Mobilna aplikacija za analizu podataka YouTube kanala

Boc, Andreas

SVEUČILIŠTE JURJA DOBRILE U PULI
FAKULTET INFORMATIKE

ANDREAS BOC

Mobilna aplikacija za analizu podataka YouTube kanala

DIPLOMSKI RAD

Pula, lipanj 2024. Godine

SVEUČILIŠTE JURJA DOBRILE U PULI
FAKULTET INFORMATIKE

ANDREAS BOC

Mobilna aplikacija za analizu podataka YouTube kanala

DIPLOMSKI RAD

JMBAG: 0165066950

Studijski smjer: Informatika

Kolegij: Mobilne aplikacije

Znanstveno područje: Društvene znanosti

Znanstveno polje: Informacijske i komunikacijske znanosti

Znanstvena grana: Informacijski sustavi i informatologija

Mentor : izv.prof.dr.sc. Siniša Sovilj

Pula, lipanj 2024. Godine

SAŽETAK

U ovom diplomskom radu izrađena je mobilna aplikacija za provođenje analize podataka YouTube-a. Aplikacija korisnicima pruža praćenje ključnih metrika performansi YouTube kanala, poput broja pregleda, broja komentara, broja pretplatnika, broja “lajkova” i slično. Aplikacija nudi detaljan pregled rasporeda objava videozapisa po danima u tjednu kao i analizu omjera pregleda u odnosu na vrijeme objave videozapisa. Aplikacija omogućava korisniku pregled te analizu komentara na videozapisima. Funkcionalnosti sustava uključivat će: prikaz metrika kanala i videozapisa, analiza komentara i angažmana publike, preporuke za optimalno vrijeme objavljivanja. Programski jezici Java i Kotlin te Android studio detaljno su opisani. Korisničke upute jasno su strukturirane i sadržavaju korake za pristup aplikaciji, pregledavanje metrika, analizu komentara, te analizu samih videozapisa.

Ključne riječi: YouTube, mobilna aplikacija, videozapisi, pregledi, pretplatnici, komentari, “lajkovi”, Java, Kotlin, Android Studio.

ABSTRACT

In this thesis, a mobile application for conducting data analysis of YouTube was developed. The application provides users with the ability to track key performance metrics of YouTube channels, such as the number of views, number of comments, number of subscribers, number of likes and more. The application offers a detailed overview of the schedule of video uploads by day of the week, as well as analysis of the view ratio concerning the time of video upload. The system's functionalities will include: displaying channel and video metrics, analyzing comments and engagement, and providing recommendations for optional upload times. The programming languages Java and Kotlin, as well as Android Studio, are described in detail. The user manual is clearly structured and contains steps for accessing the application, viewing metrics, analyzing comments, and analyzing the videos themselves.

Keywords: YouTube, mobile application, videos, views, subscribers, comments, likes, Java, Kotlin, Android Studio.

POPIS SKRAĆENICA

JSON - JavaScript Object Notation

UTC - Coordinated Universal Time

APK – Android Package Kit

IDE - Integrated Development Environment

JDK - Java Development Kit

JVM - Java Virtual Machine

JRE - Runtime Environment

SVEUČILIŠTE JURJA DOBRILE U PULI

FAKULTET INFORMATIKE

Pula, 21. veljače 2024.

DIPLOMSKI ZADATAK

Pristupnik: **Boc Andreas (0165066950)**
Studij: Sveučilišni diplomski studij Informatike

Naslov(hrv.): **Mobilna aplikacija za analizu podataka YouTube kanala**
Naslov(eng.): Mobile application for data analysis of YouTube channels

Opis zadatka: Izraditi mobilnu aplikaciju koja će korisnicima pružiti detaljnu analitiku podataka o njihovim YouTube kanalima. Za razvoj aplikacije potrebno je koristiti Android Studio s programskim jezikom Java/Kotlin. Aplikacija će korisnicima pružiti praćenje ključnih metrika performansi svog kanala poput broja pregleda, broja komentara, broja pretplatnika, broja „lajkova“ i slično. Aplikacija treba omogućiti prikaz broja pregleda unutar različitih vremenskih perioda, prikazivati broj novih pretplatnika na kanalu u određenom vremenskom periodu. Omogućiti korisnicima pregled i analizu komentara na njihovim videozapisima kako bi bolje razumjeli mišljenja publike. Također korisnici trebaju moći pregledati detaljne performanse specifičnog videozapisa. Funkcionalnosti sustava trebaju uključivati: prikaz metrika kanala i videozapisa, analiza komentara i angažmana publike, preporuke za optimalno vrijeme objavljivanja. Istražiti slična postojeće programska rješenja te navesti razlike u odnosu na vlastitu aplikaciju. Klasne dijagrame, use case dijagrame te sekvencijske dijagrame potrebno je unijeti u diplomski rad. Izraditi i kratke korisničke upute za pristupanje aplikaciji, pregledavanje metrika, analizu komentara i korištenje preporuka za optimalno vrijeme objavljivanja videozapisa.

Zadatak uručen pristupniku: 23. veljače 2024.
Rok za predaju rada: 23. veljače 2025.

Mentor:
Siniša Sovilj
izv.prof.dr.sc. Siniša Sovilj

SADRŽAJ

1. UVOD

YouTube je popularna online platforma za dijeljenje videozapisa. Ova platforma osnovana je 2005. godine. Na njoj korisnici mogu objavljivati, pregledavati, dijeliti te komentirati videozapise različitih žanrova, kao što su glazba, vlogovi, obrazovni sadržaji, igre i slično. YouTube pruža priliku pojedincima da izgrade svoju publiku te stvore online zajednice. Kreatorima koji zadovolje uvijete (1000 pretplatnika i 10 000 sati pregleda), YouTube omogućuje unovčavanje (eng. *monetization*) sadržaja. Unovčavanje sadržaja znači da kreatori imaju mogućnost zarade od oglasa, pregleda, donacija. Na taj način ova platforma potiče kreatore na stvaranje kvalitetnog sadržaja.

Slika 1.1. YouTube logo

Preuzeto s: https://www.vecteezy.com/png/17396826-youtube-studio-icons

Mogućnost zarade na YouTube-u inspirala me te sam odlučio započeti vlastiti hobi objavljivanja videozapisa. YouTube mi pruža priliku izraziti se na kreativni način i istovremeno povezivati s ljudima koji dijele slične interese. Kako bih poboljšao statistike na svom kanalu, odlučio sam analizirati podatke te usporediti svoj kanal s kanalima sličnog sadržaja. Proučavanjem statistike, poput broja pretplatnika, broja komentara te broja videozapisa, ova analiza će mi omogućiti poboljšanje mojeg sadržaja, što bi trebalo rezultirati rastom broja gledatelja i pretplatnika na mom kanalu.

Rad je organiziran na sljedeći način. U drugom poglavlju dan je pregled slične aplikacije za analizu YouTube podataka. Aplikacija se naziva YouTube Studio. U trećem poglavlju opisane su korištene tehnologije (Java, Koltin) za izradu mobilne aplikacije. Četvrto poglavlje prikazuje način kreiranja i preuzimanja YouTube Data API ključa i ID-a YouTube kanala. U petom poglavlju predstavljen je konačan proizvod, njegov izgled te funkcionalnosti. Šesto poglavlje prikazuje način i upute korištenja aplikacije. Sedmo poglavlje donosi zaključak projekta.

Izvorni kod aplikacije dostupan je na GitHub-u: https://github.com/AndreasBoc/Mobile-application-for-data-analysis-of-YouTube-channels.git

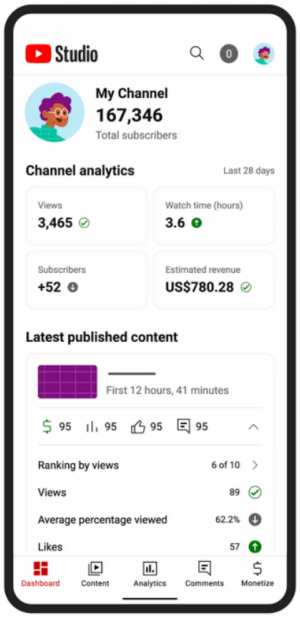
2. POSTOJEĆA APLIKACIJA ZA ANALIZU YOUTUBE PODATAKA (YOUTUBE STUDIO)

YouTube Studio mobilna je aplikacija namjenjena kreatorima sadržaja na YouTube platformi.

Slika 2.1. YouTube Studio logo

Preuzeto s: https://www.vecteezy.com/png/17396826-youtube-studio-icons

YouTube Studio aplikacija najbolji je način za razumjeti i povezati se s ljudima u svojoj zajednici. Aplikacija omogućuje brz i jednostavan pregled izvedbe sadržaja i kanala. Pomoću ove aplikacije i njene detaljne analitike moguće je razumjeti rezultate koje sadržaj i kanal ostvaruju. Dostupna je na iOS, windows i android platformama.



Slika 2.2. Korisničko sučelje aplikacije YouTube Studio

Preuzeto s: https://support.google.com/youtube/thread/188747005/%F0%9F%93%B2-big-updates-to-the-youtube-studio-app?hl=en

3. RAZVOJNO OKRUŽENJE I KORIŠTENE TEHNOLOGIJE

3.1. Android Studio

Android Studio je službeno integrirano razvojno okruženje (IDE) za razvoj Android aplikacija.

Slika 3.1.1. Android Studio

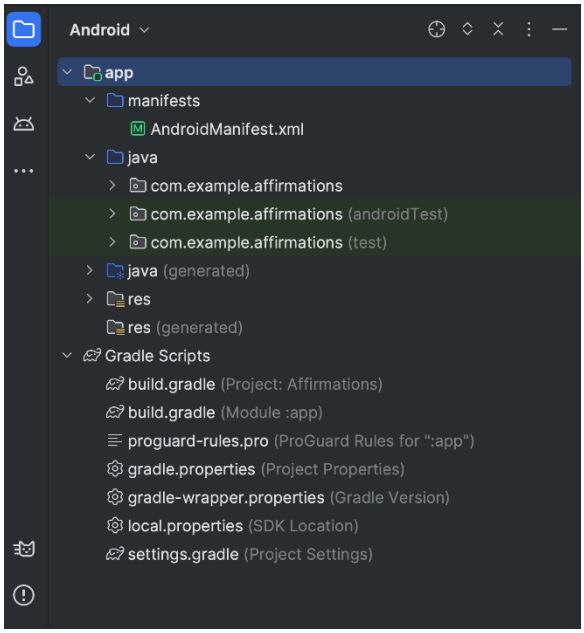
Preuzeto s: https://android-developers.googleblog.com/2020/02/android-studio-36.html

Android Studio temeljen je na IDEA platformi i pruža sve potrebne alate za razvoj, testiranje i debugiranje aplikacija. Android Studio uljučuje i Android Emulator, koji služi za simuliranje rada aplikacije. Također omogućuje testiranje aplikacije na uređajima povezanim s računalom.

Android Studio koristi Gradle kao alat za automatizaciju izrade projekata. Gradle omogućuje jednostavnu upravljanje ovisnostima (eng. *dependencies*), konfiguriranje aplikacije za različite uređaje, te jednostavno kreiranje APK datoteke. Android Studio podržava integraciju s Git-om i drugim kontrolnim sustavima kao što su Mercurial i Subversion, što olakšava timski rad. Prilikom kodiranja u stvarnom vremenu u Android Studiu, moguće je odmah vidjeti kako će aplikacija izgledati na ekranu.

Aplikacija izrađena u Android Studiu strukturirana je na sljedeći način:

- **Manifest** – sadrži datoteku AndroidManifest.xml, koja je ključni dio aplikacije i sadrži osnovne informacije o aplikaciji (naziv paketa, dozvole, aktivnosti, itd.).
- **java** - sadrži Java i Kotlin datoteke izvornog koda.
- **res** – sadrži resurse kao što su slike, layout-i, vrijednosti boja itd.



Slika 3.1.1. Android Studio

Preuzeto s: https://developer.android.com/studio/intro

Putem Android Studia moguće je razvijati aplikacije u različitim programskim jezicima, uključujući Javu, Kotlin i C++.

Android Studi podržan je na macOS, Windows te Linux operativnim sustavima. Za preuzimanje Android Studia, posjetite sljedeći link:

https://developer.android.com/studio

3.2. Java

Programski jezik Java stvoren je 1995. godine od strane James Goslinga u tvrtki Sun Microsystems, koju je kasnije otkupila tvrtka Oracle. Java je objektno orjentiran programski jezik.

Slika 3.2.1. Java[]

Preuzeto s: https://www.oracle.com/java/technologies/

Java radi na različitim platformama (Windows, macOS, Linux, Rapberry Pi, itd.). Otvoren je kod i besplatan. Brz je i siguran za korištenje. Dugovječnost Jave je impresivna, više od dva desetljeća nakon njenog stvaranja, Java je i dalje najpopularniji jezik za razvoj aplikacija.

Java kao jedan od najpoznatinijh programskih jezika korišten je za:

- Mobilne aplikacije,
- Desktop aplikacije,
- Web aplikacije,
- Web poslužitelje,
- Igre,
- Povezivanje s bazama.

Java je tehnologija koja koristi i programski jezik i softversku platformu. Za izradu aplikacije koristeći Javu, potrebno je preuzeti Java Development Kit (JDK). Program se piše u Java programskom jeziku, a zatim kompajler pretvara program u Java bytecode, skup instrukcija za Java Virtual Machine (JVM), koji je dio Java Runtime Envirionment (JRE).

3.3. Kotlin

Programski jezik Kotlin, znatno je mlađi od Jave. Kotlin je prvi put predstavljen 2016. godine. Kotlin predstavlja jezik otvorenog koda, koji također može kompilirati kod u bytecode i raditi na Java Virtual Machine (JVM), omogućujući mu tako rad na gotovo svakoj platformi. Kotlin je statički tipizirani programski jezik.

Kotlin kao kod koji je inspiriran Javom, teži pružiti poboljšan verziju koja je čišća, jednostavnija, brža te uključuje mješavinu objektno orijentiranog i funkcionalnog programiranja.

Slika 3.3.1. Kotlin

Preuzeto s: https://developer.android.com/codelabs/basic-android-kotlin-compose-first-program#0

Kotlin se fokusira na pojednostavljeno, funkcionalno kodiranje i na taj način izbjegava ponavljanje suvišnog koda. To znači da nisu potrebne točke i zarezi na kraju svake linije. Kotlin omogućuje sažetiji i kraći kod za istu funkcionalnost u usporedbi s drugim programskim jezicima. Također koristeći Kotlin, manje je vjerojatno da će se aplikacije izrađene pomoću Kotlina srušiti, što rezultira stabilnijom aplikacijom.

Neke od osnovnih razlika Kotlina i Jave su slijedeće:

1. Kod:
    a. Java – zahtjeva više koda za postizanje istih funkcionalnosti, što rezultira opsežnijim kodom, što može povećati mogućnost grešaka.
    b. Kotlin – zahtjeva manje linije koda, čineći tako kod kraćim i preglednijim.
2. Null varijable:
    a. Java- moguće je dodjeljivanje "*null*" vrijednosti bilo kojoj varijabli,a li tada se javlja "NullPointerExceptions" koju programeri moraju obraditi.

b. Kotlin – nije moguće dodijeliti "*null*" vrijednost varijablama jer tada dolazi do greške. Moguće je dodati "*null*" vrijednost tako da se varijablu označi kao nullable, dodavanjem upitnika: "*val number: Int? = null"*

3. Funkcije proširenja:
    a. Java – za proširivanje funkcija postojećeg razreda, potrebno je stvoriti novu klasu i naslijediti funkcije iz roditeljske klase.
    b. Kotlin – nije potrebno nasljeđivanje klase, da bi se funkcija proširila potrebno je ispred naziva funkcije koristiti "." notaciju.
4. Lambda izrazi
    a. Java - ne podržava lambda izraze.
    b. Kotlin – podržava lambda izraze.

U tablici 1. prikazan je kod "Hello, World!" napisan u Javi i Kotlinu.

PRIMJER KODA U JAVI	**PRIMJER KODA U KOTLINU**

CLASS A {
PUBLIC STATIC VOID MAIN(STRING ARGS[ ]){
SYSTEM.OUT.PRINTLN("HELLO, WORLD!");
}
}
	fun main(args : Array<string>)
	{
	println("Hello, World!")
	}

HELLO, WORLD!	Hello, World!

Tablica 1. Primjer "Hello, World!" u Javi i Kotlinu

4. GENERIRANJE API KLJUČA I PRISTUP ID-U YOUTUBE KANALA

Prije korištenja aplikacije korisniku je potreban API ključ (eng. *API key*), te ID YouTube kanala nad kojim će izvršavati analizu.

4.1. Generiranje API ključa

Korisnik API ključ preuzima putem Google-ove službene stranice Google Cloud Console. Za generiranje API ključa potrebno je pratiti sljedeće korake:

1. Prijava u Google Cloud Console:
    a. Posjetite Google Cloud Console.
    b. Prijavite se sa svojim Google računom.
2. Kreiranje novog projekta:
    a. U gornjem dijelu stranice pritisnite tipku “Create Project”.
    b. Dodjelite naziv svom projektu i kliktnite na “Create”.
3. Omogućavanje YouTube Data API-ja:
    a. Nakon što ste kreirali projekt, osigurajte da je projekt odabran.
    b. Pritisnite tipku “Enable Apis And Services” ili pritisnite tipku “Library” u lijevom izborniku.
    c. U tražilicu upišite “YouTube Data API v3” i kliknite na rezultat.
    d. Nakon što se pronašli željeni API, na njemu pritisnite tipku “Enable”.
4. Kreiranje API ključa:
    a. Nakon što omogućite API, u izborniku na lijevoj strain pritisnite na “Credentials”.
    b. Pritisnite na “Create Credentials” i odaberite “API Key”.
    c. Vaš API ključ će se generirati i prikazati na ekranu.
    d. API ključ možete kopirati i spremiti.

Nakon što se generirali i preuzeli API ključ možete ga koristi u aplikaciji.

4.2. Preuzimanje ID YouTube kanala

Kako biste pristupili te preuzieli ID YouTube kanala potrebno je pratiti sljedeće korake:

1. Prijaviti se na YouTube putem računa koji upravlja kanalom.
2. U desnom gornjem kutu pritisnite na sliku vašeg kanala.
3. Otvara se padajući izbornik te odaberite “Settings”.
4. U izborniku sa lijeve strane odaberite “Advanced Settings”.
5. Otvara vam se novi prozor u kojem vam je prikazan ID kanala.
6. ID kanala možete kopirati i spremiti.

Nakon što ste preuzeli ID YouTube kanala možete ga koristi u aplikaciji.

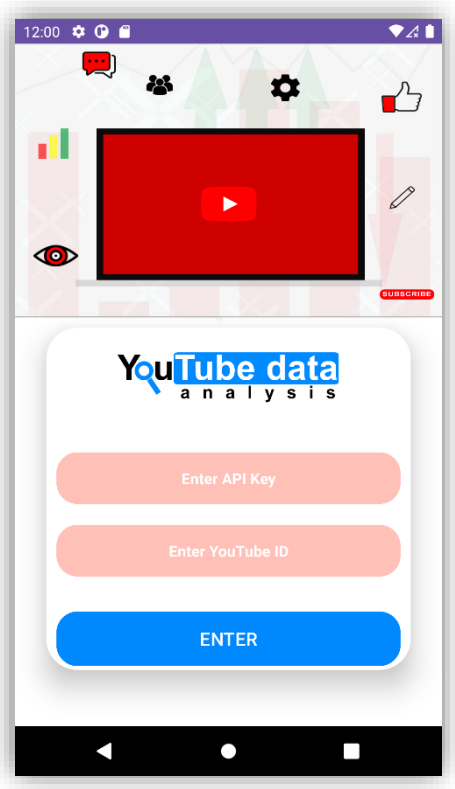
5. MOBILNA APLIKACIJA ANALIZU PODATAKA YOUTUBE KANALA

Aplikacija za analizu YouTube podataka izrađena je u razvojnom okruženju Android Studio. Aplikacija korisnicima omogućuje praćenje ključnih metrika performansi YouTube kanala. Korisnik može pristupiti broju pregleda, broju pretplatnika, komentarima. Također korisnik može pregledati rasporeda objava videozapisa po danima u tjednu kao i analizirati omjer pregleda u odnosu na vrijeme objave videozapisa.

Aplikacija se sastoji od pet glavnih aktivnosti (eng. *activity*): MainActivity, SecondActivity, UploadsActivity, AnalyticsActivity i CommentsActivity. Aktivnost AnalyticsActivity sadrži i sedam sporednih aktivnosti koji se koriste za analizu omjera pregleda u odnosu na vrijeme objave videozapisa, tj. za svaki dan u tjednu postoji posebna aktivnost koja nam prikazuje omjer broja pregleda i vrijemena objave videozapisa.

5.1. MainActivity

Pri pokretanju aplikacije prikazuje se aktivnost MainActivity prikazan na slici dolje (Slika 4.1.1.). MainActivity postavlja sučelje za unos YouTube Data API ključa i ID-a YouTube kanala. Također dohvaća podatke od YouTube API-ja te upravlja odgovorom. MainActivity sadrži i gumb "*Enter*" koji koristi API ključ i ID kanala iz korisničkog unosa, nakon čega poziva funkciju i prebacuje se na drugi ekran za prikaz dohvaćenih podataka.



Slika 5.1.1. MainActivity

MainActivity sastoji se od funkcije "*getChannelData"* koja je odgovorna za dohvaćanje podataka o YouTube kanalu. Funkcija koristi Retrofit za mrežne zahtjeve prema YouTube Data API-ju kako bi se dohvatili podatci. Retrofit je biblioteka za Android koja pretvara HTTP API u Java sučelje.

```
private fun getChannelData(apiKey:String, channelID: String){
    val retrofit = Retrofit.Builder().baseUrl("https://www.googleapis.com/youtube/v3/")
        .addConverterFactory(GsonConverterFactory.create())
        .build()
    val service = retrofit.create(YouTubeService::class.java)
    val part = "snippet,contentDetails,statistics"
    val call = service.getChannelData(part, channelID, apiKey)

    call.enqueue(object : Callback<ChannelModel> {
        override fun onResponse(call: Call<ChannelModel>, response: Response<ChannelModel>) {
            if (response.isSuccessful) {
                val api = etAPI.text.toString()
                val gson = Gson()
                val jsonContent = gson.toJson(response.body())
                val intent = Intent( packageContext: this@MainActivity, SecondActivity::class.java)
                intent.putExtra( name: "API_KEY", api)
                intent.putExtra( name: "JSON_RESPONSE",jsonContent)
                startActivity(intent)

            } else {
                val errorMessage = getString(R.string.error_message, response.code())
                resultJSON.text = errorMessage
            }
        }

        override fun onFailure(call: Call<ChannelModel>, t: Throwable) {
            val failureMessage = getString(R.string.failure_message, t.message)
            resultJSON.text = failureMessage
        }
    })
}
```

Slika 5.1.2. getChannelData

Funkcija "*getChannelData*" kreira servis "*YouTubeService*" koristeći Retrofit, koji se koristi za interakciju s YouTube API-jem. Servis sadrži jednu metodu za dohvaćanje podataka o kanalu, koja uključuje parameter u URL zahtjev. Funkcija "*getChannelData*" određuje dijelove odgovora koje API treba uključiti, a to su *snippet*, *contentDetails* i *statistics*. Ukoliko je poziv uspješno proveden, odgovor se pretvara u JSON te se pokreće "*SecondActivity*" s dohvaćenim JSON podatcima. Ukoliko poziv nije uspješan prikazuju se podatci o grešci.

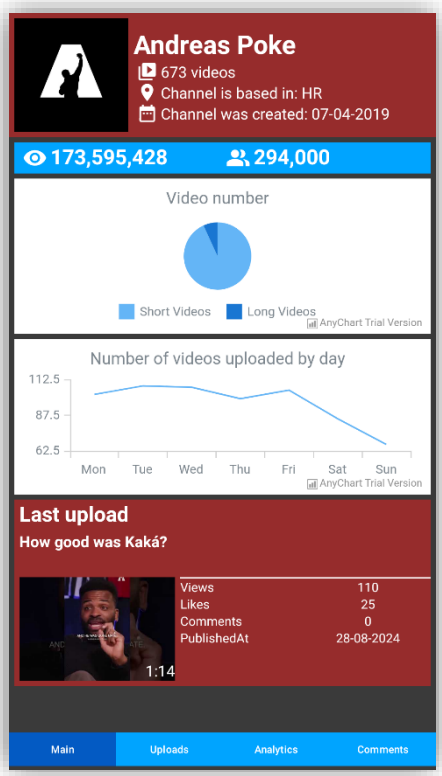
```
interface YouTubeService {
    @GET("channels")
    fun getChannelData(
        @Query("part") part: String,
        @Query("id") id: String,
        @Query("key") key: String
    ): Call<ChannelModel>
}
```

Slika 5.1.3. YouTubeService

5.2. SecondActivity

Aktivnost SecondActivity predstavlja drugu aktivnost u aplikaciji. Ova aktivnost služi za prikazivanje osnovnih detalja kanala nad kojim provodimo analitiku. Prikazuje profilnu sliku, ime kanala, broj objavljenih videozapisa, broj pretplatnika, broj pregleda, datum kreiranja kanala i državu iz koje kanal potječe. Osim osnovnih detalja, SecondActivity pruža nam uvid u omjer broja objavljenih kratkih i dugačkih videozapisa. Kratki videozapisi(eng. *Shorts*) su videozapisi koji traju 60 sekundi ili manje. Kratki videozapisi služe kako bi kreatori mogli pokrenuti svoje trendove, isprobati plesne izazove, oživjeti, podjeliti svoje ideje i još mnogo toga.

Secondactivity prikazuje broj objavljenih videozapisa po danu te detalje posljednjeg objavljenog videozapisa. Ovu aktivnost smatramo glavnom aktivnošću aplikacije jer omogućuje pristupiti svim ostalim aktivnostima.



Slika 5.2.1. SecondActivity

SecondActivity radi na način da preuzima JSON podatke koji su prosljeđeni od MainActivity i zatim ih parsira u objekt tipa ChannelModel koristeći Gson biblioteku.

ChannelModel je model koji služi za mapiranje JSON podataka iz odgovora YouTube API-a na Kotlin objekte. Klasa se sastoji od nekoliko unutarnjih klasa koje predstavljaju različite dijelove JSON odgovora.

ChannelModel klasa sadrži varijablu "*items*" koja predstavlja listu Items objekta. Polje "*items*" iz JSON odogovora mapira se na ovu varijablu i sadrži sve informacije o YouTube kanalu. Items klasa sastoji se od:

- id – string koji predstavlja jedinstveni identifikator kanala,
- snippet – objekt koji sadrži osnovne informacije o kanalu,
- contentDetails – objekt koji sadrži informacije o sadržaju kanala,
- statistics – objekt koji sadrži statističke podatke kanala.

```
data class ChannelModel (
    @SerializedName("items")
    val items: List<Items>
) {
    data class Items (
        @SerializedName("id")
        val id: String,

        @SerializedName("snippet")
        val snippet: SnippetYt,

        @SerializedName("contentDetails")
        val contentDetails: ContentDetails,

        @SerializedName("statistics")
        val statistics: StatisticsYT
    )
}
```

Slika 5.2.1. ChannelModel

SecondActivity najprije ekstraktira podatke o kanalu iz JSON odgovora. Jedan od tih podataka je i ID playliste kanala (playlistId) koji se koristi za pristup videozapisima.

Korištenjem ID-a playliste pokreće se funkcija "*getPlaylistData*" koja nam služi za dohvaćanje svih videozapisa iz playliste.

```
private fun getPlaylistData(
    youtubeApiKey: String,
    playlistId: String,
    callback: (List<PlaylistModel>?, Throwable?) -> Unit
) {
    val retrofit = Retrofit.Builder() Retrofit.Builder
        .baseUrl("https://www.googleapis.com/youtube/v3/") Retrofit.Builder
        .addConverterFactory(GsonConverterFactory.create())
        .build()

    val service = retrofit.create(YoutubePlaylistService::class.java)

    val allResponses = mutableListOf<PlaylistModel>()
    var morePages = true
    var nextPageToken: String? = ""

    service.getPlaylistItems( part: "contentDetails", playlistId, maxResults: 50, nextPageToken,
        youtubeApiKey)

    val retrofitCallback = object : Callback<PlaylistModel> {
        override fun onResponse(call: Call<PlaylistModel>, response: Response<PlaylistModel>) {
            if (response.isSuccessful) {
                val content = response.body()
                content?.let { it: PlaylistModel
                    allResponses.add(it)
                    nextPageToken = it.nextPageToken
                    morePages = nextPageToken != null
                }
                if (morePages) {
                    val callNextPage = service.getPlaylistItems(
                        part: "contentDetails",
                        playlistId,
                        maxResults: 50,
                        nextPageToken,
                        youtubeApiKey
                    )
                    callNextPage.enqueue( callback: this)
                } else {
                    callback(allResponses, null)
                }
            } else {
                val error =
                    Exception("Error retrieving data from YouTube API. Code: ${response.code()}")
                callback(null, error)
            }
        }

        override fun onFailure(call: Call<PlaylistModel>, t: Throwable) {
            callback(null, t)
        }
    }
    val call =
        service.getPlaylistItems( part: "contentDetails", playlistId, maxResults: 50, nextPageToken,
            youtubeApiKey)
    call.enqueue(retrofitCallback)
}
```

Slika 5.2.3. getPlayListData

Funkcija "*getChannelData*" koristeći Retrofit kreira servis "*YouTubePlaylistService*". Servis se sastoji od jedne metode, "*getPlaylistItems*" koja šalje HTTP GET zahtjev. Metoda ima nekoliko parametara, služe nam za precizno definiranje zahtjeva. Parametri su sljedeći:

- part – string koji predstavlja dijelove odgovora koje želimo dohvatiti,
- playlistId – string koji specificira ID playliste,
- maxResults – određuje maksimalan broj rezultata koji će biti vraćeni po zahtjevu,
- pageToken – ukoliko postoji više stranica rezultata, ovaj token koristi se za dohvaćanje iduće stranice,
- key – predstavlja API ključ koji se koristi za autentifikaciju i autorizaciju.

```
interface YoutubePlaylistService {
    @GET("playlistItems")
    fun getPlaylistItems(
        @Query("part") part: String,
        @Query("playlistId") playlistId: String,
        @Query("maxResults") maxResults: Int,
        @Query("pageToken") pageToken: String?,
        @Query("key") apiKey: String
    ): Call<PlaylistModel>
}
```

Slika 5.2.4. YouTubePlaylistService

Funkcija najprije izvršava prvi poziv za dohvaćanje, tražeći maksimalno 50 stavki po stranici. Ako je odgovor uspješan, dodaje se u odgovor "*allResponses*". Ako postoji više stranica, izvršava se novi API poziv za dohvaćanje sljedeće stranice. Kada nema više stranica, poziva se callback funkcija s prikupljenim odgovorima. Funkcija prikuplja sve podatke u listu te ih vraća putem callback funkcije.

Sljedeća funkcija koja se izvršava unutar SecondActivity je funkcija "*getAllVideoDetails*". Ova Funkcija dohvaća detalje za svaki videozapis iz liste "*videoIds*". Ukoliko je API poziv uspješan, svaki videozapis dodaje se u "*videoModelList*".

```
private fun getAllVideoDetails(
    apiKey: String,
    videoIds: List<String>,
    callback: (List<VideoModel?>) -> Unit
) {
    val retrofit = Retrofit.Builder() Retrofit.Builder
        .baseUrl("https://www.googleapis.com/youtube/v3/") Retrofit.Builder
        .addConverterFactory(GsonConverterFactory.create())
        .build()
    val service = retrofit.create(YoutubeVideosService::class.java)
    val videoModelList = mutableListOf<VideoModel?>()

    for (videoId in videoIds) {
        val call = service.getVideoDetails( part: "snippet,statistics,contentDetails", videoId, apiKey)
        call.enqueue(object : Callback<VideoModel> {
            override fun onResponse(call: Call<VideoModel>, response: Response<VideoModel>) {
                if (response.isSuccessful) {
                    videoModelList.add(response.body())
                } else {
                    println("Error retrieving data for video ID: $videoId, Code: ${response.code()}")
                    videoModelList.add(null)
                }
                if (videoModelList.size == videoIds.size) {
                    callback(videoModelList)
                }
            }
            override fun onFailure(call: Call<VideoModel>, t: Throwable) {
                println("Failure in network call for video ID: $videoId, ${t.message}")
                videoModelList.add(null)
                if (videoModelList.size == videoIds.size) {
                    callback(videoModelList)
                }
            }
            override fun onFailure(call: Call<VideoModel>, t: Throwable) {
                println("Failure in network call for video ID: $videoId, ${t.message}")
                videoModelList.add(null)
                if (videoModelList.size == videoIds.size) {
                    callback(videoModelList)
                }
            }
        })
    }
}
```

Slika 5.2.5. getAllVideoDetails

Ova funkcija služi za prikaz omjera broja objavljenih kratkih i dugih videozapisa. Prilikom poziva funkcije, za svaki videozapis provjerava se trajanje videozapisa. Za provjeru duljine videozapisa imamo dvije funkcije "*isVideoLong*" i "*isVideoShort*".

```kotlin
private fun isVideoLong(durationFormatted: String): Boolean {
    if (durationFormatted.isNotEmpty()) {
        val parts = durationFormatted.split( ...delimiters: ":")
        if (parts.size == 2) {
            val minutes = parts[0].toIntOrNull() ?: 0
            val seconds = parts[1].toIntOrNull() ?: 0
            if (minutes >= 2 && seconds >= 1) {
                return true
            }
        }
    }
    return false
}
```

Slika 5.2.6. isVideoLong

```kotlin
private fun isVideoShort(durationFormatted: String): Boolean {
    if (durationFormatted.isNotEmpty()) {
        val parts = durationFormatted.split( ...delimiters: ":")
        if (parts.size == 2) {
            val minutes = parts[0].toIntOrNull() ?: 0
            val seconds = parts[1].toIntOrNull() ?: 0
            if (minutes == 0 && seconds <= 59) {
                return true
            }
        }
    }
    return false
}
```

Slika 5.2.7. isVideoShort

Ako videozapis traje duže od jedne minute, brojač "*longVideoCount*" povećava se za 1. Ako videozapis traje kraće od jedne minute tada se brojač "*shortVideoCount*" povećava za 1. Nakon što funkcija prođe kroz sve videozapise, omjer dugih i kratkih videozapisa prikazuje se na tortnom grafičkom prikazu.

Funkcija "*getAllVideoDetails*" koristi se i za prikaz broja objavljenih videozapisa po danu. Poziva se unutar funkcije "*numberOfVideosByDay*" kako bi se provjerio dan objave svakog videozapisa.

```
private fun numberOfVideosByDay(apiKey: String, videoIds: List<String>) {
    val dayOfWeekCounter = mutableMapOf(
        "Monday" to 0,
        "Tuesday" to 0,
        "Wednesday" to 0,
        "Thursday" to 0,
        "Friday" to 0,
        "Saturday" to 0,
        "Sunday" to 0
    )

    getAllVideoDetails(apiKey, videoIds) { videoModelList ->
        for (videoModel in videoModelList) {
            videoModel?.items?.firstOrNull()?.let { item ->
                val publishedDate = item.snippet.publishedAt
                val videoDayOfWeek = convertToDayOfWeek(publishedDate)
                dayOfWeekCounter[videoDayOfWeek] =
                    dayOfWeekCounter.getOrDefault(videoDayOfWeek, defaultValue: 0) + 1
            }
        }

        val lineChartView = findViewById<AnyChartView>(R.id.line_chart_view)
        APIlib.getInstance().setActiveAnyChartView(lineChartView)

        val line = AnyChart.line()

        val data = listOf(
            ValueDataEntry( x: "Mon", dayOfWeekCounter["Monday"]),
            ValueDataEntry( x: "Tue", dayOfWeekCounter["Tuesday"]),
            ValueDataEntry( x: "Wed", dayOfWeekCounter["Wednesday"]),
            ValueDataEntry( x: "Thu", dayOfWeekCounter["Thursday"]),
            ValueDataEntry( x: "Fri", dayOfWeekCounter["Friday"]),
            ValueDataEntry( x: "Sat", dayOfWeekCounter["Saturday"]),
            ValueDataEntry( x: "Sun", dayOfWeekCounter["Sunday"])
        )

        line.title( settings: "Number of videos uploaded by day")
        line.data(data)

        line.tooltip()
            .positionMode(TooltipPositionMode.POINT)
            .anchor(Anchor.LEFT_CENTER)
            .offsetX( offset: 5)
            .offsetY( offset: 5)

        lineChartView.setChart(line)

        dayOfWeekCounter.forEach { (day, count) ->
            println("Number of videos published on $day: $count")
        }
    }
}
```

Slika 5.2.8. numberOfVideosByDay

Funkcija "*numberOfVideosByDay*" stvara mapu brojača koji označavaju broj objavljenih videozapisa u danu. Nakon toga, poziva se funkcija "*getAllVideoDetails*" i iterira kroz svaku videozapis unutar liste. Izvlači se datum objave videozapisa, koji se zatim uz pomoć funkcije "*convertToDayOfWeek*" konvertira u dan u tjednu. Nakon konverzije, ažurira se brojača za odgovarajući dan u tjednu. Rezultati, odnosno broj objavljenih videozapisa u danu prikazuje se pomoću linijskog grafa.

```
private fun convertToDayOfWeek(dateStr: String): String {
    val dateFormat = SimpleDateFormat( pattern: "yyyy-MM-dd'T'HH:mm:ss'Z'", Locale.getDefault())
    dateFormat.timeZone = TimeZone.getTimeZone( id: "UTC")
    val date = dateFormat.parse(dateStr)
    val dayFormat = SimpleDateFormat( pattern: "EEEE", Locale.getDefault())
    return dayFormat.format(date!!)
}
```

Slika 5.2.9. convertToDayOfWeek

Posljednja funkcija unutar SecondActivity je funkcija koja prikazuje detalja posljednjeg objavljenog videozapisa. Funkcija se zove "*getVideoDetails*" i koristi Retrofit za kreiranje novog servisa "*YoutubeVideosService*". Servis sadrži jednu metodu, "*getVideoDetails*" koja šalje HTTP GET zahtjev. Metoda ima tri parametra koji nam služe za precizno definiranje zahtjeva. Parametri su sljedeći:

- part – string koji predstavlja dijelove odgovora koje želimo dohvatiti,
- id – string koji specificira jedinstveni ID videozapisa,
- key – predstavlja API ključ koji se koristi za autentifikaciju i autorizaciju.

```
interface YoutubeVideosService {
    @GET("videos")
    fun getVideoDetails(
        @Query("part") part: String,
        @Query("id") id: String,
        @Query("key") key: String
    ): Call<VideoModel>
}
```

Slika 5.2.10. YouTubeVideosService

Funkcija "*getVideoDetails*" postavlja API poziv s parametrima part, id i key. Ukoliko je poziv uspješno izvršen poziva se callback funkcije s detaljima videozapisa. Funkciju pozivamo s prvim ID-om u listi, koji označava posljednji objavljeni videozapis. Nakon toga, ispisujemo i prikazujemo detalje posljednjeg objavljenog videozapisa, uključujući naslov, broj pregleda, broj lajkova, broj komentara i sliku videozapisa.

```
private fun getVideoDetails(apiKey: String, videoId: String, callback: (VideoModel?) -> Unit) {
    val retrofit = Retrofit.Builder() Retrofit.Builder
        .baseUrl("https://www.googleapis.com/youtube/v3/") Retrofit.Builder
        .addConverterFactory(GsonConverterFactory.create())
        .build()

    val service = retrofit.create(YoutubeVideosService::class.java)

    val call = service.getVideoDetails( part: "snippet,statistics,contentDetails", videoId, apiKey)
    call.enqueue(object : Callback<VideoModel> {
        override fun onResponse(call: Call<VideoModel>, response: Response<VideoModel>) {
            if (response.isSuccessful) {
                callback(response.body())
                //println("Video details successfully retrieved:")

            } else {
                val error = Exception("Error retrieving data from YouTube API. Code: ${response.code()}")
                callback(null)
                println("Error: ${error.message}")
            }
        }
        override fun onFailure(call: Call<VideoModel>, t: Throwable) {
            val error = Exception("Failure in network call: ${t.message}")
            callback(null)
            println("Error: ${error.message}")
        }
    })
}
```

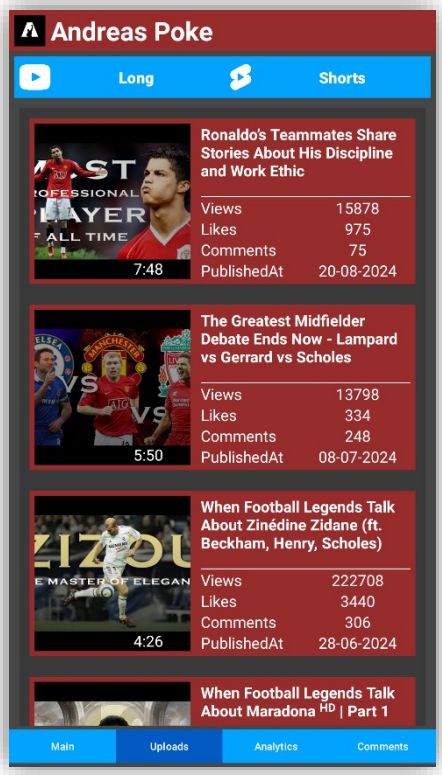
Slika 5.2.11. getVideoDetails

SecondActivity sadrži tri gumba pomoću kojih šaljemo JSON odgovor ostalim aktivnostima i možemo prijeći na iste. Aktivnosti na koje možemo prijeći su:

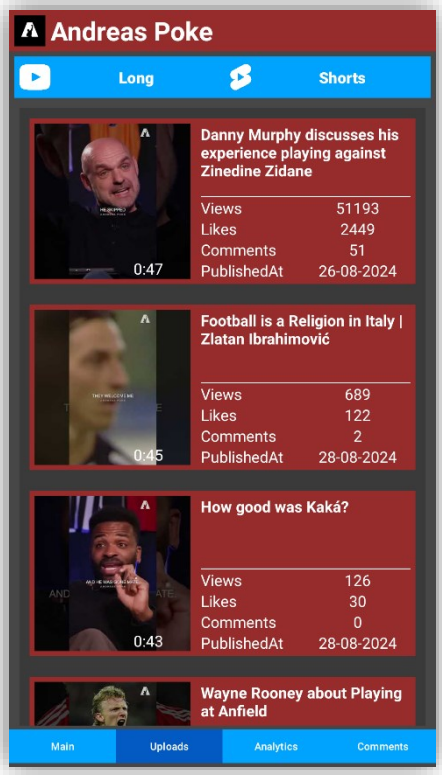
- Uploads - aktivnost za prikazivanje detalja svih objavljenih videozapisa,
- Analytics - aktivnost za prikazivanje analitike YouTube kanala,
- Comments - aktivnost za prikazivanje komentara.

5.3. Uploads Activity

UploadsActivity predstavlja aktivnost koja prikazuje detalje svih objavljenih videozapisa na YouTube kanalu. Dohvaćaju se JSON podaci, API ključ, naziv YouTube kanala, te profilna slika kanala, koji su prosljeđeni kroz Intent, te se zatim izvršava obrada i prikazivanje podataka. Objavljeni videozapisi podjeljeni su u dvije skupine: long i shorts. Long skupina videozapisa predstavlja videozapise čije je trajanje duže od jedne minute, dok shorts skupina predstavlja videozapise čije je trajanje kraće od jedne minute ili jednako jednoj minuti.



Slika 5.3.1. Long UploadsActivity



Slika 5.3.2. Shorts UploadsActivity

Prilikom pokretanja UploadsActivity, prikazuju se videozapisi iz skupine long. Pritiskom na gumb "*Shorts*" može se prijeći na kratke videozapise, isto tako pritiskom na gumb "*Long*" možemo se vratiti na videozapise duže od jedne minute.

UploadsActivity najprije koristi funkciju "*getAllVideos*", koja koristi Retrofit za dohvaćanje detalja o svakom videozapisu iz liste. Pomoću funkcije "*getAllLongVideos*" prolazi se kroz listu videozapisa, izdvajaju se videozapise duži od jedne minute te se spremaju u

zasebnu listu. Pomoću funkcije "*convertIsoToVideoDuration*" vrijeme trajanja videozapisa konvertira se iz ISO 8601 formata u format koji je čitljiv ljudima (minute i sekunde). Na primjer iz "PT2M30S" oblika u oblik 2:30.

```
fun convertIsoToVideoDuration(isoDuration: String?): String {
    isoDuration?.let { it: String
        val duration = Duration.parse(isoDuration)
        val minutes = duration.toMinutes()
        val seconds = duration.minusMinutes(minutes).seconds
        return "${minutes}:${String.format("%02d", seconds)}"
    }
    return ""
}
```

Slika 5.3.3. convertIsoToVideoDuration

Iz parsiranog objekta Duration dohvaćaju se minute, a zatim se vraža broj punih minuta. Nakon toga, iz objekta Duration oduzimaju se dohvaćene minute, a preostali dio se dohvaća koristeći metodu "*seconds*" kako bi se dobile sekunde. Minute i sekunde se formatiraju u željeni oblik (minute:sekunde), pri čemu se osigurava da su sekunde prikazane s dvije znamenke. Dobiveni format vraća se kao duljina videozapisa.

Kao i u SecondActivity, funkcije za definiranje dugih(>1min) i kratkih(≤1min) videozapise su "*isVideoLong*" i "*isVideoShort*". Funkcije primaju formatirani oblik trajanja videozapisa koji se zatim dijeli na dva dijela (minute, sekunde). Nakon što se trajanje podijeli, ti dijelovi se pretvaraju u cjelobrojne vrijednosti. Funkcija "*isVideoLong*" provjerava da li su minute veće ili jednake od 2 i jesu li sekunde veće ili jednake 1. Funkcija "*isVideoShort*" provjerava je li trajanje videozapisa manje od 1 minute.

Funkcije koje se koriste za filtiranje videozapise prema njihovoj duljini trajanja (dugi i kratki) nazivaju se "*getAllShortVideos*" i "*getAllLongVideos*".

```
private fun getAllShortVideos(apiKey: String, videoIds: List<String>, callback: (List<VideoModel?>) -> Unit) {
    getAllVideos(apiKey, videoIds) { videoModelList ->
        callback(videoModelList.filter { videoModel ->
            val durationIso = videoModel?.items?.firstOrNull()?.contentDetails?.duration
            val durationFormatted = convertIsoToVideoDuration(durationIso)
            isVideoShort(durationFormatted) ^filter
        })
    }
}
```

Slika 5.3.4. getAllShortVideos

```
@RequiresApi(Build.VERSION_CODES.O)
private fun getAllLongVideos(apiKey: String, videoIds: List<String>, callback: (List<VideoModel?>) -> Unit) {
    getAllVideos(apiKey, videoIds) { videoModelList ->
        callback(videoModelList.filter { videoModel ->
            val durationIso = videoModel?.items?.firstOrNull()?.contentDetails?.duration
            val durationFormatted = convertIsoToVideoDuration(durationIso)
            isVideoLong(durationFormatted) ^filter
        })
    }
}
```

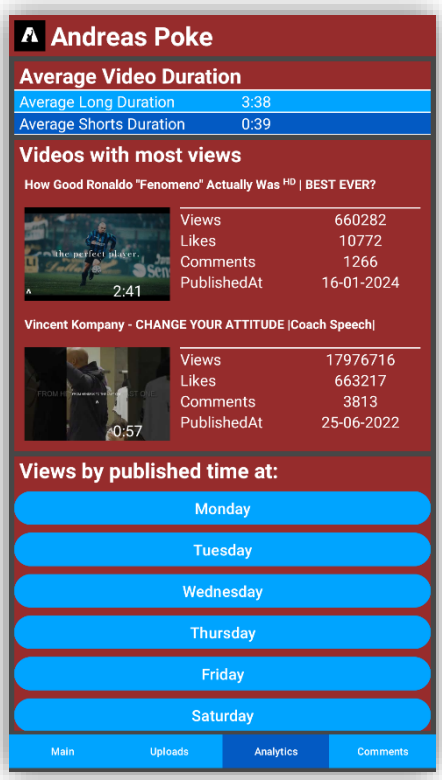
Slika 5.3.5. getAllLongVideos

Funkcija "*getAllShortVideos*" poziva funkciju "*getAllVideos*", koja vraća listu videozapisa. Dobivena lista se filtira kako bi se dobili samo kratki videozapisi. Filtiranje se provodi na način da se putem ID-eva videozapisa dohvaća duljina trajanja videozapisa. Zatim se, koristeći funkciju "*convertIsoToVideoDuration*" trajanje videozapisa konvertira u čitljivi format. Na kraju, pomoću funkcije "*isVideoShort*" provjerava se je li konvertirano trajanje videozapisa kratko. Ako jest, videozapis se zadržava u listi.

Funkcija "*getAllLongVideos*" također koristi funkciju "*getAllVideos*" za dohvaćanje svih videozapisa, a zatim koristi funkciju "*isVideoLong*" za filtriranje i dohvaćanje dugih videozapisa.

5.4. AnalyticsActivity

AnalyticsActivity predstavlja aktivnost koja se bavi analizom podataka o videozapisima s YouTube kanala. Funkcionira na način da dohvaća podatke o videozapisima, filtrira ih prema duljini, prikazuje prosječno trajanje videozapisa, prikazuje podatke o najgledanijim videozapisima te omogućava korisnicima pregledavanje statistike objavljanih videozapisa prema danima u tjednu.



Slika 5.4.1. AnalyticsActivity

AnalyticsActivity najprije pomoću funkcije "*getAllVideos*" dohvaća detalje o svim videozapisima koristeći njihove video ID-eve. Kako bi aktivnost mogla prikazati prosječno vrijeme trajanja videozapisa, potrebno je filtrirati videozapise na duge (dulje od 1 minute) i kratke (kraće od 1 minute). Za filtriranje dugih i kratkih videozapisa koriste se dvije funkcije kao i u UploadsActivity: "*getAllLongVideos*" za filtriranje dugih videozapisa i "*getAllShortVideos*" za filtriranje kratkih videozapisa.

Nakon filtriranja dugih i kratkih videozapisa, potrebno je izračunati prosječno vrijeme trajanja videozapisa. Za to se koristi funkcija "*calculateAverageDuration*".

```
private fun calculateAverageDuration(videoModelList: List<VideoModel?>): Long {
    var totalDuration = 0L
    var count = 0

    for (videoModel in videoModelList) {
        val durationIso = videoModel?.items?.firstOrNull()?.contentDetails?.duration
        totalDuration += convertIsoToSeconds(durationIso)
        count++
    }
    return if (count > 0) totalDuration / count else 0
}
```

Slika 5.4.2. calculateAverageDuration

Funkcija "*calculateAverageDuration*" prima listu videozapisa kao ulazni parameter. Funkcija se sastoji od varijable "*totalDuration*", koja sakuplja ukupno trajanje svih videozapisa u sekundama, i varijable "*count*", koja broji koliko je videozapisa obrađeno. Pomoću "*for*" petlje iteriramo kroz listu videozapisa i dohvaćamo trajanje svakog. Trajanje videozapisa je u ISO 8601 formatu, stoga se koristi funkcija "*convertIsoToSeconds*" koja pretvara trajanje u sekunde. Nakon što je trajanje pretvoreno u sekunde, provjerava se je li broj obrađenih videozapisa veći od 0, te se računa prosječno trajanje videozapisa. Za računanje prosječnog trajanja videozapisa koristi se funkcija koja dijeli ukupno trajanje videozapisa u sekundama s brojem videozapisa.

Rezultat koji vraća funkcija "*calculateAverageDuration*" konvertira se u format minuta i sekunda pomoću funkcije "*formatDurationInMinutesAndSeconds*".

```
private fun formatDurationInMinutesAndSeconds(durationInSeconds: Long): String {
    val minutes = durationInSeconds / 60
    val seconds = durationInSeconds % 60
    return "${minutes}:${String.format("%02d", seconds)}"
}
```

Slika 5.4.3. calculateAverageDuration

Funkcija "*formatDurationInMinutesAndSeconds*" prima trajanje izraženo u sekundama i dijeli ga na minute i sekunde. Minute se dobivaju tako da se trajanje podijeli s 60, dok se sekunde dobivaju korištenjem operatora "modulo" (%) koji nam vraća ostatak dijeljenja s 60. Taj ostatak predstavlja sekunde. Nakon toga, funkcija vrača format u obliku **minuta:sekunda**, pri čemu se sekunde formatiraju tako da uvijek bude prikazane kao dvocifren broj

Osim prosječnog trajanja dugih i kratkih videozapisa, AnalyticsActivity prikazuje i detalje videozapisa s najvećim brojem pregleda u svakoj kategoriji. Da bismo pronašli viedozapis najvećim brojem pregleda, prolazimo kroz listu svih videozapisa i dohvaćamo broj pregleda za svaki videozapis. Zatim se provjerava broj pregleda: ako je broj pregleda za tranutni videozapis veći od trenutne maksimalne vrijednosti, taj videozapis postavlja se kao videozapis s najvećim brojem pregleda i za njega se prikazuju detalji.

```
for (videoModel in longVideoModelList) {
    val views = videoModel?.items?.firstOrNull()?.statistics?.viewCount?.toLongOrNull() ?: 0L
    if (views > maxViews) {
        maxViews = views
        topVideoModel = videoModel
    }
}
```

Slika 5.4.4. Dio koda koji služi za dohvaćanje maksimalnog broja pregleda za dugi video

U AnalyticsActivitiu postoje 7 gumba koji služe za prikaz omjera broja pregleda videozapisa prema danu i vremenu objave. Svaki gumb predstavlja jedan dan u tjednu. Za analizu broja pregleda videozapisa na temelju dana i vremena objave koriste se tri funkcije:

- "*convertToDayOfWeek*": služi za pretvaranje datuma u dan u tjednu.
- "*convertToTime*": služi za pretvaranje datuma objave u vrijeme u danu.
- "*handleButtonClick*": glavna funkcija koja pokreće process prikazivanja omjera broja pregleda prema danu i vremenu objave videozapisa.

```
private fun convertToDayOfWeek(dateStr: String): String {
    val dateFormat = SimpleDateFormat( pattern: "yyyy-MM-dd'T'HH:mm:ss'Z'", Locale.getDefault())
    dateFormat.timeZone = TimeZone.getTimeZone( id: "UTC")
    val date = dateFormat.parse(dateStr)
    val dayFormat = SimpleDateFormat( pattern: "EEEE", Locale.getDefault())
    return dayFormat.format(date!!)
}
```

Slika 5.4.5. convertToDayOfWeek

Funkcija "*convertToDayOfWeek*" pretvara datum u dan u tjednu. Kao ulazni parameter prima datum u formatu string. Funkcija koristi format datuma "yyyy-MM-dd'T'HH:mm:ss'Z" za parsiranje i formatiranje datuma. Gdje je:

- "yyyy": godina s četiri cifre (npr. 2024).
- "MM": mjesec u godini (01 do 12).
- "dd": dan u mjesecu (01 do 31).
- "T": označava početak vremena i doslovno je slovo "T".
- "HH": sat u 24-satnom formatu (00 do 23).
- "mm": minute (00 do 59).
- "ss": sekunde (00 do 59)
- "Z": oznaka vremenske zone (Z označava UTC).

Funkcija postavlja vremensu zonu na UTC Postavljena je i vremenska zona na UTC što je važno jer se ulazni datum nalazi u UTC vremenskoj zoni. Nakon parsiranja, funkcija koristi format "EEEE" za dobivanje imena dana u tjednu. Rezultat parsiranja je objekt Date koji predstavlja određeni datum i vrijeme. Funkcija vraća ime dana u tjednu kao string. Na primjer, ako je ulazni datum "2024-06-12T12:44:33Z", funkcija će vratiti "Wednesday".

```
private fun convertToTime(dateStr: String): String {
    val dateFormat = SimpleDateFormat( pattern: "yyyy-MM-dd'T'HH:mm:ss'Z'", Locale.getDefault())
    dateFormat.timeZone = TimeZone.getTimeZone( id: "UTC")
    val date = dateFormat.parse(dateStr)
    val timeFormat = SimpleDateFormat( pattern: "HH", Locale.getDefault())
    return timeFormat.format(date!!)
}
```

Slika 5.4.6. convertToTime

Funkcija “*convertToTime*” slična je funkciji “*convertToDayOfWeek*”, ali služi za pretvaranje ulaznog datuma u 24-satni format vremena kao string. Funkcija prima datum u formatu “yyyy-MM-dd'T'HH:mm:ss'Z” i koristi format za vrijeme “HH” koji označava samo sat u 24-satnom format. Funkcija vraća samo sat kao string. Na primjer, ako je ulazni datum “2024-06-12T12:44:33Z", funkcija će vratiti "12".

```
private fun handleButtonClick(apiKey: String, videoIds: List<String>,
                              dayOfWeek: String, callback: (Map<String, List<Long>>) -> Unit) {
    getAllVideos(apiKey, videoIds) { videoModelList ->
        val timeViewsMap = mutableMapOf<String, MutableList<Long>>()
        for (videoModel in videoModelList) {
            videoModel?.items?.firstOrNull()?.let { item ->
                val publishedDate = item.snippet.publishedAt
                val videoDayOfWeek = convertToDayOfWeek(publishedDate)
                val timeOfDay = convertToTime(publishedDate)
                val views = item.statistics.viewCount.toLongOrNull() ?: 0L
                if (videoDayOfWeek == dayOfWeek) {
                    if (timeViewsMap.containsKey(timeOfDay)) {
                        timeViewsMap[timeOfDay]?.add(views)
                    } else {
                        timeViewsMap[timeOfDay] = mutableListOf(views)
                    }
                }
            }
        }
        callback(timeViewsMap)
    }
}
```

Slika 5.4.7. handleButtonClick

Funkcija “*handleButtonClick*” koristi API ključ i listu ID-eva videozapisa za dohvaćanje podatka o videozapisima, filtriranje tih podataka prema zadanom danu u tjednu i grupiranje podataka prema vremenu dana (po satu). Funkcija započinje inicijaliziranjem mape mapu za pohranu podataka, gdje su ključevi sati u 24-satnom format, a vrijednosti su liste koje predstavljaju broj pregleda videozapisa. Funkcija prolazi kroz listu videozapisa i za svaki videozapis dohvaća datum i vrijeme objave, kao i broj pregleda. Dohvaćeni datum i vrijeme pretvaraju se u dan u tjednu koristeći funkciju “*convertToDayOfWeek*”. Datum i vrijeme objave također se pretvaraju u sat kada je videozapis objavljen pomoću funkcije “*convertToTime*”. Ažuriranje mape provodi se na način da se provjerava je li dan u tjednu isti kao i dan specificiran u parametru “*dayOfWeek*”. Ako su dani isti vrši se ažuriranje mape. Provjeravaju

se ključevi i ukoliko mapa već sadrži taj ključ, tada se trenutni broj pregleda dodaje u postojeću listu pregleda za taj sat. Ukoliko mapa ne sadrži taj ključ, stvara se novi ključ, a njegova vrijednost je lista koja sadrži broj pregleda za taj sat. Na kraju, funkcija poziva povratnu funkciju s mapom podataka.

Funkcija "*handleButtonClick*" poziva se kada se pritisne gumb za svaki dan u tjednu. Funkcija prima nekoliko parametara: API ključ, lista video ID-eva, te vrijednost koja označava dan za koji želimo firtrirati podatke. Pritiskom na gumb kreira se novi Intent. Svaki dan sadrži zaseban Intent. Podatci koji se dodaju u intent uključuju:

- "*data*": JSON string koji sadrži mapirane podatke o pregledima videozapisapo satu.
- "*API_KEY*": API ključ za YouTube.
- "*title*": naslov YouTube kanala.
- "*profilePicture*": URL slike profila YouTube kanala.
- "*playlistData*": JSON string koji sadrži informacije o playlisti.

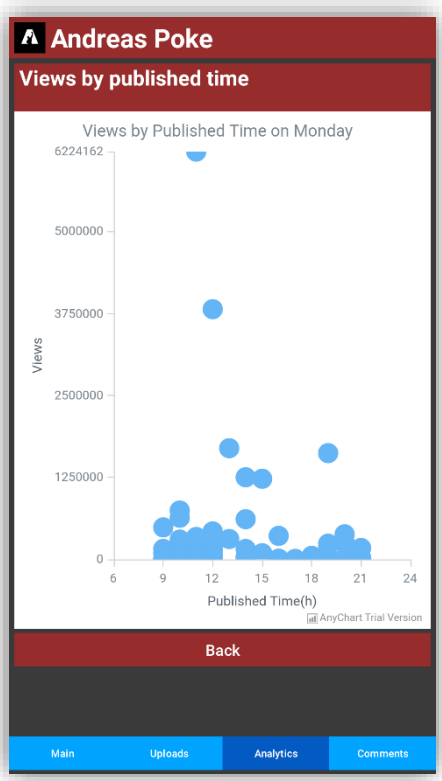
```
satbtn.setOnClickListener { it: View!
    handleButtonClick( apiKey: youtubeApiKey ?: "", videoIds, dayOfWeek: "Saturday") { data ->
        val intent = Intent( packageContext: this, SaturdayActivity::class.java)
        val dataJson = Gson().toJson(data)
        intent.putExtra( name: "data", dataJson)
        intent.putExtra( name: "API_KEY", youtubeApiKey)
        intent.putExtra( name: "title", channelTitle)
        intent.putExtra( name: "profilePicture", profilePicture)
        intent.putExtra( name: "playlistData", playlistDataJson)
        startActivity(intent)
    }
}
```

Slika 5.4.8. Dio koda koji služi za pozivanje funkcije "handleButtonClick"

Pritiskom na bilo koji od gumba: Monday, Tuesday, Wednesday, Thursday, Friday, Saturday ili Sunday, otvara se nova aktivnost koja prikazuje omjer broja pregleda prema danu i vremenu objave videozapisa.

5.4.1. MondayActivity – SundayActivity

Svaka aktivnost, nazvan po danima u tjednu (Monday, Tuesday, Wednesday, Thursday, itd.), služi za grafičko prikazivanje omjera pregleda u odnosu na vrijeme objave videozapisa. Na primjer, aktivnost MondayActivity vizualizira broj pregleda po satu objavljivanja za sve videozapise objavljene ponedjeljkom, dok ostale aktivnosti (TuesdayActivity, WednesdayActivity, itd.) pružaju slične analize za svoje odgovarajuće dane u tjednu. Aktivnosti prikazuju scatter dijagram koristeći "*AnyChart*" biblioteku. AnyChart je fleksibilno rješenje bazirano ja JavaScriptu (HTML5) koje developerima omogućuje korištenje interaktivne i vizualno atraktivne grafikone.



Slika 5.4.1.1. MondayActivity

```
private fun setupScatterChart(data: Map<String, List<Long>>) {
    val scatter = AnyChart.scatter()

    scatter.title( settings: "Views by Published Time on Monday")
    scatter.xAxis( index: 0).title( settings: "Published Time(h)")
    scatter.yAxis( index: 0).title( settings: "Views")

    val dataEntries: MutableList<DataEntry> = mutableListOf()
    for ((time, viewsList) in data) {
        for (views in viewsList) {
            dataEntries.add(ValueDataEntry(time, views))
        }
    }

    val series = scatter.marker(dataEntries)
    series.type(MarkerType.CIRCLE)
    series.size(9.0)
    series.hovered().size(7.0)
    series.hovered().fill( value: "#f48fb1")

    scatter.yScale().minimum(0)
    scatter.yScale().maximum(findMaxValue(data.values.flatten()))

    anyChartView.setChart(scatter)
}
```

Slika 5.4.1.2. setupScatterChart

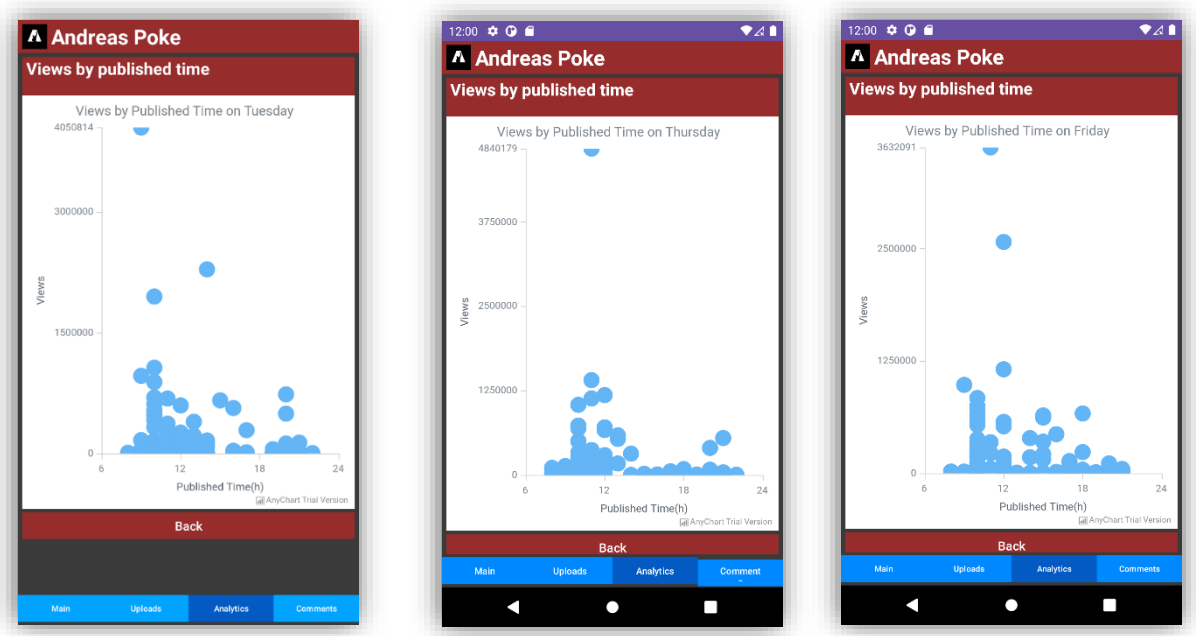
Funkcija "*setupScatterChart*" inicijalizira grafikon tipa scatter chart. Postavlja naslov grafikona te naslove x-osi i y-osi. Inicijalizira se prazna lista u koju će biti pohranjeni podaci za prikazivanje grafikona. Funkcija iterira kroz mapu koja joj je predana, gdje su ključevi vremena objavljivanja, a vrijednosti su liste brojeva pregleda. Podaci o vremenu objavljivanja i broj pregleda dodaju se u listu "*dataEntries*", te se potom oni dodaju na grafikon, kao serija markera. Markerima su konfigurirani oblik, veličina te ostala svojstva. Na y-os se definiraju minimalna i maksimalna vrijednost. Minimalna vrijednost postavlja se na 0 kako bi grafikon uvijek počinjao od 0, a za maksimalnu vrrijednost korištena je funkcija "*findMaxVlue*" koja prima listu brojeva tipa "*Long*". Ti brojevi predstavljaju broj pregleda videozapisa. Funkcija pretražuje listu te vraća najveći element u njoj. Najveća pronađena vrijednost pretvara se iz "*Long*" u "Double" korištenjem Elvis operatora "?: 0.0", koji osigurava da se pretvorba izvrši samo ako maksimalna vrijednost nije "*null*".

```
private fun findMaxValue(values: List<Long>): Double {
    return values.maxOrNull()?.toDouble() ?: 0.0
}
```

Slika 5.4.1.2. findMaxValue

Podaci korišteni za grafičko prikazivanje omjera pregleda u odnosu na vrijeme objave videozapisa prosljeđeni su iz Analitycs Activity-a.

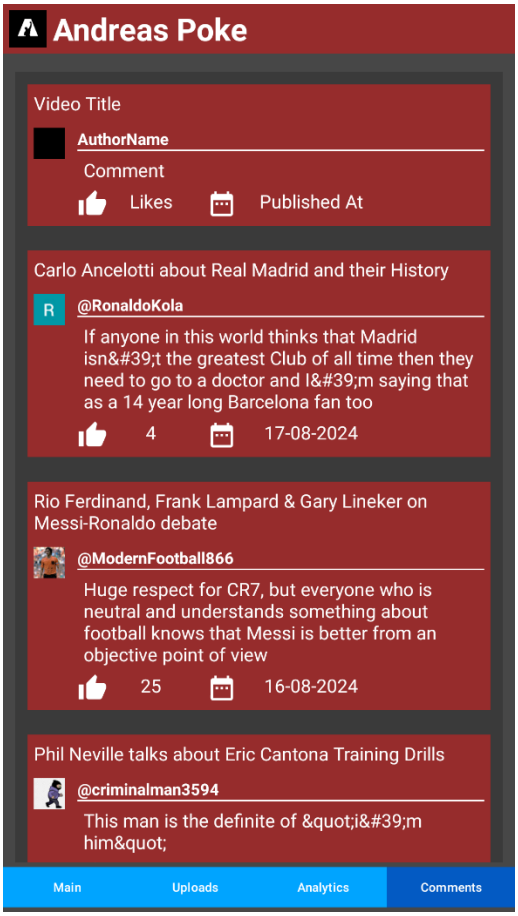
Na slici 5.4.1.1. prikazan je MondayActivity, dok na sljedećim slikama prikazani su TuesdayActivity, WednesdayActivity, ThursdayActivity, FridayActivity, SaturdayActivity i SundayActivity.



Slika 5.4.1.3. TuesdayActivity, ThursdayActivity, FridayActivity

5.5. CommentsActivity

CommentsActivity je aktivnost koja korisnicima omogućuje pregled komentara na videozapisima. Otvaranjem ove aktivnosti, korisnik dobiva uvid u sadržaj komentara, ime i profilnu sliku autora komentara, naslov videa, datum objave komentara te broj "lajkova" na komentaru.



Slika 5.5.1. CommentsActivity

Na slici 5.5.2. prikazan je primjer jednog komentar, uz jasno označene dijelove tog komentara.

Slika 5.5.2. Primjer jednog komentara

Aktivnost dohvaća ID-ove videozapisa te ih dodaje na listu, a zatim koristeći funkciju "*getVideoDetails*" dohvaća detalje o videozapisu. Nakon što funkcija "*getVideoDetails*" dohvati podatke o videozapisu poziva se funkcija "*fetchCommentsForVideos*". Funkcija "*fetchCommentsForVideos*" odgovorna je za dohvaćanje komentara za svaki videozapis.

```
private fun fetchCommentsForVideos(videoIds: List<String>, apiKey: String) {
    val retrofit = Retrofit.Builder() Retrofit.Builder
        .baseUrl("https://www.googleapis.com/youtube/v3/") Retrofit.Builder
        .addConverterFactory(GsonConverterFactory.create())
        .build()
    val service = retrofit.create(YoutubeComments::class.java)
    val container = findViewById<LinearLayout>(R.id.container_comments_layout)
    val inflater = LayoutInflater.from( context: this)

    for (videoId in videoIds) {
        val call = service.getCommentThreads( part: "snippet", videoId, apiKey)
        call.enqueue(object : Callback<CommentThreadsModel> {
            val activityContext = this@CommentsActivity
            override fun onResponse(call: Call<CommentThreadsModel>, response: Response<CommentThreadsModel>) {
                if (response.isSuccessful) {
                    val commentThreadsModel = response.body()!!
                    val commentsLayout = inflater.inflate(R.layout.comments_layout, container,  attachToRoot: false)
                    val videoTitle = commentsLayout.findViewById<TextView>(R.id.video_title)
                    val authorDisplayName = commentsLayout.findViewById<TextView>(R.id.authorDisplayName)
                    val authorProfilePicture = commentsLayout.findViewById<ImageView>(R.id.author_Profile_Picture)
                    val commenttext = commentsLayout.findViewById<TextView>(R.id.comment_text)
                    val commentlikes = commentsLayout.findViewById<TextView>(R.id.comment_like_count)
                    val commentpublishedAt = commentsLayout.findViewById<TextView>(R.id.comment_published_at)
                    for (commentThreadItem in commentThreadsModel.items) {
                        getVideoDetails(apiKey, videoId) { videoModel ->
                            if (videoModel != null) {
                                val title = videoModel.items.firstOrNull()?.snippet?.title
                                videoTitle.text = "$title"
                            }
                        }
                        val snippet = commentThreadItem.snippet.topLevelComment.snippet
                        val authorName = snippet.authorDisplayName
                        authorDisplayName.text = authorName
                        val authorPicture = snippet.authorProfileImageUrl
                        Glide.with(activityContext).load(authorPicture).into(authorProfilePicture)
                        val commentText = snippet.textDisplay
                        commenttext.text = commentText
                        val likecount = commentThreadItem.snippet.topLevelComment.snippet.likeCount
                        commentlikes.text = likecount.toString()
                        val dateFormat = SimpleDateFormat( pattern: "yyyy-MM-dd'T'HH:mm:ss'Z'", Locale.getDefault())
                        dateFormat.timeZone = TimeZone.getTimeZone( id: "UTC")
                        val publishedAt = dateFormat.parse(snippet.publishedAt)
                        val exitDateFormat = SimpleDateFormat( pattern: "dd-MM-yyyy", Locale.getDefault())
                        val datePublished = exitDateFormat.format(publishedAt!!)
                        commentpublishedAt.text = datePublished
                    }
                    container.addView(commentsLayout)
                } else {
                    println("Error: ${response.code()}")
                }
            }
            override fun onFailure(call: Call<CommentThreadsModel>, t: Throwable) {
                println("Failure: ${t.message}")
            }
        })
}
```

Slika 5.5.3. fetchCommentsForVideos

Funkcija "*fetchCommentsForVideos*" koristi Retrofit za kreiranje servis pod nazivom "*YoutubeComments*". Ovaj servis sastoji se od jedne funkcije "*getCommentThreads*", koja šalje HTTP GET zahtjev za dohvaćanje komentara. Funkcija koristi nekoliko parametara (part, videoId, key) koji nam služe za precizno definiranje zahtjeva.

- part – string koji predstavlja dijelove odgovora koje želimo dohvatiti,
- videoId – string koji specificira ID videozapisa,
- key – predstavlja API ključ koji se koristi za autentifikaciju i autorizaciju.

```
interface YoutubeComments {
    @GET("commentThreads")
    fun getCommentThreads(
        @Query("part") part: String,
        @Query("videoId") videoId: String,
        @Query("key") apiKey: String
    ): Call<CommentThreadsModel>
}
```

Slika 5.5.4. YoutubeComments

Odgovor API-ja dolazi u obliku JSON objekta koji sadrži informacije o komentarima. Retrofit koristi Gson konverter za mapiranje JSON objekta na model pod nazivom "*commentThreadsModel*".

"*CommentThreadsModel*" koristi se za parsiranje odgovora koji sadrži informacije o komentarima videozapisa. Ovaj model sadrži nekoliko klasa:

1. CommentThreadsModel – glavna klasa koja sadrži listu objekata tipa "*CommentThreadItem*". Klasa odgovara JSON polju "*items*" što je lista "*CommentThreadiItem*" koja predstavlja pojedinačne niti komentara.
2. CommentThreadItem – klasa koja predstavlja pojedinačnu nit komentara. Uključuje podatke kao što su "*snippet*", koji sadrži informacije za pojedinačne komentare, uključujući "*videoID*" i "*topLevelComment*".
3. TopLevelComment – klasa koja sadrži "*snippet*" s detaljima o samom komentaru, kao što su autor, tekst, broj lajkova i datum objave te URL autorove profilne slike.

Slika 5.5.5. CommentThreadsModel

Aplikacija kreira kontejner u koji će se dinamički ubacivati komentari. Prolazeći kroz video ID-jeve, aplikacija za svaki ID kreira API poziv. Ukoliko je poziv uspješan, dobiva se objekt "*CommentThreadsModel*" koji sadrži komentare. Novi layout se dinamički ubacuje u prethodno pripremljeni kontejner. Pomoću for petlje, aplikacija za svaki komentar, dohvaća

podatke o videozapisu teih postavlja u “*commentsLayout*”. Nakon toga, “*commentsLayout*” se dodaje u kontejner kako bi komentari bili prikazani u korisničkom sučelju.

7. ZAKLJUČAK

U ovom diplomskom radu izrađena je mobilna aplikacija za provođenje analize podataka YouTube-a. Cilj je bio izraditi aplikaciju koja kreatorima YouTube sadržaja omogućuje detaljan uvid u analitiku podataka objavljenih videozapisa. Aplikacija je osmišljena tako da pomogne kreatorima u donošenju odluka koje će doprinjeti daljnjem razvoju kanala.

Korištenje navedenih funkcionalnosti pokazalo se kao odlično sredstvo za dublje uvide u ponašanje gledatelja, pomažući tako kreatorima u da bolje razumiju svoju publiku. Kreatori mogu analizirati performance svojih sadržaja i prilagoditi svoje strategije (vrijeme objavljivanja videozapisa, naslov videozapisa, duljina videozapisa, itd.) kako bi postigli bolje rezultate.

Aplikacija ima prostora za napredak i daljni razvoj. Neke od mogućih funkcionalnosti koje bi se mogle dodati su sljedeće: prilikom detaljnog uvida u analitiku svakog videozapisa moguće je dodati gumb putem kojeg bi korisnik mogao izravno pristupiti i pogledati videozapis na YouTube-u. Također,korisnicima bi se mogao omogućiti odgovor na komentar. Moguće je dodati i pregled odgovora na komentare koje su korisnici objavili. Aplikaciju bi se moglo proširiti dodavanjem funkcionalnosti za prikaz broja pregleda i pretplatnika dobivenih u određenom vremenskom period (npr. 7 dana, 30 dana, 6 mjeseci, itd.). Web verzija ove aplikacije značajno bi povećala njezinu dostupnost i korisnost.

Jedan od trenutnai nedostatak aplikacija je što ne omogućuje pamćenje ID-a te API ključa prilikom ponovnog ulaska u aplikaciju, ali to također znači da aplikacija ne zahtjeva registraciju korisnika. Drugi veći nedostatak je što je aplikacija izrađena isključivo za Android platformu. Navedeni nedostaci mogu se riješiti u budućim verzijama aplikacije.

8. LITERATURA

- Bill Phillips, Chris Stewart, Brian Hardy & Kristin Marsicano (2015). Android Programming - The Big Nerd Ranch Guide. Dostupno na: https://aaronyeo.org/books_/Android/Android%20Programming%20%20The%20Big%20Nerd%20Ranch%20Guide.pdf , [19.05.2024.]
- Josh Skeen and David Greenhalgh (2018.). Kotlin Programming: The Big Nerd Ranch Guide. Dostupn na: https://ptgmedia.pearsoncmg.com/images/9780135161630/samplepages/9780135161630_Sample.pdf , [19.05.2024.]
- Antonio Leiva (2017.). Kotlin for Android Developers. Dostupno na: https://moodle.znu.edu.ua/pluginfile.php/253068/mod_resource/content/1/antonio_leiva_kotlin_for_android_developers.pdf , [19.05.2024.]
- Learn Kotlin for Android: https://developer.android.com/courses/android-basics-compose/course , [20.05.2024.]
- Google Cloud Console: https://console.cloud.google.com/welcome?project=youtube-data-analysis-406908, [20.05.2024.]
- Google for developers: https://developers.google.com/youtube/v3/docs, [21.05.2024.]
- YouTube studio: https://support.google.com/youtube/answer/7548152?hl=en&co=GENIE.Platform%3DAndroid, [18.05.2024.]
- Retrofit: https://square.github.io/retrofit/ , [05.06.2024.]
- AndroidStudio: https://developer.android.com/studio/intro, [05.06.2024.]
- *JAVA: https://www.w3schools.com/java/java_intro.asp*, [05.06.2024.]
- *JAVA: https://www.ibm.com/topics/java*, [05.06.2024.]
- Kotlin: https://www.imaginarycloud.com/blog/kotlin-vs-java, [05.06.2024.]
- Channel: https://developers.google.com/youtube/v3/docs/channels , [05.08.2024.]
- PlaylistItems: https://developers.google.com/youtube/v3/docs/playlistItems , [06.08.2024.]
- Videos: https://developers.google.com/youtube/v3/docs/videos , [07.08.2024.]
- Comments: https://developers.google.com/youtube/v3/docs/comments , [10.08.2024.]

- Comments: https://developers.google.com/youtube/v3/docs/comments/list , [10.08.2024.]
- Anychart: https://www.anychart.com/, [18.07.2024.]