POI Search By Route Based on PostGIS

WangDelong

2020-4-10

Software Design Memorandum

Introduction

This project implements a server based on PostGIS, which has a POI search function, accept serial points as input, and return POIs contained in the polygon which is buffered the route constitute from input points.

Project Background

We have a business idea that filters the best tourist spots to a traveler who plans to self-driving. The scenario is when a traveler statements at least 2 points to conduct a traveling route, and then find out potential places of interest along the route way.

Requirements Analysis

The first step is to analyze the true requirements and essential foundation. down below is what we need.

A core function, return POIs by serial input points, one map to show off query result.

CURD API to maintain POI data, actually that's 4 APIs.

Few admin APIs to query server status or set/reset database.

Otherwise, my server must accord with conditions:

1. portable
2. easy to setup
3. easy to scale up
4. some other effective target

Technical Selection

the technology I chose:

- Database: PostGIS, it is mature, support many spatial functions and easy to use.
- Coding Language: Node.js, naturally none-blocking, and I'm familiar with it.
- Web Framework: Express, widely used web framework on node.js
- Test Framework: Python + Locust, no special reason, totally replaceable
- Operation System: Ubuntu 18.04LTS, it's convenient to use
- Container: Docker & Docker-Compose
- Map Preview: Baidu Map

Database Model Design

We design 3 tables as basic dataset and server state.

1. Setup Table: record whether the server initialized by now, and which step it have been done.
2. Region Table: place the virtual fence locations of provinces, and country
3. POI Table: store all POIs, each must include latitude and longitude, this table support a "tags" field, for extra business purpose.

Setup Table

Field Name	Usage	Usage
phase	string	stages that record server initialized on which step
complete	boolean	does this stage complete

Region Table

Field Name	Type	Usage
id	int	auto increment primary key
code	string	post code of this region
name	string	region name
border	geometry	a polygon shape enclosure this province

POI table

Field Name	Type	Usage
id	int	auto increment primary key
source_id	int	POI id from datasource
tags	jsonb	attached infos for POI
point	geometry	POI location
updated_at	time	mark when POI was being changed

Table creation SQL Statements

```
CREATE TABLE setup (
    phase character varying(32),
    complete boolean
);

CREATE TABLE region (
    id SERIAL PRIMARY KEY,
    code character varying(32) NOT NULL UNIQUE,
    name character varying(64) NOT NULL UNIQUE,
    border geometry(Polygon,4326)
);

CREATE UNIQUE INDEX region_pkey ON region(id int4_ops);
CREATE UNIQUE INDEX region_code_key ON region(code text_ops);
CREATE UNIQUE INDEX region_name_key ON region(name text_ops);
CREATE INDEX region_gis_index ON region USING GIST (border gist_geometry_ops_2d);

CREATE TABLE poi (
    id SERIAL PRIMARY KEY,
    source_id integer NOT NULL UNIQUE,
    tags jsonb,
    point geometry(Point,4326),
    updated_at timestamp without time zone
);

CREATE UNIQUE INDEX poi_pkey ON poi(id int4_ops);
CREATE UNIQUE INDEX poi_source_id_key ON poi(source_id int4_ops);
CREATE INDEX poi_gis_index ON poi USING GIST (point gist_geometry_ops_2d);
CREATE INDEX poi_source_id_index ON poi USING HASH (source_id int4_ops);
CREATE INDEX poi_tags_jsonb_index ON poi USING GIN (tags jsonb_ops);
```

API Design

This server supply HTTP protocol interfaces, include 4 kinds of API:

1. Admin API, used for management of server, like initialized/reset.

2. POI CRUD API, used for Create/Read/Update/Delete POI info.

3. Core API, the one API, used for query POIs from database, and this is the vital foundation.
4. others, used for assist dev or usage convenient.

Admin API

Request Path	Usage
POST /admin/create-table	create table in postgis
POST /admin/import-china-bundary	import country border data
POST /admin/import-scenic-points	import default POI data
POST /admin/reset-all	remove all data and tables in postgis

POI API

Request Path	Usage
GET /poi/list	list pagination POI data
POST /poi/create	create a new POI, do nothing if source_id already exist
POST /poi/update	update a exist POI, create one if source_id not exist
GET /poi/info	return POI info by source_id, return 404 if source_id not exist
POST /poi/delete	remove a POI by source_id, do nothing if source_id is not exist

Core API

Request Path	Usage
POST /aggregate	return POIs by serial input points

Other API

Request Path	Usage
GET /ping	server alive check point
GET /toolkit	backend dashboard entrypoint

Web Server

Since we provide a server with API, so we must have a web server, and we chose Express(http://expressjs.com/), simply provide API entry points we talked before.

Input Data

The core function is POI query, and we need input data to filter what POI that user are interested in, and then we should design input parameter format to let people input the basis requirements. that's the query parameters we design within the core API.

Example:

```
{
    "points": [
        { "lat": 31, "lng": 121 },
        { "lat": 26, "lng": 120 }
    ],
    "pageSize": 2,
    "pageNum": 1,
    "filter": {
        "rank": 0,
        "city": "北京"
    },
    "filterType": "and",
    "mode": "polylineBuffer",
    "distance": 10000,
    "shrink": 0.3
}
```

Fields introduction:

- points： every points the user pass through, each contains latitude and longitude property, at least one point is required
- pageNum： pagination parameter, which page want to get
- pageSize: pagination parameter, how many points wanted in a single page
- distance: buffered distance from the route line, only affect when "mode" is "polyline-Buffer"
- mode： which kind of query mode, only "polylineBuffer" or "bundingCircle" are sup-ported
- filter: what tags you want to filter
- filterType: how to filter multi tags, support "and" / "or"

- shrink: circle shrink ratio, shrink value must between 0 and 1
- debug: return debug info, show query times, buffered polygon location

PostGIS ST_* functions

PostGIS is a spatial database extender for PostgreSQL object-relational database. It adds support for geographic objects allowing location query to be run in SQL.

What we used in project includes[3]: ST_AsText, ST_GeomFromText, ST_Buffer, STBuffer, ST_Distance, ST_Collect, ST_Centroid, ST_MinimumBoundingCircle, ST_Scale, ST_Translate, ST_Rotate.

Buffered Polyline Query

Presume user input a serial points: [Nanjing, Hangzhou, Shanghai][4], and then POIs filters followed step-by-step

1. connect points to a line in sequence

```
Select ST_AsText(
    ST_GeomFromText(
        'LINESTRING(32.06 118.78, 30.30 120.15, 31.20 121.46)'
        , 4326)
)

#> LINESTRING(32.06 118.78,30.3 120.15,31.2 121.46)
```

2. buffered the polyline

```
Select ST_AsText(
    ST_Buffer(
        ST_GeomFromText(
            'LINESTRING(32.06 118.78, 30.30 120.15, 31.20 121.46)'
            , 4326)::geography
        , 10000
    )
)

#> POLYGON((30.4969122208969 59.8396430643649,31.3625117723414
58.5691596295698,31.0377623513761 58.5106363729532,30.1207068300521
59.8502446306579,30.1242305485943 59.8679139813266,31.90378138716
61.2689240070813,32.215734444937 61.1708951551853,30.4969122208969
59.8396430643649))
```

notice that, we buffered the route with meters unit, it must transform to geography type first, and then use ST_Buffer function extend it.

3. find out which POI in the polygon

```
Select *,
    ST_Distance(
        point,
        ST_GeomFromText(
            'LINESTRING(32.06 118.78, 30.30 120.15, 31.20 121.46)'
            , 4326
        )
    ) as distance
From POI
Where
    ST_Contains(
        ST_Buffer(
            ST_GeomFromText('LINESTRING(32.06 118.78, 30.30 120.15, 31.20 
121.46)', 4326),
            10000
        )
    , point)
Order By distance
Offset 10
Limit 10

#> [<POI>,...]
```

4. ordered and pagination and then return result

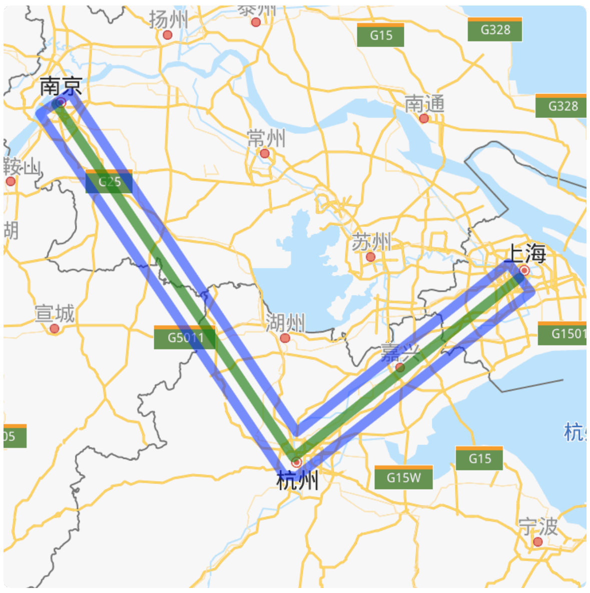
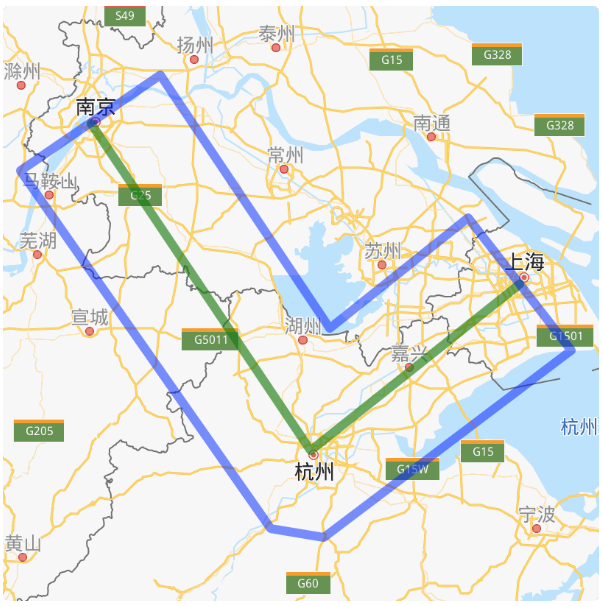
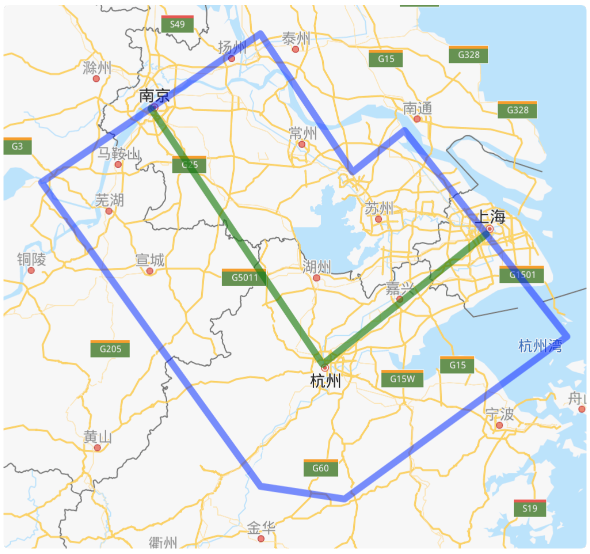
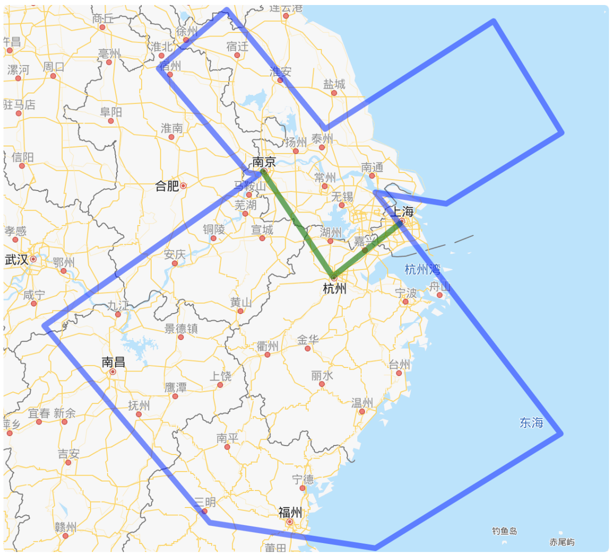
Order By distance and pagination result, our final SQL statement is like this

```
Select
    source_id, tags,
    ST_AsText(point) as point,
    ST_Distance(point, ST_GeomFromText('POINT(118.78 32.06)', 4326)) as dis-
tance,
    count(*) OVER() AS total_count
From (
    Select *, ST_Contains(polygon, point)
    From POI, ST_GeomFromText('POLYGON((120.173896300844 
30.4323794521495,121.395087294212 31.2708923342644,121.524814999751 
31.129071023049,120.167498327619 30.2063631341974,120.100397052895 
30.2154778548122,118.692601019295 32.0093637193372,118.867499476169 
32.1105836281851,120.173896300844 30.4323794521495))', 4326) as polygon ) as 
tbl
Where tbl.st_contains = true And True
Order By distance Asc, id Asc
Offset 10
Limit 10
```

By adjusting the "distance" parameter, we will get the different shape of the polygon. I list 4 results, only distinguished by buffered range.

GreenLine: route way, by connected input serial points

Blue Line: polygon, by buffered route way.

	
distance=10000 meter quite a normal route path	distance=50000 meter a wild route path
	
distance=100000 meter now it's too wild like a rect	distance=500000 meter with a big distance that will show a strange shape

Ellipse Query

Presume user input a serial points: [Nanjing, Hangzhou, Shanghai] [4], and then POIs filters followed step-by-step

1. connect points to a line in sequence and collect it

```
Select ST_AsText(
    ST_Collect(
        ST_GeomFromText(
            'LINESTRING(32.06 118.78, 30.30 120.15, 31.20 121.46)'
            , 4326
        )
    )
)

#> MULTILINESTRING((32.06 118.78,30.3 120.15,31.2 121.46))
```

2. generate a minimal circle contain all points

```
Select ST_AsText(
    ST_MinimumBoundingCircle(
        ST_Collect(
            ST_GeomFromText(
                'LINESTRING(32.06 118.78, 30.30 120.15, 31.20 121.46)'
                , 4326
            )
        ),
        2
    )
)


#> POLYGON((31.63 121.64325312389,32.7071026133661
121.197102613366,33.1532531238899 120.12,32.7071026133661
119.042897386634,31.63 118.59674687611,30.5528973866339
119.042897386634,30.1067468761102 120.12,30.5528973866339
121.197102613366,31.63 121.64325312389))
```

3. get center point of the circle

```
Select ST_AsText(
    ST_Centroid(
        ST_MinimumBoundingCircle(
            ST_Collect(
                ST_GeomFromText(
                    'LINESTRING(32.06 118.78, 30.30 120.15, 31.20 121.46)'
                    , 4326
                )
            ),
            2
        )
    )
)

#> POINT(31.63 120.12)
```

4. shrink the circle to a ellipse with a "shrink" parameter

```
Select ST_AsText(
    ST_Scale(
        ST_MinimumBoundingCircle(
            ST_Collect(
                ST_GeomFromText(
                    'LINESTRING(32.06 118.78, 30.30 120.15, 31.20 121.46)'
                    , 4326
                )
            ),
            2
        ),
        0.3,
        0
    )
)

#> POLYGON((9.489 0,9.81213078400983 0,9.94597593716696 0,9.81213078400983
0,9.489 0,9.16586921599017 0,9.03202406283305 0,9.16586921599017 0,9.489 0))
```

5. a scale function must followed by a translate function

```
Select ST_AsText(
    ST_Translate(
        ST_Scale(
            ST_MinimumBoundingCircle(
                ST_Collect(
                    ST_GeomFromText(
                        'LINESTRING(32.06 118.78, 30.30 120.15, 31.20 121.46)'
                        , 4326
                    )
                ),
                2
            ),
            0.3,
            0
        ),
        ST_X(ST_GeomFromText('POINT(31.63 120.12)', 4326)) * (1 - 0.3)
        , 0
    )
)

#> POLYGON((31.63 0,31.9531307840098 0,32.086975937167 0,31.9531307840098
0,31.63 0,31.3068692159902 0,31.173024062833 0,31.3068692159902 0,31.63 0))
```

6. rotate ellipse with the angle between first point and last point around circle centre

```
Select ST_AsText(
    ST_Rotate(
        ST_Translate(
            ST_Scale(
                ST_MinimumBoundingCircle(
                    ST_Collect(
                        ST_GeomFromText(
                            'LINESTRING(32.06 118.78, 30.30 120.15, 31.20 
121.46)'
                            , 4326
                        )
                    ),
                    2
                ),
                0.3,
                0
            ),
            ST_X(ST_GeomFromText('POINT(31.63 120.12)', 4326)) * (1 - 0.3)
            , 0
        )
        , pi()
        , ST_GeomFromText('POINT(31.63 120.12)', 4326)
    )
)

#> POLYGON((31.63 240.24,31.3068692159902 240.24,31.1730240628331
240.24,31.3068692159902 240.24,31.63 240.24,31.9531307840098
240.24,32.086975937167 240.24,31.9531307840098 240.24,31.63 240.24))
```

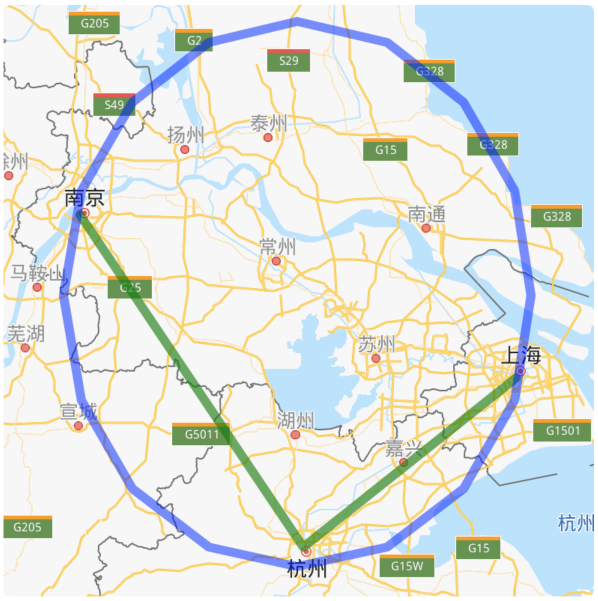
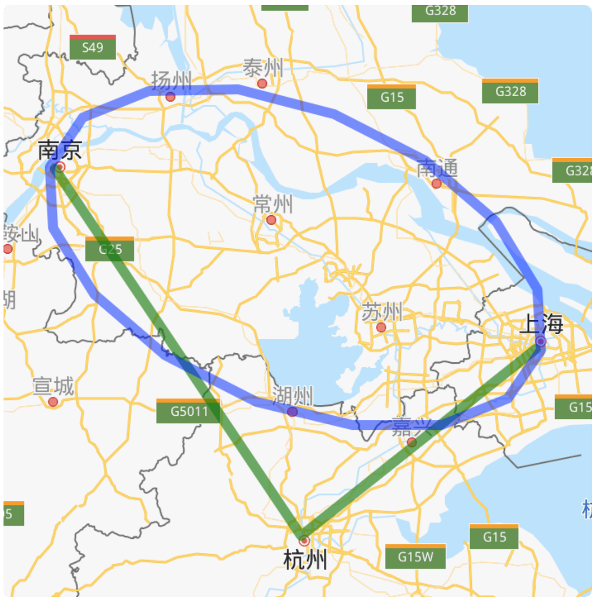
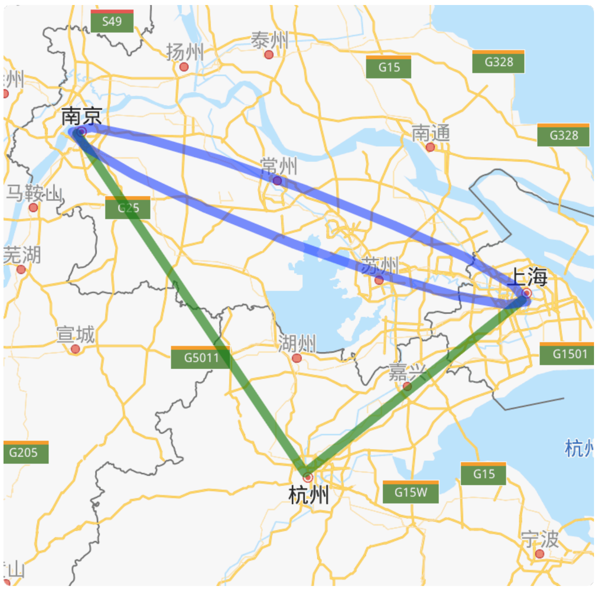
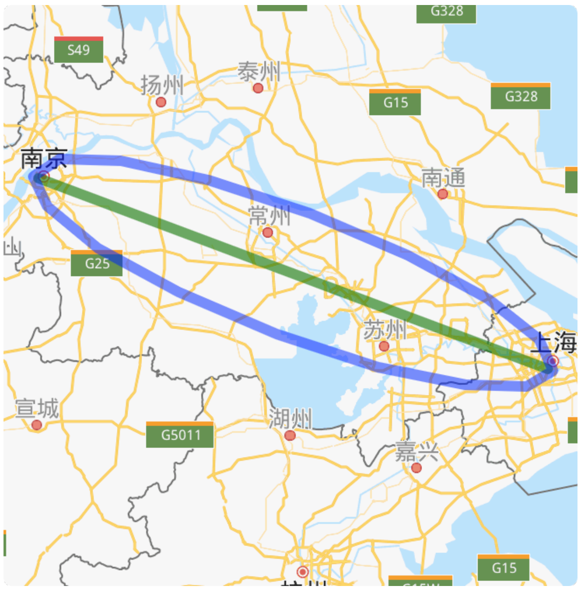
7. ordered and pagination and then return result

```
Select
    source_id, tags,
    ST_AsText(point) as point,
    ST_Distance(point, ST_GeomFromText('POINT(118.78 32.06)', 4326)) as dis-
tance,
    count(*) OVER() AS total_count
From (
    Select *, ST_Contains(polygon, point)
    From POI, ST_GeomFromText('POLYGON((121.486252151999
31.1915758019704,121.348696864095 31.1203804442821,121.024083616797
31.1267700442391,120.56183183438 31.2097718431698,120.032315160394
31.3567495696002,119.516147708373 31.5453271970376,119.091911293818
31.7467954911651,118.824192065132 31.9304827305165,118.753747848001
32.0684241980296,118.891303135905 32.1396195557178,119.215916383203
32.1332299557608,119.67816816562 32.0502281568302,120.207684839606
31.9032504303998,120.723852291627 31.7146728029624,121.148088706182
31.5132045088349,121.415807934868 31.3295172694834,121.486252151999
31.1915758019704))', 4326) as polygon ) as tbl
Where tbl.st_contains = true And True
Order By distance Asc, id Asc
Offset 10
Limit 10
```

By adjusting the "shrink" parameter, we will get the different shape of the ellipse. I list 4 results, only distinguished by shrink level.

GreenLine: route way, by connected input serial points

Blue Line: ellipse, by buffered route way.

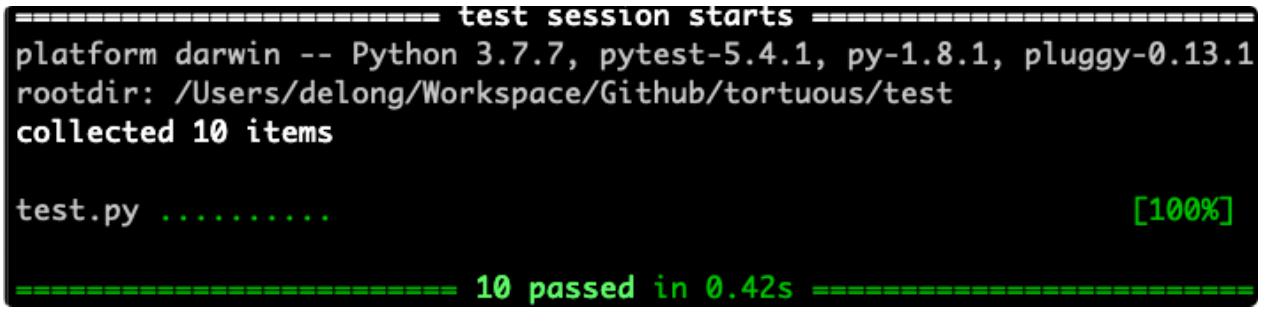
	
shrink=1 it's a circle now	shrink=0.5 it's a ellipse now, but suit for zigzag line, it will miss few section of road
	
shrink=0.1 total missed the way point	shrink=0.1 and only start point and end point it's very suitable for 2 points route

Test

As this is not a complicated project, we do only smoke test and load test, and for convenient we use Python do test functions.

Smoke test: use Pytest[1] lib, write all API http request, and just run it.

Load test: use Locust[2] framework, write a config file and use GUI test functions.


```
========================= test session starts ==========================
platform darwin -- Python 3.7.7, pytest-5.4.1, py-1.8.1, pluggy-0.13.1
rootdir: /Users/delong/Workspace/Github/tortuous/test
collected 10 items

test.py ..........                                              [100%]

========================== 10 passed in 0.42s ==========================
```

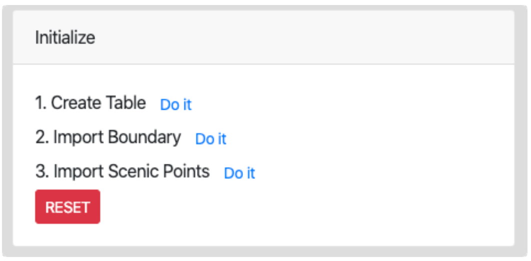
Project Quick Demo

Setup a clean Ubuntu Server, and then copy script down blow and run it

```
sudo su
apt update
apt -y upgrade
apt -y install docker docker-compose
cd /srv
if [ -d "./poi" ];then
    rm -r poi
fi
mkdir poi
cd poi
cat > docker-compose.yml << EOF
version: "3"
services:
    web:
        image: delongw/tortuous:1.0
        restart: always
        ports:
            - "3000:3000"
        depends_on:
            - "postgis"
        env_file: postgres.env
    postgis:
        image: postgis/postgis:11-3.0-alpine
        restart: always
        ports:
            - "5432:5432"
        env_file: postgres.env
EOF
cat > postgres.env << EOF
POSTGRES_PASSWORD=mysecretpassword
POSTGRES_USER=postgres
POSTGRES_DB=postgres
POSTGRES_HOST=postgis
EOF
docker stop $(docker ps -aq)
docker-compose -p poi up -d
echo complete
```

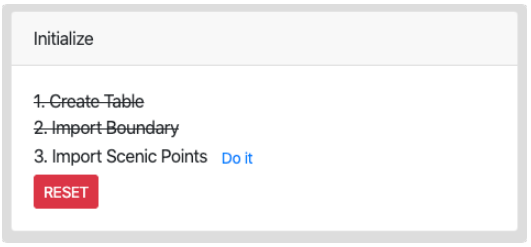
After installation, open your browser and type in http://localhost:3000/toolkit, and congratulation! you already run it.

there was 3 initialize steps, on left panel of page, click "do it" in order.


Initialize

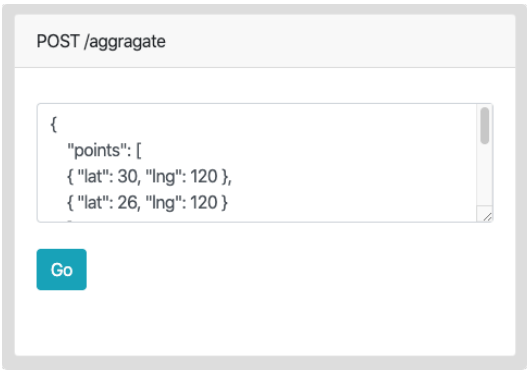
1. Create Table
2. Import Boundary
3. Import Scenic Points

after click, it will do create initial data background and then back up page, mark the step is finished simultaneously.


Initialize

~~1. Create Table~~
~~2. Import Boundary~~
3. Import Scenic Points

after complete initialization. you can enjoy you functions. try make a query request, we already fill some default parameters, just "Go".


POST /aggregate

```
{
   "points": [
   { "lat": 30, "lng": 120 },
   { "lat": 26, "lng": 120 }
```

Conclusion

Location based service need some geometry calculation, that was difficult but also mature. We should choose technology cautions, and use more the basic foundations, mix them, you will find that was very powerful.

As we use single SQL statement for query, and cached polygon request, so the time complexity is $O(N)$. and core function depends on PostGIS, so it naturally support multi-process.

References

[1]: https://docs.pytest.org/en/latest/

[2]: https://locust.io/

[3]: http://www.postgis.net/docs/PostGIS_Special_Functions_Index.html

[4]: Geo Locations of City:

Nanjing: 32.06,118.78

Shanghai: 31.20,121.46

Hangzhou: 30.30,120.15