117/linux/Network (per interface)/% Outgoing packet err...	%

Figure 9-11 Resource utilization results for PC

9.5.3 Summary

Test results show a boost in performance with frequent use since the cache for the application renders the page faster by retaining the static components. About 25 devices can be configured on a single Odroid board with 40 users accessing it simultaneously. However, performance is maximized when the number of users is less than 25. For a similar test performed in a VirtualBox implementation of BEMOSS, about 40 devices could be accommodated and 40 virtual users can access the application simultaneously. The performance on the VirtualBox is about 4 times faster than the Odroid instance of BEMOSS. Both of these instances could run the entire platform with zero or minimum errors.

9.6 Client Side Performance Evaluation

This section discusses the client side/end-user web page analysis. Different sets of pages are analyzed with different simulated devices and results are discussed. Tools used for client side entities include:

- Firefox/Chrome Developer tools
- Yslow
- Firebug

9.6.1 Web Browser Compatibility

Web-based platforms involve the use of a variety of scripting tools and some complex CSS functionalities combined to load a web page. While simple CSS and JS will work on any web browser, it is important to test the compatibility of the application on different browsers including the Chrome and Firefox.
With the advent of smart phones, tablets and devices like the Blackberry Passport, which comes with an unusual square screen, it is important to test the compatibility and the effectiveness of the web platform on these screens to ensure that the functionality is retained.

The user interface is designed as a responsive interface, which allows it to adjust responsively to different screen sizes. While this feature takes care of a majority of compatibility issues, a few of the CSS functionality that involved Webkit [102] rendering needs to be verified. This rendering is used in the zone dashboards. For example, a single yellow light bulb *png* image is used and it is modified to a grey shade to depict the light turning off using webkit.

Two frequently used browsers, Firefox and Chrome, are considered for this testing. The webkit functionality does not immediately apply to Firefox browser. For this reason, the moz-transform [103] is used.

Figures 9.12- 9.14 show screenshots from Chrome and Firefox browsers displaying light bulbs. The webkit used in Firefox does not display the light status correctly (Figure 9-12). The light bulbs display correctly when using moz-transform in Firefox (Figure 9-13) and using in Chrome browser (Figure 9-14).

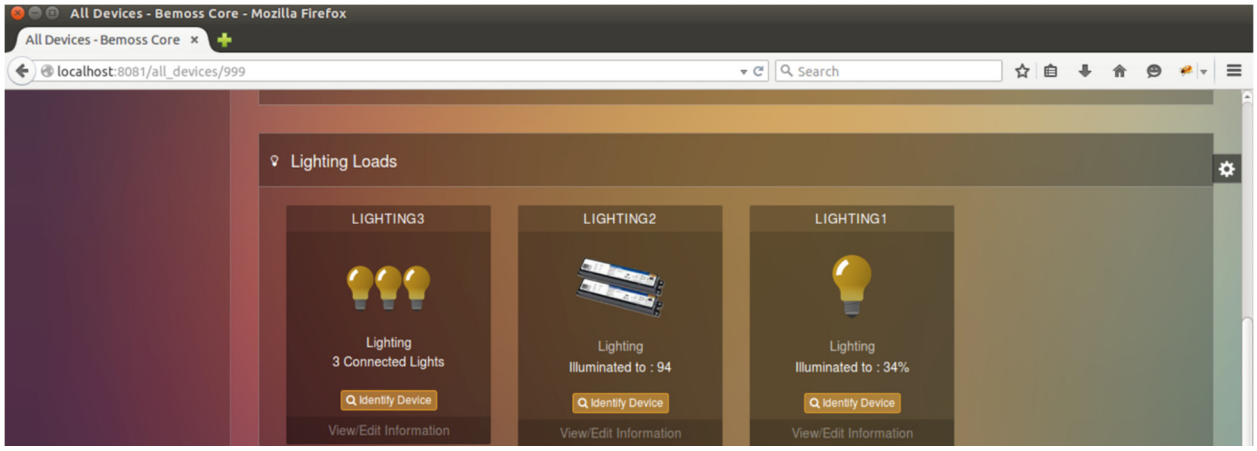


Figure 9-12 Firefox webkit (does not show light color changes)
Source: http://www.bemoss.org. Used with permission, 2015.

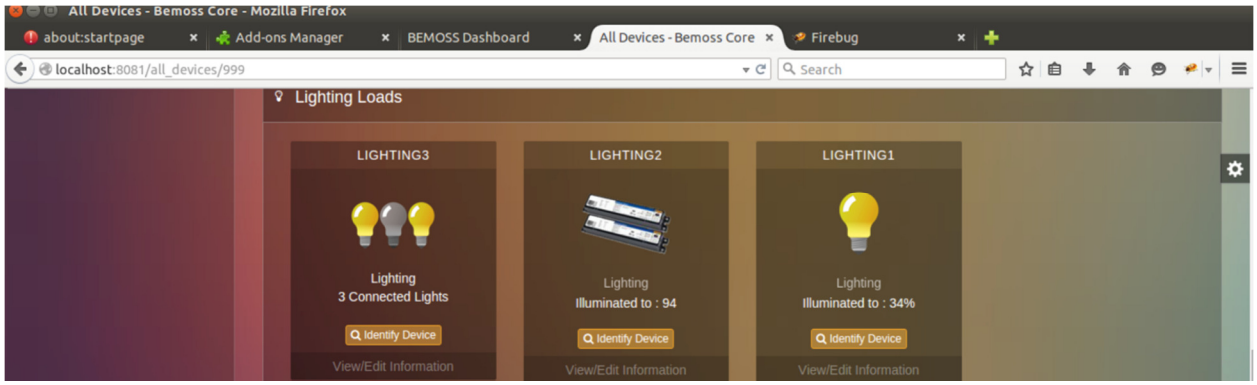


Figure 9-13 Moz transform in Firefox (light colors vary for the first Lighting widget)
Source: http://www.bemoss.org. Used with permission, 2015.

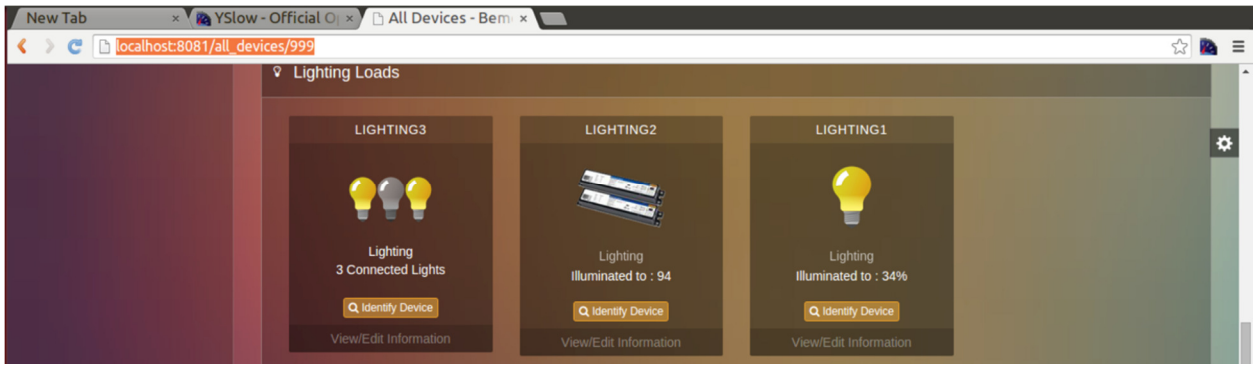


Figure 9-14 Webkit in chrome
Source: http://www.bemoss.org. Used with permission, 2015.

The responsive design is verified using the Responsive Design View module available in Firefox developer tools. The responsive design interface automatically adjusts to various screen sizes as shown in
Figure 9-15.

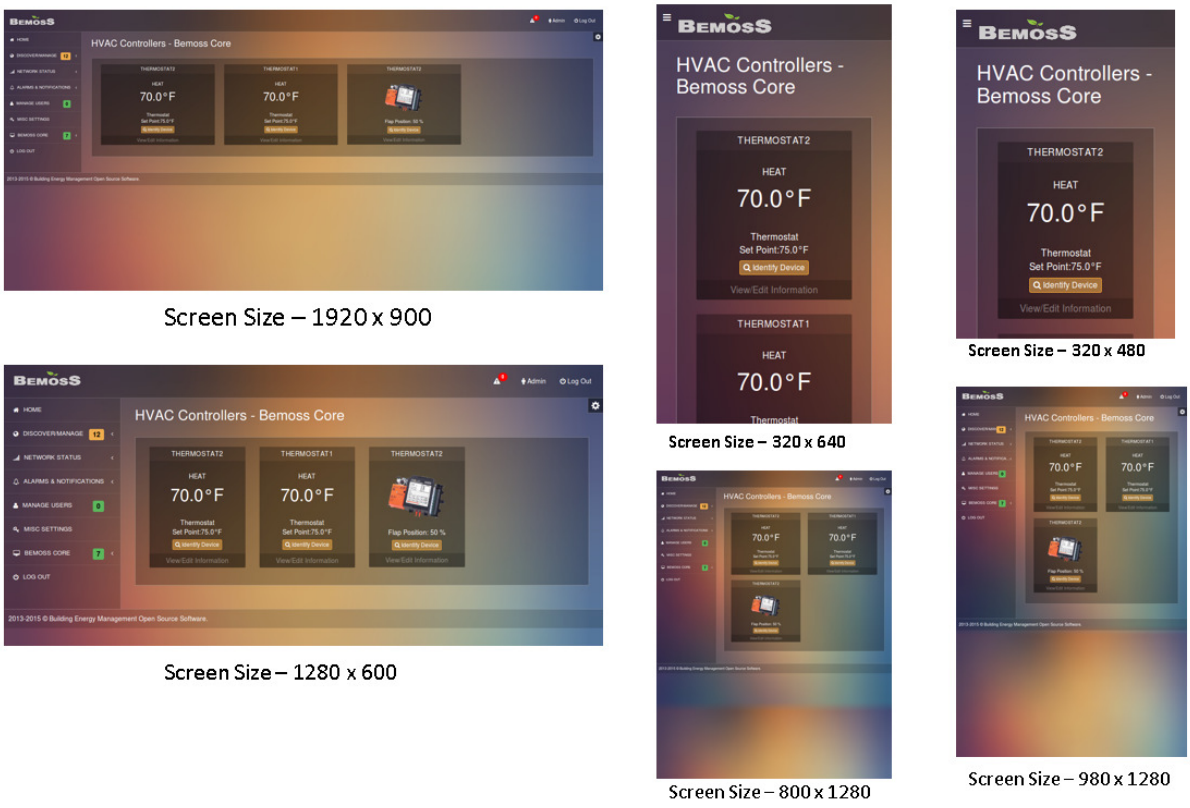


Figure 9-15 Responsive design verification
Source: http://www.bemoss.org. Used with permission, 2015.

9.6.2 Web Page Analysis

Login Page

The login screen is the most important screen in the entire application given it is the only page that is potentially viewable by everyone on the Internet. Other pages are accessible only after login and are only a fraction compared to the other millions of users who have access to the login screen.

Page load performance of the login page is evaluated for a single user view using Yslow. The Firebug statistics for the BEMOSS login page are discussed below and shown in Figure 9-16 and Figure 9-17.

About 10 GET requests are made with a total response time of 613 milliseconds (ms), and a combined page size of about 333.1 KB. Requests are displayed in the order in which they are sent to the browser. The status information shows that the page load is successful. The size of each entity that is loaded is also shown. This page is loaded partially from cache (327.4 KB) which renders the web page faster.

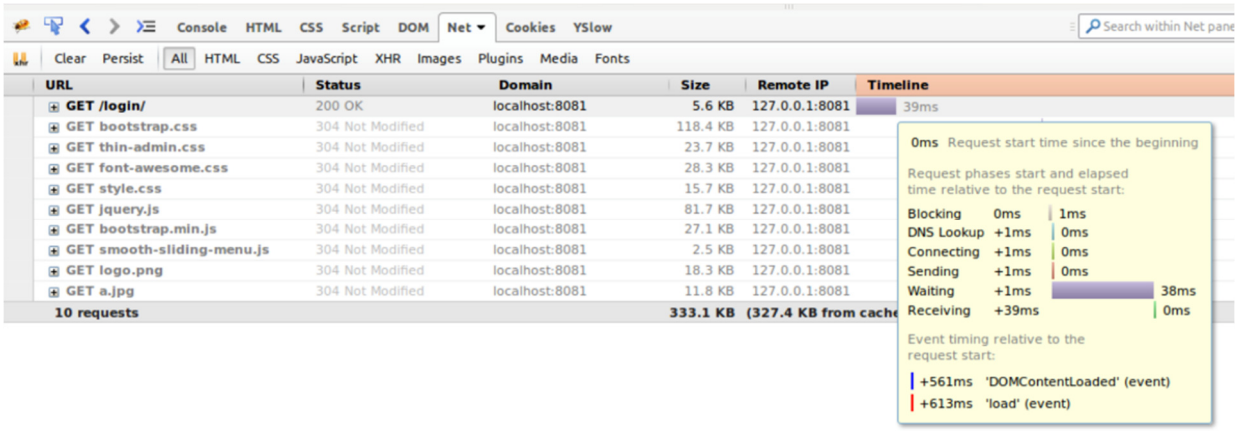


Figure 9-16 Firebug network statistics for login screen – primed browser cache

The cache is then cleared and the page is reloaded. With an empty browser cache, the page load took about 778 ms, with a total size of 376.5 KB. By caching resources, the total response time is reduced by about 20% when caching is used. This number will increase for a highly complex web page like the zone dashboard. The browser sends conditional GET request to retrieve the resource that has been previously cached, and if the resource is not modified after the last page load, the status is set to 304 (not modified) and the cached resource is used.

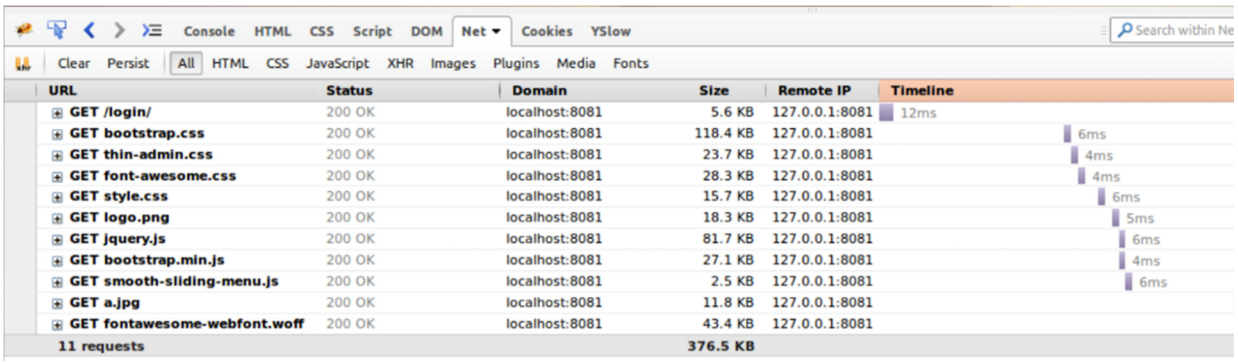


Figure 9-17 Page load - empty browser cache

YSlow Statistics for the Login Page

Results are shown in Table 9-6. For the login page, the following statistics were observed by Yslow in summary for primed cache and an empty cache. 12 HTTP requests were made for the page load with a total weight of 370.3 kB.

Table 9-6 YSlow statistics for login page

Entity	Empty Cache (bytes)	Primed Cache (bytes)
HTML/ text	5.7k	5.7k
JavaScript file	113.9k	0.0k
Stylesheet file	190.6k	0.0k
CSS image	12.1k	0.0k
Image	18.7k	0.0k
Favicon	14.5k	0.0k
Undefined	38.2k	0.0k

It can be noticed that with caching the page load weight is reduced from 370.3k to 5.7k. Most of the elements loaded on to the page are cacheable. This observation is shown in Figure 9-18.

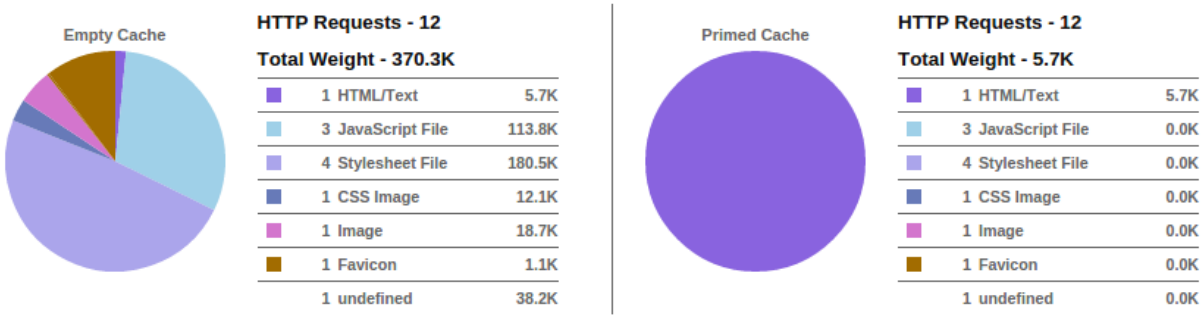


Figure 9-18 Yslow statistics for login page

Heap Snapshot for ‘Manage Devices’ page

The heap snapshot of the ‘Manage Devices’ page is shown in Figure 9-19. This page is heavy considering the amount of functionality and information rendered to the user. A total of ~8.5MB is loaded to the page in about 4.33 seconds.

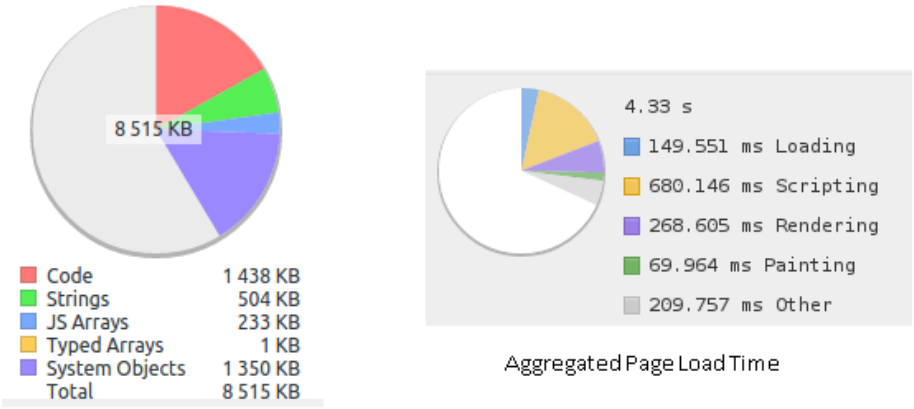


Figure 9-19 Heap snapshot - manage devices page

The page loaded three different load types – thermostats, plug loads, and lighting loads. The manage devices page has multiple functionalities embedded in different tabs and it is one of the most complex pages rendered in the entire application. This page took about 4 s to load 30 devices that needed to be managed. Some similar pages are considered for benchmarking.

For instance, the Instagram home page, which has a bunch of images loading, and has a framework that is similar to that of BEMOSS took about 4.9 s for the home page to load. This is comparable to the performance of BEMOSS. Most popular web pages take about 5 to 7 s to load on an average. BEMOSS takes only about 3 s in comparison to load the most complex page in the system.

YSlow Recommendations

Using YSlow, the most complex page ‘Manage Devices’ is tested. On a scale of A to F, the overall result on web application performance is C, most of the suggestions provided by YSlow relate to the development nature of the application. Some of the results are shown in Figure 9-20.

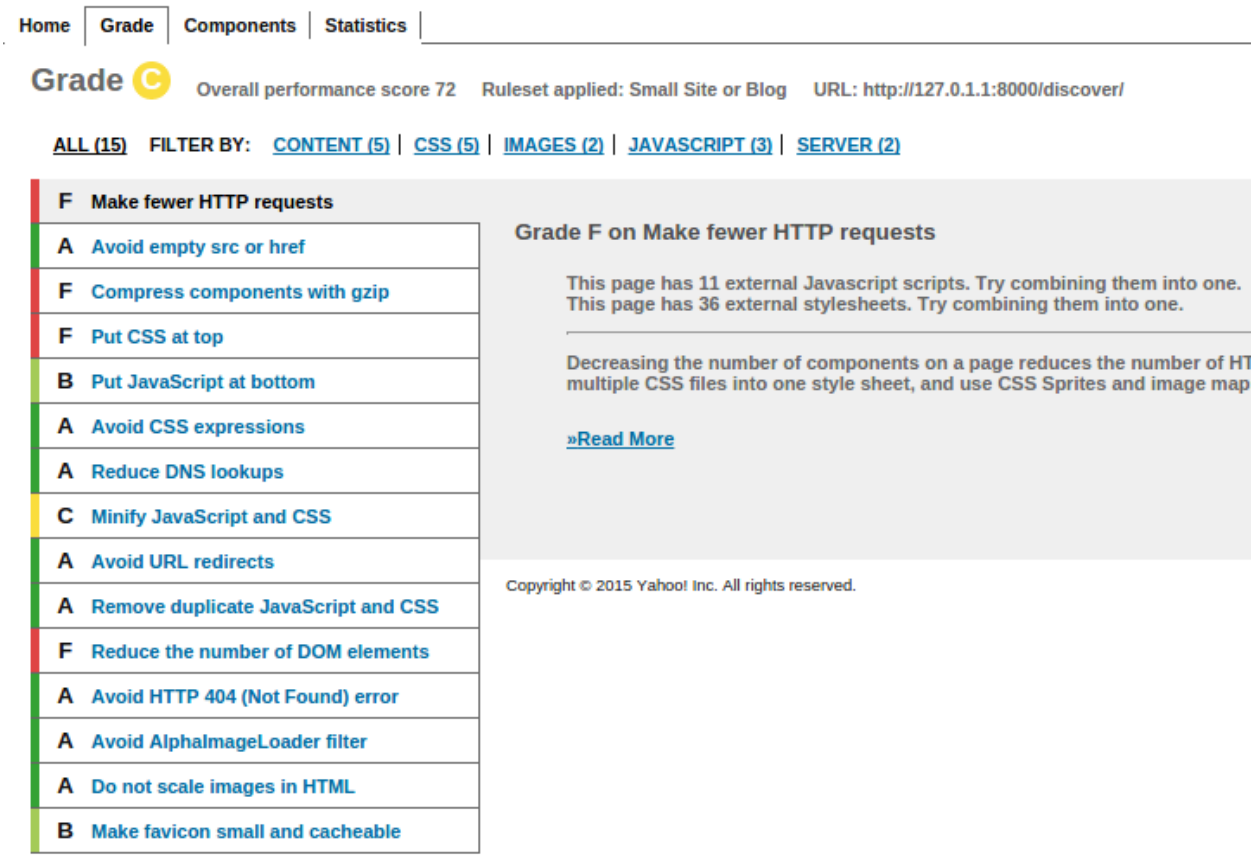


Figure 9-20 Yslow results for manage devices page

- Make fewer HTTP requests: The given page has 11 external JavaScript that is being loaded, 3 of which are loaded as part of the design requirements. All of the other scripts can be combined into one post development stages. For ease of development, these JavaScript files are kept separate.

- Compress components with gzip: The above reasoning applies here. Since CSS files are being used and modified in the course of development, these files are not compressed. Once the development is complete, these files can be merged or compressed for page load.

- Put CSS at top: The CSS discussed here refers to the theme changer script. This is an optional component that will be available as an add-on in the future and available for now in the current release.

- Minify JavaScript and CSS: The JS and CSS files will be minified post development.

- Reduce the number of OM elements: This is the only area in the entire results section that needs some upgrade to page standards. Being a complex page, multiple components are used. These components may be reduced in a future development version.

9.6.3 Summary

BEMOSS is under development and the above recommendations being implemented in the next version of BEMOSS. HTML compression will be employed and the client side scripts and style sheets will be merged into single files once the development is complete for every phase. The page load performance is comparable to other Django applications existing today in the Web.

10. Conclusions and Future Work

10.1 Conclusions

In recent years, smart devices are increasingly in buildings. This demands a secure platform for control of each of these devices from one single application. With small- and medium-sized buildings accounting for about 40% of energy consumption in the US, an affordable smart control application that is accessible from anywhere can help reduce energy consumption and improve energy savings. Energy consumptions and savings can be viewed real time from anywhere using a smart interface and control system that aligns to user requirements.

Various open source solutions are available today that allow a developer to build an energy management platform to control smart devices and thereby save energy. However, there is no turn-key solution available for small- and medium-sized buildings with all necessary applications and ease of use for end user interaction. Furthermore, most of these platforms are available as a per-user application that requires multiple set ups on different devices.

The objective of this thesis is to build a secure web platform for an open source BEM system. Methodologies and principles of the proposed web platform provide the necessary guidelines to build a user-friendly and developer-friendly, remotely accessible, open source and secure end user interface. It also provides a comprehensive data model solution for BEM systems. The web platform and communications platform security requirements and methods to achieve security are also discussed.

In this thesis, a comprehensive end-user platform is proposed which can allow multiple user roles and multiple users for each role. A session management solution for handling multiple users is also proposed and implemented in the prototype. The proposed design provides an admin-type interface as opposed to standard form-based interfaces provides for an easy-to-use platform for end users with minimal training. Developers choosing to implement more functionality can do so using the modularity of the web platform design. The solution also proposes the use of push notifications and an implementation this strategy using web sockets. Inter-platform communication using message queues is proposed that allows asynchronous communication with zero-loss of information in transit.

The proposed data model provides an abstraction of the smart device that allows metadata and other parameters like temperature and set points for thermostats, lighting load brightness levels and On/Off status. The data model is flexible and can be extended for any new device type added to the application. The proposed design also provides a security framework embedded in the platform to allow for role-based access control and prevention of common security vulnerabilities and threats that affect BEM systems.

The proposed solution is based on the design principles for a user interface platform and was iteratively evaluated for end-user acceptance and intuitiveness to ensure minimum training for users. The interface is responsive and provides access from any device connected to the network.

Security becomes important when the application controls the entire building operation. The proposed solution enumerates the threats and vulnerabilities in an open source platform and implements prototype solutions to address them. Performance of the user interface platform is evaluated for responsiveness in different screen sizes, page response times, throughput, and the performance of client side entities.

10.2 Future Work

The proposed design and the prototype implementation is a starting point to enable secure user experience in open source BEM systems. This thesis points to several future work paths along the lines of user experience and building developer- and user- friendly applications to help in building monitoring and control and energy savings.

Software-as-a-Service / Cloud deployment: The web platform currently rests on the same server as the control platform. However, to enable efficiency and seamless access when the consumer wants to run it on low cost devices, cloud-based deployment of the solution is preferable. For cloud-based deployment, remotely accessible inter-platform communication strategies should be analyzed and the security framework may be extended. Cloud-based deployment may also be developed into a Software-as-a-service (SaaS). Software-as-a-Service at affordable costs that can accommodate different unrelated buildings requires in depth analysis of the cost criteria and the application enhancement strategies.

Visualization using 2D Building Maps: A comprehensive building map that allows the administrator to upload a map of the building and drag-and-drop device models into the map at appropriate locations can provide a real-time overview of the building operations as opposed to a theoretical reference to devices in different zones. The building map is dynamic and should be adaptive to different building conditions. Research on map implementation that allows multi-building integration can be part of the future work.

Applications to support more load types: The loads currently used support a limited number of device types. The proposed design is flexible enough to build more load models' monitoring and control interfaces. Providing turn-key solutions involve developing a complete platform for end users. An analysis of the commonly used building management devices to be incorporated into the platform will be a valuable step forward.

Code-generation APIs: The proposed solution allows a developer to add more components into the system by programming a new application. However, the solution does not provide a means for new devices to be added to the system if the platform does not support it. With compatibility algorithms being implemented to allow multiple devices to connect to the system without needing a separate device driver for every device, the same should be possible at the web platform layer as well. This requires development of front-end APIs that provide drag-and-drop capability that allow an administrator to generate new front-end interfaces, associate security models, and configure role-based access control using a separate administrator panel. Developing such an administrator model will allow multiple devices to be connected using automatic code generation for the front-end interfaces.

References

1 A Look at the U.S. Commercial Building Stock: Results from EIA's 2012 Commercial Buildings Energy Consumption Survey (CBECS). [Online] Available at: http://www.eia.gov/consumption/commercial/reports/2012/buildstock/?src=‹ Consumption Commercial Buildings Energy Consumption Survey (CBECS)-b1http://www.eia.gov/consumption/commercial/reports/2012/buildstock/?src=‹ Consumption Commercial Buildings Energy Consumption Survey (CBECS)-b1 (2015, May 25, 2015))

2 BACnet Protocol. [Online]. Available at: http://www.bacnet.org/ (2015, May 25, 2015))

3 Desigo Building Automation from Siemens. [Online]. Available at: http://www.buildingtechnologies.siemens.com/bt/global/en/buildingautomation-hvac/building-automation/building-automation-and-control-system-europe-desigo/management-station/desigo-insight/Pages/desigo-insight.aspx (2015, May 25)

4 Synco – HVAC control system for small and medium-sized multipurposed buildings. [Online] Available at: http://www.buildingtechnologies.siemens.com/bt/global/en/buildingautomation-hvac/building-automation/building-automation-for-small-applications-synco/Pages/hvac-building-automation.aspx (May 25, 2015)

5 APOGEE- Siemens Building Automation Systems. [Online] Available at: http://w3.usa.siemens.com/buildingtechnologies/us/en/building-automation-and-energy-management/apogee/pages/apogee.aspx (May 30, 2015)

6 Honeywell Building Automation Systems. [Online] Available at: http://buildingcontrols.honeywell.com/Building-Automation-Systems (May 30, 2015)

7 Metasys Building Automation Systems. [Online] Available at: http://www.johnsoncontrols.com/content/us/en/products/building_efficiency/products-and-systems/building_management/metasys.html (May 30, 2015)

8 COMMERCIAL BUILDINGS ENERGY CONSUMPTION SURVEY (CBECS). 2012 Preliminary Results. [Online] Available at: http://www.eia.gov/consumption/commercial/reports/2012/preliminary/ (May 30, 2015)

9 Project Haystack. [Online] Available at: http://project-haystack.org (May 30, 2015)

10 Tridium. [Online] Available at: http://www.tridium.com (May 30, 2015)

11 Freedomotic: Open IoT Platform.[Online] Available at: http://freedomotic.com (May 30, 2015)

12 Open Home : An Open Source Automation Platform. [Online] Available at: http://www.openremote.org/display/HOME/OpenRemote (May 30, 2015)

13 Smartthings. [Online] Available at http://www.smartthings.com (May 30, 2015)

14 Haack, J., Akyol, B., Tenney, N., Carpenter, B., Pratt, R. & Carroll, T. 2013, "VOLTTRON™: An agent platform for integrating electric vehicles and Smart Grid", IEEE, pp. 81.

15 Small- and Medium-Sized Commercial Building Monitoring and Controls Needs: A Scoping Study. [Online]. Available at http://www.pnnl.gov/main/publications/external/technical_reports/PNNL-22169.pdf (May 30, 2015)

16 Active MQ. [Online] Available at http://activemq.apache.org/ (May 30, 2015)

17 Wong, B. 2010, "Gestalt principles", Nature Methods, vol. 7, no. 12, pp. 941.

18 Hick's law. [Online] Available at http://www.etl-lab.eng.wayne.edu/adrc/Human_Factors/Hicks/Hicks_law.htm (May 30, 2015)

19 Gillan, D., Holden, K., Adam, S., Rudisill, M. & Magee, L. 1990, "How does Fitts' law fit pointing and dragging?", ACM, , pp. 227.

20 Lowdermilk, T. 2013, User-centered design: a developer's guide to building user-friendly applications, O'Reilly, Beijing.

21 Java Programming Language. [Online] Available at https://www.oracle.com/java/index.html (May 30, 2015)

22 Python Programming Language [Online]. Available at https://www.python.org/ (May 30, 2015)

23 Django Web framework.[Online] Available at https://www.djangoproject.com (May 30, 2015)

24 Flask: web development one drop at a time. [Online] Available at http://flask.pocoo.org (May 30, 2015)

25 Pautasso, C. & Wilde, E. 2010, "RESTful web services: principles, patterns, emerging technologies", ACM, pp. 1359.

26 Hintjens, P. 2013, ZeroMQ, O'Reilly Media, Inc, Sebastopol, CA.

27 Videla, A. & Williams, J.J.W. 2012, RabbitMQ in Action: Distributed Messaging for Everyone, Manning Publications Company, Saintmpford; LaVergne.

28 Celery:Distributed Task Queue. [Online] Available at http://www.celeryproject.org. (May 30, 2015)

29 Cochran, D. 2012, Twitter Bootstrap Web Development, Packt Publishing, Limited, Birmingham.

30 Shklar, L. & Rosen, R. 2003, Web application architecture: principles, protocols, and practices, John Wiley, Hoboken, NJ; Chichester, West Sussex, England.

31 "The Websocket Protocol", RFC 6455, December 2011.

32 "HTML5 WebSocket" 2011, in Apress, Berkeley, CA, pp. 241-261.

33 Pimentel, V. & Nickerson, B.G. 2012, "Communicating and Displaying Real-Time Data with WebSocket", IEEE Internet Computing, vol. 16, no. 4, pp. 45-53.

34 P. Lubbers and F. Greco, “HTML5 Web Sockets: A Quantum Leap in Scalability for the Web,” SOA World Magazine, Mar. 2010; http://soa.sys-con.com/ node/1315473.
35 P. Lubbers and F. Greco, “HTML5 Web Sockets: A Quantum Leap in Scalability for the Web,” SOA World Magazine, Mar. 2010; http://soa.sys-con.com/ node/1315473.

36 Model-View-Controller Architecture [Online]. Available at https://developer.chrome.com/apps/app_frameworks (May 30, 2015)

37 Vyatkin, V., Deng, Y., Mayer, H., Pang, C., Majid, S., Dependable Communication and Computation Systems, Luleå University of Technology, Department of Computer Science,Electrical and Space Engineering & Computer Science 2013, "System-level architecture for building automation systems: Object-orientated design and simulation".

38 Guinard, D.. A Web of Things Application Architecture - Integrating the Real World into the Web. Ph.D. thesis; ETHZ; 2011.

39 Gérôme Bovet, Jean Hennebert, Will Web Technologies Impact on Building Automation Systems Architecture?, Procedia Computer Science, Volume 32, 2014, Pages 985-990, ISSN 1877-0509, http://dx.doi.org/10.1016/j.procs.2014.05.522

40 Tobias Teich, Sebastian Wolf, Tim Neumann, Sebastian Junghans, Susan Franke, Concept for a Service-oriented Architecture in Building Automation Systems, Procedia Engineering, Volume 69, 2014, Pages 597-602, ISSN 1877-7058, http://dx.doi.org/10.1016/j.proeng.2014.03.031.

41 Granzer, W.; Kastner, W.; Neugschwandtner, G.; Praus, F., "A modular architecture for building automation systems," Factory Communication Systems, 2006 IEEE International Workshop on , vol., no., pp.99,102, 0-0 0. doi: 10.1109/WFCS.2006.1704133

42 Penner, R.R.; Steinmetz, E.S., "Model-based automation of the design of user interfaces to digital control systems," Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on , vol.32, no.1, pp.41,49, Jan 2002; doi: 10.1109/3468.995528

43 Opto 22: Building an HMI that Works: New Best Practices for Operator Interface Design. [Online] Available at http://www.opto22.com/documents/2061_High_Performance_HMI_white_paper.pdf (May 30, 2015)

44 Charles Wiecha, William Bennett, Stephen Boies, John Gould, and Sharon Greene. 1990. ITS: a tool for rapidly developing interactive applications. ACM Trans. Inf. Syst. 8, 3 (July 1990), 204-236. DOI=10.1145/98188.98194 http://doi.acm.org/10.1145/98188.98194

45 Infosecurity (Magazine) - Researchers hack Google’s Australian office building. [Online] Available at http://www.infosecurity-magazine.com/news/researchers-hack-googles-australian-office/ (May 30, 2015)

46 Wired (Magazine) - Vulnerability Lets Hackers Control Building Locks, Electricity, Elevators and More. [Online] Available at http://www.wired.com/2013/02/tridium-niagara-zero-day/ (May 30, 2015)

47 The Register (Magazine) - Internet of Thieves: All that shiny home security gear is crap, warns HP. [Online] Available at http://www.theregister.co.uk/2015/02/10/iot_home_insecurity/ (May 30, 2015)

48 David Fisk. Cyber security, building automation, and the intelligent building. Intelligent Buildings International, 2012; 1 DOI: 10.1080/17508975.2012.695277.

49 Granzer, W., Praus, F. & Kastner, W. 2010, "Security in Building Automation Systems", IEEE Transactions on Industrial Electronics, vol. 57, no. 11, pp. 3622-3630.

50 Novak, T., Novak, T., Treytl, A., Treytl, A., Palensky, P. & Palensky, P. 2007, "Common approach to functional safety and system security in building automation and control systems", IEEE, , pp. 1141.

51 D. Hwang, P. Schaumont, K. Tiri, and I. Verbauwhede, “Securing embed- ded systems,” IEEE Security Privacy, vol. 4, no. 2, pp. 40–49, Mar. 2006.

52 W. Granzer, C. Reinisch, and W. Kastner, “Denial-of-service in automation systems,” in Proc. IEEE Int. Conf. Emerging Technol. Factory Autom., 2008, pp. 468–471.

53 Michele Crabb. Curmudgeon's Executive Summary. In Michele Crabb, editor, The SANS Network Security Digest. SANS, 1997. Contributing Editors: Matt Bishop, Gene Spafford, Steve Bellovin, Gene Schultz, Rob Kolstad, Marcus Ranum, Dorothy Denning, Dan Geer, Peter Neumann, Peter Galvin, David Harley, Jean Chouanard.

54 C. Cowan, C. Pu, D. Maier, J. Walpole, P. Bakke, S. Beattie, A. Grier, P. Wagle, Q. Zhang, and H. Hinton, “StackGuard: Automatic adaptive detection and prevention of buffer-overflow attacks,” in Proc. USENIX Security Conf., 1998, pp. 63–78.

55 Wassermann, G. & Su, Z. 2008, "Static detection of cross-site scripting vulnerabilities", ACM, pp. 171.

56 Alexenko, T., Alexenko, T., Jenne, M., Jenne, M., Roy, S.D., Roy, S.D. & Zeng, W. 2010, "Cross-Site Request Forgery: Attack and Defense", IEEE, pp. 1.

57 Chandrashekhar, R., Mardithaya, M., Thilagam, S. & Saha, D. 2012, "SQL Injection Attack Mechanisms and Prevention Techniques" in Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 524-533.

58 Evans, D.; Larochelle, D., "Improving security using extensible lightweight static analysis," Software, IEEE , vol.19, no.1, pp.42,51, Jan/Feb 2002 doi: 10.1109/52.976940

59 Meier, J.D., "Web application security engineering," Security & Privacy, IEEE , vol.4, no.4, pp.16,24, July-Aug. 2006; doi: 10.1109/MSP.2006.109

60 Karlof, C.; Sastry, N.; Wagner, D., “TinySec: A Link Layer Security Architecture for Wireless Sensor Networks,” SenSys 04, Proceedings of the 2nd International Conference Embedded Networked Sensor Systems, pp. 162, 175, 3-5 Nov. 2004.

61 Zalewski, J., Drager, S., McKeever, W. & Kornecki, A. 2013, "Threat modeling for security assessment in cyberphysical systems", ACM, , pp. 1.

62 Security Development Lifecycle from Microsoft. [Online] Available at https://www.microsoft.com/en-us/sdl/adopt/threatmodeling.aspx (May 30, 2015)

63 The web Origin Concept. [Online] Available at http://www.ietf.org/rfc/rfc6454.txt (May 30, 2015)

64 CurveZMQ protocol. [Online] Available at http://curvezmq.org/page:read-the-docs#toc0 (May 30, 2015)

65 GCN (Magazine) – How secure are your open source-based systems? [Online] Available at http://gcn.com/articles/2015/01/21/open-source- components.aspx. (May 30, 2015)

66 Modbus Protocol. [Online] Available at http://modbus.org/ (May 30, 2015)

67 Khamphanchai, W.; Saha, A.; Rathinavel, K.; Kuzlu, M.; Pipattanasomporn, M.; Rahman, S.; Akyol, B.; Haack, J., "Conceptual architecture of building energy management open source software (BEMOSS)," Innovative Smart Grid Technologies Conference Europe (ISGT-Europe), 2014 IEEE PES , vol., no., pp.1,6, 12-15 Oct. 2014

68 BEMOSS Website. [Online] Available at http://www.bemoss.org (May 30, 2015)

69 Wunderground Weather API. [Online] Available at http://www.wunderground.com/weather/api/ (May 30, 2015)

70 Using DRY: Between Code Duplication and High-Coupling. [Online] Available at http://www.infoq.com/news/2012/05/DRY-code-duplication-coupling (May 30, 2015)

71 Lauesen, S. 2005, User interface design: a software engineering perspective, Pearson/Addison-Wesley, New York; Harlow, England.

72 Advantages of Icon Fonts. [Online] Available at https://www.iconsmind.com/advantages-icon-fonts/ (May 30, 2015)

73 WebSockets – An introduction [Online] Available at https://gist.github.com/subudeepak/9897212 (May 30, 2015)

74 Building Data Visualization . [Online] Available at http://eetd.lbl.gov/newsletter/cbs_nl/nl08/cbs-nl8-data-viz.html (May 30, 2015)

75 Google Nest Thermostat. [Online] Available at https://nest.com/ (May 30, 2015)

76 Ecobee Thermostat. [Online] Available at https://www.ecobee.com/ (May 30, 2015)

77 CIA Triad. [Online] Available at http://www.cyber-51.com/cyber51-blog/145-cia-triad (May 30, 2015)

78 OWASP Top Ten 2013. [Online] Available at https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project#tab=OWASP_Top_10_for_2013 (May 30, 2015)

79 Runtime Protection and Recovery from Web Application Vulnerabilities. [Online] Available at http://research.microsoft.com/en-us/um/people/livshits/papers/pdf/securifly_tr.pdf (May 30, 2015)

80 Cross Site Scripting (XSS). [Online] Available at https://www.owasp.org/index.php/Cross-site_Scripting_%28XSS%29 (May 30, 2015)

81 Cross Site Request Forgery (CSRF) [Online] Available at https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF) (May 30, 2015)

82 Broken Authentication and Session Management. [Online] Available at https://www.owasp.org/index.php/Top_10_2013-A2-Broken_Authentication_and_Session_Management (May 30, 2015)

83 Missing Function Level Access Control. [Online] Available at https://www.owasp.org/index.php/Top_10_2013-A7-Missing_Function_Level_Access_Control (May 30, 2015)

84 Linux Security [Online] Available at https://github.com/VOLTTRON/volttron/wiki/Linux-Platform-Hardening-Recommendations-for-VOLTTRON-users (May 30, 2015)

85 Network Mapper. [Online] Available at http://nmap.org/ (May 30, 2015)

86 Web Application Security Frame from Microsoft. [Online] Available at https://msdn.microsoft.com/en-us/library/ms978518.aspx (May 30, 2015)

87 Microsoft's Threat Modeling for Web Applications. [Online] Available at https://msdn.microsoft.com/en-us/library/ff648006.aspx (May 30, 2015)

88 Matsumoto, M. & Nishimura, T. 1998, "Mersenne twister", ACM Transactions on Modeling and Computer Simulation, vol. 8, no. 1, pp. 3-30.

89 CSRF REASON_NO_REFERER with meta referrer tags. [Online] Available at http://python.6.x6.nabble.com/CSRF-REASON-NO-REFERER-with-meta-referrer-tags-td5085019.html (May 30, 2015)

90 PBKDF2 Algorithm. [Online] Available at https://tools.ietf.org/html/rfc2898#appendix-A.2 (May 30, 2015)

91 The TLS protocol. [Online] Available at https://tools.ietf.org/html/rfc5246 (May 30, 2015)

92 The Galois / Counter Mode of Operation (GCM). [Online] Available at http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/gcm/gcm-spec.pdf (May 30, 2015)

93 Publish-Subscribe pattern in Message Queues. [Online] Available in RFC 29 at http://rfc.zeromq.org/spec:29/PUBSUB (May 30, 2015)

94 Curve 25519 protocol. [Online] Available at https://tools.ietf.org/html/draft-josefsson-tls-curve25519-04 (May 30, 2015)

95 Systemd System and Service Manager. [Online] Available at http://www.freedesktop.org/wiki/Software/systemd/ (May 30, 2015)

96 SWEBOK Guide. [Online] Available at http://www.computer.org/web/swebok/index;jsessionid=6cb1db634a2ac440af79b49bf7d4 (May 30, 2015)

97 Hoxmeier, J. A., & DiCesare, C. (2000). "System Response Time and User Satisfaction: An Experimental Study of Browser-based Applications," In Proceedings of the 4th CollECTeR Conference on Electronic Commerce, Breckenridge 11 April 2000.

98 Liu, Z., Saifullah, Y., Greis, M. & Sreemanthula, S. 2005, "HTTP compression techniques", pp. 2495.

99 Neoload. [Online] Available at http://www.neotys.com/product/overview-neoload.html (May 30, 2015)

100 Firebug. [Online] Available at http://getfirebug.com/ (May 30, 2015)

101 YSlow Web application testing tool. [Online] Available at http://yslow.org/ (May 30, 2015)

102 Webkit. [Online] Available at https://www.webkit.org/ (May 30, 2015)

103 Moz-transform. Available at https://developer.mozilla.org/en-US/docs/Web/CSS/transform (May 30, 2015)