Django i Django Software Foundation

SWOBODA TWORZENIA

http://www.sxc.hu/

Rozmawiamy z jednym z autorów projektu Django o powstaniu Django Software Foundation i pokazujemy, jak proste jest tworzenie własnych aplikacji internetowych dzięki temu użytecznemu frameworkowi.

FRANK WILES

Latem 2005 roku w świecie open source pojawił się kolejny framework do tworzenia aplikacji internetowych w Pythonie [1]. Od tamtej pory minęły zaledwie trzy lata, a mimo to projekt nabrał ogromnego rozpędu: powstaje właśnie Django Software Foundation, organizacja, która formalnie będzie właścicielem programu. Podobny styl zarządzania ma kilka innych ważnych projektów open source, takich jak Apache, Perl czy Python.

Czym jest Django?

Django to tzw. „framework", zbiorem bibliotek, które umożliwiają programiście tworzenie aplikacji webowych, a zwłaszcza ich unikalnych, interesujących części, bez mierzenia się z nudną infrastrukturą wymaganą przez tego typu projekty. Podobnie jak inne nowoczesne frameworki, takie jak Ruby on Rails, Django korzysta z wzorca MVC (dosłownie: model, widok, kontroler – elementy odpowiedzialne za struktury danych, warstwę prezentacji i sterowanie przepływem programu).

Jedną z dużych zalet Django jest niezwykle elegancki i dobrze zaprojektowany interfejs administracyjny, który możemy automatycznie wygenerować. W artykule pokażemy, jak stworzyć niewielką aplikację Web 2.0 w stylu Twittera, która korzystała będzie między innymi z automatycznie wygenerowanego interfejsu administracyjnego.

Jednak największą zaletą Django jest to, że framework faktycznie sprawdził się w dużych projektach. Korzystają z niego między innymi takie witryny, jak EveryBlock.com, Pownce.com czy Tabblo.com. Jest to również domyślny framework w AppEngine Google; nie dziwi też fakt, że Google stosuje Django w wewnętrznych projektach. Co więcej, Django stanowi podstawowy komponent komercyjnego systemu zarządzania treścią Ellington, z którego korzysta wiele dużych witryn, takich jak na przykład „The Washington Post" [4].

Django Software Foundation

Jacob Kaplan-Moss, Prezes Django Software Foundation i jeden z twórców Django, stwierdził, że fundacja ma przenieść projekt na kolejny etap funkcjonowania: „Oczywiście zależy nam na tym, by przyciągnąć dużą, tętniącą życiem społeczność, dlatego uznaliśmy, że nadszedł czas, by społeczność »stała się właścicielem« Django. Istnienie fundacji w dużej mierze gwarantuje, że projekt przetrwa nawet wtedy, kiedy jego twórcy przestaną się nim interesować".

Kaplan-Moss dodał, że obecnie projekt może przyjmować darowizny, które zostaną przeznaczone na usprawnianie Django. W najbliższej przyszłości fundacja będzie sponsorować rozwój Django głównie poprzez organizację spotkań programistów i użytkowników, jak również innych działań wewnątrz społeczności. Zanim ukaże się wersja 1.0, planowanych jest kilka zlotów programistów – rola Fundacji polega między innymi na tym, by pomóc osobom mającym kluczowe znaczenie dla rozwoju projektu wziąć w nich udział. „Jeśli Fundacja wesprze choć odrobinę szybszy rozwój Django, będę bardzo zadowolony" – mówi Kaplan.

Zaczynamy

W chwili powstawania artykułu wersja 1.0 jeszcze się nie ukazała. Pobierzemy więc najnowszy kod z repozytorium Subversion – różnice w stosunku do ostatecznej wersji nie powinny być znaczne. Wykonujemy to poniższym poleceniem:

```
svn checkout ➥
http://code.djangoproject.com/ ➥
svn/django/trunk/
```

Charakterystyczną cechą Django jest bardzo prosta instalacja. Jeśli jesteśmy podłączeni do Internetu, wystarczy jedynie jako root wydać polecenie:

```
python setup. py install
```

Powyższa komenda zainstaluje Django w katalogu *site-packages* naszej instalacji Pythona.

Choć Django świetnie radzi sobie z obsługą MySQL i PostgreSQL, na potrzeby artykułu użyjemy bazy SQLite. W tym celu potrzebujemy pakietu pysqlite2 [5], który powinniśmy zainstalować według instrukcji zamieszczonych na stronie projektu.

W Django istnieje rozróżnienie między projektami (*projects*) i aplikacjami (*apps*). Jeśli na przykład budujemy dużą witrynę, któ-

ra ma blog, forum i sklep internetowy, to cała witryna w terminologii Django jest projektem, zaś blog, forum i sklep – poszczególnymi aplikacjami składającymi się na jedną całość. W ten sposób główny projekt zostaje podzielony na podprojekty, co ułatwia poruszanie się po całości.

Aby rozpocząć nowy projekt, wydajemy poniższe polecenie:

```
django-admin.py ➥
 startproject mytwit
```

Powyższa komenda utworzy katalog o nazwie *mytwit* z kilkoma plikami konfiguracyjnymi zawierającymi standardowe wartości. Poprosimy teraz Django, by wygenerował szablon przykładowej aplikacji, którą nazwiemy „Twit". W tym celu wydajemy poniższe polecenie w katalogu *mytwit*:

```
python manage. py ➥
startapp Twit
```

Kolej na modyfikację części ustawień, które znajdziemy w pliku *mytwit/settings.py*. Większość opcji w tym pliku powinna wyglądać znajomo. Powinniśmy na przykład zawrzeć tam wpis *DATABASE_ENGINE = 'sqlite3'*, czyli ustawić typ bazy danych na SQLite3. Z kolei zmiennej *DATABASE_NAME* przypisujemy pełną ścieżkę dostępu do *mytwit/twits. db*, czyli pliku SQLite3, który zawierał będzie naszą bazę danych. Zwróćmy uwagę, że w tym przypadku niezbędna jest pełna ścieżka, która zależy od tego, gdzie wcześniej wydaliśmy polecenie *startproject*.

Aby nie musieć później modyfikować *setings. py*, dodamy od razu dwie pozycje do listy zainstalowanych aplikacji, czyli *INSTALLED_APPS*: *django.contrib.admin*, czyli wspomniany wcześniej interfejs administracyjny, oraz *mytwit. Twit*. Możemy je dodać pod koniec listy, zwróćmy jednak uwagę na przecinki na końcu.

Podaliśmy już bazę danych; pora zbudować Model, czyli obiekt Pythona zawierający tabele, kolumny i powiązania między nimi. Ponieważ jednak nasza aplikacja korzysta z tylko jednej tabeli, wystarczy zdefiniować jedna klasę. Plik *mytwit/Twit/models.py* powinien więc wyglądać tak, jak przedstawiliśmy to na Listingu 1.

Przyjrzyjmy się bliżej zawartości pliku *mytwit/Twit/models.py*. Wiersz *from django.db import models* importuje klasy pomocnicze Django związane z obsługą baz danych. Wiersz *class Twit (models.Model):* to początek definicji nowej klasy o nazwie *Twit*, która zawierała będzie kolumnę z datą (*date = models. DateField ('Date')*) i kolumnę tekstową z właściwą zawartością rekordu (*entry = models. CharField (max_length='500')*); w tym przypadku *max_length='500'* oznacza, że pole tekstowe może mieć maksymalną długość pięciuset znaków). Następnie definiujemy specjalną metodę *__str__*, która określa sposób wyświetlania instancji obiektu w formie zmiennej łańcuchowej (w tym przypadku będzie to po prostu wyświetlenie obu łańcuchów, bez żadnej modyfikacji – *return '%s %s' % (self. date, self.entry)*). Z metody tej korzysta interfejs administracyjny, kiedy wyświetla listing rekordów z tabeli z bazy danych. Pusta klasa *Admin* informuje Django, że ma wygenerować nam interfejs administracyjny. W każdej chwili możemy sprawdzić status naszego Modelu, wydając poniższe polecenie:

```
python manage. py ➥
validate
```

Jeśli wszystko poszło dobrze, polecenie to powinno zwrócić komunikat *0 errors found*. Jeśli Model nie zgłasza błędów, możemy przejść do tworzenia tabeli – Django zrobi to dla nas, budując strukturę bazy danych na podstawie informacji zawartych w pliku *models.py*. Wystarczy wydać polecenie:

```
python manage. py ➥
syncdb
```

Po chwili powinniśmy ujrzeć na ekranie wiersze zawierające ciąg *Creating table*; niektóre z nich dotyczą tabel odpowiedzialnych za prawa dostępu grup i użytkowników, niektóre – administratora, zaś ostatni tworzy tabelę *Twit*. W tym momencie Django przygotowuje również konto administratora i prosi nas o podanie loginu i hasła – wybierzmy dowolną kombinację i dobrze ją zapamiętajmy, ponieważ będziemy z niej za chwilę korzystać.

Po utworzeniu Modelu i tabeli baz danych możemy włączyć interfejs administracyjny. W tym celu wystarczy w pliku *mytwit/urls.py* odkomentować wiersz:

```
(r'^admin/', include ➥
('django. contrib. admin. urls')),
```

Łatwo rozpoznamy tu wyrażenie regularne, które przemapuje wszystkie wywołania do adresu URL rozpoczynające się od ciągu *admin/* (po nazwie witryny), wywołując kod obsługujący interfejs administracyjny. Wypróbujmy go – wystarczy uruchomić wbudowany serwer Django:

```
python manage. py ➥
runserver
```

Domyślnie serwer uruchomiony zostanie na interfejsie lokalnym, na porcie 8000. Jeśli chcemy użyć konkretnego adresu IP bądź innego portu, wystarczy dodać odpowiedni argument do poprzedniego polecenia, na przykład:

```
python manage. py ➥
runserver 192.168.0.55: 5555
```

Przyjmijmy jednak, że używamy wartości domyślnych. Po wpisaniu w przeglądarce adresu *http://127.0.0.1:8000/admin* powinniśmy ujrzeć ekran logowania administratora naszej aplikacji. Gdy się zalogujemy (użyjemy wprowadzonej wcześniej nazwy użytkownika i hasła), zobaczymy ekran podobny do tego, który został przedstawiony na Rysunku 1.

Ponieważ budujemy aplikację osobistą, możemy na razie zignorować sekcje *Sites*

Listing 1: mytwit/Twit/models. py

```
from django. db import models

class Twit (models. Model):
 date = models. DateField ('Date')
 entry = models. CharField
(max length='500')

 def   str  (self):
 return '%s %s' % (self. date, self. entry)

 class Admin:
 pass
```

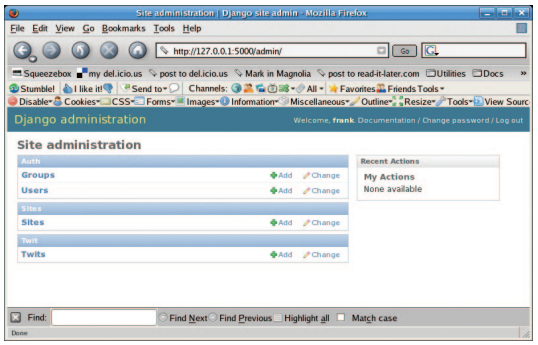
Listing 2: Rekordy w odwrotnej kolejności

```
from django. shortcuts import render_to_response
from models import Twit

def alltwits (request):
 all_entries = Twit. objects. all (). order_by ("date"). reverse ()
 return render_to_response ('all_twits. html', { 'entries': all_entries })
```

Listing 3: Przykładowy szablon służący do prezentacji danych

```
<html>
<body>
<table>
<tr>
<th>Data</th>
<th>Rekord</th>
</tr>
{% for t in entries %}
<tr>
<td>{{ t. date }}</td>
<td>{{ t. entry}}</td>
</tr>
{% endfor %}
</table>
</body>
</html>
```



Rysunek 1: Ekran administracyjny Django.

i *Admin* i po prostu kliknąć na ikonce *Add* w okienku Twit.

Możemy teraz wypełnić określony rekord danymi. Jeśli klikniemy na *Today*, Django automatycznie wprowadzi dzisiejszą datę; możemy jednak użyć poręcznej kontrolki i wybrać dowolną inną. Następnie wprowadzamy tekst rekordu i klikamy na *Save*. Zostaniemy automatycznie przeniesieni na stronę, która wylistuje wszystkie wpisy w tabeli. Jeśli klikniemy na jeden z istniejących rekordów, przejdziemy na stronę, na której będziemy mogli go edytować lub usunąć.

Sama zabawa z interfejsem administracyjnym nie wystarczy; musimy móc zaprezentować jej wyniki. W tym celu będziemy potrzebowali Widoku i szablonu. W Django Widokiem jest moduł, który zajmuje się logiką przetwarzania danych, zaś szablon związany jest z bezpośrednim przedstawieniem tych danych użytkownikowi. Jeśli przyzwyczailiśmy się do frameworków opartych na MVC, gdzie Widokiem jest sam szablon, nazewnictwo używane w Django może być dla nas nieco mylące.

Zaczniemy od dodania do *mytwit/Twit/views.py* prostej metody, która zwróci wszystkie rekordy w kolejności chronologicznej. Plik ten powinien wyglądać tak, jak na Listingu 2.

Definiujemy tam metodę *alltwits*, która najpierw pobiera wszystkie obiekty *Twit* (*Twit. objects. all ()*), porządkuje je według daty (*order_by ("date")*) i odwraca kolejność (*reverse ()*). Następnie wywołuje funkcję *render_to_response ()*, podając jako argument szablon związany z danym widokiem i zmienną słownikową zawierającą dane, które chcemy przekazać do szablonu.

Pora teraz na utworzenie samego szablonu. Oczywiście w praktyce może to być bardzo rozbudowany plik HTML, jednak dla celów poglądowych użyjemy bardzo prostego przykładu widocznego na Listingu 3.

Oczywiście dobrze jest trzymać szablony w oddzielnym katalogu, zapiszmy więc plik z Listingu 3 jako *mytwit/templates/all_twits. html*.

Jak widzimy, język szablonów Django ma kilka użytecznych cech i jest prosty w użyciu. W naszym przykładzie skorzystaliśmy z prostej pętli *for*, przechodząc przez kolejne obiekty Twit wywołane przez metodę *alltwits ()* i wyświetlając dane metodami *date* i *entry*.

Pozostaje jedynie odpowiednio skonfigurować Django, podając pełną ścieżkę dostępu do szablonu w systemie plików i przypisując danemu widokowi odpowiedni URL. Wcześniej jednak musimy ustawić zmienną *TEMPLATE_DIRS*, która będzie wskazywać katalog z szablonami (czy też kilka katalogów, zmienna ta jest bowiem listą). Również w tym przypadku konieczne jest podanie pełnej ścieżki dostępu.

Pozostaje nam jedynie zmodyfikować plik *mytwit/urls. py*, ustawiając odpowiednie mapowania URL-i. Zrobimy to tak, jak to zostało przedstawione na Listingu 4: zaimportujemy widoki odpowiednie dla danej aplikacji, dodamy URL z */twits/* i pozostawimy domyślne mapowanie interfejsu administracyjnego.

Jeśli teraz wpiszemy w przeglądarce adres *http://127.0.0.1:8000/twits*, powinniśmy ujrzeć wszystkie wpisy wprowadzone z poziomu interfejsu administracyjnego).

Wnioski

Korzystanie z wbudowanego serwera WWW i bazy danych SQLite sprawdza się jedynie w sytuacji takich jak ta – testowanie aplikacji na szybko. Jeśli mamy w planach stworzenie prawdziwej aplikacji, musimy odpowiednio zmienić konfigurację Django, by nasz projekt korzystał z Apache, mod_python i bardziej wydajnej bazy danych, takiej jak PostgreSQL. Więcej informacji na ten temat znajdziemy na witrynie projektu.

Listing 4: Mapowanie URL-i

```
from django. conf. urls. defaults import *
from mytwit. Twit import views

urlpatterns = patterns ('',
 (r'^twits/', 'mytwit. Twit. views. all-
twits'),
 (r'^admin/', include ('django. contrib.
admin. urls')),
)
```

Choć niniejszy artykuł nie pretenduje do miana przewodnika po możliwościach Django, mam nadzieję, że pobudził ciekawość. Duże znaczenie ma fakt, że strona domowa Django zawiera bardzo bogatą dokumentację, która powinna pomóc nie tylko na początku. Trudno nie wspomnieć o znakomitej książce o Django, *The Django Book*, którą można kupić lub przeczytać online [6].

Choć jestem wiernym użytkownikiem Perla, to jednak muszę przyznać, że Django zaskoczył mnie bardzo pozytywnie, zarówno pod względem łatwości użytkowania, jak i dopracowania szczegółów. Korzystając z okazji, chciałbym podziękować Jacobowi Kaplan-Mossowi i Adrianowi Holovaty'emu za ich wkład w ten projekt, jak również pomoc przy pisaniu niniejszego artykułu. ■

INFO

[1] Stronoa domowa projektu Django: *http://www.djangoproject.com*

[2] Django Software Foundation: *http://www.djangoproject.com/foundation*

[3] Witryny korzystające z Django: *http://www.djangosites.org*

[4] Kod Django: *djangoproject.com/download/*

[5] Strona domowa projektu Pysqlite2: *http://initd.org/pub/software/pysqlite/*

[6] *The Django Book*: *http://www.djangobook.com*

AUTOR

Frank Wiles jest właścicielem Revolution Systems *(http://www.revsys.com)*, firmy konsultingowej zajmującej się rozwijaniem aplikacji webowych i specjalizującej się w optymalizacji aplikacji open source.