Name: Anne Alexander
User Name: aa02036
URN: 6402256
Date: 14/07/18
Module: COMM002

Instructions for Installing and Running Software Submitted for COMM002

These instructions have been tested on a Mid 2014 MacBook Pro running macOS High Sierra Version 10.13.6. These instructions assume that COM002-Build.zip has been downloaded and unzipped in a suitable location.

Prerequisites:

Python 3 (tested with version 3.7)
XCode command line tools (tested with version 2349) (can be installed with xcode-select --install)

Setting up the Database

Install MySQL Community Server 8.0.11.
Note – If planning on using MySQL Workbench, use legacy password encryption (the new encryption is not compatible with MySQL Workbench).
https://dev.mysql.com/downloads/mysql/

Make sure that MySQL Server is running, then change directory to /COM002-Build /Database.

Open MySQL CLI with the command:

```
mysql --user root --password
```

Enter the password set up during MySQL installation when prompted.

If the mysql command is not recognised, the full path may be required – the default install location is /usr/local/mysql-8.0.11-macos10.13-x86_64/bin/mysql

From within the MySQL CLI type:

```
source Create and Populate DB.sql;
```

Alternatively: Install MySQL Workbench from
https://dev.mysql.com/downloads/workbench/

And apply /COM002-Build /Database/Create and Populate DB.sql using the GUI.

Installing the software

First of all, edit /COM002-Build/Common Files/config.ini to have the correct connection parameters for MySQL Server, and make sure MySQL Server is running. The commands in this section should be executed from /COM002-Build.

It is recommended to install the software in a virtual environment, to avoid potential configuration problems on the target machine. To create and activate a virtual environment:

```
python3 -m venv WISLEY_ENV
source WISLEY_ENV/bin/activate
```

Then run the script to carry out the installation:

```
./wisley.sh
```

1. This script will install all the dependencies listed in the requirements.txt file.
2. Download the pyproj library from github then build and install it (this step is only necessary because installing pyproj using pip into a Python 3.7 environment did not work at time of testing).
3. Build and install the wisley software.
4. Configure Flask and run the app.

Once this has been completed, the application should be accessible on http://127.0.0.1:5000/ from a web browser or API Client.

API documentation is available in OpenAPI format in the file /COM002-Build/API_doc.yaml, which can be pasted into the editor at https://editor.swagger.io/ for an interactive version. API definition and example usage are available in the project report.

Ctrl-C will stop the Flask development server.

The virtual environment created contains all the packages required by the main software and all the prototypes. The following three prototypes should be run from the virtual environment.

Database and Spatial Queries Prototype

This prototype was created to explore the process of modelling geographical data in MySQL. This section creates a database called wisley_pt, to distinguish it from the master database created above.

This prototype comprises of four Python programs which generate SQL scripts. These scripts can then be applied to the database using either MySQL CLI or MySQL Workbench. All the Python programs should be run from within the virtual environment created above. Some of the Python programs require that earlier scripts have been applied to the database, so it is important to carry out these steps in order. All these files are in the /COM002-Build/Database Prototype directory, and all commands should be run from this directory.

Apply Create DB.sql to create the wisley_pt database schema.

Run:

```
python parse_kml.py
```

This creates create_nodes.sql, create_places.sql and create_polygons.sql. Apply to wisley_pt database in any order to initialise the node, place and flower_bed tables.

Run:

```
python edges.py
```

This creates create_edges.sql. Apply to wisley_pt database to initialise the edge table.

Run:

```
python utilities.py
```

This creates update_edges_directions.sql, update_proj_coords.sql, update_beds.sql, update_place.sql and insert_plant_beds.sql. Apply to wisley_pt database in any order to update the edge table with directions and projected coordinates, update the flower_bed and place tables with nearest nodes and populate the plant_bed table.

Run:

```
python months.py
```

This creates write_months.sql. Apply to wisley_pt database to populate plant_month table.

Note, this will not create a database identical to /COMM002-Build/Database/Create and Populate DB.sql, as plant_bed and plant_month table entries are generated randomly. The contents of the node, edge, flower_bed and place tables will be the same. To replicate the

results in the report exactly, use /COMM002-Build/Database/Create and Populate DB.sql instead.

Sample Spatial Queries.sql contains some of the queries used during prototyping. These can be applied to the database using MySQL CLI or MySQL Workbench. To run these queries on the main database, just change 'USE wisley_pt;' to 'USE wisley;' at the top of the file.

The table below summarises the files in the Build/Database Prototype directory:

File	Description
parse_kml.py	Creates SQL scripts for node, place and flower_bed table population.
edges.py	Creates SQL script for edge table population.
utilities.py	Creates SQL script for updating edge, node and flower_bed tables. Also creates script for populating plant_bed table.
months.py	Creates SQL script for populating plant_month table.
adj_list.txt	Node adjacency file – required by Edges.py
Create DB.sql	SQL script to create empty database.
config.ini	Database connection parameter and coordinate projection library configuration file.
Elvetham.kml	Geographic data (flower beds) downloaded from Scribble Maps in KML format. Required by ParseKML.py.
nodes.kml	Geographic data (nodes) downloaded from Scribble Maps in KML format. Required by ParseKML.py.
plantselector.xml	Plant data sample in XML format. Required by Utilities.py.
Sample Spatial Queries.sql	Sample spatial queries used in prototyping.

XML Parsing Prototype

This prototype was developed to test different methods of parsing large amount of XML data. All files are in the /COMM002-Build/XML Parsing Prototype, and all commands should be run from here.

This prototype uses its own configuration file, /COMM002-Build/XML Parsing Prototype/config.ini. This file contains the following information:

number_of_plants = 5000
runs = 5
repeats = 3

The original benchmarking tests were done with 300000 plant records, which produces a very large XML file and is very slow to process. The configuration file can be used to control the number of plant records produced.

To generate the large XML file:

```
python write_big_file.py
```

This creates a file called /COMM002-Build/XML Parsing Prototype/big_xml.xml, containing the number of plant records specified in config.ini. This file was not included in the submission due to its large size.

Time Tests

The code for the time tests is in the /COMM002-Build/XML Parsing Prototype/Time Tests directory, split into two sub-directories as detailed below.

Directory	Python File	Description
ElementTree	etree_find_first.py	Finds first plant using ElementTree find method.
	etree_find_last.py	Finds last plant using ElementTree find method.
	etree_iter_first.py	Finds first plant using ElementTree iterparse method.
	etree_iter_last.py	Finds last plant using ElementTree iterparse method.
lxml	lxml_find_first.py	Finds first plant using lxml find method.
	lxml_find_last.py	Finds last plant using lxml find method.
	lxml_iter_first.py	Finds first plant using lxml iterparse method.
	lxml_iter_last.py	Finds last plant using lxml iterparse method.

As an example, to run the time test for finding the first plant with ElementTree.iterparse, use this command (from the /COMM002-Build/XML Parsing Prototype directory):

```
python Time\ Tests/ElementTree/etree_iter_first.py
```

These programs all output to stdout. The number or runs and repeats executed by the time tests can be adjusted in config.ini.

Memory Tests

The code for the time tests is in the /COMM002-Build/XML Parsing Prototype/Memory Tests directory, split into two sub-directories as detailed below.

Directory	Python File	Description
ElementTree	etree_find_first.py	Finds first plant using ElementTree find method.
	etree_find_last.py	Finds last plant using ElementTree find method.
	etree_iter_first.py	Finds first plant using ElementTree iterparse method.
	etree_iter_last.py	Finds last plant using ElementTree iterparse method.

lxml	lxml_find_first.py	Finds first plant using lxml find method.
	lxml_find_last.py	Finds last plant using lxml find method.
	lxml_iter_first.py	Finds first plant using lxml iterparse method.
	lxml_iter_last.py	Finds last plant using lxml iterparse method.

As an example, to run the memory test for finding the last plant with ElementTree.iterparse, use this command (from the /COMM002-Build/XML Parsing Prototype directory):

```
python –m memory_profiler Memory\
Tests/ElementTree/etree_iter_last.py
```

These programs all output to stdout. They use the same XML file generated at the beginning of this section. They use the number_of_plants value from the configuration file, but do not use the runs and repeats values.

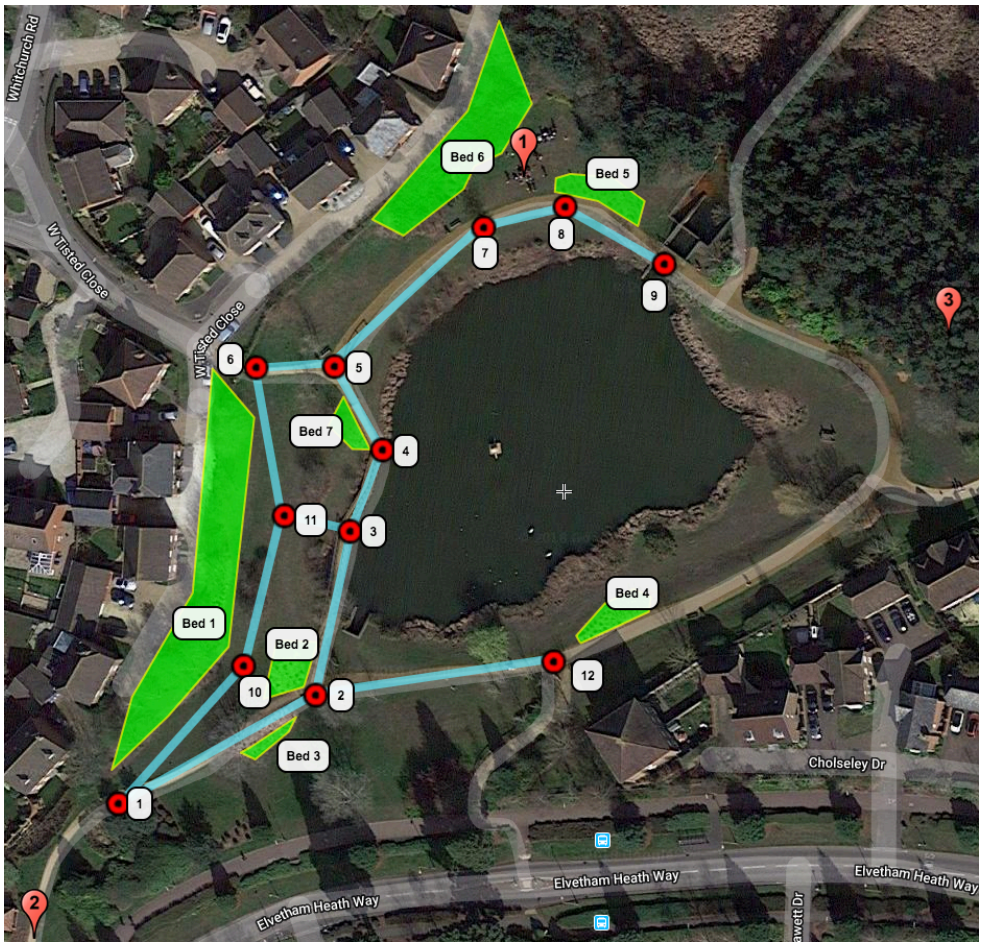
Routing Prototype

The code for the routing prototype is in /COMM002-Build/Routing Prototype. It requires the database to have been setup and for MySQL Server to be running. It uses the database connection parameters defined in /COMM002-Build/Common Files/config.ini.

It is run using the following command (from /COMM002-Build/Routing Prototype):

```
python astar_route.py
```

It prompts for source and destination nodes, then outputs the calculated shortest path and path length to the screen. Below is a copy of the test data set node network:



Unit Tests

The unit test code is in the directory /COMM002-Build/COMM002/unit tests. Files are named by putting test_ in front of the corresponding source code file name.

There is a script to run all the unit tests, to run it:

```
./run_all_tests.sh
```

From /COMM002-Build.

API Test Suite

The API Test Suite was developed using the open source version of SoapUI, which is available for download here: https://www.soapui.org/downloads/soapui.html.

To run the tests, select File -> Import Project from the SoapUI menu, and select the file /COMM002-Build/COMM002/API Testing/Wisley REST API-soapui-project.xml when prompted.

Double click on ‘Wisley TestSuite’ in the navigator pane to open the test suite window, then click the green ‘play’ arrow in the top left-hand corner of this window to run all the tests.

Note: If the Flask server has been stopped, run it with the following commands, (executed in the virtual environment):

```
export FLASK_APP=wisley
flask run
```