Ministerul Educației și Cercetării al Republicii Moldova

Universitatea Tehnică a Moldovei

Facultatea Calculatoare, Informatică și Microelectronică

Departamentul Ingineria Software și Automatică

**Proiect de curs**

*Disciplina: Analiza și Proiectarea Algoritmilor*

**Tema: Elaborarea unei aplicații software destinată gestionării resurselor umane.**

**Developing a software application intended for human resources management.**

Au efectuat: Dogotari Ianec, TI-216

Roșca Dorin, TI-216

Panco Daniel, TI-216

A verificat: asist. univ. Cebotari Daria

Chișinău - 2023

**Cuprins**

[Introducere 2](#_Toc1071212294)

[1.Analiza domeniului de studiu 3](#_Toc902330523)

[1.1 Formularea problemei prin abstractizarea unei situaţii 4](#_Toc1158375749)

[1.2 Definirea datelor problemei şi stabilirea relaţiei între date şi soluţia problemei 5](#_Toc2014592310)

[2. Algoritmul sistemului 6](#_Toc1769994521)

[2.1 Alegerea algoritmilor pentru rezolvarea problemei 7](#_Toc1919619458)

[2.2 Argumentarea alegerii algoritmilor 11](#_Toc1048387168)

[3. Realizarea sistemului 12](#_Toc175606404)

[3.1 Implementarea algoritmilor 16](#_Toc1142411435)

[3.2 Interfeţa grafica a aplicatiei 17](#_Toc151995477)

[3.3 Analiza rezultatelor obtinute 26](#_Toc334172511)

[Concluzii 28](#_Toc1580164478)

[Bibliografie 29](#_Toc1396403689)

[Anexa A 30](#_Toc1158139380)

# Introducere

În cadrul oricărei organizații moderne un rol important îl are domeniul Resurselor Umane și gestionarea acestora. Managementul resurselor umane (MRU sau RU) reprezintă o abordare strategică a gestionării eficiente a persoanelor dintr-o companie sau organizație, astfel încât acestea să-și ajute afacerea să obțină un avantaj competitiv. Acest concept este proiectat pentru a spori performanța angajaților din punct de vedere al obiectivelor strategice ale angajatorului.**[1]** Managementul resurselor umane se referă în primul rând la gestionarea persoanelor din cadrul organizațiilor, concentrându-se pe politici și sisteme. Departamentele de resurse umane sunt responsabile de supervizarea planului de beneficii ale angajaților, recrutării angajaților, instruirii și dezvoltării, evaluării performanțelor și gestionării recompenselor, cum ar fi gestionarea sistemelor de plăți și sistemelor de beneficii. HR(în engleză) se preocupă, de asemenea de schimbările organizaționale și de relațiile industriale sau de echilibrarea practicilor organizaționale cu cerințele care decurg din negocierile colective și din legile guvernamentale.

Lucrarea dată urmărește scopul de a elabora o aplicație pentru a gestiona resursele umane cu ajutorul programăriiorientate pe obiecte și a bazelor de date SQL.

Lucrarea este structurată în trei capitole, unde se va analiza concret modul de proiectare și de realizare a poiectului dat, fiind incluse informații generale despre temă și domeniul pe care îl cuprinde, explicarea algoritmilor aleși pentru a găsi o soluție și însuși codul elaborat de noi.

Primul dintre aceste capitole vine să familiarizeze cu problema aleasă de noi. Aici va fi formulată mai detaliat tema proiectului nostru și vor fi abordate diferite aspecte ale domeniului HR, cum ar trebui să arate o aplicație de gestiune a resurselor umane, funcțiile principale pe care ar trebui să le îndeplinească, avantajele utilizării unei astfel de aplicații și multe altele. De asemenea va fi formulată problema prin abstractizarea unei situații, vor fi definite datele problemei și va fi stabilită relația dintre date și soluția problemei.

Al doilea capitol se bazează mai mult pe algoritmii aleși pentru rezolvarea problemei. Alegerea dată va fi argumentată într-un mod convingător. În acest capitol de asemenea vor fi analizați diferiți algoritmi, dintre care vom alege unul singur care se potriveste problemei noastre. Vom arăta atât avantajele cât și dezavantajele algoritmului nostru.

În ultimul capitol se cuprinde partea de implementarea a codului. Aici este toată despre realizarea sistemului, toate tehnologiile folosite(limbaj/ framework/librarii). Se va aduce la cunoștință cum a fost implementat algoritmul cu explicări bazate pe cod. Interfața grafică a sistemului la fel intră în acest capitol, aici fiind atașate imagini cu funcționarea sistemului și toate oparețiile posibile. Spre sfârșit, drept generalizare se vor analiza rezultatele obținute și se vor face concluzii pe baza lor.

# 1. Analiza domeniului de studiu

Un algoritm este o secvență de instrucțiuni simple care se ocupă cu rezolvarea unor operații. Aceste operații pot avea caracter aritmetic, logic sau mixt și prelucrează datele de intrare producând date de ieșire.

Pentru ca un algoritm să fie eficient, acesta trebuie să furnizeze rezultatul într-un timp finit și cât mai scurt. Pe langă timpul furnizării rezultatului, un algoritm trebuie să fie optimizat în așa fel incat să ocupe o memorie cât mai mică față de alți algoritmi care furnizează același rezultat, dar care ocupă o memorie mai mare.

Un algoritm trebuie să fie de folos la rezolvarea a cât mai multor probleme, spre exemplu, unul care poate rezolva o singură problema nu este atât de relevant în comparație cu unul care poate rezolva un n număr de probleme. Un exemplu banal, însă relevant, ar fi atunci când creăm un program care adună două numere. Dacă programul adună doar numerele 5 și 2, atunci acesta nu va fi atât de folositor față de unul care adună orice două numere (a+b). Asta înseamnă că nu trebuie să atribuim noi valori în interiorul programului, ci să elaborăm o funcție de citire de la tastatură a acestora.

Algoritmul trebuie să aibă un caracter determinist. Aceasta ar înseamna că prin repetarea programului introducând aceleași valori, rezultatul final va fi mereu același.

Pentru ca un algoritm sa poată fi folosit în condiții optime, acesta trebuie sa conțină indicații clare pentru ca utilizatorul să poată să introducă datele necesare fiecarei etape a programului. Atunci când utilizatorul trebuie să introducă anumite date de la tastatură, programul trebuie să îi furnizeze clar ce fel de date sunt necesare respectivei etape.

Tipuri de algoritmi:

* Algoritmi back-tracking(incrementează soluții-candidat, părăsind fiecare candidat imediat ce acesta nu mai poate deveni o soluție validă);
* Algoritmi devide et impera(descompune problema în mai multe subprobleme pe care la final le combină);
* Algoritmi greedy(face la nivel local alegerea optimă pentru fiecare etapăîn speranța de a găsi un optim global).

Proiectarea algoritmului:

* Cuprinde un set de instrucțiuni pentru completarea unei sarcini;
* Deplasează problema de la faza de modelare la etapa de operare;
* Setul de instrucțiuni trebuie să fie secvențial, complet, precis și să aibă un punct final.

Atât în problema aleasă de noi cât și în altele este necesar și important să utilizăm un algoritmul potrivit întrucât aceasta face programul mult mai logic, mai ușor de înțeles/utilizat/redactat și mai estetic din punct de vedere al principiilor programării.

## **Formularea problemei prin abstractizarea unei situaţii**

Problema aleasă de noi este crearea unei aplicații destinată gestionării resurselor umane. HR este domeniul responsabil de angajații unei companii, respectiv scopul nostru este de a implementa algoritmii necesari creării unui HRMS(Human Resources Managing System) care ar efectua operațiile necesare pentru a gestiona un număr anumit de angajați ai unei companii, la fel si alte operații suplimentare.

Afacerile de succes nu au timp să-și petreacă manual îndeplinind sarcini repetitive de HR. Pe măsură ce compania ta crește, trebuie să cauți soluții care te vor ajuta să depășești provocările de HR care te țin pe loc. Aici intervine software-ul sistemului de management al resurselor umane (HRMS). Un sistem de management al resurselor umane (HRMS) este un software creat pentru ca echipele de HR să gestioneze cu ușurință sarcinile de HR și procesele companiei. Sistemele de management al resurselor umane sunt utilizate de companii pentru a eficientiza multe sisteme și funcții de HR, cum ar fi stocarea datelor angajaților, gestionarea orelor și salariilor angajaților, recrutarea, managementul performanței angajaților și înregistrările de instruire.

Investiția în software-ul HRMS permite companiilor să eficientizeze și să automatizeze sarcinile consumatoare de timp și obositoare. Sistemele de management al resurselor umane oferă perspective mai profunde asupra companiei, îmbunătățesc implicarea angajaților, cresc eficiența și eliberează personalul de HR pentru a se concentra pe proiecte cu valoare mai mare. Software-ul HRMS poate fi extrem de divers, cu diferite capacități, avantaje și dezavantaje.

Un sistem de gestionare HR ar trebui să fie cât mai clar elaborat și ușor de utilizat.

Funcțiile unui sistem HR:

* Recrutare și selecție;
* Formare și dezvoltare;
* Managementul performanței angajaților;
* Relațiile cu angajații;
* Dreptul muncii și conformitatea;
* Compensații, beneficii și administrare;
* Sisteme de salarizare.**[2]**

## **1.2 Definirea datelor problemei şi stabilirea relaţiei între date şi soluţia problemei**

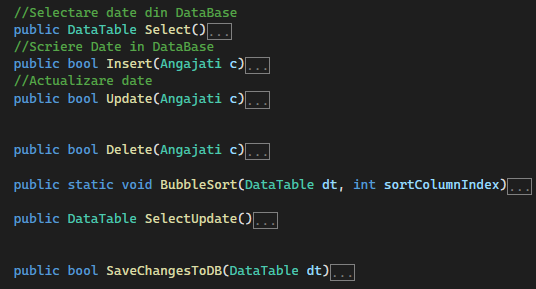
Obiectivele care ne le-am propus prin alegerea temei date sunt:

* Realizarea unei aplicații de management al resurselor umane utilizând cunoștințele posedate la momentul dat;
* Implementarea algoritmilor potriviți pentru soluționarea problemei;
* Compararea algoritmilor aleși cu alți algoritmi și argumentarea alegerii noastre prin exemple concluzive;
* Utilizarea programării orientate pe obiecte, pentru a da un aspect cât mai logic programului nostru;
* Realizarea unei interfețe grafice pentru programul nostru;
* Analizarea rezultatelor obținute în urma soluționării problemei alese.

Cum am menționat mai sus unul dintre obiectivele noastre este să utilizăm programarea orientată pe obiecte, ceea ce ar presupune crearea unei clase Angajat care ar conține toate caracteristicile unui angajat, cum ar fi:

* ID-ul;
* Numele;
* Vârsta;
* Funcția;
* Salariul;
* Domiciliul;
* Emailul;
* Limba vorbită.

Aplicația noastră va da utilizatorului următoarele posibilități: introducerea, modificarea, căutarea, sortarea după un anumit parametru și ștergerea informației despre angajați.

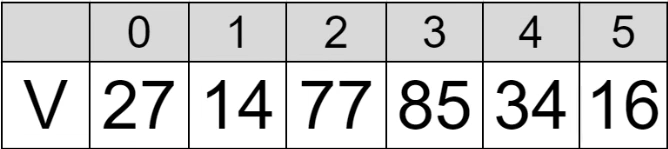


**Fig. 1 Funcțiile realizate**

În Fig. 1 putem observa funcțiile realizate în cadrul proiectului. Folosim aceste funcții pentru a faciliza interacțiunea dintre baza de date și aplicație.

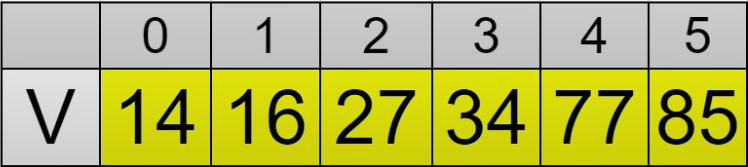
# 2. Algoritmul sistemului

Principalul algoritm al sistemului nostru va fi un algoritm de sortare. In informatica, „a sorta” inseamna a rearanja elementele dintr-o structura de date in ordine crescatoare (sau descrescatoare). Spre exemplu, avem vectorul din Fig. 2.1.



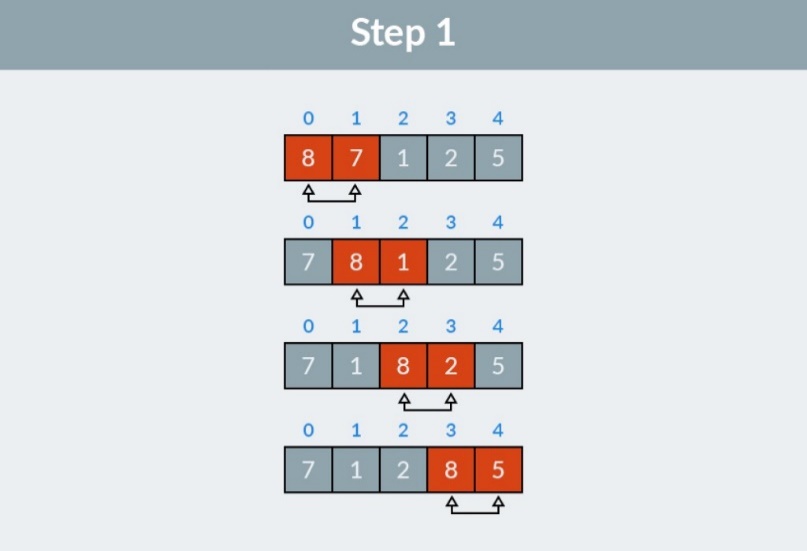
**Fig. 2.1 Vector înaintea sortării**

Dupa ce vom rearanja elementele vom obtine:



**Fig. 2.2 Vector în urma sortării**

Sortarea este o operatie fundamentala in informatica. Putem sorta datele pentru prezentarea catre utilizator (de exemplu in agenda telefonica, sau atunci cand navigam printre foldere, sau selectam o anumita melodie din telefon). Sortarea este o piatra fundamentala in algoritmii mai mari, pentru ca simplifica anumite cerinte (precum gasirea unei chei unice, sau detectarea elementelor duplicate).



**Fig. 2.3 Reprezentare a procesului sortării.**

În fig. 2.3 putem observa cum decurge algoritmul Bubblesort, un algoritm destul de simplu dar nu atât de eficient din punct de vedere a complexității.

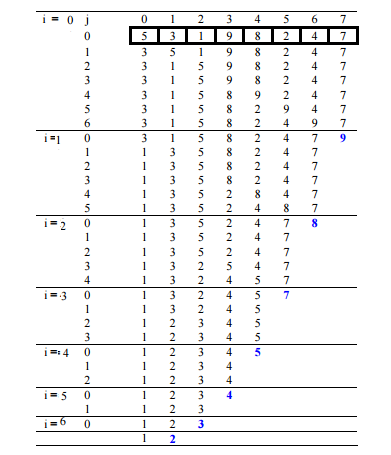
Putem sorta orice, numere, texte, orice care are un criteriu de sortare. Putem sorta un vector de obiecte după un anumit câmp al acestui obiect, spre exemplu sortarea după salariul primit. De asemenea putem sorta în dependență de numele sau prenumele unui angajat în ordine alfabetică. Așadar există destul de multe posibilități. Întrucât scopul nostru este realizarea unui HRMS(sistem de gestionare a resurselor umane), sortarea este un element esențial.

## **2.1 Alegerea algoritmilor pentru rezolvarea problemei**

În acest subcapitol vom urmări descrierea mai multor algoritmi de sortare și vom alege unul pentru soluționarea problemei propuse.

Cu am mai spus anterior, un algoritm de sortare este utilizat pentru a rearanja o anumită matrice sau o listă de elemente în funcție de un operator de comparare a elementelor. Operatorul de comparație este utilizat pentru a decide noua ordine a elementelor din structura de date respectivă.**[3]**

**Bubble Sort** (Fig. 2.4) este cel mai simplu algoritm de sortare care funcționează prin schimbarea în mod repetat a elementelor adiacente dacă acestea sunt în ordinea greșită. Acest algoritm nu este foarte potrivit pentru seturi mari de date, deoarece complexitatea sa medie și în cel mai rău caz este destul de mare.



**Fig. 2.4 Ilustrare a algoritmului Bubble Sort**

Datorită simplității sale, sortarea cu bule este adesea folosită pentru a introduce conceptul de algoritm de sortare.**[4]**

În grafica computerizată, este popular pentru capacitatea sa de a detecta o eroare mică (cum ar fi un schimb de doar două elemente) în matrice aproape sortate și de a o remedia doar cu o complexitate liniară (2n).

Sortarea cu bule durează minim (Ordinea lui n) când elementele sunt deja sortate. Prin urmare, cel mai bine este să verificăm dacă tabloul este deja sortat sau nu în prealabil, pentru a evita complexitatea timpului O( ).

Algoritmul **Insertion Sort** (Fig. 2.5)este un algoritm puțin mai complicat decât Bubble sort, dar mai efectiv. Aceasta se datorează faptului că odată cu fiecare iterație un element nesortat este aranjat la poziția lui potrivită.

Acest algoritm funcționează în mod similar sortarea cărților într-un joc de cărți. Presupunem că prima carte este deja sortată atunci, selectăm o carte nesortată. Dacă cartea nesortată este mai mare decât cartea în mână, aceasta este plasată la dreapta altfel, la stânga. În același mod, alte cărți nesortate sunt luate și puse în locul lor potrivit. O abordare similară este utilizată de Insertion Sort.

Sortarea prin inserție este utilizată atunci când numărul de elemente este mic. De asemenea, poate fi util atunci când matricea de intrare este aproape sortată, doar câteva elemente sunt deplasate greșit în matricea mare completă.



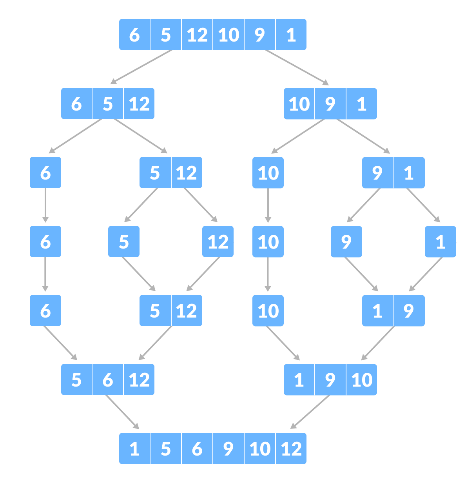
**Fig. 2.5 Reprezentare a algoritmului Insertion Sort [5]**

Algoritmul **Merge Sort** (Fig. 2.6) este un algoritm de sortare care funcționează prin împărțirea unui vector în subdiviziuni mai mici, sortând fiecare subdiviziune și apoi îmbinându-le înapoi împreună pentru a forma vectorul sortat final.

În termeni simpli, putem spune că procesul Merge Sort este de a împărți vectorul în două jumătăți, de a sorta fiecare jumătate și apoi de a îmbina din nou jumătățile sortate. Acest proces se repetă până când întregul vector este sortat.

Un lucru pe care s-ar putea să vă întrebați este care este specialitatea acestui algoritm. Avem deja o serie de algoritmi de sortare, atunci de ce avem nevoie de acest algoritm? Unul dintre principalele avantaje ale sortării prin îmbinare este că are o complexitate de timp de O(n log n), ceea ce înseamnă că poate sorta matrice mari relativ rapid. Este, de asemenea, o sortare stabilă, ceea ce înseamnă că ordinea elementelor cu valori egale este păstrată în timpul sortării.

Sortarea prin îmbinare este o alegere populară pentru sortarea seturi de date mari, deoarece este relativ eficientă și ușor de implementat. Este adesea folosit împreună cu alți algoritmi, cum ar fi sortarea rapidă, pentru a îmbunătăți performanța generală a unei rutine de sortare.



**Fig. 2.6 Reprezentare a algoritmului Merge Sort [6]**

Un alt algoritm de sortare este algoritmul **QuickSort** (Fig. 2.7), care la fel ca și algoritmul precedent este un algoritm Devide et Impera. Acest algoritm se bazează pe alegerea unui element ca pivot și împărțirea tabloului în două subtablouri care la rândul lor constau din: elementele mai mici decât pivotul și respectiv cele mai mari ca pivotul.

Există mai multe versiuni de a alege pivotul:

* Primul element;
* Ultimul element (implementat în Fig. 2.7);
* Element aleatoriu;
* Mijlocul.



**Fig. 2.7 Ilustrare a algoritmului QuickSort [7]**

## 

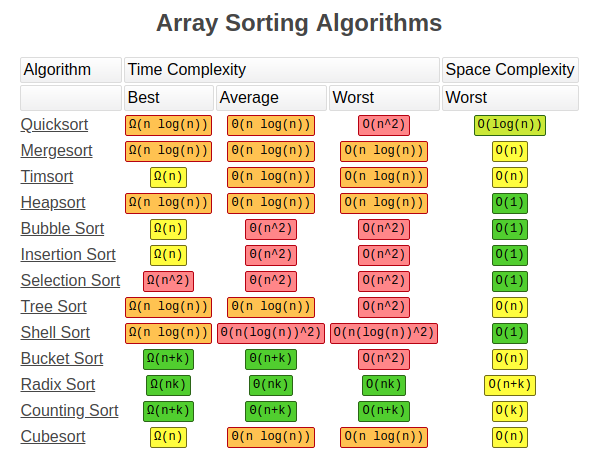
## **2.2 Argumentarea alegerii algoritmilor**

În urma analizării mai multor algoritmi de sortare în capitolul precedent, putem trece la alegerea algoritmului de sortare potrivit problemei noatstre.

În Fig. 2.8 putem analiza complexitatea mai multor algoritmi pentru cel mai favorabil, mediu și cel mai defavorabil caz.

Putem observa că Bubble sort are o complexitate mare, motiv pentru care acesta se utilizează pentru soluționarea problemelor mai simple cu o dimensiune mai mică la fel ca și algoritmul Selection Sort care are complexitatea O(n^2) pentru toate cazurile.

Posibil o alegere bună ar fi algoritmul QuickSort, dar în cel mai defavorabil caz acesta are o complexitate de O(n^2). O alternativă mai bună ar fi algoritmii Merge Sort sau Heap Sort. Putem observa că algoritmul Heap sort are o complexitate a spațiului mai mică ca cea a algoritmului Merge Sort, însă asta nu înseamnă neapărat că este mai stabil.



**Fig. 2.8 Complexitatea algoritmilor**

Întrucât am realizat aceasta aplicație de gestiune a resurselor umane în scop demonstrativ am ales un algoritm simplu de implementat și pe înțelesul tuturor. Algoritmul ales este **Bubble sort**, însă cu unele îmbunătățiri subtile pentru a optimiza într-o oarecare măsură complexitatea acestuia.

Unele avantaje ale acestui algoritm sunt:

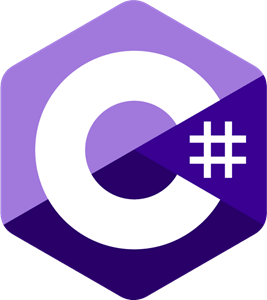
* Acesta este simplu de ințeles și ușor de implementat;
* Nu necesită spațiu adițional în memorie;
* Este adaptabil la diferite tipuri de date.

Dezavantajele sunt:

* Complexitatea mare în cazul mediu și cel mai defavorabil, O();
* Nu este potrivit pentru seturi mari de date întrucât are o viteză mică;

# 3. Realizarea sistemului

În cadrul realizării codului pentru aplicația noastră de gestine a resurselor umane am decis să utilizăm limbajul C#. IDE-ul utilizat este Microsoft Visual Studio. Am făcut alegerea dată întrucât limbajul dat este un succesor al limbajului C++, deja studiat în cadrul activităților universitare, astfel fiind mai familiar pentru noi.



**Fig. 3.1 Logo-ul limbajului C#**

C# (Fig. 3.1) este un limbaj de pentru programare orientată pe obiect, care este utilizat în principal pentru dezvoltarea de aplicații desktop, aplicații mobile și aplicații bazate pe web, pentru Windows și alte produse Microsoft. Acest limbaj a fost conceput ca un concurent pentru limbajul Java. Ca și acesta, C# este un derivat al limbajului de programare C++.

Prin utilizarea framework-ului .NET, poți dezvolta aproape orice pentru Microsoft cu C#. Deși framework-ul .NET suportă o mare varietate de limbaje de programare, precum VB.NET, C++ și F#, C# este cel mai popular din această listă.



**Fig. 3.2 Logo-ul .NET Framework**

Unul dintre beneficiile principale ale alegerii limbajului de programare C# este faptul că poate folosi librăria uriașă de instrumente pentru dezvoltare web de la Microsoft. Prin intermediul lui ASP.NET (o parte a framework-ului .NET), poți dezvolta site-uri web, pagini web, și alte aplicații web cu C#. Deși C# și .NET (Fig. 3.2) tind să meargă mână în mână, nu este necesar să folosim framework-ul pentru a programa cuC#. Pur și simplu merg bine împreună deoarece au fost dezvoltate de către Microsoft.

using System;

using System.Collections;

using System.Collections.Generic;

using System.Configuration;

using System.Data;

using System.Data.SqlClient;

using System.Diagnostics.Eventing.Reader;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Xml;

Pentru a lucra cu .NET Framework am utilizat o varietate de namespace-uri arătate mai sus. În general putem observa că toate încep cu cuvântul **System**. Namespace-ul System este o colecție de clase și interfețe care oferă funcționalitate fundamentală pentru diferite aspecte ale.NET Framework. Acesta include clase pentru tipurile de date de bază, structuri de date comune, manipularea fișierelor și a directorului, tratarea excepțiilor, reflecție și multe altele. Unele dintre clasele cele mai frecvent utilizate în spațiul de nume de sistem includ Console, String, Array, și Math.

Namespace-ul .Data oferă un set de clase pentru lucrul cu datele din.NET Framework. Acesta include clase pentru conectarea la o bază de date, executarea comenzilor și accesarea datelor. Unele dintre clasele principale din acest spațiu de nume includ:

* SqlConnection: O clasă pentru conectarea la o bază de date;
* SQL Server SqlCommand: O clasă pentru executarea comenzilor pe o bază de date SQL Server;
* SqlDataReader: O clasă pentru citirea datelor dintr-o bază de date SQL Server;
* DataSet: O clasă pentru stocarea și manipularea datelor în memorie;
* DataTable: O clasă pentru stocarea și manipularea datelor în memorie într-un format tabelar;
* DataAdapter: O clasă pentru completarea unui set de date sau DataTable cu date dintr-o bază de date.**[8]**

System.Configuration este un snamespace în .NET Framework care conține clase pentru gestionarea setărilor de configurare a aplicației. Aceste clase permit dezvoltatorilor să citească și să scrie cu ușurință date de configurare din diferite surse, cum ar fi fișierele app.config și web.config. Clasele cele mai frecvent utilizate în acest spațiu de nume includ ConfigurationManager, care oferă acces la setările aplicației și Configurare, care oferă acces la setările de configurare ale unei anumite aplicații sau asamblare.

Windows Forms este o bibliotecă de clase Graphical User Interface (GUI) care este inclusă în .Net Framework . Scopul său principal este de a oferi o interfață mai ușoară pentru dezvoltarea aplicațiilor pentru desktop, tabletă, PC-uri. Este, de asemenea, numit WinForms . Aplicațiile care sunt dezvoltate utilizând Windows Forms sau WinForms sunt cunoscute sub numele de Aplicații Windows Forms care rulează pe computerul desktop. WinForms poate fi folosit numai pentru a dezvolta aplicații Windows Forms, nu aplicații web. Aplicațiile date pot conține diferite tipuri de controale, cum ar fi etichete, casete cu listă, indicații etc.

De asemenea un rol important în realizarea aplicației noastre l-au avut bazele de date. O bază de date SQL (Structured Query Language) este un tip de bază de date care utilizează SQL pentru a gestiona și manipula datele stocate în ea. SQL este un limbaj de programare care este utilizat pentru a crea, modifica și interoga baze de date. Bazele de date SQL sunt utilizate în mod obișnuit în aplicațiile care necesită accesare și manipulare rapidă a datelor. Exemple de baze de date SQL includ MySQL, Oracle și Microsoft SQL Server. Acestea sunt adesea utilizate în aplicații de întreprindere și de afaceri, cum ar fi contabilitatea, gestionarea inventarului și gestionarea relațiilor cu clienții, deasemenea și în domeniul ales de noi.

class Angajati

{

public int ID { get; set; }

public string Nume { get; set; }

public int Varsta { get; set; }

public string Functia { get; set; }

public float Salariu { get; set; }

public string Domiciliu { get; set; }

public string Email { get; set; }

public string Limba { get; set; }

static string myconnstrng = ConfigurationManager.ConnectionStrings["connstrng"].ConnectionString;

...

}

Mai sus este reprezentată clasa de bază creată de noi și toate câmpurile ce caracterizează un angajat: ID, Nume, Varsta etc. Variabila **myconnstrng** reprezintă conexiunea dintre baza de date și program. În urma creării Serverului SQL, am creat baza de date, iar la urmă am conectat baza de date cu proiectul prin intermediul **connstrng.**

Metodele acestei clase sunt:

* Select: Selectarea datelor și afișarea lor în tabel;
* Insert: Adăugarea datelor despre un angajat;
* Update: Modificarea datelor;
* Delete: Eliminarea datelor despre un angajat;
* BubbleSort: Sortarea datelor despre angajați în ordine crescătoare conform ID-ului;
* SelectUpdate: Selectarea datelor(utilizată la sortare);
* SaveChangesToDB: Salvarea modificărilor în baza de date;

## **3.1 Implementarea algoritmilor**

În acest capitol vom aborda implementarea algoritmilor aleși pentru realizarea proiectului. Vom vorbi despre funcționalitatea lor și vom explica cum lucrează aceștia.

Principalul algoritm constă în sortarea datelor despre angajați stocată în baza de date în ordine crescătoare conform ID-ului. Cum am menționat în capitolul 2.2, am ales algoritmul Bubble Sort. Întrucât acest proiect este realizat cu scop pur demonstrativ și educativ am decis să nu complicăm situația și așa complicată. Bubble Sort este un algoritm simplu de implementat și pe înțelesul tuturor, iar în cazul nostru acesta este potrivit pentru a sorta tabloul de angajați.

public static void BubbleSort(DataTable dt, int sortColumnIndex)

{

int n = dt.Rows.Count;

bool isUnsorted;

do

{

isUnsorted = false;

for (int i = 0; i < (n - 1); i++)

{

if (Convert.ToInt32(dt.Rows[i][sortColumnIndex]) > Convert.ToInt32(dt.Rows[i+1][sortColumnIndex]))

{

isUnsorted = true;

for (; i < (n - 1); i++)

{

if (Convert.ToInt32(dt.Rows[i][sortColumnIndex]) > Convert.ToInt32(dt.Rows[i+1][sortColumnIndex]))

{

//Schimbare randuri

DataRow temp = dt.NewRow();

temp.ItemArray = dt.Rows[i].ItemArray;

dt.Rows[i].ItemArray = dt.Rows[i + 1].ItemArray;

dt.Rows[i + 1].ItemArray = temp.ItemArray;

}

}

}

}

} while (isUnsorted);

}

Am optimizat algoritmul prin faptul că am adăugat o variabilă logică **isUnsorted** prin intermediul căreia verificam dacă tabloul de elemente nu este deja sortat. Aceasta ne-a ajutat să îmbunătățim lucrul aplicației. Principiul de bază al algoritmului este interschimbarea a câte 2 elemente ale tabloului până când acesta nu este sortat în totalitate.

Am utilizat obiectul **dt** de tip **DataTable** o clasă ce răspunde de modificarea și sortarea datelor într-un format tabelar, astfel având acces la metodele predefinite ale acestei clase cum ar fi Rows care accesează rândurile tabelului.

Variabila sortColumnIndex reprezinta numarul coloanei in baza careia dorim sa sortam datele.

public DataTable SelectUpdate()

{

SqlConnection conn = new SqlConnection(myconnstrng);

DataTable dt = new DataTable();

try

{

string sql = "SELECT \* FROM TabelListaAngajati";

SqlCommand cmd = new SqlCommand(sql, conn);

SqlDataAdapter adapter = new SqlDataAdapter(cmd);

conn.Open();

adapter.Fill(dt);

**//Sortarea Tabelului dupa ID utilizand bubble sort**

**int[] ages = new int[dt.Rows.Count];**

**for (int i = 0; i < dt.Rows.Count; i++)**

**{**

**ages[i] = Convert.ToInt32(dt.Rows[i]["ID"]);**

**}**

**BubbleSort(dt,0);//Indexul Id-Ului**

DataTable sortedDt = new DataTable();

sortedDt.Columns.Add("ID", typeof(int));

sortedDt.Columns.Add("Nume", typeof(string));

sortedDt.Columns.Add("Functia", typeof(string));

sortedDt.Columns.Add("Salariu", typeof(float));

sortedDt.Columns.Add("Varsta", typeof(int));

sortedDt.Columns.Add("Domiciliu", typeof(string));

sortedDt.Columns.Add("Email", typeof(string));

sortedDt.Columns.Add("Limba", typeof(string));

for (int i = 0; i < dt.Rows.Count; i++)

{

DataRow row = sortedDt.NewRow();

row["ID"] = Convert.ToInt32(dt.Rows[i]["ID"]);

row["Nume"] = dt.Rows[i]["Nume"].ToString();

row["Functia"] = dt.Rows[i]["Functia"].ToString();

row["Salariu"] = Convert.ToSingle(dt.Rows[i]["Salariu"]);

row["Varsta"] = Convert.ToSingle(dt.Rows[i]["Varsta"]);

row["Domiciliu"] = dt.Rows[i]["Domiciliu"].ToString();

row["Email"] = dt.Rows[i]["Email"].ToString();

row["Limba"] = dt.Rows[i]["Limba"].ToString();

sortedDt.Rows.Add(row);

}

dt = sortedDt;

}

catch (Exception ex)

{

}

finally

{

conn.Close();

}

return dt;

}

În secvența dată este arătată apelarea metodei BubbleSort(evidențiată). Putem observa că parametrii ei sunt **dt** și **sortColumnIndex** care la moment are valoarea 0, ceea ce înseamnă că sortarea se face conform ID-ului. Dacă am dori sortarea după alt parametru ca de exemplu vârsta am introduce numărul coloanei care stochează vârsta. Sortarea este predefinită doar pentru elemente de tip integer, astfel este posibilă sortarea după ID, vârstă și salariu;

## **3.2 Interfeţa grafica a aplicatiei**

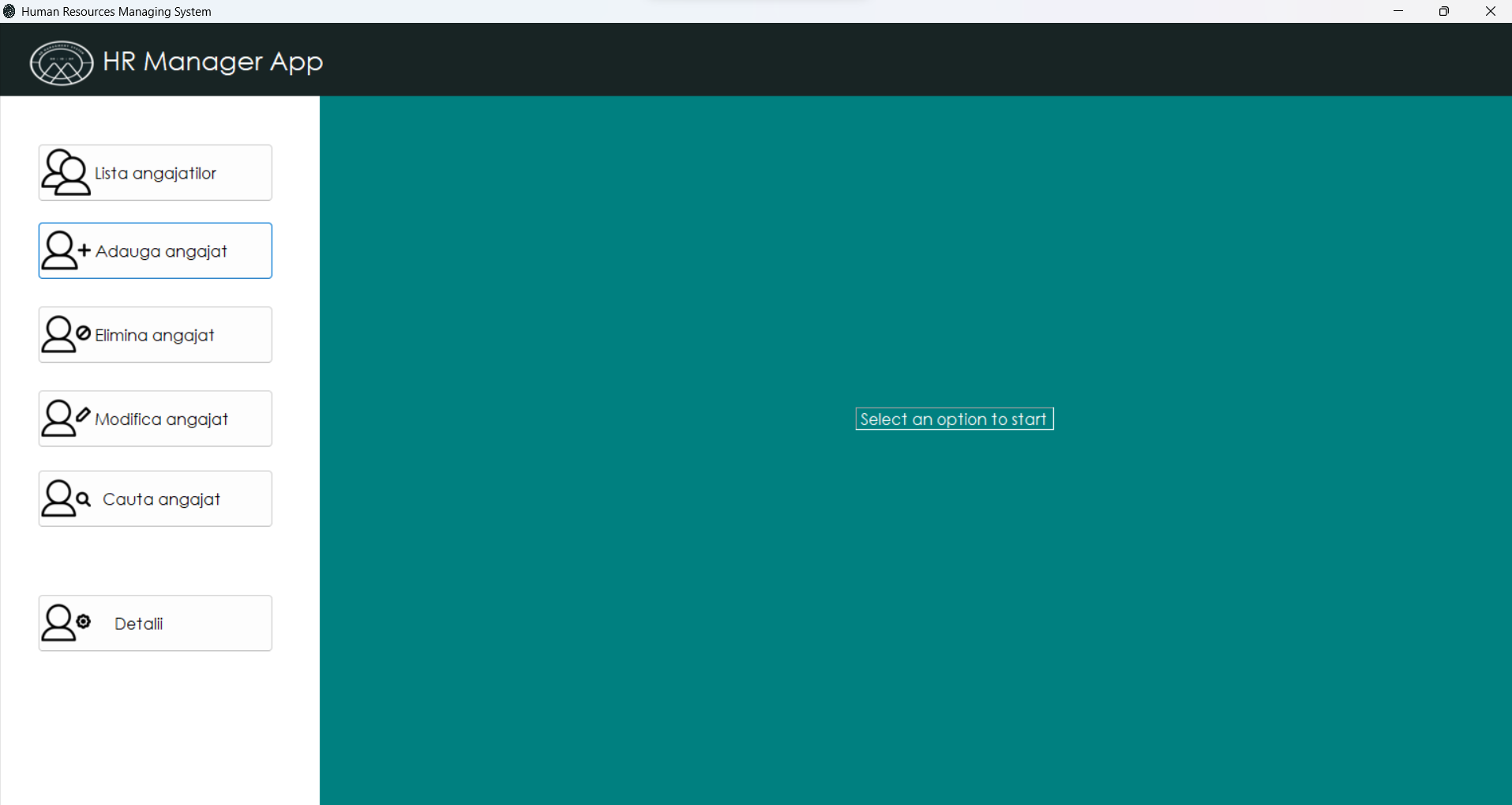
Un simplitatea realizării și funcționalitatea este cheea succesului oricărui proiect reușit. Din acest motiv am urmărit realizarea simplă, într-o manieră cât mai user-friendly a aplicației noastre.

Pentru realizarea logo-ului(Fig. 3.3) aplicației am folosit instrumentul online gratuit Canvas.



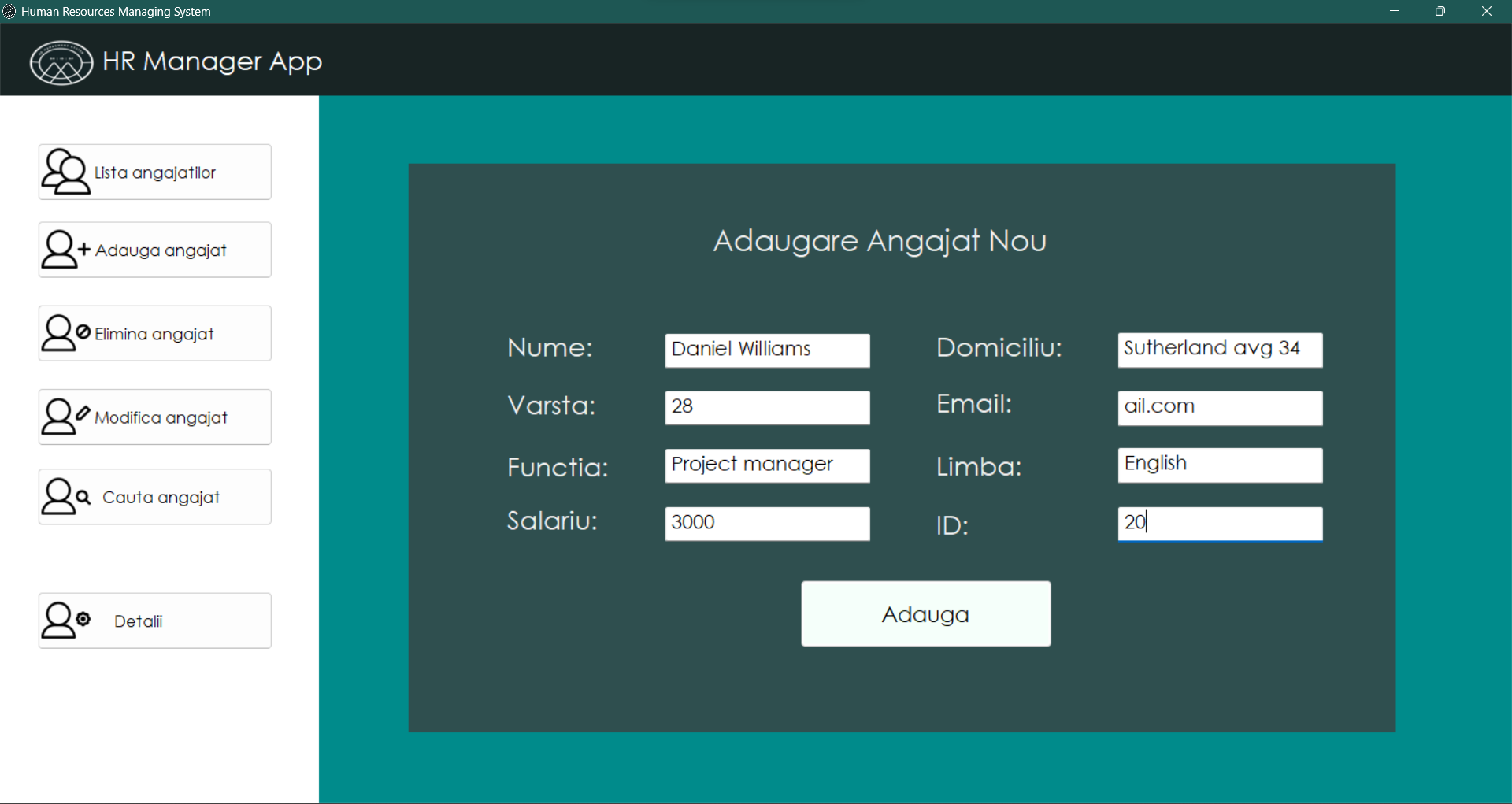
**Fig. 3.3 Logo-ul aplicației**

Interfața aplicației a fost realizată cu ajutorul Windows Forms, cum am menționat în capitolul 3. Am utilizat figuri simple pentru butoane de asemenea icoanele de pe fiecare buton au fost realizate cu ajutorul instrumentului Canvas.

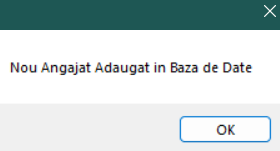


**Fig. 3.4 Pagina principală**

Pe pagina principală pe partea stângă sunt amplasate butoanele fiecărora corespunzându-le câte o funcție, iar în mijloc putem observa un scurt mesaj care sugerează selectarea unei optiuni pentru a începe lucrul.

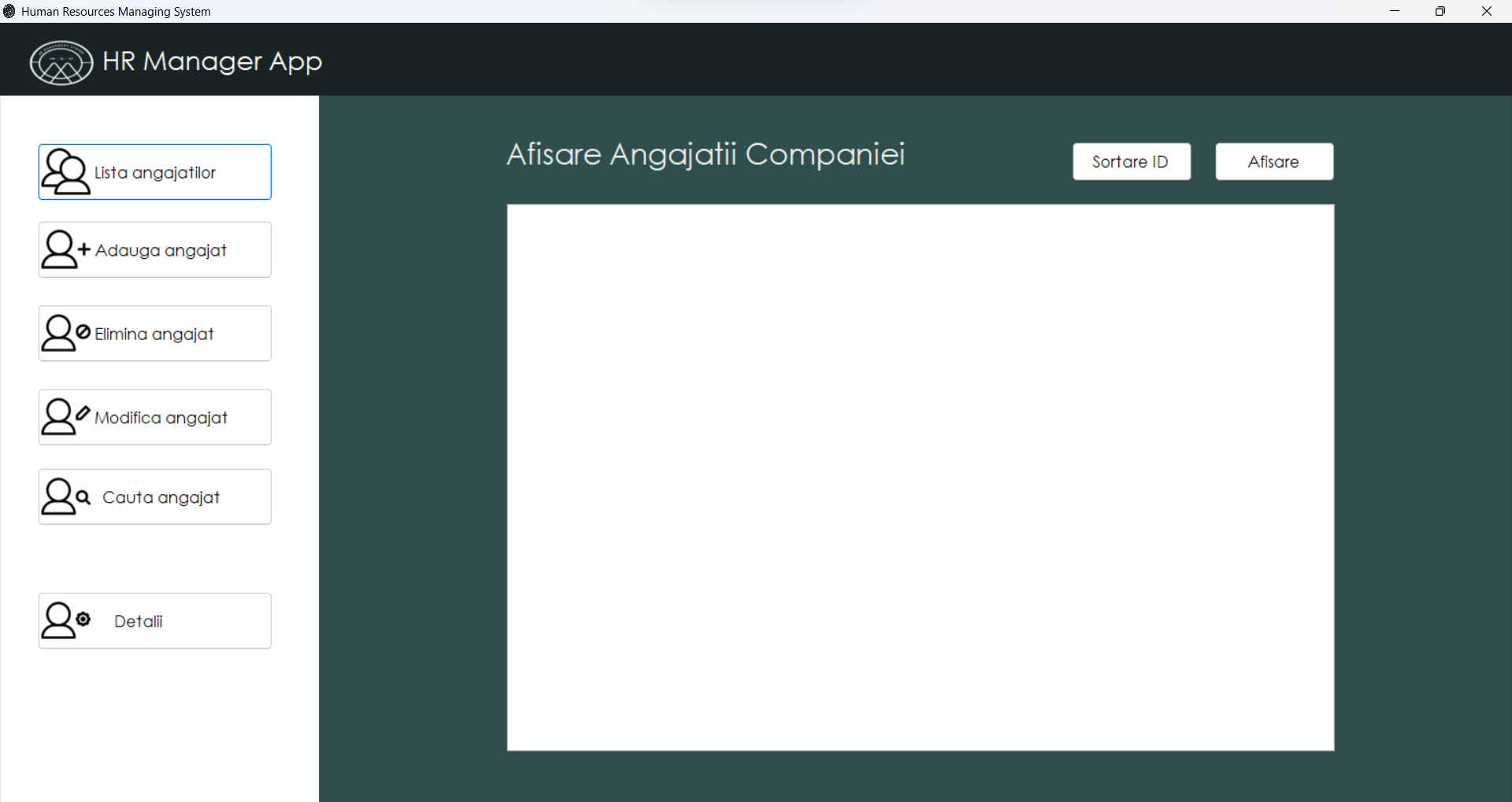


**Fig. 3.5 Adăugarea datelor unui angajat**



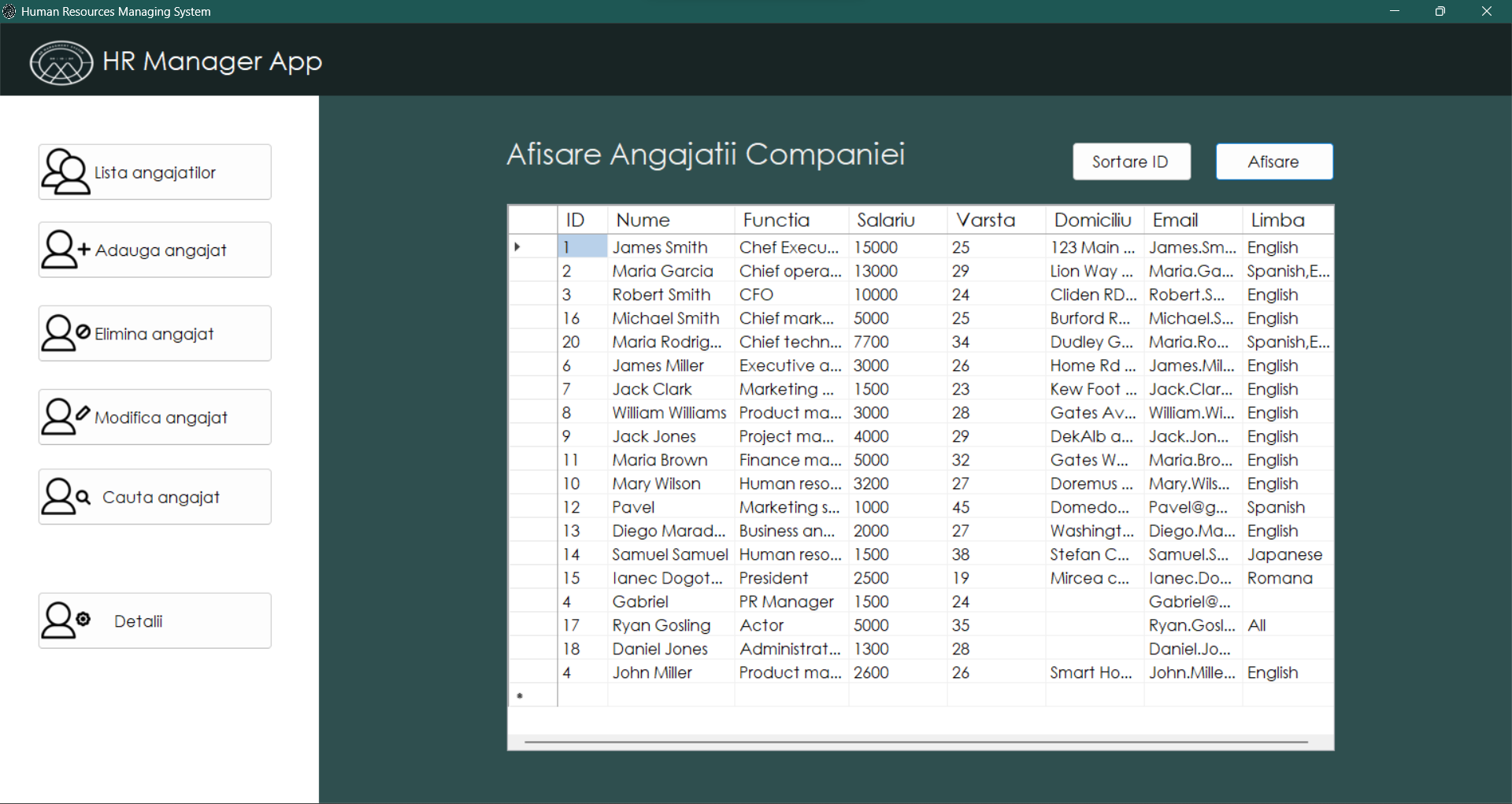
**Fig. 3.6 Mesaj afișat în urma adăugării**

În Fig. 3.5 este reprezentată adăugarea unui angajat. Fiecărei caracteristici îi corespunde un câmp în care putem introduce valorile care mai apoi în urma apăsării butonului **Adăugare** vor fi stocate în baza de date. Iar după ce apăsăm butonul **Adăugare**, în cazul în care datele corespund cerințelor, vom primi un mesaj de confirmare(Fig. 3.6).



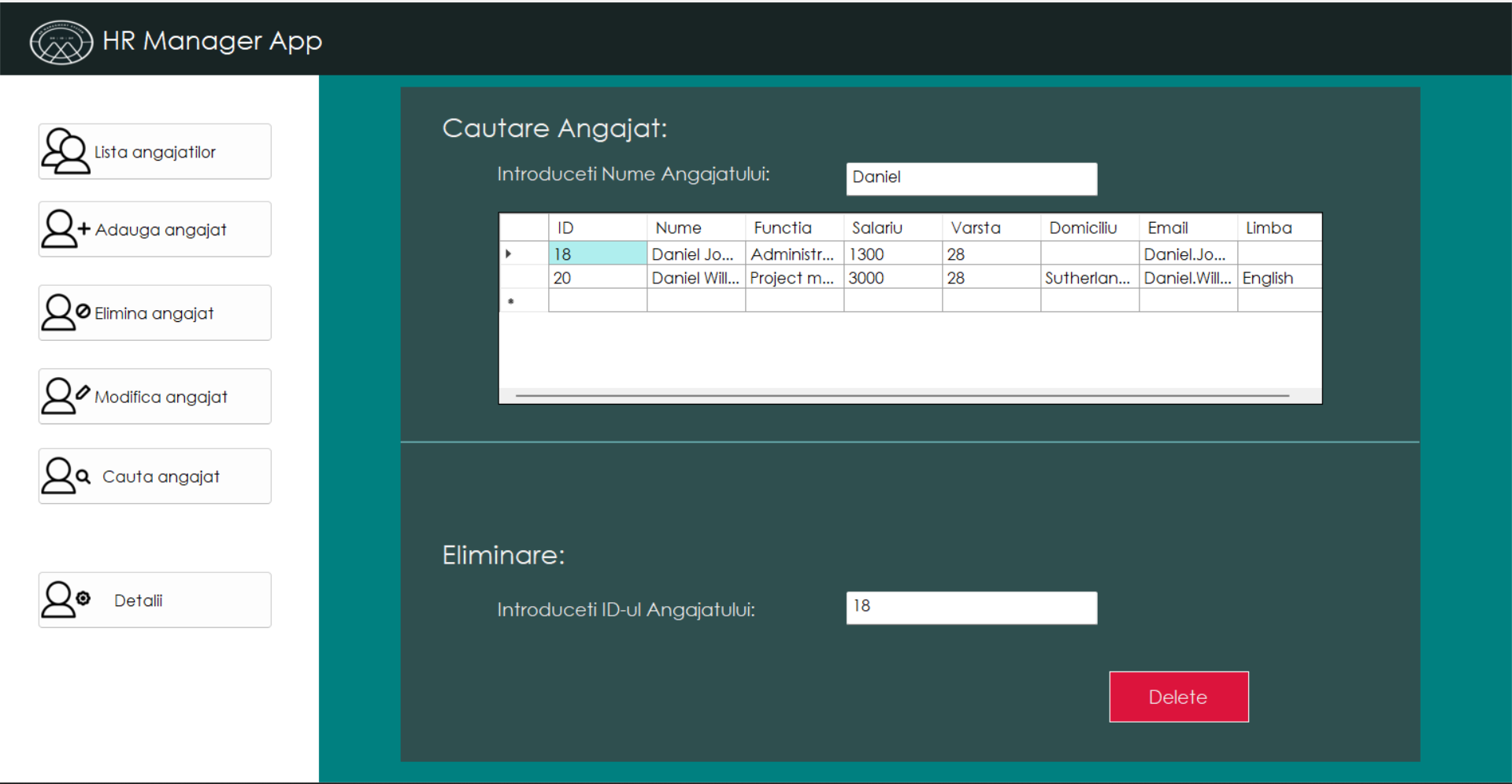
**Fig. 3.7 Lista angajaților**

Aici putem vedea lista angajaților(doar după ce introducem datele) apăsând butonul **Afișare**.



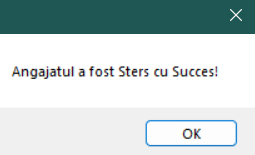
**Fig. 3.8 Lista angajaților(date adăugate)**

În urma apăsării butonului **Afișare** putem vedea toate datele introduse în tabel. Odată adăugate, datele pot fi sortate cu o simplă apăsare a butonului **Sortare ID**.

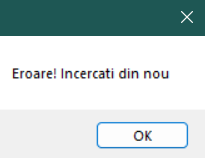


**Fig. 3.9 Eliminarea datelor unui angajat**

Pentru a elimina un angajat este nevoie să îl căutăm după nume iar mai apoi în partea de jos a paginii la **Eliminare** să introducem ID-ul angajatului pe care dorim să îl eliminăm. Cum putem observa în Fig. 3.9 există doi angajați cu numele Daniel, astfel introducerea ID-ului ajută la selectarea angajatului potrivit.



**Fig. 3.10 Mesaj în urma eliminării**



**Fig. 3.11 Mesaj în cazul introducerii unei valori incorecte**



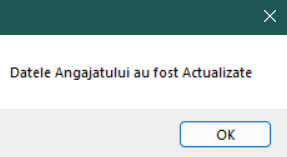
**Fig. 3.12 Modificarea datelor unui angajat**

Pentru a modifica datele unui angajat este necesar să introducem numele angajatului, și să îl selectăm cu ajutorul apăsării butonului indicat de săgeata din Fig. 3.12.

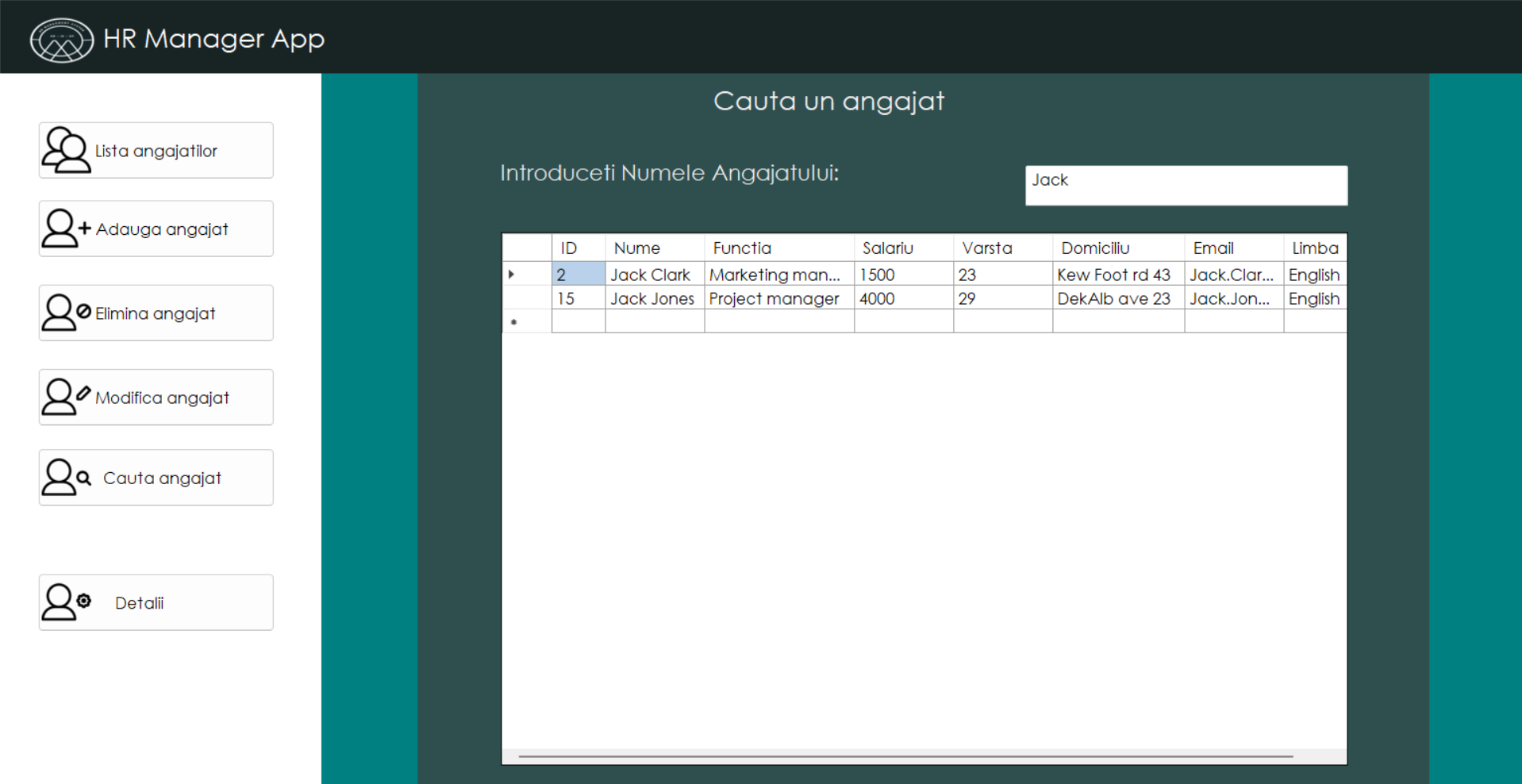


**Fig. 3.13 Modificarea datelor unui angajat(continuare)**

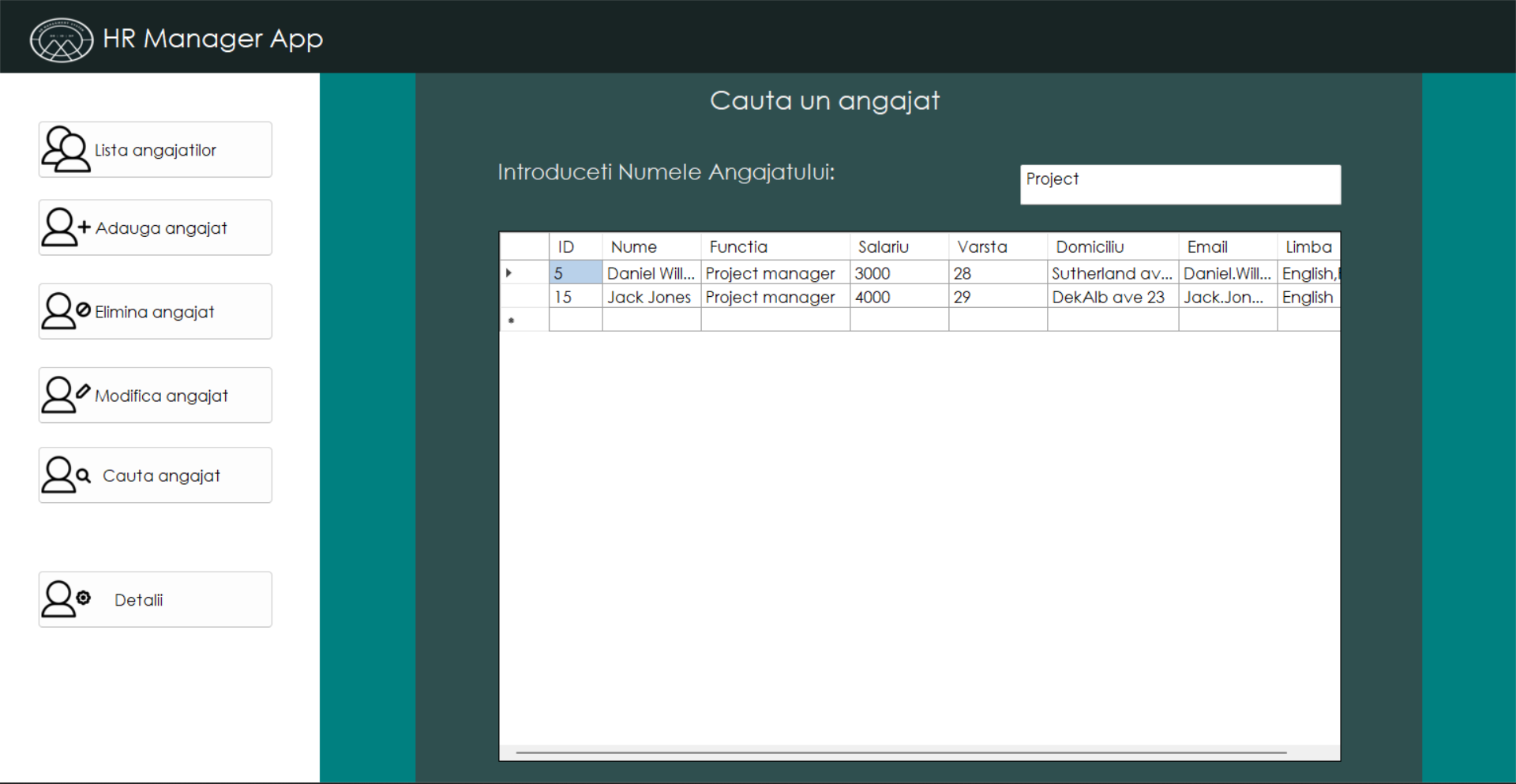
După ce am selectat angajatul datele căruia dorim să le modificăm, acestea se afișează mai jos. Pentru a le modifica ceva pur și simplu selectăm datele dorite, întroducem datele modificate și apăsăm butonul **Salvare** jos în partea dreaptă. După aceasta vom primiun mesaj scurt care confirmă că modificarea sa efectuat cu succes(Fig. 3.14). Schimbările efectuate pot fi văzute apăsând butonul **Lista angajaților**.



**Fig. 3.14 Mesaj de confirmare a modificării datelor**



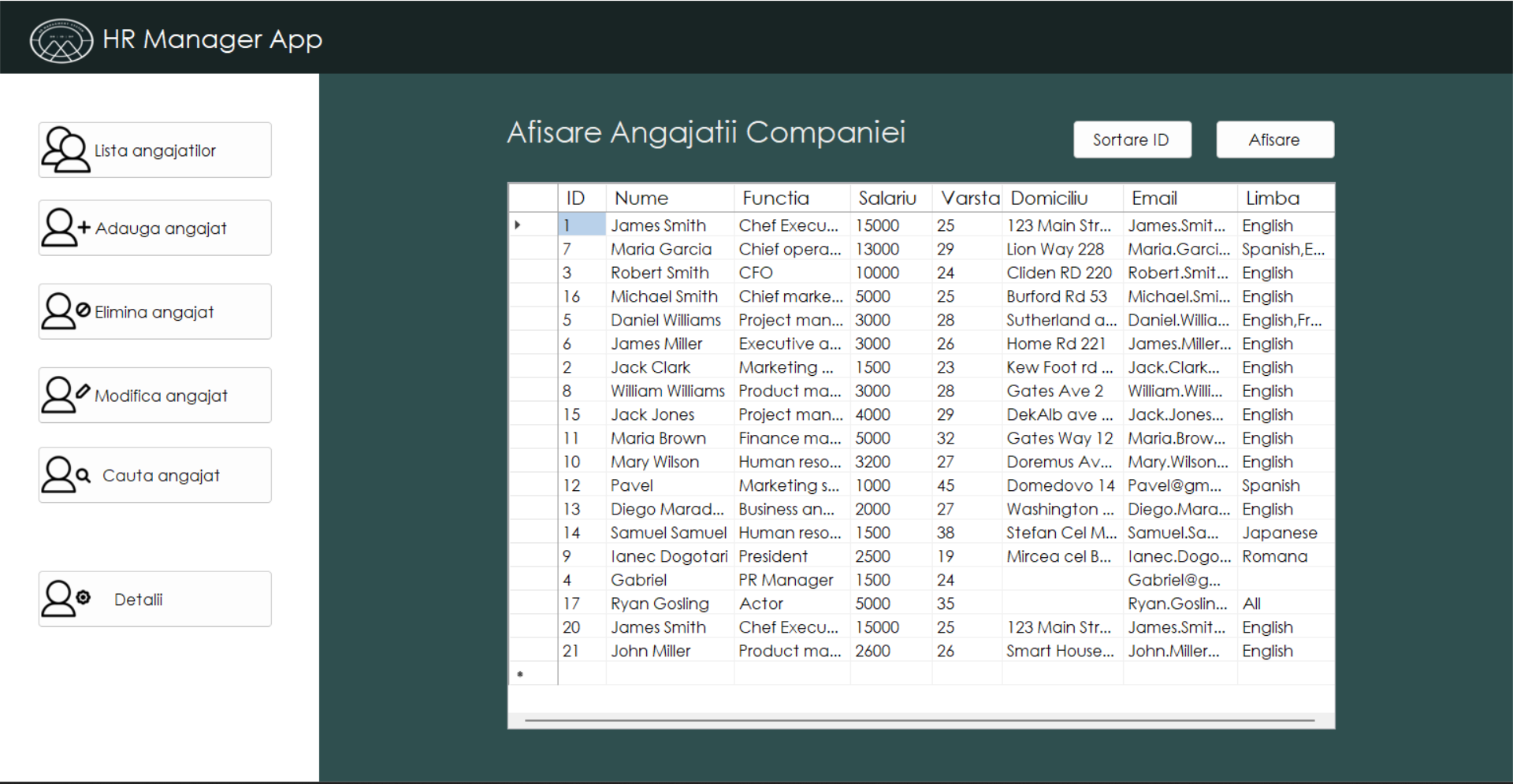
**Fig. 3.15 Căutarea unui angajat după nume**



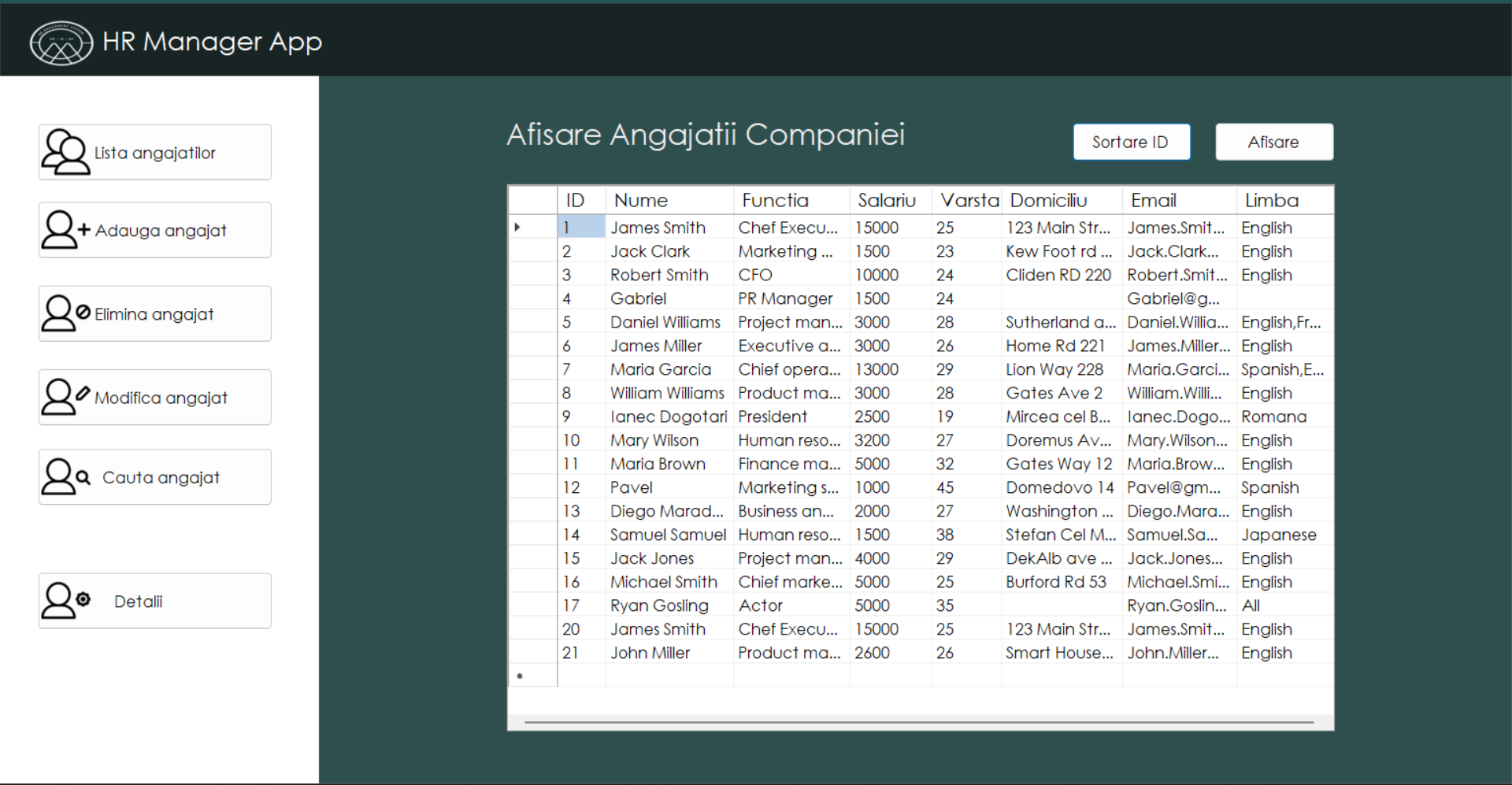
**Fig. 3.16 Căutarea unui angajat după funcție**

Mai sus vedem o demonstrație a procesului de căutare. În bara de căutare putem introduce orice informație despre angajat ca de exemplu numele(Fig. 3.15), sau funcția îndeplinită în organizație(Fig. 3.16).

Faptul că căutarea se poate efectua după oricare informație a unui angajat cum am menționat mai sus, oferă utilizatorului posibilitatea de a lucra mai rapid și mai eficient.



**Fig. 3.17 Lista angajaților(până la sortare)**



**Fig. 3.18 Lista angajaților(după sortare)**

Mai sus, în Fig. 3.17 și 3.18 este demonstrată sortarea. Cum am mai menționat în capitolul precedent sortarea se efectuează conform ID-ului. De asemenea este posibilă sortarea conform salariului și vârstei. Sortarea se efectuează în secțiunea **Lista angajaților** cu un simplu click pe butonul din partea de sus a paginii, **Sortare ID**.



**Fig. 3.19 Detalii despre proiect**

La apăsarea butonului **Detalii** din partea stângă pe ecran vor fi afișate detaliile despre proiect: denumirea proiectului, instituția de învățământ și numele fiecărui participant.

## **3.3 Analiza rezultatelor obtinute**

Spre sfârșit vom face o mică analiză a rezultatelor obținute, a problemei expuse în primul capitol și vom trece rapid peste toate aspectele proiectului nostru.

La început scopul nostru a fost de a implementa algoritmii necesari creării unui HRMS(Human Resources Managing System) care ar efectua operațiile necesare pentru a gestiona un număr anumit de angajați ai unei companii, la fel si alte operații suplimentare. Acest scop a fost îndeplinit în totalitatate, toate obiectivele fiind respectate.

Rezultatul obținut este o aplicație simplă de gestiune a resurselor umane, care oferă utilizatorului posibilitatea de a efectua diferite operații de bază, cum ar fi stocarea datelor despre angajați, modificarea acestora, ștergerea etc. Această aplicație este destul de ușor de utilizat. Fiecare pagină a aplicației este realizată într-o manieră cât mai user-friendly, astfel pentru a evita erorile și neînțelegerile. Fiecare pas este urmat de un mesaj care sau confirmă că schimbarea a fost efectuată cu succes sau atenționează utilizatorul despre faptul că ceva nu este în regulă astfel oferindu-i posibilitatea de a încerca din nou. În cazul lipsei datelor utilizatorul de asemenea primeste un mesaj de eroare care roagă utilizatorul să introducă datele înainte de a începe lucrul.

Framework-ul utilizat este .NET, care cu toate că suportă o varietate de limbaje, C# este cel mai utilizat. Datorită acestui framework am avut acces la diferite clase și funcții fără de care nu era posibilă realizarea aplicației pentru proiectul nostru.

Pentru a realiza aplicația dată am apelat la diferite tehnologii și instrumente, spre exemplu pentru realizarea interfeței grafice am utilizat Windows Forms, o bibliotecă de clase GUI inclusă în .Net Framework, care oferă posibilitatea de a crea orice aplicație pentru sistemul de operare Windows. Aplicațiile date pot conține diferite tipuri de controale, cum ar fi etichete, casete cu listă, indicații etc. Cu ajutorul WinForms am realizat un design cât mai estetic și mai plăcut.

De asemenea ne-am familiarizat cu bazele de date și cu limbajul SQL, care sunt lucruri destul de importante de cunoscut pentru viitor. Utilizarea bazelor de date SQL în cazul nostru a fost o necesitate, pentru că cu ajutorul funcțiilor predefinite am soluționat mult mai ușor unele aspecte ale problemei noastre.

# Concluzii

Odată cu dezvoltarea și progresul domeniului tehnologic, multe alte domenii au făcut și ele un mare pas în dezvoltare. Unul dintre aceste domenii care este strâns legat cu tehnologia informației este cel al resurselor umane. În combinarea acestor două domenii au apărut diferite modalități de a ușura lucrul în gestionarea acestor resurse, cum ar fi sistemele de management a resurselor umane(HRMS).

Investiția în software-ul HRMS permite companiilor să eficientizeze și să automatizeze sarcinile consumatoare de timp și obositoare. Aceste aplicații oferă perspective mult mai profunde asupra companiei. De asemenea acestea îmbunătățesc implicarea angajaților, cresc eficiența lucrului și eliberează personalul de HR pentru a se concentra pe proiecte cu valoare mai mare.

În această lucrare am urmărit realizarea unei astfel de aplicații care ar ușura îndeplinirea sarcinilor de bază a unui lucrător în domeniul HR. Pentru a obține rezultatul dorit a fost nevoie să ne documentăm despre domeniul dat, să studiem tehnologiile care urma să le folosim la elaborarea aplicației cum ar fi un nou limbaj, C#, limbajul SQL și bazele de date, .NET Framework, WinForms(pentru realizarea interfeței grafice).

Principalul algoritm utilizat în acest proiect este algoritmul de sortare, aceasta fiind un proces esențial în orice sistem de gestionare a resurselor umane. Am ales algoritmul BubbleSort din mai multe motive. Unul dintre aceste motive este faptul că acesta este destul de rapid în comparație cu alți algoritmi, chiar dacă are o complexitate mare. De asemenea acesta este foarte ușor de implementat și de înțeles. Un dezavantaj însă ar fi că are complexitatea mare și nu este tocmai potrivit pentru seturi mari de date.

Aplicația realizată în cadrul acestui proiect este doar unul din multiplele moduri de realizare a acesteia. Aplicațiile de gestionare a resurselor umane diferă de la una la alta. Fiecare aplicație are atât puncte forte cât și puncte slabe.

Cateva puncte forte ar fi:

* Comoditatea utilizării;
* Accesibilitatea;
* Simplitatea realizării;
* Aspectul estetic;

Ce ține de puncte slabe putem spune că aplicația dată nu realizează anumite funcții avansate precum ar fi: un sistem de salarizare a angajaților, sistem de compensare și de acordare a beneficiilor, managementul performanței fiecărui angajat, ceea ce ar fi complicat de realizat la momentul dat.

# Bibliografie

1. Introducere în Managementul resurselor umane; [Resursă electornică.] – Regim de acces:  
   <https://ro.wikipedia.org/wiki/Managementul_resurselor_umane>
2. Funcțiile unui HRMS; [Resursă electornică.] – Regim de acces:  
   <https://connecteam.com/best-human-resource-management-systems/>
3. Algoritmi de sortare; [Resursă electornică.] – Regim de acces:

<https://www.geeksforgeeks.org/sorting-algorithms/>

1. BubbleSort(imagini); [Resursă electornică.] – Regim de acces:

<https://www.geeksforgeeks.org/bubble-sort/>

1. Insertion Sort(imagini+informație); [Resursă electornică.] – Regim de acces:

<https://www.geeksforgeeks.org/insertion-sort/>

1. Merge Sort(imagini + informație); [Resursă electornică.] – Regim de acces:

<https://www.programiz.com/dsa/merge-sort>

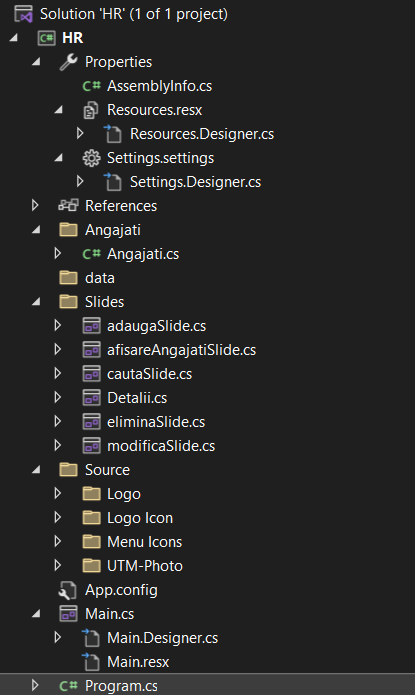
1. Quick Sort(imagini + informație); [Resursă electornică.] – Regim de acces:

[https://www.geeksforgeeks.org/quick-sort/](https://www.geeksforgeeks.org/quick-sort/%20)

1. SQL; [Resursă electornică.] – Regim de acces:

[https://en.wikipedia.org/wiki/SQL](https://en.wikipedia.org/wiki/SQL%20)

# Anexa A



**Fig. A.1 Structura proiectului**

**Codul fișierului Angajati.cs**:

using System;

using System.Collections;

using System.Collections.Generic;

using System.Configuration;

using System.Data;

using System.Data.SqlClient;

using System.Diagnostics.Eventing.Reader;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Xml;

namespace HR.Employee

{

class Angajati

{

public int ID { get; set; }

public string Nume { get; set; }

public int Varsta { get; set; }

public string Functia { get; set; }

public float Salariu { get; set; }

public string Domiciliu { get; set; }

public string Email { get; set; }

public string Limba { get; set; }

static string myconnstrng = ConfigurationManager.ConnectionStrings["connstrng"].ConnectionString;

//Selectare date din DataBase

public DataTable Select()

{

//1.Connectare

SqlConnection conn = new SqlConnection(myconnstrng);

DataTable dt = new DataTable();

try

{

//Scriu SQL Query

string sql = "SELECT \* FROM TabelListaAngajati";

//Creare cmd using si conn

SqlCommand cmd = new SqlCommand(sql, conn);

//Creare SQL Adapter using cmd

SqlDataAdapter adapter = new SqlDataAdapter(cmd);

conn.Open();

adapter.Fill(dt);

}

catch (Exception ex)

{

}

finally

{

conn.Close();

}

return dt;

}

//Scriere Date in DataBase

public bool Insert(Angajati c)

{

//Crearea return type default

bool isSuccess = false;

//1.Conectare DataBase

SqlConnection conn = new SqlConnection(myconnstrng);

try

{

//Creare SQL Qwery pentru a inserta datele

string sql = "INSERT INTO TabelListaAngajati(ID, Nume, Functia, Salariu, Varsta, Domiciliu, Email,Limba) VALUES(@ID, @Nume, @Functia, @Salariu, @Varsta, @Domiciliu, @Email,@Limba)";

//Creare SQL comanda

SqlCommand cmd = new SqlCommand(sql, conn);

//Creare Parametri pentru a adauga datele

cmd.Parameters.AddWithValue("@ID", c.ID);

cmd.Parameters.AddWithValue("@Nume", c.Nume);

cmd.Parameters.AddWithValue("@Functia", c.Functia);

cmd.Parameters.AddWithValue("@Salariu", c.Salariu);

cmd.Parameters.AddWithValue("@Varsta", c.Varsta);

cmd.Parameters.AddWithValue("@Domiciliu", c.Domiciliu);

cmd.Parameters.AddWithValue("@Email", c.Email);

cmd.Parameters.AddWithValue("@Limba", c.Limba);

//Conectarea Open

conn.Open();

int rows = cmd.ExecuteNonQuery();

//Daca se Executa,rews va fi mai mare ca 0

if (rows > 0) { isSuccess = true; } else { isSuccess = false; }

}

catch (Exception ex)

{

}

finally

{

conn.Close();

}

return isSuccess;

}

//Actualizare date

public bool Update(Angajati c)

{

bool isSuccess = false;

SqlConnection conn = new SqlConnection(myconnstrng);

try

{

string sql = "UPDATE TabelListaAngajati SET Nume=@Nume, Functia=@Functia, Salariu=@Salariu, Varsta=@Varsta, Domiciliu=@Domiciliu, Email=@Email,Limba=@Limba WHERE ID=@ID";

SqlCommand cmd = new SqlCommand(sql, conn);

cmd.Parameters.AddWithValue("@Nume", c.Nume);

cmd.Parameters.AddWithValue("@Functia", c.Functia);

cmd.Parameters.AddWithValue("@Salariu", c.Salariu);

cmd.Parameters.AddWithValue("@Varsta", c.Varsta);

cmd.Parameters.AddWithValue("@Domiciliu", c.Domiciliu);

cmd.Parameters.AddWithValue("@Email", c.Email);

cmd.Parameters.AddWithValue("@Limba", c.Limba);

cmd.Parameters.AddWithValue("@ID", c.ID);

conn.Open();

int rows = cmd.ExecuteNonQuery();

if (rows > 0)

{

isSuccess = true;

}

else

{

isSuccess = false;

}

}

catch (Exception ex)

{

}

finally

{

conn.Close();

}

return isSuccess;

}

public bool Delete(Angajati c)

{

//Crearea return type default

bool isSuccess = false;

//1.Conectare DataBase

SqlConnection conn = new SqlConnection(myconnstrng);

try

{

//Creare SQL Qwery pentru a Sterge datele

string sql = "DELETE FROM TabelListaAngajati WHERE ID=@ID";

//Creare SQL comanda

SqlCommand cmd = new SqlCommand(sql, conn);

//Creare Parametri pentru a adauga datele

cmd.Parameters.AddWithValue("@ID", c.ID);

//Conectarea Open

conn.Open();

int rows = cmd.ExecuteNonQuery();

//Daca se Executa,rews va fi mai mare ca 0

if (rows > 0) { isSuccess = true; } else { isSuccess = false; }

}

catch (Exception ex)

{

}

finally

{

conn.Close();

}

return isSuccess;

}

public static void BubbleSort(DataTable dt, int sortColumnIndex)

{

int n = dt.Rows.Count;

bool isUnsorted;

do

{

isUnsorted = false;

for (int i = 0; i < (n - 1); i++)

{

if (Convert.ToInt32(dt.Rows[i][sortColumnIndex]) > Convert.ToInt32(dt.Rows[i+1][sortColumnIndex]))

{

isUnsorted = true;

for (; i < (n - 1); i++)

{

if (Convert.ToInt32(dt.Rows[i][sortColumnIndex]) > Convert.ToInt32(dt.Rows[i+1][sortColumnIndex]))

{

//Schimbare randuri

DataRow temp = dt.NewRow();

temp.ItemArray = dt.Rows[i].ItemArray;

dt.Rows[i].ItemArray = dt.Rows[i + 1].ItemArray;

dt.Rows[i + 1].ItemArray = temp.ItemArray;

}

}

}

}

} while (isUnsorted);

}

public DataTable SelectUpdate()

{

SqlConnection conn = new SqlConnection(myconnstrng);

DataTable dt = new DataTable();

try

{

string sql = "SELECT \* FROM TabelListaAngajati";

SqlCommand cmd = new SqlCommand(sql, conn);

SqlDataAdapter adapter = new SqlDataAdapter(cmd);

conn.Open();

adapter.Fill(dt);

//Sortarea Tabelului dupa ID utilizand bubble sort

int[] ages = new int[dt.Rows.Count];

for (int i = 0; i < dt.Rows.Count; i++)

{

ages[i] = Convert.ToInt32(dt.Rows[i]["ID"]);

}

BubbleSort(dt,0);//Indexul Id-Ului

DataTable sortedDt = new DataTable();

sortedDt.Columns.Add("ID", typeof(int));

sortedDt.Columns.Add("Nume", typeof(string));

sortedDt.Columns.Add("Functia", typeof(string));

sortedDt.Columns.Add("Salariu", typeof(float));

sortedDt.Columns.Add("Varsta", typeof(int));

sortedDt.Columns.Add("Domiciliu", typeof(string));

sortedDt.Columns.Add("Email", typeof(string));

sortedDt.Columns.Add("Limba", typeof(string));

for (int i = 0; i < dt.Rows.Count; i++)

{

DataRow row = sortedDt.NewRow();

row["ID"] = Convert.ToInt32(dt.Rows[i]["ID"]);

row["Nume"] = dt.Rows[i]["Nume"].ToString();

row["Functia"] = dt.Rows[i]["Functia"].ToString();

row["Salariu"] = Convert.ToSingle(dt.Rows[i]["Salariu"]);

row["Varsta"] = Convert.ToSingle(dt.Rows[i]["Varsta"]);

row["Domiciliu"] = dt.Rows[i]["Domiciliu"].ToString();

row["Email"] = dt.Rows[i]["Email"].ToString();

row["Limba"] = dt.Rows[i]["Limba"].ToString();

sortedDt.Rows.Add(row);

}

dt = sortedDt;

}

catch (Exception ex)

{

}

finally

{

conn.Close();

}

return dt;

}

public bool SaveChangesToDB(DataTable dt)

{

bool isSuccess = false;

SqlConnection conn = new SqlConnection(myconnstrng);

//Stergere Datele

string sql = "DELETE FROM TabelListaAngajati";

SqlCommand cmd = new SqlCommand(sql, conn);

conn.Open();

cmd.ExecuteNonQuery();

//Update

try

{

sql = "SELECT \* FROM TabelListaAngajati";

SqlDataAdapter adapter = new SqlDataAdapter(sql, conn);

SqlCommandBuilder builder = new SqlCommandBuilder(adapter);

adapter.Update(dt);

isSuccess = true;

}

catch (Exception ex)

{

}

finally

{

conn.Close();

}

return isSuccess;

}

}

}

**Codul fișierului adaugaSlide.cs**:

using HR.Employee;

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Xml;

//using System.Data.SqlClient;

namespace HR

{

public partial class adaugaSlide : Form

{

public adaugaSlide()

{

InitializeComponent();

}

Angajati c = new Angajati();

private void textBox3\_TextChanged(object sender, EventArgs e)

{

}

private void AdaugaButton\_Click(object sender, EventArgs e)

{

//Obtine Valori

c.Nume = addNume.Text;

//String->Int

c.Varsta = int.Parse(addVarsta.Text);

c.Functia = addFunctia.Text;

c.Salariu = int.Parse(addSalariu.Text);

c.Domiciliu = addDomiciliu.Text;

c.Email = addEmail.Text;

c.Limba = addLimba.Text;

c.ID = int.Parse(addID.Text);

//Inserta Date

bool success = c.Insert(c);

if(success == true) { MessageBox.Show("Nou Angajat Adaugat in Baza de Date");Clear(); } else { MessageBox.Show("Eroare");Clear(); }

}

public void Clear()

{

addNume.Text = "";

addVarsta.Text = "";

addFunctia.Text = "";

addSalariu.Text = "";

addDomiciliu.Text = "";

addEmail.Text = "";

addLimba.Text = "";

addID.Text = "";

}

private void adaugaSlide\_Load(object sender, EventArgs e)

{

}

private void dataGridView1\_CellContentClick(object sender, DataGridViewCellEventArgs e)

{

}

private void addLimba\_TextChanged(object sender, EventArgs e)

{

}

private void label2\_Click(object sender, EventArgs e)

{

}

private void DomiciliuLabel\_Click(object sender, EventArgs e)

{

}

private void panel1\_Paint(object sender, PaintEventArgs e)

{

}

}

}

**Codul fișierului afisareAngajatiSlide.cs**:

using HR.Employee;

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace HR.Slides

{

public partial class afisareAngajatiSlide : Form

{

public afisareAngajatiSlide()

{

InitializeComponent();

}

Angajati c = new Angajati();

private void label1\_Click(object sender, EventArgs e)

{

}

private void AfisareButton\_Click(object sender, EventArgs e)

{

DataTable dt = c.Select();

dataGridView1.DataSource = dt;

}

private void afisareAngajatiSlide\_Load(object sender, EventArgs e)

{

}

private void dataGridView1\_CellContentClick(object sender, DataGridViewCellEventArgs e)

{

}

private void dataGridView1\_CellContentClick\_1(object sender, DataGridViewCellEventArgs e)

{

DataTable dt = c.Select();

dataGridView1.DataSource = dt;

}

private void SortareButton\_Click(object sender, EventArgs e)

{

if (dataGridView1.DataSource != null)

{

Angajati a = new Angajati();

DataTable dt = a.SelectUpdate();

dataGridView1.DataSource = dt;

a.SaveChangesToDB(dt);

}

else

{

MessageBox.Show("Nu sunt date!");

}

}

}

}

**Codul fișierului** **cautaSlide.cs**:

using HR.Employee;

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Configuration;

using System.Data;

using System.Data.SqlClient;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace HR

{

public partial class cautaSlide : Form

{

public cautaSlide()

{

InitializeComponent();

}

Angajati c = new Angajati();

private void label1\_Click(object sender, EventArgs e)

{

}

static string myconnstr = ConfigurationManager.ConnectionStrings["connstrng"].ConnectionString;

private void CautaNume\_TextChanged(object sender, EventArgs e)

{

DataTable dt = c.Select();

dataGridView1.DataSource = dt;

string keyword = CautaNume.Text;

SqlConnection conn = new SqlConnection(myconnstr);

SqlDataAdapter sda = new SqlDataAdapter("SELECT \* from TabelListaAngajati WHERE Nume LIKE '%" +keyword+ "%' OR ID LIKE '%" +keyword+ "%' OR Functia LIKE '%" +keyword+ "%' OR Email LIKE '%" +keyword+ "%'", conn);

dt = new DataTable();

sda.Fill(dt);

dataGridView1.DataSource = dt;

}

private void dataGridView1\_CellContentClick(object sender, DataGridViewCellEventArgs e)

{

DataTable dt = c.Select();

dataGridView1.DataSource = dt;

}

private void cautaSlide\_Load(object sender, EventArgs e)

{

}

private void panel1\_Paint(object sender, PaintEventArgs e)

{

}

}

}

**Codul fișierului** **eliminaSlide.cs**:

using HR.Employee;

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Configuration;

using System.Data;

using System.Data.SqlClient;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace HR

{

public partial class eliminaSlide : Form

{

public eliminaSlide()

{

InitializeComponent();

}

Angajati c = new Angajati();

static string myconnstr = ConfigurationManager.ConnectionStrings["connstrng"].ConnectionString;

private void eliminaSlide\_Load(object sender, EventArgs e)

{

}

public void Clear()

{

Nume.Text = "";

ID.Text = "";

}

private void button1\_Click(object sender, EventArgs e)

{

c.ID = Convert.ToInt32(ID.Text);

bool succes = c.Delete(c);

if (succes==true)

{

MessageBox.Show("Angajatul a fost Sters cu Succes!");

Clear();

}

else

{

MessageBox.Show("Eroare! Incercati din nou");

Clear();

}

}

private void Nume\_TextChanged(object sender, EventArgs e)

{

DataTable dt = c.Select();

dataGridView1.DataSource = dt;

string keyword = Nume.Text;

SqlConnection conn = new SqlConnection(myconnstr);

SqlDataAdapter sda = new SqlDataAdapter("SELECT \* from TabelListaAngajati WHERE Nume LIKE '%" + keyword + "%' ", conn);

dt = new DataTable();

sda.Fill(dt);

dataGridView1.DataSource = dt;

}

private void dataGridView1\_CellContentClick(object sender, DataGridViewCellEventArgs e)

{

DataTable dt = c.Select();

dataGridView1.DataSource = dt;

}

private void panel1\_Paint(object sender, PaintEventArgs e)

{

}

}

}

**Codul fișierului** **modificaSlide.cs**:

using HR.Employee;

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Configuration;

using System.Data;

using System.Data.SqlClient;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace HR

{

public partial class modificaSlide : Form

{

public modificaSlide()

{

InitializeComponent();

}

Angajati c = new Angajati();

static string myconnstr = ConfigurationManager.ConnectionStrings["connstrng"].ConnectionString;

private void label9\_Click(object sender, EventArgs e)

{

}

public void Clear()

{

addNume.Text = "";

addVarsta.Text = "";

addFunctia.Text = "";

addSalariu.Text = "";

addDomiciliu.Text = "";

addEmail.Text = "";

addLimba.Text = "";

addID.Text = "";

cautaNume.Text = "";

}

private void Nume\_TextChanged(object sender, EventArgs e)

{

DataTable dt = c.Select();

afisareAngajati.DataSource = dt;

string keyword = cautaNume.Text;

SqlConnection conn = new SqlConnection(myconnstr);

SqlDataAdapter sda = new SqlDataAdapter("SELECT \* from TabelListaAngajati WHERE Nume LIKE '%" + keyword + "%'", conn);

dt = new DataTable();

sda.Fill(dt);

afisareAngajati.DataSource = dt;

}

private void afisareAngajati\_RowHeaderMouseClick(object sender, DataGridViewCellMouseEventArgs e)

{

//Introducere date in Casete

int rowIndex = e.RowIndex;

addID.Text = afisareAngajati.Rows[rowIndex].Cells[0].Value.ToString();

addNume.Text = afisareAngajati.Rows[rowIndex].Cells[1].Value.ToString();

addFunctia.Text = afisareAngajati.Rows[rowIndex].Cells[2].Value.ToString();

addSalariu.Text = afisareAngajati.Rows[rowIndex].Cells[3].Value.ToString();

addVarsta.Text = afisareAngajati.Rows[rowIndex].Cells[4].Value.ToString();

addDomiciliu.Text = afisareAngajati.Rows[rowIndex].Cells[5].Value.ToString();

addEmail.Text = afisareAngajati.Rows[rowIndex].Cells[6].Value.ToString();

addLimba.Text = afisareAngajati.Rows[rowIndex].Cells[7].Value.ToString();

}

private void dataGridView1\_CellContentClick(object sender, DataGridViewCellEventArgs e)

{

//DataTable dt = c.Select();

//afisareAngajati.DataSource = dt;

}

private void button1\_Click(object sender, EventArgs e)

{

//Obtine Valori

c.ID = int.Parse(addID.Text);

c.Nume = addNume.Text;

//String->Int

c.Varsta = int.Parse(addVarsta.Text);

c.Functia = addFunctia.Text;

c.Salariu = float.Parse(addSalariu.Text);

c.Domiciliu = addDomiciliu.Text;

c.Email = addEmail.Text;

c.Limba = addLimba.Text;

//Inserta Date

bool success = c.Update(c);

if (success == true)

{

MessageBox.Show("Datele Angajatului au fost Actualizate");

DataTable dt = c.Select();

afisareAngajati.DataSource = dt;

Clear();

}

else

{

MessageBox.Show("Eroare");

Clear();

}

}

private void addVarsta\_TextChanged(object sender, EventArgs e)

{

}

private void modificaSlide\_Load(object sender, EventArgs e)

{

}

private void afisareAngajati\_CellContentClick(object sender, DataGridViewCellEventArgs e)

{

DataTable dt = c.Select();

afisareAngajati.DataSource = dt;

}

private void addID\_TextChanged(object sender, EventArgs e)

{

}

private void panel1\_Paint(object sender, PaintEventArgs e)

{

}

}

}