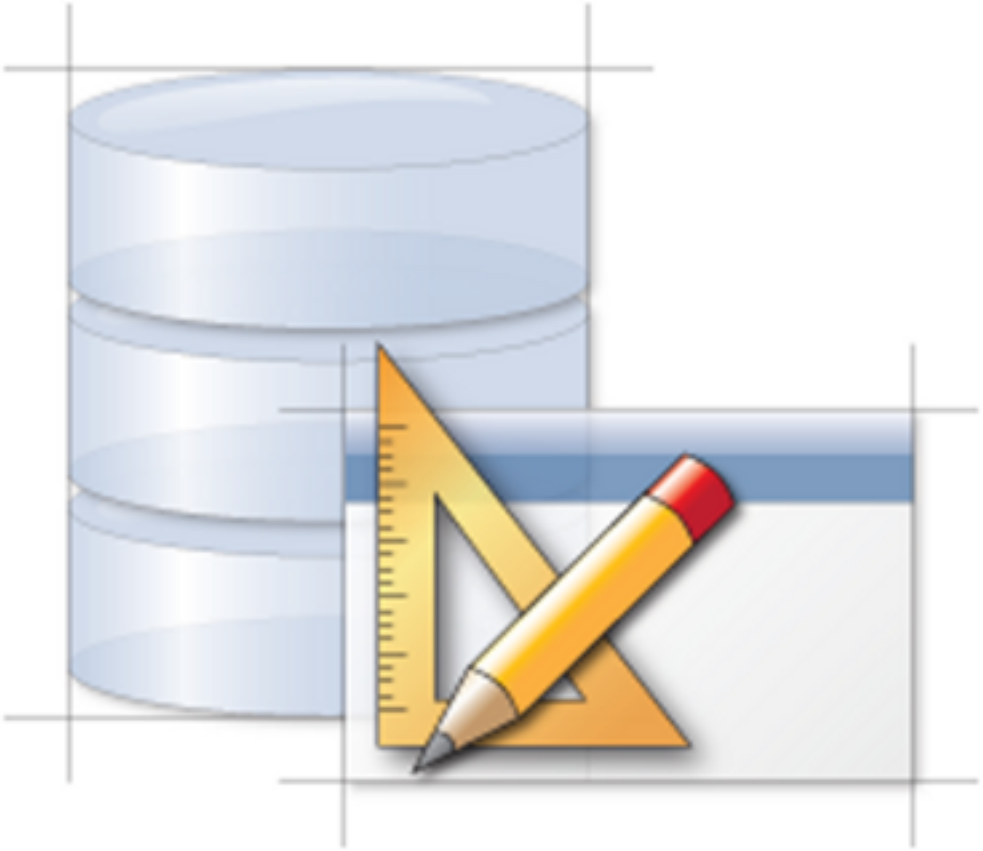

MusicLibraryBrowser mit Apex Documentation

Christopher Pahl

June 27, 2013

Contents

Zugangsdaten

Workspace	APEX_10224
Anwendungsnummer	306
Username	ORA01
Passwort	elchwald

Sinn und Zweck der Anwendung

Die Anwendung *MusicLibraryBrowser* stellt eine einfache Möglichkeit dar seine Musiksammlung zu visualisieren und im Browser anzuzeigen. Da die Daten hier serverseitig lagern kann die Musiksammlung auch Freunden zugänglich gemacht werden. Es kann und soll aber keine Möglichkeit geben die Musik abzuspielen (schon allein aus Lizenstechnischen Gründen).

Momentan umfasst die Anwendung dabei mehrere Seiten welche die Navigation in der Musikdatenbank erlaubt:

- **Songdatabasepage:** Zeigt eine Liste mit allen Songs, optional durch eine Playlist eingeschränkt.
- **Artist Detail View:** Zeigt Details zu einem bestimmten Artist in der Sammlung.
- **Album Detail View:** Zeigt Details zu einem bestimmten Album in der Sammlung.
- **Title Detail View:** Zeigt Details zu einem bestimmten Track in der Sammlung.

Die Funktion und Realisierung dieser Seiten wird weiter unten noch im Detail erklärt. Im Vergleich zur Spezifikation sind die angestrebten Elemente in den Mockups auch größtenteils implementiert worden. Lediglich die Positionierung der Regions ist komplett anders.

Importieren der Musikdatenbank

Momentan bietet die Anwendung keine Möglichkeit die Musik von der Oberfläche aus zu importieren.

Die Testdaten wurde durch ein Python Skript generiert dass die Musikdatenbank des MPD (MusicPlayer-Daemon - ein unter Unix sehr beliebter Musicplayer) abfragt. Die abgefragten Daten versieht es noch mittels libglyr (eine Library die ich passenderweise im 2ten Semester geschrieben hatte) mit Metadaten wie Lyrics, Coverart, Artistphotos und Artistbiographien.

Der Vollständigkeit halber ist das Skript hier mit angegeben:

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

'''
Convert MPD's Queue content to a set oc CSV files
that can be imported into Apex easily.

Author: Christopher Pahl
'''

import telnetlib
import csv
import plyr

# Create
db = plyr.Database('/tmp')


# Get the content of the Queue from MPD
# Use telnet to connect to MPD and talk to it via it's protocol
# http://www.musicpd.org/doc/protocol/
def create_songs_data():
    # Open the session and skip the version info
    tel = telnetlib.Telnet('localhost', 6600)
    tel.read_until(b'\n')

    # Send a command to list the Queue and read the whole repsonse blob
    tel.write(b'playlistinfo\n')
    response = tel.read_until(b'\nOK').decode()

    # Map the Protocol names to internal names
    attr_map = {
        'Artist': 'artist',
        'Album': 'album',
        'Title': 'title',
        'Time': 'duration',
        'Genre': 'genre'
    }

    # Split the blob in lines and read the information into
    # a large list of dictionaries. Also strip the values.
    songs = []
    for song in response.split('file: '):
        song_dict = {}

        file_idx = song.find('\n')
        if file_idx is not -1:
            song_dict['File'] = song[:file_idx].strip()

        for line in song.splitlines():
            name_value = line.split(':', maxsplit=1)
            if len(name_value) != 2:
                continue
```

```
54
55              name, value = name_value
56              if name in attr_map.keys():
57                  song_dict[attr_map[name]] = value.strip()
58
59          if song_dict:
60              songs.append(song_dict)
61      return songs
62
63
64  # Use libglyr to quert arbitary metadata.
65  # See here: http://sahib.github.io/python-glyr/index.html
66  def query_metadata(get_type, artist, album='', title='', download=False):
67      qry = plyr.Query(
68          artist=artist,
69          album=album,
70          title=title,
71          get_type=get_type,
72          do_download=download,
73          database=db,
74          parallel=3,
75          verbosity=2
76      )
77
78      # This will block for some time.
79      results = qry.commit()
80
81      # Return an empty String if nothing was found.
82      if len(results) >= 1:
83          return results[0].data.decode()
84      else:
85          db.insert(qry, db.make_dummy())
86          return ''
87
88
89  # Extract a list of keys from a list of dictionaries.
90  def extract_keys(songs, dest_key):
91      keys = set()
92      for song in songs:
93          for key, value in song.items():
94              if key == dest_key:
95                  keys.add(value)
96      return list(keys)
97
98
99  # Get all data and split it into keys
100 # These are used later to build up the individual tables.
101 songs = create_songs_data()
102 artists = extract_keys(songs, 'artist')
103 albums = extract_keys(songs, 'album')
104 titles = extract_keys(songs, 'title')
105 genres = extract_keys(songs, 'genre')
106
```

```
107
108 # Create the Songs table by indexing artists/albums/titles
109 songs_table = []
110 for rowid, song in enumerate(songs):
111     index = lambda data, key: data.index(song[key])
112     artist_id = index(artists, 'artist') + 1
113     album_id = index(albums, 'album') + 1
114     title_id = index(titles, 'title') + 1
115     genre_id = index(genres, 'genre') + 1
116     songs_table.append((
117         rowid + 1, artist_id, album_id,
118         title_id, genre_id, int(song['duration'])
119     ))
120
121
122 # Build the Artists Table
123 artist_table = []
124 for rowid, artist in enumerate(artists):
125     artist_table.append((
126         rowid + 1, artist, query_metadata('artistphoto', artist)
127     ))
128
129
130 # Build the Albums Table
131 album_table = []
132 for rowid, album in enumerate(albums):
133     # Find corresponding artist.
134     # This would be actually easier in SQL.
135     for song in songs:
136         if album == song['album']:
137             artist = song['artist']
138             break
139     else:
140         print('No artist found for album: ', album)
141         continue
142
143     album_table.append((
144         rowid + 1, album, query_metadata('cover', artist, album)
145     ))
146
147
148 # Build the Titles Table
149 title_table = []
150 for rowid, title in enumerate(titles):
151     # Find corresponding artist.
152     # This would be actually easier in SQL.
153     for song in songs:
154         if title == song['title']:
155             artist = song['artist']
156             break
157     else:
158         print('No artist found for title: ', title)
159         continue
```

```
160
161     title_table.append((
162         rowid + 1, title, query_metadata('lyrics', artist, title=title)
163     ))
164
165 # Build the genre Table by a list comprehensions (needs only a rowid)
166 genre_table = [(rowid + 1, genre) for rowid, genre in enumerate(genres)]
167
168 playlists_table = []
169 playlist_to_songs = []
170
171
172 # Create a Playlist that with a certain name that
173 # has "query" in the artist attribute.
174 def create_playlist(name, query):
175     playlists_table.append((
176         len(playlists_table) + 1,
177         name
178     ))
179
180     playlist_id = playlists_table[-1][0]
181
182     for song in songs_table:
183         rowid, artist_id, _, _, _, _ = song
184         artist = artist_table[artist_id - 1][1]
185
186         if query in artist:
187             playlist_to_songs.append((
188                 playlist_id, rowid
189             ))
190
191
192 # Create three dummy playlists.
193 create_playlist('Metal', 'In Flames')
194 create_playlist('Mittelalter', 'In Extremo')
195 create_playlist('FarinUrlaub', 'Farin')
196
197 similar_artists_table = []
198
199
200 # Helper function to fill the Similar Artists Table.
201 def add_relation(artist, similar):
202     artist_id = None
203     similar_id = None
204
205     for song in songs_table:
206         rowid, a_id, _, _, _, _ = song
207         song_artist = artist_table[a_id - 1][1]
208
209         if artist == song_artist:
210             artist_id = a_id
211         if similar == song_artist:
212             similar_id = a_id
```

```
213
214     if similar_id and artist_id:
215         similar_artists_table.append((
216             artist_id, similar_id
217         ))
218
219
220 # Add some static relations.
221 # Till now everything would have workd with any
222 # set of songs in the Queue.
223 add_relation('Randalica', 'In Extremo')
224 add_relation('Ougenweide', 'In Extremo')
225 add_relation('Götz Alsmann', 'In Extremo')
226 add_relation('Blind', 'In Extremo')
227 add_relation('Silbermond', 'In Extremo')
228 add_relation('Grave Digger', 'In Extremo')
229 add_relation('Farin Urlaub', 'Farin Urlaub Racing Team')
230 add_relation('In Flames', 'Grave Digger')
231
232
233 # Helper function to wirte the csv to disk.
234 def write_csv(name, data):
235     with open(name + '.csv', 'w') as f:
236         spamwriter = csv.writer(f, delimiter=',', quotechar='"')
237         for row in data:
238             spamwriter.writerow(row)
239
240
241 # Now wirte them to disk so we can import them.
242 write_csv('songs', songs_table)
243 write_csv('artists', artist_table)
244 write_csv('albums', album_table)
245 write_csv('titles', title_table)
246 write_csv('genres', genre_table)
247 write_csv('playlists', playlists_table)
248 write_csv('playlists_to_songs', playlist_to_songs)
249 write_csv('similar_artists', similar_artists_table)
```

Normalisiertes Datenbankmodell

Das normaliserte Datenbankmodell kann in *DatenbankNormalisiert* betrachtet werden.

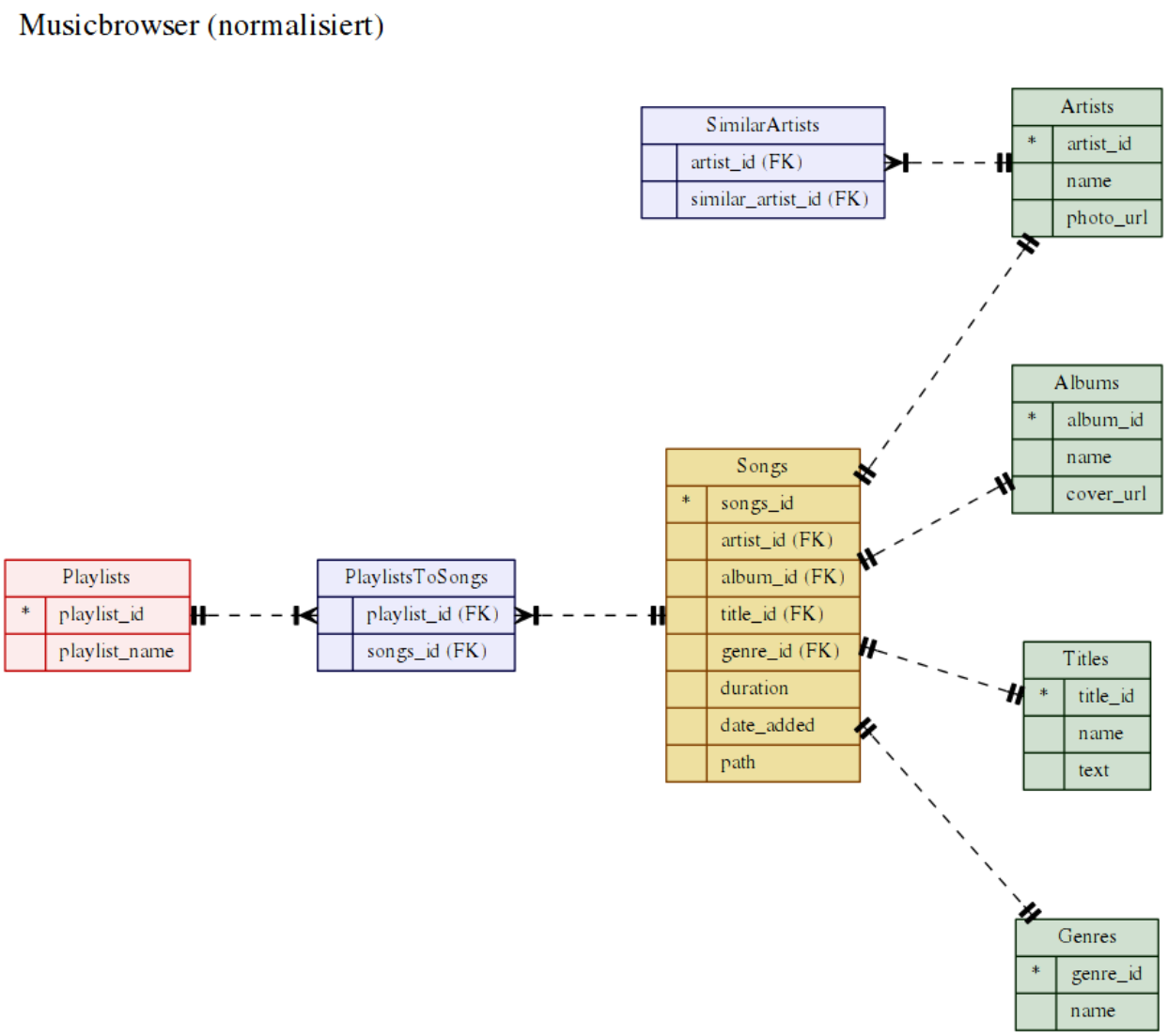

Figure 1: Normalisiertes Datenbankmodell. (mit “erwiz” gerendert)

Änderung im Vergleich zur Spezifikation

Im Vergleich zur Spezifikation ist die Datenbank größtenteils gleich geblieben. Die Songstabelle wurde lediglich um die Spalten `date_added` (ein String der das Import Datum beschreibt) und `path` (der Pfad zur Datei, relativ zum Musikverzeichniss - immer einzigartig).

Zustand vor Normalisierung

Vor der Normalisierung waren die Tabellen `artists`, `albums`, `titles`, `genres` noch direkt in der `songs` Tabelle enthalten. Die `similar_artists` Tabelle hat dementsprechend auf `songs.song_id` verlinkt und nicht auf `artists.artist_id` wie im normalisiertem Zustand.

Vor Normaliserung waren folgende Tabellen vorhanden:

- `songs`
- `playlists`
- `playlist_to_songs`
- `similar_artists`

Zur Normaliserung wurde nun die `songs` Tabelle massiv entschlackt indem statt den Spalten `artist`, `album`, `title`, `genre` nur Foreign Keys auf entsprechende neue Tabellen gespeichert wurden. Die `path` Spalte in `songs` ist bereits unique, da jedem Song nur ein Musikfile zugeordnet ist. Eine neue Tabelle wurde daher als nicht sinnvoll erachtet.

Beispieldaten

Die durch das oben gezeigte Skript generierten Daten umfassen 468 verschiedene Songs. Insgesamt wurde 8 verschiedene CSV Dateien erzeugt die händisch in Apex importiert wurden (mittels Data Workshop -> Spreadsheet Data). Die CSV Dateien sind zudem in der Abgabe-CD beigelegt.

Jede Seite besitzt eine Breadcrumbs Region die der vereinfachten Navigation dient. Siehe dazu auch die jeweils beiliegenden Screenshots.

Dies ist die Hauptseite die der Nutzer nach dem Einloggen sieht. In dieser werden entweder alle Songs in der Datenbank, oder nur die Songs einer bestimmten Playlist angezeigt. Der Songreport ist dabei interaktiv und kann daher auch durchsucht werden.

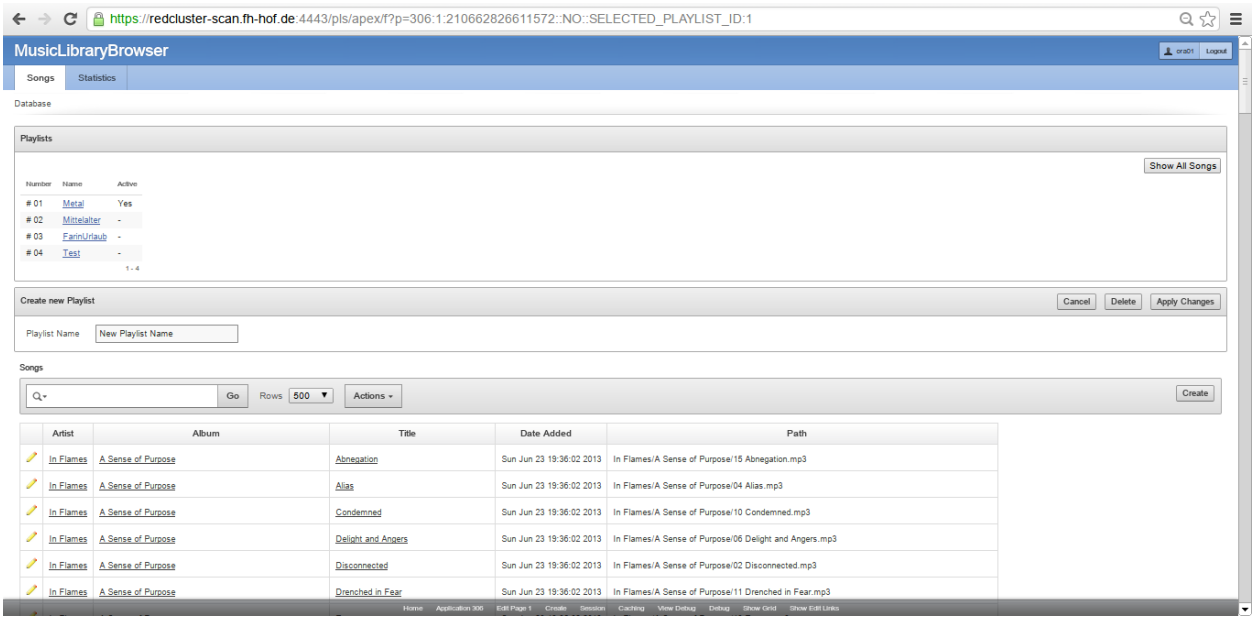

Figure 2: Ein Screenshot der Hauptseite.

Bedienelemente

Playlist Region:

In der Playlists Region können die vorhandenen Playlisten angeschaut werden. Die gerade selektierte Playlist ist dabei mit “Active” gekennzeichnet. Klickt man auf eine der Playlisten, so wird der Song Report weiter unten entsprechend gefiltert. Rechts oben in der Playlist ist zudem ein Button zu finden mit dem keine Filterung durch Playlists vorgenommen wird, sondern es werden alle Songs in der Datenbank angezeigt.

Der Playlist Report wird mit folgendem SQL Statement befüllt:

```
select
  '#' || to_char(rownum, '00'),
  playlist_id,
  playlist_name,
```

```
  (case
    when v('SELECTED_PLAYLIST_ID') = playlist_id then 'Yes'
    else ''
  end) as active
from playlists;
```

Im *Application Item* `SELECTED_PLAYLIST_ID` wird dabei die momentan selektierte Playlist ID gespeichert. Beim Drücken des "Show all" Buttons wird diese auf 0 gesetzt.

Create new Playlist Regions:

Hier können durch einen sehr einfach gehaltenen Form neue (leere) Playlisten erstellt werden, oder vorhandene gelöscht werden. Das Befüllen der Playlist ist im Kapitel zur TitleView erklärt.

Durch den Report wird die Tabelle `playlists` direkt modifiziert.

Songs Region:

Hier werden durch einen interaktiven Report alle Songs der aktuell selektierten Playlist angezeigt. Beim Klick auf den Artist, Album oder Titel wird man enstprechend auf eine Detailansicht des jeweiligen verzweigt.

Aus technischer Sicht setzt der Klick in eine Zelle eins von drei Application Items mit einer eindeutigen ID:

- `SELECTED_ARTIST_ID`
- `SELECTED_ALBUM_ID`
- `SELECTED_TITLE_ID`

Eine Suche ist dabei genau wie die Möglichkeit zu Reports bereits eingebaut.

Die Abfrage erfolgt dabei durch folgendes SQL Statement:

```
select '#' || to_char(rownum, '000'), songs.artist_id, songs.album_id,
       songs.title_id, artist, album, title, date_added, path
from songs
  join artists on songs.artist_id = artists.artist_id
  join albums on songs.album_id = albums.album_id
  join titles on songs.title_id = titles.title_id
  join playlist_to_songs on playlist_to_songs.song_id = songs.song_id
where ((v('SELECTED_PLAYLIST_ID') = 0
           AND playlist_to_songs.playlist_id >= 0)
         OR
           playlist_to_songs.playlist_id = v('SELECTED_PLAYLIST_ID'))
order by artist, album, title;
```

Auf dieser Seite findet man detallierte Informtionen zum aktuell selektierten Künstler. Der aktuell selektierte Künstler ist dabei derjenige Artist dessen `artists.artist_id` momentan in `SELECTED_ARTIST_ID` steht.

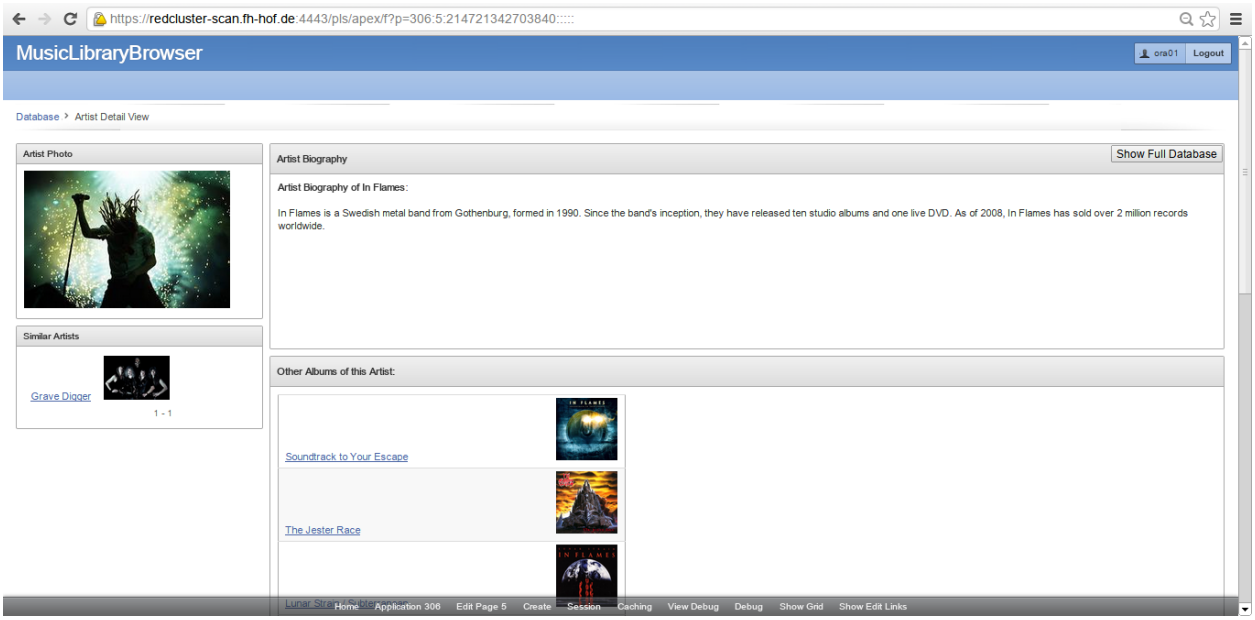

Figure 3: Ein Screenshot der Artist Seite.

Bedienelemente

Artist Photo:

In dieser Region wird einfach ein Bild des Künstlers angezeigt. Die Anzeige wurde mit einer PL/SQL Region realisiert.

```
declare
  photo_url artists.photo_url%type;
begin
  select distinct artists.photo_url into photo_url from songs
    join artists on songs.artist_id = artists.artist_id
    where artists.artist_id = v('SELECTED_ARTIST_ID');
  htp.p('<img width=250 src="' || photo_url || '"/>');
end;
```

Artist Biography:

Analog dazu funktioniert auch die Anzeige der Artist Biographie, welcher mitfile des folgenden Pl/SQL Blockes eine kurze Beschreibung des Künstlers aus der Datenbank abgfragt:

```
declare
  bio artists.artist_bio%type;
  artist artists.artist%type;
begin
  select distinct artists.artist_bio, artists.artist into bio, artist from songs
    join artists on songs.artist_id = artists.artist_id
    where artists.artist_id = v('SELECTED_ARTIST_ID');
  htp.p('<b>Artist Biography of ' || artist ||  '</b>:' || '<br /><br />' || bio);
end;
```

Other Albums of this Artist:

In diesem Report werden alle Alben dieses Künstlers dargestellt. Das Bild in der letzten Spalte wird dabei durch Setzen vom folgenden HTML Text in der Column Formatting der `cover` Column gesetzt:

```
<img width="75px" src="#COVER#" />
```

Ein Klick auf ein Album bringt einen wiederum aud die Album Detail Page.

Der Report wird doch folgendes SQL Statement befüllt:

```
select distinct albums.album_id, albums.album, (albums.cover_url) as cover
from albums
  join songs on albums.album_id = songs.album_id
  join artists on artists.artist_id = songs.artist_id
where artists.artist_id = v('SELECTED_ARTIST_ID')
```

Similar Artists:

In diesem Report werden Empfehlungen für ähnliche Künstler abgegeben. Hierbei wird der Name des ähnlichen Künstlers und das Artistphoto von diesem als Thumbnail angezeigt. Ein Klick auf den Künstlernamen verzweigt zur Artist Detail Page dieses Künstlers.

Technisch wird dies durch folgende (vermutlich leicht überkomplizierte) SQL Query realisiert:

```
select artists.artist_id, artists.artist, (artists.photo_url) as photo_url
from artists join
(
  select similar_artists.artist_id from similar_artists
    where
      similar_artists.artist_id != similar_artists.similar_artist_id AND (
        similar_artists.similar_artist_id = v('SELECTED_ARTIST_ID')
      )
  UNION
  select similar_artists.similar_artist_id from similar_artists
    where
      similar_artists.artist_id != similar_artists.similar_artist_id AND (
        similar_artists.artist_id = v('SELECTED_ARTIST_ID')
      )
```

```
) s
on s.artist_id = artists.artist_id;
```

“Show Full Database”-Button

Ein Klick auf diesem leitet wieder auf die Songdatabase Seite um.

Zeigt Informationem zum gerade selektierten Album an. Das selektierte Album ist das Album dessen `albums.album_id` momentan in `SELECTED_ALBUM_ID` steht.

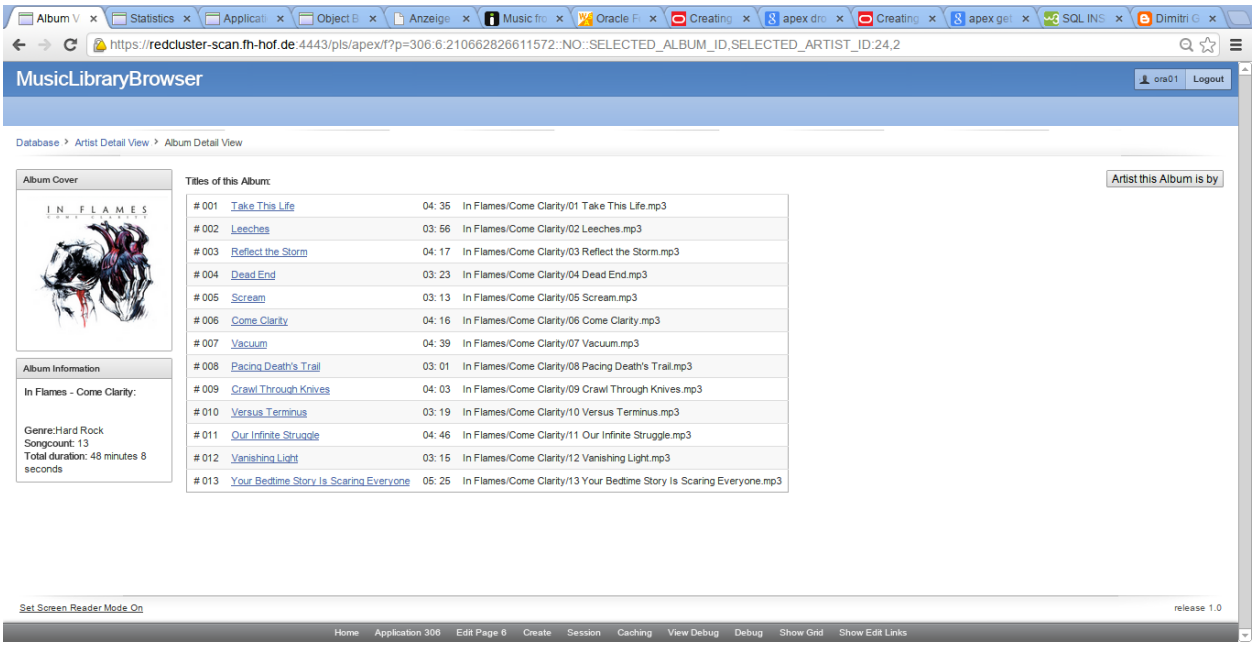

Figure 4: Ein Screenshot der Album Detail Page.

Bedienelemente

Album Cover:

Diese Region zeigt ein Cover Bild des gerade selektierten Albums an. Wie bereits das Artist Photo ist dieses mittels einer PL/SQL Region realisiert worden:

```
declare
  cover_url albums.cover_url%type;
begin
  select distinct albums.cover_url into cover_url from songs
    join albums on songs.album_id = albums.album_id
    where albums.album_id = v('SELECTED_ALBUM_ID');
  htp.p('<img width=175 height=175 src="' || cover_url || '"/>');
end;
```

Album Information:

In den Album Information Region werden allgemeine Information zu dem Album angezeigt. Dazu gehören:

- Artist des Albums
- Album Name
- Genres des Albums
- Anzahl der Songs auf diesem Album
- Gesamte Spiellänge im MM:SS Format

Dies wurde durch folgendem PL/SQL Block realisiert:

```
declare
  album albums.album%type;
  artist artists.artist%type;
  genre genres.genre_name%type;
  song_count NUMBER;
  sum_duration NUMBER;
begin
  select distinct artists.artist, albums.album, genres.genre_name
    into artist, album, genre
  from albums
    join songs on albums.album_id = songs.album_id
    join artists on artists.artist_id = songs.artist_id
    join genres on genres.genre_id = songs.genre_id
  where
    albums.album_id = v('SELECTED_ALBUM_ID') AND
    artists.artist_id = v('SELECTED_ARTIST_ID');

  select count(album_id), sum(songs.duration) into song_count, sum_duration
  from songs
  where songs.album_id = v('SELECTED_ALBUM_ID');

  htp.p('<b>' || artist || ' - ' || album || ':</b>');
  htp.p('<br /><br />');
  htp.p('<br />');
  htp.p('<b>Genre:</b>' || genre);
  htp.p('<br />');
  htp.p('<b>Songcount:</b> ' || song_count);
  htp.p('<br />');
  htp.p('<b>Total duration:</b> ' || trunc(sum_duration / 60)  ||
        ' minutes ' || mod(sum_duration, 60) ||' seconds');
end;
```

Titles of this Album:

Dieser Report zeigt eine Tracklist des Albums an. Dazu gehören:

- Songnummer
- Tracktitel

- Duration im MM:SS Format
- Pfad zur Datei (relativ zum Musikverzeichniss)

```
select distinct ('#' || to_char(rownum, '000')) as num,
       titles.title_id, titles.title,
       to_char(songs.duration / 60, '00') || ':' ||
       to_char(mod(songs.duration, 60), '00') as duration,
       songs.path
from songs
  join titles on songs.title_id = titles.title_id
  where songs.album_id = v('SELECTED_ALBUM_ID')
order by num;
```

"Artist this Album is by"-Button:

Leitet auf die Artist Detail View des Künstlers um von dem dieses Album ist.

Im Title Detail View wird der Liedtext zu einem bestimmten Titel angezeigt, sowie einige Randinformationen zu diesem.

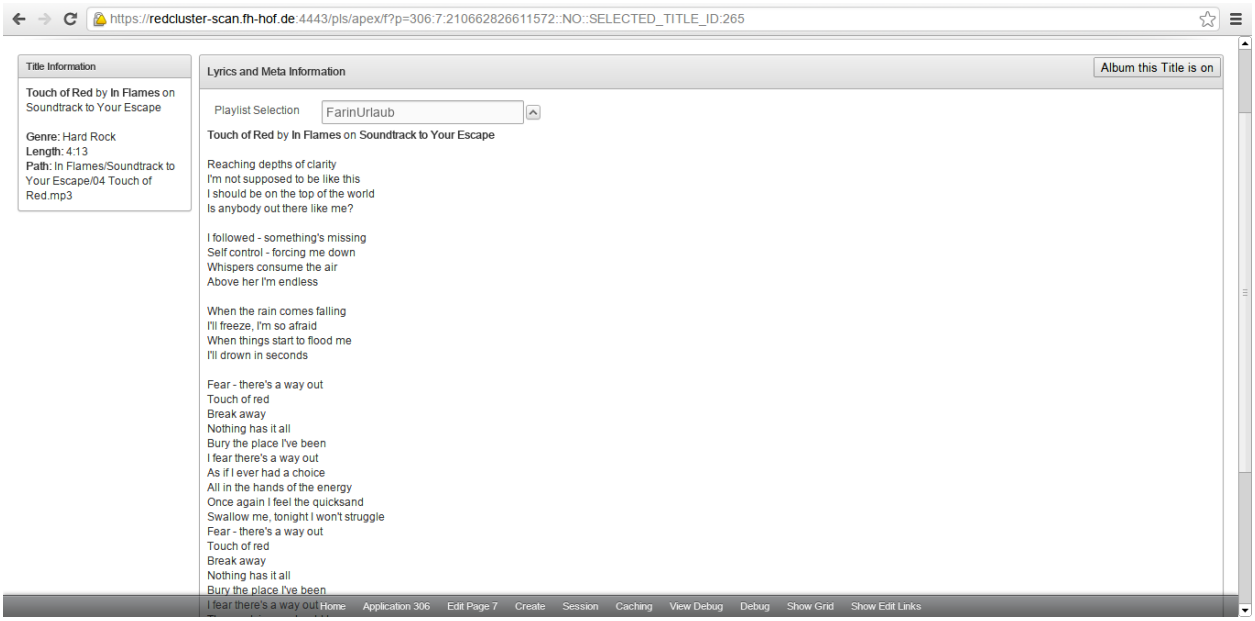

Figure 5: Ein Screenshot der Title View.

Bedienelemente

Title Information:

In dieser PL/SQL Region werden allgemeine Information zu diesem Titel angezeigt. Dazu gehört:

- Artist
- Album
- Titel
- Genre
- Tracklänge im MM:SS Format
- Pfad zur Datei

Der folgende PL/SQL Block generiert dabei den Inhalt:

```
declare
  duration songs.duration%type;
  path songs.path%type;
  artist artists.artist%type;
  album albums.album%type;
  title titles.title%type;
  genre genres.genre_name%type;
begin
  select songs.duration, songs.path, artists.artist, albums.album, titles.title, genre
    into duration, path, artist, album, title, genre
  from songs
   join artists on artists.artist_id = songs.artist_id
   join albums on albums.album_id = songs.album_id
   join titles on titles.title_id = songs.title_id
   join genres on genres.genre_id = songs.genre_id
   where songs.title_id = v('SELECTED_TITLE_ID') AND
   rownum <= 1;

  htp.p('<b>' || title || '</b> by <b>' || artist || '</b> on </b> ' || album);
  htp.p('<br />');
  htp.p('<br />');
  htp.p('<b>Genre:</b> ' || genre);
  htp.p('<br />');
  htp.p('<b>Length:</b> ' || trunc(duration / 60) || ':' || mod(duration, 60));
  htp.p('<br />');
  htp.p('<b>Path:</b> ' || path);
end;
```

Lyrics and Metadata Information:

In dieser dynamischen PL/SQL Region wird der Liedtext des aktuellen Stückes angezeigt, sowie einige weitere Meta-Information über das Stück wie:

- Pfad zur Datei (nochmals)
- Dateum an dem das Stück importiert wurde.

Der folgende PL/SQL Block generiert dabei den Inhalt:

```
declare
  lyrics titles.lyrics_text%type;
  artist artists.artist%type;
  album  albums.album%type;
  title  titles.title%type;
  file   songs.path%type;
  date_added songs.date_added%type;
begin
  select distinct titles.lyrics_text, titles.title, songs.path, songs.date_added
    into lyrics, title, file, date_added
  from songs
    join titles on songs.title_id = titles.title_id
    where titles.title_id = v('SELECTED_TITLE_ID') AND rownum <= 1;
```

```
  select distinct artists.artist into artist from songs
    join artists on songs.artist_id = artists.artist_id
    where artists.artist_id = v('SELECTED_ARTIST_ID');

  select distinct albums.album into album from songs
    join albums on songs.album_id = albums.album_id
    where albums.album_id = v('SELECTED_ALBUM_ID');

  htp.p('<b>' || title || ' </b>by<b> ' || artist || ' </b>on<b> ' ||
        album || '</b>' ||
        '<br /><br />' || lyrics ||
        '<br /><br /><br /><hr />' || '<b>File Path: </b>' || file ||
        '<br />' || '<b>Was added: </b>' || date_added
  );
end;
```

Playlist Selection:

Durch dieses Dropbdown Menü kann der momentane Song einer bestimmten Playlist zugeordnet werden. Innerhalb der Selection List werden dabei die vorhandenen Playlists angezeigt. Wählt man eine Playlist aus so wird die Ausführung des folgenden PL/SQL Codes getriggert (durch eine Dynamic Action):

```
begin
  insert into playlist_to_songs(playlist_id, song_id)
         values(:PLAYLISTBOX, v('SELECTED_TITLE_ID'));
end;
```

Sicherheitshalber wird das Setzen der Playlist durch folgende Validation noch überprüft:

```
if v('PLAYLISTBOX') in (select playlist_id from playlists)
then
   return true;
else
   return false;
end if;
```

“Album this Title is on”-Button:

Leitet auf die Album Detail View des gerade selektierten Albums zurück.

Der Statistics Tab bietet einige HTML5-Charts an die allgemeine Informationen über die Musikdatenbank vermitteln.

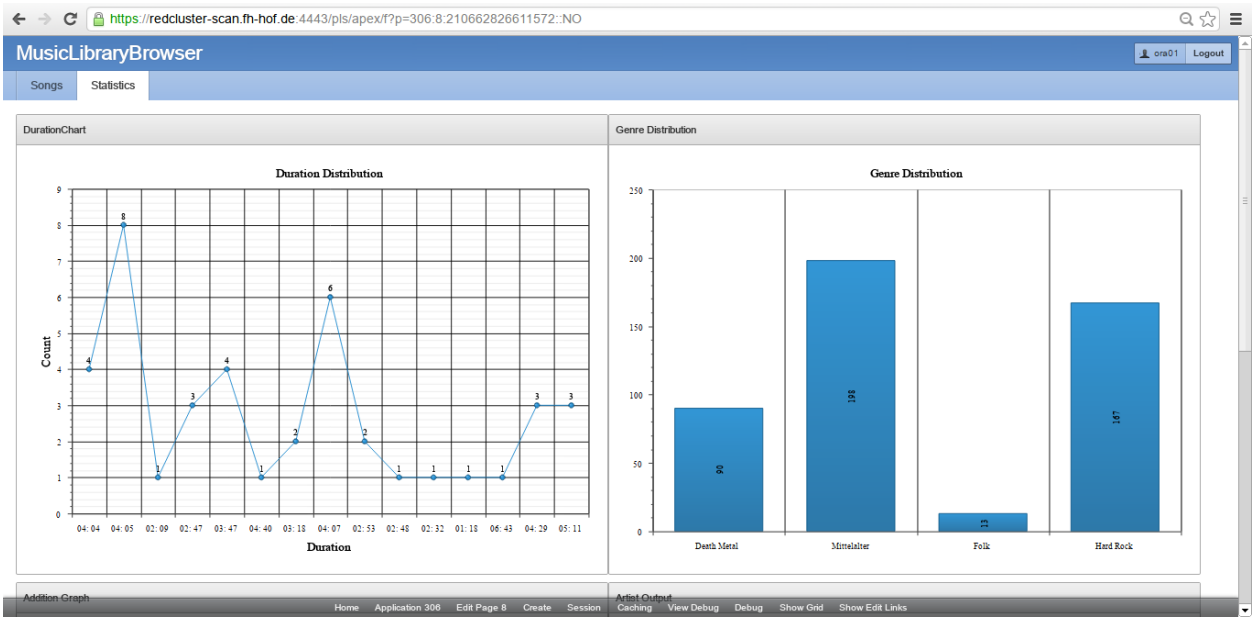

Figure 6: Ein Screenshot des Statistics Tab.

Charts

Duration Chart:

Zeigt die Anzahl von Songs die eine bestimmte Länge haben. Acht Songs haben beispielsweise die Länge 4:05.

Genre Distribution:

Zeigt die Anzahl von Lieder die zu einem Genre zugehörig sind.

Additions Graph:

Zeigt den zeitlichen Verlauf von importierten Songs. Da hier die Datenbank in einem Rutsch von einem Script genertiert wurde ist hier nur ein einzelner Wert vorhanden.

Artist Output:

Zeigt an wieviel Lieder ein einzelner Artist in der Datenbank hat.

Abgesehen von den oben gelistente Seiten gibt es noch einige Seiten die hier nur kurz aufgelistet werden, da sie bereits von Apex beim Anlegen der Applikation erzeugt wurden:

- **Login** - Seite die beim Login angezeigt wird
- **Update** - Seite die beim Editieren einer Row im interaktivieren Report angezeigt wird
- **Delete** - Seite die beim Löschen einer Row im interaktiven Report angezeigt wird
- **Success** - Wird beim erfolgreichen Inserten in den Interaktiven Report angezeigt

Es folgt eine Liste der Anforderungen, welche sinngemäß aus der Aufgabenstellung übernommen wurde, sowie eine Beschreibung wie diese in der Studienarbeit erfüllt worden sind:

Berichte:

Reports sind am häufigsten enthalten. Sie finden sich in der Auflistung der Playlist, der Auflistung der Similar-Artists (mit Bildern im Report) u.v.m.

Für die meisten Anwendungszwecke wurde hier der einfache Classic Report gewählt, da dieser genug Funktionalität bietet. Lediglich der Report auf der Mainseite wurde als Interaktiver Report realisiert.

Seitenverzweigung:

Seitenverzweigungen finden sich auf jeder Seite. Klickt man auf den Artist im Main Report so wird man zur Artist Detail View weitergeleitet.

Schaltflächen:

Buttons wurden ebenfalls großzügig genutzt. Beispielsweise auf der Hauptseite um alle Songs anzuzeigen.

Berechnungen:

Computations wurden genutzt um die vorhandenen Application Items auf sinnvolle Default Werte zu setzen.

Registerkarten:

Registerkarten wurden genutzt um die Hauptanwendung von der Statistikseite abzugrenzen.

verschiede Navigationsmöglichkeiten:

Die Navigation zwischen den verschienden DetailViews ist einerseits durch Buttons mögich anderseits auch durch Klick auf den Namen der Anwendung um jederzeit zur Hauptseite zurückzukommen.

Bilder:

Die Anwendung setzt stark auf Bilder zur Illustration. Dies umfasst:

- Artist Photos für jeden Künstler
- Album Cover für jedes Album

Dazu gesellen sich Lyrics Texte für jedes Lied und Artist Biographien für jeden Künstler. Letztere sind auf englischer Sprache und daher recht kurz weil es sich meist um deutsche Bands im Beispieldatenset handelt.

verschiedene Diagramme. :

Im *Statistik* Reiter finden sich vier verschiedene Diagramme, welche bereits weiter oben erklärt wurden.

Repräsentative Anzahl von Daten:

Insgesamt wurden 461 Songs in die Datenbank aus meiner privaten Sammlung exportiert.

Tooltips:

Zu allem Buttons wurden Tooltips hinzugefügt die deren Wirkweise genauer beschreiben.

Navigationspfade:

Es gibt mehrere Möglichkeiten durch die Anwendung zu navigieren: Durch Nutzung...

- ...der Links in den Reports
- ...der Navigationsbuttons die es in den Detail Seiten gibt.
- ...der Breadcrumbs im Header
- ...des Headers der bei einem Klick auf die Main Seite weiterleitet.

Wertelisten:

Wertelisten wurde in der Titel Detai View verwendet um den Inhalt einer Selection List zu füllen.

Forms:

Forms sind in Form eines einfachen "Add new Playlist" Form enthalten dass auf der Hauptseite zu finden ist. In dieser könnnen auf einfachste Art und Weise neue Playlisten hinzugefügt werden. Songs können in der Title Detail View bestimmten Playlists hinzugefügt werden.

Validierungen:

Validierungen wurden genutzt um die gewählte Playlist in der TitleView zu validieren. Ist sie nicht in der `playlists` Tabelle vorhanden so wird die falsche Auswahl mit einer Fehlermeldung quittiert. Normal sollten allerdings keien falschen Playlist angezeigt werden. Die Validierung dient also nur der Absicherung.

Abschließend noch meine eigene Meinung zu Apex. Apex ist sicherlich ein einfaches Framework um bereits vorhandene Datenbank zu in Reports und Charts zu visualisieren. Auch Pages lassen sich noch relativ leicht anlegen. Aber was darüber hinaus geht scheint mir sehr kompliziert gelöst, bzw. so als ob man ein vorhandenes Konzept in ein Datenbank Kontext zu quetschen versucht hätte. Deshalb fühlen sich Lösungen so an als ob man um Apex herumarbeiten müsste: So müssen beispielsweise müssen für Tooltips JavaScript HTML Attribute gesetzt werden, und das für jeden Button neu.Insgesamt hat Apex für Personen die viel mit Reports zu tun haben, wie beispielsweise Beamte oder Personen die Apex als einfaches CMS benutzen wollen durchaus eine Daseinsberichtigung. Meiner Meinung aber ist Apex für einen aber Informatikkurs nicht wirklich sinnvoll. Sinnvoller wäre hier meiner Meinung eher eine allgemeinere Vorlesung über Webentwicklung die sich nicht zu stark an ein bestimmtes Produkt knüpft.

List of Figures