**Django - szkolenie 5**

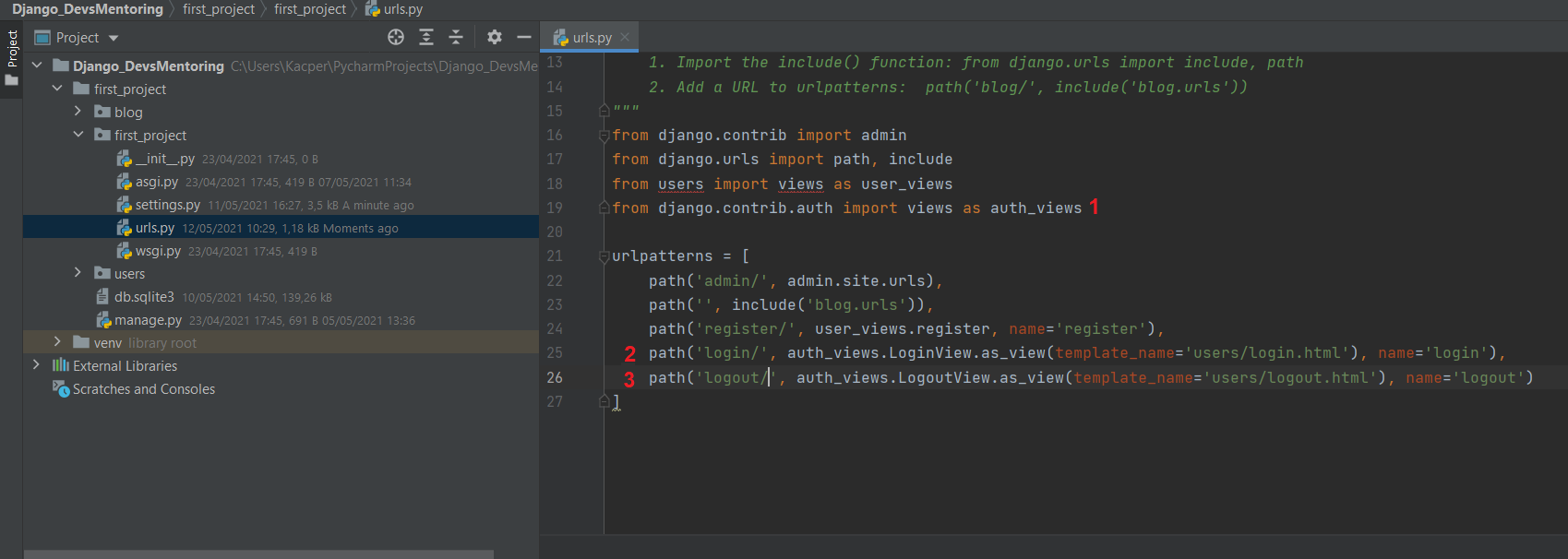
**Panel logowania**

**Wprowadzenie funkcjonalności logowania do panelu**

W poprzednim szkoleniu udało Nam się stworzyć panel rejestracji. Dzięki temu możemy teraz w łatwy sposób dodawać nowych użytkowników do tabeli Users. W Naszej aplikacji brakuje jednak jeszcze jednej, równie podstawowej i nierozłącznej funkcjonalności - możliwości logowania do utworzonego konta. Oczywiście możemy się logować do konta z poziomu panelu administracyjnego znajdującego się pod adresem /admin, jednak chcielibyśmy, aby było to umożliwione z poziomu wydzielonego endpoint-a aplikacji, jak np. /login. W tym szkoleniu zajmiemy się właśnie dostarczeniem tejże możliwości.

**Rejestracja nowych endpointów**

Zacznijmy od dodania do Naszej aplikacji nowych ścieżek dostępowych - /login oraz /logout. Udajmy się do pliku first\_project/urls.py i wprowadźmy zmiany takie jak poniżej:



from django.contrib import admin

from django.urls import path, include

from users import views as user\_views

from django.contrib.auth import views as auth\_views

urlpatterns = [

path("admin/", admin.site.urls),

path("", include('blog.urls', namespace='blog')),

path("register/", user\_views.register, name="register"),

path("login/", auth\_views.LoginView.as\_view(template\_name='users/login.html'), name="login"),

path("logout/", auth\_views.LogoutView.as\_view(template\_name='users/logout.html'), name="logout"),

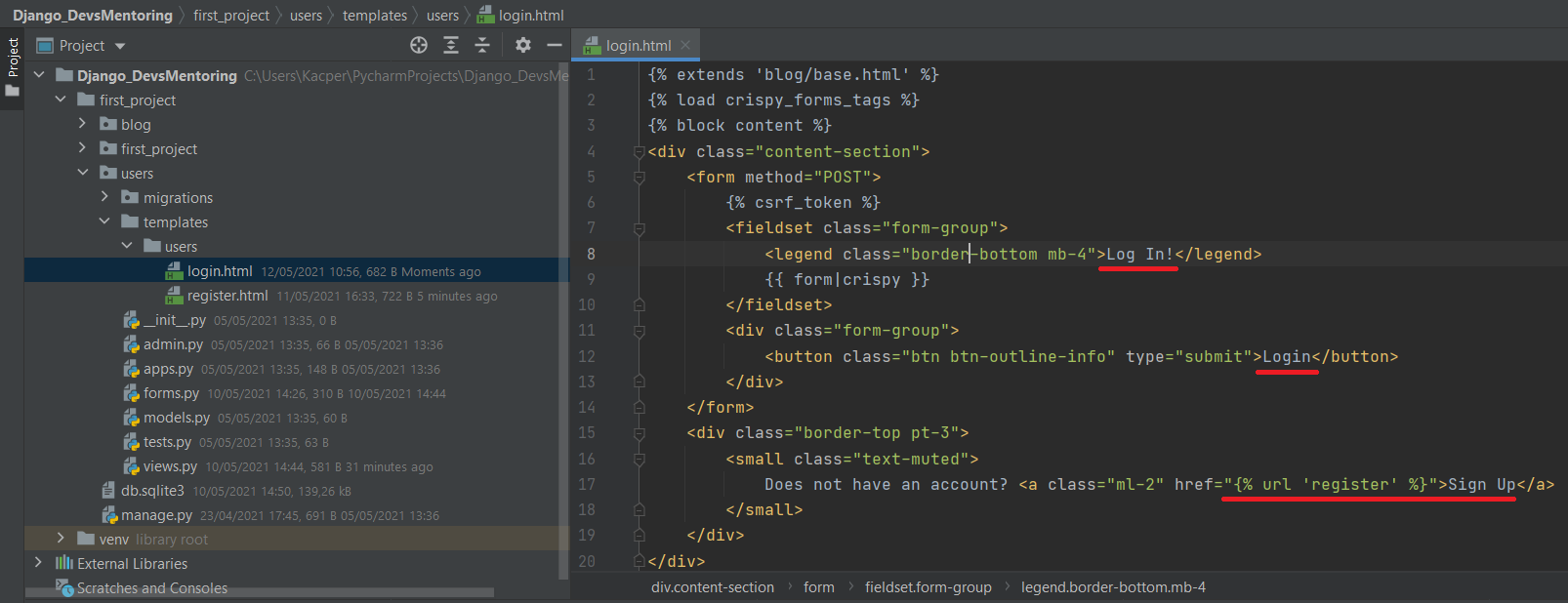
]

Punkt (1) to zaimportowanie potrzebnych widoków z biblioteki django.contrib.auth, które odpowiadają widokom związanym, np. z logowaniem, wylogowywaniem.

Punkty (2) i (3) to już rejestracje w aplikacji konkretnych endpointów. Jak widzisz, dodaliśmy endpoint login/ oraz logout/ o nazwach kolejno: ‘login’ oraz ‘logout’. Co więcej, widoki te powiązaliśmy z zaimportowanymi views z biblioteki django (auth\_views.LoginView dla logowania oraz auth\_views.LogoutView dla wylogowywania). Zauważ, że dla każdego z widoków dodaliśmy nowy argument (template\_name) będący ścieżką do pliku .html, w którym na następować renderowanie wyglądu podstron. Gdybyśmy nie podali tego argumentu, to silnik Django domyślnie szukałby plików login.html oraz logout.html w katalogu register. A tak, uciekając do rozwiązania przedstawionego na zrzucie, mamy możliwość pełnego kontrolowania plików projektowych.

Po przeprowadzeniu tak gruntownych zmian w pliku first\_project/urls.py, udajmy się do katalogu users/templates/users i utwórzmy w nim pliki, do których odnieśliśmy się z poziomu wcześniej modyfikowanego pliku urls.py. Będą to 2 szablony: login.html oraz logout.html.

Plik login.html będzie powstanie z copy-paste istniejącego już pliku register.html. Zmodyfikujemy jedynie pojedyncze pola, które przedstawiam poniżej:



{% extends 'blog/base.html' %}

{% load crispy\_forms\_tags %}

{% block content %}

<div class="content-section">

<form method="POST">

{% csrf\_token %}

<fieldset class="form-group">

<legend class="border-bottom mb-4">Log in!</legend>

{{ form|crispy }}

</fieldset>

<div class="form-group">

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

</div>

</form>

<div class="border-top pt-3">

<small class="text-muted">

Does not have an account? <a class="ml-2" href="{% url 'register' %}">Sign Up</a>

</small>

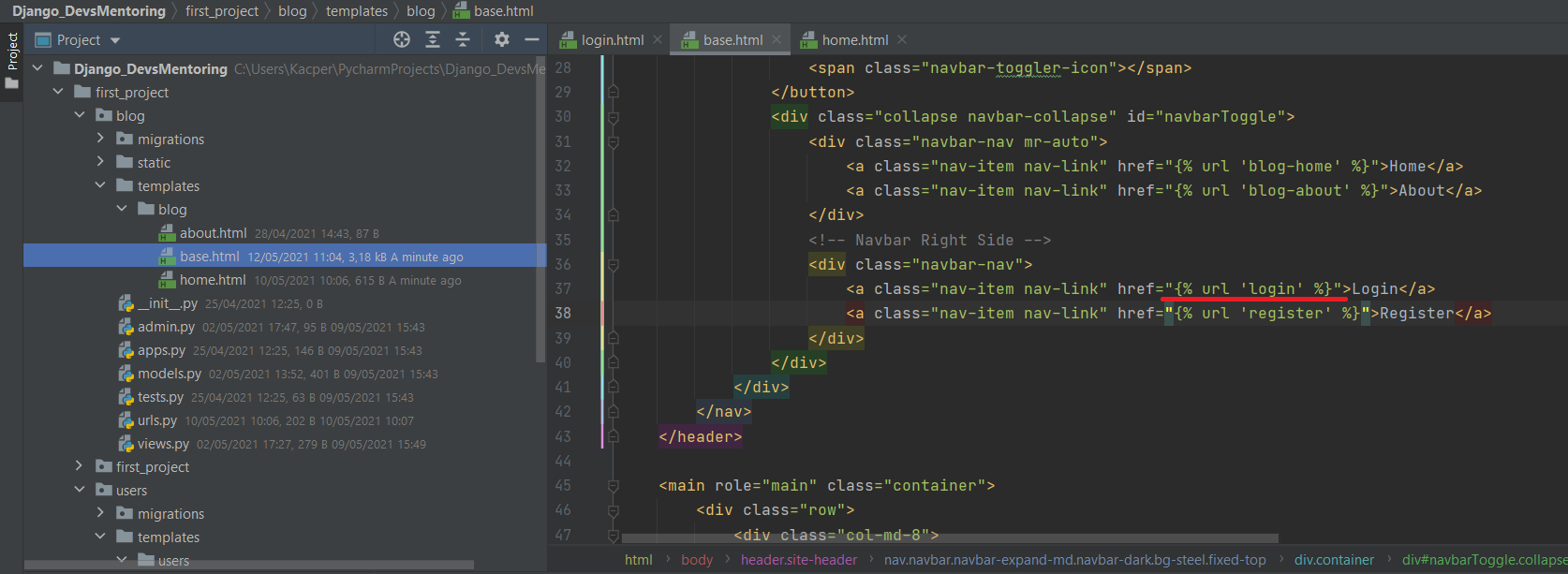
</div>

</div>

{% endblock %}

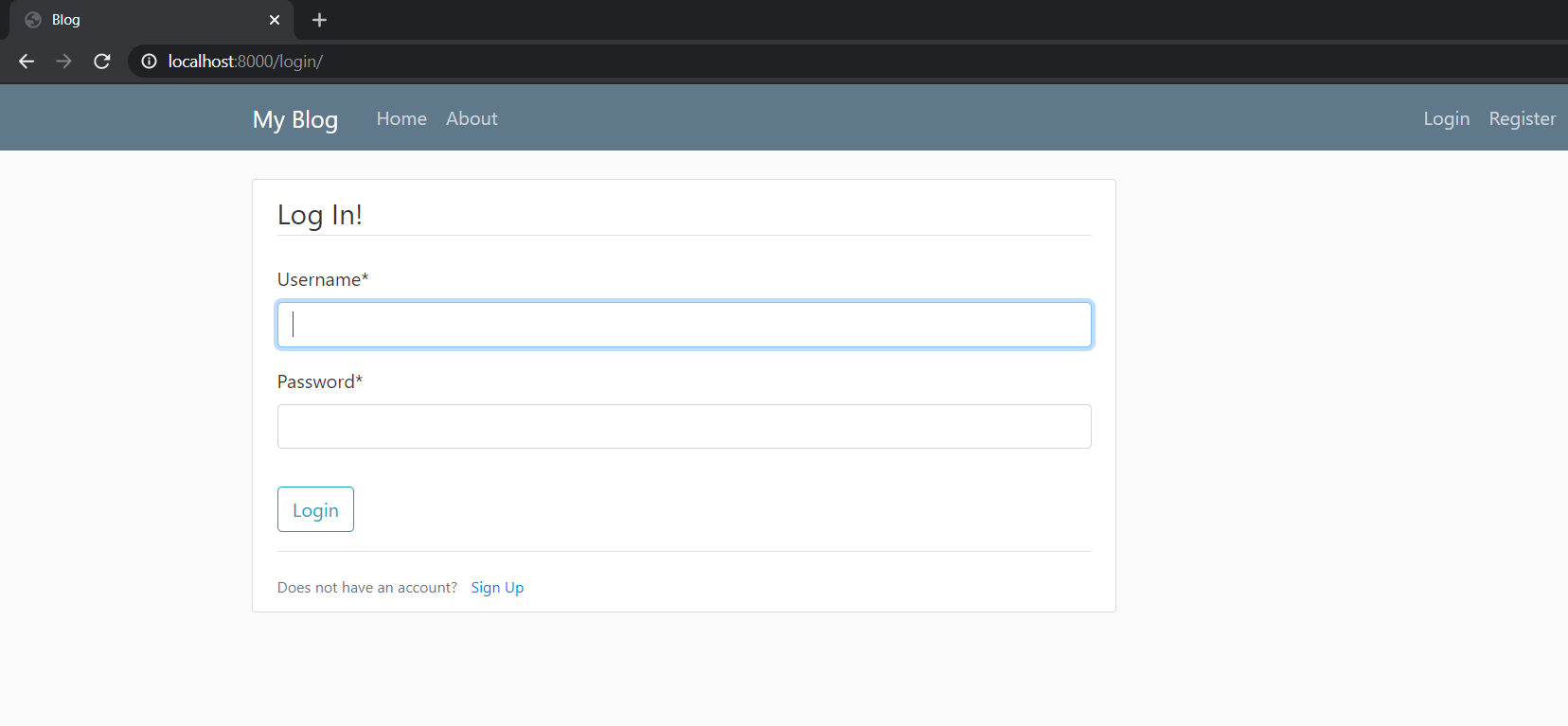
Po przekopiowaniu snippet-u kodu z pliku register.html do login.html, tak naprawdę musiałem zmodyfikować tylko pojedyncze nazwy pól - każdy napis Sign In zmieniłem na Login lub Sign Up oraz dodałem odpowiedni href (ostatnia zaznaczona na czerwono linia) tak, aby użytkownik był przekierowywany do formularza rejestracji po kliknięciu odnośnik Sign Up.

Na dopełnienie - w pliku base.html dodałem odnośnik do panelu logowania - tak, aby użytkownik po kliknięciu w napis Login na pasku nawigacyjnym, był przekierowywany do utworzonego panelu logowania.



<a class="nav-item nav-link" href="{% url 'login' %}">Login</a>

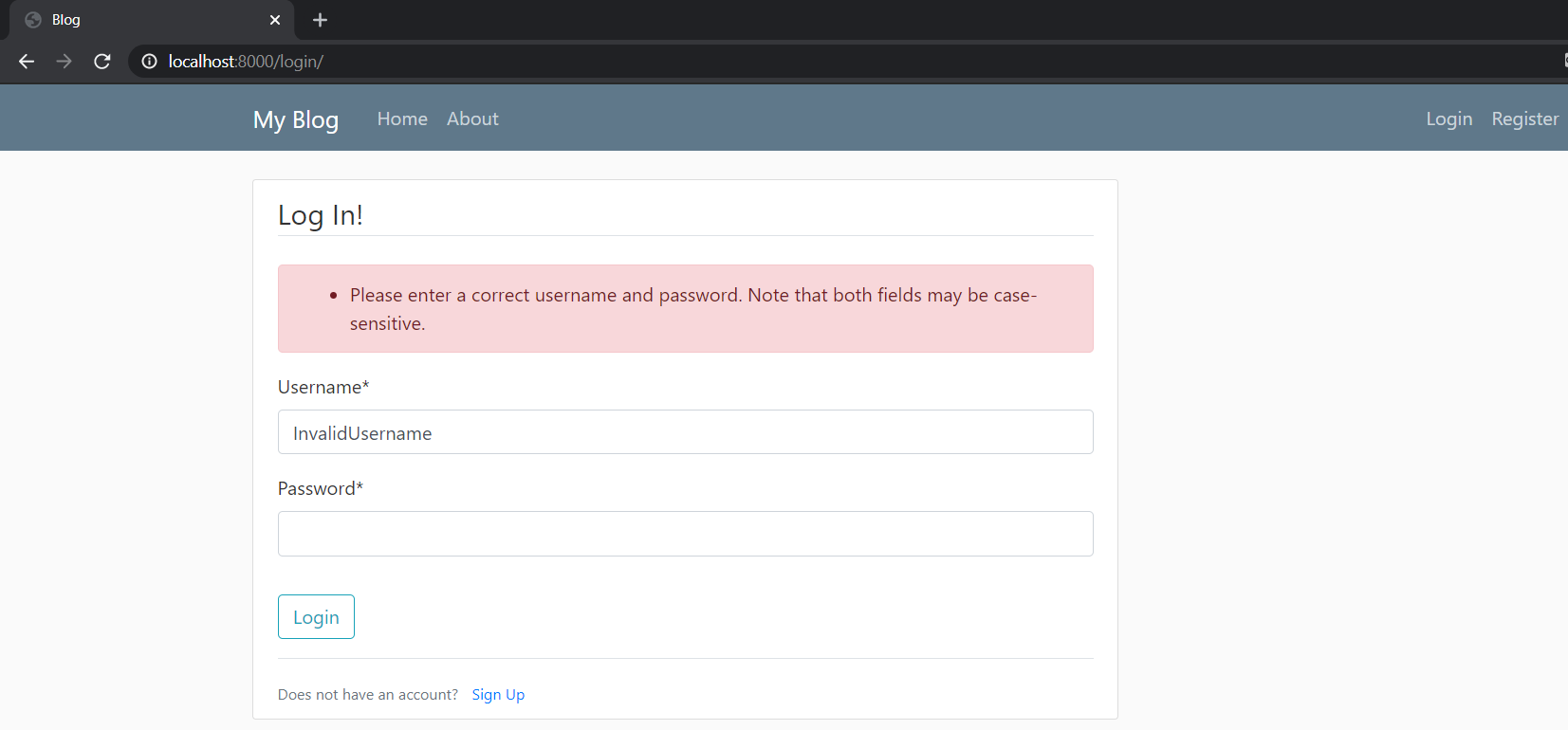
Spójrz, jaki efekt powinniśmy otrzymać po utworzeniu panelu logowania:



Jednak to nie wszystko…

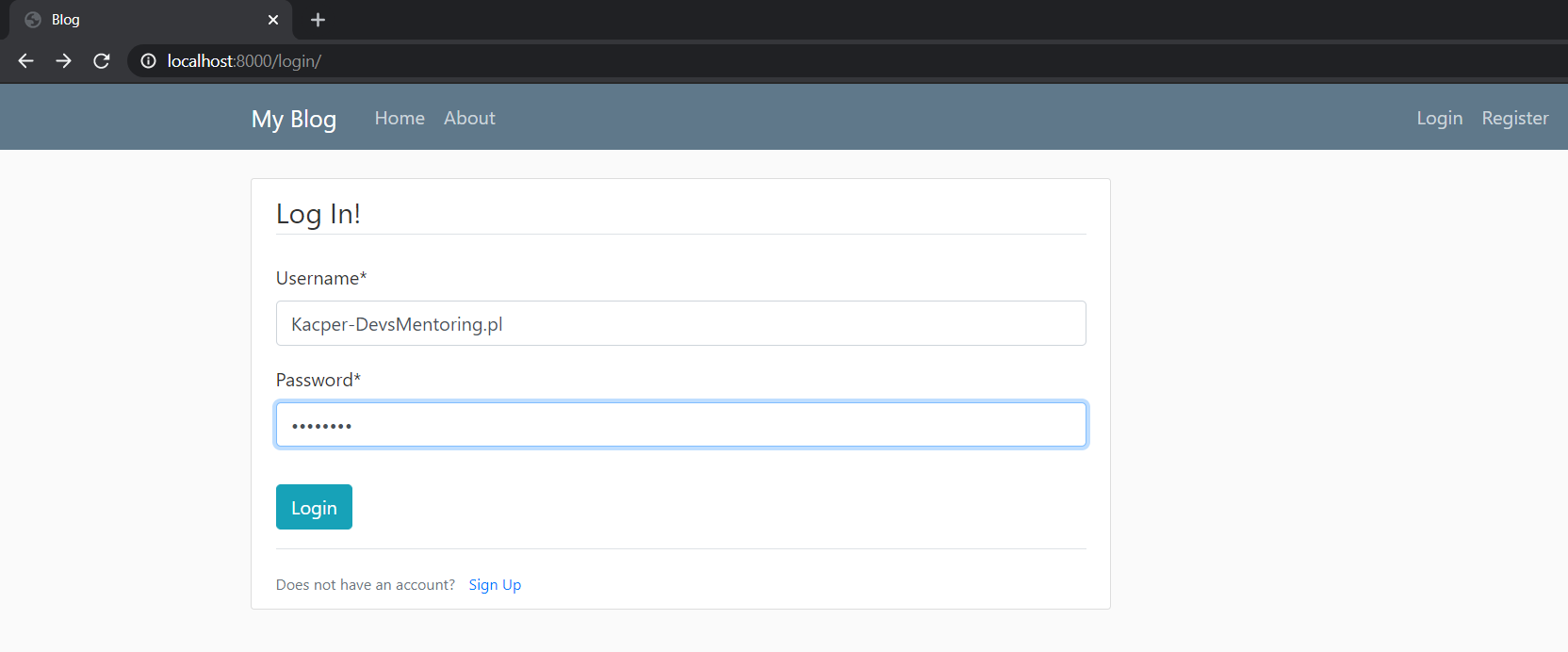
Udało Nam się utworzyć panel logowania! Sprawdźmy jednak, jak się on zachowa przy próbie podania niepoprawnych, a później poprawnych danych logowania.

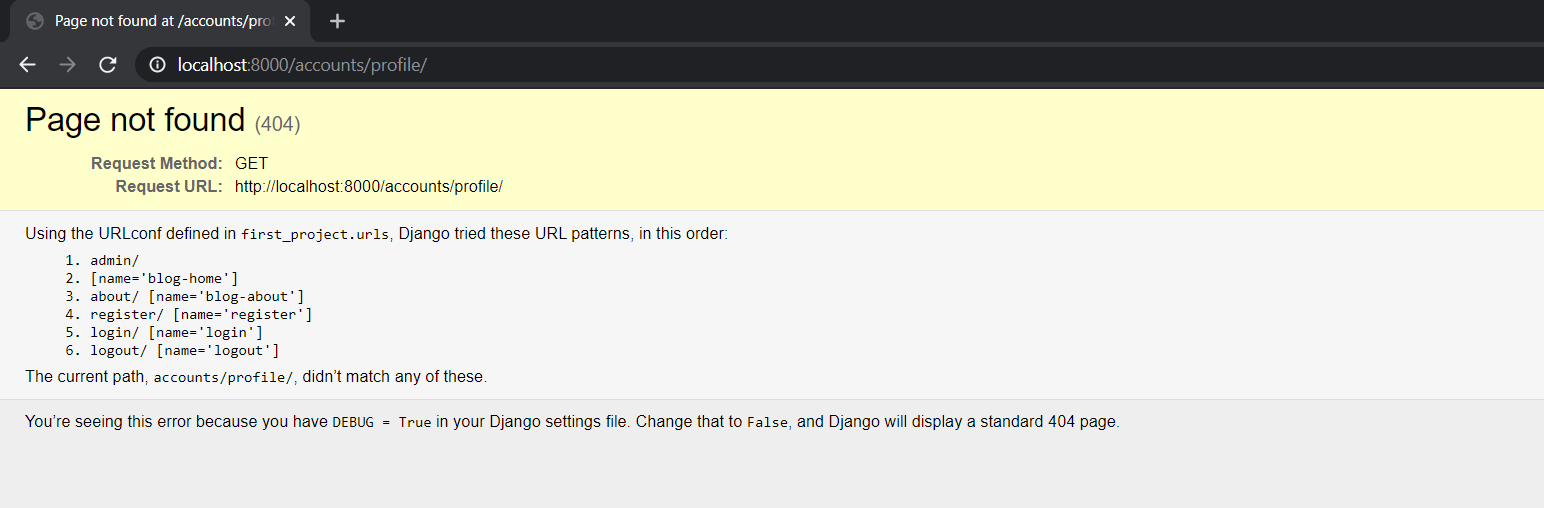
Po wpisaniu danych nieistniejącego użytkownika:



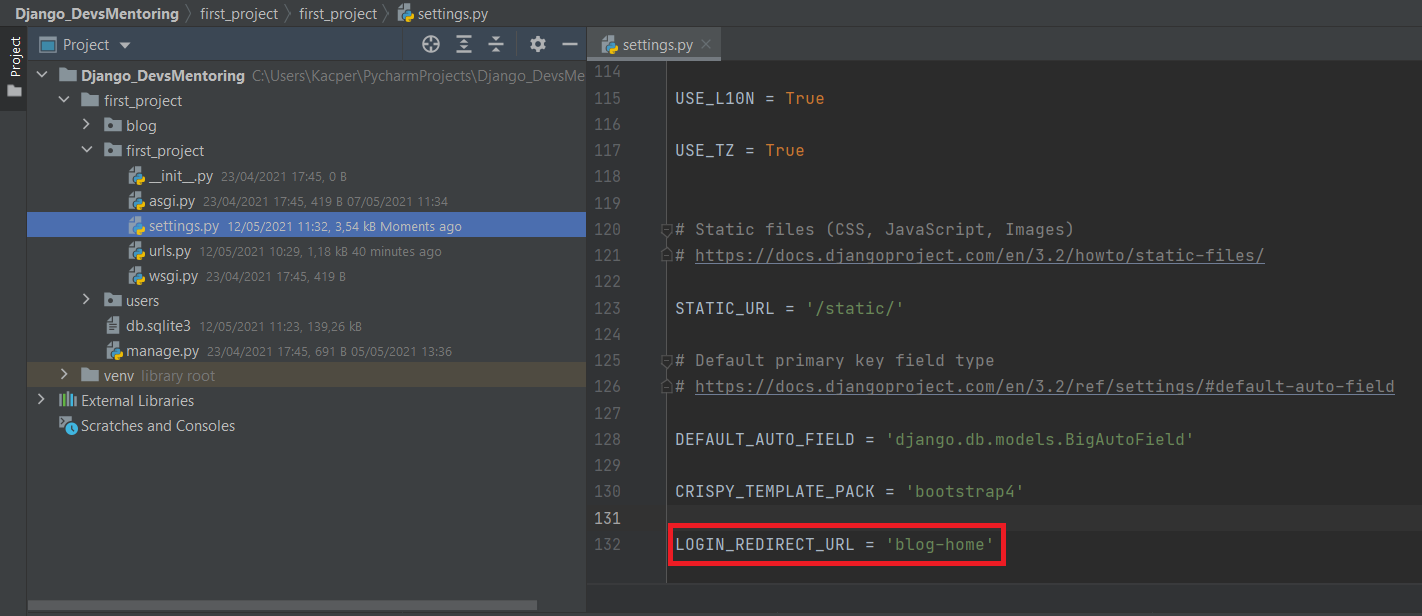
Efekt zgodny z oczekiwaniami. Nie chcemy przecież, aby system przepuszczał Nas po wprowadzeniu danych nieistniejącego użytkownika.

Sprawdźmy teraz, jak zachowa się Nasza aplikacja, gdy formularz logowania wypełnimy danymi utworzonego użytkownika we wcześniejszych szkoleniach, np. Kacper-DevsMentoring.pl



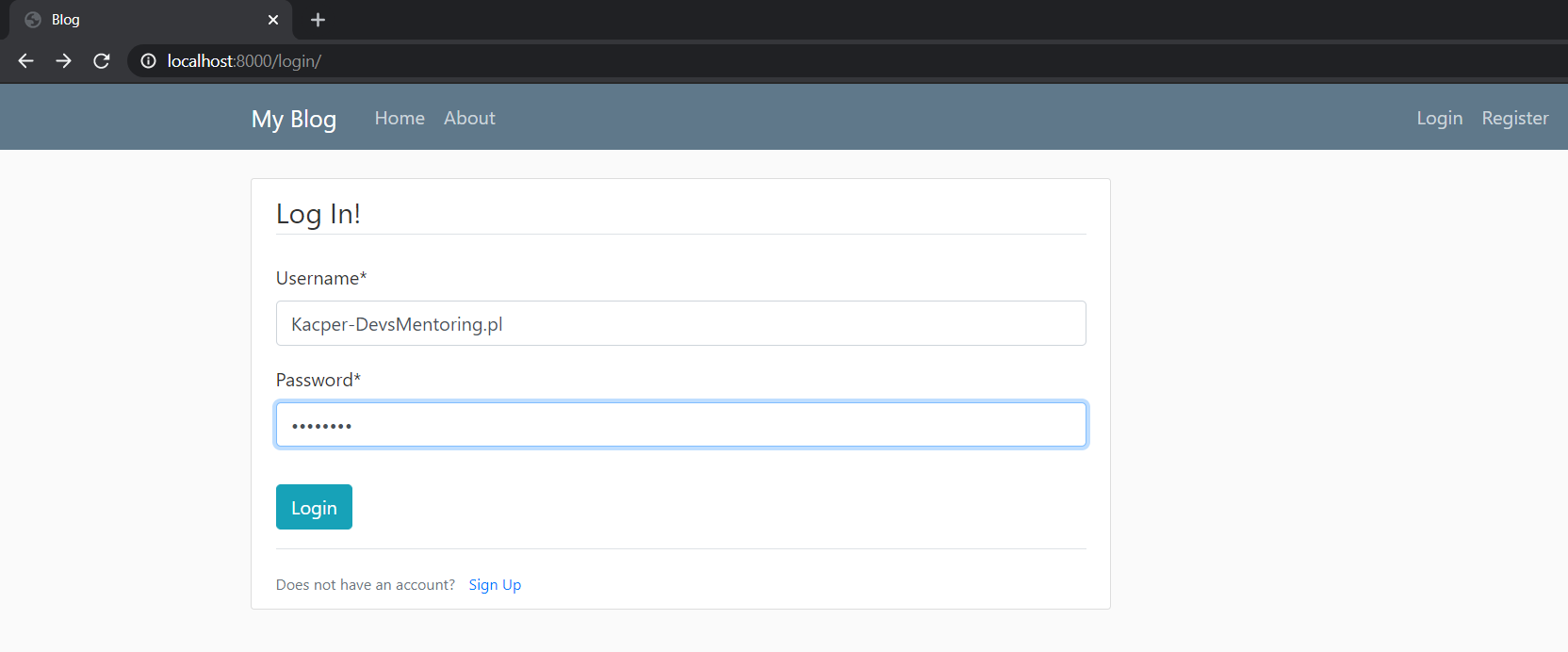


Jak widzimy, aplikacja się “wysypuje” po próbie zalogowania poprawnymi credentialsami. A to dlatego, że Django próbuje Nas domyślnie przekierować pod adres /accounts/profile/. Zmienimy to tak, aby po procesie poprawnego zalogowania, użytkownik przekierowany był na stronę główną bloga - home. Aby to wprowadzić, musimy przejść do pliku first\_project/settings.py i dodać w nim flagę LOGIN\_REDIRECT\_URL ustawioną na odpowiednią nazwę endpointa, do którego mamy być przekierowywani po prawidłowym zalogowaniu.

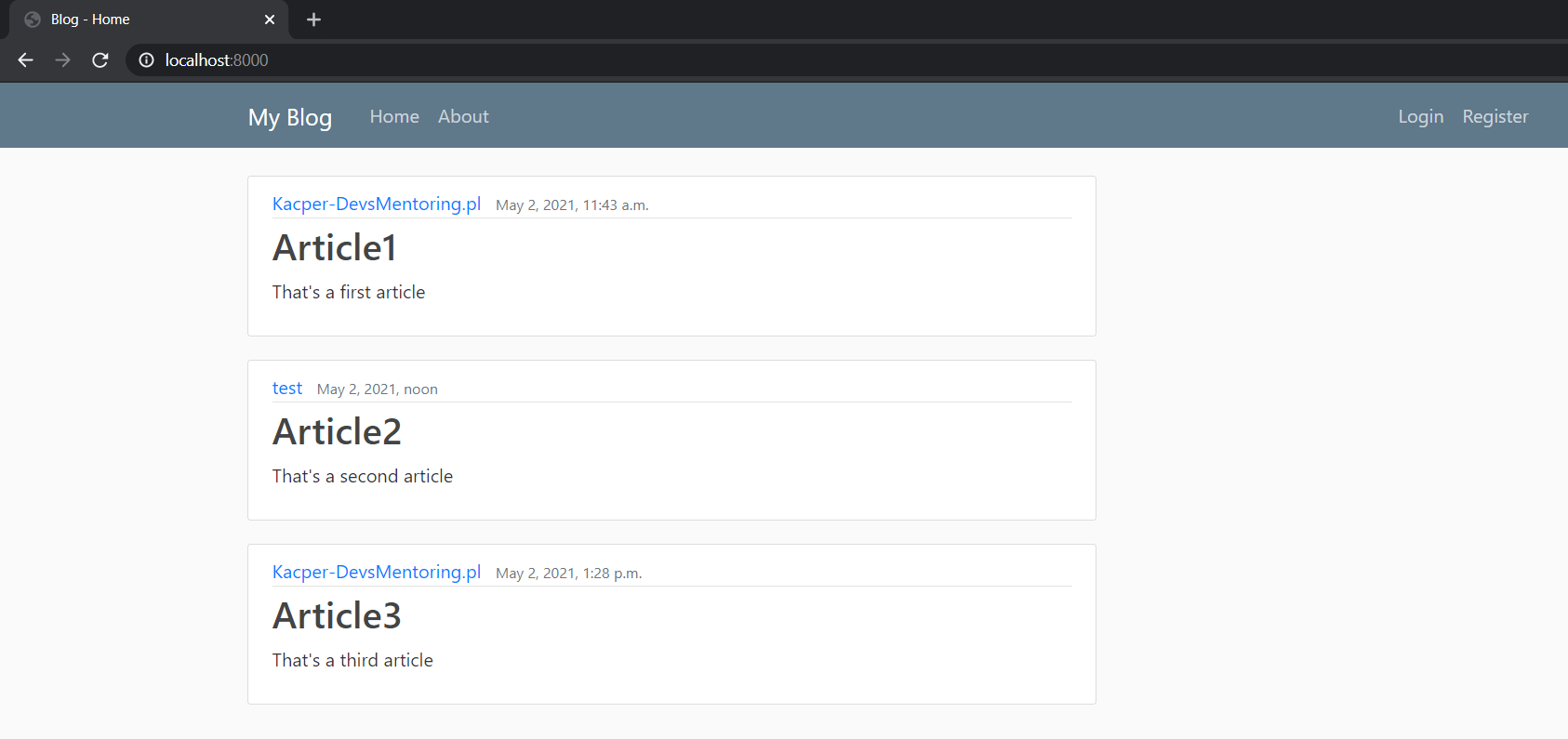


LOGIN\_REDIRECT\_URL = 'blog:blog\_home'

Sprawdźmy ponownie, czy udało Nam się poprawić funkcjonalność aplikacji. Po wpisaniu poniższych danych:



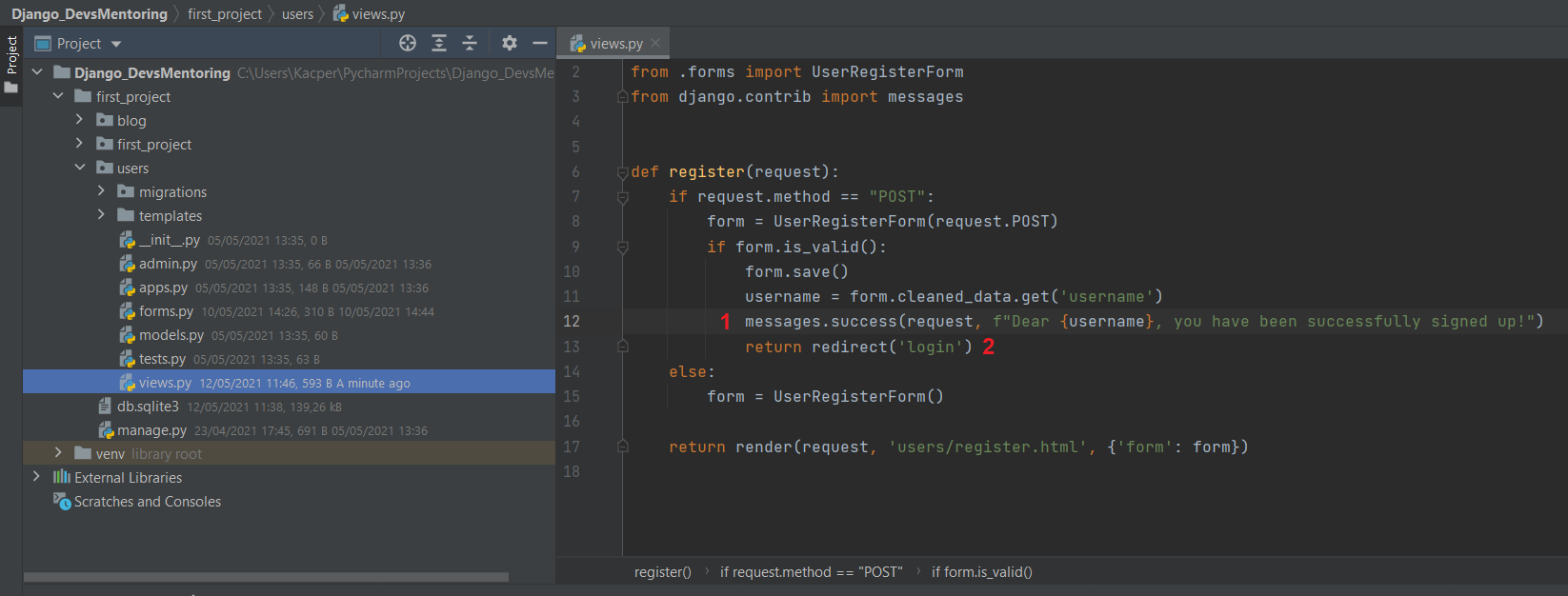
… i kliknięciu przycisku Login, otrzymamy:



Zatem proces działa prawidłowo - faktycznie zostajemy przekierowani do strony głównej (na chwilę obecną bez żadnych dodatkowych komunikatów o poprawnym zalogowaniu i możliwości przejścia do profilu użytkownika). Najważniejsze jednak, że funkcjonalność jest realizowana, póki co, w podstawowym stopniu.

**Dopracowanie zmian…**

Mając już do dyspozycji działający panel logowania, usprawnijmy nieco UX aplikacji. Nie wiem, czy pamiętasz, ale w tym momencie po przeprowadzeniu poprawnego procesu rejestracji, użytkownik przekierowywany jest na stronę blog-home. Niewątpliwie lepszym rozwiązaniem będzie, aby po rejestracji, redirectować go do panelu logowania, z poziomu którego będzie mógł bezpośrednio się zalogować.



from django.shortcuts import render, redirect

from .forms import UserRegisterForm

from django.contrib import messages

def register(request):

if request.method == 'POST':

form = UserRegisterForm(request.POST)

if form.is\_valid():

form.save()

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

messages.success(request, f"Dear {username}, you have been succesfully signed up!!")

return redirect(‘login')

else:

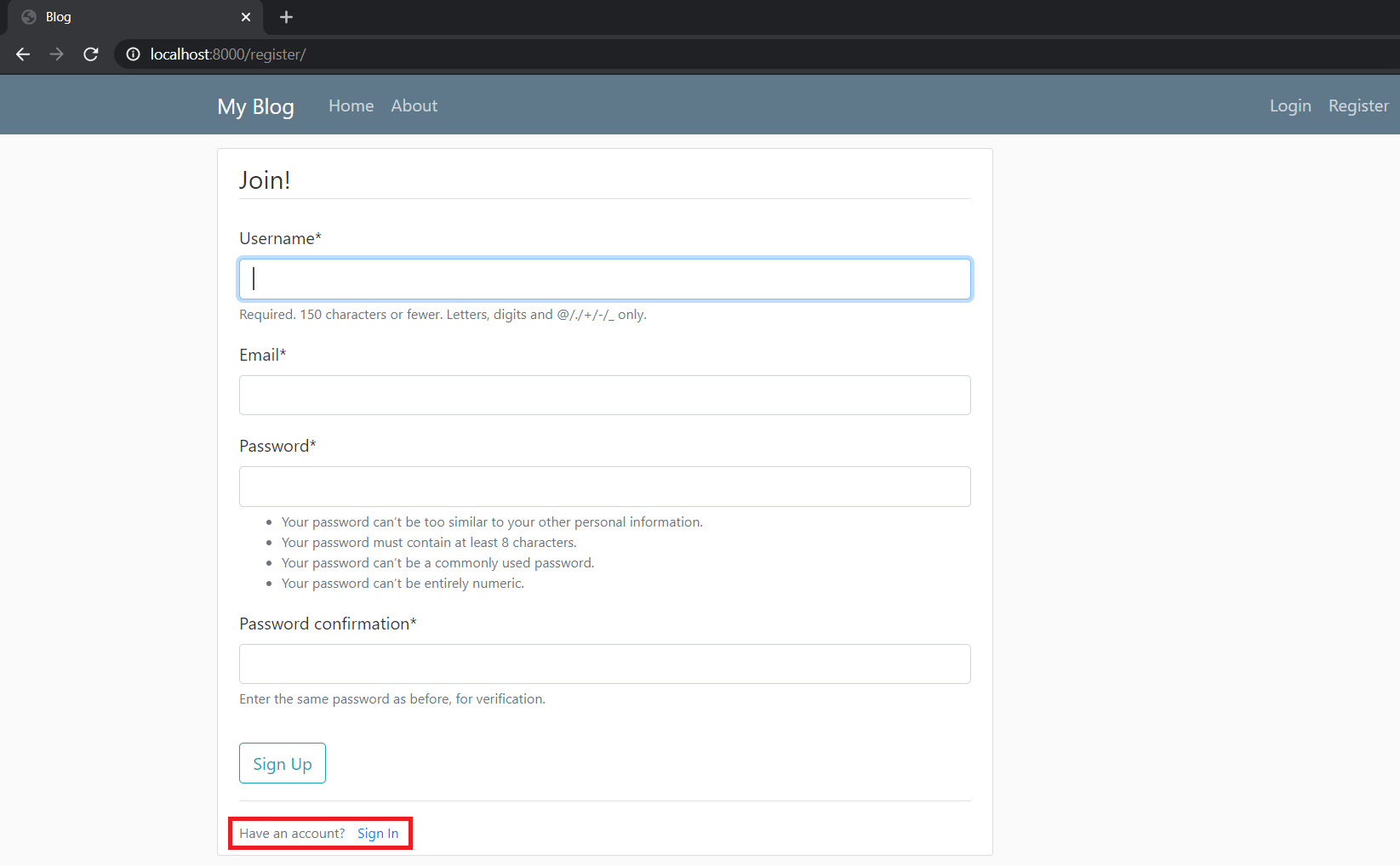
form = UserRegisterForm()

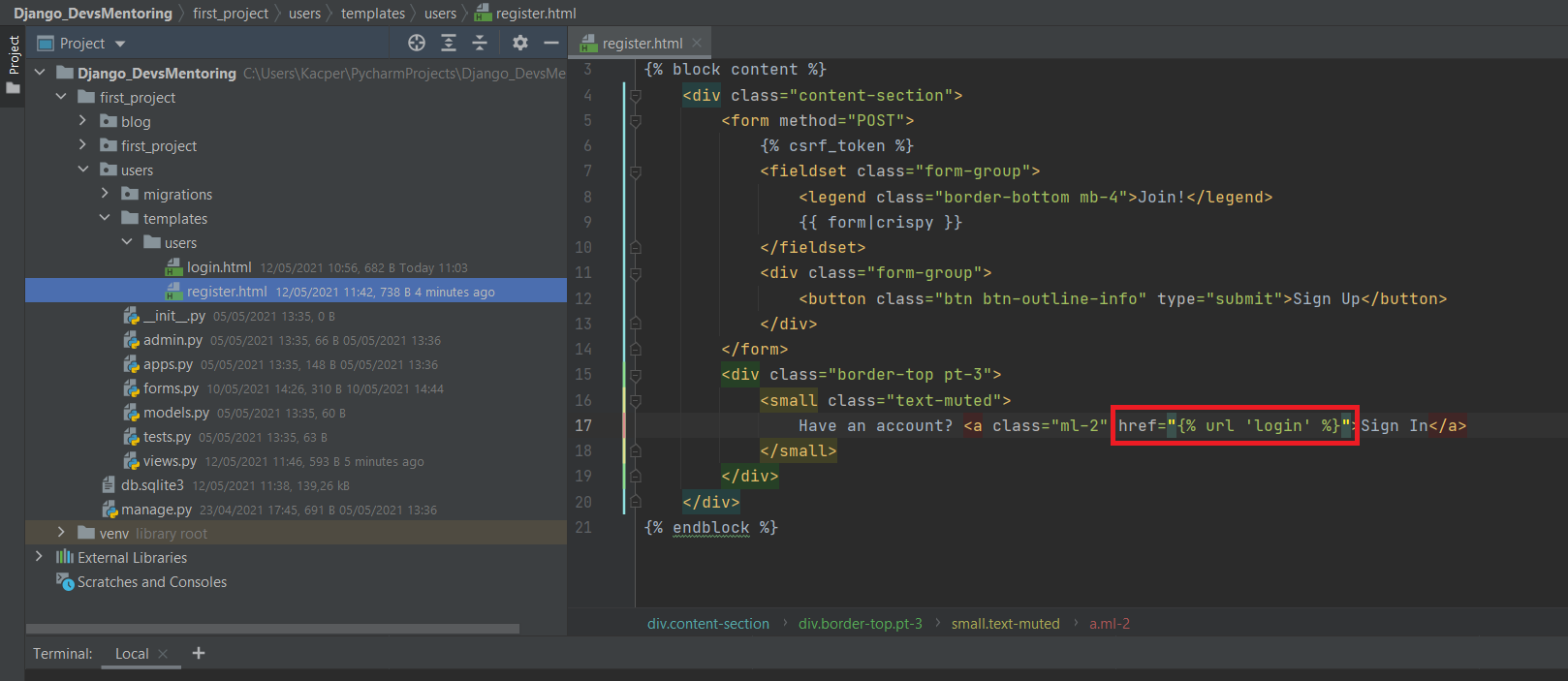
return render(request, 'users/register.html', {'form': form})

# Create your views here.

W pliku users/views.py i widoku register zmieniłem nieco wyświetlany komunikat o prawidłowym przebiegu rejestracji (1) oraz zmodyfikowałem punkt docelowy przekierowania użytkownika po rejestracji (z blog-home do endpointu login/).

Następnie zmienimy jeszcze przekierowanie do panelu logowania z poziomu rejestracji i napisu: Have an account? Sign In. Zmianę tę wprowadzimy w pliku users/templates/users/register.html.





{% extends 'blog/base.html' %}

{% load crispy\_forms\_tags %}

{% block content %}

<div class="content-section">

<form method="POST">

{% csrf\_token %}

<fieldset class="form-group">

<legend class="border-bottom mb-4">Join!</legend>

{{ form|crispy }}

</fieldset>

<div class="form-group">

<button class="btn btn-outline-info" type="submit"> Sign Up </button>

</div>

</form>

<div class="border-top pt-3">

<small class="text-muted">

Have an account? <a class="ml-2" href="{% url ‘login' %}">Sign In</a>

</small>

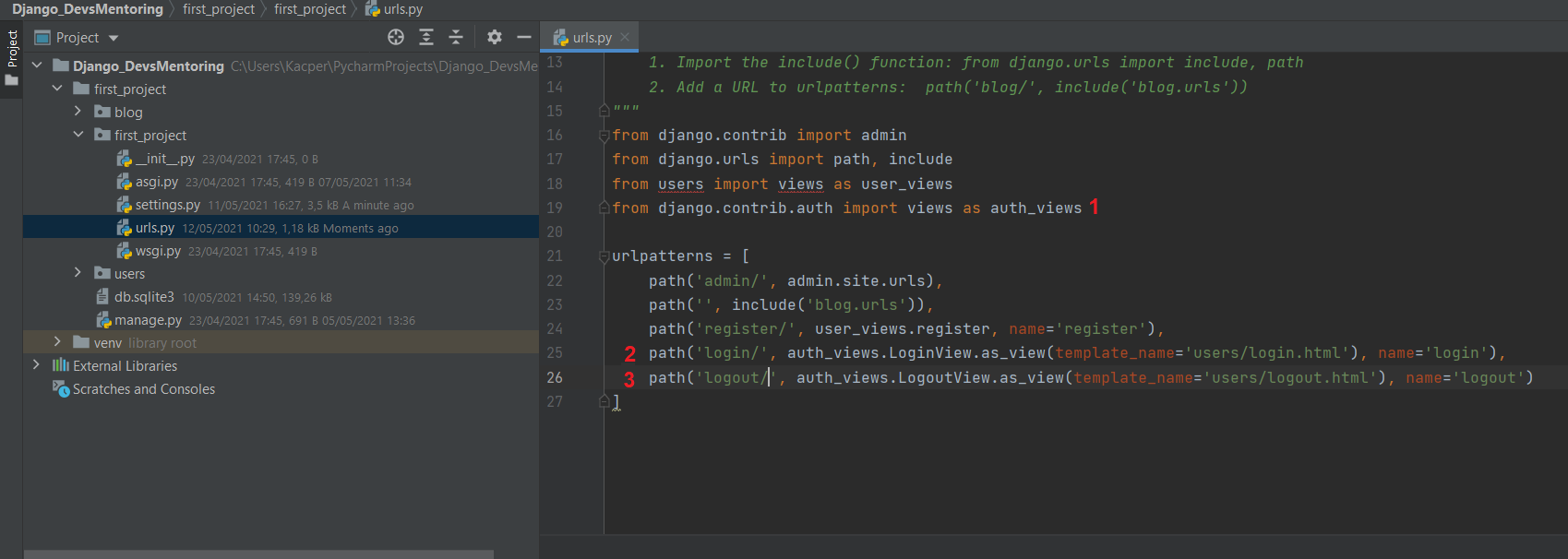
</div>

</div>

{% endblock %}

**Proces wylogowania**

Szablon logowania utworzony, nadszedł więc czas na dodanie widoku generowanego po wylogowaniu się użytkownika. Zgodnie z tym, co zarejestrowaliśmy w pliku first\_project/urls.py, dla użytkownik po przekierowaniu na endpoint /logout, generowany jest widok spod users/templates/users/logout.html.



W nowej wersji django ten widok nie działa więc robimy tak:

Urls.py:

from django.contrib import admin

from django.urls import path, include

from users import views as user\_views

from django.contrib.auth import views as auth\_views

urlpatterns = [

path("admin/", admin.site.urls),

path("", include('blog.urls', namespace='blog')),

path("register/", user\_views.register, name="register"),

path("login/", auth\_views.LoginView.as\_view(template\_name='users/login.html'), name="login"),

path("logout/", user\_views.logout\_view, name="logout"),

]

Views.py:

from django.shortcuts import render, redirect

from .forms import UserRegisterForm

from django.contrib import messages

from django.contrib.auth import logout

from django.urls import reverse

def register(request):

if request.method == 'POST':

form = UserRegisterForm(request.POST)

if form.is\_valid():

form.save()

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

messages.success(request, f"Dear {username}, you have been succesfully signed up!!")

return redirect('login')

else:

form = UserRegisterForm()

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

# Create your views here.

def logout\_view(request):

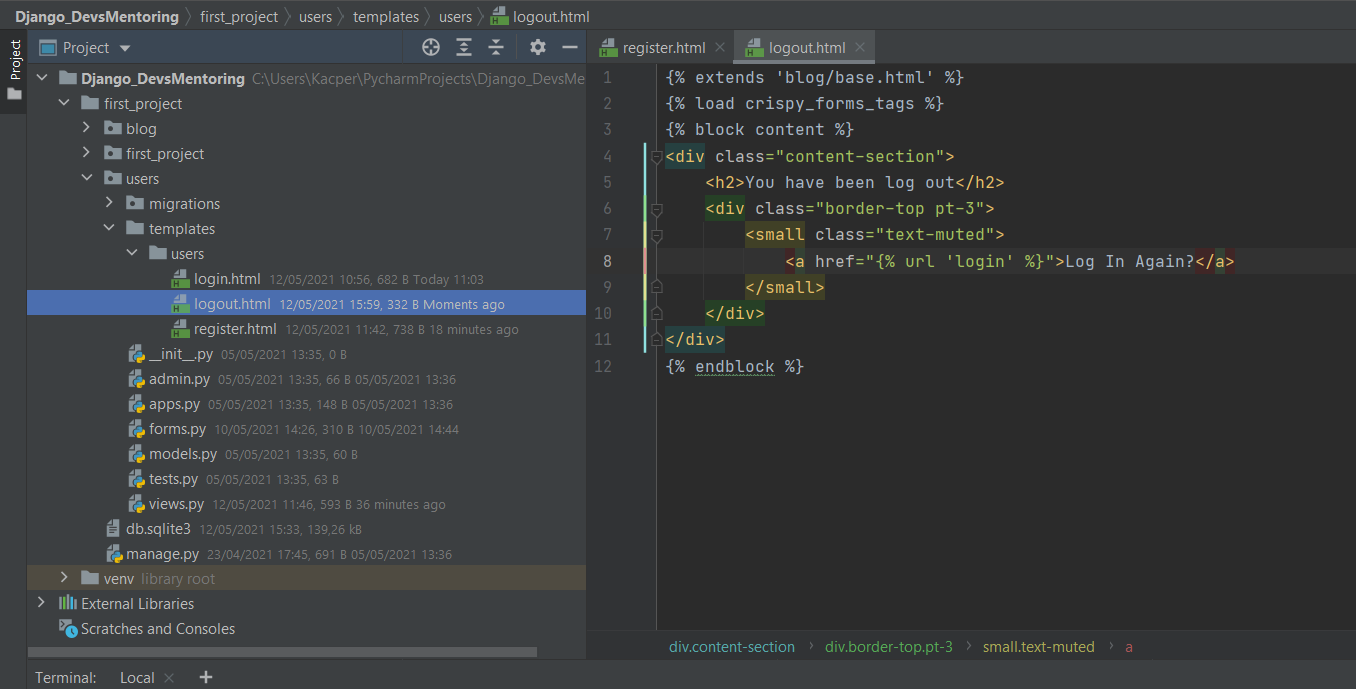
logout(request)

messages.info(request, "Logged out successfully!")

return render(request, 'users/logout.html')

Musimy zatem utworzyć nowy plik - users/templates/users/logout.html. Tak jak plik login.html, powstanie on na podstawie pliku register.html - z gruntownymi jednak zmianami.

W rezultacie nowo powstały plik będzie takiej postaci:



{% extends 'blog/base.html' %}

{% load crispy\_forms\_tags %}

{% block content %}

<div class="content-section">

<h2>You have been log out</h2>

<div class="border-top pt-3">

<small class="text-muted">

<a class="ml-2" href="{% url 'login' %}">SLog In again?</a>

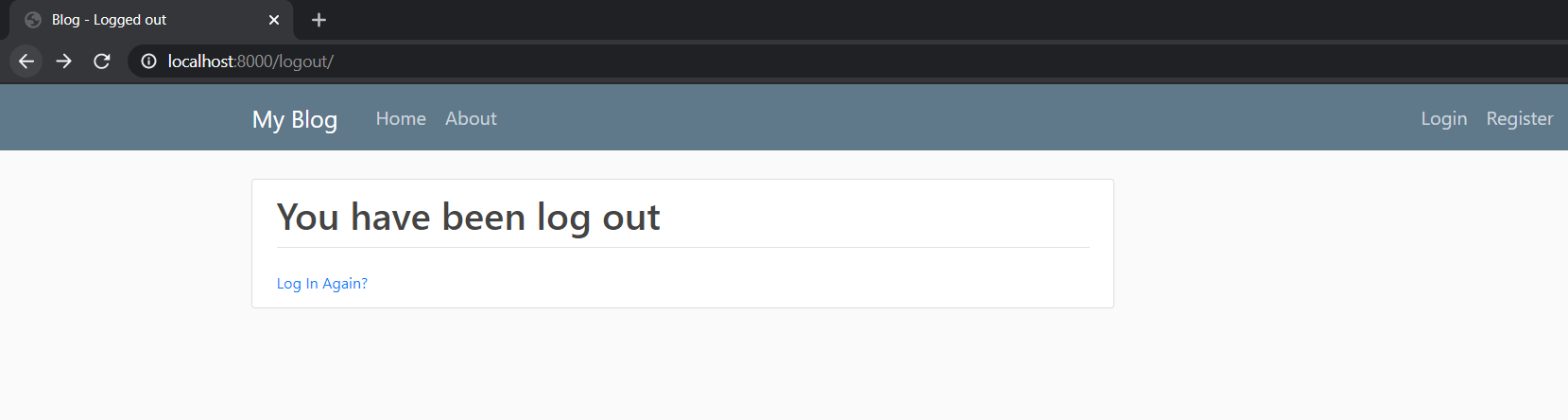
</small>

</div>

</div>

{% endblock %}

Po wpisaniu w przeglądarkę adresu z końcówką /logout, otrzymamy:

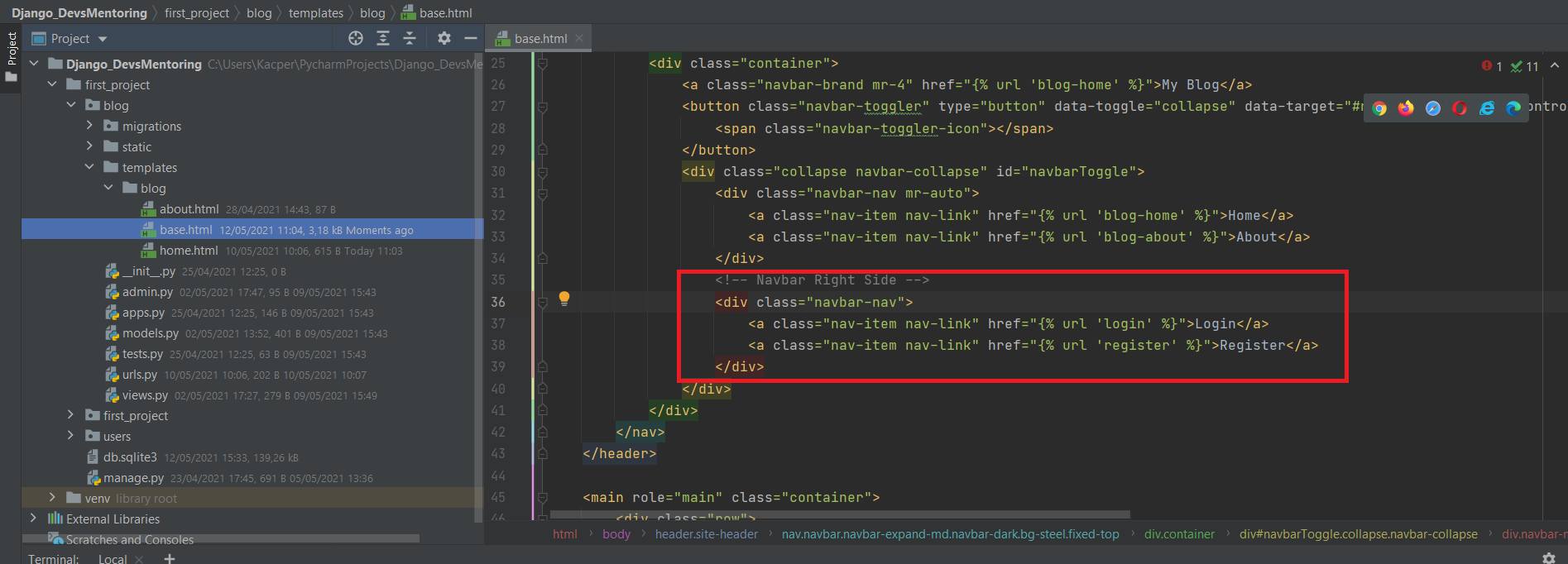


**Zmiana wyglądu paska nawigacji**

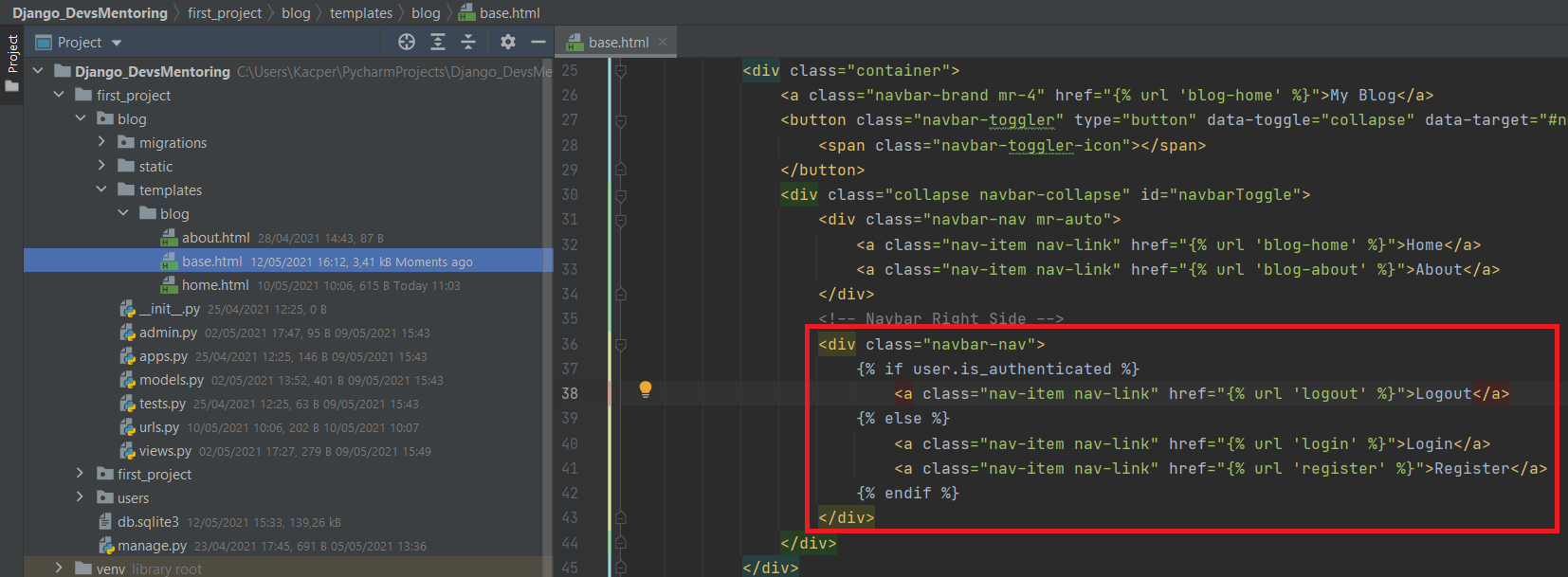
Po dodaniu funkcjonalności poprawnej rejestracji, właściwego logowania oraz wylogowywania użytkownika, przejdźmy do zakodowania zmian, które będą widoczne w zależności od tego, czy użytkownik jest zalogowany, czy też nie.

Otóż w momencie, gdy user zostanie autoryzowany, chcielibyśmy ukryć z paski nawigacji opcje Login oraz Register na rzecz odnośnika Logout.

Zmian tych dokonamy w bazowym pliku HTML - blog/templates/blog/base.html, w zaznaczonej na czerwono strefie.



Innymi słowy - za pomocą instrukcji warunkowych, będziemy chcieli sprawdzić w kodzie HTML, czy użytkownik jest zalogowany. Do tego celu pusłuży Nam zapis: user.is\_authenticated. Jeżeli tak (1), to do widocznego navbar-u, będziemy chcieli dodać pozycję Logout. Jeżeli użytkownik natomiast nie będzie zalogowany, to wyświetlać będziemy pozycje Login i Register w standardowej formie.



{% load static %}

<!DOCTYPE html>

<html>

<head>

<!-- Required meta tags -->

<meta charset="utf-8">

<meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

<!-- Bootstrap CSS -->

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css" integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T" crossorigin="anonymous">

<meta charset="UTF-8">

<link rel="stylesheet" type="text/css" href="{% static 'blog/base.css' %}" />

{% if title %}

<title>Blog - {{ title }}</title>

{% else %}

<title>Blog</title>

{% endif %}

</head>

<body>

<header class="site-header">

<nav class="navbar navbar-expand-md navbar-dark bg-steel fixed-top">

<div class="container">

<a class="navbar-brand mr-4" href="/">Django Blog</a>

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

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

</button>

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

<div class="navbar-nav mr-auto">

<a class="nav-item nav-link" href="{% url 'blog:blog\_home' %}">Home</a>

<a class="nav-item nav-link" href="{% url 'blog:blog\_about' %}">About</a>

</div>

<!-- Navbar Right Side -->

<div class="navbar-nav">

{% if user.is\_authenticated %}

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

{% else %}

<a class="nav-item nav-link" href="{% url 'login' %}">Login</a>

<a class="nav-item nav-link" href="{% url 'register' %}">Register</a>

{% endif %}

</div>

</div>

</div>

</nav>

</header>

<main role="main" class="container">

<div class="row">

<div class="col-md-8">

{% if messages %}

{% for message in messages %}

<div class = "alert alert-{{ message.tags }}">

{{ message }}

</div>

{% endfor %}

{% endif %}

{% block content %}{% endblock %}

</div>

</div>

</main>

<!-- Optional JavaScript -->

<!-- jQuery first, then Popper.js, then Bootstrap JS -->

<script src="https://code.jquery.com/jquery-3.3.1.slim.min.js" integrity="sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo" crossorigin="anonymous"></script>

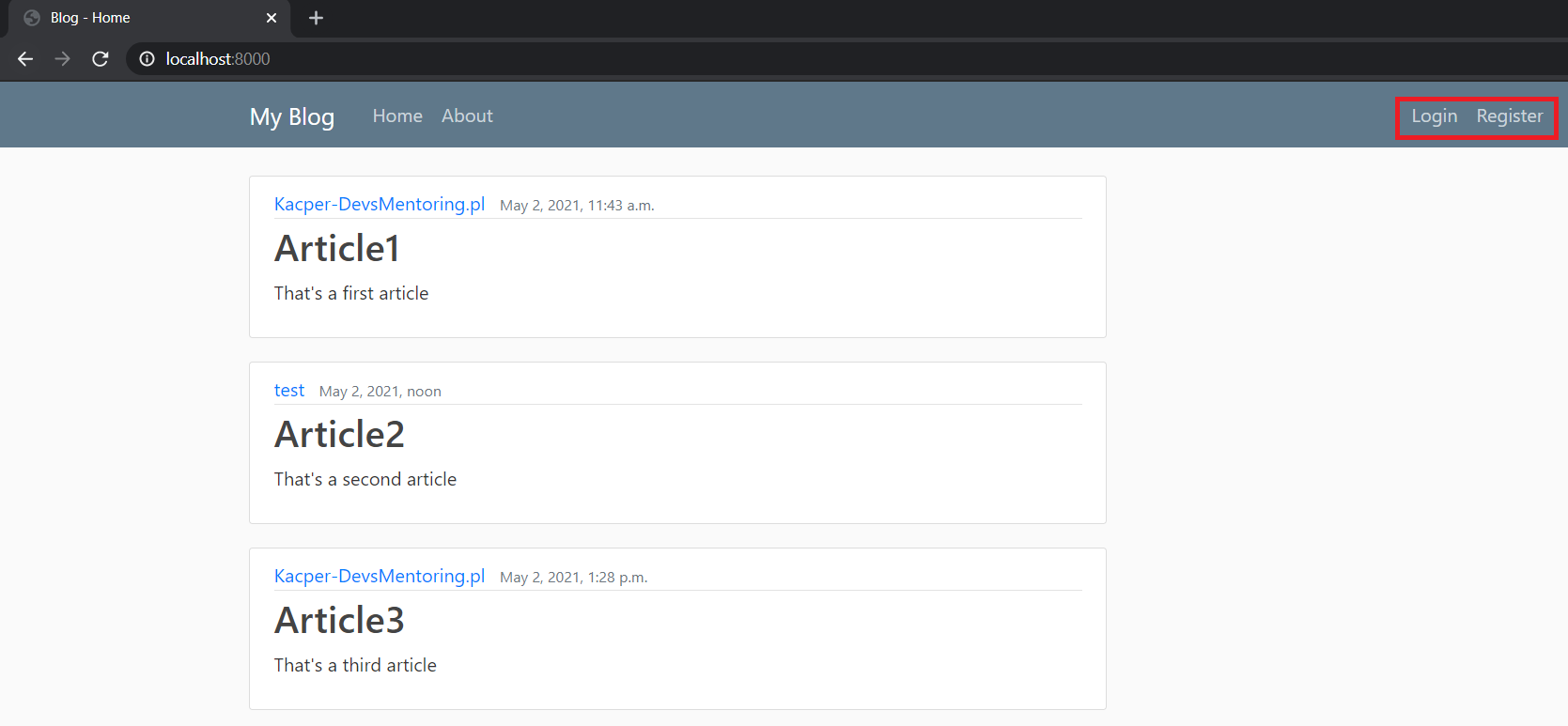
<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.js" integrity="sha384-UO2eT0CpHqdSJQ6hJty5KVphtPhzWj9WO1clHTMGa3JDZwrnQq4sF86dIHNDz0W1" crossorigin="anonymous"></script>

<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js" integrity="sha384-JjSmVgyd0p3pXB1rRibZUAYoIIy6OrQ6VrjIEaFf/nJGzIxFDsf4x0xIM+B07jRM" crossorigin="anonymous"></script>

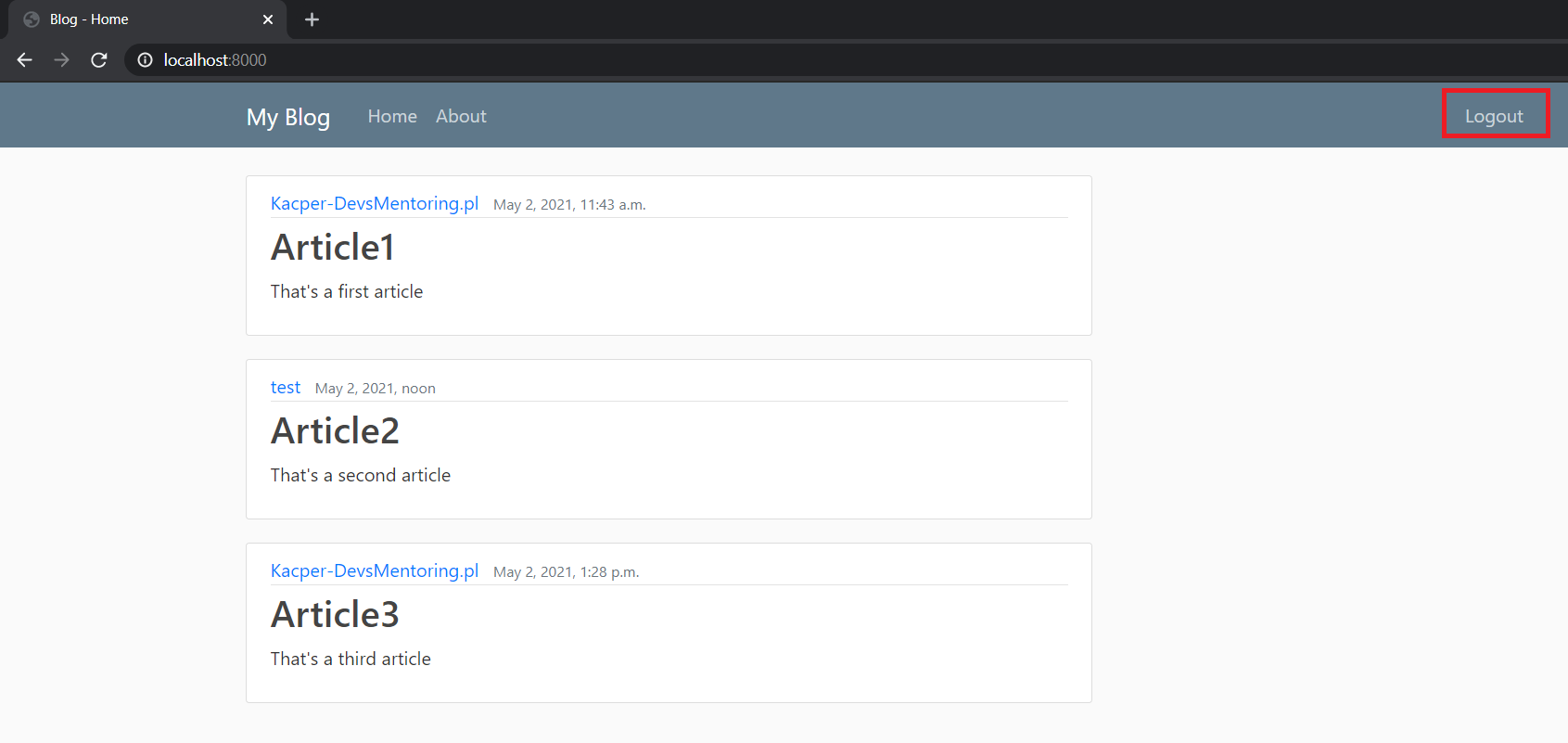
</body>

</html>

Elementy pasku nawigacji, gdy użytkownik nie jest zalogowany:



Elementy pasku nawigacji, gdy użytkownik jest zalogowany:

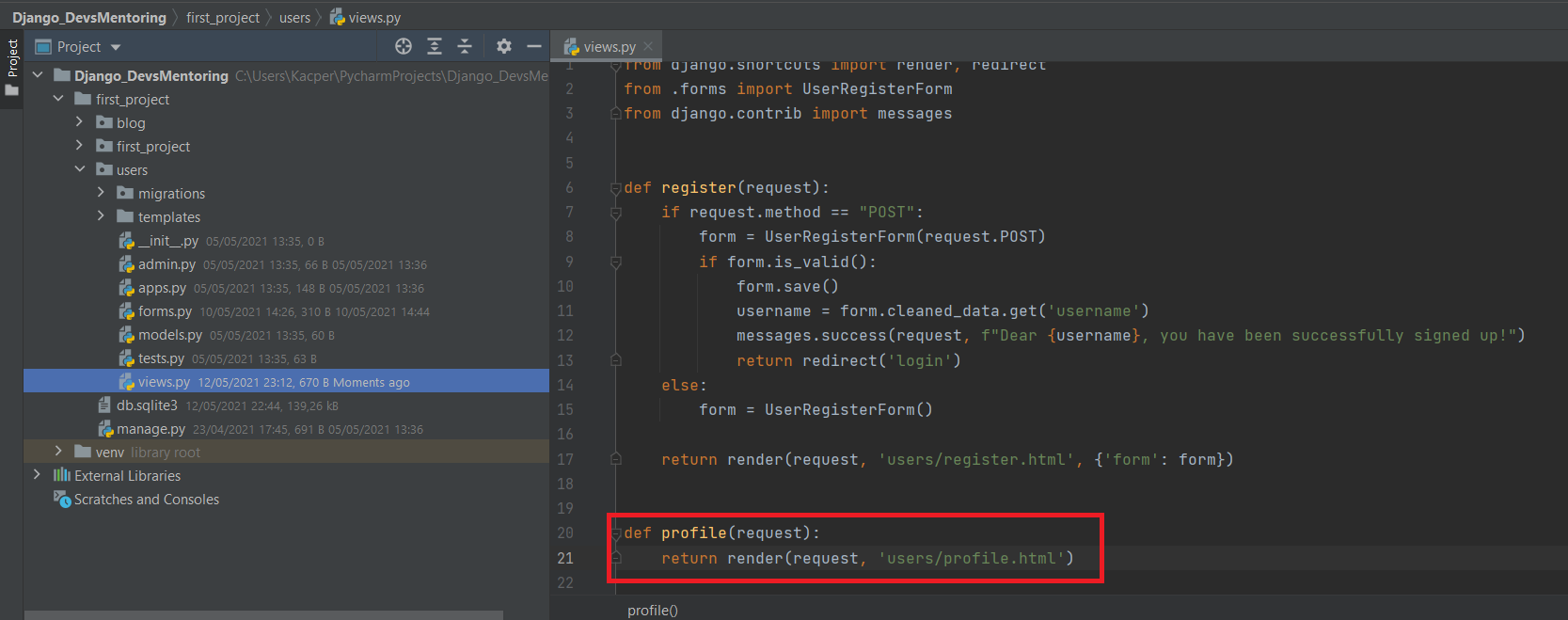


**Dodanie sekcji profile w aplikacji**

By dopełnić swego i zapewnić jeszcze bardziej kompleksowe działanie Naszej aplikacji - dodamy sekcję profile, do której będziemy mogli przejść po poprawnym zalogowaniu się na swoje konto. Dodatkowo, w sytuacji, gdy użytkownik nie jest jeszcze zalogowany, nie będziemy chcieli dopuścić go do wejścia na wspomniany adres /profile.

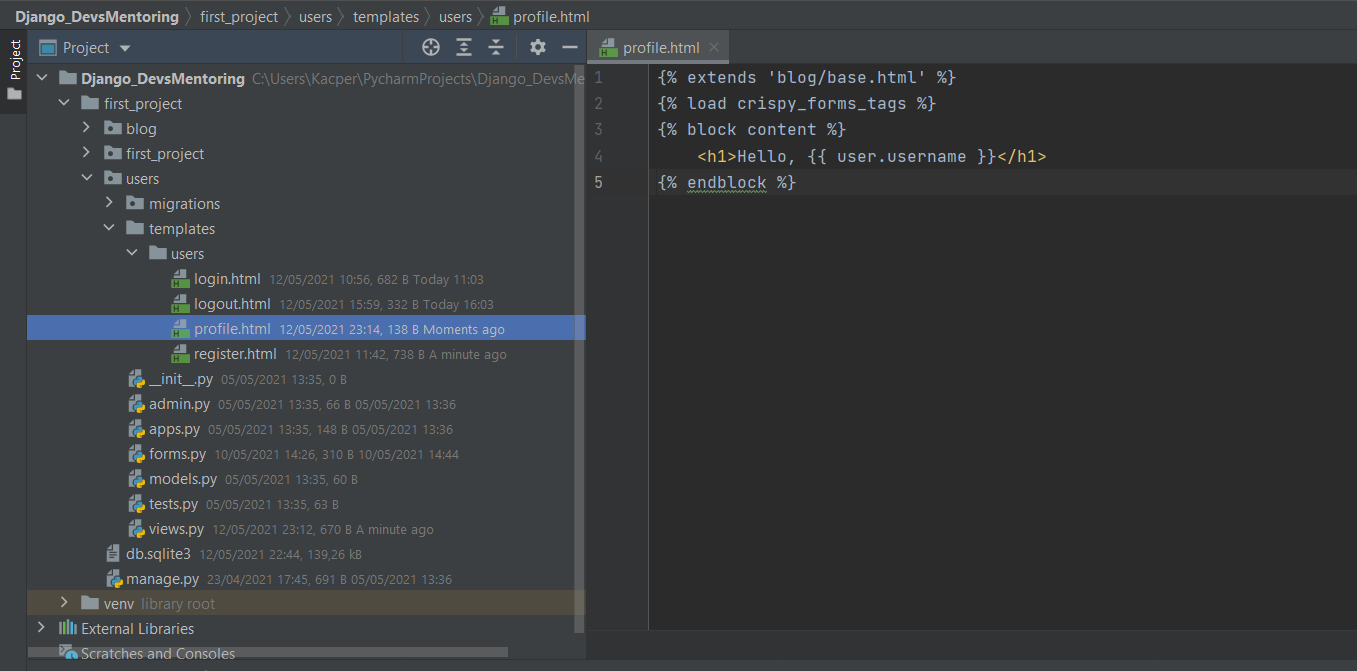
Funkcjonalność profilu będziemy realizowali w ramach aplikacji users. Na ten moment zapewnimy podstawową funkcjonalność realizowaną w ramach profilu - czyli wyświetlanie jedynie nazwy zalogowanego użytkownika.

Dodajmy do pliku users/views.py nowy widok - profile. Tak jak zostało wspomniane - na ten moment będzie on jedynie generował widok profile.html.



def profile(request):

return render(request, 'users/profile.html')



{% extends 'blog/base.html' %}

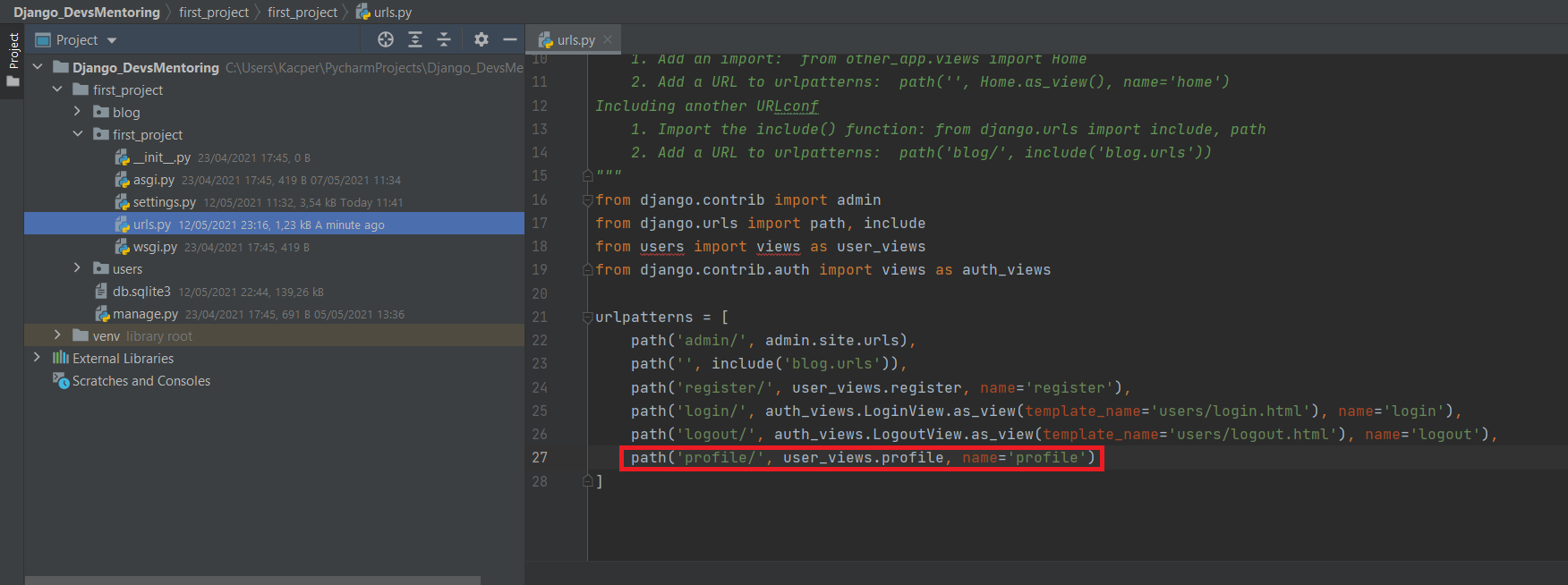
{% load crispy\_forms\_tags %}

{% block content %}

<h1>Hello, {{ user.username }}</h1>

{% endblock %}

Następnie, standardowo zarejestrujemy utworzony widok w pliku first\_project/urls.py…



from django.contrib import admin

from django.urls import path, include

from users import views as user\_views

from django.contrib.auth import views as auth\_views

urlpatterns = [

path("admin/", admin.site.urls),

path("", include('blog.urls', namespace='blog')),

path("register/", user\_views.register, name="register"),

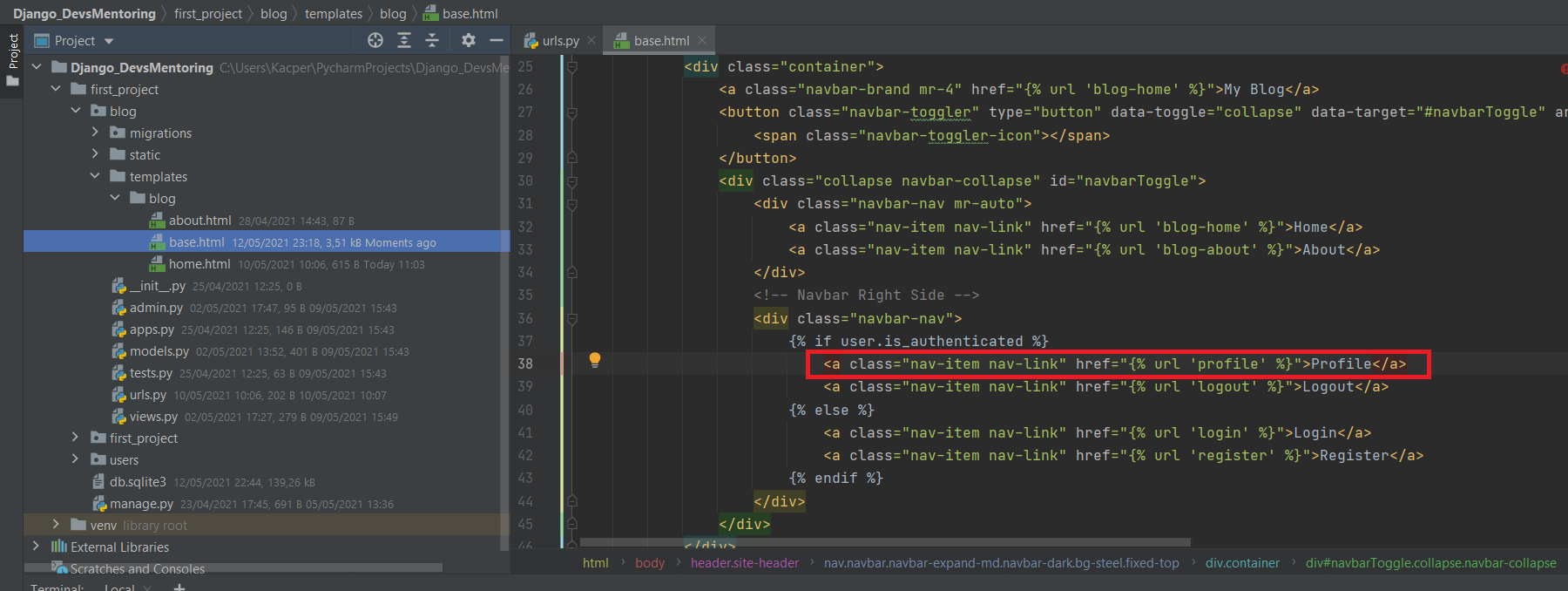
path("login/", auth\_views.LoginView.as\_view(template\_name='users/login.html'), name="login"),

path("logout/", user\_views.logout\_view, name="logout"),

path("profile/", user\_views.profile, name="profile"),

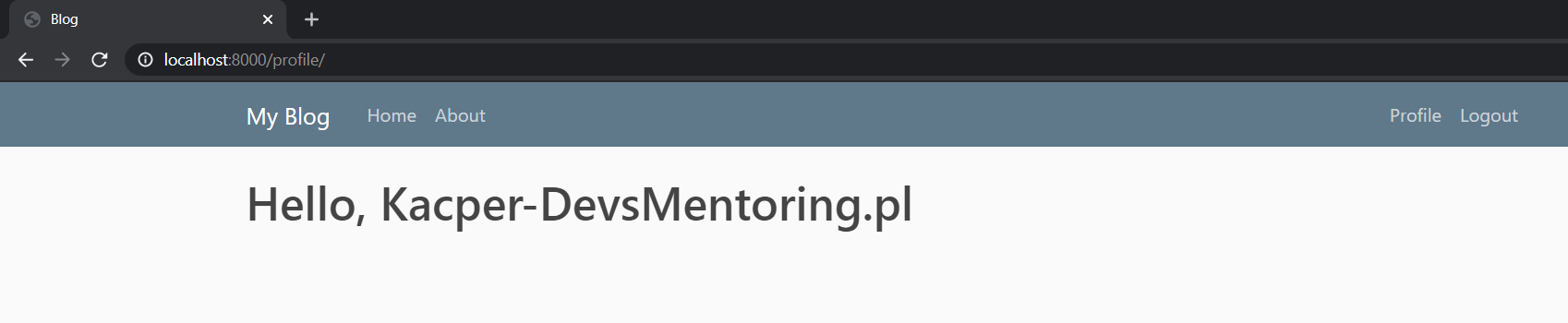
]

Oraz, na zwieńczenie, dodamy odnośnik do strony profilu na pasku nawigacyjnym. Oczywiście zrobimy tak, aby opcja ta wyświetlała się tylko wtedy, gdy użytkownik będzie zalogowany. Na czerwono zaznaczyłem zmiany, jakie były konieczne do przeprowadzenia w pliku blog/templates/blog/base.html.

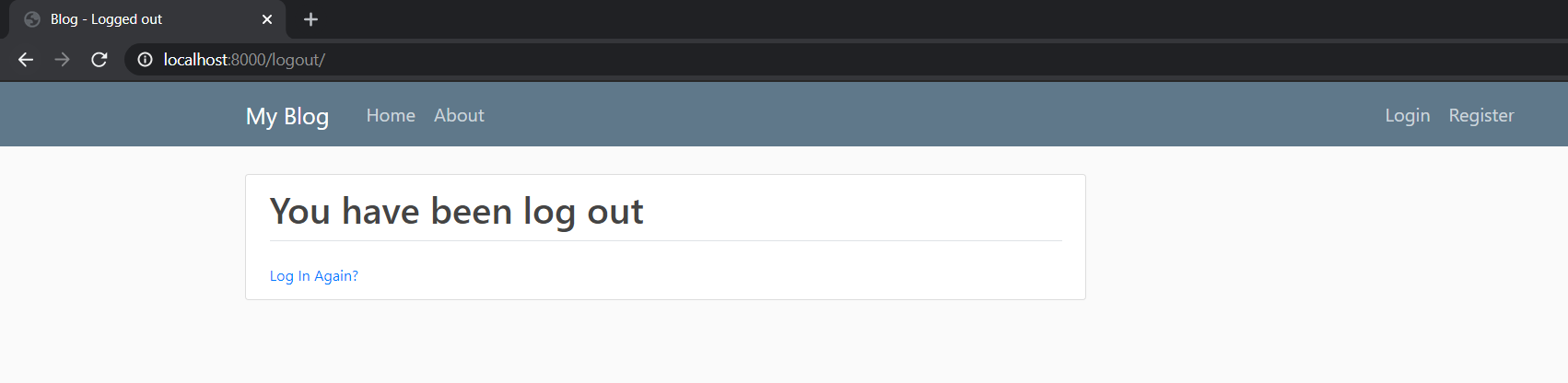


<a class="nav-item nav-link" href="{% url 'profile' %}">Profile</a>

Po uruchomieniu aplikacji, okazuje się, że wszystko działa zgodnie z założeniami. Przykładowo - Nasza podstrona /profile wygląda następująco:



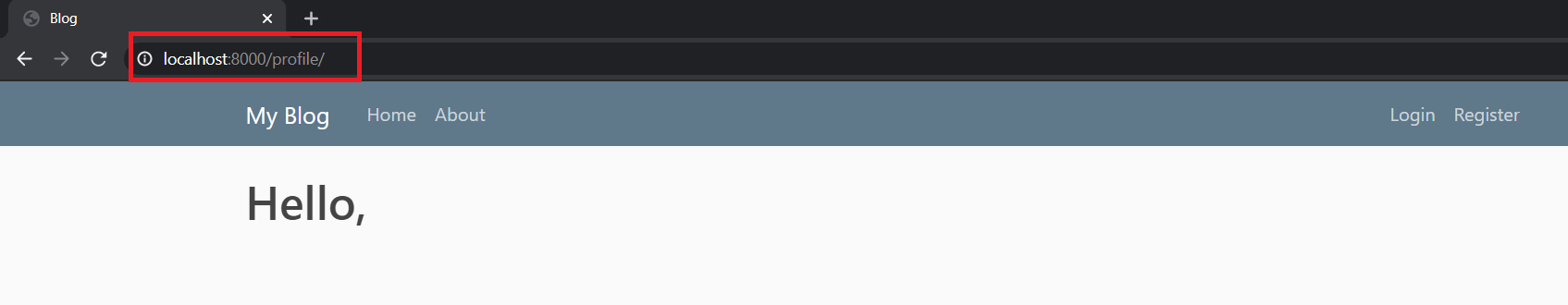
Natomiast, gdy się wylogujemy, to widzimy:



Wszystko zgodnie z oczekiwaniami! Jednak wkrada się tutaj pewien mankament…

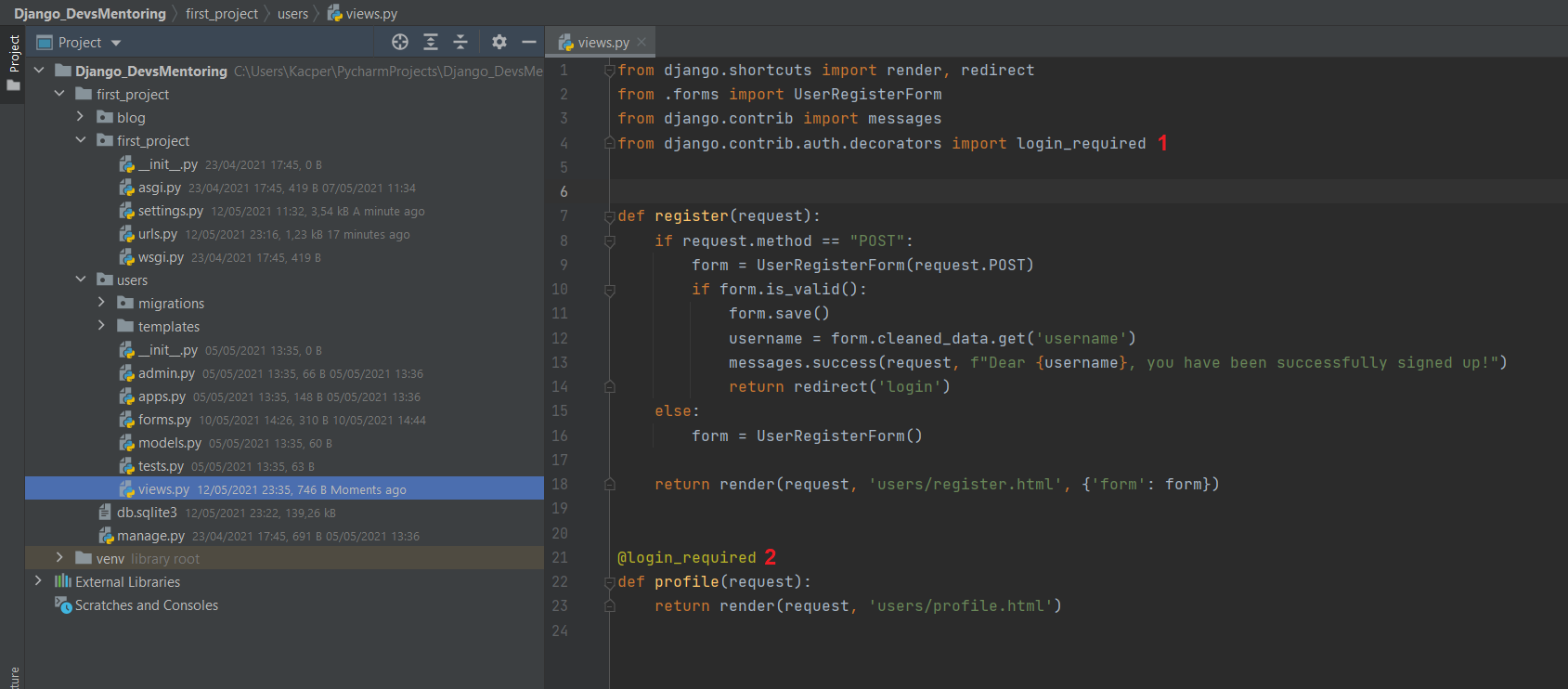
Otóż, teoretycznie nie powinniśmy mieć możliwości przejścia do podstrony /profile, będąc niezalogowanymi. Faktycznie, na pasku nawigacji nie pojawia się odnośnik Profile, ale co się wydarzy, gdy użytkownik ręcznie wpisze adres przekierowania do profilu, będąc niezalogowanym?

Po wylogowaniu się z mojego konta, wpisałem w pasek adres z końcówką /profile/ i zostałem przeniesiony do następującej podstrony.



Przyznaj, że niezbyt prawidłowe działanie Naszej aplikacji… O wiele lepszym rozwiązaniem by było, jakby aplikacja przekierowywała Nas do panelu logowania w momencie, gdy próbujemy niezalogowani przejść do profilu.

Jak się okazuje istnieje na to bardzo proste rozwiązanie. Wystarczy bowiem zaimportować odpowiedni dekorator login\_required i przyozdobić nim utworzoną w pliku users/views.py funkcję profile oraz w globalnych ustawieniach aplikacji (first\_project/settings.py) dodać odpowiednią opcję - LOGIN\_URL.



from django.shortcuts import render, redirect

from .forms import UserRegisterForm

from django.contrib import messages

from django.contrib.auth import logout

from django.urls import reverse

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

def register(request):

if request.method == 'POST':

form = UserRegisterForm(request.POST)

if form.is\_valid():

form.save()

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

messages.success(request, f"Dear {username}, you have been succesfully signed up!!")

return redirect('login')

else:

form = UserRegisterForm()

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

# Create your views here.

def logout\_view(request):

logout(request)

messages.info(request, "Logged out successfully!")

return render(request, 'users/logout.html')

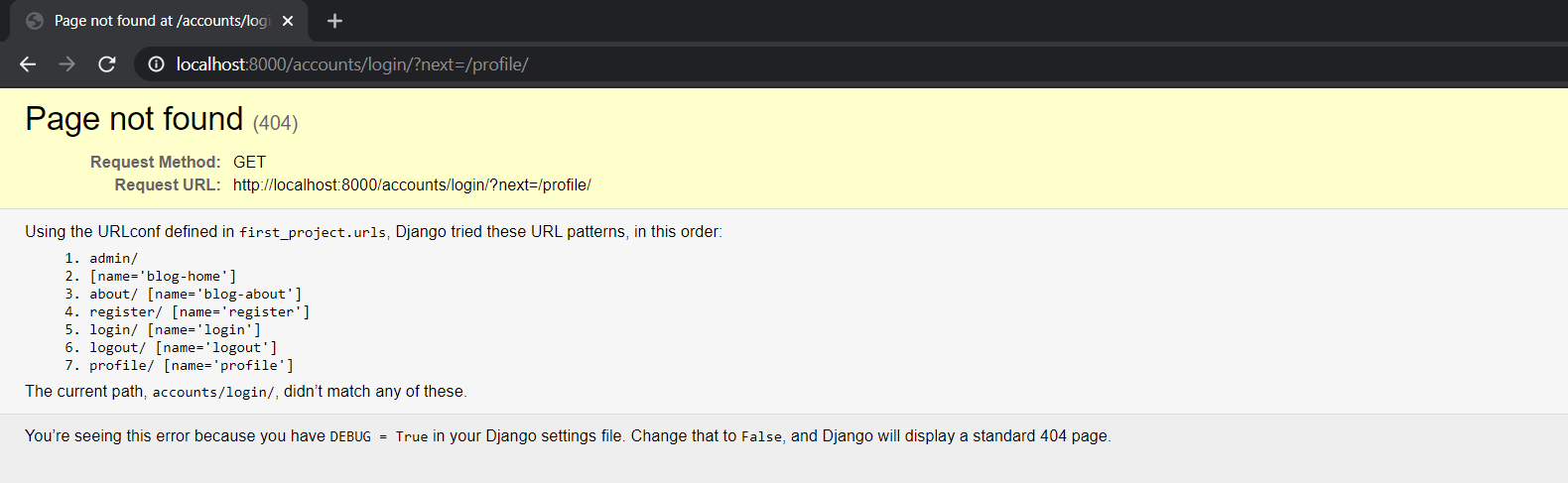
@login\_required

def profile(request):

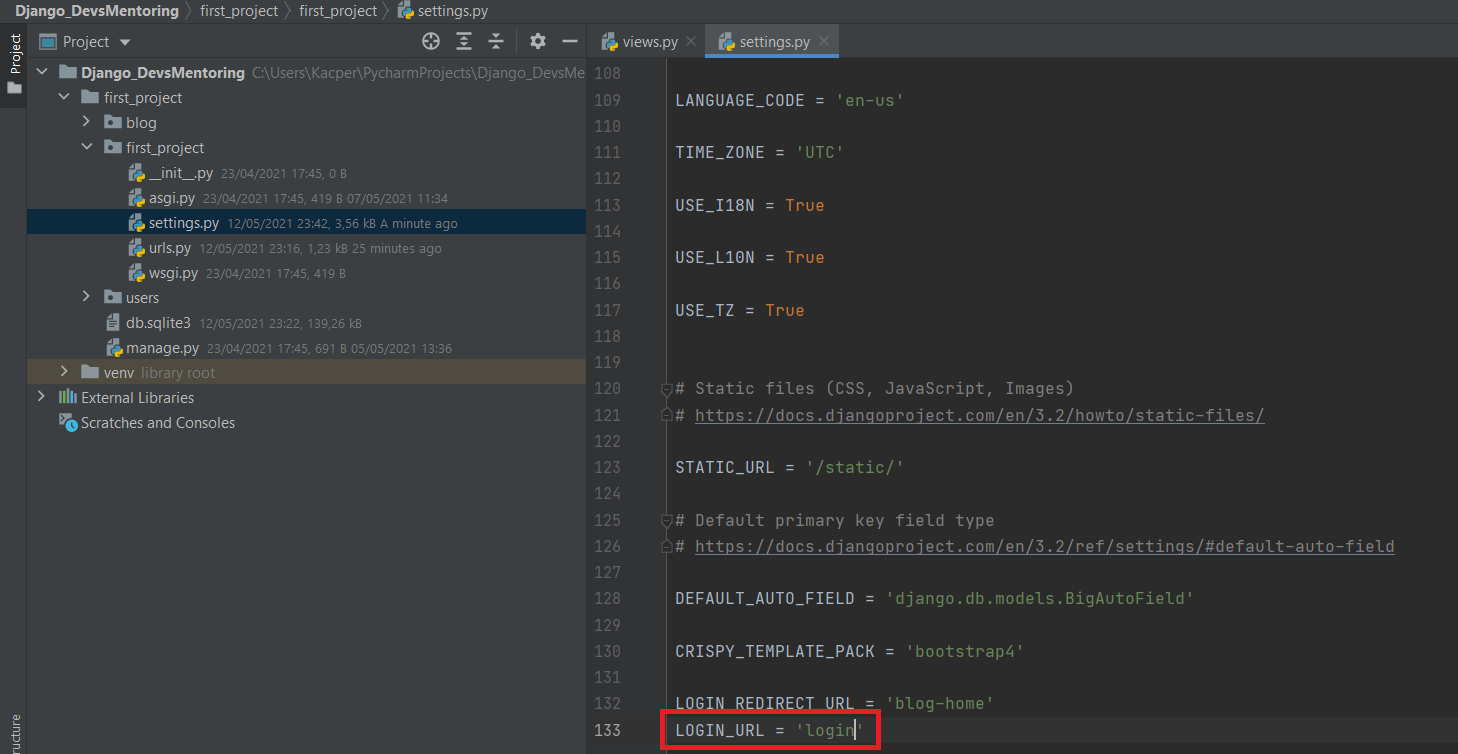
return render(request, 'users/profile.html')

W powyższym przykładzie, aby ozdobić funkcję profile dekoratorem login\_required (2), musieliśmy zaimportować ten dekoratorem (1) polecenie from django.contrib.auth.decorators import login\_required. Dzięki takiej zmianie, dostęp do tego widoku będzie niemożliwy, jeżeli użytkownik się nie zaloguje.

W celu sprawdzenia tego w praktyce - niezalogowany na jakiekolwiek konto w Naszej aplikacji, wpisałem do paska adres zakończony /profile/. Oto, co otrzymałem:



Faktycznie, wygląda na to, że jesteśmy teraz zabezpieczeni przed nieautoryzowanym wejściem na endpoint profile/, jednak aplikacja się wysypuje. A to dlatego, że po zablokowaniu dostępu do omawianej sekcji dla niezalogowanych użytkowników, w momencie próby przejścia do /profile/, przekierowuje ich domyślnie pod nieistniejący adres /accounts/login. Tak domyślne zachowanie możemy zmienić wprowadzając właśnie wcześniej wspomnianą opcję LOGIN\_URL do pliku first\_project/settings.py. Wartość, jaką przypiszemy do tej opcji to będzie nazwa zarejestrowanego widoku, do którego ma zostać przekierowany użytkownik po próbie dostania się do /profile/, nie będąc zalogowanym.



LOGIN\_URL = 'login’

Voilà! Od teraz, każda próba przejścia pod adres /profile/ bez uprzedniego zalogowania, kończyć się będzie przekierowaniem do panelu logowania.

