22.02.2021

Entwicklung einer Streaming App

Teamprojekt WS2020

StreamApp

Inhalt

- Überblick über die entwickelte App
- Verwendete Technologien/Libraries
- Softwarearchitektur
- Probleme und Lösungen beim Entwickeln
- eigene Features

Überblick über die entwickelte App

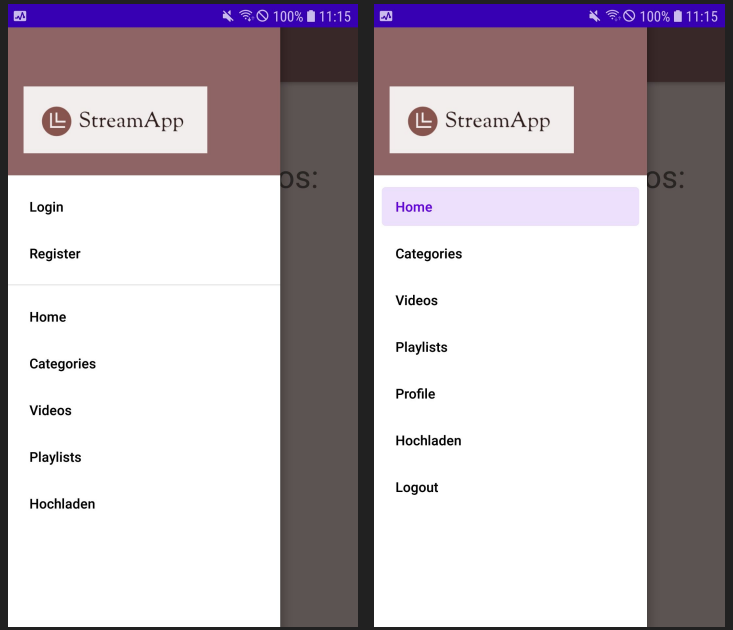

Abb. 1: Sidebar Einträge ohne/mit Login

⇒ Demo Time!

Überblick über die entwickelte App

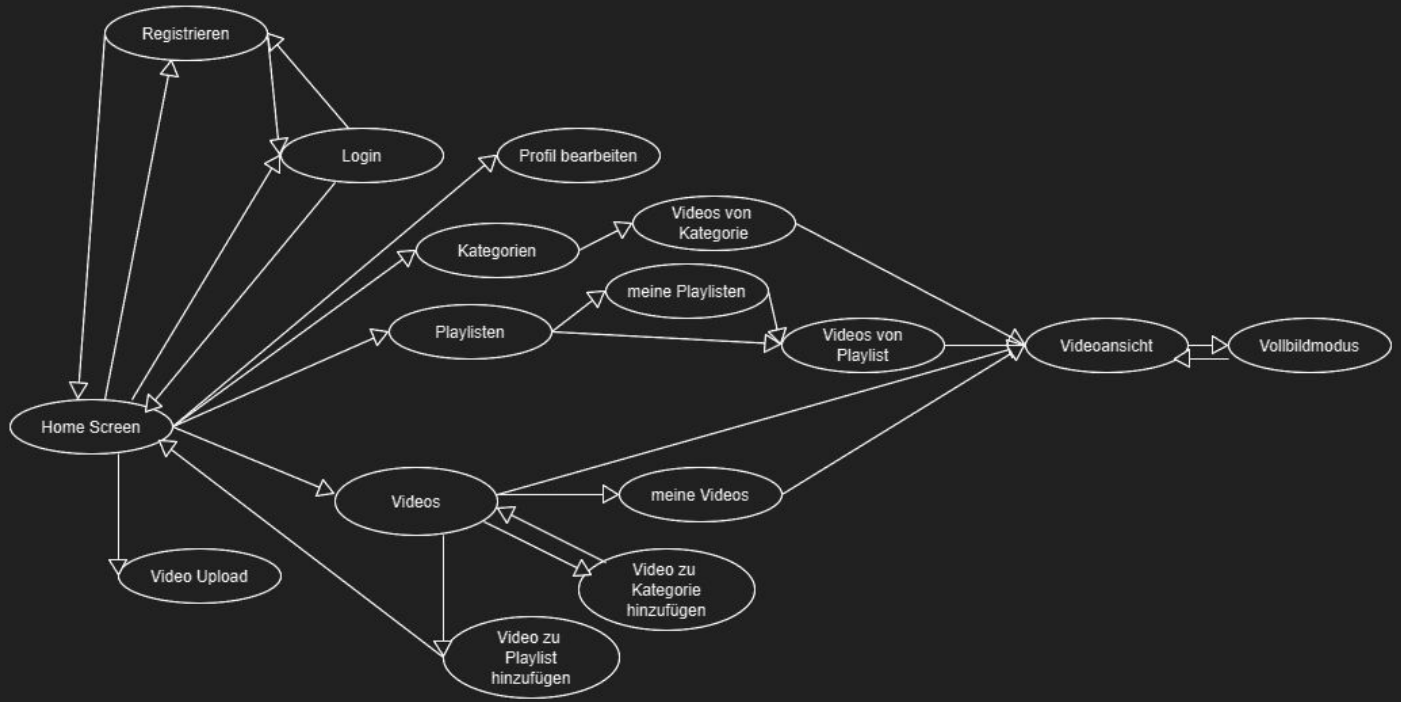

Abb. 2: Netzwerkgraph der Views

Verwendete Technologien/Libraries

HTTP Requests: Retrofit

- Open Source Projekt (Apache v2.0) von “Square Inc.”
- Endpunkte werden als Interface abgebildet

```
72  @GET("v1/users/{uid}")
73  Call<User> getUser(@Header("Authorization") String apikey,
74                     @Path("uid") int userId);
```

Abb. 3: Ausschnitt aus dem Interface KetosService

Verwendete Technologien/Libraries

JSON Convertierung: FastJSON

- Server Anfragen/Antworten sind im *JSON*-Format
  ⇒ müssen zu Java Objekten umgewandelt werden
- Wird von “Alibaba” entwickelt
- FastJSON als Retrofit kompatibler Converter mittels Github Projekt
  [ligboy/retrofit-converter-fastjson]

Einbindung von Bibliotheken über Gradle

Softwarearchitektur

MVVM Beispiel

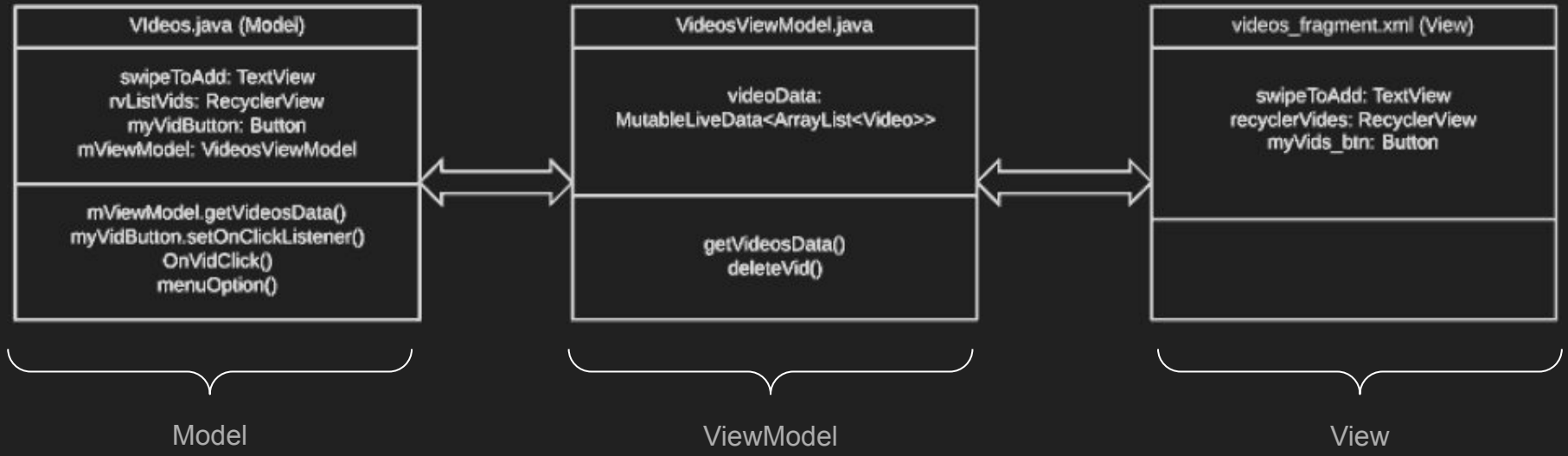

Abb. 4: MVVM Beispiel

LiveData

MutableLiveData

```
// Livedata for Videos
private MutableLiveData<ArrayList<Video>> VideosData;

MutableLiveData<ArrayList<Video>> getVideosData(int count) {

    // init if null
    if (VideosData == null) {
        VideosData = new MutableLiveData<>();
    }

    // RESTRequest instance
    RESTRequest rr = new RESTRequest();
    // do request
    try {
        VideosData.setValue(rr.getVideos( limit: 20, count));
    } catch (Exception e) {
        VideosData.setValue(null);
    }
    // return Categories
    return VideosData;
}
```

befüllen der MutableLiveData mit einer ArrayList mit Videos

Abb. 5: Beispiel LiveData

RESTRequest Klasse

Klasse kapselt die gesamte Interaktion mit dem Backend ab.

```
25  private Retrofit retrofit = new Retrofit.Builder()
26          .baseUrl(this.API)
27          .addConverterFactory(FastJsonConverterFactory.create())
28          .build();
29
30  private KetosService service = retrofit.create(KetosService.class);
```

Abb. 6: Erstellung des Retrofit-Clients innerhalb von RESTRequest

RESTRequest Klasse

Aufruf der Request mittels Methoden der Form:

```
82  public String login(final String username, final String password) throws Exception {
83      Response<ApiKey> call = service.login(new LoginBody(username, password)).execute();
84
85      if (!call.isSuccessful()) throw new Exception("Request unsuccessful");
86
87      if (call.code() == 200) {
88          this.setApikey(call.body().getApikey());
89          return this.apikey;
90      } else {
91          throw new Exception(call.errorBody().string());
92      }
93  }
```

Abb. 7: Funktionalität der login Methode

JSON Abbildung in Java

```java
class Beispiel {
    @JSONField(name = "id")
    int id;

    public Beispiel() {}

    @JSONField(name = "id")
    public void setId(int id) {
        this.id = id;
    }

    @JSONField(name = "id")
    public int getId() {
        return this.id;
    }
}
```

Abb. 8: Klasse die mit FastJSON konvertierbar ist

Anforderungen für FastJSON:

1. generischer Konstruktor
2. Setter und Getter für alle JSON Einträge
3. Variable, Getter und Setter mittels @JSONField(name = “<name>”) annotieren.

Datei Upload

Das Storage Access Framework *(SAF)* Stellt den integrierten *Picker* bereit.

```
111 // Button to select Video
112 btnSelectVideo.setOnClickListener(new View.OnClickListener() {
113     @Override
114     public void onClick(View v) {
115         Intent chooseFile;
116         Intent intent;
117         chooseFile = new Intent(Intent.ACTION_GET_CONTENT);
118         chooseFile.addCategory(Intent.CATEGORY_OPENABLE);
119         chooseFile.setType(getString(R.string.vu_MIME));
120         intent = Intent.createChooser(chooseFile, getString(R.string.vu_chooseVideo));
121         startActivityForResult(intent, ACTIVITY_CHOOSE_FILE);
122     }
123 });
```

Abb. 9: onClickListener der den Picker aufruft

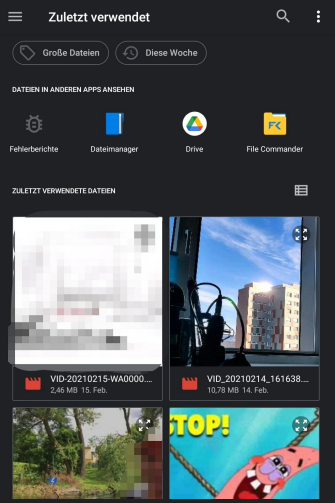

Abb. 10: Systemkomponente Picker

Datei Upload

- *Picker* liefert nur die URI.
- entweder *content* oder *virtual*.
- Je eine Methode die InputStream in OutputStream umwandelt
- Video in Temp-Datei zwischenspeichern

```
// Make the file Multipart
MultipartBody.Part video = MultipartBody.Part.createFormData(
        "file",
        file.getName(),
        RequestBody.create(MediaType.parse(mimeType), file));

RequestBody titleBody = RequestBody.create(
                MediaType.parse("text/plain"), title);
RequestBody describtionBody = RequestBody.create(
                MediaType.parse("text/plain"), describtion);
```

Abb. 11: Ausschnitt der uploadVideo-Methode

Probleme und Lösungen

- Einarbeitung anfangs schwierig
  - anfangs viel Zeit investieren
  - Alle Konzepte und Grundlagen nur von den google.developer Webseiten recherchieren

- Android Studio braucht zu viel RAM
  - Power Save Mode
  - Testumgebung auf externem Gerät

Probleme und Lösungen

- Unzureichende Backend Dokumentation
  - undurchsichtige Informationen zu benötigten Parametern
  - Teilweise fehlende Angabe von HTTP Statuscodes

Reflexion des Projekts

- ViewModels abstrahieren
  - ein ViewModel für mehrere Models um Daten auszutauschen
- Java vs Kotlin
  - Kotlin von Google empfohlen
  - mit Kotlin weniger Code nötig
  - Java bewährt und sehr viele Bibliotheken
- Alternativen zu Android Studio
  - Flutter, Ionic,...

eigene Features

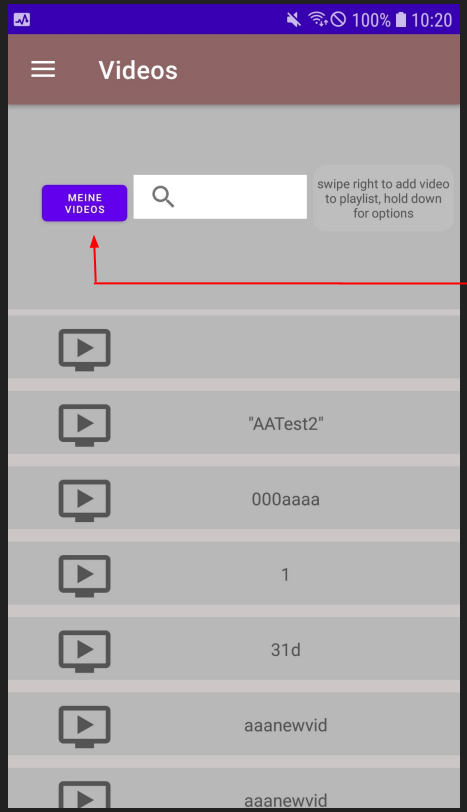

Abb. 12: eigene Videos

- eigene Videos können angezeigt werden
- trending Videos werden auf dem Home Screen angezeigt

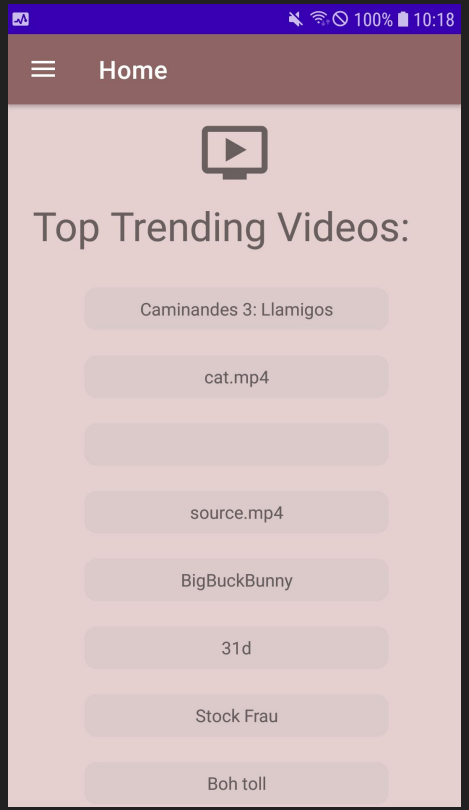

Abb. 13: Top Trending Videos

Fin

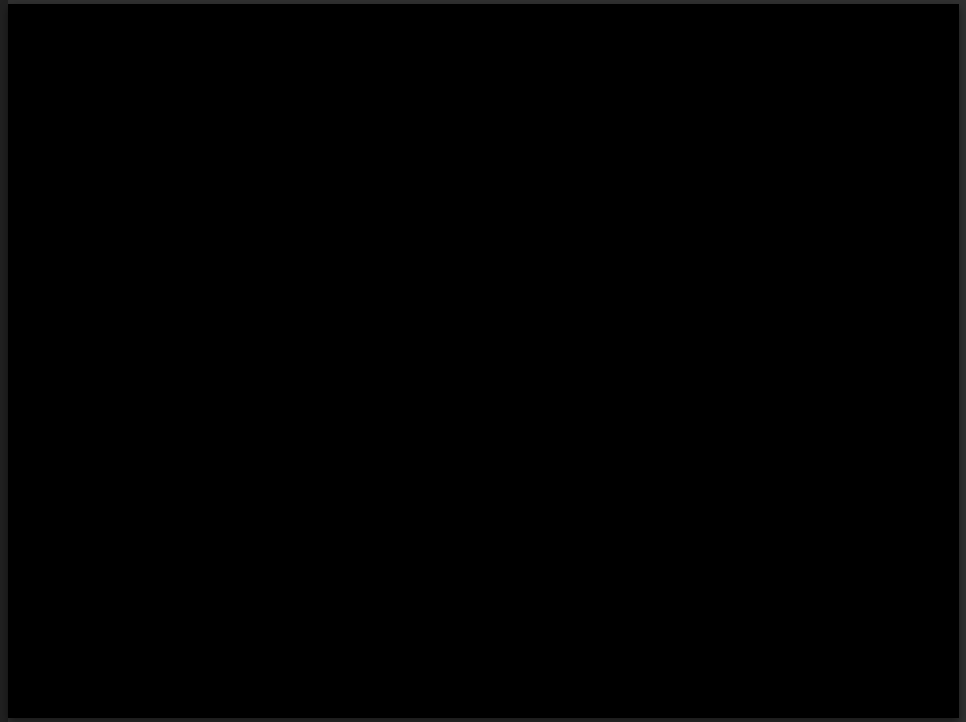

Abb. 14: Videoansicht mit Abschlussvideo