MINISTERUL EDUCAȚIEI ȘI CERCETĂRII

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
DEPARTAMENTUL CALCULATOARE

Vendify - Platforma eCommerce pentru Crearea si Administrarea Magazinelor Online

LUCRARE DE LICENȚĂ

Absolvent: **Robert ERNST**

Coordonator științific: **S.L. Dr. Ing. Raluca PORTASE**

2025

Cuprins

Capitolul 1 Introducere **1**
1.1 Transformarea Comerțului Electronic . . . 1
1.2 Gestionarea Scalabilă a Magazinelor Online . . . 1
1.2.1 Valoarea Generalizării . . . 1
1.2.2 Provocări Majore . . . 2
1.3 Direcția Propusă . . . 2

Capitolul 2 Obiectivele Proiectului **3**
2.1 Obiectiv General . . . 3
2.2 Funcționalități Esențiale . . . 3
2.3 Obiective Specifice . . . 3
2.4 Cerințe Funcționale . . . 4
2.5 Cerințe Non-Funcționale . . . 4
2.6 Provocări Specifice . . . 5
2.7 Rezultate așteptate . . . 5

Capitolul 3 Studiu Bibliografic **6**
3.1 Contextul Domeniului . . . 6
3.2 Categorizarea Abordărilor Comerciale . . . 6
3.2.1 Platforme comerciale centralizate . . . 6
3.2.2 Platforme flexibile bazate pe CMS-uri open-source . . . 6
3.2.3 Constructoare vizuale de magazine . . . 6
3.2.4 Soluții personalizate dezvoltate la comandă . . . 6
3.3 Lucrări Relevante din Literatură . . . 7
3.3.1 Arhitecturi Multi-Tenant . . . 7
3.3.2 UI/UX Adaptiv și Responsive . . . 7
3.3.3 Arhitectură cu Microservicii și Reacție în Timp Real . . . 8
3.3.4 Analiză de Date și Statistică Comercială . . . 8
3.4 Sinteză Comparativă și Justificare . . . 8

Capitolul 4 Analiză și fundamentare teoretică **9**
4.1 Arhitectura Conceptuală a Platformei Vendify . . . 9
4.2 Justificarea Alegerilor Tehnologice . . . 10
4.2.1 Microservicii cu Spring Boot . . . 10
4.2.2 Baze de date mixte: PostgreSQL și MongoDB . . . 11
4.2.3 Acces reactiv la date (R2DBC) . . . 12
4.2.4 Firebase pentru conținut media . . . 12
4.2.5 Interfață grafică în React . . . 13
4.3 Arhitectura Conceptuală a Sistemului . . . 14
4.4 Microserviciul de Conturi și Autentificare . . . 15
4.4.1 Responsabilități . . . 15
4.4.2 Autentificare bazată pe JSON Web Tokens (JWT) . . . 15
4.4.3 Gestionarea Sesiunilor prin Cookie-uri . . . 16
4.4.4 Rol în Arhitectura Platformei . . . 16
4.4.5 Importanță în Ecosistemul Vendify . . . 16

4.5 Microserviciul de Comenzi . . . 17
4.5.1 Responsabilități . . . 17
4.5.2 Rol în Arhitectura Platformei . . . 17
4.5.3 Importanță în Ecosistemul Vendify . . . 17
4.6 Microserviciul de Magazine . . . 17
4.6.1 Responsabilități . . . 17
4.6.2 Rol în Arhitectura Platformei . . . 18
4.6.3 Importanță în Ecosistemul Vendify . . . 18
4.7 Microserviciul de Produse . . . 18
4.7.1 Responsabilități . . . 18
4.7.2 Rol în Arhitectura Platformei . . . 18
4.7.3 Importanță în Ecosistemul Vendify . . . 18
4.8 Interfața grafică . . . 18
4.8.1 Structură Generală . . . 19
4.8.2 Funcționalități Principale . . . 19
4.8.3 Personalizarea Tematică a Magazinului . . . 19
4.8.4 Interacțiune cu Backend-ul . . . 20
4.8.5 Considerații de Design și Accesibilitate . . . 20

Capitolul 5 Proiectare de detaliu și implementare **21**
5.1 Arhitectura Logică a Sistemului . . . 21
5.2 Date predefinite pentru testare și demonstrație . . . 22
5.3 Microserviciul de Conturi – Modelare și Flux Logic . . . 22
5.3.1 Modelarea Entităților Utilizatori și Sesiuni . . . 23
5.3.2 Date predefinite pentru conturi și sesiuni . . . 24
5.3.3 Fluxuri Logice Principale . . . 24
5.3.4 Filtrul de Securitate . . . 27
5.3.5 Securitate declarativă per endpoint . . . 28
5.3.6 Anteturile HTTP verificate . . . 28
5.3.7 Fluxul logic al filtrării . . . 28
5.3.8 Interogări Analitice în Microserviciul de Conturi . . . 29
5.3.9 Avantaje Arhitecturale . . . 31
5.3.10 Concluzie . . . 31
5.4 Microserviciul de Magazine – Gestionarea și Configurarea Magazinelor . . . 31
5.4.1 Modelarea Entității Store . . . 31
5.4.2 Date predefinite pentru magazine . . . 32
5.4.3 Fluxuri Logice Principale . . . 32
5.4.4 Personalizarea Magazinului . . . 33
5.4.5 Avantaje Arhitecturale . . . 34
5.5 Microserviciul de Comenzi – Modelare și Fluxuri de Comandă . . . 34
5.5.1 Modelarea Entităților Order, CartItem și WishlistItem . . . 35
5.5.2 Date predefinite pentru comenzi, coș și lista de dorințe . . . 36
5.5.3 Fluxurile Logice Principale . . . 36
5.5.4 Interogări Analitice în Microserviciul de Comenzi . . . 38
5.5.5 Avantaje Arhitecturale . . . 39
5.6 Microserviciul de Produse – Gestionarea și Catalogarea Produselor . . . 39
5.6.1 Modelarea Entităților: Product și Sale . . . 39
5.6.2 Date predefinite pentru produse și reduceri . . . 40

5.6.3 Fluxuri Logice Principale . . . 41
5.6.4 Avantaje Arhitecturale . . . 41
5.6.5 Concluzie . . . 42
5.7 Implementarea Interfeței Grafice . . . 42
5.7.1 Structura aplicației . . . 42
5.7.2 Gestionarea sesiuniilor . . . 43
5.7.3 Logica de Navigație intre pagini . . . 44
5.7.4 Injectarea Tematicii Vizuale . . . 45
5.7.5 Încărcarea Fișierelor Media în Firebase . . . 45
5.7.6 Concluzie . . . 46

Capitolul 6 Testare și validare **47**
6.1 Strategia de Testare . . . 47
6.2 Scenarii de Testare Funcțională . . . 47
6.3 Instrumente Utilizate pentru Validare . . . 48
6.3.1 Postman . . . 48
6.3.2 Redux DevTools . . . 48
6.3.3 Browser DevTools . . . 49
6.3.4 Consola Firebase . . . 49
6.3.5 Logarea în Backend . . . 50
6.4 Direcții de Îmbunătățire . . . 50
6.5 Concluzie . . . 51

Capitolul 7 Manual de instalare și utilizare **52**
7.1 Cerințe de sistem . . . 52
7.2 Instrucțiuni de instalare . . . 52
7.2.1 Clonarea proiectului . . . 52
7.2.2 Configurarea bazelor de date . . . 52
7.2.3 Pornirea serviciilor . . . 53
7.2.4 Pornirea interfetei grafice . . . 53
7.3 Ghid de utilizare al aplicației . . . 53
7.3.1 Accesarea aplicației . . . 53
7.3.2 Ghid pentru achiziționarea unui produs . . . 54
7.3.3 Autentificarea sau crearea unui cont . . . 54
7.3.4 Căutarea unui produs . . . 55
7.3.5 Vizualizarea detaliilor produsului . . . 55
7.3.6 Adăugarea în coșul de cumpărături . . . 56
7.3.7 Finalizarea comenzii . . . 57

Capitolul 8 Concluzii **58**
8.1 Rezumat al contribuțiilor . . . 58
8.2 Analiza critică a rezultatelor . . . 58
8.3 Dezvoltări și direcții viitoare . . . 59

Bibliografie **60**

Anexa A Secțiuni relevante din cod **62**

Capitolul 1

Introducere

Această lucrare propune dezvoltarea unei platforme generalizate pentru gestionarea eficientă a magazinelor online, adaptată în special nevoilor micilor antreprenori aflați în căutarea unor soluții digitale accesibile și scalabile. Sub denumirea **Vendify**, proiectul urmărește să reducă barierele tehnice asociate lansării unui magazin electronic, printr-o combinație de interfețe intuitive, funcționalități preconfigurate și o arhitectură modulară, capabilă să susțină personalizări rapide și extindere ulterioară.

1.1. Transformarea Comerțului Electronic

În ultimele două decenii, digitalizarea a remodelat radical ecosistemul comercial, dând naștere unei economii în care prezența online nu mai este un avantaj competitiv opțional, ci o cerință esențială pentru supraviețuirea și dezvoltarea afacerilor. În special pentru actorii de dimensiuni mici și medii, această presiune a determinat o orientare accelerată către soluții eCommerce ușor de adoptat și menținut.

În teorie, platformele existente promit tocmai această simplitate. În practică, însă, multe dintre ele impun costuri recurente semnificative sau necesită un minim de expertiză tehnică, ceea ce le face inaccesibile unei părți considerabile a utilizatorilor interesați. În acest context, devine tot mai clară necesitatea unei soluții care nu doar să fie complet funcțională, ci și cu adevărat accesibilă din punct de vedere tehnologic și economic.

1.2. Gestionarea Scalabilă a Magazinelor Online

Pe măsură ce afacerile se extind, crește și presiunea operațională asupra administratorilor: gestionarea eficientă a stocurilor, monitorizarea comenzilor, personalizarea interfeței și menținerea unei experiențe coerente pentru utilizatori devin provocări recurente. Această complexitate, dacă nu este atenuată prin mijloace tehnologice adecvate, poate deveni un obstacol în calea creșterii.

De aceea, un obiectiv central al platformelor moderne trebuie să fie disimularea acestei complexități în spatele unor interfețe prietenoase, care permit utilizatorilor non-tehnici să desfășoare activități comerciale fără intervenția unor specialiști IT. În mod ideal, operațiuni precum adăugarea de produse, modificarea culorilor sau gestionarea comenzilor ar trebui să se realizeze prin câteva interacțiuni simple, fără a compromite controlul asupra detaliilor esențiale.

1.2.1. Valoarea Generalizării

Un sistem standardizat pentru lansarea de magazine online poate conduce la scăderi considerabile ale costurilor inițiale și operaționale, fără a limita în mod necesar capacitatea de adaptare la contexte specifice. În acest sens, reiese necesitatea de a avea urmatoarele funcționalități:

- componente esențiale preconfigurate (gestionarea produselor, comenzilor, clienților);
- opțiuni de personalizare vizuală prin teme, fonturi și culori;
- instrumente de analiză și vizualizare a performanței magazinului;
- scalabilitate facilă, atât la nivel de infrastructură, cât și de funcționalități.

Această abordare contribuie nu doar la reducerea timpului necesar lansării unui magazin, ci și la facilitarea unei tranziții organice către etape de dezvoltare ulterioare.

1.2.2. Provocări Majore

În procesul de proiectare a platformei, au fost identificate mai multe provocări structurale și de uzabilitate, printre care:

- echilibrarea între complexitate funcțională și simplitate de utilizare;
- sprijinirea utilizatorilor fără cunoștințe tehnice în configurarea magazinului;
- oferirea de suport nativ pentru administrarea mai multor magazine;
- integrarea unor mecanisme intuitive de personalizare vizuală;
- adaptabilitatea la trafic ridicat și volume mari de date, fără degradarea performanței;
- organizarea codului sursă pe principii de modularitate și microservicii.

1.3. Direcția Propusă

În lumina acestor provocări, soluția propusă dorește să constituie un cadru tehnologic coerent, care susține dezvoltarea durabilă a inițiativelor antreprenoriale prin automatizare, simplitate și flexibilitate.

Prin această abordare, proiectul va explora posibilitatea echilibrării între simplitate și putere operațională. Astfel, soluția propusă va încerca să faciliteze intrarea în comerțul digital pentru utilizatori fără competențe tehnice, fără a limita potențialul de extindere al platformei în funcție de evoluția afacerii.

În cele din urmă, proiectul aspiră să demonstreze că un sistem bine proiectat poate reduce barierele tehnologice, accelerând procesul de digitalizare pentru afacerile mici și mijlocii și contribuind la democratizarea accesului la infrastructura eCommerce.

Capitolul 2

Obiectivele Proiectului

2.1. Obiectiv General

Proiectul de față își propune dezvoltarea unei platforme de tip eCommerce destinată utilizatorilor fără experiență tehnică, care urmăresc să își creeze și să își administreze propriul magazin online într-un mod autonom. În centrul acestei inițiative se află ideea de a simplifica accesul la tehnologia comercială digitală, oferind un sistem capabil să gestioneze mai multe magazine simultan, fiecare cu propriile setări, produse și interfață personalizabilă. Astfel, obiectivul general constă în furnizarea unei soluții complete, scalabile și intuitive, adaptate ritmului actual de digitalizare a pieței antreprenoriale.

2.2. Funcționalități Esențiale

Vendify este conceput ca o platformă modulară, în care fiecare componentă reflectă o nevoie concretă identificată în cadrul ecosistemului eCommerce. Printre funcționalitățile considerate esențiale se numără:

- **Sistem de conturi și autentificare**: Permite înregistrarea și autentificarea în condiții de siguranță a utilizatorilor, inclusiv gestionarea sesiunilor și a rolurilor.
- **Administrarea mai multor magazine**: Oferă posibilitatea gestionării unui portofoliu de magazine, fiecare izolat la nivel de date și configurabil individual.
- **Gestionarea produselor**: Include adăugarea, editarea și organizarea produselor în categorii, cu atribute variabile precum imagini, prețuri, stocuri și reduceri.
- **Personalizare vizuală**: Permite adaptarea aspectului fiecărui magazin prin teme, fonturi și culori, fără necesitatea scrierii de cod.
- **Procesarea comenzilor**: Asigură vizualizarea, actualizarea și monitorizarea comenzilor plasate de clienți, inclusiv schimbarea statusului și gestionarea detaliilor.
- **Acces public prin URL dedicat**: Fiecare magazin este publicat printr-o adresă unică accesibilă online.
- **Panou de analiză statistică**: Furnizează rapoarte agregate privind comportamentul clienților, produsele cele mai vizualizate sau vândute, și indicatori de conversie.
- **Interfață adaptabilă**: Designul adaptabil asigură compatibilitate completă cu dispozitive mobile, tablete și desktop.
- **Arhitectură extensibilă**: Structura bazată pe microservicii permite adăugarea de funcționalități suplimentare și integrarea cu sisteme externe.

2.3. Obiective Specifice

În vederea atingerii scopului general, proiectul este ghidat de o serie de obiective operaționale, formulate astfel încât să reflecte cerințele reale ale unei platforme moderne:

a) **Autentificare sigură și gestionarea sesiunilor** prin mecanisme de validare robustă

și tokenizare.

b) **Administrarea mai multor magazine** într-un mod izolat și coerent, cu opțiuni distincte de configurare și acces.
c) **Funcționalități robuste de gestionare a produselor**, inclusiv reducerea manuală sau automată a prețurilor prin campanii.
d) **Structură ierarhică flexibilă** a produselor, organizate în categorii ce pot fi modificate dinamic de către administratori.
e) **Interfață de personalizare vizuală**, accesibilă și pentru utilizatorii fără competențe tehnice.
f) **Panou de comenzi administrativ** care permite urmărirea în timp real a cererilor primite.
g) **Publicare prin link unic** care reflectă identitatea fiecărui magazin (ex: `/exclusive`, `/essentials`).
h) **Acces la statistici relevante** privind comportamentul clienților și performanța produselor.
i) **Design compatibil cu dispozitive variate**, fără pierderi de funcționalitate sau de structură.
j) **Suport pentru scalabilitate orizontală**, prin intermediul unei arhitecturi orientate pe microservicii reactive.

2.4. Cerințe Funcționale

Pentru a îndeplini obiectivele definite, sistemul trebuie să îndeplinească următoarele cerințe funcționale:

- Crearea și autentificarea conturilor de utilizator.
- Gestionarea simultană a mai multor magazine dintr-un singur cont.
- Administrarea completă a produselor și a categoriilor aferente.
- Personalizarea interfeței magazinului fără scriere de cod.
- Vizualizarea comenzilor și actualizarea stării acestora.
- Accesibilitate publică printr-un URL dedicat.
- Vizualizarea de rapoarte în timp real despre activitatea din platformă.
- Funcționare corectă pe ecrane de dimensiuni diferite.
- Posibilitatea de extindere modulară fără impact negativ asupra serviciilor existente.

2.5. Cerințe Non-Funcționale

Pe lângă cerințele de ordin funcțional, sistemul trebuie să satisfacă și următoarele condiții de natură calitativă:

- **Performanță**: timp de răspuns redus la acțiunile utilizatorului.
- **Compatibilitate pe toate dispozitivele**: funcționalitate deplină pe mobil, tabletă și desktop;
- **Accesibilitate**: interfață intuitivă, compatibilă cu diverse profiluri de utilizator.
- **Modularitate**: separare clară a componentelor pentru facilitarea mentenanței.
- **Securitate**: protejarea datelor prin criptare, validare și politici stricte de acces.
- **Integrare**: capacitatea de a comunica cu servicii externe (ex: Stripe, Firebase) fără modificări structurale majore.
- **Compatibilitate cu standardele web**: funcționare optimă în toate browserele moderne.

2.6. Provocări Specifice

Implementarea unei platforme de asemenea natură nu este lipsită de dificultăți. Printre cele mai notabile provocări se numără:

- Găsirea unui echilibru între simplitate și personalizare extensivă.
- Construirea unui sistem multi-store coerent, izolat și scalabil.
- Implementarea unei personalizări vizuale fără implicare tehnică.
- Asigurarea unui control granular asupra datelor stocate și procesate.
- Validarea funcționalităților într-un context realist, cu utilizatori și date simulate.
- Menținerea unei experiențe unificate în cadrul tuturor modulelor aplicației.

2.7. Rezultate așteptate

La finalul dezvoltării, se preconizează că proiectul curent va livra un instrument complet funcțional, accesibil din browser, care să permită oricărui utilizator, indiferent de nivelul său tehnic, să lanseze și să gestioneze cu succes un magazin online. Accentul cade nu doar pe funcționalitate, ci și pe experiența de utilizare, scalabilitate și integrarea naturală în contextul actual al comerțului digital.

Capitolul 3

Studiu Bibliografic

3.1. Contextul Domeniului

Transformarea digitală a comerțului a dus la o explozie a platformelor eCommerce, în special în contextul noilor nevoi ale micilor antreprenori. În ultimul deceniu, soluțiile de tip SaaS [1] (Software as a Service) au devenit dominante, oferind lansare rapidă, funcționalități out-of-the-box și infrastructură gestionată automat. Platforme precum Shopify, Wix eCommerce sau WooCommerce sunt exemple relevante. Scopul acestui capitol este de a analiza abordările existente și contribuțiile relevante în literatura de specialitate și în industria de profil.

3.2. Categorizarea Abordărilor Comerciale

3.2.1. Platforme comerciale centralizate

Platforme precum Shopify [2] sau BigCommerce [3] oferă o abordare centralizată, gestionând complet infrastructura, scalarea și securitatea. Shopify permite comercianților să-și lanseze rapid magazine online printr-o interfață prietenoasă, însă limitează libertatea tehnică și implică costuri lunare ridicate. BigCommerce se adresează segmentului enterprise, fiind scalabil și robust, dar complex din punct de vedere operațional.

3.2.2. Platforme flexibile bazate pe CMS-uri open-source

CMS (Content Management System) reprezintă o aplicație software care permite crearea, gestionarea și modificarea conținutului digital fără a necesita cunoștințe avansate de programare. În contextul eCommerce, un CMS facilitează administrarea produselor, comenzilor și paginilor web printr-o interfață intuitivă.

WooCommerce [4] (plugin WordPress) și Magento [5] reprezintă abordări open-source. WooCommerce este potrivit pentru proiecte mici-medii, dar presupune configurări manuale și menținerea infrastructurii. Magento, deși foarte puternic, are o curbă de învățare ridicată și necesită expertiză tehnică.

3.2.3. Constructoare vizuale de magazine

Wix eCommerce [6] sau Squarespace [7] oferă un editor vizual performant și funcționalități de bază pentru comerț online. Sunt ideale pentru începători, însă nu se pretează pentru extindere tehnică sau personalizare avansată.

3.2.4. Soluții personalizate dezvoltate la comandă

O categorie aparte este reprezentată de soluțiile dezvoltate la comandă, care permit implementarea exactă a cerințelor specifice ale clientului. Aceste platforme oferă cel mai ridicat grad de flexibilitate, integrare cu sisteme existente și control asupra arhitec-

turii. Totuși, presupun investiții financiare considerabile, timpi de dezvoltare mai mari și dependență de o echipă tehnică dedicată. Din acest motiv, sunt mai potrivite pentru companii medii și mari sau proiecte cu cerințe foarte specifice.

Limitările soluțiilor comerciale actuale precum Shopify, Wix sau WooCommerce, devin vizibile în momentul în care afacerea depășește pragul standard de funcționalitate. Lipsa unui control granular asupra infrastructurii, imposibilitatea de a adapta fluxuri de lucru complexe, precum și constrângerile impuse de ecosistemul platformei (inclusiv comisioane și dependența de funcționalități proprietare) determină tot mai mulți antreprenori să caute alternative personalizate, capabile să evolueze odată cu business-ul lor.

Competitor	**Tip soluție**	**Puncte forte**	**Puncte slabe**
Shopify	Platformă eCommerce completă	Ușor de configurat, acces la o piață vastă de aplicații	Taxe ridicate, personalizare limitată a backend-ului
WooCommerce	Plugin open-source pentru WordPress	Nivel ridicat de personalizare, gratuit și flexibil	Necesită configurare manuală, dependență de WordPress
Magento	Platformă eCommerce scalabilă	Funcționalități avansate, foarte scalabilă	Curbă de învățare abruptă, necesită resurse de dezvoltare considerabile
Wix/Squarespace	Constructoare vizuale de magazine	Interfață intuitivă, potrivite pentru utilizatori fără cunoștințe tehnice	Posibilități limitate de extindere și personalizare avansată

Tabela 3.1: Comparație între platformele eCommerce analizate

Tabelul de mai sus compară principalele tipuri de platforme eCommerce din perspectiva accesibilității, costurilor, flexibilității și nivelului de control oferit utilizatorului final:

3.3. Lucrări Relevante din Literatură

3.3.1. Arhitecturi Multi-Tenant

Modelul *multi-tenant* reprezintă o abordare arhitecturală des utilizată în dezvoltarea aplicațiilor SaaS (Software as a Service), în care o singură instanță a aplicației servește mai mulți clienți (denumiți *tenants*). Fiecare client beneficiază de un spațiu izolat pentru datele și configurațiile proprii, însă resursele software și hardware sunt partajate eficient.

Această arhitectură aduce avantaje semnificative în ceea ce privește costurile de operare, mentenanța și scalabilitatea, mentionat si in lucrarea [8]. Spre deosebire de modelul *single-tenant*, unde fiecare client necesită o instanță separată, modelul multi-tenant reduce duplicarea codului și a infrastructurii, facilitând în același timp gestionarea centralizată a aplicației.

3.3.2. UI/UX Adaptiv și Responsive

Literatura de specialitate [9] evidențiază impactul major pe care designul interfeței îl are asupra conversiei. Noțiunea de UI/UX adaptiv implică o interfață care se ajustează în funcție de preferințele și comportamentul utilizatorului. De asemenea, un design responsive înseamnă că aplicația se adaptează automat la diferite dimensiuni de ecran:

mobil, tabletă, desktop, oferind aceeași experiență fluentă și coerentă.

3.3.3. Arhitectură cu Microservicii și Reacție în Timp Real

Lucrarea lui Newman [10] evidențiază avantajele arhitecturilor bazate pe microservicii în comparație cu modelele monolitice tradiționale. O aplicație este împărțită în componente independente (servicii), fiecare responsabilă de un domeniu specific (ex: gestionarea utilizatorilor, procesarea comenzilor, etc.). Acest model oferă multiple beneficii: scalabilitate pe verticală sau orizontală, actualizări izolate, toleranță ridicată la erori și posibilitatea distribuirii sarcinilor între echipe diferite.

În paralel, adoptarea programării reactive (ex: folosind biblioteci precum Project Reactor sau frameworkuri precum Spring WebFlux), evidențiată și în lucrarea [11], este tot mai frecventă în aplicațiile moderne. Abordarea reactivă permite gestionarea eficientă a cererilor concurrente prin fluxuri asincrone și non-blocante, ceea ce reduce consumul de resurse și crește performanța în scenarii cu trafic intens.

3.3.4. Analiză de Date și Statistică Comercială

Importanța colectării și analizei datelor comportamentale pentru optimizarea procesului de vanzare în eCommerce este susținută de lucrările [12, 13], unde statistica nu se limitează doar la vizualizări, ci este structurată pe două niveluri:

- *Date agregate la nivel de pagină*: se colectează informații privind numărul de vizualizări pentru fiecare produs, categorie sau pagină principală. Acest nivel de analiză permite identificarea celor mai populare produse și trasee frecvente de navigare.
- *Informații detaliate despre sesiuni*: sistemul monitorizează durata fiecărei sesiuni, acțiunile efectuate (căutări, filtrări, adăugări în coș), dar și comportamente de conversie (ex: abandonare coș vs. finalizare comandă). Astfel, se pot identifica punctele slabe din fluxul de cumpărare și optimiza performanța platformei.

Corelarea acestor niveluri analitice poate contribui semnificativ la îmbunătățirea procesului decizional, susținând inițiative strategice precum reorganizarea catalogului, adaptarea campaniilor de promovare sau ajustarea elementelor de interfață pentru a spori conversia.

3.4. Sinteză Comparativă și Justificare

Din studiul meticulos al literaturii de specialitate și ale soluțiilor existente în literatură, au rezultat următoarele limitări:

- Soluțiile comerciale sunt ușor de utilizat, dar limitate ca personalizare.
- Cele open-source oferă flexibilitate, dar presupun efort tehnic.
- Abordările științifice pun accent pe scalabilitate, modularitate și personalizare automatizată.

Pentru a depăși limitările identificate în literatură si soluțiile industriale curente, **Vendify** se poziționează ca o platformă hibridă, inspirată din bune practici academice și comerciale, oferind o soluție scalabilă, ușor de utilizat și adresabilă antreprenorilor cu resurse limitate. În plus, datorită mecanismului său automatizat de configurare și publicare, un magazin poate fi lansat imediat după completarea pașilor esențiali, fără intervenții tehnice suplimentare. Această abordare reduce semnificativ timpul de lansare și permite antreprenorilor să înceapă rapid activitatea online.

Capitolul 4

Analiză și fundamentare teoretică

4.1. Arhitectura Conceptuală a Platformei Vendify

Platforma **Vendify** este construită pe baza unei arhitecturi modulare, distribuite în microservicii, concepută pentru a asigura scalabilitate, flexibilitate și separare clară a responsabilităților funcționale. Structura logică a sistemului reflectă principalele procese operaționale implicate în gestionarea unui magazin online: administrarea produselor, procesarea comenzilor, autentificarea și gestiunea sesiunilor, precum și configurarea și personalizarea magazinului.

Arhitectura este împărțită în două niveluri majore:

- **Nivelul de prezentare (Frontend)** – responsabil pentru interacțiunea cu utilizatorii finali, construit în React, oferă o interfață intuitivă și dinamică pentru gestionarea magazinului, plasarea comenzilor și vizualizarea conținutului.
- **Nivelul de logică aplicativă (Backend)** – este format din patru microservicii independente, fiecare având o responsabilitate distinctă și fiind conectat la o bază de date dedicată.

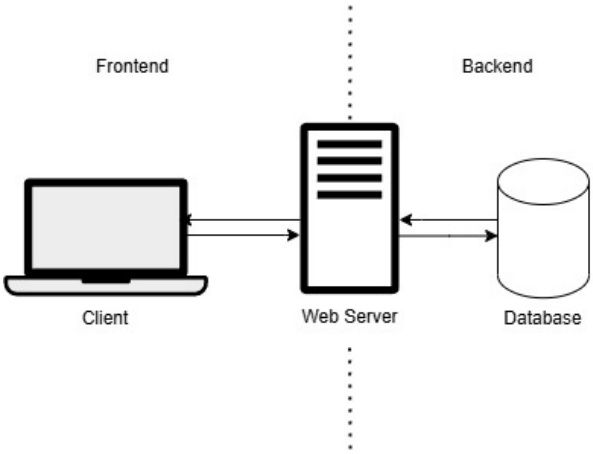

Figura 4.1: Arhitectură simplificată: comunicarea dintre client, server și baza de date

Figura 4.1 ilustrează o arhitectură web clasică în care componenta **frontend** (clientul) comunică cu un **server web** (backend), care, la rândul său, interacționează cu o **bază de date**. Această structură stă la baza oricărei aplicații moderne, iar în cazul Vendify este extinsă prin microservicii, dar respectă aceeași logică fundamentală de comunicare client-server.

Această structură logică sprijină dezvoltarea agilă, prin:

- izolarea componentelor critice pentru o mentenanță facilă;

- flexibilitatea integrării de noi funcționalități fără întreruperi;
- adaptabilitatea la schimbări de volum și scalabilitate elastică.

De asemenea, platforma utilizează stocare externă pentru resurse statice precum logo-uri, bannere sau imagini de produs, prin integrarea cu serviciul Google Firebase Storage [14], asigurând astfel performanță și disponibilitate ridicată pentru elementele media.

4.2. Justificarea Alegerilor Tehnologice

Arhitectura sistemului **Vendify** a fost fundamentată pe o serie de criterii esențiale: scalabilitate, modularitate, performanță, ușurința integrării și costuri reduse de operare. Alegerea tehnologiilor reflectă aceste cerințe:

4.2.1. Microservicii cu Spring Boot

Arhitectura Vendify este construită pe baza paradigmei microserviciilor, care permite separarea clară a responsabilităților, scalarea independentă a componentelor și dezvoltarea paralelă a echipelor. Fiecare microserviciu este implementat folosind framework-ul **Spring Boot**, ales pentru nivelul ridicat de productivitate, extensibilitate și suport integrat pentru tehnologii moderne.

Spring Boot oferă multiple avantaje fundamentale pentru implementarea unui sistem distribuit:

- **Configurare automată**: permite inițializarea rapidă a componentelor standard, reducând timpul de setup și eliminând configurația boilerplate.
- **Dependințe selective**: oferă o integrare modulară cu baze de date, framework-uri reactive, sisteme de securitate și alte tehnologii, printr-un model declarativ în `pom.xml`.
- **Server web atașat**: elimină nevoia unui server extern, permițând lansarea aplicației ca un singur jar executabil, ușor de containerizat.
- **Suport pentru arhitectură reactivă**: prin integrarea nativă cu `Spring WebFlux` și `Spring Data R2DBC/Mongo`, este posibilă dezvoltarea de API-uri non-blocante, scalabile sub sarcină mare.

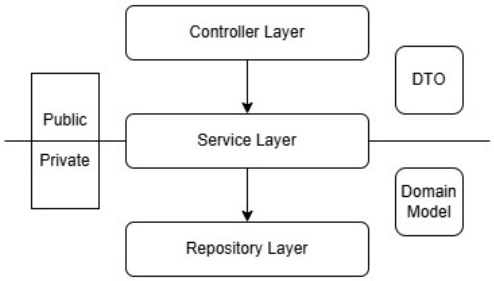

Figura 4.2: Structura internă a unui microserviciu Spring Boot

Figura 4.2 ilustrează arhitectura internă a unui microserviciu tipic în Spring Boot, organizată pe baza principiilor *Inversion of Control* și *Separation of Concerns*. Această structurare pe straturi asigură o arhitectură curată și testabilă:

- **Controller Layer** gestionează cererile HTTP primite și expune API-ul public. Utilizează DTO-uri pentru a izola reprezentarea internă a datelor.
- **Service Layer** implementează logica de afaceri, coordonează validări și transformări de date și comunică cu stratul de persistare.
- **Repository Layer** interfațează direct cu baza de date, utilizând Spring Data R2DBC (pentru PostgreSQL) sau Spring Data MongoDB. Permite scrierea de interogări declarative și reactive.

În plus, fiecare microserviciu definește modele de domeniu clare, mapate în entități persistente, și folosește anotări Spring pentru injecția de dependențe, validări și mapări REST. Aceste facilități contribuie la dezvoltarea rapidă, reduc erorile și asigură consistență între microservicii.

4.2.2. Baze de date mixte: PostgreSQL și MongoDB

Proiectul adoptă o abordare hibridă în ceea ce privește stocarea datelor, combinând avantajele oferite de bazele de date relaționale și cele NoSQL (non-relaționale), în funcție de natura și structura datelor gestionate de fiecare microserviciu. Această alegere reflectă o orientare spre o practică tot mai des întâlnită în arhitecturile distribuite, care presupune utilizarea mai multor tipuri de sisteme de gestionare a bazelor de date, fiecare adaptat la cerințele specifice ale unui context funcțional.

În cadrul acestei platforme, serviciile care implică date bine structurate, relații riguroase între entități și necesități tranzacționale stricte, precum conturile utilizatorilor, sesiunile de autentificare și starea comenzilor, utilizează PostgreSQL. Alegerea acestui sistem relațional se bazează pe conformitatea sa cu standardele ACID, suportul solid pentru interogări complexe și fiabilitatea în manipularea datelor sensibile. În plus, ecosistemul PostgreSQL oferă extensibilitate și performanță optimizată pentru aplicații scalabile, dar impune o schemă strict definită și un proces mai elaborat de modificare structurală, ceea ce poate constitui o limitare în scenarii dinamice.

Pe de altă parte, pentru componentele care gestionează date semi-structurate sau nestandardizate, cum ar fi produsele, categoriile vizuale sau configurațiile personalizate ale magazinelor, a fost ales MongoDB. Această bază de date documentară permite o flexibilitate crescută în modelarea datelor, permițând adăugarea și modificarea câmpurilor fără constrângeri schematice rigide. În contexte unde structura datelor poate varia frecvent sau conține atribute opționale, MongoDB facilitează o evoluție mai agilă a modelului de date. Totuși, lipsa de consistență tranzacțională și dificultățile asociate cu validarea relațiilor între entități pot ridica probleme în cazul unei utilizări necontrolate sau în scenarii ce impun coerență strictă.

Această separare între PostgreSQL și MongoDB nu este una pur tehnică, ci răspunde unor diferențe conceptuale între stabilitate și flexibilitate, între control tranzacțional și adaptabilitate structurală. Alegerea acestei strategii duale se aliniază totodată cu principiile de *bounded context*, permițând fiecărui microserviciu să fie proiectat și optimizat independent, în funcție de cerințele sale operaționale. Deși o astfel de diversificare poate introduce un cost suplimentar în ceea ce privește mentenanța și complexitatea infrastructurii, avantajele în materie de performanță și adaptabilitate justifică, în acest caz, această decizie arhitecturală.

- **PostgreSQL** va fi folosit în microserviciile `Accounts` și `Products`, deoarece este potrivit pentru date relaționale, cu constrângeri stricte de integritate și o schemă

stabilă (de exemplu: conturi de utilizator, sesiuni, produse). Avantajele includ:
  - consistență și normalizare puternică a datelor;
  - suport avansat pentru interogări SQL și tranzacții;
  - compatibilitate cu biblioteci reactive prin `R2DBC`.
- **MongoDB** va fi utilizat în `Orders` și `Stores`, deoarece este ideal pentru entități cu structură flexibilă sau în continuă evoluție, cum ar fi coșuri de cumpărături sau configurări personalizabile ale magazinului. Avantajele includ:
  - stocare documentară orientată pe JSON, ideală pentru obiecte complexe;
  - schemă flexibilă și adaptabilă;
  - performanță superioară la scrieri masive și citiri agregate simple.

Această împărțire permite fiecărui microserviciu să utilizeze modelul de persistență cel mai adecvat propriului domeniu, fără a impune constrângeri artificiale sau redundanțe. Mai mult, decuplarea completă între serviciu și tehnologia de stocare contribuie la menținerea unui grad ridicat de modularitate și extensibilitate.

Caracteristică	**PostgreSQL**	**MongoDB**
Model de date	Relațional (tabele)	Documente (JSON-like)
Structură schemă	Fixă, normalizată	Flexibilă, semi-structurată
Performanță la interogări complexe	Ridicată	Limitată la cazuri simple
Performanță la scriere/volum mare	Medie	Ridicată
Utilizare în Vendify	Conturi, produse, sesiuni	Coșuri, dorințe, teme, magazine

Tabela 4.1: Comparație între PostgreSQL și MongoDB în contextul Vendify

Tabelul 4.1 sintetizează principalele diferențe între PostgreSQL și MongoDB, evidențiind motivele pentru care am adoptat o arhitectură mixtă. PostgreSQL este ales pentru domenii cu relații bine definite și integritate ridicată a datelor. În schimb, MongoDB este preferat pentru entități cu structură flexibilă sau cu ierarhii variabile, unde viteza de scriere și adaptabilitatea sunt prioritare.

4.2.3. Acces reactiv la date (R2DBC)

În locul modelului clasic blocant de acces la date, s-a optat pentru *Reactive Relational Database Connectivity*, care permite procesarea asincronă a operațiilor de citire și scriere, rezultând în performanțe mai bune la încărcări mari.

4.2.4. Firebase pentru conținut media

Gestionarea eficientă a fișierelor media, precum imaginile de produs, bannerele sau elementele grafice personalizate, reprezintă o componentă esențială în arhitectura unei platforme eCommerce moderne. Alegerea unei soluții de stocare externă nu este trivială, întrucât implică un echilibru între performanță, scalabilitate, costuri și ușurința de integrare cu aplicația web.

În mod tradițional, dezvoltatorii pot opta pentru gestionarea fișierelor pe un server propriu sau prin intermediul unor sisteme de fișiere distribuite, precum Amazon S3, Google Cloud Storage sau Azure Blob Storage. Aceste soluții oferă control total, însă presupun costuri suplimentare în ceea ce privește securitatea, mentenanța și scalarea automată. În schimb, serviciile cloud gestionate se remarcă prin latență redusă, redundanță

implicită și integrare ușoară cu rețelele de livrare de conținut (CDN), reducând astfel timpul de încărcare și îmbunătățind experiența utilizatorului final.

În cadrul proiectului curent, s-a optat pentru utilizarea **Firebase Storage**, un serviciu oferit de Google ce combină avantajele unei infrastructuri scalabile cu o interfață de programare prietenoasă. Această alegere a fost motivată de mai mulți factori complementari. În primul rând, Firebase permite încărcarea și accesarea rapidă a fișierelor prin URL-uri securizate, oferind suport nativ pentru controlul accesului pe baza autentificării. În al doilea rând, integrarea directă cu front-end-ul React facilitează manipularea fișierelor fără a complica logica aplicației. Nu în ultimul rând, serviciul oferă performanță ridicată prin utilizarea infrastructurii globale Google, cu replicare automată și distribuție optimizată geografic.

Deși alternative precum Amazon S3 oferă o suită mai amplă de funcționalități avansate pentru scenarii enterprise, Firebase se aliniază mai bine cu obiectivele proiectului curent, orientat spre simplitate, costuri reduse și rapiditate în dezvoltare. Limitările sale, precum posibilitățile mai restrânse de configurare fină sau lipsa unui control al obiectelor, nu afectează semnificativ scenariul de utilizare prevăzut, ceea ce consolidează justificarea acestei alegeri.

Această alegere aduce multiple avantaje:

- **Scalare automată**: Firebase gestionează automat volumul traficului și al fișierelor, fără a necesita intervenție manuală sau configurare suplimentară a infrastructurii.
- **Acces prin linkuri publice securizate**: După încărcarea unui fișier, acesta primește un URL unic, optimizat pentru livrare rapidă prin rețele CDN (Content Delivery Network), asigurând timpi de încărcare scăzuți la nivel global.
- **Integrare facilă cu frontend-ul**: Fișierele sunt accesate direct din interfața React fără a solicita serverul backend, reducând astfel încărcarea pe microservicii și îmbunătățind performanța aplicației.
- **Costuri predictibile**: Firebase oferă un model de tarifare bazat pe volum și trafic, fiind ideal pentru proiecte cu creștere graduală.
- **Suport pentru reguli de acces**: Platforma permite definirea de reguli de acces bazate pe utilizator, tip de fișier sau momentul încărcării, sprijinind scenarii avansate de securitate.

4.2.5. Interfață grafică în React

Alegerea framework-ului **React** pentru dezvoltarea interfeței de utilizator reflectă o orientare strategică spre modernitate, performanță și extensibilitate. React, dezvoltat inițial de Facebook și menținut ca proiect open-source, a devenit în ultimul deceniu unul dintre cele mai utilizate instrumente pentru construcția de aplicații web interactive [15]. Structura sa component-based permite reutilizarea logicii vizuale și favorizează dezvoltarea incrementală a funcționalităților.

În contextul platformei **Vendify**, React oferă suport nativ pentru construirea de aplicații *Single Page Application* (SPA), permițând actualizarea dinamică a conținutului fără reîncărcarea întregii pagini, ceea ce contribuie la o experiență fluentă și intuitivă pentru utilizator. Mai mult, ecosistemul vast și matur în jurul acestui framework (inclusiv biblioteci pentru routing, gestionarea stării și interacțiuni asincrone) a susținut dezvoltarea rapidă a unei interfețe adaptabile și personalizabile, în acord cu cerințele variate ale comercianților online.

- **Componentizare**: Interfața este construită din componente reutilizabile care pot fi gestionate și testate izolat, favorizând mentenanța și extensibilitatea.

- **Stare locală și globală**: Gestionarea eficientă a stării aplicației se realizează prin `React Hooks` pentru starea locală și biblioteci precum Redux pentru starea globală.
- **Integrare cu backend-ul**: React consumă datele expuse de microserviciile backend prin apeluri HTTP asincrone (ex: `axios`, `fetch`), permițând un decuplaj clar între logică și interfață.
- **Rutare avansată**: Cu ajutorul `React Router`, aplicația suportă navigare dinamică bazată pe URL-uri, inclusiv rute parametrizate, nested sau protejate (guarded).
- **Ecosistem matur**: Fiind unul dintre cele mai populare framework-uri frontend, React beneficiază de o comunitate extinsă, numeroase resurse open-source și suport larg din partea bibliotecilor UI precum `Material UI`, `Tailwind`, `ShadCN`, etc.

Această selecție tehnologică reflectă o abordare echilibrată între nevoile de dezvoltare rapidă și cerințele de scalabilitate, fiind adaptată în special pentru scenariile comerciale cu trafic variabil și cerințe dinamice.

4.3. Arhitectura Conceptuală a Sistemului

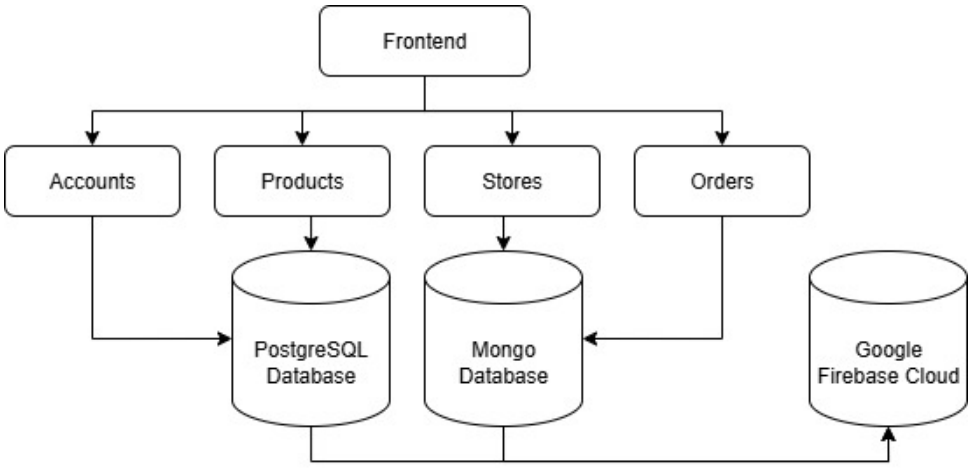

Figura 4.3: Arhitectura conceptuală a sistemului Vendify

Structura ilustrată în figura 4.3 reflectă o arhitectură distribuită, centrată pe principii moderne de modularitate și separare a responsabilităților. Scopul acestei organizări este de a oferi o infrastructură scalabilă, adaptabilă și ușor de întreținut pentru dezvoltarea unei soluții eCommerce cu mai multe magazine.

La un nivel conceptual, sistemul este compus din patru microservicii de bază: *Accounts*, *Products*, *Stores* și *Orders*, fiecare fiind responsabil pentru un domeniu logic distinct. Aceste servicii sunt accesate printr-o interfață web unificată, construită cu ajutorul React, care permite interacțiuni dinamice și o experiență fluentă pentru utilizatorul final.

Din perspectiva persistenței datelor, platforma combină două paradigme diferite de stocare. Microserviciile *Accounts* și *Products* utilizează baze de date relaționale (**PostgreSQL**), potrivite pentru modelarea entităților structurale și a relațiilor dintre acestea. În schimb, modulele *Stores* și *Orders* apelează la o bază de date documentară (**MongoDB**), aleasă pentru flexibilitatea în manipularea obiectelor ierarhice și neuniforme (precum teme personalizate, coșuri sau sesiuni de utilizator).

Componentele media, precum logo-urile, imaginile de produs sau bannerele, sunt externalizate prin integrarea cu **Google Firebase Storage**, care permite livrarea eficientă a fișierelor statice și susține scalabilitatea aplicației în contexte de trafic ridicat.

Această arhitectură oferă un echilibru între controlul asupra datelor, viteza de livrare și flexibilitatea în extinderea ulterioară a platformei. Prin separarea clară a preocupărilor și utilizarea unor soluții specializate pentru fiecare tip de resursă, Vendify poate fi adaptat ușor la cerințe noi fără a compromite coerența structurală a sistemului.

4.4. Microserviciul de Conturi și Autentificare

Serviciul de conturi și autentificare reprezintă componenta responsabilă cu gestionarea utilizatorilor platformei și a sesiunilor lor de lucru. Este unul dintre microserviciile fundamentale ale arhitecturii Vendify, întrucât asigură accesul securizat la resurse și personalizează interacțiunile utilizatorilor cu sistemul.

4.4.1. Responsabilități

Serviciul gestionează două dimensiuni cheie:

- **Identitate și autentificare**: Verificarea și controlul accesului pe baza unor credențiale sigure.
- **Monitorizare comportamentală**: Colectarea de date despre interacțiunile utilizatorilor, indiferent dacă aceștia sunt autentificați sau nu.

Prin această separare, serviciul susține atât personalizarea experienței utilizatorului, cât și funcționalități de analiză și marketing.

4.4.2. Autentificare bazată pe JSON Web Tokens (JWT)

Pentru a asigura un mecanism de autentificare scalabil și compatibil cu arhitectura distribuită a platformei, sistemul utilizează standardul **JWT** (JSON Web Token). Această soluție permite verificarea identității utilizatorilor într-un mod stateless, fără a păstra sesiuni active pe server.

Un JWT este alcătuit din trei părți codificate în Base64:

- **Header**: specifică tipul token-ului (JWT) și algoritmul de semnare (ex: HS256).
- **Payload**: conține revendicările (*claims*) referitoare la identitatea și drepturile utilizatorului.
- **Signature**: semnătura digitală generată pe baza celorlalte două părți și a unei chei secrete.

Structura standard a unui JWT este ilustrată în Tabelul 4.2:

Câmp	Tip	Descriere
`sub`	String	Subiectul token-ului – de obicei ID-ul unic al utilizatorului
`exp`	Timp Unix	Timpul de expirare al token-ului
`iat`	Timp Unix	Timpul la care token-ul a fost emis
`iss`	String	Emitentul token-ului (de ex. numele aplicației)
`aud`	String	Destinatarul așteptat al token-ului

Tabela 4.2: Exemple de câmpuri standard dintr-un token JWT

La fiecare cerere HTTP, token-ul este transmis în antetul de autorizare: `Authorization - Bearer <token>`

Această abordare facilitează:

- **Separarea autentificării de microservicii**: validarea are loc local, fără consultarea unui server central de sesiuni.
- **Securitate crescută**: token-ul este semnat criptografic și are o durată limitată de viață.
- **Scalabilitate orizontală**: microserviciile pot fi replicate fără a necesita sincronizarea stării sesiunilor.

4.4.3. Gestionarea Sesiunilor prin Cookie-uri

Pe lângă utilizarea token-urilor JWT pentru autentificare, platforma va utiliza un mecanism complementar de urmărire a comportamentului utilizatorilor, stocat în browser, sub forma unui cookie HTTP.

Acest identificator este generat la prima interacțiune a utilizatorului cu aplicația și are următoarele caracteristici:

- Este unic per utilizator și magazin.
- Este transmis automat de browser în fiecare cerere ulterioară.
- Nu conține date sensibile, ci doar un identificator de corelare cu datele de analiză.

Mecanismul de sesiune permite platformei să colecteze date utile despre comportamentul utilizatorilor, chiar și în lipsa autentificării:

- Paginile vizitate, produsele accesate sau acțiunile efectuate (ex: adăugare în coș).
- Durata și frecvența vizitelor.
- Conversii anonime sau potențial ratate.
- Corelarea ulterioară cu contul utilizatorului, dacă acesta se autentifică.

Această abordare oferă:

- compatibilitate nativă cu browserele;
- analiză comportamentală chiar și fără autentificare;
- persistență între sesiuni pentru personalizare și analiză.

4.4.4. Rol în Arhitectura Platformei

Fiind un punct central în infrastructura de microservicii, serviciul de conturi este separat pentru a respecta principiile de responsabilitate unică și scalare independentă. Această separare permite:

- Aplicarea unor politici stricte de securitate și control al accesului.
- Auditarea și monitorizarea izolată a autentificării și comportamentului utilizatorului.
- Reutilizarea mecanismelor de autentificare în aplicații externe sau viitoare extensii.

4.4.5. Importanță în Ecosistemul Vendify

Prin gestionarea accesului, securității și sesiunilor active, serviciul de conturi constituie fundamentul pentru majoritatea operațiunilor efectuate de utilizatori. Este componenta care asigură coerența și integritatea identității în întregul ecosistem al platformei Vendify.

4.5. Microserviciul de Comenzi

Microserviciul de comenzi este responsabil pentru gestionarea procesului de achiziție, de la adăugarea produselor în coșul de cumpărături până la finalizarea comenzii. Este o componentă critică a platformei Vendify, întrucât asigură fluxul principal de generare venit pentru magazinele online.

4.5.1. Responsabilități

Acest microserviciu gestionează:

- Coșul de cumpărături al fiecărui utilizator, permițând adăugarea, modificarea sau eliminarea produselor.
- Procesul de plasare a unei comenzi, inclusiv stocarea detaliilor de livrare și a metodei de plată selectate.
- Lista de dorințe (wishlist), cu posibilitatea de a transfera produsele în coș pentru achiziție ulterioară.
- Starea comenzilor (ex: în așteptare, procesare, livrată), oferind suport pentru urmărirea acestora de către utilizatori.

4.5.2. Rol în Arhitectura Platformei

Prin separarea logicii de cumpărare într-un microserviciu dedicat, platforma câștigă:

- Independență în dezvoltarea și scalarea funcționalităților comerciale, fără a afecta alte componente.
- Claritate în gestionarea fluxurilor tranzacționale și trasabilitate completă a comenzilor.
- Flexibilitate în extinderea viitoare a opțiunilor de livrare, plată sau integrare cu servicii externe.

4.5.3. Importanță în Ecosistemul Vendify

Serviciul de comenzi se află în centrul fluxului operațional al platformei Vendify, transformând intențiile utilizatorilor în tranzacții reale. Prin gestionarea eficientă a comenzilor și a stării acestora, platforma asigură un proces de cumpărare clar, fiabil și ușor de urmărit pentru clienți și administratori de magazin.

4.6. Microserviciul de Magazine

Microserviciul de magazine este responsabil pentru gestionarea instanțelor individuale de magazine online, fiecare cu o identitate vizuală și configurare proprie. Această componentă oferă utilizatorilor posibilitatea de a crea, personaliza și accesa propriile magazine în cadrul platformei Vendify.

4.6.1. Responsabilități

Acest microserviciu gestionează:

- Crearea unui magazin nou pentru un utilizator autentificat.
- Configurarea identității vizuale a magazinului: nume, URL dedicat, logo, banner și temă grafică.
- Salvarea și expunerea categoriilor de produse asociate magazinului.
- Furnizarea de informații de contact și linkuri sociale personalizate.

4.6.2. Rol în Arhitectura Platformei

Separarea logicii de gestionare a magazinelor într-un microserviciu dedicat permite:

- Extinderea ușoară a funcționalităților de personalizare și branding pentru fiecare magazin.
- Scalarea independentă în funcție de numărul de magazine active.
- Aplicarea tematicii personalizate direct în interfața frontend, pe baza datelor livrate de acest serviciu.

4.6.3. Importanță în Ecosistemul Vendify

Prin acest microserviciu, platforma Vendify devine capabilă să ofere o experiență multi-store, în care fiecare utilizator poate administra unul sau mai multe magazine distincte, complet personalizabile. Este elementul care asigură diferențierea și autonomia fiecărui magazin din punct de vedere vizual și funcțional.

4.7. Microserviciul de Produse

Microserviciul de produse este responsabil pentru gestionarea datelor comerciale esențiale despre produsele disponibile în cadrul fiecărui magazin. Acesta permite administrarea inventarului, prezentarea detaliată a produselor și aplicarea automată a reducerilor comerciale.

4.7.1. Responsabilități

Acest microserviciu gestionează:

- Crearea, modificarea și ștergerea produselor asociate unui anumit magazin.
- Afișarea produselor filtrate după magazin, categorie sau stoc disponibil.
- Aplicarea automată a reducerilor procentuale, fără intervenție manuală.
- Stocarea datelor vizuale și descriptive ale produsului: denumire, imagini, prețuri, descriere, stoc, recenzii.

4.7.2. Rol în Arhitectura Platformei

Microserviciul de produse funcționează ca sursă principală de date pentru toate componentele comerciale ale platformei. Separarea într-un microserviciu dedicat permite:

- Scalarea independentă a operațiilor de listare, căutare și actualizare produse.
- Administrarea reducerilor și evidențierea produselor promovate fără a afecta celelalte microservicii.
- Optimizarea interacțiunii cu bazele de date relaționale (PostgreSQL) pentru interogări rapide și precise.

4.7.3. Importanță în Ecosistemul Vendify

Microserviciul de produse susține întregul proces de vânzare, oferind date actualizate și coerente despre oferta disponibilă în fiecare magazin. Prin integrarea reducerilor automat

4.8. Interfața grafică

Interfața aplicației **Vendify** este construită folosind framework-ul *React*, având o structură modulară ce reflectă o separare clară a responsabilităților și facilitează reutilizarea componentelor. Această abordare asigură o dezvoltare coerentă, întreținere facilă

și integrare eficientă cu serviciile backend prin API-uri REST.

4.8.1. Structură Generală

Interfața platformei este organizat în componente reutilizabile, fiecare responsabilă pentru o funcționalitate specifică (ex: bara de navigație, listă de produse, coș de cumpărături, formular de checkout). Această abordare component-based permite o dezvoltare scalabilă și ușor de întreținut.

Platforma este structurată astfel încât fiecare magazin să aibă o identitate vizuală proprie, reflectată în adresă (ex: `/exclusive`, `/urbanwear`) și în tematica vizuală (culori, fonturi, imagini).

4.8.2. Funcționalități Principale

Interfața grafică a platformei Vendify este proiectată pentru a oferi o experiență completă, intuitivă și adaptabilă atât pentru utilizatorii finali (clienți), cât și pentru administratorii de magazine. Printre funcționalitățile principale se regăsesc:

- **Navigare multi-store dinamică**: accesarea directă a unui magazin printr-un URL dedicat (ex: `/exclusive`) cu tematică personalizată aplicată automat.
- **Gestionarea conturilor și autentificare**: utilizatorii pot crea conturi, se pot autentifica securizat prin token-uri JWT și pot accesa funcționalități personalizate (ex: wishlist, comenzi).
- **Vizualizarea și filtrarea produselor**: afișarea produselor organizate pe categorii, evidențierea celor cu reduceri, sortări și filtre dinamice pentru o căutare eficientă.
- **Reduceri automate**: produsele incluse într-o campanie de reducere sunt afișate cu prețul actualizat și vechiul preț tăiat, fără intervenție manuală.
- **Gestionarea coșului și wishlist-ului**: utilizatorii pot adăuga sau elimina produse din coș, salva produse preferate și le pot transfera în coș ulterior.
- **Procesul de checkout**: formular complet pentru plasarea unei comenzi, cu selecția metodei de plată și recapitularea comenzii.
- **Tematizare vizuală dinamică**: fiecare magazin are o identitate vizuală proprie, cu logo, banner, paletă de culori, fonturi și stiluri configurabile prin backend și aplicate instant în interfață.
- **Feedback și evaluare produse**: utilizatorii pot oferi rating-uri produselor achiziționate, contribuind la credibilitatea acestora.
- **Colectarea datelor analitice**: sistemul urmărește acțiunile utilizatorilor (vizualizări, adăugări în coș, comenzi), permițând calcularea ratelor de conversie și identificarea produselor performante.
- **Responsivitate completă**: interfața este optimizată pentru utilizare pe dispozitive mobile, tablete și desktop, menținând o experiență coerentă pe toate rezoluțiile.

4.8.3. Personalizarea Tematică a Magazinului

Pentru a permite fiecărui magazin să reflecte brandul său vizual, aplicația suportă un sistem de temă configurabilă. Printre opțiunile disponibile se numără:

- Sigla personala al fiecarui magazin.
- Poster-ul de pe pagina de acasa.
- Culoarea textelor folosite in website.
- Culoarea navbar-ului și efecte de blur.
- Selectarea fontului principal și a stilului textelor.

- Configurarea razei de rotunjire a butoanelor.

Aceste opțiuni sunt persistate în baza de date și aplicate dinamic la fiecare accesare a magazinului, fără a fi necesară reîncărcarea aplicației.

4.8.4. Interacțiune cu Backend-ul

Aplicația frontend comunică cu microserviciile backend prin apeluri HTTP, utilizând token-uri JWT pentru autentificare și autorizare. Fiecare microserviciu este accesat prin endpointuri bine definite, respectând convențiile REST.

Operațiile de comunicare urmează o structură predictibilă:

- `GET`: pentru obținerea de date (ex: listarea produselor, afișarea comenzilor, încărcarea temei unui magazin).
- `POST`: pentru operații de creare (ex: înregistrarea unui cont, plasarea unei comenzi, adăugarea unui produs).
- `PUT`: pentru actualizarea resurselor existente (ex: modificarea unui produs, salvarea preferințelor de temă).
- `DELETE`: pentru ștergerea de entități (ex: eliminarea unui produs din wishlist sau coș).

4.8.5. Considerații de Design și Accesibilitate

Interfața este proiectată cu accent pe ușurința în utilizare, claritate vizuală și compatibilitate cu dispozitive mobile. Se urmărește respectarea principiilor de accesibilitate (contrast, mărime fonturi, navigare intuitivă), astfel încât platforma să fie ușor utilizabilă de o gamă largă de utilizatori.

Capitolul 5

Proiectare de detaliu și implementare

După fundamentarea arhitecturii conceptuale și descrierea principalelor funcționalități ale platformei **Vendify**, acest capitol se concentrează pe detaliile concrete de proiectare și implementare. Sunt prezentate modelele de date, fluxurile logice esențiale, tehnologiile utilizate și deciziile practice care au ghidat realizarea fiecărei componente.

Platforma a fost proiectată pe o arhitectură modulară, orientată pe microservicii, fiecare responsabil pentru un set bine definit de funcționalități. Comunicarea între componente este realizată prin API-uri REST autentificate, folosind mecanisme moderne de securizare, precum token-urile JWT.

5.1. Arhitectura Logică a Sistemului

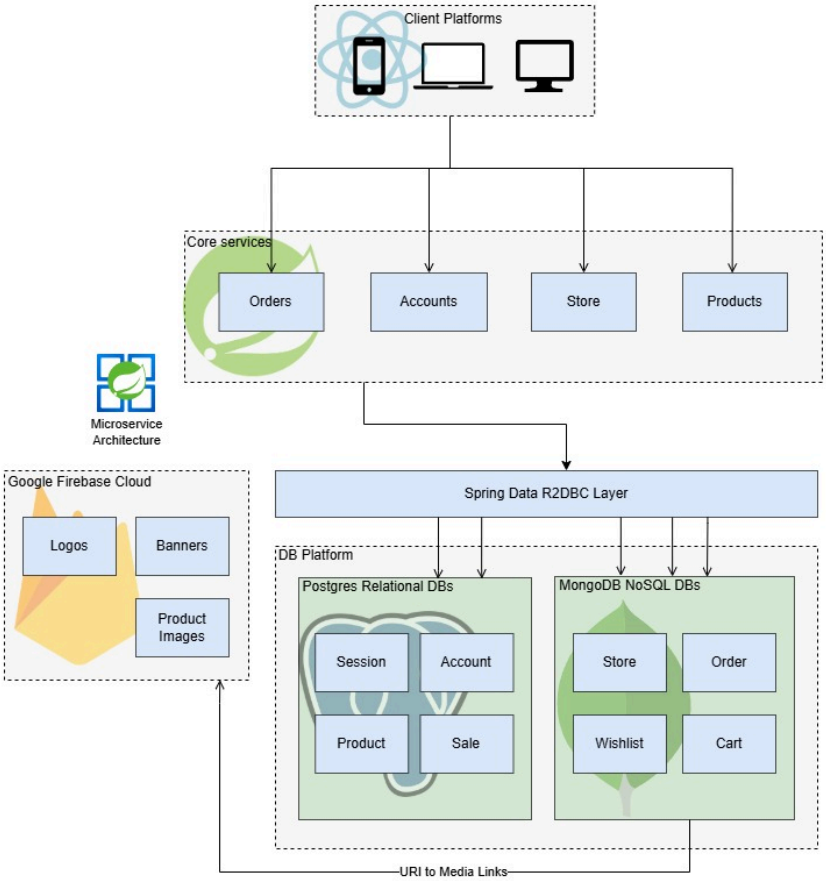

Figura 5.1: Arhitectura logică a sistemului Vendify

Înainte de a analiza în detaliu implementarea fiecărui microserviciu, este esențial să oferim o privire de ansamblu asupra modului în care acestea colaborează în cadrul ecosistemului **Vendify**. Figura 5.1 oferă o reprezentare logică a arhitecturii platformei, reliefând straturile funcționale principale și interacțiunile dintre componentele cheie. Această diagramă servește drept cadru de referință pentru înțelegerea modului în care sunt distribuite responsabilitățile în sistem și va ghida lectura secțiunilor următoare.

- **Nivelul de prezentare (Client Platform)**: Include interfețele grafice cu care interacționează utilizatorii, aplicații web responsive dezvoltate în React, compatibile cu dispozitive mobile și desktop.
- **Nivelul de servicii (Core Services)**: Patru microservicii distincte: *Orders*, *Accounts*, *Store* și *Products*, gestionează logica de afacere specifică fiecărei entități. Fiecare microserviciu este autonom și poate comunica cu celelalte prin API-uri bine definite.
- **Stratul de acces la date (Spring Data R2DBC Layer)**: Asigură abstractizarea interacțiunii cu bazele de date relaționale și non-relaționale, oferind suport reactiv pentru comunicații asincrone.
- **Platforma de stocare a datelor (DB Platform)**: Utilizează două tipuri de baze de date:
  - **PostgreSQL**: pentru entități structurale și relaționale.
  - **MongoDB**: pentru date dinamice și ierarhie complexa (ex: coșuri, liste de dorințe, teme).
- **Stocare media (Firebase Cloud)**: Fișierele statice (imagini, bannere) sunt salvate în Firebase Storage, permițând livrare rapidă, scalare automată și integrare directă cu aplicația.

Această arhitectură permite scalarea independentă a fiecărei componente, reducerea timpului de răspuns în aplicație și o dezvoltare incrementală fără impact asupra întregului sistem.

5.2. Date predefinite pentru testare și demonstrație

Pentru a demonstra funcționalitatea completă a platformei, toate microserviciile dispun de date predefinite, încărcate automat la inițializarea aplicației. Aceste date permit simularea unor scenarii reale de utilizare, precum:

- înregistrarea și autentificarea utilizatorilor;
- gestionarea magazinelor și temelor vizuale;
- afișarea produselor, reducerilor și statisticilor;
- plasarea comenzilor și urmărirea acțiunilor din sesiuni.

Popularea se realizează prin fișiere SQL (pentru PostgreSQL) sau JSON (pentru MongoDB), în funcție de microserviciu. În secțiunile următoare sunt prezentate exemple reprezentative pentru fiecare serviciu în parte.

5.3. Microserviciul de Conturi – Modelare și Flux Logic

Microserviciul de **Conturi** are un rol central în arhitectura platformei **Vendify**, responsabil de gestionarea utilizatorilor, sesiunilor și autentificării. Acesta asigură atât procesele critice de securitate (login, generare token-uri, logout), cât și colectarea datelor de analiză privind comportamentul utilizatorilor în cadrul magazinelor.

5.3.1. Modelarea Entităților Utilizatori și Sesiuni

Din punct de vedere logic, microserviciul gestionează două entități esențiale:

Utilizatorul (User) reprezintă o persoană înregistrată în platformă, asociată unui magazin (`storeId`). Fiecare utilizator este identificat printr-un set minim de atribute: nume, prenume, e-mail, număr de telefon și parolă.

Pentru o mai bună structurare logică, datele colectate din cadrul unui utilizator sunt organizate în tabelul 5.1:

Atribut	**Descriere**
`id`	Identificator unic al utilizatorului în sistem.
`firstName`	Prenumele utilizatorului.
`lastName`	Numele de familie al utilizatorului.
`email`	Adresa de email, utilizată la autentificare.
`password`	Parolă stocată (criptată) pentru autentificare.
`phoneNumber`	Număr de telefon folosit ca alternativă la autentificare.
`storeId`	Identificatorul magazinului la care aparține utilizatorul.

Tabela 5.1: Atributele entității `User`

Sesiunea (Session) reflectă o interacțiune activă a unui utilizator sau vizitator anonim cu un magazin. Aceasta stochează informații precum: începutul și sfârșitul sesiunii, paginile vizitate, acțiunile efectuate și token-urile de autentificare.

Sesiunile pot fi atât autentificate, cât și anonime, oferind o acoperire completă asupra traficului din platformă.

Pentru o mai bună structurare logică, datele colectate în cadrul unei sesiuni sunt organizate în tabelul 5.2:

Atribut	**Descriere**
`id`	Identificator unic al sesiunii.
`userId`	ID-ul utilizatorului asociat, dacă este autentificat.
`storeId`	Magazinul pentru care a fost inițiată sesiunea.
`sessionCookie`	Identificatorul cookie-ului de sesiune din browser.
`startTime`	Momentul inițierii sesiunii.
`lastActivity`	Timpul ultimei activități efectuate în sesiune.
`endTime`	Finalul sesiunii (poate fi nul dacă este activă).
`accessToken`	Token JWT folosit pentru autorizare temporară.
`refreshToken`	Token folosit pentru reînnoirea autentificării.
`pagesVisited`	Lista paginilor vizitate în cadrul sesiunii.
`actions`	Lista acțiunilor semnificative (ex: vizualizare produs).

Tabela 5.2: Atributele entității `Session`

Această organizare permite colectarea incrementală de date și susține generarea unor indicatori statistici esențiali pentru optimizarea platformei și înțelegerea comportamentului utilizatorilor.

5.3.2. Date predefinite pentru conturi și sesiuni

Pentru a permite autentificarea și gestionarea accesului în platformă, aplicația definește utilizatori si sesiuni predefinite. Aceștia includ un utilizator cu rol `ADMIN`, asociat unui magazin, precum și utilizatori simpli pentru testarea comenzilor și a sesiunilor.


```
{
  "id": 1,
  "firstName": "Robert",
  "lastName": "Ernst",
  "email": "rernst@gmail.com",
  "password": "pass123",
  "storeId": "0"
}

{
  "id": 1
  "userId": 1,
  "storeId": "6839d4104106e209e93ea39c",
  "sessionCookie": "f47ac10b-58cc-4372-a567-0e02b2c3d479",
  "accessToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0Iiwic3RvcmVJZC
",
  "refreshToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0IiwidHlwIjoiU
LR07E
",
  "startTime": "2025-06-30T12:00:23",
  "lastActivity": "2025-06-30T12:14:57",
  "endTime": "2025-06-30T12:16:28",
  "pagesVisited": ["product/123", "cart"],
  "actions": ["view_product:1", "add_to_cart:2"]
}
```


5.3.3. Fluxuri Logice Principale

Mecanismul de autentificare și autorizare în cadrul platformei **Vendify** este bazat pe paradigma JWT (JSON Web Token), permițând o gestionare stateless a sesiunilor și o verificare rapidă a identității utilizatorului în cadrul microserviciilor.

Un token JWT este compus din trei părți:*Header*, *Payload* și *Signature*, fiind codificat în format Base64. În cadrul platformei se utilizează două tipuri de token:

- **Access Token** – utilizat pentru autorizarea accesului la resurse protejate, are o valabilitate redusă;
- **Refresh Token** – utilizat pentru obținerea unui nou access token fără reautentificare, are o valabilitate extinsă.

Pașii logici ai autentificării:

1. Utilizatorul trimite o cerere POST către `/auth/login/{storeId}`, incluzând în payload adresa de email sau numărul de telefon, parola și un identificator unic de sesiune (`sessionId`).
2. Sistemul determină dacă identificatorul este o adresă de email sau un număr de telefon și caută utilizatorul în baza de date corespunzătoare magazinului (`storeId`).

În cazul în care utilizatorul nu este găsit, se returnează un răspuns de eroare `UserNotFoundException` cu codul `404`.

3. Dacă utilizatorul este găsit, parola transmisă este comparată cu cea din baza de date. În cazul unui conflict, este returnat un răspuns de eroare `InvalidCredentials` cu codul `401`.
4. Se verifică dacă utilizatorul este asociat cu `storeId`-ul specificat în cerere. În caz contrar, se returnează o eroare de tip `UserNotFoundException`.
5. Sistemul caută o sesiune activă corespunzătoare `sessionId`-ului primit. Dacă sesiunea există, aceasta este actualizată cu:
   - `userId` – identificatorul utilizatorului autentificat;
   - `accessToken` – generat pe baza adresei de email și a identificatorului sesiunii;
   - `refreshToken` – generat separat și asociat sesiunii curente.
6. Dacă sesiunea nu există, se creează o sesiune nouă cu valorile de mai sus și se salvează în baza de date.
7. Se returnează un răspuns `LoginResponse` (ilustrat in tabelul 5.4). În caz de eroare, se returnează un obiect `ErrorResponse` (ilustrat in tabelul 5.5) corespunzător cu detalii.

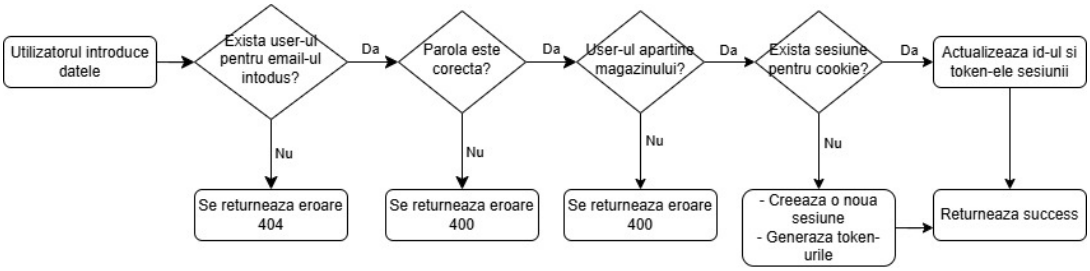

Figura 5.2: Fluxul decizional de autentificare

Diagrama din figura 5.2 prezintă logica decizională din spatele procesului de autentificare. Fiecare pas este condiționat de verificări succesive: existența utilizatorului, corectitudinea parolei, asocierea corectă cu magazinul (`storeId`) și existența unei sesiuni active. În funcție de rezultatele acestor verificări, sistemul decide între continuarea autentificării, inițializarea unei sesiuni noi sau returnarea unui mesaj de eroare.

Pașii logici ai înregistrării:

Procesul de înregistrare urmează o structură similară autentificării, dar include validări suplimentare pentru asigurarea unicității utilizatorului și crearea contului în mod sigur:

1. Se primește o cerere POST la `/auth/register`, care conține în payload:
   - datele utilizatorului: `email`, `phoneNumber`, `firstName`, `lastName`, `password`;
   - identificatorul sesiunii (`sessionId`) în antet.
2. Se verifică dacă există deja un cont cu aceeași adresă de e-mail în cadrul magazinului (`storeId`). Dacă da, se returnează o eroare de tip `UserConflictException` cu codul HTTP `400`.
3. Se efectuează aceeași verificare pentru numărul de telefon. Dacă acesta este deja folosit, se returnează un răspuns de eroare corespunzător.
4. Dacă nu există conflicte, se creează un obiect `User` cu datele oferite, care este salvat în baza de date asociată `storeId`-ului indicat.
5. După înregistrarea utilizatorului, se creează o nouă sesiune cu:

- `userId` – referință la noul utilizator;
- `storeId` – asocierea cu magazinul;
- `sessionId` – identificatorul de sesiune primit inițial.

6. Se generează un `accessToken` bazat pe email și `sessionId`, respectiv un `refreshToken` separat.
7. Sesiunea este actualizată cu cele două token-uri și salvată în baza de date.
8. Sistemul returnează un obiect de tip `LoginResponse` (ilustrat in tabelul 5.3). În caz de eroare, se returnează un `ErrorResponse` (ilustrat in tabelul 5.4) corespunzător cu detalii.

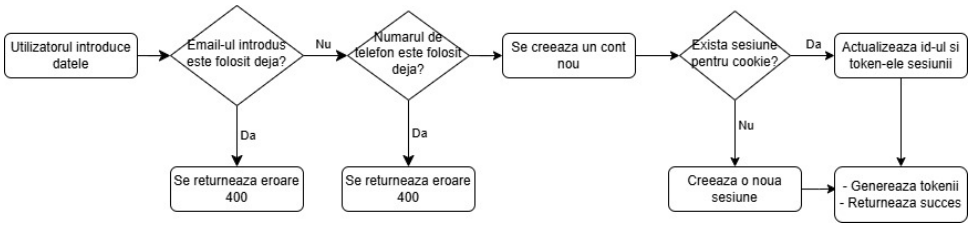

Figura 5.3: Fluxul decizional de inregistrare

Diagrama din figura 5.3 prezintă fluxul decizional al înregistrării în platforma **Vendify**. Sistemul verifică unicitatea adresei de email și a numărului de telefon. Dacă oricare dintre acestea este deja asociată unui cont, se returnează o eroare `400`. În caz contrar, se creează un cont nou și se verifică existența unei sesiuni asociate. Dacă aceasta există, este actualizată cu datele utilizatorului; altfel, se creează o sesiune nouă. În final, se generează token-urile și se returnează un răspuns de succes.

Pașii logici ai actualizării activității utilizatorului:

1. Se primește o cerere de actualizare cu parametrii: `sessionId`, `storeId`, `path` și `action`.
2. Se caută cea mai recentă sesiune asociată `sessionId`.
3. Dacă sesiunea este găsită:
   - Dacă activitatea a avut loc în ultimele 10 minute, se actualizează:
     - `lastActivity` – timpul ultimei activități;
     - `pagesVisited` – se adaugă noul path;
     - `actions` – se adaugă acțiunea efectuată.
   - În caz contrar, sesiunea se închide (setând `endTime`), și se creează o sesiune nouă cu datele actuale.
4. Dacă sesiunea nu este găsită, se creează o sesiune nouă pe baza `storeId`, `path` și `action`.
5. Se salvează modificările în baza de date și se loghează acțiunea pentru panoul de administrare.

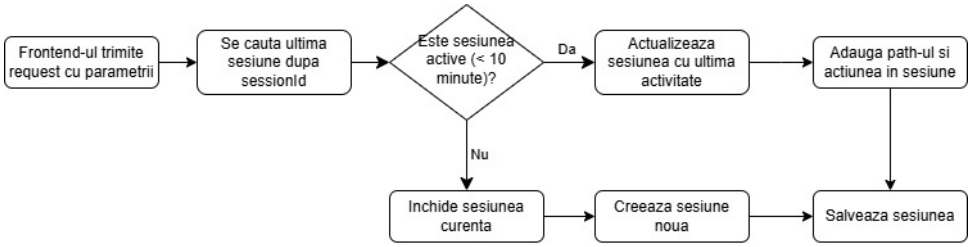

Figura 5.4: Fluxul decizional de actualizare a activității utilizatorului

Diagrama din figura 5.4 evidențiază procesul de înregistrare a unei activități noi într-o sesiune existentă sau de creare a uneia noi, în funcție de timpul scurs de la ultima activitate. Dacă sesiunea este încă activă, se actualizează direct cu noile acțiuni; în caz contrar, sesiunea curentă se închide și se inițializează una nouă.

Structura răspunsului de succes:

Câmp	Descriere
`id`	Identificatorul numeric unic al utilizatorului autentificat
`accessToken`	Token JWT utilizat pentru autorizarea accesului
`refreshToken`	Token pentru reîmprospătarea accesului fără relogare

Tabela 5.3: Structura răspunsului de succes - `LoginResponse`

Structura răspunsului de eroare:

Câmp	Descriere
`error`	Tipul erorii întâlnite (ex: `invalid_email`)
`description`	Detalii suplimentare despre motivul eșecului
`statusCode`	Codul HTTP asociat (ex: 401, 404)

Tabela 5.4: Structura răspunsului în caz de eroare - `ErrorResponse`

Codurile HTTP utilizate:

Cod	Semnificație
`400`	Cerere invalidă: date lipsă sau format incorect (ex: `invalid_email`, `duplicate_phone`).
`401`	Neautorizat: token lipsă sau invalid, acces interzis.
`404`	Resursa inexistentă: utilizatorul nu a fost găsit pentru datele oferite.
`500`	Eroare internă a serverului: excepții neașteptate în timpul procesării cererii.

Tabela 5.5: Coduri de eroare HTTP relevante

5.3.4. Filtrul de Securitate

În vederea asigurării unui control granular asupra securității endpointurilor din microserviciul **Conturi**, am adoptat în proiectul curent o abordare declarativă, bazată pe modelul `Aspect-Oriented Programming (AOP)` [16] oferit de Spring.

Verificările de securitate (precum validarea token-ului, prezența anumitor antete sau consistența traseului de cerere) tind să se repete în cadrul mai multor endpointuri. Pentru a evita duplicarea logicii și a asigura consistența acestor validări, platforma utilizează modelul Aspect-Oriented Programming. Mai exact, am implementat un aspect de securitate care interceptează automat toate metodele annotate cu adnotarea custom `@WMTSecurityMapping`. Această intercepție are loc înainte de execuția efectivă a metodei și permite aplicarea unui filtru global care controlează, într-un mod declarativ, regulile de acces și metadatele cererii. Astfel, securitatea devine un mecanism centralizat, ușor

de configurat și aplicabil unitar în întreaga aplicație, fără a polua logica funcțională a fiecărui controller.

5.3.5. Securitate declarativă per endpoint

Fiecare metodă expusă ca endpoint poate fi adnotată cu o adnotare specială definită în platformă. Aceasta specifică dacă este necesară verificarea token-ului și care este ruta de acces acceptată. În funcție de parametrii adnotării, aspectul va:

- extrage date din anteturile HTTP;
- valida structura cererii;
- autentifica token-ul (dacă este activat);
- bloca cererea dacă oricare dintre validări eșuează.

5.3.6. Anteturile HTTP verificate

Filtrul de securitate se bazează pe antetele transmise în fiecare cerere, evidențiate în tabelul 5.6:

Antet HTTP	**Descriere**
`X-FI-V-PATH`	Calea endpointului accesat, fiind utilizată pentru validarea rutei conform politicii definite în adnotare.
`X-FI-V-SITE-ID`	Identificatorul domeniului pentru care se efectuează cererea (.COM, .RO, .IT)
`X-FI-V-DEVICE`	Identificatorul dispozitivului (browser, mobil etc.), fiind util pentru trasabilitate și detecție fraude.
`X-FI-V-IP`	Adresa IP a utilizatorului, permite auditul traficului și detectarea cererilor anormale.
`Authorization`	Token-ul JWT, necesar doar dacă adnotarea indică `tokenEnabled = true`.

Tabela 5.6: Anteturile HTTP verificate de filtrul de securitate

5.3.7. Fluxul logic al filtrării

La momentul apelării unui endpoint, aspectul de securitate parcurge următorii pași:

1. Verifică prezența anteturilor obligatorii (`PATH`, `SITE-ID`, `DEVICE`, `IP`);
2. Compară calea cererii cu cea definită în adnotare;
3. Dacă este activă verificarea token-ului, extrage și validează JWT-ul;
4. În caz de neconformitate, returnează o eroare de tip `ErrorReponse`(5.4) cu un mesaj explicativ.

Această abordare oferă o serie de beneficii semnificative:

- Reducerea codului duplicat, deoarece validările sunt centralizate într-un singur punct;
- Securitate declarativă, configurabilă la nivel de metodă;
- Ușurință în întreținere, deoarece modificările în politicile de securitate nu necesită refactorizare masivă;

- Extensibilitate, deoarece adnotarea poate fi extinsă în viitor pentru a include scoruri de risc, reguli geo-IP sau tipuri de dispozitive.

5.3.8. Interogări Analitice în Microserviciul de Conturi

Pentru a sprijini procesul decizional al comercianților, platforma **Vendify** colectează și structurează o serie de date relevante din sesiunile de utilizator.

Microserviciul de **Conturi** nu doar gestionează autentificarea și sesiunile, ci și colectează date detaliate despre comportamentul utilizatorilor în cadrul magazinelor online. Printr-un punct de access dedicat, platforma **Vendify** extrage indicatori relevanți despre activitate, implicare și preferințele utilizatorilor, oferind comercianților o perspectivă valoroasă asupra audienței lor.

Indicatori de Activitate și Implicare

Pentru evaluarea generală a traficului și a implicării utilizatorilor, sunt calculați următorii indicatori rapizi:

- Număr total de sesiuni active în ultimele 24 de ore.
- Durata medie a unei sesiuni.
- Număr de utilizatori activi în ultimele 10 minute.
- Rata de utilizatori care revin (calculată ca raport între utilizatorii recurenți și totalul utilizatorilor unici).

Acești indicatori sunt agregați simultan pentru a fi expuși la începutul paginii de analitice, fiind un indicator rapid al stării magazinului.

Distribuția Temporală a Activității

Pentru a înțelege când este cel mai activ traficul utilizatorilor, sistemul oferă:

- Număr de sesiuni pe fiecare oră din ultimele 24 de ore.
- Durata medie a sesiunilor pe oră

Datele sunt generate prin corelarea între orele dintr-o zi și sesiunile din ultimele 24 de ore, ceea ce permite afișarea consistentă chiar și în absența completă a traficului pentru un interval orar.

Tipologia Sesiunilor

Un aspect important pentru comercianți este diferențierea între utilizatori. Printre una dintre interogări, sesiunile din ultimele 24 de ore sunt grupate în:

- `Logged-in`, fiind sesiuni în care utilizatorul este autentificat.
- `Non logged-in`, fiind sesiuni anonime.

Aceste date sunt utile pentru a înțelege proporția traficului generat de utilizatorii existenți față de vizitatori noi.

Acțiuni asupra Produselor

Comportamentul utilizatorilor este urmărit prin acțiunile înregistrate în câmpul `actions[]` din entitatea `Session`. Sunt extrase următoarele statistici:

- Cele mai vizualizate produse
- Cele mai adăugate produse în coș
- Produsele cele mai salvate în wishlist

Fiecare acțiune este salvată sub forma unui string de tip `actionType:productId`, iar extragerea valorilor se face prin funcții SQL, pentru a permite agregări pe produs.

Analiza Utilizatorilor Activi

Sunt calculate două metrice suplimentare pentru a evidenția utilizatorii cu cel mai mare impact:

- Utilizatorii cu cel mai mare timp total petrecut în sesiuni.
- Utilizatorii cei mai activi și pagina lor favorită.

În al doilea caz, pe lângă calculul timpului total, se extrage și pagina cea mai frecvent vizitată, utilizând o subinterogare pe lista `pages_visited`.

Pagini Cele Mai Vizitate

Platforma monitorizează și frecvența vizitării paginilor pentru fiecare magazin, oferind topul celor mai accesate rute (ex: `/product/123`, `/checkout`):

- Cele mai vizitate pagini.

Această funcționalitate permite comercianților să înțeleagă traseul tipic al utilizatorilor în cadrul magazinului și să optimizeze conținutul în consecință.
Datele colectate sunt vizualizate în grafice de analiză destinat comercianților, care permite monitorizarea în timp real a comportamentului utilizatorilor.

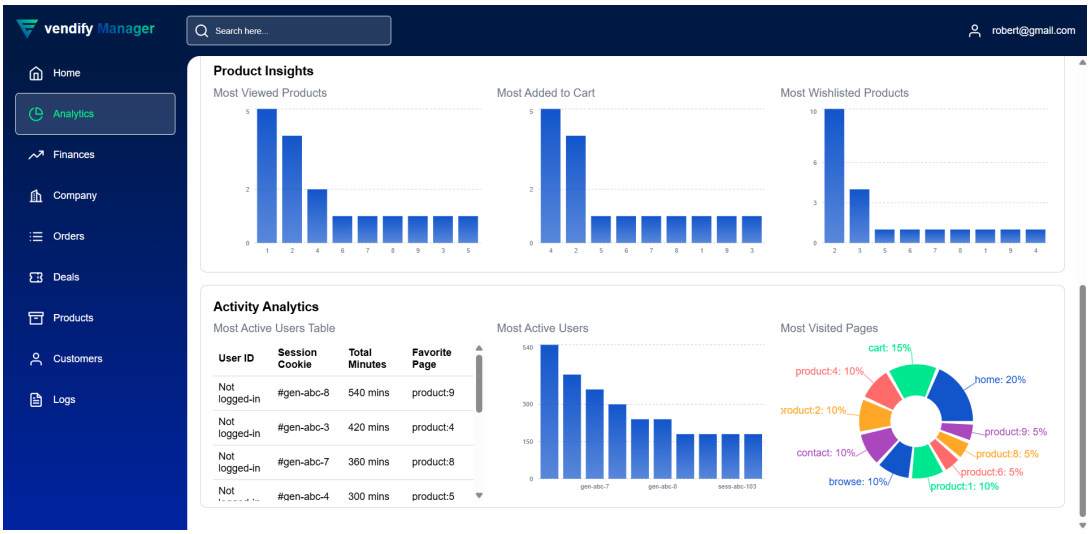

Figura 5.5: Pagina cu datele analitice ale magazinului

Figura 5.5 prezintă interfața analitică a platformei, în care comercianții pot urmări în timp real indicatori esențiali precum cele mai vândute produse, rata de conversie sau evoluția comenzilor. Aceste date sprijină luarea deciziilor informate în procesul de optimizare a ofertelor și ajustare a strategiei comerciale.

Aceste vizualizări oferă comercianților informații esențiale pentru optimizarea conținutului și îmbunătățirea ratei de conversie.

5.3.9. Avantaje Arhitecturale

Din punct de vedere teoretic, microserviciul de **Conturi** oferă următoarele beneficii arhitecturale:

- **Izolarea logicii de autentificare și autorizare**, facilitând întreținerea și extinderea separată;
- **Centralizarea datelor despre comportamentul utilizatorilor**, ceea ce permite corelarea ușoară a sesiunilor și activităților;
- **Model reactiv**, bazat pe R2DBC, pentru procesarea asincronă a sesiunilor și scalabilitate ridicată;
- **Flexibilitate în extinderea logicii de securitate**, inclusiv prin adăugarea de autentificare multiplă, restricții geografice sau sisteme de scoring pentru fraudă.

5.3.10. Concluzie

Microserviciul de **Conturi** constituie fundamentul mecanismelor de autentificare și autorizare în platforma **Vendify**, dar și un nucleu esențial pentru colectarea și structurarea datelor comportamentale ale utilizatorilor. Designul său modular, bazat pe R2DBC și JWT, permite atât protecția accesului, cât și monitorizarea granulară a sesiunilor. Această componentă este indispensabilă pentru personalizarea experienței utilizatorului, generarea de analize în timp real și extinderea viitoare a capabilităților de securitate.

5.4. Microserviciul de Magazine – Gestionarea și Configurarea Magazinelor

Microserviciul de **Magazine** este responsabil pentru gestionarea logicii legate de magazinele online create în cadrul platformei **Vendify**. Acesta permite comercianților să creeze, personalizeze și publice magazine proprii, fiecare cu identitate vizuală unică, setări tematice și informații de contact. De asemenea, microserviciul expune puncte de acces REST pentru interogarea și administrarea acestor date.

5.4.1. Modelarea Entității Store

Entitatea principală a microserviciului este `Store`, care conține toate atributele necesare pentru a descrie un magazin funcțional în platformă. Structura este gândită pentru a oferi:

- flexibilitate în personalizare (teme vizuale, fonturi, culori).
- vizibilitate publică imediată (prin câmpul `path`).
- integrare cu servicii media externe (logo-uri, bannere din Firebase).

Pentru o mai bună structurare logică, datele colectate în cadrul unui magazin sunt organizate în tabelul 5.7:

Atribut	Tip	Descriere
`id`	M	Identificator unic MongoDB pentru magazin
`owner`	M	ID-ul utilizatorului (proprietarului) care a creat magazinul
`name`	M	Numele afișat al magazinului
`path`	M	Ruta unică folosită în URL-ul public (`/magazinul-meu`)
`logo`	O	Link către logo-ul stocat în Firebase
`banner`	O	Link către imaginea de banner
`categories`	O	Listă de categorii asociate magazinului
`theme`	M	Obiect de configurare vizuală (culori, fonturi etc.)
`contactLinks`	O	Obiect ce conține canalele de contact (email, social media)

Tabela 5.7: Atributele entității `Store`

5.4.2. Date predefinite pentru magazine

Pentru a testa funcționalitățile de personalizare și afișare, aplicația definește magazine preexistente. Fiecare magazin este asociat unui cont `ADMIN` și include informații precum numele public, ruta de acces, logo, banner, categorii și setări de temă.

```
{
  "id": "6839d4104106e209e93ea39c",
  "owner": "1",
  "name": "Exclusive",
  "path": "exclusive",
  "logo": "https://firebasestorage.googleapis.com/logo.png",
  "banner": "https://firebasestorage.googleapis.com/banner.jpg",
  "categories": ["Haine", "Încălțăminte", "Accesorii"],
  "theme": {
    "navbar": "blur",
    "font": "Inter",
    "backgroundColor: "#ffffff",
    "primaryColor": "#111111",
    "titleColor": "000000",
    "textSecondaryColor": "#444444",
    "buttonRadius": 20
  },
  "contactLinks": {
    "email": "contact@exclusive.ro",
    "phoneNumber": "+40722222222",
    "instagram": "@exclusive.ro"
  }
}
```

5.4.3. Fluxuri Logice Principale

Crearea unui magazin în cadrul platformei **Vendify** este un proces automatizat și rapid, conceput pentru a permite comercianților să își publice prezența online fără

intervenție tehnică. Magazinul devine accesibil imediat după completarea datelor necesare și apelarea endpoint-ului de creare.

Pașii logici ai procesului de creare:

1. Utilizatorul completează formularul de creare cu informații de bază: nume, rută (`path`), logo, banner, temă vizuală, linkuri de contact.
2. Platforma validează unicitatea rutei (`path`) pentru a evita coliziuni între magazine.
3. Se construiește obiectul `Store` și se persistă în MongoDB.
4. Magazinul este considerat publicat instant — acesta poate fi accesat prin ruta dedicată (ex: `/exclusive`, `/mystore`).

Acest proces elimină nevoia unui pas de aprobare manuală sau configurare suplimentară, oferind comercianților o soluție imediată de expunere.

5.4.4. Personalizarea Magazinului

Platforma oferă comercianților posibilitatea de a adapta vizual și informațional fiecare magazin, fără a fi necesare cunoștințe tehnice. Personalizarea poate viza următoarele aspecte:

- paleta de culori utilizată în interfață;
- stilul și rotunjirea elementelor UI (butoane si carduri);
- fontul principal al magazinului;
- aspectul barei de navigație;
- informațiile de contact afișate (email, telefon, social media).

Aceste opțiuni sunt organizate în două structuri distincte, atașate fiecărui magazin: obiectul `Theme`, care controlează stilul vizual, și obiectul `ContactLinks`, care gestionează datele de contact afișate.

1. Structura Theme

Obiectul `Theme` permite configurarea detaliată a stilului interfeței. Printre câmpurile principale se regăsesc:

- `primaryColor` – definește culoarea principală a aplicației, folosită în butoane, linkuri și accente vizuale;
- `backgroundColor` – setează culoarea generală de fundal a magazinului;
- `titleColor` – controlează culoarea titlurilor și a textelor proeminente;
- `textSecondaryColor` – este utilizată pentru texte explicative sau subtitluri;
- `font` – stabilește fontul global folosit pentru textele magazinului;
- `buttonRadius`, `cardRadius` – permit ajustarea nivelului de rotunjire al butoanelor și cardurilor;
- `navbarStyle` – determină stilul vizual al barei de navigație (ex: transparentă cu efect de blur sau culoare opacă).

2. Structura ContactLinks

Obiectul `ContactLinks` este responsabil pentru afișarea datelor de contact relevante, personalizabile pentru fiecare magazin:

- `email` – adresa de email la care pot fi contactați comercianții;
- `phone` – un număr de telefon pentru suport sau informații suplimentare;
- `facebook`, `twitter`, `instagram` – linkuri către profilurile magazinului pe rețelele sociale.

Această arhitectură permite fiecărui magazin să reflecte identitatea vizuală a brandului său, menținând totodată o experiență unitară în cadrul platformei. Personalizarea este persistentă și poate fi ajustată oricând, pe măsură ce strategia vizuală evoluează.

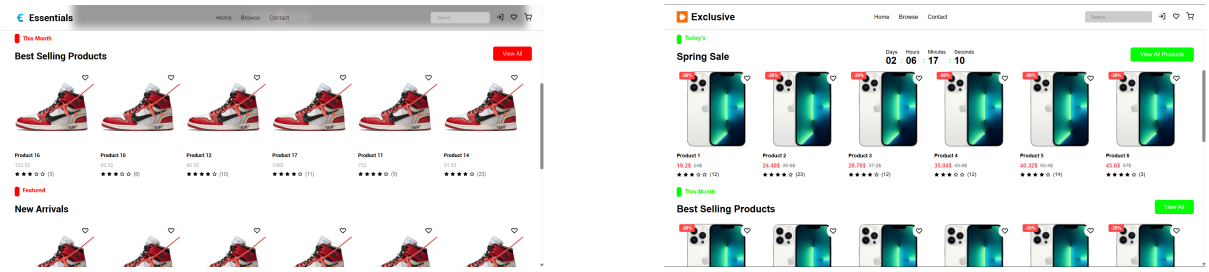

Figura 5.6: Exemple de personalizare vizuală a magazinelor în funcție de tema selectată

Cele doua exemple ilustrate in figura 5.6 evidențiază modul în care două magazine pot adopta stiluri complet diferite folosind același mecanism de personalizare din platformă.

5.4.5. Avantaje Arhitecturale

Microserviciul de **Magazine** oferă o serie de beneficii importante în cadrul arhitecturii generale Vendify:

- **Separare clară a logicii de prezentare și personalizare**, oferind comercianților un spațiu dedicat pentru configurarea identității vizuale și informaționale a magazinului;
- **Flexibilitate ridicată prin atribute opționale**, precum logo, banner, categorii sau linkuri de contact, ce pot fi completate gradual, fără a afecta publicarea magazinului;
- **Publicare rapidă și automată**, fără intervenții manuale sau procese de validare, ceea ce permite lansarea instantanee a magazinului;
- **Integrare nativă cu Firebase pentru resurse media**, asigurând acces rapid la conținut static (imagini, bannere) și simplificarea stocării externe;
- **Suport pentru personalizare extinsă prin obiectul** `Theme`, ce permite adaptarea completă a culorilor, fonturilor și aspectului general;
- **Scalabilitate și autonomie**, fiind construit ca serviciu separat cu bază de date proprie (MongoDB), fără dependențe directe de alte microservicii.

5.5. Microserviciul de Comenzi – Modelare și Fluxuri de Comandă

Microserviciul de **Comenzi** gestionează componentele esențiale legate de procesul de achiziție din cadrul platformei **Vendify**. Acesta acoperă funcționalități precum plasarea comenzilor, manipularea coșului de cumpărături și gestionarea listelor de dorințe. Acest serviciu oferă comercianților o vizibilitate completă asupra comenzilor clienților, precum și unelte de analiză și optimizare a vânzărilor.

Arhitectura sa este construită pe un model reactiv utilizând MongoDB, ceea ce permite operarea fluentă chiar și în condiții de trafic ridicat sau acțiuni simultane din partea utilizatorilor.

5.5.1. Modelarea Entităților Order, CartItem și WishlistItem

Microserviciul de **Comenzi** definește trei entități principale care reflectă diferite etape ale intenției de cumpărare:

1. Order reprezintă o comandă plasată de un utilizator pentru unul sau mai multe produse dintr-un magazin. Include detalii despre client, produse, preț total și adresă de livrare.

Pentru o mai bună structurare logică, datele colectate în cadrul unei comenzi sunt organizate în tabelul 5.8:

Atribut	Tip	Descriere
`id`	M	Identificator unic MongoDB al comenzii
`storeId`	M	ID-ul magazinului din care provin produsele
`customerId`	M	ID-ul utilizatorului care a plasat comanda
`address`	M	Obiect ce conține adresa de livrare (stradă, oraș, cod poștal etc.)
`price`	M	Costul total al comenzii
`items`	M	Listă de produse comandate, fiecare având ID și cantitate
`status`	M	Starea actuală a comenzii (`PLACED`, `SHIPPED`, etc.)
`createdAt`	M	Data și ora plasării comenzii

Tabela 5.8: Atributele entității `Order`

2. CartItem reprezintă o instanță de produs adăugat în coșul de cumpărături, dar necomandat încă.

Pentru o mai bună structurare logică, datele colectate în cadrul unui CartItem sunt organizate în tabelul 5.9:

Atribut	Tip	Descriere
`id`	M	Identificator unic MongoDB al articolului din coș
`storeId`	M	Magazinul asociat
`customerId`	M	Utilizatorul căruia îi aparține coșul
`productId`	M	Produsul adăugat în coș
`quantity`	M	Cantitatea selectată

Tabela 5.9: Atributele entității `CartItem`

3. WishlistItem un produs salvat de utilizator pentru cumpărare ulterioară.

Pentru o mai bună structurare logică, datele colectate în cadrul unei dorințe sunt organizate în tabelul 5.10:

Atribut	Tip	Descriere
`id`	M	Identificator unic MongoDB al articolului din wishlist
`storeId`	M	Magazinul căruia îi aparține produsul
`customerId`	M	Utilizatorul care a salvat produsul
`productId`	M	ID-ul produsului salvat

Tabela 5.10: Atributele entității `WishlistItem`

5.5.2. Date predefinite pentru comenzi, coș și lista de dorințe

Pentru a simula funcționalitățile platformei de comerț, microserviciul de comenzi este populat cu date predefinite relevante: comenzi plasate, produse aflate în coșul de cumpărături și produse salvate în lista de dorințe. Aceste date permit testarea funcțiilor de checkout, actualizare stoc, precum și urmărirea comportamentului utilizatorului.

Comandă (Order) predefinită:

```
{
  "id": "order123",
  "storeId": "6839d4104106e209e93ea39c",
  "userId": 1,
  "products": [
    {
      "productId": 2",
      "quantity": 1
    }
  ],
  "price": 249.99,
  "status": "DELIVERED",
  "createdAt": "2025-07-01T12:00:00",
  "coupon": "SALE20",
  "rated": false
}
```

Produs în coșul de cumpărături (CartItem):

```
{
  "userId": 1,
  "productId": 2,
  "quantity": 2
}
```

Produs în lista de dorințe (WishlistItem):

```
{
  "userId": 1,
  "productId": 2
}
```

5.5.3. Fluxurile Logice Principale

Microserviciul de **Comenzi** integrează trei fluxuri funcționale esențiale pentru experiența de cumpărare:

Fluxul de plasare a unei comenzi:

Acest proces transformă un coș activ într-o comandă completă, implicând validări și persistări:

1. Utilizatorul finalizează procesul de cumpărare, trimițând o cerere POST către endpointul de creare comandă (`/create-order`).

2. Sistemul caută toate produsele din coșul utilizatorului (`customerId`, `storeId`).
3. Dacă coșul este gol, sistemul returnează eroare.
4. Se construiește comanda (`Order`) pe baza produselor din coș și adresei primite.
5. Comanda este salvată în baza de date, iar coșul este golit.
6. Se returnează un mesaj de confirmare, sau obiectul `ErrorResponse` in caz de eroare.

Acest flux este evidențiat in diagrama 5.7:

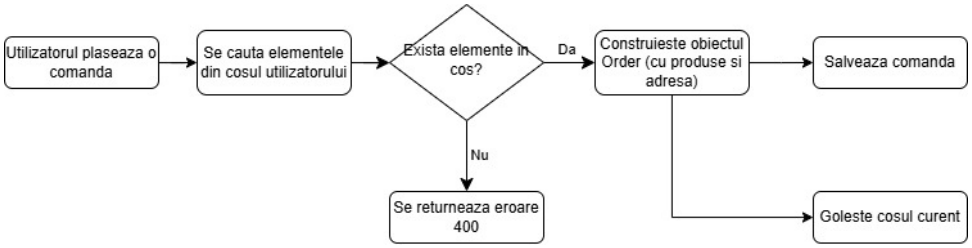

Figura 5.7: Diagrama fluxului de plasare a unei comenzi

Fluxul de gestionare a coșului de cumpărături:

Utilizatorul poate efectua următoarele acțiuni asupra coșului său de cumpărături, iar sistemul răspunde în mod reactiv:

- **Adăugare produs în coș**: dacă produsul există deja în coș (după `customerId`, `storeId`, `productId`), cantitatea este incrementată cu valoarea specificată; altfel, se creează un nou obiect `CartItem`.
- **Actualizare cantitate produs**: se setează o nouă valoare pentru cantitate:
  - dacă noua valoare este diferită de 0, cantitatea produsului este actualizată;
  - dacă noua valoare este 0, produsul este eliminat automat din coș.
- **Eliminare produs din coș**: un anumit produs este șters explicit din coșul utilizatorului fără a schimba alte articole, indiferent de cantitate.
- **Golire completă a coșului**: toate articolele din coșul utilizatorului sunt eliminate, resetând coșul la un stadiu gol.

  Acest flux este evidențiat in diagrama 5.8:

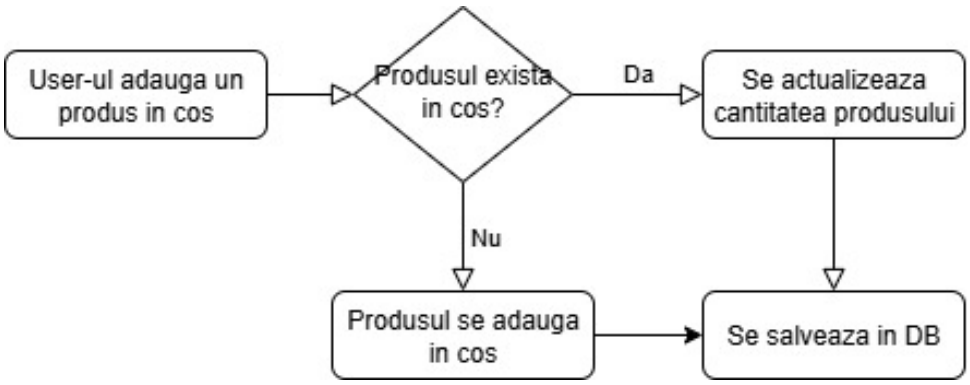

Figura 5.8: Diagrama fluxului de adăugare/actualizare produs în coș

Fluxul de promovare a produselor din wishlist:

Permite conversia produselor salvate în wishlist într-un coș activ:

1. Utilizatorul inițiază o acțiune de promovare a wishlistului.
2. Sistemul recuperează toate `WishlistItem`-urile.
3. Fiecare produs este adăugat în coș (cu cantitate implicită 1).

4. Produsele sunt eliminate din wishlist.

5.5.4. Interogări Analitice în Microserviciul de Comenzi

Microserviciul de **Comenzi** oferă pe lângă funcționalitățile operaționale, un set extins de interogări analitice menite să sprijine comercianții în evaluarea performanței și în luarea deciziilor informate. Aceste interogări sunt accesibile printr-un punct de access dedicat, care centralizează cererile și agregările necesare pentru afișarea indicatorilor în panoul de administrare.

Indicatori Rapizi și Valori Cumulative

Pentru afișarea rapidă a performanței curente, platforma utilizează următorii indicatori:
- Venit total.
- Venit generat în ultimele 24 de ore.
- Număr de comenzi plasate în ultimele 24 de ore.
- Valoarea medie a unei comenzi.

Acești indicatori sunt agregați simultan pentru a fi expuși la începutul paginii de finanțe, fiind un indicator rapid al stării magazinului.

Tendințe pe Ultimele 14 Zile

Pentru analiza evolutivă, sunt implementate trei interogări distincte:
- Venitul zilnic (pentru graficul de profit per zi).
- Numărul comenzilor zilnice (pentru graficul de număr de comenzi per zi).
- Valoarea medie zilnică a comenzilor (pentru graficul de valoare medie per comandă).

Aceste date sunt obținute pe baza câmpului `createdAt` din comenzile plasate, grupate și sortate cronologic prin agregări MongoDB. Lipsurile de date sunt completate dinamic pe client cu valoarea zero, pentru a menține coerența graficelor.

Performanța Produselor

În vederea identificării celor mai performante produse, microserviciul oferă:
- Top-ul produselor în funcție de cantitatea vândută.
- Top produse în funcție de venit generat.
- Metrici de conversie pe produs (cantitate vs. venit).

Agregările folosesc operatori MongoDB precum `$unwind`, `$group` și `$project`, pentru a izola produsele din comenzile multiple și a calcula metrice per produs.

Comportamentul Clienților

Pentru segmentarea bazei de clienți, platforma calculează:
- Clienți noi vs recurenți.
- Top-ul clienților în funcție de numărul de comenzi.
- Top-ul clienților în funcție de profit.

Datele sunt grupate după `customerId`, iar metricile sunt calculate direct din comenzile plasate. Aceste informații sunt utile pentru identificarea clienților fideli sau a potențialelor ținte pentru campanii de loializare.

5.5.5. Avantaje Arhitecturale

Microserviciul de **Comenzi** oferă o serie de beneficii arhitecturale relevante pentru funcționarea scalabilă și robustă a platformei **Vendify**:

- **Separare clară a etapelor de achiziție** – wishlist, coș, comandă – care permite gestionarea granulară a intenției de cumpărare;
- **Model reactiv pe bază de MongoDB**, care susține performanța optimă în condiții de trafic ridicat și acțiuni simultane;
- **Fluxuri logice bine definite**, care acoperă întregul parcurs al utilizatorului într-un magazin, de la selecție până la achiziție;
- **Scalabilitate ridicată** – prin izolare completă de alte microservicii și utilizarea unei baze de date document-based independente;
- **Extensibilitate ușoară** – arhitectura entităților și fluxurilor permite adăugarea de funcționalități precum reduceri, livrare multiplă sau evaluări post-comandă;
- **Corelare cu datele de analiză** – comenzile și comportamentul de cumpărare pot fi corelate cu sesiunile și activitatea utilizatorilor pentru analize complexe.

5.6. Microserviciul de Produse – Gestionarea și Catalogarea Produselor

Microserviciul de **Produse** are rolul central de a gestiona catalogul de produse al fiecărui magazin. Acesta permite adăugarea de produse, clasificarea acestora pe categorii, aplicarea reducerilor automate și oferirea unor interfețe eficiente pentru interogarea produselor disponibile.

Arhitectura sa este bazată pe Spring WebFlux și PostgreSQL accesat prin `R2DBC`, permițând astfel o procesare reactivă, scalabilă și non-blocantă.

5.6.1. Modelarea Entităților: Product și Sale

Entitatea Product descrie atributele esențiale ale unui produs comercializat în platformă.

Atribut	Tip	Descriere
`id`	M	Identificatorul unic al produsului
`store`	M	Magazinul căruia îi aparține produsul
`name`	M	Numele afișat al produsului
`images`	O	Lista de linkuri către imagini din Firebase
`category`	M	Categoria din care face parte produsul
`sizes`	O	Dimensiunile disponibile (ex: S, M, L)
`price`	M	Prețul actual al produsului
`oldPrice`	O	Prețul inițial (util în cazul reducerilor)
`description`	O	Descrierea completă a produsului
`reviews`	O	Scorul de recenzii (1–5)
`stock`	M	Cantitatea disponibilă în stoc

Tabela 5.11: Atributele entității `Product`

Entitatea Sale descrie reducerile automate aplicate unor produse selectate.

Atribut	Tip	Descriere
`id`	M	Identificatorul unic al reducerii
`name`	M	Numele promoției sau campaniei
`store`	M	Magazinul căruia îi este asociată reducerea
`percentage`	M	Procentul de reducere aplicat (0–100)
`productIds`	M	Lista de ID-uri de produse afectate
`endDate`	O	Data expirării reducerii
`status`	M	Statusul reducerii (ex. expirat, activ)

Tabela 5.12: Atributele entității `Sale`

5.6.2. Date predefinite pentru produse și reduceri

Platforma definește produse și reduceri predefinite pentru a simula o gamă variată de oferte disponibile într-un magazin. Produsele conțin informații detaliate despre nume, categorie, preț, stoc, imagini și atribute personalizate (ex: dimensiuni), în timp ce reducerile automate sunt aplicate pe baza unui procent, pentru anumite produse și cu o dată limită de valabilitate.

Produs predefinit:

```
{
  "id": 101,
  "store": "store1",
  "name": "Rochie Elegantă Albastră",
  "images": ["rochie1.png", "rochie1_2.png"],
  "category": "Rochii",
  "price": 249.99,
  "oldPrice": 299.99,
  "description": "Rochie din satin cu croială lejeră, ideală pentru evenimente.
  Rochie din satin cu croială lejeră, ideală pentru evenimente.",
  "stock": 20,
  "reviews": 4.2,
  "noReviews": 20
}
```

Reducere activă predefinita:

```
{
  "id": 1",
  "store": "exclusive",
  "code": "SALE20"
  "productIds": ["101", "103"],
  "percentage": 20,
  "endDate": "2025-07-31",
  "status": EXPIRED
}
```

5.6.3. Fluxuri Logice Principale

Microserviciul de **Produse** deservește următoarele procese funcționale esențiale pentru platforma Vendify, fără a expune în mod direct detalii de implementare tehnică (ex: rute HTTP sau payload-uri):

- **Gestionarea produselor**: comercianții pot crea, edita și publica produse pentru magazinul propriu. Produsele conțin informații precum imagini, categorie, preț, stoc și descriere, fiind asociate unui magazin specific.
- **Filtrarea și organizarea catalogului**: produsele pot fi clasificate după categorie, afișate în funcție de noutate, popularitate sau reduceri active, în scopul de a îmbunătăți experiența de navigare a clienților.
- **Gestionarea reducerilor active** (expusa in fig. 5.9): sistemul expune un endpoint pentru obținerea reducerii active, verificând dacă aceasta este expirată. În caz afirmativ, reducerea este marcată automat ca `EXPIRED`, iar sistemul caută următoarea reducere activă din baza de date.
- **Aplicarea automată a reducerilor** (expusa in fig. 5.10): comercianții pot defini promoții ce aplică un procent de reducere asupra unui set de produse. Sistemul actualizează automat prețurile curente și păstrează valoarea originală pentru referință (`oldPrice`).
- **Livrarea conținutului media**: fiecare produs include imagini stocate în Firebase, încărcate rapid în interfața clientului prin linkuri optimizate, fără a solicita serverul principal.

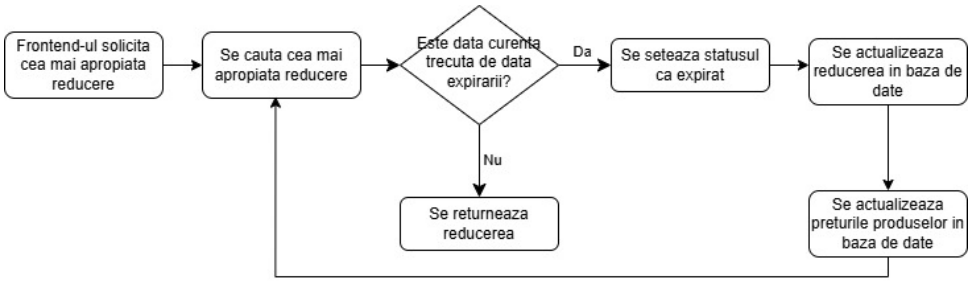

Figura 5.9: Diagrama fluxului de obținere a reducerii active

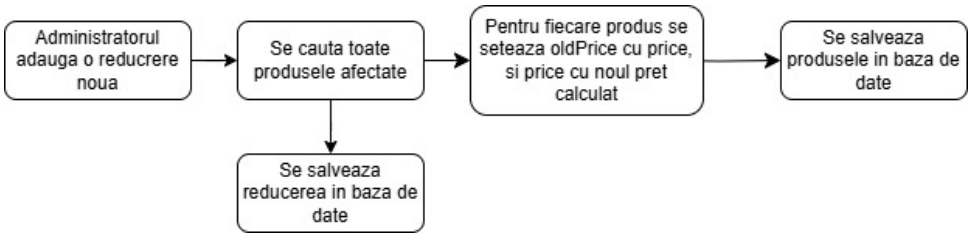

Figura 5.10: Diagrama fluxului de adaugare a unei reduceri noi

5.6.4. Avantaje Arhitecturale

- **Separarea clară a logicii comerciale** de alte componente (ex: comenzi, sesiuni);
- **Persistență relațională scalabilă** prin PostgreSQL + R2DBC, ideală pentru entități structurate;
- **Flexibilitate în afișarea produselor** prin metode dedicate (discounted, newest, related);

- **Reduceri automate asincrone**, aplicate eficient asupra grupurilor de produse;
- **Integrare media nativă** prin Firebase pentru livrarea rapidă de imagini;
- **Fluxuri reactive**, potrivite pentru interogări frecvente în contexte cu trafic ridicat.

5.6.5. Concluzie

Microserviciul de **Produse** reprezintă coloana vertebrală a experienței comerciale în platforma **Vendify**, oferind funcționalități complete pentru catalogul de produse, reduceri dinamice și interogări personalizate. Abordarea reactivă, combinată cu o structură de date clară și integrări media moderne, permite o experiență fluidă pentru utilizatori și o administrare eficientă pentru comercianți.

5.7. Implementarea Interfeței Grafice

Platforma **Vendify** beneficiază de o interfață modernă, dezvoltată în **React** cu suport pentru **TypeScript**, care asigură o experiență de utilizare rapidă, modulară și scalabilă. Arhitectura aplicației a fost deja descrisă în capitolul anterior (4), evidențiind separarea pe componente, utilizarea Redux pentru managementul stării și integrarea temelor personalizabile per magazin.

În această secțiune vom detalia aspectele concrete de implementare: structura directorului de frontend, mecanismele de rutare și randare dinamică, gestionarea stării aplicației, precum și componentele majore dezvoltate pentru interacțiunea utilizatorului și a comerciantului.

5.7.1. Structura aplicației

Interfața **Vendify** este organizată într-o manieră modulară, reflectând separarea logică a responsabilităților în cadrul aplicației. Structura directorului frontend include următoarele module principale:

- `widgets/` – conține paginile principale accesate prin rutare (`HomeWidget.tsx`, `ProductWidget.tsx`, `CheckoutWidget.tsx`, etc.), fiecare reprezentând o secțiune distinctă a platformei.
- `components/` – include componente reutilizabile precum `Navbar`, `Footer`, `ProductCard`, `RatingStars`, organizate tematic.
- `redux/` – include logica de gestionare a stării aplicației, cu subfoldere pentru fiecare domeniu funcțional (`core/`, `products/`, `orders/`, etc.), fiecare conținând `actions`, `reducers`, `selectors` și `effects` asincrone.
- `util/` – funcții auxiliare comune (`formatNumber`, `handlePriceWithDiscount`, etc.).
- `assets/` – resurse statice precum imagini, pictograme și fonturi.

Această organizare permite scalabilitate, testabilitate și reutilizare de cod, esențiale pentru o platformă multi-magazin precum **Vendify**.

Fiecare pagina din aplicație este încapsulata într-un director de tip `widget`, care combină interfața cu logica locală Redux. De exemplu, structura internă a directorului `home-widget` este organizată astfel:

- `HomeWidget.tsx` – componenta principală responsabilă cu afișarea paginii de acasa a magazinului.
- `styles.css` – stiluri locale, izolate la nivelul acestei pagini.
- `model/` – conține logica Redux specifică widgetului:
  - `types.ts` – definirea tipurilor TypeScript pentru datele și acțiunile widgetului;

– `defaultState.ts` – starea implicită locală a reducerului;
– `reducers.ts` – reducerul Redux pentru actualizarea stării;
– `selectors.ts` – funcții de extragere eficientă a datelor din state;
– `effects.ts` – efecte asincrone, pentru interacțiuni cu API-ul backend.

Această structură modulară și coerentă permite dezvoltarea și testarea independentă a fiecărui widget, facilitând întreținerea aplicației în contextul unei platforme scalabile.

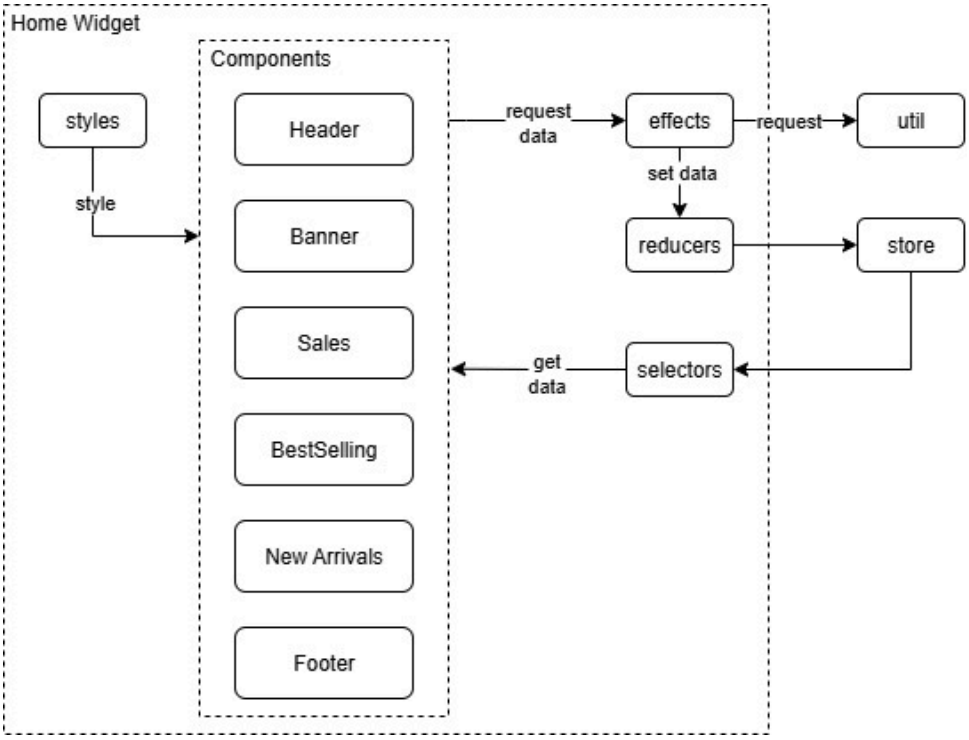

Figura 5.11: Structura logică a paginii de acasa

Această organizare este ilustrată în figura 5.11, care surprinde structura logică a paginii de acasă si componentele acesteia. Diagrama evidențiază atât componentizarea pe verticală (Header, Banner, Sales etc.), cât și interacțiunea cu infrastructura de stare globală gestionată prin Redux. Stilurile sunt aplicate local, iar componentele comunică indirect cu sursa de date printr-un lanț asincron: `effects` inițiază cereri, `reducers` actualizează starea, iar `selectors` extrag informațiile necesare. Utilitarele comune contribuie la prelucrarea datelor înainte de afișare. O astfel de arhitectură favorizează extensibilitatea și separarea preocupărilor, facilitând dezvoltarea incrementală a aplicației și întreținerea pe termen lung.

5.7.2. Gestionarea sesiuniilor

Platforma **Vendify** gestionează sesiunea utilizatorului pe partea de client, prin salvarea informațiilor esențiale într-un obiect dedicat, persistat local în `localStorage`. Această abordare permite păstrarea contextului utilizatorului chiar și după reîncărcarea paginii sau revenirea ulterioară în aplicație.

Fluxul sesiunii (prezentat si in fig. 5.12:

1. La autentificare, platforma generează un obiect de sesiune ce conține datele de identificare ale utilizatorului și magazinul asociat;

2. Obiectul este serializat și codificat în format `Base64`;
3. Datele codificate sunt salvate în `localStorage`, sub o cheie unică per magazin;
4. La fiecare accesare ulterioară a aplicației, sesiunea este decodificată și restaurată automat în starea globală;
5. În lipsa unei sesiuni valide, aplicația pornește dintr-o stare implicită.

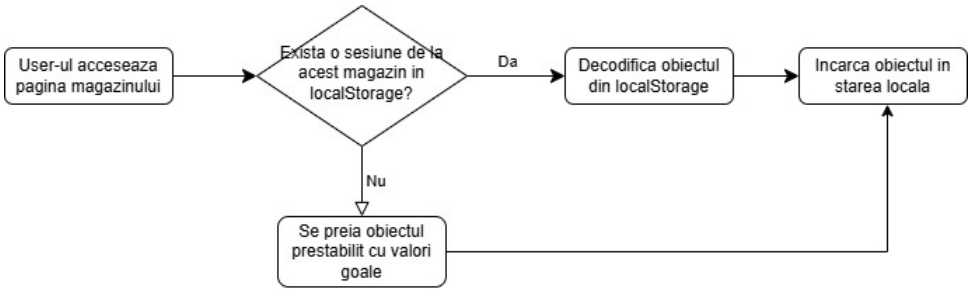

Figura 5.12: Fluxul de inițializare a sesiunii din `localStorage` în aplicația frontend

Structura obiectului de sesiune este detaliată mai jos:

Atribut	Descriere	Exemplu
`accessToken`	Token JWT pentru autorizarea cererilor	`eyJhbGciOiJIUzI1NiIs...`
`refreshToken`	Token pentru reînnoirea accesului	`eyJhbGciOiJIUzI1NiIs...`
`firstName`	Prenumele utilizatorului	`Andrei`
`lastName`	Numele de familie al utilizatorului	`Popescu`
`email`	Adresa de email asociată contului	`andrei.popescu@email.com`
`id`	Identificator unic al utilizatorului	`1`
`phoneNumber`	Număr de telefon (opțional)	`+40721234567`
`storeId`	ID-ul magazinului activ asociat sesiunii	`68bs12hj90-23df..`

Tabela 5.13: Structura datelor de sesiune stocate local, cu exemple

Acest mecanism permite gestionarea simultană a sesiunilor pentru mai multe magazine, izolate logic prin prefixarea cheilor de salvare. În timpul reîncărcării aplicației, interfața afișează componente temporare de tip `Skeleton UI` până la restaurarea completă a sesiunii, evitând astfel încărcări bruște sau ecrane goale.

Utilizarea codificării în `Base64` adaugă un strat minim de protecție și integritate a datelor salvate, fără a introduce costuri suplimentare sau dependențe externe.

5.7.3. Logica de Navigație intre pagini

Aplicația **Vendify** utilizează biblioteca `react-router-dom` pentru a gestiona navigația între pagini. Particularitatea arhitecturii constă în faptul că rutele nu sunt definite static, ci sunt generate dinamic la runtime, în funcție de magazinele existente în backend.

La inițializarea aplicației, frontend-ul trimite o cerere către microserviciul de `Magazine`, pentru a obține lista magazinelor disponibile și a configura rutele pe baza `path`-urilor fiecărui magazin (ex: `/exclusive`, `/essentials`). Astfel, aplicația creează automat rutele precum:

- `/exclusive/products/:id`
- `/essentials/cart`

- `/exclusive/checkout`

  Acest mecanism permite extinderea dinamică a platformei, fără a modifica codul sursă atunci când este adăugat un nou magazin.

Pentru rutele invalide sau în absența unui magazin valid, utilizatorul este redirecționat către o pagină generică de eroare (`/not-found`) sau către pagina de acasa a aplicației.

5.7.4. Injectarea Tematicii Vizuale

Un element distinctiv al platformei **Vendify** este capacitatea fiecărui magazin de a adopta o tematică vizuală unică, definită în backend și aplicată în mod automat în frontend.

La fiecare accesare a unei rute specifice unui magazin (`/exclusive`, `/essentials`), aplicația extrage din URL identificatorul magazinului (`storePath`) și trimite o cerere către microserviciul de `Magazine` pentru a obține obiectul `Theme` asociat.

După preluarea temei, aceasta este transmisă global, utilizând un `ThemeProvider`. Această componentă înfășoară aplicația și oferă acces la stilurile definite (culori, fonturi, margini etc.) oricărui component:

- `primaryColor`, `backgroundColor` – aplicate pe butoane, fundaluri, badge-uri.
- `titleColor`, `textSecondaryColor` – utilizate în titluri și paragrafe.
- `font` – selectat automat printr-un Google Fonts loader.
- `buttonRadius`, `cardRadius` – transmise componentelor pentru stilizare coerentă.
- `navbarStyle` – controlează transparența, blur-ul sau culoarea solidă a meniului.

5.7.5. Încărcarea Fișierelor Media în Firebase

Platforma permite utilizatorilor să încarce fișiere media precum logo-uri, bannere sau imagini de produs. Procesul implică o interacțiune directă între aplicația frontend, cloud-ul Firebase și microserviciile backend.

1. Utilizatorul selectează un fișier din interfață;
2. Aplicația frontend încarcă imaginea în Firebase, utilizând libraria Firebase;
3. După încărcare, se obține un URL public al imaginii;
4. Acest URL este trimis printr-un request HTTP către microserviciul relevant (ex: Stores sau Products);
5. Microserviciul persistă link-ul imaginii în baza sa de date.

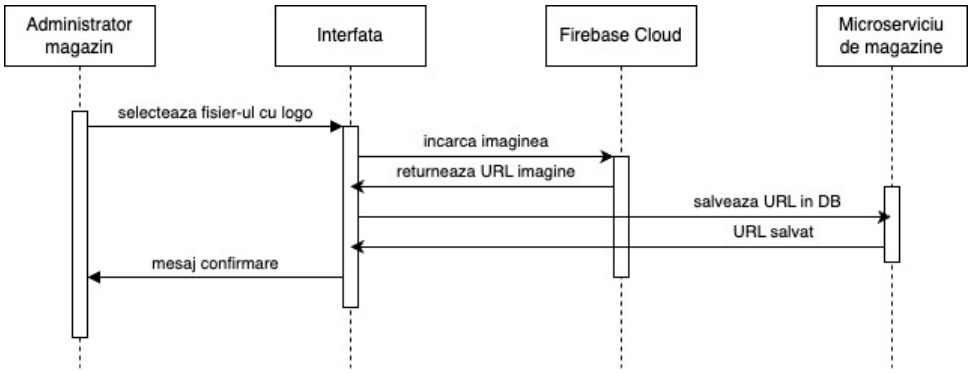

Figura 5.13: Diagramă pentru încărcarea fișierelor media în Firebase

Figura 5.13 ilustrează secvența de interacțiuni dintre utilizator, aplicația frontend, serviciul Firebase și microserviciul backend responsabil pentru persistarea datelor.

Aceasta reflectă clar separarea responsabilităților și evidențiază fluxul asincron de salvare a fișierelor media în arhitectura distribuită a platformei.

5.7.6. Concluzie

Secțiunea interfetei grafice a platformei reflectă o arhitectură modernă, orientată pe modularitate, performanță și personalizare. Prin utilizarea React și TypeScript, împreună cu Redux pentru gestionarea stării și Firebase pentru conținut media, aplicația oferă o experiență fluidă și scalabilă pentru utilizatori și comercianți. Funcționalitățile precum rutarea dinamică, persistarea sesiunii, temele personalizabile și încărcarea fișierelor în cloud evidențiază flexibilitatea și robustețea acestei componente. Structura bine organizată a codului permite extinderea facilă a platformei și menținerea clarității în dezvoltare și întreținere.

Capitolul 6

Testare și validare

6.1. Strategia de Testare

Validarea funcționalităților platformei s-a realizat prin scenarii de testare bine definite, aplicate atât asupra interfeței grafice, cât și asupra microserviciilor backend. Accentul a fost pus pe corectitudinea fluxurilor critice, persistența datelor și comunicarea între componente.

6.2. Scenarii de Testare Funcțională

Având în vedere caracterul multi-tenant al platformei, testarea funcțională se concentrează pe validarea fluxurilor principale din aplicație, dar și pe asigurarea izolării stricte între magazine. Fiecare utilizator, sesiune, produs, tematică sau comandă trebuie să fie corect asociate magazinului activ și să nu interfereze cu datele altor magazine.

Autentificare și izolare de sesiune

- **Scenariu: autentificare într-un magazin greșit** Utilizatorul încearcă să se logheze în alt magazin decât cel asociat contului său. **Așteptat:** răspuns 401 Unauthorized; nu se creează sesiunea.
- **Scenariu: sesiune stocată într-un magazin, dar URL-ul aparține altuia** La accesarea aplicației, sesiunea din `localStorage` nu se potrivește cu `storePath`-ul din URL. **Așteptat:** sesiune invalidată și revenire la stare implicită.

Navigare și separarea contextului între magazine

- **Scenariu: două magazine în taburi diferite** Utilizatorul accesează `/exclusive` și `/essentials` simultan. **Așteptat:** stări separate, sesiuni distincte și teme diferite în fiecare tab.
- **Scenariu: accesarea unui produs care nu aparține magazinului activ** Acces direct la `/exclusive/products/10` din contextul magazinului `essentials`. **Așteptat:** produsul nu este găsit, eroare 404 sau redirect.

Trimitere de comenzi și verificări de securitate

- **Scenariu: manipularea manuală a payload-ului unei comenzi** Utilizatorul modifică `storeId` în cererea către backend. **Așteptat:** microserviciul Orders returnează 403 Forbidden.
- **Scenariu: trimiterea unei comenzi valide cu datele de sesiune corecte** Comanda este trimisă dintr-o sesiune autentică și cu produse valide. **Așteptat:** răspuns 200 OK și stocarea comenzii în MongoDB.

Aceste scenarii validează nu doar funcționalitatea de bază a aplicației, ci și respectarea principiilor de securitate și izolare a contextului într-un sistem multi-magazin. Ele constituie baza testării manuale continue, alături de instrumentele descrise în secțiunea anterioară.

6.3. Instrumente Utilizate pentru Validare

Validarea funcționalităților aplicației s-a realizat printr-o serie de instrumente grafice și de depanare, fiecare contribuind la asigurarea calității aplicației în diferite etape: frontend, backend și stocare media.

6.3.1. Postman

Postman a fost utilizat pentru testarea manuală a endpoint-urilor REST expuse de microserviciile backend. Acesta permite definirea cererilor HTTP, adăugarea de antete, payload-uri JSON și inspectarea răspunsurilor.

Instrumentul a fost esențial pentru:

- Validarea autentificării și generării de token-uri JWT;
- Testarea creării de produse, comenzi și sesiuni;
- Verificarea codurilor HTTP și a mesajelor de eroare.

În figura de mai jos este prezentată o solicitare GET de comenzi și răspunsul primit:

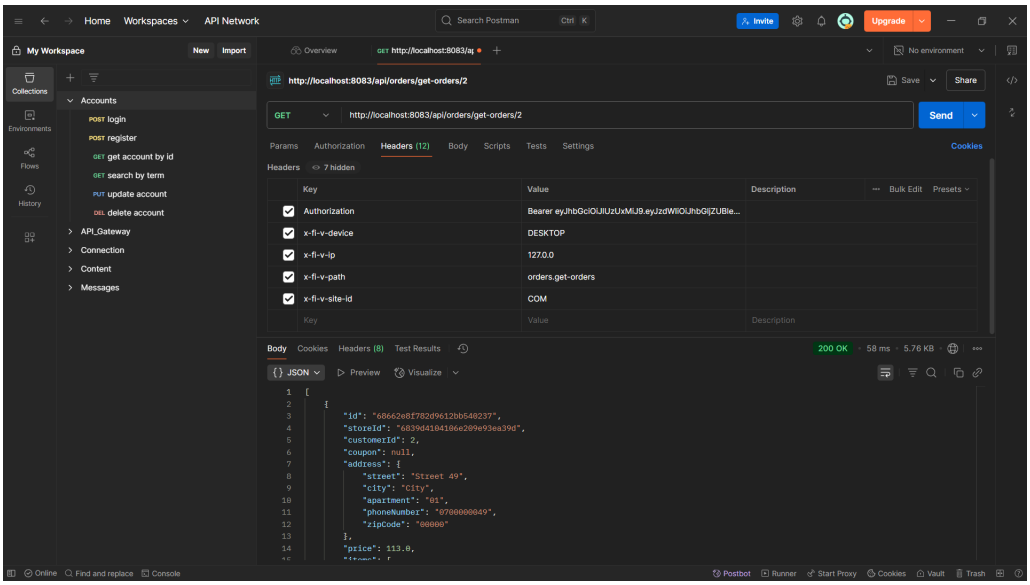

Figura 6.1: Testare a endpoint-ului de comenzi in Postman

6.3.2. Redux DevTools

Redux DevTools a fost utilizat pentru monitorizarea în timp real a modificărilor din starea globală a aplicației React. Permite urmărirea acțiunilor declanșate și evoluția datelor din Redux.

Instrumentul a fost utilizat pentru:

- Verificarea încărcării sesiunii la inițializare;
- Confirmarea aplicării temei vizuale per magazin;

- Depanare pentru fluxul de checkout și adăugare în coș.

Exemplul din figura următoare arată starea activă a sesiunii utilizatorului în Redux:

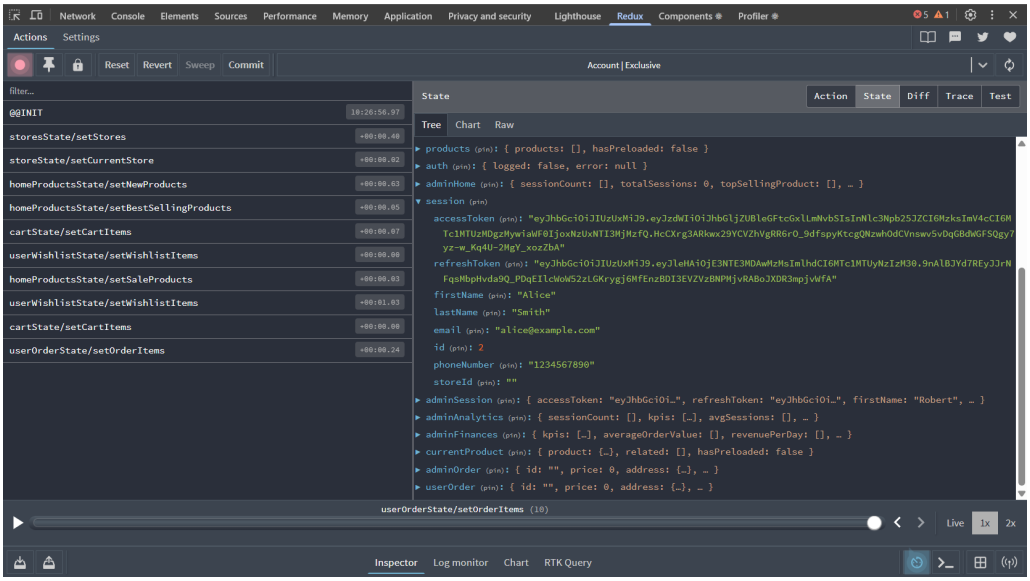

Figura 6.2: Evoluția stării de sesiune în Redux DevTools

6.3.3. Browser DevTools

Chrome DevTools a fost folosit pentru inspecția cererilor HTTP, verificarea `localStorage` și analizarea comportamentului aplicației frontend.

Utilități principale:

- Verificarea persistării corecte a obiectului de sesiune;
- Monitorizarea apelurilor către backend;
- Analiza erorilor JavaScript.

În imaginea de mai jos este evidențiat obiectul de sesiune stocat în `localStorage`:

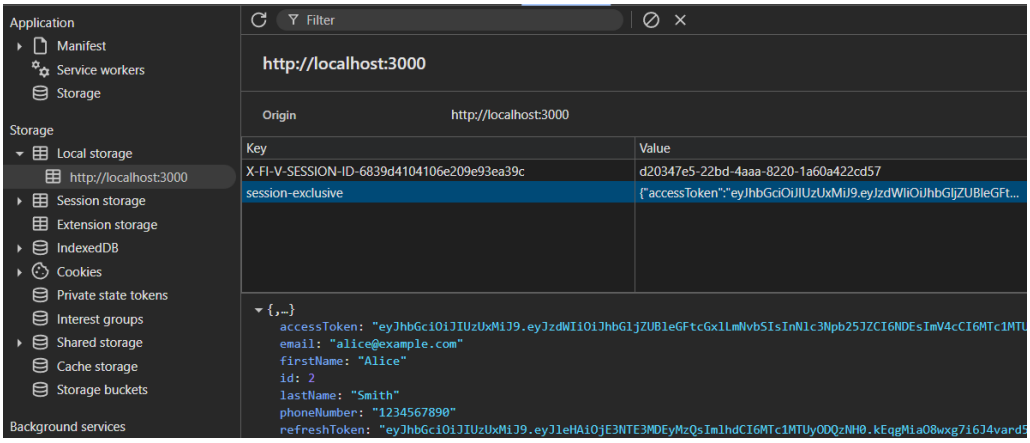

Figura 6.3: Datele de sesiune stocate în `localStorage` vizualizate cu DevTools

6.3.4. Consola Firebase

Consola Firebase oferă o interfață grafică pentru vizualizarea fișierelor media încărcate de utilizatori. Aplicația folosește Firebase Storage pentru a stoca imagini de

produse, logo-uri și bannere.

Utilitatea consolei:
- Confirmarea încărcării cu succes a fișierelor;
- Vizualizarea linkurilor publice generate automat;
- Managementul spațiului de stocare.

Figura următoare arată folderele Firebase cu fișierele încărcate:

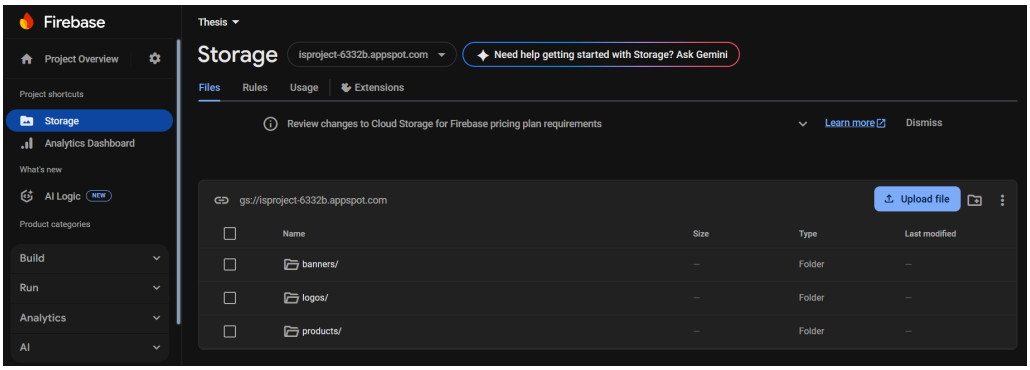

Figura 6.4: Imagini încărcate de utilizatori vizibile în Firebase Console

6.3.5. Logarea în Backend

Pentru fiecare acțiune importantă a utilizatorului (autentificare, plasare comandă, actualizare sesiune), microserviciile backend generează mesaje de log. Acestea ajută la depanare și audit.

Urmatoarele mesaje au fost logate:
- Punctul de intrare in serviciu (datele de intrare);
- Punctul de iesire din serviciu in caz de succes (datele de iesire);
- Comunicarea cu servicii externe (bazele de date, alte servicii);
- Erorile si mesajele acestora

În figura următoare este prezentat un fragment de loguri backend în timpul autentificării:

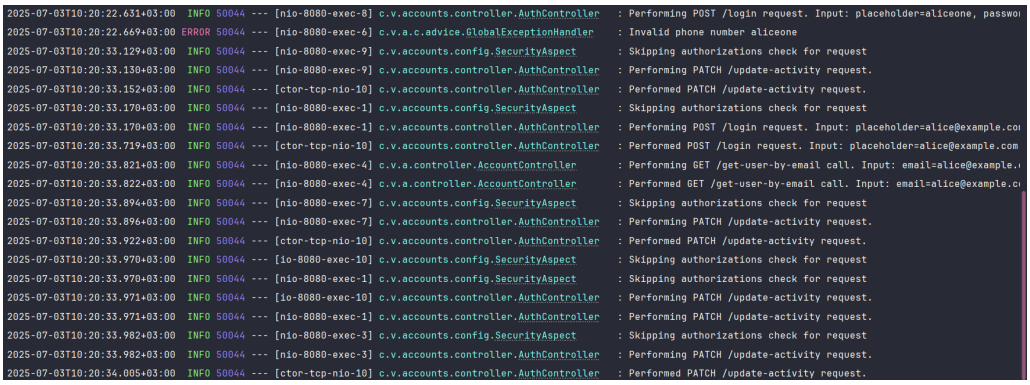

Figura 6.5: Exemplu de loguri în backend care confirmă acțiuni ale utilizatorului

6.4. Direcții de Îmbunătățire

Pentru creșterea acoperirii și automatizarea procesului de validare, sunt planificate următoarele acțiuni:

- Adăugarea de teste unitare pentru validarea logicii de business;
- Teste de integrare pentru interacțiunile dintre microservicii;
- Teste end-to-end cu instrumente precum Cypress sau Playwright;
- Validare automată a temelor vizuale și comportamentului adaptiv.

6.5. Concluzie

Testarea aplicată a demonstrat stabilitatea și coerența componentelor proiectului curent. Fluxurile esențiale au fost validate complet, iar arhitectura modulară permite adăugarea ușoară de noi scenarii de testare și instrumente de automatizare în etape viitoare.

Capitolul 7

Manual de instalare și utilizare

7.1. Cerințe de sistem

Hardware minim recomandat:

- Procesor: Quad-Core 2.5GHz+
- RAM: 8 GB
- Spațiu pe disc: minim 1 GB liber

Software necesar:

- **Node.js** v18+
- **Java** 17+
- **Maven** 3.9+
- **PostgreSQL** 14+
- **MongoDB** 6+
- **Git**

7.2. Instrucțiuni de instalare

7.2.1. Clonarea proiectului

Se recomandă folosirea Git pentru a descărca sursa aplicației:

```
git clone https://github.com/6ernst9/vendify.git
cd vendify
```

7.2.2. Configurarea bazelor de date

PostgreSQL este folosit pentru serviciile de conturi și produse. După instalarea PostgreSQL, se pot crea bazele de date necesare folosind următoarele comenzi:

```
psql -U postgres
CREATE DATABASE accounts;
CREATE DATABASE products;
```

MongoDB este folosit pentru comenzile și magazinele utilizatorilor. Nu este necesară o creare explicită a bazelor de date, însă se recomandă rularea serverului MongoDB local folosind:

```
mongod --dbpath /vendify/database
```

La prima interacțiune cu serviciile, MongoDB va crea automat bazele de date și colecțiile necesare.

7.2.3. Pornirea serviciilor

Fiecare microserviciu se poate porni dintr-un IDE precum **IntelliJ IDEA** sau **VS Code**, având ca precondiții:

- Java 17+
- Maven 3.9+
- PostgreSQL 14+
- MongoDB 6+

Fiecare serviciu are propriul director (`accounts`, `orders`, `stores` si `products`) și poate fi pornit folosind comenzile:

```
cd backend
cd accounts
mvn spring-boot:run
```

Se recomandă configurarea bazelor de date înainte de pornire (crearea schemelor necesare, dacă este cazul).

7.2.4. Pornirea interfetei grafice

Interfata este dezvoltata în React + TypeScript și se rulează local astfel:

```
cd frontend
npm install
npm run dev
```

Aplicația va fi disponibilă la adresa: `http://localhost:3000`

La prima rulare, frontend-ul va încerca automat să încarce lista magazinelor disponibile de la backend și va genera rutele necesare.

7.3. Ghid de utilizare al aplicației

7.3.1. Accesarea aplicației

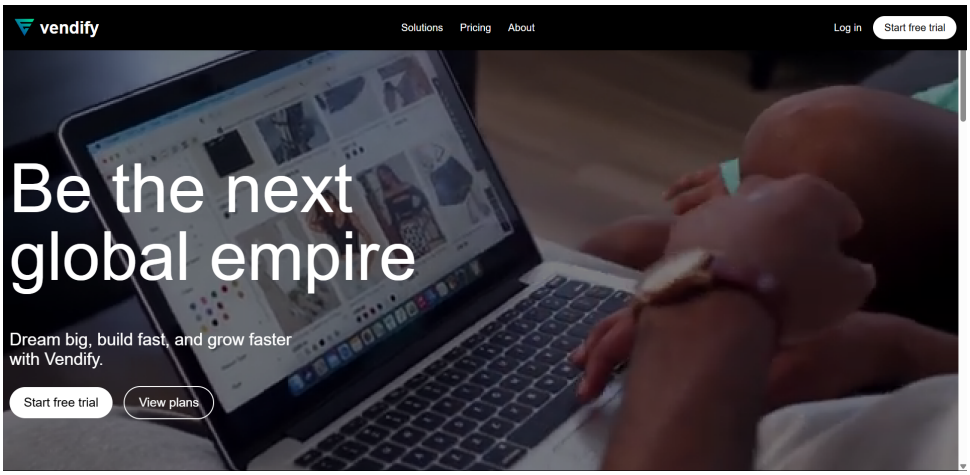

Figura 7.1: Pagina principală a magazinului

După rularea frontend-ului, aplicația poate fi accesată în browser la adresa `http://localhost:30` si va arata ca in figura 7.1.

Această pagina reprezintă aplicația principală **Vendify**, destinată utilizatorilor administratori. De aici se poate crea un cont de tip comerciant, se pot adăuga magazine noi și se pot personaliza tematicile acestora.

Pentru a accesa un magazin individual creat în platformă, utilizatorul trebuie să viziteze explicit URL-ul aferent acelui magazin. De exemplu:

- `http://localhost:3000/exclusive`
- `http://localhost:3000/essentials`

La aceste adrese, utilizatorul final (clientul) poate naviga prin produse, adăuga în coș, plasa comenzi și interacționa cu interfața magazinului.

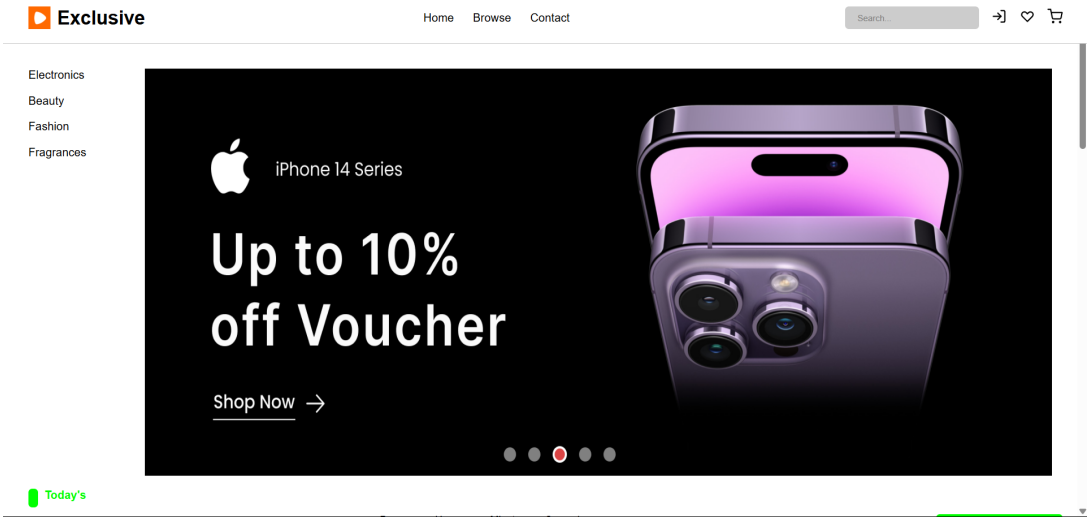

Figura 7.2: Exemplu de pagina de acasa a unui magazin accesat prin URL dedicat

Figura 7.2 ilustrează pagina principală a unui astfel de magazin, cu elemente specifice personalizate: bara de navigație, produsele afișate și stilurile tematice aplicate. Se poate observa cum identitatea vizuală a magazinului este respectată prin culori, fonturi și stiluri unice definite în backend.

7.3.2. Ghid pentru achiziționarea unui produs

Această secțiune descrie pașii necesari pentru a căuta un produs, a vizualiza detalii despre acesta, a-l adăuga în coșul de cumpărături și a finaliza comanda. Platforma a fost concepută astfel încât întregul proces să fie intuitiv și accesibil, chiar și pentru utilizatorii fără experiență tehnică.

7.3.3. Autentificarea sau crearea unui cont

1. Accesați pagina platformei: `http://localhost:3000/exclusive`.
2. În colțul din dreapta sus al ecranului, apăsați pe pictograma contului.
3. Se va deschide un formular de autentificare:
   - Dacă aveți deja un cont, introduceți adresa de e-mail și parola, apoi apăsați **„Login”**.
   - Dacă nu aveți cont, comutați pe fila **„Register”**, completați numele, adresa de e-mail, parola, apoi apăsați **„Create Account”**.

4. După autentificare, veți fi redirecționat înapoi pe pagina magazinului.

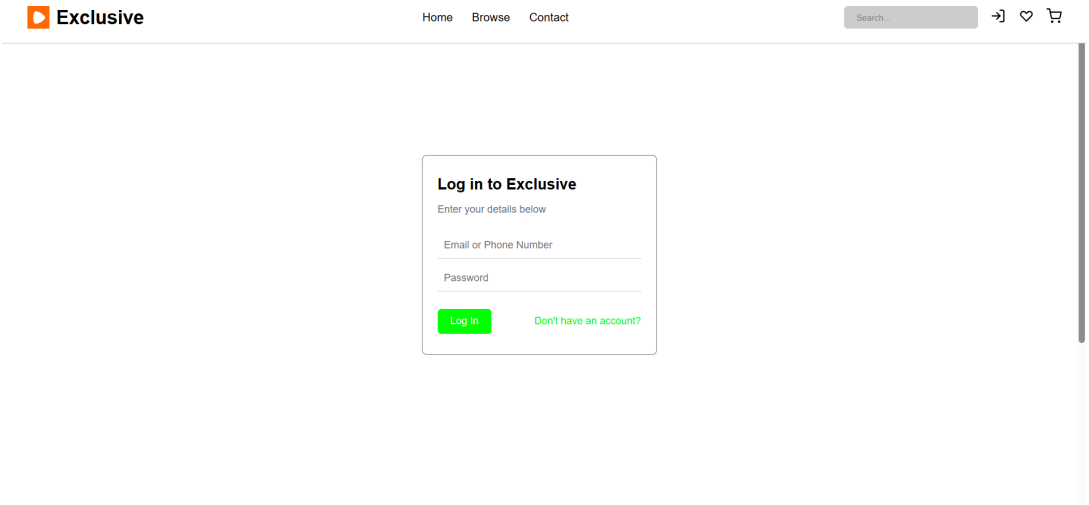

Figura 7.3: Pagina de autentificare a magazinului

Figura 7.3 prezintă interfața de autentificare, de unde utilizatorii își pot accesa contul existent sau pot crea un cont nou pentru a putea comanda produse din magazin.

7.3.4. Căutarea unui produs

1. În partea de sus a paginii, identificați bara de căutare.
2. Introduceți denumirea produsului sau un cuvânt-cheie (ex. *iPhone*).
3. Platforma va afișa automat o listă de produse care corespund căutării.

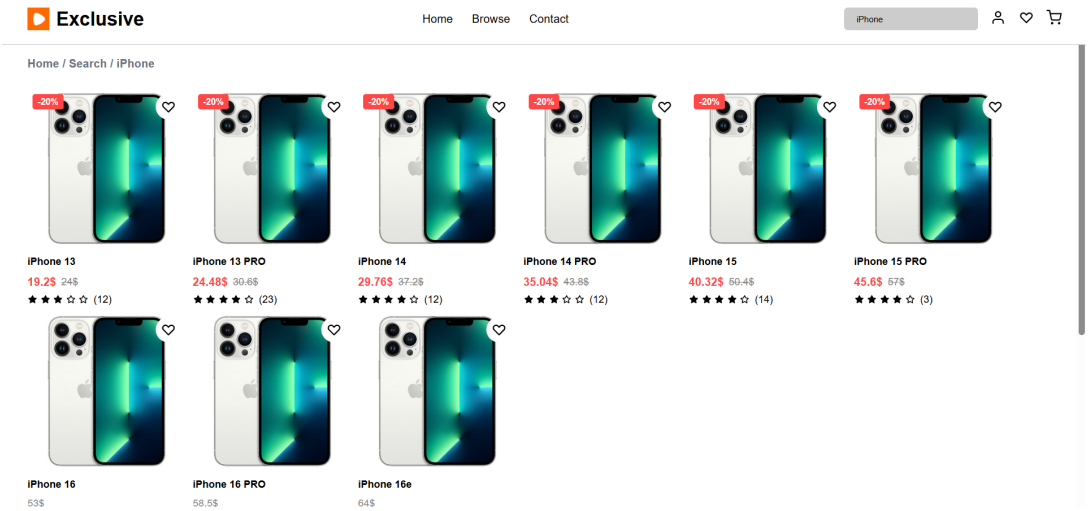

Figura 7.4: Rezultatele căutăriii ”iPhone” în bara de căutare

În figura 7.4 este ilustrat modul în care funcționează căutarea instant a produselor. Rezultatele relevante apar dinamic pe măsură ce utilizatorul tastează un cuvânt-cheie.

7.3.5. Vizualizarea detaliilor produsului

1. Din lista de rezultate, selectați produsul dorit.

2. Veți fi redirecționat către pagina de detalii a produsului.
3. Aici puteți consulta:
   - imagini detaliate ale produsului;
   - prețul actual și prețul vechi (dacă există o reducere);
   - descrierea și caracteristicile produsului;
   - ratingul oferit de alți cumpărători.

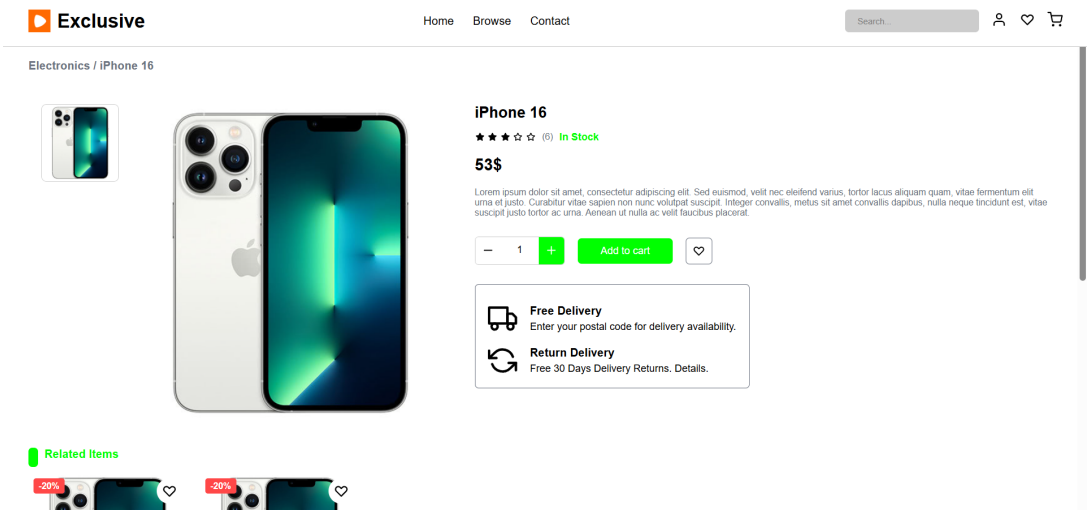

Figura 7.5: Pagina unui produs accesat a magazinului

Figura 7.5 arată pagina detaliată a unui produs, care include imagini, descriere, prețuri și evaluări, facilitând o decizie de cumpărare informată.

7.3.6. Adăugarea în coșul de cumpărături

1. Apăsați pe butonul **„Add to cart”**.
2. În colțul din dreapta sus va apărea o pictogramă a coșului actualizată cu numărul total de produse.
3. Puteți continua cumpărăturile sau accesa direct coșul printr-un click pe pictograma respectivă.

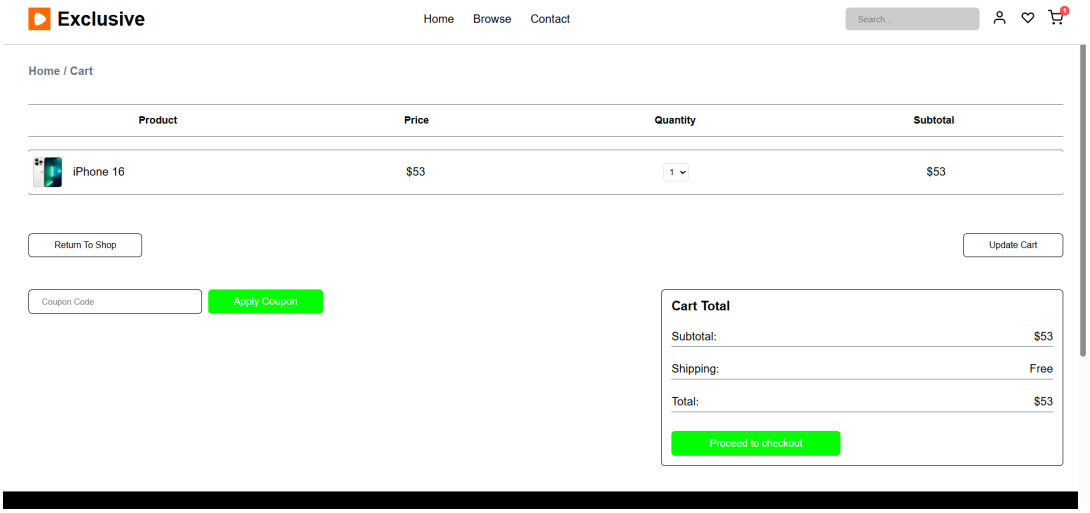

Figura 7.6: Pagina cu coșul de cumpărături a magazinului

În figura 7.6 se poate observa coșul de cumpărături, unde utilizatorul poate verifica lista articolelor adăugate, ajusta cantitățile sau merge mai departe către finalizarea comenzii.

7.3.7. Finalizarea comenzii

1. În pagina de coș, verificați produsele adăugate:
   - numele produselor;
   - cantitatea și prețul fiecărui articol;
   - totalul general.
2. Apăsați pe **„Proceed to checkout”**.
3. Completați formularul cu datele de livrare:
   - nume complet;
   - adresă de livrare;
   - număr de telefon și adresă de e-mail.
4. Apăsați pe butonul **„Place order”**.
5. Veți fi redirecționat la pagina de confirmare a comenzii.

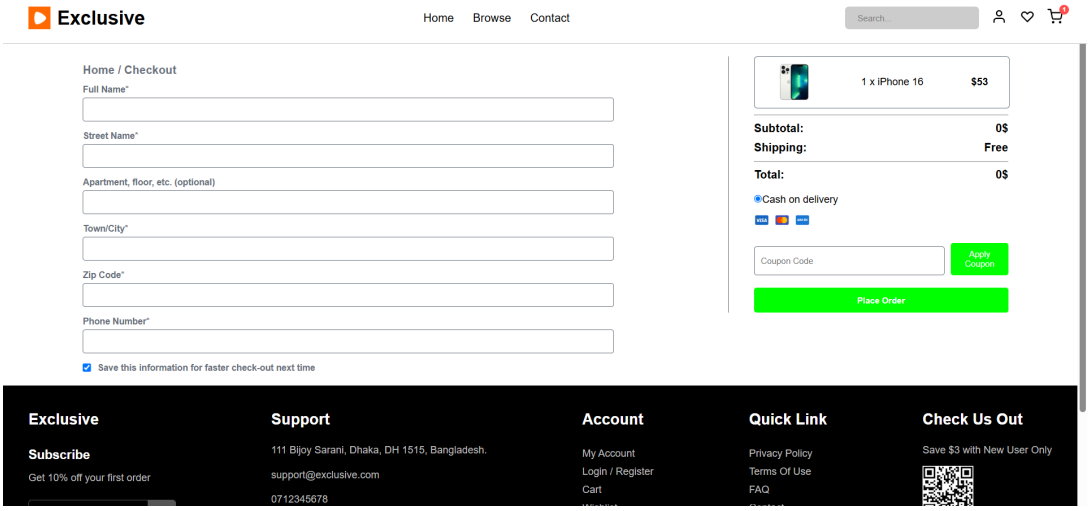

Figura 7.7: Ultimul pas înainte de plasarea unei comenzi

Figura 7.7 evidențiază ultimul pas din procesul de comandă, în care utilizatorul completează datele de livrare și plasează comanda cu un singur click.

Capitolul 8

Concluzii

8.1. Rezumat al contribuțiilor

Lucrarea de față a avut ca scop proiectarea și implementarea unei platforme moderne de eCommerce, orientată către crearea de magazine online personalizabile, într-un sistem multi-tenant. Problema identificată a fost lipsa unei soluții tehnice flexibile și scalabile care să permită utilizatorilor să își gestioneze propriul magazin, fără a depinde de platforme rigide sau costisitoare precum Shopify.

Soluția propusă oferă o platformă flexibilă și ușor de extins, care permite utilizatorilor să creeze și să administreze propriile magazine online într-un mod intuitiv și personalizabil. Fiecare magazin beneficiază de o identitate vizuală distinctă, o pagină dedicată și o experiență de navigare adaptată. Platforma susține o separare clară a rolurilor și a datelor, astfel încât fiecare utilizator și magazin operează într-un mediu izolat și securizat.

Contribuțiile principale includ:

- proiectarea unei arhitecturi modulare scalabile, cu patru microservicii independente;
- implementarea unui sistem de sesiuni multi-tenant, cu izolare logică între utilizatori și magazine;
- dezvoltarea unei interfețe moderne, responsive și ușor de extins, cu sistem tematic complet configurabil;
- integrarea cu Firebase pentru stocarea fișierelor media;
- furnizarea unui mecanism de validare funcțională prin scenarii, sesiuni și componente observabile (Redux DevTools, Postman).

8.2. Analiza critică a rezultatelor

Rezultatele obținute respectă obiectivele propuse și îndeplinește cerințele funcționale: platforma funcționează stabil într-un mediu local de dezvoltare, permițând crearea, configurarea și accesarea unor magazine distincte, fiecare cu propriile produse, utilizatori și tematică.

Arhitectura cu microservicii a oferit avantaje clare în ceea ce privește modularitatea, testabilitatea și posibilitatea de extindere ulterioară. Separarea logică a componentelor a făcut ca integrarea funcționalităților suplimentare (precum reduceri automate, analize statistice sau recomandări) să fie posibilă fără impact major asupra componentelor existente.

Cu toate acestea, lucrarea prezintă și anumite limitări:

- nu au fost incluse teste unitare sau de integrare automatizate;
- nu s-a realizat o strategie completă de deployment (ex: utilizarea de containere

Docker sau orchestratori precum Kubernetes);
- aplicația este dependentă de rularea tuturor serviciilor local pentru o experiență completă.

8.3. Dezvoltări și direcții viitoare

Platforma Vendify poate fi extinsă în mai multe direcții pentru a atinge un nivel de maturitate superior:

- **Containerizare în cloud**: utilizarea containerizării (ex. Docker) împreună cu servicii de cloud precum AWS, Google Cloud sau Azure ar permite ca platforma să fie accesibilă publicului larg, oferind scalabilitate automată, disponibilitate ridicată și costuri optimizate pentru resursele utilizate.
- **Autentificare unificată și Single Sign-On**: implementarea unui mecanism de autentificare centralizat ar permite utilizatorilor să acceseze mai multe magazine cu aceleași credențiale, fără a fi necesară autentificarea separată. Astfel, experiența de navigare ar fi mai fluidă și coerentă în cadrul ecosistemului Vendify.
- **Sistem de plăți integrat**: integrarea unor procesatori de plăți precum Stripe sau PayPal ar transforma platforma într-o soluție completă de eCommerce, permițând clienților să finalizeze comenzile direct din interfață, în siguranță și fără pași suplimentari.
- **Suport pentru marketplace-uri**: extinderea platformei astfel încât mai mulți comercianți să poată activa în același magazin ar permite scenarii de tip marketplace, precum Amazon sau eMAG. Aceasta ar implica și adăugarea de funcționalități precum comisioane per vânzător, gestionarea stocurilor per comerciant și dashboard-uri individuale.
- **Recomandări și analitice avansate**: prin analizarea comportamentului de cumpărare și a datelor colectate în sesiuni (produse vizualizate, timp petrecut, rate de conversie etc.), se pot genera recomandări personalizate pentru utilizatori. Integrarea unei componente de Inteligență Artificială, precum un sistem de recomandare bazat pe învățare automată, ar permite oferirea de sugestii inteligente și adaptative, îmbunătățind semnificativ experiența de cumpărare și ar contribui la creșterea conversiilor.

În concluzie, proiectul curent demonstrează viabilitatea unei platforme eCommerce moderne, scalabile și personalizabile, construită cu tehnologii actuale și gândită pentru extindere. Prin abordarea multi-tenant și arhitectura reactivă, aplicația oferă o bază solidă pentru dezvoltări ulterioare, putând fi adaptată cu ușurință cerințelor comerciale reale.

Bibliografie

[1] A. Benlian, T. Hess, and P. Buxmann, "Saas adoption in business: The role of transaction cost theory and trust," *European Journal of Information Systems*, vol. 20, no. 3, pp. 243–259, 2011.

[2] Shopify Inc., "Shopify documentation," https://shopify.dev/docs, 2024, accessed: 2025-06-25.

[3] BigCommerce Pty. Ltd., "Bigcommerce developer documentation," https://developer.bigcommerce.com/, 2024, accessed: 2025-06-25.

[4] Automattic Inc., "Woocommerce documentation," https://woocommerce.com/documentation/, 2024, accessed: 2025-06-25.

[5] Adobe Inc., "Magento open source user guide," https://experienceleague.adobe.com/docs/commerce.html, 2024, accessed: 2025-06-25.

[6] Wix.com Ltd., "Wix ecommerce help center," https://support.wix.com/en/commerce, 2024, accessed: 2025-06-25.

[7] Squarespace Inc., "Squarespace help: Commerce," https://support.squarespace.com/hc/en-us/categories/200353688-Commerce, 2024, accessed: 2025-06-25.

[8] C. Bezemer and A. Zaidman, "Multi-tenant saas applications: maintenance dream or nightmare?" *Proceedings of the Joint ERCIM Workshop on Software Evolution (EVOL) and International Workshop on Principles of Software Evolution (IWPSE)*, pp. 88–92, 2010.

[9] J. J. Garrett, "Elements of user experience: User-centered design for the web and beyond," 2010.

[10] S. Newman, *Building Microservices: Designing Fine-Grained Systems.* O'Reilly Media, 2015.

[11] J. Webber, *Reactive Spring.* VMware, 2020, https://docs.spring.io/spring-framework/docs/current/reference/html/web-reactive.html.

[12] M. G. Institute, *Analytics Comes of Age*, 2018, https://www.mckinsey.com/business-functions/mckinsey-analytics/our-insights/analytics-comes-of-age.

[13] H. Chen, R. H. Chiang, and V. C. Storey, "Business intelligence and analytics: From big data to big impact," *MIS Quarterly*, vol. 36, no. 4, pp. 1165–1188, 2012.

[14] Google LLC, "Firebase cloud storage documentation," https://firebase.google.com/docs/storage, 2024, accessed: 2025-07-05.

[15] Meta Platforms, Inc., "React – a javascript library for building user interfaces," https://react.dev, 2024, accessed: 2025-07-05.

[16] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. V. Lopes, J.-M. Loingtier, and J. Irwin, "Aspect-oriented programming," in *ECOOP'97 — European Conference on Object-Oriented Programming*, ser. Lecture Notes in Computer Science, M. Aksit and S. Matsuoka, Eds., vol. 1241. Springer, 1997, pp. 220–242. [Online]. Available: https://doi.org/10.1007/BFb0053381

Capitolul A

Secțiuni relevante din cod

```
 // Filtru de securitate folosind Spring Aspects.
@Before("@annotation(mapping)"
public void securityFilterChain(WMTSecurityMapping mapping){
    var request = ((ServletRequestAttributes)RequestContextHolder
            .currentRequestAttributes()).getRequest();

    var requestMapping = request.getHeader("X-FI-V-PATH");
    var siteId = request.getHeader("X-FI-V-SITE-ID");
    var device = request.getHeader("X-FI-V-DEVICE");
    var ipAddress = request.getHeader("X-FI-V-IP");

    if(requestMapping == null) throw new SecurityException("Path not found",
        "Request should contain path mapping");

    if(siteId == null) throw new SecurityException("Site ID not found",
        "Request should contain site id");

    if(device == null) throw new SecurityException("Device not found",
        "Request should contain device");

    if(ipAddress == null) throw new SecurityException("IP Address not found",
        "Request should contain ip address");

    if(!requestMapping.contains(mapping.path())) throw new
        SecurityException("Invalid path", "Request should contain endpoint path");

    var authorization = request.getHeader("Authorization");
    if (authorization == null) {
        throw new SecurityException("Authorization not found",
        "Authorization header is required");
    }
    if(!authorization.startsWith("Bearer")) {
        throw new SecurityException("Invalid authorization",
        "Authorization should start with 'Bearer'");
    }
    var token = authorization.substring(7);

    if(!jwtGenerator.validateToken(token)) {
```

```
        throw new SecurityException("Invalid token", "Token not matching claims");
    }
}



// Actualizarea activitatii utilizatorului.
public Mono<Void> updateActivity(String sessionId, String store, String path,
        String action) {
    return sessionRepository.findLatestByUser(sessionId).flatMap(session -
> {
        var log = UserActivityMapper.toAdminLog(session.getUserId(), path, action);
        StoreLogger.log(session.getStoreId(), log);

        if(session.getLastActivity().isAfter(LocalDateTime.now().minusMinutes(10))) {
            session.setLastActivity(LocalDateTime.now());
            var pages = session.getPagesVisited();
            pages.add(path);
            var actions = session.getActions();
            actions.add(action);

            session.setPagesVisited(pages);
            session.setActions(actions);
            return sessionRepository.save(session);
        } else {
            session.setEndTime(session.getLastActivity());
            return sessionRepository.save(session)
                    .then(sessionRepository.save(
                        new Session(
                            store,
                            sessionId,
                            List.of(path),
                            List.of(action)
                            )
                        )
                    );
        }
    }).switchIfEmpty(Mono.defer(() -> {
        var session = new Session(store, sessionId, List.of(path), List.of(action));

        var log = UserActivityMapper.toAdminLog(session.getUserId(), path, action);
        StoreLogger.log(session.getStoreId(), log);

        return sessionRepository.save(session);
    })).then(Mono.empty());
}
```

```
// Crearea unei comenzi din cos-ul de cumparaturi.
public Mono<Void> createOrder(OrderDTO orderDTO) {
    return cartRepository.findByCustomerIdAndStoreId(orderDTO.getCustomerId(),
        orderDTO.getStoreId())
            .collectList()
            .flatMap(cartItems -> {
                if (cartItems.isEmpty()) return Mono.error(
                    new RuntimeException("Cart is empty"));

                var items = cartItems.stream().map(c ->
                        new OrderItem(c.getProductId(), c.getQuantity()))
                    .toList();

                var order = new Order(orderDTO.getStoreId(),
                    orderDTO.getCustomerId(),
                    orderDTO.getCoupon(),
                    orderDTO.getAddress(),
                    orderDTO.getPrice(),
                    items,
                    OrderStatus.PLACED,
                    LocalDateTime.now());
                return orderRepository.save(order)
                        .then(cartRepository.deleteAll(cartItems)
                                .then(updateStock(items))
                                .then(applyCoupon(
                                    orderDTO.getCoupon(), orderDTO.getStoreId()))
                        );
            });
    }


// Actualizarea cantitatii in cosul de cumparaturi
public Mono<CartItem> updateQuantity(long customerId, String storeId,
    long productId, int quantity) {
        return cartRepository.findByCustomerIdAndStoreIdAndProductId(customerId,
            storeId, productId)
                .flatMap(existing -> {
                    if(quantity == 0) {
                        return cartRepository.deleteById(existing.getId())
                            .then(Mono.just(existing));
                    } else {
                        existing.setQuantity(quantity);
                        return cartRepository.save(existing);
                    }
                });
    }
```

```
// Promovarea listei de dorinte in cosul de cumparaturi
public Flux<CartItem> promoteWishlist(long id) {
        return wishlistRepository.findByCustomerId(id)
                .flatMap(wishlistItem -> {
                    CartItem cartItem = new CartItem(
                            wishlistItem.getStoreId(),
                            wishlistItem.getCustomerId(),
                            wishlistItem.getProductId(),
                            1
                    );
                    return cartRepository.save(cartItem)
                            .flatMap(saved -> wishlistRepository
                                .deleteById(wishlistItem.getId()).thenReturn(saved));
                });
    }


// Crearea unei reduceri
public Mono<Void> createSale(SaleDTO saleDTO) {
        Mono<List<Long>> productIdsMono;

        if (!saleDTO.getProductIds().isEmpty()) {
            productIdsMono = Mono.just(saleDTO.getProductIds());
        } else {
            productIdsMono = productsRepository
                    .findProductsByStoreAndCategory(
                        saleDTO.getStore(),
                        saleDTO.getCategory()
                    )
                    .map(Product::getId)
                    .collectList();
        }

        return productIdsMono.flatMap(productIds -> {
            var sale = new Sale(
                    saleDTO.getName(),
                    saleDTO.getCode(),
                    saleDTO.getStore(),
                    SaleStatus.ACTIVE.name(),
                    saleDTO.getPercentage(),
                    productIds,
                    saleDTO.getEndDate()
            );

            if(saleDTO.getCode() == null) {
                return saleRepository.save(sale)
                        .then(productsRepository
                            .applyDiscountToProducts(
```

```
                                saleDTO.getPercentage() / 100, productIds)
                            );
            }

            return saleRepository.save(sale).then();
        });
    }



// Preluarea dinamica a magazinului din URL
useEffect(() => {
        const pathname = window.location.pathname;
        const segments = pathname.split('/').filter(Boolean);
        const slug = segments[0];

        if (slug === currentPath) {
            return;
        }

        if(allowedPaths.includes(slug)) {
            return;
        }

        if(loaded){
            const foundStore = stores.find((store) => store.path === slug);
            if (foundStore) {
                dispatch(setCurrentStore(foundStore));
            } else {
                console.error("Store not found:", slug);
                navigate('/home')
            }
        } else {
            return;
        }
    }, [stores]);



// Atasarea cookie-ului de sesiune in localStorage
function getOrCreateSessionId(storeId: string): string {
    const key = COOKIE_KEY + storeId;
    let sessionId = localStorage.getItem(key);
    if (!sessionId) {
        sessionId = crypto.randomUUID();
        localStorage.setItem(key, sessionId);
    }
    return sessionId;
}
```

```
// Incarcarea unei imagini in cloud-ul Firebase
export const uploadImageToFirebase = async (file: File, path: string)
        :Promise<string> => {
    const uuid = crypto.randomUUID();
    const imageRef = storageRef(storage, `${path}/${uuid}`);
    const snapshot = await uploadBytes(imageRef, file);
    return await getDownloadURL(snapshot.ref);
};



// Preluarea sesiunii din storage pentru persistenta
const getSessionKey = () => {
    const path = window.location.pathname.split('/');
    const storeId = path[1];
    return `session-${storeId}`;
};

const loadSession = () => {
    try {
        const key = getSessionKey();
        const serializedSession = localStorage.getItem(key);
        if (serializedSession === null) return defaultSession;
        const session = atob(serializedSession);
        return JSON.parse(session);
    } catch (e) {
        return defaultSession;
    }
};
```