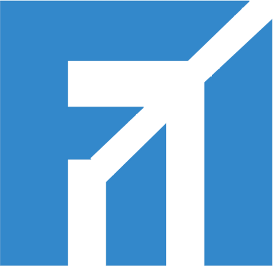
# Universitatea “Alexandru Ioan Cuza” din Iași FACULTATEA DE INFORMATICĂ



Lucrare de licență

**Sound Maker**

propusă de

***Scutaru Paul-Alexandru***

**Sesiunea:** iulie, 2021

Coordonator

**Sabin-Corneliu Buraga**

# Universitatea “Alexandru Ioan Cuza” din Iași FACULTATEA DE INFORMATICĂ

**Sound Maker**

***Scutaru Paul-Alexandru***

**Sesiunea:** iulie, 2021

Coordonator

**Sabin-Corneliu Buraga**

# Cuprins

[Copertă 1](#_Toc73368374)

[Titlu 2](#_Toc73368375)

[Cuprins 3](#_Toc73368376)

[Introducere 4](#_Toc73368377)

[Contribuții personale 6](#_Toc73368378)

[Capitolul 1: Fundamentele aplicației 7](#_Toc73368379)

[Tehnologii 7](#_Toc73368380)

[Biblioteci adiționale 8](#_Toc73368381)

[Frontend 8](#_Toc73368382)

[Backend 8](#_Toc73368383)

[Concepte, strucuri de date și algoritmi 10](#_Toc73368384)

[Componentă audio 10](#_Toc73368385)

[Oscilator 11](#_Toc73368386)

[*Noise* 11](#_Toc73368387)

[Sintetizator 11](#_Toc73368388)

[*Sampler* 11](#_Toc73368389)

[*Player* 12](#_Toc73368390)

[Secvențiator 12](#_Toc73368391)

[MIDI 12](#_Toc73368392)

[Capitolul 2: Scopurile și cerințele proiectului 13](#_Toc73368393)

[Capitolul 3: Proiectare și analiză 15](#_Toc73368394)

[Capitolul 4: Implementare 16](#_Toc73368395)

[Capitolul 5: Manual de utilizare 17](#_Toc73368396)

[Capitolul 6: Concluzii 18](#_Toc73368397)

[Bibliografie 18](#_Toc73368398)

[Anexe 19](#_Toc73368399)

# Introducere

*“Muzica este arta de a gândi prin sunete.”*

*- Jules Combarieu*

Procesul de producție muzicală se îndreaptă tot mai mult către mediul digital, făcând posibilă crearea și redarea unei compoziții muzicale de amploare folosind în exclusivitate, de exemplu, un laptop.

Ceea ce în urmă cu 20 de ani necesita adunarea unei întregi orchestre, acum se poate realiza de pe un computer, de către un singur profesionist, prin intermediul sintetizatoarelor, al instrumentelor virtuale și al varietății de *plugin*-uri cu capacități de *sampling* sau de adăugare de efecte sonore.

Deși se considera că producția muzicală se poate realiza la nivel înalt doar într-un studio profesional, evoluția rapidă a tehnologiei dovedește clar că se pot obține aceleași rezultate, dacă nu chiar superioare, folosind un singur computer echipat cu software-ul necesar.

Astfel, proiectul ***Sound Maker*** este o aplicație Web orientată către domeniul muzical, având o temă preponderent artistică, din sfera producției muzicale, dar și cu un aspect practic. Destinația proiectului este, în mod exclusiv, platforma *desktop.*

Tema se axează pe oferirea posibilității de creare, procesare și augmentare de sunet, cu scopul folosirii în compoziții muzicale, sau pur și simplu cu scop artistic liber.

Motivația alegerii acestei teme este crearea unei aplicații Web care să poată introduce utilizatorul în domeniul producției muzicale, printr-un mediu interactiv, ce poate fi de folos chiar și eventualilor experți în domeniu.

Lipsa unei aplicații Web ușor de folosit dar care să aibă și o capacitate ridicată de prelucrare a sunetului stă de asemenea la baza alegerii făcute.

Gradul de noutate al temei este adus de folosirea unor tehnologii de actualitate, mai exact *framework*-ul *React.js* pe partea de interfață, îmbinat cu *Express.js* pe partea de server. De asemenea, pe partea de procesare de sunet a fost folosit *framework*-ul *Tone.js* pe care se bazează întreaga suită de funcționalități audio ale aplicației.

Deși proiectul ***Sound Maker*** este bazat în special pe programare *frontend,* acesta pune la dispoziție și un API simplu, dar capabil, implementat cu *Express.js* tocmai datorită gradului său ridicat de eficiență, minimalism și simplitate.

Pe partea de stocare a datelor a fost folosit bine-cunoscutul serviciu de baze de date relaționale *MySQL*, un serviciu ușor de integrat și de folosit datorită gamei largi de conectori și drivere.

Dispunând de functionalitățile de *login* și *register,* aplicația stochează datele de autentificare ale utilizatorilor, pentru a oferi acestora posibilitatea de salvare a sunetelor create în baza de date *MySQL,* utilizatorii fiind capabili de asemenea să partajeze între ei sunetele salvate. Astfel, aplicația reține două tabele în baza de date, una cu datele utilizatorilor și una cu sunetele salvate de aceștia.

Utilizatorul autentificat poate să creeze, să proceseze și să modifice sunete, fie începând de la un sunet de bază, fie de la unul modificat, salvat anterior în baza de date sau pe computerul propriu.

Proiectul îi pune acestuia la dispoziție o gamă solidă de instrumente și unelte audio, fiecare cu setări caracteristice, efecte și detalii. Printre acestea se numără clasice unelte de producție muzicală, cum ar fi oscilatoare, sintetizatoare, zgomot static și tobe.

Pe langă oferirea ustensilelor tehnice de manipulare audio, aplicația dispune și de funționalități de redare a sunetului, fie prin intermediul unui *player mp3*, fie prin intermediul unui *sampler.* De asemenea, este integrată și funcționalitatea de folosire a microfonului, cu posibilitatea adăugarii de efecte pe înregistrare.

O altă caracteristică de interes din cadrul proiectului este funcționalitatea de redare a fișierelor MIDI, care pot fi descărcate de pe internet.

Accesând secțiunea “*My sounds*”, se pot revizui sunetele salvate de către utilizator, cu posibilitatea de a fi redate, partajate sau șterse. Sunt prezente și opțiuni de filtrare sau de afișare a detaliilor tehnice. Mai mult, în secțiunea “*Shared sounds*”, pot fi găsite toate sunetele partajate de utilizatori, cu opțiuni de filtrare, redare, salvare sau afișare de detalii.

Structura lucrării este alcătuită din 6 capitole.

În primul capitol, “*Fundamentele aplicației*”, se regăsesc informații succinte cu privire la conceptele, algoritmii și tehnologiile folosite, cât și despre bibliotecile adiționale utilizate în implementarea aplicației.

Capitolul următor, “*Scopurile și cerințele proiectului*”, prezintă într-un mod mai detaliat ținta aplicației cât și privirea de ansamblu. Aici se regăsesc și comparații cu aplicații asemănătoare ca scop.

În al treilea capitol, “*Analiză și proiectare*”, se află informații detaliate referitoare la fazele ingineriei software, mai exact motivele alegerii tehnologiilor aferente, modalității de stocare, algoritmi, modularitatea, cât și principii de design în interacțiunea cu utilizatorul, publicul țintă. Secțiunea conține și detaliile cu privire la modularitatea proiectului, mai exact biblioteci, API-uri, diagrame.

Capitolul “*Implementare”* prezintă descrierea amănunțită a modului de implementare propriu-zis, cu considerații despre performanță și securitate.

După cum și titlul denotă, capitolul 5 “*Manual de utilizare*” conține ghidul de folosire al aplicației, cu cazurile de utilizare.

Ultimul capitol “*Concluzii*” oferă reluarea subiectului atins într-un mod succint, cu precădere notându-se aspectele semnificative, cât și recomandări sau diverse direcții viitoare de dezvoltare.

# Contribuții personale

Drept contribuție personală, poate fi considerat faptul că, prin intermediul aplicației ***Sound Maker***, a fost realizată o agregare și implementare compactă a celor mai interesante funcționalități ale bibliotecii *Tone.js*, oferind un demo al acesteia.

Astfel, dezvoltatorii care doresc să utilizeze această bibliotecă pot avea un punct de start cu privire la modul de abordare, luând ca exemplu proiectul ***Sound Maker***.

De asemenea, au fost agregate și funcționalități adiționale, utilizând *Web Audio API,* prin înfăptuireaunui *player* capabil să redea fișiere mp3, ce dispune și de un element de vizualizare a sunetului.

O altă funcționalitate auxiliară este reprezentată de capacitatea partajării sunetelor create către alți utilizatori, fapt ce poate avea o alură de educare cu privire la modul de folosire al bibliotecii, cât și cu privire la diverse modalități de producție muzicală.

Implementări din alte perspective ale diverselor componente audio construite în cadrul proiectului mai pot fi găsite pe internet, dar acestea nu sunt neapărat implementate folosind același limbaj de programare sau *framework-uri,* unele nefiind complet funcționale, având doar scop pedagogic, nu practic.

Totuși, în aplicația ***Sound Maker,*** acestea sunt toate adunate împreună, agregate cu succes într-un mediu compact și ușor de folosit, de asemenea fiind deplin funcționale.

În plus, este adusă și posibilitatea salvării și descărcării sunetelor create, fapt ce poate fi considerat crucial.

De exemplu, sunetele pot fi folosite ulterior în alte aplicații, la latitudinea utilizatorului.

În același timp, și sunete create în afara proiectului pot fi folosite în cadrul acestuia, prin intermediul opțiunilor de redare audio sau de *sampling.*

Astfel, sinergia creată crește puterea altor aplicații din aceeași sferă, cât și puterea proiectului ***Sound Maker***.

Prin urmare, contribuția adusă domeniului de aplicații Web orientate audio poate fi considerată un succes.

# Capitolul 1: Fundamentele aplicației

## Tehnologii

Fiind unul dintre cele mai populare și capabile limbaje de programare, cu un suport imens din partea comunității programatorilor, *Javascript* conduce topul alegerilor în dezvoltarea aplicațiilor Web. Prin intermediul său se poate construi un cadru ce aduce rapiditate și dinamism experienței utilizatorului.

Rulând direct în *broswer*, codul *Javascript* dispune de viteză de execuție. De asemenea, comunicarea cu serverul în mod asincron permite interacțiunii cu utilizatorul să fie fără întreruperi pe *frontend.*

Deoarece proiectul ***Sound Maker*** trebuie să redea un mediu dinamic și interactiv, alcătuit din diverse unelte de creare, redare și procesare de sunet, se justifică alegererea tehnologiei *React.js* pe stratul de interfață al aplicației, potrivită proiectelor care necesită schimbarea datelor fără reîncărcarea paginii.

*React.js* permite crearea de componente UI reutilizabile, oferind un grad ridicat de modularizare, sincronizare si îmbinare al componentelor.

Aceste componente vor reprezenta de asemenea instrumentele muzicale și uneltele puse la dispoziție utilizatorului. De asemenea, tehnologia în cauză oferă și posibilitatea de *routing* eficient între paginile aplicației, făcând ușoară construirea unui sistem de navigație. Prin urmare, *framework*-ul este flexibil, având de asemenea și performanță ridicată.

Pe partea de server, tehnologia *Express.js* prezintă o compatibilitate mare cu *React.js*, cele două dispunând de capacitatea creării de aplicații Web într-un mod rapid și simplu, fiind alegeri destul de populare în momentul actual.

Tehnologia folosită pentru crearea și procesarea de sunet, *Tone.js,* este de asemenea una dintre cele mai capabile din sfera sa, perfectă pentru crearea interactivă de muzică în *browser*, cu o arhitectură familiară atât pentru programatori cât și pentru muzicieni, oferind capacitatea de a dezvolta aplicații orientate către domeniul audio, pe Web.

Această tehnologie dispune de funcționalități asemănătoare unei stații audio digitale, cu toate capacitățile de a crea sunet, muzică. Prin urmare, *Tone.js* oferă o gamă de instrumente și unelte audio complexe, fiecare cu setări caracteristice și efecte.

Printre acestea se află clasice ustensile de producție muzicală, precum oscilatoare, sintetizatoare, zgomot static, tobe, *sampler,* dar și modalități de sincronizare, temporizatoare, secvențiatoare sau instrument de *playback*.

De asemenea, sunt oferite și posibilități de vizualizare a sunetului, animație și analiză.

## Biblioteci adiționale

### Frontend

În afară de principalele tehnologii folosite pentru a construi baza proiectului, au fost utilizate și altele tehnologii auxiliare.

Acestea permit ușurarea, eficientizarea și modularizarea dezvoltării întregului proiect.

Pe partea de *frontend* s-au folosit biblioteci *Javascript* precum *Yup,* *Formik, Axios* și *Classnames.*

*Yup* este o bibliotecă folosită pentru parcurgere și validare de date, utilizată în cadrul proiectului la validare pe partea de client a informațiilor introduse de utilizator, în cadrul *form-*urilor.

Îmbinată cu biblioteca *Formik*, cele două permit construirea de *form-*uri personalizate, sigure și ușor de dezvoltat.

*Formik* este folosită la extragerea și introducerea datelor, validare și afișare de erori, cât și la procesarea de *form submissions*.

În contextul proiectului, sunt folosite *form-*uri pentru funcționalitatea de înregistrare a utilizatorului.

Validarea constă în verificarea corectitudinii datelor introduse, acestea trebuind să îndeplinească diverse criterii în funcție de formatul necesar a fi trimis pe *backend*.

Aceste criterii sunt, de exemplu, lungimea datelor, evitarea unor caractere speciale, precum și oprirea diverselor atacuri cu cod de către utilizatori rău-voitori.

*Axios,* o bibliotecă de capacitate ridicată în crearea de *request*-uri, prezintă o mare importanță în procesul de dezvoltare a aplicației.

Aceasta face posibilă construirea, primirea și trimiterea de *request*-uri pe partea de *frontend*, cu un API *promise-based*, cu suport împotriva *Cross-site request forgery.*

Utilizarea pachetului *Axios* permite astfel înfăptuirea mediului dinamic și interactiv pe care se bazează întreaga aplicație, contribuind la oferirea capacității de a actualiza componentele paginilor fără a fi nevoie de *refresh.*

Pe lângă acestea, a mai fost de folos un mic pachet, *Classnames,* util în contopirea condițională a numelor de clase CSS.

### Backend

Discutând despre programarea *backend*, vor fi menționate următoarele: *Bcrypt, Jsonwebtoken* și *Sequelize ORM.*

*Bcrypt* este o mică bibliotecă cu funcționalități de *hash*, folosită pentru criptarea parolelor.

Aceasta s-a utilizat în funcționalitatea de înregistrare.

Peste parola utilizatorului este aplicat un *hash* cu un secret, astfel putând fi stocată în mod sigur în baza de date.

*Jsonwebtoken*, un pachet ce permite decodarea, semnarea și generarea de *token-*uri, a fost de folos în implementarea sistemului de autentificare, fiecare utilizator primind un *token* ce va fi stocat local, după logare.

Acest *token* permite verificarea autenticității unui client ce încearcă sa facă *request*-uri la API-ul aplicației, implementând astfel un sistem securizat.

În cele din urmă, trebuie prezentat și pachetul ORM numit *Sequelize,* un pachet *promise-based* compatibil cu sistemul de stocare folosit, *MySQL.*

Acesta a fost folosit pentru a ușura dezvoltarea aplicației, mai exact a modului în care sunt făcute operațiile asupra bazei de date, în cadrul API-ului.

Astfel, inserarea, ștergerea și selectarea datelor au putut fi implementate într-un mod compact și ușor de reprezentat, la fel și crearea/alterarea tabelelor.

La fiecare rulare a serverului, funcționalitățile *Sequelize* intră în acțiune, permițând un mediu de dezvoltare sincronizat.

De altfel, toate tehnologiile și bibliotecile alese au fost selectate în așa fel încât sa permită eficientizarea și îmbunătățirea mediului de dezvoltare.

Cât despre modalitatea de stocare, *MySQL* este o tehnologie bine-cunoscută, un serviciu de baze de date relaționale, un serviciu ușor de integrat și de folosit datorită gamei largi de conectori și drivere.

O altă alternativă ar fi putut fi *MongoDB*, pentru a stoca doar obiecte de tip JSON, dar ținând cont de faptul că aplicația este orientată *frontend,* cu un *backend* nu foarte complex, *MySQL* a reușit să îndeplinească de asemenea cerințele proiectului, succesul bazându-se pe compatibilitatea și simplitatea caracteristice.

## Concepte, strucuri de date și algoritmi

***Sound Maker*** se bazează în principal pe oferirea de ustensile audio, sub forma unor componente audio interactive și dinamice, capabile de creare, procesare și augmentare de sunet.

### Componentă audio

Conceptul de “componentă audio”, sau “instrument” este ceea ce se pune în prim plan în cadrul aplicației, este unealta principală pusă la dispoziția utilizatorului.

Proiectul fost gândit în așa fel încât fiecare “instrument” dispune de butoane de redare audio, de opțiuni complexe specifice și de diverse posibilități de adăugare a efectelor sonore.

În plus, sunt prezente și functionalități de salvare a configurației actuale a unui instrument, cât și de descărcare a înregistrărilor. În funcție de instrument, pot exista functionalități auxiliare cum ar fi afișarea notelor muzicale cântate.

Conceptul de “opțiuni” aferente instrumentelor semnifică câmpuri de *input* pe care utilizatorul le poate modifica și în care poate introduce date. Aceste câmpuri pot fi de tip *slider,* *checkbox*, text sau buton.

Efectele audio prezente în cadrul instrumentelor reprezintă diverse procesări sonore adăugate peste sunetul de bază al instrumentului, schimbându-i sonoritatea.

Practic, biblioteca *Tone.js* lucrează cu obiecte de tip JSON, în cadrul API-ului. Prin urmare, reprezentarea obiectelor de tip “instrument”, ”opțiuni” sau ”efecte” se face de asemenea în același format.

De altfel, toate procesele de comunicare ale aplicației, fie comunicare *frontend-backend*, fie dintre componentele aflate în *frontend*, se realizează prin transmitere de obiecte JSON, simplificând astfel logica proiectului folosind un format ce poate fi citit atât de om cât și de computer.

Dezvoltând și mai mult ideea de instrument folosită în funcționalitățile aplicației, trebuie prezentat modul în care biblioteca reprezintă aceste obiecte.

Pachetul *Tone.js* este bazat pe *Web Audio API*, care este un sistem versatil de control audio pe Web, permitând dezvoltatorilor să creeze aplicații complexe din punct de vedere audio.

Capacitățile sale includ posibilități de alegere a surselor sonore, adăugarea de efecte, crearea de vizualizări sonore și altele.

Asfel, trebuie prezentat conceptul folosit de acest sistem, mai exact cel de “context audio”. Orice operație de procesare este aplicată peste noduri audio, care, conectate împreună, formează un graf de rutare audio.

În cadrul unui context audio, se creează surse sonore, peste acestea putând fi suprapuse noduri de efecte. Apoi, este aleasă destinația de output, ce poate fi orice sistem de redare a sunetului, de exemplu, boxele conectate la computer. Conectând toate sursele la destinație, se completează procesul de redare și manipulare sonoră folosit de *Web Audio API.*

Revenind la biblioteca *Tone.js,* aceasta suprapune propriile sale obiecte peste cele de tip nod audio, adăugând functionalități noi, orientate către posibilitatea folosirii în producție muzicală.

Nodurile pot avea ca *input* sunetul provenit de la oscilatoare, sintetizatoare, un *sampler,* tobe, sau chiar microfoane. Obiectele în cauză sunt, de fapt, clase OOP, iar caracteristicile lor sunt reprezentate sub formă de obiecte JSON.

În cele ce urmează vor fi prezentate conceptele de oscilator, *noise*, sintetizator, *sampler, player* și secvențiator, cât și formatul MIDI.

### **Oscilator**

La bază, este un circuit continuu, repetat, ce alternează undele sonore într-o anumită frecvență. În cadrul aplicației, este realizat un oscilator virtual, cu aceleași caracteristici ca unul fizic.

Acesta poate fi de mai multe tipuri, în funcție de tipul de sunet pe care îl produce.

Practic, modul în care se creează sunetul este prin trecerea unui semnal de *feedback* prin “circuit”, producând un semnal de *output*.

Este un instrument rudimentar care stă la baza creării sunetelor multor instrumente mai complexe.

### *Noise*

Acest concept se referă mai exact la simularea zgomotului static, care nu este altceva decât semnale electrice sub formă de frecvențe *random*. Poate fi de mai multe feluri, în funcție de caracteristicile sonore.

De asemenea, are capacitatea de a simula diverselor sunete cum ar fi vântul, sunetul mării.

### Sintetizator

Un sintetizator este un instrument muzical electronic, ce permite generarea de semnale audio pe anumite note. Acesta are în componența sa mai multe oscilatoare, de obicei.

În general, sintetizatoarele sunt folosite pentru a reda note cu ajutorul tastaturii, sau controlate de secvențiatoare ori MIDI.

Aplicația implementează sintetizatoare sub formă clasică de pian, cu un algoritm de selectare și parcurgere a gamelor muzicale, ce redă le redă asemeni unor clape reprezentate de butoane HTML.

### *Sampler*

*Sampler*-ul reprezintă o ustensilă de mare importanță în domeniu audio, fiind un instrument electronic ce folosește sunete înregistrate (chiar sunete ale altor instrumente, eventual) pe care le încarcă și le redă, pe anumite note muzicale.

Pentru a adapta un sunet la o anumită notă, se folosește conceptul de *pitch-shifting,* care constă în schimbarea tonului acelui sunet.

Capacitățile acestui instrument sunt limitate doar de imaginația utilizatorului.

### *Player*

Un *player* audio este o ustensilă bine-cunoscută de redare a înregistrărilor audio de pe un computer. Acesta dispune de butoane pentru control media, cum ar fi de redare, oprire, schimbare a poziției de redare, sau de volum.

În cadrul proiectului, acesta acceptă fișiere audio în format mp3. De asemenea, tot în aplicație, *player*-ul audio este conectat la o planșă de vizualizare în timp real a frecvențelor înregistrării, crescând dinamismul experienței și oferind eventuale informații tehnice pentru cunoscători.

Algoritmul ce permite această animație vizuală este unul de desenare în timp real a cadrelor de animație ce conțin amplitudinea frecvențelor sub formă de linii, reprezentând un *waveform*. Un *waveform* al unui semnal semnifică forma graficului în timp a amplitudinii semnalului.

### Secvențiator

Acestă unealtă audio simulează un aparat ce poate reda ritmic note muzicale, la un anumit tempo. Este folosit pentru a crea sau reda linii melodice.

În proiect, este reprezentat sub formă matriceală, cu un algoritm care schimbă notele reda în mod curent, la un interval fix de timp. Acest interval poate fi crescut sau scăzut, fiind de fapt tempo-ul (numărul de bătăi pe minut).

### MIDI

Ultimul concept de prezentat este formatul MIDI, care este un standard tehnic ce descrie protocolul de comunicare dintre instrumente muzicale electronice și computer.

Practic, “îi spune” computerului modul în care acesta trebuie să redea, de exemplu, o melodie aflată în acest format.

Împortanța acestei ustensile de redare este ridicată, prin intermediul ei aflându-se detalii semnificative cu privire la modul de a cânta/reda o melodie.

Majoritatea melodiilor care au măcar un mic grad de popularitate pot fi găsite pe internet sub formă de MIDI.

Singurul concept nedescris este cel de tobe, dar, se va presupunerea ca el este deja înțeles de cititor.

Astfel, este încheiată descrierea conceptelor, algoritmilor și structurilor de date principale folosite pentru crearea și redarea sunetului, în cadrul aplicației.

# Capitolul 2: Scopurile și cerințele proiectului

Acest capitol este destinat clarificării țintei urmărite, cât și a privirii de ansamblu. De asemenea, vor fi făcute și comparații cu alte soluții existente.

Se precizează faptul că proiectul este gândit pentru platforma *desktop,* acesta nefiind compatibil cu platforma *mobile.*

După cum a mai fost precizat, scopul aplicației este axat pe oferirea posibilității de creare, procesare și augmentare de sunet, cu scopul folosirii în compoziții muzicale, sau pur și simplu cu scop artistic liber.

Utilizatorul autentificat poate să creeze, să proceseze și să modifice sunete, fie începând de la un sunet de bază, fie de la unul modificat, salvat anterior în baza de date sau pe computerul propriu.

Crearea unei aplicații Web care să poată introduce utilizatorul în domeniul producției muzicale, printr-un mediu interactiv, ce poate fi de folos chiar și eventualilor experți în domeniu, semnifică ținta principală a proiectului.

De altfel, un alt scop este reprezentat și de oferirea funcționalității de partaja sunete create de alți utilizatori, sau a partaja sunetele proprii cu aceștia, creând astfel un sistem de schimb.

Acest sistem poate fi văzut și din punct de vedere al unui sistem social, considerând că muzica este o formă de comunicare interumană. Pe lângă acest aspect, se remarcă și un rol de educare în domeniul producției muzicale, observând practicile altor utilizatori mai experimentați.

Scopul de divertisment al aplicației nu este de asemenea de neglijat, mai ales pentru pasionații de muzică. Aceștia pot învăța combinând utilul cu plăcutul. Mai mult, și un utilizator care nu are cunoștințe muzicale ar putea fi surprins de dinamismul sonor al aplicației.

Muzica reprezintă o artă, prin urmare, în cadrul proiectului, s-a urmărit oferirea posibilității de dezvoltare artistică, de îndreptare către această direcție.

Se dorește astfel întărirea afirmației că oricine poate folosi aplicația cu succes, după interesele proprii, indiferent de gradul de pregătire al persoanei în domeniul producției muzicale. Pe lângă acest lucru, chiar și eventuali experți ai domeniului pot găsi funcționalitățile proiectului de folos, profitând de capacitățile artistice oferite.

Un alt scop este considerat realizarea unei implementări a celor mai interesante aspecte ale bibliotecii *Tone.js*, pentru a educa eventuali programatori cu privire la modurile de abordare ale acestui *framework*.

Într-adevăr, se pot găsi pe internet implementări parțiale, din alte perspective, ale unor funcționalități din cadrul proiectului, dar acestea nu înfăptuiesc o agregare completă sau în intregime funcțională.

Cea mai relevantă implementare din punct de vedere al gradului de asemănare cu aplicația ***Sound Maker*** este chiar demo-ul oferit de dezvoltatorii bibliotecii *Tone.js.*

Linkul implementării: [*https://tonejs.github.io/examples/*](https://tonejs.github.io/examples/)

Împreună cu pagina de descriere a API-ului *Tone.js*, cele două au servit drept punct de plecare și de inspirație pentru dezvoltarea proiectului.

Sursa prezentată spre comparare are ca scop doar prezentarea *framework-*ului, fiind lipsită de alte funcționalități adiționale.

Prin ***Sound Maker***, s-a realizat și implementarea „sistemului social” de partajare a sunetelor, cât și o prezentare mai compactă a functionalităților, într-un mod mai ușor de înțeles, luând în considerare că nu oricine știe de semnifică un API sau un obiect JSON. De asemenea, trebuie menționată și oferirea în plus a unui sistem de stocare pentru sunetele create.

Pentru a folosi biblioteca *Tone.js*, în esență, este necesar a avea, într-o oarecare măsură, cunoștințe de producție muzicală, pe lângă cunoștințele de utilizare și încadrare a API-urilor externe.

Spre deosebire de acest demo, ***Sound Maker*** oferă un mediu potrivit pentru orice tip de utilizator, nu doar un dezvoltator sau un producător muzical.

În cadrul proiectului, s-a decis omiterea selectivă a implementărilor de vizualizare sonoră sub formă de animații.

Motivația este că accentul se pune pe evitarea supraîncărcării mediului vizual.

Deși aceste implementări ar fi sporit gradul de dinamism al proiectului, menținerea unui cadru compact și simplu pentru utilizator a fost aleasă ca prioritate.

Descrierea API-ului *Tone.js*, se găsește la adresa: <https://tonejs.github.io/docs>

Exemplele regăsite pe această pagină au servit de altfel drept punct de plecare pentru dezvoltarea funcționalităților sonore ale ***Sound Maker*** (cele care depind de biblioteca *Tone.js*).

# Capitolul 3: Proiectare și analiză

## Design

Se precizează din nou faptul că aplicația este destinată platformei *desktop,* aceasta nefiind compatibilă cu platforma *mobile.*

Motivele sunt imposibilitatea folosirii unei asemenea aplicații pe un ecran mic ca dimensiuni, având prea multe setări și elemente interactive.

De asemenea, performanța ar fi fost și ea o problemă, aplicația având nevoie de putere de procesare.

Având stabilită clar ținta aplicației ***Sound Maker***, proiectarea s-a axat pe crearea de concepte ușor de înțeles, cu un design simplu și plăcut vizual.

Sursa de inspirație pentru design poate fi considerată aspectul aplicațiilor de tip stație audio digitală, deși au trebuit făcute ajustări pentru ca proiectul să fie potrivit platformei Web.

S-a decis astfel crearea unei aplicații a cărei funcționalități vor fi aflate pe o singură pagină principală, cu posibilitatea de *scroll* vertical, unde este cazul. Pagina dispune de simplitate, evitându-se supraîncărcarea vizuală.

Pe această pagină se găsesc toate instrumentele și componentele necesare pentru a crea, modifica și procesa sunet, cât și cele ce permit partajarea și salvarea sunetelor create.

Primele *mockup*-uri s-au dovedit destul de bune pentru a rămâne drept abordarea finală de design.

Toate paginile aplicației conțin un *header* cu rol de navigare pentru utilizator. În acesta sunt prezente și diverse date informative (numele utilizatorului autentificat, logo-ul aplicației).

Pe lăngă *header*, paginile prezintă o secțiune principală de conținut, centrată în cadrul *viewport*-ului. Acest conținut include funcționalități, vizualizări de date.

Mai mult, pagina principală conține și un *sidebar*, cu rol de comutare între funcționalități, structurată în așa fel încât să informeze și cu privire la scopul acestora.

Paginile de *Login,* respectiv *Register* au o structură asemănătoare. Acestea au fost proiectate în așa fel încât să fie cât mai simple și mai compacte, prezentând un container centrat în pagină ce conține câmpurile de *input* necesare funcționalității, cu afișare de mesaje validatoare.

Pentru a se evita accesarea URL a unor pagini ce nu există, a fost creată o pagină de tip “*Page not found*”, ce afișează un mesaj de eroare corespunzător alături de un *link* de redirecționare.

De menționat este și prezența unei pagini de admin, ce poate fi accesată doar de administratorul în cauză.

Pentru a intra pe aceasta, este nevoie de cunoașterea parolei adminului, cât și cunoașterea URL-ului special la care se află pagina.

*Request-*urile făcute de pe pagina de admin sunt de asemenea verificate pe *backend,* acestea necesitând prezența parolei de admin în cadrul *header*-ului.

Paleta de culori este una simplă, din trei culori principale folosite în cadrul întregului proiect (portocaliu, gri, gri deschis), pe lângă culoarea de bază, alb.

Textul a fost stabilit ca fiind negru, pe font Verdana.

Fiind o aplicație orientată audio, s-a decis și amplasarea unui buton central pe pagina principală, care s-ă oprească sunetul întregii aplicații în caz că utilizatorul dorește acest lucru.

## Interacțiunea cu utilizatorul

Deoarece s-a dorit crearea unui mediu dinamic și interactiv pentru utilizator, toate funcționalitățile cu care acesta interacționează în mod direct au fost gândite după un anumit șablon.

Șablonul în cauză este următorul:

* Fiecare componentă audio trebuie să aibă un element de redare/oprire a sunetului, vizibil în prim-plan (de exemplu, unul sau mai multe butoane).
* În cadrul fiecărei componente audio există o secțiune de opțiuni caracteristice, cu câmpuri de *input* ce pot fi modificate de către utilizator.
* Pe lângă opțiuni, orice componentă prezintă și o secțiune de aplicare a diverselor efecte sonore.
* Componentele de creare de sunet dispun de butoane de salvare și descărcare a sunetului produs.

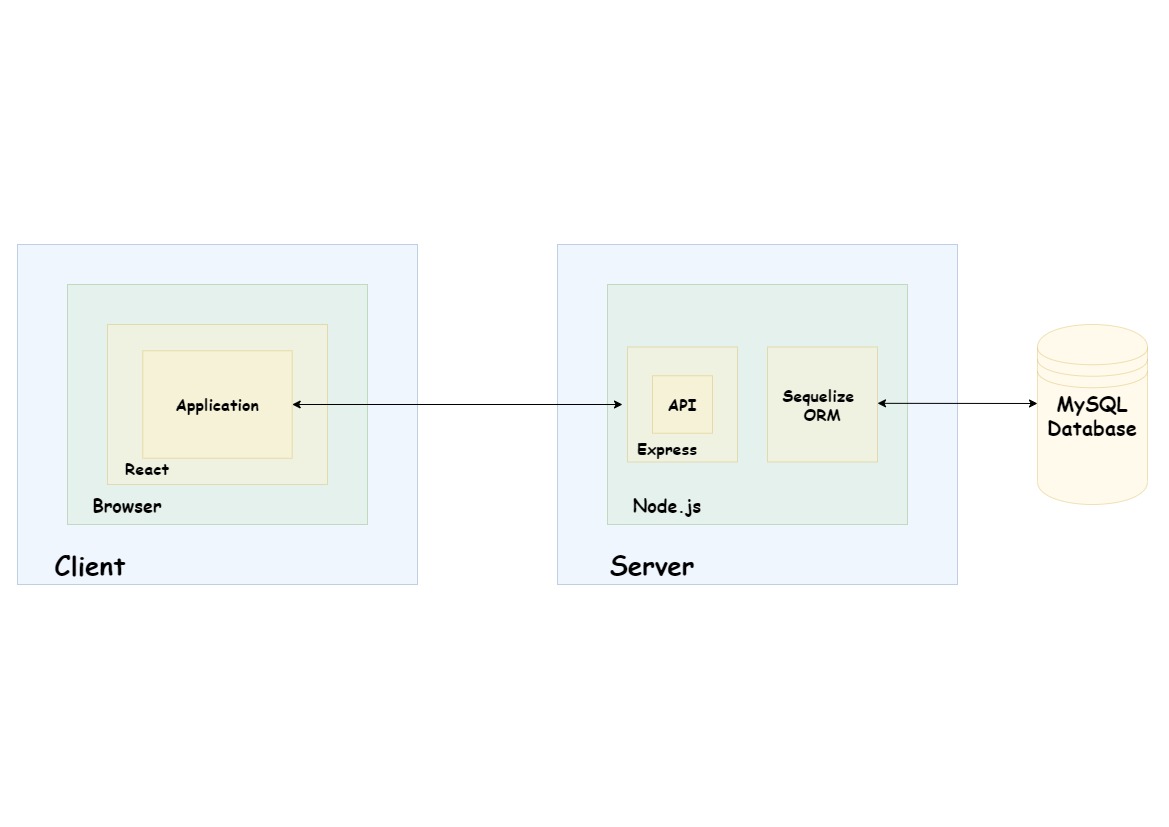
Fiind stabilit design-ul unei componente audio, trebuie precizat și că fiecare element a fost gândit încât să fie dinamic, oferindu-i utilizatorului capacitatea de modificare și augmentare.

După cum a mai fost menționat, tipul de utilizator pe care aplicația îl are drept țintă nu este unul foarte exact conturat. Practic, orice persoană dornică să creeze sunet poate folosi aplicația, fără a avea un scop practic clar, ci unul pur artistic.

Proiectarea aplicației a fost făcută în așa fel încât atât un expert în producție muzicală, cât și un utilizator obișnuit poate găsi de folos utilizarea ***Sound Maker***.

## Arhitectură *software*

Diagrama arhitecturală a proiectului:



Fiind necesar un *framework* capabil să redea un mediu dinamic și interactiv, se justifică alegererea tehnologiei *React.js* pe stratul de interfață al aplicației, perfectă pentru proiectele care necesită schimbarea datelor afișate fără reîncărcarea paginii.

Acesta comunică cu partea de server pe care se află *Express.js*, o tehnologie ce oferă eficiență, minimalism și simplitate.

Fiind o aplicație orientată către *frontend*, alegerea *Express.js* are ca motiv necesitatea unui *framework* simplist, potrivit dezvoltării unui API capabil, dar care nu are un grad de complexitate foarte ridicat.

De asemenea, prin intermediul pachetului *Sequelize ORM,* datele sunt modelate și extrase din baza de date *MySQL*, ajungând ulterior pe *frontend/backend*.

### Algoritmi

În cadrul aplicației, se poate observa implementarea unor algoritmi cu un grad de unicitate mai ridicat, de exemplu

#### Secvențiator în timp real

Algoritmul folosit de obiectul secvențiator permite vizualizarea și funcționalitatea sonoră a componentei.

Structura de date folosită este o matrice ce reprezintă un *grid,* fiecare element al acesteia fiind reprezentat de obiecte de tip notă muzicală.

Variabilele conținute de un astfel de obiect sunt numele notei (incluzând gama din care face parte), cât și o variabilă de semnalare a stării notei (activă, inactivă).

function GenerateGrid() {

    var grid = [];

    for (let i = 0; i < 16; i++) {

        let column = [

            { note: "C", isActive: false },

            { note: "D", isActive: false },

            { note: "E", isActive: false },

            { note: "F", isActive: false },

            { note: "G", isActive: false },

            { note: "A", isActive: false },

            { note: "B", isActive: false }

        ];

        grid.push(column);

    }

    return grid;

}

Fiecare notă este redată sub formă de buton, atribuindu-se alte proprietăți necesare (cheie unică, funcție *onclick*)

function NoteButton({ note, isActive, ...rest }) {

    const class\_name = isActive ? "note note-active" : "note";

    return (

        <button className={class\_name} {...rest}>

            {note}

        </button>

    );

};

În funcție de notele care au fost selectate de utilizator, algoritmul schimbă starea acestora în activă/inactivă pentru a le adăuga ulterior în matricea globală.

function handleNoteClick(clickedColumn, clickedNote) {

        // Copy of the grid with isActive updated

        let updatedGrid = grid.map((column, columnIndex) =>

            column.map((cell, cellIndex) => {

                let cellCopy = cell;

                // Inverts isActive for the cell that was clicked

                if (columnIndex === clickedColumn && cellIndex === clickedNote) {

                    cellCopy.isActive = !cell.isActive;

                }

                return cellCopy;

            })

        );

        // Updates our grid

        setGrid(updatedGrid);

    }

Astfel, matricea globală este actualizată și redată dinamic, în timp real, folosind funcționalitatea “*state*” din *React.*

La apăsarea butonului de start de către utilizator, algoritmul porneste rularea sincronizată și ritmică a parcurgerii notelor incluse ca fiind active în matrice.

let melody = []

        grid.forEach((column) => {

            let columnNotes = [];

            column.forEach(

                (shouldPlay) =>

                    // Push the given note if isActive is true

                    shouldPlay.isActive &&

                    columnNotes.push(shouldPlay.note + CHOSEN\_OCTAVE)

            );

            melody.push(columnNotes);

        });

Pașii ritmici sunt activați la intervale egale de timp, redând notele active, creând astfel obiectul de tip secvențiator.

Pentru contorizarea și semnalarea intervalelor de timp s-a folosit de transport a bibliotecii *Tone.js*.

Redarea sonoră este realizată de obiecte de tip sintetizator, conținute într-un vector la inițializare.

const Sequencer = new Tone.Sequence(

            (time, column) => {

                // Column highlight

                setCurrentColumn(column);

                // Adds the given note to synth

                synth.triggerAttackRelease(melody[column], "8n", time);

            },

            [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15],

            "8n"

        );

La fiecare pas, mai exact la trecerea unei unități de timp stabilite, se actualizează vizual și se redă sonor o coloană din matrice (doar notele active din coloană).

Proiectarea secvențiatorului îl limitează la 15 pași, pentru a nu supraîncărca interfața cu reprezentarea vizuală a unei matrice prea mari.

De asemenea, algoritmul este destul de solicitant, iar la un număr mai mare de pași ar fi apărut posibile probleme de performanță.

Documentația *Tone.js* specifică în mod clar necesitatea folosirii unității de contorizare a timpului conținută de bibliotecă, în detrimentul altor modalități, pentru a permite sincronizarea corectă a proceselor.

Din cauza acestui fapt s-a evitat implementarea funcționalității folosind unitatea de contorizare timp a *framework-*ului *React.js.*

La final, când utilizatorul apasă butonul de stop, se face eliberarea resurselor pentru a păstra performanța algoritmului.

Aceste resurse includ obiectul secvențiator creat și unitatea de transport a notelor.

#### Vizualizare grafică a unui waveform

Algoritmul de vizualizare a unui *waveform* presupune redarea undelor sonore preluate din obiectul de tip player audio pe o planșă, pentru a putea fi observate de utilizator.

După ce fișierul mp3 este încărcat în player și este apăsat butonul de redare, algoritmul începe prin a crea un *canvas* HTML, în cadrul căruia vor fi desenate și animate frecvențele sonore ale înregistrării. De asemenea, alte variabile de interes vor fi inițializate, variabile necesare procesării vizuale a sunetului.

Este folosit și un obiect de tip analizator care transformă în timp real sursa audio în informații cu privire la frecvențe. Pentru a fi realizat acest lucru, sursa trebuie convertită într-un obiect *Uint8Array.*

            audioSrc = context.createMediaElementSource(audioElement);

            analyser = context.createAnalyser()

            canvas = document.getElementById('waveform-canvas');

            canvasCtx = canvas.getContext("2d");

            analyser.fftSize = 2048;

            audioSrc.connect(analyser);

            audioSrc.connect(context.destination);

            bufferLength = analyser.fftSize;

            dataArray = new Uint8Array(bufferLength);

            audioElement.src = URL.createObjectURL(selectedFile)

            canvasCtx.clearRect(0, 0, WIDTH, HEIGHT);

Funcția care realizează desenarea efectivă a undelor sonore este una de tip animație în timp real. Folosind informațiile extrase de analizator, se desenează linii de vizualizare ce reprezintă amplitudinea frecvenței active. Astfel, este redat *waveform*-ul semnalului, ce reprezintă forma graficului în funcție de timp.

            function draw() {

                requestAnimationFrame(draw);

                analyser.getByteTimeDomainData(dataArray);

                canvasCtx.fillStyle = 'rgb(218, 218, 218)';

                canvasCtx.fillRect(0, 0, WIDTH, HEIGHT);

                canvasCtx.lineWidth = 2;

                canvasCtx.strokeStyle = 'rgb(0, 0, 0)';

                canvasCtx.beginPath();

                var sliceWidth = WIDTH \* 1.0 / bufferLength;

                var x = 0;

                for (var i = 0; i < bufferLength; i++) {

                    var v = dataArray[i] / 128.0;

                    var y = v \* HEIGHT / 2;

                    if (i === 0) {

                        canvasCtx.moveTo(x, y);

                    } else {

                        canvasCtx.lineTo(x, y);

                    }

                    x += sliceWidth;

                }

                canvasCtx.lineTo(canvas.width, canvas.height / 2);

                canvasCtx.stroke();

            };

Algoritmul se oprește la apăsarea butonului de stop de către utilizator, informațiile fiind menținute totuși în *buffer* în căzul reluării procesului de redate.

Obiectul player audio folosit este specific *Web Audio API*, cât și analizatorul.

Motivația folosirii acestora din cadrul *Web Audio API*, și nu *Tone.js* o reprezintă lipsa compatibilității *framework-*ului *Tone.js* cu funcționalități *canvas* HTML.

Astfel, o primă încercare de implementare s-a dovedit a fi un eșec, biblioteca *Tone.js* necesitând dezvoltare ulterioară în vederea îmbinării cu diverse API-uri sau funcții web.

Exemplu *waveform*:



# Capitolul 4: Implementare

## API

Structura API-ului este formată din *routes, models* și *middlewares.*

În *models* sunt descrise tabelele din baza de date, cât și relațiile dintre acestea. Baza de date are drept componență două tabele, unul în care se află datele utilizatorilor și altul ce conține sunetele salvate de aceștia.

Tabelul cu utilizatori are drept câmpuri numele de utilizator și *hash*-ul parolei (pe lângă câmpurile de bază create de *MySQL*, mai exact data cand s-a creat înregistrarea, data când a fost ultima oara modificată, id-ul cheie primară)

Tabelul cu sunete conține câmpurile: numele instrumentului, numele sunetului, datele de configurație în format JSON, efectele sonore, proprietarul și indicatorul de partajare.

De asemenea, pentru a lega sunete de utilizatorul proprietar, s-a implementat o relație *one-to-many* între o înregistrare de tip utilizator și înregistrări de tip sunet, cu ajutorul unei chei străine *UserID*.

# Capitolul 5: Manual de utilizare

# Capitolul 6: Concluzii

# Bibliografie

# Anexe