SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 913

Android aplikacija za upravljanje glazbenim profilom

Borna Nikolić

Zagreb, lipanj 2023.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 10. ožujka 2023.

ZAVRŠNI ZADATAK br. 913

Pristupnik:	**Borna Nikolić (0036534977)**
Studij:	Elektrotehnika i informacijska tehnologija i Računarstvo
Modul:	Računarstvo
Mentor:	prof. dr. sc. Igor Mekterović

Zadatak: **Android aplikacija za upravljanje glazbenim profilom**

Opis zadatka:

Na svijetu postoje mnogi glazbeni servisi i aplikacije koje omogućuju slušanje glazbe putem računala poput Spotifyja, Deezera, itd. Također, postoje i niz otvorenih aplikacijskih sučelja za strojni dohvat podataka iz glazbene domene (npr. Spotify Web API, ShazamKit, Last.fm API, itd.). Ako se svi ti izvori podataka uzmu u obzir, na raspolaganju je velik broj informacija koji bi se mogao povezati i upotrijebiti kako bi se korisniku pružila dodatna vrijednost u vidu informacija o glazbi koji sluša i prati kao i za preporuke za otkrivanje novih izvođača i pjesama. Potrebno je napraviti mobilnu aplikaciju za Android operacijski sustav koristeći programski jezik Kotlin koja će omogućiti upravljanje glazbenim profilom. Profil treba dominantno izgraditi na temelju podataka iz Spotify aplikacije, putem Spotify Wep APIja. Omogućiti pregled najslušanijih izvođača i pjesama, te različite ostale statistike koje se mogu pribaviti putem APIja. Podatke kombinirati s podatcima koje nude drugi izvori (npr. Last.fm) i ostvariti preporuke sličnih izvođača i pjesama po različitim kriterijima (sličnost, žanr, itd.). Dodatno, omogućiti prepoznavanje pjesama na temelju audio značajki (npr. ShazamKit) i ad hoc preporuke na temelju prepoznate pjesme. Donijeti ocjenu ostvarenog pristupa te smjernice za budući razvoj.

Rok za predaju rada: 9. lipnja 2023.

SADRŽAJ

1. Uvod

Razvojem Interneta korisnicima se pruža sve više različitih glazbenih servisa i aplikacija koje omogućuju slušanje glazbe. Neki od popularnijih takvih aplikacija su Spotify, Deezer, Apple Music i mnogi drugi. Kasnije su ovakvi servisi razvili i razna aplikacijska programska sučelja otvorenog koda *(eng. open source)* pomoću kojih je moguće pristupi raznim podacima. Kombiniranjem različitih izvora podataka otvaraju se mogućnosti za pružanje dodatne vrijednosti korisnicima kao što su informacije o glazbi koju slušaju, praćenje njihovih sklonosti, kao i predlaganje novih pjesama i izvođača.

Cilj ovog rada je razvijanje mobilne aplikacije za Android operacijski sustav koristeći programski jezik Kotlin. Aplikacija treba omogućiti upravljanje glazbenim profilom primarno na temelju Spotify Web API-ja. Ključni element profila jest pregled najslušanijih izvođača i pjesama. Kako bi se korisniku pružilo personalizirano iskustvo podaci iz Spotify Web API-ja **[3]** u kombinaciji s podacima koje pruža Last.fm API **[14]** će se iskoristiti za generiranje glazbenih preporuka. Još jedna ključna značajka aplikacije bit će mogućnost prepoznavanja pjesama na temelju audio značajki i Shazam API-ja **[4]**. Ta funkcionalnost omogućuje korisniku da identificira nepoznatu pjesmu i pronađe više informacija o njoj ili da ju doda u svoju glazbenu biblioteku.

Ocjena ostvarenog pristupa uvelike će ovisiti o kvaliteti implementacije mobilne aplikacije i sposobnosti aplikacije da pruži relevantne i točne informacije o glazbi. U konačnici, mobilna aplikacija za upravljanje glazbenim profilom pruža korisnicima priliku da istraže i uživaju u glazbi na temelju njihovih preferencija. Integracija različitih izvora podataka omogućuje stvaranje bogatog glazbenog iskustva i pružanje preporuka.

2. Kotlin

2.1. Općenito

Kotlin je moderni programski jezik koji se najčešće koristi za razvoj aplikacija namijenjenih operacijskom sustavu Android. Dizajniran je s namjerom da bude izražajan, siguran i interoperabilan s već postojećim Java kodom. Razvila ga je tvrtka JetBrains. Kotlin je predstavljen javnosti 2011. godine te je nešto kasnije postao i službeni jezik za razvoj Android aplikacija.

Jedna od glavnih karakteristika ovog programskog jezika jest već spomenuta interoperabilnost s programskim jezikom Java. To znači da Kotlin može koristiti već postojeće biblioteke i radne okvire namijenjene programskom jeziku Java. Tako je olakšan prijelaz s Jave na Kotlin i omogućuje programerima iskorištavanje prednosti oba programska jezika.

Već u samom dizajnu razmišljalo se o sigurnosti razvijenih aplikacija. Iz tog razloga dizajn Kotlina je takav da pomaže programerima u izbjegavanju grešaka koje vode do ranjivosti u implementaciji te u konačnosti nesigurnog sustava. Pruža se i mogućnosti *null* tipa i provjeravanja *null* vrijednosti u vremenu izvođenja što opet pomaže u smanjenju grešaka vezanih uz *null* reference.

Kotlin pruža i mnoga druga moderna rješenja jezika koje olakšavaju razvoj programske podrške. To između ostalog uključuje i lambda izraze, proširivanje funkcionalnosti klasa, raspoznavanje uzoraka i mnogobrojne kolekcije. Ove značajke programskog jezika pomažu u čitljivosti samog koda. Ukratko, Kotlin je programski jezik koji kombinira najbolje značajke programskog jezika Jave s modernim rješenjima, sigurnošću i izražajnošću. Upotreba Kotlina u Android razvoju, ali i drugim područjima programskog inženjerstva kontinuirano raste. Tome sigurno pomažu i pozitivna iskustva korisnika. Dodatne informacije o ovom programskom jeziku moguće je pronaći na JetBrains developer blogu **[1]**, Google Android Developers web stranici **[6]** i samoj Kotlin dokumentaciji **[2]**. Kroz sljedeća potpoglavlja bit će pojašnjeni osnovni koncepti razvoja Android aplikacija u kontekstu programskog jezika Kotlin.

2.2. Životni ciklus aktivnosti

Aktivnost (*eng. activity*) je osnovna komponenta Kotlin Android aplikacije. Ona predstavlja jedan zaslon s pripadajućim korisničkim sučeljem koji služi kao ulazna točka za interakciju korisnika s aplikacijom. Ujedno aktivnost sadrži i samu logiku pomoću koje se upravlja korisničkim interakcijama, elementima samog sučelja te koordinacijom drugih komponenata aplikacije (kao što su na primjer fragmenti). Sam izgled aktivnosti definira se u odgovarajućoj XML datoteci koja je povezana s aktivnošću.

Aktivnost ima životni ciklus koji se sastoji od različitih stanja i povratnih poziva *(eng. callback)* poziva. Razumijevanje ovog ciklusa je važno kako bi programer mogao na odgovarajući način raspodijeliti računalne resurse **[9]**. Aktivnost može biti u jednom od sljedećih stanja:

1. **Stvoreno (*Created*)** - Inicijalno stanje nakon stvaranja aktivnosti. Tijekom ovog stanja poziva se *callback* metoda *onCreate()*. U toj metodi obavlja se inicijalizacija aktivnosti što podrazumijeva inflaciju određenog izgleda *(eng. layout)*, povezivanja pogleda i postavljanje aktivnih i pasivnih slušača.

2. **Pokrenuto (*Started*)** - U ovom stanju aktivnost jest vidljiva korisniku, ali nije u prvom planu. Drugim riječima moguća je situacija u kojoj neka druga aktivnosti ili sistemski dijalog zasjenjuju aktivnost. Ulaskom u ovo stanje poziva se *callback* metoda *onStart()*. U ovoj metodi moguće je stvoriti animacije i dohvaćati resurse potrebne za prikaz same aktivnosti.

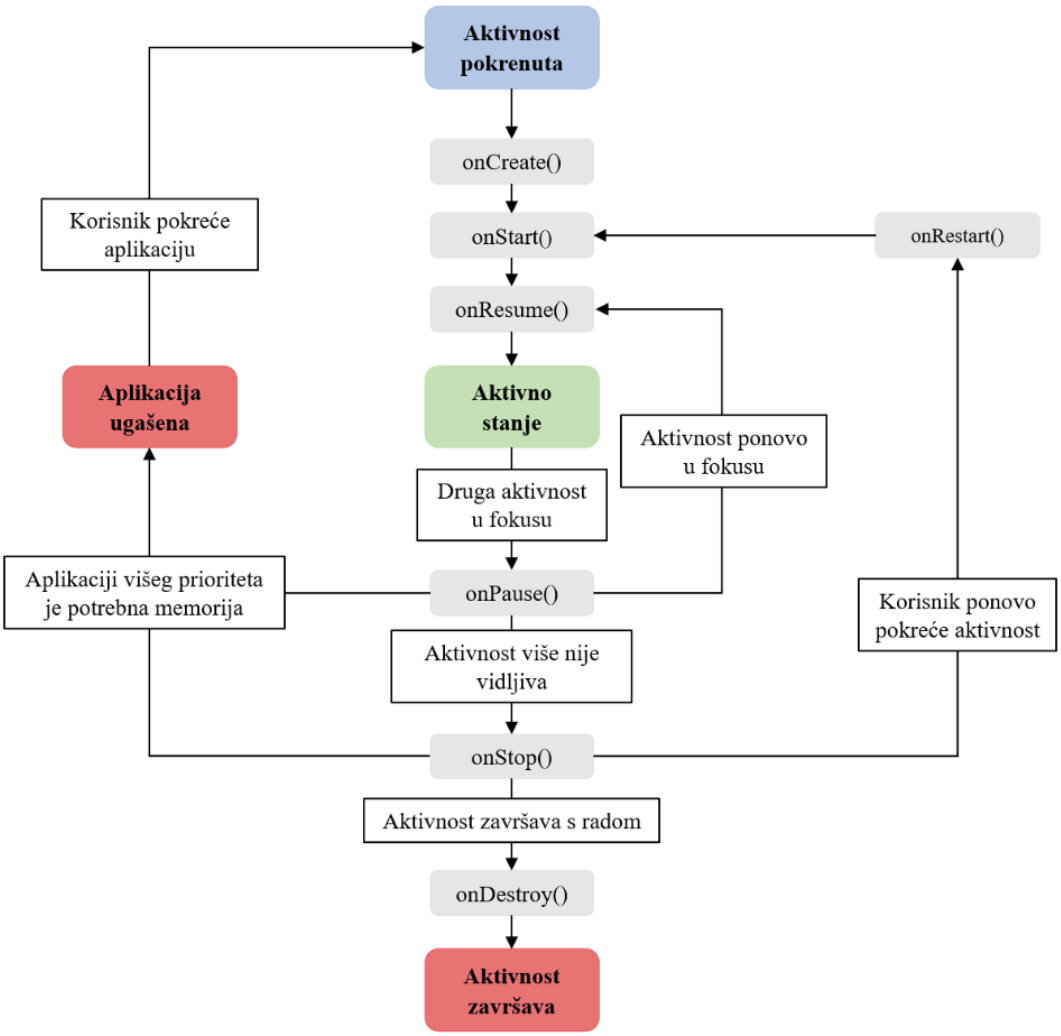
3. **Aktivno (*Resumed*)** - Kada se aktivnost nalazi u ovom stanju ona je u prvom planu, tj. niti jedna druga aktivnost ju ne zasjenjuje. Korisnik s ovom aktivnošću direktno interagira. *Callback* metoda koja se poziva ulaskom u ovo stanje jest *onResume()*. U ovoj metodi se trebaju nastaviti ili pokrenuti sve operacije koje su aktivne tijekom korisnikove interakcije s aktivnošću. Primjeri takvih operacija su: pokretanje animacija, reprodukcija zvuka ili osvježavanje podataka.

4. **Pauzirano (*Paused*)** - Ulaskom u ovo stanje aktivnost je djelomično vidljiva, no nije u korisnikovom fokusu. *Callback* metoda koja se poziva ulaskom u ovo stanje jest metoda *onPause()*. U ovoj metodi preporučuje se oslobađanje svih resursa koji nisu potrebni za izvršavanje aktivnosti u pozadini.

5. **Zaustavljeno (*Stopped*)** - U ovom stanju aktivnost više nije vidljiva korisniku. Poziva se metoda *onStop()*. U ovoj metodi trebali bi se osloboditi svi resursi koji

su potrebni aktivnosti dok je vidljiva. U ovo spadaju pozadinske dretve ili pak trajni podaci koje je potrebno spremiti.

6. **Uništeno (*Destroyed*)** - Dolaskom u ovo stanje aktivnost se uništava ili zbog završetka korisničkog rada ili zbog oduzimanja resursa od strane samog operacijskog sustava. Poziva se *callback* metoda *onDestroy()*. U ovoj metodi se trebaju osloboditi svi resursi, odjaviti slušači i izvršiti završne operacije prije uklanjanja aktivnosti iz same radne memorije.

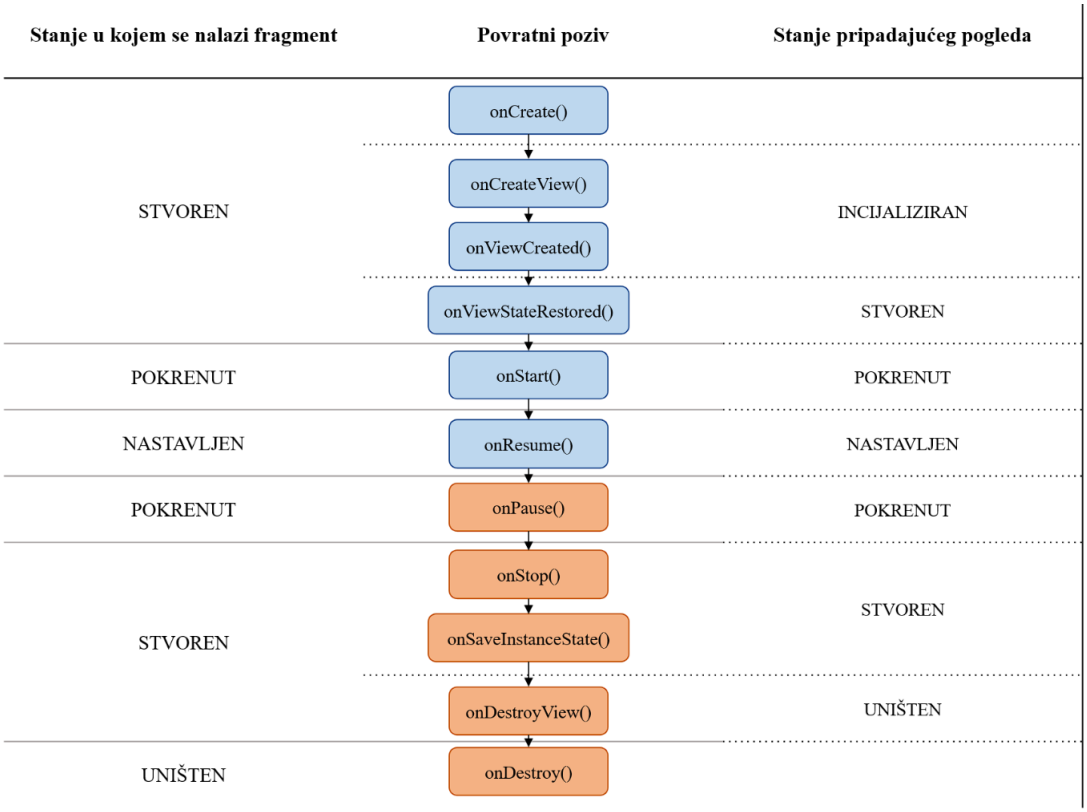


Slika 2.1: Prikaz stanja aktivnosti kroz životni ciklus te okidača koji uzrokuju prijelaze između stanja **[9]**

2.3. Fragmenti i pogledi

Slično aktivnostima u Kotlinu postoje fragmenti. To su komponente namijenjene višestrukoj uporabi. Fragmenti predstavljaju jedan dio korisničkog sučelja te se mogu kombinirati s drugim fragmentima u pojedinoj aktivnosti kako bi stvorili cjelovito sučelje. Slično kao i aktivnosti, fragmenti imaju svoj životni ciklus te su osnovna gradivna komponenta pogleda *(eng. view)*. Pomoću pogleda moguće je efektivno obraditi događaje pokrenute korisnikovim akcijama. Na **slici 2.2.** moguće je vidjeti stanja kroz koje prolazi fragment u svom životnom ciklusu i stanja kroz koje prolazi pogled kroz životni ciklus. Stanja životnog ciklusa fragmenta i pogleda slična su stanjima životnog ciklusa aktivnosti **[7]**. Nova stanja koja se pojavljuju u ovom životnom ciklusu su:

- **Povezano *(eng. attached)*** - Fragment je uspješno povezan s aktivnošću, ali još nije u potpunosti inicijaliziran. Poziva se *callback* metoda *onAttach()* pri ulasku u ovo stanje. U ovoj metodi izvršavaju se inicijalizacijski zadaci koji zahtijevaju pristup aktivnosti čiji je fragment dio. Takvi zadaci uključuju inicijalizaciju sučelja te dohvaćanje referenci na povratne pozive aktivnosti.
- **Prikaz stvoren *(eng. view created)*** - Zadana hijerarhija prikaza fragmenta jest stvorena. Metoda koja se zove ulaskom u ovo stanje jest *onCreateView()*. Ovdje se inicijalizira izgled fragmenta *(eng. inflate layout)*, povezuju se različiti pogledi i postavljaju se slušači događaja. Ovdje se može pristupiti i upravljati pogledima fragmenta.
- **Prikaz uništen *(eng. view destroyed)*** - Prikaz fragmenta se uništava. Poziva se metoda *onDestroyView()*. Ovdje je potrebno osloboditi sve resurse koje je pogled fragmenta koristio kao što su odjavljivanje slušača.
- **Fragment odvojen *(eng. detached)*** - Fragment je odvojen od roditeljske aktivnosti. Poziva se *callback* metoda *onDetach()*. Ovdje se izvršavaju završni zadaci fragmenta te je moguće resetirati roditeljsku aktivnosti i druge komponente te aktivnosti.



Slika 2.2: Prikaz životnog ciklusa fragmenta i pogleda **[7]**

Sam izgled pogleda i fragmenta se kao i izgled aktivnosti definira pomoću XML datoteka.

2.4. Zajedničke postavke *(eng. shared preferences)*

Zajedničke postavke u Kotlinu omogućuju trajno spremanje manje kolekcije parova ključ - vrijednost. Ovaj mehanizam koristan je kada tim podacima treba pristupiti brzo i jednostavno tijekom više sjednica aplikacije. U implementaciji ovaj mehanizam ćemo koristiti kako bismo spremili podatke potrebne za pristup aplikacijskim programskim sučeljima. Važno je napomenuti kako se jednom spremljeni par ključ - vrijednost može promijeniti.

Kako bismo podatke spremili u zajedničke postavke najprije je potrebno dohvatiti referencu na objekt *SharedPreferences*. Ovu referencu moguće je dohvatiti pozivom metode *getSharedPreferences()* nad trenutnim kontekstom aplikacije. Pisanje u zajedničke postavke omogućava se pozivom funkcije *edit()* nad dobivenom referencom. Nad dobivenim editorom moguće je pozivati metode *putX()* gdje X predstavlja neki jednostavni tip podatka u Kotlinu (npr. string ili cijeli broj). Kao parametre ove funk-

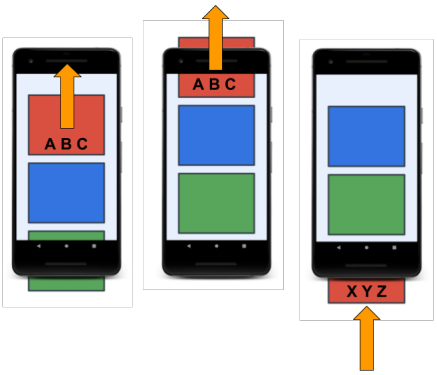
cije potrebno je redom dati ključ koji mora biti tipa string i vrijednost specificiranog tipa. Nakon upisanih promjena u postavke potrebno je pozvati *apply()* ili *commit()* kako bi se promjene trajno spremile.

Čitanje iz zajedničkih postavki obavlja se tako da se nad referencom zajedničkih postavki zove metoda *getX()* gdje X predstavlja tip podatka koji se želi dohvatiti. Kao argumente ove funkcije potrebno je zadati ključ čiju vrijednost treba dohvatiti i zadanu vrijednost koju funkcija vraća ako traženi ključ ne postoji.

Funkcijom *putX()* moguće je i osvježiti vrijednost određenog ključa. Brisanje zapisa iz zajedničkih postavki obavlja se pozivanjem funkcije *remove()* kojoj se kao argument predaje ključ vrijednosti koja se želi izbrisati **[11]**.

2.5. Recycler View

Recycler View je komponenta koja efektivno i relativno jednostavno prikazuje velike skupove podataka. Odgovornost programera pri korištenju ove komponente jest dohvat podataka i predaja podataka komponenti te definiranje izgleda pojedinog elementa u Recycler Viewu. To se vrlo lako može definirati pomoću XML datoteke. Kao što i samo ime sugerira Recycler View *"reciklira"* individualne elemente. Kad pogled elementa Recycler Viewa pri pregledavanju dođe do dna korisničkog prikaza Recycler View ne uništava taj pogled, već ga ponovo koristi kako bi prikazao nove elemente koji se tek trebaju prikazati. Tako se povećava učinkovitost i responzivnost same aplikacije **[10]**.



Slika 2.3: Prikaz rada komponente Recycler View **[8]**

Koraci implementacije ove komponente su sljedeći:

1. Definiranje dijela fragmenta, odnosno aktivnosti u kojem će se prikazivati lista implementirana Recycler Viewom. To se jednostavno može napraviti u XML

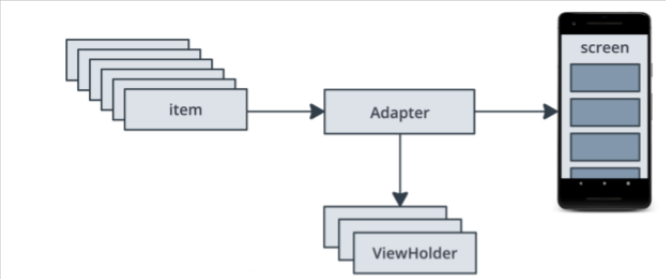
datoteci aktivnosti, odnosno fragmenta.

2. Zatim treba odrediti kako će prikazana lista izgledati. Ovdje se mogu koristiti unaprijed definirani upravitelji prikaza *(eng. Layout Managers)*. U implementaciji ovog rada korištena je horizontalna i vertikalna inačica dostupnog Linear Layout Managera.

3. Sljedeći korak u implementaciji jest definirati XML datoteku u kojoj se opisuje kako svaki element liste izgleda i na koji način se ponaša.

4. Na temelju tog dizajna potrebno je naslijediti View Holder klasu u kojoj definiramo sve funkcionalnosti za pojedini element liste. Ova klasa će se omotati klasom View te će tom klasom dalje upravljati Recycler View komponenta.

5. Posljednji korak implementacije jest definirati adapter klasu pomoću koje se predani podaci preslikavaju u pogled definiran XML datotekom i klasom View Holder.



Slika 2.4: Shematski prikaz implementacije komponente Recylcer View **[8]**

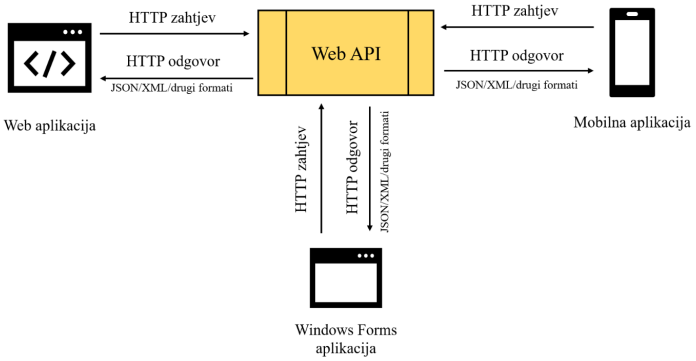
3. Aplikacijska programska sučelja

Aplikacijsko programsko sučelje *(eng. application programming interface - API)* je definiran skup pravila kojim se omogućuje komunikacija između različitih aplikacija. Pomoću programskog sučelja definiraju se metode, formati podataka koji se koriste u prijenosu i različite konvencije koje omogućuju programerima pristup određenoj funkcionalnosti neke komponente, servisa ili platforme. Tako se omogućuje integracija vanjskih sustava ili biblioteka u nove aplikacije koje nisu nužno u vlasništvu programera.

Postoje različiti oblici aplikacijskih programskih sučelja. Jedan od najčešćih jest Web API kojim aplikacije komuniciraju preko Interneta koristeći protokole http i https. Pristup resursima kroz API je tada omogućen preko skupine URL adresa i krajnjih točaka komunikacije.

Aplikacijska programska sučelja nude širok spektar funkcionalnosti, kao što su dohvat podataka, slanje podataka, provedba različitih operacija, procesiranje rezultata i mnoge druge. Najčešće se koriste u razvoju mobilne programske podrške i razvoju programske podrške za web **[12]**.

U sklopu ovog rada korištena su tri različita aplikacijska programska sučelja. U sljedećim potpoglavljima ukratko su objašnjene funkcionalnosti koje nudi svako od njih.



Slika 3.1: Shematski prikaz arhitekture Web API sučelja **[17]**

3.1. Spotify Web API

3.1.1. Općenito

Spotify Web API je API koji omogućava popularan servis za slušanje glazbe - Spotify. Pomoću ovog programskog sučelja programeri mogu dohvatiti i interagirati s različitim značajkama i podacima dostupnima na Spotify platformi. Koristeći mogućnosti Spotify Web API-ja moguće je pristupiti podacima koji su specifični za pojedinog korisnika i na temelju tih podataka stvoriti glazbeni profil, generirati različite preporuke i omogućiti dodavanje tih preporuka u korisnikovu kolekciju pjesama **[3]**. Kroz ovo potpoglavlje bit će objašnjeno kako autorizirati aplikaciju i dobiti odgovarajuće korisničke dozvole potrebne za dohvaćanje željenih podataka te na kojim krajnjim točkama je moguće dohvatiti željene podatke.

3.1.2. Autorizacija

Spotify Web API pruža četiri vrste autentifikacije ovisno o sustavu koji se nastoji izgraditi. Budući da je cilj ovog rada izgraditi mobilnu aplikaciju koja bi čim dulje imala pristup željenim podacima iskorišten je *Code Flow* način autorizacije **[5]**. Ovaj način autorizacije izvodi se u sljedećih nekoliko koraka:

1. **Registracija aplikacije** - Prije samog korištenja Spotify Web API sučelja, potrebno je aplikaciju registrirati na Spotify web stranici. Potrebno je kreirati *Spotify Developer* račun i specificirati osnovne informacije o samoj aplikaciji kao što su ime aplikacije, kratak opis funkcionalnosti te povratni poziv aplikacije koji će Spotify Web API pozvati nakon uspješne autorizacije.

2. **Dohvat klijentskih vjerodajnica** - Nakon uspješne registracije aplikacije Spotify aplikaciji pridodaje jedinstveni *Client ID* i *Client Secret*. Ove vjerodajnice bit će potrebne za dobivanje klijentskih dozvola za obavljanje API poziva.

3. **Autorizacija korisnika** - Kako bi aplikacija mogla pristupiti korisnikovim Spotify podacima, sam korisnik mora dati aplikaciji izričitu dozvolu za pristup tim podacima. Redirekcijom korisnika na URL adresu *https://accounts.spotify.com/* i krajnju točku */authorize* moguće je dobiti tu dozvolu. Kao *query* parametri u adresu stavljaju se dobivene vjerodajnice u prethodnom koraku i željeni doseg pristupa podacima kako bi Spotify Web API mogao detektirati koje dozvole korisnik dodjeljuje aplikaciji.

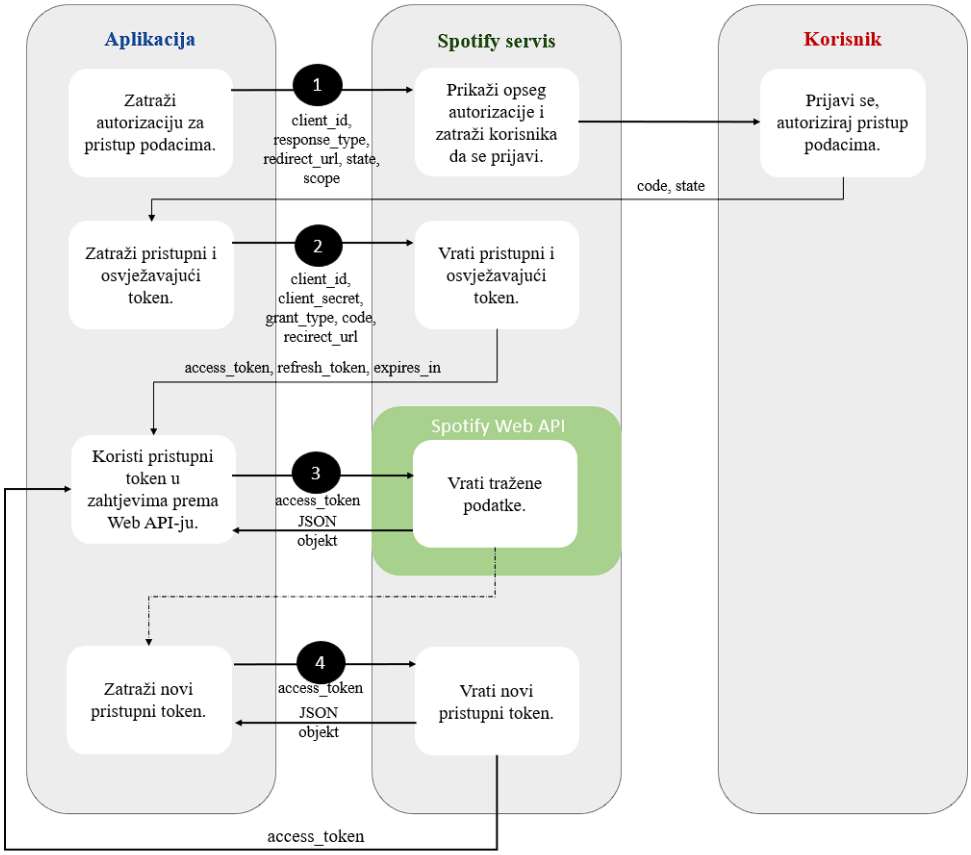
4. **Pristanak korisnika** - Na preusmjerenoj Spotify stranici traži se korisnikov pristanak da se aplikaciji omogući pristup podacima. Uspješnim prihvaćanjem poziva se povratni poziv aplikacije definiran pri samoj registraciji aplikacije.

5. **Dohvat pristupnog tokena*(eng. access token)*** - Pri povratnom pozivu Spotify Web API šalje i autorizacijski kod koji služi za dohvat pristupnog tokena. Pozivom API poziva na adresi *https://accounts.spotify.com/api/token* moguće je dohvatiti taj token. Kao parametar zaglavlja potrebno je postaviti kodirane vjerodajnice pomoću algoritma Base64. Kao sadržaj API poziva potrebno je poslati autorizacijski kod dobiven u prethodnom koraku. Kao odgovor na ovaj poziv aplikacija dobiva pristupni token *(eng. access token)*, token za osvježavanje pristupnog tokena *(eng. refresh token)*, vrijeme valjanosti pristupnog tokena i doseg pristupnog tokena.

6. **API pozivi** - Dobiveni pristupni token potrebno je staviti u zaglavlje svakog API poziva kojim se pristupa korisnikovim privatnim podacima.

7. **Osvježavanje pristupnog tokena** - Kako se ne bi narušila korisnikova sigurnost pristupni token potrebno je osvježiti svaki put kada istekne njegovo vrijeme valjanosti. Pristupni token osvježava se na način da se pošalje API zahtjev na URL adresu *https://accounts.spotify.com/api/token*. U zaglavlje poziva potrebno je postaviti kodirane vjerodajnice pomoću algoritma Base64, a u tijelo poziva je potrebno staviti *refresh token* i postaviti vrijednost *grant type* na *refresh token*. Odgovor API poziva bit će identičan odgovoru kod inicijalnog dohvaćanja pristupnog tokena pomoću autorizacijskog koda.

Kod implementacije aplikacije budući da je pristupni token potreban u svakom API pozivu i potrebno ga je dohvaćati u raznim dijelovima aplikacije, on će biti spremljen u zajedničke postavke *(eng. shared preferences)*.



Slika 3.2: Shematski prikaz autorizacije **[5]**

3.1.3. Korištene krajnje točke *(eng. endpoints)*

Osim već spomenute krajnje točke *https://accounts.spotify.com/api/token* na koji se šalje POST zahtjev s podacima opisanim u prethodnom potpoglavlju u implementaciji aplikacije korištene su i druge krajnje točke koje omogućava Spotify Web API. Krajnje točke nadovezuju se na pristupni URL: *"https://api.spotify.com"*. Važno je napomenuti da je u zaglavlje svakog API zahtjeva potrebno postaviti polje *Authorization* na vrijednost dobivenog pristupnog tokena. Korišteni su sljedeće *endpointovi* **[3]**:

- **/v1/me/top/{type}** - Pomoću ove krajnje točke dohvaća se lista korisnikovih najslušanijih pjesama i izvođača. Parametri upita su *type*, *time_range*, *limit* i *offset*. Pomoću parametra *type* definira se dohvaća li se lista korisnika ili pjesama. Parametar *time_range* definira za koje vremensko razdoblje se dohvaćaju podaci. Moguće vrijednosti su *short_term*, *medium_term* i *long_term*. Pripadna vremenska razdoblja su zadnjih mjesec dana, zadnjih pola godine i zadnjih godinu dana. Parametar *limit* postavlja ograničenje na maksimalni broj dohvaćenih elemenata. Valjane vrijednosti su cijeli brojevi iz intervala [1, 50].

Zadnji parametar *offset* određuje pomak od početka liste najslušanijih pjesama, odnosno izvođača.

- **/v1/me/tracks** - Pomoću ove krajnje točke dohvaćaju se pjesme koje je korisnik spremio u svoju knjižnicu pjesama. Dohvaćaju se vremenski zadnje spremljene pjesme. Manipulacijom parametra *limit* i *offset* moguće je dobaviti različite pjesme iz korisnikove knjižnice. Značenje i ograničenja ovih elemenata ista su kao i na prethodno opisanoj krajnjoj točci.
- **/v1/search** - Pomoću ove krajnje točke moguće je pretražiti cijeli Spotify katalog te dohvatiti različite informacije o dostupnim albumima, izvođačima i pjesmama. Važan parametar upita je *'q'*. Kao vrijednost ovog parametra postavlja se string pomoću kojeg se sužava pretraga kataloga. U pretrazi se koriste različiti filteri. Pomoću filtera *'artist:'* i *'track:'*moguće je zadati naziv pjesme i naziv izvođača. Ukoliko je poznat jedinstveni identifikator elementa Spotify kataloga umjesto ovog parametra moguće je koristiti parametar *id*. Kao vrijednost ovog parametra postavlja se identifikator elemenata. Parametri *q* i *id* su međusobno isključivo, ali pri samom pozivu jedan od njih obavezno mora biti ispravno definiran. Ostali parametri ovog upita su *limit* i *offset*. Njihova funkcionalnost je identična kao i u prethodna dva objašnjena poziva.
- **/v1/recommendations** - Pomoću ove krajnje točke moguće je dohvatiti pjesme koje Spotify algoritam predlaže na temelju specificiranih podataka. Podatke zadajemo kroz tri različita parametra upita: *'seed_genres'*, *'seed_artists'*, *'seed_tracks'*. Kao vrijednosti ovih parametra predaje se lista identifikatora pjesama, izvođača i žanrova odvojenih zarezom na temelju kojih se žele dohvatiti preporuke. Moguće je i specificirati tržište kako bi se u odgovoru nalazile pjesme koje su dostupne korisniku.

Kao odgovor na zahtjev ove krajnje točke vraćaju liste elemenata koji zadovoljavaju specificirane zahtjeve. Podaci su reprezentirani u JSON formatu. Kako bi se efikasno upravljalo dobivenim odgovorima u Kotlinu potrebno je JSON odgovor pretvoriti u Kotlin objekte. Koristeći knjižnicu *retrofit2* **[16]** moguće je poslati zahtjev na željenu krajnju točku te dobiveni odgovor u JSON formatu pretvoriti u odgovarajuće Kotlin objekte. Kako bi se pretvorba uspješno izvela potrebno je napisati klasu koja će predstavljati očekivani format odgovora te prilikom slanja HTTP poziva specificirati klasu koja modelira očekivani odgovor.

3.2. Last.Fm API

3.2.1. Općenito

Last.Fm je glazbena platforma koja pruža različite usluge povezane s glazbom **[13]**. Neke od tih usluga su reprodukcija glazbe i generiranje preporuka na temelju personaliziranih korisničkih informacija. Ova platforma također pruža API koji omogućava programerima pristup i interakciju s različitim dostupnim podacima. Sam API **[14]** pruža skup različitih metoda i krajnjih točaka koji omogućava programerima dohvat podataka o izvođačima, albumima, korisničkim profilima i mnogih drugih informacija. Ovaj API koristit će se u implementaciji kako bi se efektivno generirale preporuke pjesama i izvođača korisniku aplikacije. Specifičnost ovog programskog sučelja jest što se određene metode ne specificiraju odabirom odgovarajuće krajnje točke, već se odgovarajuća metoda poziva u ovisnosti o vrijednosti parametra upita *'method'*.

3.2.2. Autorizacija

Budući da će se preporuke generirati na temelju spremljenih pjesama iz Spotify knjižnice spremljenih pjesama aplikaciji neće biti potreban pristup korisničkim podacima koje pruža Last.Fm. Ono što jest potrebno je pristup GET metodi pomoću koje je moguće dohvatiti slične pjesme onoj koja je specificirana u tijelu zahtjeva. Kako bi aplikacija ostvarila pristup toj metodi potrebnu ju je registrirati na *https://www.last.fm /api/account/create*. Registracijom aplikaciji je dodijeljen API ključ kojeg je potrebno postaviti kao parametar upita *(eng. query parameter)* u API poziv metode.

3.2.3. Korištene metode

Iz ovog programskog sučelja u implementaciji će se koristiti GET metoda **track.getSimilar**. Metoda se poziva slanjem HTTP zahtjeva na adresu *"http://ws.audioscrobbler.com/2.0"*. Parametri upita su sljedeći:

- **method** - Ovim parametrom specificira se koja metoda programskog sučelja se poziva. Za ovaj API poziv vrijednost ovog parametra uvijek mora biti postavljena na *track.getsimilar*.
- **artist** - Ovaj parametar upita specificira izvođača pjesme za koju se traže preporuke.
- **track** - Ovim parametrom definira se naziv pjesme za koju se traže preporuke.

- **format** - Ovim parametrom moguće je definirati u kojem formatu će odgovor biti poslan. Moguće vrijednosti su 'xml' i 'json'. U implementaciji same aplikacije ovaj parametar uvijek će imati vrijednost 'json'
- **api_key** - Ovaj parametar važan je za autentifikaciju aplikacije. Kao vrijednost ovog parametra treba se postaviti API ključ dobiven registracijom aplikacije.
- **autocorrect** - Ovaj parametar zapravo je zastavica koja omogućava ispravljanje grešaka u vrijednosti parametara *artist* i *track*. Ako je vrijednost zastavice 0, onda ispravljanje nije omogućeno, a ako je 1 ispravljanje je omogućeno. Pretpostavljena vrijednost je 1.
- **limit** - Ovim parametrom ograničava se duljina odgovora, odnosno broj elemenata koji se vraća kao odgovor na API zahtjev.

3.3. Shazam API

3.3.1. Općenito

Shazam API jest aplikacijsko programsko sučelje dostupno putem platforme Rapid API **[4]**. Putem platforme Rapid API pruža se širok spektar programskih sučelja različitih funkcionalnosti. Jedan od dostupnih programskih sučelja jest upravo i Shazam API koji nudi mogućnost prepoznavanja pjesama na temelju snimljenih audio značajki. Kako bi se ovo programsko sučelje moglo uspješno koristiti programer se mora registrirati na platformi Rapid API te se pretplatiti na Shazam API. Ono što jest mana ovog programskog sučelja je što postoji ograničenje na broj zahtjeva koji korisnik može poslati. Tako je *rate limit* na ovo sučelje postavljen na 500 zahtjeva mjesečno.

3.3.2. Autorizacija

Kako bi programer pristupio navedenom programskom sučelju mora se registrirati na platformi Rapid API. Nakon registracije samog računa potrebno je dodati aplikaciju kojom će se pristupati aplikacijskom sučelju. Nakon toga pretplatom na odabrano aplikacijsko sučelje (u ovom slučaju Shazam API) dobiva se API ključ za traženo sučelje.

3.3.3. Korištene krajnje točke *(eng. endpoints)*

Različitim API metodama Shazam API programskog sučelja pristupa se odabirom odgovarajuće krajnje točke. Krajnje točke nadovezuju se na pristupni URL: *https://shazam.p.rapidapi.com*. Na krajnjoj točki **/songs/v2/detect** nalazi se POST metoda koja omogućuje prepoznavanje pjesme na temelju poslanih audio značajki. U parametrima zaglavlja potrebno je postaviti sljedeće vrijednosti:

- **content-type** - Ovim parametrom specificira se kakav sadržaj se prenosi u tijelu API zahtjeva. Vrijednost ovog parametra mora biti postavljena na *text/plain* s obzirom da će se snimljene audio značajke kodirati algoritmom Base64.
- **X-RapidAPI-Key** - Vrijednost ovog parametra postavlja se vrijednost API ključa dobivenog registracijom aplikacije i pretplatom na samo programsko sučelje
- **X-RapidAPI-Host** - Vrijednost ovog parametra postavlja se na znakovni niz koji predstavlja gdje se na platformi Rapid API nalazi željeno aplikacijsko sučelje. Vrijednost X-RapidAPI-Host parametra za Shazam API jest *shazam.p.rapidapi.com*

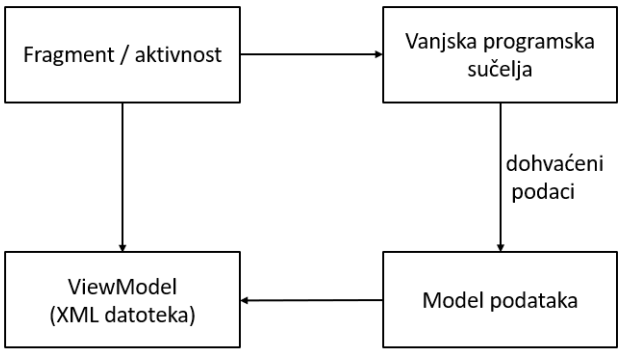
U tijelu POST zahtjeva potrebno je poslati znakovni niz kodiran algoritmom Base64. Ulaz u algoritam Base64 jest niz bajtova koji su generirani snimanjem pjesme koju korisnik želi prepoznati. Snimljeni uzorak mora biti u izvornom obliku, ne smije biti veći od 500 kilobajta, snimljeni uzorak mora pokrivati 3 do 5 sekundi izvorne pjesme. Izvorni podaci moraju biti snimljeni s uzrokovanjem od 44100 Hz, koristeći mono kanal i 16-bitni PCM *little endian* zapis. Kao odgovor na ovaj POST zahtjev vraća se lista prepoznatih pjesama na temelju poslanih audio značajki.

4. Implementacija

4.1. Implementacijski model

Aplikacija je implementirana koristeći programski jezik Kotlin i razvojnu okolinu Android Studio. Ova razvojna okolina pruža sveobuhvatan skup alata i resursa za uspješni razvoj aplikacija te otkrivanje i ispravljanje pogrešaka. Uz Kotlin i Android Studio korišten je Android SDK *(eng. Software Development Kit)* koji pruža nužne alate i programska sučelja za izgradnju aplikacija namijenjenih operacijskom sustavu Android.

Sama aplikacija sastoji se od fragmenata koji se međusobno povezuju u različite aktivnosti. Svaka aktivnost i fragment mora biti povezana s pripadajućom *ViewModel* komponentom. *ViewModel* je osnovna komponenta Android arhitekture u kojoj se pohranjuje i upravlja podacima važnim za prikaz korisničkog sučelja. U implementaciji aplikacije svaka instanca ove komponentne definirana je jednom XML datotekom. Svaki fragment aplikacije dohvaća potrebne podatke za prikaz preko jednog od prethodno opisanih programskih sučelja. Odgovori koji pristižu na poslane zahtjeve se kroz implementirane modele podataka prosljeđuju odgovarajućoj instanci *ViewModel* komponente. Tako se u aplikaciji prikazuju odgovarajući podaci.



Slika 4.1: Međusobni odnos komponenata aplikacije

4.2. Pregled implementiranih funkcionalnosti

Cjeloviti izvorni kod aplikacije dostupan je na javnom GitHub repozitoriju **[15]**. Kod je organiziran u pakete ovisno o tome na koju komponentu same aplikacije se odnosi i koju funkcionalnost obavlja. Paketi su sljedeći:

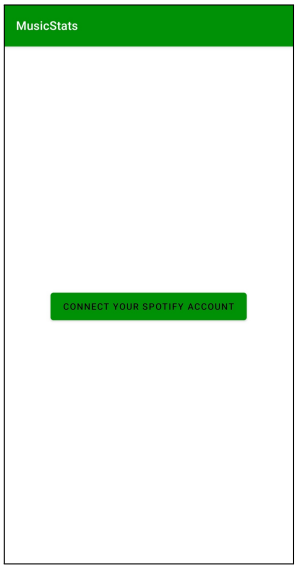
- **fragments** - U ovom paketu nalazi se izvorni kod kojim se implementiraju različiti fragmenti.
- **datamodels** - U ovom paketu nalazi se izvorni kod koji modelira i filtrira podatke koji pristižu kao odgovori na poslane API zahtjeve.
- **network** - U ovom paketu definiraju se četiri sučelja pomoću kojih je moguće slati zahtjeve na korištena vanjska aplikacijska programska sučelja.
- **adapters** - Ovaj paket sadrži kod kojim se implementiraju adapteri za odgovarajući prikaz komponente Recycler View.
- **layout** - Ovaj paket sadrži niz XML datoteka kojima se definira odgovarajući izgled korisničkog sučelja. Svaka od ovih datoteka veže se na specifični fragment ili aktivnost i predstavlja instancu komponente *ViewModel*.

Izvan paketa u src direktoriju same aplikacije definirane su dvije klase *MainActivity.kt* i *SongFound.kt*. Ove dvije klase implementiraju dvije glavne aktivnosti u aplikaciji. Kroz glavnu aktivnost *MainActivity.kt* modeliraju se sljedeće funkcionalnosti: prikaz najčešće slušanih pjesama i izvođača, generiranje glazbenih preporuka te prepoznavanje pjesme na temelju snimljenih audio značajki. Pomoću klase *SongFound.kt* modelira se aktivnost koja se pokreće, ako je prepoznavanje pjesme bilo uspješno.

4.2.1. Implementacija autorizacije Spotify Web API-ja

Kao što je i opisano u odjeljku 3.1.2. svaki korisnik aplikacije eksplicitno mora dati dozvolu aplikaciji kako bi aplikacija uspješno mogla dohvatiti korisnikove podatke sa Spotify platforme. Ovaj problem riješen je tako da se pri pokretanju same aplikacije u glavnoj aktivnosti provjerava postoji li zapis o pristupnom tokenu u zajedničkim postavkama aplikacije. Ako ne postoji zapis u zajedničkim postavkama pod ključem *'accessToken'* aplikacija u glavnu aktivnost smještava fragment kojim se obavlja autorizacija. Ovaj fragment definiran je pomoću klase *FragmentAuthorize.kt* i XML datoteke *fragment_authorize.xml*. Pokretanjem životnog ciklusa ovog fragmenta korisničko sučelje se pretvara u praznu stranicu na kojoj se nalazi gumb. Nad tim gumbom definiran je *onClickListener* koji čeka da korisnik pritisne gumb. Pritiskom gumba korisnik za-

počinje proces autorizacije detaljno opisan u odjeljku 3.1.2. Korisnika se iz aplikacije preusmjerava u web preglednik na službenu stranicu tvrtke Spotify. Na ovoj stranici traži se eksplicitna dozvola korisnika za dijeljenje njegovih podataka s aplikacijom. Pristankom na zahtjev korisnika se preusmjerava natrag u aplikaciju. Aplikacija kao odgovor dobiva autorizacijski kod koji šalje kao parametar POST zahtjeva na krajnju točku *https://accounts.spotify.com/api/token*. Kao odgovor na ovaj zahtjev aplikacija dobiva pristupni token i podatke o njegovoj valjanosti kao i odgovarajući token za osvježavanje. Sve ove informacije o pristupnom tokenu aplikacija sprema u zajedničke postavke kako bi se tim podacima lagano moglo pristupiti u svim fragmentima i aktivnostima same aplikacije.



Slika 4.2: Prikaz korisničkog sučelja prilikom autorizacije aplikacije na vanjskom sučelju Spotify Web API

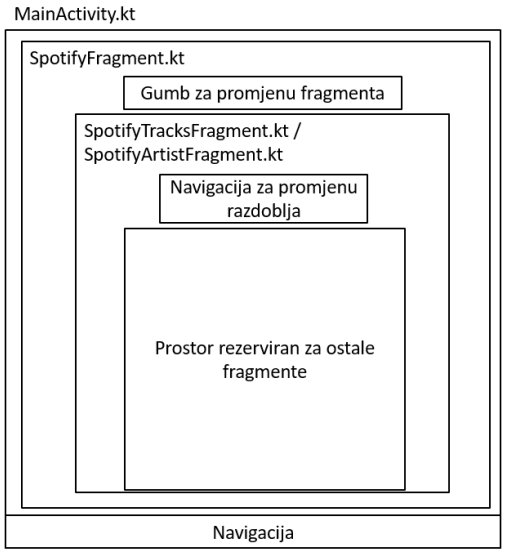
4.2.2. Početni prikaz aplikacije

Nakon dohvata pristupnog tokena za Spotify Web API aplikacija mijenja korisničko sučelje prikazano u glavnoj aktivnosti. Sam izgled aktivnosti nakon dohvata pristupnog tokena sastoji se od dva dijela: praznog rezerviranog okvira i navigacije na dnu ekrana. Pomoću navigacije na dnu ekrana moguće je mijenjati prikaz glavne aktivnosti. Ovisno o tome koji navigacijski gumb je korisnik zadnje odabrao prikazivat će se određeni fragment u praznom rezerviranom okviru. Navigacija nudi tri mogućnost koje korisnik može odabrati. Odabirom opcije ***Top tracks*** korisniku će se prikazati popis najslušanijih pjesama ili izvođača modeliran pomoću klase *SpotifyFragment.kt*. Odabirom druge opcije ***Recommendations*** u prazni rezervirani okvir umeće se *Vi-*

ewModel komponenta kojom se prikazuju odgovarajuće generirane preporuke. Odabirom treće opcije ***Recognize audio*** na rezervirano mjesto postavlja se komponenta kojom je omogućeno snimanje i prepoznavanje pjesme na temelju audio značajki.

4.2.3. Prikaz najslušanijih pjesama i izvođača

Kao što je već opisano u prethodnom odjeljku odabirom opcije **Top tracks** u navigaciji na dnu ekrana korisniku se otvara popis najslušanijih pjesama ili izvođača. Ova funkcionalnost ostvarena je pomoću 7 različitih fragmenata, 7 različitih XML datoteka i pomoću Spotify Web API programskog sučelja. Shematski prikaz izgleda aplikacije u trenutku kada se prikazuje lista najslušanijih pjesama ili izvođača prikazan je na slici 4.3.

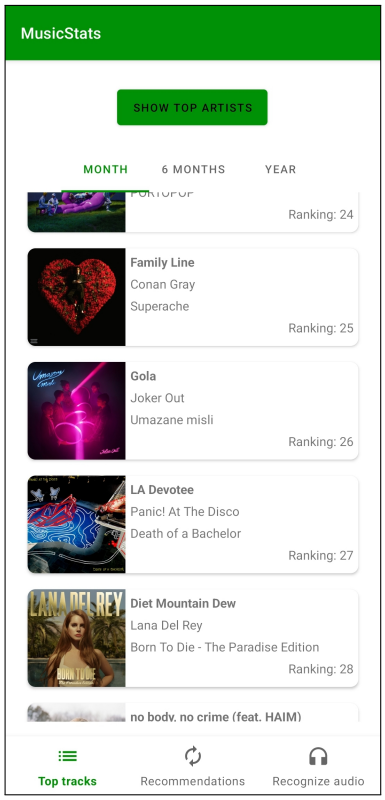


Slika 4.3: Shematski prikaz najslušanijih pjesama i izvođača

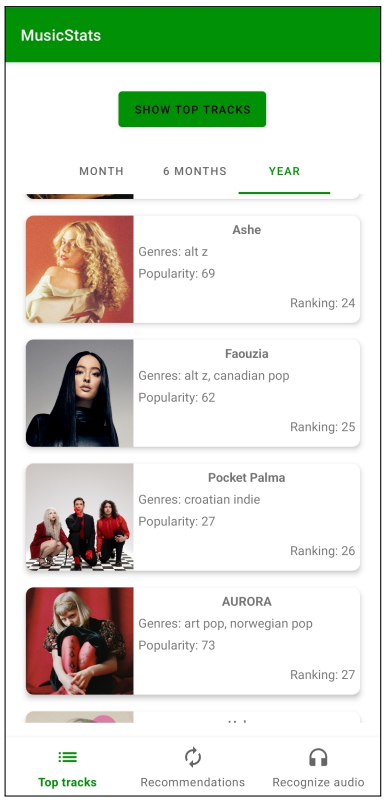
Glavna aktivnost modelirana klasom *MainActivity.kt* u prazni rezervirani okvir umeće fragment definiran klasom *SpotifyFragment.kt* i XML datotekom *fragment _spotify.xml*. Ovaj fragment definira dva elementa korisničkog sučelja. Jedan je rezervirani okvir za smještanje jednog od dvaju fragmenata: onaj koji će prikazivati listu najslušanijih pjesama ili onaj koji će prikazivati listu najslušanijih izvođača. Osim tog okvira ovaj okvir definira i gumb pomoću kojeg je moguće zamijeniti dva spomenuta fragmenta. Fragment koji prikazuje popis najslušanijih pjesama definiran je klasom *FragmentSpotifyTracks.kt* i XML datotekom *fragment_spotify_tracks.xml*. Ovaj fragment definira navigaciju koja se prikazuje na vrhu tog fragmenta. Ovom navigacijom korisnik može odabrati za koje vremensko razdoblje će se dohvaćati podaci. Dostupna

su tri vremenska razdoblja: zadnjih mjesec dana, zadnjih pola godine i zadnja godina. Ovisno o tome koja je opcija odabrana u prostoru rezerviranom za ostale fragmente povezivat će se odgovarajući fragment. Tako na primjer, ako je korisnik odabrao opciju dohvata podataka za vremensko razdoblje od zadnjih pola godine u ovaj okvir će se povezati fragment definiran klasom *TracksHalfYearFragment.kt* i XML datotekom *fragment_tracks_half_year.xml*. Svaki od fragmenata koji je moguće povezati na ovo mjesto obavlja istu funkcionalnost te rezervira samo mjesto za komponentu Recycler View. Dodatno svaki od tih fragmenata šalje GET zahtjev na Spotify Web API na krajnju točku */v1/me/top/tracks* objašnjenu u odjeljku 3.1.3. te postavlja parametar *time_range* na odgovarajuću vrijednost. Pristizanjem HTTP odgovora na API zahtjev pomoću modela podataka implementiranog u klasi *SpotifyTrackModel.kt* izdvajaju se podaci potrebni za prikaz. Potrebni podaci za prikaz su: naziv pjesme, naziv izvođača, naziv albuma te *URL* naslovne slike albuma. Ovi podaci zajedno sa pozicijom u listi koja predstavlja rang pjesme na korisnikovoj top listi predaju se instanci klasi *SpotifyTrackAdapter.kt*. Ova instanca klase predaje se adapteru komponente Recycler View. Komponenta Recycler View zatim prikazuje odgovarajuću listu elemenata na korisničkom sučelju. Ova lista ostaje aktivna dok god životni ciklus pripadajućeg fragmenta nije završio.

Na analogni način implementirana je i funkcionalnost fragmenata koji prikazuje listu najslušanijih izvođača korisnika. Pripadajući fragment definiran je klasom *FragmentSpotifyArtists.kt* i XML datotekom *fragment_spotify_artists.xml*.



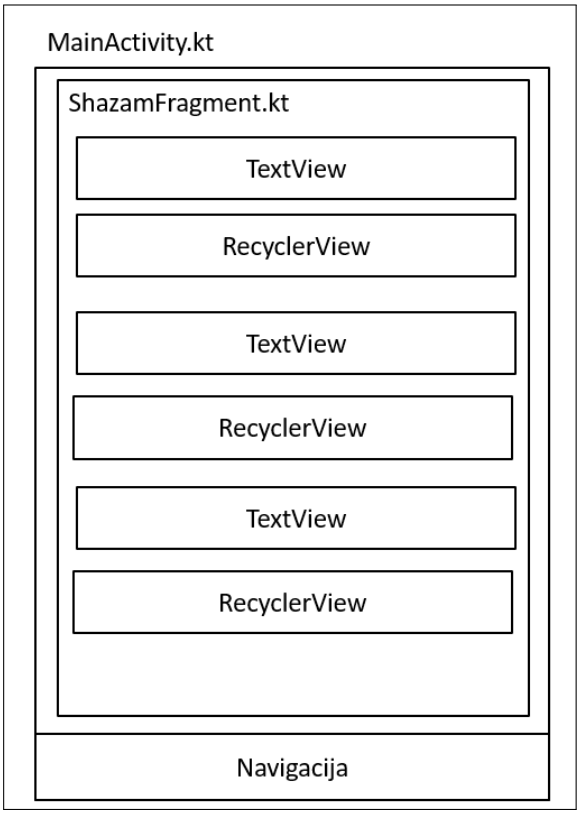
Slika 4.4: Prikaz pregleda najslušanijih pjesama u korisničkom sučelju



Slika 4.5: Prikaz pregleda najslušanijih izvođača u korisničkom sučelju

4.2.4. Generiranje glazbenih preporuka i njihov prikaz

Funkcionalnost generiranja glazbenih preporuka ostvarena je u fragmentu definiranom u klasi *ShazamFragment.kt* i XML datoteci *fragment_lastfm.xml*. Za prikaz generiranih preporuka zamišljeno je sučelje prikazano na slici 4.6. Korisničko sučelje sastoji se od tri okvira rezerviranih za tekst, te tri komponente Recycler View. Glazbene preporuke generiraju se na temelju tri pjesme koje je korisnik zadnje spremio u Spotify glazbenu knjižnicu. Naziv svake od te tri pjesme korisniku će biti prikazan u okvirima rezerviranima za tekst. Generirane preporuke prikazivat će se kao lista koju korisnik može pregledavati. Lista će ponovo biti implementirana komponentom Recycler View. Svaki element te liste prikazivat će: naziv preporučene pjesme, naziv izvođača pjesme, datum objave preporučene pjesme i sliku naslovne fotografije albuma na kojoj se nalazi preporučena pjesma. Uz ove podatke svaki element sadrži gumb *Listen now* koji korisnika preusmjerava na Spotify platformu gdje mu se otvara upravo generirana preporuka. Kroz Spotify platformu korisnik preporučenu pjesmu može spremiti u svoju knjižnicu, dodati na popis za reprodukciju ili pak podijeliti.



Slika 4.6: Shematski prikaz korisničkog sučelja namijenog prikazu glazbenih preporuka

Sam proces generiranja glazbenih preporuka može se podijeliti u nekoliko koraka:

1. **Dohvat spremljenih pjesama** - Generiranje započinje dohvatom pjesama koje je korisnik zadnje spremio u svoju vlastitu Spotify knjižnicu. Dohvaća se zadnjih 20 spremljenih pjesama te se za sve njih pokreće proces generiranja preporuka. Međutim nakon što se generiraju preporuke za tri pjesme ostali procesi generiranja se prekidaju te se prikazuju generirane preporuke za prve tri pjesme koje su dovršile proces generiranja. Ove pjesme dohvaćaju se putem Spotify Web API sučelja slanjem HTTP zahtjeva na */v1/me/tracks* krajnju točku.

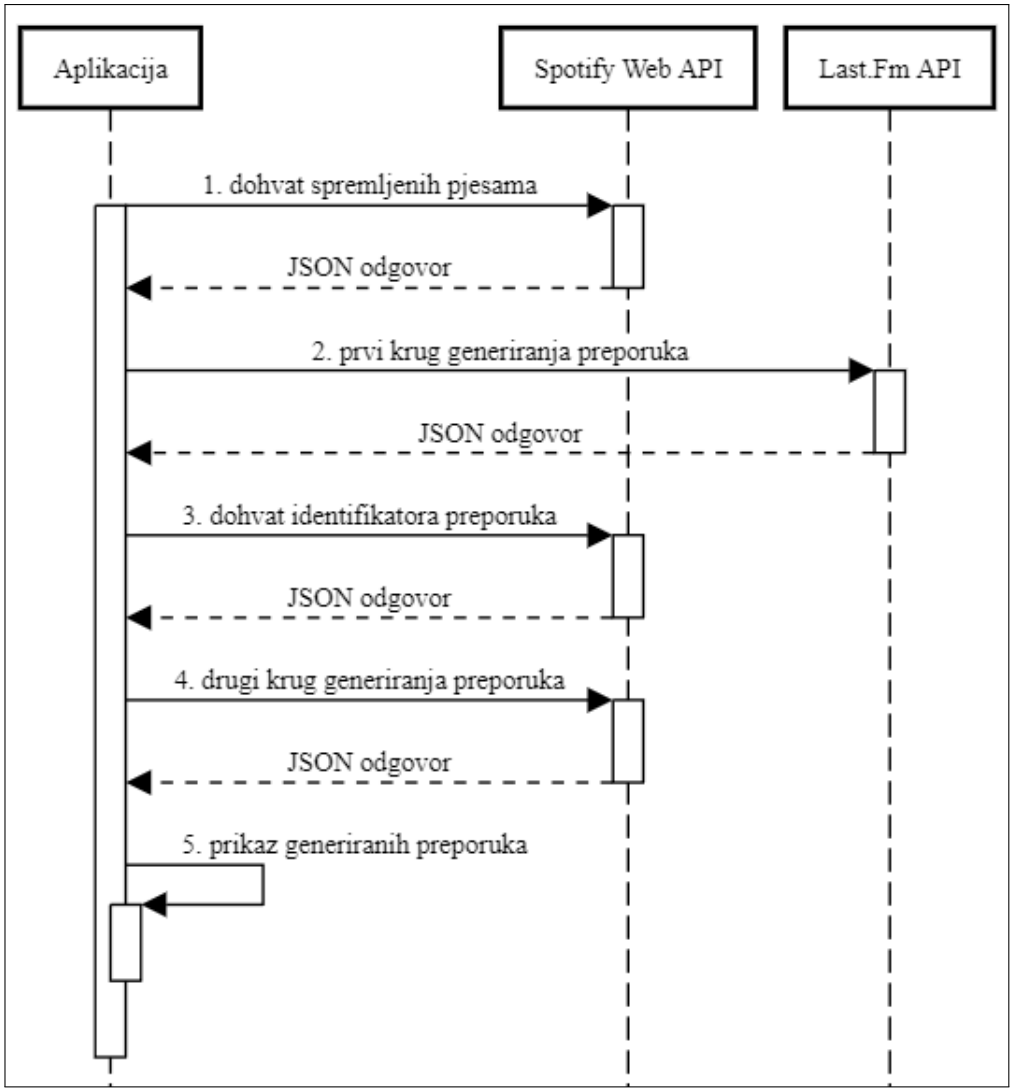
2. **Prvi krug generiranja preporuka** - Za svaku dohvaćenu pjesmu iz Spotify knjižnice dohvaća se pet preporuka koje generira platforma Last.Fm. Preporuke se dohvaćaju koristeći Last.Fm API programsko sučelje i metodu *track.getSimilar* objašnjenu u odjeljku 3.2.3. Na temelju dohvaćenih preporuka nastavlja se postupak generiranja preporuka.

3. **Dohvat Spotify identifikatora** - Za pet preporuka generiranih u prethodnom koraku šalje se HTTP zahtjev na Spotify Web API sučelje kojim se želi dohvatiti informacija o jedinstvenom identifikatoru koje Spotify platforma pridjeljuje svakoj od generiranih pjesama. Osim tog identifikatora pristupa se i jedinstvenom identifikatoru koji je pridijeljen izvođaču generirane preporuke. Ova dva identifikatora bit će važna u sljedećem koraku generiranja glazbenih preporuka. HTTP zahtjev šalje se na krajnju točku */v1/search* detaljno opisanu u odjeljku 3.1.3.

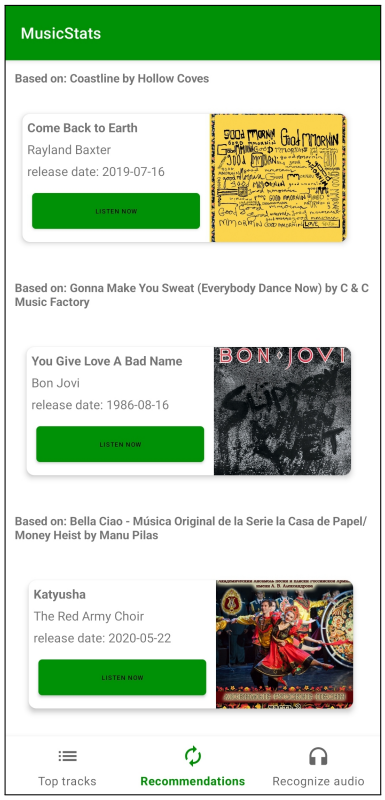
4. **Drugi krug generiranja preporuka** - Dobiveni identifikatori u prethodnom koraku šalju se kao parametri upita u HTTP zahtjevu na krajnju točku */v1/ recommendations* Spotify Web API programskog sučelja. Identifikatori se pridjeljuju kao vrijednosti poljima *seed_artists*, odnosno *seed_tracks*. Razlog zašto je u prvom krugu preporuka generirano samo pet preporuka leži u ograničenju ovog koraka. Naime, u vrijednosti polja *seed_artists* i *seed_tracks* moguće je poslati listu s najviše pet identifikatora u svakom polju pa čak i da je generirano više od pet preporuka u prvom krugu ne bismo mogli iskoristiti sve generirane preporuke.

5. **Dohvat konačnih glazbenih preporuka** - Kao rezultat prethodnog koraka dobiven je HTTP odgovor s glazbenim preporukama. Iz tog odgovora izvlače se podaci o nazivu generirane pjesme, izvođaču generirane pjesme, datumu izdavanja pjesme, URL adresi naslovne slike albuma te URL adresi pomoću koje je

moguće korisnika preusmjeriti na generiranu pjesmu na Spotify platformi. Informacije za svaku od pjesama pospremaju se kao elementi liste koja se predaje adapteru Recycler View komponente. Time je završen postupak generiranja preporuka.



Slika 4.7: Prikaz komunikacije komponenata aplikacije prilikom generiranja glazbenih preporuka

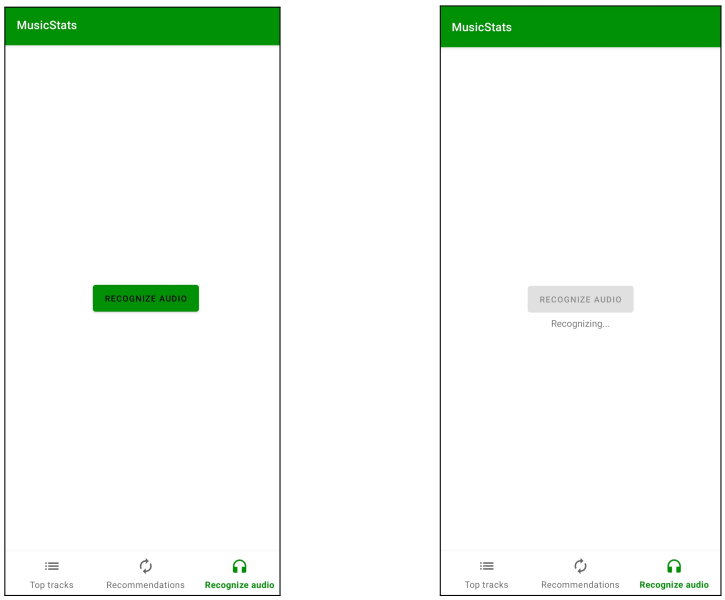


Slika 4.8: Prikaz implementiranog korisničkog sučelja namijenjenog prikazu glazbenih preporuka

4.2.5. Prepoznavanje pjesme na temelju snimljenih audio značajki

Funkcionalnost prepoznavanja pjesme na temelju snimljenim audio značajki implementirana je koristeći fragment definiran klasom *ShazamFragment.kt* i XML datotekom *fragment_shazam.xml*. Sučelje ovog fragmenta je vrlo jednostavno. Sve što postoji jest jedan gumb. Klikom na gumb korisnik pokreće proces snimanja zvuka uz pomoć mikrofona dostupnog na mobilnom uređaju. Zvuk se snima uz pomoć klase *AudioRecord* koja je dio korištenog Android SDK-a. Sam zvuk snima se u vremenskom razdoblju od 5 sekundi, uzrokovanje zvuka postavljeno je na 44100 Hz. Audio zapis snima se kao mono kanal i kodira se u PCM 16 bitnom *little endian* formatu. Kao izlaz samog snimanja klasa *AudioRecord* vraća niz bajtova koje je potrebno kodirati u znakovni niz algoritmom Base64. Zvuk mora biti snimljen isključivo na opisan način kako bi aplikacija u odgovarajućem obliku mogla poslati POST zahtjev na programsko sučelje Shazam API koristeći krajnju točku */songs/v2/detect*. U tijelu ovog POST zahtjeva šalje se upravo kodirani string algoritmom Base64 dobiven snimanjem zvuka. HTTP odgovor na poslani POST zahtjev može biti prazan ili pak može sadržavati informacije o prepoznatoj pjesmi. Ako je odgovor na zahtjev prazan korisnika se

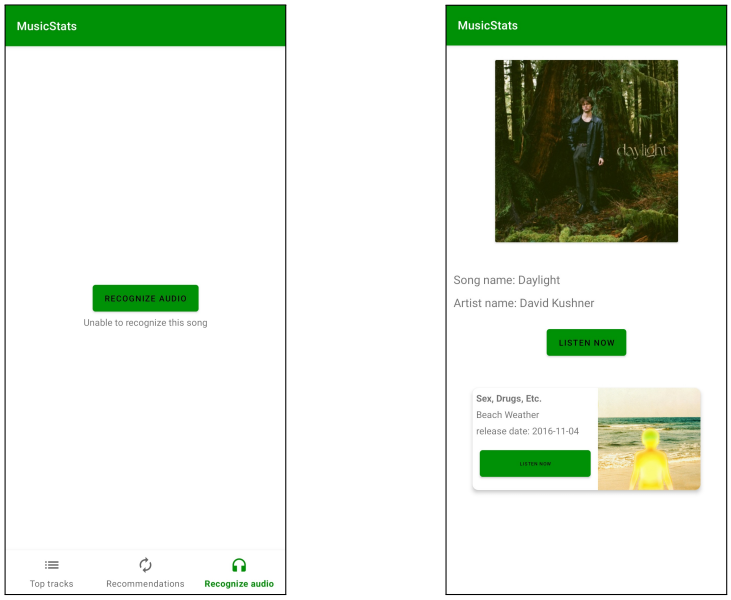
obavještava odgovarajućom porukom o nemogućnosti prepoznavanja pjesme kao što je prikazano na slici 4.10. Neuspješnim prepoznavanjem pjesme omogućuje se ponovni početak samog snimanja zvuka pa i time ponovni pokušaj prepoznavanja pjesme na temelju audio značajki.



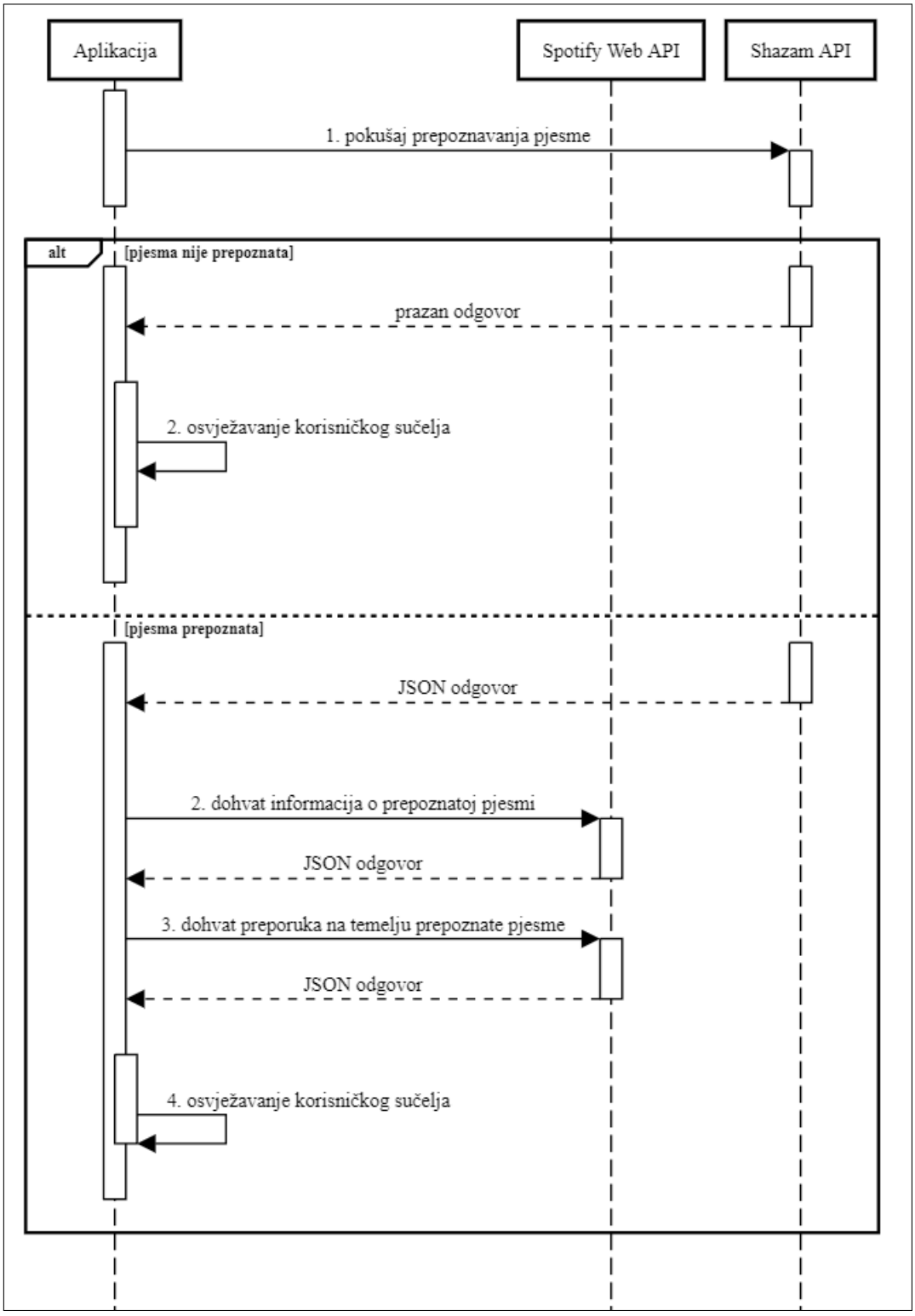
Slika 4.9: Prikaz implementiranog sučelja za prepoznavanje pjesama: standardni prikaz *(lijevo)*, prikaz tijekom prepoznavanja pjesme *(desno)*

Ukoliko se pak pjesma uspješno prepoznaje pokreće se nova aktivnost u aplikaciji definirana pomoću klase *SongFound.kt* i fragmenta *activity_song_found*. Ova aktivnost pokreće se nad glavnom aktivnosti aplikacije. Drugim riječima glavna aktivnost aplikacije ulazi u pauzirano stanje te se zove odgovarajuća *callback* metoda *onPause()*. Otvara se novi prozor s definiranom aktivnošću i novim izgledom. Korisničko sučelje ove aktivnosti puno je jednostavnije nego sučelje glavne aktivnosti. Prikazuje se naziv prepoznate pjesme, njen izvođač i naslovna slika albuma na kojem se prepoznata pjesma nalazi. Uz ove informacije u sučelju se nalazi gumb *Listen now*. Klikom na taj gumb korisnik se preusmjerava na Spotify platformu gdje mu se otvara prepoznata pjesma. Putem Spotify platforme korisnik ovu pjesmu može spremiti u svoju biblioteku pjesama. Dodatno u ovoj aktivnosti dohvaćaju se preporuke prepoznate pjesme putem Spotify Web API programskog sučelja i krajnje točke */v1/recommendations*. Dohvaća se desetak glazbenih preporuka te se one prikazuju ponovo prikazuju kroz Recycler View komponentu. U prikazu se navodi naziv preporučene pjesme, njen izvođač, datum izdavanja te ponovo postoji link koji vodi na samu pjesmu na Spotify platformi.

Budući da se ova aktivnost izvodi nad glavnom aktivnošću kada korisnik pozove akciju povratka natrag na svom mobilnom uređaju ova aktivnost završava s radom te se korisnik ponovo nalazi u glavnoj aktivnosti aplikacije. Glavna aktivnost aplikacija ulazi u aktivno stanje te nastavlja sa svojim radom tamo gdje je stala.



Slika 4.10: Prikaz implementiranog sučelja za prepoznavanje pjesama: neuspješno prepoznavanje *(lijevo)*, uspješno prepoznavanje *(desno)*



Slika 4.11: Prikaz komunikacije komponenata prilikom prepoznavanja pjesme

5. Budući rad

Unatoč uspješnoj implementaciji svih korisničkih zahtjeva postoje mjesta za nove funkcionalnosti te proširivanje postojećih. Tako na primjer kako bi se rasteretio klijent od ugniježđenih HTTP poziva mogao bi se napraviti poslužiteljski servis koji bi onda komunicirao za vanjskim servisima, a sam klijentski dio bi komunicirao direktno s implementiranim poslužiteljem.

Osim toga moguće je poboljšanje funkcionalnosti generiranja glazbenih preporuka. Aplikacija bi se mogla spajati na postojeću javno dostupnu bazu pjesama kao što je to MusicBrainz baza pjesama dostupna na *https://musicbrainz.org/*. Koristeći već integrirana aplikacijska sučelja aplikacija bi dohvatom značajki pjesama u korisnikovoj Spotify knjižnici te njihovom kombinacijom s dostupnim značajkama pjesama u bazi mogla generirati vlastite preporuke, a ne ovisiti o preporukama vanjskih servisa.

Prostor za napredak postoji i u funkcionalnosti prepoznavanja pjesme na temelju snimljenih audio značajki. Umjesto Shazam API programskog sučelja moguće je u izvorni kod aplikacije uvesti Shazam Kit koji je krajem 2021. godine izdala sama tvrtka Shazam. Ovaj softverski alat omogućuje istu funkcionalnost kao i Shazam API, no nema ograničenje na broj zahtjeva koje aplikacija može poslati.

Samo korisničko sučelje može se nadograditi koristeći smjernice za dizajn korisničkog sučelja. U ovoj fazi razvoja sam izgled korisničkog sučelja vrlo je jednostavan i pomalo primitivan.

6. Zaključak

Cilj ovog rada bio ja izraditi Android aplikaciju za upravljanje glazbenim profilom. Radom na samoj aplikaciji uspješno su implementirani korisnički zahtjevi kroz interakciju tri različita aplikacijska programska sučelja: Spotify Web API, Last.Fm API i Shazam API. U većoj mjeri uspješno su postignuti ciljevi postavljeni na samom početku rada na aplikaciji.

Kroz aplikaciju korisnici mogu istraživati svoj glazbeni profil te na temelju samog profila pregledavati generirane preporuke. Integracijom Spotify Web API programskog sučelja omogućen je pristup samim preporukama što korisniku omogućuje pristup širokom spektru nove glazbe koju može istraživati, spremiti za kasnije slušanje ili dijeliti. Shazam API obogaćuje aplikaciju funkcionalnošću prepoznavanja pjesama na temelju snimljenih audio značajki kroz samu aplikaciju. Međusobnom interakcijom Spotify Web API i Last.Fm API programskih sučelja implementirano je generiranje glazbenih preporuka na temelju postojećeg glazbenog profila. Preporuke ne samo da obogaćuju sam glazbeni profil, već i pružaju mogućnost samom korisniku da promijeni svoj glazbeni profil.

Sam proces razvoja aplikacije bio je izazovan. Jedan od prvih izazova bio je povezan sa samom autorizacijom aplikacije na tri različita vanjska sučelja čiji se sami postupci autorizacije razlikuju. Zatim dobivanje odgovarajućih dozvola s korisničke strane za pristup privatnim podacima kao što je pristup Spotify knjižnici je također bio dosta izazovan dio rada. Unatoč problemima koji su se pojavljivali u samoj implementaciji uz višesatno čitanje same dokumentacije, različite tehnike otklanjanja pogrešaka i snimanjem mrežnog prometa koje aplikacije generira implementacija je uspješno privedena kraju.

Pokazalo se da je odabir tehnologije za razvoj same aplikacije bio solidan izbor. Samo rukovanje korisničkim sučeljem putem različitih fragmenata i aktivnosti vrlo je intuitivno. Sama implementacija izgleda elemenata aplikacije XML datotekama pokazao se kao dobar i jednostavan način definicije. Međutim postoje određena ograničenja odabrane tehnologije. Razvojem u programskom jeziku Kotlin implementirana apli-

kacija ograničena je na mobilne uređaje koji podržavaju operacijski sustav Android. Ukoliko bi se ukazala potreba za prenosivošću aplikacije na uređaje drugog operacijskog sustava (primjerice iOS) bilo bi potrebno ili tražiti neka zaobilazna rješenja koja bi postojeći Kotlin kod upakiravala u kontejnere koji bi bili veće prenosivosti ili pak izgraditi aplikaciju ispočetka u programskom jeziku koji daje podršku za razvoj mobilnih aplikacija neovisno o operacijskom sustavu (primjerice Flutter).

Također ova aplikacija nema vlastiti pozadinski servis *(eng. backend)* te se svi dohvati i komunikacija s vanjskim servisima odvijaju na klijentu. Posljedica ove arhitekture može se osjetiti pri generiranju glazbenih preporuka. Zbog višestruke ugniježđenosti API poziva samoj aplikaciji treba nekoliko sekundi da dohvati preporuke. Tih par sekundi uzrokuje pad u performansama aplikacije i negativno utječe na samo korisničko iskustvo korištenja aplikacije.

Unatoč tome što je aplikacija uspješno ispunila korisničke zahtjeve, nije spremna za puštanje u pogon. Jedan od glavnih razloga je malo prije spomenut problem dugog generiranja preporuka. Još jedan razlog zbog kojeg aplikacija nije spremna za puštanje u pogon jest ograničenje Shazam API programskog sučelja na broj zahtjeva koje je moguće poslati.

LITERATURA

[1] Jet Brains. JetBrain Developers Blog. Blog, 2023. URL https://blog.jetbrains.com/search/?p=Kotlin. Pristupljeno 24. svibnja 2023.

[2] Jet Brains. Kotlin Docs. Dokumentacija, 2023. URL https://kotlinlang.org/docs/home.html. Pristupljeno 24. svibnja 2023.

[3] Spoitfy For Developers. Spotify Web API. Dokumentacija, 2023. URL https://developer.spotify.com/documentation/web-api. Pristupljeno 24. svibnja 2023.

[4] Api Dojo. Shazam. Dokumentacija, 2023. URL https://rapidapi.com/apidojo/api/shazam. Pristupljeno 24. svibnja 2023.

[5] Spotify for Developers. Authorization Code Flow. Dokumentacija, 2023. URL https://developer.spotify.com/documentation/web-api/tutorials/code-flow. Pristupljeno 25. svibnja 2023.

[6] Google. Develop Android applications with Kotlin. Interaktivne upute za upoznavanje s konceptima, 2020. URL https://developer.android.com/kotlin. Pristupljeno 24. svibnja 2023.

[7] Google. Fragment lifecycle. Priručnik, 2022. URL https://developer.android.com/guide/fragments/lifecycle. Pristupljeno 25. svibnja 2023.

[8] Google. Use recycler view to display scrollable list. Interaktivne upute, 2022. URL https://developer.android.com/codelabs/basic-android-kotlin-training-recyclerview-scrollable-list. Pristupljeno 25. svibnja 2023.

[9] Google. The activity lifecycle. Priručnik, 2023. URL https:

//developer.android.com/guide/components/activities/activity-lifecycle. Pristupljeno 24. svibnja 2023.

[10] Google. Create dynamic lists with recyclerview. Priručnik, 2023. URL https://developer.android.com/develop/ui/views/layout/recyclerview. Pristupljeno 25. svibnja 2023.

[11] Google. Save simple data with shared prefrences. Priručnik, 2023. URL https://developer.android.com/training/data-storage/shared-preferences. Pristupljeno 25. svibnja 2023.

[12] IBM. What is an API? Članak, 2021. URL https://www.ibm.com/topics/api. Pristuepljeno 25. svibnja 2023.

[13] Last.fm Music. Last.FM. Last.Fm platforma, 2023. URL https://www.last.fm/home. Pristupljeno 29. svibnja 2023.

[14] Last.fm Music. Last. FM API. Dokumentacija, 2023. URL https://www.last.fm/api/intro. Pristupljeno 24. svibnja 2023.

[15] Borna Nikolić. Izvorni kod. Git repozitorij, 2023. URL https://github.com/bnikolic2/ZavrsniRad. Pristupljeno 31. svibnja 2023.

[16] Square. Retrofit - type safe HTTP Client for Android and Java. Dokumentacija, 2015. URL https://square.github.io/retrofit/. Pristupljeno 25. svibnja 2023.

[17] TutorialandExample.com. Web API tutorial. Priručnik, 2019. URL https://www.tutorialandexample.com/web-api-tutorial. Pristupljeno 25. svibnja 2023.

Android aplikacija za upravljanje glazbenim profilom

Sažetak

Ovaj rad prati izradu i funkcionalnost Android mobilne aplikacije za upravljanje glazbenim profilom. Aplikacija je prvenstveno namijenjena Android korisnicima koji su ujedno i korisnici Spotify platforme. Aplikacija korisniku pruža uvid u njegov glazbeni profil te generira preporuke na temelju korisnikovih preferenci.

Ključne riječi: Kotlin, mobilna aplikacija, Android, aplikacijska programska sučelja, glazbeni profil, mobilni razvoj

Andorid aplication for music profile managment

Abstract

This paper presents the development and functionality of an Android mobile application for managing a music profile. Application is primarily designed for Android users who are also Spotify platform users. The application provides users with insights into their music profile and generates recommendations based on their preferences.

Keywords: Kotlin, mobile application, Android, application programming interfaces, music profile, mobile development