Design Document

Performance Monitoring System

Revision 1.0.1

Date of revision 11-11-2016

Prepared By: Suraj Shingade

Revision	**Date**	**Author**	**Verifier**	**Changes**
1.0.1	12th Oct 2016	Suraj S		Initial Doc

Contents

1. Introduction

1.1 Why this document ?

The purpose of this High Level Design (HLD) and Low Level Design Document (LLD) is to add the necessary detail to the current project description to represent a suitable model for coding. This document is

also intended to help detect contradictions prior to coding, and can be used as a reference manual for how the modules interact at a high level

1.2 Scope

The HLD & LLD documentation presents the structure of the system, such as the database architecture, application architecture (layers), application flow (Navigation), and technology architecture. The document uses non-technical to mildly-technical terms which should be understandable to the owner of the system.

1.3 Definitions

• **PMS** – Performance Monitoring System

• **MongoDB** – MongoDB is a free and open-source cross-platform document-oriented database program

• **NodeJS** – Node.js is a very powerful JavaScript-based framework/platform built on Google Chrome's JavaScript V8 Engine. A possible programming language to interface between exposed web services and PMS Database.

• **ReactJS** – The language that will be used for displaying user history and administrative functionality.

• **SailsJS** – a free, open-source implementation of Express Framework middleware which is very nice to have HTTP and socket based web services.

• **ER** – Entity Relation Diagram

• **TBQ –** Traffic-Based Queuing. Limits bandwidth at the IP/port level.

• **PMS Sync :** A possible synchronization module to manage data synchronization between storage usage data.

• **Restful Data Collectors :** Consumes restful web service calls to manage api connection and data download.

1.4 Overview

The Document will:

- present all of the design aspects and define them in detail
- describe the user interface being implemented
- describe the hardware and software interfaces
- describe the performance requirements
- include design features and the architecture of the project
- list and describe the non-functional attributes like:
  - security
  - reliability
  - maintainability
  - portability
  - re-usability
  - application compatibility
  - resource utilization
  - serviceability

2. General Description

2.1 Product Perspective

The PMS (Performance Monitoring System) will be compromised of several different components. Some of these components will be programmed, while others will be implementations of open-source programs. The language implemented will be dictated by its purpose. The handling of batch web services calls are managed by Async library of nodejs. Data can be persisted into mongodb. Later mongodb data will be used to prepare UI and human readable and understandable format using charts or any other tabular formats using Frontend libraries.

2.2 Tools used

1. **Jude**, a Java based UML design program, is used to generate all of the diagrams used in analysis and design phases of the project.

2. The project will have a NoSql **MongoDB** backend that is JSON based. The actual software used is MongoDB.

3. Interfacing with the database to display information on the user's web browser will be done using **SailsJS** and **ReactJs**. It can connect to the database and parse it into viewable HTML code.

4. Automated interfacing with the database behind the scenes will be SailsJS (A Powerful Express Framework Wrapper).

7. Storage Server Load Balance Reporter.

8. **TBQ** is a throttling mechanism based on classes.

9. **RedHat Fedora** or **UBuntu** is the development platform.

2.3 General Constraints

The Performance Monitoring System must be user friendly and as automated as possible. User should not be required to do anything besides going thru the authentication process, and users should not be required to know any of the workings. After logging in, that user then has the ability to change settings and check real time performance statistics on his screen.

2.4 Assumptions

This project is based on the idea of a Monitoring Performance Data of storages, and the goal is to make this idea a reality using Software Engineering practices. In doing so, documents are created, and it is assumed that design flaws will be found early on. It is also assumed that all aspects of this project have the ability to work together in the way the designer is expecting.

Another assumption is that the current intended documentation will suffice to make this project count towards the Software Engineering Subtract. There is also an assumption that none of the work or hardware will be stolen or sabotaged. The final assumption is that a demonstration and presentation will be possible at the end of the development phase.

2.5. Special Design aspects

One special design aspect is the understanding that without significant modification, this System will only work on performance data of its storage servers. This means system will manage its own traffic and then also take care about storage system traffic too. Based on storage system load PMS will decide about time interval of consuming web service calls.

3. Design Details

3.1. Main Design Features

The main design features include five major parts:

The architecture, The user interface Design, External interface, The database, Process relation, and Automation based on storage system load.

In order to make these designs easier to understand, the design has been illustrated in attached

diagrams (ER, Component).

Screen Shot Breakdown:

User

• #login – User login screen.

• #index – User would see if they were logged in and sticking to dashboard.

• #home - Tabular format data where user can check storage server usage data.

• #configuration – User level configuration management configurable features like

1. Manage Storage Server List.
2. Manage Performance Parameters.
3. Interval for creating Data Collector Sets.

• #history - User can check historical data.

Administrator

• #login – Administrative login screen.

• #index – Administrative would see if they were logged in and sticking to dashboard.

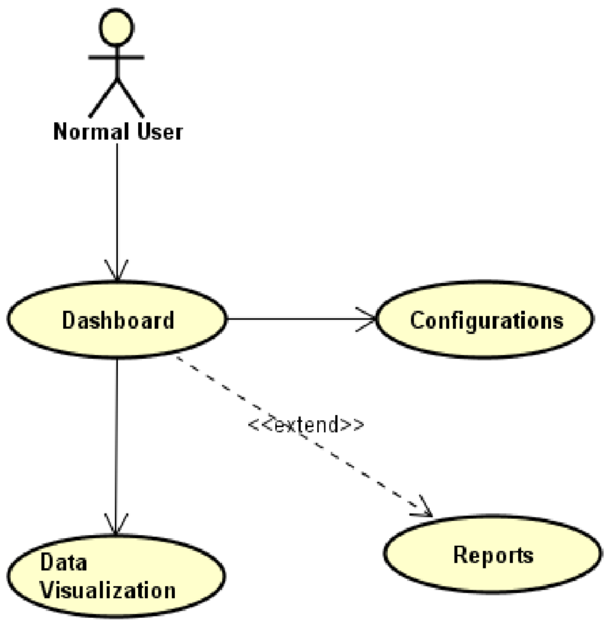
• #home - Tabular format data where user can check storage server usage data.

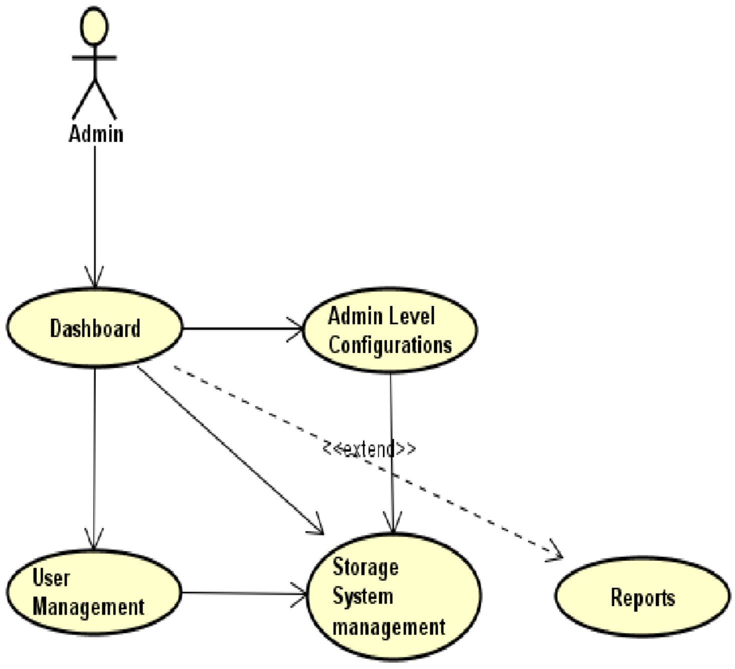
• #configuration – User level configuration management configurable features like

1. Manage Storage Server List.
2. Manage Performance Parameters.
3. Interval for creating Data Collector Sets.
4. Manage Users.
5. Autocheck Storage System Load.
6. PMS System Load.

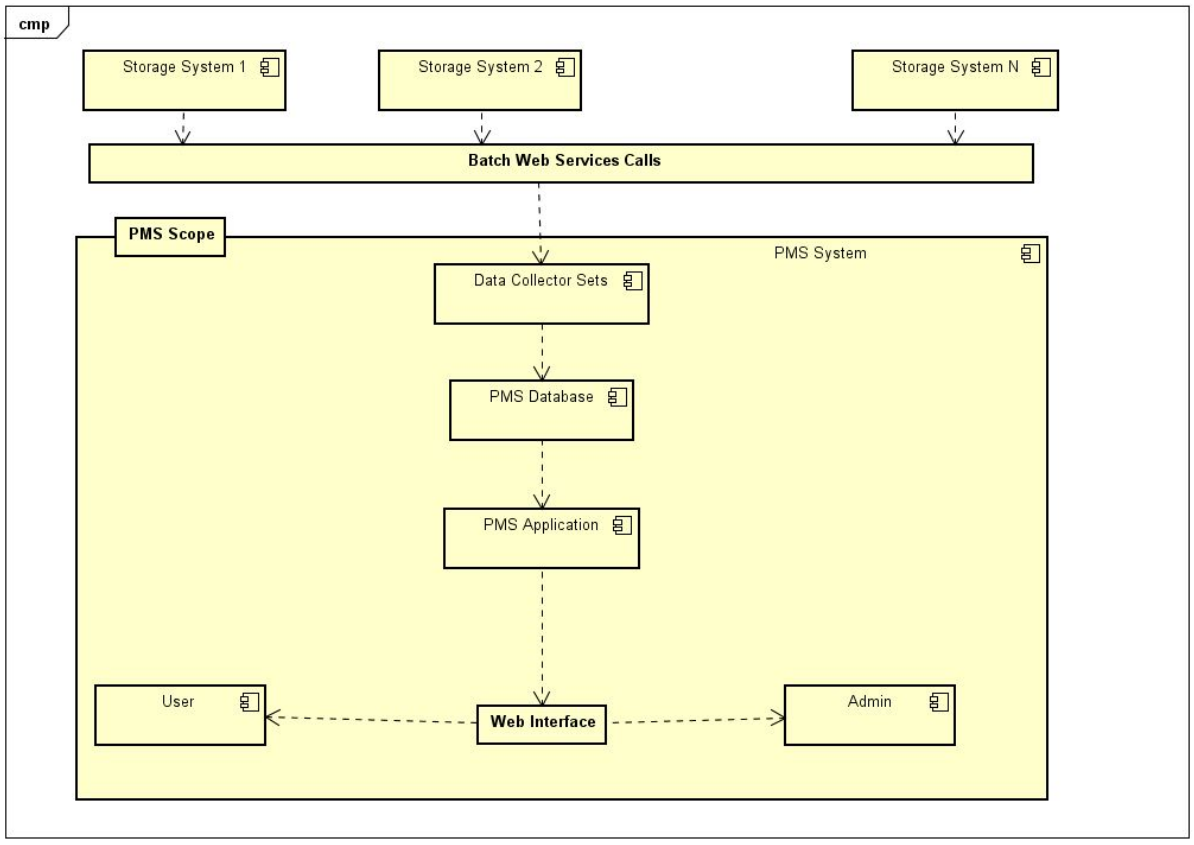
• #users - Admin can check logged in users.

3.2. Application Architecture





Component Diagram



3.3. Technology Architecture

3.3.1. Web Application Architecture

The front end of the program is a web application. Functionality will vary based user privileges if a user is logged in. Normal users are required to log in, and can view stats according to their personal configuration. Administrators will have access administrative abilities based on permissions given to them.

3.3.2. Presentation Layer

Information will include the Storage Usage including I/O per second, throughput, latency etc, all information persisting to MongoDB using batch job. Visualization screens will have access to this information, and the ability to change system settings.

3.3.3. Data Access Layer

The database will be accessible to all users, administrators, and automated services.

A login will determine what parts of the database can be accessed and changed.

3.3.4. Tools Used

See section 2.2 and 3.30 for tools used in the design of this project.

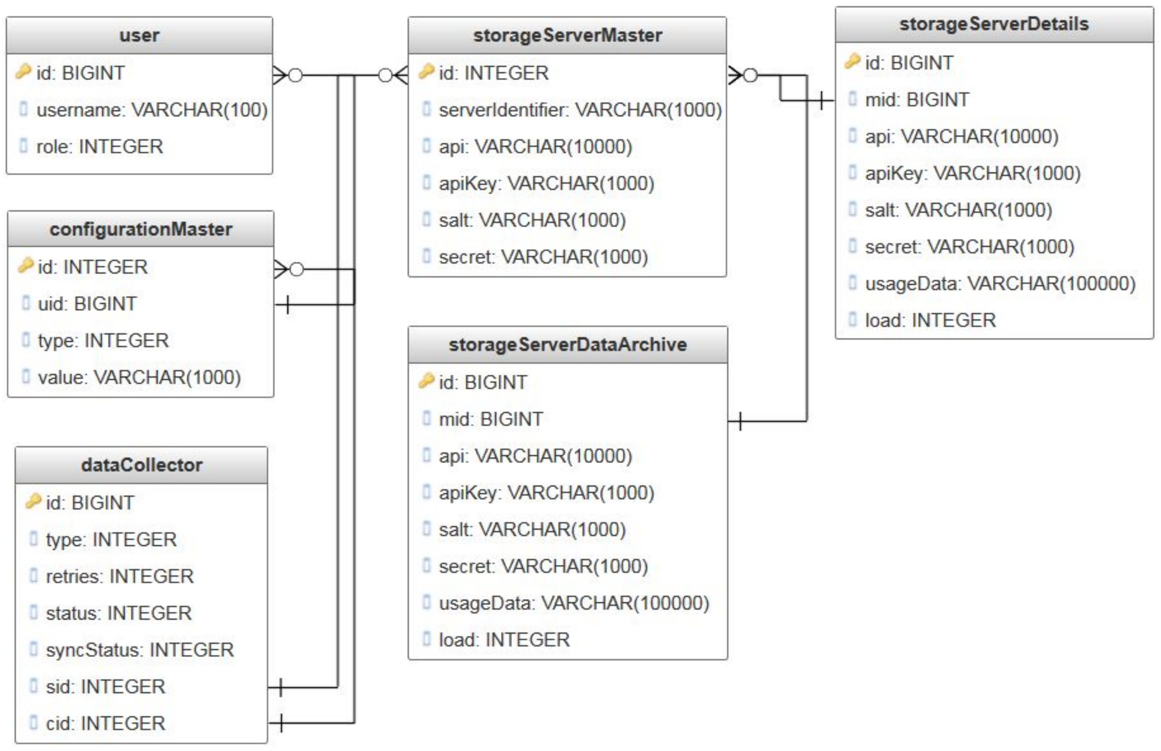
3.4. Standards

Inputs – entered through data collectors

Security – username and password are required for access to the system.

Quality – by keeping the interface simple and direct, quality should be kept at a maximum.

3.5. Database design



3.6. Files

This product will not use a large number of files. Data Collectors uses a file to maintain sync status to MongoDB association. SailsJs uses Jade pages. A file will be used to store all of the usernames and passwords and all other attributes specified for those users. This file will be accessed at login. It can be modified by the administrator at any time. This file will be accessed and modified by all users with proper permission.

3.7. User Interface

The user interface is a very simple plain layout with little to brief graphics. It will display information very clearly for the user and will primarily output information to the user through HTML pages. Administrative screens are used mainly for input through text fields in HTML pages. Screen shots have been provided to demonstrate the user and administrative interface.

3.8. Reports

The reports will display the user's average usage up to the last time the system calculated it and their history. Reports can also be created by an administrator of any storage systems traffic.

3.9. Error Handling

Should errors be encountered, an explanation will be displayed as to what went wrong. An error will be defined as anything that falls outside the normal and intended usage.

3.10. Interfaces

There are three main interfaces for this project.

First, the visualization interface, which consists of the data collected by batch job and all traffic that goes through the storage servers.

Second, the interface for the automated services which employed to collect data from storage systems.

Third, the user and administrative interface sailsjs.

3.11. Help

Help will come in the form of all the documentation created prior to coding, which explain the intended uses. Should time allow, detailed instructions will be written on how to create and implement the system with the intentions of publishing as an powerful solution.

3.12. Performance

Performance is going to be very important for this project. For everything to run smoothly for this project, the data collectors will have to be able to update data on the database and refresh the database without user interference. This is likely to be the most processor intensive aspect of the project. The PMS will also need to supply requested pages to the users at a reasonable speed. The database server will need to keep up with all database requests and transactions.

3.13. Security

Because security is not the prime focus of this project, only the minimal aspects of security will be implemented. A username and password will be required to log into an

administrative interface and database. For now, all data will be sent in plain text.

Verification of user to IP or MAC address is also outside the scope of this project. For now, there will also be no log of failed attempts of an administrator logging in.

3.14. Reliability

A redundant database server will be implemented so that if the main database server stops

responding, the PMS will automatically start using the other server. The mechanism

used for syncing these two databases has not yet been fully established. Likely subsequent

solutions include:

• Each interface to updating to both servers.

• An archive field being implemented and the redundant server constantly searching the tables for new data.

• A full periodic backup for the entire database.

• A trigger being used on the main database where all data is automatically copied to the secondary database.

3.15. Maintainability

Very little maintenance should be required for this setup. An initial configuration will be

the only system required interaction after system is put together. The only other user

maintenance would be any changes to settings after setup, and any specified special cases

where configuration need to be changed. Physical maintenance on the system's parts may be required, and would result in temporary loss of data or Internet. Upgrades

of hardware and software should have little effect on this project, but may result in downtime.

3.16. Portability

This system should have the ability that, once it is together, the entire system should be able to be physically moved to any location. Code and program portability should be possible between any environment. For everything to work properly, all components should be compiled from source.

3.17. Re-usability

The code written and the components used should have the ability to be reused with no problems. Should time allow, and detailed instructions are written on how to create this project, everything will be completely reusable to anyone.

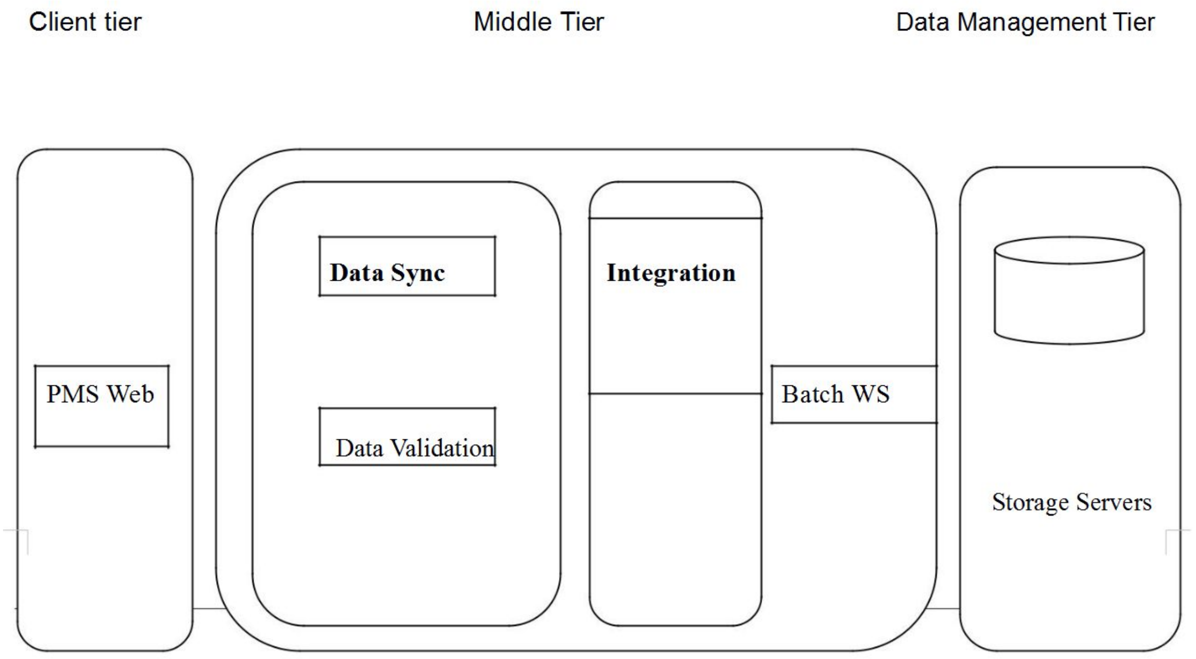
3.18. Application compatibility

The different components for this project will be using NodeJS as an interface between them. Each component will have its own task to perform, and it is the job of the NodeJS code to ensure proper transfer of information.

3.19. Resource utilization

When any task is performed, it will likely use all the processing power available until that function is finished. The gateways are likely to use their processors the hardest when they are refreshing the data, but this will depend largely on how many storage systems are being updates.

3.20. Design and Technology Stack Overview



The Business Process workflow used in our system can be broken down into three tiers:-

1) The Client Tier with the Visualization and the data that is going to interchanged
2) The middle tier which comprises of service mix and the Server with the business logic
3) The Back end tier with the database.

Technology Stack Overview:

NodeJS : Backend Language

SailsJS: Middle-ware, An efficient implementation of express framework.

Async: NodeJS library for bulk web services calls..

ReactJS: Component based front-end technology

JSON: Data interchange format

MongoDB: Non Volatile data storage solution.

7z or lzma : Data Compression algorithm.

Twitter Bootstrap : Web Framework.

Moment.JS : JS Framework

The Google Visualization API / D3.js / Raphael.js/ Dygraphs.js : Visualization Framework

----