Universitatea Tehnică a Moldovei

Facultatea Calculatoare Informatică şi Microelectronică

Departamentul Ingineria Software şi Automatică

Aplicația Teoriei și Practicii în Dezvoltarea Backend: O Abordare Integrată a Fundamentelor C#, OOP și MS SQL.

Student(ă): Chirita Stanislav

Coordonator universitate:Gaidarji Alina

Chişinău, 2023

CUPRINS

INTRODUCERE

În cadrul acestei lucrări, se va explora și analiza aplicația Teoriei și Practicii în dezvoltarea backend, cu accent pe abordarea integrată a fundamentelor C#, programării orientate pe obiect (OOP) și gestionării bazelor de date utilizând MS SQL. Această investigație se va concentra pe aspectele esențiale ale dezvoltării backend, oferind o perspectivă impersonală asupra subiectului.

Scopul acestei analize este de a ilustra modul în care teoria și practica se întrepătrund în dezvoltarea backend și de a evidenția importanța unei abordări integrate a limbajului de programare C#, a principiilor OOP și a gestionării bazelor de date MS SQL în procesul de dezvoltare a aplicațiilor backend robuste și eficiente.

Pe parcursul acestui studiu, vom explora principiile de bază ale C#, evidențiind modul în care acestea pot fi aplicate în dezvoltarea backend pentru a crea aplicații scalabile și fiabile. De asemenea, vom examina modul în care OOP poate contribui la crearea unei structuri eficiente a codului și la gestionarea complexității în dezvoltarea backend.

În plus, vom analiza aspectele legate de MS SQL și gestionarea bazelor de date, cu un accent special pe integrarea acestei tehnologii în dezvoltarea backend. Vom explora metodele de conectare la baze de date, manipularea datelor și optimizarea performanței în cadrul aplicațiilor backend.

Prin această abordare integrată a fundamentelor C#, OOP și MS SQL, vom dezvălui beneficiile și complexitatea dezvoltării backend, oferind o perspectivă cuprinzătoare asupra acestui domeniu vital al dezvoltării software.

1 ANALIZA DOMENIULUI DE STUDIU

1.1 Importanța temei

Importanța temei se manifestă în modul în care aceasta contribuie la evoluția și eficiența dezvoltării backend în domeniul software. Prin abordarea integrată a fundamentelor C#, OOP și MS SQL, se dezvăluie o perspectivă esențială asupra procesului de dezvoltare a aplicațiilor backend. Această abordare nu se limitează doar la teorie, ci aduce în prim-plan practica și aplicarea concretă a cunoștințelor.

Prin înțelegerea profundă a limbajului C#, dezvoltatorii pot construi aplicații mai consistente și mai ușor de întreținut. De asemenea, integrarea principiilor OOP permite gestionarea mai eficientă a complexității în dezvoltarea software, ceea ce duce la aplicații mai robuste și scalabile. Gestionarea bazelor de date MS SQL este esențială în asigurarea stocării și manipulării corecte a datelor, având un impact semnificativ asupra performanței și securității aplicațiilor.

În ansamblu, abordarea integrată a acestor concepte în dezvoltarea backend joacă un rol crucial în construirea unor aplicații software de înaltă calitate, care să răspundă cerințelor complexe ale utilizatorilor și să ofere o experiență fluidă și fiabilă. Astfel, tema se relevă ca fiind de o importanță majoră pentru dezvoltatorii de software și pentru întreaga industrie IT.

1.2 Descrierea scenariului

În descrierea scenariului, se deschide perspectiva asupra unui mediu în care tema dezvoltării backend cu accent pe fundamentarea în C#, OOP și MS SQL devine esențială. În acest mediu, se pot identifica următoarele aspecte relevante:

1. **Aplicații Web și Mobile**: În contextul actual al tehnologiei, dezvoltarea aplicațiilor web și mobile este la ordinea zilei. Scenariul implică crearea și gestionarea sistemelor și serviciilor backend care stau la baza acestor aplicații. Acestea trebuie să fie robuste, eficiente și să asigure o comunicare fluidă cu frontend-ul și cu baza de date.

2. **Domenii variate**: Dezvoltarea backend este esențială într-o gamă largă de domenii, de la comerțul electronic la serviciile financiare, de la sănătate la divertisment. Astfel, abordarea integrată în C#, OOP și MS SQL se aplică în diverse contexte și scenarii specifice.

3. **Complexitate și scalabilitate**: Aplicațiile moderne trebuie să facă față unor cerințe din ce în ce mai complexe și să poată escalada pentru a răspunde creșterii numărului de utilizatori. Dezvoltarea backend trebuie să țină cont de aceste aspecte, iar fundamentarea în C#, OOP și MS SQL oferă instrumentele necesare pentru gestionarea complexității și scalabilității.

4. **Securitate şi Integritatea Datelor**: Manipularea datelor sensibile este o preocupare majoră în dezvoltarea backend. Utilizarea adecvată a limbajului C# şi a principiilor OOP contribuie la construirea de aplicaţii mai sigure, în timp ce MS SQL permite gestionarea şi protejarea adecvată a bazelor de date.
5. **Ciclul de Viaţă al Dezvoltării**: Scenariul descrie ciclul de viaţă al dezvoltării backend, de la planificare şi proiectare la implementare, testare şi întreţinere. Fiecare etapă a acestui proces beneficiază de abordarea integrată în C#, OOP şi MS SQL pentru a asigura coerenţa şi eficienţa întregului proiect.

În ansamblu, scenariul descrie un mediu complex şi în continuă evoluţie, în care dezvoltatorii backend se confruntă cu provocări semnificative. Fundamentarea în C#, OOP şi MS SQL devine, astfel, un element cheie pentru succesul acestor dezvoltatori în asigurarea funcţionalităţii, performanţei şi securităţii aplicaţiilor pe care le dezvoltă.

1.3 Scop şi Obiective

Obiectivele acestei abordări includ:

1. Înţelegerea conceptelor de bază ale limbajului de programare C#:
    - Sintaxa şi semantica limbajului C#.
    - Declararea şi utilizarea variabilelor, tipurilor de date şi structurilor de control.
2. Aplicarea principiilor programării orientate pe obiect (OOP):
    - Crearea şi gestionarea claselor şi obiectelor în C#.
    - Utilizarea moştenirii, încapsulării şi polimorfismului în dezvoltarea backend.
3. Utilizarea bazei de date Microsoft SQL (MS SQL):
    - Proiectarea şi crearea structurilor de bază de date.
    - Dezvoltarea interogărilor SQL pentru a extrage, actualiza şi gestiona datele.
    - Implementarea măsurilor de securitate şi gestionarea tranzacţiilor.
4. Integrarea C#, OOP şi MS SQL în dezvoltarea backend:
    - Dezvoltarea aplicaţiilor backend care folosesc C# pentru logica de afaceri.
    - Conectarea şi comunicarea cu baza de date MS SQL.
    - Asigurarea performanţei şi scalabilităţii aplicaţiei backend.
5. Testarea şi optimizarea soluţiilor dezvoltate:

- Realizarea testelor unitare și de integrare pentru asigurarea calității codului.
- Identificarea și remedierea problemelor de performanță și de securitate.

6. Documentarea și mentenanța sistemului dezvoltat:
   - Crearea documentației pentru a ajuta dezvoltatorii și administratorii sistemului.
   - Asigurarea mentenanței și actualizării continue a aplicației backend.

2 MODELAREA ŞI PROIECTAREA SISTEMULUI INFORMATIC

2.1 Descrierea tehnologiilor utilizate

În cadrul acestei abordări integrate a dezvoltării backend, se utilizează următoarele tehnologii și instrumente:

1. **C# (C-Sharp)**: Este un limbaj de programare modern, dezvoltat de Microsoft, utilizat pentru dezvoltarea aplicațiilor pe platforma .NET. C# oferă o sintaxă clară și robustă, suport pentru programarea orientată pe obiect și facilitează dezvoltarea aplicațiilor scalabile și securizate.
2. **Programarea Orientată pe Obiect (OOP)**: Este o paradigmă de programare care se concentrează pe organizarea codului în jurul obiectelor și a interacțiunilor între acestea. OOP facilitează dezvoltarea modulară și reutilizarea codului.
3. **Microsoft SQL (MS SQL)**: Este un sistem de management al bazelor de date relaționale dezvoltat de Microsoft. MS SQL oferă funcționalități avansate pentru stocarea și gestionarea datelor, inclusiv suport pentru tranzacții, securitate și optimizare a interogărilor.

Aceste tehnologii sunt integrate pentru a dezvolta soluții backend eficiente și scalabile, folosind C# pentru logica de afaceri, OOP pentru structurarea și modularizarea codului, și MS SQL pentru stocarea și gestionarea datelor. Această abordare integrată permite dezvoltatorilor să creeze aplicații backend puternice și fiabile într-un mediu de dezvoltare coerent.

2.2 Modelarea şi proiectarea

2.2.1. Language Basics - Sarcini

1. **Program de Validare Număr:**
   - **Input:** Număr de la tastatură.

- **Proces:** Verifică dacă numărul are minim 3 cifre, dacă nu, cere alt număr. Calculează valoarea în oglindă și verifică dacă este pătrat perfect.
- **Output:** Informații utilizator despre validitatea numărului și dacă este pătrat perfect.

2. **Program de Lucru cu Array-uri:**
   - **Input:** Listă de numere.
   - **Proces:** Salvează numerele într-un array, afișează numerele care nu sunt întregi și găsește cel mai mic număr.
   - **Output:** Numerele non-întregi și cel mai mic număr.
3. **Program de Analiză Nume:**
   - **Input:** 3 nume de persoane.
   - **Proces:** Analizează fiecare nume pentru a determina frecvența fiecărui caracter.
   - **Output:** Caractere și frecvențele acestora pe linii separate.

2.2.2 OOP Basics - Sistem de Clase pentru Animale şi Mâncare

- **Clase de bază:** Animal, Mâncare.
- **Subclase:** Carnivor, Erbivor, Omnivor, Carne, Planta.
- **Metode şi Proprietăţi:**
  - Clasele animal au metode pentru Mănâncă, Aleargă, Energie.
  - Metoda Mănâncă adaugă mâncare în stomac și afișează la consolă.
  - Metoda Energie calculează nivelul de energie al animalului.
  - Metoda Aleargă calculează timpul în care animalul parcurge o distanță dată.

2.2.3. Your first program architecture - Sistem de Clase pentru un Espressor

- **Clasa principală:** Espressor.
- **Componente:** Diferite componente care pot include un rezervor de apă, un măcinător de cafea, un încălzitor etc.
- **Interfaţă utilizator:** Metode publice pentru interacțiuni cum ar fi adăugarea apei, plasarea unei căni etc.
- **Entrypoint:** O modalitate de interacție cu utilizatorul la consolă cu opțiuni afișate într-o buclă.

2.2.4. Delegates & Events - Sistem de Clase pentru Gestionarea Produselor și Reducerilor

- **Clase:** Reducere, Producător, Produs, Pret, Catalog, Client.
- **Funcționalități:**
  - Reduceri aplicate produselor.
  - Evenimente pentru schimbarea prețului sau a stocului.
  - Metode pentru abonarea și dezabonarea clienților la catalog.
  - Metode pentru aplicarea reducerilor.

3 REALIZAREA SISTEMULUI INFORMATIC

3.1 Realizarea

Sarcina 1:

Funcția Task1 citește un număr de la tastatură și se asigură că are cel puțin 3 cifre. Dacă numărul introdus are mai puțin de 3 cifre, programul îi cere utilizatorului să introducă un alt număr. Odată ce un număr valid este introdus, funcția calculează valoarea "în oglindă" a numărului (ex: din 441 în 144) și verifică dacă acest număr oglindit este un pătrat perfect.

```
void Task1()
{
    Console.WriteLine("Enter number:");
    int number = int.Parse(Console.ReadLine());
    string numStr = number.ToString();

    while (numStr.Length < 3)
    {
        Console.WriteLine("You number is wrong:");
        Console.WriteLine("Enter number:");
        number = int.Parse(Console.ReadLine());
        numStr = number.ToString();
    }

    int oglindaNumber = ReverseNumber(numStr);
    Console.WriteLine("The number is: " + numStr);
    Console.WriteLine("Reversed number is : " + oglindaNumber);

    double result = Math.Sqrt(oglindaNumber);
    if (result % 1 == 0)
    {
        Console.WriteLine("Is perfect square:");
    }
    else
    {
        Console.WriteLine("Is not perfect square:");
    }

    int ReverseNumber(string numStr)
    {
        char[] charArray = numStr.ToCharArray();
        Array.Reverse(charArray);
        string reversedStr = new string(charArray);
        int reversedNum = int.Parse(reversedStr);
        return reversedNum;
    }
}
```

1.1 Funcția ReverseNumber pentru inversarea unui număr

Sarcina 2:

Funcţia Task2 citeşte o listă de numere de la tastatură, separate prin spaţiu, şi le salvează într-un array. Apoi, parcurge array-ul şi afişează numerele care nu sunt întregi. În plus, caută şi afişează cel mai mic număr din lista introdusă, fără a folosi funcţii predefinite din .NET cum ar fi Math.Min.

```
void Task2()
{
    Console.WriteLine("Enter number: ");
    string input = Console.ReadLine();
    string[] cuvinte = input.Split(' ');
    double[] numere = new double[cuvinte.Length];
    for (int i = 0; i < cuvinte.Length; i++)
    {
        numere[i] = Convert.ToDouble(cuvinte[i]);
    }
    Console.WriteLine("The numbers are: ");
    foreach (var numar in numere)
    {
        Console.WriteLine(numar);
    }

    void IntegerOrNot()
    {
        foreach (var numar in numere)
        {
            if (numar % 1 == 0)
            {
                Console.WriteLine("Integer: " + numar);
            }
            else
            {
                Console.WriteLine("Not integer: " + numar);
            }
        }
    }
    double searchMinNumber()
    {
        double minNumber = (double)numere[0];
        foreach (var numar in numere)
        {
            if (minNumber > numar)
            {
                minNumber = numar;
            }
        }
        Console.WriteLine("Min number is: " + minNumber);
        return minNumber;
    }
```

1.2 Citirea şi prelucrarea numerelor

Sarcina 3:

Funcţia Task3 citeşte trei nume de persoane de la tastatură, apoi afişează pentru fiecare nume ce caracter apare şi de câte ori apare, indiferent dacă litera este mare sau mică. Acest lucru este realizat folosind un Dictionary pentru a stoca frecvenţa fiecărui caracter.

```
void Task3()
{
    Console.WriteLine("Enter names: ");
    string input = Console.ReadLine();
    string[] names = input.Split(' ');
    foreach (var name in names)
    {
        string nume = name.ToLower();
        Dictionary<char, int> frecventaCaractere = new Dictionary<char, int>();
        foreach (char caracter in nume)
        {
            if (Char.IsLetter(caracter))
            {
                if (frecventaCaractere.ContainsKey(caracter))
                {
                    frecventaCaractere[caracter]++;
                }
                else
                {
                    frecventaCaractere[caracter] = 1;
                }
            }
        }
        Console.WriteLine(name);
        Console.WriteLine("Frecventa caracterelor:");

        foreach (var entry in frecventaCaractere)
        {
            Console.WriteLine($"{entry.Key}: {entry.Value}");
        }
    }
}
```

1.3 Frecvența caracterelor în nume

Sarcina 4: Crearea unei clase de bază pentru toate animalele

Pentru a îndeplini această sarcină, am creat o clasă de bază numită Animal. Această clasă are două proprietăți:

- Nume: un string care reprezintă numele animalului
- Energie: un int care reprezintă nivelul de energie al animalului

Sarcina 4.1: Crearea a două clase derivate din clasa de bază Animal

Pentru a îndeplini această sarcină, am creat două clase derivate din clasa de bază Animal:

- Carnivoră: o clasă pentru animale carnivore
- Erbivor: o clasă pentru animale erbivore

Aceste două clase moștenesc toate proprietățile clasei de bază Animal. De asemenea, au proprietăți suplimentare specifice tipului de animal:

- Carnivoră: o proprietate de tip string numită TipCarne care reprezintă tipul de carne pe care îl mănâncă animalul

- Erbivor: o proprietate de tip string numită TipPlanta care reprezintă tipul de plantă pe care o mănâncă animalul

Sarcina 4.2: Crearea unei clase pentru mâncare

Pentru a îndeplini această sarcină, am creat o clasă numită Mâncare. Această clasă are două proprietăți:

- Nume: un string care reprezintă numele alimentului
- Energie: un int care reprezintă cantitatea de energie pe care o oferă alimentul

Sarcina 4.3: Crearea unei clase de testare pentru a testa funcționalitatea claselor Animal, Carnivor, Erbivor și Mâncare

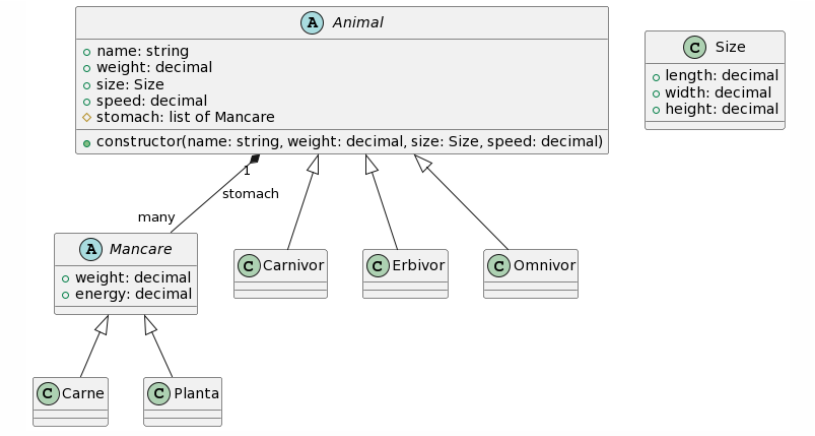
Pentru a îndeplini această sarcină, am creat o clasă numită Test. Această clasă conține metode pentru a testa următoarele funcționalități:

- Crearea de obiecte de tip Animal, Carnivor, Erbivor și Mâncare
- Modificarea proprietăților obiectelor de tip Animal, Carnivor, Erbivor și Mâncare
- Utilizarea obiectelor de tip Animal, Carnivor, Erbivor și Mâncare

Iată un exemplu de cod:

```
using System;
using System.Runtime.InteropServices;

namespace T1
{

    public struct dimensiune
    {
        public decimal lungime { get; set; }
        public decimal latime { get; set; }
        public decimal inaltime { get; set; }
        public dimensiune (decimal _lungime, decimal _latime, decimal _inaltime)
        {
            lungime = _lungime;
            latime = _latime;
            inaltime = _inaltime;
        }
    };
    public abstract class Animal
    { ... }
    public class Carnivor : Animal
    { ... }
    public class Erbivor : Animal
    { ... }
    public class Omnivor : Animal
    { ... }

    public abstract class Mancare
    { ... }
    public class Carne : Mancare
    {

    }
    public class Planta : Mancare
    {

    }
    public class Program
    { ... }
}
```

1.4 Realizarea 4



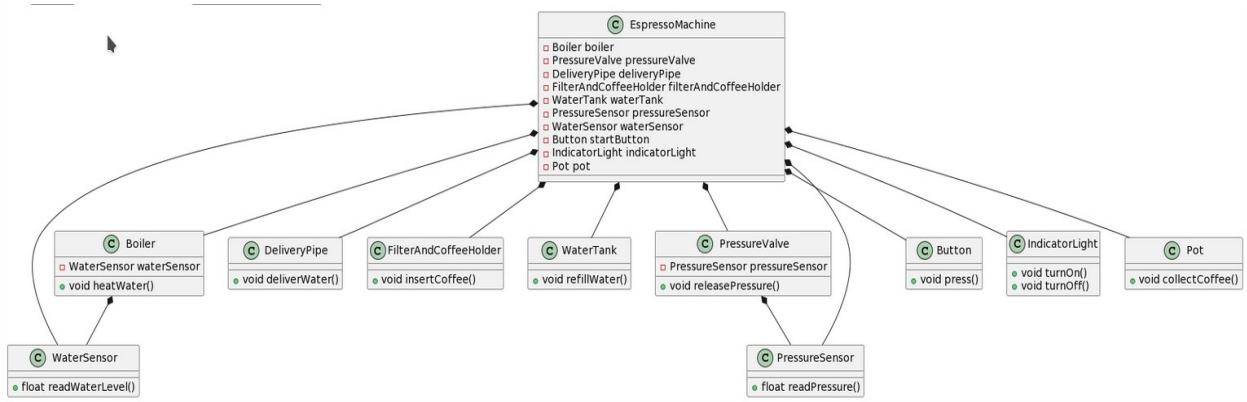
1.5 Diagrama UML

Sarcina 5 din programul de practică "Your first program architecture" durează 10 zile şi constă în crearea unui proiect pentru proiectarea unei ierarhii de clase ce alcătuiesc un Espressor. Candidaţii trebuie să utilizeze toate conceptele de Programare Orientată pe Obiect (POO) cunoscute pentru a crea această ierarhie, incluzând o clasă principală numită Espressor şi diverse componente. Proiectul trebuie să includă metode publice user-friendly pentru interacţiunea cu sistemul, iar structura sistemului trebuie să fie cât mai apropiată de limbajul natural şi de structura unui espressor fizic. În punctul de intrare al aplicaţiei, trebuie să se instanţieze Espressorul şi să se expună opţiuni de interacţiune la consolă pentru utilizator.

```
class WaterTank {…

2 references
class Boiler {…

2 references
class PressureValve {…

2 references
class DeliveryPipe {…

2 references
class FilterAndCoffeeHolder {…

2 references
class PressureSensor {…

2 references
class WaterSensor {…

2 references
class Button {…

2 references
class IndicatorLight {…

2 references
class EspressoMachine {…

0 references
class Program {…
```

1.7 Structura codului

Codul prezentat descrie funcţionarea unui aparat de făcut espresso, folosind principiile programării orientate pe obiect. Fiecare componentă a espressorului este reprezentată printr-o clasă distinctă, cum ar fi WaterTank, Boiler, PressureValve, etc. Aceste clase conţin metode care simulează diferite acţiuni ale componentelor reale, cum ar fi umplerea rezervorului cu apă, încălzirea apei şi eliberarea presiunii. Clasa principală, EspressoMachine, integrează toate aceste componente şi oferă o metodă MakeCoffee pentru a simula procesul de preparare a cafelei. Metoda MakeCoffee coordonează acţiunile diferitelor componente, verificând senzorii şi controlând luminile indicatoare pentru a finaliza procesul de preparare a cafelei. Programul este testat în metoda Main a clasei Program, unde se creează o instanţă a EspressoMachine şi se apelează metoda MakeCoffee.



1.8 Diagrama UML (Espressor)

Diagrama UML prezentată ilustrează arhitectura unei maşini de espresso, evidenţiind clasele şi relaţiile dintre acestea. Clasa centrală, `EspressoMachine`, conţine multiple instanţe ale altor clase care reprezintă diverse componente ale espressorului. De exemplu, `EspressoMachine` are un `WaterTank`, un `Boiler`, un `PressureValve`, şi aşa mai departe. Fiecare clasă componente are metode specifice, precum `refillWater()` pentru `WaterTank`, sau `heatWater()` pentru `Boiler`. Relaţiile sunt reprezentate prin linii care conectează `EspressoMachine` cu fiecare componentă, semnificând că `EspressoMachine` conţine sau foloseşte aceste componente în procesul de preparare a cafelei. Diagrama serveşte drept un ghid pentru structura de cod şi interacţiunile dintre obiectele dintr-un program de simulare a funcţionării unui espressor de cafea.

CONCLUZII

În concluzie, abordarea integrată a fundamentelor C#, programării orientate pe obiect (OOP) și utilizării bazei de date Microsoft SQL (MS SQL) în dezvoltarea backend reprezintă o strategie eficientă pentru crearea aplicațiilor backend robuste și scalabile. Această abordare oferă dezvoltatorilor un set de instrumente și cunoștințe necesare pentru a proiecta, dezvolta și întreține soluții software de înaltă calitate.

Prin combinarea puterii limbajului C# pentru implementarea logicii de afaceri, principiilor OOP pentru structurarea și modularizarea codului și MS SQL pentru gestionarea datelor, dezvoltatorii pot crea aplicații care îndeplinesc cerințele complexe ale industriei IT actuală. Această abordare permite dezvoltatorilor să creeze aplicații eficiente, ușor de întreținut și scalabile, esențiale pentru satisfacerea nevoilor organizațiilor și a utilizatorilor finali.

În plus, abordarea integrată promovează dezvoltarea unei gândiri sistemice și a unei baze solide de cunoștințe în domeniul dezvoltării software, ajutând astfel la dezvoltarea unor profesioniști în domeniu mai competenți și pregătiți să facă față provocărilor tehnologice din viitor.

În ansamblu, integrarea C#, OOP și MS SQL în dezvoltarea backend reprezintă o abordare esențială pentru dezvoltatorii software care doresc să creeze aplicații backend de succes în mediul IT în continuă evoluție.

BIBLIOGRAFIE

1. Program Practica Backend.pdf

2. C# 10 in a Nutshell

ANEXE

1. **GitHub Repository**: https://github.com/Xeoga/CodWer-Internship