CIS 4911 - Senior Project (U01)

Web Dashboard for Addigy

Javier Carmona

Francisco Marcano

October 27, 2014

Jason Dettbarn

**Final Document**

# 

# Copyright and Trademark Notices

All Addigy specific logos and slogan material is copyright Addigy, Inc. All other materials developed for this project is available open source under MIT License:

http://opensource.org/licenses/MIT

# 

# Abstract

This document will help to outline and explain all of the functional components for the Web Dashboard for Addigy project. The document will be broken down into different chapters each of which will be in charged of explaining different aspects.

The first chapter will introduce the problem while providing the scope of the system, as well as any definitions that will be used throughout the document. The second chapter talks the limitations of the current system and provide alternative solutions to the problem. The third chapter talks about Project organization and development of tasks and milestones. The fourth chapter talks about the requirements that the new system will need in order to solve the problem. The fifth chapter talks about the design of the new system and any subsystems that are needed in order to create the whole system. The sixth chapter talks about the design of the system in detail while focusing on the system models. Finally chapter seven talks about the system validation tests that were performed on the new system.

# 

# Table of Contents

[Copyright and Trademark Notices](#h.bhdzb0l3s9vi)

[Abstract](#h.5ojjhnquyf55)

[Table of Contents](#h.aw54aa5i9ql2)

[1. Introduction](#h.vloxe4snkv8d)

[1.1 Problem Definition](#h.xamyk1sa2frx)

[1.2 Scope of System](#h.xsggis6ax9na)

[1.3 Overall Development Methodology](#h.21cm2j9j4mnf)

[1.4 Terminology](#h.6v0kcvdcffo)

[1.5 Overview of The Document](#h.u4srmhpvqtu6)

[2. Feasibility Study](#h.9bma6ypkcecv)

[2.1 Description of Current System](#h.vbcn4qgnn9r1)

[2.2 Description of Alternative Solutions](#h.jx7e4t7m54cv)

[2.3 Recommendation](#h.qshsam552hfq)

[3. Project Plan](#h.qun6kp8m45i5)

[3.1 Project Organization](#h.9fnho69aojyb)

[3.1.1 Project Personnel](#h.xim3m6r90e0l)

[3.1.2 Hardware and Software Resources](#h.mj5ijln47lee)

[3.2 Identification of Tasks, Milestones, and Deliverables](#h.w33zqynytt87)

[3.3 Cost of The Project](#h.20t9h1w5s7lr)

[4. System Requirements](#h.z9m8cj7bsdto)

[4.1 Functional and Nonfunctional Requirements](#h.8rjzdvrzrpyx)

[4.2 Requirements Analysis](#h.lppxo95xh6mg)

[4.2.1. Scenarios](#h.hsf7v0f2xcz1)

[4.2.2. Use Case Model](#h.gs2sbx274odq)

[4.2.3. Static Model](#h.8pc3uwix4tn9)

[4.2.4. Dynamic Model](#h.mo31nla0422b)

[5. System Design](#h.xcan711lp35n)

[5.1 Overview](#h.xl0rpoe0qvod)

[5.2 Subsystem Decomposition](#h.90r1qg8nmnfu)

[5.3 Hardware and Software Mapping](#h.7of0ffw7fejz)

[5.4 Persistent Data Management](#h.82t3tcb208at)

[5.5 Security/Privacy](#h.7h3f67lpaerm)

[6. Detailed Design](#h.9kmlbh37m0gn)

[6.1 Overview](#h.ud2t7j8yi7ft)

[6.2 Static Model](#h.o2rt6stmk8jg)

[6.3 Dynamic Model](#h.xh64ohaavxuh)

[Multiple Machine View & Single Machine View](#h.bl8i06fzhct1)

[Multiple Machine Controller](#h.4cjdlsbcnzvf)

[Single Machine Controller & PubNub](#h.6a2e2yfvfrui)

[PagerDuty Data API & Addigy Data API](#h.t3bxyoap7ch7)

[6.4 Code Specification](#h.jrsr94huemw0)

[Multiple Machine View & Single Machine View](#h.6gmombmccu59)

[Multiple Machine Controller](#h.vqcpcaqp84q)

[Single Machine Controller](#h.opnojik23daa)

[7. System Validation](#h.f35gia8qxd0t)

[8. Glossary](#h.912xodnsah2y)

[9. Appendix](#h.ss4pjnfiy0d7)

[9.1 Appendix A - Project Schedule](#h.w9r1h4yhfx9g)

[9.2 Appendix B - All Use Cases](#h.of362didayyz)

[9.3 Appendix C - User Interface Design](#h.4se1u0mvch6j)

[9.4 Appendix D - Analysis Models](#h.ampnq1mjsf1v)

[9.5 Appendix E - Design Models](#h.fa3dm9mj98t2)

[9.6 Appendix F - Documented Class interface](#h.1rg3i6lemahy)

[9.7 Appendix G - Diary of Meeting and Task for Entire Semester](#h.ktc5kyvllelj)

# 

# 

# 

# 1. Introduction

This document will serve as the final paper which will outline the complete development cycle from start to finish. Contained in this document, is all of the information collected and developed for the new Addigy Webdashboard which aims to provide a lightweight portal to the information collected from the Addigy solution.

The following is the information you will be able find in this document: feasibility study done on the benefits of creating a new dashboard from scratch, the project plan aimed to keep the project within a 2 month timeline, the system requirements that needed to be met when creating the new dashboard, the system design which shows all of the major subsystems in the new implementation, More in depth design details showing the class interfaces for each subsystem, and finally the steps taken to validate the whole system.

## 1.1 Problem Definition

Many industry professionals in Information Technology fields lack the tools necessary to successfully manage mac computers in their networks. The Addigy product aims to solve that problem by providing statistical data on any machine that is present in their network.

Currently the Addigy product does not have any user friendly way of showcasing their statistical data to these high level professionals in the industry. The Web Dashboard for Addigy aims to solve that problem by showing the aforementioned statistical data in a nice and clean format, that is easy for any professionals in the industry to follow.

## 1.2 Scope of System

The scope of the system for this project covers minor changes to the existing system’s implementation to provide meaningful data to the users if not currently present. Also, all changes needed to the Web Dashboard in order to deliver a product that meets the client standards. Finally, any revisions needed in order to provide the client with a product they are happy with without going into scope creeping.

## 1.3 Overall Development Methodology

The overall development methodology that was approached in this project was an agile and flexible development cycle. Thanks to this, all of the features met their projected deadlines and even allowed for additional requirements through the lifespan of the development cycle. Also, the usage of modern practices such as an MVC pattern, allowed for code refactoring to be easy and fast, allowing more time to implement new features.

## 1.4 Terminology

Some of the acronyms or abbreviations used throughout this document can be found detailed here:

* **WDA**: Web Dashboard for Addigy
* **REST**: Representational State Transfer
* **API**: Application Programming Interface

## 1.5 Overview of The Document

The following is the information you will be able find in this document:

* Feasibility study done on the benefits of creating a new dashboard from scratch
* The project plan aimed to keep the project within a 2 month timeline
* The system requirements that needed to be met when creating the new dashboard
* The system design which shows all of the major subsystems in the new implementation
* More in depth design details showing the class interfaces for each subsystem
* The steps taken to validate the whole system.

# 

# 

# 2. Feasibility Study

In this chapter, this document will study how feasible is this new system by describing what already exists, what it is meant to become, its purpose, and what is required from it. Then it will explain similar solutions that already exist in the market that seek to solve the same problem. The different existing system will be compared amongst themselves and to the new system being proposed in different terms of feasibility. Finally, recommendations will be done for the project.

## 2.1 Description of Current System

The current system does NOT have an Audit Dashboard incorporated. Addigy currently provides other IT Management tools, but has not developed an Audit Dashboard for customers to understand what assets and configuration they have. The ability for customers to see the state of their assets is arguably the most important feature, and necessary for being able to decide what changes/policies need to be applied on the assets.

## 2.2 Description of Alternative Solutions

Alternative to the Addigy Web Dashboard, there are similar dashboards in the market that seek to solve the same problem of managing different computers in the network and displaying meaningful information about them.

Such a dashboard is provided by the software called IpMonitor. This software allows to track machines in the network of an organization and makes visual representation of the current state of these machines ranging from network usage to cpu usage. IpMonitor has two issues against it, which are the fact that is not free to use as are other alternatives, and that it is only able to monitor Windows systems.

Another dashboard is provided by SpiceWorks. Like IpMonitor, SpiceWorks allows you to keep track of the different computers on the network and also shows different statistics about these computers. Unlike IpMonitor, SpiceWorks is free to use since they are paid by advertisers to bring IT professionals IT related products through their software. SpiceWorks only falls short on it only being able to collect information for Windows Systems.

## 2.3 Recommendation

Based on the analysis made on the different alternatives available today, the Addigy Web Dashboard would be the best possible solution for the current project. There are a couple of reasons why this is the case.

The main factor is that the project maintains its own name; no third-party application are used to realize the project’s goal.

Second factor is that the project covers a wider audience and so it is a better alternative than the other products described.

Third factor is that a lot of the groundwork is already done on the project and the data is already available, allowing for a Web Interface to be quickly developed without the need to wait on that much, if any, backend implementation.

Fourth and last factor, is that this allows for the project to be developed in any style and with any technology that allows for rapid and agile development. This greatly increases productivity and maintainability for the project. By choosing an alternative product, the project loses its flexibility and direction.

# 

# 3. Project Plan

This project is to be organized and developed using a variety of different tools gears towards an agile development approach. By using an agile development approach, it ensures that the project is constantly being updated and available to deploy at any time during its life cycle.

## 3.1 Project Organization

The project is to be split up and organized based, again, on an agile development approach. The following series of steps will be taken in order to make sure that the development lifecycle is productive with very little downtime.

1. **Scope**
   1. Meat with client and discuss all of the functional requirements, non-functional requirements, and end goals for the project
   2. Define any additional resources needed for the completion of the project
   3. Obtain all of the resources necessary to realize the project
2. **Analysis**
   1. Define any software and hardware specifications needed in order to develop the product taking into account the functional requirements that need to be realized.
   2. Meet with development team to discuss the possibility of the requirements
   3. Discuss the financial feasibility for the project with the client
   4. Meet with the development team to organize and distribute the requirements needed based on individual strengths and weaknesses.
   5. Develop the project’s delivery timeline
   6. Obtain approval to proceed with the project
   7. Provide the tools necessary to tackle the project using agile development principles.
3. **Design**
   1. Review any software needed to complete the application.
   2. Develop functional specifications and prototypes based on functional specifications
   3. Review prototypes with client and record feedback
   4. Incorporate any feedback while trying to avoid any scope creep
   5. Obtain approval for completion

## 3.1.1 Project Personnel

The project will have a total of 3 members working on various aspects of the application. Each member will be responsible for following the project timeline while promptly delivering any tasks assigned to them. The team members are:

1. **Jason Drettbarn**
   1. **Title**
      1. Project Manager
   2. **Duties**
      1. Communicate with the client on a regular basis
      2. Provide the team with the software and hardware needed to complete the tasks at hand
      3. Provide the team with the functional requirements needed.
2. **Javier Carmona, Francisco Marcano**
   1. **Title**
      1. Developer
   2. **Duties**
      1. Communicate with the project manager and other team members with important project information.
      2. Promptly delivering the tasks assigned while following the project timetable.
      3. Provide immediate feedback of any issues that are slowing down the project development cycle.
      4. Using the provided software in order to keep the project organized and visible to other team members.

## 3.1.2 Hardware and Software Resources

Addigy will provide AWS EC2 instances for running a hosted web server capable of driving the AngularJS webclient. Developer can run local instances during the development process and/or can leverage AWS resources at the same time.

## 3.2 Identification of Tasks, Milestones, and Deliverables

1. Dev and Product servers are set up
2. First Iteration: Scenario-Based Implementation - Unit Testing
3. Second Iteration: Scenario-Based Implementation - Unit Testing
4. Third Iteration: Scenario-Based Implementation - Unit Testing
5. Fourth Iteration: Scenario-Based Implementation - System Testing
6. Fifth Iteration: Scenario-Based Implementation - System Testing
7. Fifth Iteration: Scenario-Based Implementation - System Testing
8. Sixth Iteration: Scenario-Based Implementation - System Testing
9. Final deliverable
10. Final posters
11. Showcase

## 3.3 Cost of The Project

The Addigy Web Dashboard will be developed using an underlying infrastructure that is already in place. There will be no costs for the dashboard itself.

# 

# 4. System Requirements

The new system that is proposed for the Web Dashboard for Addigy is a lightweight, easy to use web application which provides key features needed by IT professionals to manage machines. The features include way to locate machines based on geo-location, way to look at machine groups and make quick decisions, ability to service individual machines when in critical conditions, the ability to terminate processes in an individual machine, etc.

## 4.1 Functional and Nonfunctional Requirements

* **Functional requirements:**
  + The system shall display what status the computers are in from the map (green/healthy or red/issue)
  + The system shall show how many machines have encrypted disk or are unencrypted
  + The system shall show what is the make-up of the Mac assets such as macbook air, macbook pro, mac mini, etc.
  + The system shall show the uptime of the machines such as the time that have been online in the last hour / day / week.
  + The system shall show Apple warranty data such as whether the machines are still under warranty, warranty expiring within 30 days, or out of warranty.
  + The system shall display where are the computers located geographically.
  + The system shall display alerts based on critical machine states.
  + The system shall allow users to mark alerts as acknowledged.
  + The system shall allow users to mark alerts as resolved.
  + The system shall allow users to re-assign alerts to other Admin users.
  + The system shall be able to show real time information for a single computer.
  + The system shall show the different processes running on a single computer.
  + The system shall be able to start or stop different processes running on a single computer.
  + The system shall be able to run and report bandwidth stats such as download and upload speeds.
  + The system shall report what ports are being used in the target machine and report the ports with the used protocols.
  + The system shall display what network ports are being used in the machine and what protocols are being utilized.
  + The system shall provide a chat for the administrator to interact with the user of the machine.
* **Non-functional requirements:**
  + The system shall use HTML, CSS, and JavaScript.
  + The system must keep itself up to date to no more than 30 minutes delay.
  + The system shall poll the Addigy web service for updated data, and render the changes on the dashboard.
  + The system shall be able to understand HTTP/HTTPS protocols.
  + The system shall be able to interpret JSON notation.
  + The system shall use Google Map APIs.

## 4.2 Requirements Analysis

In this section, we will look at the different fictional scenarios that could occur while using the Addigy Web Dashboard. Then we will examine the Use Case Model of the dashboard. Lastly, we will go over the static and dynamic models of the software.

## 4.2.1. Scenarios

This section will contain some different scenarios the user might find themselves in when using the WDA product. This scenarios are listed in no particular order based on weight or importance. After the first scenario, which serves to show the process from a logged out state, users will be assumed to have logged into the application.

1. Bob uses his web browser and navigates over to the WDA project website. Once in the login screen, Bob uses his credentials to log in. Once logged in, Bob is presented with the Dashboard showing information.
2. Chris is in the home page for the application and decides to change some settings/preference. On the home screen, Chris clicks on the gear icon and he is taken to the settings/preferences page in which he is able to change any settings and/or preferences listed in the page. Once Chris is happy with the changes he made, he can save the changes by clicking on the “Save” button.
3. Mike wants to geolocate the computers that are having problems in order to send IT professionals to fix them. Once in the home page, Mike can look for the map feature in the page in order to see where in the map the computers are located. Hovering or clicking on the computers on the map will change the dashboard to showcase individual data for that particular computer. At this time, Mike can quickly see crucial information about the computer and successfully assess the problem and come up with a solution that matches the statistical data provided by the application.
4. John is looking at the dashboard and sees that on the map feature, there is a group of computers on one location with one computer having issues. He wants to only look at the specific group of computers in order to help him troubleshoot the issue. John clicks on the problematic location of the map, which makes the information on the dashboard be about the computers in the selected location only.
5. Michael is looking at the group of computers in a specific location of the map and sees that one computer is struggling with the available memory. He decides that he wants to see statistics about this computer only in order to help him identify what is causing the memory issues. Michael clicks on the name of the specific machine and the dashboard starts displaying real time information about the selected machine.
6. Ariel is looking at a specific computer that is having issues with memory. As he looks through the processes running in that specific machine, he notices that a non-important process has allocated most of the memory of the system. Ariel decides that he will terminate the process he identified by clicking the “kill button” for the specific process.
7. Ralph wants to backup an important folder for a specific machine that he is looking at from the dashboard. He enters the command he wants to execute in order to backup the target folder and it is executed on the machine through the dashboard.

## 

## 4.2.2. Use Case Model

Refer to **Appendix B**

The use cases show how a system administrator and the user of a machine interact with the different features of the Addigy dashboard. System administrators deal directly with the main Addigy dashboard in order to see commulative information about all the machines of an organization, and the single machine dashboard to deal with features that are specific to one machine such as bandwidth tests and processes. The user of the machines interact with the system through the chat capabilities provided by the single machine agent that was installed in the machine through the Addigy services.

## 4.2.3. Static Model

Refer to **Appendix D.**

The static model shows what are the different classes present on the Addigy Dashboard system, what are their capabilities, and the relationship between them. We attempt to reveal the interactions between the Administrator, the Main Dashboard, the Single Machine Dashboard, the existing Addigy Services, and the Publish-Subscribe architecture.

## 4.2.4. Dynamic Model

Refer to **Appendix D.**

This section shows the sequence diagrams and state diagrams for the Addigy dashboard. It reveals the sequence of execution of the system as different scenarios play out. It also attempts to clarify different interactions between the different subsystems that are interacting with the dashboard such as the Addigy Web Services, PagerDuty, and PubNub.

# 

# 5. System Design

This chapter explains how the whole system was broken down into its different components. It will explain their purpose and how they work together. The following are the use cases that were implemented for the project:

* **Multiple Machine View**
  + Examine Dashboard
  + Examine Machine Location
  + Acknowledge Machine Alerts
  + Resolve Machine Alerts
  + Re-assign Machine Alerts
* **Single Machine View**
  + Examine Single Machine
  + Start Process in One Machine
  + Terminate Process in One Machine
  + Get Machine Bandwidth
  + Chat with Machine User
  + See Active Ports in Machine

## 5.1 Overview

The Addigy Web Dashboard is a web based dashboard developed using a number of different systems and services. It followed a Three-tier architecture in order to separate the Presentation Tier from the Logic Tier and the Data Tier, this is shown in the System Design diagram of **Appendix F**.

The presentation tier consists of:

* The Web Dashboard Multi-Machine View has cummulative information for all the machines of an organization.
* The Web Dashboard Single-Machine View contains real-time data regarding one specific machine as well as ways to interact with it.
* The Agent Chat Interface provides a two-way communication channel for the user administrator and the person using the machine being remote controlled.

The Logic Tier consists of systems that work behind the scenes in order to update and handle interactions for the presentation tier. The systems in it are:

* The Multi-Machine controller is in charge of updating the dashboard and responding to user interactions such as filtering machines by their state.
* The PagerDuty Integration allows the Multi-Machine controller to show PagerDuty alerts and allows the controller to act upon them.
* The Single Machine Controller handles user interactions with the single-machine dashboard and communication with the PubNub services.
* PubNub provides a channel through which the Single Machine Controller and the Single Machine Agent can communicate in order to exchange data and commands.
* The Single Machine Agent allows the administrator to extract information from the target machine and allows the user of the machine to communicate with the administrator.

The Data Tier contains services that are in charge of maintaining persistent information about the application. These are:

* The Addigy Data API holds cummulative information about all the machines of an organization.
* The PagerDuty Data API contains alerts that have been raised by the system and provides ways of acting upon them.

## 5.2 Subsystem Decomposition

The **Web Dashboard Multi-Machine View**and**The Web Dashboard Single Machine View** were developed using HTML and CSS. It takes advantage of frameworks such as BootStrap and JQuery to display functionalities and data processed by the Logic Tier. These systems require the files to be hosted in a web server, preferably a LAMP or WAMP server with the latest version of PHP.

The **Agent Chat Interface** provides a User Interface for the user of one computer to communicate with an administrator that is using the machine remotely. The User Interface is an extension It was developed using Java 8 and therefore requires JRE8 or greater in order for it to run.

The **Mutli-Machine Controller** serves as way for the administrator to retrieve and interact with the data offered by the Addigy and PagerDuty Web Services. It was developed using the AngularJS Framework. The controller is responsible for connecting to the Addigy Web Services and retrieving information regarding all the machines for an organization. It also retrieves any alerts that may have been raised through the PagerDuty services in order to be shown through the Web Dashboard Multi-Machine View. The controller handles administrator’s interactions such as filtering machines by groups, acknowledging, or escalating alerts.

The **PagerDuty Integration** provides the Multi-Machine Controller with ways of retrieving and interacting with alerts raised through the PagerDuty services. It uses a Javascript API provided by PagerDuty in order to abstract away the API functionalities to the Multi-Machine Controller.

The **Single Machine Controller** provides information and functionality to the Web Dashboard Single-Machine View by handling communication with the PubNub services. The controller is in charge of listening to a PubNub channel for any information published by an individual machine. It interprets the messages published on the channel and takes appropriate action based on the content of the messages. Some of the messages require the controller to update the view such as a new screenshot that has been taken of the target machine, and some other messages require the controller to publish messages to the channel regarding its current status. In addition, the controller handles user interactions such as running a command by publishing a message to the PubNub channel to be interpreted by the Single Machine agent. This controller requires a user account to be setup in PubNub and API keys assigned to the account.

**PubNub** is a service that allows online communication using a Publish-Subscribe architecture over a highly scalable distributed system. It allows machines wishing to communicate to open a logical channel and register/subscribe to it. By subscribing to the channel, a machine connects a TCP Socket to the PubNub services through which it will receive any message published to the channel. A machine can also publish messages to the channel and these will be delivered to all the subscribers.

The **Single Machine Agent** is in charge of collecting information about a machine and executing tasks on behalf of the administrator. It subscribes to a PubNub channel in order to communicate with the Single Machine Controller. Whenever an administrator subscribes to the channel, the agent starts actively monitoring the machine for machine specific data such as memory usage and cpu utilization, and publishing it to the channel. The agent also interprets messages received from PubNub that have been published by the Single Machine Controller such as a command to be executed, and acts upon the message received. The agent was developed using Java 8 and therefore requires JDK8 or greater installed on the machine for it to work.

The **Addigy Data API** remotely accesses the machine of an organization at time intervals in order to collect machine specific information about them. It stores all the information in databases and catalogues the data by organization. This service makes the data collected available through different Web Service API calls.

The **PagerDuty Data API** allows different applications to create alerts and provides different ways of interacting with them through Web Service Api Calls.

## 5.3 Hardware and Software Mapping

The hardware and software that were used in this project were integrated in the following ways:

* **Server Machine**
  + A linux virtual machine was used to host all of the files and folders needed to successfully run the application. The server framework used was an Apache server running PHP. The apache server acts as the link between our virtual machines virtualized hardware and the application software code. Both the **Multiple Machine View**  and the **Single Machine View** use this server in order to server the files to the client machine.
* **Client Machine**
  + All of the users that request the pages from our server run a copy of our application in their machine. Thanks to modern browser technology, they act as the link between the client machines hardware and applications JavaScript logic. Both the **Multiple Machine View** and the **Single Machine View** subsystems and their respective **Controller** subsystems utilize the client machine in order to run the logic necessary to drive the application.
  + The **Agent Chat Interface** and the **Single Machine Agent** subsystems which run directly on the client machine also depend on the hardware resources of the client machine. This machine could be a different machine from the one that the Multiple and Single Machine views depend on.
  + The **PagerDuty Integration** subsystem depends on the machine that the Multiple and Single Machine views run in. Since this subsystem is directly tied to the Multiple Machine View and Controller subsystems.
  + The **PubNub** subsystem creates a link between the software that the **Single Machine Agent** depends on and the software that the **Single Machine View** and **Controller** depends on.

## 5.4 Persistent Data Management

The application currently doesn’t handle any data persistence directly. The only persistent data that is utilized is through the use of the **PagerDuty** **Integration** subsystem. The application currently stores data based on certain machine error events that can happen at any point. The application also polls the persistent data from the PagerDuty API and makes it available to the user. Below is a data dictionary for all of the data types that are retrieved:

***--Error*** ***Data Type***

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Length** | **Type** | **Rule** |
| Title | 120 | varchar | None |
| Description | 500 | varchar | None |
| Incident Key | 120 | varchar | Primary Key |
| Assigned User | 120 | varchar | Foreign Key |

***--User Data Type***

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Length** | **Type** | **Rule** |
| Name | 120 | varchar | None |
| Email | 500 | varchar | unique |
| User Id | 120 | varchar | Primary Key |

## 5.5 Security/Privacy

The application currently does not handle authentication as this was not a feature that our mentor needed for the initial version of the project. Currently the application is keeping track of the username that get used when logging in without authenticating against a persistent data system. This is quick feature to add in future revisions as the application already has a notion of what a user is.

For data encryption, is not being implemented in this version, mainly because is not needed. Since the application is not not using any Data Persistent systems, it is not handling any user or system sensitive information.

The default Linux firewall settings were used in the Virtual Machine that was provided to run the application. No other modifications to this were done to the Virtual Machine. Security servers were not used in this application.

# 

# 6. Detailed Design

This section explains the specific details of the subsystems that make the Addigy Web Dashboard. The information includes any static models for each subsystem, any dynamic models for each subsystem, and any code specifications for each subsystem.

## 6.1 Overview

The **Multi-Machine Web Dashboard** and **Single-Machine Web Dashboard** follow a Model-View-Controller design with the Multi Machine Controller and Single Machine Controller. By displaying the data in a user interface, Dashboards serve as the View in this paradigm. The dashboard show their information by accessing HTML documents through a browser and applying CSS files to it in order to achieve the desired look and feel.

The **Multi Machine Controller** and **Single Machine controller** are in charge of interacting with Web Services and APIs in order to retrieve information for the Dashboards or handle interactions. This makes the systems controllers for the Model-View-Controller paradigm of the Dashboard.

The **Single Machine Agent** run on the machine being serviced and acts on behalf of the administrator on the machine. The agent acts as a publisher and subscriber for the PubNub Publish-Subscribe architecture.

**PubNub** uses a distributed system to handle one-to-many communication between different machines. It follows a Publish-Subscribe architecture where individual machines subscribe to a channel and publish messages to it while delivering published messages to all the subscribers.

The **Addigy Data API** and the **PagerDuty Data API** are APIs used to store and retrieve information about the machines. Addigy stores auditing data and PagerDuty stores alarms raised by the system.

## 

## 6.2 Static Model

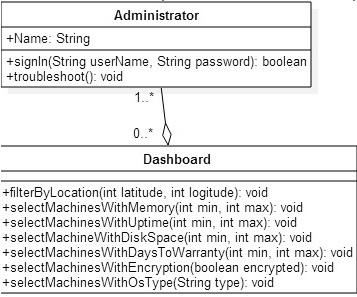
In this section, we would be looking at different parts of the system with subsystem specific class diagrams, refer to **Appendix F** for a complete Class Diagram of the System.

The Multiple Machine Dashboard - Administrator Class Diagram shown below shows the relationship between the **Multiple Machine Dashboard** and the Administrator.

The administrator has a name that serves as an identifier and can login or troubleshoot machines on the system by looking and interacting with the dashboard.

The dashboard supports a number of operations based on the machines it collects from the Addigy Data Api:

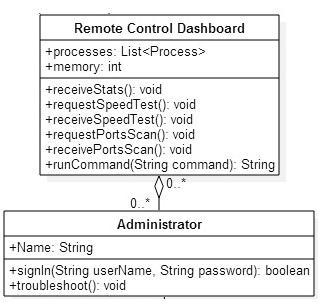
* filterByLocation - Selects the machines that are within a small distance of the specified latitude and longitude.
* selectMachinesWithMemory - Selects the machines that are within the minimum and maximum range of memory specified.
* selectMachinesWithUptime - Selects the machines that are within the minimum and maximum range of uptime specified.
* selectmachineWithDiskSpace - Selects the machines that are within the minimum and maximum range of disk space specified.
* selectMachinesWithDaysToWarranty - Selects the machines that are within the minimum and maximum amount of days to the day that the warranty will expire.
* selectMachinesWithEncryption - Selects the machines that is encrypted if a true value is passed to the operation or the ones that are unencrypted otherwise.
* selectMachinesWithOSType - Selects the machines that are currently running on the specified OS type.



*Multiple Machine Dashboard - Administrator Class Diagram*

As can be seen in the Remote Control Dashboard - Administrator Class Diagram below, the **Single Machine Dashboard** contains a list of processes currently running in a machine and how much memory is being used by the system. The dashboard could contain any number of system administrators. It supports the operations:

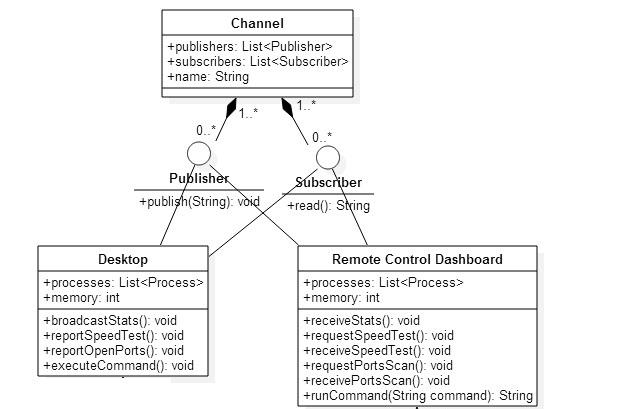
* receiveStats - Receives memory usage among other machine specific data being reported by the agent running on the remote controlled machine.
* requestSpeedTest - Requests the agent running on the remote controlled machine to run and report a speed test.
* receiveSpeedTest - Receives the results of a speed test that has been executed by the Single Machine Agent running on the machine being serviced.
* requestPortScan - Requests the Single Machine Agent to scan and report what ports are being utilized by the system and their protocols.
* receivePortScan - Received the results of a port scan that has been done on the machine being serviced.
* runCommand - Allows the dashboard to run a command on the target machine through the agent.



*Remote Control Dashboard - Administrator Class Diagram*

The Remote Control Dashboard and theDesktop use a Publish-Subscribe method modeled in the Publish-Subscribe Class Diagram below in order to communicate with each other.

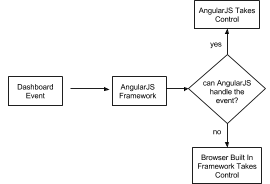
The channel is provided by **PubNub** and it allows two machines to publish and subscribe to a channel. Messages published to the channel will be delivered to all subscribers. As seen in the class diagram, both the Desktop and Remote Control Dashboard are Publishers and Subscribers of the channel. The agent running on the desktop uses the channel to publish machine specific data, speed test results and open port scans results. The agent can also execute commands as received from the dashboard by the administrator.



*Publish-Subscribe Class Diagram*

## 6.3 Dynamic Model

### Multiple Machine View & Single Machine View

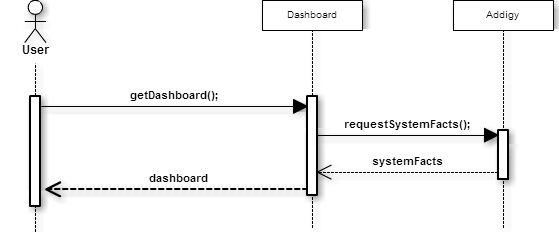
The web dashboard Multiple Machine View andSingle Machine View both utilize the same control object. In this case the main control is handled by the AngularJS framework. 

Anytime there is an event or a state change in the view, the AngularJS framework intercepts it and acts upon it based on what the user configurations. If the event cannot be handled by AngularJS, then it lets the browser built in default controller handle it.

### 

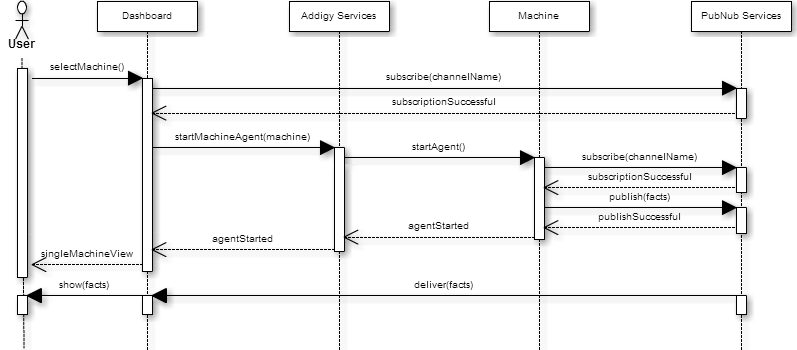
### Multiple Machine Controller

This is the controller that drives all of the flow for the Multiple Machine view. The main algorithm that this controller performs is the ability to filter machines based on the data that is currently available. It does this by checking to see if the machine data has changed in any way. If it has, then goes ahead and runs all of the different helper methods that modify the view based on the the new data that is available.



### Single Machine Controller & PubNub

This is the controller that drives all of the flow for the Single Machine view. The main algorithm that this controller performs is the ability to update all of the information for the view depending on the type of data that is returned from the **Single Machine Agent**. It does this by first checking an upcode style string which determines what type of information the Single Machine Agent returned with. This data exchange is performed by using the **PubNub** subsystem to communicate between the the Single Machine Agent and the **Single Machine View** subsystems.



### PagerDuty Data API & Addigy Data API

These subsystems use the **Multiple Machine Controller** to get data retrieved from them. This is done by calling a series of webhooks which return data depending on the subsystem that the Multiple Machine Controller is trying to call.

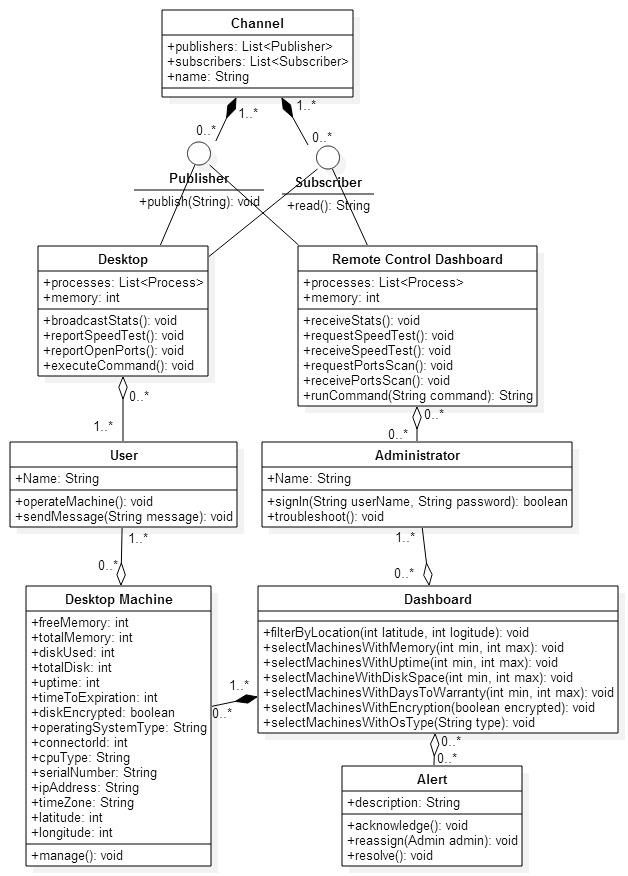
## 6.4 Code Specification

### Multiple Machine View & Single Machine View

This views do not directly have any logic or code aside from HTML and CSS. The main control object for these two views is the AngularJS framework.

### 

### Multiple Machine Controller



The Multiple Machine Controller makes use of three main class interfaces.

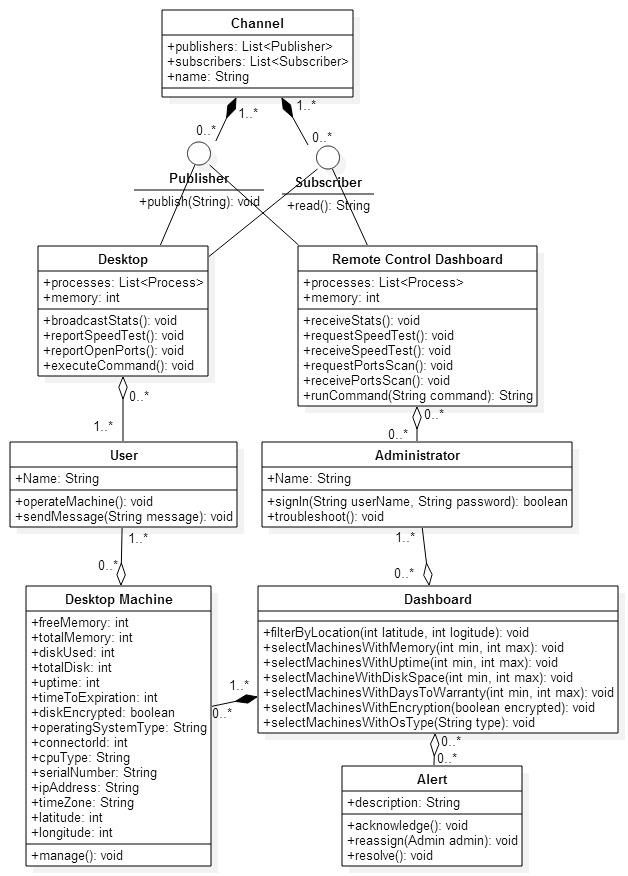
The first one is the Desktop Machine class interface. This class is used to describe all of the properties that a machine can have. This class interface will provide all of the necessary information to the rest of the class interfaces to provide a complete implementation of the Multiple Machine Controller.

The Dashboard class interface defines all of the functionality that is possible while on the Multiple Machine view. Most of the functionality outlined in the class definition is the ability to filter and select machines based on a type of machine property defined in the Desktop Machines class interface.

Finally,the Alerts class interface defines what and alert is and can do from the Multiple Machine view. The most important are the functional wrapper built around the **PagerDuty Data API** which allows the application to interact with the PagerDuty platform without any 3rd party support.

### 

### Single Machine Controller



The Single Machine Controller is made up of 3 class interfaces.

The first interface is the Desktop. This interface defines what a target machine is composed of and can perform. This essentially makes up the **Single Machine Agent**, in which the agent acts as the Desktop interface for the Single Machine controller.

The Remote Control Dashboard is the interface that defines what the web view would behave as from the perspective of the Desktop interface. The purpose of the Remote Control Dashboard is to, through the use of its functionalities, to collect information for the machine that the Desktop class interface is acting as.

Finally the Channel interface behaves as the link between both the Desktop interface and the Remote Control Dashboard interface. Facilitating the exchange of information between the two.

# 7. System Validation

The system validation for the Addigy Web Dashboard was done through manual testing of the different components due to the complexity of the Javascript libraries making it impossible to test within the constraints of a test automation framework.

**7.1. Subsystem Tests**

|  |  |
| --- | --- |
| Identifier | PubNubPublishSubscribeJson |
| Purpose | To ensure that PubNub is able to deliver JSON formatted messages published by machines using the API keys. |
| Setup | - Navigate to the PubNub developer console from two different browsers  - Connect to the channel named “mychannel” from both developer consoles |
| Input | - Enter the JSON formatted string {“msg”:”hi”} |
| Expected Output | Message {“msg”:”hi”} should be displayed on both developer consoles from the two browsers. |

**7.2. System Tests**

|  |  |
| --- | --- |
| Identifier | RunSpeedTest |
| Purpose | To ensure that speed tests are being ran and reported |
| Setup | - Navigate to the single machine dashboard for user 100007 |
| Input | - Expand the Speed Test module  - Click on the refresh button on the header of the Speed Test module. |
| Expected Output | Upload and Download markers should be spinning and after a few seconds the bandwidth speeds appear. |

|  |  |
| --- | --- |
| Identifier | KillProcess |
| Purpose | To ensure that the Halt button is able to stop selected processes |
| Setup | - Navigate to the single machine dashboard for user 100007 |
| Input | - Scroll to the processes list  - Click on the first checkbox from the list  - Click on Halt button |
| Expected Output | After a few seconds, resources held by the process are freed and process disapears from the list of processes. |

|  |  |
| --- | --- |
| Identifier | SelectMemOverEight |
| Purpose | To ensure that once a group is clicked, the list of machines in that group is shown in the left panel. |
| Setup | - Navigate to main dashboard |
| Input | - Click the memory group labeled 8gb+ |
| Expected Output | Left panel appears with a list of machines with at least 8gb of memory |

|  |  |
| --- | --- |
| Identifier | SelectMapGroup |
| Purpose | To ensure that when a group from the map is clicked, the data shown on the dashboard is only from machines in that group. |
| Setup | - Navigate to main dashboard |
| Input | - Click one group from the map |
| Expected Output | All labels now show the value of the machines only on the group clicked |

|  |  |
| --- | --- |
| Identifier | FilterReset |
| Purpose | To ensure that the FilterReset button resets all the filters set for the dashboard. |
| Setup | - Navigate to main dashboard  - Click one group from the map |
| Input | - Click on the Filter Reset Button |
| Expected Output | All labels are as when first arrived at the main dashboard |
|  |  |

|  |  |
| --- | --- |
| Identifier | RunCommand |
| Purpose | To ensure that the dashboard is able to run commands and receive output from the agent. |
| Setup | - Navigate to the single machine dashboard for user 100007  - Expand the command module. |
| Input | - Type the command “df”  - Press enter key |
| Expected Output | The command should appear on the screen followed by the command output. |

|  |  |
| --- | --- |
| Identifier | SendChatMessage |
| Purpose | To ensure that the administrator can deliver messages to the user of the machine. |
| Setup | - Navigate to the single machine dashboard for user 100007  - Expand the chat module. |
| Input | - Type the command “Hello user”  - Press enter key |
| Expected Output | A popup window, visible from the screenshot, appears on the user’s machine with the message “Hello User” |

|  |  |
| --- | --- |
| Identifier | RunNetStats |
| Purpose | To ensure that the administrator can run a netstats report on the machine and receive it through the dashboard. |
| Setup | - Navigate to the single machine dashboard for user 100007  - Expand the netstats module. |
| Input |  |
| Expected Output | A loading animation should appear and be replaced by a table with Port Numbers and Protocols of the active ports of the machine. |

**7.3. Evaluation of Tests**

|  |  |
| --- | --- |
| **Identifier** | **Result** |
| RunSpeedTest | PASS |
| KillProcess | PASS |
| SelectMemOverEight | PASS |
| SelectMapGroup | PASS |
| FilterReset | PASS |
| RunCommand | PASS |
| SendChatMessage | PASS |
| RunNetStats | PASS |

# 

# 8. Glossary

**Fact**: A piece of information about a machine.

**Agent:** A program that acts on the user’s computer on behalf of the administrator.

# 

# 9. Appendix

## 9.1 Appendix A - Project Schedule

|  |  |
| --- | --- |
| **Task Name** | **Deadline** |
| Dev and Product servers are set up | 9/8/2014 |
| First Iteration: Scenario-Based Implementation - Unit Testing | 9/8/2014 |
| Second Iteration: Scenario-Based Implementation - Unit Testing | 9/22/2014 |
| Third Iteration: Scenario-Based Implementation - Unit Testing | 10/6/2014 |
| Fourth Iteration: Scenario-Based Implementation - System Testing | 10/20/2014 |
| Fifth Iteration: Scenario-Based Implementation - System Testing | 11/3/2014 |
| Sixth Iteration: Scenario-Based Implementation - System Testing | 11/17/2014 |
| Final deliverable | 12/11/2014 |
| Final posters | 12/5/2014 |
| Showcase | 12/12/2014 |

## 

## 9.2 Appendix B - All Use Cases

***Log In***

**Actors**

User: A user is the administrator of a network.

**Entry Condition**

The user is standing in the Login page.

**Flow of Events**

The user enters his username in the username field, his password in his password field, and clicks on the login button.

**Exit Condition**

The user is successfully logged in.

**Exceptions**

An error occurred and an error message is being displayed.

**Nonfunctional Requirements**

Login form must validate user’s input.

***Examine Dashboard***

**Actors**

User: A user is the administrator of a network.

**Entry Condition**

The user is logged in

**Flow of Events**

The user logs in and is redirected to the dashboard page where the different statistics of the network can be seen.

**Exit Condition**

The user saw the status of the network.

**Exceptions**

None

**Nonfunctional Requirements**

The information of the dashboard must not be older than 30 minutes.

***Examine Machines in Location***

**Actors**

User: A user is the administrator of a network.

**Entry Condition**

The user has logged in and is looking at the main dashboard.

**Flow of Events**

User finds the location of the machines he wants to see in the map and clicks the specific location. The rest of the dashboard now displays information about the computers in the selected locations only.

**Exit Condition**

The user sees the information about the group of machines he is interested in.

**Exceptions**

None

**Nonfunctional Requirements**

None

***Acknowledge Machine Alerts***

**Actors**

User: A user is the administrator of a network.

**Entry Condition**

The user is looking at the errors in the main dashboard

**Flow of Events**

The user is presented with the alert in the main dashboard, the user then clicks on the acknowledge button for the alert of their interest

**Exit Condition**

The system received confirmation that the alert was acknowledge

**Exceptions**

The alert is not successfully acknowledge on the PagerDuty framework

**Nonfunctional Requirements**

The alert information should be polled every 10 seconds

***Resolve Machine Alerts***

**Actors**

User: A user is the administrator of a network.

**Entry Condition**

The user is looking at the errors in the main dashboard

**Flow of Events**

The user is presented with the alert in the main dashboard, the user then clicks on the resolve button for the alert of their interest

**Exit Condition**

The system received confirmation that the alert was resolved

**Exceptions**

The alert is not successfully resolved on the PagerDuty framework

**Nonfunctional Requirements**

The alert information should be polled every 10 seconds

***Re-Assign Machine Alerts***

**Actors**

User: A user is the administrator of a network.

**Entry Condition**

The user is looking at the errors in the main dashboard

**Flow of Events**

The user is presented with the alert in the main dashboard, the user then clicks on the assign dropdown and selects a the user he wishes to re-assign the alert to.

**Exit Condition**

The system received confirmation that the alert was re-assigned

**Exceptions**

The alert is not successfully re-assigned on the PagerDuty framework

**Nonfunctional Requirements**

The alert information should be polled every 10 seconds

***Examine Single Machine***

**Actors**

User: A user is the administrator of a network.

**Entry Condition**

The user has logged in and is looking at a group of computers in a specific location.

**Flow of Events**

The use identifies the machine he wants to zoom into and clicks it. The dashboard now displays real time information about that specific machine.

**Exit Condition**

The user sees specific information to one machine.

**Exceptions**

Could not connect to the machine.

**Nonfunctional Requirements**

Shall not affect performance of the dashboard or the target machine.

***Start Process in One Machine***

**Actors**

User: A user is the administrator of a network.

**Entry Condition**

The user has logged in and is looking at the specifics of one machine.

**Flow of Events**

The user enters a command in the command input field and submits the command for execution. The command is executed in the target machine.

**Exit Condition**

The user starts a desired process in the target machine.

**Exceptions**

Fail to deliver command.

Failed to connect to the machine.

**Nonfunctional Requirements**

The command should be executed on the target machine within 10 seconds.

***Terminate Process in One Machine***

**Actors**

User: A user is the administrator of a network.

**Entry Condition**

The user is looking at a specific machine and its processes. User decided that one process must be terminated.

**Flow of Events**

User clicks on the process he desires to terminate and clicks on the Terminate button for that process. The target machine executes a command to kill the target process.

**Exit Condition**

The process is terminated on the target machine.

**Exceptions**

Fail to deliver command.

Failed to connect to the machine.

**Nonfunctional Requirements**

Process must be terminated within 10 seconds of the button being clicked.

***Get Machine Bandwidth***

**Actors**

User: A user is the administrator of a network.

**Entry Condition**

The user is looking at a specific machine and need to know the download and upload speeds of the machine.

**Flow of Events**

User expands the Speed Test module of the single machine dashboard. The dashboard requests a speed test from the target machine and reports the results to the user.

**Exit Condition**

The user sees the download and upload speeds of the machine he is monitoring.

**Exceptions**

Fail to deliver speed test request.

Failed to connect to the machine.

**Nonfunctional Requirements**

None

***Chat With Machine User***

**Actors**

Admin: The network administrator inspecting the user’s machine.

User: A user is the user of the machine.

**Entry Condition**

The administrator is looking at a specific machine and needs to communicate with the person using the machine.

The user is working on his machine.

**Flow of Events**

Administrator expands the Chat module of the single machine dashboard. The dashboard reveals a chat window for the administrator to type and send a message. The administrator types a message and clicks on the send button. The dashboard sends the message to the machine. The machine opens a chat window with the administrator’s message and capability for the user to enter and send a message for the administrator. The user types and message and sends it to the administrator. The machine sends the message to the dashboard. The dashboard shows the message to the administrator through the chat module.

**Exit Condition**

The user and the administrator have exchanged messages.

**Exceptions**

Fail to deliver message.

Failed to connect to the machine.

**Nonfunctional Requirements**

None

***See Active Ports in the Machine***

**Actors**

User: A user is the administrator of a network.

**Entry Condition**

The administrator is looking at a specific machine and needs to know what ports are being used.

**Flow of Events**

The user expands the Netstats module of the dashboard. The dashboard requests a request for the ports of the machine to be scanned. The machine scans its ports and reports what ports are in use and the protocols being used. The dashboard displays the results of the scan.

**Exit Condition**

The user sees what ports are open in the machine and what protocols are being used.

**Exceptions**

Fail to deliver Netstat request..

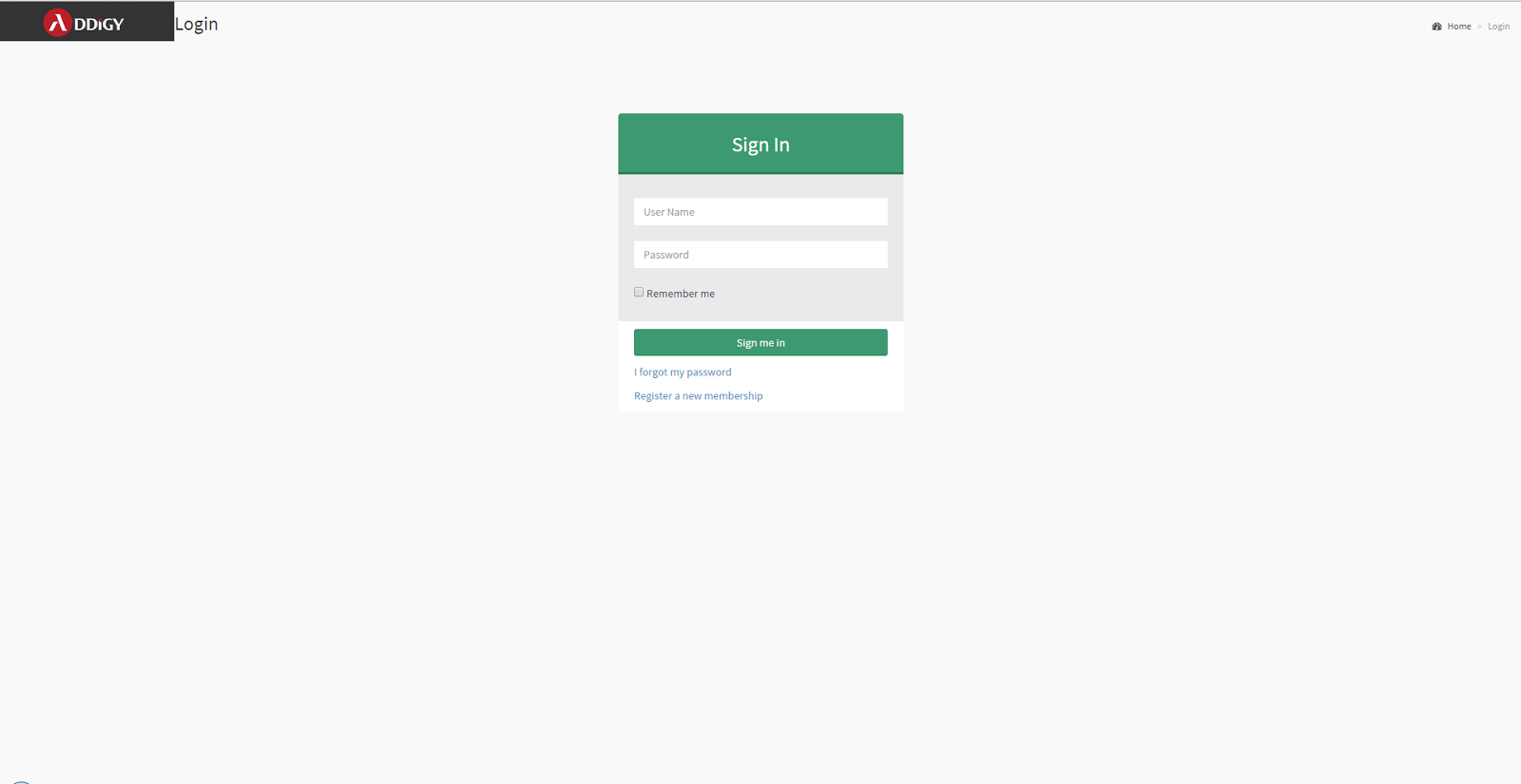
Failed to connect to the machine.

**Nonfunctional Requirements**

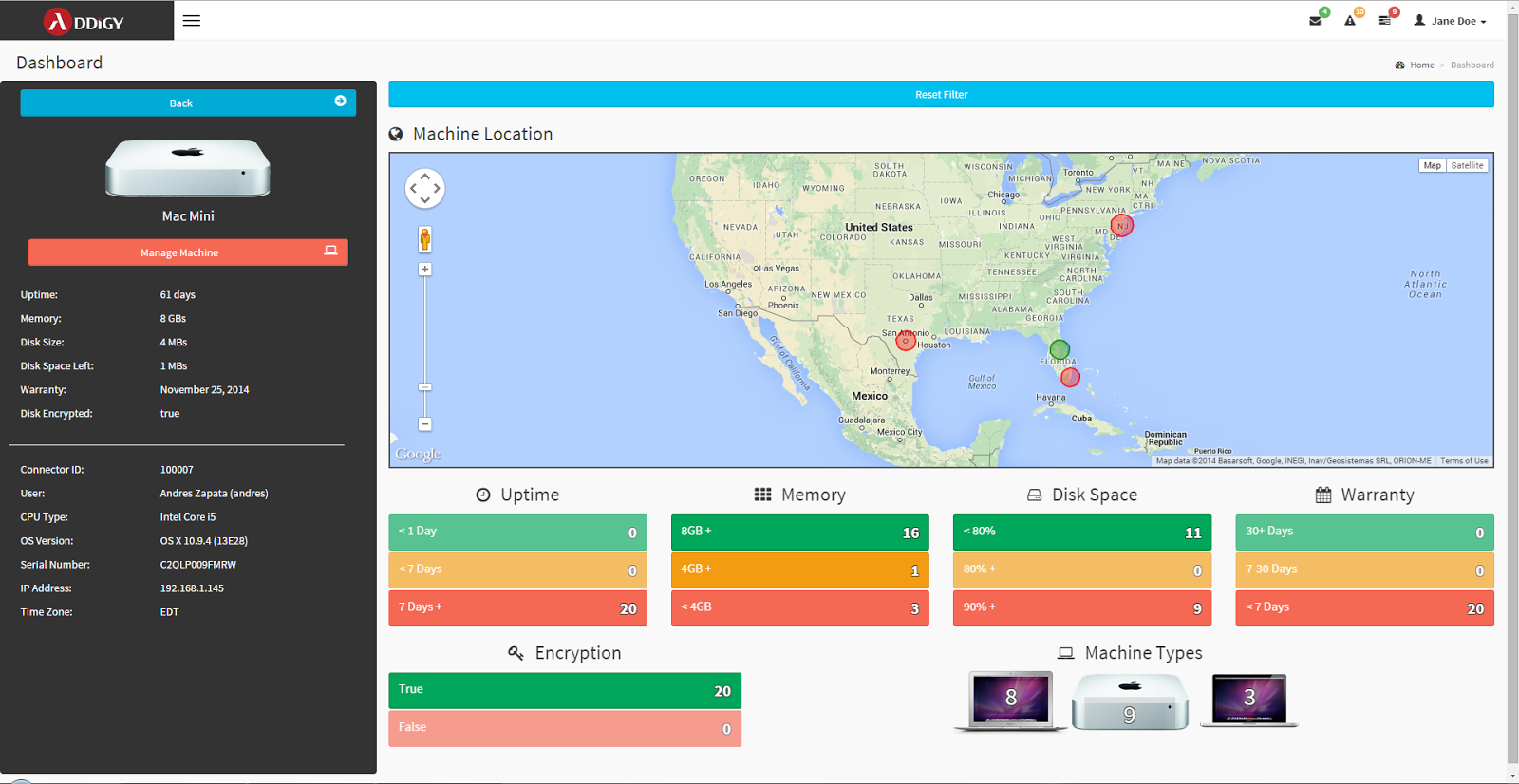
None

## 9.3 Appendix C - User Interface Design

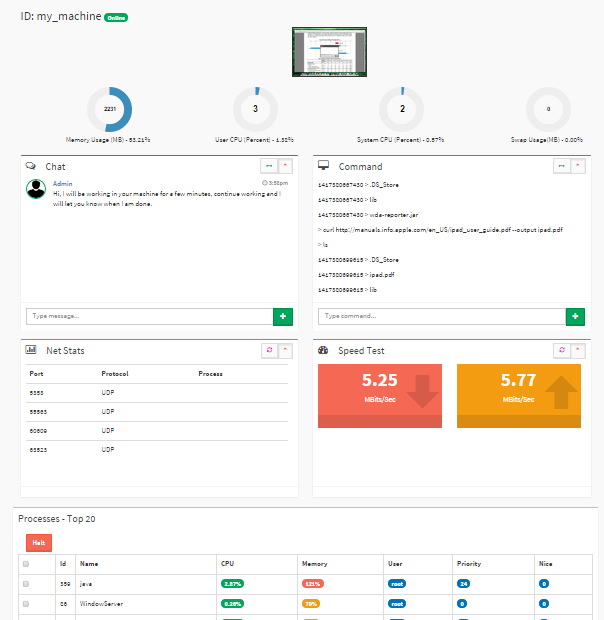
**Login**



**Multiple Machines’ Page**



**Single Machine Page**

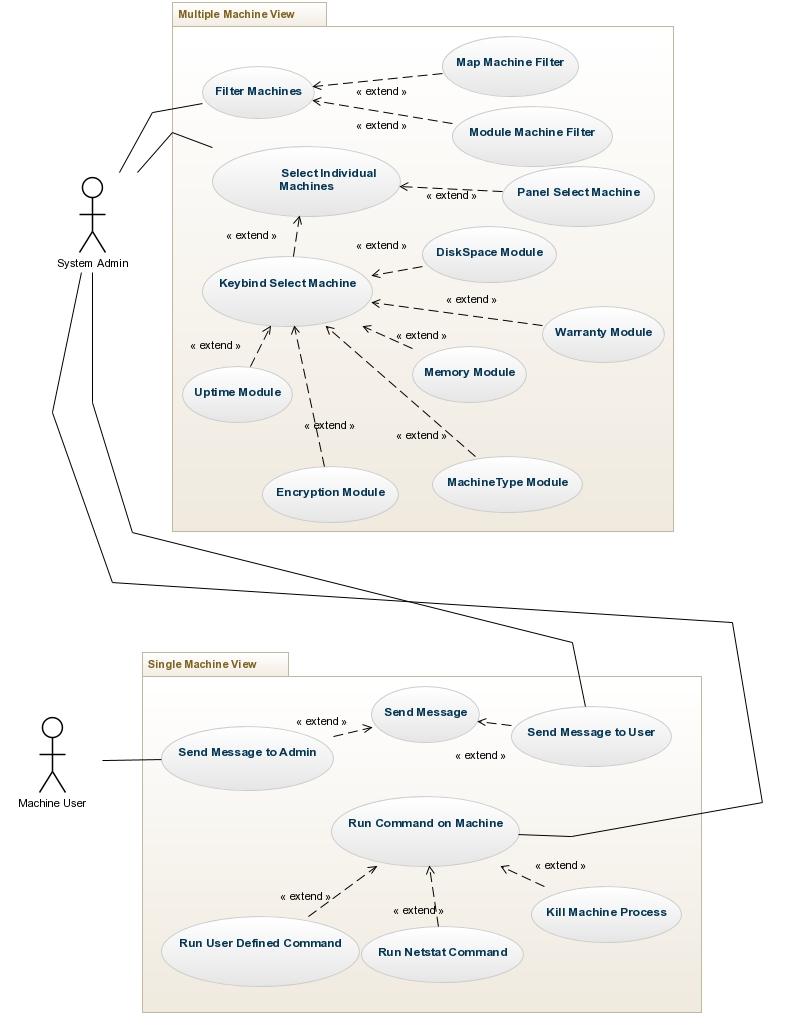


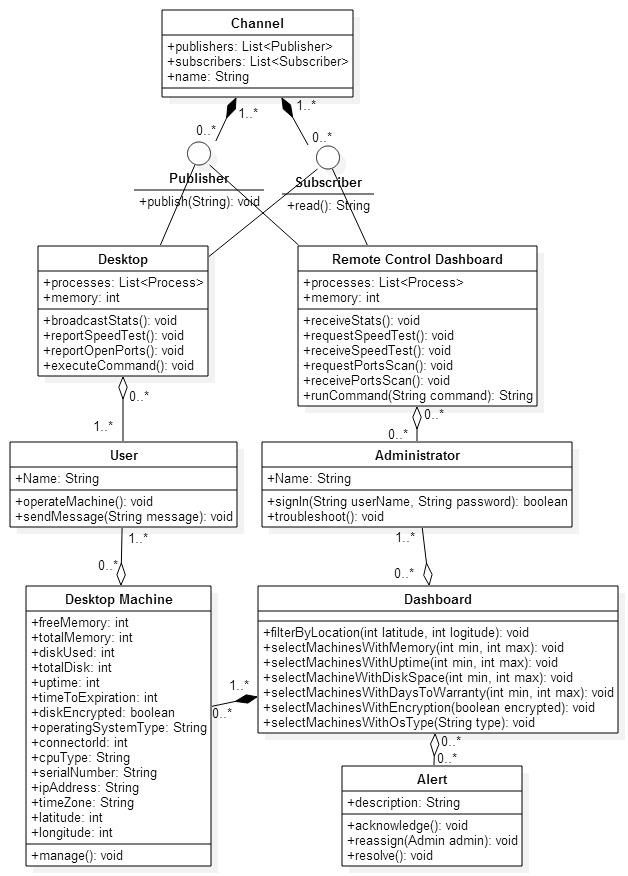
## 

## 

## 

## 9.4 Appendix D - Analysis Models

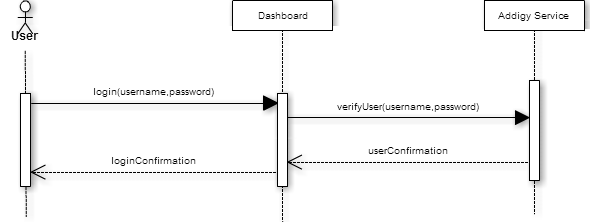




**SEQUENCE DIAGRAMS**

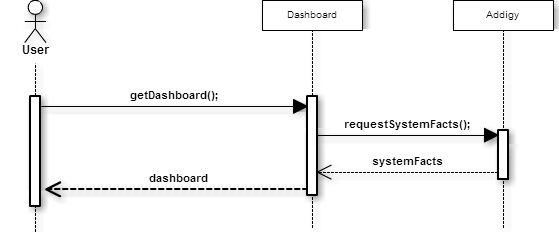
**Logging in**

The user is currently in a logged out state and successfully logs into the application.



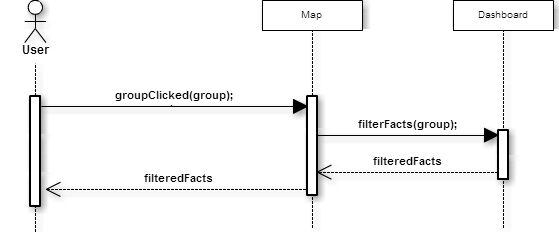
**Examine State of the System**

User is logged in and sees all the computers in the network as well as all the statistics regarding those computers in the web dashboard.



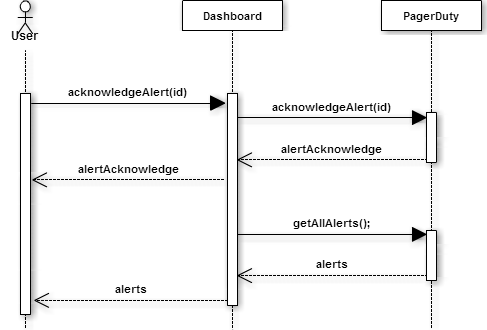
**Examine Computers in One Location**

User has seen the computers in the map and wants to see the stats for one specific computer. User clicks on one location in the map.



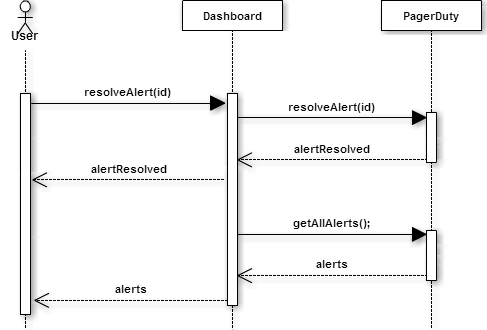
**Acknowledge a Machine Alert**

The user has seen the machine alert on the dashboard but understands it is an error that can be set aside for later.



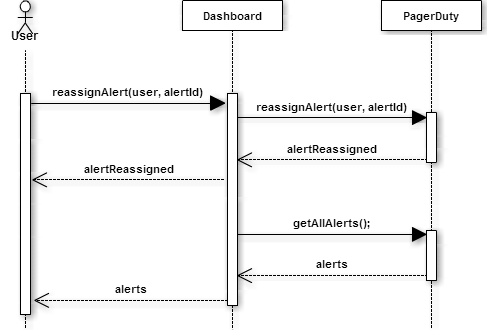
**Resolve a Machine Alert**

The user has seen the machine alert on the dashboard but understands it is an error that was already resolved.



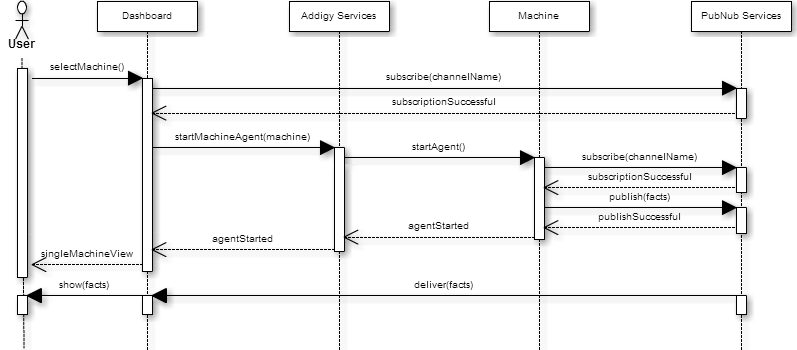
**Re-Assign a Machine Alert**

The user has seen the machine alert on the dashboard and wants to reassign it to someone else.



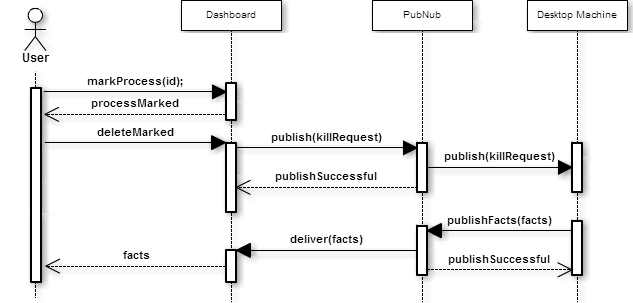
**Examine Single Computer**

User is looking at the stats for the machines in one location and wishes to narrow the stats to a single machine.



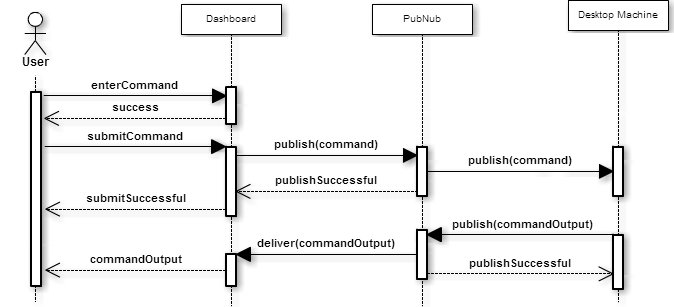
**Kill Process in One Machine**

User is looking at the stats for one specific machine and decided that one processes must be terminated.



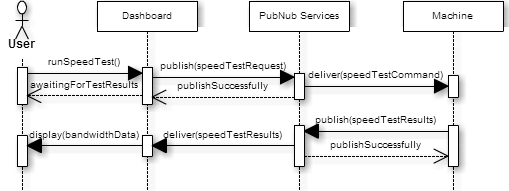
**Start Process in One Machine**

User is looking at the stats for one specific machine and wants to run a specific program in it.



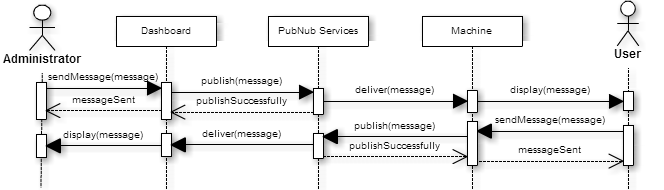
**Run speed test in one machine**

User wants to see the bandwidth speed of one machine and expands the speed test module.



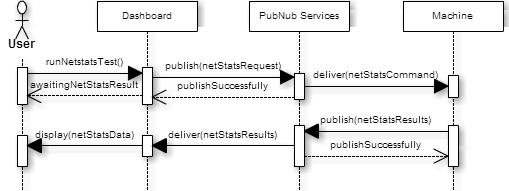
**Chat with User of the Machine**

Administrator communicates with the user of the machine through the chat module.

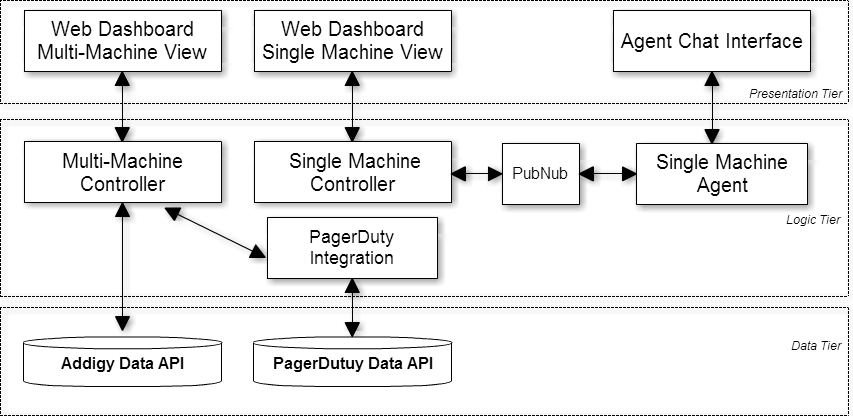


**See Active Ports in the Machine**

User looks at the active ports of the machine through the Netstats module

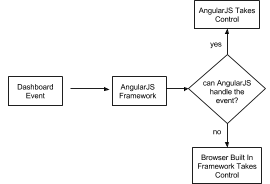


## 9.5 Appendix E - Design Models

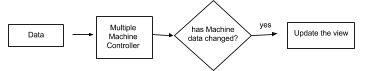


## 9.6 Appendix F - Documented Class interface

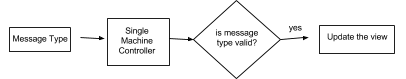
**Multiple Machine View & Single Machine View**



**Multiple Machine Controller**



**Single Machine Controller & PubNub**



## 9.7 Appendix G - Diary of Meeting and Task for Entire Semester

|  |  |
| --- | --- |
|  | Senior Project - Web Addigy Dashboard |
|  |  |
| **Members:** | Javier Carmona, Francisco Marcano |
|  |  |
| **Date** | **September 4th, 2014** |
| **Discussions** | Project was presented and the tecnologies involved were explained. |
| **Announcements** | We will be using a REST Web Service to retrieve information needed for the dashboard. Discussed ideas as to how to divide work, need more information regarding the different modules required for the project. Discussed the technologies that we would like to use for the project such as AngularJS and Bootstrap. Discussed times for future meetings. 2pm on Tuesdays and Thursdays as needed. Specific dates coming soon. Discussed upcoming deadlines and how to get deliverables done. Google docs were setup for the Requirements Document and Feasibility Study & Project Plan. |
|  |  |
|  |  |
| **Date** | **September 9th, 2014** |
| **Discussions** | Project depth was further explored and more information on project models and functionality was discussed |
| **Announcements** | During this meeting we discussed quite a few new functionalities to make the application more attrative. Some things we discussed include: Have the ability to see individual data for one single manchine when clicked, be able to map the location of a a computer using the app and location data from the phone, and using pubnub's API to live stream data for use with single computer process information. During the meeting we also asked for the following documents: list of API calls to Addigy's servers, a static JSON file for development, and an HTML template of the current application available that needs to be redone. |
|  |  |
|  |  |
| **Date** | **September 16th, 2014** |
| **Discussions** | Discussed how to broadcast information regarding a single process. |
| **Announcements** | Will be using PubNub services so that the target machine Published information about running processes to PubNub. Then the dashboard can register to listen for messages broadcasted by the target machine. The Daemon to be running on the target machine is to be written in Java and able to select the Top X amount of processes based on statistics such as Memory Usage or CPU Usage. The dashboard is to use PubNubf's angular API to connect and retrieve this information that is being broadcasted by the target machine. |
|  |  |
|  |  |
| **Date** | **September 30th, 2014** |
| **Discussions** | Discussed new features that would make the project more attractive to potential customers. |
| **Announcements** | We went over what could be done to the main dashboard and how to display information more effectively. Agreed that instead of showing lists of machines with the statuses, it would be better to show the quantities in categories such as [number of machines under guarantee][number of machines with guarantees expiring in 30 days][number of machines without guarantee]. Also discussed ways to make intelligent recomendations based on the cummulative data being collected. In the single machine page, agreed to add features for the administrator to send messages to the machine user through a chat, and run command and see the command's output. |
|  |  |
|  |  |
| **Date** | **October 6th, 2014** |
| **Discussions** | Most relevant features and compatibility |
| **Announcements** | Decided that in the single machine page, the most urgent feature would be being able to see a screenshot of the machine being monitored. Following features would be executing a command remotely and a chat interface between the admin and client. We discussed compatibility issues for the commands being executed to get system statistics, specially the top command and the flags being passed to it in linux vs mac. Also discussed what could be done for windows machines. |
|  |  |
|  |  |
| **Date** | **October 21st, 2014** |
| **Discussions** | Most relevant features and design documents |
|  | We discussed the need to get most of the basic features in the Multiple machine view done ASAP in order to move on to some mroe novel features that will have a little more complexity. In the single machine view, discussed the chat feature and the "run command" feature that will be used for online machines in the network. Discussed some different techniques to get continous diployment and testing incorporated directly into the server running the main code (program called Jenkins). Discussed the timeline for the next couple of features. |
|  |  |
|  |  |
| **Date** | **October 28th, 2014** |
| **Discussions** | Expand features |
| **Announcements** | We discussed some new features that could make the Multiple Machine view really pop. The feature was grouping machines into groups and color code them based on errors in the system. Disussed incorporating a netstat module into the Single machine view to quickly see any open socket connections on the machine |
|  |  |
|  |  |
| **Date** | **November 11th, 2014** |
| **Discussions** | New Feature |
| **Announcements** | Went over the current progress and discussed new features that could be implemented. Among these were to be able to show alerts on the multiple machine dashboard for machines that needed urgent attention. In addition, discussed the idea of having the single machines dashboard run a bandwidth speed test on the computer and report that information. |
|  |  |
|  |  |
| **Date** | **December 1st, 2014** |
| **Discussions** | Met up to discuss Alerting and Speedtest |
| **Announcements** | Met together with our mentor and discusses the posibility of creating an alerting feature for the Multiple Machine View. The alerting feature would work off of a service called PagerDuty. In the Single Machine view, discussed the possibility of incorporating a internet connection speed test module in order to record the target machine's internet connection. |
|  |  |
|  |  |
| **Date** | **December 5th, 2014** |
| **Discussions** | Met up to practice showcase presentation ethics |
| **Announcements** | Met in order to practice our showcase presentation ethics and how to handle different situations depending on the audience present. |