# User Manual

**Team Horizon:**

Siong Ming Wong (John) – Team Contact

Michael Pettit

Rohan Kancherla

Chang Liu (Charles)

Xiaoqing Wu (Luo)

**FOR:**

College of Science Health and Engineering 2019 CSE3PRA/CSE3PRB

Department of Computer Science at Latrobe University

**Lecturer:** Dr. Torab Torabi

**Supervisor:** Majed Alwateer

# Contents

# 1.0 Introduction

## 1.1 Audience Description

The document mainly aims for the Client and general users of Bendigo City.

- **Client**: They have the capability to add and delete sensor nodes to the web app. They will store and extract the data via The Things Network.
- **Users**: The general public that will have access to the webpage and interact with the sensor markers, displaying information when interacted.

## 1.2 Applicability Statement

**Hardware**:
The hardware is powered by Arduino with digital microwave sensors, temperature sensors and oscillators to measure direction, speed, the temperature of the environment and differentiate between pedestrians and cyclists. Transmission of data is done wirelessly through a LoRa shield

**Arduino:**
Arduino is an open-source electronics platform based on easy-to-use hardware and software. It's intended for anyone making interactive projects.
https://www.arduino.cc/

**LoRa Shield:**
The LoRa Shield is a long-range transceiver on an Arduino shield form factor and based on Open source library. The Lora Shield allows the user to send data and reach extremely long ranges at low data-rates. It provides ultra-long range spread spectrum communication and high interference immunity whilst minimising current consumption

**Software:**

**Backend server**: The backend server is running MySQL and node-RED for data transmission.

**Web app**: The web is run by HTML5, Bootstrap CSS for styling display and scripts are powered by react+redux.js.

**Mapbox**: Open source platform for custom online maps. Provides a collection of tools for building Unity applications from real map data. The Mapbox web services APIs allow you to programmatically access Mapbox tools and services.
https://docs.mapbox.com/api/

**Visual Studio Code**: Visual Studio Code is a source-code editor developed by Microsoft for Windows, Linux and macOS. https://code.visualstudio.com/

**System Documentation: REST API**

## 1.3 Purpose Statement

The main purpose of the system is to count the number of pedestrians and cyclists using a path in Bendigo with a device and to display the data through a user interface.

Major Applications are:

- Provide information to the client about the number of people using the path.
- Display the data to the general public.
- Offer a sustainable and cost-effective alternative to current tracking technologies.

## 1.4 Document Usage Description

The user Document contains the following sections:

1. **Introduction**: Brief Introduction of the system and its purpose and users.

2. **Pedestrian Frequency Counter User Manual**: Services the site provides and a description of each.

3. **Installation Guide:** Steps to use device and user interface

4. **System Administrator Guide**: Necessary steps to maintain the system.

5. **Appendices**: Tables of subsidiary contents.

6. **Bibliography**: List of source materials.

7. **Glossary:** Definition of various terms used

8. **Index:** Alphabetical list of names, subjects with references to the section of the document in which they occur.

## 1.5 Conventions

The conventions that are used in this User Manual include headings, sizing, spacing, section separation, navigation in order to identify different parts in this document. The document was created using Word by using different categories and macros to help navigate throughout the document for creating content, appendix and table lists.

Every table and figure have a number and description so that its easy to follow and maintain.

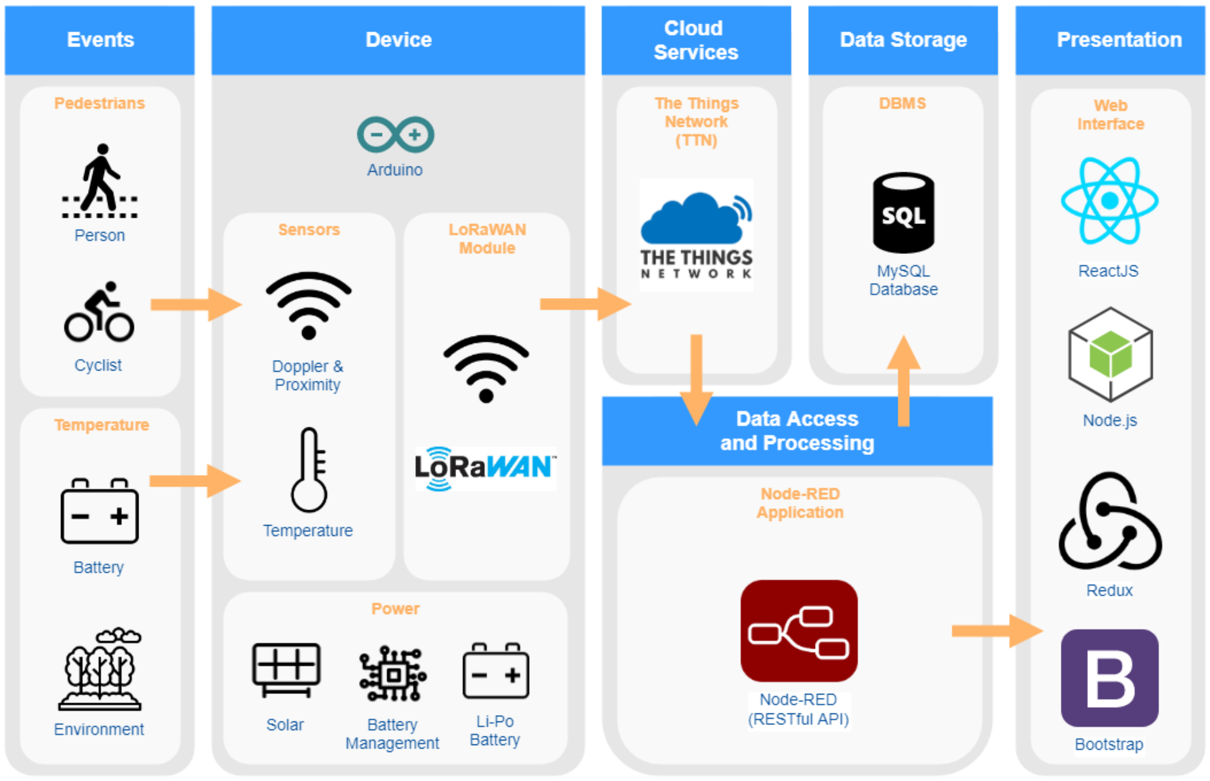
# 2.0 Introduction

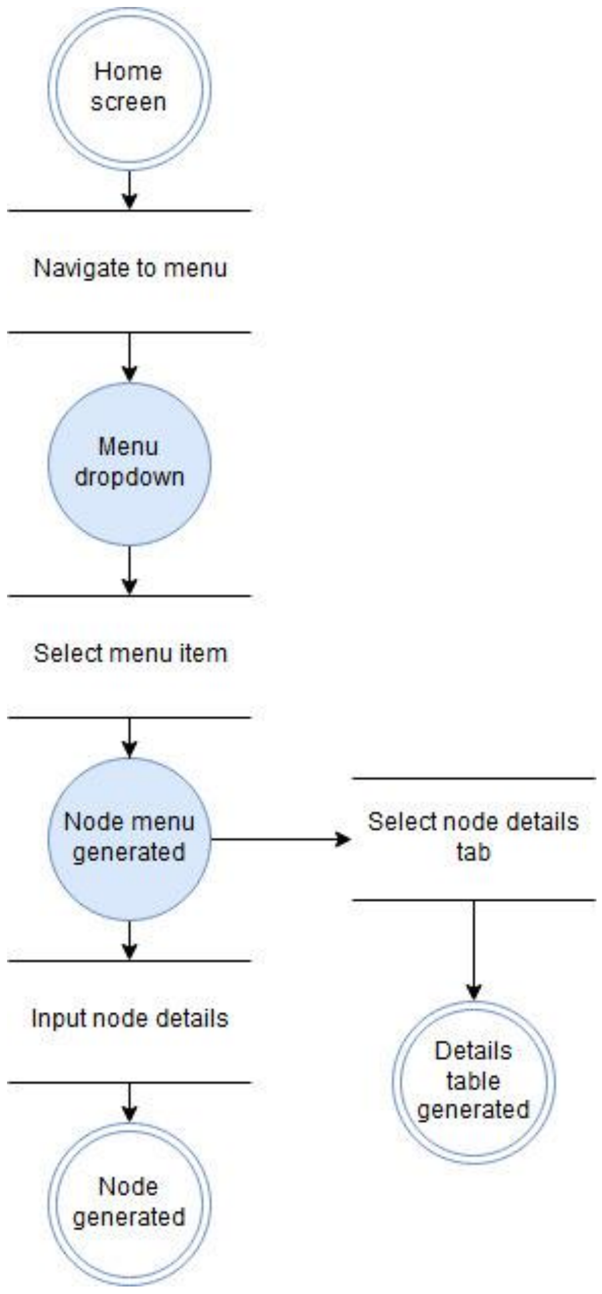
## 2.1 Pedestrian Frequency Counter Overview

Pedestrian Frequency Counter is a web app that has been developed to connect, track and visualize information from sensor nodes used in tracking passing pedestrians and cyclists of a path in the city of Bendigo. The user can select a sensor from a list to view information of the passing people. While viewing a node, a user can view the number of people and cyclists that has passed through the day along with a graph showcasing the frequency over the day.

The system is broken down into two parts:

- **Client**: They have the capability to add and delete sensor nodes to the web app. They will store and extract the data via The Things Network.
- **Users**: The general public that will have access to the webpage and interact with the sensor markers, displaying information when interacted.




Home screen
Navigate to menu
Menu dropdown
Select menu item
Node menu generated
Select node details tab
Input node details
Details table generated
Node generated

## 2.2 Getting Started

This section gives a brief overview of the steps needed for the different types of users (**Client, general user**) to get started and it is a quick overview for different actions a user can perform.

The general user has different options to get started with this website by using any smartphone, PC, laptop with internet access.

**User Actions:**

- Set location and check detail info of Bendigo map.
- Locate sensor nodes on map to identify where the sensor is.
- More information provided by clicking on sensor node to see paths used.
- Menu navbar drops when clicked on it to perform further functions.
- Clicking on key can provide colored information of the map.

**Client Actions:**

- Adding a sensor node to the database for it to display on map.
- Specifically know the number of sensors in a map by clicking on sensors.

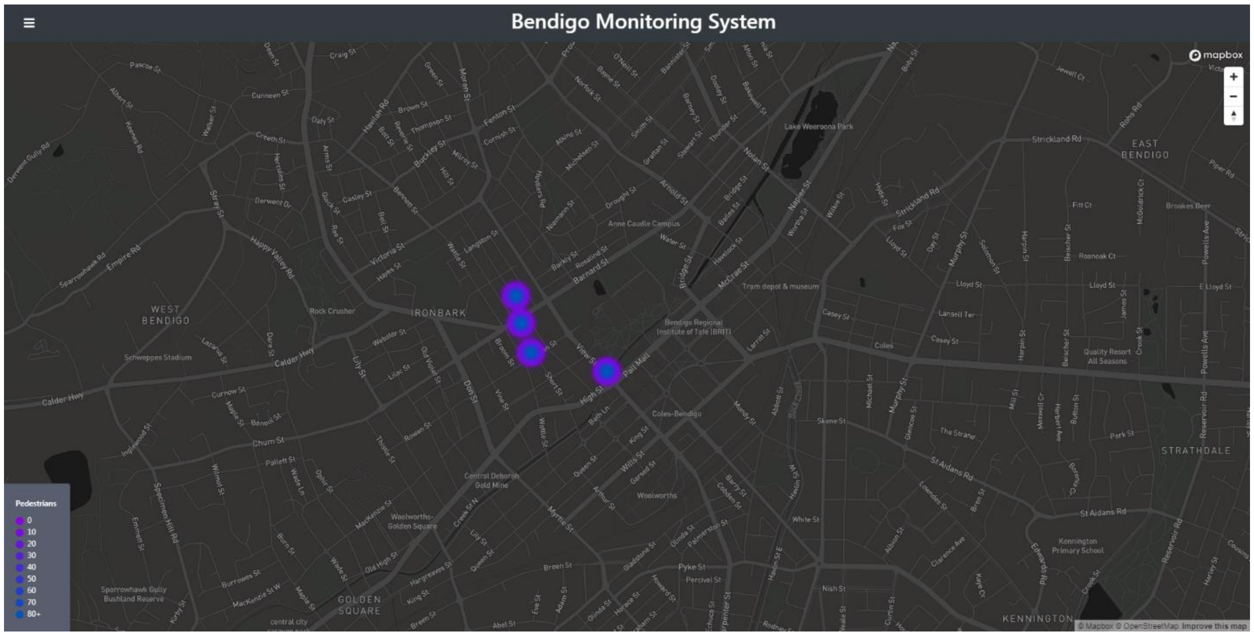
# 3.0 Pedestrian Frequency Counter User Manual

This section shows the users a comprehensive guide on how to use the app and goes over any actions they may encounter in this system.

## 3.1 Homepage

After navigating to the web address of the app, it will display the main page. When the user goes into the web page, the user can see the real map with the location of sensors on the page.

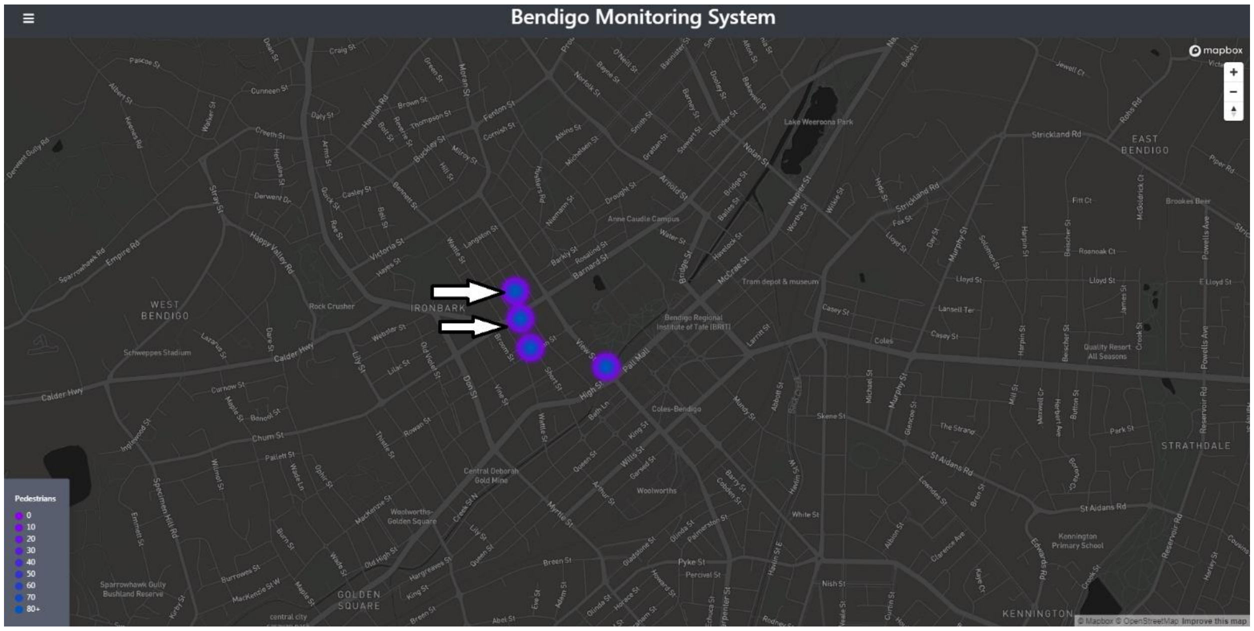
**After launching the app it'll look something like this:**



- All the sensors are located on the map.
- The heatmap gives a sense of the most used areas on the map
- When zooming in the colour of the sensor dots indicates how many people have gone past it.
- By clicking on the sensor, you can see more path usage information.
- A graph and total are shown when a sensor is clicked on, giving information at a glance.
- Location message displayed when mouse cursor is hovered on the sensor.

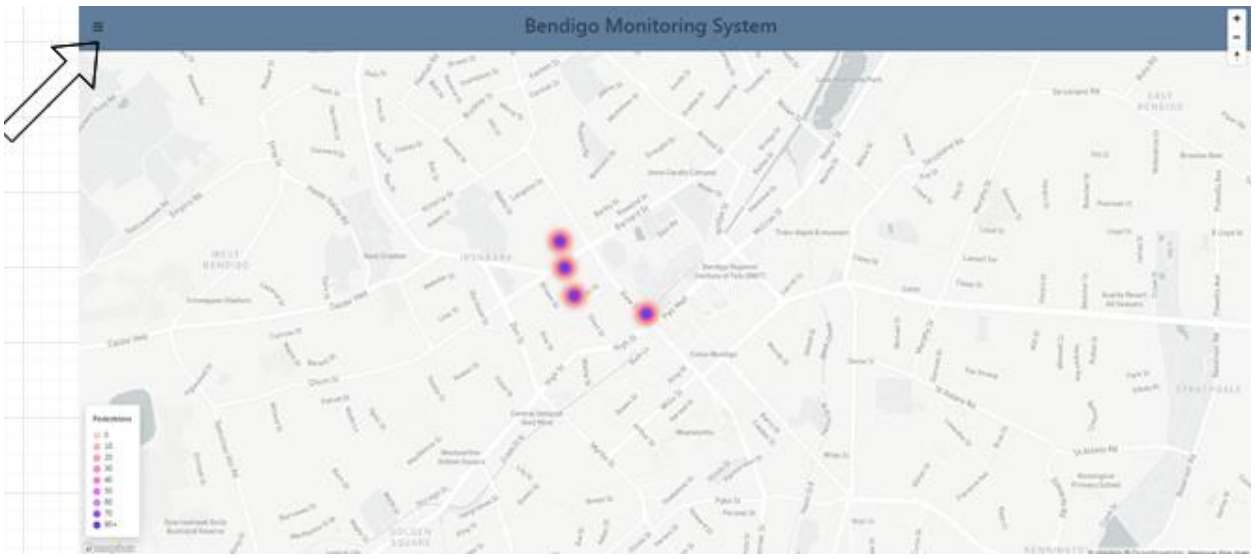
## 3.2 Interacting with Sensors

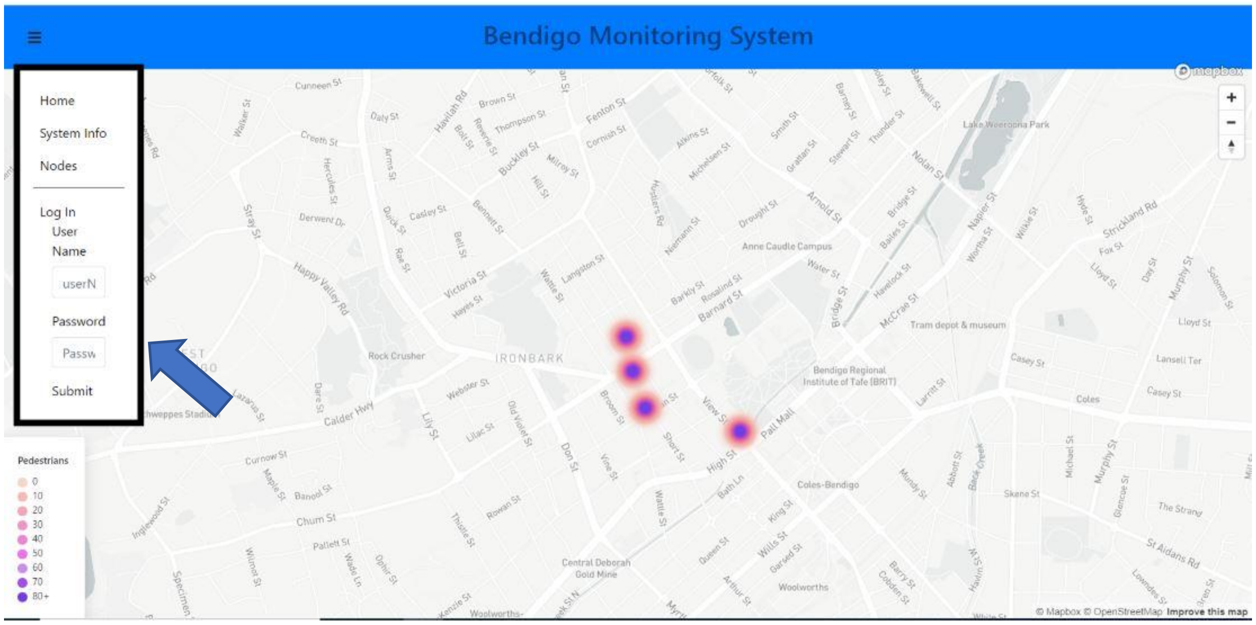
This section shows and describes that the sensors will display on the map where the user can find where the sensor node is, which is a highlighted point on the map. The sensor nodes with the location is set correctly. The user can simply interact with the homepage by zooming in and out and clicking on the sensors displayed.



## 3.3 Hamburger Menu

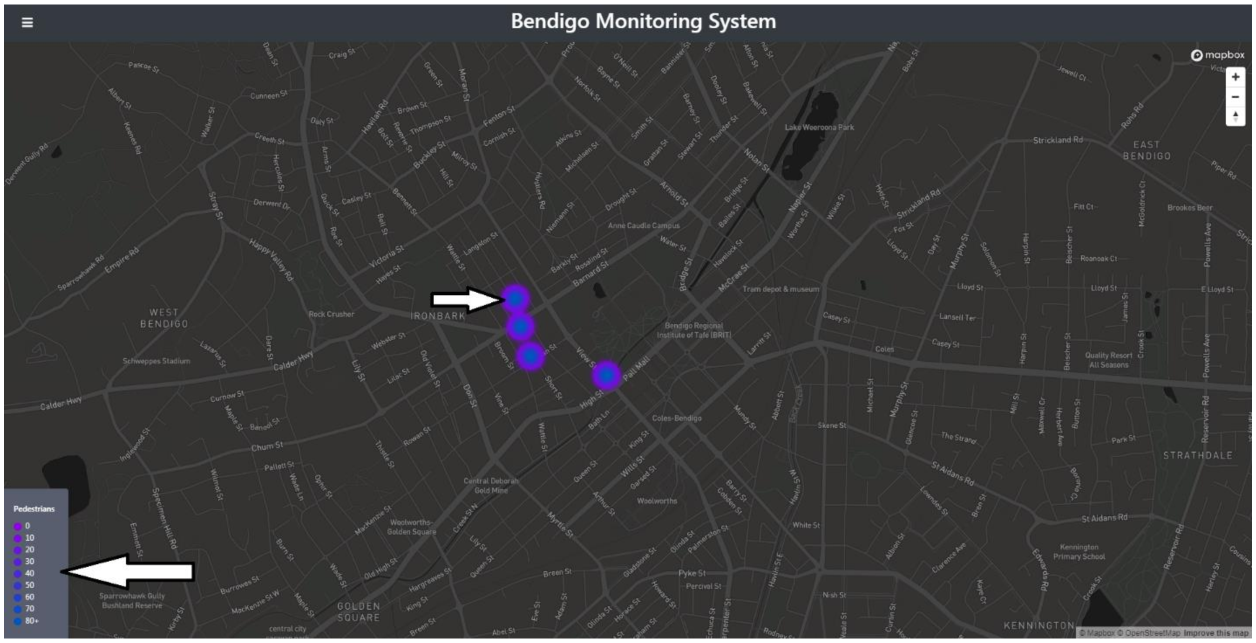
This section showcases and describes that the user can interact with the navigation bar on top of the map with a hamburger menu in the navigation bar. The user must click on the menu icon and it will drop a menu for more options.





## 3.4 Colour Code of Map

This describes the key that meaning of each sensor sensors colour in the map. Blue colour represents more pedestrians and the Light Purple colour represents fewer pedestrians.

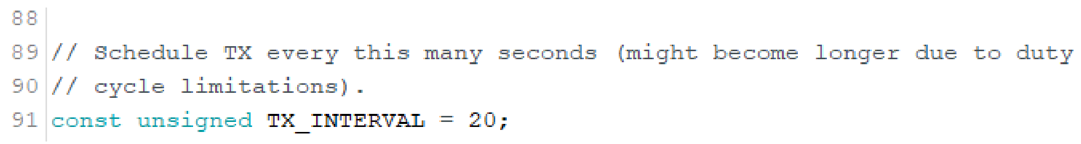


## 4.0 Installation Guide

The device requires certain libraries to run the function necessary to detect movement through its sensors. The System Administration Guide gives a detailed instruction how to deploy the project through The Things Network.

The Arduino has a microwave sensor that requires drivers installed so that it can detect speed and direction of entities passing through. Hence, the software must be pre-loaded before deploying in the field. Once installed, the software will run when given a proper power source and will periodically transmit data using the LoRa shield.

The device is set to send data every 20 seconds, mostly for testing purposes, in real life use this should be set to something 10 minutes or higher. This has to be specified in seconds, its done here:



```

// Schedule TX every this many seconds (might become longer due to duty
// cycle limitations).
const unsigned TX_INTERVAL = 20;
```

Required libraries for the Arduino:

- Arduino-lmic (requires download): https://github.com/thomaslaurenson/arduino-lmic
- MsTimer2 (requires download): https://github.com/PaulStoffregen/MsTimer2
- Hal/hal.h
- SPI.h

***Download and install the linked libraries into Arduino Studio before starting device setup***

## 4.1 Device Setup:

1. **Required Hardware:**

Arduino Uno: https://store.arduino.cc/usa/arduino-uno-rev3
LoRa Shield: https://www.dragino.com/products/module/item/102-lora-shield.html
(DFRobot) Gravity Digital Microwave Sensor: https://www.dfrobot.com/product-1403.html

2. **Hardware Setup:**

Place the LoRa Shield onto the Arduino, baking sure to match up all the pins, then connect the antenna o the LoRa Shield

The Microwave Sensor needs to be connected to the A1, GND and VCC(5v) pins of the Arduino

***Before uploading the code to each device, it needs to be added and its code configured to connect to The Things Network (see section 5.3)***

Once the device code is configured you can now upload the code to the Arduino in Arduino Studio, If everything was configured correctly the device will be able to send data to The Things Network (assuming it's in range of a gateway)

## 4.2 Web App Setup:

### 4.2.1 Building the web app for running locally or deployment on a server

***Before building the app make sure you have configured the Mapbox access key (See section 5.1)***

***This app is set up to be used with Docker for portability and ease of deployment. Ensure Docker and docker-compose are installed/setup on your machine or web service before running.***

Run the following commands to start:

```
docker-compose build
docker-compose up
```

After the containers have started run the following commands to initialise the database: (You only have to do this on first run)

```
docker exec -it pedestrian-frequency-counter_db_1 mysql
```

Then in the mysql prompt paste the following commands:

```
use ttn_db;

CREATE TABLE IF NOT EXISTS sensor_data (
      pedl int,
      pedr int,
      cycll int,
      cyclr int,
      temp double,
      devid varchar(255),
      date datetime
);

CREATE TABLE IF NOT EXISTS nodes (
      devid varchar(16) not null,
      lat   FLOAT(10,6) not null,
      lng   FLOAT(10,6) not null,
      street      varchar(255)      not null,
PRIMARY KEY (devid)
);
```

After this has been done the app will have launched in a deployment state, by default it is accessed via `http://localhost` (This is proxied to port 5000 of the frontend image by NGINX). All of the API endpoints are accessed via `http://localhost/api/` (This is proxied to port 1880 of the API image by NGINX). **`http://localhost/api/` also gives access to the configuration page of NodeRED so access to it will need to be restricted from anywhere other than on the server/host.**

***After building the app make sure you have connected it to The Things Network (see section 5.2)***

**Now the app is setup, connected and ready to deploy.**

(Check out https://aws.amazon.com/getting-started/tutorials/deploy-docker-containers/ for information on how to deploy a docker image to AWS)

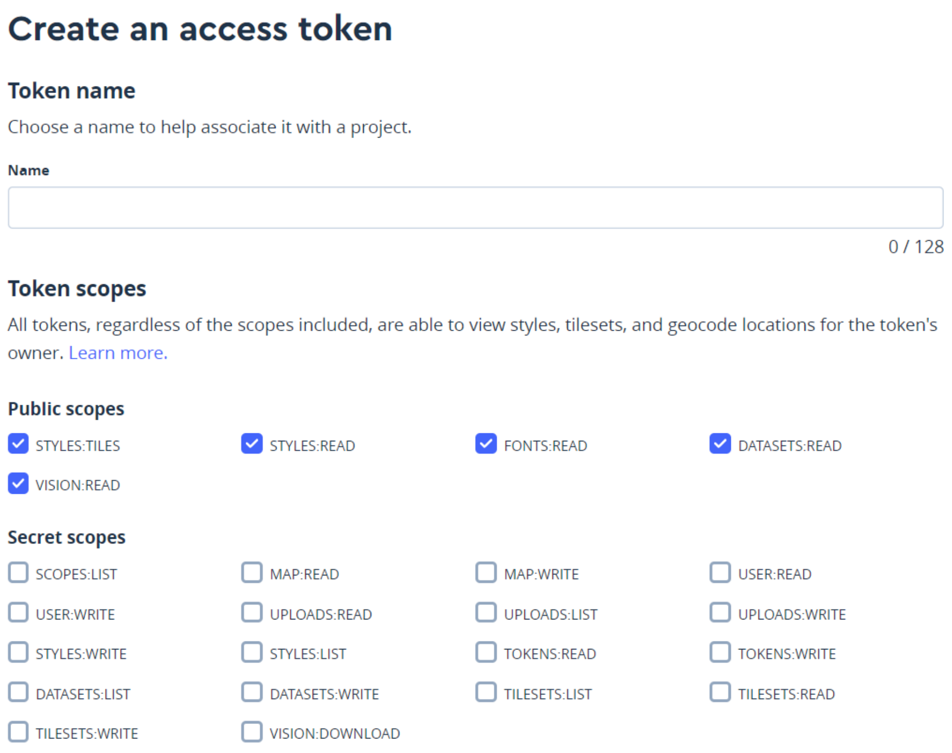
# 5.0 System Administrator Guide

## 5.1 Getting an Access Token for Mapbox

(Depending on traffic to the site, a paid version of Mapbox may be required)

Create an account at https://account.mapbox.com/

Once an account has been created navigate to https://account.mapbox.com/access-tokens/create/ to add a new access token to the account. Enter a recognizable name and leave all the tick boxes as default (see below):



Create an access token

Token name

Choose a name to help associate it with a project.

Name

0 / 128

Token scopes

All tokens, regardless of the scopes included, are able to view styles, tilesets, and geocode locations for the token's owner. Learn more.

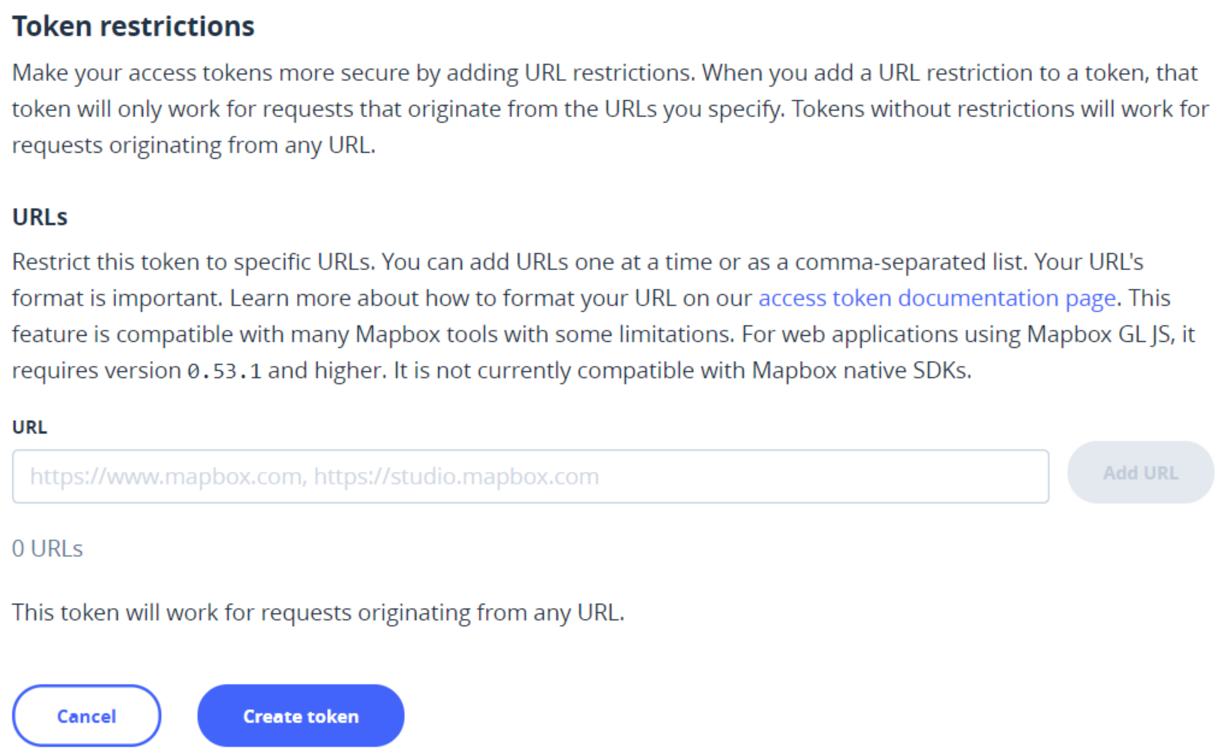
Public scopes

[✓] STYLES:TILES [✓] STYLES:READ [✓] FONTS:READ [✓] DATASETS:READ
[✓] VISION:READ

Secret scopes

[ ] SCOPES:LIST [ ] MAP:READ [ ] MAP:WRITE [ ] USER:READ
[ ] USER:WRITE [ ] UPLOADS:READ [ ] UPLOADS:LIST [ ] UPLOADS:WRITE
[ ] STYLES:WRITE [ ] STYLES:LIST [ ] TOKENS:READ [ ] TOKENS:WRITE
[ ] DATASETS:LIST [ ] DATASETS:WRITE [ ] TILESETS:LIST [ ] TILESETS:READ
[ ] TILESETS:WRITE [ ] VISION:DOWNLOAD

It is also a good idea to restrict this token so that it’s only able to be used on the application’ URL, this can put this in the field at the bottom:



**Token restrictions**

Make your access tokens more secure by adding URL restrictions. When you add a URL restriction to a token, that token will only work for requests that originate from the URLs you specify. Tokens without restrictions will work for requests originating from any URL.

**URLs**

Restrict this token to specific URLs. You can add URLs one at a time or as a comma-separated list. Your URL's format is important. Learn more about how to format your URL on our access token documentation page. This feature is compatible with many Mapbox tools with some limitations. For web applications using Mapbox GL JS, it requires version `0.53.1` and higher. It is not currently compatible with Mapbox native SDKs.

URL

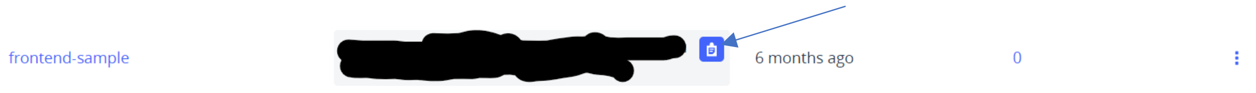
https://www.mapbox.com, https://studio.mapbox.com

0 URLs

This token will work for requests originating from any URL.

Once everything is configured properly click “Create Token”

On the next page click the blue copy button to the right of the token:



frontend-sample [redacted] 6 months ago 0

This token can now be pasted into the \frontend\src\components\Map.js file, in the mapboxgl.accessToken field, just under the imports. Paste it between the quotations:

```
import React from "react"
import mapboxgl from "mapbox-gl"
import { connect } from "react-redux"

import { filterDate } from "../redux/reducers/TableData"
import { loadMarkers, loadMarkersByDate } from "../redux/reducers/markers"
import { isMobile } from "react-device-detect";

// Public key for mapbox API
mapboxgl.accessToken = "";
```

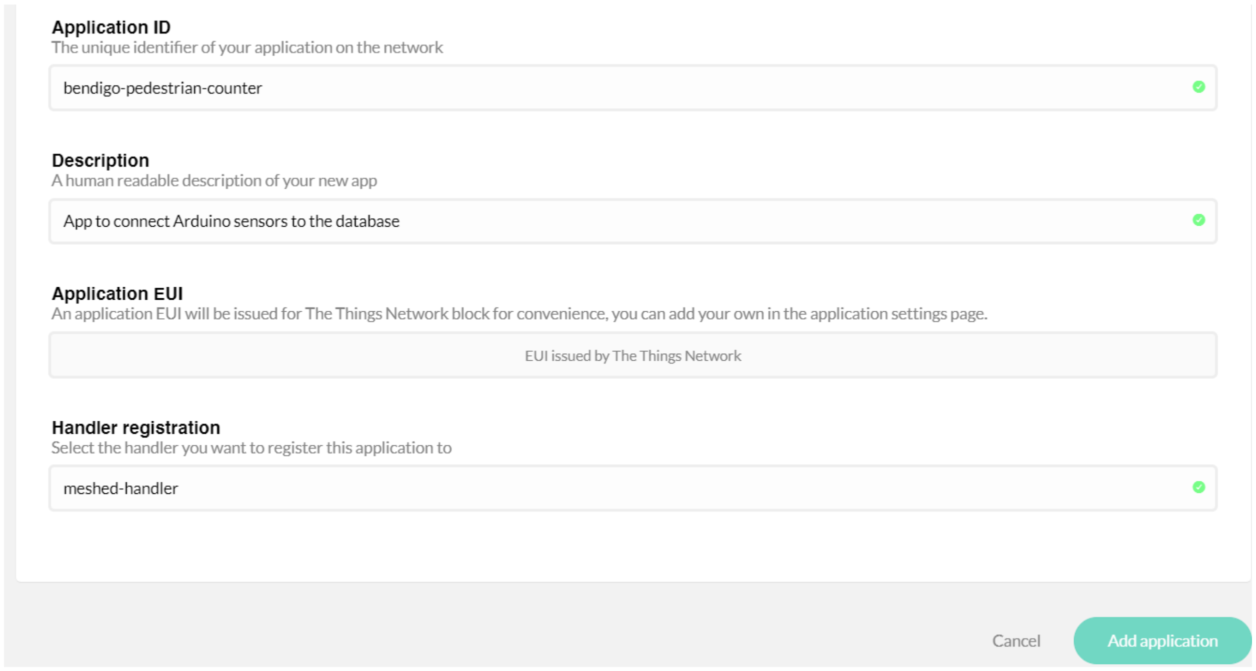
## 5.2 Creating App and Connecting to TTN

Once the web app has been built and the image is running it is now possible to add the required configuration to connect the API to The Things Network.

Go to https://www.thethingsnetwork.org/ and create an account

Once an account has been created, navigate to https://console.thethingsnetwork.org/ click on “applications” then “add application” in the top right.

Give the app a unique, identifiable name, add a description and make sure to change the handler to “meshed-handler”. Once that’s done, click “Add application”



**Application ID**
The unique identifier of your application on the network
bendigo-pedestrian-counter

**Description**
A human readable description of your new app
App to connect Arduino sensors to the database

**Application EUI**
An application EUI will be issued for The Things Network block for convenience, you can add your own in the application settings page.
EUI issued by The Things Network

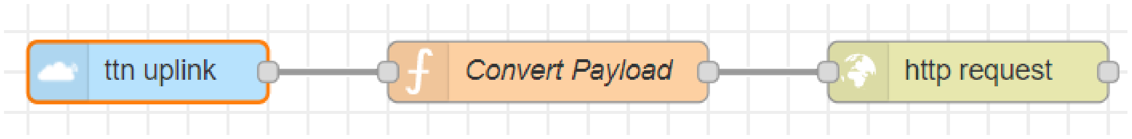
**Handler registration**
Select the handler you want to register this application to
meshed-handler

The **Access Key** that is generated after creating the app (down the bottom of the app page) is what is needed to connect the API to Things Network.

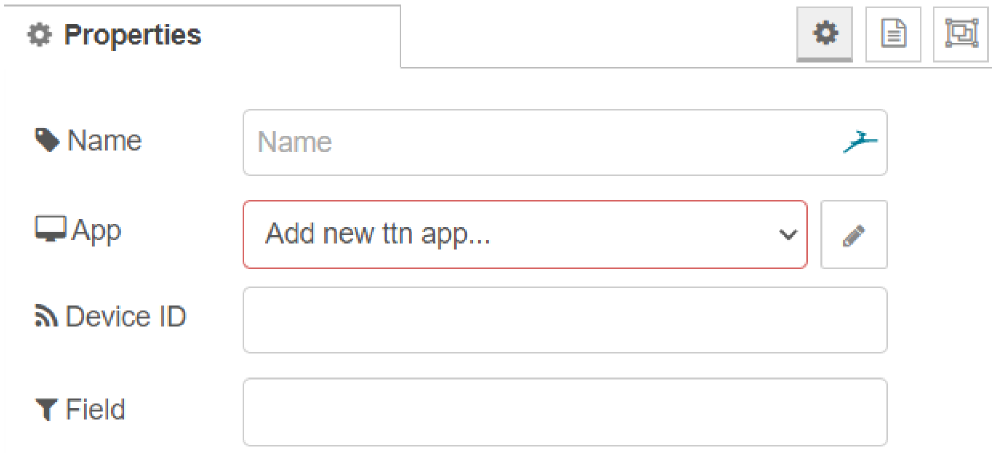
While on The Things Network application page, go to the “Payload Formats” tab and copy and paste the following into the “Decoder” section:

```
function Decoder(bytes, port) {

      return {
            PersonLeft: bytes[0],
            PersonRight: bytes[1],
            CycleLeft: bytes[2],
            CycleRight: bytes[3],
            Temp: bytes[4]/5,
      }
}
```

Next, open http://localhost/api/ and double click “ttn uplink” in the “Flow 1” tab



Then click the pencil next to the “App” field



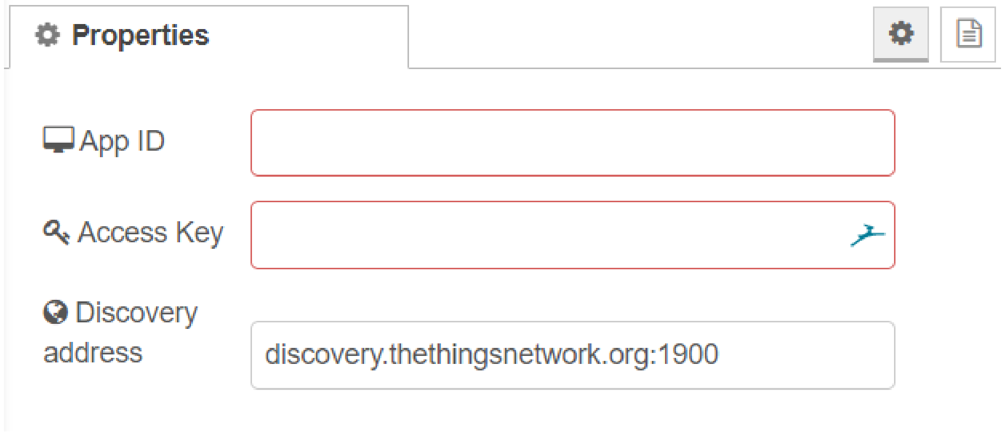
Name: Name

App: Add new ttn app...

Device ID

Field

Enter the App ID of the app that was just created and paste the **Access Key** into the Access Key field



App ID

Access Key

Discovery address: discovery.thethingsnetwork.org:1900

Click “Add” then “Done” and finally, “Deploy” in the top right-hand corner, if the configuration was successful the Docker console should output a line similar to the following:

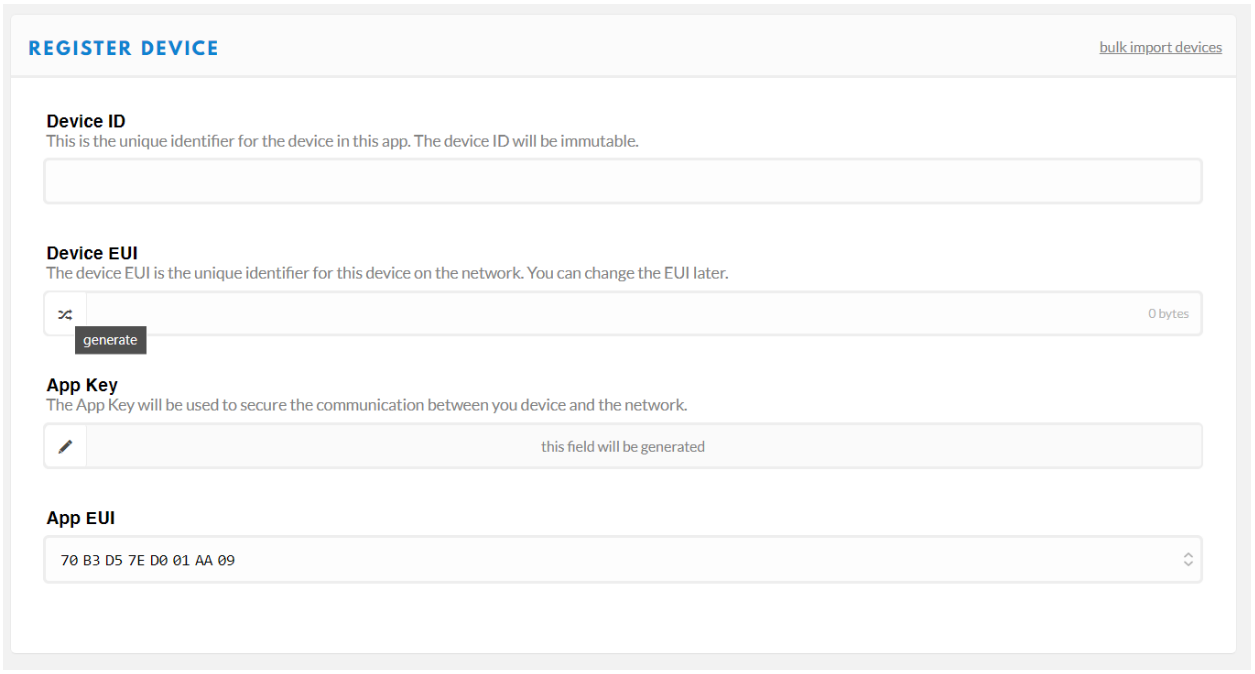
```
[info] [ttn app:9863dd2.ad0df2] Connected to TTN application test-app-18519961
```

## 5.3 Adding a Device to The Things Network and Configuring Arduino Code

(**NOTE:** This configuration needs to be done per-device, otherwise multiple devices will send data to the same device in The Things Network)

Navigate to https://console.thethingsnetwork.org/applications/, open your application, scroll down to the "Devices" section and click "register device".

Give the device a unique ID, click the "generate" button to the left of the "Device EUI" field and click "Register" at the bottom.



REGISTER DEVICE

bulk import devices

**Device ID**
This is the unique identifier for the device in this app. The device ID will be immutable.

**Device EUI**
The device EUI is the unique identifier for this device on the network. You can change the EUI later.

0 bytes

generate

**App Key**
The App Key will be used to secure the communication between you device and the network.

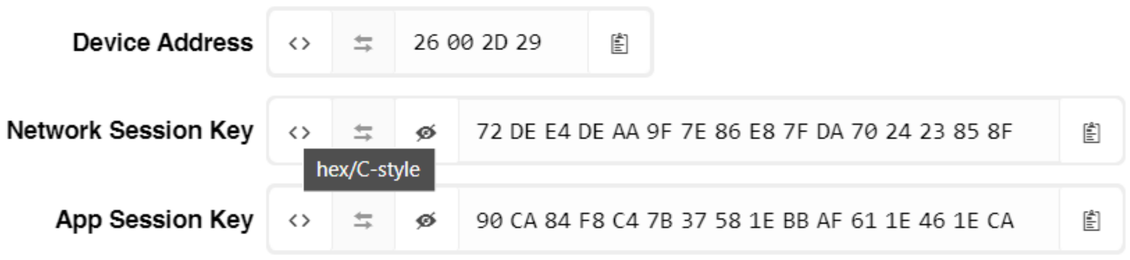
this field will be generated

**App EUI**

70 B3 D5 7E D0 01 AA 09

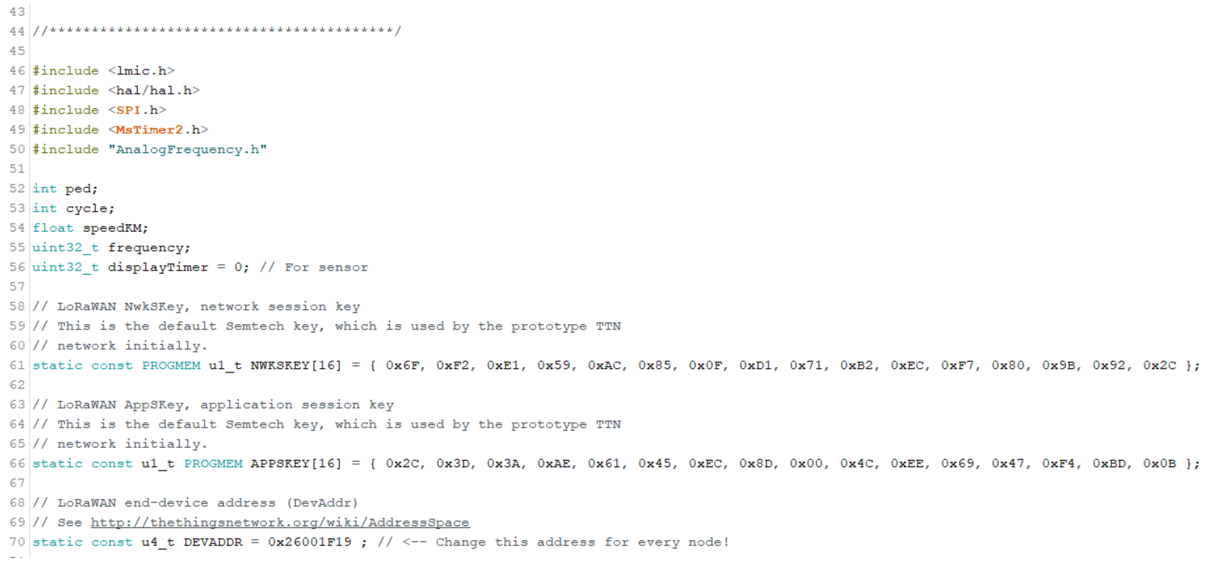
Once the device is registered, go to the device's "settings" tab, change the activation method to "ABP" and click save.

Next, copy the device's Network Session Key, App Session Key and Device Address to the device's code in Arduino Studio. (You can select the hex/C-style button ( < > ) to put the addresses into the format that the code expects.)



| | |
|---|---|
| Device Address | 26 00 2D 29 |
| Network Session Key | 72 DE E4 DE AA 9F 7E 86 E8 7F DA 70 24 23 85 8F |
| App Session Key | 90 CA 84 F8 C4 7B 37 58 1E BB AF 61 1E 46 1E CA |

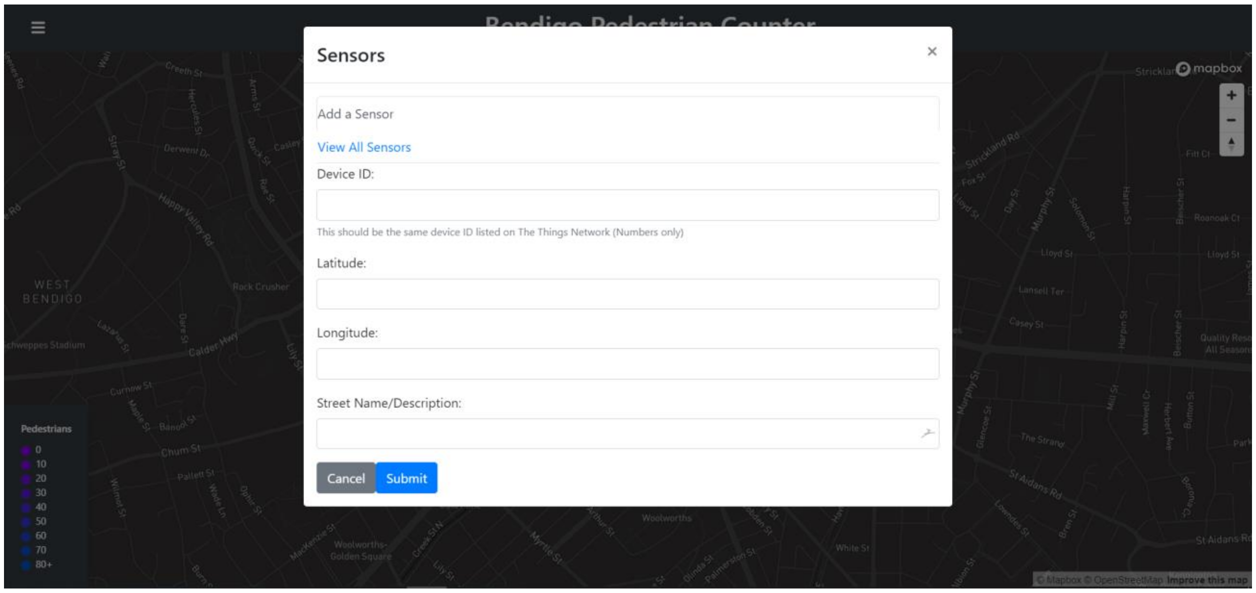
In the code these fields are found below the imports and variable declarations:



```

//*******************************************/

#include <lmic.h>
#include <hal/hal.h>
#include <SPI.h>
#include <MsTimer2.h>
#include "AnalogFrequency.h"

int ped;
int cycle;
float speedKM;
uint32_t frequency;
uint32_t displayTimer = 0; // For sensor

// LoRaWAN NwkSKey, network session key
// This is the default Semtech key, which is used by the prototype TTN
// network initially.
static const PROGMEM u1_t NWKSKEY[16] = { 0x6F, 0xF2, 0xE1, 0x59, 0xAC, 0x85, 0x0F, 0xD1, 0x71, 0xB2, 0xEC, 0xF7, 0x80, 0x9B, 0x92, 0x2C };

// LoRaWAN AppSKey, application session key
// This is the default Semtech key, which is used by the prototype TTN
// network initially.
static const u1_t PROGMEM APPSKEY[16] = { 0x2C, 0x3D, 0x3A, 0xAE, 0x61, 0x45, 0xEC, 0x8D, 0x00, 0x4C, 0xEE, 0x69, 0x47, 0xF4, 0xBD, 0x0B };

// LoRaWAN end-device address (DevAddr)
// See http://thethingsnetwork.org/wiki/AddressSpace
static const u4_t DEVADDR = 0x26001F19 ; // <-- Change this address for every node!
```

## 5.4 Adding a sensor to the map

This section describes that the client can add a sensor node to the map . A device ID, Latitude, Longitude and street name or Description of the sensor are required. The Device ID needs to be the same as the Device ID of the device in The Things Network console for the map to display its data correctly.



Sensors

Add a Sensor

View All Sensors

Device ID:

This should be the same device ID listed on The Things Network (Numbers only)

Latitude:

Longitude:

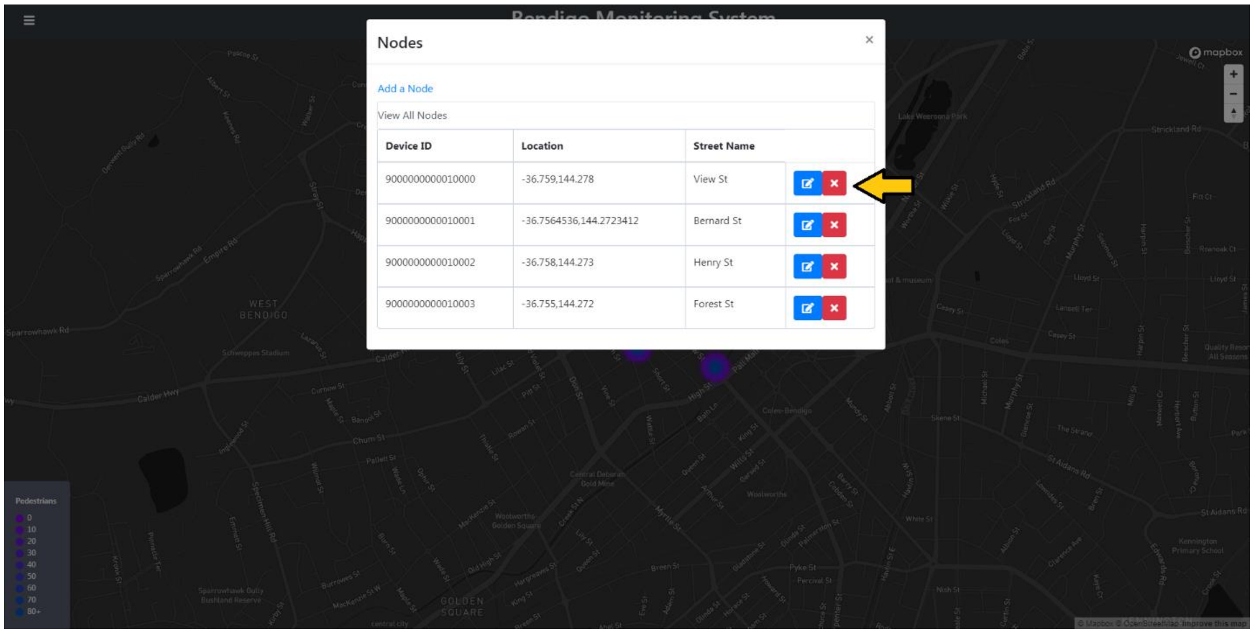
Street Name/Description:

Bendigo Pedestrian Counter
Cancel
Submit
mapbox
Pedestrians
0
10
20
30
40
50
60
70
80+
WEST BENDIGO
© Mapbox © OpenStreetMap Improve this map

## 5.5 Deleting and Editing a Sensor

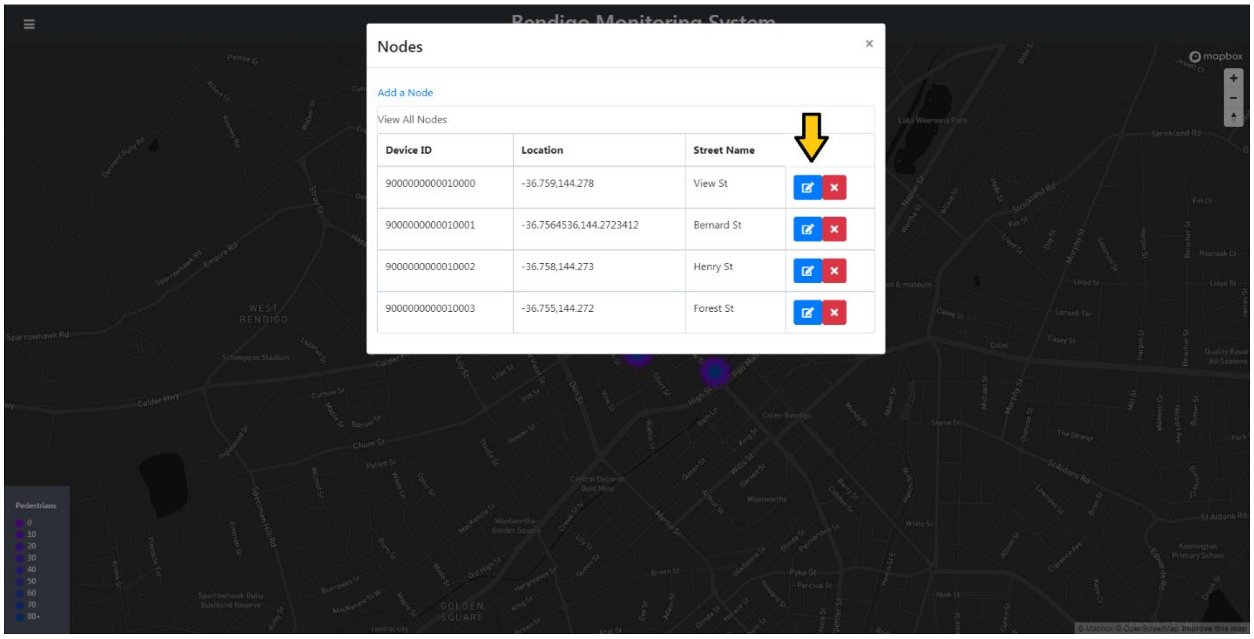
The “View All Sensors” tab shows a list of all the sensors that are stored in the database. Here you can delete or edit a sensor by simply clicking the delete or edit button next to the sensor in the sensor list. This will then switch to the “Add a Sensor” tab and populate the fields with the selected sensor’s data for editing. **NOTE:** The Device ID of the sensor can’t be edited here, in order to change this you need to delete and re-add the sensor with the correct ID.

**DELETE:**



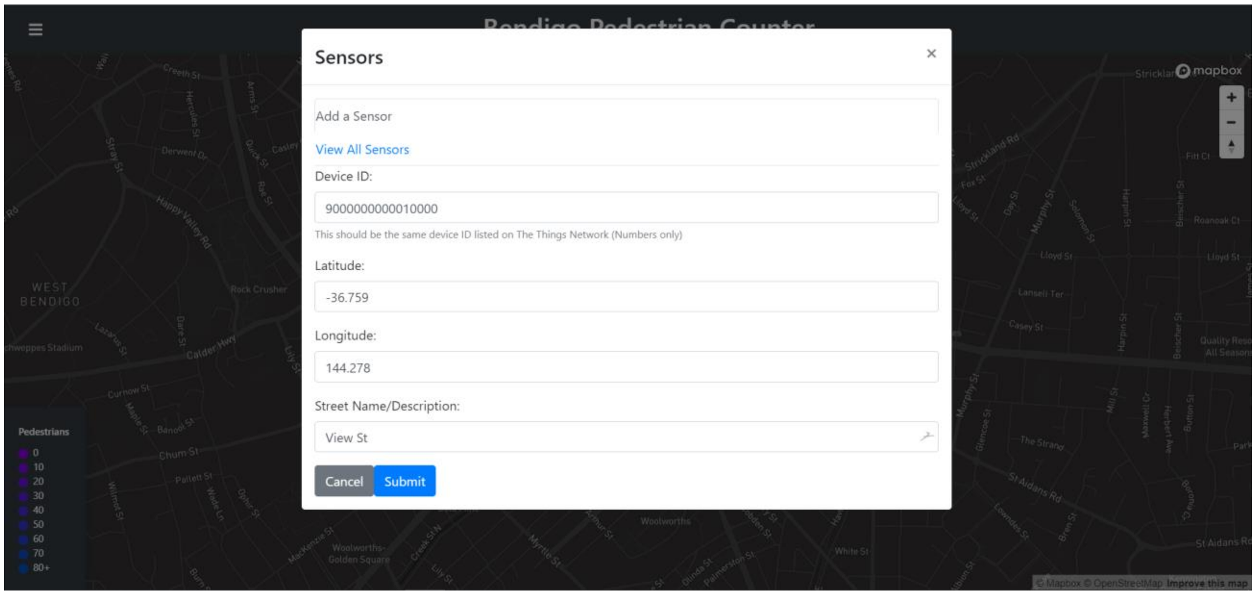
| Device ID | Location | Street Name | |
|---|---|---|---|
| 9000000000010000 | -36.759,144.278 | View St | |
| 9000000000010001 | -36.7564536,144.2723412 | Bernard St | |
| 9000000000010002 | -36.758,144.273 | Henry St | |
| 9000000000010003 | -36.755,144.272 | Forest St | |

**EDIT:**



| Device ID | Location | Street Name | |
|---|---|---|---|
| 9000000000010000 | -36.759,144.278 | View St | |
| 9000000000010001 | -36.7564536,144.2723412 | Bernard St | |
| 9000000000010002 | -36.758,144.273 | Henry St | |
| 9000000000010003 | -36.755,144.272 | Forest St | |

**After EDIT has been clicked:**



Sensors

Add a Sensor

View All Sensors

Device ID:

9000000000010000

This should be the same device ID listed on The Things Network (Numbers only)

Latitude:

-36.759

Longitude:

144.278

Street Name/Description:

View St

## 6.0 Appendices

## 7.0 Bibliography

- CSE3/5PRA & CSE3/5PRB Industry Project – 2019 Handbook.

- Lecture Materials:
    1. Week 2 – Usability Testing 2019
    2. Week 8 – Software Documentation and samples.

- Scholar sources used for:
    - Node.js
    - Bootstrap
    - CSS
    - JS
    - React
    - Redux
    - SQL
    - Visual Studio
    - Mapbox
    - Arduino
    - REST API
    - LoRaWAN
    - HTML5
    - Docker
    - Bitbucket

## 8.0 Glossary

**Architecture Diagram**

An architectural model is a rich and rigorous diagram, created using available standards, in which the primary concern is to illustrate a specific set of trade-offs inherent in the structure and design of a system.

**Bootstrap**

Bootstrap is a free and open-source front-end framework for designing websites and web applications. It contains HTML- and CSS-based design templates for typography, forms, buttons, navigation and other interface components, as well as optional JavaScript extensions.

**Visual Studio**

An integrated development environment (IDE) from Microsoft. It is used to develop computer programs, as well as web sites, web apps, web services and mobile apps.

**Bitbucket**

Bitbucket is a web-based version control repository hosting service, for source code and development projects.

**Docker**

A computer program that performs operating-system-level virtualization.

**JIRA**

Jira is a proprietary issue tracking product that allows bug tracking and agile project management.

**Node.js**

Node.js is an open-source, cross-platform JavaScript run-time environment that executes JavaScript code outside of a browser.

**React**
React is a JavaScript library for building user interfaces. It is maintained by Facebook and a community of individual developers and companies.

**The Things Network**

The Things Network is about enabling low power devices to use long-range gateways to connect to an open-source, decentralised network to exchange data with applications.

**Node – RED:** Node-RED is a flow-based development tool for visual programming developed originally by IBM for wiring together hardware devices, APIs and online services as part of the Internet of Things. Node-RED provides a web browser-based flow editor, which can be used to create JavaScript functions.

## 9.0 Index