Django. Ćwiczenia praktyczne

Autor: Piotr Maliński
ISBN: 978-83-246-1888-0
Format: A5, stron: 88

Wykorzystaj możliwości Django w swoim projekcie!

- Jak zainstalować Django?
- Jak skonfigurować framework i dopasować go do swoich potrzeb?
- Jak uruchomić Django na Google App Engine?

Django nie jest kolejnym frameworkiem napisanym w języku PHP. To elitarne rozwiązanie, wykorzystujące język Python, oparte na wzorcu projektowym MVC. Pierwotnie Django zostało opracowane z myślą o stronach „prasowych", z dużą ilością newsów. Pierwsza publiczna wersja ujrzała światło dzienne w 2005 roku. Od tego czasu Django odnotowuje ciągły wzrost popularności, a wachlarz jego zastosowań znacznie się rozrósł!

Dzięki tej książce również Ty będziesz mógł wykorzystać ten framework w swoim projekcie! Sprawdzona formuła książki, kładąca nacisk na ćwiczenia, pozwoli Ci błyskawicznie opanować sposób instalacji oraz stworzyć i skonfigurować Twój własny projekt. Dowiesz się także, jak obsłużyć błędy oraz kanały RSS czy też wygenerować pliki PDF. Ponadto nauczysz się konfigurować serwery Apache 2 oraz Nginx. Ćwiczenia zawarte w tej książce pozwolą Ci szybko i przyjemnie wdrożyć Django w Twoim projekcie aplikacji WWW!

- Sposób instalacji Django
- Tworzenie nowego projektu
- Konfiguracja projektu
- Wykorzystanie panelu administracyjnego
- Mapowanie widoków
- Obsługa formularzy
- Reagowanie na błędy
- Udostępnienie kanału RSS
- Generowanie plików PDF
- Zasady konfiguracji serwera Apache 2 z mod_python
- Konfiguracja serwera Nginx
- Przegląd firm oferujących hosting z obsługą Pythona
- Django i Google App Engine

Stwórz wydajną i przyjemną w zarządzaniu witrynę WWW!

Helion SA
ul. Kościuszki 1c
44-100 Gliwice
tel. 032 230 98 63
e-mail: helion@helion.pl

Spis treści

Księga kucharska

W rozdziale tym przedstawione zostaną różne rozwiązania, fragmenty kodu i ciekawe aplikacje Django.

ĆWICZENIE

5.1 Obsługa błędów

Ustawienie w *settings.py* zmiennej `DEBUG` na `False` spowoduje, że zamiast treści wyjątków wyświetlany będzie szablon *500.html*, a w przypadku braku podstrony — *404.html*. Zazwyczaj jednak zachodzi potrzeba obsługi części błędów na poziomie widoków, np. gdy brakuje kategorii o podanym odnośniku i chcemy wyświetlić stosowny komunikat. W przypadku Django, czy też w ogóle Pythona, jest to dość łatwe do zaimplementowania przy użyciu wyjątków. Będą one generowane przy pobieraniu rekordu za pomocą metody `GET`, jeżeli dany obiekt nie będzie istniał.

1. Otwórz plik *news/views.py* i w widoku `show_news` zmień wiersz pobierania wiadomości na:

```
                try:
    news = News.objects.get(slug=slug)
except:
    return render_to_response('error.html',
        {'error': 'Strona nie istnieje'},
        context_instance=RequestContext(request))
```

2. Stwórz szablon *error.html* o kodzie:

```
{% extends "base.html" %}

{% block main %}
{{ error }}
{% endblock %}
```

3. Otwórz stronę z dowolną wiadomością, a następnie zmień jej slug na błędny.

W przypadku gdy wiadomość o podanym odnośniku (slug) nie będzie istniała, wygenerowany zostanie wyjątek. Przechwytujemy go za pomocą `try/catch` i wyświetlamy odpowiedni komunikat.

ĆWICZENIE

5.2 Kanały RSS

W tym ćwiczeniu dodamy kanał RSS dla najnowszych wiadomości z wykorzystaniem komponentu *Feed* frameworka Django.

1. W katalogu *news* stwórz plik *feeds.py* o kodzie:

```
# -*- coding: utf-8 -*-
from blogs.news.models import *
from django.contrib.syndication.feeds import Feed

class LatestNews(Feed):
        title = 'Wiadomości z MOJASTRONA.pl'
        link = 'MOJASTRONA.pl'
        description = 'Wiadomości z MOJASTRONA.pl'
        def items(self):
                return News.objects.order_by('-id')[:15]
```

2. W *urls.py* dodaj fragment kodu:

```
from blogs.news.feeds import *
feeds = {
    'news': LatestNews,
}
```

3. Dodaj regułę mapowania:

```
(r'^rss/(?P<url>.*)/$', 'django.contrib.syndication.views.feed',
↪{'feed_dict': feeds}),
```

4. W katalogu *templates* utwórz folder *feeds*, a w nim pliki *news_title.html* i *news_desription*, odpowiednio o kodzie:

```
{{ obj.title }}
```

i

```
{{ obj.text|safe }}
```

5. Pod adresem *http://localhost:8000/rss/news/* dostępny będzie gotowy kanał RSS.

Tworzenie kanałów RSS czy też Atom za pomocą *Feed* polega na zdefiniowaniu klasy zawierającej opis kanału (`title`, `link`, `description`), a także podklasy `items` określającej listę rekordów do wyświetlenia. Dodatkowo możemy dodać proste szablony, określające, jakie pole lub pola modelu należy wyświetlać dla pól `title` i `description` w kanale RSS.

ĆWICZENIE

5.3 Mapa sitemap

Sitemap to plik XML zawierający listę stron serwisu. Sitemaps zostały wymyślone przez Google i obecnie kilka wyszukiwarek używa ich w celu usprawnienia indeksowania stron internetowych. Celem tego ćwiczenia będzie stworzenie mapy z wykorzystaniem elementów frameworka.

1. W *settings.py* do `INSTALLED_APPS` dodaj `'django.contrib.sitemaps'`.
2. Otwórz plik *news/feeds.py* i dodaj fragment kodu:

```
from django.contrib.sitemaps import Sitemap
class NewsMap(Sitemap):
        def items( self ):
                return News.objects.all()
        def lastmod( self, obj ):
                return obj.date
        def changefreq(self, obj):
                return 'monthly'
```

3. Otwórz plik *urls.py* i dopisz do niego kod:

```
sitemaps = {
        'news': NewsMap,
        }
```

4. Dodaj do reguł mapowania wpis:

```
(r'^sitemap.xml$', 'django.contrib.sitemaps.views.sitemap',
↪{'sitemaps': sitemaps})
```

5. Pod adresem *http://localhost:8000/sitemap.xml* znajdziesz mapę sitemap dla naszej strony.

ĆWICZENIE

5.4 Generowanie plików PDF

W Pythonie do generowania plików PDF możemy wykorzystać bibliotekę *ReportLab* (*http://www.reportlab.org/rl_toolkit.html*). Pakiet dla systemów MS Windows możemy pobrać ze strony projektu, natomiast użytkownicy Linuksa zapewne znajdą *ReportLab* w repozytorium dystrybucji. Tworzenie plików PDF polega tu na rysowaniu elementów na stronie z wykorzystaniem współrzędnych. *ReportLab* to narzędzie bardziej techniczne, służące, jak sama nazwa wskazuje, do generowania raportów, zestawień czy dokumentów o określonej strukturze. Jeżeli interesuje nas konwersja kodu HTML na plik PDF w większym zakresie, to pomóc nam może biblioteka *Pisa*. W tym ćwiczeniu przedstawiony zostanie prosty widok generujący plik PDF dla wiadomości.

1. Otwórz plik *news/views.py* i dodaj na początku importy:

```
from reportlab.platypus import Table, TableStyle, Paragraph
from reportlab.lib import colors
from reportlab.pdfgen import canvas
from reportlab.lib.pagesizes import A4
from reportlab.pdfbase import pdfmetrics
from reportlab.pdfbase.ttfonts import TTFont
from reportlab.lib.styles import getSampleStyleSheet
from django.http import HttpResponse, HttpResponseRedirect
from django.conf import settings
```

2. Dodaj widok generujący plik PDF o kodzie:

```
def show_news_pdf(request, slug):
   try:
      news = News.objects.get(slug=slug)
   except:
      return render_to_response('error.html',
         {'error': 'Strona nie istnieje'},
         context_instance=RequestContext(request))
   e = HttpResponse()
   e['Content-Type'] = 'application/pdf'
   e['Content-Disposition'] = 'attachment; filename="%s.pdf";'
   ↪% news.slug

   c = canvas.Canvas(e, pagesize=A4)

   width, height = A4
   pdfmetrics.registerFont(TTFont('Dejavu', settings.APP_ROOT +
   ↪'site_media/fonts/DejaVuSans.ttf'))
   c.setFont("Dejavu", 14)
              # margines, wysokość, treść
```

```
c.drawString(50,(height-60), news.title)

stylesheet=getSampleStyleSheet()
styleN = stylesheet['Normal']
styleN.fontName = 'Dejavu'
styleN.fontSize = 9

p = Paragraph(news.text, styleN)
w,h = p.wrap(width-70, height)
p.drawOn(c, 50, (height-120))

c.line(50, (70), 570, (70))
c.setFont("Dejavu", 8)
c.drawString(50,(60), unicode(news.date))
c.drawString(50,(50), u'Wiadomość ze strony
➥http://www.MOJASTRONA.pl')

c.showPage()
c.save()
return e
```

3. Dodaj w pliku *urls.py* regułę mapującą odnośnik na następujący widok:

```
(r'^news/(?P<slug>[\w\-_]+)/pdf/$', 'news.views.show_news_pdf'),
```

4. W szablonie *show_news.html* dodaj odnośnik do wersji PDF:

```
<a href="/news/{{ news.slug }}/pdf/">PDF</a>
```

5. W pliku *settings.py* zdefiniuj `APP_ROOT`, podając bezwzględną ścieżkę do katalogu z projektem:

```
APP_ROOT = 'ścieżka/do/katalogu/z/projektem'
```

6. W katalogu *site_media* dodaj folder *fonts* i umieść w nim przynajmniej jeden plik czcionki TTF (w tym przykładzie skorzystano z czcionki Dejavu — *dejavusans.ttf*).
7. Zapisz wszystkie zmiany i sprawdź wersję PDF dla wybranej wiadomości.

Jeśli chodzi o kod, zaczynamy w nim od określenia płótna (`canvas`), czyli rozmiaru strony i nazwy pliku. Następnie rejestrujemy czcionki, jakich chcemy użyć (podane pliki **.ttf* muszą być dostępne). W *ReportLab* elementy umieszczane (rysowane) są w miejscu (x, y) określonym w pikselach lub w innych jednostkach (cale, centymetry itd.). Dla strony A4 zmienne `width` i `height` przechowują dane punktu znajdującego się w lewym dolnym rogu (rozmiar strony A4). Do wyświetlania tekstu stosujemy nieformatowany (`drawString`) oraz formatowany (`Paragraph`)

Rysunek 5.1. *Plik PDF wygenerowany dla wiadomości*

tekst. Wszystkie te elementy posiadają szereg opcji odpowiedzialnych za ich wygląd, a szczegóły znajdziemy w bardzo obszernej dokumentacji *ReportLab*.

ĆWICZENIE

5.5 AJAX w Django

Wykorzystanie żądań wysyłanych AJAX-em wygląda w Django standardowo. Przesyłamy je na określony URL, a następnie do ich wykonywania używamy dowolnego frameworka JavaScript, np. jQuery, czy też jakichś prostszych skryptów. W tym ćwiczeniu skorzystamy z prostej biblioteki *ajaxroutine* (*http://www.dynamicdrive.com/dynamicindex17/ajaxroutine.htm*).

1. Otwórz plik *news/views.py* i dodaj widok:

```
def ajax(request):
   return HttpResponse(unicode(datetime.now()))
```

2. Dodaj mapowanie w *urls.py*:

```
(r'^ajax/$', 'news.views.ajax'),
```

3. W szablonie *base.html* w sekcji `HEAD` wpisz kod:

```
<script type="text/javascript"
↪src="/site_media/ajaxroutine.js"></script>
<script type="text/javascript">
function processGetPost()
        {
        var myajax=ajaxpack.ajaxobj
        var myfiletype=ajaxpack.filetype
```

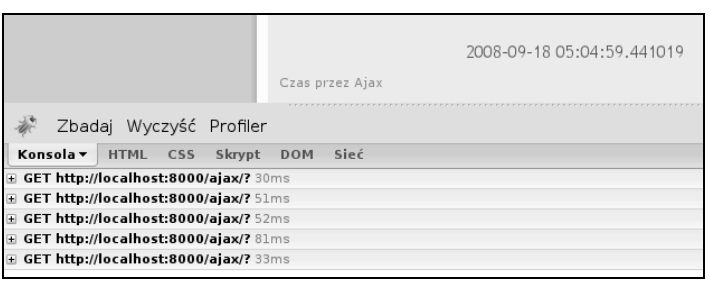
```
        if (myajax.readyState == 4)
                {
                if (myajax.status==200 ||
                ↪window.location.href.indexOf("http")==-1)
                        {
                        // Jeżeli coś zostanie zwrócone, wyświetl to
                        if (myajax.responseText.length > 3)
                                {
                                document.getElementById('aj')
                                ↪.innerHTML='<h1>' +
                                ↪myajax.responseText + '</h1>';
                                }
                        else
                                {
                                // Czyścimy komunikat
                                document.getElementById('aj')
                                ↪.innerHTML='';
                                }
                        }
                }
        }
</script>
```

4. W kodzie szablonu dopisz:

```
<div id="aj" style="text-align:center;"></div>
        <a href="#" onclick="ajaxpack.getAjaxRequest('/ajax/', '',
        ↪processGetPost, 'txt')">Czas przez AJAX</a>
```

Po zapisaniu zmian pojawi się odnośnik *Czas przez AJAX*, który będzie wysyłał żądania AJAX-a za pomocą biblioteki *ajaxroutine*. Dodana funkcja `processGetPost` będzie wstawiała otrzymane dane (to co zwróci wywołany widok) do taga `DIV` o ID `aj`:

```
document.getElementById('aj').innerHTML
```



Rysunek 5.2. *Żądania wysyłane za pomocą AJAX-a*