System Requirements Specification

Senslify

By: Christen Ford

V 1.0.0

Table of Contents

[Introduction 3](#_Toc75263147)

[Terminology 4](#_Toc75263148)

[Use Cases 5](#_Toc75263149)

[Use Case Diagram 6](#_Toc75263150)

[Use Case Descriptions 7](#_Toc75263151)

[System Requirements 15](#_Toc75263152)

[Functional Requirements 15](#_Toc75263153)

[Non-Functional Requirements 18](#_Toc75263154)

[Change Management Process 19](#_Toc75263155)

[Communications Management Plan 20](#_Toc75263156)

# Introduction

This document details the requirements of the Senslify sensor web visualization software built in collaboration with Devendra Waikuls’ Master’s thesis as part of the Cleveland State University (CSU) and Case Western Reserve University (CWRU) IOT Collaborative. This project took place during Summer 2019 and produced an asynchronous, extensible web platform for visualizing sensor data in real-time. Note that this software was developed using an iterative, agile technique. Instead of all of the requirements being gathered ahead of time as is common in agile, this set of requirements evolved over time alongside the software.

This document is organized into a set of functional system requirements and non-functional system requirements that were derived using a variety of requirements elicitation techniques including formal/informal interviews, consultation, the Delphi technique and intensive on-site client participation in the design and implementation of the software. Prior to starting work on this project, I met with both Devendra and his advisor Dr. Phillip Feng alongside my advisor, Dr. Haodong Wang several times at CWRU so I could be acquainted with the goals of Devendras’ thesis. According to Devendra, his thesis and this project were intended to be utilized as part of a startup company with the goal of providing in-house power usage and monitoring via a deployment of multiple sensor units across a location. The web visualization component would thus allow the appropriate resources to view this sensor data in real-time and adjust their power usage accordingly.

Requirements specified in the following sections have been prioritized utilizing the MoSCoW technique and are written in a standardized, industry-accepted format.

Devendra and I would like to thank the IOT Collaborative for financially sponsoring this project. We would also like to thank Case Western Reserve University and Cleveland State University for allowing us to work on this project together. Lastly, we would like to thank Dr. Phillip Feng and Dr. Haodang Wang for supervising this project and providing help and advice as needed.

# Terminology

The following list details the terminology that is utilized in the Formal Requirements and Informal Requirements section. Please reference this list if you need any clarification as to the terms present in the requirements specification.

* Adapter: A software interface that allows two or more separate APIs to communicate with each other.
* Application Programming Interface (API): A programming interface that is designed to allow working with specific hardware or software.
* Bluetooth Low Energy (BLE): The low-power Bluetooth specification utilized in power constrained environments such as wireless sensors.
* Bootstrap: An extensive CSS3 library designed by Twitter that allows developers to focus more on the functionality of websites instead of their style.
* Document-Oriented Database: A database that groups information in terms of documents. A less formal data aggregation scheme than what is present in relational databases, document-oriented database have no specified structure or schema. Related documents are stored in collections but each document within a collection may have a different number and types of data fields.
* Channel: A server-side mechanism for tracking what clients are listening to updates from each sensor.
* Gateway: A wireless device that acts as a relay between a group of sensors and the web server. The gateway is intended to serve as an aggregator for sensor readings. Note that gateways did not make it into the project by the deadline and instead, sensor upload their data directly to the web server using a REST API.
* Group Identifier: A 8-bit numeric integer that when used in combination with a Sensor Identifier uniquely identifies a sensor.
* ISO-8601: An ISO specification that details explicit standards for working with datetime formats.
* MongoDB: An open-source collection-oriented database that utilizes the BSON data interchange specification for storing data. Interacting with the database requires writing queries in the Javascript scripting language.
* Reading Type Identifier: A 8-bit numeric integer that identifies a specific type of sensor reading such as a photovoltaic reading, temperature reading, power-draw reading, etc…
* Relational Database: A database that groups information in terms of structured tables. Relational databases utilize an implementation of SQL to read, write, and modify data. Data is stored in a well-defined schema and organized into a group of related tables.
* RFC-6455: The IETF Request for Comments document that details the specification for WebSockets.
* Room: An alias for Channel.
* Sensor: Any low-power wireless device that periodically gathers information on it’s environment and uploads it to a remote data sink utilizing a well-known data communication protocol.
* Sensor Identifier: A 16-bit numeric integer that when used in combination with a Group Identifier uniquely identifies a sensor.
* Web Browser: Any application that implements the HTTP protocol as well as the CSS styling language and the Javascript scripting language. An application that allows the user to browse the Internet.

# Use Cases

This section details the use case diagram as well as the associated use case descriptions that go along with each of the use cases. Use cases are used to drive the design of a system and are meant to encapsulate the fundamental features of the system. Note that a use case diagram generally represents features at a very high level. It may not be clear exactly clear which feature is represented by which use case, hence why this document also includes use case descriptions in this section.

## Use Case Diagram



Figure 1: Use Case Diagram

## Use Case Descriptions

This section contains the use case descriptions for each use case above. These descriptions list all participants of the use case, any pre/post-conditions on the use cases, any alternative paths that may happen during a use case, and the use case descriptions. Use case descriptions are written in the standard “The user does…” and “The system responds by…” alternating fashion. Pre-conditions list any assumptions that must be true for the use case to apply. Post-conditions list any assumptions regarding the state of the system after the use case has been carried out. Use case descriptions serve a similar purpose to user stories in that they are meant to elucidate requirements that are may be more subtle and less likely to be discovered by traditional requirements engineering techniques.

|  |  |
| --- | --- |
| Use Case ID | UC-001 |
| Use Case Name | View Groups |
| Participants | User, Web Server, No-SQL Database |
| Pre-Conditions | 1. The web server is running. 2. The No-SQL database contains at least one group. |
| Post-Conditions | 1. The user can view groups stored in the No-SQL database. |
| Normal Path | 1. The user visits the system home page in their web browser. 2. The system retrieves the list of groups from the No-SQL database, renders the home page document, and sends it back to the users web browser. 3. The users web browser renders the home page. |
| Alternative Paths | N/A |
| Description | This use case allows the user to view sensors groups. |

|  |  |
| --- | --- |
| Use Case ID | UC-002 |
| Use Case Name | View Sensors |
| Participants | User, Web Server, No-SQL Database |
| Pre-Conditions | 1. The web server is running. 2. The No-SQL database contains at least one sensor related to an existing group. 3. The user supplies an existing group from which to list sensors. |
| Post-Conditions | 1. The user can view sensors stored in the No-SQL database. |
| Normal Path | 1. The user selects a group from the systems home page. 2. The system retrieves the list of sensors associated with the selected group from the No-SQL database, renders the sensor listing page, and sends it back to the users web browser. 3. The users web browser renders the sensor listings page. |
| Alternative Paths | 1. If the user supplies an invalid group, the system should respond by generating an error document and sending that back to the users web browser instead. |
| Description | This use case allows users to view sensors within a group. |

|  |  |
| --- | --- |
| Use Case ID | UC-003 |
| Use Case Name | View Sensor Information |
| Participants | User, Web Server, No-SQL Database |
| Pre-Conditions | 1. The web server is online. 2. The No-SQL database is online. 3. The user supplies a sensor to view. |
| Post-Conditions | 1. The user can view sensor information in real-time. 2. The user can request sensor reading statistics. 3. The user can download sensor readings for a given time period. |
| Normal Path | 1. The user selects a sensor from the systems sensor listings page. 2. The system generates the sensor information page and sends it back to the users web browser. |
| Alternative Paths | 1. If the user specifies an invalid sensor, the system should respond by generating an error document and sending that back to the users web browser instead. |
| Description | This use case allows the user to view sensor information in real-time, request sensors reading statistics, and download sensor readings for a given time period. |

|  |  |
| --- | --- |
| Use Case ID | UC-004 |
| Use Case Name | Subscribe to Sensor Readings |
| Participants | User, Web Server |
| Pre-Conditions | 1. The Web Server is online. 2. There is not an existing web socket connection between the users web browser and the system. |
| Post-Conditions | 1. The user is subscribed to receive all sensor readings uploaded by the sensor they subscribe to. |
| Normal Path | 1. The users web browser sends a join request to the system requesting a web socket be created between the web browser and the system. 2. The system accepts the request, thereby creating the web socket, and caches the web socket for later use. |
| Alternative Paths | 1. If the system cannot establish the web socket, it should send an error message back to the web browser which will then attempt to establish the connection up to three (3) times. If the connection cannot be established, the web browser should redirect the user back to the sensor listings page. |
| Description | This use case allows the user to subscribe to receive sensor readings and should be automatically applied in conjunction with the “View Sensor Information” use case. |

|  |  |
| --- | --- |
| Use Case ID | UC-005 |
| Use Case Name | Unsubscribe from Sensor Readings |
| Participants | User, Web Server |
| Pre-Conditions | 1. The web server is online. 2. There is an existing web socket connection between the users web browser and the system. |
| Post-Conditions | 1. The user is unsubscribed from receiving any sensor readings uploaded by the sensor they unsubscribe from. |
| Normal Path | 1. The user leaves the sensor information page causing the web socket to automatically send a LEAVE request to the system. 2. The system receives the LEAVE request and closes the web socket connection on its end while simultaneously removing the web socket from its cache. 3. The users web browser closes the web socket on its end. |
| Alternative Paths | 1. The user may also trigger a LEAVE request to be generated and sent via the web socket by closing their web browser. |
| Description | This use case allows the user to stop receiving updates for a subscribed sensor. |

|  |  |
| --- | --- |
| Use Case ID | UC-006 |
| Use Case Name | Request Different Sensor Reading Type |
| Participants | User, Web Server |
| Pre-Conditions | 1. The web server is online. 2. The user specifies a reading type. 3. There is an existing web socket connection between the users web browser and the system. |
| Post-Conditions | 1. The user now receives sensor readings corresponding to the reading type they specify for the sensor they are currently subscribed to. |
| Normal Path | 1. The user selects a different reading type through the web browser. 2. The web browser generates a request for the system to change the reading type and sends it over its web socket. 3. The system receives the request, validates it, unpacks it, changes the reading type to what the user specified, then sends a response back to the user. 4. The web socket receives the request, unpacks it, and informs the web browser to update the sensor information page to accommodate for the new reading type. |
| Alternative Paths | 1. If the user supplies an invalid reading type or the request is invalid, the system will send back an error message over its web socket instead. |
| Description | This use case allows the user to request receiving sensor readings for a different sensor reading type. |

|  |  |
| --- | --- |
| Use Case ID | UC-007 |
| Use Case Name | Request Sensor Statistics |
| Participants | User, Web Server, No-SQL Database, RDBMS |
| Pre-Conditions | 1. The web server is online. 2. The No-SQL database is online. 3. The RDBMS is online. 4. The user specifies a sensor, group, reading type, start timestamp and end timestamp. 5. There is an existing web socket connection between the users web browser and the system. |
| Post-Conditions | 1. The user receives sensor reading statistics for the sensor they are currently subscribed to for the time period they specify. |
| Normal Path | 1. The user submits a request for statistics, supplying the group, sensor, reading type, start date and end date to the web browsers web socket. 2. The web browser web socket generates the request and sends it to the system. 3. The system receives the request, validates it, unpacks it, retrieves the sensor statistics from the No-SQL and RDBMS databases, packages a response, and sends the response over the systems web socket. 4. The web browser receives the response over its web socket, unpacks it, and renders the statistics for the user. |
| Alternative Paths | 1. If the user supplies invalid request parameters or the request itself is invalid, the server will send back an error message over its web socket instead. 2. The system will send back an error message if there is an issue accessing either database. |
| Description | This use case allows the user to view sensor reading statistics for a specific sensor. |

|  |  |
| --- | --- |
| Use Case ID | UC-008 |
| Use Case Name | Download Sensor Readings for a Period of Time |
| Participants | User, Web Server, No-SQL Database, RDBMS |
| Pre-Conditions | 1. The web server is online. 2. The No-SQL database is online. 3. The RDBMS is online. 4. The user specifies a sensor, group, start timestamp and end timestamp. 5. There is an existing web socket connection between the users web browser and the system. |
| Post-Conditions | 1. The user receives all sensor readings for the sensor they are currently subscribed to for the time period they specify. |
| Normal Path | 1. The user requests a sensor reading download for a specified sensor, start timestamp and end timestamp using the web browsers web socket. 2. The system retrieves matching readings from the No-SQL database, bundles them into a response, and sends this response back through the systems web socket. 3. The web browser receives the response over its web socket, unpacks it, then saves it to the users local downloads directory. |
| Alternative Paths | 1. If the system encounters an error while retrieving the readings, it will create an error message and send this in response to the web browsers web socket instead. |
| Description | This use case allows the user to download sensor readings for a specific time period. |

|  |  |
| --- | --- |
| Use Case ID | UC-009 |
| Use Case Name | Upload Sensor Readings |
| Participants | Sensor, Web Server. No-SQL Database |
| Pre-Conditions | 1. The sensor is online. 2. The web Server is online. 3. The No-SQL database is online. 4. The sensor has generated a sensor reading. |
| Post-Conditions | 1. The No-SQL database contains the uploaded sensor reading. |
| Normal Path | 1. The sensor generates sensor reading data, packages it in a request, and uploads it to the server. 2. The server receives the request, unpacks it, validates it, stores it in the No-SQL database and sends a response back to the sensor. 3. The sensor receives the response and acts accordingly. 4. The system broadcasts the received sensor reading to any users that are subscribed to the sensor that uploaded the reading. |
| Alternative Paths | 1. If the request is invalid, the system will send back an error to the sensor in response to the request. 2. The system will send back an error message if there is an issue accessing the database. |
| Description | This use case allows a sensor (or gateway) to upload sensor readings to the system. |

|  |  |
| --- | --- |
| Use Case ID | UC-010 |
| Use Case Name | Migrate Readings from No-SQL to RDBMS |
| Participants | Service Worker, No-SQL Database, RDBMS |
| Pre-Conditions | 1. The service worker is online. 2. The No-SQL database is online. 3. The RDBMS is online. 4. The service workers timeout has expired. |
| Post-Conditions | 1. The No-SQL database contains only readings that were taken within the past thirty (30) days. 2. The RDBMS contains the readings migrated over from the No-SQL database. |
| Normal Path | 1. The system initiates the service worker with a timeout. 2. The service worker waits for the timeout to expire. Once the timeout expires, the service worker removes readings older than thirty (30) days from the No-SQL database and stores them in the RDBMS. The service worker then resets its timeout and sends the number of migrated readings back to the system. 3. The system logs the number of migrated records along with a timestamp. |
| Alternative Paths | 1. The service worker may not initialize properly or otherwise fail to initialize. In this case, the system will attempt to start the service worker n times (where n is supplied via a configuration file. If the service worker cannot be started after n attempts, the server stops attempting to initialize the service worker. 2. If there was an error when migrating the readings, the service worker rolls back its changes and sends an error message to the system. The system will then log the error along with a timestamp. 3. The system will send back an error message to the service worker if there is an issue accessing either database. |
| Description | This use case allows a service worker to migrate old sensor readings from the No-SQL database to the RDBMS. |

# System Requirements

This section contains the functional and non-functional requirements that make up the Senslify sensor visualization web application. Requirements were analyzed using the MoSCoW technique and as such are post-fixed with a M=Must have, S=Should have, C=Could have, and W=Won’t have to indicate the priority of each requirement and overall importance to the system. Requirements that are post-fixed with a [W] are impossible to test in the laboratory environment the system is being created in. As such, they will be ignored by testing and should not be implemented in the initial version of the Senslify system.

## Functional Requirements

This section details the functional system requirements of the Senslify sensor visualization software. This includes requirements pertaining to the functionality of the software.

1. The system should allow up to 50 sensors to be simultaneously connected to a single gateway [W].
2. The system shall utilize a well-defined data transfer protocol [M]
3. The system shall asynchronously accept sensor data from devices that follow the specified data transfer protocol and data format [M].
4. The system shall store data younger than three (3) months in a document-oriented database [M].
5. The system should migrate data older than three (3) months to a relational database for long-term storage [S].
6. The system shall display sensor data on a real-time chart within one (1) second of receiving it [M].
7. The system should alert the user when a sensor reading received that is greater than the mean sensor reading + 15% or less than the mean sensor reading – 15% [S].
8. The system shall require the gateway to assign a sensor identifier to a sensor data upload [M].
9. The system shall require the gateway to assign a group identifier to a sensor data upload [M].
10. The system shall allow users to select a sensor within a deployment of sensors using a sensor identifier [M].
11. The system shall allow the user to view the mean, minimum and maximum sensor data for a specific sensor for a user-specified start date and time as well as a user specified end date and time [M].
12. The system shall display the most recent 100 sensor readings in real-time alongside the real-time chart [M].
13. The system shall utilize a web interface to visualize sensor data [M].
14. The system shall allow clients to asynchronously request information from the datastore using HTTP requests [M].
15. The system shall format errors such that they contain an HTTP error code and reason for the error [M].
16. The system shall allow the user to start the software with debugging enabled [M].
17. The system shall format errors such that they contain an HTTP error code, reason for the error, line number that the error occurred and a stacktrace if debugging is enabled when the system is started [M].
18. The system shall verify that uploaded sensor readings are in the proper format before storing them in the datastore [M].
19. The system shall return an error to the uploading gateway if uploaded data is incorrectly formatted [M].
20. The system shall allow users to request individual sensor data from the system [M].
21. The system shall allow users to request sensor data in batches from the system [M].
22. The system shall allow the user to start the system with a provided configuration file [M].
23. The system shall ensure the user-specified sensor identifier and group identifier exist when a user requests data for a sensor [M].
24. The system should allow users to request data from a deployment of sensors for a given period of time [S].
25. The system should allow users to specify the number of sensor readings they want to read from the system [S].
26. The system shall require that each sensor reading declare a reading type [M].
27. The system shall allow users to retrieve the list of all reading types known to the system [S].
28. The system shall allow a gateway to upload sensor data in batches [S].
29. The system shall allow users to retrieve all sensor identifiers that are associated with a specific group identifier [S].
30. The system shall allow users to request all group identifiers from the system [M].
31. The system shall allow users to request all sensor identifiers from the system [S].
32. The system shall uniquely identify a sensor by using a combination of the sensors group identifier and sensor identifier [M].
33. The system shall prompt the user to initialize the datastore when starting the system [M].
34. The system shall return the statistics for a given sensor within 2.5 seconds of the initial request [M].
35. The system shall format Unix timestamps in the following date format: YYYY-MM-DD [M].
36. The system shall comply with ISO-8601 datetime formats when reporting datetime information to the user [M].
37. The system shall format sensor readings in the following format: ‘Time: {time}, Value: {value}’, where {time} is an ISO-8601 short datetime string and {value} is the sensor reading value [M].
38. The system shall convert sensor reading type codes to the appropriate human-readable English text corresponding to each code [M].
39. The system shall allow clients to request the creation of a persistent connection to the web server that once created, can be closed by both the client and the webserver [M].
40. The system shall utilize a well-defined protocol for establishing, tearing-down, and maintaining connections between the web server and clients [M].
41. The system shall ensure that the web server maintains a single channel per sensor [M].
42. The system shall allow the web server to broadcast messages to all clients within a given channel [M].
43. The system shall implement a handler function that gracefully shuts down a channel [M].
44. The system shall ensure that when a channel is shutdown, all connected clients are notified that the channel is being shutdown [M].
45. The system shall provide an abstract database adapter that is extendable by future maintainers of the software [S].
46. The system shall allow users to download sensor data given a start datetime timestamp and an end datetime timestamp [M].
47. The system should allow users to retrieve statistics for an entire group of sensors [S].
48. The system should allow the users web browser to attempt to establish a web socket connection up to three (3) times.
49. The system should ensure that the user is redirected back to the sensor listings page if the web browser is unable to establish a web socket connection with the server.

## Non-Functional Requirements

This section details the non-functional system requirements of the Senslify sensor visualization software. This includes requirements pertaining to the quality aspects of the software.

1. The system shall have no more than 5% downtime over the course of it’s running life [S].
2. The system shall be taken down for maintenance each Saturday night from 10PM to 12AM [S].
3. The system should be built as a self-contained, redeployable container [C].
4. The system should list group names in place of group numbers in the user interface [C].
5. The system shall not utilize closed source libraries, components, or modules in building of the system [M].
6. The system shall utilize well-tested, highly supported open-source libraries, components, and modules in building of the system [].
7. The system should support Bluetooth low energy (BLE) enabled wireless sensors [C].
8. The system shall utilize HTTP templating software for the frontend [M].
9. The system shall utilize the MongoDB document-oriented database for short-term storage of sensor data [M].
10. The system shall utilize WebSockets as defined in RFC 6455 to establish connections between the webserver and clients [M].
11. The system shall ensure that clients and the web server send messages in HTTP-encoded JSON [M].
12. The system shall utilize the Bootstrap CSS3 library [M].
13. The system shall utilize a mobile first design philosophy [M].
14. The system shall be implemented using the asynchronous design philosophy to ensure scalability [M].

# Change Management Process

This project incorporates an agile development approach. The initial iteration of the software will implement core functionality while each successive iteration will implement additional secondary functionality and backlogged requirements per priority and necessity of implementation. The addition of requirements on top of those listed in this document will be discussed in collaboration between Christen and Devendra and will be phased into the project if they are determined to be feasible.

Note that Christen will be detailing change requests in the source code for the project. These requests will be highlighted in comments at the top of the respective source code file(s) and should be easily discernable from the rest of the comments in the source code. These may or may not be integrated into the project. If they are, this will be indicated in the comment itself, otherwise, they are to be considered not implemented.

# Communications Management Plan

This section details the methods in which the project managers will communicate with each other. Now, because there are only two group members in the project, we have decided to on-demand, ad-hoc meetings with each other supplemented by regular text message and email communication. Formal meetings are to be held at the discretion of the group members.