RAZVOJ MOBILNE APLIKACIJE ZA PRONALAZAK I ORGANIZIRANJE TABLATURA ZA SVIRANJE

Šprajc, Sandro

VISOKO UČILIŠTE ALGEBRA

ZAVRŠNI RAD

RAZVOJ MOBILNE APLIKACIJE ZA PRONALAZAK I ORGANIZIRANJE TABLATURA ZA SVIRANJE

Sandro Šprajc

Zagreb, 02. 2023.

„Pod punom odgovornošću pismeno potvrđujem da je ovo moj autorski rad čiji niti jedan dio nije nastao kopiranjem ili plagiranjem tuđeg sadržaja. Prilikom izrade rada koristio sam tuđe materijale navedene u popisu literature, ali nisam kopirao niti jedan njihov dio, osim citata za koje sam naveo autora i izvor, te ih jasno označio znakovima navodnika. U slučaju da se u bilo kojem trenutku dokaže suprotno, spreman sam snositi sve posljedice uključivo i poništenje javne isprave stečene dijelom i na temelju ovoga rada“.

U Zagrebu, 23.02.2023.

Predgovor

Posebne zahvale mentoru dr.sc. Aleksanderu Radovanu i Visokom Učilištu Algebra na znanju koje je prethodilo i tako omogućilo izradu završnog rada.

Prilikom uvezivanja rada, Umjesto ove stranice ne zaboravite umetnuti original potvrde o prihvaćanju teme završnog rada kojeg ste preuzeli u studentskoj referadi

Sažetak

Dokument završnog rada produkt je istraživanja i korištenja dostupnih alata, koncepata i principa izrade mobilnih aplikacija na primjeru izrade aplikacije nazvane Chordiato. Izradom konkretnog rješenja programske podrške, pri kojoj su se primjenjivala stečena znanja, navode se analize, opisi i zaključci poduprti argumentima zasnovanim na činjenicama. Svrha implementiranog rješenja u vidu mobilne aplikacije za Android platformu, jest olakšavanje pronalaska potrebnog materijala za izvođenje skladbe, koji je u obliku teksta pjesme s akordima ili tablatura. Pronalazak se vrši pomoću komadića pjesme koji korisnik registrira mikrofonom uređaja. Chordiato aplikacija također omogućava spremanje i upravljanje prethodno prepoznatih pjesama , te tako još više olakšava pristup željenim glazbenim zapisima. Cjelovito rješenje je jednostavni, korisni alat namijenjen svakome tko želi svirati. Dokument započinje detaljnom analizom korištenog algoritma pretrage, nakon čega se opisuju funkcionalnosti i dočarava korištenje. Aplikacija se uspoređuje sa sličnim postojećim rješenjima, a zatim se duboko ulazi u proces izrade. Daje se uvid u pojedine komponente rješenja, objašnjavaju se i razlažu problemi, te opisuje njihovo rješavanje.

Ključne riječi: Mobilna aplikacija, Android, Kotlin, tablature

Abstract

The document of the undergraduate thesis is a product of research and the use of available tools, concepts, and principles of software development. By developing a concrete software solution in which acquired knowledge was applied, the document addresses analysis, descriptions, and conclusions based on facts. The purpose of the implemented software solution, which is in the form of an Android mobile application, is to facilitate the process of researching resources needed for playing a song in the form of lyrics with chords or tablatures. The finding is made by taking a song sample using the microphone of a device. The Chordiato application also allows users to save and manage already recognized songs, making it even easier to retrieve wanted music sheet data. The whole solution is a simple tool intended for anyone who wants to play. The document starts with a thorough analysis of the used music recognition algorithm. Afterward, the usage of the application is presented, and its functionalities are described. The application is compared with existing similar solutions, and then the thorough development process description starts. Insight into the solution components is given, problems are explained, decomposed, and the process of solving them is explained.

Keywords: Mobile application, Android, Kotlin, tablatures

Sadržaj

1. Uvod

Kada je čovječanstvo došlo na razinu da se ne mora u svakom trenutku brinuti za preživljavanje, višak vremena i energije otvara vrata umjetnosti. Razvijeni ljudski um prezire dosadu i žudi za stimulacijom. Inteligentno biće shvaća da može uživati u stvarima koje nemaju materijalnu vrijednost ili ne pružaju užitak za koje je programirano da mu godi. Moderna istraživanja dokazala su da sviranje izaziva aktivaciju svakog područja mozga, što nije karakteristično ni za jednu drugu aktivnost, te kao takvo, donosi dugotrajne pozitivne posljedice (Collins, 2014.). Svaki glazbenik će se složiti da je sviranje predivno iskustvo koje toliko pruža, a ne uzima ništa zauzvrat. Zbog navedenog, važno je sviranje učiniti što dostupnijim i jednostavnijim, kako bi se što više ljudi počelo aktivno baviti sviranjem. Treba se iskoristiti postojeća tehnologija da se olakša i smanji bilo koja prepreka, koliko god bila minorna. Chordiato aplikacija, čiji je razvoj i implementacija tema završnog rada, upravo olakšava proces sviranja. To čini na način da snimi i obradi uzorak pjesme dug četiri sekunde i korisniku prikaže tekst pjesme popraćene akordima ili zapis u obliku tablatura za gitaru. Većina glazbenika vježba upravo tako, uz audio i notni zapis. Uzimajući u obzir količinu vremena koja je potrebna za savladavanje vještine sviranja, ušteđene minute od pretraživanja lako postaju sati i dani. Šanse da će početnik prestati aktivno svirati su visoke, no one se zasigurno barem malo smanje kada se ne treba trošiti vrijeme na sumorno pretraživanje, već staviti fokus na ono što je zapravo bitno. Pretraživanja se spremaju, te se mogu iskoristiti za brzi pristup željenom materijalu za izvođenje pjesme. Svrha završnog rada jest objasniti kako implementirano rješenje funkcionira, detaljno i iscrpno opisati procese koji se moraju odviti da bi se krajnja funkcionalnost ostvarila. Započinje se od korištenog servisa koji sadrži algoritam pretraživanja pjesama; poglavlje koje je temeljeno na dokumentu izdanog od jednog od njegovih tvoraca. Prikazuje se aplikacija iz perspektive korisnika gdje se ističu funkcionalnosti i princip rada, nakon čega se uspoređuje sa sličnim postojećim rješenjima. Objašnjava se proces izrade i implementacije koji je bio zasnovan na dobrim praksama i preporukama iz cijenjenih i popularnih knjiga u području softver i android razvoja. Osim knjiga, koristile su se internetske stranice i internet članci. Broj izvora rastao je organski, paralelno s razvojem aplikacije.

2. Algoritam prepoznavanja pjesama

Čovjek svoju okolinu spoznaje putem osjetila. Ona su mu jedini prozor kroz koji može doživjeti svijet, a tako i iskusiti samo postojanje. Između ostalog, zbog osjetila koja su se razvijala i unapređivala milijunima godinama na milijunima drugih životinjskih vrsta, čovjek je mogao postati razumno biće (lat. *animal rationale*) svjesno samo sebe. U usporedbi s većinom drugih životinjskih vrsta, čovjek ima zavidnu paletu primatelja podražaja, takozvanih receptora, koji pružaju fiziološku percepciju okoline. Jedno od osjetila je sluh, kojim se percipiraju zvučni valovi. Dok neke životinje imaju veliki raspon frekvencija[1] koje mogu čuti (npr. šišmiš 1000-120000Hz, delfin 150-150000Hz), čovjek čuje u rasponu od 20-20000Hz, što upućuje na to da su se sve bitne zvučne informacije koje je mogao primiti, nalazile upravo u tom rasponu (Paar, 2009). Mozak sisavaca, zbog postojanja razvijene moždane kore, odličan je u prepoznavanju obrazaca. Čovjeku je duboko usađena sposobnost raspoznavanja zvuka, jer mu je u vrijeme kada je prepoznavanje zvuka nadolazeće prijetnje značila život ili smrt, ta sposobnost osiguravala opstanak. Ta ista sposobnost omogućava mu iznimno dobro i posve intuitivno razabiranje kompleksnog zvuka poput glazbe i to u vrlo kratkom vremenu. Istraživanje provedeno od strane Manchesterovog muzeja znanosti i umjetnosti ispitivalo je ljudsku sposobnost prepoznavanja pjesme. Ispitanicima bi pustili temu (engl. *hook*) tisuću najprodavanijih pjesama, te mjerili vrijeme u kojem će ispitanici prepoznati pjesmu. Najbrže prepoznata pjesma bila je *Wannabe* od benda *Spice Girls* koja je u prosjeku prepoznata u samo 2,29 sekundi. Slijedi lista pet najbrže prepoznatih pjesama s prosječnim vremenima reakcija (Batchelor, 2014):

1. *Spice Girls - Wannabe*: 2,29 sekunde
2. *Lou Bega - Mambo No. 5*: *2,48* sekundi
3. *Survivor - Eye of the tiger*: 2,62 sekunde
4. *Lady Gaga - Just Dance*: 2,66 sekundi
5. *ABBA – SOS*: 2,73 sekunde

Navedeni rezultati upućuju na to da čovjek na osnovu određene kombinacije tonova raspoređenih u vremenu može prepoznati i razaznati pjesmu od tisuće drugih koji je čuo u životu, praktički istog trena. Iznimno kompleksna pozadina tog procesa u principu radi na

[1] Broj ponavljajućih ciklusa valnog oblika u sekundi

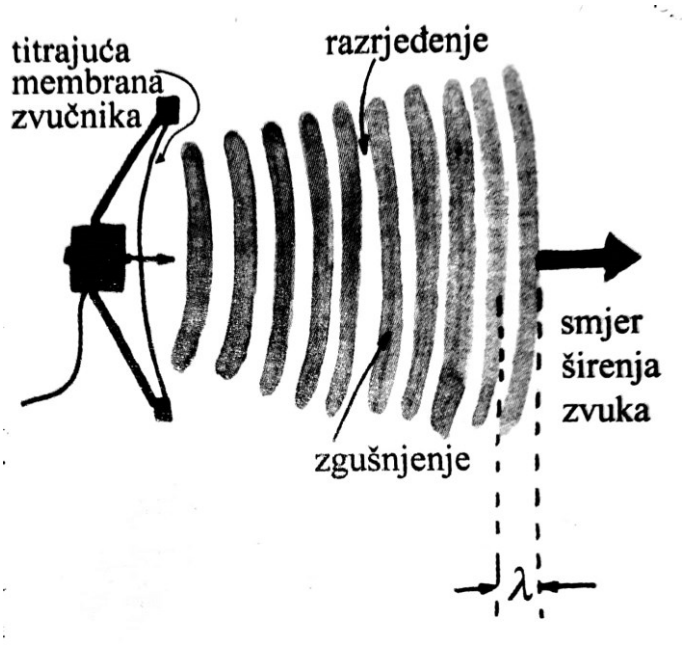
način da zvučni podražaj, to jest dani uzorak, okine bljesak moždanih stanica koje „otključaju“ postojeće podatke. Računalo bi, pošto nema intuitivno razumijevanje glazbe, moralo uzeti uzorak pjesme, te onda usporediti komadić svake pjesme iz baze, koji je dug koliko i uzorak. Stvaranje programske podrške koja može to učiniti brzo predstavlja veliki razvojni izazov. Aplikacija *Shazam* iznimno je poznata i korištena mobilna aplikacija, o čemu govori podatak da ima više od milijardu instalacija na *Apple App Storeu* (Apple, 2023), te više od pola milijarde preuzimanja na *Google PlayStoreu* (Google, 2023). Malo je poznato da je prvobitno funkcionalno rješenje bilo implementirano i dostupno za korištenje već 2002. godine. Kako bi se koristio tadašnji servis, morao se uputiti poziv, te pustiti dio pjesme dug 15 sekundi. Tek je 2008. godine sa dolaskom pametnih telefona tvrtka plasira svoj servis u obliku mobilne aplikacije dostupne na iOS i Android operacijskim sustavima. Rješenje koje su implementirali programeri tvrtke *Shazam* je zadivljujuće, a s tim i vrlo profitabilno, pa nije čudno da nisu svi važni implementacijski detalji otkriveni javnosti. 2003. godine jedan od osnivača i sadašnji glavni znanstvenik (engl. *chief scientist*) tvrtke *Shazam*, Avery Li-Chun Wang, objavljuje kratak članak (Wang, 2003), na kojem se temelji ovo poglavlje, a u njemu on generalno opisuje proces prepoznavanja, te tako daje uvid u osnovne principe rada sustava za prepoznavanje pjesama. Iako je sustav od tada napredovao, same etape koje se moraju proći pri identifikaciji pjesme su nepromijenjene, a uz to i prilično općenite, pa se kao takve, mogu opisati i lako razumjeti. Kako bi sustav bio shvatljiv, samim tim još više zadivljujući, neophodno je osnovno poznavanje fizike zvuka.

2.1. Fizika zvuka

Definicija zvuka objavljena na hrvatskom jezičnom portalu glasi: „valovito kretanje, titraji određene frekvencije koji se primaju osjetilom sluha“. Zvuk se kroz medij širi putem zvučnih valova, koji su longitudinalni. Za razliku od transverzalnih valova, smjer titranja čestica medija kod longitudinalnih valova paralelan je smjeru širenja vala. Izvor zvuka[2] svojim titranjem izaziva repetitivno kretanje čestica u mediju, to jest oscilatora u smjeru širenja zvučnog vala. Na primjeru zvučnika kao izvora zvuka i zraka kao medija, to jest sredstva širenja zvuka, može se predočiti fizika cijelog procesa. Kada se tijekom titranja membrana zvučnika giba prema van, ona potisne susjedne čestice zraka, te tako odmah neposredno uz

[2] Tijelo koje titra, proizvodi zvuk

membranu nastane usko područje zgusnutog zraka. Odmah nakon, zvučna membrana pomiče se u suprotnom smjeru, prema unutra, pa se uz njezinu vanjsku stranu poveća slobodan prostor što zapravo znači da se načas gustoća zraka neposredno kraj vanjske površine membrane smanji (Sl. 2.1). Na opisani način, pri titranju membrane zvučnika nastaju izmjenična područja smanjene i povećane gustoće zraka koja se šire dalje od membrane, a to su zapravo područja povećanog i smanjenog tlaka u zraku, a broj njihovih izmjenjivanja u sekundi jest frekvencija (1) (Paar, 2009).



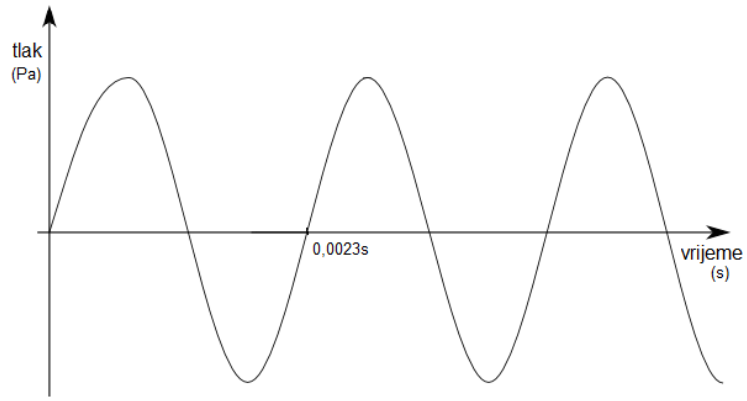
Sl. 2.1 Prikaz promjene tlaka (gustoće) zraka pri stvaranju zvuka na primjeru zvučnika

To izmjenjivanje gustoće zraka rasprostire se zrakom u obliku longitudinalnih valova, te kao takvo sadrži osnovnu karakteristiku vala (ne samo zvučnog) – frekvenciju[1], izraženu formulom (1) i mjernom jedinicom herc (Hz) koja implicira da je vrijeme „T“ izraženo u sekundama (2).

$$f = \frac{1}{T} \tag{1}$$

$$Hz = 1/s \tag{2}$$

Brzina zvuka ovisi o elastičnosti i gustoći sredstva u kojem se širi, a jakost zvuka ovisi o količini energije koja se prenosi zvučnim valovima, te iz nje proizlazi decibelna ljestvica[3] koja označuje razinu jakosti osjeta zvuka ili glasnoću. Navedene karakteristike fizikalno opisuju zvuk, dok je boja svojstvena užem pojmu - tonu. Ton dakle, odlikuje visina, jačina i boja, a definicija glasi da je ton složeni zvuk koji nastaje pravilnim i periodičnim titranjem zraka, a parcijalni su mu tonovi u maksimalno harmoničnom odnosu, stoga mu se može precizno odrediti takozvana tonska visina. Sama definicija tona implicitno ističe njegovu temeljnu karakteristiku, to jest boju. Kada na primjer gitara ili glasovir proizvodi ton „*a*[4]*“*, graf promjene tlaka u vremenu nije poput slike (Sl. 2.2), to jest proizvedeni zvuk nije čisti ton „a“.

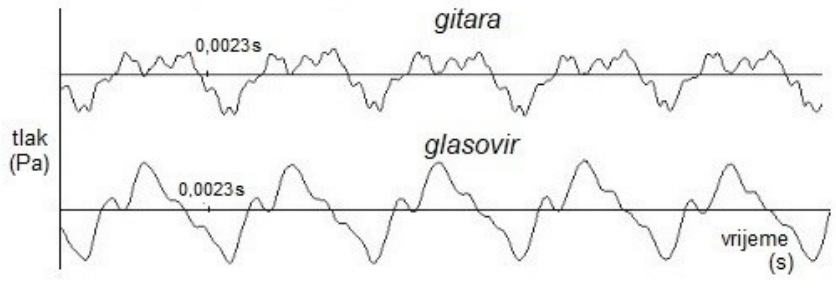


Sl. 2.2 Graf čistog tona „*a“*

Na slici (Sl. 2.3) prikazane su iste frekvencije, koje se razlikuju po obliku, što zapravo znači da se razlikuju upravo po boji. Glazbala ne proizvode samo čisti ton „a“, već i njegove harmonike, što su zapravo zvukovi frekvencija koja je višekratnik osnovnoj frekvenciji.

[3] Ljestvica koja označuje glasnoću, proizašla iz spoznaje da je sami psihofizički osjet zvuka skoro u potpunosti proporcionalan logaritmu jakosti zvuka.
[4] Ton čija frekvencija iznosi 440Hz što je općeprihvaćen dogovor nastao na međunarodnoj konferenciji u Londonu 1939.godine. Svi ostali tonovi klasičnih glazbenih ljestvica teorijski se definiraju iz relativnih odnosa tonom „*a“*.

Osnovni ton i harmonici koje glazbalo proizvodi međusobno interferiraju, pa tako produciraju rezultantni kompozitni val čiji graf može poprimiti oblike kao na slici (Sl. 2.3).



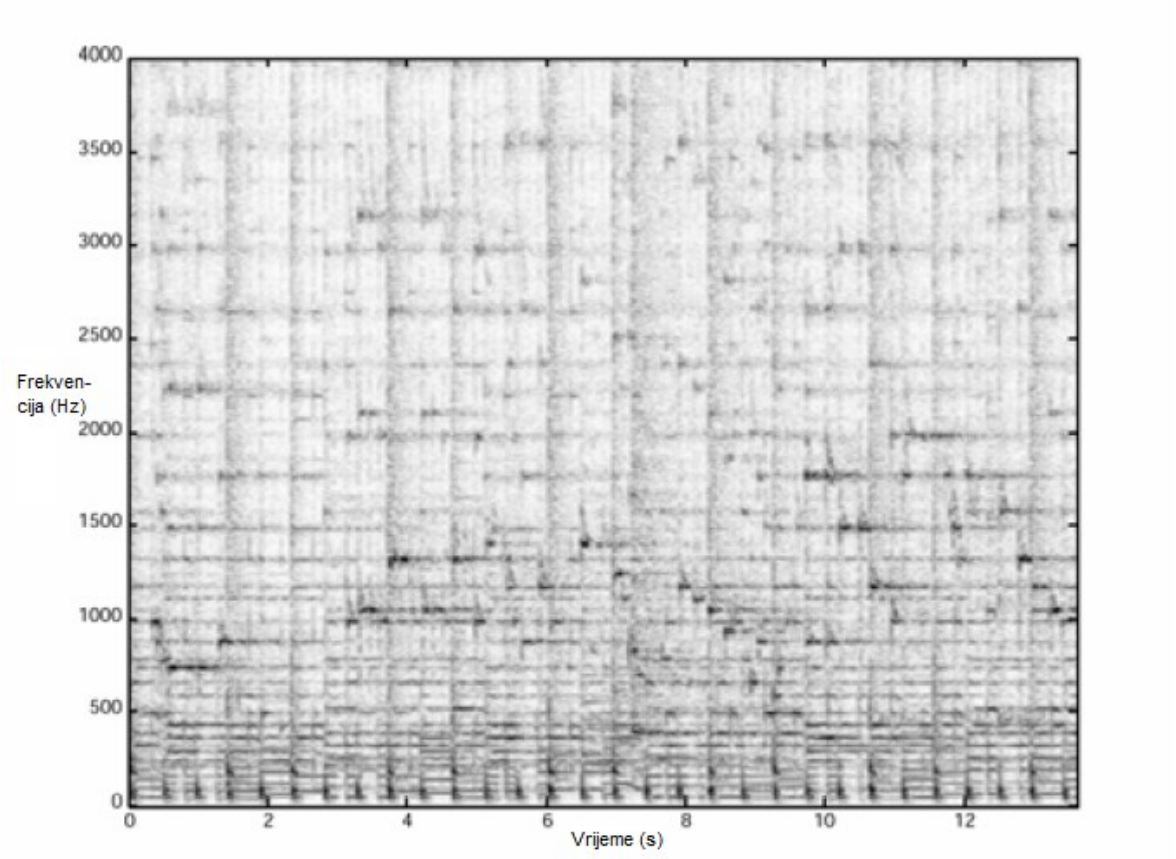
Sl. 2.3Prikaz mogućeg izgleda grafa tona a dobivenog na gitari i glasoviru

Francuski matematičar Jean-Baptiste Joseph Fourier dokazao je da svaki neharmonijski val frekvencije *f* može prikazati kao rezultat interferencije harmonijskih valova čije su frekvencije *f, 2f, 3f, 4f*, …, koje imaju određene amplitude (Paar, 2009). Val osnovne frekvencije uvijek ima veću amplitudu, u usporedbi s harmonicima. Uzimajući u obzir navedenu kompleksnost samo jednog tona, količinu tonova koju jedan instrument proizvodi, količinu instrumenata koju pjesma može imati i to sve samo u jednom djeliću sekunde, teško se može zamisliti kakav bi kompliciran oblik graf mogao poprimiti.

2.2. Proces prepoznavanja

Prethodno poglavlje opisuje zvuk i završava predstavljanjem problema koji uvodi u temelj svakog sustava koji obrađuje zvuk. Kompleksnost tona određuje da takva kratka zvučna ulazna informacija nosi ogromnu količina podataka, stoga se, za njenu vizualnu reprezentaciju koristi poseban trodimenzionalni graf – spektrogram. Prikazani spektrogram

(Sl. 2.4) na svojoj x-osi ima vrijeme, na y-osi frekvencije, a bojom, to jest intenzitetom crne boje izražena je jačina, to jest glasnoća pojedine frekvencije.



Sl. 2.4 Spektrogram

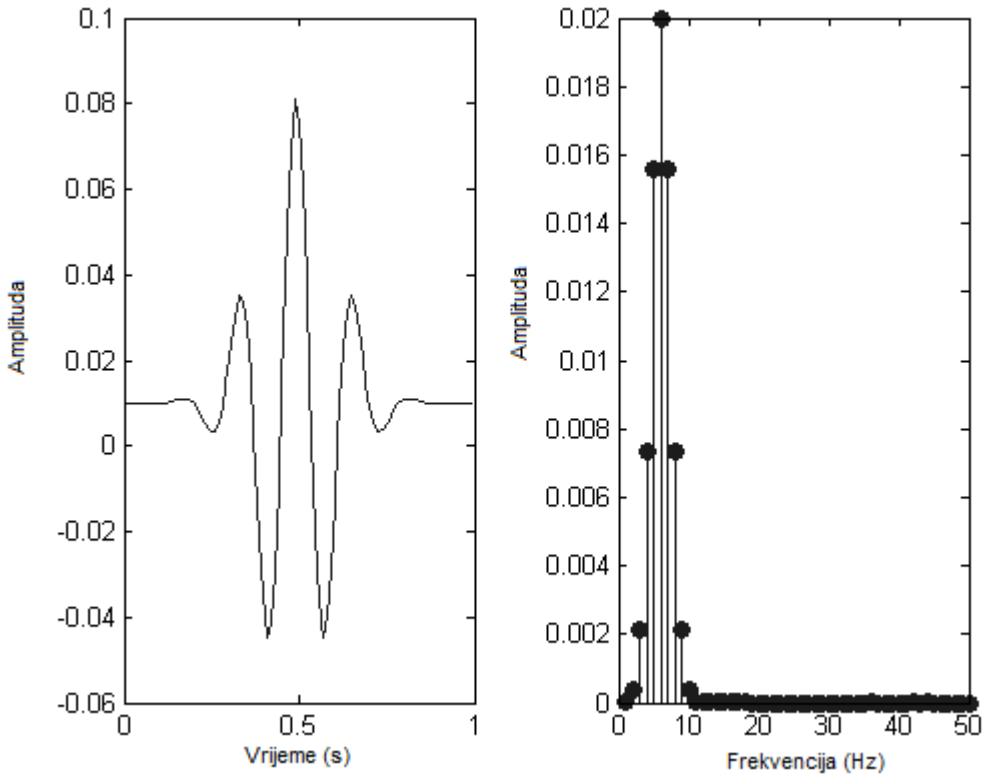
Takav graf je nešto što računalo može obraditi i spremiti kao podatke, ali ovakav bi graf nosio ogromnu količinu informacija, a što je više informacija, duže je vrijeme izračuna u pronalasku podudarnosti. Stoga je neizbježno pokušati smanjiti količinu podataka koja je potrebna za izračun u svrhu smanjivanja vremena izračuna. Smanjenje količine na adekvatan način omogućeno je upravo zbog prethodno spomenutog Fourierovog razvoja. Ulazni signal nastao je od zvučnog vala sačinjenog od mnogo različitih frekvencija. Kao takav može se opisati jačinom (amplitudom[5]) pojedinih frekvencija. Na toj činjenici zasniva se brza Fourierova transformacija (engl. *Fast Fourier transform,* skraćeno FFT). Taj vrlo koristan algoritam ima široku primjenu u području signala. Pretpostavlja se da *Shazam* koristi

[5] Amplituda je najveći otklon od srednje vrijednosti veličine kojom se opisuje val ili titranje

Cooley-Turkey algoritam koji je varijacija FFT algoritma, a ima visoko efikasnu složenost(3).

$$T(n) = O(n \log n) \tag{3}$$

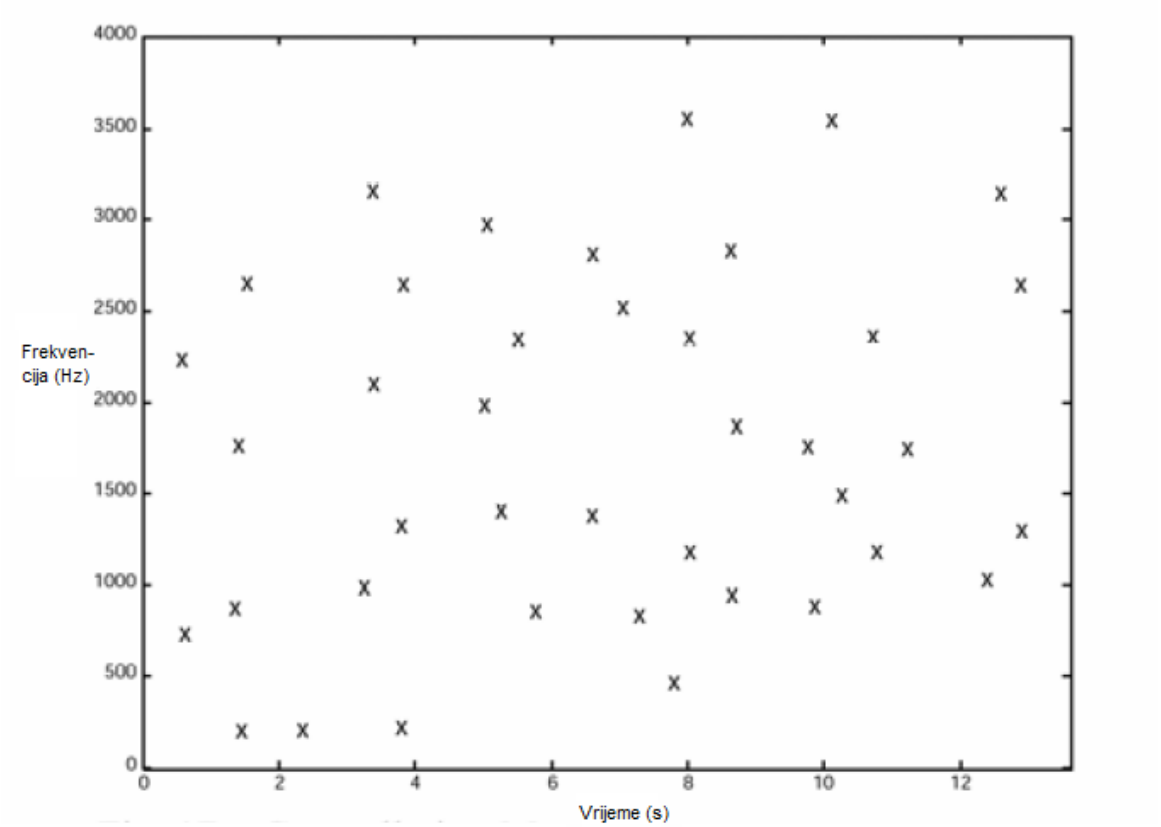
Pomoću njega prijeđe se iz vremenske domene u domenu frekvencija, te se tako zapravo dobije statička reprezentacija dinamičnog signala. Na slici (Sl. 2.5) se može vidjeti isti signal iz različitih gledišta, to jest iz vremenske i frekvencijske domene. Dakle, prikazano je ako signal izgleda prije i poslije FFT obrade (Jovanović, 2015).



Sl. 2.5 Signal prije (lijevo) i nakon (desno) FFT obrade

Računalo tako transformira spektrogram kako bi dobilo graf koji Avery Wang naziva mapa zviježđa (engl. *constellation map*) (Sl. 2.6). Takav graf prikazuje samo frekvencije koje se ističu u pojedinom trenutku, pa se na x-osi nalazi vrijeme, a na y-osi frekvencija. Čineći to,

ne samo da se trodimenzionalni graf sveo na dvije dimenzije, već se i količina podataka na grafu značajno smanjila.



Sl. 2.6 Prikaz istaknutih frekvencija, tzv. mapa zviježđa (engl. *constalation map*)

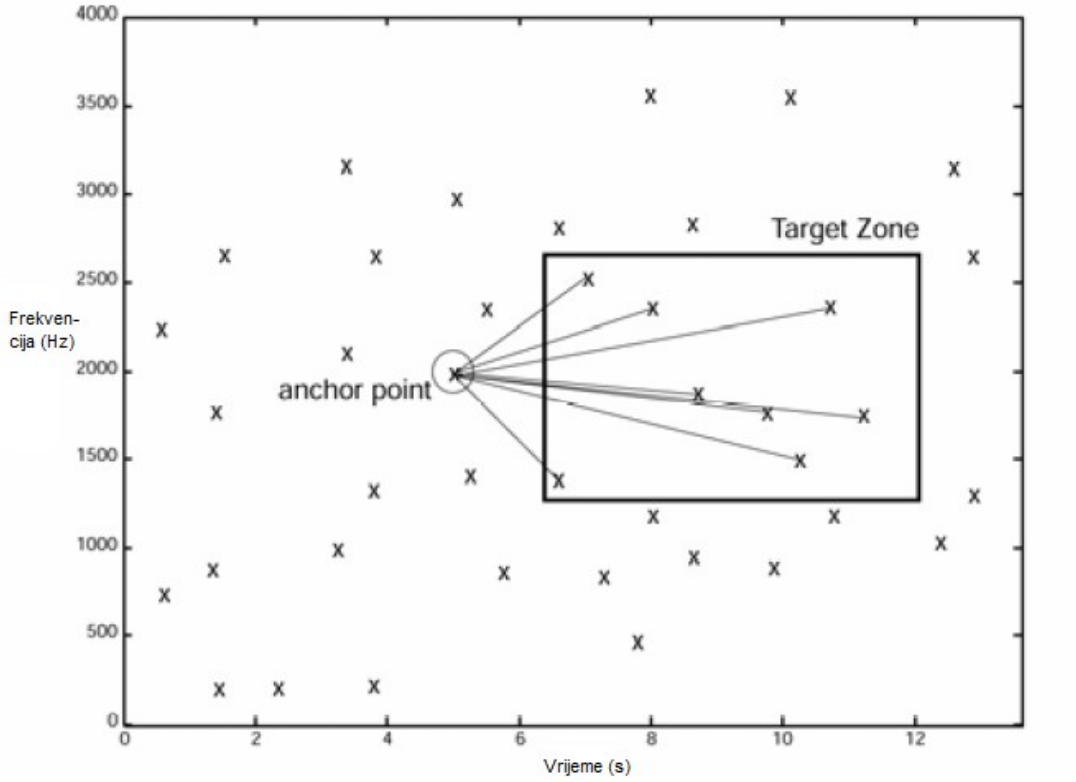
Opisani proces u kojemu se uzorak zvuka svede na pojedine dominantne frekvencije u danom vremenskom rasponu Avery Wang naziva *fingerprinting*. U određenom trenutku danog vremenskog raspona uzima se u obzir najistaknutija frekvencija iz svakog od točno određenih frekvencijskih raspona. Poznato je da aplikacija *Shazam* može prepoznati pjesmu u šumnom okruženu, a upravo navedeni korak je najbitniji pri eliminaciji pozadinske buke, to jest šuma. Svaka pjesma u *Shazam* bazi podataka zapisana je u obliku digitalnog otiska (engl. *fingerprint*). Također, za svaki uzorak pjesme koji se mora prepoznati, mora se prvo izraditi digitalni otisak uzorka. Ključna stvar jest ta, da je digitalni otisak adresa *hash* tablice. Ideja *hashiranja* je da se ulazni entiteti pravilno raspodijele, to jest, jednoliko distribuiraju kroz polje. Svaki element pretvori se u ključ, koji mu se dodijeli i služi za njegov pristup, te se na taj način omoguće pristupanje elementu konstantnom vremenskoj složenosti (4).

$$T(n) = O(1) \tag{4}$$

Hash funkcije iz proizvoljnog broja ulaznih podataka daju rezultat fiksne veličine, koji se koristi za indeksiranje *hash* tablice. Kao takve, iznimno su korisne i imaju široku primjenu u računarstvu. Dobra *hash* funkcija:

- Mora biti što je moguće lakša za izračunati i ne smije sama biti algoritam
- Treba pružiti jednoliku raspodjelu kroz *hash* tablicu
- Mora smanjiti kolizije[6] što je više moguće

Pošto je unikatnost digitalnog otiska važna, jer povećava brzinu pretrage, *Shazam* kreira *hasheve* iz parova istaknutih frekvencija. Informacije od kojih se sastoji *hash* su frekvencija svakog od para i vremenski razmak između njih. Wang predstavlja pojam vučne točke (engl. *anchor point*) koja predstavlja jednu točku (istaknutu frekvenciju) na *mapi zviježđa* na osnovu koje se određuje ciljna zona (engl. *target zone*) (Sl. 2.7). Ciljna zona je površina na grafu iz koje se svaka istaknuta točka sparuje s vučnom točkom.



Sl. 2.7 Prikaz vučne točke s pripadajućom ciljnom zonom na mapi zviježđa

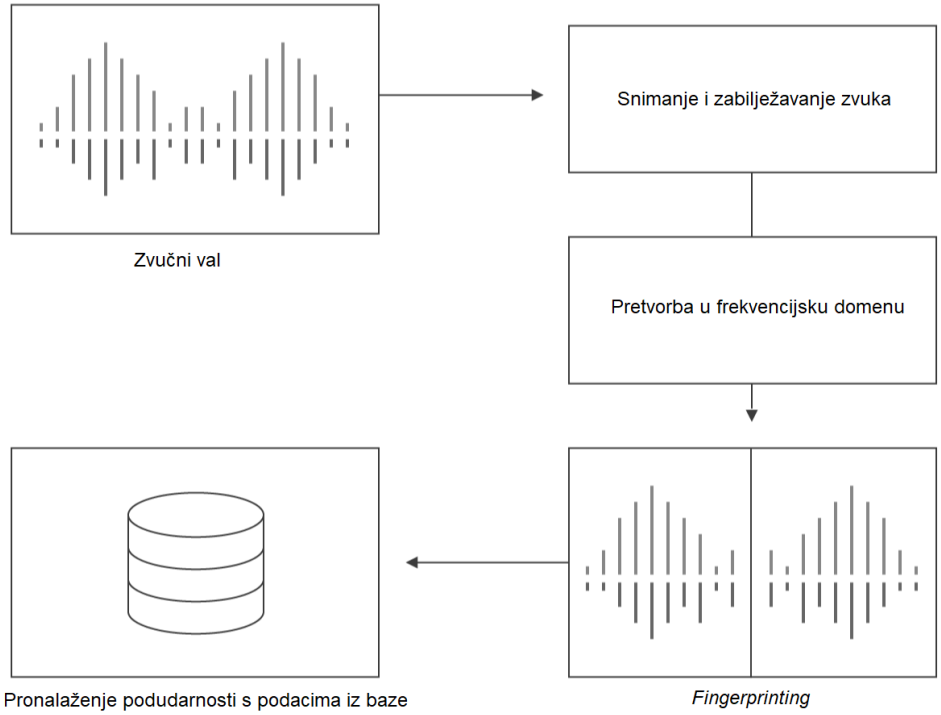
[6] Spremanje vrijednosti pod istu *hash* adresu

Jednom kada je par stvoren, sprema se kao *hash* u bazu, te osim informacija frekvencije svake točke i vremenskim razmakom, spremaju se i dodatne informacije. Dodatne informacije sačinjene su od vremenske oznake trenutka u kojem se pojavljuje vučna točka u pjesmi, te od identifikacijskog broja pjesme. U kontekstu *hashiranja,* te dodatne informacije nisu bitne, već one imaju ulogu kod pronalaska podudaranja s uzorkom. Svi *hashevi* određene pjesme čine njen digitalni otisak. Svaka pjesma u *Shazam* bazi podataka mora proći navedeni proces kako bi se mogla pronaći. Za uzorak pjesme se također stvara njegov digitalni otisak kako bi se mogla naći podudarnost, te onda nakon njegovog generiranja, slijedi pretraga. Pretraga započinje tako što se dohvate svi *hashevi* iz baze koji se podudaraju s upravo kreiranim digitalnim otiskom uzorka. Dohvaćeni podaci grupiraju se po pjesmi. To je moguće zahvaljujući prijašnje spomenutim dodatnim informacijama koje su povezane sa svakim *hashom*. Za svako potencijalno podudaranje pjesme, vrše se kompleksni proračuni potpomognuti saznanjem apsolutnog trenutka pojavljivanja koje je povezano s *hashom,* koji kao produkt stvaraju histogram[7]. Pjesma s najvećom vrijednošću je rezultat.

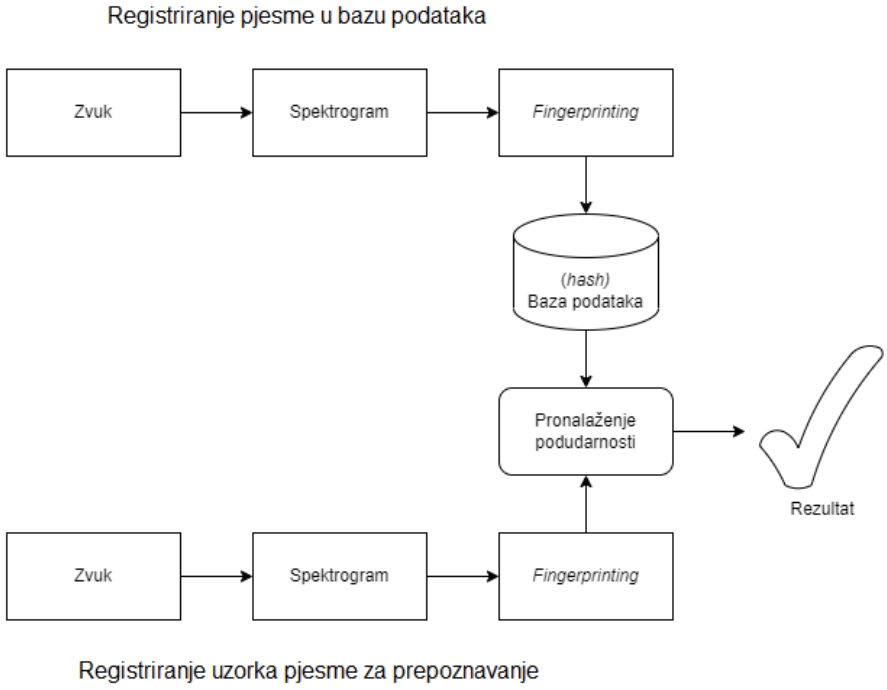
Iz perspektive korisnika, mora se prvo zabilježiti zvuk pjesme iz njegova okruženja, mikrofonom uređaja. Mikrofon pretvori mehaničku energiju zvučnog vala u električni signal i dobije njegovu digitalnu reprezentaciju. Signal se dalje obrađuje, pa se naposlijetku dobije frekvencijska domena signala koja je pogodna za izrađivanje digitalnog otiska. Dobiveni

[7] Reprezentacija distribucije numeričkih podataka jedne varijable, u ovom slučaju razine podudarnosti raspodijeljene po potencijalnim pogodcima

podaci šalju se mrežom. Na strani poslužitelja vrši se uspoređivanje s podacima iz baze podataka *Shazama*, kako bi se dobio rezultat o kojoj je pjesmi riječ (Sl. 2.8).



Sl. 2.8 Proces slanja uzorka pjesme u svrhu identifikacije



Sl. 2.9 Cjelokupni proces koji omogućava identifikaciju pjesme

Pri registriranju pjesama u bazu izračuna se spektrogram pjesme, te se izvuku magnitude iz njega. Te dominantne točke se spare u *hasheve*. Kolekcija tih *hasheva* čini digitalni otisak pjesme. Pri prepoznavanju pjesama izračuna se digitalni otisak uzorka. Nađu se *hashevi* u bazi koji odgovaraju, pa se za svaki *hash* izvrši dodatna analiza koja rezultira histogramom, to jest setom podataka poredanih po stupnju podudarnosti (Sl. 2.9). *Shazam* koristi *MongoDB* bazu podataka. *MonogoDB* baza podataka je nerelacijska baza podataka. Nerelacijska baza podataka, za razliku od relacijske, nema strogo definiranu strukturu podataka i ne prati relacijski model, stoga ime navedene karakteristike:

- Lako nadogradiva
- Iznimno skalabilna
- Zahtjeva manje održavanja
- Pogodna za velike količine podataka
- Nije strukturirana
- Namijenjena *cloudu*

MongoDB nije nimalo iznenađujući odabir kad se uzme u obzir velika potreba za skalabilnošću, a i nema osobite potrebe za relacijama jer je podatkovni model poprilično jednostavan.

Iako ovaj podosta pojednostavljeni opis identifikacije pjesama može dati općenitu, to jest širu sliku cijelog procesa, itekako može predstaviti samu genijalnost rješenja. Kada bi se morala dati vrijednost ovog intelektualnog vlasništva, vrijednost bi iznosila 400 milijuna dolara, upravo onoliko koliko ju je tvrtka *Apple* platila 2018. godine (Lunden, 2018). Pjesme općenito nalikuju jedna drugoj, pa čak i do razine plagijata. Samim tim komadići pjesama još više nalikuju jedni drugima zbog prirode samog stvaranja glazbe, gdje se uzimaju već postojeće melodije i aranžiraju tako da čovjeku nije direktno očito da je riječ o istoj temi. Uz to, pjesma se može sastojati od više djelića preuzetih iz različitih postojećih pjesama. Uzimajući u obzir još i neadekvatne uvijete snimanja uzorka ispunjene šumom, vidi se fascinirajuća otpornost na pogreške i točnost identifikacije pjesama.

3. Opis rješenja

Chordiato omogućava pristup tekstualnim materijalima potrebnima za sviranje na dva načina. Aplikacija tako, sadrži dvije osnovne funkcionalnosti. Pristup glazbenim zapisima dobiveni na osnovu zvuka glazbe iz okoline. Rezultati se spremaju, pa se njima može upravljati na način da se označe kao favoriti, kako bi se onda samo klikom gumba dohvatio glazbeni zapis koji je već jednom bio prepoznat.

3.1. Prepoznavanje pjesme

Pjesme se prepoznaju korištenjem *Shazam* servisa čiji je algoritam prethodno opisan. Zabilježi se uzorak pjesme dug 4 sekunde, koji se enkodira u *base64* tekst i šalje na javno dostupni servis, koji vrati JSON objekt s podacima o pjesmi. Dobiveni podaci se obrade, te

se uz pomoć *Serp* udaljenog servisa dohvati potrebno web mjesto stranice *UltimateGuitar*, koje se onda prikaže na glavnom ekranu (

).

Sl. 3.1 Glavni ekran Chordiato aplikacije

Akcija koja pokrene navedeni proces je klik gumba označenim mikrofon sličicom u donjem desnom kutu iznad navigacijske trake. Glavni ekran se prikazuje kad se aplikacija pokrene, pri čemu web mjesto nije učitano, pa se tada prikaže pozadinska slika (Sl. 3.2).

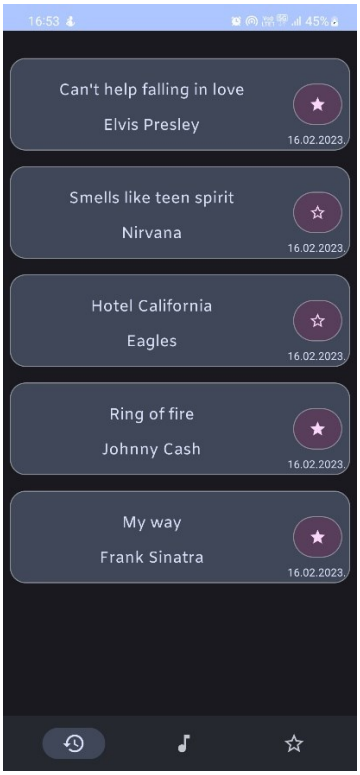
Sl. 3.2 Početan ekran prikazan nakon pokretanja aplikacije

Na glavni ekran se osim navigacijom može doći odabirom pjesme iz spremljenih favorita, pa se onda web mjesto odmah učita.

3.2. Upravljanje favoritima

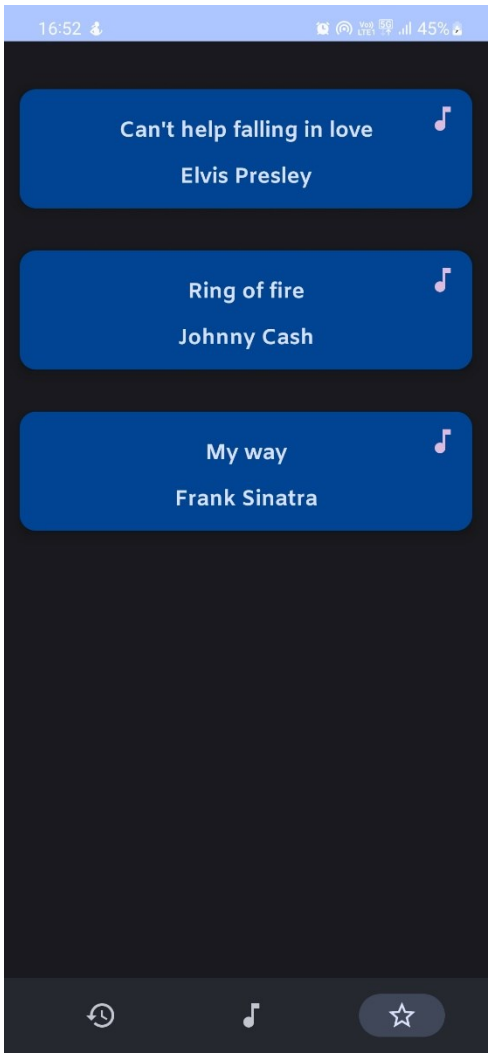
Kako se ne bi svaki put za dohvaćanje iste pjesme morao prepoznavati zvuk, implementirano je spremanje u lokalnu bazu podataka. Prilikom svake uspješne pretrage spreme se potrebni podaci, koji uključuju jedinstveni resursni lokator (engl. *Uniform Resource Locator,* skraćeno URL) stranice koji je potreban za prikazivanje web mjesta. Spremljeni podaci prikazuju se u dva oblika s dvije različite namjene. Svaka pretraga pojedine pjesme zabilježi se i vidljiva je u ekranu lijevo na navigacijskoj traci, koji zapravo služi za upravljanje favoritima (Sl. 3.3). Tu se mogu omiljene pjesme mogu dodavati i uklanjati, a osim naziva

pjesme i izvođača, za svaku pjesmu prikazan je datum zadnjeg pristupa pjesmi, odnosno njenog sviranja. Prikazane pjesme sortirane su po datumu zadnjeg pristupa.



Sl. 3.3 Ekran povijesti pretraživanja koji služi za upravljanje favoritima

Dakle, ako se prepozna ista pjesma dva puta, neće se ponovno spremati, već se za postojeću pjesmu ažuriraju vrijednosti zadnjeg pristupa, te je ona na vrhu liste. Dodavanje i uklanjanje favorita poprilično je intuitivno jer se samim klikom na pjesmu njeno stanje promijeni, to jest ako je bila favorit, više nije i obrnuto.



Sl. 3.4 Ekran spremljenih favorita

Na ekranu označenim ikonom zvijezde, desno na navigacijskoj traci, nalaze se svi odabrani favoriti (**Pogreška! Izvor reference nije pronađen.**). Klikom na jedan od favorita, korisnik se preusmjerava na glavni ekran, gdje se web mjesto s tekstom i akordima ili tablaturama učita i prikaže.

4. Usporedba s postojećim rješenjima na tržištu

Chordiato je namijenjen bilo kome tko želi svirati, bio on početnik, amater ili profesionalac. Ona je prije svega alat, no kada bi se morala kategorizirati po kategorijama koje su ponuđene na *PlayStoreu*, spadala bi u kategoriju *glazba i zvuk*. Ona je i edukacijske prirode, jer se pomoću nje uči i savladava vještina sviranja. Pomoć pri uvježbavanju sviranja kroz aplikacije može se ostvariti na razne načine, na primjer Yousitian aplikacija prati korisnikovu izvedbu u realnom vremenu i onda daje povratne informacije o točnosti. Jednostavna metronom aplikacija također je primjer jednostavnog alata pri sviranju koji uzastopnim, jednako vremenski razmaknutim otkucajima pomaže korisniku održavati ritam. Za dobru usporedbu, treba specifičnija slična aplikacija, koja ima donekle slične funkcionalnosti, a najbliže tome dolazi Chordify. Chordify je aplikacija s više od 5 milijuna preuzimanja na *PlayStoreu.* Ima i svoju web inačicu koja ima iste funkcionalnosti. Naime, Chordify može dati akorde bilo koje pjesme koja je na Youtube servisu. Aplikacija nema sve te podatke u bazi, već to radi pomoću duboke neuronske mreže (engl. *deep neural network*). Umjetna inteligencija istrenirana je na velikom broju pjesama odnosno spektrograma. Dakle, iz digitalnog zapisa glazbe, izradi se spektrogram na osnovu kojeg AI pruži akorde, tako da to može učiniti za svaku pjesmu koja je ili koja će biti na Youtubeu (Witte, 2021). Način rada ne garantira točnost, tako da se zna dogoditi da akordi nisu u potpunosti točni, za razliku od Chordiata koji podatke dobiva od stranice *UltimateGuitar,* najpouzdanije stranice za akorde, čiji se sadržaj provjerava, te je za veliki broj pjesama garantirana stopostotna točnost. Prilikom korištenja aplikacije, može se pustiti pjesma iz ugrađenog Youtube *playera.* Chordify ima mogućnost spremanja sadržaja i mjesečna pretplata iznosi osam dolara. Aplikacija slične funkcionalnosti je Songsterr. Ona ima bazu od preko osamsto tisuća pjesama zapisanih u obliku tablatura za gitare, bas gitare i bubnjeve. Imajući bazu podataka, može pružiti funkcionalnosti poput odabira brzine izvođenja, ili popratne matrice u kojoj se mogu odabrati instrumenti koji se čuju. Dio sadržaja napisali su sami korisnici, tako da glazbeni zapisi ne moraju biti nužno točni. Sadrži opciju spremanja favorita i košta deset dolara mjesečno. Obje aplikacije nemaju mogućnost pružiti sadržaj na osnovu zvuka dobivenog iz okoline, to jest zvuka zaprimljenog mikrofonom uređaja. To im omogućava više drugih funkcionalnosti koje stvaraju dobro korisničko iskustvo, o čemu govore njihove ocjene na *PlayStoreu*. S (Lunden, 2018) pedeset tisuća osvrta Songsterr ima

ocjenu 4.6 (Google, 2023; Apple, 2023), a Chordify s trideset tisuća osvrta ima ocjenu 4.4 (Google, 2023).

Tablica 4.1 Usporedba karakteristika aplikacija Chordify, Songsterr i Chordiato

Aplikacije	Spremanje favorita	Prepoznavanje zvuka iz okoline	Web inačica	Ugrađena popratna glazba
Chordify	Ima	Nema	Ima	Pjesma s youtubea
Songsterr	Ima	Nema	Ima	Prilagodljiva matrica
Chordiato	Ima	Ima	Nema	Nema

Uzimajući u obzir da su Chordify i Songsterr uistinu najsličnija postojeća rješenja, tablica baš to i ne ističe. Iz tabličnog prikaza (Tablica 4.1) usporedbe karakteristika svake od navedenih aplikacija, može se izvući očiti zaključak da su Chordify i Songsterr vrlo bliske, dok se Chordiato uvelike razlikuje skoro po svakom parametru.

5. Izvedba i implementacija aplikacije

Kako bi Chordiato postalo funkcionalno implementirano rješenje, potrebno je bilo korištenje raznih alata, servisa i postojećih rješenja koji se koriste u programiranju. Ovo poglavlje obuhvaća sve bitne komponente Chordiato aplikacije. Kreće se od platforme za koju je Chordiato izrađen, radnog okvira i programskog jezika (5.1). Zatim se objašnjava korištenje alata i izvora koji su imali ulogu u ostvarivanju izgleda korisničkog sučelja (5.2). Navedeno prethodi kako bi se implementirano rješenje moglo dekonstruirati, detaljno analizirati i potkrijepiti primjerima koda iz projekta (5.3).

5.1. Korištene tehnologije

Chordiato je mobilna aplikacija namijenjena isključivo za Android platformu, napisana u Kotlin programskom jeziku, u integriranom razvojnom okruženju (engl. *integrated development environment*, skraćeno IDE) Android Studio. Android je širok pojam koji ovisno o kontekstu može imati mnoga značenja, no opisan jednostavnim definicijama, Android je u suštini:

a. Operacijski sustav otvorenog koda (engl. *open source*)
b. Platforma otvorenog koda za razvoj mobilnih aplikacija

Android platforma najkorištenija je mobilna platforma na svijetu s trenutno više od 2.5 milijarde korisnika. Tržišni udio u domeni operacijskih sustava za mobitele iznosi 71.74%, dok drugo mjesto zauzima *Appleov* iOS sa 27,63% (GlobalStats, 2023.). Android operacijski sustav baziran je na Linux jezgri (engl. *Linux Kernel*) na kojoj se vrti *Android runtime* okruženje, koje zajedno s nativnim bibliotekama omogućuju aplikacijskom okviru stvaranje komponenti za razvoj aplikacija, koje se pružaju aplikacijskom sloju. Tvrtka *JetBrains* 2016. godine izdaje prvo stabilno izdanje programskog jezika Kotlin. Primarni cilj Kotlina je da bude konciznija, produktivnija i sigurnija alternativa Jave. Najčešću primjenu ima u razvoju mobilnih aplikacija za Android uređaje, ali je i popularan u izradi koda poslužiteljske strane. Interoperabilan je s Java kodom i odlično funkcionira s postojećim bibliotekama i radnim okvirima, a uz to ima jednake performanse kao i Java. Kotlin je statično pisani jezik (engl. *statically typed*) što znači da se svaki izraz (engl. *expression*) zna u vremenu kompajliranja (engl. *compile time*), a to ima sljedeće prednosti:

- Performanse – pozivanje metoda je brže jer se u vremenu izvođenja (engl. *runtime*) ne mora tražiti koja metoda se treba pozvati
- Pouzdanost – kompajler provjerava točnost, stoga je manje urušavanja programa u vremenu izvođenja
- Održavanje – rad s nepoznatim kodom je lakši jer se može vidjeti s kakvim objektima kod radi
- Podržavanje alata – korištenje mnogo IDE mogućnosti poput refaktoriranja ili predlaganja nadopune koda

Kotlin je objektno orijentirani programski jezik[8], ali je ujedno i funkcionalan jer posjeduje koncepte funkcionalne paradigme:

- Funkcije prvog reda (engl. *first-class functions*) funkcije se mogu spremati u varijable, slati kao parametri i vraćati kao rezultat drugih funkcija
- Nepromjenjivost – radi se s nepromjenjivim (engl. *immutable*) objektima, što garantira da se njihovo stanje neće promijeniti nakon stvaranja.
- Neposljedičnost (engl. *no side effects*) – koriste se čiste funkcije koje vraćaju isti rezultat za iste ulazne parametre i ne mijenjaju stanje ostalih objekata

Kotlin ima razne funkcionalnosti koje smanjuju često ponavljajući (engl. *boilerplate*) kod, ima bogati set aplikacijskih programskih sučelja (engl. *application programming interface,* skrećeno API) za rad s objektima i kolekcijama u stilu funkcionalne paradigme. Ima vrlo dobar način pisanja asinkronog koda pomoću *coroutina*, a i ima dobar pristup pri rješavanju takozvane „iznimke od milijardu dolara“, to jest *null pointer exeptiona*. Ovaj besplatni programski jezik otvorenog koda (engl. *open source*) ima gomilu iznimno dobrih i korisnih funkcionalnosti, a uz sve to, nije sporiji od Jave (Jemerov & Isakova, 2017). Android studio razvijen je od strane *Googlea* u suradnji s *JetBrainsom* pa je baziran na *InteliJ IDEAu.* Sadrži set uređivača koda (engl. *code editors*), alata i šablona (engl. *templates*) za razvoj korisničkog sučelja koji znatno olakšavaju i ubrzavaju proces izrade aplikacija. Posjeduje *Android Software Development Kit (SDK)* koji je nužan za izrađivanje svake aplikacije. Android SDK posjeduje Android izvorne datoteke (engl. *source files*) i kompajler koji kompajlira kod u Android format (Griffiths & Griffiths, 2021).

[8] Objektno orijentirana paradigma temelji se na organizaciji programske podrške oko strukturiranih objekata, te kao takva, posjeduje principe: enkapsulacije, abstrakcije, nasljeđivanja, polimorfizma.

Razvoj aplikacija u Android okruženju iziskuje od programera posebno obraćanje pažnje i opreznost pri konstruiranju rješenja. Android operacijski sustav optimiziran je za rad s uređajima koji imaju podosta ograničene memorijske i procesorske resurse, te nemaju konstantno napajanje, već koriste bateriju. Baterija je najveće hardver ograničenje mobitela, te se sa strane programske podrške mora, što je bolje moguće, upravljati procesima. Kako bi optimizirao rad uređaja koji je veličine dlana, Android operacijski sustav učestalo obustavlja procese i Android komponente, u bilo kojem trenutku. Zbog takvog okruženja moguće je da komponente bivaju kreirane ili uništene samostalno, bez redoslijeda kojim je programer odredio, to jest zamislio da će se odvijati. Zato je bitno da se podaci ili stanja ne pohranjuju u komponentama, te da komponente ne zavise jedna o drugoj. Glavne Android komponente su:

- Aktivnost (engl. *activity*) – komponenta koja određuje korisničko sučelje, omogućuje interakciju s korisnikom i predstavlja jedan ekran u aplikaciji
- Servis (engl. *service*) – omogućuje pozadinski rad aplikacije, to jest njome se izvršavaju dugotrajne operacije dok sama aplikacija nije vidljiva
- Prijamnik emitiranja (engl. *broadcast reciever*) – komponenta omogućuje komunikaciju između aplikacije i drugih aplikacija ili Android operacijskog sustava
- Pružatelj sadržaja (engl. *content provider*) – komponenta kojom se pružaju podaci drugim aplikacijama

Chordiato od navedenih glavnih komponenata sadrži samo aktivnost, ali sadrži i druge, sporedne i nezaobilazne komponente poput fragmenta (engl. *fragment*)[9], pogleda (engl. *view*)[10], *intenta*[11].

5.2. Dizajn

Izrada kvalitetnog korisničkog iskustva (engl. *user experience,* skraćeno UX) i korisničkog sučelja (engl. *user interface*, skraćeno UI) u poslovnom svijetu vrlo je važna, jer se tako, prije svega, optimalno zadovoljavaju zahtjevi projekta i smanjuju troškovi razvoja. Od same

[9] Fragment je dio korisničkog sučelja određen izgledom i logikom ponašanja, sadrži vlastiti životni ciklus i mora se nalaziti unutar aktivnosti.
[10] Vidljivi komadić korisničkog sučelja definiram izgledom u XML formatu i ponašanjem određenim u aktivnosti ili fragmentu (npr. gumb, polje za unos).
[11] Mehanizam komunikacije između komponenata iste aplikacije i između komponenata drugih aplikacija

početne ideje Chordiato aplikacije, postojala je i zamisao kako će sučelje izgledati, te na koji način će se korisniku servirati funkcionalnosti i sadržaj. Zbog toga, a i zbog količine funkcionalnosti, izrada dizajna u nekom od UX/UI dizajn alata, bila bi suvišna. Svejedno, aplikacija bi svakako trebala imati vizualni identitet u vidu palete boja, izgleda ikona i vizualnih komponenti, te jedinstveni logotip. Za kreiranje unikatnog logotipa, iskoristila se diskutabilno najnaprednija tehnologija današnjice. Nezaobilazna činjenica jest ta, da je čovječanstvo zakoračilo u doba umjetne inteligencije. Po nekima zastrašujuće doba koje je tek u začetcima, neosporivo nosi revolucionarne mogućnosti kakve ljudska rasa još uvijek teško može pojmiti. Očito je da će veliki dio IT industrije u budućnosti surađivati s takvim modelima, stoga je bio logičan odabir iskoristi pristup najmodernijoj tehnologiji pri izradi završnog rada, konkretno, pri izradi vizualnog identiteta Chordiato aplikacije. Servis koji pomoću umjetne inteligencije generira slike je MidJourney, koji pruža dvadeset i pet besplatnih upita na osnovu kojih generira slike. Upućeni opisi nisu odmah davali zadovoljavajuće slike kao rezultate. Nakon par poslanih upita, generira se prigodni logotip, na temelju kojega se onda mogla izraditi paleta boja koja će sačinjavati vizualni identitet (Sl. 5.1)

Sl. 5.1 Generirani logo nastao upitom: *„logo, letter C, circular, round, guitar, music related, dark tones, minimalistic“*

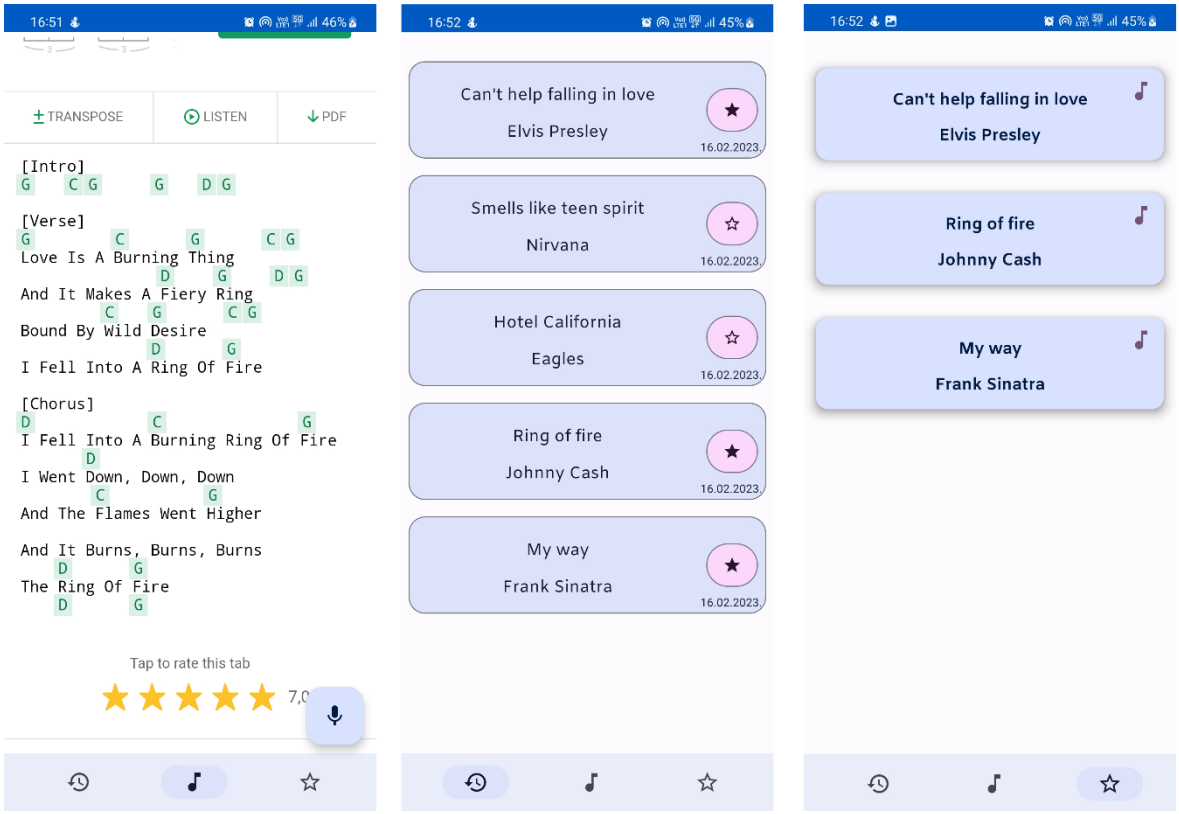
Sa saznanjem o boji, također je bilo moguće napraviti upit s (UXtools, 2022) navedenom bojom, čija će generirana slika biti pozadina aplikacije pri pokretanju (Sl. 3.2). Zaključak koji je proizašao iz rada s generatorom slika jest taj, da kako bi se postigli željeni rezultati, treba biti što specifičniji u upitima, a to znači da se treba točno znati što se traži. Može se povući poveznica sa situacijom kada klijenti koji žele proizvod, ne znaju skroz u detalje što zapravo trebaju, samo u tom slučaju, postoji kadar ljudi koji pomaže pri definiranju projekta. Iz navedenog iskustva, može se izvući pretpostavka da će u doglednoj budućnosti UI/UX

dizajneri i grafičari biti osposobljeni za slanje adekvatnih upita takvom modelu. Možda se tako izrada sučelja, grafika i sl. svede na slanje optimalnih upita, te selekciju dobivenih rezultata. Figma je trenutno najpopularniji alat za UI/UX dizajniranje (UXtools, 2022), te se iskoristio za kreiranje palete boja. Koristeći *Material Design* dodatak (engl. *plugin*), generiraju se boje, temeljene na boji logotipa, koje će biti tema Chordiata.

Sl. 5.2 Paleta boja kreirana pomoću *Material design* dodatka u Figma programu

Material design je širok pojam koji obuhvaća set pravila i naputaka za kreiranje korisničkih sučelja, ima mnoge dodatke i softver alate za razne platforme, popraćene opsežnom dokumentacijom i principima korištenja. Korisničko sučelje Chordiata izrađeno je u skladu s pravilima i uputama koje pruža *Material design*. Sve boje definirane su u pripadajućoj resursnoj datoteci, koja je povezana s resursnom datotekom za teme aplikacije (više o resursima u poglavlju 5.3). To omogućava izgled prilagođen za svijetli i tamni način prikaza

na uređaju. Stoga za svaki ekran aplikacije prikazan na slikama (Sl. 3.1), (Sl. 3.2), (Sl. 3.3), postoji varijanta za svijetli način rada (Sl. 5.3).



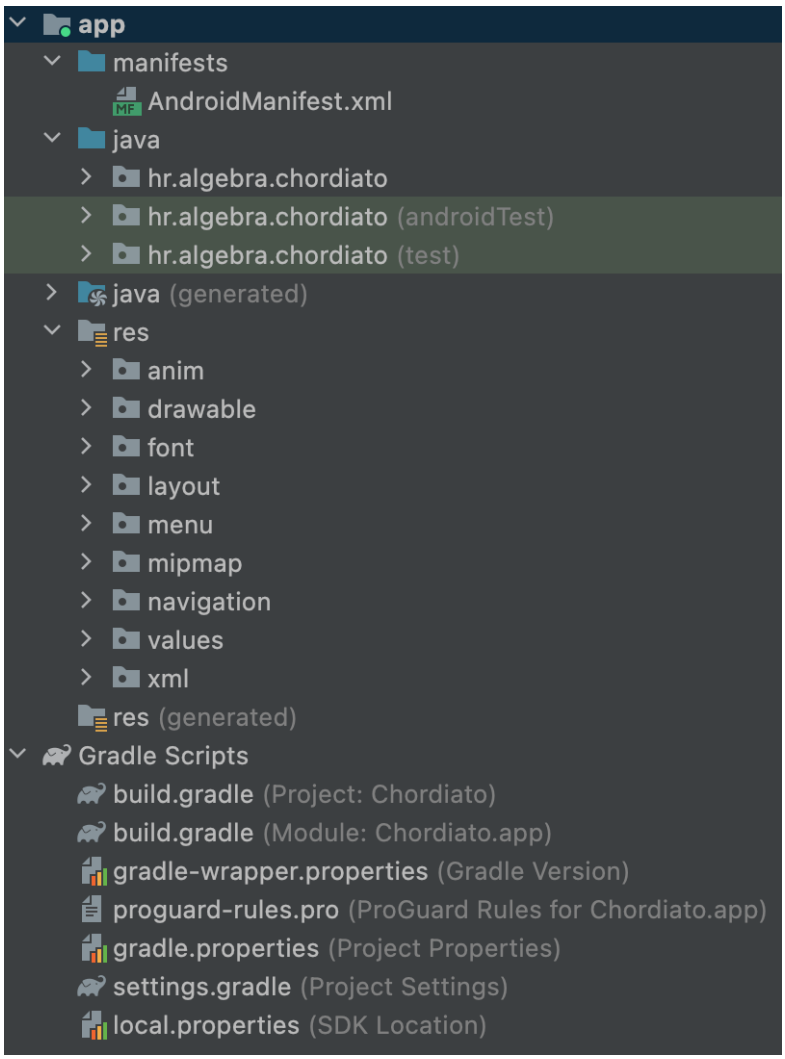
Sl. 5.3 Ekrani Chordiato aplikacije u svijetlom načinu rada uređaja

Korisničko sučelje izrađeno je korištenjem *Android Views* pristupa, a to znači da se izgled svakog vizualnog elemenata određuje i definira u pripadajućoj XML datoteci, koja može biti povezana s funkcionalnom komponentom Androida putem *view bindinga* (5.3.1).

5.3. Arhitektura implementiranog rješenja

Narodna mudrost glasi da je dobra organizacija pola posla. Ako se problemu pristupi s dobrim, razrađenim planom, izrazito su veće šanse za uspjeh. Stoga, dobra razrada i dizajniranje optimalne arhitekture snosi izrazitu važnost pri izradi bilo kakvog rješenja. Arhitektura i dizajn sustava imaju posebnu važnost u poslovnom svijetu, gdje implementacija temeljena na dobroj arhitekturi znači skalabilno i održivo rješenje, koje donosi enormne uštede kroz vrijeme. Robert C. Martin navodi da je cilj arhitekture minimizirati ljudske resurse potrebne za održavanje potrebnog sustava (Martin, 2017).

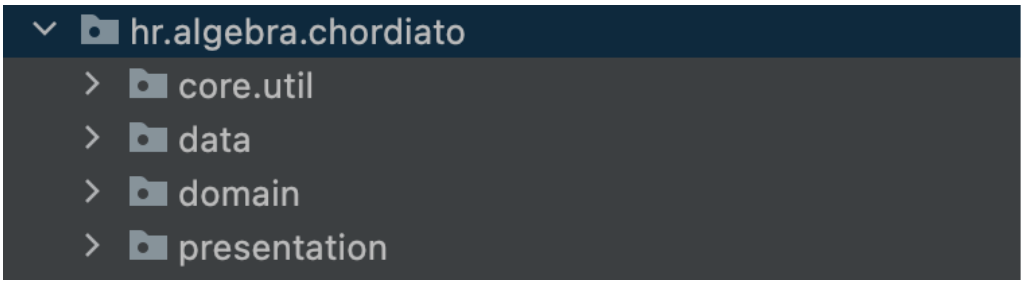
Da bi se mogla opisati arhitektura sustava, potrebno je upoznati se prvo s arhitekturom i strukturom okruženja u kojem se rješenje izrađuje (Sl. 5.4) .



Sl. 5.4 Struktura projekta u Android Studiu

Svaki projekt mora sadržavati `AndroidManifest.xml` datoteku, koja sadrži metapodatke o aplikaciji, a to između ostalog uključuje: verziju aplikacije, naziv, klasu i konfiguraciju glavnih komponenti, razine SDK-a, dozvole sustava. U *hr.algebra.chordiato* paketu nalazi se izvorni kod, dok se u java paketu nalazi pozadinski generiran kod, na primjer u Chordiato aplikaciji generiran od strane *Room* biblioteke. Osim slika, fontova i audio ili video sadržaja, svi resursi koji se nalaze u *res* paketu, zapisani su u XML formatu. Resursi sadržavaju razne datoteke, koje sadrže statične vrijednosti raznih vrsta, atributa i namjena. Oni su odvojeni od izvornog koda čime se postiže lakše održavanje. Pristup elementima datoteka omogućen je putem identifikatora. Između ostalog, pod resurse spadaju animacije (paket *anim*), vektorske grafike (paket *drawable*), tekstualne vrijednosti, boje, stilovi,

dimenzije (paket *values*). Čak je i izmjenjivanje fragmenata, to jest navigiranje kroz aplikaciju definirano XML formatom (paket *navigation*), kao i ponašanje glavnog izbornika (paket *menu*) (Google, 2023). Izgled svih fragmenata, aktivnosti ili drugih objekata tipa *ViewGroup*[12] definiran je u *layout* paketu. Android studio koristi Gradle kao moćni alat za naprednu automatiziranu izgradnju aplikacija (engl. *automated build toolkit*). On omogućuje postavljanje zavisnosti (engl. *dependencies*), brzu konfiguraciju, testiranje, potpisivanje[13], generiranje više različitih APK-ova[14].



Sl. 5.5 Osnovna struktura Chordiato aplikacije

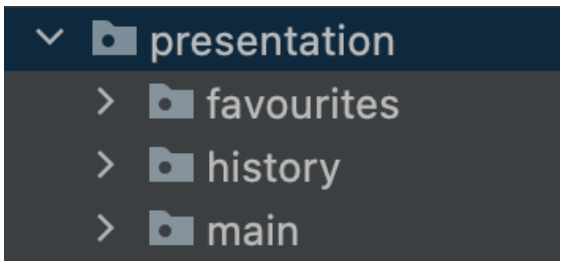
Arhitektura Chordiato aplikacije sačinjena je od tri sloja. Prateći i primjenjujući dobre prakse, Chordiato je izrađen jasnom raspodjelom po slojevima. Preporučljivo je strukturu podijeliti prvo po funkcionalnostima, pa onda po slojevima, no s obzirom na količinu funkcionalnosti Chordiata, prikazana paketna raspodjela (Sl. 5.5) je optimalna.

5.3.1. Prezentacijski sloj

Prezentacijski sloj dodirna je točka koja spaja korisnika s aplikacijom. Uloga prezentacijskog sloja je ostvarivanje interakcije s korisnikom i prikazivanje podataka aplikacije na ekranu (Google, 2023). Prezentacijski sloj treba sadržavati samo podatke koji su bitni za prikazivanje, stoga se podaci na putu iz nižih slojeva trebaju optimalno konvertirati. U pogledu Android komponenata, Chordiato se sastoji od jedne aktivnosti i više fragmenata. Pristup izradi aplikacije s jednom aktivnosti ima određene benefite, stoga se često koristi, pa čak i u izradi mnogo većih aplikacija. Može se reći da je fragment lakši komadić sučelja. Dok je aktivnost čvrsto povezana s ekranom, fragment ima manje opsežan životni ciklus, to jest kraći životni vijek i može zauzimati manji dio ekrana. Imajući jednu

[12] *ViewGroup* je specifičniji pogled (engl. *View*) koji može sadržavati druge poglede
[13] Digitalno potpisivanje certifikatom, omogućuje da se generirana APK datoteka aplikacije objavi na *PlayStoreu*
[14] APK je format datoteke koji Android koristi za instaliranje i distribuciju gotove aplikacije

aktivnost može se jasnije pratiti tijek aplikacije i njenih ekrana, pomoću navigacijske komponente Androida, koja u sebi sadrži stog odredišta, to jest fragmenata.



Sl. 5.6 Paketna struktura prezentacijskog sloja

Iz prikazane raspodjele paketa (Sl. 5.6) da se zaključiti da je prezentacijski sloj raspodijeljen po funkcionalnostima, odnosno pa ekranima. Svaki paket sadrži sve potrebne klase za prikaz jednog ekrana, a svima im je zajedničko da sadrže klasu fragmenta i klasu *viewmodela.* Korisničko sučelje (engl. *user interface,* skraćeno UI) je ono što korisnik vidi. U vidu Chordiato aplikacije, ono se sastoji od UI elemenata na ekranu s pripadajućim UI stanjem. UI je tako vizualna reprezentacija UI stanja, pa se tako svaka promjena u stanju reflektira na ekranu.

```
data class SheetState(
    val sheetUrl: String = "",
    val isSampling: Boolean = false,
    val isLoading: Boolean = false,
    val error: String = ""
)
```

Kod 5.1 Podatkovna klasa za čuvanje stanja korisničkog sučelja

Kod (Kod 5.1) prikazuje jednu od klasa stanja aplikacije, pomoću koje se na ispravan način upravlja korisničkim sučeljem, tako da se oslobodi od bilo kakvih drugih operacija i stavi fokus na jednu ulogu, a to je da pročita stanje i ispravno ažurira svoje elemente. Promjene stanja česte su, bilo da su nastale interakcijom korisnika ili promjenom podataka iz donjih slojeva. Pošto dinamična priroda aplikacija uvjetuje da će se stanje mijenjati, ističe se potreba za posrednikom, za komponentom u kojoj će se definirati logika za potrebnu reakciju na svaki događaj. Chordiato aplikacija stoga koristi *ViewModel* klase. Sadržavajući ih, nije ništa posebnija do gomile drugih aplikacija, zato što je pristup izradi s tom klasom iznimno popularan. UI klase su najčešće usko povezane s *viewmodelima* i sadrže mnogo međuovisnosti, pa tako Chordiato sadrži po jedan *viewmodel* za svaki fragment. *Viewmodel* klase Chordiata izrađene su i imaju funkcionalnosti koje preporučuju dobre prakse. One

sadrže i distribuiraju stanja koje koristi sučelje, to jest fragment. Primaju obavijesti o događajima izazvanim korisničkom interakcijom, te na njih određeno reagiraju. Sadrže okidače koje fragment može okinuti i time izazvati operaciju, koja se može, ali ne mora reflektirati na ekranu. Koristeći stanje i *viewmodele* znatno se oslobode aktivnosti i fragmenti od sadržavanja logike koju nije preporučljivo imati komponentama svojstvenima Android sustavu. Uz to, prati se dobar obrazac jednosmjernog toka podataka (engl. *unidirectional dana flow,* skraćeno UDF), gdje podaci teku samo iz nižih slojeva prema višim. *Viewmodeli* Chordiato aplikacije uglavnom u konstruktoru prime jednu ili više *UseCase* klasa iz sloja poslovne logike. One omogućuju jednostavno korištenje i ispravno izvršavanje potrebnih operacija iz podatkovnog sloja. Implementirane su da rade asinkronim tokovima podataka, koji se prenose iz podatkovnog sloja, te tako mogu kontinuirano izvještavati o stanju operacije i tako ispravno i ažurno odaslati stanje UI-a. Što se tiče protoka podataka iz *viewmodela* prema fragmentu, koristi uglavnom se koristi *stateflow*. *Stateflow* takoreći drži stanje i emitira novo kad se promjeni. To je ujedno primjer korištenja obrasca promatranja (engl. *observable pattern*) gdje objekt koji odašilje stanja, zvan subjekt, ima odgovornost izvještavati promatrače. Postoje mnoga rješenja koja mogu postići tu funkcionalnost, razvijena od strane *Googlea* i od *JetBrainsa.* Objekt stanja, `FavouritesState`, iz *viewmodela* objavljuje se tako što se izloži javni nepromjenjivi tok, a on povezan je s unutarnjim promjenjivim tokom kojem se mijenjaju stanja, to jest vrijednosti unutar *viewmodela* (Kod 5.2).

```
class FavouritesViewModel(
    private val getSongsUseCase: GetSongsUseCase
) : ViewModel() {

    private val _state =
MutableStateFlow<FavouritesState>(FavouritesState())
    val state = _state.asStateFlow()

    private var fetchJob: Job? = null

    fun getTracks() {
        fetchJob?.cancel()
        fetchJob = viewModelScope.launch {
            ...
```

Kod 5.2 Primjer *viewmodel* klase iz Chordiato aplikacije

U fragmentu se vrši promatranje promjena stanja tako što se pretplati na *sharedflow* i onda ubire njegovo zadnje stanje. Za razliku od *LiveDatae*[15] nije svjesna životnog ciklusa komponente u kojoj se nalazi, stoga se mora oprezno i ispravno koristiti tako što se pozove unutar djelokruga određenog životnog ciklusa, u ovom slučaju `lifecycleScope` objekta (Kod 5.3).

```
lifecycleScope.launchWhenStarted {
    viewModel.state.collectLatest { state ->
        ...
```

Kod 5.3 Prikaz promatranja od strane fragmenta

Viewmodel tako odašilje podatke fragmentu, to jest fragment prima stanja kada se ona promijene, te onda samo mora ažurirati ekran ovisno o potrebi. Ažuriranje ekrana Chordiato aplikacije, poduprto je *view bindingom*. To je moćan alat *Android Jetpack* kolekcije biblioteka koji uvelike olakšava interakciju s pogledima. Koristi se na način da se pogled stvori (engl. *inflate*) pomoću *binding* klase, te se njegov korijen (engl. *root*) vrati kao parametar `onCreateView()`[16] metode (Kod 5.4).

Kod 5.4 Korištenje *view bindinga* u fragmentu

```
class HistoryFragment : Fragment() {
    private lateinit var binding: FragmentHistoryBinding
    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?,
    ): View? {
        binding = FragmentHistoryBinding.inflate(
            inflater, container, false)
            ...
        return binding.root
    }
            ...
}
```

[15] Klasa slična *Stateflowu* koja služi za čuvanje stanja i obavještavanje promatrača, razvijena od strane *Googlea* i dio je *Jetpack* kolekcije biblioteka

[16] Ugrađena metoda svojstvena vidljivim Android komponentama, to jest aktivnosti i fragmentu, koja se pri implementaciji istih mora *overrideati*

Tako se dobije direktan pristup pogledima iz korijenskog (engl. *root*) pogleda. Za prikaz lista koristi se *RecyclerView,* nezaobilazni alat za prikazivanje seta podataka pri razvoju sučelja s pogledima. Pružanjem podataka i definiranjem izgleda pogleda elemenata liste, on dinamično kreira elemente kada su oni potrebni. Ime dobiva iz svoje prirode funkcioniranja. On dakle, reciklira individualne elemente tako što, kad pogled elementa prestane biti vidljiv na ekranu, ne uništava ga, već ga iskoristi za prikaz novih elemenata. To smanjuje potrošnju energije i pospješuje bolju responzivnost i performanse. U pozadini to znači da ima u *cacheu* par elemenata prije i poslije vidljivog dijela sučelja u kojem se nalazi. Nekoliko klasa rade zajedno kako bi se stvorila dinamična, vizualno složena lista. *RecyclerView* je *ViewGroup* koja sadrži poglede potrebne za prikaz željenih podataka. Svaki element iz liste sparen je s odgovarajućim pogledom koji ga prikazuje u *ViewHolder* klasi. *RecyclerView* klasa zatražuje poglede i sparuje ih s podacima pozivajući metode iz prilagođene *RecyclerView.Adapter* klase. Prilagođeni adapter Chordiata, čija klasa u potpunosti omogućava rad dinamične liste, nasljeđuje *RecyclerView.Adapter* koji je tipiziran na prilagođeni *ViewHolder*. Osim što u konstruktoru prima klasu, prima i funkciju, koja će dobiti odabrani objekt i onda izvršiti operaciju definiranu u fragmentu (**Pogreška! Izvor reference nije pronađen.**).

```
class FavouritesRecyclerViewAdapter(
    private var tracks: List<Track>,
    private val clickListener: (Track) -> Unit,
) :
RecyclerView.Adapter<FavouritesRecyclerViewAdapter.ViewHolder
>() {
```

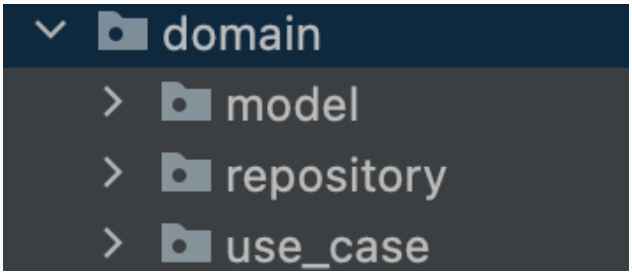
Kod 5.5 Konstruktor parametri i nasljeđivanje kod rada s *RecyclerViewom*

`ViewHolder` klasa definirana je odmah unutar `FavouritesRecyclerViewAdapter` klase, tako da sve funkcionalnosti potrebne za rad budu centralizirane na jednom mjestu.

5.3.2. Sloj poslovne logike

Sloj poslovne logike ili sloj domene, nalazi se između prezentacijskog i podatkovnog sloja. On sadrži kompleksnu poslovnu logiku ili jednostavnu logiku koja se koristi na više mjesta od strane prezentacijskog sloja (Google, 2023). On je opcionalni sloj, ali je nužan za stvaranje, održivog, skalabilnog rješenja. Chordiato nije velika aplikacija,

nema puno ekrana ni funkcionalnosti. Sloj domene na toj skali može se smatrati pretjerivanje u pogledu nepotrebne modularizacije projekta (engl. *overengineering*). Usprkos navedenom, veću težinu nosi argument da programeri moraju težiti savršenstvu. Mora se iskoristiti svaka prilika za pisanje lijepog, strukturiranog, čistog koda, makar pod cijenu kojeg paketa sa samo jednom klasom. Općenito, sloj domene olakšava testiranje aplikacije. Sprječava ponavljanje istog koda i postojanje velikih klasa, te omogućuje bolju raspodjelu odgovornosti. Čitljivost i razumljivost koda znatno se poveća postojanjem sloja poslovne logike.



Sl. 5.7 Paketna struktura sloja poslovne logike

Po dobrim praksama organizacije, u *model* paketu paketa *domain* (Sl. 5.7) nalaze se podatkovne klase izgrađene samo od svojstava koje trebaju prezentacijskom sloju. Njima je mjesto upravo u sloju domene. Podatkovne klase mapiraju se u nju, i iz nje u podatkovnom sloju. Sučelja repozitorija također se trebaju nalaziti u sloju domene. Implementacija repozitorija sa svom logikom su u podatkovnom sloju, ali dodavanjem njihovih sučelja u sloj domene, dobije se vrlo skladna arhitektura koja ne ovisi i nije čvrsto spojena s ostalim modulima aplikacije. Sloj domene znatno olakšava pisanje koda na prezentacijskom sloju i čini kod puno intuitivnijim za shvatiti. Sloj domene Chordiato aplikacije koristi klase koje u imenu imaju sufiks *usecase*. Sami sufiks daje naznaku da je riječ o nekoj funkcionalnosti, o nekom slučaju upotrebe. Te klase implementiraju određenu funkcionalnost, to jest odrađuju jednu operaciju i pri tom se nazivaju imenom koje eksplicitno i sažeto opisuje šta rade. To je upravo primjer takozvane vrišteće arhitekture (engl. *screaming architecture*). Takva arhitektura podrazumijeva da se komponente visoko u hijerarhiji nazivaju upravo po onome šta rade (Martin, 2017). Takve klase pogodno je nazivati tako da prva riječ bude glagol, druga imenica i sufiks *UseCase* (Kod 5.6). Takva arhitektura u kombinaciji s odličnim funkcionalnostima Kotlin programskog jezika omogućuje pisanje izuzetno čitkog, razumljivog i čistog koda.

```
class RecognizeSongUseCase(
    private val repository: TrackRemoteRepository
```

```
) {
    operator fun invoke(requestBody: RequestBody):
                Flow<Resource<Track>> = flow {
                ...
```

Kod 5.6 Jedna od *usecase* klasa Chordiato aplikacije

U kodu (Kod 5.6) je prikazana klasa koja prepoznaje pjesmu, tako što prosljeđuje enkodirani zvučni uzorak namijenjenoj metodi iz repozitorija. U Kotlinu, ključna riječ `operator` omogućuje prilagođavanje implementacije ugrađenim operatorima. Tako se zapravo omogući da se pruži implementacija `invoke()` metode što zapravo znači da se klasu može pozivati slično kao funkciju, što čini kod na prezentacijskom sloju čišćim. U navedenom slučaju povratni tip podatka je `Flow<Resource<Track>>`. *Flow* je tip podatka koji može emitirati više vrijednosti kroz vrijeme, to jest on je asinkroni protok (engl *stream*) podataka.

```
sealed class Resource<T>(val data: T? = null, val message:
String? = null) {
    class Loading<T>(data: T? = null): Resource<T>(data)
    class Success<T>(data: T?): Resource<T>(data)
    class Error<T>(message: String, data: T? = null):
Resource<T>(data, message)
}
```

Kod 5.7 Resource klasa

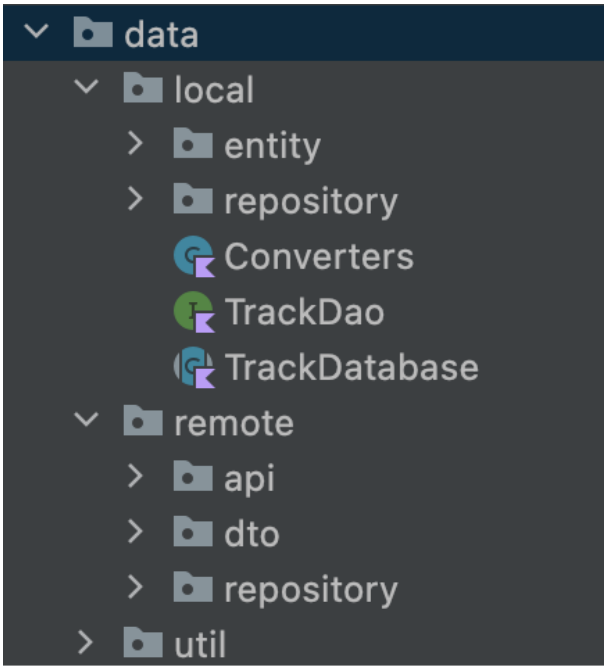
Prikazana `Resource` klasa je *sealed* klasa, koja ima strogo definirane podklase koje mogu postojati i znaju se u vremenu kompajliranja, u ovom primjeru `Loading, Success i Error` (Kod 5.7). Generička je i može imati raznovrsnu primjenu pa se nalazi u *core* paketu aplikacije. Ona je omotač (engl. *wrapper*) oko objekta koji se treba slati, stoga se ona šalje u metodi `emit()` Kotlin *flowa*. Tako se prezentacijski sloj izvještava o trenutnom stanju dohvaćanja podataka.

5.3.3. Podatkovni sloj

Podatkovni sloj upravlja protokom podataka tako što postavlja pravila koja određuju kako aplikacija kreira, sprema i mijenja podatke. Podatkovni sloj odgovoran je da omogući pristup podacima iz aplikacije. U njemu su centralizirane sve promjene koje nad podacima. On sadrži logiku vezanu za podatke pa tako treba rješavati konflikte između više izvora podataka

(Google, 2023). Drugi slojevi arhitekture ne bi nikad smjeli imati pristup izvoru podataka direktno, već to mora biti omogućeno pomoću repozitorijskih klasa. Generalno, podatkovni sloj treba izložiti mogućnosti okidanja jednokratnih *CRUD*[17] operacija, te obavještavati kad se podaci mijenjaju. Svi podaci koji podatkovni sloj pruža trebali bi biti neizmjenjivi (engl. *immutable*). Potrebno je voditi se *single source of truth* principom pri kreiranju podatkovnog sloja, tako što odredimo jedan izvor podatka koji će biti konzistentan, točan i uvijek ažuriran. Također, potrebno je da rad s izvorima podataka i repozitorijima bude *main-safe,* to jest siguran za pozivanje iz glavne dretve, a to znači da se izvršavanje operacija odvija u adekvatnim dretvama. Biblioteke korištene u podatkovnom sloju Chordiato aplikacije (*Room, Retrofit*) same po sebi pružaju *main-safe* operacije koje se mogu pravilno iskoristiti.

Sl. 5.8 Paketna struktura podatkovnog sloja



Chordiato ima tri izvora podataka, od kojih su dva mrežna i jedan lokalni, stoga je glavna podjela podatkovnog sloja na lokalne i udaljene (Sl. 5.8). Objekti podatkovnog sloja pretvaraju se u objekte sloja poslovne domene pomoću ekstenzijskih funkcija koje se nalaze u datoteci `TrackMapper` u *util* paketu. Ekstenzijske funkcije iznimno korisna su

[17] Akronim za četiri nužne funkcije potrebne za implementiranje spremišta podataka, a to su stvaranje (engl. *create*), ažuriranje (engl. *update*), umetanje (engl. *insert*), brisanje (engl. *delete*)

funkcionalnost Kotlina, jer omogućuju da se klasi iz bilo koje datoteke, vrlo jednostavno nadoda funkcija, bez potreba za nasljeđivanjem i to bez usporavanja performansi (Kod 5.8).

```
fun TrackEntity.toTrack() : Track {
    return Track(
        id = id,
        name = name,
        artist = artist,
        favourite = favourite,
        link = link,
        timestamps = openedTimestamps
    )
}
```

Kod 5.8 Ekstenzijska funkcija kod pretvorbe entiteta podatkovne klase u domensku

Što se tiče udaljenih izvora podataka, Chordiato aplikacija sadrži svega dva upita na dva različita servisa. Jedan upit je *post* metoda na *ShazamAPI,* koja kao parametar šalje zvučni uzorak enkodiran u *string*, a kao uspješan odgovor dobiva podatke o prepoznatoj pjesmi. Drugi upit je *get* metoda na *SerpApi* u kojoj šalje parametre koji se onda pretražuju na *Googleu*, a povratno se dobije složena JSON datoteka koja predstavlja prvu stranicu rezultata *Google* pretrage. Jednostavno dohvaćanje podataka s mreže omogućeno jer *Retrofitom,* to jest najpoznatijom Android bibliotekom za komunikaciju s REST servisima. *Retrofit* je iznimno lagan i intuitivan za korištenje, a uz to sve operacije izvršava asinkrono. Http metode se uz pomoć anotacija na jednostavan način raspišu u sučelje(engl. *interface*) (Kod 5.9). To sučelje se ne implementira, već se *Retrofit* klasi *builder patternom* uz ostale parametre proslijedi *Class* instanca sučelja (Kod 5.10).

```
interface ShazamApi {
    @Headers(
        "X-RapidAPI-Key:
      b111c9527bmsh481bdae5f21f558p175feejsna6bbda25faae",
        "X-RapidAPI-Host: shazam.p.rapidapi.com"
    )
    @POST("songs/detect")
    suspend fun detect(
      @Body encodedRawAudio: RequestBody):ShazamResponseDto
}
```

Kod 5.9 *Retrofit* sučelje s jednom http post metodom

```
val retrofit = Retrofit.Builder()
          .baseUrl(SHAZAM_API_URL)
          .addConverterFactory(GsonConverterFactory.create())
          .client(okhttpClient())
          .build()
shazamApi = retrofit.create(ShazamApi::class.java)
```

Kod 5.10 Generiranje *Retrofit* klijenta (*shazamApi*)

Retrofit zadovoljava sve potrebe pri upućivanju http zahtijeva, na primjer u kodu (Kod 5.10) vidi se da je uz osnovni URL udaljenog API-a, dodana klasa za serijalizaciju i deserijalizaciju objekata (Gson instanca koju pruža `GsonConverterFactory.create()`). Dodan je i prilagođeni klijent, a on služi da pruži presretača (engl. *interceptor*), to jest doda `ShazamInterceptor` klasu, implementaciju sučelja Interceptor, koji doda određene atribute pri svakom API pozivu. Repozitorij mrežnog dijela podatkovnog sloja implementacija je sučelja koji je definiran u sloju domene, to jest sloju poslovne logike. On ovisi o oba *retrofit* sučelja pa ih zato prima u konstruktoru. Sve klase koje se serijaliziraju i deserijaliziraju i prenose mrežom nalaze se u *dto* paketu. Generirane su pomoću dodatka (engl. *plugin*) *JsonToKotlinClass*, s kojim se vrlo jednostavno iz JSON sheme dobiju ispravno anotirane klase za Gson serijalziaciju. *SQLite* je nativna C/C++ biblioteka s implementiranom lakom, brzom, stabilnom bazom podataka optimiziranom za jednog korisnika. *Room* je biblioteka izrađena od strane *Googlea,* koja je nadogradnja nad ugrađenu *SQLite* bazu. *Room* zadržava sve benefite *SQLitea,* ali uz jednostavno korištenje s manje koda. Znatno olakšava izradu i pristup bazi, te rad s njom, tako što između ostalog, omogućava da se struktura baze i upita definiraju korištenjem anotiranih Kotlin klasa. *Room* biblioteka sastoji se od aplikacijskog programskog sučelja, anotacija i kompajlera. Aplikacijsko programsko sučelje sadrži klase koje se nasljeđuju kako bi se definirala i kreirala instanca baze podataka. Anotacijama se naznačuju razne stavke poput koje klase predstavljaju bazu podatka, funkcije za upite, entiteti tablice i dr. Kompajler procesuira anotirane stavke i generira implementaciju baze (Sills, et al., 2022). *Room* biblioteka omogućila je izradu i implementaciju jednostavne baze podataka Chordiato aplikacije u svega par klasa i to s vrlo malo koda. Baza podataka sadrži samo jednu tablicu koja predstavlja pjesmu.

```
@Entity
data class TrackEntity(
    @PrimaryKey val id: Int,
```

```
    val name: String,
    val artist: String,
    val favourite: Boolean? = false,
    val link: String? = null,
    val lastTimestamp: LocalDate? = LocalDate.now()
)
```

Kod 5.11 Deklariranje entiteta, koji je tablica u bazi podataka

Tablice se generiraju tako što se Kotlin podatkovna klasa anotira s oznakom `@Entity`, te joj se odredi primarni ključ anotacijom `@Primary key` (Kod 5.11). Sami upiti, definirani pripadajućim anotacijama, nalaze su u sučelju (engl. *interface*) koje se anotira sa `@Dao`(Kod 5.12). Upiti za dohvaćanje podataka anotirani `@Query` anotacijom sadrže tekstualni SQL iskaz (engl. *statement)*, a *Room* čak omogućava *type-safety* u *stringu*.

```
@Dao
interface TrackDao {
    @Insert(onConflict = OnConflictStrategy.IGNORE)
    suspend fun insertTrack(trackEntity: TrackEntity)

    @Update
    suspend fun updateTrack(trackEntity: TrackEntity)

    @Query("SELECT * FROM trackentity")
    suspend fun getAllTracks() : List<TrackEntity>

    @Query("SELECT * FROM trackentity WHERE favourite = 1")
    suspend fun getFavouriteTracks() : List<TrackEntity>
}
```

Kod 5.12 Sučelje koje definira upravljanje bazom podataka

Nakraju, konkretno kreiranje instance *Dao* sučelja izvrši se u klasi koja naslijedi klasu `RoomDatabase` i anotirana je anotacijom `@Database`. Toj se klasi anotacijama definiraju potrebni podaci, poput entiteta, definiranih pretvarača (engl. *converters*), verzije, a u tijelu klase, definira se apstraktna vrijednost ili funkcija koja je tipa *Dao*, konkretno u primjeru `TrackDao`(Kod 5.13).

Kod 5.13 Klasa baze podataka Chordiato aplikacije

```
@Database(
    entities = [TrackEntity::class],
```

```
    version = 1
)
@TypeConverters(Converters::class)
abstract class TrackDatabase : RoomDatabase() {
    abstract val dao: TrackDao
}
```

`TrackDatabase` je *single source of truth* Chordiato aplikacije. Zbog navedenih konkretnih primjera koda iz Chordiato aplikacije, može se izvesti zaključak da je *Room* biblioteka odličan izbor pri izradi lokalne baze podataka. Puno implementacijskog i konfiguracijskog posla odradi sama umjesto programera i to bez eksplicitnog prikazivanja u kodu. Ima napredne funkcionalnosti koje smanjuju *boilerplate* kod i čine ga vrlo jasnim i razumljivim. Podatkovni sloj sadrži repozitorije, to jest implementacije sučelja repozitorija koji se nalaze u sloju domene. U njima se implementira sva potrebna logika vezana za upravljanje lokalnim i udaljenim izvorima podataka. Zbog naprednosti korištenih biblioteka kod u repozitorijima je izuzetno kratak.

Zaključak

Koristeći postojeće alate na inovativan način, može se napraviti zanimljivo i korisno rješenje, makar samo za jednu malu nišu korisnika. Izrada i implementacija aplikacije predstavlja izazov, kojem se može pametno i strukturirano pristupiti. Korištene alate treba dobro proučiti i pokušati shvatiti kako rade kako bi se došlo do spoznaje njihove genijalnosti. Detaljan opis algoritma za pretraživanje pjesama, kreće od samog čovjeka i njegova osjeta zvuka, prelazi preko osnova fizike zvuka, da bi nakraju stigao u opsežan svijet digitalnog signala, a sve to kako bi se algoritam bolje predočio i stavio u adekvatan kontekst. Nadalje, opisane su funkcionalnosti aplikacije, koje upotpunjene slikama, dočaravaju rad Chordiato aplikacije. Jedinstvenost i posebnost Chordiata ističu se kao glavni atributi aplikacije u usporedbi sa sličnim postojećim rješenjima. Nakon toga, predstavlja se okruženje u kojem je Chordiato izrađen, jezik kojim je izrađen i platforma za koju je izrađen. Analiziraju se alati koji su pomogli pri izradi vizualnog identiteta, nakon čega se daje detaljan pregled i presjek arhitekture. Za svaki od sloja predstavljene su funkcije, najbitnije korištene komponente, te struktura. Opisuje se proces izrade, predočava se arhitektura aplikacije, iznose se spoznaje i zaključci uglavnom vezani za važnost praćenja utemeljenih, poznatih i dobrih praksi razvoja, a temeljeni su na stvarnom iskustvu stečenom pri izrađivanju Chordiata.

Popis kratica

API	*Aplication Programing Interface*	aplikacijsko programsko sučelje
IDE	*Integrated Development Enviorment*	integrirano razvojno okruženje
FFT	*Fast Fourier Tranform*	brza Fourierova transformacija
SDK	*Software Development Kit*	jedinstveni resursni lokator
UI	*User Interface*	korisničko sučelje
XML	*Extensible Markup Language*	jezik za označavanje podataka
URL	*Uniformed Resource Locator*	jedinstveni resursni lokator
UX	*User Experience*	korisnički doživljaj
CRUD	*Create Read Update Delete*	stvaranje, čitanje, ažuriranje, brisanje
APK	*Android Package Kit*	Android alatni paket

Popis slika

Popis tablica

Popis kôdova

Literatura

[1] Wang, A. L.-C., 2003. *An Industrial-Strength Audio Search Algorithm.* [Mrežno] Dostupno na: https://www.ee.columbia.edu/~dpwe/papers/Wang03-shazam.pdf, 12.01.2023.

[2] Paar, V., 2009. *Fizika 3.* Zagreb: an.

[3] Jovanović, J., 2015. *Music Recognition, Algorithms, FIngerprinting, and Processing.* [Mrežno] Dostupno na: https://www.toptal.com/algorithms/shazam-it-music-processing-fingerprinting-and-recognition, 16.01.2023.

[4] Batchelor, M., 2014. *Top 20 Most Recognizable Songs Of All Time.* [Mrežno] Dostupno na: https://ceoworld.biz/2014/11/04/top-20-recognizable-songs-time/ [Pokušaj pristupa 16 1 2023], 18.01.2023.

[5] Martin, R. C., 2017. *Clean Architecture: A Craftsman's Guide to Software Structure and Design.* s.l.:Prentice Hall.

[6] Griffiths, D. & Griffiths, D., 2021. *Head First Android Development.* Sebastopol: O'Reilly Media.

[7] Jemerov, D. & Isakova, S., 2017. *Kotlin in action.* Shelter Island: Manning.

[8] Sills, B., Gardner, B., Marsicano, K. & Stewart, C., 2022. *Android Programming: The Big Nerd Ranch Guide.* Atlanta: Addison-Wesley Professional.

[9] Collins, A., 2014.. [Mrežno] Dostupno na: https://ed.ted.com/lessons/how-playing-an-instrument-benefits-your-brain-anita-collins, 09.02.2023

[10] Witte, J., 2021. *AI-technology begind the chords of Chordify.* [Mrežno] Dostupno na: https://chordify.net/pages/technology-algorithm-explained/, 16.02.2023

[11] GlobalStats, 2023.. *statcounter GlobalStats.* [Mrežno] Dostupno na: https://gs.statcounter.com/os-market-share/mobile/worldwide, 29.01.2023

[12] Lunden, I., 2018. *TechCrunch.* [Mrežno] Dostupno na: https://techcrunch.com/2018/09/24/apple-closes-its-shazam-acquisition-and-says-the-music-recognition-app-will-soon-become-ad-free/, 19.01.2023

[13] Google, 2023. *PlayStore.* [Mrežno] Dostupno na: https://play.google.com/store/apps/details?id=com.shazam.android, 16.02.2023

[14] Google, 2023. *PlayStore.* [Mrežno] Dostupno na: https://play.google.com/store/apps/details?id=net.chordify.chordify, 16.02.2023

[15] Google, 2023. *PlayStore.* [Mrežno] Available at: https://play.google.com/store/apps/details?id=com.songsterr, 16.02.2023

[16] Apple, 2023. *App Store.* [Mrežno]
Dostupno na: https://apps.apple.com/us/app/shazam-music-discovery/id284993459, 16.02.2023

[17] UXtools, 2022. *UXtools.* [Mrežno]
Dostupno na: https://uxtools.co/survey/2022/ui-design/#:~:text=Most%20Popular%20UI%20Design%20Tools&text=Counting%20Number%20of%20responses.,9%2C%20Other%3A%2045%2C., 16.02.2023

[18] Google, 2023. *Android for developers.* [Mrežno]
Dostupno na: https://developer.android.com/topic/architecture, 13.02.2023