GeoDjango Tutorial (foss4g-it Lugano 2010) Documentation

Release 0.1

Paolo Corti, Alessandro Pasotti, Alessandro Furieri

February 06, 2010

CONTENTS

Index of tutorial:

CHAPTER

ONE

CONFIGURAZIONI INIZIALI

1.1 Virtual Machine

L'organizzazione di foss4g-it 2010 mette a disposizione una VM per VirtualBox con tutto l'ambiente necessario a seguire il tutorial. Si riportano le caratteristiche di tale VM in maniera tale da poter replicare lo stesso ambiente e seguire in maniera ottimale il tutorial se non si avesse a disposizione la VM in questione.

Sistema operativo: Ubuntu 9.10 x86 Desktop

Segue un elenco del software installato.

Software installato dagli official sources:

- apache 2.2.12
- python-virtualenv
- subversion
- ipython 0.10
- postgres 8.4.2
- pgadmin III 1.10
- postgis 1.4.0
- psycopg 2.0.8
- cgi-mapserver 5.6.1
- python-mapscript

Software installato da ubuntugis-unstable:

- qgis 1.4
- libgdal1-1.6.0
- libgeos-3.1.1
- proj 4.7.0

Software installato dai source e compilando:

- libspatialite 2.3.1

- spatialite-tools 2.3.1
- pysqlite2 2.5.6
- geoip 1.4.4
- GeoIP-Python-1.2.2

Altre tipologie di software installato:

- Plugin Firefox: sqlite manager, webdeveloper e firebug

1.2 Configurazioni

1.2.1 Ubuntu

Usare queste credenziali per entrare nella virtual machine: user: geodjango password: geodjango

1.2.2 Postgres/PostGis

L'utente amministrativo di postgres ha queste credenziali: user: postgres password: postgres

L'utente utilizzato nel tutorial e in tutte le applicazioni django/geodjango disponibili sulla VM ha queste credenziali: user: geodjango password: geodjango

1.3 gfoss4-it GeoDjango Tutorial

Per provare immediatamente il tutorial, si puo' procedere in questo modo. Sulla VM esiste un virtualenv con django 1.2 collocato in: /home/geodjango/tutorial/django-1.2-alpha-1-env. Attivare il virtualenv (basato su django 1.2-alpha1 e python 2.6) e lanciare l'applicazione, che e' gia' configurata per girare su PostGis (il database e' gia' stato creato).

Utente e password dell'utente di amministrazione di django sono: user: admin pwd: admin

C'e' un WMS basato su mapserver in caso non ci sia connessione internet. Il mapfile si trova qui e viene chiamato da openlayer: /home/geodjango/tutorial/django-1.2-alpha-1-env/foss4git/mapserver/italia.map

1.4 GeoDjango-basic-apps

Vedere: http://code.google.com/p/geodjango-basic-apps/

Sono tre demo che mostrano l'uso di GeoDjango, tutte configurate per accedere a postgres su locahost. Tutte e tre si trovano in: /home/geodjango/tutorial/django-1.2-alpha-1-env/geodjango-basic-apps/projects In tutti i casi utente e password dell'admin di django sono: user: admin pwd: admin

In particolare:

- geographic_admin mostra l'utilizzo delle application Django Admin e Databrowse abilitate spazialmente con GeoDjango (utilizzando OpenLayers)

(todo)

Per provare immediatamente le basic-apps, si puo' procedere in questo modo. Sulla VM si trovano nel virtualenv con django 1.0 collocato in: /home/geodjango/tutorial/django-1.0-env/geodjango-basic-apps. Attivare il virtualenv (basato su django 1.0 e python 2.5) e lanciare le applicazioni. Sono gia' configurate per girare su PostGis (i database sono gia' stati creati).

C'e' un WMS basato su mapserver in caso non ci sia connessione internet. Il mapfile si trova qui e viene chiamato da openlayer: /home/geodjango/software/mapserver-stuff/cape.map

CHAPTER

TWO

UTILIZZO DI DJANGO

2.1 Creazione ed attivazione del virtualenv

Creare e attivare un virtualenv con django 1.2 alpha 1 e python 2.6 oppure utilizzare quello gia' disponibile sulla VM:

```
geodjango@geodjango-laptop:~/tutorial/django-1.2-alpha-1-env$ source bin/activate
```

2.2 Creazione del progetto Django per il tutorial

Creare il progetto Django:

```
django-admin.py startproject foss4git
cd foss4git/
```

2.3 Editing del file settings.py

a questo punto editare il file di configurazione settings.py:

```
import os

# aggiungere queste due righe che settano due variabili utilizzate poi nel seguito
ROOT_PROJECT_FOLDER = os.path.dirname(__file__)
STATIC_FILES = os.path.join(ROOT_PROJECT_FOLDER,'static')
...
DATABASES = {
    'default': {
        'ENGINE': 'postgresql_psycopg2', # Add 'postgresql_psycopg2', 'postgresql', 'mysql', 'sqlite
        'NAME': 'tutorial',                      # Or path to database file if using sqlite3.
        'USER': 'geodjango',                     # Not used with sqlite3.
        'PASSWORD': 'geodjango',                 # Not used with sqlite3.
        'HOST': 'localhost',                     # Set to empty string for localhost. Not used with
        'PORT': '',                      # Set to empty string for default. Not used with sqlite3.
    }
}
...
LANGUAGE_CODE = 'it-IT' # setta in italiano la localizzazione
...
```

```
# setta i path dei media files
MEDIA_ROOT = STATIC_FILES
MEDIA_URL = '/static/'
ADMIN_MEDIA_PREFIX = '/media/'
```

Creiamo il db. Utilizzeremo PostGis, anche se e'./man possibile seguire il tutorial, con piccole differenze, con Spatialite (sqlite3):

```
(django-1.2-alpha-1-env)geodjango@geodjango-laptop:~/tutorial/django-1.2-alpha-1-env/foss4git$ psql -
psql (8.4.2)
Type "help" for help.

template1=> CREATE DATABASE tutorial WITH TEMPLATE=template_postgis ENCODING='UTF8';
CREATE DATABASE
template1=> \q
```

Sincronizziamo il db:

```
(django-1.2-alpha-1-env)geodjango@geodjango-laptop:~/tutorial/django-1.2-alpha-1-env/foss4git$ ./mana
Creating table django_content_type
Creating table auth_permission
Creating table auth_group_permissions
Creating table auth_group
Creating table auth_user_user_permissions
Creating table auth_user_groups
Creating table auth_user
Creating table auth_message
Creating table django_site
Creating table django_session

You just installed Django's auth system, which means you don't have any superusers defined.
Would you like to create one now? (yes/no): yes
Username (Leave blank to use 'geodjango'): admin
E-mail address: pcorti@gmail.com
Password:
Password (again):
Superuser created successfully.
Installing index for auth.Permission model
Installing index for auth.Message model
```

Abbiamo gia' le tabelle base per un'applicazione django create nel database:

```
(django-1.2-alpha-1-env)geodjango@geodjango-laptop:~/tutorial/django-1.2-alpha-1-env/foss4git$ psql -
psql (8.4.2)
Type "help" for help.

                        List of relations
 Schema |               Name                |   Type   |   Owner
--------+-----------------------------------+----------+-----------
 public | auth_group                        | table    | geodjango
 public | auth_group_id_seq                 | sequence | geodjango
 public | auth_group_permissions            | table    | geodjango
 public | auth_group_permissions_id_seq     | sequence | geodjango
 public | auth_message                      | table    | geodjango
 public | auth_message_id_seq               | sequence | geodjango
 public | auth_permission                   | table    | geodjango
```

```
 public | auth_permission_id_seq            | sequence | geodjango
 public | auth_user                         | table    | geodjango
 public | auth_user_groups                  | table    | geodjango
 public | auth_user_groups_id_seq           | sequence | geodjango
 public | auth_user_id_seq                  | sequence | geodjango
 public | auth_user_user_permissions        | table    | geodjango
 public | auth_user_user_permissions_id_seq | sequence | geodjango
 public | django_content_type               | table    | geodjango
 public | django_content_type_id_seq        | sequence | geodjango
 public | django_session                    | table    | geodjango
 public | django_site                       | table    | geodjango
 public | django_site_id_seq                | sequence | geodjango
 public | geometry_columns                  | table    | postgres
 public | spatial_ref_sys                   | table    | postgres
(21 rows)
\q
```

2.4 Primo avvio del progetto

Usando il server di django:

```
(django-1.2-alpha-1-env)geodjango@geodjango-laptop:~/tutorial/django-1.2-alpha-1-env/foss4git$ ./mana
Validating models...
0 errors found

Django version 1.2 alpha 1, using settings 'foss4git.settings'
Development server is running at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

A questo punto andando su localhost:8000 dovrebbe presentarsi la schermata iniziale.

2.5 Creazione dell'applicazione del tutorial

Creeremo un'applicazione denominata fauna, con due modelli (Animale, Avvistamento):

```
(django-1.2-alpha-1-env)geodjango@geodjango-laptop:~/tutorial/django-1.2-alpha-1-env/foss4git$ ./mana
```

Notare la struttura iniziale del progetto e dell'applicazione:

```
(django-1.2-alpha-1-env)geodjango@geodjango-laptop:~/tutorial/django-1.2-alpha-1-env/foss4git$ tree
.
|-- __init__.py
|-- __init__.pyc
|-- fauna
|   |-- __init__.py
|   |-- models.py
|   |-- tests.py
|   `-- views.py
|-- manage.py
|-- settings.py
|-- settings.pyc
|-- settings.py~
```

```
`-- urls.py

1 directory, 11 files
```

2.6 Installazione delle applicazioni

Nel file settings.py installiamo l'applicazione fauna e contrib.admin:

```
...
INSTALLED_APPS = (
'django.contrib.auth',
'django.contrib.contenttypes',
'django.contrib.sessions',
'django.contrib.sites',
'django.contrib.messages',
'django.contrib.admin',
'foss4git.fauna',
)
...
```

2.7 Creazione dei modelli

Creiamo i due modelli nel file models.py:

```
from django.db import models

# modelli
class Animale(models.Model):
    """Modello per rappresentare gli animali"""
    nome = models.CharField(max_length=50)
    foto = models.ImageField(upload_to='animali.foto')

    def __unicode__(self):
        return '%s' % (self.nome)

    def image_url(self):
        print '***%s****' % self.foto.url
        return u'<img src="%s" alt="%s" width="80"></img>' % (self.foto.url, self.nome)
    image_url.short_description = "Foto"
    image_url.allow_tags = True

    class Meta:
        ordering = ['nome']
        verbose_name_plural = "Animali"

class Avvistamento(models.Model):
    """Modello spaziale per rappresentare gli avvistamenti"""

    LIVELLI_INTERESSE = (
        (1, '*'),
        (2, '**'),
        (3, '***'),
        (4, '****'),
```

```
        (5, '*****'),
    )
    data = models.DateTimeField()
    note = models.TextField()
    interesse = models.IntegerField(choices=LIVELLI_INTERESSE)
    animale = models.ForeignKey(Animale)

    def __unicode__(self):
        return '%s' % (self.data)

    class Meta:
        ordering = ['data']
        verbose_name_plural = "Avvistamenti"
```

Prima di sincronizzare il db, verifichiamo che operazioni effettuera' la sincronizzazione:

```
(django-1.2-alpha-1-env)geodjango@geodjango-laptop:~/tutorial/django-1.2-alpha-1-env/foss4git$ ./mana
BEGIN;
CREATE TABLE "fauna_animale" (
    "id" serial NOT NULL PRIMARY KEY,
    "nome" varchar(50) NOT NULL,
    "foto" varchar(100) NOT NULL
)
;
CREATE TABLE "fauna_avvistamento" (
    "id" serial NOT NULL PRIMARY KEY,
    "data" timestamp with time zone NOT NULL,
    "note" text NOT NULL,
    "interesse" integer NOT NULL,
    "animale_id" integer NOT NULL REFERENCES "fauna_animale" ("id") DEFERRABLE INITIALLY DEFERRED
)
;
COMMIT;
```

Sincronizziamo il database:

```
(django-1.2-alpha-1-env)geodjango@geodjango-laptop:~/tutorial/django-1.2-alpha-1-env/foss4git$ ./mana
Creating table django_admin_log
Creating table fauna_animale
Creating table fauna_avvistamento
Installing index for admin.LogEntry model
Installing index for fauna.Avvistamento model
```

2.8 Utilizzo dell'applicazione contrib.admin

contrib.admin e' stata gia' installata, a questo punto e' sufficiente configuare la prima url nel file urls. Nello stesso file va anche impostata la url dei file statici:

```
...
from settings import STATIC_FILES
from django.contrib import admin
admin.autodiscover()
...
```

```
    # Uncomment the next line to enable the admin:
    (r'^admin/', include(admin.site.urls)),
    # static files
    (r'^static/(?P<path>.*)$', 'django.views.static.serve', {'document_root': STATIC_FILES, 'show_inc
)
```

Lanciando il server e andando sull'admin, pero' i due modelli non sono presenti. Creiamo un file fauna/admin.py cosi' costituito:

```
from django.contrib import admin
from models import *

class AvvistamentoAdmin(admin.ModelAdmin):

    model = Avvistamento

    list_display = ['data', 'animale', 'interesse']
    list_filter = ['data', 'animale', 'interesse']
    date_hierarchy = 'data'
    fieldsets = (
      ('Location Attributes', {'fields': (('data', 'animale', 'note', 'interesse'))}),
    )

class AnimaleAdmin(admin.ModelAdmin):

    model = Animale
    list_display = ['nome', 'image_url',]

# registriamo per l'admin
admin.site.register(Animale, AnimaleAdmin)
admin.site.register(Avvistamento, AvvistamentoAdmin)
```

A questo punto e' possibile creare delle istanze dei modelli con l'admin. Inserire qualche record, ovviamente il database si aggiorna di conseguenza:

```
geodjango@geodjango-laptop:~/tutorial/django-1.2-alpha-1-env/foss4git$ psql -U geodjango tutorial
psql (8.4.2)
Type "help" for help.

tutorial=> select * from fauna_animale;
 id | nome  |          foto
----+-------+------------------------
  1 | Lupo  | animali.foto/lupo.jpg
  2 | Volpe | animali.foto/volpe.jpg
(2 rows)

tutorial=> select * from fauna_avvistamento;
 id |          data          |  note   | interesse | animale_id
----+------------------------+---------+-----------+------------
  1 | 2010-02-05 01:38:08+01 | note... |         3 |          1
(1 row)
```

2.9 Uso della shell di Django

Usiamo la shell per dimostrare i metodi CRUD:

```
>>> from fauna.models import *
>>> animali = Animale.objects.all()
>>> animali
[<Animale: Lupo>, <Animale: Volpe>]
>>> lupo.nome
u'Lupo'
>>> lupo.foto
<ImageFieldFile: animali.foto/lupo.jpg>
>>> a1 = Avvistamento.objects.get(id=1)
>>> a1
<Avvistamento: 2010-02-04 18:38:08>
>>> a1.animale = lupo
>>> a1.interesse = 5
>>> a1.save()
>>> for a in avvistamenti:
   ....:     print a.animale
   ....:     print a.interesse
   ....:     print a.data
Lupo
3
2010-02-04 18:38:08
Lupo
3
2010-02-04 18:46:23
Volpe
1
2010-02-04 18:46:37
>>> nuovo_avvistamento = Avvistamento(data=datetime.now(), animale=lupo, interesse=5)
>>> nuovo_avvistamento.save()
>>> avvistamenti.count()
3
>>> avv_interessanti = Avvistamento.objects.filter(interesse__gte=4)
>>> avv_interessanti.count()
2
```

2.10 Creazione di una vista e di un template

Proviamo a creare una vista ed un template che presentino, nel lato pubblico dell'applicativo, un elenco degli avvistamenti inseriti nel template.

Come prima cosa inseriamo la nuova url nel file urls.py, ricordandosi di importare fauna.views (dove inseriremo la nuova vista):

```
from django.conf.urls.defaults import *
from settings import STATIC_FILES
from fauna.views import *
...
    # Uncomment the next line to enable the admin:
    (r'^admin/', include(admin.site.urls)),
    # indirizzi non soggetti ad autenticazione
```

```
    (r'^avvistamenti/', avvistamenti),
...
```

Poi creaimo la vista nel file views.py:

```
from django.shortcuts import render_to_response, get_object_or_404
from django.contrib.gis.shortcuts import render_to_kml
from fauna.models import *

# vista per visualizzare tutti i punti di avvistamento
def avvistamenti(request):
    avvistamenti  = Avvistamento.objects.all()
    return render_to_response("avvistamenti.html", {'avvistamenti' : avvistamenti})
```

Infine creiamo la directory fauna/templates ed al suo interno inseriamo il template avvistamenti.html:

```
<html>
  <head></head>
  <body>
    <h3>Avvistamenti in Italia</h3>
    <ul>
    {% for avv in avvistamenti %}
        <li>{{avv.animale.image_url|safe}} {{avv.data|date:"d M Y"}}, Interesse: {{avv.interesse}}</l
    {% endfor %}
    </ul>
  </body>
</html>
```

Si può a questo punto raggiungere la pagina http://localhost:8000/avvistamenti che dovrebbe visualizzare l'elenco degli avvistamenti.

CHAPTER

THREE

UTILIZZO DI GEODJANGO

3.1 Importazione di GeoDjango

Aggiungere l'applicazione contrib.gis nel file settings.py del progetto:

```
...
INSTALLED_APPS = (
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.sites',
    'django.contrib.messages',
    'django.contrib.admin',
    'django.contrib.gis',
    'foss4git.fauna',
)
...
```

3.2 Aggiunta del campo geometrico nel modello

Vogliamo aggiungere un campo geometrico puntuale nel modello Avvistamento. Per fare questa cosa bisogna far ereditare il modello a django.contrib.gis.db anzichè a django.db.models. Una voltà fatto ciò sarà possibile usare il campo geometrico. Ricordarsi di abilitare models.GeoManager al posto del manager di default (models.objects) per avere a disposizione tutte le funzionalità di geodjango sul modello.

Effettuare le seguenti modifiche su models.py:

```
from django.db import models
from django.contrib.gis.db import models as gismodels

# modelli
class Animale(models.Model):
    """Modello per rappresentare gli animali"""
    nome = models.CharField(max_length=50)
...
class Avvistamento(gismodels.Model):
    """Modello spaziale per rappresentare gli avvistamenti"""

    LIVELLI_INTERESSE = (
        (1, '*'),
        (2, '**'),
```

```
        (3, '***'),
        (4, '****'),
        (5, '*****'),
    )
    data = gismodels.DateTimeField()
    note = gismodels.TextField()
    interesse = gismodels.IntegerField(choices=LIVELLI_INTERESSE)
    animale = gismodels.ForeignKey(Animale)
    geometry = gismodels.PointField(srid=4326)
    objects = gismodels.GeoManager()
...
```

Prima di effettuare la sincronizzazione è necessario cancellare la tabella fauna_avvistamento:

```
(django-1.2-alpha-1-env)geodjango@geodjango-laptop:~/tutorial/django-1.2-alpha-1-env/foss4git$ psql -
psql (8.4.2)
Type "help" for help.

tutorial=> drop table fauna_avvistamento;
DROP TABLE
```

A questo punto sincronizzare il database:

```
(django-1.2-alpha-1-env)geodjango@geodjango-laptop:~/tutorial/django-1.2-alpha-1-env/foss4git$ ./mana
Creating table fauna_avvistamento
Installing index for fauna.Avvistamento model
```

3.3 Gestire dati geometrici nell'admin

Dobbiamo abilitare l'applicazione admin a gestire il data entry del dato geometrico. Useremo la funzionalità di GeoDjango che abilita l'editing dei dati geografici utilizzando OpenLayers su cartografia OpenStreetMap.

Per far questo modifichiamo il file admin.py:

```
from django.contrib import admin
from django.contrib.gis.admin import GeoModelAdmin
from models import *

class AvvistamentoAdmin(GeoModelAdmin):

    model = Avvistamento

    list_display = ['data', 'animale', 'interesse']
    list_filter = ['data', 'animale', 'interesse']
    date_hierarchy = 'data'
    fieldsets = (
      ('Caratteristiche avvistamento', {'fields': (('data', 'animale', 'note', 'interesse'))}),
      ('Mappa', {'fields': ('geometry',)}),
    )

    # Openlayers settings
    scrollable = False
    map_width = 500
    map_height = 500
    #openlayers_url = '/static/openlayers/lib/OpenLayers.js'
```

```
    default_zoom = 6
    default_lon = 13
    default_lat = 42

class AnimaleAdmin(admin.ModelAdmin):
...
```

3.4 Importazione di shapefile

A questo punto creiamo due geomodelli per la gestione delle regioni e delle province.

I dati relativi a regioni e province sono nei due shapefile omonimi sotto la directory data/shapefiles e sono georiferiti nel sistema geografico WGS 1984 (srid=4326).

Possiamo analizzare i due shapefiles con l'utility orginfo presente nel pacchetto GDAL/OGR (usando l'opzione so, summary only):

```
(django-1.2-alpha-1-env)geodjango@geodjango-laptop:~/tutorial/django-1.2-alpha-1-env/foss4git$ ogrini
INFO: Open of `data/shapefile/regioni.shp'
      using driver `ESRI Shapefile' successful.

Layer name: regioni
Geometry: Polygon
Feature Count: 20
Extent: (6.627586, 35.493472) - (18.521529, 47.093684)
Layer SRS WKT:
GEOGCS["WGS 84",
    DATUM["WGS_1984",
        SPHEROID["WGS 84",6378137,298.257223563,
            AUTHORITY["EPSG","7030"]],
        TOWGS84[0,0,0,0,0,0,0],
        AUTHORITY["EPSG","6326"]],
    PRIMEM["Greenwich",0,
        AUTHORITY["EPSG","8901"]],
    UNIT["degree",0.0174532925199433,
        AUTHORITY["EPSG","9108"]],
    AUTHORITY["EPSG","4326"]]
gid: Integer (10.0)
objectid: Integer (10.0)
regione: String (255.0)
cod_rip1: Integer (10.0)
cod_rip2: Integer (10.0)
cod_reg: Integer (10.0)
shape_area: Real (24.15)
shape_len: Real (24.15)
boundingbo: String (255.0)

(django-1.2-alpha-1-env)geodjango@geodjango-laptop:~/tutorial/django-1.2-alpha-1-env/foss4git$ ogrini
INFO: Open of `data/shapefile/province.shp'
  using driver `ESRI Shapefile' successful.

Layer name: province
Geometry: Polygon
Feature Count: 107
Extent: (6.627586, 35.493472) - (18.521529, 47.093684)
Layer SRS WKT:
```

```
GEOGCS["WGS 84",
    DATUM["WGS_1984",
        SPHEROID["WGS 84",6378137,298.257223563,
            AUTHORITY["EPSG","7030"]],
        TOWGS84[0,0,0,0,0,0,0],
        AUTHORITY["EPSG","6326"]],
    PRIMEM["Greenwich",0,
        AUTHORITY["EPSG","8901"]],
    UNIT["degree",0.0174532925199433,
        AUTHORITY["EPSG","9108"]],
    AUTHORITY["EPSG","4326"]]
gid: Integer (10.0)
objectid: Integer (10.0)
cod_prov: Integer (10.0)
cod_ipi: Integer (10.0)
provincia: String (255.0)
sigla: String (255.0)
cod_reg: Integer (10.0)
shape_area: Real (24.15)
shape_len: Real (24.15)
boundingbo: String (255.0)
```

In entrambi i casi importeremo i campi relativi a nome, codice (regione e cod_reg per regioni, provincia e cod_prov per province) e geometria.

Come prima cosa aggiungiamo i due geomodelli che gestiscono le entità Regione e Provincia:

```
class Regione(gismodels.Model):
    """Modello spaziale per rappresentare le regioni"""
    codice = gismodels.IntegerField()
    nome = gismodels.CharField(max_length=50)
    geometry = gismodels.MultiPolygonField(srid=4326)
    objects = gismodels.GeoManager()

    def __unicode__(self):
        return '%s' % (self.nome)

class Provincia(gismodels.Model):
    """Modello spaziale per rappresentare le regioni"""
    codice = gismodels.IntegerField()
    nome = gismodels.CharField(max_length=50)
    geometry = gismodels.MultiPolygonField(srid=4326)
    objects = gismodels.GeoManager()

    def __unicode__(self):
        return '%s' % (self.nome)
```

Prima di effettuare la sincronizzazione del database, verifichiamo gli oggetti che verranno creati nel database:

```
(django-1.2-alpha-1-env)geodjango@geodjango-laptop:~/tutorial/django-1.2-alpha-1-env/foss4git$ ./mana
BEGIN;
CREATE TABLE "fauna_animale" (
    "id" serial NOT NULL PRIMARY KEY,
    "nome" varchar(50) NOT NULL,
    "foto" varchar(100) NOT NULL
)
;
```

```
CREATE TABLE "fauna_avvistamento" (
    "id" serial NOT NULL PRIMARY KEY,
    "data" timestamp with time zone NOT NULL,
    "note" text NOT NULL,
    "interesse" integer NOT NULL,
    "animale_id" integer NOT NULL REFERENCES "fauna_animale" ("id") DEFERRABLE INITIALLY DEFERRED
)
;
CREATE TABLE "fauna_regione" (
    "id" serial NOT NULL PRIMARY KEY,
    "codice" integer NOT NULL,
    "nome" varchar(50) NOT NULL
)
;
CREATE TABLE "fauna_provincia" (
    "id" serial NOT NULL PRIMARY KEY,
    "codice" integer NOT NULL,
    "nome" varchar(50) NOT NULL
)
;
COMMIT;
```

A questo punto sincronizziamo il database:

```
(django-1.2-alpha-1-env)geodjango@geodjango-laptop:~/tutorial/django-1.2-alpha-1-env/foss4git$ ./mana
Creating table fauna_regione
Creating table fauna_provincia
Installing index for fauna.Regione model
Installing index for fauna.Provincia model
```

Creiamo uno script carica_dati.py per importare i due shapefile nel database spaziale usando l'utility di GeoDjango LayerMapping:

```
"""Utility per inserire regioni e province sul database dagli shapefile allegati"""

import os
os.environ['DJANGO_SETTINGS_MODULE'] = 'settings'

from django.contrib.gis.utils import mapping, LayerMapping
from fauna.models import Regione, Provincia

print 'carico regioni...'

regioni_mapping = {
    'codice' : 'cod_reg',
    'nome' : 'regione',
    'geometry' : 'MULTIPOLYGON',
}

regioni_shp = 'data/shapefile/regioni.shp'
regioni =  LayerMapping(Regione, regioni_shp, regioni_mapping, transform=False, encoding='iso-8859-1'
regioni.save(verbose=True, progress=True)

print 'carico province...'

province_mapping = {
    'codice' : 'cod_prov',
```

```
    'nome' : 'provincia',
    'geometry' : 'MULTIPOLYGON',
}

province_shp = 'data/shapefile/province.shp'
province =  LayerMapping(Provincia, province_shp, province_mapping, transform=False, encoding='iso-88
province.save(verbose=True, progress=True)
```

Lanciamo lo script per effettuare il caricamento:

```
(django-1.2-alpha-1-env)geodjango@geodjango-laptop:~/tutorial/django-1.2-alpha-1-env/foss4git$ pythor
carico regioni...
Saved: PIEMONTE
Saved: VALLE D'AOSTA
...
Saved: LOMBARDIA
Saved: LAZIO
carico province...
Saved: TORINO
Saved: VERCELLI
...
Saved: MEDIO CAMPIDANO
Saved: CARBONIA-IGLESIAS
```

3.5 Creazione di template con OpenLayer e GeoDjango

Creeremo due template che mostrano l'utilizzo di OpenLayers e GeoDjango:

```
* un template denominato italia.html, generato dalla view italia che visualizza tutti i punti di avvi
* un template denominato regione.html, generato dalla view regione che visualizza tutti i punti di av
```

Se avanza tempo provare a creare un template che visualizzi i punti per provincia ed una vista che visualizzi i punti per tipologia di animale avvistato.

Inoltre creeremo una view denominata all_kml che alimenta il template di geodjango gis/kml/placemarks.kml. Tale view fornisce un elenco completo in formato kml delle geometrie inserite nel sistema che servirà ad alimentare il layer vettoriale di OpenLayers per la visualizzazione degli avvistamenti sulla mappa.

Come prima cosa dichiariamo queste tre view sul file urls.py:

```
(r'^admin/', include(admin.site.urls)),
# indirizzi non soggetti ad autenticazione
(r'^avvistamenti/', avvistamenti),
(r'^kml/', all_kml),
(r'^$', italia),
(r'^regione/(?P<id>[0-9]*)/', regione),
# TODO (esercizio aggiuntivo, zoom su provincia): (r'^provincia/(?P<id>[0-9]*)/', provincia),
# TODO (esercizio aggiuntivo, filtrato su animale): (r'^animale/(?P<id>[0-9]*)/', animale),
# static files
(r'^static/(?P<path>.*)$', 'django.views.static.serve', {'document_root': STATIC_FILES, 'show_indexes

)
```

A questo punto creiamo le tre view necessarie sul file views.py:

```
from django.shortcuts import render_to_response, get_object_or_404
from django.contrib.gis.shortcuts import render_to_kml
from fauna.models import *

def avvistamenti(request):
    ...

def all_kml(request):
    """vista per generare il kml di tutti i punti di avvistamento"""
    avvistamenti  = Avvistamento.objects.kml()
    return render_to_kml("gis/kml/placemarks.kml", {'places' : avvistamenti})

def italia(request):
    """vista con zoom su italia e il numero dei punti di avvistamento inseriti nel sistema"""
    num_avvistamenti = Avvistamento.objects.all().count()
    return render_to_response('italia.html', {'num_avvistamenti' : num_avvistamenti})

def regione(request, id):
    """vista con zoom su regione e l'elenco dei punti di avvistamento inseriti nel sistema per la reg
    regione = get_object_or_404(Regione, codice=id)
    avvistamenti = Avvistamento.objects.filter(geometry__intersects=regione.geometry)
    return render_to_response("regione.html", { 'regione': regione, 'avvistamenti': avvistamenti })
```

Infine creiamo i due template necessari: italia.html e regione.html.

Notare che, nel caso non ci fosse una connessione internet, abbiamo creato un WMS con mapserver da utilizzare come base layer al posto del basic layer del WMS di Metacarta.

italia.html:

```
<html>
  <head>
    <script src="/static/openlayers/lib/OpenLayers.js"></script>
    <style type="text/css"> #map { width:500px; height: 500px; } </style>
    <script type="text/javascript">
        var map, base_layer, kml;
        var ms_url = "http://localhost/cgi-bin/mapserv?map=/home/geodjango/tutorial/django-1.2-alpha-
        function init(){
            map = new OpenLayers.Map('map');
            base_layer = new OpenLayers.Layer.WMS( "OpenLayers WMS",
               "http://labs.metacarta.com/wms/vmap0", {layers: 'basic'} );
            var regioni = new OpenLayers.Layer.WMS("Regioni",
               ms_url, {layers : 'regioni'} );
            var province = new OpenLayers.Layer.WMS("Province",
               ms_url, {layers : 'province'} );

            kml = new OpenLayers.Layer.GML("KML", "/kml",
               { format: OpenLayers.Format.KML });
            map.addLayers([base_layer, regioni, province, kml]);

            map.addControl(new OpenLayers.Control.LayerSwitcher());
            map.setCenter(new OpenLayers.LonLat(13,42),6);
            }
    </script>
  </head>
  <body onload="init()">
    <h3>Avvistamenti in Italia</h3>
    <div id="map"></div>
```

```
    <p>Sono stati inseriti {{num_avvistamenti}} avvistamenti.</p>
  </body>
</html>
```

regione.html:

```
<html>
  <head>
    <script src="http://openlayers.org/api/OpenLayers.js"></script>
    <style type="text/css"> #map { width:400px; height: 400px; } </style>
    <script type="text/javascript">
        var map, base_layer, kml;
        var ms_url = "http://localhost/cgi-bin/mapserv?map=/home/geodjango/tutorial/django-1.2-alpha-
        function init(){
            map = new OpenLayers.Map('map');
            base_layer = new OpenLayers.Layer.WMS( "OpenLayers WMS",
               "http://labs.metacarta.com/wms/vmap0", {layers: 'basic'} );
            var regioni = new OpenLayers.Layer.WMS("Regioni",
               ms_url, {layers : 'regioni'} );
            var province = new OpenLayers.Layer.WMS("Province",
               ms_url, {layers : 'province'} );

            var format = new OpenLayers.Format.GeoJSON()
            regione = format.read({{ regione.geometry.geojson|safe}})[0];
            // We mark it 'safe' so that Django doesn't escape the quotes.

            regione.attributes = { 'nome': "{{regione.nome}}", 'type': 'regione'};
            vectors = new OpenLayers.Layer.Vector("Data");
            vectors.addFeatures(regione);
            for (var i = 0; i < points.length; i++) {
                point = format.read(points[i])[0];
                point.attributes = {'type':'point'};
                vectors.addFeatures(point);
            }
            map.addLayers([base_layer, regioni, province, vectors]);

            map.addControl(new OpenLayers.Control.LayerSwitcher());
            map.zoomToExtent(regione.geometry.getBounds());
}
    </script>
  </head>
  <body onload="init()">
    <h3>Avvistamenti nella regione: {{ regione.nome}}</h3>
    <div id="map"></div>
    In questa regione ci sono stati {{avvistamenti.count}} avvistamenti.<br>
    <script> var points = []; </script>
    <ul>
    {% for avvistamento in avvistamenti %}
        <li>{{ avvistamento.data }}, {{ avvistamento.animale.nome }}</li>
        <script>points.push({{avvistamento.geometry.geojson|safe}});</script>
    {% endfor %}
    </ul>
  </body>
</html>
```

A questo punto eventualmente rilanciare il server di django e verificare che le viste in questione presentino i risultati previsti.

CHAPTER

FOUR

UTILIZZO DELL'API DI GEODJANGO

4.1 Creazione di un modello di test

Aggiungere il seguente modello, che useremo come sandbox per fare un po di test con l'API di GeoDjango:

```
class SandboxLayer(gismodels.Model):
    """Modello spaziale per effettuare test con l'API GeoDjango"""
    nome = gismodels.CharField(max_length=50)
    geometry = gismodels.GeometryField(srid=3395) # WGS84 mercatore
    objects = gismodels.GeoManager()

    def __unicode__(self):
        return '%s' % (self.nome)
```

Sincronizzare il database:

```
(django-1.2-alpha-1-env)geodjango@geodjango-laptop:~/tutorial/django-1.2-alpha-1-env/foss4git$ ./mana
Creating table fauna_sandboxlayer
Installing index for fauna.SandboxLayer model
```

4.2 Utilizzo dell'API di GeoDjango

Si effettuino alcune prove utilizzando l'API di GeoDjango e verificando i risultati ad es con QGis e il layer SandboxLayer caricato.

Copiamo una regione su sandboxlayer, avendo cura di trasformare il sistema di riferimento della geometria (per accedere alla shell con tutti i path impostati correttamente usare ./manage shell):

```
>>> from fauna.models import *
>>> lazio = Regione.objects.get(codice=12)
>>> geom4326 = lazio.geometry
>>> geom3395 = geom4326.transform(3395, clone=True)
>>> geom3395.srid
3395
>>> geom4326.srid
4326
>>> geom3395.envelope.wkt
'POLYGON ((1274660.3642747686244547 4952975.3447027457877994, 1561642.0010516194161028 4952975.344702
>>> geom4326.envelope.wkt
'POLYGON ((11.4504688728938255 40.7864493978667753, 14.0284687786766451 40.7864493978667753, 14.0284
```

```
>>> sand_lazio = SandboxLayer(geometry=geom4326,nome='lazio')
>>> sand_lazio.save()
```

Analogamente copiamo nel sandboxlayer tutti gli avvistamenti:

```
>>> points = Avvistamento.objects.all()
>>>: for p in points:
   ....:        sand_point = SandboxLayer(geometry=p.geometry.transform(3395,clone=True), nome=p.inte
   ....:        sand_point.save()
   ....:
```

Creaimo un buffer utilizzando undo dei punti appena creati sul sandboxlayer:

```
>>> p = SandboxLayer.objects.filter(geometry__intersects=sand_lazio.geometry)
>>> p
[<SandboxLayer: lazio>, <SandboxLayer: 5>]
>>> avv = p[1]
>>>buffer = SandboxLayer(geometry=avv.geometry.buffer(20000),nome='buffer')
>>>buffer.save()
```

Utilizziamo un operatore spaziale:

```
>>>sand_features = SandboxLayer.objects.all()
>>> sand_features
[<SandboxLayer: lazio>, <SandboxLayer: 3>, <SandboxLayer: 5>, <SandboxLayer: buffer>]
>>> sand_features[0].geometry.contains(sand_features[3].geometry)
True
```

CHAPTER

FIVE

INDICES AND TABLES

- *Index*
- *Module Index*
- *Search Page*