FIPLY

Daniel Bersenkowitsch, Andreas Denkmayr, Gerald Irsiegler

19. Februar 2016

Contents

Andreas Denkmayr 19. Februar 2016

1 Musik

1.1 Lokalisierung der Musikdateien

Am Beginn der Arbeit wurde der Music-Ordner nach mp3-Files durchsucht.

`Dabei wurde der Music-Ordner mit Hilfe eines FileExtensionFilters nach mp3-Dateien durchsucht.`

```
File home = new File(Environment.getExternalStorageDirectory().
    getAbsolutePath() + "/Music";);
songs = new ArrayList<>();
if (home.listFiles(new FileExtensionFilter()) != null) {
  for (File file : home.listFiles(new FileExtensionFilter())) {
        HashMap<String, String> hm = new HashMap<>();
                hm.put("songTitle", file.getName());
                  hm.put("songPath", file.getPath());
                   songs.add(hm);
  }
}
```

Im Laufe der Entwicklung stellte sich heraus, dass jeder Benutzer seine Musikdateien in einem anderen Ordner und in verschiedenen Dateiformaten abspeichert. Die Lösung für dieses Problem stellt der Android Mediastore dar. Über diesen können Abfragen nach verschiedenen Medientypen z.B.: Musik, Fotos oder Videos durchgeführt werden. Von diesen Medientypen können Name, Pfad, Dateigröße und vieles mehr ausgelesen werden.

`Die Datenabfrage gegenüber dem Mediastore erfolgt über einen ContentResolver mit Hilfe der query()-Methode.`

```
public final @Nullable Cursor query(@NonNull Uri uri, @Nullable
    String[] projection, @Nullable String selection, @Nullable
    String[] selectionArgs, @Nullable String sortOrder) {
  return query(uri, projection, selection, selectionArgs,
    sortOrder, null);
}
```

`Für den Musicplayer benötigen wir alle Audiodateien die Musik beinhalten.`

```
ContentResolver cr = context.getApplicationContext().
    getContentResolver();
Uri uri = MediaStore.Audio.Media.EXTERNAL_CONTENT_URI;
```

```
String selection = MediaStore.Audio.Media.IS_MUSIC + "!= 0";
String sortOrder = MediaStore.Audio.Media.TITLE_KEY + " ASC";
Cursor cur = cr.query(uri, null, selection, null, sortOrder);
```

Für jede Zeile im Cursor cur wird eine HashMap aus 2 Strings erstellt die den Titel und den Pfad eines Songs beinhaltet. Diese HashMaps werden anschließend zu einer ArrayList hinzugefügt.

```
songs = new ArrayList<>();
HashMap<String, String> hm = new HashMap<>();
while(cur.moveToNext())
{
  hm.put("songTitle", cur.getString(cur.getColumnIndex(
    MediaStore.Audio.Media.TITLE)));
  hm.put("songPath", cur.getString(cur.getColumnIndex(MediaStore
    .Audio.Media.DATA)));
  songs.add(hm);
}
```

1.2 Verwalten von Playlists

Im ersten Screenshot sieht man die Auswahl der erstellten Playlists, wobei die Playlist "All' nicht bearbeitet werden kann und alle eingelesenen Songs darstellt. Zusätzlich zu der "All"-Playlist kann der Benutzer eigene Playlists anlegen und diese auch bearbeiten. Der All und der None Button helfen dem Benutzer schnell alle Songs zu markieren oder die Markierung aller Elemente aufzuheben.

Wird der Save-Button gedrückt wird für alle Positionen abgespeichert ob das Element an der jeweiligen Position markiert ist. Dies erfolgt über ein SparseBooleanArray. Anschließend wird eine Liste erstellt in der nur die angekreuzten Elemente enthalten sind. Diese Liste wird als eine neue Playlist in die PlaylistSongs-Tabelle gespeichert. Dabei wird als Playlistname der eingetragene Titel in das EditText unter der ListView übernommen.

```
SparseBooleanArray checked = lvSongs.getCheckedItemPositions();
for (int i = 0; i < songs.size(); i++) {
    if (checked.get(i)) {
        checkedSongs.add(songs.get(i));
    }
}
psrep.reenterPlaylist(etName.getText().toString(), checkedSongs)
    ;
```

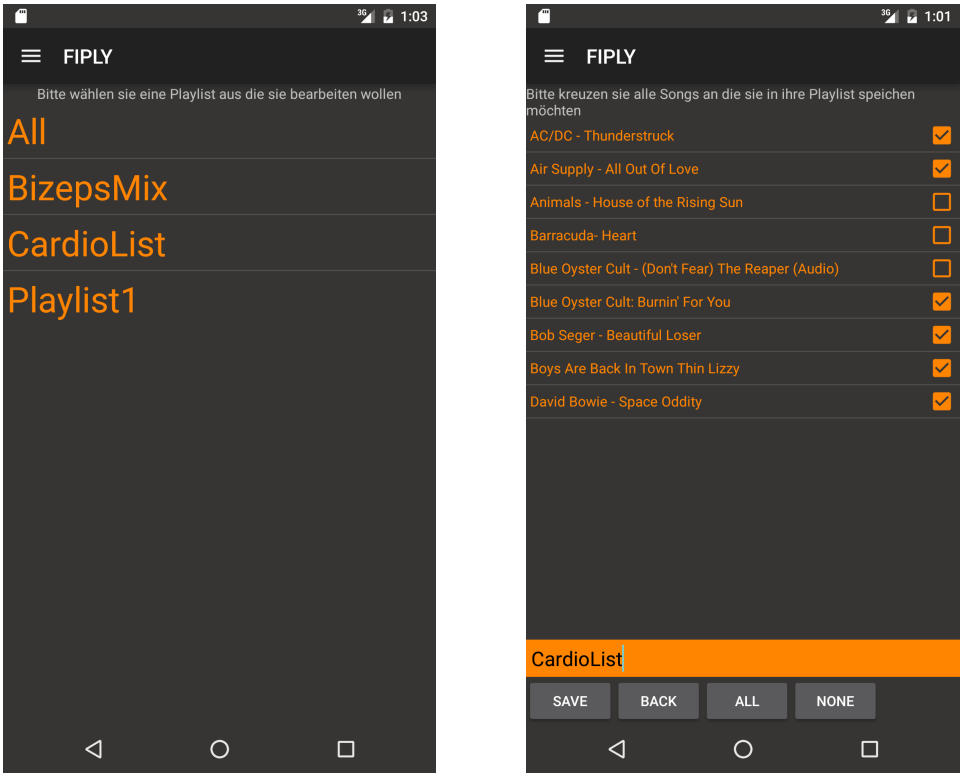

Figure 1: Bei klicken eines Elements in Screenshot 1 werden alle Songs angezeigt (Screenshot 2) und mittels der Checkbox sind alle Songs gekennzeichnet die sich in der ausgewählten Playlist befinden.

1.3 Abspielen der Playlists

Während der Benutzer sich in einer Traingssession befindet kann jederzeit die Musikwiedergabe gestartet werden. Bei Klick auf den Play-Button wird die "All"-Playlist in alphabetisch aufsteigender Reihenfolge abgespielt.
Die Playlist beziehungsweise der aktuell abgespielte Song kann geändert werden indem in den Musikmodus gewechselt wird.
Im Musikmodus wird über den Spinner die Playlist gewechselt.
In der aktuell ausgewählten Playlist kann man direkt zu einem bestimmten Song wechseln. Dieser wird abgespielt und nach Ende des Songs wird sofort der nächste Playlisteintrag gestartet.

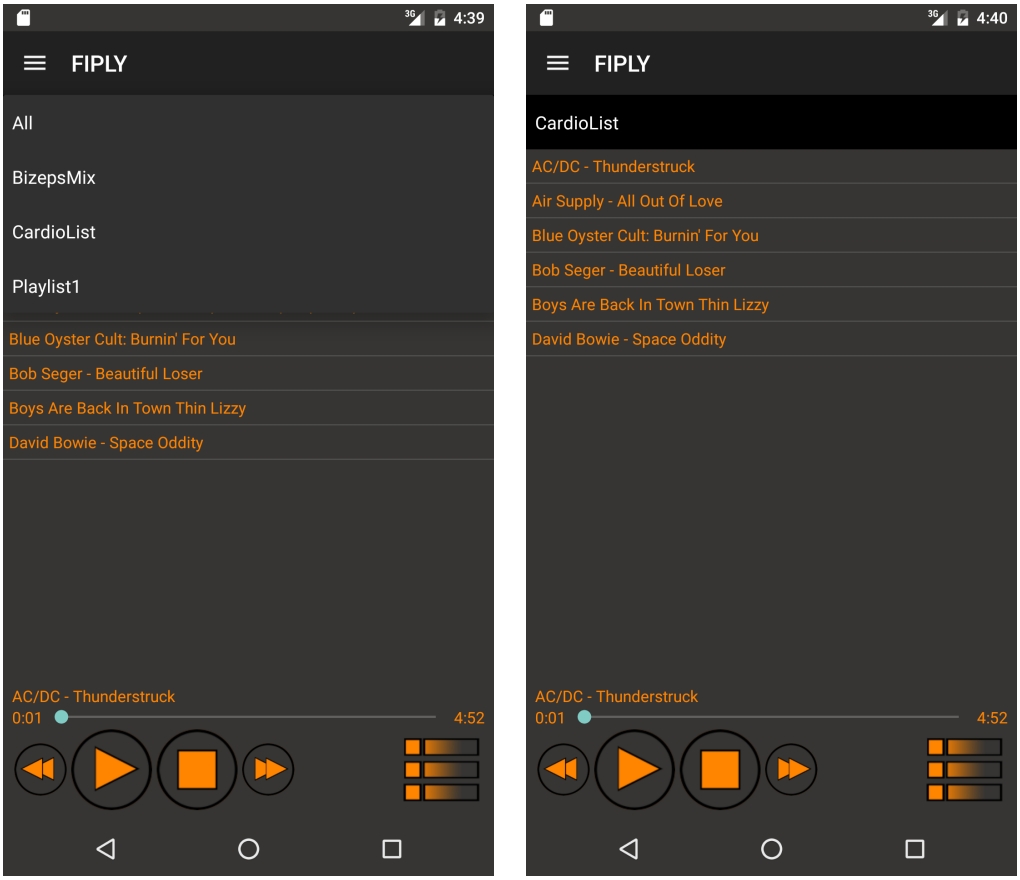

Figure 2: In einem Spinner kann eine Playlist ausgewählt werden. Bei Klick auf einen Song wird dieser abgespielt.

1.4 MusicControls

- Durch den Zurück und durch den Weiter Button kann auf den vorherigen beziehungsweise auf den nächsten Song gewechselt werden.
- Der Play Button dient dem starten der Musikwiedergabe.
  Während der Musikwiedergabe erscheint an dieser Stelle der Pause Button mit dem man die Musik pausieren kann.
- Der Stop Button beendet die aktuelle Wiedergabe und setzt den Wiedergabefortschritt auf den Beginn des Songs.
- Der Musikmodus Button stellt den aktuellen Modus durch Einfärbung des Buttons dar und ermöglicht einen Wechsel zwischen dem Übungsmodus und dem Musikmodus.

Figure 3: Die MusicControls in Ruhe und während einer Wiedergabe.

Im Übungsmodus wird die aktuelle Übung und die Anweisungen zum Trainieren angezeigt.
Der Musikmodus hingegen ermöglicht ein Wechseln der Playlist und den manuellen Wechsel auf einen bestimmten Song.

- Links von der Fortschrittsleiste wird der Fortschritt des aktuellen Songs im hh:mm:ss Format (ISO 8601) angezeigt.
- Die Fortschrittsleiste wird durch eine SeekBar über diesen Buttons implementiert. Diese SeekBar stellt den aktuellen Fortschritt des aktuellen Songs dar.
  Bei Klicken auf oder Ziehen an der Fortschrittsleiste kann man den Fortschritt der Wiedergabe manipulieren.
- Rechts von der Fortschrittsleiste wird die Gesamtdauer des Songs im hh:mm:ss Format (ISO 8601) angezeigt.
- Der Name des aktuellen Songs wird in einer TextView über den Fortschrittsanzeigen dargestellt.

Andreas Denkmayr 20. Dezember 2015

2 Fragments

2.1 Was sind Fragments?

Ein Fragment stellt einen Teil der Benutzeroberfläche einer Activity zur Verfügung, dabei kann man mehrere Fragments in einer Activity verwenden und diese zur Laufzeit auswechseln. Da Fragments in mehreren Activities wiederverwendet werden können, müssen Ansichten wie Detailviews oder Listen nur einmal programmiert werden und können überall eingesetzt werden. Fragments werden ab Android 3.0 (API level 11) zur Verfügung gestellt.

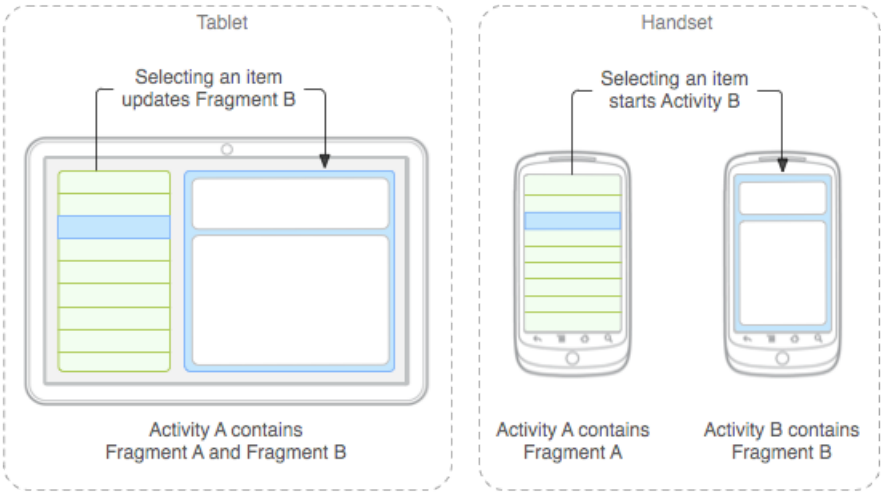

Figure 4: Beispiel man kann mithilfe von Fragments Views erstellen, die sowohl auf einem Tablet als auch auf einem Handy ein optimales Benutzerinterface anbieten

2.2 Der Lifecycle

Ein Fragment ist immer eingebunden in eine Activity und ist direkt vom Lifecycle der übergeordneten Activity abhängig. Wird die übergeordnete Activity pausiert oder zerstört werden auch alle untergeordneten Fragments pausiert oder zerstört. Das Aufbauen der Benutzeransicht erfolgt in der onCreateView() Methode.

`In einem Fragment:`

```
1 @Override
2 public View onCreateView(LayoutInflater inflater, ViewGroup
    container, Bundle savedInstanceState) {
3         super.onCreate(savedInstanceState);
4         return inflater.inflate(R.layout.example_fragment,
    container, false);
5     }
```

2.3 Fragment Transactions

Mittels Transaktionen lassen sich Fragments hinzufügen, entfernen oder ersetzen. Es werden mehrere dieser Aktionen hintereinander abgesetzt und zusammen nach einem commit() ausgeführt. Ein Fragment kann mittels addToBackStack() auch zum BackStack hinzugefügt werden um dadurch, ähnlich wie bei Activites, Navigation mit dem BackButton zu ermöglichen. Dabei ist zu beachten, dass alle Aktionen vor einem commit() gemeinsam auf den BackStack gelegt werden und bei drücken des BackButtons alle gemeinsam aufgehoben werden. Wird addToBackStack() nicht aufgerufen, wird ein Fragment beim Schließen oder beim Wechseln auf ein anderes Fragment zerstört und kann nicht mehr aufgerufen werden.

```
1         FragmentManager fragmentManager = getFragmentManager();
2         FragmentTransaction fragmentTransaction =
    fragmentManager.beginTransaction();
3         fragmentTransaction.addToBackStack(null);
4         fragmentTransaction.replace(R.id.fraPlace, fragment);
5         fragmentTransaction.commit();
```

2.4 Verwendung von Fragments

In dieser Arbeit werden Fragments verwendet, um die Benutzeransichten, ausgenommen des NavigationDrawers, anzuzeigen. Dabei wird ein FrameLayout im Layoutfile der MainActivity durch ein Fragment mittels der displayView() Methode ersetzt. Navigation durch diese Fragments wird mittels den Buttons im MFragment oder dem NavigationDrawer ermöglicht.

`MainActivity.java und MFragment.java:`

```
1 private void displayView(Fragment fragment) {
2         FragmentManager fragmentManager = getFragmentManager();
3         FragmentTransaction fragmentTransaction =
    fragmentManager.beginTransaction();
4         fragmentTransaction.addToBackStack(null);
5         fragmentTransaction.replace(R.id.fraPlace, fragment);
6         fragmentTransaction.commit();
7     }
```

`activity_main.xml:`

```
1 <FrameLayout
2         android:id="@+id/fraPlace"
3         android:layout_width="match_parent"
4         android:layout_height="match_parent" />
```

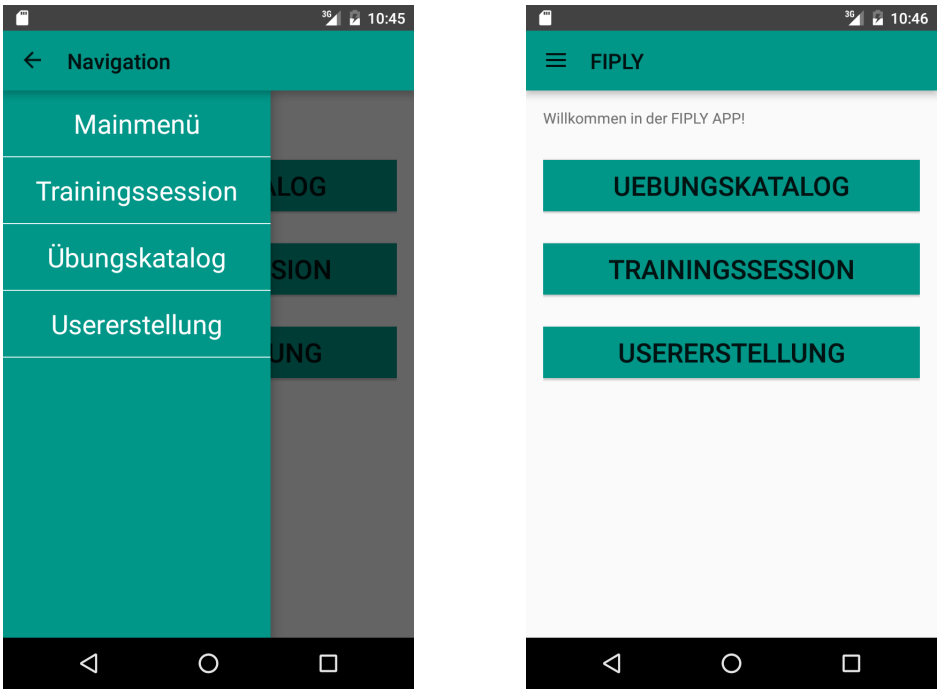

Figure 5: Bild des NavigationDrawers und des FMains

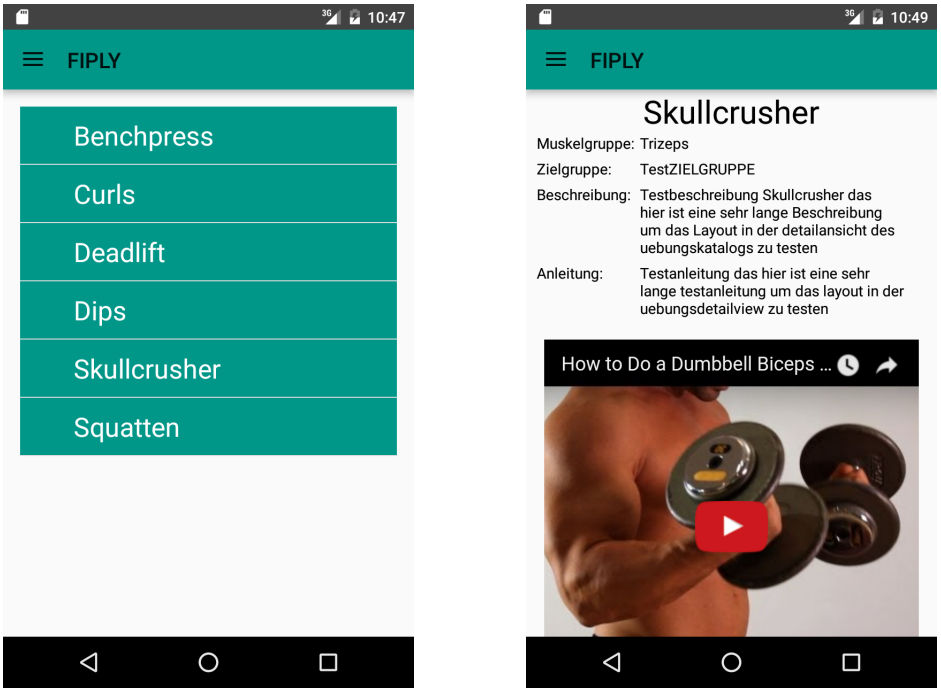

Figure 6: Bei Klicken eines Elements in der ListView wird die zugehörige DetailView aufgerufen