PYTHONIMDB ADVANCED

INDIVIDUELLES PROJEKT

Alexandros Konstantinidis | inf2260 | Matrikel-Nr.: 669451

Dozent: Herr Mahler
Datum:20.01.2017

INHALTSVERZEICHNIS

INHALTSVERZEICHNIS

INHALTSVERZEICHNIS

Abbildungsverzeichnis

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.

PYTHONIMDBADVANCED

PROJEKTSTRUKTUR

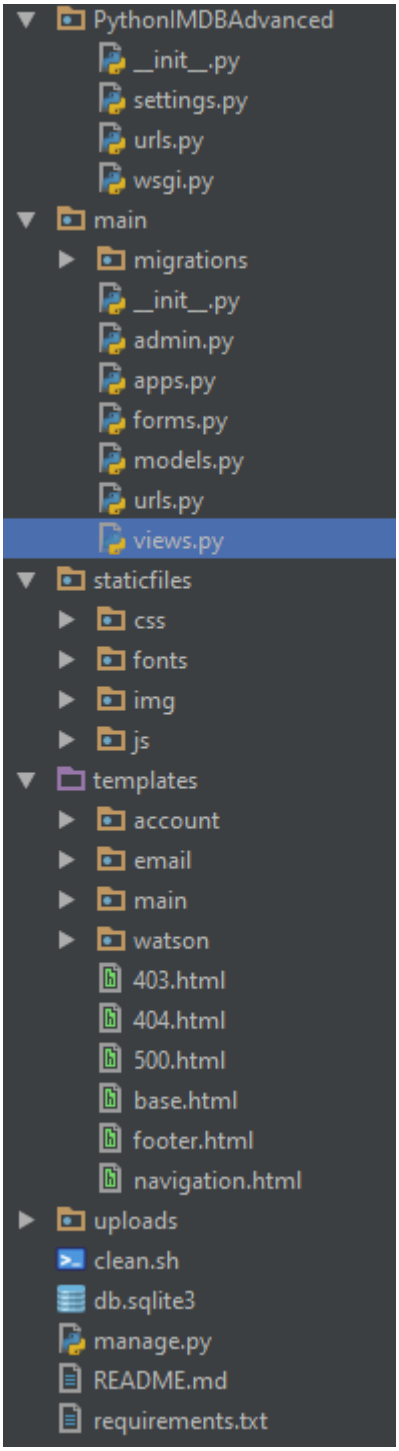

Abbildung 1: Projektstruktur

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.

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

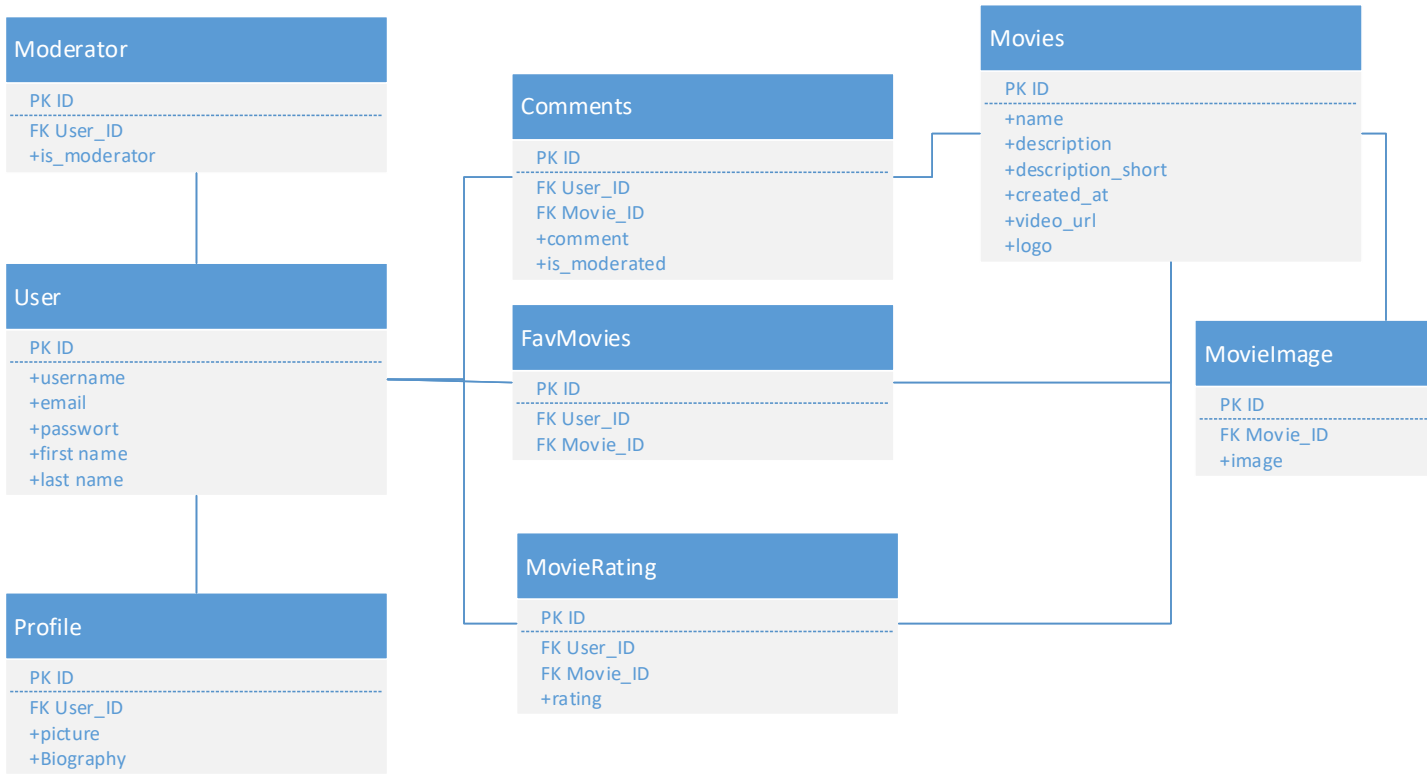

Abbildung 2: Datenbankstruktur

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)
```