Pythonimdbadvanced

individuelles Projekt

Alexandros Konstantinidis | inf2260 | Matrikel-Nr.: 669451

Dozent: Herr Mahler  
Datum:20.01.2017

[Einleitung 1](#_Toc472892951)

[Anforderungen der Anwendung 1](#_Toc472892952)

[Verwendete Technologien 1](#_Toc472892953)

[Projektstruktur 2](#_Toc472892954)

[Python erste Schritte 3](#_Toc472892955)

[Initialisierung der Anwendung 3](#_Toc472892956)

[Datenbankanbindung 3](#_Toc472892957)

[Routing 4](#_Toc472892958)

[Formulare 5](#_Toc472892959)

[Erste Schritte 5](#_Toc472892960)

[Registrierungsformular 5](#_Toc472892961)

[Registrierungstemplate 6](#_Toc472892962)

[Bearbeitung der Daten 7](#_Toc472892963)

[Validierung 8](#_Toc472892964)

[Fehlerausgabe 8](#_Toc472892965)

[Andere Formulare 8](#_Toc472892966)

[Login 9](#_Toc472892967)

[Sicherheit der Nutzer 9](#_Toc472892968)

[Seiten Navigation 10](#_Toc472892969)

[Movie listing 10](#_Toc472892970)

[Favorisieren 10](#_Toc472892971)

[Kommentare 11](#_Toc472892972)

[Nutzerprofil 13](#_Toc472892973)

[Admin Panel 13](#_Toc472892974)

[Suche 14](#_Toc472892975)

[Rating 14](#_Toc472892976)

[RSS 15](#_Toc472892977)

[Kontaktfunktion 16](#_Toc472892978)

[SQL 17](#_Toc472892979)

[Datenbankstruktur 17](#_Toc472892980)

[Quellen 18](#_Toc472892981)

[Python Code 19](#_Toc472892982)

[Pythonimdbadvanced/settings.py 19](#_Toc472892983)

[Pythonimdbadvanced/urls.py 22](#_Toc472892984)

[Pythonimdbadvanced/wsgi.py 22](#_Toc472892985)

[main/admin.py 22](#_Toc472892986)

[main/apps.py 23](#_Toc472892987)

[main/forms.py 23](#_Toc472892988)

[main/models.py 25](#_Toc472892989)

[main/urls.py 27](#_Toc472892990)

[main/views.py 28](#_Toc472892991)

[manage.py 33](#_Toc472892992)

Abbildungsverzeichnis

[Abbildung 1: Projektstruktur 2](https://d.docs.live.net/2e1a9487b2ce9f87/Documents/Uni/Serverseitig/IMDB%20Clone%20in%20Python.docx#_Toc440767822)

[Abbildung 2: Datenbankstruktur 17](#_Toc440767823)

# Einleitung

## Anforderungen der Anwendung

Ziel ist es eine Plattform bereitzustellen, in der Filme, inklusive Beschreibung, Trailer und Bilder, gelistet sind. Nutzer haben die Möglichkeit sich anzumelden und Filme aus der Liste zu kommentieren, zu favorisieren und ggf. Filme, die nicht in der Datenbank vorhanden sind, selber anzulegen, über neue Updates informiert zu werden und ihre Favoriten zu verwalten. Nutzer sollten außerdem in der Lage sein, die Profile und Favoritenliste anderer Nutzer einzusehen. Um die Qualität der Inhalte zu gewährleisten, sollte eine Admin-Oberfläche zur Verfügung stehen und Moderatoren sollten in der Lage sein Kommentare zu editieren.

## Verwendete Technologien

Die Anwendung wurde in Python 3.5 mit dem-Framework Django programmiert. Die Datenbankanforderungen wurden mit SQLite umgesetzt. Für die clientseitige korrekte Anzeige der Informationen, wurde eine Kombination aus Django-Templatemodul, Bootstrap und Javascript in Kombination mit JQuery eingesetzt.

## http://puu.sh/tuLL6/ec8f1b6c42.pngProjektstruktur

Das Projekt setzt sich aus folgenden Dateien und Ordnern zusammen (s. Abbildung 1):

**PythonIMDBAdvanced:** Ist der Hauptprojekt Ordner

**main:** Die Hauptdjangoapp

**\_\_init\_\_.py:** Initialisierungsdatei der Anwendung.

**settings.py:** Globale Konfigurationsparameter.

**wsgi.py:** Ermöglicht es, dass Web-Server die Anwendung aufrufen können.

**urls.py:** Beinhaltet die verschiedenen Routen der Anwendung.

**admin.py:** Hier werden die verschiedenen Einstellung bzgl. der Adminoberfläche festgelegt.

**views.py**: Beinhaltet die verschiedenen Hauptfunktionen der Routen.

**models.py:** Modellierung der Datenbank.

**app.py:** Hier werden verschiedene Funktionen definiert.

**forms.py:** Definitionen von Formularen, welche die Templates nutzen.

**Staticfiles:** Ordner welcher die benötigten css, js, font und Bilder Dateien beinhaltet.

**Templates:** Alle benötigten Templates für die korrekte Anzeige der Anwendung.

Abbildung : Projektstruktur

**clean.sh:** Skript, welcher die initiale Datenbank und Admin festlegt.

**db.sqlite3:** Die Datenbankdatei.

**manage.py:** Startet und verwaltet die Anwendung.

**Requierements.txt;** Beinhaltet alle benötigten Abhängigkeiten der Funktionen, welche installiert werden müssen.

# Python erste Schritte

## Initialisierung der Anwendung

Damit die Anwendung starten kann, muss auf dem System Python 3.5 mit den erforderlichen Modulen installiert sein.   
Die Anwendung wird durch die Datei **manage.py,** mit dem Befehl „runserver“,gestartet:

#!/usr/bin/env python  
*import* os  
*import* sys  
  
*if* \_\_name\_\_ == **"\_\_main\_\_"**:  
 os.environ.setdefault(**"DJANGO\_SETTINGS\_MODULE"**, **"PythonIMDBAdvanced.settings"**)  
  
 *from* django.core.management *import* execute\_from\_command\_line  
  
 execute\_from\_command\_line(sys.argv)

Es wird zunächst die Standard Umgebung, Pfade und Konfigurationsdatei bestimmt. **Manage.py** nimmt Paramater aus der Kommandozeile entgegen und führt diese aus. In unserem Fall, wird die Anwendung mit folgendem Befehl ausgeführt: „Python3 manage.py runserver“.

## Datenbankanbindung

Die Datenbankanbindung findet über das Django Datenbankmodul statt. Dies ermöglicht es, dass jegliche Tabelle als Objekt behandelt wird. Das ermöglicht, dass die Datensätze beliebig verwendbar und bearbeitbar sind. Hierfür ist es lediglich erforderlich die Verbindungsdaten in der **settings.py** zu hinterlegen und die Datenbank einmalig als Modell in **models.py** anzulegen und zu initialisieren.

**Beispiel:**

**settings.py**

DATABASES = {  
 **'default'**: {  
 **'ENGINE'**: **'django.db.backends.sqlite3'**,  
 **'NAME'**: os.path.join(BASE\_DIR, **'db.sqlite3'**),  
 }  
}

**models.py**

*from* django.db *import* models  
*from* django.db.models *import* Avg  
*from* django.contrib.auth.models *import* User  
*from* django.core.files.storage *import* FileSystemStorage  
*from* django.conf *import* settings  
*from* watson *import* search *as* watson  
*import* datetime  
  
fs = FileSystemStorage(location=settings.MEDIA\_ROOT)  
  
*class* Profile(models.Model):  
 user = models.OneToOneField(User, on\_delete=models.CASCADE)  
 picture = models.ImageField(storage=fs, blank=*True*)  
 biography = models.TextField(max\_length=3000, blank=*True*)  
  
 *def \_\_str\_\_*(self):  
 *return* **"Profile {email}"**.format(  
 email=self.user.email  
 )

Um die angelegten Modelle benutzen zu können, müssen die gewünschten Modelle in den Header importiert werden.

**Beispiel:**

**views.py**

*from* main.models *import* Movies, MovieImage, FavMovies, Comments, Genre, MovieGenre, MovieRating

## Routing

Mit Routing bestimmt man, welche Aktion bei welchem Parameter durchgeführt wird.

**Beispiel:**

**urls.py**

urlpatterns = [  
 url(**r'^$'**, views.home, name=**'home'**),

}

Hier wird definiert, dass bei der Rootdomain das die Funktionen **home** aus der **views.py** ausgeführt wird.

# Formulare

## Erste Schritte

Für die Templates wird das integrierte Modul von Django genutzt. Die Templates werden im Ordner **templates** hinterlegt.

Da die Webseite Elemente hat, welche sich wiederholen, wird ein Haupttemplate **base.html** erstellt**.** Dieses wird in jedes weitere Template integrieren.  
Damit der Bereich, in welchen die neuen Templates geladen werden sollen, festgelegt werden kann, wird an der gewünschten Stelle in **base.html** folgender Code eingefügt:

{% **block** content %}{% **endblock** %}

In die Templates, die um den Inhalt der **base.html** erweitern werden sollen, wird am Anfang

{% **extends** 'base.html' %}

und am Ende

{% **endblock** %}

eingefügt.

Um Formulare zusammen mit den Templates nutzen zu können, bemächtigen wir uns dem Django Formmodul.

Damit die Formulare abgesichert werden können, wird in der **settings.py** einen Secret Key bestimmt, der bei jedem Aufruf eines Template abgefragt wird.

SECRET\_KEY = **'gqrnr3j)-%#mz21=ygcv0%z9t#-68wnill@!i8++-@f9\_un29#'**

Die Formulare werden alle in der Datei **forms.py** definiert. Für den Anfang werden die benötigten Komponente aus den genutzten Modulen importiert.

*from* django *import* forms

## Registrierungsformular

Um die Registrierung von Nutzern zu ermöglichen, bemächtigen wir uns dem allauth Modul von Django. Damit wir dieses für unsere zwecke anpassen können, muss ein Formular für das Template bereitgestellt werden. Dafür müssen die Daten vom Server entgegengenommen und in die Datenbank abgespeichert werden.

Zuerst wird das Modul in **settings.py** konfiguriert:

ACCOUNT\_USERNAME\_REQUIRED=*False*ACCOUNT\_EMAIL\_REQUIRED=*True*ACCOUNT\_AUTHENTICATION\_METHOD=**"email"**ACCOUNT\_SIGNUP\_FORM\_CLASS = **'main.forms.SignupForm'**LOGIN\_REDIRECT\_URL=**'/'**

Hier bestimmen wir, ob der Benutzername oder die E-Mail als Anmeldename verwenden werden, was die Validierungsmethode ist, ob wir das Formular erweitern wollen und wo der Benutzer nach dem erfolgreichen Anmeldeprozess weitergeleitet werden soll.

Parallel wird ein Formular in **forms.py** angelegt:

*class* SignupForm(forms.Form):  
 first\_name = forms.CharField(max\_length=30, label=**'Vorname'**, widget=forms.TextInput(attrs={**'placeholder'**: **'Vorname'**}))  
 last\_name = forms.CharField(max\_length=30, label=**'Nachname'**, widget=forms.TextInput(attrs={**'placeholder'**: **'Nachname'**}))  
  
 *def* signup(self, *request*, *user*):  
 *user*.first\_name = self.cleaned\_data[**'first\_name'**]  
 *user*.last\_name = self.cleaned\_data[**'last\_name'**]  
 *user*.save()  
 Profile.objects.create(  
 user=*user*,  
 )

Jedes neue Formular muss als **class** hinterlegt werden, gefolgt von dem **Namen.** In unserem Beispiel werden die Felder aus der **Form** geerbt**.**

Für die Registrierung werden folgende Felder benötigt:

* Vorname: Der Vorname des Nutzers
* Nachname: Der Nachname des Nutzers
* email: Email des Nutzers
* password: Gewünschtes Passwort
* password\_confirm: Zur Absicherung wird die erneute Eingabe des Passworts verlangt

## Registrierungstemplate

Damit das Formular auch im Template angezeigt wird, muss es dort hinterlegen:

{% **block** account %}  
<**h1**>{% **trans** "Registrieren" %}</**h1**>  
  
<**p**>{% **blocktrans** %}**Besitzen Sie bereits ein Konto?** <**a** href=**"**{{ login\_url }}**"**>**Login**</**a**>**.**{% **endblocktrans** %}</**p**>  
  
<**form** class=**"signup"** id=**"signup\_form"** method=**"post"** action=**"**{% **url** 'account\_signup' %}**"**>  
 {% **csrf\_token** %}  
 <**table**>  
 {{ form.as\_table }}  
 </**table**>  
 {% **if** redirect\_field\_value %}  
 <**input** type=**"hidden"** name=**"**{{ redirect\_field\_name }}**"** value=**"**{{ redirect\_field\_value }}**"** />  
 {% **endif** %}  
 <**button** type=**"submit"** class=**"primaryAction btn btn-primary btn-md"**>{% **trans** "Registrieren" %} **&raquo;**</**button**>  
</**form**>  
  
{% **endblock** %}

Die Daten werden mit der **POST-Methode** an die Anwendung weitergesendet. Um die Formularfelder anzuzeigen, müssen diese abgerufen werden. Hier werden die Formularfelder als „Table“ gerendert.

<**table**>  
 {{ form.as\_table }}  
</**table**>

Wie weiter oben beschrieben, wird das Formular abgesichert, indem am Ende der **Secret\_Key** geprüft wird.

{% **csrf\_token** %}

## Bearbeitung der Daten

Damit das richtige Template angezeigt wird, werden die Routen in **urls.py** bestimmt

url(**r'^accounts/'**, include(**'allauth.urls'**)),

Sobald der Server eine /**accounts** GET Anfrage mit dem Parameter **/signup** bekommt, wird das Template **signup.html** mit dem Formular **SignupForm** geladen.

Damit der Server die an ihm weitergeleiteten Daten verarbeiten kann, muss vorher ein View definiert werden, die den **POST** bearbeitet. Gut zu sehen an unserem **addMovie** Beispiel:

@verified\_email\_required  
*def* addMovie(*request*):  
 *if request*.method == **'POST'**:  
 movie\_form = MovieForm(*request*.POST, *request*.FILES)  
 *if* movie\_form.is\_valid():  
 movie = Movies.objects.create(  
 name = movie\_form.cleaned\_data[**'name'**],  
 description = movie\_form.cleaned\_data[**'description'**],  
 description\_short = movie\_form.cleaned\_data[**'description\_short'**],  
 logo = *request*.FILES[**'logo'**],  
 created\_at = movie\_form.cleaned\_data[**'created\_at'**],  
 video\_url = movie\_form.cleaned\_data[**'video\_url'**]  
 )  
 messages.success(*request*, **'Film wurde erfolgreich angelegt.'**)  
 *return* redirect(**'addMoviesMedia'**, movie.id)  
 *else*:  
 messages.error(*request*, **'Fehler beim Speichern des Filmes. Bitte überprüfen Sie alle Felder.'**)  
 *return* render(*request*, **'main/movie\_new.html'**, {**'movie\_form'**: movie\_form})  
 *else*:  
 context = { **'movie\_form'**: MovieForm(),}  
 *return* render(request=*request*, template\_name=**'main/movie\_new.html'**, context=context)

Es wird zu Beginn geprüft, ob es sich um eine **POST** Anfrage handelt, im Anschluß werden die Anfrage Daten dem Formular übergeben und es wird geprüft ob die Daten valide sind.

Wenn alles valide ist, wird ein neues **movie** Objekt mit den weitergeleiteten Daten erstellt und in der Datenbank gespeichert .Der Nutzer wird benachrichtigt, ob sein Upload erfolgreich war.

## Validierung

Die Validierung findet auf zwei Ebenen statt. Auf der Ersten werden die angegebenen Validatoren geprüft:

**Beispiel:**

first\_name = forms.CharField(max\_length=30, label=**'Vorname'**, widget=forms.TextInput(attrs={**'placeholder'**: **'Vorname'**}))

Was der Validator bestimmt hier, dass das Feld Ausgefüllt werden muss (**InputRequiered** ist in Default „True“**)** und welche Länge nicht überschritten werden darf.

Sollte eine der Bedingungen nicht erfüllt sein, wird eine Fehlermeldung an das Formular übergeben.

## Fehlerausgabe

Für eine, für den Nutzer sichtbare, Anzeige der Fehlerausgabe im Template, muss diese in die Templates implementiert werden.

{{ movie\_form.name.errors }}

**Beispiel:**

{{ movie\_form.name.errors }}  
<**div** class=**"form-group"**>  
 <**label** for=**"**{{ movie\_form.name.id\_for\_label }}**"** class=**"col-lg-2 control-label"**>{{ movie\_form.name.label\_tag }}</**label**>  
 <**div** class=**"col-lg-10"**>  
 {{ movie\_form.name }}  
 </**div**>  
</**div**>

## Andere Formulare

Nach gleichem Prinzip wird auch bei den anderen Formularen vorgegangen.

# Login

## Sicherheit der Nutzer

Um zu gewährleisten, dass kein Dritter sich unbefugt Zugriff verschaffen kann, wird das Passwort zusätzlich zur Hash Speicherung auch mit einem Salt versehen.  
Das Django **allauth** Modul speichert den gehashten String wie folgt:

pbkdf2:sha256:24000$salt$hash

Hier wird ein Salt ans Passwort angehangen und daraus ein Hash generiert. Der Vorgang wird vierundzwanzigtausend Mal wiederholt und erst dann in der Datenbank gespeichert.  
Der weiter oben angegebene String bestimmt die verwendete Methode zum Haschen, deren Wiederholungen und welcher Salt benutzt worden soll.  
Es wird außerdem sichergestellt, dass alle gespeicherten Hashes die gleiche Länge haben, unabhängig davon, wie lang das Passwort ursprünglich ist und das kein Hash identisch zu einem Anderen sein darf.

# Seiten Navigation

## Movie listing

Die Filmliste der Datenbank wird auf der Hauptseite angezeigt.  
Damit die Filme angezeigt werden können, muss in **views.py** beim laden des Templates **index.html** die Filme als ein Objekt mitgeben werden.

*def* home(*request*):  
 context = {  
 **"movies"**: Movies.objects.all(),  
 }  
 *return* render(request=*request*, template\_name=**'main/home.html'**, context=context)

Für eine richtige Verlinkung der Nutzer zum ausgewählten Film, wird beim Erstellen der Objekte die **id** jedes Filmes gespeichert, welche zur Erstellung des Links benutzt wird.

<**a** href=**"**{% **url** 'showMovie' movie.id %}**"**>

## Favorisieren

Beim Favorisieren wird die **id** des ausgewählten Films und die des aktiven Nutzers an die Anwendung übergeben.  
Anschließend wird geprüft, ob der Film vom aktuellen Nutzer schon favorisiert worden ist. Ansonsten wird ein neuer Eintrag in der Datenbank gemacht und der Nutzer wird benachrichtigt, ob sein Versuch erfolgreich war.

@login\_required  
*def* like\_movie(*request*, *movie\_id*):  
 *try*:  
 movie = get\_object\_or\_404(Movies, id=*movie\_id*)  
 FavMovies.objects.create(  
 user=*request*.user,  
 movie=movie  
 )  
 messages.success(*request*, **"Sie haben den Film erfolgreich favorisiert."**)  
 *return* redirect(**'showMovie'**, *movie\_id*)  
 *except*:  
 messages.error(*request*, **"Ein Fehler ist aufgetretten, bitte versuchen Sie es noch einmal."**)  
 *return* redirect(**'showMovie'**, *movie\_id*)

Im Fall, dass der Nutzer den Film nicht in seinen Favoriten haben möchte, ist der Ablauf ähnlich. Die **id** wird wieder übergeben und es wird geprüft, ob der Film bereits Favorisiert worden ist. Bei einem positiven Befund wird der Eintrag aus der Datenbank gelöscht und der Nutzer erhält eine Benachrichtigung darüber, ob sein Versuch erfolgreich umgesetzt werden konnte.

@login\_required  
*def* unlike\_movie(*request*, *movie\_id*):  
 *try*:  
 movie = get\_object\_or\_404(Movies, id=*movie\_id*)  
 fav = FavMovies.objects.get(user=*request*.user, movie=movie)  
 fav.delete()  
 messages.success(*request*, **"Sie favorisieren den Film nicht mehr."**)  
 *return* redirect(**'showMovie'**, *movie\_id*)  
 *except*:  
 messages.error(*request*, **"Ein Fehler ist aufgetretten, bitte versuchen Sie es noch einmal."**)  
 *return* redirect(**'showMovie'**, *movie\_id*)

## Kommentare

Die Kommentare werden in **movie\_show.html** angezeigt, in welchem die Kommentare nach Datum sortiert und mit dem Ersteller sowie dessen Profilbild angezeigt werden.  
Dort wird außerdem geprüft ob der Nutzer angemeldet ist, falls ja, kann er selber Kommentare posten, ansonsten werden ihm nur die Kommentare angezeigt. Sollte der Nutzer ein Moderator sein, wird ihm unter jedem Kommentar ein Feld angezeigt, in dem er das Kommentar editieren kann.

**forms.py**

*class* CommentsForm(forms.ModelForm):  
  
 comment = forms.CharField(max\_length=3000, label=**'Kommentar'**, widget=TinyMCE(attrs={**'cols'**:70, **'rows'**: 5, **'placeholder'**: **'Geben Sie hier Ihr Kommentar ein...'**}), required=*False*)  
  
 *class* Meta:  
 model = Comments  
 fields = [**'comment'**]

**movie\_show.html**

{% **if** user.is\_authenticated %}  
<**div** class=**"col-md-12"**>  
 <**form** class=**"form-horizontal"** enctype=**"multipart/form-data"** method=**"POST"** action=**"**{% **url** 'comment' movie.id %}**"**>  
 <**fieldset**>  
 {% **csrf\_token** %}  
 <**div** class=**"form-group"**>  
 <**label** for=**"**{{ comment\_form.comment.id\_for\_label }}**"** class=**"col-lg-2 control-label"**>{{ comment\_form.comment.label\_tag }}</**label**>  
 <**div** class=**"col-lg-10"**>  
 {{ comment\_form.comment }}  
 </**div**>  
 <**button** type=**"submit"** class=**"btn btn-primary btn-md edit"**>**Kommentieren**</**button**>  
 </**div**>  
 </**fieldset**>  
 </**form**>  
</**div**>  
{% **endif** %}  
{% **if** movie.comments\_set.all %}  
<**div** class=**"col-md-12 comments"**>  
 {% **for** comment in movie.comments\_set.all %}  
 <**div** class=**"col-md-2 profile"**>  
 <**a** href=**"**{% **url** 'profile\_show' comment.user.id %}**"**>  
 {% **if** comment.user.profile.picture %}  
 <**div** class=**"profile-img"** style=**"***background-image*: url({{ comment.user.profile.picture.url }})**"**></**div**>  
 {% **else** %}  
 <**div** class=**"profile-img"** style=**"***background-image*: url({% **static** 'img/placeholder.png' %})**"**></**div**>  
 {% **endif** %}  
 </**a**>  
 </**div**>  
 {% **if** user.moderator.is\_moderator %}  
 <**form** class=**"form-horizontal"** enctype=**"multipart/form-data"** method=**"POST"** action=**"**{% **url** 'commentEdit' movie.id comment.id %}**"**>  
 <**fieldset**>  
 {% **csrf\_token** %}  
 <**div** class=**"col-lg-10 comment"**>  
 <**a** href=**"**{% **url** 'profile\_show' comment.user.id %}**"**> <**h6**>{{ comment.user }}</**h6**></**a**>  
 {% **if** comment.is\_moderatored %}  
 <**strong**>**Moderator edit:** </**strong**>{{ comment.comment }}  
 {% **else** %}  
 {{ comment.comment }}  
 {% **endif** %}  
 <**div** class=**"line-separator"**></**div**>  
 <**h5**>**Edit Comment:**</**h5**>  
 {{ comment\_edit.comment }}  
 <**button** type=**"submit"** class=**"btn btn-primary btn-sm edit"**>**Editieren**</**button**>  
 </**div**>  
 </**fieldset**>  
 </**form**>  
 {% **else** %}  
 <**div** class=**"col-lg-10 comment"**>  
 <**a** href=**"**{% **url** 'profile\_show' comment.user.id %}**"**> <**h6**>{{ comment.user }}</**h6**></**a**>  
 {% **if** comment.is\_moderatored %}  
 <**strong**>**Moderator edit:** </**strong**>{{ comment.comment }}  
 {% **else** %}  
 {{ comment.comment }}  
 {% **endif** %}  
 </**div**>  
 {% **endif** %}  
 {% **endfor** %}  
</**div**>  
{% **endif** %}

{% **endfor** %}  
</**ul**>

Damit das abgesendete Formular in der Datenbank gespeichert wird, werden ähnlich, wie bereits in der Registrierung (Seite 3) erläutert, die Daten validiert und im Anschluss als neues Objekt in die Datenbank geschrieben.

Da die Kommentare in einer Zwischentabelle gesichert sind, muss dafür gesorgt werden, dass diese im jeweiligen Film gelistet und auch die dazugehörigen Nutzer angezeigt werden. Hierfür bietet das Django Datenbankmodul Aushilfe. Sobald ein Model zwei oder mehr Fremdschlüssel hat, kann dieses Model, als ein Objekt des verknüpften Models, aufgerufen werden:

movie.comments\_set.all

## Nutzerprofil

Im Profil können Nutzer ihre angegebenen Daten und Favorisierte Filme einsehen und bearbeiten. Nutzer haben auch die Möglichkeit die Profile andere Nutzer anzusehen, um z.B. deren favorisierten Filme einzusehen.

*def* profile\_show(*request*, *user\_id*):  
 """  
 Profile page  
 """  
 user = get\_object\_or\_404(User, id=*user\_id*)  
 context = {  
 **'user'**: user,  
 **'contactform'** : ContactForm(),  
 **'fav'**: FavMovies.objects.filter(user\_\_id=*user\_id*)  
 }  
 *return* render(*request*, **'main/profile\_show.html'**, context)

## Admin Panel

Es wurde auch ein simples Admininterface implementiert. Hierfür wurde das Django Admin Modul verwendet. Damit es genutzt werden kann, muss es erstmal definiert und konfiguriert werden.  
Die Definition finden in **admin.py** statt.

*class* MovieImageInline(admin.TabularInline):  
 model = MovieImage  
  
*class* FavMoviesInline(admin.TabularInline):  
 model = FavMovies  
  
*class* MovieAdmin(admin.ModelAdmin):  
 inlines = [MovieImageInline, FavMoviesInline]  
 search\_fields = [**'name'**]  
  
*class* ModeratorInline(admin.StackedInline):  
 model = Moderator  
 verbose\_name\_plural = **'moderator'***class* ProfileInline(admin.StackedInline):  
 model = Profile  
 verbose\_name\_plural = **'profile'***class* UserAdmin(BaseUserAdmin):  
 inlines = [ProfileInline, ModeratorInline]  
  
admin.site.register(Movies, MovieAdmin)  
admin.site.register(Genre)  
admin.site.unregister(User)  
admin.site.register(User, UserAdmin)

## Suche

Für die korrekte Implementierung der Suche wurde **watson** als Suchmodul verwendet. Hierfür muss in **app.py** **watson** konfiguriert werden

@watson.update\_index()  
*def* ready(self):  
 Movies = self.get\_model(**"Movies"**)  
 watson.register(Movies, fields=(**"name"**, **"description"**))

Und im Anschluss in **models.py** das Model registriert werden:

*def* update\_movies\_index(instance, *\*\*kwargs*):  
 *for* movies *in* instance.movies\_set.all():  
 watson.default\_search\_engine.update\_obj\_index(movies)

## Rating

Die Benutzer haben die Möglichkeit Filme zu bewerten. Erstmal wurde ein **MovieRating** Model in  **models.py** definiert.

*class* MovieRating(models.Model):  
 NEUTRAL = 0  
 VERY\_BAD = 1  
 BAD = 2  
 OK = 3  
 GOOD = 4  
 VERY\_GOOD = 5  
 RATING\_CHOICES = (  
 (NEUTRAL, **'0'**),  
 (VERY\_BAD, **'1'**),  
 (BAD, **'2'**),  
 (OK, **'3'**),  
 (GOOD, **'4'**),  
 (VERY\_GOOD, **'5'**),  
 )  
 movie = models.ForeignKey(Movies, on\_delete=models.CASCADE)  
 user = models.ForeignKey(User, on\_delete=models.CASCADE)  
 rating = models.IntegerField(choices=RATING\_CHOICES, default=NEUTRAL)

Es wurde eine Auswahl an Bewertungsmöglichkeiten vordefiniert (von 0 bis 5).

Damit wir gewährleisten, dass keine doppelten Einträge in die Datenbank gespeichert werden. Haben wir **movie** und **user** unique verlinkt.

*class* Meta:  
 unique\_together = [**"movie"**, **"user"**]

Anschließend haben wurde ein Formular angelegt:

*class* MovieRatingForm(forms.ModelForm):  
  
 rating = forms.ChoiceField(choices = MovieRating.RATING\_CHOICES, label=**""**, initial=**''**, widget=forms.Select(), required=*False*)  
  
 *class* Meta:  
 model = MovieRating  
 fields = [**'rating'**]

Welches die möglichen Bewertungen aus dem **MovieRating** Model bezieht.

In der **views.py** wird auch geprüft, ob der Nutzer bereits den Film bewertet hat:

*if request*.user *in* [movierating.user *for* movierating *in* movie.movierating\_set.all()]:  
 messages.error(*request*, **'Sie haben schon eine Bewertung abgegeben.'**)  
 *return* redirect(**'showMovie'**, movie.id)

Sollte dies der Fall sein, erhält er eine Fehlermeldung und wird zum Film zurückgeleitet. Bei einer negativen Prüfung wird ein neues **MovieRating** Objekt erstellt:

MovieRating.objects.create(  
 movie = movie,  
 user = *request*.user,  
 rating = rating\_form.cleaned\_data[**'rating'**]  
)  
messages.success(*request*, **'Erfolgreich bewertet.'**)  
*return* redirect(**'showMovie'**, movie.id)

Der Nutzer wird benachrichtig, dass seine Bewertung erfolgreich war und wird weitergeleitet.

## RSS

Damit die Benutzer stets über neue Filme informiert bleiben, wurde eine RSS Funktion implementiert. Der Feed ist unter **/rss/movie/** erreichbar.

*class* LatestMovieFeed(Feed):  
 title = **"Neuesten Filme"** link = **"/rss/"** description = **"Die neuesten Filme werden hier angezeigt."** *def* items(self):  
 *return* Movies.objects.order\_by(**'-created\_at'**)[:5]  
  
 *def* item\_title(self, *item*):  
 *return item*.name  
  
 *def* item\_description(self, *item*):  
 *return item*.description  
  
 *def* item\_link(self, *item*):  
 *return* reverse(**'showMovie'**, args=[*item*.pk])

## Kontaktfunktion

Benutzer haben die Möglichkeit andere Benutzer über ihr Profil zu kontaktieren. Dafür wurde in **forms.py** ein Formular angelegt:

*class* ContactForm(forms.Form):  
 email = forms.CharField(max\_length=140, label=**'E-Mail'**, widget=forms.TextInput(attrs={**'placeholder'**: **'E-Mail'**}))  
 subject = forms.CharField(max\_length=400, label=**'Betreff'**, widget=forms.TextInput(attrs={**'placeholder'**: **'Betreff'**}))  
 message = forms.CharField(max\_length=2000, widget=forms.Textarea(attrs={**'placeholder'**: **'Geben Sie hier bitte Ihre Nachricht ein ...'**}))  
 name = forms.CharField(max\_length=50, label=**'Name'**, widget=forms.TextInput(attrs={**'placeholder'**: **'Name'**}))

Dort kann der Benutzer seine E-Mail, seinen Namen, einen Betreff und die Nachricht hinterlegen. Für die Kontaktfunktion wurde extra **contact.html** angelegt. Welches beliebig mit dem Code

{% **include** "main/contact.html" %}

In jedes Template eingefügt werden kann. Bei einer positiven Validierung des Formulars wird an den zu kontaktierenden Benutzer eine E-Mail versendet.

*if* contact\_form.is\_valid():  
 body = render\_to\_string(  
 template\_name=**'email/contactmessage.txt'**,  
 context={  
 **'name'**: user.get\_full\_name(),  
 **'sender'**: contact\_form.cleaned\_data[**'name'**],  
 **'sender\_email'**: contact\_form.cleaned\_data[**'email'**],  
 **'message'**: contact\_form.cleaned\_data[**'message'**]  
 }  
 )  
 email = EmailMessage(  
 subject = contact\_form.cleaned\_data[**'subject'**],  
 body = body,  
 to = [user.email, contact\_form.cleaned\_data[**'email'**]],  
 from\_email = **'python@imdb.advanced'** )  
 email.send()  
 messages.success(*request*, **"E-Mail erfolgreich gesendet!"**)  
 *return* redirect(**"profile\_show"**, *user\_id*)

# SQL

## Datenbankstruktur

Die Datenbank besteht aus vier aktiven Tabellen



Abbildung 2: Datenbankstruktur

Jede Tabelle wurde als Datenbank Modell in **models.py.**

# Quellen

<https://docs.djangoproject.com/en/1.10/>

https://www.twoscoopspress.com/products/two-scoops-of-django-1-8

# Python Code

## Pythonimdbadvanced/settings.py

"""  
Django settings for untitled3 project.  
  
Generated by 'django-admin startproject' using Django 1.10.4.  
  
For more information on this file, see  
https://docs.djangoproject.com/en/1.10/topics/settings/  
  
For the full list of settings and their values, see  
https://docs.djangoproject.com/en/1.10/ref/settings/  
"""  
  
*import* os  
  
# Build paths inside the project like this: os.path.join(BASE\_DIR, ...)  
BASE\_DIR = os.path.dirname(os.path.dirname(os.path.abspath(\_\_file\_\_)))  
  
  
# Quick-start development settings - unsuitable for production  
# See https://docs.djangoproject.com/en/1.10/howto/deployment/checklist/  
  
# SECURITY WARNING: keep the secret key used in production secret!  
SECRET\_KEY = **'\_kjhhp9xt$6epv)os8r)as)t^ym\*f0x7q^u(&w5i40nt%g7b7e'**# SECURITY WARNING: don't run with debug turned on in production!  
DEBUG = *True*ALLOWED\_HOSTS = [**'localhost'**, **'127.0.0.1'**]  
  
SITE\_ID = 1  
  
# Application definition  
  
INSTALLED\_APPS = [  
 **'main.apps.MainConfig'**,  
 **'django\_cleanup'**,  
 **'embed\_video'**,  
 **'django.contrib.sites'**,  
 **'django.contrib.admin'**,  
 **'django.contrib.auth'**,  
 **'django.contrib.contenttypes'**,  
 **'django.contrib.sessions'**,  
 **'django.contrib.messages'**,  
 **'whitenoise.runserver\_nostatic'**,  
 **'django.contrib.staticfiles'**,  
 **'allauth'**,  
 **'allauth.account'**,  
 **'allauth.socialaccount'**,  
 **'watson'**,  
 **'tinymce'**,  
]  
  
MIDDLEWARE = [  
 **'django.middleware.security.SecurityMiddleware'**,  
 **'whitenoise.middleware.WhiteNoiseMiddleware'**,  
 **'django.contrib.sessions.middleware.SessionMiddleware'**,  
 **'django.middleware.common.CommonMiddleware'**,  
 **'django.middleware.csrf.CsrfViewMiddleware'**,  
 **'django.contrib.auth.middleware.AuthenticationMiddleware'**,  
 **'django.contrib.messages.middleware.MessageMiddleware'**,  
 **'django.middleware.clickjacking.XFrameOptionsMiddleware'**,  
 **'watson.middleware.SearchContextMiddleware'**]  
  
AUTHENTICATION\_BACKENDS = (  
 **'django.contrib.auth.backends.ModelBackend'**,  
 **'allauth.account.auth\_backends.AuthenticationBackend'**,  
)  
  
# Django-allauth settings  
ACCOUNT\_USERNAME\_REQUIRED=*False*ACCOUNT\_EMAIL\_REQUIRED=*True*ACCOUNT\_AUTHENTICATION\_METHOD=**"email"**ACCOUNT\_SIGNUP\_FORM\_CLASS = **'main.forms.SignupForm'**LOGIN\_REDIRECT\_URL=**'/'**#ACCOUNT\_ADAPTER = 'profiles.forms.ValidatingEmailField'  
  
#Django-tinymce settings  
TINYMCE\_DEFAULT\_CONFIG = {  
 **'plugins'**: **"table,paste,searchreplace"**,  
 **'theme'**: **"advanced"**,  
 **'cleanup\_on\_startup'**: *True*,  
 **'custom\_undo\_redo\_levels'**: 10,  
}  
TINYMCE\_COMPRESSOR = *True*# Mail settings  
EMAIL\_BACKEND = **'django.core.mail.backends.console.EmailBackend'**ROOT\_URLCONF = **'PythonIMDBAdvanced.urls'**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'**,  
 ],  
 },  
 },  
]  
  
WSGI\_APPLICATION = **'PythonIMDBAdvanced.wsgi.application'**# Database  
# https://docs.djangoproject.com/en/1.10/ref/settings/#databases  
  
DATABASES = {  
 **'default'**: {  
 **'ENGINE'**: **'django.db.backends.sqlite3'**,  
 **'NAME'**: os.path.join(BASE\_DIR, **'db.sqlite3'**),  
 }  
}  
  
  
# Password validation  
# https://docs.djangoproject.com/en/1.10/ref/settings/#auth-password-validators  
  
AUTH\_PASSWORD\_VALIDATORS = [  
 {  
 **'NAME'**: **'django.contrib.auth.password\_validation.UserAttributeSimilarityValidator'**,  
 },  
 {  
 **'NAME'**: **'django.contrib.auth.password\_validation.MinimumLengthValidator'**,  
 },  
 {  
 **'NAME'**: **'django.contrib.auth.password\_validation.CommonPasswordValidator'**,  
 },  
 {  
 **'NAME'**: **'django.contrib.auth.password\_validation.NumericPasswordValidator'**,  
 },  
]  
  
  
# Internationalization  
# https://docs.djangoproject.com/en/1.10/topics/i18n/  
  
LANGUAGE\_CODE = **'de-de'**TIME\_ZONE = **'UTC'**USE\_I18N = *True*USE\_L10N = *True*USE\_TZ = *True*# Static files (CSS, JavaScript, Images)  
# https://docs.djangoproject.com/en/1.10/howto/static-files/  
  
STATIC\_ROOT = os.path.join(BASE\_DIR, **"staticfiles"**)  
STATIC\_URL = **"/static/"**MEDIA\_ROOT = os.path.join(BASE\_DIR, **"uploads"**)  
MEDIA\_URL = **"/media/"**

## Pythonimdbadvanced/urls.py

*from* django.conf *import* settings  
*from* django.conf.urls *import* url, include  
*from* django.conf.urls.static *import* static  
*from* django.contrib *import* admin  
  
urlpatterns = [  
 url(**r'^admin/'**, admin.site.urls),  
 url(**r'^'**, include(**'main.urls'**)),  
 url(**r'^accounts/'**, include(**'allauth.urls'**)),  
 url(**r'^tinymce/'**, include(**'tinymce.urls'**)),  
] + static(settings.MEDIA\_URL, document\_root=settings.MEDIA\_ROOT)

## Pythonimdbadvanced/wsgi.py

"""  
WSGI config for untitled3 project.  
  
It exposes the WSGI callable as a module-level variable named ``application``.  
  
For more information on this file, see  
https://docs.djangoproject.com/en/1.10/howto/deployment/wsgi/  
"""  
  
*import* os  
  
*from* django.core.wsgi *import* get\_wsgi\_application  
  
os.environ.setdefault(**"DJANGO\_SETTINGS\_MODULE"**, **"PythonIMDBAdvanced.settings"**)  
  
application = get\_wsgi\_application()

## main/admin.py

*from* django.contrib *import* admin  
*from* main.models *import* Movies, MovieImage, FavMovies, Genre, Profile, Moderator  
*from* django.contrib.auth.models *import* User  
*from* django.contrib.auth.admin *import* UserAdmin *as* BaseUserAdmin  
  
*class* MovieImageInline(admin.TabularInline):  
 model = MovieImage  
  
*class* FavMoviesInline(admin.TabularInline):  
 model = FavMovies  
  
*class* MovieAdmin(admin.ModelAdmin):  
 inlines = [MovieImageInline, FavMoviesInline]  
 search\_fields = [**'name'**]  
  
*class* ModeratorInline(admin.StackedInline):  
 model = Moderator  
 verbose\_name\_plural = **'moderator'***class* ProfileInline(admin.StackedInline):  
 model = Profile  
 verbose\_name\_plural = **'profile'***class* UserAdmin(BaseUserAdmin):  
 inlines = [ProfileInline, ModeratorInline]  
  
admin.site.register(Movies, MovieAdmin)  
admin.site.register(Genre)  
admin.site.unregister(User)  
admin.site.register(User, UserAdmin)

## main/apps.py

*from* django.apps *import* AppConfig  
*from* watson *import* search *as* watson  
  
  
*class* MainConfig(AppConfig):  
 name = **'main'** @watson.update\_index()  
 *def* ready(self):  
 Movies = self.get\_model(**"Movies"**)  
 watson.register(Movies, fields=(**"name"**, **"description"**))

## main/forms.py

*from* django *import* forms  
*from* django.forms *import* FileInput  
*from* tinymce.widgets *import* TinyMCE  
*import* datetime  
*from* datetimewidget.widgets *import* DateWidget  
*from* main.models *import* Profile, Movies, Comments, Genre, MovieRating  
  
DATE\_FORMATS = [  
 **'%d.%m.%Y'**]  
  
  
*class* ContactForm(forms.Form):  
 email = forms.CharField(max\_length=140, label=**'E-Mail'**, widget=forms.TextInput(attrs={**'placeholder'**: **'E-Mail'**}))  
 subject = forms.CharField(max\_length=400, label=**'Betreff'**, widget=forms.TextInput(attrs={**'placeholder'**: **'Betreff'**}))  
 message = forms.CharField(max\_length=2000, widget=forms.Textarea(attrs={**'placeholder'**: **'Geben Sie hier bitte Ihre Nachricht ein ...'**}))  
 name = forms.CharField(max\_length=50, label=**'Name'**, widget=forms.TextInput(attrs={**'placeholder'**: **'Name'**}))  
  
  
*class* SignupForm(forms.Form):  
 first\_name = forms.CharField(max\_length=30, label=**'Vorname'**, widget=forms.TextInput(attrs={**'placeholder'**: **'Vorname'**}))  
 last\_name = forms.CharField(max\_length=30, label=**'Nachname'**, widget=forms.TextInput(attrs={**'placeholder'**: **'Nachname'**}))  
  
 *def* signup(self, *request*, *user*):  
 *user*.first\_name = self.cleaned\_data[**'first\_name'**]  
 *user*.last\_name = self.cleaned\_data[**'last\_name'**]  
 *user*.save()  
 Profile.objects.create(  
 user=*user*,  
 )  
  
  
*class* MovieForm(forms.Form):  
 name = forms.CharField(  
 max\_length=Movies.\_meta.get\_field(**'name'**).max\_length,  
 label=**"Filmname\*"** )  
 description = forms.CharField(  
 max\_length=Movies.\_meta.get\_field(**'description'**).max\_length,  
 label=**"Beschreibung\*"**,  
 widget=TinyMCE  
 )  
 description\_short = forms.CharField(  
 max\_length=Movies.\_meta.get\_field(**'description\_short'**).max\_length,  
 label=**"Kurzbeschreibung\*"** )  
 logo = forms.ImageField(  
 label=**'Logo\*'** )  
 created\_at = forms.DateField(  
 input\_formats=DATE\_FORMATS,  
 label=**'Erscheinungsdatum\*'**,  
 widget=DateWidget(usel10n=*True*, bootstrap\_version=3)  
 )  
 video\_url = forms.CharField(  
 max\_length=Movies.\_meta.get\_field(**'video\_url'**).max\_length,  
 label=**"Video URL (YouTube)"**,  
 required=*False* )  
  
*class* CommentsForm(forms.ModelForm):  
  
 comment = forms.CharField(max\_length=3000, label=**'Kommentar'**, widget=TinyMCE(attrs={**'cols'**:70, **'rows'**: 5, **'placeholder'**: **'Geben Sie hier Ihr Kommentar ein...'**}), required=*False*)  
  
 *class* Meta:  
 model = Comments  
 fields = [**'comment'**]  
  
  
*class* CommentsEditForm(forms.ModelForm):  
  
 comment = forms.CharField(max\_length=3000, label=**'Kommentar'**, widget=TinyMCE(attrs={**'rows'**: 2, **'placeholder'**: **'Geben Sie hier das neue Kommentar ein...'**}), required=*False*)  
  
 *class* Meta:  
 model = Comments  
 fields = [**'comment'**]  
  
  
*class* ProfileEditForm(forms.ModelForm):  
  
 biography = forms.CharField(max\_length=3000, label=**'Biographie'**, widget=TinyMCE(attrs={**'cols'**:80, **'placeholder'**: **'Erzählen Sie etwas über sich hier...'**}), required=*False*)  
 picture = forms.ImageField(label=**'Profilbild'**, required=*False*)  
  
 *class* Meta:  
 model = Profile  
 fields = [**'biography'**, **'picture'**]  
  
  
*class* MovieRatingForm(forms.ModelForm):  
  
 rating = forms.ChoiceField(choices = MovieRating.RATING\_CHOICES, label=**""**, initial=**''**, widget=forms.Select(), required=*False*)  
  
 *class* Meta:  
 model = MovieRating  
 fields = [**'rating'**]

## main/models.py

*from* django.db *import* models  
*from* django.db.models *import* Avg  
*from* django.contrib.auth.models *import* User  
*from* django.core.files.storage *import* FileSystemStorage  
*from* django.conf *import* settings  
*from* watson *import* search *as* watson  
*import* datetime  
  
fs = FileSystemStorage(location=settings.MEDIA\_ROOT)  
  
*class* Profile(models.Model):  
 user = models.OneToOneField(User, on\_delete=models.CASCADE)  
 picture = models.ImageField(storage=fs, blank=*True*)  
 biography = models.TextField(max\_length=3000, blank=*True*)  
  
 *def \_\_str\_\_*(self):  
 *return* **"Profile {email}"**.format(  
 email=self.user.email  
 )  
  
  
*class* Moderator(models.Model):  
 user = models.OneToOneField(User, on\_delete=models.CASCADE)  
 is\_moderator = models.BooleanField()  
  
 *def \_\_str\_\_*(self):  
 *return* self.user.email  
  
  
*class* Movies(models.Model):  
 name = models.CharField(max\_length=90)  
 description = models.TextField(max\_length=4000)  
 description\_short = models.CharField(max\_length=80)  
 logo = models.ImageField(storage=fs)  
 created\_at = models.DateField()  
 video\_url = models.CharField(max\_length=100, blank=*True*)  
  
 *def \_\_str\_\_*(self):  
 *return* **"Movie {name} - {year}"**.format(  
 name=self.name,  
 year=self.created\_at  
 )  
  
 *def* update\_movies\_index(instance, *\*\*kwargs*):  
 *for* movies *in* instance.movies\_set.all():  
 watson.default\_search\_engine.update\_obj\_index(movies)  
  
 @property  
 *def* rating(self):  
 p = MovieRating.objects.filter(movie=self).aggregate(Avg(**'rating'**))  
 p1 = str(p)  
 p2 = p1.split(**':'**)[1]  
 p2 = p2.split(**'}'**)[0]  
 *if* p2 == **' None'**:  
 *return* **'None'** *else*:  
 *return* p2  
  
 @property  
 *def* votes(self):  
 *return* MovieRating.objects.filter(movie=self).count()  
  
  
*class* FavMovies(models.Model):  
 user = models.OneToOneField(User, on\_delete=models.CASCADE)  
 movie = models.OneToOneField(Movies, on\_delete=models.CASCADE)  
  
 *class* Meta:  
 unique\_together = [**"movie"**, **"user"**]  
  
  
*class* MovieRating(models.Model):  
 NEUTRAL = 0  
 VERY\_BAD = 1  
 BAD = 2  
 OK = 3  
 GOOD = 4  
 VERY\_GOOD = 5  
 RATING\_CHOICES = (  
 (NEUTRAL, **'0'**),  
 (VERY\_BAD, **'1'**),  
 (BAD, **'2'**),  
 (OK, **'3'**),  
 (GOOD, **'4'**),  
 (VERY\_GOOD, **'5'**),  
 )  
 movie = models.ForeignKey(Movies, on\_delete=models.CASCADE)  
 user = models.ForeignKey(User, on\_delete=models.CASCADE)  
 rating = models.IntegerField(choices=RATING\_CHOICES, default=NEUTRAL)  
  
 *class* Meta:  
 unique\_together = [**"movie"**, **"user"**]  
  
  
*class* Comments(models.Model):  
 movie = models.ForeignKey(Movies, on\_delete=models.CASCADE)  
 user = models.ForeignKey(User, on\_delete=models.CASCADE)  
 comment = models.TextField(max\_length=3000, blank=*True*)  
 is\_moderatored = models.BooleanField()  
  
  
*class* MovieImage(models.Model):  
 movie = models.ForeignKey(Movies, on\_delete=models.CASCADE)  
 image = models.ImageField(blank=*True*, null=*True*)  
  
 *def \_\_str\_\_*(self):  
 *return* **"{image}"**.format(  
 image = self.image  
 )

## main/urls.py

*from* django.conf.urls *import* url, include  
  
*from* main.views *import* LatestMovieFeed  
*from* . *import* views  
  
urlpatterns = [  
 url(**r'^$'**, views.home, name=**'home'**),  
 url(**r'^movie/new'**, views.addMovie, name=**'addMovie'**),  
 url(**r'^movie/(?P<movie\_id>[0-9]+)/comment/(?P<comment\_id>[0-9]+)/edit'**, views.commentEdit, name=**'commentEdit'**),  
 url(**r'^movie/(?P<movie\_id>[0-9]+)/comment'**, views.comment, name=**'comment'**),  
 url(**r'^movie/(?P<movie\_id>[0-9]+)/media'**, views.addMoviesMedia, name=**'addMoviesMedia'**),  
 url(**r'^movie/(?P<movie\_id>[0-9]+)/like'**, views.like\_movie, name=**'like\_movie'**),  
 url(**r'^movie/(?P<movie\_id>[0-9]+)/unlike'**, views.unlike\_movie, name=**'unlike\_movie'**),  
 url(**r'^movie/(?P<movie\_id>[0-9]+)'**, views.showMovie, name=**'showMovie'**),  
 url(**r'^profiles/(?P<user\_id>[0-9]+)/contact'**, views.profile\_contact, name=**'profile\_contact'**),  
 url(**r'^profile/(?P<movie\_id>[0-9]+)/delete'**, views.unlike\_movie, name=**'unlike\_movie'**),  
 url(**r'^profile/(?P<movie\_id>[0-9]+)/rate'**, views.rate\_movie, name=**'rate\_movie'**),  
 url(**r'^profiles/(?P<user\_id>[0-9]+)'**, views.profile\_show, name=**'profile\_show'**),  
 url(**r'^profile/'**, views.profile\_edit, name=**'profile\_edit'**),  
 url(**r'^search/'**, include(**"watson.urls"**, namespace=**"watson"**)),  
 url(**r'^imprint/'**, views.imprint, name=**'imprint'**),  
 url(**r'^legal/'**, views.legal, name=**'legal'**),  
 url(**r'^rss/movie/$'**, LatestMovieFeed()),  
]

## main/views.py

*from* allauth.account.decorators *import* verified\_email\_required  
*from* django.contrib.auth.decorators *import* login\_required  
*from* django.forms *import* inlineformset\_factory, FileInput  
*from* django.contrib.syndication.views *import* Feed  
*from* django.shortcuts *import* render, redirect, get\_object\_or\_404  
*from* django.contrib *import* messages  
*from* main.forms *import* MovieForm, ContactForm, ProfileEditForm, CommentsForm, CommentsEditForm, MovieRatingForm  
*from* main.models *import* Movies, MovieImage, FavMovies, Comments, Genre, MovieGenre, MovieRating  
*from* django.contrib.auth.models *import* User  
*from* django.template.loader *import* render\_to\_string  
*from* django.core.mail *import* EmailMessage  
*from* django.core.urlresolvers *import* reverse  
  
  
  
*def* home(*request*):  
 """  
 The Homepage  
 """  
 context = {  
 **"movies"**: Movies.objects.all(),  
 }  
 *return* render(request=*request*, template\_name=**'main/home.html'**, context=context)  
  
  
*def* showMovie(*request*, *movie\_id*):  
 *if not request*.user.is\_authenticated() *or not request*.user.moderator.is\_moderator:  
 movie = get\_object\_or\_404(Movies, id=*movie\_id*)  
 context = {  
 **'movie'**: movie,  
 **'comment\_form'**: CommentsForm(),  
 **'fav'**: FavMovies.objects.filter(user\_\_id=*request*.user.id, movie\_\_id=*movie\_id*),  
 **'rating'**: MovieRatingForm(),  
 }  
 *return* render(*request*, template\_name=**'main/movie\_show.html'**, context=context)  
 *else*:  
 *if request*.user.moderator.is\_moderator:  
 movie = get\_object\_or\_404(Movies, id=*movie\_id*)  
 context = {  
 **'movie'**: movie,  
 **'comment\_form'**: CommentsForm(),  
 **'comment\_edit'**: CommentsEditForm(),  
 **'fav'**: FavMovies.objects.filter(user\_\_id=*request*.user.id, movie\_\_id=*movie\_id*),  
 **'rating'**: MovieRatingForm(),  
 }  
 *return* render(*request*, template\_name=**'main/movie\_show.html'**, context=context)  
  
  
*def* profile\_show(*request*, *user\_id*):  
 """  
 Profile page  
 """  
 user = get\_object\_or\_404(User, id=*user\_id*)  
 context = {  
 **'user'**: user,  
 **'contactform'** : ContactForm(),  
 **'fav'**: FavMovies.objects.filter(user\_\_id=*user\_id*)  
 }  
 *return* render(*request*, **'main/profile\_show.html'**, context)  
  
  
*def* profile\_contact(*request*, *user\_id*):  
 *if request*.method == **'POST'**:  
 user = User.objects.get(id=*user\_id*)  
 contact\_form = ContactForm(*request*.POST)  
 print(**'1'**)  
 *if* contact\_form.is\_valid():  
 body = render\_to\_string(  
 template\_name=**'email/contactmessage.txt'**,  
 context={  
 **'name'**: user.get\_full\_name(),  
 **'sender'**: contact\_form.cleaned\_data[**'name'**],  
 **'sender\_email'**: contact\_form.cleaned\_data[**'email'**],  
 **'message'**: contact\_form.cleaned\_data[**'message'**]  
 }  
 )  
 email = EmailMessage(  
 subject = contact\_form.cleaned\_data[**'subject'**],  
 body = body,  
 to = [user.email, contact\_form.cleaned\_data[**'email'**]],  
 from\_email = **'python@imdb.advanced'** )  
 email.send()  
 messages.success(*request*, **"E-Mail erfolgreich gesendet!"**)  
 *return* redirect(**"profile\_show"**, *user\_id*)  
 *else*:  
 messages.error(*request*, **"Ihre E-Mail konnte nicht gesendet werden!"**)  
 *return* render(*request*, **'main/movie\_new.html'**, {**'contact\_form'**: contact\_form})  
 *else*:  
 print(**'2'**)  
 *return* redirect(**"profile\_show"**, *user\_id*)  
  
  
@login\_required  
*def* profile\_edit(*request*):  
 """  
 Profile Edit Page  
 """  
 *if request*.method == **'POST'**:  
 profileedit\_form = ProfileEditForm(*request*.POST, *request*.FILES, instance=*request*.user.profile)  
 *if* profileedit\_form.is\_valid():  
 profileedit\_form.save()  
 messages.success(*request*, **'Ihre Änderungen wurden erfolgreich gespeichert.'**)  
 *return* redirect(**'profile\_edit'**)  
 *else*:  
 print(profileedit\_form.errors.as\_data())  
 messages.error(*request*, **'Fehler beim Speichern Ihrer Änderungen.'**)  
 *return* redirect(**'profile\_edit'**)  
 *else*:  
 context = {  
 **'profileedit\_form'**: ProfileEditForm(instance=*request*.user.profile),  
 **'fav'**: FavMovies.objects.filter(user\_\_id=*request*.user.id),  
 }  
 *return* render(request=*request*, template\_name=**'main/profile\_edit.html'**, context=context)  
  
  
@verified\_email\_required  
*def* addMovie(*request*):  
 *if request*.method == **'POST'**:  
 movie\_form = MovieForm(*request*.POST, *request*.FILES)  
 *if* movie\_form.is\_valid():  
 movie = Movies.objects.create(  
 name = movie\_form.cleaned\_data[**'name'**],  
 description = movie\_form.cleaned\_data[**'description'**],  
 description\_short = movie\_form.cleaned\_data[**'description\_short'**],  
 logo = *request*.FILES[**'logo'**],  
 created\_at = movie\_form.cleaned\_data[**'created\_at'**],  
 video\_url = movie\_form.cleaned\_data[**'video\_url'**]  
 )  
 messages.success(*request*, **'Film wurde erfolgreich angelegt.'**)  
 *return* redirect(**'addMoviesMedia'**, movie.id)  
 *else*:  
 messages.error(*request*, **'Fehler beim Speichern des Filmes. Bitte überprüfen Sie alle Felder.'**)  
 *return* render(*request*, **'main/movie\_new.html'**, {**'movie\_form'**: movie\_form})  
 *else*:  
 context = {  
 **'movie\_form'**: MovieForm(),  
 }  
 *return* render(request=*request*, template\_name=**'main/movie\_new.html'**, context=context)  
  
  
*def* addMoviesMedia(*request*, *movie\_id*):  
 movie = get\_object\_or\_404(Movies, id=*movie\_id*)  
 ImageFormSet = inlineformset\_factory(Movies, MovieImage, fields=(**'image'**,), widgets={**'image'**: FileInput()},  
 can\_delete=*True*, extra=1)  
 *if request*.method == **'POST'**:  
 formset = ImageFormSet(*request*.POST, *request*.FILES, instance=movie)  
 *if* formset.is\_valid():  
 formset.save()  
 messages.success(*request*, **'Bilder wurden erfolgreich Hinzugefügt.'**)  
 *return* redirect(**'showMovie'**, movie.id)  
 *else*:  
 messages.error(*request*, **'Fehler beim Speichern des Filmes. Bitte überprüfen Sie alle Felder.'**)  
 *return* render(*request*, **'main/movie\_media.html'**, {**'formset'**: formset})  
 *else*:  
 context = {  
 **'movie'**: movie,  
 **'formset'**: ImageFormSet(instance=movie),  
 }  
 *return* render(request=*request*, template\_name=**'main/movie\_media.html'**, context=context)  
  
@login\_required  
*def* comment(*request*, *movie\_id*):  
 movie = get\_object\_or\_404(Movies, id=*movie\_id*)  
 *if request*.method == **'POST'**:  
 comment\_form = CommentsForm(*request*.POST)  
 *if* comment\_form.is\_valid():  
 Comments.objects.create (  
 comment = comment\_form.cleaned\_data[**'comment'**],  
 is\_moderatored = *False*,  
 user = *request*.user,  
 movie = movie  
 )  
 messages.success(*request*, **'Erfolgreich kommentiert.'**)  
 *return* redirect(**'showMovie'**, movie.id)  
 *else*:  
 messages.error(*request*, **'Fehler beim Speichern Ihres Kommentares.'**)  
 *return* render(*request*, **'main/movie\_show.html'**, {**'comment\_form'**: comment\_form})  
 *else*:  
 *return* redirect(**'showMovie'**, *movie\_id*)  
  
  
@login\_required  
*def* commentEdit(*request*, *movie\_id*, *comment\_id*):  
 movie = get\_object\_or\_404(Movies, id=*movie\_id*)  
 *if request*.method == **'POST'**:  
 comment\_form = CommentsForm(*request*.POST)  
 comment = get\_object\_or\_404(Comments, id=*comment\_id*)  
 *if* comment\_form.is\_valid():  
 comment.comment = comment\_form.cleaned\_data[**'comment'**]  
 comment.is\_moderatored = *True* comment.save()  
 messages.success(*request*, **'Erfolgreich editiert.'**)  
 *return* redirect(**'showMovie'**, movie.id)  
 *else*:  
 messages.error(*request*, **'Fehler beim Editieren Ihres Kommentares.'**)  
 *return* render(*request*, **'main/movie\_show.html'**, {**'comment\_form'**: comment\_form})  
 *else*:  
 *return* redirect(**'showMovie'**, *movie\_id*)  
  
  
@login\_required  
*def* rate\_movie(*request*, *movie\_id*):  
 movie = get\_object\_or\_404(Movies, id=*movie\_id*)  
 *if request*.method == **'POST'**:  
 rating\_form = MovieRatingForm(*request*.POST)  
 *if* rating\_form.is\_valid():  
 *if request*.user *in* [movierating.user *for* movierating *in* movie.movierating\_set.all()]:  
 messages.error(*request*, **'Sie haben schon eine Bewertung abgegeben.'**)  
 *return* redirect(**'showMovie'**, movie.id)  
 *else*:  
 MovieRating.objects.create(  
 movie = movie,  
 user = *request*.user,  
 rating = rating\_form.cleaned\_data[**'rating'**]  
 )  
 messages.success(*request*, **'Erfolgreich bewertet.'**)  
 *return* redirect(**'showMovie'**, movie.id)  
 *else*:  
 *return* redirect(**'showMovie'**, movie.id)  
 *else*:  
 *return* redirect(**'showMovie'**, movie.id)  
  
  
@login\_required  
*def* unlike\_movie(*request*, *movie\_id*):  
 *try*:  
 movie = get\_object\_or\_404(Movies, id=*movie\_id*)  
 fav = FavMovies.objects.get(user=*request*.user, movie=movie)  
 fav.delete()  
 messages.success(*request*, **"Sie favorisieren den Film nicht mehr."**)  
 *return* redirect(**'showMovie'**, *movie\_id*)  
 *except*:  
 messages.error(*request*, **"Ein Fehler ist aufgetretten, bitte versuchen Sie es noch einmal."**)  
 *return* redirect(**'showMovie'**, *movie\_id*)  
  
  
@login\_required  
*def* like\_movie(*request*, *movie\_id*):  
 *try*:  
 movie = get\_object\_or\_404(Movies, id=*movie\_id*)  
 FavMovies.objects.create(  
 user=*request*.user,  
 movie=movie  
 )  
 messages.success(*request*, **"Sie haben den Film erfolgreich favorisiert."**)  
 *return* redirect(**'showMovie'**, *movie\_id*)  
 *except*:  
 messages.error(*request*, **"Ein Fehler ist aufgetretten, bitte versuchen Sie es noch einmal."**)  
 *return* redirect(**'showMovie'**, *movie\_id*)  
  
  
*def* imprint(*request*):  
 *return* render(request=*request*, template\_name=**'main/imprint.html'**)  
  
  
*def* legal(*request*):  
 *return* render(request=*request*, template\_name=**'main/legal.html'**)  
  
  
*class* LatestMovieFeed(Feed):  
 title = **"Neuesten Filme"** link = **"/rss/"** description = **"Die neuesten Filme werden hier angezeigt."** *def* items(self):  
 *return* Movies.objects.order\_by(**'-created\_at'**)[:5]  
  
 *def* item\_title(self, *item*):  
 *return item*.name  
  
 *def* item\_description(self, *item*):  
 *return item*.description  
  
 *def* item\_link(self, *item*):  
 *return* reverse(**'showMovie'**, args=[*item*.pk])

## manage.py

#!/usr/bin/env python  
*import* os  
*import* sys  
  
*if* \_\_name\_\_ == **"\_\_main\_\_"**:  
 os.environ.setdefault(**"DJANGO\_SETTINGS\_MODULE"**, **"PythonIMDBAdvanced.settings"**)  
  
 *from* django.core.management *import* execute\_from\_command\_line  
  
 execute\_from\_command\_line(sys.argv)