

Berufsakademie Sachsen  
Staatliche Studienakademie Leipzig



# MediaDB Viewer

## Pflichtenheft

Autoren

Markus Then cs13-2  
Nico Schallehn cs13-2

# Inhaltsverzeichnis

|   |    |
|---|----|
| 1. Vorüberlegungen.....                                       | 4  |
| 1.1. Benutzer und Zielgruppe.....                             | 4  |
| 2. Spezifiziertes Ziel.....                                   | 4  |
| 2.1. Muss-Kriterien.....                                      | 5  |
| 2.2. Soll-Kriterien.....                                      | 5  |
| 2.3. Kann-Kriterien.....                                      | 6  |
| 3. Systemvoraussetzungen.....                                 | 6  |
| 3.1. Android Client.....                                      | 6  |
| 3.2. API Server.....  | 7  |
| a) Datenbank.....   | 7  |
| b) PHP und Apache.....  | 7  |
| 4. Umsetzung.....   | 8  |
| 4.1. Datenbankstruktur.....                                   | 8  |
| 4.2. Oberfläche Client.....                                   | 9  |
| a) Navigation.....  | 10 |
| b) Listenansichten der Filme, Serien, Staffeln, Episoden..... | 10 |
| c) Filtermenü für Filme.....                                  | 11 |
| d) Detailansicht Film, Episode.....                           | 11 |
| e) Einstellungslayout.....                                    | 12 |
| 4.3. Datenmanagement Client.....                              | 12 |
| a) Erstellung der URL.....                                    | 12 |
| b) Parsen der JSON-Strings.....                               | 12 |
| c) Download der Cover.....                                    | 13 |
| 4.4. Umsetzung der API.....                                   | 13 |
| 5. Abkürzungsverzeichnis.....                                 | 14 |

## Abbildungsverzeichnis

|   |    |
|---|----|
| Abbildung 1: Anwendungsfalldiagramm.....        | 4  |
| Abbildung 2: Datenbankschema.....               | 8  |
| Abbildung 3: Zustandsdiagramm.....              | 9  |
| Abbildung 4: Navigationsmenü.....               | 10 |
| Abbildung 5: Filtermenü.....                    | 10 |
| Abbildung 6: Listenansicht.....                 | 10 |
| Abbildung 7: Detailansicht.....                 | 11 |
| Abbildung 8: Einstellungen.....                 | 11 |
| Abbildung 9: Attributauswahl.....               | 11 |
| Abbildung 10: Cover.....                        | 13 |
| Abbildung 11: Grundgerüst der JSON Ausgabe..... | 13 |

# 1. Vorüberlegungen

Ziel dieses Projekts ist es, eine Verwaltungsoberfläche für eine bereits existierende Datenbank zu entwerfen und zu implementieren. Aufgrund der weiten Verbreitung soll Android als Zielplattform für die Clientsoftware genutzt werden. Somit wird gleichzeitig gewährleistet, dass man den Viewer nahezu immer bei sich hat.

## 1.1. Benutzer und Zielgruppe

Zur Zielgruppe gehört jeder, der eine Film- bzw. eine Seriensammlung besitzt. Darüber hinaus aber auch dessen Gäste, die sich bei ihrem Besuch für einen Film entscheiden sollen, der anschließend angesehen wird. Für diese Zielgruppe soll eine übersichtliche grafische Darstellung der Filme und Serien angeboten werden. Anhand der angebotenen Informationen soll die Auswahl erleichtert werden.

## 2. Spezifiziertes Ziel

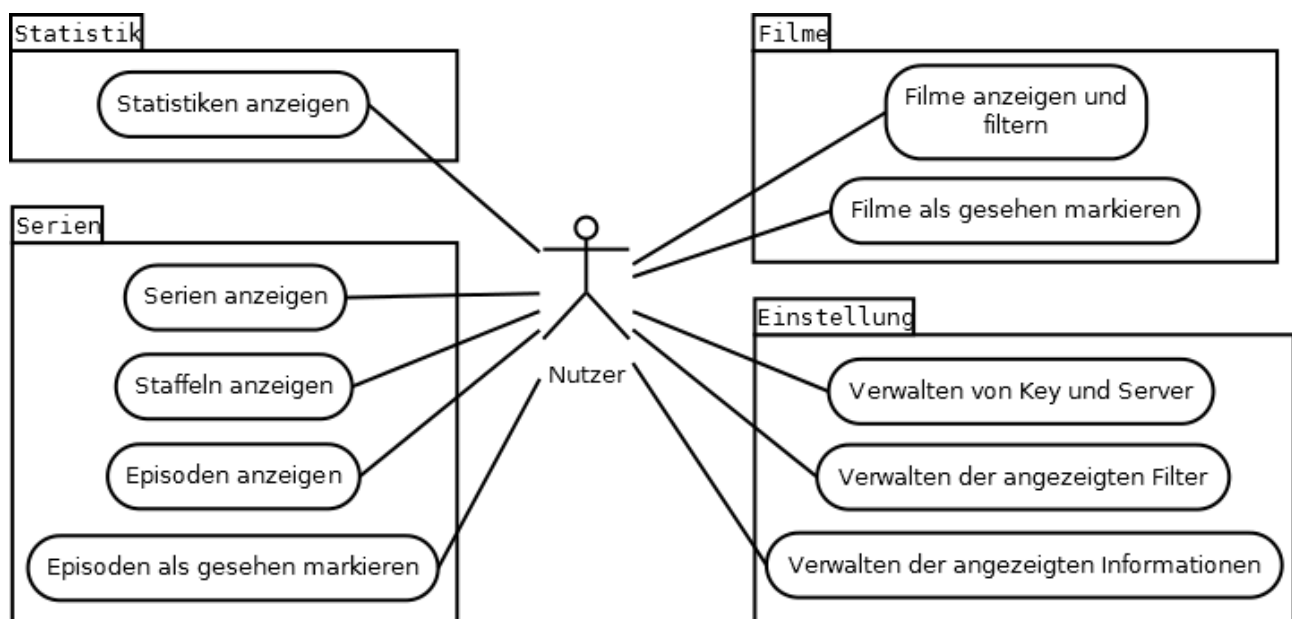


Abbildung 1: Anwendungsfalldiagramm

Im Mittelpunkt des Projekts steht die Entwicklung der Android Anwendung unter dem Namen *MediaDB Viewer*. Da die Anwendung auch für sehr unerfahrene Nutzer bedienbar sein soll, muss ein simples, aber gleichzeitig elegantes Design realisiert werden. Die Kernaufgabe der App besteht in der Darstellung der Datenbankinhalte. Die wichtigsten Funktionen sind in Abbildung 1 zu sehen.

Da die Daten von der Datenbank zur App übertragen werden müssen, wird als Grundlage eine API implementiert. Sie soll in PHP geschrieben und als Server betrieben werden. Die Daten werden dann von der App bei der API via HTTP angefragt und als JSON Objekt übertragen.

Für eine ästhetische Darstellung mit Covern wird noch eine dritte Komponente benötigt, die diese bereitstellt. Der Einfachheit halber sollen sie von demselben Webserver angeboten werden, der auch die API betreibt.

Es ist nicht Teil dieses Projektes die Datenbank mit Daten zu befüllen. Sie wird als bereits existent und mit Daten initialisiert angesehen. In Abbildung 2 ist die Struktur der Datenbank zu sehen und kann entsprechend selber implementiert werden.

## 2.1. Muss-Kriterien

Nachfolgende Kriterien müssen während des Projektes umgesetzt werden:

- Implementierung der API auf der Serverseite
  - Bereitstellung sämtlicher Daten die in der App angezeigt werden sollen:
    - Filmübersicht, Serienübersicht, Staffelübersicht, Episodenübersicht
    - detaillierte Daten zu einer Episode / einem Film
  - Funktionen zum aktualisieren der Datenbankeinträge (für Episoden als auch Filme):
    - Markierung als fehlerhaft / fehlerfrei
    - Anzahl der Wiedergaben (Views) inkrementieren
    - Kommentar setzen
  - Absicherung der API durch Zugangsschlüssel mit verschiedenen Rechten
- Implementierung der Android Anwendung
  - Liste aller Filme
  - Liste aller Serien, Staffeln und Episoden
  - Detailansicht für Film und Episode
  - Einstellungsmenü um den Server auswählen zu können und den Zugangsschlüssel zu übergeben
  - Prüfe auf gesetzten Zugangsschlüssel und Server (Popup wenn nicht gesetzt)

## 2.2. Soll-Kriterien

Nachfolgende Kriterien sollten während des Projektes umgesetzt werden:

- Implementierung der API auf der Serverseite
  - Sortierungen und Filter für Filme:
    - Sortierung nach verschiedenen Kriterien
    - aufsteigende, absteigende Sortierung

- Möglichkeit der Abfrage von Statistiken (entsprechen Views in Datenbank)
- Verhinderung von SQL Injections
- Implementierung der Android Anwendung
  - Filter und Sortierung für Filme implementieren
  - Übersicht über Statistiken der bereits gesehenen Filme und Serien
  - Auswahl der anzuzeigenden Detailinformationen zu Filmen und Serien über Einstellungen

## 2.3. Kann-Kriterien

Nachfolgende Kriterien können während des Projektes umgesetzt werden (falls Zeit bleibt):

- Implementierung der API auf der Serverseite
  - Bereitstellung der Berechtigungsauskunft zu einem Zugangsschlüssel
- Implementierung der Android Anwendung
  - Zwischenspeicherung von Datensätzen zur Datenvolumeneinsparung
  - Abfrage der Berechtigungen eines Zugangsschlüssels und entsprechende Anpassung der Benutzeroberfläche (z.B. Verstecken des Update-Buttons)
  - Coverflow auf Startseite der App

## 3. Systemvoraussetzungen

### 3.1. Android Client

Für die Nutzung des Clients wird ein Gerät mit Android OS benötigt. Dabei spielt es keine Rolle, ob es ein Tablet oder Smartphone ist. Das Design der App wird so ausgelegt, dass es dynamisch auf verschiedene Displaygrößen und -auflösungen skaliert. Da möglichst aktuelle Design- und Navigationskonzepte genutzt werden sollen, wurde das minimal unterstützte API Level auf 14 gesetzt. Dieses entspricht der Android Version 4.0 *Ice Cream Sandwich*. Da jedoch ca. 90 % aller Android Geräte mindestens über diese Android Version verfügen, wurde die unterstützte Zielgruppe als ausreichend groß eingeschätzt.

Um die Informationen anzeigen zu können, muss der Client eine Netzwerkverbindung mit Internetzugriff oder lokalen Zugriff auf den Server haben.

Der Sekundärspeicher des Client muss lediglich ausreichen, um die Cover lokal zwischenspeichern zu können. Damit soll Datenvolumen bei der Datenabfrage gespart werden. Ein fester Wert kann hier nicht angegeben werden, da dieser sowohl von der Anzahl der Film- und Seriencover sowie deren Auflösung abhängt.

Sowohl an den RAM, als auch an die CPU des Android Gerätes gibt es keine besonderen Ansprüche. Da keine grafisch anspruchsvollen Effekte genutzt und die Cover bereits auf dem Server sinnvoll skaliert hinterlegt werden sollen, wird der Arbeitsspeicher nur minimal beansprucht. Eine sehr geringe Prozessorleistung würde sich maximal in einem Ruckeln beim Scrollen der Listen auswirken und ist somit unkritisch.

## 3.2. API Server

Damit der Android Client die Daten aus der Datenbank abrufen kann, wird ein Server benötigt, der immer dann verfügbar ist, wenn Daten angefordert werden. Um das zu vereinfachen läuft der Server 24 Stunden am Tag und 7 Tage in der Woche.

Leistungstechnisch reicht es, den Server auf einen Raspberry Pi zu betreiben. In unseren Fall soll eine Virtuelle Maschine die Aufgaben (Datenbankserver und Webserver) übernehmen.

In der Virtuellen Maschine wird ein Ubuntu Server in der Version 14.04.2 LTS als Testsystem laufen, dabei stehen dem Server 2 CPU Kerne mit einer Taktfrequenz von 1,9 GHz und 2 GB RAM zur Verfügung.

Damit die API auch aus dem Internet erreichbar ist, müssen folgende Dinge berücksichtigt werden:

- Einrichten einer Domain (über Dyn-DNS) für den Zugriff (ggf. auch über eine statische IP-Adresse möglich)
- Weiterleitung des Port 80 an den Server
- Installation und Einrichtung der nachfolgenden Komponenten

Es ist Außerdem möglich den API Server nur Lokal im Netzwerk zu betreiben, dafür entfallen die ersten 2 Punkte und man greift auf die lokale IP zu.

### a) Datenbank

Als Datenbankserver soll der MySQL-Server eingesetzt werden, da er unter der GPL Lizenz frei verfügbar ist. Auf unserem Testsystem wird ein MySQL-Server in Version 5.5.46 laufen. Die MediaDB (siehe Kapitel 4.1) muss angelegt sein und Benutzer mit entsprechenden Rechten müssen existieren.

### b) PHP und Apache

Als Schnittstelle zwischen der Datenbank und dem Web soll ein Webserver mit einer Erweiterung für dynamischen Seiteninhalt dienen. Für die Implementierung soll als Webserver der Apache2 und für das Erzeugen der dynamischen Inhalte PHP verwendet werden. Auf dem Testsystem ist Apache2 in Version 2.4.7 und PHP in der Version 5.5.9 installiert. Folgende zusätzliche Erweiterungen für PHP müssen installiert sein php5-json, php5-mysql und php5-sqlite.

## 4. Umsetzung

### 4.1. Datenbankstruktur

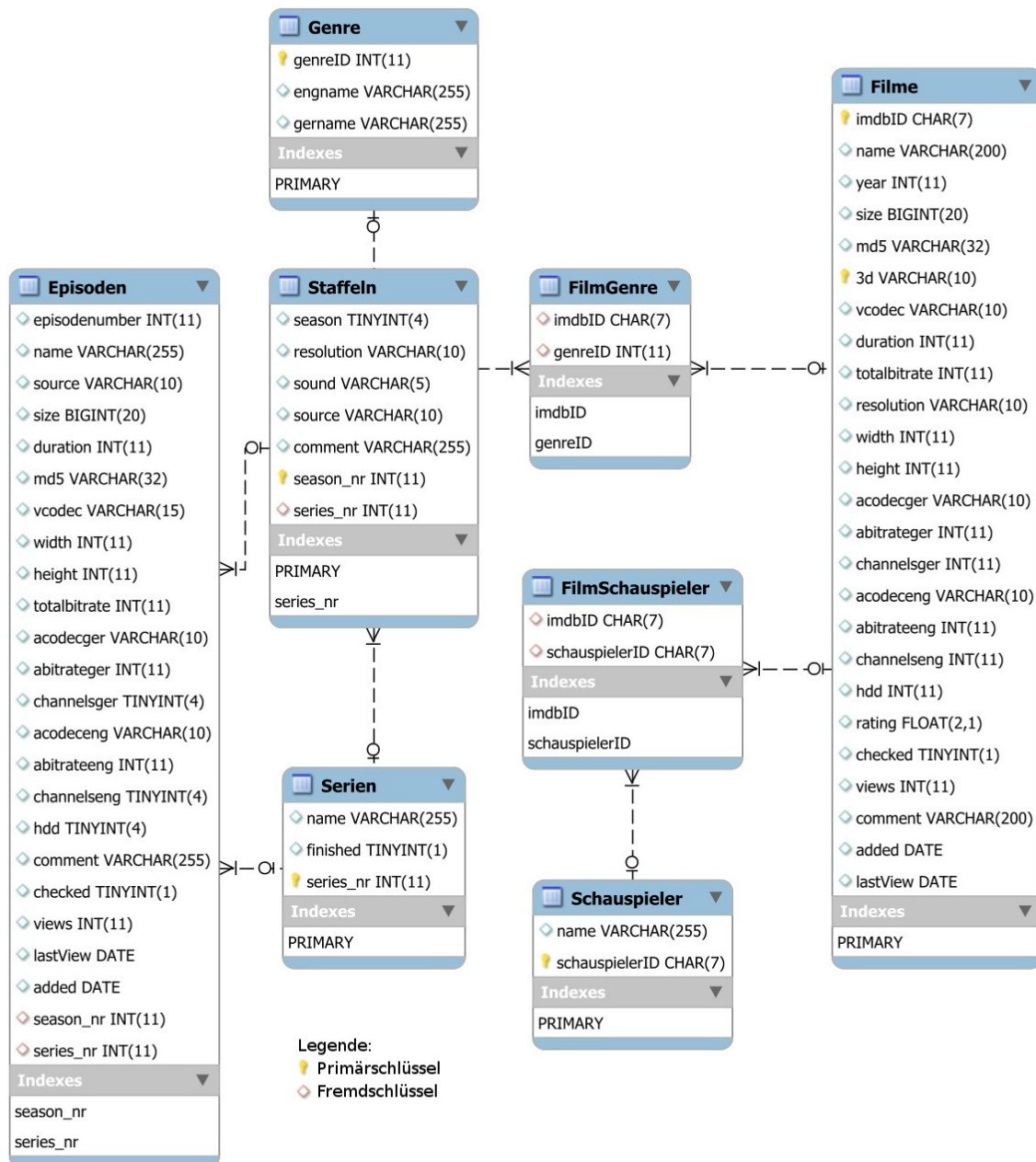


Abbildung 2: Datenbankschema



In Abbildung 2 ist die Struktur der zugrunde liegenden Datenbank zu sehen. Da die Datenbank bereits vor dem Beginn dieses Projektes existierte, wird nicht genauer auf jedes Attribute und die Befüllung der Datenbank eingegangen. Zusätzlich zu den Tabellen existieren noch Views, die auf den Daten der Tabelle beruhen und verschiedene Statistiken generieren. Die Spalte *lastView* wird durch einen Trigger auf das aktuelle Datum gesetzt, wenn der Inhalt der Spalte *views* verändert wird. D.h. wenn ein Film gesehen wurde wird das Datum gespeichert, damit man später nachvollziehen kann, wann er das letzte Mal angesehen wurde.

## 4.2. Oberfläche Client

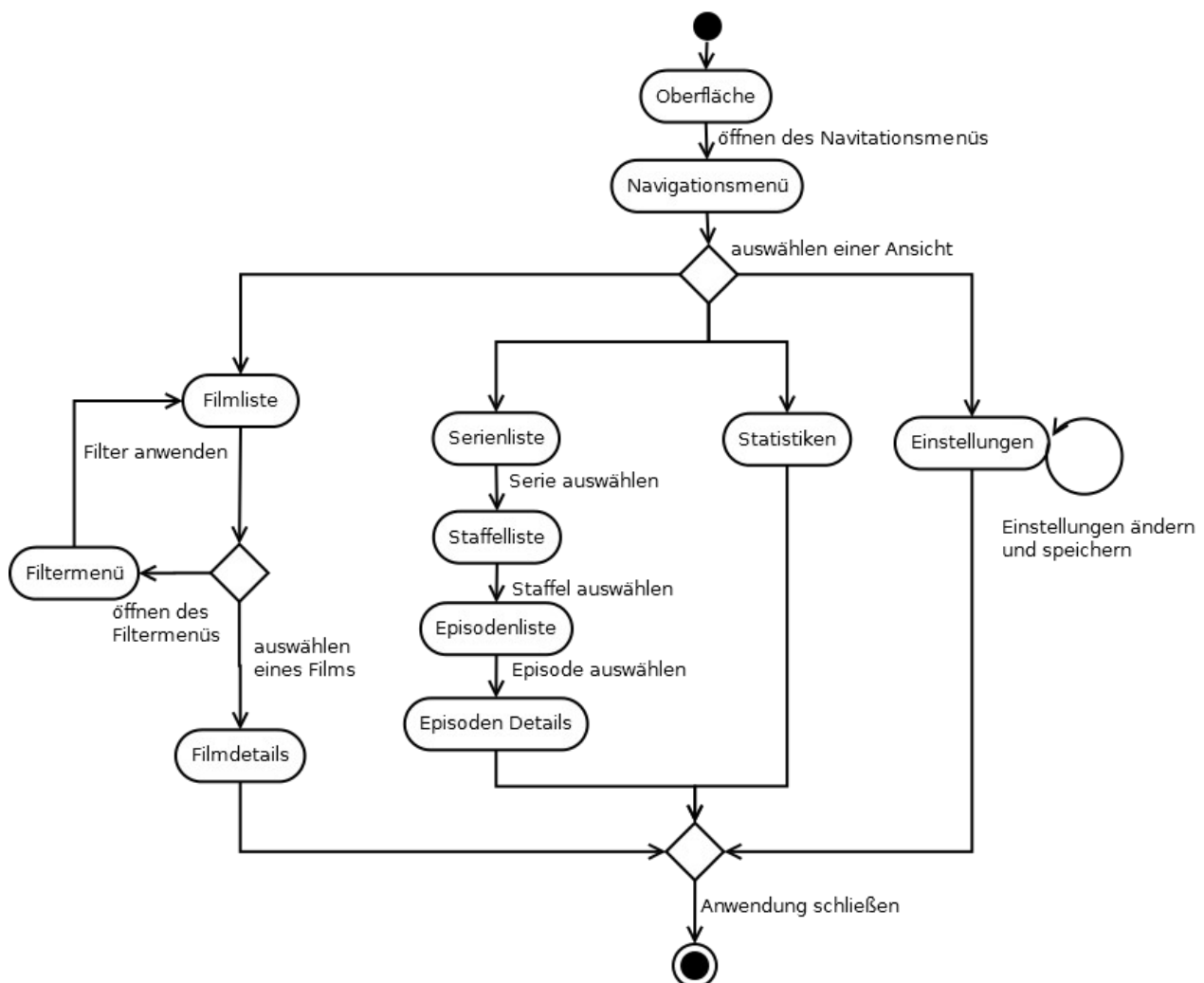


Abbildung 3: Zustandsdiagramm

Die in Abbildung 3 zu sehenden Zustände und Aktionen sollen über die Oberfläche möglich sein. Um die Übersichtlichkeit des Diagramms zu gewährleisten, wurden Aktionen wie zurück gehen weggelassen. Es soll von jedem Zustand aus auch möglich sein zurück in den vorherigen zu wechseln.

Um ein modulares und dynamischen Design zu entwerfen, wird als grundlegendes Layout das

sogenannte *DrawerLayout* genutzt. Dieses ermöglicht es wie in Abbildung 4 und 5 Drawer zu erstellen, die dann durch Wischen von links nach rechts oder umgekehrt eingeblendet werden können. Ein Drawer ist lediglich ein Panel, welches Navigationselemente oder Filter enthalten kann.

Da es sich bei dem *DrawerLayout* um ein modulares Layout handelt, bei dem die Drawer immer benutzbar sein sollen, werden alle Ansichten und Layouts als sogenannte Fragmente realisiert. Würde man für jede Ansicht ein neues Layout nehmen, wären die Drawer nicht immer aufrufbar. Fragmente sind dynamisch kombinierbare Oberflächenelemente, die es ermöglichen einzelne Teile des Layouts während der Laufzeit auszutauschen oder anzupassen. Man kann sie als eine Art Container für Layouts betrachten.

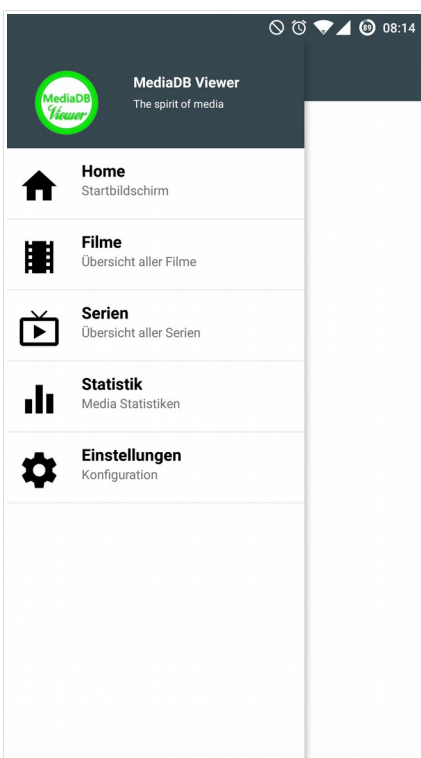


Abbildung 4: Navigationsmenü

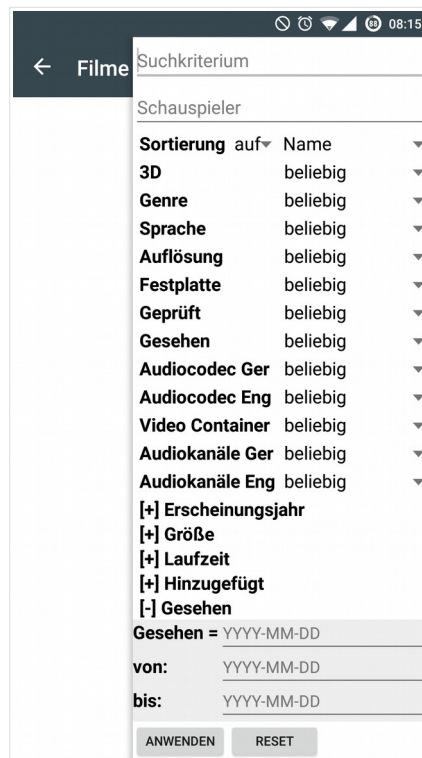


Abbildung 5: Filtermenü

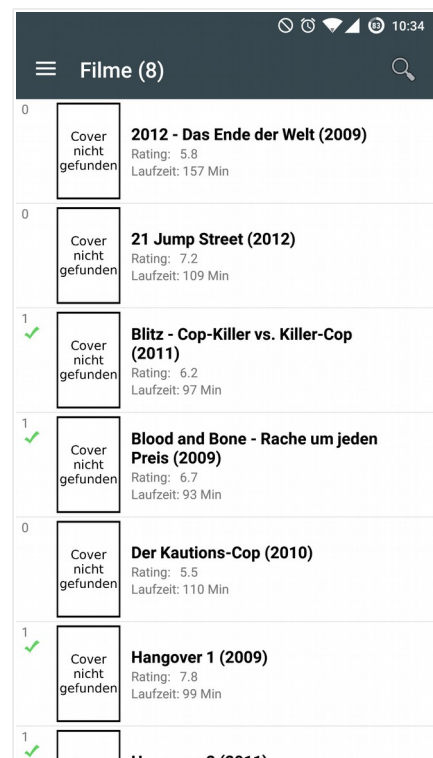


Abbildung 6: Listenansicht

## a) Navigation

Als zentrales Navigationselement der App dient der linke Drawer (siehe Abbildung 4). Dieser enthält immer Elemente mit denen zur Home-, Film-, Serien-, Statistik- oder Einstellungsansicht navigiert werden kann. Bei Auswahl eines Elements wird das aktuelle Fragment durch das entsprechend neue zu erstellende ersetzt und so die gewünschte Ansicht erzeugt.

## b) Listenansichten der Filme, Serien, Staffeln, Episoden

Für die Darstellung der Übersichten, werden *ListViews* verwendet (siehe Abbildung 6). Diese können über die Zuweisung eines Adapters befüllt werden. Dieser Adapter wird durch Vererbung und Überschreiben von Methoden an die jeweilige Liste angepasst.

Der Adapter kümmert sich darum, dass die Elemente der Liste erst erzeugt werden, wenn sie auch sichtbar sind. Darüber hinaus werden nicht mehr sichtbare Elemente recycelt. Durch diese Methoden wird der Arbeitsspeicher des Gerätes stark entlastet. Wird ein Element der Liste ausgewählt, so wird das aktuelle Fragment durch das der nächst tieferen Ebene ersetzt (siehe Abbildung 3).

### c) Filtermenü für Filme

Wie in Abbildung 5 zu sehen, soll über einen Drawer auf der rechten Seite die Möglichkeit bestehen nach Filmen und Schauspieler zu suchen. Darüber hinaus sollen Filter für alle Filmattribute existieren, bei denen eine Filterung sinnvoll durchgeführt werden kann. Zu unterscheiden sind komplexe und einfache Filter. Bei den einfachen, gibt es eine vorgegebene Auswahl von Elementen nach denen gefiltert werden kann. Bei komplexen Filter gibt es die Möglichkeit einen Bereich anzugeben in dem der Wert des Attributs liegen soll, um somit noch feiner Filter zu können.



Abbildung 7: Detailansicht

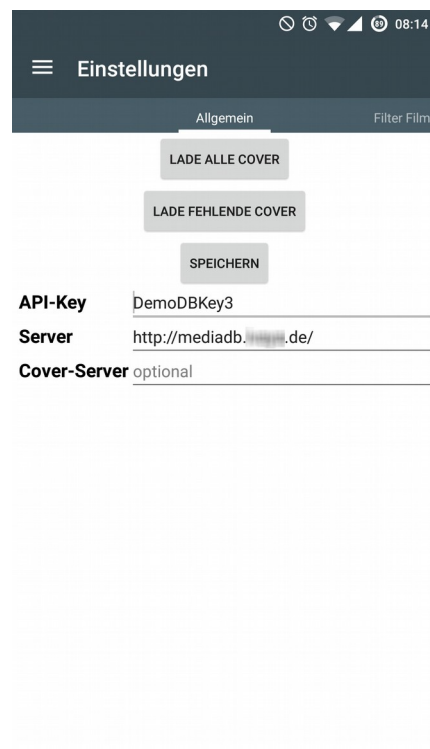


Abbildung 8: Einstellungen



Abbildung 9: Attributauswahl

### d) Detailansicht Film, Episode

Wird ein Film bzw. eine Episode ausgewählt, soll eine detaillierte Ansicht angezeigt werden. Die Attribute mit ihren Werten werden wie in Abbildung 7 als Tabelle dargestellt. Unterhalb der Tabelle gibt es einen Button, der ein Popup mit der Möglichkeit zum Update der in Kapitel 2.1 geforderten Daten in der Datenbank anzeigt. Handelt es sich bei dem dargestellten Element um einen Film, so wird zusätzlich oberhalb der Tabelle ein Cover dargestellt.

## e) Einstellungslayout

Das Einstellungsfragment besteht aus *SlidingViews*, die eine einfache Navigation durch mehrere Einstellungsseiten ermöglichen. Wird mit dem Finger von rechts nach links über den Bildschirm gestrichen oder oben in der Tableiste der nächste Tab ausgewählt, so öffnet sich eine neue Einstellungsseite.

Auf der ersten Seite sollen allgemeine Einstellungen, wie der Zugangsschlüssel, der Server und ein eventuell abweichender Server für den Download der Cover gesetzt werden können. Zusätzlich können an dieser Stelle die Cover heruntergeladen oder aktualisiert werden (siehe Abbildung 8).

Auf den nächsten Seiten sollen über Checkboxes die anzuzeigenden Detailinformation für Filme und Episoden sowie die anzuzeigen Filter für die Filmliste angepasst werden können (siehe Abbildung 9).

## 4.3. Datenmanagement Client

### a) Erstellung der URL

Beim Abruf von Informationen aus der Datenbank muss eine URL erzeugt werden, um bei deren anschließenden Aufruf die gewünschten Daten von der API als HTTP-Response zu erhalten. Diese URLs sollen von selbst zu implementieren *QueryFactory*-Klassen erstellt werden. Diesen Klassen müssen die Struktur der Datenbank und die zulässigen Parameter bekannt sein. Beim erstellen einer URL mithilfe dieser Fabriken können dann Syntaxfehler und falsche URLs vermieden werden.

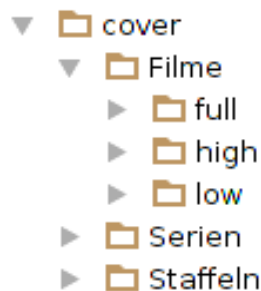
### b) Parsen der JSON-Strings

Wurde eine URL aufgerufen, kann der HTTP-Response der JSON-String mit allen gewünschten Informationen entnommen werden. Dieser String muss geparkt und in Objekten gespeichert werden. Zum Speichern der Daten sollen *HashMaps* mit Schlüssel-Wert-Paaren genutzt werden.

Anschließend kann dann über den Schlüssel auf die Werte des Media-Objekts zugegriffen werden. Da z.B. für die Filmliste mehrere Media-Objekte auf einmal abgefragt werden können, wird für jedes Objekt eine eigene *HashMap* angelegt. Diese *HashMaps* werden am Ende in einer *ArrayList* gespeichert.

Zum eigentlichen Parsen des JSON-Strings muss eine Klasse implementiert werden, die die Struktur der API-Ausgaben kennt und die Daten in die zuvor beschriebene Struktur umwandeln kann.

### c) Download der Cover



Damit Cover angezeigt werden können, muss zuerst ein initialer Download durchgeführt werden. Hierfür wird ebenfalls eine Klasse implementiert, die die Ordnerstruktur auf dem Server kennt und anschließend alle Cover zu den in der Datenbank existierenden Filmen, Serien und Staffeln herunterlädt und lokal speichert. Die Cover werden alle als JPEG Dateien mit dem jeweiligen Primärschlüssel als Dateinamen abgelegt. Sie sind nach dem Media-Typ in Ordner aufgeteilt und liegen in verschiedene Qualitätsstufen vor (siehe Abbildung 10).

Abbildung 10: Cover

## 4.4. Umsetzung der API

Für die API wurde in PHP eine eigene Klasse erstellt. Sie soll sämtliche Funktionen für das Zusammenstellen der Daten bereitstellen. Dabei muss sich der Client durch einen 10-stelligen Schlüssel authentifizieren. Zu jedem Key sollen in einer Sqlite-Datenbank die Anmeldedaten und Zugriffsrechte für die MySQL-Datenbank gespeichert werden. Damit ist es möglich auf verschiedene Datenstände (die in verschiedenen Datenbanken bzw. auf verschiedenen Servern liegen) zuzugreifen. Zusätzlich können Zugangsschlüssel ausgestellt werden, welche keine Rechte für den Zugriff auf die Statistiken bieten.

Nachdem der Client sich durch einen gültigen Schlüssel authentifiziert hat, gibt es die Möglichkeit über sogenannte GET-Parameter Daten anzufordern. Über den Parameter „action“ wird die gewünschte Funktion übergeben, mit weiteren Parametern können z.B. die Daten selektiert, gefiltert oder sortiert werden. Danach werden die Daten im JSON-Format ausgegeben.

In Abbildung 11 ist das Grundgerüst einer

```
{
  "API_VERSION": 0.001,
  "API_KEY": "0123456789",
  "API_Laufzeit": "0.00017",
  "Antwort": {
    [%ACTION% - Abhängige Antwort]
  }
}
```

Abbildung 11: Grundgerüst der JSON Ausgabe

solchen Ausgabe zu sehen. Dieses wird durch die Funktion `json_encode` generiert. Sie konvertiert das temporär erzeugte Array in das JSON-Format.

Sollen Änderungen an der Datenbank vorgenommen werden, werden diese über sogenannte POST-Paramter übergeben und von PHP in eine SQL-Anfrage umgewandelt.

Alle bereits beschriebenen Anfragen an den Datenbankserver werden über die von PHP bereitgestellte Klasse `mysqli` durchgeführt, diese Klasse gibt u.a. die Daten einer MySQL-Anfrage als Array zurück.

Wird eine fehlerhafte Anfrage gestellt oder tritt ein Fehler während der Verarbeitung auf, so gibt es eine JSON-Antwort mit einer Beschreibung des Fehlers.

## 5. Abkürzungsverzeichnis

| Abkürzung | Bedeutung                         |
|-----------|-----------------------------------|
| API       | Application Programming Interface |
| CPU       | Central Processing Unit           |
| Dyn-DNS   | Dynamic Domain Name System        |
| GPL       | General Public License            |
| HTTP      | Hypertext Transfer Protocol       |
| JSON      | JavaScript Object Notation        |
| PHP       | PHP Hypertext Processor           |
| RAM       | Random Access Memory              |