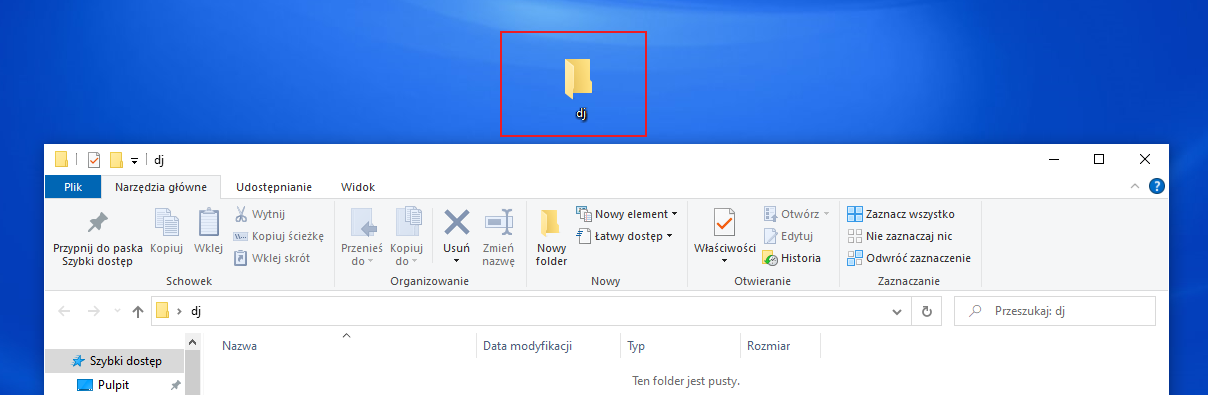
https://mega.nz/folder/mwlADAJR#tWRZp8vuJIBBvjxYcG5BRA/folder/q8ckQboY

Zakładam, że django już jest zainstalowane.

Odcinek 1:

Tworzę folder o dowolnej nazwie (u mnie dj):



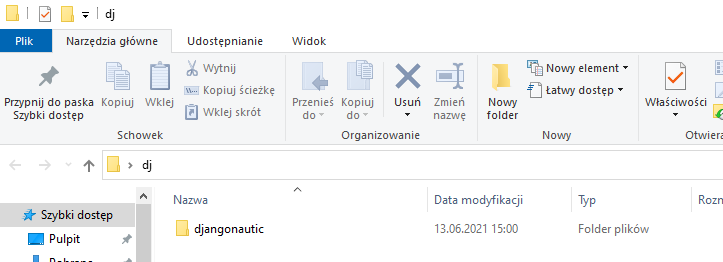
Otwieram CMD, przechodzę do stworzonego folderu (*cd* *C:\Users\TPS\_target\Desktop\dj*) i piszę:

*django-admin startproject djangonautic*

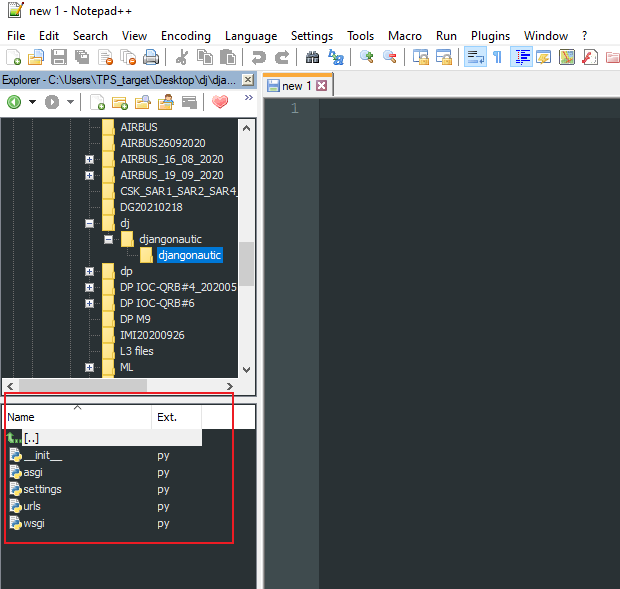
gdzie djangonautic to nazwa projektu:



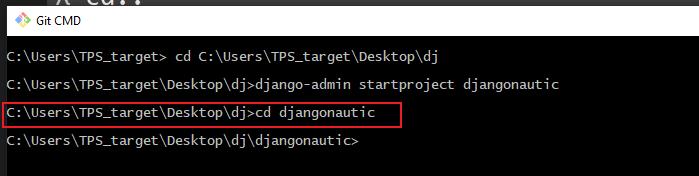
W folderze dj mamy projekt djangonautic:



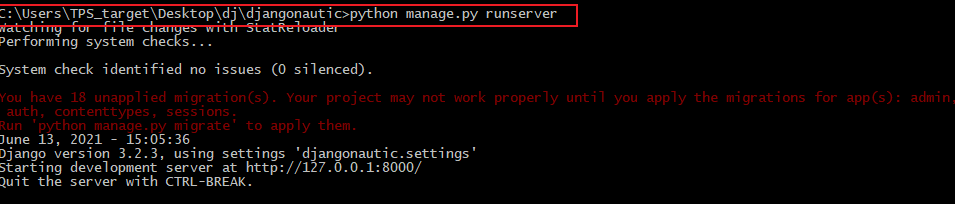
W folderze djangonautic mamy kolejny folder djngonautic. Otwieram Notepad++ w którym widać jakie pliki zostały automatycznie stworzone:



Przechodzę do folderu djangonautic:

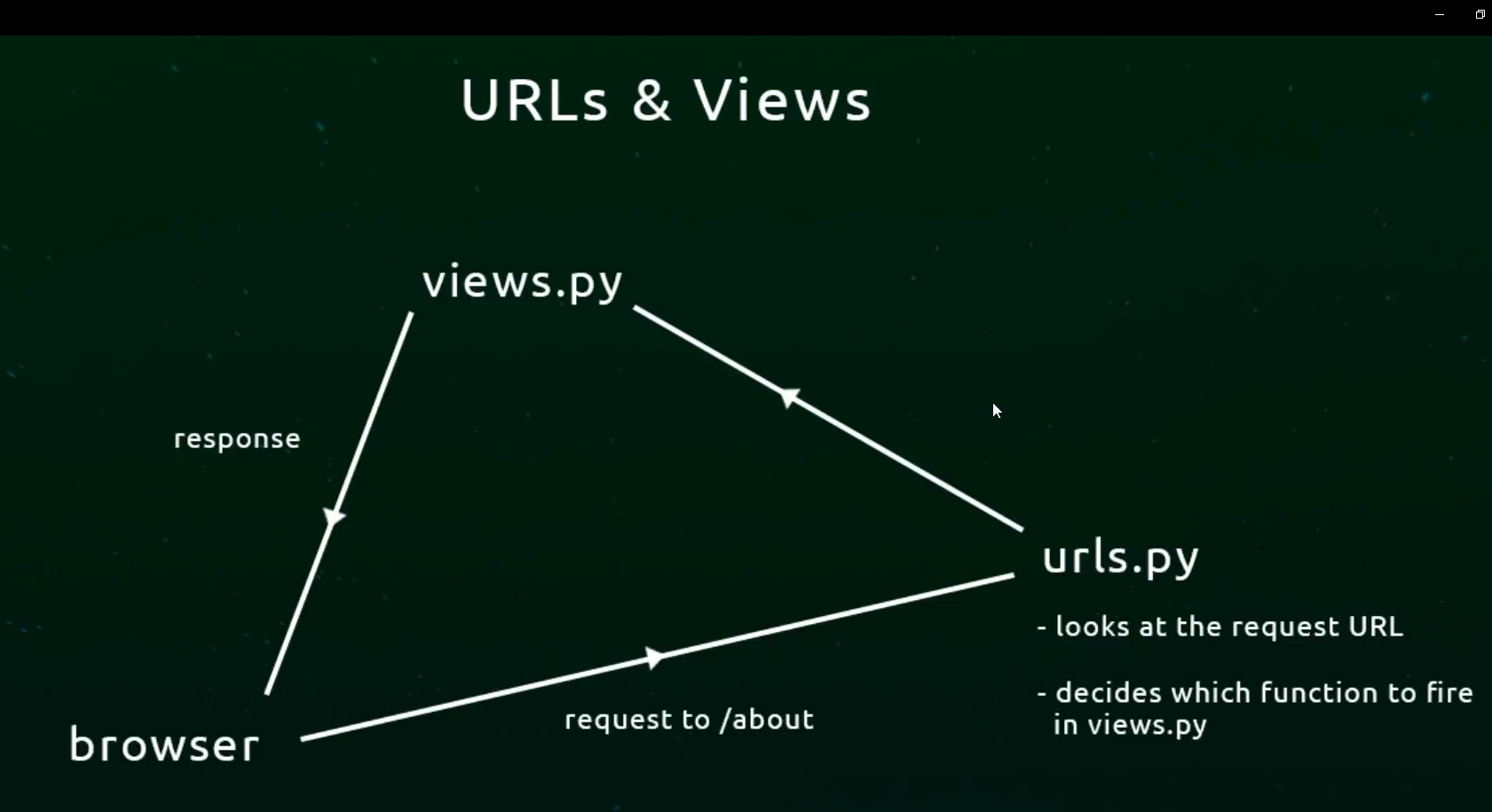


Odpalam server komendą *python manage.py runserver:*



Odcinek 2:

W tym odcinku konfigurujemy konkretne strony.



Otwieram plik urls.py:



Dodaję taki kod:

*from django.conf.urls import url*

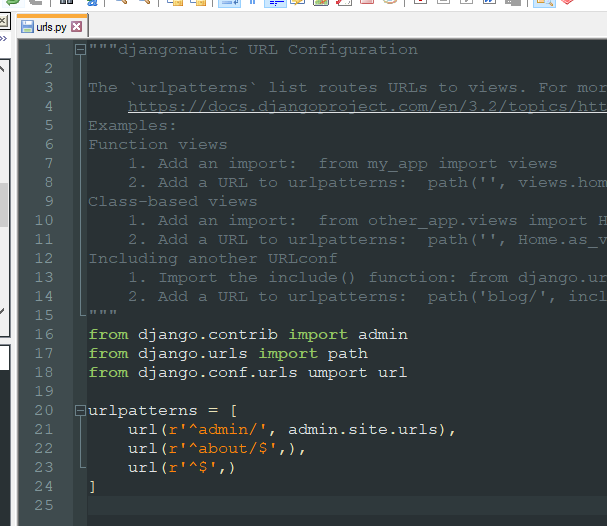
*urlpatterns = [*

*url(r'^admin/', admin.site.urls),*

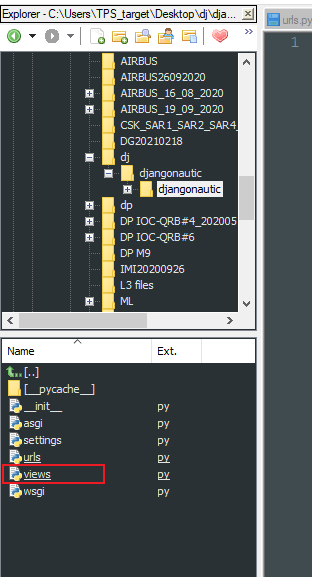
*url(r'^about/$',),*

*url(r'^$',)*

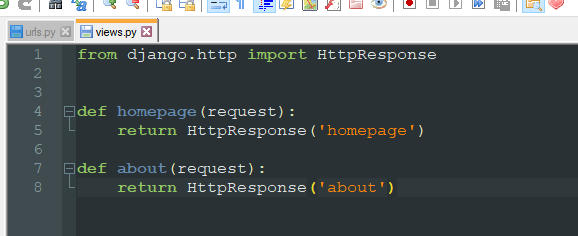
*]*



Tworzę plik views.py:



Zawartość pliku views.py:



*from django.http import HttpResponse*

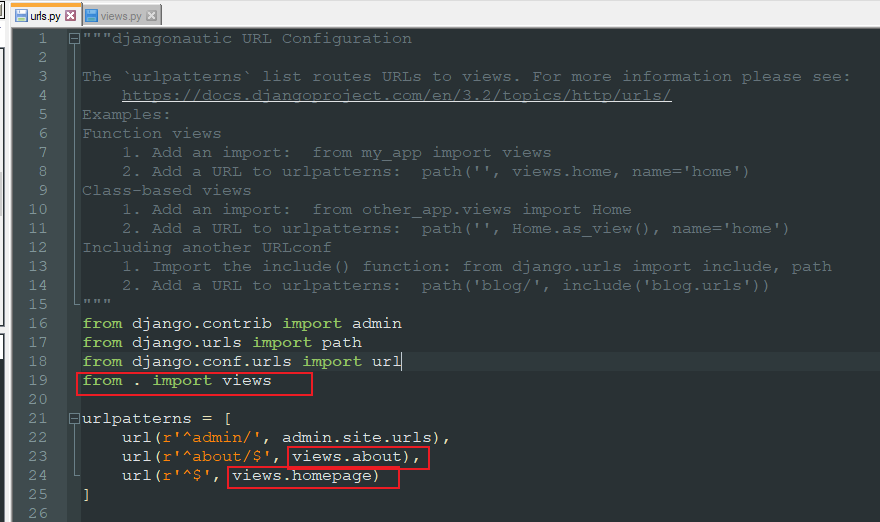
*def homepage(request):*

*return HttpResponse('homepage')*

*def about(request):*

*return HttpResponse('about')*

Modyfikujemy plik urls.py:



*from . import views*

*urlpatterns = [*

*url(r'^admin/', admin.site.urls),*

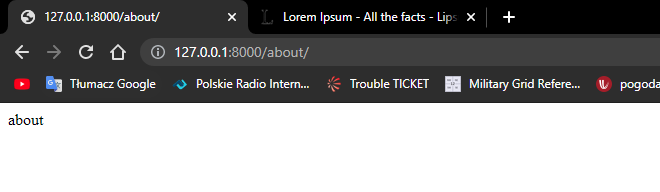
*url(r'^about/$', views.about),*

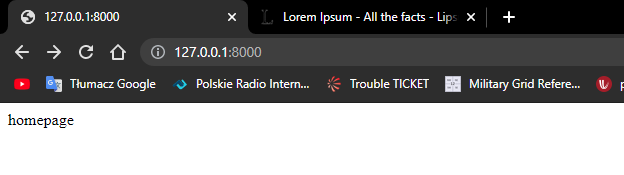
*url(r'^$', views.homepage)*

*]*

Dołączamy plik views a następnie określamy jakie funkcje mają być wywoływane po wejściu w about i po wejściu na stronę.

Tak to wygląda w przeglądarce:





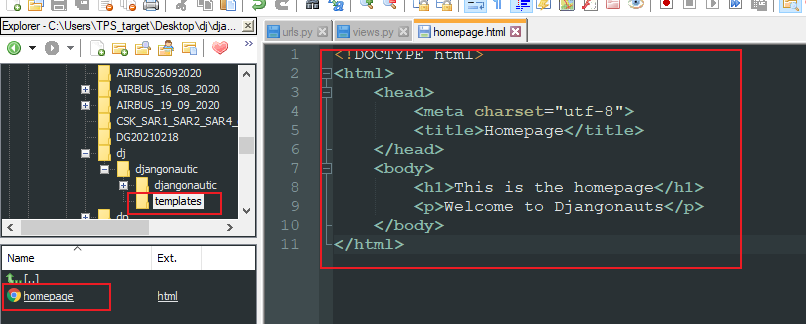
Odcinek 3:

W tym odcinku konfigurujemy strone tak żeby korzystała z plików html.

Tworzymy template directory:



W środku tego katalogu robimy plik homepage.html z następującą zawartością:



*<!DOCTYPE html>*

*<html>*

*<head>*

*<meta charset="utf-8">*

*<title>Homepage</title>*

*</head>*

*<body>*

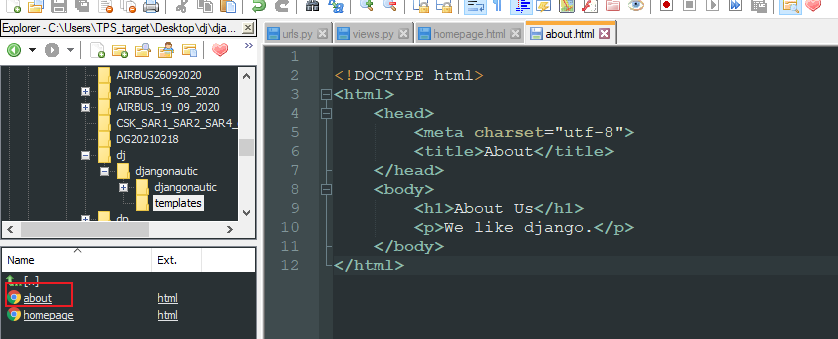
*<h1>This is the homepage</h1>*

*<p>Welcome to Djangonauts</p>*

*</body>*

*</html>*

Analogicznie tworzymy plik about.html:



*<!DOCTYPE html>*

*<html>*

*<head>*

*<meta charset="utf-8">*

*<title>About</title>*

*</head>*

*<body>*

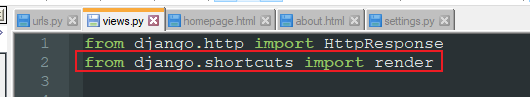
*<h1>About Us</h1>*

*<p>We like django.</p>*

*</body>*

*</html>*

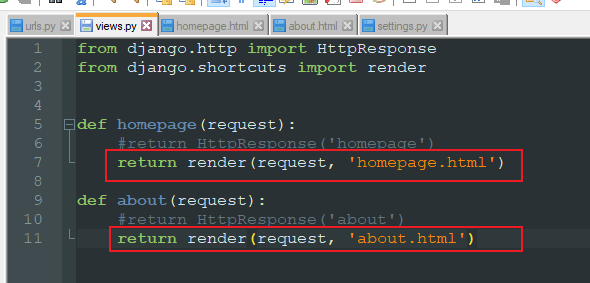
Przechodzimy do pliku views.py i uzupełniamy go o następujący kod:



*from django.shortcut import render*

Pozwala to na renderowanie plików html.

Następnie wskazujemy które pliki html mają być renderowane dla naszych funkcji:



*def homepage(request):*

*#return HttpResponse('homepage')*

*return render(request, 'homepage.html')*

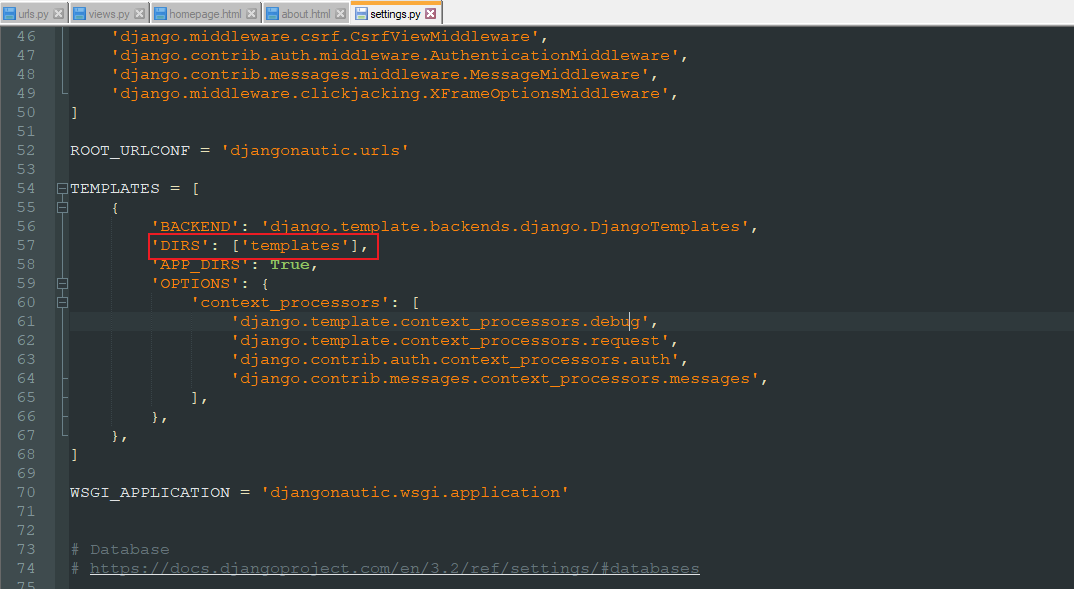
*def about(request):*

*#return HttpResponse('about')*

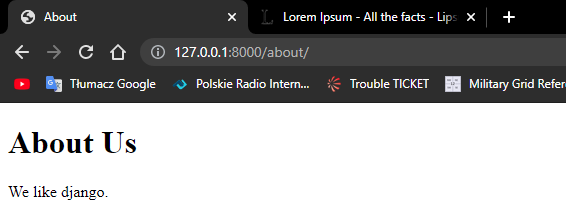
*return render(request, 'about.html')*

Funkcja render jako pierwszy argument zawsze przyjmuje request. Drugi argument to strona jaka ma być renderowana.

Przechodzimy do pliku settings.py i wskazujemy gdzie nasze templatki htmlowe są przechowywane:

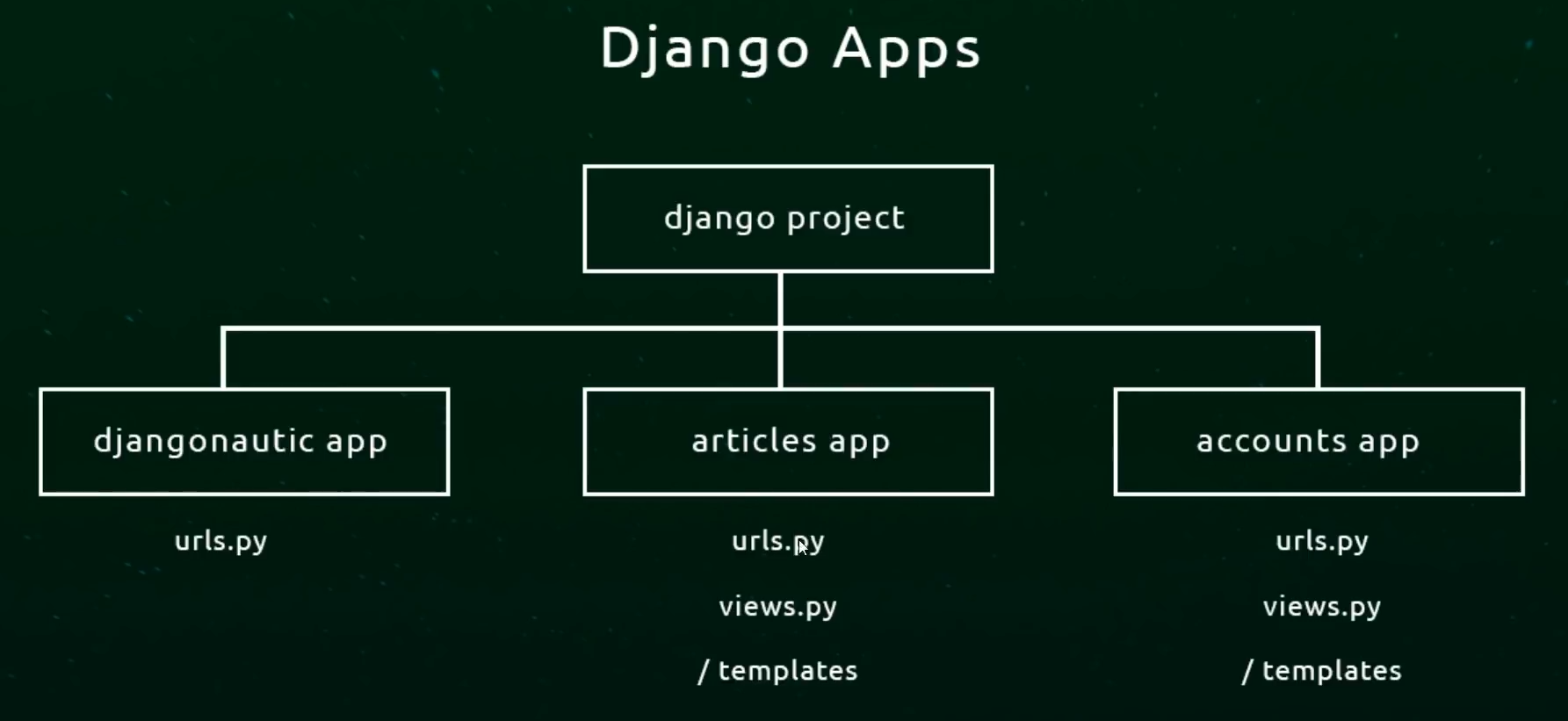


Patrzymy czy nasze htmlki są renderowane:

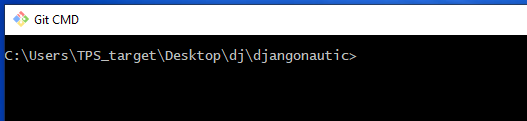


Odcinek 5:

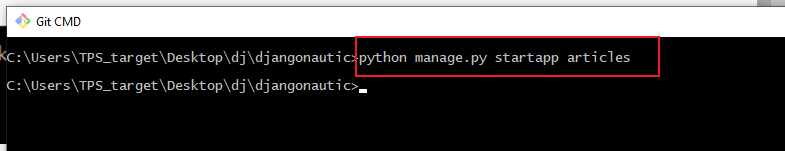
Tworzymy aplikacje w naszym projekcie. Docelowo będziemy mieli 3 aplikacje:



Tworzymy aplikację articles. W tym celu przechodzimy do „zewnętrznego” pliku djangonautic:

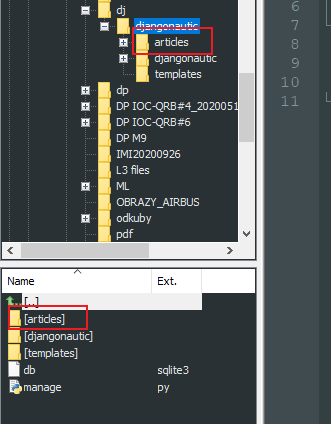


I wpisujemy komendę:

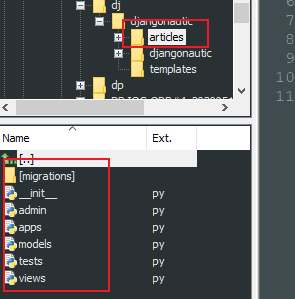


*python manage.py startapp articles*

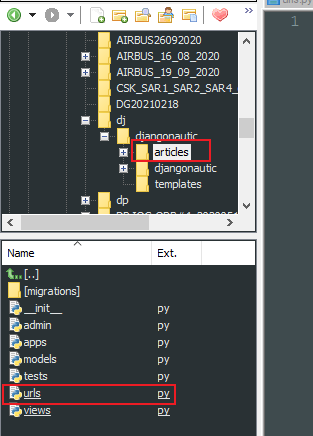
W djangonautics pojawił nam się folder articles:



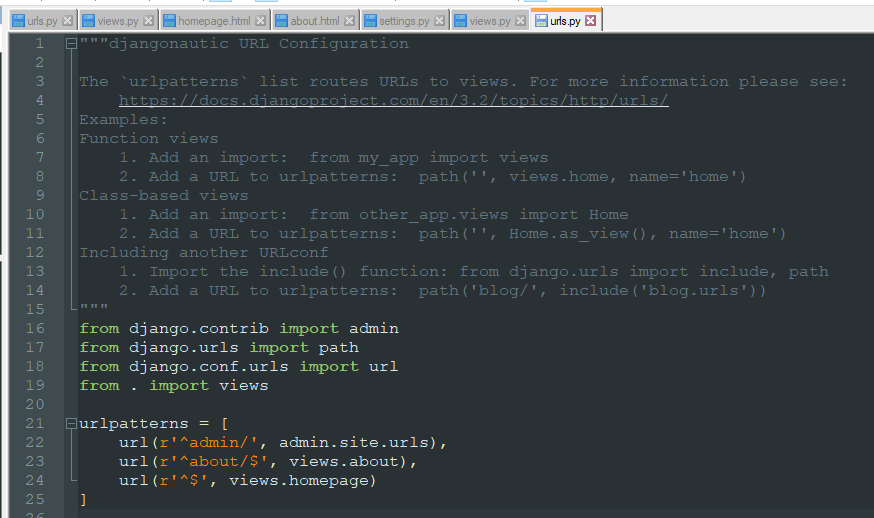
W środku mamy następujące pliki:



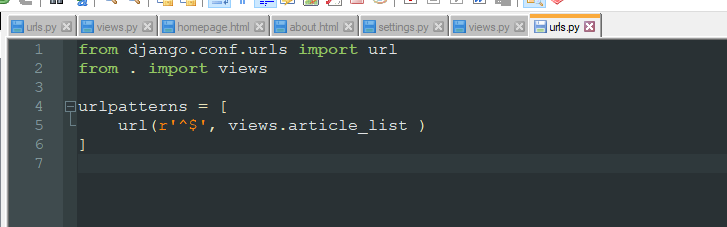
Tworzymy plik urls.py:



I kopiujemy do niego zawartość pliku urls.py z katalogu djangonautic:



Modyfikujemy tę zawartość do postaci docelowej:



*from django.conf.urls import url*

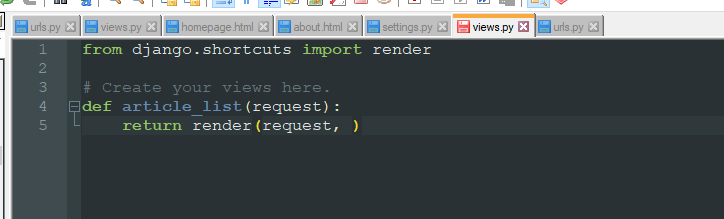
*from . import views*

*urlpatterns = [*

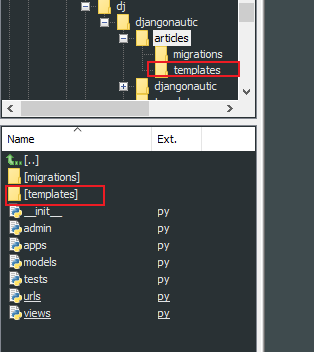
*url(r'^$', views.article\_list )*

*]*

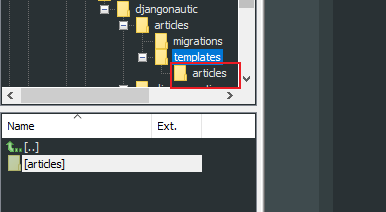
I przechodzimy do pliku views.py i wstępnie przygotowujemy kod, który ma renderować htmlkę:



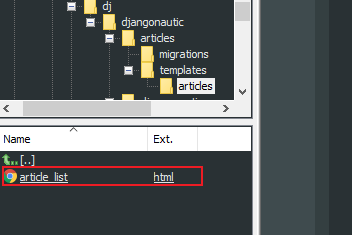
Teraz przygotowujemy folder w którym mają być templatki dla aplikacji artices. W folderze articles tworzymy folder templates:



W tym stworzonym folderze templates tworzymy z kolei folder articles:



W środku tworzymy plik article\_list.html:



Ten plik ma mieć następującą zawartość:

*<!DOCTYPE html>*

*<html>*

*<head>*

*<meta charset="utf-8">*

*<title></title>*

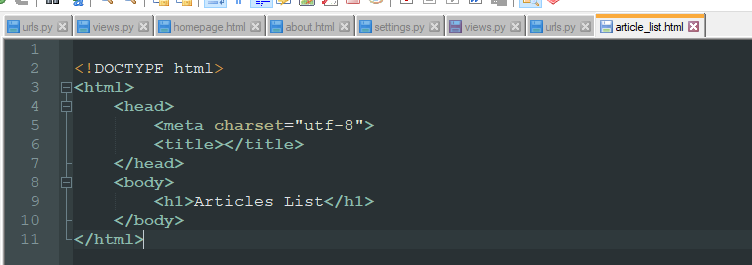
*</head>*

*<body>*

*<h1>Articles List</h1>*

*</body>*

*</html>*



Wracamy do views.py i uzupełniamy kod tak żeby renderował stworzoną przed chwilą htmlkę:



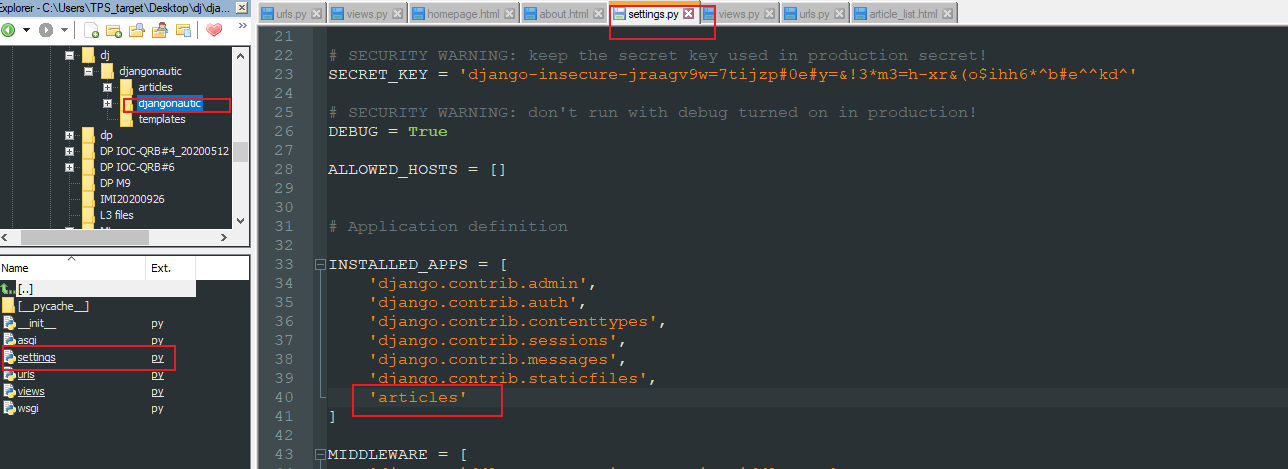
*from django.shortcuts import render*

*# Create your views here.*

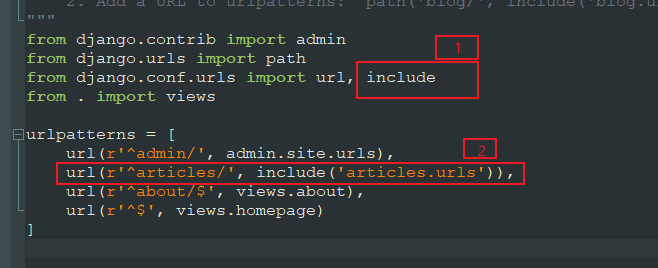
*def article\_list(request):*

*return render(request, 'articles/article\_list.html')*

Za każdym razem gdy tworzymy nową aplikację (w naszym przypadku articles) należy ją zarejestreować w projekcie. Robi się to w pliku settings.py znajdującym się w katalogu djangodautic:



Teraz chcemy spowodować, żeby po wejściu w link strony kończący się na /articles zostaniemy przekierowani do strony aplikacji articles. W tym celu musimy dokonać odpowiedniej konfiguracji w pliku urls.py znajdującym się w katalogu djangonautic urls.py. Importujemy moduł include (1), który pozwoli nam odwołać się do innej aplikacji (articles) i dodajemy linijkę w której wskazujemy link i odwołanie (2):



*from django.contrib import admin*

*from django.urls import path*

*from django.conf.urls import url, include*

*from . import views*

*urlpatterns = [*

*url(r'^admin/', admin.site.urls),*

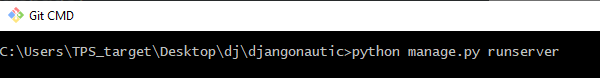
*url(r'^articles/', include('articles.urls')),*

*url(r'^about/$', views.about),*

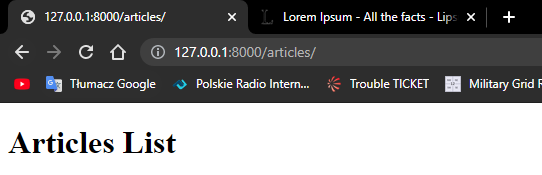
*url(r'^$', views.homepage)*

*]*

W razie potrzemy restartujemy serwer w cmd:

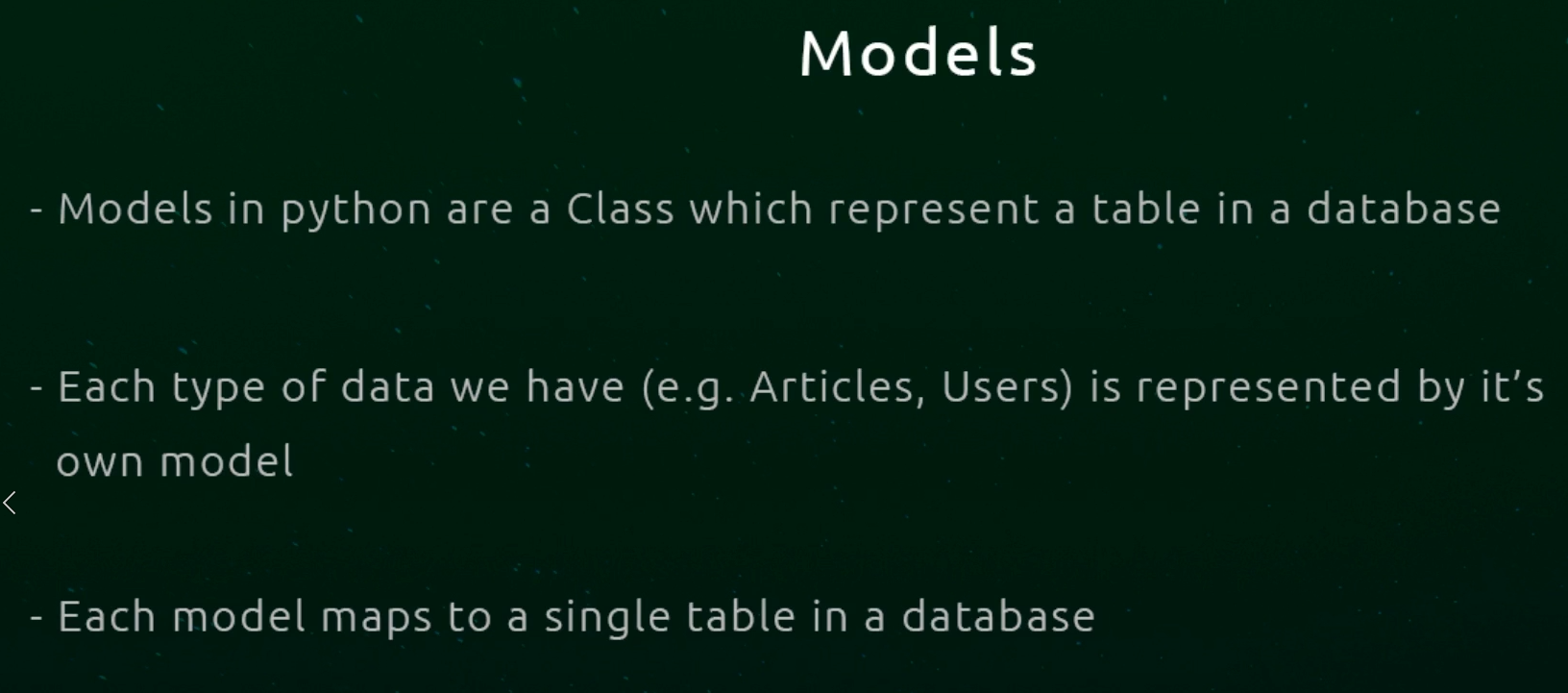


I już articles powinno działać:



Odcinek 6:

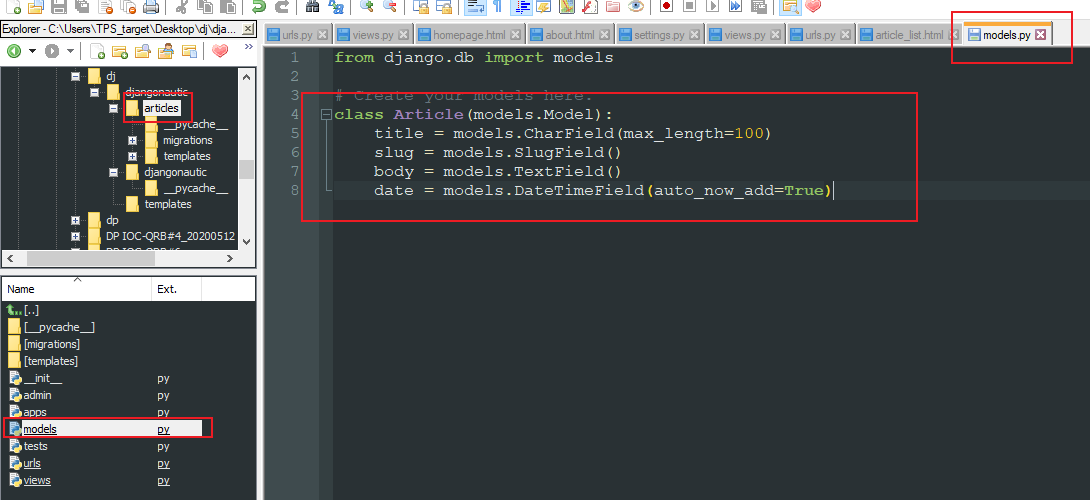
Stworzymy model czyli tabelę w bazie danych.





Później będziemy używać ORM żeby wprowadać dane do bazy.

Wchodzimy do models.py w folderze articles. Tworzymy tam model, który natępnie zostanie zmapowany na tabelę Article:



*class Article(models.Model):*

*title = models.CharField(max\_length=100)*

*slug = models.SlugField()*

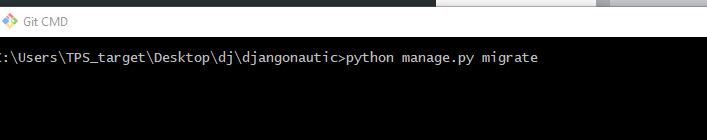
*body = models.TextField()*

*date = models.DateTimeField(auto\_now\_add=True)*

Odcinek 7:

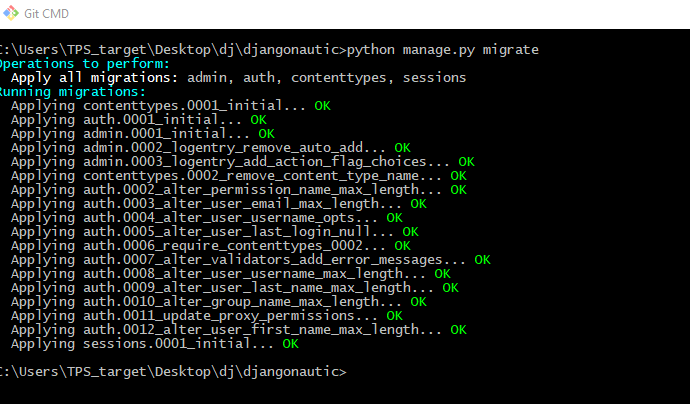
Do tej pory stworzyliśmy tylko model a tabela jeszcze nie istnieje. Trzeba teraz zrobić migrację, która spowoduje stworzenie tabeli.

W cmd wykonujemy następującą komendę:

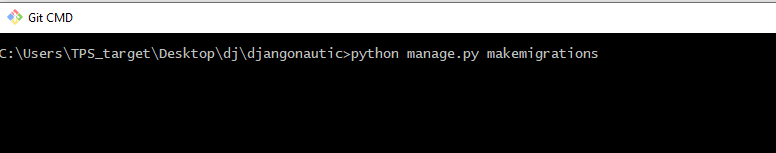


*python manage.py migrate*

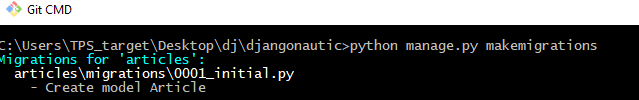
Poniżej wynik:



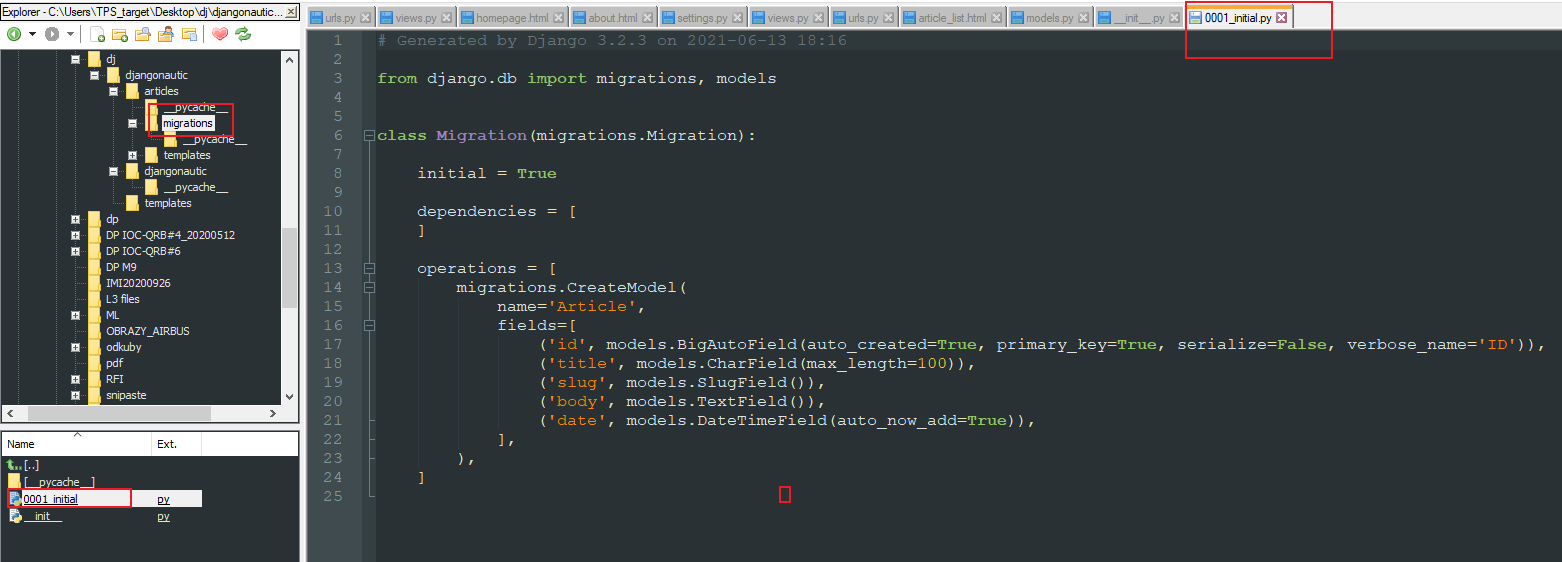
Tworzymy plik migracji:



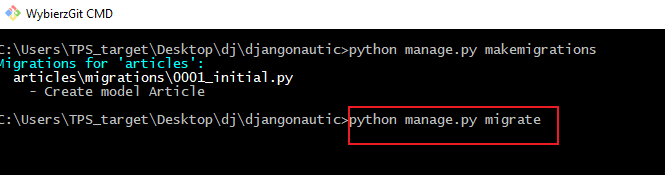
Wynik:



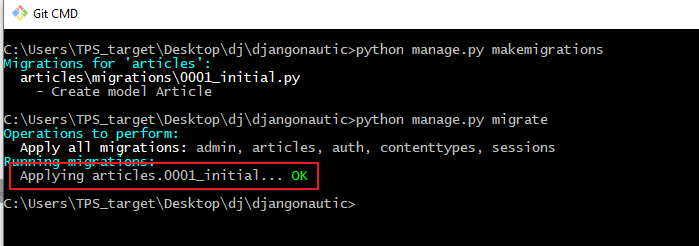
W katalogu migrations powstał taki plik:



Następnie ponownie trzeba zmigrować ten nowo powstały plik:



Wynik:

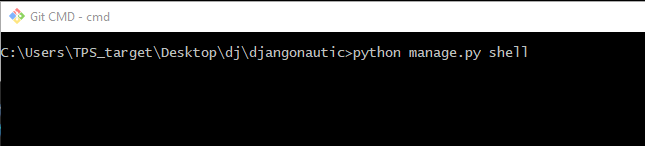


Model został zmigrowany. Za każdym razem przy zmianie trzeba będzie robić plik za pomocą komendy makemigrations a następnie migrate.

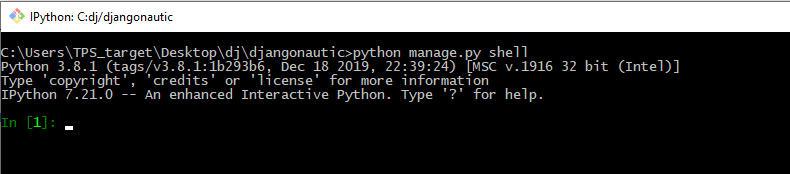
Odcinek 8:

ORM – interakcja z bazą danych.

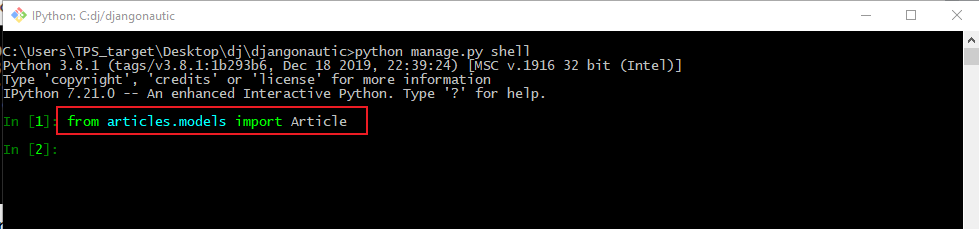
Otwieramy w cmd „interactive shell”:



Wynik:

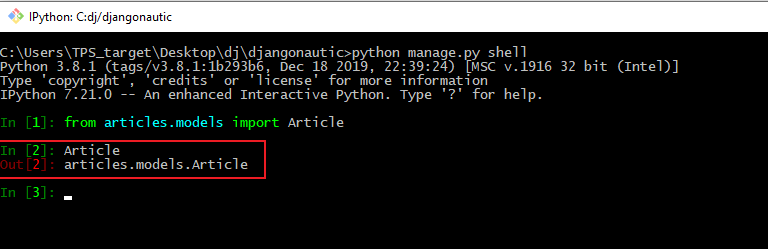


Imporujemy tabelę Article:

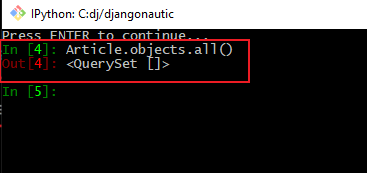


*From articles.models import Article*

Teraz możemy przywołać Article:

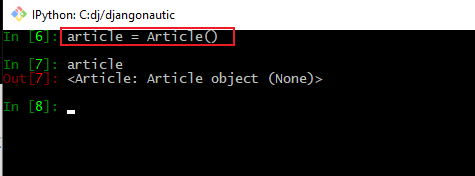


Chcemy zobaczyć wszystkie rekordy w tej tabeli:



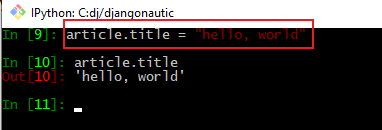
*Article.objects.all()*

Tabela jest póki co pusta więc komenda nic nam nie zwróciła. W związku z tym dodamy rekord. Tworzymy obiekt klasy Article:



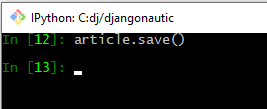
*article = Article()*

W polu title dodajemy „hello world”:



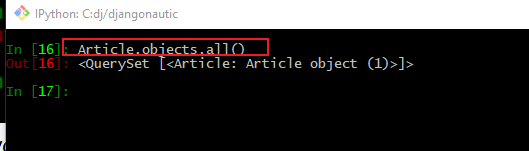
*article.title = „hello world”*

Teraz zapisujemy w bazie obiekt article:



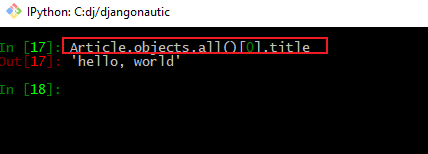
*article.save()*

Teraz sprawdzamy jakie obiekty klasy Article mamy:



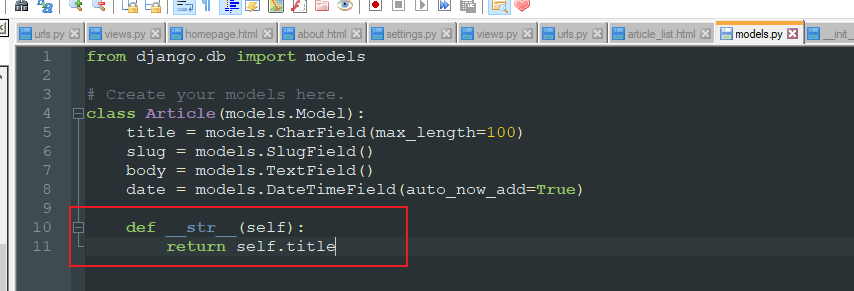
*Article.objects.all()*

Możemy odwołać się do pierwszego obiektu i jego konkretnego parametru:



*Articles.objects.all()[0].title*

Możemy zrobić tak, że po odwołaniu się do obiektu będzie się nam wyświetlał konkretny jego parametr, np. title. W tym celu wchodzimy do models.py i dodajemy następujący kod:

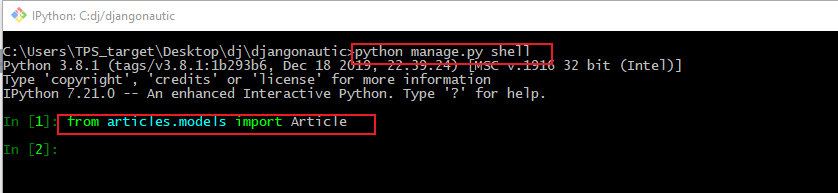


*def \_\_str\_\_(self):*

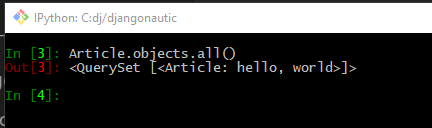
*return self.title*

Uważaj zeby zrobić wcięcia tak, żeby metoda \_\_str\_\_ należała do klasy Article!

Musimy teraz ponownie uruchomić shell i zaimportować model Articles:

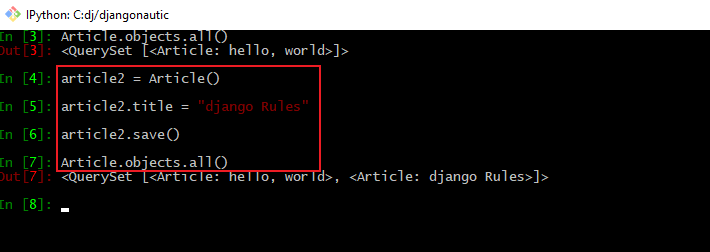


A następnie odwołujemy się do obiektów klasy Article:



I widzimy, że wyświetla nam się tytuł.

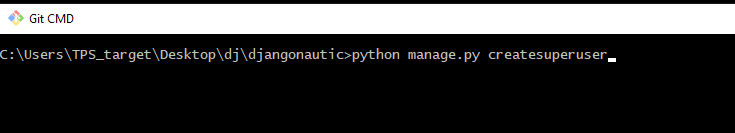
Dodajemy kolejny rekord:



Odcinek 9:

Django Admin.

Konfigurujemy konto administarcyjne:



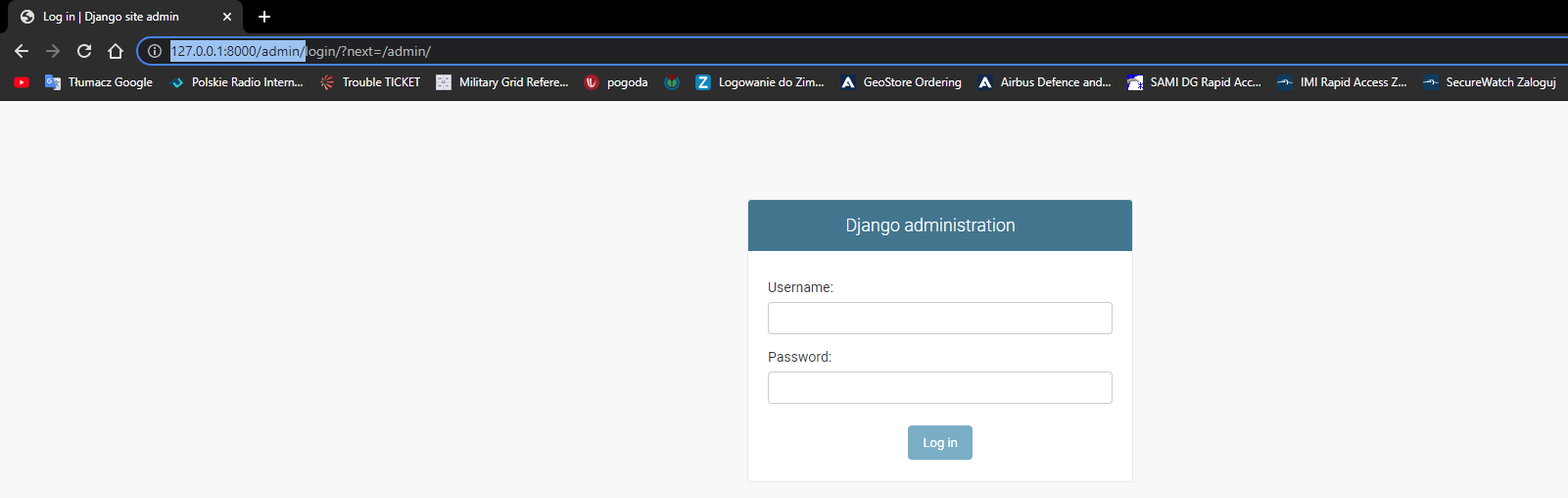
*python manage.py createsuperuser*

Username: admin

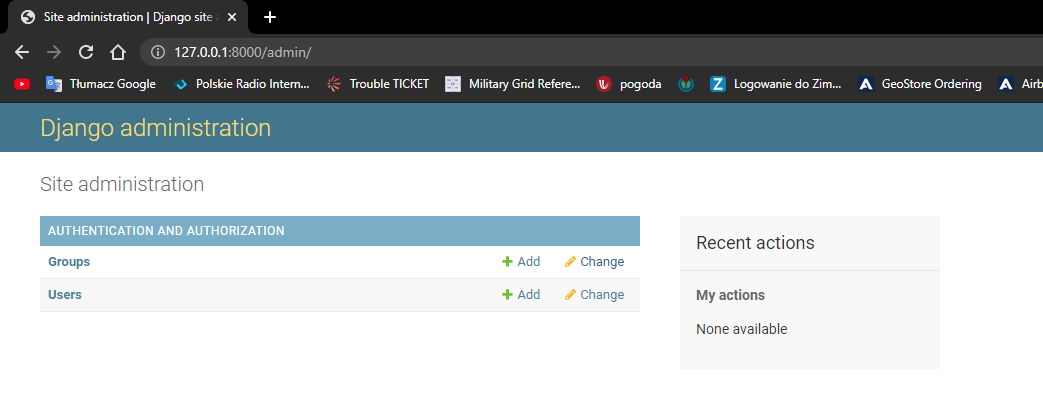
Mail: [admin@admin.com](mailto:admin@admin.com)

Password: admin

Przechodzimy do <http://127.0.0.1:8000/admin/>:



I logujemy się wcześniej wprowadzonymi credentialami. Tak wyglada nasze admin area:



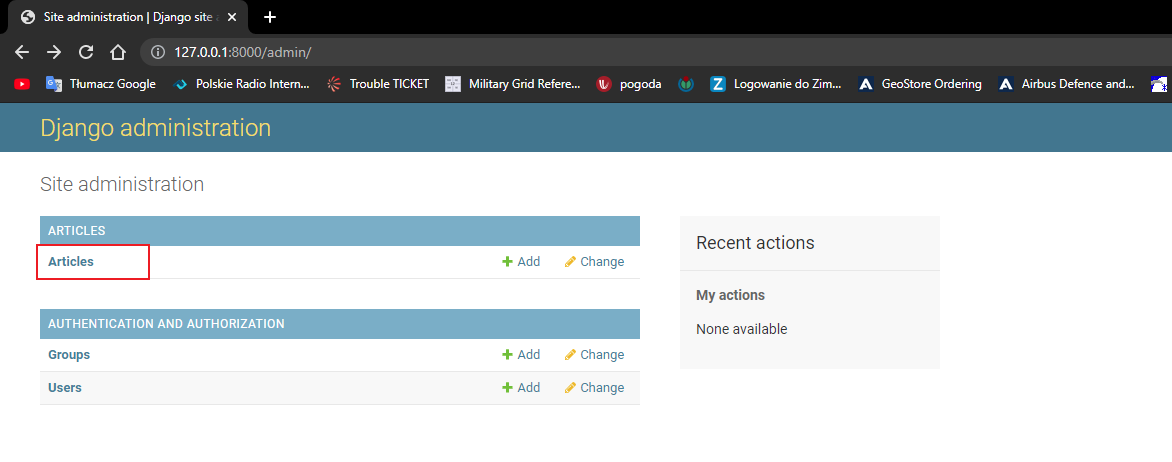
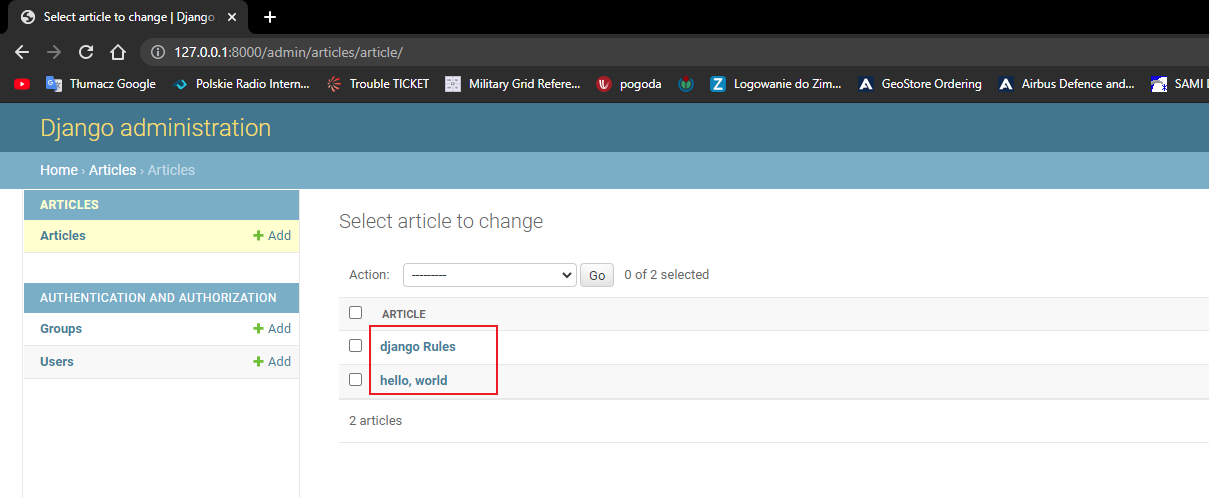
Chcemy żeby pojawił się nasz moduł Articles. W tym celu przechodzimy do pliku admin.py w katalogu articles i rejestrujemy w tym pliku moduł Articles: 

*from .models import Article*

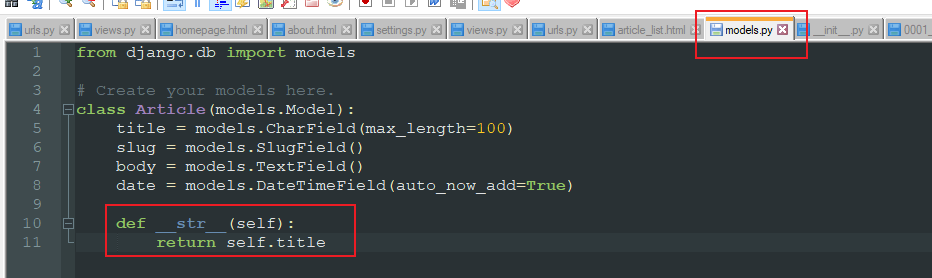
*# Register your models here.*

*admin.site.register(Article)*

Po kliknięciu w Aricles na stronie administracyjnej widzimy nasze artykuły:

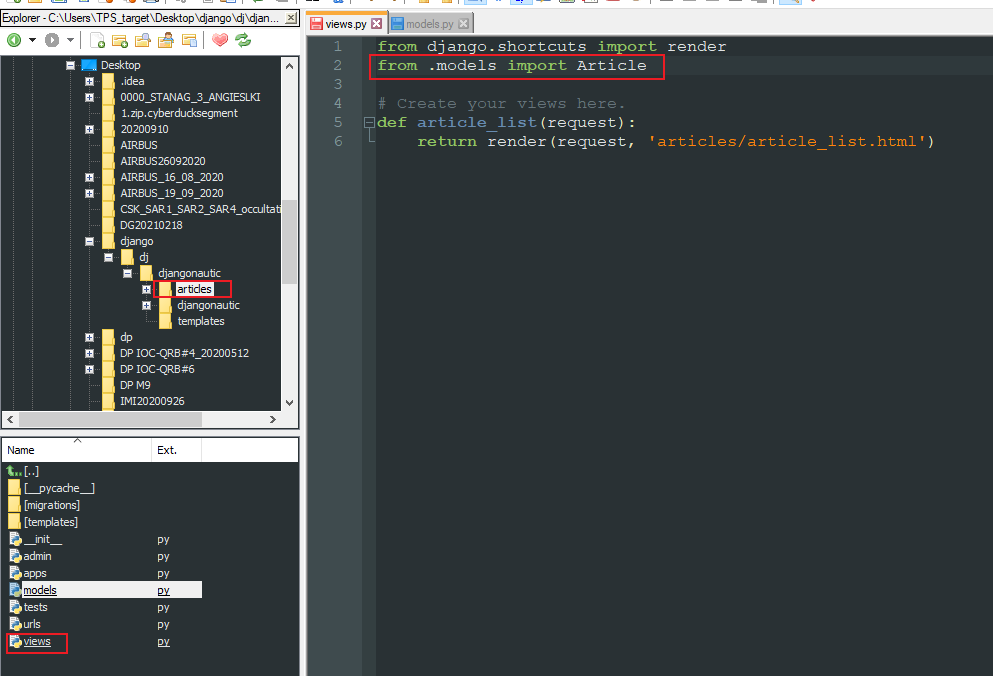
Widzimy ich tytuły bo wcześniej w pliku models.py skonfigurowaliśmy, że to właśnie tytuły mają się wyświetlać:



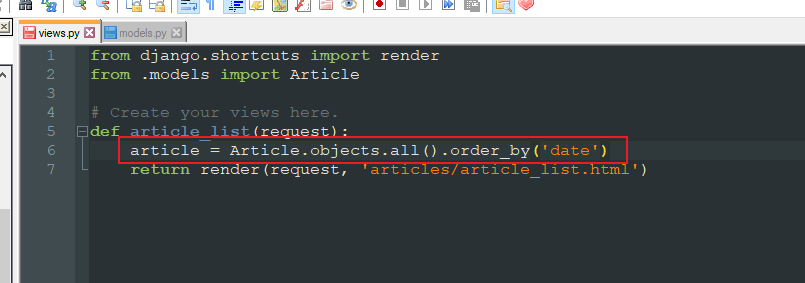
Odcinek 10:

Template tags. Po wejściu na stronę articles mają nam się wyświetlić wszystkie artykuły.

W pliku views aplikacji Articles importujemy metodę Article z pliku models:

 *from .models import Article*

Tworzymy zmienną articles (na screenie jest błędna nazwa) w którą wrzycimy wszytskie rekordy z tabeli Article. Dodatkowo robimy sortowanie:



*from django.shortcuts import render*

*from .models import Article*

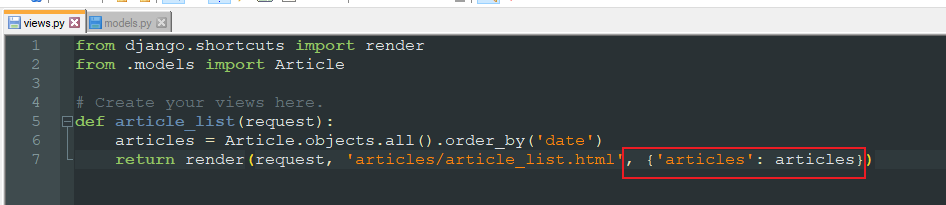
*# Create your views here.*

*def article\_list(request):*

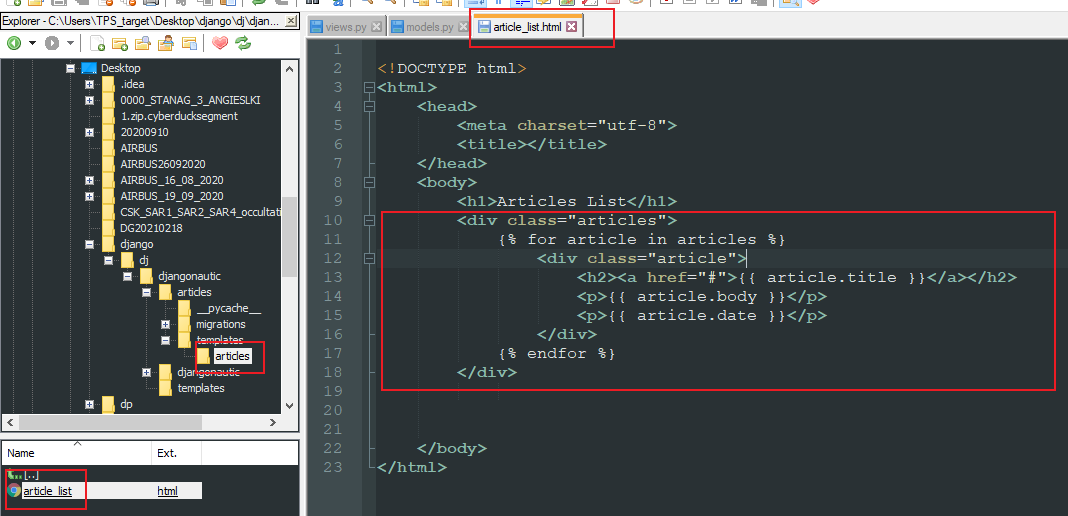
*articles = Article.objects.all().order\_by('date')*

*return render(request, 'articles/article\_list.html', {'articles': articles})*

Teraz do templatki article\_list.html przekazujemy zawartość zmiennej articles:



Przechodzimy do templatki article\_list.html i okodowujemy to co jest przekazywane w zmiennej articles. W pętli for mielimy wszystkie artykuły i wyświetlamy ich nagłówek (title) i treść (body):



*<div class="articles">*

*{% for article in articles %}*

*<div class="article">*

*<h2><a href="#">{{ article.title }}</a></h2>*

*<p>{{ article.body }}</p>*

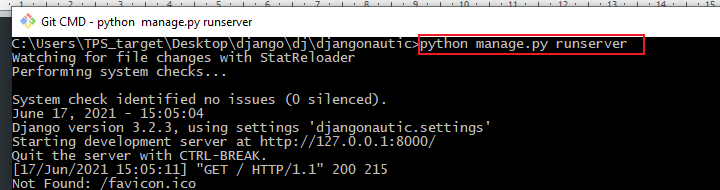
*<p>{{ article.date }}</p>*

*</div>*

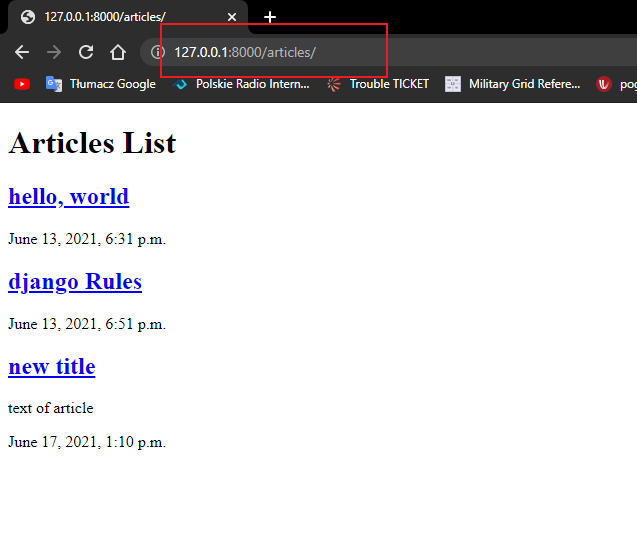
*{% endfor %}*

*</div>*

Jeśli server nie jest odpalony to go odpalamy:



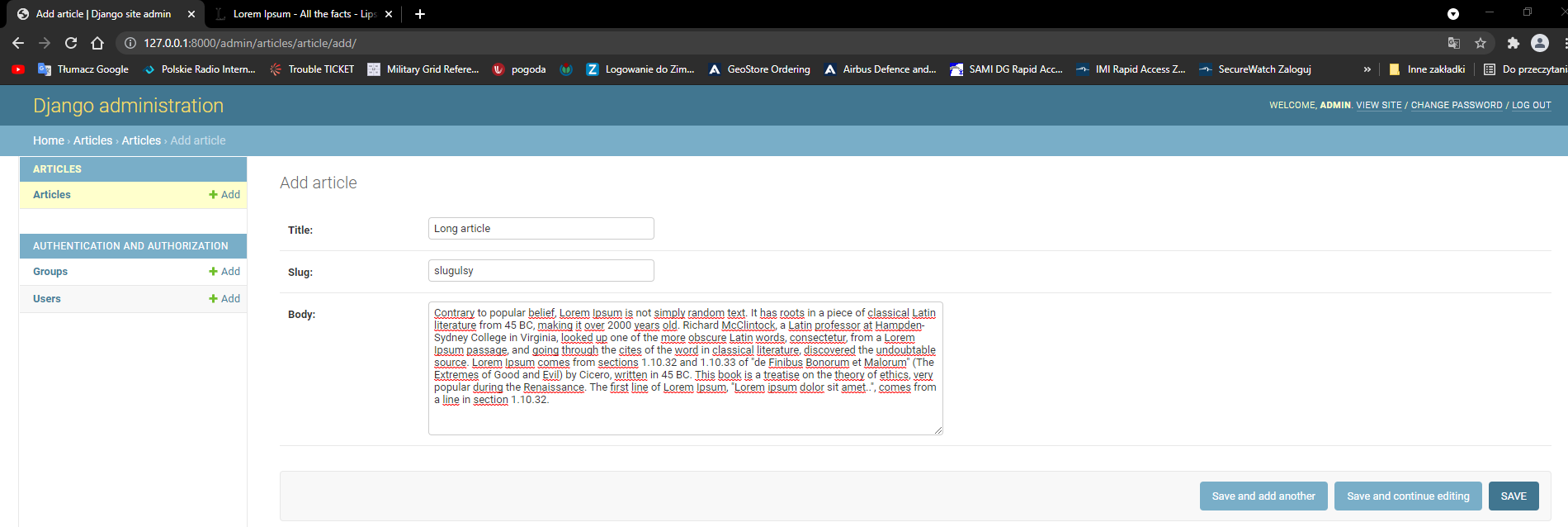
I przechodzimy do strony articles żeby zobaczyć czy widzimy nasze artykuły:



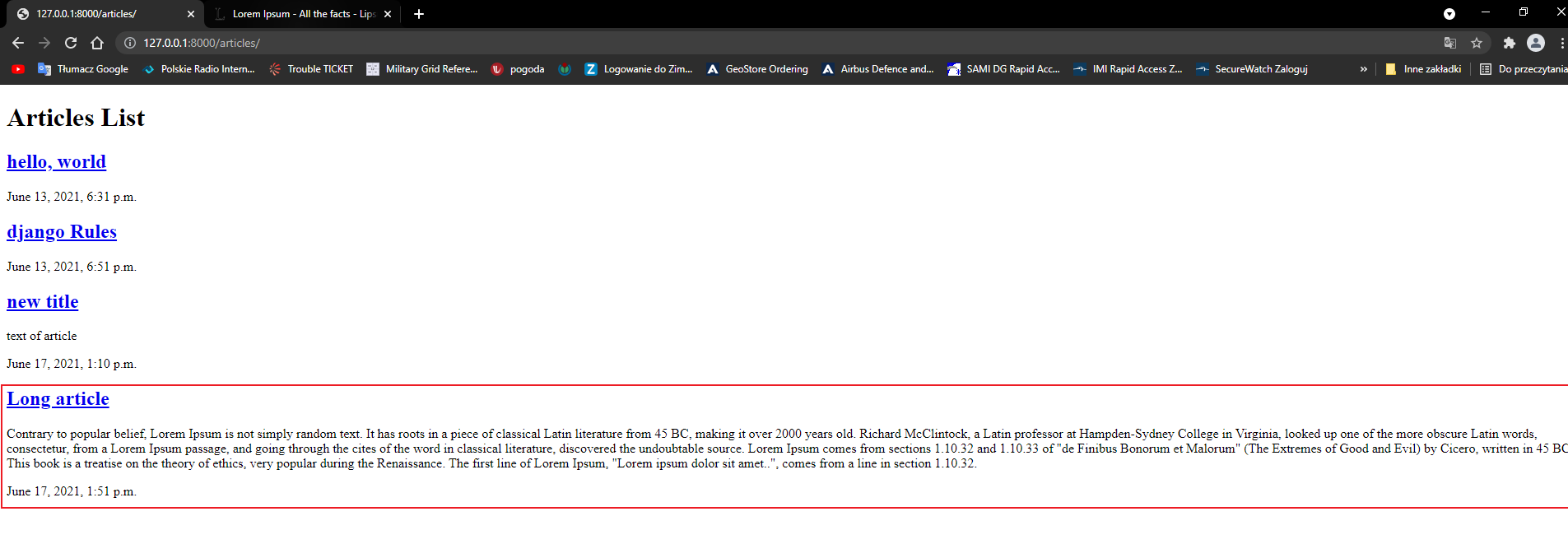
Odcinek 11:

Metoda w modelu.

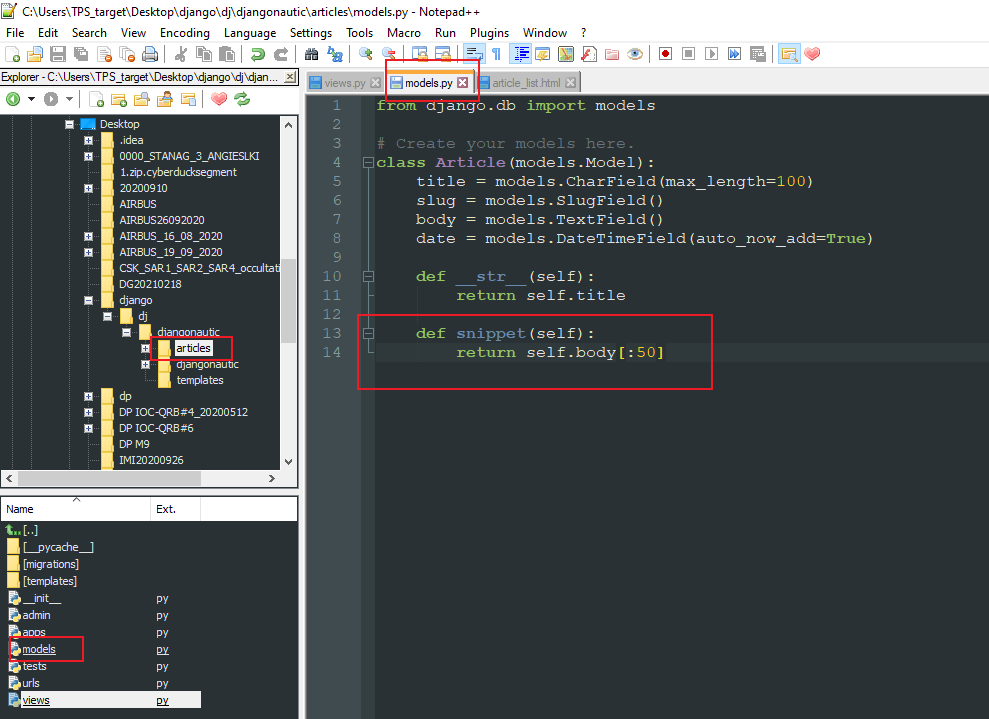
Przechodzimy do <http://127.0.0.1:8000/admin> i tworzymy nowy artykuł, który w body będzie miał długi tekst:



Przechodzimy do artykułów i widzimy, że nasz artykuł jest wyświetlony w całości zaś my chcemy, żeby wyświetlało się tutaj jedynie kilkanaście początkowych znaków:



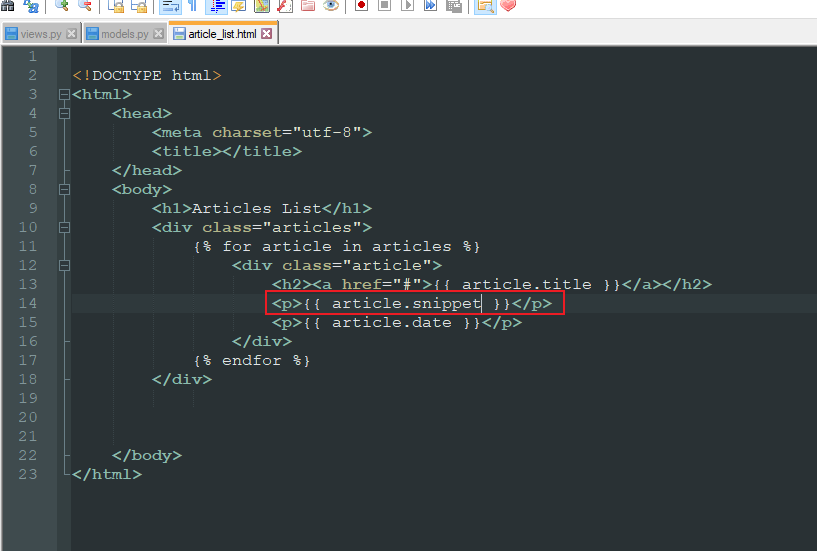
W tym celu w models tworzymy funkcję, która będzie wyłapywała jedynie 50 początkowych znaków w body:



*def snippet(self):*

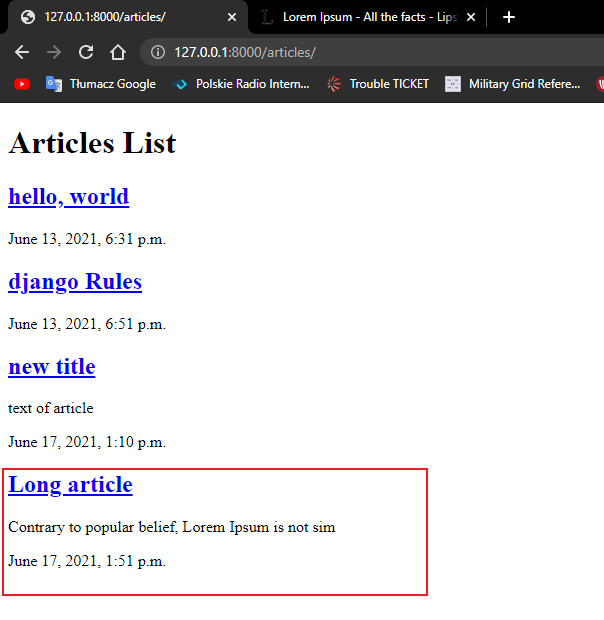
*return self.body[:50]*

Następnie w article\_list.html dokonamy modyfikacji w taki sposób żeby nie wyświetlało nam się już całe body tak jak do tej pory a jedynie to co zwraca funkcja snippet:

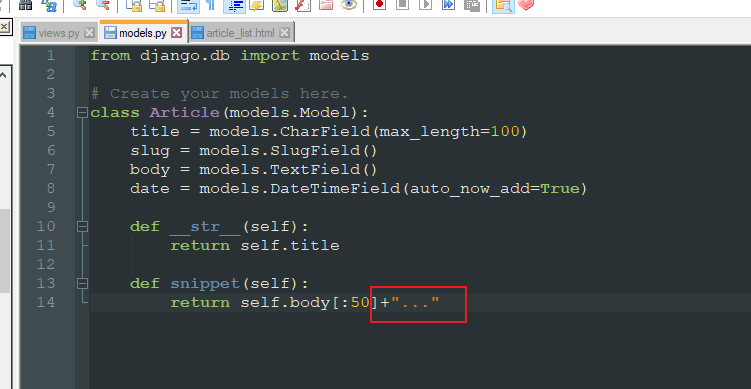


*<p>{{ article.snippet }}</p>*

Artykuł wyświetla się w porządany sposób:

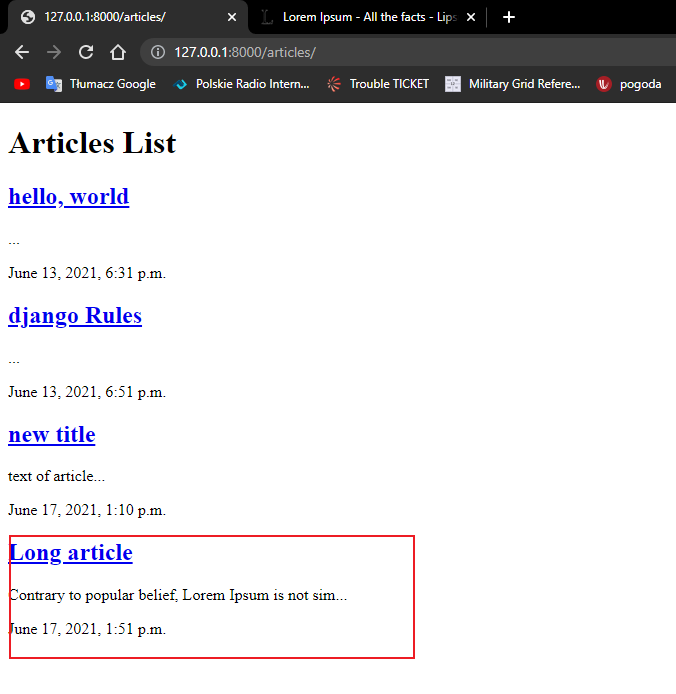


Teraz dodamy trzy kropki na końcu body. Dokonujemy modyfikacji w models.py:



*return self.body[:50]+"..."*

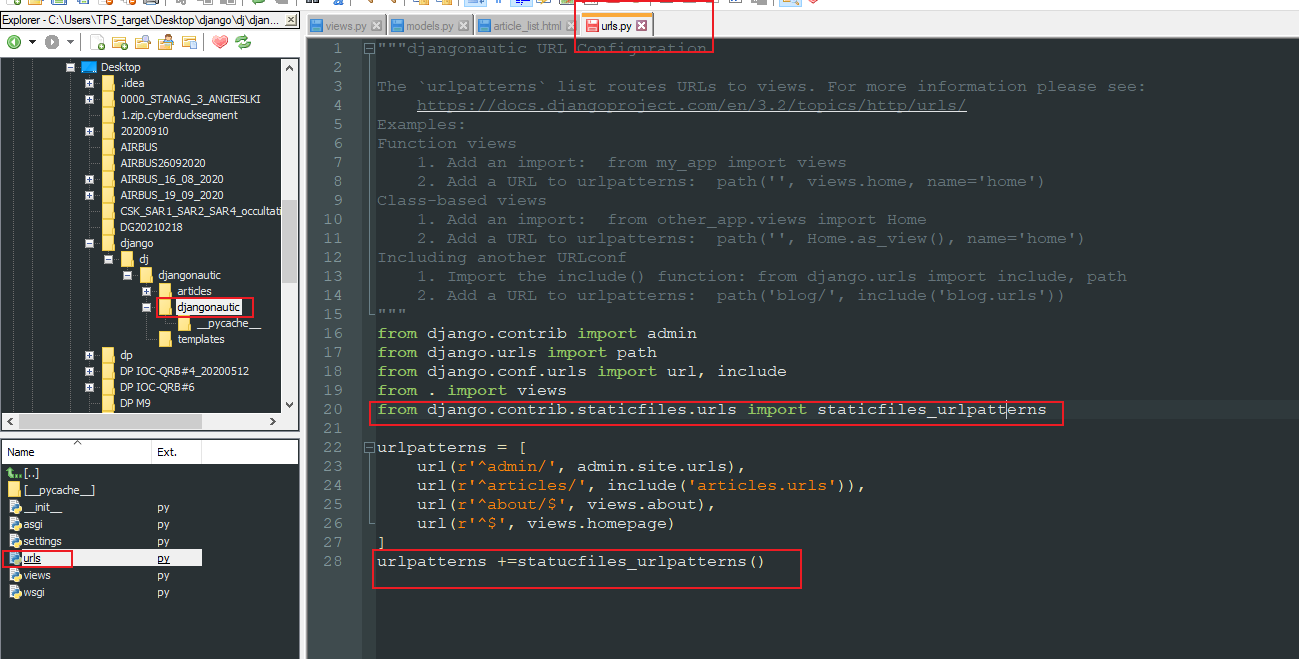
Sprawdzamy stronę:



Odcinek 12:

Static files. Obsługa plików takich jak zdjęcia, pliki js...

Przechodzmy do urls w aplikacji djangonautic i uzupełniamy plik o następujący kod:



from django.contrib import admin

*from django.urls import path*

*from django.conf.urls import url, include*

*from . import views*

*from django.contrib.staticfiles.urls import staticfiles\_urlpatterns*

*urlpatterns = [*

*url(r'^admin/', admin.site.urls),*

*url(r'^articles/', include('articles.urls')),*

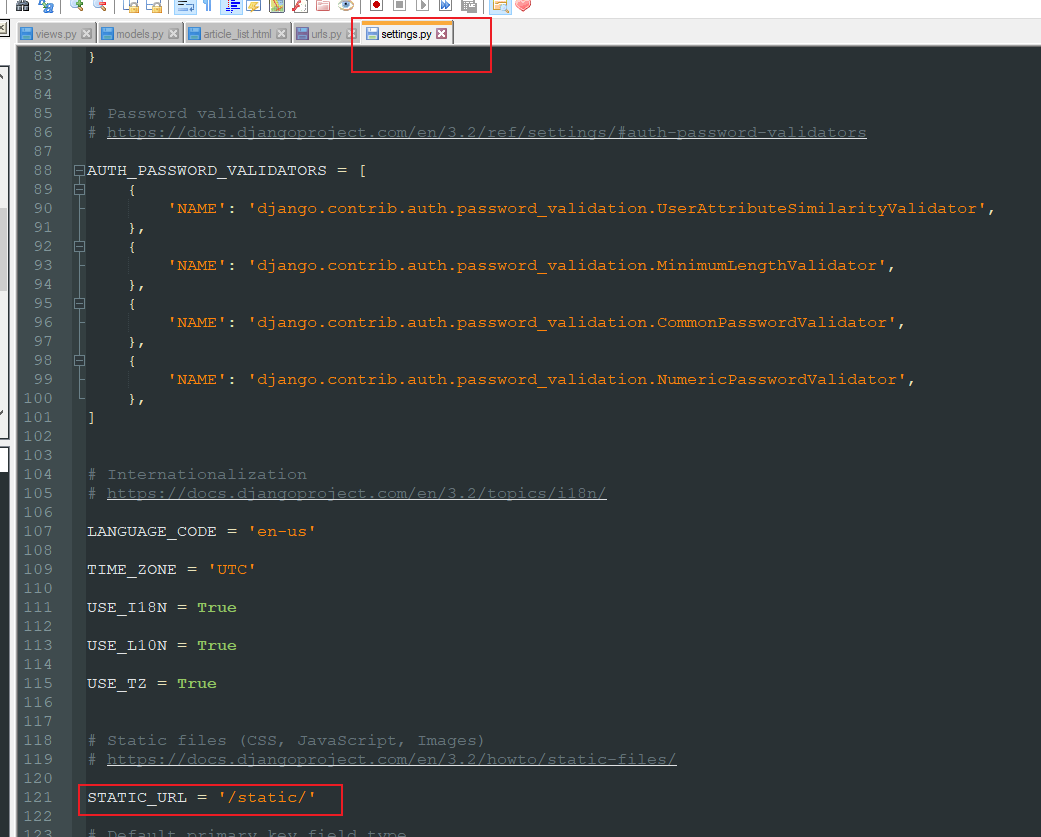
*url(r'^about/$', views.about),*

*url(r'^$', views.homepage)*

*]*

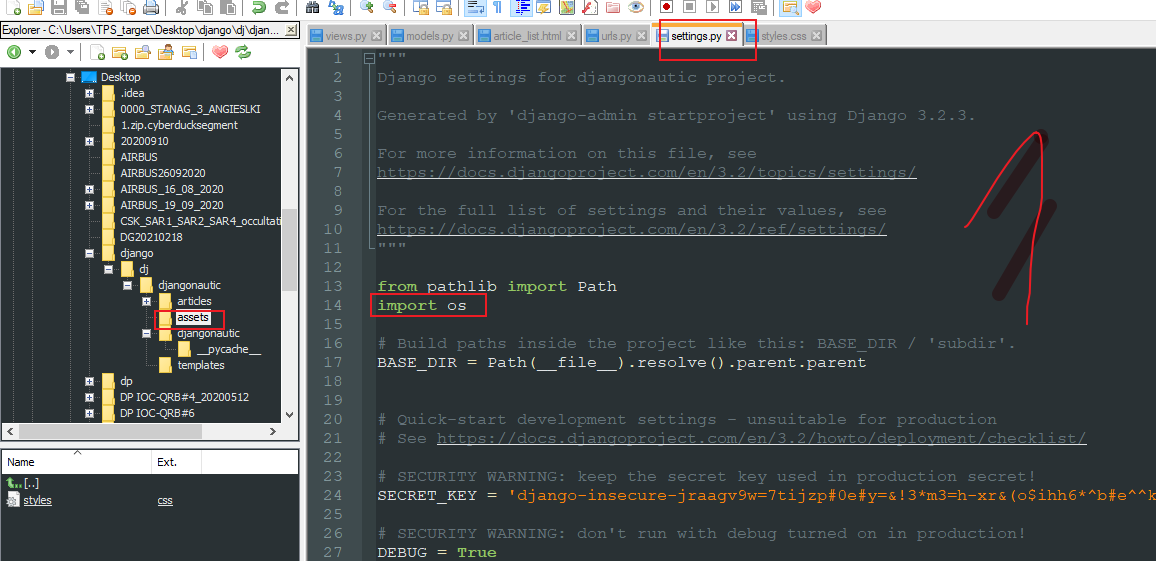
*urlpatterns +=statucfiles\_urlpatterns()*

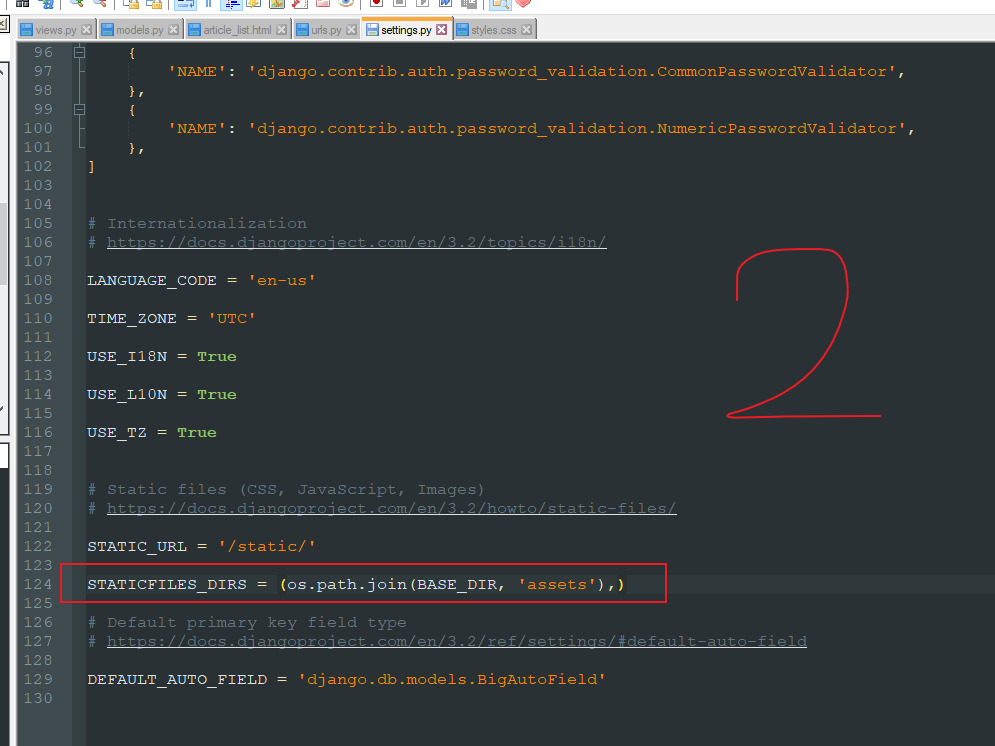
Lokalizacja plików statycznych jest ustalona w pliku settings.py:



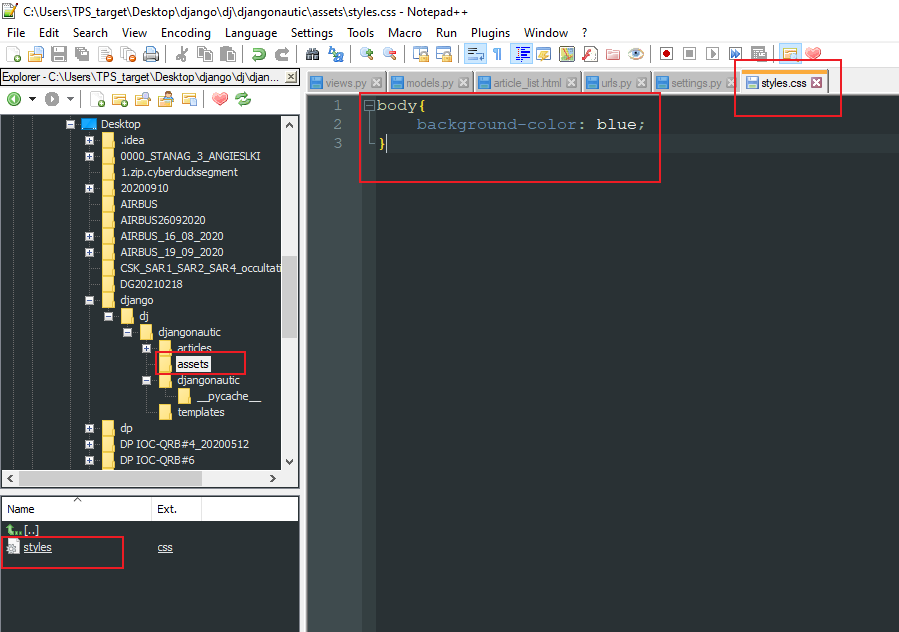
To tutaj python będzie ich szukał.

Nasze pliki umieścimy w katalogu assets i uzupełnimy plik settings następująco:





Następnie w katalogo assets umiescimy plik css o następującej zawartości:

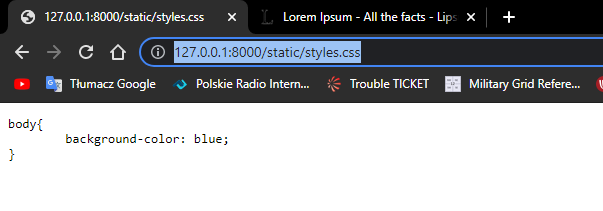


*body{*

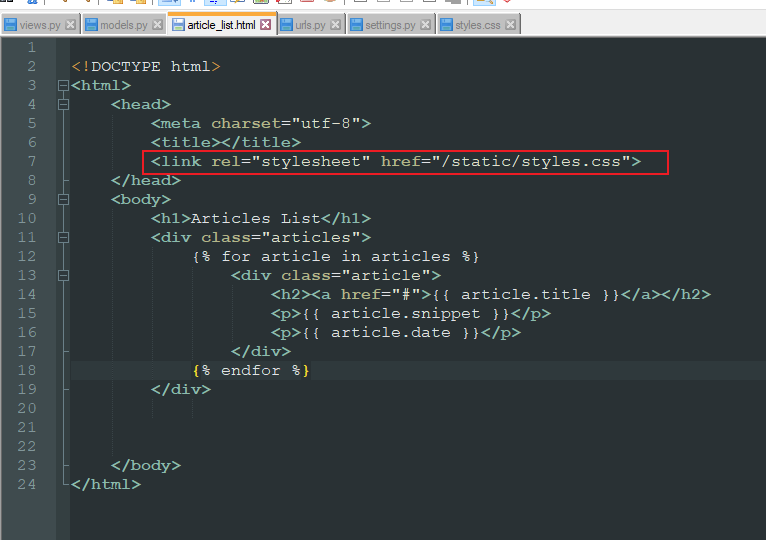
*background-color: blue;*

*}*

Wchodzimy na adres <http://127.0.0.1:8000/static/styles.css> i sprawdzamy czy wyświetla nam się zawartość pliku css:



Teraz połączymy nasz plik statyczny css z article\_template.html. W tym celu przechodzimy do pliku article\_list.html i uzupełniamy go o następującą linijkę:

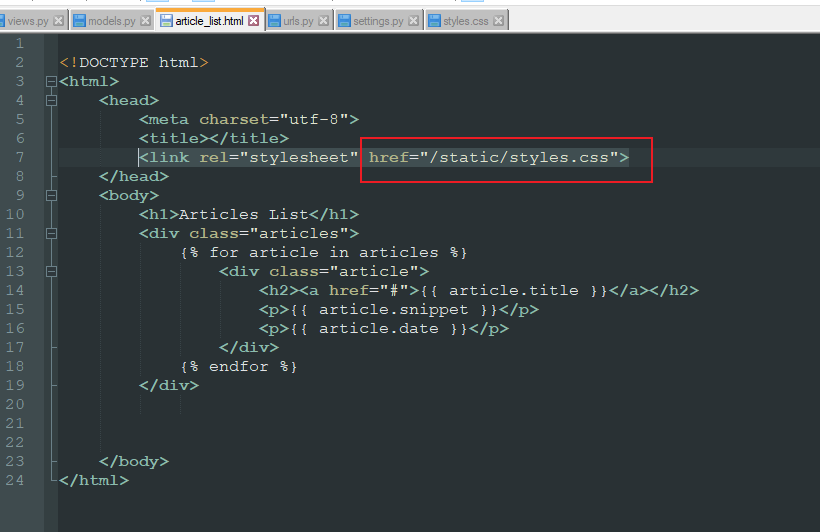


*<link rel="stylesheet" href="/static/styles.css">*

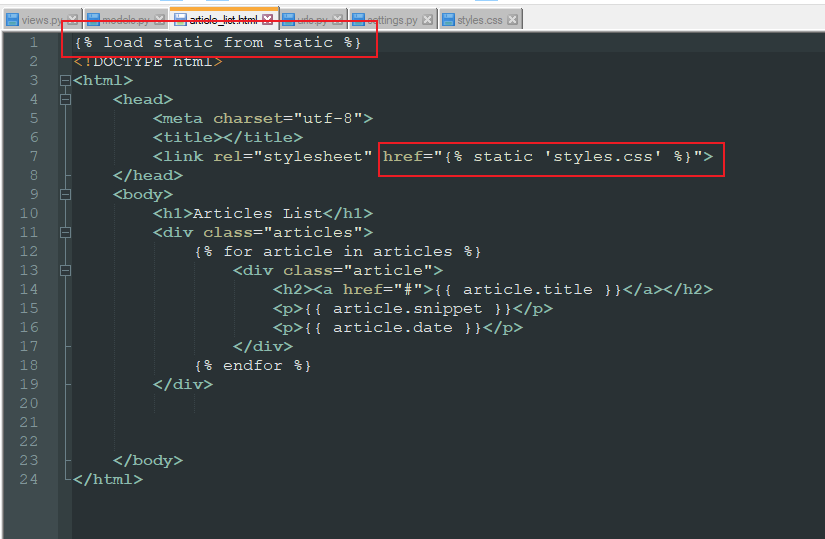
Przechodzimy co <http://127.0.0.1:8000/articles/> I sprawdzamy czy kolor tła został ustawiony:



W tym momencie mamy zahardkodowaną ścieżkę do pliku styles.css:



Jeśli ścieżka do plików statycznych się zmieni to trzeba też będzie ręcznie w templatkach to zmieniać. A możemy zamiast tego zrobić dynamicznie zmieniającą się ścieżkę. Uzupełniamy templatkę o następujący kod:



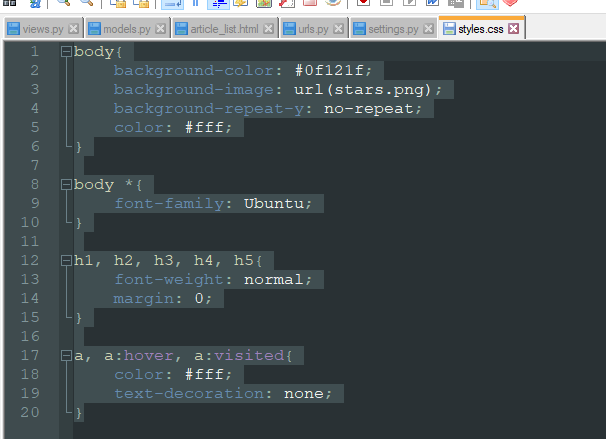
*{% load static from static %}*

*<link rel="stylesheet" href="{% static 'styles.css' %}">*

Zmieniłem kolor tła na zielony zeby sprawdzić czy dynamiczny adres działa:



Podmieniamy kod w pliku css na następujący:



*body{*

*background-color: #0f121f;*

*background-image: url(stars.png);*

*background-repeat-y: no-repeat;*

*color: #fff;*

*}*

*body \*{*

*font-family: Ubuntu;*

*}*

*h1, h2, h3, h4, h5{*

*font-weight: normal;*

*margin: 0;*

*}*

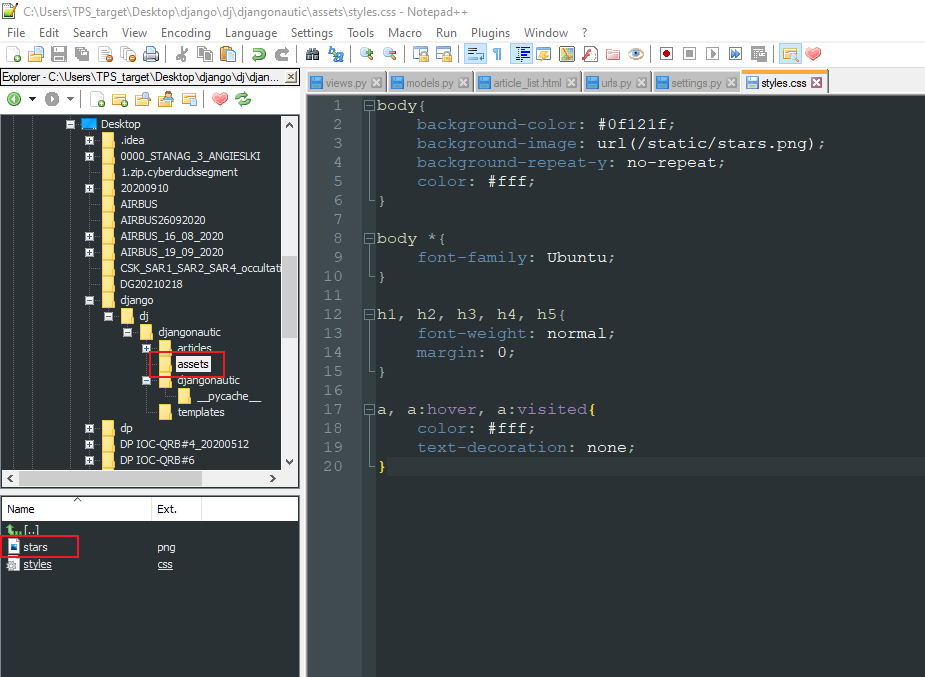
*a, a:hover, a:visited{*

*color: #fff;*

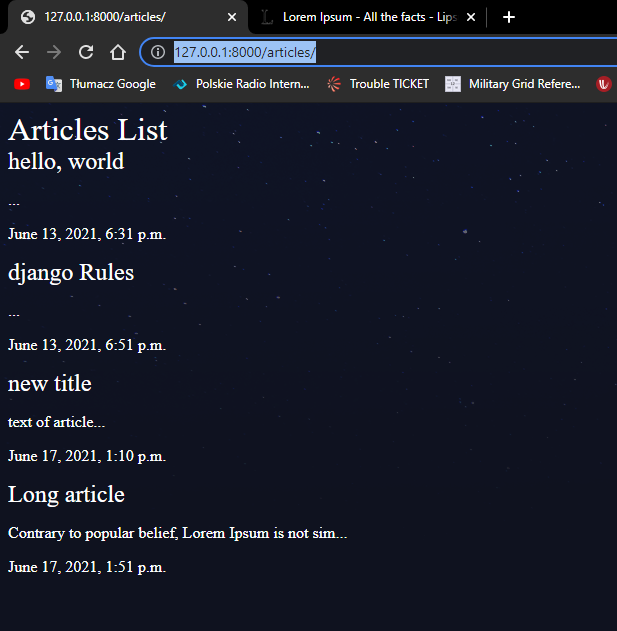
*text-decoration: none;*

*}*

Css korzysta z pliku stars.png. Plik umieszczamy w katalogu assets:



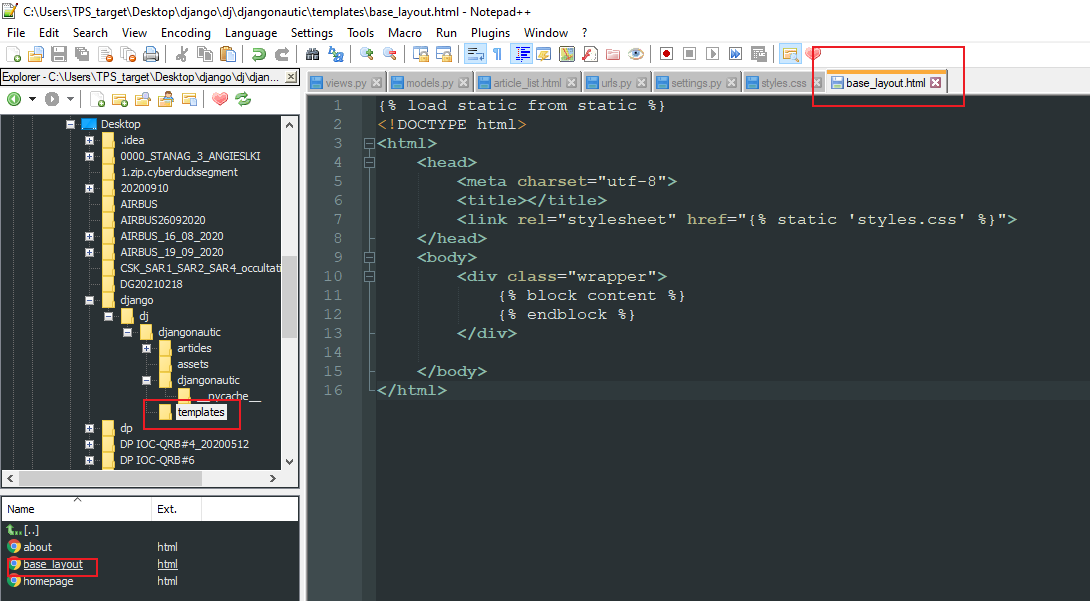
I sprawdzamy czy strona articles (http://127.0.0.1:8000/articles/) działa jak należy:



Odcinek 13:

Extending templates.

W przypadku gdybyśmy mieli wiele templatek to w każdej z nich powtarzał by się ten sam header i inne rzeczy. Możemy zrobić globalną templatkę bazową z której kędą korzystały wszystkie inne i uniknąć tej redundancji. Tworzymy w katalogu templates plik base\_layout.html:



*{% load static from static %}*

*<!DOCTYPE html>*

*<html>*

*<head>*

*<meta charset="utf-8">*

*<title></title>*

*<link rel="stylesheet" href="{% static 'styles.css' %}">*

*</head>*

*<body>*

*<div class="wrapper">*

*{% block content %}*

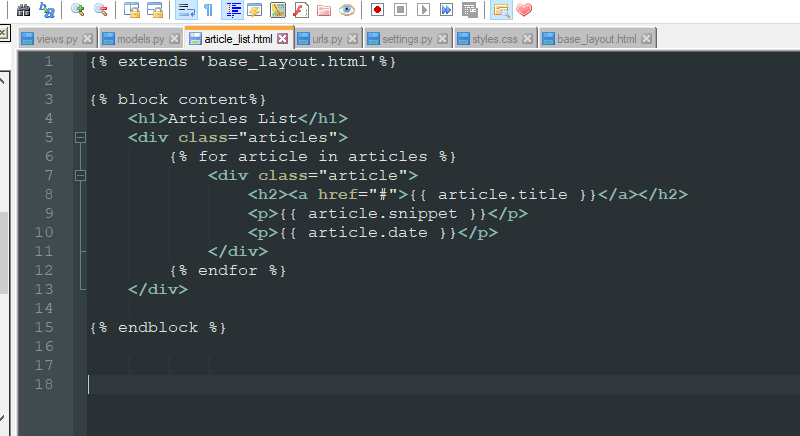
*{% endblock %}*

*</div>*

*</body>*

*</html>*

Skoro mamy już bazową templatkę to możemy teraz zmodyfikować odpowiednio templatkę article\_list:



*{% extends 'base\_layout.html'%}*

*{% block content%}*

*<h1>Articles List</h1>*

*<div class="articles">*

*{% for article in articles %}*

*<div class="article">*

*<h2><a href="#">{{ article.title }}</a></h2>*

*<p>{{ article.snippet }}</p>*

*<p>{{ article.date }}</p>*

*</div>*

*{% endfor %}*

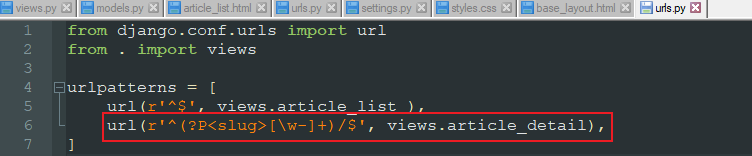
*</div>*

*{% endblock %}*

Odcinek 14:

Będziemy robić przechodzenie do artykułu po kliknięciu na niego na stronie article list.

Przechodziy do pliku urls.py w katalogu articles i uzupełniamy go następująco:



*from django.conf.urls import url*

*from . import views*

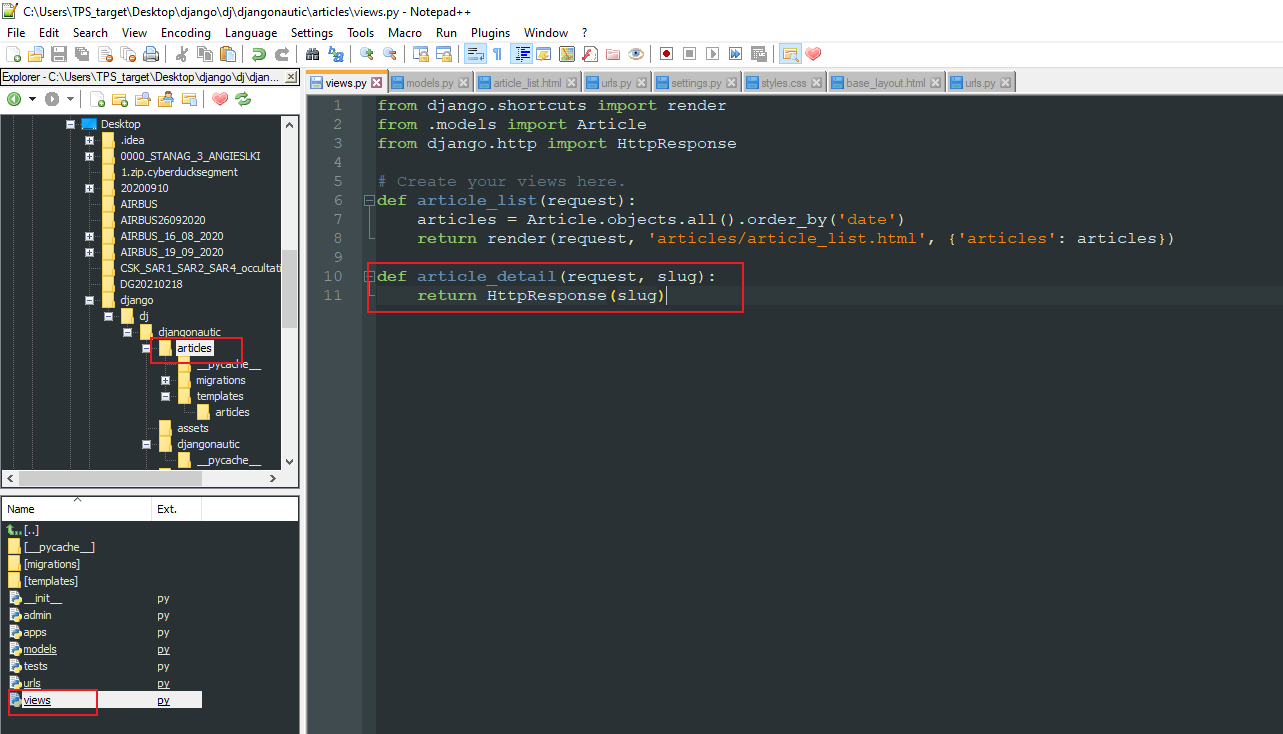
*urlpatterns = [*

*url(r'^$', views.article\_list ),*

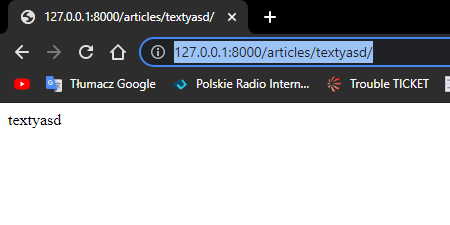
*url(r'^(?P<slug>[\w-]+)/$', views.article\_detail),*

*]*

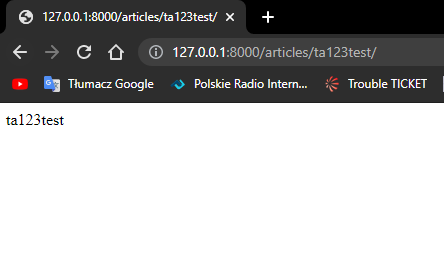
Wygląda to tak, że do funkcji article\_detail znajdującej się w views podajemy teraz zmienną slug. Ta zmienna przyjmie wartość tego co napiszemy w linku. Obok niej są dziwne znaczki żeby mogła znaleźć się tam liczba, litera duża/mała w dowolnej liczbie. Teraz przechodzimy do views i tworzymy funkcję, która będzie wywoływana:



Wpisujemy w adres przeglądarki <http://127.0.0.1:8000/articles/textyasd/>



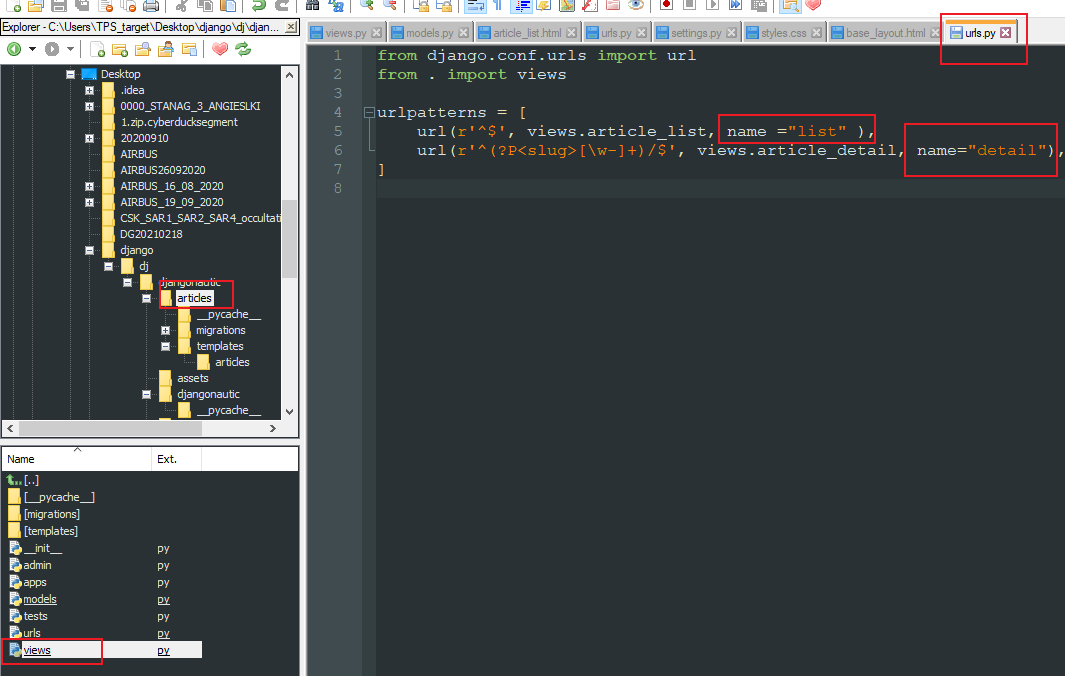
http://127.0.0.1:8000/articles/ta123test/



Odcinek 15:

Sprawiamy żeby po kliknięciu w artykuł przekierowywało nas na odpowiednią stronę. Najpierw wszedłem w moduł admina i zmieniłem nazwy slugów dla wszystkich artykułów nazwyając je po kolei slug1, slug2...

Dodajemy parametr w pliku urls.py:



*from django.conf.urls import url*

*from . import views*

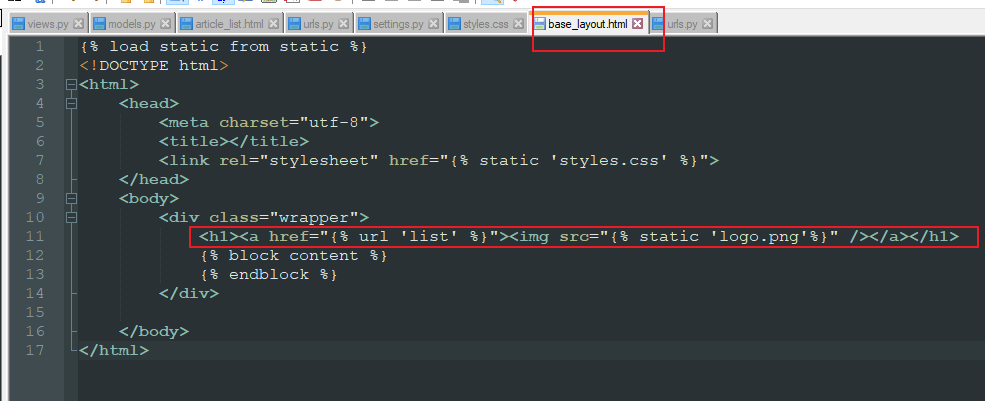
*urlpatterns = [*

*url(r'^$', views.article\_list, name ="list" ),*

*url(r'^(?P<slug>[\w-]+)/$', views.article\_detail, name="detail"),*

*]*

I modyfikujemy base\_layout.html:



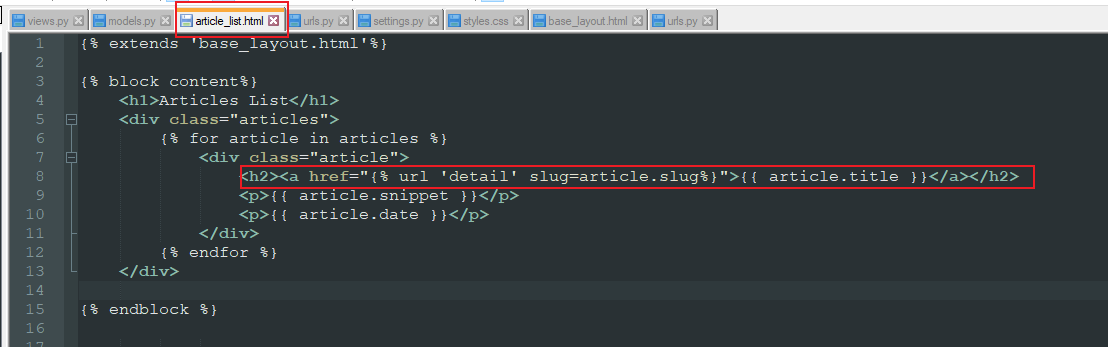
*<h1><a href="{% url 'list' %}"><img src="{% static 'logo.png'%}" /></a></h1>*

Plik logo.png umieszczamy w katalogu assets:



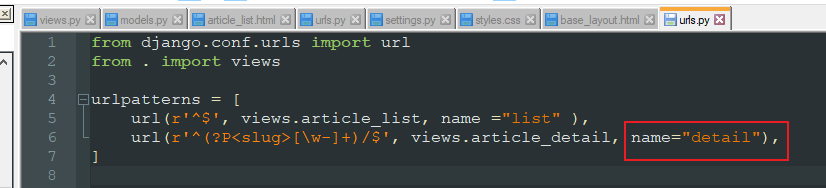
Po wejściu na stronę articles pojawia nam się logo, które po kliknięciu przekierowuje nas na stronę articles.

Teraz ogarniamy przekierowywanie po kliknięciu na artykuł. W tym celu w pliku article\_list.html wpisujemy:

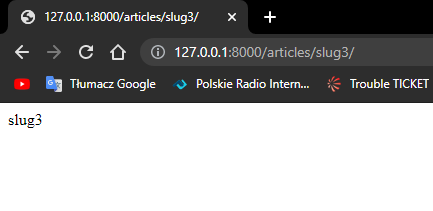


*<h2><a href="{% url 'detail' slug=article.slug%}">{{ article.title }}</a></h2>*

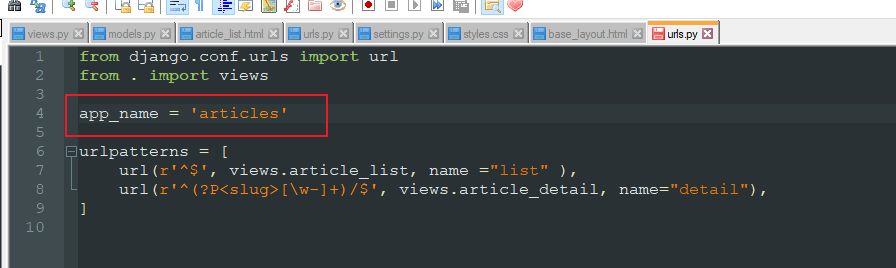
Czyli z url o nazwie detail:



Bierze sobie to co siedzi w slugu. I po kliknięciu w artykuł rzeczywiście przekierowywuje:

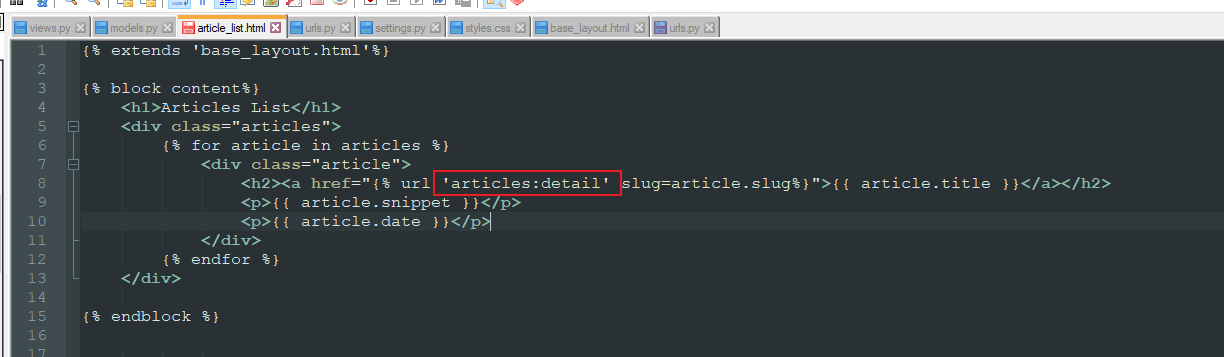


W momencie gdy wiele aplikacji miałoby url o nazwie np „detail” django nie wiedziałoby do urla z której apki się odwołujemy. Dlatego modyfikujemy plik url.py w naszej apce następująco:



*app\_name = 'articles'*

oraz article\_list.py następująco:



*<h2><a href="{% url 'articles:detail' slug=article.slug%}">{{ article.title }}</a></h2>*

Analogicznie w base\_layout musimy wskazać o którego urla nam chodzi:

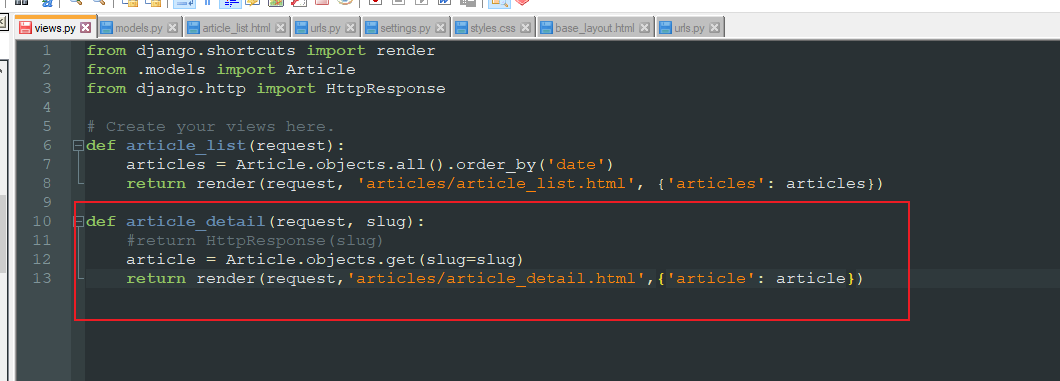


*<h1><a href="{% url 'articles:list' %}"><img src="{% static 'logo.png'%}" /></a></h1>*

Odcinek 16:

Article detail page. Teraz po kliknięciu w artykuł przechodzimy na stronę na której jest napis taki jak końcówka adresu. Teraz chcemy zrobic tak, żeby po kliknięciu na artykuł przeszukiwana była baza danych w poszukiwaniu odpowiedniego artykułu.

Aby to osiągnąć przechodzimy do pliku views.py i modyfikujemy odpowiednio funkcję article\_detail:



*def article\_detail(request, slug):*

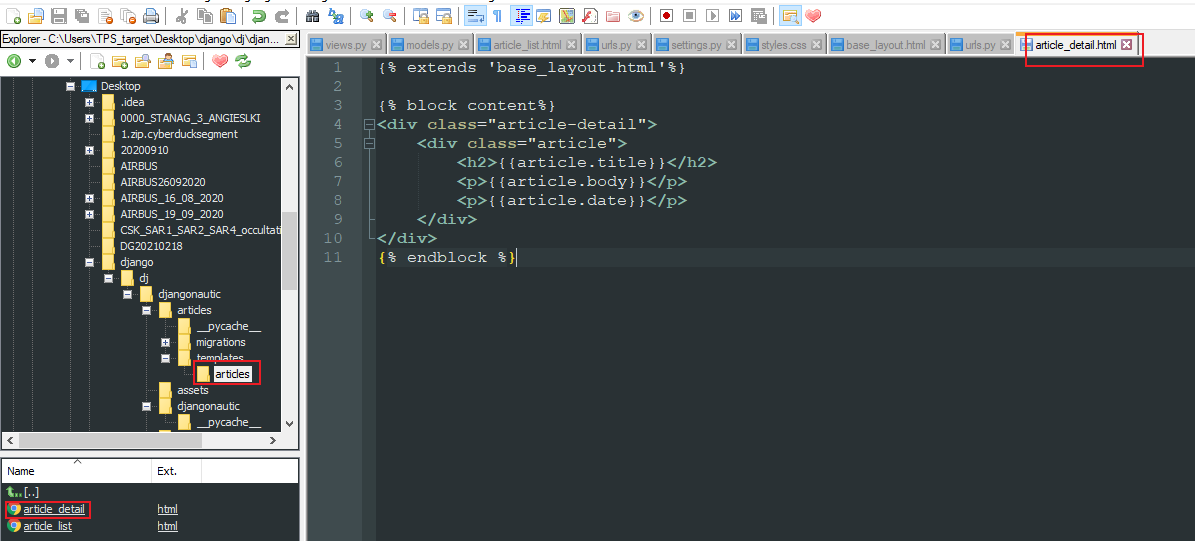
*#return HttpResponse(slug)*

*article = Article.objects.get(slug=slug)*

*return render(request,'articles/article\_detail.html',{'article': article})*

W zmiennej article umieszczamy wyszukany po slugu artykuł. Następnie renderujemy templatkę article\_detail, którą wkrótce utworzymy, przekazując do niej zmienną article.

Tworzymy templatkę article\_detail.html:



*{% extends 'base\_layout.html'%}*

*{% block content%}*

*<div class="article-detail">*

*<div class="article">*

*<h2>{{article.title}}</h2>*

*<p>{{article.body}}</p>*

*<p>{{article.date}}</p>*

*</div>*

*</div>*

*{% endblock %}*