

UNIVERSITATEA “ALEXANDRU IOAN CUZA” DIN IAȘI  
FACULTATEA DE INFORMATICĂ



LUCRARE DE LICENȚĂ

**Acces la informații din domeniul muzical via un  
mashup Web bazat pe microservicii**

propusă de

**Ciripescu Teodor**

**Sesiunea:** *iunie/iulie, 2021*

**Coordonator științific**

**Conf. Dr. Buraga Sabin-Corneliu**

UNIVERSITATEA “ALEXANDRU IOAN CUZA” DIN IAȘI  
FACULTATEA DE INFORMATICĂ

**Acces la informații din domeniul muzical via  
un mashup Web bazat pe microservicii**

**Ciripescu Teodor**

**Sesiunea:** *iunie/iulie, 2021*

**Coordonator științific**

**Conf. Dr. Buraga Sabin-Corneliu**

# Cuprins

<b>Copertă .....</b>	<b>1</b>
<b>Titlu .....</b>	<b>2</b>
<b>Cuprins .....</b>	<b>3</b>
<b>Introducere.....</b>	<b>4</b>
<b>Capitolul 1: Fundamentele aplicației .....</b>	<b>6</b>
<b>Concepte, structuri de date și algoritmi.....</b>	<b>6</b>
<b>Tehnologii .....</b>	<b>8</b>
Backend .....	8
Frontend .....	9
<b>Biblioteci adiționale .....</b>	<b>10</b>
Backend .....	10
Frontend .....	11
<b>Capitolul 2: Scopurile și cerințele aplicației .....</b>	<b>12</b>
<b>Capitolul 3: Proiectare și analiză .....</b>	<b>15</b>
<b>Design.....</b>	<b>15</b>
<b>Interacțiunea cu utilizatorul .....</b>	<b>17</b>
<b>Arhitectură software .....</b>	<b>19</b>
<b>Structuri de date.....</b>	<b>22</b>
<b>Capitolul 4: Implementare .....</b>	<b>24</b>
<b>Structura și funcționalitățile proiectului.....</b>	<b>24</b>
Backend .....	24
Frontend .....	28
<b>Performanță .....</b>	<b>30</b>
<b>API.....</b>	<b>31</b>
<b>Capitolul 5: Manual de utilizare.....</b>	<b>33</b>
<b>Studiu de caz: utilizare pe platforme desktop.....</b>	<b>33</b>
<b>Studiu de caz: utilizare pe platforme mobile .....</b>	<b>39</b>
<b>Concluzii.....</b>	<b>42</b>
<b>Bibliografie.....</b>	<b>44</b>

## Introducere

Căutarea informațiilor referitoare la un anumit subiect rareori se oprește cu descoperirea unei prime surse relevante. De obicei, este necesară consultarea mai multor surse pentru a putea afla o informație completă. Nevoia de a reduce efortul și timpul căutării se regăsește în marea majoritate a domeniilor, inclusiv în cel al muzicii.

În momentul de față există o multitudine de platforme ce se ocupă cu distribuția digitală a muzicii, printre acestea, numărându-se *Spotify*, *Apple Music*, *SoundCloud* și *YouTube*. Aceste platforme sunt concentrate pe livrarea conținutului muzical către consumatorii lor și mai puțin pe detalii despre artist. De asemenea, pe lângă aceste platforme, au apărut și diverse comunități online ce se ocupă cu adunarea informațiilor referitoare la melodii, precum versuri și semnificația lor, anul de lansare al acestora și genul muzical. Astfel, au fost create adevărate enciclopedii din domeniul muzicii, cea mai mare dintre acestea fiind *Genius*.

Scopul de bază al proiectului este de a obține, centraliza și expune cât mai multe informații de interes referitoare la un artist și arta sa, din domeniul muzical, având ca surse principale marile enciclopedii și platforme de distribuție muzicală.

Astfel, soluția propusă constă într-o aplicație Web cu o structură bazată pe microservicii ce accesează API-urile publice a două platforme, Spotify și Genius, cu scopul de a agrega informații despre artiști, albume, melodii și topuri muzicale.

Aplicația este destinată atât platformelor *desktop*, cât și *mobile*, având o interfață interactivă și *responsive*.

Pentru dezvoltarea părții de *backend* a aplicației am folosit *Node.js*, *framework*-ul *Express* și *Mongoose*, iar pentru secțiunea de *frontend* am utilizat *React.js* împreună cu *Bootstrap*.

Ca funcționalități de bază, aplicația permite vizualizarea următoarelor tipuri de conținut:

- profiluri ale artiștilor, ce conțin informații generale despre artiști, poze cu aceștia, statistici referitoare la popularitate și numărul de urmăritori în funcție de timp, legături externe către paginile de rețele media sociale, topuri cu cele mai faimoase melodii și albume, legături externe către platformele unde pot fi ascultate melodiile lor și artiști similari
- liste interactive unde sunt prezentate toate albumele unui artist
- descrieri ale albumelor și ce piese cuprind acestea

- pagini de prezentare ale melodiilor, ce constau în poze, descrieri generale, legături către platformele unde se pot asculta acestea, scoruri de popularitate, videoclipuri muzicale, detalii audio și artiștii ce au contribuit.
- topul celor mai apreciate cincizeci de melodii la nivel global în funcție de o dată aleasă

Structura lucrării este formată din următoarele cinci capitole:

1. *Fundamentele aplicației*, este primul capitol și constă într-o descriere a conceptelor, structurilor de date, tehnologiilor și bibliotecilor adiționale folosite în lucrare.
2. *Scopurile și cerințele aplicației*, descrie ținta urmărită și privirea de ansamblu asupra soluției abordate.
3. *Proiectare și analiză*, clarifică etapele ingineresti ale aplicației dezvoltate, făcând referire la arhitectură *software*, structuri de date, *design* și interacțiunea cu utilizatorul.
4. *Implementare*, abordează structura și funcționalitățile proiectului, performanța și modul în care este realizat API-ul aplicației.
5. *Manual de utilizare*, este ultimul capitol și cuprinde două studii de caz referitoare la utilizarea aplicației pe platformele *desktop* și *mobile*.

# Capitolul 1: Fundamentele aplicației

## Concepte, structuri de date și algoritmi

Vizitarea frecventă a mai multor sit-uri Web cu scopul de a afla dacă conținutul a fost modificat sau dacă a apărut conținut nou poate dura mult timp. Nevoia de a reduce acest timp pierdut și efort necesar a dus la apariția agregatoarelor. Rezultatul tehnologiei de agregare poate fi un sit web ce conține într-o singură pagină informații noi sau actualizate adunate din diverse surse. Aplicația dezvoltată în cadrul lucrării este, prin definiție, un agregator. Aceasta urmărește să consolideze într-un singur loc informații obținute de la mai multe API-uri externe referitoare la artiștii din domeniul muzical.

Conceptul de Artist își face apariția de multe ori în lucrare, prin acesta înțelegându-se artiștii strict din domeniul muzical, și reprezintă principalul element cu care lucrăm în aplicație. Totul a fost dezvoltat cu o atenție înspre ideea de artist și ce reprezintă acesta. Fiecare cântăreț sau trupă are o descriere a sa și a carierei sale și o multitudine de albume și de melodii. Toate acestea pot fi accesate individual în aplicație, însă accentul se pune în prealabil pe artiști, toate celelalte informații centralizându-se în jurul acestora.

Se consideră de interes și cunoașterea conceptului de scor de popularitate. Fiecare artist și fiecare conținut muzical au asociate un scor ce reprezintă popularitatea acestora la timpul respectiv. Acest număr poate fluctua în funcție de diverși factori precum: artistul tocmai a lansat un album nou și a căpătat o atenție deosebită din partea ascultătorilor, sau, invers, a lansat o melodie ce a ajuns să fie sub așteptări. Această informație este pusă în valoare în cadrul aplicației și este obținută prin API-ul furnizat de *Spotify*.

Fiecare melodie prezentă în aplicație se deosebește printr-o serie de caracteristici:

- *Acousticness* ne indică dacă în melodie sunetul este unul obținut prin mijloace acustice, adică prin instrumente, sau dacă provine din mijloace digitale. Cu cât scorul este mai slab, cu atât mai multe sunete artificiale se regăsesc în piesă.
- *Danceability* descrie cât de potrivită pentru dans este o melodie.
- *Energy* reprezintă o măsură a intensității și activității. Piese energetice se simt rapide, puternice și zgomotoase.
- *Instrumentalness* indică dacă piesa are ca element principal aspectul instrumental sau aspectul vocal. Cu cât indicele este mai mic, cu atât cresc șansele ca melodia să nu prezinte conținut vocal.

- *Liveness* detectează prezența unui public în înregistrare. O valoare ridicată ne poate indica faptul că piesa a fost înregistrată în cadrul unui concert live.
- *Speechiness* detectează cuvintele rostite într-o piesă.
- *Valence* ne spune pozitivitatea muzicală pe care o piesă o transmite. Melodiile cu o valență ridicată sună mai pozitiv, în timp ce melodiile cu valență scăzută sună mai negativ.
- *Loudness* ne indică volumul general al piesei în decibeli.
- *Tempo* reprezintă ritmul estimat al melodiei în ritmuri pe minut. În muzică, tempo-ul este viteza sau ritmul unei piese și este obținut direct din durata medie a piesei.

Marea majoritate a aplicațiilor web moderne folosesc o arhitectură bazată pe microservicii, fiind principala alternativă la abordarea monolitică. O astfel de arhitectură permite construirea unei aplicații sub forma unui grup de servicii. Fiecare serviciu este de dimensiuni reduse, are rolul său și comunică cu restul aplicației folosind un protocol, în cazul nostru, *http*. Spre deosebire de abordarea monolitică, arhitectura pe microservicii este mai ușor de întreținut, componentele sale nu sunt la fel de puternic interconectate și pot fi modificate mai ușor, iar aducerea aplicației în mediul *cloud* este mult mai facilă.

Un concept necesar înțelegerii ideilor din spatele dezvoltării aplicațiilor Web cu Node.js este cel de *Promises* și de programare asincronă. Un *Promise* este un obiect ce poate produce o singură valoare în viitor. Această valoare nu este cunoscută în momentul în care un *Promise* este creat, ci atunci când procesul din spatele acestuia s-a finalizat cu un răspuns de tip succes sau eșec. Acest concept ne este folositor pentru cazurile în care vrem să ne folosim de programarea asincronă. Un bun exemplu este momentul în care încercăm să accesăm o resursă de la un API extern. Printr-un *Promise*, putem să așteptăm rezultatul și să asociem o funcție ce va fi executată în momentul în care s-a primit un răspuns înapoi. Astfel, putem paraleliza multiple apeluri către unul sau mai multe API-uri și să procesăm aceste răspunsuri în momentul în care apelurile au fost finalizate. Aceste noțiuni oferă dinamism aplicației, evitându-se situațiile în care aplicația rămâne blocată așteptând răspunsul unui singur apel.

Atât pe partea de *backend*, cât și pe cea de *frontend*, se lucrează intens cu Obiecte JavaScript. Acestea au un format extrem de similar cu structurile de date de tip JSON, ce ne sunt întoarse ca răspuns în urma apelurilor către API-urile publice realizate pe parcursul rulării aplicației. Același tip de structură este folosit și în API-ul nostru intern, atât în comunicarea dintre servicii cât și în comunicarea dintre client și *endpoint*.

## Tehnologii

Tehnologiile de bază ce facilitează realizarea și implementarea lucrării sunt *Node.js* și *React.js*, iar limbajul folosit predominant este *JavaScript*.

În continuare vom aborda în mod separat tehnologiile folosite în vederea dezvoltării secțiunilor de *backend*, respectiv *frontend*.

### Backend

Pe partea de *backend*, avem o arhitectură bazată pe microservicii, implementată folosind *JavaScript* împreună cu *Node.js* și framework-ul *Express*.

*JavaScript* este un limbaj popular, simplu și rapid. Este unul dintre cele mai folosite limbaje în tehnologiile web, rulând inițial în mod exclusiv pe partea de *browser*. Odată cu creșterea în popularitate a limbajului, s-a născut dorința de a rula *JavaScript* și pe partea de *backend*. Așa și-a făcut apariția *Node.js*, o tehnologie bazată pe *V8 Engine* din *Google Chrome*, ce ne permite rularea de cod *JavaScript* în afara unui *browser*.

*Node.js* funcționează pe bază de operații I/O nebloccante, ceea ce permite procesare de cereri concurente. Astfel, cererile către server sunt procesate eficient și fără întârzieri, în mod asincron. Acesta caracteristică este de o importanță notabilă, întrucât aduce un spor de performanță ridicat comparativ cu tehnologiile ce procesează cereri în mod sincron, una câte una.

Fiind o tehnologie deosebit de îndrăgită, *Node.js* beneficiază de cel mai voluminos registru de biblioteci din lume. Una dintre aceste biblioteci este *Express*, care, în esență, facilitează dezvoltarea de aplicații *Web* și *API-uri*. În lucrare, vom folosi acest *framework* pentru a realiza structura serviciilor noastre. O parte din funcționalitățile importante oferite de *Express*, pe lângă posibilitatea de dezvoltarea a unui API REST, sunt: configurarea ușoară, definirea rutelor aplicației folosind metode *http*, prezența mai multor module *middleware* pentru efectuarea de sarcini suplimentare la cerere și răspuns, conectarea la baze de date MongoDB, servirea de fișiere statice și resurse și gestionarea ușoară a erorilor.

Alte doua componente cheie folosite în dezvoltarea aplicației sunt reprezentate de API-urile oferite de *Spotify* și *Genius*. Acestea ne ajută să agregăm toate informațiile



referitoare la artiști, albume și melodii, necesare pentru formarea unui profil complet al fiecărui artist în parte.

## Frontend

Pentru dezvoltarea interfeței grafice a aplicației, am decis să folosim *React.js*, o bibliotecă populară, *open-source*, întreținută de Facebook. Punctele forte ale *React* constau în posibilitatea de a crea componente *UI* reutilizabile și crearea de interfețe interactive, ce sunt actualizate eficient în momentul în care informațiile din pagină se modifică.

Precum am intenționat și în secțiunea referitoare la *backend*, *frontend*-ul aplicației permite o modularizare avansată și o ușoară întreținere și îmbinare a componentelor sale în diversele elemente *UI* întocmite pe parcursul dezvoltării.

## Biblioteci adiționale

### Backend

Pe lângă tehnologiile principale menționate, aplicația întrebuințează și alte câteva biblioteci adiționale.

Toate aceste biblioteci sunt obținute prin intermediul managerului de pachete *NPM*.

În *backend*, utilizăm biblioteci precum:

- *body-parser*
- *dotenv*,
- *http-status-codes*
- *mongoose*
- *node-cron*
- *spotify-web-api-node*

*Body-parser* ne ajută să parsăm într-un *middleware* conținutul cererilor ce ajung la server înainte ca acestea să fie gestionate.

*Dotenv* este o bibliotecă ce ne permite să folosim fișiere de configurări cu variabile de *environment*, precum cheia secretă utilizată la apelul unor API-uri publice, numele serviciului, port-ul la care rulează serviciul respectiv și datele de conectare la baza de date specifică serviciului. De asemenea, se pot folosi mai multe fișiere de configurări specifice modului de utilizare: dezvoltare sau producție.

*Http-status-codes* oferă o modalitate ușoară de returnare a codurilor de eroare *http* înapoi către client.

*Mongoose* este o bibliotecă *ODM(Object Data Modeling)* ce facilitează conexiunea și toate operațiile dintre backend și baza de date MongoDB. Prin aceasta putem extrage și salva diverse informații referitoare la artiștii din aplicație.

*Node-cron* este un modul ce ajută la planificarea de sarcini ce urmează a fi executate într-un moment clar definit. Se poate specifica de câte ori să fie rulat procesul și când ar trebui să se repete acesta. Este folosit, de exemplu, la agregarea repetitivă și periodică de date de la API-urile publice folosite și reîmprospătarea informațiilor de autentificare la aceste API-uri.

*Spotify-web-api-node* este o bibliotecă ce facilitează accesul ușor la API-ul oferit de Spotify, folosind metode bazate pe *Promises*. Astfel, ni se permite accesul la informații precum melodiile dintr-un album, cele mai ascultate melodii ale unui artist sau artiști similari cu cel căutat.

## Frontend

Partea *frontend* a aplicației se folosește, la rândul său, de câteva biblioteci adiționale, necesare pentru aspecte precum apelul la resurse din *backend* și componente grafice predefinite: butoane, bara de navigare, grafice, liste.

Bibliotecile folosite sunt:

- *Axios*
- *Bootstrap*
- *Recharts*

*Axios* este un client *http* similar cu API-ul *Fetch* din *JavaScript* nativ, iar rolul său este de a oferi o experiență îmbunătățită, comparativ cu *Fetch*. *Axios* are la bază concepte precum *Promises* și ne oferă posibilitatea de a ne folosi de *async* și *await* din *JavaScript* pentru un cod mai lizibil și performant. Astfel, prin *Axios* putem trimite cereri *http* la API *endpoint*-ul nostru într-un mod mai convenabil și organizat.

*Bootstrap* este un framework CSS *open-source* orientat către dezvoltarea de sit-uri și aplicații Web *responsive*. Acesta oferă numeroase șabloane HTML și CSS pentru elementele interfețelor grafice, precum butoane și formulare. Alegerea *Bootstrap* este motivată prin faptul că se integrează cu ușurință cu *JavaScript* și *React* și oferă opțiuni moderne din punct de vedere vizual de creare a interfețelor grafice *responsive*. Interfața ce va fi realizată cu această tehnologie va avea și un caracter familiar pentru potențialii utilizatori, *Bootstrap* fiind folosit în foarte multe aplicații Web populare. De asemenea, este unul dintre cele mai apreciate framework-uri UI din lume.

*Recharts* este o bibliotecă bazată pe *React* și *D3*. Scopul principal al acesteia este de a desena grafice în aplicații de tip *React* cu cât mai multă ușurință. Este o bibliotecă ce funcționează bine împreună cu *Bootstrap* și oferă abilitatea de a încadra graficele în interfețe *responsive*.

## Capitolul 2: Scopurile și cerințele aplicației

În acest capitol, se doresc conturarea imaginii de ansamblu a proiectului și clarificarea scopului acestuia.

Scopul de bază al proiectului este de a obține și expune cât mai multe informații de interes referitoare la un artist și arta sa, din domeniul muzical, prin mijloace diverse și surse diferite.

În acest moment, există o multitudine de platforme de *streaming* extrem de populare ce permit utilizatorilor lor să asculte albumele și melodiile artiștilor preferați. Aceste platforme sunt concentrate pe livrarea conținutului muzical către consumatorii lor și mai puțin pe informații referitoare la artist.

Pe lângă marile platforme de distribuție muzicală, au apărut și diverse comunități online ce se ocupă cu adunarea informațiilor referitoare la melodii, precum versuri și semnificația lor, anul de lansare al acestora și genul muzical. Astfel, au fost create adevărate enciclopedii din domeniul muzicii.

Artiștii, în general, sunt indivizi creativi care doresc să își expună arta către ceilalți oameni. În momentul de față, marea majoritate a artiștilor activi folosesc, mai mult sau mai puțin, diverse platforme pentru a își promova și lansa creațiile. Vizitarea constantă a tuturor platformelor pe care este prezent un artist de către fani, pentru a afla noi informații, este un proces deseori obositor și amețitor. Aplicația propusă dorește să aducă o soluție modernă la această necesitate și să expună informații orientate în mod principal către artist.

În soluția dezvoltată, au fost alese două surse principale din care vor fi agregate informațiile necesare. Una dintre surse este *Spotify*, o platformă extrem de populară de distribuție a conținutului muzical. Cealaltă sursă, poate mai puțin cunoscută, este reprezentată de *Genius*.

*Spotify* oferă acces la API-ul său către dezvoltatorii înregistrați pe platformă. Conținutul oferit constă într-o multitudine de informații referitoare la artiști, album și melodii. De asemenea, se pot accesa și diverse topuri muzicale din mai multe zone geografice. În aplicație, ne vom folosi de o mare parte din informațiile obținute de la *Spotify*, în vederea creării profilurilor artiștilor, a căutării acestora, cât și întocmirea de albume și informații despre melodii.

*Genius* este cea mai mare enciclopedie digitală de muzică, cu o comunitate de peste două milioane de contribuitori pasionați. Pe *Genius* contribuie atât oamenii obișnuiți, pasionați de arta unui cântăreț sau trupe, cât și artiștii. Contribuțiile aduse de utilizatorii *Genius* constau, în principal, în furnizarea de versuri și explicații referitoare la acestea. De asemenea, *Genius* se ocupă și cu listarea conturilor de rețele media sociale ale artiștilor, cu furnizarea de descrieri ale artiștilor și ale melodiilor, cât și legături externe la alte platforme de *streaming* unde pot fi ascultate melodiile. Toate aceste informații vor fi complementare celor provenite de la Spotify, ajungând astfel la un produs minim viabil ce ne poate descrie cât mai bine viziunea asupra soluției propuse. Astfel, prin alegerea adecvată a surselor, se urmărește obținerea unei forme cât mai dezvoltate a ideii propuse.

Aplicația urmărește să aducă cele mai noi informații către utilizator. Un bun punct de plecare în experiența oferită către client este prezentarea unui top muzical, dinamic, în funcție de timp. Acesta trebuie să conțină cele mai apreciate melodii dintr-o zi, iar utilizatorul să aibă opțiunea de a naviga prin topurile din alte zile din trecut. Astfel, vom putea să afișăm atât cele mai noi informații, cât și să le consultăm pe cele ce nu mai sunt de actualitate, pentru a contura un profil evolutiv al muzicii de top. Aceste topuri trebuie să fie interactive, să conțină legături către melodii și artiști ce au contribuit la ele și să ofere un scor orientativ de popularitate. De asemenea, este indicat ca momentul colectării informațiilor să fie la final de zi, moment în care topul se află în forma sa finală, după eventualele modificări suferite pe parcursul zilei .

Toate componentele aplicației trebuie să fie interconectate pentru a oferi o experiență cât mai fluidă utilizatorului. Această fluiditate constă în faptul că pagina fiecărui artist trebuie să conțină legături către melodii, albume și artiști similari care să ghideze utilizatorul către cât mai multe informații de interes, fără să plictisească.

În ceea ce privește conținutul paginii unui artist, acesta este format, în mare parte, dintr-o poză de prezentare relevantă artistului, o descriere a acestuia și a genurilor muzicale abordate, un scor de popularitate, legături către paginile sale pe rețelele media sociale, o listă cu cele mai populare melodii, o listă cu cele mai populare albume, cât și alți artiști similari. Fiecare artist va avea asociată și o pagină cu albumele pentru o navigare mai ușoară. De asemenea, cei mai populari artiști vor beneficia și de statistici referitoare la indexul de popularitate și urmăritori, în funcție de timp. Aceste informații sunt agregate împreună cu cele de la topurile muzicale zilnice.

Prezentarea albumelor unui artist va fi făcută prin implementarea unui sistem de paginare, prin care un utilizator să poată naviga cu ușurință. Fiecare pagină va conține o listă cu albume. Pe lângă titlul albumului și o legătură către acesta, elemente din listă pot conține atât o imagine reprezentativă a albumului, cât și câteva etichete cu informații de interes.

Fiecare album în parte va avea o pagină proprie de prezentare. Aceasta va include coperta albumului, informații referitoare la album, un scor de popularitate al albumului, artiștii ce au contribuit și lista numerotată a melodiilor. Ca și la celelalte componente, se pune accentul pe prezența legăturilor interne dintre artist, album și melodie.

Fiecare melodie dintr-un album va avea o pagină cu toate informațiile necesare. Printre acestea, se numără: o imagine specifică, o descriere a melodiei, detalii legate de data de lansare, albumul pe care se află, legături către multiple platforme de *streaming* pe care se poate asculta melodia, caracteristici audio și artiștii ce au contribuit la realizarea piesei.

Deși problema pe care proiectul încearcă să o rezolve nu este una neapărat nouă, putem spune că s-a încercat crearea unei soluții diferite față de cele deja existente pe piață.

Toate ideile prezentate în acest capitol reprezintă punctul principal de plecare în dezvoltarea aplicației, dorind să clarifice ținta pe care vrem să o atingem și modul în care proiectăm implementarea acesteia.

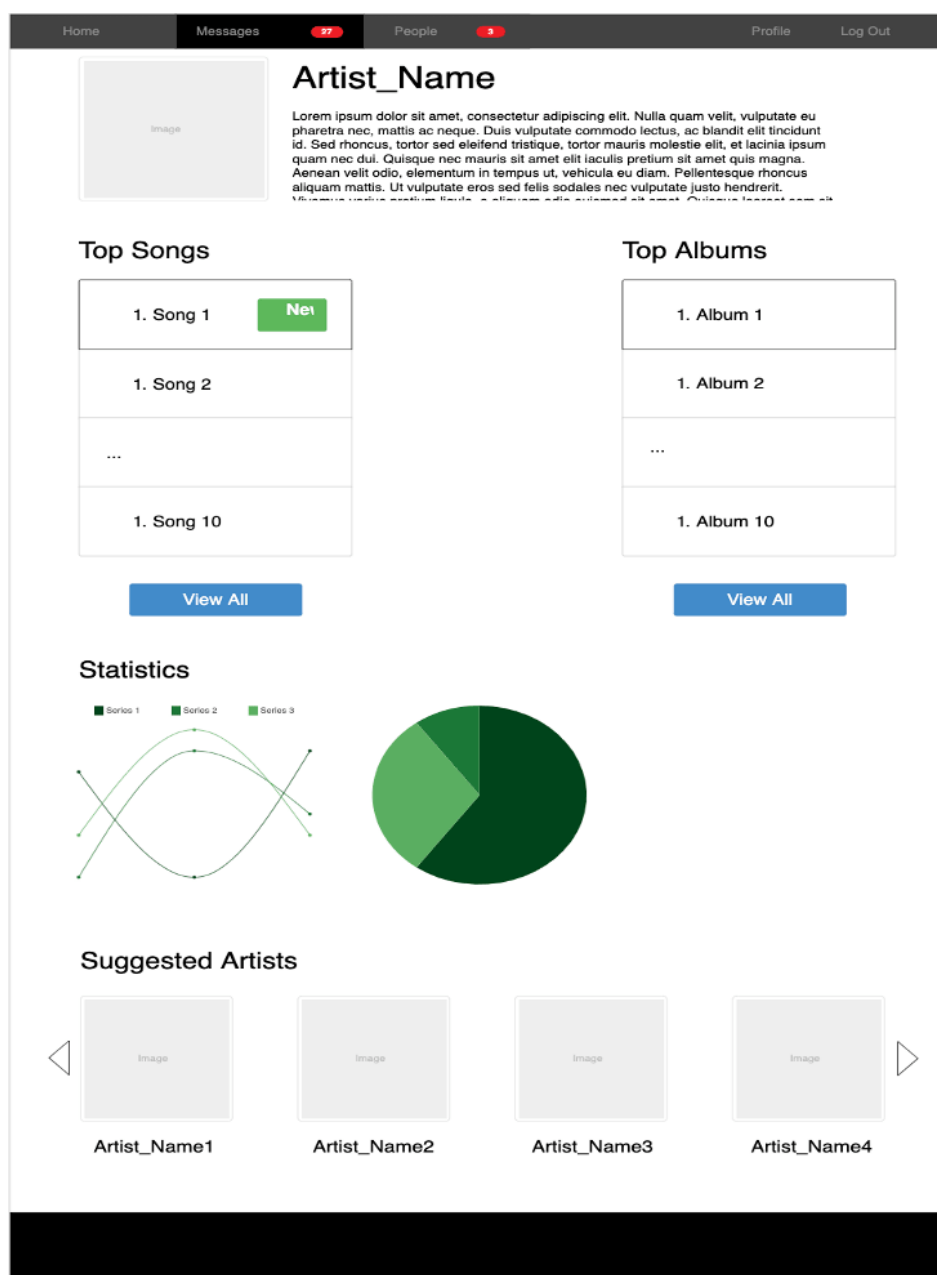
## Capitolul 3: Proiectare și analiză

### Design

Aplicația a fost proiectată cu elemente de *design responsive*, având ca scop acomodarea cât mai multor dimensiuni de ecrane.

Publicul țintă folosește atât platforme *mobile*, cât și *desktop*, pentru a asculta muzică și pentru a căuta informații referitoare la artiștii preferați. Acest fapt face esențială capacitatea aplicației de a se comporta *responsive* în interacțiunea cu utilizatorul.

Figură 1 Mockup



În stadiile incipiente ale proiectării aplicației, am realizat un *Mockup* pe care să îl folosim drept punct de plecare în ceea ce privește aranjarea în pagină și elementele de design. Acesta prezintă componente similare celor utilizate în varianta finală.

Fiecare pagină conține un *navbar*, element care facilitează accesul la paginile principale ale aplicației și totodată găzduiește bara de căutare. Acesta are o formă diferită în funcție de platforma pe care este rulată aplicația. De exemplu, pe variantele *mobile*, *navbar*-ul se extinde și își dezvăluie conținutul abia atunci când utilizatorul apasă un buton specific. Pe variantele *desktop*, acesta este afișat direct, cu toate elementele sale.

Un alt element aflat pe fiecare pagină este *footer*-ul, ce este situat în josul paginii.

Secțiunea dintre *navbar* și *footer* cuprinde conținutul principal al paginii.

Primele elemente cu care utilizatorul este întâmpinat pe paginile artiștilor, albumelor, sau melodiilor este o imagine sugestivă, împreună cu o descriere scurtă. Apoi, în funcție de pagină, pot fi vizualizate liste de albume și melodii, statistici, topuri, artiști relevanți și câteva etichete asociate resurselor respective.

Zonele comune tuturor paginilor sunt în mare parte colorate cu alb și negru, împreună cu un efect de „umbră” în spatele elementelor din pagină. Alegerea de a nu folosi culori impunătoare în interfața de bază a aplicației este motivată de dorința de a pune în lumină pozele artiștilor, copertile albumelor și a melodiilor. De asemenea, pentru a evidenția informațiile de interes, unele pagini de prezentare ale melodiilor adoptă un fundal parțial colorat, într-o nuanță reprezentativă și complementară pozei albumului.

Culoarea textului este neagră, iar fontul folosit în aplicație este *Helvetica Neue*, cel prestabilit de *Bootstrap*.



## Interacțiunea cu utilizatorul

Aplicația este gândită pentru a putea fi folosită de un public cât mai larg, motiv pentru care s-a evitat crearea de funcționalități care să necesite cunoștințe speciale pentru utilizare.

Paginile aplicației conțin elemente comune construite după anumite șabloane. De exemplu, pagina artistului X va avea același format ca pagina artistului Y. Acest fapt oferă un sentiment de familiaritate utilizatorului atunci când navighează prin aplicație, fiindu-i mult mai ușor să recunoască elemente din interfață și să identifice instinctiv cum ar trebui să se comporte acestea în momentul în care va interacționa cu ele.

Pentru recunoașterea mai ușoară a paginilor, se disting următoarele șabloane:

- Artist, constituie structura paginilor cu artiști din aplicație. Pagina artistului conține o imagine cu acesta, numele lui, un scor de popularitate, genurile muzicale pe care le abordează, numărul de urmăritori pe Spotify, legături către paginile pe rețelele media sociale, o descriere a artistului, un top cu zece cele mai populare melodii, scorurile lor de popularitate și legături către platformele unde se pot asculta, un top cu albumele artistului, o legătură către pagina cu albumele artistului și o secțiune de artiști similari sugerați de aplicație. Mai mult, în cazul celor mai populari artiști sunt prezente și grafice cu statistici referitoare la indexul de popularitate și numărul de urmăritori în funcție de timp.
- Albumele Artistului, reprezintă paginile pe care pot fi găsite toate albumele unui artist. Este prezent un mecanism de paginare prin care utilizatorul poate naviga și vizualiza liste cu albumele artistului. Fiecare pagină conține până la douăzeci de albume afișând titlul, numărul de melodii de pe album și o poză cu albumul.
- Album, constituie o pagină de prezentare a unui album împreună cu lista de melodii numerotate ce se regăsesc în album. În descrierea albumului sunt notate data de lansare, numărul de melodii, casa de discuri, drepturile de autor și un scor de popularitate.
- Melodie, reprezintă structura paginilor pe care se află melodiile din aplicație. Fiecare melodie are o poză, un titlu, o dată de lansare, o durată, un scor de popularitate, un număr ce semnifică poziția în album, legături către platformele de *streaming*, caracteristici audio și artiști, producători și scriitori ce au contribuit. De asemenea, se regăsesc o descriere a melodiei, o legătură externă către versurile ei și un videoclip muzical.
- Top Muzical, este șablonul prin care sunt afișate listele cu cele mai populare cincizeci de melodii la nivel global. Pagina conține și un calendar interactiv pe care utilizatorul îl poate folosi pentru a alege ziua din care vrea să vizualizeze topul.

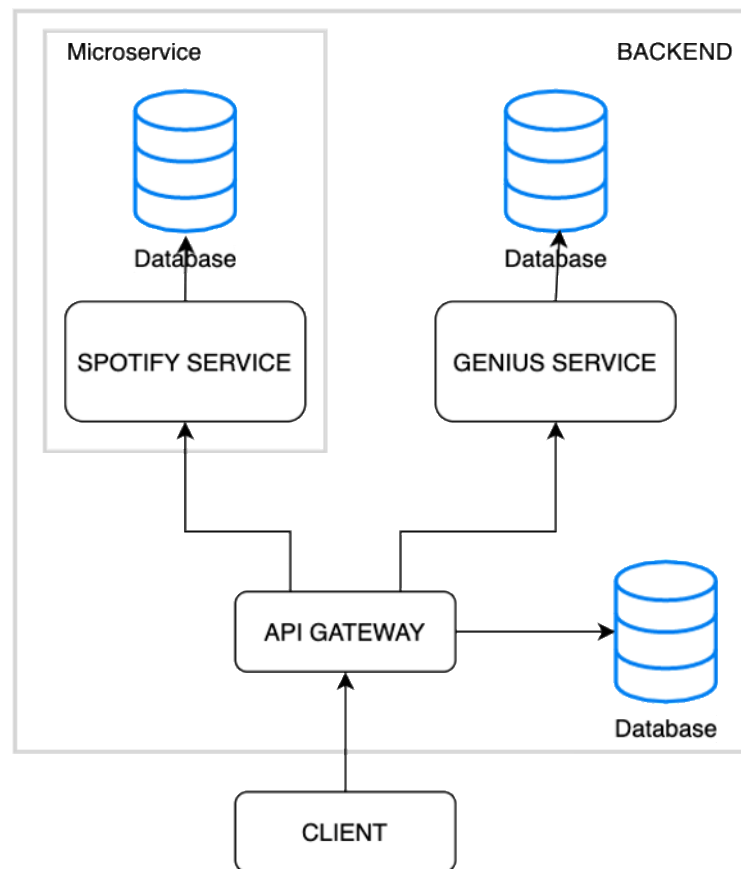
- Pagina de căutare, reprezintă lista cu artiști întoarsă în urma căutării, prin intermediul barei de căutare, a unui artist. Pentru fiecare artist din lista rezultată sunt afișate poza, numele și scorul de popularitate.

Pentru o mai bună experiență pentru utilizatori, au fost gândite elemente care să sugereze că aplicația este în curs de încărcare în momentele în care se așteaptă un răspuns de la *backend*. În acest sens, au fost dezvoltate animații de încărcare, atât pentru întreaga pagină, cât și pentru acele componente din pagină a căror încărcare necesită mai mult timp.

## Arhitectură software

În această secțiune vom prezenta modul de implementare a structurii pe microservicii a proiectului și vom analiza avantajele acestei abordări comparativ cu structura monolitică.

Figură 2 Diagrama Arhitectura pe Microservicii



Un aspect ce a ușurat procesul de dezvoltare este alegerea arhitecturii ce are la bază microserviciile.

Din câte se observă din diagramă, avem câteva componente principale:

- Client
- API Gateway
- Spotify Service
- Genius Service

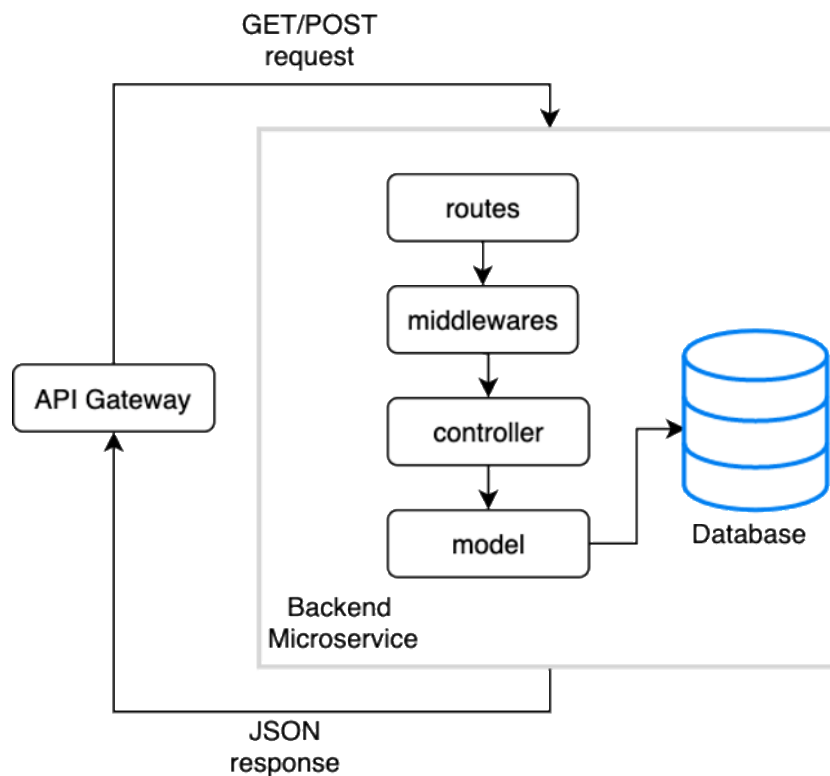
Cursul aplicației funcționează după următorii pași: clientul face o cerere la API Gateway, apoi, acesta, la rândul său, distribuie cerea către cele două servicii, Spotify Service și Genius Service. Cele două servicii întorc câte un răspuns înapoi la API Gateway, care formulează un alt răspuns bazat pe cele primite. În cele din urmă, API Gateway întoarce la Client un răspuns final al cererii inițiale.

Fiecare serviciu are o bază de date proprie, pe care doar el o poate accesa. Alegerea de nu a folosi o singură bază de date pentru tot proiectul este dată de faptul că se dorește o modularizare cât mai bună. Fiecare serviciu va putea fi rulat într-un mediu separat, fără să interfereze cu celelalte, și să funcționeze independent.

Vorbind despre un agregator, nevoia de a adăuga o nouă sursă de informații în aplicație poate apărea oricând. Prin faptul că lucrăm cu servicii, ne este foarte ușor să adăugăm încă unul și să îl adaptăm în structura aplicației. Întrucât serviciile au o structură asemănătoare, integrarea unei noi surse din care să agregăm informații nu ar trebui să fie o problemă dificilă. Dacă am fi avut o abordare monolitică a proiectului, adăugarea unei noi funcționalități de acest gen ar fi devenit un proces mult mai anevoios, prin simplul fapt că toate componentele ar fi fost interconectate.

O privire mai apropiată asupra legăturii dintre API Gateway și celelalte servicii:

*Figură 3 Comunicarea între API Gateway și Microserviciu*



Structura unui microserviciu din *backend* este realizată după modelul MVC, singura componentă ce lipsește fiind *View*, ce este necesară doar în cazul aplicațiilor cu interfață grafică.

De asemenea, trebuie menționat că structura API Gateway este extrem de similară cu cea a celorlalte microservicii. Scopul său principal este de a unifica informațiile primite de la celelalte microservicii și de a le servi clientului.

Pe partea de *frontend*, arhitectura este una clasică, specifică aplicațiilor construite cu *React.js*, bazată în mare parte pe principiile de pagini și componente. Astfel, se poate modulariza codul foarte ușor și să se refolosească componentele în mai multe pagini din aplicație.

## Structuri de date

Aplicația dezvoltată lucrează intens cu formatul de date JSON. Atât intern, în formatul datelor transmise între microservicii și client, cât și extern, între microservicii și API-urile furnizate de Spotify și Genius, se utilizează date în format JSON.

S-a ales ca soluție de stocare MongoDB, o bază de date orientată pe documente, ce face parte din familia bazelor de date de tip NoSQL. Este alegerea naturală pentru aplicațiile ce lucrează predominant cu date în format JSON.

În aplicația dezvoltată se apelează periodic API-uri externe. În mod implicit, nu avem control asupra resurselor acestor API-uri, care pot suferi modificări fără înștiințare. Spre exemplu, dacă Spotify decide să adauge sau să șteargă câmpuri din răspunsurile întoarse de către API, sau să modifice tipul de date al informațiilor transmise, pot apărea probleme în momentul în care vrem să salvăm aceste informații în baza de date. O bază de date relațională nu ar reuși să se descurce cu ușurință la astfel de cerințe. MongoDB, în schimb, permite scheme dinamice de date, ceea ce ne oferă flexibilitate în cazurile în care structura datelor nu este una fixă și garantată.

Exemplu de schemă MongoDB cu valoarea câmpurilor mixtă:

```
{
  "id": {
    "type": "String"
  },
  "artists": {
    "type": [
      "Mixed"
    ]
  }
}
```

La baza aplicației funcționează un sistem de depozitare a informațiilor primite de la API-urile externe pe care le apelăm. Scopul stocării este unul multiplu. Pe lângă faptul că stocarea internă a informațiilor va aduce un spor de performanță în momentul în care urmărim să reaccesăm informații, dorim să avem, de asemenea, acces la cât mai multe date cu caracter istoric pe care să le refolosim în alte scopuri.

Acest sistem de *cache* se dovedește folositor și din alt motiv destul de important. Spotify și Genius au o limită la numărul de apeluri pe care le putem face către serviciile lor în decursul unui minut. Asta înseamnă că, dacă am avea un număr de ordinul zecilor de

utilizatori, riscăm ca informațiile să nu ajungă la aceștia, din cauza limitărilor descrise mai sus.

## Capitolul 4: Implementare

### Structura și funcționalitățile proiectului

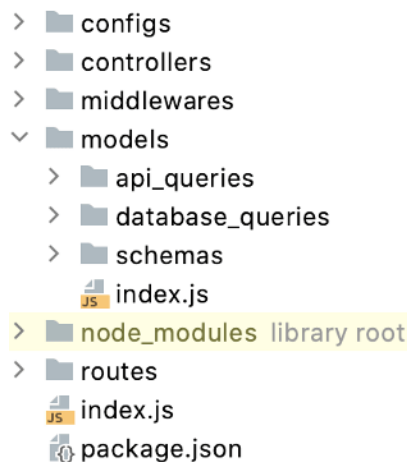
#### Backend

În *backend*-ul aplicației rulează trei microservicii de bază:

- Spotify Service
- Genius Service
- API Gateway

Toate cele trei microservicii beneficiază de o structură comună. Principalele motive ce fac baza alegerii unei structuri comune sunt: ușurința în dezvoltarea proiectului, întreținerea mai facilă a codului sursă și dorința de a putea extinde funcționalitățile aplicației prin introducerea de noi microservicii similare, cum ar fi, de exemplu, adăugarea unei noi surse de informații pe care să o agregăm.

Figură 4 Structură microserviciu



În *configs* se găsesc două fișiere de configurări, unul pentru dezvoltare și unul pentru producție. Acestea conțin constante referitoare la numele *microserviciului*, portul la care rulează, și informații de autentificare la API-uri externe și baze de date.

*Models* găzduiește atât procedurile și funcțiile ce lucrează cu baze de date și API-uri, cât și schemele de documente din baza de date.

În *routes* găsim toate rutele API-ului din microserviciu. Aici se decide structura API-ului, iar fiecare *endpoint* are asociat câte un controller ce se va ocupa cu cererile primite de la client.

Directorul *controllers* conține funcțiile ce captează cererile de la client, apelează funcții din *models* și trimite răspunsuri în format JSON la client.



În *Middlewares* se găsesc funcțiile ce fac verificări și prelucrări ale cererilor primite de la client până ca acestea ajung în controller.

*Node\_modules* este un director ce conține toate bibliotecile folosite în cadrul microserviciului.

Fișierul *package.json* conține *metadata* referitoare la proiect și ce dependențe are acesta.

În continuare, vom prezenta funcționalitățile modulelor pentru fiecare microserviciu în parte.

### *Spotify Service*

Acest *microserviciu* se ocupă cu agregarea informațiilor de pe Spotify. Când *microserviciul* este pornit, se realizează o conexiune cu baza de date și încep două rutine: una ce reîmprospătează o dată pe oră *token*-ul folosit pentru accesarea API-ului Spotify, și alta ce agregă date referitoare la top cincizeci de melodii la nivel global. Această agregare are loc o dată pe zi, la finalul zilei, și captează atât lista melodiilor din top, cât și o statistică referitoare la numărul de urmăritori și popularitatea artiștilor ce se regăsesc în top.

Toate rutele ce pot fi accesate încep cu prefixul */api* și apoi continuă cu denumiri specifice: */artist*, */album*, */track*, */stats/top50global*.

În momentul în care se vor face cereri către API-ul microserviciului, clientul trebuie să asigure prezența unor parametri specifici fiecărei resurse, cum ar fi numele sau numărul de identificare de pe Spotify al artistului, albumului sau melodiei dorite. Fiecare resursă de pe Spotify are un *id* unic pe care îl vom folosi și în aplicația noastră.

În *controllere* se vor extrage parametrii din cererile primite, și vor fi folosiți în apelul funcțiilor din *models*, care accesează API-ul public de la Spotify și care efectuează operații pe baza de date. Aceste funcții sunt organizate în două submodule, *api\_queries* și *database\_queries*.

*Api\_queries* conține proceduri precum: *getArtist*, *getArtistByName*, *getArtistAlbums*, *getArtistTopTracks*, *getRelatedArtists*, *searchArtists*, *getAlbum*, *getTrack* și *getTrackAudioFeature*. Toate aceste proceduri accesează și prelucrează informațiile oferite de Spotify cu scopul de a le trimite către *frontend* și de a le stoca în baza de date.

În *database\_queries* se fac operații de tip CRUD peste colecțiile de documente din baza de date. Conținutul documentelor face referire la informațiile ce au fost agregate de pe Spotify. Printre aceste colecții se numără: *artists*, *albums*, *tracks*, *top\_albums*, *top\_artists*, *top\_tracks*, *related\_artists* și *top50songs*.

Portul la care rulează microserviciul este 3500.

### *Genius Service*

*Genius Service* va aduce informații complementare celor agregate de pe *Spotify*. Asemănător microserviciului din urmă, în momentul pornirii, *Genius Service* își asigură o conexiune la baza de date și fixează un *token* de autorizare în *header*-ul fiecărei cereri către API-ul public de la *Genius*.

Rutele API-ului încep cu prefixul */api* și continuă cu */artist* sau */track*. Numele parametrilor acceptați sunt *name* și *id*, ce fac referire la un artist sau o melodie căutată. *Id* este un cod de identificare unic specific resurselor *Genius API*.

Structura modulelor cât și conținutul acestora este foarte similar cu cel de la *Spotify Service*. Diferența constă în faptul că aici apelăm la API-ul celor de la *Genius*, și implicit, schemele documentelor din baza de date sunt diferite.

Un aspect demn de a fi menționat este modul în care ajungem să extragem informațiile despre artiști. Fiind o platformă orientată spre conținutul melodiilor, *Genius* nu posedă o modalitate de căutare directă a unui artist după nume. În schimb, suntem nevoiți să facem o căutare după melodii și să extragem prima melodie ce îl are ca autor pe artistul pe care încercăm să îl găsim. În acest punct, avem acces la codul de identificare al artistului și putem accesa informații despre acesta prin apelul altei cereri către API. Din fericire, această căutare ineficientă va fi necesară doar pentru artiștii ce nu au fost încă agregați de aplicație.

Portul la care rulează microserviciul este 4500.

### *API Gateway*

Acest *microserviciu* are rolul de punct principal de comunicare dintre *frontend* și *backend*. Rutele sale sunt identice cu cele de la *Spotify Service* și *Genius Service*, având ca rol principal servirea către clienți a tuturor informațiilor agregate referitoare la un artist sau o melodie. Astfel, dacă primește o cerere referitoare la un anumit artist, *API Gateway* va apela la *Spotify Service* și *Genius Service* și va face o reuniune a informațiilor primite.

Acest *microserviciu* de unificare a informațiilor se dovedește deosebit de folositor atunci când se dorește adăugarea unei noi surse de informare în aplicație.

O funcționalitate interesantă ce merită prezentată este crearea topului de melodii al unui artist. Dorim să agregăm primele zece cele mai apreciate melodii ale unui artist împreună cu legături externe ale acestora la platformele de *streaming*. Un prim pas este apelarea API-ului microserviciului *Spotify Service* cu GET la */api/artist/top\_tracks*

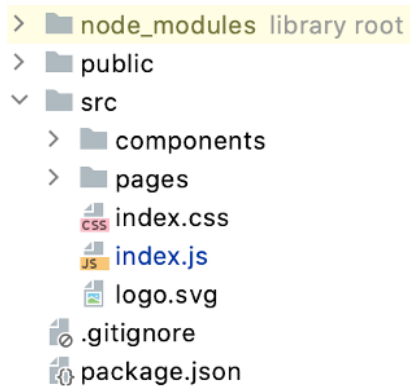
împreună cu unul dintre parametrii *name* sau *id*. Resursa primită va consta în lista cu cele zece melodii, dar fără legături la platformele de *streaming*. Pentru a obține și restul informațiilor, vom folosi API-ul microserviciului *Genius Service*. Construim un *array* cu *Promises* ce conține pentru fiecare melodie în parte un apel de tip GET la */api/track* împreună cu parametrul *name*, un șir de caractere cu numele artistului și al melodiei. Folosim un *array* de *Promises* cu scopul de a face toate apelurile în mod concurent, scăzând astfel timpii de așteptare. După ce am primit toate răspunsurile, avem acces la legăturile externe către platformele de *streaming* pentru fiecare melodie în parte. Tot ce mai rămâne de făcut este să construim un obiect care să conțină reuniunea informațiilor primite de la cele două microservicii, și să îl trimitem ca răspuns clientului.

Microserviciul API Gateway rulează la portul 3030.

## Frontend

Partea de *frontend* a aplicației are o structură formată dintr-un director *src* ce conține componente UI și pagini ce folosesc aceste componente, un director *public* ce conține diverse resurse care să fie servite clienților, un alt director *node\_modules* cu bibliotecile folosite și un fișier *package.json* cu *metadata* specifice proiectului.

Figură 5 Structură frontend



Componentele și paginile sunt formate din câte două fișiere, un fișier CSS și un fișier JavaScript.

În proiect avem următoarele componente: *Header*, *Navbar*, *Footer*, *ArtistPreview*, *ArtistThumbnail*, *ArtistDescription*, *SuggestedArtists*, *LineChartStats*, *ArtistAlbumsList*, *LoadingSpinner*, *AlbumThumbnail*, *AlbumDescription*, *AlbumTracks*, *AlbumArtists*, *SearchArtistList*, *Top50GlobalList*, *TopAlbums*, *TopSongs*, *TrackArtists*, *TrackAudioFeatures*, *TrackDescription*, *TrackProducerWriterArtists*, *TrackThumbnail*.

De asemenea, avem următoarele pagini:

- *App*, pagină principală ce descrie rutele aplicației și paginile care se găsesc la acestea.
- *Artist*, conține toate informațiile referitoare la un artist.
- *ArtistAlbums*, o pagină cu toate albumele unui artist.
- *Album*, cuprinde lista melodiilor unui album și informații despre acesta.
- *Track*, pagină de prezentare a unei melodii.
- *SearchArtists*, reprezintă pagina cu rezultatele din urma căutării unui artist folosind bara de căutare.
- *Top50Global*, pagină cu topul celor mai apreciate cincizeci de melodii la nivel global.

Pentru a accesa informații referitoare la artiști, albume, melodii și topuri, paginile fac apel la API Gateway.

O funcționalitate interesantă ce merită menționată este procesul prin care pagina de *Artist* obține resursele necesare din *backend*. Pentru a se popula toate componentele

din pagină cu informații, trebuie făcute patru cereri către *backend*. Primul *request* ce trebuie făcut este GET la */api/artist* cu parametrii *name*, *spotifyId* sau *geniusId*. În urma acestui *request* obținem toate informațiile de bază referitoare la un artist, printre care, și *id*-urile unice ale acestuia ce ne vor ajuta să facem următoarele trei cereri către *backend*. În continuare, se vor face următoarele *request*-uri în mod concurent: GET */api/artist/top\_tracks* cu parametrul *spotifyId*, GET */api/artist/albums* cu parametrii *spotifyId* și *page*, și GET */api/artist/related* cu parametrul *spotifyId*. Răspunsurile la toate aceste *request*-uri vor încarcă în pagină următoarele componente: ArtistThumbnail, ArtistDescription, LineChartStats, TopSongs, TopAlbums și SuggestedArtists. De asemenea, componentele au o animație de încărcare ce este activată pe durata așteptării răspunsurilor.

## Performanță

Apelurile constante la API-uri publice pentru a agrega informații pot duce la timpi ridicați de încărcare a paginilor și la epuizarea cotei de apeluri pe care aceste API-uri o alocă aplicației noastre. Pentru a evita situațiile în care utilizatorul așteaptă prea mult pentru încărcarea paginii, sau, mai rău, situațiile în care lipsesc informații din pagină, am implementat un sistem de *cache*.

Sistemul se asigură că fiecare resursă în format JSON ce poate fi accesată prin API-ul aplicației este salvată în baza de date. Această salvare are loc în anumite condiții și după efectuarea în prealabil a unor verificări. De fiecare dată când un client face o cerere asupra unei resurse din backend, se verifică dacă această resursă se află în baza de date.

Dacă resursa nu se află în baza de date, atunci aceasta este construită cu ajutorul API-urilor externe(*Spotify* și *Genius*) și adăugată, împreună cu două *timestamp*-uri: unul cu data la care a fost creată înregistrarea și unul cu data ultimei modificări. În final, resursa este trimisă clientului.

Dacă resursa se află deja în baza de date, atunci se verifică data ultimei modificări. În cazul în care vechimea resursei depășește un anumit prag, se apelează API-uri externe pentru a o reconstrui, iar apoi se actualizează în baza de date. În schimb, dacă pragul nu este depășit, resursa este servită direct ca răspuns clientului.

## API

API-ul proiectului are ca rol principal servirea de resurse referitoare la artist, albume, melodii și topuri muzicale.

Resursele pot fi accesate folosind metoda GET la următoarele rute:

- [/api/artist](#), parametri: *name*, *spotifyId*, *geniusId*  
*Name* reprezintă numele artistului căutat, iar *spotifyId* și *geniusId* sunt coduri de identificare ale artistului pe *Spotify* și *Genius*. Resursa poate fi accesată și în condițiile în care doar unul dintre acești parametri este prezent.  
Conținutul resursei va avea în componență informații despre artist, agregate de pe *Spotify* și *Genius*, cât și statistici referitoare la popularitate și număr de urmăritori.
- [/api/artist/top\\_tracks](#), parametri: *spotifyId*  
Resursa conține primele zece cele mai apreciate melodii ale artistului. Pe lângă informații de bază precum titlu, autori și durată, avem legături externe către platforme unde se pot asculta piesele. Aceste platforme sunt: *Spotify*, *YouTube*, *SoundCloud* și *Apple Music*.
- [/api/artist/albums](#), parametri: *spotifyId*, *page*  
Resursa conține o secțiune din catalogul de albume al artistului. Se poate naviga prin catalogul de albume folosind parametrul *page*.
- [/api/artist/related](#), parametri: *spotifyId*  
Constă într-o listă cu artiști similari pe care *Spotify* îi recomandă.
- [/api/artist/search](#), parametri: *name*  
Rezultatul este o listă cu artiști ce au numele similar cu valoarea parametrului *name*.
- [/api/album](#), parametri: *spotifyId*, *page*  
Conține informații despre un anumit album împreună cu lista de piese de pe acesta. Parametrul *page* este util pentru navigarea prin albumele cu mai mult de cincizeci de melodii.

- [/api/track](#), parametri: *name*, *spotifyId*, *geniusId*  
Constă în informații referitoare la o anumită melodie. *Name* este un parametru folosit pentru căutarea unei melodii în cazurile în care parametrii *spotifyId* și *geniusId* lipsesc.
- [/api/stats/top50global](#), parametri: *date*  
Va conține o listă cu top cincizeci de melodii la nivel global dintr-o anumită zi. Ziua poate fi specificată prin parametrul *date*, de tip dată calendaristică. Pentru cazul special în care se dorește cel mai recent top, *date* poate avea valoarea *latest*.

Toate resursele API-ului au două *timestamp*-uri: unul cu momentul la care a fost creată resursa, și altul cu momentul în care a fost actualizată ultima oară.



## Capitolul 5: Manual de utilizare

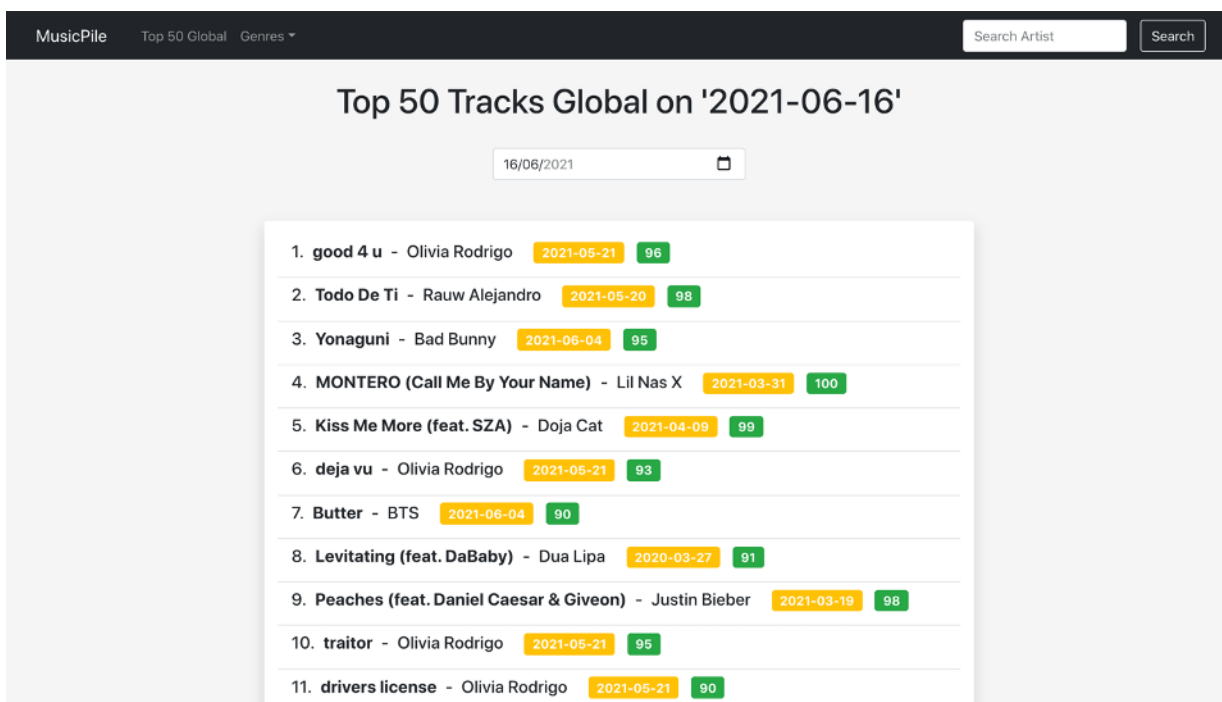
În acest capitol se vor prezenta și analiza modalitățile de utilizare ale aplicației pe baza a două studii de caz.

Primul studiu de caz va consta în interacțiunea unui utilizator *desktop* cu aplicația, iar cel de-al doilea, în folosirea aplicației de pe dispozitive *mobile*. Fiecare studiu va încerca să prezinte moduri diferite de utilizare a soluției oferite.

### Studiu de caz: utilizare pe platforme *desktop*

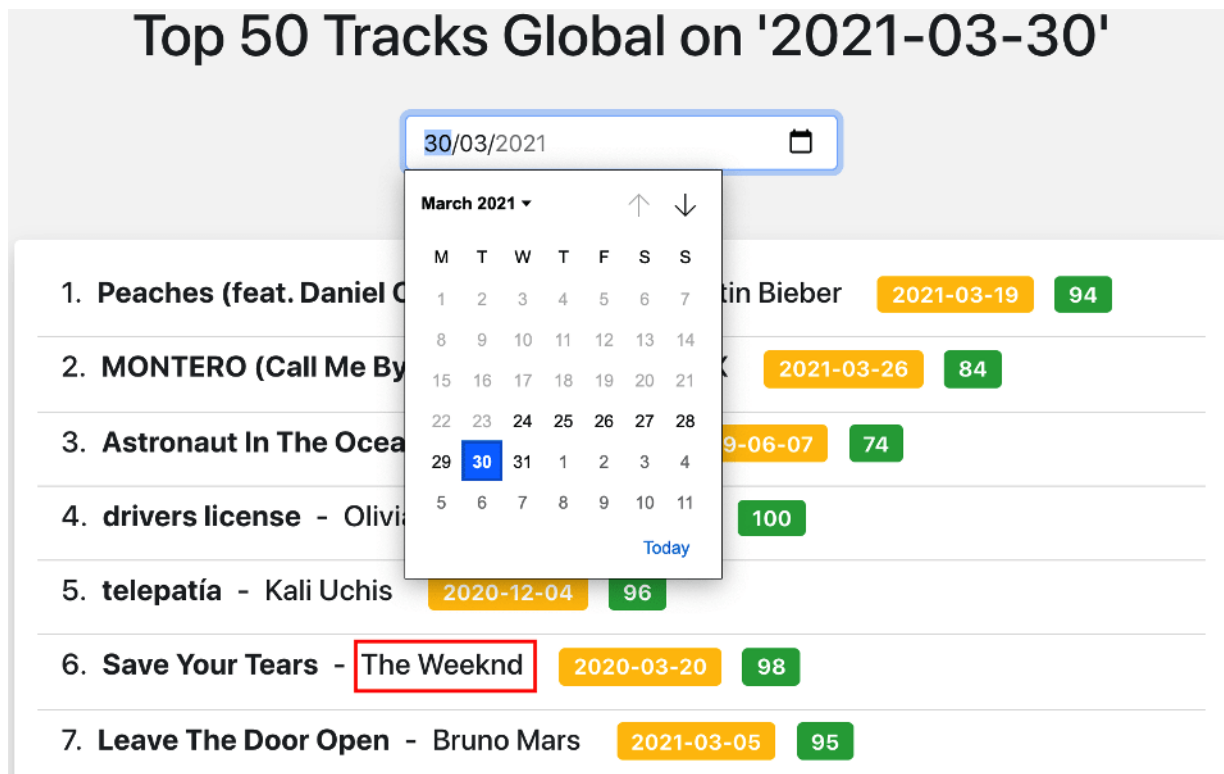
Vom lua în calcul un utilizator nou ce dorește să descopere funcționalitățile de bază ale proiectului. Când acesta intră pentru prima oară în aplicație, este întâmpinat cu o pagină ce conține topul celor mai apreciate cincizeci de melodii din ziua respectivă.

Figură 6 Top 50 Melodii



În cele ce urmează, utilizatorul se va folosi de calendarul interactiv din aplicație pentru a vizualiza un istoric al acestui top. Selectând o dată din calendar, lista de melodii se reîmprospătează, fără ca pagina să se reîncarce.

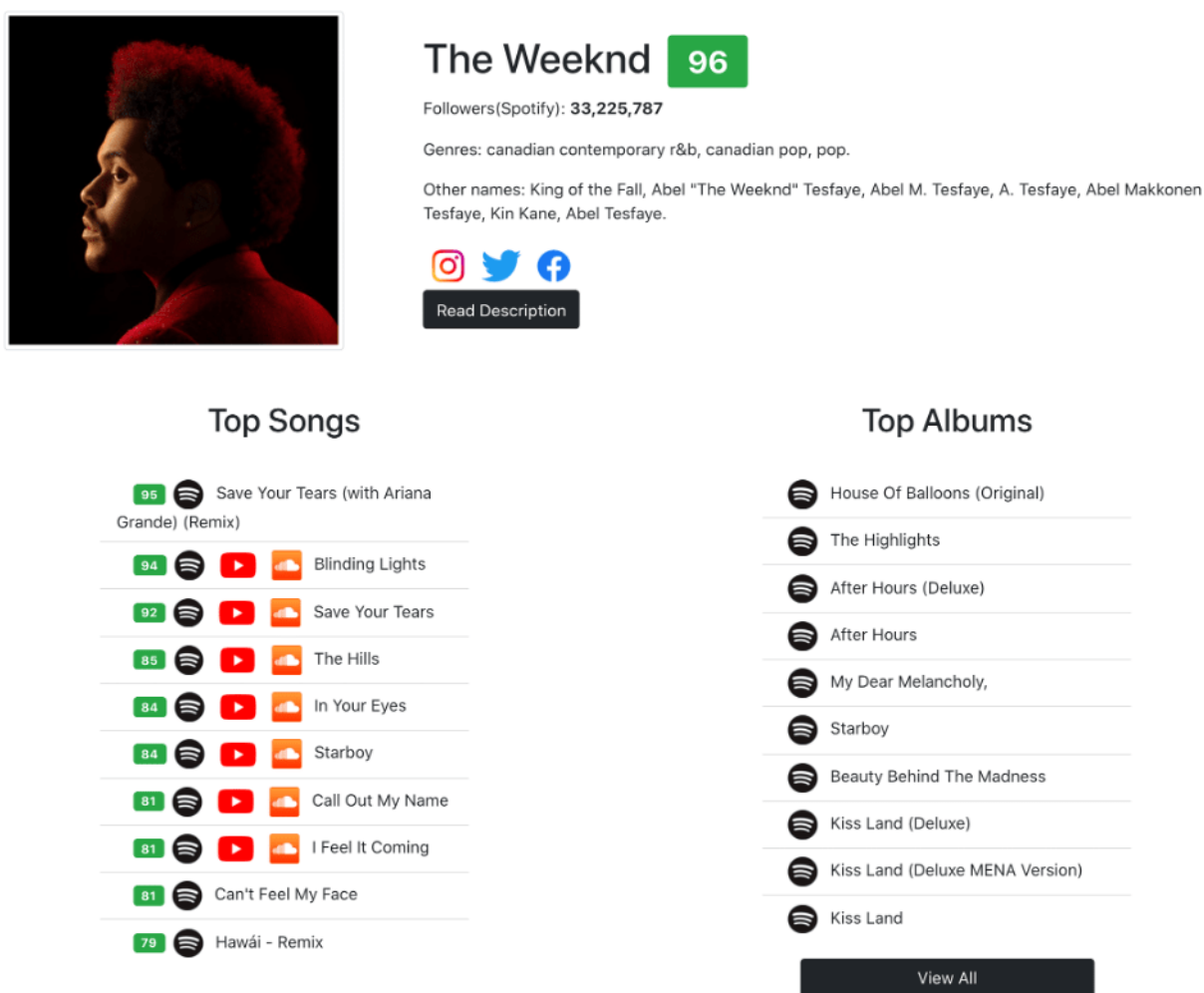
Figură 7 Vizualizare top după data calendaristică



Utilizatorul alege să vizualizeze topul din data de 30 martie, iar apoi selectează autorul melodiei *Save Your Tears*.

Pagina cu profilul artistului *The Weeknd* este încărcată, mai multe elemente de interes făcându-și apariția. Printre acestea se numără: o poza cu artistul, numele său, un scor de popularitate, numărul de urmăritori, genurile muzicale pe care le abordează, alte nume ale acestuia, legături către paginile sale de pe rețelele media sociale, un buton ce deschide o descriere mai pe larg a artistului, câte un top cu cele mai cunoscute melodii și albume, și un buton către pagina cu toate albumele artistului.

Figură 8 Pagina Artistului



**The Weeknd** 96

Followers(Spotify): **33,225,787**

Genres: canadian contemporary r&b, canadian pop, pop.

Other names: King of the Fall, Abel "The Weeknd" Tesfaye, Abel M. Tesfaye, A. Tesfaye, Abel Makkonen Tesfaye, Kin Kane, Abel Tesfaye.

[Read Description](#)

### Top Songs

Rank	Score	Song
95	95	Save Your Tears (with Ariana Grande) (Remix)
94	94	Blinking Lights
92	92	Save Your Tears
85	85	The Hills
84	84	In Your Eyes
84	84	Starboy
81	81	Call Out My Name
81	81	I Feel It Coming
81	81	Can't Feel My Face
79	79	Hawái - Remix

### Top Albums

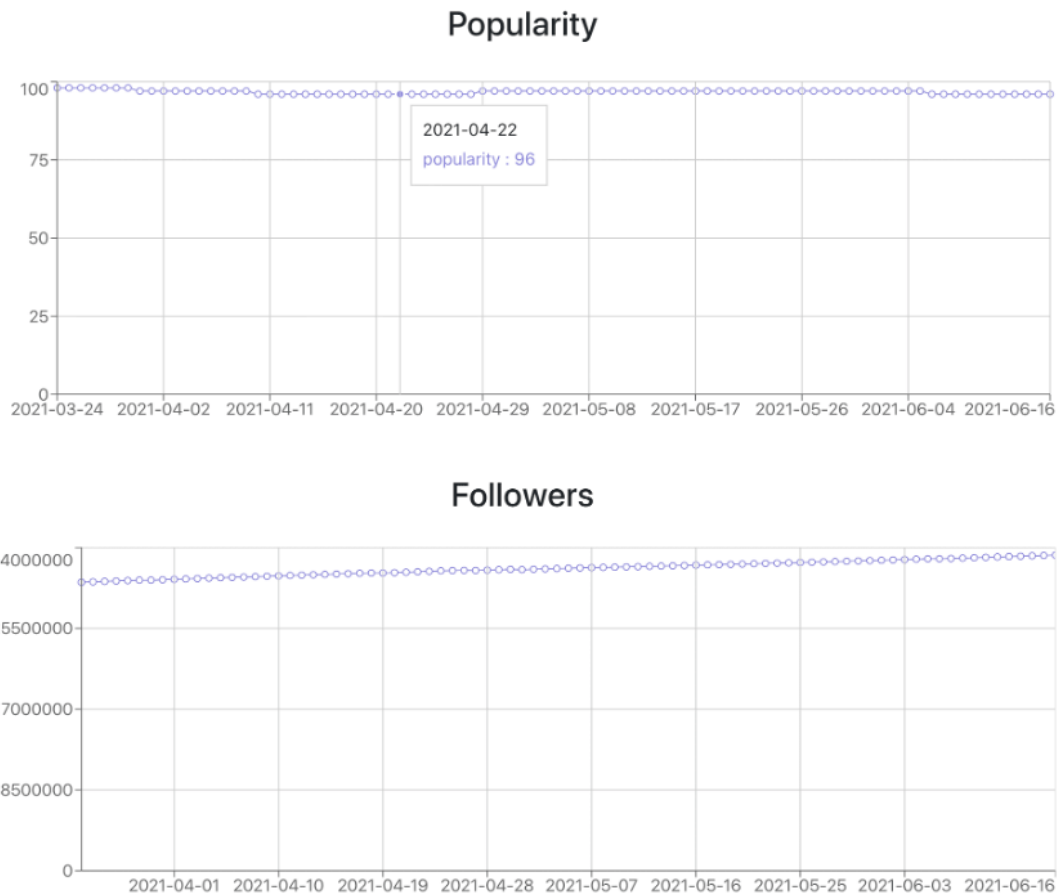
Album
House Of Balloons (Original)
The Highlights
After Hours (Deluxe)
After Hours
My Dear Melancholy,
Starboy
Beauty Behind The Madness
Kiss Land (Deluxe)
Kiss Land (Deluxe MENA Version)
Kiss Land

[View All](#)

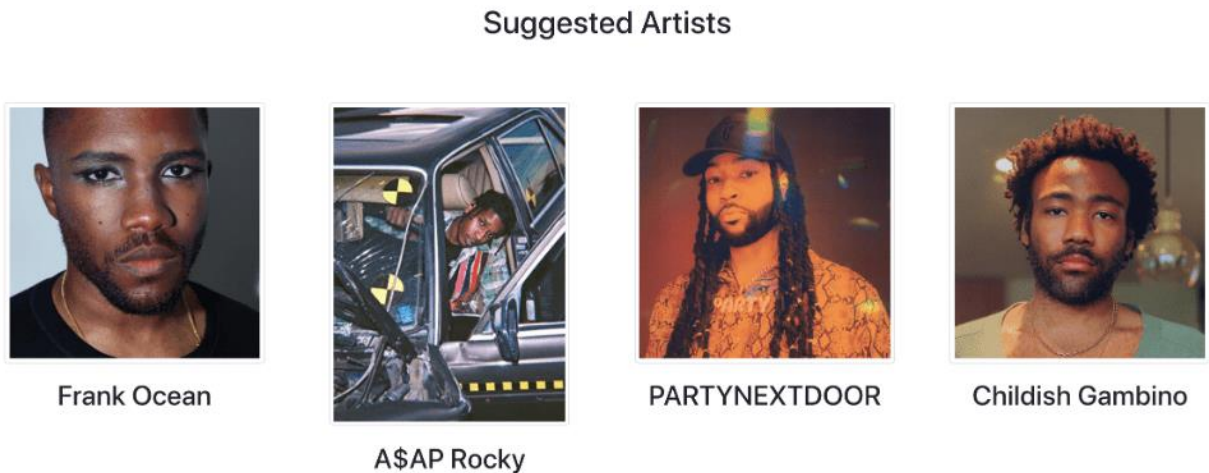
Se poate observa că melodiile de la secțiunea *Top Songs* au atribuite scoruri de popularitate și pictograme cu legături externe către platformele unde se pot asculta acestea. În mod similar, *Top Albums* prezintă o legătură externă către conținutul albumului.

Alte elemente ce se găsesc pe pagina de prezentare a artistului sunt graficele referitoare la evoluția popularității și numărului de urmăritori în funcție de timp, și un grup de artiști pe care aplicația îi sugerează.

Figură 9 Statistici ale artistului

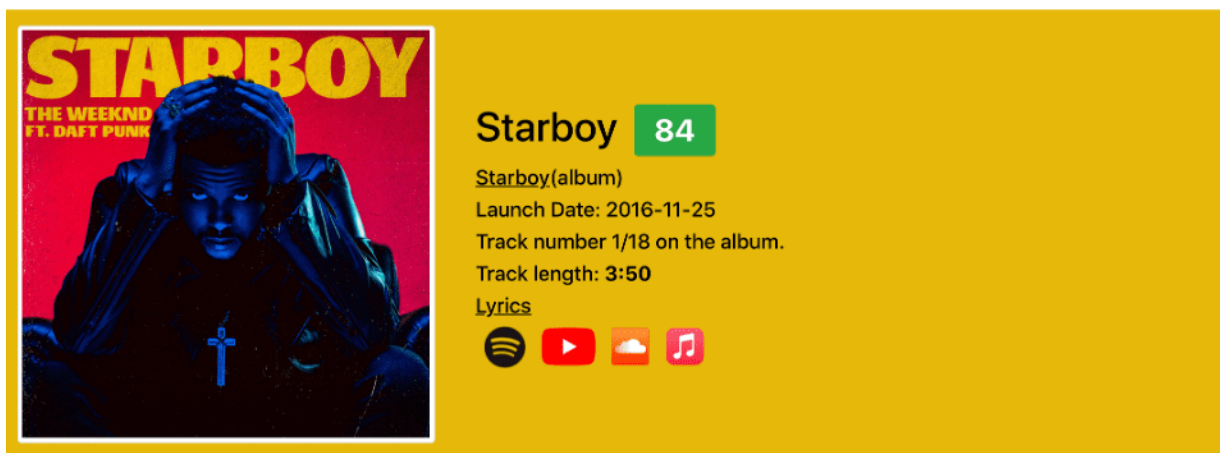


Figură 10 Artiști sugerați de aplicație



De asemenea, toate melodiile și albumele din topuri au legături interne către paginile de prezentare ale acestora. Astfel, utilizatorul alege una dintre melodiile din top, și o pagină cu aceasta se încarcă în aplicație.

Figură 11 Pagina melodiei



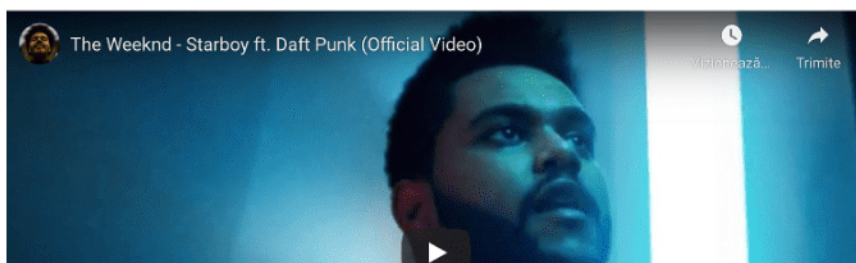
**Starboy** 84

Starboy(album)  
 Launch Date: 2016-11-25  
 Track number 1/18 on the album.  
 Track length: 3:50  
[Lyrics](#)

[Spotify](#) [YouTube](#) [Apple Music](#) [Other](#)

## Description

"Starboy" is The Weeknd's realization of his mega-stardom. Transforming from mysterious Canadian act to Billboard mainstay, Abel has come a long way, but with negative consequences in his journey to the top, he has developed mixed feelings towards fame. "Starboy" potentially marks a new era for The Weeknd. On the release date of the song, Abel posted an Instagram picture with the caption "r.i.p @abelxo," marking the turn of his character and sound. The song features production from French electronic duo Daft Punk, marking the first collaboration between The Weeknd and Daft Punk, and the latter's first single since "Give Life Back To Music" was released in early-2014. A month before release Abel and Daft Punk had spent time together in the studio, and it was obviously so fruitful he decided to make one of their collaborations the album's lead single. "Starboy" is the title track and lead single from The Weeknd's third studio album, released on November 25th, 2016.



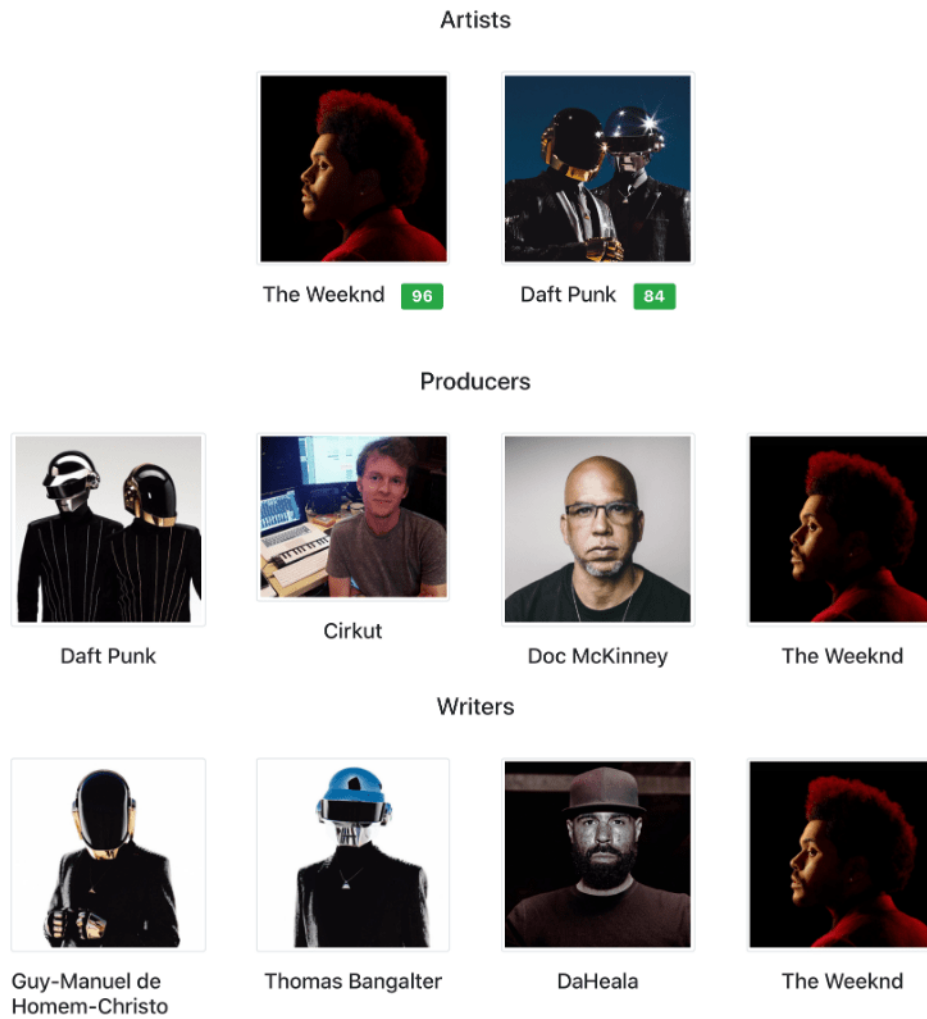
Audio Feature	Value
Acousticness	14.10%
Danceability	67.90%
Energy	58.70%
Instrumentalness	0.00%
Liveness	13.70%
Speechiness	27.60%
Valence	48.60%
Loudness	-7.015
Tempo	186.003

După un format similar cu cel al paginii artistului, pagina de prezentare a melodiei conține o poză, un titlu, un scor de popularitate, o legătură către pagina de prezentare a albumului din care provine, o dată de lansare, numărul de poziționare pe album, durata melodiei, o legătură externă la versuri, cât și alte legături către platformele pe care poate fi ascultată melodia.

Pe lângă acestea, se găsește o descriere a melodiei, un tabel cu statistici referitoare la caracteristici audio, un videoclip muzical și toți artiștii implicați în realizarea piesei.

Contribuitorii unei melodiei pot face parte din trei categorii: cântăreți, producători și scriitori.

Figură 12 Cântăreți, Producători și Scriitori ce au contribuit la piesă

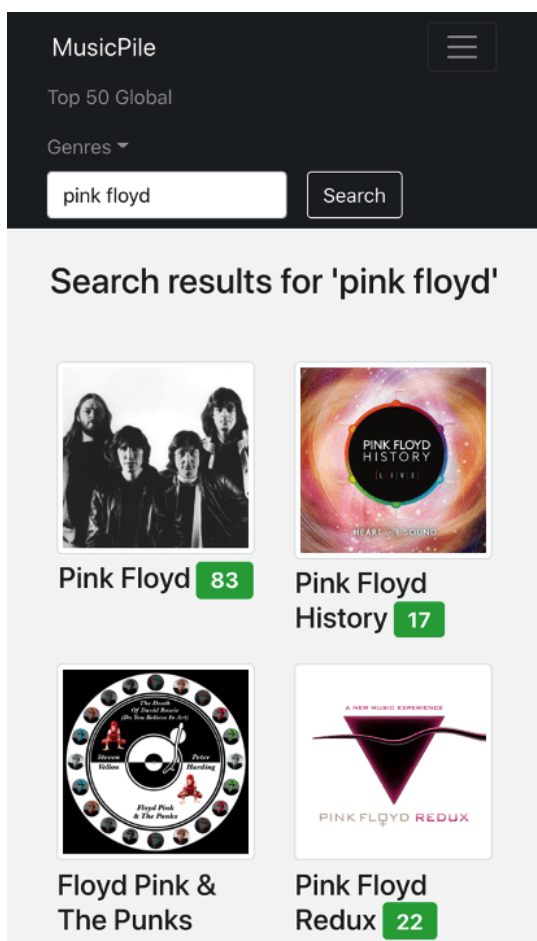


## Studiu de caz: utilizare pe platforme *mobile*

Vom lua în considerare un utilizator ce rulează aplicația pe o platformă *mobile*, cu un dispozitiv al cărui ecran este de dimensiuni reduse. În acest caz, interfața *responsive* a aplicației se dovedește a fi de folos.

Clientul are ca scop găsirea unui album ce aparține trupei *Pink Floyd*. Acesta începe prin a introduce numele trupei în bara de căutare situată în *navbar*-ul aplicației. După ce numele a fost introdus, se va apăsa butonul *Search*, iar pagina cu rezultate se va încărca.

Figură 14 Rezultatul unei căutări



Figură 13 Pagina trupei Pink Floyd



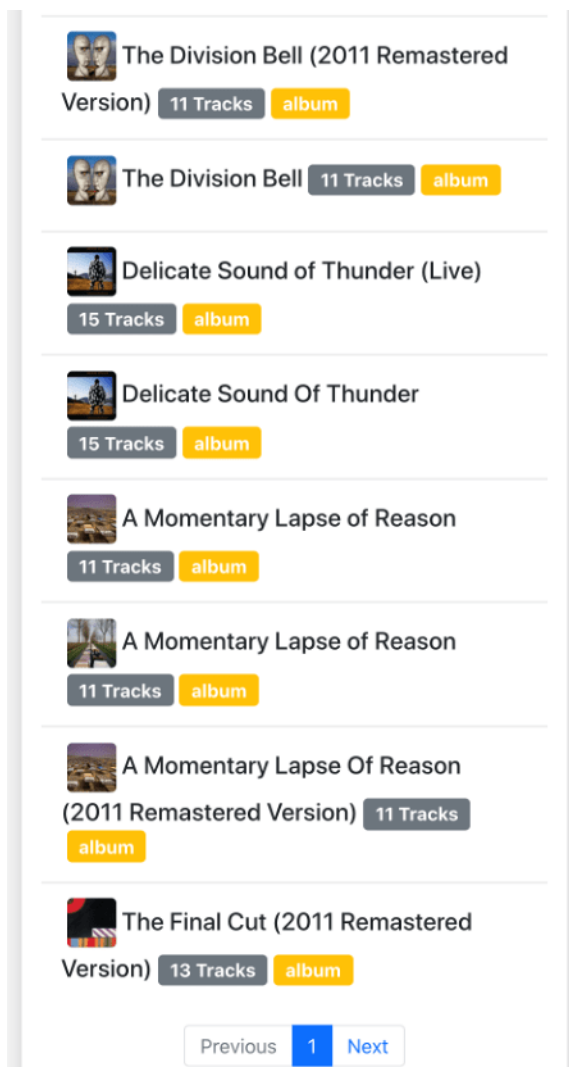
Utilizatorul apasă pe primul rezultat din căutări, iar pagina trupei se încarcă. Navigând pe pagină, se identifică butonul *View All* din dreapta secțiunii *Top Albums*, ce ne va redirecționa către pagina cu albumele artiștilor.



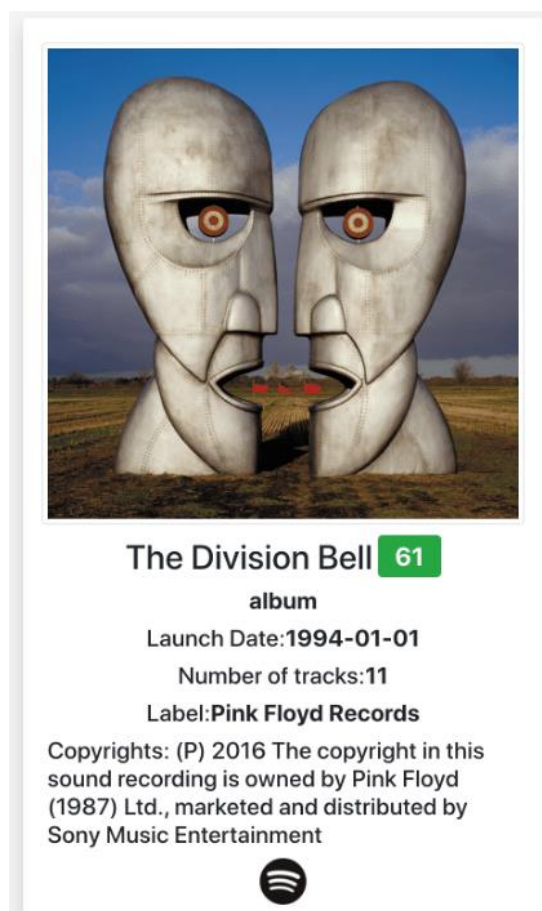
Ajuns pe pagina cu albume, utilizatorul poate naviga prin lista cu toate albumele trupei folosind butoanele de *Next* și *Previous*. Când acesta găsește albumul pe care îl dorește, va apăsa în dreptul acestuia.

După ce pagina de prezentare a albumului se încarcă, clientul poate să vadă informații referitoare la data de lansare, numărul de melodii de pe album, casa de discuri și drepturile de autor.

Figură 16 Listă cu albume



Figură 15 Pagină de album





Navigând mai jos pe pagina albumului, se poate găsi lista numerotată cu piese de pe acesta. Toate piesele au asociate câte o legătură internă la paginile lor de prezentare. De asemenea, utilizatorul este înștiințat dacă melodiile din listă au conținut *Explicit* sau nu, prin intermediul unei etichete situate lângă titlul acestora.

Figură 17 Melodii de pe album

1.Cluster One 5:55

2.What Do You Want from Me 4:21

3.Poles Apart 7:03

4.Marooned 5:30

5.A Great Day for Freedom 4:16

6.Wearing the Inside Out 6:49

7.Take It Back 6:12


8.Coming Back to Life 6:19

9.Keep Talking 6:10

10.Lost for Words 5:14 Explicit

11.High Hopes 8:30

Artists



Pink Floyd 83

## Concluzii

Orientând prezentarea spre un final sumativ, proiectul dezvoltat este o aplicație Web de tip agregator ce realizează profilul muzical al unui artist, prezentând o imagine de ansamblu asupra carierei și creațiilor sale, prin intermediul unor surse dinamice de informare și a unor modalități moderne de expunere a acestora.

Aplicația se axează pe oferirea de informații și legături relevante între acestea prin intermediul unui mediu interactiv. Resursele oferite constau în imagini, grafice și topuri interactive, texte descriptive, legături externe către rețele media sociale și conținut audio/video.

Fiecare pagină din aplicație are multiple legături interne către alte resurse de interes cu scopul de a fluidiza procesul de documentare și navigare prin aplicație al utilizatorului.

Conținutul oferit este îndreptat către artist și către albumele și melodiile sale. Utilizatorii aplicației pot accesa informații ce au la bază următoarele șabloane:

- artist
- albumele unui artist
- album
- melodie
- top muzical

Fiecare șablon va fi populat cu date din mai multe surse externe, precum *Spotify* și *Genius*, creând un profil complet al artistului, melodiei sau albumului.

Grație funcționalităților de agregare ale aplicației, putem vizualiza grafice referitoare la popularitatea și numărul de urmăritori al artiștilor în funcție de timp, dar și să analizăm topul celor mai apreciate cincizeci de melodii la nivel global, folosind un calendar interactiv.

Aplicația oferă posibilitatea de căutare directă a artiștilor folosind bara de căutare. Astfel, utilizatorii pot găsi un artist specific în mod simplu și rapid.

Toate resursele ce sunt construite folosind API-uri externe beneficiază de un sistem de *cache*, acestea fiind salvate și actualizate în baza de date a microserviciului ce le-a agregat.

Proiectul permite aducerea cu ușurință de noi funcționalități și modificări datorită arhitecturii pe bază de microservicii și a modului de structurare al acestora. Printre posibilele îmbunătățiri ce se pot aduce aplicației se numără:

- agregarea mai multor surse de informații, precum *Apple Music*, *MusicBrainz* sau *Discogs*
- adăugarea unor topuri de melodii specifice fiecărei țări în parte
- crearea unui sistem de căutare multicriterială

- dezvoltarea unui sistem prin care utilizatorii pot contribui la informațiile ce se găsesc pe paginile artiștilor

Astfel, oricine este pasionat sau curios cu privire la domeniul muzical, poate folosi aplicația pentru a descoperi cei mai populari artiști din ultimul timp și melodiile lor, dar și să reviziteze artiști și melodii deja cunoscute.

## Bibliografie

- [1] Marijn Haverbeke, *Eloquent Javascript, 3rd Edition: A Modern Introduction to Programming*, No Starch Press, 2018
- [2] David Flanagan, *JavaScript: The Definitive Guide: Master the World's Most-Used Programming Language*, O'Reilly, 2020
- [3] Nicholas C. Zakas, *Understanding EcmaScript 6: The Definitive Guide for JavaScript Developers*, No Starch Press, 2016
- [4] Eve Porcello & Alex Banks, *Learning React: Modern Patterns for Developing React Apps*, O'Reilly, 2020
- [5] Shannon Bradshaw, Eoin Brazil & Kristina Chodorow, *MongoDB: The Definitive Guide 3e: Powerful and Scalable Data Storage*, O'Reilly, 2019
- [6] Michael Thelin, *Spotify Web API Node*, <https://github.com/thelinmichael/spotify-web-api-node>
- [7] Faraz Kelhini, *How to make HTTP requests with Axios*, <https://blog.logrocket.com/how-to-make-http-requests-like-a-pro-with-axios/>
- [8] Jack Rhodes, *Data Visualisation in React — Part I: An Introduction to Recharts*, <https://medium.com/swlh/data-visualisation-in-react-part-i-an-introduction-to-recharts-33249e504f50>
- [9] Chris Coyier, *A Complete Guide to Flexbox*, <https://css-tricks.com/snippets/css/a-guide-to-flexbox/>
- [10] Jason Arnold, *Using dotenv package to create environment variables*, <https://medium.com/@thejasonfile/using-dotenv-package-to-create-environment-variables-33da4ac4ea8f>
- [11] Chris Nwamba, *How To Use node-cron to Run Scheduled Jobs in Node.js*, <https://www.digitalocean.com/community/tutorials/nodejs-cron-jobs-by-examples>
- [12] \* \* \*, *Express web framework (Node.js/JavaScript)*, [https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express\\_Nodejs](https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs)
- [13] \* \* \*, *Using Promises*, [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Using\\_promises](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Using_promises)
- [14] \* \* \*, *MongoDB*, <https://docs.mongodb.com/manual>
- [15] \* \* \*, *React Bootstrap*, Copyright (c) 2014-present Stephen J. Collings, Matthew Honnibal & Pieter Vanderwerff, <https://react-bootstrap.github.io/getting-started/introduction>
- [16] \* \* \*, *Bootstrap*, Copyright (c) 2011-2021 Twitter, Inc., <https://getbootstrap.com/docs>

[17] \* \* \*, *React: A JavaScript library for building user interfaces*, Copyright © 2021 Facebook Inc., <https://reactjs.org/docs>