УДК 004

Метеорологическое web-приложение на Django

Романов Даниил Алексеевич
Приамурский государственный университет имени Шолом-Алейхема
студент

Аннотация
Данная статья написана с целью создания web-приложения, показывающего погоду в разных городах мира. Приложение будет создано на основе готового сервера Django с использованием языка программирования Python. Будут использованы гео-данные с сайта OpenWeatherMap.
Ключевые слова: web-приложение, Python, Django сервер, гео-данные, OpenWeatherMap

Meteorological web application on Django

Romanov Daniil Alekseevich
Sholom-Aleichem Priamursky State University
student

Abstract
This article is written with the aim of creating a web application showing the weather in different cities of the world. The application will be created on the basis of a ready-made Django server using the Python programming language. Geo-data from the OpenWeatherMap website will be used.
Keywords: web application, Python, Django server, geo-data, OpenWeatherMap

1 Введение

1.1 Актуальность

На сегодняшний день почти каждому человеку нужно знать погоду в определённом месте. Например, чтобы решить ехать ли ему в это место или что с собой взять. Гео-данные являются важной информаций для многих отраслей науки. Поэтому будет не лишним уметь создавать приложения с использованием этих данных.

1.2 Обзор исследований

В своей работе C. Dewi и R. C. Chen описывали возможности сайта OpenWeather и его применение в различных веб-проектах [1]. Р. Р. Крапивин, Г. А. Гареева исследовали способ получения доступа к данным путем авторизации аккаунта с помощью библиотеки Requests в языке Python [2]. В. П. Капитонов рассматривал реализацию архитектуры rest API с помощью Python для взаимодействия с сервисами приложения.

1.3 Цель исследования

Цель исследования - создать и понять принцип работы погодного приложения, написанного на языке программирования Python с использованием библиотеки Django, которое будет обрабатывать гео-данные и показывать погоду в определённых городах.

2 Материалы и методы

Для создания программы нам потребуется несколько вещей. Во-первых, это готовый web-сервер Jango в текстовом редакторе Atom с настроенным внешним видом, сделать который можно с помощью статей: “Создание сервера Jango в редакторе Atom” [4] и “Оформление внешнего вида сайта с помощью Bootstrap” [5]. В-третьих, сайт OpenWeatherMap [6]. Данный сайт предоставляет нам API при помощи которого можно получать информацию о погоде в разных городах мира.

3 Результаты и обсуждения

Потребуется сайт OpenWeatherMap. Для начала следует зарегистрироваться на сайте, для этого нажмите на Sign in (рис.1).

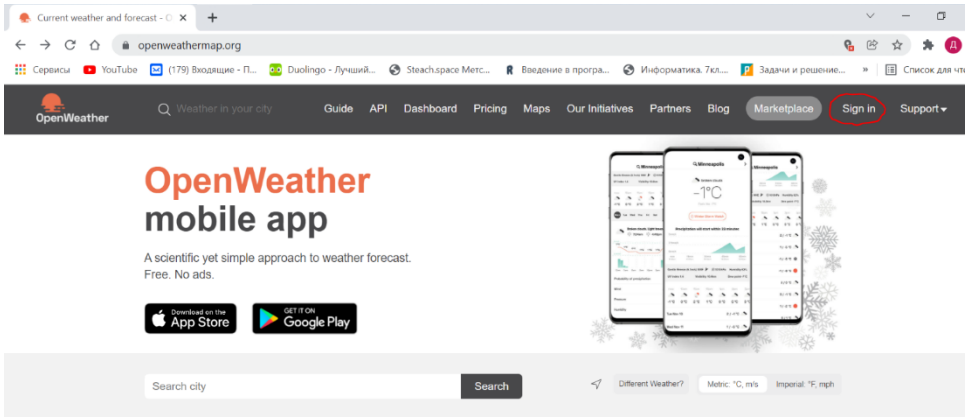

Рисунок 1 - Сайт OpenWeatherMap

После регистрации должно высветиться меню, в котором будет указан пункт API keys. В нём будет указан ключ, который позволит отсылать различные API-запросы к данному сайту. Желательно где-нибудь сохранить данный ключ. Далее в главном меню сайта нажимаем API и выбираем Current Weather Data, API doc. Чтобы отправить какой-либо API-запрос необходимо сформулировать URL-адрес, в котором мы указываем имя города. Отыскав пункт Examples of API calls копируем частично ссылку и вставляем в поисковик (рис.2).

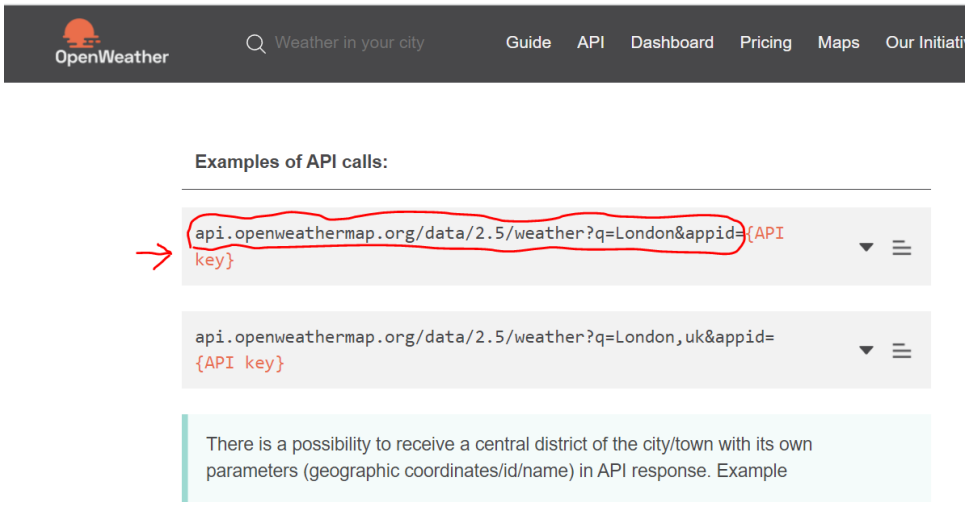

Рисунок 2 – URL-адрес

И сразу за ссылкой вставляем наш API-ключ. Загрузив полученную ссылку, должна вывестись информация о погоде города (рис.3).

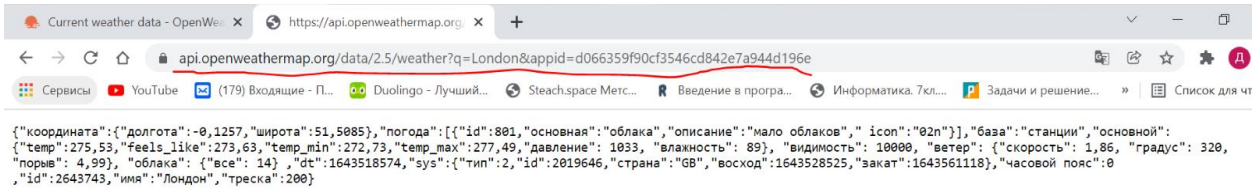

Рисунок 3 - Информация о погоде города

URL-адрес нужен для того, чтобы получать информацию о погоде в определённом городе. Чтобы это осуществить понадобится модуль requests. Для этого прописываем в терминале команду “pip install requests”. Затем в файле views.py необходимо вставить полный URL-адрес и подключить модуль requests с помощью которого мы формируем запрос. Чтобы проверить корректность получаемых данных с помощью функций print и text выведем в консоль полученный из запроса ответ (рис.4).

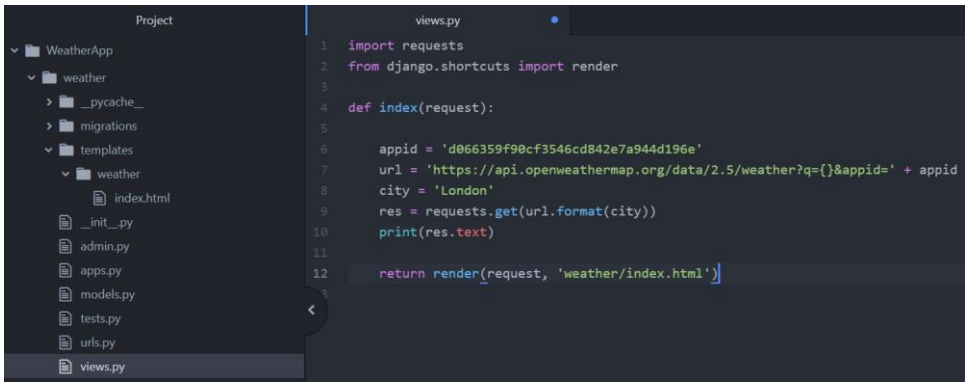

Рисунок 4 - Изменения в файле views.py

Далее запускаем локальный сервер и перезапускаем страницу сайта. В файле views.py следует добавить функцию, преобразующую полученные

данные в словарь и поменять url-запрос чтобы настроить единицы измерения. Используются не все полученные данные, а лишь название города, погода и иконка погоды. Для этого нужно создать новый словарь city_info. Переменная context передает данные ф html-шаблон (рис. 5).

views.py

```python
import requests
from django.shortcuts import render

def index(request):

    appid = 'd066359f90cf3546cd842e7a944d196e'
    url = 'https://api.openweathermap.org/data/2.5/weather?q={}&units=metric&appid=' + appid
    city = 'London'
    res = requests.get(url.format(city)).json()

    city_info = {
        'city': city,
        'temp': res['main']['temp'],
        'icon': res['weather'][0]['icon']
    }

    context = {'info': city_info}

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

Рисунок 5 - Обработка полученных данных в views.py

Теперь в шаблоне index.html можно отобразить все полученные данные (рис.6).

```
      </div>
      <div class='col-4 offset-1'>
        <h1>Информация</h1>
        <div class="alert alert-danger">
          <b>Город:</b> {{ info.city }}<br>
          <b>Температура:</b> {{ info.temp }}<sup>o</sup><br>
          <img src="http://openweathermap.org/img/w/{{ info.icon }}.png" alt="Фото погоды" class="img-thumnail">
        </div>
      </div>
    </div>
```

Рисунок 6 - Изменения в файле index.html

После чего на главной странице должны отображаться полученные данные (рис.7).

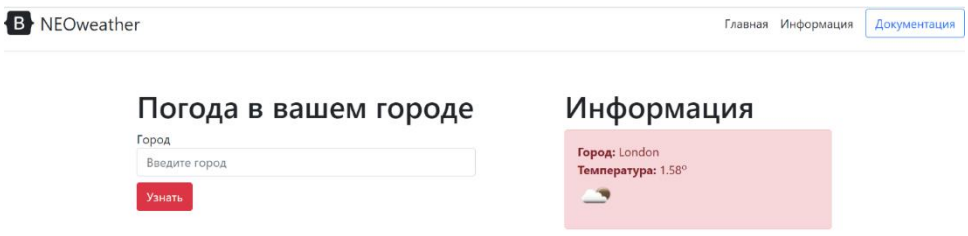

Рисунок 7 - Полученные данные на главной странице

Далее необходимо зайти в models.py (рис.8).

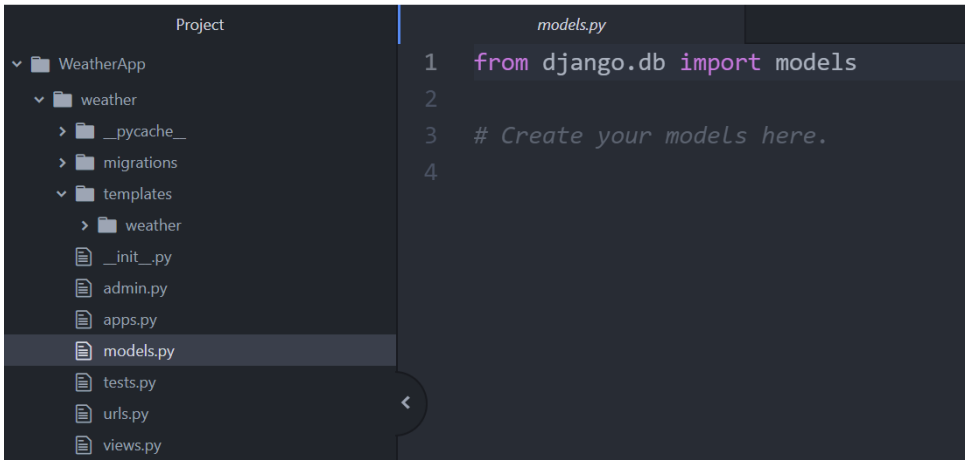

Рисунок 8 - Файл models.py

И создать таблицу чтобы взаимодействовать с данными из поисковой строки (рис.9).

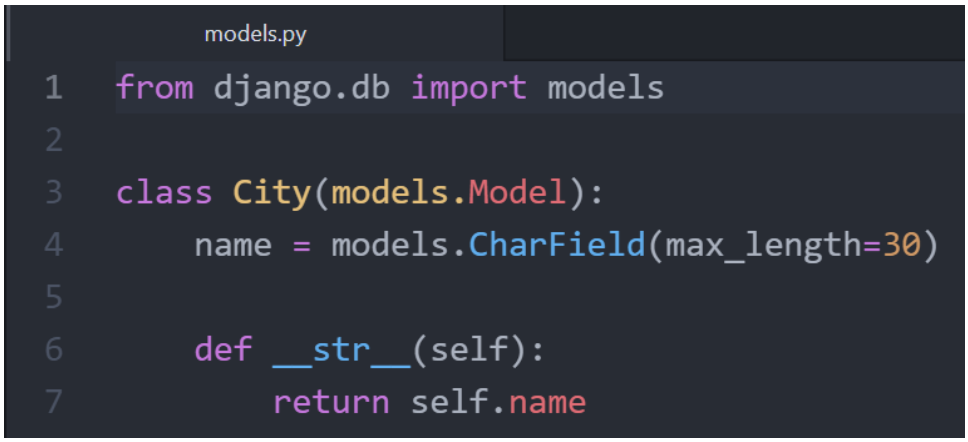

Рисунок 9 - Изменения в файле models.py

Осталось выполнить различные миграции. Для этого пишем в терминале команду “python manage.py make migrations”. Дальше используем команду “python manage.py migrate” и выполняем миграции. Затем переходим в файл admin.py (рис.10).

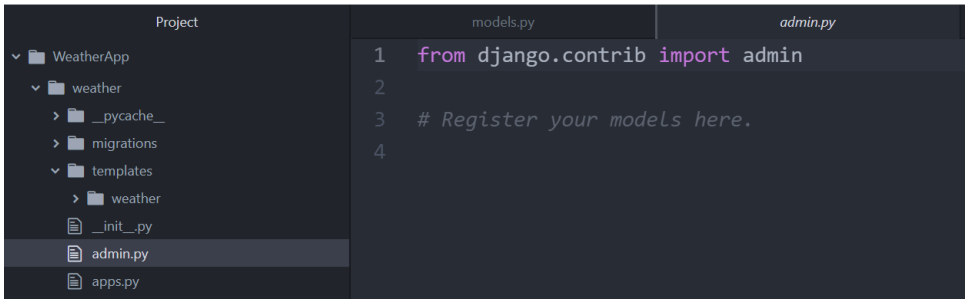

Рисунок 10 - Файл admin.py

В этом файле добавляем таблицу городов, которая будет отображаться в панели администратора (рис.11).

Рисунок 11 - Создание таблицы в файле admin.py

После этого переходим на страницу администратора и видим таблицу Cities (рис.12).

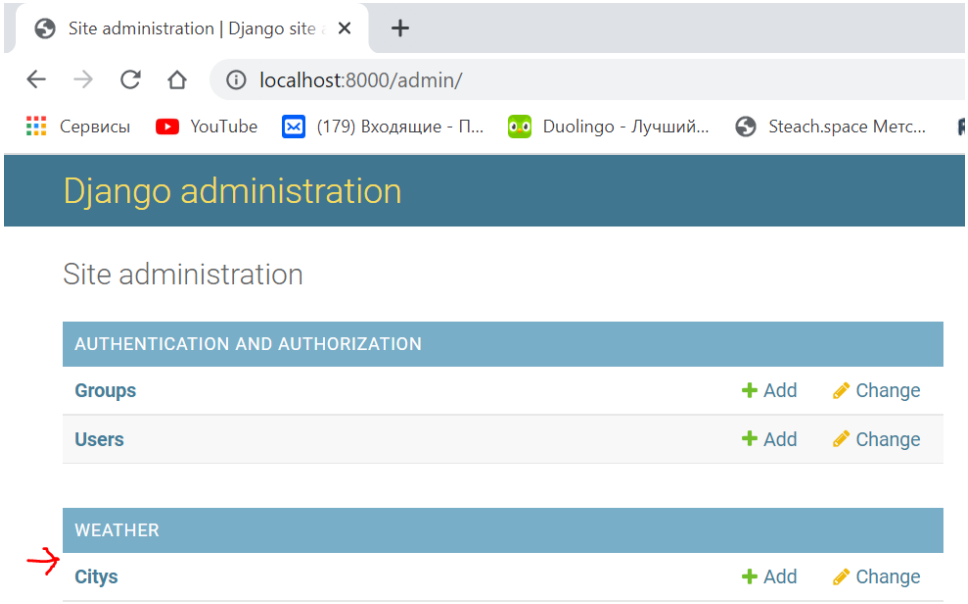

Рисунок 12 - Новая таблица Cities на странице администратора

Перейдя в неё, можно добавлять города, прописывая их названия на английском в строке Name и нажимая на Save and add another. Например, добавим London, Las Vegas и New York (рис. 13).

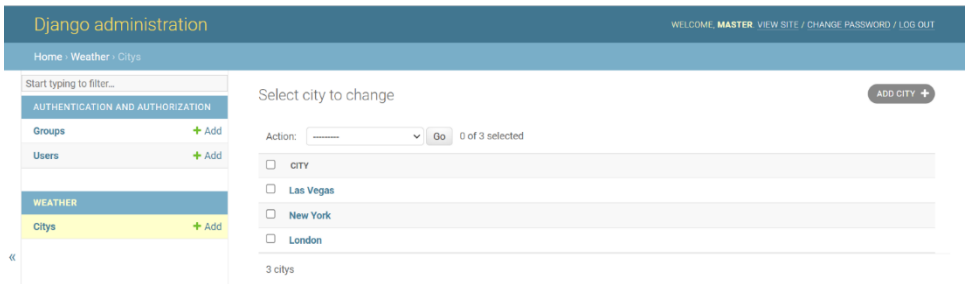

Рисунок 13 - Результат добавления городов в таблицу Cities

Всё что осталось сделать, это получить данные из таблицы и вывести их на главной странице в блоке информации. Для этого заходим в файл views.py и импортируем таблицу City (рис.14).

views.py

```
1  import requests
2  from django.shortcuts import render
3  from .models import City
```

Рисунок 14 - Импорт таблицы City

Дальше создаём переменную cities, которая будет принимать все объекты из таблицы City. Затем создаём цикл, который будет перебирать весь массив данных в переменной cities и добавлять весь массив внутрь объекта city_info (рис.15).

views.py

```
8   url = 'https://api.openweathermap.org/data/2.5/weather?q={}&units=m
9
10  cities = City.objects.all()
11
12  all_cities=[]
13
14  for city in cities:
15      res = requests.get(url.format(city.name)).json()
16      city_info = {
17          'city': city.name,
18          'temp': res['main']['temp'],
19          'icon': res['weather'][0]['icon']
20      }
21
22      all_cities.append(city_info)
23
24  context = {'all_info': all_cities}
25
26  return render(request, 'weather/index.html', context)
```

Рисунок 15 - Обработка данных в файле views.py

Теперь необходимо зайти в index.html и несколько раз в цикле вывести блоки с “alert” (рис.16).

```
{% for info in all_info %}
<div class="alert alert-info">
  <div class="row">
    <div class="col-9">
      <b>Город:</b> {{ info.city }}<br>
      <b>Температура:</b> {{ info.temp }}<sup>o</sup>
    </div>
    <div class="col-2 offset-1">
      <img src="http://openweathermap.org/img/w/{{ info.icon }}.png" alt="
    </div>
  </div>
</div>
{% endfor %}
```

Рисунок 16 - Изменения в файле index.html

Перезагружаем страницу и видим, что справа отображается информация о введённых ранее городах. Всё что осталось сделать, это добавить функционал поисковой строки, чтобы можно было ввести название города, нажать на кнопку “Узнать”, этот город добавлялся в таблицу администратора и отображался в блоке информации. Для этого создаём файл forms.py в weather (рис.17).

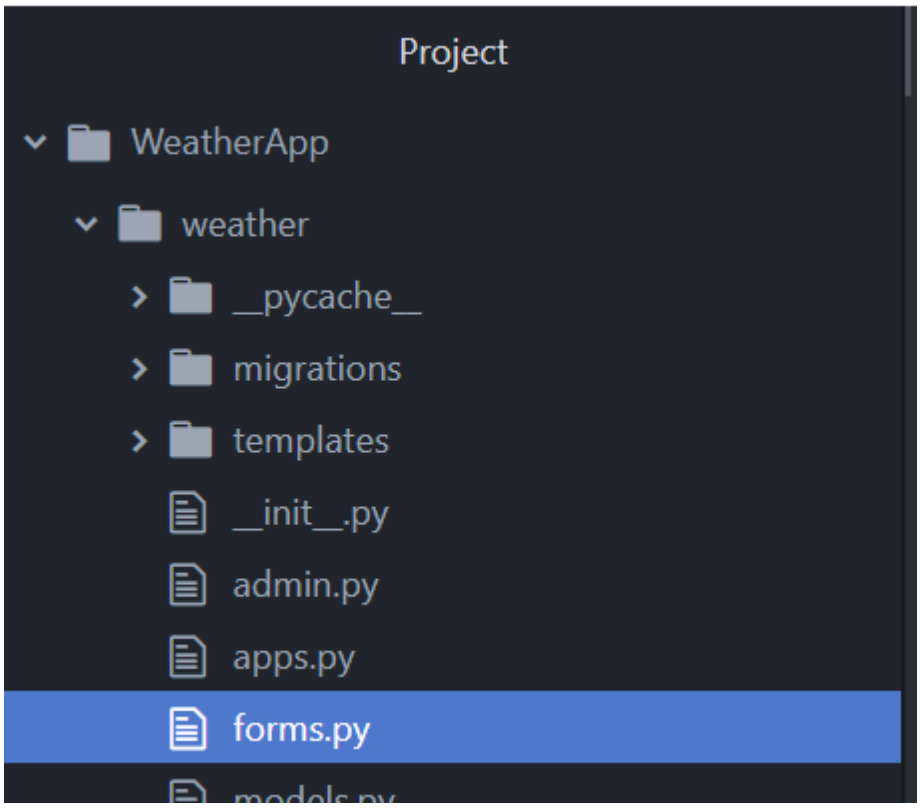

Рисунок 17 - Создание forms.py

Этот файл служит для работы с формой. В нём мы импортируем нужные модели и формы. Далее переходим в views.py. В нём необходимо подключить будущую формочку CitiForm для работы с запросами и создаём блок кода, который будет сохранять запросы в таблице и обновлять страницу (рис.18).

forms.py views.py

```python
import requests
from django.shortcuts import render
from .models import City
from .forms import CityForm

def index(request):

    appid = 'd066359f90cf3546cd842e7a944d196e'
    url = 'https://api.openweathermap.org/data/2.5/weather?

    if(request.method == 'POST'):
        form = CityForm(request.POST)
        form.save()

    form = CityForm()

    cities = City.objects.all()

    all_cities=[]
```

Рисунок 18 - Обработка и сохранение запросов

Также необходимо передать формочку в шаблон. Для этого в объект context добавляем новый параметр form со значением form (рис.19).

```
    context = {'all_info': all_cities, 'form': form}
```

Рисунок 19 - Передача формы в новый шаблон

В index.html нужно прописать вывод поля name и прописать обязательный параметр, который необходимо прописывать чтобы форма работала корректно (рис.20).

```
<form action="" method="post">
{% csrf_token %}
<label for="city">Город</label>
{{ form.name }}
<input type="text" id="city" class="form-control
```

Рисунок 20 - Настройка формы

Возвращаемся в файл forms.py где нужно импортировать классы City, ModelForm и TextInput, отвечающий за атрибуты к полям формы. Здесь создаём новый класс CityForm в котором мы прописываем и настраиваем атрибуты для внешнего вида формы (рис.21).

```python
from .models import City
from django.forms import ModelForm, TextInput

class CityForm(ModelForm):
    class Meta:
        model = City
        fields = ['name']
        widgets = {'name' : TextInput(attrs=
        {'class': 'form-control',
        'name': 'city',
        'id': 'city',
        'placeholder': 'Введите город'})}
```

Рисунок 21 - Настройка внешнего вида формы

Теперь поисковая строка стала полноценно функциональной. Если ввести в неё город и нажать “Узнать” этот город добавится в базу данных, страница перегрузится и город отобразится город в поле информации. Например, введём город Toronto (рис. 22).

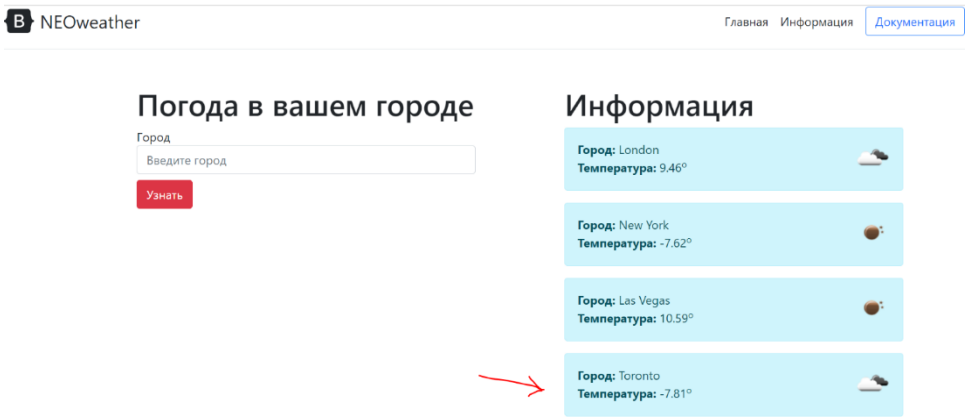

Рисунок 22 - Тест работоспособности сайта

Тоже самое можно проделать с другими городами.

Выводы

В данной работе было создано веб-приложение на основе сервера Jango, которое показывает погоду в настоящий момент времени в разных городах.

Библиографический список

1. Dewi C., Chen R. C. Integrating Real-Time Weather Forecasts Data Using OpenWeatherMap and Twitter //International Journal of Information Technology and Business. 2019. Т. 1. №. 2. С. 48-52.
2. Крапивин Р. Р., Гареева Г. А. Получение доступа к данным путем авторизации в аккаунт с помощью библиотеки Requests в языке Python //Инновационные технологии, экономика и менеджмент в промышленности. 2021. С. 206-208.
3. Капитонов В. П. Реализация архитектуры rest API с помощью Python для взаимодействия с сервисами приложения и изоляции базы данных CouchDB //Проблемы науки. 2018. №. 5 (29). С. 30-33.
4. Создание сервера Jango в редакторе Atom URL: https://1drv.ms/w/s!AtdxmxWoArjpgV1a3kcnAfRymFkq?e=CXbNYy
5. Оформление внешнего вида сайта с помощью Bootstrap URL: https://1drv.ms/w/s!AtdxmxWoArjpgWA7J263aVPecQK9?e=dksIka
6. OpenWeatherMap URL: https://openweathermap.org