MAKESENSE SYSTEM MANUAL

TABLE OF CONTENTS

1. HARDWARE

1.1 STAMPS

The McPhillips Stamps are programmed in C using ContikiOS. The entire codebase for ContikiOS can be found in the hardware code repository[1]. Compiling and loading code onto the stamps must be done through the Instant Contiki virtual machine[2]. This is an Ubuntu virtual machine that has built in scripts to make the process of building and deploying ContikiOS to hardware easier.

The code for the Stamps in the MakeSense system is contained in two files, collectSensor.c[3] for the sensor node code and collectGateway.c [4] for the gateway node code. These code files cannot be moved from the directory in which they currently reside as compilation relies on a series of make files which expect certain files to be in certain directories.

1.1.2 DEPLOYING CODE TO STAMPS

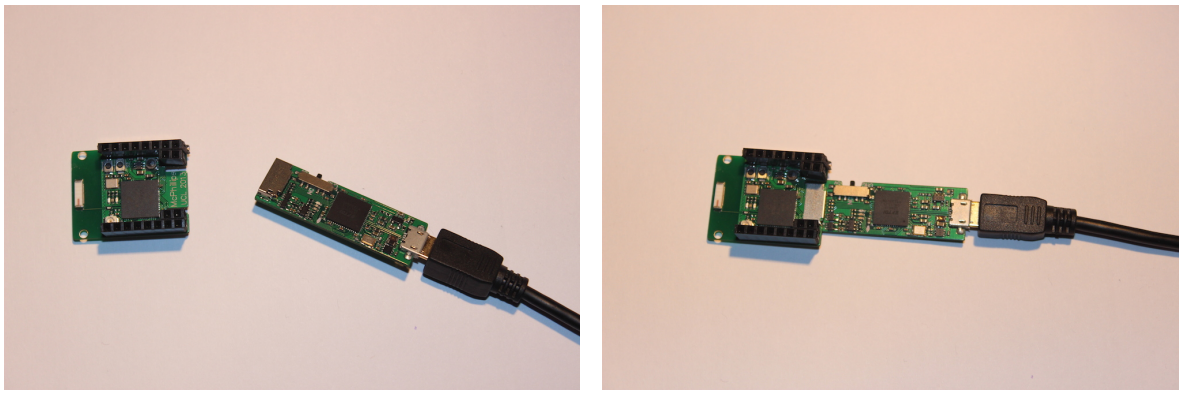


FIG. 1

To deploy code to the stamps, they must first be plugged in using the serial adaptor as shown in Fig. 1.

NOTE: The stamps are not powered by the serial adaptor and must therefore be plugged in to batteries or another source of power to load code.

For the gateway Stamp, navigate to the directory where collectGateway.c is stored in the terminal and run the command

```
make collectGateway.upload
```

This compiles the code and loads it onto the Stamp hardware.

---

[1] https://github.com/andrewgrex/MethHardware
[2] http://sourceforge.net/projects/contiki/files/Instant%20Contiki/
[3] https://github.com/andrewgrex/MethHardware/blob/master/examples/OrisenStamp/collectSensor.c
[4] https://github.com/andrewgrex/MethHardware/blob/master/examples/OrisenStamp/collectGateway.c

Similarly, for the sensor node Stamps, navigate to the directory where collectSensor.c is stored in the terminal and run the command

```
make collectSensor.upload
```

1.1.3 COLLECTSENSOR.C

This code has two functions, read sensor data from the Engduino over the Stamp's serial port and sending the data using the collect networking mechanism.

The code consists of two protothreads[5], one to continuously read the serial port and another to transmit the data.

A full data point has been received when the serial read thread detects a newline '\n' character. It then calls the collect send thread. This thread prepares the complete data for the sensor:

Timestamp: This is the time in seconds since the gateway booted up, the web application uses timestamps in UNIX time format (seconds since the epoch). Full UNIX timestamps are calculated on the Raspberry Pi using the a function of the time the first data was received from a sensor and the difference in time between the seconds since boot up of the first reading and the current reading being processed

Sensor Address: This is the sensors RIME address, the address it uses within the network to identify itself. This address is unique for every Stamp.

Ontology Code: This is currently hardcoded for each Stamp, it denotes the type of data the Stamp is transmitting (in the prototype this is fixed to '1' for light sensor data).

Data Point: The data from the Engduino that has just been read

This data is collated into a string and sent using the `collect_send()` method[6]

1.1.4 COLLECTGATEWAY.C

When it starts up, the Gateway code calls the `collect_set_sink()` method which designates it as the node towards which all data in the mesh network will flow towards. This thread then goes into a loop to wait for data.

When data is received, the `recv()` function is called. This function is very simple, it simply prints out the data received to the serial pins on the Stamp. The string is already formatted so that it is ready to be processed by the Pi.

1.2 RASPBERRY PI

The Raspberry Pi's function is to upload sensor data and information to the MakeSense database. When it boots up it automatically runs the piGateway.py script[7].

[5] http://dak664.github.io/contiki-doxygen/a01676.html
[6] https://github.com/contiki-os/contiki/blob/master/core/net/rime/collect.c
[7] https://github.com/andrewgrex/MethHardware/blob/master/piGateway.py

This script makes use of two Python modules (besides Python's standard modules): PySerial[8] and MySQLdb[9]. PySerial is used to access and read the serial connection on the Pi's GPIO pins (to which the Stamp is connected).

The code opens a connection to the serial connection with the `serial.Serial()` constructor. It then opens a connection to the database with the `MySQLdb.connect()` method.

The code then generates the network ID for the sensor network, this is simply the MAC address of the Pi which is read from a file using the `getserial()` method defined at the top of the code.

A loop is then started where the Pi continuously reads its serial connection until it detects a full data string has been received. It then splits up the string to process the data. The timestamp is calculated by checking the sensor ID against the sensor ID's in a hash table to fetch the first timestamp received by that sensor and calculate the difference in time and thus generate a UNIX timestamp.

A database query is made to see if the sensor read currently exists in the database, if it doesn't , it is added.

The final step is then to insert all the sensor data into the database.

1.3 ENGDUINO

The Engduino runs a simple script[10] to read its light sensor and print the reading to its serial pins. The script is compiled and loaded onto the Engduino using the special Engduino version of the Arduino IDE[11].

1.4 HARDWARE CONFIGURATION

The following is information about how the hardware is wired together:

1.4.1 SENSOR NODES

There are 3 connections between the Engduino and Stamp in the sensor nodes. The Stamp is powered using the Engduino's battery, hence there is a wire for ground and a wire for power in. The third wire connects the Engduino's serial TX to the Stamp's serial RX.

A picture of this setup can be seen in Fig.3.

A diagram of the pin layout on the Engduino and Stamp can be seen in Fig.2.

[8] http://pyserial.sourceforge.net/
[9] http://mysql-python.sourceforge.net/MySQLdb.html
[10] https://github.com/andrewgrex/MethHardware/blob/master/Engduino%20Code/EngduinoSerial1Test/EngduinoSerial1Test.ino
[11] http://www.engduino.org/

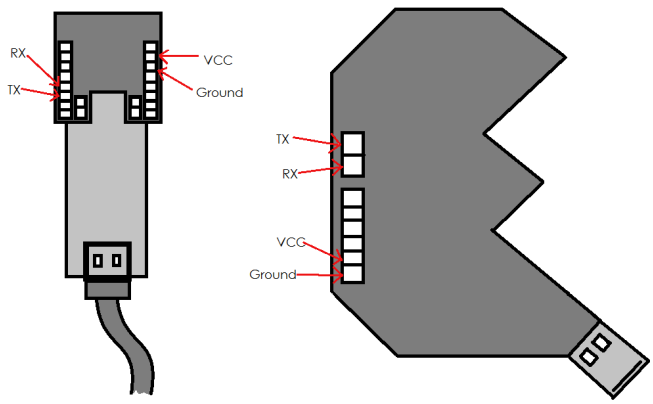


FIG. 2

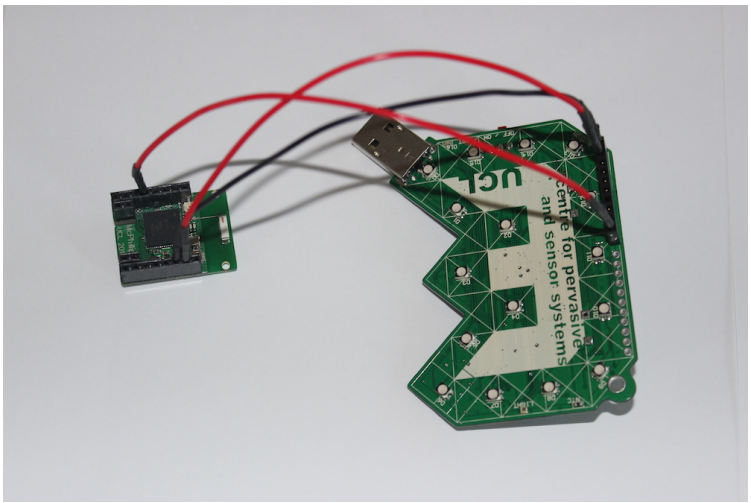


FIG. 3

1.4.2 Gateway Platform

There are actually two gateway platforms, a USB based solution that uses any computer as the main data processor, or the aforementioned Raspberry Pi.

To use any machine as a gateway, a Stamp is attached to the serial to USB converter and powered with batteries as shown in Fig.4. Then the serialLinkDB.py[12] code is run on the computer to which the stamp is connected.

Note: The hard coded port in the serial connection constructor must be changed to match the correct USB port on your system. The name of the port the Stamp is connected to can be found using the PortTest.py[13] script.

---

12 https://github.com/andrewgrex/MethHardware/blob/master/SerialLinkDB.py

13 https://github.com/andrewgrex/MethHardware/blob/master/PortTest.py

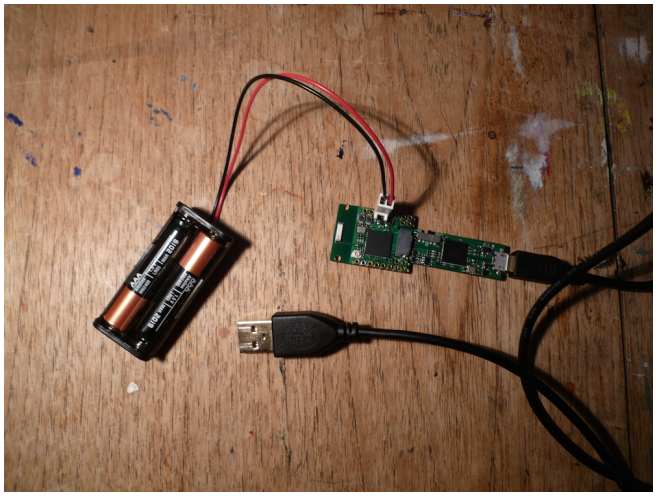


FIG. 4

The main gateway platform product consists of a Raspberry Pi with a Stamp wired to its GPIO pins. The Pi is connected to the internet via Ethernet. The Pi is powered from the mains and the Stamp is powered via the Pi's GPIO pins. The configuration is shown in Fig.5.

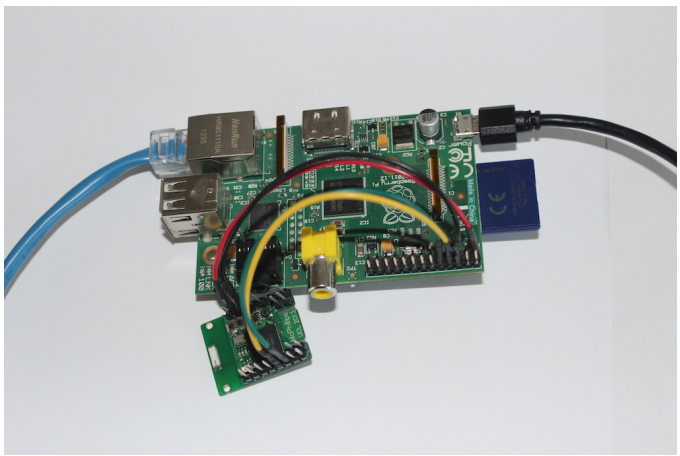
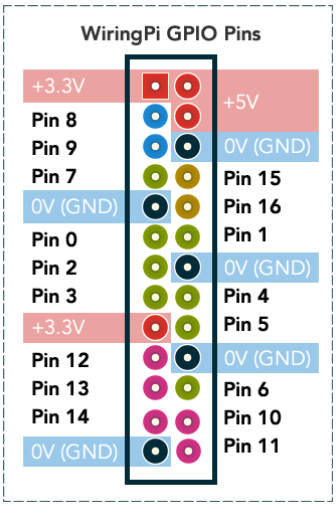


FIG. 5



The Pi's +3.3V pin (see Fig.6) is wired to the Stamp's VCC pin (Fig.2). The 0V GND pin is wired to the Stamps GND pin. Pin 16 on the Pi is the Serial RX and is hence connected to the Stamp's TX and Pin 15 on the Pi is the Serial TX and hence connected to the Stamp's RX.

FIG.6
SOURCE: HTTP://PI.GADGETOID.COM/WIRINGPI-GPIO-PINOUT.PNG?V=1

2. DATABASE

2.1 USER MANAGEMENT

A user account based system is necessary in order to allow users to set up sensor networks for their own, specific needs. The web application has been made in conjunction with the open source, user management PHP-framework called Usercake[14] to achieve this.

The Usercake framework files are in the same directory as other pages in the web application.

The following pages are accessible to all users:

- Registration page (*register.php*)
- Login Page (*login.php*)
- User Settings Page (*user_settings.php*)
- Logout (*logout.ph*p)
- Resend email activation page (*resend_activation.php*)
- Forgot password page (*forgot-password.php*)

The following are only accessible to administrators:

- Configuration page (*admin_configuration.php*)
- Permission group management pages (admin_*permissions.php*)
- User management pages (*admin_users.php*)
- Page management pages (*admin_pages.php*)

2.1.1 USER INFO IN THE DATABASE

The database connection details are location in *models/db_settings.php*. All pages should call '*models/config.php*' . This file connects with the database and initiates all of Usercake's functions.

The corresponding user management tables in our database are as follows:

- uc_configuration
  - Standard configuration information
- uc_pages
  - A list of pages and the corresponding privacy options.
  - '0' denotes available to all
  - '1' denotes available only to specified users
- uc_permissions
  - A list showing each type of user, for example Administrator or logged in member.
- uc_permissions_page_matches
  - A list showing which types of users have access to which pages.
- uc_users
  - A list of users with corresponding information. Passwords have been encrypted using salt (25 char) with an SHA2 hash.
- uc_user_permission_matches
  - A list of users matching to a permission tag.
  - '1' denotes a standard registered user
  - '2' denotes an administrator

[14] http://usercake.com/

2.1.2 KEY FUNCTIONS

There are three basic functions required to protect pages:

(1) In order to secure individual pages. Using this allows an administrator to control permissions for each page without having to directly edit the database.

```
securePage($_SERVER['PHP_SELF']);
```

(2) The following function checks whether or not a user has been granted a certain permission level. It's useful if there happen to be features on a page which should only be available to specific user types.

```
$loggedInUser->checkPermission(ARRAY);
```

(3) This function checks whether not users are logged in.

```
isUserLoggedIn();
```

2.1 SENSOR DATA AND INFORMATION

There are 5 main entities involved in the sensor network data as displayed in the entity relationship diagram in Fig.7

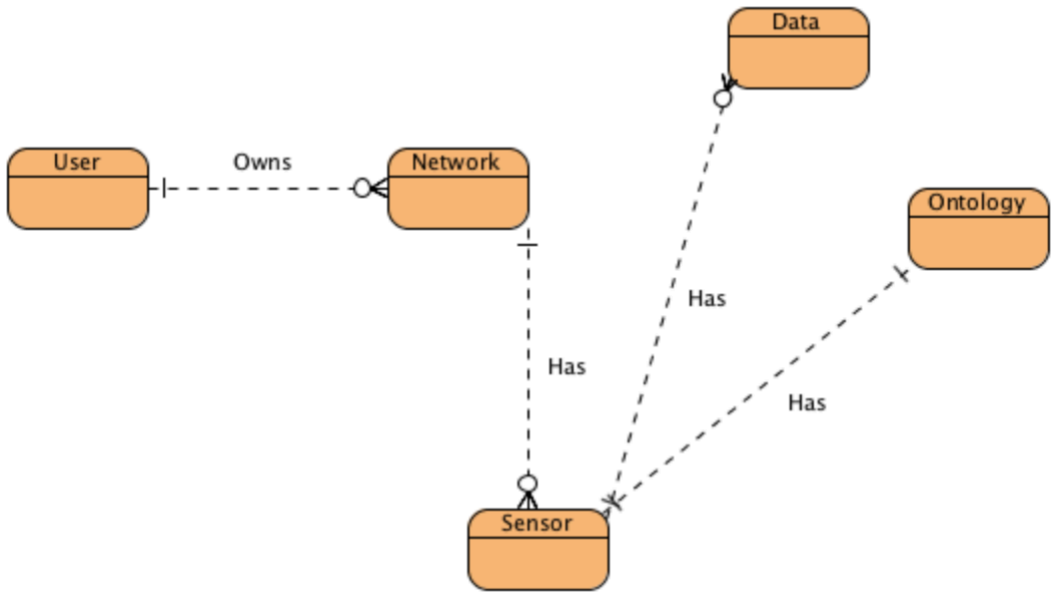


FIG. 7

This entity relationship resulted in the tables shown in Fig.8

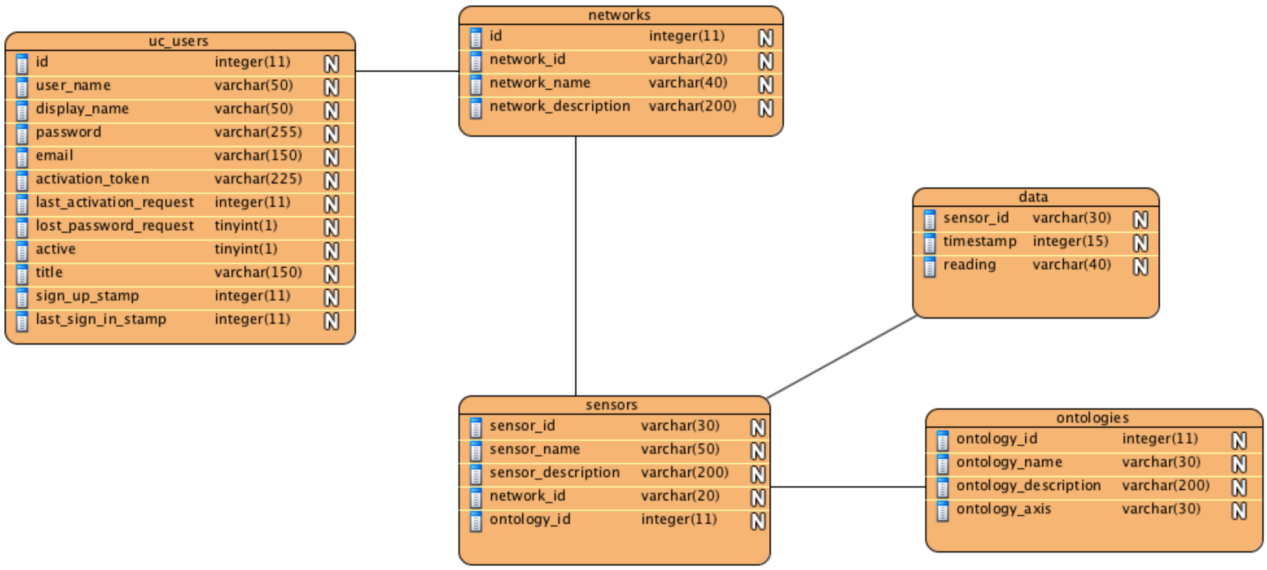


FIG. 8

The id value for each user account generated by Usercake and stored in the uc_users table is used to link sensor networks to an account.

The network_id is the MAC address of the Pi being used as a gateway for a network. This is unique for every Pi and thus suitable for use as a primary key for the networks . The primary key for the sensors is the sensor_id which is the RIME address of each sensor, these are also unique for every Stamp therefore suitable for use as a primary key. The ontology_id acts as the primary key for the ontologies table, hence if adding functionality where an ontology is added to the database, the code must catch and deal with possible exceptions if an attempt to add an existing ontology_id is made. Finally, the primary key for the data table is a composite of the sensor_id and timestamp, as no sensor can send more than one data point at a time.

3. Web Application

The web application consists of a PHP backend which connects to the database. The structure of the application is a lose Model View Controller architecture. Pages are rendered using HTML5, CSS and JQuery echoed from PHP scripts linked to the usercake architecture to check user permissions.

The database connection is handled by the datamanager.php[15] class. An object of this class is instantiated in each script which receives post requests from the user interface to fetch or input data from the database.

The current structure of the web application is a mixture of direct datamanager access from user interface related scripts and post requests (Fig.9). The aim is to move to a more logical structure as shown in (Fig.10).

[15] https://github.com/scarrobin/meth-web-app/blob/master/datamanager.php

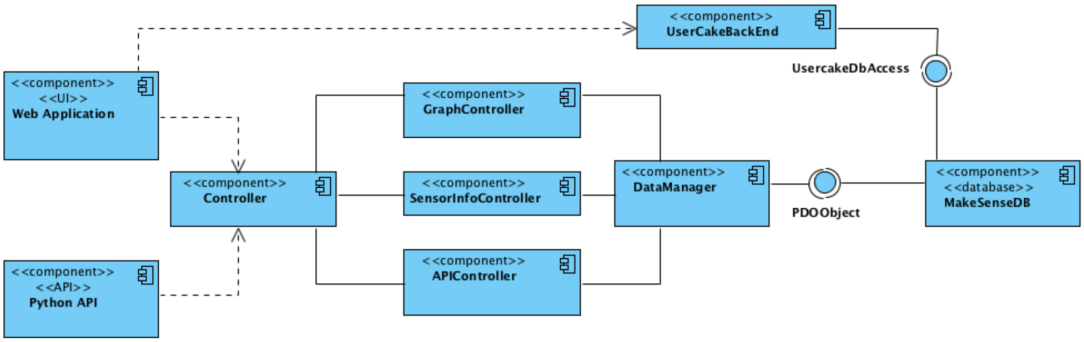


FIG. 9

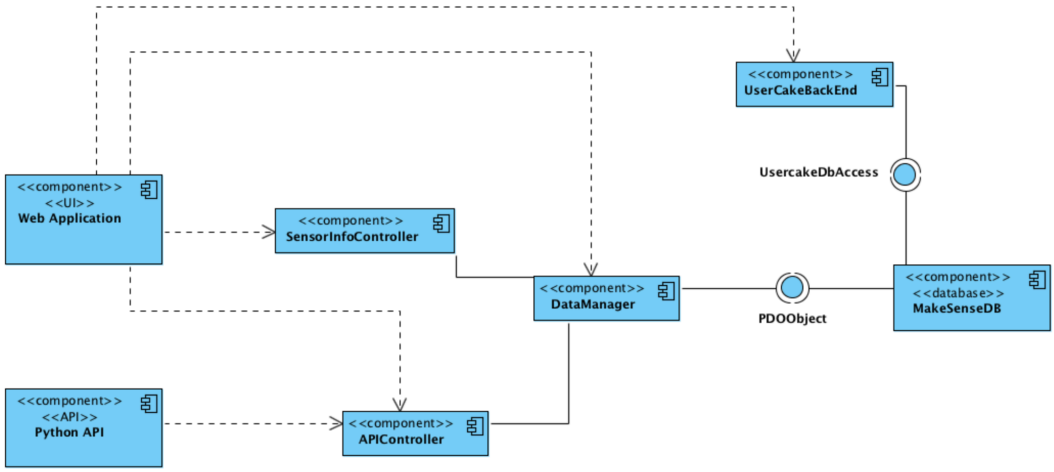


FIG. 10

3.1 Graphing

The graphing is handled by the graphdata.php[16]. This instantiates a data model object to fetch the required data from the database. The sensors for which data is required for are passed to the script from an ajax request in the `loadGraph()` method from the graph.php[17] page.

The graphdata.php script creates the Highcharts[18] graph code and returns it to the graph.php page for rendering. The code can create graphs with multiple sensors or a single sensor.

3.2 Sensor and Network Information

The selection UI elements are generated using various scripts. The network list is generated directly from the datamanager.php. The sensor list is generated using sensorlist.php[19] and the ontology list is generated using ontologylist.php[20].

[16] https://github.com/scarrobin/meth-web-app/blob/master/graphdata.php
[17] https://github.com/scarrobin/meth-web-app/blob/master/graph.php
[18] http://www.highcharts.com/
[19] https://github.com/scarrobin/meth-web-app/blob/master/sensorlist.php
[20] https://github.com/scarrobin/meth-web-app/blob/master/ontologylist.php

The sensor and network information is updated using POST requests to the sensorinfocontroller.php[21]. This is more in line with the desired structure of the web application and elements that do not get generated using this model need to be ported so that they do.

[21] https://github.com/scarrobin/meth-web-app/blob/master/sensorinfocontroller.php