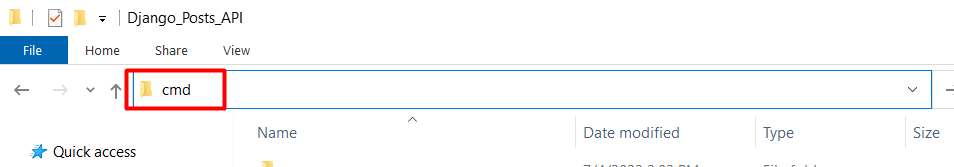
**Лаб. работа №2. Разработка приложения «Posts» на Django Framework**

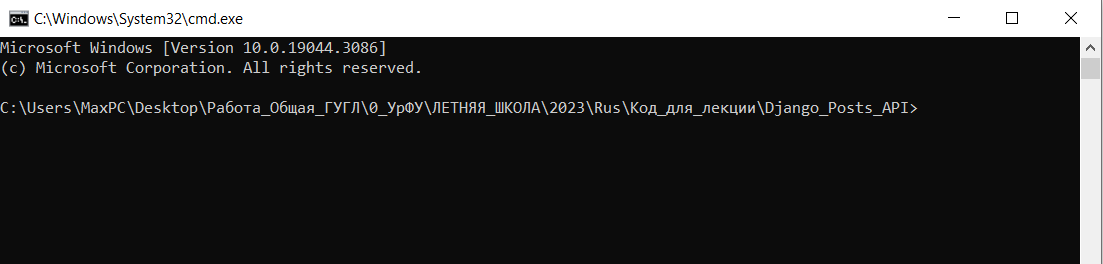
Создайте новую папку на рабочем столе с именем "Django\_Posts\_API".

Откройте командную строку из текущей папки.

Чтобы сделать это, вы можете просто ввести команду «**cmd**» в строке поиска текущей папки и нажать **Enter**.



После этого откроется командная строка:



**Шаг 1: Настройка проекта Django**

**Создадим** виртуальную среду для нашего проекта. Создание виртуальной среды необходимо для того, чтобы при наличии множества проектов на одном локальном компьютере они не конфликтовали друг с другом, а каждый работал в своей собственной виртуальной среде.

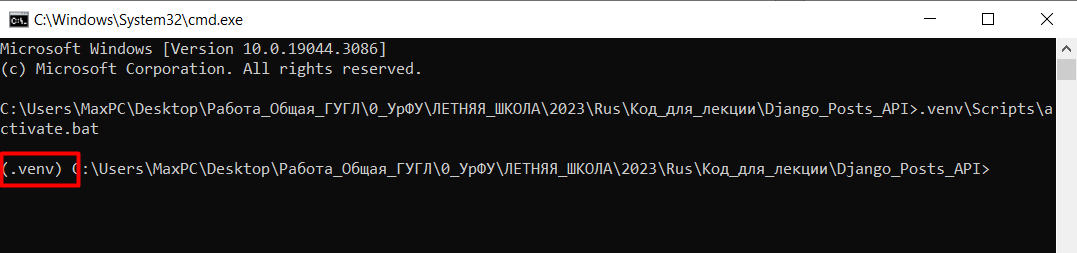
Чтобы сделать это, введите приведенную ниже команду в командной строке и нажмите **enter**:

python –m venv .venv

**Активируйте** виртуальную среду с помощью следующей команды:

.venv\Scripts\activate.bat

Если виртуальная среда была успешно активирована, то вы увидите специальные символы в консоли:



**Установим** Django. Для этого запустим следующую команду:

pip install django

в командной строке или терминале.

**Создадим** новый проект Django:

django-admin startproject project\_name

(вы можете заменить "project\_name" на желаемое название проекта). Но мы будем использовать это название для проекта.

**Перейдите** в каталог проекта, используя команду:

cd project\_name.

По умолчанию Django имеет встроенную систему безопасности для блокирования запросов из разных доменов. Чтобы отключить эту защиту, мы установим модуль «django-cors-headers», используя приведенную ниже команду :

pip install django-cors-headers

**Шаг 2: Создание приложения Django**

Создайте новое приложение Django командой:

python manage.py startapp app\_name

(вы можете заменить "app\_name" на желаемое название приложения).

Теперь нам нужно открыть проект в VS Code:

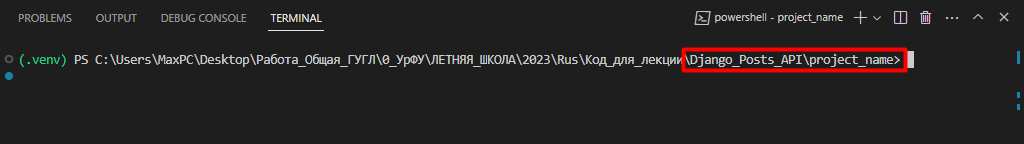
* + закройте командную строку,
  + запустите VS Code,
  + откройте папку проекта через меню *File -> Open Folder*.
  + запустите терминал из текущего проекта. Это можно сделать через меню *Terminal -> New Terminal*.

В *терминале* нам нужно перейти в папку проекта, чтобы дальнейшие команды выполнялись в соответствующей папке.

Чтобы сделать это, введите команду в терминале:

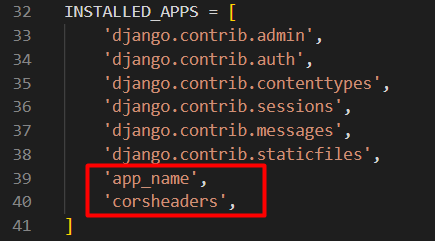
cd project\_name

После ввода команды терминал должен выглядеть примерно так:



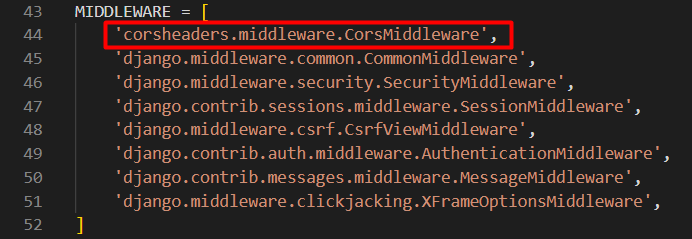
Добавим созданное приложение в настройки проекта.

Для этого: откройте **project\_name/settings.py** и добавьте **'app\_name'** в список **INSTALLED\_APPS**. Кроме того, нам нужно добавить сюда ссылку на модуль **'django-cors-headers'**», который мы установили ранее:

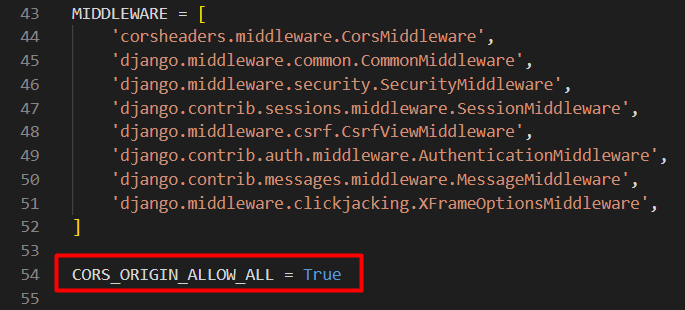


Кроме того, в **settings.py** файл добавим одну строку в промежуточное программное обеспечение, как показано на рисунке ниже:

'corsheaders.middleware.CorsMiddleware',



Ниже раздела промежуточного программного обеспечения добавьте разрешение на подключение любых внешних приложений к проекту, как показано на рисунке ниже:



Мы специально не используем модули безопасности в нашем приложении. Это делается только для упрощения и понимания кода. Этого не следует делать в рабочей версии по соображениям безопасности. В реальном проекте здесь будет записан массив с разрешенными доменами, например:

CORS\_ALLOWED\_ORIGINS= [

'http://localhost:3030',

]

**Шаг 3: Определение моделей**

Откройте **app\_name/models.py** файл и добавьте в него код для определения модели для постов:

from django.db import models

class Post(models.Model):

    title = models.CharField(max\_length=255)

    content = models.TextField()

    def \_\_str\_\_(self):

        return self.title

Далее необходимо отправить новые модели в базу данных. Для этого необходимо выполнить команды для миграции данных.

Первой командой создаем файл миграций:

python manage.py makemigrations

и далее отправляем изменения в базу данных:

python manage.py migrate

Теперь давайте создадим пользователя, который будет иметь административный доступ к приложению. Чтобы сделать это, введите следующую команду в терминале и следуйте инструкциям:

python manage.py createsuperuser

**Шаг 4: Создание представлений**

Откроем файл **app\_name/views.py** и определим представления для отображения, создания, обновления и удаления записей:

from django.shortcuts import render, get\_object\_or\_404, redirect

from .models import Post

def post\_list(request):

    posts = Post.objects.all()

    return render(request, 'post\_list.html', {'posts': posts})

def post\_create(request):

    if request.method == 'POST':

        title = request.POST['title']

        content = request.POST['content']

        Post.objects.create(title=title, content=content)

        return redirect('post\_list')

    return render(request, 'post\_create.html')

def post\_edit(request, pk):

    post = get\_object\_or\_404(Post, pk=pk)

    if request.method == 'POST':

        post.title = request.POST['title']

        post.content = request.POST['content']

        post.save()

        return redirect('post\_list')

    return render(request, 'post\_edit.html', {'post': post})

def post\_delete(request, pk):

    post = get\_object\_or\_404(Post, pk=pk)

    if request.method == 'POST':

        post.delete()

        return redirect('post\_list')

    return render(request, 'post\_delete.html', {'post': post})

**Шаг 5: Определим URL-адреса и шаблоны**

Создайте **urls.py** файл в каталоге приложения (**app\_name/urls.py**), если он еще не существует, и добавьте в него следующий код:

from . import views

from django.urls import path, include

from .api import PostViewSet

urlpatterns = [

    path('', views.post\_list, name='post\_list'),

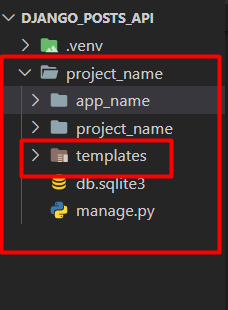
    path('create/', views.post\_create, name='post\_create'),

    path('edit/<int:pk>/', views.post\_edit, name='post\_edit'),

    path('delete/<int:pk>/', views.post\_delete, name='post\_delete'),

]

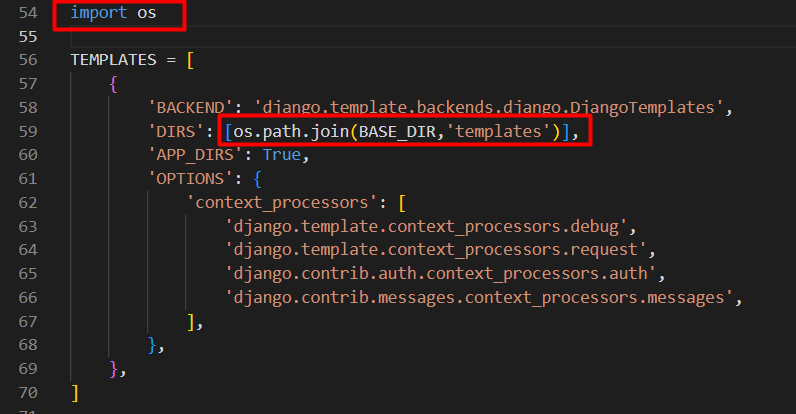
Создадим папку «**templates»** в основной папке проекта, где находится файл «**manage.py**»:



Внутри папки **templates** создайте необходимые файлы шаблонов представлений:

**post\_list.html, post\_create.html, post\_edit.html, post\_delete.html, base.html.**

Внесем изменения в файл **settings.py** проекта **project\_name** в разделе **TEMPLATES**. Здесь мы показываем, где находится папка с шаблонами, которые будут использоваться представления для вывода данных на страницу. Также добавим строку «**import os**» выше раздела **TEMPLATES**, чтобы была возможность использовать встроенный метод «**path**» для указания пути:

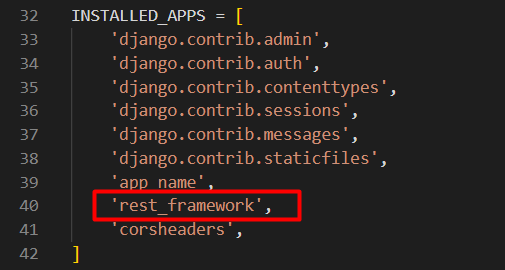


**Step 6: Настройка API**

Для обеспечения доступа к данным проекта из внешних источников, установим дополнительный модуль **'djangorestframework'** командой:

pip install djangorestframework

Добавим **'rest\_framework'** к списку установленных приложений **NSTALLED\_APPS** в файле **project\_name/settings.py**:



Создадим новый файл с названием **api.py** в директории приложения (**app\_name/api.py**) и добавим в него следующий код:

from rest\_framework import serializers, viewsets

from .models import Post

class PostSerializer(serializers.ModelSerializer):

    class Meta:

        model = Post

        fields = '\_\_all\_\_'

class PostViewSet(viewsets.ModelViewSet):

    queryset = Post.objects.all()

    serializer\_class = PostSerializer

Обновим код в файле **app\_name/urls.py**, чтобы добавить новые маршруты для работы с API (замените код полностью):

from . import views

from django.urls import path, include

from .api import PostViewSet

from rest\_framework.routers import DefaultRouter

router = DefaultRouter()

router.register('posts', PostViewSet)

urlpatterns = [

    path('', views.post\_list, name='post\_list'),

    path('create/', views.post\_create, name='post\_create'),

    path('edit/<int:pk>/', views.post\_edit, name='post\_edit'),

    path('delete/<int:pk>/', views.post\_delete, name='post\_delete'),

    path('', include(router.urls)),

]

Обновим файл маршрутов основного проект **project\_name/urls.py** для того, чтобы все созданные маршруты приложения **app\_name** были доступны из основного проекта (замените код файла полностью):

from django.contrib import admin

from django.urls import path, include

urlpatterns = [

    path('admin/', admin.site.urls),

    path('', include('app\_name.urls')),

]

Теперь давайте реализуем разметку шаблонов, которые были добавлены в приложение ранее (файлы шаблонов **post\_create.html**, **post\_edit.html, post\_delete.html**, **post\_list.html** и **base.html**, наряду со стилями CSS):

***post\_create.html***:

{% extends 'base.html' %}

{% block content %}

  <h1>Create a New Post</h1>

  <form method="POST">

    {% csrf\_token %}

    <div class="form-group">

      <label for="title">Title:</label>

      <input type="text" name="title" class="form-control" required>

    </div>

    <div class="form-group">

      <label for="content">Content:</label>

      <textarea name="content" rows="4" class="form-control" required></textarea>

    </div>

    <button type="submit" class="btn btn-primary">Create</button>

  </form>

{% endblock %}

***post\_edit.html:***

{% extends 'base.html' %}

{% block content %}

  <h1>Edit Post</h1>

  <form method="POST">

    {% csrf\_token %}

    <div class="form-group">

      <label for="title">Title:</label>

      <input type="text" name="title" class="form-control" value="{{ post.title }}" required>

    </div>

    <div class="form-group">

      <label for="content">Content:</label>

      <textarea name="content" rows="4" class="form-control" required>{{ post.content }}</textarea>

    </div>

    <button type="submit" class="btn btn-primary">Save Changes</button>

  </form>

{% endblock %}

***post\_delete.html:***

{% extends 'base.html' %}

{% block content %}

  <h1>Delete Post</h1>

  <p>Are you sure you want to delete the post "{{ post.title }}"?</p>

  <form method="POST">

    {% csrf\_token %}

    <button type="submit" class="btn btn-danger">Delete</button>

    <a href="{% url 'post\_list' %}" class="btn btn-secondary">Cancel</a>

  </form>

{% endblock %}

***post\_list.html:***

{% extends 'base.html' %}

{% block content %}

  <h1>Posts</h1>

  <table class="table table-striped">

    <thead>

      <tr>

        <th>Title</th>

        <th>Content</th>

        <th>Actions</th>

      </tr>

    </thead>

    <tbody>

      {% for post in posts %}

        <tr>

          <td>{{ post.title }}</td>

          <td>{{ post.content }}</td>

          <td>

            <a href="{% url 'post\_edit' post.pk %}" class="btn btn-primary">Edit</a>

            <a href="{% url 'post\_delete' post.pk %}" class="btn btn-danger">Delete</a>

          </td>

        </tr>

      {% empty %}

        <tr>

          <td colspan="3">No posts available.</td>

        </tr>

      {% endfor %}

    </tbody>

  </table>

  <a href="{% url 'post\_create' %}" class="btn btn-success">Create New Post</a>

{% endblock %}

***base.html***

<!DOCTYPE html>

<html>

<head>

  <meta charset="UTF-8">

  <title>My Django App</title>

  <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">

  <!-- Add any additional CSS files or stylesheets here -->

<style>

    body {

      font-family: Arial, sans-serif;

    }

    h1 {

      color: #333;

    }

    form {

      margin-bottom: 20px;

    }

    .form-group {

      margin-bottom: 15px;

    }

    .btn {

      padding: 8px 16px;

      border-radius: 4px;

      font-size: 14px;

      text-decoration: none;

    }

    .btn-primary {

      background-color: #007bff;

      color: #fff;

      border: none;

    }

    .btn-danger {

      background-color: #dc3545;

      color: #fff;

      border: none;

    }

    .btn-secondary {

      background-color: #6c757d;

      color: #fff;

      border: none;

    }

    .btn-success {

      background-color: #28a745;

      color: #fff;

      border: none;

    }

    .table {

      width: 100%;

      border-collapse: collapse;

    }

    .table th,

    .table td {

      padding: 8px;

      border-bottom: 1px solid #ccc;

    }

    .table th {

      background-color: #f8f9fa;

      text-align: left;

    }

    .table td {

      vertical-align: top;

    }

    .table td:last-child {

      text-align: right;

    }

    .table-striped tbody tr:nth-child(even) {

      background-color: #f2f2f2;

    }

  </style>

</head>

<body>

  <nav class="navbar navbar-expand-lg navbar-dark bg-dark">

    <a class="navbar-brand" href="/">My Django App</a>

    <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarNav" aria-controls="navbarNav" aria-expanded="false" aria-label="Toggle navigation">

      <span class="navbar-toggler-icon"></span>

    </button>

    <div class="collapse navbar-collapse" id="navbarNav">

      <ul class="navbar-nav">

        <li class="nav-item">

          <a class="nav-link" href="{% url 'post\_list' %}">Posts</a>

        </li>

        <!-- Add more navigation links if needed -->

      </ul>

    </div>

  </nav>

  <div class="container">

    {% block content %}

    {% endblock %}

  </div>

  <!-- Add any additional JavaScript files or scripts here -->

  <script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>

  <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>

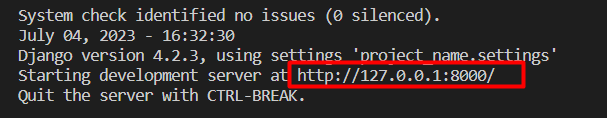
</body>

</html>

На этом шаге вам нужно проверить, все ли работает правильно. Для этого запустите приложение с помощью команды:

python manage.py runserver

и перейдите по указанному URL-адресу в браузере:



Теперь давайте добавим функциональность авторизации и регистрации в приложение Django и ограничим доступ к публикуемой информации для незарегистрированных пользователей.

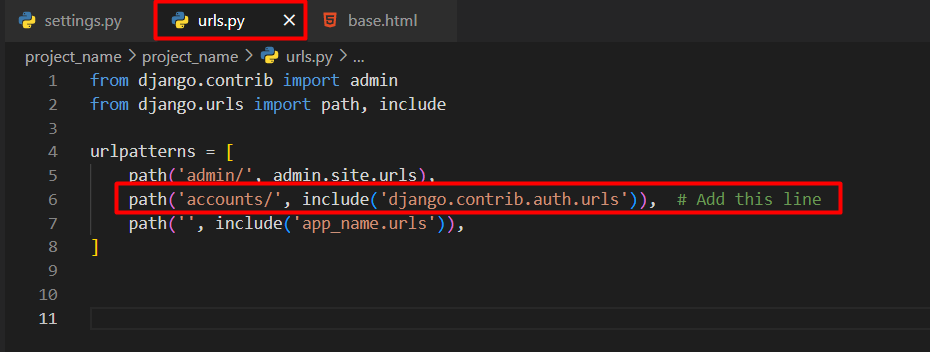
**Шаг 1: Установка необходимых пакетов**

Установим систему аутентификации Django, запустив команду:

pip install django.contrib.auth

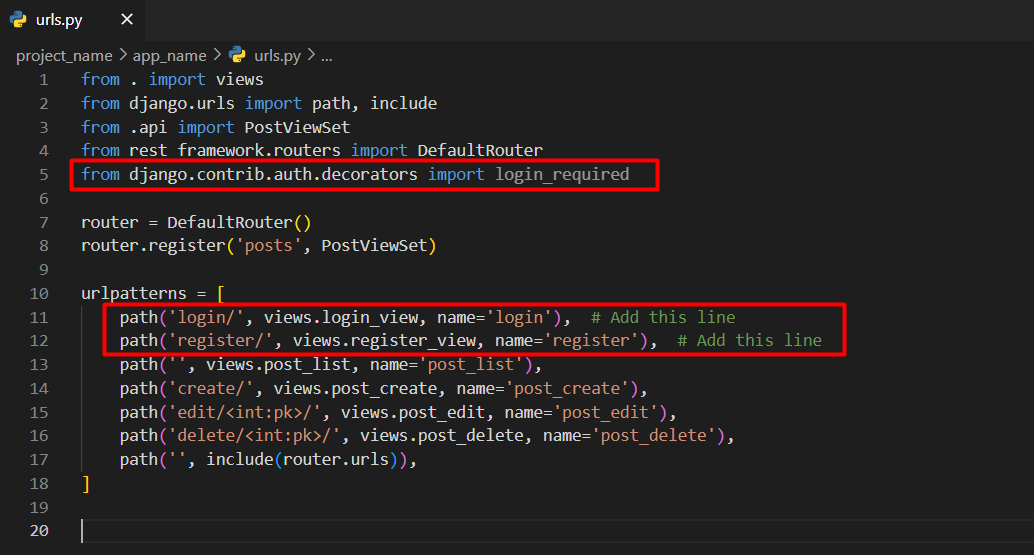
**Шаг 2: Обновим URL-адреса проекта**

Откройте **project\_name/urls.py** и обновите шаблоны URL-адресов следующим образом:

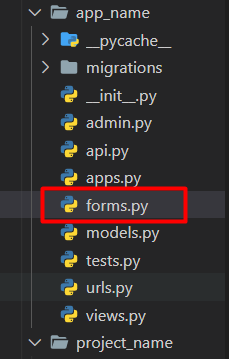


**Шаг 3: Обновим URL-адреса приложения**

Откройте **app\_name/urls.py** файл, добавьте следующие инструкции импорта в верхней части файла и обновите шаблоны URL-адресов, чтобы включить маршруты для входа в систему и регистрации:



Создадим форму **UserRegistrationForm** в файле **forms.py** (файл необходимо создать, если его нет) каталога приложения (**app\_name/forms.py**):



и добавьте в него приведенный ниже код (**forms.py**):

from django import forms

from django.contrib.auth.forms import UserCreationForm

from django.contrib.auth.models import User

class UserRegistrationForm(UserCreationForm):

    email = forms.EmailField()

    class Meta:

        model = User

        fields = ['username', 'email', 'password1', 'password2']

В **app\_name/views.py** добавьте следующие импорты:

from django.contrib.auth.decorators import login\_required

from .forms import UserRegistrationForm

from django.contrib.auth import authenticate, login

from django.contrib.auth.forms import AuthenticationForm

from django.contrib import messages

а также добавьте функции представлений **login\_view** и **register\_view** (вставьте представленный ниже код сразу после объявления импортов):

def login\_view(request):

    if request.method == 'POST':

        form = AuthenticationForm(request, data=request.POST)

        if form.is\_valid():

            username = form.cleaned\_data.get('username')

            password = form.cleaned\_data.get('password')

            user = authenticate(username=username, password=password)

            if user is not None:

                login(request, user)

                return redirect('post\_list')

        messages.error(request, 'Invalid username or password.')

    else:

        form = AuthenticationForm()

    return render(request, 'login.html', {'form': form})

def register\_view(request):

    if request.method == 'POST':

        form = UserRegistrationForm(request.POST)

        if form.is\_valid():

            form.save()

            messages.success(request, 'Registration successful. You can now log in.')

            return redirect('login')

    else:

        form = UserRegistrationForm()

    return render(request, 'register.html', {'form': form})

**Шаг 5: Обновим шаблоны**

Создадим шаблон **login.html** в папке **templates**:

{% extends 'base.html' %}

{% block content %}

  <h1>Login</h1>

  <form method="POST">

    {% csrf\_token %}

    {{ form.as\_p }}

    <button type="submit" class="btn btn-primary">Login</button>

  </form>

{% endblock %}

Создадим шаблон **register.html** в папке **templates**:

{% extends 'base.html' %}

{% block content %}

  <h1>Register</h1>

  <form method="POST">

    {% csrf\_token %}

    {{ form.as\_p }}

    <button type="submit" class="btn btn-primary">Register</button>

  </form>

{% endblock %}

Обновите шаблон **post\_list.html**, включающий условную логику для отображения постов только для прошедших проверку подлинности пользователей:

{% extends 'base.html' %}

{% block content %}

{% if user.is\_authenticated %}

<h1>Posts</h1>

<table class="table table-striped">

  <table class="table table-striped">

    <thead>

      <tr>

        <th>Title</th>

        <th>Content</th>

        <th>Actions</th>

      </tr>

    </thead>

    <tbody>

      {% for post in posts %}

      <tr>

        <td>{{ post.title }}</td>

        <td>{{ post.content }}</td>

        <td>

          <a href="{% url 'post\_edit' post.pk %}" class="btn btn-primary">Edit</a>

          <a href="{% url 'post\_delete' post.pk %}" class="btn btn-danger">Delete</a>

        </td>

      </tr>

      {% empty %}

      <tr>

        <td colspan="3">No posts available.</td>

      </tr>

      {% endfor %}

    </tbody>

  </table>

</table>

<a href="{% url 'post\_create' %}" class="btn btn-success">Create New Post</a>

{% else %}

<h1>Please log in to view posts.</h1>

<p><a href="{% url 'login' %}" class="btn btn-primary">Login</a></p>

<p><a href="{% url 'register' %}" class="btn btn-secondary">Register</a></p>

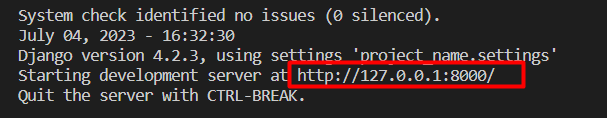
{% endif %}

{% endblock %}

Запустите проект, чтобы проверить его работоспособность. Для этого выполните приведенную ниже команду:

python manage.py runserver

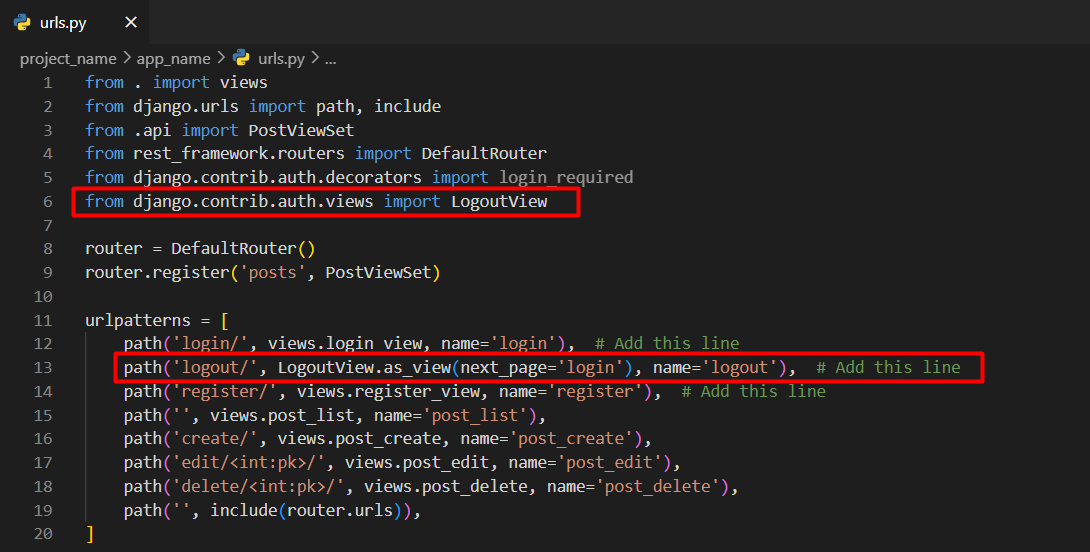
и перейдите по адресу, указанному в терминале:



Теперь давайте добавим в приложение функцию выхода из системы. Чтобы сделать это, нам нужно соответствующим образом обновить URL-адреса и представления:

**Шаг 1: Обновим URL-адреса проекта**

Откройте файл **app\_name/urls.py**, добавьте следующую инструкцию импорта и обновите шаблоны URL-адресов, чтобы включить путь выхода из системы:



**Шаг 2: Обновим шаблон «base.html»**

Обновим раздел навигации в шаблоне **base.html** для добавления в разметку ссылки на выход из системы.

Для этого: замените весь код в разделе **<body>** приведенным ниже кодом:

<body>

  <nav class="navbar navbar-expand-lg navbar-dark bg-dark">

    <a class="navbar-brand" href="/">My Django App</a>

    <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarNav"

      aria-controls="navbarNav" aria-expanded="false" aria-label="Toggle navigation">

      <span class="navbar-toggler-icon"></span>

    </button>

    <div class="collapse navbar-collapse" id="navbarNav">

      <ul class="navbar-nav">

        <li class="nav-item">

          <a class="nav-link" href="{% url 'post\_list' %}">Posts</a>

        </li>

**<!-- Add a logout link -->**

**{% if user.is\_authenticated %}**

**<li class="nav-item">**

**<a class="nav-link" href="{% url 'logout' %}">Logout</a>**

**</li>**

**{% endif %}**

      </ul>

    </div>

  </nav>

  <div class="container">

    {% block content %}

    {% endblock %}

  </div>

  <!-- Add any additional JavaScript files or scripts here -->

  <script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>

  <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>

</body>

**Шаг 3: Добавим шаблон подтверждения выхода из системы (опционально)**

Если вы хотите показать пользователю сообщение с подтверждением, когда он нажимает на ссылку выхода из системы, вы можете создать шаблон под названием **logout.html** в каталоге **templates**:

{% extends 'base.html' %}

{% block content %}

  <h1>Logout</h1>

  <p>Are you sure you want to log out?</p>

  <form method="POST" action="{% url 'logout' %}">

    {% csrf\_token %}

    <button type="submit" class="btn btn-primary">Logout</button>

  </form>

{% endblock %}

Теперь вы можете запустить приложение с помощью приведенной ниже команды и проверить его производительность:

python manage.py runserver

**Это всё ☺** Приложение создано. Оно обладает минимальной функциональностью, которая при необходимости может быть расширена. Оно также предоставляет возможность подключать стороннее приложение для получения информации о постах через встроенный API и создания новых постов в базе данных. Мы постараемся реализовать эту функциональность в **React.js** приложении в следующей лабораторной работе.