Movie Metadata Database Documentation

Release 1.0

Christoph Piechula, #02746310 Dozent: Prof. Dr. Horst Heineck

July 01, 2013

KAPITEL 1

VORWORT

1.1 Abweichungen zur ersten Spezifikation

Die aktuelle Dokumentation enthält Abweichungen zu der erst vorgestellten Spezifikation. Diese Abweichungen sind aufgrund der vorliegenden Metadaten entstanden, hier musste für die Datenerhebung auf den Open Source Scraper vom XBMC [1] zurückgegriffen werden, welcher ein anderes Metadaten Set wie erwartet geliefert hat (siehe *Datenerhebung* (Seite 8)).

1.2 Umsetzung der geforderten Bestandteile der Anwendung

Es folgt nun eine Liste der Anforderungen, welche aus der Aufgabenstellung übernommen wurde. Unter jedem Punkt findet sich eine kurze Beschreibung oder Referenz auf ein Kapitel welches beschreibt wie diese in der Studienarbeit umgesetzt wurden.

Berichte und Forms:

Reports sind in der Anwendung sehr häufig enthalten. Die Movie Suche sowie Artist Suche und Darstellungen von Movie Covern und Movie Fanart wurde mit Reports realisiert. Auch die Favoriten Listen sind mittels Reports/Forms realisiert. Siehe hierzu beispielsweise:

- *Movie Search Region* (Seite 19)
- *Actor Search Region* (Seite 26)
- *Actor known for Region* (Seite 28)
- *Your Favourites Region* (Seite 31)

Schaltflächen:

Buttons wurden zur besseren Navigation auf vielen Seiten verwendet. Siehe hierzu beispielsweise:

- *Movie Details Navigation/Breadcrumb Region und Previous Button* (Seite 24)
- *Movieart Navigation/Breadcrumb und Previous Button* (Seite 26)
- *Actor Details Navigation/Breadcrumbs und Previous Button* (Seite 29)
- *Show Movieart Button* (Seite 23)
- *Favourites Add Button* (Seite 24)

Seitenverzweigungen:

[1] http://www.xbmc.org

Die Anwendung macht starken Gebrauch von Seitenverzweigungen. Wenn man bei der Film Suche einen Film anklickt wird man auf die Movie Details Seite weitergeleitet. Wenn man einen Actor in der Actor Suche anklickt wird man auf die Actor Details Seite weitergeleitet. Klickt man unter Movie Details den *show Movieart* Button an, wird man auf die Movieart Seite weitergeleitet.

Berechnungen:

Computations werden verwendet um beispielsweise die Shared Components *MOVIE_ID* und *ACTOR_ID* auf sinnvolle Standardwerte zu initialisieren. Siehe hierzu:

- *Computations and Shared Components* (Seite 18)

Registerkarten:

Registerkarten werden verwendet um die verschiedenen Bereiche/Seiten sinnvoll abzugrenzen. Siehe hier beispielsweise die Übersichtsscreenshots der jeweiligen Bereiche Movies, Actors, Statistics und Favourites.

- *Movies Page* (Seite 19)
- *Actors Page* (Seite 26)
- *Statistics Page* (Seite 29)
- *Favourites Page* (Seite 30)

Navigationsmöglichkeiten:

Die Applikation bietet verschiedene Möglichkeiten der Navigation.

- Navigationsbuttons und Breadcrumbs, siehe
  - *Movie Details Navigation/Breadcrumb Region und Previous Button* (Seite 24)
  - *Movieart Navigation/Breadcrumb und Previous Button* (Seite 26)
  - *Actor Details Navigation/Breadcrumbs und Previous Button* (Seite 29)
- Verlinkungen in den jeweiligen Reports Movie -> Movie Details, Actor -> Actor Details, etc...
- Navigation über Main-Header

Bilder:

Die Applikation setzt sehr stark auf Bilder als Komponenten. Siehe hierzu beispielsweise die Detail Pages Movie Details, Actor Details oder die Movieart Seite:

- *Movieart Page* (Seite 25)
- *Movie Cover Region* (Seite 21)
- *Actor Photo Region* (Seite 28)
- *Cover und Fanart Region* (Seite 25)

Diagramme:

Diagramme werden verwendet um Actor Statistiken und Movie Statistiken zu erstellen, siehe hier:

- *Top 15 Actors Chart Region* (Seite 27)
- *Movie by country Chart Region* (Seite 29)

- *Movie by genre Chart Region* (Seite 30)

Repräsentative Anzahl von Daten:

Um eine möglichst große und Repräsentative Datenmenge zu erhalten wurden alle Filme unserer Studenten WG eingelesen und in die Datenbank eingepflegt. Die Datensatz Menge an Filmen beträgt 519, die Schauspieler Menge liegt bei 5033. Siehe hierzu auch:

- *Datenerhebung* (Seite 8)

Tooptips:

Tooltips werden auf verschiedenen Buttons verwendet um dem Benutzer eine Hilfestellung zu geben. Siehe hierzu beispielsweise:

- *Favourites Add Button* (Seite 24)
- *Movie Details Navigation/Breadcrumb Region und Previous Button* (Seite 24)
- *Movieart Navigation/Breadcrumb und Previous Button* (Seite 26)
- *Actor Details Navigation/Breadcrumbs und Previous Button* (Seite 29)
- *Show Movieart Button* (Seite 23)
- *Favourites Add Button* (Seite 24)

Wertelisten:

Wertelisten werden verwendet um einen Film auf der Movie Details Seite zu einer vom Benutzer selektierten Favoriten Liste hinzuzufügen. Siehe hierzu:

- *Favourites Select List* (Seite 24)

Forms:

Forms werden verwendet um neue Favoriten Listen zu erstellen. Siehe hierzu:

- *Add or Delete Form Region* (Seite 31)

Validierungen:

Validierungen werden verwendet um sicherzustellen, daß keine Favoriten Listen mit dem gleichen Namen eingepfegt werden können. Siehe hierzu:

- *Form Validation FAV_NAME* (Seite 31)

KAPITEL 2

ZUGANGSDATEN

Folgend die Zugangsdaten für die Movie Metadata Applikation:

Application ID	389
Workspace ID	APEX_10225
Benutzername	Ora01
Passwort	katzenbaumx

Direkter Link zur Applikation: https://redcluster-scan.fh-hof.de:4443/pls/apex/f?p=389::4148785593919

Bei Fragen oder Problemen erreichen Sie mich unter der FH E-Mail Adresse:

cpiechula@hof-university.de

KAPITEL 3

SPEZIFIKATION

3.1 Funktionsumfang

Im Rahmen der Studienarbeit Entwicklung von Webanwendungen soll eine Apex Applikation realisiert werden. Das gewählte Thema umfasst eine kleine Film Metadaten Datenbank welche Freunden online zugänglich gemacht werden kann um den nächsten gemeinsamen Filmeabend zu planen.

Die Film-Datenbank soll die grundlegende Metadaten verwalten können und folgende Funktionalität bieten:

- Film Suchfunktion
- Anzeige von Film Metadaten
- Anzeige von Film Covern und Fanart
- Actor Suchfunktion
- Actor Photos
- Auflistung welcher Schauspieler in welchen Filmen mitspielt
- Statistiken zu in der Datenbank befindlichen Daten
- Erstellen von Favoriten Listen

3.2 Tabellen vor der Normalisierung

Folgende Daten liegen zur Verarbeitung vor:

Movie	**Actor**	**Director**	**Credits**	**Favourites**
id	id	id	id	id
imdbid	name	name	name	name
title	photo			
originaltitle	rollenname			
rating				
country				
genre				
runtime				
rating				
year				
cover				
fanart				

KAPITEL 4

ENTWURF

4.1 Übersicht vom Seitenaufbau

Folgende Seiten mit den gelisteten Funktionalitäten sollen umgesetzt werden:

Movies Page

- Film Suchfunktion
- Anzeige der Movie Top/Flop 10 aus Datenbank

Movie Detail Page

Diese Seite soll erscheinen sobald ein bestimmter Film ausgewählt wurde

- Anzeige vom Movie Cover
- Verlinkung auf Seite mit alternativ Covern und Film Fanart
- Anzeige der Genre
- Anzeige des Regisseurs
- Anzeige der Drehbuchautoren
- Anzeige vom Film Metadaten wie Titel, Originaltitel, Jahr, Spielzeit, Rating, Anzahl der Votes
- Anzeige vom Film-Plot
- Anzeige von Schauspielern die in dem Film mitspielen + Rollenname
- Anzeige ob Film in einer Favoriten Liste ist
- Möglichkeit Film in eine bestehende Favoriten Liste hinzuzufügen

Movieart Page

Diese Seite soll über die Movie Details Page verlinkt werden

- Anzeige von alternativ Covern des jeweiligen Films
- Anzeige von Fanart des jeweiligen Films

Actors Page

- Actor Suchfunktion
- Anzeige der Top 15 Schauspieler als Statistik
- Anzeige der Top 10 Schauspieler Photos

Actor Detail Page

Diese Seite soll erscheinen wenn ein bestimmter Schauspieler ausgewählt wurde

- Anzeige des Schauspieler Fotos und Anzeige in welchen Filmen dieser mitspielt

Statistics Page

- Statistik über Anzahl von Filmen pro Herstellungsland
- Statistik über Anzahl von Filmen pro Genre

Favourites Page

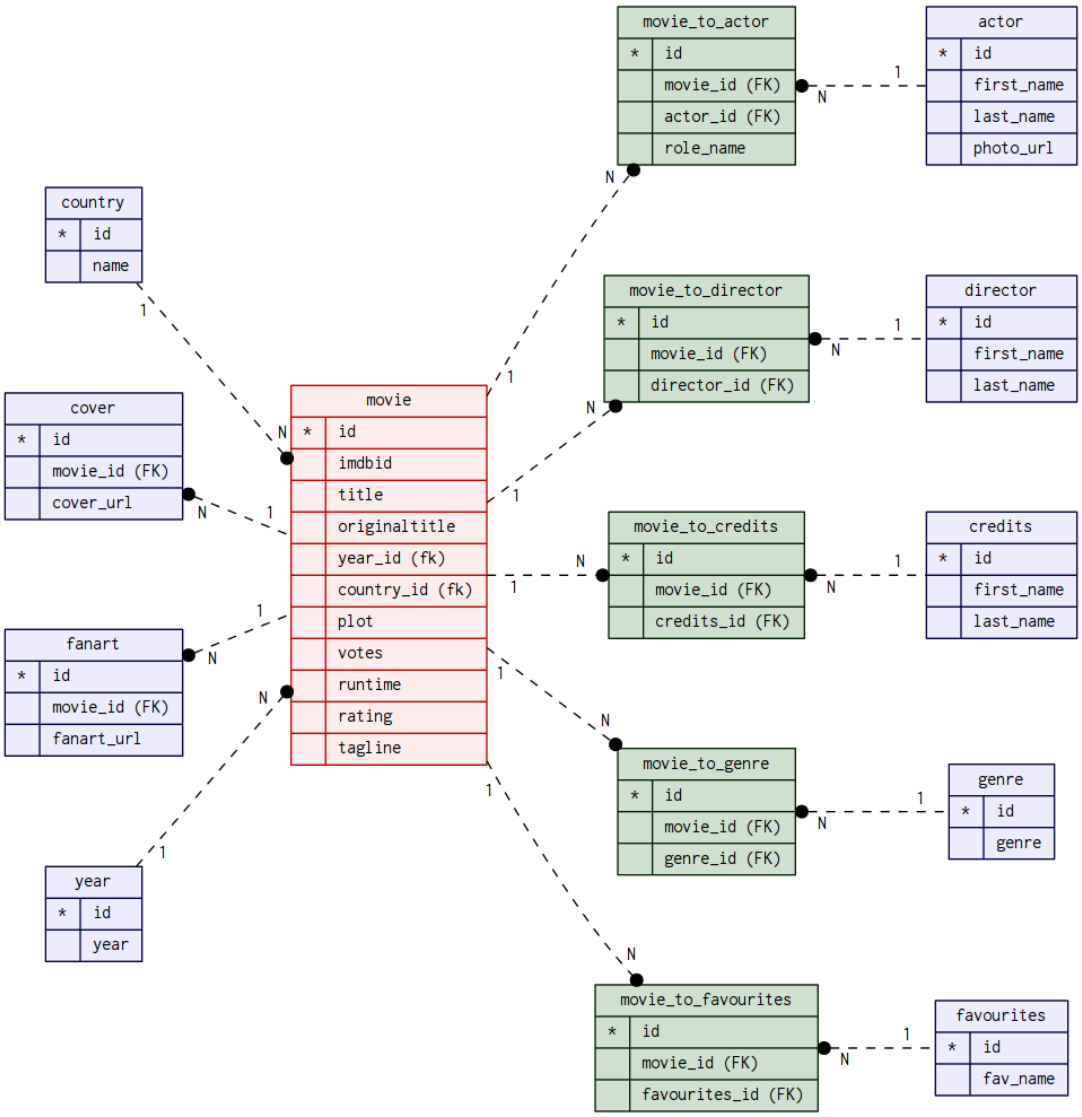
- Anzeige der aktuellen Favoriten Listen mit den jeweiligen Filmen die in diesen enthalten sind
- Möglichkeit der Erstellung einer neuen Favoriten Liste

4.2 Normalisierung

Durch die Normalisierung wurde die Movie Tabelle stark entschlackt und Zwischentabellen eingeführt. Die Actor, Credits und Director Tabellen wurden in atomare Einheiten zerlegt. Zusätzlich ist hier noch eine Favoriten Tabelle dazu gekommen. Die Folgende Grafik zeigt das normalisierte Datenbankschema, die Grafik wurde mit dem Java Open Source Tool erwiz [1] erstellt.

[1] http://www.erwiz.de

4.3 Normalisiertes Datenbankschema



4.4 Datenerhebung

Um eine *repräsentative* Datenmenge zu bekommen wurden alle Filme unserer Studenten WG mit dem XBMC Media Center [2] eingelesen um die Metadaten über den dort eingebauten Metadaten Scraper zu sammeln.

Das XBMC hat pro Film eine XML-Datei mit folgendem Inhalt erstellt (verkürzte Darstellung):

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<movie>
    <title>Pulp Fiction</title>
    <originaltitle>Pulp Fiction</originaltitle>
    <rating>9.000000</rating>
    <year>1994</year>
    <votes>764,746</votes>
```

[2] http://www.xbmc.org

```
    <plot>Vincent Vega und Jules Winnfield holen für ihren Boss Marsellus
        Wallace eine schwarze Aktentasche aus einer Wohnung [...]</plot>
    <tagline>Drei schwere Jungs und ein Flittchen [...] </tagline>
    <runtime>154</runtime>
    <thumb aspect="poster" preview="http://cf2.imgobject.com/Y8WI.jpg">
        http://cf2.imgobject.com/Y8WI.jpg
    </thumb>
    ... 1 - n
    <fanart>
        <thumb preview="http://cf2.imgobject.com/IAGy.jpg">
            http://cf2.imgobject.com/IAGy.jpg
        </thumb>
        ... 1 - n
    </fanart>
    <id>tt0110912</id>
    <genre>Krimi</genre>
    <genre>Thriller</genre>
    <country>United States of America</country>
    <credits>Quentin Tarantino</credits>
    <credits>Roger Avary</credits>
    <director>Quentin Tarantino</director>
    <actor>
        <name>Bruce Willis</name>
        <role>Butch Coolidge</role>
        <thumb>
            http://cf2.imgobject.com/TbRc5sCQGi8a.jpg
        </thumb>
    </actor>
    ... 1 - n
</movie>
```

Die Metadaten Elemente sind selbsterklärend. Aus diesen Metadaten ergibt sich das Daten-Set, aufgrund dessen die erste Spezifikation überarbeitet werden musste. Insgesamt wurden in die Datenbank 519 Filme und 5033 Schauspieler eingepflegt.

Zur Auswertung der im XML-Format vorliegenden Dateien wurde ein Python Script geschrieben. Dieses extrahiert die benötigten Daten und generiert CSV Dateien welche von der Datenbank importiert werden können. Folgend das Script zum erstellen der CSV Dateien:

```
#!/usr/bin/env python
# encoding: utf-8

"""xbmc2apex

Usage:
  xbmc2apex.py -d PATH
  xbmc2apex.py (-h | --help)
  xbmc2apex.py --version

Options:
  -h --help     Show this screen.
  --version     Show version.

"""

"""
Converts XBMC XML Metadata Files to a set of CSV files
```

```
19
20  Author: Christoph Piechula
21  """
22
23  from xmltodict import parse as dict_from_xml
24  from docopt import docopt
25  import csv
26  import os
27
28
29  class Movie:
30      """
31      Wrapper object around xbmc xml files to retrieve metadata easier
32      """
33      def __init__(self, data):
34          self._xmldict = data
35          self._no_image = "http://up.nullcat.de/noimage.jpg"
36          self._art = ['thumb', 'fanart']
37          self._listitem = ['director', 'actor', 'credits', 'genre', 'country']
38          self._singleitem = ['title', 'originaltitle', 'year', 'id',
39                              'runtime', 'plot', 'rating', 'votes']
40
41      def _get_single_value(self, tag):
42          """
43          returns single value for given tag
44          """
45          retv = []
46          try:
47              if tag == 'fanart':
48                  retv = self._xmldict.get('movie').get(tag).get('thumb')
49              else:
50                  retv = self._xmldict.get('movie').get(tag)
51          except AttributeError:
52              pass
53          return retv or []
54
55      def _get_value_list(self, tag):
56          """
57          returns a list with values for given tag
58          """
59          retv = []
60          item = self._get_single_value(tag)
61          if item:
62              if not isinstance(item, list):
63                  item = [item]
64              for item in item:
65                  item = self._sanitize_string(item)
66                  retv.append(item)
67          return retv
68
69      def _get_actor_list(self):
70          """
71          cleans and creates actor list
72          returns actor list with tuples (name, role, thumb)
73          """
74          retv = []
75          actors = self._sanitize(self._get_single_value('actor') or [])
76          for actor in actors or []:
```

```
77              data = actor['name'], actor['role'], actor['thumb']
78              retv.append(data)
79          return retv
80
81      def _get_fanart(self):
82          """
83          returns movie fanart as a list of fanart urls
84          """
85          artlist = []
86          col = self._get_single_value('fanart') or []
87          if isinstance(col, list):
88              for item in col:
89                  artlist.append(item['@preview'])
90          else:
91              for item in col.items():
92                  if item[0] == '@preview':
93                      artlist.append(item[1])
94          return artlist
95
96      def _get_thumb(self):
97          """
98          returns movie coverart as a list of coverart urls
99          """
100         artlist = []
101         col = self._get_single_value('thumb') or []
102         if isinstance(col, list):
103             for item in col:
104                 artlist.append(item['@preview'])
105         else:
106             for item in col.items():
107                 if item[0] == '@preview':
108                     artlist.append(item[1])
109         return artlist
110
111     def _sanitize(self, collection):
112         """
113         cleans dirty actor collection, replaces unavaiable photo urls
114         with specified no photo url and returns cleand list
115         """
116         if not isinstance(collection, list):
117             collection = [collection]
118         for actor in (collection or []):
119             for k, v in actor.items():
120                 if v is None:
121                     actor[k] = self._no_image
122                 else:
123                     actor[k] = self._sanitize_string(actor[k])
124         return collection
125
126     def _sanitize_string(self, string):
127         """
128         Cleans comma in given string
129         """
130         if string:
131             return string.replace("'", "")
132         return ''
133
```

```
134     def is_valid(self):
135         """
136         checks internal movie attibutes
137         """
138         for item in self._singleitem + self._listitem:
139             if not self._get_single_value(item):
140                 return False
141         return True
142 
143     def __getitem__(self, tag):
144         if tag in self._singleitem or tag == 'tagline':
145             return self._get_single_value(tag)
146         elif tag in self._listitem:
147             if 'actor' in tag:
148                 return self._get_actor_list()
149             else:
150                 return self._get_value_list(tag)
151         elif tag in self._art:
152             if tag == 'fanart':
153                 return self._get_fanart()
154             else:
155                 return self._get_thumb()
156         else:
157             raise KeyError('Unknown attribute:', tag)
158 
159 #
160 # Helper funcions
161 #
162 
163 MOVIE = ['id', 'imdbid', 'title', 'originaltitle', 'year_id', 'country_id',
164          'plot', 'votes', 'runtime', 'rating', 'tagline']
165 ACTOR = ['id', 'first_name', 'last_name', 'photo_url']
166 YEAR = ['id', 'year']
167 DIRECTOR = ['id', 'first_name', 'last_name']
168 CREDITS = ['id', 'first_name', 'last_name']
169 COUNTRY = ['id', 'name']
170 GENRE = ['id', 'name']
171 COVER = ['movie_id', 'cover_url']
172 FANART = ['movie_id', 'fanart_url']
173 
174 COUNTRYTOMOVIE = ['country_id', 'movie_id']
175 YEARTOMOVIE = ['year_id', 'movie_id']
176 
177 MOVIETOACTOR = ['movie_id', 'actor_id', 'role_name']
178 MOVIETOCREDITS = ['movie_id', 'credits_id']
179 MOVIETODIRECTOR = ['movie_id', 'director_id']
180 MOVIETOGENRE = ['movie_id', 'genre_id']
181 
182 
183 TITLES = {'movie': MOVIE,
184           'actor': ACTOR,
185           'year': YEAR,
186           'director': DIRECTOR,
187           'credits': CREDITS,
188           'country': COUNTRY,
189           'genre': GENRE,
190           'cover': COVER,
```

```
            'fanart': FANART,
            'countrytomovie': COUNTRYTOMOVIE,
            'yeartomovie': YEARTOMOVIE,
            'movietoactor': MOVIETOACTOR,
            'movietodirector': MOVIETODIRECTOR,
            'movietocredits': MOVIETOCREDITS,
            'movietogenre': MOVIETOGENRE}


def get_xmlfile(path):
    """
    returns xml file with movie metadata if avaiable at given path
    """
    for item in path:
        if item.lower().endswith('nfo'):
            return item


def read_file(path):
    with open(path, 'r') as f:
        return f.read()


def get_dictlist_from_movie(path):
    """
    creates a movie metadata list from given path
    """
    movielist = []
    for item in os.walk(path):
        xmlfile = get_xmlfile(item[2])
        if xmlfile:
            movie = Movie(
                dict_from_xml(
                    read_file(
                        os.path.join(
                            item[0], xmlfile
                        )
                    )
                )
            )
            if movie.is_valid():
                movielist.append(movie)
    return movielist


def create_movie_table(movielist):
    """
    creates movie dict from given movielist
    """
    movies = list()
    dupes = []
    rowid = 1
    for movie in movielist:
        items = {'imdbid': movie['id'],
                 'title': movie['title'],
                 'originaltitle': movie['originaltitle'],
                 'year': movie['year'],
```

```
                    'country': movie['country'],
                    'runtime': movie['runtime'],
                    'rating': movie['rating'],
                    'actor': movie['actor'],
                    'credits': movie['credits'],
                    'genre': movie['genre'],
                    'votes': movie['votes'],
                    'thumb': movie['thumb'],
                    'fanart': movie['fanart'],
                    'tagline': movie['tagline'],
                    'plot': movie['plot'],
                    'director': movie['director']
                    }
            items['votes'] = items['votes'].replace(',', '')
            if items['imdbid'] not in dupes:
                items.setdefault('rowid', rowid)
                movies.append(items)
                rowid += 1
                dupes.append(items['imdbid'])
    return movies


def create_n_to_m_table(movielist, tag):
    dlist = []
    tmp = []
    tmp_item = []
    for movie in movielist:
        items = movie[tag]
        if isinstance(items, str):
            items = [items]
        for item in items:
            if tag == 'actor':
                tmp_item = item[0], item[2]
                item = item[0], item[1], item[2]
                if tmp_item not in tmp:
                    tmp.append(tmp_item)
                    dlist.append(item)
            else:
                if item not in dlist:
                    dlist.append(item)

    return {k: v for k, v in enumerate(dlist, 1)}


def create_1ton_table(movielist, tag):
    dlist = []
    for movie in movielist:
        items = movie[tag]
        for item in items:
            item = [movie['rowid'], item]
            if item not in dlist:
                    dlist.append(item)
    return dlist


def get_mapping_table(movies, tag):
    tabletotable = []
```

```
305     c = create_table_for(movielist, tag)
306     for movie in movies:
307         for item in movie[tag]:
308             for k, v in c.items():
309                 if tag == 'actor':
310                     if item[0] == v[0]:
311                         tabletotable.append([movie['rowid'], k, item[1]])
312                 elif item == v:
313                     tabletotable.append([movie['rowid'], k])
314     return tabletotable
315
316
317 def create_table_for(movielist, tag):
318     if tag == 'fanart' or tag == 'thumb':
319         return create_1ton_table(movielist, tag)
320     elif tag == 'movie':
321         return create_movie_table(movielist)
322     else:
323         return create_n_to_m_table(movielist, tag)
324
325
326 def create_nto1_table(movielist, fklist, tag):
327     fk_list = []
328     for movie in movielist:
329         for fk in fklist.items():
330             if tag == 'country':
331                 tmp = [fk[1]]
332             else:
333                 tmp = fk[1]
334             if movie[tag] == tmp:
335                 movie[tag] = fk[0]
336                 fk_list.append([fk[0], movie['rowid']])
337     return fk_list, movielist
338
339
340 def split_name(string):
341     retv = string.rsplit(' ', 1)
342     if len(retv) == 2:
343         return retv
344     else:
345         return retv + retv
346
347
348 def normalize_with_namesplit(vdict):
349     nor_table = []
350     for item in vdict.items():
351         fn, ln = split_name(item[1])
352         nor_table.append([item[0], fn, ln])
353     return nor_table
354
355
356 def normalize_simple(vdict):
357     nor_table = []
358     for item in vdict.items():
359         nor_table.append(item)
360     return nor_table
361
```

```
362
363 def normalize_actor_table(vdict):
364     nor_table = []
365     for item in vdict.items():
366         fn, ln = split_name(item[1][0])
367         nor_table.append([item[0], fn, ln, item[1][2]])
368     return nor_table
369
370
371 def normalize_movie_table(vdict):
372     nor_table = []
373     for item in vdict:
374         nor_table.append([item['rowid'], item['imdbid'],
375                           item['title'], item['originaltitle'],
376                           item['year'], item['country'], item['plot'],
377                           item['votes'], item['runtime'],
378                           item['rating'], item['tagline']])
379     return nor_table
380
381
382 def get_normalized_table(table, tag):
383     if tag in ['country', 'year', 'genre']:
384         return normalize_simple(table)
385     elif tag in ['credits', 'director']:
386         return normalize_with_namesplit(table)
387     elif tag in ['actor']:
388         return normalize_actor_table(table)
389     else:
390         return normalize_movie_table(table)
391
392
393 def write_csv(dataset, path):
394     csv_writer = csv.writer(
395         open(os.path.join('csv', path + '.csv'), 'w'), delimiter=',',
396         quotechar='"', quoting=csv.QUOTE_ALL
397     )
398     head = TITLES[path]
399     dataset.insert(0, head)
400     for item in dataset:
401         csv_writer.writerow(item)
402
403
404 def write_movietotable_csv(movie_table, values):
405     for item in values:
406         write_csv(get_mapping_table(movie_table, item), 'movieto' + item)
407
408
409 if __name__ == '__main__':
410     args = docopt(__doc__, version='xbmc2apex 1.0')
411     movielist = get_dictlist_from_movie(args['PATH'])
412
413     movie_table = create_table_for(movielist, 'movie')
414     tables = ['country', 'year', 'genre', 'credits', 'director', 'actor']
415
416     dirs = {}
417     normalized = {}
418     # create dirs with simple tables
419     for table in tables:
```

```
420             dirs.setdefault(table, create_table_for(movie_table, table))
421             normalized.setdefault(table, get_normalized_table(dirs[table], table))
422             # writes country, year,  genre, credits, director and actor csv files
423             write_csv(normalized[table], table)
424
425         # create country and year fk's in movie table
426         country_to_movie, movie_table = create_nto1_table(movie_table,
427                                                           dirs['country'],
428                                                           'country')
429         year_to_movie, movie_table = create_nto1_table(movie_table,
430                                                        dirs['year'],
431                                                        'year')
432         # writes fanart and cover csv files
433         write_csv(create_1ton_table(movie_table, 'fanart'), 'fanart')
434         write_csv(create_1ton_table(movie_table, 'thumb'), 'cover')
435
436         # writes 'movie to table' csv files for director, actor, genre, credits
437         write_movietotable_csv(movie_table, ['director', 'actor',
438                                              'credits', 'genre'])
439         # writes movie table csv
440         write_csv(get_normalized_table(movie_table, 'movie'), 'movie')
```

KAPITEL 5

IMPLEMENTIERUNG

Im Folgenden Abschnitt werden die implementierten Elemente mit einer Abbildung und dem dazugehörigen Quelltext erläutert.

5.1 Computations and Shared Components

Da sich aufgrund der gewählten Anwendung keine weiteren Berechnungen ergeben, wurden Computations genutzt um beispielsweise die Übergabevariablen *MOVIE_ID*, *ACTOR_ID* zu initialisieren.

Die Übergabevariablen wurden hier unter

```
Application Builder > Application 389 > Shared Components > Application Items
```

eingepflegt und unter

```
Application Builder > Application 389 > Shared Components > Application
Computations
```

wurden diese mit Standardwerten initialisiert.

5.2 Einpflegen der Metadaten in die Datenbank

Die Metadaten stehen nach der Datenerhebung (siehe *Datenerhebung* (Seite 8)) zum import bereit. Das Script hat die 13 CSV Dateien mit Tabellen generiert. Die fehlenden Tabellen *favourites* und *movie_to_favourites* wurden über das Apex Menü händisch eingepflegt. Bei den CSV Dateien wurde für die Zwischentabellen zusätzlich ein von der Datenbank generierter primär Schlüssel verwendet.

5.3 Restful API mit Apex

Eine nachträgliche manuelle Pflege der Daten bzw das Erfassen neuer Filme ist nicht vorgesehen da dies händisch sehr aufwendig ist. Hierzu würde sich jedoch ein mit Apex erstelltes RESTful interface gut eignen. So wäre es problemlos möglich neue Filme direkt über einen der großen Metadaten Anbieter wie IMDB [1] oder TMDB [2] einzupflegen. Aus zeitlichen Gründen ist dies jedoch im Umfang der Studienarbeit nicht realisierbar.

[1] http://www.imdb.com
[2] http://www.themoviedb.org

5.4 Movies Page

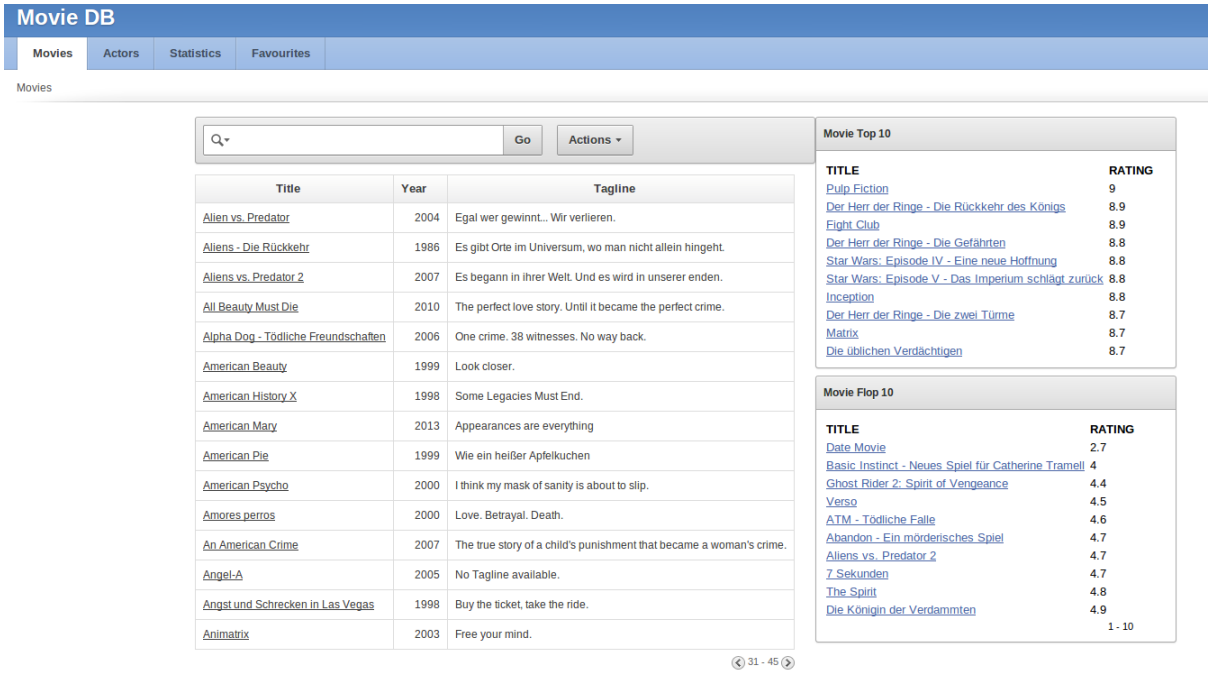


Abbildung 5.1: Übersicht Movie Page

5.4.1 Movie Search Region

Die Movies Page enthält einen Interaktiven Report mit Suchfunktion über den alle in der Datenbank vorliegenden Filme. Dargestellt werden hier die Attribute Titel, Jahr und die Tagline um eine Kurzinfo zum Film zu erhalten. Die Suchergebnisse haben eine Verlinkung auf die Movie Details Page, die beim auswählen eines bestimmten Films aufgerufen wird. Um die Movie ID an die Details Page weiterzureichen wird die shared Komponente *MOVIE_ID* verwendet.

Folgender Quellcode wurde hierfür verwendet:

```
1  select
2      movie.id, movie.title, year.year, movie.tagline
3  from movie
4      join year on year.id = movie.year_id
5  where
6      movie.year_id = year.id
7  order by 2;
```

5.4.2 Movie Top/Flop 10 Region

Für die Darstellung der im Screenshot zu sehenden Top und Flop 10 Movies wurde ein klassischer Report verwendet.

Folgender Quellcode wurde hierfür verwendet:

```
1  select id, title, rating
2  from (
```

```
3       select id, title, rating from movie
4       order by rating desc
5     )
6   where rownum <= 10;
```

Die Anzeige der Flop 10 erfolgt analog mit absteigender Reihenfolge.

5.4.3 Movies Navigation/Breadcrumb Region

Für die Movie Seite wurde zur besseren Navigation ein Movie Breadcrumb eingerichtet von dem Movie Details und Movie Fanart ableiten.

5.5 Movie Details Page

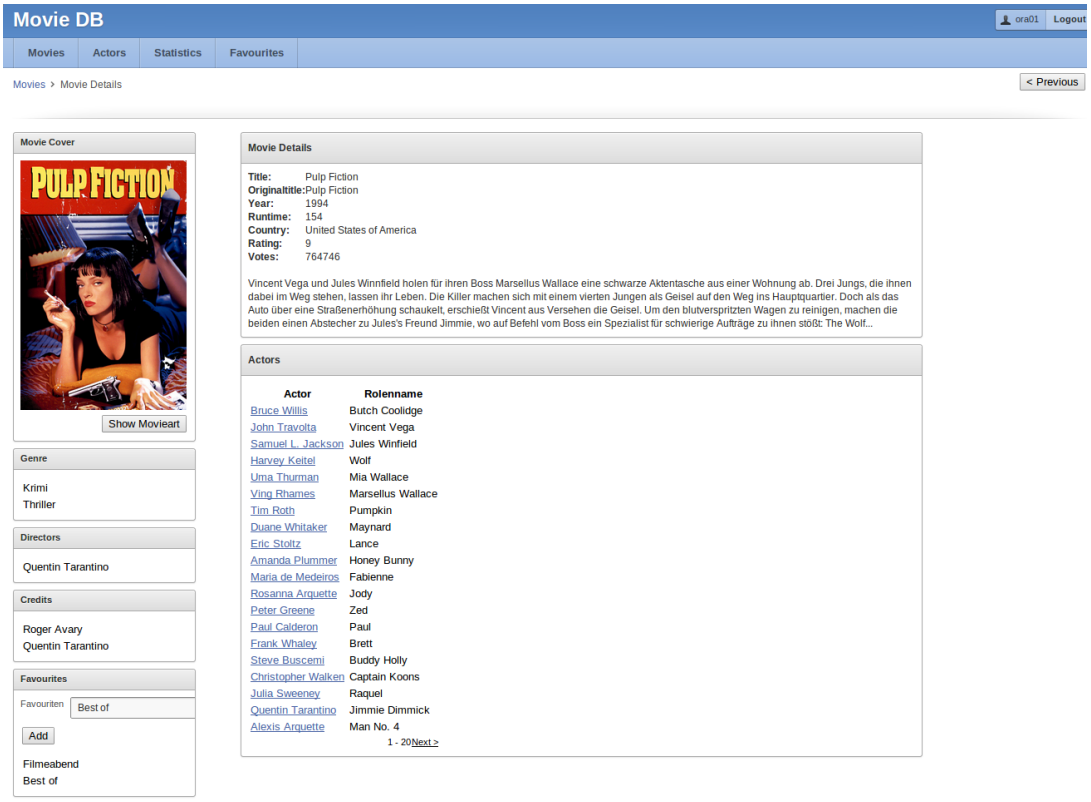


Abbildung 5.2: Übersicht Movie Details Page

5.5.1 Movie Cover Region

Für die Anzeige des Moviecovers wurde folgender PLSQL Block implementiert:

```
1  declare
2      mcover cover.cover_url%type;
3  begin
4  select
5      cover.cover_url
6  into
7      mcover
8  from movie
9      join cover on cover.movie_id = movie.id
10 where movie.id = v('MOVIE_ID')
11 and rownum <= 1;
12
13     htp.p('<img src="' || mcover || '"width="200" />');
14 end;
```

5.5.2 Movie Details Region

Dieser PLSQL Block holt sich die Grundlegenden Metadaten aus der Movie-Tabelle sowie die Filmbeschreibung.

```
declare
    mtitle movie.title%type;
    mplot movie.plot%type;
    motitle movie.originaltitle%type;
    myear year.year%type;
    mvotes movie.votes%type;
    mrating movie.rating%type;
    mruntime movie.runtime%type;
    mcountry country.name%type;
begin
select
    movie.title, movie.originaltitle, country.name , year.year,
    movie.votes, movie.rating , movie.runtime, movie.plot
into
    mtitle, motitle, mcountry, myear, mvotes, mrating, mruntime, mplot
from movie
    join country on country.id = movie.country_id
    join year on year.id = movie.year_id
where movie.id = v('MOVIE_ID')
and rownum <= 1;

htp.p('
<table border="1">
<tr>
    <td><b>Title:</b></td>
    <td>'|| mtitle ||'</td>
</tr>
<tr>
    <td><b>Originaltitle:  </b></td>
    <td>'|| motitle ||'</td>
</tr>
<tr>
    <td><b>Year:</b></td>
    <td>'|| myear ||'</td>
</tr>
<tr>
    <td><b>Runtime:</b></td>
    <td>'|| mruntime ||'</td>
</tr>
<tr>
    <td><b>Country:</b></td>
    <td>'|| mcountry ||'</td>
</tr>
<tr>
    <td><b>Rating:</b></td>
    <td>'|| mrating ||'</td>
</tr>
<tr>
    <td><b>Votes:</b></td>
    <td>'|| mvotes ||'</td>
</tr>
</table><br />'
|| mplot || '');
end;
```

5.5.3 Show Movieart Button

Beim Klicken dieses Buttons wird man auf die Fanart Page weitergeleitet. Die *MOVIE_ID* Variable wird hier an die Fanart Page weitergeleitet. Als Hilfestellung wurde hier ein entsprechender Tooltip bei den Button Attributes hinterlegt.

```
onmouseover="toolTip_enable(event,this,'Show alternaitve Movie Covers and Fanart')"
```

5.5.4 Actors Region

Diese Region zeigt die Schauspieler und den Rollennamen an. Beim Klicken auf einen Schauspieler wird man auf die Actor Page des jeweiligen Schauspielers weitergeleitet, wo man wiederum sieht in welchen anderen Filmen dieser mitspielt.

```
select
    actor.id, (actor.first_name || ' ' || actor.last_name) as Actor,
    movie_to_actor.role_name as Rolename
from movie
    join movie_to_actor on movie_to_actor.movie_id = movie.id
    join actor on actor.id = movie_to_actor.actor_id
where movie.id = v('MOVIE_ID')
```

5.5.5 Genre, Directors, Credits Region

Die Regions Genre, Directors und Credits wurden als Report implementiert und durch folgende Codesnippets realisiert:

Codesnippet für Genre:

```
select
    genre.id, genre.name as Genre
from  genre
    join movie_to_genre on movie_to_genre.genre_id = genre.id
    join movie on movie_to_genre.movie_id = movie.id
and movie.id = v('MOVIE_ID');
```

Codesnippet für Directors:

```
select
  director.id, (director.first_name || ' ' || director.last_name) as Director
from director
    join movie_to_director on director.id = movie_to_director.director_id
    join movie on  movie.id = movie_to_director.movie_id
and movie.id = v('MOVIE_ID')
```

Codesnippet für Credits:

```
select
    distinct(credits.id), concat(credits.first_name, ' ' || credits.last_name) as Writer
from credits
    join movie_to_credits on credits.id = movie_to_credits.credits_id
    join movie on movie.id = movie_to_credits.movie_id
and movie.id = v('MOVIE_ID')
```

5.5.6 Favourites Region

Der Favourites Region Report zeigt an in welchen Favoriten Listen der Film bereits ist. Ist dieser Film noch in keiner Liste, so wird *no data* angezeigt.

```
select
    distinct(movie.id), favourites.fav_name
from favourites
    join movie_to_favourites on movie_to_favourites.favourites_id = favourites.id
    join movie on movie_to_favourites.movie_id = movie.id
where movie.id = v('MOVIE_ID')
```

5.5.7 Favourites Select List

Über das Select List Item *FAVITEMS* kann der Benutzer eine Favoriten Liste auswählen in die er den aktuellen Film hinzufügen möchte. Die Liste wird mit folgendem Code befüllt:

```
select
    fav_name, id
from
    favourites order by 1;
```

5.5.8 Favourites Add Button

Um den Film in die gewählte Favoriten Liste hinzuzufügen wird der *ADD* Button verwendet. Er triggert zwei dynamische Aktionen. Die *Add to favs* und die *refresh* Aktion. Die erste Aktion startet einen PLSQL Block der das gewählte Item in die Datenbank schreibt, die zweite Aktion führt einen *refresh* auf die *Favourites* Region, so sieht der Benutzer gleich wie der Film in der Favoriten Liste erscheint. Bei diesem Button wurde ein entsprechender Tooltip hinterlegt.

```
onmouseover="toolTip_enable(event,this,'Click to add movie to selected favourites item')
```

Favourites Dynamic PLSQL Action on Add Button

Diese dynamische Aktion wird beim drücken des *Add Buttons* ausgeführt.

```
begin
    insert into movie_to_favourites(favourites_id, movie_id)
    values(:FAVITEMS, v('MOVIE_ID'));
end;
```

Favourites Dynamic Refresh Action on Button

Beim Drücken des *Add* Buttons wird ebenso eine dynamische Aktion ausgeführt welche die Region Favourites aktualisiert.

5.5.9 Movie Details Navigation/Breadcrumb Region und Previous Button

Zur besseren Navigation wurde hier ein Breadcrumb eingerichtet der von der Movie Page ableitet. Der Previous Button ist als zusätzliches Navigationselement zu sehen welcher den Benutzer zurück auf die

Movie Page bringt. Bei diesem Button wurde unter Button Attributes ein entsprechender Hilfetext als Tooltip hinterlegt.

```
onmouseover="toolTip_enable(event,this,'go back to movie page')"
```

5.6 Movieart Page

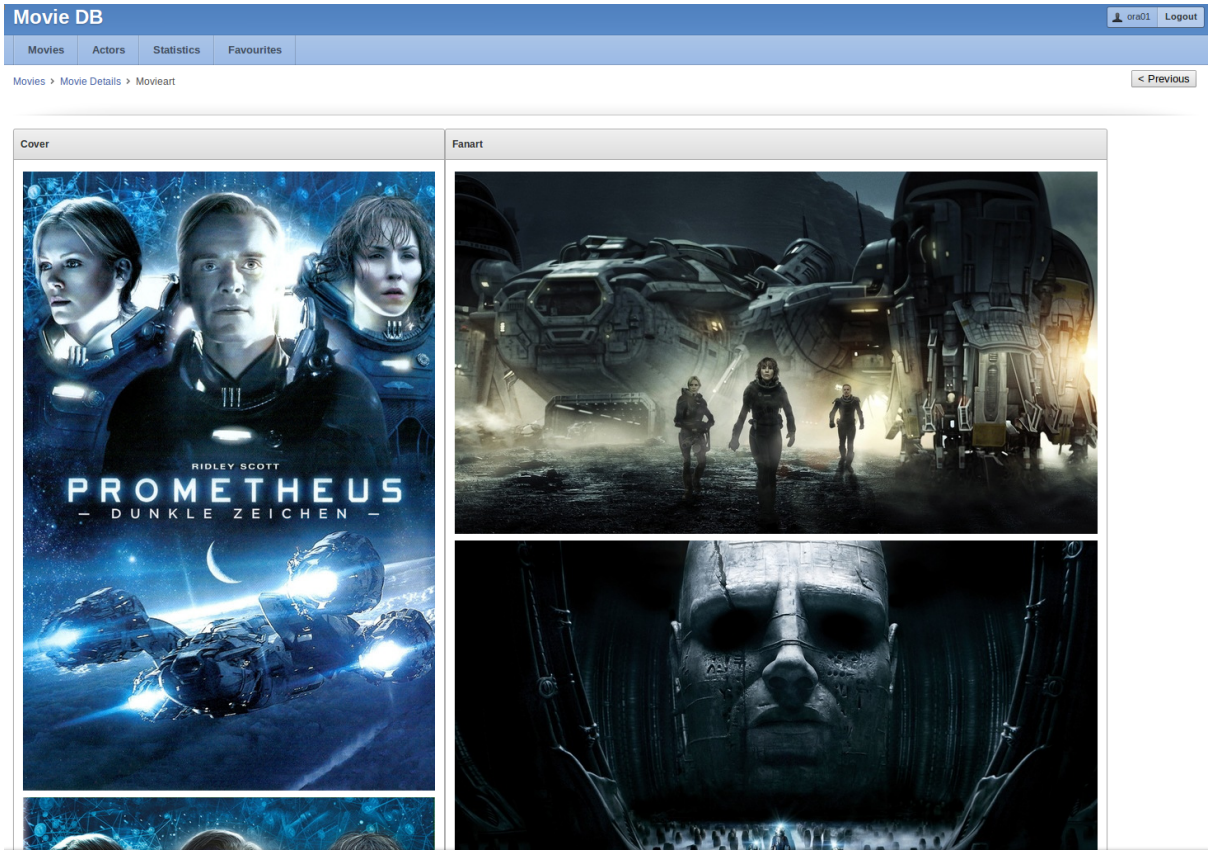


Abbildung 5.3: Übersicht Movieart Page

Diese Seite wird getriggert wenn der Benutzer den *show Movieart* Button auf der Movie Details Page klickt. Sie zeigt dem Benutzer alternative Cover zum Film soviele Fanart.

5.6.1 Cover und Fanart Region

Die Cover und Fanart Bilder werden mit Hilfe eines Reports angezeigt. Beispielhaft die SQL Query für den Cover Report, der Fanart Report funktioniert analog zum diesem jedoch mit der Fanart Tabelle.

```
1 select
2     cover_url
3 from cover
4     join movie on cover.movie_id = movie.id
5 where movie.id = v('MOVIE_ID')
```

5.6.2 Movieart Navigation/Breadcrumb und Previous Button

Auch auf dieser Seite wurde ein *PREVIOUS* Button mit dem Tooltip

```
onmouseover="toolTip_enable(event,this,'go back to movie details page')"
```

hinzugefügt. Beim klicken wird der Benutzer hier auf die Movie Details Seite geleitet. Ebenso werden Breadcrumbs verwendet.

5.7 Actors Page

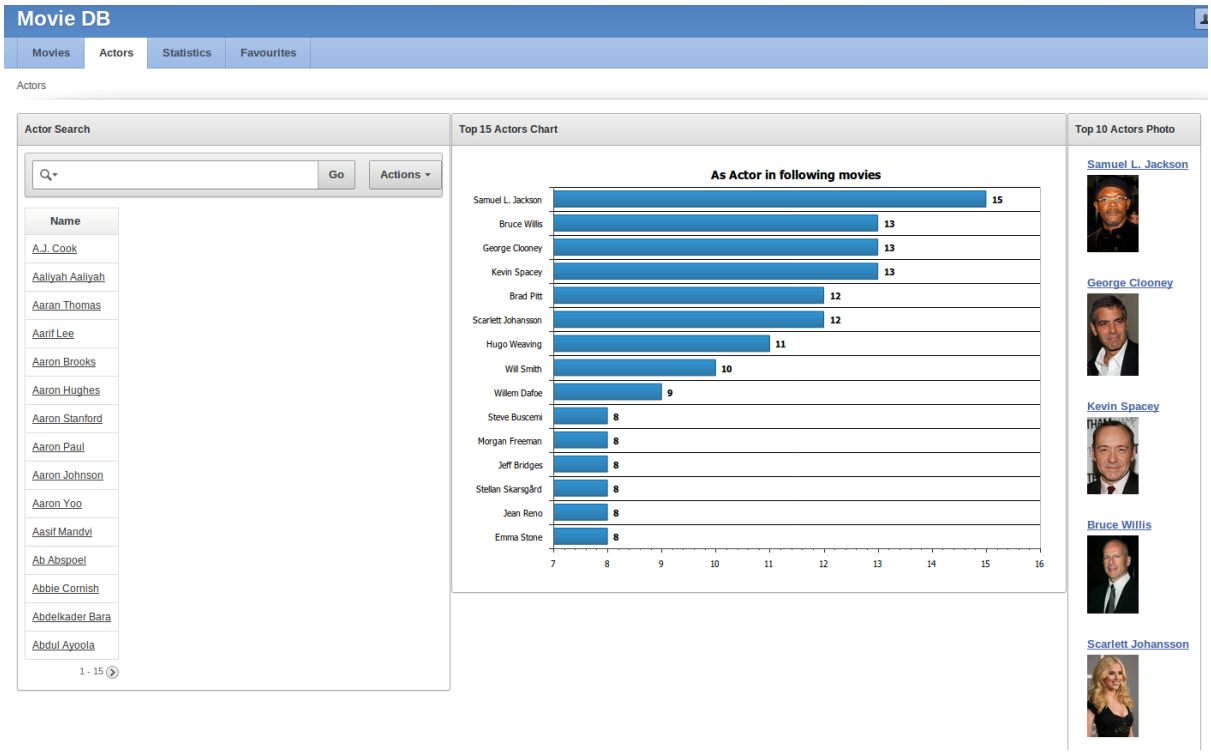


Abbildung 5.4: Übersicht Actor Page

5.7.1 Actor Search Region

Wie bei der Movie Suche ist auch eine Actor Suche möglich. Diese wurde ebenso mit einem Interaktiven Report realisiert. Durch auswählen eines Schauspielers wird man auch hier auf eine Details Page weitergeleitet, die Actor Details Page. Als Übergabewert wird hier die Acotr ID über die *ACTOR_ID* shared Komponente weitergegeben. Folgendes Codesnippet realisiert die Suchfunktion:

```
1 select
2     actor.id, (actor.first_name || ' ' || actor.last_name) as Name
3 from actor
4 order by actor.first_name;
```

5.7.2 Top 15 Actors Chart Region

Das Chart zeigt schnell auf den ersten Blick eine Auflistung der Top 15 Schauspieler in verschiedenen Filmen nach Häufigkeit.

Folgendes SQL Query wird zum generieren des Charts verwendet:

```
select * from (
    select null link,  actor.first_name || ' ' || actor.last_name,
        count(actor.id)
    from actor join movie_to_actor on actor.id = movie_to_actor.actor_id
        join movie on movie.id = movie_to_actor.movie_id
    group by actor.first_name, actor.last_name, actor.id
    order by 3 desc
)
where rownum <= 15;
```

5.7.3 Top 10 Actors Photo Region

Neben dem Chart sollen die Bilder der Top 10 Schauspieler angezeigt werden. Dies wird mittels einem Report gemacht indem die Photo Spalte mit folgender HTML Expression formatiert wird:

```
<img src="#PHOTO#" width="60px" />
```

Die SQL Query für diesen Report:

```
select * from
(
    select
        actor.id, count(actor.id),
        (actor.first_name || ' ' || actor.last_name),
        actor.photo_url
    from actor
        join movie_to_actor on actor.id = movie_to_actor.actor_id
        join movie on movie.id = movie_to_actor.movie_id
    group by actor.id, actor.first_name, actor.last_name, actor.photo_url
    order by 2 desc
) where rownum <= 10;
```

5.7.4 Actors Navigation/Breadcrumb Region

Für die Actor Page wurde eine neuer Actor Breadcrumb angelegt um den Benutzer parallel zum Film auch durch die Actor Hierarchie zu navigieren. Die Seite Actor Details leitet von diesem ab.

5.8 Actor Details Page

Beim selektieren eines bestimmten Schauspielers wird man auf dessen Detail Page weitergeleitet. Hier findet man Informationen in welchen weiteren Filmen dieser unter welchen Rollennamen mitspielt.

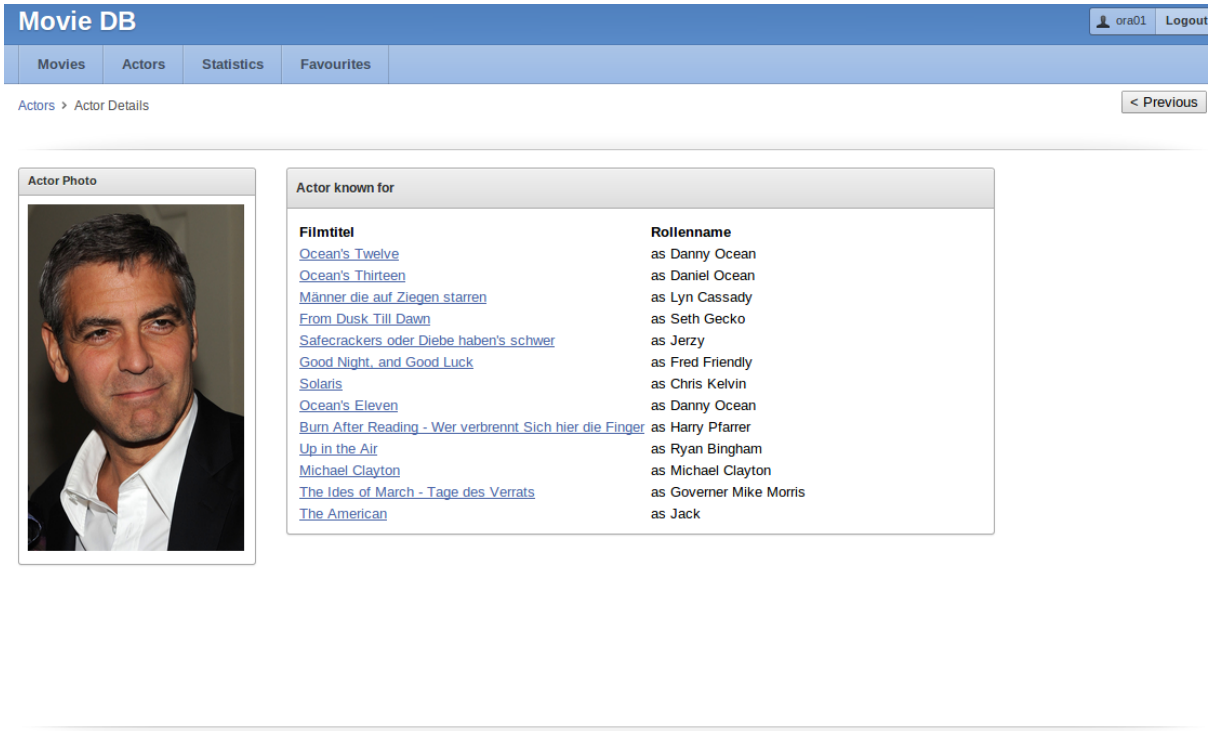


Abbildung 5.5: Übersicht Actor Details Page

5.8.1 Actor Photo Region

Um das Schauspieler Photo anzuzeigen wird wie beim Movie Cover ein PL/SQL Block ausgeführt. Im falle des Schauspielers ist das folgender Block:

```
declare
    mfname actor.first_name%type;
    mlname actor.last_name%type;
    mrole movie_to_actor.role_name%type;
    mpurl actor.photo_url%type;
begin
    select
        actor.first_name, actor.last_name, movie_to_actor.role_name, actor.photo_url
    into
        mfname, mlname, mrole, mpurl
    from movie
        join movie_to_actor on movie_to_actor.movie_id = movie.id
        join actor on movie_to_actor.actor_id = actor.id
    where actor.id = v('ACTOR_ID')
    and rownum <= 1;
    htp.p('<img src="' || mpurl|| '"width="200" />');
end;
```

5.8.2 Actor known for Region

Um anzuzeigen in welchen Filmen der gewählte Schauspieler mitspielt wird ein Report mit folgendem SQL Statement ausgeführt:

```
select
    movie.id, movie.title as Movietitle,
    concat('as ', movie_to_actor.role_name) as Rollenname
from movie
    join movie_to_actor on movie_to_actor.movie_id = movie.id
    join actor on movie_to_actor.actor_id = actor.id
where actor.id = v('ACTOR_ID')
```

5.8.3 Actor Details Navigation/Breadcrumbs und Previous Button

Für die Navigation werden wie auch zuvor bei der Movie Details Tabelle Breadcrumbs sowie ein Previous Button verwendet. Der Button bringt einen auf die Actor Page und enthält folgendes Hilfetext Attribut:

```
onmouseover="toolTip_enable(event,this,'go  back to previous page')"
```

5.9 Statistics Page

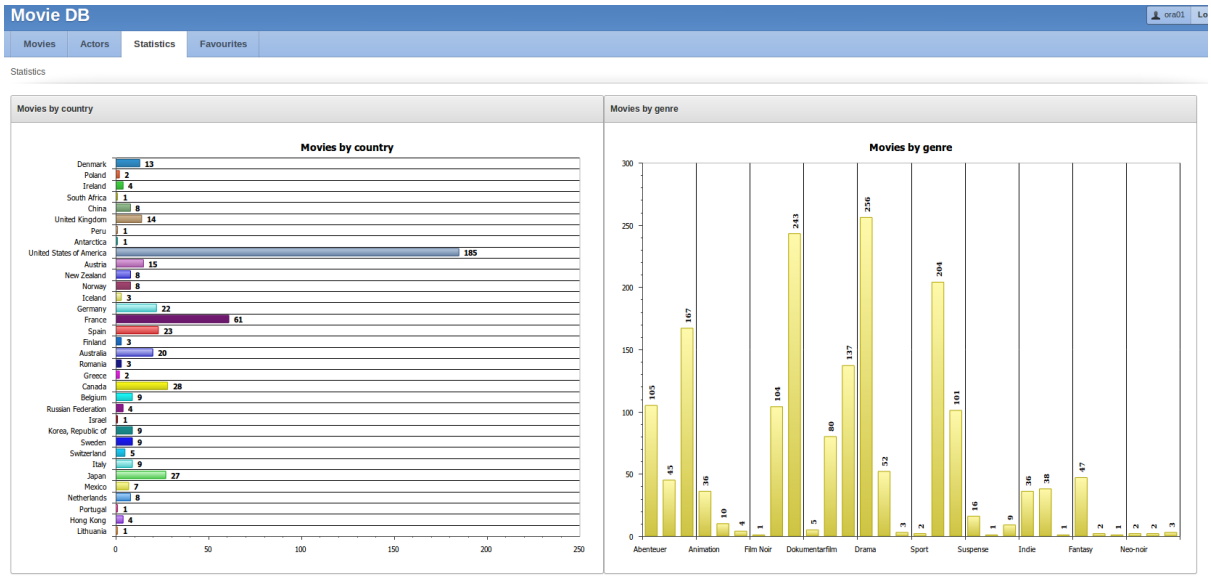


Abbildung 5.6: Übersicht Statistics Page

Die Statistics Page enthält Diagramme welche die Verteilung der Filme nach Herstellungland und nach Genre zeigen.

5.9.1 Movie by country Chart Region

Folgende SQL Query wird verwendet um die Verteilung nach Herstellungsland zu visualisieren:

```
select
    null link, country.name, count(country.name)
from movie
    join country on movie.country_id = country.id
group by country.name;
```

5.9.2 Movie by genre Chart Region

Die Visualisierung über die verschiedenen Genre wurde mit folgender SQL Query realisiert:

```
select
    null link, genre.name, count(genre.name)
from genre
    join movie_to_genre on genre.id = movie_to_genre.genre_id
    join movie on movie.id = movie_to_genre.movie_id
group by genre.name;
```

5.9.3 Statistics Navigation/Breadcrumb Region

Wegen dem einheitlichem *look and feel* wurde auf der Statistics Page ein eigener Breadcrumb erstellt.

5.10 Favourites Page

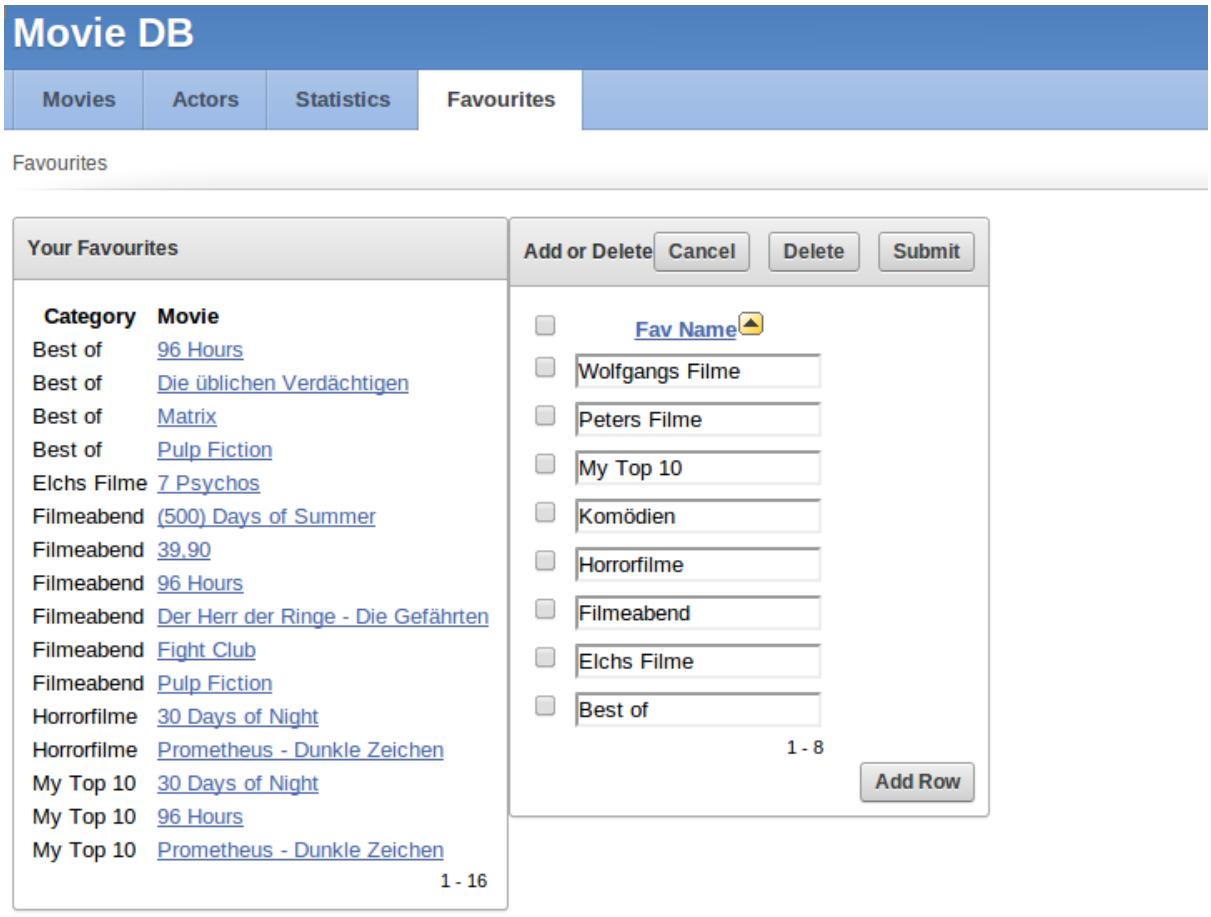


Abbildung 5.7: Übersicht Favourites Page

Über diese Seite kann der Benutzer Favoriten Listen Pflegen und bekommt einen Überblick darüber welcher Film in welcher Favoriten Liste bereits eingetragen ist.

5.10.1 Your Favourites Region

Dieser Report macht eine Abfrage auf die Favoriten Liste und visualisiert diese.

```
select
    favourites.fav_name as Category,
    movie.title as Movie, movie.id
from movie
    join movie_to_favourites on movie_to_favourites.movie_id = movie.id
    join favourites on movie_to_favourites.favourites_id = favourites.id
group by favourites.fav_name, movie.title, movie.id
order by favourites.fav_name
```

5.10.2 Add or Delete Form Region

Dieses Formular wird verwendet um Favoriten Listen zu erstellen oder zu löschen. Es enthält folgenden autogenerierten Code:

```
select
    "ID",
    "FAV_NAME"
from "#OWNER#"."FAVOURITES"
```

Die Buttons, Cancel, Delete, Submit und Add vom Formular wurden mit entsprechenden Tooltips hinterlegt (Reihenfolge entsprechend der Aufzählung):

```
onmouseover="toolTip_enable(event,this,'throw away not submitted stuff')"
onmouseover="toolTip_enable(event,this,'Delete selected favlists.')"
onmouseover="toolTip_enable(event,this,'Submits newly added favlist.')"
onmouseover="toolTip_enable(event,this,'Add a new Favlist, dont forget to submit it!')"
```

5.10.3 Form Validation FAV_NAME

Um zu vermeiden dass eine Favoriten Liste doppelt hinzugefügt wird, wurde eine Validierung implementiert. Diese prüft beim *Submit* mit folgendem Code

```
:FAV_NAME not in (select fav_name from favourites)
```

ab ob die Favoriten Liste bereits in der Tabelle existiert. Existiert diese bereits in der Tabelle, so kommt die Fehlermeldung:

```
Error, favourites list already exists.
```

Das Textfeld wird hierbei rot und die Fehlermeldung erscheint. Bei Erfolg verlinkt die Seite auf sich selbst.

5.10.4 Favourites Navigation/Breadcrumb

Wegen der Einheitlichkeit hat die Favourites Seite auch einen eigenen Breadcrumb namens Favourites enthalten.

5.10.5 Datenbank Trigger

Um beim Löschen einer Favoriten Liste auch alle Beziehungen in der movie_to_favourites Tabelle zu löschen wurde ein Datenbank Trigger auf der Favourites Tabelle erstellt, dieser schaut wie folgt aus:

```
1 CREATE OR REPLACE TRIGGER clean_movie_to_favourites
2 BEFORE DELETE ON favourites
3 FOR EACH ROW
4  BEGIN
5        delete from movie_to_favourites
6        WHERE movie_to_favourites.favourites_id = :old.id;
7 END clean_movie_to_favourites;
```

5.10.6 Validierung

Alle hier abgebildeten SQL Codesnippets wurden mit dem Oracle SQL Developer getestet indem das Ergebnis mit den erwarteten Werten verglichen wurde. Die Funktionalität der Validierung (*Form Validation FAV_NAME* (Seite 31)) des Add und Delete Formulars (*Add or Delete Form Region* (Seite 31)) wurde getestet indem versucht wurde eine bereits existierende Favoriten Liste nochmals hinzuzufügen. Hierbei ist die erwartete Fehlermeldung aufgetreten.