**CAPITOLUL 1. INTRODUCERE**

**1.1 Situatia actuala și motivatia temei**

Fenomenul cel mai reprezentativ pentru societatea actuala este dezvoltarea accelerata a tehnologiei care a inceput acuma cateva zeci de ani, deoarece aceasta a devenit un element foarte important in viata fiecarui om. Astfel in ziua de astazi, dispozitivele electronice ne fac viata mai usoara și mai comoda avand posibilitatea de a accesa și trimite informatii cu ajutorul a catorva clickuri sau atingeri de ecran.

Cu timpul, companiile s-au adaptat treptat la acest stil de viata al oamenilor, devenind in cele din urma un standard ca acestea sa aiba o prezenta online intr-un fel sau altul, fie prin existenta unor website-uri pentru clienti prin care acestia pot lua contact cu firma sau se pot informa, fie prin exista unor platforme destinate angajatilor care le permite acestora sa gestioneze și sa automatizeze anumite elemente cheie din cardul business-ului. In general companiile sunt dispuse sa investeasca tot mai mult in dezvoltarea unor astfel de platforme pe masura, iar cele care aleg sa nu investeasca in domeniul online pierd o multime de beneficii fiind depasite de catre ceilalti competitori.

Un exemplu extrem de intalnit este prezenta unei aplicatii de e-commerce pentru magazine. Cu ajutorul unei astfel de aplicatii, se pot gestiona automat stocuri și vanzari, eliminandu-se erorile umane daca aplicatia este proiectata corect și respecta cerintele impuse. De altfel este important ca navigarea aplicatiei sa fie intuitivă și usoara. De asemenea un site cu un aspect neplacut sau neingrijit le poate inspira utilizatorilor nesiguranta și neincredere pierzandu-se astfel posibili clienti in 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 economiseste timp și bani, 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 urmareste dezvoltarea unei platforme online special concepute pentru management-ului unui magazin care comericalizeaza articole sportive pentru ski și snowboarding.ăîțșâșșșșțțțțâââââ

Clientii au la dispozitie o interfață intuitivă și prietenoasă prin care pot vizualiza și filtra produsele dupa mai multe criterii, pe care le pot salva ulterior intr-o lista de produse favorite sau intr-un cos de cumparaturi al carui continut se utilizeaza pentru generarea unei comenzie dorite, deci implicit este prezent și un sistem de autentificare.

Utilizatorii autentificati cu un cont de administrator, au privilegiul de a modifica datele despre produse si de a actualiza starile comenzilor plasate de catre client.

Prin intermediul acestei aplicatii software, se pot aduce imbunătățiri semnificative in gestionarea elementelor de logistică și in relația cu clientii, crescând satisfacțiaacestora și vâzările afacerii.

**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- Cerintele sistemului**

Aplicatia conceputa va trebui sa indeplineasca cateva cerinte minime pentru a-si atinge scopul. Programatorul trebuie sa urmeze aceste cerinte pentru ajunge la produsul final dorit.

La final aplicatia trebuie sa indeplineasca functiile de baza specifice oricarei platfome de eCommerce și anume:

* Autentificare și autorizare pentru fiecare utilizator in parte ( atat pentru clienti cat și pentru angajati).
* Efectuarea de operatii CRUD (Create Retrieve Update Delete) asupra datelor prin prezenta unor interfețe specializate.
* Stocarea articolelor sportive in baza de date trebuie sa fie cat mai detaliata. Echipamentele destinate sporturilor de iarna sunt foarte diversificate iar stocarea acestora trebuie sa reflecte realitatea cat mai bine.
* Avand in vedere modelul de date, trebuie implementata o functie de filtrare pe masura, cat mai amanuntita.
* Interfata clienților trebuie sa fie simpla și intuitiva. Trebuie sa se poata ajunge in orice punct al aplicatiei in cel mult 3 sau 4 clickuri.
* Clientii trebuie sa aiba posibilitatea de a salva produse intr-o lista de favorite.
* Clientii sa poata sa initializeze comenzi pe baza unui cos de cumparaturi.
* Clientii pot vizualiza starea curenta a comenzilor plasate și istoricul comenzilor

Cateva dintre aceste cerinte pot fi descrise cu ajutorul unor diagrame.

In continuare se va prezenta diagrama cazurilor de utilizare pentru un site de ecommece.

**2.2 - Diagrama cazurilor de utilizare (Use case)**

Orice program software trebuie sa interactioneze cu exteriorul fie cu alti oameni, fie cu alte sisteme.

Diagrama cazurilor de utilizare (use case diagram) este o diagrama UML, utilizata in faza de proiectare și analiza a unui sistem software pentru a defini functionalitatile sistemului și pentru a clarifica cerintele și nevoile impuse atunci cand acestea trebuiesc transmise altor personae, intr-un mod usor de inteles, independent de tehnologia utilizata in implementarea aplicatiei.

Programatorii, pot urmarii aceste diagrame pe parcursul intregului proces de dezvoltare pentru a se asigura ca sistemul este construit conform specificatiilor și cerintelor initiale.

O diagrama use case are cateva componente specifice: actori, use case-uri și relatiile dintre ele.

**Actorii**

Acestia reprezinta niste entitati externe care pot avea roluri diferite și care interactioneaza cu sistemul descris. Actorii pot fii niste simplii utilizatori, sisteme software sau dispozitive hardware și se reprezinta grafic printr-o persoana.

Shape

Description automatically generated with low confidence

Figura 2.1. Reprezentarea grafica a unui actor

**Use case-uri**

Acestea reprezinta actiunile pe care le poate executa un actor in cadrul sistemului. De obicei sunt evidentiate cele mai relevante actiuni, și nu se prezinta detalii cu privire la modul de executie sau la implementarea acestora

Din punct de vedere grafic, un use case se reprezinta printr-un oval in interiorul caruia se afla numele cazului de utilizare. Acesta trebuie sa fie simplu și usor de inteles pentru oameni și sa reprezinte o actiune.

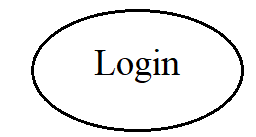


Figura 2.2. Reprezentarea grafică a unui use-case care descrie actiunea de logare

**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. Aceasta sugereaza comunicarea intre componentele pe care le uneste și se poate reprezenta grafic printr-o linie.

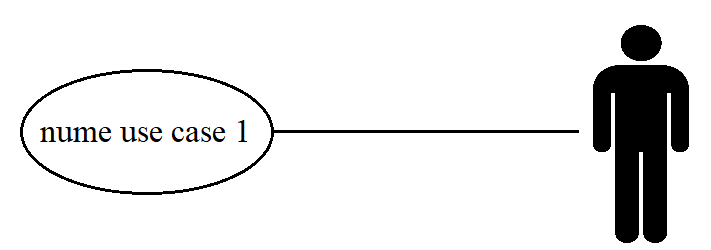


Figura 2.4. Reprezentarea grafica a unei relatii de asociere intre un use case și un actor

**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 folosita atunci cand executarea unui use case este conditionata de executia altui use case.

In exemplul urmator, Use Case-ul A include use Case-ul B și se reprezinta grafic printr-o sageata cu linie punctata pe care scrie “include”, iar sageata este indreptata 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 fara A.

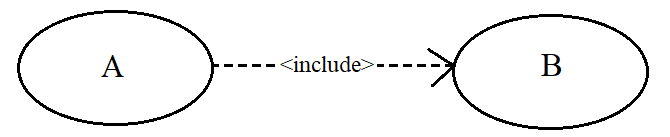


Figura 2.5. Depeneta de tip **include**

**Relaxia de extindere** este folosita pentru a-i adauga sistemului o noua functionalitate, pentru ca un Use Case il extinde pe celalalt. In exemplul urmator se prezinta faptul ca use case-ul B extinde use case-ul de baza A. A-ul trebuie sa aiba sens de unul singur, iar B-ul practic ii adauga ceva in plus lui A și este dependent de A.

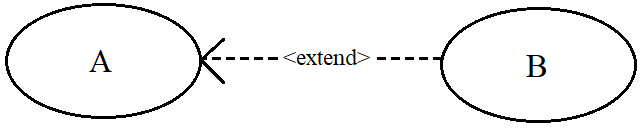


Figura 2.6. Dependenta de tip **extend**

**Relatia de generalizare** este similara relatiei de mostenire din programarea orientata pe obiect. Se arata ca un actor mosteneste rolul și use case-urile parintelui și se foloseste uneori cand se doreste evidentierea unor specializari sau generalizari. De exemplu actor “Customer” este mostenit de catre alti doi actori “Existing user” și “Guest user”.Customer-ul poate sa se uite la produse. Intre timp, mostenitorii sai au atat 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

In cele ce urmeaza, 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 in baza de date . Acesta poate vizualiza produse și aplica filtre de cautare. Are posibilitatea de a se autentifica cu contul propriu pe baza caruia poate executa alte cateva actiuni: adaugare de produse intr-o lista de favorite și un cos de cumparaturi și vizualizarea acestora. Pe baza continutului din cosul de cumparaturi poate plasa comenzi, ale caror stari poate fi urmarit.

- Guest User este orice utilizator care nu este inregistrat in baza de date. Acesta se poate inregistra și poate vizualiza catalogul de produse. Se evidentiaza faptul ca un simplu user vizitator are privilegii limitate in comparatie cu un user prezent in 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 fiecarui user si comenzile acestuia. De asemenea are datoria de a edita starea comenzilor.

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

In pas foarte important în dezvoltarea unei aplicații este determinarea datelor care trebuie stocate. Pentru aceasta este nevoie sa analizăm cerințele sistemului. După ce au fost creeate relatiile dintre entități, acestea trebuie analizate, idendificate posibilele probleme de proiectare si modificare a relatiilor.

In acest stagiu al proiectării, se vor prezenta la nivel de concept entitățile aplicatiei si relațiile dintre acestea fără a se oferii detalii amănunțite. 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 detaliata a bazei de date.

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

La fel ca si in cazul oricărui tip de diagram, exista cateva reguli generale pentru reprezentarea grafică a acestora.

**Entitățile s**unt obiecte care contin informații si sunt independente. Acestea 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 nedirectionale reprezentate prin linii simple. La contactul dintre linii si entitati se afiseaza **cardinalitatea**, folosindu-se doua simboluri grafice.

Cardinalitatea arată numărul maxim de asocieri pe care o entitate le poate avea cu altă entitate. 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 entitati. 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 colecție organizată de informatie, de obicei localizată pe un dispozitiv de stocare. In general o bază de date este controlată de un sistem de gestiune a bazelor de date (DBMS – database management system). Impreună, baza de date, DBMS-ul și aplicatiile asociate cu acestea formeaza un sistem informational cu baze de date.

Database System Concepts" de Abraham Silberschatz, Henry F. Korth și S. Sudarshan

In 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

**3.1.1 – Avantajele unei baze de date centralizate**

Datele se afla pe un singur dispozitiv. Orice informatie poate fi obtinuta, adaugata sau modificata rapid deoarece totul este intr-un singur loc.

Securitatea datelor este usor de intretinut deoarece toate datele sunt stocate intr-o singura locatie iar masurile de securitate pot fi concentrate pe singurul punct de acces. In cazul pierderii de date acestea pot fi recuperate daca exista o copie de rezerva actualizata periodic și stocata pe un alt dispozitiv.

Modelul datelor este simplu de proiectat in comparatie cu o baza de date distribuita. Datele sunt consistente datorita relatiilor stabilite intre tabele. In cazul unei baze de date distribuite, datele sunt implicit mult mai complexe și mai greu de manipulat.

Redundanță minimă- acest avantaj este o urmare a modelului simplu al datelor. Informatia redundanta este cea care este intalinita in doua sau mai multe locuri. Sansele de aparitie a acestora sunt proportionale cu complexitatea modelului datelor.

**3.1.3 Dezavantajele unei baze de date centralizate**

Limitari din cauza vitezei- daca sunt multi utilizatori care acceseaza acelasi set de informatii intr-un timp scurt, pot aparea probleme de performanta semnificative, fenomen care nu apare atat de des intr-o baza de date decentralizata.

Dacă exista probleme legate de sistemul care detine baza de date, aceasta poate fi inaccesibilă până la remedierea problemelor, lucru rezolvat in cadrul bazei de date distribuite: daca o locație “cade”, celelalte locații îi vor indeplinii funcția, iar serviciul corespunzator funcționează în continuare.

Riscul distrugerii informației- in cazul in care se intampla ceva cu sistemul fizic, există posibilitatea ca toate datele stocate sa fie pierdute pentru totdeauna dacă nu a fost facuta o copie de rezerva a datelor inaintea incidentului.

**3.1.4 - Proiectarea bazei de date**

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

După stabilirea entităților in diagrama ER generală si modificarea acesteia pentru rezolvarea unor posibile erori de proiectare putem transforma schema conceptuală în schema principală a bazei de date.

Fiecare entitate conceptuală se transformă într-un tabel, ale cărui capete sunt atributele entității si de asemnea, relațiile dintre ele sunt prezentate prin intermediul cheilor care impun anumite reguli si constrângeri.

**3.1.5 - Câmpurile unui tabel**

Un camp al unui tabel reprezintă o coloană din tabelul respectiv. Acesta trebuie să reprezinte un atribut sau o informație a entității pe care o descrie. 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. Acesta semnifică tipul de informațtie care se poate stoca. 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**

O cheie a unui tabel este unul/mai multe atribute care ajută la identificarea unică a unei instanțe a unei entități. 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 o combinatie de atribute care pot identifica in mod unic un rând dintr-un tabel. Prin urmare, orice cheie este o supercheie, dar nu si orice supercheie este o cheie.
2. Cheie Primară – un atribut utilizat pentru identificarea unică a instanțelor din cadrul unei entități. 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 poate 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 conține o cheie străină provenită din tabela Țări.

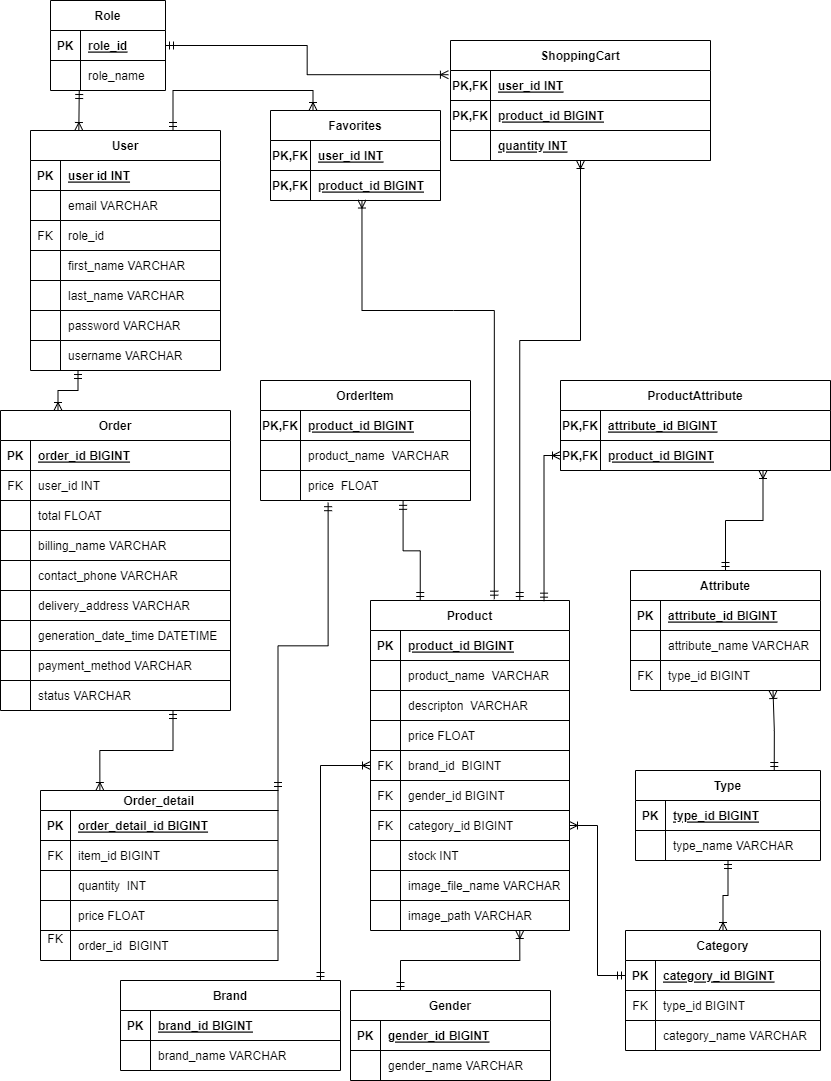
****

Figura 3.1.7 – Diagrama bazei de date

**3.3.3 - 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)

**3.3.4 - 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).

**3.3.5 - 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.

**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.