UNIVERSITATEA DIN BUCUREŞTI

FACULTATEA DE MATEMATICĂ ŞI INFORMATICĂ

SPECIALIZAREA INFORMATICĂ

Lucrare de licenţă

VINYLLA

Aplicaţie web de gestionare a unui magazin online de viniluri

Absolvent

Pop Maria

Coordonator ştiinţific

Lect. Dr. Ana Cristina Iova

Bucureşti, septembrie 2023

Rezumat

Aplicația web „Vinylla” este o aplicație destinată gestionării unui magazin online de viniluri care își propune să aducă iubitorii de muzică mai aproape de pasiunea lor prin oferirea posibilității de a achiziționa sau colecționa albumele lor preferate, asigurând în același timp o experiență UI/UX plăcută utilizatorilor. Prin intermediul său, userii pot explora o gamă variată de produse, navigând eficient prin vasta colecție de viniluri datorită funcționalităților de căutare, sortare în funcție de titlu sau preț sau filtrare în funcție de diverse criterii, precum anul lansării și genul muzical în care se încadrează.

De asemenea, Vinylla pune un accent deosebit pe feedback-ul clienților, oferindu-le posibilitatea de a își împărtăși opinia despre un album prin intermediul recenziilor, o abordate ce oferă o perspectivă valoroasă viitorilor potențiali cumpărători ce permite luarea de decizii informate în ceea ce privește orice achiziție. Aplicația se deosebește prin faptul că este destinată atât clienților, cât și administratorilor, ceea ce îi oferă un plus de complexitate și funcționalitate. În ceea ce privește spațiul destinat administrării magazinului, acesta se concentrează pe asigurarea unei gestionări transparente și eficiente a întregului proces de vânzare și administrare a produselor.

Prin intermediul său, administratorii au acces la un set complet de instrumente menite să faciliteze gestionarea catalogului, având control asupra tuturor albumelor din stoc, codurilor de discount, conturilor utilizatorilor și rapoartelor de feedback ale acestora.

Prin urmare, aplicația destinată administratorilor vine ca o completare a aplicației de user, ceea ce face ca „Vinylla” să se caracterizeze prin complexitate, eficiență și versatilitate, devenind o soluție cuprinzătoare pentru pasionații de muzică și comercianții de viniluri.

Abstract

The web application "Vinylla" is designed to manage an online vinyl shop, aiming to bring music lovers closer to their passion by offering the opportunity to purchase or collect their favorite albums, while ensuring a pleasant UI/UX experience for users. Through this application, users can explore a diverse range of products, efficiently navigate through the extensive vinyl collection, thanks to search, sorting options based on title or price, and filtering based on various criteria such as release year and music genre.

Additionally, Vinylla places special emphasis on customer feedback, allowing them to share their opinions about an album through reviews. This approach provides valuable insights for potential buyers, enabling them to make informed decisions regarding their purchases.

What sets Vinylla apart is its dual focus on both customers and administrators, offering enhanced complexity and functionality. Regarding the administration space, the application ensures transparent and efficient management of the entire sales and product administration process. Administrators have access to a comprehensive set of tools that facilitate catalog management, providing control over all albums in stock, discount codes, user accounts, and feedback reports.

As a result, the administrator-focused application complements the user application, making Vinylla stand out for its complexity, efficiency, and versatility, becoming a comprehensive solution for music enthusiasts and vinyl merchants.

Cuprins

Capitolul 1 – Introducere

1.1 Motivație

Încă din copilărie am fost pasionată de muzică, de poveștile artiștilor și motivațiile lor de a crea artă, iar această pasiune mă însoțește până în prezent, fapt care m-a învățat de-a lungul timpului că fiecare artist are o poveste unică și că fiecare piesă poate fi o călătorie în lumea lor interioară. De-a lungul anilor, am explorat diverse genuri muzicale, fapt care m-a deschis către noi experiențe și perspective muzicale. Muzica reprezintă un mijloc de conectare cu diferite culturi și tradiții și depășește barierele culturale și lingvistice, devenind o limbă universală.

Într-o lume aflată într-o continuă evoluție digitală, adaptarea și inovația sunt esențiale, iar unul din domeniile care a reușit să supraviețuiască în ciuda avântului tehnologic din ultimii ani îl reprezintă piața vinilurilor, motiv pentru care mi-am dorit să găsesc un punct comun între aceasta și pasiunea mea pentru muzică, iar „Vinylla” vine ca o consecință a acestei dorințe, o modalitate de a-mi împărtăși pasiunea cu toata lumea.

De aceea, implementarea unui magazin online de viniluri poate deveni un răspuns firesc la nevoile iubitorilor de muzică de pretutindeni, reprezentând o modalitate de a aduce această pasiune în viața cotidiană. Fiecare album, fiecare piesă are o poveste unică, ceea ce face ca un magazin online de viniluri să devină un spațiu virtual în care aceste povești să fie explorate și trăite.

Prin intermediul acestei aplicații destinate achiziționării de viniluri, oferim iubitorilor de muzică posibilitatea de a-și completa colecțiile personale și de a descoperi noi talente și genuri muzicale. De la albumele memorabile ale marilor artiști, până la discuri rare și colecționabile, „Vinylla” își propune să ofere o selecție de calitate, pe placul tuturor.

1.2 Structura lucrării

Prin prezenta lucrare scrisă vor fi analizate implementarea și funcționalitatea aplicației atât din punct de vedere tehnic, cât și din punct de vedere conceptual. Vor fi atinse punctele cheie care contribuie la conturarea unei imagini clare asupra a ceea ce reprezintă și ce își propune să dezvolte „Vinylla”.

În cadrul celui de-al doilea capitol, va fi prezentată o analiză clară și în detaliu asupra tehnologilor folosite atât pentru partea de client, cât și pentru partea de server, la care se va prezenta o argumentare clară asupra motivelor folosirii acestora și avantajelor pe care le aduc.

Capitolul cu numărul 3 va fi dedicat componentei de UI/UX a proiectului și va conține detalii despre toate elementele de design folosite, de la paleta de culori și fonturi, până la componentele reutilizabile care au contribuit la performanța procesului de dezvoltare.

În cuprinsul capitolului 4 va fi descrisă pe larg funcționalitatea aplicației, însoțită de capturi de ecran menite să ajute la înțelegerea clară a implementării, vor fi atașate imagini cu paginile aplicației pentru realizarea unei analize relevante a produsului finit.

În cadrul capitolului 5 va fi explicat în detaliu cum a fost gândită întreaga bază de date și structura acesteia, care au fost colecțiile utilizate, relațiile dintre acestea, tipurile de date folosite și validările acestora. De asemenea, vor fi detaliare și deciziile luate pentru asigurarea integrității datelor și modalitatea de stocare a parolelor utilizatorilor.

Ultimul capitol este destinat analizei concluziilor, a experienței de dezvoltare și a impactului pe care aplicația îl poate avea asupra utilizatorilor săi.

Capitolul 2 - Tehnologiile folosite

Pentru dezvoltarea proiectului „Vinylla”, a fost aleasă o soluție frontend bazată pe tehnologiile ReactJs și SCSS, în combinație cu Redux Toolkit pentru integrarea cu backend-ul și gestionarea stării și comportamentului componentelo. Această alegere a permis dezvoltarea unei interfețe moderne care face ca experiența utilizatorilor să fie una plăcută.

În ceea ce privește partea de backend, am optat pentru folosirea tehnologiilor ca NodeJs, Mongoose și MongoDB care au permis dezvoltarea unei aplicații server-side care să răspundă cerințelor clienților, utilizând TypeScript ca limbaj de programare.

În cadrul acestui capitol, voi face o analiză în detaliu asupra motivelor pentru care am făcut aceste alegeri, subliniind avatajele pe care le aduc și modul în care au contribuit la succesul proiectului..

2.1 TypeScript

TypeScript este un limbaj de programare de nivel înalt, open-source care a fost dezvoltat de Microsoft pe parcursul a 2 ani, a cărui primă versiune a fost lansată în octombrie 2012 și care adaugă tipizare statică cu adnotări opționale în JavaScript, proiectat pentru dezvoltarea de aplicații mari. TypeScript vine la pachet cu toate feature-urile oferite de JavaScript, însă vine cu ceva în plus, și anume conceptul de „type safety”, deoarece impune o specificare clară a tipurilor variabilelor și parametrilor folosiți, fapt care eficientizeaza procesul de debugging și previne alte potențiale erori de conversie între tipuri de valori. Toate programele JavaScript sunt sintactic valide în TypeScript, dar pot avea erori de compilare din cauza aceasta. Pe lângă conceptul de „type safety”, cuprinde și alte concepte și feature-uri, cum ar fi interfețe, clase abstracte, suprascrieri de funcții etc.

2.2 De ce TypeScript și nu JavaScript?

Pentru realizarea aplicației „Vinylla”, am optat pentru folosirea limbajului TypeScript deoarece se poate caracteriza prin siguranță, scalabilitate și robustitate și este destinat dezvoltării

de aplicații de înaltă calitate. Însă de ce am optat pentru TypeScript și nu JavaScript? Pentru a răspunde la această întrebare, voi realiza o paralelă între cele două.

În primul rând, JavaScript este un limbaj de programare cu tipizare slabă și în același timp dinamică, motiv pentru care variabilele pot fi declarate fără a specifica tipul acestora, putând fi reasignate ulterior cu orice alt tip de date, în timp ce TypeScript adaugă un sistem de tipuri static, astfel că variabilele pot fi declarate doar cu tipuri specifice și sunt verificate în timpul compilării.

În al doilea rând, JavaScript permite flexibilitate în scrierea de cod, însă poate duce la erori de compilare care în unele cazuri pot fi greu de detectat în timpul procesului de development, în timp ce TypeScript impune o strictețe mai mare în ceea ce înseamnă scrierea de cod prin verificarea tipurilor obiectelor, parametrilor și variabilelor la compilare, fapt care rezultă la o detectare mai rapidă a erorilor de implementare.

În al treilea rând, TypeScript cuprinde câteva concepte în plus față de ceea ce oferă deja JavaScript, cum ar fi cel de interfață, modul, tipuri avansate de date, fapt care ajută la dezvoltarea unor aplicații mai robuste, cu un cod ușor de menținut.

În al patrulea rând, JavaScript este frecvent utilizat atât în proiecte de dimensiuni mici și medii, cât în dezvoltarea de aplicații web și funcționalități de frontend, în timp ce TypeScript este preferat pentru proiecte mari și complexe, datorită beneficiilor pe care îl aduce sistemul de menționare a tipului de date asupra dezvoltării și menținerii de cod.

În al cincilea rând, TypeScript beneficiază de interoperabilitate cu JavaScript și poate utiliza biblioteci și framework-uri comune la care se adaugă consistența oferită de feature-urile aduse în plus.

Coroborând cele menționate în paragrafele anterioare, am ales să implementez aplicația „Vinylla” prin intermediul limbajului TypeScript deoarece oferă dezvoltatorilor un set extins de caracteristici și beneficii, care îl fac potrivit pentru proiecte mai mari și complexe. În ciuda celor argumentelor anterioare, JavaScript rămâne un limbaj versatil, popular și ușor de folosit, mai ales de către începători, însă alegerea dintre cele două depinde atât de nevoile și cerințele specifice ale proiectului, cât și de preferințele dezvoltatorilor.

2.3 ReactJS

Pentru realizarea interfeței aplicației web „Vinylla” am optat pentru folosirea framework-ului ReactJS, o bibliotecă open-source de JavaScript, dezvoltată de Facebook și de o comunitate de dezvoltatori și companii individuale deoarece este renumită pentru diverse aspecte importante

pe care le voi analiza în următoarele paragrafe.

În primul rând, acest framekwork promovează conceptul de componente reutilizabile, ceea ce înseamnă că pot fi definite o dată și utilizate oriunde în întreaga aplicație, permițând o gestionare mai ușoară a codului și o mai mare eficiență în ceea ce privește timpul și resursele. Dat fiind acest argument, am decis să organizez componentele folosite conform unui concept numit „Atomic design" despre care voi discuta mai amănunțit în capitolul 3.

În al doilea rând, ReactJS urmează un model de flux de date unidirecțional care face ca datele să se deplaseze într-o singură direcție, adică de la componente părinte la componenete copii, fapt care facilitează gestionarea și actualizarea stării aplicației într-un clar și concis.

În al treilea rând, ReactJS se folosește de conceptul de „Virtual DOM", o reprezentare în memorie a arborelui DOM. Atunci când starea componentelor suferă modificări de orice fel, ReactJS compară arborele virtual cu versiunea anterioară a arborelui, iar odată ce diferențele sunt identificate, actualizează doar acele elemente care necesită modificări, în loc de a reconstrui întregul arbore de la capăt, fapt care reduce timpul de procesare și minimizează resursele și efortul necesar reflectării schimbărilor în cadrul interfeței.

Luând în considerare argumentele dezoltate în cadrul paragrafelor anterioare, ReactJS reprezintă o alegere optimă în ceea ce privește realizarea unei interfețe performante.

2.4 SCSS

Pentru stilizarea interfeței aplicației și a componentelor acesteia, am ales să folosesc SCSS, cunoscut și ca „Sassy CSS", care este un preprocesor de CSS ce aduce cu sine câteva funcționalități în plus care permit facilitarea dezvoltării și menținerii stilurilor CSS, cum ar fi existența variabilelor, conceptului de moștenire, funcții, operații matematice, etc., lucru care duce la o organizare mai bună a codului. Modul în care am folosit particularitățile sale va fi descris mai pe larg în cadrul capitolului 3, unde voi exemplifica ce concepte au fost folosite și unde.

2.5 NodeJS

În cadrul procesului de dezvoltare al acestui proiect, am ales să utilizez NodeJS ca principală tehnologie pentru implementarea backend-ului. Motivele care stau la baza acestei alegeri sunt multiple și vizează atât performanța, cât și scalabilitatea, necesare dezvoltării unei platforme solide și flexibile care să gestioneze cerințele specifice aplicației.

Principalul motiv pentru care am ales această tehnologie este natura sa asincronă și

nonblocantă, ceea ce înseamnă că serverul NodeJS poate manipula și suporta mai multe request-uri simultan, fără a aștepta finalizarea uneia pentru a trece la următoarea, ceea ce reprezintă o abordare ideală în cazul unei aplicații de ecommerce care poate primi multiple request-uri în același timp din diferite părți, asigurând un timp de răspuns rapid, crescând performanța.

În plus, NodeJS dispune de o gamă largă de instrumente și biblioteci care facilitează implementarea de funcționalități complexe la nivel de server, cum ar fi „Express.js", cu ajutorul căruia am putut construi rutele serverului responsabil de funcționarea optimă a aplicației, sau „Mongoose", o bibliotecă despre a cărei folosire voi vorbi mai pe larg în următorul subcapitol.

De asemenea, NodeJS a devenit o alegere tot mai populară de-a lungul ultimilor ani în rândul dezvoltatorilor de aplicații web, ceea ce indică faptul că există o comunitate activă, o documentație bogată și o multitudine de resurse care vin în sprijinul oricui.

Având în vedere cele menționate, NodeJS reprezintă alegerea perfectă pentru implementarea unui server datorită eficienței sale în realizarea și gestionarea sarcinilor asincrone, dar și a scalabilității sale.

2.6 Mongoose

Mongoose este o bibliotecă de JavaScript concepută și dezvoltată pentru simplificarea interacțiunii cu baza de date MongoDB care oferă un set de instrumente necesare gestionării de scheme, validări de date și efectuare de operații de tip CRUD.

Un motiv important pentru care am ales această librărie este abordarea sa orientată pe obiecte ce ne permite definirea de scheme care reflectă exact structura datelor pe care dorim să le stocăm în baza de date și prelucrarea lor într-un mod eficient, astfel că putem să definim relații între entități, să adăugam funcții pentru prelucrarea acestor entități, cum ar fi indexarea și denormalizarea datelor.

De asemenea, Mongoose oferă o gamă variată de funcții utile pentru validarea datelor cu care putem defini reguli de validare pentru câmpurile obiectelor din colecții, ceea ce asigură integritatea și consistența datelor.

Având acestea în vedere, Mongoose reprezintă o alegere optimă pentru modelarea unei baze de date consistente care răspunde cerințelor pieții și clienților.

2.7 MongoDB

Pentru stocarea tuturor datelor necesare funcționării aplicației, am ales MongoDB, o bază de date NoSQL non-relațională și orientată pe obiecte.

Motivul principal pentru care am ales această tehnologie este flexibilitatea sa care poate fi utilă în cazul unei schimbări de structură a vreunui document sau colecții pe parcursul procesului de dezvoltare. Pe lângă asta, mai oferă suport și pentru replicare, astfel că ne putem asigura că datele sunt disponibile în mod redundant, ceea ce adaugă o rezistență mai mare a infrastructurii și oferă posibilitatea de a recupera rapid datele în cazul unei defecțiuni care poate afecta un nod.

Important de menționat este și tipul de stocare al datelor, și anume în format BSON (Binary JSON), care asigură o stocare compactă a datelor, ceea ce înseamnă că reduce spațiul necesar păstrării lor și mărește viteza accesării și prelucrării lor.

De asemenea, aceasta a devenit de-a lungul timpului o tehnologie tot mai răspândită în rândul dezvoltatorilor de pretutindeni, fapt care a rezultat într-o creștere a comunității și a disponibilității resurselor și documentației.

2.8 Redux Toolkit

Pentru ceea ce presupune management-ul stării componentelor și integrarea backend-ului cu frontend-ul, am decis să folosesc Redux Toolkit, ce reprezintă un pachet conceput pentru o implementare standard de Redux.

Motivul acestei decizii îl reprezintă faptul că prin intermediul său, sunt configurate în mod implicit anumite aspecte ale Redux-ului pentru a nu mai fi necesară configurarea manuală. Pe deasupra, include un store de Redux preconfigurat cu suport încorporat pentru middleware precum „Redux Thunk” sau „Redux Saga”.

Capitolul 3 – UI/UX

3.1 Elemente de design

Aplicația „Vinylla” își propune să ofere utilizatorilor, adică atât cumpărătorilor, cât și administratorilor, o experiență plăcută din punct de vedere vizual, acesta fiind obiectivul principal în realizarea aplicației deoarece interfața reprezintă un aspect central când vine vorba de experiența oricărui utilizator. „Vinylla” a fost concepută pentru a oferi o platformă intuitivă , atrăgătoare, cu un design vizual menit să atragă cumpărători, să evidențieze calitățile produselor, să ofere informații clare și o experiență plăcută de cumpărare. Pentru administratori, aplicația pune la dispoziție o interfață de administrare intuitivă și eficientă, permițând o navigare ușoară și oferirea de informații relevante într-un mod clar și concis.

3.1.1 Paleta de culori

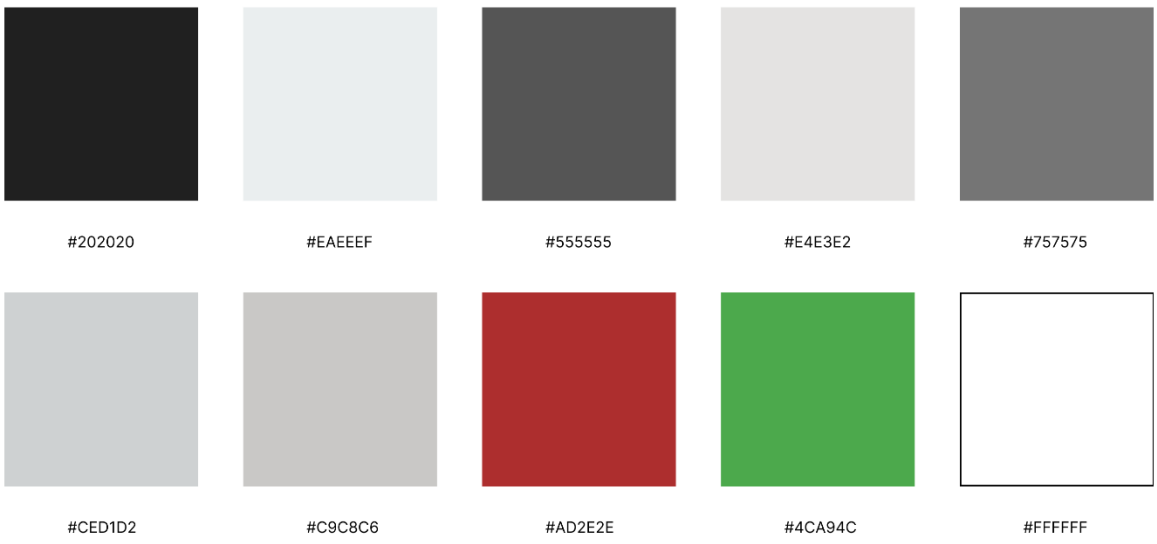

Figura 3.1 – Paleta de culori a aplicației

Unul din pricipalele scopuri ale aplicației, și anume acela de a atrage cumpărători, este realizat cu ajutorul unei palete de culori caracterizată prin coerență, consistență și vizibilitate, fiind alcătuită dintr-o gamă diversificată de tonuri de gri pentru texte și componente, dar și nuanțe de verde și roșu folosite pentru validări de input-uri. Utilizarea nuanțelor de gri poate contribui la

crearea unui design minimalist și simplu în aplicație, fiind folosit pentru a evidenția liniile și formele clare, oferind un aspect curat și organizat.

Stocarea valorilor hexa ale culorilor din cadrul paletei de mai sus a fost realizată cu ajutorul variabilelor, o particularitate a SCSS-ului, într-un fișier separat, special creat pentru această funcționalitate, fapt care oferă aplicației un plus de consistență și face codul mai ușor de întreținut.

```
$color-white: #ffffff;
$color-1: #eaeeef;
$color-2: #e4e3e2;
$color-3: #757575;
$color-4: #555555;
$color-5: #202020;
$color-6: #ced1d2;
$color-7: #c9c8c6;
$color-black: #000000;
$color-red: #ad2e2e;
$color-green: #4ca94c;
```

Figura 3.2 – Stocarea codurilor culorilor în variabile

3.1.2 Fonturi

Un alt factor care a contribuit la îmbunătățirea aspectului vizual al aplicației este alegerea fonturilor, preluate de pe Google Fonts, o bibliotecă online de fonturi gratuite, drept pentru care au fost alese 2, și anume: Space Mono și Roboto. Acestea au fost stocate, la rândul lor, cu ajutorul unor variabile într-un fișier separat pentru un plus de performanță.

```
@import url('https://fonts.googleapis.com/css2?family=Space+Mono&display=swap');
@import url('https://fonts.googleapis.com/css2?family=Roboto&display=swap');

$primary-font: 'Space Mono', monospace;
$secondary-font: 'Roboto', sans-serif;
```

Figura 3.3 – Stocarea fonturilor în variabile

3.1.3 Realizarea logo-ului

Pentru realizarea logo-ului aplicației a fost folosită platforma Looka, specializată în crearea și customizarea de logo-uri, ce folosește o inteligență artificială pentru generarea unor

sugestii de logo-uri, în funcție de numele brand-ului tău, câteva cuvinte cheie,
culori preferate și alte preferințe specifice. Am ales acest tool atât datorită acestui feature, cât și datorită posibilității de personalizare și descărcare a sugestiilor generate.

Figura 3.4 – Logo-ul aplicației „Vinylla”

3.1.4 Iconițe

Pentru folosirea de iconițe am decis să preiau de pe platforma Icon Finder iconițele necesare pentru reprezentarea anumitor butoane și opțiuni din meniu, iar pentru folosirea lor, am ales o stocare internă într-un folder separat, în format .png, fapt ce oferă control asupra gestionării și utilizării acestora în aplicație.

3.1.5 Imagini de fundal

Pentru implementarea anumitor pagini din cadrul aplicației, a fost nevoie de folosirea unor imagini de fundal, drept pentru care am folosit platforma Pexels care oferă o vastă colecție de imagini și videoclipuri gratuite, utilizabile în scopuri personale și comerciale, licențiate sub licența Creative Commons Zero (CC0), ceea ce înseamnă că tot content-ul poate fi folosit în mod liber și fără a fi nevoie de acordare de credite.

3.1.6 Animaţii

Pentru a îmbunătăți exprienta utilizatorilor și a face interacțiunea dintre user și aplicație mai interactivă, am integrat și anumite animații pentru care am folosit conceptul de @keyframes, menit să controleze pașii intermediari într-o secvență de animație pentru definirea stilurilor de-a lungul secvenței.[5] Ele au fost folosite pentru stilizarea componentelor de tip modal, cum ar fi cart-ul, wishlist-ul, formularele pentru adăugare, editare sau ștergere de adrese, opțiunile de filtre sau sortare, meniul din stânga, etc.

```
@keyframes fadeIn {
  0% {
```

```
    transform: translateY(-1rem);
    opacity: 0;
  }
  100% {
    transform: translateY(0);
    opacity: 1;
  }
}
```

Figura 3.5 – Exemplu de animație folosită

3.2 Atomic Design

Atomic Design este o metodă de proiectare și dezvoltare a interfeței prin organizarea componentelor într-un mod modular și scalabil. Aceasta a fost introdusă de designer-ul Brad Frost printr-un articol publicat în anul 2013 cu numele „Atomic Design by Brad Frost” în care a prezentat conceptele fundamentale ale acestui mod de lucru.

În cadrul lucrării sale, autorul susține că o interfață poate fi descompusă în componente din ce în ce mai mici, așa cum materia ce ne înconjoară poate fi descompusă în particule, astfel că el a realizat o ierarhie în care se pot organiza componentele:

- **Atomi**

Așa cum în știință, atomul reprezintă cea mai mică unitate de materie care alcătuiește un element chimic, similar, în design, atomii sunt reprezentați de componentele de bază, cum ar fi butoanele, câmpuri de text, etichete etc.

În cadrul proiectului „Vinylla”, am ales să adopt principiile Atomic Design pentru a-mi organiza componentele într-un mod care îmi permite să construiesc interfața într-un mod eficient, am ales să implementez la nivel de atomi componente precum butoanele, avatarul de user, tooltip-urile, etichetele, iconitele și componenta de notificare a cantității, atât din cadrul coșului de cumpărături, cât și din cadrul listei de produse favorite aflate în componența meniului de navigare din partea superioară a ecranului.

BUTTON

Pentru a exemplifica modalitatea în care am adoptat această ierarhie în ceea ce privește organizarea codului, voi realiza o analiză succintă a componentei de buton pe care am ales să o implementez integral. Această decizie a fost luată întrucât am dorit să obțin o soluție care să

îndeplinească în totalitate cerințele și nevoile interfeței aplicației, luând în același timp în considerare principiile Atomic Design.

```
const Button = ({
  ref, className, iconClassName, onClick, onSubmit, name, id,
  hasIconLeft = false,  hasIconRight = false, iconLeft, iconRight,
  type, hasIconOnly = false, icon, onMouseEnter, onMouseLeave, onClickCapture,
  disabled, style,
}: Props) => {
  return (
    <button
      style={style}
      ref={ref}
      disabled={disabled}
      className={className}
      onClick={onClick}
      onSubmit={onSubmit}
      id={id}
      type={type}
      onMouseEnter={onMouseEnter}
      onMouseLeave={onMouseLeave}
      onClickCapture={onClickCapture}
    >
      {hasIconOnly && icon ? (
        <img src={icon} className={iconClassName} />
      ) : (
        <>
          {hasIconLeft && iconLeft && (
            <img src={iconLeft} className={iconClassName} />
          )}
          {name}
          {hasIconRight && iconRight && (
            <img src={iconRight} className={iconClassName} />
          )}
        </>
      )}
    </button>
  );
};
```

Figura 3.6 – Componentă de buton personalizată

Componenta ilustrată mai sus a fost implementată în așa fel încât să primească o serie proprietăți configurabile și care permit personalizarea aspectului și comportamentului atomului. Ea returneaza un element de tip <button> în care sunt incluse iconițele și textul corespunzătoare

configurației primite. În funcție de proprietățile primite, se afișează iconițele în stânga și/sau în dreapta butonului, înaintea sau după textul său. Dacă butonul are doar o iconiță și nu conține text, atunci aceasta este afișată în mod exclusiv.

Important de precizat este și faptul că tipul butonului este întotdeauna alocat cu ajutorul unei structuri de tip enum care cuprinde toate tipurile de buton cunoscut, mai exact: „button”, „submit” și „reset”.

INPUT

Utilizând aceeași abordare a fost folosită și pentru implementarea unei componente de tip <input> customizate astfel încât să răspundă cerințelor aplicației.

```
const TextInput = ({
  label, labelClassName, inputClassName, placeholder, type, onChange, onBlur,
  htmlFor, id, name, value,
}: Props) => {
  return (
    <div className='text-input-container'>
      <input
        id={id}
        name={name}
        value={value}
        type={type}
        placeholder={placeholder}
        className={inputClassName}
        onChange={onChange}
        onBlur={onBlur}
      />
      {label && (
        <label className={labelClassName} htmlFor={htmlFor}>
          {label}
        </label>
      )}
    </div>
  );
};
```

Figura 3.7 – Componentă de input personalizată

Ea permite personalizarea mai avansată a câmpului de introducere de text, fiind ușor de utilizat și integrat în diferite părți ale aplicației, cum ar fi formularele de înregistrare, autentificare,

adăugare sau editare de adrese sau scriere de recenzii.

AVATAR

Componenta de avatar a fost dezvoltată special pentru a oferi o reprezentare vizuală a inițialelor unui utilizator în aplicație. Aceasta primește două proprietăți: onClick, care specifică acțiunea de deschidere a profilului utilizatorului atunci când este făcut clic pe avatar, și userName, care reprezintă numele utilizatorului asociat cu avatarul.

În cadrul figurii de mai jos, se poate observa că se efectuează o verificare pentru a asigura existența unui userName valid. Dacă userName-ul nu este undefined, atunci se separă numele în cuvinte separate și se realizează o concatenare dintre primele litere ale fiecărui cuvânt.

Prin includerea componentei Avatar în diferite părți ale aplicației, cum ar fi bara de navigare din partea superioară a ecranului sau profilul utilizatorului, se poate crea o experiență personalizată și vizual atrăgătoare pentru utilizatori.

```
const Avatar = ({ onClick, userName }: Props) => {
  let firstLetters;
  if (userName) {
    const names = userName.split(' ');
    firstLetters = names[0][0] + names[1][0];
  }
  return (
    <div onClick={onClick} className='avatar--container'>
      {firstLetters && firstLetters.toUpperCase()}
    </div>
  );
};
```

Figura 3.8 – Componentă de avatar personalizată

TOOLTIP

Componenta de tooltip a fost implementată pentru a afișa un mic mesaj informativ despre elementul pe care-l însoțește pentru momentul în care utilizatorul își plasează cursorul peste acesta. El primește două proprietăți: text, care reprezintă mesajul afișat și children, care reprezintă elementul peste care se va afișa tooltip-ul.

Pentru dezvoltarea sa, a fost utilizat hook-ul useState pentru a gestiona starea afișării tooltip-ului. Atunci când utilizatorul plasează cursorul peste element, se activează funcția handleMouseEnter, care actualizează starea showTooltip la true, afișând astfel tooltip-ul. În momentul în care cursorul părăsește elementul, se activează funcția handleMouseLeave, care

setează starea showTooltip la false, ascunzând tooltip-ul.

```
const Tooltip = ({ text, children }: Props) => {
  const [showTooltip, setShowTooltip] = useState(false);

  const handleMouseEnter = () => {
    setShowTooltip(true);
  };

  const handleMouseLeave = () => {
    setShowTooltip(false);
  };

  return (
    <div
      className='tooltip--container '
      onMouseEnter={handleMouseEnter}
      onMouseLeave={handleMouseLeave}
    >
      {children}
      {showTooltip && <div className='tooltip--text animate'>{text}</div>}
    </div>
  );
};
```

Figura 3.9 – Componentă de tooltip personalizată

Acestea ar fi câteva dintre componentele care au fost încadrate în categoria atomilor și au fost personalizate astfel încât să raspundă cerințelor aplicației.

- **Molecule**

Conform metodologiei „Atomic Design”, moleculele reprezintă îmbinări de componente atomice care împreună creează funcționalități mai complexe. Acestea pot fi considerate ca fiind grupuri de atomi care au o legătură semantică și funcțională.

Ceea ce deosebește moleculele de atomi este faptul că sunt mai cuprinzătoare și aduc o valoare mai consistentă în ceea ce privește funcționalitatea interfeței, prezenând un nivel superior de complexitate.

În cadrul proiectului „Vinylla”, am folosit conceptul de molecule pentru implementarea componentei de search bar, care conține atomul de buton și cu ajutorul căreia am adăugat funcționalitatea de căutare a unui produs dorit.

```
return (
    <div className={`searchBox ${searchBar ? 'active' : ''}`}>
```

```
      <input
        className='searchInput'
        type='text'
        name=''
        placeholder='what are you looking for?'
        onChange={handleSearch}
        onKeyDown={searchText ? handleKeyDown : undefined}
      />
      <Button
        className='searchButton'
        iconClassName='menu-icon'
        hasIconLeft={true}
        iconLeft={searchBar ? searchBarIcons[0].icon : searchBarIcons[1].icon}
        onClick={handleSearchBar}
      />
      {!searchBar && (
        <div className='search-bar-text'>
          <FormattedMessage id='components.navbar.search' />
        </div>
      )}
    </div>
  );
```

Figura 3.10 – Componentă de search bar personalizată

Folosind această metodologie, s-a reușit implementarea mai multor molecule, cum ar fi dropdown-uri, componente folosite pentru reprezentarea de informații despre adrese, recenzii, comenzi, notificări etc.

- **Organisme**

În metodologia „Atomic Design”, organismele reprezintă un tip mai complex de componente, care sunt implementate prin combinarea de atomi și molecule, ele putând fi responsabile de reprezentarea de anumite secțiuni de pagină și fiind mult mai interactive.

Acest concept este reflectat în dezvoltarea aplicației prin intermediul unor componente precum meniul de navigare din partea superioară a ecranului și meniul de navigare din partea stângă a ecranului care pot fi considerate organisme, deoarece conțin multiple molecule și atomi, precum butoane, elemente de tip dropdown sau pictograme.

Alte organisme implementate în aplicație includ secțiunea care cuprinde lista de recenzii ale unui produs, coșul de cumpărături și lista de produse favorite. Aceste organisme sunt mai complexe, deoarece implică interacțiunea dinamică cu date, precum adăugarea sau ștergerea de

produse, actualizarea cantității sau gestionarea recenziilor.

De asemenea, au fost implementate organisme sub formă de modale pentru acțiuni specifice, cum ar fi adăugarea, editarea sau ștergerea unei adrese, precum și raportarea unei recenzii. Aceste modale conțin molecule și atomi precum câmpuri de formular, butoane și mesaje de eroare, oferind o interfață coerentă și ușor de utilizat.

```
return (
    <div
      ref={menuRef}
      className={`menu--container ${isMenuDisplayed ? 'expanded' : ''}`}
    >
      <Button
        className='menu--close'
        iconClassName='menu--close-icon'
        onClick={() => setIsMenuDisplayed(!isMenuDisplayed)}
        hasIconLeft={true}
        iconLeft={require('../../../assets/icons/CloseIcon.png')}
        name={<FormattedMessage id='components.menu.close' />}
      />
      <RecursiveDropdown fontSize={24} children={sideMenuItems} />
    </div>
  );
```

Figura 3.11 – Componentă de meniu personalizată

Componenta prezentată mai sus reprezintă implementarea meniului din partea stângă a ecranului, care oferă user-ului opțiuni de navigare și acces rapid la diverse funcționalități, fiind configurată astfel încât să fie extensibilă, ceea ce înseamnă că poate fi afișată sau ascunsă în funcție de acțiunile user-ului. Pentru implementarea sa, a fost folosit atât atomul de „Button”, pentru închiderea sau deschiderea sa, cât și molecula de „Recursive Dropown”, pentru a afișa lista de opțiuni.

Capitolul 4 – Fluxul Aplicației

Deoarece un magazin online de viniluri implică o multitudine de componente, procese și funcționalități complexe, este esențială existența unui sistem de gestiune care să vină atât în sprijinul utilizatorilor, cât și în spijinul administratorilor. De aceea, întregul proiect poate fi divizat în 2 aplicații diferite destinate celor 2 tipuri de utilizatori, reprezentând cele două elemente fundamentale care contribuie la funcționarea și organizarea optimă a unui astfel de magazin.

4.1 Aplicația destinată cumpărătorilor

Aplicația destinată cumpărătorilor a fost dezvoltată în așa fel încât să ofere utilizatorilor ei o experiență plăcută în timpul procesului de navigare și cumpărare. Prin intermediul ei, aceștia își pot explora catalogul de produse, pot căuta sau vizualiza detalii despre un album sau își pot crea un cont pe platforma pentru a beneficia de proprietăți în plus.

4.1.1 Sign up și Log in

„Vinylla” oferă utilizatorilor săi posibilitatea de a naviga prin aplicație fără a fi autentificat, însă nu oferă aceleași beneficii și functionalități. Acest lucru se poate vedea încă din pagina de produs, unde un utilizator nu are posibilitatea de a adăuga un produs în coșul de cumpărături sau la lista de produse favorite și nu poate să adauge sau să raporteze recenzii. În cazul în care încearcă să facă acest lucru, pe ecran va apărea un modal cu un mesaj care solicită logarea pentru a continua. De aici utilizatorul poate alege să navigheze către pagina în care poate să-și creeze un cont sau către pagina care îi permite autentificarea.

Figura 4.1 – Modalul pentru autentificare

Figura 4.2 – Pagina de creare de cont

Dacă utilizatorul decide să-și creeze un cont, el va fi redirecționat către pagina reprezentată în figura de mai sus unde el este nevoit să completeze toate câmpurile pentru a continua. Pentru implementarea formularului de înscriere a fost folosită librăria open-source „Formik” pentru validarea câmpurilor și procesarea valorilor. Odată ce formularul este completat cu valorile corespunzătoare, utilizatorul va fi redirecționat către pagina de profil a contului său, unde va avea acces la funcționalități specifice unei pagini de profil.

Figura 4.3 – Pagina de autentificare

Dacă decide în schimb să se autentifice, el va fi redirecționat către pagina de log in, unde va trebui să completeze câmpurile corespunzătoare pentru email și parolă. Ambele ecrane, atât de creare de cont, cât și de autentificare, au atașate validări pentru fiecare câmp.

De asemenea, aplicația oferă utilizatorilor și posibilitatea de a se deconecta de la contul lor.

4.1.2. Pagina de produs

Pagina de produs reprezintă o componentă importantă deoarece oferă utilizatorilor informații relevante despre un album disponibil în baza de date a magazinului. Aceasta constituie un punct central în procesul de achiziție și totodată un factor decisiv care influențează decizia cumpărătorului.

În cadrul acesteia, utilizatorul poate vizualiza date esențiale despre album, cum ar fi numele său și al artistului, durata, data lansării, prețul, imaginile de prezentare, lista de piese cuprinse, premii obținute, recenziile primite, rating-ul total și alte date adiționale. Ei au posibilitatea de a selecta dimensiunea dorită a vinilului din cele trei opțiuni disponibile: diametru de 7, 10 și 12 inch. Această funcționalitate le permite să aleagă formatul potrivit înainte de a adăuga produsul în coșul de cumpărături sau în lista de produse favorite.

Figura 4.4 – Secțiune de recenzii

Sub secțiunea destinată informațiilor despre vinil se află secțiunea destinată recenziilor, acolo unde rating-ul final este calculat în funcție de notele primite de album. El este determinat prin calcularea mediei aritmetice a tuturor notelor. Prin aplicarea formulei de calcul adecvate, se obține rating-ul care reflectă evaluările și opinii exprimate de utilizatori.

După cum am menționat anterior, utilizatorii autentificați au opțiunea de a adăuga recenzii pentru albumele dorite și, de asemenea, pot raporta comentarii considerate nepotrivite. Această

funcționalitate permite utilizatorilor să-și exprime părerea despre un album și să ofere feedback util celorlalți potențiali cumpărători. În plus, opțiunea de raportare a comentariilor nepotrivite ajută la menținerea integritatea platformei și protejarea utilizatorilor de conținut tulburător sau ofensiv.

4.1.3. Pagina de profil

Pagina de profil reprezintă un element de o importanță crucială în orice aplicație, deoarece oferă utilizatorilor posibilitatea de a-și gestiona informațiile.

Prin intermediul acesteia, utilizatorii platformei „Vinylla” au acces la functionalități importante, cum ar fi vizualizarea istoricului de comenzi plasate, gestionarea listei de adrese salvate pe care le poate folosit în timpul procesului de checkout și consultarea inbox-ului. În plus, au posibilitatea de a edita informații specifice contului, cum ar fi numele, prenumele și numărul de telefon, dar și de a-și schimba parola pentru asigurarea securității contului, la toate acestea adăugându-se posibilitatea de a ieși din cont.

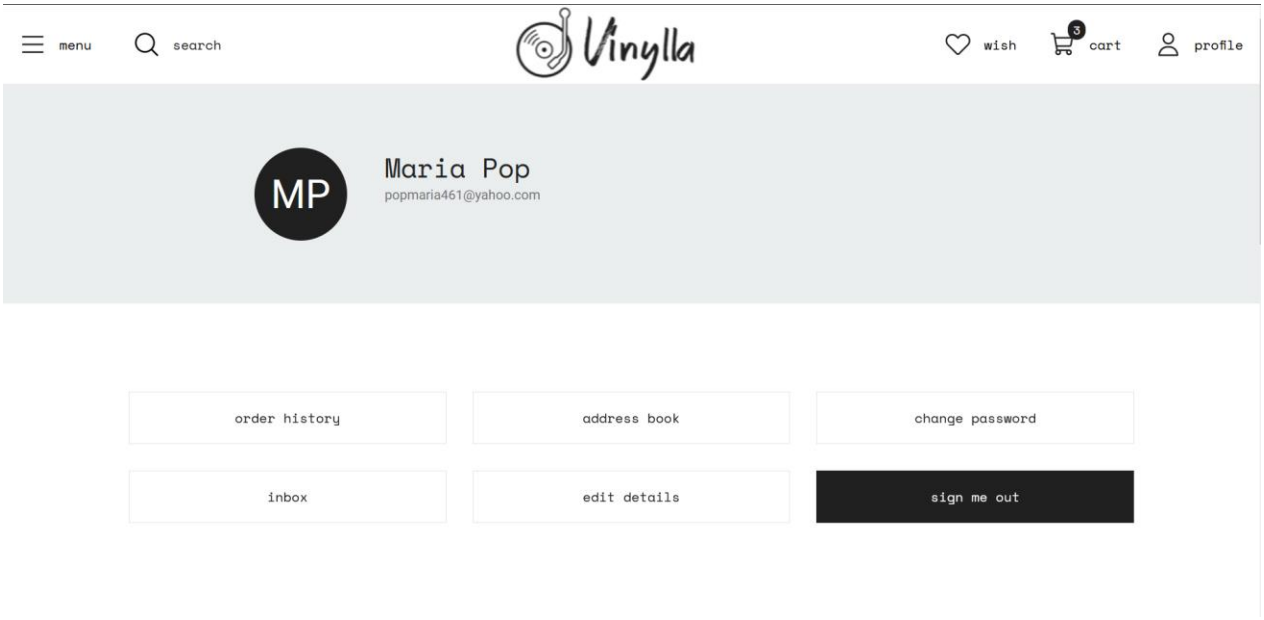

Figura 4.5 – Pagina de profil

4.1.4. Schimbarea parolei

Funcționalitatea de schimbare a parolei reprezintă o componentă esențială pentru orice platformă, de aceea pentru proiectul „Vinylla” a fost dedicată o pagină separată pentru a putea realiza această operație.

Procesul de schimbare a parolei implică introducerea parolei vechi, apoi a celei noi de două ori pentru confirmarea acesteia. După apăsarea butonului "change password", se va efectua o solicitare către un endpoint de tip "UPDATE" care va verifica inițial corectitudinea parolei vechi. Dacă parola veche este introdusă corect, cea nouă o va înlocui, fiind criptată folosind biblioteca bcrypt descrisă în cadrul capitolului 5.

4.1.5 Inbox

Pagina de profil oferă acces și către pagina de inbox care joacă un rol esențial în ceea ce privește gestionarea comunicării și interacțiunii cu clienții deoarece oferă un spațiu centralizat și eficient pentru schimbul de informații între vânzător și cumpărător.

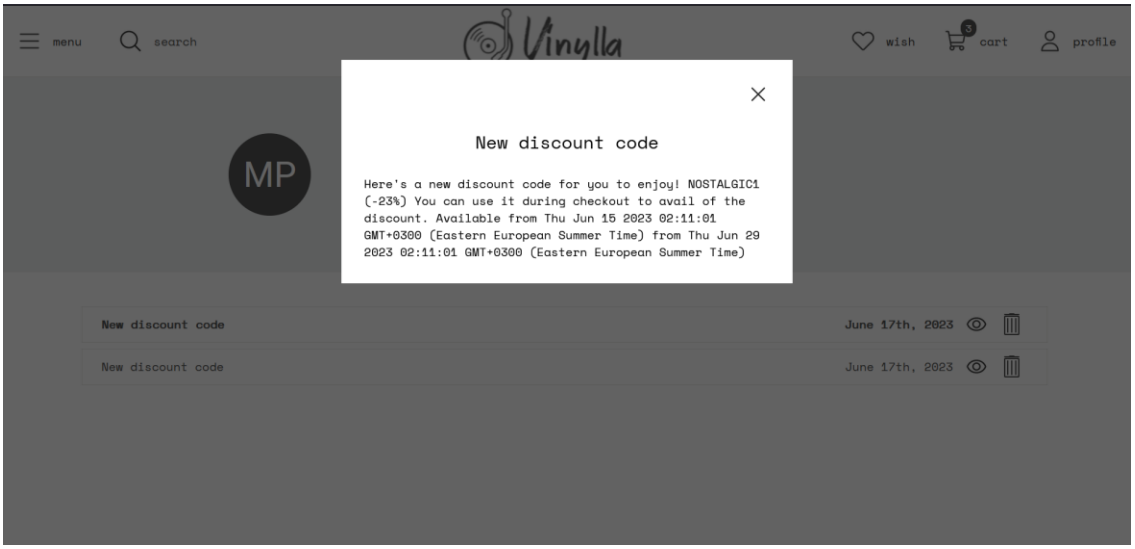

Figura 4.6 – Notificare de cod nou de reducere

Aici utilizatorii pot primi notificări legate de coduri de reducere nou apărute pe care le pot folosi în timpul procesului de cumpărare pentru a beneficia de reduceri. De asemenea, pot primi feedback în legătură cu rapoartele pe care le-au făcut pentru anumite recenzii, de exemplu, el poate primi o notificare care îl informează că motivul pentru care a raportat un comentariu este unul valid, motiv pentru care acesta a fost șters, iar autorul său a primit o avertizare de suspendare a

contului.

Ei au posibilitatea de a vizualiza conținutul unei notificări prin apăsarea iconiței din dreapta cu textul „see more” sau pot alege să-l șteargă prin apăsarea iconiței cu textul „delete”. De asemenea, mesajele sunt afișate sub formă de listă, cele nevizualizate având titlul scris cu litere îngroșate pentru realizarea diferențierii între cele vizualizate și cele nevizualizate.

4.1.6 Pagina de adrese

Această pagină a fost implementată pentru a le oferi clienților oportunitatea de a-și reține adresele pe care le pot folosi în timpul procesului de checkout pentru simplificarea plasării de comenzi.

Ei pot vedea întreaga listă a adreselor salvate și le pot gestiona eficient, fapt ce adaugă un nivel suplimentar de comoditate și eficiență în procesul de cumpărare, reducând efortul necesar pentru introducerea repetată a informațiilor de adresă la fiecare comandă.

Dezvoltarea acestei funcționalități este realizată cu ajutorul unui modal intuitiv, care facilitează atât adăugarea de adrese noi, cât și editarea sau ștergerea celor existente. Pentru acestea, a fost folosită biblioteca „Formik”, folosită pentru implementarea tuturor formularelor din cadrul aplicației. Atunci când se dorește realizarea unei operații de adăugare sau editare, va fi folosit un modal ce conține doar câmpuri obligatorii pentru: nume (de exemplu, „Acasă”, „Casa bunicilor”, etc), nume, prenumele și numărul de telefon al persoanei căreia îi aparține adresa, țara, regiunea, adresa propriu-zisă și codul poștal.

Figura 4.7 – Modal pentru adăugarea de adrese

4.1.7 Pagina de produse

Ca orice magazin online, „Vinylla” oferă utilizatorilor săi o pagină destinată răsfoirii catalogului de produse în cadrul căreia este pus la dispoziție un set de instrumente menite să rafineze căutările potențialilor cumpărători.

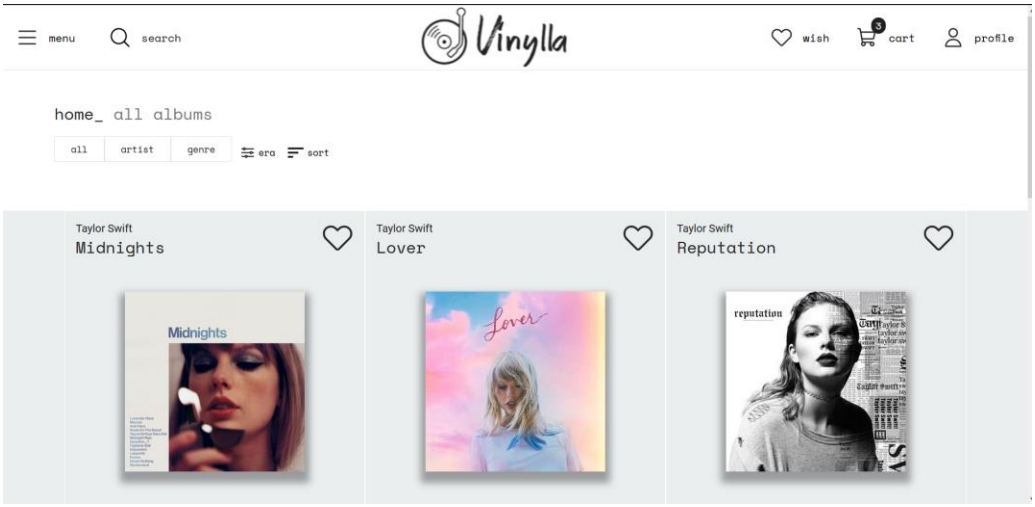

Figura 4.8 – Pagina de produse

De aceea, un utilizator are posibilitatea de a alege dacă dorește să vadă toate produsele sau să caute un album în funcție de numele artistului sau genului muzical. Pe lângă asta el poate alege decada din care să facă parte albumele pe care pagina le va afișa.

În plus față de cele menționate anterior, clientul are posibilitatea de a folosi și un tip de sortare din cele patru pe care aplicația le pune la dispoziție. Acestea pot fi în funcție de titlu sau de preț, în ordine crescătoare sau descrescătoare. Pentru realizarea acestei funcționalități, a fost nevoie de utilizarea unui singur endpoint de tip GET a cărui implementare a fost descrisă în cadrul capitolului 5.

4.1.8 Procesul de cumpărare

Procesul de cumpărare reprezintă funcționalitatea cheie a oricărui magazin online, de aceea, „Vinylla” își propune să furnizeze o experiență cât mai plăcută pentru utilizatorii săi, motiv pentru care au fost stabiliți câțiva pași pe care clienții trebuie să-i urmeze pentru a putea plasa comenzi.

În primul rând, aceștia trebuie să-și adauge produsele favorite în coșul de cumpărături de

unde pot vedea și seta cantitatea dorită. Tot aici ei pot vedea subtotalul care reprezintă produsul dintre cantitatea și prețul vinilului, dar și prețul final al tuturor produselor. De aici, ei pot accesa pagina coșului de cumpărături care oferă toate informațiile necesare pentru un potențial cumpărător. Următorul pas este pagina de checkout unde ei trebuie să completeze datele adresei de livrare sau pot pur și simplu să selecteze o adresă din cele salvate. Pe lângă asta, pot aplica un cod de reducere care dacă este valabil, va oferi o scădere a tarifului total egală cu valoarea reducerii codului.

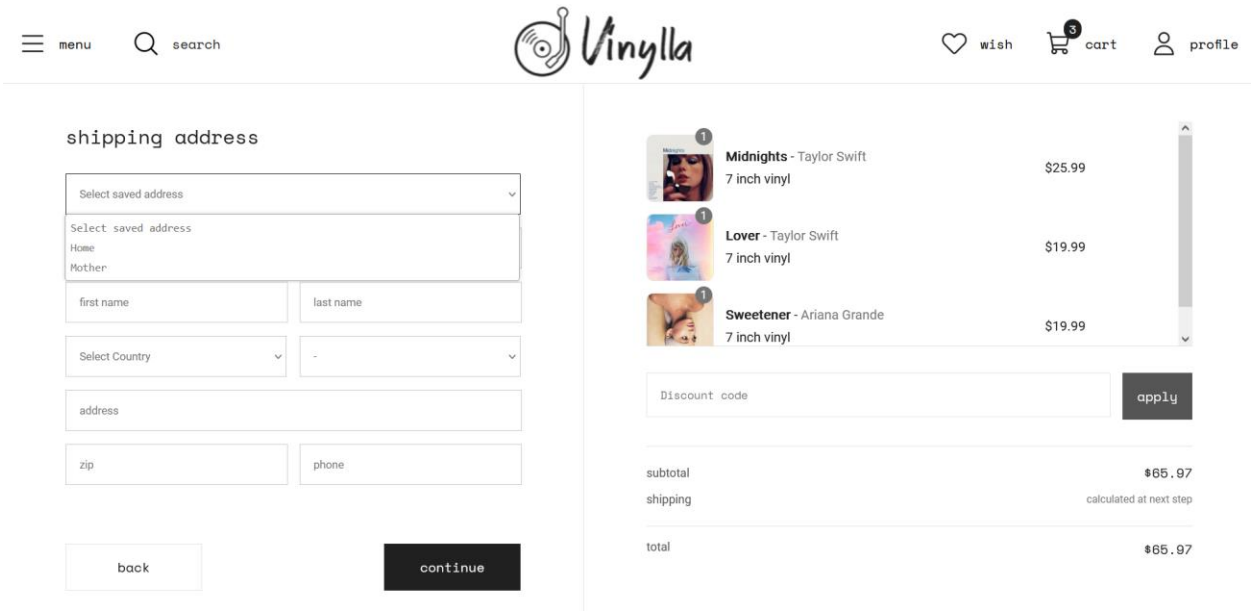

Figura 4.9 – Pagina de checkout

Următorul pas îl reprezintă atât alegerea tipului de livrare, care poate fi de 3 feluri: „SameDay”, „Cargus” și „FAN Courier”, cât și efectuarea plății propriu-zise. Aceasta a fost implementată cu ajutorul „Braintree”, o platformă de procesare a plăților online recunoscută pentru serviciile sale avansate și soluțiile securizate.

Unul din principalele beneficii pe care îl aduce utilizarea sa este faptul că oferă posbilitatea de a folosi multiple metode de plată. Comercianții pot accepta plăți facute cu carduri de credit sau debit ce pot aparține unor rețele precum Visa, Mastercard, American Express, PayPal, Venmo sau chiar și Apple Pay.

De asemenea, „Braintree” este cunoscut pentru securitatea sa avansată datorată tehnologiilor puternice de criptare pe care le folosește pentru a proteja date personale și financiare, fapt care îl face să fie o alegere de încredere când vine vorba de plăți online.

Un alt aspect esențial este faptul că oferă funcționalități suplimentare, cum ar fi facturarea

recurentă sau gestionarea abonamentelor, ceea ce permite automatizarea procesului de plată.

Important de reținut este că „Braintree” funcționează ca un fel de intermediar între o aplicație și cei care primesc sau procesează plățile, cum ar fi băncile, deoarece facilitează securitatea și procesarea plăților prin intermediul API-urilor pe care le pune la dispoziție.

În cadrul „Vinylla”, plata oricărei comenzi se poate face doar cu ajutorul cardului, iar pentru integrarea sa în proiect, au fost folosite două funcții:

```
const getClientToken = async () => {
    try {
      const { data } = await axios.get(
        'http://localhost:8000/api/braintree/token'
      );
      setClientToken(data.clientToken);
    } catch (err) {
      console.log(err);
    }
  };

  const handleBuy = async () => {
    try {
      const { nonce } = await instance.requestPaymentMethod();
      console.log(nonce);
      const response = await axios.post(
        'http://localhost:8000/api/braintree/payment',
        { nonce }
      );
      console.log(response.data);
    } catch (err) {
      console.log(err);
    }
  };
```

Figura 4.10 – Funcțiile pentru procesarea de plăți

Funcția „getClientToken” este responsabilă pentru obținerea token-ului de client necesar pentru inițializarea Braintree, trimițând o cerere de tip GET către ruta sa specifică. Token-ul de client este returnat și setat într-o variabilă de tip state.

Funcția „handleBuy” este funcția care este apelată după ce datele cardului au fost introduse, iar utilizatorul confirmă plata. În interiorul ei, se obține un identificator unic generat de Braintree pentru metoda de plată selectată de client. Nonce-ul este apoi utilizat într-o cerere de tip POST către ruta sa stabilită în codul backend-ului și inclus în corpul cererii pentru a indica metoda de

plată.

După ce plata a fost efectuată cu succes, utilizatorul este redirecționat către o pagină care îl informează despre acest lucru, de unde are posibilitatea să se întoarcă înapoi la catalogul de produse, putând oricând să urmărească statusul comenzii din pagina de istoric al comenzilor.

4.2 Aplicația destinată administratorilor

Aplicația destinată administratorilor a fost concepută și dezvoltată pentru a facilita gestionarea eficientă a tuturor componentelor și proceselor esențiale care contribuie la funcționarea magazinului online. Prin intermediul acestui panou de control specializat, administratorii au acces la un set complet de instrumente și funcționalități care îi ajută să administreze și să monitorizeze activitatea magazinului.

4.2.1 Log in

Aplicația de administrare a magazinului este configurată pentru a functiona exclusiv intern, ceea ce înseamnă că este destintă utilizării doar de către administratorii magazinului și personalul autorizat, fiind o cu totul altă aplicație față de cea destinată utilizatorilor, făcând-o imposibil de accesat de cumpărători. Conturile administratorilor sunt create la momentul configurării aplicației, ceea ce înseamnă că ei nu trebuie decât să treacă prin procesul de autentificare pentru a accesa panoul principal de control.

Acest proces se realizează prin completarea câmpurilor pentru email și parolă, urmând ca administratorul să primească o eroare sau să fie redirecționat către panoul de control. De asemenea, aplicația le oferă posibilitatea de a ieși din cont, moment în care ei sunt redirecționați către prima pagină a aplicației.

4.2.2 Pagina de produse

Pagina de produse reprezintă una dintre cele 5 pagini principale la care un administrator are acces și în oferă acestuia posibilitatea de a vizualiza întreaga listă de albume stocate în baza de date a aplicației.

În cadrul său, administratorul se poate folosi de funcționalitatea de căutare după titlu pentru a găsi un album căruia poate să-i editeze informațiile. Pe lângă acestea, pagina de produse oferă și

opțiuni de vizualizare a detaliilor unui album, ceea ce-i permite accesul la informații precum titlul, artistul, anul lansării, link-urile folosite pentru imaginile de prezentare ale produselor, etc. Acestor funcționalități i se adaugă și posibilitatea de a șterge un album din baza de date, în cazul în care acesta nu mai este disponibil sau nu mai corespunde cerințelor pieței.

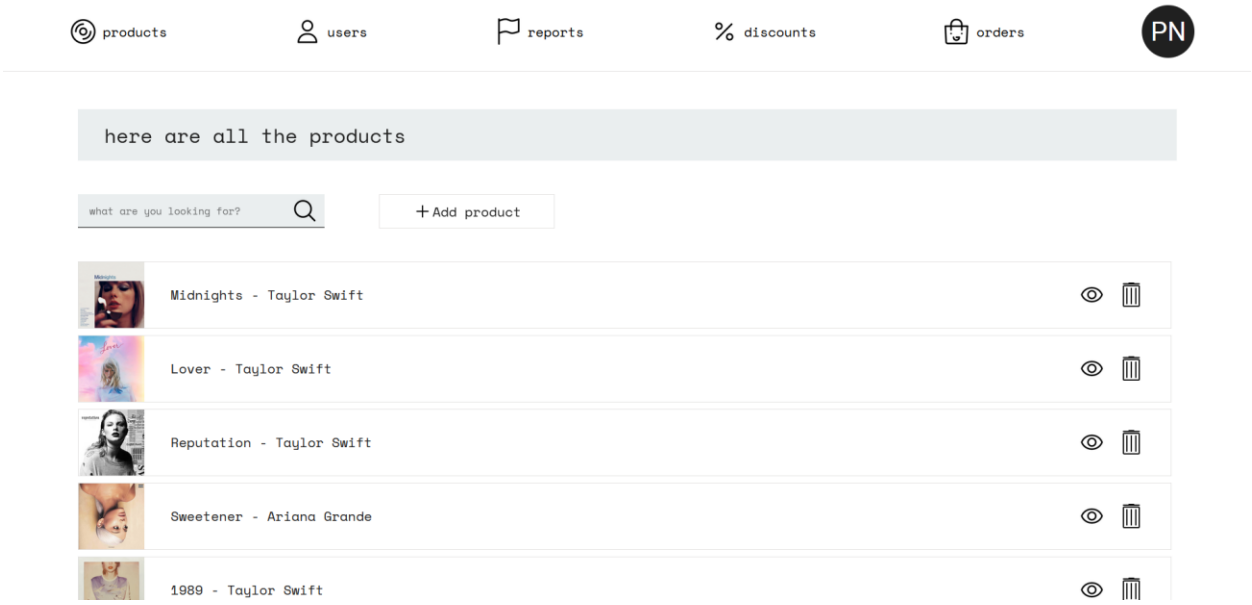

Figura 4.11 – Pagina de produse a administratorilor

4.2.3 Pagina de utilizatori

Pagina de utilizatori reprezintă de asemenea un punct cheie al panoului de control al administratorilor, deoarece le oferă posibilitatea de a gestiona și monitoriza conturile utilizatorilor înregistrati. Aceasta oferă informații precum numele utilizatorilor, adresele de email, data înregistrării, statusul contului (suspendat sau nu), numărul de abateri de la politicile platformei etc. Pe lângă asta, el are acces la funcționalitatea de suspendare a contului unui utilizator în cazul în care consideră că se abate de la regulile de integritate ale aplicației. Această vedere de ansamblu asupra utilizatorilor permite administratorului să le monitorizeze activitatea și să identifice eventuale probleme sau neconformități.

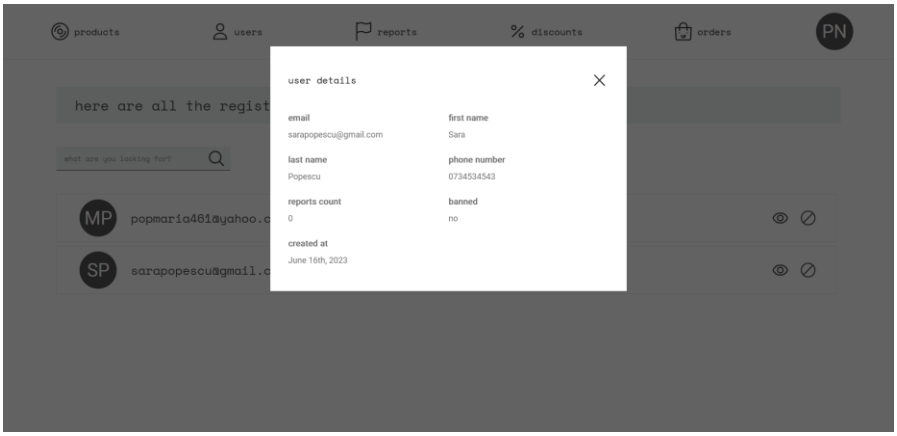

Figura 4.12 – Modal pentru detaliile unui utilizator

4.2.3 Pagina de rapoarte

Pagina de rapoarte, la fel ca și celelalte, reprezintă un instrument de o importanță crucială în ceea ce privește păstrarea integrității platformei. Aceasta furnizează administratorului informații valoroase despre experiența utilizatorilor și feedback-ul primit, permițându-i să ia măsurile necesare pentru îmbunătățirea interacțiunii cu aplicația.

Prin intermediul ei, el poate vedea lista de rapoarte pe care utilizatorii le trimit atunci când un comentariu li se pare ofensiv sau nepotrivit. Această funcționalitate reprezintă o modalitate prin care ei pot semnala conținul inadecvat și îi permit administratorului să ia măsuri.

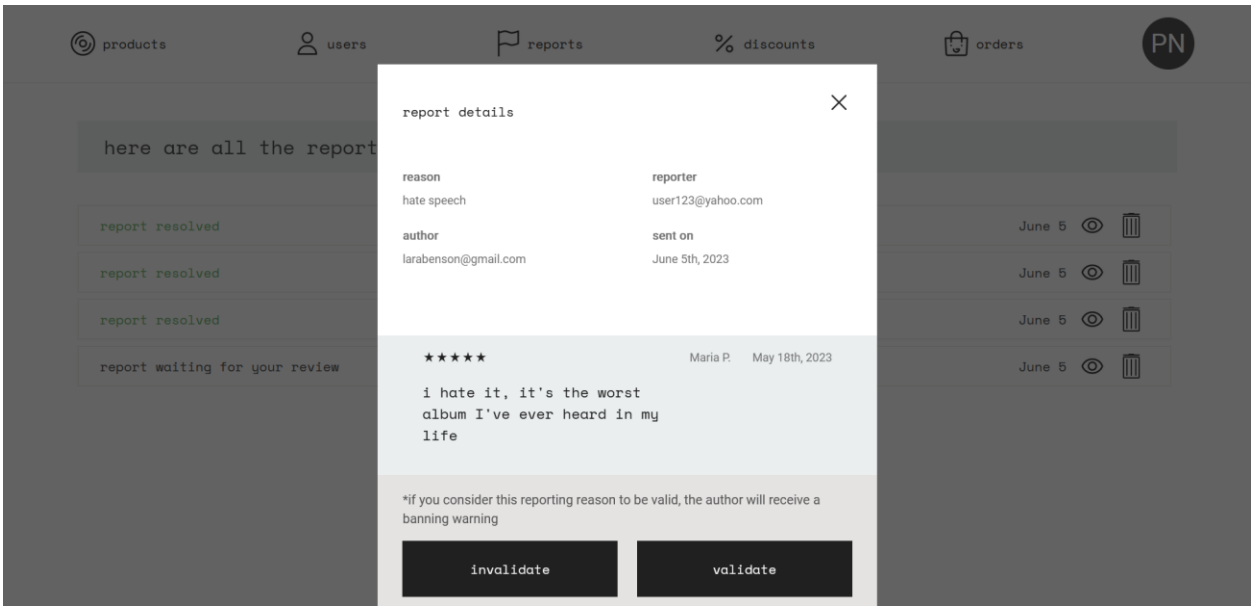

Figura 4.13 – Modal pentru revizuirea unui raport

Dacă administratorul decide că motivul pentru care o recenzie a fost raportată este unul întemeiat, el poate să apese de butonul „validate”, moment în care se întâmplă următoarele lucruri:

- Autorul comentariului primește o avertizare de suspendare a contului, iar la a treia avertizare primită contul său va fi suspendat permanent în mod automat;
- Cel care a raportat recenzia va primi o notificare care îl va informa că au fost luate măsuri împotriva autorului;
- Recenzia este ștearsă din baza de date;

  În caz contrar, cel care raportează comentariul va primi o notificare conform căreia obiectul raportului nu încalcă nicio regulă. În ambele cazuri, raportul din lista administratorului va fi marcat ca fiind rezolvat.

Pe lângă această funcționalitate, el are posibilitatea de a șterge un raport pentru cazul în care este rezolvat și ocupă spațiu de stocare inutil în baza de date.

4.2.4 Pagina de coduri de reducere

Pagina de coduri de reducere reprezintă o componentă a panoului de administrare prin intermediul căreia administratorii pot adăuga, edita sau șterge un cod de reducere. Pe lângă acestea, ei pot vizualiza detaliile unui cod, cum ar fi valoarea sa, valoarea reducerii pe care o oferă în timpul plasării de comenzi, data de început și data de sfârșit a valabilității sale.

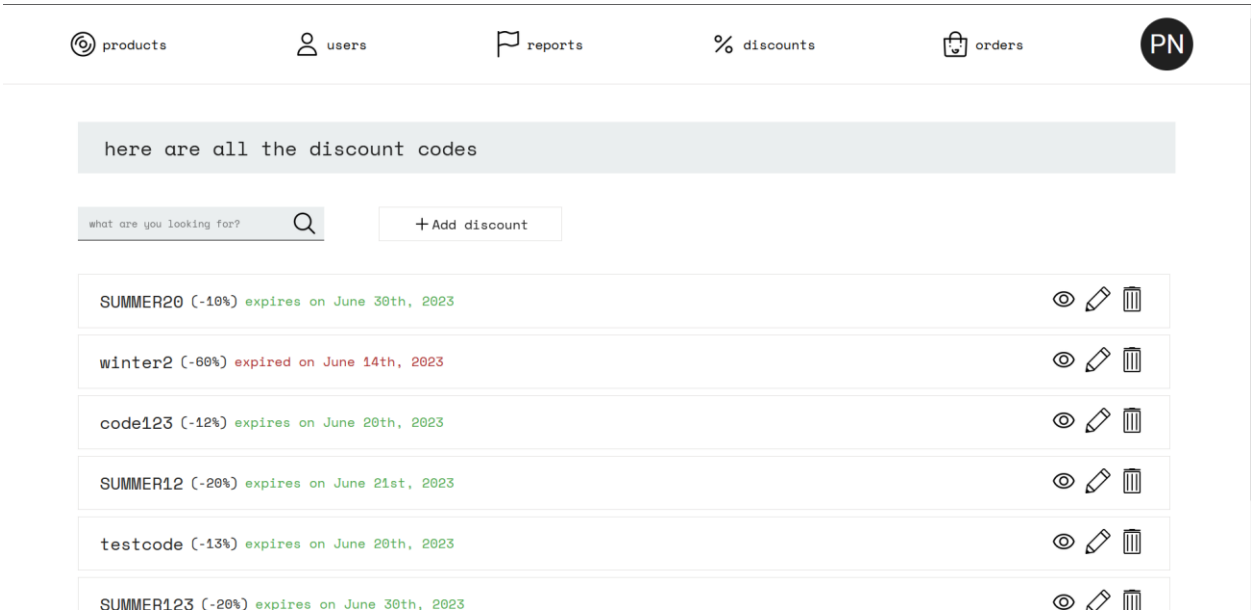

Figura 4.14 – Pagina de coduri de reducere a administratorilor

4.2.5 Pagina de comenzi

Pagina de comenzi reprezintă printre cele mai importante componente ale panoului de control pe care administratorii îl au la dispoziție în cadrul aplicației. Ea oferă o vedere cuprinzătoare asupra tuturor comenzilor plasate de către clienți.

Prin intermediul ei, este permisă o monitorizare și o gestionare eficientă a procesului de comandă, oferind posibilitatea de a vizualiza produsele achiziționate, cantitățile, adresa de livrare, datele de contact ale cumpărătorului și statusul comenzii.

Capitolul 5 – Backend

5.1 Baza de date

Așa cum am menționat și în cadrul capitolului 2, am creat baza de date a aplicației utilizând MongoDB, un sistem de gestionare a bazelor de date NoSQL. Având în vedere că datele sunt stocate ca documente care aparțin unor colecții, se facilitează stocarea și interogarea datelor într-un mod flexibil și scalabil. Prin intermediul funcționalităților avansate ale MongoDB, am putut gestiona eficient colecțiile de date, interoga informațiile și realiza operațiuni complexe de căutare, filtrare și sortare.

În ciuda faptului că este un sistem de gestionare a datelor NoSQL care nu impune restricții de tip relațional între entități, putem adapta conceptele diagramelor entitate-relație în contextul utilizării MongoDB pentru a înțelege mai ușor și mai clar relațiile dintre colecțiile stocate. Putem identifica entitățile dintr-o bază de date de tip SQL ca fiind colecțiile dintr-o bază de date MongoDB, atributele entităților ca fiind câmpurile documentelor din colecții, iar relațiile dintre entități drept referințe între documente sau înglobări de documente.

Entitățile care contribuie la conturarea bazei de date necesare sunt următoarele:

- USER: această colecție stochează informații despre utilizatori, cum ar fi numele, prenumele, adresa de mail, numărul de telefon, parola (criptată) etc., fiecare document reprezentând un utilizator al aplicației. Acesta poate fi un cumpărător sau un administrator.
- ADDRESS: această colecție este menită să rețină datele unei adrese ale unui utlizator, cum ar fi țara, orașul/regiunea respectivei țări, codul poștal etc
- INBOX ITEM: este folosită pentru a modela structura unei notificări pe care user-ul o primește referitor la un raport pe care l-a făcut, feedback pentru acel raport sau apariția unui nou cod de reducere
- ALBUM: entitatea care reprezintă elementul cheie al scopului aplicației, și anume structura unui produs din baza de date care conține toate datele necesare despre un album care poate fi adăugat în coșul de cumpărături sau lista de produse favorite;
- REVIEW: folosit pentru stocarea unei recenzii lăsate de către un utilizator pentru a-și oferi feedback-ul despre un produs pe care l-a achiziționat sau vizualizat și conține date precum titlul, comentariul sau rating-ul primit de un album;

- REPORT: reprezintă structura documentului care stocheaza informații despre rapoartele create de utilizatori atunci când aceștia raportează o recenzie a unui produs, conținând date precum motivul raportului, cine a scris comentariul și cine l-a raportat
- CART-ITEM: stochează detalii despre un element aflat în coșul de cumpărături al unui utilizator, reținând atât detaliile produsului adăugat, cât și identificatorul utilizatorul căruia îi aparține;
- WISHLIST-ITEM: are o structură similară cu „CART-ITEM" și reține date despre albumele pe care un utilizator le adaugă în lista sa de produse favorite;
- DISCOUNT: reprezintă colecția de documente în care se află informații despre un cod de reducere care poate fi aplicat în timpul unui proces de checkout; el conține date precum valoarea codului respectiv, valoarea reducerii pe care o oferă codul respectiv, data de la care acesta este valabil și data până la care este valabil;
- ORDER: structura unui document care conține toate datele necesare despre o comandă pe care un utilizator autentificat o poate plasa, cum ar fi identificatorul user-ului la care trebuie să ajungă produsele, datele adresei de livrare, codul unic al comenzii etc

Toate aceste colecții și documente din cadrul bazei de date oferă o perspectivă completă și organizată asupra datelor necesare cerințelor pe care proiectul trebuie să le îndeplinească. Fiecare colecție are un rol bine definit, permițând aplicației să funcționeze și să ofere funcționalități complexe utlizatorilor săi, fie cumpărători, fie administratori.

Pentru vizualizarea și evidențierea relațiilor dintre aceste entități, am realizat diagrama entitate-relație din figura de mai jos care ne ajută să înțelegem mai clar structura bazei de date și interdependențele sale. Am inclus în structura fiecărei entități numele câmpului însoțit de tipul de date pe care le poate reține.

Câmpurile scrise cu text îngroșat reprezintă identificatorul documentului a cărui valoare este generată automat în MongoDB și pot reprezenta echivalentul cheilor primare din tabelele bazelor de date de tip SQL. Câmpurile subliniate marchează prezența unei referințe către un document care aparține altei colecții, putând fi folosit ca o cheie străină. Câmpurile marcate cu * simbolizează faptul că sunt obligatorii.

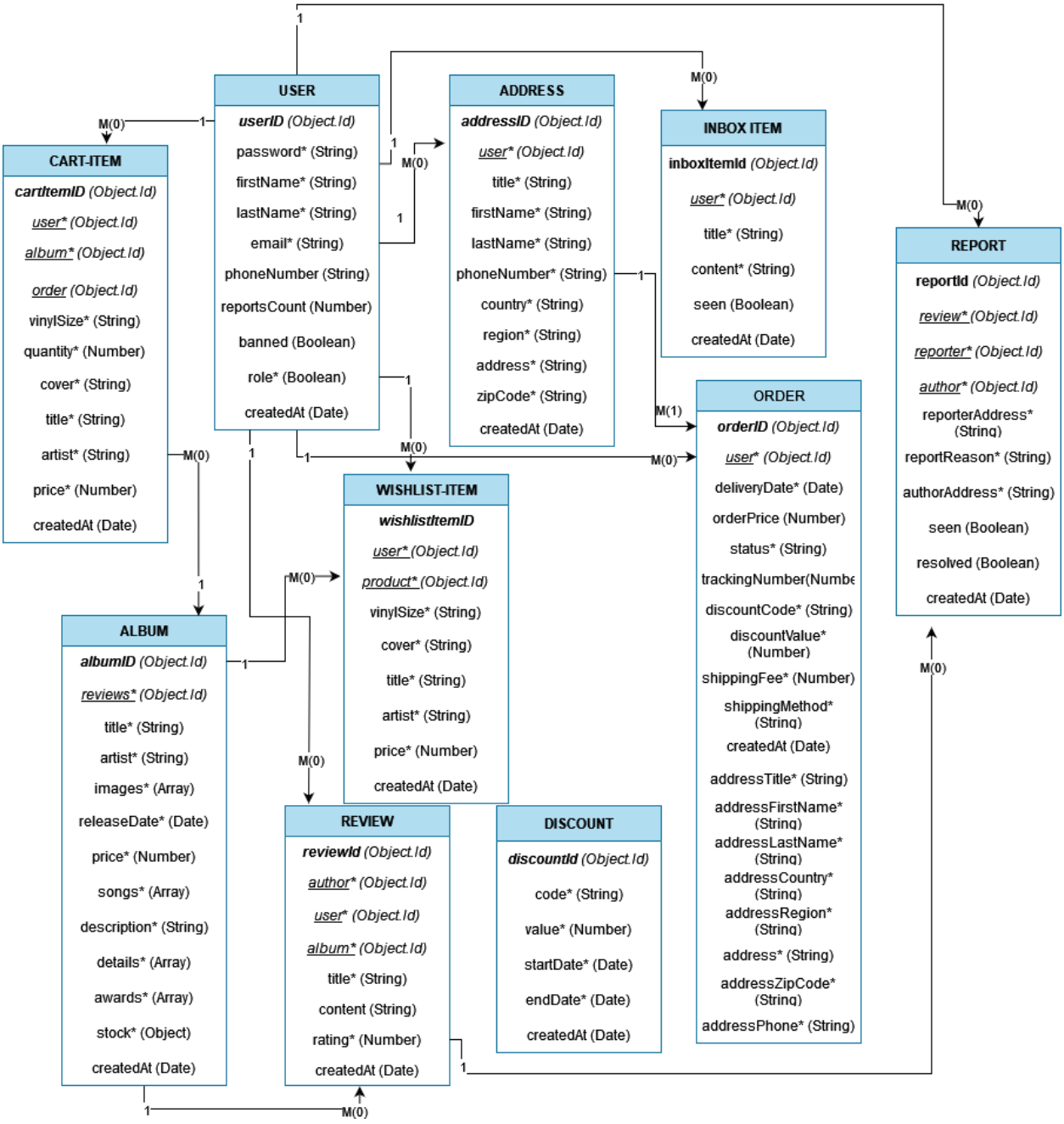

Figura 5.1– Diagrama entitate-relație

5.2 Validări de câmpuri

Pentru asigurarea integrității și coerenței informațiilor introduse de utilizatori, validarea datelor reprezintă o etapă esențială în procesul de dezvoltare a unei aplicații, drept pentru care acest subcapitol este dedicat unei analize detaliate asupra informațiilor pe care colecțiile bazei de date MongoDB le cuprind.

Pentru realizarea acestor validări, au fost folosite funcții și proprietăți specifice pe care librăria „Mongoose” le oferă, cum ar fi:

- „type”: cu ajutorul acestuia, putem stabili tipul de date al valorilor reținute în fiecare câmp, astfel că putem întâlni în structura documentelor create date de tip String, Number, Date, Boolean sau de tip Object.Id folosit pentru referințe către alte colecții de documente, fiind folosit pentru fiecare câmp al fiecărei scheme;
- „required”: această proprietate este folosită pentru a specifica dacă un câmp este sau nu obligatoriu și este de tip bool și poate avea doar valorile true sau false; a fost folosit pentru validarea tuturor câmpurilor din fiecare schemă’
- „unique”: proprietate folosită pentru a specifica dacă valoarea respectivului câmp este unică sau nu în cadrul colecției; acceptă valori de tip bool, iar atunci când i se atribuie valoarea „true”, se asigură faptul că nu vor exista alte documente în colecție care să aibă aceeași valoare în acel câmp; a fost folosit pentru câmpul de email din schema utilizatorilor;
- „default”: proprietate cu ajutorul căreia putem seta o valoare implicită a unui câmp; în cazul în care nu este specificată o valoare anume în timpul creării sau actualizării documentului, atunci câmpul va avea valoarea implicită; poate fi utilă dacă dorim asigurarea populării cu date indiferent de intrări; a fost folosită, de exemplu, pentru câmpurile „createdAt” din structura fiecărui document pentru a înregistra momentul creării fiecărui obiect;
- „trim”: proprietate de tip bool folosită pentru a elimina spațiile albe de la începutul și sfârșitul unui șir de caractere înainte de a salva datele; asigurând coerența datelor; a fost utilizată în proiect pentru câmpuri precum nume, prenume, adresa etc;
- „maxLength”: proprietate folosită pentru a specifica lungimea maximă admisă pentru un șir de caractere; dacă datele de intrare au o lungime mai mare decât cea stabilită, atunci va fi generată o eroare de validare; a fost utilizată, spre exemplu, pentru validarea câmpului pentru codul de reducere al documentelor de tip „DISCOUNT”; în mod similar funcționează și proprietatea de „minLength”, dar de data asta este specificată lungimea minimă admisă;
- „max”: utilizată pentru a preciza valoarea maximă a unui câmp de tip Number; la fel ca în cazul proprietății „maxLength”, Mongoose va genera un mesaj de eroare dacă datele de intrare sunt mai mari decât valoarea specificată; a fost folosită, spre exemplu, pentru a limita valoare de reducere a unui cod de discount la 100; pentru a specifica valoarea minimă admisă în schimb, se poate folosit proprietatea „min”
- „validate”: poate fi utilizată pentru a defini o funcție personalizată de validare pentru un

anumit câmp; în aplicația dată, a fost folosit, de exemplu, pentru validarea câmpului pentru reținerea numărului de telefon, astfel că primește ca argument datele de intrare și returnează o valoare de tip bool dacă acestea reprezintă un număr de telefon sau nu; pentru verificarea propriu-zisă a fost folosită expresia regulată de mai jos, cu ajutorul căreia ne asigurăm că se înregistrează un șir de caractere de lungime 10 ce cuprinde doar cifre;

```
validate: {
      validator: (value: string) => {
        return /^[0-9]{10}$/.test(value);
      },
      message: "Phone number must be a 10-digit number.",
    },
```

Figura 5.2 – Validator pentru numărul de telefon

5.3 Relaţiile dintre colecţii

În cadrul bazelor de date de tip SQL, relațiile dintre tabele pot fi clasificate în 3 categorii: one-to-one, one-to-many și many-to-many, ele descriind modul în care înregistrările dintr-o tabelă din baza de date sunt legate de înregistrările altei tabele.

Deși MongoDB este o bază de date de tip NoSQL, putem simula aceste tipuri de relații cu ajutorul referințelor:

- **One-to-many**:

Într-o relație dintre tabele de acest gen, o înregistrare dintr-o tabelă este asociată cu mai multe înregistrări aparținând altei tabele, iar pentru a simula acest lucru în MongoDB, am folosit conceptul de referințe, incluzând un câmp de înregistrare în colecția principală care să fie populat cu date de tip referință către documente din colecția secundară. De exemplu, în cadrul „Vinylla”, un utilizator poate avea asociate mai multe adrese, motiv pentru care, un document aparținând unei adrese conține și un câmp ce reprezintă referința către documentul utilizatorului căruia îi aparține aceasta.

Pentru celelalte tipuri de relații nu a fost nevoie de simulare deoarece ele nu sunt prezente în cadrul diagramei.

5.4 Operaţii CRUD

CRUD este un acronim pentru cele 4 operații de bază utilizate în cea mai mare parte a aplicațiilor care folosesc o bază de date și se referă la Create, Read, Update și Delete, ce reprezintă

un set de funcționalități necesare gestionării datelor și facilitării interacțiunii dintre utilizator și aplicație. Datorită acestor operații, backend-ul permite crearea, accesarea, actualizarea și ștergerea datelor de către utilizatori într-un mod eficient care contribuie la scalabilitatea și performanța aplicației.

Pentru implementarea acestora, au fost folosite diverse metode pentru a gestiona cererile clienților în interacțiunea cu baza de date:

- CREATE: operația de adăugare a unui document în cadrul unei colecții care implică următorii pași:
  - Definirea unui model: etapa în care se stabilește structura datelor și permite utilizarea metodelor oferite de Mongoose pentru interacțiunea cu baza de date
  - Instanțierea obiectului: etapa în care se setează valorile câmpurilor obiectului pentru a crea o nouă înregistrare
  - Salvarea înregistrării: etapa în care se trimite cererea de creare către baza de date realizată cu ajutorul metodei „save()”
  - Răspunsul la cerere: etapa în care putem trimite un răspuns clientului pentru a confirma crearea cu success a obiectului sau informarea cu privire la erorile întâlnite;

Pentru implementarea acestei operații a fost folosită metoda POST, una dintre metodele HTTP folosite pentru a trimite date către server. Prin trimiterea cererii către rutele corespunzătoare care au fost create, datele necesare sunt incluse în corpul cererii. De exemplu, în cadrul „Vinylla” acest tip de operație a fost realizat pentru înregistrearea datelor despre utilizatori, albume, recenzii, comenzi, etc

```
export const postReview = async (req: Request, res: Response) => {
  try {
    const { body } = req;
    await ReviewModel.create({ ...body });
    res.send({ status: "Ok, success!" });
  } catch (error) {
    res.send({ error, status: "error" });
  }
};
```

Figura 5.3 – Exemplu de operație de tip POST

- READ: operația de citire care este responsabilă de obținerea datelor din baza de date și returnarea acestora către utilizatori sau componente ale aplicației

  Pentru implementarea sa a fost folosită metoda GET, una dintre cele mai comune metode

HTTP folosită pentru a solicita date de la server, în următoarele etape:

- Definirea rutelor care vor determina adresa URL la care se va face cererea
- Configurarea funcției în cadrul căreia se pot folosi parametrii cu ajutorul cărora putem utiliza functii de citire oferite de biblioteca Mongoose, cum ar fi funcția „find”, ‚findById”, etc;
- Trimiterea răspunsului către client și tratarea erorilor;

Această operație a fost folosită pretutindeni în cadrul aplicației pentru returnarea de date precum detalii despre un album, lista de albume, lista de user, lista de adrese aparținând unui user, etc

```
export const getAlbums = async (req: Request, res: Response) => {
  try {
    const albums = await AlbumModel.find({});
    res.status(200).send({ status: "Ok, succes!", albums });
  } catch (error) {
    res.status(400).send({ error, status: "error" });
  }
};
```

Figura 5.4 – Exemplu de operație de tip GET

- UPDATE: operație care permite modificarea datelor datelor existente prin intermediul căreia le este permis utilizatorilor să actualizeze informațiile existente în aplicație

Pentru implementarea sa a fost folosită metoda UPDATE cu ajutorul căreia au fost actualizate date precum adresa unui utilizator, datele unui album, etc.

```
export const editAlbum = async (req: Request, res: Response) => {
  try {
    const albumId = req.params.id;
    const updatedAlbum = req.body;

    const album = await AlbumModel.findByIdAndUpdate(albumId, updatedAlbum, {
      new: true,
    });

    if (!album) {
      return res.status(404).send("Album not found");
    }

    return res.status(200).json(album);
  } catch (error) {
    console.error(error);
```

```
    return res.status(500).send("Error updating album");
  }
};
```

Figura 5.5 – Exemplu de operație de tip UPDATE

- DELETE: operație care permite ștergerea unuia sau a mai multor documente dintr-o colecție;

Pentru implementarea sa a fost folosită metoda DELETE cu ajutorul căreia se pot realiza ștergeri de adrese, utilizatori, albume, coduri de discount, etc;

```
export const deleteAllAlbums = async (req: Request, res: Response) => {
  AlbumModel.deleteMany({})
    .then((result) => {
      console.log(`Deleted ${result.deletedCount} albums.`);
      res.send(`Deleted ${result.deletedCount} albums.`);
    })
    .catch((error) => {
      console.log(`Error deleting albums: ${error}`);
      res.status(500).send("Error deleting albums.");
    });
};
```

Figura 5.6 – Exemplu de operație de tip DELETE

5.5 Tipuri de utilizatori

Pentru dezvoltarea aplicației „Vinylla”, am decis să definesc 3 tipuri de useri ce pot interacționa cu interfața și funcționalitățile acesteia:

1. ***Utilizatori de tip vizitator (guest user)***

Această categorie de utilizatori include vizitatorii care accesează feature-urile aplicației fără a fi înregistrați sau autentificați. Ei pot explora colecția de viniluri, atât din homepage, cât și din pagina care conține toate produsele, pot căuta anumite albume după titlu, pot folosi toate filtrele de căutare, toate tipurile de sortare disponibile, pot vedea detaliile unui produs, cum ar fi lista de piese conținute, genul în care se încadrează albumul, premiile pe care le-a obținut de-a lungul timpului, când a fost lansat și poate citi recenziile pe care le-a primit.

2. ***Utilizatori înregistrați (registered users)***

Această categorie de utilizatori este reprezentată de userii care și-au creat un cont în aplicație și sunt autentificați cu acel cont. Ei au privilegii extinse și beneficiază de toate funcționalitățile aplicației destinată clienților, astfel că, odată ce un user este autentificat, el poate, pe lângă tot ce poate face un guest user, să adauge produse atât în coș, cât și în lista de produse favorite, poate finaliza comenzi, poate lăsa review-uri produselor vizualizate, poate raporta un review care i se pare nepotrivit, își poate vedea istoricul comenzilor, lista de adrese salvate pe care le poate folosi în timpul procesului de checkout pe care le poate adăuga, edita sau șterge, își poate edita detaliile contului, cum ar fi numele, prenumele sau numărul de telefon, își poate vedea notificările și, nu în ultimul rând, poate beneficia de coduri de reducere pe care le poate aplica atunci când dorește să plaseze o comandă.

3. Administratori

Această categorie de useri este una specială, diferită de toate celelalte, deoarece ea interacționează cu o aplicație diferită, dezvoltată special pentru acest tip de user, având puteri extinse și responsabilități de gestionare. La fel ca celelalte tipuri de user, ei trebuie să fie autentificați cu un cont creat pe platformă, având acces la panoul de administrare și totodată, la aspectele cheie ale aplicației. Ei pot vedea lista de produse, pot căuta un album în funcție de titlul său, pot adăuga unul nou sau pot șterge sau edita unul vechi.

În ceea ce privește lista de useri, ei pot vedea lista lor reținută în baza de date și au posibilitatea fie de a vizualiza datele acestora, fie de a lua măsuri de suspendare ale contului pentru cazul în care contul pare fals sau cuprinde date sau comportamente care se află în contradictoriu cu regulile și politicile platformei.

De asemenea, are acces la lista de rapoarte, care conține informații cu privire la review-urile raportate de către utilizatori, având posibilitatea de a decide dacă un raport este unul valid sau nu sau de a-l șterge. Pe lângă asta, are acces la lista de coduri de reducere, având posibilitatea de a crea unul nou și de a edita sau șterge unul vechi.

5.6 Cum se realizează diferențierea?

Diferențierea dintre aceste 3 tipuri de useri se realizează cu ajutorul unui câmp din schema entității user-ilor, numit „role” și care poate primi doar 2 valori:

- „registered”, pentru utilizatorii aplicației destinate clienților care și-au creat un cont pe platformă

- „admin”: pentru administratorii care au acces la panoul de control al componentelor principale

```
role: {
    type: String,
    required: true,
    enum: Object.values(userRole),
  },
```

Figura 5.7 – Câmpul destinat rolului utilizatorului

```
export const userRole = Object.freeze({
  admin: "admin",
  registered: "registered",
});
```

Figura 5.8 – Enumerarea pentru rolul utilizatorului

Aceste valori sunt alocate atunci cand se execută requestul de creare a unui user, și din moment ce există două aplicații separate, există 2 endpoint-uri diferite de tip POST care atribuie rolul utilizatorului corespunzător în funcție de aplicația din care se face request-ul.

```
await UserModel.create({
      ...body,
      password: hashedPassword,
      role: userRole.registered,
    });
```

Figura 5.9 – Momentul atribuirii rolului utilizatorului

Așa cum se poate observa în imaginea de mai sus, valoarea user-ului este asociată în mod implicit în structura endpoint-ului specific.

5.7 Stocarea parolelor

Pentru stocarea parolelor utilizatorilor în cadrul bazei de date, s-a ales folosirea „bcrypt”, o bibliotecă de hashing folosită în domeniul securității informației, în special pentru criptarea parolelor.

Motivația de a folosi aceasta librărie este securitatea pe termen lung pe care o oferă,

deoarece utilizează o combinație de funcții de hashing și salt pentru a crea un hash foarte securizat al parolei, ceea ce înseamnă că și dacă baza de date a întregii aplicații este compromisă, parolele user-ilor vor rămâne protejate în fața atacurilor de tip brute-force sau chiar și de tip dicționar.[8]

În primul rând, „Bcrypt” este proiectat să fie un algoritm lent, lucru care face ca atacurile de tip brute-force să fie dificile și mai ales costisitoare, ceea ce înseamnă că și dacă un atacator are acces la hash-urile parolelor, va fi foarte dificil să găsească parola inițială[9].

În al doilea rând, el include un salt unic pentru fiecare parolă, salt-ul fiind un șir aleatoriu adăugat parolei înainte de procesul propriu-zis de hashing, fapt care face ca „Bcrypt” să fie rezistent și la atacuri de tip rainbow table, care presupun ca atacatorii să folosească tabele precalculate cu care găsesc rapid hash-urile corespunzătoare parolelor comune, iar utilizarea unui salt unic pentru hashing-ul fiecărei parole face ca aceste tabele să fie inutile.

În al treilea rând, „Bcrypt” permite specificarea unui cost de hashing, ceea ce permite ajustarea timpului necesar calculării hash-ului în funcție de puterea de calcul disponibilă în cadrul server-ului, fapt care îi oferă un plus de flexibilitate și accesibilitate.

Având în vedere argumentele prezentate anterior, am considerat folosirea bibliotecii „Bcrypt” alegerea optimă pentru sporirea securității datelor utilizarilor.

În aplicația "Vinylla", algoritmul de hashing furnizat de biblioteca bcrypt a fost utilizat în două scopuri principale: criptarea parolei la crearea contului utilizatorului și verificarea corectitudinii parolei în timpul procesului de autentificare.

În momentul creeării contului utilizatorului, algoritmul bcrypt a fost utilizat pentru a securiza parola introdusă de utilizator. Parola în clar a fost trecută prin procesul de hashing bcrypt, utilizând un salt unic generat automat și un cost de hashing specificat. Aceasta a generat un hash criptografic unic, care a fost stocat în baza de date în locul parolei în clar.

```
export const postUser = async (req: Request, res: Response) => {
  try {
    const { body } = req;
    const { password } = body;

    const saltRounds = 10;
    const hashedPassword = await bcrypt.hash(password, saltRounds);

    await UserModel.create({
      ...body,
      password: hashedPassword,
      role: userRole.registered,
    });
```

```
    res.status(200).send({ status: "Ok, success!" });
  } catch (error) {
    res.status(400).send({ error, status: "error" });
  }
};
```

Figura 5.10 – Funcția pentru crearea unui utilizator

În timpul procesului de autentificare, algoritmul a fost utilizat pentru a verifica corectitudinea parolei introduse de utilizator. Parola introdusă de utilizator a fost comparată cu hash-ul stocat în baza de date folosind funcția de verificare oferită la dispoziție de bcrypt. Dacă parolele se potriveesc, autentificarea este considerată reușită, iar utilizatorul are acces la datele contului său. Această verificare se realizează fără a dezvălui sau a accesa parola în clar, ceea ce adaugă un nivel suplimentar de securitate.

```
export const loginUser = async (req: Request, res: Response) => {
  try {
    const { body } = req;
    const { email, password } = body;
    const user = await UserModel.findOne({ email });

    if (!user) {
      return res
        .status(401)
        .send({ error: "Invalid email or password", status: "error" });
    }

    const isPasswordValid = await bcrypt.compare(password, user.password);

    if (!isPasswordValid) {
      return res
        .status(401)
        .send({ error: "Invalid email or password", status: "error" });
    }




    res.status(200).send({ status: "Ok, success!", user });
  } catch (error) {
    res.status(400).send({ error, status: "error" });
  }
};
```

Figura 5.11 – Funcția pentru autentificarea unui utilizator

Prin utilizarea bcrypt în aceste două etape cruciale, aplicația "Vinylla" asigură securitatea și confidențialitatea datelor utilizatorilor. Utilizarea unui algoritm de hashing puternic, precum bcrypt, ajută la protejarea parolelor împotriva atacurilor brute-force și a accesului neautorizat la informațiile utilizatorilor.

5.8 Procesul de căutare și navigare

În orice aplicație, în special în cazul unei aplicații de ecommerce specializate în vânzarea de produse, cum ar fi un shop online de viniluri, funcționalitatea de căutare de produse joacă un rol crucial în asigurarea unei interacțiuni plăcute și eficiente.

De accea, am implementat o funcție de căutare care să sară în ajutorul unui user în cazul în care aceste dorește să caute un album anume, aceasta realizându-se în funcție de numele unui album. Acest tip de căutare reprezintă una dintre cele mai comune moduri prin care userii caută un anumit produs, permițându-le să introducă în bara de căutare un titlu sau o parte din acesta pentru a găsi rezultate relevante care să corespundă interogării făcute.

```
export const searchAlbum = async (req: Request, res: Response) => {
  try {
    const title = req.params.title;
    const regex = new RegExp(title, "i");
    const albums = await AlbumModel.find({ title: regex });
    res.send({ status: "Ok, success!", albums });
  } catch (error) {
    res.send({ error, status: "error" });
  }
};
```

Figura 5.12 – Funcția pentru căutarea unui album

Funcționalitatea de căutare a fost implementată folosind un endpoint care primește ca și parametru textul în funcție de care se va face căutarea. Punctul cheie al acestei bucăți de cod o reprezintă construirea unei expresii regulate „regex” care primește 2 parametri în cadrul constructorului:

- „title”: variabila care conține titlul albumului pe care utilizatorul îl caută
- „i”: modificator folosit pentru ca procesul de căutare să nu depinde de formatarea literelor din titlu, astfel că un text formatat „abc” va fi echivalent cu un text formatat „AbC” sau „ABC”

De aceea, expresia regulată a fost folosită în acest context pentru a permite o căutare precisă și flexibilă în funcție de numele unui album, acoperind o gamă largă de posibilități de scriere a titlurilor albumelor.

În dezvoltarea aplicației destinată cumpărătorilor, această functionalitate este disponibilă doar din bara de navigare, însă în panoul de control al administratorilor, ea apare în:

- Pagina de albume din stoc, unde similar cu aplicația principală pentru cumpărători, administratorii pot căuta albume în funcție de titlul lor. Aceasta le oferă posibilitatea de a găsi rapid un album specific în stocul magazinului.
- Pagina de utilizatori, în cadrul căreia ei au acces la o pagină specială unde pot căuta utilizatori înregistrați pe platformă. Căutarea se realizează în funcție de adresa de email a utilizatorilor, permițându-se astfel identificarea rapidă a unui anumit utilizator înregistrat.
- Pagina de coduri de discount care permite administratorilor să gestioneze codurile de discount disponibile în magazin. Funcționalitatea de căutare este adăugată pentru a permite căutarea codurilor de discount în funcție de valoarea lor, fapt care facilitează administrarea și actualizarea rapidă a codurilor de discount existente.

5.9 Filtre, sortări și paginație

De asemenea, în cadrul aplicației au fost adăugate și funcționalitățile de sortare, filtrare și paginare a rezultatelor de căutare, instrumente esențiale pentru ajutarea utilizatorilor de a găsi mai rapid și eficient albumele căutate, îmbunătățind totodată experiența generală.

Filtrele reprezintă opțiuni de selecție pentru restrângerea gamei de produse afișate deja, care în cadrul proiectului „Vinylla”, folosește diferite criterii pentru a permite utilizatorilor să-și rafineze căutarea. Aceste criterii pot include genul muzical, artistul și decada în care albumul a fost lansat. În plus, filtrele mai sunt folosite și pentru secțiunea de recenzii a fiecărui produs pentru afișarea lor fie în întregime, fie în funcție de numărul de stele acordat.

Sortarea, în schimb, permite utilizatorilor să ordoneze rezultatele căutării, astfel că există opțiunile de sortare în funcție de titlul sau prețul unui album, în ordine crescătoare sau descrescătoare. Pe lângă asta, funcționalitatea de sortare este folosită și pentru afișarea recenziilor oferite unui produs în funcție de numărul de stele primit sau de data la care au fost scrise, în ordine crecătoare sau descrescătoare.

Atât filtrele, cât și sortările, pot fi aplicate în același timp, ceea ce oferă un plus de complexitate întregului proces de navigare prin colecția de viniluri disponibile. Toate acestea au

fost realizate cu ajutorul unui singur endpoint, în cadrul căruia se extrag parametrii de interogare care pot include: „genre”, „artist”, „era” (decada în care a fost lansat albumul), „sortBy” (criteriul de sortare, adică titlul sau prețul) și „order”, adică ordinea de sortare. După aceea, se definește un obiect care reprezintă query-ul cu ajutorul căruia va fi interogată baza de date. Important de menționat este faptul că se poate aplica fie filtrul pentru genul muzical, fie cel pentru artist, iar acesta poate fi combinat cu cel pentru eră. La aceste filtre se poate adăuga și opțiunea de sortare.

De asemenea, paginația este folosită pentru împărțirea rezultatelor interogării în blocuri mai mici, astfel încât să nu fie afișate toate rezultatele simultan pe o singură pagină. Aceasta a fost folosită atât pentru pagina de produse, cât și pentru secțiunea de recenzii a produselor.

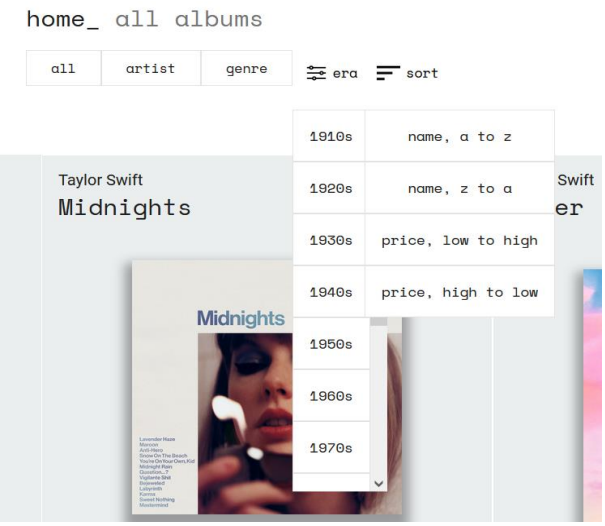

Figura 5.13 – Filtrul pentru decadă și tipurile de sortări

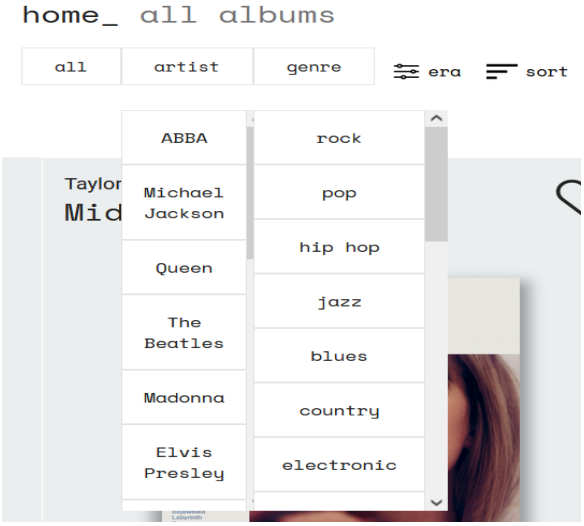

Figura 5.14 – Filtrele pentru artist și genul muzical

Capitolul 6 – Concluzii

6.1 Concluzii

Prin prezenta lucrare, am urmărit a pune în lumină atât conceptul, cât și procesul de cumpărare în cadrul unui magazin online de viniluri. Am subliniat importanța acestui proces atât pentru comerciant, cât și pentru clienți, evidențiind avantajele și beneficiile oferite de această platformă virtuală.

“Vinylla” își propune să devină o oportunitate inestimabilă pentru pasionații de muzică de a-și completa colecțiile de viniluri cu albume de calitate, motiv pentru care, prin intermediul unei interfețe prietenoase și bine structurate, utilizatorii pot explora diverse categorii de muzică și căuta albumele.

În final, magazinul online de viniluri reprezintă un spațiu captivant pentru pasionații de muzică, oferindu-le posibilitatea de a explora și achiziționa albume autentice.

Bibliografie

[1] Wikipedia contributors. "TypeScript." Wikipedia, Wikimedia Foundation, 10 iunie 2023, URL: https://en.wikipedia.org/wiki/TypeScript, last accessed: 10 iunie 2023

[2] Abhimanyu Krishnan, „TypeScript vs JavaScript: Which is Best in 2023", 15 noiembrie 2022, url: https://hackr.io/blog/typescript-vs-javascript, last accessed: 10 iunie 2023

[3] MongoDB. „Why Use MongoDB and When to Use It?", url: https://www.mongodb.com/why-use-mongodb, last accessed: 10 iunie 2023

[4] Redux Toolkit, „Getting Started with Redux Toolkit", url: https://redux-toolkit.js.org/introduction/getting-started, last accessed: 11 iunie 2023

[5] Mozilla Contributors. (2021). @keyframes. Mozilla Developer Network. https://developer.mozilla.org/en-US/docs/Web/CSS/@keyframes, last accessed: 11 iunie 2023

[6] Brad Frost, 2013, „Atomic Design", url: https://www.softouch.on.ca/kb/data/Atomic%20Design.pdf, last accessed: 11 iunie 2023

[7] Adish wirteups, „Principles of atomic design", 21 iunie 2022 https://medium.com/galaxy-ux-studio/principles-of-atomic-design-7b03a30c3cb6, last accessed: 11 iunie 2023

[8] Dan Aries, „Hashing in Action: Understanding bcript" https://auth0.com/blog/hashing-in-action-understanding-bcrypt/, last accessed: 12 iunie 2023

[9] Rishi Raman, „The Ultimate Guide to Bcrypt and Authentication Protocols", 02 noiembrie 2021, https://clerk.com/blog/bcrypt-hashing-authentication-encryption, last accessed: 12 iunie 2023

[10] MongoDB, „Model One-to-Many Relationships with Document References", https://www.mongodb.com/docs/manual/tutorial/model-referenced-one-to-manyrelationships-between-documents/, last accessed: 10 iunie 2023

[11] Alexandra, „What are CRUD Operations: How CRUD Operations Work, Examples,

Tutorials & More”, 24 februarie 2023, https://stackify.com/what-are-crud-operations/, last accessed: 15 iunie 2023

[12] MongoDB, „MongoDB CRUD Operations”, https://www.mongodb.com/docs/manual/crud/, last accessed: 15 iunie 2023

[13] Nathan Sebhastian, „Building forms with Formik in React”, 28 ianuarie 2022, https://blog.logrocket.com/building-forms-formik-react/, last accessed: 16 iunie 2023