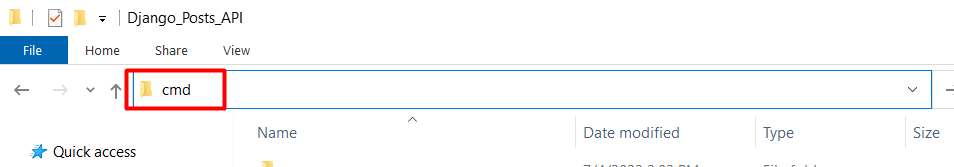
**Lab Work #1. Building a Posts Application with Django Framework**

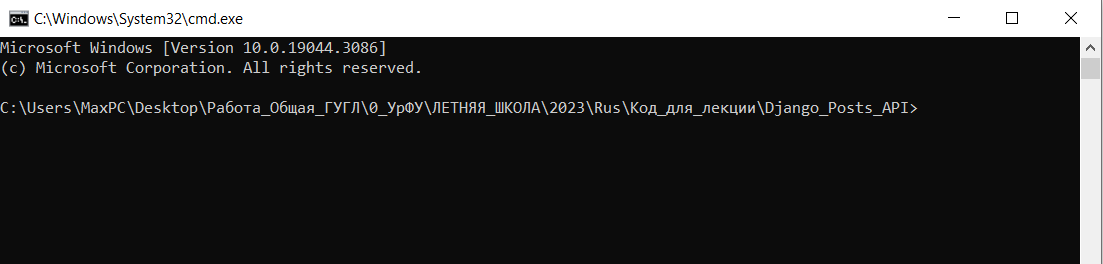
Create a new folder on the desktop with the name "Django\_Posts\_API".

Open the command prompt from the current folder.

To do this, you can simply enter the «**cmd**» command in the search bar of the current folder and press Enter.



After that, the command prompt opens:



**Step 1: Installing Django project**

**Create** a virtual environment for our project. The creation of a virtual environment is necessary so that if there are many projects on the same local machine, they do not conflict with each other, but each work in its own virtual environment.

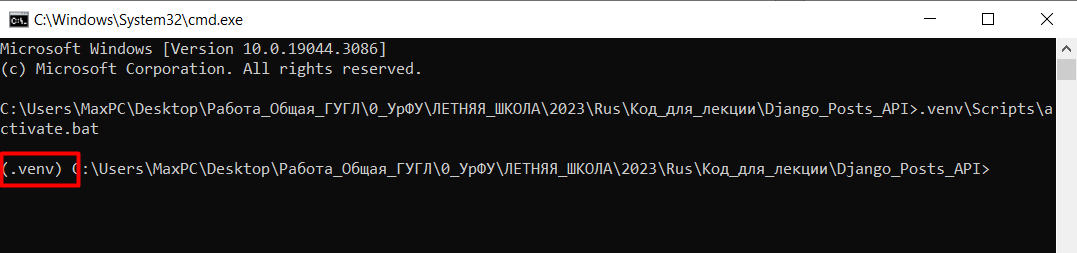
To do this, type the command below in the command prompt and press enter:

python –m venv .venv

**Activate** the virtual environment with the following command:

.venv\Scripts\activate.bat

If the virtual environment has been activated, then you will see special characters in the console:



**Install** Django by running

pip install django

in your command prompt or terminal.

**Create** a new Django project by running

django-admin startproject project\_name

(you can replace "project\_name" with your desired project name).

**Navigate** into the project directory using

cd project\_name

By default, Django has a built-in security system to block requests from different domains. To disable this protection, we will install the module «*django-cors-headers* » using the command below:

pip install django-cors-headers

**Step 2: Create the Django App**

Create a new Django app by running

python manage.py startapp app\_name

(you can replace "app\_name" with your desired app name).

Now we need to open the project in VS Code:

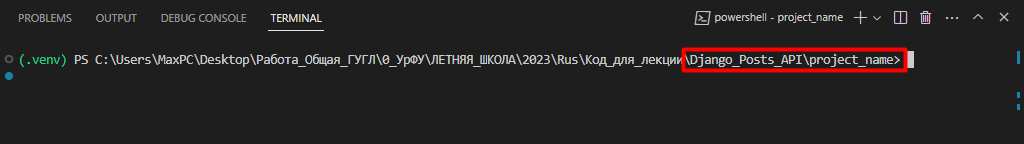
* + close the command prompt,
  + run VS Code,
  + open the project folder via the *File -> Open Folder* menu.
  + launch the terminal from the current project. This can be done via the *Terminal -> New Terminal* menu.

In the *terminal*, we need to go to the project folder so that further commands are executed in the appropriate folder.

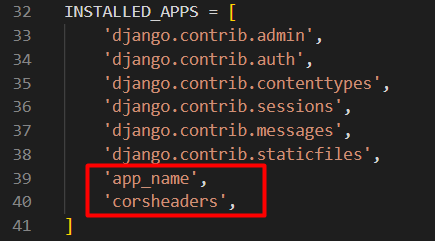
To do this, type the command in the terminal:

cd project\_name

After entering the command, the terminal should look something like this:

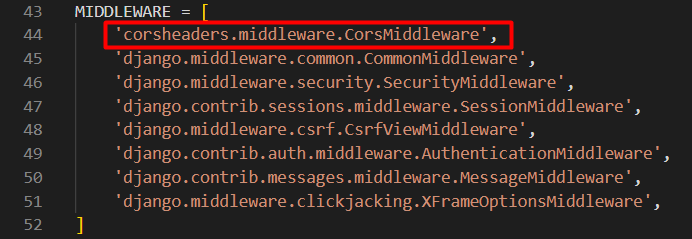


Add the app to the project's settings by opening **project\_name/settings.py** and adding **'app\_name'** to the **INSTALLED\_APPS** list. In addition, we need to add a link here to the «**django-cors-headers**» module that we installed earlier:

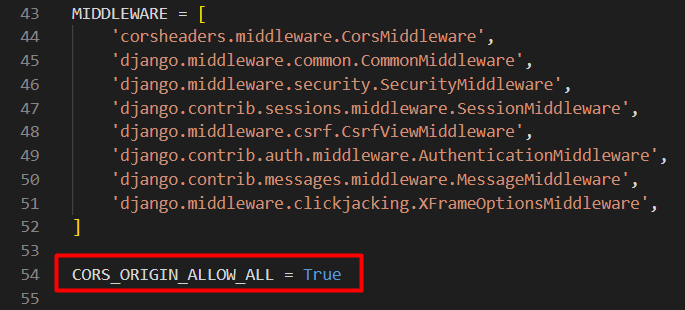


Also, in the **settings.py** file, add one line to the middleware, as shown in the figure below:

'corsheaders.middleware.CorsMiddleware',



Below the middleware section, add permission to connect any external applications to the project, as shown in the figure below:



We do not specifically use security modules in our application. This is done only to simplify and understand the code. This should not be done in the production version for security reasons. In a real project, an array with allowed domains will be written here, for example:

CORS\_ALLOWED\_ORIGINS= [

'http://localhost:3030',

]

**Step 3: Define the Model**

Open the **app\_name/models.py** file and define a model for your posts:

from django.db import models

class Post(models.Model):

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

    content = models.TextField()

    def \_\_str\_\_(self):

        return self.title

Run database migrations by executing command:

python manage.py makemigrations

and then:

python manage.py migrate

Now let’s create a user who will have administrative access to the application. To do this, type the following command in the terminal and follow the instructions:

python manage.py createsuperuser

**Step 4: Creating Views**

Open **app\_name/views.py** and define views for listing, creating, updating, and deleting posts:

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})

**Step 5: Define URLs and Templates**

Create a **urls.py** file in the app directory (**app\_name/urls.py**) if it doesn't already exist, and add the following code:

from . import views

from django.urls import path, include

from .api import PostViewSet

urlpatterns = [

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

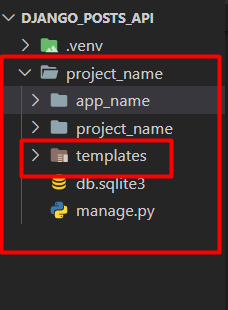
    path('create/', views.post\_create, name='post\_create'),

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

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

]

Create a «**templates»** folder in the main project folder where the file «**manage.py**» is located:



Create the templates for the views in a **templates** directory:

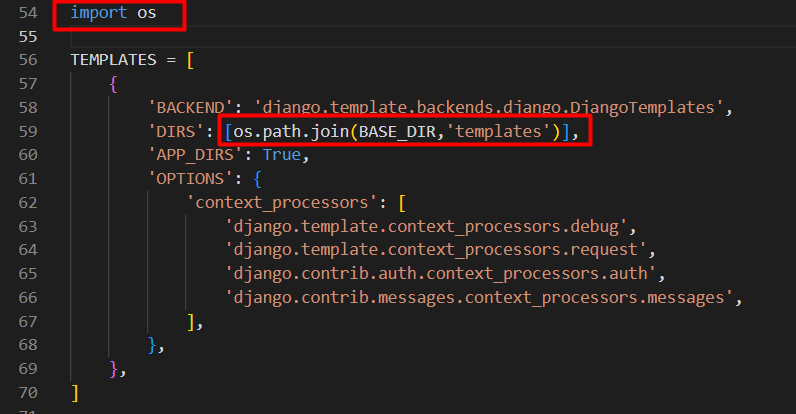
**post\_list.html,**

**post\_create.html,**

**post\_edit.html,**

**post\_delete.html.**

Make changes to the **settings.py** file of **project\_name**. Add the path to the templates folder and «**import os**»:

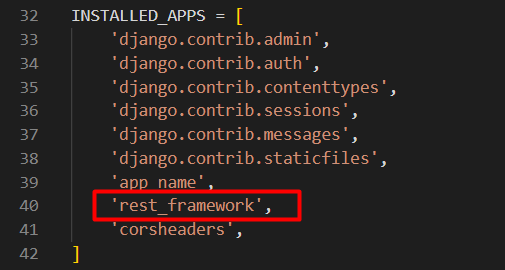


**Step 6: Set Up API**

Install the Django REST framework by running this command in terminal:

pip install djangorestframework

Add **'rest\_framework'** to the **INSTALLED\_APPS** list in **project\_name/settings.py**.



Create a new file called **api.py** in the app directory (**app\_name/api.py**) and add the following code:

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

Update **app\_name/urls.py** to include the API URLs:

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)),

]

Update **project\_name/urls.py** to include the URLs of application:

from django.contrib import admin

from django.urls import path, include

urlpatterns = [

    path('admin/', admin.site.urls),

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

]

Now let's implement the markup of templates that were added to the application earlier (template files **post\_create.html, post\_edit.html, post\_delete.html**, and **post\_list.html**, along with CSS styles):

***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>

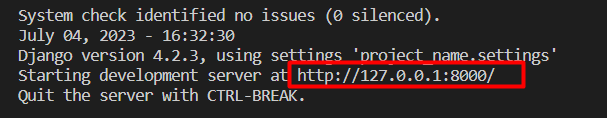
</body>

</html>

At this step, you need to check if everything is working correctly. To do this, run the application using the command

python manage.py runserver

and go to the specified url in the browser:



Now let’s add authorization and registration functionality to Django application and restrict access to post information for unregistered users.

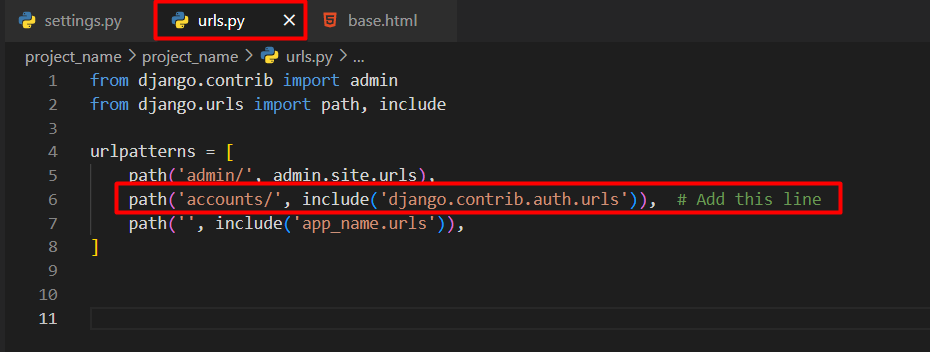
**Step 1: Install Required Packages**

Install Django's built-in authentication system by running:

pip install django.contrib.auth

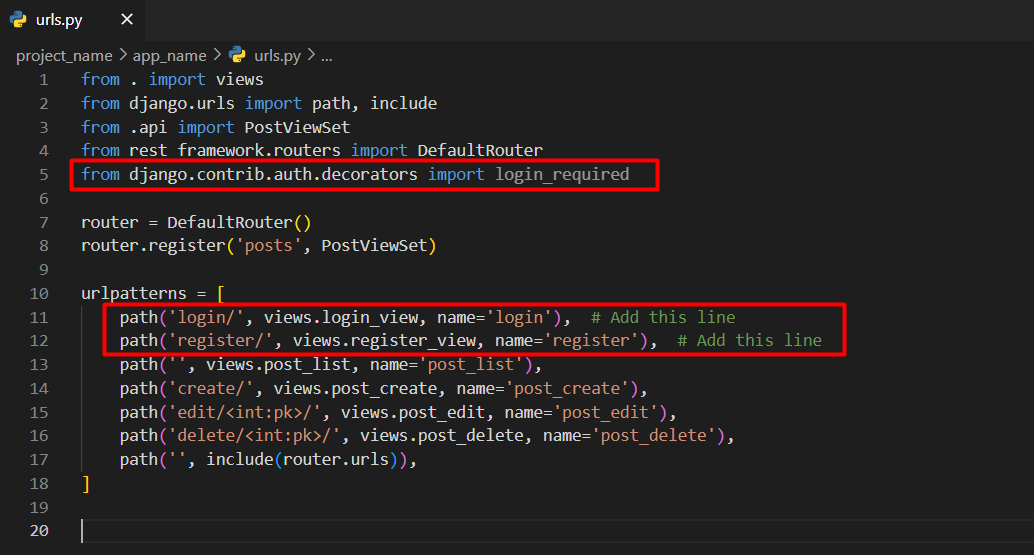
**Step 2: Update the Project's URLs**

Open the **project\_name/urls.py** file and update the URL patterns as follows:

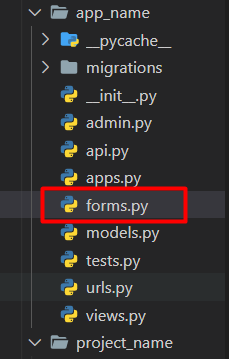


**Step 3: Update the App's URLs**

Open the **app\_name/urls.py** file, add the following import statements at the top and update the **URL patterns** to include login and registration paths:



Create the **UserRegistrationForm** form in a new **forms.py** file in your app directory (**app\_name/forms.py**):



and the code below to this file (**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']

In **app\_name/views.py** add the followings imports:

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

and create the **login\_view** and **register\_view** views (insert the code below right after «imports»):

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})

**Step 5: Update the Templates**

Create the **login.html** template in **templates** directory:

{% 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 %}

Create the **register.html** template in **templates** directory:

{% 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 %}

Update the **post\_list.html** template to include conditional logic to display the posts only for authenticated users:

{% 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>

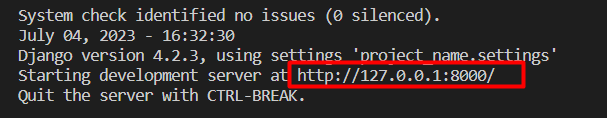
{% endif %}

{% endblock %}

Run the project to check its operability. For this, run the command below:

python manage.py runserver

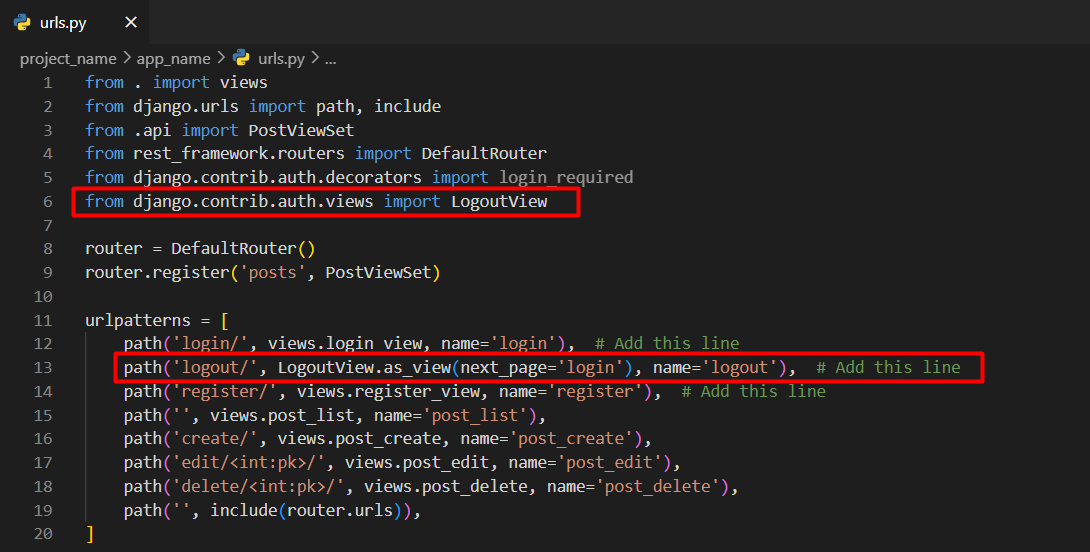
and go to the address specified in the terminal:



Now let’s add the logout functionality to the application. In order to do this we need to update the URLs and views accordingly:

**Step 1: Update the Project's URLs**

Open the **app\_name/urls.py** file, add the following import statement at the top and update the **URL patterns** to include the logout path:



**Step 2: Update the «base.html» template**

Update the navigation section in your **base.html** template to include a logout link.

For this: Replace all the code in **<body>** section with the code below:

<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>

**Step 3: Add a Logout Confirmation Template (optional)**

If you want to show a confirmation message to the user when they click on the logout link, you can create a template called **logout.html** in your app's **templates/app\_name** directory:

{% 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 %}

Now you can run the application with the command below and check its performance:

python manage.py runserver

**That's it ☺** The application has been created. It has minimal functionality, which can be expanded if necessary. It also provides the ability to connect a third-party application to receive posts through the built-in API and create new posts in the database. We will try to implement this functionality in the React.js application in the next lab work.

**Task:**

1. If the user has logged in to the site, print the following text next to the "Logout" link: "Hello, user\_name!".
2. Add a 100px footer to the main page (background color is the same as for navigation menu) with the text: "If there are rights, they are not protected. You can copy".