

Django, egy példán keresztül, I

Horék 'RePa' Gyuri, 2010

Az alapok

Ahogy korábban már írtam, rögzítettem tervezem valami tutorial szerűs, megírtam, hirtelen eljutottam ide - kicsit lassabban, mint terveztem, az utóbbi időszak nem volt eszem, nyei miatt. A teljes tutorial több cikket fog magába foglalni, ez az első, és a django kórnízet kialakításáról fog szólni.

A folyamatot egy olyan példán keresztül szeretném bemutatni, amit később saját címlapon használni is fogok, néhány ponton ez, lehet, hogy nem a legegyszerűbb megoldásokat alkalmazom. A problémát még irányból közelítem meg, mint a [hivatalos django tutorial](#), ez, azt sem értettem, és, illetve csak ajánlani tudom a [django dokumentációját](#), ami szerintem kifejezetten jó.

A kórnízet kialakítása

Mielőtt írtam, nekem a django kórnízet kialakítására, mindenkinek ajánlom figyelmbe, Gabor barátom nemrégiben publikált [Fejlesztői kórnízet kialakítása Python alapú fejlesztéshez című írását](#), és, nagyon hasonlóan szoktam eljárni minden új projektcske esetében. Ebben az esetben is első lépésként, létrehoztam egy `django` nevű kórnízetet, majd ide installáltam a szükséges dolgokat:

```
$ mkvirtualenv dj keptar
$ workon dj keptar
$ pip install django pysqlite
```

Az egyszerűsége kedvéért, az `sqlite` adatbázist használtam, kisebb dolgokhoz, és fejlesztéshez, tényleg, nem kell szenvedni *rendes* adatbázis szerverrel. Ezután a `django-admin.py` parancs segítségével hozzuk létre a django projektünket. Én szeretem minden kórnízetet verziókezelő rendszerben tartani, így rendszeres commitolás mellett nem gond visszatérni valami korábbi állapotra, könnyen meg tudom osztani másokkal, és a modern DVCS-eknek hála több helyre is elküldhetem, hogy biztonságban tudjam. Én magam a [Mercurial](#)-t preferélok (python ügyeből), de ugyanolyan jó tapasztalataim vannak a git-tel is, ráadásul aki az egyiket megismeri, az a másikkal is el fog boldogulni.

```
$ django-admin.py startproject dj keptar
$ cd dj keptar
$ hg init
$ hg add
$ hg commit -m 'project létrehozása'
```

A django egyik alapja a beilleszthető/újrafelhasználható alkalmazások (*pluggable/reusable apps*) rendszere, ami a DRY (*don't repeat yourself*) filozófiát követi, azaz ha már egyszer megcsináltam valamit (esetleg valaki más csinálta meg), akkor felesleges újra megcsinálni, használjuk azt, ami már van, és, valaki már valahol, legutóbb azokat a kórnízeteket, amit nekünk kell, ha újat csinálunk. Az interneten kb. végtelen ilyen django appot találunk, vannak köztük egyszerecskék, tehát mielőtt valami újba belekezdünk, mindenképp nézzünk körül, és, ha szémunkra tényleg, letes megoldás nem is született még, egy forkolásra meg, retti verzió nagy valószínűséggel találunk. (Például saját blogot mindenki ír/írni fog django-ban. Tényleg, letes meg, nem írtam.)

Mivel minden új, nyelvi munkát ezek az appok csinálnak, ezért bármit szeretnék csinálni, közzétennem kell hozzá egy appot:

```
$ python manage.py startapp keptar
```

A parancs semmi mászt nem csinél, csak l, trehoz egy keptar nev, ktnyvtérat - ami egy python modul lesz egy, bk, nt -, s a ktnyvtérban pér kvézi-tes python file:

```
keptar/
  __init__.py
  models.py
  tests.py
  views.py
```

A file-ok nevei el, g besz, desek, a models.py fogja tartalmazni az alkalmazsunk modell r, sz, t, a views.py a view-kat, a tests.py meg a teszteket. Az elnevez, sekb...l mér lészik, hogy django egy [MVC-szer,](#) framework, de mivel (szerintem) a webes vilégra a [hagyományos MVC minta](#) jelenleg csak kicsit er...ltetve illeszthet... ré, ez, rt ...k ki is mondjék, hogy csak valami hasonlft csinélnek.

Beállítások

A következ... l, p, s a django projektünk konfigurélés, melyet a settings.py file-on keresztül lehet megtenni. (Illetve a t, méban az oldalon is széletett mér [n, hény bejegyz, s.](#))

Itt minden, ppen éltsuk be az adatbázishoz kapcsoldf param, tereket, illetve adjuk hozzá a k, szél... alkalmazsunkat, s az admin alkalmazést az INSTALLED_APPS listához (csak a l, nyeg, ami nem maradt alap, rtelmezetten):

```
import os.path
PROJECT_DIR = os.path.dirname(__file__)
DATABASES = { 'default': {
    'ENGINE': 'django.db.backends.sqlite3',
    'NAME': os.path.join(PROJECT_DIR, 'keptar.sqlite'),
}}
MEDIA_ROOT = os.path.join(PROJECT_DIR, 'media')
TEMPLATE_DIRS = (
    os.path.join(PROJECT_DIR, 'templates'),
)
INSTALLED_APPS = (
    # néhány default, azokat ne bantsuk
    'django.contrib.admin',
    'keptar',
)
```

Amint lérthatf egy apro tríkkít alkalmaztam a direkt el, r, si ^ttal megadandf dolog hely, nek megéllep...téséhez, m, gpedig hogy a file elej, n megéllep...tom az ... hely, t a filerendszeren, , s ehhez k, pest relatív mfdon adom meg a títbbit (pl. MEDIA_ROOT vagy TEMPLATES_DIR).

Szinte k, sz is vagyunk, az urls.py file-ban (errél bérvebben majd k...sébb) enged, lyezzék az admin el, r, s, t:

```
from django.conf.urls.defaults import *
from django.contrib import admin
admin.autodiscover()

urlpatterns = patterns('',
```

```
) url(r'^admin/', include(admin.site.urls)),
```

Szinkronizáljuk az adatbázist az appjaink modelljeivel:

```
$ python manage.py syncdb
```

Erre az, mert van szükség, mert - bár mi magunk még nem kaptunk semmi olyat, aminek adatbázisban a helye - az admin felülethez, illetve a felhasználók kezeléséhez alapból tartoznak modellek. Első futtatáskor még is kiderül az első *admin* felhasználó adataira. Később, ha új appot adunk a rendszerhez, vagy változik a modellünk(*), akkor a syncdb management parancs újabb futtatása szinkronizálja a változásokat.

(*): Ez az, mert sajnos nem ennyire egyszerű, ha egy már beszinkronizált modellünk sémája változik, azt az alap Django nem tudja kezelni. Azonban erre is van megoldás, mégpedig a [south](#), amit én először leírtam a projektben el is helyeztem, de most nem szeretném újraírni, mert kaptam cikket, rővidem.

Ha minden jól ment, akkor a kórnyezet látrehozásával kész is vagyunk, a fejlesztői szerver futtatva ellenőrizhetjük, hogy minden rendben működik-e:

```
$ python manage.py runserver
Validating models...
0 errors found

Django version 1.2.3, using settings 'djkeptar.settings'
Development server is running at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

A fejlesztői szerver más porton (illetve publikus IP-címen) is elindíthatjuk, simén paraméterként [[ip/host:]port] módon megadva, pl:

```
$ python manage.py runserver djkeptar.hu:5000
```

Kész is vagyunk a kórnyezet látrehozásával, sőt, a böngészőnkbe beírva a `http://localhost:8000/` URL-t egy szép 404-es hibaüzenet fogad minket, mellyel akkor találkozhatunk, ha a settings.py-ben a `DEBUG` változó értéke `True`. (Ez a rendszeren ez szigorúan tilos!)

A hibaüzenetben látszik, hogy bár az `/` címen nincs semmi, de a `/admin/` címen van valami. Ezt megnevezve a Django-t *"ingyen"* kapott admin felülettel találkozhatunk, ahol jelen pillanatban felhasználókat tudunk csak kb. látrehozni. Érdemes ismerkedni vele, nagyon hasznos dolog, fejlesztés kezdeti szakaszában tényleg hasznélható, sőt van annyira flexibilis, hogy az esetek nagy részében sikerül a megrendelt követelményeknek megfelelően testre szabni, s így megspórolhatjuk egy teljesen új admin felület látrehozását.

Látrehoztunk tehát egy Django projektet, ami már képes a futásra, használ adatbázist, felhasználókat kezel, de egyelőre semmire nem jó :)

Lehet, hogy még egy management parancsokkal megismerkedve is sok olyan lépés van, amit minden egyes új projektünkön végrehajtani - bár valójában leginkább nem kell, mert naponta többször is, ezt mégis fel lehet picit gyorsítani, pl. ha egy közepesen felkonfigurált Django projektet elterelünk kedvenc verziókezelőnkben, majd azt vesszük alapul a következőknél. (pl. íme [Gébor saját django-boilerplate-je](#))

View-k és template-ek

Djangóban a view-k felelnek meg nagyjából az MVC minta controllereinek. Tipikusan olyan függvények - vagy függvények, nem viselkednek... objektumok -, amelyekhez hozzá van rendelve valamilyen URL-minta, és ha a felhasználó a böngészőbe az adott mintának megfelelő URL-t ír be, akkor a view lefut, az általa visszaadott válasz (általában valami HttpResponse objektum) pedig a megfelelő formában visszajut a böngészőbe, és ott megjelenik a kívánt tartalom.

URL kezelés

Természetesen, hogy milyen URL-minta esetén milyen view fusson le, azt nekünk kell megadnunk, amit nagyon egyszerűen a projekt könyvtárban található `urls.py` fájlban tehetünk meg. Az itt található `urlpatterns` listát kell bővítenünk url (regularis kifejezés, view függvény, egyeb opciók, és paraméterek, például a view-nak, tárandó argumentumok) bejegyzésekkel. (Az url függvény meghívása helyett használhatunk sima python felsorolásokat (tuple), ez esetben a django maga hív meg velük az url függvényt. %n szeretem kiadni.)

A reguláris kifejezés lesz a minta, amire illeszkednie kell az URL-nek, egyébként egy hagyományos python regexp (`r'minta'`), ami tartalmaz nevesített illesztéseket (pl. `(?P<postId>\d+)`), akkor azokat a view függvényünk paraméterként megkapja.

Az egyszerűség kedvéért a view függvény konkrét megadása helyett megadhatjuk csak a pontokkal elválasztott elemeket, mint stringet (*dotted path*), sőt, ha valamelyik appunk rendelkezik saját `urls.py`-vel, arra itt hivatkozhatunk az `include(appneve.urls)` direktívával.

Léssunk egy példát az egyszerű (urls.py):

```
from django.conf.urls.defaults import *

from django.contrib import admin
admin.autodiscover()

urlpatterns = patterns('',
    url(r'^/?$', 'app.views.index'),
    url(r'^page/(?P<page_id>\d+)/(?P<slug>[\w-]+)/$', 'app.views.page'),
    url(r'^about/$', 'app.views.page', {'page_id': 1, 'slug': 'about'}),
    url(r'^admin/', include(admin.site.urls)),
)
```

A fenti példában a főoldal leírása, sekor az index név, view fut le, a `/page/2/valami/` meglátogatásakor a page név, view hívódik meg `page_id=2`, és `slug='valami'` argumentumokkal, a `/about/` hatására szintén a page fut le, de az előző megadott paraméterekkel, még ha az URL `/admin/-`nal kezdődik, akkor az `admin.site.urls` modul szerint folytatódik a view-feloldás.

Vannak, akik nem ilyen pontosított módon szeretik kezelni az url-szabályokat, hanem valahogy a view közelében szeretnék a mintát a view-hoz hozzárendelni. Kis trükkkel erre is van lehetőség, például itt található [egy dekorátoros megoldás](#).

A view-k

Ahogy fentebb írtam, a view-k djangóban egyszer, függvény, nyek. Első bemenő paraméterként kitételezően egy HttpRequest objektum - ezen keresztül jutnak hozzá a GET, POST, session, és egyéb hasonló dolgokhoz -, illetve egy HttpResponse objektumot adnak vissza futás után. Nézzünk erre egy egyszerű példát:

```

from django.http import HttpResponse

def index(request):
    return HttpResponse('Hello, world!')

```

Ez egy szuper egyszerű, viszont igen rondén néz ki, ha komplett HTML oldalakat szeretnénk megírni, ezt a view-inkon belül, ezt a megoldást el, így ritkén alkalmazzuk. Helyette template-eket használunk, azokban írjuk le a visszaadni kívánt adatok megjelenését, a view-ekben ezeket a template-eket rendereljük visszaadható állapotba.

Egyébként a sima HttpResponse mellett a Django rendelkezik még ennek speciális levezetéseivel, melyekkel egyszerűen tudunk szabványos módon értesíteni (HttpResponseRedirect), vagy hibázeneteket visszaadni (HttpResponseForbidden, HttpResponseNotFound). A *Not found (404)* hibázenetet egyébként a Http404 exception (kivétel) dobásával is elrhetjük, ez sok esetben kényelmesebb.

Egyszerűen a template használatra:

```

# views.py
from django.template import Context, loader
from django.http import HttpResponse

def hello(request, name='László'):
    if name == 'Sanyi':
        # Sanyi-t nem szeretjük, neki nem köszönünk
        return HttpResponseForbidden('Utálak...')

    t = loader.get_template('hello.html')
    c = Context({
        'name': name
    })

    return HttpResponse(t.render(c))

```

```

{# hello.html #}
<html>
<h1>Hello kedves {{ name }}! </h1>
<p>Hogy vagy?</p>
</html>

```

A rendereléshez szükséges megadni a kontextust, egy Context objektum formájában, ami kb. egy python dict-et tartalmaz, ennek segítségével adhatunk adatokat a template-nek. Mivel sokszor így van, hogy request is elrhet... legyen a template-ben, ezt a sima Context helyett RequestContext-et szoktam használni, ami ugyan olyan, csak második argumentumként meg kell adni a request objektumot.

Az utolsó néhány esetben mindig ugyanígy szerepelne a view függvényeinkben, ezért a Django srácok csináltak egy wrapper függvényt, hogy egyszerűen tudjunk a dolgokat, mint az előző view-timátrebbben:

```

# views.py
from django.shortcuts import render_to_response

def hello(request, name='László'):

    return render_to_response('hello.html', {'name': name})

```

A template-ek

A django template nyelve nem fog sok meglepetést okozni azoknak, akik használtak már valamilyen template nyelvet. A vezérlési szerkezeteket {% , %} közzé kell tenni, a változók {{ valami }} módjára írhatjuk ki, illetve megjegyzéseket is írhatunk hasonlóan: {# megjegyzés #}. A template-ekben blokkokat definiálhatunk, leszerelhetjük belőlük a szerelvényeket, illetve saját template-taget is tudunk készíteni.

A settings.py fájlban a TEMPLATE_DIRS listában tudjuk megadni, hogy a django hol keresse a template-eket, emellett a django motorban, azaz a telepített app-ok templates könyvtárába is, ha valamit nem talál az általunk megadott helyeken.

Vissza a képtárhoz

Első lépésben amolyan file-browzert akartam csinálni a képtárhoz. Egy (settings.py-ben megadott) könyvtár tartalmát bing, szethet, a felhasználó, azaz itt találhatók, képeket nézhet meg. Csináltam persze általánosabb - djangotól független - függvényeket, amiket az utils.py modulban helyeztem el a keptar appon belül.

Az utils.py tartalma nem téma a tutorialnak, de nyugodtan bele lehet nézni, fileok, és könyvtárak listázására, thumbnail készítésére, és egyébből hasonló dolgokra találhatók benne függvények.

Képtár view-t definiáltam ebben a lépésben, az egyik egy konkrét képet, a másik pedig egy könyvtár tartalmának megjelenítését, szerkesztését (csak a listát):

```
# views.py
from keptar.utils import get_file_list, get_abspath, get_parent, enrich

def listdir(request, path=""):
    try:
        files = get_file_list(path)
    except:
        return HttpResponseForbidden('Access Forbidden')

    return render_to_response('listdir.html', {
        'path': path,
        'parent': get_parent(path),
        'files': files,
    }, context_instance = RequestContext(request))

def showfile(request, fname):
    try:
        abspath = get_abspath(fname)
        fdata = enrich([fname])[fname]
    except:
        return HttpResponseForbidden('Access Forbidden')

    return render_to_response('showfile.html', {
        'parent': get_parent(fname),
        'fname': fname,
        'fdata': fdata,
    }, context_instance = RequestContext(request))
```

```
{# base.html #}
<!doctype html >
<html>
<head>
  <meta charset="utf-8"/>
  <title>{% block 'title' %}Keptar{% endblock %}</title>
  <link rel="stylesheet" href="/media/css/style.css"/>
</head>
<body>
  <div id="container">
    <div id="main">
      {% block 'main' %}
      {% endblock %}
    </div>
  </div>
</body>
</html>
```

```
{# listdir.html #}
{% extends 'base.html' %}
{% block 'main' %}
<h1>{{ path }}</h1>
<a href="{% url listdir parent %}">parent{% if parent %} ({{ parent }}){% endif %}</a>

<ul>
{% for fname, fdata in files.items %}
  <li><a href="{% fdata.url %}"> {{ fname }}</a></li>
{% endfor %}
</ul>
{% endblock %}
```

```
{# showfile.html #}
{% extends 'base.html' %}
{% block 'main' %}
<h1>{{ fname }}</h1>
<a href="{% url listdir parent %}">parent{% if parent %} ({{ parent }}){% endif %}</a>

<div>
  
</div>
{% endblock %}
```

Amint láthatjuk maguk a viewok nem túl bonyolultak, az `utils.py` függvényei segít, g, velünk, rájuk a `file/` könyvtár listát, illetve a `k, p` adatokat (*amit jelen esetben felfoghatunk modellnek is*), majd az adatokkal lerendereltettük a megfelelő... template-et.

Az `if`, `s` a `for` template-tageket nem magyarul nézünk, ellenben említ, `st`, `rd` emel az `url` tag, ami `{% url viewneve param1 param2 %}` módon visszaadja az adott view adott param, terezs, hez tartozó URL-t az `rv`, nyben `l`, `v`... `url` s. `py` alapján. Ha már szívesen keressük fel bele az `^j` view-kat:

```
url_patterns = patterns('',
    url(r'^/?$', 'keptar.views.listdir'),
    url(r'^list/(?P<path>.*)$', 'keptar.views.listdir', name='listdir'),
    url(r'^show/(?P<fname>.*)$', 'keptar.views.showfile', name='showfile'),
    url(r'^admin/', include(admin.site.urls)),
)
```


A `base.html` template-ben hivatkozok `keptar` stíluslapra is (stílus CSS), amit a Django is ki tud szolgáltatni statikus tartalomként, ehhez fel kell venni az URL minték kódot, az alábbi sort:

```
url(r'^media/(?P<path>.*)*$', 'django.views.static.serve', {'document_root': settings.MEDIA_ROOT}),
```

Természetesen, leírtam ezt nagyon nem javasolom, a webserver maga sokkal gyorsabban tud kiszolgálni statikus fájlokat, mint a Django.

Az extra `name` paraméterrel hivatkozhatunk a szabályunkra röviden, vö. az `{% url %}` tagben.

Ahogy említettem az `urlpatterns` használt paramétereket is a `settings.py`-ben kell beállítani, egyébként nem kell a felhasználóknak valami extra fájlban is turkolniuk, ha az alkalmazásomat ...kísérni szeretnék, k:

```
KEPTAR_ROOT = '/var/www/foto'
KEPTAR_URL = 'http://dyuri.horak.hu/foto/'
KEPTAR_EXTENSIONS = ['jpg', 'jpeg', 'png']
KEPTAR_THUMBDIR = '.tn'
KEPTAR_THUMB_SIZE = (120, 120)
KEPTAR_SHOW_HIDDEN = False
KEPTAR_ICONS = {
    'dir': 'http://dyuri.horak.hu/keptar/icons/tn_dir.jpg',
}
```

A következő ezeket a változókat egyébként az alábbi módon írtuk le:

```
from django.conf import settings
valami = settings.KEPTAR_ROOT
```

Elvileg később is vagyunk, a fejlesztői szerver futtatva (`python manage.py runserver`) bűnös, szívesen is a `settings.KEPTAR_ROOT` kódot tartalmazzuk.

A modell

Egy MVC jellegű alkalmazás legfontosabb része a modell, a legelső dolog, amit meg kell terveznünk, el kell készítenünk. (Bár, na, végülis, hagyom, végülis, szívesen, hogy eddig is volt modell a példában, megpedig az utolsó `py` modulon keresztül el, a fájlerendszer.)

A modellünket természetesen nem írja meg helyettünk a Django, de rendelkezik egy `elgő` `ORM`-mel, segít a validációban, és ugye van egy automatikusan generált admin felület, ahol végül is a modellünket piszkálhatjuk.

A modellünket - nem meglepő módon - az `app`unk `models.py` moduljában kell definiálnunk. Semmi trükköt nem kell elkövetni, hagyományos python osztályokat kell létrehozni, annyi megkötéssel, hogy az osztályoknak a `django.db.models.Model` osztályból kell származniuk, illetve az adatbázisban elhelyezkedő mezőket el kell definiálnunk kell.

Akár, párba szerettem volna egy foto-blog szerű funkciót, ami annyit tesz, hogy a fájlerendszer bűnös, szépen megjelölhetünk képeket, amik aztán a megjelölt képek sorrendjében jelennek meg a "blogban". A bejegyzéseknek szeretnék egy címet, és címkéket adni.

Mivel a Django alapból nem rendelkezik címkékkel, ez, tehát lehet, hogy nincs: vagy mi magunk csinálunk valami hasonlót, vagy keresünk egy későbbi megoldást Django `app` formájában, és azt használjuk. Úgy utóbbi mellett döntöttem - főként, hogy megmutassam hogyan kell egy későbbi Django `app`ot használni a projektünkben -, és [Alex Gaynor](#) `django-taggit` nevű munkéjét választottam. Ahhoz, hogy elröhögj, vö. a

modelljeink számára telepíteni kell (`pip install django-taggit`), s hozzáadni a `settings.py` modulunk `INSTALLED_APPS` listájához. Ha ezzel k, sz vagyunk, k, sztszk el a modell definícióját a `models.py` moduljában:

```
class BlogEntry(models.Model):

    path = models.CharField(max_length=1000, unique=True)
    title = models.CharField(max_length=200)
    user = models.ForeignKey(User)
    mark_date = models.DateTimeField('date marked', auto_now_add=True)
    tags = TaggableManager()

    class Meta:
        verbose_name = 'Blog bejegyzes'
        verbose_name_plural = 'Blog bejegyzesek'

    def is_valid(self):
        """ellenorzi, hogy a 'path' utvonalon levo file letezik-e"""
        abspath = get_abspath(self.path)
        return os.path.isfile(abspath)

    @property
    def fdata(self):
        """a fizikai filehoz tartozo adatok"""
        return enrich([self.path])[self.path]

    def __unicode__(self):
        return u"%s (%s)" % (self.title, self.path)
```

Els... ktrben a modell mez...it definiáljuk: - `path`: A k, p el, r, si ^tja, ahogy listézéskor is hivatkozunk rá. Egyedi, azaz egy k, pet csak egyszer jelölhetnk meg blogbejegyz, nek. - `title`: A bejegyz, stnk c.me. - `user`: A felhasználó, aki megjelölte a k, pet. A `ForeignKey` adattípuson keresztül hivatkozhatunk más modell-objektumokra. - `mark_date`: A megjelöl, s dátuma. Az `auto_now_add` param, ter miatt ezt a django majd automatikusan kitölti nekünk. - `tags`: Az el...bb installált `django-taggit` var, zs mez...je, ami a c.mk, z, st v, gzi.

Az adatmez...ktn k...v...l más dolgok is helyet kaptak az osztályban, ne felejtsek el a modell nem csak a perzisztencia r, teget, de az üzleti logikét is jelenti (bár jelen p, ldét igen...s t...lzes üzleti logikének nevezni :), annyit szerettem volna mondani ezzel, hogy nyugodt sz...vvel használjunk itt , rtelmes metódusokat, , s ne a view-inkban manipulálgassuk a modell-objektumainkat valami varézs f...ggv, nyekkel):

§ `is_valid`: Ellen...rzi, hogy a k, p fizikailag megtalálható-e.

§ `fdata`: Egy olyan `property`, ami a fizikai filehoz tartozó infókat adja vissza.

§ `__unicode__`: Az objektum `unicode` reprezentációja, amikor valahol `unicode`-d (illetve stringg,) kell konvertálni az objektumot, akkor ez hívódik meg. A legegyszer...bb ilyen eset a `print` objektum parancs.

Term, szetesen attól az adatbázisunkba nem kerül bele az ^j modell s, méja, mert bele...rtuk a `models.py` fileba, ki kell ehhez adnunk pár parancsot:

```
$ python manage.py validate
0 errors found
$ python manage.py syncdb
...
```

Ha mindezzel k, sz vagyunk, akkor parancssorból ki is tudjuk próbálni a modellünket. Ehhez a django szint, n ny^jt tēmogatést a shell management parancs formájában:

```
>>> from keptar.models import PBlogEntry
>>> from django.contrib.auth.models import User
>>> import datetime
>>> e = PBlogEntry(path='proba/kep.jpg', title='Proba Kep', mark_date=datetime.datetime.now(), user=User.objects.get())
>>> e.save() # ez menti el az adatbázisba
>>> e.tags.add('proba')
>>> e.tags.add('kep')
>>> masike = PBlogEntry.objects.get() # mar az adatbázisból szerdjuk ki a legelső PBlogEntry objektumot
>>> masike.tags.all()
[<Tag: proba>, <Tag: kep>]
>>> print masike
Proba Kep (proba/kep.jpg)
```

Ha az admin felületen is látni, s piszkálni szeretn, nk a modellünket, akkor ahhoz regisztrálnunk kell ...t az admin appnál. Ezt a regisztrációt elvileg bérhol megtehetn, nk - pl. magában a models.py-ben is, de, rdemes az appunk gytker, be egy admin.py nev, fileban megtenni, az admin app ezeket beh^zza. A regisztrácif maga egy admin.site.register(PBlogEntry) parancssal megoldhatf lenne, de lehet...s, g^nk van kicsit testre szabni az admin által generált listét, illetve formot, p, ldéul:

```
from keptar.models import PBlogEntry
from django.contrib import admin

class PBlogEntryAdmin(admin.ModelAdmin):

    # a listában metódusokat is szerepel tethetünk
    list_display = ('path', 'title', 'user', 'mark_date', 'is_valid')
    search_fields = ('path', 'title')
    date_hierarchy = 'mark_date'
    list_filter = ('user',)

admin.site.register(PBlogEntry, PBlogEntryAdmin)
```

Ha k, sz vagyunk, a fejleszt...i szerveret elindítva már láthatjuk is az ^j modellünket az admin oldalon, s...t az el...bb felvett proba/kep.jpg elemünk is megvan, amir...l látszik is a listében, hogy nem valid (javaslom ez, rt a ttrl, s, t, mert csak bekavar k, s...bb).

...rlapok

Gondolhatnénk, hogy ha már a django az admin felületre k, pes a modellekhez „rlapokat generálni, akkor miért ne lenne k, pes erre a m„veletre a mi k, r, s^nkre. %s valfban, amellet, hogy a django.forms modul elemeib...l k, zzel k, sz^ten, nk „rlapokat, az adminos mfkéhoz hasonlóan a modellekhez k, pes a django is „rlapot generálni (, n ezeket szint, n k^ltn, a forms.py modulba szoktam elhelyezni):

```
from django import forms
from keptar.models import PBlogEntry

class PBlogEntryForm(forms.ModelForm):
    class Meta:
        model = PBlogEntry
        exclude = ('user',)
        widgets = {
            'path': forms.HiddenInput(),
        }
```

Amint látható annyi a dolgunk, hogy leszermazunk a `django.forms.ModelForm` osztályból, s a belső... Meta osztályon belül mondhatjuk meg, hogy pl. melyik modellhez szeretnénk „rölapot” (`model = ModelClass`), illetve kiegészíthetünk olyan dolgokat, mint a generátorral, hogy milyen mezők maradjanak ki (exklúde), vagy hogy ha valami mezőt nem az alap, értelmezett widgettel (ez mi magyarul? •pftéelem?) szeretnénk megjeleníteni (widgets).

A shell management parancs segítségével meg is nézhetjük, hogy hogyan néz ki a generált formunk, egyszer „en” annyit kell tennünk, hogy pipálgatjuk az osztályt:

```
>>> from keptar.forms import PBlogEntryForm
>>> f = PBlogEntryForm()
>>> print f.as_p() # ha simán kiírjuk, akkor tr/td elemeket használ, amit en nem szeretek
<p><label for="id_title">Title:</label> <input id="id_title" type="text" name="title" maxlength="200" /></p>
<p><label for="id_tags">Tags:</label> <input type="text" name="tags" id="id_tags" /> A comma-separated list of tags.
<input type="hidden" name="path" id="id_path" /></p>
```

Tehát ahhoz, hogy a form megjelenjen az oldalunkon, elég annyit tennünk, hogy a view-nkban létrehozunk egy `PBlogEntryForm` objektumot, ennek esetleg adunk néhány alap, értelmezett „röket” (pl. path), ezt átadjuk a context objektumon keresztül a template-nek, ahol egy `<form>` tag-en belül kiírjuk.

Helyezzük el hát az „rölapot” a kívánt oldalon, közvetlen a kívánt helyre, csak akkor, ha beletett felhasználó nézi az oldalt. Ehhez módosítani kell a `showfile` nevű view-nkat:

```
# részlet a keptar/views.py fileból
def showfile(request, fname):

    try:
        abspath = get_abspath(fname)
        fdata = enrich([fname])[fname]
    except:
        return HttpResponseForbidden('Access Forbidden')

    # ha be van lépve valaki, akkor beteheti a képet a photoblogba
    if request.user.is_authenticated:
        try:
            # ha az elem már szerepel az adatbázisban, akkor a formban az o
            # adatait szeretnénk látni
            form = PBlogEntryForm(instance=PBlogEntry.objects.get(path=fname))
        except PBlogEntry.DoesNotExist:
            # ha nem szerepel, akkor új, üres formot szeretnénk
            form = PBlogEntryForm(initial={'path': fname})
    else:
        # nincs beépve senki, nem kell urlap
        form = None

    return render_to_response('showfile.html', {
        'pbform': form, # a template-nek pbform neven adjuk át az urlapot
        'parent': get_parent(fname),
        'fname': fname,
        'fdata': fdata,
    }, context_instance = RequestContext(request))
```

Illetve a `templates/showfile.html` template-ünkben is helyezzük el az „rölapot”:

Annyit tennünk még hozzá, hogy a django alap, értelmezetten bekapcsolt **CSRF** védelemmel rendelkezik, azaz a `settings.py` modulban a `MIDDLEWARE_CLASSES` listában szerepel a `django.middleware.csrf.CsrfViewMiddleware` osztály. Ezen esetben az összes „rölapunknál” használni kell a `{% csrf_token %}` template-taget, ami egy hidden mezőben tartalmazni fogja a felhasználó biztonsági kódját, ami nélkül az „rölap” nem nyílt.

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

{% block 'main' %}
<h1>{{ fname }}</h1>
<a href="{% url listdir parent %}">parent{% if parent %} ({{ parent }}){% endif %}</a>

{% if pbform %}
<form action="{% url submitpbentry %}" method="post">
  {% csrf_token %}
  {{ pbform.as_p }}
  <p><input type="submit" name="submitpe" value="Ok" /></p>
</form>
{% endif %}

<div>
  
</div>
{% endblock %}
```

A form action param, ter, nek egy külön view-t adtam meg, ami az „rlap hibáinak kezel, se szempontjéből (amit a django szintén „gyesen t, mogat, de most nem foglalkozni•k vele) nem felt, tlen el...nyts, viszont , n szeretem külön tudni a form kezel... view-kat, levélasztva ...ket a tisztén megjelen...t, s, rt felel... r, szekr...l (kicsit controller-view jelleg, sz, tvélasztés, de ne er...tessék), illetve „gy könnyebb az „rlapot több különbtz... oldalon is használni. Az „rlap feldolgozó view ilyenkor egy étirény...téssel tovább...tja a c, lhelyre a bng, sz...t, aminek az az el...nye is megvan, hogy a bng, sz... ..jrat...s gombjának hatására nem kldi el ^jra az „rlapot. Léssuk hét az „rlap feldolgozó view-nkat:

```
def submitpbentry(request):

    # ha nincs belépve, akkor nem szabad
    if not request.user.is_authenticated:
        return HttpResponseRedirect('Access Forbidden')
    try:
        # ha az adott kep mar szerepel az adatbazisban, akkor az o adatait szeretnenk frissiteni
        f = PBlogEntryForm(request.POST, instance=PBlogEntry.objects.get(path=request.POST['path']))
    except PBlogEntry.DoesNotExist:
        # ha nem szerepel, akkor uj elemet hozunk létre a form alapjan
        f = PBlogEntryForm(request.POST)

    # ha a felhasználót nem raknánk hozzá, akkor simán menthetnénk,
    # így viszont külön kell menteni a kapcsolódó adatokat is (tag)
    pbe = f.save(commit=False)
    pbe.user = request.user
    pbe.save()
    # kapcsolódó adatok (tag-ek) mentése
    f.save_m2m()

    return HttpResponseRedirect(reverse('showfile', args=[pbe.path]))
```

Kicsit bonyolultabb eset ez annél, amivel kezdeni kellene (rdemes megn,zni a jfval egyszer„bb [hivatalos django tutorial ide vonatkozó r, sz, t](#)), de jf p, lda arra, hogy adhatunk a be, rkez... „rlaphoz olyan adatokat, amit nem szeretn, nk semmik, pp a felhasználófra b...zni. Term, szetesen az urljeink k...z, is fel kell venni az ^j view-t, amit az url s. py modulban az alábbi mfdon tehetnk meg:

```
urlpatterns = patterns('',
    # sokminden ...
    url(r'^submitpe$', 'keptar.views.submitpbentry', name='submitpbentry'),
)
```

Ezek után b...szen jeltölgethetjük a képeinket, amiket utánna az admin oldalon szerkeszthetünk is. M... csak egy ^j view/tempalte p...rosra van szükségünk, hogy blog szer...en n, zegethessük a megjelölt képeket:

```
# keptar/views.py

def pblog(request, id=None, slug=None):

    try:
        # ha az id nincs megadva, akkor a legutolsot jelenitjuk meg
        if id is None:
            pbe = PblogEntry.objects.latest('mark_date')
        else:
            pbe = PblogEntry.objects.get(pk=id)
    except PblogEntry.DoesNotExist:
        # hibas id volt megadva, vagy nincs meg bejegyzes
        return render_to_response('pblog.html',
            {},
            context_instance = RequestContext(request))

    # elozo es kovetkezo elem meghatarozasa idorendi sorrendben
    next = PblogEntry.objects.filter(mark_date__gt=pbe.mark_date).order_by('mark_date')[:1]
    # python 2.6+ eseten ez sokkal szebb lenne:
    # next = next[0] if len(next) > 0 else None
    if next:
        next = next[0]
    prev = PblogEntry.objects.filter(mark_date__lt=pbe.mark_date).order_by('-mark_date')[:1]
    if prev:
        prev = prev[0]

    return render_to_response('pblog.html', {
        'pbe': pbe,
        'next': next,
        'prev': prev,
    }, context_instance = RequestContext(request))
```

```
{# templates/pblog.html #}
{% extends 'base.html' %}

{% block 'main' %}
{% if pbe %}
<h1>{{ pbe.title }}</h1>
<div class="nav">
    {% if prev %}<a href="{% url pblog prev.id prev.title|slugify %}">Previous</a>{% endif %}
    <a href="{% url showfile pbe.path %}">Browse</a>
    {% if next %}<a href="{% url pblog next.id next.title|slugify %}">Next</a>{% endif %}
</div>

<h2 class="tags">Tags:
    {% for tag in pbe.tags.all %}
    <span class="tag">{{ tag }}</span>{% if not forloop.last %}, {% endif %}
    {% endfor %}
</h2>

<div>
    
</div>
{% else %}
<h1>The photoblog is still empty...</h1>
{% endif %}
{% endblock %}
```

```
# urls.py részlet
urlpatterns = patterns('',
```

```
url(r'^pblog/(?P<id>\d+)/(?P<slug>[\w-]*)/$', 'keptar.vi.ews.pblog', name='pblog'),
url(r'^pblog/(?P<id>\d+)/$', 'keptar.vi.ews.pblog'),
url(r'^/?$', 'keptar.vi.ews.pblog'),
# ...
)
```

A sima `/pblog/<id>/`, s a f...oldal (`/`) url-, n k...l legels... helyen egy olyan mintét adtam meg, ami az azonos...t...f után egy *slugot* v...r, ami egy bet...kb...l, sz...mokb...f...l, s k...t...jelb...l...l...f valami. Ezt is le...rolhatn...m a modellemben, de, rtelme legink...bb az, rt van, hogy *szebbek* legyenek az url-ek, az ilyesmit a keres...k is jobb helyre szokt...k sorolni, s az emberek is sz...vesebben kattintanak r...l. Ezt a slugot a template-en bel...l a sulgi fy sz...r...vel k, sz...thetj...k el.

Elind...tjuk, s **fr, l, nk!**

Id...k...t...ben r...j...ttem, hogy finoman sz...f...lva buta dolog, hogy a settings.py-be beledr...ftoztam a saj...t g...pemen l...v... k, pek el, r, si ...tj...t, ez, rt eln, z, st k, rek, a [relingdir](#) mercurial c...mke alatt el, rhet... az a verzif, ahol azt ...talak...tottam egy relat...v el, r, ss, , ami a projekt i...mages k...t...nyvt...ra, illetve ide el is helyeztem k, t k, pet. Sz...f...val, ha valaki csak ...gy lett...lti, s elind...tja, akkor ez a verzif j...f es, llyel produk...l valami, rtelmes eredm, nyt.

Konkl...tzi...t

Mit is ad nek...nk a django? Egy MVC jelleg... keretrendszer, modell oldalon okos ORM t...mogat...ssal, egy j...f...l haszn...lhat...f, kib...v...thet... template nyelvet, ...rlap kezel, st, URL routingot, middleware (*k...t...ztesr...teg-modul? omg*) rendszert, egy meglep...en j...f...l haszn...lhat...f automatikusan gener...lt admin fel...lletet, illetve ami szerintem a legnagyobb ereje - f...k, nt ...j project indul...skor -, azok a k...t...nnyed, n be, p...thet... alkalmaz...sok rendszere. Mindemellett nagy el...ny, hogy nem k...tti meg a kez...nket, a felsorolt dolgok k...t...z...l semmit sem k...ttelez... haszn...l...nunk. Nem tetszik a template nyelv? Seba...j, haszn...lhatunk b...rmi m...s (python) template nyelvet, pl. [Jinjat](#). Nem tetszik az ...rlap kezel, s (ez mondjuk meglepne), csin...lhatunk saj...tot, vagy ott van a [WTForms](#). Az ORM a sz...k keresztmetszet? H...t nem k...ttelez... haszn...l...ni, haszn...lhatunk tiszt...n SQL-t, vagy ak...r valami nem rel...c...ifs adatb...zist is, pl. ott a [mongoengine](#) (ez esetben mondjuk az automatikusan gener, lt admin fel, lett...l is el kell b...cs...znunk).

Pr...f...b...ltam min, l teljesebb k, pet adni, m, gis most ...gy, rzem, hogy csak a felsz...nt karcolgattam. De h...t ha nem lenne m...r mir...l ...rni, akkor az oldalra sem lenne sz...tks, g tov...bb :) Mindenkinek tr...tmteli ismerked, st k...v...n...k a djangoval, s ha k, rd, setek van, ne tarts...t...k magatokban! Term, szetesen a projekt messze nincs m, g k, sz, ha id...m enged... folytatni fogom, s ha valami, rdekeset csin...lok, akkor arr...f...l megpr...f...b...lok besz...m...olni. Egy, bk, nt meg az eg, sz fent van a [bitbucketen](#), szabad forkolni, s sz...vesen veszem a *pull-requesteket* :)

A cikksorozat r, szi:

Š [Django, egy p, ld...n kereszt...l I. - Az alapok](#)

Š [Django, egy p, ld...n kereszt...l II. - View-k, s template-ek](#)

Š [Django, egy p, ld...n kereszt...l III. - A modell](#)

A teljes projekt szabadon el, rhet... a [bitbucketen](#).