Snowdonia Documentation

Release 0.1

Mariam Maarouf

Dec 02, 2016

Contents

CHAPTER 1

Introduction

1.1 About

This API was mainly built to provide an endpoint to collect emissions containing location and path data from different public transportation vehicles around Snowdonia. Emissions are sent every 20 seconds from nearly 1000 vehicles.

Docs below include documentation for the available endpoints and unit tests.

Logo Credit.

1.2 Installation

1. Clone the github repo

```
$ git clone https://github.com/blaringsilence/snowdonia.git
$ cd snowdonia
```

2. Install virtualenv and activate it

```
$ pip install virtualenv
$ virtualenv -p python3 venv
$ . venv/bin/activate
```

3. Install requirements

```
$ pip install -r requirements.txt
```

4. Install postgresql (see download page) and create a user and a database

```
$ sudo -u postgres createuser -s $USER
$ createdb -U $USER snowdonia
```

5. Edit the config.py file in the (second) snowdonia dir (replace USER and PASSWORD with your postgresql user and password)

6. Create the database tables. In a python interpreter:

```
>> from snowdonia import db
>> db.create_all()
```

7. Run the app using gunicorn (replace 6 with the suitable number of workers for your testing):

```
$ gunicorn snowdonia:app -w 6
```

CHAPTER 2

Testing

2.1 Unit Tests

Unit tests for the API endpoint (local, postgresql installation/configuration in the app required) are documented below and can be executed as follows:

```
$ chmod a+x test.py
$ ./test.py
```

2.2 Stress/Load Tests

For the purposes of simulating the API's behavior, we're using Locust, an open source load testing tool that uses gevent to swarm a website with requests whose behavior is described in a local configuration file.

To load the testing tool (while in the virtualenv where the requirements are installed)

```
$ cd stress_tests
$ locust
```

Note that this would run tests on the deployed herokuapp, not locally/on your deployed app. To do that, either change the host in the locustfile, or more easily, add the host parameter:

```
$ locust --host=[YOUR_HOST_AND_PORT_HERE]
```

Then head to localhost:8089, and set it to simulate 1000 users with 50 new locusts hatched/second. This should simulate the use of the system the way it's intended to be:

- 1000 concurrent behicles at max
- Emit every 20 seconds
- For each vehicle, a UUID
- For each emission, a location that is within the 50km radius of Snowdonia, a valid type, a valid timestamp, and a valid heading.

Please note that the new points for each vehicle are random so not necessarily in the direction their previous point was supposed to be headed.

In order to see which vehicle is emitting what data, the name of the request (in the Locust web interface) is set to:

```
TYPE_OF_VEHICLE-INDEX_OF_VEHICLE at (LATITUDE, LONGITUDE)
```

Where:

- TYPE_OF_VEHICLE is tram, bus, train, or taxi
- INDEX_OF_VEHICLE is the index of this vehicle in relation to others of its kind

For example, the 10th bus' emission at lat=53.1725575782715 and long=-4.3319528534938545:

```
bus-10 at (53.1725575782715, -4.3319528534938545)
```

CHAPTER 3

Docs

3.1 API

For the purposes of this API, we decided to skip the hassle of registering, so if this is the first time you're using this API, the app will register new vehicles on the fly. Just send the data and you're golden.

To send data from a vehicle, point to the following link:

```
/api/v1/emissions/<VEHICLE_UUID>
```

Where the vehicle UUID is a valid UUID4.

The request should be of type PUT, and the data that are expected by the API are:

- **latitude**: floating point between -90 and 90
- **longitude**: floating point between -180 and 180
- **type**: the vehicle's type. Allowed types: taxi, tram, train, and bus.
- **timestamp**: a string in the following format: DD-MM-YYYY hh:mm:ss
- **heading**: an angle between 0 (True North) and 359 that indicates where the vehicle's heading.

Please note that this API is only for public vehicles in Snowdonia, so any co-ordinates outside of Snowdonia's 50km radius will yield an error. See snowdonia.register_emission(vehicleID) below for details.

class snowdonia.**Emission**(*vehicle_id*, *latitude*, *longitude*, *timestamp*, *heading*)
Database model for emissions. Contains:

•id (int, auto-incremented)

•vehicle_id (foreign key referencing the UUID in vehicles)

•latitude (float from -90 to 90)

•longitude (float from -180 to 180)

•timestamp (DateTime)

•heading (int, angle, from 0 - True North - to 359)

class snowdonia.**Vehicle**(*id*, *type*)
Database model for vehicles. Contains:

•id (str, UUID4)

•type (str): taxi, tram, bus, or train

snowdonia.**distance_from_center**(*latitude*, *longitude*)
Calculates the distance between the provided point and the town center by using Vincenty's formula that calculates the distance between two points on a spheroid, given:

•Radius of the Earth (min and max) as well as its flattening.

•Latitude and longitude of both the point & the center in radians.

Even though there exists the Haversine formula that calculates the distance between two points on a sphere, and it is less computationally expensive than Vincenty (no iterations), the Haversine formula, when calculating distances on the Earth, can have an error up to 0.55%, though generally below 0.3%, so Vincenty provides greater accuracy that is actually needed in this situation, where exactly where vehicles were is valuable data.

More on how Vincenty's formula works: https://en.wikipedia.org/wiki/Vincenty's_formulae

snowdonia.**home**()
A brief summary page and the landing page for the app.

snowdonia.**in_range**(*latitude*, *longitude*)
Determins if distance from center is <= 50, Snowdonia's radius in kms.

snowdonia.**register_emission**(*vehicleID*)
The API endpoint that collects emissions. URL:

```
/api/v1/emission/<VEHICLE_ID>
```

How it works:

•The UUID4 for the vehicle is provided in the API endpoint URL

•**The data that have to accompany the PUT request:**

– latitude: float between -90 and 90

– longitude: float betwen -180 and 180

– timestamp: string of the timestamp in the form: DD-MM-YYYY hh:mm:ss

– heading: int from 0 to 359

Responses:

•Success [200]: 'Success!'

•Co-ordinates or heading invalid/Co-ordinates are too far [400]: 'Co-ordinates/heading invalid'

•Vehicle ID or vehicle type invalid [400]: 'Vehicle ID or vehicle type is invalid'

•Invalid data types [400]: 'Invalid value(s) provided'

•Other exception [400]: 'Unexpected error'

snowdonia.**snowdonia_center = (0.926226843425401, -0.07113187104941002)**
(lat, long) radian co-ordinates for the town center. Fun fact: This points to the Snowdonia region in Wales.

snowdonia.**valid_point**(*lat_val*, *long_val*, *heading*)
Checks:

•Latitude is between -90 and 90

•Longitude is between -180 and 180

•Heading is between 0 and 359

•Point is within the town borders (less than 50km from town center)

snowdonia.**valid_types = ['taxi', 'bus', 'tram', 'train']**
Valid types of vehicles in Snowdonia.

snowdonia.**valid_vehicle**(*vID*, *vType*)
Checks:

- Vehicle ID is a valid UUID4
- Vehicle type is a valid type (train, tram, taxi, or bus)

3.2 Unit Tests

class test.**TestCase**(*methodName='runTest'*)
Test case class.

emit(*vID*, *type_val*, *lat_val*, *long_val*, *timestamp*, *heading*)
Simulate an emission.

setUp()
Set testing = True and assign this class's app to a test client.

test_far_point()
Tests a far lat, long point.

test_invalid_data()
Tests an emission with invalid data.

test_invalid_heading()
Tests an invalid heading value.

test_invalid_id()
Tests an invalid UUID format.

test_invalid_timestamp()
Tests an invalid timestamp format.

test_invalid_type()
Tests an invalid type of vehicle.

test_valid_emit()
Tests a valid emission that should pass.

s

t

D

E

H

I

R

S

T

V