User Manual

This user manual is to show how one can how to use Django and GeoDjango to build a location-based web application from scratch. How you can Use the GeoDjango sub-framework to implement geolocation features in your Django application and use a spatial database.

Tools used:

- The Python programming language
- The Django web framework
- The PostgreSQL database for persisting data
- The PostGIS extension for supporting spatial features in the PostgreSQL database
- pip for installing dependencies
- The venv module for managing a virtual environment

Prerequisites

In this section, you'll be installing the prerequisites needed before you can bootstrap your project, such as Python 3 and GeoDjango dependencies (GEOS, GDAL, and PROJ.4)

Installing Python 3

There is a big chance that you already have Python 3 installed on your system. If you don't, you can simply head to the official website and download the binaries for your operating system.

Depending on your system, you may also be able to install Python 3 or upgrade it to the latest version if it's already installed by using the official package manager.

If you have a problem installing Python 3 or want more information, you can check the Python 3 Installation & Setup Guide, which provides different ways to install Python 3 on your system.

Finally, you can check if you have Python 3 installed by running the following command:

```
(.venv) PS C:\Users\Mahlatse.Moloto\dev\GeojangoProj\Useracc> python
Python 3.10.2 (tags/v3.10.2:a58ebcc, Jan 17 2022, 14:12:15) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

GeoDjango requires a spatial database and a set of open-source geospatial libraries:

- **GEOS** is an open-source geometry engine and a C++ port of the JTS (Java Topology Suite). It's required by GeoDjango for performing geometric operations.
- **PROJ.4** is an open-source GIS library for easily working with spatial reference systems and projections. You need it because you'll be using PostGIS as the spatial database.

- GDAL is an open-source geospatial data abstraction library for working with raster and vector data formats. It's needed for many utilities used by GeoDjango.

You can refer to the docs for more information about spatial databases and the required libraries.

Refer to the docs for detailed instructions about how to install these dependencies on Windows. below link will guide you:

GeoDjango Installation | Django documentation | Django (djangoproject.com)

Setting up Your Project

Now that you have a spatial database set up and ready, you can go ahead and setup your Django project. In this section, you'll use venv to create an isolated virtual environment for your project and install all the required packages such as Django.

First you need t o go the directory where you will be saving your project

```
C:\Users\Mahlatse.Moloto\dev\GeojangoProj\Useracc> cd C:\Users\Mahlatse.Moloto\dev
```

Create a directory, in my case Django_projets and cd into that project

```
C:\Users\Mahlatse.Moloto\dev\GeojangoProj\Useracc> mkdir  django_projects
```

```
C:\Users\Mahlatse.Moloto\dev\GeojangoProj\Useracc> cd .\django_projects\
```

Creating a Virtual Environment

A virtual environment allows you to create an isolated environment for the dependencies of your current project. This will allow you to avoid conflicts between the same packages that have different versions.

In Python 3, you can create virtual environments using virtualenv or the venv module.

For more information about Python virtual environments, check out Python Virtual Environments: A Primer.

Now, open up your terminal and run the following command to create a virtual environment based on Python 3:

```
C:\Users\Mahlatse.Moloto\dev\GeojangoProj\Useracc> py -3 -m venv .venv
```

Next, you need to activate the following command:

```
PS C:\Users\Mahlatse.Moloto\dev\GeojangoProj\Useracc> .venv\scripts\activatez
```

That's it. You now have your virtual environment activated, and you can install the packages for your project.

Installing Django

The first step after creating and activating a virtual environment is to install Django. The Django package is available from the Python Package Index (so you can simply use pip to install it by running the following command in your terminal:

```
(.venv) PS C:\Users\Mahlatse.Moloto\dev\GeojangoProj\Useracc> python -m pip install django
```

Creating a Django Project

Command to create a project

```
(.venv) PS C:\Users\Mahlatse.Moloto\dev\GeojangoProj\Useracc> django-admin startproject
 Useracc
```

psycopg2-binary in your virtual environment using the following:

```
(.venv) PS C:\Users\Mahlatse.Moloto\dev\GeojangoProj\Useracc> pip install psycopg2 psyc
opg2-binary
```

Adding GeoDjango

GeoDjango is a framework that makes it as easy as possible to build GIS and location aware web applications. You can add it by simply including the gis contrib module in the list of installed apps.

Open the settings.py file and locate the INSTALLED_APPS array. Then add the 'django.contrib.gis' module:

```
Useracc > Useracc > settings.py > ...
# Application definition

INSTALLED_APPS = [

    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'django.contrib.gis',
    'kortazaapp.apps.KortazaappConfig',
]
```

Creating a Django Application

A Django project is made up of applications. By default, it contains several core or built-in apps like django.contrib.admin, but you will usually add at least one app that contains your custom project's code.

Note: For simple projects, you may only need one app, but once your project becomes bigger and has different requirements, you can organize your code in multiple separate apps.

Now that you have created a Django project, configured the connection with the spatial database, and added GeoDjango to the project, you need to create a Django application that you may call shops.

The geoDjango application will contain the code for creating and displaying user's details. 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

First run the following command to create the app:

```
$ python manage.py kortazaapp shops
```

Next, you need to add it to the list of installed apps in the settings.py file, which will make Django recognize it as a part of your project:

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

In this case, you need one model that represents a shop in the database. You'll create a kortazaapp model that has the following fields:

- **Home address:** the address where the user live
- **Phone number:** phone number of the user
- **Location:** in a form of mar/ coordidates

Open the models.py file and add the following code:

```python
from django.db import models
from django.contrib.gis.db import models
from django.contrib.auth.models import User
from PIL import Image

class Userdetails(models.Model):
    user = models.OneToOneField(User, on_delete= models.CASCADE)
    home_address = models.CharField(max_length=200, blank=True)
    phone_number = models.CharField(max_length=10, unique=True)
    location = models.PointField()
```

```python
# Override the save method of the model
def save(self):
    super().save()

    img = Image.open(self.image.path) # Open image

    # resize image
    if img.height > 300 or img.width > 300:
        output_size = (300, 300)
        img.thumbnail(output_size) # Resize image
        img.save(self.image.path) # Save it again and override the larger image
```

Creating the Database Tables

With Django, you don't need to use SQL to create the database tables thanks to its ORM. Let's create the database tables by using the makemigrations and migrate commands. Head back to your terminal and run the following:

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

Adding a Super User

You 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.

Open the admin.py file and add the following code:

```
from msilib.schema import Class

from pyexpat import model

from tabnanny import verbose

from django.contrib import admin

from django.contrib.gis.admin import OSMGeoAdmin

from .models import Userdetails, Userprofile

from django.contrib.auth.models import User

from django.contrib.auth.admin import UserAdmin


class userdetailsadmin(admin.StackedInline):

    model = Userdetails

    can_delete = False

    verbose_name_plural = 'Userdetails'


class CustomizedUserdetails(UserAdmin):

    inlines = (userdetailsadmin, )


admin.site.unregister(User)

admin.site.register(User, CustomizedUserdetails )

admin.site.register(Userprofile)


@admin.register(Userdetails)
class UserAdmin(OSMGeoAdmin):

    list_display = ('home_address', 'location')
```

```
$ python manage.py runserver
```

Open the admin.py file and add the following code:

```python
from django.apps import AppConfig


class KortazaappConfig(AppConfig):
    default_auto_field = 'django.db.models.BigAutoField'
    name = 'kortazaapp'


class UsersConfig(AppConfig):
    name = 'users'

    def ready(self):
        import kortazaapp.signals
```

Open the admin.py file and add the following code:

```python
from django import forms
from django.contrib.auth.models import User
from django.contrib.auth.forms import UserCreationForm
from .models import Userprofile


class UserRegisterForm(UserCreationForm):
    email = forms.EmailField()

    class Meta:
        model = User
        fields = ['username', 'email', 'password1', 'password2']

# Create a UserUpdateForm to update username and email
class UserUpdateForm(forms.ModelForm):
    email = forms.EmailField()

    class Meta:
        model = User
        fields = ['username', 'email']

# Create a ProfileUpdateForm to update image
class ProfileUpdateForm(forms.ModelForm):
    class Meta:
        model = Userprofil
```

```
        fields = ['image']
```

Open the signal.py file and add the following code:

```
from django.db.models.signals import post_save #Import a post_save signal when a
user is created
from django.contrib.auth.models import User # Import the built-in User model,
which is a sender
from django.dispatch import receiver # Import the receiver
from .models import Userprofile


@receiver(post_save, sender=User)
def create_profile(sender, instance, created, **kwargs):
    if created:
        Userprofile.objects.create(user=instance)


@receiver(post_save, sender=User)
def save_profile(sender, instance, **kwargs):
    instance.Userprofile.save()
```

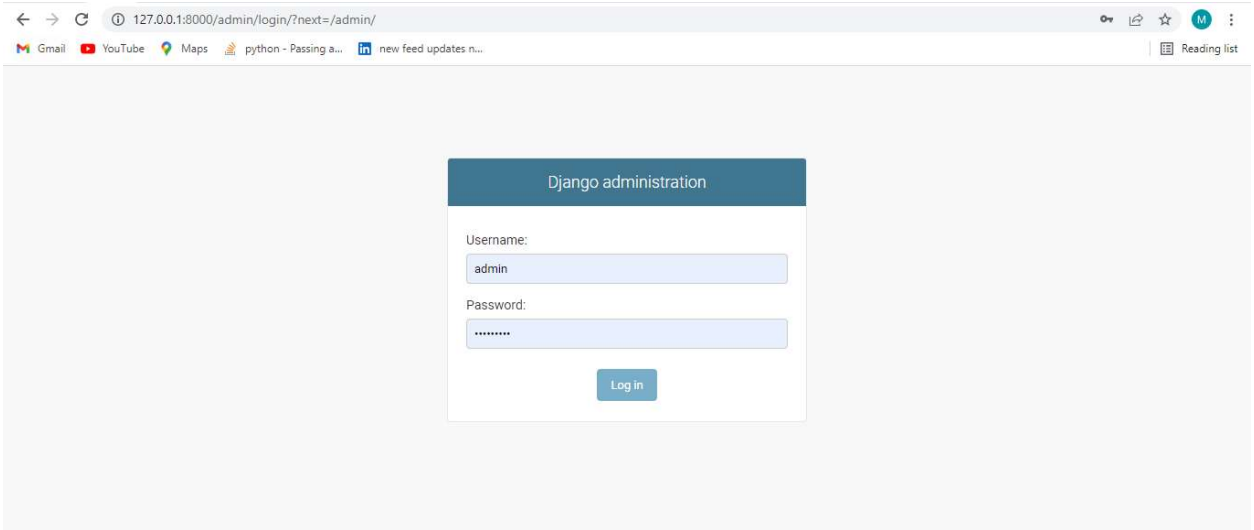
and additional code is on the Kortaza app folder.

To run the application, run the below:

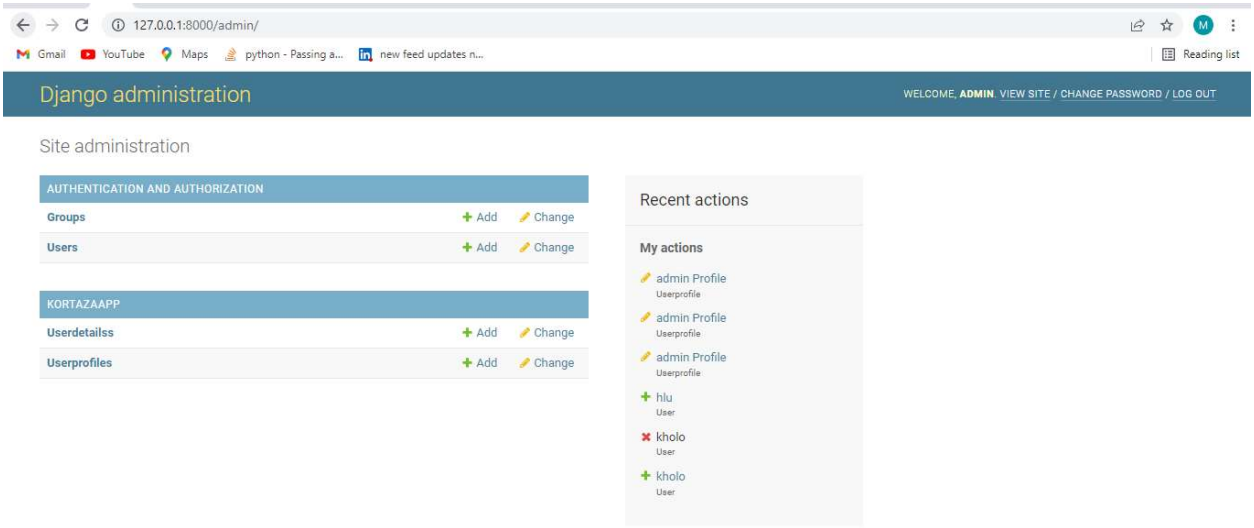
```
$ python manage.py runserver
```

You application will be running from localhost:8000, and you can access the admin interface from localhost:8000/admin.

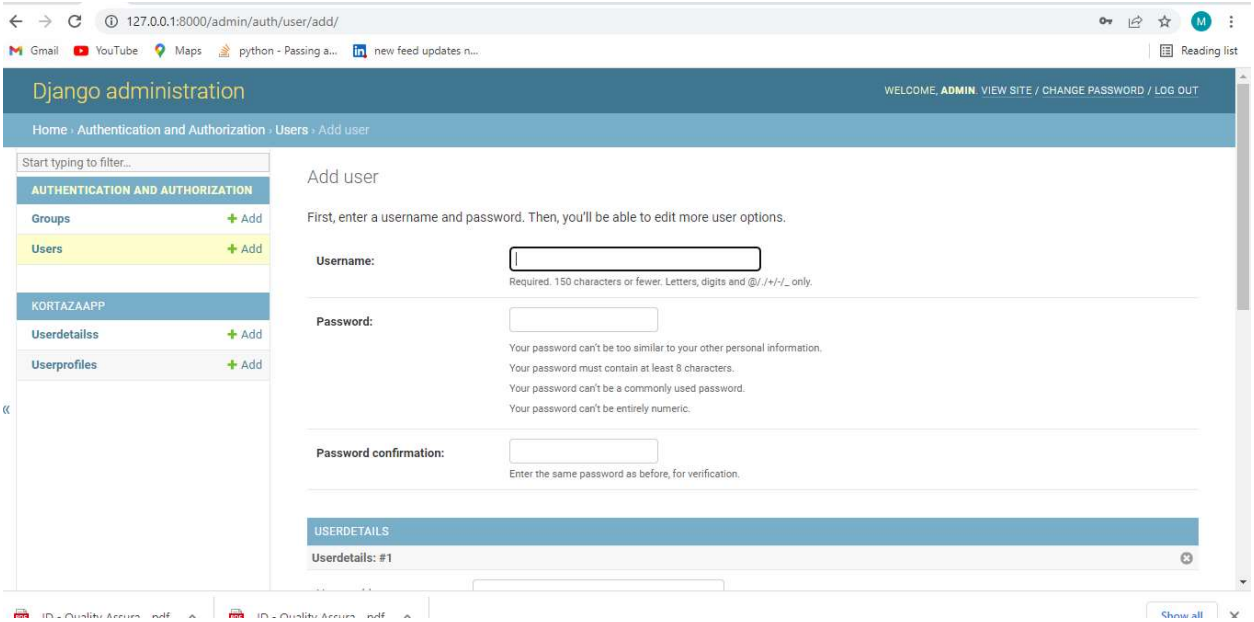
You will be able to login using the username and password



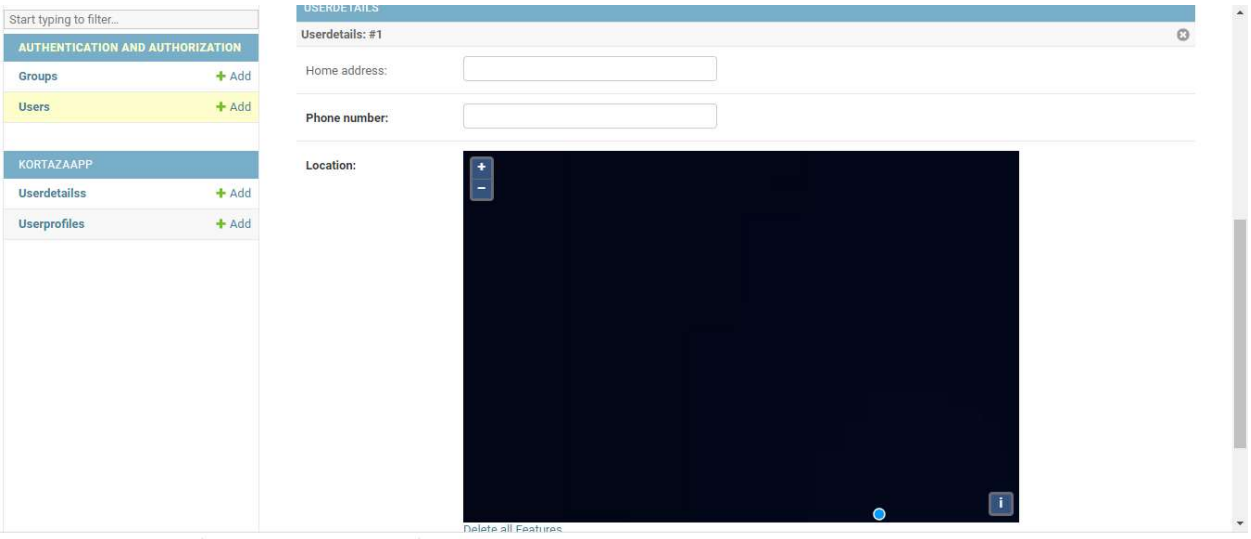
This is a screenshot from the localhost:8000/admin interface:



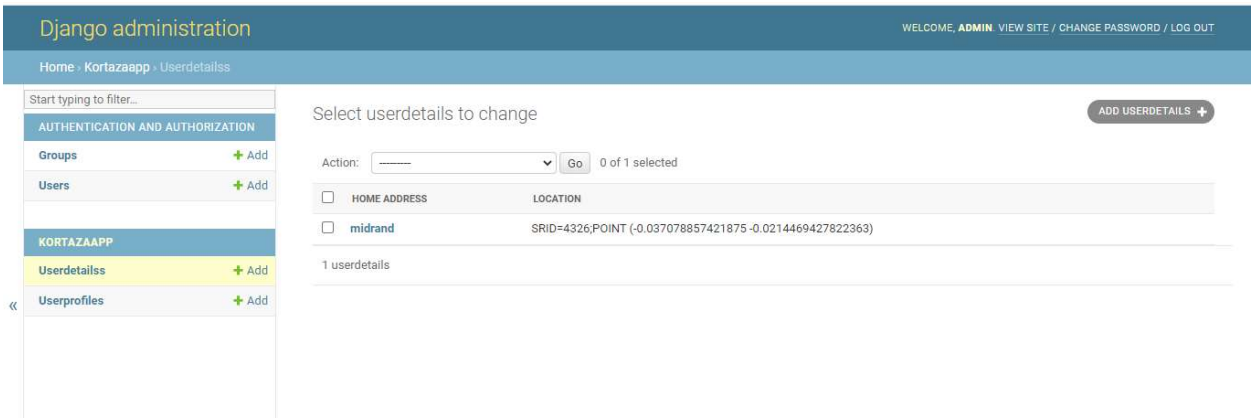
The screenshot below shows the add user page with the url:
http://127.0.0.1:8000/admin/auth/user/add/



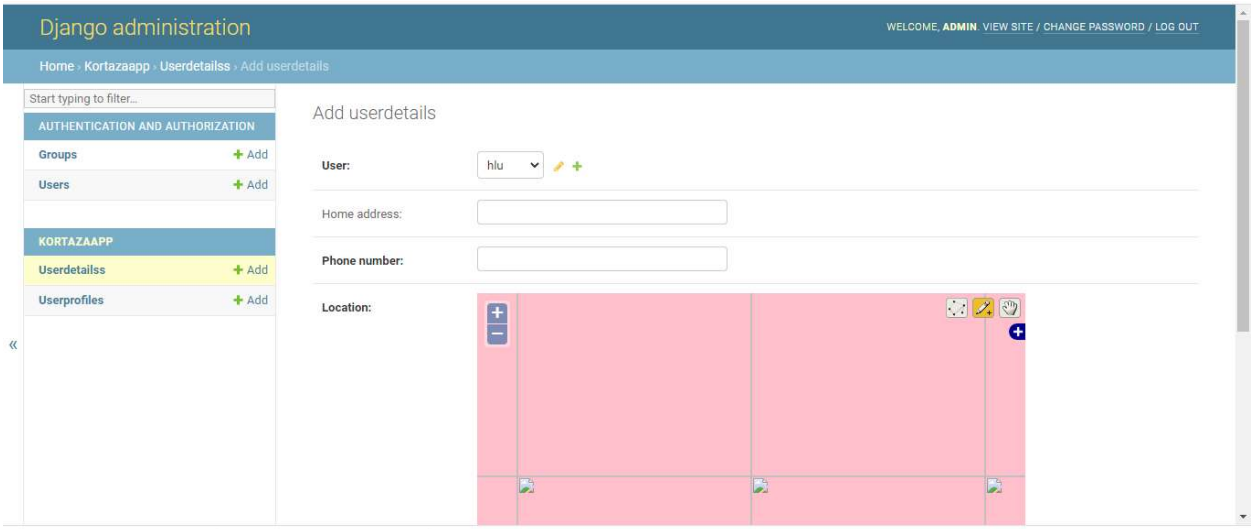
Addition Extend the django user module to add further details for user profile, such as home address, phone number, location (point geometry) where they live



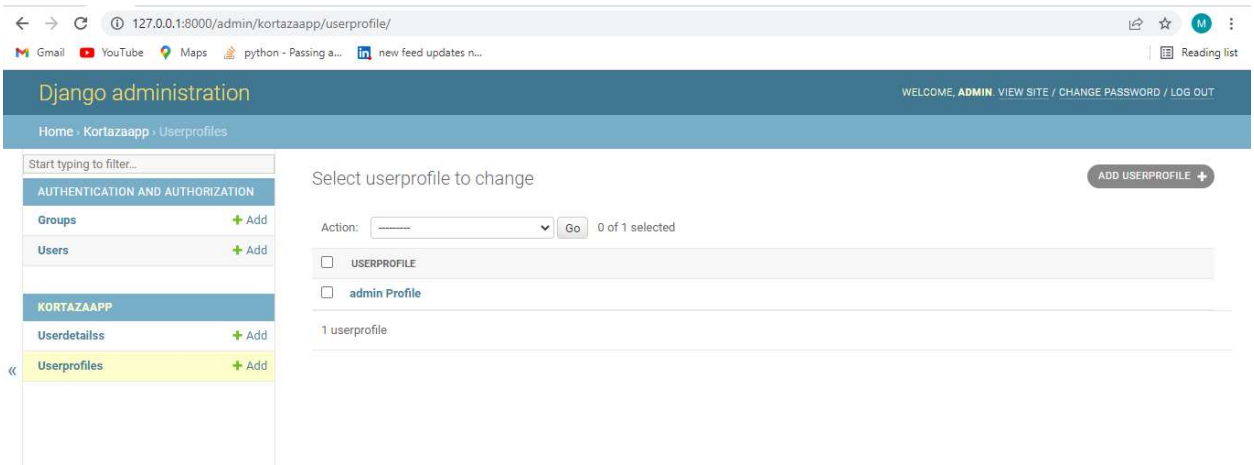
The screenshot below show the add user page with the url: http://127.0.0.1:8000/admin/kortazaapp/userdetails/



The screenshot below shows the add user page with the url:
http://127.0.0.1:8000/admin/kortazaapp/userdetails/add/



The screenshot below shows the add user page with the url:
http://127.0.0.1:8000/admin/kortazaapp/userprofile/



You are able to change password and logout