Jan Niedziółka

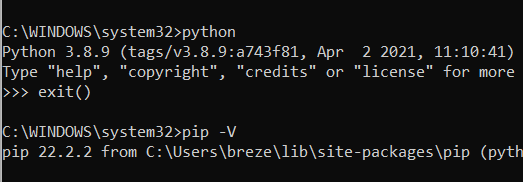
**TECHNOLOGIE PROGRAMISTYCZNE – SYSTEMY INTERNETOWE**

**Czym jest Django?**

Django to wolny i otwarty framework Pythona przeznaczony do tworzenia dynamicznych aplikacji internetowych z renderowaniem po stronie serwera. Pozwala na budowę bardziej złożonych projektów niż lżejszy Flask, czyli inny popularny framework Pythona o podobnych zastosowaniach. Inne technologie które do pewnego stopnia są zbliżone do Django to m.in. Node.js/Express, Spring Web Flow czy Spring Boot. Django wyróżnia się dużą liczbą wbudowanych mechanizmów, pakietów i komponentów obsługujących powszechne aspekty takich aplikacji, m.in. systemy szablonów, formularze, uwierzytelnienie, autoryzacja, wsparcie dla zabezpieczeń CSRF, SQL injection i innych, obsługę popularnych baz danych, wsparcie dla wzorców architektonicznych takich jak model-template-view itd.

**Instalacja Django**

1. Uruchom wiersz poleceń i sprawdź czy masz zainstalowanego Pythona wpisując „**python**” lub (szczególnie na Linuxie) „**python3**”:



Uruchomi to skrypt Pythona z którego należy wyjść wpisując „**exit()**”. Warto również upewnić się czy zainstalowany jest pip, którego w przypadku starszych wersji może brakować. Jeśli nie zostaną wyświetlone informacje o wersji należy zainstalować Pythona pobierając jedną z wersji stąd: <https://www.python.org/downloads/windows/> i przechodząc przez kolejne kroki instalacji. Od wersji 2.5.x Python jest instalowany razem z lekką bazą danych SQLite która wystarczy na potrzeby tego ćwiczenia.

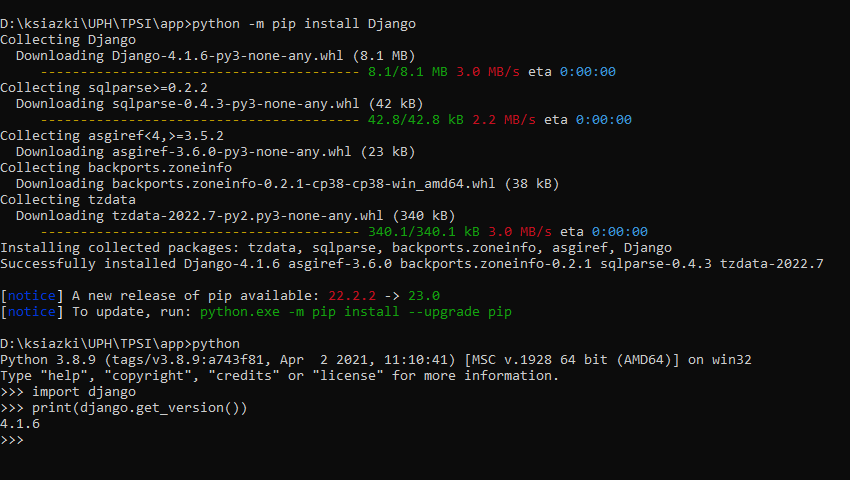
1. Z wiersza poleceń (uruchomionego w trybie administratora) zainstaluj Django poleceniem:  
   **python -m pip install Django**

a następnie zweryfikuj instalację wpisując:

**python**

**import django**

**print(django.get\_version())**

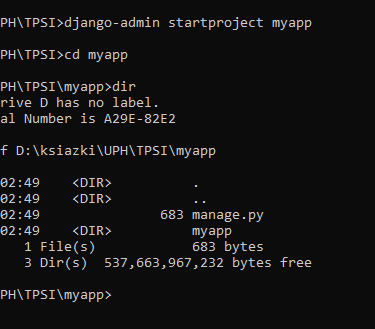


**exit()**

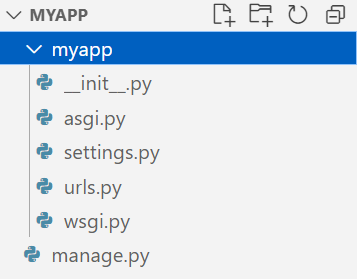
Jeśli instalacja przebiegła pomyślnie, wyświetlona będzie wersja Django. Alternatywą jest polecenie:  
**python -m django --version**

**Zbudowanie pierwszego projektu**

Przechodzimy do folderu w którym chcemy zbudować nasz pierwszy projekt i używamy polecenia:  
**django-admin startproject <nazwa>**



W ten sposób zostały utworzone następujące pliki:



Plik manage.py – skrypt wiersza poleceń umożliwiający zarządzanie projektem Django, np. uruchamianie serwera deweloperskiego, wykonywanie migracji bazy danych itp.

Katalog myapp zawierający:  
- init.py - jest to pusty plik, który oznacza, że katalog jest katalogiem Pythona.  
- asgi.py - jest to plik umożliwiający korzystanie z aplikacji Django jako aplikacji ASGI (Asynchronous Server Gateway Interface).  
- settings.py - jest to plik konfiguracyjny projektu Django, w którym znajdują się ustawienia takie jak DEBUG, INSTALLED\_APPS, DATABASES itp.  
- urls.py - jest to plik zawierający mapowanie adresów URL aplikacji na widoki.  
- wsgi.py - jest to plik umożliwiający korzystanie z aplikacji Django jako aplikacji WSGI (Web Server Gateway Interface).

**Zbudowanie prostego widoku, routing, rendering szablonu**

1. Otwórz plik urls.py w katalogu myapp

Dodaj do niego dwa importy:

**from django.urls import path**

**from . import views**

do urlpatterns dodaj ścieżkę:

**path('', views.index, name='index'),**

Ostatecznie plik powinien wyglądać tak:

Widok z VSC, pokazanie jak pisać i formatować kod pythona i jak wyglądają pliki Django.
Plik zaczyna się od komentarza, Django inicjalizuje swoje pliki z pobieżną instrukcją 

Pamiętaj, że w Pythonie bardzo ważne są wcięcia (taby)! Bez nich program nie będzie dobrze działać.

1. Utwórz plik views.py w katalogu myapp

Dodaj następujący kod:

from django.shortcuts import render

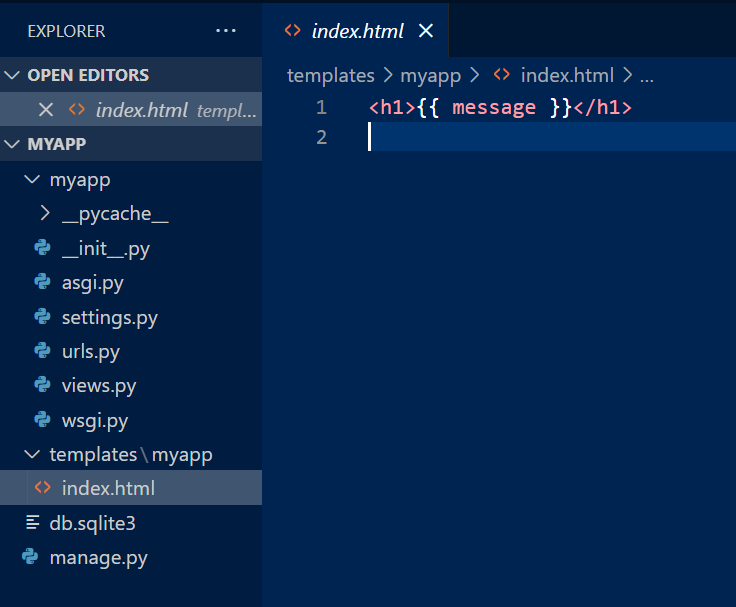
def index(request):

    return render(request, 'myapp/index.html', {'message': 'Hello World!'})

1. W katalogu głównym aplikacji (tym który mieści katalog myapp i plik manage.py) dodaj katalog templates, w nim katalog myapp, a w nim plik index.html.

W pliku index.html umieść linię:

**<h1>{{ message }}</h1>**



1. Otwórz plik settings.py w katalogu myapp

Na początku pliku zaimportuj:

**import os**

Następnie znajdź sekcję TEMPLATES i pustą linijkę ‘DIRS’ podmień na:

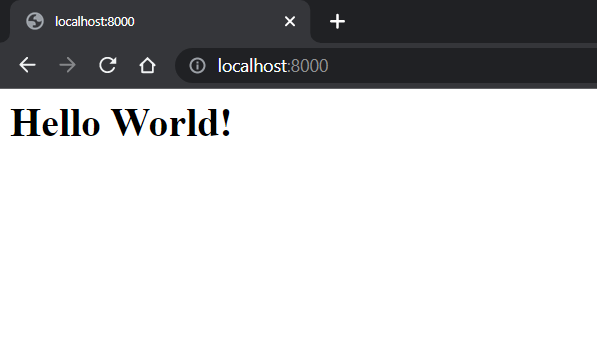
        'DIRS': [os.path.join(BASE\_DIR, 'templates')],

1. Z poziomu wiersza poleceń lub terminala VSC uruchom serwer poleceniem:

**python manage.py runserver**

Uwaga: na niektórych systemach/wersjach Django w tym miejscu i w kolejnych krokach zamiast „**python**” należy używać „**py**”.

Po przejściu w przeglądarce na adres <http://localhost:8000> zobaczysz wiadomość „Hello World!”:



**Dodanie szablonu ze zmiennymi i dynamiczną zawartością**

1. Edytuj plik

templates/myapp/index.html:

<!DOCTYPE html>

<html>

<head>

  <title>MyApp</title>

</head>

<body>

  <h1>{{ message }}</h1>

  <p>Dzisiaj jest {{ current\_date }}</p>

</body>

</html>

1. Edytuj plik

Myapp/views.py:

from django.shortcuts import render

from django.http import HttpResponse

import datetime

def index(request):

    now = datetime.datetime.now()

    context = {

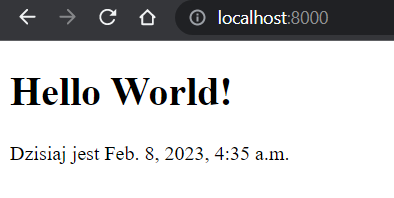
        'message': 'Hello World!',

        'current\_date': now

    }

    return render(request, 'myapp/index.html', context)

Na localhost:8000 można zobaczyć:



**Połączenie z bazą danych SQLite, utworzenie modelu**

1. Upewnij się, czy BD jest skonfigurowana w pliku settings.py

Odnajdź sekcję „Databases”, w przypadku SQLite powinna wyglądać na przykład tak:

DATABASES = {

    'default': {

        'ENGINE': 'django.db.backends.sqlite3',

        'NAME': BASE\_DIR / 'db.sqlite3',

    }

}

Engine określa jakiej bazy danych używamy, inne wartości które można tu wpisać to np. 'django.db.backends.postgresql' czy 'django.db.backends.mysql'.  
Name określa nazwę bazy (plik który się utworzy) wraz ze ścieżką, **BASE\_DIR /**oznacza, że plik zostanie utworzony w katalogu głównym aplikacji.

1. Stwórz bazę danych poleceniem w konsoli:

**python manage.py migrate**

To polecenie stworzy plik o nazwie podanej jako wartość atrybutu NAME, w wierszu poleceń powinno wyglądać to mniej więcej tak:

Widok z CMD pokazujący efekt polecenia migrate:
"Running migrations:" i lista inicjalizowanych modułów

Jeśli plik bazy danych się utworzył, a polecenie **python manage.py migrate**zwraca komunikat podobny do tego:

Widok z CMD w przypadku nieudanej inicjalizacji bazy danych:
"Running migrations: no migrations to apply" 

To należy usunąć plik bd poleceniem **del "db.sqlite3"** (oczywiście konieczne jest ustawienie się w cmd we właściwym katalogu), a potem jeszcze raz użyć „migrate”.

1. W myapp/myapp stwórz plik models.py, w nim będą definiowane modele

Stwórzmy model Contact którego użyjemy później do dodawania danych kontaktowych podanych w formularzu do bazy danych. Do utworzonego pliku models.py należy dodać ciało modelu:

from django.db import models

class Contact(models.Model):

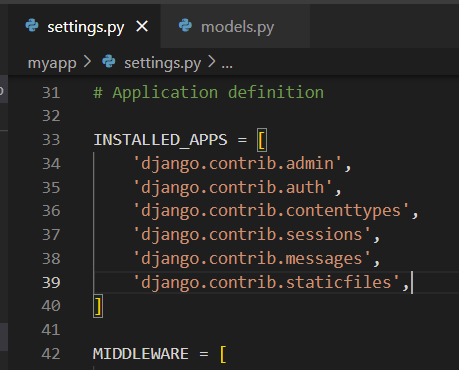
    name = models.CharField(max\_length=100)

    email = models.EmailField()

    message = models.TextField()

    created\_at = models.DateTimeField(auto\_now\_add=True)

W pliku settings.py należy dodać aplikację do dziedziny (scope) kompilacji. Wyszukajmy sekcję INSTALLED\_APPS, pierwotnie wygląda ona tak:



Dodaj nazwę swojej aplikacji do listy obsługiwanych modułów:

INSTALLED\_APPS = [

    'django.contrib.admin',

    'django.contrib.auth',

    'django.contrib.contenttypes',

    'django.contrib.sessions',

    'django.contrib.messages',

    'django.contrib.staticfiles',

    'myapp',

]

Teraz użyj polecenia **python manage.py makemigrations**  
Wyświetli się informacja o utworzeniu modelu:

Widok z CMD potwierdzający utworzenie modelu:
"Migrations for 'myapp' :
- Create model Contact"

Jeśli zamiast tego wyświetla się komunikat „No changes detected” to spróbuj użyć tego polecenia wraz z nazwą aplikacji, np. **python manage.py makemigrations myapp**Jeśli to nie zadziała, należy kolejno:  
- sprawdzić czy w pliku settings.py w sekcji INSTALLED\_APPS dodano aplikację  
- usunąć plik bazy danych  
- jeszcze raz użyć polecenia „migrate”  
- jeszcze raz użyć polecenia „makemigrations”

Jeśli proces tworzenia modelu się powiódł, należy jeszcze raz użyć polecenia  
**python manage.py migrate**

Przeprowadzenie całej procedury poprawnie zostanie potwierdzone w wierszu poleceń komunikatem „Applying myapp.0001\_initial…”:

Widok z CMD:
"Running migrations:
Applying OK"

**Tworzenie formularza dodanego do widoku z danymi dodawanymi do bazy danych**

1. Stwórz formularz

W pliku **myapp/forms.py** dodaj następujący kod:

from django import forms

class ContactForm(forms.Form):

    name = forms.CharField(label='Your Name', max\_length=100)

    email = forms.EmailField(label='Your Email')

    message = forms.CharField(widget=forms.Textarea)

1. Stwórz widok

W pliku **myapp/views.py**, pod definicją widoku indexdodaj:

def success\_view(request):

    return render(request, 'myapp/success\_view.html')

from django.shortcuts import render, redirect

from .forms import ContactForm

def contact\_form\_view(request):

    if request.method == 'POST':

        form = ContactForm(request.POST)

        if form.is\_valid():

            # Pobierz dane z formularza i przetwórz je

            name = form.cleaned\_data['name']

            email = form.cleaned\_data['email']

            message = form.cleaned\_data['message']

            #Przetwórz dane zgodnie z potrzebami aplikacji

            return redirect('success\_view')

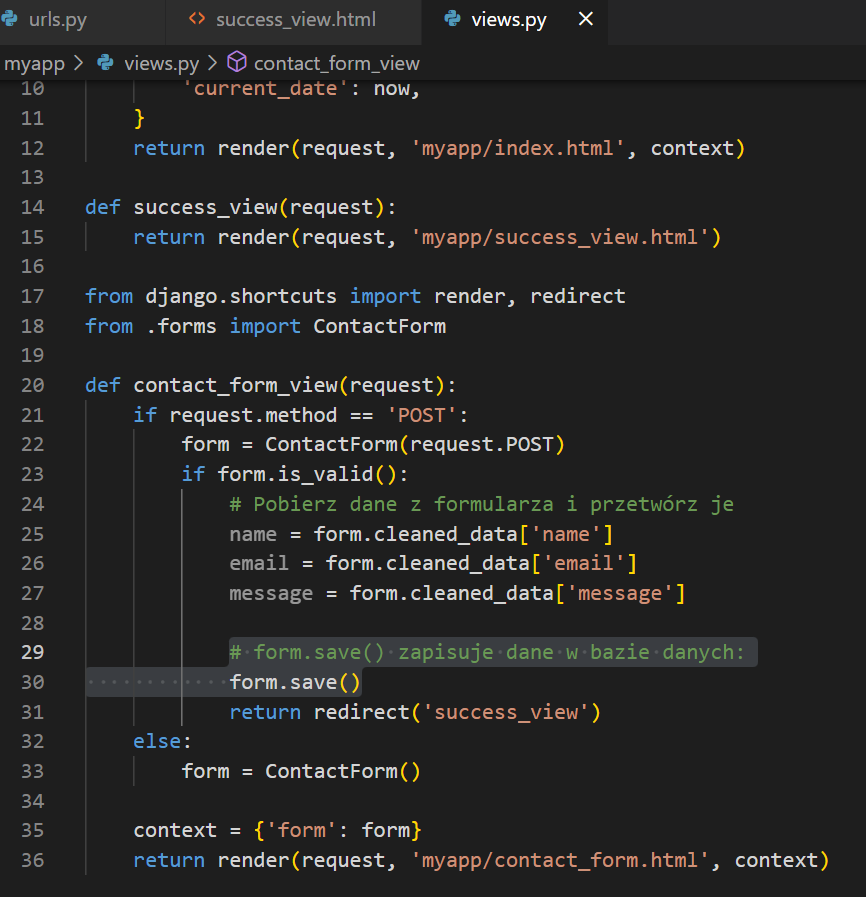
    else:

        form = ContactForm()

    context = {'form': form}

    return render(request, 'myapp/contact\_form.html', context)

Ponieważ chcemy, żeby dane były zapisywane do BD, użyjemy metody form.save() którą należy dodać nad „return redirect”. Finalnie plik **views.py** powinien wyglądać tak:



1. Stwórz szablon

W katalogu **templates/myapp** dodaj:

<form method="post">

    {% csrf\_token %}

    {{ form.as\_p }}

    <input type="submit" value="Submit">

  </form>

<a href="/">Strona główna</a>

Do pliku **templates/index.html** możemy dodać link do formularza:

<a href="/contact">Formularz kontaktowy</a>

W przeciwnym razie musielibyśmy ręcznie podawać adres localhost:8000/contact/

Dodaj też landing page która zostanie wyświetlona po poprawnym przetworzeniu danych z formularza, czyli **templates/success\_view.html**:

<!DOCTYPE html>

<html>

<head>

  <title>Sukces</title>

</head>

<body>

  <h1>Dane zostały dodane do bazy!</h1>

  <a href="/">Strona główna</a>

  <a href="/contact">Formularz</a>

</body>

</html>

Aby takie link działały, musimy jednak najpierw dopasować URL z pliku **myapp/urls.py**

Do listy istniejących **urlspatterns** dodaj nowe:

urlpatterns = [

    path('admin/', admin.site.urls),

    path('', views.index, name='index'),

    path('contact/', views.contact\_form\_view, name='contact\_form'),

    path('success/', views.success\_view, name='success\_view'),

]

**Interfejs Django admin**

Upewnij się, że aplikacja admin jest dodana do projekcie w sekcji INSTALLED\_APPS w pliku **settings.py**

INSTALLED\_APPS = [

    'django.contrib.admin',

Następnie dodaj plik **admin.py**

from django.contrib import admin

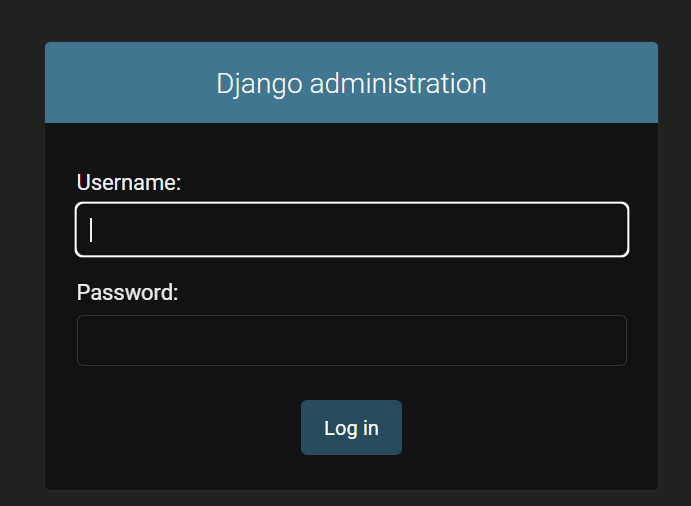
from .models import Contact

admin.site.register(Contact)

admin.site.register() odpowiada za rejestrację modelu w pliku.  
Teraz można stworzyć konto admina którego użyjesz do zarządzania swoją aplikacją. W cmd/terminalu uruchom polecenie:

**python manage.py createsuperuser**

Podaj login, hasło (wystarczy admin admin) i maila. Teraz możesz uruchomić serwer (**py manage.py runserver**) i wejść na stronę localhost:8000/admin/  
Wyświetli ci się interfejs Django admin, podaj dane użyte do rejestracji aby się zalogować i otrzymać dostęp do podstawowych funkcji admina.



Aby dodać customowe funkcje edytuj plik **myapp/admin.py**

Możesz dodać na przykład metody do klasy opisującej model i oznaczyć je jako akcję:

def mark\_as\_read(modeladmin, request, queryset):

    queryset.update(is\_read=True)

mark\_as\_read.short\_description = "Oznacz jako przeczytane"

class ContactAdmin(admin.ModelAdmin):

    actions = [mark\_as\_read]

admin.site.register(Contact, ContactAdmin)

Tutaj `mark\_as\_read` ustawia pole `is\_read` na `True` dla wybranych rekordów.  
Aby dostosować interfejs admina, można dostosować klasę opisującą model w pliku **admin.py**. Pamiętaj o rejestracji klasy poprzez metodę register().

Edytuj ContactAdmin:

class ContactAdmin(admin.ModelAdmin):

    list\_display = ['name', 'email', 'created\_at']

    list\_filter = ['created\_at']

    search\_fields = ['name', 'email', 'message']

    actions = [mark\_as\_read]

Finalnie:

from django.contrib import admin

from .models import Contact

def mark\_as\_read(modeladmin, request, queryset):

    queryset.update(is\_read=True)

mark\_as\_read.short\_description = "Oznacz jako przeczytane"

class ContactAdmin(admin.ModelAdmin):

    list\_display = ['name', 'email', 'created\_at']

    list\_filter = ['created\_at']

    search\_fields = ['name', 'email', 'message']

    actions = [mark\_as\_read]

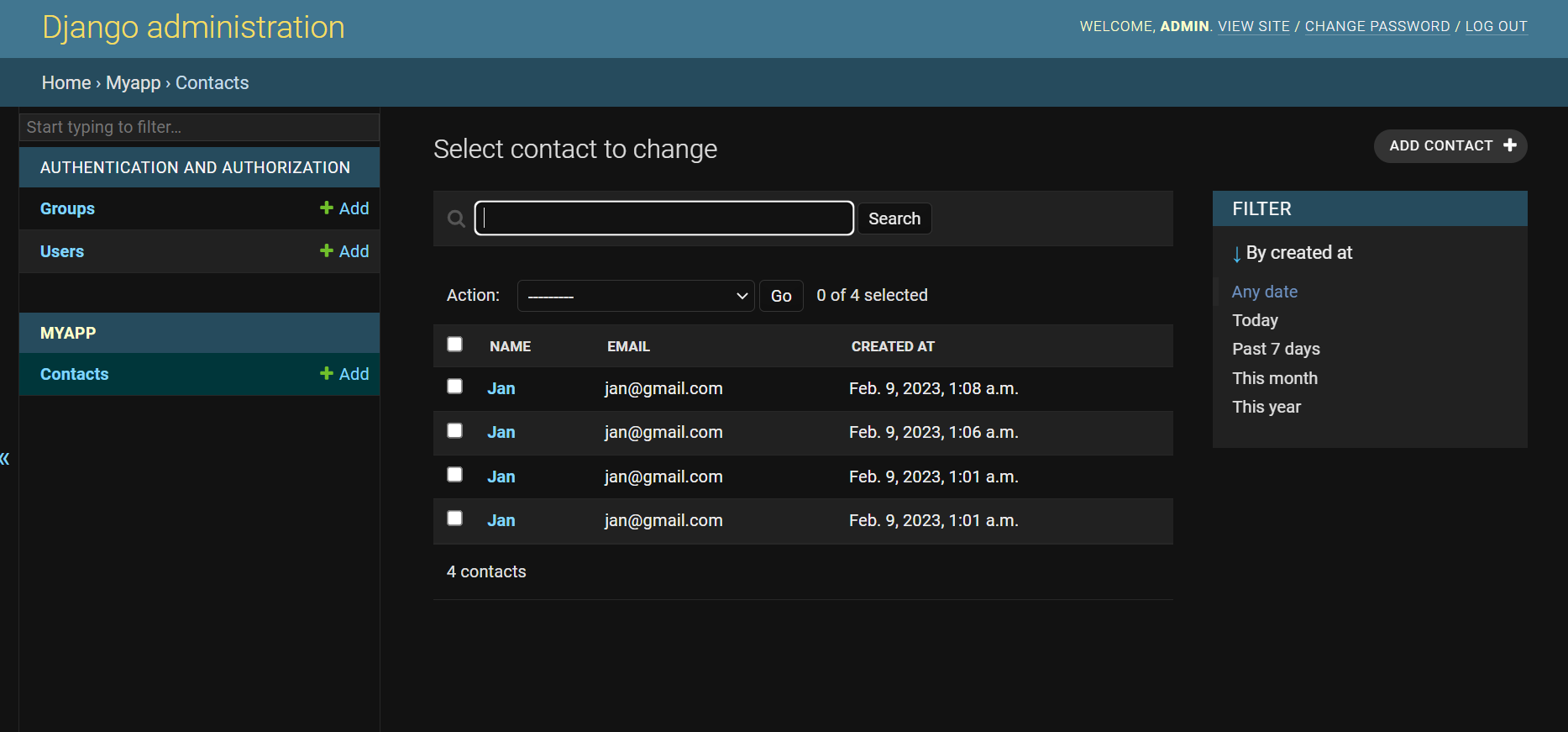
admin.site.register(Contact, ContactAdmin)

Zwróć uwagę na

class ContactAdmin(admin.ModelAdmin):

    list\_display = ['name', 'email', 'created\_at']

Ten fragment pozwala ci na przeglądanie modelu Contact w interfejsie Django Admin. Możesz przeglądać i edytować dane w tym modelu:



**Przykładowe zadanie do samodzielnego wykonania**

Zbuduj niestandardowy widok, który korzysta z danych z bazy danych i wyświetla je na stronie internetowej. To zadanie powinno obejmować bardziej zaawansowane koncepcje w Django, takie jak korzystanie z widoków opartych na klasach i widoków generycznych.

Inne przykładowe zadania mogą obejmować integrację z JavaScript i umożliwienie np. obsługi AJAX do dynamicznego odświeżania stron.