**SERIS**

Solar Energy Research Institute Singapore



Cloud Based Real-time Analytical Monitoring of Photovoltaic Systems and Weather Parameters Project

High Level Design (HLD)

|  |  |
| --- | --- |
| Filing Reference | SE25PT7SERIS/TECH/DESIGN/HLD/HLD.doc |
| Document Title | High Level Design (HLD) |
| Version | 1.6 |
| Author | Treza Bawm Win |
| Date Created | 14/03/2018 |

|  |  |  |
| --- | --- | --- |
| **Approved by:** | | |
| Name | Designation | Date |
|  |  |  |
| **Authorized by:** | | |
| Name | Designation | Date |
|  |  |  |

**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Revision** | **Date** | **Author** | **Description** |
| 1.0 | 14/03/2018 | Treza Bawm Win | Initial version |
| 1.1 | 25/03/2018 | Treza Bawm Win | * Product vision * Multitier arch * Technologies stack * Performance |
| 1.2 | 22/04/2018 | Treza Bawm Win | Update the design diagram   * Add AWS Kinesis and S3 * Remove SNS * Add overview use-case diagram |
| 1.3 | 22/04/2018 | Treza Bawm Win | Update as per review comments. |
| 1.4 | 05/05/2018 | Treza Bawm Win | Add AWS Container session for Docker. |
| 1.5 |  | Kaung Myat Bo | Updated Architecture Diagram, Use cases diagram, Web application architecture |
| 1.6 | 03/01/2019 | Kaung Myat Bo | Updated latest architecture diagram, use cases diagram, web application architecture |
| 1.7 | 19/01/2019 | Kaung Myat Bo | Update doc based on ORM-14011918 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **For Internal use** | | **Date** | **Department** |
| Authorized By |  |  |  |
| Released By |  |  |  |

Table of Contents

[1 INTRODUCTION 5](#_Toc535657789)

[1.1 Scope 5](#_Toc535657790)

[1.2 Definitions 5](#_Toc535657791)

[1.3 Overview 6](#_Toc535657792)

[2 General Description 6](#_Toc535657793)

[2.1 Product Perspective 6](#_Toc535657794)

[2.2 Tools used 6](#_Toc535657795)

[2.3 General Constraints 7](#_Toc535657796)

[3 Design Details 7](#_Toc535657797)

[3.1 Main Design Features 7](#_Toc535657798)

[3.2 Application architecture 9](#_Toc535657799)

[3.3 Technology Architecture 10](#_Toc535657800)

[3.3.1 Overall system architecture 10](#_Toc535657801)

[3.3.2 Data volume for sending data 10](#_Toc535657802)

[3.3.3 Web Application Architecture 11](#_Toc535657803)

[3.4 Standard 11](#_Toc535657804)

[3.5 Database design 12](#_Toc535657805)

[3.6 Files 13](#_Toc535657806)

[3.7 User interface 13](#_Toc535657807)

[3.8 Help 13](#_Toc535657808)

[3.9 Performance 13](#_Toc535657809)

[3.10 Reliability 13](#_Toc535657810)

[3.11 Maintainability 14](#_Toc535657811)

[3.12 Portability 14](#_Toc535657812)

[3.13 Reusability 14](#_Toc535657813)

[3.14 Resource utilization 14](#_Toc535657814)

# INTRODUCTION

This document describes how the modules interact with each other in a high-level perspective. It also clearly states the detail necessary design model and architecture design prior to the actual implementation or coding. The architecture proposed in this document is designed to address the demand for the performance as well as the scalability and high availability.

## **Scope**

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

## Definitions

* IoT – Internet of thing
* AWS IoT Core - is a managed cloud platform that lets connected devices easily and securely interact with cloud applications and other devices.
* AWS S3 – is Amazon object storage service to store and retrieve any amount of data from anywhere.
* Elasticsearch - is a distributed, RESTful search and analytics engine
* AWS Elasticsearch Service - is a fully managed service that delivers Elasticsearch’s easy-to-use APIs and real-time analytics capabilities alongside the availability, scalability, and security that production workloads require.
* EC2 - Amazon Elastic Compute Cloud (Amazon EC2)
* Tomcat – a free, open-source implementation of Java Servlet and Java Server Pages technologies developed under the Jakarta project at the Apache Software Foundation.
* Apache - An open source Web server.
* ER – Entity Relation Diagram
* JAVA – A programming language to interface between Client side and MySQL.
* Spring - is an application **framework** and inversion of control container for the Java platform.
* Spring boot - makes it easy to create stand-alone, production-grade Spring based Applications that can be "just run"
* JSON - (JavaScript Object Notation) is a lightweight data-interchange format.
* Structured data – is a common JSON format for easier, accelerated analysis and business value.
* React – A JavaScript library for building user interfaces as components.
* MySQL - is a database management system.
* Firewall - Functionality that can allow or block certain ports and addresses.

## Overview

The HLD 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
  + reusability
  + application compatibility
  + resource utilization
  + serviceability

# General Description

## Product Perspective

As the SERIS is a real-time cloud based analytical monitoring system, the system is required to be designed with the vision of handling high volume real-time data transaction per second. The AWS cloud services have been chosen as the pivotal integrator of the project components as it comes with options to support systems that are highly scalable and available in nature.

On the other hands, the web-based system will be developed which will allow multiple concurrent different types of users to access the system in order to see live analytical data in a very good graphical user interface (GUI).

## Tools used

* Enterprise Architect is used to generate all of the diagrams used in requirement and design phases of the project
* The project will have a relational database backend that is SQL based, MySQL, and a NoSQL database, Elasticsearch.
* Interfacing with the database to client side browsers will be done using Spring Boot. It can connect to the database, parse data into JSON and return.
* Client side user interface will be done by using React
* Tomcat compiles JSP pages into servlets to be displayed through Apache.
* Apache - An open source web server that will display requested pages.
* Automated interfacing with the database behind the scenes will be JAVA JPA-Hibernate.
* One EC2 instance to host Spring Boot Application.
* AWS IoT service to receive data from IoT devices and pass data as a argument into AWS lambda function.
* AWS S3 to be used for uploading a batch of slow data.
* Elasticsearch Service to stored sensor data for analysis.
* Spring tool suite – for web application development.
* Postman – for API unit testing.

## General Constraints

* Raw data should be transformed based on a configuration instead of hard coded inside the program.
* After logging in, user can view either real time analytical graph or history data on graph.
* After logging in, admin user can do setting up stations, users and station configurations.
* Web application must be user friendly.
* Respective sensor data should be sent to browsers for respective logged in users at real time.

# Design Details

## 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. In order to make these designs easier to understand, the design has been illustrated in attached diagrams (ER, Use Case, and Screen Shots).

Screens Breakdown:

All users

* Login
* Forgot password
* Reset password

System users

* Dashboard – To display station locations on map and real-time fast-data in rich interface. Summary of health statistics will also be displayed.
* Health – To display real-time stations health status and to send the error reports.
* Download – To submit a download request for slow-data and to download the generated zip file.
* History record – To display the slow-data in graph.

System admins

* Stations – After clicking on Stations, this allows the user to search the stations information by name.
* Station – After clicking on either Add or station row in Stations screen, this allows the user to create a new station or modify the existing station information.
* Users – After clicking on Users, this allows the user to search the users information by user name.
* User – After click on either Add or user row in Users screen, this allows the user to create a new user or modify the existing user information.
* Error reports – After click on Error reports, this allows the user to search the error reports by either reference no or station name. User can also close an error report in this screen.

## Application architecture



Figure 1 Application architecture 1

## Technology Architecture

### Overall system architecture

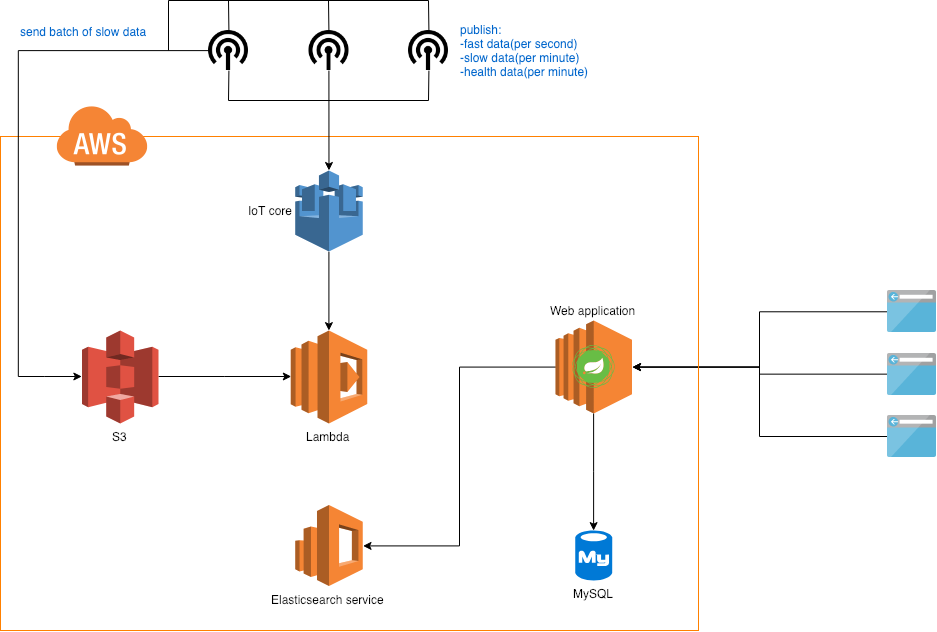


Figure 2 Over system architecture

Overall architecture can be divided into two major parts:

* Receiving sensor data in JSON format
* Web application for visualization

### Data volume of sending data

Average data size for each record – 250 bytes

Fast data volume per day for 1 station = 60 x 60 x 24 = 86400 (no. of records) = 22 megabyte (approximate)

Slow data volume per day for 1 station = 60 x 24 = 1440 (no. of records) = 0.4 megabyte (approximate)

Health data volume per day for 1 station = 60 x 24 = 1440 (no. of records) = 0.4 megabyte (approximate)

Total data **volume per day for 1 station** = 22.8 megabyte (approximate)

Total data **volume per year for 1 station** = 8.322 gigabyte (approximate)

### Web Application Architecture

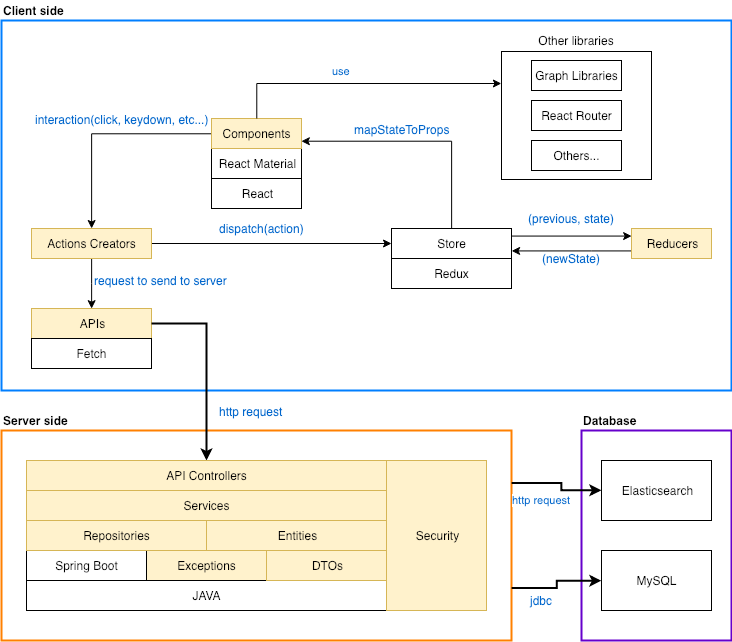


Figure 3 Web Application Architecture

This application will be a multitier architecture as described below:

* **Presentation Layer**: will be implemented with React.
* **Business Layer**: Java back-end application which has business process handlings and expose as REST web services. Spring Boot framework will be used for back-end.
* **Data access Layer**: will be implemented using JPA-Hibernate and http request/response.

## Standard

Database – relational and NoSQL

Inputs

* Sensor data will be sent by stations in JSON string format.
* The rest will be entered through text field and stored in database.

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

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

## Database design

Both SQL and NoSQL databases will be used for this project. All the sensor data will be stored as documents inside Elasticsearch(NoSQL database) for analysis. The rest will be stored inside SQL database.

Sensor fast-data will be indexed day by day basic. The last three indices will be kept in the database and earlier indices will be deleted on daily basic to prevent data duplications, leggy performance and to reduce physical hard disk space usage.

Sensor slow-data indices and health-data indices will also be created on yearly basic. Check the following diagram for overall storage structure of all types of sensor data.

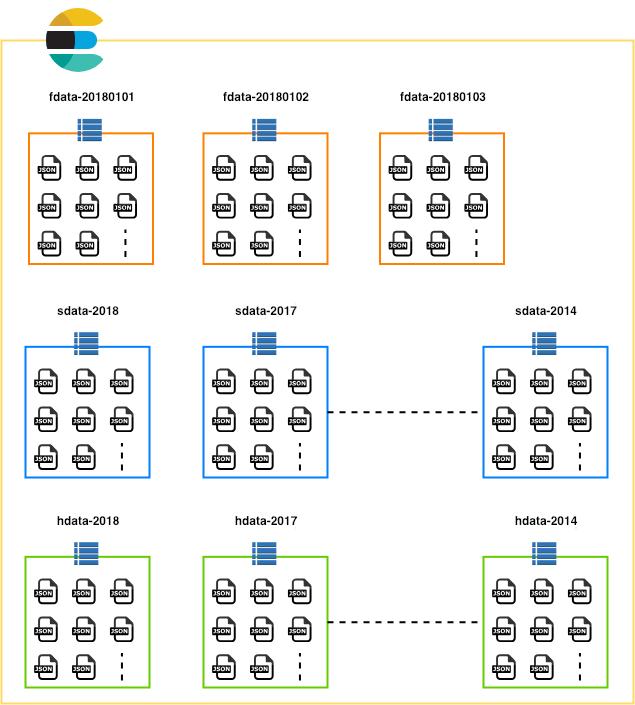


Figure 4 Overview document indexing diagram

Both MySQL and Elasticsearch databases will need to have back up instances. In the future, Elasticsearch may also need another node(server) for clustering.

## Files

This system will use a large number of files.

* HTML files for email templates.
* CSV and zip files will be generated for download data.
* JAVA files to develop backend.
* JavaScript files for frontend development.
* JSON files for reference data.

## User interface

AWS services - Admin user must use AWS Management console.

Web Application - Administrative screens are used mainly for input through text fields in HTML page. Screens for system user are still needed to discuss.

## 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 Open Source solution.

## Performance

The system is required to collect data from all IoT devices tagged to the station. As of now each station is equipped with two stations. For the first implementation phase, the system needs to process data for a total of 10 stations. However, the number of stations to be serviced by the application is set to increase in the future. Therefore, it is necessary to design the system to perform efficiently taking into consideration the landscape of devices and stations to be served by the system in the future.

## Reliability

The system is required to collect data from all IoT devices tagged to the station. As of now each station is equipped with two stations. For the first implementation phase, the system needs to process data for a total of 5 stations. However, the number of stations to be serviced by the application is set to increase in the future.  Therefore, it is necessary to design the system to perform reliably taking into consideration the landscape of devices and stations to be served by the system in the future. For a more reliable implementation, we must address the issues around loss of data packets sent from the sensors until they have been successfully received and recorded with in the system.

## Maintainability

AWS services – are already managed by AWS.

Web Application – Very little maintenance should be required for this Web Application.

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

## Portability

Because of relying on AWS services, they whole system cannot be moved to other location. Web Application will be developed by JAVA, so it can be run on any OS. The same goes for Logstash, can be run on any OS.

## Reusability

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.

## Resource utilization

When the transforming is performed, it will likely use all the processing power available in Logstash EC2 instance until that transforming is finished. But resource consumption will depend largely on how many stations are being implemented to the system.