T U T O R I A L που συνοδεύει την παρουσίαση:

«Γεωχωρικά συστήματα με Python - Πως να κάνετε μία εφαρμογή γραμμένη σε Django γεωγραφική χρησιμοποιώντας GeoDjango»

στην:

Συνάντηση Ελλήνων Προγραμματιστών Python, Αθήνα,
http://lanyrd.com/2011/pygr-june/, 8 Ιουνίου 2011

Υλικό (slides, software, το παρόν tutorial):
http://itia.ntua.gr/1151

Στέφανος Κοζάνης,
S.Kozanis@itia.ntua.gr, @skozan

Στο παρόν κείμενο περιγράφονται τα βήματα μετατροπής μίας «συμβατικής» εφαρμογής Python/Django σε μία εφαρμογή GIS με δυνατότητες γεωγραφικής απεικόνισης και εκτέλεσης geospatial queries. Η εφαρμογή με την οποία θα ασχοληθούμε είναι διαθέσιμη από την ιστοσελίδα http://itia.ntua.gr/1151 και έγινε για τις ανάγκες σχετικής παρουσίασης στην συνάντηση Ελλήνων Προγραμματιστών Python 2011 (8 Ιουνίου 2011).

Απαραίτητα για να τρέξει η εφαρμογή είναι: Python (≥2.6), Python/Django (≥1.2), Django-south (≥0.7), PostgreSQL (≥8.3). Στην συνέχεια για την εφαρμογή GIS είναι απαραίτητα PostGIS (≥1.3), libGEOS (≥3.1), python-gdal (≥1.6.3) και proj4. Ανάλογα με το σύστημα σας, ίσως και να μην χρειάζονται τα gdal, proj4, πιθανότατα να εγκαθίστανται μαζί με το PostGIS. Επιπλέον οι εκδόσεις σε παρένθεση είναι αυτές που έγιναν δοκιμές, ίσως τρέχει και σε παλαιότερες εκδόσεις. Τα συστήματα που δοκιμάστηκε είναι Ubuntu 10.04 και 11.04.

Αφού κατεβάσετε το αρχείο **2011-06-Kozanorama.tar.gz** και το αποσυμπιέσετε, θα δείτε δύο subdirectories: Στο **wogis** βρίσκεται η «συμβατική» εφαρμογή και στο **gis** βρίσκεται η GIS εφαρμογή.

Εγκατάσταση εφαρμογής wogis

Πρέπει να δημιουργηθεί μία βάση στην PosgreSQL, π.χ.:

```
#su -c 'createdb -U kozanorama -O soulman' postgres
```

όπου **kozanorama** είναι το όνομα της βάσης και **soulman** το όνομα χρήστη (καλό είναι να υπάρχει τόσο ως όνομα χρήστη βάσης όσο και ως όνομα χρήστη συστήματος για να κάνει login στη βάση με ident). Ο χρήστης soulman τίθεται ως owner της βάσης.

Πάμε στο /etc/postgresql/8.4/main ή όπου αλλού βρίσκεται το configuration της PostgreSQL και πειράζουμε τα pg_hba.conf και pg_ident.conf και μετά reload την PostgreSQL εφόσον θέλουμε να κάνουμε login με ident.

Βάζουμε τα στοιχεία σύνδεσης (χρήστης / password αν δεν κάνουμε χρήση ident) στο **settings.py**. Πάμε τώρα στην directory wogis/kozanorama και τρέχουμε:

```
wogis/kozanorama$ ./manage.py syncdb
wogis/kozanorama$ ./manage.py migrate core
```

Αν δεν θέλουμε να χρησιμοποιήσουμε *south* (προτείνω να έχετε *south* και να αφήσετε τον κώδικα ως έχει), μπορούμε να παραλείψουμε την εντολή migrate, αφού πρώτα έχουμε αφαιρέσει το south από τα INSTALLED_APPS στο **settings.py**. Σε αυτήν την περίπτωση όμως θα πρέπει να drop την βάση και να την ξαναδημιουργήσουμε στο β' στάδιο της παρουσίασης με το gis.

Αν τώρα τρέξουμε τον development server με:

```
wogis/kozanorama$ ./manage.py runserver 0.0.0.0:8000
```

τότε έχουμε τις ιστοσελίδες στην http://localhost:8000/ και το admin interface στο http://localhost:8000/admin/ (στο οποίο μπαίνουμε με το user / pass που δηλώσαμε στο πρώτο syncdb του Django).

Στην συνέχεια θα βάλουμε και λίγα δεδομένα, εναλλακτικά θα μπορούσαμε να τα βάλουμε με το χέρι π.χ. μέσω του admin interface. Κατεβαίνουμε στην directory **wogis** και δίνουμε:

```
wogis$ psql -d kozanorama -U soulman -f initial_data.sql
```

όπου στη θέση του soulman βάζετε το όνομα χρήστη. Τώρα η βάση έχει γεμίσει και αν ξεκινήσουμε πάλι την εφαρμογή με *manage.py runserver 0.0.0.0:800* και επισκεφτούμε την ιστοσελίδα http://localhost:8000/ θα δούμε να τρέχει η εφαρμογή με φουλ δεδομένα.

Περιγραφή της «συμβατικής» εφαρμογής

Για τις ανάγκες της παρουσίασης, επινοήθηκε ένας φανταστικός οδηγός διασκέδασης με τίτλο «ΚΟΖΑΝΟΡΑΜΑ» (κατά το ΑΘΗΝΟΡΑΜΑ, σε συνδυασμό με το επίθετό μου :-) Η προτεινόμενη βάση για την παρουσίαση μάλιστα έχει εξ' ολοκλήρου φανταστικά δεδομένα. Χαρακτηριστικές μάλιστα είναι οι 7 φανταστικές περιοχές της Αθήνας που φαίνονται στον παρακάτω χάρτη.

Σχήμα 1: Χάρτης της Αθήνας με τις περιοχές του «ΚΟΖΑΝΟΡΑΜΑ»

Τα βασικά Entities της βάσης είναι Κινηματογράφοι ή Σινεμάδες όπως αποκαλούνται στην εφαρμογή (Cinema), τα Ψητοπωλεία (Psitopoleio), οι ταινίες (Film) και οι Περιοχές (Region). Κάθε Cinema έχει FK στα Film. Επιπλέον Cinema, Psitopoleio έχουν FK στα Region. Τα Cinema, Psitopoleio είναι τοποθετημένα σε έναν μονοδιάστατο χώρο όπου δεν έχουν οριστεί μετρικές (ούτε αποστάσεις ούτε γειτνιάσεις). Έτσι μπορούν να εκτελεστούν π.χ. Lookups ανά περιοχή, το αποτέλεσμα

όμως του lookup δεν λέει τίποτα για την πραγματική απόσταση, ούτε υπάρχει η δυνατότητα αναζήτησης μεταξύ δύο γειτονικών περιοχών.

Η Django Application της περίπτωσής μας ονομάζεται **core** και βρίσκεται κάτω από την **wogis/kozanorama/core**. Για να είναι εκτελέσιμη, την βάζουμε στην λίστα **INSTALLED_APPS** (ως kozanorama.core) στο **settings.py**. Στο **core/models.py** ορίζονται τα σχετικά μοντέλα του Django ως οι παρακάτω κλάσεις:

```
from django.db import models

class Region(models.Model):
    name = models.CharField(max_length=128)
    def __unicode__(self):
        return self.name


class Film(models.Model):
    name = models.CharField(max_length=128)
    def __unicode__(self):
        return self.name


class Cinema(models.Model):
    name = models.CharField(max_length=128)
    region = models.ForeignKey(Region)
    film = models.ForeignKey(Film)
    def __unicode__(self):
        return self.name

    class Meta:
        verbose_name = "Cinemas"
        verbose_name_plural = "Cinemades"


class Psitopoleio(models.Model):
    name = models.CharField(max_length=128)
    rating = models.IntegerField(default=2)
    region = models.ForeignKey(Region)
    def asterakia(self):
        return self.rating*'* '
    def __unicode__(self):
        return self.name

    class Meta:
        verbose_name = "Psitopoleio"
        verbose_name_plural = "Psitopoleia"
```

Όπως βλέπουμε δεν έχει κάτι το ιδιαίτερο. Και τα τέσσερα μοντέλα έχουν ένα πεδίο ονομασίας: "name". Στο Ψητοπολείο έχουμε και έναν ακέραιο με την βαθμολογία του ψητοπωλείου σε αστεράκια. Πάμε στο view layer, ορίζεται μέσω του **core/views.py**:

```
from django.shortcuts import render_to_response, get_object_or_404
from django.views.generic import list_detail

from kozanorama.core.models import *
from kozanorama.core.decorators import *
```

```
def cinema_detail(request, *args, **kwargs):
    cinema = get_object_or_404(Cinema, pk=kwargs["object_id"])
    pshtopoleia_sthn_perioxh = \
        Psitopoleio.objects.filter(region__id=cinema.region.id).order_by('-
rating')
    kwargs["request"] = request
    kwargs["template_name"] = "cinema_detail.html"
    kwargs["extra_context"] =
{"pshtopoleia_sthn_perioxh":pshtopoleia_sthn_perioxh, }
    return list_detail.object_detail(*args, **kwargs)


def psitopoleio_detail(request, *args, **kwargs):
    psitopoleio = get_object_or_404(Psitopoleio, pk=kwargs["object_id"])
    cinemades_sthn_perioxh = \
        Cinema.objects.filter(region__id=psitopoleio.region.id).order_by('film__
name')
    kwargs["request"] = request
    kwargs["template_name"] = "psitopoleio_detail.html"
    kwargs["extra_context"] =
{"cinemades_sthn_perioxh":cinemades_sthn_perioxh, }
    return list_detail.object_detail(*args, **kwargs)


@sort_by
@filter_by
def cinema_list(request, queryset, *args, **kwargs):
    kwargs["template_name"] = "cinema_list.html"
    queryset = queryset
    return list_detail.object_list(request,queryset=queryset, *args, **kwargs )


@sort_by
@filter_by
def psitopoleio_list(request, queryset, *args, **kwargs):
    kwargs["template_name"] = "psitopoleio_list.html"
    queryset = queryset
    return list_detail.object_list(request,queryset=queryset, *args, **kwargs )
```

Πάλι τίποτα το ιδιαίτερο. Στις λίστες των ψητοπωλείων και των cinema υπάρχει ένας decorator @sort_by ο οποίος προτρέπει τα αποτελέσματα των lookups να είναι ταξινομημένα σύμφωνα με το URL param ?sort=... Ο decorator @filter_by επιστρέφει φιλτραρισμένα αποτελέσματα π.χ. αν τεθεί σε URL param: ?region=... Μπορείτε να δείτε και το **decorators.py**.

Τα cinema_list, psitopoleio_list επιστρέφουν μία πλήρη λίστα με Σινεμάδες, Ψητοπωλεία. Τα cinema_detail, psitopoleio_detail επιστρέφουν τα χαρακτηριστικά ενός ψητοπωλείου ή ενός σινεμά, επιπλέον έχουν ένα subquery που επιστρέφει για παράδειγμα, τους σινεμάδες της περιοχής που βρίσκεται το ψητοπωλείο κλπ.

Το **admin.py** υπάρχει εφόσον έχουμε βάλει στο **settings.py/INSTALLED_APPS** την εφαρμογή `django.contrib.admin`. Δεν έχει κάτι πέρα από τα τέσσερα μοντέλα, ώστε να τα βλέπουμε στο admin interface. Στο **kozanorama/templates** είναι τα html templates, βλ. και το σχετικό στο **settings.py**, `TEMPLATE_DIRS`.

Εφαρμογή gis

Δουλεύοντας λίγο με την «συμβατική» εφαρμογή βλέπουμε πως δεν εξυπηρετεί απόλυτα. Δηλαδή βρίσκουμε την ταινία που θέλουμε σε ποιον Σινεμά το παίζει και σε ποια περιοχή. Έπειτα μας λέει ποια ψητοπωλεία είναι σε αυτήν την περιοχή. Αυτό δεν βοηθάει 100% καθώς μας ενδιαφέρει να ξέρουμε πόσο κοντά ακριβώς είναι τα ψητοπωλεία στον Σινεμά που θα πάμε, ακόμα και αν δεν είναι στην ίδια περιοχή της Αθήνας (μπορεί ο Σινεμάς να είναι κοντά στο σύνορο μεταξύ δύο περιοχών). Το πρόβλημα θα λυνόταν αν οι οντότητες είχαν περιγραφεί με την ακριβή θέση στο χώρο και δημιουργούνταν μεταξύ τους σχέσεις αποστάσεων, επικάλυψης κλπ. Αυτό γίνεται χρησιμοποιώντας GeoDjango (django.contrib.gis) στην θέση του Django. Το django.contrib.gis είναι διαθέσιμο με την κύρια εγκατάσταση του Django.

Εγκατάσταση

Η εφαρμογή βρίσκεται στην directory **gis**. Σε περίπτωση που διατηρήσετε την «συμβατική» εφαρμογή (**wogis**) με *django.south* τότε ήδη έχει δημιουργηθεί η βάση που θα δουλέψουμε (kozanorama), αλλιώς θα πρέπει να κάνετε drop και ξανά createdb. Αυτό που θα κάνουμε τώρα είναι να εγκαταστήσουμε τα extensions του PostGIS στην βάση. Αυτό γίνεται ως εξής:

```
createlang -U postgres plpgsql kozanorama
psql -d kozanorama -U postgres -f postgis.sql
psql -d kozanorama -U postgres -f postgis_comments.sql
psql -d kozanorama -U postgres -f spatial_ref_sys.sql
psql -U postgres kozanorama
  grant select on spatial_ref_sys to soulman;
  grant all on geometry_columns to soulman;
  \q
```

Τα *.sql αρχεία βρίσκονται ανάλογα την εγκατάσταση σε διαφορετικούς καταλόγους. Για το Ubuntu 10.04 π.χ. είναι στο: /usr/share/postgresql/8.4/contrib

Το παραπάνω λίγο πιο quick'n dirty, τρέξτε τα *.sql με -U soulman, όπου soulman το όνομα του χρήστη στον οποία ανήκει η βάση. Μετά δεν χρειάζονται τα grant.

Πάμε τώρα να κάνουμε migrate (με django.south) στο νέο σχήμα όπου περιέχονται και τα γεωγραφικά features:

```
gis/kozanorama$ ./manage.py migrate core
```

Εννοείται πως έχουμε βάλει τα σωστά στοιχεία σύνδεσης στο settings.py (username / password εφόσον αυτό χρειάζεται). Αν δεν έχετε *south* θα κάνετε μόνο *syncdb* και όχι *migrate core*. Αν δημιουργείτε τη βάση πρώτη φορά (ανεξαρτήτως αν έχετε ή όχι south) και δεν κάνετε migrate από το **wogis**, θα πρέπει να βάλετε τα **initial_data.sql** με τον τρόπο που περιγράψαμε στο **wogis**.

Στο σημείο αυτό πρέπει να δώσουμε συντεταγμένες στα σημεία των Ψητοπωλείων και των Σινεμάδων. Σε πραγματικές συνθήκες θα μπορούσε να γίνει π.χ.

με το gdal, ή με το layer mapping του GeoDjango (βλ. tutorial GeoDjango). Στην περίπτωσή μας γίνεται με ένα σκριπτάκι που βάζει τις οντότητες σε τυχαίες θέσεις, λαμβάνοντας υπόψη τον χάρτη του Σχήματος 1. Δώστε την custom command:

```
gis/kozanorama$ ./manage.py random_points
```

και είστε έτοιμοι! Αν έχουν πάει όλα καλά με το:

```
wogis/kozanorama$ ./manage.py runserver 0.0.0.0:8000
```

στήνετε τον development server και στην http://localhost:8000/ έχετε την γεωγραφική εφαρμογή με χάρτες κλπ.

Περιγραφή

Τα βήματα για να γίνει η εφαρμογή γεωγραφική είναι τα παρακάτω:

Στο **settings.py**, στα **ISTALLED_APPS** βάζουμε το **django.contrib.gis**:

```
    'django.contrib.sites',
    'django.contrib.messages',
# GIS: Add the django.contrib.gis to installed apps
    'django.contrib.gis',
    'south',
```

Πάμε στο **core/models.py**. Αποφασίζουμε πως τα Cinema, Psitopoleio έχουν γεωγραφική υπόσταση και θα τους βάλουμε geospatial field. Θα μπορούσαμε και στις περιοχές (Region), αυτό όμως θα πάει λίγο μακριά την εφαρμογή σε σχέση με της ανάγκες της παρουσίασης, επιπλέον το μόνο πλεονέκτημα που θα μας παρείχε θα ήταν η απεικόνιση στον χάρτη των περιοχών.

Αντικαθιστούμε το **django.db** με το **django.contrib.gis.db**:

```
#from django.db import models
#GIS: use django.contrib.gis.db instead of
#     django.db
from django.contrib.gis.db import models
from django.contrib.gis.measure import D
from django.contrib.gis.geos import Point
```

Στο Μοντέλο Cinema προσθέτουμε το πεδίο **ref_point** το οποίο είναι ένα σημείο στον χώρο:

```
class Cinema(models.Model):
    name = models.CharField(max_length=128)
    region = models.ForeignKey(Region)
    film = models.ForeignKey(Film)
    #GIS: Add point geospatial field...
    ref_point = models.PointField(null=True, blank=True)
```

Επιπλέον βάζουμε τον GeoManager ώστε να μπορούμε στα Cinema.objects να

κάνουμε γεωγραφικά queries.

```
    #GIS: Replace the default manager with GeoManager to
    #     enable geospatial queries
    objects = models.GeoManager()
```

Οι επόμενες function είναι προαιρετικές, τις έκανα για να μπορώ να έχω έναν αριθμό από τα / και τα κοντινά ψητοπωλεία όταν δείχνω την λίστα με τους Σινεμάδες:

```
    #GIS: add these methods for ajax table
    def kontina_psitopoleia_count(self):
        return
Psitopoleio.objects.filter(ref_point__distance_lte=(self.ref_point,
D(km=2.5))).count()
    def kontinotero_psitopoleio(self):
        return Psitopoleio.objects.distance(self.ref_point).order_by('distance')
[0]
```

Για τα παραπάνω queries θα αναφερθούμε στο view layer.

Όπως τροποποιούμε το Cinema, το ίδιο κάνουμε και στο Psitopoleio:

```
class Psitopoleio(models.Model):
    name = models.CharField(max_length=128)
    rating = models.IntegerField(default=2)
    region = models.ForeignKey(Region)
    #Add point geospatial field...
    ref_point = models.PointField(null=True, blank=True)
    #GIS: Replace the default manager with GeoManager to
    #     enable geospatial queries
    objects = models.GeoManager()
    def asterakia(self):
        return self.rating*'* '
    #GIS: add these methods for ajax table
    def kontina_cinema_count(self):
        return Cinema.objects.filter(ref_point__distance_lte=(self.ref_point,
D(km=2.5))).count()
    def kontinotero_cinema(self):
        return Cinema.objects.distance(self.ref_point).order_by('distance')[0]
    def __unicode__(self):
        return self.name

    class Meta:
        verbose_name = "Psitopoleio"
        verbose_name_plural = "Psitopoleia"
```

Στο **core/admin.py** αντικαθιστούμε το **django.contrib.admin** με **django.contib.gis.admin**. Στην συνέχεια στα ψητοπωλεία και στους σινεμάδες χρησιμοποιούμε **GeoModelAdmin** αντί **ModelAdmin**:

```
#from django.contrib import admin
# GIS Admin
from django.contrib.gis import admin
from kozanorama.core.models import *

admin.site.register(Region, admin.ModelAdmin)
admin.site.register(Cinema, admin.GeoModelAdmin)
admin.site.register(Psitopoleio, admin.GeoModelAdmin)
```

```
admin.site.register(Film, admin.ModelAdmin)
```

Αυτό μας δίνει την δυνατότητα να έχουμε έναν χάρτη στο admin interface σε κάθε οντότητα.

Πάμε στο **core/views.py**. Κάνουμε import τα παρακάτω:

```
#GIS specific modules
from django.contrib.gis.measure import D
from django.contrib.gis.shortcuts import render_to_kml
from django.contrib.gis.geos import Polygon, Point
```

Το D μας επιτρέπει να διατυπώνουμε στα queries αποστάσεις, με το render_to_kml θα φτιάξουμε το KML service για την σχεδίαση του χάρτη. Τα Point, Polygon είναι γεωγραφικές (geospatial) δομές του πακέτου geos.

Το KML service δίνεται από την παρακάτω function:

```
#GIS: return map display data as KML
#     Psitopoleio, Cinema in different layer, get the GET parameter to
#     return the desired model data.
def kml(request):
    model_classes = {'psitopoleio': Psitopoleio, 'cinema': Cinema,}
    try:
        if request.GET.__contains__('model'):
            model_class = model_classes[request.GET['model']]
        else:
            raise
        queryres = model_class.objects.all()
    except Exception, e:
        raise Http404
    for arow in queryres:
        if arow.ref_point:
            arow.kml = arow.ref_point.kml
    return render_to_kml("placemarks.kml", {'places': queryres})
```

Η javascript εφαρμογή στις αντίστοιχες ιστοσελίδες (βλ. τέλος tutorial), περνάει στο URL την παράμετρο model που μπορεί να είναι είτε cinema, είτε psitopoleio ώστε να παραχθεί αντίστοιχο KML χωριστά για τους σινεμάδες και τα ψητοπωλεία, έτσι ώστε να είναι χωριστά layer στον χάρτη που θα μπορεί να τα εμφανίζει ή να τα κρύβει ο χρήστης. Το **placemarks.kml** είναι το template για να παρχθεί η KML και βρίσκεται στην template directory και είναι το παρακάτω:

```
{% extends "gis/kml/base.kml" %}
{% block placemarks %}{% for place in places %}
  <Placemark>
    <name>{% if place.name %}{{ place.name }}{% else %}{{ place }}{% endif
%}</name>
    <id>{{ place.id }}</id>
    {{ place.kml|safe }}
  </Placemark>{% endfor %}{% endblock %}
```

όπως βλέπουμε τα tags <name> και <id> είναι custom tags. Επιστρέφουμε στο **views.py**. Έστω πως θέλουμε να δούμε τα details ενός σινεμά και τα κοντινά σε

αυτό ψητοπωλεία. Αντικαθιστούμε το φίλτρο βάσει του region_id (. filter(region__id=cinema.region.id) με το παρακάτω:

```
def cinema_detail(request, *args, **kwargs):
    cinema = get_object_or_404(Cinema, pk=kwargs["object_id"])
#    pshtopoleia_sthn_perioxh = \
#        Psitopoleio.objects.filter(region__id=cinema.region.id).order_by('-
rating')
#GIS: Filter psitopoleia results by maximum distance from cinema=2.5km,
#     Then sort results by distance from cinema.
    pshtopoleia_sthn_perioxh = \
        Psitopoleio.objects.filter(ref_point__distance_lte=(cinema.ref_point,
                                   D(km=2.5))).distance(cinema.ref_point).order_
by('distance')
    kwargs["request"] = request
    kwargs["template_name"] = "cinema_detail.html"
    kwargs["extra_context"] =
{"pshtopoleia_sthn_perioxh":pshtopoleia_sthn_perioxh, }
    return list_detail.object_detail(*args, **kwargs)
```

Το .objects όπως ορίσαμε στα **models.py** βασίζεται στον GeoManager. Το .filter(ref_point__distance_lte=(cinema.ref_point, D(km=2.5))) είναι ένα distance query, και περιγράφεται στην τεκμηρίωση του GeoDjango. Το συγκεκριμένο επιστρέφει τα ψητοπωλεία που είναι σε μία ακτίνα 2.5 km από τον Σινεμά. Στην συνέχεια ταξινομούμε τα αποτελέσματα σύμφωνα με την εγγύτητα από τον σινεμά με: .distance(cinema.ref_point).order_by('distance'). Η μέθοδος .distance μας δημιουργεί ένα "virtual" πεδίο distance με το οποίο μπορούμε να κάνουμε στη συνέχεια order_by και επιπλέον να το απεικονίσουμε στο html template χρησιμοποιώντας όπως βλέπουμε στο **cinema_detail.html**:

```
{% for item in pshtopoleia_sthn_perioxh %}
…
    <td>{{ item.distance.km|floatformat:2 }}</td>
```

Με τον ίδιο ακριβώς τρόπο λειτουργεί και το psitopoleio_detail()

Για την λίστα των ψητοπωλείων, σινεμάδων, καταργούμε τις αρχικές function cinema_list() Και psitopoleio_list() που είχαμε στη συμβατική εφαρμογή (οι σελίδες αυτές θα σερβίρονται πλέον static, βλ. και **urls.py**). Στη GIS εφαρμογή, θα δίδεται λίστα των σινεμάδων ή των ψητοπωλείων σύμφωνα με το viewport του χάρτη και θα ανανεώνεται όταν γίνεται zoom/unzoom και pan. Αυτό το κάνουμε με λίγο AJAX. Ο server απαντάει στα request που προκαλούνται από τις κινήσεις στον χάρτη με την παρακάτω function:

```
def object_list_ajax(request, queryset, *args, **kwargs):
    if request.GET.__contains__('bound_box'):
        minx,miny,maxx,maxy = [float(x) for x in
request.GET['bound_box'].split(',')]
        geom=Polygon(((minx,miny),(minx,maxy),(maxx,maxy),(maxx,miny),
(minx,miny)),srid=4326)
        center=Point(0.5*(minx+maxx), 0.5*(miny+maxy), srid=4326)
    else:
        raise Http404
#GIS:  Check if point is inside the map bounding box
```

```
    queryset =
queryset.filter(ref_point__bboverlaps=geom).distance(center).order_by('distance'
)
    return list_detail.object_list(request,queryset=queryset, *args, **kwargs )
```

Δηλαδή, διαβάζουμε τα URL params: *minx*, *maxx*, *miny* και *maxy*. Από τα σημεία αυτά δημιουργείται ένα ορθογώνιο παραλληλόγραμμο, το geom, το οποίο είναι το viewport του χάρτη. Το query που γίνεται φιλτράρεται σύμφωνα με την τομή του πολυγώνου και των σημείων των οντοτήτων: .filter(ref_point__bboverlaps=geom). Όπου ref_point το σημείο αναφοράς του σινεμά ή του ψητοπωλείου. βλ. σχετικά στα Spatial Queries στην τεκμηρίωση του GeoDjango. Στο τέλος ταξινομούμε τα αποτελέσματα σύμφωνα με την απόσταση από το κέντρο center του χάρτη. Προσέξτε τόσο στον υπολογισμό του geom όσο και του center την τιμή srid=4326, αυτή αντιπροσωπεύει το σύστημα WGS-84 που χρησιμοποιείται από την εφαρμογή του χάρτη. Η χρήση της object_list_ajax γίνεται από την **urls.py** ως εξής:

```
views.object_list_ajax,
    dict(cinemas.items()+[['template_name', 'cinema_list_ajax.html'],]),
     'cinema_list_ajax'),
    (r'^psitopoleia_ajax/$',
     views.object_list_ajax, dict(psitopoleia.items()+[['template_name',
'psitopoleio_list_ajax.html'],]),
     'psitopoleia_list_ajax'),
```

Αυτό είναι όλο, μπορούμε να δούλε λίγο τι γίνεται και στις σελίδες με τους χάρτες. Η απεικόνιση του χάρτη γίνεται με χρήση μίας javascript library, της OpenLayers (http://openlayers.org/)

Στο **site_media/js** θα βρούμε δύο custom javascript, το olmap-settings.js το οποίο τρέχει πρώτο και το olma.js που τρέχει δεύτερο. Ας τα δούμε λίγο:

olmap-settings.js

Στις επτά πρώτες γραμμές ορίζονται τα όρια του χάρτη (ΝοτιοΔυτικά -> ΒορειοΑνατολικά):

```
var bl_point = new OpenLayers.LonLat(23.65,37.95);
var tr_point = new OpenLayers.LonLat(23.78,38.04);

var bounds = new OpenLayers.Bounds();
bounds.extend(bl_point);
bounds.extend(tr_point);
bounds.transform(new OpenLayers.Projection("EPSG:4326"),
                 new OpenLayers.Projection("EPSG:900913"));
```

Στην συνέχεια ορίζουμε διάφορα layers με χαρτογραφικό υπόβαθρο. Το Κτηματολόγιο Α.Ε. δίνει και αυτό πολύ καλό υπόβαθρο (αεροφωτογραφίες) για την Ελλάδα σε μορφή WMS service. Τα osm, ocm είναι από το OpenStreetMap (http://openstreetmap.org/)

```
var wms1_base = new OpenLayers.Layer.WMS("Υπόβαθρο «ΚΤΗΜΑΤΟΛΟΓΙΟ Α.Ε.»",
  "http://gis.ktimanet.gr/wms/wmsopen/wmsserver.aspx",
    {   layers: 'KTBASEMAP', transparent: false},
```

```
    {   isBaseLayer: true,
        projection: new OpenLayers.Projection("EPSG:900913"),
        iformat: 'image/png', maxExtent: bounds, numZoomLevels:
        15, units: "m", maxResolution: 900,
        attribution: ""});

var osm = new OpenLayers.Layer.OSM.Mapnik("Υπόβαθρο \"Open Street Map\"",
{isBaseLayer: true,
        attribution: "Map by <a href='http://www.openstreetmap.org/'>OSM</a>"});

var ocm = new OpenLayers.Layer.OSM.CycleMap("Υπόβαθρο \"Open Cycle Map\"",
{isBaseLayer: true,
        attribution: "Map by <a href='http://www.openstreetmap.org/'>OSM</a>"});

/* Uncoment to use google maps layer */
// var google_map = new OpenLayers.Layer.Google("Google map",
{isBaseLayer:true});
/* Note, in order to use google maps, you should include the google
 * maps API in the main html file, provided by Google.*/

/* Add base layers to this array by the order of apearance */
var base_layers = [osm, ocm, wms1_base];
```

Εικονίδια για τις οντότητες, ένας γύρος πίτα και ένα σινεμά :-)

```
var markers_icons = {psitopoleio: 'images/gyros.png',
                     cinema: 'images/cinema.png'};
```

Πάμε να δούμε και το κυρίως js, το **olmap.js**:

Μία βοηθητική function που χρησιμοποιώ για να κάνω join δύο attribute array:

```
Object.extend = function(destination, source) {
    for (var property in source)
        destination[property] = source[property];
    return destination;
};

var map = null;
```

Η παρακάτω function μου χρειάζεται καθώς κάνω clustering. Δηλαδή αν πέσουν μαζί πάνω από 2 features σε μία ακτίνα μικρότερη από 20 pixel, γίνεται σμήνος (cluster). Με την παρακάτω ζητάω τα attributes της πρώτης οντότητας που ανήκει στο cluster.

```
function get_attribute(afeature, attrib)
{
    if(!afeature.cluster)
        return afeature.attributes[attrib];
    else
        return afeature.cluster[0].attributes[attrib];
}
```

Με αυτήν την function δημιουργώ τα layers μου. Είναι πάρα πολύ αυτά που κάνει, για περισσότερα πρέπει να δείτε το documentation του OpenLayers. Προσέξτε που στο params['model'] περνάω τη μεταβλητή ObjectName. Αυτή θα είναι είτε "cinema" είτε "psitopoleio". Στην συνέχεια αυτά τα params περνάνε στο URL ως url params, τα διαβάζει ο server στο request και σερβίρει ανάλογα είτε σινεμάδες είτε

ψητοπωλεία (βλ. και views.py:kml())

```
function CreateLayer(AName, ObjectName){
    var params={request: 'GetFeature'};
    params['srs'] = 'EPSG:4326';
    params['version'] = '1.0.0';
    params['service'] = 'WFS';
    params['format'] = 'WFS';
    params['model'] = ObjectName;
    var labelvalue = "";
    var labeling_opts = {
            label : labelvalue, fontColor: "#704060",
            fontSize: "10px", fontFamily: "Verdana, Arial",
            fontWeight: "bold", labelAlign: "cm"
    };
    var general_opts = {
            externalGraphic: MEDIA_URL+markers_icons[ObjectName],
            graphicWidth: 32, graphicHeight:32, graphicXOffset:-15,
            graphicYOffset: -34, fillOpacity: 1
    };
    general_opts = Object.extend(general_opts, labeling_opts);
    AURL = ROOT_URL+"kml/";
```

Στα strategies είναι το Fixed που σημαίνει πως φέρνει όλα τα σημεία άπαξ και Cluster που σημαίνει πως κάνει Clustering (βλ. πάνω). Πρωτόκολλο μεταφοράς είναι το HTTP, format το KML, params οι παραπάνω URL params που αναφερθήκαμε.

```
    var alayer = new OpenLayers.Layer.Vector(AName,
    {
        strategies: [
                    new OpenLayers.Strategy.Fixed(),
                    new OpenLayers.Strategy.Cluster({distance: 20,
                    threshold: 2})
                    ],
        protocol: new OpenLayers.Protocol.HTTP({
                        url: AURL,
                        format: new OpenLayers.Format.KML(),
                        params: params}),
        projection: new OpenLayers.Projection("EPSG:4326"),
        formatOptions: { extractAttributes:true },
        styleMap: new OpenLayers.StyleMap({
            "default": new OpenLayers.Style(
                  OpenLayers.Util.applyDefaults(general_opts,
                        OpenLayers.Feature.Vector.style["default"]),
                        {context: {aname: function(feature) { return
get_attribute(feature, "name"); }}})
        })
    } );
```

Δώστε σημασία στο "${aname}" και πιο πάνω εκεί που έχει context: {… , έτσι διαβάζεται το attribute *name* που μεταφέρεται με το KML και απεικονίζεται στον χάρτη ως label του ψητοπωλείου ή του σινεμά.

```
    var defaultStyle = alayer.styleMap.styles["default"].defaultStyle;
    defaultStyle["label"]="${aname}";
    alayer.styleMap.styles["default"].setDefaultStyle(defaultStyle);
    return alayer;
}
```

Τώρα χρησιμοποιώντας την CreateLayer φτιάχνουμε τα δύο Layers: Ψητοπωλεία και Σινεμάδες, μετά τα βάζουμε και σε ένα array.

```
var psitopoleia = CreateLayer("Ψητοπωλεία", "psitopoleio");
var cinemades = CreateLayer("Σινεμάδες", "cinema");
var categories = [psitopoleia, cinemades];
```

Η παρακάτω init καλείται π.χ. από το html με <body onload=”init()”>, βλ. και base.html στα templates.

```
function init() {
    var options = {
        'units' :   "m",
        'numZoomLevels' :   15,
        'sphericalMercator': true,
        'maxExtent': bounds,
        'projection'    :   new OpenLayers.Projection("EPSG:9009313"),
        'displayProjection':    new OpenLayers.Projection("EPSG:4326")
    };
```

Η μεταβλητή map είναι το αντικείμενο OpenLayers που αντιπροσωπεύει τον χάρτη, το 'map' στην παρένθεση είναι ένα html element π.χ. το <div id='#map>, βλ. cinema_list.html στα templates.

```
    map = new OpenLayers.Map('map', options);
    map.addControl(new OpenLayers.Control.ScaleLine());
```

Ακολουθούν διάφορα controls όπως για να κάνουμε zoom in – out, κλπ.

```
    var ANavToolBar = new OpenLayers.Control.NavToolbar();
        map.addControl(ANavToolBar);
    $("div.olControlNavToolbar").css("top","14px");
    $("div.olControlNavToolbar").css("left","11px");
    pzb = new OpenLayers.Control.PanZoomBar();
    pzb.zoomWorldIcon = false;
    map.addControl(pzb);
    map.addControl(new OpenLayers.Control.MousePosition());
    map.addControl(new OpenLayers.Control.OverviewMap());
    ls = new OpenLayers.Control.LayerSwitcher();
    map.addControl(ls);
    map.addLayers(base_layers);
    map.addLayers(categories);
    map.zoomToExtent(bounds);
```

Τρέχουμε μία φορά το ajax_fetch για να γεμίσει ο πίνακας στην σελίδα με τους σινεμάδες. Μετά ορίζουμε ένα event ώστε σε κάθε pan zoom in /out να τρέχει το ajax_fetch.

```
    ajax_fetch();
    map.events.register("move", map, ajax_fetch);
    ls.baseLbl.innerHTML='Base layers';
    ls.dataLbl.innerHTML='Data layers';
}
```

Αυτό είναι το ajax_fetch, μεταφέρει το map.getExtent() ως URL params έτσι ώστε να επιστρέψει ο πίνακας με τα ορατά αποτελέσματα. Το AJAX_URL ορίζεται στο template, π.χ. στο **cinema_list.html**

```
function ajax_fetch(evt) {
   $.get(AJAX_URL, {
        bound_box: map.getExtent().transform(new
OpenLayers.Projection("EPSG:900913"), new
OpenLayers.Projection("EPSG:4326")).toBBOX(7)
   }, function(data){
       $('#list').html(data);
   });
}
```

Σχετικό υλικό

Django: https://www.djangoproject.com/

GeoDjango: http://geodjango.org/

GeoDjango tutorial:
https://docs.djangoproject.com/en/dev/ref/contrib/gis/tutorial/

Άλλα σχετικά tutorials:
http://www.oscon.com/oscon2009/public/schedule/detail/7844
http://invisibleroads.com/tutorials/geodjango-googlemaps-build.html

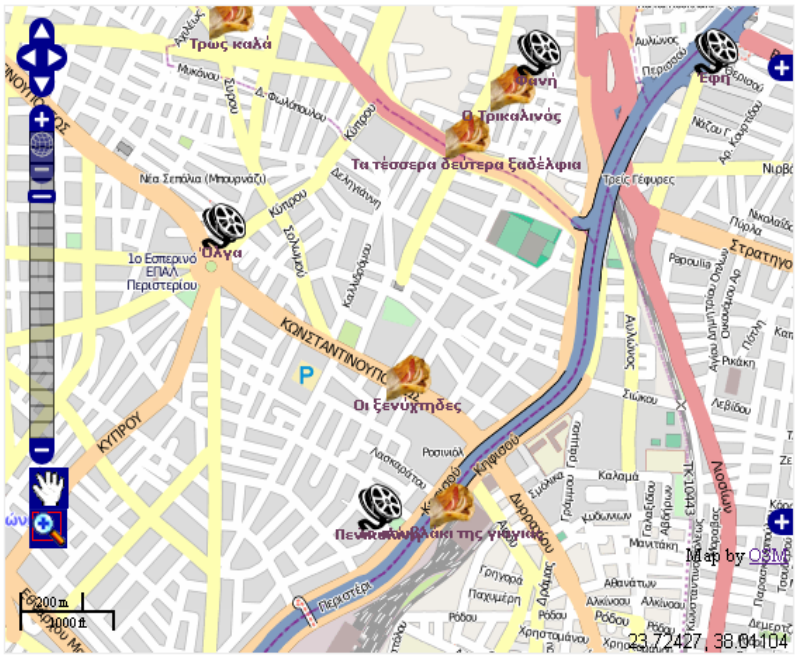
OpenLayers: http://openlayers.org/

PostGIS: http://postgis.refractions.net/

Διάφορα screens

Το ΚΟΖΑΝΟΡΑΜΑ

Σινεμάδες



id	Ονομασία	Ταινία	Περιοχή	Ψητοπωλεία σε ακτίνα 2.5 km	Κοντινότερο ψητοπωλείο
44	Όλγα	Ο Παπασούζας	Νέα Τεμπέλη	11	Τρως καλά, 0.54 km
35	Πενικυλίνη	Οι Χούλιγκανς	Λούτζα	16	Το σουβλάκι της γιαγιάς, 0.20 km
41	Φανή	Η μεγάλη απόφραξη	Νέα Τεμπέλη	13	Ο Τρικαλινός, 0.11 km
43	Έφη	Οι Χούλιγκανς	Νέα Τεμπέλη	15	Ο Τρικαλινός, 0.55 km

Ο χάρτης με τους σινεμάδες και με τα ψητοπωλεία. Ο Πίνακας από κάτω ενημερώνεται (AJAX) καθώς κάνουμε zoom in / out, pan και απεικονίζεται μόνο ότι φαίνεται στον χάρτη. Ταξινομούνται δε σύμφωνα με την απόσταση από το κέντρο του χάρτη. Για κάθε σινεμά παρουσιάζεται ο αριθμός των ψητοπωλείων σε μία ακτίνα 2.5 km καθώς και το κοντινότερο ψητοπωλείο καθώς και η απόσταση σε km. Το αυτό και για τα ψητοπωλεία (χάρτης, λίστα με ψητοπωλεία).

Το Κ Ο Ζ Α Ν Ο Ρ Α Μ Α

Σινεμάς: Πενικυλίνη

Παίζει την ταινία: Οι Χούλιγκανς

Περιοχή: Λούτζα

Ψητοπωλεία σε ακτίνα 2.5 km από τον «Πενικυλίνη»

id	Ψητοπωλείο	Αστεράκια	Περιοχή	Απόσταση (km)
38	Το σουβλάκι της γιαγιάς	* *	Λούτζα	0.20
42	Οι ξενύχτηδες	* * *	Νέα Τεμπέλη	0.34
39	Τρίτος γύρος	* * *	Λούτζα	0.68
47	Τα τέσσερα δεύτερα ξαδέλφια	* *	Νέα Τεμπέλη	0.98
45	Ο Τρικαλινός	* * *	Νέα Τεμπέλη	1.13
43	Τρως καλά	*	Νέα Τεμπέλη	1.32
34	Τα οκτώ αδέλφια	*	Λούτζα	1.57
33	Πας γυρεύοντας	* * *	Λούτζα	1.68
35	Αναμένα κάρβουνα	* *	Λούτζα	1.75
46	Ο Καπετάν Μιχάλης	*	Νέα Τεμπέλη	1.91
36	Τα τραπεζάκια έξω	* * *	Λούτζα	1.92
37	Το περιβόλι της θείας	*	Λούτζα	1.96
40	Tour de France	*	Λούτζα	2.06
30	Η πίτα του φαλάκρα	* * *	Σουτζουκάκια	2.10
23	Το τζατζίκι του Βασίλη	* *	Πετρογκάζι	2.22
24	Το στέκι του κόκορα	* * *	Πετρογκάζι	2.26

...Πίσω στη λίστα με τους Σινεμάδες

Παρουσίαση (detail) ενός σινεμά με τα κοντινότερα (σε ακτίνα 2.5 km) ψητοπωλεία, ταξινομημένα σύμφωνα με την απόσταση από τον σινεμά. Το αυτό και για το detail των ψητοπωλείων.