

Webentwicklung mit Django

Michael Ziegler

WebDevFulda
standpy.de

3. März 2010

Inhalt

- 1 Webentwicklung - wo ist das Problem?
- 2 Was kann denn mehr?
- 3 O RLY?
- 4 Was ist alles möglich?
- 5 Und jetzt?

Hobbybereich

- Kaum Organisation
- Chaotischer Code
- Immer wieder die gleichen Fehler (SQL Injections, XSS, Unicode in Latin1-Feldern)
- Oft SQL, PHP und HTML in *einer* Datei

```
1  $ergebnis = safe_query("SELECT * FROM ".PREFIX.  
2      "user WHERE nickname = '$nickname' ");  
3  $num = mysql_num_rows($ergebnis);  
4  if($num)  
5      $error []=$_language->module['nickname_inuse'];
```

- ⇒ schlecht wartbar

Professionell

- Mehr Organisation als nötig
- viele Frameworks zielen auf Riesenprojekte ab
- die meisten Projekte sind zu klein für ihre Umgebung
- ⇒ Overhead (z. B. Deploy-Zeiten)

```
14:59:57,679 INFO [ServerImpl] JBoss (Microcontainer)
[5.1.0.GA (build: SVNTag=JBoss_5_1_0_GA
date=200905221053)] Started in 32s:233ms
```

USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT
root	2269	10.5	8.5	1337148	700232	pts/0	S1+

- ⇒ genauso schlecht wartbar

Inhalt

- 1 Webentwicklung - wo ist das Problem?
- 2 Was kann denn mehr?**
- 3 O RLY?
- 4 Was ist alles möglich?
- 5 Und jetzt?

Darf ich vorstellen:



www.python.org



www.djangoproject.com

Python

- Freie Software unter der Python License
- wird in Linuxen für alle Arten von Scripts eingesetzt
- für viele OS verfügbar (Windows, Linux, OSX, BSD, Symbian uvm)
- hat eine *enorm* große Standardbibliothek
- es gibt viele Erweiterungen und Module (Debian: ca. 1200)
- Folge: man kann *alles* damit machen!
- Legokasten: alle Teile sind vorhanden, nur noch zusammenfügen

Django

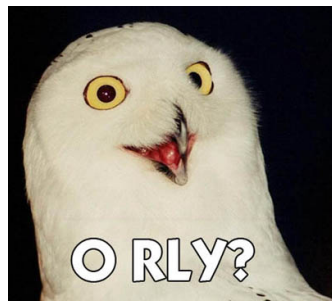
- Beschreibung von djangoproject.com:

Django is a high-level Python Web framework that encourages rapid development and clean, pragmatic design.

- Freie Software unter der BSD-Lizenz
- für alle OS verfügbar, auf denen Python läuft
- viele fertige Apps im Internet

Inhalt

- 1 Webentwicklung - wo ist das Problem?
- 2 Was kann denn mehr?
- 3 **O RLY?**
 - Rapid development
 - Clean, pragmatic design
 - MVC-orientiertes Design
 - DRY: Don't repeat yourself
- 4 Was ist alles möglich?
- 5 Und jetzt?



Rapid development

- **Entwickler sind faul!**
- Django erledigt die Basics - besser als wir je Bock hätten
- der Entwickler konzentriert sich direkt auf seine App
- Django liefert den Rest - **vollständig**
 - Datenbank-Abstraktion
 - Formulare (inkl. Validierung)
 - Template-System
 - I18n
 - URLs auflösen/erzeugen
- ⇒ Rad nicht neu erfinden
- ⇒ Genauso wichtig: Vermeidung von Overhead

Clean, pragmatic design

- “der Entwickler konzentriert sich direkt auf seine App”
- Struktur der App muss **durchdacht** sein!
 - wenn ja, kann man in wenig Zeilen Code viel erreichen
 - wenn nicht, gibt's Schwierigkeiten
 - ⇒ wenn man auf Probleme stößt: Design überdenken!
- Übungssache
- Unterstützung seitens Django:
 - MVC-orientiertes Design: Model-View-Controller
 - DRY: Don't Repeat Yourself

MVC-orientiertes Design

- normalerweise: Model-View-Controller
- in Django eher: Model-View-Template
- Code wird getrennt in drei Teilbereiche:
 - 1 Models: Datenmodellierung als Objekte
 - Daten als Variablen, Operationen als Methoden
 - Legt fest welche Daten es gibt
 - 2 View: Steuerung der Daten-Präsentation
 - **Nur** Präsentation, keine Operationen!
 - Legt fest welche Daten angezeigt werden
 - 3 Template: Darstellung der Daten, z. B. als HTML
 - Auch andere textbasierte Formate möglich
 - Legt fest, wie die Daten aussehen
- Models und Views sind *reiner* Python-Code
- Templates sind HTML-Code mit Annotationen

Beispiel: Model "Repository"

```
1 class Repository( models.Model ):
2     name          = models.SlugField()
3     owner         = models.ForeignKey( User )
4     basepath     = models.CharField( max_length=200 )
5     description  = models.TextField( blank=True )
6     writers      = models.ManyToManyField( User, blank=True )

8     tipctx = property( get_tip_ctx, doc="A HG change context "
9                       "object for the tip version." )

11    def get_file( self, name ):
12        """ Get the file given in name from the tip version of
13            this repository.
14        """
15        return self.tipctx.filectx( name ).data()
```

Beispiel: View “showpygfile”

```
1 def showpygfile( request, username, reponame, filename ):
2     repo = Repository.objects.get(
3         owner__username=username, name=reponame )

5     contents = highlight( repo.get_file(filename),
6                           get_lexer_for_filename(filename),
7                           HtmlFormatter() )

9     return render_to_response(
10        'viewfile.htm',
11        { 'Owner':      user,
12          'Repo':       repo,
13          'File':       filename,
14          'Contents':  contents
15        },
16        context_instance = RequestContext( request )
17    )
```

Beispiel: Template "viewfile.htm"

```
1  {% extends "base_nosidebar.html" %}
2  {% block content %}
3  <p>
4    Viewing
5    / {{ Owner.username }} / {{ Repo.name }} / {{ File }}
6  </p>
7  <div id="pygmented">
8    {{ Contents|safe }}
9  </div>
10 {% endblock %}
```

Beispiel: Und so sieht's aus

About Michael | About Ilico | About self | Blog | Mercurial | Impressum

Viewing / Svedrin / python-django-pdns / pdns/models.py [raw] (Rendered using Pygments Lexer: NumPy)

```
# -*- coding: utf-8 -*-

import time

from django.conf import settings
from django.db import models
from django.db.models import Q
from django.contrib.auth.models import User

from pdns.conf import settings as pdns_settings

class Supermaster(models.Model):
    """ SuperMasters are master servers which are allowed to push domains,
        even if they are not listed with us as a slave domain yet.

        For more info about SuperSlave operation, see:
        http://downloads.powerdns.com/documentation/html/generic-mysql-backends.html#AEN6061
    """

    ip = models.CharField( max_length=25 )
    nameserver = models.CharField( 'Name des Nameservers', max_length=255 )
    account = models.CharField( 'Accountname', max_length=40, unique=True )

    def __unicode__(self):
        return self.nameserver

    class Meta:
        db_table = 'supermasters'

class Domain(models.Model):
    """ Registers a domain with the server as a NATIVE, MASTER or SLAVE domain.

        In SuperSlave operation, the "account" field will specify which SuperMaster
        we received this domain from.
    """
```

DRY: Don't repeat yourself

- Wiederholungen vermeiden
- aus **einer** Spezifikation möglichst viel ableiten
- Spezifikation gegebenenfalls erweitern, aber **nie** kopieren

```
1 class Repository( models.Model ):
2     name          = models.SlugField()
3     owner         = models.ForeignKey( User )
4     basepath      = models.CharField( max_length=200 )
5     description   = models.TextField( blank=True )
6     writers       = models.ManyToManyField( User, blank=True )
```

- Django nutzt die Angaben in den Models für...

... das Datenbank-Schema (bspw. PGSQL)

```
1 BEGIN;
2 CREATE TABLE "django_mercurial_repository" (
3     "id" serial NOT NULL PRIMARY KEY,
4     "name" varchar(50) NOT NULL,
5     "owner_id" integer NOT NULL REFERENCES "auth_user" [...],
6     "basepath" varchar(200) NOT NULL,
7     "description" text NOT NULL
8 )
9 ;
10 CREATE TABLE "django_mercurial_repository_writers" (
11     "id" serial NOT NULL PRIMARY KEY,
12     "repository_id" integer NOT NULL REFERENCES [...],
13     "user_id" integer NOT NULL REFERENCES "auth_user" [...],
14     UNIQUE ("repository_id", "user_id")
15 )
16 ;
17 CREATE INDEX "django_mercurial_repository_name" ON [...];
18 CREATE INDEX "django_mercurial_repository_owner_id" ON [...];
19 COMMIT;
```

... Formulare und Validierung

repository ändern

⊖ Bitte die aufgeführten Fehler korrigieren.

⚠ Bitte ein gültiges Kürzel, bestehend aus Buchstaben, Ziffern, Unterstrichen und Bindestrichen, eingeben.

Name:

⚠ Dieses Feld ist zwingend erforderlich.

Owner: +

Description:

Writers: +

Halten Sie die Strg-Taste (⌘ für Mac) während des Klickens gedrückt, um mehrere Einträge auszuwählen.

... und kombiniert mit einer admin.py...

```
1 from django.contrib import admin
2 from models          import Repository

4 class RepoAdmin( admin.ModelAdmin ):
5     list_display    = [ 'owner', 'name', 'basepath' ];
6     list_filter     = [ 'owner' ];
7     search_fields  = [ 'name', 'basepath' ];
8     ordering       = [ 'owner', 'name' ];

10 admin.site.register( Repository, RepoAdmin );
```

... gibt's ein Admin-System:

Django-Verwaltung

Willkommen, **Svedrin**. [Passwort ändern](#) / [Abmelden](#)

Start > [Django_mercurial](#) > [Repositories](#)

repository zur Änderung auswählen

[repository hinzufügen](#) +

Suche

Aktion:

<input type="checkbox"/>	Owner ▾	Name	Basepath
<input type="checkbox"/>	Svedrin	python-django-pdns	/srv/standpy.de/repos/Svedrin/python-django-pdns

1 repository

Filter

Nach owner

Alle

- nico
- nomail-nospam.com
- ablage.p-roland-schwarzkopf.de
- Svedrin
- Info-zichi24.de

Inhalt

- 1 Webentwicklung - wo ist das Problem?
- 2 Was kann denn mehr?
- 3 O RLY?
- 4 Was ist alles möglich?**
- 5 Und jetzt?

Möglichkeiten

- “geht nicht” gibt’s nicht
- im Web bietet Django die wichtigsten Funktionen extrem elegant an
- außerhalb des Web kann Django problemlos integriert werden, z. B.
 - als Auth-Backends, z. B. für Webserver oder OpenVPN
 - zur Konfiguration von Diensten
 - als Munin-Plugin zum Erstellen von Graphen
 - in Shell-Scripts
 - und vieles mehr!
- was Django kann hängt vor allem vom Programmierer ab

Inhalt

- 1 Webentwicklung - wo ist das Problem?
- 2 Was kann denn mehr?
- 3 O RLY?
- 4 Was ist alles möglich?
- 5 Und jetzt?**

Einstieg

- Einstieg in Python und Django gleichzeitig ist möglich: Ich hab's selbst so gemacht. :-)
- Sehr empfehlenswert, weil 4 Seiten kurz und dennoch komplett: Das Django-Tutorial (auch auf Deutsch)
- Python-Einstieg:
 - Beginner's Guide
 - ohne Programmiererfahrung
 - mit Programmiererfahrung
- Usenet: de.comp.lang.python
- IRC: #python und #django auf irc.freenode.net