

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**”:

```
C:\WINDOWS\system32>python
Python 3.8.9 (tags/v3.8.9:a743f81, Apr  2 2021, 11:10:41)
Type "help", "copyright", "credits" or "license" for more
>>> exit()

C:\WINDOWS\system32>pip -V
pip 22.2.2 from C:\Users\breze\lib\site-packages\pip (python 3.8.9)
```

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.

2. 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())
```

```
D:\ksiazki\UPH\TPSI\app>python -m pip install Django
Collecting Django
  Downloading Django-4.1.6-py3-none-any.whl (8.1 MB)
----- 8.1/8.1 MB 3.0 MB/s eta 0:00:00
Collecting sqlparse>=0.2.2
  Downloading sqlparse-0.4.3-py3-none-any.whl (42 kB)
----- 42.8/42.8 kB 2.2 MB/s eta 0:00:00
Collecting asgiref<4,>=3.5.2
  Downloading asgiref-3.6.0-py3-none-any.whl (23 kB)
Collecting backports.zoneinfo
  Downloading backports.zoneinfo-0.2.1-cp38-cp38-win_amd64.whl (38 kB)
Collecting tzdata
  Downloading tzdata-2022.7-py2.py3-none-any.whl (340 kB)
----- 340.1/340.1 kB 3.0 MB/s eta 0:00:00
Installing collected packages: tzdata, sqlparse, backports.zoneinfo, asgiref, Django
Successfully installed Django-4.1.6 asgiref-3.6.0 backports.zoneinfo-0.2.1 sqlparse-0.4.3 tzdata-2022.7

[notice] A new release of pip available: 22.2.2 -> 23.0
[notice] To update, run: python.exe -m pip install --upgrade pip

D:\ksiazki\UPH\TPSI\app>python
Python 3.8.9 (tags/v3.8.9:a743f81, Apr  2 2021, 11:10:41) [MSC v.1928 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import django
>>> print(django.get_version())
4.1.6
>>>
```

```
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>

```
PH\TPSI>django-admin startproject myapp

PH\TPSI>cd myapp

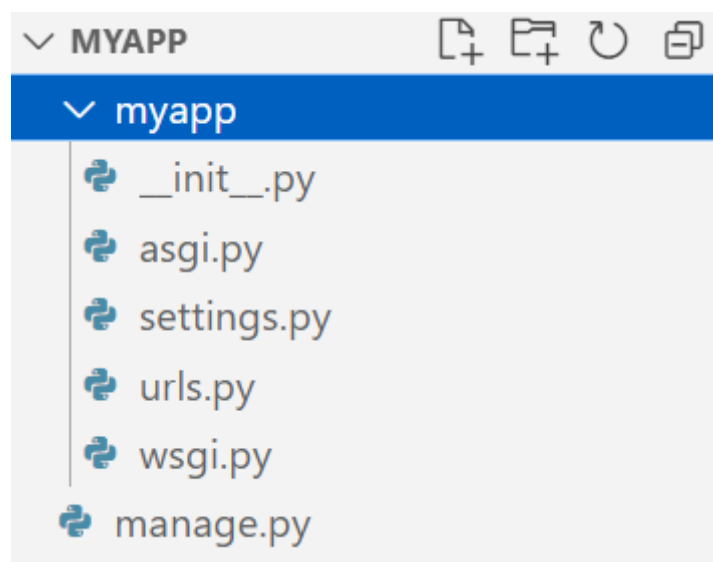
PH\TPSI\myapp>dir
Drive D has no label.
Serial Number is A29E-82E2

Volume in drive D: \ksiazki\UPH\TPSI\myapp

02:49    <DIR>          .
02:49    <DIR>          ..
02:49                683 manage.py
02:49    <DIR>          myapp
1 File(s)                683 bytes
3 Dir(s)  537,663,967,232 bytes free

PH\TPSI\myapp>
```

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:

```
14      2. Add a URL to urlpatterns: path('blog/  
15      ""  
16      from django.contrib import admin  
17      from django.urls import path  
18  
19      from django.urls import path  
20      from . import views  
21  
22      urlpatterns = [  
23          path('admin/', admin.site.urls),  
24          path('', views.index, name='index'),  
25      ]  
26
```

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

2) Utwórz plik views.py w katalogu myapp

Dodaj następujący kod:

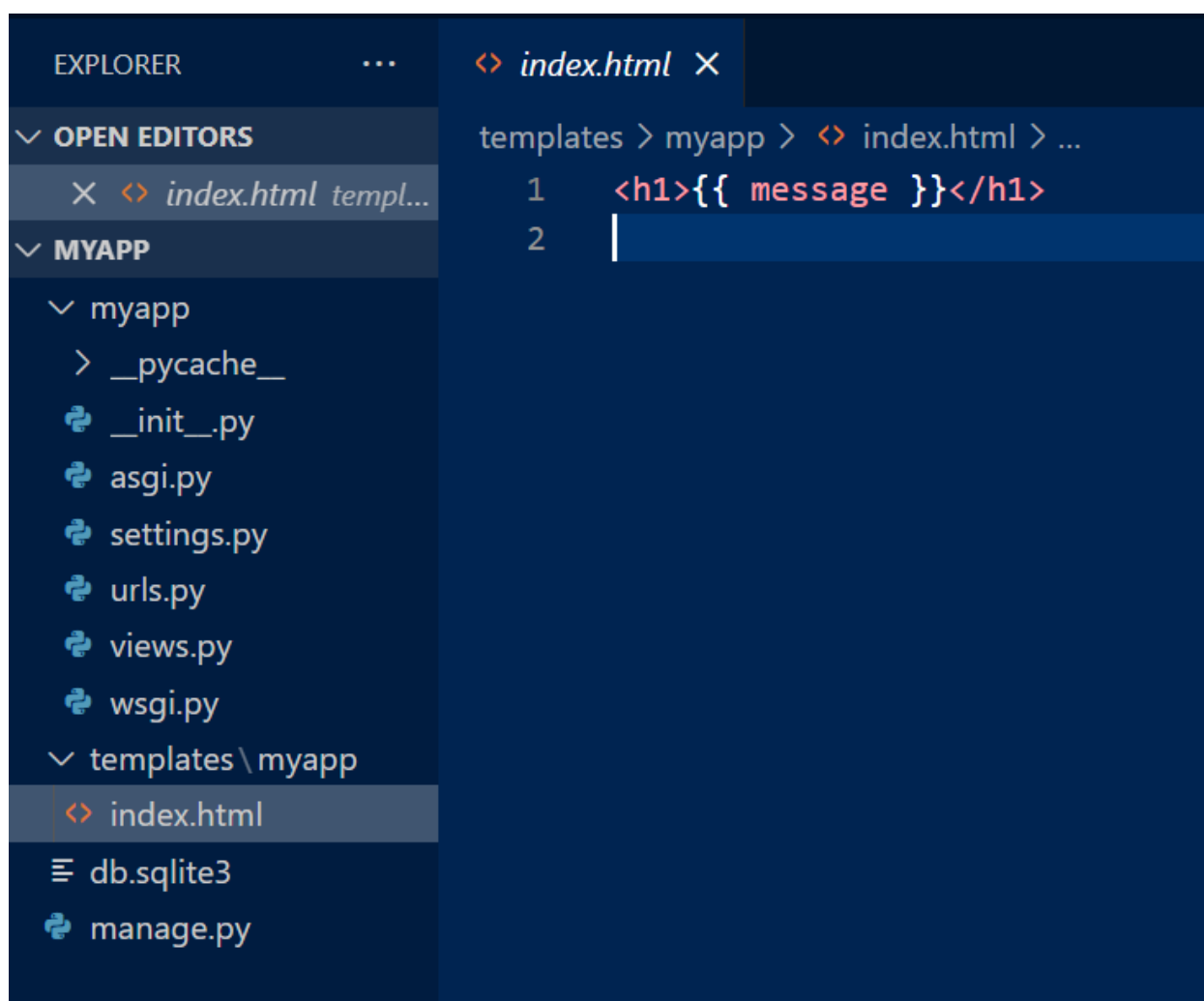
```
from django.shortcuts import render

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

3) 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>



4) Otwórz plik settings.py w katalogu myapp

Na początku pliku zaimportuj:

import os

Następnie znajdź sekcję TEMPLATES i pustą linię 'DIRS' podmień na:

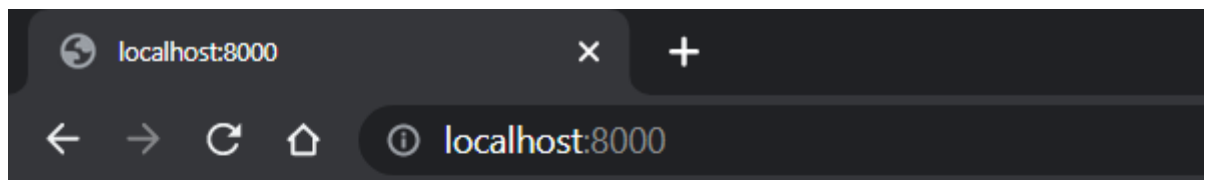
```
'DIRS': [os.path.join(BASE_DIR, 'templates')],
```

5) 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!”:



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>
```

2) 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:



Hello World!

Dzisiaj jest Feb. 8, 2023, 4:35 a.m.

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.

2) 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:

```
D:\ksiazki\UPH\TPSI\myapp>py manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying auth.0012_alter_user_first_name_max_length... OK
  Applying sessions.0001_initial... OK
D:\ksiazki\UPH\TPSI\myapp>py manage.py makemigrations
```


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

```
D:\ksiazki\UPH\TPSI\myapp>python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions
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”.

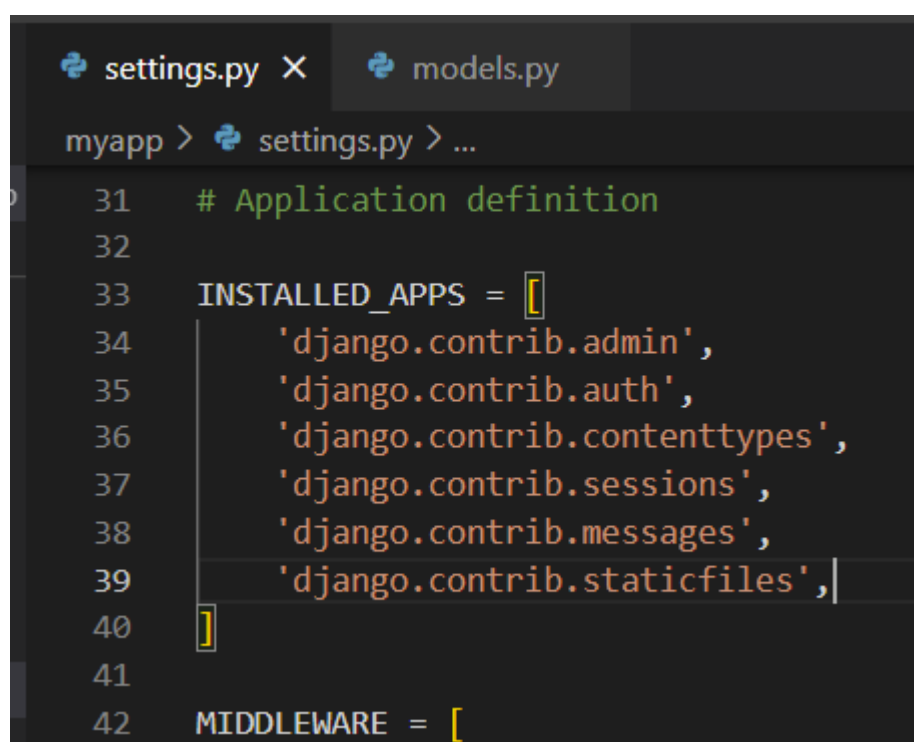
3) W myapp/myapp stwórz plik models.py, w nim będą definiowane modele

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:



```
myapp > settings.py > ...

31 # Application definition
32
33 INSTALLED_APPS = [
34     'django.contrib.admin',
35     'django.contrib.auth',
36     'django.contrib.contenttypes',
37     'django.contrib.sessions',
38     'django.contrib.messages',
39     'django.contrib.staticfiles',
40 ]
41
42 MIDDLEWARE = [
```

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:

```
D:\ksiazki\UPH\TPSI\myapp>py manage.py makemigrations  
Migrations for 'myapp':  
  myapp\migrations\0001_initial.py  
    - Create model Contact  
  
D:\ksiazki\UPH\TPSI\myapp>_
```

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...”:

```
D:\ksiazki\UPH\TPSI\myapp>py manage.py migrate  
Operations to perform:  
  Apply all migrations: admin, auth, contenttypes, myapp, sessions  
Running migrations:  
  Applying myapp.0001_initial... 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)
```

2) Stwórz widok

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

```
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:

```
urls.py    success_view.html    views.py X
myapp > views.py > contact_form_view
10         'current_date': now,
11     }
12     return render(request, 'myapp/index.html', context)
13
14     def success_view(request):
15         return render(request, 'myapp/success_view.html')
16
17     from django.shortcuts import render, redirect
18     from .forms import ContactForm
19
20     def contact_form_view(request):
21         if request.method == 'POST':
22             form = ContactForm(request.POST)
23             if form.is_valid():
24                 # Pobierz dane z formularza i przetwórz je
25                 name = form.cleaned_data['name']
26                 email = form.cleaned_data['email']
27                 message = form.cleaned_data['message']
28
29                 # form.save() zapisuje dane w bazie danych:
30                 form.save()
31                 return redirect('success_view')
32             else:
33                 form = ContactForm()
34
35         context = {'form': form}
36         return render(request, 'myapp/contact_form.html', context)
```

3) 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 **urlpatterns** 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 projektu w sekcji `INSTALLED_APPS` w pliku **settings.py**

```
INSTALLED_APPS = [  
    'django.contrib.admin',
```

Następnie dodaj plik **admin.py**

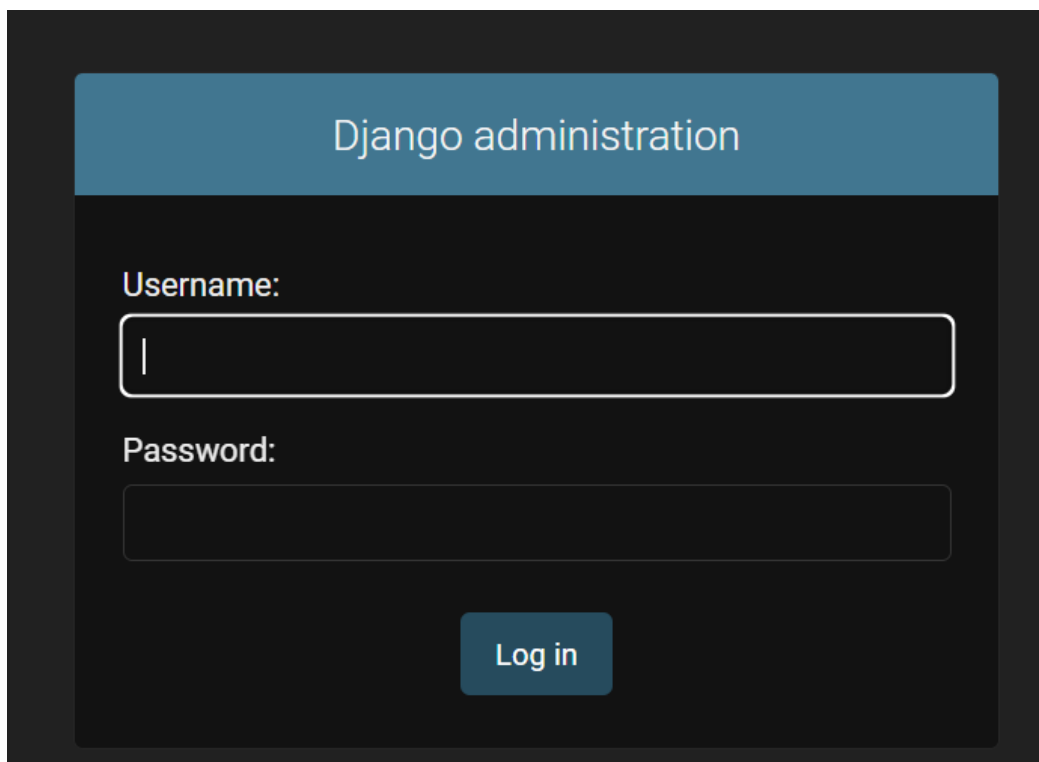
```
from django.contrib import admin  
from .models import Contact  
  
admin.site.register(Contact)
```

`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.



Django administration

Username:

Password:

Log in

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"

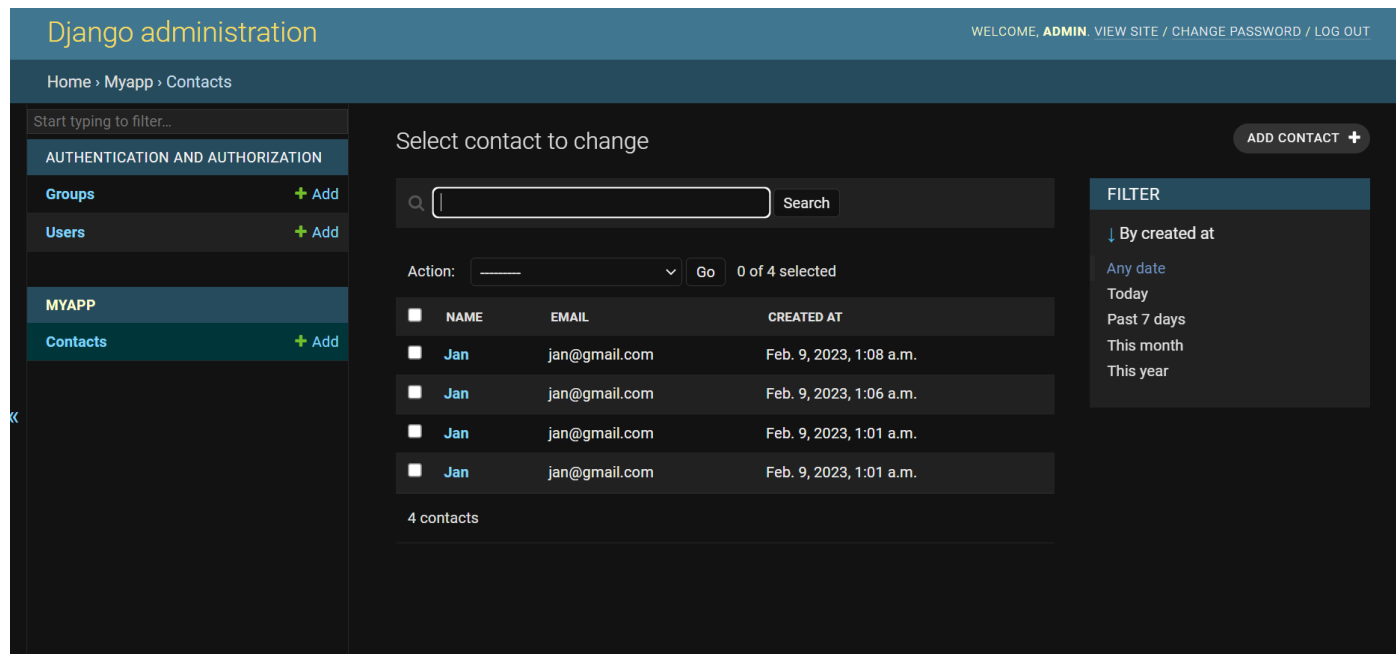
class ContactAdmin(admin.ModelAdmin):
    list_display = ['name', 'email', 'created_at']
    list_filter = ['created_at']
    search_fields = ['name', 'email', 'message']
    actions = [mark_as_read]

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.