UNIVERSITATEA DE MEDICINĂ, FARMACIE,
ȘTIINȚE ȘI TEHNOLOGIE "GEORGE EMIL
PALADE" DIN TÂRGU MURES
FACULTATEA DE INGINERIE ȘI
TEHNOLOGIA INFORMAȚIEI

SPECIALIZAREA: AUTOMATICĂ ȘI INFORMATICĂ APLICATĂ

PROIECT DE DIPLOMĂ

Gestiunea centralizată a datelor pentru comercializarea echipamentelor pentru sporturi de iarnă

Îndrumător ştiinţific: Absolvent:

Şef lucr. Ing. Marius MUJI Ștefan Sergiu- Cătălin

Sesiunea Iulie 2023

UNIVERSITATEA DE MEDICINĂ, FARMACIE, ȘTIINȚE ȘI TEHNOLOGIE "GEORGE EMIL PALADE" DIN TÂRGU-MUREȘ
FACULTATEA DE INGINERIE ȘI TEHNOLOGIA INFORMAȚIEI
Specializarea: Automatică și Informatică Aplicată Viza facultăţii

TEMĂ PROIECT/LUCRARE DE DIPLOMĂ/LICENȚĂ/DISERTAȚIE

Coordonator științific: Şef lucr. Ing. Marius MUJI

Candidat (a): Ștefan Sergiu- Cătălin

Anul absolvirii: ..2023.....

Tema proiectului/lucrării de diplomă/licență/disertație:

Dezvoltarea unui magazine online pentru comercializarea de articole sportive pentru sporturi de iarnă

Problemele principale care vor fi tratate:

Proiectarea bazei de date, Analiza cerințelor unei aplicații software ,

Implementarea unei aplicații web și a bazei de date

Bibliografie recomandată:

C. J. Date, An Introduction to Database Systems (8th edition), Addison-Wesley, 2003.

William J. Lewis, Data Warehousing and E-Commerce, Prentice Hall PTR, 2001.

Termene obligatorii de consultaţii: Săptămânal

Locul practicii: laboratoarele Facultății de Inginerie

Primit la data de: aprilie 2022

Termen de predare: 07.07.2023

Semnătura director de partament

Semnătura conducătorului

Semnătura candidatului

Cuprins

1. Introducere

1.1. Scopul lucrării

Se dorește dezvoltarea unui website de e-commerce conceput pentru vânzarea unor articole sportive pentru sporturi de iarnă.

Printr-o interfață simplă si intuitivă, clienții vor putea vizualiza și filtra produsele comercializate, după mai multe criterii. Ulterior, le pot salva într-o listă de produse favorite sau într-un cos de cumpărături al cărui conținut se utilizează pentru generarea unei comenzi, deci implicit este prezent și un sistem de autentificare.

Utilizatorii autentificați și autorizați cu un cont de administrator, au privilegiul de a modifica datele despre produse și de a actualiza stările comenzilor plasate de către clienți. Interfața trebuie să fie dinamică, iar modificările făcute asupra produselor să nu necesite cunoștințe de programare.

Prin intermediul unei astfel de aplicații software, se pot crește vânzările firmei prin îmbunătățirea relațiilor cu clienții și creșterea productivității prin automatizarea unor procese de logistică din cadrul business-ului.

1.2. Situația actuală și motivația temei

Acum câteva zeci de ani a început fenomenul cel mai reprezentativ pentru societatea actuală, și anume dezvoltarea accelerată a tehnologiei. Aceasta a devenit un element foarte important în viața fiecărui om. Astfel în ziua de astăzi, dispozitivele electronice ne fac viața mai ușoară și mai comodă deoarece avem posibilitatea de a accesa și trimite tot felul informații la distanțe mari cu ajutorul a câtorva click-uri sau atingeri de ecran.

Majoritatea companiilor s-au adaptat treptat la stilul de viață dominat de tehnologie, devenind în cele din urmă un standard ca acestea sa aibă o prezență online intr-un fel sau altul. Acest standard presupune existenta unor website-uri sau platforme online destinate clienților, prin care aceștia pot lua contact cu firma și se pot informa despre serviciile oferite, sau prin existența unor platforme destinate angajatilor care le permite acestora să gestioneze și să automatizeze anumite elemente cheie din cardul business-ului.

În general companiile sunt dispuse să investească din ce în ce mult în dezvoltarea unor astfel de platforme online pe măsură, în timp ce, cele care aleg să nu investească în domeniul online pierd o mulțime de beneficii fiind ulterior depășite de către ceilalți competitori.

Un exemplu extrem de întâlnit este prezența unei aplicații de e-commerce pentru magazine. Cu ajutorul unei astfel de aplicații, se pot gestiona automat stocuri și vânzări, eliminându-se erorile umane dacă aplicația este proiectată corect și respectă cerințele impuse. De altfel este important ca navigarea aplicației sa fie intuitivă și ușoară. Un site cu un aspect neplăcut sau neîngrijit le poate inspira utilizatorilor nesiguranță și neîncredere pierzându-se astfel posibili clienți în ciuda serviciilor oferite mai ales in cadrul unor firme mai puțin cunoscute.

În concluzie, o platformă online este o nevoie pentru aproape orice business, deoarece se economisesc bani și mult timp prin automatizarea unor procese, se elimină erorile umane din managementul contabilității și logisticii și le oferă posibililor clienți oportunitatea de a apela la serviciile firmei rapid și ușor.

1.3. Structura lucrării

Lucrarea este împărțită in 4 capitole principale .

Capitolul 1 – se prezinta motivația temei, scopul lucrării și câteva concepte generale

Capitolul 2 –se vor analiza cerințele sistemului informatic folosind diagrame

Capitolul 3 – se va proiecta aplicația și vor fi prezentate câteva aspecte importante din diferite perspective folosind diagrame

Capitolul 4 – se vor prezenta detalii tehnice legate de implementarea propriu-zisă a aplicației si despre baze de date.

Capitolul 5 – se va prezenta concluzia

2. Analiza sistemului

2.1.Cerințele sistemului

Aplicația concepută va trebui să îndeplinească câteva cerințe minime pentru a-și atinge scopul. Programatorul trebuie sa urmeze aceste cerințe pentru ajunge la produsul final dorit.
Aplicația va trebui să îndeplinească funcțiile de bază specifice oricărei platforme de e-Commerce și anume:

- Autentificare și autorizare pentru fiecare utilizator în parte (atât pentru clienți cât și pentru angajați).
- Efectuarea de operații CRUD (Create Retrieve Update Delete) asupra datelor de către administrator prin intermediul unor interfețe specializate.
- Stocarea articolelor sportive în baza de date trebuie să fie cât mai detaliată. Echipamentele destinate sporturilor de iarnă sunt foarte diversificate iar stocarea acestora în baza de date trebuie să reflecte realitatea cât mai bine.
- Având în vedere modelul de date, trebuie implementată o funcție de filtrare după mai multe criterii
- Interfața clienților trebuie să fie simplă și intuitivă. Trebuie să se poată ajunge in orice punct al aplicației în cel mult 3 sau 4 click-uri.
- Clienții trebuie să aibă posibilitatea de a salva produse într-o listă de favorite.
- Clienții să poată să inițializeze comenzi pe baza unui coș de cumpărături.
- Clienții pot vizualiza starea curentă a comenzilor plasate și istoricul comenzilor

Câteva dintre aceste cerințe pot fi descrise cu ajutorul unor diagrame.
In continuare se va prezenta diagrama cazurilor de utilizare pentru aplicație.

2.2.Diagrama cazurilor de utilizare(Use Case)

Diagrama cazurilor de utilizare (use case diagram) este o diagramă UML, utilizată în faza de proiectare și analiză a unei aplicații software pentru a descrie funcționalitățile sistemuluiPrin intermediul unei astfel de diagrame se pot clarifica cerințele și nevoile impuse atunci când acestea trebuiesc transmise altor persoane, într-un mod ușor de înțeles, independent de tehnologia utilizată în implementarea aplicației.

Programatorii, pot urmării aceste diagrame pe parcursul întregului proces de dezvoltare pentru a se asigura că sistemul este construit conform specificațiilor și cerințelor inițiale. O diagramă use case are câteva componente specifice: actori, use case-uri și relațiile dintre ele.

Actorii

Aceștia sunt elementele externe care interacționează cu sistemul descris. Actorii pot fii niste simplii utilizatori, sisteme software sau dispozitive hardware și se reprezinta grafic printr-o persoană. Aceștia pot lua diverse roluri în funcție de care vor avea anumite capabilități în cadrul sistemului.

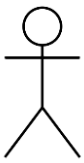

Fig 2.1. Reprezentarea grafică a unui actor

Use case-uri

Acestea reprezintă acțiunile pe care le poate executa un actor în sistem. Acestea arată cele mai importante acțiuni pe care le poate efectua actorul cu care sunt asociate. Nu se specifică alte detalii precum modul de execuție sau de implementare al acestora.

Din punct de vedere grafic, un use case se reprezintă printr-un oval in interiorul căruia se afla numele său. Acesta trebuie sa fie simplu și ușor de înțeles pentru oameni și să reprezinte o acțiune.

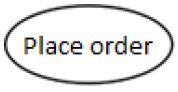

Fig 2.2. Reprezentarea grafică a unui use-case care descrie acțiunea de plasare a unei comenzi

Tipuri de relatii

Între use case-uri și actori se stabilesc relații care sunt importante pentru descrierea modului in funcționează sistemul și cerințele de implementare ale acestuia.

Există câteva tipuri de relații: asociere, dependență și generalizare.

Relatia de asociere se poate defini între use case-uri sau între actori și use case-uri. Acesta sugerează comunicarea intre componentele pe care le unește și se poate reprezenta grafic printr-o linie.

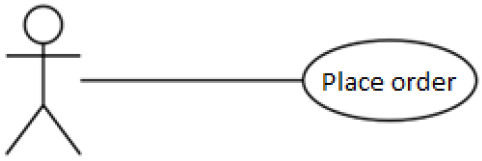

Fig 2.3. Reprezentarea grafică a unei relații de asociere între un use case și un actor

Relații dependență - acestea se pot forma între doua cazuri de utilizare și cuprinde două tipuri: relație de includere sau de extindere.

Relatia de includere este folosită pentru a descrie o situație când un useCase este inclus în altul. În exemplul urmator, Use Case-ul A include use Case-ul B și se reprezintă grafic printr-o sageata cu linie punctata pe care scrie "include", iar sageata este indreptată catre use Case-ul inclus.

Cu alte cuvinte, atunci cand se execută cazul A și cazul B se execută obligatoriu, iar cazul B este incomplet fără A. În figura 2.4 useCase-ul AddToShoppingCart îl include pe useCase-ul Browse Products.

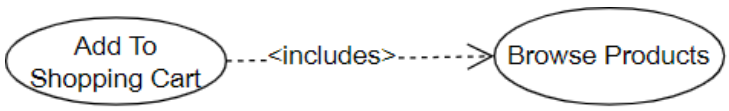

Fig 2.4. Dependență de tip **include**

Relaxia de extindere se folosește în cazul în care un use case îi adaugă ceva în plus altui caz de utilizare. În exemplul următor, use case-ul B extinde use case-ul de bază A. Execuția lui B este opțională și nu are sens fără A. În exemplul următor useCase-ul Update Order State extinde useCase-ul de bază View Orders. Update Order State este opțional.

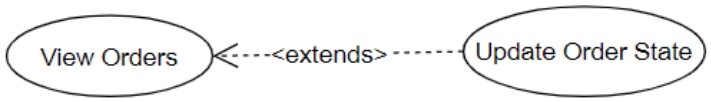

Fig 2.5. Dependență de tip **extend**

Relatia de generalizare se aseamănă cu relația de moștenire din programarea orientată obiect. Se arată că un actor primește rolul și toate cazurile de utilizare ale părintelui și pe de-asupra are useCase-urile proprii. De exemplu actor "Customer" este mostenit de catre alti doi actori "Existing user" și "Guest user". Customer-ul poate să se uite la produse. Între timp, moștenitorii săi au atât use case-urile proprii, cat și use case-ul Customer-ului.

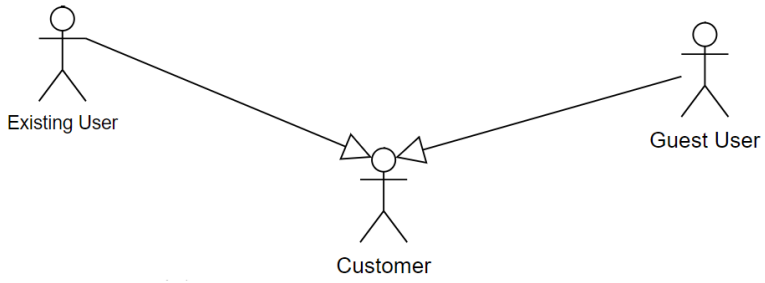

Fig 2.6. Reprezentare grafica a relatiei de generalizare

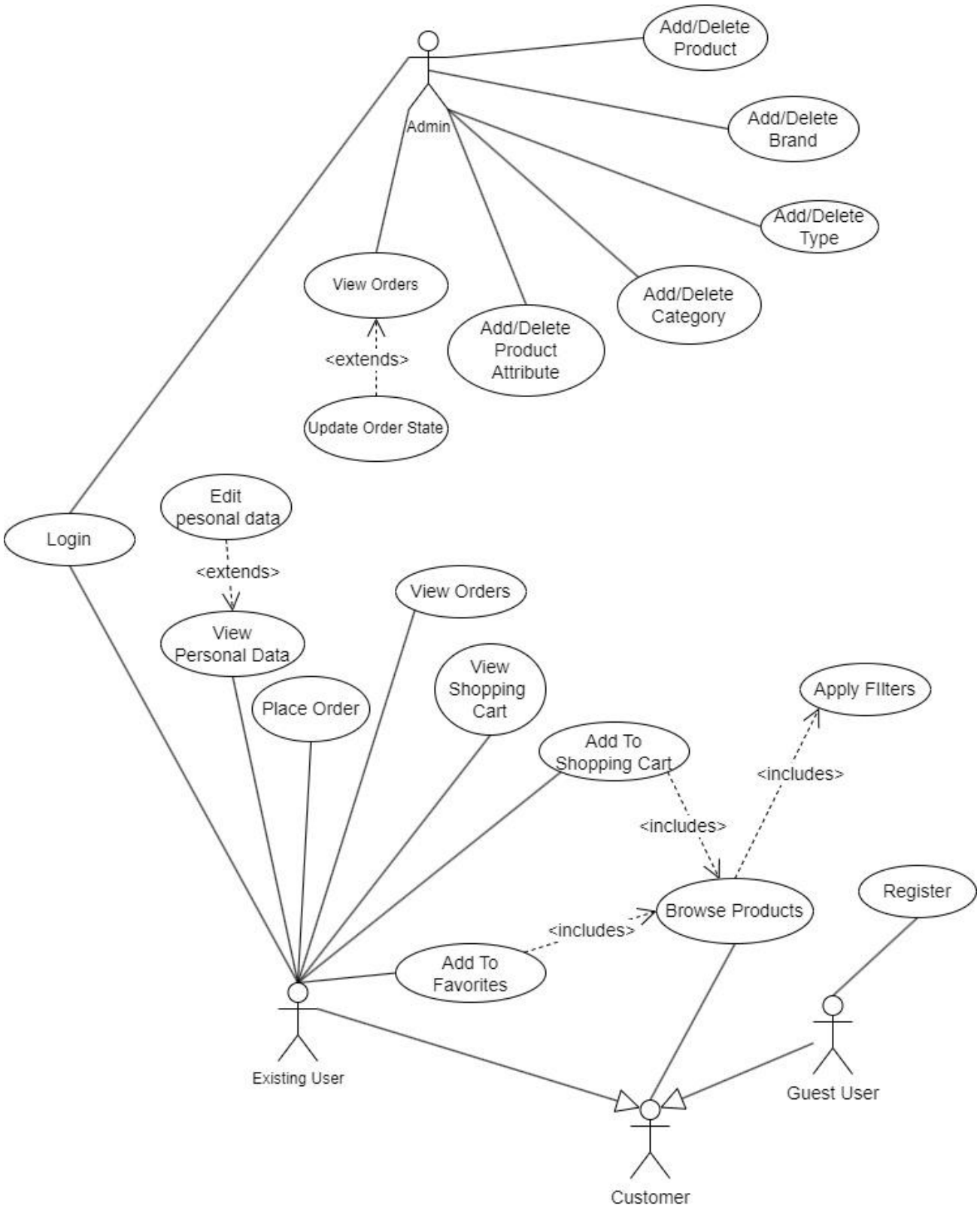

Fig 2.7. Diagrama Use Case

În cele ce urmează, se vor explica componentele diagramei și semnificatia cazurilor de utilizare:

-Avem 4 actori: Customer, Existing User ,Guest User și Admin

-Customer este orice utilizator al site-ului. Acesta are cele mai puține cazuri de utilizare și anume vizualizarea și aplicarea de filtre asupra listei de produse. Este un actor care este moștenit de către alți doi actori.

-Existing User este orice utilizator care este prezent în baza de date. Acesta poate să vizualizeze și implicit, să filtreze produsele. Se poate autentifica cu contul propriu. Are posibilitatea de a adăuga produsele într-un coș de cumpărături sau într-o listă de favorite pe care le poate vizualiza ulterior, fapt reprezentat prin alte două useCase-uri. Adăugarea la lista de favorite sau în coșul de cumpărături include vizualizarea produselor în prealabil. Clientul mai are posibilitatea de a plasa comenzi și de a-și vedea propriul istoric al comenzilor. De asemenea are posibilitatea de a-și vizualiza datele personale și de a le modifica.

- Guest User este orice utilizator care nu este înregistrat în baza de date. Acesta moștenește actorul Customer, iar pe lângă, are posibilitatea de a-și creea un cont nou în baza de date. Se evidențiază faptul că un simplu user vizitator are privilegii limitate în comparație cu un user prezent.

-Admin – este un actor care are alte privilegii și alt rol fata de orice Customer. Acesta are datoria de a modifica catalogul de produse, prin adăugarea, ștergerea și modificarea produselor și ale tabelelor componente, precum branduri, categorii, tipuri de produse, atribute, etc. În plus, el poate actualiza stările comenzilor, useCase care extinde vizualizarea acestora.

2.3.Diagrama entități relații(ERD)

2.3.1. Concepte generale

În acest stagiu al proiectării, se vor prezenta la nivel de concept entitățile sistemului si relațiile dintre acestea fără a se oferii detalii amănunțite despre structura fiecărei entități. Inițial este necesară identificarea datelor sau informațiilor care trebuie salvate, după care se trece la construirea diagramei propriu-zise. O astfel de diagramă este utilă în înțelegerea legăturilor dintre elementele principale ale aplicației. Ulterior această diagramă se va dezvolta prin transformarea entităților în tabele și adăugarea de câmpuri.[3]

La fel ca și în cazul oricărui tip de diagramă, există câteva reguli generale pentru reprezentarea grafică a acestora.

Entitățile sunt obiecte care pot reprezenta elemente fizice, palpabile din lumea reală(ex. articolele dintr-un magazin) sau elemente teoretice, impalpabile (ex. comenzi, categorii de articole).

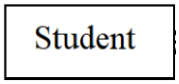

Fig 2.8 - Reprezentare grafică a unei entități

Intre entități, trebuie să se stabilească **relații**. Acestea sunt asocieri reprezentate prin linii simple. La contactul dintre linii si entități se afișează cardinalitatea, folosindu-se doua simboluri grafice, obținându-se 3 cardinalități.

[3]-"Database Management Systems" de Raghu Ramakrishnan și Johannes Gehrke p27,28

Cardinalitatea arată numărul maxim de asocieri care se pot realiza între instanțele a două entități. În funcție de situație se pot stabili 3 tipuri de cardinalități:

1. Cardinalitate 1 la 1 (one to one) – semnifică faptul că unei entități îi corespunde o singură instanță din cealaltă entitate si reciproc.

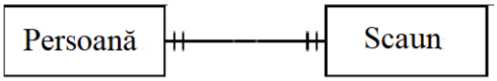

Fig 2.9 - Reprezentare grafică a unei relații One-to-One

2. Cardinalitate 1 la n (one to many) – semnfică faptul că unei intanțe a unei entități îi corespund mai multe instanțe ale altei entități. De exemplu în relația dintre entitățile Product și Category. Mai multe produse îi sunt atribuite unei categorii.

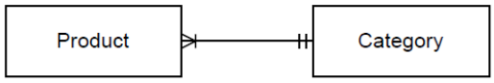

Fig 2.10- Reprezentare grafică a relației One-to-Many

3. Cardinalitate n la m (many to many) – semnifică faptul că unei instanțe dintr-o entitate îi corespund mai multe instanțe din altă entitate si invers. De exemplu în relația dintre entitățile Product și Attribute. Un produs poate avea mai multe atribute și un atribut poate fi deținut mai de către mai multe produse. Orice relație Many-to-Many se creează practic un tabel intermediar.

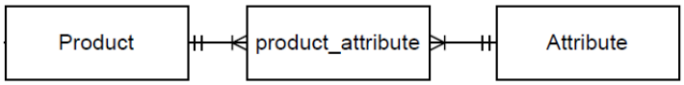

Fig 2.11 – Reprezentare grafică a relației Many – to Many

2.3.2. Prezentarea diagramei

Utilizând regulile amintite mai sus, se pot forma relațiile dintre entitățile sistemului informatic, reprezentate prin următoarea diagramă.

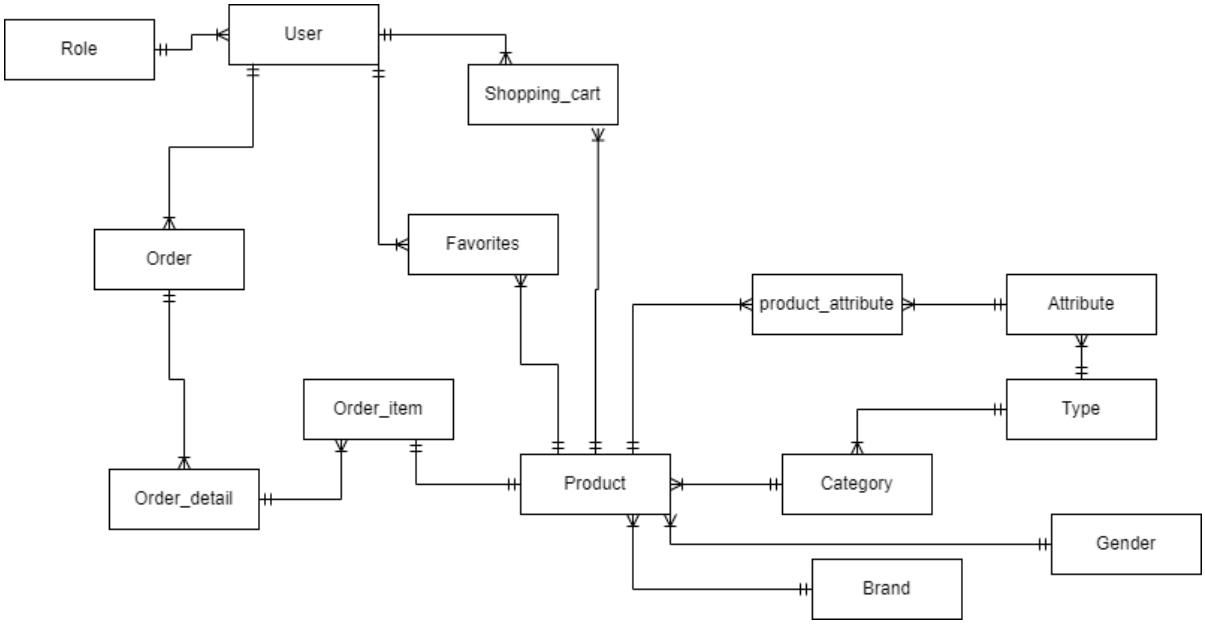

Fig 2.12. Diagrama ERD completă

Pentru a facilita explicațiile diagramei, am separat-o în 3 diagrame diferite.

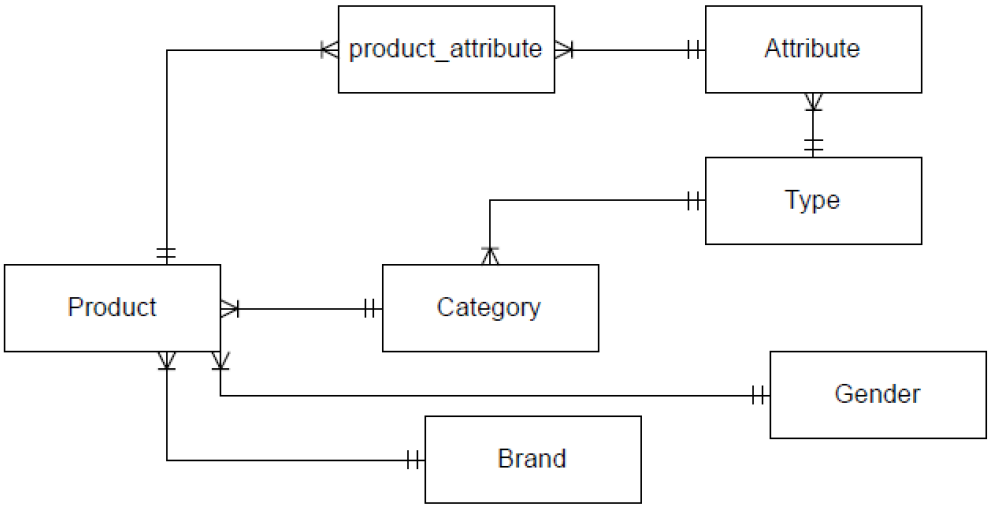

Fig 2.13 – Reprezentarea entității Produs si asocierile cu alte entități

În Figura 2.13 se evidențiază partea de produs. Fiecare produs va trebui să aibă câte un brand, iar fiecare brand va avea mai multe produse, deci se poate stabili o cardinalitate de One-to-Many. Același lucru se întâmplă și în cazul entităților “Gender” și “Category”. Mai multe categorii vor putea fi asociate cu același “Type”, în timp ce un singur tip de produs îi poate fi atibuit unei categorii de produs. De exemplu, exemple de entități Type sunt: “Ski”,”Snowboard”, iar categorii pot fii “Ski Alpin”,”Ski Nordic”, “Snowboard Freestyle”, “Snowboard All-mountain. În acest exemplu, primul tip este asociat cu primele două categorii menționate.

Dată fiind construcția fiecărui tip de echipament, fiecare va avea atribute, diferite se stabilește o cardinalitate One-to-Many între Type si Atribute. Spre exmplu, snowboard va fi caracterizat de alte atribute, comparativ cu un ski sau o cască de protecție. În plus, între entitățile Product și Atribute se stabilește o cardinalitate Many-to-Many deoarece un singur produs poate să aibă mai multe atribute, în timp ce un atribut poate să fie asociat cu mai multe produse. Cerințele aplicației necesită evidențierea acestei relații prin introducerea unui tabel intermediar unde se găsește fiecare asociere dintre entitäile Product și Attribute.

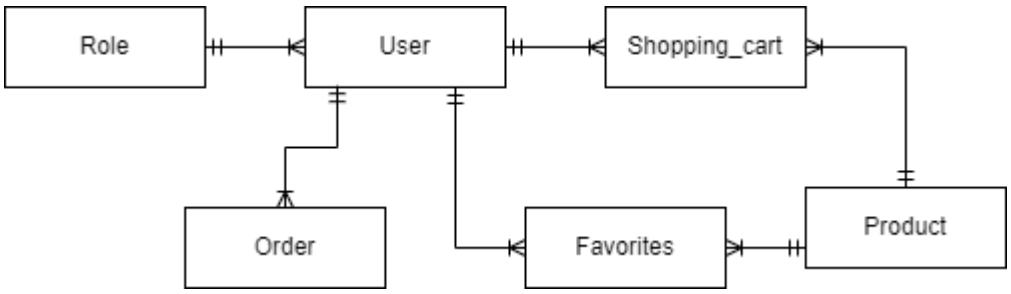

Fig 2.14 – Reprezentarea entitatii user si asocierile acesteia

Un alt concept important din cadrul aplicației care trebuie ilustrat în diagrama de entități și relații este partea de utilizator reprezentată prin entitatea User în Figura 2.14. Fiecărui User îi este atribuit câte un rol, iar un rol poate fi deținut de către mai mulți useri, de unde cardinalitatea One-to-Many.

Un user poate avea mai multe comenzi (instanțe din entitatea Order), în timp ce o comandă îi corespunde unui singur utilizator, de aceea am stabilit o cardinalitate de tip One-To-Many. Aceeași situație este prezentă și în relațiile cu entitățile Favorite și Shopping_cart, deoarece fiecare client va avea mai multe produse favorite sau produse adăugate în coșul de cumpărături, iar același produs poate să fie asociat cu mai multe liste de produse favorite sau coșuri de cumpărături. De asemenea se identifică relațiile Many-to-Many între User și Product, unde tabelele intermediare sunt Favorite și Shopping_cart.

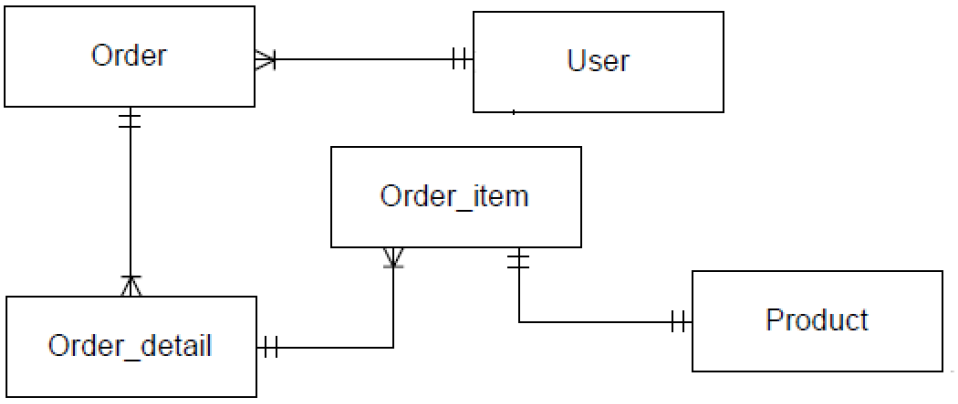

Fig 2.15– Reprezentarea entității Order și asocierile cu entitățile relevante

Entitatea Order din cadrul diagramei are scopul de a stoca comenzile plasate de către fiecare client. În diagrama din figura 2.15 se arată relațiile entității order, deoarece o comandă este puțin mai complexă.

În cadrul oricărei comenzi, un utilizator poate să adauge mai multe produse (conținutul lui Shopping_cart), iar aceste produse sunt identificate prin prezența entității Order_detail. Un Order_detail va trebui să țină minte numele produsului, mărimea dar și numărul de bucăți. În plus, mai este prezentă o entitate Order_item, care este practic o referință către Product. De fiecare dată când se plasează o comandă, se creează o nouă instanță order_item care îndeplinește rolul de istoric al produselor comandate. Dacă s-ar fi legat entitatea Order direct de Product atunci la ștergerea unei entități din Product s-ar șterge și o instanță din order_detail lucru care ar duce la date inconsistente.

3. Proiectare

3.1. Baza de date

3.1.1. Baze de date- general

O bază de date este o grupare organizată de date, care de obicei este reprezentată prin tabele salvate pe unul sau mai multe dispositive de stocare. Pentru a manipula datele, este nevoie de un Sistem de Gestiune al Bazelor de Date (SGBD).[20]

Un sistem de gestiune al bazelor de date este un software special conceput pentru manipularea bazelor de date și a informațiilor din acestea. Un SGBD poate să creeze baze de date noi, integogheză datele (prin niște query-uri) folosind un limbaj specific, pe care le poate modifica sterge sau adăuga altele noi. În limba engleză, un astfel de program este adesea întâlnit sub denumirea de Database Management System sau DBMS.[4]

În functie de cerintele care trebuie satisfăcute, pot exista o mulțime de tipuri de baze de date, iar cateva dintre acestea sunt:

3.1.2. Avantajele unei baze de date centralizate

Datele se află pe un singur dispozitiv. Orice informație poate fi obținută, adăugată sau modificată rapid deoarece totul este într-un singur loc compact.

Securitatea datelor este ușor de întreținut deoarece baza de date se află într-o singură locație fizică. Măsurile de securitate pot fi concentrate pe singurul punct de acces, lucru care face ca aceasta să fie mai greu de spart.

Datele au o structură foarte simplă, reprezentând de cele mai multe ori niște tabele. În cazul bazelor de date distribuite, acestea pot deveni foarte complexe, devenind greu de manipulat.

Redundanță minimă- acest avantaj este o urmare a modelului simplu al datelor. Informația redundantă este cea care este întâlnită în două sau mai multe locuri. Șansa de apariție a acesteia este proporțională cu complexitatea modelului datelor. [19]

4 - DATABASE SYSTEMS The Complete Book Second Edition- pag.1

20 - https://www.oracle.com/database/what-is-database/

3.1.3. Dezavantajele unei baze de date centralizate

Apar probleme de performanță dacă sunt o mulțime de utilizatori care accesează aceeași parte a bazei de date într-un timp foarte scurt, până la scăderea numărului de acțiuni pe baza de date. Pentru evitarea unor astfel de probleme este indicată urmărirea unor reguli pentru proiectarea bazei de date precum normalizarea

Dacă există probleme legate de sistemul care deține baza de date, aceasta poate fi inaccesibilă până la remedierea problemelor. Problema este rezolvată in cadrul unei baze de date distribuite: dacă se întâmplă ceva cu o locație, responsabilitatea acesteia este distribuită către celelalte puncte, iar serviciul funcționează fără întreruperi.

Riscul distrugerii informației- în situația în care se întâmplă ceva cu sistemul fizic de stocare, datele se pierd definitv. O simplă rezolvare a unei asftel de întâmplări este pregătirea periodică a unei copii de rezervă a datelor. Dacă un set de date se pierde, backup-ul remediază situația în funcție de cât de recent a fost efectuat față de momentul incidentului. [19]

3.1.4. Proiectarea bazei de date

Având in vedere ceințele sistemului si clasificările amintite, am ales bazele de date relaționale, centralizate.

În general, după stabilirea entităților din diagrama ER generală se trece la o verificare serioasă a acesteia cu scopul identificării unor probleme de proiectare. Dacă au fost descoperite aceste probleme, se modifică diagrama în mod corespunzător și se repetă procesul până la eliminarea tuturor problemelor. La final se poate trece la transformarea în schema principală a bazei de date.

Prin acest proces, entitățile devin niște tabele, iar atributele entităților devin câmpurile tabelelor. Cardinalitățile dintre entități se transformă în relații cu aceleași denumiri care se reprezintă prin prezența cheilor străine și grafic prin aceleași simboluri.

Câmpurile din tabel

Câmpurile din tabel sunt informații/atribute ale entității descrise. De exemplu, în cadrul dezvoltării unui magazin online în tabela “Produs” vom avea mai multe câmpuri, precum: nume, marcă, categorie, etc.

19 - https://www.insightsforprofessionals.com/it/storage/distributed-vs-centralized-database

La stabilirea câmpurilor se încearcă normalizarea datelor, adică evitarea datelor redundante. Pe de-asupra, se sugerează ca datele să fie stocate în așa manieră încât acestea să nu se modifice constant. De exemplu într-o presupusă tabelă "Student" să nu i se salveze vârsta (care se schimba frecvent), ci data nașterii deoarece rămâne fixă.

În plus, fiecărui câmp ii este atribuit un tip de date, care reprezină tipul de informație care se poate salva în câmpul resprectiv. Aceste tipuri diferă în funcție de tipul bazei de date si de DBMS pe care le suportă. Aceste tipuri de date pot fi clasificate in:

- Primitive: Numere întregi (Integer), Numere cu virgulă (Floatig-point-number), Siruri de caractere (String), Valori booleene (Adevărat sau Fals)
- Complexe: Date, Ore, Numere binare, Tipuri structurate (tablouri, JSON, XML)

Unui câmp i se pot atribui constrângeri, transformându-l într-o cheie.

Cheia unui tabel

Cheia unui tabel este un câmp care ajută la identificarea unui rând din tabel. De exemplu in cazul in care avem o listă de studenți si vrem să extragem un anumit student, ne vom folosi de un câmp unic, "nr. matricol". Alte posibile atribute precum "nume" sau "adresă" nu pot fi considerate chei, deoarece mai mulți studenți pot avea același nume sau aceeași adresă.

În funcție de situație, cheile pot fi clasificate in mai multe feluri:

1. Cheie Primară – un câmp din tabel, unic, după care se identifică o instanță într-un tabel. Cheia primară nu poate să fie goală (null). De exemplu într-un catalog de produse, fiecare produs are un id unic. Orice cheie primară este o cheie candidată.
2. Cheie Candidată – orice cheie care ar putea fi utilizată ca și cheie primară. De exemplu in inregistrările unui magazin online pentru calculatoare: in ciuda faptului că fiecare computer are o adresă MAC unică magazinele identifică fiecare computer după o altă cheie. Adresa MAC ar fi putut îndeplini funcția de cheie primară, dar există altă opțiune mai bună.
3. Cheie Străină – este orice cheie candidată din alt tabel. Deși in general se folosesc cheile primare, acest lucru nu este obligatoriu. Prin aceasta se definesc relațiile dintre entitățile sistemului informatic. De exemplu in relatia dintre două tabele :Țări si Orașe, tabela Orașe ar putea să conțină o cheie străină provenită din tabela Țări (id_țară).

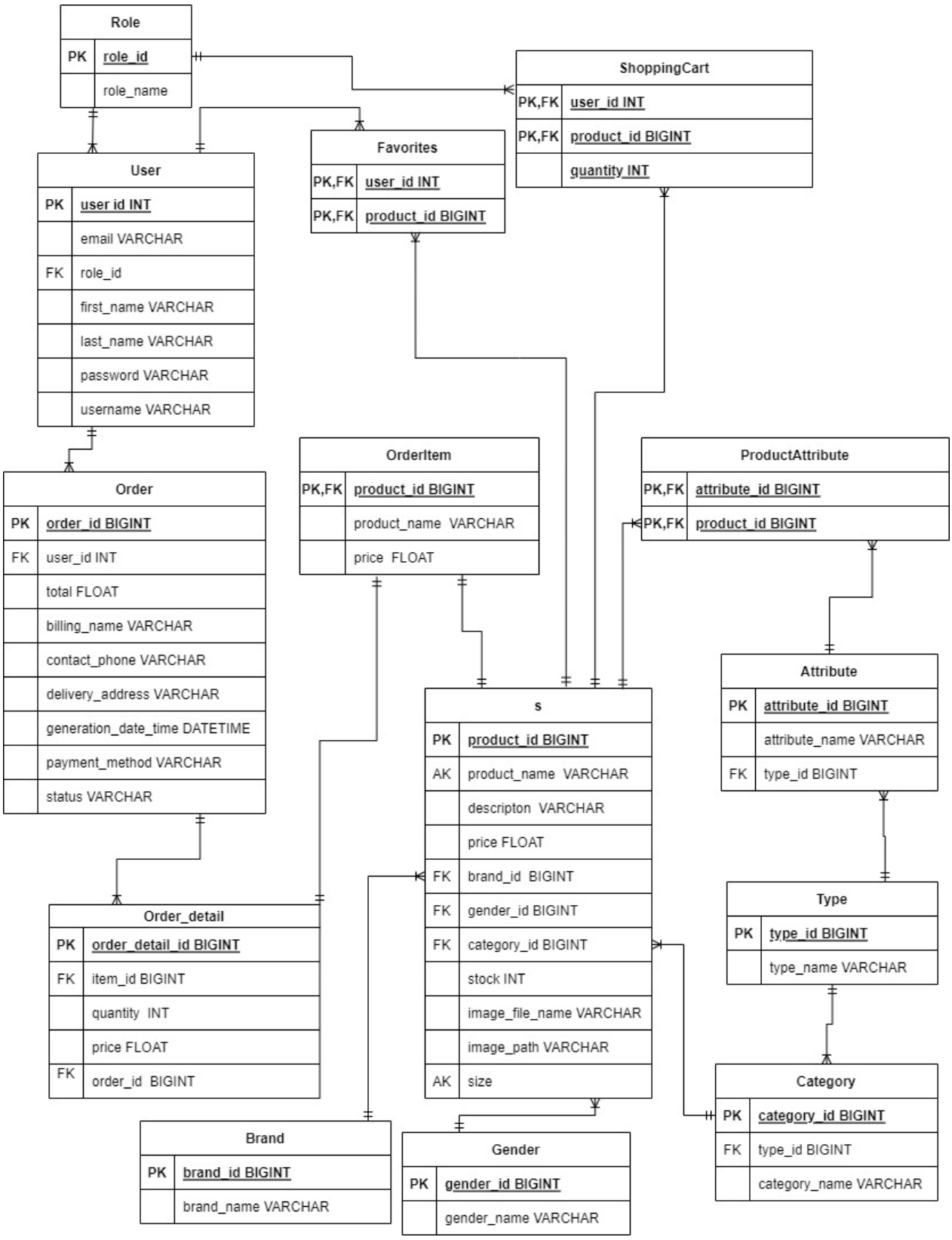

Figura 3.1– Diagrama bazei de date

Tabelele bazei de date

PRODUCT (product_id, product_name, description, price, brand_id, gender_id, category_id, stock, image_file_name, image_path, size)

USER (user_id, email, role_id, first_name, last_name, password, username)

ORDER_DETAIL (order_detail_id, item_id, quantity, price, order_id)

FAVORITES (user_id, product_id)

SHOPPING_CART (user_id, product_id, quantity)

CATEGORY (category_id, type_id, category_name)

TYPE (type_id, type_name)

ATTRIBUTE (attribute_id, attribute_name, type_id)

PRODUCT_ATTRIBUTE (attribute_id, product_id)

ROLE (role_id, role_name)

ORDER (order_id, user_id, total, billing_name, contact_phone, delivery_address, generation_date_time, payment_method, status)

BRAND (brand_id, brand_name)

GENDER (gender_id, gender_name)

ORDER_ITEM (product_id, product_name, price)

3.1.5. Organizarea produselor în baza de date

În continuare voi descrie tabelele bazei de date si voi motiva alegerile făcute.

Produsele comercializate sunt elementul cel mai important din cadrul unui magazin online și de aceea consider că este nevoie de o descriere a bazei de date cu concentrare asupra tabelei Product.

Tabelul **Product** este cel mai complex tabel. Aici se salvează informațiile comune pentru toate articolele. Fiecare produs are un nume, o descriere, un preț, o mărime, un brand, o categorie și un gender. Împreună, câmpurile nume și mărime formează împreună o cheie compusă unică, dar nu și separat. Mai multe produse pot avea același nume sau aceeași mărime, dar combinația nume+mărime este unică. Am introdus această regulă pentru a permite stocarea mai multor mărimi pentru același produs. Prețul a două produse de același model, dar cu mărimi diferite, va putea fi

diferit și se va putea ține o evidență mai clară asupra vânzărilor. Mai există și două câmpuri care fac referire la locația unei imagini a produsului. Am introdus acest lucru deoarece nu este în regulă stocarea imaginilor în baza de date a produsului.

Tabelele **Gender**, **Brand** si **Category** sunt tabele asemănătoare care pot fi discutate simultan. Un brand/gender/categorie are mai multe produse, iar un produs are un singur brand/gender/categorie (fapt care motivează prezența cheilor străine brand_id, gender_id, cartegory_id din tabelul product. Pe lângă propria cheie primară, tablele Gender, Brand și Category mai au câte un câmp cu numele propriu.

Unei categorii(instanță din tabela Category) îi corespunde unui singur tip de produs (tabela Type), iar unui tip îi corespund mai multe categorii. (prezența cheii străine type_id din tabela de categorii. Ambele tabele, Type și Category au câte un câmp cu numele propriu.

Fiecare tip de produs are mai multe Atribute. In tabela Atribute se salvează orice atribut care poate fi deținut de către articol comercializat. Numele său este salvat într-un câmp. ExmpluȘ culoare sau masă. Este foarte important de menționat faptul că, prin această organizare, putem avea mai multe tipuri de produse care pot avea calități total diferite, deci fiecare produs poate fi descris destul de amănunțit.

De exemplu, în tabela Type putem avea: ‘Snowboard’, ‘Ochelari’, ‘Clăpari’, etc. Câteva atribute asociate cu type-ul Snowboard (prin cheia străină type_id) pot fi dificultate, profil, elasticitate, tip de teren, etc. în timp ce atribute asociate cu type-ul Cască pot fi: duritate, culoare , etc. Astfel, atribute specifice unei plăci nu vor fi introduse într-un produs de tip cască, deoarece cu ajutorul acestei structuri este imposibil.

Datorită faptului că unui produs ii pot corespunde mai multe atribute, iar fiecare atribut poate fi oferit mai multor produse, între cele două tabele se va stabili o relație Many-to-Many. Asta înseamnă că, este nevoie de o tabelă intermediară ‘Product_Attribute’ unde se vor salva toate asocierile dintre produse si atribute și încă un câmp cu valoarea atributului respectiv (exemplu de completare a unui rând: product_id : 1, attribute_id: 3, value: S). Spre deosebire de celelalte situații prezentate până acum, aici vom avea două chei străine(attribute_id, product_id) care sunt și chei primare, împreună formând o cheie compusă.

3.1.6. Organizare clienți și comenzi

Tabela User conține informațiile userilor. Aici se salvează un username și o parolă folosite la autentificare. Username-ul este unic, iar parola este salvată sub formă criptată pentru a introduce un fel de formă de securitate. Se observă prezența cheii role_id provenind din tabelul Role care este folosit în aplicație pentru autorizarea unor activități.

'**Favorites'** și '**Shopping_Cart**' sunt tablele intermediare din relația Many-To-Many dintre User si Produs. În shopping_cart avem si un câmp numit 'Quantity' care arată câte produse de un anumit fel sunt în coșul de cumpărături. Pe baza acestui coș, se vor plasa comenzi în cadrul aplicației.

Tabela Order conține id-ul user-ului, datele de plată, adresa de livrare precum, data si ora generării și starea comenzii(aceasta va fi modificată de către un administrator). Un utilizator poate avea mai multe comenzi, în timp ce o comandă îi poate fi atribuită unui singur utilizator (relație One-to-Many).

Din cauza faptului că într-o comandă se putem avea mai multe produse comandate, am introdus o altă tabelă **order_detail** care conține id-ul comenzii din care provine (FK order_id), id-ul produsului comandat (FK item_id), numărul de bucăți și prețul total (calculat în funcție de cantitate si prețul unitar). Intrările în acest tabel sunt creeate în funcție de cele din Shopping_cart. După plasarea comenzii, conținutul coșului de cumpărături va fi șters.

În timpul implementării am observat o greșeală de proiectare și anume că la ștergerea unui produs care a fost deja comandat, se ștergea și instanța din order_detail deoarece cheia străină nu mai avea sursă, lucru care duce la date inconsistente. Pentru corectarea acestei greșeli, am introdus o tabelă nouă **order_item** care este un fel de copie superficială a produsului original. Aceasta conține doar id-ul, numele si prețul produsului original pentru a se afișa un istoric intact al comenzilor în aplicație.

3.2.Diagrama Fluxului de date(DFD)

Diagramele de flux de date se concentrează pe mișcarea datelor și sunt folosite pentru a reprezenta grafic diferite concepte în cadrul unui sistem.

Este folosită de obicei pentru analiza sistemelor informatice, dar este utilă și pentru comunicarea și modelarea proceselor și conceptelor care implică mai multe elemente fizice sau logice/teoretice. Poate fi folosită într-o gamă largă de domenii, de la procesarea comenzilor în restaurante până la afișarea proceselor complexe în bănci. Sunt adesea utilizate de către proiectanți și programatori în dezvoltarea de aplicații software, oferindu-le un suport vizual și împreună cu alte tipuri de diagrame, mărind productivitatea și probabilitatea de depistare a unor posibile greșeli de proiectare.

Diagramele DFD sunt compuse din patru simboluri distincte: flux de date, proces, depozit de date și entitate externă. În cadrul acestui proiect se va folosi notația Gane-Sarson care a apărut în anul 1979, iar descrierile simbolurilor sunt oferite în cartea Essentials of Systems Analysis and Design-Fifth Edition -de Joseph S. Valacich, Joey F. George și Jeffrey A. Hoffer - p. 155, 156. [6].

Un **flux de date** reprezintă date care se deplasează în interiorul sistemului între diferite locații. Acesta să reprezinte atât informațiile despre un anumit produs cât și informațiile unui formular completat de client la plasarea unei comenzi sau la înregistrarea unui cont, iar locațiile sunt niște , depozite de date, procese sau surse externe. Fluxul de date se reprezintă grafic printr-o săgeată unidirecțională pe care se află scrisă denumirea datei transmise. Denumirea trebuie să fie clară și ușor de înțeles. De asemenea este interzis ca numele să fie acțiuni, deoarece trebuie să se arate informația transmisă. Fiecare flux de date trebuie să unească fie o sursă externă cu un proces, fie un proces cu o locație de stocare.

Fig 3.2 – Reprezentarea grafică a unui flux de date într-o diagramă DFD

6 - Essentials of Systems Analysis and Design-FIFTH EDITION -de Joseph S. Valacich, Joey F. George și Jeffrey A. Hoffer - p. 155, 156

Un **proces** este o acțiune sau mai multe acțiuni care se ocupă cu manipularea datelor. De regulă aici se procesează după caz datele primite, iar apoi se transmit mai departe către următoarea locație. Aici se pot reprezenta atât acțiuni automate, cât și acțiuni manuale. Grafic, se ilustrează printr-un dreptunghi cu colțurile rotunjite și un text scurt desemnează procesul.

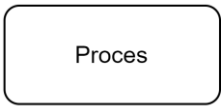

Fig 3.3 – Reprezentarea grafică a unui proces într-o diagramă DFD

Un **depozit de date** este o locație unde se salvează datele primite de la procese în cadrul sistemului. De cele mai multe ori este o bază de date sau o parte din ea. Acestea se reprezintă printr-un dreptunghi incomplet.

Fig 3.4 – Reprezentarea grafică a unui depozit de date intr-o diagramă DFD

O sursă/**entitate externă** reprezintă entitatea care comunică cu sistemul descris în diagramă. Acesta poate fi un operator uman, un dispozitiv hardware sau alt sistem software. Pe grafic, se reprezintă sub forma unui dreptunghi, în mijlocul căruia se află numele entității ilustrate.

Fig 3.5- Reprezentarea grafică a unei surse într-o diagramă DFD

Este important de menționat faptul că o diagramă poate avea diferite grade de complexitate pe niveluri pornind de la 0. De exemplu, într-o diagramă de nivel 0, procesele și locațiile sunt foarte generale, iar cu cât nivelul diagramei crește, acestea se descompun reprezentând fluxuri de date tot mai precise. Astfel, diagrama din figura 3.6 este de nivel 0, iar diagramele 3.7 și 3.8 vor fi de nivel 1 pentru că se detaliază anumite procese din 3.6.

În continuare se va prezenta diagrama fluxurilor de date de nivel 0 pentru aplicația proiectată.

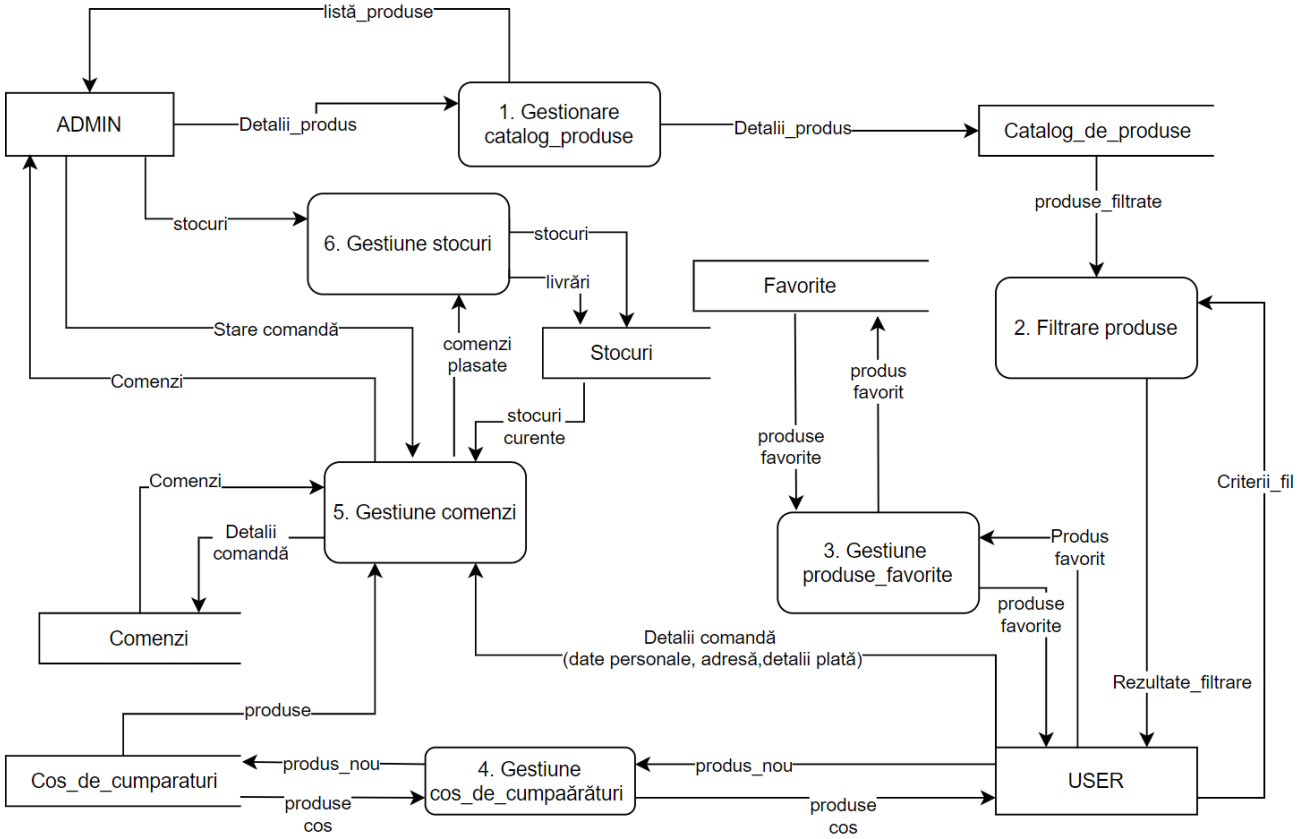

Fig 3.6 – Diagrama Fluxurilor de Date – nivel 0

Avem două entități externe ADMIN care este un administrator ce se ocupă cu modificarea catalogului de produse și actualizarea stărilor fiecărei comenzi și USER care reprezintă clienții.

Procesele "Gestionare catalog_produse", "Gestiune comenzi", "Filtrare produse", "Gestiune cos_de_cumpărături", "Gestiune produse_favorite" și"Gestiune stocuri" vor fi explicate în următoarele două diagrame, deoarece unele procese sunt prea complexe pentru a prezenta totul într-o singură diagramă.

Pe scurt, procesul 1 se ocupă cu modificarea catelogului de produse, procesul 6 actualizeaază stocurile, procesul 5 se ocupă cu inițializarea comenzilor, confirmarea lor și

comunică cu procesul 5 în vederea actualizării stocurilor. Procesele 2, 3 și 4 se ocupă cu filtrarea produselor și salvarea acestora în lista de produse favorite, respectiv în coșul de cumpărături.

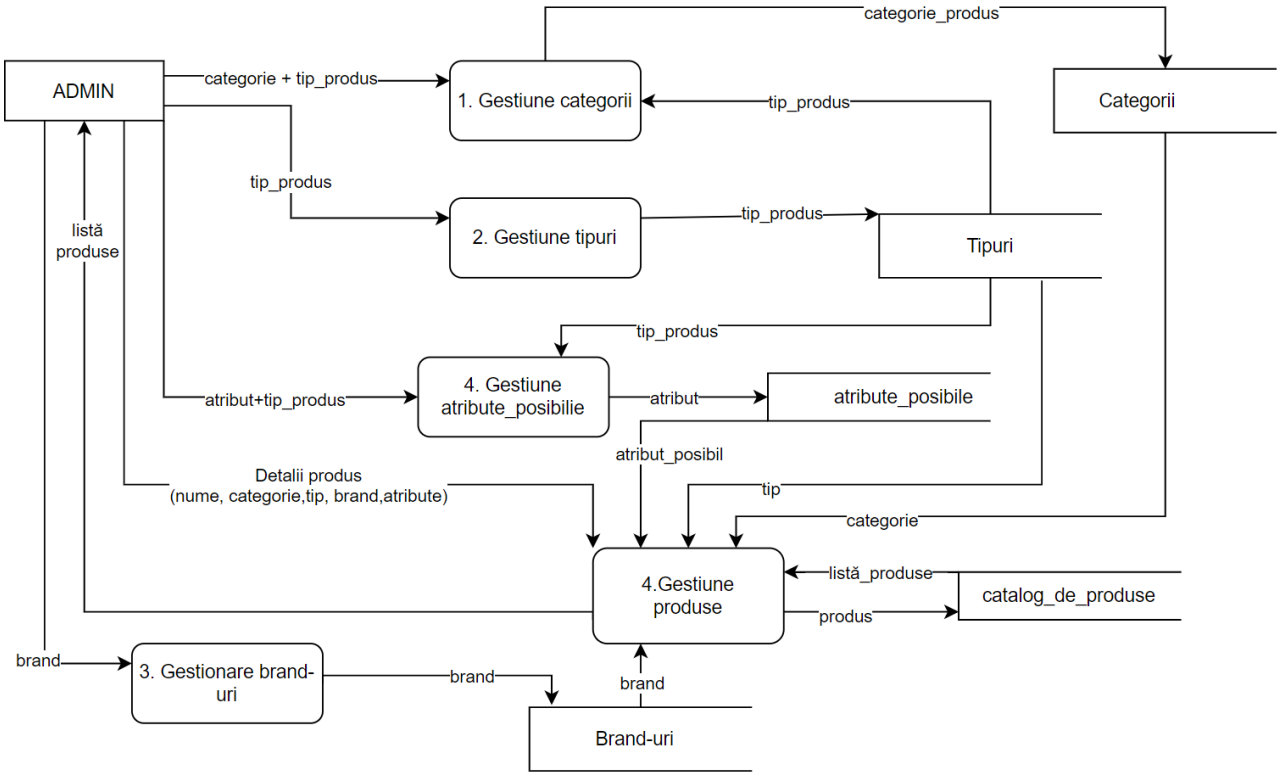

Fig.3.7- Diagrama Fluxurilor de Date – gestionare produse

Dată fiind structura bazei de date, gestionarea catalogului de produse presupune existența mai multor subprocese.

Proces 1: Administratorul primește lista categoriilor sau poate trimite categorii noi, cu condiția ca tipul specificat în fluxul de date să fie deja prezent în sistem. Acest lucru implică extragerea din locația “Tipuri” a tipului respectiv. Dacă nu se poate extrage tipul dorit, atunci procesul nu execută acțiunea.

Proces 2: Administratorul primește lista de tipuri de produse sau poate introduce în sistem altele noi.

Proces 3: La fel ca și la procesul precedent, administratorul primeste lista de brand-uri sau poate să adauge altul nou.

Proces 4: Administratorul adaugă un nou atribut posibil, cu condiția ca tipul specificat să fie deja prezent în sistem la fel ca și în cazul procesului 1.

Proces 5: Administratorul poate să trimită informații despre produs către proces, care sunt validate doar dacă sunt prezente informațiile necesare în sistem, situație similar ca și în cazul proceselor 1 și 4, doar că acum procesul extrage date din mai multe locații pentru a valida creearea unui produs și acționează corespunzător.

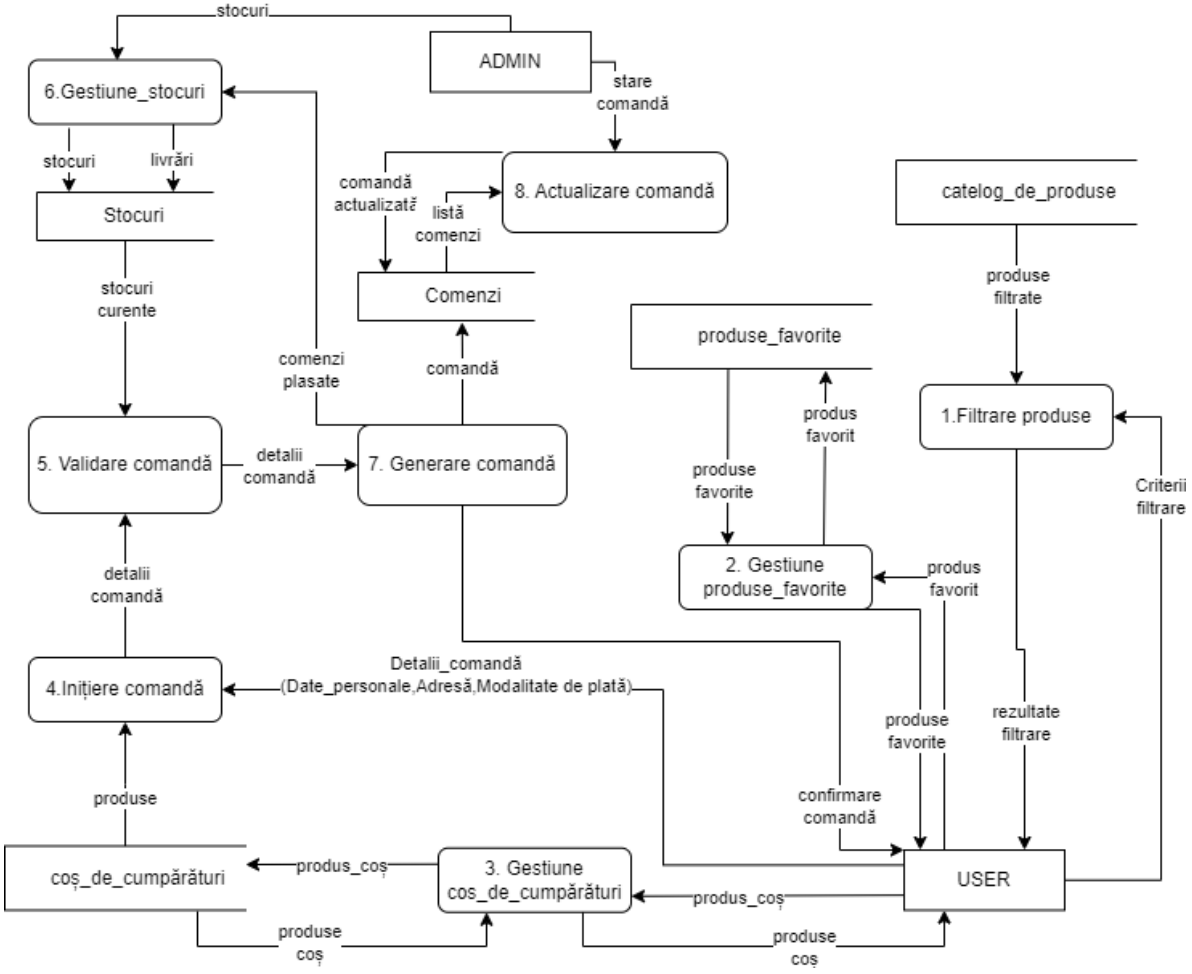

Fig 3.8- Dagrama Fluxurilor de Date- gestionare comenzi

Procesul de gestionare a comenzilor se descompune de asemenea în mai multe procese:

Proces 1 :Clientul(USER) transmite criterii de filtrare către proces, acesta extrage produsele din catalogul de produse și transmite lista acestora ca răspuns către client.

Proces 2 & Proces 3: Clientul poate adăuga produse la favorite prin p.2 sau în coșul de cumpărături prin p.3 sau poate să primească listele produselor salvate. Deși partea de produse favorite nu face parte din procesul de gestionare al comenzilor, am considerat că este relevant pentru user.

Gestionarea comenzilor include mai multe etape.

Proces 4: La inițierea comenzii de către client se validează datele introduse în formularul de plasare al comenzii și se extrage conținutul coșului de cumpărături, deci este necesară popularea acestuia în prealabil prin cel de-al 3-lea proces.

Proces 5: Comanda este verificată în funcție de stocurile curente.

Proces 7: Dacă se trece de validare, se salvează comanda în lista de comenzi, se trimite un mesaj de confirmare către client și se actualizează stocurile.

Proces 6: Administratorul modifică stocurile inițiale prin setarea numărului de bucăți pentru fiecare produs în parte. Acest lucru se va întâmpla practic la completarea formularului de creare a produsului, dar se poate modifica oricând.

Proces 8: Administratorul are datoria de a modifica starea fiecărei comenzi corespunzător

3.3.Diagrama de activitate

Diagrama de activitate (sau Activity diagram) este folosită pentru a descrie logica unui proces sau niște proceduri din cadrul unui sistem. Într-o astfel de diagramă, spre deosebire de o diagramă Use Case se prezintă doar ceea ce se întâmplă în sistem, dar nu se specifică autorul acțiunii. În schimb, este utilă pentru a descrie ordinea în care trebuie să se întâmple acțiunile. [12]

Acestea pot să fie secvențiale sau paralele sau ramificate, lucru foarte important pentru anumite situații.În plus, în combinație cu alte tipuri de diagrame, o diagramă de activitate este o unealtă foarte folositoare în detectarea și vizualizarea unor probleme legate de structura unui sistem sau optimizarea acestuia pentru a obține performanțe mai bune.

Ca și pentru oricare tip de diagramă UML, există câteva componente standard folosite în creearea acestei diagrame:

Starea inițială este punctul de start din care începe diagrama.

Starea finală este punctul de final al procesului descris de diagramă.

Activitatea este o acțiune descrisă în cadrul diagramei.

Punctul de decizie este elementul prin care se fac alegeri în funcție de rezulatul unei operații.

Bara de sincronizare este un element în care mai multe activități paralele se întâlnesc într-un punct comun.

Bara de ramificație este un element în care se încep mai multe activități paralele. Toate activitățile paralele trebuie să ajungă într-un punct de sincronizare.

Având în vedere cele prezentate mai sus, se vor prezenta două diagrame de activitate, care descriu acțiunile clienților (Fig. 3.9), respectiv a administratorului (Fig. 3.10).

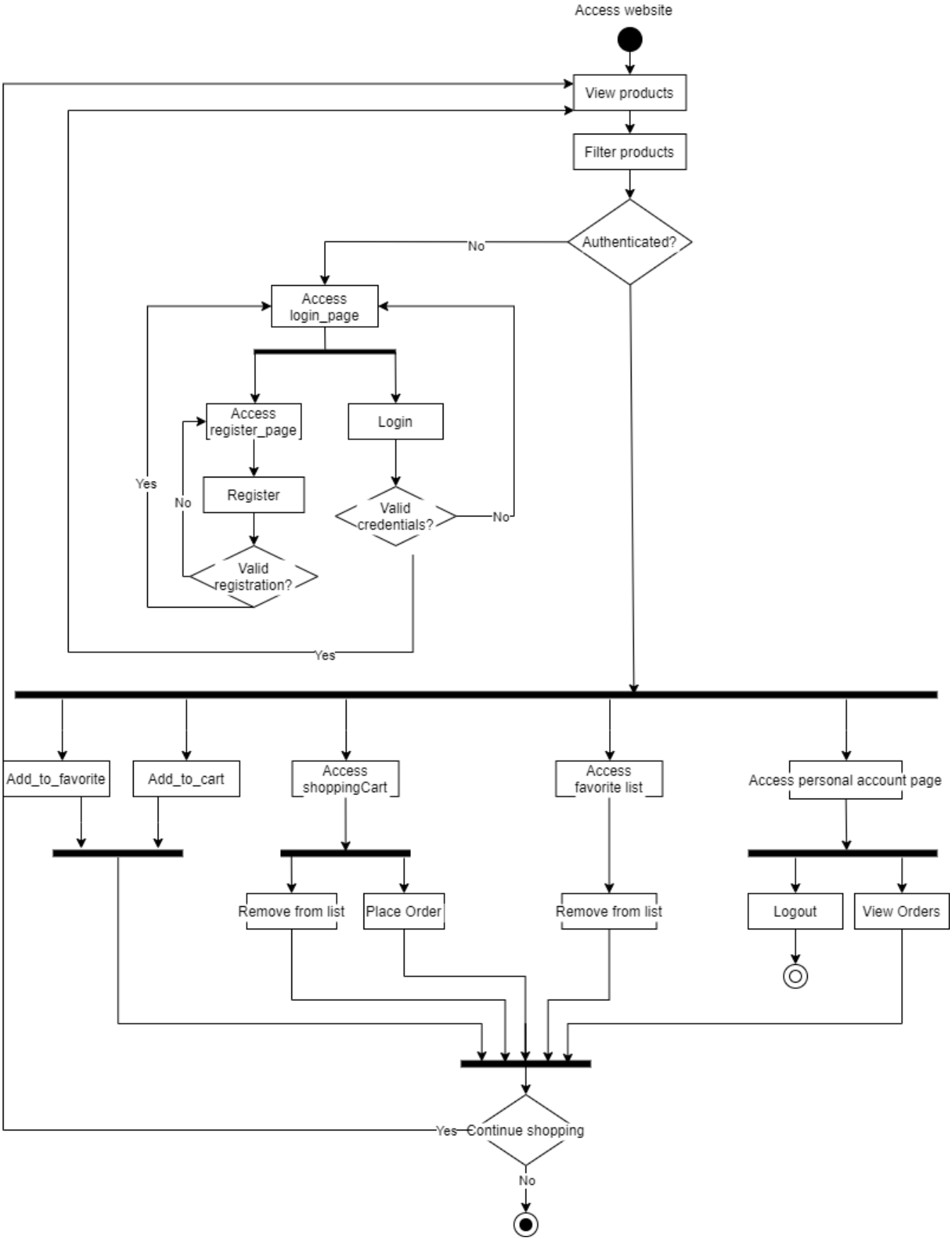

Fig. 3.9 – Diagrama de activitate pentru clienți

Am menționat initial că diagramele în sine, nu spun cine execută acțiunile, dar în schimb se poate prezica acest lucru în titlul diagramei.

În diagrama din figura 3.9 se descrie activitatea unui client. Acesta când accesează pagina este trimis practic la o pagină de vizualizare a produselor pe care le filtrează.

Pentru executarea următoarelor acțiuni este necesară autentificarea cu propriul cont. Dacă în blocul de decizie se rezultă că utilizatorul nu este logat, suntem trimiși la o pagină de logare. Aici avem două opțiuni, de a intra în cont sau de a inregistra unul nou. Dacă formularele sunt completate corespunzător, clientul își creează propriul cont și se poate loga, obținând access la următoarele activități.

În punctul current de ramificație utilizatorul poate să aleagă mai multe opțiuni: să adauge un produs în lista de favorite sau în coșul de cumpărături , să acceseze cele două liste, sau să acceseze o pagină cu detaliile personale ale contului. Aici are posibilitatea de așI vedea datele personale și propriile comenzi plasate. În ultima pagină menționată, utilizatorul are o opțiune de Log-out ceea ce înseamnă că activitatea sa s-a terminat, finalitate arătată prin cele două cercuri concentrice.

Se observă că majoritatea ramificațiilor sunt aude într-un punct de sincronizare, de unde se trece într-un bloc de decizie. Dacă se dorește continuarea activității pe site, suntem trimiși la pagina cu produse de unde se pot repeta acțiunile anterioare, altfel se sfârșește activitatea pe site.

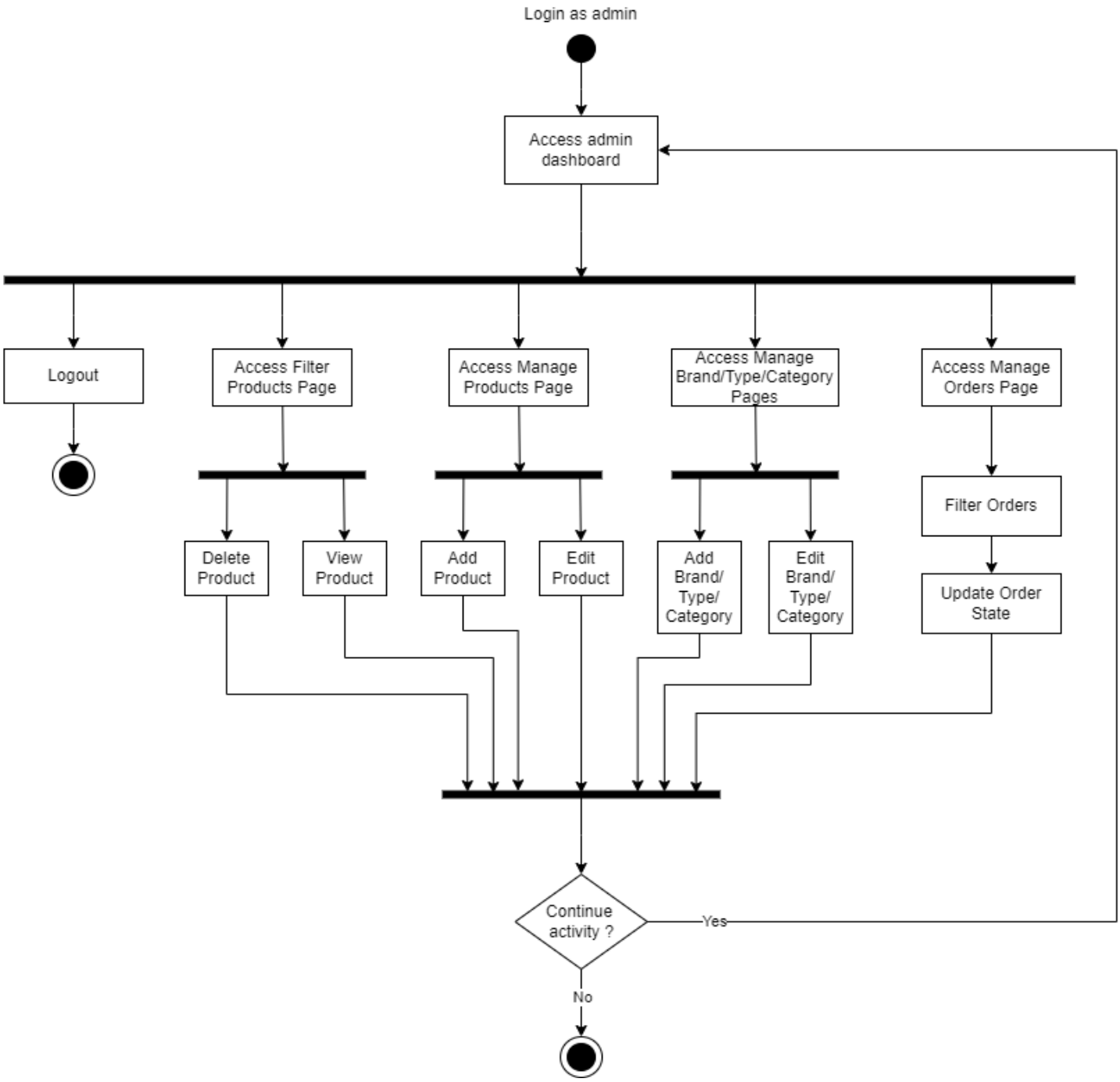

Fig. 3.10 – Diagrama de activitate pentru administrator

Diagrama 3.10 arată activitatea administratorului, ceea ce presupune logarea cu contul de administrator în prealabil. Activitatea Access admin dashboard este practic un panou de navigație de unde se poate merge pe fiecare pagină.

Prima acțiune disponibilă este cea de părăsire a contului care marchează și sfârșitul activității. Dacă nu se optează pentru această opțiune, administratorul poate accesa diferite pagini prin intermediul cărora se pot modifica anumite elemente legate de produse și comenzi.

Prin Access Filter Products Page se accesează practic aceeași pagină cu produse ca și la clienți, cu o singură modificare și anume că, nu mai avem opțiunea de a salva produse la lista de favorite ci, avem un buton de stergere produs.

Prin Access Manage Products Page se ajunge la pagina de unde se adaugă produsele prin completarea unui formular . De asemenea de aici se pot modifica și datele produsului.

Prin următoarea activitate Access Manage Brand/Type/Category se deschid pagini de unde fie putem să adăugăm sau să modificăm tabele corespunzătoare din baza de date.

În ultima activitate este cea de accesare a unei pagini unde se obțin toate detaliile despre comenzi. Aici putem separa comenzile în funcție de starea acestora și apoi le putem modifica starea.

Toate activitățile prezentate se intersectează în același punct comun, de unde se intra într-un bloc de decizie. Dacă se dorește continuarea activității pe site, suntem trimiși la început, altfel se marchează sfârșitul sesiunii.

3.4.Diagrama de secvență

Diagrama de secvență este o diagramă care UML care arată comunicarea între diferite obiecte. O astfel de diagramă este mai specifică și mai detaliată comparativ cu restul diagramelor prezentate până acum, deoarece aceasta se focusează pe mesajele transmise între componente și ordinea în care se execută acțiunile ilustrând modul în care obiectele vizate colaborează pentru a îndeplini o funcționalitate. Se poate spune că acest tip de diagramă îi este complementară diagramei cazurilor de utilizare, deoarece într-o diagramă se prezintă detalii despre cum se îndeplinește un useCase. În plus se pune în special accentul pe aspectul temporal, deoarece evenimentele sunt ordonate cronologic. [11] Pe diagramă, axa timpului pornește de sus și se scurge în jos.

La fel ca și pentru celelalte tipuri de diagrame, există câteva notații standard pentru reprezentarea grafică a componentelor.

Obiectele

Obiectele sunt elementele sistemului care participă la interacțiune prin trimitere, prelucrare sau recepționare de mesaje. Ele se reprezintă grafic printr-un dreptunghi plasat orizontal, în interiorul căruia se află numele obiectului și de la care pornește vertical o linie punctată.

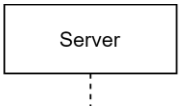

Fig. 3.11 – Reprezentarea grafică a unui obiect în diagrama de secvență

Liniile de viață

Liniile de viață reprezintă durata de viață a unui obiect în execuția unui scenariu. Atunci când acestea trimit un răspuns sau își termină partea din scenariul discutat, linia de viață se încheie

11 - UML 2.0 in a Nutshell de Dan Pilone, Neil Pitman - Iunie 2005 – pag. 128

de regulă. Acestea se reprezintă printr-un dreptunghi vertical deasupra liniei punctate care pornește de la obiect.

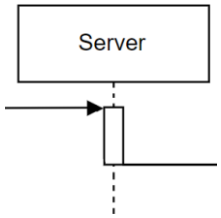

Fig. 3.12 – Reprezentarea grafică a unei linii de viață

Mesajele

Mesajele sunt elementele prin intermediul cărora elementele sistemului comunică între ele. Comunicarea dintre obiecte se poate realiza pe mai multe căi, incluzând apele de metode, semnale, generare de instanțe, distrugerea unui obiect, etc. Astfel, prin mesaje ne referim la toate aceste evenimente.

Mesajele oferă informații despre tipul de mesaj, transmițător și receptor. De cele mai multe ori, mesajele sunt apeluri de metode, iar în acest caz se pot pune și parametrii. Din punct de vedere grafic, se pot reprezenta mai multe tipuri de mesaje: asincrone sau sincrone.

Mesajele sincrone sunt cele în urma cărora se așteaptă un răspuns și determină suspendarea execuției obiectului. Acestea se reprezintă printr-o săgeată cu vârf plin.

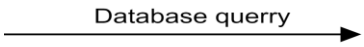

Fig. 3.12 – Reprezentarea grafică a unui mesaj sincron

Mesajele asincrone sunt cele după care un obiect nu așteaptă un răspuns pentru a-și continua activitatea. Acestea se reprezintă printr-o săgeată cu vârf “deschis”.

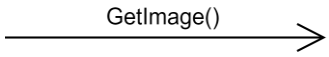

Fig. 3.12 – Reprezentarea grafică a unui mesaj asincron

Răspunsurile se reprezintă grafic similar cu cele asincrone doar că linia este punctată și au rolul de a notifica emițătorul, că activitatea destinatarului s-a încheiat.

În continuare se va prezenta diagrama de secvență pentru extragerea produselor din baza de date. Am ales acest scenariu deoarece în acest proces participă mai multe elemente ale aplicației.

În diagramă se vor aminti câteva elemente importante ale aplicației, care for fi explicate și detaliate începând cu capitolul 4 unde se povestește despre implementare.

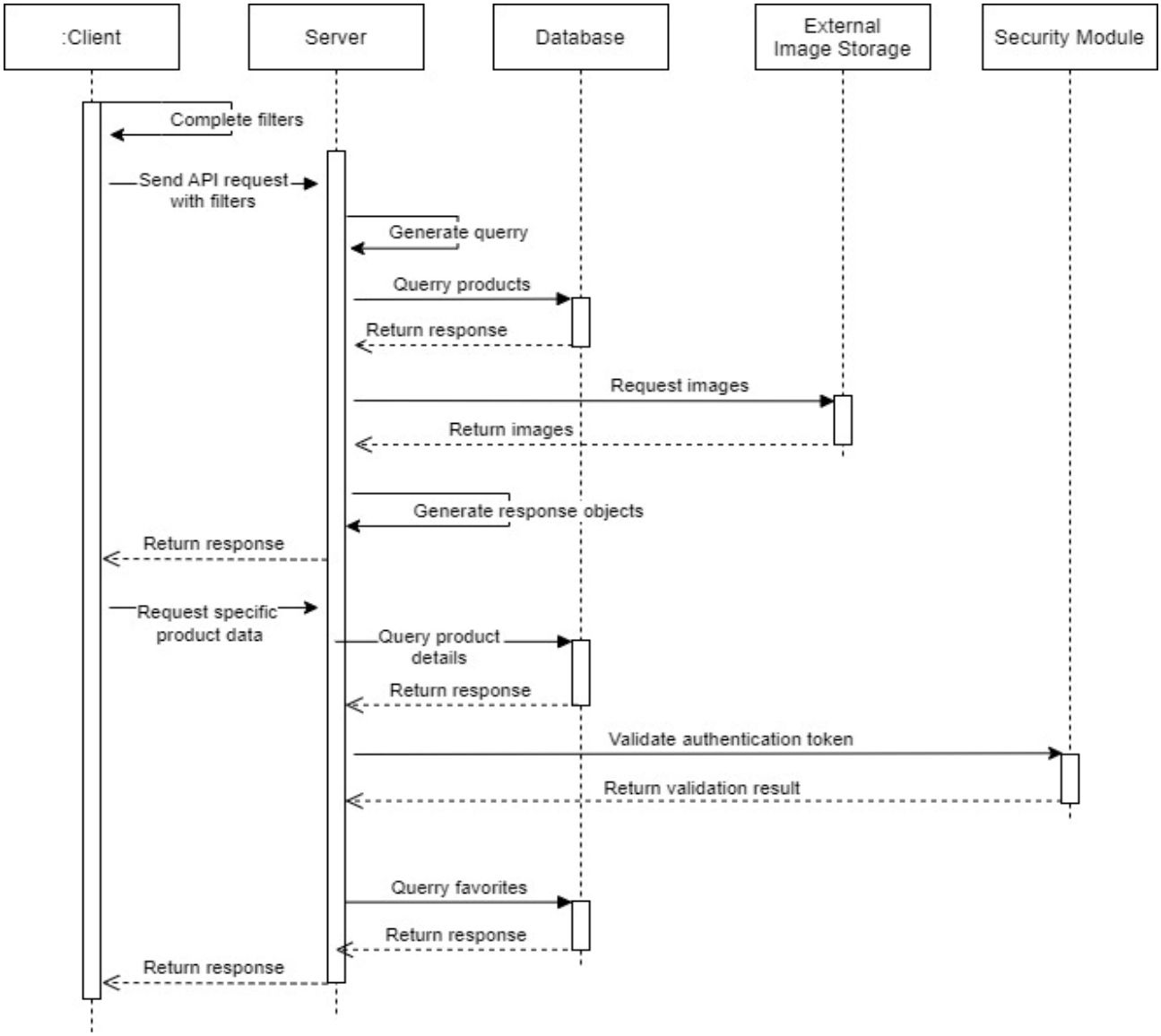

Fig. 3.13. – Diagrama de secvență pentru scenariul de extragere a produselor din baza de date

Evenimentul de extragere a produselor din baza de date pentru a fișarea acestora în interfața clienților se poate ilustra într-o diagramă de secvență prin reprezentarea câtorva componente importante sub forma unor obiecte care comuncă între ele: Client-ul, Server-ul , Baza de date (Database), Stocare externă de imagini și un modul de Securitate.

Afișarea produselor pe pagină, presupune implicit filtrarea acestora după criterii multiple care este primul pas prezentat în diagramă printr-un mesaj care pornește de la obiectul Client către sine (mesajul "Complete filters"). După completarea filtrelor (care pot să rămână și goale de altfel) se face un apel către server (denumit și API request) prin care se cer produsele. În acest apel se vor transmite criteriile de filtrare completate inițial.

Odată ce serverul primește apelul, acesta prelucrează criteriile de filtrare și generează niște apeluri destinate sistemului de gestiune al bazei de date (sau DBMS).

În continuare, așa cum este specificat și în cerințele sistemului fiecare produs afișat trebuie să aibă câte o imagine. Astfel, pentru fiecare obiect recepționat în răspunsul primit dinspre baza de date, serverul trimite un alt apel către un serviciu extern de stocare a imaginilor, deoarece fiecare obiect primit conține numele și locația propriei imagini.

Filtrarea produselor se poate încheia aici într-o oarecare măsură, dar utilizatorul are posibilitatea de a vedea detaliile unui produs. Astfel printr-un click pe unul dintre produse, se mai trimite încă un apel către server care conține numele produsului căutat și un jeton de autentificare. Fiecare produs are mai multe variante, așa că acestea sunt extrase din baza de date incluzându-se toate detaliile aparținătoare de acestea.

Aplicația trebuie să cunoască obiectele favorite ale utilizatorilor pentru a fi afișate și în pagina de detalii a produsului, unde intervine jetonul de autentificare primit de la Client. Jetonul este trimis către un modul de securitate unde este validat. Dacă validarea are success, modulul de securitate trimite datele de identificare ale contului atașat la jeton. Cu ajutorul datelor primite, serverul mai trimite un alt apel către baza de date prin care se dorește extragerea produselor favorite asociate cu contul primit în ultimul răspuns.

La sfârșit se generează produsele complete prin îmbinarea tuturor răspunsurilor acumulate în server, după care îi sunt transmise clientului prin răspuns la apel.

4. Implementare

4.1. Arhitectura

În etapa de proiectare a aplicației urmărește o arhitectură bazată pe nivele, unde fiecare nivel are câte un rol specific: Nivelul de prezentare, Nivelul de Aplicație și Nivelul de Date. Aceste nivele reprezintă practic separarea fizică a aplicației. Astfel, codul de la fiecare nivel poate fi rulat pe câte un sistem diferit. Utilizarea unei astfel de arhitecturi oferă avantaje precum separarea clară a conceptelor, obținerea unui cod organizat și ușor de întreținut.

De asemenea, datorită modularității obținute, modificările efectuate într-o parte a aplicației nu afectează celelalte părți, astfel se pot forma echipe de mai mulți programatori care pot lucra pe același proiect cu ușurință. Se poate spune că la nivel teoretic, s-a urmărit într-o oarecare măsură modelul de design Model-View-Controller(MVC) datorită separării conceptelor.

Dacă implementarea este corectă, modificarea sau schimbarea unei tehnologii de la un nivel, nu implică nici o problemă la celelalte nivele. Arhitectura utilizată este adesea întâlnită sub denumirea de "Arhitectura pe trei niveluri" sau în engleză "Three Tier Architecture", unde un "tier" este un nivel fizic.

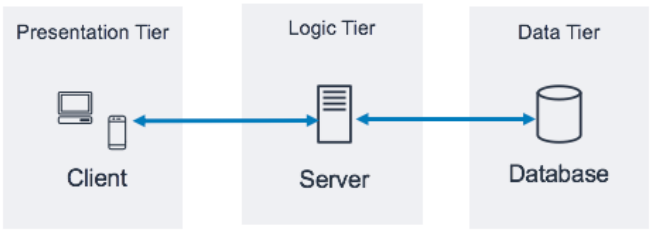

Fig.4.1 – Arhitectura pe 3 nivele [22]

Nivelul de prezentare reprezintă interfața prin intermediul căreia user-ul interacționează cu aplicația. Afișarea de informații și colectarea de date de la utilizator reprezintă rolurile principale ale acestui nivel. De obicei la acest nivel rulează o aplicație în browser. Acest nivel poate fi implementat utilizând tehnologii precum HTML, Javascript și CSS pe baza cărora se pot dezvolta librării și framework-uri precum React, Angular sau PHP.

Nivelul de aplicație reprezintă partea care se ocupă cu logica aplicației. Aici sunt implementate funcționalitățile de bază ale proiectului. Se poate spune că acesta este "creierul aplicației", deoarece aici ajung datele de la nivelul de prezentare unde sunt procesate și în funcție de nevoie, se interacționează cu baza de date. Aplicațiile de la acest nivel se pot implementa utilizând o mulțime de limbaje de programare precum C#, Java, C++, Python etc.

În plus este important de menționat că nivelul fizic de prezentare comunică cu nivelul de date, indirect prin nivelul de aplicație. Dacă nivelul de prezentare are nevoie de date, i le cere nivelului de mijloc, care la rândul său comunică cu baza de date.

Nivelul de date este componenta aplicației care se ocupă cu manipularea informației în baza de date. Aceasta este reprezentată de obicei printr-un sistem de baze de date(Database Management System sau DBMS amintit în faza de proiectare a bazei de date. Cele mai populare DBMS-uri sunt MySQL, PostgreSQL, Microsoft SQL Server, MongoDB.

4.2. Tehnologii utilizate

Dacă separăm conceptele din punctul de vedere al utilizatorilor, aplicația este impărțită în două părți diferite la care de acum se va face referire prin Frontend și Backend. Frontend-ul este aplicația care rulează la nivelul de prezentare. Pentru implementarea acestuia s-a utilizat React. Partea de Backend este constituită din nivelul de aplicație unde s-a utilizat Java Spring Boot și MySQL pentru nivelul de date.

React este o bibliotecă **JavaScript** folosită pentru dezvoltarea de interfețe de utilizator complexe și **Typescript** , care exinde sintaxa limbajului JavaScript, permițându-se definirea tipurilor de date in mod static și nu dinamic precum în JavaScript-ul simplu.

Câteva avantaje importante în folosirea acestei biblioteci:

-utilizează JSX(JavaScript Extension), o sintaxă care permite îmbinarea unor elemente HTML cu sintaxă JavaScript.

-are o structură bazată pe componente reutilizabile. Aceste componente pot fi înlănțuite și combinate între ele pentru a obține interfețe complexe.

-permite actualizarea flexibilă a componentelor cu ușurință, fără redarea repetată a întregii pagini datorită utilizării unui DOM Virtual. DOM-ul este o interfață web unde sunt reprezentate toate elementele unei pagini în vederea interacțiunii cu alte limbaje de programare, în timp ce DOM Virtual este o copie fidelă a DOM-ului original.

Biblioteca React a fost utilizată pentru creearea unor aplicații populare precum Facebook, Netflix, Instagram, Discord și altele.

Pentru nivelul de aplicație, se va folosi **Java Spring Boot,** un framework al limbajului Java care ofera suport pentru dezvoltarea de aplicații web. Se vor folosi mai multe dezvoltări sau dependențe ale framework-ului pentru adăugarea unor funcționalități de bază: Spring Web – oferă support pentru apeluri HTTP, Spring Data JPA – oferă suport pentru interacțiunea cu baza de date, Spring Boot Security – suport pentru funcționalitatea de autentificare și autorizare

La nivelul de Date, se folosește o bază de date MySQL.

4.3. Implementare backend

În acest subcapitol se va discuta despre aplicația principală din backend implementată cu ajutorul Java Spring Boot. Structura aplicației este bazată pe trei nivele teoretice : strat de prezentare, strat de servicii, strat de access de date. Fiecare strat are câte un rol bine definit. În ciuda faptului că au denumiri asemănătoare cu nivelurile arhitecturii descrise în capitolul 4.1 putem spune că acestea definesc structura teoretică internă a nivelului de aplicație din arhitectura pe 3 niveluri.

Înainte de discutarea straturilor este necesară definirea conceptului de **bean**.

În cadrul aplicațiilor implementate în Java Spring, un bean este o instanță a unei clase pe care framework-ul o utilizează. Există mai multe tipuri de bean-uri pe care Spring le recunoaște prin utilizarea adnotărilor precum @Bean, @Component, @Service, @RestController, etc. Odată ce un bean a fost detectat, acesta este introdus în contextul aplicației, definit de framework-ul Spring. Atunci când este nevoie, aplicația accesează bean-urile din context și le utilizează unde este nevoie.

Stratul de prezentare este responsabil cu interacțiunile cu exterior și este reprezentat de niște “controllere”. Acestea sunt bean-uri specializate în recepționarea de apeluri HTTP provenite de la partea de front-end. Stratul de prezentare interacționează cu stratul de servicii pentru a executa anumite funcționalități și spre a pregăti un răspuns pentru apelul primit. Un controller este practic o clasă care are adnotarea @RestController. În figura 4.2 se prezintă implementarea unui controller și definirea comunicației cu nivelul de servicii prin adnotația @Autowired.

```
@RestController
@RequestMapping("/products")
@CrossOrigin(origins = "http://localhost:3000")
public class ProductController {
    7 usages
    @Autowired
    ProductService productService;

    3 usages
    @Autowired
    FileStore fileStore;
    1 usage
    @Autowired
    UserService userService;

    1 usage
    @Autowired
    FavoriteService favoriteService;
```

Fig.4.2 – Definirea unui controller

În fiecare controller se implementează metode (figura 4.3) ce reprezintă punctele de acces prin care se face comunicarea dintre frontend si backend. Aceste puncte de acces sunt marcate prin adnotări precum @GetMapping, @PostMapping, @PutMapping, @DeleteMapping prin care se

recepționează apeluri HTTP de GET,POST, PUT, DELETE și se specifică forma utl-ului recepționat.

```
@PostMapping("/admin/add")
public ResponseEntity<ProductDto> addProduct(@RequestBody final CreateNewProductModel createNewProductModel){
    Product product = productService.addProduct(createNewProductModel);
    return new ResponseEntity<>(ProductDto.from(product), HttpStatus.OK);
}
```

Fig.4.3 – Implementarea unei metode din controller

De exemplu, endpoint-ul din figura 4.3 se accesează prin trimiterea unui apel la url-ul “http://localgost:8080/products/admin/add”. În antetul apelului se specifică în principal metoda http POST. Dacă în antet s-ar fi specificat o altă metodă, atunci apelul nu ar mai fi fost recepționat de către endpointul exemplificat.

Prin @RequestBody se prezintă formatul datelor care se așteaptă. Prin urmare, mesajele transmise dinspre partea de frontend vor trebui să se plieze pe intrările din backend. După primirea apelului și procesarea sa, se generează un răspuns transmis printr-un Dto.

Dto-urile sunt obiecte prin care se transferă informații între backend și frontend din motive de eficiență și securitate. Dacă se doresc informații din două modele de date diferite, se extrag doar informațiile necesare și se combină într-un dto. De exemplu, în pagina de filtrare se cer mai multe produse. În loc să se transmită tot modelul de date despre fiecare produs se generează un Dto specific care include strict datele de care este nevoie.

Autentificare și autorizare

Există situații în care doar anumiți utilizatori autentificați și autorizați pot accesa anumite metode din controllere. Acestea se specifică prin adnotarea @PreAtuthorize provenită din pachetul Spring Security. Înainte ca apelul să fie procesat, acesta trebuie să treacă prin modulul de securitate fiind bazat pe un jeton JWT (JSON Web Token).

Un jeton JWT este un standard de transmitere securizată a datelor între procese sub forma unui obiect JSON. Securitatea provine dintr-o semnătură unică criptată a fiecărui proces emițător, iar cu ajutorul ei se verifică integritatea datelor transmise.[6]

Un JWT are trei părți componente separate de un punct: antet, încărcătură și semnătură.

- Antetul conține tipul de jeton (în cazul de față JWT) și algoritmul folosit pentru criptare.

6 -https://jwt.io/introduction

- Încărcătura conține informații cu privire la mesajul transmis. Există câteva standarde opționale care recomandă folosirea anumitor informații precum subiectul mesajului, un timp de expirare etc.
- Semnătura generată cu un algoritm: se iau antetul și încărcătura în formă criptată, un secret emis de către back-end și se generează semnătura cu algoritmul HMAC-SHA256. Această semnătură verifică dacă încărcătura a fost modificată pe parcurs autenticitatea mesajului . [6]

Stratul de servicii este responsabil cu executarea logicii din interiorul aplicației. Acesta este stratul principal în care se prelucrează datele și în mod similar cu situația precedentă se prezintă sub forma unor clase care au o adnotare, iar în cazul de față se folosește adnotarea @Service. Un serviciu, interacționează de regulă cu stratul de date prin utilizarea adnotării @Autowired.

În secvența de cod din figura 4.4 se prezintă implementarea unui serviciu care se ocupă de logica de manipulare a produselor. Se observă utilizarea @Service asupra clasei și @Autowired care face legătura cu stratul de date. Metoda afișată primește un id, iar serviciul comunică cu stratul de date pentru obținerea datelor produsului. La acest nivel se tratează și erorile, iar în cazul de față avem inexistența produsului cu id-ul indicat.

6 -https://jwt.io/introduction

```java
@Service
public class ProductService {
    1 usage
    @Autowired
    FileStore fileStore;
    8 usages
    @Autowired
    ProductRepository productRepository;
    1 usage
    @Autowired
    BrandRepository brandRepository;
    1 usage
    @Autowired
    GenderRepository genderRepository;
    1 usage
    @Autowired
    CategoryRepository categoryRepository;
    1 usage
    @Autowired
    ProductSpecification productSpecification;
    no usages
    @Autowired
    ProductAttributeRepository productAttributeRepository;
    6 usages  sergiusergiu1234 *
    public Product getProduct(Long productId) {
        return productRepository.findById(productId).orElseThrow(()->
                new EntityNotFoundException("Product with id " + productId + " does not exist"));
    }
```

Fig. 4.4- Implementarea unui serviciu

Stratul de date este partea aplicației care comunică cu baza de date. Acesta cuprinde două tipuri de componente:

- Modele de date. Sunt clase java cu adnotarea @Entity (figura 4.5), ce definesc practic tabele din baza de date. Câmpurile și cheile tabelelor se exprimă prin @Id, @Column, iar relațiile dintre tabele se realizează cu ajutorul adnotărilor @OneToMany, @ManyToMany și @OneToOne (figura 4.6).

```java
@Entity
public class Product{
    no usages
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id", nullable = false)
    private Long id;
```

Fig. 4.5 – Definirea modelului de date pentru produs

```
@ManyToOne
@JoinColumn(name = "category_id",nullable = false)
private Category category;
```

Fig. 4.6 – Definirea relației One To Many dintre Product și Category

- Interfețe specializate în comunicarea cu baza de date sau în engleză "repository". Metodele din aceste interfețe sunt apelate de către stratul de servicii. Acestea au la bază o interfață din pachetul Spring Data JPA și generează query-urile pentru baza de date. Este necesară specificarea sub formă de parametrii a tipului entității filtrate și tipul de dată al cheii primare. În figura 4.7 se observă că nu s-au scris query-uri obișnuite, ci doar cod java. Acest lucru permite schimbarea cu ușurință a bazei de date iar aplicația generează query-uri specifice bazei de date alese.

```
public interface ProductRepository extends JpaRepository<Product,Long>, JpaSpecificationExecutor<Product> {
    sergiusergiu1234
    List<Product> findAll(Specification<Product> spec);
    sergiusergiu1234
    Page<Product>findAll(Specification<Product>spec, Pageable pageable);
    no usages  sergiusergiu1234
    Product findByName(String name);
    no usages  sergiusergiu1234
    Optional<Product> findByBrand(Brand brand);
    1 usage  sergiusergiu1234
    List<Product> findByBrandName(String brandName);
}
```

Fig.4.7- Implementarea unui repository pentru modelul de date Product

Pentru conectarea cu baza de date MySQL, se modifică fișierul "application.yml". Aici se introduc datele de conectare la baza de date. Cele mai importante de menționat sunt:

- "spring:jpadatabase-platform: org.hibernate.dialect.MySQLDialect" – prin aceasta setare se alege dialectul utilizat în baza de date pentru generarea query-urilor de către repository
- „spring:datasource:username:<username>" – se selectează username-ul pentru conectare
- "spring:datasource:url:jdbc:mysql://localhost:3306/<dbName>" - URL-ul pentru conectarea la baza de date
- "spring:datasource:password: <password>" – parola pentru conectare la conexiunea cu baza de date

4.4. Implementare frontend

În continuare se va discuta despre partea de frontend a proiectului reprezentată de aplicația interfeței. Aceasta a fost implementată în limbajul TypeScript și s-a folosit biblioteca React.

Pentru rularea unui proiect în React, trebuie instalat Node.js care este un program ce îi permite un calculator să ruleze JavaScript fără un web browser. De asemenea este necesar să avem și npm (Node Package Manager) folosit în instalarea de pachete JavaScript. După instalarea celor două unelte, se rulează comanda „npx create-react-app <nume_aplicație> --template typescript" care generează automat un proiect react funcțional care se rulează cu comanda "npm start".

Componente React

Elementul de bază al unei aplicații React este componenta. Aceasta este un cod care redă elemente din interfața implementată și include funcționalitatea lor. Se utilizează sintaxa JSX care îmbină JavaScript cu Html. Fiecare componentă este reutilizabilă, astfel că se pot combina mai multe componente între ele pentru a obține interfețe complexe. Astfel la proiectarea aplicației am descompus interfața în mai multe elemente, fiecare având propria funcționalitate.

React hooks

Biblioteca React le pune la dispoziție programatorilor conceptul de "hook". Acesta este o funcție special pe care o pot folosi componentele pentru menținerea și modificarea stărilor. Există mai multe tipuri de hook-uri, iar cele mai folosite sunt: useState și useEffect, useContext.

UseState permite creearea unei stări locale a unei. Este un concept similar cu conceptul de variabilă, doar că aici se ține minte starea unei componente.

Hook-ul are două părți, o variabilă x și o funcție care modifică acea variabilă f sub forma :

```
const [x,setX]=useState(1);    -> variabila x are valoarea "1"
setX(2);  "                    -> variabila x ia valoarea 2
```

UseEffect este un hook care permite executarea unei secvențe de cod atunci când se modifică o condiție. De cele mai multe ori s-a utilizat pentru extragerea unor date din partea de backend precum lista de produse, de favorite sau conținutul coșului de cumpărături la încărcarea paginilor.

“ useEffect(()=>{ *cod de executat*},[*condiție*]); “

Dacă condiția este nulă, atunci se execută doar la încărcarea componentei.

UseContext le permite mai multor componente să acceseze un set de valori la comun. Acest hook s-a folosit la funcționalitatea de autorizare, deoarece mai multe componente și pagini au nevoie de aceleași informații despre utilizatorul autentificat.

Fig. 4.8- Prezentare componente din interfață

În figura 4.8 am introdus o captură de ecran a paginii de filtrare a produselor, iar pentru exemplificarea structurii aplicației am încadrat fiecare componentă în dreptunghiuri de culori diferite. Fiecare pagină în sine este o component numită “părinte” iar celelalte înglobate se numesc “copii”.

În chenarul portocaliu avem o componentă prezentă pe fiecare pagină a aplicației și este folosită pentru navigare.

În chenarul verde este redată pagina curentă selectată în componenta de navigare. Aceasta are două componente copil, și anume o bară de filtrare în ghenarul roșu și mai multe cartonașe cu produse în chenarele maronii. Fiecare componentă își execută propria funcționalitate.

Dacă se fac schimbări într-o componentă părinte, atunci toate componentele copil se reîncarcă. Dacă în schimb se schimbă starea unui copil, componentele de grad egal sau mai mare sunt ne-afectate.

De exemplu, din această interfață se pot adăuga produse la lista de produse favorite prin apăsarea pictogramei în formă de inimă. Acest lucru duce la modificarea stării produsului prin modificarea culorii inimii, în timp ce restul componentelor rămân neafectate.

Comunicarea cu backend-ul

Fiecare utilizator este autentificat printr-un formular de autentificare. Trimiterea acestuia presupune trimiterea unei cereri http cu username-ul și parola, acestea sunt verificate prin modulului de securitate în backend, iar dacă verificarea are succes, se răspunde cu tokenul de acces JWT și rolul utilizatorului.

Prin utilizarea hook-ului useContext unde se stochează datele de autentificare și cu ajutorul token-ului JWT, ori se limitează accesul la anumite pagini sau funcționalități, ori anumite componente sunt modificare în funcție de rolul utilizatorului autentificat.

```
const toggleFavorite =()=>{
   //verify if authenticated
   if(auth.accessToken){
    const token = window.localStorage.getItem('accessToken')
    if(!favorited){
        //send server request
            fetch(`http://localhost:8080/favorites/add/${product.id}`,{
                method: 'POST',
                headers: {
                    'Authorization': `Bearer ${token}`
                }
            })
            .then(response => response.json())
            .then(data =>{setFavorited(true);
            })
    }else{
                    //send server request
                    fetch(`http://localhost:8080/favorites/delete/${product.id}`,{
                        method: 'DELETE',
                        headers: {
                            'Authorization': `Bearer ${token}`
                        }
                    })
                    .then(response => response.json())
                    .then(data =>{
                                        setFavorited(false);
                    })
    }
```

Fig.4.9- Modificarea stării de favorit a unui produs

În figura 4.9 se prezintă o funcție “toggleFavorite” care modifică starea de favorit a unui produs. În primă fază se verifică dacă este prezent un token JWT. Dacă este prezent un token în memorie, înseamnă că utilizatorul este logat, altfel este rugat să se autentifice.

În continuare se trimite o cerere http prin funcția fetch() către un endpoint din backend cu o metodă POST sau DELETE și tokenul în antet. Răspunsul primit se transformă în format JSON, iar apoi se se modifică starea produsului în mod corespunzător prin funcția setFavorited(). Motivul pentru care este nevoie de un token, este ca aplicația să știe cu ce utilizator să se asocieze produsul .

Plasarea comenzilor

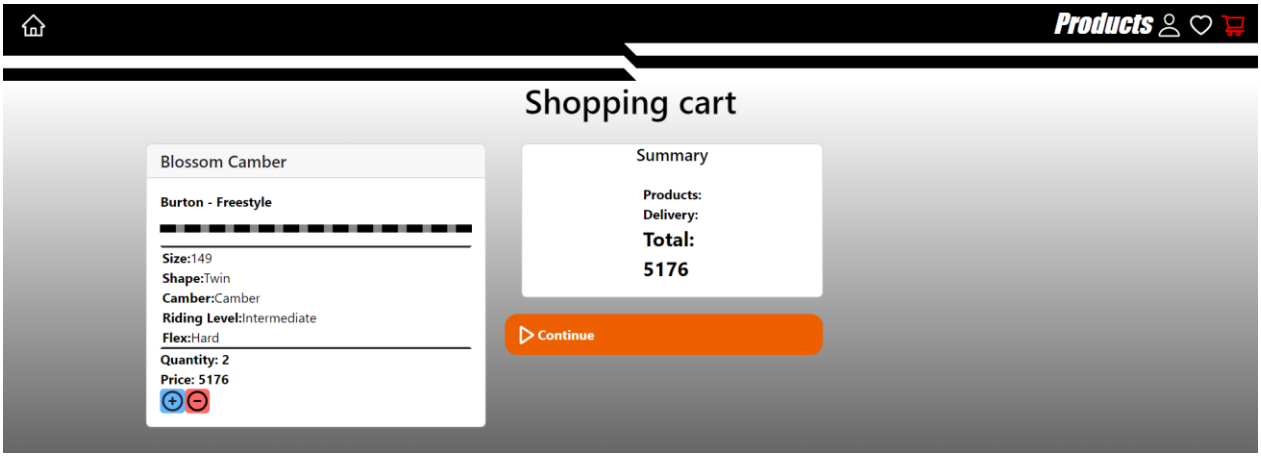

Fig.4.10- Interfața coșului de cumpărături

În figura 4.10 se prezintă interfața coșului de cumpărături. Prin apăsarea simbolurilor de la fiecare produs, se selectează cantitatea dorită și prețul total se modifică corespunzător. La apăsarea butonului de continuare se deschide o pagină unde se completează formularul de generare al unei comenzi, iar apoi se va trece la altă pagină de confirmare a comenzii finale și verificare înainte de plasarea comenzii. Toate aceste pagini comunică între ele prin hook-ul useContext, prin care se actualizează același obiect, al comenzii.

5. Concluzii

În concluzie, dezvoltarea acestui proiect a fost o experiență interesantă, din care am învățat o mulțime de lucruri noi.

Pe parcursul dezvoltării proiectului am realizat cât de important este ca planul aplicației să fie cât făcut cât mai detaliat înaintea începerii implementării practice. Deși am mai auzit acest lucru de nenumărate ori înainte, niciodată nu i-am dat importanță acestui îndemn, până acum. Informații generale din domeniul economic și bune practici în gestionarea unei astfel de aplicații, ar fi fost foarte bine venite, întrucât întocmirea planului ar fi fost mai eficientă. Am întâmpinat situații în care dezvoltarea aplicației a fost încetinită și regresată din cauza lipsei unui plan bine pus la punct.

Consider că pe parcurs, mi-am dezvoltat abilitatea de a căuta informații într-un mod mai eficient, reușind să asimilez destul de multă informație nouă, având în vedere că am fost nevoit să învăț framework-ul Spring Boot și librăria React întrucât nu știam mare lucru la început. Totuși, procesul a fost unul foarte interesant și captivant și sunt convins că, cunoștințele dobândite pe partea de programare îmi vor fi de folos și mai departe în cariera profesională.

Aplicația a fost concepută pentru a reprezenta un magazin online care include funcționalitățile specifice unui astfel de magazin, precum filtrarea de produse, adăugarea acestora într-un wishlist și într-un coș de cumpărături, pe baza căruia se plasează comenzi.

Îmbunătățiri posibile

În concluzie, pot spune că aplicația a ajuns într-un stadiu care cuprinde aproape toate funcționalitățile pe care mi le-am propus la început.

1. Mi-aș fi dorit ca aplicația să fie capabilă să execute plăți reale cu cardul, dar din cauza lipsei inițiale de informație cu privire la aspectele legislative ce țin de organizarea bazei de date și a lipsei de timp, nu am implementat acest lucru.

2. Pasul precent presupune publicarea aplicației pe un server public.

3. Dezvoltarea securității- estionarea mult mai complexă și mai eficientă a securității aplicației

4. Îmbunătățirea bazei de date pentru stocarea mai multor imagini pentru un singur produs. În momentul de față, un produs are câte o imagine, pe când într-un magazin profesionist fiecare produs are cel puțin 3 poze.

5. Dezvoltarea bazei de date în zona utilizatorilor, pentru a oferi posibilitatea de stocare a mai multor carduri sau adrese de livrare.

6. Modificarea interfeței adminstratorilor pentru un control mai bun asupra produselor

7. Implementarea autentificării în colaborare cu alte servicii precum Google și Facebook.

8. Modificarea designului interfeței

6. Bibliografie

1 - Alexandru Lelutiu, Perenitatea conceptelor promovate de bazele de date, Cluj-Napoca: Casa Cartii de stiinta, 2002.

2- C. J. Date, An Introduction to Database Systems (8th edition), Addison-Wesley, 2003.

3 -]Database Management Systems de Raghu Ramakrishnan și Johannes Gehrke

4 - DATABASE SYSTEMS The Complete Book Second Edition- pag.1

5David C. Hay, Data Model Patterns: A Metadata Map, Morgan Kaufmann, 2006.

6Essentials of Systems Analysis and Design-FIFTH EDITION -de Joseph S. Valacich, Joey F. George și Jeffrey A. Hoffer

7International Organization for Standardization. (2008) ISO/IEC 9075-1:2008 (SQL/Framework). Document.

8 - Joseph S. Valacich, Joey F. George, and Jeffrey A. Hoffer, Essentials of Systems Analysis and Design, Pearson, 5nd Ed., 2011.

9 - Lex de Haan and Toon Koppelaars, Applied Mathemathics for Database Professionals, Apress, 2007.

10Marius Muji, Baze de Date, curs pentru studenti, format electronic, 2017

11UML 2.0 in a Nutshell de Dan Pilone, Neil Pitman - Iunie 2005

12UML Distilled: A Brief Guide to the Standard Object Modeling Language" by Martin Fowler.

13William J. Lewis, Data Warehousing and E-Commerce, Prentice Hall PTR, 2001.

14https://creately.com/blog/diagrams/use-case-diagram-relationships/

15https://elogic.co/blog/ecommerce-architecture/

16https://jwt.io/introduction

17https://ro.wikipedia.org/wiki/Baz%C4%83_de_date

18https://www.baeldung.com/rest-with-spring-series

19https://www.insightsforprofessionals.com/it/storage/distributed-vs-centralized-database

20https://www.oracle.com/database/what-is-database/#link1

21https://www.section.io/engineering-education/spring-boot-amazon-s3/

22https://docs.aws.amazon.com/whitepapers/latest/serverless-multi-tier-architectures-api-gateway-lambda/three-tier-architecture-overview.html