

Wspomagacze Django

Jan Filipowski

25 maja 2010

TDD – przypomnienie

- ▶ testy są fajne
- ▶ testujemy co tylko się da
- ▶ cykl: test – code – refactor
- ▶ dwa frameworki - unittest i doctest

Asercje

- ▶ `assertEqual`
- ▶ `assertRaises`
- ▶ `assertTrue`
- ▶ `assertAlmostEqual`
- ▶ itd.

Co testować w Django?

- ▶ modele
- ▶ widoki

Testowanie modeli

```
class PostStrTest(TestCase):  
    def setUp(self):  
        self.post = Post()  
  
    def testWithTitle(self):  
        self.post.title = "kaka"  
        self.assertEqual(str(self.post), "Post:  
        kaka")  
  
    def testWithoutTitle(self):  
        self.assertEqual(str(self.post), "Post:  
        untitled")
```

Odpalamy maszynerię

```
$ python manage.py test blog
```

i oczywiście zaliczamy faila.

Po implementacji `__str__` asercje przechodzą.

Testowanie widoków

```
class ViewIndexTest(TestCase):
    def setUp(self):
        self.client = Client() # + post1, post2

    def testTemplate(self):
        response = self.client.get('/blog/')
        self.assertTemplateUsed(response, 'blog/
            index.html')

    def testPostList(self):
        response = self.client.get('/blog/')
        self.assertEqual(len(response.context['
            posts']), 2)
```

Test Client

- ▶ udaje przeglądarkę
- ▶ ma dostęp do kontekstu
- ▶ ma dostęp do templaty
- ▶ ma dostęp do nagłówków HTTP (zarówno request jak i response)
- ▶ umie się zalogować (`django.contrib.auth`)

Django asercje

- ▶ `assertContains`
- ▶ `assertFormError`
- ▶ `assertTemplateUsed`
- ▶ `assertRedirects`

Testy integracyjne?

- ▶ nieizolowane testy widoków (gdy strona bez JS)
- ▶ Selenium

Warto poznać

- ▶ Sane – biblioteka dla testów jednostkowych i integracyjnych
- ▶ banana – DSL do pisania testów

Migracje South

Problem

Polecenie syncdb działa tylko raz – po utworzeniu modelu. My chcemy być elastyczni i zmieniać schemat bazy danych częściej niż co model.

Rozwiązanie

Migracje South

Standardowe użycie

- ▶ tworzymy aplikację, jej modele i dorzucamy south do appek
- ▶ tworzymy migrację inicjującą:

```
$ python manage.py schemamigration appname  
    --init
```

- ▶ dodajemy pola do modeli
- ▶ tworzymy migrację automatycznie:

```
$ python manage.py schemamigration appname  
    --auto
```

- ▶ migrujemy

```
$ python manage.py migrate
```

Ficzery

- ▶ automatycznie generowane migracje
- ▶ niezależne od RDBMS
- ▶ niezależne na poziomie aplikacji
- ▶ odporne na VCS

Thumbnailowanie Sorl

Problem

Uploadujemy obrazki i chcemy mieć ich wersje w różnych rozmiarach.

Rozwiązanie

sorl-thumbnail

Przykład użycia

1. dodajemy sorl do appek
2. pole 'image' to uploadowany obrazek w modelu
3. dodajemy do templaty:

```
{% load thumbnail %}
```

4. by wyświetlić obrazek:

```
{% thumbnail obj.image 250x250 autocrop %}
```

Ficzery

- ▶ cachowanie
- ▶ wbudowane różne filtry
- ▶ można pisać własne filtry
- ▶ chyży

Problem

Potrzebujemy wyszukiwarki.

Rozwiązanie

Haystack + Whoosh, Solr albo Xapian. My wybieramy Whoosh.

Przykład użycia cz. 1

- ▶ dodajemy haystack do appek
- ▶ ustawiamy HAYSTACK_SITECONF oraz HAYSTACK_SEARCH_ENGINE
- ▶ ustawiamy HAYSTACK_WHOOSH_PATH
- ▶ definiujemy search_indexes.py

```
import haystack  
haystack.autodiscover()
```

Przykład użycia cz. 2

Definiujemy `search_indexes.py` dla aplikacji

```
class PostIndex(SearchIndex):  
    content = CharField(document=True)  
    author = CharField(model_attr='user')  
  
    def get_queryset(self):  
        return Post.objects.all()
```

I odpowiedni template do wyszukiwarki w `search/search.html`

Ficzery

- ▶ wsparcie dla Solr, Xapian i Whoosh
- ▶ możliwość podświetlania znalezionych fraz
- ▶ "podobne"

Chunks

Idea

Edytowalne małe fragmenty strony, zarządzane z poziomu admina.

Użycie

W templatce:

```
{% load chunks %}  
{% chunk "dada" %}
```

Registration

Idea

Rejestracja i autoryzacja użytkowników na stronie (bez użycia django admin).

Użycie

- ▶ dodaj registration do apppek
- ▶ syncdb
- ▶ dodaj do urls.py

```
(r '^accounts/', include('registration.  
backends.default.urls'))),
```

- ▶ zdefiniuj templaty

Tagging

Idea

Tagowanie bytów na stronie.

Użycie cz. 1

- ▶ dodaj tagging do appek
- ▶ syncdb
- ▶ zarejestruj modele w tagging

```
tagging.register(Post)
```

Tagging

Użycie cz. 2

- ▶ przypisz tagi

```
>>> post.tags = "kaka, dudud, lala"  
>>> post.tags  
[<Tag: kaka>, <Tag: dudud>, <Tag: lala>]
```

- ▶ wypisz obiekty modelu z tagiem

```
>>> TaggedItem.objects.get_by_model(Post, 'kaka')
```

- ▶ można wypisać powiązane (tagami) obiekty

Problem

Ręczny deploy jest strasznie nudny i czasochłonny.

Rozwiązanie

Fabric albo Capistrano (albo masa innych).

Fabric – ficzery

- ▶ działanie na wielu hostach
- ▶ automatyczne wywoływanie komend shellowych
- ▶ dostęp do sudo

Fabric – podstawy

- ▶ `run()`
- ▶ `local()`
- ▶ `sudo()`
- ▶ `put()`
- ▶ `get()`
- ▶ `prompt()`

- ▶ south.aeracode.org
- ▶ code.google.com/p/sorl-thumbnail
- ▶ haystacksearch.org
- ▶ github.com/clintecker/django-chunks
- ▶ www.bitbucket.org/ubernostrum/django-registration
- ▶ code.google.com/p/django-tagging
- ▶ docs.fabfile.org

TDD w Django
South
Sori
Haystack + Whoosh
Mafe, a cieszy
Deployment
Koniec

Koniec

Dziękuję za uwagę!