Senslify Test Plan

By: Christen Ford

V 1.0.0

Table of Contents

[Section 1: Introduction and Software Description 3](#_Toc76150296)

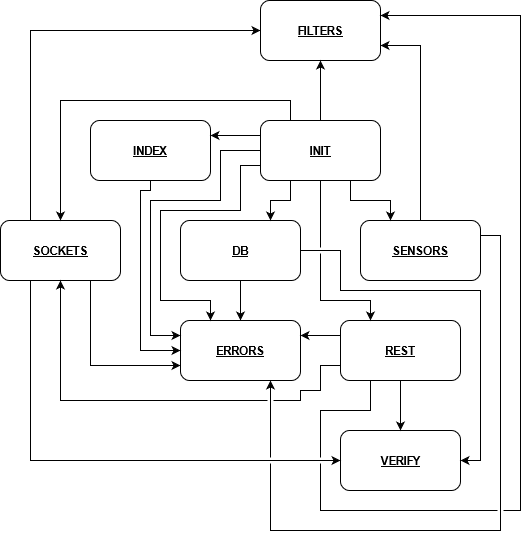
[Section 2: Functionality Review 4](#_Toc76150297)

[Section 3: Test Methodology and Miscellaneous Topics 25](#_Toc76150298)

# Section 1: Introduction and Software Description

The purpose of this document is to detail the test plan for the ‘Senslify’ Python library. This document will cover which functionality to test, the test design methods to be utilized during test case generation, the testing schedule and any other miscellaneous testing necessities. First, I will cover which functionalities of the library will be tested. This includes testing at the unit level, the integration level, and the system level. Next, I will discuss the test design methods that I will use to generate test cases as well as the plan of action with regards to the black box methods I will utilize to carry out testing. Lastly, I will present a preliminary testing schedule detailing the steps needed to successfully carry out this test plan. This portion of this document will also discuss any extra testing information not aforementioned including any testing frameworks and tools that I may utilize and the platform that I will be conducting testing on. These are stated in an effort to maintain reproducibility in these testing efforts so that future testers of the ’Senslify’ library with a similar hardware configuration may also receive similar results as presented by this effort.

‘Senslify’ is a Python software package enabling real-time visualization of live sensor data as it is taken from the source. This package is built from many different systems and incorporates two custom designed application-level protocols enabling communication between sensors and gateways, the web server, and connected clients. Furthermore, unlike traditional web applications, ‘Senslify’ is entirely asynchronous. This design paradigm extends to the way clients communicate with the web server to how the web server communicates with the database and how clients (sensors/gateways and web clients) indirectly communicate with each other. The asynchronous paradigm is notoriously more difficult to test than the synchronous paradigm as messages and responses can occur in any order throughout the system. This leads naturally to an event-driven design where many components wait to receive event triggers and run specific code that is tied to these events when these triggers are executed. For reference, the ‘Senslify’ project structure is shown here (supporting diagrams may be found in the images) folder. This diagram showcases how the different modules communicate with each other (though it is not a UML communication diagram).



# Section 2: Functionality Review

In this section, I review the functionality of the `Senslify` Python software package. This review includes all of the server-side unit-level functionality as well as all of the client-side unit-level functionality. In the following tables, the header lists the module name while each processing row lists each function, whether or not I plan to test that function, a description of the function, and a description of the input/output of the function. Because I wrote ‘Senslify’ myself, I am already intimately familiar with it’s workings, but I will repeat them here. Note that due to the nature of the software package, much of this functionality cannot be automatically tested without use of a platform like Selenium – which at the time of writing, I am unfamiliar with. As such, the following functionality will all be tested by hand by live running the web application on my development machine and sending/receiving requests through Postman. Also note that ‘Senslify’ was built to run in a Python 3.7+ environment. While past versions of Python3 may work with ‘Senslify’, the `asyncio` module may or may not function correctly and because the `aiohttp` depends on `asyncio`, you want to adhere to the Python 3.7+ environment when testing or running the web application.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **CLIENT** | | | | | |
| **ID** | **Function** | **Test?** | **Description** | **Input** | **Output** |
| CLI-000 | teardown | Yes | Sends a request to the web server to tear down the web socket connection between the client and web server on its end. | N/A | N/A |
| CLI-001 | sendJoinRequest | Yes | Sends a request to the web server to add the client to the list of listening clients for the user-supplied sensor identifier and group identifier. | N/A | ‘rqst’: A stringified JSON Object. |
| CLI-002 | errorHandler | Yes | Handles RESP\_ERROR\* commands from the web server. | ‘resp’: A JSON object. | N/A |
| CLI-003 | joinHandler | Yes | Handles RESP\_JOIN commands from the web server. | ‘resp’: A JSON object. | N/A |
| CLI-004 | readingHandler | Yes | Handles RESP\_READING commands from the web server. | ‘resp’: A JSON object. | N/A |
| CLI-005 | downloadHandler | Yes | Handles RESP\_DOWNLOAD commands from the web server. | ‘resp’: A JSON object. | N/A |
| CLI-006 | statsHandler | Yes | Handles RESP\_STATS commands from the web server. | ‘resp’: A JSON object. | N/A |
| CLI-007 | streamHandler | Yes | Handles RESP\_STREAM commands from the web server. | ‘resp: A JSON object. | N/A |
| CLI-008 | onWSClose | Yes | Overrides the same function provided by the WebSocket API. Frees up reserved browser resources when the web socket is closed. | N/A | N/A |
| CLI-009 | onWSOpen | Yes | Overrides the same function provided by the WebSocket API. Sends the initial join request for the user-supplied group identifier and sensor identifier. | err: A Javascipt String. | N/A |
| CLI-010 | onWSReceive | Yes | Overrides the same function provided by the WebSocket API. Unpacks received messages and routes them to the appropriate handler function. | msg: A stringified JSON object. | N/A |
| CLI-011 | createChart | Yes | Utility function that initializes the real-time chart for sensor visualization. | N/A | N/A |
| CLI-012 | getRandomInt | No | Utility function that returns an integer in the range [0, max). | max: An integer. | An integer in the range [0, max). |
| CLI-013 | pushAlert | Yes | Utility function that generates and pushes an alert to the alerts list in the alerts modal box. | reading: A JSON object containing the prior reading. | N/A |
| CLI-014 | reloadChart | Yes | Utility function that destroys the current chart and redraws it (in Chart.js, this is the only way to redraw a chart in preparation for displaying different data). | N/A | N/A |
| CLI-015 | reloadList | Yes | Utility function that empties the real-time list of all currently loaded sensor readings. This function will also load new sensor readings into the list. | dataset: A JSON array. | N/A |
| CLI-016 | onAlertsCleared | Yes | Event handler that clears the alerts listing from the alerts modal whenever the ‘Clear Alerts’ button is pressed in the alerts modal box. | N/A | N/A |
| CLI-017 | onDownloadRequested | Yes | Event handler that generates a download request when the user clicks the download button in the download modal pop-up box. | N/A | ‘rqst’: A stringified JSON Object. |
| CLI-018 | onStatsRequested | Yes | Event handler that generates a statistic request when the user clicks the “Get Stats” button in the statistics modal pop-up box. | N/A | ‘rqst’: A stringified JSON Object. |
| CLI-019 | onStreamChanged | Yes | Even handler that generates a stream change request when the user uses the reading type dropdown to change the type of readings they receive from the web server. | N/A | ‘rqst’: A stringified JSON Object. |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **DB** | | | | | |
| **ID** | **Function** | **Test?** | **Description** | **Input** | **Output** |
| DB-000 | database\_shutdown\_handler | Yes | `aiohttp` handler function that gracefully shuts down the database in the event that the web server enters an unrecoverable state and needs restarted. | app: An `aiohttp` Application instance. | N/A |
| DB-001 | \_\_init\_\_ | Yes | Initializes a new instance of a database adapter. | conn\_str: A Python String.  db: A Python String. | A DatabaseProvider object. |
| DB-002 | get\_connection | Yes | Context manager that temporarily opens a connection to a database by using the Python ‘with’ construct. | conn\_str: A Python String.  db: A Python String. | A DatabaseProvider object in the bounds of a context manager. |
| DB-003 | close | Yes | Co-routine that closes the connection to the database | N/A | N/A |
| DB-004 | init | Yes | Initializes the database according to the ERD schema in the design documentation. | N/A | N/A |
| DB-005 | delete\_group | Yes | Co-routine that deletes a group from the database. This operation cascades to sensors and readings. | groupid: A positive integer. | The number of records deleted. |
| DB-006 | delete\_reading | Yes | Co-routine that deletes a reading from the database. | sensorid: A positive integer.  groupid: A positive integer.  rtypeid: A positive integer.  ts: A UNIX timestamp. | The number of records deleted (either 0 or 1). |
| DB-007 | delete\_readings | Yes | Co-routine that deletes readings from the database. This operation optionally deletes only readings that match a specified reading type. If no reading type is provided, then all readings matching the supplied parameters are deleted. | sensorid: A positive integer.  groupid: A positive integer.  rtypeid: A positive integer (optional). | The number of records deleted. |
| DB-008 | delete\_rtype | Yes | Co-routine that deletes a reading type from the database. This operation cascades to readings. | rtypeid: A positive integer. | The number of records deleted. |
| DB-009 | delete\_sensor | Yes | Co-routine that deletes a sensor from the database. This operation cascades to readings. | sensorid: A positive integer.  groupid: A positive integer. | The number of records deleted. |
| DB-010 | does\_group\_exist | Yes | Co-routine that determines if a specified group identifier already exists in the database. | groupid: A positive integer. | True if the group exists, False otherwise. |
| DB-011 | does\_rtype\_exist | Yes | Co-routine that determines if a specified reading type identifier already exists in the database. | rtypeid: A positive integer. | True if the reading type exists, False otherwise. |
| DB-012 | does\_sensor\_exist | Yes | Co-routine that determines if a specified sensor identifier already exists in the database. | sensorid: A positive integer.  groupid: A positive integer. | True if the sensor exists, False otherwise. |
| DB-013 | find\_max\_groupid | Yes | Co-routine that determines the maximum group identifier stored in the database. | N/A | A DBError exception if one occurs, otherwise a Python dict. |
| DB-014 | find\_max\_sensorid\_in\_group | Yes | Co-routine that determines the maximum sensor identifier for a specified group identifier in the database. | groupid: A positive integer. | A DBError exception if one occurs, otherwise a Python dict. |
| DB-015 | get\_groups | Yes | Co-routine that returns a Python generator function that yields group identifiers. | N/A | A Python generator function that yields group identifiers. |
| DB-016 | get\_rtypes | Yes | Co-routine that returns a Python generator function that yields reading type identifiers. | N/A | A Python generator function that yields reading type identifiers. |
| DB-017 | get\_sensors | Yes | Co-routine that returns a Python generator function that yields sensor identifiers. | groupid: A positive integer. | A Python generator function that yields sensor identifiers. |
| DB-018 | get\_readings | Yes | Co-routine that returns a Python generator function that yields sensor readings. | sensorid: A positive integer.  groupid: A positive integer.  limit: A positive integer. | A Python generator function that yields sensor readings. |
| DB-019 | insert\_group | Yes | Co-routine that inserts a group into the database. | groupid: A positive integer.  alias: A human-readable string representing the groups name. | A tuple containing a boolean that indicates whether the group was successfully inserted as well as an Exception object if one occurred. |
| DB-020 | insert\_readings | Yes | Co-routine that inserts multiple readings into the database. | readings: An iterable containing Python dictionaries.  batch\_size: A positive integer. | A tuple containing a boolean that indicates whether the readings were successfully inserted as well as an Exception object if one occurred. |
| DB-021 | insert\_sensor | Yes | Co-routine that inserts a sensor into the database. | sensorid: A positive integer.  groupid: A positive integer.  alias: A human readable string representing the sensors name. | A tuple containing a boolean that indicates whether the sensor was successfully inserted as well as an Exception object if one occurred. |
| DB-022 | is\_open | Yes | Determines whether the database connection is open or closed. | N/A | True if the database connection is open, False otherwise. |
| DB-023 | open | Yes | Opens the database connection. | N/A | N/A |
| DB-024 | stats\_group | Yes | Co-routine that returns a Python generator function that yields statistics for a user-specified group identifier, sensor identifier, reading type identifier, start timestamp, and end timestamp. | groupid: A positive integer.  rtypeid: A positive integer.  start\_ts: A Python datetime object.  end\_ts: A Python datetime object. | A generator function that yields statistics for sensors within s group. |
| DB-025 | stats\_sensor | Yes | Co-routine that returns sensor reading statistics for a sensor. | sensorid: A positive integer.  groupid: A positive integer.  rtypeid: A positive integer.  start\_ts: A Python datetime object.  end\_ts: A Python datetime object. | A Python dictionary containing statistics for a group. |
| DB-026 | get\_readings\_by\_period | Yes | Co-routine that returns a Python generator that yields sensor readings for a period. | sensorid: A positive integer.  groupid: A positive integer.  start\_ts: A Python datetime object.  end\_ts: A Python datetime object. | A Python generator function that yields sensor readings. |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ERRORS** | | | | | |
| **ID** | **Function** | **Test?** | **Description** | **Input** | **Output** |
| ERR-000 | generate\_error | No | Utility function that generates error pages for clients. | text: A String.  status: A positive integer. An HTTP status code. | An `aiohttp` Response object. |
| ERR-001 | traceback\_str | No | Utility function that generates a traceback string. Useful for server-side debugging. | exception: A Python Exception to trace. | A String. |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **FILTERS** | | | | | |
| **ID** | **Function** | **Test?** | **Description** | **Input** | **Output** |
| FIL-000 | filter\_date | Yes | Filters a UNIX timestamp into a YYYY-MM-DD formatted String suitable for HTML date input controls. | d: A UNIX timestamp.  locale: A string. | A formatted date String. |
| FIL-001 | filter\_datetime | Yes | Utility function that generates `i18n` compliant datetime strings from timestamps for use by the jinja2 template engine. | dt: A datetime instance.  fmt: A String.  locale: A String. | A formatted datetime String. |
| FIL-002 | filter\_reading | Yes | Generates a formatted string for a sensor reading. | reading: A Python dict. | A formatted sensor reading String. |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **INDEX** | | | | | |
| **ID** | **Function** | **Test?** | **Description** | **Input** | **Output** |
| IND-000 | build\_sensors\_url | Yes | Utility function that creates a URL for a supplied group identifier. | request: An aiohttp Request object.  groupid: A positive integer. | A String representing a route or an `aiohttp` Response object. |
| IND-001 | index\_handler | Yes | Defines an HTML GET endpoint for the index page (home page). | request: An aiohttp Request object. | A Python dictionary containing template parameters or an aiohttp Response object. |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **INIT** | | | | | |
| **ID** | **Function** | **Test?** | **Description** | **Input** | **Output** |
| INI-000 | create\_db | Yes | Returns an instance of a subclass of the DatabaseProvider class appropriate for the functions input. Will prompt the user if authentication is required to connect to the target database. If it is, this function will securely obtain that authentication from the user. | conn\_str: The connection string for the database. Specified in the configuration file.  db\_provider: A string representing the DatabaseProvider subclass to instantiate and return from this function. Specified in the configuration file.  auth\_required: A boolean that indicates whether database authentication is required or not. | An instance of a subclass of the DatabaseProvider class. |
| INI-001 | get\_local\_ip | Yes | Gets the IP address of the machine running the server. Used during WebSocket initialization to ensure that clients do not have to manually supply the servers IP address during WebSocket creation on the client. Also used to define the host address for the web server. | N/A | A String representing the IP address of the web server. |
| INI-002 | build\_app | Yes | Factory function that is used to create new instances of the web server. | config\_file: The path to the servers configuration file. This parameter is optional. | An aiohttp Application instance. Essentially an instance of the web server. |
| INI-003 | main | Yes | Entry point for the web server. When running this file from the command line or through the `senslify` command, this function is what gets called first. | N/A | N/A |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **REST** | | | | | |
| **ID** | **Function** | **Test?** | **Description** | **Input** | **Output** |
| RST-000 | \_generate\_alias | Yes | Utility function used to generate plain-English aliases. | n: A positive integer > 0. | A string. This function may sometimes return a value less than n |
| RST-001 | \_download\_handler | Yes | Co-routine that handles download commands for the rest\_handler function. | request: An aiohttp Request object.  params: A Python dict. | An aiohttp Response object. |
| RST-002 | \_find\_handler | Yes | Co-routine that handles find commands for the rest\_handler function. | request: An aiohttp Request object.  params: A Python dict. | An aiohttp Response object. |
| RST-003 | \_stats\_handler | Yes | Co-routine that handles stats commands for the rest\_handler function. | request: An aiohttp Request object.  params: A Python dict. | An aiohttp Response object. |
| RST-004 | \_provision\_handler | Yes | Co-routine that handles provision commands for the rest\_handler function. | request: An aiohttp Request object.  params: A Python dict. | An aiohttp Response object. |
| RST-005 | \_upload\_handler | Yes | Co-routine that handles upload commands for the rest\_handler function. | request: An aiohttp Request object.  params: A Python dict. | An aiohttp Response object. |
| RST-006 | rest\_handler | Yes | Co-routine that routes REST requests to the appropriate \_ handler function. | request: An aiohttp Request object. | An aiohttp Response object. |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **SENSORS** | | | | | |
| **ID** | **Function** | **Test?** | **Description** | **Input** | **Output** |
| SEN-000 | info\_handler | Yes | Defines an HTML GET endpoint for the sensor information page. | request: An `aiohttp` Request object. | An `aiohttp` Response object. |
| SEN-001 | build\_info\_url | Yes | Utility function that creates a URL for a sensor identifier. | request: An `aiohttp` Request object.  sensor: A Python dictionary containing sensor parameters. | A String representing a route or an `aiohttp` Response object. |
| SEN-002 | sensors\_handler | Yes | Defines an HTML GET endpoint for the sensor listings page. | request: An `aiohttp` Request object. | An `aiohttp` Response object. |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **SOCKETS** | | | | | |
| **ID** | **Function** | **Test?** | **Description** | **Input** | **Output** |
| SOC-000 | \_does\_room\_exist | Yes | Determines if a room/channel already exists. | rooms: A Python dict.  groupid: A positive integer.  sensorid: A positive integer. | True if a room already exists for a groupid/sensorid tuple, False otherwise. |
| SOC-001 | \_does\_ws\_exist | Yes | Determines if a websocket already exists. | rooms: A Python dict.  groupid: A positive integer.  sensorid: A positive integer.  ws: An aiohttp.web.WebSocket object. | True if the websocket already exists for the inferred sensor room, False otherwise. |
| SOC-002 | \_leave | Yes | Co-routine that allows a websocket to stop receiving updates for a sensor. | rooms: A Python dict.  groupid: A positive integer.  sensorid: A positive integer.  ws: An aiohttp.web.WebSocket object. | N/A |
| SOC-003 | \_join | Yes | Co-routine that allows a websocket to receive updates for a sensor. | rooms: A Python dict.  groupid: A positive integer.  sensorid: A positive integer.  ws: An aiohttp.web.WebSocket object. | True if the websocket was successfully subscribed to the sensor room, False otherwise. |
| SOC-004 | \_change\_stream | Yes | Co-routine that allows a websocket to change the type of sensor readings it receives. | rooms: A Python dict.  groupid: A positive integer.  sensorid: A positive integer.  ws: An aiohttp.web.WebSocket object.  rtypeid: A positive integer | N/A |
| SOC-005 | message | Yes | Co-routine that broadcasts received sensor readings to a room/channel of websockets. | rooms: A Python dict.  groupid: A positive integer.  sensorid: A positive integer.  msg: A Python dict. | N/A |
| SOC-006 | ws\_handler | Yes | Co-routine that routes received websocket requests to the appropriate handler function. | request: An `aiohttp` Request object. | N/A |
| SOC-007 | socket\_shutdown\_handler | Yes | Co-routine that allows the web server to gracefully close all existing web socket connections before system shutdown. | app: An `aiohttp` Application object. | N/A |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **VERIFY** | | | | | |
| **ID** | **Function** | **Test?** | **Description** | **Input** | **Output** |
| VER-000 | \_verify\_find\_request | Yes | Co-routine that verifies a REST request that has the `find` cmd. | request: An `aiohttp` Request object.  params: A Python dictionary. | A tuple containing a boolean and a status message. The tuple returns as the Python values True and None if there the request verified successfully, otherwise it returns with the Python value False as well as a status string. |
| VER-001 | \_verify\_stats\_request | Yes | Co-routine that verifies a REST request that has the `stats` request or a RQST\_SENSOR\_STATS command received via a WebSocket. | request: An `aiohttp` Request object.  params: A Python dictionary. | A tuple containing a boolean and a status message. The tuple returns as the Python values True and None if there the request verified successfully, otherwise it returns with the Python value False as well as a status string. |
| VER-002 | \_verify\_download\_request | Yes | Co-routine that verifies a REST request that has the `download` command or a RQST\_DOWNLOAD command received via a WebSocket. | request: An `aiohttp` Request object.  params: A Python dictionary. | A tuple containing a boolean and a status message. The tuple returns as the Python values True and None if there the request verified successfully, otherwise it returns with the Python value False as well as a status string. |
| VER-003 | \_verify\_upload\_request | Yes | Co-routine that verifies a REST request that has the `upload` command. | request: An `aiohttp` Request object.  params: A Python dictionary. | A tuple containing a boolean and a status message. The tuple returns as the Python values True and None if there the request verified successfully, otherwise it returns with the Python value False as well as a status string. |
| VER-004 | \_verify\_provision\_request | Yes | Co-routine that verifies a REST request that has the `provision` command. | request: An `aiohttp` Request object.  params: A Python dictionary. | A tuple containing a boolean and a status message. The tuple returns as the Python values True and None if there the request verified successfully, otherwise it returns with the Python value False as well as a status string. |
| VER-005 | \_verify\_join\_command | Yes | Co-routine that verifies the RQST\_JOIN command received via WebSockets. | request: An `aiohttp` Request object.  params: A Python dictionary. | A tuple containing a boolean and a status message. The tuple returns as the Python values True and None if there the request verified successfully, otherwise it returns with the Python value False as well as a status string. |
| VER-006 | \_verify\_close\_command | Yes | Co-routine that verifies the RQST\_CLOSE command received via WebSockets. | request: An `aiohttp` Request object.  params: A Python dictionary. | A tuple containing a boolean and a status message. The tuple returns as the Python values True and None if there the request verified successfully, otherwise it returns with the Python value False as well as a status string. |
| VER-007 | \_verify\_stream\_command | Yes | Co-routine that verifies the RQST\_STREAM command received over WebSockets. | request: An `aiohttp` Request object.  params: A Python dictionary. | A tuple containing a boolean and a status message. The tuple returns as the Python values True and None if there the request verified successfully, otherwise it returns with the Python value False as well as a status string. |
| VER-008 | verify\_rest\_request | Yes | Co-routine that verifies a received REST request. | request: An `aiohttp` Request object. | True if the request is valid, False otherwise. |
| VER-009 | verify\_ws\_request | Yes | Co-routine that verifies a request received over a WebSocket. | request: An `aiohttp` Request object. | True if the request |

# Section 3: Test Methodology and Miscellaneous Topics

Automated testing on the Senslify platform requires utilizing a platform such as Selenium to simulate clicks on the user interface. At the time of this writing, I am only familiar with Selenium on a “I know what it does and why you would use it” basis. I have not actually had the incentive to use it on projects in the past as none of them were large enough to warrant using it. I do know that it is an extensive platform that does come with quite a bit of a learning curve. I also do not know how to automate the testing surrounding the WebSocket connection. Some of that functionality is setup automatically when a user views the sensor information page, but the remaining functionality needs to be manually triggered via the UI components on the page. These factors aside, I do not have the time to learn the Selenium platform for this project and instead, I will be conducting testing manually through the user interface. While the REST API could be automatically tested, assuming the web server and database are online, I am still going to test it manually as well. These tests will be conducted throughout the development of the project (during and after each iteration). They will not all be performed at once.

Included with this document is a test specification that lists every test case I was able to generate using a variety of both white-box and black-box testing techniques. The tests in this specification serve as a formal listing of the tests I will carry out over time as the implementation moves closer to initial completion. Future maintainers that are familiar with automated testing techniques for web application can implement these test cases in their testing platform, but as stated earlier, I will conduct each of these test cases manually. Note that each of the test cases in the test specifications are generated from the features in this document. Each feature will have one passing test case and at least one failure case (the exact number of which is dependent on the number of inputs the function accepts).