Web aplikacija za analizu audio podataka

Zvonar, Ivan

VISOKO UČILIŠTE ALGEBRA

DIPLOMSKI RAD

Web aplikacija za analizu audio podataka

Ivan Zvonar

Zagreb, rujan 2019.

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

Sažetak

Tema ovog diplomskog rada je izrada web aplikacije za analizu zvuka koristeći algoritam za generiranje digitalnog audio potpisa. Digitalni audio potpis služi za identifikaciju bilo kojeg audio ili video zapisa, a najčešće se koristi za prepoznavanje pjesama i dohvaćanje metapodataka o pjesmi. Algoritmi za analizu zvuka imaju mnogo primjena u svakodnevnom životu, primjerice, omogućuju korisniku da sazna sve informacije o nepoznatoj pjesmi, koju trenutno sluša na radiju. Također se koriste za prepoznavanje reklama i sadržaja koji se prenosi preko radio signala. Najbolji primjer poznate aplikacije koja radi prepoznavanje pjesme i dohvaćanje metapodataka pomoću audio potpisa je Shazam. U nastavku ovog rada će biti opisane neke od postojećih aplikacija za analizu zvuka, objasniti će se osnove analize zvuka i postupak generiranja audio potpisa. Nakon toga, objasniti će se arhitektura i tehnologije korištene za izradu web aplikacije, te dijelovi implementacije.

Ključne riječi: audio potpis, prepoznavanje pjesama, Shazam.

This paper topic is development of web application for audio analysis by using audio fingerprinting algorithm. Audio fingerprint can be used to identify any audio sample, with the most common use case being song identification. Acoustic fingerprinting algorithms have many practical uses, some of them include identifying songs, audio advertisements and radio broadcasts. Example of one of the most popular music discovery applications is Shazam. This paper will describe some of the commercially available audio fingerprinting applications, basics of sound analysis and process of generating audio fingerprint. Lastly, all of the architecture details and technologies used in development process will be described.

Ključne riječi: audio fingerprint, song identification, Shazam.

Sadržaj

1. Uvod

Digitalni sadržaj je svugdje i neizbježan je dio svakodnevnice većine ljudi. Može biti u različitim formatima i moguće ga je konzumirati kroz razne medije. Jedan od najčešćih formata digitalnog sadržaja je audio digitalni sadržaj. Popularnost i dostupnost raznih audio sadržaja čini ovaj format pogodnim za detaljniju analizu. Analiza kod audio sadržaja podrazumijeva izvlačenje informacija iz audio signala. Neke od motivacija za dobivanje informacija iz audio zapisa su međustrojna komunikacija, prikupljanje metapodataka (engl. *metadata*) i zaštita autorskih prava.

Ovaj rad opisuje metodu analize zvuka poznatu kao audio potpis (engl. *audio fingerprint*), u literaturi se također opisuje pod pojmom *acoustic fingerprint*. Audio potpis je sažeta reprezentacija cjelovitog audio sadržaja, bilo to neka pjesma, zvuk iz nekog filma ili videa na internetu. Generiranje audio potpisa ima široki spektar primjena, a najčešće se koristi za prepoznavanje pjesama i dohvaćanje metapodataka o pjesmi što je i tema ovog rada. U ovom radu opisati će se princip rada algoritma za generiranje audio potpisa, te detalji implementacije i tehnologije korištene za izradu *backend* i *frontend* modula aplikacije.

2. Aplikacije za analizu zvuka

Postoji više vrsta aplikacija za analizu i obradu zvuka. Najčešće korištene su aplikacije za obradu i uređivanje zvuka. Primjenjuju se u glazbenoj i filmskoj industriji, a neki od najpopularnijih alata za obradu zvuka su:

- Adobe Audition
- Logic Pro
- Ableton Live
- Acid Pro

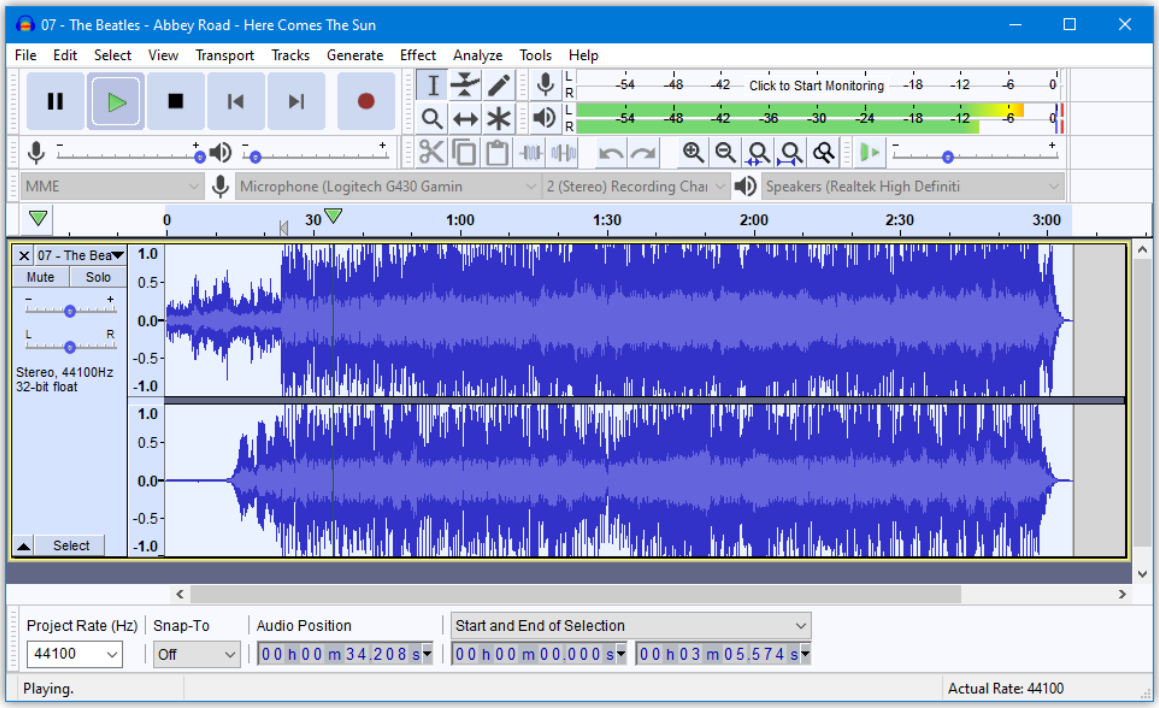
Naravno, ti alati nisu besplatni i koriste se u profesionalne svrhe. Neki od besplatnih alata za amatersku obradu zvuka su:

- WavePad
- Audacity
- Oceanaudio
- GarageBand

Svi ti alati (Digital audio editors https://en.wikipedia.org/wiki/Comparison_of_digital_audio_editors, 2019.) omogućuju obradu i kreiranje zvukova (Slika 2.1). Većina ima mogućnosti dodjeljivanja efekata, propuštanja signala kroz filtere i podešavanje kanala. Također neke od drugih funkcionalnosti koje karakteriziraju aplikacije za obradu zvuka su mogućnost uvoza (engl. *import*) i izvoza (engl. *export*) audio datoteka u raznim formatima. Snimanje zvuka iz jednog ili više ulaza. Uređivanje trajanja, odnosno dužine pjesme, te samog početka i kraja pjesme. Mogu kombinirati više izvora zvuka ili audio datoteka (engl. *audio mixing*). Za svaki izvor ili audio datoteku moguće je zasebno podešavanje karakteristika zvuka. Na svaki izvor se može dodijeliti široki spektar raznih efekata, filtera i algoritama za obradu audio signala. Moguća je i konverzija između različitih audio formata i različitih razina kvalitete zvuka.

Aplikacije mogu uređivati zvuk na destruktivan (engl. *destructive*) način ili u realnom vremenu (engl. *real-time*). Destruktivno uređivanje audio datoteke mijenja samu reprezentaciju originalne audio datoteke, što znači da se sve zadane promjene, filteri i procesiranja odmah primjenjuju na podatke. Primjer je brisanje dijela početka pjesme, nakon brisanja datoteka je odmah manja za obrisani dio. Za razliku od destruktivnog uređivanja datoteke, uređivanje u realnom vremenu ne primjenjuje zadane promjene na datoteku.

Promijenjeni podaci se drže u memoriji i sve promjene se odražavaju u letu (engl. *on the fly*) tijekom reprodukcije pjesme. Za razliku od destruktivnog uređivanja datoteke, ako se obriše bilo koji dio pjesme, reprezentacija same datoteke ostaje nepromijenjena. Obrisani podaci su samo sakriveni i biti će preskočeni tijekom reprodukcije pjesme, ali još uvijek se nalaze u memoriji. ([5].)



Slika 2.1 Prikaz „Audacity“ alata za uređivanje audio zapisa

Nakon opisa aplikacija za uređivanje zvuka, objasniti će se aplikacije za analizu zvuka koje svoj rad baziraju na generiranju audio potpisa.

Najkorištenije takve aplikacije su:

- Shazam
- SoundHound
- Chromaprint
- ACRCloud

Shazam je najpoznatija i najkorištenija aplikacija za prepoznavanje pjesama, filmova, audio reklama i televizijskih emisija. Ima mogućnost prepoznavanja glazbe iz bilo kojeg izvora pod uvjetom da nema previše pozadinske buke i da se pjesma za koju se uzima uzorak nalazi u bazi podataka.

Tvrtka koja je razvila uslugu i algoritam pokrenuta je 1999. godine, a servis ulazi u produkciju 2002. godine. U ranim danima usluga je bila dostupna samo preko broja koji su

korisnici imali mogućnost nazvati da bi saznali koja se pjesma trenutno reproducira u njihovom okruženju. Nakon uspostavljanja poziva, zvuk bi se prenosio do servera koji radi audio potpis uzorka. Poziv bi trajao 15 sekundi i nakon toga bi automatski završio. Rezultat identifikacije pjesme bi se poslao korisniku putem SMS poruke, a sadržavao je naziv pjesme i autora. Kasnije je u rezultat dodana i poveznica (engl. *link*) na pjesmu preko koje je korisniku omogućeno da kupi i preuzme (engl. *download*) pjesmu preko interneta. (An Industrial-Strength Audio Search Algorithm https://www.ee.columbia.edu/~dpwe/papers/Wang03-shazam.pdf lipanj 2019., 2019.)

Aplikacija za mobilne uređaje se pojavljuje 2008. godine i bila je dostupna samo za Apple iPhone pametne telefone (engl. *smartphone*). Godinu dana kasnije postaje dostupna i na Android (Slika 2.2 Prikaz „Shazam“ aplikacije na Android operacijskom sustavu) platformi, a nakon toga 2010. godine, pojavljuje se i na Windows operacijskom sustavu (engl. *operating system*). Tvrtku je u 2018. godini preuzeo Apple. Shazam je danas dostupan na svim najpoznatijim platformama. Aplikacije su preuzete preko milijardu puta i uslugu koristi preko 150 milijuna korisnika mjesečno. (Shazam https://medium.com/swlh/the-story-of-shazam-the-startup-days-6bccebd17d84 lipanj 2019.)

Slika 2.2 Prikaz „Shazam“ aplikacije na Android operacijskom sustavu

SoundHound je tvrtka koja razvija istoimenu aplikaciju za identifikaciju audio zapisa i govora. Tvrtka razvija tehnologije za prepoznavanje glasa, NLU, odnosno razumijevanje prirodnog govora (engl. *natural language understanding*), prepoznavanje zvuka i

tehnologije za pretraživanje podataka. Osim SoundHound aplikacije za prepoznavanje glazbe, tvrtka razvija i AI platformu Houndify koja se temelji na glasovnoj umjetnoj inteligenciji (engl. *artificial intelligence*). Tvrtka se također bavi razvojem i glasovno pogonjenog digitalnog asistenta (engl. *voice-enabled digital assistant*), naziva Hound. Zadnjih nekoliko godina SoundHound najviše surađuje s tvrtkama iz auto industrije. Prema tome, njihove tehnologije najčešće se pronalaze u novijim automobilima koji nude mogućnosti glasovnog upravljanja. (SoundHound https://www.wired.com/2015/06/soundhound-hound/ lipanj 2019.)

Chromaprint algoritam za generiranje audio potpisa je ključni dio AcoustiD web servisa (engl. *web service*) za identifikaciju cjelovitih glazbenih datoteka. AcoustiD je servis otvorenog koda (engl. *open source*). Chromaprint algoritam je dostupan kao klijentska biblioteka za programske jezike C i Python. Za razliku od ostalih prethodno navedenih servisa, Chromaprint može prepoznati samo cjelovite glazbene datoteke. Korisnici mogu poslati generirane audio potpise na servis, čime se omogućuje svim korisnicima servisa dohvaćanje metapodataka o pripadajućem audio zapisu. Metapodaci pojedinog zapisa se također mogu dohvaćati slanjem identifikacijskog koda na servis. Usluga je potpuno besplatna i u bazi sadrži preko 34 milijuna potpisa. (Chromaprint https://oxygene.sk/2011/01/how-does-chromaprint-work/ lipanj 2019.)

ACRCloud je web platforma za automatsko prepoznavanje sadržaja i bazirana je na algoritmima za generiranje audio potpisa. Omogućuje korisnicima da učitaju (engl. *upload*) svoj sadržaj na platformu koja će generirati audio potpise. Nakon učitavanja sadržaja na platformu, korisnici mogu programski poslati kratke isječke zvuka i dobiti odgovor s metapodacima za traženi zapis. Platforma ima mogućnost praćenja sadržaja koji se emitira uživo što omogućuje jednostavno prepoznavanje sadržaja, mjerenje koliko puta se neki sadržaj reproducirao i popratno tome, zaštitu autorskih prava. ACRCloud baza podataka sadrži preko 30 tisuća indeksiranih radio stanica i preko 68 milijuna audio zapisa za koje su generirani audio potpisi. Indeksirane radio stanice omogućuju korisnicima praćenje sadržaja koji se reproducira uživo bez povezivanja na tok podataka (engl. *data stream*) same postaje. (ACRCloud
https://docs.acrcloud.com/docs/acrcloud/ lipanj 2019.)

3. Analiza zvuka

Zvuk je val koji se širi u zraku, vodi ili u nekom drugom sredstvu. Uzrok nastanka zvučnih valova su vibracije, valovi se zatim šire periodičnim titranjem čestica sredstva kroz koji putuju. ([12], 2019.)

Pojava zvuka detektira se osjetilom sluha, a definirana je frekvencijom i valnom duljinom. Frekvencija zvuka se definira kao broj titraja izvora zvuka u jednoj sekundi, a mjerna jedinica joj je Herc (Hz). S druge strane, valna duljina zvuka definira se kao razmak između dva susjedna najveća zgušnjenja, i kao razmak između dva susjedna razrjeđenja, sredstva kroz koju se val širi. Normalno ljudsko uho može čuti zvukove u rasponu frekvencija od 20 Hz do 20 000 Hz, taj frekvencijski raspon se smanjuje starenjem. Zvukovi ispod frekvencije od 16 Hz nazivaju se infrazvukovima ili podzvukovima, a zvukovi iznad frekvencije od 20 000 Hz nazivaju se ultrazvukovima ili nadzvukovima. (Zvuk https://hr.wikipedia.org/wiki/Zvuk 2019.)

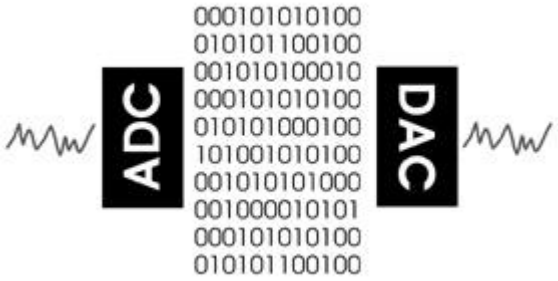
Glazbenici, kada proizvedu glazbu, reprezentirana je u analognom formatu. Da bi računalo, pametni telefon, mp3 *player* ili cd *player* imali mogućnost pohrane i reprodukcije glazbe, ona prvo treba biti pretvorena u digitalni format.

Pretvorbu iz analognog signala u digitalni signal obavlja poseban elektronički sklop ADC[1]. Sklop omogućuje jako brzo uzimanje uzoraka (engl. *sampling*) iz analognog signala. U tom procesu uzorak kontinuiranog signala se pohranjuje kao numerička vrijednost (Slika 3.1 Slikovita reprezentacija snimanja i reprodukcije zvuka kroz ADC/DAC). Proces uzimanja uzorka iz analognog signala biti će opisan u idućem poglavlju (**Error! Reference source not found.**).

Za obrnuti proces, odnosno pretvorbu digitalnog signala u analogni, koristi se poseban elektronički sklop DAC[2]. U tom obrnutom procesu numeričke vrijednosti se interpoliraju i ponovno pretvaraju u kontinuirani analogni signal koji mogu interpretirati uređaji poput zvučnika (Slika 3.1 Slikovita reprezentacija snimanja i reprodukcije zvuka kroz ADC/DAC). U prijevodu, to znači da je lista diskretnih vrijednosti ponovno pretvorena u kontinuiranu funkciju. Spomenuti proces interpolacije označava glatki prijelaz (engl. *smooth transition*)

[1] ADC – sklop za pretvaranje analognog signala u digitalni
[2] DAC – sklop za pretvaranje digitalnog signala u analogni

po numeričkim vrijednostima koje se koriste za pretvaranje. (Music and Computers, theoretical and historical approach http://sites.music.columbia.edu/cmc/MusicAndComputers/ lipanj 2019., 2019.)



[3]

Slika 3.1 Slikovita reprezentacija snimanja i reprodukcije zvuka kroz ADC/DAC

U nastavku će se opisati proces pretvorbe analognog signala u digitalni signal koji je pogodan za manipulaciju i detaljniju analizu.

3.1. Uzimanje uzoraka

Iz prethodnog poglavlja je poznato da za reprezentaciju signala u digitalnom obliku potrebno je uzeti uzorke iz analognog signala. Proces uzimanja uzoraka naziva se *audio sampling*.

Analogni signali su kontinuirani signali, to znači da ako se uzme jednu sekunda analognog signala, moguće ju je podijeliti na beskonačno mnogo dijelova. Naravno, u digitalnom svijetu ograničenje predstavlja količina diskovnog prostora. Prema tome se može zaključiti da uzimanje svakog uzorka (engl. *sample*) iz kontinuiranog signala nije moguće.

Koliko puta je potrebno uzeti uzorak analognog signala da bi se dobila kvalitetna i potpuna digitalna reprezentacija?

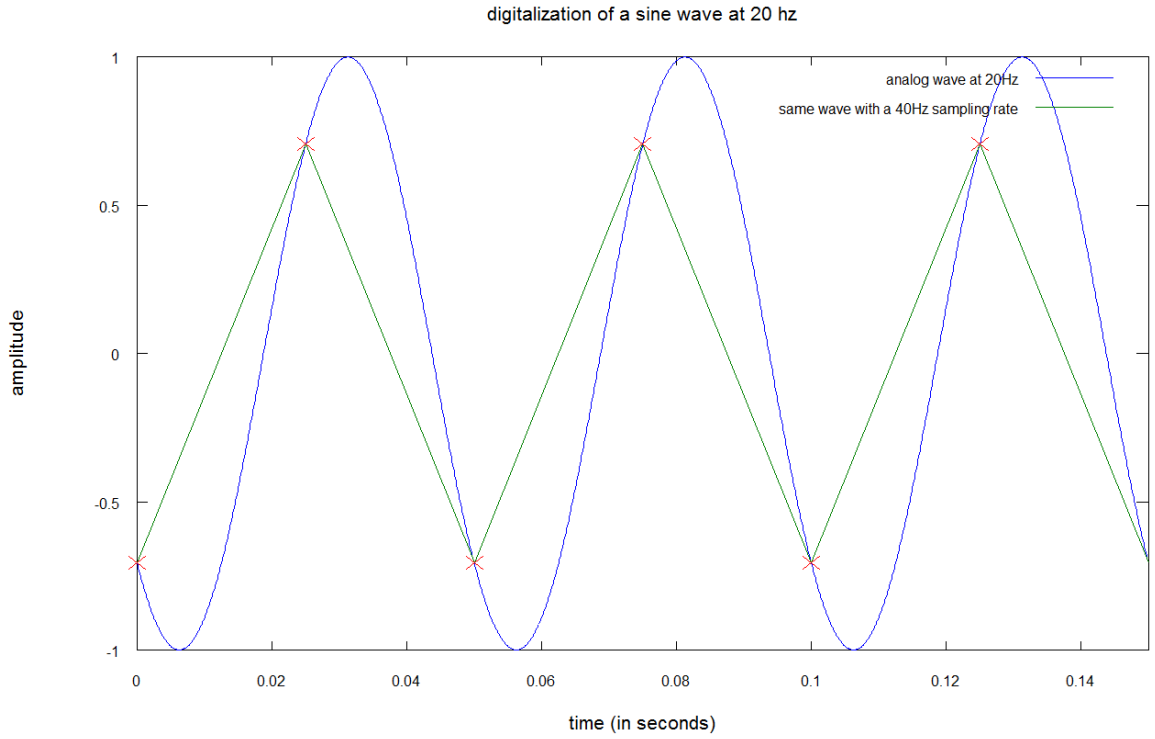
Odgovor na to pitanje pruža Nyquist–Shannon teorem uzimanja uzoraka (engl. *Nyquist–Shannon sampling theorem*).

Nyquist–Shannon teorem navodi da frekvencija uzimanja uzoraka mora biti najmanje dva puta veća od najviše frekvencije zvuka u analognom signalu.

Standardna jedinica uzimanja uzoraka za digitalni zvuk je 44 100 uzoraka (engl. *samples*) po sekundi, odnosno 44.1 kHz. Na početku 3. poglavlja spomenuto je da ljudsko uho čuje zvukove od 20 Hz do 20 000 Hz, dakle, prema Nyquist–Shannon teoremu, da bi digitalizacija signala bila uspješna i kvalitetno reprezentirana uzorci se moraju uzimati barem 40 000 puta po sekundi.

[3] Slika 3.1 (http://sites.music.columbia.edu/cmc/MusicAndComputers/chapter2/02_01.php)

Ideja teorema je da se dva puta po ciklusu sinusoidnog signala na nekoj frekvenciji f mora uzeti uzorak. Ako je frekvencija uzimanja uzoraka barem dva puta veća od frekvencije signala zvuka, uzorci bi trebali izgledati kao na Slika 3.2.

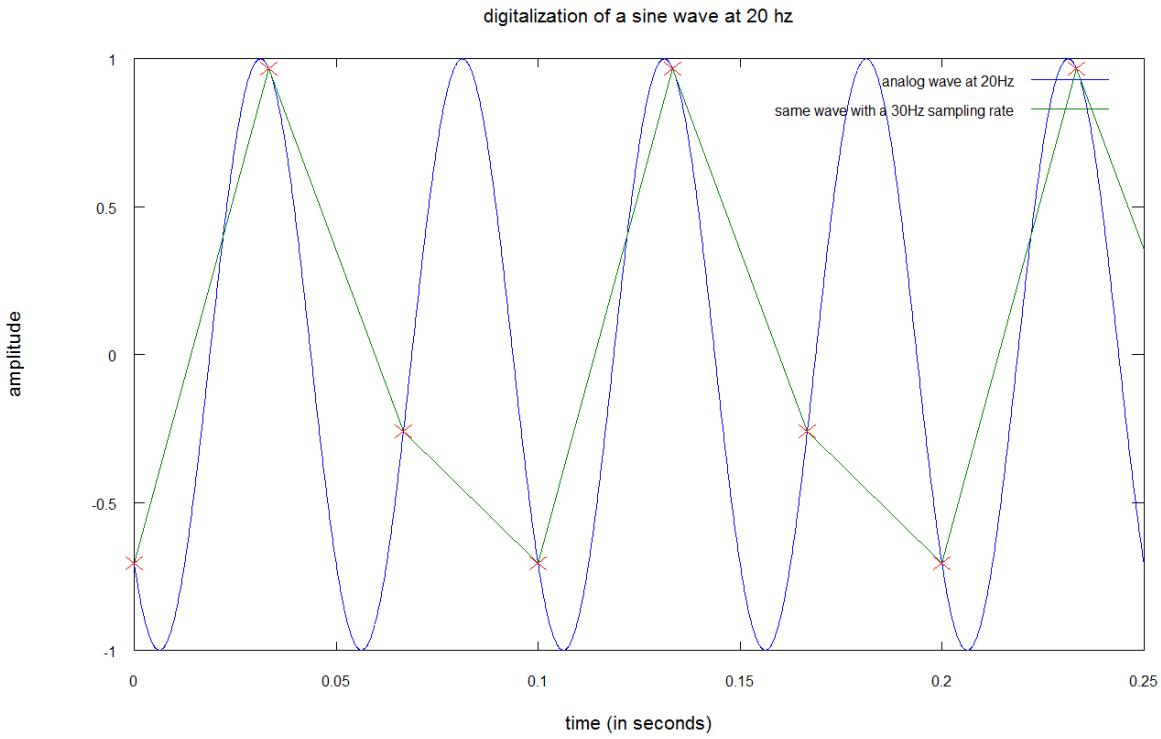


[4]

Slika 3.2 Pravilno uzimanje uzoraka

U obrnutom slučaju, ako je frekvencija uzimanja uzoraka premala, neće biti dovoljno informacija za konstrukciju cjelovitog signala u digitalnom obliku, kao što je prikazano na Slika 3.3. Ovaj slučaj je poznat kao *undersampling*.

[4] Slika 3.2 preuzeta je s web stranice (http://coding-geek.com/how-shazam-works/#Sampling)



[5]

Slika 3.3 Nepravilno uzimanje uzoraka (*undersampling*)

3.2. Kvantizacija

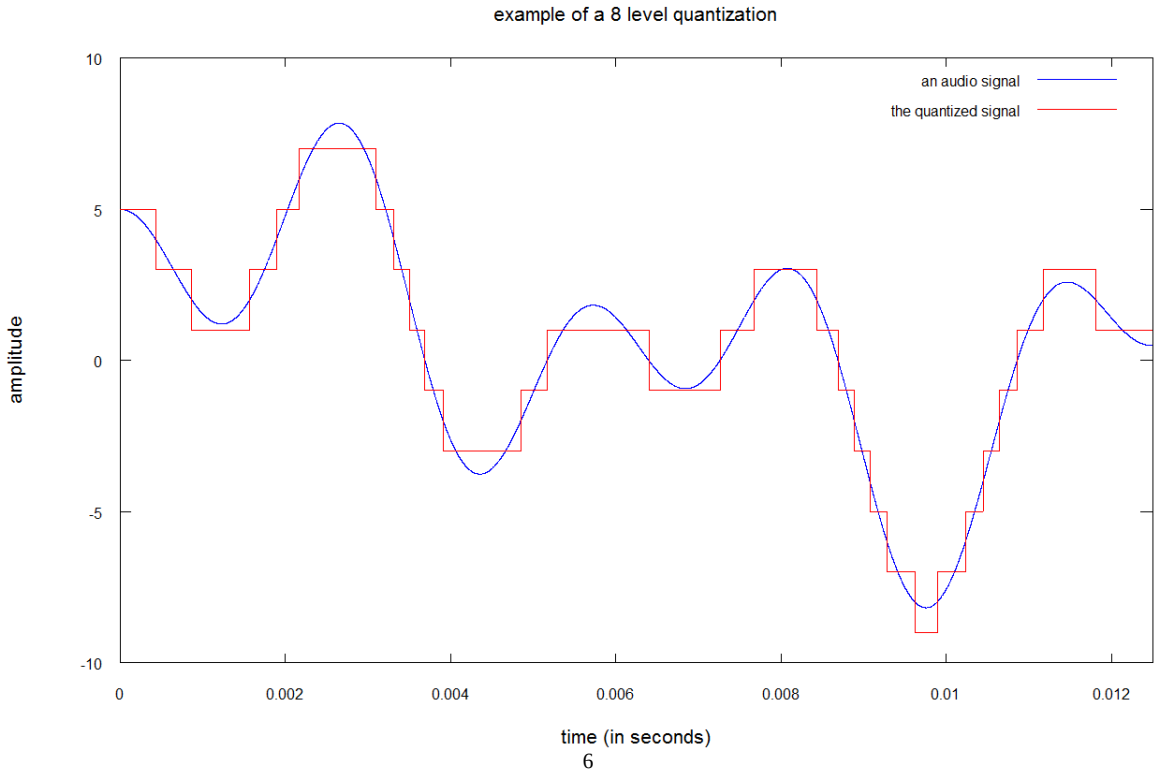
U prethodnom poglavlju opisan je proces digitalizacije signala kroz postupak uzimanja uzoraka, ali nije spomenuto na koji način se digitalizira glasnoća iz analognog signala.

Problem je isti kao i kod uzimanja uzoraka, potrebno je iz kontinuiranog signala s beskonačno mnogo kombinacija vrijednosti glasnoća napraviti konverziju u diskretnu listu vrijednosti. Opisani problem se rješava procesom kvantizacije.

Kvantizacija signala se postiže na način da se amplituda uzorka kontinuiranog signala mapira na n-bitova. Raspon amplitude je podijeljen u 2^n diskretnih nivoa.

Uređaj ili algoritamska funkcija koja odrađuje proces kvantizacije, naziva se kvantizator. Primjer takvog uređaja je ADC elektronički sklop spomenut na početku 3. poglavlja.

[5] Slika 3.3 preuzeta je s web stranice (http://coding-geek.com/how-shazam-works/#Sampling)



Slika 3.4 Niska kvantizacija audio signala

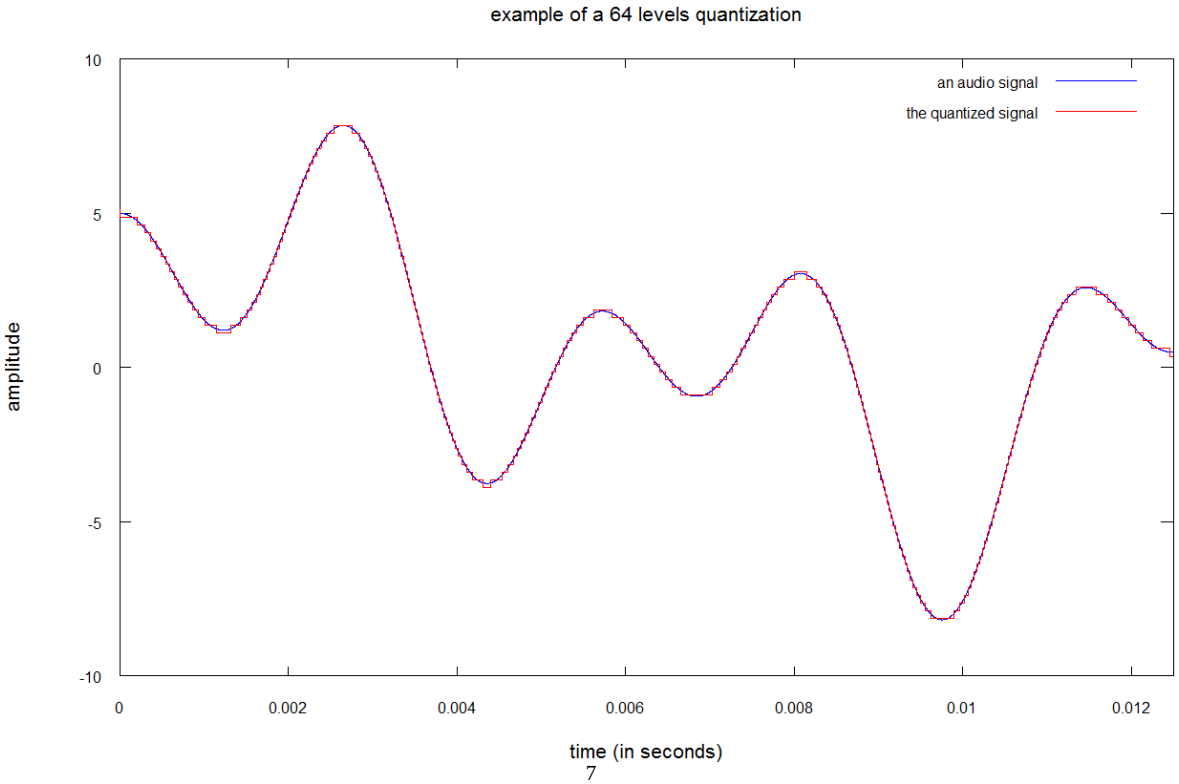
Slika 3.4 prikazuje 8 razina kvantizacije signala, odnosno kvantizaciju s 3 bita jer da bi se postiglo 8 razina kvantizacije potrebna su 3 bita (2^3). Na slici se može vidjeti da je rezultat kvantizacije (signal u crvenoj boji) jako isprekidan i da nije sličan originalnom signalu.

Razlika između ulaznog signala i kvantizirane vrijednosti naziva se kvantizacijska greška ili kvantizacijski šum.

Slika 3.5 prikazuje signal s 64 razine kvantizacije, odnosno 6 bita (2^6). Iako je signal na slici još uvijek isprekidan, izgleda, ali i zvuči puno sličnije originalnom signalu.

Kod digitalizacije signala najčešće se koristi 16 bitova za kvantizaciju, odnosno 65 536 razina. Korištenjem 16 bitova, kvantizacijski šum je dovoljno nizak za ljudske uši. Za profesionalnu obradu glazbe koriste se 24 bita za kvantizaciju originalnog signala. (How does Shazam work http://coding-geek.com/how-shazam-works/ lipanj 2019.)

[6] Slika 3.4 preuzeta je web stranice (http://coding-geek.com/how-shazam-works/#Quantization)



[7]

Slika 3.5 Kvantizacija signala s 64 razine, odnosno 6 bita

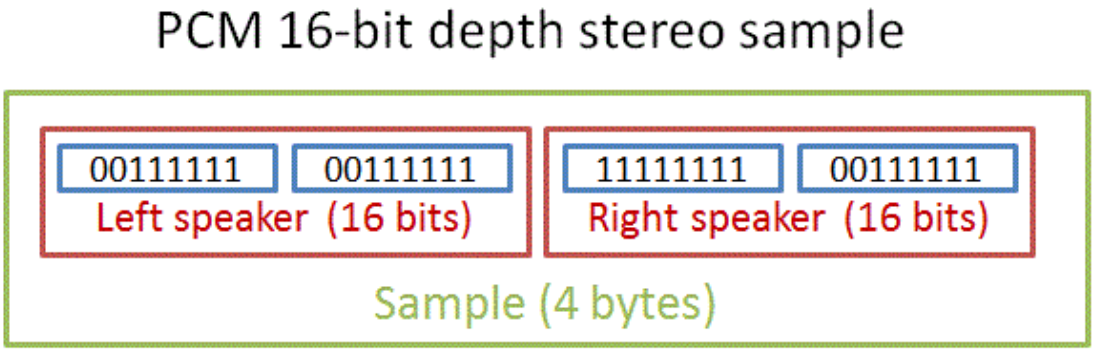
3.3. Pulse Coded Modulation

Pulse Coded Modulation ili PCM je standard koji se koristi za reprezentaciju digitalnih signala dobivenih procesom uzimanja uzoraka. Koristi se u gotovo svim uređajima za reprezentaciju digitalnog signala; od računala, CD-a, digitalne telefonije i drugih audio primjena.

PCM je niz pravilno posloženih bitova. Može sadržavati više kanala, primjer tome je stereo audio zapis koji je sastavljen od dva kanala. PCM niz je podijeljen na uzorke, a broj uzoraka po sekundi je jednak broju uzoraka po sekundi uzetim iz originalnog analognog signala. Prema tome, ako je frekvencija uzimanja uzoraka iz analognog signala 44.1 kHz, PCM niz će sadržavati 44 100 uzoraka po sekundi.

Postoji više vrsta PCM formata, a najkorišteniji je LPCM (engl. *Linear Pulse Coded Modulation*) s frekvencijom od 44.1 kHz, 16 bitova i s dva kanala (Slika 3.6).

[7] Slika 3.5 preuzeta je web stranice (http://coding-geek.com/how-shazam-works/#Quantization)



[8]

Slika 3.6 Jedan uzorak iz PCM niza

Slika 3.6 prikazuje jedan uzorak PCM niza s ukupno 44 000 uzoraka za svaku sekundu audio sadržaja. Svaki uzorak ima 4 bajta (engl. *byte*), od toga po 2 bajta za svaki kanal (lijevi i desni zvučnik).

[8] Slika 3.6 preuzeta je s web stranice (http://coding-geek.com/how-shazam-works/#Pulse_Coded_Modulation)

4. Generiranje audio potpisa

U 3. poglavlju opisan je proces pretvorbe analognog signala u digitalni signal. Pretvorba omogućuje reprodukciju, manipulaciju i analizu signala na računalu. Za uspješno generiranje audio potpisa potrebno je izvući frekvencijske karakteristike analiziranog digitalnog signala.

Fourierova transformacija je metoda prebacivanja funkcije iz vremenske domene u funkciju frekvencijske domene. Najčešće je izražena na sljedeći način:

$$F(s) = \int_{-\infty}^{\infty} f(x) e^{-2\pi s x i}\, dx$$

Obrnuti postupak, odnosno prebacivanje funkcije iz frekvencijske domene u vremensku domenu izražava se na sljedeći način:

$$f(x) = \int_{-\infty}^{\infty} F(s) e^{2\pi i s x}\, ds$$

Prethodno definirane funkcije rade dobro za transformaciju kontinuiranih signala, ali kao što je opisano u prethodnim poglavljima, u digitalnom okruženju koriste se diskretne vrijednosti. (Fourierov red i Fourierova transformacija https://hrcak.srce.hr/file/121546 lipanj 2016. 2019.)

Za transformaciju funkcija iz jedne domene u drugu s diskretnim ulaznim vrijednostima koristi se diskretna Fourierova transformacija.

Diskretna Fourierova transformacija za prebacivanje funkcije iz vremenske domene u frekvencijsku domenu definira se na sljedeći način:

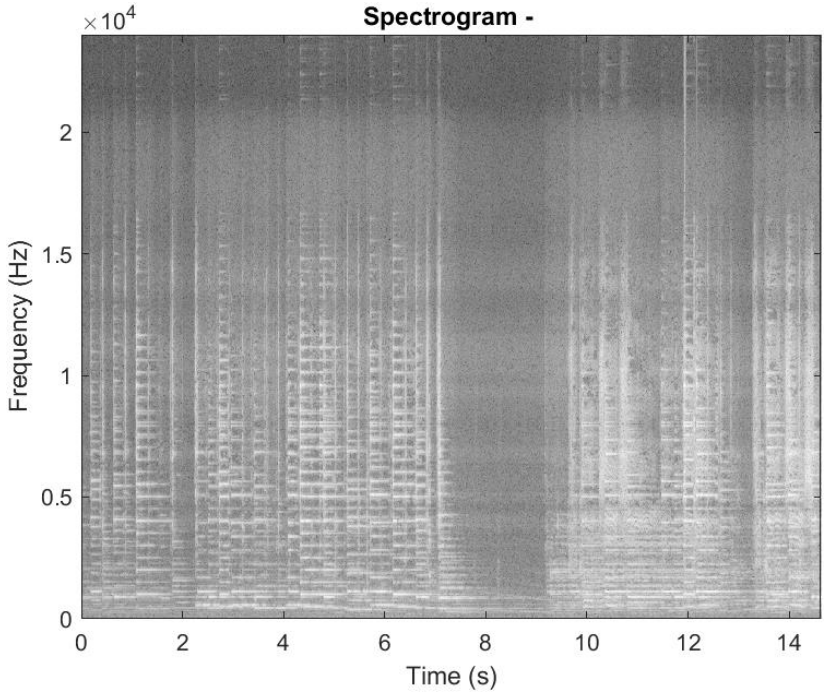
$$X_k = \sum_{n=0}^{N-1} x_n\, e^{-\frac{2\pi k n i}{N}}$$

Obrnuti postupak, odnosno inverzna diskretna Fourierova transformacija za prebacivanje funkcije iz frekvencijske domene u vremensku domenu izražava se na sljedeći način (Diskretna Fourierova transformacija http://www.mathos.unios.hr/~scitowsk/MP/17-Mihaljcic-Petrijevcanin-2.pdf, 2019.):

$$x_n = \frac{1}{N} \sum_{n=0}^{N-1} X_k\, e^{\frac{2\pi k n i}{N}}$$

Za generiranje audio potpisa koristi se diskretna Fourierova transformacija. Omogućuje analizu digitalno pohranjenog signala. Korištenjem transformacije moguće je analizirati magnitude frekvencija na određenim vremenskim intervalima. Postupak analize opisati će se u sljedećem poglavlju.

4.1. Princip rada algoritma



Slika 4.1 Spektogram uzorka snimljenog mikrofonom

Slika 4.1 prikazuje spektogram zvuka snimljenog mikrofonom. Spektogram predstavlja sliku zvuka, a izračunava se pomoću diskretne Fourierove transformacije. Prikazuje frekvencije od kojih se zvuk sastoji, od niskih do visokih i njihovu promjenu kroz vrijeme od lijeva na desno. Na slici se također vidi da u različitim trenutcima u vremenu postoje frekvencije veće od drugih frekvencija.

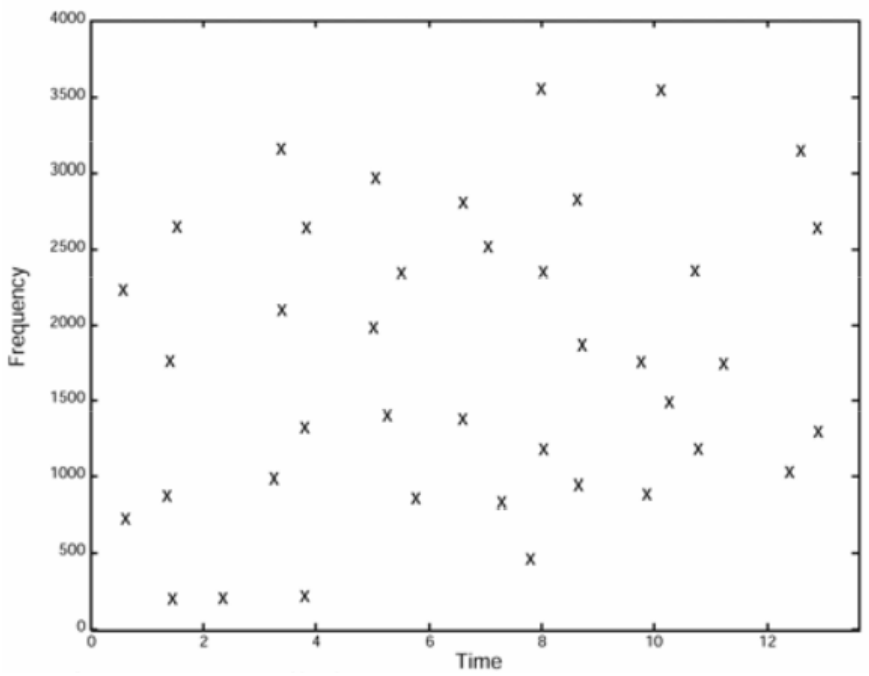
Nakon što je dobivena frekvencijska domena signala, potrebno je pronaći lokalne maksimume u amplitudi u određenom frekvencijskom rasponu. Kao optimalni frekvencijski rasponi pokazali su se sljedeći parovi.

- 40 – 80 Hz
- 80 – 120 Hz
- 120 – 180 Hz
- 180 – 300 Hz

Ove frekvencije su odabrane zato što se najviše aktivnosti u pjesmama događa u tom rasponu.

Maksimumi su definirani kao parovi vremena i frekvencije, a dobivaju se iz prethodnog spektograma. Slika 4.2 prikazuje filtrirani spektogram zvuka gdje su označene samo najsnažnije frekvencije.

Nakon što su dobivene ključne frekvencije za svaki frekvencijski raspon potrebno je upotrijebiti algoritam sažimanja nad cijelim skupom ključnih frekvencija. Rezultat sažimanja se sprema u *hash* tablicu zajedno s identifikatorom audio zapisa i pripadajućim vremenom zapisa koji se analizira. Vrijeme zapisa je važno za pretraživanje podudarajućih audio zapisa (4.1.1). Opisani proces se ponavlja za cijelu dužinu audio zapisa. Na kraju se reprezentacija *hash* tablice sprema u bazu podataka.



[9]

Slika 4.2 Filtrirani spektogram zvuka

4.1.1. Pretraživanje baze

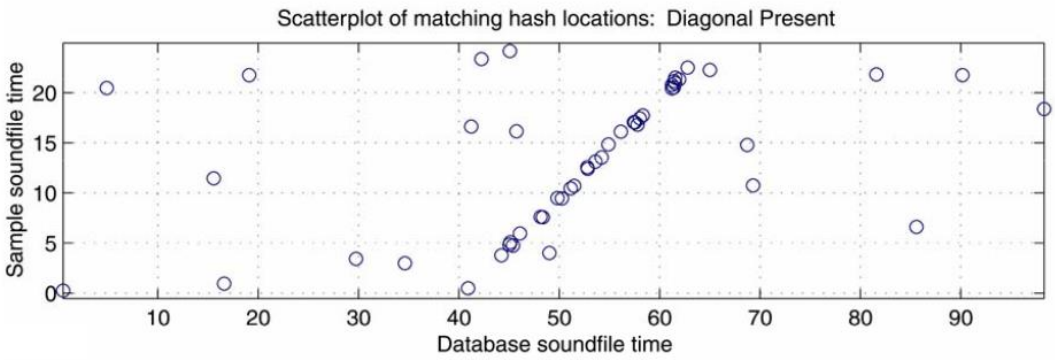
Proces pretraživanja započinje uzimanjem kratkog uzorka nekog zapisa koji je prethodno indeksiran u bazi. Uzorak može biti snimljen ili isječak iz originalnog audio zapisa. Svaki

[9] Slika 4.2 preuzeta je s web stranice (http://coding-geek.com/how-shazam-works/#Spectrogram_filtering)

uzorak prolazi isti proces generiranja audio potpisa kao i cjelovita audio datoteka kod inicijalnog indeksiranja. Nakon što uzorak prođe proces opisan na početku poglavlja (4), rezultat je također *hash* tablica.

Traženje započinje na način da se za svaku *hash* vrijednost dobivenu iz uzorka dohvaća lista s informacijama o vremenima u originalnim zapisima. Treba imati na umu da se vremena originalnih zapisa neće poklapati s vremenima iz uzorka jer kod uzimanja uzoraka zabilježena je apsolutna vremenska vrijednost od početka zapisa. Kratki uzorak može biti iz bilo kojeg dijela originalnog zapisa i sadržava relativne vremenske vrijednosti. Prema tome, može se izračunati razlika između te dvije vrijednosti i rezultat spremiti zajedno s identifikatorom audio zapisa.

Postupak se ponavlja za svaku vrijednost iz *hash* tablice kratkog uzorka. Na kraju se pregledaju rezultati i za najbolju predikciju se uzima identifikator zapisa kod kojega se razlika apsolutnog i relativnog vremena najviše puta ponavlja. Slika 4.3 prikazuje poklapanje kratkog uzorka s zapisom iz baze. Na slici se može vidjeti da se uzorak počinje poklapati tek od sredine originalnog zapisa.



[10]

Slika 4.3 Poklapanje zapisa iz baze s kratkim uzorkom

[10] Slika 4.3 preuzeta je s web stranice (https://www.ee.columbia.edu/~dpwe/papers/Wang03-shazam.pdf)

5. Arhitektura i tehnologije

Sljedeća poglavalja obrađuju arhitekturu i tehnologije korištene na *backend* i *frontend* modulima aplikacije. Opisati će se arhitekturalni modeli pojedinog modula, te proces interakcije s različitim dijelovima sustava.

5.1. Backend modul

Poslužiteljski dio sustava sastavljen je od PostgreSQL baze podataka koje komunicira s Phoenix programskim okvirom (engl. *framework*) napisanim u Elixir programskom jeziku. U ovom poglavlju opisati će se značajke PostgreSQL baze podataka, Elixir programskog jezika i funkcionalnosti Phoenix programskog okvira.

5.1.1. Baza podataka

Aplikacija koristi PostgreSQL bazu podataka. PostgreSQL je najnaprednija relacijska baza podataka (RDBMS[11]) otvorenog koda dizajnirana s naglaskom na portabilnost. Može se koristiti na MacOS, Windows i UNIX[12] operacijskim sustavima i potpuno je besplatna.

PostgreSQL pruža korisnicima mnogo naprednih mogućnosti koje se inače pronalaze samo u komercijalnim bazama podataka koje nisu besplatne. (PostgreSQL https://www.ionos.com/digitalguide/server/know-how/postgresql/, 2019.)

Neke od glavnih značajki PostgreSQL baze podataka su:

- korisnički definirani tipovi
- nasljeđivanje tablica
- sofisticirani sustav zaključavanja
- referencijalni integritet stranog ključa
- pogledi, korisnički definirana pravila, podupiti
- ugniježdene transakcije
- više-verzijska kontrola u više-korisničkom okruženju (MVCC[13])
- asinkrona replikacija

[11] RDBMS - relational database management system
[12] UNIX - vrsta operacijskih sustava
[13] MVCC - *multi-version concurrency control*

- integracija s Microsoft Windows Server operacijskim sustavom
- korisnički definirana reprezentacija same baze podataka
- oporavak u točci vremena (engl. *point-in-time recovery*)

Jedan od razloga za korištenje PostgreSQL baze podatka je jednostavna integracija s Phoenix programskim okvirom koji je opisan u poglavlju 03.

5.1.2. Elixir

Elixir je programski jezik koji se pokreće nad BEAM[14] Erlang virtualnom okolinom. Erlang razvojna platforma omogućuje izradu skalabilnih i distribuiranih sustava koji pružaju visoku dostupnost bez ispada sustava (engl. *system downtime*). Platforma je originalno razvijena 1986. godine za telekom sustave kako bi se zadovoljila potreba za visokom dostupnošću. Visoka dostupnost sustava na Erlang platformi omogućena je kroz sljedeće značajke:

- tolerancija na kvarove
- skalabilnost
- distribucija
- vrijeme odaziva
- ažuriranje produkcijskog sustava (engl. *live update*)

Tolerancija na kvarove označava sposobnost sustava da nastavi s radom čak i ako se dogodi neka nepredvidiva greška u kodu ili nekom od vanjskih sustava s kojima se komunicira.

Skalabilnost sustava označava jednostavnu mogućnost dodavanja više hardverskih resursa kao odgovor na povećan broj korisničkih zahtjeva. Primjer tome može biti objava neke nove aplikacije i velika zainteresiranost korisnika, kako se broj korisnika povećava, odaziv sustava se smanjuje. Skaliranje treba provesti bez zaustavljanja već pokrenutog servisa kako se ne bi narušilo korisničko iskustvo (engl. *user experience*).

Da bi se napravio sustav koji nikada ne prestaje raditi, potrebno ga je distribuirati na više strojeva. Pristup distribuciji sustava na više strojeva pruža cjelokupnu stabilnost sustava, jer ako jedan stroj na kojem se vrti neka usluga iz nekog razloga prestane raditi, njegovu ulogu može preuzeti drugi stroj.

[14] BEAM je virtualna okolina nad kojom se pokreće Erlang i Elixir izvorni kod

Vrijeme odaziva sustava uvijek treba biti nisko kako se ne bi narušilo korisničko iskustvo. Odaziv sustava treba ostati što manji i u slučaju velikog broja zahtjeva. Treba imati na umu da zahtjevnije kalkulacije ne bi smjele blokirati sustav kao cjelinu niti daljnju korisničku interakciju prema sustavu.

Jedna od čestih situacija sustava u produkcijskom okruženju je ispravljanje grešaka (engl. *bugs*) ili dodavanje novih funkcionalnosti. Nadogradnja na novu verziju sustava ne bi smjela zahtijevati ponovno pokretanje sustava. (S. Jurić, Elixir *in action*, Manning, 2019.)

Elixir je funkcijski (engl. *functional*) i konkurentni (engl. *concurrent*) programski jezik otvorenog koda koji omogućuje izradu brzih, distribuiranih i pouzdanih sustava s naglaskom na skalabilnost i održivost aplikacija i sustava. Kod napisan u Elixiru pretvara se u strojni kod podržan od strane BEAM virtualne okoline (engl. *virtual machine*). Ta činjenica omogućuje korištenje Erlang biblioteka (engl. *library*) u Elixir kodu i obrnuto.

Elixir je semantički sličan Erlang kodu, ali ga unaprjeđuje na dosta načina kako bi bio pristupačniji programerima. Neka od ključnih unaprjeđenja su:

- manje repetitivnog koda i duplikacije
- dodaje poboljšanja sintakse (engl. *syntactic sugar*)
- pruža jedinstveni alat za upravljanje paketima

Jedno od ključnih unaprjeđenja Elixira je smanjivanje repetitivnog koda i duplikacije što na kraju rezultira jednostavnijim, čitljivijim i lakše održivim kodom.

Primjer Elixir koda:

```
defmodule Calc do
  def fib(0), do: 0
  def fib(1), do: 1
  def fib(n), do: fib(n-2) + fib(n-1)
end
```

Kôd 5.1 Elixir modul za računanje Fibonaccijevog broja

Kôd 5.1 prikazuje modul za računanje Fibonaccijevog broja. Svaka funkcija je definirana za točno određeni ulazni podatak, s određenim povratnim izrazom.

Elixir sintaksa također omogućuje pisanje *macro* funkcija. *Macro* je kod koji se izvrši kod kompilacije programa i zapravo omogućuje svojevrsno definiranje ključnih riječi u kodu. *Macro* funkcije omogućuju jednostavan način proširivanja jezika na način da novo kreirane ključne riječi izgledaju kao dio samog jezika.

Erlang i Elixir su funkcijski programski jezici s nepromjenjivim varijablama (engl. *immutable variables*) i oslanjaju se isključivo na funkcije za transformaciju podataka. Prednost tog pristupa je automatsko odvajanje kod u male, ponovno iskoristive funkcije. Jezičnim ograničenjima poput nepromjenjivih varijabli i jednostavnom mogućnošću za kompoziciju funkcija automatski se zadovoljava dio principa konkurentnosti čistog koda. ([2])

5.1.3. Poslužiteljski dio sustava

Na poslužiteljskoj strani korišten je Phoenix programski okvir koji je napisan u Elixir programskom jeziku. Olakšava izradu distribuiranih sustava s malim vremenom odaziva i tolerancijom na kvarove. Phoenix se često koristi za izradu MVC aplikacija, ali za izradu aplikacije čija je implementacija opisana u 6. poglavlju, Phoenix programski okvir koristi se za izradu REST API servisa.

Razlog odabira REST arhitekture za izradu servisa je interoperabilnost sustava. REST se temelji na jednostavnim standardiziranim elementima i omogućuje CRUD operacije nad resursima pomoću URI-ja kroz HTTP protokol. CRUD definira kreiranje, dohvaćanje, ažuriranje i brisanje resursa. Komunikacija sa servisom putem HTTP protokola kroz JavaScript[15] klijentsku aplikaciju će biti opisana u 6. poglavlju.

Također, zbog odabira REST arhitekture, u budućnosti se može jednostavno napisati aplikacija u bilo kojoj tehnologiji pod uvjetom da zna komunicirati s HTTP protokolom.

Glavne značajke Phoenix programskog okvira:

- podržava velik broj istovremenih konekcija
- *Plug* specifikacija
- *Ecto* alat za podatkovno mapiranje
- kanali za komunikaciju u realnom vremenu

Phoenix podržava jako velik broj istovremenih konekcija i dvosmjernu komunikaciju u realnom vremenu bez dodatnih biblioteka.

Phoenix u svoj tijek konekcije (engl. *lifecycle*) uvodi specifikaciju pod nazivom *Plug*, koja omogućuje kompoziciju modula između web aplikacija. Sve komponente Phoenixa su

[15] JavaScript je programski jezik za izradu klijentskih web aplikacija

napisane kao *plugovi* što omogućuje jednostavno ubacivanje vanjskih ili vlastitih *plugova* u tijek konekcije. Primjer *plug* funkcionalnosti je provjera da li konekcija na server sadržava token, odnosno da li je korisnik autenticiran. (Phoenix https://hexdocs.pm/phoenix/up_and_running.html srpanj 2019.)

Ecto alat za komunikaciju s bazom sastoji se od četiri modula ([20].):

- repozitorij
- shema
- validacija podataka
- upiti

Ecto repozitorij modul se koristi kao omotač (engl. *wrapper*) za druge module koji će komunicirati s izvorom podataka (engl. *data source*). Svaki repozitorij sadrži generičke CRUD operacije za manipulaciju s entitetima iz baze.

Ecto shema modul se koristi za mapiranje entiteta iz baze u Elixir strukturu. Najčešće se koristi za mapiranje tablice iz baze u tip podataka podržan od strane Elixira.

Ecto modul za validaciju podataka omogućuje programerima jednostavan način za filtriranje ulaznih podataka i prebacivanje vanjskih parametara u pravilne Elixir tipove podatka. Također pruža mogućnost validacije promjena prije spremanja u bazu, kako korisnik ne bi poslao podatke u nepravilnom formatu.

Ecto modul za pisanje upita omogućuje pisanje kompleksnih upita za dohvaćanje podatka pomoću Elixir sintakse. Baziran je na ranije opisanim *macro* funkcijama i zapravo predstavlja DSL[16]. Modul također sadrži sigurnosne značajke i automatski izbjegava česte sigurnosne propuste kao što je SQL *injection*.

Kanali za komunikaciju u realnom vremenu (engl. *real-time communication*) su jedna od najzanimljivijih funkcionalnosti Phoenix programskog okvira. Omogućuju komunikaciju s više milijuna spojenih korisnika. Primjena za komunikacija u realnom vremenu ima mnogo, a neki od najčešćih slučajeva korištenja uključuju:

- chat aplikacije
- obavijesti u aplikacijama
- događaji u online igrama
- dobivanje senzorskih informacija

Implementacija kanala pomoću Phoenix programskog okvira je jednostavna, klijent se spoji na servis i pretplati se (engl. *subscribe*) na neku temu. Kada se dogodi neka promjena na serveru ili klijentu, svi pretplatnici su odmah obaviješteni.

[16] DSL predstavlja specijalizirani jezik za određenu domenu, primjer je SQL

Kanali u Phoenixu podržavaju sve vrte klijenata, od najobičnijeg internet pretraživača, mobilnog uređaja pa sve do pametnih satova ili ugradbenih uređaja (engl. *embedded device*). Kanali u pozadini koriste websocket protokol, ukoliko ga klijent koji se spaja na servis podržava, u slučaju da klijent ne podržava navedeni protokol konekcija se spusti razinu niže na periodičko provjeravanje informacija HTTP protokolom.

5.2. Frontend modul

Klijentski dio sustava je SPA aplikacija izrađena pomoću React JavaScript radnog okvira i nekoliko drugih pomoćnih biblioteka koje će biti opisane u ovom poglavlju.

Single-page application je tip web aplikacije koja se pokreće u internet pretraživaču. Sastoji se od jedne web stranice koja ne zahtijeva ponovno učitavanje (engl. *page refresh*) tijekom korištenja. Sve promjene na stranici odrađuje JavaScript, ili češće jedna od mnogih JavaScript biblioteka ili radnih okvira. Pokazalo se da SPA aplikacije pružaju puno bolje korisničko iskustvo u odnosu na tradicionalne aplikacije gdje server nakon svake interakcije isporučuje novu stranicu koju zatim pretraživač treba nanovo iscrtati.

5.2.1. React

React je JavaScript biblioteka za izradu korisničkog sučelja. React biblioteku razvija Facebook i *open source* zajednica.

React sam po sebi je zadužen samo za iscrtavanje pogleda, odnosno React biblioteka nije dovoljna za razvoj cijele klijentske web aplikacije. Za izradu potpune web aplikacije, uz React, potrebno je koristiti dodatne biblioteke koje će biti opisane u sljedećim poglavljima. (React
https://reactjs.org/docs/react-api.html srpanj 2019.)

React pruža korisnicima mnogo različitih funkcionalnosti za manipulaciju pogledima (engl. *views*).

Ovdje će se navesti samo nekoliko najvažnijih funkcionalnosti:

- deklarativna izgradnja korisničkog sučelja
- izgradnja sučelja bazirana je na komponentama
- virtualni DOM
- kompatibilnost s raznim platformama

Deklarativan pristup izgradnji korisničkih sučelja omogućuje korisnicima da se fokusiraju na samu aplikaciju i kompoziciju komponenata unutar pogleda, bez trošenja vremena na pisanje koda za ažuriranje pojedinih dijelova korisničkog sučelja.

Baziranje sučelja na malim ponovno iskoristivim komponentama (engl. *reusable components*) omogućuje korisnicima lakšu izgradnju sučelja, ali i znatno pojednostavljuje održavanje elemenata korisničkog sučelja. Svaka komponenta u Reactu ima svoje stanje s kojim upravlja. Stanje komponente u kombinaciji s JavaScript logikom omogućuje izgradnju kompleksnijih elemenata korisničkog sučelja. Sve komponente se mogu deklarativno slagati jedna s drugom, što omogućuje izgradnju velikih i kompleksnih korisničkih sučelja, bez velike kompleksnosti za programera.

Virtualni DOM predstavlja koncept gdje se trenutna, odnosno virtualna reprezentacija korisničkog sučelja drži u memoriji i prema potrebi sinkronizira s pravim DOM-om. Koncept virtualnog DOM-a je temelj deklarativnog pristupa u React aplikacijama, potrebno je samo definirati stanje u kojem bi korisničko sučelje trebalo biti i React će odraditi posao ažuriranja pravog doma s virtualnim. Prije samog ažuriranja pravog DOM-a React napravi usporedbu promijenjenih elemenata korisničkog sučelja kako bi se u pravi DOM zapisale samo promjene. Na taj način se izbjegava korištenje previše skupih operacija za manipulaciju s pravim DOM-om. Ovaj pristup znatno olakšava razvoj klijentskih web aplikacija jer kod tradicionalnog pristupa razvoju JavaScript aplikacija proces ažuriranja DOM-a je potpuno ručni i ovisi o programeru.

React biblioteka je kompatibilna i s mobilnim platformama iOS i Android. Svi principi Reacta s weba prenose se na mobilne uređaje. Biblioteke napisane u JavaScriptu se mogu dijeliti između web aplikacije i mobilnih aplikacija što može uštediti dosta vremena programerima. Sav kod napisan u JavaScriptu se na kraju pretvori u *native* kod za iOS i Android operacijske sustave.

5.2.2. Redux

Redux je *open source* biblioteka koja omogućuje upravljanje cijelim stanjem aplikacije. Najčešće se koristi kao dopuna React aplikacijama, ali je totalno neovisna o bilo kojoj biblioteci za razvoj korisničkog sučelja. Redux je moguće koristi čak i s običnim JavaScript-om bez korištenja radnog okvira. (Redux https://redux.js.org/api/api-reference 2019.)

Najvažnije značajke Redux biblioteke su:

- predvidivost
- centralizacija
- jednostavnost otklanjanja pogrešaka
- fleksibilnost rada s bilo kojim radnim okvirom

Redux omogućuje predvidivost i konzistentnost aplikacije zbog načina na koji se odvija promjena podataka u stanju aplikacije. Stanje je moguće promijeniti samo na određen i konzistentan način, i to na samo jednom mjestu. Tok podataka u aplikacijama koje koriste Redux je jednosmjeran što omogućuje jednostavnije pronalaženje i otklanjanje pogrešaka.

Zbog centralizirane arhitekture Redux biblioteke, moguće je jednostavno implementirati funkcionalnosti poništavanja zadnje akcije (engl. *undo*), vraćanja zadnje akcije (engl. *redo*) i spremanja cijelog stanja u lokalnu pohranu (engl. *local storage*) internet pretraživača ili u bazu podataka.

Stanje aplikacije je moguće promijeniti slanjem jednostavnih objekata koji opisuju koje promjene se trebaju izvršiti nad stanjem aplikacije (Kôd 5.2).

```
{ type: 'ADD_USER', name: 'Marko' }
```

Kôd 5.2 Jednostavan JavaScript objekt koji opisuje promjene stanja

Funkcija koja odrađuje promjenu stanja naziva se *reducer*. Prima dva parametra, trenutno stanje aplikacije i akcijski objekt koji opisuje promjenu stanja. Povratna vrijednost funkcije je novo stanje aplikacije. *Reducer* uvijek vraća novo kreirani objekt stanja kako bi se promijenila referenca objekta u memoriji koja se interno koristi kao znak promjene stanja. Na znak promjene stanja React odgovara ponovnim iscrtavanjem korisničkog sučelja. Za integraciju Reacta i Reduxa koristi se „react-redux“ biblioteka.

5.2.3. Webpack

Webpack je *open source* JavaScript *module bundler*. U prijevodu to znači da ima zadaću uzeti sve JavaScript module u projektu skupa s njihovim ovisnim modulima (engl. *dependencies*), te generirati jedan statički paket koji će web server slati korisnicima. Webpack dolazi s uključenim razvojnim serverom koji prati promjene u kodu i automatski isporučuje statičke datoteke tijekom razvoja aplikacije. Webpack također napravi prilagodbu

JavaScript koda za korištenje u starijim internet preglednicima. (Webpack https://webpack.js.org/concepts/ srpanj 2019.)

5.2.4. Websockets

Websocket protokol se koristi za razvoj modernih *real time* aplikacija. Prije pojava websocket protokola aplikacije su se koristile *polling* metodama, odnosno metodama periodičkog dohvaćanja podatka za simulaciju dvosmjerne komunikacije između servera i klijenta. (WebSockets https://www.linode.com/docs/development/introduction-to-websockets/ srpanj 2019.)

Polling je zapravo sinkroni (engl. *synchronous*) zahtjev za provjeru novih podatka od klijenta prema serveru. *Polling* radi najbolje u slučajevima kada se zna interval pojavljivanja novih poruka na serveru za klijenta. U modernim *real time* aplikacijama frekvencija pojava novih poruka je uglavnom nepredvidiva.

Long polling je druga komunikacijska metoda za simulaciju *real time* aplikacija. Funkcionira na sljedeći način. Otvori se konekcija prema serveru na određen vremenski period i započne se provjera za nove informacije, odnosno poruke. Ukoliko server nema novih informacija za klijenta, konekcija se svejedno drži otvorenom do pojave novih poruka, ili do isteka definiranog vremenskog ograničenja (engl. *timeout*)

Polling i *Long polling* metode simulacije *real time* aplikacija su zastarjele i nisu preporučene za korištenje u modernim aplikacijama.

Websocket protokol je predstavljen kao dio HTML5 specifikacije i omogućuje pravu dvosmjernu komunikaciju (engl. *full-duplex*) između klijenta i servera. Komunikacija se odvija kroz TCP protokol na portu 80, odnosno 443 u slučaju zaštićenih konekcija (engl. *encrypted connections*). Korištenje standardnih portova je prednost za korporacijska okruženja gdje se većina konekcija s drugih portova automatski blokira.

Websocket protokol specifikacija definira `ws` i `wss` URI sheme za nezaštićenu i zaštićenu komunikaciju i osim kroz internet pretraživače može se koristiti i na mobilnim platformama.

Većina internet pretraživača podržava websocket protokol, uključujući Google Chrome, Firefox, Microsoft Edge, Safari i Opera.

```
var ws = new WebSocket("wss://echo.websocket.org");
```

Kôd 5.3 Primjer spajanja na server pomoću Websocket protokola u JavaScriptu

6. Implementacija

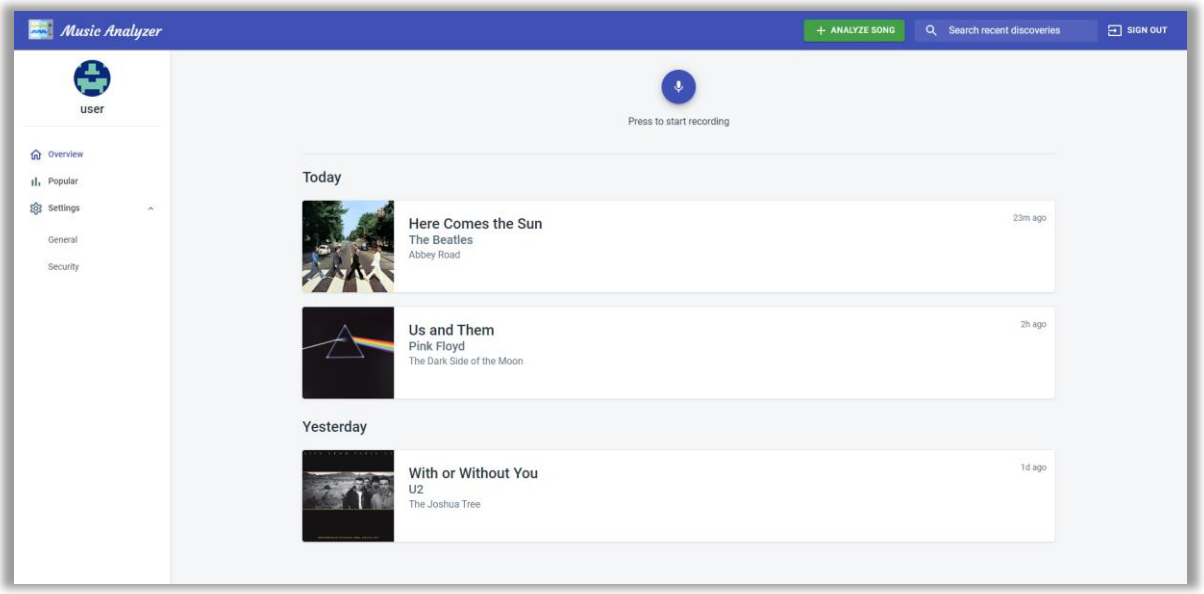
Razvijena aplikacija se sastoji od dva modula, *backend* modula i *frontend* modula. Oba modula su razvijena koristeći Visual Studio Code editor uz nekoliko pomoćnih konzolnih alata. U ovom poglavlju prvo će se opisati sve funkcionalnosti aplikacije, zatim detalji implementacije poslužiteljske aplikacije, odnosno *backend* modula, te klijentske aplikacije, odnosno *frontend* modula.

6.1. Funkcionalnosti aplikacije

Aplikacija omogućuje korisnicima prepoznavanje glazbe snimanjem kratkog uzorka pomoću mikrofona ugrađenog u računalo. U ovom poglavlju opisati će se funkcionalnosti aplikacije kroz prikaz korisničkog sučelja.

Korisničko sučelje se sastoji od nekoliko dijelova:

- zaglavlje (engl. *header*)
- navigacijski izbornik
- prikaz odabranog sadržaja



Slika 6.1 Prikaz sučelja aplikacije

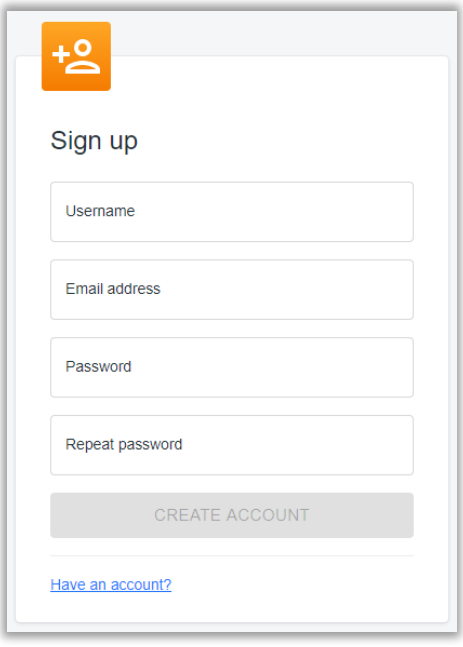
Slika 6.1 prikazuje cijelo korisničko sučelje aplikacije s odabranom opcijom „*Overview*“ koja prikazuje kontrolu za snimanje audio uzorka na vrhu i sve prijašnje rezultate na donjem dijelu stranice.

Zaglavlje aplikacije sadržava kontrolu za odjavljivanje korisnika, polje za pretraživanje starih rezultata snimljenih uzorka i gumb za otvaranje prozora za dodavanje novog zapisa u bazu.

Navigacijski izbornik omogućuje korisnicima jednostavnu navigaciju kroz sučelja aplikacije. Ponuđene opcije su pregled povijesti snimljenih uzorka, pregled statistika aplikacije i upravljanje korisničkim postavkama. Opcija „*Settings*“ sadržava pod opcije za uređivanje informacija o korisniku i funkcionalnost promjene korisničke lozinke.

6.1.1. Registracija i prijava korisnika

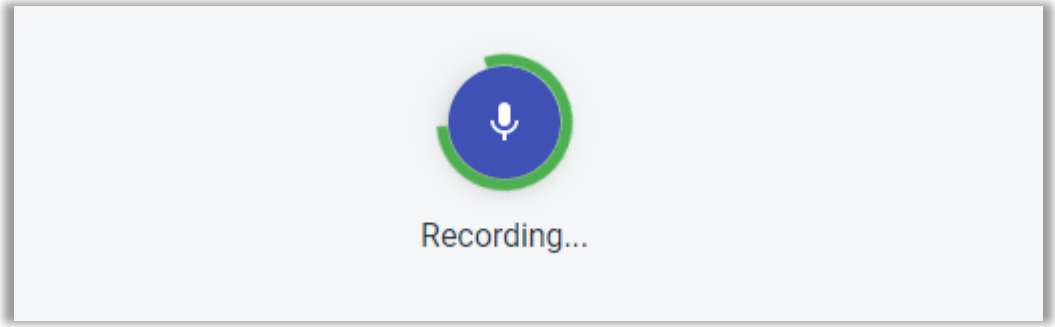
Slika 6.2 prikazuje formu za registraciju korisnika. Da bi se podaci s forme uspješno poslali na server korisnik mora ispuniti polje za unos korisničkog imena (engl. *username*), polje za unos email adrese, te polje za odabir željene lozinke za prijavu u aplikaciju s njezinom potvrdom. Kako bi se registracija uspješno provela korisnik mora ispuniti sva polja. Ukoliko korisnik ispuni neko polje na nepravilan način, odmah se pojavljuje poruka s opisom greške ispod neispravno ispunjenog polja. Nakon uspješno ispunjene forme, korisniku se omogućuje pritisak „*Create Account*“ gumba. Nakon što server uspješno obradi registraciju, korisnika se automatski preusmjerava na stranicu za prijavu gdje može upisati svoje korisničko ime i lozinku. Nakon upisa ispravnih podataka za prijavu, korisnika se preusmjerava na glavno sučelje aplikacije (Slika 6.1). U slučaju unosa neispravnih podataka, na sučelju se prikazuje poruka s greškom.



Slika 6.2 Forma za registraciju korisnika

6.1.2. Snimanje audio uzorka

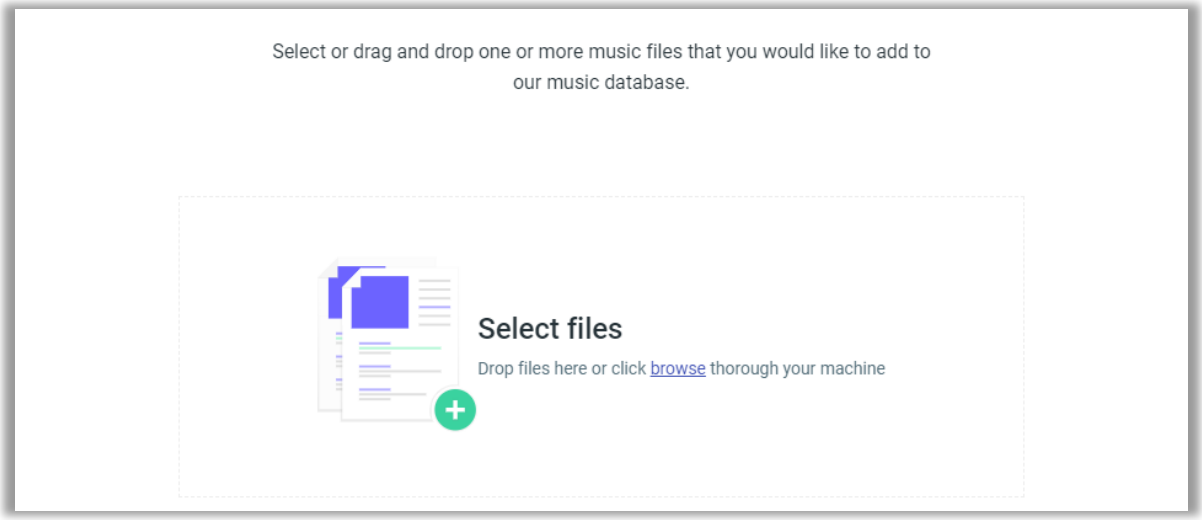
Snimanje audio uzorka može započeti nakon što korisnik odabere opciju „*Start recording*“ u gornjem dijelu „Overview“ opcije navigacijskog izbornika. Slika 6.3 prikazuje stanje sučelja nakon pritiska na „*Start recording*“ gumb. Nakon završetka snimanja korisnik dobiva povratnu informaciju o statusu audio uzorka i gumb se vraća u stanje prije početka snimanja.



Slika 6.3 Prikaz sučelja tijekom snimanja audio uzorka

6.1.3. Dodavanje novog audio zapisa

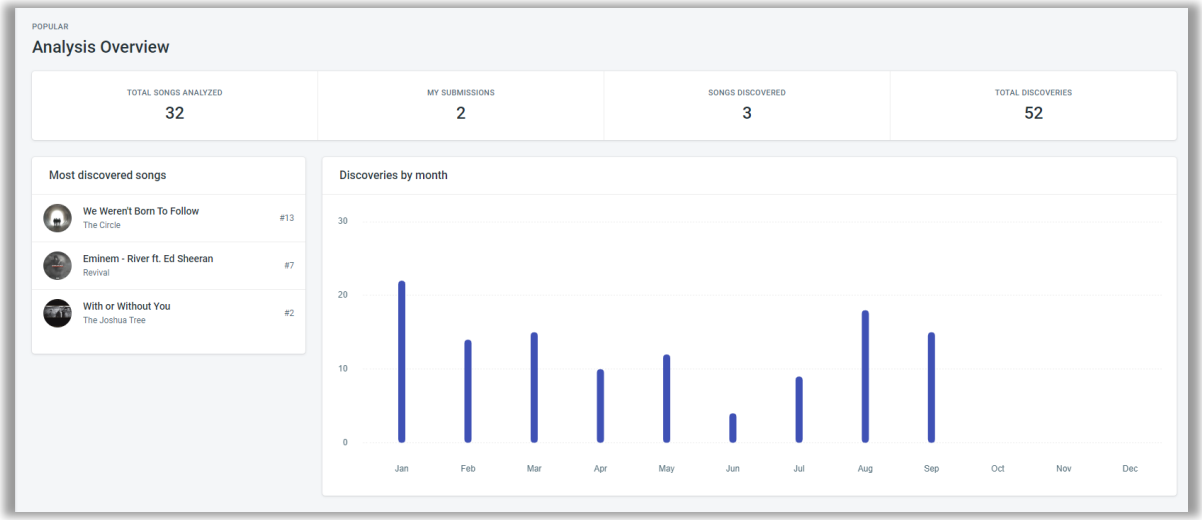
Prozor za dodavanje novog audio zapisa (Slika 6.4) se otvara pritiskom na gumb u zaglavlju aplikacije. Korisnik ima mogućnost odvlačenja datoteka u prozor ili klikom na sam prozor se otvara izbornik operacijskog sustava za odabir datoteke. Nakon što korisnik odabere datoteku, pojavljuje se gumb za slanje podataka na server. Završetkom slanja korisnika se obavještava o statusu završenog prijenosa i prozor se automatski zatvara.



Slika 6.4 Prozor za dodavanje novog audio zapisa

6.1.4. Pregled statistika

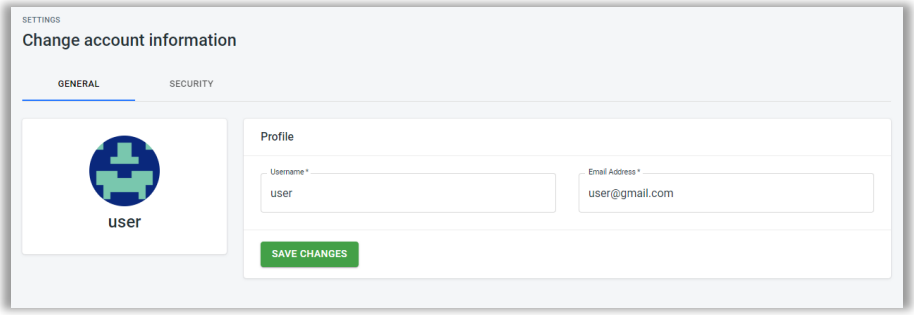
Sučelje za prikaz statistika cijele aplikacije (Slika 6.5) omogućuje korisnicima pregled aktivnosti svih korisnika aplikacije. Do sučelja se dolazi odabirom „*Popular*“ opcije u navigacijskom izborniku. Na sučelju su prikazane pjesme za koje je najviše puta snimljen audio uzorak, ukupan broj poslanih pjesama od strane svih korisnika, ukupan broj poslanih pjesama od strane prijavljenog korisnika, ukupan broj uspješno prepoznatih uzoraka prijavljenog korisnika i ukupan broj uspješno prepoznatih uzoraka svih korisnika.



Slika 6.5 Prikaz statistika cijele aplikacije

6.1.5. Korisničke postavke

Sučelje za prikaz korisničkih postavki omogućuje korisniku promjenu osnovnih informacija korisničkog računa. Do sučelja se dolazi odabirom „*Settings*“ opcije u navigacijskom izborniku. Pod kraticom „*General*“ su opcije promjene korisničkog imena i email adrese. Pod kraticom „*Security*“ ponuđena je opcija promjene korisničke lozinke. Slika 6.6 prikazuje sučelje na zadanoj „*General*“ kratici.



Slika 6.6 Sučelje za promjenu korisničkih postavki

6.2. Poslužiteljska aplikacija

U ovom poglavlju opisati će se neki detalji implementacije poslužiteljske Phoenix aplikacije. Opisati će se konfiguracijske postavke, proces autentikacije i autorizacije, pokazati će se proces primanja i spremanja podataka, te na kraju će se pokazati implementacija *real time* komunikacije.

6.2.1. Konfiguracija

Phoenix poslužiteljsku aplikaciju pokreće Elixir `Application` modul. `Application` modul je zadužen za učitavanje konfiguracije i pokretanje životnog ciklusa aplikacije. Moduli koji koriste `Application` modul moraju implementirati `start` metodu životnog ciklusa. Kôd 6.1 Implementacija MusicAnalyzer.Application modula prikazuje implementaciju `MusicAnalyzer.Application` modula koji je zaslužan za pokretanje aplikacije. Unutar metode `start` definira se lista procesa koju će `Supervisor` modul nadzirati. `Supervisor` modul upravlja procesom koji se koristi za komunikaciju s bazom, a u kodu je definiran kao `MusicAnalyzer.Repo` modul. Također upravlja i s `MusicAnalyzerWeb.Endpoint` modulom koji je zadužen za obradu svih ulaznih zahtjeva prema serveru, ali i svih odgovora prema klijentima.

```
defmodule MusicAnalyzer.Application do
  use Application

  def start(_type, _args) do
    # Child processes to be supervised
    children = [
      MusicAnalyzer.Repo,
      MusicAnalyzerWeb.Endpoint
    ]

    opts = [strategy: :one_for_one,
    name: MusicAnalyzer.Supervisor]
    Supervisor.start_link(children, opts)
  end
end
```

Kôd 6.1 Implementacija MusicAnalyzer.Application modula

Procesi koji će biti pod nadzorom se pokreću pozivom `Supervisor.start_link` metode koja prima listu procesa i listu s opcijama. Lista opcija `opts` definira strategiju oporavka procesa od grešaka. Definirana strategija `:one_for_one` govori `Supervisor` modulu da u slučaju bilo kakve greške ponovno pokrene samo onaj proces koji je prestao raditi.

6.2.2. Rad s bazom

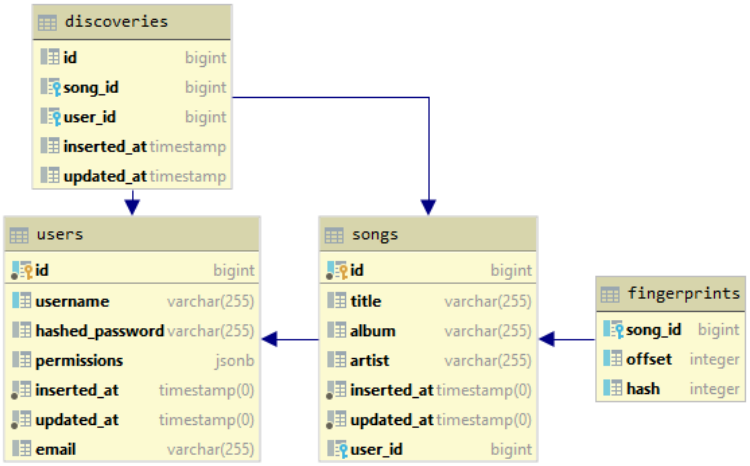
Komunikaciju s bazom odrađuju Ecto moduli opisani u poglavlju 5.1.3. Da bi se entiteti mogli spremati u bazu, potrebno je definirati shemu pomoću `Ecto.Shema` modula (Kôd 6.2 Shema korisničkog modula). Postavljanjem izraza `use Ecto.Schema` u modulu definira se novi modul za reprezentaciju entiteta.

```
defmodule MusicAnalyzer.Users.User do
  use Ecto.Schema
  import Ecto.Changeset

  schema "users" do
    field :hashed_password, :string
    field :permissions, :map
    field :username, :string
    field :password, :string, virtual: true
    timestamps()
  end

  def changeset(%User{} = user, attrs) do
    user
    |> cast(attrs, [:username, :password, :permissions])
    |> validate_required([:username, :password,
:permissions])
    |> unique_constraint(:username)
    |> put_hashed_password()
  end
end
```

Kôd 6.2 Shema korisničkog modula

U modul koji predstavlja shemu je također uključen `Ecto.Changeset` modul koji se koristi za validaciju podataka prije spremanja promjena u bazu. Izrazom `schema "users"` definiraju se elementi korisničkog entiteta. Promatrajući Kôd 6.2 Shema

korisničkog modula može se primijetiti da polje `:password` sadrži ključnu riječ `virtual` u definiciji. Ključna riječ `virtual` označava da entitet sadržava polje `:password`, ali ga neće nikada izložiti vanjskim zahtjevima korisnika. Za validaciju entiteta se koristi `changeset` metoda iz `Ecto.Changeset` modula. Metoda kao prvi parametar prima korisničku strukturu i ulazne atribute, nakon toga se pozivaju metode iz `Ecto.Changeset` modula koje rade validaciju definiranih polja. Nakon što su definirani svi elementi entiteta, pokretanjem naredbe `mix ecto.migrate`, promjene entiteta primjenjuju se na bazu.



Slika 6.7 Model baze podataka

Primjer poziva metode za kreiranje korisnika (Kôd 6.3) koristi `MusicAnalyzer.Users` repozitorij baziran na `Ecto.Repo` modulu. Metoda `create` kao ulazni parametar prima reprezentaciju korisničkog objekta u JSON formatu. Pozivom `create_user` metode na `MusicAnalyzer.Users` repozitoriju kreira se novi korisnik u bazi. Ukoliko metoda vrati strukturu oblika `{:ok, %User{}}` pozivatelju se odgovori reprezentacijom novog korisnika u JSON formatu.

```
  def create(conn, %{"user" => user_params}) do
    with {:ok, %User{} = user} <-
Users.create_user(user_params) do
      conn
      |> put_status(:created)
      |> put_resp_header("location",
Routes.user_path(conn, :show, user))
      |> render("show.json", user: user)
    end
  end
```

Kôd 6.3 Metoda kontrolera za kreiranje novog korisnika

6.2.3. Komunikacija u realnom vremenu

Komunikacija u realnom vremenu implementirana je korištenjem Phoenix `Channels` modula. `Channels` modul omogućuje brzu uspostavu websocket kanala.

Da bi se uspostavila dvosmjerna konekcija, potrebno je definirati pristupnu točku u `MusicAnalyzerWeb.Endpoint` modulu. Nakon što je definirana pristupna točka potrebno je napisati implementaciju modula koji će odrađivati proces komunikacije s klijentima.

```
defmodule MusicAnalyzerWeb.InfoChannel do
  use Phoenix.Channel

  def handle_in("data", %{"body" => body}, socket) do
    # hanndle data
    {:noreply, socket}
  end

  def join("info:hub", _message, socket) do
    {:ok, socket}
  end
end
```

Kôd 6.4 Modul za dvosmjernu komunikaciju u realnom vremenu

Kôd 6.4 prikazuje implementaciju modula za *real time* komunikaciju. Modul sadrži dvije metode iz `Phoenix.Channel` modula. Metoda `handle_in` obrađuje ulazne poruke od klijenta i šalje povratne informacije u realnom vremenu. Metoda `join` omogućuje korisnicima pretplatu na informacije vezane za `"info:hub"` kanal.

6.3. Klijentska aplikacija

U ovom poglavlju opisati će se neki detalji implementacije klijentske React aplikacije. Opisati će se dijelovi koda za snimanje zvuka i proces komunikacije s poslužiteljem.

6.3.1. Snimanje zvuka

Snimanje zvuka u aplikaciji realizirano je pomoću Recorder.js biblioteke. Da bi se moglo započeti snimanje, potrebno je od korisnika zatražiti dopuštenje za korištenje mikrofona

pomoću `navigator.mediaDevices.getUserMedia` metode. Ako korisnik dopusti korištenje mikrofona snimanje zvuka automatski započinje (Kôd 6.5).

Treba napomenuti da većina modernih internet pretraživača omogućuje pamćenje postavke za dopuštenje korištenja mikrofona, prema tome korisnik mora samo jednom dati dopuštenje za korištenje mikrofona.

Snimanje započinje inicijalizacijom `Recorder` objekta. `Recorder` objekt prima dva parametra.

Prvi parametar je input objekt koji vraća metoda `createMediaStreamSource` `audioContext` objekta. Drugi parametar predstavlja konfiguracijske postavke za snimanje. Samo snimanje započinje pozivom `record` metode na `rec` objektu.

```
const startRecording = () => {
    var constraints = { audio: true, video:false };
  navigator.mediaDevices.getUserMedia(constraints).then(f
unction(stream) {
        var audioContext = new AudioContext();
        input =
audioContext.createMediaStreamSource(stream);
        var rec = new Recorder(input,{numChannels:1});
        rec.record();

        console.log('Recording...');
  }).catch(function() {
        console.log('Error!');
  });
}
```

Kôd 6.5 Proces zahtjeva za dopuštenje snimanja i početak snimanja

6.3.2. Komunikacija s poslužiteljem

Komunikacija s poslužiteljem odvija se uz pomoć Axios biblioteke. Klijentska React aplikacija koristi Axios biblioteku na puno mjesta pa je napravljen *singleton* JavaScript `api` modul koji olakšava pozivanje metoda (Kôd 6.6). `Api` modul poziva `create` metodu s unaprijed definiranim postavkama za spajanje na server, i uvijek vraća tu istu instancu kod svakog poziva `api` modula.

```
import axios from 'axios';

const instance = axios.create({
    baseURL: process.env.REACT_APP_API
});

export default instance;
```

Kôd 6.6 JavaScript modul za komunikaciju s poslužiteljem

Kôd 6.7 prikazuje slanje POST zahtjeva za prijavu korisnika koristeći prethodno opisan `api` modul. Metoda `post` na `api` objektu kao prvi parametar prima putanju do nekog resursa na poslužitelju, a drugi parametar je objekt koji sadrži informacije koje se šalju. Vraća objekt tipa `Promise` koji predstavlja akciju koje će se izvršiti u nekom budućem vremenu. Tijekom izvršavanja zahtjeva, `Promise` objekt je u stanju *pending*. Kada zahtjev završi uspješno, stanje mu je *fulfilled*, a u slučaju neuspješnog zahtjeva, bilo zbog greške na serveru ili u konekciji stanje mu je *rejected*.

```
api.post('/identity/login', {
                username: username,
                password: password,
            })
            .then((response) => response.data)
```

Kôd 6.7 Primjer korištenja modula za komunikaciju s poslužiteljem

Zaključak

U ovom radu opisan je jedan mali dio mogućnosti koje pruža široko područje audio analize. Digitalni svijet današnjice prepun je uređaja koji obrađuju zvuk na neki način, od virtualnih kućnih asistenata, automobila i mobilnih uređaja koje svatko nosi sa sobom. Primjene za audio analizu su neograničene.

Korištenjem jednostavnih web tehnologija izrađena je aplikacija za prepoznavanje audio zapisa snimanjem kratkih uzoraka zvuka. Fokus ovog rada je bio na prepoznavanju glazbenih audio zapisa i dohvaćanju podataka o prepoznatim pjesmama. Na temelju istih algoritama i tehnologija mogla se je razviti aplikacija koja prepoznaje korisnikov glas i potom odrađuje neku akciju. Prema tome se može zaključiti, da, čak i za ovaj mali segment iz područja audio analize ima nebrojeno mnogo primjena.

Daljnjim razvojem ove aplikacije mogla bi se dodati integracija s raznim servisima za pretraživanje glazbe koja bi omogućila korisnicima još lakši i jednostavniji pronalazak omiljenih melodija.

Popis kratica

ADC	*Analog to digital converter*	analogno digitalni pretvarač
API	*Application programming interface*	aplikacijsko programsko sučelje
CRUD	*Create, Read, Update, Delete*	stvaranje, čitanje, ažuriranje, brisanje
DAC	*Digital to analog converter*	digitalno analogni pretvarač
DOM	*Document Object Model*	objektni model dokumenta
HTTP	*HyperText Transfer Protocol*	
MVC	*Model View Controller*	model pogled upravitelj
NLU	*Natural language understanding*	razumijevanje prirodnog jezika
PCM	*Pulse Coded Modulation*	
SDK	*Software development kit*	paket za razvoj programa
SMS	*Short Message Service*	usluga kratkih poruka
SPA	*Single Page Application*	jednostranična aplikacija
SQL	*Structured Query Language*	strukturni jezik za upite
URI	*Uniform Resource Identifier*	usklađeni identifikator resursa

Popis slika

Popis kôdova

Literatura

[1] S. JURIĆ, *Elixir in action*, Manning, 2019.

[2] R. C. MARTIN, *Clean Code: A Handbook of Agile Software Craftsmanship*,Prentice Hall, 2008.

[3] I. LJUBI, B. MIHALJEVIĆ, *Interoperabilnost informacijskih sustava*, Algebra d.o.o, Zagreb, 2013.

[4] J.G. PROAKIS, D.K. MANOLAKIS, *Digital Signal Processing*, Prentice Hall, 2006

[5] Audio editing software
https://en.wikipedia.org/wiki/Audio_editing_software lipanj 2019.

[6] Digital audio editors
https://en.wikipedia.org/wiki/Comparison_of_digital_audio_editors lipanj 2019.

[7] Shazam
https://medium.com/swlh/the-story-of-shazam-the-startup-days-6bccebd17d84 lipanj 2019.

[8] An Industrial-Strength Audio Search Algorithm
https://www.ee.columbia.edu/~dpwe/papers/Wang03-shazam.pdf lipanj 2019.

[9] SoundHound
https://www.wired.com/2015/06/soundhound-hound/ lipanj 2019.

[10] Chromaprint
https://oxygene.sk/2011/01/how-does-chromaprint-work/ lipanj 2019.

[11] ACRCloud
https://docs.acrcloud.com/docs/acrcloud/ lipanj 2019.

[12] Širenje zvuka
http://dog.zesoi.fer.hr/predavanja/HTML/Sirenje%20zvuka.htm lipanj 2019.

[13] Zvuk
https://hr.wikipedia.org/wiki/Zvuk lipanj 2019.

[14] How does Shazam work
http://coding-geek.com/how-shazam-works/ lipanj 2019.

[15] Music and Computers, theoretical and historical approach
http://sites.music.columbia.edu/cmc/MusicAndComputers/ lipanj 2019.

[16] Fourierov red i Fourierova transformacija
https://hrcak.srce.hr/file/121546 lipanj 2016.

[17] Diskretna Fourierova transformacija
http://www.mathos.unios.hr/~scitowsk/MP/17-Mihaljcic-Petrijevcanin-2.pdf lipanj 2019.

[18] PostgreSQL
https://www.ionos.com/digitalguide/server/know-how/postgresql/ srpanj 2019.

[19] Phoenix
https://hexdocs.pm/phoenix/up_and_running.html srpanj 2019.

[20] Ecto
https://hexdocs.pm/ecto/Ecto.html srpanj 2019.

[21] React
https://reactjs.org/docs/react-api.html srpanj 2019.

[22] Redux
https://redux.js.org/api/api-reference srpanj 2019.

[23] Webpack
https://webpack.js.org/concepts/ srpanj 2019.

[24] WebSockets
https://www.linode.com/docs/development/introduction-to-websockets/ srpanj 2019.

„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,_____.