**Django - szkolenie 2**

**Funkcjonalność projektu, routes. Szablony**.

**Jak budować efektywny projekt?**

Muszę się przyznać, że dotąd nie wspomniałem o jeszcze jednej dość kluczowej rzeczy związanej z budową projektów. Otóż każdy efektywny projekt Django powinien składać się z odpowiednich submodułów, nazywanych aplikacjami. Aplikacje to niezależne od siebie byty realizujące pewne funkcjonalności, które składają się na całą integralną aplikację Django.

Prostym przykładem może być projekt sklepu. Aby zapewnić enkapsulację funkcjonalności, moglibyśmy stworzyć w nim choćby takie aplikacje jak: koszyk, produkty czy profil użytkownika. Każdy z tych modułów realizowałby pewne charakterystyczne funkcjonalności, a po ich połączeniu ze sobą, otrzymalibyśmy kompletny projekt.

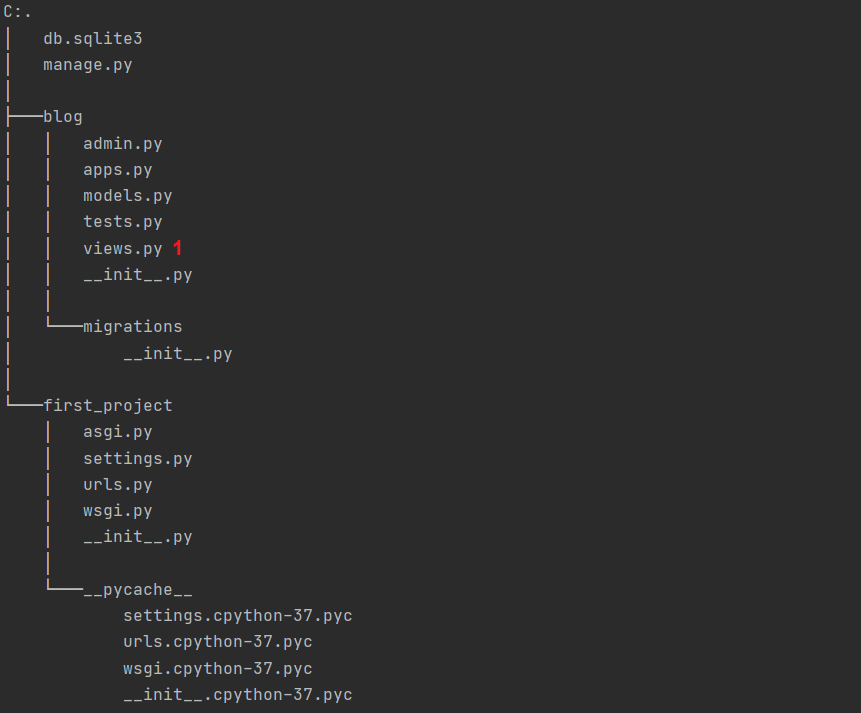
**Tworzenie aplikacji w projekcie**

Utwórzmy pierwszą aplikację projektową. Z racji, iż docelowo będziemy chcieli stworzyć stronę działającą na wzór bloga, pierwszym modułem będzie aplikacja o dość trywialnej nazwie - blog. Odpowiedzialna ona będzie za zarządzanie publikowanymi artykułami. Takie artykuły będzie mógł tworzyć każdy z zarejestrowanych użytkowników (tworzonych w ramach jeszcze innej aplikacji - którą utworzymy niebawem - users).

Aby ją utworzyć, należy w konsoli wywołać polecenie: python manage.py startapp blog



Po wywołaniu tej komendy, powinien utworzyć się nowy katalog, położony równolegle do katalogu projektowego first\_project, o nazwie blog.



*Struktura utworzonego projektu*

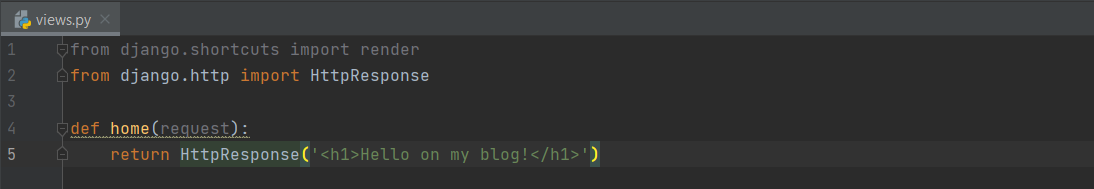
**Dodawanie endpointów do aplikacji**

Załóżmy, że chcemy dodać do Naszej aplikacji dwa endpointy. Będą to: /home oraz /about. Jak zaraz zobaczysz, dodanie takich ścieżek do projektu nie ogranicza się do zmodyfikowania tylko jednego pliku.

Ale po kolei… zacznijmy od zmodyfikowania pliku views.py znajdującego się w katalogu blog.

1. Plik blog/views.py

To plik, w którym przechowywać będziemy “zachowanie” określonych widoków projektu, czyli po prostu pewnych endpointów aplikacji. Pojedynczy widok będziemy tworzyli w formie funkcji z odpowiednio zaprogramowanym ciałem. Utwórzmy pierwszy endpoint - home. Póki co będzie on realizował proste założenia, wyświetlał będzie napis “Hello on my blog!”.



*Plik blog/views.py*

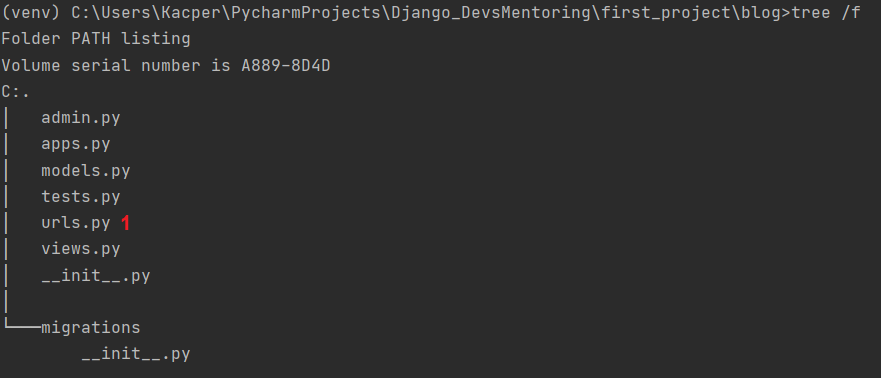
from django.shortcuts import render

from django.http import HttpResponse

def home(request):

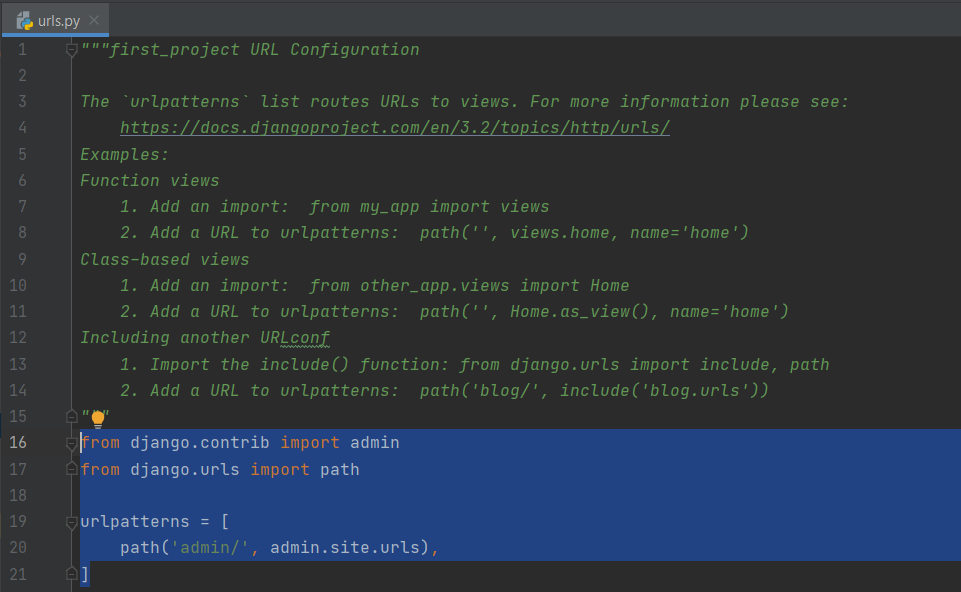
return HttpResponse('<h1>Hello</h1>')

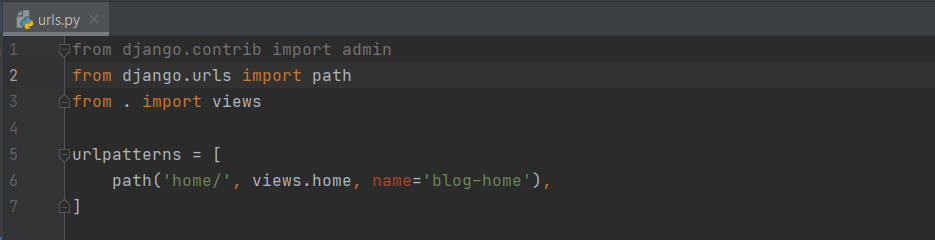
Następnym krokiem, będzie powiązanie stworzonego widoku (home) z konkretnym endpointem (/home) w Naszej aplikacji. W tym celu konieczne będzie utworzeniu nowego pliku - urls.py w katalogu blog.



Tak jak wspomniałem, będzie to plik, w którym zdefiniujemy, który widok aplikacji blog ma zostać wygenerowany po wpisaniu określonego adresu URL w przeglądarce. Plik ten będzie bardzo podobny do obecnego już pliku urls.py, ale znajdującego się w katalogu first\_project.

Możemy niejako skopiować z niego część kodu, pamiętając o właściwych modyfikacjach.





*Nowo utworzony plik blog/urls.py*

from django.contrib import admin

from django.urls import path

from . import views

urlpatterns = [

path("home/", views.home, name='blog-home'),

]

Zwróć również uwagę na to, iż do przekopiowanego kodu z pliku first\_project/urls.py, dodałem linię: from . import views oraz zmodyfikowałem element path(...) znajdujący się wewnątrz listy urlpatterns.

Na uwagę zasługuje również pierwszy i jedyny element listy urlpatterns. Znajduje się w nim ścieżka do widoku views.home reprezentowana przez endpoint /home oraz nazwana jako “blog-home”. Z nazwy tej będziemy korzystali na dalszym etapie rozwoju programu. Na ten moment zdaj sobie sprawę, iż właściwe nazywanie pewnych endpointów, pozwoli Nam przyspieszyć dalszy proces rozwoju projektu. Dzięki temu nie będziemy musieli podawać całego endpointa do danego widoku - wystarczy jedynie przywołanie jego nazwy (w naszym przypadku blog-home).

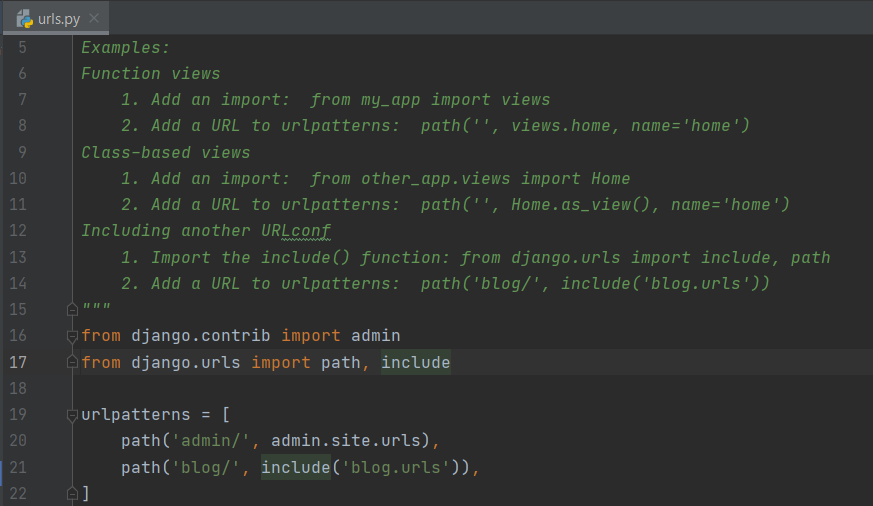
Ostatnim krokiem, aby móc korzystać z endpointu /home jest dodanie odpowiedniej ścieżki w pliku first\_project/urls.py.

Zapytasz, po co rejestrować ten sam widok w dwóch różnych plikach urls.py?

Otóż w pliku first\_project/urls.py nie do końca będziemy rejestrowali widok, ale dostęp do całej aplikacji blog. Zrobimy to jednorazowo, tak więc nie będzie potrzeby dalszej modyfikacji tego pliku przy dodawaniu nowych endpointów do aplikacji blog.

Tak więc w zasobie first\_project/urls.py dodajemy ścieżkę do całej nowo utworzonej aplikacji, jaką jest, np. blog. Dzięki takiemu rozwiązaniu, silnik WSGI będzie wiedział, pod jakim adresem ma dostęp do wszystkich zasobów tej aplikacji.

Spójrz, jak będzie wyglądał omawiany plik po modyfikacji:



*Plik first\_project/urls.py*

from django.contrib import admin

from django.urls import path, include

urlpatterns = [

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

path("blog/", include('blog.urls')),

]

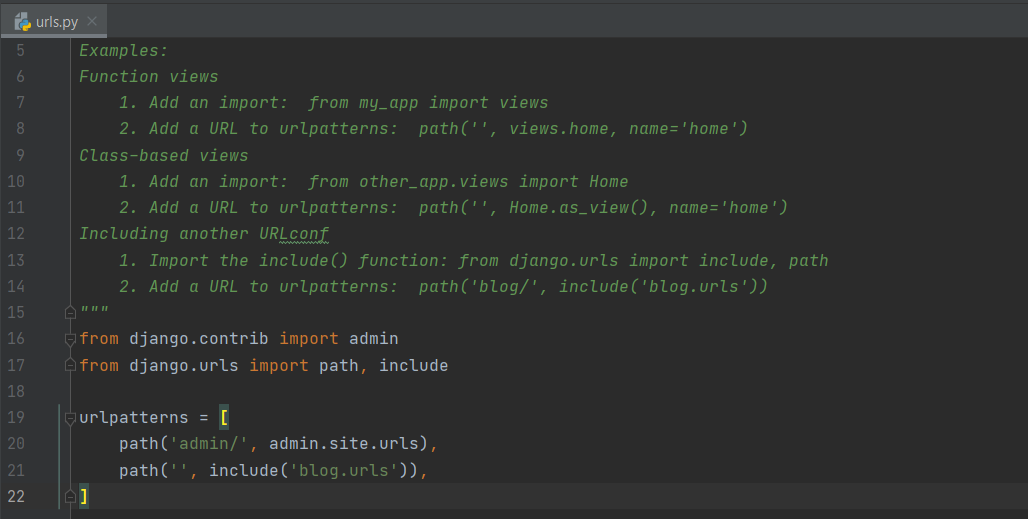
Do pliku dodaliśmy ścieżkę dostępową do **całej aplikacji** - path('blog/', include('blog.urls')). Konieczne było również zaimportowanie funkcji include(), która umożliwia kojarzenie ścieżki z określonym zasobem (w Naszym przypadku plikiem blog/urls.py).

Tak więc od tej pory, chcąc dostać się do określonego widoku z aplikacji blog, konieczny będzie zapis, np. localhost:8000/blog/home.

Zwróć uwagę na prefix /blog, który powstał po dodaniu takiej ścieżki w pliku first\_project/urls.py.

To właśnie do tego służy dodawanie pewnych ścieżek w pliku first\_project/urls.py. Są to ścieżki, które zapewniają Nam dostęp do całej aplikacji (/blog), a dopiero później do konkretnego endpointu (/blog/home).

Na ten moment, nie potrzebujemy aż tak szczegółowo wydzielać ścieżki dostępowej do aplikacji bloga, dlatego plik first\_project/urls.py sprowadzimy do postaci:



from django.contrib import admin

from django.urls import path, include

urlpatterns = [

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

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

]

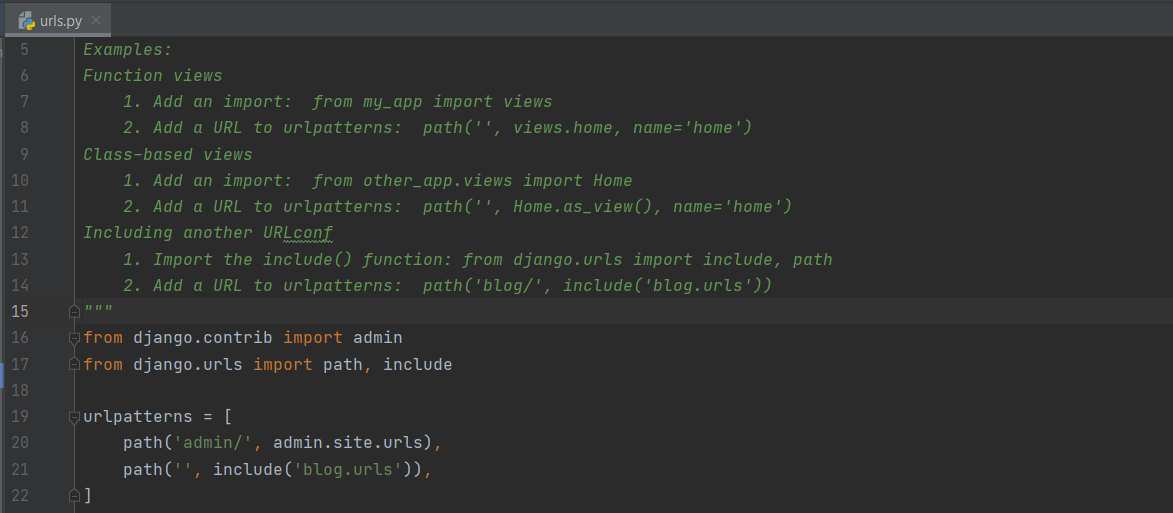
Dostęp do endpointu, np. home spod bloga będzie zatem możliwy dzięki wpisaniu: localhost:8000/home (nie jest już konieczne umieszczanie URL do całej aplikacji w adresie URL).

**Modyfikacja endpointów aplikacji blog**

Super, wiemy już, na jakich zasadach funkcjonuje dodawanie projektowych ścieżek URL. Adresy istniejącego projektu zmodyfikujemy zatem do następującej postaci:

* Adres localhost:8000/ prowadzić ma do strony home bloga
* Adres localhost:8000/about prowadzić ma do strony about aplikacji blogowej

Tak więc pliki first\_project/urls.py oraz blog/urls.py zmienimy do następującej postaci:



*Plik first\_project/urls.py*

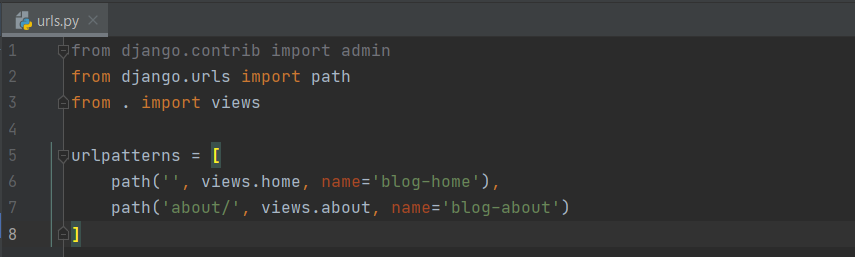
*from django.contrib import admin*

*from django.urls import path, include*

*urlpatterns = [*

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

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

*]*

Plik blog/urls.py

from django.contrib import admin

from django.urls import path

from . import views

app\_name = 'blog'

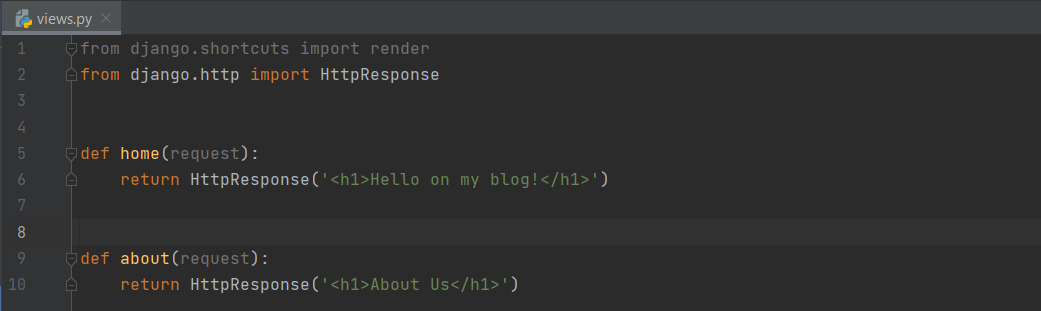
urlpatterns = [

path("", views.home, name='blog\_home'),

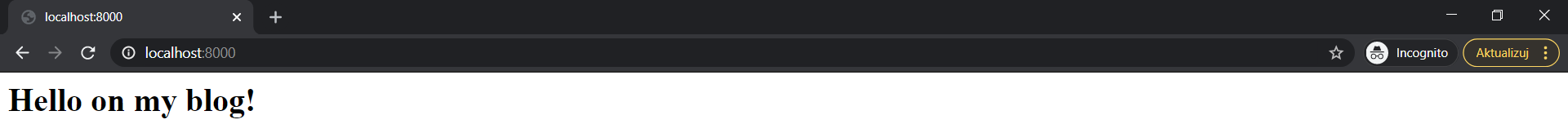
path("about/", views.about, name='blog\_about'),

]

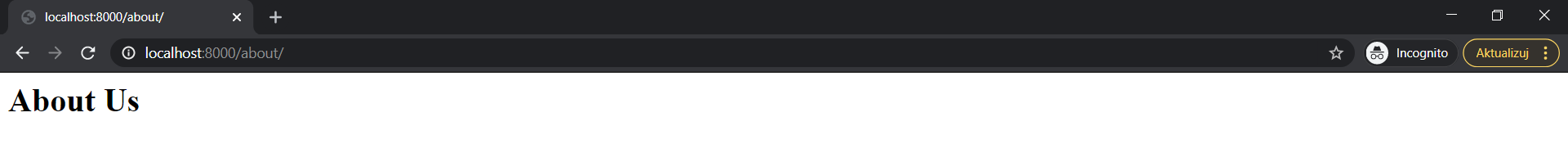
Oraz do pliku blog/views.py i znajdującej się w nim funkcji home, dodamy nowy endpoint - about:



W rezultacie otrzymamy następującą funkcjonalność (zwróć uwagę na wpisywane adresy URL w przeglądarce):



*Adres localhost:8000/*



*Adres localhost:8000/about*

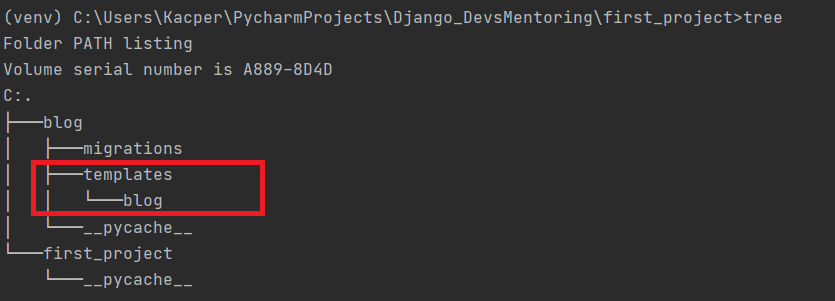
**Ulepszanie front-endu aplikacji**

Wyśmienicie! Stworzyliśmy pierwszą prostą aplikację z dwiema podstronami. Chcielibyśmy ją teraz na pewno ulepszyć i dodać na przykład generowanie widoku przez pliki .html. Pójdziemy również krok dalej i wprowadzimy dziedziczenie szablonów, a także zadbamy o dobrze wyglądające ułożenie artykułów bloga na stronie.

**Szablony**

Pliki z kodem źródłowym strony będziemy porządkowali w postaci hierarchii szablonów. Nie będę rozdrabniał się nad tym, czym jest dziedziczenie szablonów, bo wiedzę tę pokryliśmy w szkoleniu z Flaska (szybki recap - dziedziczenie umożliwiało tworzenie Nam jednego głównego pliku html, po którym dziedziczyły potomki i w rezultacie zawierały wspólne elementy - takie jak, np. ten sam navbar, tło strony itp.)

W pierwszej kolejności utwórzmy nowy katalog templates w folderze aplikacji - blogs. Następnie w katalogu tym będziemy musieli utworzyć jeszcze jeden folder - o takiej samej nazwie co katalog Naszej aplikacji. W rezultacie więc struktura Naszego projektu powinna wyglądać następująco:



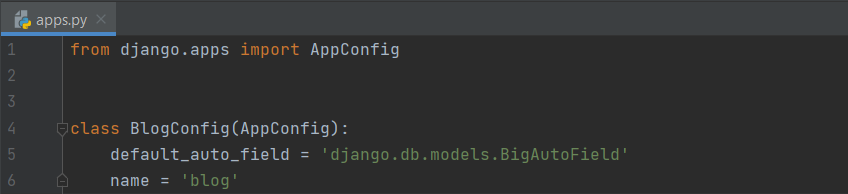
Kolorem czerwonym zaznaczyłem nowo dodane katalogi. To właśnie w katalogu blog/templates/blog dodawać będziemy Nasze pliki .html.

Tak, wiem, wiem. Też uważam, że dodawanie tylu nowych katalogów, aby móc dodać szablony do Naszego projektu jest nieco pracochłonne, ale nie mamy na to za dużego wpływu. Silnik Django szuka odpowiednich plików właśnie według takiego wzorca katalogów.

**Dokonfigurowanie aplikacji**

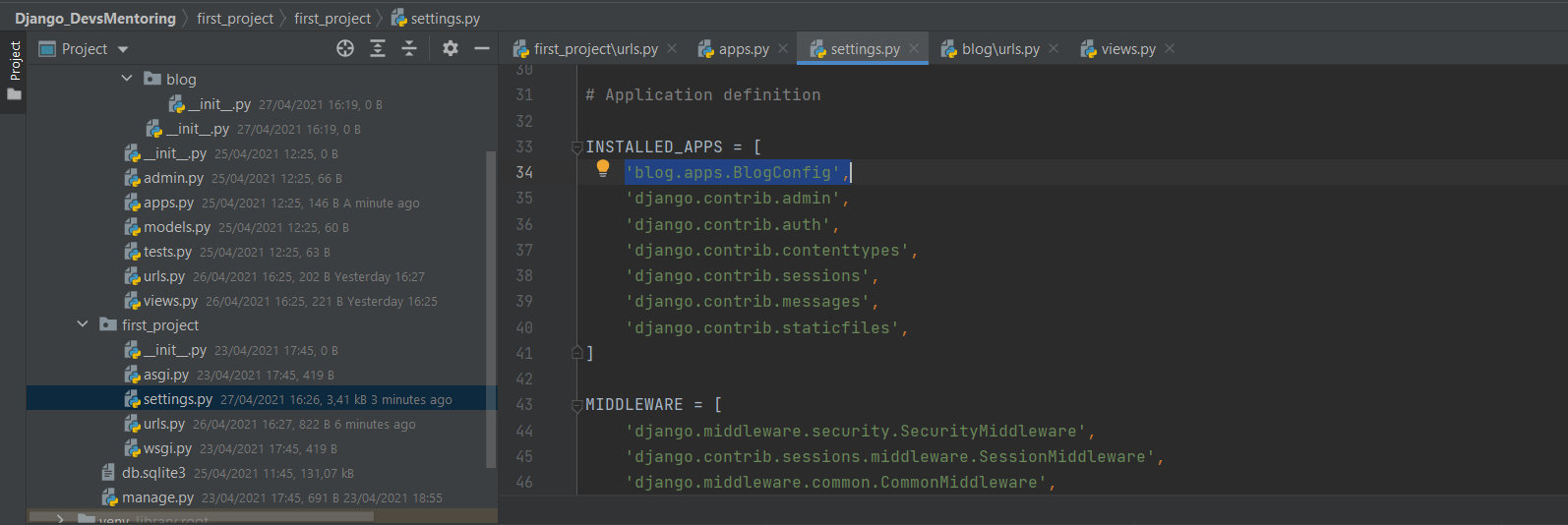
Zanim przystąpimy do tworzenia kodu HTML w utworzonych katalogach, musimy jeszcze zapewnić pełną konfigurację aplikacji blog.

Zauważ, iż w pliku blog/apps.py znajduje się następująca zawartość:

*Plik blog/apps.py*

Aby zapewnić prawidłowe funkcjonowanie aplikacji (czyli choćby generowanie szablonów), musimy dodać referencję do wyżej przedstawionej klasy konfiguracyjnej BlogConfig w pliku first\_project/settings.py.

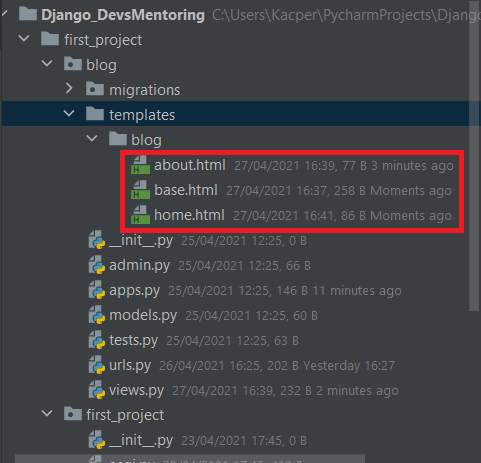
Referencję tę dodamy jako ścieżkę zapisaną w postaci stringa do listy INSTALLED\_APS.



"blog.apps.BlogConfig",

**Dodanie plików html**

Po utworzeniu nowych katalogów, dodaniu odpowiedniej referencji w konfiguracji aplikacji, możemy w końcu przejść do tworzenia szablonów. Zacznijmy od utworzenia klasy bazowej dla podstron / oraz /about w Naszej aplikacji.





*Plik base.html*

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

{%if title %}

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

{% else %}

<title>Blog</title>

{% endif %}

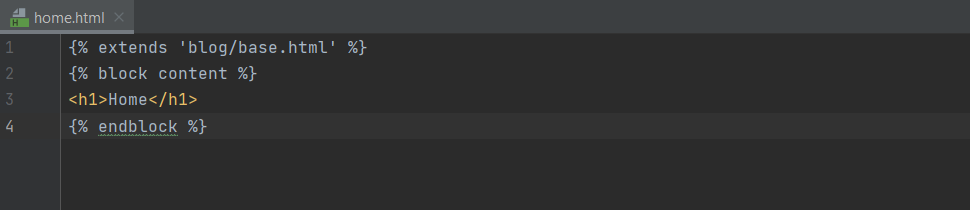
</head>

<body>

{% block content %}{% endblock %}

</body>

</html>



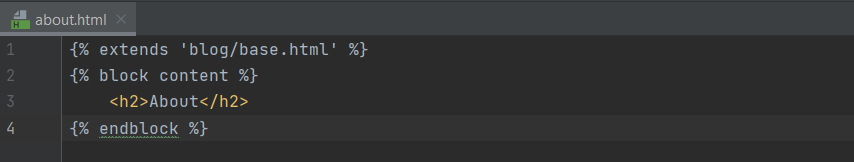
*Plik home.html*

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

{% block content %}

<h1>Home</h1>

{% endblock %}



*Plik about.html*

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

{% block content %}

<h2>About</h2>

{% endblock %}

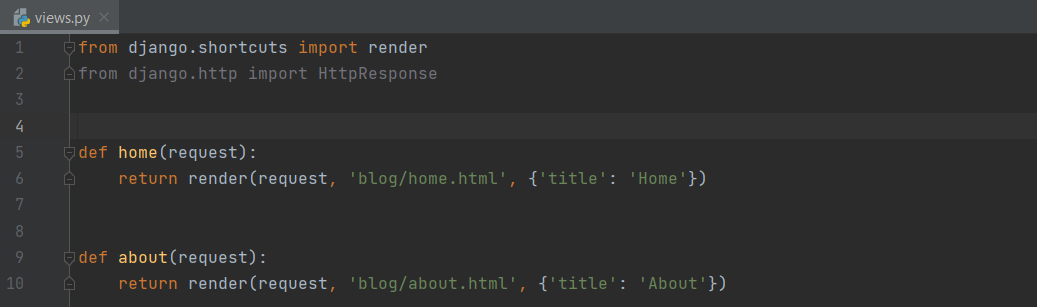
**Renderowanie pliku html**

W ostatnim kroku pozostało wprowadzenie zmian w pliku blog/views.py. Do tej pory widok danego endpointu generowaliśmy za pomocą metody HttpResponse, do której podawaliśmy bezpośrednio w formie HTMLa, co ma zostać wygenerowane.

Teraz nadeszła pora, aby widok renderować z utworzonych plików .html. Do tego celu posłuży Nam metoda render (która importowana jest z biblioteki django.shortcuts). Przyjmować będzie ona 3 argumenty: request, ścieżkę do pliku .html, z którego ma zostać wygenerowany widok oraz context, czyli wartości, które chcemy przekazać do szablonu i je wyświetlić.

To, co jest dość istotne to fakt, iż argumenty przesyłane do kontekstu HTML powinny być w postaci słownika. Argument, z którego będziemy korzystać w kodzie szablonu, powinien być odpowiednim kluczem słownika wraz z określoną wartością.

Sprawdźmy, jak mogłyby teraz wyglądać funkcje home oraz about:



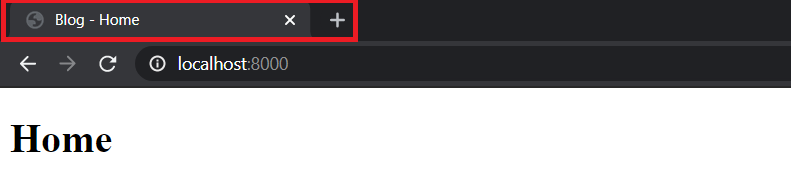
def home(request):

return render(request, 'blog/home.html', {'title': 'Home'})

def about(request):

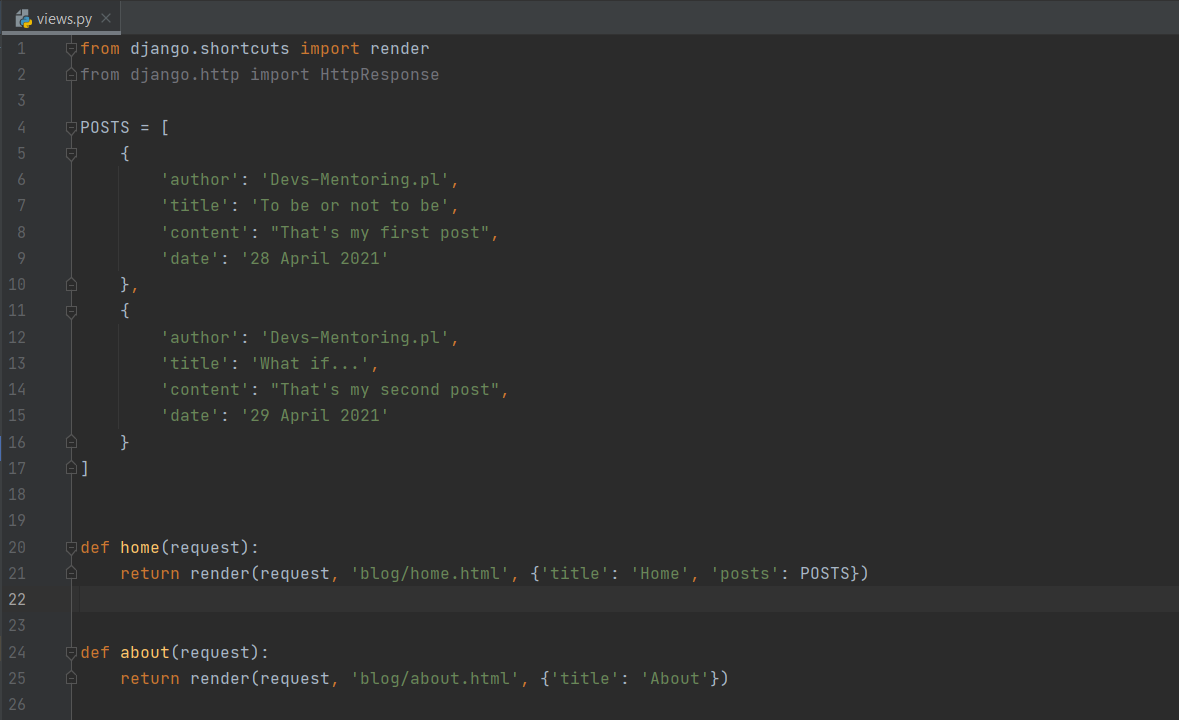
return render(request, 'blog/about.html', {'title': 'About'})

Dzięki takiemu rozwiązaniu, odpowiednio dostosujemy nazwy danych podstron.



Pójdźmy krok dalej! Dodajmy do widoku home możliwość przekazywania zawartości postów i ich wyświetlania. Chcielibyśmy, aby pojedynczy post składał się z następujących pól: title, author, content, date.

Oczywiście przy dalszym rozwoju aplikacji, posty te będziemy przechowywali w bazie danych, ale na ten moment chcielibyśmy utworzyć kilka z góry i po prostu przekazać je do szablonu.



*Zmodyfikujmy nieco plik views.py:*

from django.shortcuts import render

from django.http import HttpResponse

POSTS = [

{

'author': 'Devs-Mentoring.pl',

'title': 'To be or not to be',

'content': 'That is my first post',

'date': '28 April 2021'

},

{

'author': 'Devs-Mentoring.pl',

'title': 'What if..',

'content': 'That is my second post',

'date': '29 April 2021'

}

]

def home(request):

return render(request, 'blog/home.html', {'title': 'Home', 'posts': POSTS})

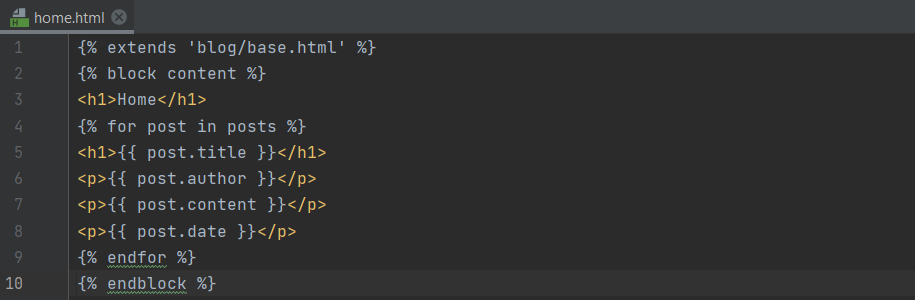
def about(request):

return render(request, 'blog/about.html', {'title': 'About'})

Jak widzisz, dodaliśmy w nim listę POSTS przechowującą dwa słowniki. Każdy pojedynczy słownik to jeden artykuł do wyświetlenia. Zwróć raz jeszcze szczególną uwagę na to, jak przekazujemy wartości do szablonu:

* Tworzymy słownik wraz z odpowiednimi kluczami: {'title': 'Home', 'posts': POSTS}
* Do kluczy przypisujemy wartości, które chcemy obsługiwać w szablonie html, np. ‘Home’, POSTS

Pozostało teraz zmienić nieco plik home.html - tak aby Nasze artykuły spod klucza posts były prawidłowo wyświetlane:



*Plik home.html*

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

{% block content %}

<h1>Home</h1>

{% for post in posts %}

<h1>{{ post.title}} </h1>

<p>{{ post.author }}</p>

<p>{{ post.content }}</p>

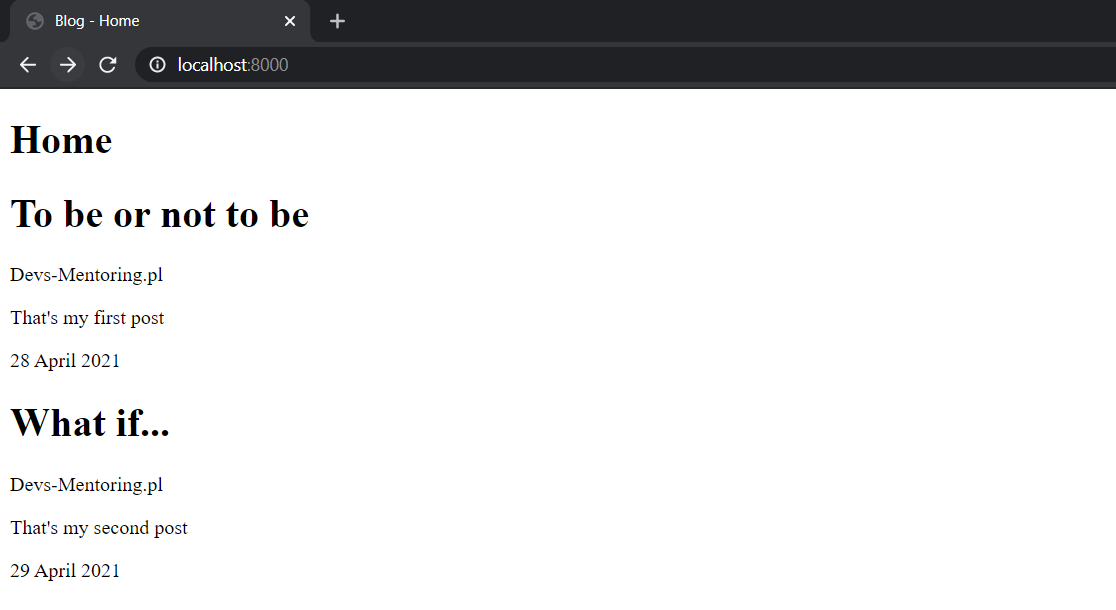
<p>{{ post.date }}</p>

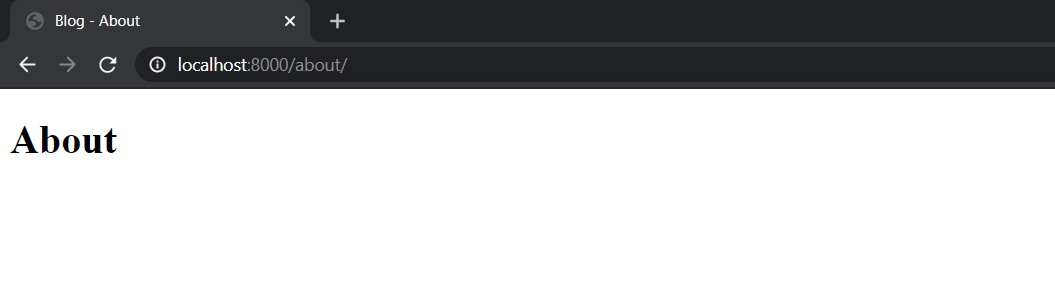
{% endfor %}

{% endblock %}

W formie dygresji chciałbym zwrócić Ci tylko uwagę na to, jak odwołujemy się do pewnych kluczy spod pojedynczego artykułu (post), który jest de facto słownikiem. Wykorzystujemy tutaj zapis z wykorzystaniem kropki. Tak więc, chcąc odwołać się do określonego klucza spod słownika post, zapiszemy post.title, a nie post[title] itd.

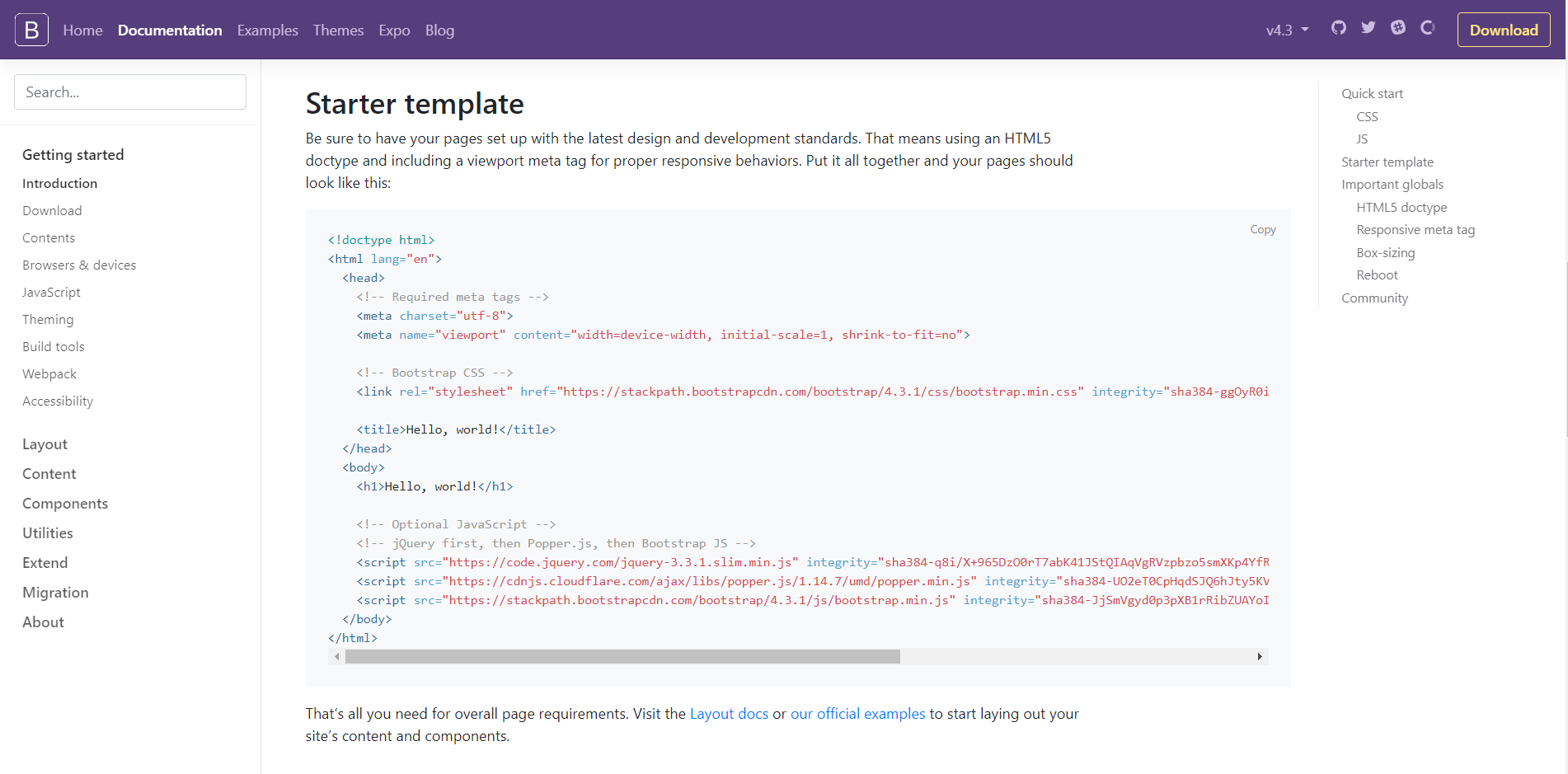
Po uruchomieniu aplikacji znanym już poleceniem python manage.py runserver, możemy ujrzeć rezultat Naszej pracy.





**Oprawa graficzna**

Nadszedł czas na naniesienie zmian strony wizualnych w Naszej stronie. Przyznaj, że w tym momencie tak prosty wygląd stron aplikacji nie jest zbyt porywający. W celu upiększenia bloga, wykorzystamy Bootstrap-a. Zacznijmy od nadaniu ogólnego kontekstu graficznego, poprzez zmianę, np. kroju i rozmiaru czcionki. W tym celu stworzę Nasz projekt zgodnie z szablonem przedstawionym tutaj: [Introduction · Bootstrap](https://getbootstrap.com/docs/4.3/getting-started/introduction/) w sekcji Starter template.

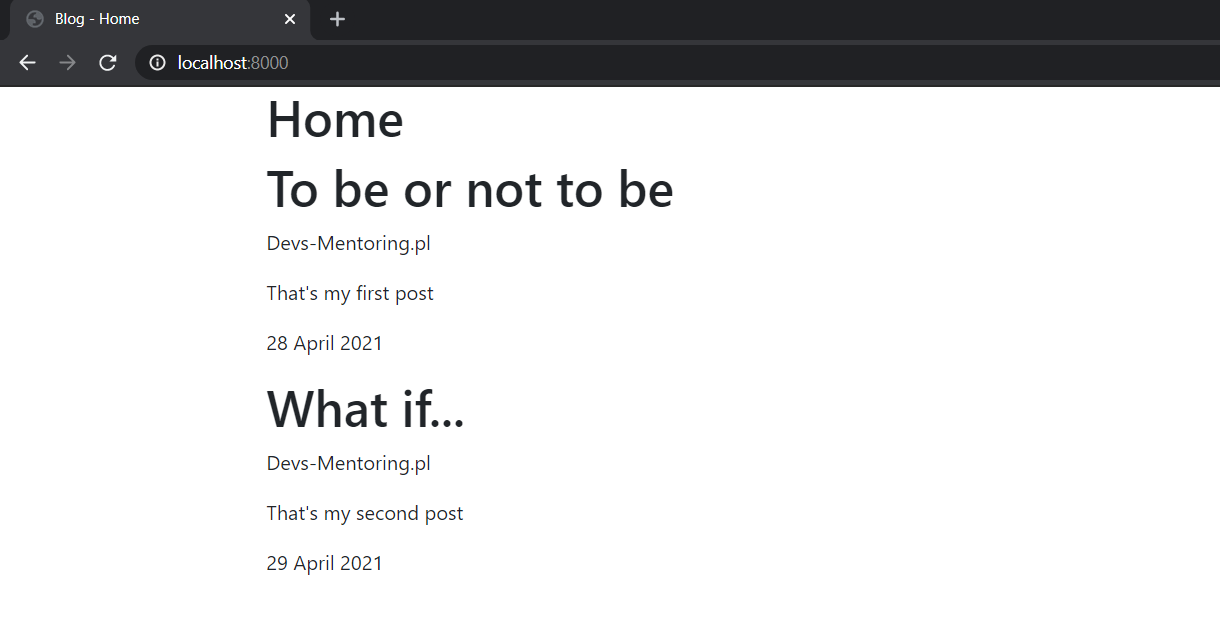


Dodatkowo, zawartość bloga umieszczę w klasie container. Jeżeli nie kojarzysz, czym są Bootstrapowe klasy, to w ramach krótkiego wyjaśnienia: umożliwiają Nam one uporządkowanie pewnych sekcji strony, nadawanie im określonych zachowań (np. wszystko, co umieszczamy wewnątrz klasy container jest m.in w pełni responsywny).

Kod pliku base.html zamienimy zatem do następującej postaci (znajdziesz go również w dołączonym do szkolenia pliku base.html):

|  |
| --- |
| <!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">   {% if title %}  <**title**>Blog - {{ title }}</**title**>  {% else %}  <**title**>Blog</**title**>  {% endif %} </**head**> <**body**>  <**div** class="container">  {% block content %}{% endblock %}  </**div**>  <!-- 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**> |

W efekcie otrzymamy następujący wygląd strony:



Następnie dodamy do projektu, panel nawigacyjny i zmodyfikujemy go w kodzie CSS (przy okazji poznasz, w jaki sposób załączać tak zewnętrzne pliki do html-a.).   
  
W celu dodania pasku nawigacji, wykorzystaj kod znajdujący się w pliku navbar.html umieszczonego w katalogu Szkolenie 2 - Snippets.

Przekopiuj go do pliku base.html, nad sekcję

<div class="container">

{% block content %}{% endblock %}

</div>

W rezultacie powinieneś otrzymać następujący plik base.html:

|  |
| --- |
| <!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">   {% 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="/">My 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="/">Home</a>  <a class="nav-item nav-link" href="/about">About</a>  </div>  <!-- Navbar Right Side -->  <div class="navbar-nav">  <a class="nav-item nav-link" href="#">Login</a>  <a class="nav-item nav-link" href="#">Register</a>  </div>  </div>  </div>  </nav>  </header>   <**div** class="container">  {% block content %}{% endblock %}  </**div**>  <!-- 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**> |

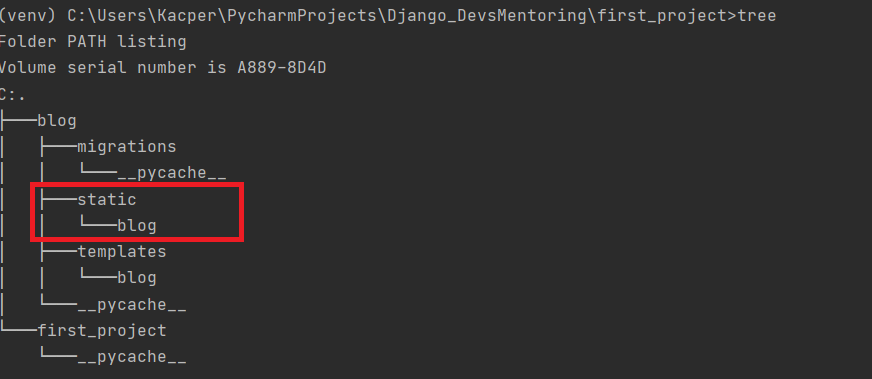
Od razu dodajmy do tego pliku html zdefiniowany plik css (base.css). Jego zawartość również znajdziesz w katalogu dołączonym do szkolenia.

Aby jednak móc utworzyć plik będący css-em, konieczne jest utworzenie nowego katalogu w Naszej aplikacji. Przechowywać będziemy w Nim wszystkie “statyczne” pliki, czyli, np. pliki .css, pliki .js.

Przy tworzeniu postępujemy podobnie jak przy tworzeniu katalogu template.

Znajdując się w folderze /blog, tworzymy folder /static, a w nim kolejny folder o takiej samej nazwie, co Nasza aplikacja (czyli folder blog).

W rezultacie powinniśmy otrzymać następującą strukturę:



Plik base.css (przekopiuj poniższy kod lub przejdź do załączonego pliku w snippets - base.css)

|  |
| --- |
| body {  background: #fafafa;  color: #333333;  margin-top: 5rem; }  h1, h2, h3, h4, h5, h6 {  color: #444444; }  ul {  margin: 0; }  .bg-steel {  background-color: #5f788a; }  .site-header .navbar-nav .nav-link {  color: #cbd5db; }  .site-header .navbar-nav .nav-link:hover {  color: #ffffff; }  .site-header .navbar-nav .nav-link.active {  font-weight: 500; }  .content-section {  background: #ffffff;  padding: 10px 20px;  border: 1px solid #dddddd;  border-radius: 3px;  margin-bottom: 20px; }  .article-title {  color: #444444; }  a.article-title:hover {  color: #428bca;  text-decoration: none; }  .article-content {  white-space: pre-line; }  .article-img {  height: 65px;  width: 65px;  margin-right: 16px; }  .article-metadata {  padding-bottom: 1px;  margin-bottom: 4px;  border-bottom: 1px solid #e3e3e3 }  .article-metadata a:hover {  color: #333;  text-decoration: none; }  .article-svg {  width: 25px;  height: 25px;  vertical-align: middle; }  .account-img {  height: 125px;  width: 125px;  margin-right: 20px;  margin-bottom: 16px; }  .account-heading {  font-size: 2.5rem; } |

Aby dodać referencję do tak utworzonego pliku base.css, wprowadzimy dwie zmiany do blog/base.html:

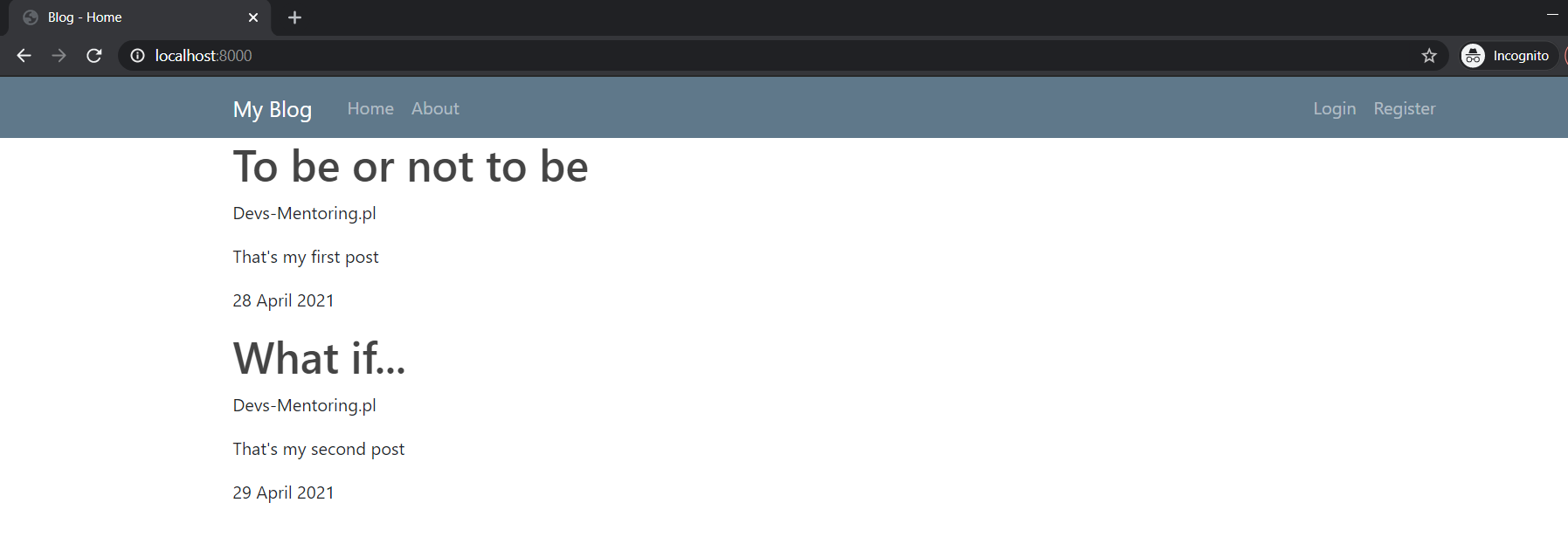
* Umieścimy linię {% load static %} na początku pliku base.html
* Dodamy odnośnik do pliku css (w sekcji head) o następującej zawartości:   
  <link rel="stylesheet" type="text/css" href="{% static 'blog/base.css' %}" />

W gwoli wyjaśnienia: {% load static %} umożliwia Nam łatwe odwoływanie się do pewnych ścieżek, przez podanie jedynie relatywnej nazwy katalogu (np. blog zamiast C:\Users\Example\PycharmProjects\Django\first\_project\blog), do którego chcemy się odwołać. Dzięki temu, plik blog/base.css, możemy zaadresować, np. w następujący sposób: href="{% static 'blog/base.css' %}

Po wprowadzeniu wszystkich zmian, otrzymamy następujący plik base.html:

|  |
| --- |
| {% 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="/">Home</**a**>  <**a** class="nav-item nav-link" href="/about">About</**a**>  </**div**>  <!-- Navbar Right Side -->  <**div** class="navbar-nav">  <**a** class="nav-item nav-link" href="#">Login</**a**>  <**a** class="nav-item nav-link" href="#">Register</**a**>  </**div**>  </**div**>  </**div**>  </**nav**>  </**header**>   <**div** class="container">  {% block content %}{% endblock %}  </**div**>  <!-- 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**> |

Efekt wizualny:



Ostateczne poprawki…

Na koniec jeszcze dopracujemy nieco wygląd Naszej strony. Będziemy chcieli:

* Utworzyć sekcję “main”
* Umieścić każdy artykuł w oddzielnej sekcji “article”

W pliku base.html obecny blok content umieścimy w sekcji <main> i dodatkowych <div>. Tak, że zamiast takiego kodu...

|  |
| --- |
| <**div** class="container">  {% block content %}{% endblock %} </**div**> |

… otrzymasz następujący:

|  |
| --- |
| <**main** role="main" class="container">  <**div** class="row">  <**div** class="col-md-8">  {% block content %}{% endblock %}  </**div**>  </**div**> </**main**> |

{% 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">

<a class="nav-item nav-link" href="#">Login</a>

<a class="nav-item nav-link" href="#">Register</a>

</div>

</div>

</div>

</nav>

</header>

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

<div class="row">

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

{% 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>

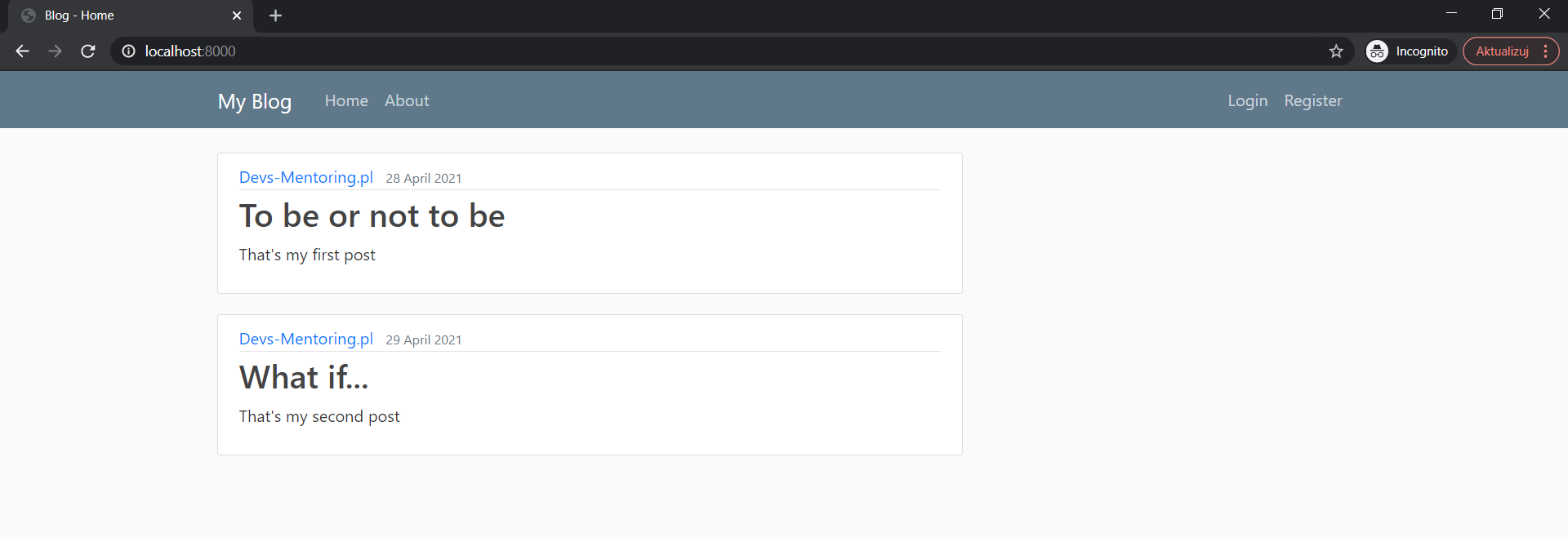
</body>

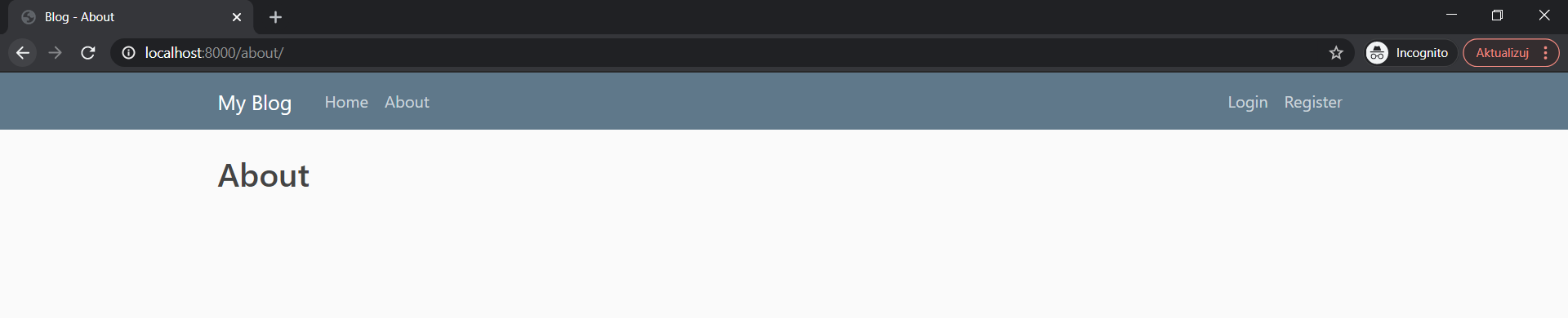
</html>

A plik home.html zmodyfikujemy do postaci:

|  |
| --- |
| {% extends 'blog/base.html' %} {% block content %}  {% for post in posts %}  <**article** class="media content-section">  <**div** class="media-body">  <**div** class="article-metadata">  <**a** class="mr-2" href="#">{{ post.author }}</**a**>  <**small** class="text-muted">{{ post.date }}</**small**>  </**div**>  <**h2**><**a** class="article-title" href="#">{{ post.title }}</**a**></**h2**>  <**p** class="article-content">{{ post.content }}</**p**>  </**div**>  </**article**>  {% endfor %} {% endblock %} |

Jako rezultat otrzymamy następujący wygląd strony:





**Na zakończenie…**

Ostatnia rzecz, na jaką chciałem zwrócić uwagę, to ułatwienie sobie pracy ze ścieżkami plikowymi.

Zwróćmy uwagę na poniższe linie kodu znajdujące się w pliku base.html (w sekcji dotyczącej navbar):

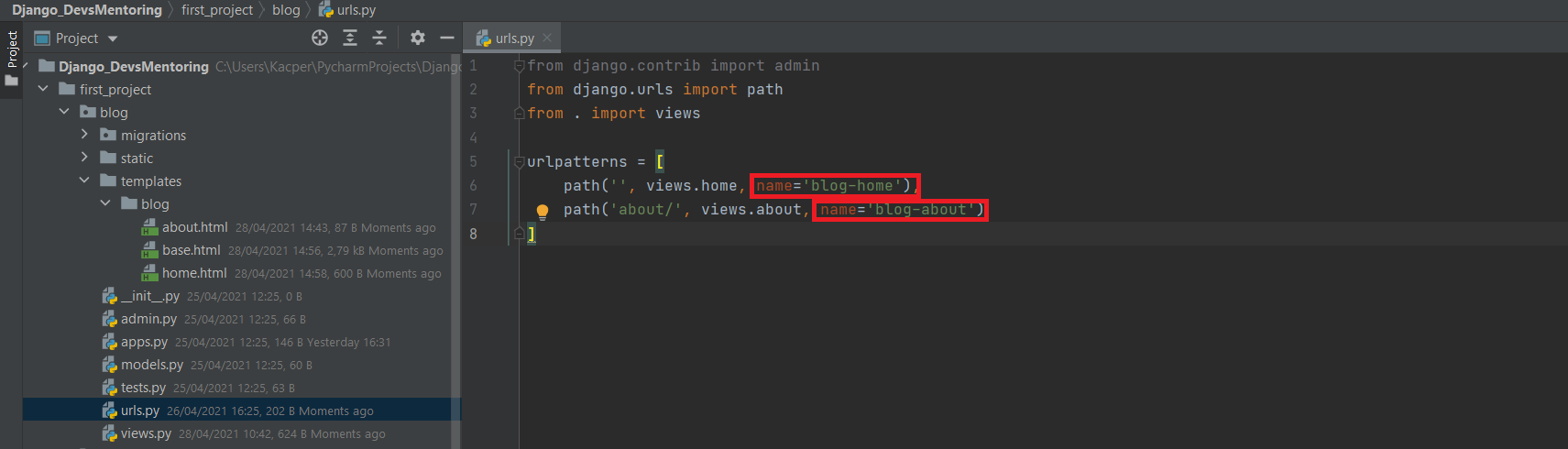
<a class="nav-item nav-link" href="/">Home</a>

<a class="nav-item nav-link" href="/about">About</a>

To, co chciałbym Ci zaproponować, to zmiana wartości znacznika href, na taki, abyśmy byli maksymalnie zwolnieni z konieczności pamiętania wszystkich adresów URL danych podstron, ale również, aby Nasza praca została nieco zautomatyzowana.

Otóż podawanie ścieżek “na sztywno”, np. w postaci: /about, /, nie jest najlepszym rozwiązaniem w przypadku dynamicznie rozbudowujących się aplikacji. Wyobraź sobie, że w pewnym momencie chcielibyśmy, aby endpoint /about został zmieniony na /about-us. Wówczas w celu dokonania takiej zmiany, musielibyśmy w każdym pojedynczym przekierowaniu na tę podstronę, zmienić wartość znacznika href z “/about” na “/about-us”. Przyznaj, że niezbyt wygodne rozwiązanie.

Dlatego, aby rozwiązać ten problem, wykorzystamy nazwy danych endpointów, które określiśmy w pliku /blog/urls.py na początku pracy z projektem.



Od tej pory, chcąc zbudować link umożliwiający przejście do endpointu związanego z sekcją about, wystarczy zapisać kod zgodny ze wzorem:

{% url ‘nazwa’ %}, czyli np. {% url ‘blog-about %}

Efekt będzie identyczny, jak przy podawaniu “na sztywno” ścieżek do endpointów, ale za to my nie będziemy martwili się o poprawność działania programu przy bardziej gruntownych zmianach i modyfikacjach nazw endpointów.

Wcześniej przedstawione znaczniki <a> z pliku base.html zmodyfikujemy zatem do postaci:

<a class="nav-item nav-link" href="{% url ‘blog-home’ %}">Home</a>

<a class="nav-item nav-link" href="{% url ‘blog-about’ %}">About</a>

{% 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-home’ %}">Home</a>

<a class="nav-item nav-link" href="{% url ‘blog-about’ %}">About</a>

</div>

<!-- Navbar Right Side -->

<div class="navbar-nav">

<a class="nav-item nav-link" href="#">Login</a>

<a class="nav-item nav-link" href="#">Register</a>

</div>

</div>

</div>

</nav>

</header>

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

<div class="row">

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

{% 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>

To wszystko jeśli chodzi o szkolenie nr 2. Efekt pracy znajdziesz w załączniku Szkolenie 2. Templates.