



Universitatea  
Transilvania  
din Brașov  
FACULTATEA DE MATEMATICĂ  
ȘI INFORMATICĂ

Programul de studii:  
Informatica

# Lucrare de licență

## Aplicație Web pentru căutarea produselor, cu sistem de notificare

Autor:

**Paul Matei**

Coordonator științific:

**Conf. univ. dr. Deaconu Adrian**

Brașov, 2019



# Cuprins

1. Introducere	5
2. Baza de date	6
2.1. Introducere	6
2.2. Clasificarea Bazelor de Date	7
2.2.1. Modelul de date relational	7
2.2.2. Modelul de date orientat obiect	7
2.2.3. Modelul de date obiect-relațional	7
2.2.4. Modelul de date ierarhic	7
2.3. Securitatea Bazelor de date	7
2.4. Modelul entitate-relație	8
2.5. Normalizarea bazelor de date	10
2.5.1. Forme Normale	10
2.5.2. Exemplu Normalizare	11
2.5.2. Diagrama Relațiilor dintre Entități(ERD – Entity Relationship Diagram)	14
2.6. SQL Server	14
2.7. Limbajul SQL	15
2.7.1. Tipuri de date în SQL	15
2.7.2. Interogări	16
2.7.3. Instrucțiuni de manipulare a datelor	16
2.7.4. Tranzacții	16
2.7.5. Limbajul de definire a datelor	17
3. Platforma .NET și Limbajul C#	18
3.1. .NET	18
3.1.1. Componentele .NET:	18
3.1.1.1. Common Type System(CTS)	18
3.1.1.2. Common Language Specification(CLS)	18
3.1.1.3 Common Language Runtime (CLR)	19
3.1.1.4. Common Intermediate Language (CIL)	19
3.1.1.5. Base Class Libraries (BCL)	20
3.1.2 Spații de nume	20
3.2. Prezentare Generală a limbajului C#	20
3.2.1. Tipuri de date	21
3.2.2. Declararea și inițializarea variabilelor	22
3.2.3. Tipuri referință și tipuri valoare	23
3.2.4. Tipul class	23
3.2.5. Tipul Struct	24
3.2.6. Tipul Interface	24
3.3. Programarea orientată-obiect	25
3.4. Arhitectura MVC.	26
3.4.1. Componente:	27
3.4.2. Avantaje:	27
3.4.3. Dezavantaje:	28
3.5. Entity Framework	28
3.5.1. Arhitectura Entity Framework	28
3.5.2. Modele de programare	29
3.5.3. EDM (Entity Data Model)	29
3.5.4 Servicii în Entity Framework.	30
3.5.4.1. Clasa <i>DbContext</i>	30



3.5.4.2. Clasa DbSet	32
3.5.6. Interogări în Entity Framework	32
4. Prezentarea aplicației	34
4.1 Procesatorul	35
4.1.1. Procesatorul de mesaje	35
4.1.2. Actualizarea datelor	44
4.1.3. Sistemul de notificare-alertă preț	48
4.1.4. Ustensile generale	51
4.2 Interfața utilizatorului.	53
4.2.1. Înregistrare	53
4.2.2. Autentificare	55
4.2.3. Căutarea Produselor	57
4.2.3.1. Inserția cautării în baza de date și în coada Azure	57
4.2.3.2 Căutarea produselor în baza de date	58
4.2.4. Setarea unei alerte	61
4.2.5. Produse Urmărite	63
4.2.6. Delogarea	63
5.Posibile extinderi și concluzii	64
Bibliografie	65



## 1. Introducere

Una dintre cele mai vechi forme de tranzacționare desfășurate online a fost procesarea tranzacțiilor online a IBM dezvoltată în anii 1960 și a permis prelucrarea tranzacțiilor financiare în timp real. Sistemul de rezervare a biletelor computerizat dezvoltat pentru American Airlines, numit Semi-Automatic Business Research Environment (SABRE), a fost una dintre aplicațiile sale. Aici, terminalele de calculatoare situate în diferite agenții de turism au fost conectate la un mare calculator mainframe IBM, care procesa simultan tranzacțiile și le coordona astfel încât toți agenții de turism să aibă acces la aceleași informații în același timp.

Apariția de cumpărături on-line, după cum știm astăzi, s-a dezvoltat odată cu apariția internetului. Inițial, această platformă funcționa doar ca un instrument publicitar pentru companii, oferind informații despre produsele sale. Acesta a avansat rapid de la acest utilitar simplu la tranzacția reală de cumpărături online datorită dezvoltării de pagini web interactive și de transmisii securizate. În mod specific, creșterea Internetului ca un canal de cumpărături securizată s-a dezvoltat începând cu anul 1994, cu primele vânzări ale albumului Sting "Ten Summoner's Tales". În curând au urmat vinuri, ciocolate și flori și s-au numărat printre categoriile de comerț de pionierat care au contribuit la creșterea cumpărăturilor online.

De multe ori ne gândim că avem nevoie de anumite lucruri, însă nu știm niciodată care este magazinul cu cea mai bună ofertă. În lucrarea de față mi-am propus dezvoltarea unei aplicații web ce permite utilizatorului să își caute produsele de interes pe diverse magazine on-line. Această aplicație se va ocupa atât de căutarea produsului dorit, cât și de notificarea utilizatorului în momentul în care acel produs a ajuns sub o anumită limită impusă de utilizator.

Motivul pentru care am ales această temă, a fost dorința de a realiza o aplicație web, utilizând un model arhitectural ce îmi permite generarea unei ierarhii simple și puternice în cadrul unei aplicații web, acest model fiind MVC(*model-vizualizare-controlor*, provenind din engleză *model-view-controller*). Această aplicație a fost pentru mine, începutul perfect spre un nou univers al programării.



## 2. Baza de date

### 2.1. Introducere

O bază de date este o colecție partajată de date elementare sau structurate, între care există relații logice, proiectată pentru a satisface nevoile informaționale ale unei organizații.

Baza de date este un depozit unic de date, care este definit o singură dată și este utilizat simultan de mai mulți utilizatori. În loc să existe fișiere separate cu date redundante.

Caracteristica principală a aplicațiilor de baze de date constă în faptul că accentul este pus pe operațiile de memorare și regăsire efectuate asupra unui volum mare de date și mai puțin asupra operațiilor de prelucrare a acestora. Principala operație care apare în aplicațiile de baze de date este regăsirea datelor în scopul obținerii de informații. O bază de date este creată pentru a fi interogată.

Alături de operația de regăsire, apar mai mult sau mai puțin frecvent operațiile de:

- memorare – pentru introducerea de noi informații în baza de date
- ștergere – pentru datele devenite inutile sau redundante
- actualizare – în cazul informațiilor deja existente în baza de date

Avantajele bazelor de date:

- Controlul centralizat al datelor, putând fi desemnată o persoană ca responsabil cu administrarea bazei de date.
- Viteză mare de regăsire și actualizare a informațiilor
- Sunt compacte: volumul ocupat de sistemele de baze de date este mult mai redus decât documentele scrise
- Flexibilitatea ce contă în posibilitatea modificării structurii bazei de date fără a fi necesară modificarea programelor de aplicație
- Redundanță scăzută a datelor memorate, care se obține prin partajarea datelor într-un mai mulți utilizatori și aplicații. În sistemele de baze de date, mai multe aplicații pot folosi date comune, memorate o singură dată.
- Posibilitatea introducerii standardelor privind modul de stocare a datelor, ceea ce permite interschimbarea datelor între organizații
- Menținerea integrității datelor prin politica de Securitate, prin gestionarea tranzacțiilor și prin refacerea datelor în caz de funcționare defectuoasă a diferitelor componente fizice.
- Sistemul de gestionare a bazelor de date oferă o vizualizare a datelor, care nu se modifică atunci când se schimbă suportul de memorare fizic, ceea ce asigură imunitatea structurii bazei de date și a aplicațiilor.



## **2.2. Clasificarea Bazelor de Date**

Majoritatea sistemelor de baze de date actuale sunt realizate în modelul de date relațional sau în modelul de date orientat obiect. Dezvoltarea continuă a acestor modele a condus către o nouă categorie de baze de date numite obiect-relaționale, care combină caracteristicile modelului relațional cu caracteristicile modelului orientat obiect.

### **2.2.1. Modelul de date relational**

Acest model se bazează pe noțiunea de relație din matematică, care corespunde unei entități de același tip și are o reprezentare ușor de înțeles și de manipulat, ce constă dintr-un tabel bidimensional, compus din linii și coloane. Fiecare linie din tabel reprezintă o entitate și este compusă din mulțimea valorilor atributelor entității respective, fiecare atribut corespunzând unei coloane a tabelului.

### **2.2.2. Modelul de date orientat obiect**

Este un concept unificator în știința calculatoarelor, fiind aplicabil în programare, în proiectarea hardware, a interfețelor, a bazelor de date etc. Sistemele de baze de date orientate obiect se bazează pe limbaje de programare orientate obiect cu capacități de persistență, în care datele sunt independente de timpul de viață al programelor care le creează sau accesează, prin memorare pe suport magnetic (disc)

### **2.2.3. Modelul de date obiect-relațional**

Acest model reprezintă extinderea modelului relațional cu caracteristici ale modelului obiect, extindere necesară pentru realizarea bazelor de date care definesc și prelucrează tipuri de date complexe.

### **2.2.4. Modelul de date ierarhic**

O bază de date se reprezintă printr-o structură ierarhică de înregistrări de date (records) conectate prin legături (links). Modelul ierarhic a fost primul model folosit pentru dezvoltarea bazelor de date

## **2.3. Securitatea Bazelor de date**

Referitor la protecția și securitatea datelor, în literatură de specialitate se definesc următoarele concepte de bază:

- *Securitatea datelor* – totalitatea măsurilor de protecție împotriva distrugerii accidentale sau intenționate, a modificărilor neautorizate sau a divulgării acestora



- *Caracterul secret* – este un concept ce se aplica la un individ sau organizație și consta în dreptul acestora de a decide ce informație se poate folosi și în ce condiții.
- *Confidențialitatea* – se aplica la date și se referă la statutul acordat, aceasta reprezentând nivelul, sau gradul de protecție ce trebuie acordat informației respective
- *Integritatea* – se referă la restricția ca sensul datelor să nu difere față de cel înscris pe documentul sursă, impunând totodată ca datele să nu fie alterate accidental sau voit.

## 2.4. Modelul entitate-relație

Modelul relațional este simplu, având la bază o solidă fundamentare teoretică fiind bazat pe teoria seturilor și pe logica matematică.

Cu ajutorul acestui model pot fi reprezentate toate tipurile de structuri de date de mare complexitate.

Modelul este caracterizat prin:

- structura de date;
- operatorii care acționează asupra structurii;
- restricții de integritate.

Conceptele care stau la baza definirii unei baze de date relaționale sunt:

- domeniul – un ansamblu de valori caracterizat prin nume. Acesta poate fi explicit sau implicit;
  - tabela (relația) – un subansamblu al produsului cartezian al mai multor domenii. Este caracterizat printr-un nume prin care se definesc atributele ce aparțin aceleiași clase de entități;
  - atributul – coloana unei tabeli, caracterizată prin nume;
  - tuplul – este reprezentat de linia dintr-o tabelă. Acesta nu are nume. Ordinea tuplurilor și a atributelor (coloanelor) dintr-o tabelă nu are nici o relevanță;
  - cheia – este un atribut sau un ansamblu de atribute ce identifică în mod unic un tuplu dintr-o tabelă;
  - schema tabeli – este formată din numele tabeli, de lista atributelor, iar pentru fiecare atribut este specificat domeniul asociat.

La baza modelului relațional se regăsesc o mulțime de operatori relaționali. Aceștia sunt operatori din algebra relațională și operatori de calcul relațional. În algebra relațională se regăsesc o colecție de operații formale aplicate asupra tabelilor (relațiilor), a fost concepută de E.F Codd. Operațiile sunt compuse din operatori și operanți. Sunt aplicate în expresii algebrice relaționale, care sunt cereri de regăsire.



Au fost introduși șase operatori de bază și doi operatori derivați.

Operatorii din algebra relațională pot fi grupați în două categorii :

1. Operatori pe submulțimi ( $R_1$ ,  $R_2$  și  $R_3$  relații (tabele)):

- reuniunea :  $R_1 = R_2 \cup R_3$ .  $R_1$  va conține tupluri unice din  $R_2$  sau  $R_3$ ;
- diferența :  $R_1 = R_2 \setminus R_3$ .  $R_1$  va conține tupluri ce se regăsesc în  $R_2$  și nu se regăsesc în  $R_3$ ;
- produsul cartezian :  $R_1 = R_2 \times R_3$ .  $R_1$  va fi construit din perechi  $x_1x_2$  unde  $x_1 \in R_2$  și  $x_2 \in R_3$ ;
- intersecția :  $R_1 = R_2 \cap R_3$ .  $R_1$  va conține tupluri ce se regăsesc atât în  $R_2$  cât și în  $R_3$ ;

2. Operatori relaționali speciali ( $R_1$ ,  $R_2$  și  $R_3$  relații (tabele)):

- selecția : din  $R_1$  se obține o subtabelă  $R_2$  care va conține toate tuplurile conținute de  $R_1$  care satisfac o anumită condiție. Numărul atributelor din  $R_1$  este egal cu numărul atributelor din  $R_2$ , numărul tuplurilor din  $R_2$  este mai mic decât numărul tuplurilor din  $R_1$ ;
- proiecția : din  $R_1$  se obține o subtabelă  $R_2$  care va conține toate tuplurile conținute de  $R_1$  fără duplicate. Numărul atributelor din  $R_2$  este mai mic decât numărul atributelor din  $R_1$ ;
- joncțiunea : este o derivație a produsului cartezian. Presupune utilizarea unui calificator care permite compararea valorilor unor atribute din  $R_2$  și  $R_3$ .  $R_2$  și  $R_3$  trebuie să conțină cel puțin un atribut comun care are valori comune.

Relațiile într-o bază de date pot fi:

- (1 : 1) – fie  $T_1$  și  $T_2$  două tabele. Spunem că avem relație 1 : 1 între tabela  $T_1$  și  $T_2$  dacă unui tuplu din tabela  $T_1$  îi corespunde un singur tuplu din tabela  $T_2$ . Exemplu: o persoană are un singur act de identitate.
- (1 : n) – fie  $T_1$  și  $T_2$  două tabele. Spunem că avem relație 1 : n între  $T_1$  și  $T_2$  dacă unui tuplu din tabela  $T_1$  îi corespund mai multe tupluri din tabela  $T_2$ . Exemplu: o cautare generează mai multe produse
- (n : n) – fie  $T_1$  și  $T_2$  două tabele. Spunem că avem relație n : n între tabela  $T_1$  și  $T_2$  dacă unui tuplu din tabela  $T_1$  îi corespund mai multe tupluri din tabela  $T_2$  și unui tuplu din tabela  $T_2$  îi corespund mai multe tupluri din tabela  $T_1$ . Pentru crearea acestui tip de relație este necesară introducerea unei tabele  $T_3$ , în care se specifică cu ajutorul cheilor din  $T_1$  și  $T_2$  relațiile dintre acestea. Exemplu: O cautare poate genera mai multe produse și un produs poate apărea în mai multe cautări.





## 2.5. Normalizarea bazelor de date

Normalizarea este o teorie construită pe baza conceptului de forme normale (FN), este un proces formal de analiză a relațiilor bazate pe chei. Normalizarea presupune un set de reguli care pot fi urmate până când baza de date poate ajunge la un anumit grad de normalizare. În cazul în care o cerință nu este satisfăcută, relația trebuie discompusă în mai multe relații păstrând informațiile (atributele) de legătură, care individual satisfac cerințele formei normale.

### 2.5.1. Forme Normale

**FN1.** O tabelă se află în formă normală 1 dacă toate atributele ei conțin valori elementare (atomice, nedecompozabile). Fiecare tuplu nu trebuie să aibă date la nivel de grup sau repetitive. În cazul în care există astfel de informații, structurile trebuie discompuse în tabele cu atribute atomice. O tabelă ce se află în FN1 este predispusă la anomalii de actualizare, datorită eventualelor dependențe funcționale incomplete.

**FN2.** O tabelă este în formă normală 2 dacă și numai dacă este în FN1 și fiecare atribut care nu este cheie a tabelii este dependent funcțional complet de cheie. Un atribut B al unei tabeli depinde funcțional de atributul A al aceleiași tabeli, dacă fiecărei valori a lui A îi corespunde o singură valoare a lui B, care îi este asociată în tabelă. Un atribut este dependent funcțional complet de un ansamblu de atribute A în cadrul aceluiași tabel, dacă B este dependent funcțional de întreg ansamblul A (nu numai de un atribut din ansamblu). Forma normală doi trebuie verificată doar la relațiile care au cheie compusă pe poziție de cheie primară. Relațiile la care cheia primară se compune dintr-un singul atribut, este în 2NF. O tabelă în FN2 reprezintă o anomalie la actualizare, datorită eventualelor dependențe tranzitive. Eliminarea dependențelor incomplete se face prin discompunerea tabelii în două tabele, ambele vor conține atributul intermediar B.

**FN3.** O tabelă este în formă normală 3 dacă și numai dacă este în FN2 și fiecare atribut care nu este cheie depinde în mod netranzitiv de cheia tabelii. Prin alte cuvinte, fiecare atribut al tabelii trebuie să depindă numai și numai de cheia primară a tabelii. Într-o tabelă T fie A, B și C trei atribute unde A este cheie primară. Dacă B depinde de A ( $A \twoheadrightarrow B$ ) și C depinde de B ( $B \twoheadrightarrow C$ ) atunci C depinde de A în mod tranzitiv. Eliminarea dependențelor incomplete se face prin discompunerea tabelii în două tabele, ambele vor conține atributul intermediar B.

**FN4.** O tabelă este în formă normală 4 dacă și numai dacă este în FN3 și nu conține două sau mai multe dependențe multivaloare. Într-o tabelă T, fie A, B și C atribute. În tabela T se menține dependența multivaloare A dacă și numai dacă mulțimea valorilor



lui B ce corespunde unei chei de date (A,C), depinde numai de o valoare a lui A și este independentă de valorile lui C.

**FN5.** O tabelă este în forma normal 5 dacă și numai dacă este în FN4 și fiecare dependența joncțiune este generată printr-un candidat la cheie al tabelului. În 67 tabela T (A, B, C) se numește dependența joncțiune (AB, AC) dacă și numai dacă T menține dependența multivaloare  $A \diamond B$  sau C.


O bază de date se consideră normalizată dacă este cel puțin în FN3.

### 2.5.2. Exemplu Normalizare

Pentru exemplificare procesului de normalizare și implicând relațiile dintre tabele și formele normale în proiectarea unei baze de date, vom lua ca exemplu baza de date pentru aplicația scurt prezentată în introducere.

Avem astfel următoarele entități:


**Produs:** reprezintă un produs din mediul on-line;

Produs (dbo)	
	Id
	Url
	Denumire
	Pret
	Stock
	Url_Image
	Id_Vanzator
	SKU
	EAN
	Data_Creat
	Cod_Denumire_Produs
	Sters

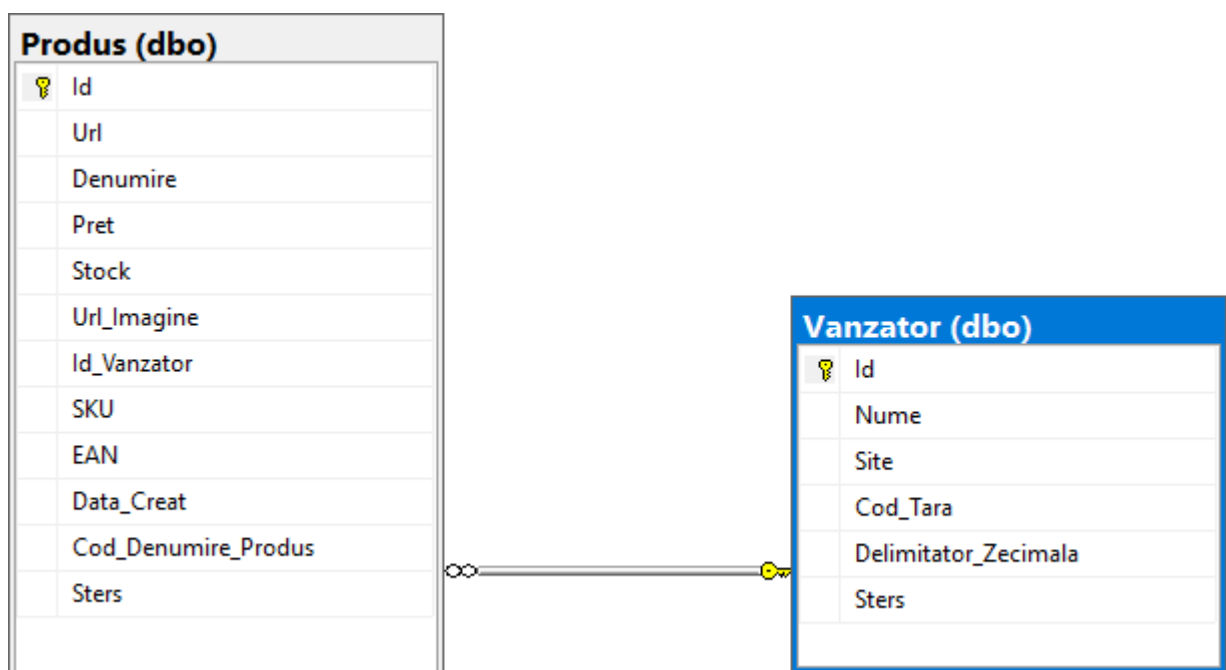
Dat fiind că această aplicație are la bază o multitudine de produse, această entitate are câte o relație cu fiecare din celelalte entități



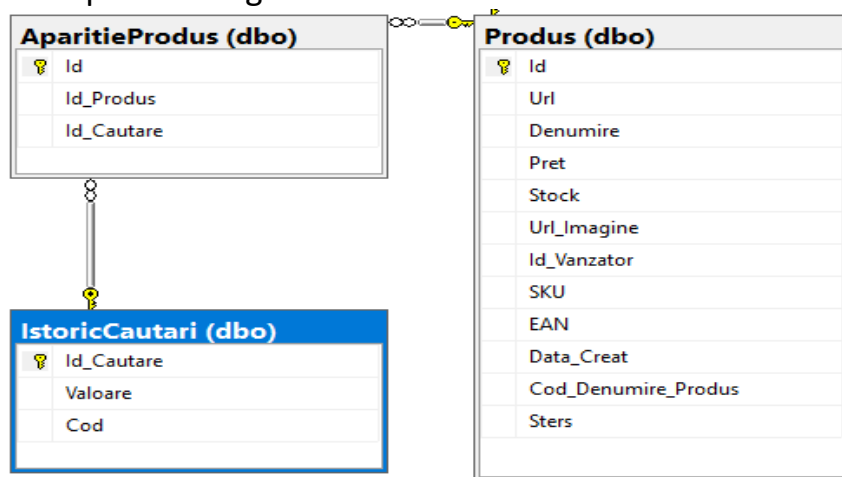
**Vânzător:** reprezentarea in baza de date a unui magazin on-line;

Vanzator (dbo)	
	Id
	Nume
	Site
	Cod_Tara
	Delimitator_Zecimala
	Sters

Relație Vânzător – Produs, Relație 1-N

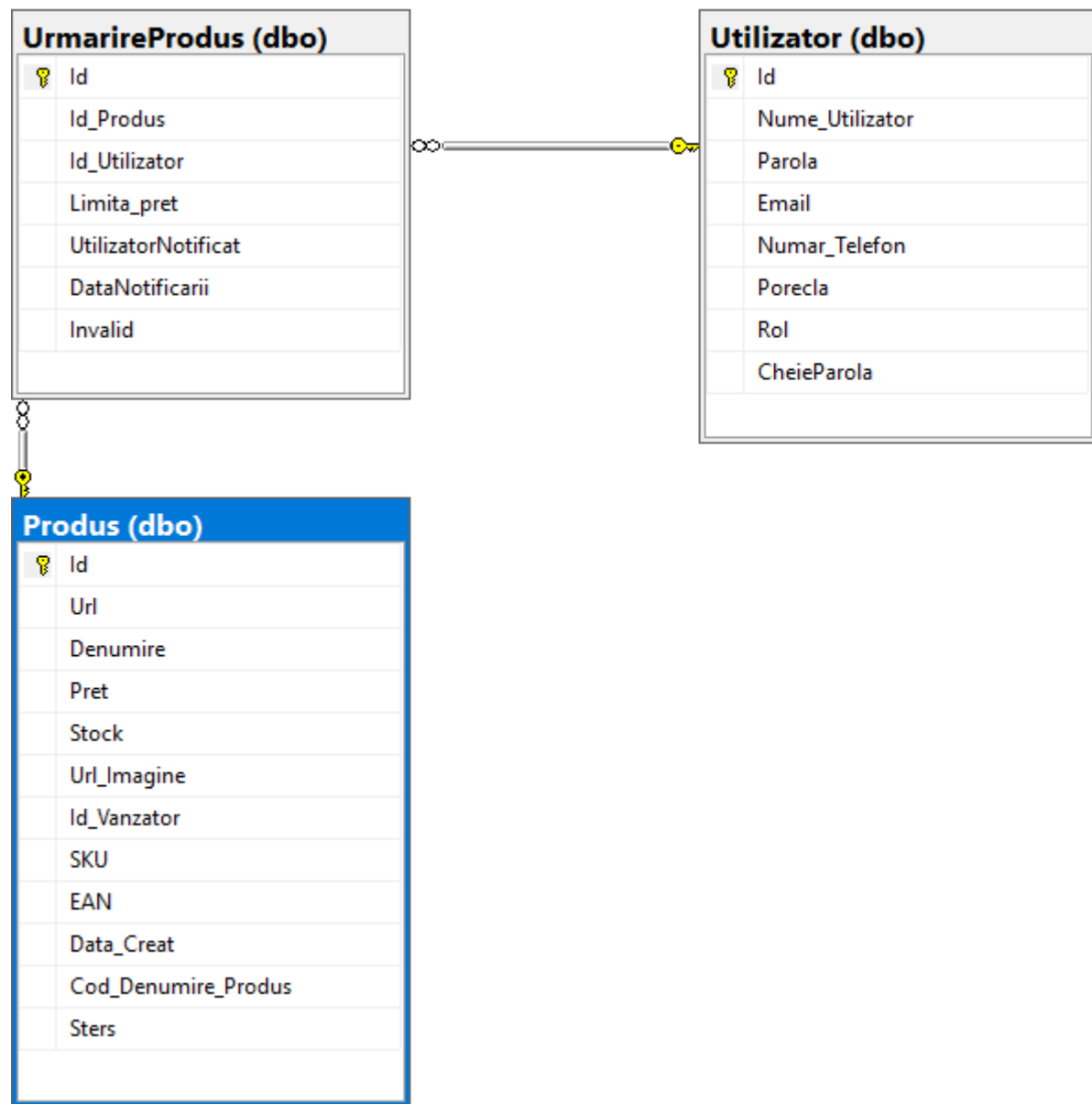


**Istoric Căutări :** Aceasta entitate reține toate căutările ce au fost efectuate și implicit toate produsele generate de acele căutări



În acest caz putem observa că toate atributele depind de cheia primară a fiecărei entități, astfel fiind în FN2. În continuare, observăm că niciun atribut nu este tranzitiv și examinând tabelele de mai sus, observăm că nu avem nicio dependență, însemnând că aceste tabele sunt în FN3.

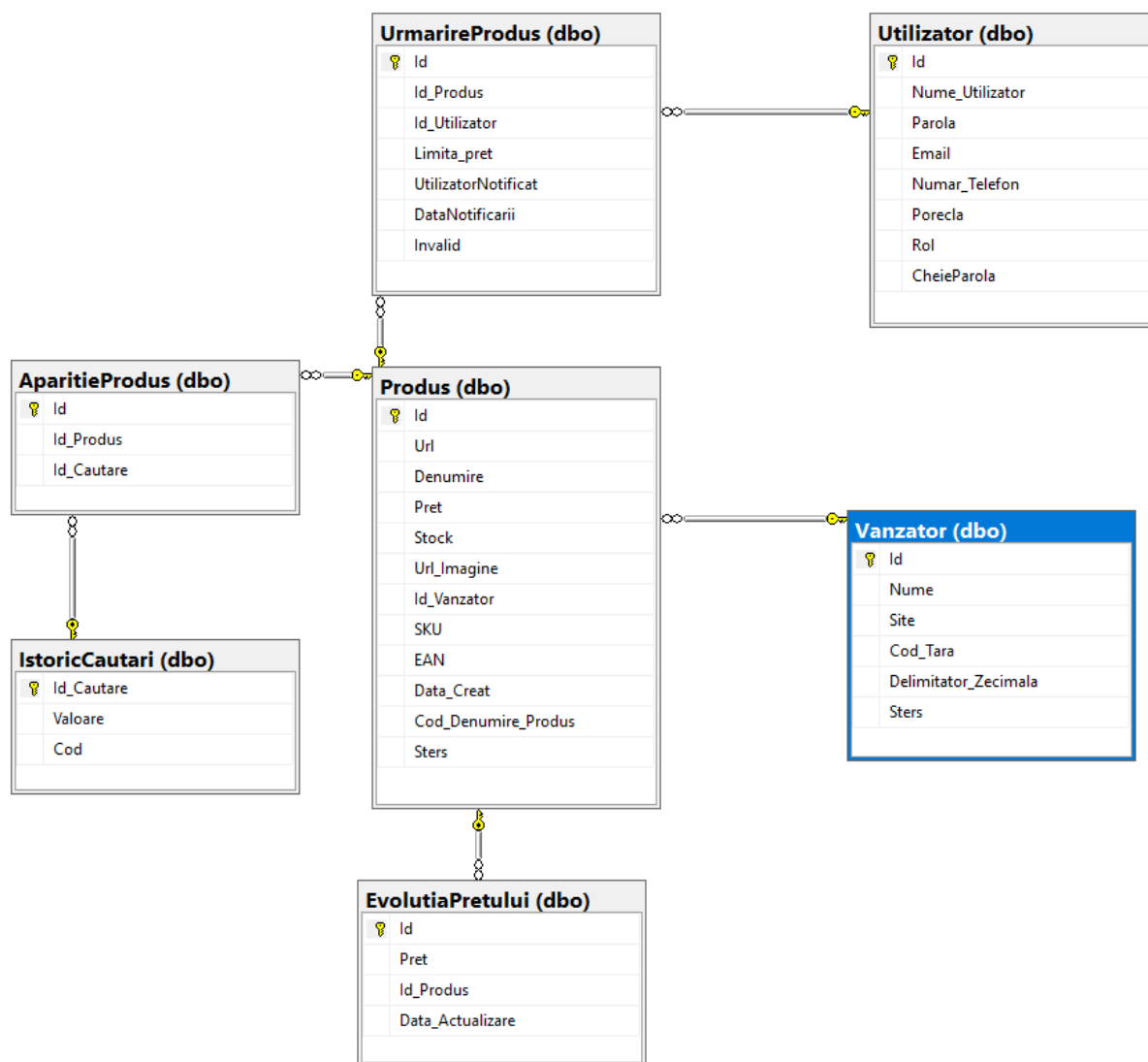
**Utilizator:** această entitate este reprezentarea din baza de date a unui utilizator al aplicației.



În acest caz putem observa că toate atributele depind de cheia primară a fiecărei entități, astfel fiind în FN2. În continuare, observăm că niciun atribut nu este tranzitiv și examinând tabelele de mai sus, observăm că nu avem nicio dependență, însemnând că aceste tabele sunt în FN3.



## 2.5.2. Diagrama Relațiilor dintre Entități(ERD – Entity Relationship Diagram)



Conform relațiilor prezentate în capitolul de mai sus și ținând cont de faptul că această bază de date a fost creată cu ajutorul ustensilelor din *EntityFramework* putem spune că această bază de date este în o formă normală acceptată.

## 2.6. SQL Server

Microsoft SQL Server este un sistem de gestionarea de baze de date relaționale produs de compania Microsoft. Are ca limbaj de interogare SQL (Structured Query Language). Acesta fiind cel mai răspândit limbaj de baze de date. Se poate aplica bazelor de date de dimensiuni foarte mari.



Toate operațiile care pot fi invocate în SQL Server sunt comunicate către acesta utilizând un format numit Tabular Data Strama (TDS). TDS este un protocol utilizat pentru transferul datelor între server și client. Inițial a fost proiectat și dezvoltat de Sybase Inc pentru Sybase SQL Server. Mai târziu este preluat de Microsoft în Microsoft SQL Server.

SQL Server suportă diferite tipuri de date. De la tipuri primitive precum: Integer, Float, Decimal, Char, Varchar, Text. Până la tipuri definite de utilizator (UDTs). Pe lângă date, o bază de date SQL server poate conține proceduri stocate, indexi, constrângeri. O bază de date SQL Server poate conține maxim  $2^{31}$  obiecte și poate fi compusă din mai multe fișiere la nivel SO. Fișiere ce pot avea dimensiunea maxima de un exabyte. Acestea au extensia .mdf

## 2.7. Limbajul SQL

**SQL** este un limbaj de programare specific pentru manipularea datelor în sistemele de manipulare a bazelor de date relaționale, iar la origine este un limbaj bazat pe algebra relațională. Acesta are ca scop inserarea datelor, interogații, actualizare și ștergere, modificarea și crearea schemelor, precum și controlul accesului la date. A devenit un standard în domeniu, fiind cel mai popular limbaj utilizat pentru crearea, modificarea, regăsirea și manipularea datelor de către SGBD-urile (Sistemele de Gestiune a Bazelor de Date) relaționale.

### 1.7.1. Tipuri de date în SQL

Oricărei coloane (sau câmp) dintr-un tabel SQL îi este asignat un *tip de dată*, la fel ca în toate celelalte limbaje de programare. Tipurile de date sunt următoarele:

- **CHARACTER** (sau **CHAR**) - șir de caractere
- **INTEGER** (sau **SMALLINT**) - număr întreg
- **FLOAT**, **REAL** sau **DOUBLE PRECISION** - număr real
- **NUMERIC**(precision, scale) sau **DECIMAL**(precision, scale) - număr zecimal , unde "precision" înseamnă numărul de cifre din partea întreagă, "scale" înseamnă numărul de zecimale.
- **DATE** - data zilei.
- **TIME** - ora.

Funcția sistem **NOW** întoarce data și ora curentă.



### 2.7.2. Interogări

Cea mai des utilizată instrucțiune în SQL este instrucțiunea **SELECT**.

SELECT [ALL | DISTINCT] coloana1 [,coloana2]

[INTO fișier]

FROM tabel1 [,tabel2]

[WHERE condiție] [ AND | OR condiție...]

[GROUP BY listă-coloane]

[HAVING condiții]

[ORDER BY listă-coloane [ASC | DESC] ]

- **Clauza INTO** este utilizată pentru a transfera rezultatul interogării într-o nouă tabelă; valabil în Microsoft Access, dar nu în toate platformele SQL.
- **Clauza WHERE** este utilizată pentru a specifica condiții trebuie să îndeplinească coloanele din care se face selecția. Această condiție este o expresie logică obținută prin aplicarea operatorilor conjuncție (AND) și disjuncție (OR) asupra unor expresii logice elementare obținute cu ajutorul operatorilor: = (egal), <> (diferit), < (mai mic decât), <= (mai mic sau egal decât), > (mai mare decât), >= (mai mare sau egal decât), **LIKE** (permite selectarea potrivirilor parțiale cu ajutorul operatorului %);
- **Clauza GROUP BY** permite gruparea coloanelor multiple în scopul prelucrării acestora prin **funcțiile agregate**: AVG - media aritmetică; COUNT - numărul articolelor; MAX - maximul; MIN - minimul; SUM - suma.
- **Clauza HAVING** Spre deosebire de clauza WHERE, acționează asupra rândurilor rezultate din clauza GROUP BY , aplicându-le condiția, spre a fi ulterior prelucrare prin funcțiile agregate.
- **Clauza ORDER BY** Ordonează rezultatele interogării în ordine alfabetică după unul sau mai multe câmpuri. ASC înseamnă în ordine crescătoare, iar DESC - ordine descrescătoare. Ordinea implicită este crescătoare.
- **Alte cuvinte cheie**: **ALL** - Toate articolele; **DISTINCT** - Numai articolele unice, fără duplicate.

### 2.7.3. Instrucțiuni de manipulare a datelor

- **INSERT** - inserează un articol într-o tabelă:
- **UPDATE** - actualizează un set de articole:
- **DELETE** - șterge un set de articole:

### 2.7.4. Tranzacții

Tranzacțiile sunt utilizate pentru a stabili în ce condiții se va desfășura un anumit set de instrucțiuni de manipulare a datelor.

- **START TRANSACTION** (sau **BEGIN WORK**, **BEGIN TRANSACTION**, în funcție de dialectul SQL) Început de tranzacție.



- **SAVE TRANSACTION** (sau **SAVEPOINT**) salvează starea bazei într-un punct al tranzației
  - **COMMIT** Operează toate operațiile tranzației ca fiind permanente.
  - **ROLLBACK** Anulează toate operațiile tranzației începând cu ultimul COMMIT.
- Instrucțiunile **COMMIT** și **ROLLBACK** termină tranzația curentă și deblochează datele.

### 2.7.5. Limbajul de definire a datelor

- **CREATE TABLE** creează un tabel în mod linie de comandă:

```
CREATE TABLE tabel(
    câmp1 tip1,
    câmp2 tip2,
    ...
    PRIMARY KEY (index1, index2, ...)
);
```

- **ALTER TABLE** modifică structura unui tabel existent prin edenumirea/adăugarea/ștergerea/schimbarea structurii unei coloane sau index:

#### Redenumirea unui tabel

```
ALTER TABLE tabel RENAME TO nume_nou_tabel;
```

#### Adăugarea de câmpuri noi

```
ALTER TABLE table_name ADD ( câmp1 def1, col2 def2, ... );
```

#### Modificarea structurii unui câmp

```
ALTER TABLE table_name MODIFY (câmp1 tip1, câmp2 tip2, ... );
```

#### Ștergerea unui câmp

```
ALTER TABLE tabel DROP COLUMN câmp;
```

#### Redenumirea unui câmp

```
ALTER TABLE tabel RENAME COLUMN nume_vechi TO nume_nou;
```

- **TRUNCATE TABLE** – Șterge toate articolele unui tabel:  
TRUNCATE TABLE tabel;

- **DROP TABLE** – Șterge tabelul:  
DROP TABLE tabel;





## **3.Platforma .NET și Limbajul C#**

### **3.1 .NET**

Până în anul 2002 când a apărut platforma .NET și limbajul de programare C#, programatorii care dezvoltau aplicații pentru sistemul de operare Windows erau nevoiți să folosească COM (Component Object Modeling). COM permitea crearea librăriilor ce puteau fi utilizate în diverse limbaje de programare.

În prezent majoritatea aplicațiilor pentru Windows nu mai sunt create după modelul COM, fie acestea aplicații desktop, site-uri web, librării, toate acestea sunt create utilizând platforma .NET. Trebuie să menționăm faptul că platforma .NET oferă posibilitatea găzduirii aplicațiilor nu doar pentru Windows, ci și pentru alte sisteme de operare, cum ar fi Mac OS și o multime de distribuții Unix/Linux.

Principalele beneficia ale platformei .NET:

- Interoperabilitatea cu aplicațiile existente
- suportă mai multe limbaje – C#, Visual Basic, F#
- deține o multitudine de librării ce sar in ajutorul dezvoltatorului în procesul de creare a aplicațiilor (de la aplicații grafice până la aplicații web)
- un mod de lansare mult mai simplificat.

#### **3.1.1. Componentele .NET:**

##### **3.1.1.1. Common Type System(CTS)**

În .NET tipul se poate referi la unul din următorii termeni:

- class
- interface
- structure
- enumeration
- delegate

CTS este responsabil cu modul ă n care timpurile sunt definite astfel încât acestea să poată fi înțelese de către CLR.

##### **3.1.1.2. Common Language Specification(CLS)**



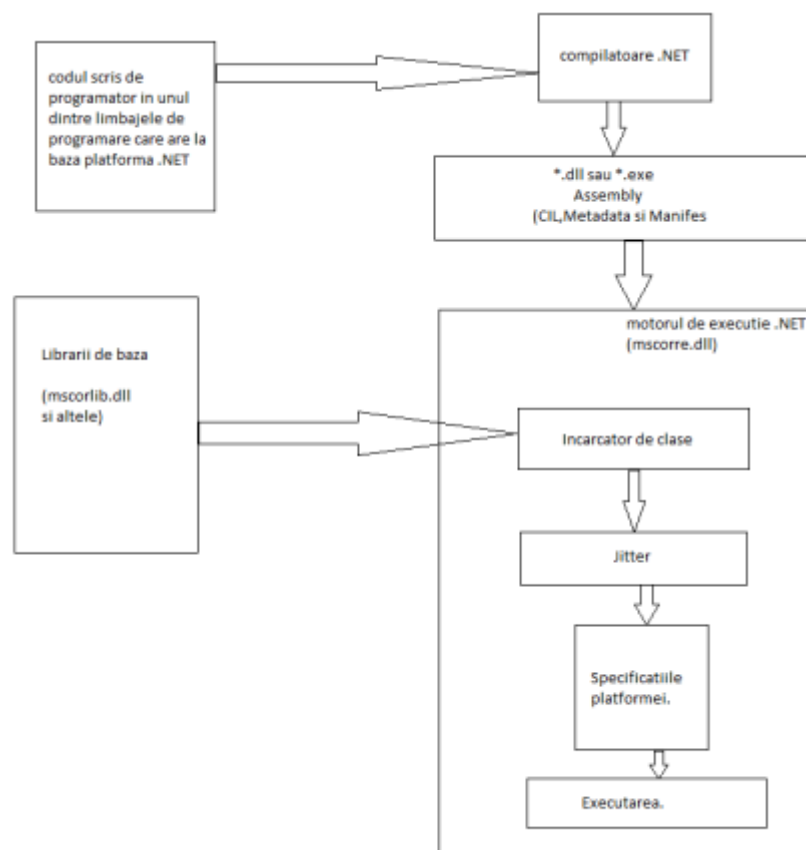
În platforma .NET fiecare limbaj de programare are propria sintaxă, total diferită de la un limbaj la altul.

CLS-ul este o multitudine de reguli necesare CLR-ului să ruleze codul trimis. Aceste reguli sunt implementate la nivelul compilatoarelor, și trebuie respectate cu riguritate.

### 3.1.1.3 Common Language Runtime (CLR)

Este o colecție de servicii de care este nevoie pentru a rula un cod compilat. CLR-ul este un sistem de rulare având ca proprietate un “layer” ce poate fi folosit de toate limbajele de programare care au la baza .NET.

Componenta cea mai importantă al CLR-ului este reprezentă fizic de biblioteca mscorlib.dll (cunoscută și sub numele de Common Object Runtime Execution Engine). Când o aplicație pornește, mscorlib.dll este încărcat automat.



### 3.1.1.4. Common Intermediate Language (CIL)

Atunci când construim un assembly folosind un limbaj gestionat (C#, VB, F# etc) compilatorul asociat transformă codul sursă în termeni CIL. La fel ca orice limbaj de



programare, CIL oferă numeroase structuri și tokeni. Dat fiind faptul că CIL este un alt limbaj de programare .NET, nu este o surpriză faptul că un assembly poate fi creat direct în CIL.

#### **3.1.1.5. Base Class Libraries (BCL)**

Platforma .NET oferă un set de biblioteci de bază care este disponibil pentru toate limbajele de programare .NET. Pe lângă faptul că aceste biblioteci conțin tipuri ce sunt de ajutor la operații cu fire de execuție, operații cu stream-uri de intrare/ieșire, sisteme de redare grafice și interacțiunea cu diferite dispozitive hardware externe, BCL oferă suport pentru numeroase servicii solicitate de multe aplicații din lumea reală.

#### **3.1.2 Spații de nume**

.NET pune la dispoziție conceptul de spațiu de nume, pentru a ține tipurile definite, bine organizate într-o librărie. Un spațiu de nume este o grupare de tipuri legate, conținute într-un assembly sau posibil răspândite în mai multe assembly-uri înrudite. De menționat faptul că un assembly poate conține oricâte spații de nume și fiecare spațiu de nume poate conține oricât de multe tipuri.

Spații de nume importante în .NET:

- System – are în componență multe tipuri ce se ocupă de operații matematice, generare de numere aleatoare, excepții, etc.
- System.Collection – conține containere și interfețe ce au un rol important în construirea colecțiilor specializate.
- System.IO – definește tipuri ce interacționează cu citirea/scrierea datelor.
- System.Net – este unul din spațiile de nume care ajută la construirea aplicațiilor web.
- System.Threading – conține tipuri de date care ajută la administrarea firelor de execuție

### **3.2. Prezentare Generală a limbajului C#**

C# este unul dintr multele limbaje de programare care au la bază platforma.NET. C# poate fi folosit la crearea aplicațiilor web, serviciilor, aplicațiilor desktop sau pentru librării ce pot fi folosite în diferite aplicații. C# este o evoluția a limbajelor C și C++, și este un limbaj orientat-obiect.



C# este un limbaj sensibil la majuscule. Singura regulă este ca toate cuvintele cheie din C# să fie scrise cu minuscule (*public*, *class*, *new*), în timp ce spațiile de nume, tipurile, membrii unui tip, conform convenției, încep cu majuscule, iar fiecare cuvânt încorporat va începe de asemenea cu majusculă.

### 3.2.1. Tipuri de date

Ca orice limbaj de programare, C# definește cuvinte cheie pentru tipurile fundamentale de date, care sunt utilizate pentru reprezentarea variabilelor locale, membrilor unor clase, tipul de returnare a unei metode sau parametri.

În tabelul următor avem exemplificate toate tipurile de date de sistem, și detalii despre acestea.

Cuvant cheie C#	Tip de sistem	Limite	Numar de biti pentru stocare	Valoare implicita	Descriere
bool	System.Boolean	true sau false	8	false	Tip boolean
sbyte	System.Sbyte	-128 – 127	8	0	Tip întreg cu semn
byte	System.Byte	0 – 255	8	0	Tip întreg fără semn
short	System.Int16	-32,768 – 32,767	16	0	Tip întreg cu semn
ushort	System.UInt16	0 – 65,535	16	0	Tip întreg fără semn
int	System.Int32	-2,147,483,648 – 2,147,483,647	32	0	Tip întreg cu semn.
uint	System.Uint32	0 – 4,294,967,295	32	0	Tip întreg fără semn
long	System.Int64	-9,223,372,036,854,775,808 – 9,223,372,036,854,775,807	64	0L	Tip întreg cu semn
ulong	System.UInt64	0 – 18,446,744,073,709,551,615	64	0	Tip întreg fără semn
char	System.Char	U +0000 – U +ffff	16	'\0'	Tip caracter din setul Unicode
float	System.Single	-3.4 x 10 <sup>38</sup> – + 3.4 x 10 <sup>38</sup>	32	0.0F	Tip cu virgulă mobilă, simplă precizie
double	System.Double	(+/-)5.0 x 10 <sup>-324</sup> to (+/-)1.7 x 10 <sup>308</sup>	64	0.0D	Tip cu virgulă mobilă, dublă precizie
decimal	System.Decimal	(-7.9 x 10 <sup>28</sup> to 7.9 x 10 <sup>28</sup> ) / 100 - 28	128	0.0M	Tip zecimal cu 28 cifre



					semnificative
string	System.String	Limitat doar de memoria sistemului	-	null	O secvență de caractere
object	System.Object	Poate reține orice tip de date	-	null	Clasă de bază a tuturor tipurilor din .NET

### 3.2.2. Declararea și inițializarea variabilelor

Declaraarea unei variabile în interiorul scopului unui membru se face specificând tipul variabilele, urmat de numele acesteia.

*string declarareaUnuiSirDeCaractere;*

Conform standardelor de bune practici, este recomandată doar folosirea variabilelor inițializate, astfel evitându-se erorile de compilare. Acest lucru se poate realiza în mai multe moduri:

*int i =0;*

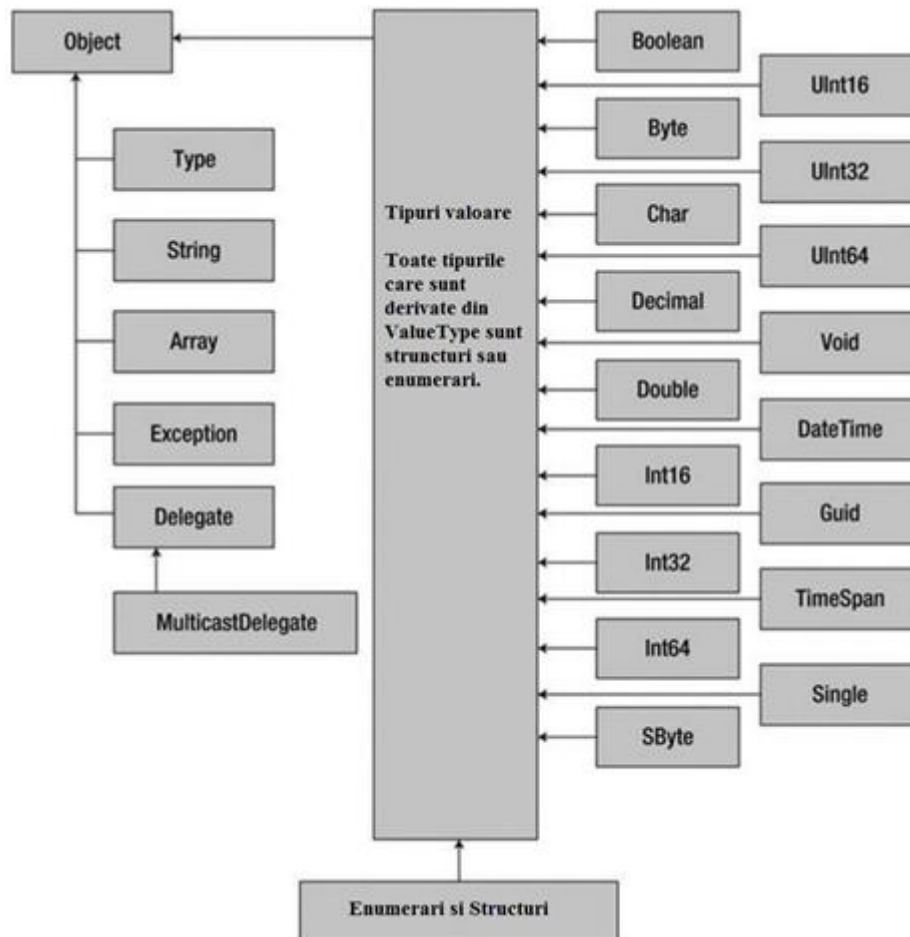
*string s;*  
*s= "inițializarea unui sir de caractere";*

Declararea mai multor variabile de același tip:

*bool b, b1, b2;*  
*bool varBool = true, varBool2 = false;*

Tipurile de date de bază din C# sunt aranjate într-o anumită ierarhie. Această ierarhie poate fi observată în imaginea de mai jos.





### 3.2.3. Tipuri referință și tipuri valoare

În c# există două feluri de tipuri:

- Tipuri referinta – variabila de acest tip conține referința către o adresă unde este memorată valoarea sa
- Tipuri valoare – variabila va conține direct valoarea acesteia

### 3.2.4. Tipul class

Tipul *class* este componenta fundamentala in .NET. Acest tip, este un tip referință și poate conține constructori, destructori, operatori, proprietăți, metode, constante, câmpuri.

Modificatorii de acces:

- **public** : acces nelimitat;
- **protected** : poate fi accesat doar în clasa în care este declarat și în clasele derivate din aceasta;



- **internal** : acces nelimitat în assembly-ul în care este definită clasa conținătoare;
- **protected internal** : poate fi accesat doar în assembly-ul în care este definită clasa conținătoare sau în clasele derivate din aceasta;
- **private** : poate fi accesat doar în clasa în care este declarat. Acesta este și modificatorul de acces implicit pentru un membru al unei clase.

În C# sunt definiți următorii membri ale unei clase:

- **constante** : valoarea acestui membru, nu poate fi modificată, și este evaluată la compilare.
- **Câmpul**: un membru asociat fiecărei instanțe a clasei
- **Metoda**: reprezintă o succesiune de acțiuni. Aceasta poate fi accesată prin intermediul instanței clasei, sau prin intermediul numelui clasei când aceasta este statică.
- **Proprietatea**: acest membru oferă accesul la unul din câmpurile clasei
- **Constructor**: este apelat la initializarea unei instanțe a clasei, și conține o serie de acțiuni
- **Destructorul**: total opus constructorului ca și funcționalitate, acesta este apelat în momentul în care se dorește distrugerea obiectului.

O clasă poate avea unul sau mai mulți constructori. Aceștia pot avea un număr variat atât de parametri, cât și de tipuri. În cazul în care nu este definit niciun constructor, compilatorul va apela constructorul implicit.

### 3.2.5. Tipul Struct

În aparență, acest tip de date este tot un fel de clasă, însă în realitate, diferența este majoră. Acest tip de date, este un tip valoare în timp ce clasa este un tip referință.

### 3.2.6. Tipul Interface

O interfață nu este nimic mai mult decât un set de membri abstracți. Un membru abstract este un simplu protocol care nu are o implementare implicită. Membrii pe care îi poate conține o interfață sunt :

- metode;
- proprietăți;
- indexatori;
- evenimente



O interfață nu poate conține constante, câmpuri, operatori, constructori, destructori sau tipuri. Membrii unei interfețe sunt automat publici, aceștia nu pot avea un alt modificador de acces. De asemenea, membrii nu pot fi statici. O interfață poate implementa oricâte alte interfețe.

Pentru a implementa o interfață, o clasă sau o structură trebuie să definească o implementare (sau să îl declare abstract) pentru fiecare membru al interfeței. Fiecare membru trebuie să aibă același nume și semnătură, trebuie să fie public și nu poate fi static.

Utilizând interfețe putem include un comportament unei clase din surse multiple. Această capacitate este importantă în C# deoarece acesta nu suportă moștenire multiplă a claselor. În plus trebuie să utilizăm interfețe pentru a simula moștenirea în cazul structurilor, pentru că acestea nu pot deriva dintr-o clasă sau o altă structură.

Interfețele în .NET sunt prefixate cu litera "I". Atunci când cream propriile interfețe, este considerată o bună practică să facem același lucru.

### 3.3. Programarea orientată-obiect

În anii 1960, programare orientată – obiect a fost folosită pentru prima dată împreună cu limbajul Simula, care a introdus concepte importante ce stau la baza programării orientate obiect din zilele noastre. Printre aceste concepte se numără clasa, obiectul, moștenirea.

POO este o paradigmă a programării bazată pe ideea de obiect, ce poate conține date și cod, în forma procedurilor (cunoscute și sub numele de metode).

Principiile de bază a programării orientate-obiect :

- **Abstractizarea** – Este posibilitatea ca un program să ignore unele aspecte ale informației pe care o manipulează, adică posibilitatea de a se concentra asupra esențialului. Fiecare obiect în sistem are rolul unui “actor” abstract, care poate executa acțiuni, își poate modifica și comunica starea și poate comunica cu alte obiecte din sistem fără a dezvălui cum au fost implementate acele facilități. Procesele, funcțiile sau metodele pot fi de asemenea abstracte, și în acest caz sunt necesare o varietate de tehnici pentru a extinde abstractizarea:
- **Încapsularea** – numită și *ascunderea de informații*: Asigură faptul că obiectele nu pot schimba starea internă a altor obiecte în mod direct (ci doar prin metode puse la dispoziție de obiectul respectiv); doar metodele proprii ale





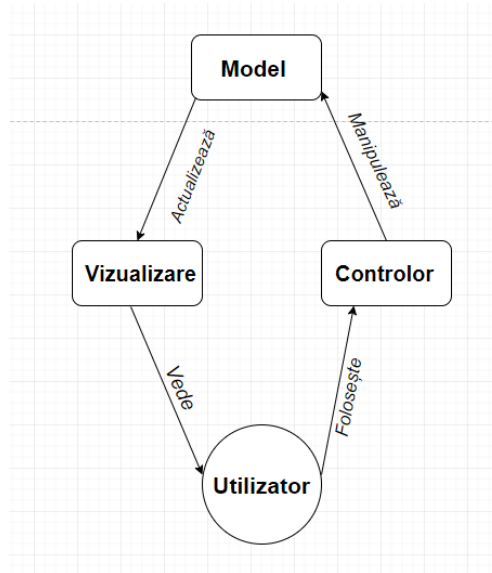
obiectului pot accesa starea acestuia. Fiecare tip de obiect expune o interfață pentru celelalte obiecte care specifică modul cum acele obiecte pot interacționa cu el.

- **Polimorfismul** – Este abilitatea de a procesa obiectele în mod diferit, în funcție de tipul sau de clasa lor. Mai exact, este abilitatea de a redefini metode pentru clasele derivate. De exemplu pentru o clasă Figura putem defini o metodă arie. Dacă Cerc, Dreptunghi, etc. ce vor extinde clasa Figura, acestea pot redefini metoda arie.
- **Moștenirea** – Organizează și facilitează polimorfismul și încapsularea, permițând definirea și crearea unor clase specializate plecând de la clase (generale) deja definite - acestea pot împărtăși (și extinde) comportamentul lor, fără a fi nevoie de a-l redefini. Aceasta se face de obicei prin gruparea obiectelor în clase și prin definirea de clase ca extinderi ale unor clase existente. Conceptul de moștenire permite construirea unor clase noi, care păstrează caracteristicile și comportarea, deci datele și funcțiile membru, de la una sau mai multe clase definite anterior, numite *clase de bază*, fiind posibilă redefinirea sau adăugarea unor date și funcții noi. Se utilizează ideea: "Anumite obiecte sunt similare, dar în același timp diferite". O clasă moștenitoare a uneia sau mai multor clase de bază se numește *clasă derivată*. Esența moștenirii constă în posibilitatea refolosirii lucrurilor care funcționează.

### 3.4. Arhitectura MVC.

MVC(model-view-controller, model-vizualizare-controlor) este un model arhitectural des folosit în crearea interfețelor pentru utilizatori, ce împarte aplicația în 3 părți interconectate. Succesul modelului se datorează izolării logicii de business față de considerentele interfeței cu utilizatorul, rezultând o aplicație unde aspectul vizual și nivelele inferioare ale regulilor de business sunt mai ușor de modificat, fără a afecta alte nivele.





#### 3.4.1. Componente:

Conform și numelui, această arhitectură prezintă 3 componente principale:

**Modelul:** este componenta centrală. Scopul acestei componente este de a se ocupa de date, logica și regulile aplicației. Aceasta primește informațiile trimise de utilizator, prin intermediu controlorului.

**Vizualizarea :** Acestui membru al familiei îi corespunde reprezentarea grafică, sau mai bine zis, exprimarea ultimei forme a datelor: interfața grafică ce interacționează cu utilizatorul final.

**Controlor:** prin intermediul acestei componente, controlăm accesul la aplicația noastră, primește informațiile de la utilizator și le transformă în comenzi pentru model.

#### 3.4.2. Avantaje:

- Dezvoltare simultană : mai mulți programatori pot lucra simultan la componentele acestei arhitecturi fără a exista conflicte. Spre exemplu, o echipa de programatori poate fi împărțită în 2 grupe: front-end și back-end. Cei din back-end pot lucra la structura datelor și modul în care utilizatorul va interacționa cu ele, fără ca interfața să fie terminată, în



același timp cei din front-end pot crea și testa un plan general al unei aplicații înainte ca structura informației să fie disponibilă.

- Coeziune ridicată : coeziunea măsoară puterea relațiilor dintr-o metodă și informația unei clase.
- Cuplare redusă
- Ușurința la modificare: datorată separației responsabilităților.
- Un model poate avea mai multe vizualizări.

#### 3.4.3. Dezavantaje:

- Navigarea prin cod: din cauza introducerii a noilor straturi de abstractizare, navigarea prin cod poate deveni greoaie.
- Învățarea poate fi dificilă: programarea unei aplicații folosind MVC necesită o gamă destul de mare de cunoștințe în mai multe tehnologii.

### 3.5. Entity Framework

Entity framework este un set de tehnologii în ADO.NET ce suportă dezvoltarea de aplicații software cu baze de date, aplicații orientate pe obiecte. Comenzile din ADO.NET lucrează cu scalari (date la nivel de coloană dintr-o tabelă) în timp ce Entity Framework lucrează cu obiecte.

#### 3.5.1. Arhitectura Entity Framework

- *Provideri specifici pentru sursa de date (Data Source)* ce abstractizează interfețele ADO.NET pentru conectarea la baza de date.
- *Provider specific* ce translatează comenzile Entity SQL în cereri native SQL.
- *Parser EDM și mapare vizualizări* prin trararea specificațiilor SDL (storage data language – model de memorare) al modelului de date, stabilirea asociațiilor dintre modelul relațional și modelul conceptual.
- *Servicii pentru metadata* ce gestionează metadata entităților, relațiilor și mapărilor.
- *Tranzacții* – pentru a putea suporta posibilitățile tranzacționale ale bazei de date.
- *Utilitare pentru dezvoltare*, incluse în mediul de dezvoltare.
- *API pentru nivelul conceptual* – runtime ce expune modelul de programare pentru a scrie cod folosind nivelul conceptual.
- *Componente deconectate* – realizează un cache local pentru dataset și mulțimile entităților.



### 3.5.2. Modele de programare

- Aplicație centrată pe baza de date presupune că baza de date este deja creată și se generează modelul logic ce conține tipurile folosite în logica aplicației. Acest lucru se realizează prin intermediu mediului de dezvoltare. Prin această metodă se generează clasele și fișierele necesare pentru nivelul conceptual.
- Aplicație contrată pe model. În acest caz, accesul se face pe baza modelului conceptual care conține obiectele problemei. Astfel, putem folosi 2 metode :
  - Code-first – dezvoltatorul scrie toate clasele modelului și clasa derivată din DbContext cu toate entitățile necesare, iar apoi cu ajutorul mediului de dezvoltare, se creează și se generează baza de date.
  - Model Design First - mediul de dezvoltare permite generarea unei diagrame a modelului aplicației și pe baza acesteia, se va crea și genera baza de date, tabelele din bază și informațiile adiționale pentru EF.

### 3.5.3. EDM (Entity Data Model)

Entity framework folosește un model numit Entity Data Model, dezvoltat din Entity Relationship Modeling (ERM).

Conceptele principale introduse de EDM sunt :

- Entitatea: acest concept reprezintă structura unei înregistrări, indentificată printr-o cheie.
- Relația: reprezintă legătura dintre entități.

EDM suporta diverse construcții ce extind aceste concepte primare:

- Moștenirea: tipurile entitate pot fi definite astfel încât să fie derivate din alte tipuri. În acest caz, moștenirea este una structurală, adică nu se moștenește comportamentul, ci doar structura tipului entitate de bază. În plus, la această moștenire se satisface relația “este un” dintre tipul derivat și tipul de bază.
- Tipuri complexe: EDM suportă definirea tipurilor complexe și folosirea acestora ca membri ai tipului entitate.

EDM este modelul pe partea de client și constituie fundamentul pentru EF.

Acesta cuprinde 3 niveluri ce sunt independente:

- *Nivelul Conceptual* poate fi modelat prin scriere directă de cod(model de programare code-first) sau folosind un utilitar pentru generarea entităților în



cazul în care avem baza de date deja proiectată(model de programare database-first). Sintaxa pentru nivelul conceptual este definită de *Conceptual Schema Definition Language(CSDL) – Limbajul conceptual de definire a schemei*.

- *Nivelul de memorare*- din EDM definește tabelele, coloanele, relațiile, și tipuri de date ce sunt mapate la baza de date. Sintaxa pentru modelul de memorare este definită de *Store Schema Definition Language(SSDL)*.
- *Nivelul de mapare*- definește maparea(legătura) dintre nivelul conceptual și nivelul de memorare. Printre altele, acest nivel definește cum sunt mapate proprietățile din clasele entitate la coloanele tabelor din baza de date. *Mapping Specification Language(MSL) – Limbaj de specificare a mapărilor* definește sintaxa pentru nivelul de memorare.

### 3.5.4 Servicii în Entity Framework.

Până la versiunea 4 , accesul la serviciile EF era definit numai de clasele *ObjectContext* și *ObjectSet*. Începând cu versiunea 4.1 s-au creat clasele *DbContext* și *DbSet*.

#### 3.5.4.1. Clasa *DbContext*

Această clasă se folosește pentru a executa cereri asupra EDM și a efectua actualizarea bazei de date (insert, update, delete).

```
public partial class DbModelContext : DbContext
{
    public DbModelContext()
        : base("name=DbModelContext")
    {
    }

    public virtual DbSet<AparitieProdus> AparitieProdus { get; set; }
    public virtual DbSet<EvolutiaPretului> EvolutiaPretului { get; set; }
    public virtual DbSet<IstoricCautari> IstoricCautari { get; set; }
    public virtual DbSet<Produs> Produs { get; set; }
    public virtual DbSet<UrmarireProdus> UrmarireProdus { get; set; }
    public virtual DbSet<Utilizator> Utilizator { get; set; }
    public virtual DbSet<Vanzator> Vanzator { get; set; }
```

Dacă folosim modelul de programare code-first, atunci această clasă trebuie să fie scrisă de noi, în caz contrar(folosirea modelului de programare database-first) această clasă este generată de un utilitar apelat de mediul de dezvoltare.

Odată ce avem o instanță a tipului derivat din *DbContext*, putem executa operații asupra bazei de date.



Accesarea unei proprietăți *DbSet* din cadrul contextului reprezintă definiția unei cereri ce returnează entitățile de tipul specificat. Faptul ca accesăm o proprietate nu înseamnă ca cererea se executa.

O cerere este executată cand:

- Este enumerată în *foreach*
- Este enumerată de o operațiune din coletie cum ar fi *ToArray*, *ToDictionary* sau *ToList()*.
- Sunt specificați în cadrul cererii operatorii LINQ *First* sau *Any*

Timpul de viață al contextului

Un context este valabil din momentul în care s-a creat o instanță a acestuia și este disponibil până când se apelează metoda *Dispose* sau este luat în considerare de către *Garbage Collector*.

Exemplu de folosire a clasei *DbContext*:

```
using (var dbContext = new DbModelContext())
{
    var item = dbContext.UrmareProdus.Where(m => m.Id_Produs.Equals(productId) && m.Id_Utilizator.Equals(userId)).FirstOrDefault();
    item.Invalid = true;
    dbContext.SaveChanges();
}
```

Recomandări Microsoft pentru lucrul cu contextul în EF :

- Performanța aplicației poate scădea în cazul în care încărcăm în memorie multe obiecte din cadrul aceluiasi context. Consum sporit de memorie, plus accesarea obiectelor are nevoie de mai mult timp de procesare.
- Dacă o exceptie are ca efect trecerea contextului într-o stare pe care sistemul nu o poate gestiona, este posibil ca aplicația sa fie terminată
- În aplicațiile web, trebuie să folosim instanța unui context pe fiecare cerere și apoi sa eliberăm instanța respectivă. Astfel se evită problemele generate de întreruperea conexiunii, probleme legate de timpul de viață a unei sesiuni, etc
- În aplicațiile WPF sau Windows Forms, se recomanda utilizarea unei instanțe la nivelul unei ferestre.

Proprietăți pentru clasa *DbContext*:

- *ChangeTracker*: ofer acces la funcționalitățile contextului ce se ocupă de urmărirea schimbărilor entităților.
- *Configuration* : ofer acces la opțiunile de configurare pentru context.



- *Database* : creează o instanță de tip Database ce oferă acces la metodele de creare, ștergere, verificări asupra bazei de date.

Metode pentru DbContext:

- *Entry(object)* sau *Entry<TEntry>(TEntry)* : aduce un obiect de tipul *DbEntityEntry* pentru entitatea cerută, oferind acces la informațiile despre entitatea respectivă și abilitatea de a executa acțiuni supra acesteia.
- *OnModelCreating()* : Această metodă este apelată când se a fost inițializat un context derivat, dar înainte ca modelul să fie blocat și folosit pentru crearea acelui context. Implementarea implicită nu face nimic, dar poate fi suprascrisă într-o clasă derivată astfel încât modelul să poată fi configurat în continuare.
- *SaveChanges()* : salvează toate modificările din context, în baza de date.
- *SaveChangesAsync()* : salvează asincron modificările din context, în baza de date.

#### 3.5.4.2. Clasa DbSet

Această clasă este folosită când tipul entității nu este cunoscut în momentul construcției. Este versiunea non-generică pentru *DbSet<TEntity>*.

Metode pentru DbSet:

- *Add, AddRange*: adaugă entitatea în context, aceasta fiind inserată în baza de date la apelul metodei *SaveChanges*.
- *Attach*: atașează o entitate la context. Această entitate va avea stare *unchanged*, și va apărea ca și cum ar fi fost doar citită din baza de date.
- *Create*: Creează o instanță a unei entități. Aceasta nu va fi adăugată sau atașată în set.
- *Find*: găsește o entitate pe baza cheii primare a acesteia. Dacă entitate există în acel context, ea va fi returnată imediat, fără a trimite o cerere. În caz contrar, se lansează o cerere, dacă este găsit, va fi atașată contextului și returnată, altfel va fi returnat *null*.
- *Include*: Include în rezultatul returnat de o interogare, obiecte ce se află în legătură.

#### 3.5.6. Interogări în Entity Framework



Entity Framework 6 suportă 3 tipuri de interogări ce sunt transformate ulterior în interogări SQL asupra bazei de date:

- **Linq-to-Entities:** Language-Integrated Query (LINQ) este un limbaj puternic de interogare introdus de Visual Studio în 2008. După cum sugerează și numele, acest limbaj operează pe un set de entități pentru a accesa datele.

```
using (var dbContext = new DbModelContext())
{
    var item = dbContext.UrmarireProdus.Where(m => m.Id_Produs.Equals(productId) && m.Id_Utilizator.Equals(userId)).FirstOrDefault();

    item.Invalid = true;

    dbContext.SaveChanges();
}

using (var dbContext = new DbModelContext())
{
    tempProds = (from products in dbContext.Produs
        join followedProducts in dbContext.UrmarireProdus on products.Id equals followedProducts.Id_Produs
        where followedProducts.Id_Utilizator == userId
        where followedProducts.Invalid == false
        select products).Include(p=>p.Vanzator).ToList();
}

return tempProds;
```

- **Entity SQL:** este o altă modalitate de a crea o interogare, ce este procesată direct de serviciile obiectelor entity framework și returnează un obiect de tipul *ObjectQuery*.

```
string sqlString = "SELECT VALUE st FROM SchoolDBEntities.Students " +
    "AS st WHERE st.StudentName == 'Bill'";

var objctx = (ctx as IObjectContextAdapter).ObjectContext;

ObjectQuery<Student> student = objctx.CreateQuery<Student>(sqlString);
Student newStudent = student.First<Student>();

using (var con = new EntityConnection("name=SchoolDBEntities"))
{
    con.Open();
    EntityCommand cmd = con.CreateCommand();
    cmd.CommandText = "SELECT VALUE st FROM SchoolDBEntities.Students as st where st.StudentName='Bill'";
    Dictionary<int, string> dict = new Dictionary<int, string>();
    using (EntityDataReader rdr = cmd.ExecuteReader(CommandBehavior.SequentialAccess | CommandBehavior.CloseConnection))
    {
        while (rdr.Read())
        {
            int a = rdr.GetInt32(0);
            var b = rdr.GetString(1);
            dict.Add(a, b);
        }
    }
}
```

- **Limbaj nativ SQL.**





```

using (var ctx = new SchoolDBEntities())
{
    var studentName = ctx.Students.SqlQuery("Select studentid, studentname, standardId
                                              from Student
                                              where studentname='Bill'")
                                              .FirstOrDefault<Student>();
}

```

## 4. Prezentarea aplicației

Aplicația poate fi adăugată în categoria motoarelor de căutare, funcționând după cum urmează:

- Utilizatorul se autentifică
- Navighează pe pagina de căutare a produselor
- Introduce un text spre a fi căutat (de obicei, un șir relativ scurt de caractere)
- Procesatorul preia textul și îl trimite spre procesul de căutare a produselor
- Produsele găsite sunt inserate în baza de date
- Pe interfața utilizatorului fiind aduse din bază
- Dacă utilizatorul s-a hotărât în privința unui produs, poate să își seteze o alertă pentru respectivul produs.
- O altă parte a procesatorului se va ocupa de verificarea continuă a alertelor; dacă un produs a îndeplinit toate condițiile impuse, această parte a procesatorului va trimite un e-mail notificând utilizatorul.

Aplicația propusă este destinată doar utilizatorilor normali, fără diferite roluri sau drepturi speciale. Aceasta este creată din 2 mari componente : interfața pentru



utilizator și partea de procesare în care se procesează mesaje, se trimit notificări, se actualizează informații deja existente;

## 4.1 Procesatorul

Această componentă stă la baza proiectului, și este formată din 3 componente :

- procesatorul de mesaje – această componentă rulează continuu
- partea de actualizare a datelor – componenta de actualizare este setată să ruleze o dată la 12 ore
- sistemul de notificare. – pornește la 4 ore

Cele 3 componente sunt pornite în paralel fiecare urmându-și căile separat.

```
class Program
{
    static void Main(string[] args)
    {
        Parallel.Invoke(
            () => StartSearchUtilities(),
            () => StartMainCrawl(),
            () => StartAlertCheck()
        );
    }
}
```

### 4.1.1. Procesatorul de mesaje

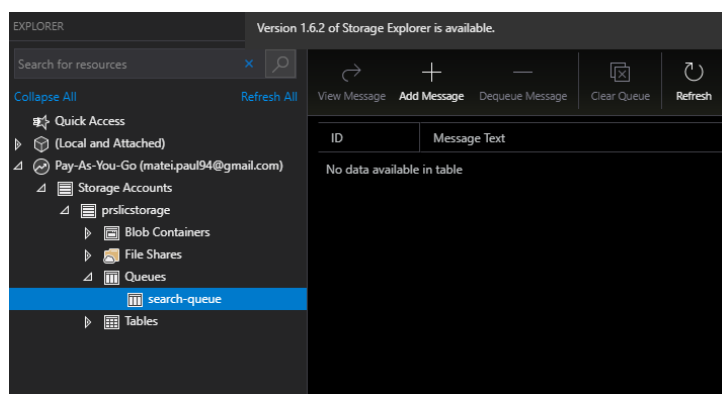
Conform scurtei descrieri a aplicației propuse, în momentul în care un utilizator caută un produs, textul introdus de acesta în pagina de căutare este preluat de această componentă.

Fiind două componente independente, am hotărât să folosesc Cozile Azure și baza de date ca fiind componente de legătură.

Mediul de stocare de tip coadă oferit de Microsoft(), este un serviciu de stocare a unui număr mare de mesaje ce poate fi accesat de oriunde în lume, apeluri autentificate folosind unul din protocoalele HTTP sau HTTPS. Un singur mesaj poate fi de maxim 64Kb și o coadă poate conține milioane de mesaje, până la limita impusă de capacitatea maximă a contului pe care a fost făcută această coadă. Acest serviciu a fost creat pentru a oferi un mediu de transmitere a mesajelor prin nor(cloud) între 2

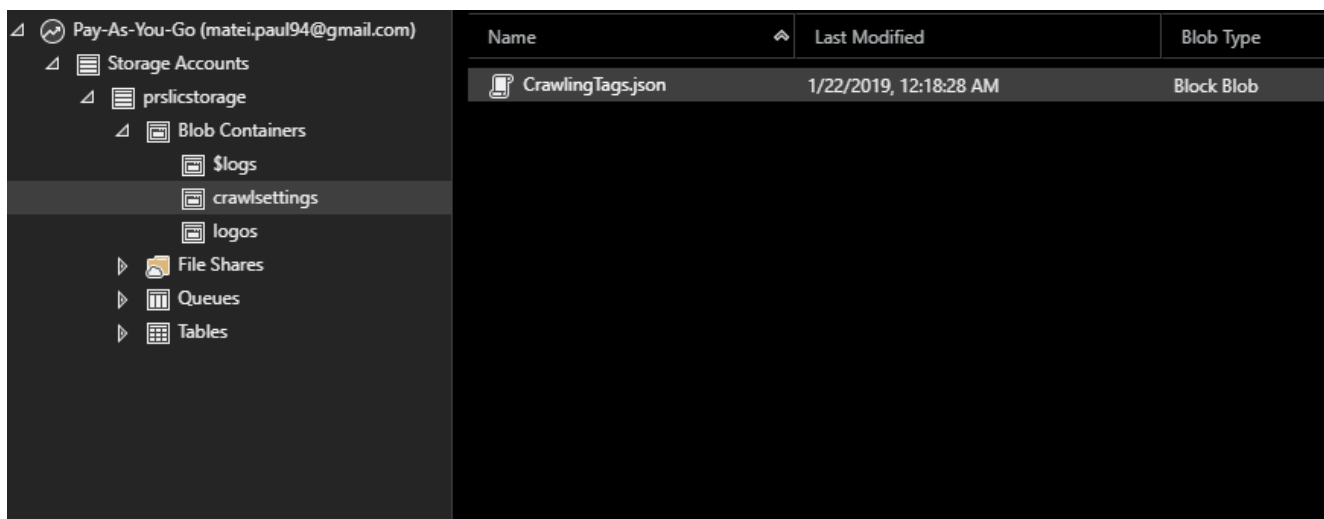


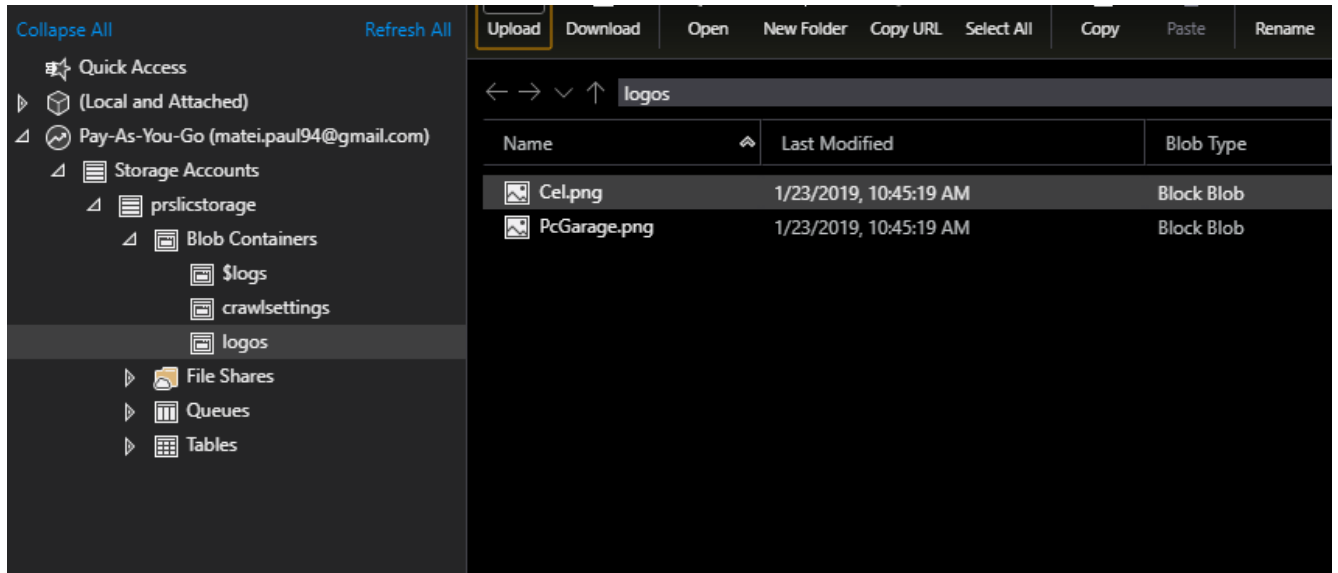
componente decuplate, asta însemnând că cele două componente pot fi scalate independent.



În același timp, am folosit și serviciul de stocare Blob, ce permite programelor să stocheze date nestructurate și binare ce pot fi accesate printr-o cale HTTP sau HTTPS. La fel ca orice componenta oferita de Microsoft, aceasta conține și ea componente de securitate ce controlează accesul si drepturile asupra datelor.

Acest tip de stocare a fost folosit atât pentru reținerea informațiilor legate de procesul de căutare și actualizare, cât și pentru reținerea imaginilor. Această metodă de stocare este foarte eficientă pentru ca evită eventuale conflicte în momentul în care trebuie făcută o modificare minoră asupra unuiuia dintre fișiere; ex: un vânzător își schimbă logoul sau structura site-ului unui vânzător se schimbă.





- Încărcarea setărilor din blob-ul Azure :  
Folosindu-mă de funcționalitățile din spațiul de nume *Microsoft.WindowsAzure.Storage*, un obiect de tip *CloudStorageAccount* se creează folosind metoda *Parse* primind ca parametru șirul de caractere pentru conexiunea la contul Azure.

```
CloudStorageAccount.Parse(ConfigurationManager.ConnectionStrings["AzureConnectionString"].ConnectionString);
```

În fișierul de configurație al aplicației este setat :

```
<add name="AzureConnectionString"
connectionString="DefaultEndpointsProtocol=https;AccountName=prslcstorage;AccountKey=5gomNFZt0wDRBSb5nDgJAS05A3kynuvu7Bz3oXRb0i8G/p3nKiUQV08KgSvPtbtSLLcS+iACwn5R6CKMctmyJA==;EndpointSuffix=core.windows.net"/>
```

După care urmează o serie de apeluri specifice Bloburilor Azure

```
CloudBlobClient blobClient = storageAccount.CreateCloudBlobClient();
CloudBlobContainer containerRef = blobClient.GetContainerReference(container);
CloudBlockBlob blockBlob = containerRef.GetBlockBlobReference(fileName);
blockBlob.DownloadToFile(Path.Combine(path, fileName), FileMode.Create);
```

În acest moment avem fișierul de tip JSON ce conține toate informațiile necesare căutării produselor.



```
{
  "Retailers": [
    {
      "CrawlingTags": {
        "NextProductPageTag": "//div[@class='pageresults']/span[@class='selectat']/following-sibling::a",
        "ProductCellTag": "//div[contains(@class,'productListing-tot')]",
        "ProductImage": "//div[@class='bigImage']/a/img",
        "ProductName": "//h2[@itemprop='name']",
        "ProductPrice": "//span[@itemprop='price']",
        "ProductStock": "//button[contains(@class,'btn-buy')]",
        "SearchUrlFormatDefault": "http://www.cel.ro/cauta/{0}/",
        "SearchUrlFormatPriceAsc": "http://www.cel.ro/cauta/{0}/0c-1",
        "SearchUrlFormatPriceDesc": "http://www.cel.ro/cauta/{0}/0i-1",
        "UrlTag": "//h4[@class='productTitle']/a"
      },
      "RetailerName": "Cel"
    },
    {
      "CrawlingTags": {
        "NextProductPageTag": "//div[@class='pageresults']/span[@class='selectat']/following-sibling::a",
        "ProductCellTag": "//div[@class='product-box']",
        "ProductImage": "//img[@itemprop='image'][1]",
        "ProductName": "//h1[@class='p-name']",
        "ProductPrice": "//meta[@itemprop='price']",
        "ProductStock": "//link[@itemprop='availability']",
        "SearchUrlFormatDefault": "https://www.pcgarage.ro/cauta/{0}",
        "SearchUrlFormatPriceAsc": "https://www.pcgarage.ro/cauta/{0}",
        "SearchUrlFormatPriceDesc": "https://www.pcgarage.ro/cauta/{0}",
        "UrlTag": "//div[@class='pb-name']/a"
      },
      "RetailerName": "PcGarage"
    }
  ]
}
```

Acest fișier urmează a fi deserializat folosind ustensilele disponibile in spatiul de nume *Newtonsoft.Json*; Disponibil in pachetul NuGet;

Dupa deserializare, este creat un obiect de tipul `public class CrawlSettingsModel` ce va retine toate informatiile prezente in document.

```
public class CrawlingTags
{
    public string SearchUrlFormatDefault { get; set; }
    public string SearchUrlFormatPriceAsc { get; set; }
    public string SearchUrlFormatPriceDesc { get; set; }

    public string ProductCellTag { get; set; }
    public string UrlTag { get; set; }
    public string NextProductPageTag { get; set; }

    public string ProductName { get; set; }
    public string ProductPrice { get; set; }
    public string ProductStock { get; set; }
    public string ProductImage { get; set; }
}

public class RetailerConfiguration
{
    public string RetailerName { get; set; }
    public CrawlingTags CrawlingTags { get; set; }
}

public class CrawlSettingsModel
{
    public List<RetailerConfiguration> Retailers { get; set; }
}
```



- Incărcarea mesajului din coada Azure  
Folosind pași asemanatori cu cei de la încărcarea setărilor pentru căuturi, vom verifica daca avem un mesaj in coada. Folosind metoda urmatoare:

```
public bool QueueHasItems()
{
    bool hasItems = false;

    CloudQueueClient queueClient = storageAccount.CreateCloudQueueClient();

    CloudQueue queue = queueClient.GetQueueReference("search-queue");

    var message = queue.PeekMessage();

    if (message != null)
    {
        hasItems = true;
    }

    return hasItems;
}
```

Dacă in coadă exista un mesaj, îl vom extrage si vom porni căutarea :

```
public CloudQueueMessage GetQueueItem()
{
    CloudQueueClient queueClient = storageAccount.CreateCloudQueueClient();

    CloudQueue queue = queueClient.GetQueueReference("search-queue");

    var message = queue.GetMessage();

    if (message != null)
    {
        return message;
    }

    return null;
}
```

Putem observa că in setări este reprezentată o listă de vânzători(*Retailers*), fiecare cu obiectele sale specifice;

Utilitatea ce se va ocupa de procesul de căutare, este pornită in paralel pentru fiecare vânzător/retailer:

```
Parallel.ForEach(model.Retailers
    , new ParallelOptions() { MaxDegreeOfParallelism = 2 }
    , retailer => SearchMessageCrawler.StartMessageCrawling(message,retailer)
    );
```

- Generarea componentei de descărcare a informațiilor despre produse.



Din cauza faptului că fiecare retailer are un website specific, trebuie sa generăm cate un obiect personalizat pentru fiecare. Aceste obiecte implementează o interfață, *IDownloader*

```
public interface IDownloader
{
    void GetProducts(CloudQueueMessage givenMessage);
}
```

- Descărcarea unor produse:

```
public void GetProducts(CloudQueueMessage message)
{
    url = GetSearchUrl(message, retailerConfiguration);
    NavigateLink(url);
}

private void NavigateLink(string url)
{
    string htmlSting = HttpUtils.GetWebRequestResponse(url);
    var htmlDocument = HtmlDocumentUtilities.GetHtmlDocument(htmlSting);
    ExtractProducts(htmlDocument);
    products.SaveProducts(givenMessage);
}
```

Putem observa că avem implementată interfața prin existența metodei *GetProducts*

Metoda *GetSearchUrl* este responsabilă cu generarea unui url catre o pagină de căutare ce corespunde cu mesajul primit de la utilizator și setările vânzătorului respectiv.

Clasele *HttpUtils* și *HtmlDocumentUtilities* sunt 2 clase statice. Acestea se vor ocupa cu trimiterea unui request catre serverul vânzătorului și generarea unui *HtmlDocument*, obiect disponibil în spațiul de nume *HtmlAgilityPack*, disponibil prin intermediul NuGet.

*HttpUtils.cs*



```

private static HttpClient client = new HttpClient();

public static string GetHttpRequestResponse(string url)
{
    Thread.Sleep(2000);
    string html = string.Empty;
    try
    {
        html = client.GetStringAsync(url).Result;
    }
    catch (HttpRequestException requestException)
    {
        GenericLogger.Error($"Http Exception Popped for {url} when trying to GetHttpRequestResponse(string url)", requestException);
    }
    catch (Exception ex)
    {
        GenericLogger.Error("Generic exception Popped in GetHttpRequestResponse(string url)", ex);
    }

    return html;
}

public static string GetWebResponseResponse(string url)
{
    string html = string.Empty;
    try
    {
        Uri uri = new Uri(url);
        HttpWebRequest request = (HttpWebRequest)WebRequest.Create(uri);
        request.Accept = "text/html, */*";
        request.MaximumAutomaticRedirections = 10;
        request.AllowAutoRedirect = true;
        request.AutomaticDecompression = DecompressionMethods.Deflate | DecompressionMethods.GZip;
        request.UserAgent = "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/70.0.3538.110 Safari/537.36";
        request.Host = uri.Host;
        request.Accept = "Accept: text / html,application / xhtml + xml,application / xml; q = 0.9,image / webp,image / apng,*/*;q=0.8";
        request.Headers["Accept-Encoding"] = "gzip, deflate";
        request.Headers["Accept-Language"] = "en-US,en;q=0.9,ro;q=0.8";

        using (HttpWebResponse response = (HttpWebResponse)request.GetResponse())
        {
            using (Stream stream = response.GetResponseStream())
            {
                using (StreamReader reader = new StreamReader(stream))
                {
                    html = reader.ReadToEnd();
                }
            }
        }
    }
    catch (WebException webEx)
    {
        GenericLogger.Error($"WebException popped when tring to extract {url}, \n Ex : {webEx.Message} \n {webEx.StackTrace}");
    }
    catch (Exception ex)
    {
        GenericLogger.Error($"Exception {ex.Message} popped when trying to get source for {url}, \n \n {ex.StackTrace}");
    }

    return html;
}

```

## HtmlDocumentUtilities





```

public static class HtmlDocumentUtilities
{
    public static HtmlDocument GetHtmlDocument(string htmlSourceString)
    {
        HtmlDocument doc = new HtmlDocument();

        if (string.IsNullOrEmpty(htmlSourceString))
        {
            return null;
        }

        doc.LoadHtml(htmlSourceString);

        return doc;
    }
}

```

- Extragerea informatiilor despre produs:

După ce am obtinut un document Html valid, începem extragerea informațiilor despre produse:

Componenta de descărcare se va ocupa de extragerea Url-urilor produselor găsite pe pagina de căutare, dupa care le va trimite către o altă componentă ce se va ocupa cu analiza acelu url.

```

private void ExtractProducts(HtmlDocument htmlDocument)
{
    try
    {
        var productUrls = htmlDocument.DocumentNode.SelectNodes(retailerConfiguration.CrawlingTags.UrlTag)
            .Select(m => m.GetAttributeValue("href", string.Empty)).ToList();

        if (productUrls.Count < 1)
        {
            GenericLogger.Info($"No products were found for {givenMessage}");
            return;
        }

        ExtractProductInformation(productUrls);
    }
    catch (Exception ex)
    {
    }
}

private void ExtractProductInformation(List<string> productUrls)
{
    foreach (var url in productUrls)
    {
        var parser = new CelProductParser(url, givenMessage, retailerConfiguration);
        products.AddProduct(parser.GetProduct());
    }
}

```

- Adunarea propriu-zisă a informațiilor despre produse:



În interiorul unui analizator vom găsi următoarele metode:

```
public class CelProductParser:IParseProduct
{
    private string url;
    private string givenMessage;
    private RetailerConfiguration retailer;
    private HtmlDocument doc;

    public CelProductParser(string url, string givenMessage, RetailerConfiguration retailer)
    {
        this.url = url;
        this.givenMessage = givenMessage;
        this.retailer = retailer;
        doc = GetProductDocument(url);
    }

    private HtmlDocument GetProductDocument(string url)
    {
        string html = HttpUtils.GetWebRequestResponse(url);
        doc = HtmlDocumentUtilities.GetHtmlDocument(html);
        return doc;
    }

    public Produs GetProduct()
    {
        Produs prd = new Produs();

        prd.Url = url;
        prd.Denumire = HtmlDocumentUtilities.ExtractNodeValue(doc, retailer.CrawlingTags.ProductName, m => m.InnerText.Replace(",", string.Empty));
        prd.Url_Imagine = HtmlDocumentUtilities.ExtractNodeValue(doc, retailer.CrawlingTags.ProductImage, m => m.GetAttributeValue("src", string.Empty));
        prd.Pret = decimal.Parse(HtmlDocumentUtilities.ExtractNodeValue(doc, retailer.CrawlingTags.ProductPrice, m => Regex.Match(m.InnerText.Replace("&#46;", string.Empty), @"\d+\.\d+(?:\d+)?").Value.Trim()));
        prd.Stock = (HtmlDocumentUtilities.ExtractNodeValue(doc, retailer.CrawlingTags.ProductStock, m => string.IsNullOrEmpty(m.InnerText) ? "OutOfStock" : "InStock"));

        return prd;
    }
}
```

Constructorul :

```
public CelProductParser(string url, string givenMessage, RetailerConfiguration retailer)
```

Metoda publica GetProduct ce returneaza un produs:

```
public Produs GetProduct()
```

Metoda privată pentru generarea unui document Html despre care am discutat mai sus, folosind aceleași clase statice:

```
private HtmlDocument GetProductDocument(string url)
```

În clasa *HtmlDocumentUtilities* există metoda *ExtractNodeValue*

```
public static string ExtractNodeValue(HtmlDocument doc, string XPath, Func<HtmlNode, string> func)
{
    try
    {
        if (string.IsNullOrEmpty(XPath))
            return string.Empty;

        var node = doc.DocumentNode.SelectSingleNode(XPath);

        return (func != null && node != null) ? func(node) : string.Empty;
    }
    catch (Exception ex)
    {
        return string.Empty;
    }
}
```

Folosita pentru extragerea informatiei despre un produs, aceasta folosind funcționalitățile pachetului *HtmlAgilityPack*.

```
prd.Denumire = HtmlDocumentUtilities.ExtractNodeValue(doc,
retailer.CrawlingTags.ProductName, m => m.InnerText.Replace(",", string.Empty));
```

- Salvarea produselor in baza de date:



După ce tot procesul de descărcare se termină, obținem un obiect de tipul *RetailerCrawlProductCollection* care moștenește un obiect de tipul *List<Produs>*;

```
public class RetailerCrawlProductCollection : List<Produs>
```

La adăugarea fiecărui produs in această colecție, se vor adăuga mai multe informații care nu țin neaparat de procesul de extragere, cum ar fi id-ul vanzatorului care comercializează produsul respectiv, id-ul produsului,etc

```
public void AddProduct(Produs prd)
{
    prd.Id_Vanzator = RetailerID;
    prd.Cod_Denumire_Produs = GetDescriptionCode(prd.Denumire);
    prd.Id = StringToGuid(prd.Url);
    prd.Sters = false;
    prd.EvolutiaPretului.Add(GetNewPriceForEvolution(prd));
    prd.Data_Creat = DateTime.UtcNow;
    this.Add(prd);
}

private EvolutiaPretului GetNewPriceForEvolution(Produs product)
{
    EvolutiaPretului evol = new EvolutiaPretului();
    evol.Id_Produs = product.Id;
    evol.Pret = product.Pret;
    evol.Id = Guid.NewGuid();
    evol.Data_Actualizare = DateTime.UtcNow;
    return evol;
}

private Guid StringToGuid(string url)
{
    MD5 md5Hasher = MD5.Create();
    byte[] data = md5Hasher.ComputeHash(Encoding.Default.GetBytes(url));
    return new Guid(data);
}
```

Putem observa că id-ul unui produs este generat pe baza url-ului. Astfel asigurând unicitatea(atât din cauza faptului ca un url nu poate conduce la mai mult de 1 produs, cât și din cauza faptului că este generat un obiect de tip GUID - globally unique identifier)acestui camp, fiind și cheie primara in baza de date. Acest lucru a fost creat pentru a evita eventuale conflicte la inserția în baza de date.

Inserția și toate celelalte operațiuni pe baza de date este realizată prin intermediul EntityFramework.

#### 4.1.2. Actualizarea datelor



Aplicația propusă are în componență și un sistem de urmărire a evoluției pretului unui produs. Pentru a putea genera un grafic al evoluției avem nevoie de mai multe date reale. Pentru aceasta, a fost creat procesul de actualizare a datelor, care rulează o dată la 12 ore. Această componentă fiind independentă de celelalte și rulând pe un fir de execuție diferit, putem întârzia execuția acestuia prin intermediul metodelor spațiului de nume *System.Threading*.

```
GenericLogger.Info("Starting CrawlUtilities");  
var model = CrawlSettingsHelper.LoadCrawlSettings("Update");  
MainCrawlStarter.CrawlStart(model);  
GenericLogger.Info($"Update process finished starting again in 12 hours");  
Thread.Sleep(1000 * 60 * 60 * 12);
```

Putem observa apelul metodei *Sleep* al clasei statice *Thread* ce primește ca parametru un număr de milisecunde.  $1000 * 60 * 60 * 12$  este numărul de milisecunde dintr-un interval de 12 ore. Astfel a fost "programat" executarea funcționalităților de actualizare, o dată la 12 ore.

După încărcarea setărilor de extragere(aceleași setări menționate mai sus în procesul de căutare a produselor), se lansează execuția *CrawlStart*. În care pleacă 2 fire de execuție diferite pentru fiecare vânzător în parte.

```
public static void CrawlStart (CrawlSettingsModel model)  
{  
    try  
    {  
        Parallel.ForEach  
            (model.Retailers,  
            new ParallelOptions() { MaxDegreeOfParallelism = 2 },  
            m => StartCrawling(m)  
            );  
    }  
    catch (Exception ex)  
    {  
        GenericLogger.Error("Exception thrown at MainCrawlStarter", ex);  
    }  
}  
  
private static void StartCrawling(RetailerConfiguration retailer)  
{  
    CrawlManager manager = new CrawlManager(retailer);  
    manager.StartCrawling();  
}
```

Inițializarea fiecărei componente ce rulează în paralel, se face la fel ca în cazul procesului de actualizare a datelor, în funcție de vânzător/retailer(se creează o instanță



a unui analizator care implementează interfața *IParseProduct*) și se populează un obiect de tipul *RetailerCrawlProductCollection* prin intermediul *EntityFramework*. De menționat că în momentul în care populăm acest obiect, vom include și tabela *EvolutiaPretului* astfel încât să putem adăuga noi informații fără să fim nevoiți să facem o serie de validări asupra acestei tabele.

```
private IParseProduct parser;

private void GetParser()
{
    switch (RetailerConfig.RetailerName)
    {
        case "Cel":
        {
            parser = new CelProductParser(RetailerConfig);
            break;
        }
        case "PcGarage":
        {
            parser = new PcGarageProductParser(RetailerConfig);
            break;
        }
        default:
            break;
    }
}

private void GetProductsFromDB(DbModelContext dbContext)
{
    retailerId = dbContext.Vanzator.Where(m => m.Nume.Equals(RetailerConfig.RetailerName)).Select(m => m.Id).FirstOrDefault();

    products = new RetailerCrawlProductCollection(RetailerConfig.RetailerName);

    var prods = (from prod in dbContext.Produs
        join evol in dbContext.EvolutiaPretului on prod.Id equals evol.Id_Produs
        join retailer in dbContext.Vanzator on prod.Id_Vanzator equals retailer.Id
        where retailer.Id.Equals(retailerId)
        select prod).Include(p => p.EvolutiaPretului).ToList();

    products.AddRange(prods);
}
```

```
public interface IParseProduct
{
    void GetProduct(ref Produs product);
}
```

În continuare, vom trimite spre analizator (prin intermediul interfeței), pe rând, produsele trimise prin referință, urmând ca după extragerea informațiilor, să salvăm modificările făcute



```
private void ParseProducts(DbModelContext dbContext)
{
    int counter = 0;

    do
    {
        Thread.Sleep(3 * 1000);

        if (products.Count > 0)
        {
            var currentProduct = products[counter];
            parser.GetProduct(ref currentProduct);
            counter++;

            try
            {
                dbContext.SaveChanges();
            }
            catch (Exception ex)
            {
                GenericLogger.Error($"Error when trying to save changes for updating process for {currentProduct.Url}, \n ex : {ex.Message} \n {ex.StackTrace} \n {ex.InnerException.InnerException.Message}");
            }
        }
    } while (counter < products.Count);
}
```

De menționat este faptul că, a fost folosită o întârziere de 3 secunde (*Thread.Sleep(3\*1000)*) a firului de execuție. Acest lucru a fost setat pentru a nu suprasolicita serverele vânzătorilor. Un rezultat direct al acestei abordări este evitarea blocării din partea serverelor. Dat fiind faptul că din ce în ce mai mulți vânzători apar în spațiul on-line, crește numărul de roboți programați să adune informații despre produse.

Conform celor menționate mai sus, paralelizarea acestui proces nu este recomandată, mai mult nu pot fi folosite metode care țin cont de starea obiectelor ce fac parte din colecția ce se trimite spre a fi paralelizată.

Extragerea propriu-zisă a informațiilor despre produs, se va face cu același obiect folosit ca analizator în procesul de căutare a produselor, care implementează interfața *IParseProduct* ;

```
public PcGarageProductParser(RetailerConfiguration retailerConfiguration)
{
    updateConfiguration = retailerConfiguration;
}

public void GetProduct(ref Produs product)
{
    _document = GetProductDocument(product.Url);
    ExtractNeededInformation(ref product);
}

private void ExtractNeededInformation(ref Produs product)
{
    product.Denumire = (HtmlDocumentUtilities.ExtractNodeValue(_document, updateConfiguration.CrawlingTags.ProductName, m => m.InnerText.Trim().Replace(",", string.Empty)));
    product.Pret = Decimal.Parse(HtmlDocumentUtilities.ExtractNodeValue(_document, updateConfiguration.CrawlingTags.ProductPrice, m => Regex.Match(m.InnerText.Replace("&#46;", string.Empty), @"^\d+(\.\d+)?").Value.Trim()));
    product.Stock = (HtmlDocumentUtilities.ExtractNodeValue(_document, updateConfiguration.CrawlingTags.ProductStock, m => string.IsNullOrEmpty(m.InnerText) ? "OutOfStock" : "InStock"));
    product.Url_Imagine = (HtmlDocumentUtilities.ExtractNodeValue(_document, updateConfiguration.CrawlingTags.ProductImage, m => m.GetAttributeValue("src", string.Empty)));

    EvolutiaPretului priceEvolution = new EvolutiaPretului();
    priceEvolution.Id = Guid.NewGuid();
    priceEvolution.Pret = product.Pret;
    priceEvolution.Id_Produs = product.Id;
    priceEvolution.Data_Actualizare = DateTime.UtcNow;
    priceEvolution.Produs = product;

    product.EvolutiaPretului.Add(priceEvolution);
}
```

Putem observa că ne folosim de aceleași metode ale claselor statice *HtmlDocumentUtilities* și *HttpUtils* pentru a extrage informațiile necesare despre produs, exact ca și în cazul procesului de cautare. Diferența constă în faptul că acum generăm și o instanță al obiectului *EvolutiaPretului* ce conține informația despre prețul actual al produsului, aceasta ne va ajuta în generarea graficului evoluției prețului unui produs, și care va fi inserat în baza de date, fiind în relație cu produsul despre care am extras detaliile.



### 4.1.3. Sistemul de notificare-alertă preț

O altă funcționalitate a acestei aplicații, este notificarea clientului. În momentul în care un utilizator a setat o alertă pentru un produs, îi este impus să ofere o limită de preț. Această limită este prețul maxim la care utilizatorul ar dori să cumpere un anumit produs. Când produsul ales ajunge sub limita impusă, se pornește procesul de notificare. Un e-mail va fi trimis doar în cazul în care utilizatorul nu a fost deja notificat pentru produsul respectiv, sau a fost ultima notificare a fost cu 2 zile în urma.

Acest proces pornește o data la 4 ore, folosind metoda mai sus enunțată în cazul actualizării produselor. În condițiile în care acest proces rulează de 6 ori pe zi, constrângerea de notificare(o data la 2 zile) a fost setată pentru a nu polua casuța postală a utilizatorului.

```
private static void StartAlertCheck()
{
    while (true)
    {
        Thread.Sleep(2 * 1000);
        AllertCheckUtils utils = new AllertCheckUtils();
        utils.StartAlertCheck();

        Thread.Sleep(4 * 3600 * 1000);
    }
}
```

În continuare, la instanțierea obiectului de tip *AllertCheckUtils* se inițializează lista alertelor înregistrate. După care este trimisă, prin referință, la metoda *CheckFollowedInformation*

```
try
{
    using (var dbModel = new DbModelContext())
    {
        var followInformation = GetFollowedInformation(dbModel);

        CheckFollowedInformation(ref followInformation);

        dbModel.SaveChanges();
    }
}
catch (Exception ex)
{
}
```



Inițializarea reprezintă popularea obiectului cu informații din baza de date, lucru executat prin intermediul *EntityFramework*.

```
private List<UrmarireProdus> GetFollowedInformation(DbModelContext dbContext)
{
    var information = new List<UrmarireProdus>();

    information = (from info in dbContext.UrmarireProdus
        join user in dbContext.Utilizator on info.Id_Utilizator equals user.Id
        join product in dbContext.Produs on info.Id_Produs equals product.Id
        where info.Invalid == false
        select info ).Include(p => p.Utilizator).Include(p => p.Produs).ToList();

    return information;
}
```

De menționat că a fost nevoie să includem atât tabela Utilizator, cât și tabela produs. Notificarea trebuie trimisă către utilizator, în cazul în care prețul produsului este sub valoarea setată.

În continuare, procesul iterează prin toate alertele setate, verifică dacă sunt îndeplinite toate condițiile și trimite un e-mail de notificare către utilizator, în legătură cu produsul curent.

```
foreach (var information in followInformation)
{
    try
    {
        decimal followPrice = information.Limita_pret;
        decimal currentPrice = information.Produs.Pret;

        if (followPrice <= currentPrice)
        {
            if ((!information.UtilizatorNotificat) ||
                (information.UtilizatorNotificat &&
                 information.DataNotificarii.Value.ToShortDateString()
                 .Equals(DateTime.UtcNow.AddDays(-2).ToShortDateString())))
            {
                EmailHelper.SendEmail(destination: information.Utilizator.Email,
                    subject: $"Alerta Pret pentru produs {information.Produs.Denumire.Substring(0, 50)}...",
                    body: GenerateBodyFromInfo(information));

                information.UtilizatorNotificat = true;
                information.DataNotificarii = DateTime.UtcNow;
            }
        }
    }
    catch (Exception ex)
    {
    }
}
```





De menționat:

```
information.UtilizatorNotificat = true;  
information.DataNotificarii = DateTime.UtcNow;
```

Aceste 2 proprietăți au primit valori noi, pentru ca putea fi setată constrângerea de notificare o dată la doua zile; acest lucru este verificat prin clauza *IF* prezenta

```
if ((!information.UtilizatorNotificat) ||  
    (information.UtilizatorNotificat &&  
    information.DataNotificarii.Value.ToShortDateString()  
    .Equals(DateTime.UtcNow.AddDays(-2).ToShortDateString())))
```

Se creează un corp al unui email cu ajutorul obiectului de tip *StringBuilder* disponibil in cadrul spațiului de nume *System.Text*, după care se trimite,împreună cu destinatarul(adresa de e-mail a utilizatorului) și un subiect la Clasa *EmailHelper* care este responsabilă cu trimiterea notificării.

```
public static void SendEmail(string destination, string subject, string body)  
{  
    lock (syncObject)  
    {  
        if (string.IsNullOrEmpty(destination))  
        {  
            return;  
        }  
  
        try  
        {  
  
            using (SmtpClient smtpClient = new SmtpClient("smtp.gmail.com"))  
            {  
                smtpClient.DeliveryMethod = SmtpDeliveryMethod.Network;  
                smtpClient.Port = 587;  
                smtpClient.EnableSsl = true;  
                smtpClient.UseDefaultCredentials = false;  
                smtpClient.Credentials = new NetworkCredential(@"notifications.prs.lic@gmail.com", "EBXmw63Sh7w6eUh");  
  
                using (MailMessage mail = new MailMessage())  
                {  
                    mail.BodyEncoding = Encoding.GetEncoding("utf-8");  
                    mail.From = new MailAddress(@"notifications.prs.lic@gmail.com");  
                    mail.To.Add(destination);  
                    mail.Subject = subject;  
                    mail.Body = body;  
                    mail.IsBodyHtml = true;  
  
                    smtpClient.Send(mail);  
                }  
            }  
        }  
        catch (Exception e)  
        {  
        }  
    }  
}
```

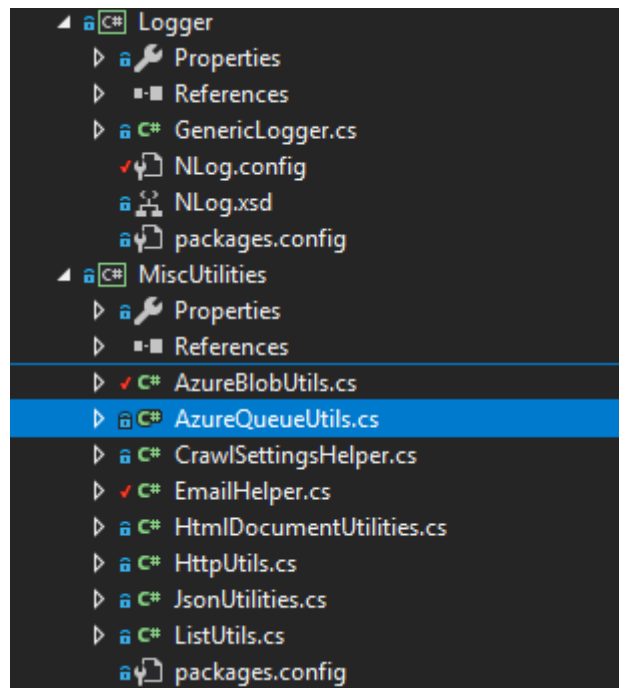
Folosind spațiile de nume *System.Net* și *System.Net.Mail* și adaugând setările necesare, putem trimite notificări prin Gmail, folosind protocolul *smtp* din interiorul aplicației.

Pentru a putea trimite o notificare a fost nevoie de generarea unei adrese noi de e-mail, [notification.prs.lic@gmail.com](mailto:notification.prs.lic@gmail.com)



#### 4.1.4. Ustensile generale

Introduse peste tot în acest proiect, sunt clasele ajutătoare, unele fiind statice, altele fiind *singleton*. Acestea sunt incluse în librăria create *MiscUtilities* fiind acum un spațiu de nume. Alături de aceasta, este și librăria care se ocupă doar de logarea informațiilor de avertizare și aparițiilor posibilelor erori.



#### Logarea informațiilor și a erorilor

```
using NLog;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Logger
{
    public static class GenericLogger
    {
        private static NLog.Logger log = LogManager.GetLogger("GenericLogger");

        public static void Error(string infoToLog)
        {
            log.Fatal($"Error occured : {infoToLog} ");
        }

        public static void Error(string infoToLog, Exception ex)
        {
            log.Fatal($"Error occured {ex.Message} \n {infoToLog} \n {ex.StackTrace}");
        }

        public static void Fatal(string infoToLog)
        {
            log.Fatal($"FatalError occured : {infoToLog} ");
        }

        public static void Fatal(string infoToLog, Exception ex)
        {
            log.Fatal($"FatalError occured {ex.Message} \n {infoToLog} \n {ex.StackTrace}");
        }

        public static void Info(string infoToLog)
        {
            log.Info(infoToLog);
        }
    }
}
```



Pentru crearea acestei librării a fost folosit spațiul de nume *NLog* disponibil prin intermediul pachetelor NuGet;

Avantajul acestei librării este că prin intermediul unui singur apel la una dintre metodele disponibile, informația este afișată atât în consolă, cât și în un fișier. Acest lucru a fost posibil prin configurarea librăriei *NLog*

```
<?xml version="1.0" encoding="utf-8" ?>
<nlog xmlns="http://www.nlog-project.org/schemas/NLog.xsd"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.nlog-project.org/schemas/NLog.xsd NLog.xsd"
      autoReload="true"
      throwExceptions="false"
      internalLogLevel="Off" internalLogFile="c:\temp\nlog-internal.log">
  <variable name="myvar" value="myvalue"/>
  <targets>
    <target xsi:type="ColoredConsole" name="ConsoleLogger"/>
    <target xsi:type="File" fileName="{basedir}/../LOGS/${shortdate} - logs.txt" name="FileLogger"/>
  </targets>
  <rules>
    <logger name="GenericLogger" writeTo="ConsoleLogger" />
    <logger name="GenericLogger" writeTo="FileLogger"/>
  </rules>
</nlog>
```



## 4.2 Interfața utilizatorului.

### 4.2.1. Înregistrare

Acasa Înregistrare Autentificare

## Inregistrare

Nume Utilizator :

Adresa email :

Numar Telefon :

Parola :

La fel ca orice aplicație web ce va interacționa cu un utilizator, aceasta are partea de înregistrare. Funcționalitatea a fost realizată cu ajutorul formelor web, fiind o modalitate, simplă și puternică pentru ceea ce este nevoie.

```
@using (Html.BeginForm())
{
    @Html.ValidationSummary("", "Datele introduse nu sunt corecte");
    <div>
        @Html.LabelFor(m => m.Nume_Utilizator)
    </div>
    <div>
        @Html.TextBoxFor(m => m.Nume_Utilizator)
    </div>
    <div>
        @Html.LabelFor(m => m.Email)
        @Html.ValidationMessageFor(m=>m.Email)
    </div>
    <div>
        @Html.TextBoxFor(m => m.Email)
    </div>
    <div>
        @Html.LabelFor(m => m.Numar_Telefon)
    </div>
    <div>
        @Html.TextBoxFor(m => m.Numar_Telefon)
    </div>
    <div>
        @Html.LabelFor(m => m.Parola)
    </div>
    <div>
        @Html.PasswordFor(m => m.Parola)
    </div>
    <input type="submit" value="Inregistraza-te"/>
}
```



```

public class UserModel
{
    public Guid id { get; set; }

    [Display(Name = "Nume Utilizator :")]
    public string Nume_Utilizator { get; set; }

    [Required]
    [DataType(DataType.Password)]
    [Display(Name = "Parola :")]
    public string Parola { get; set; }

    [Required]
    [EmailAddress]
    [Display(Name = "Adresa email : ")]
    public string Email { get; set; }

    [Display(Name = "Numar Telefon : ")]
    public string Numar_Telefon { get; set; }
}

```

Folosindu-ne de funcționalitățile formelor web, și de ustensilele acestora putem valida cu ușurință datele introduse:

```
@Html.ValidationSummary("", "Datele introduse nu sunt corecte");
```

Studiază corectitudinea datelor, bazându-se pe atributele setate în clasa *UserModel*, clasă folosită ca model pentru structurarea acestei forme de autentificare.

În același timp, din motive de securitate, parola va fi criptată folosindu-ne de o librărie dedicată, disponibilă în pachetele NuGet, numită *SimpleCrypto*.

```

using (var dbContext = new DbModelContext())
{
    var crypto = new SimpleCrypto.PBKDF2();

    var encPass = crypto.Compute(user.Parola);

    var tempUser = dbContext.Utilizator.Create();

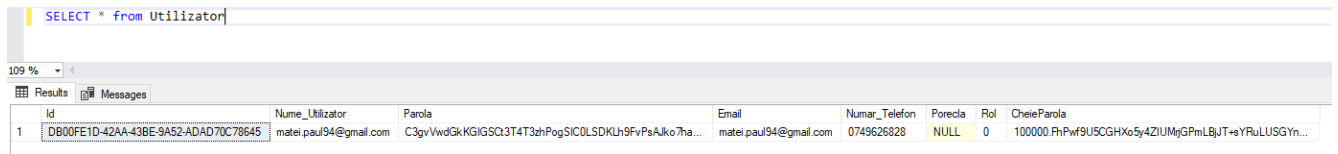
    tempUser.Id = Guid.NewGuid();
    tempUser.Numar_Telefon = user.Numar_Telefon;
    tempUser.Nume_Utilizator = user.Nume_Utilizator;
    tempUser.Parola = encPass;
    tempUser.Porecla = user.Porecla;
    tempUser.Rol = "0";
    tempUser.Email = user.Email;
    tempUser.CheieParola = crypto.Salt;

    dbContext.Utilizator.Add(tempUser);
    dbContext.SaveChanges();
}

```

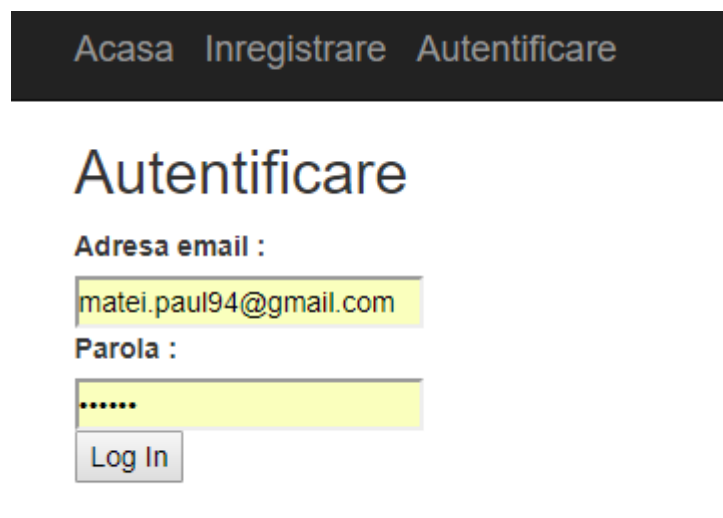


Conform imaginii, observăm ca la inserția utilizatorului în baza de date, vom salva atât parola criptată, cât și cheia de criptare. Aceasta, ne va ajuta în procesul de autentificare.



	Id	Nume_Utilizator	Parola	Email	Numar_Telefon	Porecla	Rol	CheieParola
1	DB00FE1D-42AA-43BE-9A52-ADAD70C78645	matei.paul94@gmail.com	C3gvVwdGkKGIGSQ3T4T3zhiPogSICOLSDKLh9FvPsAko7ha...	matei.paul94@gmail.com	0749626828	NULL	0	100000.FhPwf9U5CGHx05y4ZIUMyGPmLBjT+eYRuLUSGYn...

#### 4.2.2. Autentificare



Acasa Inregistrare Autentificare

## Autentificare

Adresa email :

matei.paul94@gmail.com

Parola :

.....

Log In

La fel ca și în cazul înregistrării, autentificarea a fost făcută cu ajutorul formelor web, permițând astfel, o verificare imediată a corectitudinii din punct de vedere structural al informației introduse. După ce informațiile au fost introduse direct din acest punct de vedere, se face validarea informației din punct de vedere al conținutului .

```
private bool IsValid(ref UserModel userModel)
{
    var crypto = new SimpleCrypto.PBKDF2();
    var isValid = false;

    var user = UserDbUtilities.GetUser(userModel.Email);

    if (user != null)
    {
        if (user.Parola == crypto.Compute(userModel.Parola, user.CheieParola))
        {
            isValid = true;
            userModel.id = user.Id;
        }
    }

    return isValid;
}
```



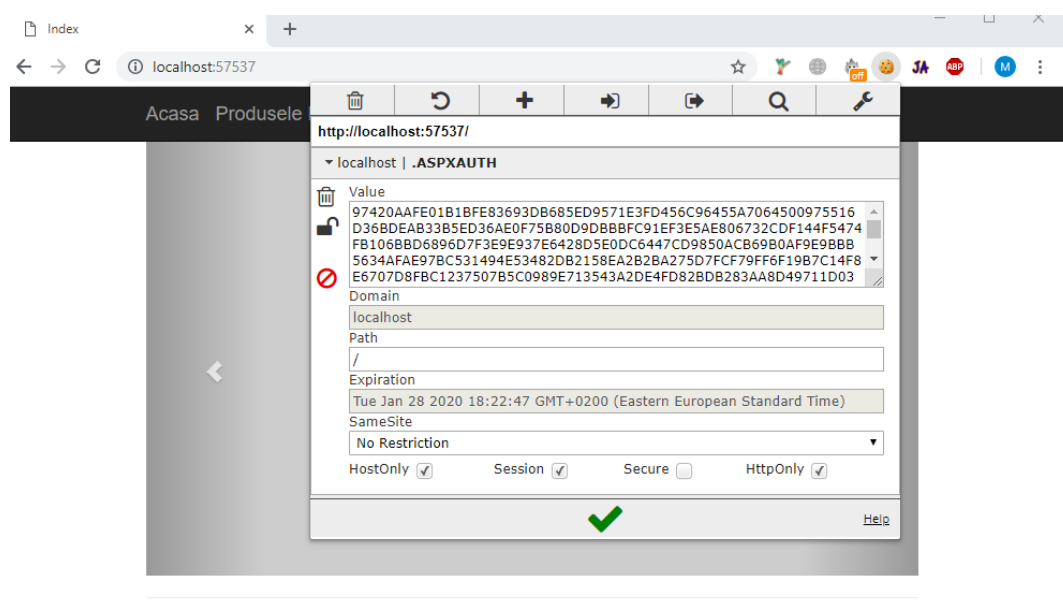
Folosind aceeași librărie de criptare, împreună cu cheia parolei, putem hotărî dacă datele de autentificare sunt valide.

```
public ActionResult Login(UserModel user)
{
    if (ModelState.IsValid)
    {
        if(IsValid(ref user))
        {
            ≤2,688ms elapsed
            string userData = string.Join("|",user.id, user.Email);

            FormsAuthenticationTicket ticket = new FormsAuthenticationTicket(
                1,
                user.Email,
                DateTime.Now,
                DateTime.Now.AddMinutes(10),
                false,
                userData,
                FormsAuthentication.FormsCookiePath);
            string encryptedTicket = FormsAuthentication.Encrypt(ticket);
            HttpCookie cookie = new HttpCookie(FormsAuthentication.FormsCookieName, encryptedTicket);
            cookie.HttpOnly = true;
            Response.Cookies.Add(cookie);

            return RedirectToAction("Index", "Home");
        }
        else{
            ModelState.AddModelError("", "Datele introduse nu sunt corecte");
        }
    }
    return View(user);
}
```

Dupa validare, se creaza un cookie ce se atribuie sesiunii. Acest cookie detine informațiile criptate despre utilizatorul curent autentificat.



### 4.2.3. Căutarea Produselor

## Cauta produse

 Search Filtru cautare : ☒ RELEVANTA ☐ PRET CRESCATOR ☐ PRET DESCRESCATOR

În această pagină utilizatorul va introduce un șir de caractere reprezentând denumirea unui produs pe care vrea să îl caute.

În același timp, i se oferă posibilitatea de a selecta modul de căutare ce va fi realizată:

- După relevanță
- După preț – Crescător
- După preț – Descrescător

După alegerea acestor detalii, următorii 2 pași sunt urmați:

#### 4.2.3.1. Inserția căutării în baza de date și în coada Azure

```
17 public static void NewSearchProductsInDatabase(string stringToSearch, string order)
18 {
19     // 1ms elapsed
20     string searchIdiomCode = GetSearchCode(stringToSearch);
21     InsertIdiomInDatabase(stringToSearch);
22     InsertCompleteIdiomInDatabase(stringToSearch, searchIdiomCode);
23     QueueUtilities.InsertIdiomInQueue($"{stringToSearch}|{order}");
24 }
25
```

100 %

Watch 1

Name	Value	Type
stringToSearch	"ASUS GEFORCE ROG STRIX RTX 2080TI "	string
order	"relevance"	string

Se inserează, pe rând fiecare element al căutării, urmat de șirul complet de caractere. În același timp, se generează un cod al căutării *searchIdiomCode*, care se adaugă în bază împreună cu mesajul căutat. Acest lucru a fost setat pentru a accelera viitoarele căutări ale produselor în baza de date. Coloana ce reține acest cod este indexată în baza de date, astfel căutările sunt mult mai rapide. Păstrând un istoric al căutărilor și al produselor generate de acea căutare, în momentul în care un alt utilizator caută aceleași cuvinte, pot fi și în ordine diferită, vizualizarea produselor va fi mult mai rapidă.





```

114 private static string GetSearchCode(string stringToSearch)
115 {
116     var pieces = stringToSearch.Trim().ToUpper().Split(' ');
117     int completeSearchCode = 1;
118     foreach (var piece in pieces)
119     {
120         try
121         {
122             completeSearchCode = completeSearchCode ^ piece.GetHashCode();
123         }
124         catch (Exception ex)
125         { }
126     }
127     return completeSearchCode.ToString();
128 }
129
130

```

Name	Value	Type
stringToSearch	"ASUS GEFORCE ROG STRIX RTX 2080TI "	string
order	"relevance"	string
GetSearchCode(stringToSearch)	"-1653614341"	string

```

Immediate Window
GetSearchCode("ASUS ROG GEFORCE RTX STRIX 2080TI")
"-1653614341"
GetSearchCode("GEFORCE ASUS 2080TI RTX ROG STRIX")
"-1653614341"

```

Putem observa in imaginea de mai sus, că nu contează în ce ordine sunt așezate cuvintele, codul căutării va fi mereu același.

```

GetSearchCode("ASUS ROG GEFORCE RTX STRIX 2080TI")
"-1653614341"
GetSearchCode("GEFORCE ASUS 2080TI RTX ROG STRIX")
"-1653614341"
GetSearchCode("ASUS GEFORCE ROG STRIX RTX 2080TI ")
"-1653614341"

```

După inserarea mesajului căutat, în baza de date, se trimite mesajul, alături de parametrul căutării in coada azure, de unde va fi preluat de Procesatorul de mesaje, despre care am discutat mai sus.

```

static CloudStorageAccount storageAccount = CloudStorageAccount.Parse(ConfigurationManager.ConnectionStrings["AzureConnectionString"].ConnectionString);

public static void InsertIdiomInQueue(string idiom)
{
    CloudQueueClient client = storageAccount.CreateCloudQueueClient();
    CloudQueue queue = client.GetQueueReference("search-queue");

    CloudQueueMessage message = new CloudQueueMessage(idiom);

    queue.AddMessage(message, TimeSpan.FromSeconds(10));
}

```

#### 4.2.3.2 Căutarea produselor în baza de date

Prin intermediul *EntityFramework* și cu ajutorul librăriei din spațiul de nume *System.Linq* se realizează căutarea produselor in baza de date, fiind filtrată de codul generat pe baza șirului de caractere introdus de utilizator.

Urmatoarea acțiune fiind ordonarea produselor in funcție de parametrul ales de utilizator. Această ordonare, se face folosind aceeași librărie *Linq* folosind una din metodele disponibile *OrderBy(func)* sau *OrderBYDescending(func)* . Cazul implicit al formulei switch fiind ordonarea dupa relevanță.



```

public static IEnumerable<Produs> SearchProductsInDatabase(string stringToSearch, string order)
{
    ≤6ms elapsed
    var products = new List<Produs>();

    using (DbModelContext dbContext = new DbModelContext())
    {
        string searchIdiomCode = GetSearchCode(stringToSearch);

        int attempts = 0;
        while (true)
        {
            Thread.Sleep(5 * 1000);
            products = (from appear in dbContext.AparitieProdus
                        join search in dbContext.IstoricCautari on appear.Id_Cautare equals search.Id_Cautare
                        join product in dbContext.Produs on appear.Id_Produs equals product.Id
                        join retailer in dbContext.Vanzator on product.Id_Vanzator equals retailer.Id
                        where search.Cod == searchIdiomCode
                        select product).Include(p => p.Vanzator).ToList();

            if (products.Count >= 1 || attempts > 30)
            {
                break;
            }
            Thread.Sleep(1000);
            attempts++;
        }
    }

    switch (order)
    {
        case "price-asc":
        {
            products = products.OrderBy(m => m.Pret).ToList();
            break;
        }


        case "price-desc":
        {
            products = products.OrderByDescending(m => m.Pret).ToList();
            break;
        }
    }
}

```

După ce acești doi pași au fost executați cu succes, produsele rezultate căutării sunt trimise către interfața utilizatorului.

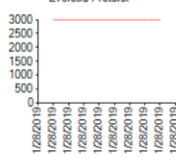
Acasa Produsele Mele Cautare Produse Delogeaza-te [matel.pau94@gmail.com](mailto:matel.pau94@gmail.com)

Laptop Gaming Asus TUF FX504GE Intel Core Coffee Lake (8th Gen) i5-8300H 1TB 8GB nVidia GeForce GTX 1050 Ti 2GB FullHD FX504GE-E4628



2999.00 lei

Evolutia Pretului




2999.00 lei

Cel



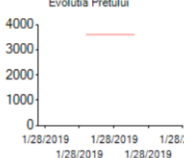
Urmareste Produs

Notebook / Laptop ASUS Gaming 15.6" TUF FX504GE FHD Processor Intel® Core™ i7-8750H (9M Cache up to 4.10 GHz) 8GB DDR4 1TB SSHD GeForce GTX 1050 Ti 4GB FreeDos Black



3598.99 lei

Evolutia Pretului



3598.99 lei

PcGarage



Urmareste Produs

Laptop Gaming Asus TUF FX504GE Intel Core Coffee Lake (8th Gen) i7-8750H 1TB 8GB nVidia GeForce GTX 1050 Ti 4GB FullHD fx504ge-e4059




2999.00 lei

Evolutia Pretului



2999.00 lei

Cel



Urmareste Produs



```

@foreach (var item in Model)
{
    <div class="product row" id="product-@item.Id">
        <div class="col-lg-5">
            <div class="product-details">
                <h4 class="product-title">@WebUtility.HtmlDecode(item.Denumire)</h4>
                <div class="product-image">
                    
                </div>
            </div>
        </div>
        <div class="col-lg-4">
            <div class="price-evolution">
                <span class="price-evolution">
                    
                </span>
            </div>
            <h3 class="product-price w-100" style="text-align: center;">@string.Format("{0} lei", item.Pret)</h3>
        </div>
        <div class="col-lg-3 pull-right">
            <div class="retailer">
                <div class="retailer-name">
                    <h4>@item.Vanzator.Nume</h4>
                </div>
                <div class="retailer-image">
                    
                </div>
                @using (Html.BeginForm("FollowProduct", "ProductSearch", new { @productID = item.Id }, FormMethod.Post))
                {
                    <input class="btn btn-info" type="submit" value="Urmaresti Produs" />
                }
            </div>
        </div>
    </div>
}

```

Pentru generarea graficului evoluției pretului a fost folosită o acțiune ce corespunde unei metode din controlor, trimittând ca parametru identificatorul produsului.

```

<span class="price-evolution">
    
</span>

```

```

public ActionResult LineChart(string productId)
{
    using (DbModelContext dbContext = new DbModelContext())
    {
        ArrayList xValues = new ArrayList();
        ArrayList yValues = new ArrayList();

        xValues.AddRange((from priceEvolution in dbContext.EvolutiaPretului where priceEvolution.Id_Produs == new Guid(productId)
            orderby priceEvolution.Data_Actualizare descending select priceEvolution.Data_Actualizare).Take(10).ToList());
        yValues.AddRange((from priceEvolution in dbContext.EvolutiaPretului where priceEvolution.Id_Produs == new Guid(productId)
            orderby priceEvolution.Data_Actualizare descending select priceEvolution.Pret).Take(10).ToList());

        new Chart(width: 200, height: 180, theme: ChartTheme.Vanilla).
            AddTitle("Evolutia Pretului").
            AddSeries(null, chartType: "Line", xValue: xValues, yValues: yValues).
            Write("bmp");
    }
    return null;
}

```

Pentru popularea graficului, datele au fost extrase din baza de date, fiind selectate doar ultimele 10 înregistrări, ordonate descendent după data la care au fost introduse.



#### 4.2.4. Setarea unei alerte

În dreptul fiecărui produs, sub logoul vânzătorului de care aparține este butonul *Urmareste Produs*.

Dacă un utilizator dorește să urmărească produsul respectiv, trebuie să apese pe acel buton, fiind redirecționat către pagina de urmărire produs.

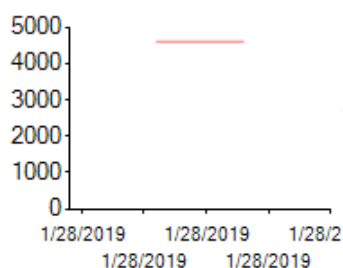
## Urmărire produs

Notebook / Laptop ASUS Gaming 15.6"  
TUF FX504GE FHD 120Hz Procesor Intel®  
Core™ i7-8750H (9M Cache up to 4.10  
GHz) 8GB DDR4 1TB 7200 RPM + 256GB  
SSD GeForce GTX 1050 Ti 4GB FreeDos  
Black



4598.99 lei

Evoluția Prețului



Seteaza Alerta

În spațiul liber, ar trebui introdusă limita prețului pentru care se va seta alerta.



```

public ActionResult SetAlert(string price, Guid productId)
{
    Guid userId = CookieUtilities.GetUserIdFromCookie(Request);

    ProductFollowUtilities.AddProductAlert(productId, userId, price);

    return RedirectToAction("MyProducts", "MyProducts");
}

```

La setarea alertei, se va adauga o nouă înregistrare in tabela *UrmarireProdus*. Pentru aceasta operație, avem nevoie de identificatorul utilizatorului, cel al produsului si prețul impus.

Identificatorul produsului împreună cu pretul îl vom primi ca parametri din acțiunea declanșată, iar identificatorul utilizatorului va fi extras din cookie-ul setat inițial în pricesul de autentificare.

```

internal static void AddProductAlert(Guid productId, Guid UserId, string price)
{
    using (var dbContext = new DbModelContext())
    {
        try
        {
            UrmarireProdus tempAlert = new UrmarireProdus();

            tempAlert.Id = Guid.NewGuid();
            tempAlert.Id_Produs = productId;
            tempAlert.Id_Utilizator = UserId;
            tempAlert.Limita_pret = decimal.Parse(price);
            tempAlert.Invalid = false;
            tempAlert.UtilizatorNotificat = false;
            tempAlert.DataNotificarii = DateTime.UtcNow;

            dbContext.UrmarireProdus.Add(tempAlert);
            dbContext.SaveChanges();
        }
        catch (Exception ex)
        {
        }
    }
}

```

La finalizarea acestor pași utilizatorul va fi redirecționat catre pagina cu produsele urmarite de el.




#### 4.2.5. Produse Urmărite

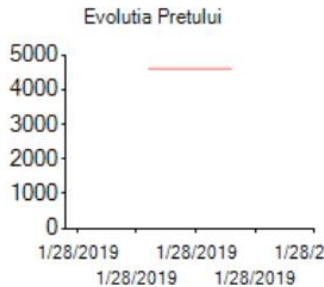
Această pagină conține toate produsele urmărite de utilizatorul curent.

Acasa Produsele Mele Cautare Produse Delogheaza-te matei.paul94@gmail.com

Notebook / Laptop ASUS Gaming 15.6" TUF FX504GE  
FHD 120Hz Procesor Intel® Core™ i7-8750H (9M Cache  
up to 4.10 GHz) 8GB DDR4 1TB 7200 RPM + 256GB  
SSD GeForce GTX 1050 Ti 4GB FreeDos Black




Evolutia Pretului



4598.99 lei

PcGarage

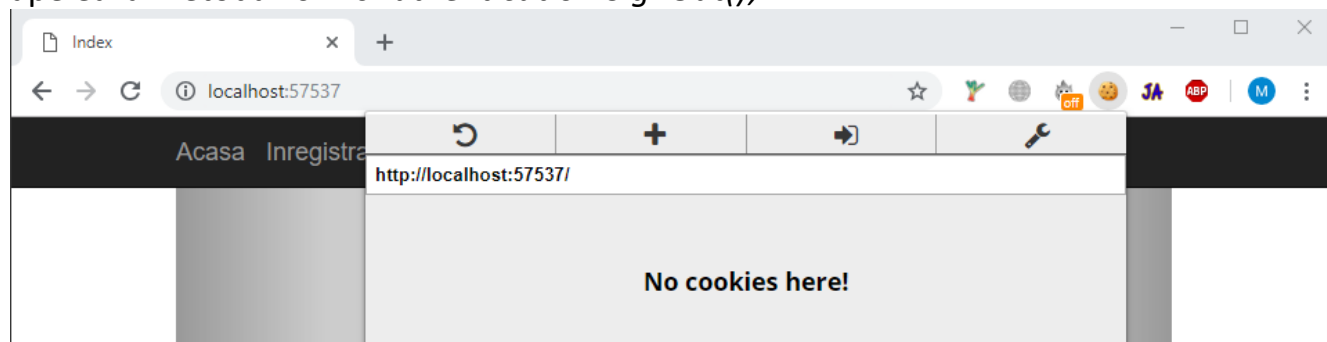


Sterge Alerta

Utilizatorul are opțiunea de a șterge o alertă, prin intermediul butonului *Sterge Alerta*, astfel declanșându-se o acțiune, ce va marca înregistrarea din baza de date ca fiind invalidă. După ce acest lucru se întâmplă, fereastra va fi reîmprospătată astfel fiind afișate doar produsele urmărite, rămase.

#### 4.2.6. Delogarea

În momentul apăsării butonului *Delogheaza-te* utilizatorul este redirecționat către pagina de început, se șterge cookie-ul ce menține informația despre utilizator și se apelează metoda *FormsAuthentication.SignOut()*;



## 5.Posibile extinderi și concluzii

În procesul dezvoltării aplicației prezentate am învățat cum este posibilă crearea mai multor pagini web folosind arhitectura MVC și generarea de legături între acestea, având la dispoziție multitudinea de avantaje ce care arhitectura le oferă. Împreună entity framework si mediile de stocare in cloud(azure) formează un mediu perfect si puternic de dezvoltare.

Fiind prima aplicație de acest gen și în același timp un proiect relativ mare, problemele au fost în mare parte de natură arhitecturală, dar am reușit să le rezolv prin studiu, documentație și experiența celor mai avansați, prin intermediul platformelor dedicate unde se puteau găsi rezolvări ale problemelor si răspunsuri la întrebările mele. O altă problemă ce poate interveni este ridicată din cauza dependenței directe față de site-urile vânzătorilor, dacă ceva se modifică acolo, trebuie modificat și în aplicație.

În orice proiect pot fi aduse o gamă variată de îmbunătățiri și extinderi, printre astea se enumeră:

- acceptarea mai multor vânzători
- un sistem de urmărire (când un utilizator se hotărăște că va cumpăra un produs prin fiind notificat de această aplicație) pentru a calcula eficiența sistemelor de notificare.
- o modalitate de plată.



## Bibliografie

- [https://en.wikipedia.org/wiki/Online\\_shopping#History](https://en.wikipedia.org/wiki/Online_shopping#History)
- <http://www.cs.ubbcluj.ro/~vcioban/Matematica/Anul3/BD/Bd.pdf>
- <http://vega.unitbv.ro/~cataron/Courses/BD/>
- [https://en.wikipedia.org/wiki/Microsoft\\_SQL\\_Server](https://en.wikipedia.org/wiki/Microsoft_SQL_Server)
- <https://ro.wikipedia.org/wiki/SQL>
- <https://docs.microsoft.com/en-us/azure/storage/queues/>
- <https://docs.microsoft.com/en-us/azure/storage/common/storage-introduction#blob-storage>
- <https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/>
- C# 6.0 and the .NET 4.6 Framework, Seventh Edition. Andrew Troelsen si Philip Japikse. Editura Apress
- <https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller#Components>
- [https://profs.info.uaic.ro/~iasimin/Special/Curs\\_EntityFramework.pdf](https://profs.info.uaic.ro/~iasimin/Special/Curs_EntityFramework.pdf)
- <https://docs.microsoft.com/en-us/dotnet/api/system.data.entity?view=entity-framework-6.2.0>
- <http://www.entityframeworktutorial.net/Querying-with-EDM.aspx>
- <https://app.pluralsight.com/player?name=aspdotnet-mvc5-fundamentals-m6-ef6&mode=live&clip=0&course=aspdotnet-mvc5-fundamentals&author=scott-allen>
- <https://app.pluralsight.com/library/courses/c-sharp-fundamentals-with-visual-studio-2015/>
- <https://app.pluralsight.com/library/courses/entity-framework-6-getting-started>
- 

