Razvoj web-aplikacije za dodavanje zvučnih efekata i snimanje glasa

Aničić, Adrian

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 1587

RAZVOJ WEB-APLIKACIJE ZA DODAVANJE ZVUČNIH EFEKATA I SNIMANJE GLASA

Adrian Aničić

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 1587

RAZVOJ WEB-APLIKACIJE ZA DODAVANJE ZVUČNIH EFEKATA I SNIMANJE GLASA

Adrian Aničić

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 4. ožujka 2024.

ZAVRŠNI ZADATAK br. 1587

Pristupnik:	**Adrian Aničić (0036529543)**
Studij:	Elektrotehnika i informacijska tehnologija i Računarstvo
Modul:	Računarstvo
Mentor:	doc. dr. sc. Mario Brčić

Zadatak: **Razvoj web-aplikacije za dodavanje zvučnih efekata i snimanje glasa**

Opis zadatka:

Ovaj rad zahtijeva razvoj web-aplikacije koja će omogućiti korisnicima da uživo reproduciraju svoj glas putem mikrofona na web-stranici te da na taj glas dodaju različite efekte poput kompresije, ekvalizatora i odjeka. Korisnicima će biti omogućeno snimanje modificiranih audio zapisa i njihovo preuzimanje. Za razvoj aplikacije koristit će se Next.js framework zajedno s Tone.js bibliotekom za manipulaciju zvukom. Cilj je omogućiti korisnicima intuitivno iskustvo u radu s audio efektima na web platformi.

Rok za predaju rada: 14. lipnja 2024.

Sadržaj

1 Uvod 3

2 Razvojno okruženje 4

2.1 Typescript 4

2.2 React 4

2.3 Next.js 4

2.4 TailwindCSS 5

2.5 Tone.js 5

3 Organizacija koda 6

3.1 Public mapa 6

3.2 App mapa 6

3.3 Components mapa 7

3.4 Context mapa 8

3.5 Mape types i utils 9

4 Implementacija 10

4.1 StreamingContext.tsx 10

4.2 EffectContext.tsx 11

4.3 4.3 EffectReducer.ts i initial-state.ts 13

4.3.1 EffectReducer.ts 13

4.3.2 initial-state.ts 15

4.4 connectEffect.ts 15

4.5 EffectList.tsx i EffectContainer.tsx 17

4.5.1 EffectList.tsx 17

4.5.2 EffectContainer.tsx 19

4.6 Postavke reprodukcije 20

4.7 Vizualizator 23

4.8 Efekti 25

4.8.1 Izrada pomoćnih komponenti 25

4.8.2 Reverb 29

4.8.3 Delay 30

4.8.4 Distorzija 31

4.8.5 Tremolo 33

4.8.6 Chorus 34
4.8.7 Filter 35
5 Zaključak 37
Literatura 38
Preuzete Slike 39

1 Uvod

Web aplikacije jedan su od najučestalijih načina na koji korisnici ulaze u interakciju s digitalnim sadržajima. Iz tog razloga, cilj ovog rada istražiti je suvremene pristupe i tehnologije u razvoju web aplikacija, fokusirajući se na alate poput Next.js, React, Tone.js, TypeScript i Tailwind CSS. Glavni cilj rada je izraditi funkcionalnu web aplikaciju koja omogućuje dodavanje efekata na audio zapise, pružajući korisnicima mogućnost da prenose audio datoteke ili streamaju zvuk uživo putem mikrofona.

Audio aplikacije nisu česte u web okruženju, a ovaj projekt nudi priliku za istraživanje i implementaciju aplikacije koja korisnicima nudi mogućnost obrade zvuka u pregledniku, u stvarnom vremenu. Iz tog razloga je odabrana ova tema.

Next.js je korišten za izgradnju brze i optimizirane aplikacije s jednostavnom integracijom raznih funkcionalnosti. React je korišten za izradu komponenti koje čine sveukupno korisničko sučelje. Tone.js je library koji je korišten za manipulaciju zvukom, omogućavajući primjenu različitih audio efekata. TypeScript je korišten za olakšavanje održivosti koda kroz statičko tipiziranje, dok je Tailwind CSS omogućio brzu i efikasnu stilizaciju aplikacije.

Ovaj rad pruža sveobuhvatan pregled i analizu korištenih tehnologija kroz praktičan primjer izrade web aplikacije za dodavanje efekata na audio zapise. Na taj način, pokazat ćemo kako se moderni alati i tehnologije mogu integrirati u stvaranju inovativne i korisnički usmjerene web aplikacije.

2 Razvojno okruženje

Za razvoj ovog projekta korišteni su TypeScript zbog svoje stroge tipiziranosti, React i Next.js za izradu korisničkog sučelja, Tailwind CSS za stilizaciju, te Tone.js za manipulaciju zvukom. Detaljniji opis ovih tehnologija nalazi se u sljedećim potpoglavljima.

2.1 Typescript

TypeScript je snažno tipiziran, objektno orijentiran, kompajlirani programski jezik koji se nadograđuje na JavaScript. On je nadskup jezika JavaScript, osmišljen da vam pruži bolje alate na bilo kojoj skali.

Glavni arhitekt iza TypeScripta je Anders Hejlsberg, dizajner C# jezika u Microsoftu. TypeScript je otvorenog koda, podržan od strane Microsofta, i smatra se i jezikom i skupom alata. TypeScript se opisuje kao "JavaScript sa sintaksom za tipove." Ukratko, to je JavaScript s nekim dodatnim značajkama. [1]

2.2 React

React je JavaScript biblioteka koja se koristi za izradu korisničkih sučelja u web aplikacijama. Ona omogućava programerima da razvijaju dinamička i interaktivna korisnička sučelja kroz kompoziciju malih, ponovno iskoristivih komponenti.

Jedna od glavnih karakteristika Reacta je „reconciliation.“. To je proces koji omogućuje efikasnije ažuriranje korisničkog sučelja putem brzog uspoređivanja virtualnog i stvarnog DOM-a te ažuriranja samo promijenjenih dijelova stranice. Ova tehnika čini React vrlo efikasnim u radu s dinamičkim podacima i velikim aplikacijama.

React se često koristi u kombinaciji s drugim tehnologijama kao što su Redux za upravljanje stanjem aplikacije, React Router ili Next.js za upravljanje rutama, i razni CSS okviri za stilizaciju korisničkog sučelja poput Bootstrapa ili Tailwinda, koji je korišten u ovom radu. [2]

2.3 Next.js

Next.js je popularni open-source framework za razvoj web aplikacija, posebno fokusiran na React.js, koji omogućuje izgradnju brzih i skalabilnih web stranica i aplikacija. Neke njegove

značajke su recimo Server-side rendering (SSR), što omogućuje generiranje HTML-a na serveru prije nego što se stranica pošalje pregledniku. To rezultira bržim prikazom sadržaja na stranici i poboljšava SEO. Nadalje, Next.js podržava i generiranje statičkih stranica. Ovo je korisno za stranice koje se ne mijenjaju često i omogućuje brži pristup sadržaju. Next.js ima ugrađen sustav za rutiranje, što olakšava upravljanje navigacijom između stranica u aplikaciji, lako se integrira s drugim alatima i tehnologijama, poput TypeScript-a, GraphQL-a, i raznih CMS-ova. [6]

2.4 TailwindCSS

TailwindCSS je CSS framework koji se fokusira na "utility-first" pristup stiliziranju web stranica i aplikacija. Ovo znači da umjesto korištenja predefiniranih komponenti ili klasa, poput "button" ili "card", Tailwind CSS pruža veliki skup pojedinačnih CSS klasa koje se mogu koristiti direktno u HTML-u kako bi se definirali stilovi.

2.5 Tone.js

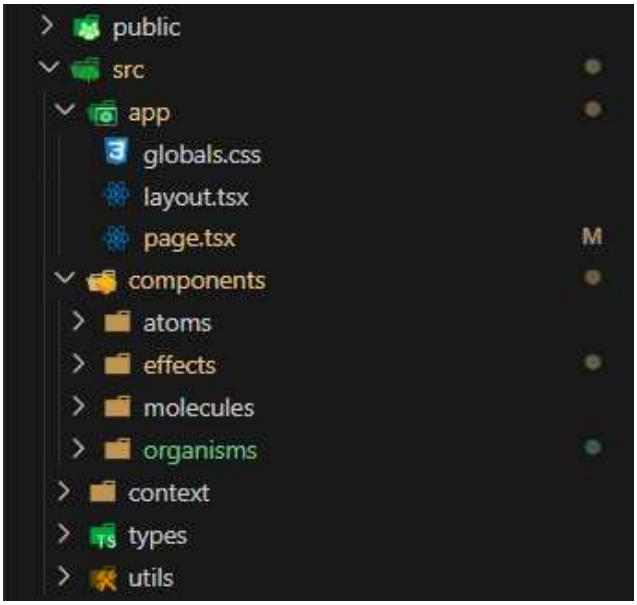
Tone.js je open-source JavaScript biblioteka za manipulaciju zvukom u web aplikacijama. Izgrađena je nad Web Audio API-jem, standardnom JavaScript API specifikacijom koja omogućuje programski pristup zvučnom sustavu i obradi zvuka unutar web preglednika.

Web Audio API omogućuje developerima pristup raznim zvukovnim mogućnostima iz JavaScripta. To uključuje reprodukciju audio datoteka, generiranje zvukova, audio obradu, sinkronizaciju i analizu zvuka. Na primjer, možete reproducirati MP3 datoteke, generirati tonove koristeći oscilatore, primijeniti efekte poput reverba ili delay-a, sinkronizirati zvuk s vizualnim elementima na stranici te analizirati amplitude i frekvencije zvuka.

Tone.js nadograđuje ove mogućnosti pružajući dodatne alate i apstrakcije za lakši rad s Web Audio API-jem. Omogućuje snimanje i reprodukciju zvuka u stvarnom vremenu te dodavanje raznih ugrađenih efekata i filtera na zvukove. Ovo čini Tone.js snažnim alatom za stvaranje bogatih audio iskustava unutar web aplikacija, uključujući glazbene aplikacije, igre, interaktivne iskustva i mnogo više. [3]

3 Organizacija koda

U ovom poglavlju detaljno je opisana struktura direktorija i organizacija koda projekta, što je prikazano na slici Slika 3.1. Pravilna organizacija koda ključna je za održivost i proširivost projekta.



Slika 3.1 Struktura direktorija

3.1 Public mapa

Public mapa sadrži statičke datoteke, kao što su slike i fontovi, koje su direktno dostupne putem URL-a na vašoj web stranici. Sve datoteke unutar ove mape se serviraju klijentu bez dodatne obrade ili prevođenja od strane vašeg build alata ili frameworka.

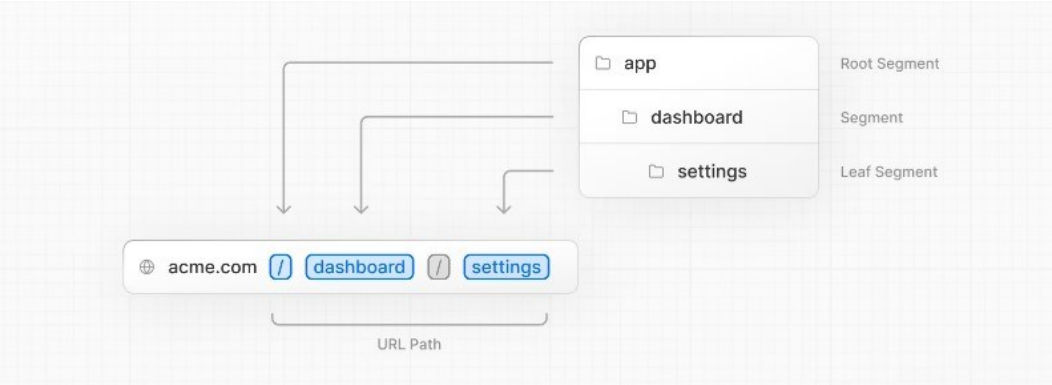
3.2 App mapa

Od verzije 13, Next.js je uveo promjene u načinu rukovanja rutama, zamjenjujući mapu "pages" mapom "app". Ova promjena olakšava organizaciju projekta i rutiranje unutar aplikacije. Svaka podmapa unutar mape "app" odgovara određenom segmentu URL rute. Na primjer, ako imate rutu "/blog", možete stvoriti podmapu "blog" unutar mape "app" i definirati sve stranice povezane s blogom unutar te podmape. Ovo je bolje prikazano na slici Slika 3.2.

Ovaj pristup pojednostavljuje strukturu projekta i čini je intuitivnijom, jer se strukturno podudara s URL rutama. Također, ovo uklanja potrebu za korištenjem dodatnih biblioteka poput React Router za upravljanje rutama, što čini razvoj aplikacija u Next.js još jednostavnijim i efikasnijim.

Uz to, Next.js omogućuje kreiranje specifičnih datoteka poput not-found.tsx koja se automatski prikazuje kada određena ruta nije pronađena. Također, možete koristiti datoteku layout.tsx kako biste definirali zajedničke elemente prikazane na svim stranicama unutar određene podmape ili na istoj razini mape "app".

Ove dodatne funkcionalnosti omogućuju veću fleksibilnost i kontrolu nad izgledom i ponašanjem aplikacije, pružajući programerima bolje alate za organizaciju koda i upravljanje prikazom stranica.



Slika 3.2 Prikaz routing sistema koji koristi Next.js

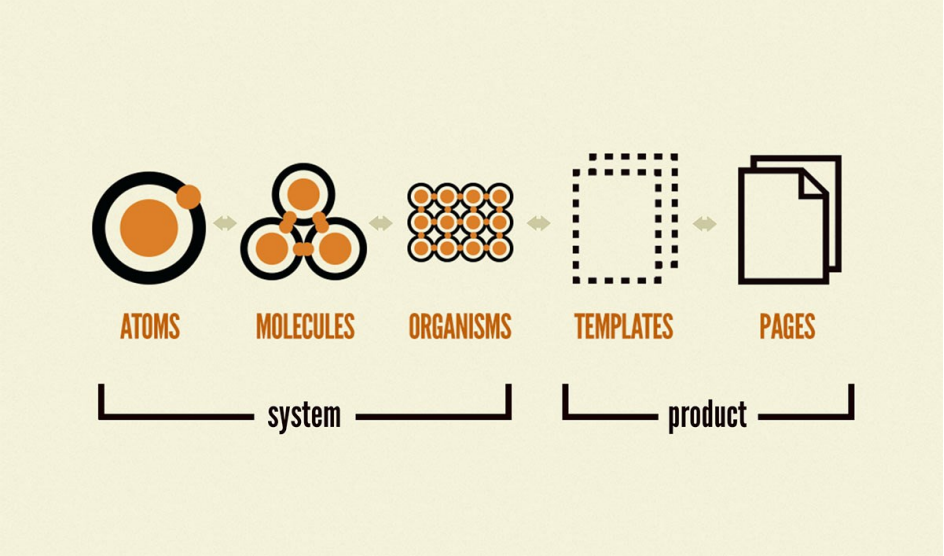
3.3 Components mapa

Atomic Design je metodologija dizajna koju je predstavio Brad Frost, a zagovara razbijanje korisničkih sučelja na najmanje, najfundamentalnije građevne blokove. Ti građevni blokovi, ili "atomi", mogu se kombinirati kako bi se stvorile složenije komponente poznate kao "molekule". Te molekule, pak, tvore različite "organizme" koji čine kompletno sučelje. [4]

U ovom projektu, organizacija datoteka slijedi princip Atomic Designa (Slika 3.3). Unutar mape "components" nalaze se sve komponente koje definiraju izgled korisničkog sučelja. Svaka komponenta predstavlja određeni građevni blok sučelja, počevši od najmanjih elemenata kao što su gumbi, pa sve do složenijih komponenti. Izostavljene su mape „Templates“ i „Pages“, jer ih opseg projekta nije zahtjevao.

Dodatno, u projektu se koristi i mapa "Effects" u kojoj su smještene komponente vezane za svaki efekt koji se može dodati na audio, a koji je implementiran u ovom projektu. Razlog za odvajanje

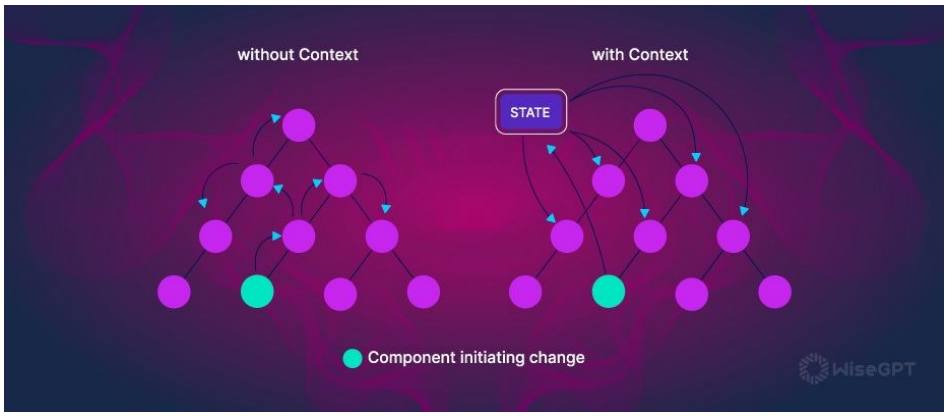
ovih komponenti u zasebnu mapu je što su dovoljno specifične i imaju poseban utjecaj na funkcionalnost sučelja.



Slika 3.3 Atomic Design

3.4 Context mapa

U okviru mape "context" u projektu se primjenjuje React-ov Context API za očuvanje globalnog stanja određenih varijabli, što omogućuje dijeljenje informacija između komponenti unutar aplikacije. Kontekst se koristi za praćenje važnih stanja poput vrste audiozapisa, statusa reprodukcije, redoslijeda i parametara efekata. Kroz implementaciju kontekstnih datoteka, kao što su one priložene u projektu, stvorena je struktura za globalno upravljanje stanjem, pružajući pristup podacima i funkcionalnostima unutar komponentnog stabla aplikacije bez potrebe za proslijeđivanjem „propova“ kroz cijelu hijerarhiju komponenti (Slika 3.4).



Slika 3.4 Prednost korištenja konteksta

Svaka kontekstna datoteka, poput one koja upravlja efektima audiozapisa ili streamingom audiozapisa, pruža jasnu organizaciju i strukturu za dijeljenje specifičnih informacija unutar aplikacije. Kroz korištenje Context API-ja, postiže se čistoća koda i olakšava se održavanje i proširenje aplikacije. Osim toga, pravilno upravljanje stanjem i ažuriranjem konteksta osigurava dosljednost podataka kroz cijelu aplikaciju, čineći je skalabilnom i prilagodljivom promjenama. Implementacija React-ovog Context API-ja predstavlja ključni dio arhitekture projekta, pružajući temelj za efikasno upravljanje stanjem i globalnim podacima unutar aplikacije.

3.5 Mape types i utils

U mapi "types" nalaze se datoteke koje definiraju tipove podataka korištenih unutar projekta. Ova organizacija omogućuje jednostavan pristup i upravljanje tipovima podataka na jednom centralnom mjestu.

U mapu "utils" smještene su pomoćne funkcije, uključujući i reducer funkcije. Reducer funkcije su posebno važne jer se koriste za upravljanje stanjem unutar aplikacije. One primaju trenutno stanje i akciju kao ulaz te vraćaju novo stanje aplikacije na temelju te akcije.

4 Implementacija

U ovom poglavlju detaljnije je opisano kako je ovaj projekt implementiran

4.1 StreamingContext.tsx

Datoteka StreamingContext.tsx ključna je komponenta aplikacije koja upravlja stanjem reprodukcije zvuka. Zajedno s EffectContext.tsx, ova datoteka omogućuje učinkovito upravljanje zvukom kroz cijelu aplikaciju.

StreamingContext (Slika 4.1) je kreiran pomoću React Context API-ja. Ovaj kontekst omogućava dijeljenje stanja reprodukcije zvuka (isStreaming) i funkcija za njegovo upravljanje među svim komponentama koje koriste ovaj kontekst.

```tsx
type StreamingContextType = {
  isStreaming: boolean;
  toggleIsStreaming: () => void;
  setIsStreaming: (newValue: boolean) => void;
};

const StreamingContext = createContext<StreamingContextType | undefined>(
  undefined
);

export const StreamingContextProvider: FC<{ children: ReactNode }> = ({
  children,
}) => {
  const [isStreaming, setStreaming] = useState(false);

  const toggleIsStreaming = () => {
    setStreaming((prev) => !prev);
  };

  const setIsStreaming = (newValue: boolean) => {
    setStreaming(newValue);
  };

  return (
    <StreamingContext.Provider
      value={{ isStreaming, toggleIsStreaming, setIsStreaming }}
    >
      {children}
    </StreamingContext.Provider>
  );
};

export const useStreaming = (): StreamingContextType => {
  const context = useContext(StreamingContext);
  if (!context) {
    throw new Error(
      "useStreaming must be used within a StreamingContextProvider"
    );
  }
  return context;
};

```

Slika 4.1 src/context/StreamingContext.tsx

StreamingContextType definira tipove podataka koji će biti pohranjeni u kontekstu: boolean vrijednost isStreaming, funkcija toggleIsStreaming za promjenu stanja i funkcija setIsStreaming za postavljanje specifične vrijednosti.

StreamingContextProvider je komponenta koja obuhvaća druge komponente i pruža im pristup kontekstu. Koristi useState za upravljanje stanjem isStreaming, koje prati je li reprodukcija zvuka aktivna. toggleIsStreaming mijenja trenutno stanje na suprotno, omogućavajući jednostavno prebacivanje između reprodukcije i pauze, a setIsStreaming postavlja stanje isStreaming na specifičnu vrijednost, što omogućuje precizno kontroliranje reprodukcije zvuka.

useStreaming je prilagođeni hook koji omogućuje komponentama pristup StreamingContext-u.

Ako se useStreaming koristi izvan StreamingContextProvider-a, baca se greška, osiguravajući da je kontekst uvijek dostupan samo unutar pravilno strukturiranih dijelova aplikacije.

4.2 EffectContext.tsx

Ovo je drugi kontekst i ključna komponenta unutar naše aplikacije koja upravlja efektima primijenjenim na reprodukciju zvuka. Umjesto osnovnog upravljanja stanjem s useState hook-om, odlučili smo se za upotrebu useReducer hook-a zbog složenosti promjene ovog stanja.

Glavna svrha ove komponente je definirati i pratiti stanje svih efekata. Svaki efekt je zaseban objekt sa svojim imenom i statusom aktivnosti (enabled), te setom parametara (params) koji su specifični za svaki efekt. Svi efekti spremljeni su u listu. Osim toga, pratimo i druge bitne podatke (Slika 4.2) kao što su izvor zvuka (audioSrc), koji se koristi u slučaju kada je sourceType „file“, tip izvora (sourceType) te trenutna pozicija reprodukcije (currentPosition) i vrijeme zadnjeg pauziranja (lastPausedTime) koje se koriste za nastavak reprodukcije od pozicije na kojem je stalo. Kod promjene „audioSrc“ ili „sourceType“, postavljaju se na 0.

```tsx
const [state, dispatch] = useReducer(effectReducer, initialState);

const [toneUserMedia, setToneUserMedia] = useState<Tone.UserMedia | null>(
  null
);
const [audioSrc, setAudioSrc] = useState<string | null>(null);
const [sourceType, setSourceType] = useState<SourceType>("usermedia");
const [audioPlayer, setAudioPlayer] = useState<Tone.Player | null>(null);
const [currentPosition, setCurrentPosition] = useState<number>(0);
const [levels, setLevels] = useState<Float32Array>();
const [lastPausedTime, setLastPausedTime] = useState<number>(0);
```

Slika 4.2 Stanja korištena u EffectContext.tsx

Kad dođe do promjena u stanju efekata, reprodukcija se automatski zaustavlja i ponovno pokreće kako bi se izbjeglo višestruko reproduciranje zvuka. Ova funkcionalnost je ključna kako bi osigurali da uvijek imamo samo jednu aktivnu reprodukciju u svakom trenutku. Također, zadržavamo informacije o trenutnom vremenu kako bismo omogućili nastavak reprodukcije od iste točke.

```tsx
const startStreaming = async () => {
  try {
    let audioInput: Tone.ToneAudioNode | null = null;

    if (sourceType === "usermedia") {
      const newUserMedia = new Tone.UserMedia({});
      await newUserMedia.open();
      Tone.start();
      audioInput = newUserMedia;
      setToneUserMedia(newUserMedia);
    } else if (sourceType === "file" && audioSrc) {
      const player = new Tone.Player(audioSrc, () => {
        const elapsedTime = Tone.now() - lastPausedTime;
        const startPosition = currentPosition + elapsedTime;
        player.start(0, startPosition);
      });
      player.loop = true;
      audioInput = player;
      setAudioPlayer(player);
    }

    if (audioInput) {
      state.effects
        .filter((effect) => effect.enabled)
        .forEach((effect) => {
          audioInput = connectEffect(audioInput!, effect);
        });

      const fft = new Tone.FFT({ size: 128 });
      audioInput.connect(fft);
      audioInput.toDestination();

      const updateLevels = () => {
        const values = fft.getValue();
        setLevels(values);
        requestAnimationFrame(updateLevels);
      };
      requestAnimationFrame(updateLevels);
    }
  } catch (err) {
    console.error("Error in startStreaming:", err);
  }
};
```

Slika 4.3 src/context/EffectContext.tsx - startStreaming funkcija

Funkcija startStreaming (Slika 4.3) pokreće reprodukciju ovisno o odabranom izvoru (sourceType). Prije reprodukcije, svi odabrani efekti se primjenjuju na izvor zvuka kako bi se integrirali u reprodukciju. Nakon toga, zvuk se preusmjerava na odredište. Fourierove transformacije (FFT) se koriste za dobivanje frekvencijskih podataka audio signala, koji se zatim koriste za generiranje razine signala u određenim rasponima frekvencija (stanje „levels“). Ova informacija koristi se za kasniji vizualni prikaz reprodukcije. Korištenje useReducer hook-a omogućava nam složeno upravljanje stanjem, što je posebno korisno u ovakvim scenarijima.

4.3 EffectReducer.ts i initial-state.ts

Ove dvije datoteke ključne su za inicijalizaciju stanja i upravljanje promjenama globalnog stanja liste efekata u našoj aplikaciji. Korištene su u liniji:

const [state, dispatch] = useReducer(effectReducer, initialState);

Zbog složenosti upravljanja ovim stanjem, korištenje useReducer hook-a predstavlja znatno bolje rješenje od upotrebe useState-a. Funkcija effectReducer prima trenutno stanje i akciju te na temelju njih vraća novo stanje. S druge strane, initialState predstavlja početno stanje efekata.

Razdvajanje ovih logičkih dijelova u zasebne datoteke doprinosi boljoj organizaciji i čitljivosti koda.

4.3.1 EffectReducer.ts

U našoj aplikaciji postoje četiri načina za utjecanje na globalno stanje liste efekata, za što je kreiran tip "AudioAction" (Slika 4.4). Svaki slučaj je definiran kao objekt koji sadrži tip akcije i objekt "payload" koji sadrži podatke potrebne za tu promjenu. Moguće akcije su TOGGLE_EFFECT, MOVE_EFFECT, EDIT_EFFECT i REORDER.

```ts
export type AudioAction =
  | { type: "TOGGLE_EFFECT"; payload: { name: EffectName } }
  | { type: "MOVE_EFFECT"; payload: { name: EffectName; direction: Direction } }
  | {
      type: "EDIT_EFFECT";
      payload: { name: EffectName; params: Record<string, any> };
    }
  | { type: "REORDER"; payload: { newOrder: EffectState } };
```

Slika 4.4 src/types/audio.ts - definicija AudioAction tipa

Korištenje useReducer hook-a s effectReducer i initialState omogućava složeno upravljanje stanjem efekata na modularan i čitljiv način. Definiranjem tipa AudioAction i korištenjem tog tipa za akcije unutar effectReducer funkcije, TypeScript nam automatski nudi sve moguće slučajeve kada koristimo switch za tip akcije. Ovo značajno poboljšava iskustvo razvijanja, jer omogućuje brzu identifikaciju svih mogućih akcija i osigurava da su sve akcije pravilno obrađene.

Funkcija effectReducer (Slika 4.5) upravlja promjenama stanja liste efekata na temelju primljenih akcija. Definirana je kao funkcija koja prima trenutno stanje (state) i akciju (action), te vraća novo stanje.

```ts
export const effectReducer = (state: EffectState, action: AudioAction) => {
  switch (action.type) {
    case "EDIT_EFFECT":
      const newState = {
        ...state,
        effects: state.effects.map((effect) =>
          effect.name === action.payload.name
            ? { ...effect, params: action.payload.params }
            : effect
        ),
      };
      return newState;

    case "MOVE_EFFECT":
    //Kod za pomicanje efekta naprijed/natrag u listi ovisno o Direction propu

    case "TOGGLE_EFFECT":
    //Kod za promjenu propa enabled na odabranom efektu

    case "REORDER":
    //Kod za promjenu redoslijeda efekata određen argumentom

    default:
      return state;
  }
};
```

Slika 4.5 src/utils/EffectReducer.ts

Unutar funkcije, koristeći switch izjavu, identificira se tip akcije kako bi se primijenile odgovarajuće promjene na stanje:

- TOGGLE_EFFECT: Pronalazi efekt s imenom iz argumenta action. Mijenja njegov status "enabled" na suprotni (true postaje false i obrnuto).
- MOVE_EFFECT: Pronalazi indeks efekta koji treba biti pomaknut. Izračunava novi indeks na temelju smjera (gore ili dolje). Ako je novi indeks valjan, efekt se premješta na novu poziciju unutar liste.
- EDIT_EFFECT: Pronalazi efekt s imenom iz argumenta action. Ažurira njegove parametara s onima navedenima u akciji.
- REORDER: Postavlja novi poredak na predanu vrijednost

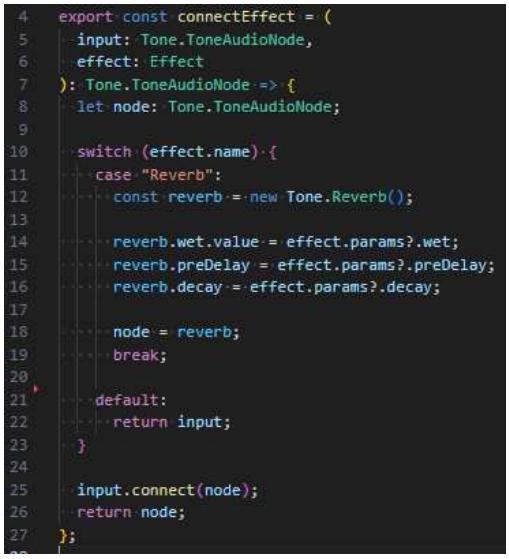
Korištenjem useReducer umjesto useState omogućava se bolje upravljanje složenim promjenama stanja, čime se poboljšava održavanje i proširivost koda.

4.3.2 initial-state.ts

U datoteci se nalazi početno stanje liste efekata za našu aplikaciju. Svaki efekt ima definirano ime, a njegov status "enabled" je inicijalno postavljen na false, što znači da su svi efekti na početku isključeni. Osim toga, svaki efekt ima definirane početne vrijednosti svojih parametara, koje su specifične za taj efekt. Ova inicijalna konfiguracija omogućava da se aplikacija pokrene s jasno definiranim stanjem efekata, koje se kasnije može mijenjati kroz interakcije korisnika.

4.4 connectEffect.ts

Nakon postavljanja početnog stanja efekata, potrebno je omogućiti korisnicima dodavanje efekata, koji se povezuju na izvor zvuka i šalju konačno uređen zvuk na izlaz. Poredak efekata u listi je izuzetno bitan jer direktno utječe na konačni zvuk, pa je ključno očuvati ovaj poredak pri spajanju efekata. Funkcija connectEffect (Slika 4.6) je zadužena za postizanje tog cilja.

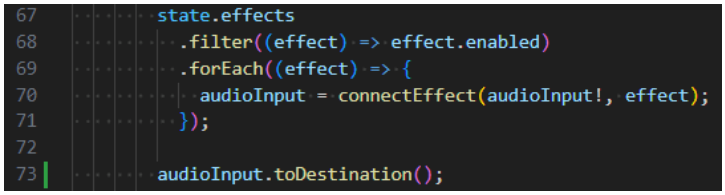


```
export const connectEffect = (
  input: Tone.ToneAudioNode,
  effect: Effect
): Tone.ToneAudioNode => {
  let node: Tone.ToneAudioNode;

  switch (effect.name) {
    case "Reverb":
      const reverb = new Tone.Reverb();

      reverb.wet.value = effect.params?.wet;
      reverb.preDelay = effect.params?.preDelay;
      reverb.decay = effect.params?.decay;

      node = reverb;
      break;

    default:
      return input;
  }

  input.connect(node);
  return node;
};
```

Slika 4.6 src/utils/connectEffect.ts

Funkcija connectEffect prima dva argumenta: input i effect. Ovdje su objašnjeni njeni koraci:

- Kreiranje novog ToneAudioNode objekta: Ovisno o tipu efekta, funkcija kreira odgovarajući audio node iz biblioteke Tone.js.
- Podešavanje parametara: Parametri efekta se postavljaju na vrijednosti proslijeđene u argumentu. Na primjer, za efekt tipa "Reverb", postavljaju se parametri kao što su wet, preDelay, i decay.
- Spajanje node-a na input: Novi node se povezuje sa prethodnim inputom, čime se osigurava da su svi efekti spojeni u ispravnom redoslijedu.
- Vraćanje novog node-a: Nakon što je efekt povezan, funkcija vraća novi node, koji postaje input za sljedeći efekt u listi.



```
state.effects
  .filter((effect) => effect.enabled)
  .forEach((effect) => {
    audioInput = connectEffect(audioInput!, effect);
  });

audioInput.toDestination();
```

Slika 4.7 src/context/EffectContext.tsx - isječak koda koji je zadužen za ulančano spajanje efekata

Unutar datoteke EffectContext.tsx, funkcija connectEffect se koristi kako bi se svi aktivni efekti u listi povezali u ispravnom redoslijedu (Slika 4.7). To se postiže iteracijom kroz sve efekte koji su omogućeni (enabled) i pozivanjem connectEffect za svaki od njih. Na kraju, svi povezani efekti se šalju na izlaz (toDestination).

4.5 EffectList.tsx i EffectContainer.tsx

Ove dvije komponente omogućuju korisniku da upravlja efektima (Slika 4.8). Jedna koja služi za određivanje poretka i uključenosti efekata, a unutar druge se uključenim efektima mogu konfigurirati parametri.

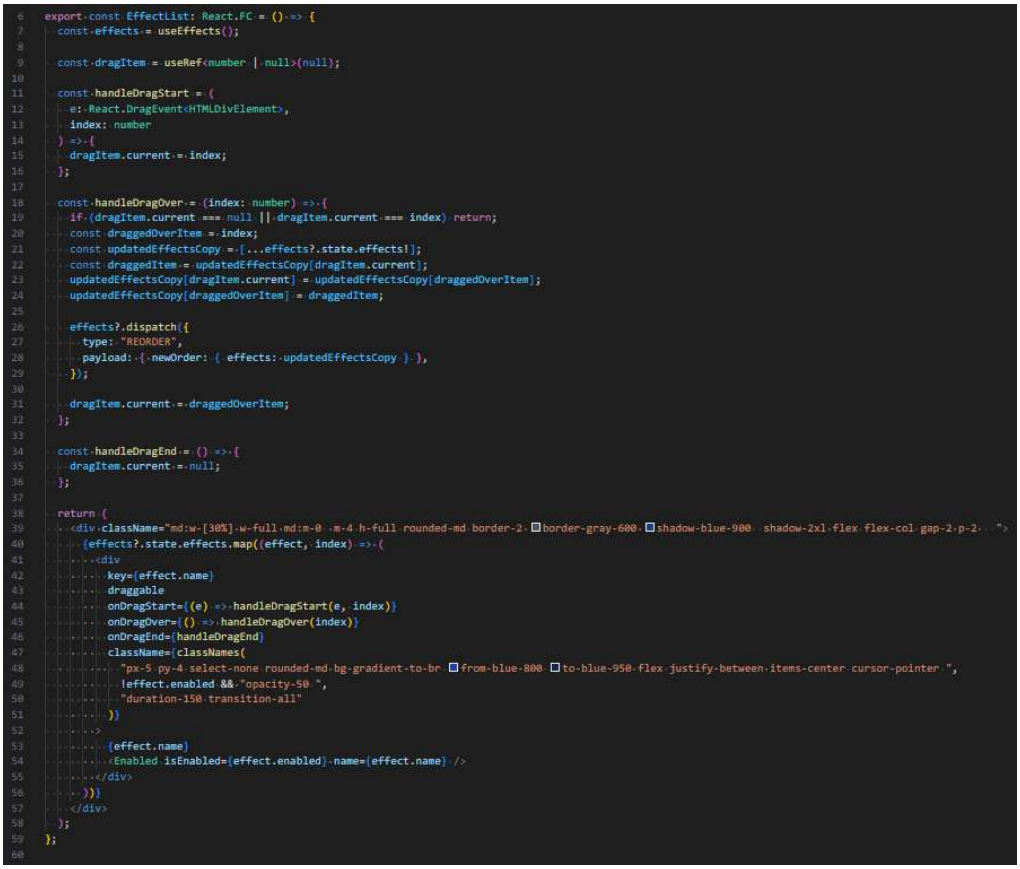
Slika 4.8 Korisničko sučelje

4.5.1 EffectList.tsx

Komponenta EffectList (Slika 4.9) je osmišljena kako bi prikazala listu audio efekata koje korisnik može koristiti, svaki s vlastitim imenom i indikatorom koji označava je li efekt uključen ili isključen. Njen je cilj omogućiti korisnicima jednostavan pregled i manipulaciju redoslijedom efekata putem funkcionalnosti "drag and drop".

Funkcionalnosti komponente:

- Prikaz liste efekata: Komponenta prikazuje sve efekte u listi, svaki ispod prethodnog. Svaki element liste sadrži ime efekta i indikator (kvadrat) koji pokazuje status efekta.
- Indikator statusa: Ako je efekt uključen (enabled je true), indikator je ispunjen bojom, a cijeli element liste je vidljiv s punom neprozirnošću (opacity: 100%). Ako je efekt isključen (enabled je false), indikator je prazan, a cijeli element liste ima smanjenu neprozirnost (opacity: 50%). Klikom na indikator, status se mijenja.
- Promjena redoslijeda efekata: Korištenjem useRef i onDrag događaja koje pružaju React i JSX, omogućena je funkcionalnost promjene redoslijeda efekata kroz "drag and drop". Korisnici mogu povući i premjestiti efekte kako bi promijenili njihov redoslijed u listi.



```tsx
export const EffectList: React.FC = () => {
  const effects = useEffects();

  const dragItem = useRef<number | null>(null);

  const handleDragStart = (
    e: React.DragEvent<HTMLDivElement>,
    index: number
  ) => {
    dragItem.current = index;
  };

  const handleDragOver = (index: number) => {
    if (dragItem.current === null || dragItem.current === index) return;
    const draggedOverItem = index;
    const updatedEffectsCopy = [...effects?.state.effects!];
    const draggedItem = updatedEffectsCopy[dragItem.current];
    updatedEffectsCopy[dragItem.current] = updatedEffectsCopy[draggedOverItem];
    updatedEffectsCopy[draggedOverItem] = draggedItem;

    effects?.dispatch({
      type: "REORDER",
      payload: { newOrder: { effects: updatedEffectsCopy } },
    });

    dragItem.current = draggedOverItem;
  };

  const handleDragEnd = () => {
    dragItem.current = null;
  };

  return (
    <div className="md:w-[30%] w-full md:m-0 m-4 h-full rounded-md border-2 border-gray-600 shadow-blue-900 shadow-2xl flex flex-col gap-2 p-2 ">
      {effects?.state.effects.map((effect, index) => (
        <div
          key={effect.name}
          draggable
          onDragStart={(e) => handleDragStart(e, index)}
          onDragOver={() => handleDragOver(index)}
          onDragEnd={handleDragEnd}
          className={classNames(
            "px-5 py-4 select-none rounded-md bg-gradient-to-br from-blue-800 to-blue-950 flex justify-between items-center cursor-pointer ",
            !effect.enabled && "opacity-50 ",
            "duration-150 transition-all"
          )}
        >
          {effect.name}
          <Enabled isEnabled={effect.enabled} name={effect.name} />
        </div>
      ))}
    </div>
  );
};

```

Slika 4.9 src/components/organisms/EffectList.tsx

Detaljniji opis funkcionalnosti:

- useEffects: Koristi se za dobivanje trenutnog stanja efekata i dispatch metode iz konteksta.
- dragItem: useRef hook koristi se za praćenje indeksa efekta koji se trenutno vuče.
- handleDragStart: Funkcija koja se poziva kada korisnik započne vučenje efekta. Postavlja indeks trenutnog efekta koji se vuče.
- handleDragOver: Funkcija koja se poziva kada korisnik vuče efekt preko drugog efekta. Ažurira redoslijed efekata i šalje novu listu efekata putem dispatch metode.
- handleDragEnd: Funkcija koja se poziva kada korisnik završi vučenje efekta. Resetira trenutni indeks vučenog efekta.

4.5.2 EffectContainer.tsx

Komponenta EffectContainer (Slika 4.10) predstavlja sučelje za manipulaciju efektima unutar aplikacije. Ona omogućuje korisnicima pregled i interakciju s aktivnim efektima, prikazujući komponente za svaki efekt čiji je status "enabled" postavljen na true. Unutar mape /components/effects nalaze se zasebne komponente za svaki efekt, pružajući modularan pristup razvoju i održavanju aplikacije.

```
return (
  <div
    className="custom-scrollbar md:w-[70%] w-full h-[395px] md:m-0 m-4 overflow-y-auto rounded-md border-2 border-gray-600 shadow-blue-900
      shadow-2xl flex flex-col gap-2 p-2 text-sm "
  >
    {effects?.state.effects
      .filter((effect) => effect.enabled)
      .map((effect) => (
        <div key={effect.name}>{EffectRecord[effect.name]}</div>
      ))}
  </div>
);
};
```

Slika 4.10 src/components/organisms/EffectContainer.tsx

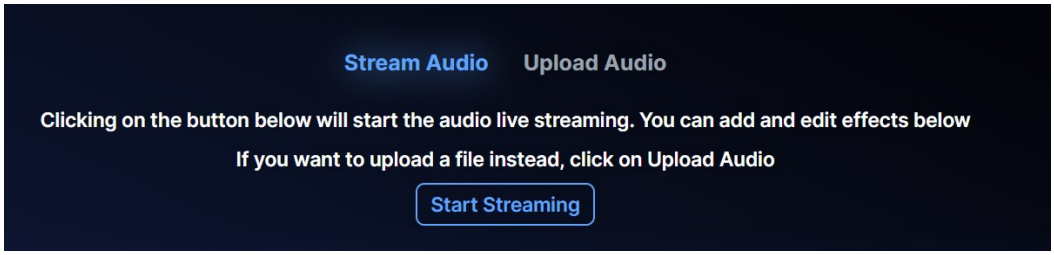
Ključna datoteka unutar ovog sustava je effect-record.tsx (Slika 4.11), koji predstavlja mapu u kojoj se za svako moguće ime efekta mora imati pridruženu komponentu. Unutar EffectContainer, za svaki efekt prikazuje se njegova komponenta. Ovo osigurava dosljednost u implementaciji i sprječava nastanak grešaka pri dodavanju novih efekata. Ukoliko za neki efekt nedostaje odgovarajuća komponenta, sustav će generirati grešku, što olakšava održavanje i razvoj aplikacije.

```
export const EffectRecord: Record<EffectName, ReactNode> = {
  Delay: <DelayEffect key="delay" />,
  Reverb: <ReverbEffect key="reverb" />,
  Distortion: <DistortionEffect key="distortion" />,
  Filter: <FilterEffect key="phaser" />,
  Tremolo: <TremoloEffect key="tremolo" />,
  Chorus: <ChorusEffect key="chorus" />,
};
```

Slika 4.11 src/components/effects/effect-record.tsx

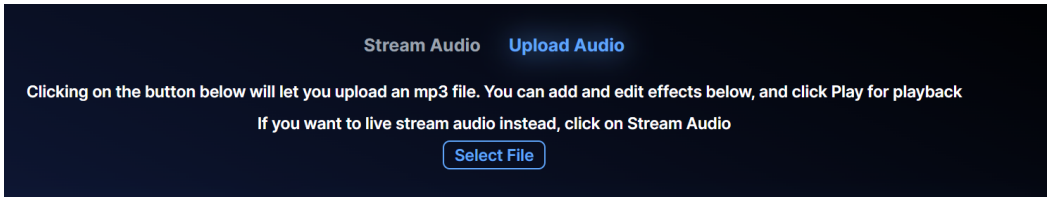
4.6 Postavke reprodukcije

Korisnicima aplikacije omogućeno je pokretanje reprodukcije zvuka s dvije opcije: prijenos zvuka uživo s mikrofona ili prijenos audio datoteke koja se potom reproducira.



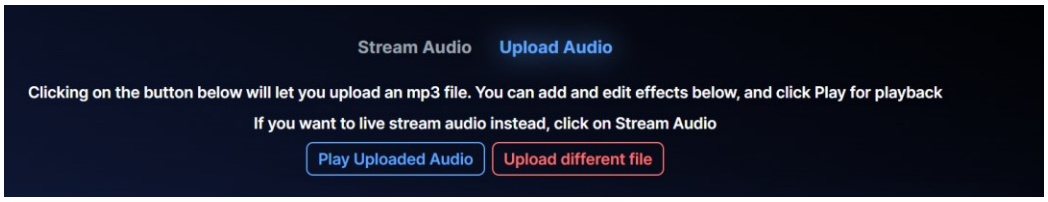
Slika 4.12 Isječak korisničkog sučelja - opcija za reprodukciju zvuka uživo s mikrofona

Kad korisnik odabere opciju "Stream Audio" (Slika 4.12), prikazuje mu se informativni tekst koji ukratko objašnjava kako koristiti aplikaciju za prijenos zvuka uživo. Ispod tog teksta nalazi se gumb za pokretanje prijenosa. Ova opcija omogućava korisnicima da koriste mikrofon svog uređaja za prijenos zvuka u realnom vremenu.



Slika 4.13 Isječak korisničkog sučelja - opcija za prijenos audio datoteke

Ako korisnik odabere opciju "Upload Audio" (Slika 4.13), prikazuje mu se sličan informativni tekst. Ispod tog teksta nalazi se gumb za prijenos audio datoteke s uređaja korisnika. Nakon uspješnog prijenosa datoteke, korisniku se nude dodatni gumbi za reprodukciju prenesene datoteke i za prijenos nove datoteke (Slika 4.14).



Slika 4.14 Izgled korisničkog sučelja nakon prenesene datoteke

Prikaz sadržaja i navigacija između različitih opcija za reprodukciju ostvareni su korištenjem komponente PaginatedSection (Slika 4.15). Ova komponenta prima listu tabova kao prop (Slika 4.16). Svaki tab ima svoj naziv i ReactNode, koji predstavlja odgovarajuću komponentu za taj tab. Korisniku se prikazuju nazivi svih tabova, a klikom na neki od naziva, prikazuje se sadržaj vezan uz odabrani tab. Ovakav način organizacije omogućuje jednostavnu i intuitivnu navigaciju kroz različite opcije reprodukcije zvuka.

```
export const PaginatedSection: FC<IPaginatedSectionProps> = ({ tabs }) => {
  const [selectedTab, setSelectedTab] = useState<Tab>(tabs[0]);
  return (
    <div className="w-full ">
      <div className="flex items-center md:w-[1200px] w-full justify-center gap-2 border-gray-600 ">
        {tabs.map((tab) => (
          <div
            className="flex relative items-center justify-center"
            key={tab.title}
          >
            <p
              onClick={() => setSelectedTab(tab)}
              className={classNames(
                tab.title == selectedTab.title
                  ? "text-blue-400"
                  : "text-gray-400",
                "px-3 py-1 flex items-center justify-center font-bold text-xl z-10  cursor-pointer hover:text-blue-200 transition-all duration-100"
              )}
            >
              {tab.title}
            </p>
            {selectedTab.title == tab.title && (
              <p className="text-blue-400 absolute blur-lg font-bold text-xl">
                {tab.title}
              </p>
            )}
          </div>
        ))}
      </div>
      <div className="h-40">{selectedTab.tab}</div>
    </div>
  );
};
```

Slika 4.15 src/components/atoms/PaginatedSection.tsx

```tsx
export const StreamingSettings: FC = () => {
  return (
    <div>
      <PaginatedSection
        tabs={[
          {
            title: "Stream Audio",
            tab: <StreamAudio />,
          },
          {
            title: "Upload Audio",
            tab: <UploadAudio />,
          },
        ]}
      />
    </div>
  );
};
```

Slika 4.16 Korištenje komponente PaginatedSection

Unutar StreamAudio komponente (Slika 4.17) implementirana je funkcionalnost koja omogućava korisnicima pokretanje reprodukcije zvuka uživo. Ova komponenta koristi dva hooka: useStreaming i useEffects, koji služe za upravljanje kontekstima aplikacije.

Korištenjem useEffect hooka, osigurava se da se prilikom učitavanja komponente postavlja SourceType na "usermedia", što omogućava aplikaciji da zna kako postupiti prilikom započinjanja prijenosa zvuka s mikrofona. Također, postavljamo isStreaming na false i poziciju audio datoteke na 0, kako bismo osigurali da se prilikom promjene taba s "Upload Audio" na "Stream Audio" resetiraju sve postavke.

```tsx
export const StreamAudio = () => {
  const { isStreaming, setIsStreaming, toggleIsStreaming } = useStreaming();
  const { setSourceType, setCurrentPosition, setLastPausedTime } = useEffects();

  useEffect(() => {
    setSourceType("usermedia");
    setIsStreaming(false);
    setCurrentPosition(0);
    setLastPausedTime(0);
  }, []);

  return (
    <div className="w-full flex justify-center items-center flex-col text-white text-lg font-semibold p-5 gap-2">
      <p>
        Clicking on the button below will start the audio live streaming. You
        can add and edit effects below
      </p>
      <p>If you want to upload a file instead, click on Upload Audio</p>
      <div className="flex items-center justify-center gap-2">
        <Button
          actionType="regular"
          label={isStreaming ? "Stop Streaming" : "Start Streaming"}
          className={classNames(isStreaming && "animate-pulse")}
          onClick={() => {
            toggleIsStreaming();
          }}
        />
      </div>
    </div>
  );
};
```

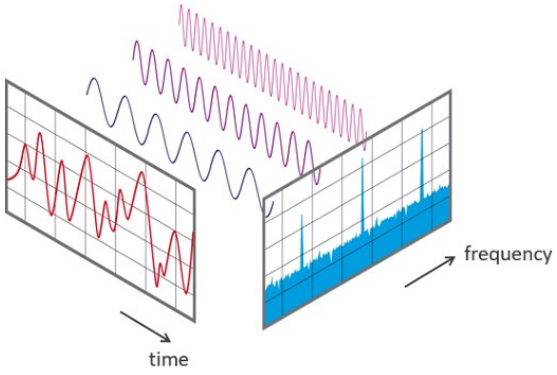
Slika 4.17 src/components/molecules/StreamAudio.tsx

4.7 Vizualizator

Ispod efekata u aplikaciji nalazi se vizualizator spektra frekvencija (Slika 4.18). Ovaj vizualizator dinamički mijenja duljinu linija ovisno o promjenama u spektru frekvencija audio inputa, pružajući korisniku vizualni prikaz kako efekti utječu na zvuk.

Slika 4.18 Korisničko sučelje: prikaz vizualizatora

FFT (Fast Fourier Transform) je algoritam koji se koristi za brzo računanje Fourierove transformacije i njezine inverzne transformacije. Fourierova transformacija je matematička metoda koja razlaže složene signale na njihove sastavne sinusoide, odnosno frekvencije. To znači da FFT može uzeti vremenski signal (poput zvučnog vala) i pretvoriti ga u frekvencijski spektar, što omogućava analizu amplituda različitih frekvencija unutar signala (Slika 4.19). [7]



Slika 4.19 Prikaz signala u vremenskoj i frekvencijskoj domeni

U aplikaciji koristimo „Tone.FFT“ iz biblioteke Tone.js za analizu audio inputa i prikaz spektra frekvencija u realnom vremenu (Slika 4.20). Kreiramo novi FFT objekt s veličinom (brojem uzoraka) od 128. Manja veličina rezultira bržim računanjem ali manjom razlučivosti u

frekvencijskom spektru, dok veća veličina pruža veću preciznost ali zahtijeva više računalne snage. Audio input se povezuje s FFT-om kako bi se analizirali zvučni podaci.

```
73        const fft = new Tone.FFT({ size: 128 });
74        audioInput.connect(fft);
75        audioInput.toDestination();
76
77        const updateLevels = () => {
78          const values = fft.getValue();
79          setLevels(values);
80          requestAnimationFrame(updateLevels);
81        };
82        requestAnimationFrame(updateLevels);
```

Slika 4.20 src/context/EffectContext.tsx – isječak koda u kojem se pomoću FFT-ja u stvarnom vremenu mijenja lista "values"

Funkcija updateLevels dohvaća trenutne vrijednosti frekvencijskog spektra pomoću fft.getValue(), te te vrijednosti postavlja u stanje komponente pomoću setLevels(values). To znači da je u stanje levels spremljena lista od 128 brojeva, koji označavaju amplitudu svakog raspona frekvencija. requestAnimationFrame(updateLevels) osigurava da se funkcija ponovo poziva pri svakom osvježavanju ekrana, što omogućava kontinuirano ažuriranje vizualizatora.

```
4   export const Visualizer = () => {
5     const { levels } = useEffects();
6     const normalizeLevel = (level: number) => {
7       const minLevel = -100;
8       const maxLevel = 0;
9       return ((level - minLevel) / (maxLevel - minLevel)) * 100 + 40;
10    };
11    return (
12      <div className="text-sm h-[200px] w-full text-white flex justify-center items-start mt-5 ">
13        {levels &&
14          Array.from(levels).map((level, index) => (
15            <div
16              key={index}
17              className={`md:w-2 w-1 bg-blue-600 shadow-blue-600 blur-[2px] rounded-full shadow-2xl `}
18              style={{ height: normalizeLevel(level) + "%" }}
19            >
20              <p className="text-xs"></p>
21            </div>
22          ))}
23      </div>
24    );
25  };
```

Slika 4.21 src/components/organisms/Visualizer.tsx - komponenta za prikazivanje vizualizatora

U komponenti Visualizer (Slika 4.21) prikazujemo 128 linija koje su spremljene u stanju levels. Ove linije predstavljaju amplitude različitih frekvencijskih komponenti koje se dobiju korištenjem FFT algoritma na audio inputu. Koristimo useEffects hook za pristup stanju levels, koje sadrži 128 vrijednosti amplituda različitih frekvencijskih komponenti dobivenih FFT algoritmom. Funkcija normalizeLevel pretvara vrijednosti nivoa iz domene između -100 dB i 0 dB u postotke između 40% i 140%. Ovo osigurava da su linije vizualizatora dovoljno visoke za prikaz. U povratnoj funkciji return, koristimo Array.from(levels).map za iteriranje kroz levels i stvaranje div elemenata koji predstavljaju linije vizualizatora. Svaka linija dobiva visinu koja odgovara normaliziranoj vrijednosti nivoa, boju i stil za vizualni efekt.Ovaj kod prikazuje frekvencijski spektar audio signala pomoću 128 vertikalnih linija, gdje visina svake linije odgovara amplitudi određene frekvencijske komponente.

4.8 Efekti

U ovom potpoglavlju detaljnije su opisani efekti te njihova implementacija.

4.8.1 Izrada pomoćnih komponenti

U aplikaciji za upravljanje audio efektima, tri ključne komponente su Select, Knob, i EffectBox. Svaka od ovih komponenti ima specifičnu ulogu u omogućavanju korisnicima da pregledavaju, odabiru i prilagođavaju različite audio efekte.

Select komponenta (Slika 4.22) omogućuje korisniku odabir opcije iz popisa. To je jednostavna komponenta koja prikazuje dostupne opcije i omogućuje korisniku da odabere jednu od njih. Korisnik vidi sve opcije koje su dostupne, a kada klikne na određenu opciju, ta se opcija postavlja kao odabrana. Roditeljska komponenta se obavještava o promjeni putem onChange funkcije. Odabrana opcija vizualno se razlikuje od neodabranih opcija, što korisniku olakšava pregled trenutnog odabira.

```tsx
interface ISelectProps {
  label: string;
  options: string[];
  onChange: (value: string) => void;
}

export const Select: FC<ISelectProps> = ({ label, onChange, options }) => {
  const [selectedOption, setSelectedOption] = useState<string>(options[0]);

  useEffect(() => {
    onChange(selectedOption);
  }, [selectedOption]);

  return (
    <div className="flex flex-col gap-1">
      <div>
        {options.map((option) => (
          <div
            key={option}
            onClick={() => {
              setSelectedOption(option);
            }}
            className={classNames(
              selectedOption === option ? "bg-gray-950" : "bg-gray-800",
              "cursor-pointer p-3 border-[1px] border-gray-500 w-min"
            )}
          >
            {option}
          </div>
        ))}
        <p>{label}</p>
      </div>
    </div>
  );
};
```

Slika 4.22 src/components/atoms/Select.tsx

Knob komponenta (Slika 4.23, Slika 4.24) služi za postavljanje parametara audio efekata. Može funkcionirati na dva načina: kada se komponenti predaju minimalna i maksimalna vrijednost, korisnik može prilagoditi parametar unutar tog raspona, ili kada se komponenti predaje lista opcija, korisnik može birati među nekoliko fiksnih vrijednosti.

```tsx
const [angle, setAngle] = useState<number>(0);
const startYRef = useRef<number>(0);
const startAngleRef = useRef<number>(0);
const knobRef = useRef<HTMLDivElement>(null);

const getValueFromAngle = (angle: number) => {
  if ("minValue" in props) {
    const range = props.maxValue - props.minValue;
    return props.minValue + ((angle + 120) / 240) * range;
  } else {
    const optionCount = props.options.length;
    const index = Math.round(((angle + 120) / 240) * (optionCount - 1));
    return props.options[index];
  }
};

const handleMouseMove = (event: MouseEvent) => {
  const deltaY = startYRef.current - event.clientY;
  let newAngle = startAngleRef.current + deltaY;
  if (newAngle > 120) newAngle = 120;
  if (newAngle < -120) newAngle = -120;
  setAngle(newAngle);
  const newValue = getValueFromAngle(newAngle);
  setValue(newValue);
  props.setValue(newValue);
};

const handleMouseUp = () => {
  document.removeEventListener("mousemove", handleMouseMove);
  document.removeEventListener("mouseup", handleMouseUp);
};

const handleMouseDown = (event: React.MouseEvent) => {
  startYRef.current = event.clientY;
  startAngleRef.current = angle;
  document.addEventListener("mousemove", handleMouseMove);
  document.addEventListener("mouseup", handleMouseUp);
};

useEffect(() => {
  if ("minValue" in props) {
    setAngle(
      ((value - props.minValue) / (props.maxValue - props.minValue)) * 240 -
        120
    );
  } else {
    const index = props.options.indexOf(value);
    setAngle((index / (props.options.length - 1)) * 240 - 120);
  }
}, [value, props]);
```

Slika 4.23 src/components/atoms/Knob.tsx - stanja, funkcije i useEffect

U oba slučaja, vrijednost se podešava na principu "dragganja" (povlačenja). Pomicanjem kursora prema gore povećava se vrijednost (okreće se udesno), dok pomicanjem prema dolje smanjuje vrijednost (okreće se ulijevo). Kroz promjenu kuta okretanja ručice, vrijednost se dinamički ažurira i prenosi roditeljskoj komponenti. Komponenta je stilizirana tako da jasno prikazuje trenutnu vrijednost i omogućuje intuitivno podešavanje.

```
return (
  <div className="flex flex-col items-center justify-center w-max h-max select-none">
    <p className="text-[12px]">{props.label}</p>
    <div
      ref={knobRef}
      onMouseDown={handleMouseDown}
      className=" rounded-full bg-gradient-to-tl from-gray-500 to-gray-700 h-14 w-14 flex items-center justify-center cursor-pointer"
      style={{ transform: `rotate(${angle}deg)` }}
    >
      <div className="rounded-full w-11 h-11 bg-gray-700 relative flex items-start justify-center ">
        <div className="absolute w-1 h-6 bg-gradient-to-b from-blue-700 to-blue-400"></div>
      </div>
    </div>
    <p className=" text-white">
      {"minValue" in props ? value.toFixed(1) : value.toFixed(2)}
    </p>
  </div>
);
};
```

Slika 4.24 src/components/atoms/Knob.tsx – Izgled

EffectBox (Slika 4.25, Slika 4.26) je "container" komponenta koja služi kao zajednički okvir za sve efekt komponente. Njena glavna svrha je osigurati da svi efekti unutar aplikacije imaju konzistentan izgled i osjećaj, što smanjuje potrebu za ponavljanjem koda. Komponenta sadrži gumb koji, kada se klikne, aktivira dispatch funkciju iz useEffects konteksta s akcijom EDIT_EFFECT. Kroz props se prenose ime efekta i parametri koji se trebaju primijeniti, omogućujući jednostavno i učinkovito upravljanje promjenama efekata.

```
interface IEffectBoxProps {
  children: ReactNode;
  effectName: EffectName;
  params: Object;
}

export const EffectBox: FC<IEffectBoxProps> = ({
  children,
  effectName,
  params,
}) => {
  const { dispatch } = useEffects();

  const handleChange = () => {
    dispatch({
      type: "EDIT_EFFECT",
      payload: { name: effectName, params: params },
    });
  };

  return (
    <div className="w-full h-full relative rounded-md border-[1px] border-blue-400 p-5 px-10 flex gap-10 justify-between items-center ">
      <div className="flex gap-10 items-center justify-center ">{children}</div>
      <Button
        className="w-max"
        label="Apply Changes"
        actionType="regular"
        onClick={handleChange}
      />
      <div className="absolute bottom-0 right-0 p-2">
        <p className="text-xs font-bold opacity-30 capitalize ">{effectName}</p>
      </div>
    </div>
  );
};

```

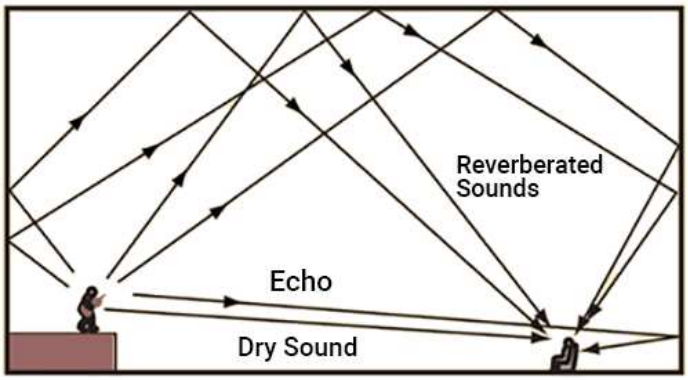
Slika 4.25 src/components/molecules/EffectBox.tsx

```tsx
export const DelayEffect = () => {
  const [delayParams, setDelayParams] = useState<DelayParams>({
    delayTime: 0.5,
    wet: 0.5,
  });

  return (
    <EffectBox effectName="Delay" params={delayParams}>
      <Knob
        label="Wet"
        maxValue={100}
        minValue={0}
        setValue={(value) =>
          setDelayParams({ ...delayParams, wet: value / 100 })
        }
      />
      <Knob
        label="Delay Time"
        options={[1, 1 / 2, 1 / 4, 1 / 8, 1 / 16]}
        setValue={(value) =>
          setDelayParams({ ...delayParams, delayTime: value })
        }
      />
    </EffectBox>
```

Slika 4.26 Korištenje komponente EffectBox na primjeru efekta Delay

4.8.2 Reverb

Reverb, skraćenica od reverberation, je audio efekt koji simulira odjek zvuka u prostoru. Kada zvuk udari o površine u prostoru, poput zidova, poda ili stropa, odbija se i stvara reverberaciju (Slika 4.27). Ovaj efekt dodaje dubinu i prostornu dimenziju zvuku, čineći ga bogatijim i prirodnijim. [9]



Slika 4.27 Vizualni prikaz audio efekta "Reverb"

U aplikaciji, komponenta ReverbEffect omogućuje korisniku podešavanje parametara reverb efekta (Slika 4.28). Tri ključna parametra koje aplikacija mijenja su:

- Wet: Ovaj parametar kontrolira količinu primijenjenog reverb efekta na originalni zvuk. Vrijednost wet parametra predstavlja postotak primijenjenog efekta u odnosu na originalni zvuk.
- Decay: Parametar decay kontrolira brzinu kojom reverb efekt slabi s vremenom. To znači koliko brzo odjek zvuka nestaje nakon što je izvorni zvuk prestao. Ovaj parametar omogućuje korisniku prilagodbu trajanja reverberacije.
- Pre-delay: Pre-delay parametar predstavlja vremenski interval između originalnog zvuka i početka reverberacije. Kontrolira koliko vremena prođe prije nego što se čuje prvi odjek nakon originalnog zvuka. Pomoću ovog parametra korisnik može utjecati na osjećaj prostornosti u zvuku.

Slika 4.28 Komponenta ReverbEffect u korisničkom sučelju

Kroz te tri ključna parametra, korisnik može prilagoditi reverb efekt prema svojim željama i potrebama, dodajući dubinu i prostornost zvuku u skladu s karakteristikama prostora ili glazbenog djela.

4.8.3 Delay

Delay (kašnjenje) je audio efekt koji reprodukciju zvuka odgađa za određeni vremenski period nakon izvorne reprodukcije (Slika 4.29). Ovaj efekt se često koristi za dodavanje dubine i teksture zvuku, stvarajući efekt višestrukog odjeka ili ponavljanja zvuka. [10]

Slika 4.29 Vizualni prikaz efekta "Delay"

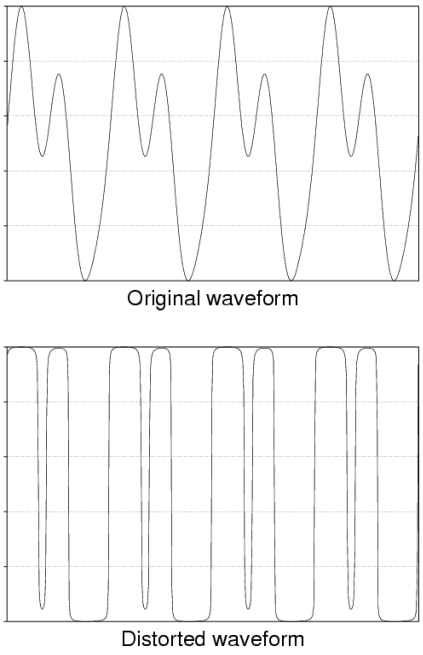
U aplikaciji za upravljanje audio efektima, komponenta DelayEffect omogućuje korisniku podešavanje parametara delay efekta (Slika 4.30). Dva ključna parametra koje aplikacija mijenja su:

- Wet: predstavlja postotak primijenjenog efekta u odnosu na originalni zvuk.
- Delay Time: Parametar delayTime određuje vremenski period za koji se odgađa reprodukcija zvuka. Korisnik može odabrati vrijednost delay vremena između različitih opcija, kao što su 1, 1/2, 1/4, 1/8 ili 1/16 sekunde. Ovaj parametar omogućuje korisniku prilagodbu duljine odgode reprodukcije zvuka.

Slika 4.30 Komponenta DelayEffect u korisničkom sučelju

4.8.4 Distorzija

Distorzija je audio efekt koji mijenja ton zvuka dodajući harmoničke komponente, što rezultira izobličenjem originalnog zvuka (Slika 4.31). Ovaj efekt je često korišten u glazbenoj produkciji kako bi se postigao specifičan zvuk gitare, basa ili drugih instrumenata, te dodao snaga, gustoća ili agresivnost zvuku. [11]



Slika 4.31 Vizualni prikaz efekta "Distortion"

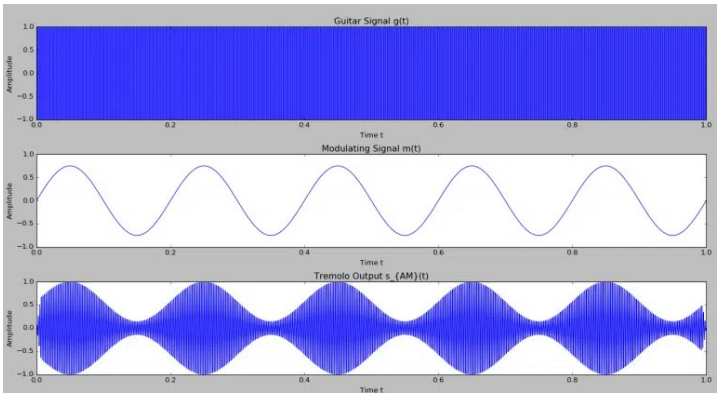
U komponenti DistortionEffect, omogućeno je prilagođavanje parametara distorzije (Slika 4.32) kroz sljedeće:

- Wet: Ovaj parametar kontrolira intenzitet primijenjenog distorzijskog efekta na originalni zvuk. Podešavanjem ovog parametra korisnik određuje koliko jakog distorzijskog efekta želi dodati izvornom zvuku.
- Oversample: Parametar "Oversample" omogućuje korisniku odabir razine oversamplinga, što može poboljšati kvalitetu distorzije. Korisnik može birati između opcija "none" (bez oversamplinga), "2x" i "4x", prilagođavajući razinu preciznosti distorzijskog efekta prema svojim preferencijama.

Slika 4.32 Komponenta DistortionEffect u korisničkom sučelju

4.8.5 Tremolo

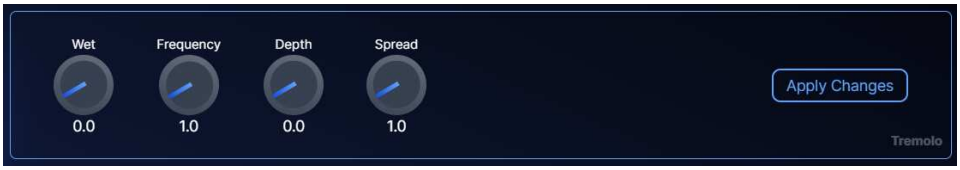
Tremolo je audio efekt koji periodično mijenja amplitudu zvuka, stvarajući karakterističan oscilirajući zvuk (Slika 4.33). Ovaj efekt se koristi u glazbi kako bi dodao dinamičku varijaciju i teksturu zvuku, pružajući tako dodatnu dimenziju izvedbi. U digitalnoj obradi zvuka, tremolo se često koristi kao dio audio efekata radi postizanja željenog zvučnog stila ili atmosfere. [12]



Slika 4.33 Vizualni prikaz efekta "Tremolo"

U komponenti TremoloEffect, korisnik može prilagoditi različite parametre tremolo efekta (Slika 4.8) kako bi oblikovao zvuk prema svojim preferencijama:

- Wet: Ovaj parametar određuje jačinu primjene tremolo efekta na izvorni zvuk. Podešavanjem ovog parametra korisnik može kontrolirati intenzitet tremolo efekta.
- Frequency: Frekvencija određuje brzinu oscilacije amplitude zvuka. Viša vrijednost ovog parametra rezultira bržim oscilacijama, dok niže vrijednosti rezultiraju sporijim oscilacijama.
- Depth: Dubina (Depth) određuje maksimalnu promjenu amplitude zvuka tijekom oscilacija. Povećanje ovog parametra rezultira većim varijacijama u amplitudi zvuka.
- Spread: Ovaj parametar određuje raspon (u stupnjevima) unutar kojeg se tremolo efekt širi po stereo polju. Veće vrijednosti ovog parametra rezultiraju širim rasponom oscilacija, što može stvoriti bogatiji zvučni prostor.

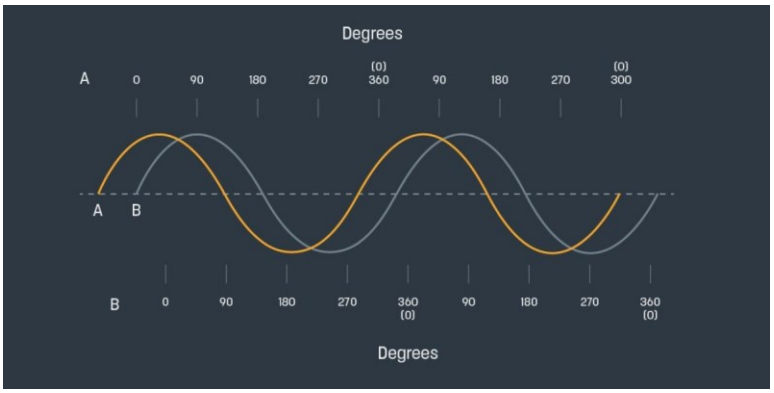


Slika 4.34 Komponenta TremoloEffect u korisničkom sučelju

Prilagođavanjem ovih parametara, korisnik može postići širok raspon zvučnih efekata, od suptilnih oscilacija do intenzivnih i dinamičnih promjena u zvuku. Također, primjena tremolo efekta u glazbenoj produkciji može dodati emotivnu i atmosfersku dubinu glazbi, stvarajući tako bogatije i slojevitije audio iskustvo.

4.8.6 Chorus

Chorus je audio efekt koji stvara iluziju višestrukih zvučnih izvora tako što reproducira originalni zvuk uz dodatak laganih varijacija u frekvenciji i vremenu (Slika 4.35). Ovaj efekt se koristi kako bi se postigao efekt širenja zvuka i stvaranja dubine, čime se stvara dojam da je zvuk širi i puniji. Chorus se često koristi u glazbenoj produkciji kako bi se dodala tekstura i bogatstvo zvuku, posebno na vokalima, gitarama i klavijaturama. [13]



Slika 4.35 Vizualni prikaz efekta "chorus"

U komponenti ChorusEffect, korisnik može prilagoditi različite parametre chorus efekta (Slika 4.36) kako bi oblikovao zvuk prema svojim preferencijama:

- Wet: Podešavanjem ovog parametra korisnik može kontrolirati intenzitet primjene chorus efekta.
- Depth: Dubina (Depth) određuje intenzitet varijacija u frekvenciji zvuka. Povećanje ovog parametra rezultira dubljim varijacijama, što može rezultirati širim zvučnim prostorom.
- Frequency: Frekvencija određuje brzinu varijacija u frekvenciji zvuka. Više vrijednosti ovog parametra rezultiraju bržim varijacijama, dok niže vrijednosti rezultiraju sporijim varijacijama.
- Delay Time: Ovaj parametar određuje vremenski razmak između originalnog zvuka i njegovih varijacija. Povećanje ovog parametra rezultira dužim vremenskim razmacima, što može stvoriti dublji i širi zvuk.
- Spread: Spread određuje raspon (u stupnjevima) unutar kojeg se varijacije u frekvenciji šire po stereo polju.

Slika 4.36 Komponenta ChorusEffect u korisničkom sučelju

4.8.7 Filter

Filter efekt je audio efekt koji se koristi za filtriranje određenih frekvencijskih komponenti iz zvučnog signala (Slika 4.37). Ovisno o postavkama, filter može propustiti samo visoke frekvencije (highpass filter) ili samo niske frekvencije (lowpass filter). [15]

Slika 4.37 Vizualni prikaz efekta "filter"

U komponenti FilterEffect, korisnik može prilagoditi različite parametre filter efekta (Slika 4.38) kako bi oblikovao zvuk prema svojim preferencijama:

- Frequency: Ovaj parametar određuje frekvenciju od koje filter počinje djelovati. Za highpass filter, to bi značilo da će se propustiti sve frekvencije iznad ove vrijednosti, dok će se za lowpass filter propustiti sve frekvencije ispod ove vrijednosti.
- Type: Ovaj parametar određuje tip filtera koji će se primijeniti na zvuk. Korisnik može odabrati između highpass i lowpass filtera. Highpass filter propušta samo visoke frekvencije, dok lowpass filter propušta samo niske frekvencije.

Slika 4.38 Komponenta FilterEffect u korisničkom sučelju

Prilagođavanjem ovih parametara, korisnik može postići različite zvučne efekte, od smanjenja buke i "čišćenja" zvuka do stvaranja specifičnih zvučnih karakteristika, pružajući tako dodatnu kreativnost i kontrolu nad zvukom u glazbenoj produkciji.

5 Zaključak

Ovaj rad predstavlja istraživanje i implementaciju naprednih tehnologija u kontekstu razvoja web aplikacija za obradu audio zapisa u stvarnom vremenu. Fokus je na integraciji React-a, Tone.js-a, TypeScript-a i Tailwind CSS-a kako bi se stvorila funkcionalna i efikasna aplikacija, prilagođena za obradu zvuka direktno u web pregledniku. Implementacija efekata u aplikaciji omogućila je korisnicima da eksperimentiraju s raznim audio transformacijama. Tone.js, temeljen na Web Audio API-ju, pružio je moćne alate za manipulaciju zvukom, uključujući reprodukciju, efekte poput reverba, delay-a, distorzije, tremola, chorusa i filtera, prilagođavajući ih prema specifičnim zahtjevima korisnika. Svaki od ovih efekata integriran je kroz jasno definirane komponente koje olakšavaju upotrebu unutar aplikacije. Upotrebom TypeScript-a osigurana je stabilnost i održivost koda kroz statičko tipiziranje, što je ključno za složene aplikacije poput ove. Tailwind CSS je omogućio brzu i efikasnu stilizaciju aplikacije, pružajući nam fleksibilnost u definiranju izgleda komponenti. Organizacija koda prema principima Atomic Designa doprinijela je jasnom razgraničenju komponenti, čineći aplikaciju preglednijom i lako proširivom.

Ovaj projekt ne samo da demonstrira snagu integracije naprednih tehnologija za real-time obradu zvuka na webu, već i pruža platformu za daljnja istraživanja i inovacije u području audio tehnologija, potičući razvoj novih aplikacija u glazbenoj produkciji i srodnim područjima.

Literatura

[1] Meredith Shubel, What is TypeScript?, The New Stack, (2022, srpanj). Poveznica: https://thenewstack.io/what-is-typescript

[2] React, Wikipedia. Poveznica: https://en.wikipedia.org/wiki/React_(JavaScript_library)

[3] Jim Bennett, Let's Make Music with JavaScript and Tone.js, Medium, (2018, listopad). Poveznica: https://medium.com/dev-red/tutorial-lets-make-music-with-javascript-and-tone-js-f6ac39d95b8c

[4] Brad Frost, Atomic Design, (2016, studeni). Poveznica: https://atomicdesign.bradfrost.com/

[5] Rakesh Purohit, An Enlightening Journey To Understand When And How To Use React Context API, DhiWise, (2023, listopad). Poveznica: https://www.dhiwise.com/post/an-enlightening-journey-to-understand-when-and-how-to-use-react-context-api

[6] Vercel, Next.JS Docs, (2024, svibanj). Poveznica: https://nextjs.org/docs

[7] NTIAudio, Fast Fourier Transformation FFT – Basics. Poveznica: https://www.nti-audio.com/en/support/know-how/fast-fourier-transform-fft

[8] SoftDb, What is Reverberation in Acoustical Analysis?, (2019, svibanj). Poveznica: https://www.softdb.com/blog/what-is-reverberation-in-acoustical-analysis/

[9] Avid, WHAT IS REVERB? TYPES, PARAMETERS, & USES EXPLAINED, (2024, ožujak). Poveznica: https://www.avid.com/resource-center/what-is-reverb

[10] John Gibson, Delay Effects, Indiana University Bloomington. Poveznica: https://cecm.indiana.edu/361/rsn-delay.html

[11] Aden Russell, Distortion and Saturation: The Complete Guide, EDMProd, (2024, siječanj). Poveznica: https://www.edmprod.com/distortion-saturation-guide/

[12] Dusti Miraglia, What Is Tremolo? Discover The Thrilling Power Of Pitch Oscillation, Unison Audio, (2023, svibanj). Poveznica: https://unison.audio/what-is-tremolo/

[13] What is the chorus effect?, Native Instruments, (2023, svibanj). Poveznica: https://blog.native-instruments.com/chorus-effect/

[14] Simon Haven, Audio Filters Explained: Low-Pass, High-Pass and Beyond, EDMProd, (2022, studeni). Poveznica: https://www.edmprod.com/audio-filters/

[15] Kevin McHugh, Essential audio filters guide: How to use high-pass, low-pass, and band-pass filters, Native Instruments, (2023, veljača). Poveznica: https://blog.native-instruments.com/audio-filters-guide/

[16] Tone,js docs. Poveznica: https://tonejs.github.io/docs/15.0.4/index.html

Preuzete Slike

- Slika 3.2: https://nextjs.org/docs/app/building-your-application/routing; preuzeto 12.6.2024
- Slika 3.3: https://atomicdesign.bradfrost.com/chapter-2/; preuzeto 12.6.2024
- Slika 3.4: https://www.dhiwise.com/post/an-enlightening-journey-to-understand-when-and-how-to-use-react-context-api; preuzeto 12.6.2024
- Slika 4.19: https://www.nti-audio.com/en/support/know-how/fast-fourier-transform-fft; preuzeto 12.6.2024
- Slika 4.27: https://www.softdb.com/blog/what-is-reverberation-in-acoustical-analysis/; preuzeto 12.6.2024
- Slika 4.29: https://cecm.indiana.edu/361/rsn-delay.html; preuzeto 12.6.2024
- Slika 4.31: https://www.edmprod.com/distortion-saturation-guide/; preuzeto 12.6.2024
- Slika 4.33: https://unison.audio/what-is-tremolo/; preuzeto 12.6.2024
- Slika 4.35: https://blog.native-instruments.com/chorus-effect/; preuzeto 12.6.2024
- Slika 4.37: https://www.edmprod.com/audio-filters/; preuzeto 12.6.2024

Razvoj web-aplikacije za dodavanje zvučnih efekata i snimanje glasa

Sažetak

Ovaj rad se bavi izradom web aplikacije za dodavanje audio efekata na ulazni zvuk. Za manipulaciju zvukom korišten je Tone.js, dok su za izradu korisničkog sučelja primijenjeni React i Tailwind CSS. Implementirane su funkcionalnosti za reprodukciju uživo putem mikrofona te za učitavanje i manipulaciju audio datoteka. Korisnici mogu eksperimentirati s raznim efektima poput reverb-a, delay-a, tremola i drugih. Dodatno, implementiran je vizualizator koji intuitivno prikazuje karakteristike audio signala.

Ključne riječi: web aplikacija, audio efekti, Tone.js, React, TypeScript, Tailwind CSS, Next.js, digitalna obrada zvuka, vizualizacija signala

Development of a web application for adding sound effects and recording voice

Abstract

This paper focuses on developing a web application for applying audio effects to input sound. Tone.js is used for sound manipulation, while React and Tailwind CSS are employed for building the user interface. The application features live microphone input playback and supports loading and manipulation of audio files. Users can experiment with various effects such as reverb, delay, tremolo, among others. Additionally, a visualizer has been implemented to intuitively display the characteristics of the audio signal.

Keywords: Web application, audio effects, Tone.js, React, TypeScript, Tailwind CSS, Next.js, digital signal processing, signal visualization