**­**

City Mapping APP

|  |  |  |
| --- | --- | --- |
| **Name** | **Date** | **Version** |
| Samson Demma | Friday, September 21, 2018 | 1.0 |

Table of Contents

1 Introduction and Purpose3

2 Requirements3

3 Resources 4

4 Controlling the City Mapping app 4

5 Structure of the API 5

6 Testing .7

7 Future 8

­­

**1 Introduction and Purpose**

What & Why?  
If we would like to find out which capital city belongs to which European Country we can use this super simple Mock API

**Clients [with City Name] citymaping-app [with City + Country]**

Figure.1

* The **City Mapping API** is a web service which will be residing on Linux Server. The API given a **City** as a parameter with the request will return the **City** mapped to the corresponding **Country** name for that City. The app itself is written in **Flask** (a Python Web-Framework).For more information on Flask framework, see <http://flask.pocoo.org/docs/1.0/>.

**2 Requirements**

The API needs to cover the following main points:

* **JSON** data *JavaScript Object Notation* data format
* **CRUD** operations –*Create Read Update Delete* operations needs to be accommodated.
* **REST** **standard** API needs to operate on resource representations and tries as much as possible to follow **REST** (Representational State Transfer) design patterns.
* **Easy** to maintain and flexible- well commented and documented code.

**3 Resources**

* The resources used to develop (mandatory modules) the API can be found on the automatically generated simple text file named “requirements” which comes with the app. This information can also be found on the **README.md** file. Besides resources the **README.md** file contains procedure to control and test the API end points, run web server and other useful information about the service.

**4 Controlling the CITY Mapping app Service**

Access & Run service

* The citymapping-app can be run in 2 ways, i.e. for development and production.
* if we are working on development environment., we can simply run the app as we would any other Python script:

Command: **python app.py**

* After we ran the above command we should see the server is running at the given address and port number.
* For production we should be using **gunicorn** web server. Gunicorn is a Python WSGI (Web Server Gateway Interface) HTTP Server for UNIX. Nginx can be used to act as a reverse proxy. Gunicorn can also be used in daemon mode.

**Start**

Command: **gunicorn -w 4 -b 127.0.0.1:8000 app:app**

The above command uses 4 workers and binds the localhost to port 8000

**Stop**

**Command: pkill –P1 gunicorn**

The overall structure with web server and client request,response in detail can be seen below:

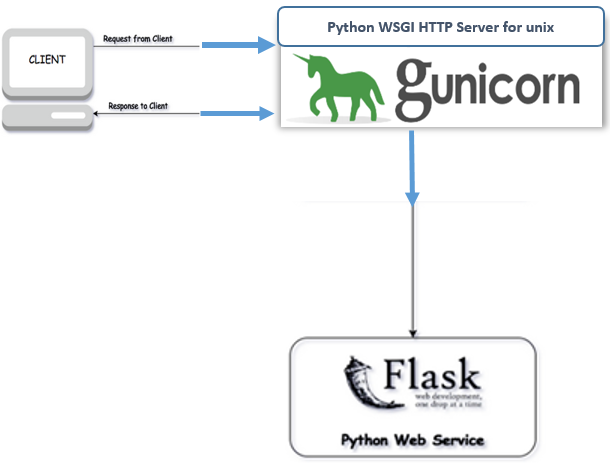


Figure.2

In the above picture we are using following technologies and components:

* Flask – server/backend , app is written and served with Python
* Client – even though we can test with any device, main client in this case refers to ME Unit devices.
* Gunicorn – deploy flask app in production. Supports multiple workers, speedy, a Pre-fork worker model.

**5. Structure of the API**

* Structure of the API follows the common request /response modern paradigm.
* Clients sends a **GET, POST, PUT, DELETE** request with JSON payload, if incorrect request is sent, client will be informed.
* The API depending on the request will return the resource or create one.
* Data is stored in **SQLite** tiny database named **city.db**

Figure.3 shows the structure:

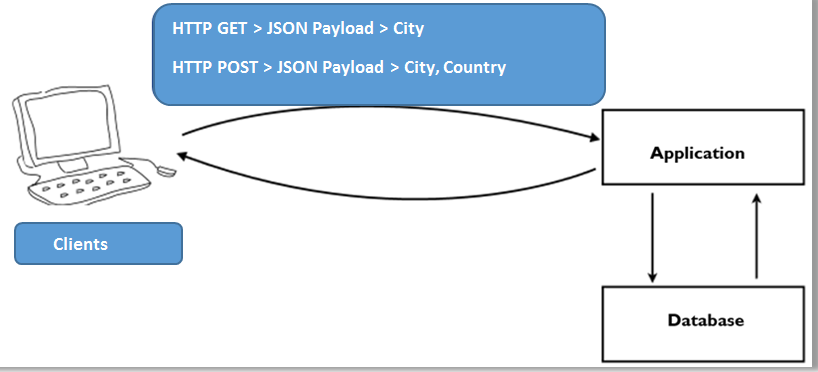


Figure.3

* Application main application is represented by a specific module named **app.py**
* Database the database is handled and run by module named **models.py**, and the Schemas are run from the module named **schemes.py**. **SQLAlchemy** (which is an ORM (Object Relational Mapper) is used to manage/convert database data from an object-oriented programming perspective relating database entries with classes and database fields with properties. This fortunately eliminates the need to write a lot of pure SQL which can be tedious An ORM is a mechanism. For more details see <http://www.sqlalchemy.org/>, http://flask-sqlalchemy.pocoo.org/2.3/) is used to manage the database part.

Overall structure of the app:

**|-app.py**

**|-models.py**

**|-schemes.py**

**|-README.md**

**|-requirements.txt**

**|-cities.db**

**6 Testing**

In *Development Environment* after we run the app using the following command:

Command: **python app.py**

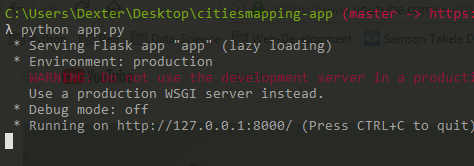
****

Figure.4

We will get notification that our server is running on local host at port 8000 .

Before we get our web server up and running we should **create** the database and add (**seed**) data into it. Look at figure below:

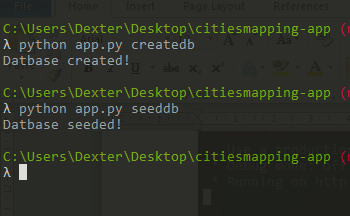


Figure.5

Once we have the server up & running, we can make a call to the API using **cURL**:

Command: **curl –v** [**localhost:8000/city\_mapper/read/Amsterdam**](http://127.0.0.1:7779/TESTBROADBANDALNNZ2)

The above command is making a call to the api with the CITY as a parameter. Once we hit enter we should receive a successful message back from server presenting us with CITY and COUNTRY name.

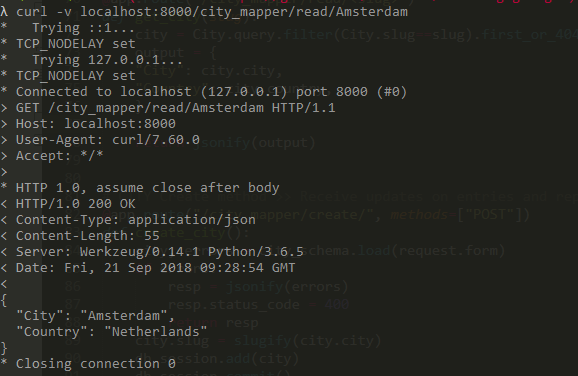
****

Figure.7

If we send an incorrect CITY name to the API >> we should get back a 404-custom message saying “Invalid request”:

Command: **curl –v http://127.0.0.1:8000/city\_mapper/read/Amsterdamm**

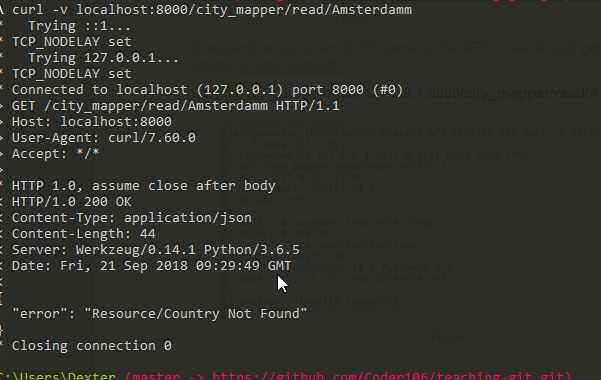
****

Figure.8

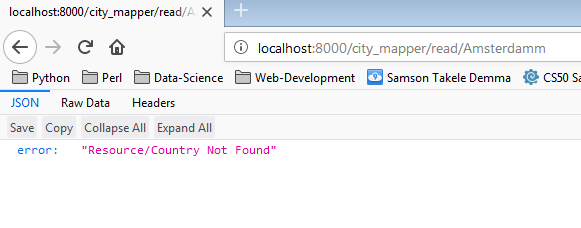


Figure.9

DELETE and UPDATE, methods also are covered , sample screen shots below[figure 10]:

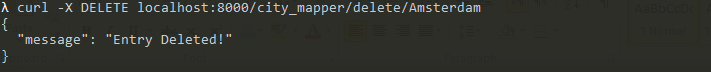


Figure.10

If there is a need to READ all the Entries we can make a call to the API at this path: http:127.0.0.1:8000/city\_mapper/

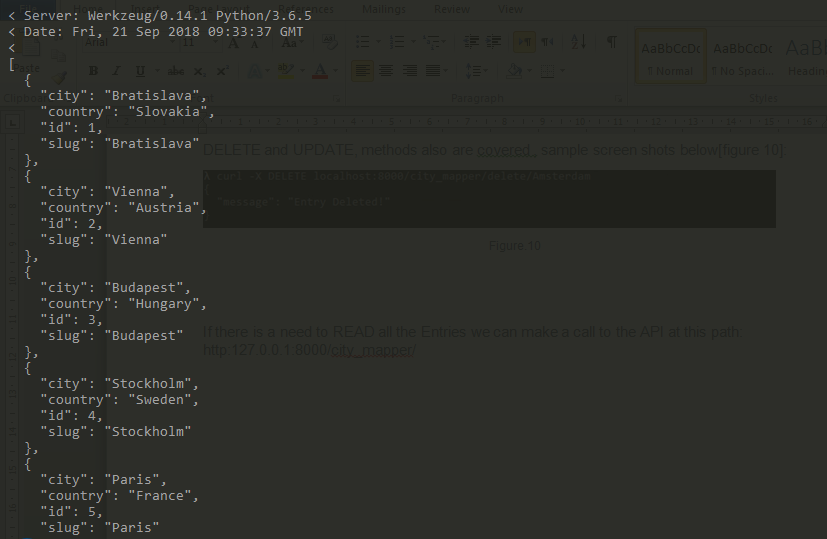


Figure.11

If a bulk entry is needed we can use a custom JSON file which contains the CITY+COUNTRY pair and or IP block information and send a POST request to create resources in the database for furthere use. Example:

Command: **curl -kL -d "@update.json" -X POST** [**http://127.0.0.1:8000/city\_mapper/update/**](http://127.0.0.1:8000/city_mapper/update/)

Content of the update.json file can be:

**{**

**“city” : “Amsterdam”,**

**“country”: “Netherlands”**

**}**

Authentication

There is also a mechanism for the Units to authenticate themselves, if there is a need in the future to authenticate Units before processing their requests:

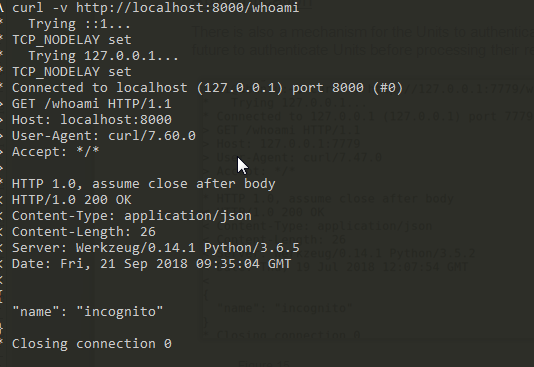


Figure.15

As you can see since the unit is not authenticated the API replied with default city name which is set to “incognito”.

Conclusion

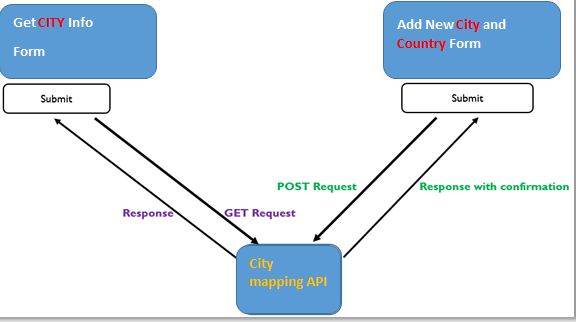
At this moment we can do the following with the API:

* Send and receive JSON data format from the API, with **GET, POST, PUT, DELETE** methods (this is optional, data format can also be xml)
* API is REST full:
* Client/server > web service and client are not together but separated.
* Stateless > minimum information is required from client so that server can perform the request. There is no storage of information given by client (no reuse of information).
* “code at fingertips” > we can write own scripts to send requests to the API
* Code base is simple
* Can perform CRUD operations after defining the models (conversion > database objects to python objects and vice versa)

TODO: Future

Develop the front-end part of the app, so that the users can interact with the API with ease. The front end can have 3 buttons, and 4 fields:

* A form named “CITY” where users input CITY name (at the backend this can be realized as a GET request)
* A submit button under the form
* A form named “CITY, COUNTRY”, where users input the new Cites, Countries. This can be realized on the backend as a POST request)
* A submit button under the second form



**City mapping API**

Figure.16

Resources for further Reading:

<http://docs.gunicorn.org/en/0.16.0/index.html>

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

<https://curl.haxx.se/docs/httpscripting.html>

<https://www.getpostman.com/docs/v6/>

<https://www.python.org/>