Django výpisky

1 Instalace

- instalace Python
- odstranění staré verze Django
- instalace Django příkazem *python setup.py*
- ověření správné instalace příkazem *import django* v interpretu
- přidání *C:\Python26;C:\Python26\Lib\site-packages\django\bin;* do systémové proměnné PATH

2 Vytvoření projektu

Příkaz *django-admin.py startproject myproject* vytvoří v aktuálním adresáři projekt s názvem *myproject*. Projekt nesmí mít název, shodující se s nějakou build-in komponentou Pythonu a Django. Kód není vhodné z hlediska bezpečnosti ukládat do kořenového adresáře webserveru (*/var/www*). Toto se u Django, narozdíl od PHP, nedělá.

Nový projekt obsahuje soubory:

- *__init__.py* – vytvoření package
- *manage.py* – utilita pro správu projektu
- *settings.py* – nastavení projektu
- *urls.py* – deklarace URL adres

2.1 Manage.py

- *manage.py runserver* – spustí vývojový webserver na adrese http://127.0.0.1:8000/, není vhodný pro ostrý provoz (lepší použít Apache), při vytvoření nového souboru v projektu je server nutno restartovat
- *manage.py runserver 1.2.3.4:8080* – spustí vývojový webserver na dané IP a portu
- *manage.py syncdb* – vytvoří všechny potřebné tabulky, indexy a příkazy (které ještě neexistují) v databázi pro všechny aplikace, definované v proměnné *INSTALLED_APPS*, (provede příkazy z *manage.py sqlall myapp*), při první synchronizaci aplikace *django.contrib.auth* rovněž nabídne vytvoření účtu superuživatele
- *manage.py createsuperuser* – vytvoří účet superuživatele
- *manage.py validate* – kontroluje chyby v definici modelů aplikací
- *manage.py sql myapp* – vypíše SQL kód pro vytvoření tabulek (*CREATE TABLE*) databáze aplikace
- *manage.py sqlcustom myapp* – vypíše SQL kód pro všechny uživatelské příkazy
- *manage.py sqlclear myapp* – vypíše SQL kód pro odstranění tabulek (*DROP TABLE*) databáze aplikace
- *manage.py sqlindexes myapp* – vypíše SQL kód pro vytvoření indexů (*CREATE INDEX*) databáze aplikace
- *manage.py sqlall myapp* – kombinace příkazů *sql*, *sqlcustom*, *sqlindexes*
- *manage.py shell* – spustí interpret a nastaví prostředí projektu (přidá *myproject* do systémové proměnné PATH a nastaví proměnnou prostředí *DJANGO_SETTINGS_MODULE*)

2.2 Settings.py

- *DATABASE_ENGINE* – typ databáze ('postgresql_psycopg2', 'postgresql', 'mysql', 'sqlite3', 'oracle')
- *DATABASE_NAME* – název databáze nebo absolutní cesta k sqlite3 ('D:/myproject/sql.db'), pokud soubor sqlite3 neexistuje, bude vytvořen při první synchronizaci, u databází PostgreSQL nebo MySQL je třeba databázi vytvořit příkazem *CREATE DATABASE jmeno_databaze;*
- *DATABASE_USER* – databázové uživatelské jméno (v sqlite3 se nepoužívá)
- *DATABASE_PASSWORD* – databázové heslo (v sqlite3 se nepoužívá)
- *DATABASE_HOST* – adresa databázového serveru, pokud je databáze na stejném serveru jako Django, nechá se řetězec prázdný (v sqlite3 se nepoužívá)
- *DATABASE_PORT* – port databázového serveru
- *TEMPLATE_DIRS* – cesta k adresářům se šablonami HTML
- *INSTALLED_APPS* – seznam aplikací
    - *django.contrib.auth* – autentifikační systém

- *django.contrib.contenttypes* – framework pro typy obsahu
- *django.contrib.sessions* – framework pro sessions
- *django.contrib.sites* – framework pro spravování několika samostatných webů v jedné instanci Djanga
- *django.contrib.admin* – administrační rozhraní
- *myproject.myapp* – vlastní aplikace

3 Vytvoření aplikace

Příkaz *python manage.py startapp myapp* vytvoří v aktuálním adresáři (pro zjednodušení nastavíme adresář projektu *myproject*) aplikaci s názvem *myapp*. Důsledkem zjednodušení bude vzájemné propojení aplikace a projektu.

Nová aplikace obsahuje soubory:

- *__init__.py* – vytvoření package
- *models.py* – definice modelů (popis dat), struktura databáze s dodatečnými metadaty, jeden model je jedna databázová tabulka
- *views.py* – definice pohledů (šablony webové stránky)

3.1 Models.py

Modely popisují data aplikace (strukturu databáze). Ctí princip DRY (Don't repeat yourself). Každý model je jedna databázová tabulka a je potomkem třídy *django.db.models.Model*. Proměnné modelu definují jednotlivá databázová pole – instance třídy *Field*. Pojmenováním pole *Field* se definuje strojové jméno, přes které se bude přistupovat k hodnotě pole a v databázi bude využito k pojmenování sloupce databázové tabulky. Volitelně lze v prvním argumentu instance *Field* definovat human-readable jméno pole.

```
from django.db import models
import datetime

class Poll(models.Model):                                #definice modelu
    question = models.CharField(max_length=200)          #textove pole o delce 200 znaku
    pub_date = models.DateTimeField('date published')#datum s human-readable jmenem

    def __unicode__(self):                               #definice unicode metody
        return self.question

    def was_published_today(self):                       #definice vlastni metody
        return self.pub_date.date() == datetime.date.today()

    #human-readable pojmenovani metody
    was_published_today.short_description = 'Publikovano dnes?'

class Choice(models.Model):                              #definice dalsiho modelu
    poll = models.ForeignKey(Poll)                       #cizi klic
    choice = models.CharField(max_length=200)            #textove pole o delce 200 znaku
    votes = models.IntegerField()                        #celociselne pole

    def __unicode__(self):                               #definice unicode metody
        return self.choice
```

Unicode metody jsou užitečné při práci v interaktivním režimu pro výpis (např. *>>> Poll.objects.all()*) a používají se v automaticky generovaném administračním rozhraní.

Modely je potřeba po vytvoření aktivovat příkazem *manage.py syncdb*. Poté se vytvoří databázové schéma aplikace a databázové API pro přístup k datům. Název tabulky se automaticky generuje z názvu aplikace a názvu modelu. Primární klíč *'id'* se definuje automaticky. Název cizího klíč se automaticky generuje z názvu pole a suffixu *'_id'*.

3.2 Urls.py

V adresáři s projektem v souboru *urls.py* v proměnné *urlpatterns* jsou deklarovány URL adresy. Pro určitý regulární výraz, odpovídající nějaké URL adrese, se zavolá odpovídající funkce z *views.py*. Příklad volání: *detail(request=<HttpRequest object>, poll_id='23')* pro regulární výraz *r'^polls/(?P<poll_id>\d+)/$'*, přičemž část textu v závorkách se odešle jako argument do view funkce a *'?P<poll_id>'* definuje název, kterým se zachycená část pojmenuje. Tyto regulární výrazy neprohledávají parametry GET, POST, a ani názvy domén. Při požadavku na *http://www.example.com/myapp/?page=3*, bude opět hledat *myapp/*.

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

from django.contrib import admin
admin.autodiscover()

urlpatterns = patterns('',
    (r'^polls/$', 'myproject.myapp.views.index'),
    (r'^polls/(?P<poll_id>\d+)/$', 'myproject.myapp.views.detail'),
    (r'^polls/(?P<poll_id>\d+)/results/$', 'myproject.myapp.views.results'),
    (r'^polls/(?P<poll_id>\d+)/vote/$', 'myproject.myapp.views.vote'),
    (r'^admin/', include(admin.site.urls)),
)
```

Každá aplikace může mít definovány svoje vlastní *urls.py*, které se pak importují do hlavního *urls.py* projektu.

```
# urls.py projektu
from django.conf.urls.defaults import *

from django.contrib import admin
admin.autodiscover()

urlpatterns = patterns('',
    (r'^polls/', include('myproject.myapp.urls')),
    (r'^admin/', include(admin.site.urls))
)
```

```
# urls.py aplikace
from django.conf.urls.defaults import *

# prvni argument je zkratka k views, aby se nemusela opakovat v kazdem callbacku
urlpatterns = patterns('myproject.myapp.views',
      (r'^$', 'index'),
      (r'^(?P<poll_id>\d+)/$', 'detail'),
      (r'^(?P<poll_id>\d+)/results/$', 'results'),
      (r'^(?P<poll_id>\d+)/vote/$', 'vote'),
)
```

3.3 Views.py

Příklad základního view:

```
from django.http import HttpResponse

def index(request):
    return HttpResponse("Hello, world. You're at the poll index.")

def detail(request, poll_id):
    return HttpResponse("You're looking at poll %s." % poll_id)

def results(request, poll_id):
    return HttpResponse("You're looking at the results of poll %s." % poll_id)

def vote(request, poll_id):
    return HttpResponse("You're voting on poll %s." % poll_id)
```

Každé view vrací objekt HttpResponse, nebo výjimku (např. Http404). Příklad jednoduchého view, které něco dělá:

```
from django.http import HttpResponse
from myproject.myapp.models import Poll

# zobrazi poslednich 5 serazenych otazek z ankety
def index(request):
    latest_poll_list = Poll.objects.all().order_by('-pub_date')[:5]
    output = ', '.join([p.question for p in latest_poll_list])
    return HttpResponse(output)
```

View s použitím HTML šablony [1]:

```
from django.template import Context, loader
from django.http import HttpResponse
from myproject.myapp.models import Poll

# zobrazi poslednich 5 serazenych otazek z ankety v HTML sablone
def index(request):
    latest_poll_list = Poll.objects.all().order_by('-pub_date')[:5]
    t = loader.get_template('myapp/index.html')
    c = Context({
        'latest_poll_list': latest_poll_list,
    })
    return HttpResponse(t.render(c))
```

View s použitím HTML šablony a funkce *render_to_response()*:

```
from django.shortcuts import render_to_response
from myproject.myapp.models import Poll

# zobrazi poslednich 5 serazenych otazek z ankety v HTML sablone
def index(request):
    latest_poll_list = Poll.objects.all().order_by('-pub_date')[:5]
    return render_to_response('myapp/index.html', {'latest_poll_list':
latest_poll_list})
```

View s voláním chyby 404 [2]:

```
from django.shortcuts import render_to_response
from django.http import Http404
from myproject.myapp.models import Poll

# zobrazi detail ankety s osetrenim na chybu 404
def detail(request, poll_id):
    try:
        p = Poll.objects.get(pk=poll_id)
    except Poll.DoesNotExist:
        raise Http404
    return render_to_response('myapp/detail.html', {'poll': p})
```

View s voláním chyby 404 a funkce *get_object_or_404()*, která přijímá jako první argument model Djanga, a na dalších pozicích libovolný počet argumentů, které se předají funkci *get()*. Pokud se podle dodaných parametrů objekt nenalezne, vyvolá se vyjímka *Http404*. Existuje také funkce *get_list_or_404()*, která používá metodu *filter()*. Funkce vyvolá výjimku *Http404* pokud je seznam prázdný.

```
from django.shortcuts import render_to_response, get_object_or_404
from myproject.myapp.models import Poll

# zobrazi detail ankety s osetrenim na chybu 404
def detail(request, poll_id):
    p = get_object_or_404(Poll, pk=poll_id)
    return render_to_response('myapp/detail.html', {'poll': p})
```

View pro zpracování formuláře [3]. Z objektu *request.POST* můžeme vytahovat odeslaná data s pomocí jejich jména. Hodnoty z *request.POST* se vrací vždy v podobě řetězce. Podobně funguje i *request.GET*. Funkce *reverse()* přesměrovává na určité view.

```
from django.shortcuts import get_object_or_404, render_to_response
from django.http import HttpResponseRedirect, HttpResponse
from django.core.urlresolvers import reverse
from django.template import RequestContext
from myproject.myapp.models import Choice, Poll

def vote(request, poll_id):
    p = get_object_or_404(Poll, pk=poll_id)
    try:
        selected_choice = p.choice_set.get(pk=request.POST['choice'])
    except (KeyError, Choice.DoesNotExist):
        # zobrazime formular znovu, tentokrat s chybovou hlaskou
        return render_to_response('myapp/detail.html', {
            'poll': p,
            'error_message': "Nevybral(-a) jste zadnou anketni volbu.",
        }, context_instance=RequestContext(request))
    else:
        selected_choice.votes += 1
        selected_choice.save()
        # Po kazdem uspesnem zpracovani POST dat je treba vyvolat
        # HttpResponseRedirect. Tato praktika zabrani dvojimu
        # odeslani (a uzlozeni) dat z formulare v situacich, kdy
        # uzivatel klikne na tlacitko "Zpet" ve svem webovem
        # prohlizeci.
        return HttpResponseRedirect(reverse('myproject.myapp.views.results',
args=(p.id,)))
```

View s použitím HTML šablony a funkce *render_to_response()* [4]:

```
from django.shortcuts import get_object_or_404, render_to_response
from myproject.myapp.models import Choice, Poll

def results(request, poll_id):
    p = get_object_or_404(Poll, pk=poll_id)
    return render_to_response('myapp/results.html', {'poll': p})
```

3.4 Šablony

Cesta k šablonám se nastavuje v souboru *settings.py* v proměnné *TEMPLATE_DIRS*. Každá aplikace má svůj adresář se šablonami. Příklad HTML šablony *myapp/index.html* [1]:

```
{% if latest_poll_list %}
    <ul>
    {% for poll in latest_poll_list %}
        <li>{{ poll.question }}</li>
    {% endfor %}
    </ul>
{% else %}
    <p>No polls are available.</p>
{% endif %}
```

Příklad HTML šablony *myapp/detail.html* [2]:

```
<h1>{{ poll.question }}</h1>
<ul>
{% for choice in poll.choice_set.all %}
    <li>{{ choice.choice }}</li>
{% endfor %}
</ul>
```

Šablona pro chybu 404 se ukládá do kořenového adresáře se šablonami do souboru *404.html*. View 404 se vyvolá, pokud není nalezena shoda v regulárních výrazech URLconf. Lze použít výchozí view 404. V Debug režimu se view 404 nepoužije. Zobrazuje se traceback. Stejným způsobem lze definovat šablonu pro chybu 500 (chyba serveru). Ta je vyvolána v případě runtime chyby. Šablona se ukládá do souboru *500.html*.

Příklad HTML šablony *myapp/results.html* [4]:

```
<h1>{{ poll.question }}</h1>
<ul>
{% for choice in poll.choice_set.all %}
    <li>{{ choice.choice }} - {{ choice.votes }} vote{{ choice.votes|pluralize }}</li>
{% endfor %}
</ul>
```

3.5 Formuláře

Příklad HTML formuláře *myapp/detail.html*. Po odeslání formuláře se zavolá view *Vote* [3].

```
<h1>{{ poll.question }}</h1>

{% if error_message %}<p><strong>{{ error_message }}</strong></p>{% endif %}

<form action="/polls/{{ poll.id }}/vote/" method="post">
{% csrf_token %}
{% for choice in poll.choice_set.all %}
    <input type="radio" name="choice" id="choice{{ forloop.counter }}" value="{{
choice.id }}" />
    <label for="choice{{ forloop.counter }}">{{ choice.choice }}</label><br />
{% endfor %}
<input type="submit" value="Vote" />
</form>
```

4 Databázové API

```
>>> from myproject.myapp.models import Poll, Choice
>>> import datetime

# vypise seznam vsech anket modelu Poll
>>> Poll.objects.all()

# vytvoreni noveho zaznamu ankety
>>> p = Poll(question="Tak co?", pub_date=datetime.datetime.now())
>>> p.save()

# vypis jednotlivych poli modelu Poll
>>> p.id
>>> p.question
>>> p.pub_date

# uprava pole
>>> p.pub_date = datetime.datetime(2007, 4, 1, 0, 0)
>>> p.save()

# vyhledavani v databazi
>>> Poll.objects.filter(id=1)
>>> Poll.objects.filter(question__startswith='Tak')
>>> Poll.objects.get(pub_date__year=2007)
>>> Poll.objects.get(id=1)

# vyhledavani podle primarniho klice a volani vlastni metody
>>> p = Poll.objects.get(pk=1)
>>> p.was_published_today()

# vytvoreni noveho zaznamu odpovedi, pridani do sady odpovedi a vlozeni do databaze
>>> c = p.choice_set.create(choice='Just hacking again', votes=0)

# pristup k souvisejicim objektum
>>> c.poll
>>> p.choice_set.all()
>>> p.choice_set.count()

# pokrocile filtrovani
>>> Choice.objects.filter(poll__pub_date__year=2007)

# smazani zaznamu
>>> c.delete()
```

5 Administrační rozhraní

Pro aktivaci administračního rozhraní je potřeba přidat *'django.contrib.admin'* do seznamu *INSTALLED_APPS*, provést synchronizaci databáze a správně upravit *urls.py*. Admin pak běží na adrese http://127.0.0.1:8000/admin/. Pro přidání správy aplikace *myapp* do administračního rozhraní je potřeba vytvořit soubor *admin.py* v adresáři aplikace.

```
from myproject.myapp.models import Poll
from django.contrib import admin

admin.site.register(Poll)                #pridani modelu Poll
```

5.1 Vzhled administračních formulářů

Formuláře jsou automaticky generovány podle modelů. Zobrazení administračních formulářů lze ovlivňovat. Prohození políček formuláře:

```
# administratorsky objekt
class PollAdmin(admin.ModelAdmin):
    fields = ['pub_date', 'question']

admin.site.register(Poll, PollAdmin)     #pridani modelu Poll
```

Formuláře lze rozdělit na skupiny. Libovolnému poli lze přiřadit HTML třídu (class), např. *collapse* pro zabalení skupiny:

```
# administratorsky objekt
class PollAdmin(admin.ModelAdmin):
    fieldsets = [
        (None,               {'fields': ['question']}),
        ('Date information', {'fields': ['pub_date'], 'classes': ['collapse']}),
    ]

admin.site.register(Poll, PollAdmin)     #pridani modelu Poll
```

Zobrazení jednotlivých polí v libovolném pořadí ve výpisu všech záznamů modelu (change list):

```
list_display = ('question', 'pub_date', 'was_published_today')
```

Filtr ve výpisu záznamů modelu:

```
list_filter = ['pub_date']
```

Vyhledávání ve výpisu záznamů modelu:

```
search_fields = ['question']
```

Hierarchická navigace podle data:

```
date_hierarchy = 'pub_date'
```

5.2 Přidání souvisejících objektů

Přidání souvisejícího objektu *Choice* lze provést pomocí funkce *admin.site.register()*, nebo můžeme související objekt *Choice* spravovat přímo v administraci *Poll*. Pro zobrazení souvisejícího modelu můžeme využít řádkového zobrazení *admin.StackedInline* nebo tabulkového zobrazení *admin.TabularInline*.

```
from myproject.myapp.models import Poll
from myproject.myapp.models import Choice
from django.contrib import admin

# radkove zobrazeni souvisejiciho modelu
class ChoiceInline(admin.StackedInline):
    model = Choice                          #objekt modelu
    extra = 3                               #zobrazi 3 pole

# administratorsky objekt
class PollAdmin(admin.ModelAdmin):
    inlines = [ChoiceInline]                #pridani souvisejiciho objektu

admin.site.register(Poll, PollAdmin)        #pridani modelu Poll
admin.site.register(Choice)                 #pridani modelu Choice
```

5.3 Úprava vzhledu prostředí

Pro úpravu vzhledu administračního prostředí si vytvoříme vlastní šablony. Cestu k šablonám musíme nastavit v souboru *settings.py* v proměnné *TEMPLATE_DIRS*:

```
TEMPLATE_DIRS = (
    "D:/DJANGO/myproject/templates ",    #adresar se sablonami
)
```

Zkopírujeme například */Python26/Lib/site-packages/django/contrib/admin/templates/base_site.html* do *D:/DJANGO/myproject/templates/admin/base_site.html*. Nyní můžeme šablonu libovolně upravovat.