Prototyp -
Django mit MongoDB
Projekt: DigitalSchoolNotes

Projekt Team: Adler, Brinnich, Hohenwarter, Karic, Stedronsky

Version 2.0 **12.10.2015** **Status: [RELEASE]**

	Datum	Name	Unterschrift
Erstellt	05.10.2015	Selina Brinnich	
Geprüft	12.10.2015	Niklas Hohenwarter	
Freigegeben			
Git-Pfad: */doc/evaluation*		**Dokument:** *prototype_django_mongodb.doc*	

Inhaltsverzeichnis

Version	Datum	Status	Bearbeiter	Kommentar
0.1	2015-10-05	Erstellt	Selina Brinnich	Erstellt
1.0	2015-10-07	QA	Niklas Hohenwarter	QA
1.1	2015-10-12	Bearbeitet	Niklas Hohenwarter	Verbesserungsvorschläge der Abnahme durchgeführt

1 Einleitung

Ein Prototyp, in welchem Django zusammen mit MongoDB verwendet wird soll erstellt werden. Der Prototyp soll die Funktionen zum Auslesen, Erstellen, Bearbeiten, sowie Löschen von Daten in der Datenbank implementieren. Dazu wird eine kleine Applikation erstellt, die eine Liste an Lieblingsspeisen anzeigen soll. Es sollen beliebig viele neue Speisen zur Liste hinzugefügt werden können. Für jede Speise soll gevotet werden können (+1 Punkt oder -1 Punkt). Speisen sollen jederzeit aus der vorhandenen Liste entfernt werden können.

2 Installation und Konfiguration

Zunächst wird Django mit folgendem Befehl installiert:

```
apt-get install python3-django
```

Anschließend wird MongoDB folgendermaßen installiert:

```
sudo apt-key adv --keyserver keyserver.ubuntu.com --recv 7F0CEB10
echo "deb http://repo.mongodb.org/apt/debian wheezy/mongodb-org/3.0 main" |
sudo tee /etc/apt/sources.list.d/mongodb-org-3.0.list
sudo apt-get update
sudo apt-get install -y mongodb-org
sudo service mongod start
```

Mit folgendem Befehl wird nun ein neues Projekt namens “prototype” erstellt:

```
django-admin startproject prototype
```

Nun muss noch MongoEngine, das Verbindungsstück zwischen Django und MongoDB, installiert werden. Dabei sollte die Version 0.9.0 verwendet werden, da die neueste Version 0.10.0 aktuell einen Bug hat, durch welchen man sich nicht mehr mit einer Datenbank verbinden kann:

```
pip3 install mongoengine==0.9.0
```

Um Django mitzuteilen, dass MongoDB im Hintergrund verwendet werden soll, müssen im File *prototype/settings.py* folgende Zeilen auskommentiert bzw. hinzugefügt werden:

```
#DATABASES = {
#    'default': {
#        'ENGINE': 'django.db.backends.sqlite3',
#        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
#    }
#}

import mongoengine

DATABASES = {
    'default': {
        'ENGINE': '',
    },
}

SESSION_ENGINE = 'mongoengine.django.sessions'
```

```
_MONGODB_HOST = 'localhost' #Hostname
_MONGODB_NAME = 'testy' #DB-Name
_MONGODB_DATABASE_HOST = 'mongodb://%s' % (_MONGODB_HOST)

mongoengine.connect(_MONGODB_NAME, host=_MONGODB_DATABASE_HOST)
```

Um eigene html-Templates erstellen zu können muss außerdem noch ein Ordner "templates" erstellt werden und folgende Zeilen im File *prototype/settings.py* hinzugefügt werden:

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, 'templates')],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
```

3 Read

Für die Anzeige wird zunächst ein Model im File *prototype/models.py* geschrieben:

```
from mongoengine import *
class Food(Document):
    name = StringField(max_length=50)
    votes = IntField(default=0)
```

Dieses Model repräsentiert einen Datensatz in der Datenbank. Jedes Element *Food* hat hier zwei Attribute, *name* und *votes*, wobei der Name maximal 50 Zeichen lang sein darf und *votes* den Wert 0 erhält, sollte kein anderer Wert vorhanden sein.

Nun muss ein Template *foodlist.html* im Ordner *templates* erstellt werden:

```
<h2>Favorite Food List</h2>
<ul>
{% for food in foodlist %}
    <li>{{ food.name }}<br />{{ food.votes }}
    </li>
{% endfor %}
</ul>
```

Diesem Template wird nachher eine Liste an Speisen in der Variable *foodlist* übergeben. Mithilfe der for-Schleife wird diese Liste durchgegangen und jedes einzelne Element in ein Listenelement geschrieben. Hier kann mit z.B. *food.name* auf das Attribut *name* des Objektes *Food* zugegriffen werden.

Anschließend muss eine Methode zum Anzeigen der Liste im File *prototype/views.py* erstellt werden:

```
from prototype.models import Food
from django.http import HttpResponseRedirect, HttpResponse
from django.shortcuts import get_object_or_404, render
from django.core.urlresolvers import reverse
def show_food(request):
    foodlist = Food.objects()
    context = {'foodlist': foodlist}
    return render(request, 'foodlist.html', context)
```

In der Methode werden zunächst alle *Food*-Elemente aus der Datenbank abgefragt und in *foodlist* gespeichert. Anschließend wird ein Context erstellt, der diese Liste enthält. Zuletzt wird das Template *foodlist.html*, welches vorher erstellt wurde, gerendert. Dabei wird der Context übergeben, sodass das Template nun unsere Speisen-Liste zur Verfügung hat und alle Elemente auf der Webseite anzeigen kann.

Als letztes muss noch eine Route für die Anzeige-Funktion erstellt werden. Dazu wird im File *prototype/urls.py* folgender Code eingefügt:

```
...
    url(r'^$', 'prototype.views.show_food', name='show_food'),
...
```

Hier wird eine neue Route für '/' definiert. Sobald ein Benutzer auf diesen Pfad zugreift, wird die Methode 'prototype.views.show_food', die im vorherigen Schritt erstellt wurde, aufgerufen. Diese rendert anschließend das Template, welches dann auf der Webseite angezeigt wird:

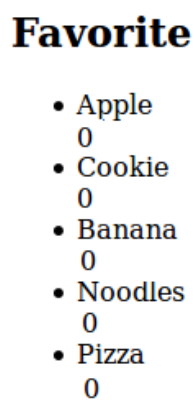

Figure 1 Prototyp Step1

4 Create

Um nun zusätzlich eine Erstell-Funktion einzubauen, muss das Template folgendermaßen erweitert werden:

```
<h2>Insert New Food</h2>
{% if error_message %}<p><strong>{{ error_message }}</strong></p>{% endif
%}
<form action="{% url 'insert_food' %}" method="post">
    {% csrf_token %}
    Food name: <input type="text" name="name" value="" />
```

```
    <input type="submit" value="Insert" />
</form>
```

Hier wird ein neues Formular erstellt. Dieses wird an eine Route namens ‘insert_food’, die im Folgenden noch erstellt wird, geschickt. Das Formular enthält ein Textfeld zum Eingeben des Namen der neuen Speise und einen Submit-Button.

Anschließend wird eine neue Methode im File *prototype/views.py* erstellt:

```
def insert_food(request):
    newfood = Food(name=request.POST['name'],votes=0)
    newfood.save()
    return HttpResponseRedirect(reverse('show_food'))
```

Diese Methode erstellt zunächst ein neues Food-Objekt mit dem Namen, den der Benutzer im Formular eingegeben hat. Anschließend wird dieses Objekt mit *.save()* gespeichert und damit in die Datenbank gespeichert. Zuletzt wird auf die Anzeigeseite über die definierte Route *show_food* zurückgeleitet.

Zuletzt muss noch eine neue Route im File “prototype/urls.py” erstellt werden:

```
...
url(r'^insert/', 'prototype.views.insert_food', name='insert_food'),
...
```

Die Route hier wird aufgerufen, sobald der Benutzer eine neue Speise erstellt und das Formular abschickt. Die Daten des Formulars werden anschließend über die Route zur Methode ‘prototype.views.insert_food’ weiergeleitet, in der die neue Speise in die Datenbank eingefügt wird und der Benutzer auf die Haupt-Anzeigeseite zurückgeleitet wird, in der nun die neue Speise stehen sollte:

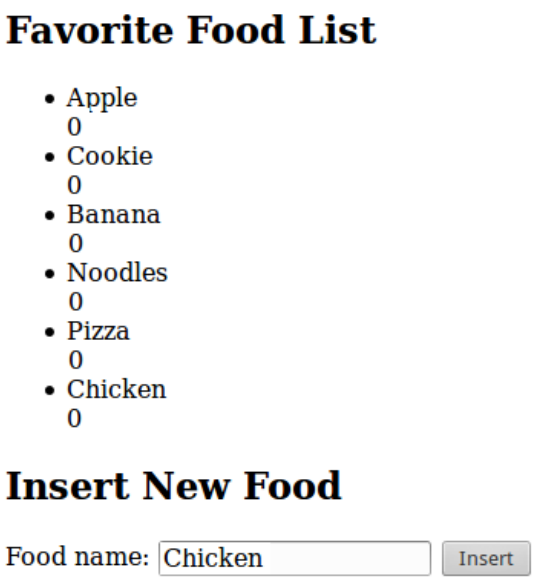

Figure 2 Prototyp Step 2

5 Update

Um den Benutzer für seine Lieblingsspeise voten zu lassen, muss zunächst wieder das Template angepasst werden:

```
<h2>Favorite Food List</h2>
<ul>
{% for food in foodlist %}
    <li>{{ food.name }}<br />{{ food.votes }}
        <a href="{% url 'vote' foodname=food.name votetype='+' %}">+1</a>
        <a href="{% url 'vote' foodname=food.name votetype='-' %}">-1</a>
    </li>
{% endfor %}
</ul>
```

Hier wird ein neuer Link für +1 und -1 Punkt bei jeder Speise erstellt. Dieser Link wird über die Route namens 'vote' geleitet. Zudem werden der Route zwei Parameter übergeben: *name*, um die Speise zu identifizieren, für die gevotet werden soll, und *votetype*, um zu definieren, ob positiv oder negativ gevotet wurde.

Nun muss wieder eine neue Methode im File *prototype/views.py* erstellt werden:

```
def vote(request, foodname, votetype):
    food = Food.objects(name=foodname).first()
    if votetype == '+':
        food.votes += 1
    else:
        food.votes -= 1
    food.save()
    return HttpResponseRedirect(reverse('show_food'))
```

Hier werden die beiden Parameter von vorhin der Methode übergeben. Die Methode sucht dann das Objekt mit dem entsprechenden Namen aus der Datenbank und speichert das Objekt in *food.* Anschließend wird, je nachdem ob positiv oder negativ gevotet wurde, ein Punkt zu *votes* hinzugefügt bzw. abgezogen und das Objekt wieder abgespeichert. Zuletzt wird wieder auf die Anzeigeseite zurückgeleitet.

Als Letztes muss noch eine neue Route erstellt werden:

```
...
url(r'^(?P<foodname>[A-Za-z]+)/vote/(?P<votetype>[+-])/',
'prototype.views.vote', name='vote'),
...
```

In dieser Route werden zwei Parameter verarbeitet: *foodname* und *votetype*, wobei der Parameter *foodname* nur aus Buchstaben und *votetype* nur aus einem Zeichen '+' oder '-' bestehen darf. Ein Beispiel-Link, der diesem Pattern entspricht wäre '*/Cookie/vote/+/*'.

Nun sollte die Anzeigeseite folgendermaßen aussehen:

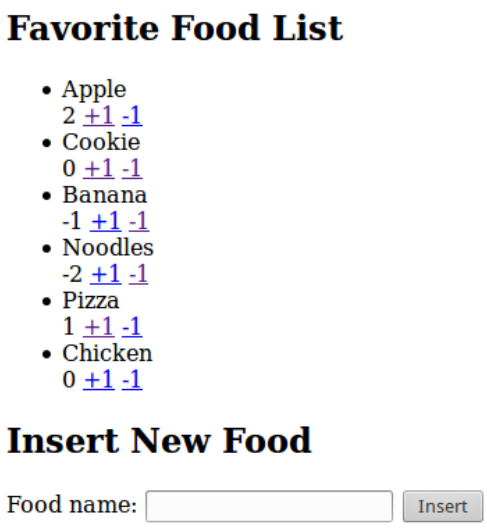

Figure 3 Prototyp Step 3

6 Delete

Um zuletzt noch eine Löschen-Funktion einzubauen, wird das Template folgendermaßen bearbeitet:

```
<h2>Favorite Food List</h2>
<ul>
{% for food in foodlist %}
    <li>{{ food.name }}<br />{{ food.votes }}
        <a href="{% url 'vote' foodname=food.name votetype='+' %}">+1</a>
        <a href="{% url 'vote' foodname=food.name votetype='-' %}">-1</a>
        <a href="{% url 'delete_food' foodname=food.name %}">Remove
Food</a>
    </li>
{% endfor %}
</ul>
```

Nun haben wir noch einen Link mehr pro Element. Dieser wird über die Route ‘delete_food’ geleitet. Dabei wird wieder der Name der entsprechenden Speise zur Identifikation übergeben.

Anschließend wird wieder eine neue Methode im File *prototype/views.py* erstellt:

```
def delete_food(request, foodname):
    food = Food.objects(name=foodname).first()
    food.delete()
    return HttpResponseRedirect(reverse('show_food'))
```

Diese Methode erhält als Parameter den Namen einer Speise, sucht sich die entsprechende Speise aus der Datenbank und speichert das Objekt in *food*. Anschließend wird das Objekt mit *.delete()* aus der Datenbank gelöscht. Zuletzt wird wieder auf die Anzeigeseite zurückgeleitet.

Zu guter Letzt muss nun noch eine letzte Route erstellt werden:

```
...
url(r'^(?P<foodname>[A-Za-z]+)/delete/', 'prototype.views.delete_food',
name='delete_food'),
...
```

Hier wird wieder ein Parameter *foodname* in der URL verarbeitet. Ein Beispiel-Link für diese Route wäre *'/Banana/delete/'*.

Nun sollte die Seite folgendermaßen aussehen:

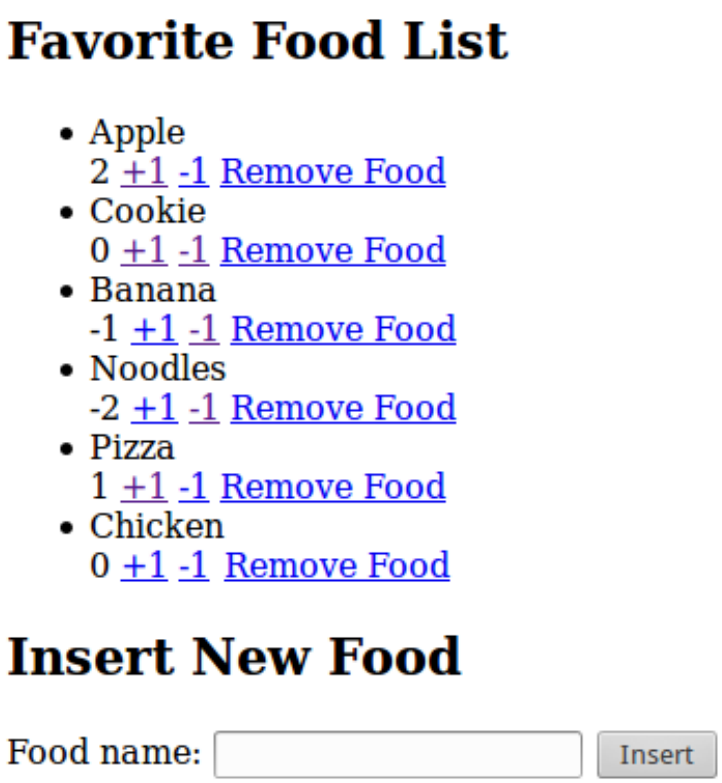

Figure 4 Prototyp Step 4

7 Abbildungsverzeichnis