
*Bachelorthesis*

# Konzeption und Realisierung einer Android App zur Erweiterung von Mach-Mit.TV

von

**Hendrik Kegel**

vorgelegt dem
Fachbereich Mathematik, Naturwissenschaften und Informatik
an der
Technischen Hochschule Mittelhessen

zur Erlangung des akademischen Grades
*Bachelor of Science (B. Sc.)*

September 2019

Referent: Prof. Dr. Frank Kammer
Korreferent: Prof. Dr. Achim Kaufmann

# Eidesstattliche Erklärung

Hiermit versichere ich, die vorliegende Arbeit selbstständig und unter ausschließlicher Verwendung der angegebenen Literatur und Hilfsmittel erstellt zu haben.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Gießen, den 23. September 2019

Hendrik Kegel

I would like to dedicate this thesis to my loving parents ...

# Abstract


## Deutsche Version

**Kontext** Mach-Mit.TV ist ein Anbieter für lokales Webfernsehen, der es Gemeinden erlaubt, eine Digitalisierung des Dorflebens zu verwirklichen. Die Plattform bietet ihren Nutzern die Möglichkeit, ihr eigenes TV-Programm zu gestalten. Hierzu laden die Nutzer einzelne Videos hoch, welche durch einen Redaktionsrat überprüft und bei Eignung ins Programm aufgenommen werden. Das Programm wird daraufhin als zusammenhängender Stream zur Verfügung gestellt. Hierzu bietet Mach-Mit.TV bisher eine Wordpress Seite mit eingebettetem Videoplayer, sowie einen redaktionellen Bereich. Diese Arbeit befasst sich mit der Entwicklung einer Android App zur Erweiterung dieser Plattform.

**Konzept** Diese Anwendung wird so entwickelt, dass sie sowohl auf Smartphones als auch auf Smart-TVs eingesetzt werden kann. Auf dem TV wird sie zur reinen Wiedergabe des Streams dienen, während auf dem Smartphone weitere Funktionen zur Verfügung stehen werden. Hierzu gehört ein QR-Code-Scanner, der das Scannen von an Sehenswürdigkeiten platzierten QR-Codes und damit verknüpften Videos ermöglicht. Im Anschluss wird ein Uploadfeature integriert, mit dessen Hilfe die Nutzer eigene Videos zum Projekt beitragen können.

**Ergebnis** Die umgesetzten Funktionen finden Anklang beim Nutzer und die App kann als sinnvolle Erweiterung der Plattform betrachtet werden. Das Durchführen einer Evaluation zeigte dennoch Schwächen auf. Der Dateiupload ist zwar funktional, doch wünschen sich die Nutzer mehr Kontrolle über die beigetragenen Inhalte. Es fehlt eine Möglichkeit, hochgeladene Dateien wieder zu entfernen, sowie eine Statusmitteilung über deren Verwendung. Zur Marktreife fehlt außerdem eine Mediathek. Final ist festzustellen, dass die entwickelte App eine solide Basis schafft und durch die nötigen Erweiterungen einen erheblichen Mehrwert für Mach-Mit.TV erzielen wird.

# Abstract


## English Version

**Context** Mach-Mit.TV is a provider of local web television that allows communities to digitize village life. The platform offers its users the opportunity to design their own TV program. For this purpose, users upload individual videos, which are reviewed by an editorial board and included in the program if they are suitable. The program is then made available as a contiguous stream. For this purpose, Mach-Mit.TV offers a Wordpress site with embedded video player, as well as an editorial area. This thesis deals with the development of an Android app to extend this platform.

**Concept** An app will be developed that can be used on both smartphones and smart TVs. On the TV, the application will be used to watch the stream only, while further features will be available on the smartphone. This includes a QR-Code-Scanner that allows you to scan codes placed on landmarks and stream an associated video. Afterwards, an upload feature will be integrated, with the help of which users can contribute their own videos to the project.

**Result** The implemented functions appeal to the user and the app can be regarded as a meaningful extension of the platform. Nevertheless, conducting an evaluation revealed weaknesses. The file upload is functional, but users want more control over the contributed content. There is no way to remove uploaded files, and no status message about their use. To market readiness it is also missing a media library. Finally, it can be stated that the developed app creates a solid base for further development and therefore will achieve a considerable add in value for Mach-Mit.TV.

# Inhaltsverzeichnis

# Kapitel 1

# Einleitung

Das folgende Kapitel erklärt dem Leser was Mach-Mit.TV ist und warum es um eine mobile Anwendung ergänzt werden sollte. Des Weiteren wird definiert, welche Funktionen die App vereinen soll und wo sie in der Landschaft heutiger Anwendungen einzuordnen ist.

## 1.1 Mach-Mit.TV

Das Projekt Mach-Mit.TV der Technischen Hochschule Mittelhessen befasst sich mit der Realisierung eines lokalen Webfernsehen. Ziel dessen ist es, Gemeinden und Städten eine Möglichkeit zu bieten, das Dorfleben zu digitalisieren, ohne auf Internet-Großkonzerne aus dem Ausland angewiesen zu sein. Hiermit bleibt die lokale Infrastruktur bestehen und Daten werden nach geltendem EU-Recht behandelt.

Mit der Digitalisierung möchte man erreichen, dass besonders die junge Generation wieder mehr ins Ortsleben einbezogen und dauerhaft daran beteiligt wird. Zusätzlich wird ein Kanal für Werbung geschaffen, der den ortsansässigen Geschäften helfen soll, sich gegen die immer mehr dominierende Konkurrenz der Onlinehändler wie Amazon und Zalando zu behaupten [30].


Abbildung 1.1 Mach-Mit.TV: Logo

Mach-Mit.TV sticht hierbei aus anderen Anbietern lokaler Webfernsehen heraus. Die Gemeinden Offenbach, Altötting oder Mühldorf haben mit ihren jeweiligen eigenen WebTV Plattformen [vgl. 16, 17, 1] gezeigt, dass die größte Schwierigkeit, die permanente Lieferung von neuem interessanten Content ist. Die Generierung dessen wenigen Personen aus der Gemeindeverwaltung, der Lokalpresse, lokalen Vereinen o.ä. zu überlassen, hat sich als nicht ausreichend herausgestellt, um tagesaktuelle Nachrichten zu senden. Mach-Mit.TV ermöglicht daher den Nutzern, eigene Videos hochzuladen und damit selbst Teil des Programms zu werden.

Derzeit besteht ein Kooperationsvertrag mit der Gemeinde Biebertal, die damit die Subdomain biebertal.mach-mit.tv erhält. Zukünftige Kooperationen sind angepeilt und bieten den Gemeinden eine bestehende Infrastruktur. Mach-Mit.TV stellt hierbei neben einer Wordpress Seite ein Interface zur Verwaltung der Videos und einen Streaming-Server. Partner können sich demnach vollends auf die Generierung von neuem Content konzentrieren.

## 1.2 Motivation

Im Jahr 2018 verfügen bereits 77,9% der Haushalte in Deutschland über ein Smartphone. Über 85% besitzen einen Flachbild-TV [26]. Die Schaffung einer mobilen Anwendung zielt auf diesen Markt ab und verspricht damit das Erreichen eines deutlich größeren Publikums.

Die ältere Generation, die im Umgang mit Computern oftmals weniger gute Kenntnisse oder geringes Interesse vorweist, hat derzeit keine Möglichkeit zu partizipieren. Eine TV-App, die, installiert durch ein Familienmitglied, mit wenigen Klicks zu öffnen ist und direkt zum Live-Stream der Plattform führt ermöglicht ihnen die Teilnahme am digitalisierten Dorfleben. Zusätzlich besteht die Möglichkeit die App für 24-Stunden-Streams einzusetzen, wie sie oftmals in Apotheken, Krankenhäusern oder in Rathäusern vorzufinden sind.

Für die jüngere Generation, die auch als Content-Creator in Frage kommt, ist das Problem ein Anderes: Zeit. Neben der Arbeit, dem Besuch einer Universität oder der Schule jene für ein solches Projekt aufzubringen erfordert Überzeugung gegenüber diesem. Besonders in der Anfangszeit einer solchen Plattform ist eben diese nicht vorhanden und muss nach und nach aufgebaut werden. Ein komplexer Vorgang zum Beitrag von Content ist hierbei kontraproduktiv. Hier kommt die Smartphone App ins Spiel. Sie ermöglicht, Videos direkt nach der Aufnahme mit wenigen Klicks beizusteuern, statt diese mühselig auf den Computer zu transferieren und über die Online Plattform hochzuladen. Zusätzlich erlaubt die Mobilität

der Geräte die Nutzung von weniger wertvoller Zeit, wie beispielsweise dem Warten auf Züge, Behandlung beim Arzt oder ähnlichem.

## 1.3 Zielsetzung

Das Ziel dieser Arbeit ist es, die folgenden Funktionen übersichtlich und intuitiv bedienbar in einer Anwendung zu vereinen.

1. Stream
   (a) Wiedergabe von Videos aus einer Playlist auf Smartphones und Fernsehern unter Berücksichtigung der zur Verfügung stehenden Bandbreite des Nutzers
   (b) Adäquate Navigationsmöglichkeiten zur Steuerung des Streams
   (c) Auswahl verschiedener Video- und Tonspuren sowie Untertitel falls vorhanden
   (d) Prebuffering des nächsten Videos in der Playlist zur Gewährleistung eines ununterbrochenen Streams
   (e) Dynamische Aktualisierung der Playlist ohne Einfluss auf die aktuelle Wiedergabe
2. Ortsbezogene Wiedergabe
   (a) QR-Codes für direkten Aufruf von Videos
   (b) QR-Code-Scanner, der nur auf QR-Codes von Mach-Mit.TV reagiert
   (c) Wiederverwendung des Players aus (1) zum Abspielen des Videos
3. Dateiuploads
   (a) Uploadsystem für eigene Videos
   (b) Wiederaufnehmbare Downloads - Pausieren und Fortsetzen anhand von Netzwerkbedingungen (z.B. nur im WLAN) oder Nutzerauswahl
   (c) Parallele Uploads
   (d) Oberfläche mit Fortschrittsanzeige
   (e) Notifcations zur Verfolgung von Uploads

## 1.4 Einordnung

Die hier entwickelte App enthält Komponenten, die in ähnlicher Weise in den Mediathek-Apps der öffentlich-rechtlichen TV Sender zu finden sind. Hierzu zählt vor allem die Wiedergabe eines vordefinierten Programms als zusammenhängender Stream. Die Apps *ARD Mediathek* oder *ZDFmediathek & Live TV* bieten zudem eine Mediathek, die den Nutzern ermöglicht einzelne Videos gezielt auszuwählen. Dieses Feature haben sie der Mach-Mit.TV App voraus, was sie aber nicht bieten, ist die Wiedergabe von einzelnen Videos via QR-Code sowie die Möglichkeit zur Partizipation am Programm durch Dateiuploads. Zum Scannen von QR-Codes existieren viele Apps auf dem Markt, bekannte sind beispielsweise *QR Droid Code Scanner* oder *Blitz QR Scanner*. Üblicherweise umfassen sie allerdings keinen Videoplayer und dienen lediglich dem Aufruf von Websites. Das Feature für Dateiuploads und die damit verbundene Idee findet sich in ähnlicher Weise in der App *YouTube* wieder.

Die Plattform Mach-Mit.TV lässt sich ebenfalls mit YouTube vergleichen. Beide bieten ihren Nutzern die Möglichkeit, Videos zu veröffentlichen. Im Gegensatz zu YouTube zielt Mach-Mit.TV allerdings weniger auf die Unterhaltung seiner Nutzer ab. Die Verteilung von Nachrichten steht im Vordergrund. Wenngleich eine Realisierung über YouTube möglich wäre, so steht dies im Widerspruch zur von Mach-Mit.TV gewünschten Datensicherheit.

# Kapitel 2

# Grundlagen

Das folgende Kapitel vermittelt grundlegende Informationen zu Android, der Entwicklung von Anwendungen für Android und Streaming. Weniger erfahrenen Lesern gewährt es einen Einstieg in die Materie, während erfahrene Leser es überspringen können, ohne befürchten zu müssen, spätere Entscheidungen nicht nachvollziehen zu können.

## 2.1 Android

Android ist ein auf einem modifizierten Linux Kernel basierendes Betriebssystem, dass vor allem auf mobilen Geräten wie Smartphones, Tablets und Smartwatches zum Einsatz kommt. Doch immer mehr Geräte setzen auf das unter der Apache License laufende Betriebssystem. Mittlerweile kommt es auf Fernsehern mit Android TV, in Fahrzeugen mit Android Auto und auf Uhren mit Wear OS zum Einsatz.

Entwickelt wurde Android ursprünglich von der von Andy Rubin 2003 gegründeten Firma Android Inc., die das Betriebssystem zur Steuerung von Digitalkameras verwenden wollte [20]. Im Sommer 2005 kaufte Google das Unternehmen [6]. Zwei Jahre später gründete Google die Open Handset Alliance zusammen mit 33 weiteren Partnern. Dieser Unternehmenszusammenschluss von heute 84 rechtlich und wirtschaftlich selbständigen Unternehmen sollte von nun an die Entwicklung von Android fortführen [19].

Mit dem HTC Dream, auch bekannt als T-Mobile G1, kam am 22. Oktober 2008 dann das erste auf dem Android-Betriebssystem basierende Gerät auf den Markt. Seit diesem Zeitpunkt erbrachte Android Google einen Gesamtumsatz von 31 Mrd. US-Dollar (Stand: Januar 2016) [24].

### 2.1.1 Architektur

Android basiert auf einem **Linux Kernel**. Dieser bietet bereits grundlegende Sicherheitsfeatures und erlaubt es Herstellern, Treiber für einen ihnen bereits bekannten Kernel zu entwickeln. Des Weiteren verwaltet er Threading Operationen sowie die low-level Speicherverwaltung.

Über dem Kernel findet sich die **Hardware Abstraction Layer (HAL)**. Diese besteht aus verschiedenen Modulen, die je ein Interface pro Hardware Komponente implementieren. Diese Interfaces abstrahieren nun die konkrete Hardware und bieten dem Java API Framework eine Schnittstelle zu dieser.

Auf der nächsten Ebene befinden sich **native C/C++ Libraries**. Einige Kernkomponenten Androids basieren auf nativem Code, welcher weitere native Bibliotheken verwendet. Bei der Appentwicklung kann über das Android NDK auf einige dieser nativen Bibliotheken zugriffen werden.

Seit Android 5.0 erhält jede App ihren eigenen Prozess mit einer eigenen Instanz der **Android Runtime (ART)**. Diese findet sich auf der Ebene der nativen Libraries und dient dem Ausführen mehrerer virtueller Maschinen auf Geräten mit niedrigem Speicher. Vor Android 5.0 war Dalvik die Android Runtime. Zusätzlich ist eine Sammlung von Laufzeitbibliotheken vorhanden, welche einige Features der Programmiersprache Java zur Verfügung stellt.

Diese werden vom **Java API Framework** verwendet, um die Grundlage zur Entwicklung von eigenen Apps zu schaffen. Das Framework bietet die Schnittstelle zu allen von Android zur Verfügung gestellten Features und liefert dem Entwickler alle Bausteine, die er benötigt.

Auf der obersten Ebene stehen die **System Apps**. Android bietet auch hier einige Standardanwendungen für Telefonie, SMS-Versand, Browsing, Kamera und vieles mehr. Die vorinstallierten Apps haben keine Sonderstellung gegenüber Drittanbieterapps, so dass der Nutzer die freie Wahl hat, sein System zu gestalten.

Die folgende Abbildung zeigt eine Übersicht der Architektur.

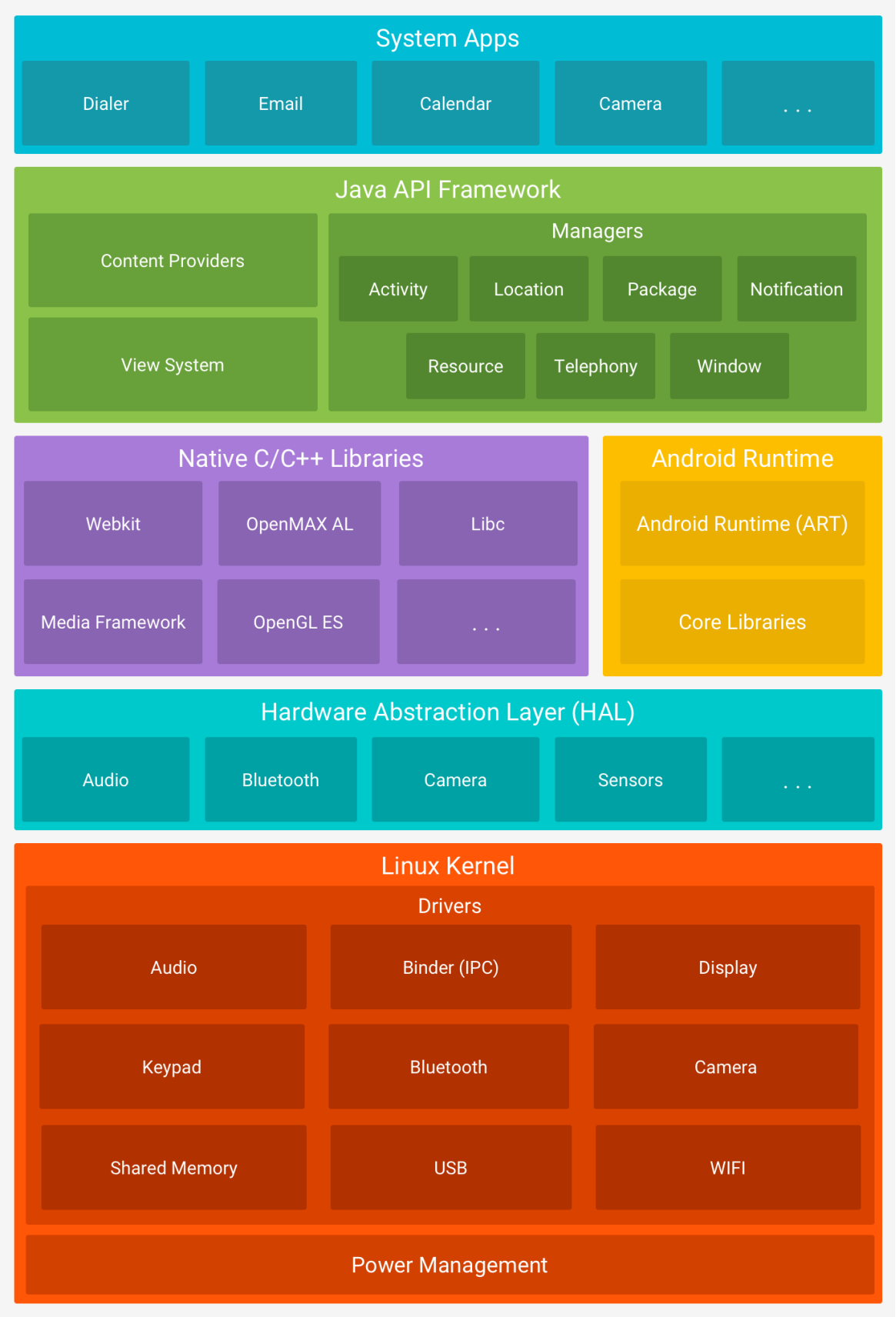


Abbildung 2.1 Android Architektur [8]

### 2.1.2 Plattform Versionen

Mit Android 10 veröffentlichte Google am 03. September 2019 die neueste Version seines Betriebssystems und verabschiedet sich gleichzeitig von den vorher verwendeten Codenamen [15]. Android 10 ist damit das erste Android ohne Namenszusatz. Für die Entwicklung ist dieser natürlich weniger relevant als das mit der Version verknüpfte API Level. Mit dem Festlegen des Min API Level bestimmt der Entwickler bis zu welcher Android Version seine App kompatibel ist. Geräte mit einem niedrigeren API Level werden demnach nicht unterstützt. Da neue Versionen auch neue Features mitbringen, sollte der Entwickler das API Level überlegt wählen. Ein niedrigeres API Level führt zu einer höheren Abdeckung der auf dem Markt vorhandenen Geräte, erschwert aber möglicherweise die Entwicklung. Ein Beispiel hierfür ist die Einführung von neuen Java 8 Sprachfunktionen ab API 24, womit beispielsweise die Nutzung von Javas Streams ermöglicht wird. Auch Libraries werden gegen ein API-Level entwickelt, was bei deren Nutzung zu beachten ist. Die eigene Min-API kann hierbei nicht niedriger sein, als die der Library mit der höchsten Min-API.

Tabelle 2.1 Überblick bisheriger Android Versionen [9]

| Version | Codename | Release | API |
|---|---|---|---|
| 1.0 | „Base“ | September 2008 | 1 |
| 1.1 | „Base_1.1“ | Februar 2009 | 2 |
| 1.5 | „Cupcake“ | April 2009 | 3 |
| 1.6 | „Donut“ | September 2009 | 4 |
| 2.0.x / 2.1 | „Éclair“ | Oktober 2009 | 5, 6, 7 |
| 2.2.x | „Froyo“ (Frozen Yogurt) | Mai 2010 | 8 |
| 2.3.x | „Gingerbread“ | Dezember 2010 | 9, 10 |
| 3.x.x | „Honeycomb“ | Februar 2011 | 11, 12, 13 |
| 4.0.x | „Ice Cream Sandwich“ | Oktober 2011 | 14, 15 |
| 4.1.x / 4.2.x / 4.3.x | „Jelly Bean“ | Juli 2012 | 16, 17, 18 |
| 4.4.x | „Kitkat“ | Oktober 2013 | 19, 20 |
| 5.0.x / 5.1.x | „Lollipop“ | November 2014 | 21, 22 |
| 6.0.x | „Marshmallow“ | Oktober 2015 | 23 |
| 7.0.x / 7.1.x | „Nougat“ | August 2016 | 24, 25 |
| 8.0 / 8.1 | „Oreo“ | August 2017 | 26, 27 |
| 9 | „Pie“ | August 2018 | 28 |
| 10 | | September 2019 | 29 |

### 2.1.3 Das Manifest

Die Grundlage jedes Android Projektes ist das Manifest. In jeder App muss daher die Datei *AndroidManifest.xml* im Root Ordner der Sourcefiles liegen. Darin finden sich essentielle Information für die Android Build Tools, das Betriebssystem und Google Play.

Das Manifest legt unter anderem folgende Dinge fest:

- Paketname
- Permissions
- Hardware & Softwarefeatures
- Komponenten
  - Activities
  - Services
  - Broadcast Receiver
  - Content Provider
- Eigenschaften dieser Komponenten

### 2.1.4 Aufbau von einfachen Anwendungen

Eine einfache Anwendung besteht primär aus **Activities** [3]. Eine Activity ist ein Fenster, mit dem der Nutzer interagieren kann. Sie besteht aus einer Java Klasse, beispielsweise *MainActivity.java*, und einer zugehörigen XML Datei, hier *activity_main.xml*. In der XML Datei wird das Layout der Oberfläche beschrieben. Der Entwickler kann hier verschiedene Elemente wie Buttons, Textfelder, Bilder und viele weitere Elemente anordnen. Die Java Klasse ergänzt das Layout um Funktionalität, die zur Laufzeit verwendet wird. Eine nutzbare App entsteht durch das Verknüpfen mehrerer Activities. Hierbei kann zu jedem Zeitpunkt nur exakt eine Activity aktiv sein. Zum Aufrufen einer neuen Activity wird ein **Intent** verwendet. Ein Intent ist eine abstrakte Beschreibung einer durchzuführenden Aktion und kann zum Beispiel dazu verwendet werden, Daten aus aufrufenden an aufgerufene Activities zu übergeben [10]. Damit bei einem Wechsel der Activity der Speicher sinnvoll freigegeben werden kann, durchlaufen alle Activities einen eigenen fest definierten Lebenszyklus, auf den im Folgenden Abschnitt näher eingegangen wird.

### 2.1.5 Activity Lifecycle

Die geöffneten Activities werden von einem Activitystack verwaltet. Wird eine neue Activity gestartet, wird diese auf den Stack gelegt und zur neuen *running* Activity. Die vorherige Activity bleibt im Stack darunter vorhanden bis die aktuelle geschlossen wird. Jede Activity erhält hierbei einen Lifecyclestate. Folgende Zustände sind möglich:

1. **Active** or **Running** Die derzeit oberste Activity auf dem Stack. Sie liegt im Vordergrund und ist üblicherweise die Activity, mit der der Nutzer interagiert.

2. **Visible** Als „sichtbar“ eingestuft sind solche Activities, die nicht fokussiert aber dennoch sichtbar sind. Dies kann zum Beispiel dann auftreten, wenn die Activity von einem Dialog nur teilweise überdeckt wird. Die Activity selbst bleibt hierbei vollständig vorhanden, d.h. sie behält alle Informationen bezüglich ihres Zustands und ihrer Elemente und bleibt am Window Manager angebracht.

3. **Stopped** or **Hidden** Die Activity ist vollständig verdeckt und so für den Nutzer nicht mehr sichtbar. Sie bleibt vorerst vollständig vorhanden, doch wird für das System als *killable* eingestuft. Damit kann sie im Falle von Speicherknappheit zerstört werden. Hierbei gehen sämtliche Informationen verloren.

4. **Destroyed** Eine solche zerstörte Activity kann nicht wiederhergestellt und muss entsprechend neu gestartet werden.

Der Verlust von Informationen durch Zerstören von Activities kann vom Entwickler verhindert werden, in dem er die Informationen vor dem Zerstören adäquat persistiert.

Die folgende Grafik zeigt den Lebenszyklus einer Activity sowie Callback Methoden, die beim Durchleben dieser aufgerufen und vom Entwickler implementiert werden können.

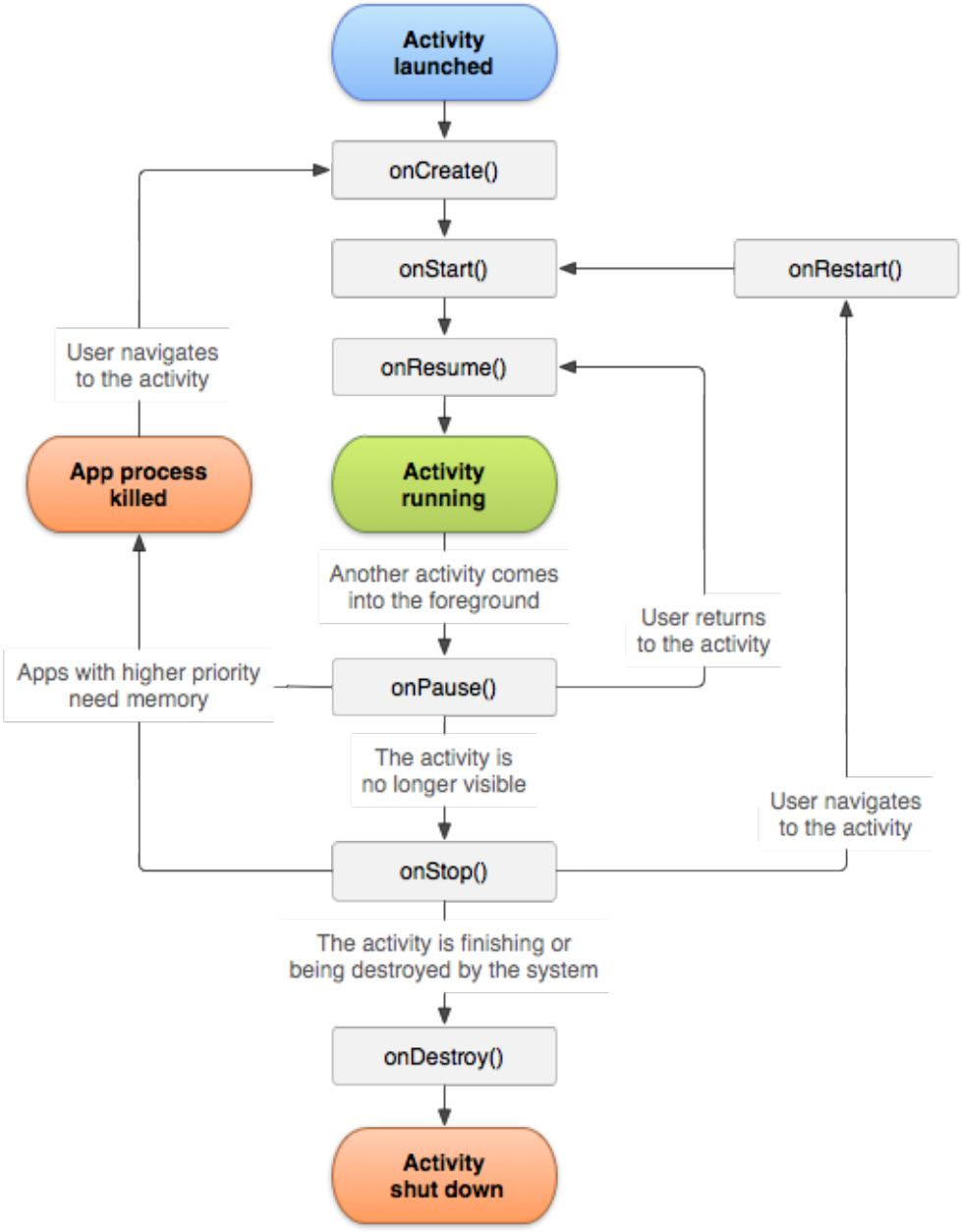


Abbildung 2.2 Android Lifecycle [11]

### 2.1.6 Aufbau von komplexen Anwendungen

Komplexere Anwendungen sollten nach dem **Model-View-Controller Pattern (MVC)** aufgebaut werden. Dies ermöglicht eine bessere Trennung zwischen Logik, Oberfläche und Datenstrukturen womit der Entwicklungsprozess erleichtert wird [18]. Der Quellcode wird hierbei auf die folgenden logischen Komponenten verteilt.

- Das **Model** umfasst Datenstrukturen und die Verwaltung von Datenbankoperationen.
- Der **View** umfasst die Oberflächen zur Dartstellung der im Model deklarierten Daten.
- Der **Controller** ergänzt die Logik zur Aktualisierung der Daten.

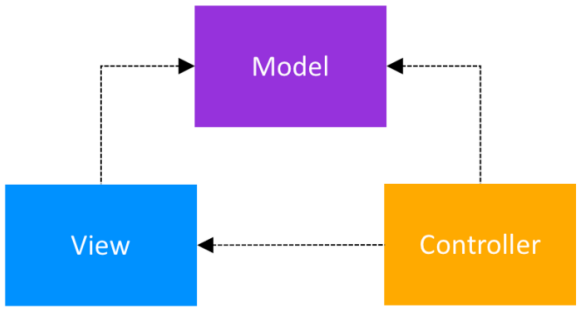


Abbildung 2.3 MVC [18]

Für das MVC-Pattern existieren eine **aktive** und eine **passive** Umsetzung des Models. Bei der passiven Realisierung werden alle Aktionen vom Controller durchgeführt. Nach der Aktualisierung der Daten im Model ist er weitergehend dafür zuständig, dem *View* mitzuteilen, dass sich die Daten geändert haben und eine Erneuerung der Oberfläche notwendig ist. Die aktive Realisierung bedient sich dem Observer-Pattern. Hierzu wird im *Model* ein Observer deklariert, der auf Veränderung der Daten reagiert. Der zugehörige View implementiert dann das Observer Interface und registriert sich damit als Observer. Bei Veränderung der Daten im Model werden dann alle registrierten Observer über die Änderung informiert und können entsprechend mit der Aktualisierung der Oberfläche reagieren.

## 2.2 HTTP Live Streaming (HLS)

HLS ist ein auf HTTP basierendes Protokoll für adaptives Streaming, das von Apple 2009 initial veröffentlicht wurde [4]. Dank des zu Grunde liegenden HTTP-Protokolls können Streams über konventionelle HTTP-Server angeboten und auch über jegliche Firewalls oder Proxy Server, die Standard HTTP-Traffic erlauben, abgerufen werden.

Die Adaptivität des Streams wird dadurch ermöglicht, dass dessen Eingabevideo in Segmente (chunks) geteilt und dem Nutzer mittels einer auf M3U basierenden Playlist zur Verfügung gestellt wird [14]. Hierin können für den gleichen Inputstream verschiedene Enkodierungen vermerkt sein, wodurch es dem Client ermöglicht wird, anhand seiner derzeitig verfügbaren Bandbreite die Qualität des Streams anzupassen. Des Weiteren kann die Master-Playlist mehrere Varianten des gleichen Streams beinhalten, so dass die Segmente gleichzeitig von verschiedenen Servern angeboten werden. Ist einer der Server temporär nicht verfügbar, kann der entsprechende Part von einem anderen geladen werden. Hierdurch kann eine ständige Verfügbarkeit der Inhalte gewährleistet werden.

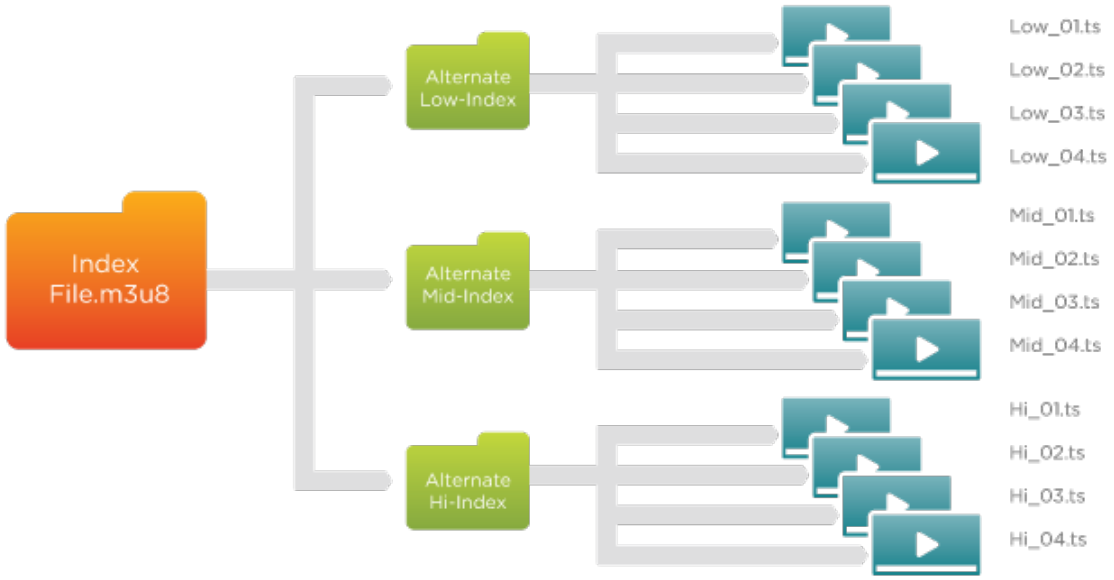


Abbildung 2.4 Segmentierung unter HLS [34]

### 2.2.1 Architektur

Um erfolgreich ein Video mit HLS zu verbreiten, bedarf es der folgenden Infrastruktur:

1. Ein **Web Server** enkodiert und segmentiert das wiederzugebene Video und erzeugt die Indexierung mittels .m3u8-Playlist.
2. Ein **Datenhost** stellt alle erzeugten Dateien zur Verfügung. In kleineren Projekten ist eine Realisierung im lokalen Speicher des Webservers denkbar. Bei Großprojekten kommt ein separater Webserver zum Einsatz, der alle für das Streaming nötigen Ressourcen via HTTP zugänglich macht.
3. Der **Client** lädt zuerst die Playlistdatei und dann die verfügbaren Mediensegmente. Eine hier installierte Software setzt diese dann zusammen und präsentiert sie dem Nutzer als zusammenhängendes Video.

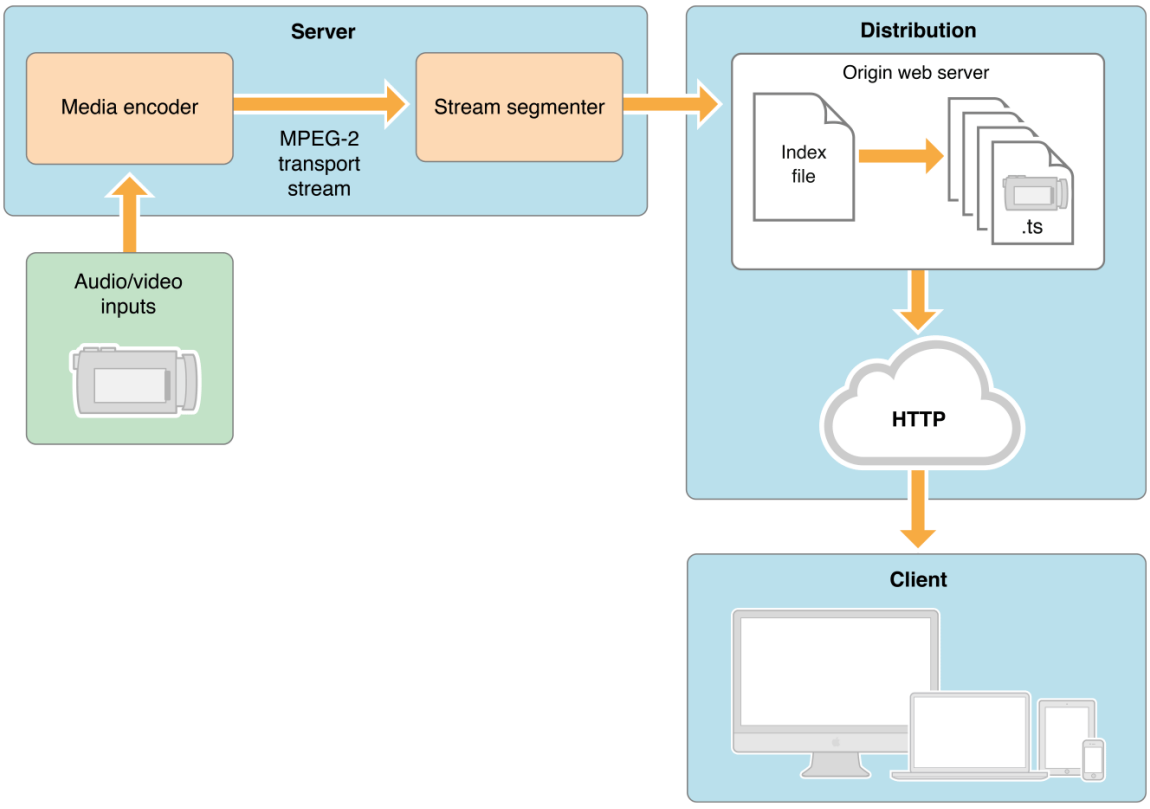


Abbildung 2.5 HLS: Vom Video zum Stream [2]

### 2.2.2 Dateiformate

Der Encoder kodiert das Video zu H.264 [2] sowie Audiospuren zu AAC, MP3, AC-3 oder Enhanced AC-3 [21] und erzeugt eine für MPEG-2 Transport Streams transportierbare Datei [22]. Diese wird vom Segmenter in einzelne MPEG-2 Fragmente mit gleicher Länge geteilt, welche im .ts Format gespeichert werden. Dazu erzeugt er die zugehörige Playlistdatei, die auf die jeweiligen Fragmente verweist. Diese wird im Format .m3u8 gespeichert.

# Kapitel 3

# Enwtwurfsentscheidungen

## 3.1 Plattform

Eine Entwicklung der App für mehrere Plattformen kam aus Zeitgründen nicht in Frage, daher wurde sich für Android entschieden. Diese Entscheidung basiert maßgeblich auf dem derzeitigen Marktanteil der Plattform. Mit einem Anteil von über 75% ist es das derzeit klar dominante Betriebssystem für Smartphones. Der Hauptkonkurrent Apple erreicht hingegen mit iOS nur etwas mehr als 22%. Andere Betriebssysteme sind mit unter 2% vollständig zu vernachlässigen.

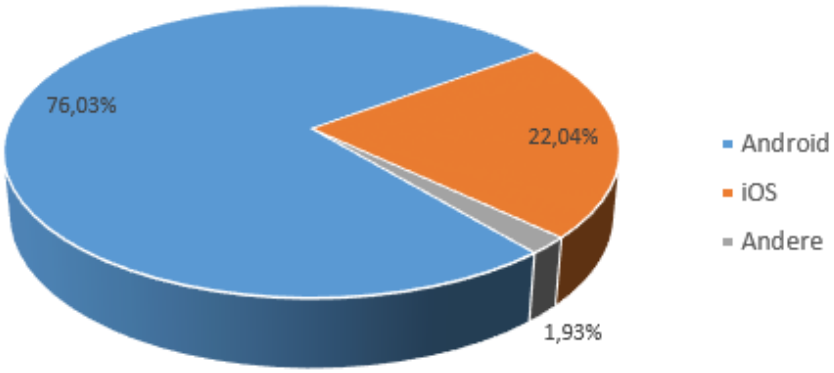


Abbildung 3.1 Marktanteile Smartphones [25]

Des weiteren bietet Android den Vorteil, als Betriebssystem für Smart-TVs einen Fuß am Markt gefasst zu haben. Die App kann also ohne großen Mehraufwand in der Entwicklung für die Nutzung auf einem TV verwendet werden. Die Marktanteile bei Fernsehern sind allerdings deutlich verstreuter als die bei Smartphones. Mit 10% kann sich Android bereits als Big-Player im Vergleich der Betriebssysteme bezeichnen, wobei sich diese Zahl nur auf die offiziell von Google unterstützten Geräte mit Zugang zum Play Store bezieht. Weitere androidbasierte Betriebssysteme ohne Zugang zum Playstore, wie sie beispielsweise auf einigen TV-Boxen zu finden sind, sind möglicherweise in der Lage, die App via Sideload zu installieren, tauchen aber nicht in dieser Statistik auf. Der Marktführer ist Samsung, der mit Tizen auf ca. 21% Marktanteile kommt. Eine Entwicklung hierfür würde aber mit einem deutlich höheren Aufwand in Verbindung stehen, da hierzu eine vollständig neue App entwickelt werden müsste.

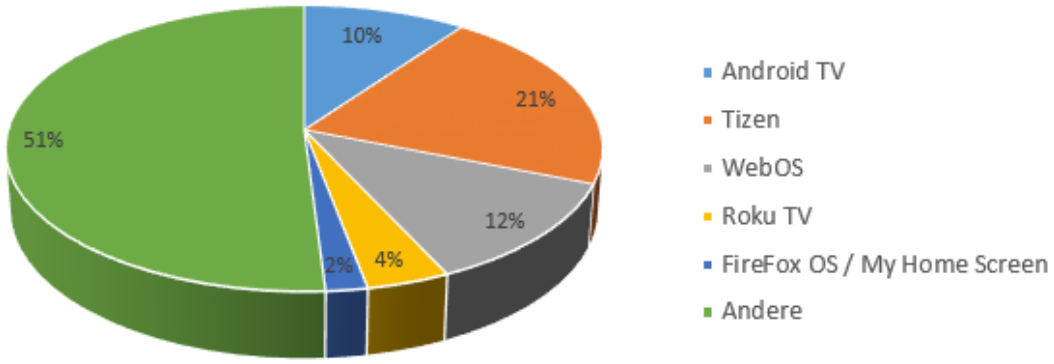


Abbildung 3.2 Marktanteile von TV-OS [32]

## 3.2 Plattform Version

Bei der Plattform Version wurde sich für Android 5 bzw. Api-Level 21 entschieden. Diese Entscheidung basiert maßgeblich auf dem Marktanteil der jeweiligen Version. Hierzu ist anzumerken, dass die Versionen untereinander vorwärts- aber nicht rückwartskompatibel sind. Zur Visualisierung wird die Tabelle 2.1 um den Marktanteil der jeweiligen Version [9] ergänzt.

Tabelle 3.1 Android Versionen und ihre Marktanteile

| Version | Codename | Release | API | Marktanteil |
|---|---|---|---|---|
| 1.0 | „Base“ | September 2008 | 1 | - |
| 1.1 | „Base_1.1“ | Februar 2009 | 2 | - |
| 1.5 | „Cupcake“ | April 2009 | 3 | - |
| 1.6 | „Donut“ | September 2009 | 4 | - |
| 2.0.x / 2.1 | „Éclair“ | Oktober 2009 | 5, 6, 7 | - |
| 2.2.x | „Froyo“ (Frozen Yogurt) | Mai 2010 | 8 | - |
| 2.3.x | „Gingerbread“ | Dezember 2010 | 9, 10 | 0,3% |
| 3.x.x | „Honeycomb“ | Februar 2011 | 11, 12, 13 | - |
| 4.0.x | „Ice Cream Sandwich“ | Oktober 2011 | 14, 15 | 0,3% |
| 4.1.x / 4.2.x / 4.3.x | „Jelly Bean“ | Juli 2012 | 16, 17, 18 | 3,2% |
| 4.4.x | „Kitkat“ | Oktober 2013 | 19, 20 | 6,9% |
| 5.0.x / 5.1.x | „Lollipop“ | November 2014 | 21, 22 | 14,5% |
| 6.0.x | „Marshmallow“ | Oktober 2015 | 23 | 16,9% |
| 7.0.x / 7.1.x | „Nougat“ | August 2016 | 24, 25 | 19,2% |
| 8.0 / 8.1 | „Oreo“ | August 2017 | 26, 27 | 28,3% |
| 9 | „Pie“ | August 2018 | 28 | 10,4% |
| 10 | | September 2019 | 29 | - |

* Marktanteile geringer als 0,1% wurden vernachlässigt.

Mit Api-Level 21 werden also 89,3% der im Google Playstore registrierten Geräte abgedeckt. Anhand der reinen Marktanteile stellt sich die Frage, warum das Api Level nicht auf 19 festgelegt wurde, obwohl dies eine um weitere 6,9% höhere Abdeckung zur Folge hätte. Die Antwort darauf ist in erster Linie Bequemlichkeit. Mit Api 21 wurden diverse neue Features hinzugefügt, die die Entwicklung deutlich vereinfachen. Weitere Informationen diesbezüglich bietet die offizielle Dokumentation [vgl. 7].

## 3.3 Streaming

Die Plattform Mach-Mit.TV, die den Stream bereits über ihre Webpräsenz zur Verfügung stellt, verwendet diesbezüglich HLS. HLS ist neben MPEG-DASH das am weitesten verbreitete Streaming Protokoll für Streaming via HTTP. Genauere Informationen zur Funktionsweise finden sich im Grundlagenkapitel.

Zur Wiedergabe des Streams wird Appseitig der ExoPlayer von Google verwendet. Dieser wurde ursprünglich als Player für Googles eigene Anwendungen wie YouTube und Google Play Movies entwickelt [33] und ist im Gegensatz zu dem in Android integrierten MediaPlayer einfach anzupassen und zu erweitern. Zudem wird er stetig weiterentwickelt [vgl. 12].

## 3.4 Upload Protokoll

Der Videoupload soll primär wiederaufnehmbar sein. Der Nutzer soll in der Lage sein, seine Uploads manuell oder automatisch anhand der aktuellen Netzwerkverbindung zu pausieren und zu einem späteren Zeitpunkt fortzusetzen. Zur Realisierung wird das TuS Protokoll verwendet. TuS basiert und erweitert das beliebte HTTP-Protokoll und ist damit leicht in vorhandene Anwendungen zu integrieren. Entwickelt von Transloadit Ltd steht es öffentlich zugänglich (open source) auf Github zur Verfügung [vgl. 29]. Im Gegensatz zu JavaScript Libraries wie *Resumable.js* oder eigenen Umsetzungen bietet TuS neben einer fertigen serverseitigen auch clientseitige Implementierungen. Hierunter finden sich neben dem hier relevanten Android-Client ebenfalls ein JavaScript, Java und iOS-Client. Durch die vielseitigen Implementierungsmöglichkeiten bietet TuS Mach-Mit.TV viele Erweiterungsmöglichkeiten für die Zukunft. Durch den vorhandenen Android-Client lässt es sich des Weiteren einfach in die App integrieren und ist damit die passende Wahl für die hiesigen Zwecke.

# Kapitel 4

# Bedienung & Implementierung

Das folgende Kapitel beschreibt das Umfeld in dem entwickelt wurde, sowie die Bedienung und technische Zusammenhänge, die in Relation zu dieser stehen. Hierzu werden die einzelnen Funktionen der App zuerst aus Nutzersicht vorgestellt und im Folgenden durch technische Informationen ergänzt. Es empfiehlt sich, den sich im Anhang befindlichen Quellcode der App parallel vor Augen zu halten, da des öfteren auf Stellen in diesem verwiesen wird. Vollständiger Quellcode wird aus Platzgründen nicht eingebunden. Gezeigter Code ist teilweise verändert und/oder gekürzt, um Zusammenhänge ersichtlich zu machen. Es gelten folgende Anmerkungen.

Listing 4.1 Formatierung von Quellcode

```
1  // Kontext
2  Klasse::Methode
3
4      // Annahme: vorher sichergestellt
5      assert x = 0;
6      assert z = 2;
7
8      // Nicht weiter erläuterte Methoden folgen in ihrer Funktion ihrem Namen
9      doNothingWith(x,y,z);
10
11     // Relevante Informationen werden mit einfachem Kommentar ergänzt
12     maybeeDoSomethingWith(x, /*doNothing:*/ true);
13
14     // Resultat: Nachvollziehbarer Code ohne vollständige Methoden
15     if (x == z) error();
```

## 4.1 Umgebung

Die App wird in Android Studio entwickelt. Die IDE basiert auf der bekannten Plattform IntelliJ IDEA und bietet damit eine für die meisten Entwickler vertraute Oberfläche. Android Studio wird von Google entwickelt und verspricht damit die beste Kompatibilität für die Entwicklung.

Zur Versionierung wird Github in Kombination mit GitKraken verwendet. Warum Git zur Versionierung eingesetzt werden sollte und welche Vorteile der Einsatz bringt wird hier nicht weiter erläutert. Der interessierte Leser kann sich diesbezüglich der Literatur [23, 31] bedienen. GitKraken ist eine grafische Oberfläche zur Vereinfachung des Workflows und ist nicht zwingend nötig, erleichtert aber die Arbeit.

Zum Testen wird bevorzugt der Android Studio interne Emulator verwendet. Dieser bietet die einfachste Möglichkeit für schnelles Debugging und kann jede relevante Androidversion emulieren. Für weitere Tests kommt ein HTC One M8S mit Android 8.1 und ein Sony Bravia Smart TV mit Android 8 zum Einsatz. Die verwendeten Geräte wurden hierfür nicht speziell ausgewählt sondern entspringen dem Bestand. Es empfielt sich, neben dem Emulator auch auf realen Geräten zu testen. Besonders optische Aspekte stehen hier in einer besseren Relation zum Gerät. Auch technische Probleme können beispielsweise bei Nutzung der Kamera oder einer Fernbedienung in der virtualisierten Umgebung schnell untergehen.

Abbildung 4.1 Logos der verwendeten Tools [13]

## 4.2 Tests & Fehlerbehandlung

Android bietet die Möglichkeit zur Verwendung der aus Java bekannten jUnit-Tests. Diese dienen weitgehend der Sicherstellung, dass Methoden stets das vorhergesehene Ergebnis liefern. Aufgrund der Arbeit mit flexiblen Daten entstünde allerdings ein relativ hoher Aufwand, qualitativ hochwertige Tests in jUnit zu schreiben, weshalb sie in dieser Arbeit nicht verwendet werden. Auch wenn hier keine derartigen Tests eingesetzt werden empfiehlt sich die Verwendung dieser. Für weitere Informationen zu jUnit kann auf die Literatur [27] zurückgegriffen werden.

Die hier entwickelte Anwendung verwendet die androideigene *Log* Klasse. Diese bietet mit den Methoden *Log.v()*, *Log.d()*, *Log.i()*, *Log.w()* und *Log.e()* die Möglichkeit, Einträge in Logcat zu schreiben. Die einzelnen Buchstaben stehen hierbei für die jeweilige Priorität des Eintrages: **V** steht für Verbose (niedrigste Priorität), **D** für Debug, **I** für Info, **W** für Warning und **E** für Error. Logcat liefert dann mittels eigenen Fensters in Android Studio Informationen zur laufenden Anwendung. Es bietet des weiteren Filtermöglichkeiten, so dass Fehler gezielt gesucht und behoben werden können.

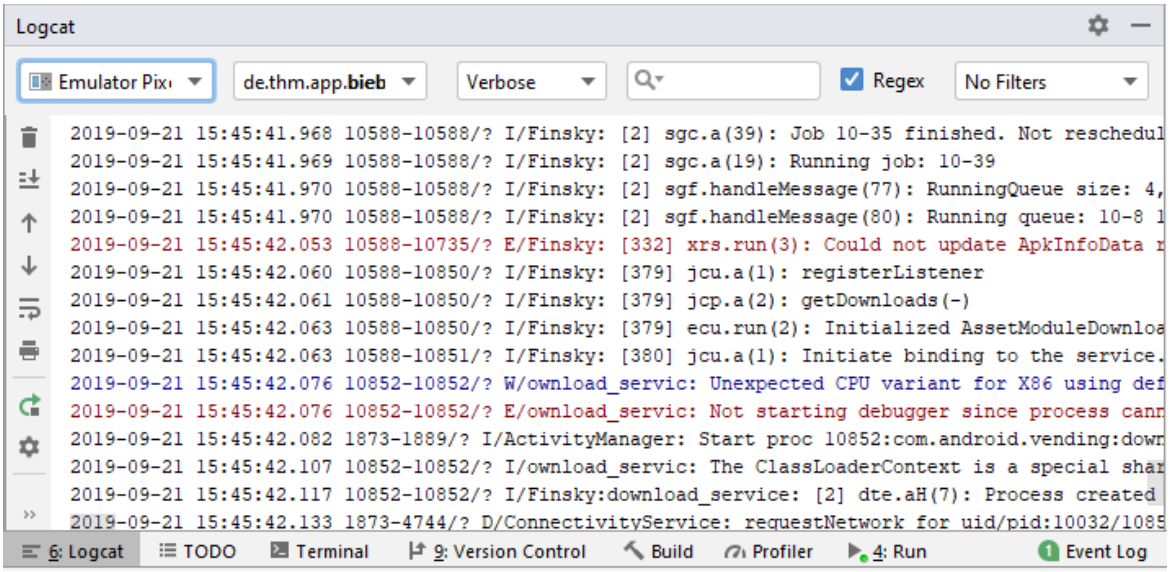


Abbildung 4.2 Ausschnitt aus Logcat

Neben Logcat bietet Android Studio eine Speicheranalyse. In Kombination mit einem Dauertest der Anwendung kann diese auf etwaige Memory Leaks hinweisen. Da die Anwendung auch auf Fernsehern verwendet wird und anzunehmen ist, dass der Stream auf unbegrenzte Zeit gelooped wiedergegeben wird, ist die Durchführung eines solchen Tests von besonderer Relevanz. Die Anwendung wurde hierzu simultan auf zwei Emulatoren und einem realen TV einem mehrstündigen Test unterzogen. Wie sich herausstellen sollte, war dies auch notwendig. Der Test offenbarte ein Memory Leak bei Server Requests, welcher zu einem Absturz der Anwendung führte. Dieser trat hierbei etwa drei Stunden nach Beginn des Tests auf und konnte durch Analyse des Stacks lokalisiert werden. Durch Betrachtung der Allokationen in diesem zeigte sich, dass Instanzen der Klasse RequestQueue aus der Volley Library nicht korrekt vom Garbage Kollektor entfernt werden konnten. Da selbige bei jedem neuen Serverrequest angelegt wurden, lief der Arbeitsspeicher voll. Gelöst wurde das Problem mittels einer globalen RequestQueue, welche vor und nach jedem Request geleert wird. Nach der Eliminierung des Memory Leaks wurde die App einem weiteren 24 stündigen Dauertest unterzogen. Dieser zeigte keine weiteren Probleme.

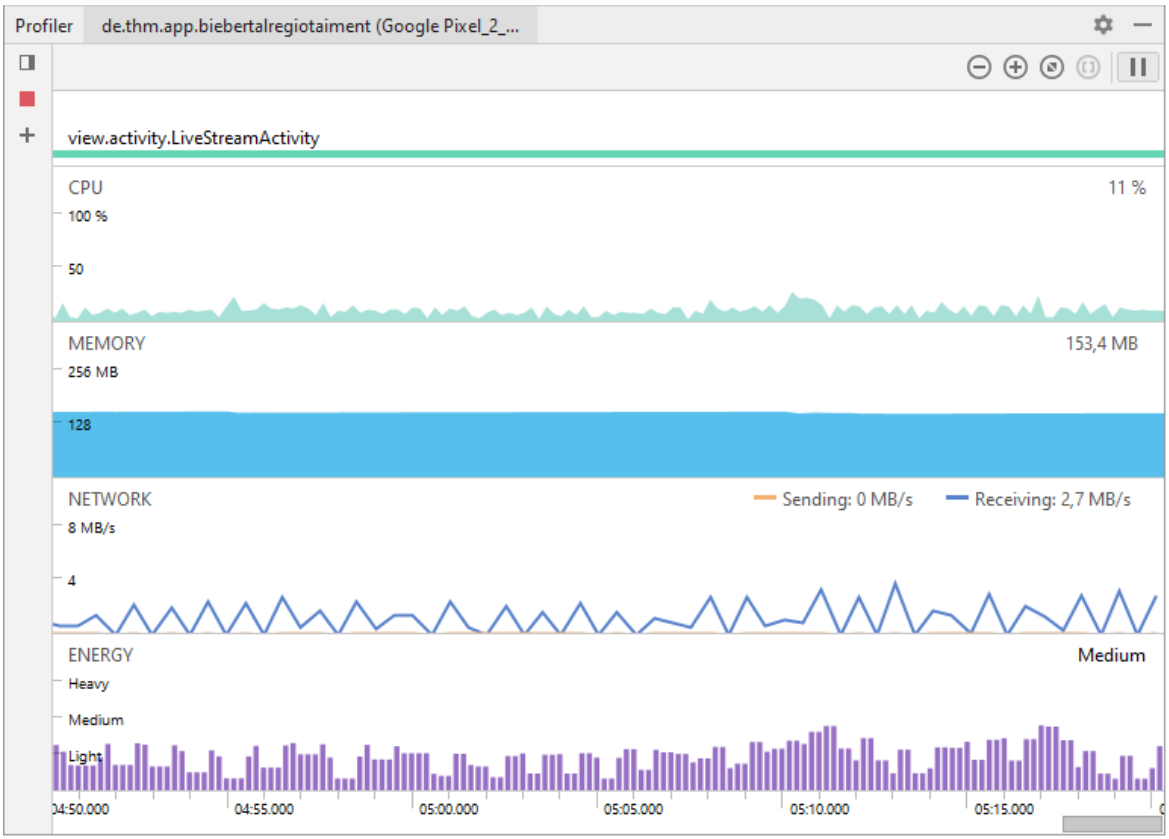


Abbildung 4.3 Android Studio Profiler

## 4.3 Startscreen

### Bedienung

Die App führt den Nutzer beim Start auf die in Abbildung 4.4 zu sehende Oberfläche. Diese wird im Folgenden als Startscreen bezeichnet. Von hier aus ist jede Funktion der App mittels Klick auf die Selbige zugänglich.


Abbildung 4.4 Der Startbereich der Anwendung

### Implementierung

Im Code findet sich der Startscreen als *SimpleMainActivity*. Das dazugehörige Layout ist in *activity_simple_main.xml* definiert und ordnet die zu sehenden Elemente auf dem Bildschirm an. Hierbei wird ein verschachteltes LinearLayout verwendet. Die darin enthaltenen LinearLayouts vereinen jeweils einen Text mit dem zugehörigen Icon und benennen eine onClick Methode, welche in der Activity Klasse umgesetzt wird.

Um einen Wechsel des Standartscreens zu ermöglichen, wird die Activity *MainActivity* erstellt. Diese wird beim Appstart geöffnet. Sie selbst hat keine Oberfläche und dient nur dem Start der als Startscreen festgelegten Acitvity. Im Anschluss beendet sie sich selbst.

## 4.4 Streaming

### Bedienung

Der Stream wird über den Unterpunkt *Stream* auf dem Startscreen geöffnet und startet umgehend im Vollbild-Modus. Durch einen Klick auf den Bildschirm wird der Controller angezeigt. Dieser beinhaltet die für einen Videoplayer typischen Interaktionsmöglichkeiten: Vor- und Zurückspulen, Pausieren oder Fortsetzen und Video zurücksetzen und überspringen. Zusätzlich kann über das Menü rechts oben auf zusätzliche Einstellungen zugegriffen werden. Hier können Videoeinstellungen vom Nutzer getroffen werden. Neben der Videoqualität können sowohl die Audiospur als auch Untertitel verwaltet werden. Die Abbildungen 4.5, 4.6, 4.7 und 4.8 bebildern das Vorgehen.


Abbildung 4.5 Der Stream


Abbildung 4.6 Der Controller des Streams

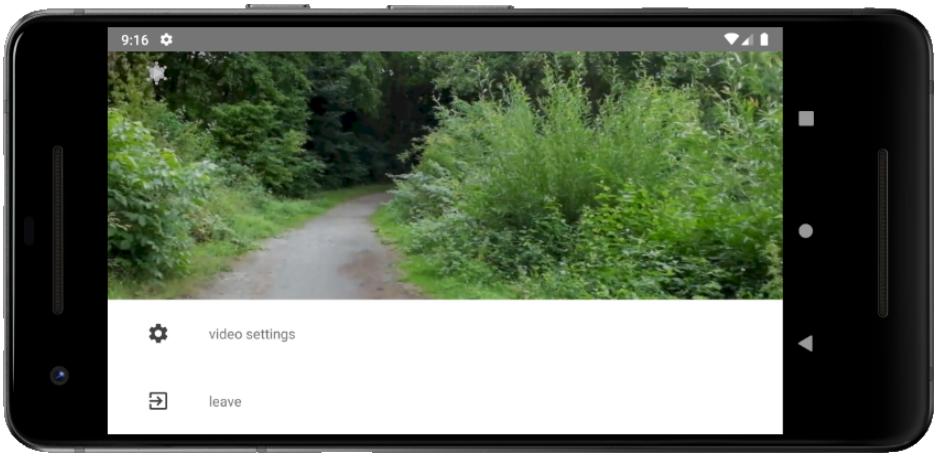


Abbildung 4.7 Erweiterte Einstellungen

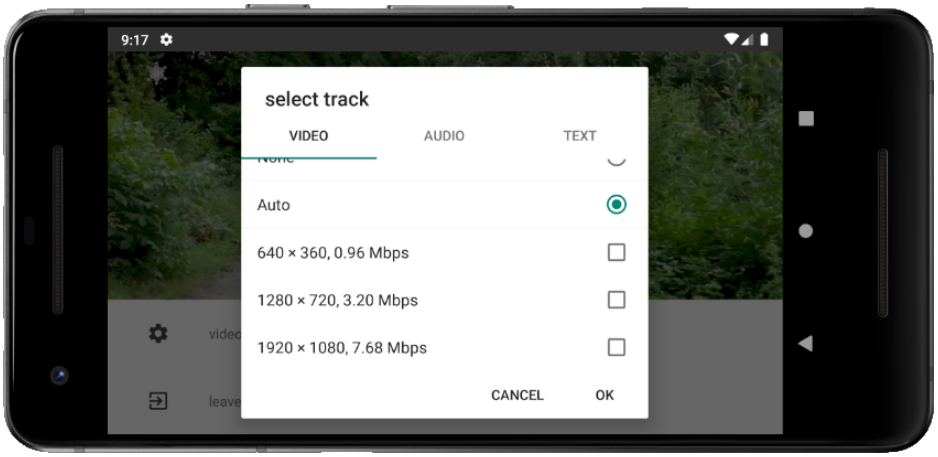


Abbildung 4.8 Video-Einstellungen

## Implementierung

### PlaybackManager - die Logik

Die Klasse *PlaybackManager* implementiert die Logik zur Wiedergabe von Videos. Ihre Kernaufgaben sind die Initialisierung des Players, die Verwaltung der lokalen Playlist, das Behandeln von Fehlern sowie das Informieren der zugehörigen Oberfläche bei Änderungen der aktuellen Playlistposition. Für den Player wird eine Instanz des ExoPlayers mit einem *BandwidthMeter* und einem adaptiven *TrackSelector* initialisiert. Mittels des Abschätzens der zukünftigen Downloadgeschwindigkeit wird ermöglicht, eine variable Streamingqualität zu wählen, sofern der Stream diese Möglichkeit bietet. Zusätzlich werden die Lifecyclehooks des *PlaybackManager* an den Player gebunden und weitere Videoeinstellungen vorgenommen. Die zur Initialisierung verwendeten Klassen stammen ebenfalls aus der Library des ExoPlayers.

Listing 4.2 Initialisierung des Players

```
1  PlaybackManager::
2      PlaybackManager(Context context,
3                      PlayerView playerView,
4                      QueuePositionListener queuePositionListener,
5                      List<Film> playlist)
6
7      defaultBandwidthMeter = new DefaultBandwidthMeter.Builder(context).build();
8      defaultTrackSelector = new DefaultTrackSelector(adaptiveTrackSelectionFactory);
9
10     exoPlayer = ExoPlayerFactory.newSimpleInstance(
11             context,
12             new DefaultRenderersFactory(context),
13             defaultTrackSelector,
14             new DefaultLoadControl(),
15             null,
16             defaultBandwidthMeter);
17
18     exoPlayer.addListener(this);
19     exoPlayer.setRepeatMode(REPEAT_MODE_ALL);
20     exoPlayer.setVideoScalingMode(C.VIDEO_SCALING_MODE_SCALE_TO_FIT_WITH_CROPPING);
```

Die Aktualisierung der lokalen Playlist erfolgt durch Entfernen aller Elemente und Inserieren der neuen Elemente an ihrer jeweiligen Position. Diese entpricht ihrer Position in der Liste aus der sie stammen. Hierdurch wird ermöglicht, das aktuell wiedergegebene Video von der Aktualisierung auszuschließen, wodurch dieses zu Ende gespielt wird. Dieses Vorgehen ist etwas laufzeitintensiver, was aber für den Nutzer keine Auswirkungen hat. Eine Aktualisierung der Playlist findet generell eher selten statt und läuft asynchron zur Wiedergabe. Der Ausschnitt in 4.3 zeigt diese Routine.

Listing 4.3 Aktualisierung der Playlist

```
1  PlaybackManager::insertAll(List<Film> films)
2
3      // Fügt alle Filme in die mediaQueue ein. Der Index des jeweiligen Films
       ↪  entspricht der Position in der Wiedergabeliste, die er einnehmen soll.
4
5      for (int i = 0; i < films.size(); i++) {
6          if (films.get(i).equals(getItem(getCurrentItemIndex()))) continue;
7          insertItem(films.get(i), i);
8      }
9
10 PlaybackManager::updateQueue(List<Film> newMediaQueue)
11     if (mediaQueue == null || newMediaQueue == null || newMediaQueue.isEmpty())
12         return INPUT_ERROR;
13     if (mediaQueue.equals(newMediaQueue)) return NOT_NEEDED;
14
15     clearMediaQueue(/*keepCurrent:*/ true);
16     insertAll(newMediaQueue);
17     return SUCCESS;
```

Auf Fehler bei der Wiedergabe wird mittels Lifecyclehook reagiert. Hierbei wird effektiv nur auf Fehler des Typs *TYPE_SOURCE* reagiert. Dieser tritt auf, wenn ein angefragtes Video nicht verfügbar ist. In diesem Fall wird das Video aus der Playlist entfernt und als Konsequenz vom Player übersprungen. War es das Einzige in der Playlist, so wird der Nutzer über die Nichterreichbarkeit informiert. Sollte ein anderer Fehler auftreten, wird die Playlist automatisch neu eingelesen und der Player aktualisiert.

Listing 4.4 Errorhandling

```
1  PlaybackManager::onPlayerError(ExoPlaybackException error)
2      switch (error.type) {
3          case ExoPlaybackException.TYPE_SOURCE:
4              removeItem(getCurrentItemIndex());
5              if (!mediaQueue.isEmpty()) break;
6
7              exoPlayer.stop(/*reset:*/ false);
8              informUser("Video cannot be played");
9              return;
10         case ExoPlaybackException.TYPE_RENDERER: [...]
11         case ExoPlaybackException.TYPE_UNEXPECTED: [...]
12         case ExoPlaybackException.TYPE_OUT_OF_MEMORY: [...]
13         case ExoPlaybackException.TYPE_REMOTE: [...]
14     }
15     exoPlayer.prepare(concatenatingMediaSource, false, false);
```

### Die Oberfläche

Die abstrakte Klasse *BasePlaybackAcitvity* erweitert den Player um eine einfache Oberfläche. Diese verknüpft die Elemente des Controllers mit Funktionen, bietet die Möglichkeit einen *ProgressDialog* anzuzeigen, verwaltet Touch- und Keyevents und sorgt für das Ausblenden der System-UI zwecks direkter Wiedergabe im Vollbild. Durch die Abstraktion der Klasse wird die Möglichkeit geboten diese um verschiedene Eigenschaften zu ergänzen. Die *SinglePlaybackActivity* und *LiveStreamActivity* bedienen sich der vorhandenen Grundfunktionen und erweitern diese um den jeweiligen Einsatzzweck. Im Fall des Streamings kommt die *LiveStreamAcitvity* zum Einsatz. Sie erweitert den Player um den Servercall zur Aktualisierung der Playlist. Hierbei fragt Sie beim Server in festen Intervallen die aktuelle Playlist an und übergibt sie dem *PlaybackManager*. Dieser überprüft, ob sich die Playlist verändert hat. Bei Veränderung wird die lokale Playlist an die soeben angefragte angepasst, andernfalls geschieht nichts.

## 4.5 QR-Codes Scannen

### Bedienung

Vom Startscreen aus wird die Funktion *Scan* gestartet. Beim ersten Ausführen wird die Berechtigung zur Nutzung der Kamera abgefragt. Diese muss gewährt werden, um das Feature zu nutzen. Hierauf wird der Nutzer im Falle des Ablehnens hingewiesen. Hat der Nutzer akzeptiert, kann er mittels Kamera beginnen, Barcodes zu scannen. Das Scannen von Barcodes ist auf solche limitiert, die auf die Domain von Mach-Mit.TV verweisen. Wird ein valider Barcode gescannt, so startet die Wiedergabe des dazugehörigen Videos. Diese erfolgt im selben Format wie der Stream. Ist das Video zu Ende wird es automatisch geschlossen. Der Nutzer findet sich auf dem Startscreen wieder.

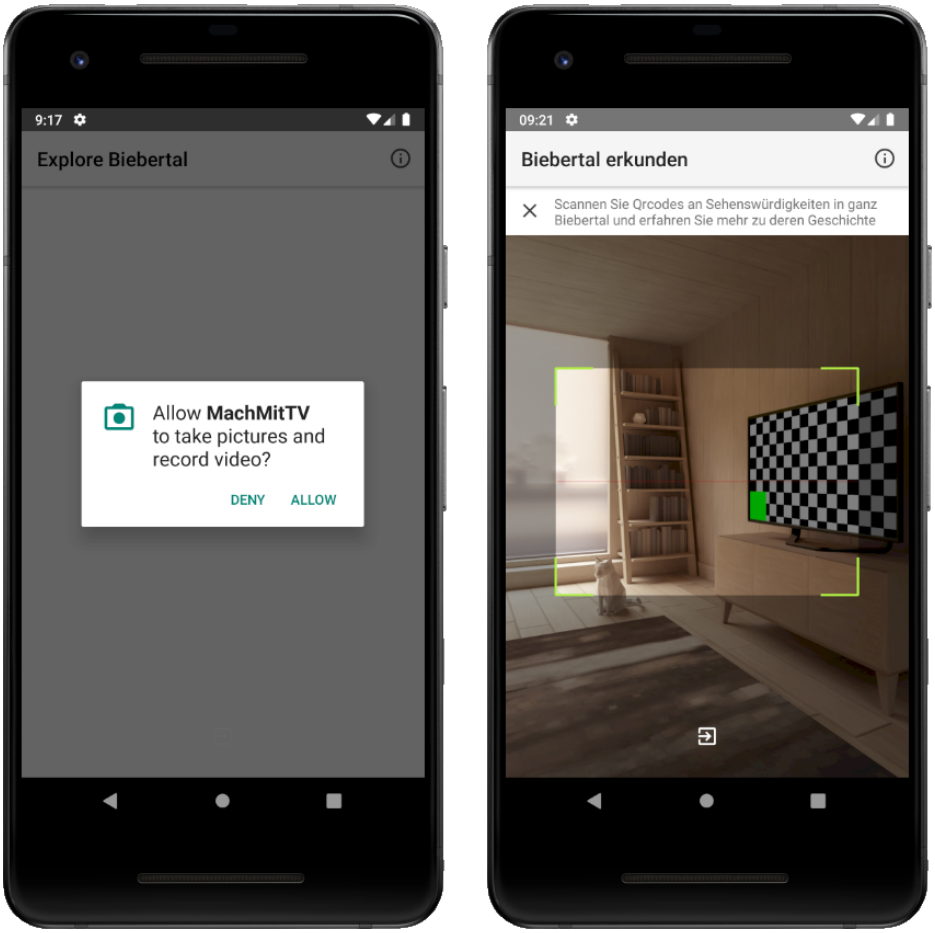


Abbildung 4.9 QR-Code Scanner

## Implementierung

Der QR-Code Scanner basiert auf dem Scanner des Github Nutzers **dm77** [vgl. 5]. Dieser bietet eine vollständige Implementierung, die über ein Contentframe dargestellt wird. Ergänzt wurde der Scanner lediglich um ein Verlassen- und ein Infosymbol. Bei einem validen Scan wird die *SinglePlaybackActivity* gestartet. Ihre Aufgabe ist es, das korrespondierende Video wiederzugeben und sich im Anschluss zu beenden. Dazu wird die *BasePlaybackActivtiy* wie folgt erweitert.

Listing 4.5 SinglePlaybackActivity

```
Anmerkung: Hier wird vollständiger Quellcode gezeigt, um zu veranschaulichen wie
 ↪  simple die Klasse durch das Erweitern der BasePlaybackActivity wird.

public class SinglePlaybackActivity extends BasePlaybackActivity {

    private static final String TAG = SinglePlaybackActivity.class.getName();

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        String route = getIntent().getStringExtra("route");
        playbackManager = new PlaybackManager(this, playerView, this, new
         ↪  Film(route));
        playbackManager.setRepeatMode(Player.REPEAT_MODE_OFF);

        controls.setMobile();
    }

    @Override
    public void onQueuePositionChanged(int previousIndex, int newIndex) {
        Log.i(TAG, "onQueuePositionChanged: " + previousIndex + " -> " + newIndex);
        if (newIndex==PlaybackManager.INDEX_UNSET) {
            playbackManager.release();
            finish();
        }
    }
}
```

## 4.6 Upload

### Bedienung

Wieder wird die Funktion vom Startsceen aus gestartet. Mit einem Klick auf Upload öffnet sich beim ersten Start ein Hinweis mit Informationen zum Inhalt von hochgeladenen Videos. Der Nutzer muss akzeptieren, dass er nur GEMA-freien Kontent hochlädt, der eine Länge von 3 Minuten nicht überschreitet. Wie bereits beim QR-Code ist das Akzeptieren verpflichtend und die Funktion kann nur unter Vorbehalt dessen verwendet werden.

Hat der Nutzer akzeptiert, kann er mit dem (+) unten rechts seinen installierten Dateimanager starten und das entsprechend hochzuladene Video auswählen. Nach der Auswahl öffnet sich eine Form zum Anpassen der Metadaten. Pflichtfelder sind entsprechend markiert, andere sind optional. Die Felder „Ersteller“ und „E-Mail“ werden nach der Eingabe gespeichert und bei zukünftigen Uploads vorausgefüllt. Die Version wird mit 1, der Autor mit dem Namen des Erstellers initialisiert. Alle Werte können vom Nutzer angepasst werden.

Nach dem Ausfüllen der Metadaten wird das Video der Liste hochzuladener Dateien angefügt. Von hier aus kann der Nutzer dieses mit einem Klick hochladen. Der Upload kann an dieser Stelle ebenfalls pausiert oder abgebrochen werden. Dies geschieht durch einen einfachen Klick auf das jeweilige Listenelement. Zum Entfernen eines Uploads aus der Uploadliste wird dessen Eintrag gedrückt gehalten und damit markiert. Jetzt können noch weitere Einträge durch einfaches Anklicken hinzugefügt werden. Alternativ können alle Elemente der Liste durch Klicken des entsprechenden Symbols (oben rechts neben dem Mülleimer) markiert werden. Mit einem Klick auf das Löschen Icon werden die markierten Einträge gelöscht. Es ist anzumerken, dass bereits hochgeladene Inhalte nicht nachträglich gelöscht werden können. Es werden lediglich die lokalen Einträge entfernt.

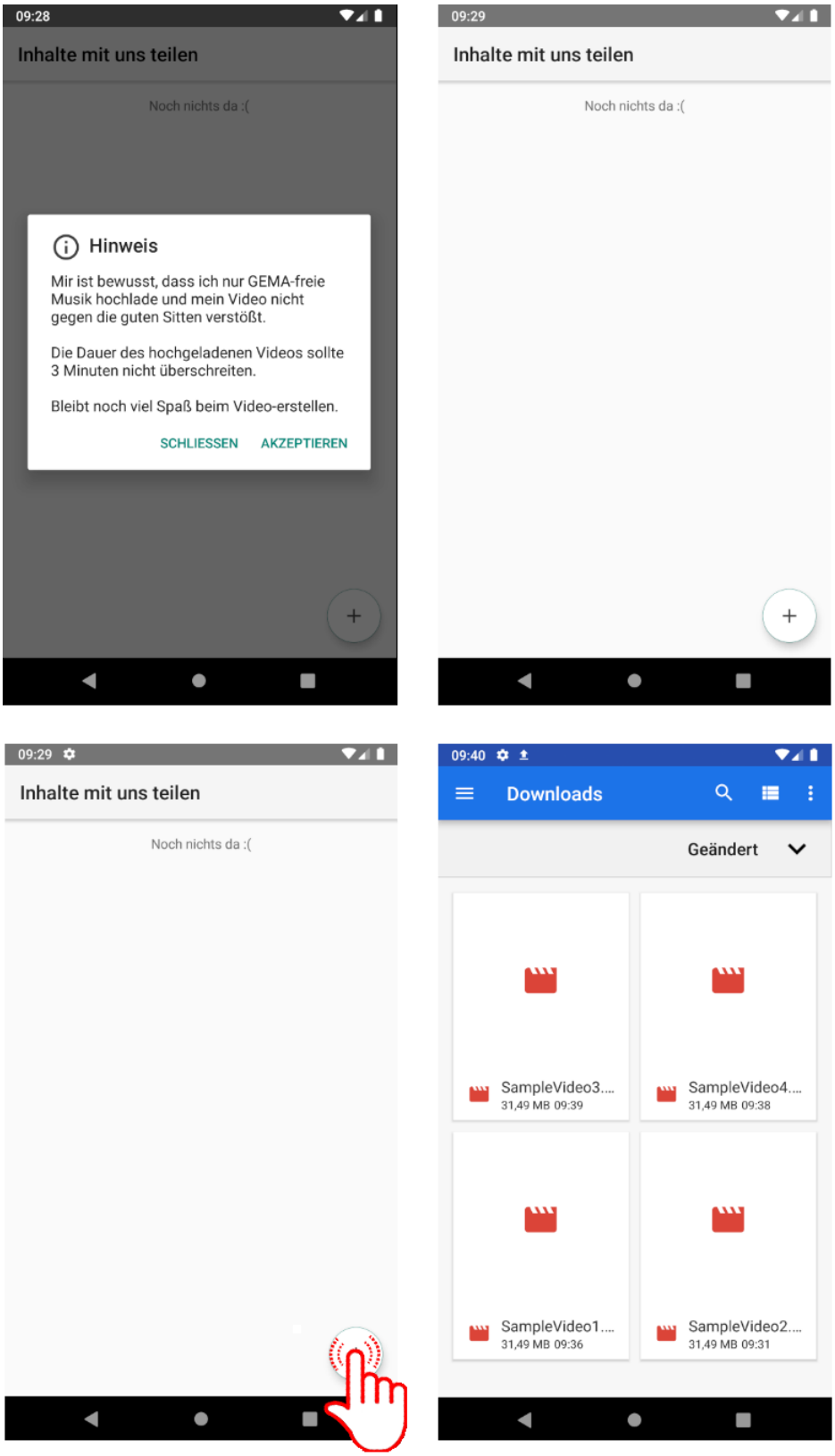


Abbildung 4.10 Disclaimer und Dateiauswahl

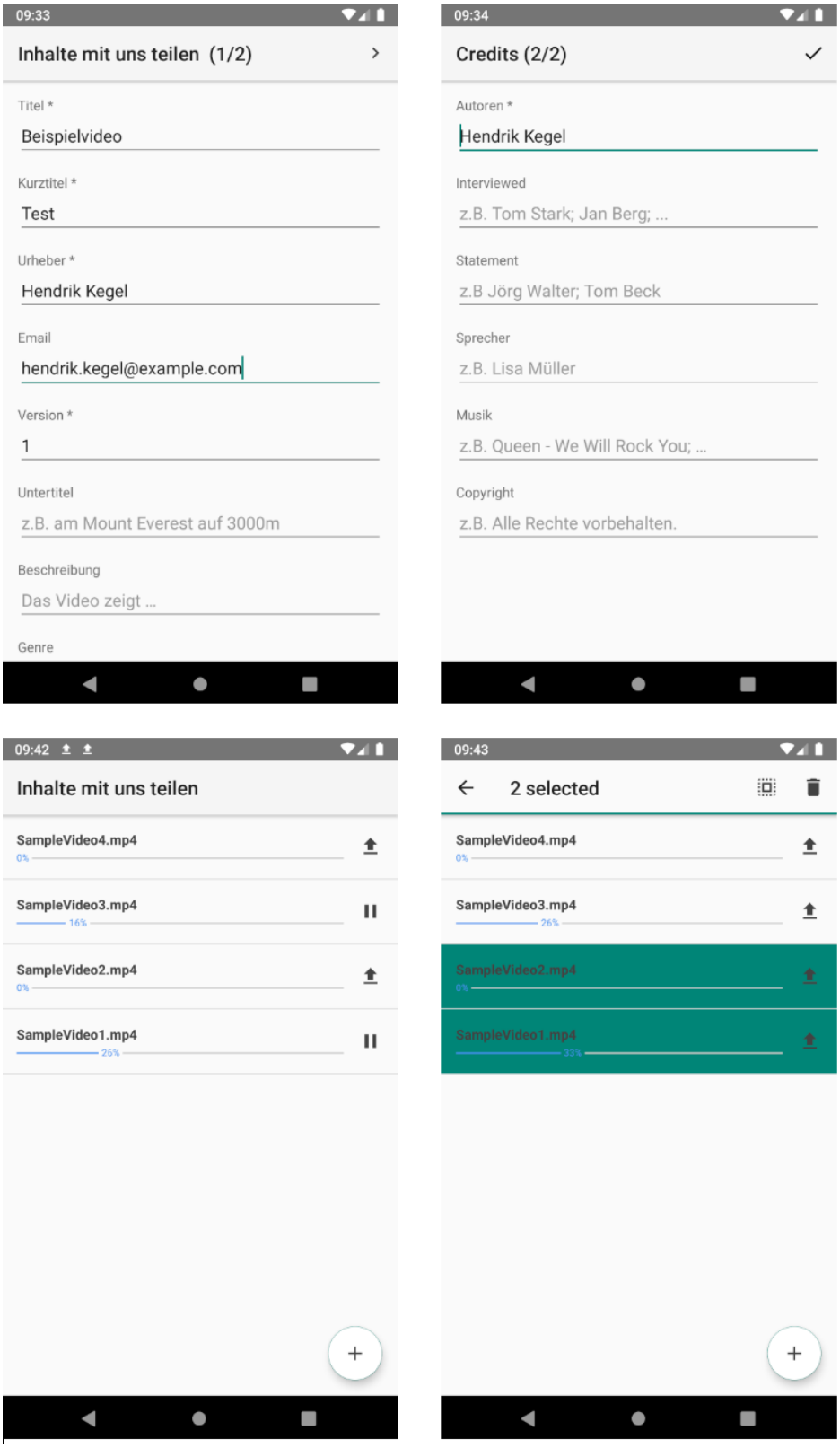


Abbildung 4.11 Metadaten & Uploadverwaltung

## Implementierung

Der Hinweis beim ersten Start des Uploadbereichs ist durch einen einfachen *AlertDialog* realisiert. Akzeptiert der Nutzer die Nutzungsbedingungen, wird ein Flag in den *Preferences* gesetzt und er kann fortfahren. Solange dieses Flag nicht vorhanden ist, wird beim Start der Activity stets der AlertDialog angezeigt, welcher die Selbe im Fall der Ablehnung wieder schließt.

Listing 4.6 Disclaimer zum Dateiupload

```
Anmerkung:
Die Variablen title, message, yes und no wurden zur Veranschaulichung eingefügt und
 ↪  sind im realen Code als Referenzen zu den entsprechenden Strings umgesetzt.

UploadActivity::onCreate
    if (!App.Preferences.get(C.REF_KEY_FIRST_UPLOAD, /*default:*/ false)){

        new AlertDialog.Builder(this)
                .setTitle(title)
                .setMessage(message)
                .setPositiveButton(yes), (dialog, which) ->
                 ↪  App.Preferences.put(C.REF_KEY_FIRST_UPLOAD, true))
                .setNegativeButton(no), (dialog, which) -> finish())
                .setCancelable(false) // do not allow to dismiss the dialog
                .setIcon(R.drawable.ic_info_outline_dark)
                .show();
    }
```

Das Hinzufügen neuer Videos geschieht wie vorher beschrieben über das (+) unten rechts. Diese Art von Button wird als *Floating Action Button* oder kurz *FAB* bezeichnet. Durch das Hinzufügen eines *onClickListeners* erhält dieser seine Funktionalität. Hier wird diesbezüglich ein Intent definiert, der in Kombination mit dem Aufruf *startActivityForResult* das Starten eines Dateimanagers bewirkt. Dieser ermöglicht, über *onActivityResult* auf dessen Antwort zu reagieren. Seine Antwort beinhaltet eine Content-Uri zu der hochzuladenen Datei. Befindet sich die Datei nicht auf der Liste der hochzuladenen Dateien, wird wiederum mittels Intent und *startActivityForResult* die *UploadFormActivity* gestartet. Sie ergänzt die Datei um ihre Metadaten und gibt sie an die *UploadActivity* zurück, welche sie der Liste der hochzuladenen Dateien anfügt.

Listing 4.7 Dateien zum Uploud wählen

```
1  UploadAcitvity::onCreate
2      fab.setOnClickListener(v -> {
3          Intent intent = new Intent(Intent.ACTION_OPEN_DOCUMENT);
4          intent.addCategory(Intent.CATEGORY_OPENABLE);
5          intent.setType("video/*");
6          startActivityForResult(intent, REQUEST_FILE_SELECT);
7      });
8
9  UploadActivity::onActivityResult
10     assert resultCode == RESULT_OK;
11
12     if (requestCode == REQUEST_FILE_SELECT) {
13         Uri uri = data.getData();
14
15         UpFile upFile = new UpFile(getFileName(uri), uri.toString());
16
17         if (uploads.contains(upFile)){
18             Toast.makeText(this, "allready in list", Toast.LENGTH_SHORT).show();
19             return;
20         }
21
22         Intent intent = new Intent(this, UploadFormActivity.class);
23         intent.putExtra("uri", upFile.getContentUri().toString());
24         startActivityForResult(intent, REQUEST_METADATA_SELECT);
25     }
```

Der Upload selbst wird wie beschrieben durch einen Klick auf das jeweilige Listenelement gestartet. Hierbei wird ein neuer *UploadTask* erstellt. Dieser verwendet den offiziellen Android-Client von TuS [vgl. 28] zum asynchronen Upload der Datei. Laufende Tasks werden in einer Map verwaltet, so dass kein Mehrfachstarten des selben Uploads möglich ist.

Listing 4.8 Dateien hochladen

```
1  UploadAcitvity::beginUpload(UpFile upFile)
2      UploadTask task = new UploadTask(upFile, this, upFile.getUniqueId());
3
4      if (!tasks.containsKey(task)){
5          tasks.put(upFile, task);
6          task.executeOnExecutor(AsyncTask.THREAD_POOL_EXECUTOR);
7          return;
8      }
9
10 UploadTask::onPreExecute
11     upload = new TusAndroidUpload(file.getContentUri(), weakActivity.get());
12
13     Map<String, String> metaData = file.getMetaData().get();
14     if (!metaData.isEmpty()) upload.setMetadata(metaData);
15
16
17 UploadTask::doInBackground
18     TusUploader uploader = client.resumeOrCreateUpload(upload);
19
20     long totalBytes = upload.getSize();
21     long uploadedBytes;
22
23     uploader.setChunkSize(1024 * 1024);
24
25     while(!isCancelled() && uploader.uploadChunk() > 0) {
26         uploadedBytes = uploader.getOffset();
27         publishProgress(uploadedBytes, totalBytes);
28     }
29
30     uploader.finish();
31     return uploader.getUploadURL();
```

## 4.7 Einstellungen

### Bedienung

Der Bereich Einstellungen, abermalig erreicht durch Klicken auf *Settings* auf dem Startscreen, enthält diverse Einstellungen, die appweit angewandt werden. Der Nutzer kann hier seine Standartansicht festlegen. Hiermit kann der Startscreen angepasst werden. Zur Auswahl stehen „Einfach“ - Der Standartstartscreen aus Abbildung 4.4, „Web“ - eine komplexerer Startscreen, der die Website von Mach-Mit.TV im Hintergrund anzeigt und „Stream“ - hiermit startet die App direkt in den Livestream des Programms. Des Weiteren kann ein globales Qualitätslimit für den Stream gewählt, sowie Untertitel und Audiodeskription im Voraus aktiviert werden. Der Upload kann darauf beschränkt werden, nur dann Dateien hochzuladen, wenn der Nutzer mit einem WLAN Netzwerk verbunden ist. Im Bereich „Über“ werden Informationen über die App bereitgestellt und dem Nutzer ermöglicht, ein Feedback zur App zu verfassen. Das Feedback wird per E-Mail an den Entwickler gesandt, wobei automatisch Informationen zum Gerät angehängt werden.

### Implementierung

Die *SettingsActivity* verwendet den im Paket Androidx enthaltenen *PreferenceScreen*. Die einzelnen Elemente werden hierzu in der Datei *root_preferences.xml* angegeben und in der *SettingsActivity* über den *FragmentManager* geladen. Vorgenommene Änderungen werden in Folge automatisch in den *Preferences* der App gespeichert. Das Feedback Feature wurde wie folgt realisiert und über das *SettingsFragment* per *onClickListener* mit dem zugehörigen Punkt in den Einstellungen verknüpft.

Listing 4.9 Das Feedback Feature

```
1 App::sendFeedback(Context context) // static
2     String body = [...]; // Erstellen der in Abbildung 4.10 sichtbaren Nachricht
3
4     Intent intent = new Intent(Intent.ACTION_SENDTO, Uri.fromParts(
5             "mailto", C.FEEDBACK_EMAIL_ADDRESS, null));
6     intent.putExtra(Intent.EXTRA_EMAIL, new String[]{C.FEEDBACK_EMAIL_ADDRESS});
7     intent.putExtra(Intent.EXTRA_SUBJECT, C.SUBJECT_EMAIL_ADDRESS);
8     intent.putExtra(Intent.EXTRA_TEXT, body);
9     context.startActivity(Intent.createChooser(intent,
      ↪   context.getString(R.string.choose_email_client)));
```

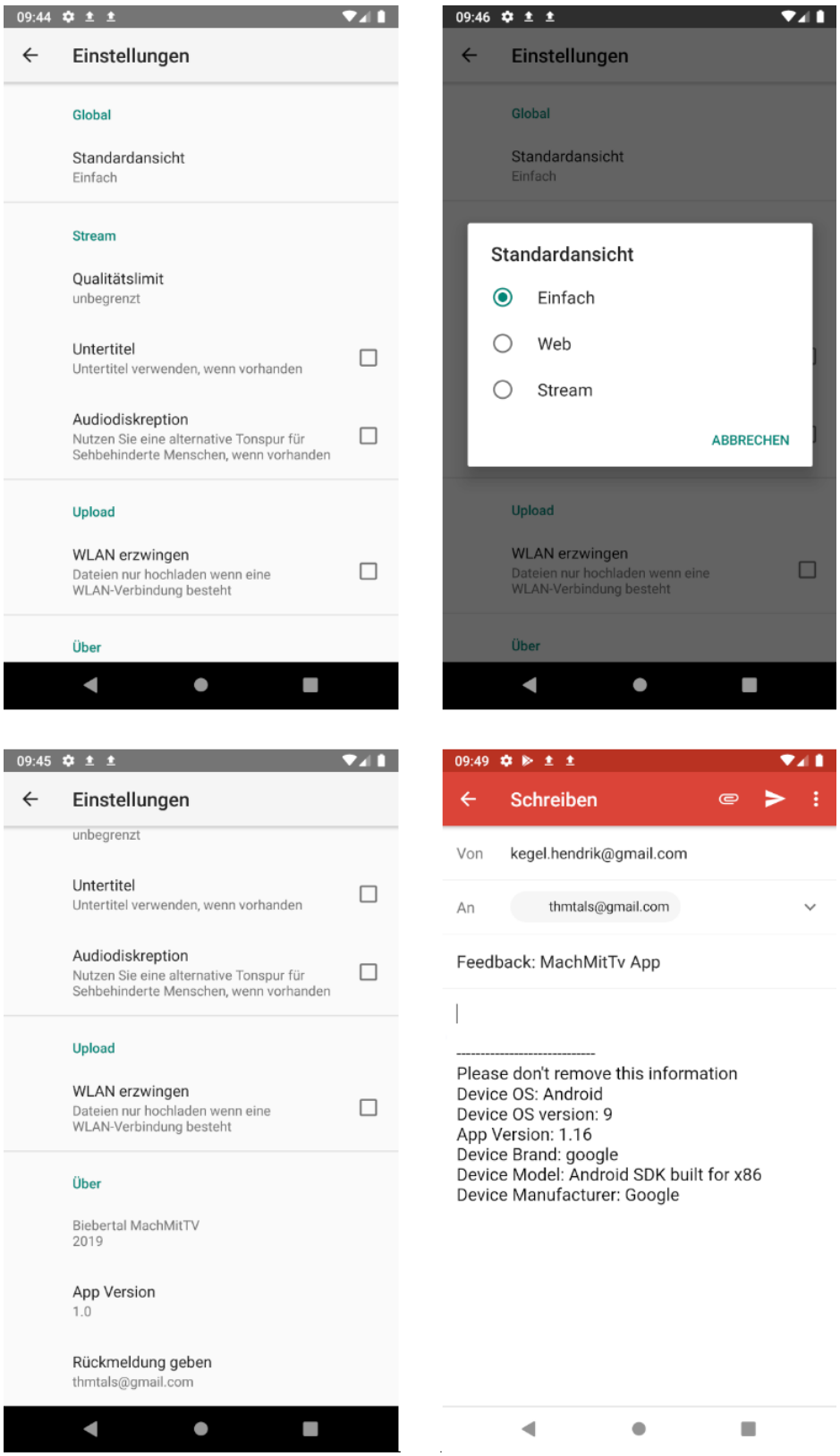


Abbildung 4.12 Einstellungen

## 4.8 TV

### Bedienung

Auf dem TV wird der Livestream direkt gestartet. Die anderen Funktionen sind hier weder verfügbar noch erreichbar. Zur besseren Bedienbarkeit wurde der Controller angepasst. Die Bedienung erfolgt über die Fernbedienung des Gerätes, hierzu können sowohl das Steuerkreuz als auch die direkt zugeordneten Funktionstasten verwendet werden.

### Implementierung

Die auf dem TV zu startende Activity wird mittels Intentfilter im Manifest festgelegt. Hier wird die LiveStreamActivity verwendet. Diese überprüft beim Start auf welchem Gerätetyp sie zum Einsatz kommt und passt den Controller entsprechend an.

Listing 4.10 Startactivity für TV festlegen

```
AndroidManifest::application
    <activity
        android:name=".view.activity.LiveStreamActivity"
        [...]
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LEANBACK_LAUNCHER" />
        </intent-filter>
    </activity>
```

Listing 4.11 Anpassung des Controllers

```
LiveStreamActivity::onCreate(Bundle savedInstanceState)
    if (App.isDeviceTV()) controls.setTv();
    else controls.setMobile();

BasePlaybackActivity.PlayerControls::setTV
    assert LEAVE, INFO, SETTINGS & MORE are Views and initialized

    LEAVE.setVisibility(View.VISIBLE);
    INFO.setVisibility(View.VISIBLE);
    SETTINGS.setVisibility(View.GONE);
    MORE.setVisibility(View.GONE);
```


Abbildung 4.13 Livestream auf dem TV


Abbildung 4.14 Angepasster Controller

# Kapitel 5

# Evaluation

Die App wurde zur Erlangung einer nachvollziehbaren Bewertung einer Evaluation unterzogen, deren Ablauf und Ergebnisse im Folgenden beschrieben werden.

## 5.1 Vorgehen

Die Evaluation wurde im Rahmen einer Familienfeier durchgeführt. Anwesend waren 26 Personen verschiedensten Alters. Hieraus wurden 12 ausgewählt, die in ihrem Alltag ein Android Gerät verwenden. Ihnen wurde mitgeteilt, dass die Durchführung der Evaluation eine für die Hochschule relevante Übung sei. Die App wurde dabei als zufällig gewählt beschrieben, so dass die Teilnehmer nicht durch ihre familiäre Zugehörigkeit zum Entwickler beeinflusst wurden. Nach einer Einführung zu Mach-Mit.TV wurden die Probanden gebeten, die App anhand der folgenden Aufgaben zu testen und mögliche Probleme oder Anregungen zu notieren.

1. Sehen Sie sich den Stream von Mach-Mit.TV an.
2. Starten Sie ein Video mittels QR-Code.
3. Laden Sie ein Video hoch.
4. Finden Sie die Version der App.
5. Senden Sie dem Entwickler eine Nachricht mittels der integrierten Feedback Funktion.

Für den Test wurde ein HTC One M8S (Android 8.1) und ein Sony Xperia Z1 Compact (Android 9) zur Verfügung gestellt. Beide Geräte verfügen über ein CustomRom, was auf den Test aber keinen weiteren Einfluss hat. Zusätzlich wurden vorhandene Geräte der Probanden verwendet.

## 5.2 Sinnhaftigkeit

Zum Erhalt eines einheitlichen Ergebnisses, beantworteten die Tester die folgenden Fragen. Mangels eines TV-Geräts konnte die TV App nicht gezeigt werden. Der Inhalt wurde daher anhand des Streaming-Features der Smartphone-App erklärt.

- Ist Mach-Mit.TV als Dienst Ihrer Meinung nach sinnvoll?
- Ist die Ergänzung von Mach-Mit.TV um eine Android-App Ihrer Meinung nach sinnvoll? Unterscheiden Sie zwischen TV und Smartphone App.
- Sind die in der App realisierten Features Ihrer Meinung nach sinnvoll? Beziehen Sie sich nur auf die Smartphone App.

Das Ergebnis der Befragung wird in der folgenden Grafik zusammengefasst. Es zeigt sich, dass der Großteil der Befragten sowohl die Erweiterung der Plattform um eine mobile Anwendung als auch die umgesetzten Features als sinnvoll erachtet. Eine kleine Ausnahme ist, dass etwa 30% der Befragten den Stream auf dem Smartphone als unnötig empfinden. Sie gaben hierzu an, aufgrund der geringen Displaygröße keine Videos auf dem Smartphone anzusehen.

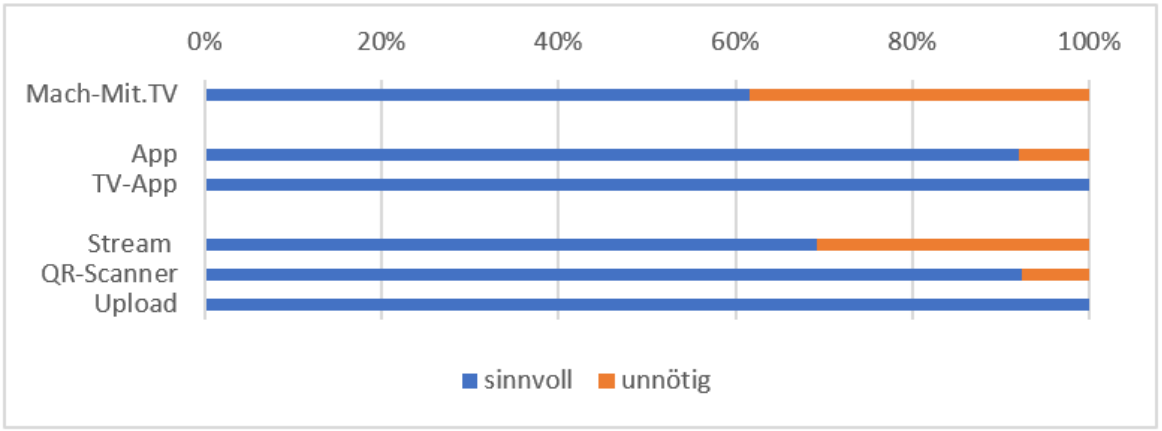


Abbildung 5.1 Ergebnis zur Sinnhaftigkeit

## 5.3 Umsetzung

Da die TV-App lediglich zur Wiedergabe des Streams dient und in Funktion und Optik dem Unterpunkt Stream der Smartphone-App sehr ähnelt, wird hier nur auf letztere eingegangen. Zur Überprüfung der Qualität der Umsetzung wurden die Probanden gebeten, die einzelnen Features der App sowohl technisch als auch optisch zu bewerten. Die technische Umsetzung umfasst Aspekte wie ruckelfreies Abspielen des Streams, flüssige Navigation zwischen den verfügbaren Funktionen und natürlich die Fehlerfreiheit selbiger. Die optischen Aspekte hingegen sind deutlich subjektiver. Hierzu gehört beispielsweise das Farbschema, die Anordnung und Sichtbarkeit von Elementen aber auch die Intuitivität der Bedienung.

Das Ergebnis der Befragung zeigt eine allgemeine Zufriedenheit mit der technischen Umsetzung bei Stream und QR-Code-Scanner. Beim Upload offenbarte sich ein Problem mit den Benachrichtigungen. Hier fehlt eine Funktion den Upload über diese zu pausieren und fortzusetzen, womit die Bedienung erleichtert würde. Optisch zeichnet sich ein ähnliches Bild. Der Stream und der QR-Code-Scanner gefällt den Testern sehr gut, während die Oberfläche des Uploadbereichs als „eher funktional“ gesehen wird. Selbiges gilt für den Startscreen. Das sehr minimalistische Design spaltet die Meinung der Tester.

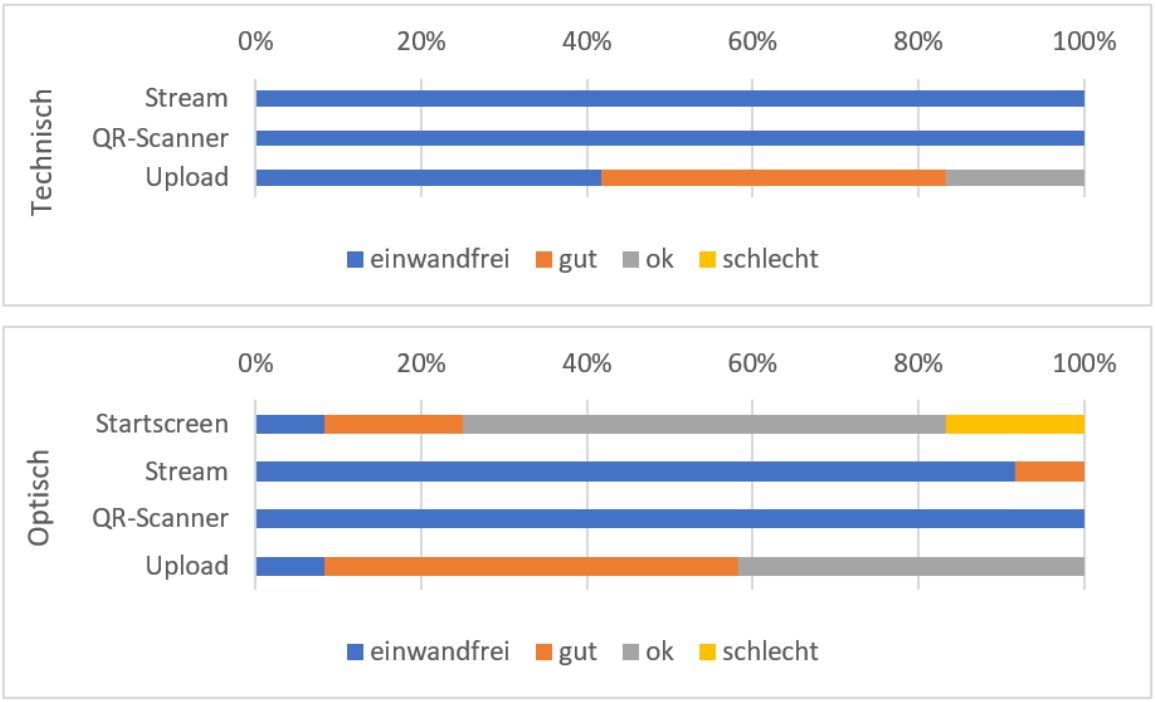


Abbildung 5.2 Ergebnisse zur Umsetzung

## 5.4 Anregungen

Der folgende Abschnitt umfasst Anregungen zu neuen und Verbesserungsvorschläge zu aktuellen Features. Hierzu wurden während der Durchführung der Evaluation aufgekommene Probleme und Fragen zusammengefasst und zur Abstimmung bezüglich deren Relevanz gestellt. Die folgende Grafik zeigt die aufgekommenen Verbesserungsvorschläge in Relation zu ihrer geschätzten Wichtigkeit.

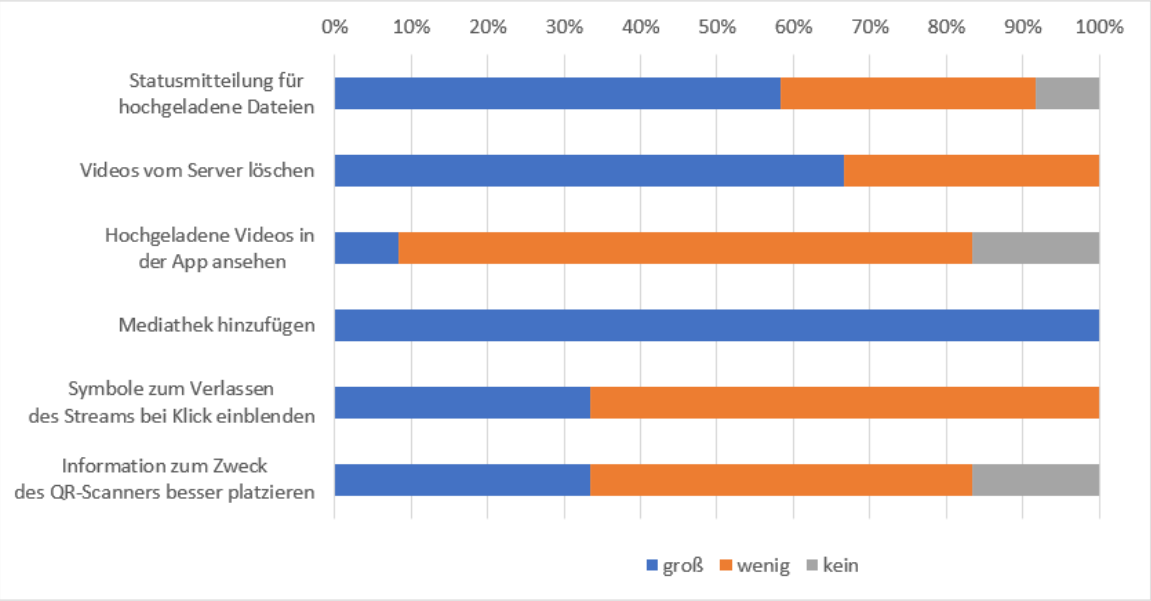


Abbildung 5.3 Anregungen in Relation zu ihrer geschätzten Wichtigkeit

Es wird deutlich, dass sich die Probanden mehr Informationen über die von Ihnen hochgeladenen Dateien wünschen. Nach dem Upload erhält der Nutzer derzeit lediglich die Information, das selbiger abgeschlossen sei. Hier wäre eine Statusmeldung mit Informationen über den Stand der Verarbeitung des Videos erfreulich. Außerdem wird sich mehr Kontrolle über die hochgeladenen Dateien gewünscht. Hierzu sollen diese wieder vom Server gelöscht werden können, sofern der Nutzer dies möchte. Neben den Verbesserungen für den Upload empfindet jeder der Teilnehmer die Umsetzung einer Mediathek als notwendige Ergänzung der Anwendung. Der Konsens besteht hierbei darin, dass Videos gezielt anzusehen sein sollen. Einigen Testern war außerdem unklar, wofür der QR-Code Scanner eingesetzt werden soll. Die zugehörige Information scheint schlecht platziert zu sein. Selbiges gilt für das Verlassen des Streams. Durch das Öffnen im Vollbild-Modus werden alle Bedienelemente des Smartphones ausgeblendet und können nur durch ein Wischen wieder in den Vordergrund geholt werden. Die Tester wünschen sich daher, dass die Bedienelemente bei Klick auf den Bildschirm wieder eingeblendet werden.

# Kapitel 6

# Fazit

Das Ergebnis der Evaluation hat deutlich gezeigt, dass die hier entwickelte Anwendung für eine effiziente Nutzung noch um einige Features ergänzt werden sollte. Hierbei sind insbesondere die Erweiterung des Uploads um Statusmitteilungen und das Löschen der Dateien vom Server gemeint. Außerdem sollte die App um eine Mediathek ergänzt werden. Der Nutzer fordert in Zeiten von Netflix, YouTube, Spotify und Co Content gezielt abrufen zu können. Abgesehen vom Fehlen dieser Erweiterungen lässt sich dennoch ein positives Fazit ziehen. Die App hat mit der vollständigen Implementierung des Players, des QR-Code-Scanners und des TuS-Uploads alle technischen Grundlagen für eine erfolgreiche Erweiterung der Plattform. Der Player beherrscht alle geforderten Funktionen und kann zur Realisierung der Mediathek einfach wiederverwendet werden. Der QR-Code-Scanner ist vollständig umgesetzt und bedarf lediglich einer besseren Platzierung der Information bezüglich seines Einsatzzweckes. Die Dateiuploads mittels TuS-Protokoll basieren auf einer erprobten und gewarteten Technologie und bieten ebenfalls alle Möglichkeiten für zukünftige Erweiterungen. Dazu kommt mit der TV Anwendung eine hier nur recht kurz beschriebene aber dennoch für Mach-Mit.TV bedeutende Erweiterung der Plattform. Mit ihr kann das Programm über die Geräte in Apotheken oder dem Rathaus verbreitet und den Bewohnern der Gemeinden näher gebracht werden. Die wesentliche Erkenntnis dieser Theses ist allerdings, dass eine solch komplexe Anwendung stets Potential zur Erweiterung bietet und eine vollständig perfekte App wahrscheinlich nicht existiert. Damit eine Anwendung ihren gewünschten Mehrwert erbringt, muss sie stets weiterentwickelt werden. Denn …

*Wer aufhört, besser zu werden, hat aufgehört, gut zu sein. - Philip Rosenthal*

# Literaturverzeichnis


[1] Akissko. WebTV von Offenbach. http://www.offenbachtv.net, Aufruf: 15.08.2019 .

[2] Apple Inc. HTTP Live Streaming Overview. https://developer.apple.com/library/archive/documentation/NetworkingInternet/Conceptual/StreamingMediaGuide/Introduction/Introduction.html, Aufruf: 05.08.2019 .

[3] Arno Becker and Marcus Pant. In Android 2. dpunkt.verlag, 2010. ISBN 9783898646772.

[4] Romain Bouqueau. Apple HLS: comparing versions vom 01.12.2014. https://www.gpac-licensing.com/2014/12/01/apple-hls-comparing-versions, Aufruf: 03.08.2019 .

[5] Dushyanth. qrcodescanner. https://github.com/dm77/barcodescanner, Aufruf: 05.09.2019 .

[6] Ben Elgin. Google Buys Android for Its Mobile Arsenal (2005). https://web.archive.org/web/20111021061828/http://www.businessweek.com/technology/content/aug2005/tc20050817_0949_tc024.htm, Aufruf: 08.09.2019 .

[7] Google LLC. Android 5.0 APIs. https://developer.android.com/about/versions/android-5.0, Aufruf: 05.09.2019 .

[8] Google LLC. Platform Architecture. https://developer.android.com/guide/platform, Aufruf: 30.07.2019 .

[9] Google LLC. Distribution Dashboard. https://developer.android.com/about/dashboards, Aufruf: 29.07.2019 .

[10] Google LLC. Intent. https://developer.android.com/reference/android/content/Intent, Aufruf: 31.07.2019 .

[11] Google LLC. Understand the Activity Lifecycle. https://developer.android.com/guide/components/activities/activity-lifecycle, Aufruf: 05.09.2019 .

[12] Google LLC. ExoPlayer Release Page on Github. https://github.com/google/ExoPlayer/releases, Aufruf: 15.08.2019 .

[13] Google LLC, GitHub Inc., and desarrollo-ypfb. Logos. https://git.ypfb.gob.bo/desarrollo-ypfb, https://de.wikipedia.org/wiki/Android_Studio, https://github.com/logos, Aufruf: 02.09.2019 .

[14] Larry Jordan. The Basics of http Live Streaming vom 09.06.2013. https://larryjordan.com/articles/basics-of-http-live-streaming, Aufruf: 03.08.2019 .

[15] Marcel Laser. Android 10: Google macht Schluss mit Süssigkeiten im Namen (2019). https://www.pocketpc.ch/magazin/news/android/android-10-google-neuer-name-69569/, Aufruf: 08.09.2019 .

[16] Margha GmbH. WebTV von Altötting. http://www.muehldorf-tv.de/altoetting, Aufruf: 15.08.2019 .

[17] Margha GmbH. WebTV von Mühldorf. http://www.muehldorf-tv.de, Aufruf: 15.08.2019 .

[18] Florina Muntenescu. Android Architecture Patterns Part 1: Model-View-Controller aus 2016. https://medium.com/upday-devs/android-architecture-patterns-part-1-model-view-controller-3baecef5f2b6, Aufruf: 31.07.2019 .

[19] Open Handset Alliance. FAQ. http://www.openhandsetalliance.com/oha_faq.html, Aufruf: 29.07.2019 .

[20] Ingo Pakalski. Android war ursprünglich für Digitalkameras gedacht. https://www.golem.de/news/andy-rubin-android-war-urspruenglich-fuer-digitalkameras-gedacht-1304-98779.html, Aufruf: 08.09.2019 .

[21] Roger Pantos and William May. HTTP Live Streaming - Abschnitt 3.2. https://tools.ietf.org/html/draft-pantos-http-live-streaming-20#section-3.2, Aufruf: 05.08.2019 .

[22] Roger Pantos and William May. HTTP Live Streaming - Abschnitt 3.4. https://tools.ietf.org/html/draft-pantos-http-live-streaming-20#section-3.4, Aufruf: 05.08.2019 .

[23] René Preißel and Bjørn Stachmann. In Git: Dezentrale Versionsverwaltung im Team – Grundlagen und Workflows. dpunkt.verlag, 2019. ISBN 9783864906497.

[24] Joel Rosenblatt and Jack Clark. Google's Android Generates $31 Billion Revenue, Oracle Says. https://www.bloomberg.com/news/articles/2016-01-21/google-s-android-generates-31-billion-revenue-oracle-says-ijor8hvt, Aufruf: 08.09.2019 .

[25] StatCounter. Mobile Operating System Market Share Worldwide. http://gs.statcounter.com/os-market-share/mobile/worldwide, Aufruf: 29.07.2019 .

[26] Statistisches Bundesamt. Ausstattung privater Haushalte mit Informations- und Kommunikationstechnik Deutschland. https://www.destatis.de/DE/Themen/Gesellschaft-Umwelt/Einkommen-Konsum-Lebensbedingungen/Ausstattung-Gebrauchsgueter/Tabellen/a-evs-infotechnik-d.html, Aufruf: 27.07.2019 .

[27] Michael Tamm. In JUnit-Profiwissen: Effizientes Arbeiten mit der Standardbibliothek für automatische Tests in Java. dpunkt.verlag, 2013. ISBN 978-3864900204.

[28] Transloadit Ltd. TuS Android Client. https://github.com/tus/tus-android-client, Aufruf: 20.09.2019 .

[29] Transloadit Ltd. TuS. https://github.com/tus, Aufruf: 05.09.2019 .

[30] Ruth van Doornik. Die Big Player wachsen, die Kleinen bleiben auf der Strecke. https://www.welt.de/regionales/bayern/article176572174/Ladensterben-auf-dem-Land-Die-Big-Player-wachsen-die-Kleinen-bleiben-auf-der-Strecke.html, Aufruf: 27.07.2019 .

[31] Sujeevan Vijayakumaran. In Versionsverwaltung mit Git. mitp, 2019. ISBN 9783898646772.

[32] David Watkins. Global Connected TV Device Vendor Market Share Q4 2018. https://news.strategyanalytics.com/press-release/intelligent-home/strategy-analytics-samsungs-tizen-os-leads-global-smart-tv-market, Aufruf: 29.07.2019 .

[33] Olly Woodman. ExoPlayer 2 - Why, what and when? vom 14.06.2016. https://medium.com/google-exoplayer/exoplayer-2-x-why-what-and-when-74fd9cb139, Aufruf: 15.08.2019 .

[34] Zencoder.com. HLS-Guide. https://zencoder.com/de/hls-guide, Aufruf: 03.08.2019 .

# Abbildungsverzeichnis

# Listings