Django – The Web framework for perfectionists with deadlines.

Julian Moritz, public@julianmoritz.de

28. Februar 2008

Einführung

- Rasantes Wachstum im Internet, Streben nach Dynamik
- Einzellösungen für dynamische Teile
- Für umfangreichere Seiten wird eine Portalsoftware eingesetzt

Einsatz eines CMS

- Da es meist zu aufwändig ist, Benutzerverwaltung, Rechtemanagement, Rollenverteilung usw selbst zu erstellen, wird ein CMS eingesetzt
- Auch haben Erfahrungen gezeigt, dass selbstentworfene Designs auf Grund von Zeitmangel schlecht durchdacht oder schlecht umgesetzt werden
- Vorteile liegen klar auf der Hand: Einfach zu installieren, erprobte Technik, meist großer Einsatzbereich durch viele Zusatzmodule
- Die Nachteile zeigen sich erst später: Problem mit Lizenzen, Problem mit Sicherheitslücken, schwer anzupassen, nicht auf die eigenen Bedürfnisse zugeschnitten

Warum Django?

- **Rapid** Web Development
- Arbeiten nach dem DRY-Prinzip:

  Every piece of knowledge must have a single, unambiguous, authoritative representation within a system.

- Vorteile von Python nutzen!

Aufbau eines einfachen Models

```
1 #ein einfaches Model
2 from django.db import models
3
4 class Person(models.Model):
5     name = models.CharField(max_length=250)
6     alter = models.IntegerField()
7     angelegt = models.DateTimeField(auto_now_add=True)
8     geschlecht = models.CharField(max_length=1,choices = (("m","Mann"),("w","Frau")) )
9
10    class Admin:
11        pass
```

```
1 # Personen anlegen
2 >>> person_1 = Person(name="Hans Meier",alter=30,geschlecht="m")
3 >>> person_1.save()
4 >>> person_2 = Person(name="Petra Meier",alter=20,geschlecht="w")
5 >>> person_2.save()
6 >>> person_3 = Person(name="Petra Musterfrau",alter=25,geschlecht="w")
7 >>> person_3.save()
8 # Personen aus der Datenbank holen
9 >>> personen = Person.objects.all()
10 >>> for person in personen:
11 ...     print person.name
12 Hans Meier
13 Petra Meier
14 Petra Musterfrau
15 >>> quit()
```

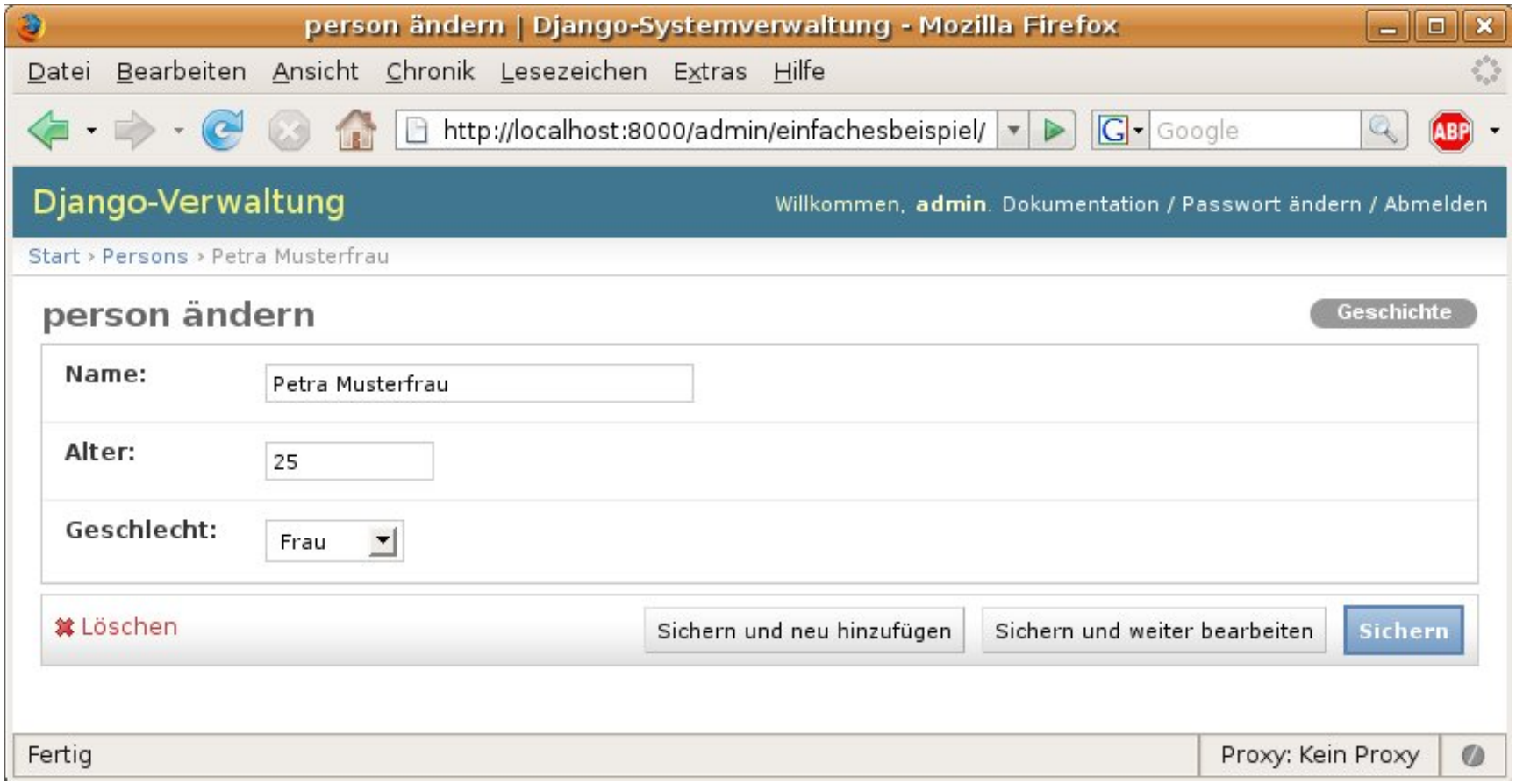
Arbeiten mit einem einfachen Model

```
1 # Personen sortieren und filtern
2 >>> personen = Person.objects.filter(name__exact="Hans Meier")
3 >>> for person in personen:
4 ...     print person.name
5 Hans Meier
6 >>> personen = Person.objects.all().order_by("alter")
7 >>> for person in personen:
8 ...     print person.name + " ist " + str(person.alter) + " Jahre alt"
9 Petra Meier ist 20 Jahre alt
10 Petra Musterfrau ist 25 Jahre alt
11 Hans Meier ist 30 Jahre alt
12 >>> personen = Person.objects.filter(name__contains="Petra").order_by("-alter")
13 >>> for person in personen:
14 ...     print person.name
15 Petra Musterfrau
16 Petra Meier
17 >>> personen = Person.objects.filter(name__startswith="Petra").filter(alter__lte=22)
18 >>> for person in personen:
19 ...     print person.name
20 Petra Meier
21 >>> quit()
```

Wir können Ergebnismengen sortieren, filtern, vereinigen, ausschließen, zählen und das letzte Objekt ausgeben.

Ein einfaches Model im Admininterface

Ein einfaches Model im Admininterface



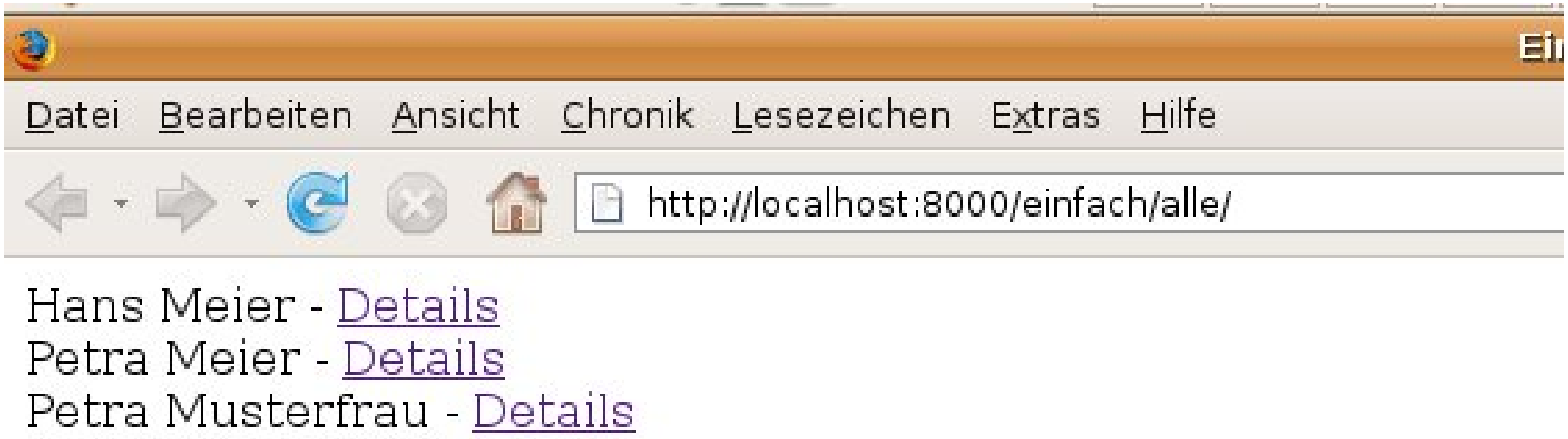
Ein einfacher View

```
1
2 from einfachesbeispiel.models import Person
3 from django.shortcuts import render_to_response
4
5 def allepersonen(request):
6     personen = Person.objects.all()
7     return render_to_response("einfach_alle_personen.html",{"personen":personen})
```

Ein einfaches Template

```
1 <html>
2  <head>
3   <title>Einfaches Beispiel</title>
4  </head>
5  <body>
6   {% for person in personen %}
7    {{person.name}} - <a href="/einfach/person/{{person.id}}/">Details</a><br>
8   {% endfor %}
9  </body>
10 </html>
```

Screenshot


Datei Bearbeiten Ansicht Chronik Lesezeichen Extras Hilfe
http://localhost:8000/einfach/alle/
Hans Meier - Details
Petra Meier - Details
Petra Musterfrau - Details

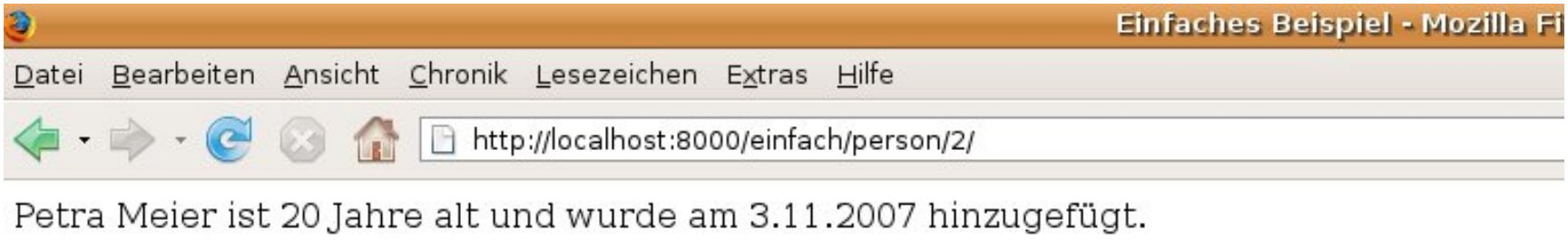
Noch ein View

```
1 from einfachesbeispiel.models import Person
2 from django.shortcuts import render_to_response, get_object_or_404
3
4 def eineperson(request, id):
5     person = get_object_or_404(Person, id__exact=id)
6     return render_to_response("einfach_eine_person.html",{"person":person})
```

Ein weiteres Template

```
1 <html>
2  <head>
3   <title>Einfaches Beispiel</title>
4  </head>
5  <body>
6   {{person.name}} ist {{person.alter}} Jahre alt. <br>
7    Sie wurde am {{person.angelegt|date:"j.n.Y"}} hinzugef&uuml;gt.
8  </body>
9 </html>
```

Screenshot


Einfaches Beispiel - Mozilla Fi
Datei Bearbeiten Ansicht Chronik Lesezeichen Extras Hilfe
http://localhost:8000/einfach/person/2/
Petra Meier ist 20 Jahre alt und wurde am 3.11.2007 hinzugefügt.

Kurzes Intermezzo: Wie kommt man von der URL zum View?

- Lösung: In der urls.py wird von einer URL auf die dazugehörige Funktion gezeigt!
- (r“^einfach/alle/$“, “einfachesbeispiel.views.allepersonen“)
- (r“^einfach/person/(?P<id>\d+)/$“, “einfachesbeispiel.views.eineperson“)
- Endlich keine hässlichen URLs mehr:
- http://www.stura.uni-leipzig.de/index.php?id=259&tx_ttnews[tt_news]=253&tx_ttnews[backPid]=21&cHash=d640edb53b
- Schön: http://www.mahner.org/weblog/kategorie/django_und_python/
- (r“^person.php?id=(?P<id>\d+)$“, “einfachesbeispiel.views.eineperson“)

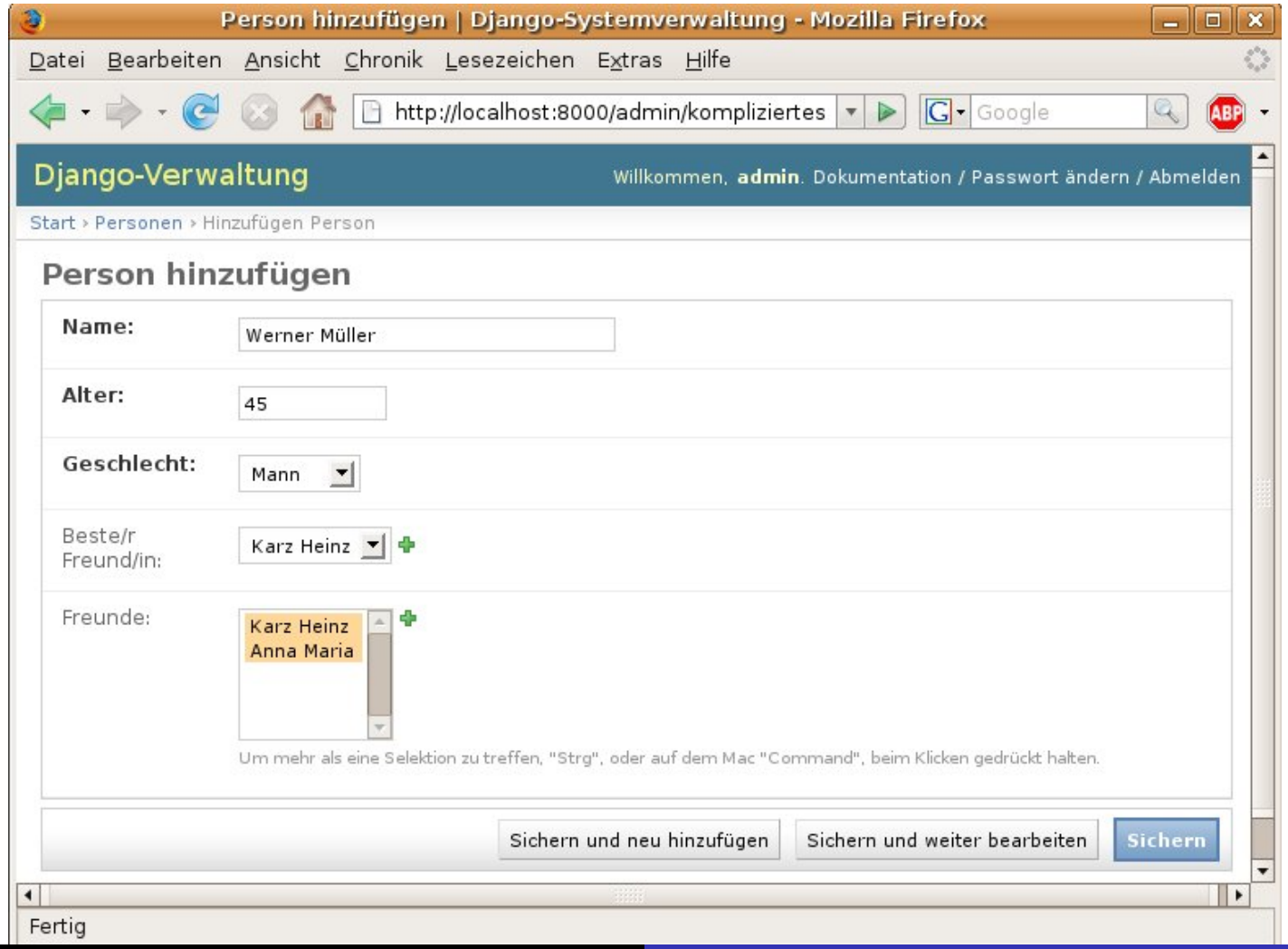
Beispiel für DRY an benannten URL-Patterns

- Problem: Eine URL steht in der urls.py, wird eventuell in Views/Templates verwendet.
- Lösung: url(r“^einfach/alle/“, “einfachesbeispiel.views.allepersonen“, name=“einfach-alle-personen“)
- Im Template: {% url einfach-alle-personen %}
- Im Python-Code: reverse(“einfach-alle-personen“)
- Im Model mit dem permalink-Dekorator
- Mit Parametern:
  - {% url zeige-person id=person.id %}
  - reverse(“zeige-person“,{“id“:person.id})

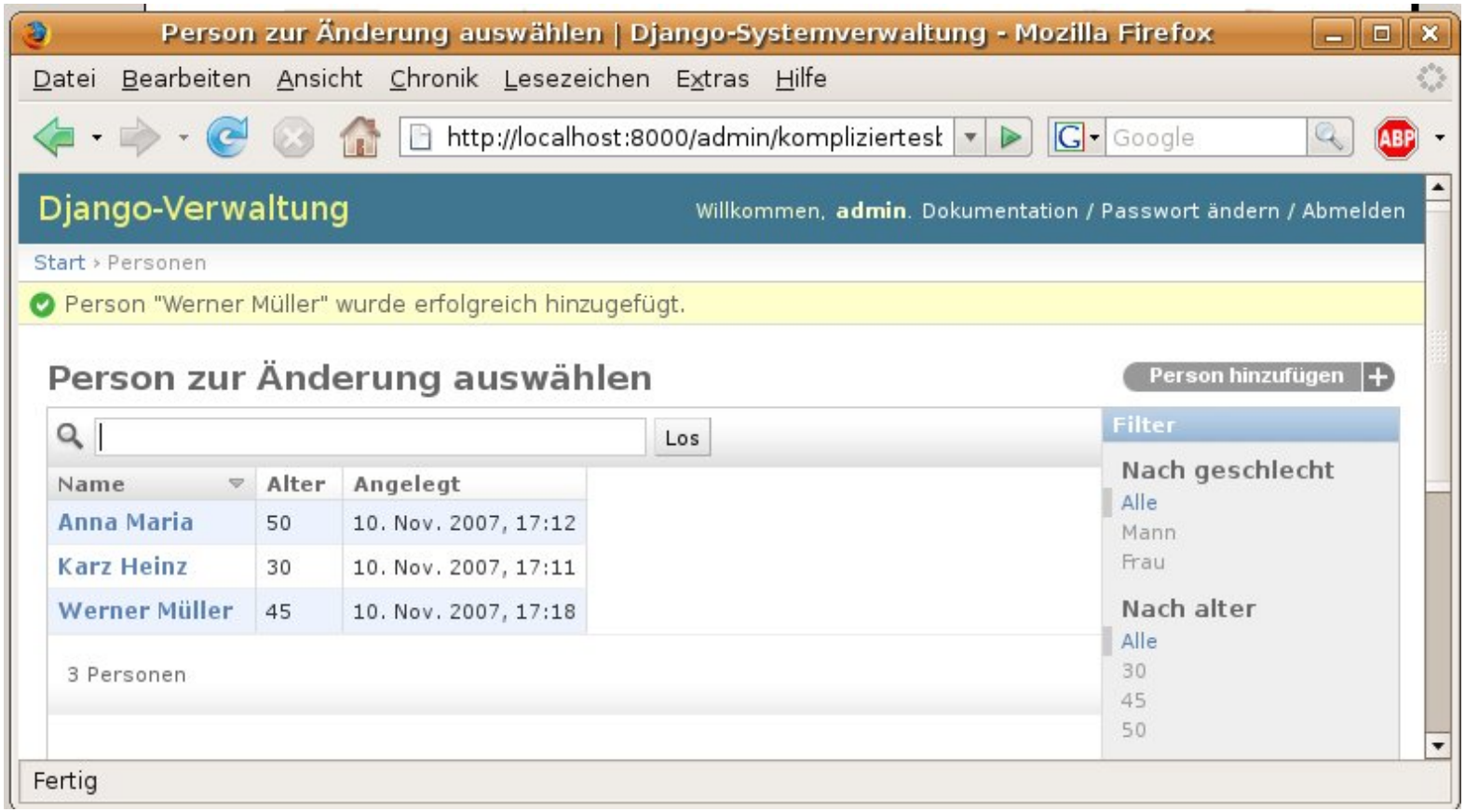
Erstellen eines komplizierteren Models

```
1  #ein komplizierteres Model
2  from django.db import models
3
4  class Person(models.Model):
5      name = models.CharField(max_length=250)
6      alter = models.IntegerField()
7      angelegt = models.DateTimeField(auto_now_add=True)
8      geschlecht = models.CharField(max_length=1,choices = (("m","Mann"),("w","Frau")) )
9      bester_freund = models.ForeignKey("Person",blank=True,null=True,verbose_name="Beste/
10     freunde = models.ManyToManyField("Person",blank=True,null=True)
11
12     class Admin:
13         list_display = ("name","alter","angelegt",)
14         list_filter = ("geschlecht","alter",)
15         ordering = ("name","alter",)
16         search_fields = ("name",)
17
18     class Meta:
19         verbose_name = "Person"
20         verbose_name_plural = "Personen"
21
22     def __unicode__(self):
23         return self.name
```

Ein kompliziertes Model im Admininterface - hinzufügen



Ein kompliziertes Model im Admininterface - Übersicht



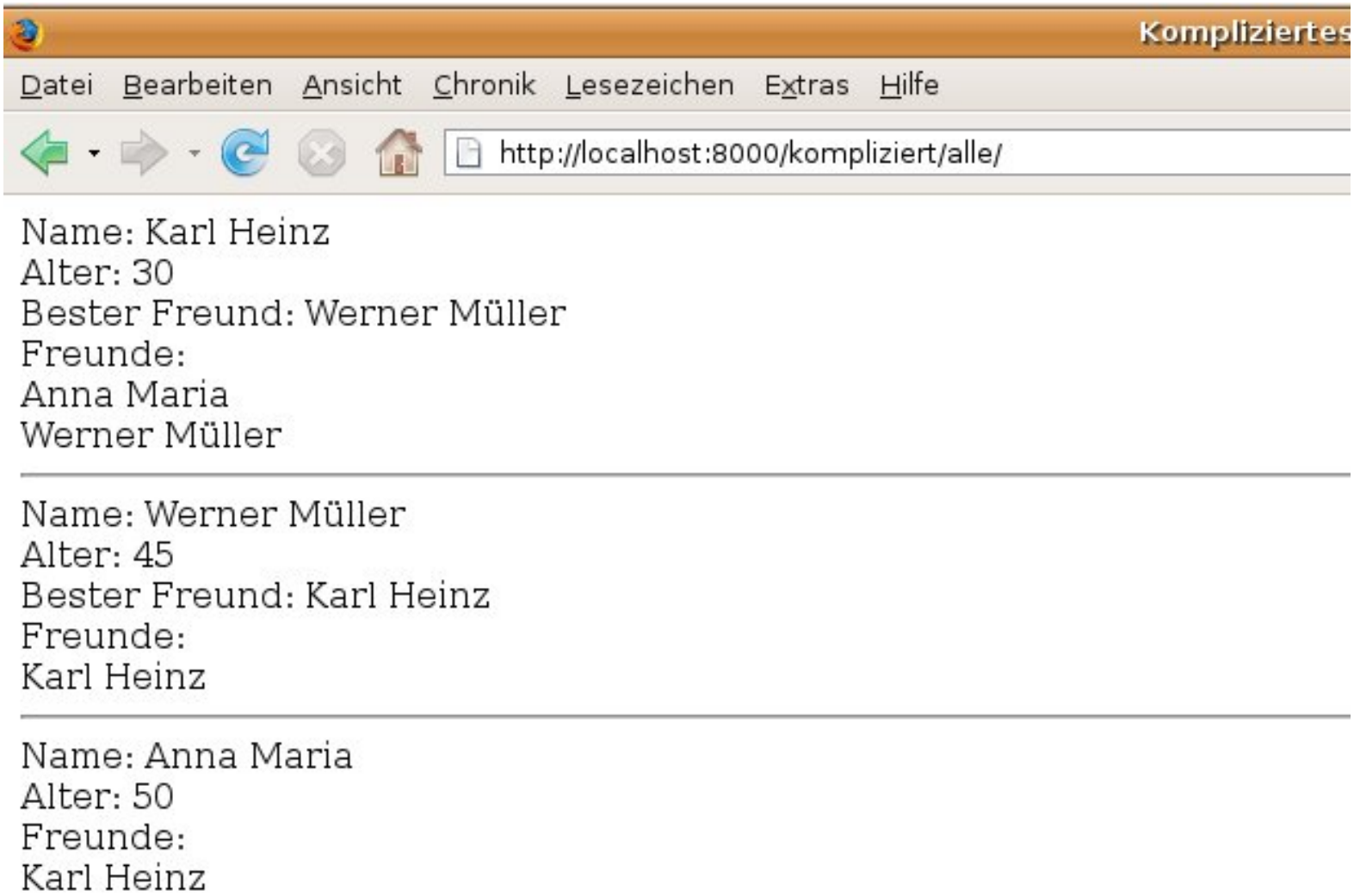
Ein generischer View

```
1  from django.conf.urls.defaults import *
2  from kompliziertesbeispiel.models import Person
3
4  kompliziert_alle = {  "queryset": Person.objects.all(),
5                "template_name":"kompliziert_alle_personen.html" }
6
7  urlpatterns = patterns("",
8      (r"^kompliziert/alle/",
9      "django.views.generic.list_detail.object_list",
10     kompliziert_alle))
```

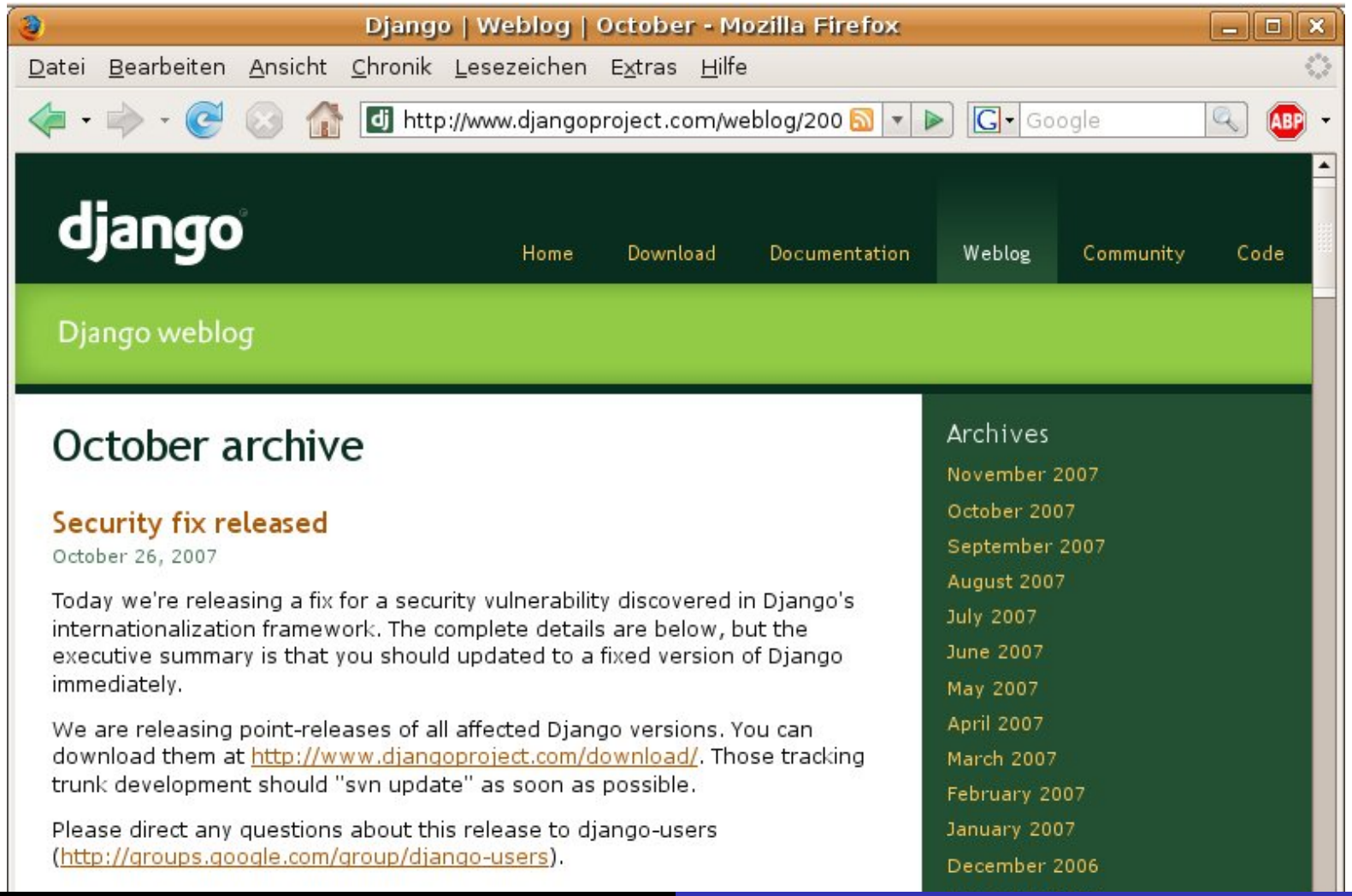
Ein kompliziertes Template

```
 1 <html>
 2 <head>
 3 <title>Kompliziertes Beispiel</title>
 4 <body>
 5 {% for person in object_list %}
 6 Name: {{person.name}}<br>
 7 Alter: {{person.alter}}<br>
 8
 9 {% if person.bester_freund %}
10 Bester Freund: {{person.bester_freund.name}}<br>
11 {% endif %}
12
13 {% if person.freunde.all %}
14 Freunde: <br>
15
16 {% for freund in person.freunde.all %}
17 {{freund.name}}<br>
18 {% endfor %}
19
20 {% endif %}
21
22 <hr>
23 {% endfor %}
24 </body>
25 </html>
```

Screenshot


Kompliziertes
Datei Bearbeiten Ansicht Chronik Lesezeichen Extras Hilfe
http://localhost:8000/kompliziert/alle/
Name: Karl Heinz
Alter: 30
Bester Freund: Werner Müller
Freunde:
Anna Maria
Werner Müller
Name: Werner Müller
Alter: 45
Bester Freund: Karl Heinz
Freunde:
Karl Heinz
Name: Anna Maria
Alter: 50
Freunde:
Karl Heinz

Screenshot datumsbasierter generischer Views



Caching

- Seitenweises Caching: CACHE_BACKEND = “memcached://127.0.0.1:11211/“ in der settings.py
- Caching je View: @cache_page(60 * 15) als decorator für den View
- Low-Level-Cache:

```
1 >>> cache.set("'mein_schluessel"', "'Hallo Welt!"', 30)
2 >>> cache.get("'mein_schluessel"', "'Falls nicht vorhanden, nimm dies!"')
```

Benutzer registrieren, Gruppen, Rechte

- Das request-Objekt hat ein Attribut user, welches ein Benutzer-Objekt ist. Es gibt an, ob der Benutzer angemeldet ist und falls ja hat es Attribute wie Vorname, Nachname, Benutzername, Passwort, Gruppen usw.
- Zusätzlich auch Methoden wie has_perm(perm), get_all_perms() und is_authenticated()
- Falls wir Views nur angemeldeten Nutzern zugänglich machen wollen, reicht der @login_required Dekorator für den jeweiligen View
- Falls der View nur mit einem bestimmten Recht aufgerufen werden darf, reicht der @user_passes_test(lambda u: u.has_perm("person.can_edit")) Dekorator für den View

Internationalisierung

- Es wird die Funktion ugettext_lazy genutzt, abgekürzt auch _
- Im Model: name = models.CharField(max_length=150,verbose_name=_("person name verbose_name"))
- Im Template: <title >{% trans "Das ist der Titel!" %}</title >
- Nach dem Aufrufen des i18n-Skriptes haben wir nun eine sprachspezifische Datei, die folgendes enthält:

```
#: vortrag/einfachesbeispiel/models.py:23
msgid "person name verbose_name"
msgstr ""
#: vortrag/templates/einfaches_template.html:8
msgid "Das ist der Titel!"
msgstr ""
```

Hier können wir nun die gewünschten Werte für die jeweilige Sprache eintragen. Nach dem Kompilieren sind diese dann verfügbar.

Was ist noch möglich?

- Formular-Verarbeitung mit OOP
- RSS-Feeds und Atom-Feeds mit wenigen Zeilen Code
- Template-Vererbung zur HTML-Minimierung
- Und vieles mehr!

Quellen und Literatur

- http://www.djangoproject.com/
- http://www.djangobook.com/
- http://www.djangosites.org/
- Django. Einführung in das Python Web Framework, Open Source Press.
- The Definitive Guide to Django, Apress.
- Python in a Nutshell, O'Reilly.