**CAPITOLUL 1. INTRODUCERE**

**1.1 Situatia 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 cu ajutorul a câtorva clickuri sau atingeri de ecran.

Companiile 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 sau 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 ceilalti 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 aplicatiei sa fie intuitivă și usoară. Un site cu un aspect neplacut sau neingrijit le poate inspira utilizatorilor nesiguranță și neîncredere pierzându-se astfel posibili clienți în ciuda serviciilor oferite mai ales in cadrul unor firme mai putin cunoscute.

In concluzie o platformă online este o nevoie pentru aproape orice business, deoarece se economisesc bani și mult timp prin automatizarea unor procese, se elimina erorile umane din managementul contabilitatii și logisticii și le ofera posibililor clienti oportunitatea de a apela la serviciile firmei rapid și usor.

**1.2 Scopul lucrarii**

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

Printr-o interfață simplă si intuitive, clienții pot 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 cu un cont de administrator, au privilegiul de a modifica datele despre produse și de a actualiza stările comenzilor plasate de catre 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.

**1.3 Structura lucrării**

Lucrarea este împărțită in 4 capitole principale .

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

Capitolul 2 –se vor analiza cerintele 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 propriuzisă a aplicației si despre baze de date.

**CAPITOLUL 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 platfome de eCommerce și anume:

* Autentificare și autorizare pentru fiecare utilizator în parte (atât pentru clienți cât și pentru angajați).
* Efectuarea de operatii 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 iarna sunt foarte diversificate iar stocarea acestora trebuie sa reflecte realitatea cat 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 clickuri.
* Clientii trebuie să aibă posibilitatea de a salva produse într-o listă de favorite.
* Clienții să poată să inițializeze comenzi pe baza unui cos 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 functionalitățile sistemului. Se foloseste pentru a clarifica cerintele și nevoile impuse atunci cand acestea trebuiesc transmise altor persoane, intr-un mod usor de inteles, independent de tehnologia utilizată în implementarea aplicației.

Orice program software trebuie sa interactioneze cu niște elemente externe, acestea putând să fie oameni sau alte sisteme hardware sau software.

Programatorii, pot urmării aceste diagrame pe parcursul întregului proces de dezvoltare pentru a se asigura că sistemul este construit conform specificatiilor și cerintelor initiale.

O diagramă use case are cateva 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.

Shape

Description automatically generated with low confidence

Figura 2.1. Reprezentarea grafica 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 reprezinta printr-un oval in interiorul caruia se afla numele său. Acesta trebuie sa fie simplu și usor de inteles pentru oameni și să reprezinte o acțiune.

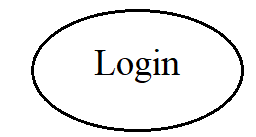


Figura 2.2. Reprezentarea grafică a unui use-case care descrie actiunea de logare

**Tipuri de relatii**

Intre use case-uri și actori se stabilesc relatii care sunt importante pentru descrierea modului in functioneaza sistemul și cerintele de implementare ale acestuia.

Exista cateva tipuri de relatii: asociere, dependenta și generalizare.

**Relatia de asociere** se poate defini intre use case-uri sau intre actori și use case-uri. Acosta sugereaza comunicarea intre componentele pe care le uneste și se poate reprezenta grafic printr-o linie.

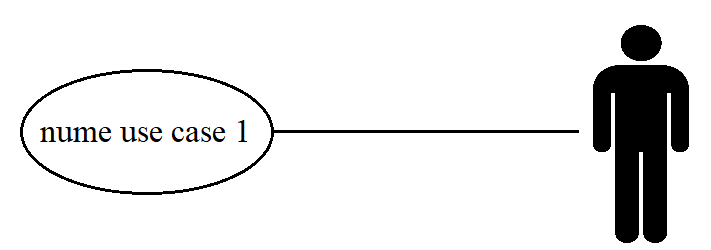


Figura 2.4. Reprezentarea grafica a unei relatii de asociere intre un use case și un actor

**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.

In 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 executa cazul A și cazul B se executa obligatoriu, iar cazul B este incomplet fără A.

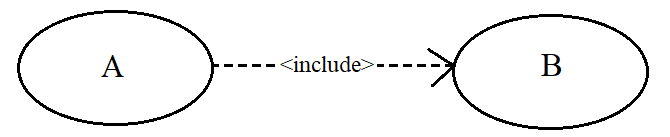


Figura 2.5. Depeneta de tip **include**

**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.

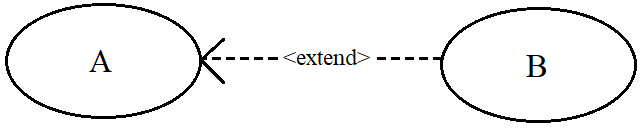


Figura 2.6. Dependenta de tip **extend**

**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.

A picture containing diagram, line, sketch

Description automatically generated

Reprezentare grafica a relatiei de generalizare

A picture containing screenshot, circle, black and white

Description automatically generated

Figura 2.7 – Diagrama Use Case pentru un magazin online

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 posibilitatea de a vizualiza și de a filtra produsele

-Existing User este orice utilizator care este prezent în baza de date . Acesta poate vizualiza produse și aplica filtre de căutare. Are posibilitatea de a se autentifica cu contul propriu pe baza căruia poate executa alte cateva actiuni: adăugare de produse intr-o lista de favorite și un cos de cumpărături și vizualizarea acestora. Pe baza conținutului din coșul de cumpărături poate plasa comenzi, ale căror stări pot fi urmărite.

- Guest User este orice utilizator care nu este înregistrat în baza de date. Acesta se poate înregistra și poate vizualiza catalogul de produse. Se evidențiază faptul că un simplu user vizitator are privilegii limitate în comparație cu un user prezent în baza de date.

-Admin – un admin este un actor care are alte privilegii fata de orice Customer. Acesta poate edita catalogul de produse și poate vizualiza datele de contact ale fiecărui user si comenzile acestuia. De asemenea are datoria de a edita starea comenzilor.

**2.8 - Diagrama entitiati si relatii (ERD)**

In acest stagiu al proiectării, se vor prezenta la nivel de concept entitățile aplicatiei si relațiile dintre acestea fără a se oferii detalii amănunțite, dar înainte trebuie identificate datele care trebuie stocate. O astfel de diagramă este utilă in ințelegerea legăturilor dintre elementele principale ale aplicației. Ulterior această diagramă se va dezvolta prin adăugarea de atribute, obținându-se schema detaliată a bazei de date.

"Database Management Systems" de Raghu Ramakrishnan și Johannes Gehrke p27,28

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

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

A picture containing font, text, screenshot, graphics

Description automatically generated

Fig 2.8 - Reprezentare grafică a unei entități

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

**Cardinalitatea** arată numărul maxim de asocieri care se pot realiza între instanțele a două entități. În functie 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.

A black line on a white background

Description automatically generated with low confidence

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

1. 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 Stundent si Clasă, exstă o cardinalitate one to many. O clasă are mai mulți studenți.

A black line on a white background

Description automatically generated with low confidenceFig 2.10- Reprezentare grafică a relației One-to-Many

1. 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 Film si Actor, un film poate avea mai mulți actori, in timp ce un actor poate juca in mai multe filme.

A black line on a white background

Description automatically generated with low confidence

Fig 2.11 – Reprezentare grafică a relației Many – to Many

Utilizând regulile amintite mai sus, se pot forma relatiile dintre entitățile sistemului informatic, care vor fi completate cu atribute si explicate in detaliu in faza de proiectare a bazei de date.

A diagram of a product

Description automatically generated with low confidence

Fig2.13 – Reprezentarea entității Produs si asocierile acesteia

A picture containing text, diagram, rectangle, line

Description automatically generated

Fig 2.12 – Reprezentarea entitatii user si asocierile acesteia

A diagram of a user

Description automatically generated with low confidence

Fig 2.14 – Reprezentarea entității Comandă si asocierea cu produsele si user-ii

**CAPITOLUL 3 – Proiectare**

**3.1 – 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 Database Management System ( prescurtat DBMS).

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

1. B.D. relationale – cel mai intâlnit tip de baze de
2. B.D. non-relationale – alt tip comun de baza de date
3. B.D. JSON
4. B.D. orientate pe obiecte

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

**3.1.1 – 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 usor de intretinut deoarece baza de date se află într-o singură locație fizică. Măsurile de securitate pot fi concentrate pe singurul punct de access, lucru care face ca aceasta să fie 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. Informatia redundantă este cea care este întâlinită în două sau mai multe locuri. Șansa de aparitie a acesteia este proportională cu complexitatea modelului datelor.

**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. În plus, dacă și aplicația software are probleme de performanță acestea limitarea bazei de date poate fi problema. În această situație se recomandă optimizarea aplicației sau adoptarea unei baze de date decentralizate.

Dacă există probleme legate de sistemul care detine 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.

**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 entitățtlor 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 process, entitățile devin niște tabele, iar atributele entităților devin capete de tabel. Legăturile dintre entități se reprezintă prin prezența unor diferite tipuri de chei care vor impune reguli și constrângeri.

**3.1.5 - Câmpurile unui 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.

La stabilirea câmpurilor se incearcă 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. În cartea “Database Management Systems” T. E. de Raghu Ramakrishnan și Johannes Gehrke p. 11 se prezintă un exemplu ca intr-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.

**3.1.6 -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. Supercheie – un atribut sau mai multe atribute, care combinate pot identifica unic, un rând dintr-un tabel.
2. 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.
3. 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.
4. 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).

**A picture containing text, diagram, parallel, plan

Description automatically generated**

Figura 3.1.7 – Diagrama bazei de date

**3.1.8 - Tabelele bazei de date:**

PRODUCT (product\_id, product\_name, description, price, brand\_id, gender\_id, category\_id, stock, image\_file\_name, image\_path)

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)

**3.1.9 - 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 din perspectiva tabelei **Product**.

Tabelele **Gender**, **Brand** si **Category** sunt tabele asemanatoare din perspectiva tabelului Product. Un brand/gender/categorie are mai multe produse, iar un produs are un singur brand/gender/categorie ( fapt care motiveaza prezența foreign key-urilor brand\_id, gender\_id, cartegory\_id, prin urmare am ales relatia One-to-Many.

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.

Fiecare Type are mai multe Atribute. In tabelul atribute se salveaza orice atribut care poate fi deținut de către articol comercializat, de exemplu: culoare, dimensiune, 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 foarte amănunțit.

De exemplu, in tabela Type putem avea: ‘Snowboard’, ‘Ochelari’, ‘Clăpari’, etc. Câteva atribute asociate cu type-ul Snowboard (prin cheia străină type\_id) pot fi dificultate, lungime, profil, elasticitate, tip de teren, etc. în timp ce atribute asociate cu type-ul Cască pot fi: mărime, duritate, culoare , etc.

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 + o valoare( 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 formează o cheie primară compusă).

**3.1.10 - Organizare useri și comenzi**

**Tabela User** conține informațiile de autentificare si autorizare ale userilor. Se observă prezența cheii role\_id, provenind din tabelul Role. Aici se specifică privilegiile utilizatorilor în aplicație (de administrator sau de client).

‘**Favorites’** și ‘**Shopping\_Car**t’ 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 – Dagrama Fluxului de Date (DFD)**

Diagrama flxului de date se foloseste pentru a reprezenta în mod grafic diferite concepte dintr-un sistem cu accentul pus asupra mișcărilor datelor.

În general aceasta este folosită cu scopul de a analiza un sistem informatic dar ajută și la comunicarea sau modelarea unor procese și concepte ce înglobează mai multe elemente fizice sau logice/teoretice. Se pot utiliza intr-o mare varietate de domenii, pornind de la procesarea unei comenzi într-un restaurant până la reprezentarea unor procese complicate din cadrul unei bănci. Sunt adesea folosite de către proiectanți și programatori în dezvoltarea de aplicații software, oferindu-le un support visual 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 dinstincte: flux de date, proces, deposit 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 la paginile 155, 156.

Un **flux de date** reprezintă date care se deplasează în interiorul sistemului între diferite locații. Acesta poate reprezenta atât informațiile unui formular completat de client la plasarea unei comenzi cât și informațiile despre un anumit produs, iar locațiile sunt niște procese, depozite de date sau surse externe. Acestea se reprezintă grafic printr-o săgeată unidirecțională pe care se scrie denumirea clară datei transmise . Fiecare flux de date trebuie să unească fie o sursă externă cu un proces, fie un proces cu o locație de stocare.

Un **proces** este o acțiune sau mai multe care se ocupă cu manipularea de date. De regulă aici se procesează datele primite și transmiterea acestora către o nouă locație. Acțiunile pot să fie atât automate, cât și manuale.

Într-un **depozit de date** sunt datele din sistem, care nu se află în mișcare. Aici se stochează datele din sistemul vizat, iar de cele mai multe ori este o bază de date sau o parte din aceasta.

O sursă/**entitate externă** poate reprezenta atât un operator uman, cât și un alt sistem extern cu care se comunică.

Este important de menționat faptul că o diagramă poate avea diferite grade de complexitate pe nivele 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.

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

A picture containing diagram, text, plan, technical drawing

Description automatically generated

Fig 3.2.1 – Diagrama Fluxurilor de Date – nivel 0

Procesul “Gestionare catalog\_produse” se poade descompune în mai multe procese, fiind una dintre cele mai complexe activități din sistem.

A diagram of a company

Description automatically generated with low confidence

Fig.3.2.2- Diagrama Fluxurilor de Date – gestionare produse

Dată fiind structura bazei de date, gestionarea catalogului de produse presupune existența mai multor procese care trebuie să se execute întro ordine specifică. Entitatea externă ADMIN transmite datele necesare către procesele “Gestionare categorii”, “Gestiune tipuri” și “Gestiune atribute\_posibile”,” Gestionare Brand-uri” care populează tabelele corespunzătoare din baza de date. Aceste procese verifică baza de date dacă nu sunt deja prezente în baza de date.

În diagramă se observă și câteva fluxuri de date care pornesc de la depozitele de date. Acestea sunt date de care procesele au nevoie pentru a-și îndeplini funcțiile. De exemplu, la adăugarea unei categorii noi sau a unui nou atribut\_posibil, este nevoie ca tipul de produs specificat să fie deja prezent în sistem. La fel se petrece și în cazul gestionării catalogului de produse: categoria și brand-ul trebuie să fie deja prezente în sistem, iar atributele trebuie să fie deja asociate cu tipul din care face categoria parte. Dacă vreuna dintre aceste verificări ejuează, atunci nu se mai adaugă nimic în baza de date.

A picture containing text, diagram, plan, technical drawing

Description automatically generated

Fig 3.2.3- Dagrama Fluxurilor de Date- gestionare comenzi

Clientul(USER) poate să filtreze produsele în funcție de niște criterii de filtrare prin procesul 1. Acesta mai poate adăuga produse în lista de produse favorite prin procesul 2.

Gestionarea comenzilor include mai multe etape. La inițierea comenzii(process 4) de către user 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. Comanda este verificată în funcție de stocurile curente (process 5), după care este generată și salvată în lista de comenzi(process 7), iar stocurile sunt actualizate(process 6). De aici, administratorul are datoria de a modifica starea fiecărei comenzi corespunzător(process 8).

**Capitolul 4 – implementare**

Aplicatia este constituita din 3 parti componente principale:

1. Frontend-ul : reprezinta interfața prin care utilizatorii (atat administratorii cat și clientii) iau contact cu aplicatia. S-a folosit React, o biblioteca JavaScript folosita pentru dezvoltarea de interfețe de utilizator și Typescript , care exinde sintaxa limbajului JavaScript, permitandu-se definirea tipurilor de date in mod static și nu dinamic precum in JavaScript-ul simplu. Framework-ul React a fost folosit pentru creearea unor aplicatii precum Facebook, Netflix, Instagram, Discord și multe altele.

2. Backend-ul: reprezinta “creierul” aplicatiei. In functie de datele primite de la partea de frontend, acesta introduce informatii in baza de date sau le extrage și le trimite sub forma de raspuns inapoi catre frontend. Am folosit Java Spring Boot. Acestea este un framework al limbajului Java care ofera suport pentru dezvoltarea de aplicatii web. Aici am definit modelele entitatiilor bazei de date și API-ul prin care se face comunicarea cu frontend-ul. De asemenea am implementat și o parte de autentificare și autorizare a utilizatorilor prin Spring Security care este tot un framework al limbajului Java.

3. Baza de date: S-a folosit o baza de date relationala MySQL, unde am salvat toate informatiile aplicatiei, pornind de la tipurile de produse la datele utilizatorilor.