Django Class Notes

Clarusway

Weather App (API Project)

Nice to have VSCode Extentions:

- Djaneiro - Django Snippets (Be carefull about other conflicting extentions!)

Needs

- Python, add the path environment variable
- pip
- virtualenv

Summary

- Create project
- Secure your project
  - .gitignore
  - django-decouple
- Create app
- Get API to serve weather data
- requests module in Python
- Create City model
- How to get icon URL
- Getting city from user
- Add a delete button for cities
- Next Steps

Create project

- Create a working directory, name it as you wish, cd to new directory
- Create virtual environment as a best practice:

```
python3 -m venv env # for Windows or
python -m venv env # for Windows
virtualenv env # for Mac/Linux or;
virtualenv yourenv -p python3 # for Mac/Linux
```

- Activate scripts:

```
.\env\Scripts\activate  # for Windows
source env/bin/activate  # for MAC/Linux
```

- See the (env) sign before your command prompt.
- Install django:

```
pip install django
```

- See installed packages:

```
pip freeze

# you will see:
asgiref==3.3.4
Django==3.2.4
pytz==2021.1
sqlparse==0.4.1

# If you see lots of things here, that means there is a problem with your virtual
env activation.
# Activate scripts again
```

- Create requirements.txt same level with working directory, send your installed packages to this file, requirements file must be up to date:

```
pip freeze > requirements.txt
```

- Create project:

```
django-admin startproject weather .
# With . it creates a single project folder.
# Avoiding nested folders
```

- Various files has been created!
- Check your project if it's installed correctly:

```
python manage.py runserver
py -m manage.py runserver
```

Secure your project

.gitignore

Add standard .gitignore file to the project root directory.

Do that before adding your files to staging area, else you will need extra work to unstage files to be able to ignore them.

python-decouple

- To use python decouple in this project, first install it:

```
pip install python-decouple
```

- For more information about python-decouple
- Import the config object on `settings.py` file:

```
from decouple import config
```

- Create .env file on root directory. We will collect our variables in this file.

```
SECRET_KEY=o5o9...
```

- Retrieve the configuration parameters in `settings.py`:

```
SECRET_KEY = config('SECRET_KEY')
```

- Now you can send you project to the github, but be sure you added a .gitignore file which has .env on it.

Create app

- Start app

```
python manage.py startapp weatherapp
```

- Go to `settings.py` and add the app to the INSTALLED_APPS:

```
'weatherapp'
```

- Migrate tables:

```
python manage.py migrate
```

- In order to login admin site, we need to create a user who can login to the admin site. Run the following command:

```
python manage.py createsuperuser  # or with the parameters
python manage.py createsuperuser --username admin --email admin@mail.com
```

- Enter your desired username, email adress, and password twice.
- Run server.
- Go to http://127.0.0.1:8000/admin/ You should see the admin's login screen.
- Even though we will not create so many apps, it is a best practice to build a loosely coupled url architecture. Go to urls.py and add:

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include("weatherapp.urls")),
]
```

- Create urls.py under weatherapp, and add:

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

urlpatterns = [
    path('', home, name="home"),
]
```

- Create your view to comply with the template:

```
from django.shortcuts import render

def home(request):
    return render(request, "weatherapp/home.html")
```

- Create your template under weatherapp/templates/weatherapp/

```
<h1>Weather App</h1>
```

Get API to serve weather data

- Go to the openweather webpage. This is the place where we get the free weather API.
- Create an account if you do not have one, and sign in.
- Go to API page. There are a lot of services like forecast, historical data, weather alerts, and current data.
- Go to Current weather data API doc. In this project, we will show only the current weather to our customers.
- Select from right side menu and go to Built-in API request by city name. There are other options, but we will serve weather info using city name.
- We will use the endpoint:

```
https://api.openweathermap.org/data/2.5/weather?q={city name}&appid={API key}
```

- We need city name and API key.
- Go to API keys section on your profile page.
- Add your API key to .env file.

```
API_KEY=202a624b8955a589b89584f57ee522
```

- Add a variable named "url" to your view. This will be the API endpoint according to the the openweather API documentation:

```
# def home(request):
    url = 'https://api.openweathermap.org/data/2.5/weather?q={city name}&appid=
{API key}'
    # return render(request, "weatherapp/home.html")
```

- We will add `&units=metric` at the end to see results in celcius degree format.
- We need to add city name and API key variables inside this url. So, create city_name and API_key variables before that.
- API_key will be imported from .env file using decouple.
- At first, let's use a hardcoded city name as Amsterdam.
- Now our view is like:

```
from decouple import config

def home(request):
    city_name = 'Amsterdam'
    API_key = config('API_KEY')

    url = f'https://api.openweathermap.org/data/2.5/weather?q={city_name}&appid=
{API_key}&units=metric'

    return render(request, "weatherapp/home.html")
```

requests module in Python

- Requests is an HTTP library, written in Python, for human beings. Basic GET usage:

```
import requests

r = requests.get('https://www.python.org')
r.status_code
200
b'Python is a programming language' in r.content
True
r.headers['content-type']
'application/json; charset=utf8'
r.encoding
'utf-8'
r.text
'{"authenticated": true, ...'
```

```
r.json()
{'authenticated': True, ...}
```

- Requests allows you to send HTTP/1.1 requests easily. There's no need to manually add query strings to your URLs, or to form-encode your PUT & POST data — but nowadays, just use the json method! There are a lot more to do with requests module. Look at the examples in requests_examples.py.
- Use requests in your view:

```
import requests

def home(request):
    url = f'https://api.openweathermap.org/data/2.5/weather?q={city_name}&appid=
{API_key}&units=metric'

    # First get the data with requests module:
    r = requests.get(url)

    # Use json() method to convert incoming JSON data to a python dictionary.
    content = r.json()

    return render(request, "weatherapp/home.html")
```

- Now, we have weather info as a Python dictionary. To see the details, we may use pprint. The pprint module provides a capability to "pretty-print" arbitrary Python data structures in a form which can be used as input to the interpreter.

```
from pprint import pprint

pprint(content)
```

- Need to keep cities in our db, users will choose city to see weather info, we will show weather data since user does not delete city from the db.

Create City model

- Let's create a simple model for cities:

```
class City(models.Model):
    name = models.CharField(max_length=100)

    def __str__(self):
        return self.name
```

- Register models from `admin.py`:

```
from .models import City

admin.site.register(City)
```

- Deal with migrations:

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

- Create a couple of cities in admin page.
- Now, we do not need the hardcoded city anymore. We can get it from our db.

```
from .models import City

def home(request):
    # city_name = 'Amsterdam'
    cities = City.objects.all()

    # make a loop to assign all cities to city_name
    for city in cities:
        url = f'https://api.openweathermap.org/data/2.5/weather?q={city}&appid=
{API_key}'
        r = requests.get(url)
        content = r.json()
```

- Things I want to show on my page are:
    - Name of the city from db
    - Description,
    - Icon,
    - Temperature.
- Add those things as data to our view. In this step, there may be multiple cities, so we need to add all city infos inside a list, later we can iterate on this list and show data as context in our template.

```
from django.shortcuts import render
import requests
from decouple import config
from pprint import pprint
from .models import City

def home(request):
    API_key = config('API_KEY')

    cities = City.objects.all()
```

```
    # Create an empty list before and use append method to add things
    city_data = []

    for city in cities:
        url = f'https://api.openweathermap.org/data/2.5/weather?q={city}&appid=
{API_key}&units=metric'
        r = requests.get(url)
        content = r.json()

        data = {
            'city': city,
            'temp': content.get('main').get('temp'),
            'desc': content['weather'][0]['description'],
            'icon': content['weather'][0]['icon'],
        }
        city_data.append(data)

    context = {
        'city_data': city_data,
    }
    return render(request, "weatherapp/home.html", context)
```

- Let's add city_data to template:

```
<h1>Weather App</h1>

{% for city in city_data %}
    {{ city.city.name }} <br>
    {{ city.temp }} <br>
    {{ city.desc }} <br>
    <hr>
{% endfor %}
```

How to get icon URL

- There is a description on document about getting the icons. This is an endpoint like:

```
http://openweathermap.org/img/wn/10d@2x.png
```

- We need to modify this endpoint to a dinamic one which accepts different icons.

```
http://openweathermap.org/img/wn/{{ city.icon }}.png
```

- Time to add this img tag to our template:

```
<img src="http://openweathermap.org/img/wn/{{ city.icon }}.png" alt="Icon">
```

- And also we may add a celcious degree icon next to temperature with html code `<span>&#8451;</span>`:

```
{{ city.temp }} <span>&#8451;</span>
```

Getting city from user

- Users can visit our page, write a city name to see weather condition.
- First we need a form to accept city name as user input.
- Add the form to template:

```
<form action="" method="get">
    <input type="text" name="name" id="">
    <input type="submit" value="Add">
</form>
```

- We can grab that input using:

```
user_city = request.GET.get('name')
```

- Second, we need to get this user input using our view.
    - If there is no city name like the input, give a worning message.
    - If city name is ok, and there is a city already exists on our page, give another worning
    - If city name is ok, and there is not such a city on the page, return a success message
- Add this conditional to our view:

```
from django.contrib import messages

    user_city = request.GET.get('name')
    if user_city:
        url = f'https://api.openweathermap.org/data/2.5/weather?q=
{user_city}&appid={API_key}&units=metric'
        r = requests.get(url)
        if r.ok:  # if response.status_code == 200:
            content = r.json()
            response_city = content["name"]
            if City.objects.filter(name=response_city):
                messages.warning(request, 'City already exists!')
```

```
            else:
                City.objects.create(name=response_city)
                messages.success(request, 'City successfully created!')
        else:
            messages.warning(request, 'City name not found!')
```

- Then, we need to add messages to our template:

```
{% if messages %}
    {% for message in messages %}
        {{ message }}
    {% endfor %}
{% endif %}
```

- After the last querry, we need to redirect to home page for not to see parameters on the url. Add a simple redirect at the end of the city creation.

```
return redirect('home')
```

Add a delete button for cities

- Users may want to delete some cities. To add this function we need to create a delete view and a button to template.

- First create the view:

```
from django.shortcuts import get_object_or_404

def delete(request, id):
    city = get_object_or_404(City, id=id)
    city.delete()
    messages.success(request, 'City successfully deleted!')
    return redirect('home')
```

- Second, add correspanding url path:`

```
path('delete/<int:id>', delete, name="delete"),
```

- Lastly, add a button to template having a link to delete path:

```
<button>
    <a href="{% url 'delete' city.city.id %}">Delete</a>
```

```
</button>
```

Next Steps

- Continue designing frontend
- Learn web scraping with requests module
- Use different API's from openweather
- Use API's from other sources

Frontend Example

- A good home page template is below, you may need to slightly modify django variables inside it:

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css"
rel="stylesheet"
        integrity="sha384-
1BmE4kWBq78iYhFldvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqyl2QvZ6jIW3"
crossorigin="anonymous">
    <title>WeatherOfMine</title>
</head>

<body class="m-0 p-0">

    <div class="messages">
        {% if messages %}
        <div class="">
            {% for message in messages %}
            <p{% if message.tags %} class="text-center alert alert-{{ message.tags
}}" {% endif %}>
                {% if message.level == DEFAULT_MESSAGE_LEVELS.ERROR %}Important:
{% endif %}
                {{ message }}
            </p>
            {% endfor %}
        </ul>
        {% endif %}
    </div>

    <div class="container">

        <div class="row">

            <div class="col-md-6">
```

```
                <h1 class="d-2 text-center m-5">Weather App</h1>

                <form class="d-grid gap-4 mb-5 shadow p-5">
                    <input class="form-control mt-3" type="text" name="city"
placeholder="City name">

                    <input class="btn btn-success mb-3" type="submit" value="ADD">
                </form>


            </div>

            <div class="col-md-6 mt-0 mt-md-5">
                <div class="container">
                    {% for city in city_data %}

                    <div class="row">
                        <div class="col-6">
                            <h3>{{ city.city }}</h3>
                            <span>{{ city.temp }} <span>&#8451;</span></span>
                        </div>

                        <div class="col-6">
                            <h6>{{ city.desc }}</h6>
                            <img src="http://openweathermap.org/img/w/{{ city.icon
}}.png" alt="icon">
                            <a href="{% url 'delete' city.city.id %}" class="btn
btn-danger">Del</a>
                        </div>
                    </div>
                    <hr>

                    {% endfor %}
                </div>
            </div>

        </div>
    </div>

    <script
src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.10.2/dist/umd/popper.min.js"
        integrity="sha384-
7+zCNj/IqJ95wo16oMtfsKbZ9ccEh31eOz1HGyDuCQ6wgnyJNSYdrPa03rtR1zdB"
crossorigin="anonymous">
    </script>
    <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.min.js"
        integrity="sha384-
QJHtvGhmr9XOIpI6YVutG+2QOK9T+ZnN4kzFN1RtK3zEFEIsxhlmWl5/YESvpZ13"
crossorigin="anonymous">
    </script>

</body>
```

```
</html>
```

- With this template, add `settings.py` this variable to display error message in red color:

```
from django.contrib.messages import constants as messages
MESSAGE_TAGS = {
    messages.ERROR: 'danger',
}
```

☺ **Happy Coding!** ✍

Clarusway