**Django - szkolenie 6**

**Profil użytkownika**

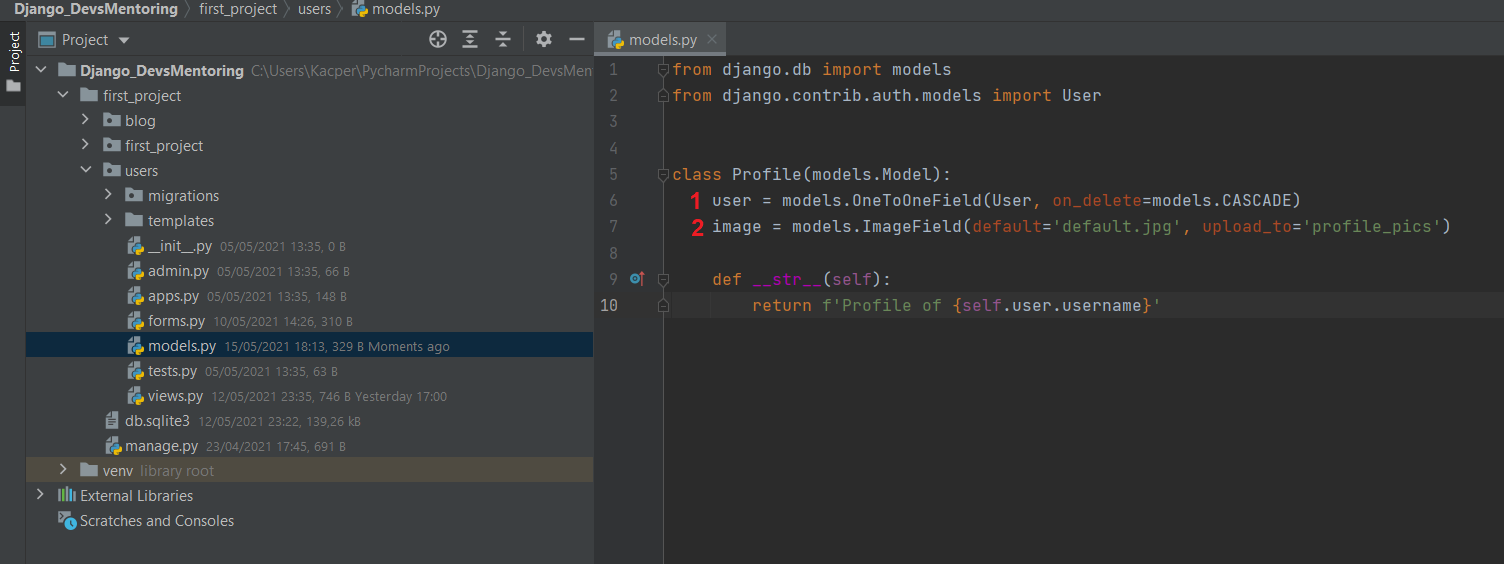
**Wstęp**

Przyszedł czas na rozszerzenie Naszej aplikacji przez stworzenie indywidualnego profilu dla każdego z użytkowników. W ramach takiego profilu, pojedynczy użytkownik będzie mógł uploadować własny awatary i modyfikować podstawowe informacje. Brzmi na dość skomplikowaną funkcjonalność, ale jak się przekonasz realizacja tego w Django nie jest wcale tak złożona. Nie owijając w bawełnę, przystąpmy zatem do działania!

**Utworzenie nowej tabeli - Profile**

Proces developmentu rozpoczniemy od utworzenia nowej tabeli Profile w ramach istniejącej aplikacji users. Będzie to tabela odwołująca się do zarejestrowanych użytkowników oraz będziemy przechowywać w niej informacje ściśle powiązane z profilem użytkownika, jak np. ścieżka katalogowa do obrazka (przechowywanego po stronie serwera), który dany użytkownik wykorzystuje dla swojego awatara.

Przejdźmy zatem do pliku models.py w katalogu users i uzupełnijmy go następującym kodem:



from django.db import models

from django.contrib.auth.models import User

class Profile(models.Model):

user = models.OneToOneField(User, on\_delete=models.CASCADE)

image = models.ImageField(default='default.jpg', upload\_to='profile\_pics')

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

return f'Profile of {self.user.username}'

Zwróć uwagę na umieszczone pola (1) i (2). User jest kluczem obcym definiującym, do którego użytkownika odnosi się dany profil. Image natomiast to pole będące obiektem ImageField. Do konstruktora takiego obiektu, przekazujemy dwa argumenty: default - czyli nazwę domyślnego obrazka, który ma być pobrany, aby został powiązany z danym profilem użytkownika, a upload\_to jest nazwą katalogu, w którym aplikacja Django trzymać będzie wszystkie awatary użytkowników.

W klasie Profile umieściliśmy również metodę specjalną \_\_str\_\_, aby móc łatwo formatować jej obiekty i wyświetlać w przejrzystej formie.

Przejdźmy do utworzenia tabeli Profile, wykorzystując poznanie uprzednio polecenie python manage.py makemigrations.

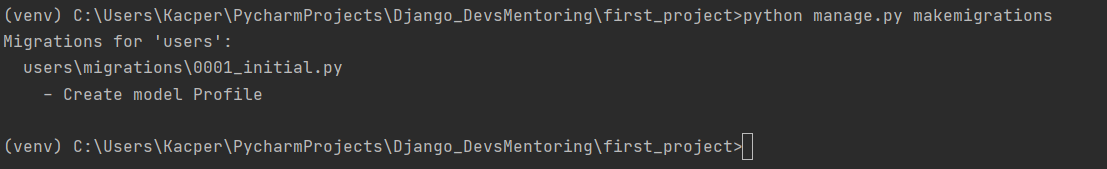


Po wykonaniu polecenia, napotkaliśmy jednak błąd taki jak powyżej. Okazuje się, że obiekt ImageField możemy utworzyć dopiero wtedy, gdy doinstalujemy moduł Pillow. Biblioteka ta służy do łatwego otwierania, zapisywania i sprawnej manipulacji plikami będącymi obrazami (z poziomu Pythona).

Bibliotekę tę zainstalujemy przez polecenie pip install Pillow.



Spróbujmy raz jeszcze wykonać polecenie python manage.py makemigrations.



Teraz już operacja zakończyła się powodzeniem. Przypomnę Ci tylko, że polecenie to umożliwia mapowanie obiektowego kodu bazy danych na relacyjnego SQL-a i potrzebne dependencje zapisuje w katalogu users/migrations.

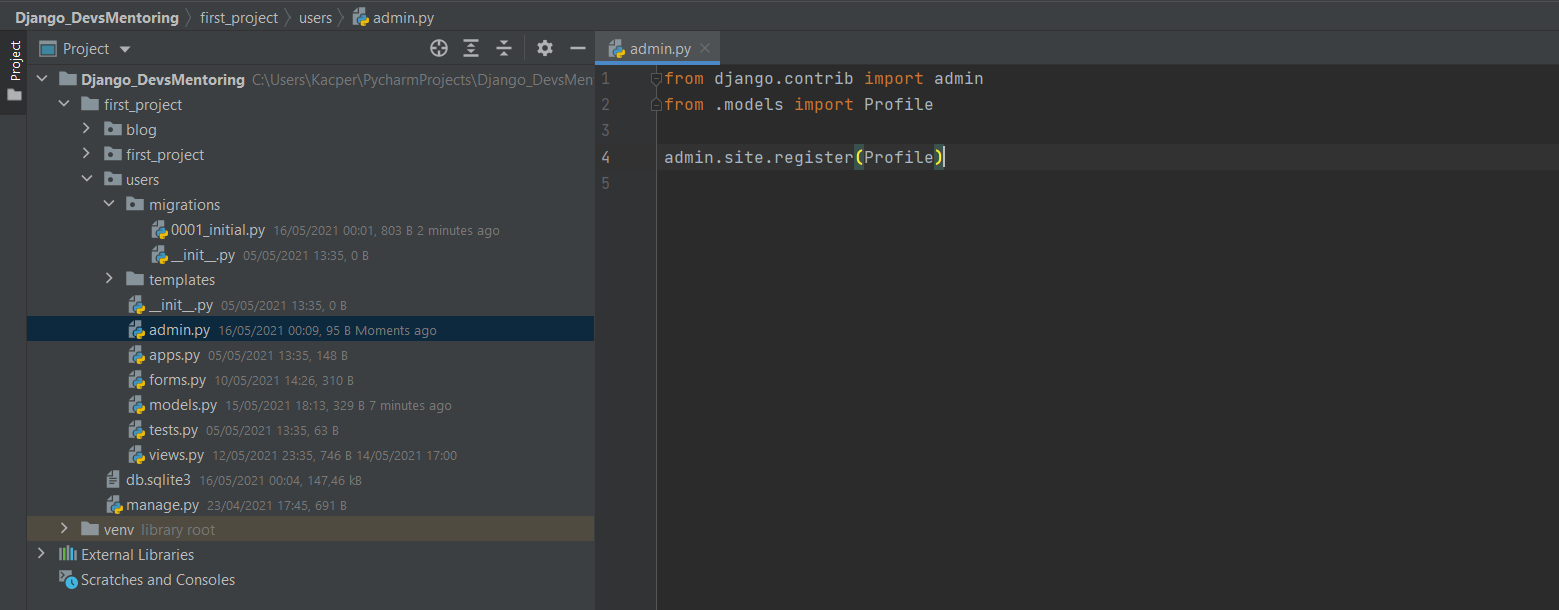
Od wygenerowania potrzebnej tabeli dzieli Nas jedno polecenie - python manage.py migrate. Spowoduje ono wykonanie kodu zapisanego pod users/migrations i w rezultacie utworzy tabelę Profile.



**Dodanie utworzonej tabeli do panelu admina**

Udało Nam się utworzyć tabelę Profile, a więc nie stoi nic na przeszkodzie, aby zacząć umieszczać w niej profile pierwszych użytkowników. W ramach przypomnienia - rekordy do tabel możemy umieszczać przez powłokę skryptową silnika Django (uruchamianą poleceniem python manage.py shell), jednak dużo wygodniej będzie dodawać nowe profile przez interfejs panelu administratora dostępnego pod endpointem /admin. Aby jednak móc “przeklikać” się przez tabelę Profile, musimy zarejestrować ją w pliku users/admin.py (co ponownie nie powinno być dla Ciebie nowością - mieliśmy już do czynienia z takim zabiegiem przy rejestrowania nowej tabeli Article).

W rezultacie, plik users/admin.py powinien wyglądać następująco:



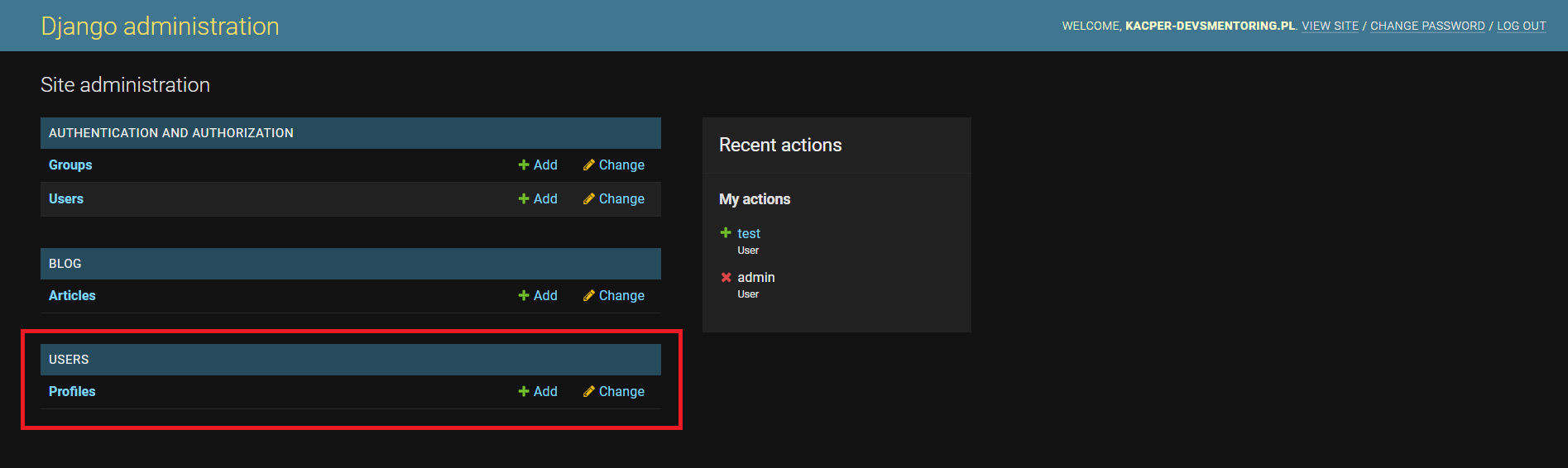
from django.contrib import admin

from .models import Profile

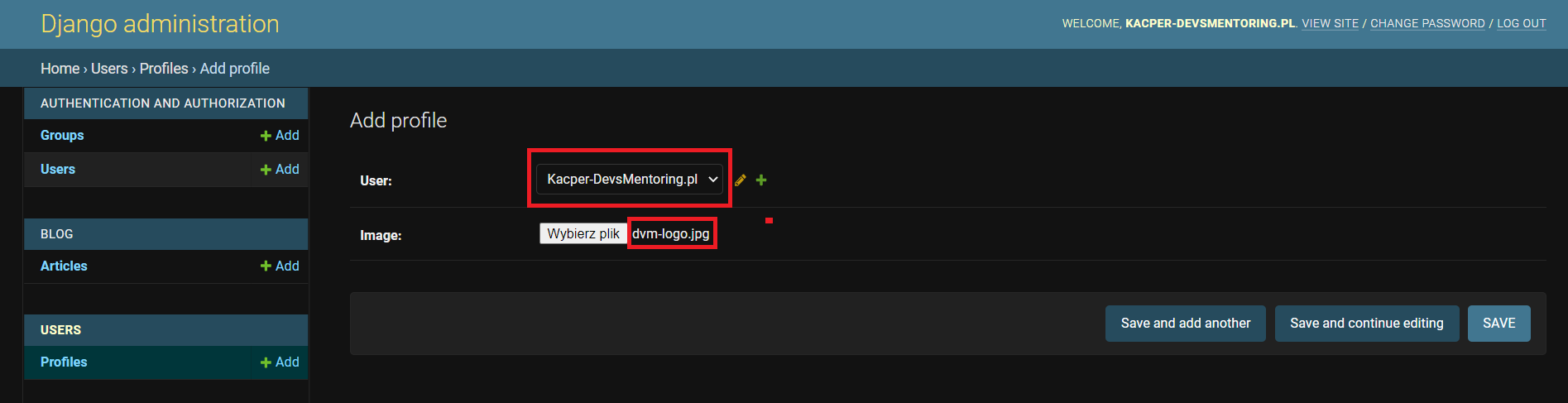
# Register your models here.

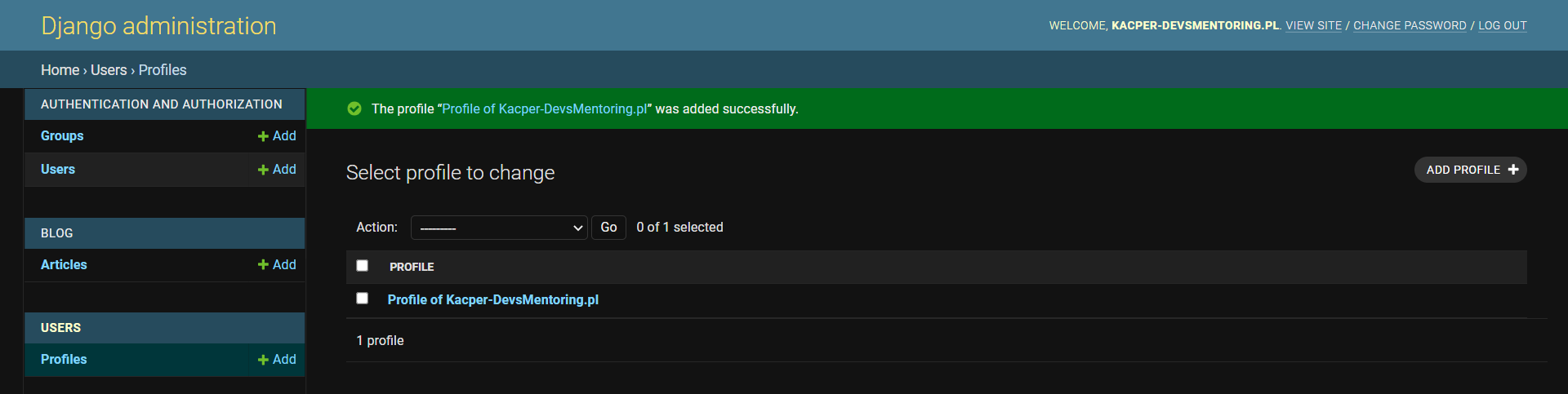
admin.site.register(Profile)

Po zarejestrowaniu tabeli, możemy śmiało przejść pod endpoint z panelem logowania - /admin i zalogować się na swoje administratorskie konto. Ja już to uczyniłem i oto widok, który zastałem:

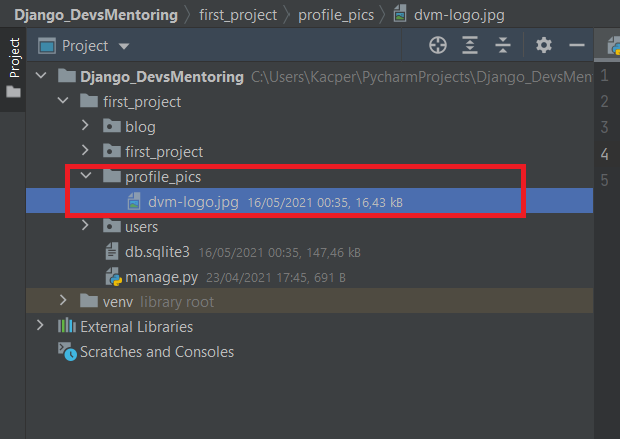


Kliknijmy przycisk Add w zaznaczonej zakładce Profiles. Zostaniemy wówczas przekierowani do strony, z poziomu której będziemy mogli dodawać nowe profile użytkowników. Ja z już istniejącym - Kacper-DevsMentoring.pl - powiązałem jeden z obrazów znajdujących się na dysku.





Po takim zauploadowaniu obrazu, w Naszej strukturze projektowej powinien utworzyć się nowy katalog profile\_pics wraz z umieszczonym w nim obrazem. Zastanawiasz się może teraz, po co to w ogóle ma miejsce i czy czasem nie kopiujemy istniejącego pliku do innej ścieżki katalogu…

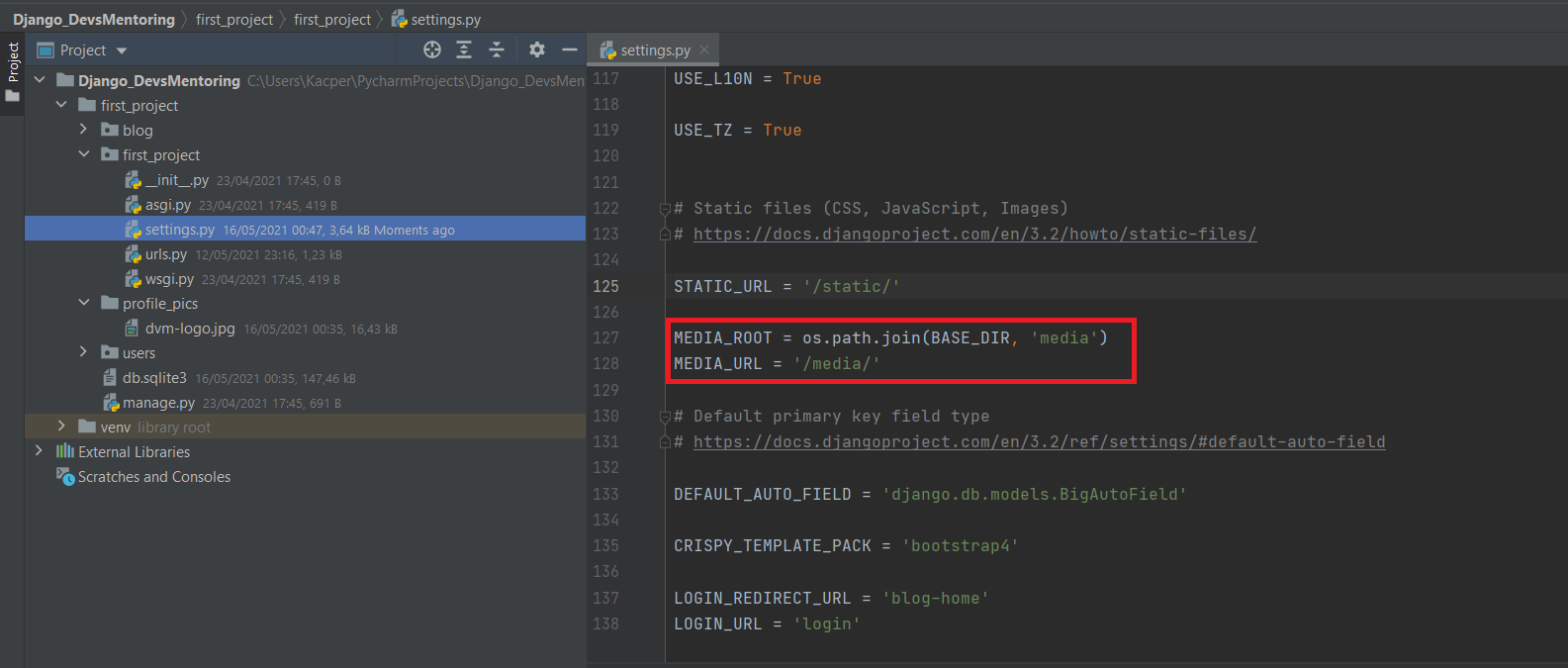


Słuszna uwaga! Zauważ jednak, że w tym momencie Naszą aplikację uruchamiamy lokalnie i faktycznie wydawać by się mogło, że taki upload pliku - do nowego katalogu profile\_pics - nie ma większego sensu. Ale weź pod uwagę fakt, iż Naszą aplikację będziemy chcieli z czasem zdeployować gdzieś na zewnętrznym serwerze. Wówczas taka modyfikacja zawartości projektowej aplikacji Django będzie konieczna, aby móc przechowywać zdjęcia po stronie serwera i w rezultacie efektywnie je wyświetlać. Dzięki takiemu zabiegowi, aplikacja Django wie, iż w celu pobrania dowolnego obrazu, musi przeszukać katalog profile\_pics.

Pewien mankament…

Wraz z utworzeniem takiego katalogu profile\_pics i umieszczeniem w nim nowego obrazu pojawia się pewne niedociągnięcie. Otóż profile\_pics znajduje się w globalnej strukturze projektu - na tym samym poziomie co plik manage.py. Nie jest to zbyt trafna organizacja katalogów projektu, dlatego też musimy ją nieco udoskonalić. Naszym celem będzie zmiana konfiguracji projektu tak, aby nowe obrazy były umieszczanie nie tylko w samym katalogu profile\_pics, ale dodatkowo wewnątrz nowo utworzonego folderu /media.

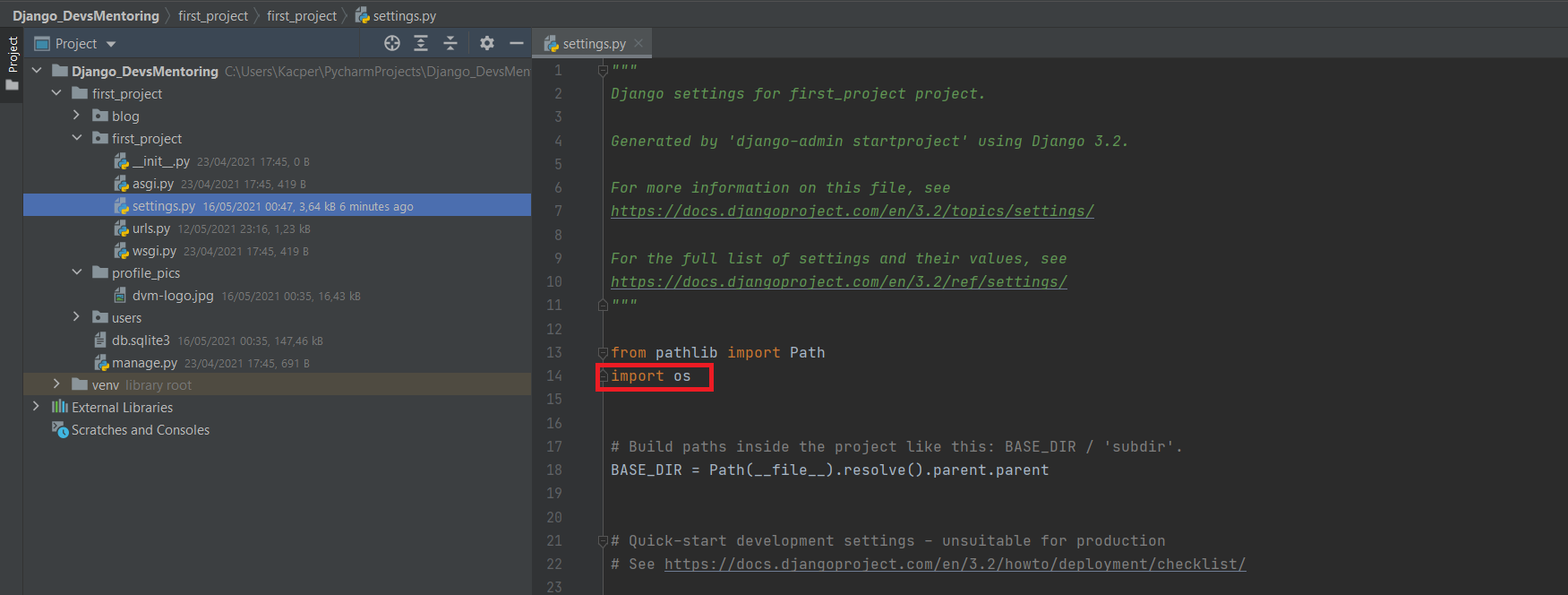
Przejdźmy zatem do pliku first\_project/settings.py i dodajmy poniższe linie kodu:



MEDIA\_ROOT = os.path.join(BASE\_DIR, 'media')

MEDIA\_URL = '/media/'

Nie zapomnijmy również o zaimportowaniu modułu os w pierwszych liniach pliku first\_project/settings.py:



Przechodząc już do objaśnienia:

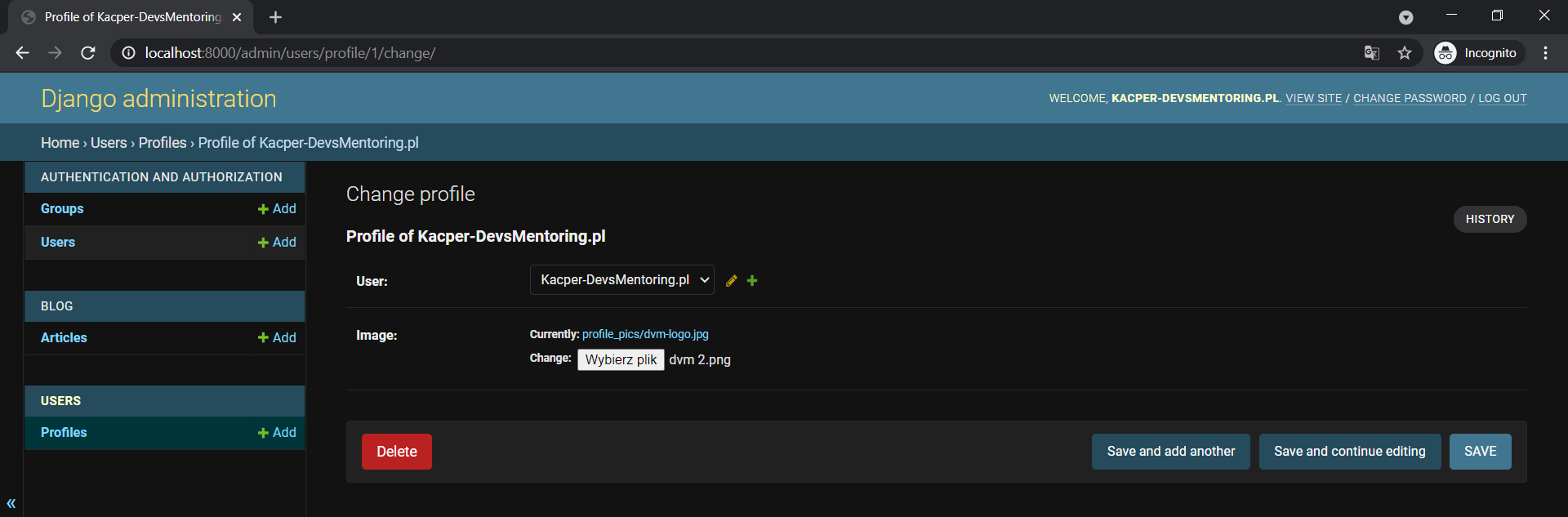
MEDIA\_ROOT to parametr, który służy do ustawienia ścieżki katalogu, w którym zapisywane będą uploadowane na serwer obrazy użytkownika. W celu ustalenia ścieżki, wykorzystaliśmy moduł os i jego metodę path.join(BASE\_DIR, ‘media’) w celu stworzenia uniwersalnej ścieżki dostępowej do obrazów - dostępnej właśnie pod katalogiem roboczym /media. To, co robi zapis os.path.join(...) to tworzenie ścieżki na podstawie podanych do niego parametrów (BASE\_DIR oraz ‘media’).

BASE\_DIR to stała, która jest zdefiniowana na poziomie ustawień Django i tak jak można się domyślić, przyjmuje wartość głównej ścieżki projektowej.

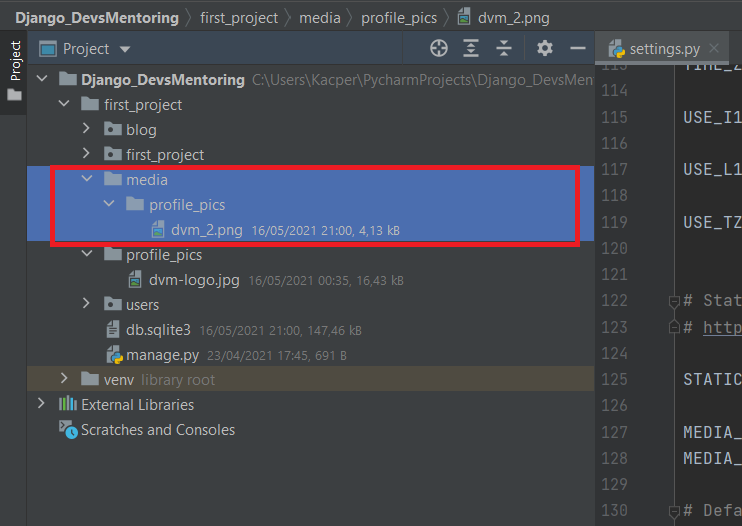
MEDIA\_URL odpowiada natomiast za określenie ścieżki dostępowej do przesłanych multimediów z poziomu przeglądarki. Gdybyśmy zatem chcieli przejrzeć przesłany na serwer plik, wystarczy będzie przejść do endpointu /media/nazwa\_obrazu.

**Sprawdzenie nowej ścieżki plików**

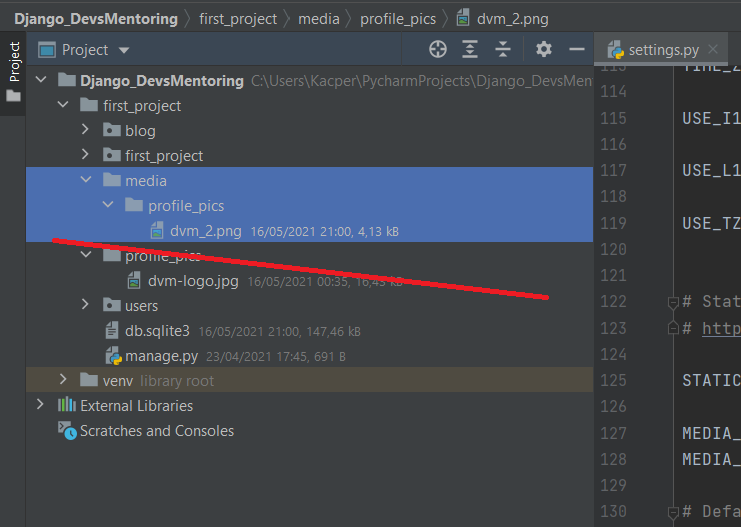
Zacznijmy od sprawdzenia, czy MEDIA\_ROOT faktycznie pozwoliło Nam zmienić docelowy katalog przechowywania obrazków. Wróćmy do panelu administratora i spróbujmy raz jeszcze przesłać awatar dla użytkownika Kacper-DevsMentoring.pl. Wybrałem dowolny nowy plik z katalogu i zastąpiłem obecny awatar nowym:



Po wciśnięciu przycisku Save i przejściu z powrotem do Naszego projektu, okazało się, że katalog media został poprawnie utworzony, a w nim również katalog profile\_pics (utworzył się, ponieważ ustawiliśmy tę opcję przy tworzeniu tabeli) wraz z nowo przesłanym plikiem. Wszystko zatem działa zgodnie z oczekiwaniami - stworzyliśmy wydzielony katalog media dla awatarów z możliwością dodawania kolejnych katalogów, które mogłyby odzwierciedlać również inne przesłane media, np. filmy wideo.

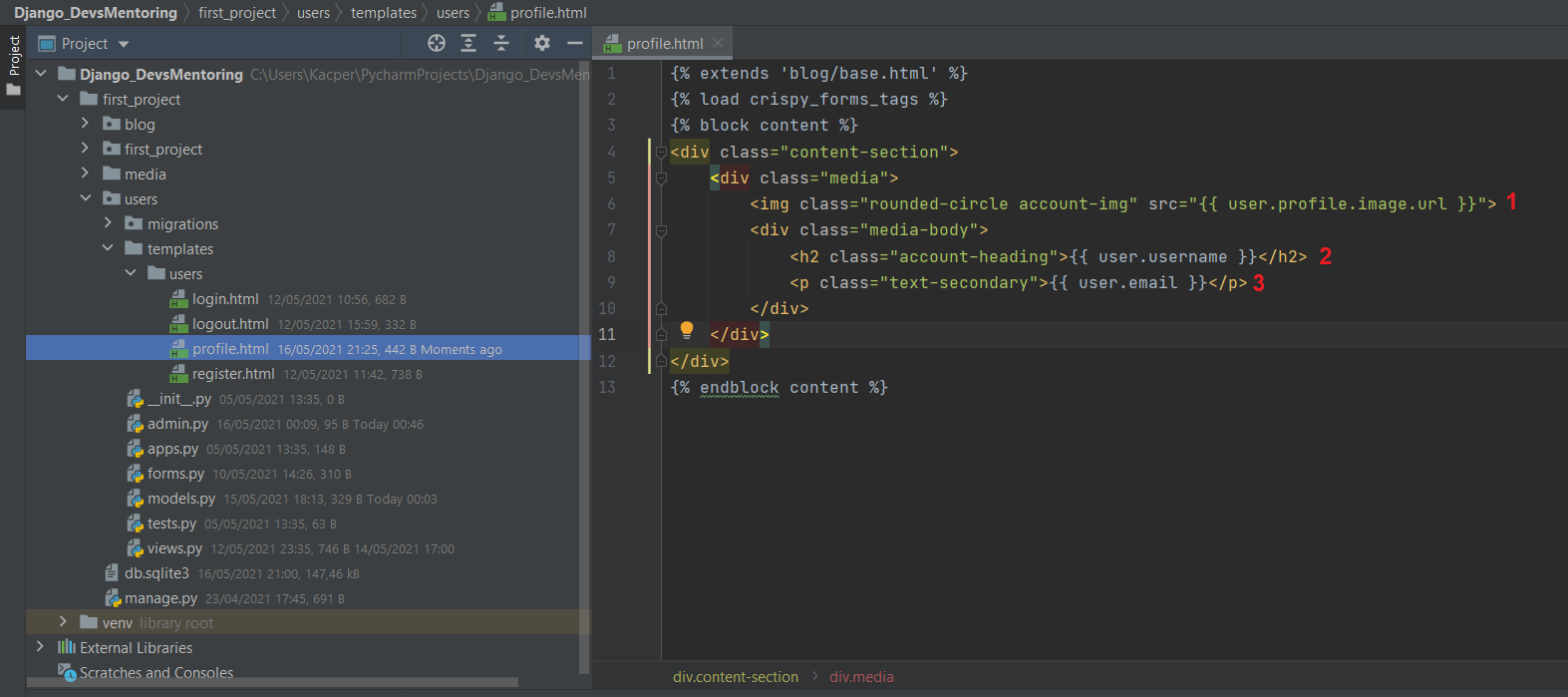


Katalog profile\_pics znajdujący się w głównym katalogu projektu nie będzie Nam już potrzebny, możemy więc go śmiało usunąć.



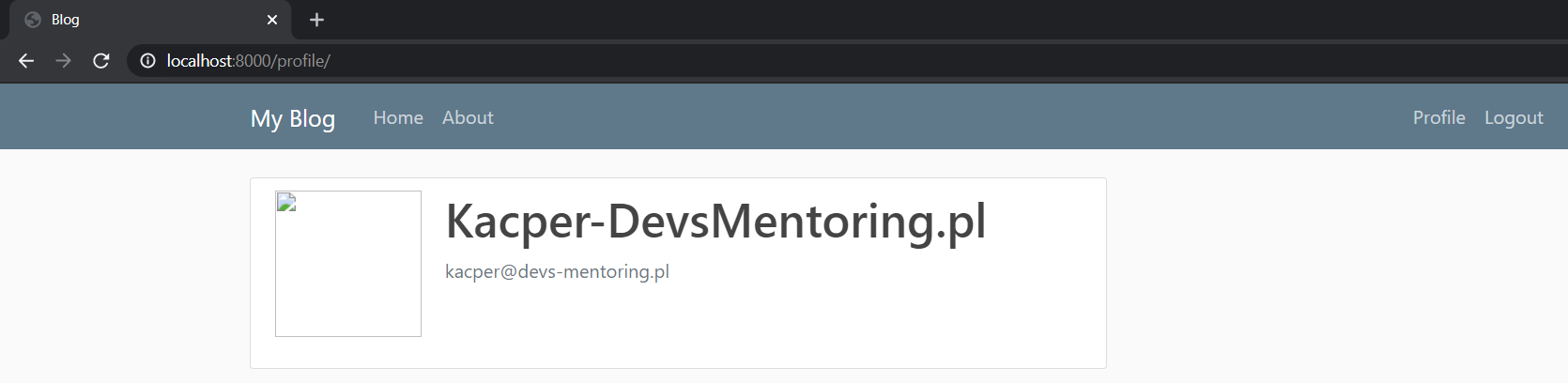
**Wyświetlanie przesłanych zdjęć w obrębie profilu użytkownika**

Następnym krokiem będzie sformatowanie nieco szablonu users/profile.html, tak aby zdjęcie użytkownika wyświetlane było obok jego nazwy. Aby zapewnić jak najlepszy wygląd podstrony profile, wykorzystamy gotowy kod Bootstrap, który znajdziesz w załącznikach do szkolenia. W rezultacie Nasz plik users/profile.html powinien wyglądać następująco:



Kolejnymi cyframi oznaczyłem odnośniki do pól z tabeli User. Zaczynając od końca - (3) to adres email użytkownika, dla którego wyświetlamy overview profilu, (2) jego nazwa, a (1) to pole, które pozwala tworzyć Nam odnośnik do przesłanego obrazu - przechowywanego po stronie serwera.

Mega! Szablon utworzony, jeżeli jednak spróbujemy przejść do endpointu /profile, ujrzymy:



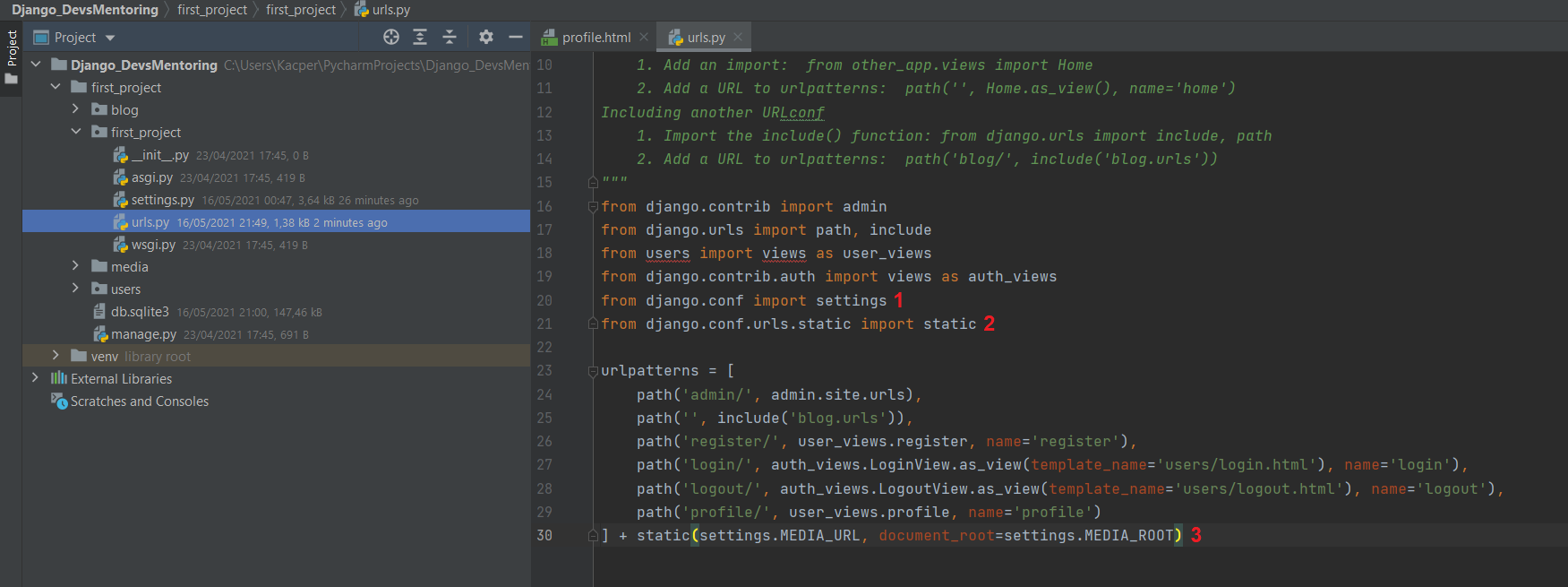
Nasz awatar nie wczytuje się prawidłowo. Wynika to z tego, iż jeszcze nie zdefiniowaliśmy w pełni endpointa, z którego moglibyśmy pobierać przesłane pliki użytkowników. Co prawda dodaliśmy uprzedniu w pliku settings.py parametr MEDIA\_URL, ale jak się okazuje - to nie wszystko. Musimy dokonfigurować Naszą aplikację w pliku first\_project/urls.py.

Dokładny proces takiej konfiguracji został opisany w dokumentacji Django: [Managing static files (eg images, JavaScript, CSS)](https://docs.djangoproject.com/en/3.2/howto/static-files/) i właśnie na nim się wzorowałem, tworząc aplikację.

Zwróć również uwagę na fakt, iż z dokumentacji wynika jasno - inaczej będziemy konfigurowali aplikację, którą będziemy chcieli wystawić na produkcję, a inaczej taką, która służyć Nam będzie tylko i wyłącznie w procesie developmentu (czyli tym, który przeprowadzamy teraz).



Zaznaczony na screenshot-cie kod przeniesiemy w odpowiedniej postaci do 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

from django.conf import settings

from django.conf.urls.static import static

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"),

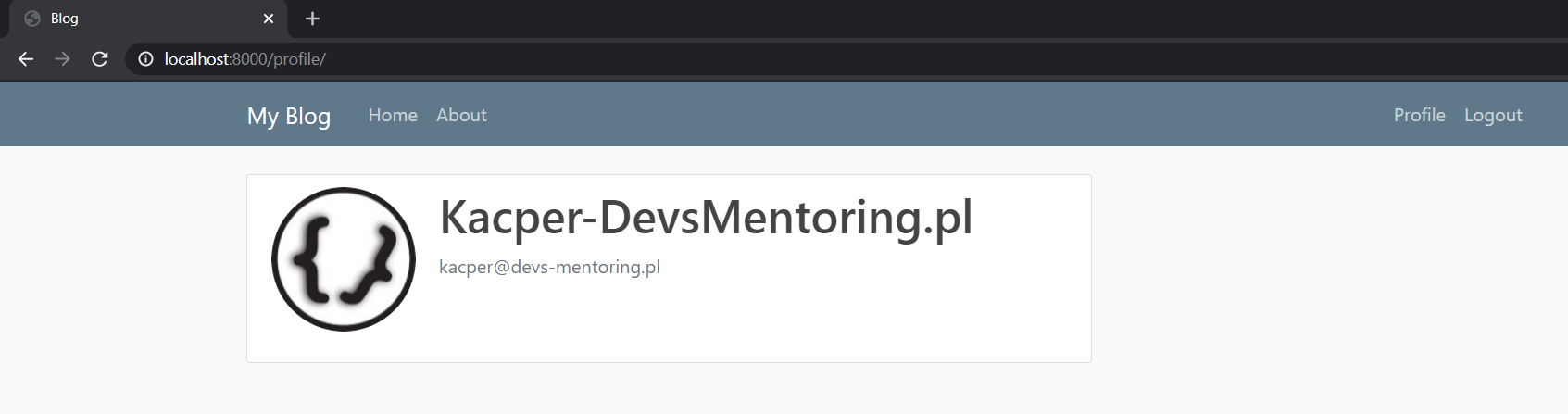
path("profile/", user\_views.profile, name="profile"),

] + static(settings.MEDIA\_URL, document\_root=settings.MEDIA\_ROOT)

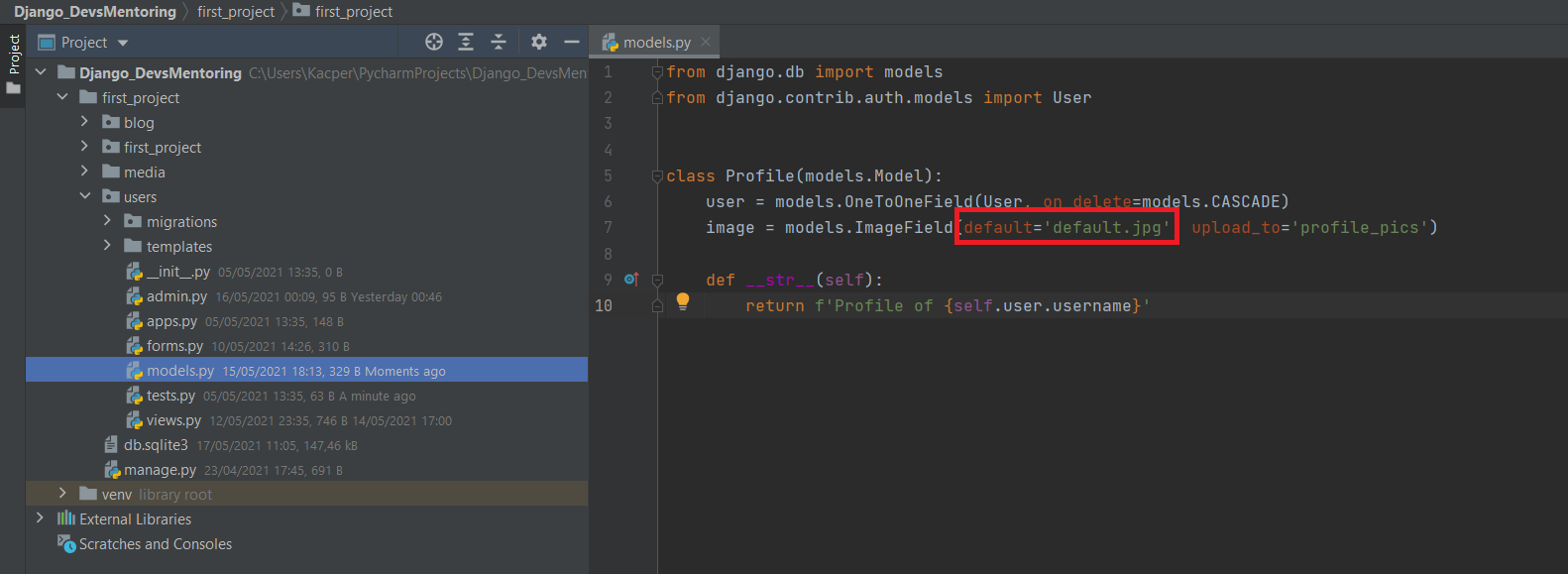
Wprowadzone zmiany oznaczyłem kolejnymi cyframi (1), (2) i (3).

**Z nadzieją na działający endpoint profilu…**

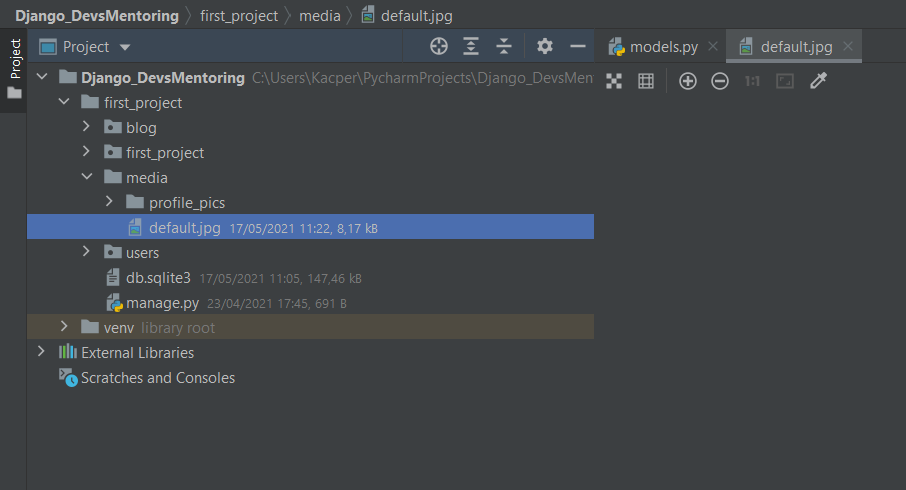
Po przeprowadzeniu powyższych zmian, myślę, że jesteśmy gotowi zalogować się na konto, do którego przypisaliśmy awatar i przejść na endpoint /profile. Powinniśmy wówczas otrzymać następujący widok:



Wszystko działa zgodnie z przewidywaniami! Musimy jednak zadbać o jeszcze jedną rzecz - umożliwienie przypisywania domyślnego awatara do użytkowników, którzy nie wybrali jeszcze zdjęcia. Pamiętasz, jak w pliku users/models.py podczas tworzeniu klasy Profile i definiowaniu pola image określiliśmy parametr default jako: ‘default.jpg’?



To właśnie nazwa obrazu umieszczonego w katalogu /media, który będzie pełnił rolę domyślnie przypisywanego obrazka do profilu użytkownika, jeżeli ten nie zdecyduje się zuploadować własnego awatara. Tak więc, biorąc pierwszy lepszy obraz - umieśćmy go we wspomnianym katalogu. Pamiętaj aby posiadał on nazwę zgodną z wartością zadeklarowaną w obrębie parametru default - czyli default.jpg.



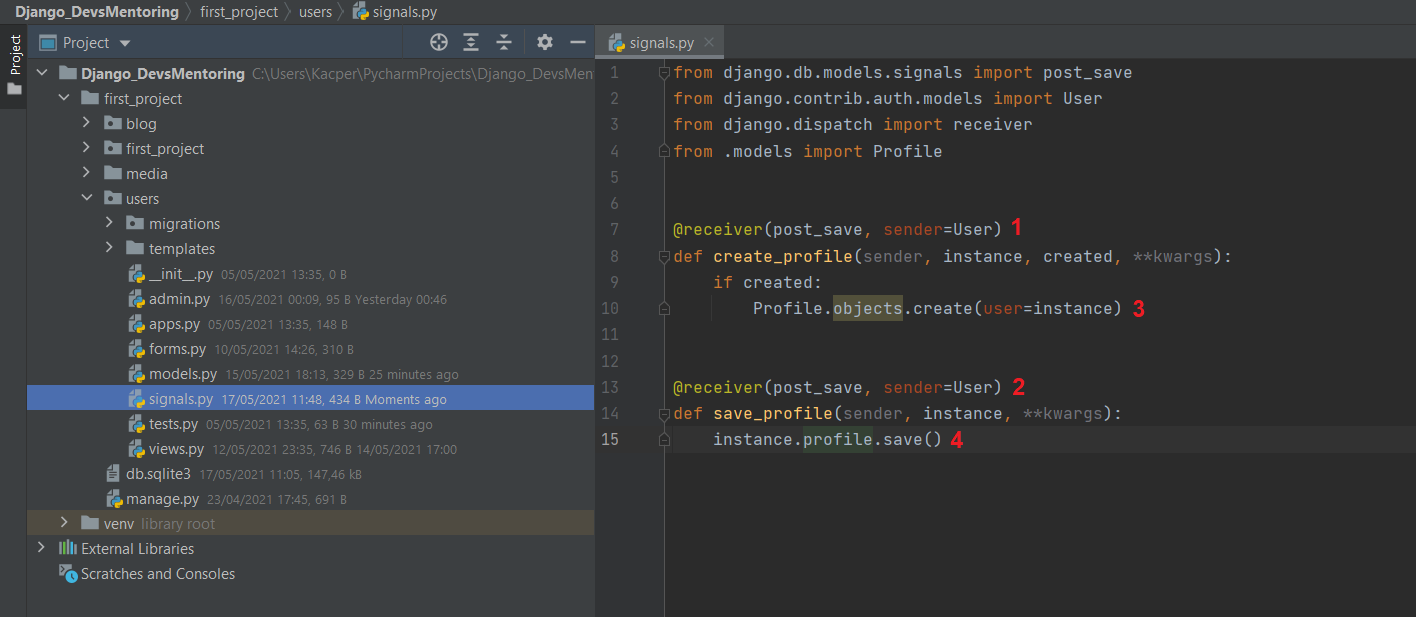
Od teraz każdy nowo utworzony profil, dla którego nie przypisaliśmy zdjęcia, zaciągnie domyślny obraz.

**Ostatnia zmiana**

Do tej pory, chcąc przypisać profil do danego użytkownika, musieliśmy przejść do panelu administracyjnego, tabeli Profile i ręcznie utworzyć nowy profil dla wybranego usera. Jest to rozwiązanie dość niepraktyczne, ponieważ musielibyśmy tworzyć co rusz nowy profil przy każdej kolejnej rejestracji do Naszego serwisu. Dlatego musimy wprowadzić modyfikację, która polegałaby na tym, iż nowy profil byłby tworzony automatycznie i przypisywany do nowo zarejestrowanego użytkownika.

Aby móc zaimplementować taką funkcjonalność, skorzystamy z tzw. sygnałów. W skrócie - sygnały umożliwiają notyfikację pewnych komponentów programu (receiverów) po wywołaniu pewnego zdarzenia (eventu) w programie. W Naszym przykładzie - będziemy chcieli, aby po utworzeniu każdego nowego użytkownika (utworzenie/zapis to event; użytkownik to sender), tworzony był nowy obiekt profilu (funkcja będąca receiver-em) i przypisywany do odpowiedniego użytkownika.

Aby zaimplementować całą wyżej opisaną logikę, utwórzmy nowy plik users/signals.py i umieśćmy w nim poniższy kod:



from django.db.models.signals import post\_save

from django.contrib.auth.models import User

from django.dispatch import receiver

from .models import Profile

@receiver(post\_save, sender=User)

def create\_profile(sender, instance, created, \*\*kwargs):

if created:

Profile.objects.create(user=instance)

@receiver(post\_save, sender=User)

def save\_profile(sender, instance, \*\*kwargs):

instance.profile.save()

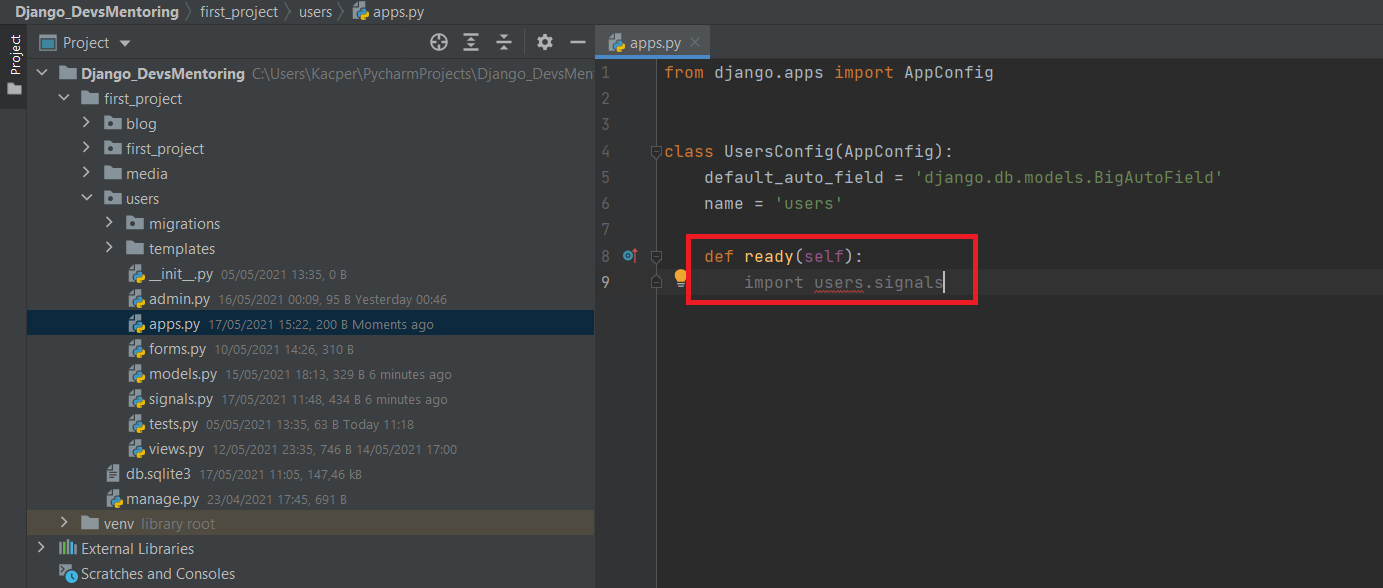
(1) - to dekorator określający, iż funkcja create\_profile będzie jednym z receiverów wywoływanym w momencie utworzenia nowego obiektu (czyli triggerowaniu post\_save). Parametr sender definiuje natomiast, przy tworzeniu obiektu jakiej klasy (User) ma być tworzony ów sygnał.

(2) - musimy również kontrolować zapis nowego profilu do tabeli, podczas gdy zakończy się funkcja create\_profile; stąd obecność drugiego @receiver wraz z funkcją save\_profile.

(3) - odpowiada za utworzenie nowego obiektu profilu z kluczem obcym jako userem - senderem sygnału

(4) - linia ta odpowiada za zapisywanie nowo utworzonego rekordu profilu (wykonywanego w ramach funkcji create\_profile) do tabeli bazy danych

Na koniec, aby zapewnić prawidłowe triggerowanie sygnału z poziomu klasy User, musimy przejść do pliku users/apps.py i zmodyfikować znajdującą się tam klasę UserConfig w następujący sposób:

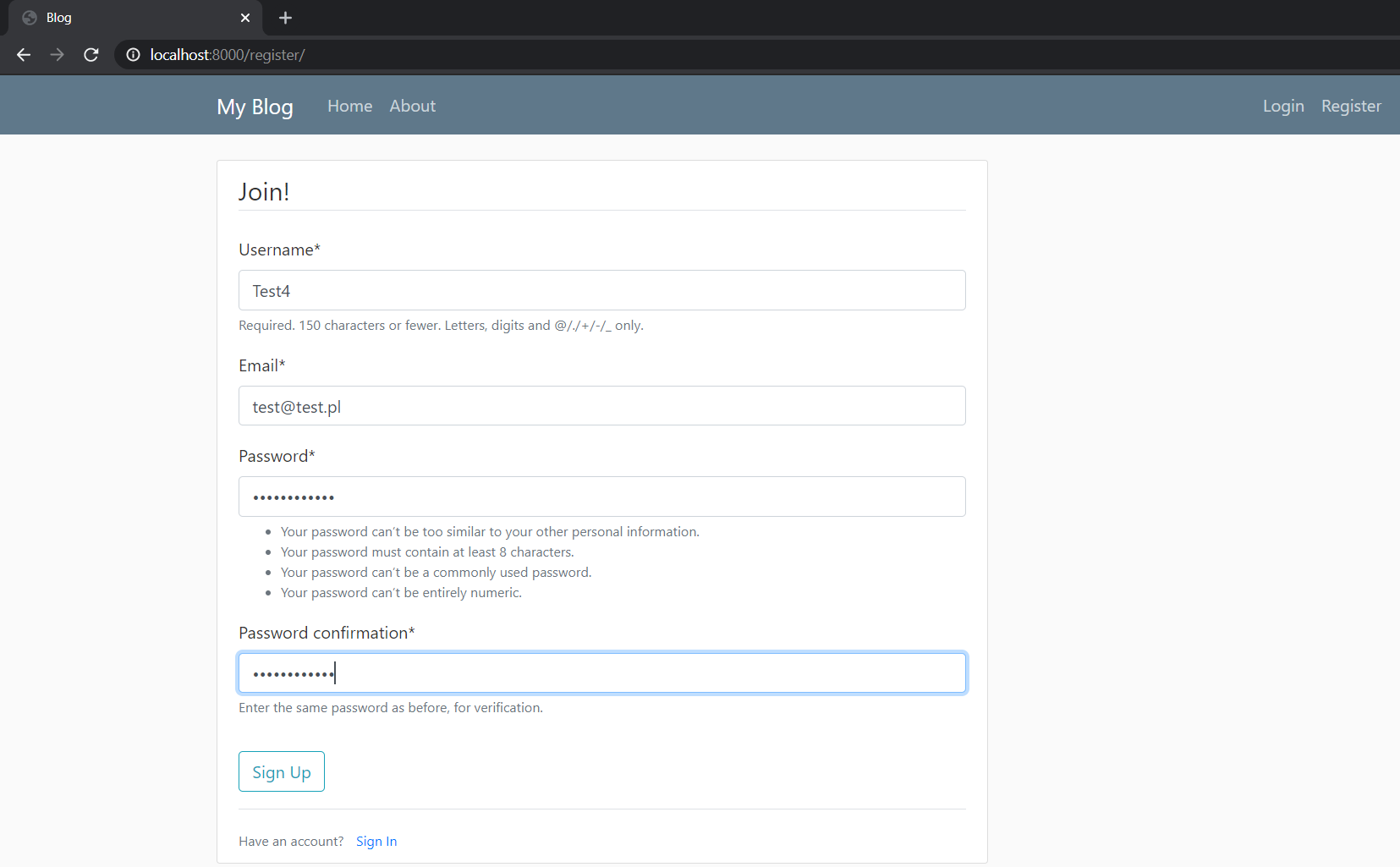


Jak widzisz na załączonym screenie, IDE może podświetlać na czerwono napis users przy import users.signals, ale niczemu to nie wadzi - aplikacja Django uruchamia się prawidłowo.

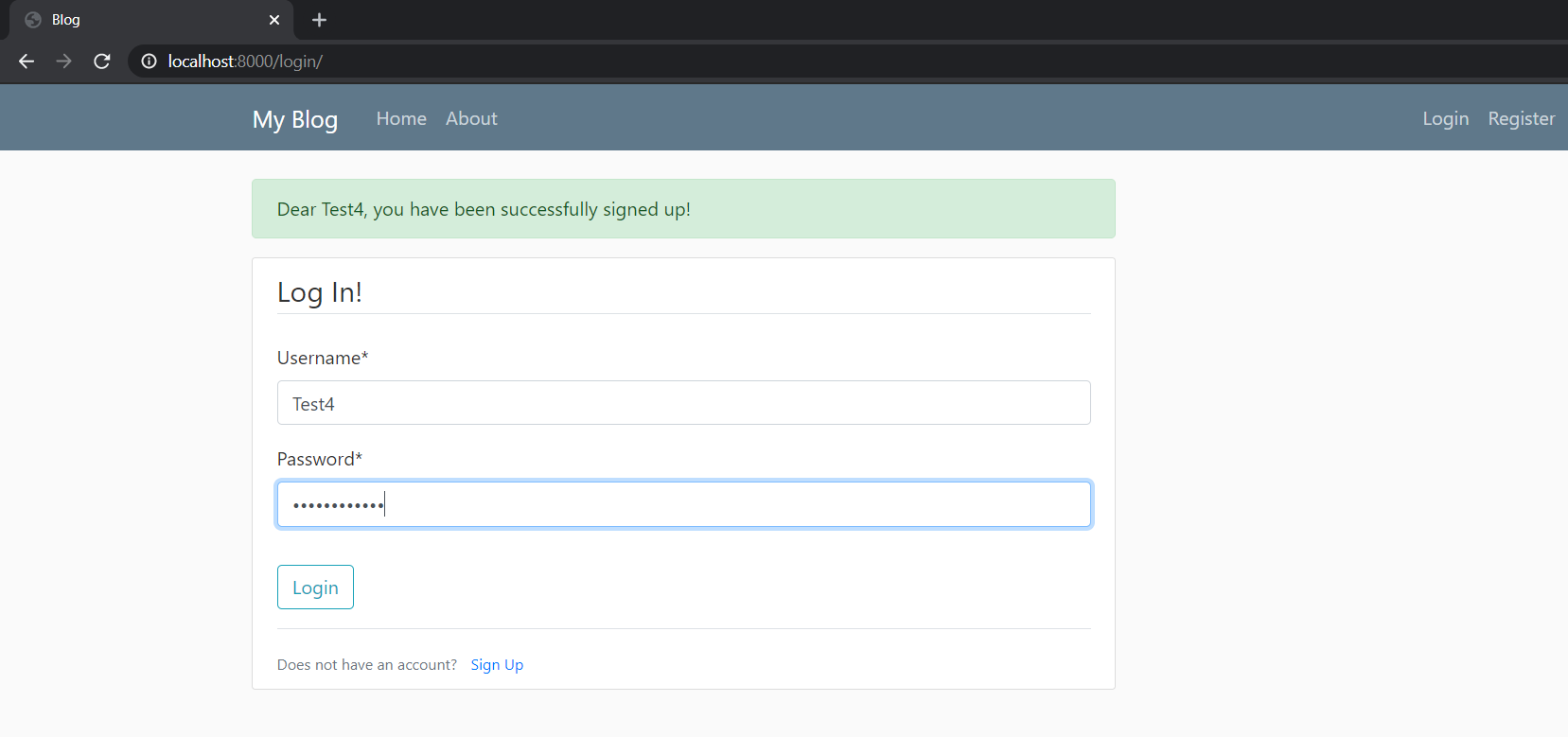
Sprawdźmy zatem jej funkcjonalność.

**Ostateczny test**

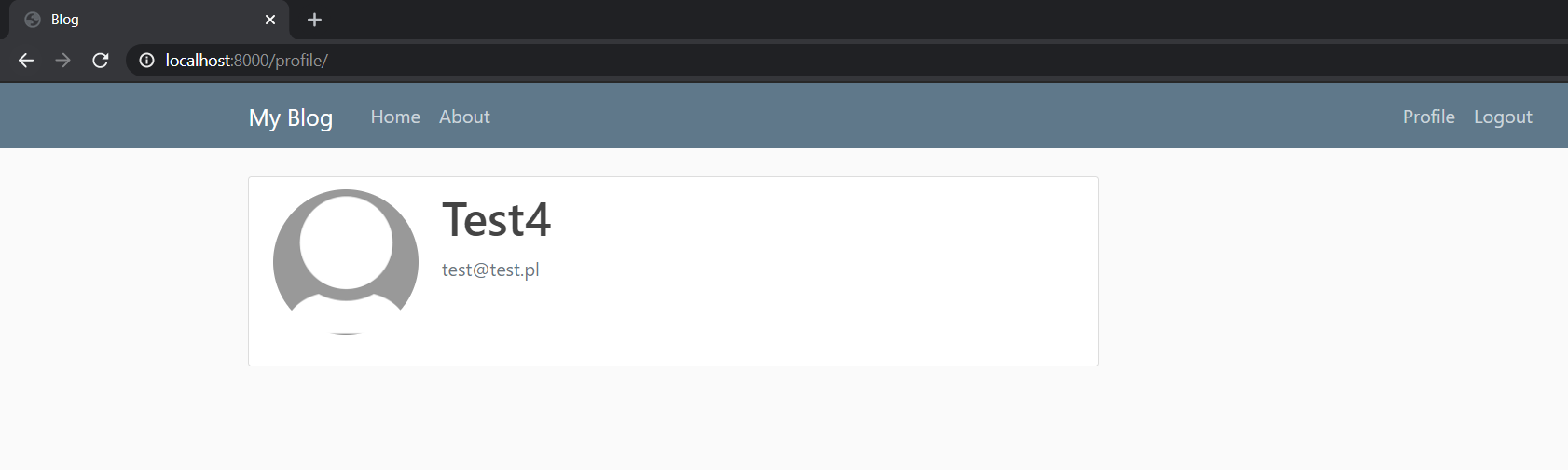
A więc reasumując - od teraz aplikacja powinna automatycznie przypisywać świeży profil do nowo utworzonego użytkownika wraz z domyślnym awatarem. Niestety na ten moment, użytkownik swoje zdjęcie będzie mógł zmienić tylko z poziomu /admin, ale zajmiemy się dodaniem możliwości zmiany swojego zdjęcia wraz z innymi informacjami z poziomu profilu w następnych szkoleniach.



*Utworzenie nowego użytkownika*



*Zalogowanie do konta*



*Podgląd profilu użytkownika Test4*