Android aplikacija za ugađanje gitare

Paradžik, Antonio

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Sveučilišni studij

ANDROID APLIKACIJA ZA UGAĐANJE GITARE

Završni rad

Antonio Paradžik

Osijek, 2023.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

Obrazac Z1P - Obrazac za ocjenu završnog rada na preddiplomskom sveučilišnom studiju

Osijek, 15.09.2023.

Odboru za završne i diplomske ispite

Prijedlog ocjene završnog rada na preddiplomskom sveučilišnom studiju

Ime i prezime Pristupnika:	Antonio Paradžik
Studij, smjer:	Programsko inženjerstvo
Mat. br. Pristupnika, godina upisa:	R4546, 27.07.2020.
OIB Pristupnika:	41556551124
Mentor:	doc. dr. sc. Hrvoje Leventić
Sumentor:	,
Sumentor iz tvrtke:	
Naslov završnog rada:	Android aplikacija za ugađanje gitare
Znanstvena grana rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak završnog rad:	Istražiti i opisati postojeće metode obrade zvuka radi ugađanja gitare. Opisati potrebne teorijske osnove iz obrade signala. Za praktični dio izraditi Android aplikaciju koja će omogućiti ugađanje gitare preko mikrofona telefona. Rezervirano za: Antonio Paradžik
Prijedlog ocjene završnog rada:	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 2 bod/boda Jasnoća pismenog izražavanja: 2 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene od strane mentora:	15.09.2023.
Datum potvrde ocjene od strane Odbora:	24.09.2023.

Potvrda mentora o predaji konačne verzije rada:	*Mentor elektronički potpisao predaju konačne verzije.*
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA
I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTI RADA

Osijek, 25.09.2023.

Ime i prezime studenta:	Antonio Paradžik
Studij:	Programsko inženjerstvo
Mat. br. studenta, godina upisa:	R4546, 27.07.2020.
Turnitin podudaranje [%]:	7

Ovom izjavom izjavljujem da je rad pod nazivom**: Android aplikacija za ugađanje gitare**

izrađen pod vodstvom mentora doc. dr. sc. Hrvoje Leventić

i sumentora ,

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SADRŽAJ

1. UVOD

Gitara drži mjesto jednog od najpopularnijih instrumenata današnjice. Kvalitetno ugađanje gitare ključno je kako bi se uživalo u optimalnom zvuku i izvedbi na gitari. Stoga se mnogi profesionalni i amaterski glazbenici suočavaju s izazovom pravilnog ugađanja gitare.

Napretkom tehnologije i sve većim korištenjem pametnih telefona se, uz ostale tehnike ugađanja, razvijaju mnoge aplikacije koje glazbenicima pružaju pomoć pri ugađanju tona gitare. Izrada ovog završnog rada fokusirana je na razvoj i implementaciju Android aplikacije za ugađanje gitare putem ugrađenog mikrofona u pametnom telefonu.

Zadatak ovog rada je pružiti uvid u ovu tehnologiju, od teorijskog dijela sve do implementacije aplikacije. U narednom poglavlju obrađuju se pojmovi zvuka, tona, gitare i procesa ugađanja koji su potrebni kako bi se razumjela svrha i cilj ove aplikacije. Nadalje, u trećem poglavlju objašnjeno je korištenje par postojećih rješenja u obliku aplikacija za ugađanje gitare te Flight fizičkog kromatskog štimera za gitaru. U četvrtom poglavlju opisane su korištene tehnologije, alati te biblioteka Tarsos DSP koja je korištena pri procesiranju snimljenog zvuka. Zatim, u petom poglavlju prikazana je implementacija aplikacije u Android Studio integriranom razvojnom okruženju. Pojašnjeni su tehnički aspekti, izazovi te implementirana rješenja prilikom izrade aplikacije. Također, u istom poglavlju provedeno je testiranje i evaluacija rezultata u svrhu procjene performanse aplikacije. Konačno, posljednje poglavlje sadrži zaključak cjelokupnog rada.

1.1. Zadatak završnog rada

Zadatak završnog rada je napraviti Android aplikaciju za ugađanje tona gitare u Android Studio razvojnom okruženju. Potrebno je dati teorijsku pozadinu zadatka i navesti tehnologije koje se koriste. Nakon toga treba prikazati i objasniti korake razvoja aplikacije te provesti testiranje i evaluaciju rezultata.

2. ZVUK, TON I GITARA

Ovo poglavlje pojašnjava ključne pojmove u glazbi poput zvuka, tona i frekvencije. Osim toga, navode se osnove gitare i način ugađanja. Razumijevanje ovih pojmova i njihovog međusobnog odnosa važno je za proces ugađanja gitare i shvaćanje svrhe ove aplikacije.

2.1. Osnovno o zvuku i tonu

Prema [1], zvuk je senzacija koja se osjeti putem organa sluha kada vibracije dosegnu uho. Vibracije stvaraju oscilacije u zraku koje se šire u obliku longitudinalnih valova. Raspon zvučnih valova jest od 16 Hz do 20 000 Hz što je raspon u kojem ga čuje ljudsko uho. Zvuk niži od 16 Hz naziva se infrazvuk, a zvuk viši od 20 000 Hz ultrazvuk [2].

Prema [3], ton je složeni zvuk koji nastaje pravilnim i periodičnim titranjem zraka. Parcijalni tonovi su mu u maksimalno harmoničnom odnosu, stoga mu se može odrediti visina. Ton se smatra osnovnim elementom glazbe i ima četiri ključna obilježja: visinu, intenzitet, trajanje i boju. Visina se određuje frekvencijom titranja izvora tona. Intenzitet određuje amplituda titranja izvora. Trajanje ovisi o vremenu titranja. Boja tona ovisi o međudjelovanju različitih elemenata poput materijala, veličine, građe i oblika glazbala.

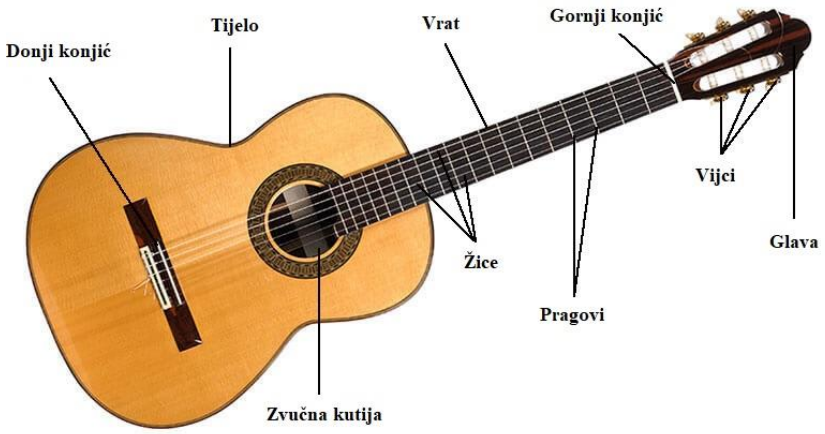
Kako je ranije navedeno, zvuk se određuje frekvencijom vibracije. Frekvencija se definira kao broj ciklusa kompresije i rarifikacije koji se događaju u jedinici vremena, odnosno broj ponavljanja neke periodične pojave u jedinici vremena. Mjerna jedinica je herc [Hz] [1].

2.2. Gitara i proces ugađanja

Gitara je žičani instrument korišten u širokom rasponu glazbenih žanrova. Obično su građene od različitih vrsta drveta koje sa sobom nose specifična svojstva proizvedenog tona te mogu biti različitih oblika. Gitara se sastoji od tri glavna dijela: glava, vrat i tijelo. Dijelovi gitare prikazani su na slici 2.1. Na glavi gitare nalaze se vijci kojima se žice podešavaju. Vrat gitare sadrži pragove kojima se može manipulirati duljinom titranja žice, a time i tonom. Tijelo gitare služi za pojačavanje zvuka vibrirajućih žica [4, str.34].

Gitara najčešće ima šest žica koje mogu biti metalne ili najlonske. Svaka žica je određene debljine i napetosti koji utječu na njenu frekvenciju. Pričvršćene su na jednom kraju na mašinici s vijcima te se pružaju cijelim vratom gdje su na drugom kraju pričvršćene za tijelo gitare.

Okidanje žice će uzrokovati periodičko titranje koje će prenijeti vibracije sve do tijela gitare te će se proizvesti ton.



Slika 2.1. Građa gitare

Proces ugađanja postupak je *naštimavanja* žice na specifičnu frekvenciju kako bi se postigao određeni tonalitet. Ton žice ovisi o tri stavke. To su duljina, debljina i napetost žice. U slučaju ugađanja otvorenih žica bez pritiska na tijelo gitare, stavka duljine je nepromjenjiva. Debljina je također nepromjenjiva te je određena za svaku od šest žica. Žice su raspoređene od najdeblje prema najtanjoj gdje najdeblja žica proizvodi najdublji ton, a najtanja proizvodi najviši ton. Tako da je u procesu ugađanja bit namještati napetost žice vijcima na glavi gitare. Zatezanjem žice visina tona se povisuje, a otpuštanjem snižava [4, str.68].

Frekvencije žica standardnog uštimavanja u zapadnoj glazbi navedene su u tablici 2.1.

Tablica 2.1. Frekvencije žica standardnog uštimavanja [5]

Žica (Ton)	Frekvencija	Znanstvena notacija tona
1 (e)	329.63 Hz	E4
2 (B)	246.94 Hz	B3
3 (G)	196.00 Hz	G3
4 (D)	146.83 Hz	D3
5 (A)	110.00 Hz	A2
6 (E)	82.41 Hz	E2

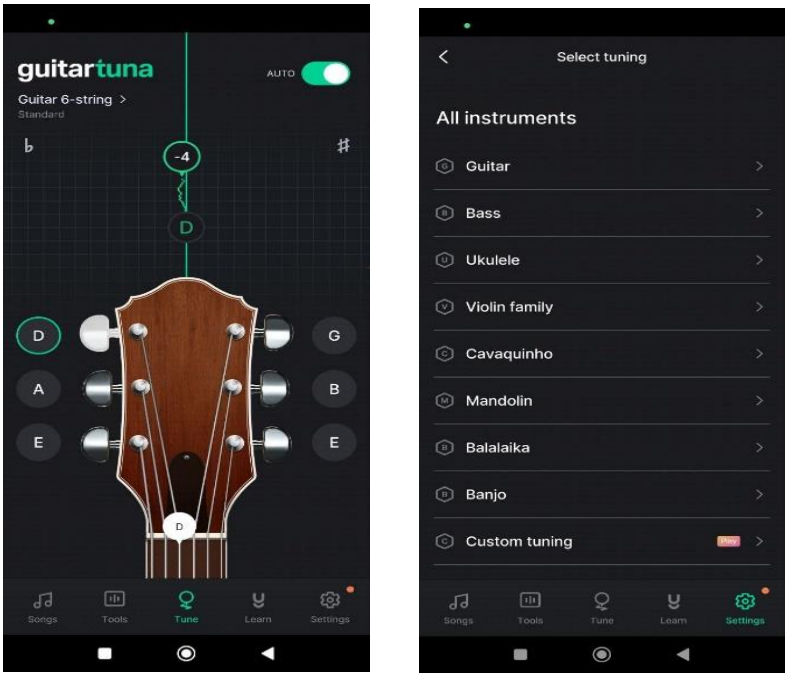
3. POSTOJEĆA RJEŠENJA

Postoji mnogo Android aplikacija za ugađanje gitara koje su dostupne u Google Play trgovini. U ovoj sekciji bit će predstavljeno nekoliko rješenja koja su najpopularnija i koja su korisnici najbolje ocijenili. Ove aplikacije nude vrlo praktično i precizno ugađanje gitare putem mikrofona pametnog telefona. Također, navest će se i primjer fizičkog štimera Flight FTC-77.

3.1. GuitarTuna : Chords, Tuner, Songs

Prema [6], GuitarTuna je u ovom trenutku najpopularnija i najviše ocijenjena na trgovini Google Play s ocjenom 4.7 od 5. Broji preko 100 milijuna korisnika koji hvale njenu preciznost i jednostavnost korištenja. Aplikacija nudi privlačan vizualni prikaz ugađanja tona žice. Korisnik ima mogućnost odabrati između 11 različitih instrumenata s dodatnim izborom različitog broja žica na određenim instrumentima.

Vidljivo na slici 3.1., na početnom zaslonu prikazana je glava instrumenta s odabranim rasporedom tonova te sviranjem žice pokazivač pokazuje odstupa li ton i koliko je odstupanje od željenog tona. Uz standardni raspored tonova, također se nude prilagođeni rasporedi tonova.

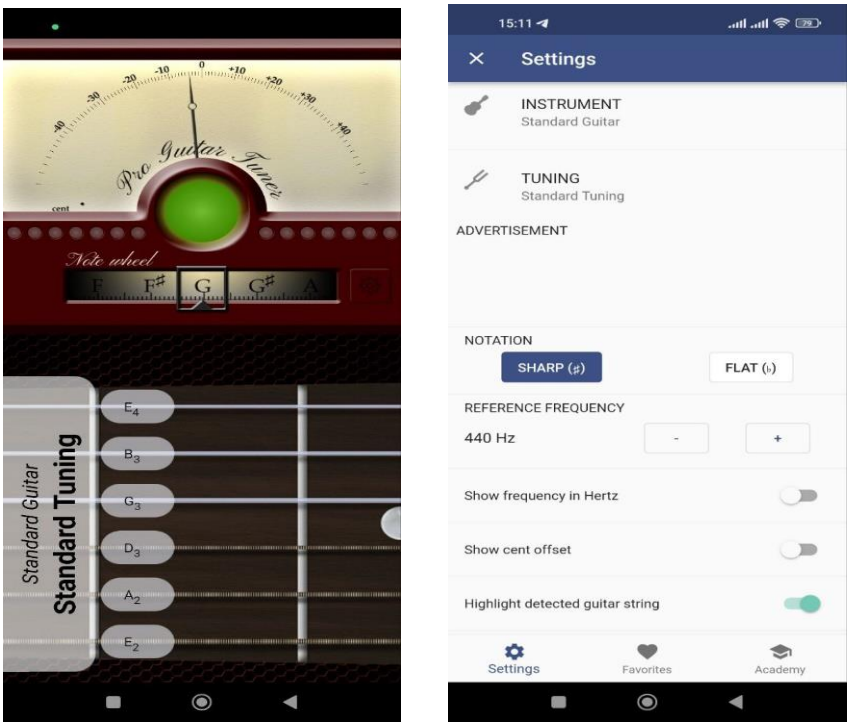


Slika 3.1. GuitarTuna – Izgled korisničkog sučelja

Moguće je izabrati automatski ili ručni odabir određenog tona. Odabirom opcije plaćanja napredne verzije otvara se mogućnost odabira novih instrumenata te se ukidaju reklame.

3.2. Pro Guitar Tuner

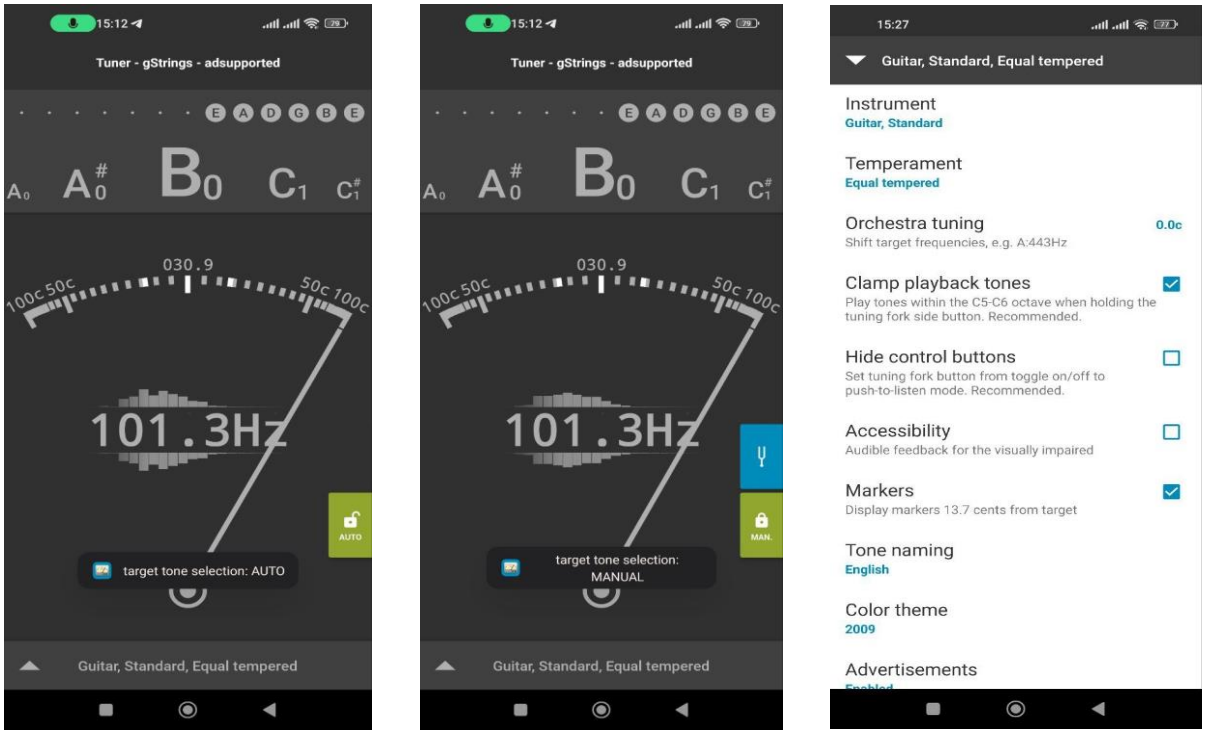
Prema [7], ova aplikacija također je vrlo cijenjena među korisnicima s ocjenom 4.5 od 5 te s preko 10 milijuna preuzimanja. Korisnici hvale intuitivno sučelje aplikacije. Aplikacija nudi kromatski prikaz tonova koji korisniku daje slobodu pri uštimavanju žice u svih 12 tonova ljestvice kao što se vidi na slici 3.2. Također sadrži mnogo instrumenata i razne rasporede tonova od standardnih do alternativnih. Kao i kod GuitarTuna plaćanjem napredne verzije dobivaju se druge pogodnosti aplikacije.



Slika 3.2. Pro Guitar Tuner – Izgled korisničkog sučelja

3.3. Tuner – gStrings

Prema [8], Tuner – gStrings aplikacija za ugađanje gitare i drugih instrumenata trenutno ima ocjenu 4.6 od 5, a koristi ju preko 10 milijuna korisnika. Poput Pro Guitar Tuner-a raspored tonova je kromatski što znači da nudi ugađanje žice u bilo koji željeni ton. Na slici 3.3. vidljivo je kako pritiskom na lokot postoji mogućnost izmjenjivanja automatskog i ručnog odabiranja određenog tona. Uz to nudi nekoliko korisnih opcija za različite potrebe korisnika.



Slika 3.3. gStrings – Izgled korišničkog sučelja

3.4. Flight FTC-77

Flight FTC-77 jedan je od pristupačnijih fizičkih štimera na tržištu. Ovaj štimer idealan je za gitare, basove, ukulele i slično. Pouzdan je i jednostavan za upotrebu. Važna stavka je da radi na principu vibracije što znači da ga se može koristiti za precizno ugađanje i u bučnim okruženjima.

Svijetao i jasni zaslon prikazuje digitalni mjerač koji pozeleni kada je odabrana žica savršeno u tonalitetu. Dodatno, frekvencija se može podesiti između 430 i 450 Hz (standardno je 440 Hz). Na slici 3.4. vidi se izgled sučelja kada je, u ovom slučaju, E ton šeste žice ugođen [9].

Slika 3.4. Flight FTC – 77 – Izgled štimera

4. PRIMIJENJENE TEHNOLOGIJE I ALATI

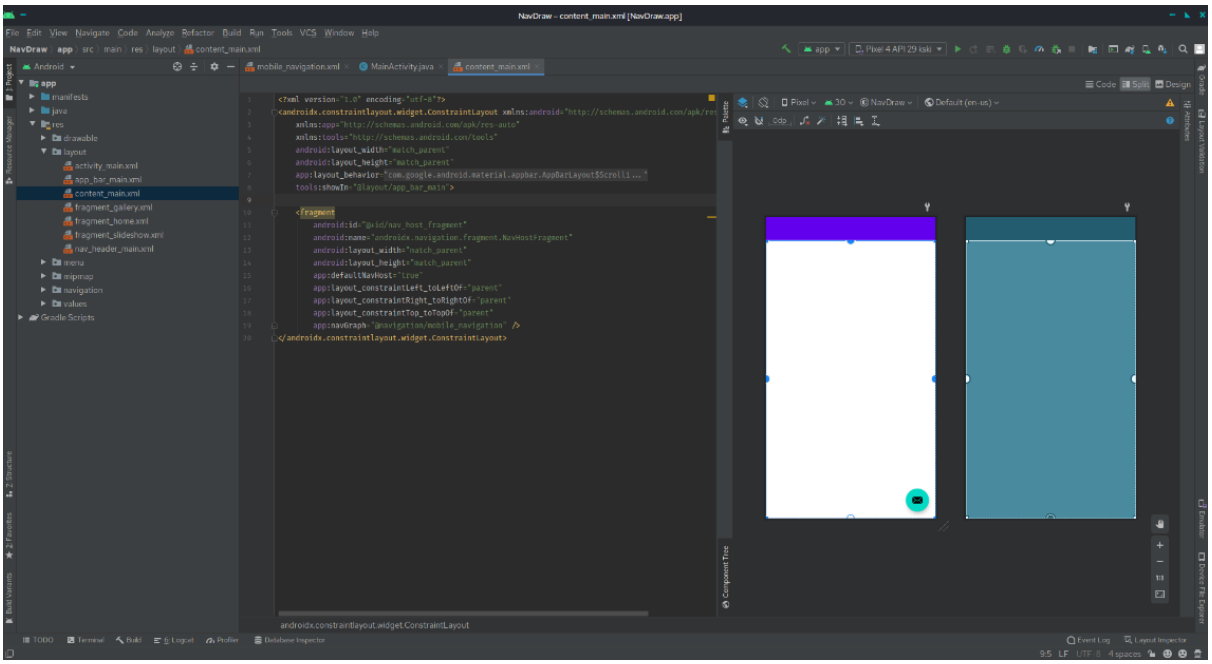
4.1. Android Studio

Prema [10], Android Studio je službeno integrirano razvojno okruženje (IDE) za razvoj Android aplikacija. Baziran na moćnom uređivaču koda i alatima za razvoj iz IntelliJ IDEA, Android Studio nudi velik broj mogućnosti koje poboljšavaju produktivnost pri izgradnji Android aplikacija. Na slici 4.1. vidljiv je izgled sučelja razvojnog okruženja.

Neke od značajki za unapređenje produktivnosti su:

- Jedinstveno okruženje za razvijanje za sve Android uređaje
- Sustav izgradnje temeljen na alatu Gradle
- Brzi i bogati emulator
- Napredni alati za testiranje

Jedna od bitnih karakteristika je podrška za programski jezik Kotlin koji drži poziciju jednog od popularnijih programskih jezika među Android programerima. Također, Android Studio podržava i tradicionalni programski jezik Javu čime se pruža fleksibilnost u odabiru jezika.



Slika 4.1. Android Studio – Izgled korisničkog sučelja

4.2. Kotlin programski jezik

Kotlin je novi, moderni, ali već zreo programski jezik osmišljen kako bi razvojnim programerima pružio veće zadovoljstvo, a sve se više koristi i za razvoj Android aplikacija. On je koncizan, siguran, interoperabilan s Javom i drugim jezicima te pruža mnoge načine za ponovnu upotrebu koda na više platformi kako bi se postigla produktivnost u programiranju [11]. Interoperabilnost s Javom jedna je od značajnijih prednosti. To znači da se postojeći Java kod može lako integrirati s Kotlinom što pojednostavljuje i olakšava postupno prebacivanje postojećih aplikacija ili korištenje oba jezika u istom projektu.

4.3. XML

XML (engl. *EXtensible Markup Language*) je proširiv jezik za označavanje kojim se zapisuju dokumenti i podaci u tekstualnom formatu. XML format je čitljiv za tekst editore kao što su MS Word ili za razne programske editore. XML pripada istoj skupini jezika kao i HTML koji je najpoznatiji iz te skupine. XML nije programski jezik kao Kotlin, Java, C++ i slično jer se dokument napisan u XML-u ne može „izvoditi“. Primjer XML koda prikazan je u kodu 4.1. [12].

Linija	***Kod***
1:	`<TextView`
2:	`android:layout_width="wrap_content"`
3:	`android:layout_height="wrap_content"`
4:	`android:text="Hello World!" />`

Programski kod 4.1. XML kod primjer

4.4. Tarsos DSP

Tarsos DSP (engl. *Digital Signal Processing*) je biblioteka za obradu zvuka napisana u Java programskom jeziku. Prema [13], biblioteka pruža praktične algoritme obrade zvuka koji su implementirani u čistoj Javi i bez ikakvih vanjskih zavisnosti na što jednostavniji mogući način. Biblioteka pokušava istovremeno pronaći najbolju granicu između sposobnosti obavljanja stvarnog zadatka i jednostavnosti i kompaktnosti.

Tarsos DSP posjeduje implementacije raznih algoritama za detekciju osnovne frekvencije tona kao što su YIN, AMDF, DYNAMIC_WAVELET, Mcleod-ov algoritam, FFT_YIN, FFT_PITCH. Također nudi algoritme za skraćivanje i produljivanje zvučnih zapisa, filtriranje zvuka, jednostavne sinteze i mnoge druge.

4.4.1. Algoritmi za detekciju osnovne frekvencije tona

Tarsos DSP biblioteka nudi šest algoritama za detekciju osnovne frekvencije tona. Ovi algoritmi implementiraju razne pristupe i tehnike u svrhu detekcije fundamentalnih frekvencija u zvučnom signalu. Ne postoji najbolji algoritam za detekciju tona, stoga se odgovarajući algoritam odabire ovisno o potrebi i karakteristici signala koji se analizira te o kvaliteti i čistoći zvuka.

Prema [14], AMDF algoritam izvlači prosječnu razliku magnituda (engl. *Average Magnitude Difference*, AMDF) iz zvučnog spremnika. Ta vrijednost predstavlja dobru mjeru visine tona signala. AMDF se izračunava kao razlika između valnog oblika koji zbraja vremenski odgođenu verziju samoga sebe.

Kod algoritma DYNAMIC_WAVELET visina tona je glavna frekvencija valnog oblika (tona koji se svira). Algoritam se ističe po preciznosti (razlika manja od 0.05 polutona), vrlo niskom vremenu odziva (ispod 23 ms) i vrlo niskoj stopi pogreške. Temeljito je testiran na ljudskom glasu [15].

Mcleod-ov algoritam (MPM) je precizna i robusna metoda za pronalaženje visine tona u monofoničnim glazbenim zvukovima. Prema [16], MPM koristi posebnu normaliziranu verziju funkcije kvadrirane razlike (engl. *Squared Difference Function*, SDF) u kombinaciji s algoritmom odabira vrhova. MPM radi u stvarnom vremenu sa standardnom brzinom uzorkovanja od 44,1 kHz. Algoritam radi bez upotrebe niskopropusnog filtriranja kako bi mogao raditi sa zvukom visokih harmoničnih frekvencija i pouzdano prikazivati promjene u visini tona od jednog centa.

YIN algoritam temelji se na poznatoj metodi autokorelacije s nizom prilagodbi koje se kombiniraju u svrhu sprječavanja pogrešaka. Stope pogrešaka su otprilike tri puta niže od najboljih konkurentnih metoda. Pogodan je za detekciju glazbe i glasova s visokim frekvencijama iz razloga što nema gornju granicu za područje pretraživanja frekvencija. Algoritam je relativno jednostavan, koristi malo parametara koji se moraju podešavati i može se učinkovito implementirati s niskim kašnjenjem [17].

Prema [18], FFT_YIN je algoritam koji koristi implementaciju YIN algoritma za praćenje visine tona uz korištenje FFT, odnosno brze Fourierove transformacije, koja u kontekstu signala omogućuje analizu frekvencijskog sadržaja signala. Ovakav pristup čini izračunavanje razlike funkcije bržim.

FFT_PITCH je algoritam čija implementacija prati visinu tona pronalaženjem najznačajnije frekvencijske komponente u signalu [19].

U ovom radu koristit će se YIN algoritam. Usporedbom i testiranjem rada ostalih algoritama zaključak je da YIN algoritam ima niže stope pogrešaka u odnosu na druge algoritme i dopušta nam velik raspon pretrage iz razloga što nema gornju granicu za područje pretraživanja frekvencija. Također, algoritam se može učinkovito implementirati s malim vremenom odziva te jednostavno daje kvalitetnije rezultate u kontekstu obrade zvuka gitare od ostalih algoritama. Detaljniji opis rada YIN algoritma objašnjen je u nastavku.

4.5. YIN algoritam

Kao što je rečeno, algoritam za detekciju osnovne frekvencije tona koji se koristi u ovom radu je YIN algoritam. Koristi se za procjenu fundamentalne frekvencije govora ili glazbenih zvukova. Prema [17], metoda se sastoji od šest koraka koji se međusobno nadograđuju u svrhu smanjivanja stope pogrešaka.

1. Autokorelacijska metoda

   Funkcija autokorelacije diskretnog signala x_t definira se kao:

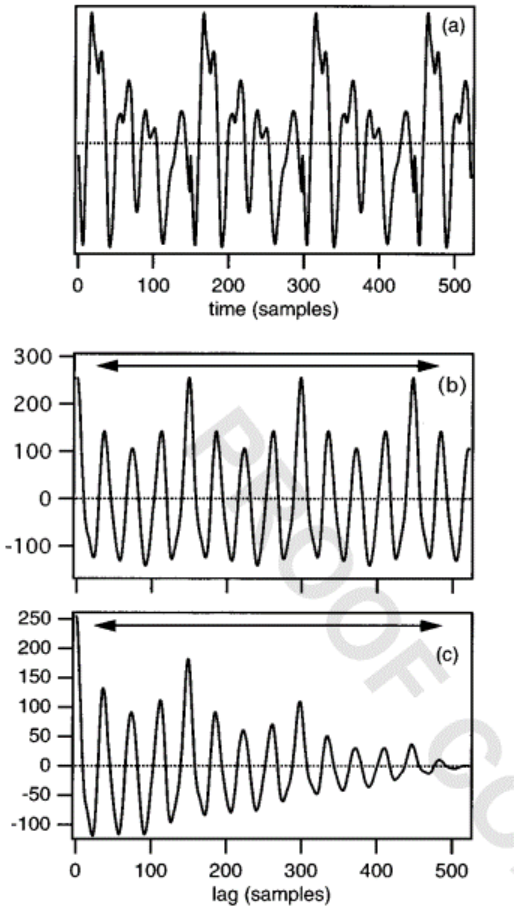
$$r_t(\tau) = \sum_{j=t+1}^{t+W} x_j x_{j+\tau} \quad za\ j = 0,1,\dots,W \tag{4-1}$$

   gdje je $r_t(\tau)$ funkcija autokorelacije za kašnjenje (engl. *lag*) označeno s τ, a W je veličina prozora za integraciju. Ova funkcija mjeri i uspoređuje signal s njegovom pomaknutom verzijom.

   Kod obrade signala često se koristi malo drugačija definicija:

$$r'_t(\tau) = \sum_{j=t+1}^{t+W-\tau} x_j x_{j+\tau} \quad za\ j = 0,1,\dots,W \tag{4-2}$$

Funkcije autokorelacije nad nacrtanim primjerom signala govora dobivene s dvije prethodno napisane formule prikazane su na slici 4.2. Druga funkcija se sužava prema nuli zbog manjeg broja članova u sumiranju pri većim vrijednostima τ. Horizontalne strelice simboliziraju raspon pretrage za period.



Slika 4.2. a) Prikaz primjera signala govora, b) Funkcija autokorelacije nad signalom prema formuli (4-1), c) Funkcija autokorelacije nad signalom prema formuli (4-2) [17]

Kod periodičnog signala funkcija autokorelacije pokazuje vrhove u višekratnicima perioda. Metoda autokorelacije bira najveći vrh s nenultim kašnjenjem putem potpune pretrage unutar raspona kašnjenja. Međutim, ako je donja granica preblizu nule, algoritam može pogrešno odabrati vrh s nultim kašnjenjem. Isto tako ako je gornja granica previsoka, algoritam može pogrešno odabrati vrh višeg reda. Sljedeći koraci osmišljeni su s ciljem smanjivanja stope pogrešaka.

2. Funkcija razlike

   Započinje se modeliranjem signala x_t kao periodične funkcije s periodom T koja je definirana kao invarijantna za pomak vremena T:

$$x_t - x_{t+\tau} = 0, \qquad za\ svaki\ t \tag{4-3}$$

   Funkcija razlike dobiva se računanjem kvadratne razlike između trenutnog dijela signala i pomaknutog dijela signala za različite vrijednosti pomaka τ:

$$d_t(\tau) = \sum_{j=1}^{W} (x_j - x_{j+\tau})^2 \tag{4-4}$$

   Cilj je pronaći vrijednost τ za koje je funkcija razlike jednaka nuli. Postoji beskonačan broj takvih vrijednosti gdje je svaka vrijednost višekratnik perioda. Funkcija razlike se može izraziti i koristeći funkciju autokorelacije signala:

$$d_t(\tau) = r_t(0) + r_{t+\tau}(0) - 2r_t(\tau) \tag{4-5}$$

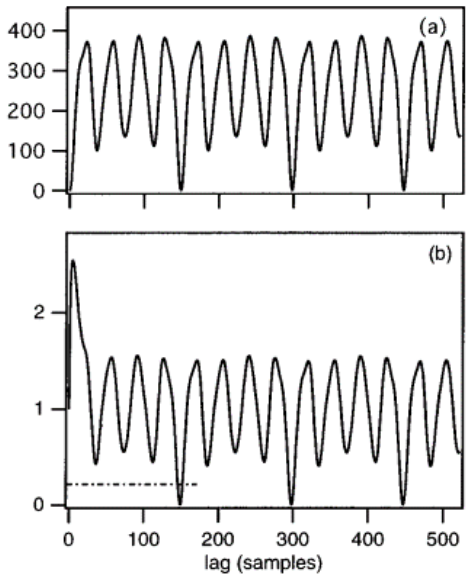
   Ovime stopa pogreške pada na 1,95 % za funkciju razlike u odnosu na 10,0 % za funkciju autokorelacije. Razlog tomu jest što je funkcija autokorelacije prilično osjetljiva na promjene amplitude signala zbog toga što povećanje amplitude signala s vremenom uzrokuje rast amplitude vrhova funkcije autokorelacije s pomakom umjesto da ostanu konstanti. Funkcija razlike je imuna na takav problem. Također, funkcija razlike otvara put za izvedbu sljedeća dva koraka koji se bave „previsokim“ i „preniskim“ pogreškama.

3. Kumulativna srednja normalizirana funkcija razlike

   Funkcija razlike ima vrijednost 0 na nultom pomaku i često ima vrijednosti različite od 0 na periodama zbog nesavršenosti periodičnosti signala. Bez postavljanje donje granice u pretraživanju, algoritam bi odabrao najdublji jarak na nultom pomaku umjesto na periodi što će rezultirati pogrešnim rezultatom. No, i u slučaju postavljanja donje granice, snažna rezonancija prvog formanta može prouzrokovati sekundarne jarke koji mogu biti dublji od periodičnih jaraka. Stoga, postavljanje donje granice prilikom pretraživanja nije zadovoljavajući način rješavanja ovog problema. Rješenje je zamjena razlike funkcije kumulativnom srednjom normaliziranom funkcijom razlike ($d'_t(\tau)$).

$$d'_t(\tau) = \begin{cases} 1, & ako\ je\ \tau = 0 \\ \dfrac{d_t(\tau)}{\left(\frac{1}{\tau}\right)\sum_{j=1}^{\tau} d_t(j)} & inače \end{cases} \tag{4-6}$$

Kumulativna srednja normalizirana funkcija razlike dobiva se dijeljenjem svake vrijednosti osnovne funkcije s njenim prosjekom u kraćim intervalima. Nova funkcija započinje s vrijednošću 1 te opada ispod 1 samo gdje funkcija razlike pada ispod prosjeka. Stopa pogreške smanjuje se sa 1,95 % na 1,69 %. Dodatne prednosti su uklanjanje gornje granice frekvencijskog područja pretraživanja kako bi se izbjegao nulti jarak i normalizacija funkcije za naredni korak u smanjenju pogrešaka. Prikaz funkcije razlike i kumulativne srednje normalizirane funkcije razlike nad primjerom signala sa slike 4.2 prikazan je na slici 4.3. Treba primijetiti da prikaz funkcije pod b) počinje s vrijednosti 1 umjesto 0 i ostaje visoka sve do pada na periodu.



Slika 4.3. a) Funkcija razlike nad signalom, b) Kumulativna srednja normalizirana funkcija razlike nad signalom [17]

4. Apsolutni prag

   Zadatak ovog koraka je rješavanje problema u slučaju kada je jedan od jarkova višeg reda u funkciji razlike dublji od periodičnog jarka. Ako takav jarak pada unutar područja pretrage, dolazi do subharmonijske pogreške. Predloženo rješenje je postavljanje apsolutnog praga i odabir najmanje vrijednosti τ koja daje minimum kumulativne srednje normalizirane funkcije dublji od tog praga. Ako nije pronađena takva vrijednost, odabire se globalni minimum.

   Prag od 0.1 utječe na smanjenje pogreške s 1,69 % na 0,78 % zbog smanjenja „preniskih“ pogrešaka uz vrlo malo povećanje „previsokih“ pogrešaka.

5. Parabolična interpolacija

   Prethodno navedeni koraci funkcioniraju u slučaju kada je period višekratnik perioda uzorkovanja. Ako to nije slučaj, procjena može biti pogrešna do polovice perioda uzorkovanja. Također, veća vrijednost kumulativne srednje normalizirane funkcije uzorkovana izvan jarka može utjecati na proces odabira jarka, što može dovesti do značajne pogreške.

   Rješenje je parabolična interpolacija. Lokalni minimumi kumulativne srednje normalizirane funkcije i njihovi susjedi aproksimiraju se parabolama čime se dobiva točnija procjena perioda.

   Također, parabolična interpolacija koristi se u svrhu izbjegavanja pogrešaka vezanih za jake komponente visokih frekvencija u signalu. Metoda je imala malen učinak na pogrešku s bazom podataka (smanjenje sa 0,78 % na 0,77 %), moguće iz razloga jer su F_0-ovi manji u usporedbi s frekvencijom uzorkovanja. No, testiranje sa sintetskim stimulima dalo je zaključak da metoda parabolične interpolacije povećava preciznost nad svim F_0-ovima te izbjegava veće pogreške kod viših F_0-ova.

6. Najbolja lokalna procjena

   Cilj ovog koraka je osigurati stabilne procjene koje ne osciliraju na vremenskoj skali temeljnog perioda signala na način da se istražuje okolina svake točke analize u potrazi za boljom procjenom. Algoritam oko svakog vremenskog indeksa t pretražuje male intervale $[t - \frac{T_{max}}{2},\ t + \frac{T_{max}}{2}]$ i traži minimum funkcije $d'_\theta(T_\theta)$ gdje je T_θ procjena perioda u vremenu, a T_{max} je najveći očekivani period. Na temelju inicijalne procjene, algoritam za procjenu primjenjuje se ponovno, no s ograničenim rasponom pretraživanja u svrhu dobivanja konačnog rezultata. Korištenjem $T_{max} = 25\ ms$ i konačnog raspona pretraživanja od ± 20 % od inicijalne procjene, ovaj korak smanjuje stopu pogreške s 0.77 % na 0.5 %.

Kombinacija koraka od 1 do 6 čini YIN algoritam. Koraci se nadograđuju jedan na drugi. Zamjena autokorelacijske metode (1) s funkcijom razlike (2) priprema put za kumulativnu srednju normaliziranu funkciju razlike (3) na kojoj se temelji apsolutni prag (4) i koja kasnije služi za najbolju lokalnu procjenu (6). Parabolična interpolacija (5) je korak koji je neovisan o ostalim koracima, iako se oslanja na spektralne osobine autokorelacijske metode [17].

5. RAZVOJ APLIKACIJE I EVALUACIJA REZULTATA

U ovom poglavlju detaljno je objašnjen razvoj Android aplikacije za ugađanje gitare putem ugrađenog mikrofona u pametnom telefonu. U narednim potpoglavljima bit će opisano dodavanje Tarsos DSP biblioteke i potrebnih dozvola, izgled aplikacije postignut XML jezikom za označavanje, funkcionalnosti postignute Kotlin programskim jezikom te evaluacija rezultata testiranja. Aplikacija je razvijena u Android Studio razvojnom okruženju.

5.1. Dodavanje Tarsos DSP biblioteke

Tarsos DSP biblioteka dodaje se pomoću alata Gradle. Gradle predstavlja napredni set alata koji se koriste za automatizaciju i upravljanje procesom izgradnje dok u isto vrijeme dopušta definiranje prilagođenih konfiguracija izgradnje. Svaka konfiguracija izgradnje definira vlastiti skup koda i resursa dok istovremeno koristi zajedničke dijelove za sve verzije aplikacije [20]. Kako bi se biblioteka mogla koristiti potrebno je implementirati određene stavke unutar Android Studio-a u skripte *build.gradle* i *settings.gradle*. U kodu 5.1. prikazane su sve potrebne implementacije zajedno s implementacijom Tarsos DSP biblioteke unutar *build.gradle* skripte. U kodu 5.2. vidljiva je *settings.gradle* skripta gdje je bilo potrebno dodati naziv i putanju do Tarsos DSP repozitorija na Internetu.

```
dependencies {
    implementation 'androidx.core:core-ktx:1.7.0'
    implementation 'androidx.appcompat:appcompat:1.5.1'
    implementation 'com.google.android.material:material:1.7.0'
    implementation 'androidx.constraintlayout:constraintlayout:2.1.4'
    testImplementation 'junit:junit:4.13.2'
    androidTestImplementation 'androidx.test.ext:junit:1.1.5'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.5.1'
    implementation 'be.tarsos.dsp:core:2.5'
    implementation 'be.tarsos.dsp:jvm:2.5'
}
```

Programski kod 5.1. Prikaz *build.gradle* skripte

```
dependencyResolutionManagement { DependencyResolutionManagement it ->
    repositoriesMode.set(RepositoriesMode.FAIL_ON_PROJECT_REPOS)
    repositories { RepositoryHandler it ->
        google()
        mavenCentral()
        maven {
            name = "TarsosDSP repository"
            url = "https://mvn.0110.be/releases"
        }
    }
}
```

Programski kod 5.2 Prikaz *settings.gradle* skripte

5.2. Dodavanje dozvola

Unutar *AndroidManifest.xml* datoteke potrebno je dodati dozvolu za korištenje mikrofona. Prema [21], *AndroidManifest.xml* je datoteka koja opisuje osnovne informacije o aplikaciji, Android alatima za izgradnju, Android operacijskom sustavu i Google Play-u. Ovaj korak potrebno je napraviti kako bi aplikacija mogla primati ulazni signal mikrofona uređaja. U kodu 5.3. prikazana je linija koda kojom se dozvola dodaje.

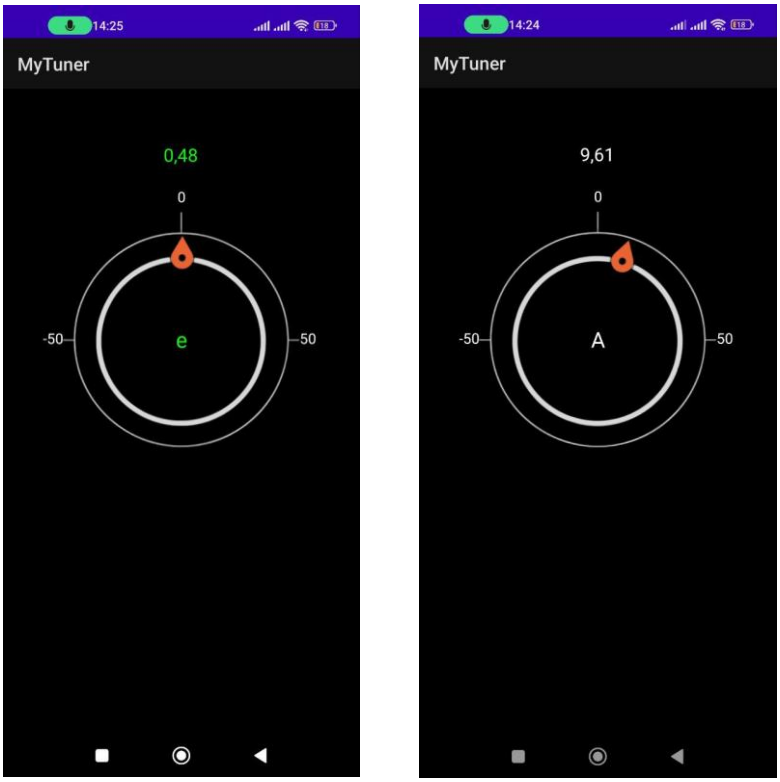
```
<uses-permission android:name="android.permission.RECORD_AUDIO" />
```

Programski kod 5.3. Prikaz dozvole u *AndroidManifest.xml* datoteci

5.3. Izgled aplikacije

Za prikaz procesa ugađanja tona gitare koristi se jedna aktivnost koja prikazuje kotač čija je zadaća pokazati preciznost tona snimljenog mikrofonom uređaja. Na zaslonu se također prikazuje naziv note prema kojoj se ugađa te iznos u centima. Cent [c] je mjerna jedinica kojom se mjeri preciznost tijekom ugađanja tona. Jedan cent predstavlja stoti dio razlike između dva polutona. Stoga, na kotaču je dio kružnice označen brojevima -50 i 50 po kojoj se kreće pokazivač i prikazuje odstupanje od traženog tona kojeg je cilj poravnati s brojem 0 kako bi se postigao ugođen ton. Također, točnost tona signalizira i promjena boje naziva note i iznosa u centima u zelenu boju.

Na slici 5.1. prikazan je glavni zaslon aplikacije s kotačem u dva slučaja kada je ton ugođen i kada nije ugođen. Realizacija zaslona postignuta je u *activity_main.xml* datoteci korištenjem 2 *ImageView* elementa i 5 *TextView* elemenata.

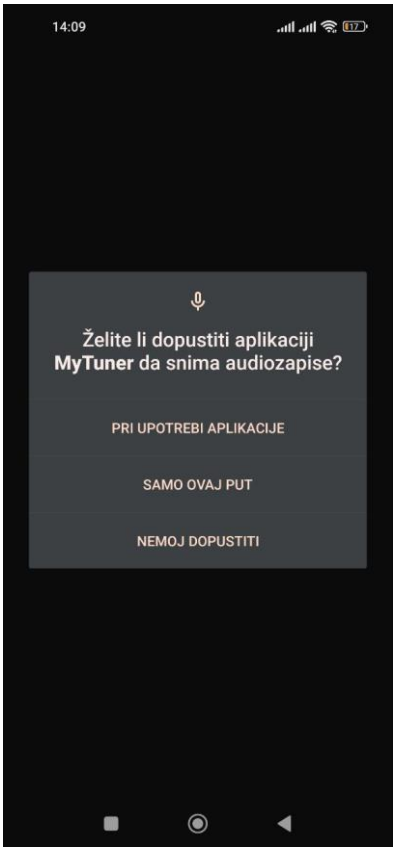


Slika 5.1. Prikaz glavnog zaslona u slučaju ugođenog i neugođenog tona

5.4. Funkcionalnost aplikacije

Aplikacija se sastoji od jedne aktivnosti napisane Kotlin programskim jezikom unutar datoteke *MainActivity.kt*. Prvi dio koda nalazi se unutar *onCreate()* metode koja se poziva pri prvom stvaranju aktivnosti i u ovom slučaju koristi se za inicijalizaciju animacije kotača, prikaza tona u centima, prikaza note i zatraživanja dozvole korisnika za korištenje mikrofona.

Metoda *requestRecordAudioPermission()* služi za pokretanje zatraženja dozvole, vidljivo na slici 5.2., i osigurava inicijalizaciju objekta klase *AudioRecord* koja služi za snimanje tona putem mikrofona.



Slika 5.2. Prikaz zatraženja dozvole

Nadalje, unutar metode *onStart(),* koja se poziva kada je aktivnost vidljiva korisniku, poziva se metoda *startRecording().* Ova metoda je odgovorna za početak procesa snimanja zvuka i rukovanje detektiranim tonom koristeći Tarsos DSP biblioteku.

Funkcija *startRecording()* sastoji se od 4 osnovna dijela:

- Inicijalizacija potrebnih varijabli i objekata
- Inicijalizacija *pitchDetectionHandlera* i njegove *handlePitch* metode
- Inicijalizacija *pitchProcessora* i odabir YIN algoritma za detekciju
- Pokretanje niti za obradu zvuka snimljenog u *buffer* i procesiranje

Na vrhu funkcije postavljena je varijabla *isRecording* koja predstavlja zastavicu za kontroliranje trajanja procesa snimanja zvuka. U liniji 123 u programskom kodu 5.4. inicijalizirana je varijabla *buffer* u koju se učitava zvuk s mikrofona. Tip podataka *ShortArray* odabran je iz razloga što klasa *AudioRecord* korištena za snimanje zvuka snimljeni zapis sprema u tom obliku. Veličina *buffera* definirana je pomoću vrijednosti varijable *bufferSize* koja je postavljena na početku koda klase *MainActivity* zajedno s ostalim često korištenim varijablama. Zajedno s njom definirana je

i varijabla *sampleRate* koju je važno spomenuti uz varijablu *bufferSize*. Vrijednost varijable *sampleRate* predstavlja koliko će uzoraka zvučnih podataka biti snimljeno po sekundi, dok vrijednost *bufferSize* varijable predstavlja broj uzoraka koji će biti spremljeni u *buffer* prije procesiranja. Varijabla *sampleRate* iznosi 44 100 što je standard za snimanje zvuka, dok *bufferSize* iznosi 7056 što predstavlja, u ovom slučaju, najbolji omjer što se tiče količine podataka u međuspremniku za procesiranje i brzine odziva. Zatim, *TarsosDSPAudioFormat* definira format koji se predaje kao argument objektu klase *AudioEvent* koji je dio Tarsos DSP biblioteke i služit će za kontrolu međuspremnika i prosljeđivanje na daljnje procesiranje.

Pri inicijalizaciji objekta klase *PitchProcessor* u liniji 158, koji se koristi za obradu zvuka, u argumentima se odabire YIN algoritam za detekciju, *sampleRate*, *bufferSize* i prethodno definirani *pitchDetectionHandler*.

```
119     private fun startRecording() {
120
121         /* Inicijalizacija varijabli i objekata */
122         isRecording = true
123         val buffer = ShortArray(bufferSize)
124         audioRecord.startRecording()
125         val tarsosDSPAudioFormat = TarsosDSPAudioFormat(sampleRate.toFloat(),
126             sampleSizeInBits: 16, channels: 1, signed: true, bigEndian: false)
127         val audioEvent = AudioEvent(tarsosDSPAudioFormat)
128
129         /* Inicijalizacija pitchDetectionHandlera */
130         val pitchDetectionHandler = object : PitchDetectionHandler {
131             /* Implementacija handlePitch() callback funkcije */
132             override fun handlePitch(pitchDetectionResult: PitchDetectionResult, audioEvent: AudioEvent) {...}
155         }
156
157         /* Inicijalizacija pitchProcessora */
158         val pitchProcessor = PitchProcessor(PitchProcessor.PitchEstimationAlgorithm.YIN,
159             sampleRate.toFloat(), bufferSize, pitchDetectionHandler)
160
161         /* Pokretanje niti koja učitava zvuk s mikrofona i procesira ga */
162         Thread {...}.start()
174     }
```

Programski kod 5.4. Prikaz metode *startRecording()*

PitchDetectionHandler model definira *callback* funkciju *handlePitch()* koja će biti objašnjenja kasnije zbog lakšeg razumijevanja redoslijeda odvijanja koda.

Potom se pokreće nit koja se aktivira metodom *start()* te se odvija u pozadini i kontinuirano čita snimljeni zvuk. Odgovorna je za procesiranje podataka i provođenje detekcije tona. Kod niti

prikazan je u programskom kodu 5.5. Prvotno je potrebno inicijalizirati varijablu *floatBuffer* tipa podatka *FloatArray* iz razloga što objekt *audioEvent* koji prosljeđuje podatke za procesiranje očekuje podatke u tom obliku. U niti se nalazi *while* petlja koja se odvija sve dok je *isRecording* varijabla *true*. Unutar petlje u liniji 163 audiozapis se iz objekta *audioRecord* čita u *buffer*. Zatim je potrebno pretvoriti svaki snimljeni podatak tipa podatka *short* u tip podatka *float* u rasponu [-1.0, 1.0] na način da se broj dijeli s 32 768 jer to predstavlja raspon *short* tipa podataka. Konačno se može pokrenuti metoda *process()* na *pitchProcessoru*.

```
160         Thread {
161             val floatBuffer = FloatArray(bufferSize)
162             while (isRecording) {
163                 audioRecord.read(buffer,  offsetInShorts: 0, bufferSize)
164                 for (i in 0 ≤  until  < bufferSize) {
165                     floatBuffer[i] = buffer[i].toFloat() / 32768.0f // Pretvaranje short u float u rasponu [-1.0, 1.0]
166                 }
167 
168                 audioEvent.floatBuffer=floatBuffer
169                 pitchProcessor.process(audioEvent)
170             }
171         }.start()
172     }
```

Programski kod 5.5. Prikaz koda u *Threadu*

Metoda *process()* okida metodu *handlePitch()* koja pripada *pitchDetectionHandleru* čiji je kod prikazan u kodu 5.6. Kod se odvija unutar posebne niti za korisničko sučelje koja se pokreće s *runOnUiThread*. To omogućuje ažuriranje korisničkog sučelja rezultatima detekcije tona te osiguravanje responzivnosti dok se odvija pozadinsko snimanje i procesiranje. Metoda najprije sprema procesirane frekvencije u hercima unutar međuspremnika *circularBuffer* veličine 8 spremljene u vrijednosti *circularBufferSize* na početku koda. Nakon toga u liniji 138 vidi se poziv metode *calculatePitch()*.

```
130 ⓘ↑      override fun handlePitch(pitchDetectionResult: PitchDetectionResult, audioEvent: AudioEvent) {
131             val pitchInHz=pitchDetectionResult.pitch
132             runOnUiThread {
133                 if(pitchInHz != -1F) {
134                     val isInSpan = spans.any { (_, spanRange) -> pitchInHz in spanRange }
135                     if (isInSpan) {
136                         circularBuffer[circularBufferIndex] = pitchInHz
137                         if (circularBufferIndex == circularBufferSize - 1) {
138                             calculatePitch()
139
140                         }
141                         circularBufferIndex = (circularBufferIndex + 1) % circularBufferSize
142                     }
143                 }
144                 else{
145                 /* Postavi default vrijednosti */
146                 }
147             }
148         }
149     }
```

Programski kod 5.6. Prikaz metode *handlePitch()*

Logika kod odabira spremanja frekvencija jest provjera je li frekvencija u ikojem od šest prethodno inicijaliziranih listi gdje svaka pojedinačna lista predstavlja raspon od 50 cenata (pojam centa pojašnjen je u potpoglavlju 5.3.) prije i nakon određenog tona žice standardnog *štima* gitare. Svaka lista sadrži raspon te odgovarajući ključ tog raspona. Ako je frekvencija unutar listi, sprema se u međuspremnik. Inicijalizacija listi unutar varijable *spans* vidljiva je u kodu 5.7.

```
44      private val spans = listOf(
45          "E_span" to 80.06f..84.82f,
46          "A_span" to 106.87f..113.22f,
47          "D_span" to 142.65f..151.13f,
48          "G_span" to 190.42f..201.74f,
49          "B_span" to 239.91f..254.18f,
50          "e_span" to 320.25f..339.29f
51      )
```

Programski kod 5.7. Prikaz varijable *spans*

Unutar metode *calculatePitch()*, koja se vidi u kodu 5.8., najprije se u varijablu *majoritySpan* sprema izlaz iz metode *getMajoritySpan()* koja vraća raspon s najviše primjeraka u međuspremniku. Prethodni korak potreban je jer se zatim u liniji 202 filtriraju frekvencije koje

pripadaju tom rasponu. Zatim se računa prosječna vrijednost kako bi ugađanje tona bilo uglađenije i preciznije. Završni korak prikazan u liniji 211 jest računanje vrijednosti u centima, odnosno vrijednost udaljenosti od željene frekvencije. Za računanje potrebno je poznavati prosječnu vrijednost frekvencije i referentnu frekvenciju željene žice. Varijabla *referenceFrequencyDict* jest prethodno inicijalizirani rječnik koji sadrži frekvenciju svake žice standardnog *štima* gitare.

```
200     private fun calculatePitch(){
201         val majoritySpan = getMajoritySpan(circularBuffer)
202         val filteredFrequencies = circularBuffer.filter { frequency ->
203             frequency in spans.first { it.first == majoritySpan }.second
204         }
205
206         var sum = 0f
207         for (element in filteredFrequencies) {
208             sum += element
209         }
210         val average=sum / filteredFrequencies.size
211         val pitchInCents=1200* log( x: average/ referenceFrequencyDict[majoritySpan[0]]!!, base: 2.0)
212
213         displayPitchAndNote(pitchInCents, majoritySpan)
214     }
```

Programski kod 5.8. Prikaz metode *calculatePitch()*

Konačno se poziva metoda *displayPitchAndNote()* koja provjerava je li vrijednost izračunatih cenata u dopuštenom rasponu, u ovom slučaju između -6 i 6 (razlog tomu pojašnjen je u potpoglavlju 5.5.), te obavlja ažuriranje *TextView* elemenata i animacije kotača.

Na kraju unutar *onStop()* metode, koja se koristi kada aktivnost prestaje biti vidljiva korisniku, poziva se metoda *stopRecording()* unutar koje se zaustavlja snimanje zvuka te se u liniji 239, vidljive u kodu 5.9., zaustavljaju animacije.

```
230     private fun stopRecording() {
231         isRecording = false
232         audioRecord.stop()
233         audioRecord.release()
234     }
235
236     override fun onStop() {
237         super.onStop()
238         stopRecording()
239         wheelImageView.clearAnimation()
240     }
```

Programski kod 5.9. Prikaz metoda *stopRecording()* i *onStop()*

5.5. Testiranje rada aplikacije i evaluacija rezultata

Bitan korak izrade aplikacije za ugađanje tona gitare jest testiranje postoje li i kolika su odstupanja između frekvencije tona koju proizvodi gitara i frekvencije koju aplikacija prikazuje. Ako je odstupanje preveliko, aplikacija neće biti pouzdana za uporabu.

Kod instrumenata poput gitare često se nailazi na odstupanja u visini tona od tražene visine zbog različitih vanjskih čimbenika poput temperaturne razlike, vlage i trenja. Prema [22], ljudsko uho može primijetiti razliku između 2 tona ako je između njih minimalna razlika od 5-6 cenata (pojam centa pojašnjen je u potpoglavlju 5.3.). Takva odstupanja smatraju se dovoljno dobrima za amaterske glazbenike, dok profesionalni glazbenici traže veću preciznost. Obično svaki štimer ovisno o kvaliteti ima i određenu toleranciju u centima. Kvalitetniji štimeri imaju puno veću preciznost od drugih pa je i odstupanje manje. Kao što se vidjelo u razvoju aplikacije, naša aplikacija ima dopušteno odstupanje od ± 6 cenata.

Za testiranje koristit će se fizički štimer Flight FTC-77 budući da je ovaj štimer pouzdan i već dugo na tržištu. Iz tog razloga će se njegov izlaz uzimati kao referentna vrijednost. Ovaj štimer ima dopušteno odstupanje od ± 1 centa. Prvi korak bit će raštimati gitaru u kontroliranim uvjetima. Zatim će se pomoću štimera provesti ugađanje tona gitare te će se takav ugođen ton iskoristiti kako bi se izmjerilo odstupanje naše mobilne aplikacije i ostalih aplikacija prethodno navedenih u postojećim rješenjima. U tablici 5.1. vide se prosječni rezultati nakon nekoliko provedenih mjerenja.

Tablica 5.1. Prikaz prosječnih odstupanja aplikacija u centima po notama

Nota	Odstupanje po aplikaciji / [c]			
	MyTuner	GuitarTuna	Pro Guitar Tuner	gStrings
e	0,50	0,80	1,20	0,20
B	1,20	0,80	1,00	0,50
G	2,00	1,30	1,50	0,80
D	-1,80	-1,50	-1,20	-1,00
A	2,50	1,80	2,00	1,50
E	-3,50	-2,40	-2,80	-1,20

Analizirajući rezultate vidljivo je kako naša aplikacija realizirana uz pomoć Tarsos DSP biblioteke daje rezultate koji konkuriraju drugim aplikacijama na tržištu. Potrebno je naglasiti kako rezultati također ovise i o kvaliteti same gitare i njenih žica te o jačini okidanja žica gitare. Primjerice, jače okidanje žice može dovesti do većih oscilacija u frekvenciji te izlaz aplikacije neće biti pouzdan dok se titranje žice ne uravnoteži.

6. ZAKLJUČAK

Cilj ovog rada bila je realizacija Android aplikacije za ugađanje tona gitare putem ugrađenog mikrofona u pametnom telefonu. Pojašnjene su osnovne informacije o zvuku i tonu, kao i o gitari i procesu ugađanja gitare. Osim toga, pregledane su postojeće aplikacije na Google Play trgovini koja pružaju slična rješenja kako bi se dobio uvid u konkurenciju i najbolje prihvaćene prakse.

Aplikacija je realizirana unutar Android Studio integriranog razvojnog okruženja te je korištena Tarsos DSP biblioteka za obradu zvuka. Objašnjen je rad YIN algoritma koji je korišten u aplikaciji za detekciju osnovne frekvencije tona. Ovaj algoritam odabran je nakon usporedbe i testiranja svakog dostupnog algoritma te je zaključeno da YIN daje najkvalitetnije rezultate u kontekstu obrade zvuka gitare. Nakon uspješne realizacije prikazan je način provedbe izgleda i funkcionalnosti aplikacije. Izrada korisničkog sučelja postignuta je putem XML jezika za označavanje. Kotlin programski jezik korišten je za implementaciju logike aplikacije, iskorištavajući pogodnosti Tarsos DSP biblioteke. Zatim su provedena testiranja i analiza rezultata naše aplikacije u usporedbi s prethodno spomenutim sličnim aplikacijama. Testiranje je pokazalo kako je naša aplikacija po preciznosti u rangu s drugim popularnim aplikacijama, a istovremeno je jednostavna za korištenje. Odlična je opcija za amaterske glazbenike, no profesionalni glazbenici bi se ipak mogli odlučiti za uređaj ili aplikaciju sa što manjom tolerancijom na odstupanje od željenog tona.

Aplikaciji je moguće dodatno unaprijediti izgled i dodati različite funkcionalnosti poput alternativnih štimova gitare ili metronoma koji mogu biti korisni glazbenicima. Također, moguće je implementirati i opciju ručnog ugađanja tona umjesto automatskog koje je implementirano u ovoj aplikaciji, što bi omogućilo veću prilagodljivost. Nadalje, u aplikaciji se mogu implementirati drugi algoritmi za detekciju osnovne frekvencije, promijeniti brzina uzorkovanja i druge značajke ovisno o specifičnim potrebama korisnika.

LITERATURA

[1] B. Benward i M. Saker, Music in Theory and Practice Volume 1, McGraw-Hill, New York, 2020., dostupno na: https://parkarts.pbworks.com/w/file/fetch/120179109/Music%20Theory%20and%20Practice%20Textbook.pdf

[2] Hrvatska enciklopedija, zvuk, mrežno izdanje. Leksikografski zavod Miroslav Krleža, 2021., dostupno na: https://www.enciklopedija.hr/natuknica.aspx?ID=67594 [Posljednje pristupljeno: 19.8.2023.]

[3] Hrvatska enciklopedija, ton, mrežno izdanje. Leksikografski zavod Miroslav Krleža, 2021., dostupno na: https://www.enciklopedija.hr/natuknica.aspx?ID=61734 [Posljednje pristupljeno: 19.8.2023.]

[4] R. Denyer, The Guitar Handbook: A Unique Source Book for the Guitar Player – Amateur or Professional, Acoustic or Electric, Rock, Blues, Jazz, or Folk, Dorling Kindersley Limited, London, 1982., dostupno na: https://pdfcoffee.com/the-guitar-handbook-by-ralph-denyer-2-pdf-free.html

[5] Skupina autora, Guitar tunings: Standard, Wikipedia, 2023. dostupno na: https://en.wikipedia.org/wiki/Guitar_tunings [Posljednje pristupljeno: 18.9.2023.]

[6] Yousician Ltd., GuitarTuna, Google Play, 2023., dostupno na: https://play.google.com/store/apps/details?id=com.ovelin.guitartuna [Posljednje pristupljeno: 19.8.2023.]

[7] ProGuitar, Pro Guitar Tuner, Google Play, 2023., dostupno na: https://play.google.com/store/apps/details?id=com.jrinnovation.proguitartuner [Posljednje pristupljeno: 19.8.2023.]

[8] Cohortor.org, Tuner - gStrings, Google Play, 2023., dostupno na: https://play.google.com/store/apps/details?id=org.cohortor.gstrings [Posljednje pristupljeno: 19.8.2023.]

[9] FLIGHT FTC-77 Clip-on kromatski štimer, Music Max, dostupno na: https://musicmax.hr/flight-ftc77-clipon-kromatski-stimer~p10960hr/ [Posljednje pristupljeno: 19.8.2023.]

[10] Meet Android Studio, Developers, 2023., dostupno na: https://developer.android.com/studio/intro [Posljednje pristupljeno: 19.8.2023.]

[11] Get started with Kotlin, Kotlin, 2023., dostupno na: https://kotlinlang.org/docs/getting-started.html [Posljednje pristupljeno: 19.8.2023.]

[12] D. Kirasić, XML tehnologija i primjena u sustavima procesne informatike, MIPRO, str. 1, Opatija, 2005., dostupno na: https://www.fer.unizg.hr/_download/repository/mipro_xml_tekst.pdf

[13] J. Six, TarsosDSP, Github, 2012., dostupno na: https://github.com/JorenSix/TarsosDSP [Posljednje pristupljeno: 19.8.2023.]

[14] E. Souza, J. Six, Class AMDF, Joren Six, dostupno na: https://0110.be/releases/TarsosDSP/TarsosDSP-2.4/TarsosDSP-2.4-Documentation/index.html?be/tarsos/dsp/pitch/AMDF.html [Posljednje pristupljeno: 19.8.2023.]

[15] A. Schmitt, J. Six, Class DynamicWavelet, Joren Six, dostupno na: https://0110.be/releases/TarsosDSP/TarsosDSP-2.4/TarsosDSP-2.4-Documentation/be/tarsos/dsp/pitch/DynamicWavelet.html [Posljednje pristupljeno: 19.8.2023.]

[16] P. McLeod, J. Six, Class McLeodPitchMethod, Joren Six, dostupno na: https://0110.be/releases/TarsosDSP/TarsosDSP-2.4/TarsosDSP-2.4-Documentation/be/tarsos/dsp/pitch/McLeodPitchMethod.html [Posljednje pristupljeno: 19.8.2023.]

[17] A. de Cheveigne, H. Kawahara, YIN, a fundamental frequency estimator for speech and music, Journal of the Acoustical Society of America, Volume 111, Issue 4, str. 1917-1930, Travnja 2002., dostupno na: http://recherche.ircam.fr/equipes/pcm/cheveign/ps/2002_JASA_YIN_proof.pdf [Posljednje pristupljeno: 19.8.2023.]

[18] M. Mauch, J. Six, P. Brossier, Class FastYin, Joren Six, dostupno na: https://0110.be/releases/TarsosDSP/TarsosDSP-1.9/TarsosDSP-1.9-Documentation/be/tarsos/dsp/pitch/FastYin.html [Posljednje pristupljeno: 19.8.2023.]

[19] J. Six, Class FFTPitch, Joren Six, dostupno na: https://0110.be/releases/TarsosDSP/TarsosDSP-2.4/TarsosDSP-2.4-Documentation/be/tarsos/dsp/pitch/FFTPitch.html [Posljednje pristupljeno: 19.8.2023.]

[20] Configure your build, Developers, 2023., dostupno na: https://developer.android.com/build [Posljednje pristupljeno: 19.8.2023.]

[21] App manifest overview, Developers, 2023., dostupno na: https://developer.android.com/guide/topics/manifest/manifest-intro [Posljednje pristupljeno: 19.8.2023.]

[22] D.B. Loeffler, Instrument Timbres and Pitch Estimation in Polyphonic Music, Department of Electrical and Computer Engineering, Georgia Tech, 2006.

SAŽETAK

U ovom završnom radu prikazana je izrada Android aplikacije za ugađanje tona gitare. Sučelje aplikacije omogućuje korisnički prilagođeno ugađanje tona s lako razumljivim prikazom ugađanja. Aplikacija je realizirana u Android Studio integriranom razvojnom okruženju. Za razvoj aplikacije i njezinih funkcionalnosti korišten je Kotlin programski jezik te Tarsos DSP biblioteku za obradu zvuka. Izgled i struktura sučelja aplikacije implementirani su pomoću XML opisnog jezika. Željeni cilj i način rada aplikacije uspješno su ostvareni. Koraci realizacije aplikacije i njeno testiranje objašnjeni su unutar ovog završnog rada.

Ključne riječi: Android, gitara, glazba, Kotlin, mobilna aplikacija

ABSTRACT

This final paper presents the development of an Android application for tuning guitar tones. The application's interface enables user-friendly tuning with an easily understandable tuning display. The application was implemented using the Android Studio integrated development environment. For the development of the application and its functionalities the Kotlin programming language and the Tarsos DSP library for sound processing were utilized. The layout and structure of the application's interface were implemented using the XML descriptive language. The desired objective and functionality of the application have been successfully achieved. The steps of the application's implementation and its testing are explained within this paper.

Keywords: Android, guitar, music, Kotlin, mobile application

ŽIVOTOPIS

Antonio Paradžik rođen je 04. svibnja 2001. godine u Slavonskom Brodu. Pohađao je Osnovnu školu Ivana Mažuranića u Sibinju. Nakon završetka osnovne škole upisuje Prirodoslovno – matematičku gimnaziju „Matija Mesić“ u Slavonskom Brodu. Za nastavak obrazovanja odabire Fakultet elektrotehnike, računarstva i informacijskih tehnologija u Osijeku gdje 2020. godine upisuje preddiplomski sveučilišni studij Računarstva, smjer Programsko inženjerstvo.