

UNIVERSITATEA POLITEHNICA BUCUREŞTI
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
DEPARTAMENTUL CALCULATOARE



PROIECT DE DIPLOMĂ

SwapIt: Aplicație colaborativă pentru gestionarea schimbului de obiecte

Olteanu Alexandru

Coordonator științific:
Prof. Dr. Ing. Mariana Ionela Mocanu

BUCUREŞTI

2024

Cuprins

Sinopsis	4
Mulțumiri	4
1 Introducere	5
1.1 Context	5
1.2 Problema	6
1.3 Obiective	7
1.4 Structura lucrării.....	8
2 Analiza și specificarea cerințelor.....	9
2.1 Motivația	9
2.2 Cerințe non-funcționale	9
2.2.1 Cerințe tehnice pentru utilizatori	9
2.2.2 Confidențialitate și Securitate	10
2.2.3 Performanță și viteză	10
2.2.4 Scalabilitate.....	10
2.2.5 Mantenabilitate	10
2.3 Cerințe funcționale.....	10
2.4 Diagrama use-case a aplicației	12
3 Studiu de piață / Abordări existente.....	13
3.1 Sondaj De Opinie.....	14
3.2 Analiză Schimburi.ro	17
3.3 Analiză Listia.com.....	19
3.4 Analiză Ilovefreegle.org.....	21
3.5 Concluzii	23
4 SOLUȚIA PROPUȘĂ.....	24
4.1 Tehnologii utilizate	24
4.1.1 React.js	24
4.1.2 Firebase.....	25
4.1.3 Spring Boot.....	25
4.1.4 PostgreSQL.....	26
4.2 Arhitectura aplicației.....	26

4.2.1	Microservicii.....	26
4.2.2	Server de configurări.....	30
4.2.3	Baza de date.....	32
4.3	Descrierea fluxului funcționalităților	36
5	DETALII DE IMPLEMENTARE	39
5.1	Securitate	39
5.1.1	JWT token	40
5.2	Autentificare și Înregistrare	42
5.2.1	BCryptPasswordEncoder.....	42
5.2.2	Autentificare și înregistrare locală	42
5.2.3	Autentificare și înregistrare prin Oauth2.....	44
5.2.4	Generarea unui username unic	46
5.3	Audit al acțiunilor	47
5.4	Operații Programate.....	48
5.5	Paginare și întoarcerea eficientă a rezultatelor	49
5.6	Distanța Levenshtein.....	51
5.7	Integrarea librăriei Lucene	51
5.8	Caching	55
5.9	Tratarea erorilor	56
6	Evaluarea rezultatelor.....	59
6.1	Corectitudinea Aplicației.....	59
6.2	Performanța Aplicației	60
6.2.1	Analiza performanței folosind Lighthouse.....	60
6.2.2	Analiza performanței folosind JMeter	62
6.3	Experiența utilizatorilor & Design-ul interfeței.....	64
6.3.1	Pagina Acasă	64
6.3.2	Pagina de Autentificare.....	66
6.3.3	Profilul Utilizatorului	68
6.3.4	Acțiunile Administratorului.....	72
6.3.5	Crearea si Modificarea Unui Produs	74
6.3.6	Pagina de Produs	76
6.3.7	Categoriile de Produse	77

7	Concluzii	77
7.1	Dezvoltări Ulterioare	78
8	Bibliografie	79
9	Anexe	81

SINOPSIS

Într-o lume în care industriile sunt preocupate tot mai mult de sustenabilitate și de reducerea costurilor, aplicația SwapIt vine în ajutorul acestei mișcări sub forma unei platforme ce facilitează schimbul de obiecte între utilizatori într-o manieră simplă, încercând să pună o amprentă diferită în piață în comparație cu aplicațiile ce au la bază tranzacțiile obișnuite. Platforma permite utilizatorilor să schimbe obiecte de care nu mai au nevoie (pe care era dificil să le vândă cuiva) cu produse ce le sunt necesare. Un exemplu ar fi jucăriile copiilor mici de care aceștia se plăcătesc după câteva săptămâni. Este de preferat realizarea schimbului de jucării cu altă persoană și astfel să rezolvăm problema cumpărării altora noi.

Aplicația oferă o gamă largă de funcționalități, printre care căutarea avansată a produselor în funcție de cuvinte cheie ce oferă rezultate relevante chiar și la scrierea greșită a acestora, filtrarea după anumite criterii, gestionarea pe categorii a produselor, adăugarea de noi obiecte în platformă sau în lista de produse favorite.

Securitatea reprezintă un aspect esențial al platformei, asigurând protejarea datelor utilizatorilor și a informațiilor cu caracter privat printr-un sistem adecvat de autentificare și criptare a informațiilor. Sistemul de administrare permite gestionarea eficientă a conținutului și utilizatorilor, oferind totodată instrumente pentru monitorizare și control. Funcționalitățile administrative includ ștergerea produselor neadecvate, vizualizarea jurnalului de activitate în cadrul aplicației și gestionarea utilizatorilor, inclusiv posibilitatea de a aplica sancțiuni temporare sau permanente.

Aplicația SwapIt demonstrează cum tehnologia poate facilita schimburile de obiecte într-un mod simplu și ecologic. Cu funcționalitățile sale, platforma ajută utilizatorii să economisească resurse și să reducă risipa. Este oferită astfel o soluție eficientă și convenabilă pentru cei care doresc să adopte un stil de viață sustenabil, transformând gestionarea bunurilor personale într-o experiență plăcută și utilă.

MULȚUMIRI

Pe această cale, aş dori să îi mulțumesc doamnei Prof. Dr. Ing. Mariana Ionela Mocanu pentru îndrumarea atentă pe parcursul realizării acestei lucrări. Vă mulțumesc pentru feedback-ul regulat și constructiv, pentru ideile valoroase privind funcționalitățile și pentru clarificarea tuturor aspectelor neclare întâlnite în acest proces. Aprecierea mea se îndreaptă către dumneavoastră și pentru răbdarea și susținerea oferită, care au fost esențiale în finalizarea acestui proiect.

1 INTRODUCERE

1.1 Context

Lumea ce ne înconjoară aderă spre o digitalizare continuă și astfel utilizarea eficientă a resurselor devine de o importanță tot mai ridicată. În acest sens, formele de îndemn la reciclare și de adoptare a unei economii circulare câștigă tot mai multă atracție. Aplicația SwapIt urmează această tendință, venind cu o soluție relativ nouă pe piață, oferind utilizatorilor posibilitatea de a-și gestiona eficient și ecologic bunurile materiale.

În ultimii ani, conștientizarea impactului asupra mediului a activităților de risipă ce continuă să se accentueze a determinat oamenii să caute alternative la cumpărăturile tradiționale. Schimbul de bunuri reprezintă o soluție practică pentru prelungirea vieții produselor, reducerea costurilor și economisirea resurselor. De asemenea, o dată cu pandemia COVID-19, această nevoie a avut un trend ascendent, fiind stimulate interacțiunile online și schimburile fără contact direct.

Conform unui studiu realizat de Census Bureau în 2020, pandemia a dus la o creștere semnificativă a comerțului online, inclusiv a platformelor de schimb și marketplace-urilor. De exemplu, comerțul electronic a înregistrat o creștere de 43% în 2020, primul an al pandemiei, crescând de la 571,2 miliarde USD în 2019 la 815,4 miliarde USD în 2020 [1]. De asemenea, un raport realizat de Legatum Center for Development and Entrepreneurship de la MIT arată că pandemia a accelerat semnificativ comerțul electronic în regiunea MENA, evidențiind creșterea rapidă a comportamentului de cumpărare online și adaptarea companiilor la noile condiții de piață [2].

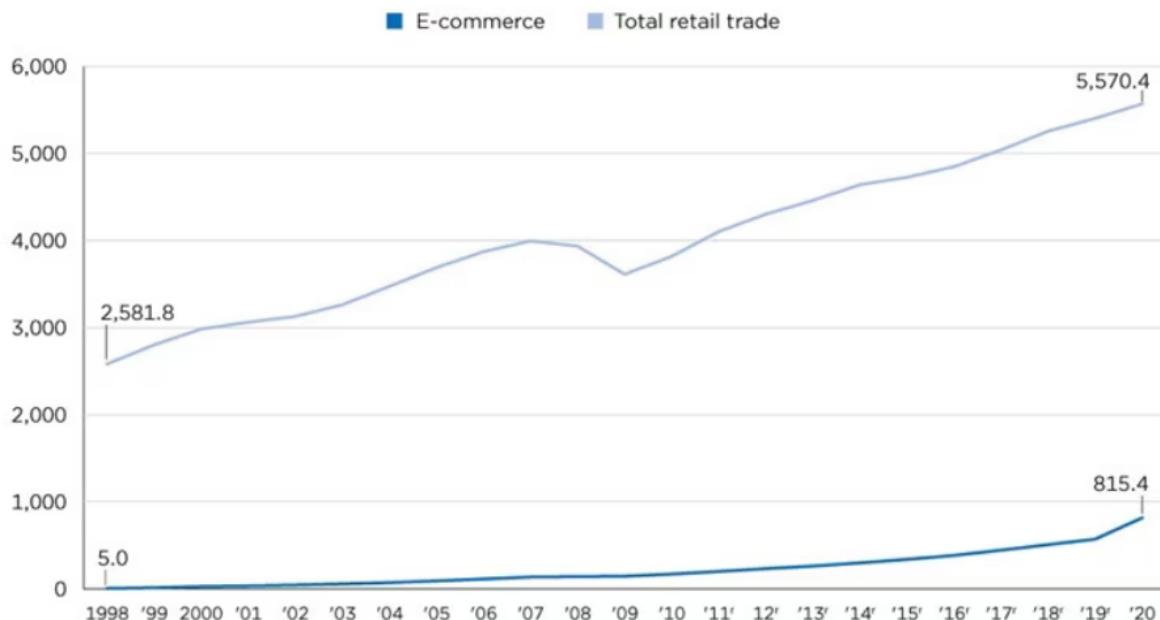


Figura 1: Vânzările anuale estimate ale comerțului cu amănuntul și comerțului electronic din SUA 1998-2020 [1]

Un factor esențial în dezvoltarea platformelor de schimb este dorința de a construi un mediu sigur în care utilizatorii interacționează pentru a-și satisface nevoile într-un mod sustenabil, astfel că SwapIt nu doar că facilitează schimbul de bunuri, dar încurajează și un stil de viață responsabil.

Pentru a răspunde cerințelor actuale, SwapIt își propune să ofere un design modern și o gamă largă de funcționalități precum căutare optimizată de produse după cuvinte cheie, filtrarea eficientă a rezultatelor și o gestionare detaliată a profilurilor utilizatorilor și produselor. Securitatea datelor și a tranzacțiilor este o prioritate, asigurând o experiență sigură și de încredere pentru toți utilizatorii platformei.

1.2 Problema

Observ cu îngrijorare cât de rapid devin inutile multe dintre obiectele pe care le deținem. De exemplu, echipamentele sportive, aparatelor electronice sau hainele sunt deseori lăsate deoparte după ce nu mai sunt necesare sau devin demodate. Aceasta nu doar că duce la risipă, dar și la cheltuieli continue pentru a cumpăra noi produse, contribuind la o cultură a consumului excesiv.

Piața de schimb și vânzare a obiectelor uzate este extrem de fragmentată și ineficientă. În prezent, majoritatea schimburilor de acest tip se realizează prin intermediul grupurilor de Facebook sau a altor platforme sociale similare. Aceste soluții, deși populare, prezintă numeroase probleme, printre care lipsa securității și protecției datelor personale, riscul de fraudă și dificultatea în a găsi parteneri de schimb de încredere. Tranzacțiile se desfășoară adesea în medii nesigure, iar utilizatorii sunt expuși la diverse riscuri.

Mai mult, lipsa unui sistem bine pus la punct pentru administrarea și monitorizarea produselor și utilizatorilor face dificilă gestionarea eficientă a acestor platforme. Produsele neadecvate sau utilizatorii neserioși rămân deseori necontrolați, ceea ce afectează negativ experiența globală a utilizatorilor. Situațiile în care produsele nu corespund descrierii sau în care utilizatorii sunt neserioși sunt frecvente și descurajante pentru cei care ar dori să participe la astfel de schimburi.

În plus, aceste platforme fragmentate nu oferă funcționalități avansate de căutare și filtrare, ceea ce îngreunează găsirea rapidă și eficientă a obiectelor dorite. De multe ori, utilizatorii sunt frustrați de rezultatele irelevante sau de dificultatea în utilizarea platformelor, ceea ce reduce semnificativ eficiența schimburilor.

Astfel, există o nevoie clară și urgentă de o soluție centralizată, sigură și eficientă care să faciliteze schimbul de obiecte într-o manieră sustenabilă. Este esențial să existe o platformă care să protejeze datele utilizatorilor, să ofere un sistem de administrare robust și să încurajeze practici responsabile de consum. O astfel de soluție ar contribui semnificativ la reducerea risipei și la promovarea unui stil de viață mai sustenabil și responsabil, transformând gestionarea bunurilor personale într-o experiență pozitivă și utilă.

1.3 Obiective

Obiectivul principal al acestei platforme este de a inspira cât mai multe persoane să ofere o nouă viață produselor nefolosite. Scopul este dezvoltarea unei platforme ușor de utilizat, cu un design plăcut, securitate ridicată și un sistem de gestionare și căutare a produselor bine organizat. Pentru a atinge acest obiectiv, este necesară îndeplinirea unei serii de cerințe specifice, structurate pe categorii:

Cerințe de Piață:

- **Acces Simplificat la Funcționalitățile Platformei:** Simplificarea procesului de înregistrare și gestionare atât a profilului cât și a produselor pentru a atrage și reține un număr cât mai mare de clienți.
- **Suporț Îmbunătățit:** Integrarea în platformă a unui buton de mesagerie în timp real, asigurând astfel o comunicare promptă și eficientă între utilizatori și echipa de suport.
- **Asigurarea Securității și Încrederii în Platformă:** Dezvoltarea unui sistem complet și sigur de verificare a utilizatorilor și de evaluare a produselor ce are ca scop minimizarea riscurilor și construirea încrederii în cadrul comunității. În prim-plan se află securitatea și transparența în procesul de schimb al unui produs față de utilizatori, ceea ce va duce la un procent mai mare de utilizatori care ajung să realizeze schimburi multiple în cadrul aplicației.

Cerințe de Design:

- **Interfață Intuitivă:** Realizarea unei interfețe care să fie ușor de navigat și de înțeles, folosind un design simplu și modern pentru a minimiza confuzia și a maximiza eficiența utilizatorilor.
- **Organizare și Structurare:** Asigurarea unei structuri logice a funcționalităților platformei, care să permită utilizatorilor să găsească într-un timp util produsele dorite, beneficiind de funcționalități avansate de căutare și paginare eficientă a rezultatelor.
- **Design și Funcționalitate:** Realizarea unui design vizual atrăgător, care să îmbine elemente de interacțiune moderne cu funcționalitatea, oferind astfel o platformă atractivă și utilă, care să satisfacă nevoile utilizatorilor și să încurajeze să revină.

Cerințe Tehnice:

- **Arhitectură pe Microservicii:** Implementarea unei arhitecturi de microservicii pentru a asigura scalabilitatea și mențenanța ușoară a platformei. Fiecare microserviciu va gestiona independent propriile tabele din baza de date, garantând astfel o performanță optimă și izolare în caz de erori.
- **API Gateway:** Implementarea unui API Gateway pentru redirecționarea tuturor apelurilor, simplificând gestionarea fluxului de date între frontend și backend și asigurând o securitate eficientă.

- **Funcționalitate de Căutare Avansată:** Implementarea funcționalității de căutare de produse, cu algoritmi de potrivire ce permit o toleranță pentru greșeli de tastare, îmbunătățind astfel precizia și relevanța rezultatelor căutărilor.
- **Gestionarea Erorilor:** Dezvoltarea unui sistem de gestionare a erorilor, cu propagare între microservicii prin intermediul antetelor de răspuns până în frontend. Fiecare eroare va avea un cod specific, facilitând afișarea mesajelor corecte către utilizatori.
- **Caching:** Utilizarea cache-ului pentru a îmbunătăți considerabil performanța aplicației prin stocarea temporară a datelor frecvent accesate.
- **Performanță și Testare:** Testarea riguroasă a platformei pentru a gestiona un volum mare de date, demonstrând capacitatea de a funcționa eficient cu peste 50.000 de produse adăugate.

1.4 Structura lucrării

În continuare vom analiza structura prezentării acestei lucrări după cum urmează:

- Capitolul 2 - Analiza și specificarea cerințelor: În acest capitol doresc să pun în evidență cerințele non-funcționale pe care utilizatorul și platforma trebuie să le îndeplinească, cât și cele funcționale prin care utilizatorul poate interacționa cu platforma. Vor fi, de asemenea, descrise toate fluxurile aplicației prin diagrame corespunzătoare.
- Capitolul 3 - Studiu de piață / Abordări existente: Această secțiune va cuprinde rezultatele unor studii făcute pe baza unui set de întrebări adresate potențialilor clienți legat de necesitatea și aprecierea ideii unei astfel de aplicații la momentul actual. Vor fi analizate platforme deja existente, în special punctele tari și slabe ale acestora și cum au performat până în prezent.
- Capitolul 4 - Soluția propusă: În acest capitol voi prezenta întreaga arhitectură a soluției. Capitolul va cuprinde tehnologiile folosite, motivația din spatele alegerii acestor tehnologii, performanțe comparative cu alte opțiuni valabile. De asemenea, vom aborda și soluția tehnică, prelucrarea datelor, securitatea lor și cum utilizatorul interacționează cu acestea pentru a obține un rezultat final.
- Capitolul 5 - Detalii de implementare: Acest capitol este destinat descrierii detaliate a funcționalităților, cum au fost implementate segmente specifice din fluxurile aplicației, ce metode am folosit pentru îmbunătățirea considerabilă a securității, a vitezei de răspuns pentru procesare și potențialul de scalare viitoare.
- Capitolul 6 - Studiu de caz / Evaluarea rezultatelor: În urma prezentării motivației și soluției alese, urmează să vedem rezultatele, atât de design, funcționalitate, cât și de performanță. Voi face acest lucru prin statistici, imagini relevante cu fluxurile aplicației și stress-testing prin parcurgerea acțiunilor într-un mediu cu o multitudine de produse adăugate în platformă.

- Capitolul 7 - Perspective viitoare: Acest capitol va explora direcțiile potențiale de dezvoltare și îmbunătățire a aplicației pe termen lung. Vom discuta despre noi funcționalități și modalități de extindere a pieței pentru a răspunde cerințelor utilizatorilor viitori.

2 ANALIZA ȘI SPECIFICAREA CERINȚELOR

2.1 Motivația

Un raport realizat de Ellen MacArthur Foundation arată că economia circulară ar putea genera economii anuale de 700 miliarde de euro în Europa, doar prin optimizarea utilizării resurselor [3]. Aceasta înseamnă că prin adoptarea unor modele de afaceri care favorizează reciclarea și reutilizarea, companiile pot obține avantaje competitive semnificative. De exemplu, colectarea și reutilizarea hainelor ar putea aduce economii de 71 miliarde de dolari în materiale. Astfel, SwapIt contribuie nu doar la reducerea risipei, ci și la economisirea resurselor materiale pentru utilizatori.

În același timp, piața de schimb și vânzare a obiectelor uzate este extrem de fragmentată. Studiile realizate de Accenture subliniază dificultățile întâmpinate de utilizatori în a găsi parteneri de schimb de încredere și în a se proteja împotriva fraudelor [4]. SwapIt abordează aceste probleme prin implementarea unui sistem robust de verificare a utilizatorilor și de evaluare a produselor, asigurând un mediu sigur și de încredere pentru toți utilizatorii platformei.

Pandemia COVID-19 a accelerat semnificativ digitalizarea și a schimbat comportamentele de consum. Conform unui raport al PwC, a crescut considerabil utilizarea platformelor digitale pentru schimburi fără contact direct [5]. În acest context, SwapIt oferă o soluție convenabilă și sigură pentru schimbul de bunuri, răspunzând perfect nevoilor actuale de interacțiune online și de reducere a contactului fizic.

2.2 Cerințe non-funcționale

Cerințele non-funcționale sunt esențiale pentru asigurarea unei experiențe de utilizare optimă și a funcționării eficiente a platformei SwapIt. Aceste cerințe se referă la aspectele generale ale sistemului, care nu sunt direct legate de funcționalitățile de business, dar sunt cruciale pentru performanța, securitatea și fiabilitatea platformei.

2.2.1 Cerințe tehnice pentru utilizatori

- **Dispozitive compatibile:** Utilizatorii trebuie să aibă un dispozitiv care suportă un browser web modern, cum ar fi un computer, laptop, tabletă sau smartphone.
- **Conexiune la internet:** Este necesară o conexiune stabilă la internet pentru a accesa platforma. O viteză mare a internetului este recomandată pentru a asigura o navigare fluidă și rapidă.
- **Securitate:** Este recomandat ca utilizatorii să aibă un software antivirus actualizat și un firewall activ pentru a proteja datele personale și a preveni accesul neautorizat.

- **Browser web:** Utilizatorii trebuie să utilizeze un browser web actualizat (de exemplu, Google Chrome, Mozilla Firefox, Brave, Safari sau Microsoft Edge) pentru a accesa platforma SwapIt.

2.2.2 Confidențialitate și Securitate

- **Autentificare și autorizare:** Platforma trebuie să utilizeze mecanisme sigure de autentificare și autorizare pentru a se asigura că numai utilizatorii autorizați au acces la anumite funcționalități și date.
- **Criptarea datelor:** Datele sensibile transmise între client și server trebuie criptate folosind protocoale securizate (SSL/TLS) pentru a preveni interceptarea acestora.

2.2.3 Performanță și viteză

- **Timp de încărcare a paginilor:** Pagina principală și paginile de produs trebuie să se încarce în mai puțin de o secundă pentru a menține interesul utilizatorilor și a evita abandonarea navigării.
- **Timp de răspuns:** Platforma SwapIt trebuie să ofere un timp de răspuns rapid, ideal sub o secundă pentru 95% dintre cereri, pentru a asigura o experiență fluidă și plăcută pentru utilizatori.

2.2.4 Scalabilitate

- **Extinderea utilizatorilor:** Platforma trebuie să fie capabilă să gestioneze creșterea numărului de utilizatori și produse fără a afecta negativ performanța. Arhitectura de microservicii aleasă va facilita scalabilitatea orizontală.
- **Adăugarea de funcționalități:** Platforma trebuie să permită adăugarea de noi funcționalități și servicii fără a necesita modificări majore în arhitectura existentă, asigurând astfel flexibilitate și extensibilitate.

2.2.5 Mantenabilitate

- **Facilitatea întreținerii:** Sistemul trebuie să fie ușor de întreținut, cu un cod bine documentat și modular, pentru a permite actualizări și schimbări de business rapide.

2.3 Cerințe funcționale

Cerințele funcționale țin de acțiunile pe care utilizatorii le pot realiza în cadrul platformei. Aplicația SwapIt dispune de 3 tipuri de utilizatori: utilizatorul anonim, utilizatorul autentificat și administratorul. Funcționalitățile sunt împărțite în mai multe categorii în funcție de tipul de utilizator ce folosește aplicația:

Funcționalități globale (prezente pentru toate tipurile de utilizatori):

- Vizualizare produse recomandate
- Vizualizare produs individual
- Căutarea produselor după cuvinte cheie
- Filtrarea produselor în funcție de categorie / subcategorie

- Utilizarea chatbot-ului pentru contactul cu echipa de suport în timp real
- Vizualizarea profilurilor utilizatorilor înregistrați pe platformă
- Filtrarea produselor după criterii precum popularitate, cele mai noi adăugate sau ordine random
- Vizualizarea produselor adăugate de alții utilizatori în platformă

Funcționalități specifice utilizatorilor anonimi:

- Autentificare pe baza username-ului și parolei
- Înregistrare pe baza email-ului (prin un scenariu ce include cod de securitate) și a parolei
- Autentificare / Înregistrare cu Oauth2 prin Google
- Recuperare cont pe baza email-ului în cazul în care utilizatorul a uitat parola contului

Funcționalități specifice utilizatorilor autentificați (nu include utilizatorul administrator):

- Adăugarea unui nou produs în platformă
- Modificarea specificațiilor unui produs
- Ștergerea unui produs pe care aceștia l-au creat
- Adăugarea produselor la lista de favorite
- Vizualizarea produselor favorite din pagina de profil
- Actualizarea datelor personale de bază (Nume, Adresă, număr de telefon, etc)
- Schimbarea fotografiei de profil
- Actualizarea datelor importante (username, email, parolă)
- Deconectarea de pe platformă

Funcționalități specifice administratorilor:

- Ștergerea unui produs neadecvat din platformă
- Restrictionarea unui utilizator temporar sau permanent + Ridicarea acestor restricții
- Vizualizarea auditului compus din acțiunile relevante ce au loc în cadrul aplicației (înregistrarea unui utilizator, adăugarea unui nou produs, restrictionarea sau scoaterea restricțiilor unui utilizator realizată de către administratori)
- Actualizarea datelor personale de bază (Nume, Adresă, număr de telefon, etc)
- Schimbarea fotografiei de profil
- Actualizarea datelor importante (username, email, parolă)
- Deconectarea de pe platformă

2.4 Diagrama use-case a aplicației

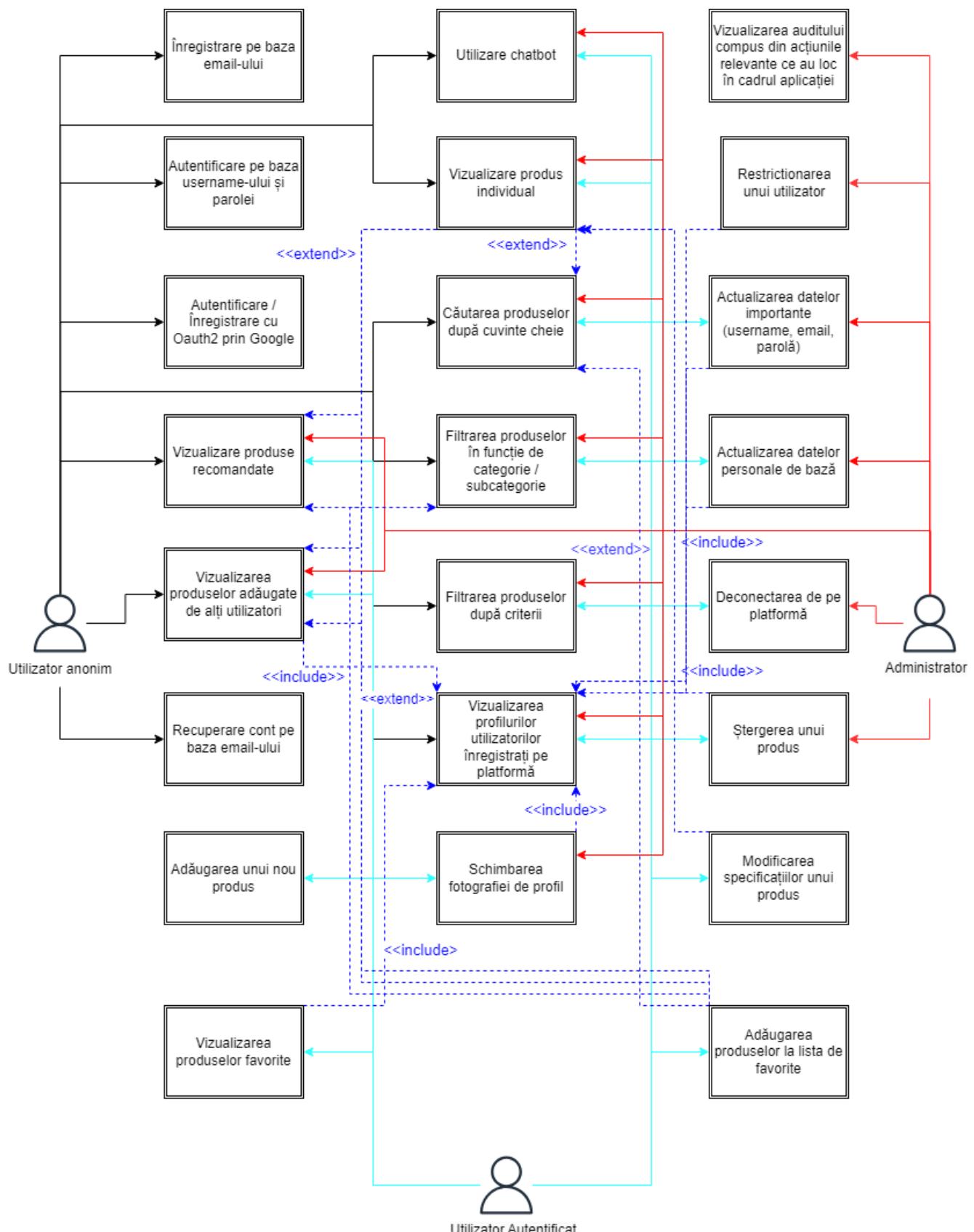


Figura 2: Diagrama use-case pentru utilizatorii anonimi, autentificati si administratori

3 STUDIU DE PIAȚĂ / ABORDĂRI EXISTENTE

Pentru ca o aplicație precum SwapIt să aibă succes în piață este esențial să înțelegem ce soluții similare sunt deja prezente și cum ne putem diferenția. Acest capitol analizează platformele de schimb de obiecte existente, evidențiind punctele lor forte și slabe. Vom discuta problemele întâmpinate de utilizatori și vom identifica oportunitățile de îmbunătățire pentru platformă, asigurând astfel că avem potențial de a oferi o experiență superioară și de a răspunde mai bine nevoilor utilizatorilor.

Vom începe prin a observa principalele statistici din domeniul e-commerce. Conform unui studiu realizat de TechJury în ianuarie 2024 [7], am extras câteva date importante. De exemplu, 53% din potențialii clienți părăsesc website-ul după un timp de încărcare mai mare de 3 secunde. De asemenea, prima opțiune după o experiență dezamăgitoare este să se folosească de alternativele oferite de competiție. Astfel, putem observa cât de importantă este asigurarea că platforma se ridică la standardele profilului utilizatorului comun.

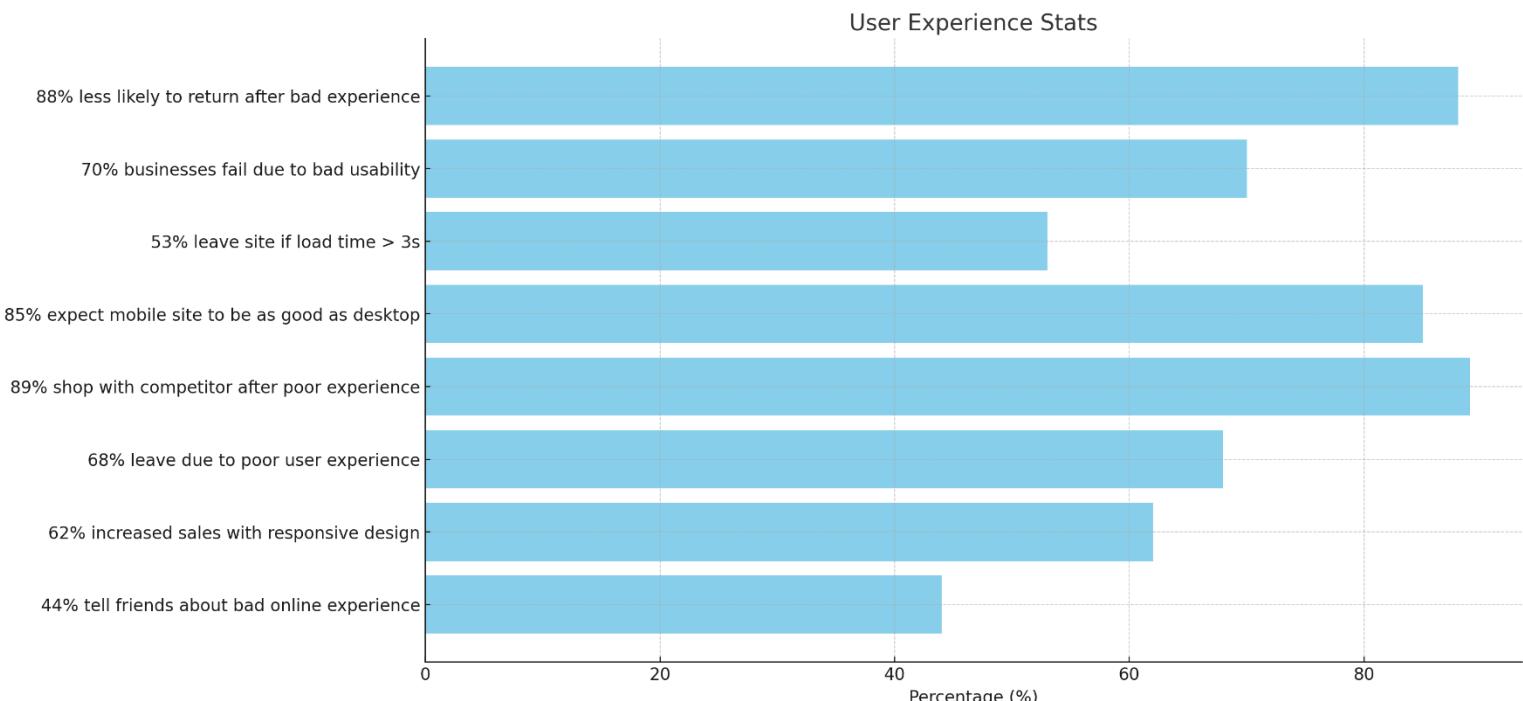


Figura 3: Statistici legate de experiența utilizatorilor pe un site web [7]

Prin aplicația SwapIt îmi propun să dezvolt o platformă care să satisfacă în primul rând dorințele clientului, să ofere o performanță optimă (sub 0,5 secunde pentru fiecare flux de business) și să fie intuitivă și ușor de utilizat. În continuare, am încercat să înțeleg publicul cu potențial de a deveni client și am realizat, în acest sens, un sondaj de opinie referitor la experiențele trecute și potențiale ale utilizatorilor.

3.1 Sondaj De Opinie

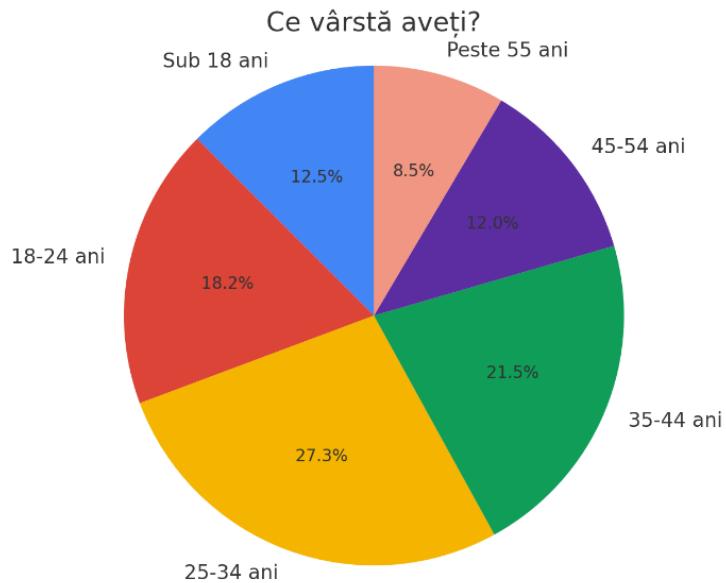


Figura 4: Intervalele de vârstă ale potențialilor clienți

Cât de frecvent constatați că aveți obiecte pe care nu le mai folosiți?

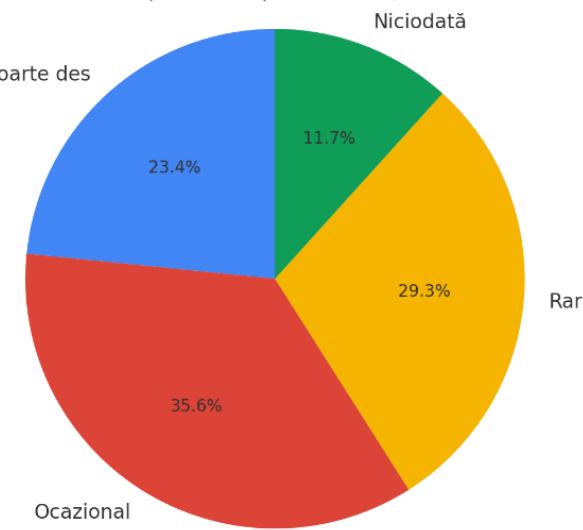


Figura 5: Identificarea problemei pe care o va rezolva platforma SwapIt

Analizând datele din Figura 5, se observă că problema obiectelor nefolosite este semnificativă. Majoritatea respondenților întâlnesc ocazional (35.6%) sau foarte des (23.4%) obiecte pe care nu le mai folosesc. 29.3% le găsesc rar, iar doar 11.7% nu se confruntă cu această situație.

Grupa de vârstă predominantă, 25-34 de ani, este în perioada de maximă cheltuială și achiziție de bunuri. Acest lucru explică frecvența ridicată a obiectelor nefolosite, subliniind necesitatea soluțiilor de reducere a risipei.

Cum evaluați ideea unui website pentru schimbul de produse ca un supliment la platformele tradiționale de vânzare-cumpărare?

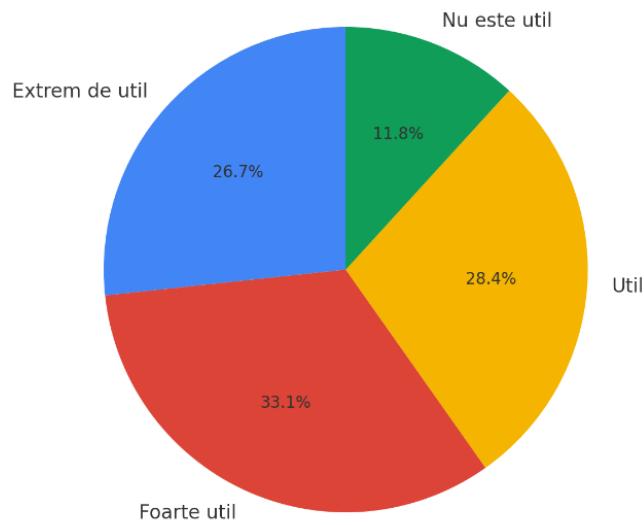


Figura 6: Părerea clienților despre o platformă ce facilitează schimbul de obiecte

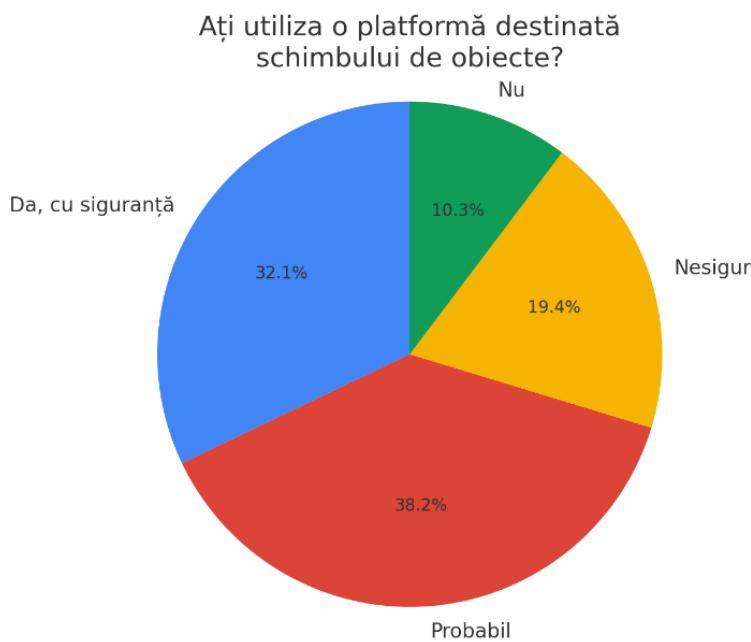


Figura 7: Dorința individului de a utiliza o astfel de platformă

Deloc surprinzător, un procent mare de persoane este încântat de prezența unui astfel de website în piață. Înțelegem astfel că problema nu este lipsa de dorință, ci de oportunitate în legătură cu un astfel de proces de schimb de obiecte.

Ce aspect considerați cel mai important pentru un website de schimb de obiecte?

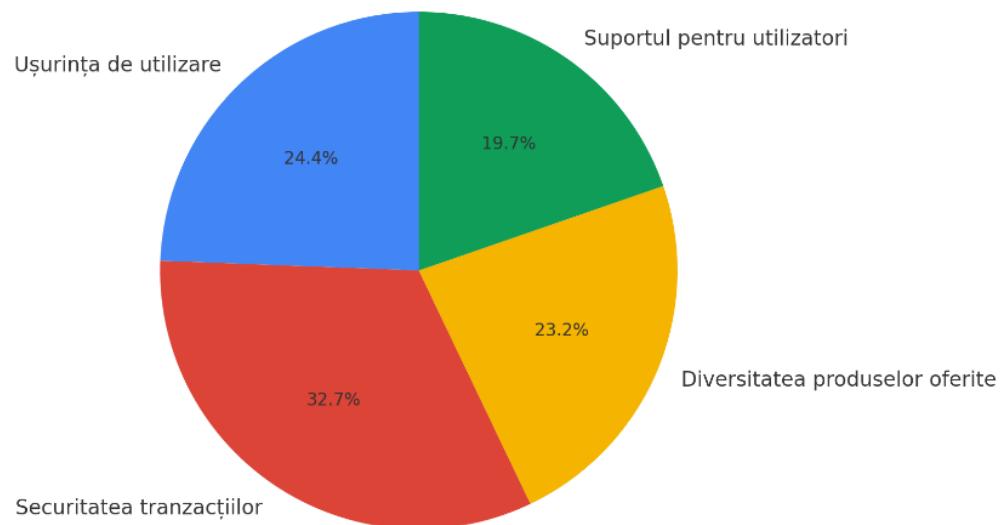


Figura 8: Așteptările clientului referitor la experiența pe site

Ce considerați a fi cel mai mare obstacol în utilizarea unui website de schimb de obiecte?

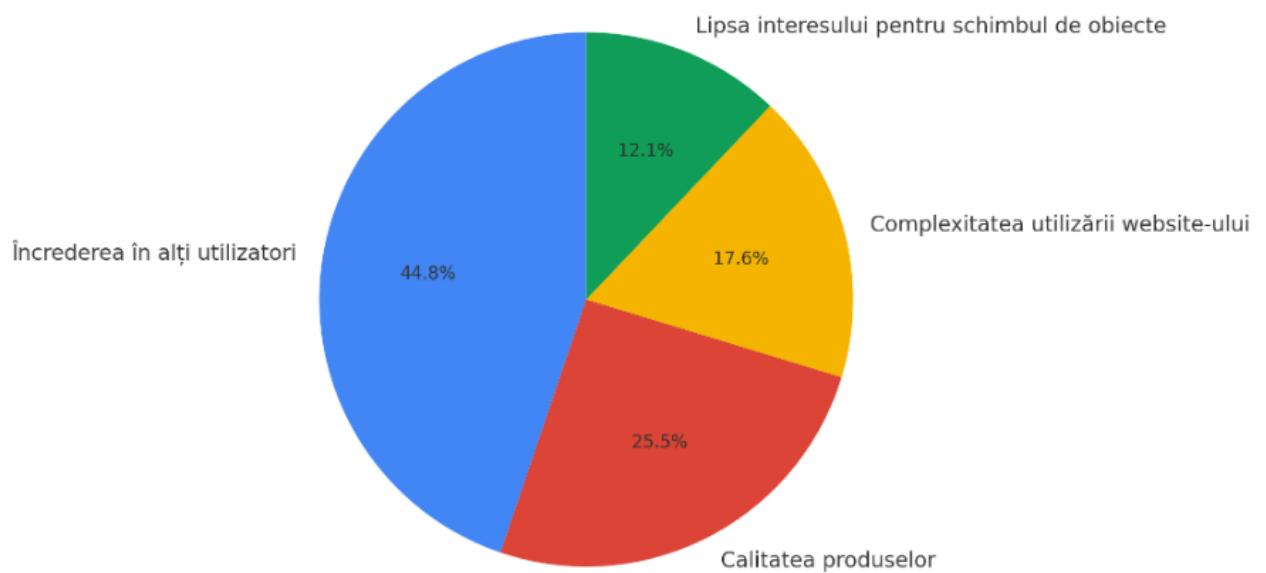


Figura 9: Obstacole în utilizarea unei astfel de platforme

Putem vedea în Figura 8 echilibrul procentual între aşteptările utilizatorului cu privire la o platformă precum SwapIt. Clienții doresc atât diversitate de produse și ușurință de utilizare a site-ului, cât și securitate și suport pentru a se putea încrede într-un astfel de sistem de tranzacții. Dorința de securitate este întărită de rezultatul din Figura 9, unde observăm că pentru aproape jumătate din potențialii clienți, încrederea în ceilalți utilizatori este cea mai mare problemă. Din acest motiv, SwapIt își propune să ofere o transparentă cât mai vizibilă în legătură cu profilurile celor înregistrați, dar și un control sporit asupra comportamentului în cadrul platformei.

3.2 Analiză Schimburi.ro

The screenshot shows the Schimburi.ro website homepage. At the top, there is a navigation bar with links for 'Prima pagina', 'Cautare', 'Adaugare anunt', 'Indicatii', and 'Contact'. Below the navigation bar, there are links for 'Autentificare', 'Cont nou', 'Filozofia', and 'Intrebări frecvente'. On the right side of the header, it says '8 Iunie 2024' and has links for 'Schimburi auto', 'Schimburi imobiliare', and 'Schimburi telefoane'. The main content area features a search bar with placeholder text 'Bine ai venit! Daca esti nou pe schimburi.ro, iti recomandam sa citesti indicatiile! [detaliu]'. Below the search bar, there is a section titled 'Cele mai noi anunțuri' showing three recent ads:

Titlu anunț	Județ	Pret	Grad SCH	Vrea în schimb
Bautura acidulata Coca Cola Vanilla...	MURES	1 EURO	Ridicat	Alta categorie
Bautura Coca Cola Cherry import...	BUCURESTI	1 EURO	Ridicat	Auto - Moto - Velo
Bobinaj magnetouri motocultoare	SATU MARE	447.206 RON	Ridicat	Auto - Moto - Velo

On the left side, there is a sidebar with 'Căutare' and 'Cuvinte-cheie' fields, a 'Categorii' section listing various product categories, and a 'Reclamă' section. On the right side, there are sections for 'Anunțuri recente' (with a list of 24, 7, 14, and 21 days old ads), 'Statistică' (with user statistics and an 'Attractive' ad for Coca-Cola), and 'Reclamă' (with ads for Transport auto, Ai un eveniment?, and Contul Meu).

Figura 10: Captură de ecran a paginii acasă

Site-ul <https://www.schimburi.ro/> este o platformă românească de schimburi de obiecte ce urmărește același scop pe care SwapIt îl are. Observăm că interfața utilizatorului este destul de învechită, suntem întâmpinați de un design demodat și o multitudine de informații. Consider că date precum numărul de utilizatori activi și data din calendar sunt informații irelevante, în special pentru un astfel de site. De asemenea, vedem o repetiție a funcționalităților (secțiune specială de căutare dar și bară de căutare în stânga sus, autentificare și cont nou, etc) care reușesc în mare parte să deruteze clientul mai mult decât să îl ajute.

Nr. 65579

Bobinaj magnetouri motocultoare

[schimburi.ro](#) » [Auto - Moto - Velo](#) » [Utilitare](#)

[Inainte](#)

Descriere
Bobinez la preturi ieftine si cu garantie magnetouri pentru motocultoare de orice tip

Detalii comerciale

Pret:	447.206 RON
Accepta schimburile:	DA. Asteapta oferte.
Vrea ceva din categoria:	Auto - Moto - Velo
Grad SCH:	Ridicat

Date de contact

ofrant:	mirceafechet
Telefon:	0770.742.522
Oras:	Mofntiu Mare
E-mail:	fecheteilonka8@gmail.com

Detalii anunt

Nr. anunt:	65579
Data publicarii:	25.05.2024

contacteaza

alte anunturi

recomanda

salveaza

tiparescă

Figura 11: Captură de ecran a paginii de produs

De asemenea, pagina de produs este aglomerată, iar imaginile, care ar trebui să ocupe cea mai mare parte a paginii, sunt extrem de mici. De asemenea, se pot observa specificații a căror semnificație nu este clară, precum "Grad SCH".

Cauta

In categoria: Toate categoriile si in subcategori
cauta in titluri si in descriere doar cu poza

Cuvinte-cheie:

Nr. anunt:

Sortare după: Data

In ordine: Descrescătoare

Accepta schimburile: da nu indiferent

Grad SCH:

Vrea in schimb:

Pret: intre si

Valuta:

Judet:

Cauta > **Cautare nouă**

Figura 12: Captură de ecran a paginii de căutare produse

Pagina de căutare este, de asemenea, dificil de utilizat și solicită un volum considerabil de informații din partea utilizatorului. În general, clienții tind să abandoneze astfel de pagini care, pentru o funcționalitate de bază, necesită un grad ridicat de implicare. De asemenea, se ridică întrebarea cu privire la diferența dintre opțiunile "căutare" și "căutare nouă" oferite în partea de jos a paginii.

3.3 Analiză Listia.com



Figura 13: Captură de ecran a erorii de conectare în platformă

Surprinzător, în încercarea de a accesa platforma <https://www.listia.com/>, am întâmpinat un mesaj de eroare. Acest incident sugerează în mod clar că site-ul nu este scalat corespunzător pentru a susține numărul de utilizatori pe care îl înregistrează într-o zi obișnuită.

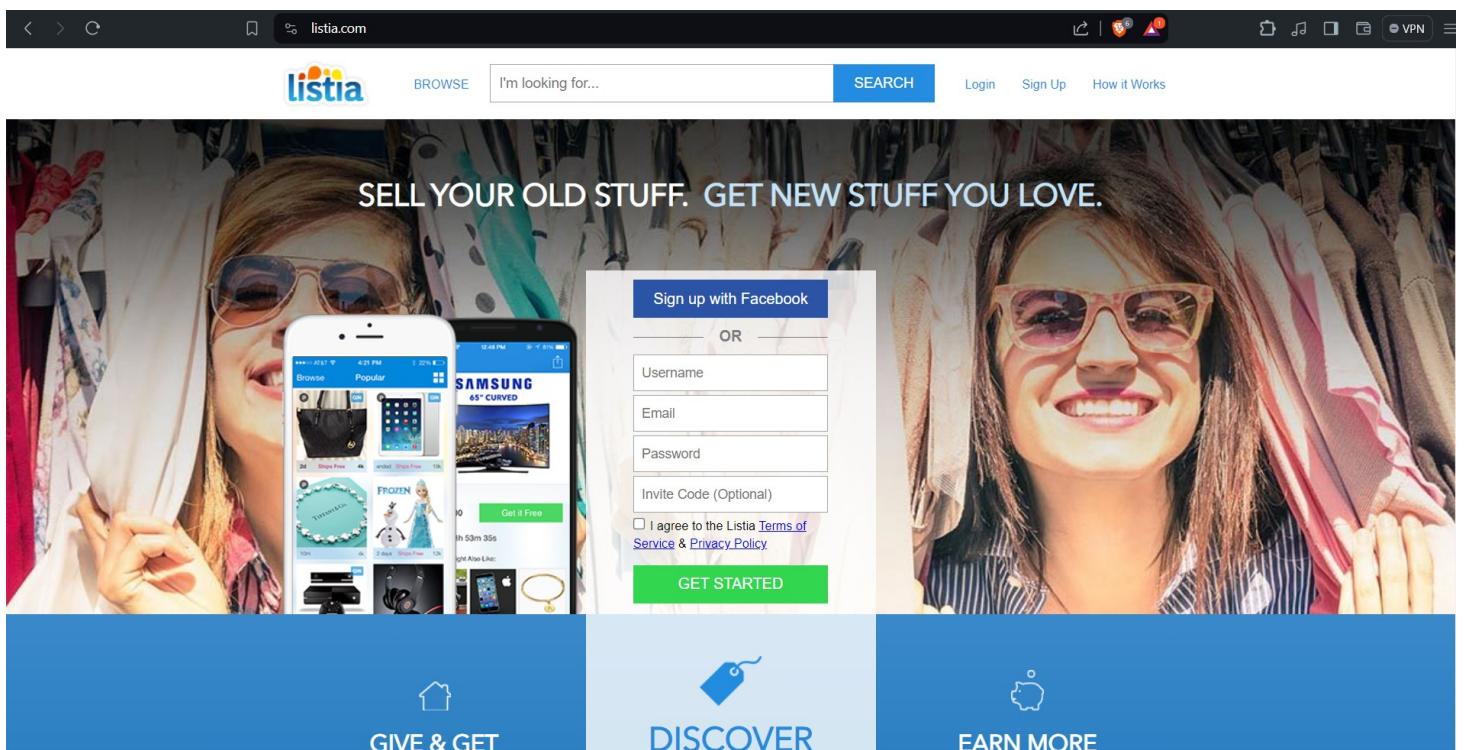


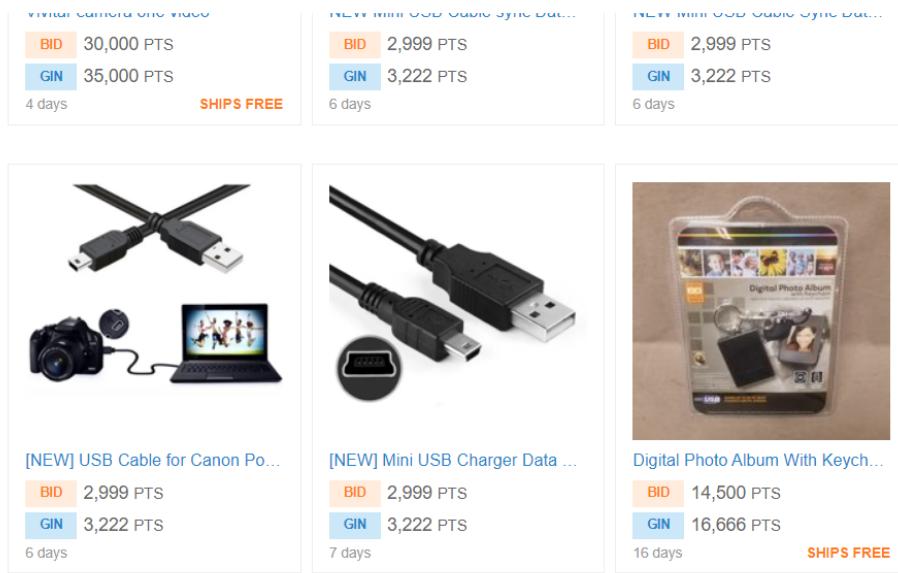
Figura 14: Captură de ecran a paginii de acasă

După câteva minute de așteptare, am reușit să accesez site-ul. Designul acestuia este considerabil mai organizat comparativ cu cel analizat anterior, deși persistă impresia unei platforme ușor învechite. Consider că, pentru a spori retenția utilizatorilor, pagina principală ar trebui să prezinte o serie de produse relevante, în locul paginii de înregistrare.

The screenshot shows a product listing for a "Pankoo 40 x 60 high power monocular telescope with smartphone holder". The main image displays the telescope standing upright. To the right, there's a "Get Started" button and a sidebar with user statistics and a video titled "How Listia Works". Below the main image, there are smaller thumbnail images and a "View larger" link. The right side of the page features a sidebar with a user profile for "bigtoon1", a "Related things you might like" section with thumbnails for "Halo Selfie light" and "Pankoo 40 x 60 high power monocu...", and a "How Listia Works" video.

Figura 15: Captură de ecran a paginii de produs

Pagina de produs este considerabil îmbunătățită; cu toate acestea, rămâne dificil de înțeles modalitatea exactă prin care se poate efectua schimbul de produse, precum și identitatea persoanei care a postat anunțul. În plus, există butoane irelevante pentru distribuirea acestui produs pe rețelele sociale (Pinterest și X).



← Previous 1 2 Next →

Results per page: 30 60 90

Figura 16: Captură de ecran a organizării produselor pe pagini

Posibilitatea de a naviga către pagina următoare de produse se află doar în partea de jos a listei, ceea ce reprezintă un inconvenient suplimentar pentru utilizatori.

3.4 Analiză Ilovefreegle.org

Freegle - like online dating for stuff.

Got stuff you don't need? Looking for something?

We'll match you with someone local. All completely free.

[Give Stuff](#) [Ask for Stuff](#)

Don't throw it away, give it away!

Just looking?

See what's being freegled near you:

[GET IT ON Google Play](#) [Download on the App Store](#)

Figura 17: Captură de ecran a paginii de acasă

Website-ul <https://www.ilovefreegle.org/> este un alt exemplu de platformă din același domeniu. Designul acestuia este puțin straniu, în special caruselul de imagini din interiorul ramei de tablou. În încercarea de a naviga pe site, am observat mai multe aspecte ce ar putea fi îmbunătățite:



Figura 18: Captură de ecran a subsolului platformei

În primul rând, subsolul platformei constă într-un singur rând de butoane de dimensiuni reduse, dificil de observat. Unele dintre aceste secțiuni (About, Contact, Donate) ar fi fost mult mai eficiente dacă ar fi fost plasate în antetul site-ului.

Figura 19: Captură de ecran a căutării de produs.

Asemănător ca la platforma schimburi.ro, este nevoie de multe informații pentru a căuta un produs și, de asemenea, nu există o modalitate prin care putem vedea produse recomandate, fără a ști exact ce căutăm.

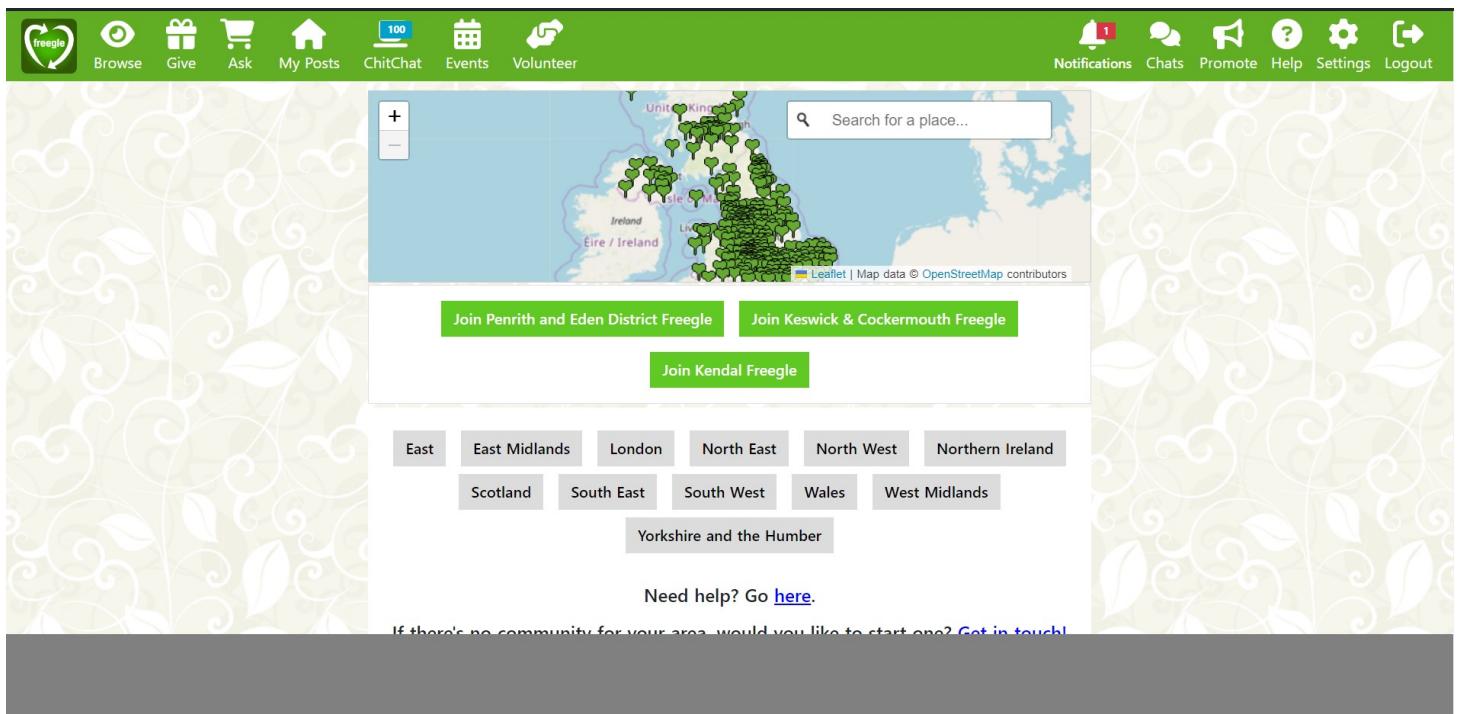


Figura 20: Captură de ecran a paginii de acasă după înregistrare

Am considerat că ar fi util să mă și înregistrez pentru a vedea mai multe funcționalități pe care platforma le oferă și în continuare nu avem produse recomandate iar butoanele sunt de data aceasta prea mari. Pentru a putea totuși vedea produsele a trebuit să înaintez prin a îmi selecta o comunitate din care fac parte. Un alt lucru deranjant este bara de culoare gri din partea de jos a site-ului care ocupă cam 15% din înălțime.

3.5 Concluzii

Platforma SwapIt promite să adreseze eficient deficiențele evidențiate în cadrul analizelor comparative ale platformelor existente. Printr-un design modern și intuitiv, SwapIt va elimina navigarea complicată și elementele de design învechite, întâlnite frecvent pe site-urile analizate. Platforma va pune un accent deosebit pe securitate și transparența utilizatorilor, sporind astfel încrederea și siguranța necesare pentru a facilita schimbul de obiecte.

În plus, SwapIt va oferi o experiență de utilizare îmbunătățită prin funcționalități simplificate și o varietate de produse accesibile cu ușurință. Aceste măsuri vor transforma platforma într-o soluție eficientă și accesibilă, răspunzând nevoilor utilizatorilor și promovând economia circulară într-un mod practic și sigur. Astfel, SwapIt se poziționează ca o alternativă superioară, capabilă să adreseze problemele actuale și să ofere o soluție viabilă pentru schimbul de obiecte.

4 SOLUȚIA PROPUȘĂ

4.1 Tehnologii utilizate

4.1.1 React.js

Pentru dezvoltarea interfeței aplicației am ales să utilizez React.js [11], aceasta fiind una dintre cele mai populare biblioteci JavaScript. React oferă dezvoltatorilor posibilitatea de a crea componente reutilizabile, fapt ce contribuie la o structură modulară și ușor de întreținut pe termen lung. Principalele avantaje ale utilizării React sunt următoarele:

- Performanță ridicată: React utilizează un Document Object Model (DOM) virtual, ceea ce îmbunătățește considerabil actualizările obiectelor. Acest mecanism de actualizare eficientă reduce semnificativ timpul de reîncărcare a paginilor și îmbunătățește experiența utilizatorului
- Control asupra stărilor: React dispune de un control complet asupra schimbărilor de stare cu ajutorul funcțiilor precum `useEffect` (cod executat de fiecare dată la schimbarea cel puțin a uneia din parametrii date) și `useState` (funcție de asignare de valori și schimbare dinamică a datelor din interiorul componentei).
- Flux de date unidirecțional: Fiind unidirecțional, fluxul de date în React permite o mai bună gestionare a stării aplicației și a stărilor următoare, facilitând astfel întreținerea și depanarea codului
- Comunitate excelentă: De-a lungul anilor, React a fost integrat într-o multitudine de proiecte, beneficiind astfel de o comunitate vastă și activă. Această comunitate oferă suport continuu, soluții la probleme comune și o bogată bază de resurse și tutoriale, ceea ce contribuie la o curba de învățare mai lină și la o dezvoltare mai eficientă a aplicațiilor
- Moștenirea metodelor și a datelor: Dacă într-o componentă folosim o componentă copil, aceasta poate primi ca parametru metode sau date din componenta părinte, lucru ce oferă foarte multă flexibilitate

```
br >  UserDetails.jsx > ...
import React from 'react';

const UserDetails = ({userData, isTemporaryBanned, isPermanentBanned, banExpiryDate, isUserProfileAuth}) => {
  return (
    <div className="user-details">
      {isTemporaryBanned || isPermanentBanned} && (
        <div className="d-flex align-items-center br-10 pl-4 pt-4" style={{ color: 'red' }}>
          <div className="d-flex br-10 align-items-center justify-content-center flex-shrink-0 ml-n4" style={{ width: '50px', height: '50px' }}>
            <i className="fa-solid fa-lg fa-triangle-exclamation" />
          </div>
        {isTemporaryBanned && (
          <div className="m-0"> User is banned till {banExpiryDate} </div>
        )}
        {isPermanentBanned && (
          <div className="m-0"> User is permanently banned </div>
        )}
      </div>
    )
    <div className="d-flex align-items-center br-10 pl-4 pt-4">
      <div className="d-flex br-10 align-items-center justify-content-center flex-shrink-0 ml-n4" style={{ width: '50px', height: '50px' }}>
        <i className="fa fa-lg fa-user" />
      </div>
      <div className="text-light m-0">{userData.name} {userData.surname}</div>
    </div>
  )
}
```

Figura 21: Exemplu secvență de cod scrisă în React.js

4.1.2 Firebase

Firebase[12] este o platformă dezvoltată de Google, care oferă o gamă largă de funcționalități, inclusiv stocarea datelor, autentificare, hosting pentru site-uri web, notificări push și altele. Am ales această platformă datorită scalabilității, fiabilității și ușurinței de integrare în aplicațiile web moderne. În cadrul aplicației am ales să utilizez Firebase pentru stocarea datelor publice, în special a fotografiilor produselor și ale utilizatorilor.

Interacțiunea platformei SwapIt cu Firebase se realizează exclusiv în partea de frontend, conform următorului flux operațional:

- Utilizatorul încarcă fie o fotografie de profil, fie o imagine a unui produs.
- Imaginea este preluată și i se generează un identificator unic universal (UUID) în frontend, după care este trimisă către Firebase.
- Ulterior, se efectuează o cerere către Firebase pentru a obține URL-ul imaginii recent create. Acest URL este apoi stocat în baza de date de către componenta de backend.
- De fiecare dată când se dorește afișarea unei imagini, URL-ul corespunzător este preluat din backend și redat ca imagine în frontend.

Acest proces asigură o gestionare eficientă și securizată a resurselor media utilizate în cadrul platformei SwapIt. Prin utilizarea Firebase, beneficiază de avantajele unei platforme gestionate de un lider în tehnologie, care asigură disponibilitate ridicată și performanță optimă, esențiale pentru buna funcționare a aplicației.

4.1.3 Spring Boot

Am ales să folosesc Spring Boot[13] pentru dezvoltarea componentei de backend a aplicației. Spring Boot este, în prezent, cel mai popular framework pentru Java, oferind o multitudine de integrări cu diverse tehnologii și librării. Principalele motive pentru care am beneficiat de avantajele Spring Boot în acest proiect sunt următoarele:

- Ușurința creării controllerelor REST funcționale pentru apelurile HTTP/S, cu ajutorul anotărilor precum „@GetMapping, @RestController, @PostMapping”
- Prezența multor opțiuni pentru configurările de securitate
- Facilitatea integrării rapidă a tuturor nevoilor de dezvoltare zilnice, reducând considerabil timpul necesar pentru configurare și implementare
- Procesarea asincronă a datelor, ceea ce îmbunătățește semnificativ performanța
- Permiterea configurațiilor extensive
- Injectarea de dependențe făcută automat pentru clasele annotate cu „@Service, @Component, @Repository, etc”, astfel eficientizând memoria folosită prin design pattern-ul Singleton
- Ușurința de creare și folosire a obiectelor cu ajutorul anotărilor „@Builder, @Data, etc”
- Posibilitatea de a folosi Hibernate, care este un ORM (Object–relational mapping) ce face posibilă interacțiunea într-un mod ușor cu baza de date

4.1.4 PostgreSQL

Pentru gestionarea bazei de date a aplicației SwapIt, am ales PostgreSQL[14] datorită performanței, fiabilității și multitudinii de caracteristici avansate. PostgreSQL este un sistem de gestionare a bazelor de date relaționale (RDBMS), iar principalele avantaje ale utilizării acestuia includ:

- Performanță și scalabilitate: Este proiectat pentru a gestiona volume mari de date și pentru a suporta multiple operațiuni simultane fără a compromite performanța
- Suport SQL avansat: Oferă funcționalități avansate care permit realizarea de interogări complexe și gestionarea eficientă a tranzacțiilor.
- Extensibilitate: Utilizatorii pot adăuga tipuri de date noi și funcții personalizate, adaptând baza de date la nevoile specifice ale aplicației
- Securitate: Oferă autentificare și control detaliat al accesului bazat pe roluri, asigurând protecția datelor private

4.2 Arhitectura aplicației

4.2.1 Microservicii

În cadrul aplicației am ales să dezvolt o arhitectură bazată pe microservicii, arhitectură ce favorizează scalarea pe orizontală și menținerea funcționalităților majore de business independente. Această abordare prezintă multiple avantaje, după cum urmează:

- Fiecare microserviciu poate fi scalat independent, în funcție de volumul de cereri ale utilizatorilor
- Scalarea independentă permite utilizarea eficientă a resurselor, evitând costurile suplimentare asociate cu scalarea întregii aplicații
- Echipele pot dezvolta, testa și livra microservicii în mod independent, ceea ce reduce timpul de lansare pe piață și permite un proces iterativ mai rapid
- Fiecare microserviciu poate fi dezvoltat folosind tehnologia cea mai potrivită pentru funcționalitatea respectivă
- Microserviciile pot fi actualizate individual fără a necesita repornirea întregii aplicații, reducând timpul de nefuncționare
- Izolarea erorilor: O problemă într-un microserviciu nu afectează întreaga aplicație, ceea ce îmbunătățește stabilitatea generală și ușurează procesul de localizare a erorilor.

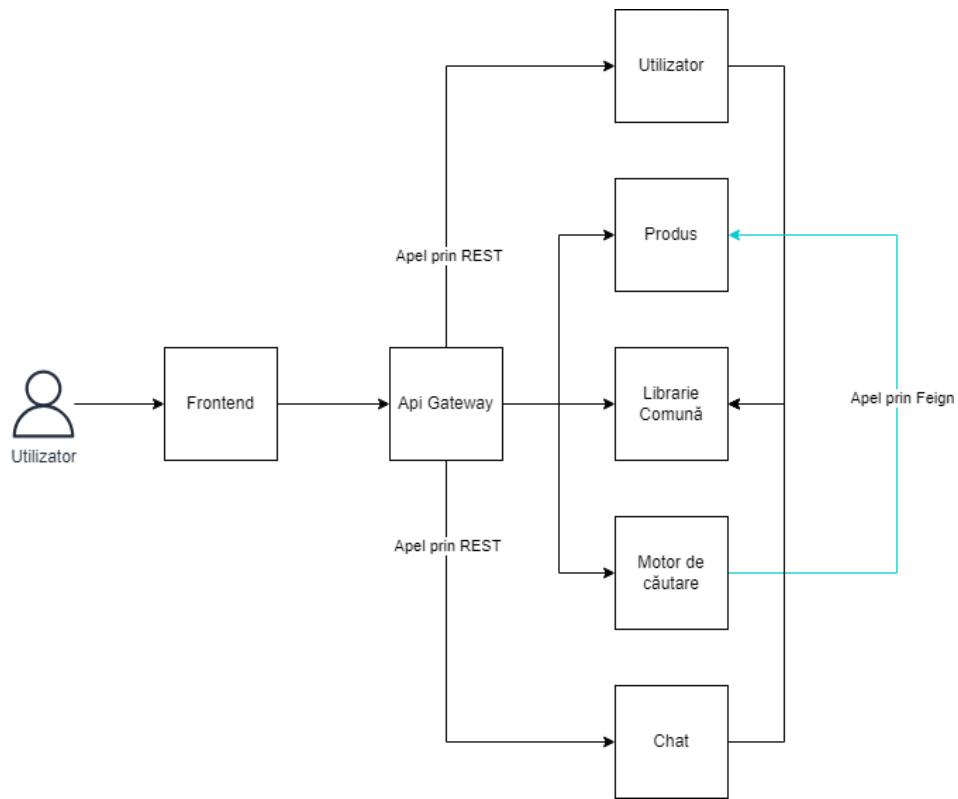


Figura 22: Diagrama arhitecturii bazată pe microservicii

Platforma SwapIt este alcătuită din 5 microservicii principale și o librerie comună, fiecare având un rol specific după cum urmează:

4.2.1.1 *Api gateway*

Acest microserviciu are rolul de a redirecționa apelurile făcute de Frontend către microserviciile corespunzătoare. O astfel de abordare este benefică, în primul rând, din punct de vedere al securității componentei de backend, deoarece Frontend-ul va schimba date doar prin intermediul acestui punct comun.

API Gateway-ul este lipsit de logică de business și este conceput să realizeze întotdeauna apeluri REST către celelalte microservicii. Această abordare facilitează o îmbunătățire viitoare a arhitecturii prin mutarea celorlalte microservicii într-o zonă separată de rețea, accesul făcându-se doar pe baza unui certificat validat din API Gateway.

De asemenea, API Gateway-ul permite centralizarea autentificării și autorizării, asigurându-se că toate cererile către microservicii sunt autentificate și autorizate corect. Aceasta simplifică gestionarea securității și reduce riscul de vulnerabilități, deoarece toate verificările de securitate sunt centralizate într-un singur loc.

Diagrama UML se poate găsi la *Anexa A1*

4.2.1.2 Utilizator

Microserviciul de utilizator gestionează toate funcționalitățile legate de utilizatori în cadrul sistemului, inclusiv înregistrarea, autentificarea, gestionarea profilului, schimbarea informațiilor private și altele. Soluția propusă utilizează două metode de autentificare și înregistrare: autentificarea prin token-uri JWT și OAuth2 prin Google, scopul fiind de a oferi o încredere mai mare utilizatorilor și a crește numărul de clienți înregistrați. Modificările critice ale informațiilor, cum ar fi parola și adresa de email, sunt securizate prin un cod de verificare trimis pe emailul utilizatorului.

De asemenea, microserviciul include funcții programate (cron job-uri) care resetează periodic aceste coduri de securitate, oferind astfel o protecție suplimentară conturilor utilizatorilor. Administratorii au posibilitatea de a gestiona utilizatorii, inclusiv de a restricționa accesul acestora în caz de comportament inadecvat. Mai mult, microserviciul oferă vizibilitate asupra acțiunilor realizate pe site, permitând monitorizarea și auditarea activităților utilizatorilor pentru a detecta și preveni orice activități suspecte sau neautorizate. Această abordare integrată asigură o securitate crescută și o gestionare eficientă a utilizatorilor în cadrul platformei.

Diagrama UML se poate găsi la *Anexa A2*

4.2.1.3 Produs

Microserviciul destinat gestionării produselor administrează, în mod direct sau indirect, toate interacțiunile utilizatorilor cu produsele. Aceasta include funcționalități esențiale precum adăugarea și actualizarea produselor, returnarea produselor recomandate sau favorite ale unui utilizator. O altă funcționalitate importantă este implementarea paginării pentru extragerea datelor în blocuri, care, împreună cu extragerea previzualizărilor produselor, reușește să îmbunătățească considerabil performanța căutărilor și recomandărilor de produse. Produsele sunt organizate pe categorii și subcategorii, oferind astfel posibilitatea de a căuta produse ce aparțin unui set specific de subcategorii.

Microserviciul de produs este în strânsă relație cu cel de căutare deoarece microserviciul de căutare al produselor nu are acces direct la tabela de produse, ci își realizează cererile prin intermediul microserviciului produs.

În concluzie, microserviciul produs îmbunătățește semnificativ experiența utilizatorului prin optimizarea interacțiunilor acestuia cu produsele, asigurând atât o performanță ridicată în procesul de căutare și recomandare, cât și o administrare eficientă a informațiilor despre produse. Această abordare modulară facilitează adaptabilitatea și scalabilitatea sistemului, contribuind astfel la o gestionare mai eficientă și flexibilă a portofoliului de produse.

Diagrama UML se poate găsi la *Anexa A3*

4.2.1.4 Motor de căutare

Microserviciul de căutare joacă un rol esențial, fiind cel mai complex în ceea ce privește logica de business. Aceasta gestionează adăugarea produselor într-un dicționar indexat pentru căutare pe baza anumitor câmpuri (ID-ul produsului și Metadata). ID-ul produsului este utilizat pentru a identifica în mod unic produsele, iar metadatele includ o serie de informații relevante stocate în dicționar pentru a facilita numărul de hit-uri pe respectivele produse. În metadate sunt incluse titlul și descrierea produsului. Implementarea utilizează paginarea rezultatelor, realizată într-un mod distinct cu ajutorul bibliotecii Lucene, proces detaliat ulterior în capitolul dedicat implementării.

Căutarea unui produs este tolerantă la greșeli de scriere, case insensitive, și permite utilizarea mai multor cuvinte, separate prin virgulă, punct și alte semne de punctuație. Pentru a evita rezultate irelevante am adăugat un analizator care exclude anumite cuvinte de legătură, precum "de", "pe", "cum", "pentru" etc. Astfel, rezultatele obținute sunt mult mai relevante pentru utilizatori.

Acest microserviciu gestionează și categoriile de produse, fiind strâns legat de microserviciul de produs. De fiecare dată când este necesară găsirea unor produse pe baza anumitor criterii, acesta implementează logica de selecție și determină ce produse trebuie extrase, apoi realizează un apel Feign[15] pentru a prelua produsele și a le returna utilizatorului.

Diagrama UML se poate găsi la *Anexa A4*

4.2.1.5 Chat

Microserviciul de chat nu este încă folosit în fluxul end-to-end al aplicației deoarece implementarea în frontend a sistemului de chat necesită un timp mai mare de lucru. Integrarea funcționalităților acestuia cu frontend-ul este următorul pas de dezvoltare în cadrul aplicației. Microserviciul dispune de 3 funcționalități principale:

- Trimiterea unui mesaj privat (Implementarea este gândită încât să fie ușor de scalat la potențiale conversații de grup sau adăugarea altor tipuri de mesaje decât clasicul text simplu)
- Returnarea previzualizărilor conversațiilor pe care utilizatorul le are, ordonate după data ultimului mesaj. Aceste previzualizări conțin numele corespondentului, poza acestuia și ultimul mesaj trimis în conversație. În să menționez că retragerea previzualizărilor se realizează tot prin paginare.
- Vizualizarea unei conversații complete

De asemenea, am ținut cont de securitatea datelor, astfel că toate mesajele sunt criptate folosind algoritmul "AES/CBC/PKCS5Padding"[17].

Consider că aceste funcționalități de început reprezintă un MVP pregătit de integrare cu frontend-ul, însă și una din componentele cu o complexitate potențial deosebit de mare.

Diagrama UML se poate găsi la *Anexa A5*

4.2.1.6 Librerie comună

În cadrul platformei SwapIt am considerat de folos o librărie comună care să deservească funcționalitățile comune ale tuturor microserviciilor, dar mai ales a dependențelor de diverse librării. Câteva din funcționalitățile tehnice comune sunt următoarele:

- Serviciu de gestionare a cache-ului cu posibilitatea de adăugare sau ștergere a unei chei
- Configurările necesare pentru serverul local de Hazelcast, inclusiv configurare de serializare a datelor
- Logica de tratare a erorilor în aplicație
- Generator de cod random pentru securitate
- Serviciu de criptare a datelor

Astfel, am reușit să centralizez dependențele și funcționalitățile comune, fiind un bonus pentru posibilele scalări viitoare ale aplicației.

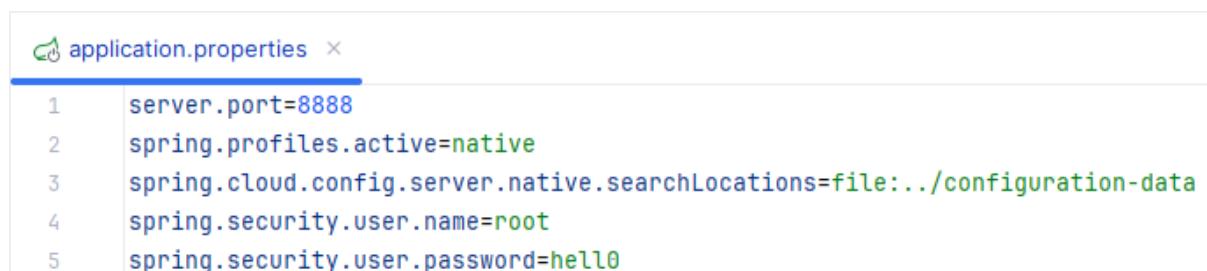
4.2.2 Server de configurări

Pentru a centraliza configurațiile aplicației, atât pentru acces la baza de date al microserviciului, alte configurații comune sau individuale, am ales să folosesc un microserviciu separat ce acționează ca un server de configurații. Acest microserviciu trebuie să primească anotația "@EnableConfigServer" pentru a putea prelua toate funcționalitățile unui astfel de server.

```
@SpringBootApplication  ↳ AlexandruOlteanu
@EnableConfigServer
public class ConfigurationServerApplication {

    public static void main(String[] args) {  ↳ AlexandruOlteanu
        SpringApplication.run(ConfigurationServerApplication.class, args);
    }
}
```

Figura 23: Exemplu inițializare server de configurații



```
application.properties
1 server.port=8888
2 spring.profiles.active=native
3 spring.cloud.config.server.native.searchLocations=file:../configuration-data
4 spring.security.user.name=root
5 spring.security.user.password=hello
```

Figura 24: Configurații necesare pentru ca serverul să știe locația de unde va încărca fișierele de configurație pentru microserviciile ce se vor conecta

O dată initializat, pentru ca un microserviciu să fie conectat la acest server de configurații, trebuie să adăugăm configurațiile următoare în application.properties:

```
application.properties
1 spring.cloud.config.enabled=true
2 spring.application.name=apiGateway
3 spring.profiles.active=local
4 spring.cloud.config.uri=http://localhost:8888
5 spring.cloud.config.username=root
6 spring.cloud.config.password=hello
7 spring.config.import=optional:configserver:http://localhost:8888
```

Figura 24: Setări necesare de conectare la serverul de configurații

Odată configurat, microserviciul preia setările corespunzătoare ierarhic, conform următoarei diagrame:

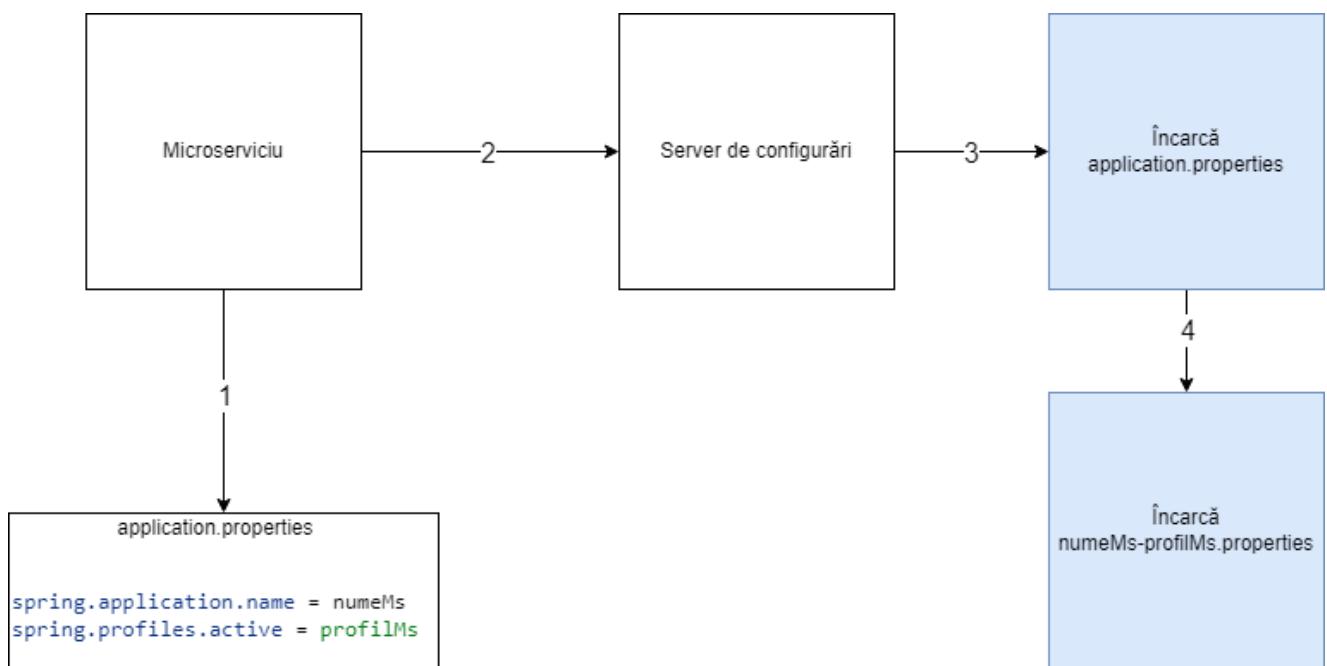


Figura 25: Proces de încărcare a configurațiilor pentru un microserviciu

De menționat este că putem avea proprietăți identice în application.properties și în fișierul pentru profilul actual, cel din urmă suprascriind configurația părintelui. Astfel, avem același comportament ca moștenirea unei clase.

4.2.3 Baza de date

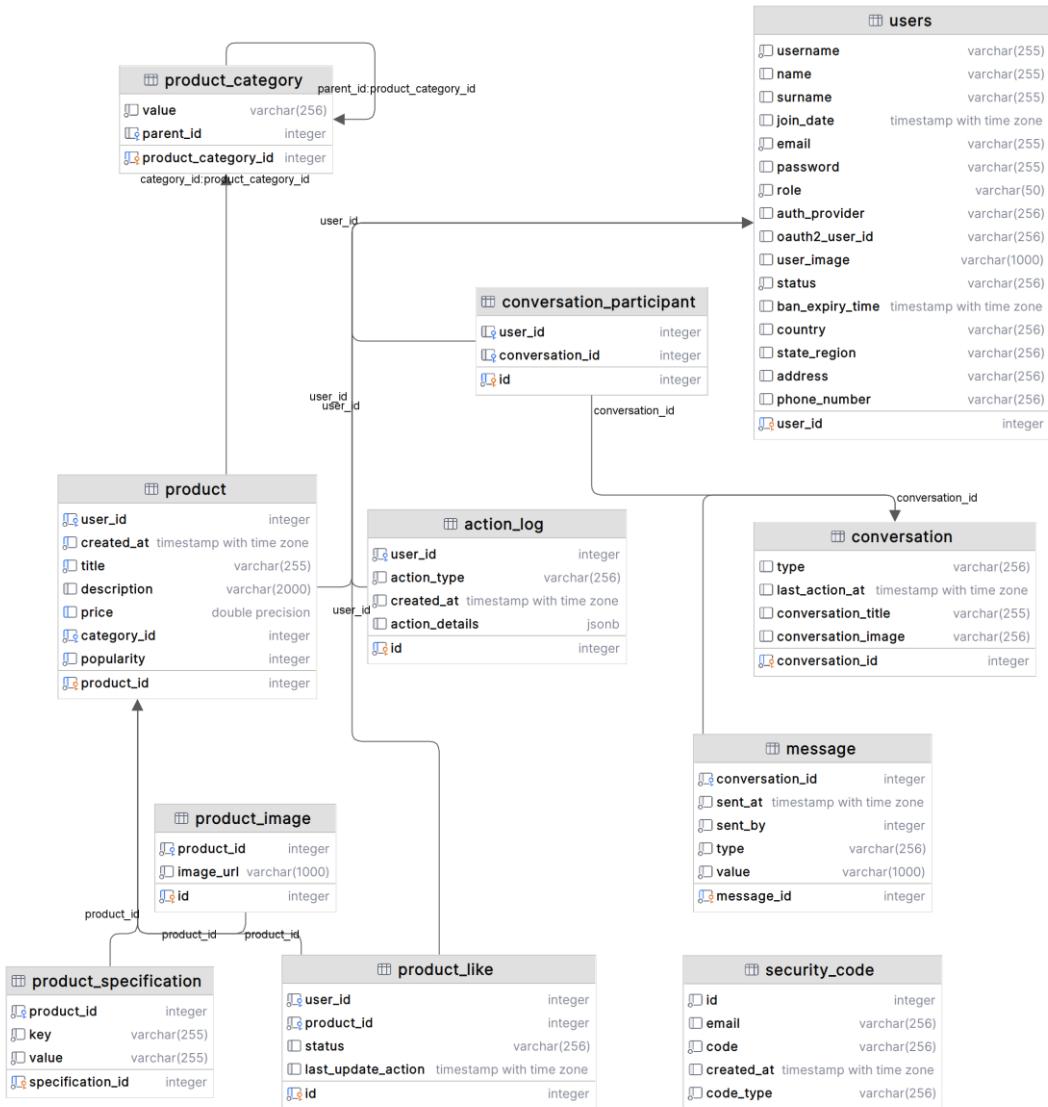


Figura 26: Structura bazei de date

Baza de date are în componență 11 tabele, fiecare având un rol important în cadrul aplicației după cum urmează:

alex

1. Tabela users: Această tabelă include detalii complete ale unui utilizator cât și date despre statusul contului acestui utilizator, conținând următoarele coloane:
 - user_id -> cheia primară a tabelei, include id-ul unic al utilizatorului după care se realizează majoritatea funcționalităților - Integer

- username -> identificatorul general al utilizatorului – varchar(255)
 - name, surname -> numele și prenumele utilizatorului – varchar(255)
 - join_date -> data la care utilizatorul s-a înregistrat, inclusiv ora – timestamp with time zone
 - email -> adresa de email a utilizatorului – varchar(255)
 - password -> parola utilizatorului – varchar(255)
 - role -> rolul utilizatorului în sistem – varchar(50)
 - auth_provider -> furnizorul de autentificare – varchar(256)
 - oauth2_user_id -> id-ul utilizatorului pentru autentificarea prin OAuth2 – varchar(256)
 - user_image -> URL-ul imaginii de profil a utilizatorului – varchar(1000)
 - status -> statusul contului utilizatorului – varchar(256)
 - ban_expiry_time -> data și ora la care expiră o eventuală interdicție a utilizatorului – timestamp with time zone
 - country, state_region, address -> adresa completă a utilizatorului – varchar(256)
 - phone_number -> numărul de telefon al utilizatorului – varchar(256)
2. Tabela product: Această tabelă conține informații detaliate despre produsele disponibile, conținând următoarele coloane:
- product_id -> cheia primară a tablei, id-ul unic al produsului – Integer
 - user_id -> cheie străină, id-ul utilizatorului care a adăugat produsul – Integer
 - created_at -> data și ora adăugării produsului – timestamp with time zone
 - title -> titlul produsului – varchar(255)
 - description -> descrierea detaliată a produsului – varchar(2000)
 - price -> prețul produsului – double precision
 - category_id -> id-ul categoriei din care face parte produsul – Integer
 - popularity -> popularitatea produsului – Integer
3. Tabela product_image: Această tabelă stochează imaginile asociate produselor, conținând următoarele coloane:
- id -> cheie străină, id-ul imaginii de produs - Integer
 - product_id -> cheie străină, id-ul produsului – Integer
 - image_url -> URL-ul imaginii produsului – varchar(1000)
4. Tabela product_category: Această tabelă stochează categoriile de produse, permitând structurarea acestora în subcategorii prin intermediul unui parent_id, conținând următoarele coloane:
- product_category_id -> cheia primară a tablei, id-ul unic al categoriei de produs - Integer
 - value -> denumirea categoriei – varchar(256)
 - parent_id -> cheie străină, id-ul categoriei părinte, pentru a crea o structură ierarhică – Integer

5. Tabela `product_specification`: Această tabelă conține specificațiile produselor, conținând următoarele coloane:
- `specification_id` -> cheia primară a tableei, id-ul unic al specificației – Integer
 - `product_id` -> cheie străină, id-ul produsului – Integer
 - `key` -> cheia specificației – varchar(255)
 - `value` -> valoarea specificației – varchar(255)
6. Tabela `product_like`: Această tabelă stochează informații despre aprecierile produselor de către utilizatori, conținând următoarele coloane:
- `id` -> cheia primară a tablelei, id-ul unic al aprecierii – Integer
 - `user_id` -> cheie străină, id-ul utilizatorului care a apreciat produsul – Integer
 - `product_id` -> cheie străină, id-ul produsului apreciat – Integer
 - `status` -> statusul aprecierii – varchar(256)
 - `last_update_action` -> data și ora ultimei acțiuni de actualizare – timestamp with time zone
7. Tabela `action_log`: Această tabelă stochează loguri de acțiuni efectuate de utilizatori, conținând următoarele coloane:
- `id` -> cheia primară a tablelei, id-ul unic al acțiunii – Integer
 - `user_id` -> id-ul utilizatorului care a efectuat acțiunea – Integer
 - `action_type` -> tipul acțiunii – varchar(256)
 - `created_at` -> data și ora acțiunii – timestamp with time zone
 - `action_details` -> detalii despre acțiune – jsonb
8. Tabela `conversation`: Această tabelă conține informații despre conversațiile între utilizatori, conținând următoarele coloane:
- `conversation_id` -> cheia primară a tablelei, id-ul unic al conversației – Integer
 - `type` -> tipul conversației – varchar(256)
 - `last_action_at` -> data și ora ultimei acțiuni din conversație – timestamp with time zone
 - `conversation_title` -> titlul conversației – varchar(256)
 - `conversation_image` -> URL-ul imaginii conversației – varchar(256)
9. Tabela `conversation_participant`: Această tabelă conține informații despre participanții la conversații, fiind o tabelă de legătură, conținând următoarele coloane:
- `id` -> cheia primară a tablelei, id-ul unic al înregistrării – Integer
 - `user_id` -> cheie străină, id-ul utilizatorului participant – Integer
 - `conversation_id` -> cheie străină, id-ul conversației – Integer
10. Tabela `message`: Această tabelă conține mesajele trimise în cadrul conversațiilor, conținând următoarele coloane:
- `message_id` -> cheia primară a tablelei, id-ul unic al mesajului – Integer

- conversation_id -> cheie străină, id-ul conversației din care face parte mesajul – Integer
 - sent_at -> data și ora trimiterii mesajului – timestamp with time zone
 - sent_by -> id-ul utilizatorului care a trimis mesajul – Integer
 - type -> tipul mesajului – varchar(256)
 - value -> conținutul mesajului – varchar(1000)

11. Tabela security_code: Această tabelă conține codurile de securitate generate pentru utilizatori, continând următoarele coloane:

- id -> cheia primară a tabelei, id-ul unic al codului de securitate – Integer
 - email -> adresa de email asociată codului de securitate – varchar(256)
 - code -> codul de securitate – varchar(256)
 - created_at -> data și ora generării codului – timestamp with time zone
 - code_type -> tipul codului de securitate – varchar(256)

4.2.3.1 Accesul la baza de date

Pentru a interacționa cu baza de date, am ales să folosesc una dintre cele mai populare metode, JpaRepository. Aceasta oferă o mulțime de interogări preconstruite care pot fi folosite pentru a simplifica operațiunile comune de CRUD (Create, Read, Update, Delete).

Un alt aspect important al utilizării JpaRepository este că se bazează pe Hibernate, care este implementarea implicită a JPA (Java Persistence API) în Spring Boot. Hibernate este un ORM (Object-Relational Mapping) puternic și flexibil care facilitează maparea obiectelor Java la tabelele din baza de date. Aceasta reduce considerabil complexitatea codului necesar pentru a gestiona datele persistente.

```
20 @Data 37 usages ± AlexandruOlteanu
21 @Builder
22 @NoArgsConstructor
23 @AllArgsConstructor
24 @org.hibernate.annotations.Cache(usage = CacheConcurrencyStrategy.READ_WRITE)
25 @Entity
26 @Table(name = "users")
27  public class User implements UserDetails {
28
29     @Id
30     @GeneratedValue(strategy = GenerationType.IDENTITY)
31     @Column(name="user_id")
32     private Integer userId;
33
34     @Column(name = "username")
35     private String username;
36
37     @Column(name = "join_date")
38     private ZonedDateTime joinDate;
39
40     @Column(name = "name")
41     private String name;
42
43     @Column(name = "surname")
44     private String surname;
45
46     @Column(name = "email")
47     private String email;
48
49     @Column(name = "country")
50     private String country;
```

Figura 27: Exemplu de declarare a unei clase ce reprezintă entitatea tabelăi din baza de date

```

11 public interface UserRepository extends JpaRepository<User, Integer> { 12 usages ± AlexandruOlteanu
12
13     Optional<User> findUserByUsername(String username); 5 usages ± AlexandruOlteanu
14     Optional<User> findUserByEmail(String email); 4 usages ± AlexandruOlteanu
15     Optional<User> findUserByOAuth2UserId(String oauth2UserId); 1 usage ± AlexandruOlteanu
16     @Query("select u.username from User u where u.username like :prefix%") 1 usage ± AlexandruOlteanu
17     Optional<List<String>> getUsernameStartingWith(String prefix);
18     Optional<List<User>> findAllByStatus(UserStatus status); 1 usage ± AlexandruOlteanu
19 }

```

Figura 28: Exemplu de utilizare a JpaRepository pentru clasa de utilizator

4.3 Descrierea fluxului funcționalităților

Pentru a descrie corespunzător fluxul de funcționalități, am ales să reprezent acțiunile posibile la fiecare pas pe care le are un utilizator în funcție de tipul acestuia: utilizator anonim, utilizator autentificat și administrator, după cum urmează:

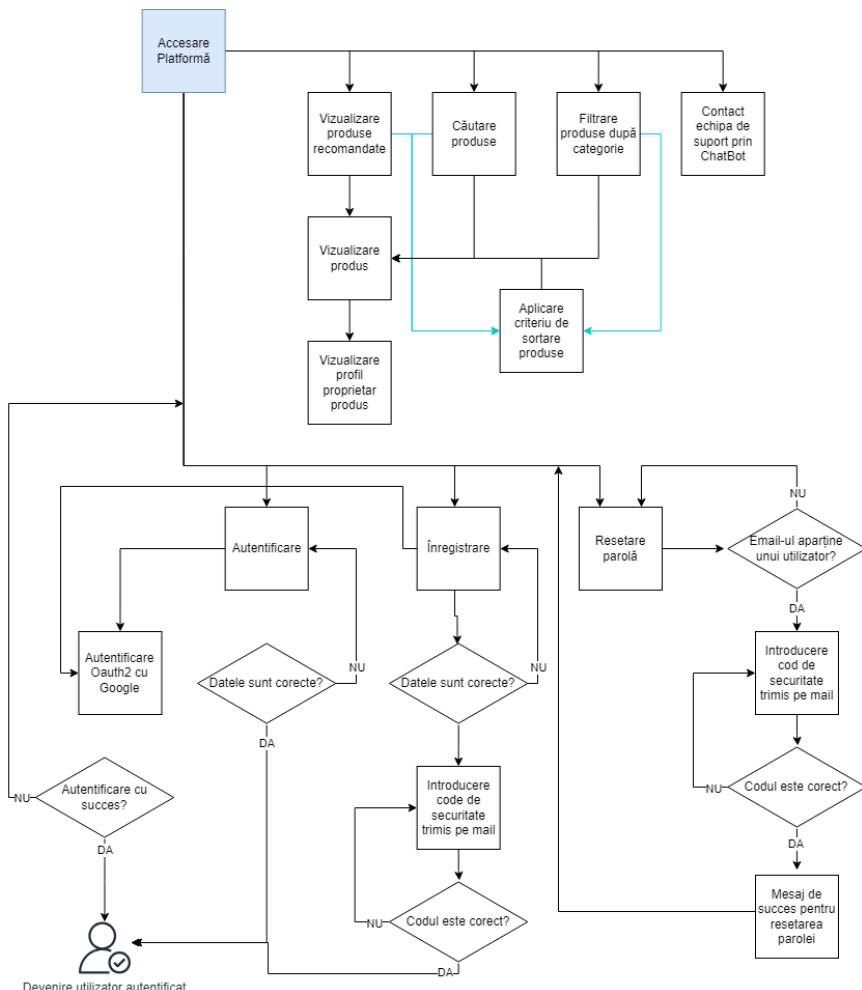


Figura 29: Diagrama de flux pentru utilizatorii anonimi

Observăm că utilizatorii anonimi au acces la vizualizarea datelor publice precum căutarea de produse, vizualizarea individuală a acestora, posibilitatea de a vizualiza cine a postat un produs în platformă, interacțiunea cu echipa de suport, etc. De asemenea, în figura de mai sus este descris procesul de autentificare, înregistrare sau resetare a parolei și cum utilizatorii anonimi pot deveni autentificați.

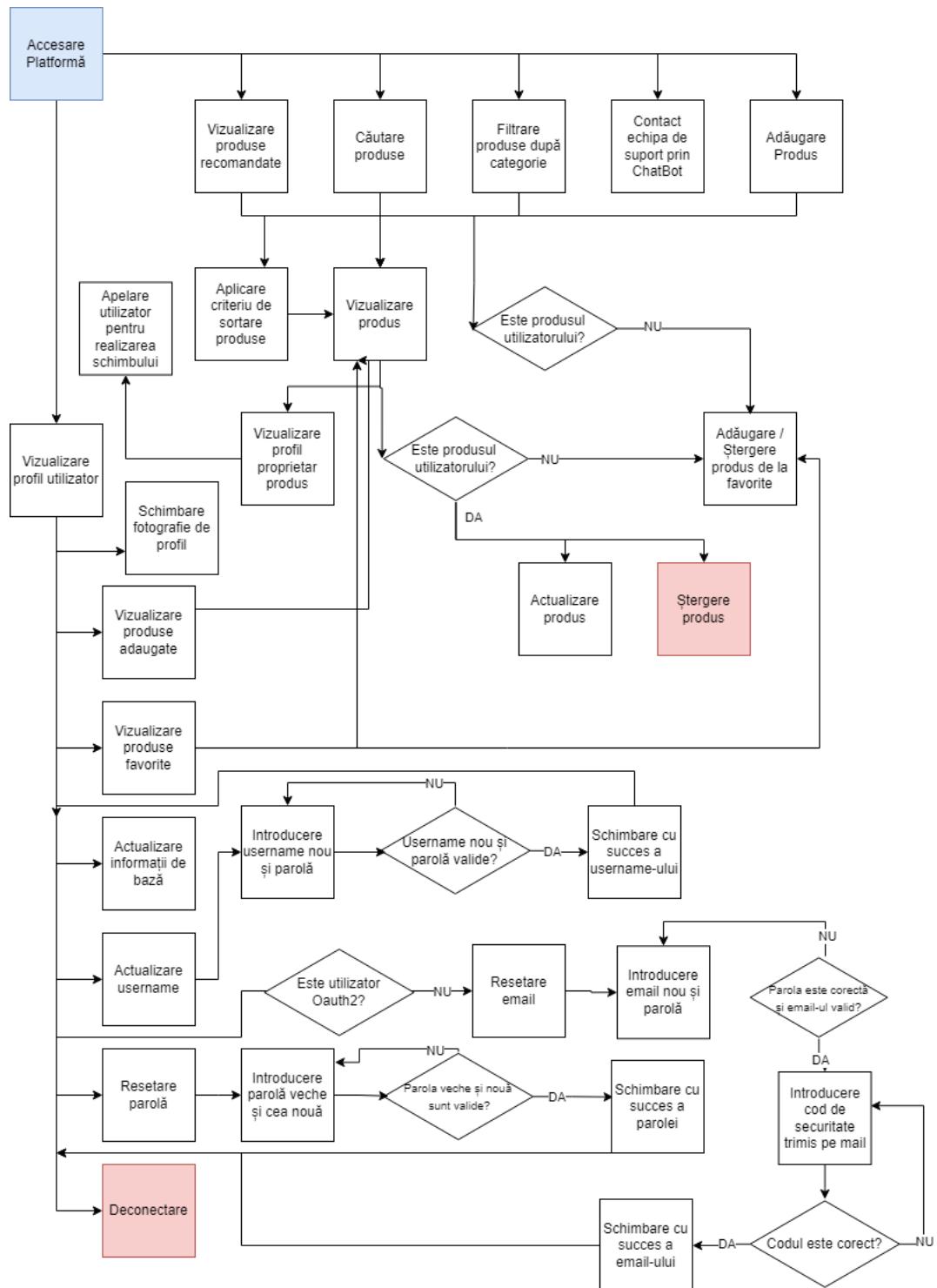


Figura 30: Diagrama de flux pentru utilizatorii autentificați

Se poate observa că utilizatorii autentificați pot interacționa mai mult cu produsele, având posibilitatea să le creeze, actualizeze, șteargă sau să le adauge la lista de favorite. De asemenea, odată autentificat, utilizatorul are acces la administrarea propriului cont de unde își poate schimba poza de profil, datele de bază sau cele private precum parola, email-ul sau username-ul. Pentru a realiza un schimb de produs, utilizatorul apelează telefonic deținătorul produsului.

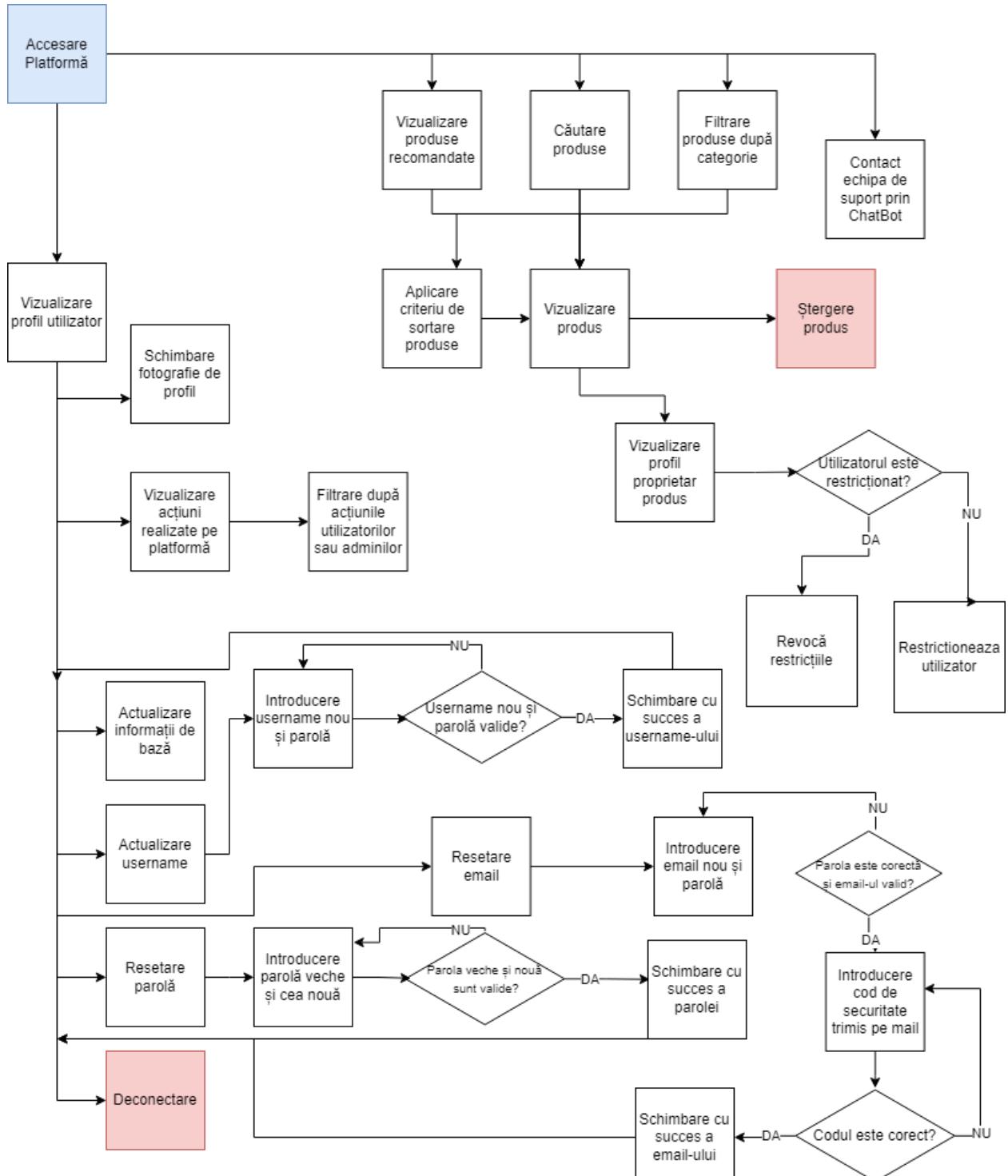


Figura 31: Diagrama de flux pentru administrator

Ca administrator, pe lângă funcționalitățile clasice, acesta are acces la acțiunile realizate de utilizatori, poate să șteargă produsele neadecvate și de asemenea să restricționeze accesul unui alt utilizator. Fiind administrator, acesta nu are produse favorite sau posibilitatea de a crea produse.

5 DETALII DE IMPLEMENTARE

5.1 Securitate

Securitatea aplicației a fost realizată în totalitate în microserviciul Api Gateway. Înainte de procesarea oricărei cereri, aceasta trece prin filtre de securitate și este respinsă în caz de neautorizare. Api-urile apelate sunt de 4 tipuri:

- publice: /api/v1/swapIt/apiGateway/publicAccess/** (oferă acces oricărui client ce folosește aplicația, indiferent dacă este sau nu autentificat)
- de autentificare: /api/v1/swapIt/apiGateway/auth/** (acces pentru autentificare, tot public)
- private: /api/v1/swapIt/apiGateway/** (necessită un JWT token valid sau cookie cu o sesiune Oauth2 validă)
- administrative: /api/v1/swapIt/apiGateway/adminAction/** (pe langă autentificare necesită autoritatea de administrator)

```

24     * @EnableMethodSecurity
25     public class SecurityConfiguration {
26
27         private static final String[] WHITE_LIST_URL = { 1 usage
28             "/api/v1/swapIt/apiGateway/auth/**",
29             "/v3/**",
30             "/swagger-ui/**",
31             "/api/v1/swapIt/apiGateway/publicAccess/**"
32         };
33
34         private final JwtAuthenticationFilter jwtAuthFilter;
35         private final AuthenticationProvider authenticationProvider;
36         private final CustomAuthenticationEntryPoint customAuthenticationEntryPoint;
37         private final CustomAuthenticationSuccessHandler customAuthenticationSuccessHandler;
38         private final CustomLogoutHandler customLogoutHandler;
39         @Bean
40         public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
41             return http
42                 .addFilterBefore(jwtAuthFilter, UsernamePasswordAuthenticationFilter.class)
43                 .csrf(AbstractHttpConfigurer::disable)
44                 .authorizeHttpRequests(req -> {
45                     req.requestMatchers(WHITE_LIST_URL).permitAll();
46                     req.requestMatchers(2 "/api/v1/swapIt/apiGateway/adminAction/**").hasAnyAuthority(ADMINISTRATOR.name());
47                     req.anyRequest().authenticated();
48                 })
49                 .oauth2Login(oauth2Configurer -> oauth2Configurer.successHandler(customAuthenticationSuccessHandler))
50                 .exceptionHandling(exceptionHandlingConfigurer -> exceptionHandlingConfigurer.authenticationEntryPoint(customAuthenticationEntryPoint))
51                 .authenticationProvider(authenticationProvider)
52                 .logout(httpSecurityLogoutConfigurer -> {
53                     httpSecurityLogoutConfigurer.logoutUrl("/api/v1/swapIt/apiGateway/auth/logout");
54                     httpSecurityLogoutConfigurer.addLogoutHandler(customLogoutHandler);
55                     httpSecurityLogoutConfigurer.logoutSuccessHandler((_, response, _) -> response.setStatus(HttpStatus.SC_OK));
56                     httpSecurityLogoutConfigurer.invalidateHttpSession(true);
57                     httpSecurityLogoutConfigurer.clearAuthentication(true);
58                     httpSecurityLogoutConfigurer.deleteCookies( ...cookieNamesToClear, "JSESSIONID");
59                 })
60             .build();
61         }
62     }

```

Figura 32: Setările de configurare pentru securitate

Se poate observa cum fiecare cerere este filtrată, trecută mai întâi printr-un serviciu de verificare al token-ului JWT ce va fi detaliat în secțiunea următoare. Dacă JWT token-ul este valid sau cererea face parte din lista de whitelist unde am pus rutele api-urilor publice și cele de Swagger[19], atunci aceasta devine autentificată. Fiind autentificată, dacă cererea este pentru un api administrativ, verifică adițional rolul utilizatorului care este acum preluat din contextul de securitate al spring-ului. În cazul în care rolul nu e de administrator, accesul este respins, altfel cererea este acceptată și poate fi procesată.

Un al doilea flux de funcționalitate ce ne este oferit este prin autentificare Oauth2 care este interceptată automat la redirecționarea din Google către aplicația noastră.

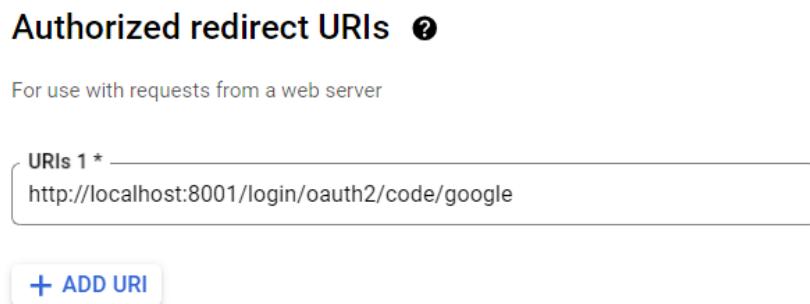


Figura 33: Configurare pentru redirecționarea după autentificarea Oauth2 cu succes.

Validarea identității și autorizării prin JWT token este responsabilitatea platformei, însă autentificarea prin Oauth2 depinde de o aplicație externă, aplicația SwapIt interceptând contextul de autentificare după ce se realizează autentificarea externă.

Un alt punct important este serviciul de deconectare al utilizatorului care realizează invalidarea autentificării și a cookie-urilor pentru autentificarea Oauth2.

5.1.1 JWT token

Autentificarea prin JWT token este una dintre cele mai populare metode folosite pentru securitate și autentificare, fiind o metodă de autentificare fără stare ce stocă o cheie semnată de către server ce este validată cu ajutorul unei chei private din aplicația de backend. Am ales să folosesc această metodă de autentificare datorită ușurinței de implementare și de gestionare atât din backend, cât și din frontend. Așa cum am amintit anterior, pentru ca o cerere să fie validată și autentificată prin JWT token, am adăugat un filtru ce mă ajută la interceptarea cererii, extragerea token-ului venit pe antet și validarea acestuia după cum urmează:

```

@Component  ± AlexandruOlteanu
@RequiredArgsConstructor
@Slf4j
public class JwtAuthenticationFilter extends OncePerRequestFilter {

    private final JwtService jwtService;
    private final UserDetailsService userDetailsService;
    private final UserRepository userRepository;
    private final String AUTHORIZATION = "Authorization";  1 usage
    private final String BEARER = "Bearer";  2 usages
    private static final String USER_ID_JWT_KEY = "user_id";  1 usage

    @Override  no usages  ± AlexandruOlteanu
    protected void doFilterInternal(@NotNull HttpServletRequest request, @NotNull HttpServletResponse response, @NotNull FilterChain filterChain)
        String authorizationHeader = request.getHeader(AUTHORIZATION);
        if (authorizationHeader != null && authorizationHeader.startsWith(BEARER)) {
            String token = authorizationHeader.replace(BEARER, replacement: "");
            Integer userId = jwtService.getClaimFromTokenByName(token, USER_ID_JWT_KEY, Integer.class);
            User user = userRepository.findById(userId).orElse(User.builder().build());
            String userName = user.getUsername();

            if (userName != null && SecurityContextHolder.getContext().getAuthentication() == null) {
                UserDetails userDetails = userDetailsService.loadUserByUsername(userName);

                if (!jwtService.isTokenExpired(token)) {
                    UsernamePasswordAuthenticationToken authenticationToken = new UsernamePasswordAuthenticationToken(
                        userDetails, credentials: null, userDetails.getAuthorities());
                    authenticationToken.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));
                    SecurityContextHolder.getContext().setAuthentication(authenticationToken);
                }
            }
        }
        filterChain.doFilter(request, response);
    }
}

```

Figura 34: Serviciul de validare al token-ului JWT

Token-ul este extras din antetul cererii. Dacă nu există sau nu este de tip JWT (un token JWT trebuie să aibă prefixul Bearer), atunci serviciul nu face nimic și lasă cererea să fie filtrată în continuare după celelalte criterii. În caz contrar, token-ul efectiv este extras, apoi este extras câmpul de id utilizator cu ajutorul căruia încărcăm utilizatorul din baza de date. Dacă utilizatorul există, token-ul este validat, se creează identitatea utilizatorului în contextul de Spring și astfel cererea este autentificată.

```

@Service  ± AlexandruOlteanu *
@RequiredArgsConstructor
public class JwtService {

    @Value("${application.security.jwt.secretKey}")
    private String secretKey;

    @Value("${application.security.jwt.expiration}")
    private long jwtExpirationMs;

    @Value("${application.security.jwt.refreshToken.expiration}")
    private long refreshExpirationMs;

    @Getter
    private Key signingKey;

    @PostConstruct  ± AlexandruOlteanu
    public void init() {
        this.signingKey = Keys.hmacShaKeyFor(secretKey.getBytes(StandardCharsets.UTF_8));
    }

    private String createToken(Map<String, Object> claims, String subject) {  1 usage  new *
        return createToken(claims, subject, jwtExpirationMs);
    }

    private String createToken(Map<String, Object> claims, String subject, long expirationMs) {
        long currentTimeMillis = System.currentTimeMillis();
        return Jwts.builder()
            .setClaims(claims)
            .setSubject(subject)
            .setIssuedAt(new Date(currentTimeMillis))
            .setExpiration(new Date(currentTimeMillis + expirationMs))
            .signWith(signingKey, SignatureAlgorithm.HS256)
            .compact();
    }
}

```

Figura 35: Serviciul de creare și semnare a unui JWT token

5.2 Autentificare și Înregistrare

5.2.1 BCryptPasswordEncoder

BCryptPasswordEncoder este o componentă din Spring Security folosită pentru criptarea securizată a parolelor, utilizând algoritmul BCrypt care este conceput să reziste atacurilor brute-force. Procesul implică generarea unei sări unice pentru fiecare parolă, astfel încât parolele identice produc hash-uri diferite, apoi parola este hashuită împreună cu sarea folosind algoritmul BCrypt, care aplică multiple runde de hashing pentru a spori securitatea. Hash-ul rezultat, care include sarea și factorul de cost (numărul de runde), este stocat în baza de date. La verificare, hash-ul stocat este împărțit pentru a extrage sarea și factorul de cost, apoi parola introdusă este hashată cu aceste valori și comparată cu hash-ul stocat; dacă hashurile coincid, parola este corectă.

5.2.2 Autentificare și Înregistrare locală

Pentru autentificarea locală utilizatorul trebuie să facă o cerere de autentificare pe baza numelui de utilizator și a parolei, cerere care este verificată de managerul de autentificare prezent în librăria Spring. Parola, fiind stocată în baza de date prin aplicarea algoritmului BCrypt, este verificată automat în același mod. Dacă autentificarea eșuează, eroarea este interceptată și se returnează un mesaj corespunzător indicând o parolă sau un nume de utilizator incorecte. În cazul unei autentificări reușite, se generează un token JWT, care este trimis în răspuns pentru ca frontend-ul să îl poată prelua și utiliza ca antet de autorizare în cererile viitoare.

```
@Configuration("authSecurityConfiguration") AlexandruOlteanu
@RequiredArgsConstructor
public class AuthConfig {

    private final UserRepository userRepository;
    @Bean AlexandruOlteanu
    public UserDetailsService userDetailsService() {
        return username -> userRepository.findUserByUsername(username)
            .orElseThrow(() -> new UsernameNotFoundException("User not found"));
    }

    @Bean AlexandruOlteanu
    public AuthenticationManager authenticationManager(AuthenticationConfiguration config) {
        return config.getAuthenticationManager();
    }

    @Bean AlexandruOlteanu
    public AuthenticationProvider authenticationProvider() {
        DaoAuthenticationProvider authProvider = new DaoAuthenticationProvider();
        authProvider.setUserDetailsService(userDetailsService());
        authProvider.setPasswordEncoder(passwordEncoder());
        return authProvider;
    }

    @Bean AlexandruOlteanu
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }
}
```

Figura 36: Configurările necesare pentru ca managerul de autentificare să verifice corect parola și identitatea utilizatorului

```

@Override 1 usage  ± AlexandruOlteanu
public LoginResponse login(LoginRequest request) {
    try {
        authenticationManager.authenticate(new UsernamePasswordAuthenticationToken(request.getUsername(), request.getPassword()));
    } catch (Exception e) {
        throw exceptionFactory.create(ExceptionType.WRONG_USERNAME_OR_PASSWORD);
    }
    User user = userRepository.findUserByUsername(request.getUsername())
        .orElseThrow(() -> exceptionFactory.create(ExceptionType.USER_NOT_FOUND));
    if (user.getStatus().equals(UserStatus.TEMPORARY_BANNED) || user.getStatus().equals(UserStatus.PERMANENT_BANNED)) {
        throw exceptionFactory.create(ExceptionType.USER_BANNED);
    }
    return LoginResponse.builder()
        .jwtToken(jwtService.generateToken(user))
        .build();
}

```

Figura 37: Serviciul de autentificare locală

Pentru înregistrarea unui utilizator local am ales ca procesul să se realizeze în 3 pași distincti, pași ce favorizează interacțiunea cu frontend-ul:

- verificarea datelor introduse: se verifică dacă username-ul sau email-ul ales este deja înregistrat în platformă, caz în care se întoarce o eroare corespunzătoare
- trimitera codului de securitate: odată verificate datele, în această fază a procesului se va trimite codul de securitate pe email-ul utilizatorului
- finalizarea: după ce codul a fost trimis urmează faza finală în care se validează codul de securitate și se salvează datele noului utilizator în baza de date, inclusiv ștergerea codului de securitate folosit

Aceste trei etape din procesul de înregistrare, precum și din alte funcționalități precum resetarea parolei sau a adresei de email, au fost necesare datorită următorului scenariu: dacă am fi ales să realizăm cele trei etape separat ar fi fost nevoie de trei API-uri diferite. În cazul în care am fi încercat să combinăm trimitera codului cu verificarea datelor utilizatorul ar fi trebuit să aștepte pe ecranul de trimitere a codului aproximativ o secundă și jumătate până la trimiterea cu succes a codului. Nu puteam să îl trimitem indiferent de timpul de finalizare al apelului pe ecranul de introducere a codului, deoarece puteau apărea probleme la verificarea datelor, caz în care utilizatorul ar fi trebuit să reintroducă alte date. Astfel, am fi ajuns la practici incorecte fie în frontend, fie în backend.

Prin urmare, parcurgând independent cele trei etape, utilizatorul se poate înregistra, parola este stocată ca rezultat al aplicării algoritmului BCrypt, iar în răspuns este returnat token-ul JWT proaspăt generat.

```

@Override 1 usage  ± AlexandruOlteanu
@Transactional
public RegisterResponse register(RegisterRequest request) {

    Optional<User> existingUser = userRepository.findUserByUsername(request.getUsername());
    if (existingUser.isPresent()) {
        throw exceptionFactory.create(ExceptionType.USERNAME_ALREADY_EXISTS);
    }
    existingUser = userRepository.findUserByEmail(request.getEmail());
    if (existingUser.isPresent()) {
        throw exceptionFactory.create(ExceptionType.EMAIL_ALREADY_EXISTS);
    }

    if (RegisterProcessPhase.SEND_SECURITY_CODE.equals(request.getProcessPhase())) {
        securityCodeService.sendSecurityCode(SendSecurityCodeRequest.builder()
            .email(request.getEmail())
            .securityCodeType(REGISTRATION)
        .build());
    }

    if (RegisterProcessPhase.FINALIZE.equals(request.getProcessPhase())) {
        securityCodeRepository.findByEmailAndCodeAndCodeType(request.getEmail(), request.getSecurityCode(), REGISTRATION)
            .orElseThrow(() -> exceptionFactory.create(ExceptionType.WRONG_SECURITY_CODE));
        securityCodeRepository.deleteByEmailAndCodeAndCodeType(request.getEmail(), request.getSecurityCode(), REGISTRATION);
        User user = User.builder()
            .username(request.getUsername())
            .name(request.getName())
            .surname(request.getSurname())
            .email(request.getEmail())
            .password(passwordEncoder.encode(request.getPassword()))
            .userRole(UserRole.USER)
            .status(UserStatus.ACTIVE)
            .authProvider(AuthProvider.LOCAL)
            .userImage(request.getUserImage())
        .build();
        user = userRepository.save(user);
        return RegisterResponse.builder()
            .userId(user.getUserId())
            .jwtToken(jwtService.generateToken(user))
        .build();
    }
    return RegisterResponse.builder().build();
}

```

Figura 38: Serviciul de înregistrare locală

5.2.3 Autentificare și înregistrare prin Oauth2

Una dintre provocări a fost răspunsul la următoarea întrebare: „Cum asociem utilizatorii Oauth2 cu utilizatorii stocați în baza de date?”. Odată realizată autentificarea Oauth2 prin Google, apelul este redirecționat către Api Gateway și mai apoi preluat de un serviciu din setările de securitate discutate anterior. Contextul de autentificare este preluat și putem extrage toate datele publice pe care Google ni le oferă despre utilizatori. Printre aceste date se află ID-ul unic al contului de Google, imaginea de profil, numele, prenumele și email-ul. Astfel, având aceste date, putem recunoaște un utilizator Oauth2 după acest ID unic, să realizăm asocierea cu utilizatorul corespunzător din baza de date și astfel să îl integrăm în aplicația noastră ca și cum ar fi un utilizator obișnuit.

De menționat este că în Api Gateway se extrag aceste date și se apelează API-ul din microserviciul de utilizator pentru autentificare sau înregistrare, iar în cazul unei înregistrări se auditează această acțiune. Pentru a restricționa accesul unui utilizator care are statutul de restricție temporară sau permanentă, extragem statutul acestuia și, în caz de restricție, invalidăm sesiunea și îl redirecționăm către pagina corespunzătoare de eroare din frontend.

```

public class CustomAuthenticationSuccessHandler extends SavedRequestAwareAuthenticationSuccessHandler {

    private final UserRepository userRepository;
    private final ExternalOperationsService externalOperationsService;
    @Value("${apiGateway.ui.userBanPage.redirect.uri}")
    private String uiUserBanPage;
    @Value("${ui.successfulAuthentication.route}")
    private String successfulAuthenticationUri;
    private static final String OAUTH2_USER_ID = "sub"; 1 usage
    private static final String USER_IMAGE = "picture"; 1 usage
    private static final String NAME = "family_name"; 1 usage
    private static final String SURNAME = "given_name"; 1 usage
    private static final String EMAIL = "email"; 1 usage

    @Override no usages ± AlexandruOlteanu
    public void onAuthenticationSuccess(HttpServletRequest request, HttpServletResponse response, Authentication authentication)
        OAuth2User oAuth2User = ((OAuth2AuthenticationToken) authentication).getPrincipal();
        OAuth2Request oauth2Request = OAuth2Request.builder()
            .oauth2UserId(oAuth2User.getAttribute(OAUTH2_USER_ID))
            .userImage(oAuth2User.getAttribute(USER_IMAGE))
            .name(oAuth2User.getAttribute(NAME))
            .surname(oAuth2User.getAttribute(SURNAME))
            .email(oAuth2User.getAttribute(EMAIL))
            .build();
        OAuth2Response loginResponse = externalOperationsService.oauth2Login(oauth2Request);
        if (loginResponse.getRegisteredNow().equals(Boolean.TRUE)) {
            externalOperationsService.auditUserAction(AuditUserActionRequest.builder()
                .actionType(ActionType.USER_REGISTER)
                .userId(loginResponse.getUserId())
                .build());
        }
        User user = userRepository.findById(loginResponse.getUserId()).orElse(null);
        assert user != null;
        if (user.getStatus().equals(UserStatus.TEMPORARY_BANNED) || user.getStatus().equals(UserStatus.PERMANENT_BANNED)) {
            HttpSession session = request.getSession(false);
            if (session != null) {
                session.invalidate();
            }
            response.sendRedirect(uiUserBanPage);
            return;
        }
        response.sendRedirect(successfulAuthenticationUri);
    }
}

```

Figura 39: Extragerea și procesarea datelor utilizatorului Oauth2

```

@Override 1 usage ± AlexandruOlteanu
public OAuth2Response oauth2Login(OAuth2Request request) {
    User existingUser = userRepository.findUserByOauth2UserId(request.getOauth2UserId())
        .orElse(null);
    Integer userId;
    if (existingUser != null) {
        existingUser.setEmail(request.getEmail());
        existingUser.setName(request.getName());
        existingUser.setSurname(request.getSurname());
        userRepository.save(existingUser);
        userId = existingUser.getId();
    } else {
        String username = generateNewUsername(request.getSurname(), request.getName());
        User newUser = User.builder()
            .username(username)
            .name(request.getName())
            .surname(request.getSurname())
            .email(request.getEmail())
            .userRole(UserRole.OAUTH2_USER)
            .status(UserStatus.ACTIVE)
            .authProvider(AuthProvider.OAUTH2)
            .oauth2UserId(request.getOauth2UserId())
            .userImage(request.getUserImage())
            .build();
        userId = userRepository.save(newUser).getId();
    }
    return OAuth2Response.builder()
        .userId(userId)
        .registeredNow(existingUser == null)
        .build();
}

```

Figura 40: Procesul de autentificare și înregistrare a unui utilizator Oauth2

5.2.4 Generarea unui username unic

În momentul înregistrării unui utilizator în platformă prin OAuth2, acesta nu trece prin procesul de alegere a unui username, aşa că aplicația va trebui să îi atribuie unul automat. Ideal ar fi ca username-ul să fie de forma prenume_nume, numele și prenumele venind de la Google. Problema apare în momentul în care avem deja un utilizator înregistrat cu acest nume, ceea ce înseamnă că ideal să adăugăm și un număr la final. Dar ce se întâmplă dacă avem și utilizatori cu același username, dar și cu numere la final? Aici, soluția este să găsim primul număr care nu este folosit și să îl atribuim la finalul username-ului curent.

Pentru a realiza acest lucru, trebuie în primul rând să extragem toți utilizatorii ce au username-ul de formă „prenume_nume” urmat de orice caractere. Pentru a obține acest rezultat, am folosit o interogare corespunzătoare în baza de date:

```
@Query("select u.username from User u where u.username like :prefix%")
Optional<List<String>> getUsernameStartingWith(String prefix);
```

Figura 41: Interogare pentru aflarea tuturor username-urilor ce încep cu un anumit prefix

Apoi, am extras doar username-urile care conțin numere la final, am creat o listă cu acestea și le-am ordonat pentru a putea aplica algoritmul de căutare binară [20] pentru a găsi primul indice pentru care valoarea din sir este diferită de valoarea indicelui. Astfel, complexitatea este $O(n * \log_2 n)$, fiind optimă pentru găsirea unui număr ce poate fi adăugat la finalul username-ului.

```
private String generateNewUsername(String surname, String name) { 1 usage  ± AlexandruOlteanu
    if (surname != null) surname = surname.toLowerCase();
    if (name != null) name = name.toLowerCase();
    String joinedPrefix = Stream.of(surname, name).filter(Objects::nonNull)
        .collect(Collectors.joining("_"));
    List<String> results = userRepository.getUsernameStartingWith(joinedPrefix)
        .orElse(new ArrayList<>());
    Pattern pattern = Pattern.compile("joinedPrefix + "(\\d*)$");
    List<Integer> suffix = new ArrayList<>();
    results.forEach(hit -> {
        Matcher matcher = pattern.matcher(hit);
        if (matcher.find()) {
            Integer value = matcher.group(1).isEmpty() ? 0 : Integer.parseInt(matcher.group(1));
            suffix.add(value);
        }
    });
    suffix.sort(Comparator.naturalOrder());
    String endValue = suffix.isEmpty() ? "" : String.valueOf(findFirstMissing(suffix));
    return joinedPrefix + endValue;
}

private Integer findFirstMissing(List<Integer> numbers) { 1 usage  ± AlexandruOlteanu
    int low = 0, high = numbers.size() - 1;
    int answer = -1;
    while (low <= high) {
        int middle = low + (high - low) / 2;
        if (numbers.get(middle).equals(middle)) {
            answer = middle;
            low = middle + 1;
            continue;
        }
        high = middle - 1;
    }
    return answer + 1;
}
```

Figura 42: Algoritmul de generare a unui username nou pe baza numelui și prenumelui

5.3 Audit al acțiunilor

Pentru a urmări acțiunile utilizatorilor și a permite scalabilitatea sistemului, am implementat o soluție bazată pe stocarea datelor în format JSON. Am creat o tabelă în care coloana folosită pentru detaliile acțiunii este de tip jsonb, astfel încât să putem stoca diverse date despre acțiuni cu posibilitate de scalare ulterioră.

Pentru ca Hibernate să poată lucra cu o astfel de coloană, am implementat un JsonNodeConverter. JsonNodeConverter este un convertor personalizat care permite Hibernate să convertească automat datele între formatul JSON și obiectele Java, facilitând astfel salvarea și extragerea datelor.

```
@Converter(autoApply = true)  ~ AlexandruOlteanu
public class JsonNodeConverter implements AttributeConverter<JsonNode, String> {

    private static final ObjectMapper objectMapper = new ObjectMapper();  2 usages

    @Override  no usages  ~ AlexandruOlteanu
    public String convertToDatabaseColumn(JsonNode attribute) {
        try {
            return attribute == null ? null : objectMapper.writeValueAsString(attribute);
        } catch (JsonProcessingException e) {
            throw new IllegalArgumentException("Error converting JsonNode to String", e);
        }
    }

    @Override  no usages  ~ AlexandruOlteanu
    public JsonNode convertToEntityAttribute(String dbData) {
        try {
            return dbData == null ? null : objectMapper.readTree(dbData);
        } catch (IOException e) {
            throw new IllegalArgumentException("Error converting String to JsonNode", e);
        }
    }
}
```

Figura 43: Implementarea unui JsonNodeConverter

Astfel, la fiecare operație de salvare sau extragere a unei acțiuni, Hibernate va folosi automat metoda corespunzătoare pentru a manipula valorile.

Odată implementată interacțiunea cu baza de date, pentru salvarea unor câmpuri în obiectul JSON final, ne folosim de funcționalitățile clasei ObjectMapper din Spring. Clasa ObjectMapper este utilizată pentru a serializa și deserializa obiectele Java în și din format JSON, oferind o modalitate simplă și eficientă de a lucra cu datele JSON.

```

Map<String, Object> actionDetailsMap = new HashMap<>();
if (request.getActionType().equals(ActionType.USER_ADD_PRODUCT)) {
    actionDetailsMap.put(PRODUCT_ID, request.getAddProductAction().getProductId());
    actionDetailsMap.put(PRODUCT_TITLE, request.getAddProductAction().getProductTitle());
    JsonNode actionDetailsNode = objectMapper.valueToTree(actionDetailsMap);
    ActionLog actionLog = ActionLog.builder()
        .userId(request.getUserId())
        .actionType(ActionType.USER_ADD_PRODUCT)
        .actionDetails(actionDetailsNode)
        .build();
    actionLogRepository.save(actionLog);
}

```

Figura 44: Exemplu de creare a unei acțiuni de adăugare de produs în platformă

Acstea acțiuni sunt categorisite după tipul acțiunii, unele fiind destinate administratorilor și altele utilizatorilor obișnuiți. Structura în care folosim un obiect JSON ca valoare în coloană ne permite să avem acțiuni structurate total diferit fără a fi nevoie să afectăm arhitectura bazei de date.

5.4 Operații Programate

O altă problemă care a necesitat adresare a fost legată în primul rând de securitatea pe care codurile generate pentru procesele de înregistrare, resetare a parolei sau schimbare de email ar trebui să o ofere. Odată trimis, codul de securitate trebuie stocat în baza de date pentru o perioadă scurtă de timp, maxim 15-30 de minute, pentru a minimiza riscul de utilizare neautorizată. O altă problemă este legată de expirarea automată a timpului de restricționare al utilizatorilor. Aceste funcționalități trebuie să aibă loc automat, astfel că am ales să folosesc operațiile programate de care beneficiază Spring-ul. Aceste operații sunt realizate cu o anotare ce primește ca parametru un regex specific de exprimare a periodicității la care să ruleze metoda automat, fără implicarea directă a utilizatorilor.

```

@Override ✉ AlexandruOlteanu
@Transactional
@Scheduled(cron = "${user.cron.security.codes.expire.scheduler}", zone = "UTC")
@Retryable(retryFor = {Exception.class}, maxAttempts = 10, backoff = @Backoff(delay = 10000))
public void securityCodesExpire() {
    log.info("Starting Security Codes Expire Cron");
    List<SecurityCode> securityCodes = securityCodeRepository.findAll();
    ZonedDateTime now = ZonedDateTime.now();
    securityCodes.forEach(securityCode -> {
        ZonedDateTime createdAt = securityCode.getCreatedAt();

        Duration duration = Duration.between(createdAt, now);
        if (duration.toSeconds() > maxSeconds) {
            securityCodeRepository.deleteById(securityCode.getId());
        }
    });
    log.info("Successfully Finished Security Codes Expire Cron");
}

```

Figura 45: Operația programată pentru ștergerea codurilor de securitate

```

@Override  ↳ AlexandruOlteanu
@Transactional
@Scheduled(cron = "${user.cron.remove.users.ban.scheduler}", zone = "UTC")
@Retryable(retryFor = {Exception.class}, maxAttempts = 10, backoff = @Backoff(delay = 30000))
public void removeUsersBan() {
    log.info("Starting Removing Users Ban Cron");
    List<User> inactiveUsers = userRepository.findAllByStatus(UserStatus.TEMPORARY_BANNED)
        .orElseThrow(() -> exceptionFactory.create(ExceptionType.USER_NOT_FOUND));
    inactiveUsers.forEach(inactiveUser -> {
        if (inactiveUser.getBanExpiryTime().isBefore(ZonedDateTime.now())) {
            inactiveUser.setStatus(UserStatus.ACTIVE);
            inactiveUser.setBanExpiryTime(null);
        }
    });
    log.info("Successfully Finished Removing Users Ban Cron");
}
}

```

Figura 46: Operația programată pentru eliminarea restricțiilor expirate

Metodele pot fi de asemenea configurate să reîncerce rularea ori de câte ori la intervalul dorit în cazul în care apar erori și procesul nu este dus la final cu ajutorul anotării @Retryable, așa cum se poate vedea în figurile anterioare.

Consider că metodele programate aduc un plus de valoare decuplării aplicației și simplificării funcționalităților de business datorită rulării independente.

5.5 Paginare și întoarcerea eficientă a rezultatelor

Pentru ca rezultatele de căutare să fie eficient propagate din baza de date până în frontend și, de asemenea, pentru a evita încărcarea excesivă a memoriei, am implementat paginarea rezultatelor pentru toate funcționalitățile aplicației ce necesită afișarea rezultatelor multiple.

Paginația reprezintă procedeul prin care, în urma unei căutări de produse, de exemplu după un cuvânt cheie, în situația în care ar trebui să primim 10.000 de rezultate în total, pentru a evita un răspuns de asemenea dimensiuni și o interacțiune ineficientă cu baza de date, se grupează acest răspuns în pagini de câte N produse, unde N poate avea o valoare variabilă, de obicei aleasă de frontend.

Obiectivul nostru este de a dezvolta o funcționalitate de backend care să poată primi numărul paginii dorite și numărul de elemente incluse într-o singură pagină. Astfel, dacă se solicită pagina 3 cu dimensiunea de 30 de produse per pagină, rezultatele incluse vor fi produsele cu indicii 61-90 din căutarea efectuată fără paginare. Această metodă asigură eficiența căutării și, când utilizatorul accesează pagina următoare, se realizează un nou apel pentru un număr relativ mic de produse.

Din fericire, deși acest proces poate părea dificil de implementat, în Spring există clasa PageRequest, care este integrată cu JpaRepository și poate fi folosită pentru a specifica pagina de date dorită, dimensiunea acesteia, dar și criteriul de ordonare după un anumit câmp din entitate.

```
Pageable pageable;
Page<ProductProjection> data;
if (sortCriteria.equalsIgnoreCase(ProductSortCriteria.RANDOM.name())) {
    pageable = PageRequest.of(chunkNumber, nrElementsPerChunk);
    data = productRepository.findAllRandom(pageable);
} else {
    pageable = PageRequest.of(chunkNumber, nrElementsPerChunk, Sort.by(getProductSortingCriteria(sortCriteria)).descending());
    data = productRepository.findAllProd(pageable);
}
```

Figura 47: Crearea și apelarea unei funcționalități folosind PageRequest

```
@Query("SELECT p.productId as productId, p.title as title, p.price as price, p.popularity as popularity FROM Product p WHERE p.userId = :userId")
Page<ProductProjection> findAllByUserId(Integer userId, Pageable pageable);
```

Figura 48: Interrogare în baza de date cu parametru de pageable

Rezultatul interogării făcute este întors sub forma unui obiect de tip Page, care, pe lângă datele rezultate din pagina curentă, întoarce și detalii despre numărul total de pagini, dacă există o pagină viitoare după cea curentă, numărul total de elemente, etc. Aceste informații sunt critice pentru a informa frontend-ul, astfel încât acesta să afișeze corespunzător în interfață numărul de pagini disponibile utilizatorului.

```
private GetProductsByCategoryResponse createGetProductsByCategoryResponse(Page<?> data,
    return GetProductsByCategoryResponse.builder()
        .products(productDTOS)
        .currentPage(data.getNumber())
        .totalPages(data.getTotalPages())
        .totalItems((int) data.getTotalElements())
        .itemsPerPage(data.getSize())
        .hasNextPage(data.hasNext())
        .hasPreviousPage(data.hasPrevious())
        .build();
}
```

Figura 49: Exemplu de accesare a datelor relevante din obiectul de tip Page

Această optimizare bazată pe paginare a fost critică în funcționarea cursivă a aplicației, în special pentru scalări viitoare cu un număr mult mai mare de produse.

5.6 Distanța Levenshtein

În funcționalitatea de căutare a produselor am inclus și asocierea rezultatelor după similitudinea termenilor de căutare introdusi de utilizator și metadatele stocate în dicționarul de căutare. Pentru a evita potrivirile perfecte între cuvinte, am integrat librăria Lucene, implementare despre care voi vorbi pe larg în secțiunea următoare. Această librărie are la bază căutarea rezultatelor cu ajutorul algoritmului de distanță Levenshtein. Acest algoritm calculează optim distanța de editare între două cuvinte, unde o acțiune de editare reprezintă ori o ștergere, ori o adăugare, ori o schimbare a unui caracter. Distanța optimă este calculată cu ajutorul programării dinamice și este definită de relația:

$$\text{lev}(a, b) = \begin{cases} |a| & \text{if } |b| = 0, \\ |b| & \text{if } |a| = 0, \\ \text{lev}(\text{tail}(a), \text{tail}(b)) & \text{if } \text{head}(a) = \text{head}(b), \\ 1 + \min \begin{cases} \text{lev}(\text{tail}(a), b) \\ \text{lev}(a, \text{tail}(b)) \\ \text{lev}(\text{tail}(a), \text{tail}(b)) \end{cases} & \text{otherwise} \end{cases}$$

Figura 50: Relația matematică pentru distanța Levenshtein[21]

O astfel de abordare asigură o relevanță crescută a rezultatelor, greșelile gramaticale fiind des întâlnite și de foarte multe ori neintenționate. Utilizând distanța Levenshtein, sistemul poate identifica și corecta aceste erori, oferind utilizatorilor rezultate pertinente chiar și atunci când există mici diferențe între termenii de căutare și cuvintele stocate în dicționar.

5.7 Integrarea librăriei Lucene

Apache Lucene este o librărie de căutare text open-source scrisă în Java, utilizată pentru a adăuga capabilități de căutare și indexare de text în aplicații software. Permite indexarea rapidă a documentelor și oferă funcționalități avansate de căutare, inclusiv căutări multiple și cu marjă pentru erori de scriere, analiză text și eliminarea cuvintelor de legătură. Lucene stă la baza unor aplicații populare precum Elasticsearch și Apache Solr, care extind funcționalitățile de bază ale Lucene pentru a oferi soluții scalabile și distribuite de căutare și analiză.

În integrarea cu această librărie a trebuit să tratez 4 operații majore:

- Indexarea unui nou document relevant cu informații despre un produs
- Actualizarea unui document corespunzător unui produs
- Ștergerea unui document din indexare
- Căutarea de produse ce include, de asemenea, paginare pentru eficiență

Pentru adăugarea unui document cu date despre un produs a trebuit să creez niște câmpuri de date printre care "product_id" pentru a reține ID-ul produsului, "metadata" pentru a stoca

datele după care se va face asocierea între interogarea primită și produs și câmpuri ce servesc la ordonarea produselor precum un câmp de popularitate și data de creare a produsului. Astfel, putem face cereri de produse după aceste criterii, având funcționalitate similară cu recomandarea de produse.

```

@Override 2 usages ▲ AlexandruOlteanu
public void addProductInSearchDictionary(Integer productId) throws IOException {
    ProductDTO productDTO = externalOperationsService.getProductById(productId);
    assert productDTO != null;
    List<String> categories = productCategorizeService.getCategoryTree(productDTO.getCategoryId()).getParentCategories().stream()
        .map(CategoryTreeValueDTO::getValue).toList();
    String categoryChain = getCategoryChain(categories);
    IndexWriterConfig config = new IndexWriterConfig(standardAnalyzer);

    try (IndexWriter indexWriter = new IndexWriter(directory, config)) {
        Document document = new Document();
        document.add(new TextField(PRODUCT_ID_KEY, String.valueOf(productId), Field.Store.YES));
        document.add(new TextField(METADATA_KEY, joinValues(productDTO.getTitle(), productDTO.getDescription(), categoryChain), Field.Store.YES));
        document.add(new LongPoint(CREATION_DATE_KEY, productDTO.getCreationDate().toInstant().toEpochMilli()));
        document.add(new StoredField(CREATION_DATE_KEY, productDTO.getCreationDate().toInstant().toEpochMilli()));
        document.add(new NumericDocValuesField(CREATION_DATE_KEY, productDTO.getCreationDate().toInstant().toEpochMilli()));
        document.add(new IntPoint(POPULARITY_KEY, productDTO.getPopularity()));
        document.add(new StoredField(POPULARITY_KEY, productDTO.getPopularity()));
        document.add(new NumericDocValuesField(POPULARITY_KEY, productDTO.getPopularity()));
        indexWriter.addDocument(document);
    }
}

```

Figura 51: Implementarea indexării datelor unui nou produs

Microserviciul de căutare apelează microserviciul produs pentru preluarea detaliilor produsului ce se dorește adăugat. Apoi, se formează o ierarhie a categoriei din care produsul face parte și, împreună cu titlul și descrierea produsului, se formează metadatele produsului. Pe lângă aceste configurații, avem și setarea unui analizator de text ce poate fi configurat să nu realizeze asocieri pe cuvinte de legătură:

```

@Bean(name = "standardAnalyzer") ▲ AlexandruOlteanu
public StandardAnalyzer getStandardAnalyzer() throws FileNotFoundException {
    CharArraySet charArraySet = new CharArraySet(DEFAULT_SIZE, ignoreCase: true);
    Pattern pattern = Pattern.compile(regex: "[\\w]+");
    File file = ResourceUtils.getFile(filePath);
    String fp = file.getAbsolutePath();
    try (BufferedReader reader = new BufferedReader(new FileReader(fp))) {
        String line;
        while ((line = reader.readLine()) != null) {
            if (line.startsWith("#")) continue;
            pattern.matcher(line).results()
                .forEach(matchResult -> charArraySet.add(matchResult.group()));
        }
    } catch (IOException e) {
        throw new RuntimeException(e);
    }
    return new StandardAnalyzer(charArraySet);
}

```

Figura 52: Configurarea analizatorului de text

Cuvintele de legătură sunt stocate într-un fișier static în interiorul microserviciului și pot fi actualizate ori de câte ori este necesar:

```
# English
but,be,with,such,then,for,no,will,not,are,and,their,if,this,on,into,a,or,there,in,that,they,was
is,it,an,the,as,at,these,by,to,of
# Romanian
acea,aceasta,aceasta,aceea,acei,aceia,acel,acela,acele,acelea,acest,acesta,aceste,acestea,acesti,
acestia,acolo,acum,ai,aia,aiba,aici,al,ala,ale,alea,altcineva,am,ar,are,aș,asadar,asemenea,
asta,asta,astazi,astea,astea,astia,asupra,atî,au,avea,avem,aveti,azi,bine,bucur,buna,ca,ca,caci,cand,
care,carei,caror,carui,cât,câte,cati,catre,catva,ce,cel,ceva,chiar,cand,cine,cineva,cat,cate,cati,catva,
contra,cu,cum,cumva,curand,curand,da,da,daca,dar,datorita,de,deci,deja,deoarece,departe,desi,din,
dinaintea,dintre,drept,dupa,ea,ei,el,ele,eram,este,estii,eu,face,fara,fi,fie,fiecare,fii,fim,fiti,
iar,ieri,ii,il,imi,impozitiva,in,inainte,inaintea,incit,incit,incotro,intre,intruat,intruat,ati,la,
langa,le,li,langa,lor,lui,ma,maine,mea,mei,mele,mereu,meu,mi,mine,mult,multa,mulți,ne,nicaieri,nici,
nimici,niste,noastră,noastre,noi,nostri,nostru,nu,ori,oricand,oricare,oricat,orice,oricand,oricine,
oricat,oricum,oriunde,pana,pe,pentru,peste,pana,poate,pot,prea,prima,primul,prin,printre,sa,sa,sai,sale,
sau,sau,se,și,sant,santem,santeti,spre,sub,sunt,suntem,sunteti,ta,tai,tale,tau,te,ti,tie,tine,toata,
toate,tot,toti,totusi,tu,un,una,unde,undeva,unei,unele,uneori,unor,va,vi,voastră,voastre,voi,vostri,
vostru,voua,vreau,vreun,si
```

Figura 53: Exemplu de fișier pentru cuvintele de legătură

Odată pregătit pentru indexare, documentul este scris și stocat în folder-ul “dictionary”, care poate fi modificat în viitor să fie în Cloud. În timpul procesului de indexare, textul din documente este tokenizat, iar token-urile sunt stocate într-un index inversat. Acest index utilizează structuri de date asemănătoare arborilor binari (B-trees) pentru a organiza și accesa eficient token-urile. Arborii binari permit inserarea rapidă de noi token-uri și căutarea eficientă a celor existente, asigurând performanțe ridicate pentru operațiile de căutare și indexare, astfel încât documentele relevante să fie găsite rapid în timpul căutărilor.

Pentru a șterge un document din indexare se caută documentul după câmpul de product_id și se face ștergerea acestuia. În cazul actualizării se apelează mai întâi ștergerea respectivului document și apoi adăugarea din nou, astfel datele despre produs sunt actualizate și valide.

```
@Override 1 usage  ± AlexandruOlteanu *
public void updateProductInSearchDictionary(Integer productId) throws IOException {
    deleteProductFromSearchDictionary(productId);
    addProductInSearchDictionary(productId);
}

@Override 2 usages  ± AlexandruOlteanu
public void deleteProductFromSearchDictionary(Integer productId) throws IOException {
    IndexWriterConfig config = new IndexWriterConfig(standardAnalyzer);
    try (IndexWriter indexWriter = new IndexWriter(directory, config)) {
        String documentId = String.valueOf(productId);
        Term term = new Term(PRODUCT_ID_KEY, documentId);
        Query query = new TermQuery(term);
        indexWriter.deleteDocuments(query);
    }
}
```

Figura 54: Actualizarea și ștergerea unui document

Pentru funcționalitatea de căutare a trebuit în primul rând să formez query-urile de căutare. Pentru a obține rezultate cât mai relevante, am ales să despart cuvintele și să fac câte o căutare pentru fiecare. Astfel, dacă doresc o căutare cu interogarea “chitară electrică”, produsul “chitară electrică” să fie afișat înaintea produsului “mașină electrică” deoarece face asociere pe 2 termeni de căutare, nu doar unul.

```
private List<String> processQuery(String query) { 1 usage  ± Alexar
    String lowerCaseQuery = query.toLowerCase();
    String[] words = lowerCaseQuery.split( regex: "[^a-zA-Z]+");
    return Arrays.asList(words);
}
```

Figura 55: Extragerea cuvintelor individuale, fără semne de punctuație

```
BooleanQuery.Builder booleanQuery = new BooleanQuery.Builder();
queryValues.forEach(currentQueryValue -> {
    Term term = new Term(METADATA_KEY, currentQueryValue);
    Query curQuery = new FuzzyQuery(term);
    booleanQuery.add(curQuery, BooleanClause.Occur.SHOULD);
});

Query finalQuery = booleanQuery.build();
Sort sort = determineSortCriteria(sortCriteria);

int start = pageNumber * pageSize;
ScoreDoc lastScoreDoc = getLastScoreDoc(searcher, finalQuery, sort, start);

TopDocs results = searcher.searchAfter(lastScoreDoc, finalQuery, pageSize, sort);

List<Integer> productScoresList = new ArrayList<>();

for (ScoreDoc scoreDoc : results.scoreDocs) {
    StoredFields storedFields = searcher.storedFields();
    Document doc = storedFields.document(scoreDoc.doc);
    Integer productId = Integer.valueOf(doc.get(PRODUCT_ID_KEY));
    productScoresList.add(productId);
}
```

Figura 56: Căutarea produselor în indexul curent

Implementarea căutării cu librăria Lucene a fost una dintre cele mai mari provocări în aria tehnică deoarece nu există o documentație actuală pentru nevoile aplicației SwapIt și a trebuit să experimentez destul de mult pentru a obține un rezultat satisfăcător. În cele din urmă, totuși, rezultatele au fost performante atât pentru actualizările indexurilor cât și pentru căutarea paginată.

5.8 Caching

Caching-ul datelor a reprezentat un alt aspect important în dezvoltarea soluției, îmbunătățind considerabil accesul la date. Cache-ul este pe două niveluri:

- Primul nivel: caching-ul făcut manual de către dezvoltator cu ajutorul anotării `@Cacheable` deasupra metodei ce se dorește a fi cache-uită. Această metodă este rar încurajată datorită necesității implicării active a dezvoltatorilor pentru a asigura validitatea datelor și scenariile în care o cheie cache ar trebui invalidată
- Al doilea nivel: cache realizat la nivel de Hibernate cu ajutorul unui tool precum Hazelcast. Acest cache se realizează automat și este la nivelul bazei de date. Atunci când o intrare este accesată după cheia primară, rămâne în cache până la restartul aplicației sau până la un update adus respectivului obiect. Hazelcast știe astfel să gestioneze update-urile și să livreze valori cache-uite valide

În aplicația SwapIt am folosit primul nivel de cache doar pentru extragerea tuturor categoriilor de produse, funcționalitate ce se schimbă foarte rar și este ușor de menținut. În rest, doar cache de nivel doi pe toate microserviciile. Pentru a putea folosi cache de nivel doi, am configurat un server local de Hazelcast:

```
@Configuration
public class HazelcastConfig {

    @Value("${hazelcast.member.ip}")
    private String memberIp;
    @Value("${hazelcast.instance.name}")
    private String hazelcastInstanceName;

    @Bean
    public Config hazelcastConfiguration() {
        Config config = new Config();

        config.setInstanceName(hazelcastInstanceName);

        NetworkConfig networkConfig = config.getNetworkConfig();
        JoinConfig joinConfig = networkConfig.getJoin();
        joinConfig.getMulticastConfig().setEnabled(false);

        TcpIpConfig tcpIpConfig = joinConfig.getTcpIpConfig();
        tcpIpConfig.setEnabled(true);
        tcpIpConfig.addMember(memberIp);
        CompactSerializationConfig compactSerializationConfig = config.getSerializationConfig().getCompactSerializationConfig();
        compactSerializationConfig.addSerializer(new ZonedDateTimeCompactSerializer());

        return config;
    }
}
```

Figura 57: Configurare server local de Hazelcast (1)

```

#####
# Cache
#####
hazelcast.instance.name=hazelcast_instance
spring.jpa.properties.hibernate.cache.use_second_level_cache=true
spring.jpa.properties.hibernate.cache.region.factory_class=com.hazelcast.hibernate.HazelcastLocalCacheRegionFactory
spring.jpa.properties.hibernate.cache.hazelcast.instance_name=${hazelcast.instance.name}
spring.jpa.properties.hibernate.show_sql=true
spring.cache.type=hazelcast
hazelcast.member.ip=localhost

```

Figura 58: Configurare server local de Hazelcast (2)

Pentru testarea cache-ului, am făcut câteva rulări din Postman pentru un set de date mai mare, atât cu cache, cât și fără, iar diferența a fost de aproximativ 2 secunde pentru 45 de mii de linii de date.

5.9 Tratarea erorilor

Tratarea erorilor corespunzător este crucială într-o aplicație web. Frontend-ul trebuie să fie informat corect în legătură cu motivul pentru care o cerere nu a fost finalizată cu succes. Spre exemplu, în funcționalitatea de înregistrare a unui utilizator poate apărea atât eroarea de email deja existent, cât și cea de username deja existent. Cum poate frontend-ul să afișeze eroarea corectă? În acest scop, am realizat un mod de creare al excepțiilor, gestionarea lor prin detalii relevante (cod de eroare, mesaj de eroare și status HTTP). Pentru a putea declara o eroare nouă și a o folosi în contextul Spring, procedăm în felul următor:

```

@getter 5 usages ± AlexandruOlteanu
public class MicroserviceException extends RuntimeException{
    private final String errorCode;
    private final String errorMessage;
    private final HttpStatus httpStatus;

    public MicroserviceException(String errorCode, String errorMessage, HttpStatus httpStatus) {
        super(errorMessage);
        this.errorCode = errorCode;
        this.errorMessage = errorMessage;
        this.httpStatus = httpStatus;
    }
}

```

Figura 59: Crearea unei clase de eroare custom ce conține detaliile de care avem nevoie

```

public enum ExceptionType { 1 usage ± AlexandruOlte

    // Security Exceptions
    UNAUTHORIZED_ACTION, no usages

    // User Exceptions
    USER_NOT_FOUND, no usages
    USERNAME_ALREADY_EXISTS, no usages
    EMAIL_ALREADY_EXISTS, no usages
    WRONG_SECURITY_CODE, no usages
    INVALID_USER_UPDATE_FIELD, no usages
    WRONG_USERNAME_OR_PASSWORD, no usages
    WRONG_PASSWORD, no usages
    USER_BANNED, no usages

    // Chat Exceptions
    CONVERSATION_NOT_FOUND, no usages

    // Product Exceptions
    PRODUCT_NOT_FOUND, no usages

    // Search Engine Exceptions
    PARENT_CATEGORY_NOT_FOUND, no usages
    PRODUCT_CATEGORY_ALREADY_EXISTS, no usages
    PRODUCT_CATEGORY_NOT_FOUND no usages
}

```

Figura 60: Crearea unei enumerări cu denumirea erorilor posibile

```

#####
# Error Codes
#####
# Security Exceptions
exception.application.unauthorized_action.code=100
exception.application.unauthorized_action.message=Unauthorized action
exception.application.unauthorized_action.status=403

# User Exceptions
exception.application.username_already_exists.code=1100
exception.application.username_already_exists.message=Username already in use
exception.application.username_already_exists.status=400

exception.application.email_already_exists.code=1101
exception.application.email_already_exists.message=An account is already associated with this email
exception.application.email_already_exists.status=400

exception.application.user_not_found.code=1102
exception.application.user_not_found.message=User not found
exception.application.user_not_found.status=400

exception.application.wrong_security_code.code=1103
exception.application.wrong_security_code.message=Wrong security code
exception.application.wrong_security_code.status=400

```

Figura 61: Declararea erorilor în fișierul de configurări

```

@Service no usages ± AlexandruOlteanu
@RequiredArgsConstructor
public class ExceptionFactory {

    private final Environment environment;

    public MicroserviceException create(ExceptionType exceptionType) { no usages ± AlexandruOlteanu
        String property = "exception.application." + exceptionType.name().toLowerCase();
        String code = environment.resolvePlaceholders( text: "${" + property + ".code}" );
        String message = environment.resolvePlaceholders( text: "${" + property + ".message}" );
        int status = Integer.parseInt(environment.resolvePlaceholders( text: "${" + property + ".status}" ));
        HttpStatus httpStatus = HttpStatus.valueOf(status);
        return new MicroserviceException(code, message, httpStatus);
    }
}

```

Figura 62: Crearea unei excepții pe baza enum-ului

Pentru a reuși să creăm o eroare custom, a trebuit să o declarăm în configurații și apoi să o extragem din contextul de Spring de fiecare dată când aruncăm în cod o excepție. Acest serviciu de erori este gestionat în librăria comună și poate fi folosit în oricare microserviciu.

Mai departe, pentru gestionarea efectivă a erorilor, a trebuit să tratez cazurile particulare în care eroarea apărea la trecerea dintr-un microserviciu în altul (eroare de Feign), dintr-un microserviciu în Api Gateway (eroare de tip RestClientResponseException) și la trecerea din Api Gateway în Frontend. Aceste treceri au fost realizate cu adăugarea codului și mesajului de eroare pe antetul de răspuns al apelurilor, fiind apoi preluat de frontend și folosit mai departe în interfața grafică.

Pentru o detaliere a implementării acestor propagări ale erorilor în funcție de caz, se poate vedea Anexa A6.

6 EVALUAREA REZULTATELOR

6.1 Corectitudinea Aplicației

Pentru a mă asigura că aplicația funcționează într-un mod corect, am ales să realizez o serie de teste manuale folosind o aplicație destinață lucrului cu apelurile HTTP/S precum Postman[8]. Consider că aceste teste au fost esențiale pentru a verifica dacă fiecare funcționalitate procesează datele corect și corespunde specificațiilor de business dorite. Testarea s-a desfășurat pe măsură ce fiecare funcționalitate nouă a fost adăugată, precum și integrarea cu alte funcționalități asociate ce pot să impacteze rezultatele. Acest proces a permis identificarea rapidă a eventualelor erori de business sau tehnice și remedierea acestora încă din fazele incipiente ale dezvoltării. În final, după realizarea frontend-ului, am testat multiple funcționalități pentru a asigura o experiență completă și performantă pentru utilizatori.

Un aspect crucial al corectitudinii aplicației îl constituie validarea datelor. După implementarea fiecărei funcționalități care implică introducerea sau manipularea datelor, am efectuat teste pentru a verifica dacă datele sunt validate corespunzător de sistem. Am trimis seturi de date invalide pentru a observa cum gestionează aplicația aceste situații. De asemenea, am testat limitele de validare, cum ar fi dimensiunile maxime ale câmpurilor și formatele acceptate, pentru a mă asigura că aplicația oferă mesaje de eroare clare și utile în caz de date incorecte.

De asemenea, timpul de răspuns este un indicator important al corectitudinii și eficienței aplicației. După implementarea fiecărei funcționalități care implică operațiuni ce pot afecta performanța, am măsurat durata necesară pentru procesarea cererilor și întoarcerea răspunsurilor. Analiza timpilor de răspuns mi-a permis să realizez ajustările necesare pentru optimizarea aplicației și pentru a asigura un timp de răspuns rapid și constant.

În final, am testat aplicația cu utilizatori mulți: utilizatori obișnuiți, administratori, dar și utilizatori înregistrați OAuth2. Am validat funcționalitățile permise doar unei anumite categorii de utilizatori și per total cum aplicația se comportă cu peste 40 de utilizatori și 60.000+ produse adăugate.

The screenshot shows the Postman interface. On the left, there's a sidebar with a tree view of API endpoints under 'Product'. The main area shows a 'GET' request to 'http://localhost:8001/api/v1/swapt/apiGateway/publicAccess/getRecommendedProducts?chunkNumber=0&nrElementsPerChunk=30&sortCriteria=newest'. Under 'Params', there are three checked items: 'chunkNumber' (Value: 0), 'nrElementsPerChunk' (Value: 30), and 'sortCriteria' (Value: newest). Below this is a 'Response' section.

Figura 63: Exemplu mediu de testare utilizând Postman

6.2 Performanța Aplicației

Pentru a testa corespunzător aplicația, am populat baza de date cu un set amplu de date de test, incluzând 46 de utilizatori și peste 60.000 de produse. Acest demers a avut ca scop evaluarea performanței aplicației într-un mediu cu volum mare de date și utilizatori, asigurând astfel validitatea și fiabilitatea funcționalităților în condiții similare celor dintr-un scenariu real de utilizare. Am ales să analizez aplicația prin două metode: folosind JMeter [9] și Lighthouse [10] oferit de browserele web.

6.2.1 Analiza performanței folosind Lighthouse

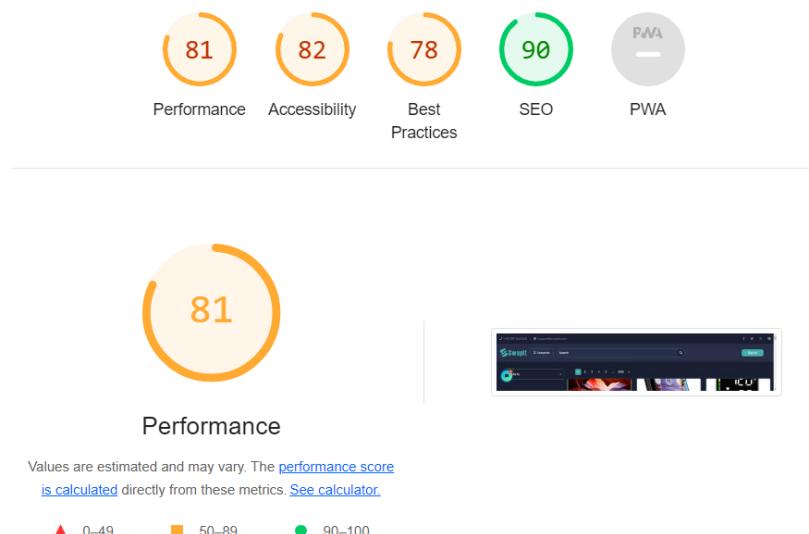


Figura 64: Rezultatele performanței pe pagina acasă

Se poate observa că scorul obținut pe pagina principală depășește pragul de 80, considerat un scor ce tinde spre limita superioară. Este important de menționat că pagina principală a aplicației include una dintre cele mai complexe funcționalități, și anume încărcarea produselor recomandate. Astfel, rezultatul obținut este unul deosebit de bun. În continuare am realizat teste similare pe cele mai importante pagini după cum urmează:

Pagina	Performanță	Accesibilitate	Cele mai bune practici	SEO
Acasă	81	82	78	90
Profil utilizator	87	82	78	90
Vizualizare produs	83	81	78	90
Căutare produse	78	90	78	90
Adăugare produs	83	85	78	90
Modificare produs	84	84	78	90
Logistică Admin	85	84	78	90
Vizualizare produse pe categorie	80	82	78	90

Tabel 1: Analiza paginilor web folosind Lighthouse

Pagina de profil a utilizatorului se remarcă prin cel mai înalt scor de performanță, un rezultat foarte bun având în vedere că majoritatea schimbărilor de date, vizualizarea produselor favorite sau adăugate de utilizator sunt realizate din această pagină.

De asemenea, alte pagini importante precum pagina de produs și secțiunile de adăugare și modificare a produselor au înregistrat scoruri solide de performanță între 83 și 84, demonstrând o consistență în calitatea și rapiditatea încărcării. Pagina de logistică prezintă de asemenea scorul de 85, oferind astfel un mediu performant pentru administratori de a monitoriza acțiunile realizate în cadrul platformei.

Căutarea produselor, deși a înregistrat cel mai scăzut scor de performanță egal cu 78, rămâne în limite acceptabile, având în vedere complexitatea logicii de business implicate. Aceasta sugerează că, deși este o zonă ce necesită atenție suplimentară pentru îmbunătățire, performanța generală este încă bună.

În concluzie, aplicația prezintă un nivel ridicat de performanță pe toate paginile esențiale, asigurând o navigare rapidă și funcționalități eficiente. Aceste rezultate reflectă o optimizare reușită și o implementare eficientă a bunelor practici de dezvoltare, contribuind la o experiență de utilizare plăcută și intuitivă.

6.2.2 Analiza performanței folosind JMeter

Am utilizat JMeter pentru a realiza teste de stres asupra aplicației, evaluându-i performanța sub sarcină intensă. Am realizat multiple apele concurente pentru a simula condiții reale de utilizare și a analiza cum se comportă aplicația sub presiune. Aplicația fiind deocamdată un MVP și, de asemenea, pentru că rulează local, am decis să testeze câteva funcționalități cu 100 de utilizatori concurenți.

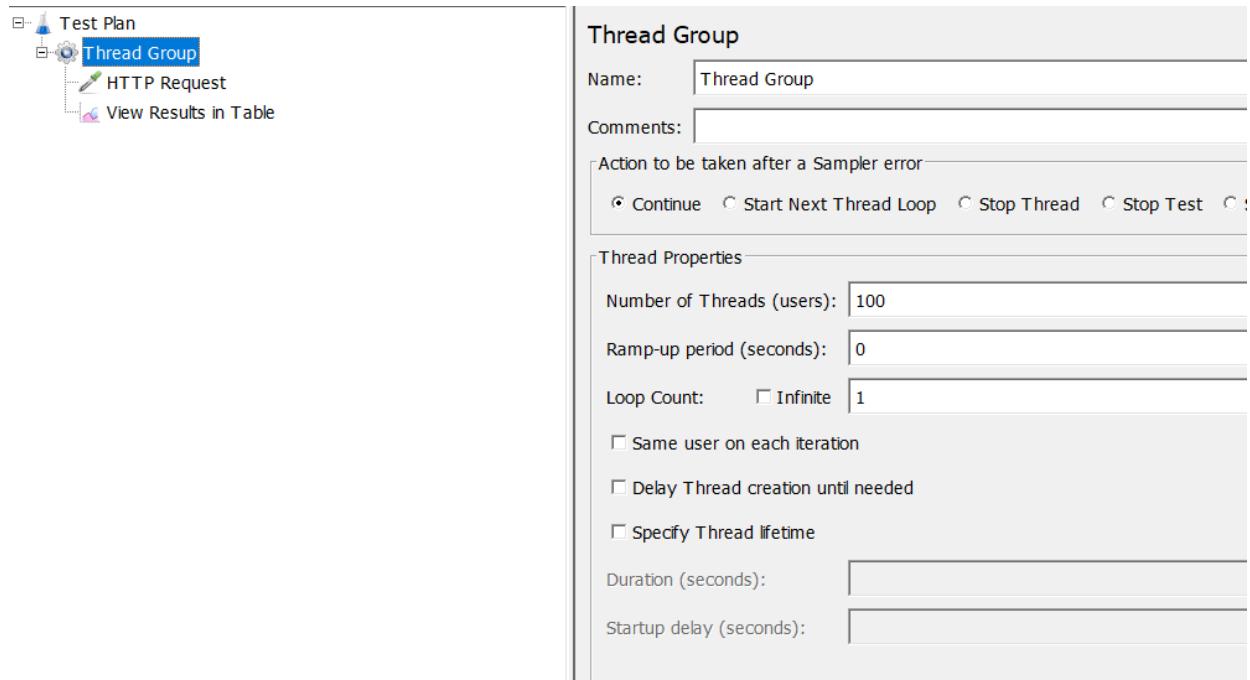


Figura 65: Setarea numărului de threaduri concurente

The screenshot shows the 'Basic' tab of the JMeter 'HTTP Request' configuration dialog. It includes fields for 'Web Server': 'Protocol [http]:' set to 'http', 'Server Name or IP:' set to 'localhost', and 'Port Number:' set to '8001'. Under 'HTTP Request', the method is set to 'GET' and the path is '/api/v1/swapiIt/apiGateway/publicAccess/getProductById?productId=1'. Other options include 'Content encoding:', 'Follow Redirects' (checked), 'Use KeepAlive' (checked), 'Use multipart/form-data' (unchecked), and 'Browser-compatible headers' (unchecked). At the bottom, there are tabs for 'Parameters', 'Body Data', and 'Files Upload', and a section for 'Send Parameters With the Request' with columns for 'Name', 'Value', 'URL Enc...', 'Content-Type', and 'Include Equ...'. There is also a note: 'Name: Value URL Enc... Content-Type Include Equ...'.

Figura 66: Configurarea api-ului apelat

Sample #	Start Time	Thread Name	Label	Sample Time(ms)	Status	Bytes	Sent Bytes	Latency /	Connect Time(ms)
1	00:01:12.534	Thread Group 1-2	HTTP Request	76	✓	1195	183	76	2
2	00:01:12.535	Thread Group 1-5	HTTP Request	79	✓	1195	183	78	1
3	00:01:12.536	Thread Group 1-9	HTTP Request	80	✓	1195	183	80	0
4	00:01:12.534	Thread Group 1-3	HTTP Request	87	✓	1195	183	86	2
5	00:01:12.535	Thread Group 1-6	HTTP Request	88	✓	1195	183	87	0
6	00:01:12.534	Thread Group 1-1	HTTP Request	99	✓	1195	183	99	2
7	00:01:12.536	Thread Group 1-12	HTTP Request	99	✓	1195	183	99	0
8	00:01:12.537	Thread Group 1-13	HTTP Request	100	✓	1195	183	100	1
9	00:01:12.535	Thread Group 1-7	HTTP Request	114	✓	1195	183	114	0
10	00:01:12.536	Thread Group 1-10	HTTP Request	124	✓	1195	183	123	1
11	00:01:12.539	Thread Group 1-35	HTTP Request	125	✓	1195	183	125	0
12	00:01:12.537	Thread Group 1-24	HTTP Request	131	✓	1195	183	127	1
14	00:01:12.541	Thread Group 1-28	HTTP Request	141	✓	1195	183	141	0
13	00:01:12.539	Thread Group 1-4	HTTP Request	142	✓	1195	183	142	1
15	00:01:12.539	Thread Group 1-14	HTTP Request	144	✓	1195	183	144	1
16	00:01:12.541	Thread Group 1-19	HTTP Request	144	✓	1195	183	144	0
17	00:01:12.539	Thread Group 1-40	HTTP Request	153	✓	1195	183	153	1

Figura 67: Rezultatele apelurilor făcute

Rezultatele obținute pe cele mai importante funcționalități sunt următoarele:

API	Latență	Număr apeluri terminate cu succes	Număr apeluri terminate cu eroare
/publicAccess/searchProducts	412 ms	100	0
/publicAccess/getUserDetails	295 ms	100	0
/publicAccess/getRecommendedProducts	571 ms	100	0
/publicAccess/getProductById	362 ms	100	0
/publicAccess/searchProductsByCategory	632 ms	100	0
adminAction/getUserActions	314 ms	100	0

Tabel 2: Rezultatele de performanță folosind Jmeter

Rezultatele obținute pentru cele mai importante funcționalități arată o rată de succes de 100% pentru toate cele 100 de apeluri efectuate pentru fiecare API în parte, ceea ce sugerează că aplicația nu are probleme de interacțiune concurrentă cu baza de date sau limite de gestionare concurrentă. Latențele variază între 295 ms pentru vizualizarea profilului unui utilizator și 632 ms pentru căutarea produselor pe categorii. Acest lucru demonstrează că aplicația este stabilă și performantă în condiții de testare cu 100 de apeluri concurente.

Pentru a asigura o scalare eficientă și a menține performanța la niveluri ridicate pe măsură ce volumul de utilizatori și numărul de cereri crește, va trebui să distribui aplicația pe mai multe noduri. De asemenea, va trebui să se acorde o atenție specială configurării garbage collector-ului pentru a preveni potențialele probleme de memorie și pentru a optimiza timpii de răspuns. Astfel, monitorizarea și ajustarea continuă a resurselor vor fi cruciale pentru menținerea performanței și fiabilității aplicației în mediul de producție.

6.3 Experiența utilizatorilor & Design-ul interfeței

6.3.1 Pagina Acasă

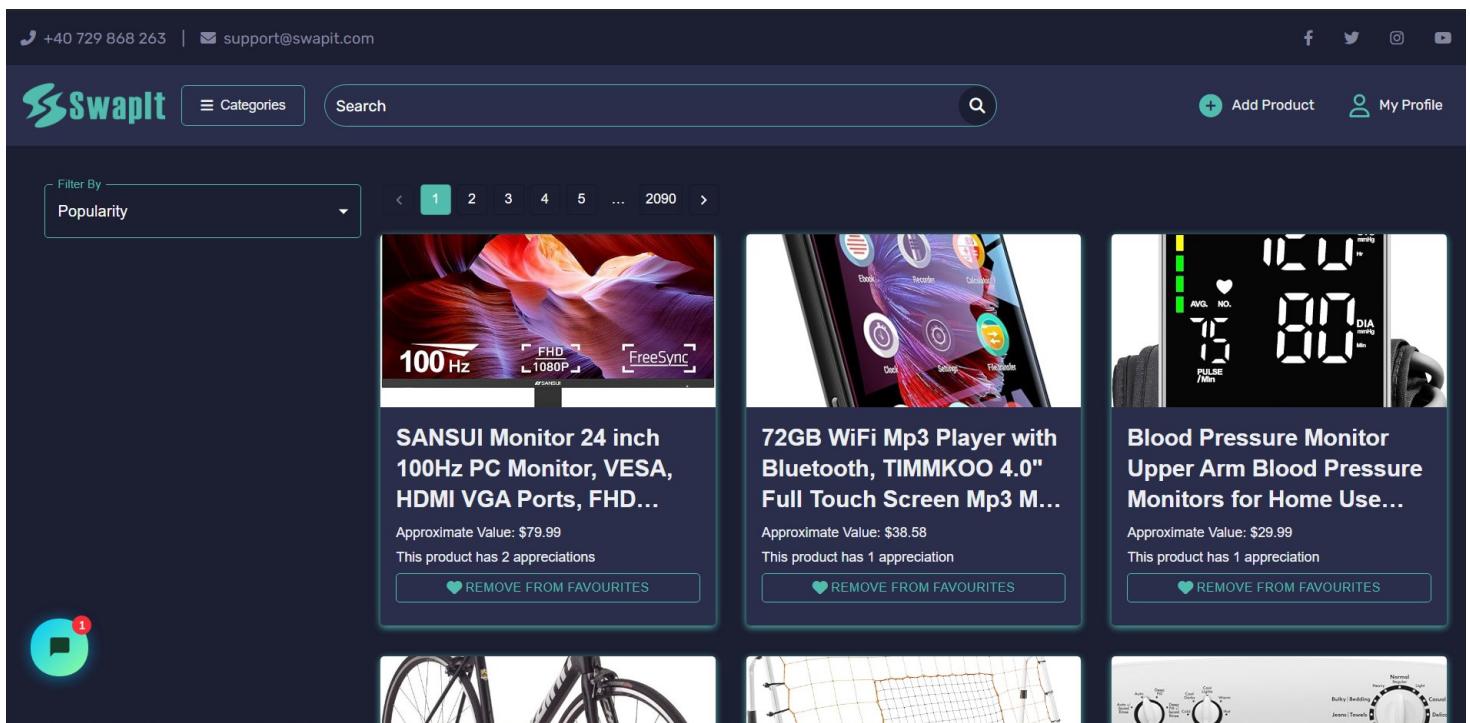


Figura 68: Design-ul paginii acasă din perspectiva unui utilizator autentificat

Pagina principală a site-ului cuprinde în special produse recomandate care sunt ordonate după popularitate. Design-ul este modern și conține toate funcționalitățile de bază de care are nevoie utilizatorul. În partea de sus avem detalii cu privire la numărul de telefon și mail- ul pentru suport și, de asemenea, în partea dreaptă link-uri către platformele sociale ale platformei SwapIt, aceste elemente încercând să inducă încredere utilizatorilor. Bara de navigare cuprinde emblema platformei, un buton cu categoriile de produse prezente, căutarea după cuvinte cheie și, în dreapta, posibilitatea de a adăuga un produs sau de vizualizare a produsului. Produsele sunt organizate în 3 coloane, pe pagini, paginarea fiind accesibilă atât în partea de sus cât și în cea de jos a listării. Acestea au buton de adăugare la favorite și detaliile de bază precum poza, valoarea aproximativă și numărul de aprecieri.

De asemenea, un alt element distinctiv este butonul de chatBot din stânga jos, de unde utilizatorul poate lua contact direct cu suportul în legătură cu diverse probleme.

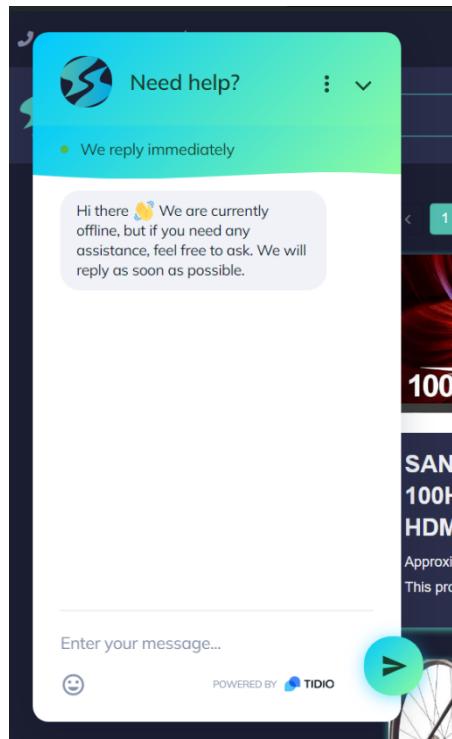


Figura 69: Fereastra de discuție cu echipa de suport

Produsele recomandate pot fi ordonate după mai multe criterii: popularitate, cele mai noi adăugate și random.

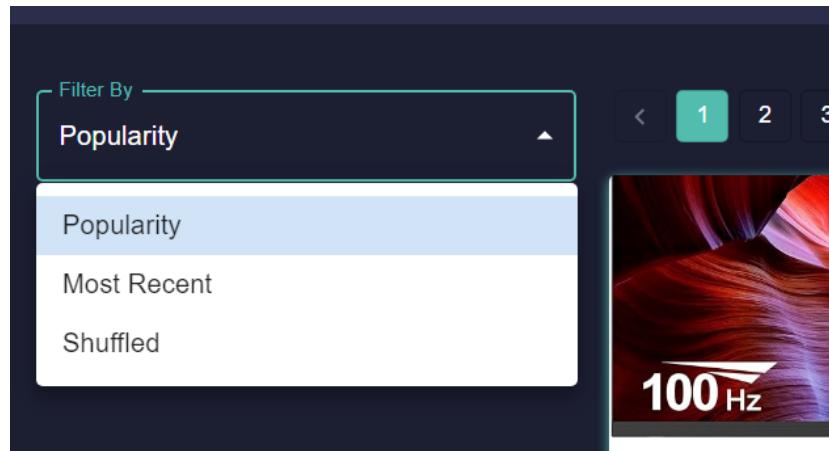


Figura 70: Metodele de sortare a produselor

Toate metodele de sortare sunt la fel de eficiente, astfel că utilizatorii vor beneficia de rezultate prompte.

6.3.2 Pagina de Autentificare

În cazul în care utilizatorul nu este autentificat, în locul butonului de adăugare produs și cel de vizualizare al profilului apare un buton de autentificare. Formularul de autentificare conține două câmpuri, unul de username și altul pentru parolă. În cazul în care utilizatorul introduce greșit datele, va fi întâmpinat cu un mesaj de eroare. Acesta are de asemenea posibilitatea să se autentifice cu Google.

The screenshot shows a 'Login' page with a dark header and footer. The main area has a white background. It contains two input fields: one for 'john_doe' and another for a password represented by dots. Below these is a teal 'Login' button. To the right of the button are links for 'Forgot password?' and 'Don't have an account? Signup'. At the bottom is a 'Login with Google' button featuring the Google logo.

Figura 71: Pagina de autentificare

This screenshot is identical to Figure 71, but the password field now contains a red error message: 'Wrong username or password'. The rest of the interface, including the teal 'Login' button and other links, remains the same.

Figura 72: Mesaj de eroare în cazul autentificării greșite

Dacă utilizatorul a uitat parola, o poate resetă făcând click pe link-ul de „Forgot Password”, de unde trebuie să își introducă email-ul, după care va primi un cod de securitate și va trebui să îl introducă pentru validare și resetarea parolei.

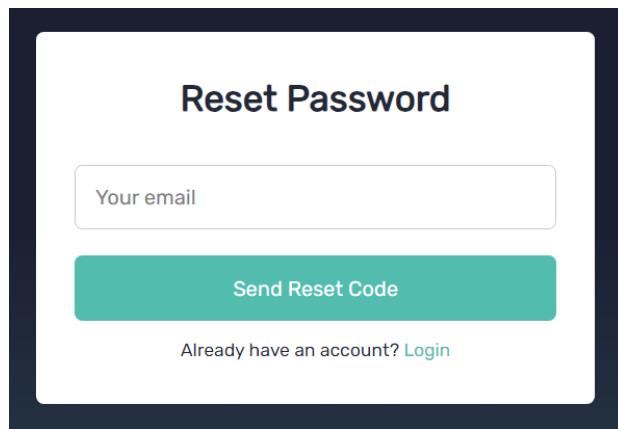


Figura 73: Formular pentru resetarea parolei

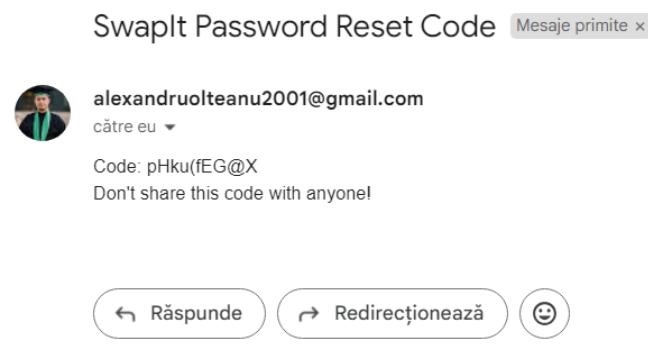


Figura 74: Email de resetare al parolei

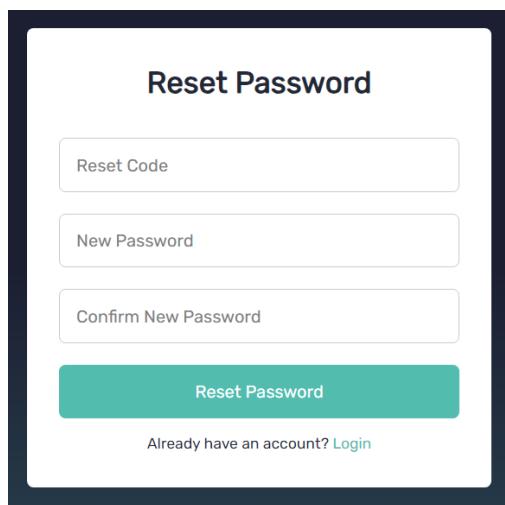


Figura 75: Ultimul pas în procesul de resetare al parolei

Pentru înregistrarea în platformă, utilizatorul poate folosi în continuare autentificarea cu Google sau să completeze formularul de înregistrare compus din mai multe câmpuri: username, nume, prenume, email, parolă și confirmare parolă. În cazul în care utilizatorul introduce un email sau username deja existent, parolele nu coincid sau lasă câmpuri libere, va fi atenționat printr-o eroare corespunzătoare la fel ca la pagina de autentificare.

The image shows a mobile-style sign-up form titled "Signup". The form consists of six input fields arranged vertically: "Username", "Name", "Surname", "Email", "Create password", and "Confirm password". Each field has a placeholder text inside it. Below the fields is a large teal-colored button labeled "Sign Up". At the very bottom of the form, there is a small link that reads "Already have an account? [Login](#)". The entire form is set against a dark background.

Figura 76: Pagina de înregistrare

Consider că pagina de înregistrare și autentificare este proiectată pentru a fi simplă și intuitivă, asigurând utilizatorilor o experiență fluidă. Designul este modern, caracterizat de elemente minimalistă, fiind completat de un contrast evident între culorile folosite, facilitând navigarea și interacțiunea. Utilizatorii pot parcurge ușor etapele de autentificare, resetare a parolei și înregistrare, datorită unei structuri clare și bine organizate a paginii..

6.3.3 Profilul Utilizatorului

Pagina de profil a utilizatorului este realizată modern și oferă multe informații comparativ cu paginile clasice de profil. În momentul în care utilizatorul apasă pe butonul de profil, este întâmpinat de o organizare pe 3 coloane. Prima coloană oferă detalii precum poza de profil (pe care o poate schimba dând click pe aceasta), data înregistrării și alte detalii de bază cum ar fi numărul de telefon, numele, adresa de email, etc. Toate aceste detalii de bază pot fi actualizate din a doua secțiune din dreapta, care se schimbă dacă selectăm schimbarea parolei, a adresei de email sau a username-ului. Ultima coloană conține, în mod implicit, produsele adăugate de utilizator sau lista de produse favorite, dacă selectăm această opțiune din panoul din stânga. În cazul profilului unui administrator, acesta nu are produse favorite sau adăugate, în schimb are o secțiune de audit a acțiunilor realizate în cadrul platformei, atât

de alți administratori (colorate cu roșu), cât și ale altor utilizatori. Dacă dorim să vedem profilul altui utilizator, vom putea vedea doar produsele adăugate de acesta și câteva date de bază.

The screenshot shows the SwapIt user profile page for a user named Amelia White (@amelia_white). The profile includes a photo, the user's name, member since date (5/31/2024), contact information (phone number +61-7-3403-8888, email amelia.white@example.com), and a list of products (Car Jump Starter with Built-In Flashlight and USB Ports, Water-Resistant Hiking Backpack with Multiple Compartments).

Figura 77: Pagina de profil a utilizatorului

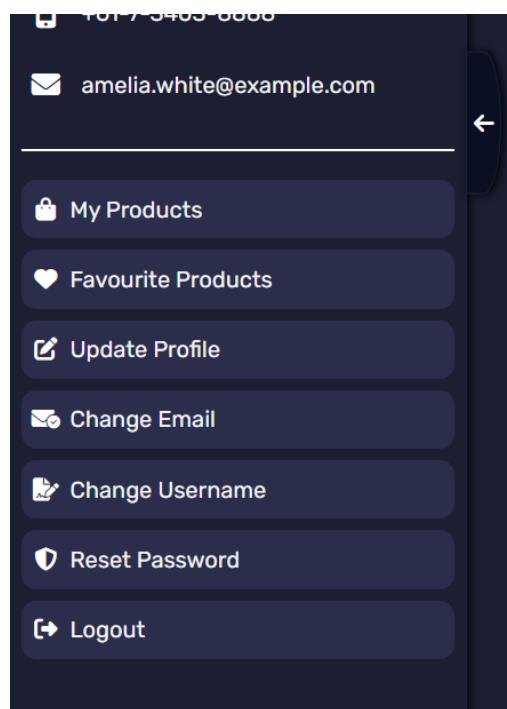


Figura 78: Panoul de acțiuni disponibile utilizatorului



Figura 79: Utilizatorul nu are produse favorite

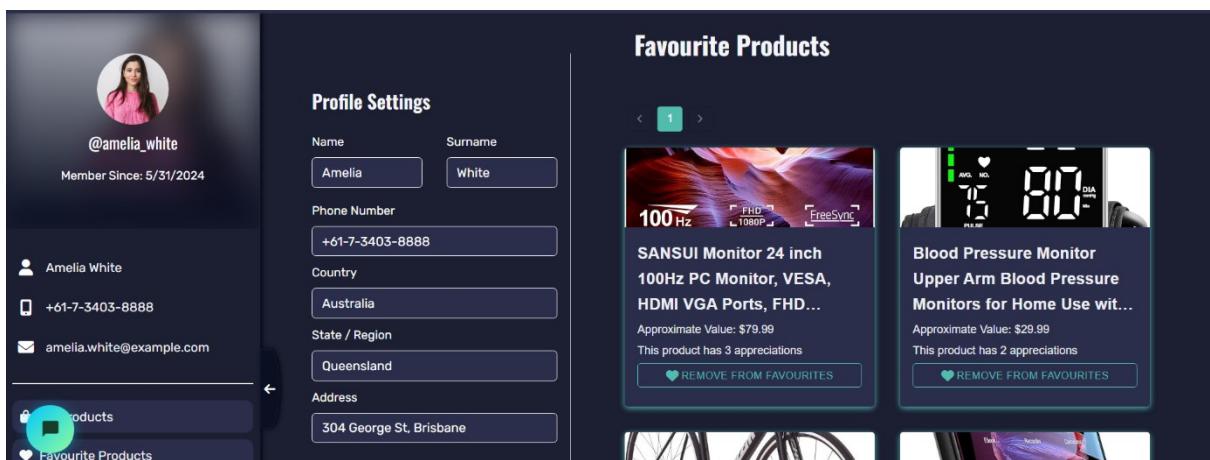


Figura 80: Utilizatorul are produse favorite

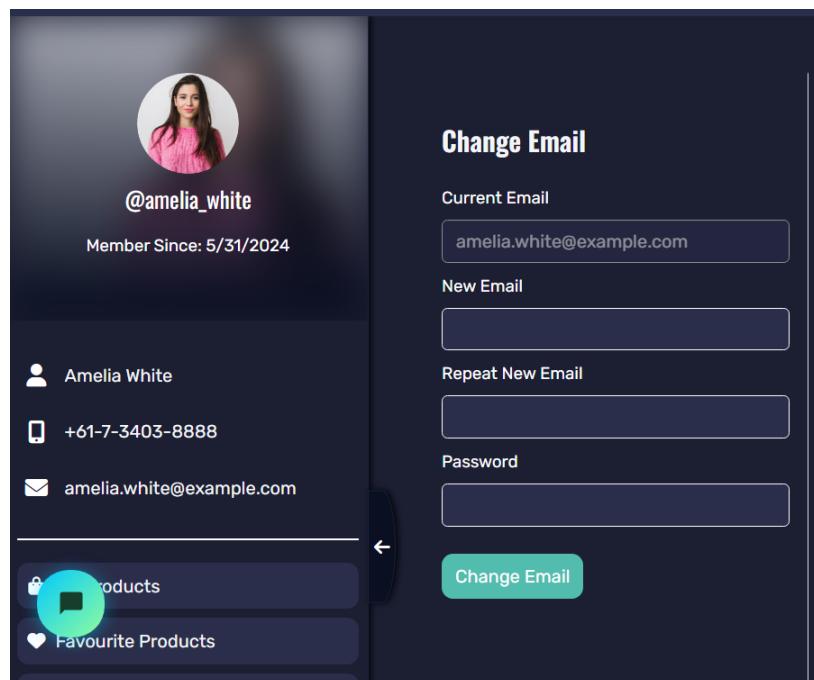


Figura 81: Secțiune de schimbare a email-ului

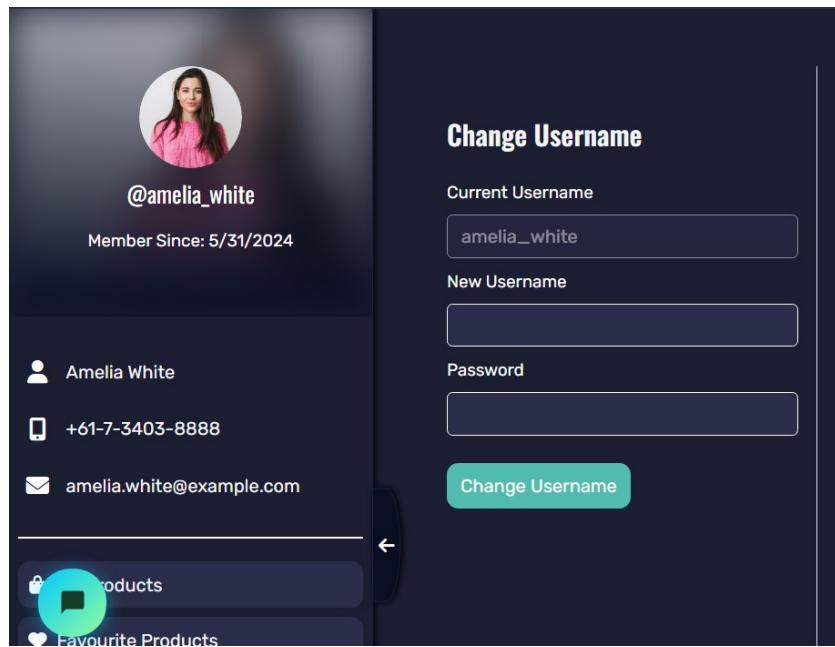


Figura 82: Secțiune de schimbare username

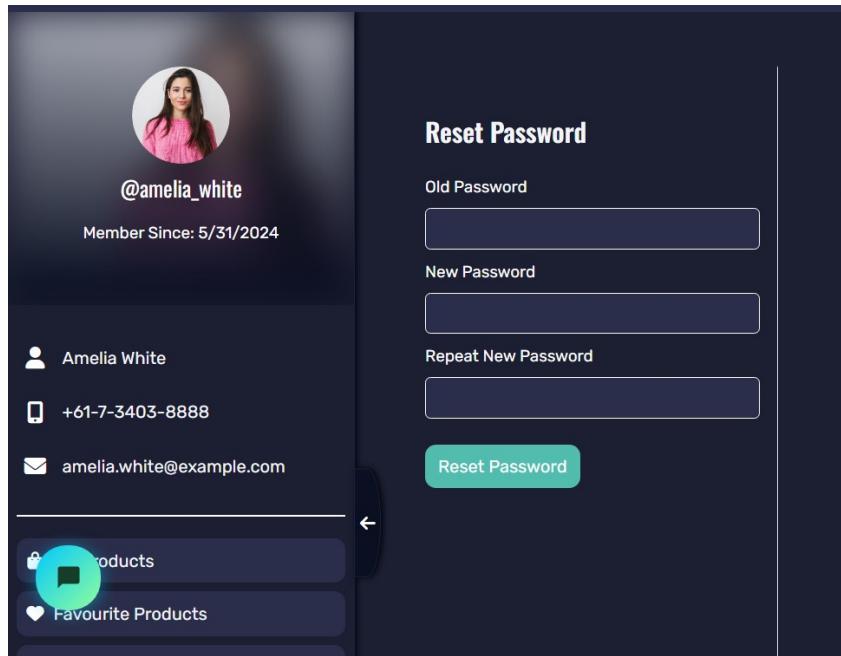


Figura 83: Secțiune de resetare a parolei

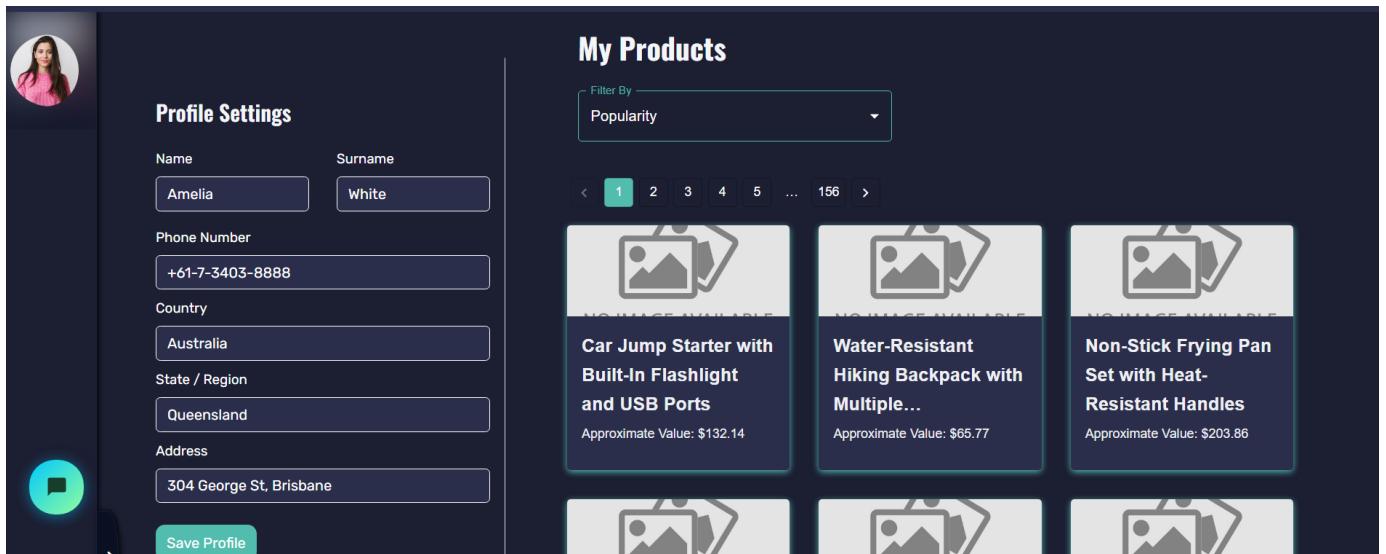


Figura 84: Profilul utilizatorului cu panoul de setări minimizat

6.3.4 Acțiunile Administratorului

Administratorii au funcționalități adiționale comparativ cu utilizatorii obișnuiți. Aceștia pot să șteargă produsul oricărui utilizator dacă îl consideră neadecvat, pot să restricționeze accesul pe o perioadă determinată sau permanent (și să revoce această restricție) și de asemenea au acces la istoricul acțiunilor relevante realizate în cadrul platformei. Rolul de administrator fiind unul important, un astfel de utilizator are în permanență o atenționare în partea de sus a ecranului care îi amintește că are drepturi de administrare și să aibă grijă cu acțiunile pe care le întreprinde.

The screenshot shows an administrator profile page. At the top, there's a notice: '⚠️ Notice: You have administrator privileges. Please exercise caution in all actions taken on the platform.' Below this are contact details: phone number +40 729 868 263 and email support@swapit.com. The header includes a 'Categories' menu, a search bar, and a 'My Profile' link. The main content area has two sections: 'Profile Settings' (with fields for Name (John), Surname (Doe), Phone Number (+1-305-555-3456), Country (USA), State / Region (Florida), and Address (101 Pine St, Miami, FL)) and 'Website Actions Log' (filtered by 'Include All'). The log shows two recent actions: one where @john_doe removed a ban from user @logan_clark on Jun 4, 2024, at 08:56 AM, and another where @john_doe banned user @logan_clark with a 5-day duration on Jun 4, 2024, at 08:55 AM.

Figura 85: Pagina de profil a administratorului

Acțiunile pot fi filtrate după următoarele criterii: toate acțiunile, acțiunile administrative sau acțiunile utilizatorilor. Aceste notificări sunt interactive, permitând accesarea noilor produse adăugate sau a profilului celor ce au realizat o anumită acțiune.

The screenshot shows a dark-themed interface titled "Website Actions Log". At the top, there is a dropdown menu labeled "Filter By" with "Users Actions" selected. Below the filter are page navigation buttons (1, 2, 3, 4, 5, ..., 15673). The main area displays two user actions in cards:

- @noah_king added a new product**
Product Title: Water-Resistant Hiking Backpack with Multiple Compartments
Jun 3, 2024, 08:28 PM
- @aiden_king added a new product**
Product Title: Blood Pressure Monitor Upper Arm Blood Pressure Monitors for Home Use with 2x120 Reading Memory Adjustable Arm Cuff 8.7"-15.7" LED Background Light Large Display Machine with...
Jun 3, 2024, 08:28 PM

Figura 86: Filtrarea acțiunilor utilizatorilor

The screenshot shows a user profile card for "@noah_king". It includes the following information:

- User icon: A placeholder person icon.
- Username: @noah_king
- Member Since: 5/31/2024
- Contact details:
 - Profile picture: Placeholder image.
 - Name: Noah King
 - Phone: +61-7-3403-8888
 - Email: noah.king@example.com
- Product section: "King's Products" (with a lock icon).
- Action button: "Ban This User" (with a crossed-out user icon).

Figura 87: Vizualizarea profilului altui utilizator de către administrator

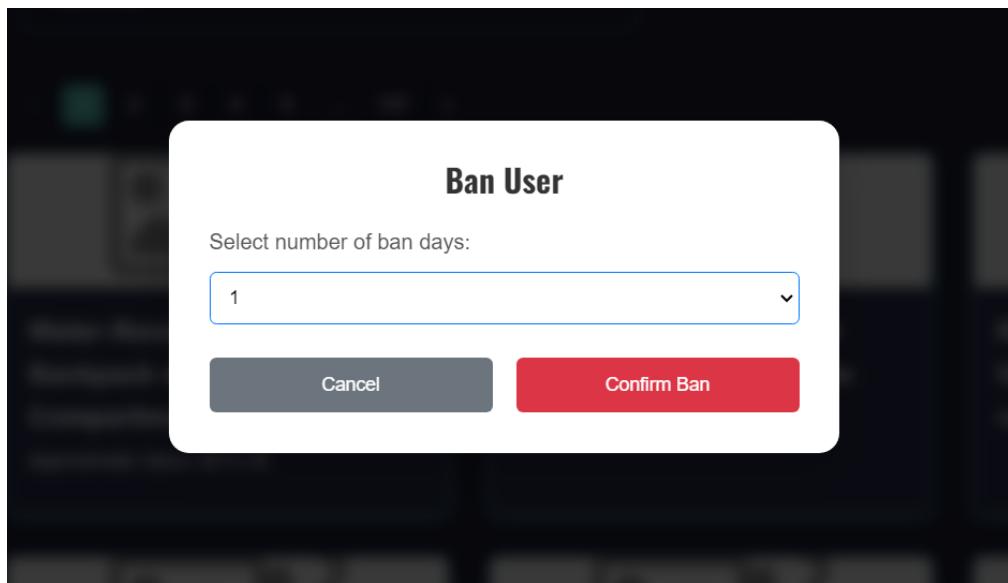


Figura 88: Dialogul de restricționare al utilizatorului

6.3.5 Crearea si Modificarea Unui Produs

Crearea unui produs se realizează de la butonul de adăugare produs, utilizatorul fiind dus pe o pagină unde își cere să adauge titlul, descrierea, specificații despre produs, să aleagă categoria și subcategoria din care acest produs face parte, valoarea aproximativă și imagini pentru produs. Dacă utilizatorul se răzgândește cu privire la specificații sau imagini, le poate șterge sau adăuga altele noi. Pentru modificarea produsului îi este afișată o pagină similară doar că având datele produsului precompletate, gata să fie modificate.

A screenshot of a "Create New Product" form. The title is "Create New Product". It has fields for "Title *", "Description *", and "Product Specifications". Under "Product Specifications", there are fields for "Specification *" and "Value *", with a "Value Reference *" field below. A green button labeled "Add Product Specification" is visible. The background is dark blue.

Figura 89: Pagina de creare produs

Update Product

Title *
Blood Pressure Monitor Upper Arm Blood Pressure Monitors for Home Use with 2x120 Reading Memory Adjustable Arm Cuff 8.7"-15.7" LE

Description *
The Blood Pressure Monitor Upper Arm is an ideal choice for home use, providing accurate and reliable readings. Designed with convenience and user-friendliness in mind, this monitor features a large LED background light display, making it easy to read results even in low light conditions. The adjustable arm cuff fits a range of sizes from 8.7" to 15.7", ensuring a comfortable and secure fit for all users. With a memory capacity of 2x120 readings, it allows for long-term tracking and comparison of blood pressure data for two users.

Product Specifications

Specification * Product Type	Value * Blood Pressure Monitor
Specification * Usage	Value * Upper Arm
Specification * Memory Capacity	Value * 2x120 Readings
Specification * Cuff Size	Value * Adjustable, 8.7" - 15.7"

Figura 90: Pagina de modificare a unui produs (1)

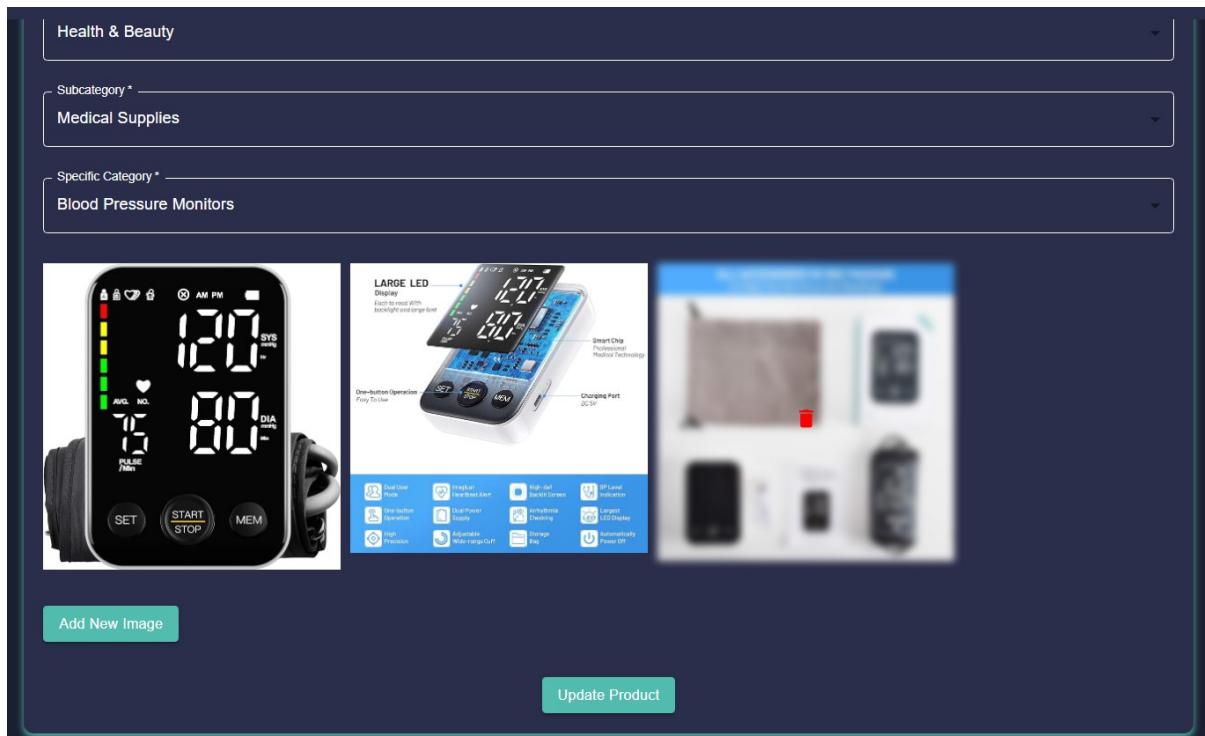


Figura 91: Pagina de modificare a unui produs (2)

6.3.6 Pagina de Produs

Pagina de produs dispune de un design modern, asemănător cu cel al platformelor pe care le utilizăm zi de zi. Odată aflat pe pagina unui produs, utilizatorul poate vedea imaginile, titlul și descrierea acestuia. Poate trece de la o imagine la alta apăsând pe varianta micșorată a acestora din partea din stânga jos sau cu ajutorul săgeților dispuse de o parte și de alta a imaginii principale. Dacă produsul îi aparține utilizatorului, acesta are buton de modificare produs sau de ștergere produs. Dacă nu, poate adăuga sau șterge produsul de la favorite. În partea de sus poate vedea arborele ierarhic al categoriei din care produsul face parte, iar în partea de jos poate citi specificațiile sale.

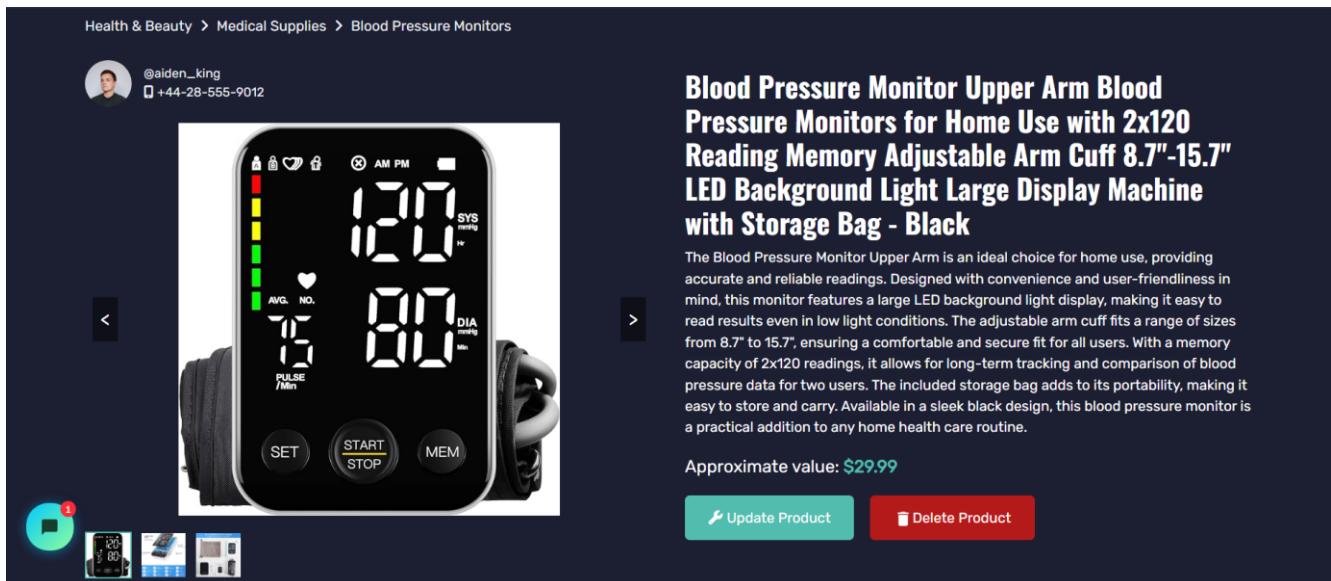


Figura 92: Pagina de produs

A screenshot of the product specifications table for the blood pressure monitor. The table has four rows:

- Product Type:** Blood Pressure Monitor
- Usage:** Upper Arm
- Memory Capacity:** 2x120 Readings
- Cuff Size:** Adjustable, 8.7" - 15.7"

At the top of the table, there's a large image of the blood pressure monitor showing its digital display and control buttons. To the right of the table is the approximate value '\$29.99' and a 'Update Product' button. At the very bottom of the table is a small red circular icon with the number '1'.

Figura 93: Specificațiile produsului:

6.3.7 Categoriile de Produse

Utilizatorii pot vizualiza toate categoriile produselor ținând cursorul deasupra butonului de categorie din componenta barei de navigare. De aici, aceștia pot selecta orice categorie doresc, după care vor fi trimiși către o pagină cu toate produsele găsite sub acea categorie, precum și în subcategoriile copil. Aceeași funcționalitate este obținută și la apăsarea unei categorii din ierarhia categoriilor de pe pagina unui produs.

The screenshot shows the SwapIt mobile application's interface. At the top, there is a header with the SwapIt logo, a 'Categories' button, a search bar, and a magnifying glass icon. On the left, there is a sidebar titled 'Filter By Popularity' with a list of categories: Electronics & Appliances, Computers & Accessories, Home & Garden, Sports & Outdoors (which is highlighted in teal), Fashion & Accessories, Health & Beauty, Toys & Games, Automotive, Books & Audible, and Music & Instruments. The main content area displays a grid of categories under several headings: Fitness Equipment, Camping & Hiking, Cycling, Water Sports, Team Sports, Outdoor Clothing, Fishing, Soccer, Basketball, Baseball, Volleyball, Hockey, Treadmills, Exercise Bikes, Dumbbells, Yoga Mats, Resistance Bands, Tents, Sleeping Bags, Backpacks, Camping Furniture, Camping Cookware, Mountain Bikes, Road Bikes, Electric Bikes, Bike Helmets, Bike Accessories, Kayaks, Paddleboards, Life Jackets, Swim Goggles, Snorkeling Gear, Jackets, Hiking Boots, Outdoor Pants, Base Layers, Gloves, Fishing Rods, Reels, Bait & Lures, Fishing Line, and Fishing Accessories.

Figura 94: Exemplu navigare categorii de produse

7 CONCLUZII

Consider că aplicația SwapIt reușește să își atingă obiectivele, oferind o soluție practică și necesară pentru reducerea risipei și promovarea sustenabilității. Într-un context global marcat de preocupări pentru mediu și resurse, SwapIt facilitează schimbul de obiecte, transformând bunurile nefolosite în resurse valoroase pentru alții. Aceasta nu doar că ajută utilizatorii să economisească bani, dar și contribuie semnificativ la reducerea deșeurilor, susținând astfel economia circulară.

Am încercat în special să rezolv problema încrederei într-o astfel de platformă, motiv pentru care am integrat funcționalitatea de ChatBot, vizualizarea profilelor utilizatorilor și un sistem de administrare bine pus la punct. În urma unor dezvoltări viitoare, unul dintre principalele obiective va fi să asigur un mediu sigur de tranzacționare și de discuție între utilizatorii platformei.

Comparativ cu alte platforme disponibile pe piață, SwapIt se evidențiază printr-un design contemporan și o experiență de utilizare intuitivă. Printr-o analiză riguroasă a deficiențelor identificate la competitorii existenți, SwapIt a fost proiectată să ofere o interfață clară și atrăgătoare, optimizată pentru a facilita navigarea și utilizarea funcționalităților. În contrast cu designurile învechite și structurile complicate ale altor platforme, SwapIt prioritizează simplitatea și accesibilitatea, eliminând astfel barierele care ar putea descuraja utilizatorii.

Calitatea rezultatelor obținute, prezentate în capitolul 6, demonstrează eficiență și impactul pozitiv al platformei SwapIt. Deși funcțională, platforma este într-un proces continuu de îmbunătățire, reflectând deschiderea la feedback și angajamentul pentru perfecționare constantă. Fiecare decizie luată în dezvoltarea SwapIt a fost orientată către îmbunătățirea experienței utilizatorilor și adaptarea la nevoile acestora.

În concluzie, sper ca aplicația SwapIt să devină un nume de referință pe piața schimbului de obiecte, contribuind semnificativ la reducerea risipei și promovarea unui stil de viață sustenabil. SwapIt are potențialul de a transforma modul în care oamenii gestionează resursele, oferind o soluție eficientă, sigură și ecologică într-o lume din ce în ce mai conștientă de importanța protejării mediului.

7.1 Dezvoltări Ulterioare

Aplicația SwapIt are un mare potențial de dezvoltare ulterioară pentru a oferi funcționalități tot mai adecvate pentru o astfel de platformă de tranzacționare de obiecte. Printre primele îmbunătățiri și adiții la funcționalitățile existente se află:

- Implementarea funcționalității de live chat între utilizatori în frontend
- Implementarea unei metodologii de tranzacționare care să nu implice utilizatorii în mod direct printr-un acord de schimb în platformă și, de asemenea, un proces de livrare
- Filtrarea produselor după oraș și distanță
- Îmbunătățirea rezultatelor relevante ale produselor după criterii precum istoricul de schimb al utilizatorului
- Integrarea unui sistem de raportare a utilizatorilor ce dă dovadă de un caracter neadecvat în cadrul platformei
- Îmbunătățirea logistica pentru administratorii aplicației
- Îmbunătățiri continue de design și de experiență a utilizatorului

8 BIBLIOGRAFIE

- [1] Census Bureau, "E-Commerce Sales Surged During the Pandemic", preluat de pe <https://www.census.gov/library/stories/2022/04/ecommerce-sales-surged-during-pandemic.html>, [Ultima accesare: 15:06:2024]
- [2] Legatum Center for Development and Entrepreneurship, "How COVID-19 Unlocked the Adoption of E-commerce in the MENA Region", preluat de pe <https://legatum.mit.edu/wp-content/uploads/2021/03/170321-MIT-Wamda-E-Commerce-COVID19-report-EN-01.pdf#:~:text=URL%3A%20https%3A%2F%2Flegatum.mit.edu%2Fwp>, [Ultima accesare: 15:06:2024]
- [3] Ellen MacArthur Foundation, "Towards the circular economy Vol. 2: opportunities for the consumer goods sector", preluat de pe:
<https://www.ellenmacarthurfoundation.org/towards-the-circular-economy-vol-2-opportunities-for-the-consumer-goods>, [Ultima accesare: 15:06:2024]
- [4] Accenture, "Cracking the code on consumer fraud", preluat de pe
<https://www.accenture.com/content/dam/accenture/final/industry/public-service/document/Accenture-Public-Safety-Cracking-Code-Consumer-Fraud.pdf#zoom=40>
[Ultima accesare: 16:06:2024]
- [5] PwC, "The global consumer: Changed for good", preluat de pe <https://www.pwc.com/gx/en/consumer-markets/consumer-insights-survey/2021/gcis-june-2021.pdf#:~:text=URL%3A%20https%3A%2F%2Fwww.pwc.com%2Fgx%2Fen%2Fconsumer>, [Ultima accesare: 15:06:2024]
- [6] Gartner, "How to Build a Digital Business Technology Platform", preluat de pe <https://www.gartner.com/smarterwithgartner/how-to-build-a-digital-business-technology-platform>, [Ultima accesare: 15:06:2024]
- [7] TechJury, "31+ User Experience Stats 2024", preluat de pe <https://techjury.net/blog/user-experience-stats/>, [Ultima accesare: 15:06:2024]
- [8] Free Code Camp, "How to Use an API with Postman – A Step-by-Step Guide", preluat de pe <https://www.freecodecamp.org/news/how-to-use-an-api-with-postman/>, [Ultima accesare: 13:06:2024]
- [9] Javapoint, "Jmeter Tutorial", preluat de pe <https://www.javatpoint.com/jmeter-tutorial>,
[Ultima accesare: 13:06:2024]
- [10] Google Developer Support, preluat de pe
<https://developer.chrome.com/docs/lighthouse/overview/>, [Ultima accesare: 17:06:2024]
- [11] Documentație React, preluat de pe <https://react.dev/learn>

[12] Documentație Firebase, preluat de pe <https://firebase.google.com/docs>

[13] Spring.io, "Why Spring?", preluat de pe <https://spring.io/why-spring>

[14] Despre PostgreSQL, preluat de pe <https://www.postgresql.org/about/>

[15] Documentație Spring, preluat de pe https://cloud.spring.io/spring-cloud-netflix/multi/multi_spring-cloud-feign.html

[16] Documentație Hazelcast, preluat de pe <https://docs.hazelcast.com/home/>

[17] Baeldung, "Java AES Encryption and Decryption", preluat de pe <https://www.baeldung.com/java-aes-encryption-decryption>, [Ultima accesare: 17:06:2024]

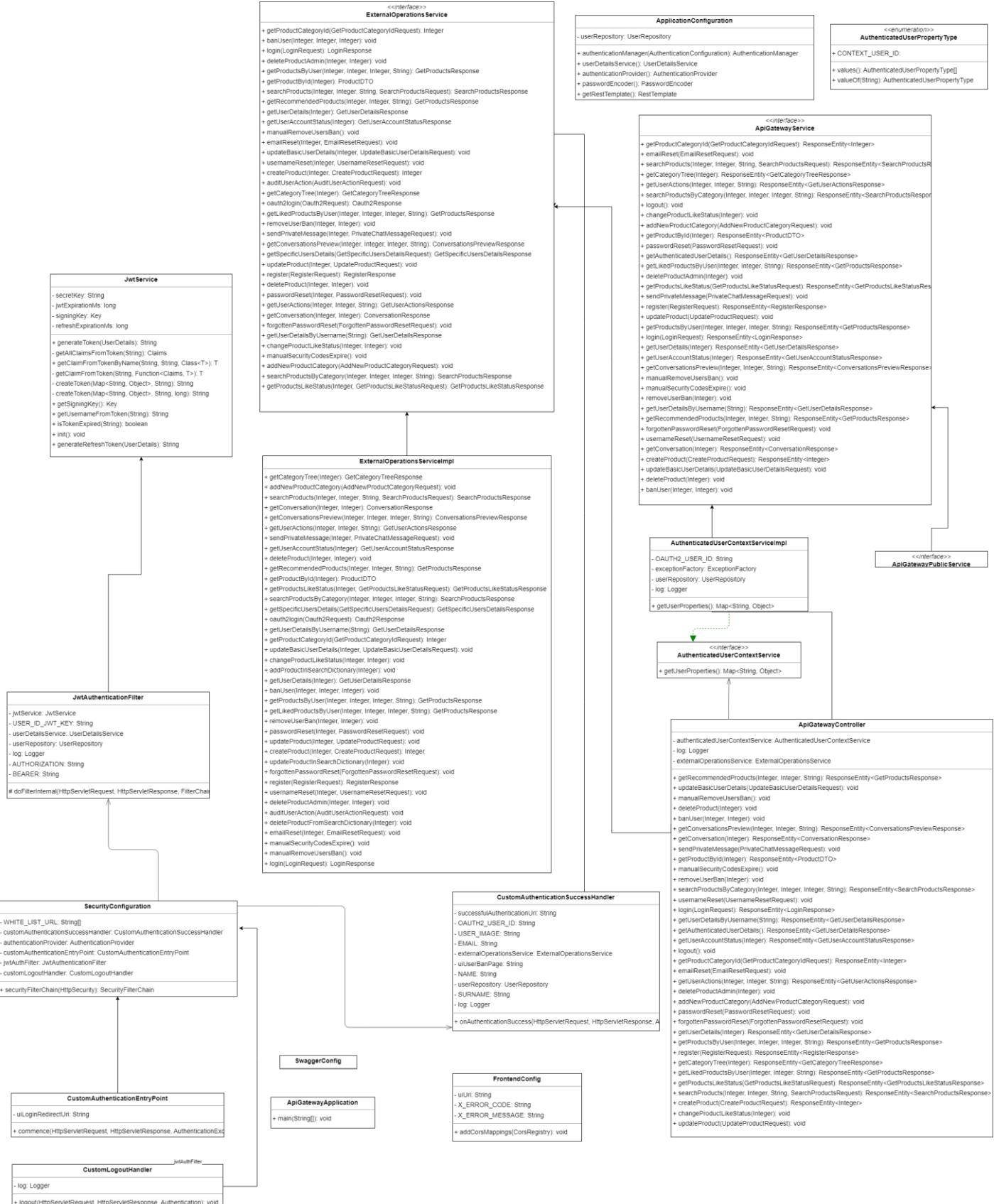
[18] Ghid Spring, " Accessing Data with JPA", preluat de pe <https://spring.io/guides/gs/accessing-data-jpa>, [Ultima accesare: 17:06:2024]

[19] Documentație Swagger, preluat de pe <https://swagger.io/docs/>

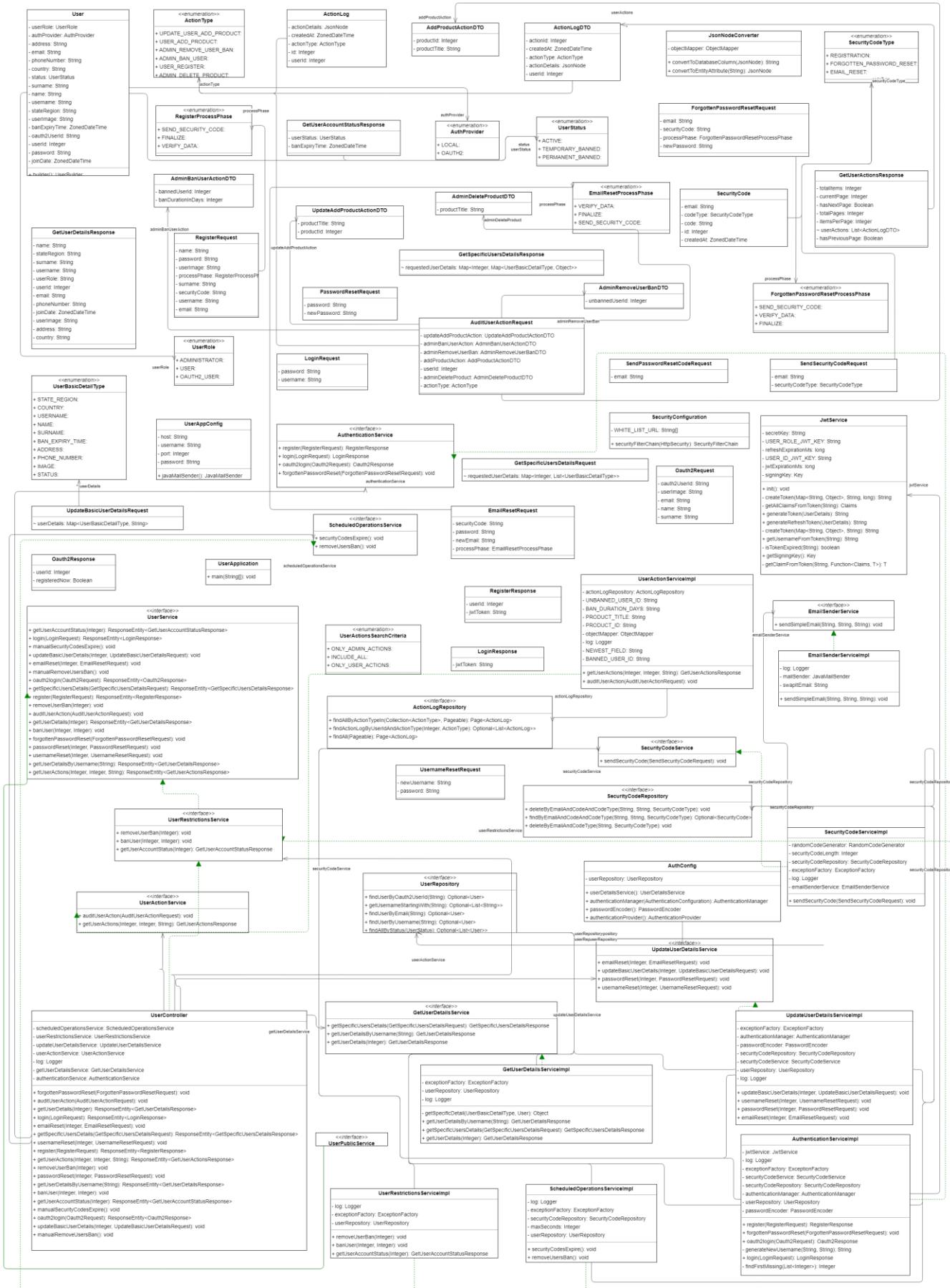
[20] Geeksforgeeks, " Binary Search Algorithm – Iterative and Recursive Implementation", preluat de pe <https://www.geeksforgeeks.org/binary-search/>, [Ultima accesare: 17:06:2024]

[21] Wikipedia, "Levenshtein distance", preluat de pe https://en.wikipedia.org/wiki/Levenshtein_distance#:~:text=The%20Levenshtein%20distance%20between%20two,one%20word%20into%20the%20other

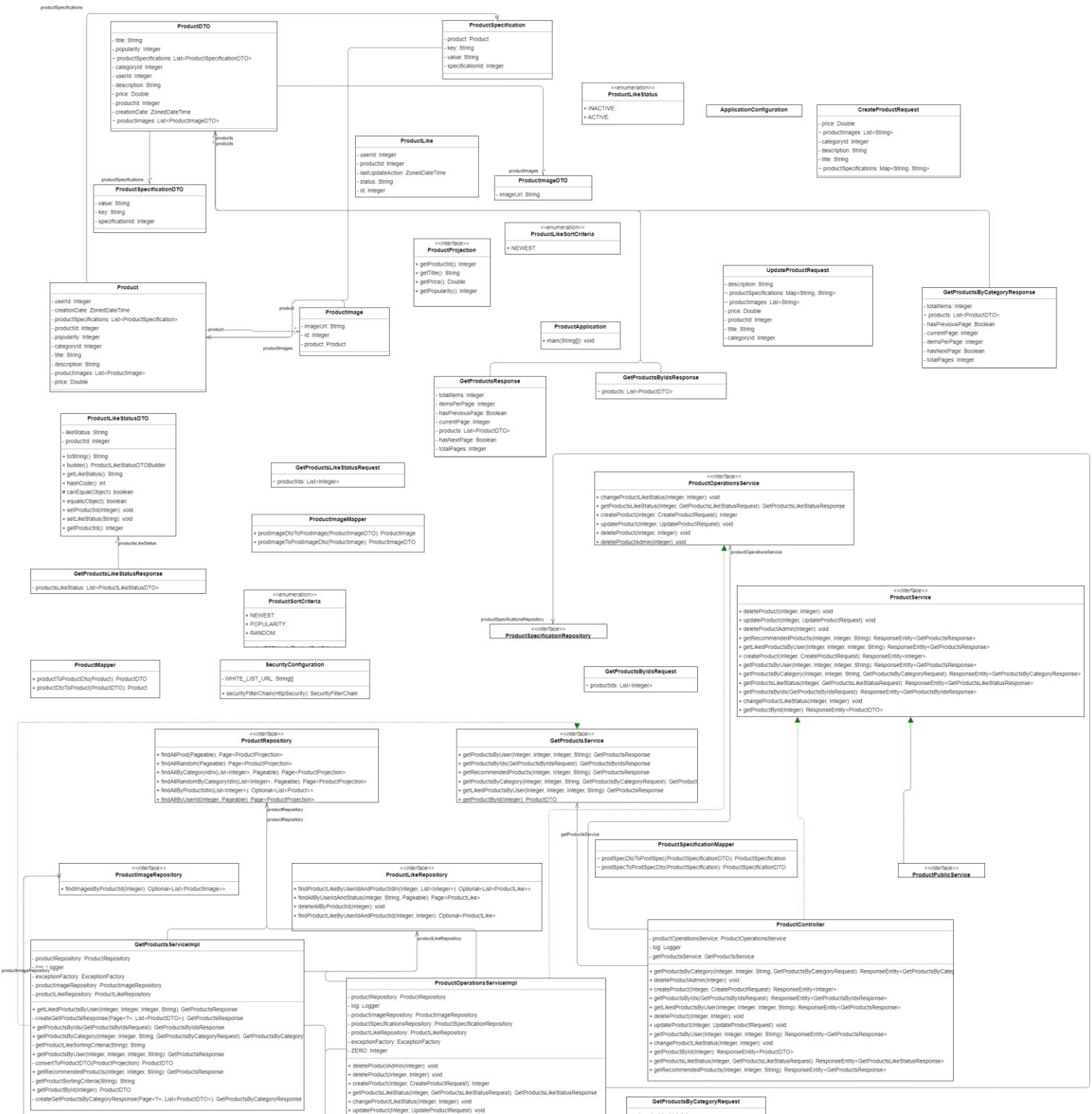
9 ANEXE



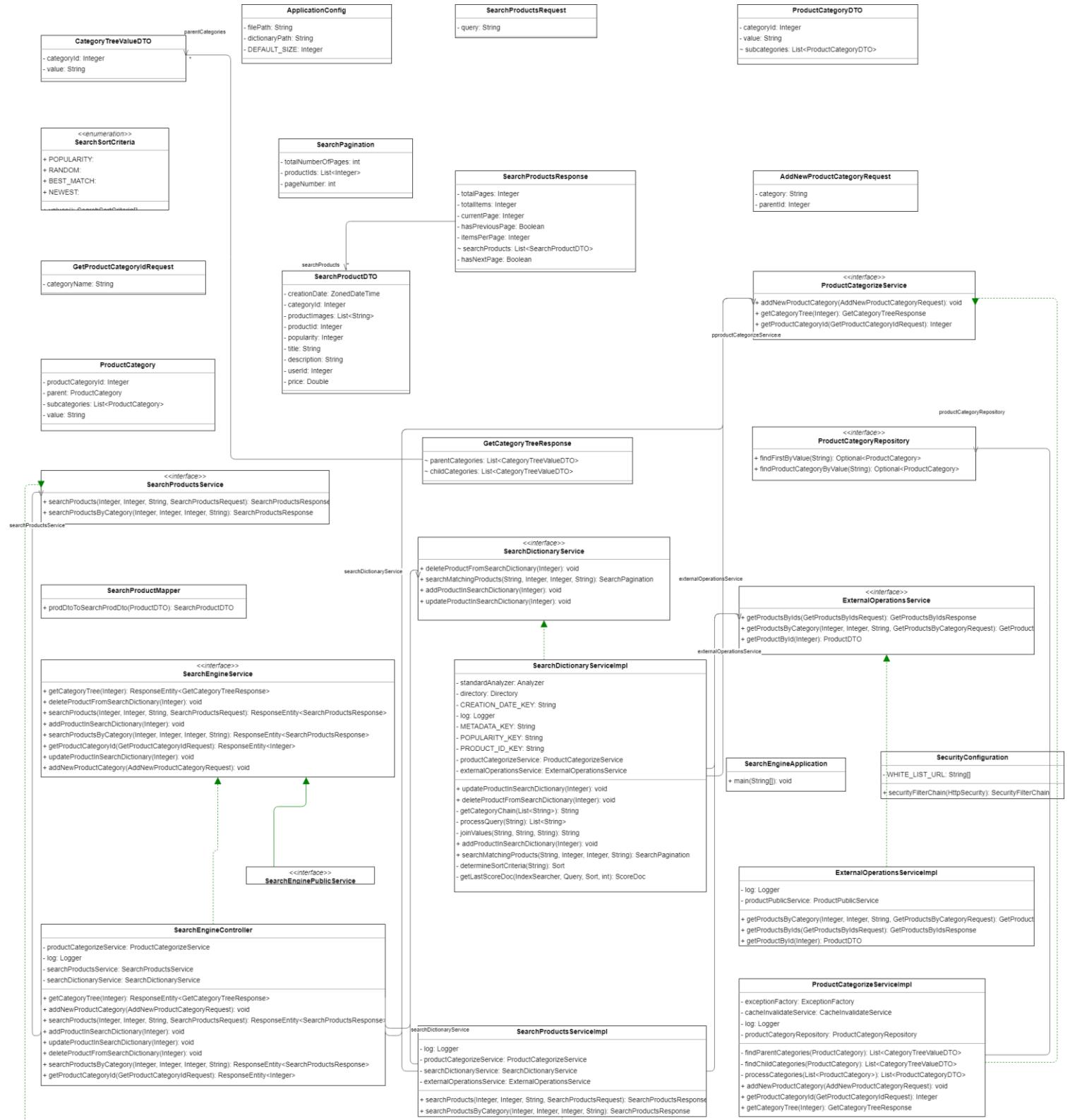
Anexa A1 - Diagrama UML a microserviciului Api Gateway



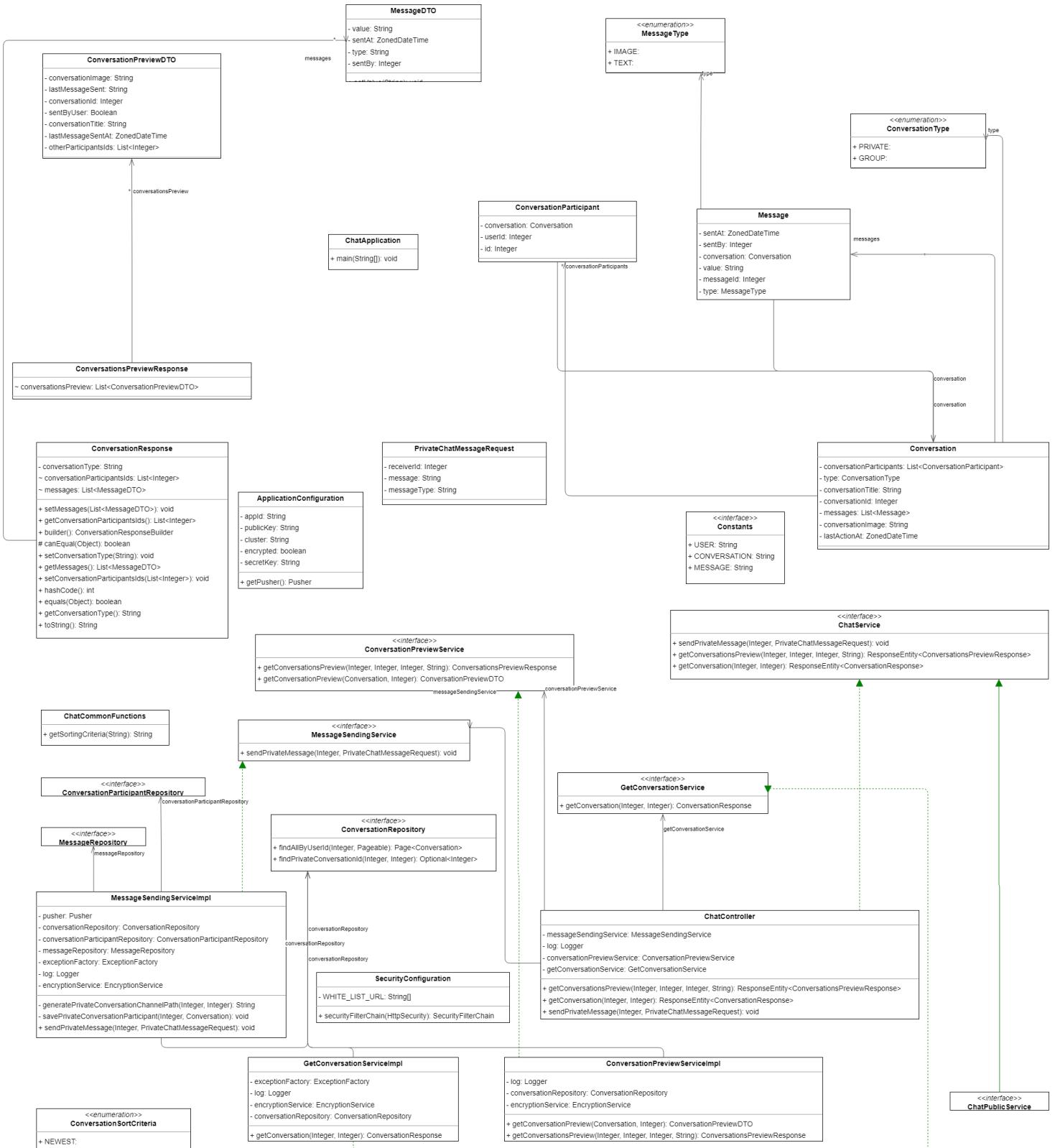
Anexa A2 – Diagrama UML a microserviciului utilizator



Anexa A3 – Diagrama UML a microserviciului produs



Anexa A4 – Diagrama UML a microserviciului motor de căutare



Anexa A5 – Diagrama UML a microserviciului de chat

```

@ControllerAdvice  ↴ AlexandruOlteanu
@Slf4j
public class ExceptionHandler extends ResponseEntityExceptionHandler {

    private static final String X_ERROR_CODE = "x_error_code";  8 usages
    private static final String X_ERROR_MESSAGE = "x_error_message";  7 usages
    @org.springframework.web.bind.annotation.ExceptionHandler({ Throwable.class })  ↴ AlexandruOlteanu
    public ResponseEntity<Object> handleMicroserviceThrownErrors(Throwable ex) {
        HttpHeaders headers = new HttpHeaders();
        MicroserviceException microserviceException;
        String errorCode = null, errorMessage = null;
        HttpStatus httpStatus = null;
        if (ex instanceof CompletionException) {
            ex = ex.getCause();
        }
        if (ex instanceof MicroserviceException mex) {
            headers.add(X_ERROR_CODE, mex.getErrorCode());
            headers.add(X_ERROR_MESSAGE, mex.getMessage());
            errorCode = mex.getErrorCode();
            errorMessage = mex.getErrorMessage();
            httpStatus = mex.getHttpStatus();
        }
        if (ex instanceof FeignException feignException) {
            Map<String, Collection<String>> responseHeaders = feignException.responseHeaders();
            if (responseHeaders.containsKey(X_ERROR_CODE)) {
                errorCode = responseHeaders.get(X_ERROR_CODE).iterator().next();
                headers.add(X_ERROR_CODE, errorCode);
            }
            if (responseHeaders.containsKey(X_ERROR_MESSAGE)) {
                errorMessage = responseHeaders.get(X_ERROR_MESSAGE).iterator().next();
                headers.add(X_ERROR_MESSAGE, errorMessage);
            }
            httpStatus = HttpStatus.resolve(feignException.status()) != null ? HttpStatus.resolve(feignException.status()) : HttpStatus.INTERNAL_SERVER_ERROR;
        }
        if (ex instanceof RestClientResponseException restClientResponseException) {
            headers.addAll(Objects.requireNonNull(restClientResponseException.getResponseHeaders()));
            if (headers.containsKey(X_ERROR_CODE)) {
                errorCode = Objects.requireNonNull(headers.get(X_ERROR_CODE)).getFirst();
            }
            if (headers.containsKey(X_ERROR_MESSAGE)) {
                errorMessage = Objects.requireNonNull(headers.get(X_ERROR_MESSAGE)).getFirst();
            }
            restClientResponseException.getStatusCode();
            httpStatus = (HttpStatus) restClientResponseException.getStatusCode();
        }
        if (!headers.containsKey(X_ERROR_CODE)) {
            headers.add(X_ERROR_CODE, headerValue: "-1");
            errorCode = "-1";
            errorMessage = ex.toString();
            errorMessage = errorMessage.replace( oldChar: '\n', newChar: ' ');
            headers.add(X_ERROR_MESSAGE, errorMessage);
            httpStatus = HttpStatus.INTERNAL_SERVER_ERROR;
        }
        microserviceException = new MicroserviceException(errorCode, errorMessage, httpStatus);
        microserviceException.setStackTrace(ex.getStackTrace());
        log.error("Error: {}", ex.getMessage(), microserviceException);
        assert httpStatus != null;
        return new ResponseEntity<>(headers, httpStatus);
    }
}

```

Anexa A6 – Propagarea erorilor între microservicii