

# **Software Design Document**

**for**

## **Muridae Visualization Utility**

**Prepared by Shane Chamberlain,  
Justin Rice  
and Sean Walsh**

**CPSC 430, Software Engineering, Fall 2016**

**November 29, 2016**

# Table of Contents

<b>1. Introduction</b>	<b>5</b>
1.1 Purpose	5
1.2 Scope	5
1.3 Overview	5
1.4 Reference Material	6
1.5 Definitions and Acronyms	6
<b>2. System Overview</b>	<b>7</b>
<b>3. System Architecture</b>	<b>7</b>
3.1 Architectural Design	7
3.2 Decomposition Description	8
3.2.1 Mouse Simulation module	8
3.2.2 Data Import and Validation module	10
3.2.3 File Upload module	13
3.2.4 Upload-Validation-Simulation-Database Integration	13
3.2.5 View and Update Data module	14
3.2.6 Login module	15
3.2.7 Heatmap Module	17
3.3 Design Rationale	20
<b>4. Data Design</b>	<b>20</b>
4.1 Data Description	20
4.2 Data Dictionary	21
4.2.1 Dataset Table	21
4.2.2 User Table	22

---

---

<b>5. Component Design</b>	<b>22</b>
Mouse Simulation subsystem	22
Data Import and Validation subsystem	22
File Upload subsystem	23
Login subsystem	23
Heatmap Module	24
<b>6. Human Interface Design</b>	<b>25</b>
6.1 Overview of User Interface	25
6.2 Screen Images	26
6.3 Screen Objects and Actions	28
<b>7. Requirements Matrix</b>	<b>33</b>
<b>8. Appendices</b>	<b>34</b>
Appendix A: A valid data file	34

---

## Table of Figures

1: The M-VU System Decomposition Diagram	8
2: The Python list data structure which will store a subject's location path	9
3: The Python dictionary data structure which will store a subject's heatmap data	9
4: Mouse Simulation subsystem class diagram	9
5: Mouse Simulation subsystem sequence diagram	10
6: Data Import and Validation Subsystem decomposition diagram	12
7: File Upload Subsystem flow diagram	14
8: View and Update Data flow diagram	15
9: Login Subsystem login flow diagram	16
10: Login Subsystem logout flow diagram	17
11: Heatmap Module Design Flow	19
12: Entity Relationship Diagram (ERD) for M-VU Database	21
13: Login Screen	26
14: Main Application Screen	26
15: View Maps Screen	27
16: Manage Data Screen	27
17: Manage Users Screen	28
18: How to login	28
19: Header	29
20: Main Page Navigation	29
21: Header Navigation	29
22: Return to Main Page button	29
23: (Left) View Map Panel, (Right) 'Select Data Set' dropdown active	30
24: (Left) Manage Data Panel, (Right) 'Select Data Set' dropdown active	30
25: Clicking 'Upload Data Set' will display a file chooser	31
26: selected filename displayed on screen	31
27: Define Sensor Grid panel	32
28: Manage Users panel	32
29: Requirements Matrix	33
30: A sample of M-VU data for upload	34

---

# 1. Introduction

## 1.1 Purpose

The purpose of this Software Design Document (SDD) is to describe in detail the system design for the Muridae Visualization Utility (M-VU). This document will outline a full breakdown of M-VU subsystems and their architecture. The intended audience for this document is project developers and testers.

## 1.2 Scope

The M-VU software will be a web-based application which will parse gathered data to create user-defined vector and heat maps. The primary purpose of M-VU will be to create these visual representations of the gathered data for further analysis and study. Datasets can be saved and managed, allowing comparison over a broad range of time periods. Both vector and heat maps are customizable, presenting several viewing options to suit the user's particular needs.

## 1.3 Overview

This SDD is organized into eight sections, the first being this introduction to the SDD and the M-VU software. The second section provides a more detailed system overview. The third section breaks M-VU into its component modules, and shows the interactions between each module. The fourth section describes the main data pieces for the system, and how they are store on M-VU's database.

The fifth section is a guide to the design of individual components, including algorithms used for computation. The user interface is outlined in section six, and section seven is a requirements matrix linking components and data structures to the functional requirements outlined in the M-VU Software Requirements Specification (SRS)[1]. The eighth section includes additional material and references.

## 1.4 Reference Material

[1] Parker C., Rice J., Thompson A., Software Requirements Specification for Muridae Visualization Utility, University of Mary Washington, Fredericksburg, Va., September, 2016.

---

## **1.5 Definitions and Acronyms**

### **1.5.1 Admin User**

A user level which is permitted to import and manage datasets.

### **1.5.2 Heat Map**

A map in the M-VU system which shows frequency and duration of a subject's visit to a particular location.

### **1.5.3 Grid**

The Grid is a pre-defined array of sensors, forming a roughly square area over which subject location readings are taken.

### **1.5.4 M-VU**

Muridae Visualization Utility is the software which this SRS describes (pronounced EM-VIEW).

### **1.5.5 SDS**

This Software Design Document.

### **1.5.6 SRS**

The M-VU Software Requirements Specification.

### **1.5.7 User**

A base-level user which is permitted to view data.

### **1.5.8 Vector Map**

A map in the M-VU system which shows a directed graph of a subject's travel over the grid of study.

## **2. System Overview**

The purpose of M-VU is to visualize data gathered from placing subjects (each identified by a unique RFID) in a grid of sensors, and then recording their location as they move about the grid. The data is read and collected in a Microsoft Excel spreadsheet, and exported to a comma-separated values file, with each line in the CSV representing a subject's time in a particular location.

---

M-VU will allow a user to upload a data file, and the system will parse the data into individual data lines, inserting each into an online database. M-VU will validate data as it is inserted into the database. It will allow a user to manage the uploaded data sets, deleting old data and uploading new sets.

Users will then be able to visualize uploaded data, in either a vector map or a heat map. The data can be viewed statically (i.e. the map at the end point of the uploaded data, or the map at the midpoint), or can be played, with the map incremented and redrawn as it steps through at user-defined time increments.

Users will have several customization options for these views. Users can select which subjects from a particular data set to include, or can mix and match subjects from different data sets. The display options (color and labels) are also customizable.

## **3. System Architecture**

### **3.1 Architectural Design**

M-VU will utilize a Model-View-Controller architecture, attempting to split the simulation aspect of the subjects moving through a data set from the software designed to display the web site, and the functionality to allow a user to select which data (selecting from different data sets and different subjects in the data set) they view and how it is displayed.

Systems will be split to log in to M-VU, to import new data, to view uploaded data, to manage user accounts (for administrator accounts and data-viewing accounts), and to simulate the data over user-defined steps of time (see Figure 1).

M-VU will be designed with the Flask architecture, with Python to support server-side operations and JavaScript for client-side interaction. The simulation components will be coded in Python, for simple interaction with server-side code.

The site will be hosted on Digitalocean with an Ubuntu Linux build as the operating system, and will include a PostgreSQL database to store uploaded data. Bootstrap dynamic web page elements will be used to style the M-VU web pages.

---

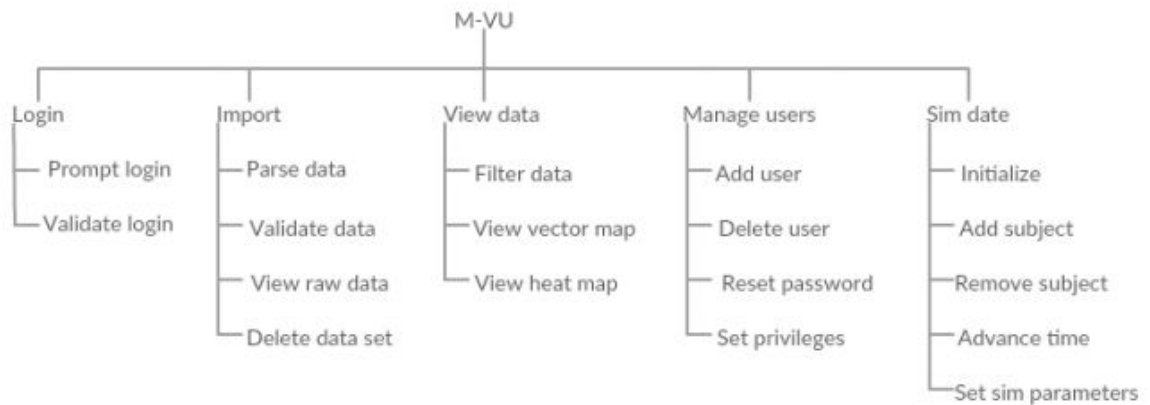


Figure 1: The M-VU System Decomposition Diagram

## 3.2 Decomposition Description

### 3.2.1 Mouse Simulation module

The Mouse Simulation module (see Figure 4 and Figure 5) is a relatively uncomplicated object-oriented system which simulates the movement of subjects through the Grid, by constructing mouse objects and advancing through the simulation at user-defined time intervals. Based on the given dataset, a mouse's location is recorded and updated at each time interval.

At any point during the simulation, member functions generate a path for each subject (tracing its steps through the grid) or generate heat map data (by reporting how much time each subject has spent in each grid location). A subject's path is a list of strings, sorted in the order in which the subject visited a particular location (see Figure 2 and Figure 3). But iterating over the path, a subject's movement through the grid can be retraced.

The Mouse Simulation will be coded in Python, for simple interaction with M-VU's Python server code. The module will use exclusively Python 3.0 standard libraries, and will not be multithreaded.

---



[“RFID2”, “RFID25”, “RFID6”, “RFID2”, “RFID14”, “RFID2”, “RFID15”]

Figure 2: The Python list data structure which will store a subject’s location path. This particular subject visited seven Grid locations, starting at RFID2 and ending at RFID15.

{“RFID2”: 2340, “RFID6”: 453, “RFID25”: 4930, “RFID15”:100, “RFID14”: 8493}

Figure 3: The Python dictionary data structure which will store a subject’s heatmap data. Each key represents a Grid location, and each value represents the total number of time ticks spent at that location.

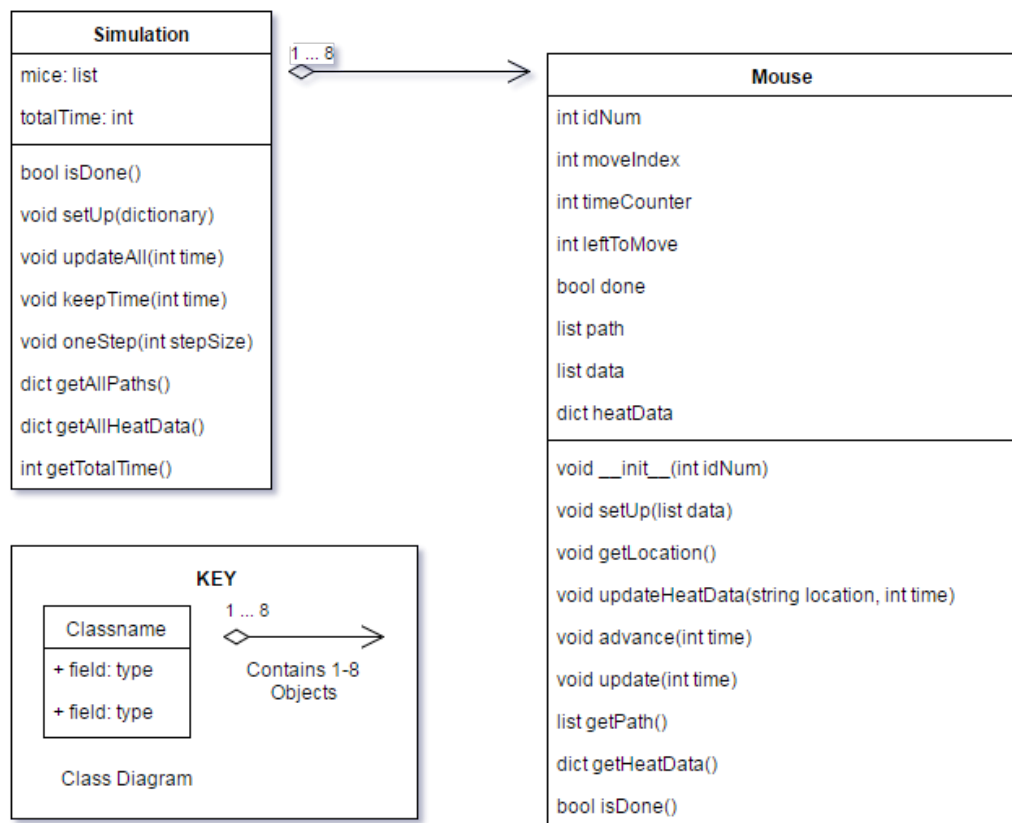


Figure 4: Mouse Simulation subsystem class diagram

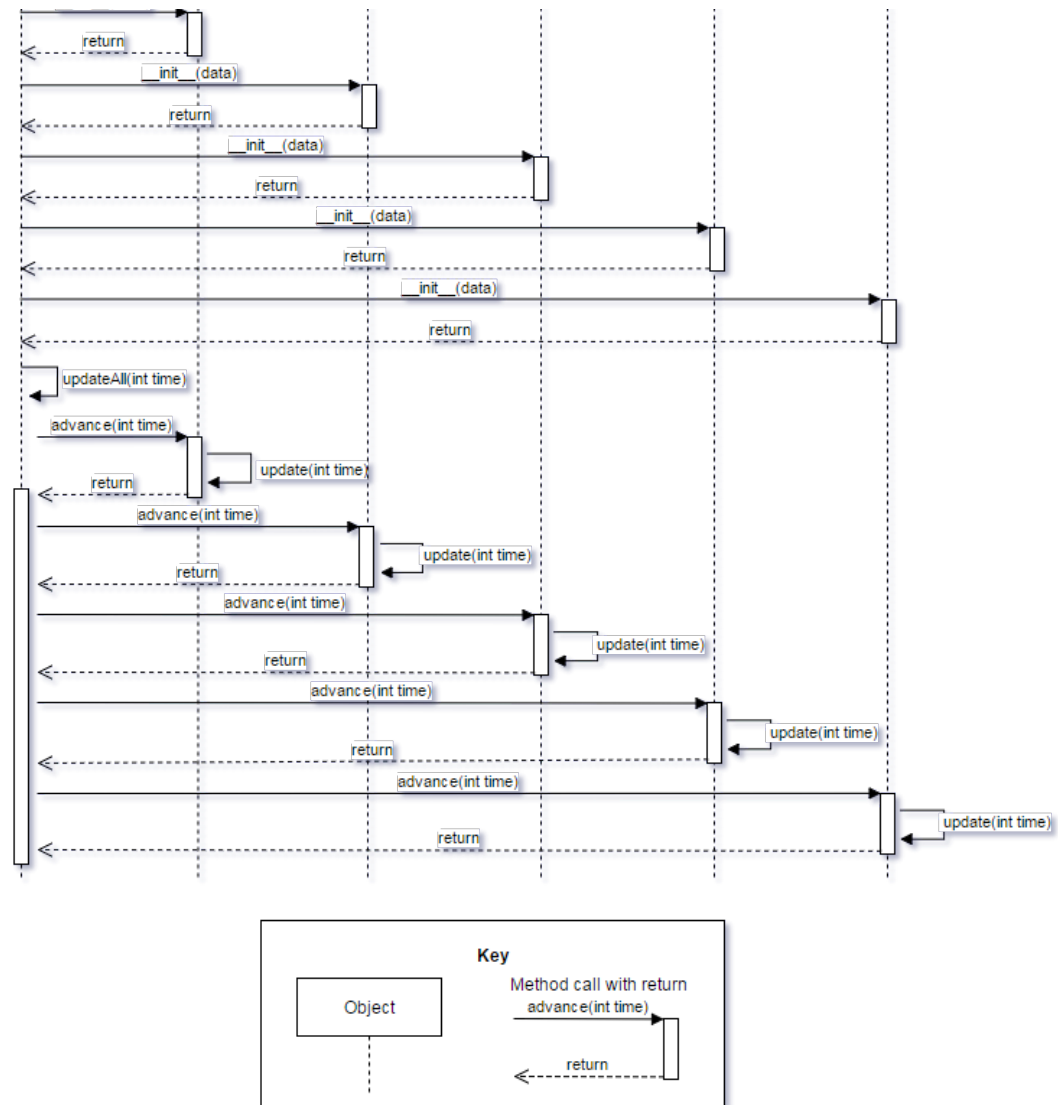


Figure 5: Mouse Simulation subsystem sequence diagram for one time advance over five subjects in a dataset.

### 3.2.2 Data Import and Validation module

The Data Import and Validation module is required to parse an uploaded Excel dataset into an INSERT statement into the M-VU database. The file is split into a series of lines, and each line is split and parsed for the key values which consist of a dataset member (see Figure 6).

Each line is reassembled in the proper format, and then the series of lines are assembled into a complete dataset, and then formed into a query for insertion into the database.

The Data Import and Validation module will be coded in Python, for simple interaction with M-VU's Python server code and for direct access to the M-VU database. The subsystem will not be multithreaded.

---

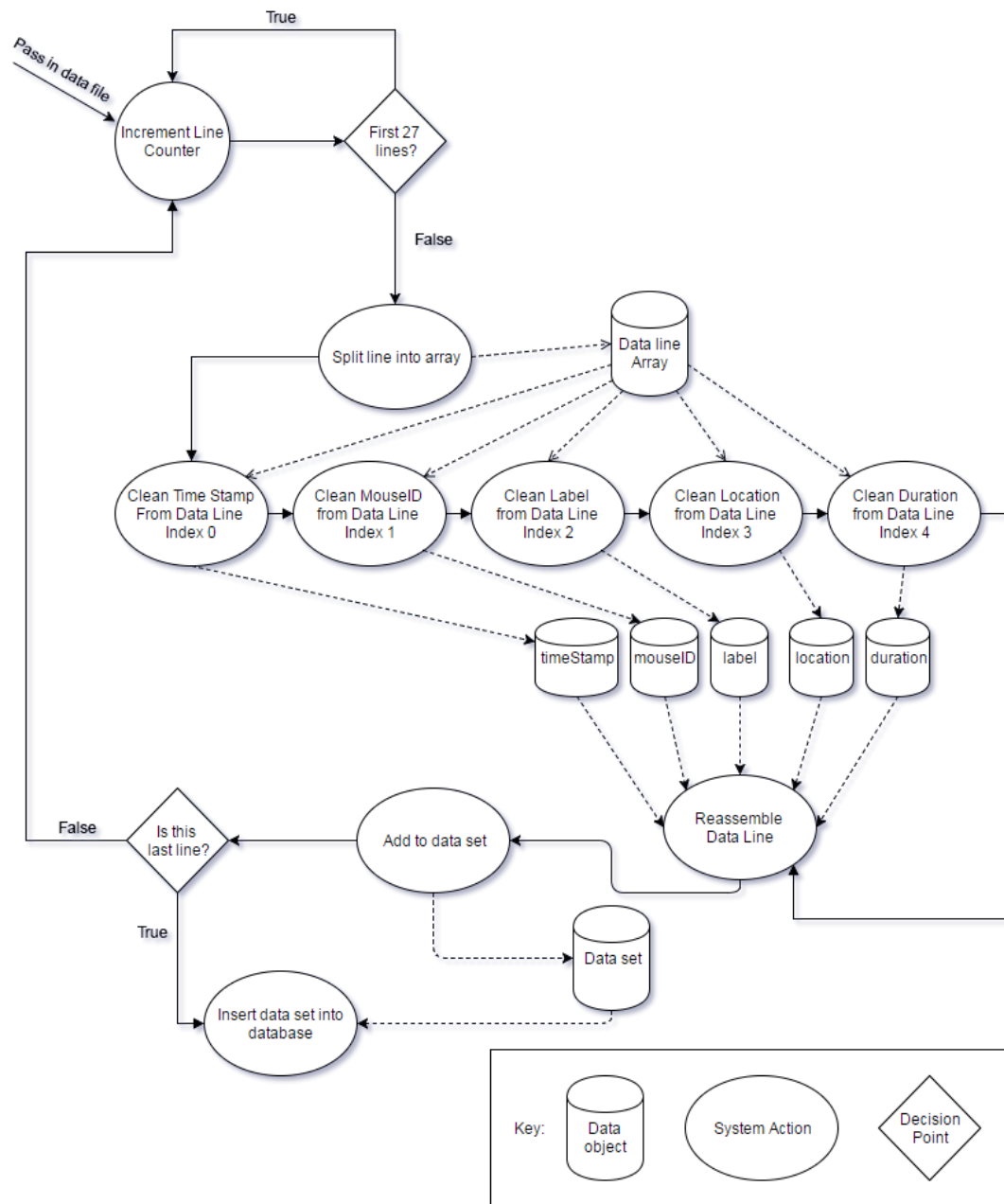


Figure 6: Data Import and Validation Subsystem decomposition diagram

### 3.2.3 File Upload module

A Python Flask plugin (Werkzeug) has been implemented in order to handle user specified file uploads. This particular plugin has been chosen because of its ease of use and user friendly features.

The upload system interface consists of an upload widget with security and file compatibility validation features. The default acceptance setting for the upload plugin is to accept all file formats. The final implementation for this application will only accept Microsoft Excel compatible formats.

The “secure\_filename” function is utilized to ensure that the user specified file does not have issues associated with its file name, if it does then it will be made safe. Potential issues include someone using a regular expression as a file name. The “allowed\_file” function is responsible for ensuring that only files with the proper extensions are uploaded.

Additional information on this plugin can be accessed in its documentation located:

<http://werkzeug.pocoo.org/>

Once a file is selected for upload, the Data Import and Validation module generate and display data lines, and the user defines the grid to apply to the data set, either defining a new grid, or mimicking a grid from a previously uploaded data set. The file is then parsed with the Data Import and Validation module, and then passed to the Mouse Simulation module to generate vector and heat map data for upload.

### 3.2.4 Upload-Validation-Simulation-Database Integration

The File Upload, Data Import and Validation, and Mouse Simulation modules work in concert to add a data set to the M-VU database. Data is passed between these three modules to construct a properly formatted PostgreSQL insert statement, thus making this data available for map views (see Figure 7).

The integration is done in Python as server-side operations. The uploaded file is passed to the Data Import and Validation module, where it is parsed into a series of data lines. The data is then displayed in the Upload user interface, and the user is prompted to either define a grid of sensors for this data, or to mimic the grid of a previously-uploaded data set.

---

Once a valid grid is entered, the user is presented with the option to save the data. Saving data will send the data lines to a Mouse Simulation object, which generates the Heat Map and Vector Map Data.

Finally, the Mouse Simulation data is paired with the sensor grid mapping and either a user-defined or a default data set name, and the data is inserted into the M-VU database.

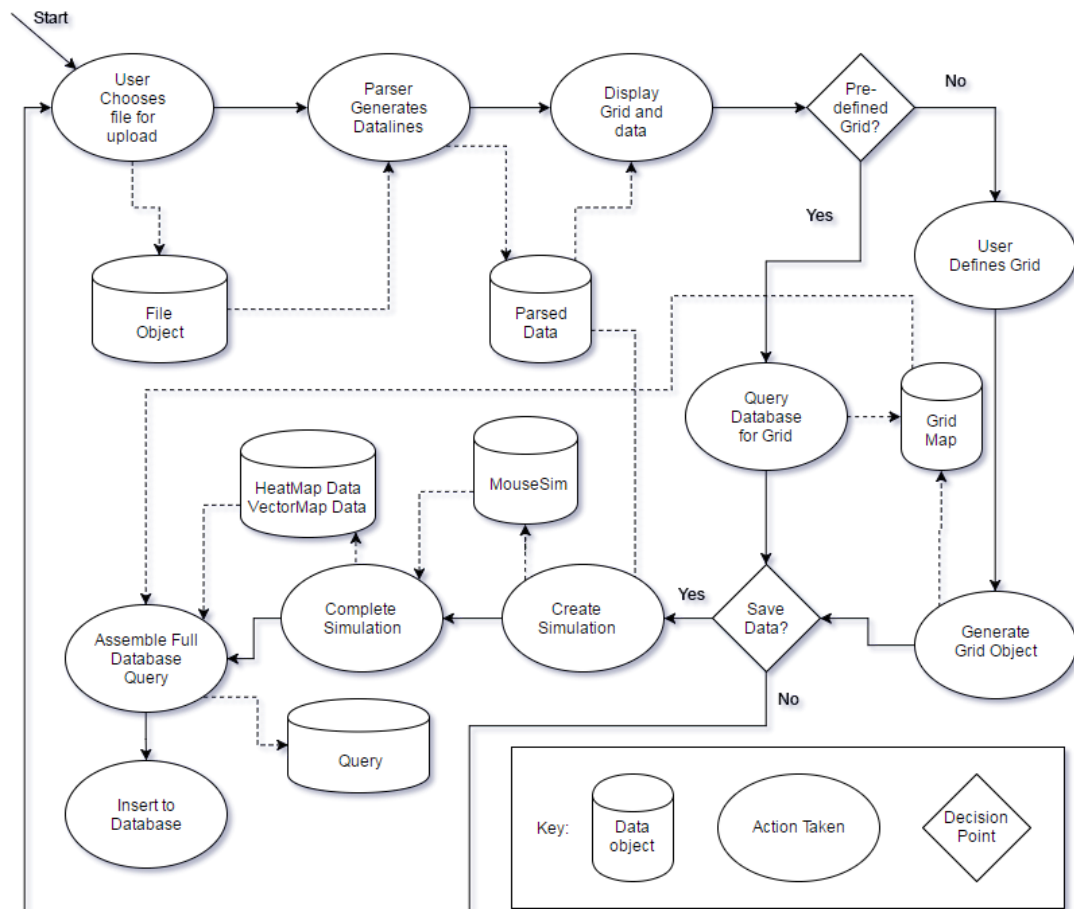


Figure 7: File upload Subsystem flow diagram

### 3.2.5 View and Update Data module

Admin users have the ability to view the parsed Vector Map and Heat Map data for an uploaded data set, and edit the data set's name and sensor grid mapping. Admin users can also delete a previously uploaded data set from the M-VU database (see Figure 8).

The View Data page is implemented with Flask and HTML. A sensor grid is presented for the user to define the mapping. Alternatively, a drop-down menu is populated with all other data sets uploaded to M-VU, and the user can select one of these to mimic - a copy of the locations map will be created and added to the new data set's database entry.

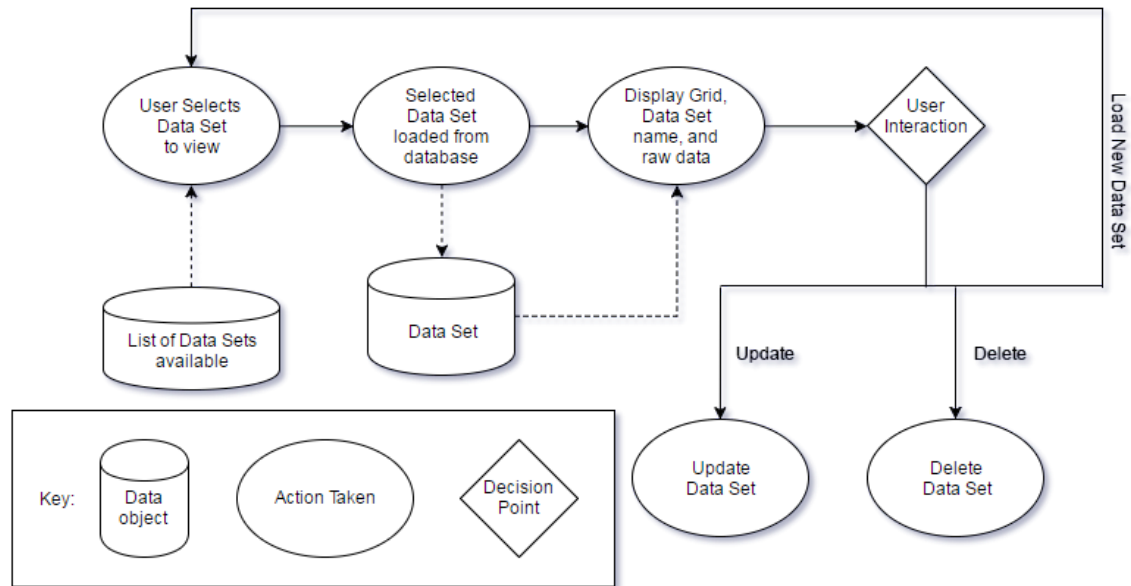


Figure 8: View and Update Data flow diagram

### 3.2.6 Login module

For the login module we will be implementing a Flask plugin, Flask-Login, to handle signing users in and out, and keeping track of their status (whether they are currently logged in). We chose this plugin due to previous experience and ease of use.

The module will consist of a user entering in their unique username and password on the applications “login” screen. A database query will be made to the M-VU Users database upon the submission of the credentials to determine their validity. When a successful query is made, the user's information will be stored in the Flask-Login system and be designated as the “current\_user”. The main index page will be loaded along with the application's page header, which will include among other things, the current user's username and an option for that user to “logout”. A flow diagram for this process is shown in Figure 9.

Finally, when a user “logs out”, their information will be removed from the Flask-Login system and the user will be returned to the application’s “login” screen. A flow diagram for this process is shown in Figure 10.

Additional information on this plugin can be accessed in its documentation located:  
<https://flask-login.readthedocs.io/>

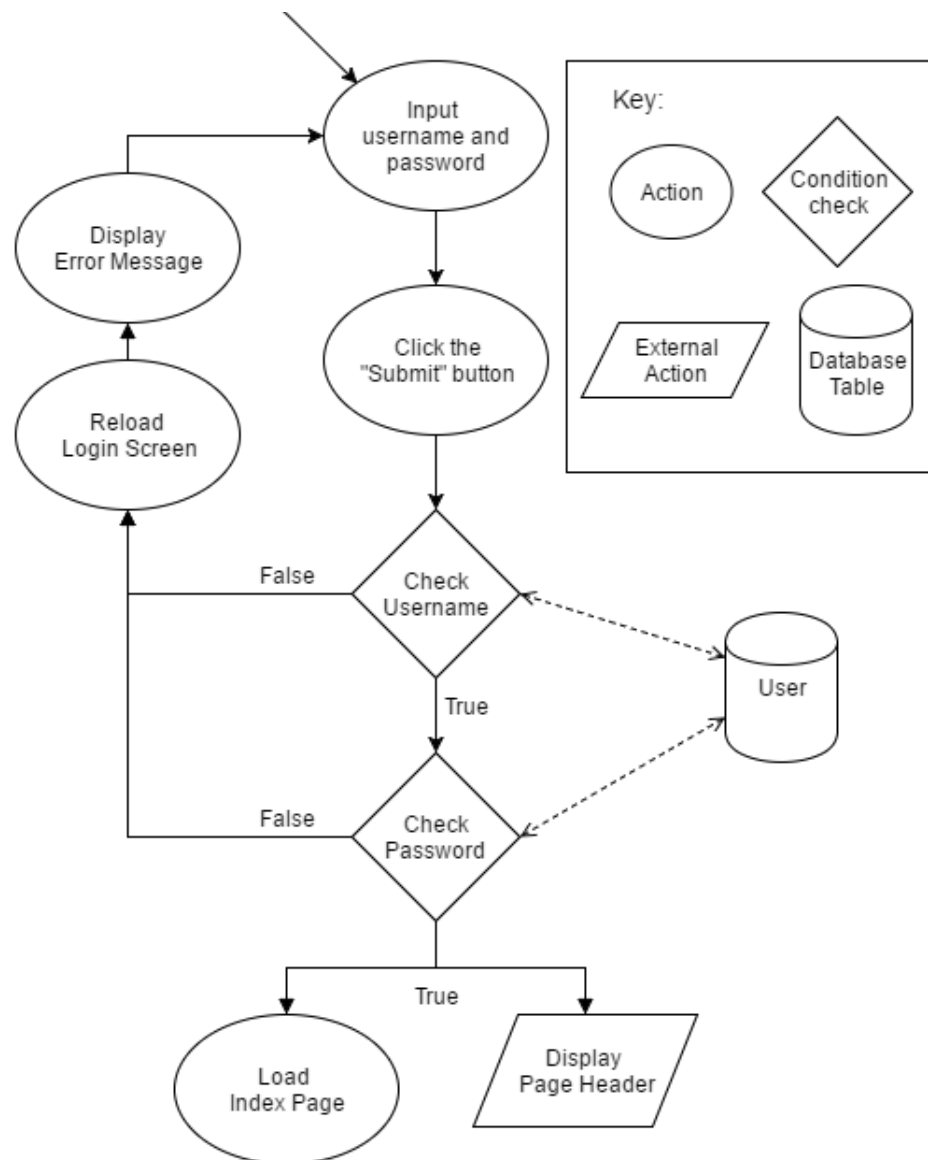


Figure 9: Login Subsystem login flow diagram



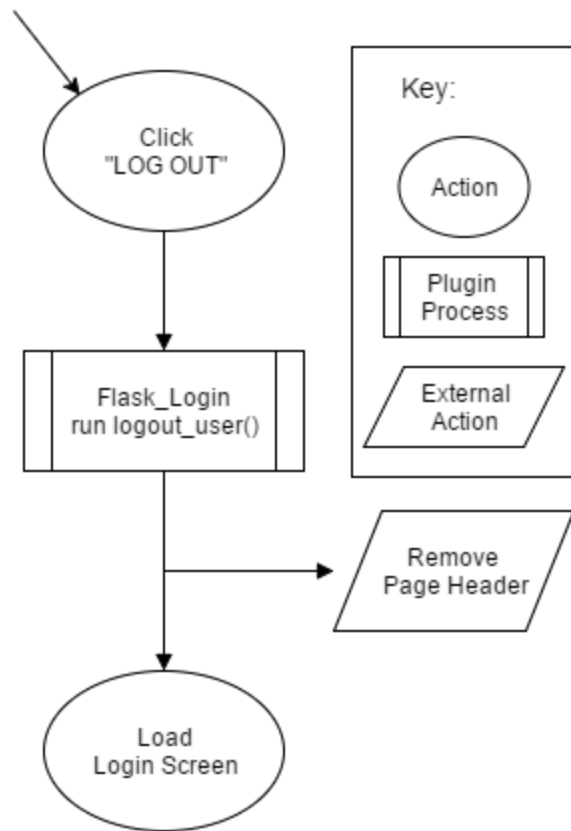


Figure 10: Login Subsystem logout flow diagram

### 3.2.7 Heatmap Module

The Heatmap Module utilizes an external plugin (heatmap.js) designed and built in-house by the development team. It provides the ability, given a dataset, to create and display a grid-style heatmap inside of a HTML Canvas object. The purpose for this module is to visually display heatmap data of different subjects (mice) from datasets loaded into the application by a user.

The plugin itself is outfitted with several options to modify the appearance of the heatmap. These options include: **rows** - the number of grid rows, **cols** - the number of grid columns, **border** - the whitespace border around the grid, **gutter** - the whitespace between the grid squares, **colors** - (set as: {low, high}) the preferred color gradient, **radius** - the radius curve of the grid square corners, **stroke** - whether the grid squares will display a border.

The Heatmap Module itself (See Figure 11) is an angular controller paired with socketio messaging. When a user loads the Maps screen of the application the controller will request a list of available datasets from the server. The controller will then populate the 'Select Data Set' dropdown menu. When a user selects a dataset from the 'Select Data Set' dropdown menu a socketio message is sent to the server to retrieve the dataset stored in the 'datasets' table of the database. When the server returns the dataset, the controller will populate the 'Select Mouse' dropdown menu with the available mice for that dataset. It will also reformat the dataset and get it ready to be displayed by the controller. The user will then select a mouse id from the 'Select Mouse' dropdown menu and click the 'Display Heat Map' button. The controller will map a mouse's heat map data to a grid mapping and pass it to the heatmap.js plugin to be displayed. Figure 11 shows a flow diagram for this process.

Additional information on this plugin can be accessed in its documentation located:

<https://github.com/KeberXela87/heatmap.js>

---

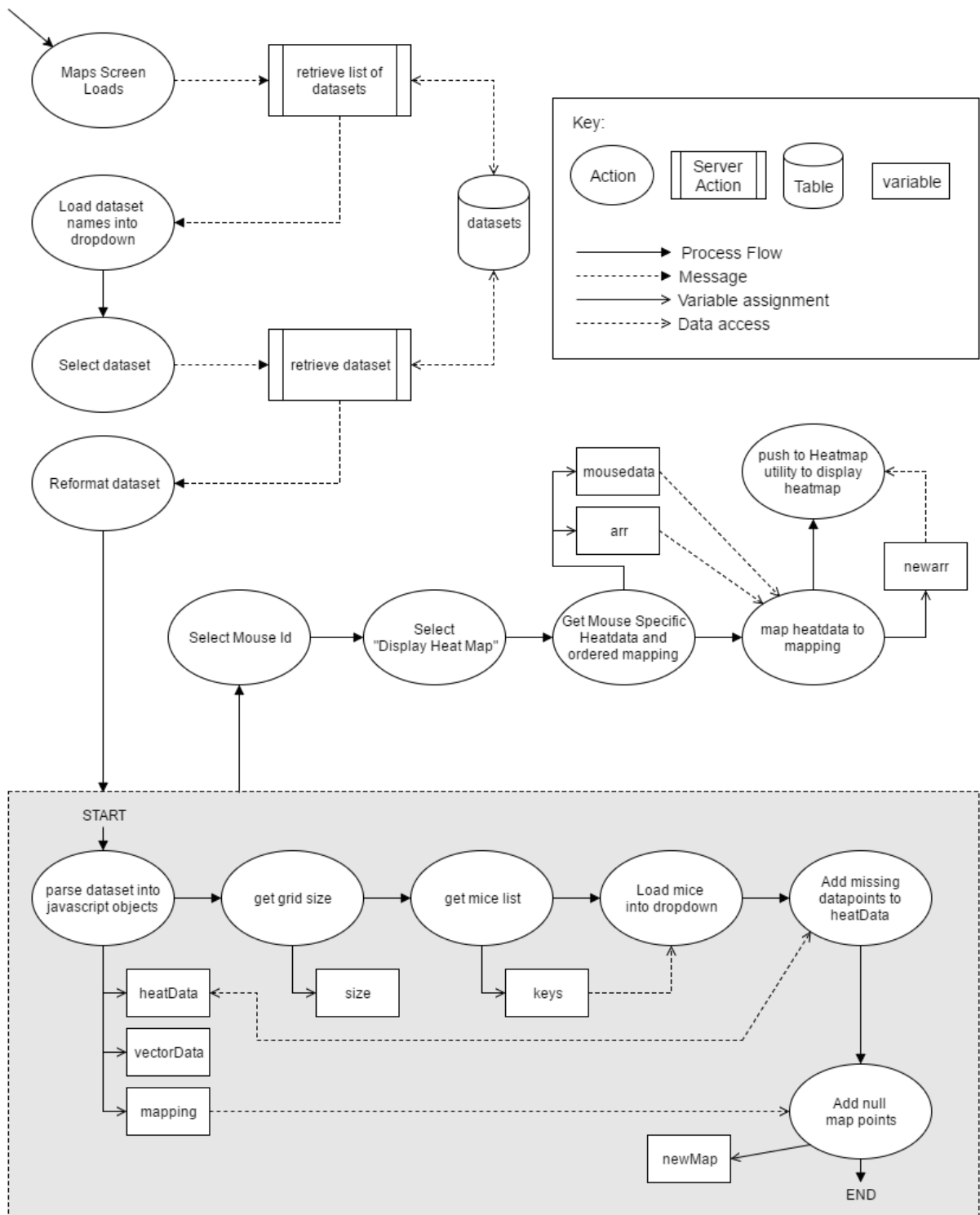


Figure 11: Heatmap Module Design Flow

### 3.3 Design Rationale

Model-View-Controller as a principle design philosophy was chosen because its intrinsic modularity will allow for easy distribution of implementation work (different individuals can work on different subsystems simultaneously) and more control as the system is developed through several iterations (subsequent features can be added without adversely affecting other previously-implemented subsystems).

Flask, Python and Postgresql were all selected because of familiarity for the developers. Though other architectures can be chosen to accomplish the same task, but the developers' familiarity with these systems and their API will allow more effort to be diverted to other tasks.

Likewise for Bootstrap, with which a compelling web site can be developed without excessive attention dedicated to overcoming otherwise minor implementation hurdles.

Digitalocean was chosen over Amazon Web Services for web hosting strictly because of costs. M-VU is not designed to accommodate a large swell of traffic (in nearly all instances, the software will be accessed by one user at a time). Therefore performance or access speed comparisons are not relevant. Digitalocean offered a much more affordable option than Amazon Web Services.

## 4. Data Design

### 4.1 Data Description

Data uploaded to the M-VU system will be in the form of a CSV file produced by Microsoft Excel. The Data Import and Validation subsystem will split this file line by line and insert parsed data lines into the M-VU database (see Figure 12).

Data recalled from the database via queries will be structured into a Python dictionary, which will be used to instantiate a new simulation object for the Mouse Simulation subsystem.

---

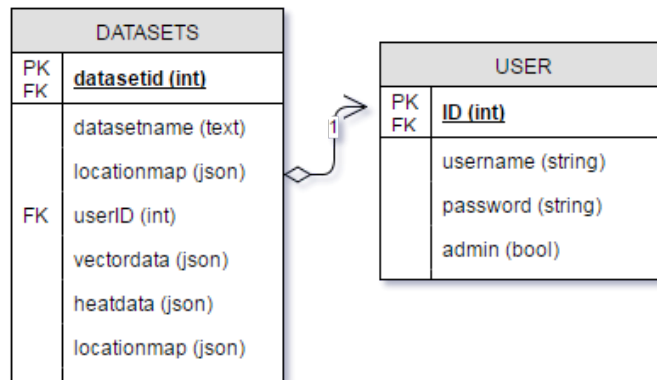


Figure 12: Entity Relationship Diagram (ERD) for M-VU Database

## 4.2 Data Dictionary

### 4.2.1 Dataset Table

Each entry in the Dataset table represents a complete Mouse Simulation, generated from the parsing of an uploaded file. The complete Vector Map, Heat Map, and Sensor Grid Map are compressed into JavaScript Object Notation (JSON) and inserted as complete entities.

**datasetID (PK):** An auto-incrementing integer which will identify an individual dataset.

**datasetname:** A user-defined name to help identify the data set during user interactions.

**uploadDate:** A date-type entry which will store the date the dataset was uploaded.

**userID (FK):** A integer to link the user who uploaded this particular dataset with the users table.

**vectordata:** A JSON object which stores the sensor reading-to-sensor reading movements of the subject. The object provides all necessary information for M-VU to reconstruct a vector map for this data set.

**heatdata:** A JSON object which stores the total time values for each sensor, providing all the data M-VU needs to reconstruct a heat map for this data set.

**locationmap:** A JSON object which defines how the sensor grid was aligned for this particular data. The locationmap is used by M-VU to orient either a Heat Map or Vector Map.

#### 4.2.2 User Table

The user table stores system users and log-in information.

**Admin:** A boolean to identify users with administration privileges.

**Id (PK, FK):** A unique identification number for this user, used to link to the Dataset Table to identify the user which uploaded a given dataset.

**Password:** The hashed password for the given user.

**Username:** The username for this user, to use for logging into the system.

## 5. Component Design

### Mouse Simulation subsystem

Load file

Parse file data

Initiate sim

For each subject:

    Initiate subject

For each time increment:

    Advance subject

    Update time stamps

If all subjects not finished, repeat

### Data Import and Validation subsystem

Open File

---

Skip First 27 lines // Static length of file header

For Each Line:

    Split Line by “;” char // values in data structure separated by “;”

    Assign each value to proper key

    Construct data dictionary for these values

    Add line data dictionary to master data dictionary

Return master data dictionary

## **File Upload subsystem**

User selects file for upload (Figure 25)

User clicks “submit” button

If the user has submitted a valid file then proceed

    File is sent to parser and the parsed data is stored in the database

    User is shown a message stating that the upload is successful

Else if the file is an invalid format

    The user is shown an error message stating that they have used an invalid file type

## **Login subsystem**

User inputs user name and password on the login page (see Figure 13)

User clicks “Submit” button

The system checks the validity of the username by querying the database

    If the username is not valid

        Reload login screen

        Display error message

The system checks the validity of the user's password by querying the database

    If the password is not valid

        Reload login screen

        Display error message

Load Index page

Display page header

## **Heatmap Module**

Map screen loads (see Figure 15)

    Retrieve dataset list

User selects dataset from list

    Retrieve dataset

        Reformat dataset

            Parse dataset into javascript objects

            Get grid size

            Get mouse id list

            Load mouse id list

            Add missing heatData data points

            Add null map points

User selects mouse id from list

User selects “Display Heat Map” button

    Map heat map data to dataset mapping

Pass mapped data to heatmap utility and display on screen

---



## 6. Human Interface Design

### 6.1 Overview of User Interface

The design of the User Interface is intended to provide a clean, simple, and modern look and feel to the user. With a focus on current technology, the design's main priority will be to up-to-date web browsers and computers, such as Chrome and Firefox.

When a user first arrives at the application's login screen (see Figure 13), they are presented with a simple panel with the application logo and name as well as the login form.

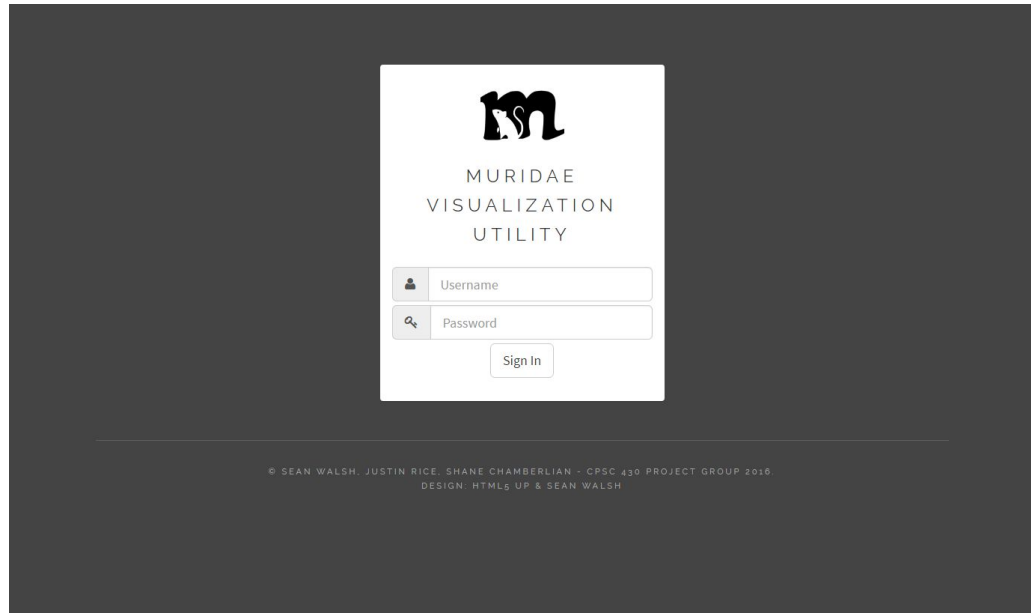
Upon entry, the user will be presented with a few options (see Figure 14). They are able to logout and return to the login screen or select one of three main application features - View Maps, Manage Data, Manage Users.

- **View Maps** will allow the user to select a data set and display various maps of selected data.
- **Manage Data** will allow the user to upload data sets, view them in table form, and delete them from the database.
- **Manage Users** will allow the user to modify user account information.

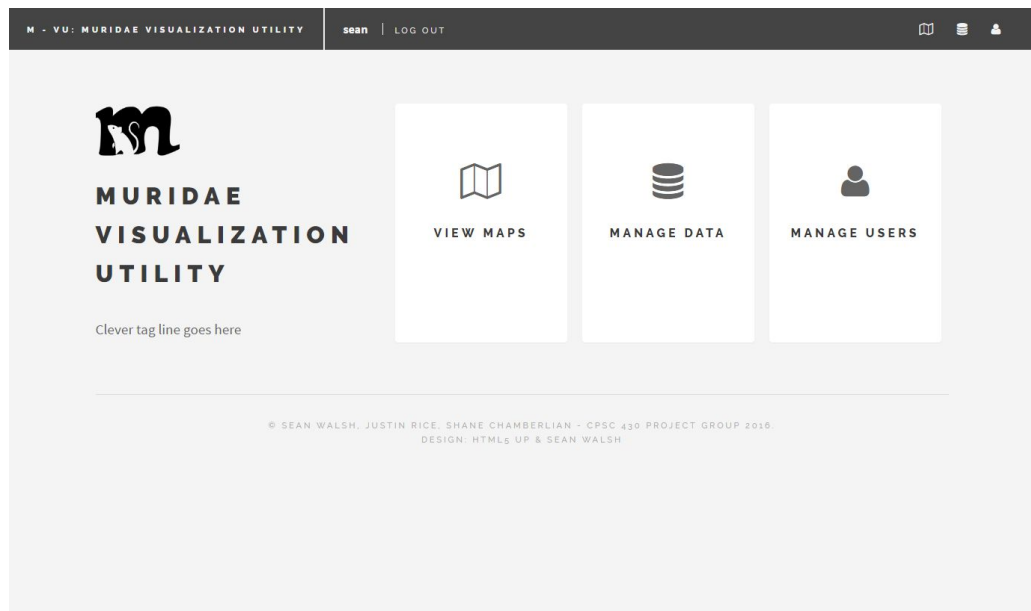
The overall structure of the application is created with modern web technologies including HTML5, CSS3, and Javascript. Special frameworks have been implemented as well - Bootstrap and JQuery,

---

## 6.2 Screen Images



*Figure 13: Login Screen*



*Figure 14: Main Application Screen*

---

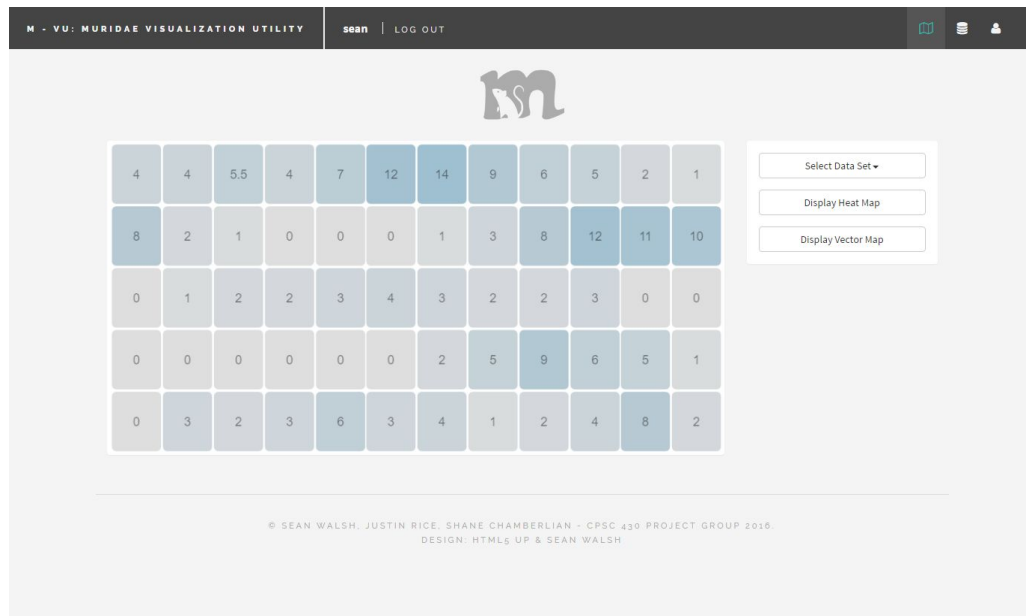


Figure 15: View Maps Screen

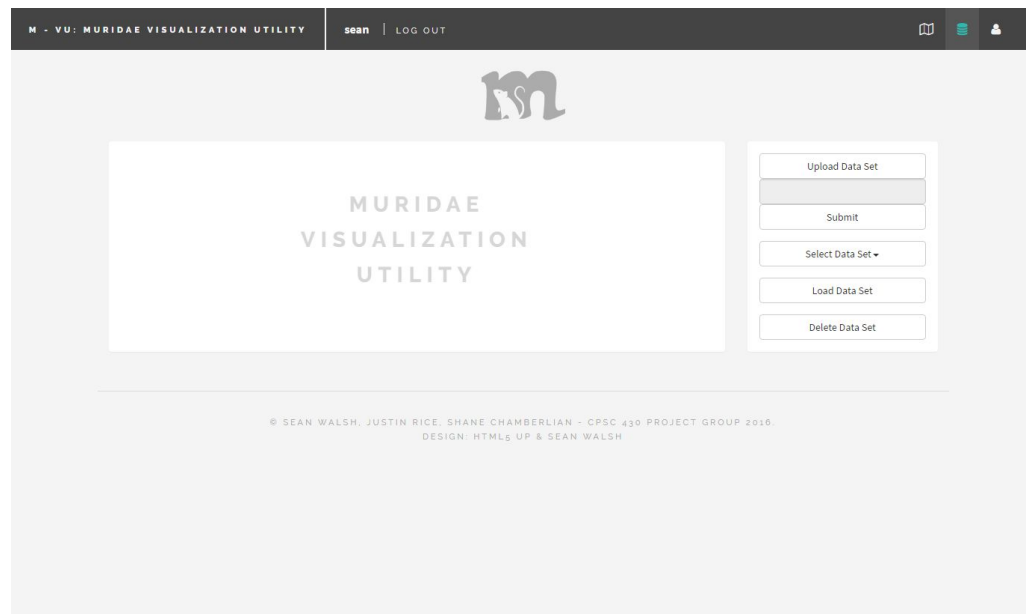


Figure 16: Manage Data Screen

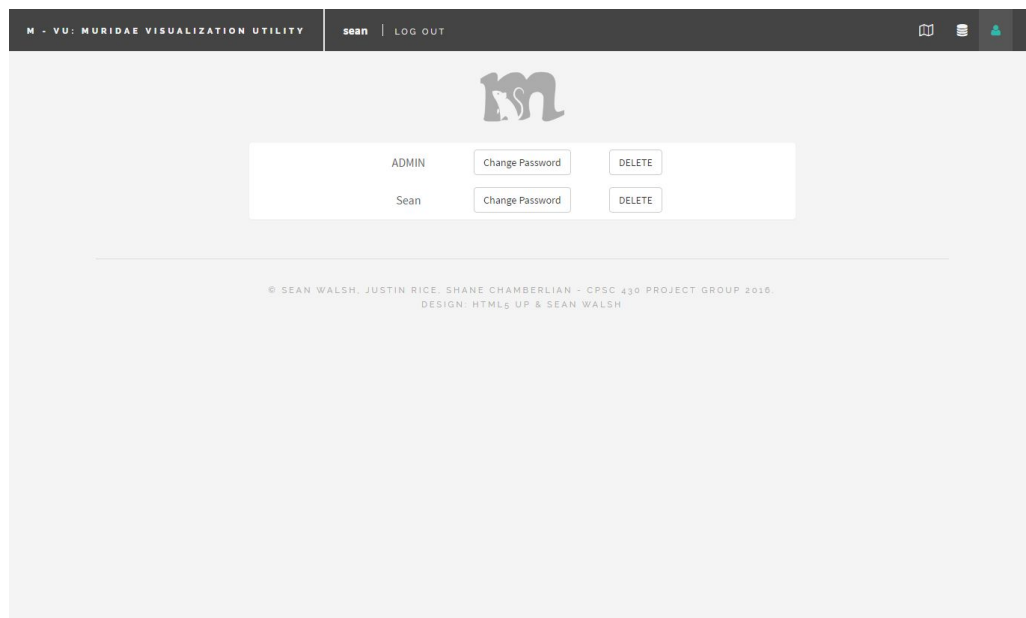


Figure 17: Manage Users Screen

## 6.3 Screen Objects and Actions

**Login:** log into the application

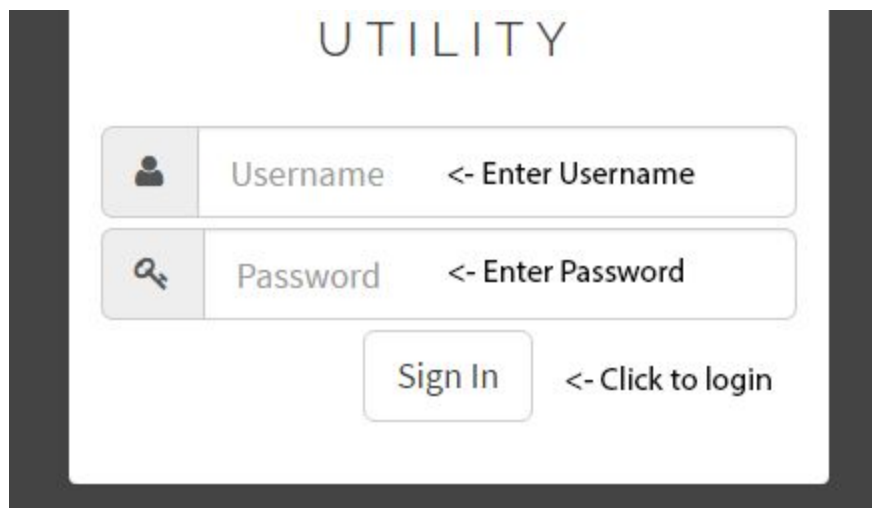


Figure 18: How to login

**Header:** Click the Application name to return to home page; Click LOG OUT to logout and return to login screen.

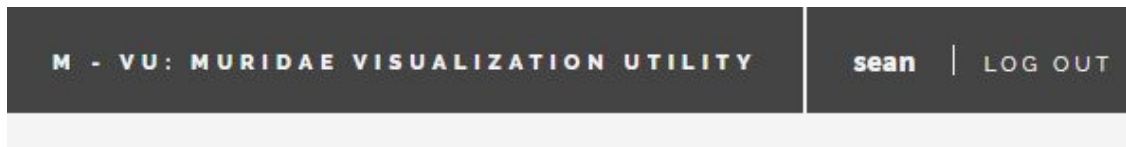


Figure 19: Header

**Main Page Navigation:** Click the corresponding button to go to that application page.

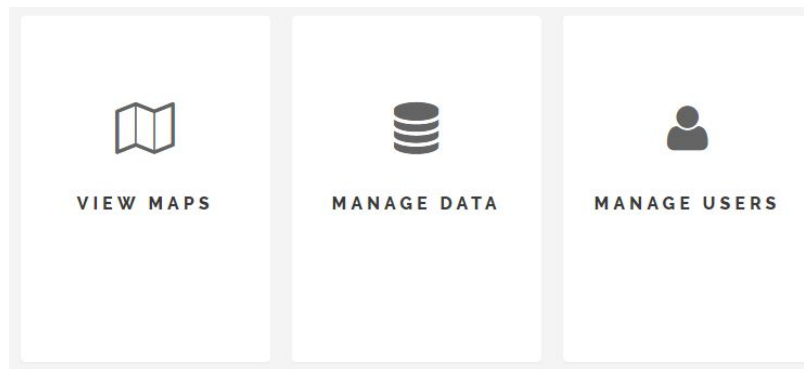


Figure 20: Main Page Navigation

**Header Navigation:** Click the corresponding button to go to that application page.

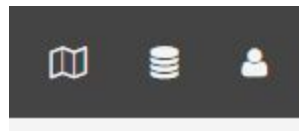


Figure 21: Header Navigation

**Return-to-Main-Page Icon:** click the muted application logo to return to the main page.



Figure 22: Return to Main Page button

**View Map panel:** various View Map options.

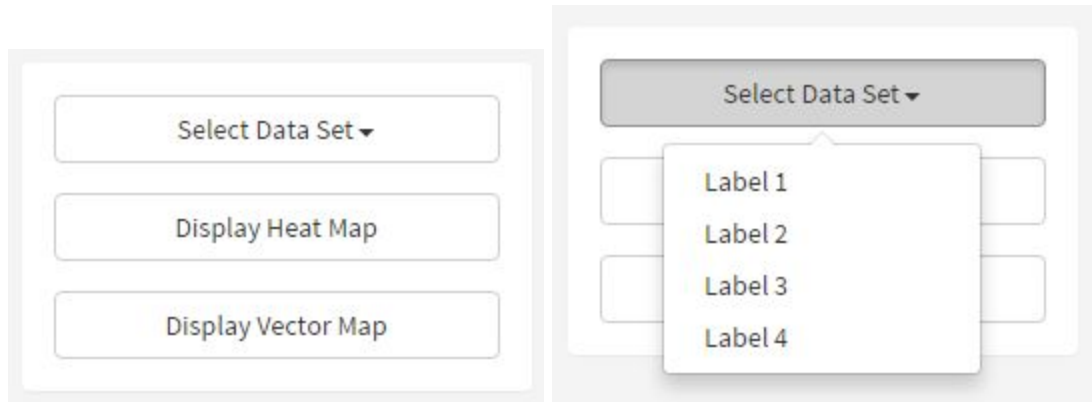


Figure 23: (Left) View Map panel, (Right) 'Select Data Set' dropdown active

**Manage Data panel:** various Manage Data options

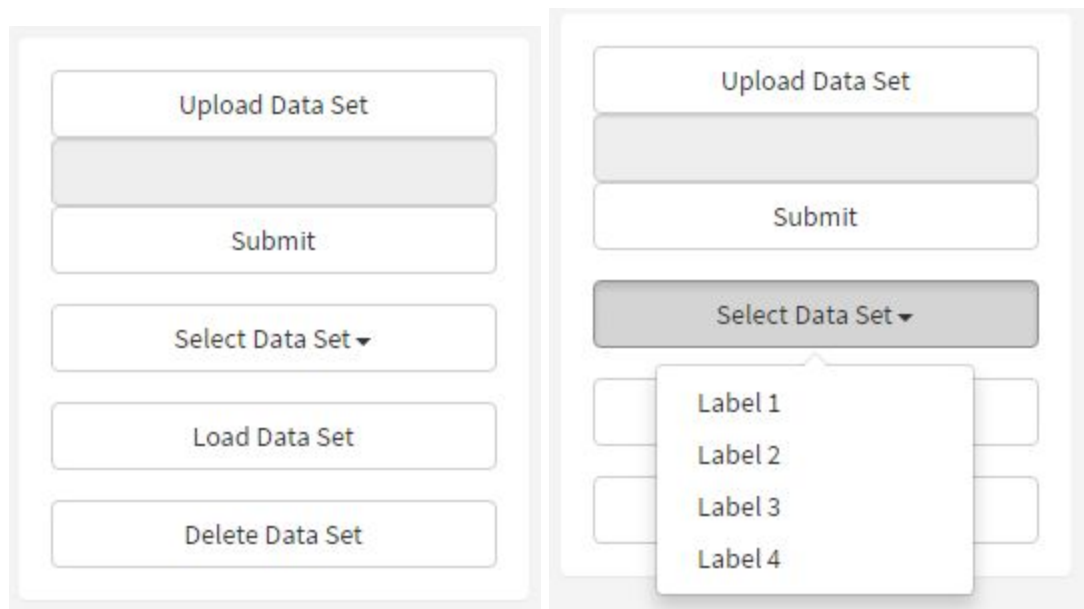


Figure 24: (Left) Manage Data panel, (Right) 'Select Data Set' dropdown active

**Upload Data Set feature:** When the user clicks 'Upload Data Set', a file chooser loads and allows the user to select a file. The filename will display above the 'Submit' button once selected.

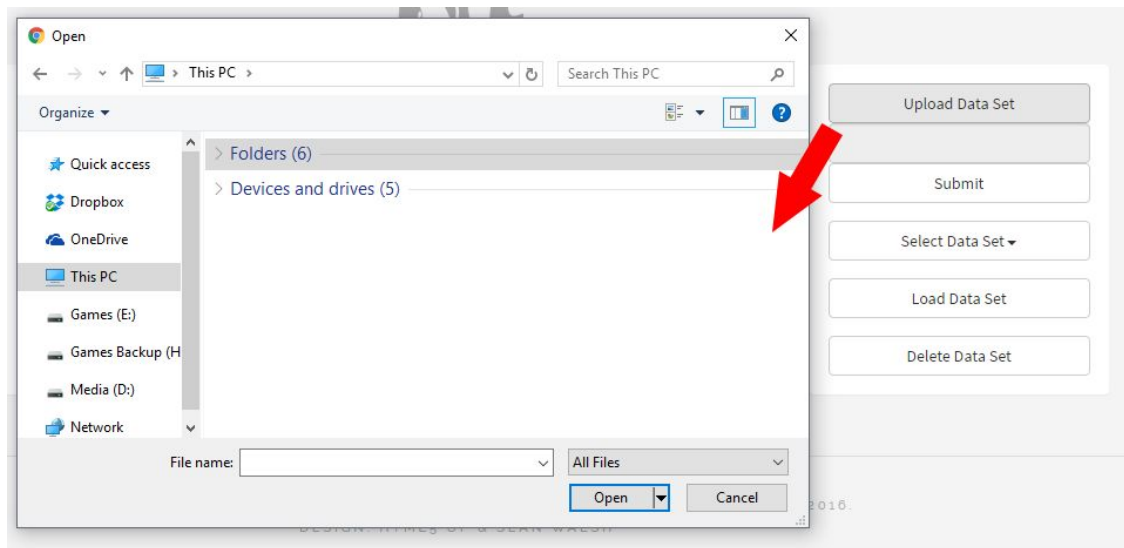


Figure 25: Clicking 'Upload Data Set' will display a file chooser

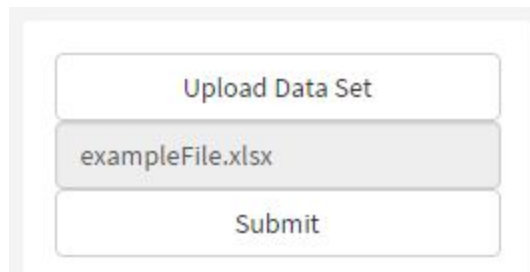
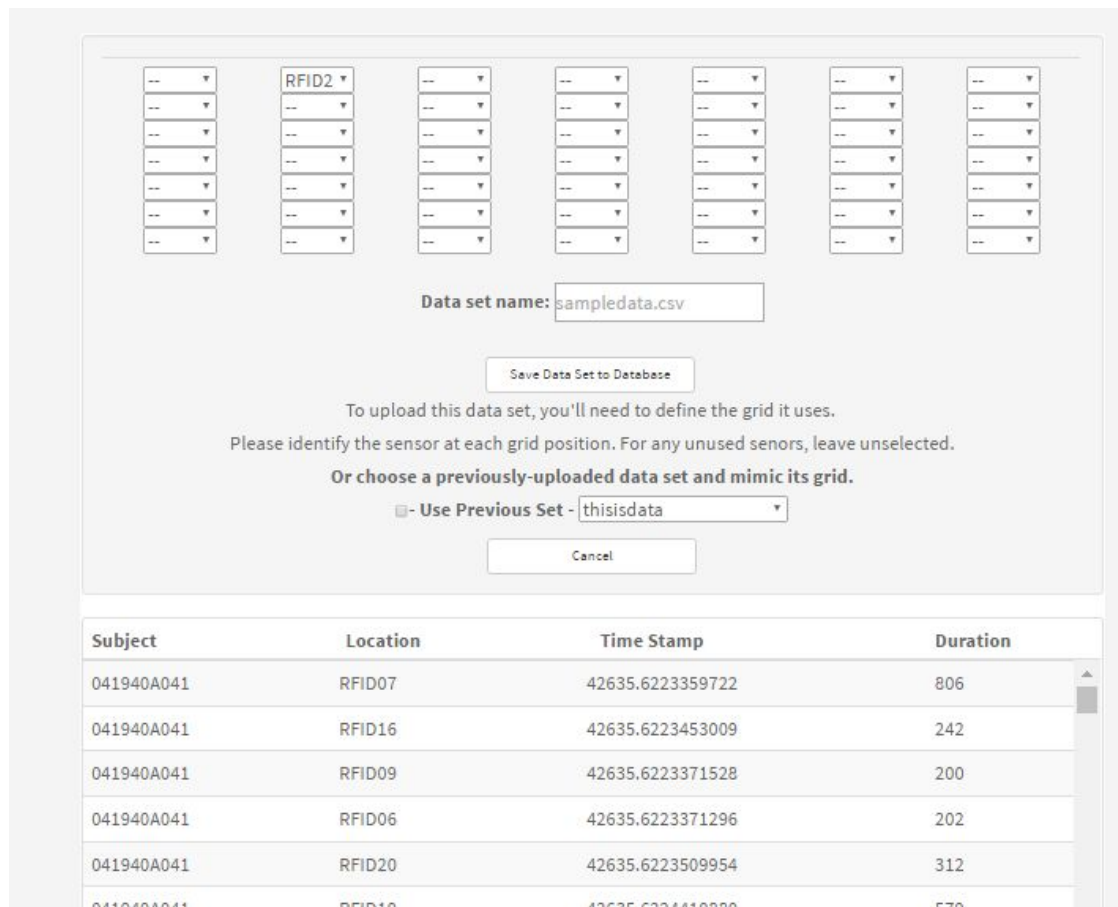


Figure 26: selected filename displayed on screen

**Define Sensor Grid Panel:** With a file selected for upload, the data is displayed in raw format, and the user is given selection choices to either define a sensor grid for this data set, or to mimic the sensor grid used with a previous data set.

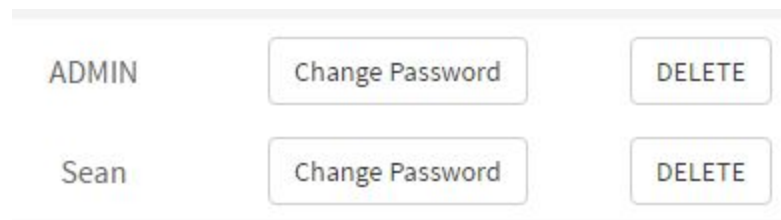


The panel contains a 7x7 grid of dropdown menus. The second dropdown in the first row is selected and shows 'RFID2'. Below the grid is a text input field labeled 'Data set name:' containing 'sampledata.csv'. A 'Save Data Set to Database' button is below the input field. Below the button is instructional text: 'To upload this data set, you'll need to define the grid it uses. Please identify the sensor at each grid position. For any unused sensors, leave unselected. Or choose a previously-uploaded data set and mimic its grid.' Below this text is a checkbox labeled 'Use Previous Set' which is checked, followed by a dropdown menu showing 'thisisdata'. A 'Cancel' button is at the bottom of the panel.

Subject	Location	Time Stamp	Duration
041940A041	RFID07	42635.6223359722	806
041940A041	RFID16	42635.6223453009	242
041940A041	RFID09	42635.6223371528	200
041940A041	RFID06	42635.6223371296	202
041940A041	RFID20	42635.6223509954	312
041940A041	RFID10	42635.62234410880	570

Figure 27: Define Sensor Grid panel

**Manage Users panel:** This panel will allow the user to change a user's password and the ability to delete the user.



The panel displays a list of users with their names and associated action buttons. The first user is 'ADMIN' and the second is 'Sean'. Each name is followed by a 'Change Password' button and a 'DELETE' button.

ADMIN	Change Password	DELETE
Sean	Change Password	DELETE

Figure 28: Manage Users panel



## 7. Requirements Matrix

	SDD requirement	SRS requirement	Code	Testing
User Login Page	3.2.6, 4.2.2	3.1.1	index.html	
User Management Page	3.2.6, 4.2.2	3.1.2	users.html, create_account.html, change_password.html , delete_account.html	
Data Administration Page	3.2.1, 3.2.2, 3.2.5, 3.2.3, 3.2.4	3.1.3	mouseDB.sql, parse.py	
Upload Data Page	3.2.1, 3.2.2, 3.2.5, 3.2.3, 3.2.4	3.1.4	Loadeddata.html, Data.html, Mice.py, mouse.py, sim.py,	
View Map Page	3.2.1, 3.2.2, 3.2.3, 3.2.7	3.1.5	maps.html, controller.js	
Hardware Interface	3.3	3.2	server.py	
Software Interfaces	3.3	3.3	Server.py, index.html, layout.html	
Performance Requirements	3.3	5.1	server.py	
Security Requirements	3.3	5.2	server.py	

Figure 29: Requirements Matrix reflects the SRS requirements

## 8. Appendices

### Appendix A: A valid data file

Following is an example of a valid data file. These files will be uploaded to M-VU, parsed into their individual data lines, and uploaded to the M-VU database.

42635.6222972338;041940A1C3;unknown;RFID3;513;;;3;;;;;
42635.6222903125;041940C822;unknown;RFID10;1253;;;4;;;;;
42635.6223064468;041940A1C3;unknown;RFID4;200;;;2;;;;;
42635.6223198032;041940A1C3;unknown;RFID3;627;;;2;;;;;
42635.6223241435;041940A1C3;unknown;RFID4;723;;;3;;;;;
42635.6223331019;0419408444;unknown;RFID16;1052;;;1;;;;;
42635.6223359722;041940A041;unknown;RFID7;806;;;1;;;;;
42635.6223319329;0419408444;unknown;RFID18;203;;;1;;;;;
42635.6223331019;0419408444;unknown;RFID20;248;;;1;;;;;
42635.6223453009;041940A041;unknown;RFID16;242;;;1;;;;;
42635.6223371528;041940A041;unknown;RFID9;200;;;1;;;;;
42635.6223371296;041940A041;unknown;RFID6;202;;;1;;;;;
42635.6223371296;041940CE89;unknown;RFID13;202;;;1;;;;;
42635.6223388773;0419408444;unknown;RFID19;198;;;1;;;;;
42635.6223353935;041940CE89;unknown;RFID15;550;;;2;;;;;
42635.6223394676;0419408444;unknown;RFID24;198;;;1;;;;;
42635.6223342824;0418CC1D10;unknown;RFID11;848;;;2;;;;;
42635.6223400694;0419408444;unknown;RFID23;348;;;2;;;;;
42635.6223429282;0419408444;unknown;RFID21;1134;;;1;;;;;
42635.6223192014;041940C822;unknown;RFID10;2365;;;5;;;;;
42635.6223446875;041940C822;unknown;RFID3;232;;;1;;;;;
42635.6223453357;041940C822;unknown;RFID7;176;;;1;;;;;
42635.6223481134;0418CC1D10;unknown;RFID16;186;;;1;;;;;
42635.6223365509;041940CE89;unknown;RFID17;1373;;;4;;;;;
42635.6223509954;041940A041;unknown;RFID20;312;;;1;;;;;
42635.6223546065;041940A1C3;unknown;RFID6;188;;;1;;;;;
42635.6223575;041940CE89;unknown;RFID23;188;;;1;;;;;
42635.6223726968;041940CE89;unknown;RFID17;203;;;2;;;;;
42635.6223811921;041940CE89;unknown;RFID15;188;;;1;;;;;

Figure 30: A sample of M-VU data for upload.