PROJECT TOPIC –
APPLICATION IN PYTHON FOR LOCATING PERSONS/THINGS.

END TERM REPORT
BY

NAME – ASHISH PATEL
ROLL NO. – 09
SECTION – K18TM
REG NO. – 11804795
GROUP – 1ST

GitHub Link: https://github.com/ap6512068/k18TM-G-1/upload/master

DEPARTMENT OF INTELLIGENT SYSTEMS

SCHOOL OF COMPUTER SCIENCE ENGINEERING

LOVELY PROFESSIONAL UNIVERSITY, PHAGWARA PUNJAB

SUBMITTED BY –

ASHISH PATEL

SUBMITTED TO -

DR. DHANPRATAP SINGH

Student Declaration

This is to declare that this report has been written by me/us. No part of the report is copied from other sources. All information included from other sources have been duly acknowledged. I/We aver that if any part of the report is found to be copied, I/we are shall take full responsibility for it.

NAME – ASHISH PATEL

ROLL NO. – 09

REG. NO. – 11804795

TABLE OF CONTENTS :-

INTRODUCTION *:-*

The project we are creating is a web application that lists shops sorted by distance so your users will be able to discover the shops that are close to their location.

The web application makes use of python for easily implementing location requirements like calculating the distances of shops from the user's location and ordering the shops by distance.

Using python, we can get and display the nearest shops that are stored in a PostgreSQL database configured with the Post GIS extension to enable spatial operations.

SQL Database:- *Now we are created a project, let's continue by configuring the connection to the PostgreSQL and Post GIS spatial database.*

```
DATABASES = {
    'default': {
        'ENGINE': 'django.contrib.gis.db.backends.postgis',
        'NAME': 'gis',
        'USER': 'user001',
        'PASSWORD': '123456789',
        'HOST': 'localhost',
        'PORT': '5432'
    }
}
```

creating a Application:-

A project is made up of applications. By default, it contains several core, but you will usually add at least one app that contains your custom project's code.

The `shops` *application will contain the code for creating and displaying the shops closest to a user's location. In the next steps, you are going to perform the following tasks:*

- *• Create the app*
- *• Add a* `Shop` *model*
- *• Add a data migration for loading initial demo data (shops)*
- *• Add a view function*

- • *Add a template*

Creating a Model :-

After creating the `shops` *application, which will contain the actual code of your project, you need to add models in your app. Code uses an Object Relational Mapper, which is an abstraction layer between code and the database that transforms Python objects or models into database tables.*

In this case, we need one model that represents a shop in the database. We will create a `Shop` *model that has the following fields:*

- `name`*: the name of the shop*
- `location`*: the location of the shop in latitude and longitude coordinates*
- `address`*: the address of the shop*
- `city`*: the city the shop is in*

Open the `shops or models.py` *file and add the following code:*

```
from django.contrib.gis.db import models

class Shop(models.Model):
    name = models.CharField(max_length=100)
    location = models.PointField()
    address = models.CharField(max_length=100)
    city = models.CharField(max_length=50)
```

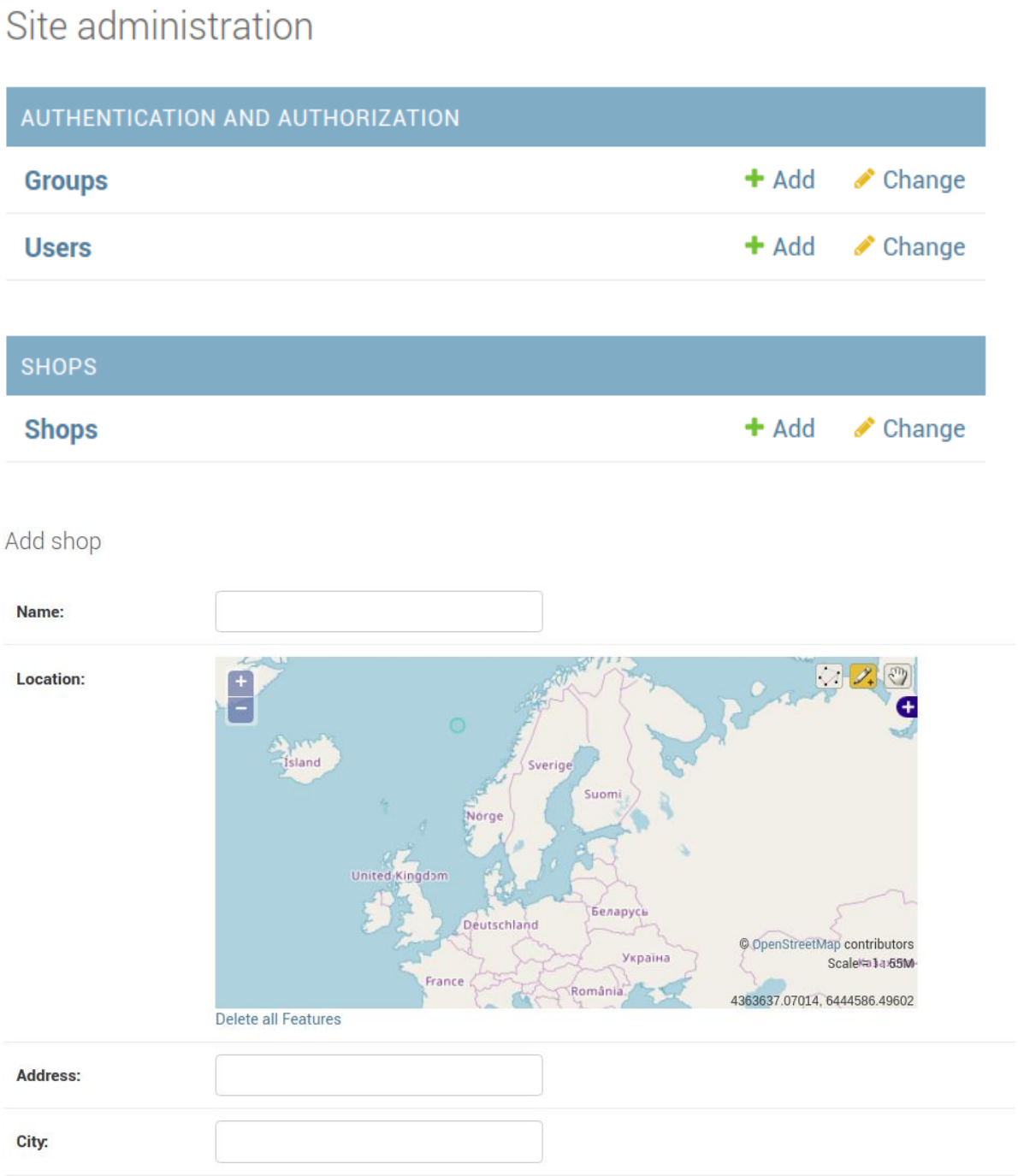
Creating the Database Tables :- *With code, we don't need to use SQL to create the database. Let's create the database tables by using the* `make migrations` *and* `migrate` *commands.*

```
$ python manage.py makemigrations
$ python manage.py migrate
```

Adding a Super User :- *we need to create a super user so you can access the admin interface. This can be done using the following command:*

```
$ python manage.py createsuperuser
```

The prompt will ask you for the username, email, and password you want to use for accessing the user account. Enter them and hit `Enter`*.*

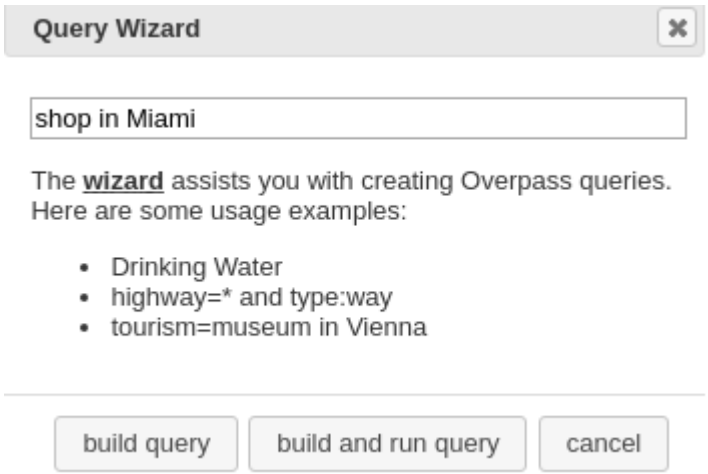


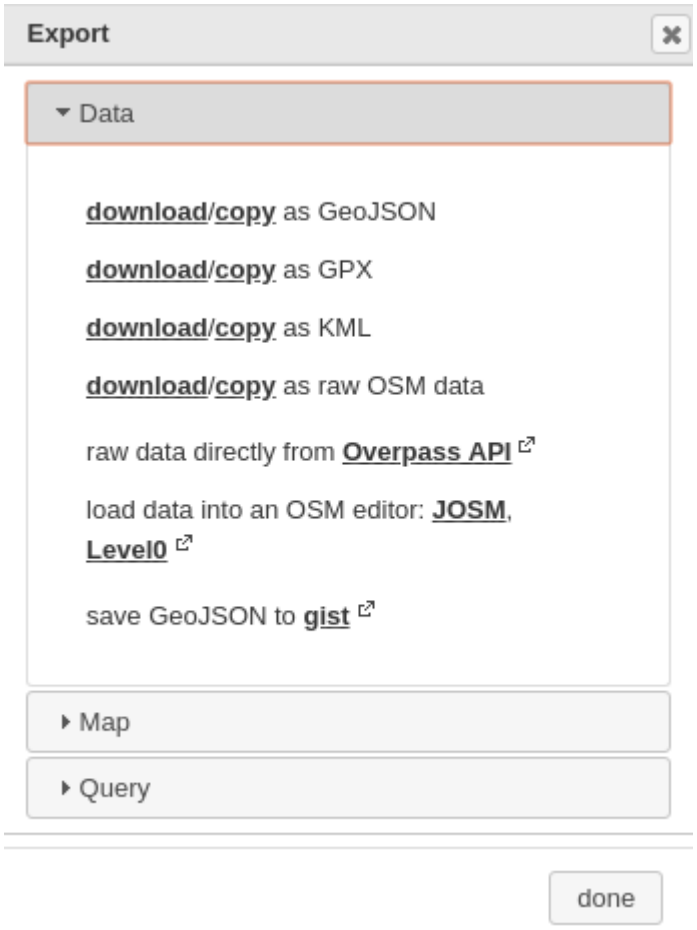
We can see that the `Location` *geometric field is displayed as an interactive map. We can zoom the map in and out, and we can choose different selectors at the top right corner of the map to select a location, which is marked by the green circle.*

Adding Initial Data :-

we need some initial demo data for your application, but instead of manually adding data, you can use a data migration.

Data migrations can be used for multiple scenarios, including adding initial data in your database. For more information, check out data migrations.
In your case, we want to get all the shops in a city. Simply click on the wizard button. A small window will pop up. In the text field, write a query like "shop in Miami" and click on build and run query.





This is a screenshot of example data from the file:

```
{
  "version": 0.6,
  "generator": "Overpass API 0.7.55.4 3079d8ea",
  "osm3s": {
    "timestamp_osm_base": "2018-10-18T20:58:02Z",
    "timestamp_areas_base": "2018-10-18T20:10:02Z",
    "copyright": "The data included in this document is from www.open
  },
  "elements": [
    {
      "type": "node",
      "id": 1535875967,
      "lat": 30.4260072,
      "lon": -9.6199602,
      "tags": {
        "name": "Municipal Fish Market",
        "shop": "seafood"
      }
    },
```


We are importing the `Path` class from the pathlib package for accessing low-level system functions, the json package for working with JSON, the built-in migrations API, and fromstr() , part of the geos package.

```
DATA_FILENAME = 'data.json'
def load_data(apps, schema_editor):
    Shop = apps.get_model('shops', 'Shop')
    jsonfile = Path(__file__).parents[2] / DATA_FILENAME

    with open(str(jsonfile)) as datafile:
        objects = json.load(datafile)
        for obj in objects['elements']:
            try:
                objType = obj['type']
                if objType == 'node':
                    tags = obj['tags']
                    name = tags.get('name','no-name')
                    longitude = obj.get('lon', 0)
                    latitude = obj.get('lat', 0)
                    location = fromstr(f'POINT({longitude}
{latitude})', srid=4326)
                    Shop(name=name, location =
location).save()
```

```
            except KeyError:
                pass
```

we are using the with statement, so you don't have to explicitly close the file, and an f-string for formatting the argument of `fromstr()`.
`fromstr()` *takes a* `srid` *as the second parameter. srid stands for Spatial Reference System Identifier. It's a unique value to identify spatial reference systems (projection systems used for interpreting the data in the spatial database). Next, add the migration class to execute the above function when you run the* `migrate` *command:*

```
class Migration(migrations.Migration):

    dependencies = [
        ('shops', '0005_auto_20181018_2050'),
    ]

    operations = [
        migrations.RunPython(load_data)
    ]
```

That's it. we can now return to your terminal and run the following:

```
$ python manage.py migrate
```

The data from the `data.json` *file will be loaded on your database. Run your Django server and head to your admin interface. You should see your data in the table. In my case, this is a screenshot of a portion of the table:*

NAME	LOCATION
concesionaria	SRID=4326;POINT (-80.1890705 25.796773)
7 eleven	SRID=4326;POINT (-80.20867079999999 25.8468261)
El Gladiolo Florist	SRID=4326;POINT (-80.26399019999999 25.7740443)
bed bath beyond	SRID=4326;POINT (-80.2501864 25.7504642)
O'sole Boutique	SRID=4326;POINT (-80.2435747 25.7272859)
Guantanamera Cigars & Cafe	SRID=4326;POINT (-80.2188683 25.7659209)
GNC	SRID=4326;POINT (-80.25834829999999 25.7792495)
Whole foods	SRID=4326;POINT (-80.18948949999999 25.7724836)
Ricky's Meats & Deli	SRID=4326;POINT (-80.2394322 25.8041931)
Miami Motostop	SRID=4326;POINT (-80.1912227 25.8024988)
Miami Design District Shopping Center	SRID=4326;POINT (-80.1909071 25.8127712)

Displaying Nearby Shops :-

• The `shops` application, which encapsulates the code for creating and getting nearby shop.
• The `Shop` model and the corresponding tables in the database
• The initial admin user for accessing the admin interface
• The initial demo data for loading real-world shops in the database that you can play with without manually entering a lot of fake data

You also registered the `Shop` model in the admin application so you can create, update, delete, and list shops from the admin interface.
Let's start by adding a template and a view function that will be used to display the nearby shops from a user's location.

```
from django.views import generic
from django.contrib.gis.geos import fromstr
from django.contrib.gis.db.models.functions import Distance
from .models import Shop
```

In this part, we will simply hard code the user's location, but this ideally should be specified by the user or retrieved automatically from the user's browser with their permission using JavaScript and html. Finally, add the following view class:

Finally, add the following view class:

```
class Home(generic.ListView):
```

```
    model = Shop
    context_object_name = 'shops'
    queryset =
Shop.objects.annotate(distance=Distance('location',
    user_location)
    ).order_by('distance')[0:6]
    template_name = 'shops/index.html'
```

Class-based views are an alternative way to implement views as Python classes instead of functions. They are used to handle common use cases in web development without re-inventing the wheel. To get the nearby shops, you simply use. now let's add the `shops/index.html` template with the following content:

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8">
        <title>Nearby Shops</title>
    </head>
<body>
    <h1>Nearby Shops</h1>
    {% if shops %}
    <ul>
    {% for shop in shops %}
        <li>
        {{ shop.name }}: {{shop.distance}}
        </li>
    {% endfor %}
    </ul>
    {% endif %}
</body>
</html>
```

The nearest shops are available from the `shops` context object that you specified as the `context_object_name` in the class-based view. You loop through the `shops` object and you display the name and distance from the user's location for each shop.

```
from django.urls import path
from shops import views

urlpatterns = [
    # [...]
    path('', views.ShopList.as_view())
]
```

The home page will display a simple un-styled list with the nearest shops from the hard-coded user's location. This is an example screenshot:

Conclusion :-

On creating your location based web application using python code, which aims to become a world-class geographic framework for implementing GIS apps. Nowadays, location aware apps (apps that know your location and help you discover nearby objects and services by offering you results based on your location) are all the rage.

…… THANK YOU ……..