Zrąb webowy dla perfekcjonistów z terminami

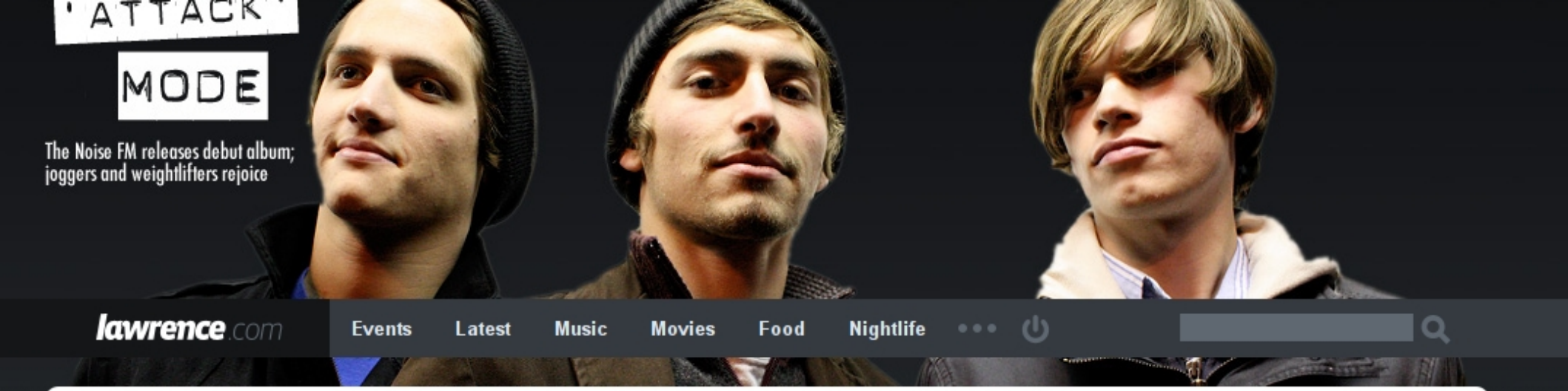
autor: Kamil Adamczyk

Django

- **napisany w Pythonie**
- **DRY** czyli zasada **„nie powtarzaj się”** w odniesieniu do tworzenia aplikacji
- **opensource'owy** zrąb służący do tworzenia aplikacji internetowych (licencja BSD)
- opiera się na **wzorcu projektowym MVC**

Historia

- Prace nad Django rozpoczęły się jesienią 2003 roku
- Trzy główne strony gazety World Online
  - LJWorld.com (wiadomości),
  - Lawrence.com (rozrywka/muzyka)
  - KUsports (sport),
  - wiele nagród w dziedzinie internetowej publicystyki
- Dwa lata później, latem 2005 roku otwarcie kodu zrębu



Best bets | Upcoming

Punch Bros. featuring Chris Thile

The former Nickel Creek frontman and mandolin virtuoso finds himself with new bandmates who are among the most in-demand performers in the worlds of bluegrass, folk, and traditional music

today at 7:00pm
The Granada, $18

Also today

Film screening: "Testimony: The Maria Guardado Story" @Spencer Museum of Art

Tom Page and Dustin Arbuckle / RedLefty @Dempsey's Pub

Smackdown! @The Bottleneck

All of today's events

Search events

November events

S	M	T	W	T	F	S
26	27	28	29	30	31	1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29

Staph blogs

Scene Stealers
The reality of 'Rachel Getting Married'
0 comments

Safe in the Fire Swamp
Aaaaaargh!

Django

- **powstało z aplikacji** (Ellington CMS)
- bardzo **duża skalowalność** i **wydajność** pod obciążeniem
- przejrzystość kodu
- przyjazne adresy dokumentów z możliwością dowolnego ich kształtowania

Django

- własny, prosty serwer do testowania aplikacji
- interaktywna konsola
- wsparcie dla współdzielenia treści między stronami
- współpracuje z Apache poprzez mod_python oraz z innymi serwerami poprzez protokoły FastCGI i SCGI
- świetny ORM

Django

- zrąb RSS
- zrąb komentarzy
- zrąb cache'owania
- automatyczny admin (panel edytorski)
- internacjonalizacja
- system szablonów

Jak rozpocząć pracę?

- $ django-admin startproject mysite
- $ cd mysite/
- $ ls
- __init__.py manage.py settings.py urls.py

Te pliki to:

- **__init__.py**: plik informujący Pythona o tym, że katalog nadrzędny powinien być traktowany jako pakiet Pythona.
- **manage.py**: Działające z linii poleceń narzędzie, które pozwala na interakcję z projektem Django.
- **settings.py**: Ustawienia/konfiguracja dla tego projektu Django.
- **urls.py**: Deklaracja adresów URL dla tego projektu Django; “mapa serwisu”.

Jak rozpocząc pracę? cd..

- $ python ./manage.py startapp myapp
- $ cd myapp
- $ ls myapp/
- __init__.py models.py views.py

- *(... implementacja ...)*

- $ python ./manage.py syncdb
- $ python ./manage.py runserver 8080

Model

- idealnie odwzorowuje strukturę bazy danych
- całkowita abstrakcja od SQL
- informacja o modelu tylko w modelu
- brak magii

Model - simple

```python
from django.db import models

class Person(models.Model):
    first_name = models.CharField(max_length=30)
    last_name = models.CharField(max_length=30)
```

```
CREATE TABLE myapp_person (
    "id" serial NOT NULL PRIMARY KEY,
    "first_name" varchar(30) NOT NULL,
    "last_name" varchar(30) NOT NULL
);
```

Model

```
class Person(models.Model):
    name = models.CharField(max_length=128)

    def __unicode__(self):
        return self.name

class Group(models.Model):
    name = models.CharField(max_length=128)
    members = models.ManyToManyField(Person, through='Membership')

    def __unicode__(self):
        return self.name

class Membership(models.Model):
    person = models.ForeignKey(Person)
    group = models.ForeignKey(Group)
    date_joined = models.DateField()
    invite_reason = models.CharField(max_length=64)
```

Model

```
>>> ringo = Person.objects.create(name="Ringo Starr")
>>> paul = Person.objects.create(name="Paul McCartney")
>>> beatles = Group.objects.create(name="The Beatles")
>>> m1 = Membership(person=ringo, group=beatles,
...     date_joined=date(1962, 8, 16),
...     invite_reason= "Needed a new drummer.")
>>> m1.save()
>>> beatles.members.all()
[<Person: Ringo Starr>]
>>> ringo.group_set.all()
[<Group: The Beatles>]
>>> m2 = Membership.objects.create(person=paul, group=beatles,
...     date_joined=date(1960, 8, 1),
...     invite_reason= "Wanted to form a band.")
>>> beatles.members.all()
[<Person: Ringo Starr>, <Person: Paul McCartney>]
```

Model - SQL

```
$ python ./manage.py sqlall myapp
```

```
BEGIN;
CREATE TABLE "myapp_person" (
    "id" integer NOT NULL PRIMARY KEY,
    "name" varchar(128) NOT NULL
)
;
CREATE TABLE "myapp_group" (
    "id" integer NOT NULL PRIMARY KEY,
    "name" varchar(128) NOT NULL
)
;
CREATE TABLE "myapp_membership" (
    "id" integer NOT NULL PRIMARY KEY,
    "person_id" integer NOT NULL REFERENCES "myapp_person" ("id"),
    "group_id" integer NOT NULL REFERENCES "myapp_group" ("id"),
    "date_joined" date NOT NULL,
    "invite_reason" varchar(64) NOT NULL
)
;
CREATE INDEX "myapp_membership_person_id" ON "myapp_membership" ("person_id");
CREATE INDEX "myapp_membership_group_id" ON "myapp_membership" ("group_id");
COMMIT;
```

Model - QuerySet

Leniwe wylicznie, poniższy kod to tylko 1 zapytanie do bazy w ostatniej linijce

```
>>> q = Entry.objects.filter(headline__startswith="What")
>>> q = q.filter(pub_date__lte=datetime.now())
>>> q = q.exclude(body_text__icontains="food")
>>> print q
```

SELECT foo FROM Entry LIMIT 5

```
>>> Entry.objects.all()[:5]
```

SELECT foo FROM Entry OFFSET 5 LIMIT 5

```
>>> Entry.objects.all()[5:10]
```

SELECT foo FROM Entry ORDER BY headline LIMIT 1

```
>>> Entry.objects.order_by('headline')[0]
```

Szablony

- kierowane do nieprogramistów

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

{% block title %}My amazing blog{% endblock %}

{% block content %}
{% for entry in blog_entries %}
    <h2>{{ entry.title }}</h2>
    <p>{{ entry.body }}</p>
{% endfor %}
{% endblock %}
```

Szablony

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
    <link rel="stylesheet" href="style.css" />
    <title>{% block title %}My amazing site{% endblock %}</title>
</head>

<body>
    <div id="sidebar">
        {% block sidebar %}
        <ul>
            <li><a href="/">Home</a></li>
            <li><a href="/blog/">Blog</a></li>
        </ul>
        {% endblock %}
    </div>

    <div id="content">
        {% block content %}{% endblock %}
    </div>
</body>
</html>
```

Widok

```python
from django.http import HttpResponse, Http404
from django.shortcuts import render_to_response
from mysite.myapp.models import Person


def person(request, p_id):
    try:
        p = Person.objects.get(id = p_id)
    except Person.DoesNotExist:
        raise Http404
    return render_to_response('person/detail.html', {'person': p})

```

***detail.html*:**

{{ person.id }}
{{ person.first_name }}
{{ person.last_name }}

```python
urlpatterns = patterns('',
    (r'^person/(?P<p_id>\d+)/$', 'myapp.views.person'),
)
```

URL dispatcher

```
urlpatterns = patterns('',
    (r'^articles/2003/$', 'news.views.special_case_2003'),
    (r'^articles/(?P<year>\d{4})/$', 'news.views.year_archive'),
    (r'^articles/(?P<year>\d{4})/(?P<month>\d{2})/$', 'news.views.month_archive'),
    (r'^articles/(?P<year>\d{4})/(?P<month>\d{2})/(?P<day>\d+)/$', 'news.views.article_detail'),
)
```

/articles/2005/03/

news.views.month_archive(request, year='2005', month='03')

Admin

- odzielna aplikacja
- automatycznie generowny na podstawie modeli
- pozwala w dowolny sposób modyfikować modele
- prawa dostępu
- logowanie historii zmian
- możliwość dostosowania do potrzeb
- wielojęzyczny

Admin - Model

```
1  from mysite.myapp.models import *
2  from django.contrib import admin
3
4
5  class PersonAdmin(admin.ModelAdmin):
6      list_display = ('first_name', 'last_name')
7
8
9  class MembershipAdmin(admin.ModelAdmin):
10     list_display = ('person', 'group', 'date_joined')
11     list_filter = ('person', 'date_joined')
12     ordering = ('-date_joined',)
13     search_fields = ('group',)
14
15
16 admin.site.register(Person, PersonAdmin)
17 admin.site.register(Group)
18 admin.site.register(Membership, MembershipAdmin)
19
20
```

Admin

- Pokaz interfejsu admina na żywo

Autoryzacja

- użytkownicy
- uprawnienia
- role (grupy)
- wiadomości systemowe

Autoryzacja

```
from django.http import HttpResponseRedirect

def my_view(request):
    if not request.user.is_authenticated():
        return HttpResponseRedirect('/login/?next=%s' % request.path)
    # ...
```

```
from django.contrib.auth.decorators import login_required

def my_view(request):
    # ...
my_view = login_required(my_view)
```

```
from django.contrib.auth.decorators import login_required

@login_required
def my_view(request):
    # ...
```

Rejestracja

```
1  from django import oldforms as forms
2  from django.http import HttpResponseRedirect
3  from django.shortcuts import render_to_response
4  from django.contrib.auth.forms import UserCreationForm
5
6  def register(request):
7      form = UserCreationForm()
8
9      if request.method == 'POST':
10         data = request.POST.copy()
11         errors = form.get_validation_errors(data)
12         if not errors:
13             new_user = form.save(data)
14             return HttpResponseRedirect("/books/")
15     else:
16         data, errors = {}, {}
17
18     return render_to_response("registration/register.html", {
19         'form' : forms.FormWrapper(form, data, errors)
20     })
21
22
```

Cache'owanie

- obsługuje różne backendy
  - memcached, db, filesystem, localmem,
  - deweloperskie: simply, dummy
- zarządzanie
  - Ilość, timeout
- settings.py

```
CACHE_BACKEND = "locmem:///?timeout=30&max_entries=400"
```

Testowanie

```python
import unittest
from myapp.models import Animal

class AnimalTestCase(unittest.TestCase):
    def setUp(self):
        self.lion = Animal.objects.create(name="lion", sound="roar")
        self.cat = Animal.objects.create(name="cat", sound="meow")

    def testSpeaking(self):
        self.assertEquals(self.lion.speak(), 'The lion says "roar"')
        self.assertEquals(self.cat.speak(), 'The cat says "meow"')
```

```
$ ./manage.py test
```

```
$ ./manage.py test animals
```

```
$ ./manage.py test animals.AnimalTestCase
```

```
$ ./manage.py test animals.AnimalTestCase.testFluffyAnimals
```

Testowanie

```
import unittest
from django.test.client import Client

class SimpleTest(unittest.TestCase):
    def test_details(self):
        client = Client()
        response = client.get('/customer/details/')
        self.failUnlessEqual(response.status_code, 200)

    def test_index(self):
        client = Client()
        response = client.get('/customer/index/')
        self.failUnlessEqual(response.status_code, 200)
```

Podsumowanie

- prostota
- łatwość nauki
- duże możliwości
- wysoka wydajność
- stałe i aktywne rozwijanie
- stabilość (wiele dużych i udanych wdrożeń)

Django w sieci

- grono.net
- washingtonpost.com
- lawrence.com - wielokrotnie nagradzana witryna o lokalnych wydarzeniach kulturalnych, opowiadaniach, zespołach, drinkach itp.
- everyblock.com - ogólnodostępna baza przestępstw i zdarzeń z Chicago, New York i San Francisco
- LJWorld.com - witryna jednej z gazet

Metro

METRO
The District
Maryland
Virginia
Columns & Blogs
Crime
Education
Obituaries
Transportation
Traffic
Weather
Religion
Special Reports
Lottery

MORE
Going Out Guide
Local Business
Community Guides
High School Sports
Community Extras
LoudounExtra.com
Express Night Out

Information on Your Community

Use the menus below to search quickly.

Weather Traffic

Location | Blogs & Columns | Tools & Data

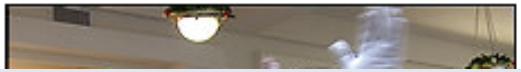
Ceilings Sag as Inspections Lag

District inspectors have failed to respond to reports chronicling hazards including collapsed ceilings, water leaks and no heat at almost a dozen apartment complexes, despite requests from the city agency charged with protecting tenants.

- **Special Report: Forced Out**

Donald Green in front of the apartment that is above his unit. (Michael Williamson/Post)

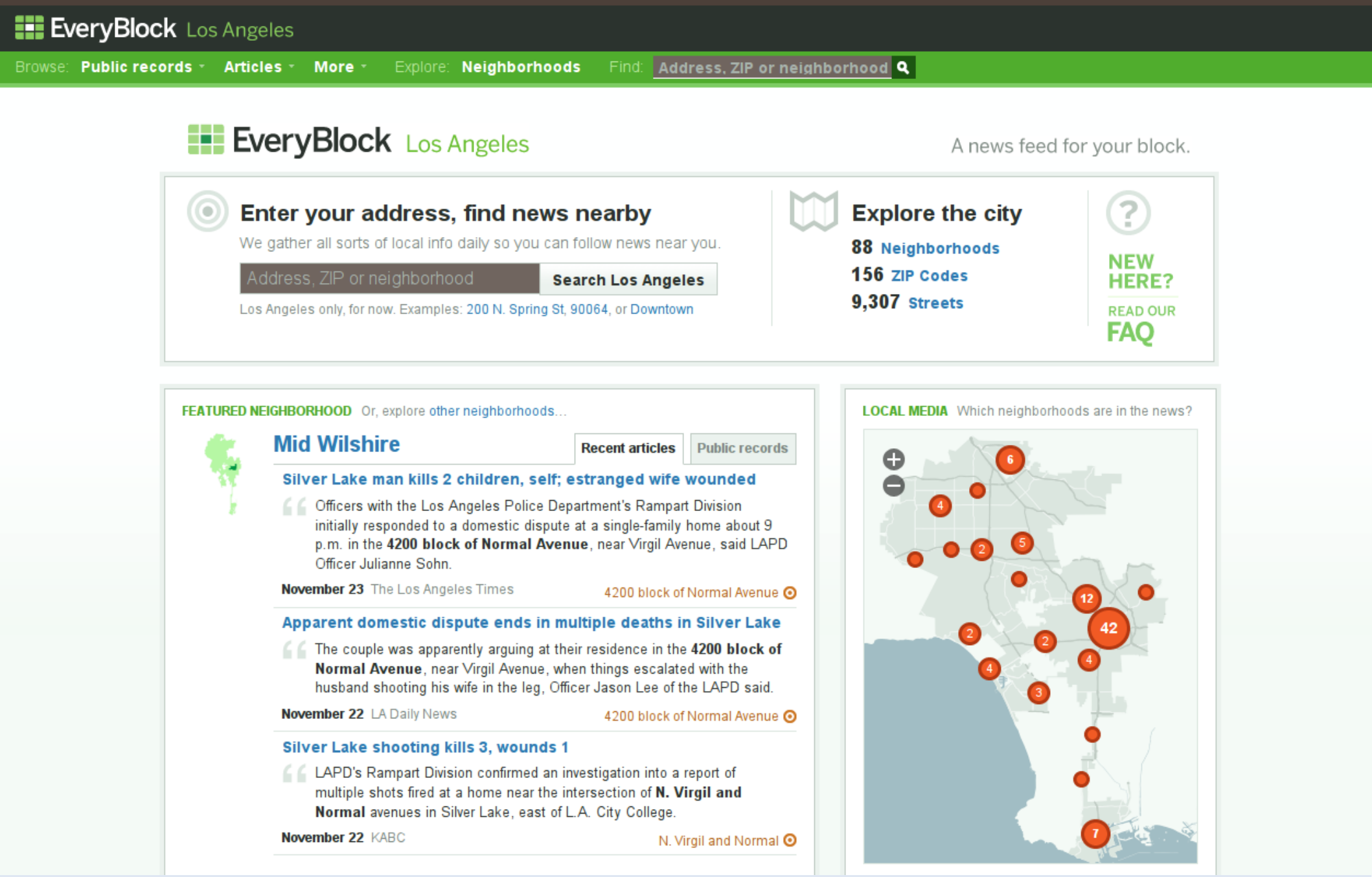
HOT Lanes Will Devour



GOING OUT GUIDE

Making History Again

The National Museum of American History reopens with music, tours and a weekend of


EveryBlock Los Angeles
Browse: Public records Articles More Explore: Neighborhoods Find: Address, ZIP or neighborhood
EveryBlock Los Angeles
A news feed for your block.
Enter your address, find news nearby
We gather all sorts of local info daily so you can follow news near you.
Address, ZIP or neighborhood
Search Los Angeles
Los Angeles only, for now. Examples: 200 N. Spring St, 90064, or Downtown
Explore the city
88 Neighborhoods
156 ZIP Codes
9,307 Streets
NEW HERE?
READ OUR FAQ
FEATURED NEIGHBORHOOD Or, explore other neighborhoods...
Mid Wilshire
Recent articles
Public records
Silver Lake man kills 2 children, self; estranged wife wounded
Officers with the Los Angeles Police Department's Rampart Division initially responded to a domestic dispute at a single-family home about 9 p.m. in the 4200 block of Normal Avenue, near Virgil Avenue, said LAPD Officer Julianne Sohn.
November 23 The Los Angeles Times
4200 block of Normal Avenue
Apparent domestic dispute ends in multiple deaths in Silver Lake
The couple was apparently arguing at their residence in the 4200 block of Normal Avenue, near Virgil Avenue, when things escalated with the husband shooting his wife in the leg, Officer Jason Lee of the LAPD said.
November 22 LA Daily News
4200 block of Normal Avenue
Silver Lake shooting kills 3, wounds 1
LAPD's Rampart Division confirmed an investigation into a report of multiple shots fired at a home near the intersection of N. Virgil and Normal avenues in Silver Lake, east of L.A. City College.
November 22 KABC
N. Virgil and Normal
LOCAL MEDIA Which neighborhoods are in the news?
6
4
2
5
12
42
2
2
4
4
3
7

Gdzie szukać dalej?

- **djangoproject.com** – oficjalna strona Django
- **www.djangobook.com** - darmowa książka o Django
- **django.pl** - Polska Społeczność Django

- svn co http://code.djangoproject.com/svn/djangoproject.com