Assignment Cover Letter

(Individual Work)

Student Information:		**Surname**	**Given Names**	**Student ID Number**
	1.	**Otnieliem**	**Kevin Herman**	**2301891550**

Course Code	**: COMP6502**	**Course Name**	**: Introduction to Programming**
Class	**: L1AC**	**Name of Lecturer(s)**	**: Ida Bagus Kerthyayana Manuaba**
Major	**: CS**		
Title of Assignment (if any)	**: MyWeather website**		
Type of Assignment	**: Final Project**		
Submission Pattern			
Due Date	**: 13-01-20**	**Submission Date**	**: 13-01-20**

“MyWeather”

Name : Kevin Herman Otnieliem

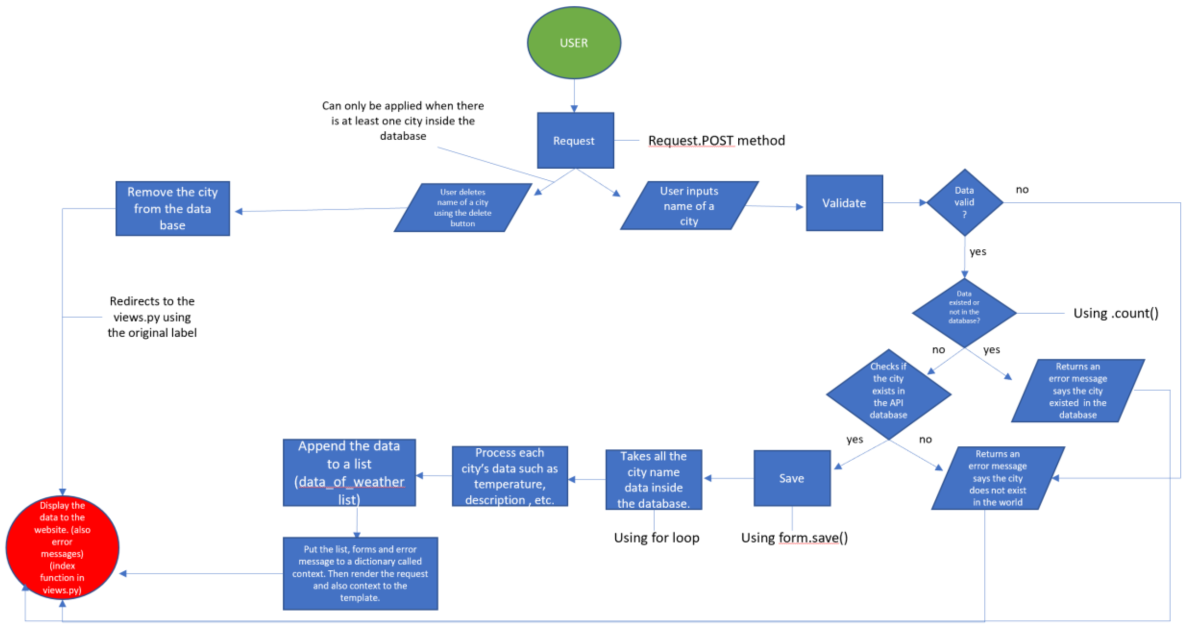
ID : 2301891550

I. Description

Project specification:

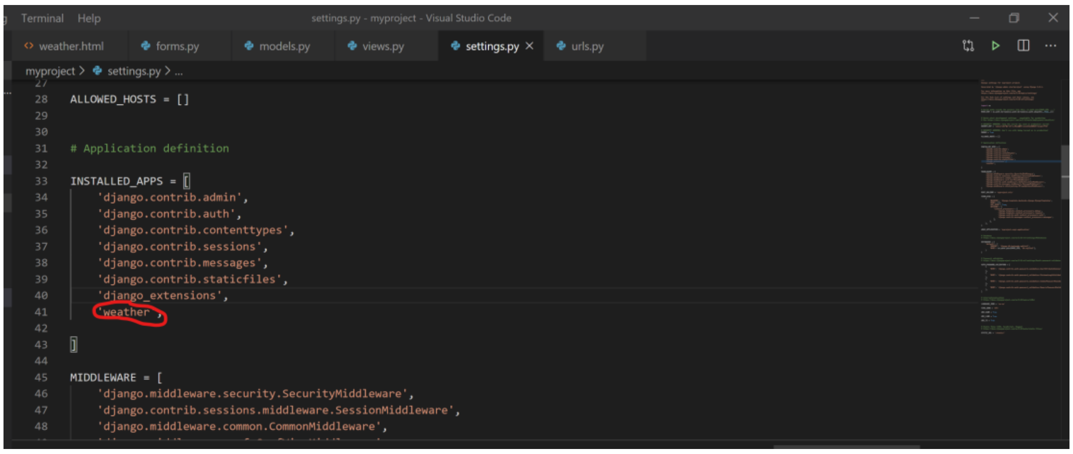
The purpose of making this website is to help people to know how is the weather around the world. This website uses API from openweathermap.org to take the data and display it in MyWeather website using Django. User just need to input the name of the city, then if the city exists it will automatically display in the webpage but if it doesn’t then it will return a warning notification. The website uses html to display the data from python.

II. Solution Design


USER
Can only be applied when there is at least one city inside the database
Request
Request.POST method
Remove the city from the data base
User deletes name of a city using the delete button
User inputs name of a city
Validate
Data valid ?
no
yes
Data existed or not in the database?
Using .count()
no
yes
Checks if the city exists in the API database
Returns an error message says the city existed in the database
yes
no
Redirects to the views.py using the original label
Append the data to a list (data_of_weather list)
Process each city’s data such as temperature, description , etc.
Takes all the city name data inside the database.
Save
Returns an error message says the city does not exist in the world
Using for loop
Using form.save()
Display the data to the website. (also error messages) (index function in views.py)
Put the list, forms and error message to a dictionary called context. Then render the request and also context to the template.

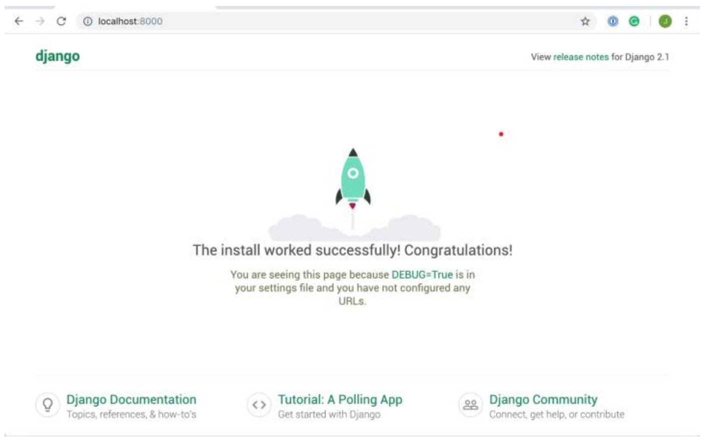
III. Explanation

First of all I create a virtual environment in command prompt with the format python -m venv env then activate it. To activate it I write env\Scripts\activate.bat in the command prompt. Then, it will automatically activate your virtual environment named env in the left side of the new line in command prompt. The importance of using virtual environment is to help making a new environment for my project where it won't mix one another if I have a new project later on, so it prevents my projects to crash one another. Then I create a new project with this format django-admin startproject myproject in command prompt. So now I have a new project named myproject. Inside the myproject folder I have a file named manage.py then I cut and paste it outside of the folder so that I can easily manage my file to make my app. Then I start an app named weather with the format python manage.py startapp weather. After making my app named weather, I then register my app in the setting.py file in the INSTALLED_APPS list, inside myproject folder.


Terminal Help
settings.py - myproject - Visual Studio Code
weather.html
forms.py
models.py
views.py
settings.py
urls.py
myproject > settings.py > ...

```python
ALLOWED_HOSTS = []


# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'django_extensions',
    'weather',

]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
```

Then I test if my project run or not by starting the server with python manage.py runserver. After that, user should go to 'http://127.0.0.1:8000/' url.

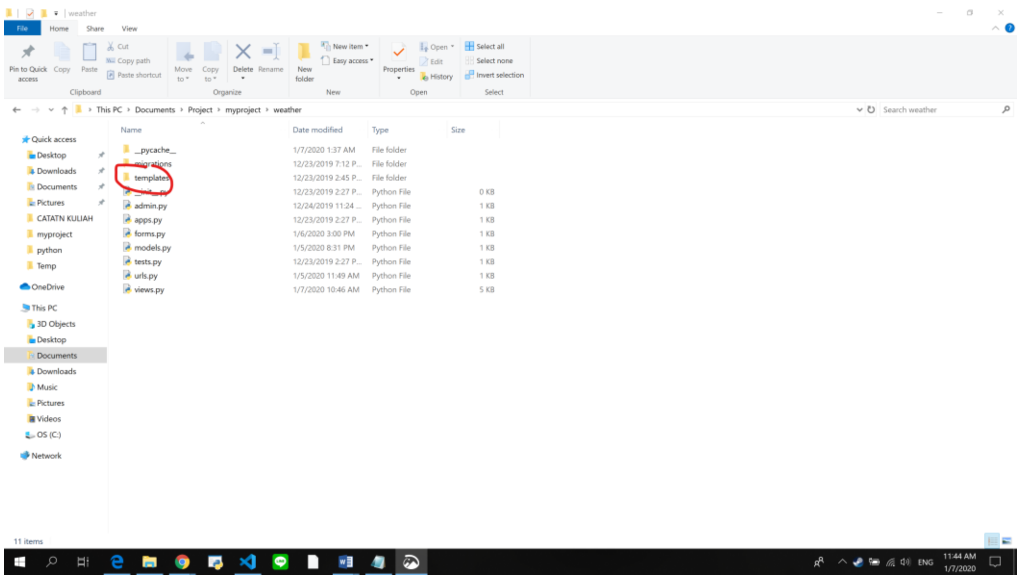


The install worked successfully! Congratulations!

You are seeing this page because DEBUG=True is in your settings file and you have not configured any URLs.

If the user get the same page as the example above it means that our Django project has been successfully run.

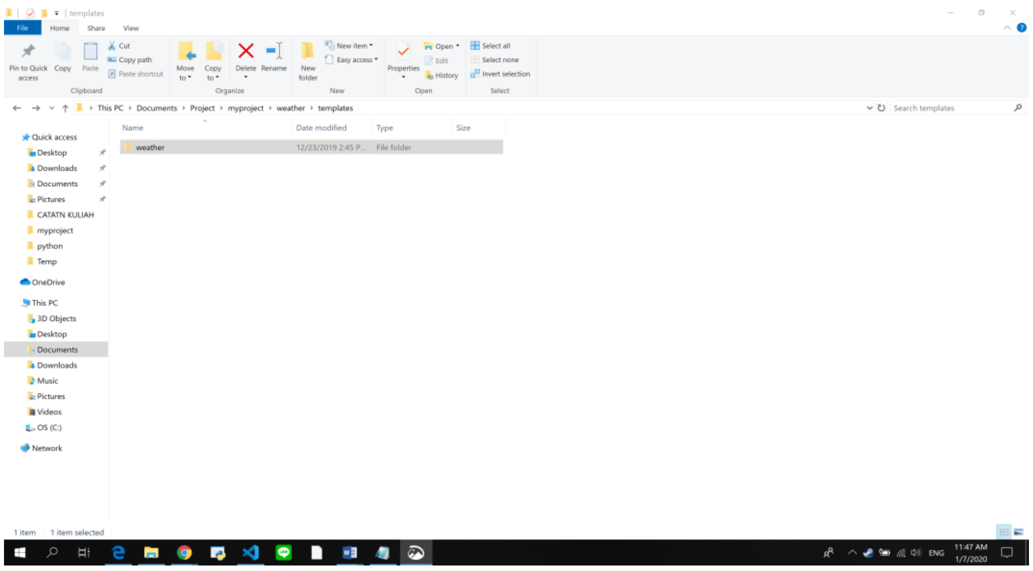
To be honest, I used a html file named weather.html from the github repository that I use it as my reference in this project. So what I do next is to create a new folder named tempelate inside weather app folder, create a new folder again named weather and paste the html file into it, so I can use it become my template for my website later on.



This PC › Documents › Project › myproject › weather

Name	Date modified	Type	Size
__pycache__	1/7/2020 1:37 AM	File folder	
migrations	12/23/2019 7:12 P...	File folder	
templates	12/23/2019 2:45 P...	File folder	
__init__.py	12/23/2019 2:27 P...	Python File	0 KB
admin.py	12/24/2019 11:24 ...	Python File	1 KB
apps.py	12/23/2019 2:27 P...	Python File	1 KB
forms.py	1/6/2020 3:00 PM	Python File	1 KB
models.py	1/5/2020 8:31 PM	Python File	1 KB
tests.py	12/23/2019 2:27 P...	Python File	1 KB
urls.py	1/5/2020 11:49 AM	Python File	1 KB
views.py	1/7/2020 10:46 AM	Python File	5 KB

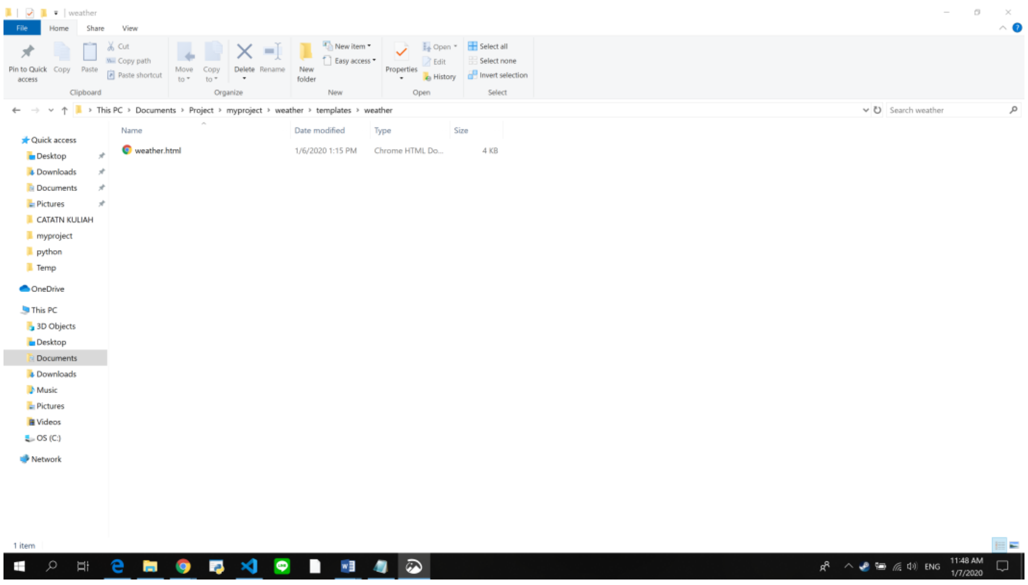
11 items



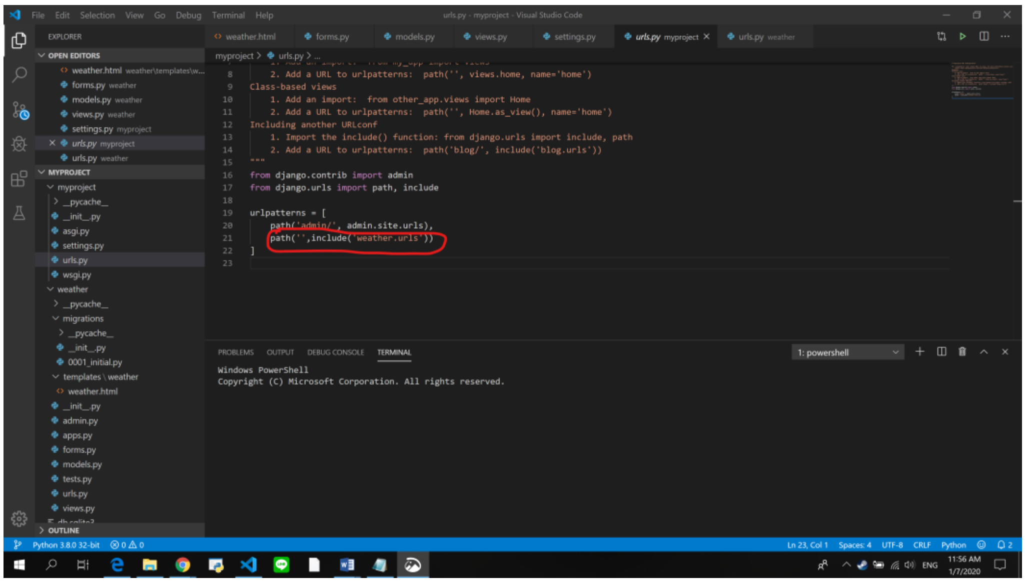
This PC › Documents › Project › myproject › weather › templates

Name	Date modified	Type	Size
weather	12/23/2019 2:45 P...	File folder	

1 item   1 item selected



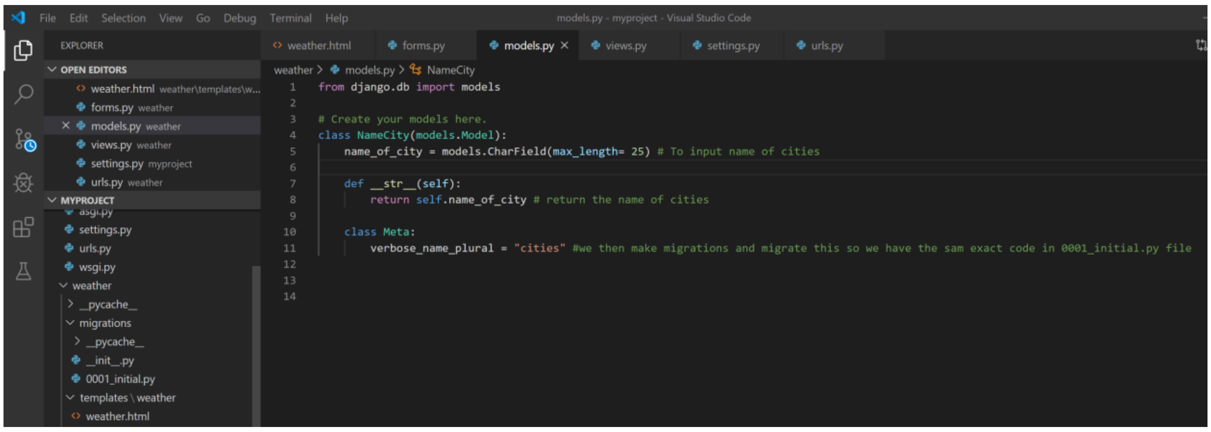
Then I add a url path of the weather app in the myproject folder file named urls.py. I first import include from django.urls and use it in the line like the example below. I had an empty string inside the path because I don't have additional endpoint. Django will now redirect everything that comes into 'http://127.0.0.1:8000/' to weather.urls and looks for further instructions there.



```
    2. Add a URL to urlpatterns:  path('', views.home, name='home')
Class-based views
    1. Add an import:  from other_app.views import Home
    2. Add a URL to urlpatterns:  path('', Home.as_view(), name='home')
Including another URLconf
    1. Import the include() function: from django.urls import include, path
    2. Add a URL to urlpatterns:  path('blog/', include('blog.urls'))
"""
from django.contrib import admin
from django.urls import path, include

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

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.
```

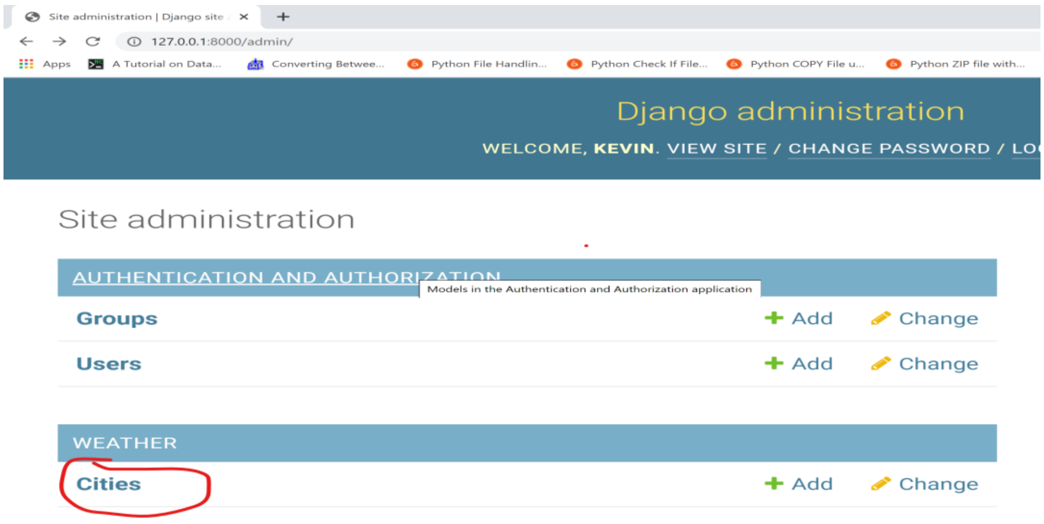
- Models.py



```python
from django.db import models

# Create your models here.
class NameCity(models.Model):
    name_of_city = models.CharField(max_length= 25) # To input name of cities

    def __str__(self):
        return self.name_of_city # return the name of cities

    class Meta:
        verbose_name_plural = "cities" #we then make migrations and migrate this so we have the sam exact code in 0001_initial.py file
```

In this part, I import models from Django.db then I create a class with NameCity label which inherits from Model (a parent class included in Django that defines the basic functionality of a model). Only one attribute is in the NameCity class called name_of_city. The name_of_city attribute is a Charfield (a piece of data that's made up of characters, or text). The website uses CharField to store a small amount of text, in this case name of a city. When I define a CharField attribute, I have to tell Django how much space it should reserve in the database. Here I give it a max_length of 25 characters, which should be enough to hold name of cities around the world. Then I use a __str__() method to display a simple presentation of the model I create. In this code, I define __str__ method to that returns the string stored in name_of_city attribute. I also use inner class called Meta, the purpose of creating meta class is to change the city plural form in the admin site of the website, if its more than one city the admin page will display cities instead of citys.

- Admin site of website



Django administration

WELCOME, **KEVIN**. VIEW SITE / CHANGE PASSWORD / LO

Site administration

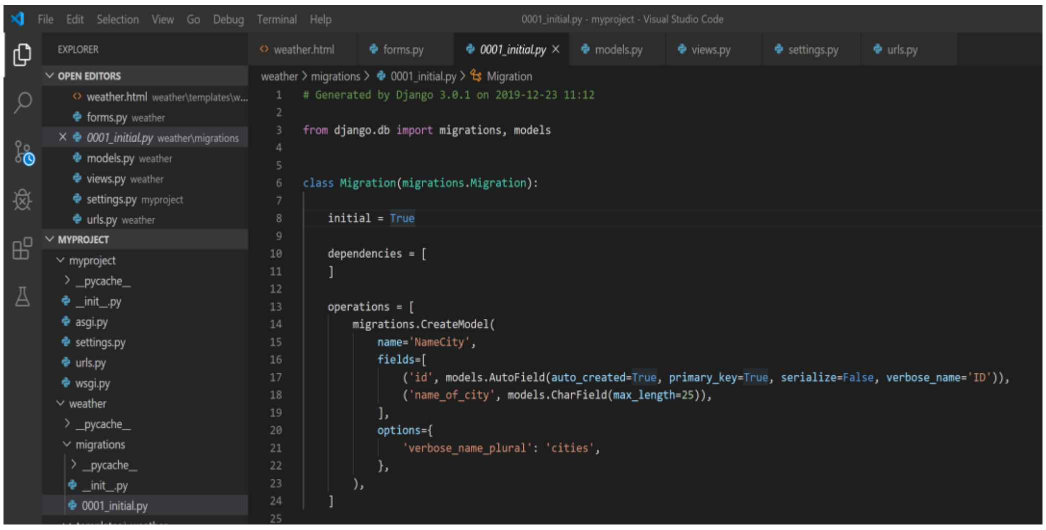
AUTHENTICATION AND AUTHORIZATION		
Groups	+ Add	Change
Users	+ Add	Change

Models in the Authentication and Authorization application

WEATHER		
Cities	+ Add	Change

Then after I write this code I make new migrations by calling it in command prompt, with the format $ python manage.py makemigrations then I migrate it with the format python manage.py migrate .in the migration folder I will get a new file named 0001_initial.py. The command makemigrations tells Django to figure out how to modify the database so it can store the data associated with any new models I've defined. This migration will create a table for the model NameCity in the database.

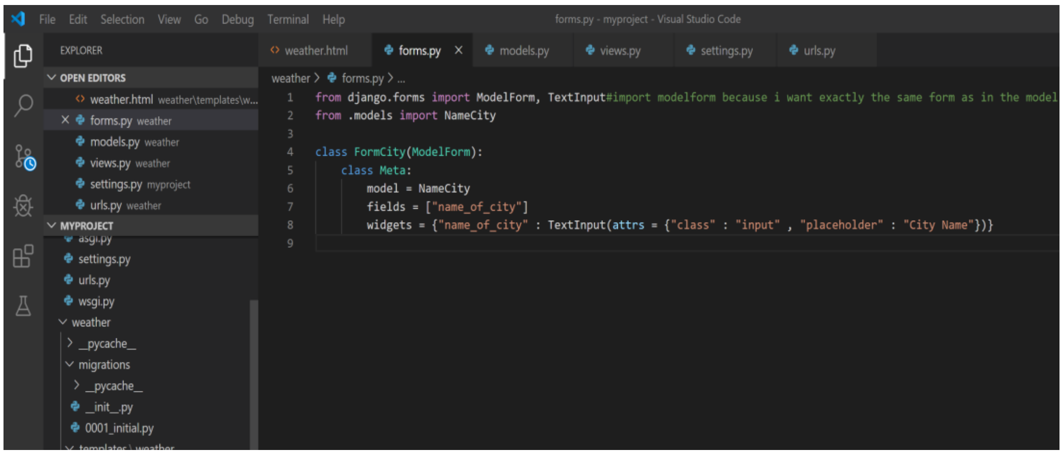
- 0001_initial.py



```python
# Generated by Django 3.0.1 on 2019-12-23 11:12

from django.db import migrations, models


class Migration(migrations.Migration):

    initial = True

    dependencies = [
    ]

    operations = [
        migrations.CreateModel(
            name='NameCity',
            fields=[
                ('id', models.AutoField(auto_created=True, primary_key=True, serialize=False, verbose_name='ID')),
                ('name_of_city', models.CharField(max_length=25)),
            ],
            options={
                'verbose_name_plural': 'cities',
            },
        ),
    ]
```

- Forms.py

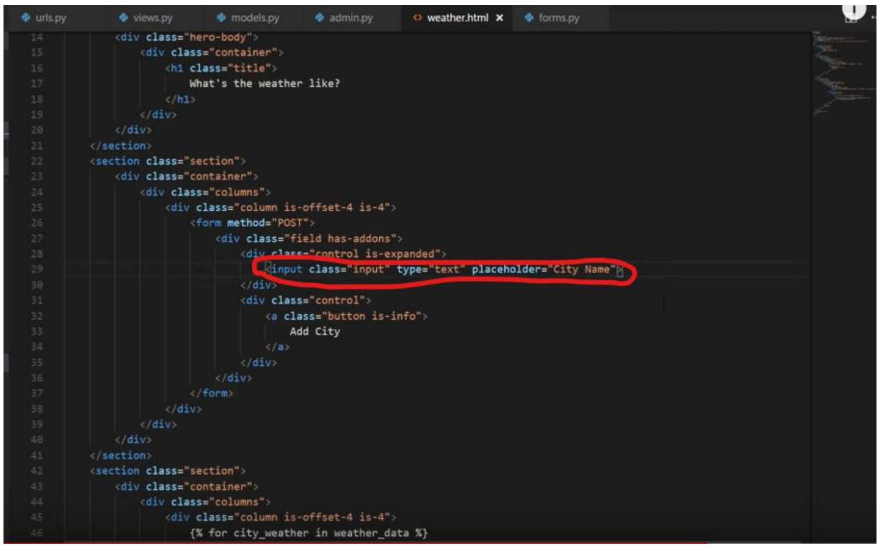
I create a new file named forms.py inside the weather app folder.



```python
from django.forms import ModelForm, TextInput#import modelform because i want exactly the same form as in the model
from .models import NameCity

class FormCity(ModelForm):
    class Meta:
        model = NameCity
        fields = ["name_of_city"]
        widgets = {"name_of_city" : TextInput(attrs = {"class" : "input" , "placeholder" : "City Name"})}
```

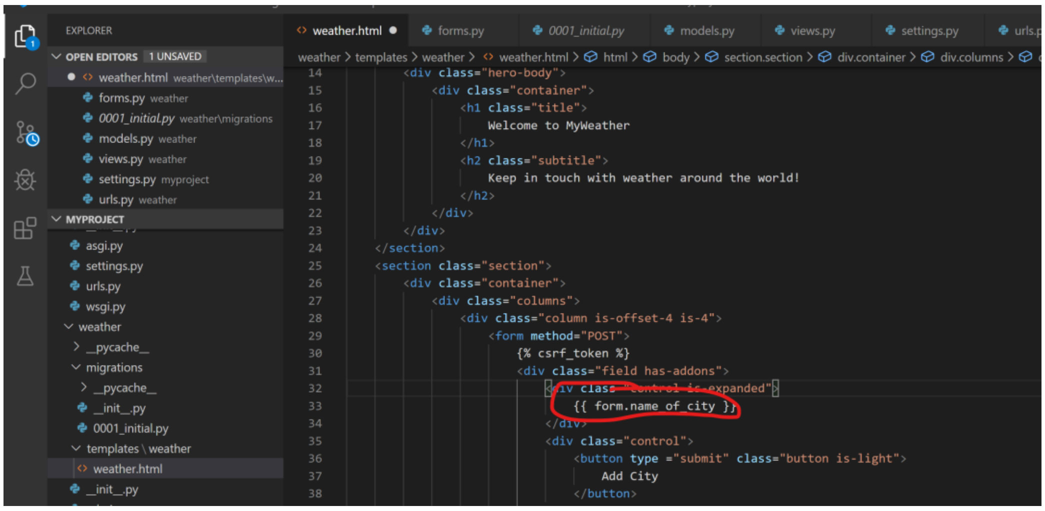
In this part, first I import the Modelform from django.forms library and also the model I'll work with, NameCity. After that I define a class named FormCity, which inherits from ModelForm. The simplest version of a ModelForm consists of a nested Meta class telling Django which model to base the form on and which fields to include in the form. Then, I build a form from the NameCity model and include only the name_of_city field. I also include widgets attribute. Widgets is an Html form element, such as a single-line text box, multi-line text area, or drop-down list. By including widgets attribute I can override Django's default widget choices. By using TextInput element, I customized the input widget for the field 'name_of_city'. TextInput will have the attributes of class as the key and input as the value also place holder as key and City Name as value. The purpose of making that kind of syntax is so that the user can add a city name based on the model I've created before. It will render from python type in widgets inside forms.py to html with the syntax `<input type="text" ...>`.

In the weather.html I erase this part because in that kind of input style I only input city name that has no relation with the model I created before. It means that if I keep on using the input style below it won't add the city to the database. Only add but will not display it later on



```
        <div class="hero-body">
            <div class="container">
                <h1 class="title">
                    What's the weather like?
                </h1>
            </div>
        </div>
    </section>
    <section class="section">
        <div class="container">
            <div class="columns">
                <div class="column is-offset-4 is-4">
                    <form method="POST">
                        <div class="field has-addons">
                            <div class="control is-expanded">
                                <input class="input" type="text" placeholder="City Name">
                            </div>
                            <div class="control">
                                <a class="button is-info">
                                    Add City
                                </a>
                            </div>
                        </div>
                    </form>
                </div>
            </div>
        </div>
    </section>
    <section class="section">
        <div class="container">
            <div class="columns">
                <div class="column is-offset-4 is-4">
                    {% for city_weather in weather_data %}
```

The weather.html will look this way after we erased it before:



```
        <div class="hero-body">
            <div class="container">
                <h1 class="title">
                    Welcome to MyWeather
                </h1>
                <h2 class="subtitle">
                    Keep in touch with weather around the world!
                </h2>
            </div>
        </div>
    </section>
    <section class="section">
        <div class="container">
            <div class="columns">
                <div class="column is-offset-4 is-4">
                    <form method="POST">
                        {% csrf_token %}
                        <div class="field has-addons">
                            <div class="control is-expanded">
                                {{ form.name_of_city }}
                            </div>
                            <div class="control">
                                <button type ="submit" class="button is-light">
                                    Add City
                                </button>
```

- Views.py

```
import requests #import requests that has been installed, it is build-
in library
from django.shortcuts import render, redirect
from .models import NameCity
from .forms import FormCity

# Create your views here.
def index(request):
    url = 'http://api.openweathermap.org/data/2.5/weather?q={}&units=metric
&appid=ec6bd904126723e79d1dd3261ba31e20'

    error_message = ""
    msg = ""
    class_msg = ""
    if request.method == "POST":
        form = FormCity(request.POST)
        if form.is_valid():
            new_city = form.cleaned_data["name_of_city"]
            existed_city = NameCity.objects.filter(name_of_city__iexact = n
ew_city).count()
            if existed_city == 0:
                r = requests.get(url.format(new_city)).json()
                if r["cod"] == 200:
                    form.save()
                else:
                    error_message = "The city " + new_city + " does not exi
st! Try another name!"
            else:
                error_message = "The city " + new_city  + " already exist!
Try another one!"
        if error_message:
            msg = error_message
            class_msg = "is-danger"
        else:
            msg = "City does exist and has been succesfully added!"
            class_msg = "is-success"

    form = FormCity()

    cities_name = NameCity.objects.all()

    data_of_weather = []

    for city in cities_name:

        r = requests.get(url.format(city)).json()
        city_weather = {
            "name_of_city" : city.name_of_city,
            "temperature" : r["main"]["temp"],
            "feels_like" : r["main"]["feels_like"],
            "description" : r["weather"][0]["description"],
            "icon" : r["weather"][0]["icon"] ,
        }
        data_of_weather.append(city_weather)

    context = {
        "data_of_weather" : data_of_weather ,
        "form" : form ,
        "message" : msg,
        "message_class": class_msg,
        }

    return render(request , 'weather/weather.html', context)

def delete_city_option(request , city_name):
    NameCity.objects.get(name_of_city = city_name).delete()

    return redirect("original")
```

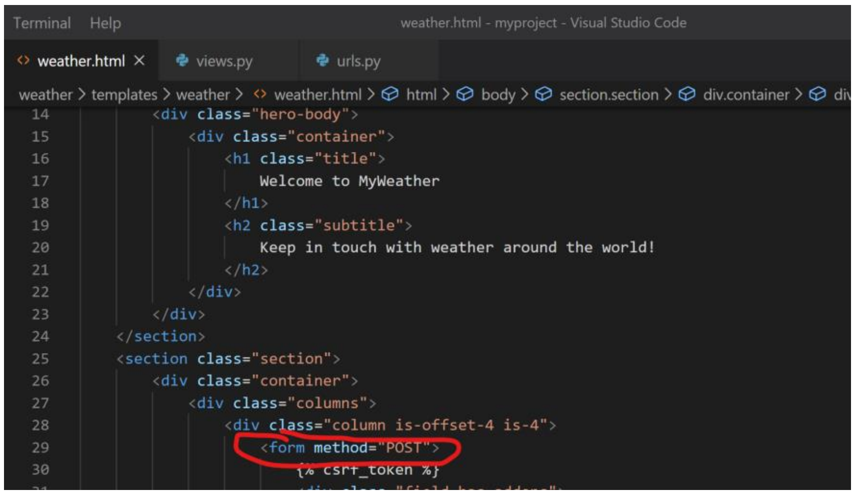
This is the most important part of the website, in this file we take the data from the models.py and the forms.py to work with. The views.py file purpose is to display all the data into the website.

```
   import requests #import requests that has been installed, it is build-
in library
from django.shortcuts import render, redirect
from .models import NameCity
from .forms import FormCity

# Create your views here.
def index(request):
    url = 'http://api.openweathermap.org/data/2.5/weather?q={}&units=metric
&appid=ec6bd904126723e79d1dd3261ba31e20'
```

So, we start with install requests in in the command prompt. After that, we import requests, render and redirect from django.shortcuts library, NameCity class from models.py file and FormCity class from forms.py file. Then define a function called index with the parameter request object. Then I create a variable url that contains the API from the openweather.org. The …q = {}… stands for query means that by inputting the name of the city we will able to get the weather data from the website openweather.org the symbol “{}” means it is a format type so it will later on be filled by a city name. The … units = metric… means that I wanted the degree in Celcius, if we want the degree in Fahrenheit we must change the units become imperial or by default (if you empty the units) it will be in Kelvin.

```
    error_message = ""
    msg = ""
    class_msg = ""
    if request.method == "POST":
        form = FormCity(request.POST)
        if form.is_valid():
            new_city = form.cleaned_data["name_of_city"]
            existed_city = NameCity.objects.filter(name_of_city__iexact = n
ew_city).count()
            if existed_city == 0:
                r = requests.get(url.format(new_city)).json()
                if r["cod"] == 200:
                    form.save()
                else:
                    error_message = "The city " + new_city + " does not exi
st! Try another name!"
            else:
                error_message = "The city " + new_city  + " already exist!
Try another one!"
        if error_message:
            msg = error_message
            class_msg = "is-danger"
        else:
            msg = "City does exist and has been succesfully added!"
            class_msg = "is-success"
```

In this part I create a variable error_message, msg for the message and class_msg for the CSS notification background color. All of the variable I fill with an empty string because I want to put a data inside of it later on . Then, on the next line I have if request.method == “POST” this means that request has a method option which the method has to action POST or GET. In this case it’s a POST, we create an instance of FormCity named form and pass the data entered by the user, stored in request.POST.


Terminal Help
weather.html - myproject - Visual Studio Code
weather.html
views.py
urls.py
weather > templates > weather > weather.html > html > body > section.section > div.container > di

```
        <div class="hero-body">
            <div class="container">
                <h1 class="title">
                    Welcome to MyWeather
                </h1>
                <h2 class="subtitle">
                    Keep in touch with weather around the world!
                </h2>
            </div>
        </div>
    </section>
    <section class="section">
        <div class="container">
            <div class="columns">
                <div class="column is-offset-4 is-4">
                    <form method="POST">
                        {% csrf_token %}
```

The weather.html screenshot shows us the method accepted ("POST"). POST means that we are going to post and display data or information through form to the database.

If form.is_valid() conditional statement means it checks whether the form we input in this case a city name is shorter than 25 length of characters. This is related to the models.py file where we use CharField with the maxlength of 25. The form.is_valid() returns a Boolean value whether it is true or false. If its true then we are going to the next line of the code. Inside the if statement I create a new variable new_city which takes every city name that the user input. With form.cleaned_data ["name_of_city"] syntax in the new_city variable, it means that the new_city will always have every data that has been validated from the form.is_valid(). I takes the field of name_of_city so if the city name the user input is smaller than 25 characters we can add it to the database. Then I create existed_city variable, the purpose of creating this variable is to prevent duplicate city to be added to the database. Existed_city variable will take all attributes from the NameCity class that we have imported from models.py before. After taking all objects from NameCity which only consist of name of the city I use the command filter inside of it I have name_of_city__iexact = new_city. So, how the code work is that it will filter all the data in name_of_city, if it has the same value with the new_city that the user going to input, it will be filtered. I add .count() at the end because I want to count how many city that the user input in new_city variable in the database. __iexact is a function called to prevent case sensitive, for example I have the city named Tokyo and tokyo by using __iexact it will value the same between Tokyo with capital t and tokyo with a small t.

Then in the next line, I make a conditional statement if existed_city == 0: the code means that If the counter is zero then it will run the code inside the if statement, if it is greater than zero then it will return a message said the city you input already exist in the database. So, we go to the code inside the if existed_city == 0: statement.

```
  if existed_city == 0:
          r = requests.get(url.format(new_city)).json()
          if r["cod"] == 200:
             form.save()
           else:
            error_message = "The city " + new_city + " does not exist! Try
another name!"
```

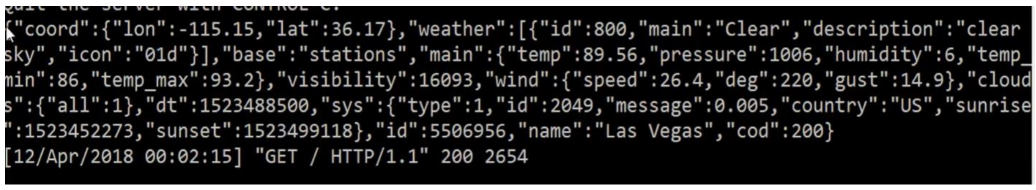
Inside the if statement, I create a r variable. The r variable uses the requests method that I have installed and imported before, and use the get to take the data from the API inside the url variable. Then, inside the get I have url.format(new_city)).json(). I use the Url.format to fill in the query that I left empty before (q={}). Then, fill it with the name of a city the user input in new_city variable. I use the json form because it converts the result of requests into json object. So, it means that the data we take from the API will convert to the combination of python list and dictionary. After that, I make an if r[“cod”] == 200: statement. The r[“cod”] == 200 means that inside the r I will have the cod key and value 200. In the API, if the cod key has a value 200 it means that the city user input does exist in the API database. Then, if it exists I will use form.save() to save the data to my database. If the cod key has value other than 200, it will return an error message to tell the user that the city that has been inputted before does not exist.

```
if error_message:
            msg = error_message
            class_msg = "is-danger"
        else:
            msg = "City does exist and has been succesfully added!"
            class_msg = "is-success"
```

In this part I make an if statement again to check if the error_message has a value in it then I will put the error_message to the msg variable and I assign the is-danger string to the class_msg so it will display a red background color in the notification bar later on in the website. And if the error_message doesn’t have any value, it will return the message “City does exist and has been successfully added!” put it inside the msg variable and I also will add the is-success string to the class_msg so it will display a green background color in the notification bar.

```python
    form = FormCity()

    cities_name = NameCity.objects.all()

    data_of_weather = []

    for city in cities_name:

        r = requests.get(url.format(city)).json()

        city_weather = {
            "name_of_city" : city.name_of_city,
            "temperature" : r["main"]["temp"],
            "feels_like" : r["main"]["feels_like"],
            "description" : r["weather"][0]["description"],
            "icon" : r["weather"][0]["icon"] ,
        }
        data_of_weather.append(city_weather)

    context = {
        "data_of_weather" : data_of_weather ,
        "form" : form ,
        "message" : msg,
        "message_class": class_msg,
        }

    return render(request , 'weather/weather.html', context)
```

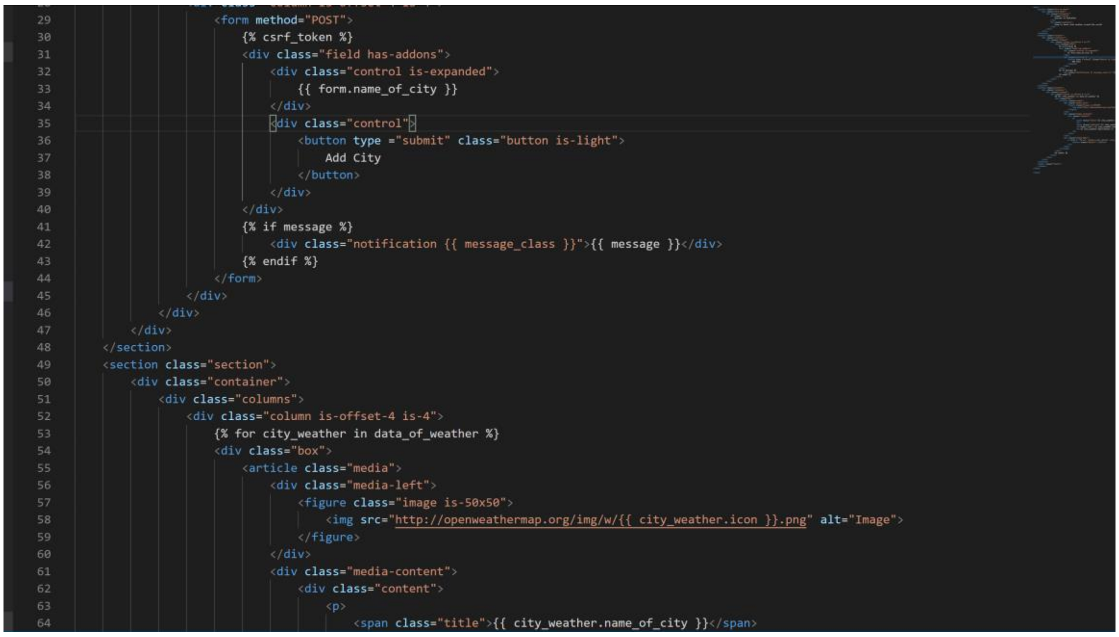
In this part, I then change the value inside the form. By inputting form = FormCity() means that every time a city has been added it will give the form in an empty value. I then declare a variable named cities_name = NameCity.objects.all(). So now, the cities_name variable has the value of all city inside the NameCity which consist of city name. Then I create a list called data_of_weather the purpose of this list is to collect all the data of a city. After that, I loop through all the city inside the cities_name variable and the same as I did before, I use the requests.get but this time the format I want to input inside the query (q = {}) is all the city inside the cities_name variable. Again with the json object, I wanted to take the data from the API in combination of python list and dictionary like the example below:



```
Quit the server with CONTROL-C.
{"coord":{"lon":-115.15,"lat":36.17},"weather":[{"id":800,"main":"Clear","description":"clear
sky","icon":"01d"}],"base":"stations","main":{"temp":89.56,"pressure":1006,"humidity":6,"temp_
min":86,"temp_max":93.2},"visibility":16093,"wind":{"speed":26.4,"deg":220,"gust":14.9},"cloud
s":{"all":1},"dt":1523488500,"sys":{"type":1,"id":2049,"message":0.005,"country":"US","sunrise
":1523452273,"sunset":1523499118},"id":5506956,"name":"Las Vegas","cod":200}
[12/Apr/2018 00:02:15] "GET / HTTP/1.1" 200 2654
```

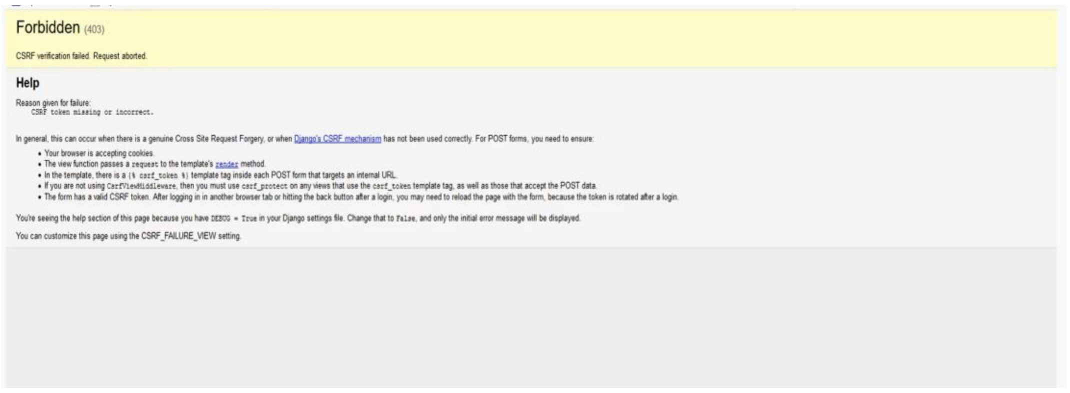
In the next part, I create a dictionary called city_weather and this dictionary will take the name_of_city, temperature, feels_like, description, and icon as the key(this key will be applied later on in the html file). And the value you can look above how to access it. Then, I append the city_weather dictionary to the data_of_weather. So now, inside the data_of_weather list will consist of dictionaries which contain a city weather information. Then, I create a dictionary again named context outside of the loop. Inside the context dictionary I make data_of_weather, form, message and message_class keys. And the value is the data taken from this file. For data_of_weather key it contains the value from the data_of_weather list, form contains the data from the form object, message contains the value from the msg variable, and message_class contains the value from class_msg variable. The purpose of making this dictionary is to pass it later on in the `return render(request , 'weather/weather.html', context)` to the weather.html file. So, the render function is to pass all the data from context variable inside this python file to the html file.

- Weather.html



```
                <form method="POST">
                    {% csrf_token %}
                    <div class="field has-addons">
                        <div class="control is-expanded">
                            {{ form.name_of_city }}
                        </div>
                        <div class="control">
                            <button type ="submit" class="button is-light">
                                Add City
                            </button>
                        </div>
                    </div>
                    {% if message %}
                        <div class="notification {{ message_class }}">{{ message }}</div>
                    {% endif %}
                </form>
            </div>
        </div>
    </div>
</section>
<section class="section">
    <div class="container">
        <div class="columns">
            <div class="column is-offset-4 is-4">
                {% for city_weather in data_of_weather %}
                <div class="box">
                    <article class="media">
                        <div class="media-left">
                            <figure class="image is-50x50">
                                <img src="http://openweathermap.org/img/w/{{ city_weather.icon }}.png" alt="Image">
                            </figure>
                        </div>
                        <div class="media-content">
                            <div class="content">
                                <p>
                                    <span class="title">{{ city_weather.name_of_city }}</span>
```

First I add the {% csrf_token %} it is a template tag used by Django to prevent attackers from using the form to gain unauthorized access to the server(this kind of attack is called cross-site request forgery). If I didn't add the csrf token then when I try to submit a city name the website will redirect the error page below.



Forbidden (403)

CSRF verification failed. Request aborted.

Help

Reason given for failure:
    CSRF token missing or incorrect.

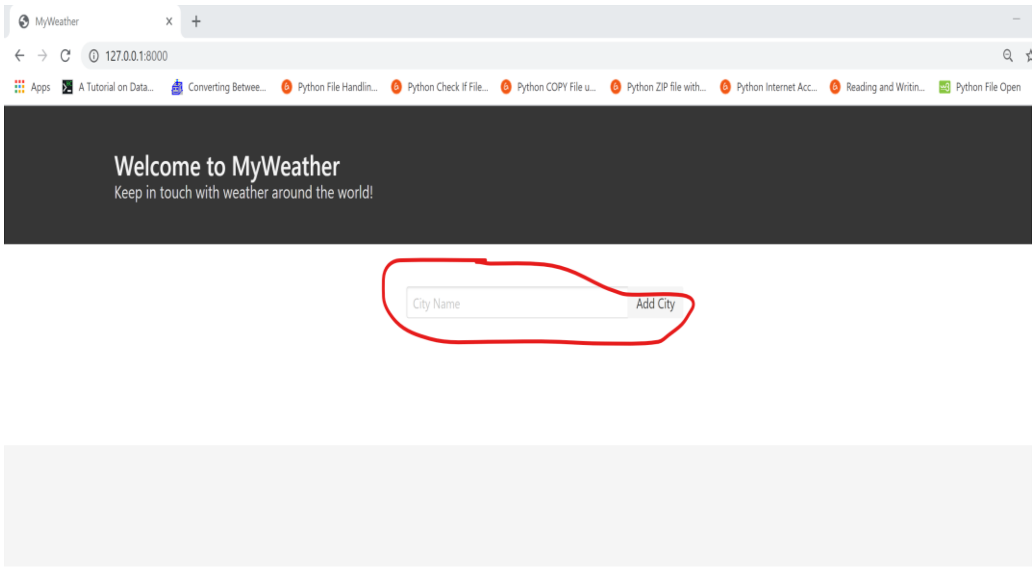
In general, this can occur when there is a genuine Cross Site Request Forgery, or when Django's CSRF mechanism has not been used correctly. For POST forms, you need to ensure:

- Your browser is accepting cookies.
- The view function passes a `request` to the template's `render` method.
- In the template, there is a `{% csrf_token %}` template tag inside each POST form that targets an internal URL.
- If you are not using `CsrfViewMiddleware`, then you must use `csrf_protect` on any views that use the `csrf_token` template tag, as well as those that accept the POST data.
- The form has a valid CSRF token. After logging in in another browser tab or hitting the back button after a login, you may need to reload the page with the form, because the token is rotated after a login.

You're seeing the help section of this page because you have `DEBUG = True` in your Django settings file. Change that to `False`, and only the initial error message will be displayed.

You can customize this page using the CSRF_FAILURE_VIEW setting.

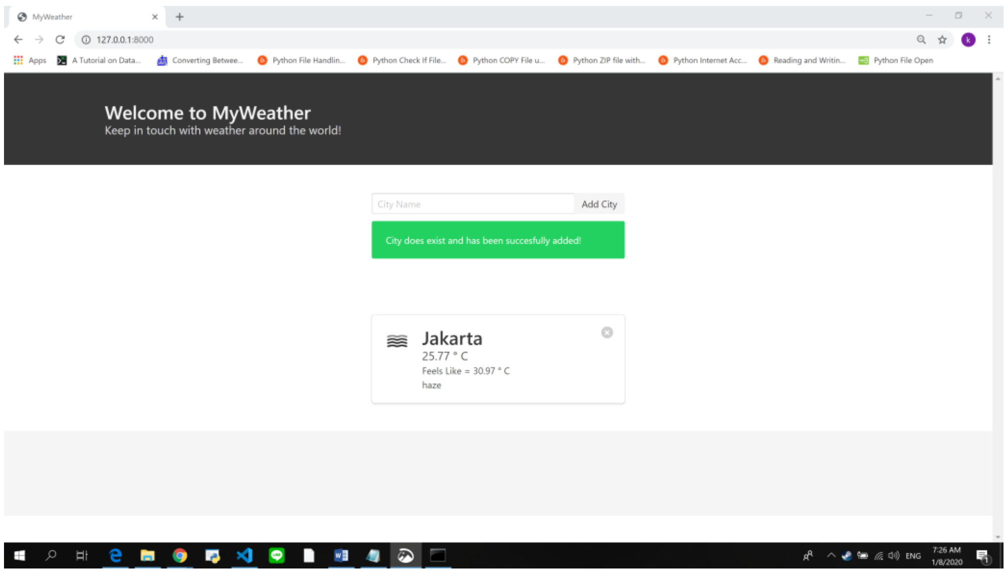
Then, I display the form with the name_of_city field. So, by doing that I will be able to input name of the city in here:



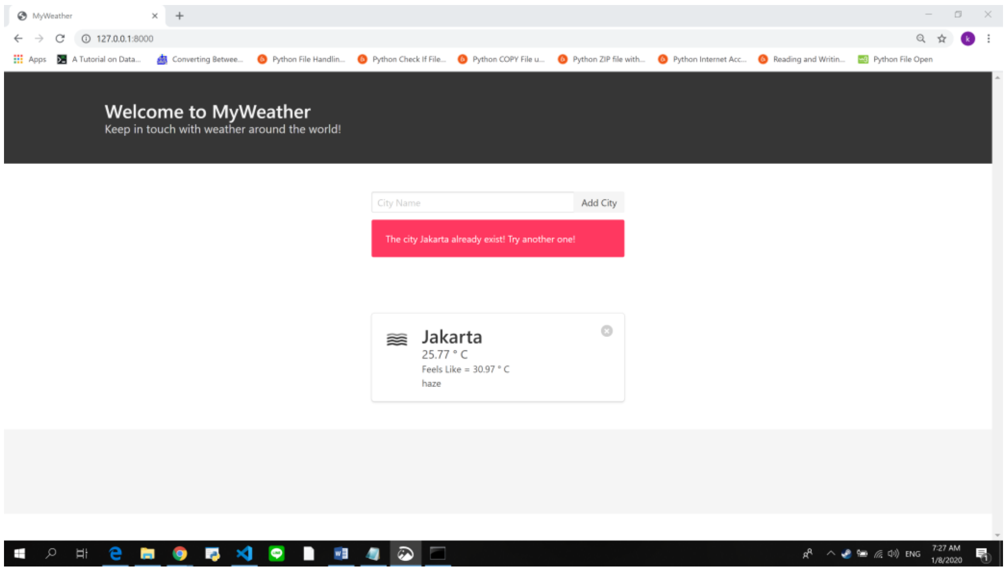
The code takes the form from the views.py. In the views.py the form object I've created takes the data from FormCity class in forms.py file, it takes the field which contains name_of_city taken from the NameCity class form models.py. And to display the input bar I use the attribute widget from forms.py file.

The next step, I add a button type to submit the city and the button has the label Add City. Then, I add an if message in line 41 the if message takes the data that was rendered from views.py to weather.html. Inside the if message I define a class notification, the purpose of that is to make a notification bar under the input box.

So, if the city does exist in the world it will return something like this:



Or if the city already exist in the database:

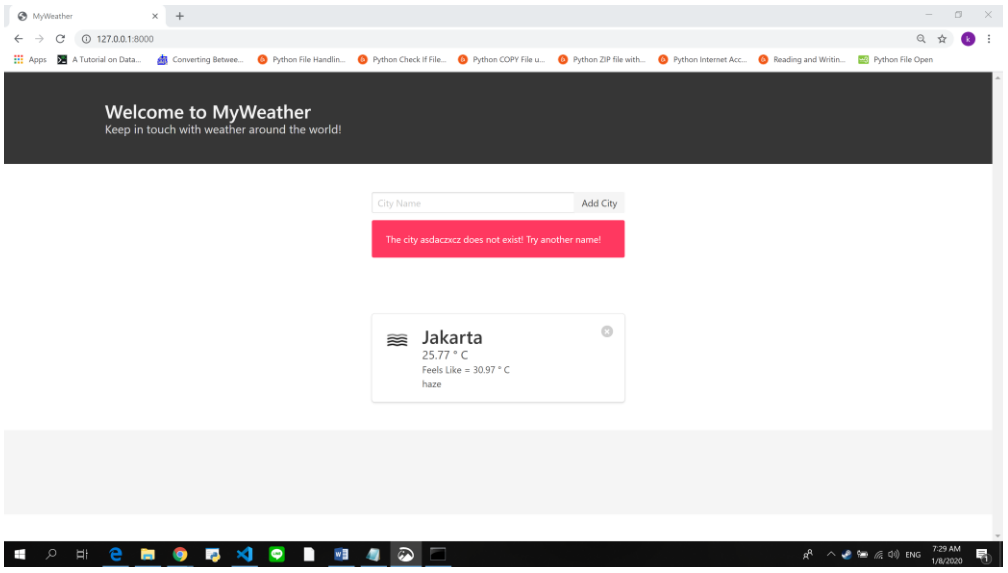


Welcome to MyWeather
Keep in touch with weather around the world!

The city Jakarta already exist! Try another one!

Jakarta
25.77 ° C
Feels Like = 30.97 ° C
haze

Or if the city does not exist in the world:

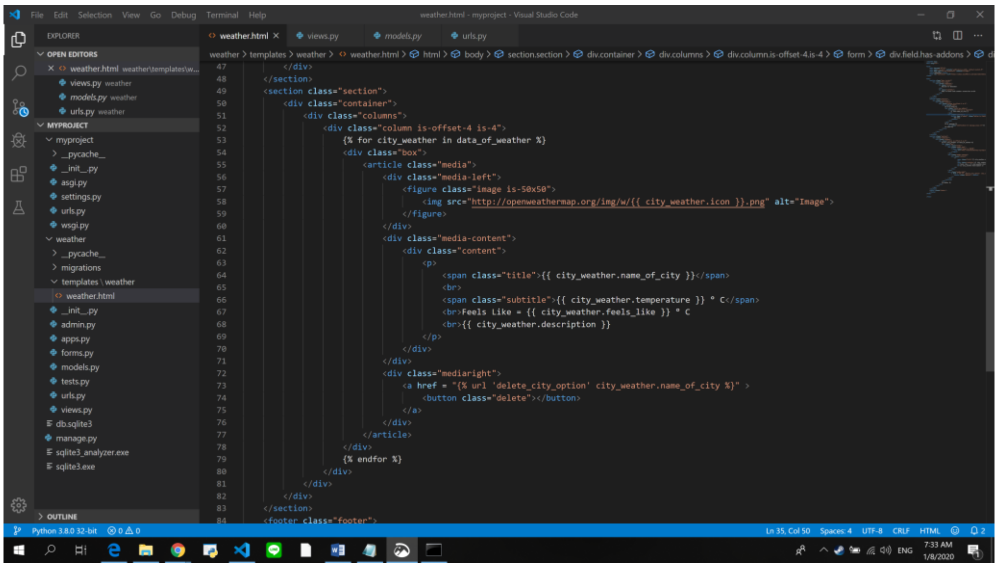


Welcome to MyWeather
Keep in touch with weather around the world!

The city asdaczxcz does not exist! Try another name!

Jakarta
25.77 ° C
Feels Like = 30.97 ° C
haze

Then, I create a for loop in the html file. It will look like this:



```
        </div>
    </section>
    <section class="section">
        <div class="container">
            <div class="columns">
                <div class="column is-offset-4 is-4">
                    {% for city_weather in data_of_weather %}
                    <div class="box">
                        <article class="media">
                            <div class="media-left">
                                <figure class="image is-50x50">
                                    <img src="http://openweathermap.org/img/w/{{ city_weather.icon }}.png" alt="Image">
                                </figure>
                            </div>
                            <div class="media-content">
                                <div class="content">
                                    <p>
                                        <span class="title">{{ city_weather.name_of_city }}</span>
                                        <br>
                                        <span class="subtitle">{{ city_weather.temperature }} ° C</span>
                                        <br>Feels Like = {{ city_weather.feels_like }} ° C
                                        <br>{{ city_weather.description }}
                                    </p>
                                </div>
                            </div>
                            <div class="mediaright">
                                <a href = "{% url 'delete_city_option' city_weather.name_of_city %}" >
                                    <button class="delete"></button>
                                </a>
                            </div>
                        </article>
                    </div>
                    {% endfor %}
                </div>
            </div>
        </div>
    </section>
    <footer class="footer">
```

I create the for loop inside the box so that every city will have its own box containing the weather information. With the format {% for city_weather in data_of_weather %} means that I will loop everything inside the data_of_weather list which contains dictionary of city_weather that is passed/rendered from the views.py so every city the user input will create a new box contains name of the city, temperature, feels like, description and also icon. To display the name, description, temperature, feels like and icon to the website, we just need to call the city_weather variable from the html loop followed by key of the city_weather dictionary inside data_of_weather list in views.py file and it will automatically display all the things we wanted to. But in the icon part is a special one. It takes a new class image of html and we take the icon from the openweathermap.org make in a .pmg file and call it the same way as the other to display it in the website.

Finally, I create a delete button to delete a city that the user wanted to. I define a delete function named delete_city_option, looks like this:

```
def delete_city_option(request , city_name):
    NameCity.objects.get(name_of_city = city_name).delete()

    return redirect("original")
```

So, first I define delete_city_option function it takes the parameter city_name and request inside the function it takes the object from NameCity class from models.py and takes all the object that has the same value as the city_name and delete it from the database. This function returns redirect means that every time I delete a city, it will automatically refresh to the new page where the city we delete disappears.

After that, I create a new file called urls.py. The purpose of creating this file is to add the url path for the views.py.

```
from django.urls import path
from . import views
urlpatterns = [
    path('', views.index, name = "original" ),#refers to index function in vie
ws.py file
    path("delete/<city_name>/", views.delete_city_option , name = "delete_city
_option"), #"delete/<city_name>/" means in the html we need that form of delet
e
]
```

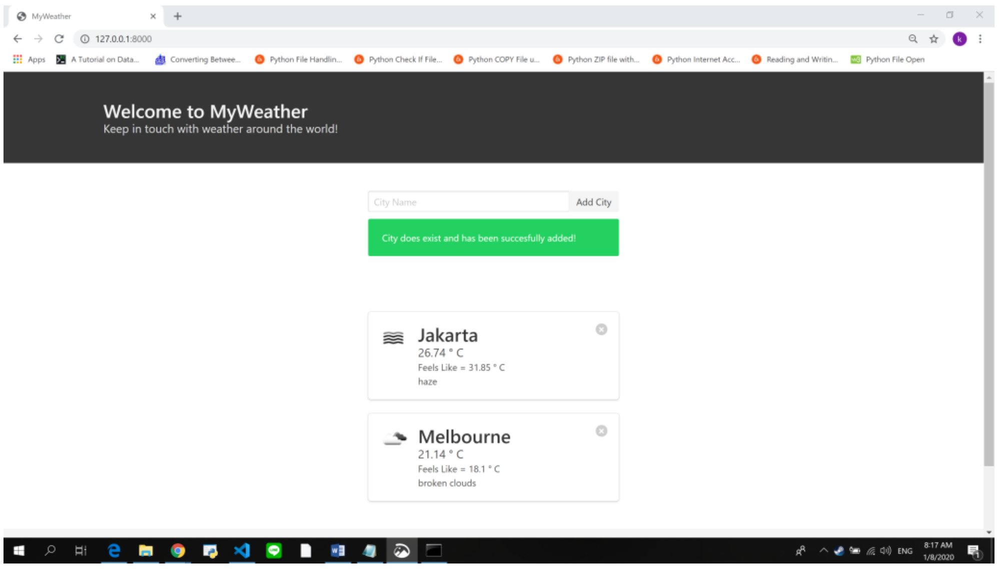
This a copy of the urls,py, we create this inside the weather app folder. We start by importing path and also views.py file. Then, we add 2 url patterns. One is path for the index function and label it as original (this gives the website an address to display the view of the website (default view) using the function index in the views.py file). The other one, is path for the delete_city_option function that has an end point with the format delete/<city_name>/ that we will use later on in the weather.html file and label it as delete_city_option (this gives the website an address to delete a city using the function I created in the in the views.py file). So now we know that every time we erase a city name it will refresh the page back to the index function in views.py.

In the weather.html I create a button to delete like the example below:

```
<div class="mediaright">
    <a href = "{% url 'delete_city_option' city_weather.name_of_city %}" >
        <button class="delete"></button>
    </a>
```

In the href line we can see it contains the end point of the path for delete_city_option that we have created in urls.py of the weather app. It starts with the delete option then followed by the name of the city.

Example:



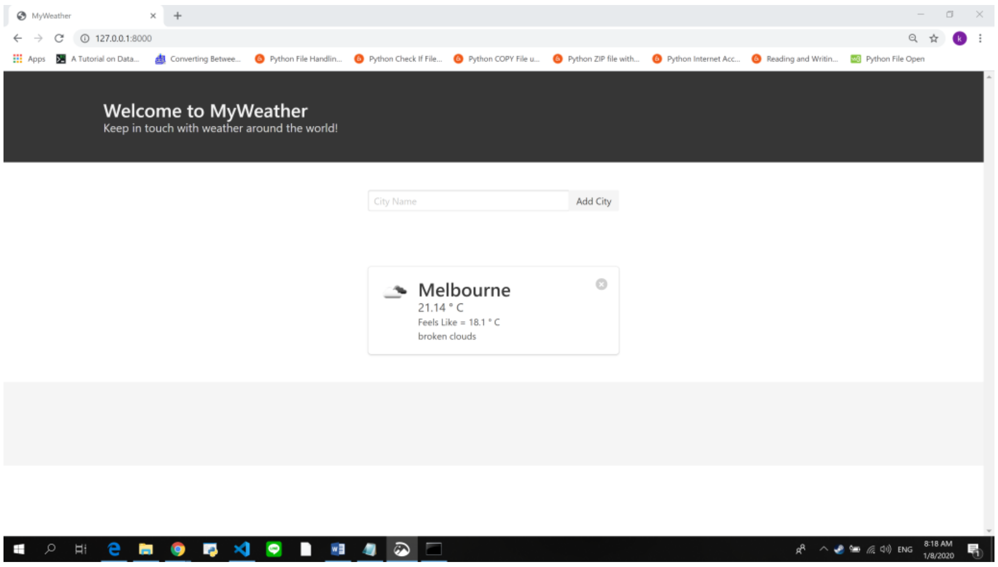
Welcome to MyWeather
Keep in touch with weather around the world!

City does exist and has been succesfully added!

Jakarta
26.74 ° C
Feels Like = 31.85 ° C
haze

Melbourne
21.14 ° C
Feels Like = 18.1 ° C
broken clouds

Then I try press the delete button in Jakarta's box.



Welcome to MyWeather
Keep in touch with weather around the world!

Melbourne
21.14 ° C
Feels Like = 18.1 ° C
broken clouds

It will redirect us to the page where Jakarta box has disappeared. This shows us that if we press the button delete in MyWeather website it will automatically erase the name of the city and display a new page where the city we deleted has gone.

IV. Problems While Making The Website

Creating this Django website is not an easy task. It takes a lot of libraries, working with a lot of files, I also need to learn how to work with urls and API. The problem comes when I wanted to display the data to the website, I often got an error but by searching through the internet I finally got my mistake. I also had a problem with the case sensitive, so the problem is if I input Tokyo with capital t and tokyo with small t it will counted as different city.

V. Problems with The Website

This website has one problem that I can't figure out to be honest. The problem is when the user input everything in the lowercase, uppercase or even mixed forms. The website will display it so until this point I can't figure out how to auto capitalize the users input.

VI. References

- https://github.com/PrettyPrinted/weather_app_django/blob/master/weather.html
- https://stackoverflow.com (website I used when I was trying to fix the errors)
- https://www.youtube.com/watch?v=v7xjdXWZafY&t=1353s