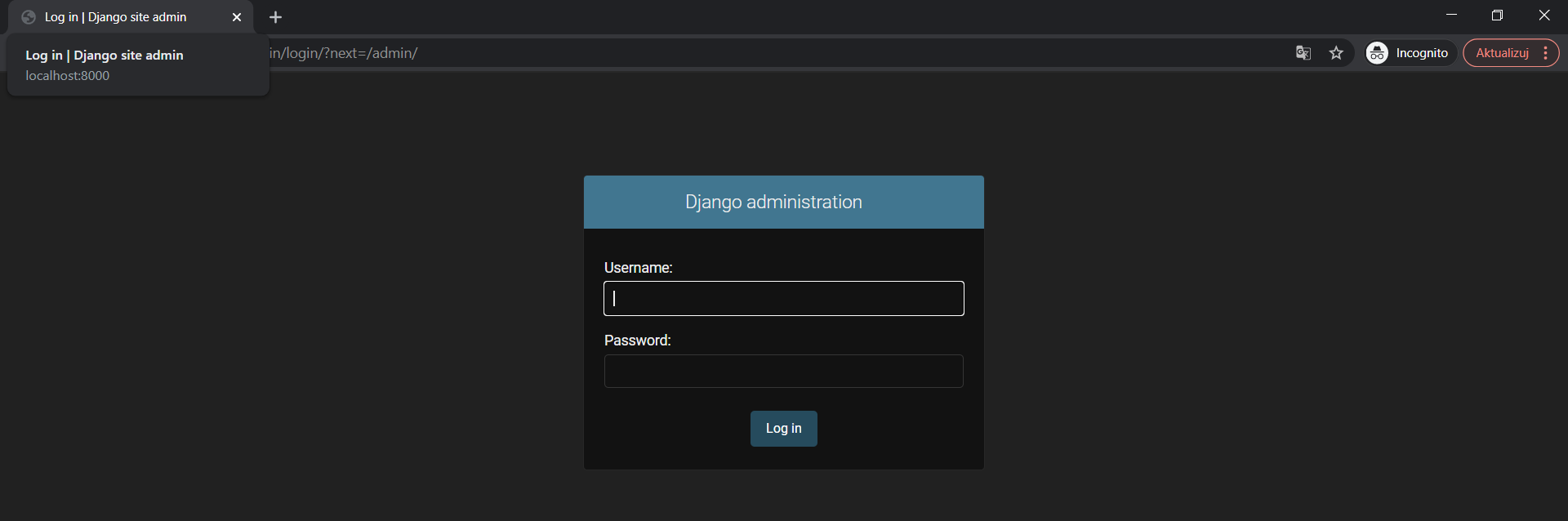
**Django - szkolenie 3**

**Panel administratora**

**Wprowadzenie**

Pamiętasz jeszcze z początku szkolenia, co napotkamy po przejściu na adres: localhost:8000/admin?



Po wpisaniu powyższego adresu, wyświetla Nam się panel logowania do konsoli administracyjnej. Na ten moment jest to dla Nas niejako nieosiągalna Ziemia Obiecana (skąd bowiem uzyskać potrzebne dane do logowania). Dlatego właśnie przedstawię Ci sposób, jak w łatwy sposób stworzyć konto administratorskie, które posłuży Nam, np. do zarządzania innymi użytkownikami aplikacji, utworzonymi tabelami bazy danych etc.

**Dodanie superusera**

Aby dodać pierwsze konto administratorskie, należy w konsoli Twojej aplikacji wykonać następujące polecenie (uprzednio zamykając aplikację wywołaną poleceniem python manage.py runserver):

python manage.py createsuperuser

jeżeli napotkasz błędy po jego wywołaniu, to dodatkowo wykonaj:

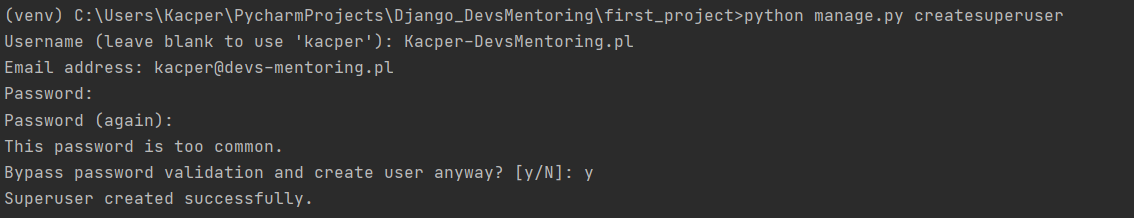
python manage.py makemigrations

**Dlaczego makemigrations?**

Polecenie createsuperuser tworzy w budowanej tabeli auth\_user nowego użytkownika o określonych danych. Auth\_user tak jak, np. django\_session, jest jedną z domyślnie tworzonych tabel aplikacji. Aby jednak korzystać z tych tabel, konieczne jest właśnie wykonanie polecenia makemigrations, aby stworzyć odpowiednie struktury.

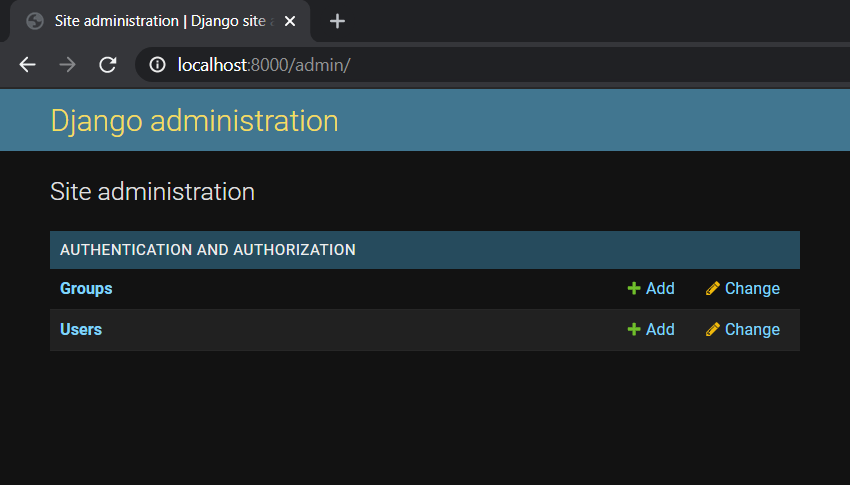
Utworzę mojego pierwszego użytkownika o nazwie: Kacper-DevsMentoring.pl, email-u [kacper@devs-mentoring.pl](mailto:kacper@devs-mentoring.pl) i haśle admin123. Oczywiście nie muszę chyba Ci przypominać, że dodając użytkowników do aplikacji, którą będziemy deploy-owali w sieci, konieczne jest utworzenie znacznie silniejszego hasła niż to, które podałem.

Prześledź ten proces:



**Logowanie do konta**

Mają tak utworzone konto, spróbujmy zalogować się do panelu, podając dane przed chwilą utworzonego admina. Jeżeli wszystko przebiegło zgodnie z oczekiwaniami, to powinniśmy ujrzeć widok taki jak poniżej:

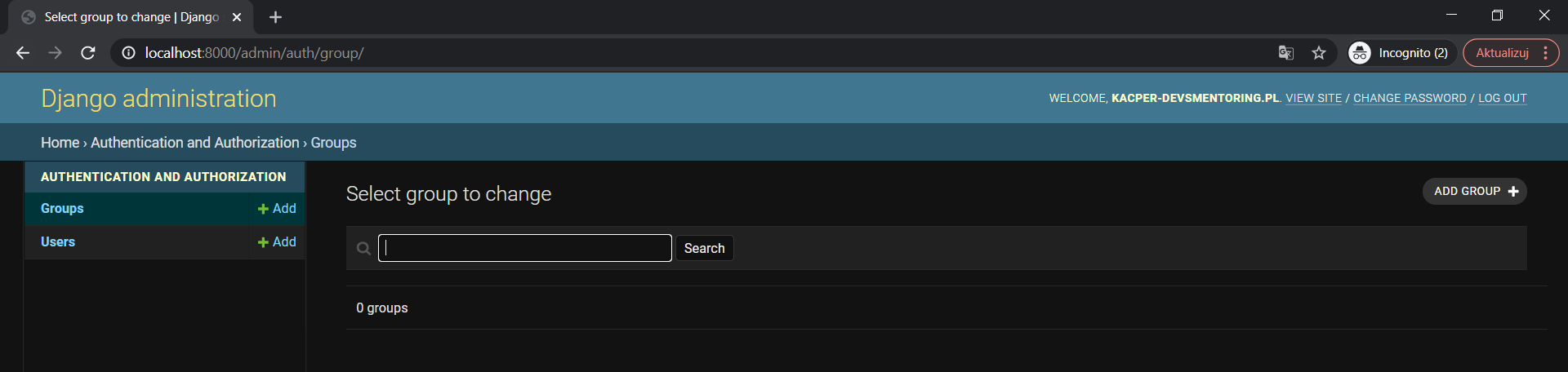


*Panel administracyjny*

Na pierwszy rzut oka widzimy 2 zakładki - Groups oraz Users. Za chwilę przeanalizujemy każdą z nich, ale słowem wstępu powinieneś wiedzieć, iż pod zakładką Groups znajdują się wszystkie utworzone grupy użytkowników w aplikacji, a pod Users wszyscy użytkownicy.

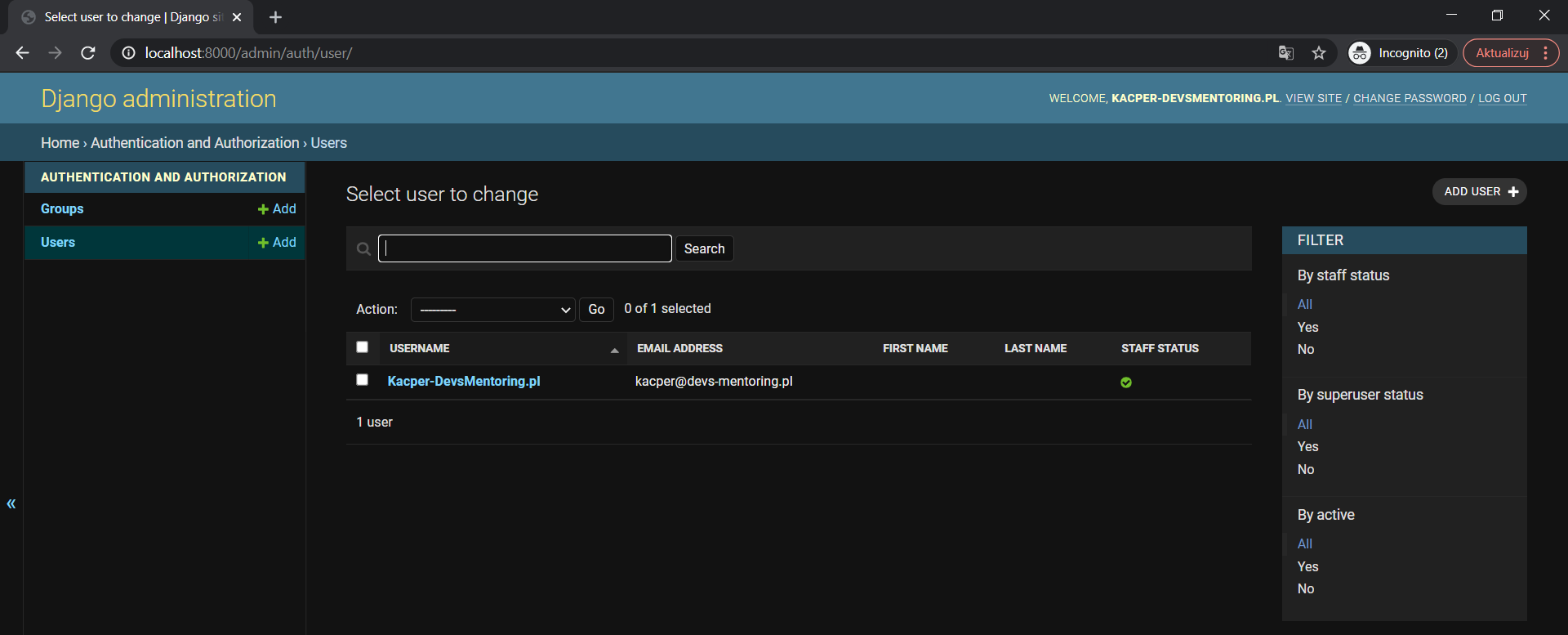
Wszystko zależy od nas, jakie grupy i jakich użytkowników będziemy chcieli utworzyć w aplikacji. Dana grupa służyć może ujednoliceniu pewnych uprawnień dla różnych użytkowników, np. pracowników, administratorów bloga itd.

Na ten moment zakładka Groups jest pusta:

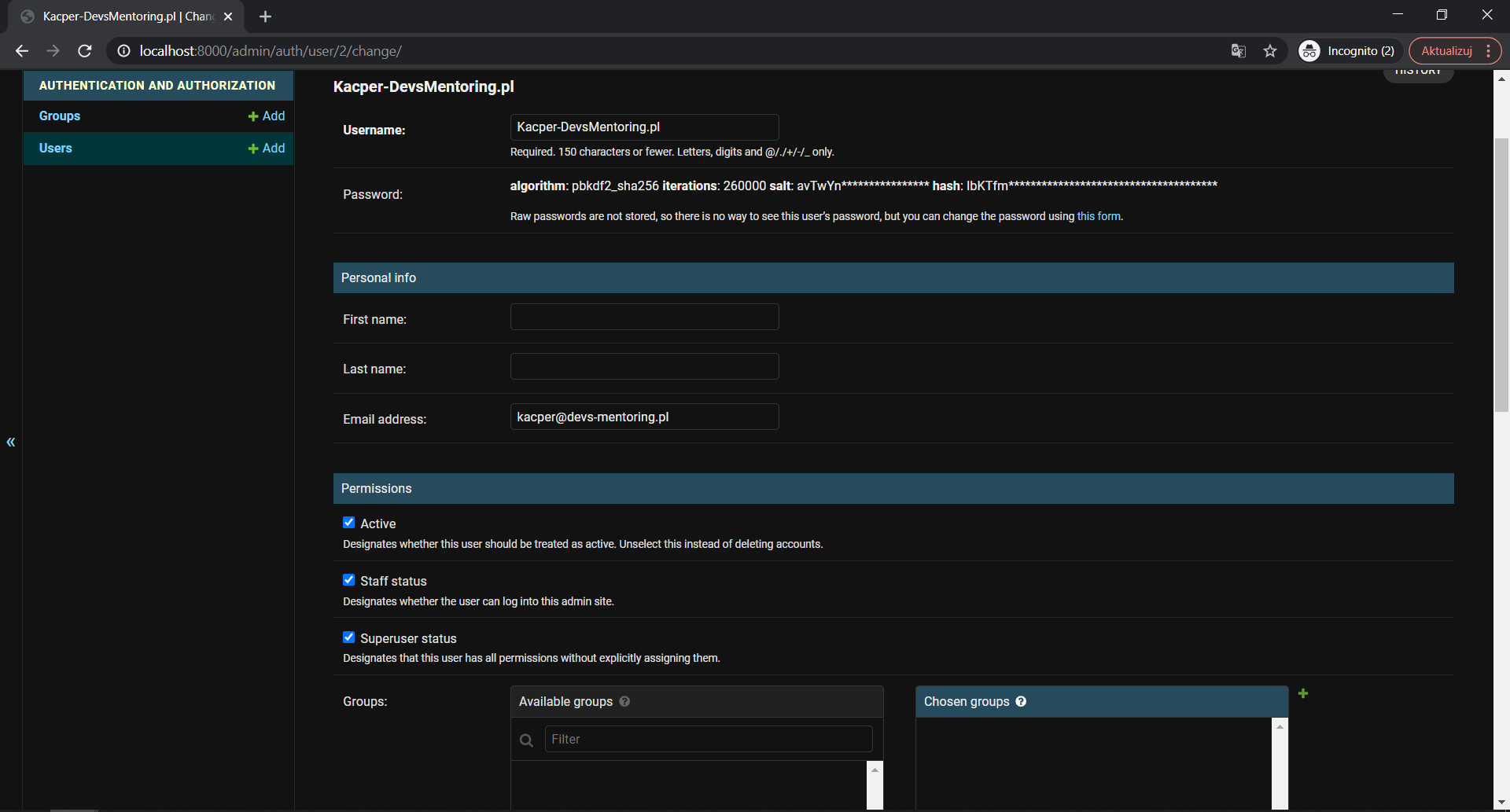


*Po przejściu do zakładki Groups*

Przechodząc natomiast do Users, możemy zaobserwować Nasze konto administratorskie:

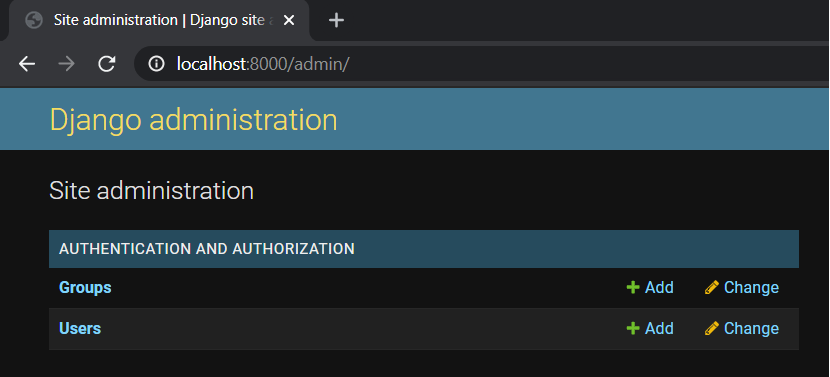


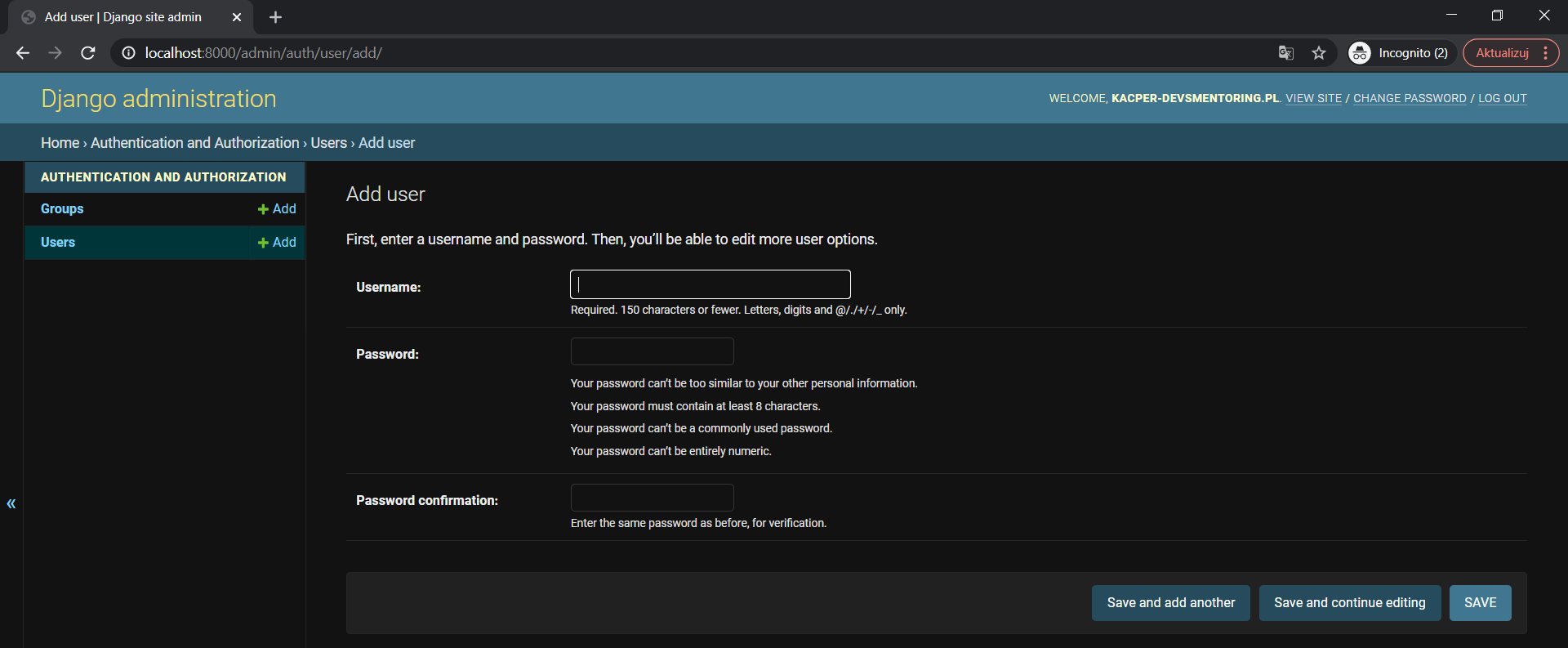
Z poziomu konsoli możemy go dowolnie edytować - zmieniać nazwę, uprawnienia czy status aktywności.



*Ustawienia konta Kacper-DevsMentoring.pl*

Równocześnie, jak już zapewne zauważyłeś, możemy dodawać dowolnych użytkowników. Mogą to być nawet kolejni administratorzy (dzięki temu zwolnieni jesteśmy z konieczności wykorzystywania konsoli i wykonywania polecenie createsuperuser).





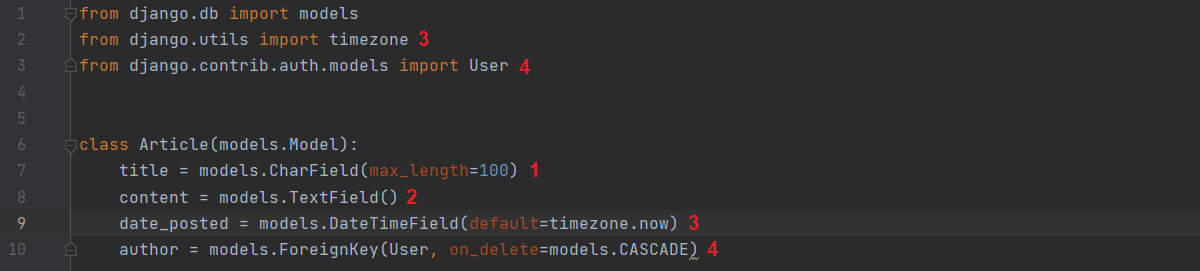
**Tworzenie tabeli w bazie**

Póki co, nie ma potrzeby, abyśmy głębiej analizowali konsolę administratora. Jak zapewne zauważyłeś, obsługa panelu jest bardzo intuicyjna i nie powinna stanowić dużego problemu. Dlatego pójdźmy krok dalej i nauczmy się, jak tworzyć własne tabele w Django-wej bazie danych.

Chcielibyśmy, aby w Naszej aplikacji znalazła się tabela, w której przechowywalibyśmy wszystkie artykuły bloga. Ponadto każdy z tych artykułów powiązany by był z tabelą użytkowników (autorów). To, jakich użytkowników będziemy mogli przypisać do określonego artykułu, determinować będzie tabela auth\_users, czyli zarejestrowanych użytkowników aplikacji.

Każdą kolejną tabelę, którą będziemy chcieli umieścić w Naszej aplikacji, dodawać będziemy w formie oddzielnej klasy w pliku blog/models.py. Spójrzmy, jak wyglądać będzie zatem klasa Article, która odzwierciedlać będzie pojedynczy rekord artykułu w bazie.

Każdy artykuł posiadać będzie takie pola jak: title - tytuł artykułu, content - treść artykułu, date\_posted - data opublikowania, author - klucz obcy odnoszący się do użytkownika, który opublikował post.



*Zawartość klasy (tabeli) Article*

from django.utils import timezone

from django.db import models

from django.contrib.auth.models import User

class Article(models.Model):

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

content = models.TextField()

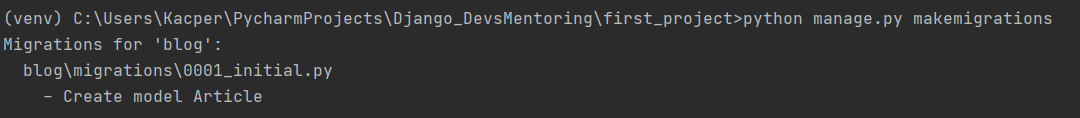
date\_posted = models.DateTimeField(default=timezone.now)

author = models.ForeignKey(User, on\_delete=models.CASCADE)

1. Pierwsze pole rekordu - title - typu CharField o ograniczonej długości 100 znaków
2. Content - kolejne pole tekstowe typu TextField. Różnica między CharField a TextField jest taka, iż pierwszy typ używamy, gdy chcemy określić maksymalną długość tekstu w danym polu. Dobrą zasadą jest, aby w pozostałych przypadkach używać TextField.
3. Date\_posted - pole określające datę utworzenia artykułu. Zwróć uwagę na argument default=timezone.now i umieszczony import from django.utils import timezone na początku pliku. Umożliwia on pobieranie aktualnej daty, w której został utworzony post, jeżeli nie zdefiniujemy dla niego daty dodania.
4. Pole author - klucz obcy w artykule. Odnosi się do określonego użytkownika spośród wszystkich utworzonych. Zwróć uwagę, że konieczne było dokonanie importu - from django.contrib.auth.models import User. Argument on\_delete pozwala określić Nam, jak zachować się ma rekord artykułu, gdy zostanie usunięty jego autor. W tym przypadku definiujemy zachowanie kaskadowe (przez przypisanie do pola wartości models.CASCADE), a więc gdy usuniemy określonego użytkownika, również wszystkie jego artykuły zostaną zlikwidowane.

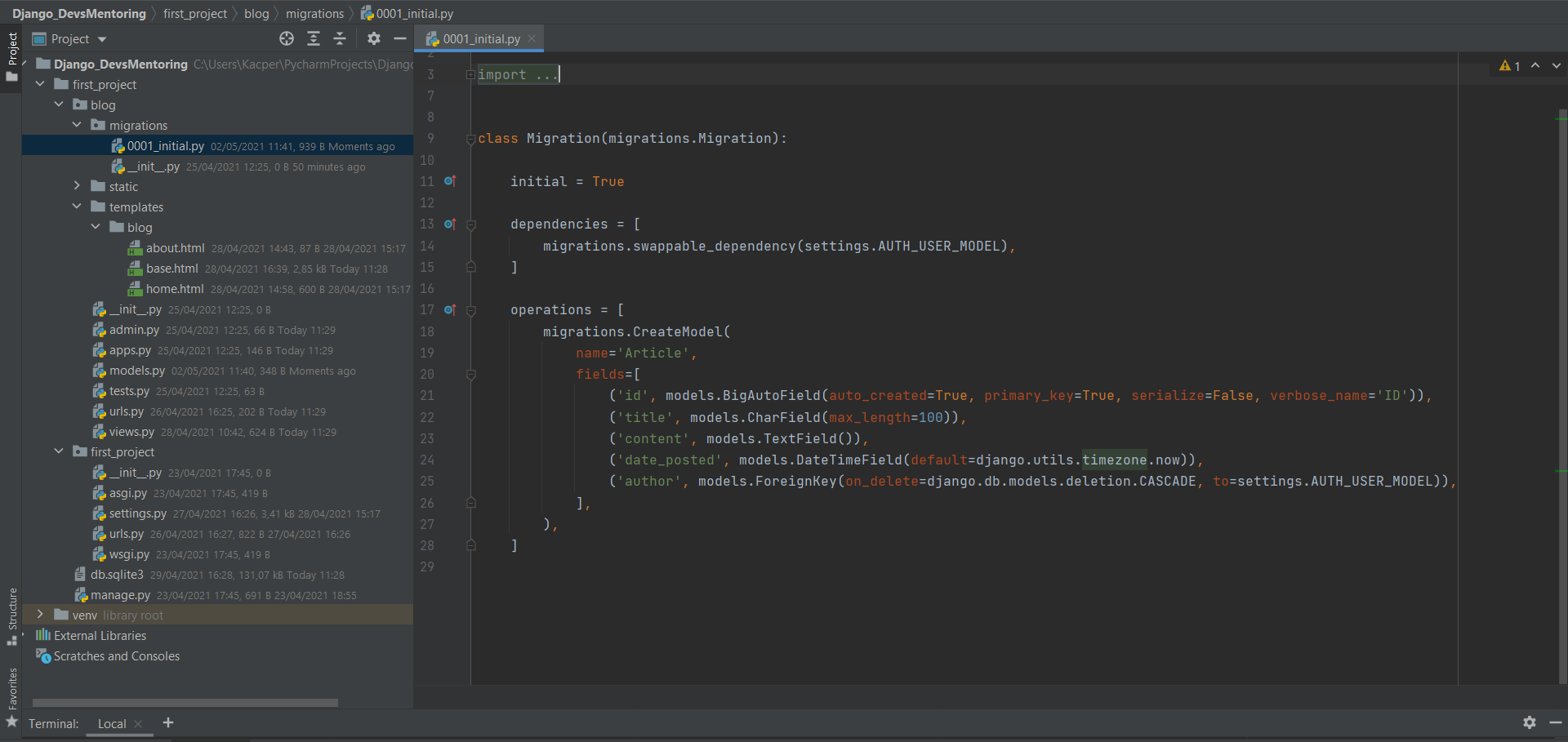
Zdaj sobie sprawę, że samo umieszczenie kodu w pliku nie spowoduje jeszcze wygenerowania tabeli Article. Mając tak przygotowany kod tabeli z ORM-owej bazy danych, musimy kolejno stworzyć odpowiedni kod SQL i wykonać jego migrację - co dopiero spowoduje utworzenie oczekiwanej tabeli. W tym celu wykonamy w konsoli dwa polecenia:

* python manage.py makemigrations



Makemigrations tworzy odpowiednik tego, co zapisaliśmy w pliku models.py, ale w nieco bardziej skomplikowanej, gotowej do zmigrowania, wersji.

Efekt polecenia możemy podejrzeć w katalogu migrations i pliku, który powstał w wyniku powyższego polecenia. W moim przypadku plik ma nazwę 0001\_initial.py i wygląda następująco:



Zanim przystąpimy do migrowania kodu i aplikowania nowej tabeli do aplikacji, chciałbym pokazać Ci jeszcze jedno polecenie: python manage.py sqlmigrate blog 0001 (gdzie blog to nazwa Naszej aplikacji a 0001, to numer migracji utworzonej w katalogu migrations).

Rezultatem polecenia powinno być wyświetlenie kodu SQL, który odnosić się będzie do tabeli Article, którą chcemy utworzyć:

|  |
| --- |
| CREATE TABLE "blog\_article" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "title" varchar(100) NOT NULL, "content" text NOT NULL, "date\_posted" datetime NOT NULL, "author\_id" integer NOT NULL REFERENCES "auth\_user" ("id") DEFERRABLE INITIALLY DEFERRED); CREATE INDEX "blog\_article\_author\_id\_905add38" ON "blog\_article" ("author\_id"); COMMIT; |

To właśnie ten kod SQL zostanie za chwilę wykonany, po wpisaniu przez Nas polecenia python manage.py migrate. Oczywiście wcale nie musieliśmy wykonywać polecenia sqlmirate - jak widzisz, służy ono tylko temu, aby podejrzeć kod do utworzenia tabeli. Jednak pokazuje Ci je po to, aby uświadomić Cię, jak wygodne jest korzystanie z ORM-owego podejścia przy tworzeniu kolejnych tabel czy manipulowania danymi bazy danych. My - jako programiści - ograniczyliśmy się jedynie do utworzenia w pliku models.py odpowiedniej klasy wraz z pewnymi polami. Nie musieliśmy budować ręcznie całego polecenia SQL (wyświetlonego po wykonaniu polecenia migrate), ponieważ silnik Django sam o to zadbał. Pełna prostota i wygoda w programowaniu!

* Czas na zaaplikowanie wszystkich zmian i utworzenie nowej tabeli Article. Wykonajmy więc długo wyczekiwane polecenie python manage.py migrate

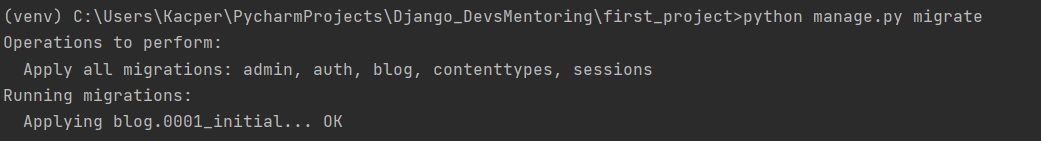
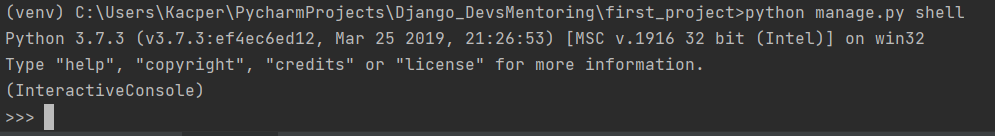


Tabela Article wraz z odpowiednimi kolumnami została utworzona!

**Tryb sandboxowy i zapoznanie się z poleceniami**

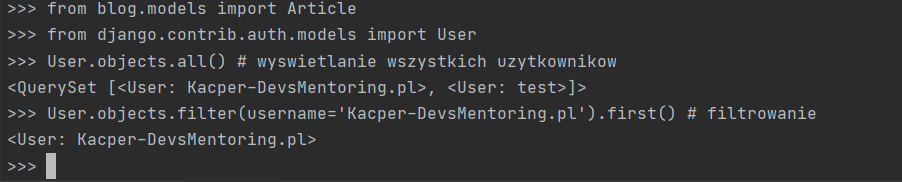
Zanim przejdziemy do dalszego rozwijania Naszej aplikacji, chciałbym zaprezentować Ci wygodny sposób na testowanie i wykonywanie poleceń realizujących pewne operacje w obrębie aplikacji. Skorzystamy z, tzw. Django shell, czyli API frameworka. Z poziomu takiego API będziemy mogli w pełni manipulować wszystkimi informacjami w bazie - czyli, np. dodawać do odpowiednich tabel nowe rekordy, wyświetlać aktualną zawartość bazy itd.

Aby uruchomić shell, wykonajmy polecenie python manage.py shell.

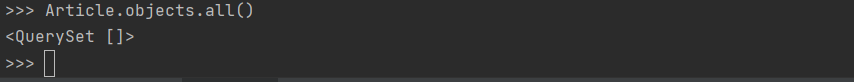


Od tego momentu możemy efektywnie pracować z API.

1. Sprawdźmy, jacy użytkownicy znajdują się w tabelu User oraz czy faktycznie znajduje się w niej utworzony wcześniej użytkownik Kacper-DevsMentoring.pl. Pamiętaj również o odpowiednich importach.



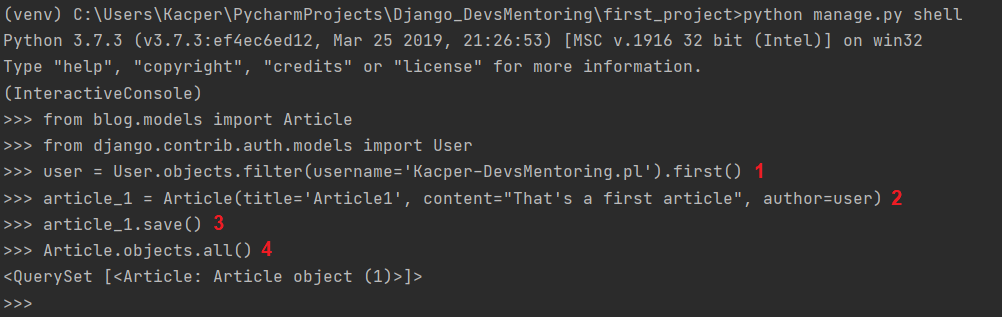
Analogiczne operacje możemy przeprowadzić w odniesieniu do Article.



Co ukazuje, że na ten moment, tabela jest pusta i nie zawiera żadnych rekordów.

1. Dodawanie pierwszego artykułu do tabeli.

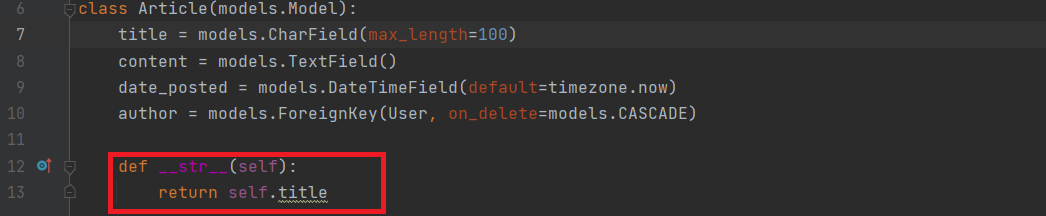
Dodawanie nowych rekordów do tabeli nie jest trudną operacją. Prześledźmy kolejne kroki (pamiętaj o importach!):



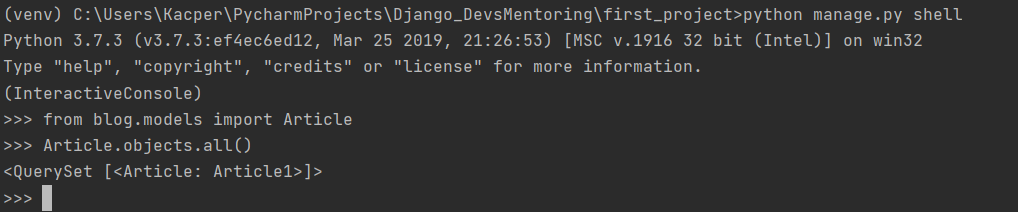
1. W pdpkt. 1 wyłuskaliśmy obiekt Usera o nickname Kacper-DevsMentoring.pl i przypisaliśmy go do zmiennej user. Zostanie on wykorzystany w następnym pdpkt., aby móc dodać go jako klucza obcy w tabeli Article.
2. Pdpkt. 2 to tworzenie obiektu nowego artykułu. Zwróć uwagę na argument author=user. To właśnie on jest referencją do odpowiedniego obiektu użytkownika - autora artykułu.
3. W celu zapisania zmian do tabeli i bazy danych, wywołujemy metodę .save().
4. Linię tę wywołujemy, aby zweryfikować, czy obiekt nowego artykułu, został prawidłowo dodany do tabeli.

Jak zauważyłeś, otrzymaliśmy wynik: <QuerySet [<Article: Article object (1)>]>. Wygląda na to, że prawidłowo utworzyliśmy i dodaliśmy nowy artykuł! Jedyne, co mi rzuca się do modyfikacji, to format, w jakim wyświetlamy zawartość tabeli Article. Zmieńmy Naszą aplikację, tak aby zamiast zapisu: Article object (1), otrzymać bardziej czytelny format artykułu, np. jego tytuł.

W tym celu przeładujemy metodę specjalną \_\_str\_\_() w klasie Article w pliku blog/models.py. Po zmianie, klasa Article, powinna wyglądać następująco:



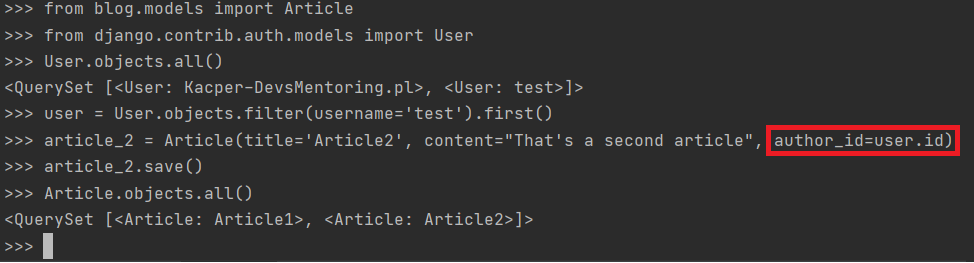
Wyłączmy i włączmy raz jeszcze Nasz shell (aby wyłączyć konsolę, wykonaj kombinację klawiszy CTRL + Z i zatwierdź ją przyciskiem ENTER). Następnie ponownie wykonajmy polecenie Article.objects.all().



Znacznie lepiej! Teraz zamiast <Article: Article object (1)> otrzymaliśmy po prostu <Article: Article1>, gdzie Article1 to tytuł artykułu. Usprawni to weryfikowanie poprawności działania aplikacji przy dodawania kolejnych rekordów.

1. Dodajmy jeszcze 2 następne artykuły do tabeli. Zaprezentuję Ci jednak nieco inne sposoby na ich dodanie.

3.1 Dodanie artykułu z wykorzystaniem ID usera:

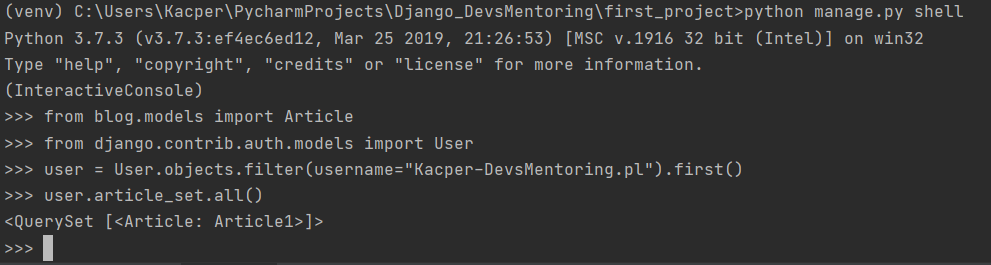


Zwróć uwagę na zaznaczony obszar. Nie zawsze potrzebne jest odwołanie do obiektu konkretnego użytkownika. Powiązania między rekordami w tabelach możemy urzeczywistniać, wykorzystując, np. ID.

3.2 Dodanie nowego artykułu za pośrednictwem obiektu usera

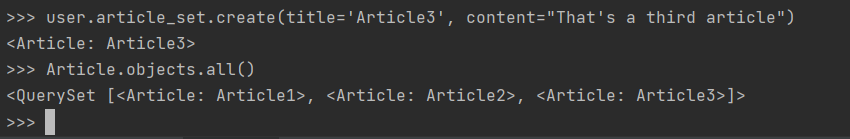
Do tego momentu dodaliśmy dwa nowe artykuły. Pierwszy z nich przypisaliśmy do usera Kacper-DevsMnetoring.pl, natomiast drugi do test. Jak się okazuje, również z poziomu obiektu danego użytkownika, możemy odczytać, w relacji z jakimi artykułami się on znajduje. Każdy obiekt usera przechowuje bowiem pole, np. article\_set, które zawiera referencje do wszystkich rekordów z tabeli, z którą użytkownik jest w relacji.

Sprawdźmy zatem, z jakimi artykułami jest powiązany teraz użytkownik Kacper-DevsMentoring.pl (wykorzystamy polecenie user.article\_set.all()).



Jak możesz zauważyć, rezultatem polecenia user.article\_set.all() jest <QuerySet [<Article: Article1>]>. Zatem wszystko działa zgodnie z przewidywaniami.

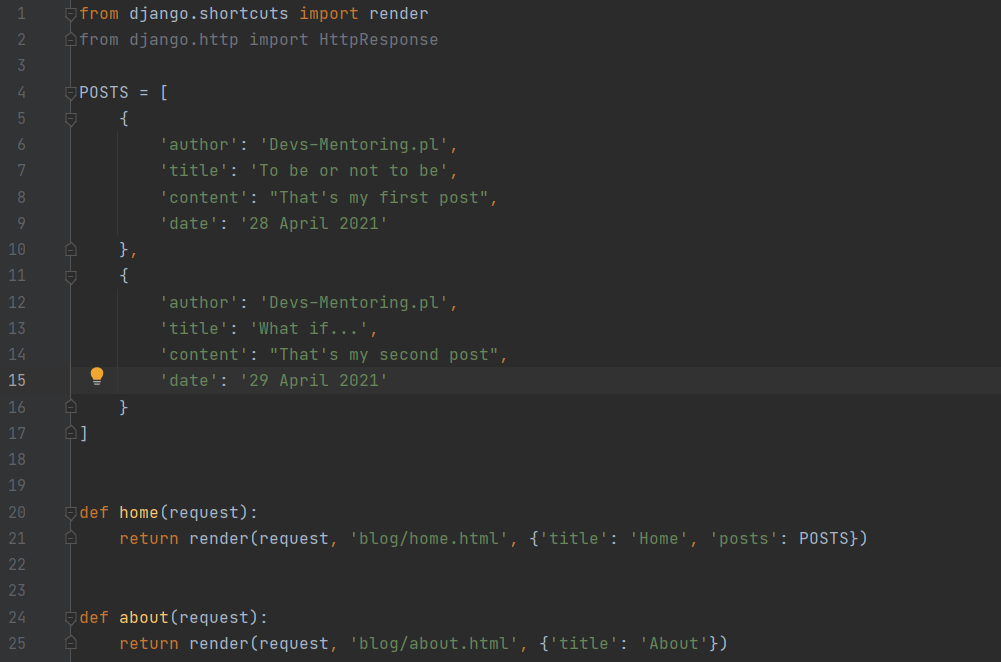
Uzbrojeni w taką wiedzę, możemy teraz utworzyć artykuł nr 3 i bezpośrednio przypisać go do użytkownika Kacper-DevsMentoring.pl. W tym celu należy wykonać polecenie: user.article\_set.create(title=’Article3’, content=”That’s a third article”)



**Wyświetlanie artykułów z bazy**

Mam nadzieję, że praca z powłoką Django nie sprawiła Cię wiele kłopotów i tak jak ja, uważasz ją za przydatne narzędzie w pracy z danymi.

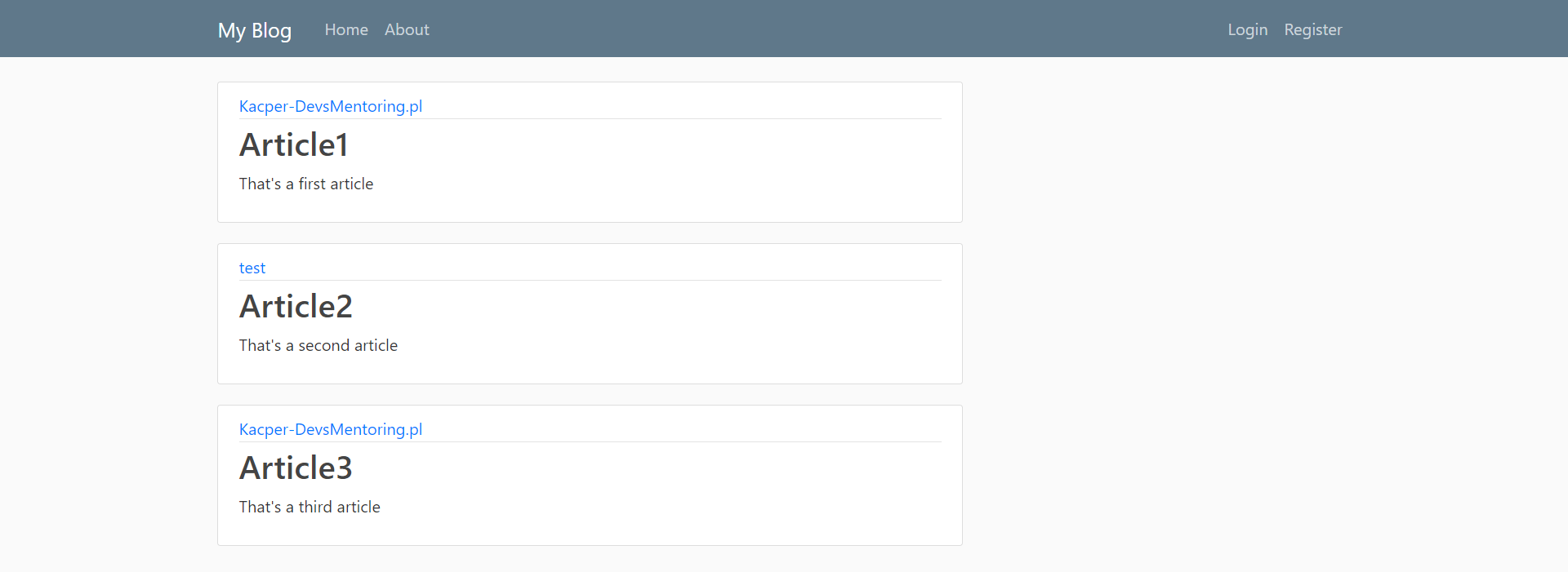
Wróćmy do pliku blog/views.py. Pamiętasz jeszcze, w jaki do tej pory sposób przechowywaliśmy zawartość artykułów? Mieliśmy utworzoną listę POSTS, w której trzymaliśmy słowniki odzwierciedlające każde pojedyncze artykuły.



W celu lekkiej modyfikacji i zmienienia, aby artykuły na stronie pobierane były z utworzonej tabeli Article, zapiszemy (zaznaczony obszar, to snippet kodu, który należało wprowadzić do pliku; pamiętaj również o poprawnych importach!):



W rezultacie, na stronie, otrzymamy:



Działa!

Uważni czytelnicy zauważą jednak pewną nieścisłość... Zwróć uwagę, że przy artykułach brakuje ich daty dodania. Co należy zmienić w kodzie, aby wyświetlana również była prawidłowa data przy każdym z artykułów?

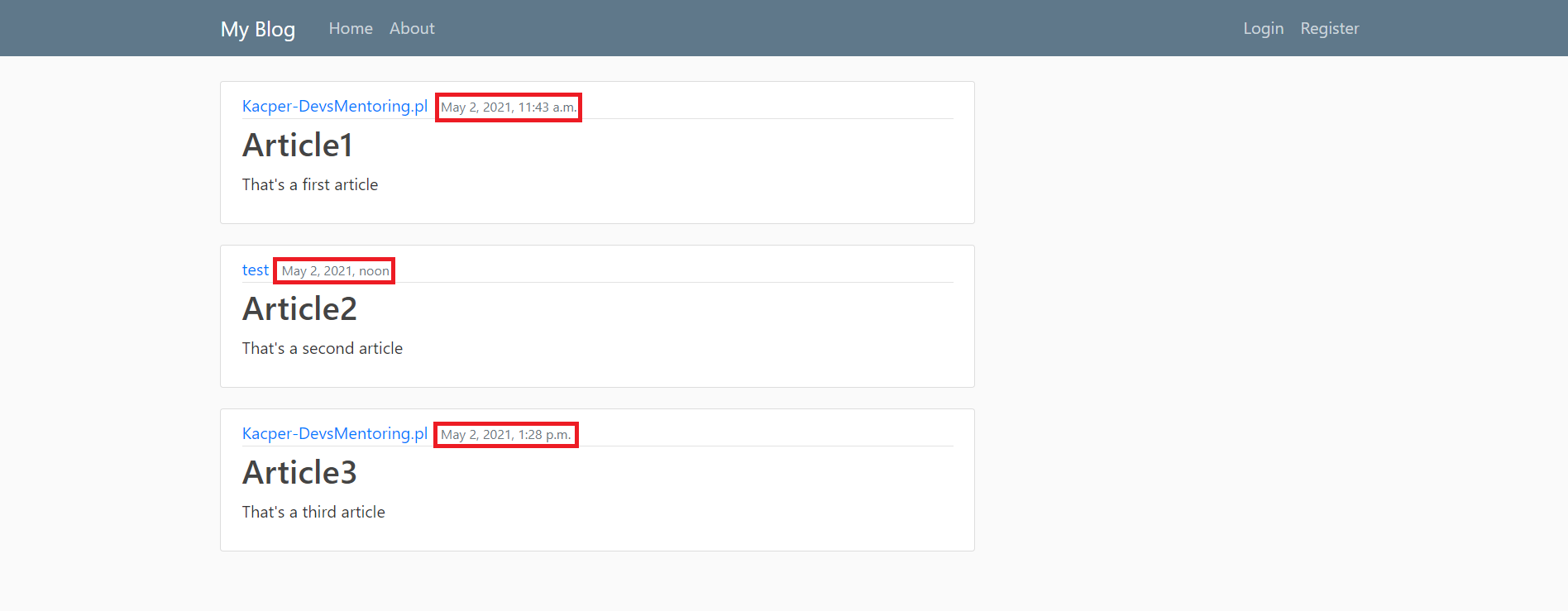
**Podpowiedź**:

Zwróć uwagę, czy w pliku home.html, w prawidłowy sposób odwołujemy się do daty każdego artykułu.

**Odpowiedź:**

W pliku .html do tej pory datę reprezentowaliśmy pod polem date, teraz w wyniku utworzenia nowej tabeli, pole przeobraziło się do date\_posted.

Ostateczny wygląd Naszej aplikacji powinien odzwierciedlać następujący screenshot:

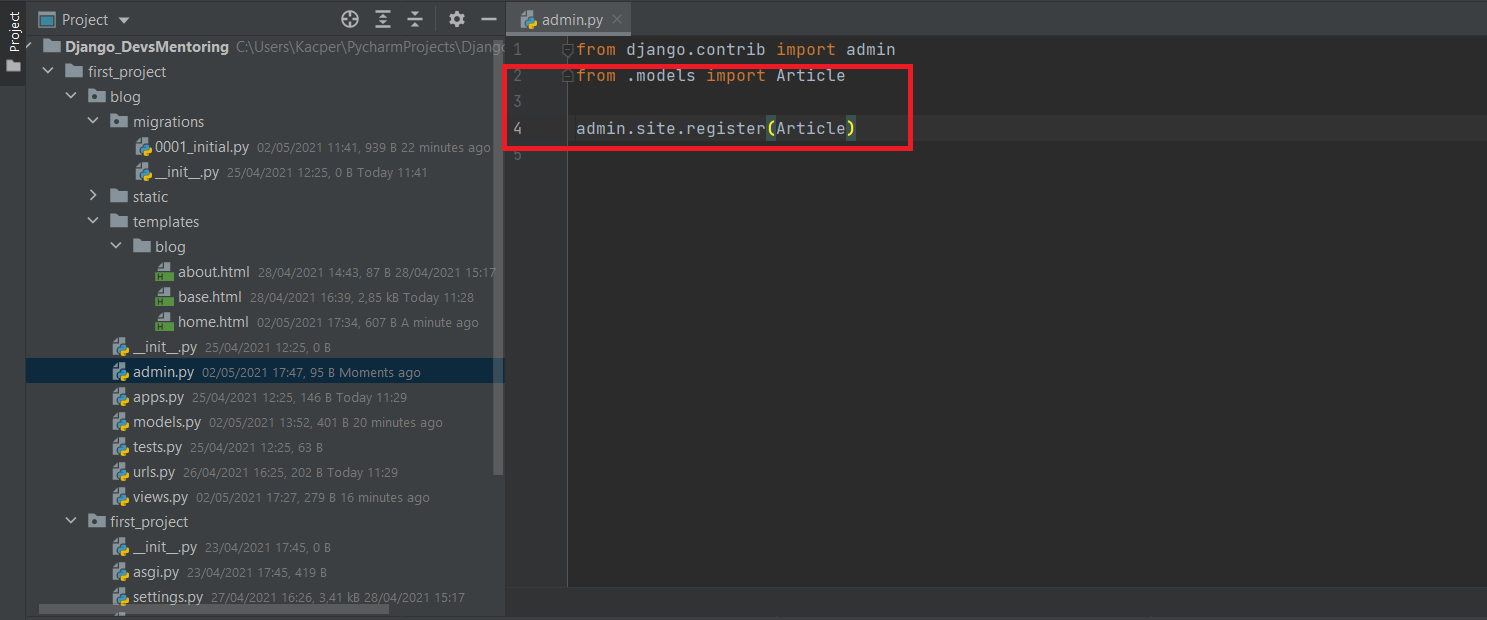


Dodanie panelu do tworzenia nowych rekordów tabeli

Gratulacje! Udało Nam się zintegrować aplikację z bazą danych. Przyznasz jednak, że kosztowało Nas to trochę zachodu, a dodawanie nowych rekordów do danych tabel (z wykorzystaniem shella) jest nieco niepraktyczne i niezbyt wygodne. Dlatego istnieje na to lekarstwo!

Pamiętasz panel administracyjny Naszej aplikacji? Dostawaliśmy się do niego przez endpoint /admin i podanie właściwych danych uwierzytelniających. Do tej pory mieliśmy w nim możliwość interaktywnego dodawania kolejnych grup i użytkowników. Jak zaraz zobaczysz, możemy go również zmodyfikować w taki sposób, iż będziemy mieli interaktywny dostęp do utworzonej tabeli Article, a dodawanie kolejnych obiektów będzie możliwe przez “przeklikanie” w przeglądarce. Mega użyteczne rozwiązanie, a co więcej bardzo proste w implementacji!

Wystarczy bowiem udać się do pliku blog/admin.py i zarejestrować w nim Naszą tabelę.

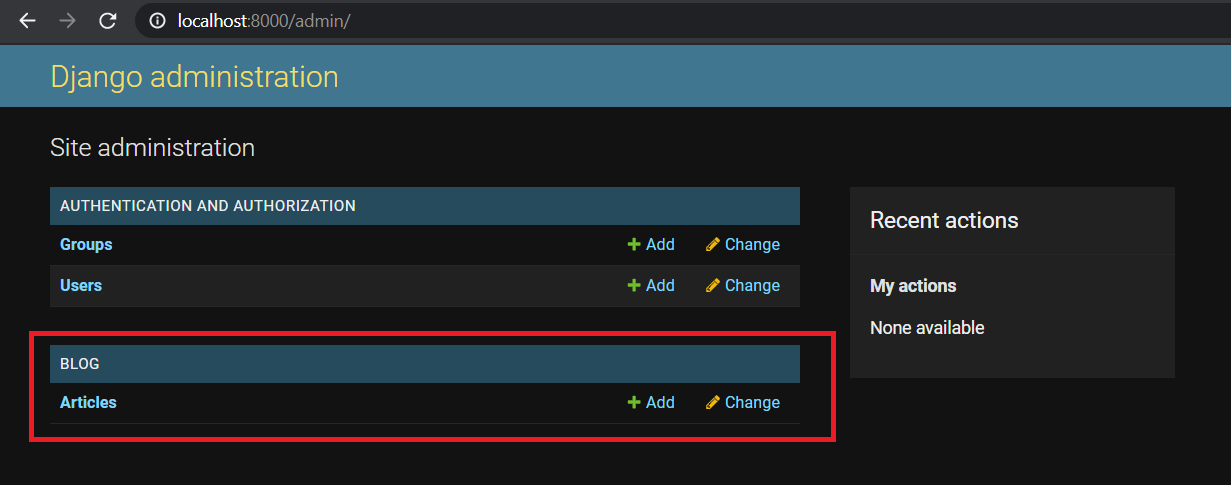


from django.contrib import admin

from .models import Article

admin.site.register(Article)

Przejściu do panelu administracyjnego (czyli podstrony localhost:8000/admin) i zalogowaniu się na swoje konto administratorskie (mam nadzieję, że pamiętasz jeszcze, jakich danych użyłeś do rejestracji :)), powinniśmy ujrzeć w panelu dodatkową pozycję - Articles.



To właśnie z jej poziomu możemy bardzo łatwo dodawać nowe artykuły, czy zarządzać już istniejącymi.

