Das Django Web-Framework dargestellt anhand des Praktischen Beispieles eines Inventarisierungssystemes

Clemens Dautermann

2. Januar 2019 bis 30. Januar 2019

Inhaltsverzeichnis

1 Einleitung

Diese Facharbeit soll einen grundlegenden Überblick über die wichtigsten Funktionen des Django Web-Frameworks geben.
Das Django Web-Framework ist ein größtenteils in Python geschriebenes[1] Framework zum entwickeln von Webservern. Es ist aufgrund seiner ausgeprägten Modularität und der Existenz einer Vielzahl von Datenbanktreibern besonders gut für die Entwicklung von Webservern geeignet, die eine Datenbank erfordern.
Django stellt eine grundlegende Struktur für die Entwicklung zur Verfügung. So zum Beispiel:

- Eine settings.py Die genutzt werden kann um Konfigurationsmöglichkeiten zentral zu bündeln
- Eine library um einfache Zugriffe auf Datenbanken zu tätigen und sogenannte Models um Datenbankobjekte zu verwalten
- Ein Routingsystem um eine einfachere Verwaltung von Urls zu gewährleisten
- Eine Grundstruktur, die Modularität unterstützt und das einfache Installieren oder Entfernen von sogenannten ”Apps” ermöglicht

Es ist also kaum notwendig, jedoch durchaus möglich, als Entwickler noch SQL zu schreiben wenn man mit dem Django Web-Framework entwickelt.

2 Struktur

Ein typischer Django Server ist aus sogenannten ”Apps” aufgebaut. Diese werden entweder vom Entwickler selber geschrieben oder können via pip (dem Python Paket Manager) installiert werden. Ein standard Verzeichnisaufbau ist in Abbildung 1 dargestellt.

[1] Offizielle Django GitHub Seite https://github.com/django/django

```
server
├── manage.py
├── db.sqlite3
├── server
│   ├── __init__.py
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
└── app1
    ├── __init__.py
    ├── admin.py
    ├── apps.py
    ├── forms.py
    ├── models.py
    ├── tests.py
    ├── urls.py
    ├── views.py
    └── migrations
        └── 0001_initial.py
```

Abbildung 1: Die typische Verzeichnisstruktur eines Django Servers

2.1 Erstellung

Ein Django Projekt kann mit dem Befehl $ django-admin startproject server initialisiert werden. Dadurch wird folgende Ordnerstruktur erstellt:

```
server
├── manage.py
└── server
    ├── __init__.py
    ├── settings.py
    ├── urls.py
    └── wsgi.py
```

Abbildung 2: Verzeichnisstruktur, die der $ django-admin startproject server Befehl erzeugt

2.2 manage.py

Die manage.py wird, wie der Name schon sagt, verwendet um den Server zu verwalten. Mit Hilfe der manage.py können beispielsweise Migrierungen an der Datenbank erstellt werden, Datenbanknutzer erstellt werden oder der Testserver zur Entwicklung kann gestartet werden. Die gleiche Funktionalität stellt auch der django-admin Befehl zur Verfügung[2].

[2]Django Dokumentation https://docs.djangoproject.com/en/2.1/ref/django-admin/

2.3 db.sqlite3

In dieser Datei wird die SQL Datenbank gespeichert, die der Server nutzt. Sie wird automatisch erstellt. Es können jedoch auch andere Datenbanken, wie zum Beispiel MongoDB oder eine extern gehostete Datenbank, verwendet werden.

2.4 server/server

2.4.1 __init__.py

Diese Datei befindet sich im Wurzelverzeichnis jeder App. Sie macht für Python erkennbar, dass es sich bei dem Inhalt dieses Ordners um ein Python Modul handelt. Somit kann die App einfach geteilt und von anderen Nutzern verwendet werden.[linenos, frame=lines, framesep=2mm]Python

2.4.2 settings.py

In dieser Datei befinden sich die Einstellungen für den Django Server. Mit ihrer Hilfe werden Zeitzone, Sprache, Datenbankkonfiguration und viele andere Konfigurationen verwaltet. Man kann sie auch verwenden um eigene Einstellungsmöglichkeiten anzubieten. Dafür definiert man eine Konstante (in Python typischerweise durch Großbuchstaben ausgedrückt) und einen Wert. Zum Beispiel
LOGIN_REDIRECT_URL = "/". Im Falle dieses Projektes wurde beispielsweise die Konstante LOGFILE = 'serverlog.log' definiert um zentral auf die Logdatei zugreifen zu können. Auf die in der settings.py definierten Werte kann aus jeder App zugegriffen werden, indem unter Benutzung der Anweisung
`from django.conf import settings` diese importiert wird. Anschließend kann via `file = settings.LOGFILE` beispielsweise auf die Konstante LOGFILE zugegriffen werden.

2.4.3 urls.py

Diese Datei ist der erste und wichtigste Teil des Django Routing Systems. Über sie werden die grundlegenden URL Strukturen des Servers definiert. Sie importiert urls.py Dateien aus anderen Apps und definiert wie auf eine Bestimmte URL reagiert werden soll. Im Falle dieses Projektes sieht sie folgendermaßen aus:

```
1         from django.contrib import admin
2         from django.urls import path, include
3
4         urlpatterns = [
5         path('admin/', admin.site.urls),
6         path('accounts/', include('django.contrib.auth.urls')),
7         path('', include('usermanager.urls')),
8         path('add/', include('objectadder.urls')),
9         path('list/', include('objectlister.urls')),
10        path('settings/', include('settingsapp.urls')),
11        ]
```

Hier werden zuerst Pakete für das admin-Interface und Pakete für das URL-Routing importiert. Anschließend wird eine Liste urlpatterns definiert. Diese enthält alle URL Pfade. include() Importiert dabei die urls.py Dateien aus den anderen apps.

2.4.4 wsgi.py

Diese Datei stellt ein ”application” genanntes WSGI Objekt zur Verfügung. Es wird zum starten des Testservers genutzt. In Verbindung mit ”mod_wsgi” kann auch Apache dieses Objekt nutzen.[3]

2.5 server/app1

Dieses Verzeichnis enthält alle Dateien, die eine App ausmachen. In diesem Fall heißt die App beispielhaft ”app1”. Die Dateien, die in Abb 1 erscheinen, jedoch hier nicht erwähnt werden, haben die selbe Funktion wie die gleichnamigen Dateien im ”server/server” Ordner.

2.5.1 admin.py

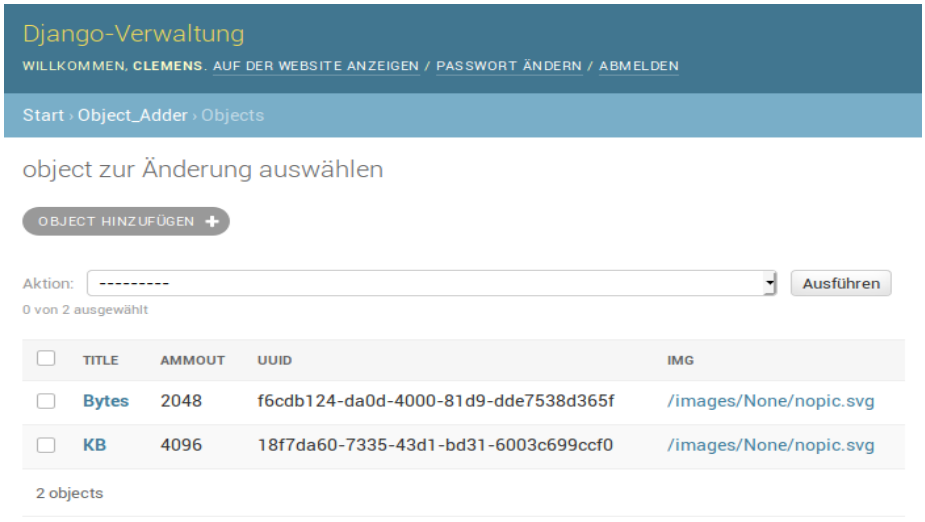
Die admin.py dient dazu das Django Admin Interface zu konfigurieren. In ihr werden die Models registriert, die auf der admin Seite zu sehen sind, und wie diese dargestellt werden sollen. Sie sieht beispielsweise in der ”object_adder” app folgendermaßen aus:

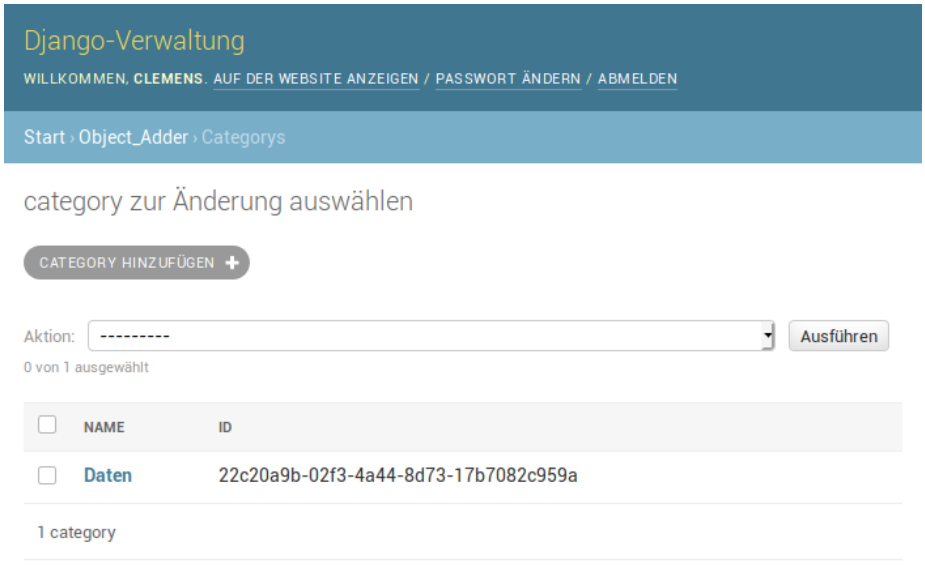
```
1  from django.contrib import admin
2  from .models import Object, Category
3
4
5  class ObjectAdmin(admin.ModelAdmin):
6          listdisplay = ('title', 'ammout', 'uuid', 'img')
7
8
9  class CategoryAdmin(admin.ModelAdmin):
10         listdisplay = ('name', 'id')
11
12
13 # Register your models here.
14 admin.site.register(Object, ObjectAdmin)
15 admin.site.register(Category, CategoryAdmin)
```

In Zeile 2 werden die in der models.py definierten Models importiert um Zugriff auf diese zu erlangen.
Anschließend wird für jedes Model eine Klasse erstellt. In dieser Klasse wird jeweils eine ”list_display” Variable definiert, die ein Tupel enthält, mit dem alle darzustellenden Attribute an die admin library übergeben werden können. Diese Abmin-Klassen

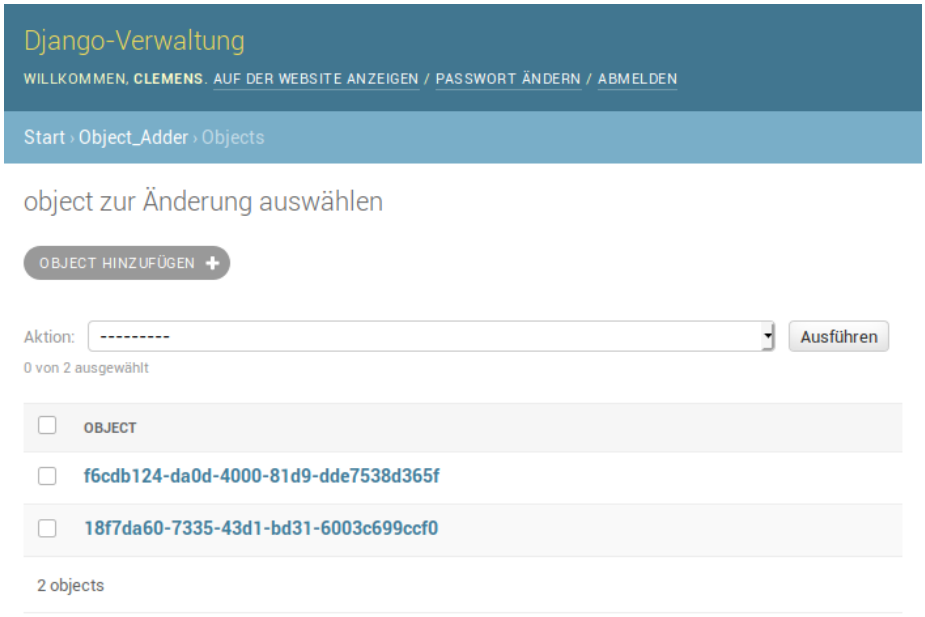
[3] Django Dokumentation https://docs.djangoproject.com/en/2.1/howto/deployment/wsgi/

müssen jetzt noch zusammen mit dem Model an die admin library übergeben werden. Dies erfolgt duch die Befehle in Zeile 14 und 15. Dieser Code erzeugt demnach folgendes Admin-Interface:





Ohne die ”list_display” Angebe würde nur der primary key des Models angezeigt und das Interface sähe folgendermaßen aus:



2.5.2 apps.py

Hier ist der Name der app definiert. Diese Datei ist außerdem notwendig, damit die App von Django als solche erkannt wird.

```
1 from django.apps import AppConfig
2
3 class ObjectAdderConfig(AppConfig):
4         name = 'objectadder'
```

2.5.3 forms.py

In dieser Datei werden Formulare definiert. So beispielsweise das Formular zur Erstellung von Objekten und Kategorien, das folgendermaßen aussieht:

```
1 from django.forms import ModelForm, TextInput
2 from .models import Object, Category
3
```

```
4
5  class ObjectForm(ModelForm):
6          class Meta:
7                  model = Object
8                  fields = ('ammout', 'title', 'img',
9                  'description', 'category')
10                 widgets = -
11                         'title': TextInput(),
12
13
14
15 class CategoryForm(ModelForm):
16         class Meta:
17                 model = Category
18                 fields = ['name']
19                 widgets = -
20                         'name': TextInput()
21
```

Hier werden zwei Formulare definiert. Es wird angegeben mit welchem model das Formular asoziiert werden soll und welche Felder angezeigt werden sollen. Außerdem wird definiert, dass für das name und das title Feld ein ”TextInput()” Feld genutzt werden soll.

3 Routing in Django

Das grundlegende und wichtigste Prinzip in Django ist das sogenannte ”Routing”. Dieses Prinzip beschreibt den Weg, den eine Anfrage zurücklegt um zu einer Antwort zu führen. Routing in Django funktioniert, indem die Url mit Hilfe von Regular Expressions in Teile aufgespalten wird, die dann einzeln bis letztendlich ein HTML Template zurück gegeben wird verfolgt werden.

3.1 Root Url Config

Diese Datei ist die grundlegende Url Konfigurationsdatei. In ihr schaut der Server zuerst nach. Sie befindet sich im server/server/urls.py Ordner und gehört zur Hauptapp des Servers. Sie kann beispielsweise folgendermaßen aussehen:

```
1  """invsystem URL Configuration
2
3  The urlpatterns list routes URLs to views. For more
4  information please see:
5  https://docs.djangoproject.com/en/2.1/topics/http/urls/
6  Examples:
```

```
Function views
1. Add an import:  from myapp import views
2. Add a URL to urlpatterns:  path('', views.home, name='home')
Class-based views
1. Add an import:  from otherapp.views import Home
2. Add a URL to urlpatterns:  path('', Home.asview(), name='home')
Including another URLconf
1. Import the include() function:
from django.urls import include, path
2. Add a URL to urlpatterns:  path('blog/', include('blog.urls'))
"""
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
path('admin/', admin.site.urls),
path('accounts/', include('django.contrib.auth.urls')),
path('', include('usermanager.urls')),
path('add/', include('objectadder.urls')),
path('list/', include('objectlister.urls')),
path('settings/', include('settingsapp.urls')),
]
```

Die Kommentare werden beim Erstellen des Projektes automatisch generiert. Diese Datei erzeugt also sechs URL Pfade. Alle Pfade bis auf den ”/admin” Pfad zeigen hier auf die urls.py Datei einer anderen App. Dies wird mit dem ”include()” Statement erreicht. Die ”/admin” Url wird mit hilfe des Admin Paketes gehandhabt, dass automatisch ein Admin Interface zur Datenbankverwaltung erzeugt.

3.2 Weitere Url Konfigurationsdateien

Mithilfe der ”include()” Statements kann die Anfrage jetzt theoretisch durch eine unbegrenzte Zahl von urls.py Dateien geleitet werden. In diesem Beispiel wird in der settings_app.urls jedoch direkt der sogenannte ”View” zurück gegeben.

```
from django.urls import path
from . import views

urlpatterns = [
path('', views.index, name='settingsindex'),
]
```

Dies liegt daran, dass in der ”path()” Funktion kein ”include()” Statement mehr steht sondern ”views.index”. Außerdem wird in der zweiten Zeile die ”views.py” der App importiert.

3.3 View

Im sogenannten View wird findet die eigentliche Programmlogik statt. Hier werden Daten an die ”render()” Funktion übergeben, die das gerenderte HTML Template zurück gibt. Diese Daten können beispielsweise aus einem Formular stammen oder von der Datenbank abgefragt sein. Hier werden auch POST Anfragen bearbeitet.

4 Template rendering