**Universitatea de Stat din Moldova**

**MaTEMATICĂ SI INFORMATICĂ**

**INFORMATICĂ**

**DASCĂL ALEXANDRU**

**INTERACȚIUNEA CU BAZELE DE DATE IN CADRUL LIMBAJULUI C#**

**CIFRUL ŞI DENUMIREA SPECIALITĂŢII**

**Teză de licenţă**

Sef Catedră **\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

(semnătura)

Conducător ştiinţific: **\_\_\_\_\_\_\_\_\_\_\_\_\_ Arnaut, Vsevolod, gradul ştiinţific**

(semnătura)

Autorul: \_\_\_\_\_\_\_\_\_\_\_\_\_\_

(semnătura)

**Chişinău-2016**

CUPRINS

[Lista Abrevierilor 4](#_Toc451669733)

[INTRODUCERE 5](#_Toc451669734)

[I. TEHNOLOGIILE .NET PENTRU BAZELE DE DATE 7](#_Toc451669735)

[1.1 Concepte fundamentale a modelelor de stocare a datelor 7](#_Toc451669736)

[1.2 Tehnologia ADO.NET 11](#_Toc451669737)

[1.1.1 ADO.NET metoda conectării permanente 16](#_Toc451669738)

[1.1.2 ADO.NET metoda desconectării 17](#_Toc451669739)

[1.3 Entity Framework 18](#_Toc451669740)

[II. CAPITOLUL 21](#_Toc451669741)

[2.1 Proiectarea Aplicației 21](#_Toc451669742)

[2.1.1 Structura Organizatiei 21](#_Toc451669743)

[2.1.2 Cerintele fata de sistemul informatic 22](#_Toc451669744)

[BIBLIOGRAFIE 25](#_Toc451669745)

# Lista Abrevierilor

C# - c șarp

ADO.NET -

IT – Tehnologii informationale

UDA – Universal Data Acces

ORM – Object Relational Mapper

DRSA – Directie raionala pentru siguranta alimentelor

SI – sistem Informatic

SQL – Standard Language Query

LINQ – Language Integrated Query

# INTRODUCERE

Lucrarea dată este o analiză a metodelor de utilizare a bazelor de date cu ajutorul limbajului C#. Evidențiind avantajele utilizării bazelor de date în favoarea altor posibilități de stocare a datelor în format electronic, evoluția metodelor din limbajul C# pentru utilizarea bazelor de date și argumentarea celei mai eficiente metode și des utilizate în elaborarea softurilor, partea practică constă în elaborarea unui aplicații în care se utilizează una din tehnologiile descrise.

Acestă temă are o importanță majoră, deoarece aplicațiile ce sunt realizate în zilele noastre au nevoie de metode și spații de stocare a datelor care pe parcursul utilizarii aplicației este nevoie de adăugare, ștergere, reactualizare, si efectuarea altor acțiuni care necesită interacțiunea peramanentă cu datele stocate ale aplicației, la fel pe parcursul utilizării aplicației numarul datelor stocate crește foarte repid, ceia ce necesită spații mai largi de stocare, acești doi factori numarul mare de date și volumul mare de memorie ocupată duc la frînarea aplicației și îneficienței acestiea.

Pentru soluționare acestei probleme, cea mai eficientă metodă de stocare a datelor este utilizarea unei baze date. Bazele de date au ca principii structurarea datelor pentru a modela aspecte ale realitații cu ajutotul entitaților și legaturilor dintre ele, acest fapt duce la eliminarea repetării informațiilor și eficientizarea întretineri stocurilor mari de informații. Din aceste motive majoritatea aplicațiilor care necesită stocarea și utilizarea volumurilor mari de informații utilizeaza ca metode de stocare a datelor bazele de date.

Teza dată are două capitole, primul este capitolul teorectic și capitolul doi este un capitol practic în care este realizată o aplicație ce utilizeaza tehnologiile analizate în primul capitol.

Primul capitol al acestei teze are ca scop o analiza ampla a factorilor care favorizeaza stocarea datelor în baze de date, analiza elementelor componente ale unei baze de date, un alt aspect analizat este crearea corecta a bazei de date în concordanță cu anumite reguli și pași.

În partea a doua a acestui capitol se analizează evoluția modalitaților de utilizare a bazei de date cu ajutorul limbajului C#, se accentuiază tehnologiile ADO.NET care este un set de spații de denumiri ce conțin clase cu ajutorul carora se realizeaza conectarea la baza de date, extragerea datelor necesare, salvarea modificarilor și deconectarea de la baza de date, cea mai importantă tehnologie analizată este tehnologia Entity Framework care este cea mai utilizată și ofera cele mai mari facilitați pentru lucrul cu bazele de date, este și tehnologia preferată de majoritatea companiilor IT și este cea mai recomandata de catre specialisti.

Ultimul capitol este capitolul practic care are ca scop implementarea unei tehnologii analizate în primul capitol, această tehnologie este Entity Framework.

# TEHNOLOGIILE .NET PENTRU BAZELE DE DATE

## Concepte fundamentale a modelelor de stocare a datelor

Pentru a înțelge necesitatea și importanța modelelor de structurare și stocare a datelor se va analiza noțiunile de dată și informație ce au semnificații diferite. *Datele* sunt fapte colectate din viaţa bazate pe observaţii și măsurători. *Informaţia* este rezultatul prelucarii și interpretării datelor de către un subiect în dependența de subiect datele prelucrate pot oferi inforamții diferete. Datele au un caracter obiectiv, în timp ce informaţia este subiectivă. Pentru a putea obține inforamații cit mai corecte datele trebuie colectate, memorate, organizate și prelucrate, pentru efectuarea acestor acțiuni deobicei se utilizează fișierele și bazele de date.

Organizarea datelor înseamnă definirea şi structurarea datelor în memoria internă sau în memoria externă (în colecţii) şi stabilirea legăturilor între date, conform unui model de date. În decursul dezvoltării metodelor de structurare au fost utilizate mai multe concepte, prezentate in tabelul 1.1

[2] Tabelul 1.1 *Evolutia sistemelor de stocare a datelor*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Etape | Pina la 1960 | Anii 1960 | Anii 1970 | Sfirșitul 1970 pîna acum | Sfirșitul 1980  pîna acum |
| Mod de organizare | Fișiere secvențiale | Fișiere secvențiale indexate | Baze de date arbosescene | Baze de date relaţionale | Baze de date orientate obiect |

[1] Pe parcursul dezvoltării modulelelor de organizare a datelor, inițial datele se stocau în fișiere. Acest model de stocare a datelor permite stocarea datelor în mai multe fișiere.

Pentru o orgnizație se generaza fișiere diferite pentru fiecare department, în loc ca aceseste date sa fie stocate centralizat. De exemplu:

Departamentul Contabilitate stochiaza:

(Nume, Prenume, Adresă, DataNaşterii, Salariu, NumărSecţie)

Departamentul Resurse Umane stochează:

(Nume, Prenume, Adresă , DataNaşterii, NumărTelefon, Funcţie, NumărSecţie)

Se observă o dublare a datelor în acest exemplu, defapt dublarea datelor exista în toate sistemele bazate pe fișiere pe lînga acest neajuns acest model de organizare are mai multe limitari:

1. *separarea şi izolarea datelor* are ca efect faptul că, pentru anumite aplicaţii, este necesară prelucrarea şi sincronizarea a două sau mai multe fişiere.
2. *dublarea datelor* implică două dezavantaje:
   1. risipa de spaţiu de stocare
   2. posibila alterare a integrităţii datelor
3. *dependenţa de date* implică modificarea tuturor programelor de aplicaţie în cazul modificării tipului sau dimensiunii unui câmp de date, activitate ce necesită mult timp şi este supusă apariţiei erorilor.
4. *incompatibilitatea fişierelor,* structura fişierelor este încorporată în programele aplicaţie, ea este dependentă de limbajul în care sunt scrise acestea.
5. *Interogarea fixă a programelor aplicaţie* a fost generată de faptul că sistemele bazate pe fişiere sunt în mare măsură dependente de programatorul de aplicaţii. Interogările sau rapoartele necesare trebuie să fie scise de către acesta. Din acest motiv apăr două situaţii:
   1. tipurile de interogări sau rapoarte care pot fi realizate sunt fixe şi nu exista nici un fel de facilitate de a realiza interogări neplanificate.
   2. apăritia unui mod specific pentru fiecare organizatie de scriere în fișiere si a aplicațiilor

Toate limitările tratării bazate pe fişiere se datorează următorilor doi factori:

1. Definiţia datelor este încorporată în programele aplicaţie, în loc de a fi stocată separat şi independent
2. Nu există un control al accesului şi manipulării datelor, în afară de cel impus de către programele aplicaţie.

Pentru a evita toate aceste probleme ce apar în urma utilizării unui sistem bazat pe fișiere, cel mai eficient este utilizarea unei baze de date.

*Baza de date* este o colecţie partajată de date elementare sau structurate, între care există relaţii logice, mai detaliat este un depozit de date unic, care este definit o singură dată şi este utilizabil simultan de mai mulţi utilizatori, toate datele sunt integrate, cu o dublare minimă și este o resursă comună, partajată, ea conţine nu numai datele operaţionale, ci şi o descriere a acestora.

Caracteristica principală a aplicaţiilor de baze de date constă în faptul că accentul este pus pe operaţiile de memorare şi regăsire efectuate asupra unui volum mare de date şi mai puţin asupra operaţiilor de prelucrare a acestora. Principala operaţie care apare în aplicaţiile de baze de date este regăsirea datelor în scopul obţinerii de informaţii din baza de date. Utilizarea unei baze de date ofera urmatoarele avantaje:

1. *Centralizarea datelor* ofera urmatoarele avantaje:
   1. *Reducerea redundanţei datelor memorate*. În cazul sistemelor bazate pe fişiere, este posibil ca aceleaşi date să apară de mai multe ori în fişiere diferite, aparţinând unor aplicaţii diferite.
   2. *Evitarea inconsistenţei datelor memorate*: atunci când există mai multe copii ale aceleiaşi date este posibil ca prin actualizarea numai a unora dintre ele, să existe valori diferite pentru una şi aceeaşi dată. Aceasta atrage după sine inconsistenţa bazei de date.
   3. *Posibilitatea partajării datelor*, se referă nu numai la posibilitatea utilizării în comun a datelor de către mai mulţi utilizatori, ci şi la posibilitatea de a dezvolta aplicaţii noi folosind datele deja existente în baza de date.
   4. *Posibilitatea aplicării restricţiilor de securitate*: existând controlul centralizat al datelor, se pot introduce verificări de autorizare a accesului la date. Se pot impune restricţii diferite pentru fiecare tip de acces (regăsire, actualizare, ştergere), pentru fiecare tip de dată şi la nivelul fiecărui utilizator.
   5. *Posibilitatea introducerii unor proceduri de validare a datelor* în cazul operaţiilor de actualizare, pentru a asigura cerinţa ca baza de date să conţină date corecte.

2*. Independenţa între date şi program*. Baza de date, ca imagine a unei anumite realităţi, trebuie actualizată permanent. Acest lucru nu trebuie să afecteze aplicația. Pentru aceasta trebuie ca fiecare program să aibă o viziune proprie asupra bazei de date.

3. *Realizarea legăturilor între entităţile de date*, necesare pentru exploatarea eficientă a bazei de date.

4. *Integritatea datelor asigură fiabilitatea şi coerenţa bazei de date*. Pentru aceasta trebuie definite restricţii de integritate cum ar fi:

* Apartenenţa la o listă de valori sau la un interval
* Apartenenţa la un anumit format
* Reguli de coerenţă cu alte date, aceste reguli trebuie să respecte atât aspectele statice, cât şi cele dinamice. De exemplu, după o actualizare de creştere a salariului, nivelul acestuia trebuie să fie mai mare decât cel precedent.

5*. Securitatea datelor*. Baza de date trebuie să fie protejată împotriva unei distrugeri logice (anomalii de actualizare) sau fizice. Pentru aceasta, există instrumente care permit:

* Crearea unor *puncte de repriză* ce reprezintă salvarea din timp în timp a unor copii coerente ale bazei de date
* Gestiunea unui *jurnal de tranzacţii* ce este lista operaţiilor realizate asupra bazei de date după ultimul punct de repriză. Dacă apare o anomalie care ar putea distruge baza de date, pe baza copiei de la ultima repriză şi a jurnalului de tranzacţii se poate reface baza de date în forma iniţială.

6. *Confidenţialitatea datelor* este asigurată prin proceduri de:

* Identificare a utilizatorilor prin nume sau cod
* Autentificarea prin parole
* Autorizarea accesului diferenţiat prin drepturi de creare, consultare, modificare sau ştergere pentru anumite sectoare de date

*7. Partajarea datelor* permite înlănţuirea tranzacţiilor solicitate simultan pe aceeaşi înregistrare din baza de date, prin blocarea cererilor în aşteptare şi deservirea ulterioară a acestora.

[9] Unul dintre cele mai importante și des utilizate sisteme de baze de date este sistemul de baze de date relational. Modelul relaţional a fost propus de către IBM şi a revoluţionat reprezentarea datelor făcând trecerea la generaţia a doua de baze de date. Modelul este simplu, are o solidă fundamentare teoretică fiind bazat pe teoria seturilor (ansamblurilor) şi pe logica matematică. Pot fi reprezentate toate tipurile de structuri de date de mare complexitate, din diferite domenii de activitate. Modelul relaţional este definit prin: structura de date, operatorii care acţionează asupra structurii şi restricţiile de integritate.

## Tehnologia ADO.NET

[4] În zielele de astăzi, datele sunt păstrate în diferite formate, variind de la date păstrate în baze de date pînă la informații salvate în documente Word, mesaje de pe poșta electronică și multe alte tipuri. ADO sau denumirea din limba engleză ActiveX Data Objects este o tehnologie de acces a datelor ce simplifică utilizarea lor de la diferiți furnizori, eliberînd astfel programatorii de învățarea tehnologiilor pentru utilizarea datelor de la diferiți furnizori și dezvoldînt aplicații nedependente de furnizorul de date. Filosofia celor de la Microsoft din spatele ecestei tehnologii este ( UDA ) Accesul Universal al Datelor în engleza Universal Data Access. UDA nu este o tehnologie ci mai degrabă o strategie pentru a rezolva problema accesului la date, al cărui scop este un acces eficient și puternic de date, indiferent de sursa de date sau de limbaj de dezvoltare. Mai mult decât atât, acest acces universal are menirea de a elimina nevoia transformării datelor existente dintr-un format în altul. Cum sa specificat mai sus ADO este o tehnologie independentă față de limbajul de programare. Acest lucru înseamnă că, indiferent de limbajul de dezvoltare a aplicației, Visual Basic, VBScript, Visual Basic for Applications (VBA), Visual C ++, Visual J ++, sau JavaScript - interfața de dezvoltare este identică.

Odată cu apriția platformei .NET dezvoltată la fel de către cei de la Microsoft a fost schimbată și tehnologia ADO, care din ADO a devenit ADO.NET și bine înțeles cu multe alte îmbunătațiri care au fost făcut în aceasta tehnologie.

[5] Tabel 1.2 *Evolutia tehnologiei ADO*

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Anul lansării | 1996 | 1998 | 2000 | 2002 | 2004 | 2008 | 2010 |
| Versiunea tehnologiei | ADO 1.0 | ADO 2.1 | ADO 2.5 | ADO.NET | ADO.NET 2.0 | ADO.NET  3.5 | ADO.NET 4.0 |

[3] Platforama .NET definește un set de spații de denumiri și clase care fac perte din tehnologia ADO.NET ce permite interațiunea cu bazeele de date cum sunt Microsoft SQL Server, Oracle Server, MySQL Server și altele. ADO.NET are la bază două metode de a interacționa cu baza de date, prima fiind conectarea permanentă la baza de date și a doua este conectarea doar în cazul reînoirii datelor.

ADO.NET nu are un sigur set de spații de denumiri și clase care comunică cu toate tipurile de sisteme de baze de date. Dar oricum, ADO.NET suportă mai mulți furnizori, fiecare furnizor avînd un spațiu de denumiri cu clase similare și fiind specifice fiecărui sistem, pe lînga spațiile de denumiri pentru fiecare furnizor ADO.NET conține și un spațiu de denumiri *System.Data.Common* cu toate componenetele cumune ale tuturor furnizorilor.

Primul beneficiu al acestei abordări este posibilitatea de a utiliza spațiul de denumiri pentru un furnizor spefic și a utiliza toate funcționalitățile spefice acestui sistem.

Tabel 1.3 *Principalele componente comune ale sistemelor de baze de date*

|  |  |  |  |
| --- | --- | --- | --- |
| Tipul obiectului | Clasa de baza | Interfața | Rolul obiectului |
| Connction | DbConnection | IDbConnection | Oferă posibilitatea de conectare la și deconectare de la sistemul de date. |
| Command | DbCommand | IDbCommand | Reprezintă o interogare SQL sau o procedură stocastică. De asemenea, oferă acces la datele citite de la furnizor |
| DataReader | DbDataReader | IDbDataReader | Oferă posibilitatea de citire unidirecțională a datelor direct de la server |
| DataAdapter | DbDataAdapter | IDbDataAdapter | Administrează transferurile de date între aplicație și sistemul de date. DataAdaptor conține o conexiune și un set de patru obiecte pentru a selecta, însera, actualiza și șterge datele din sistemul de date. |
| Parameter | DbParameter | IDbParameter | Reprezintă un parametru definit pentru o interogare |
| Transaction | DbTransaction | IDbTransaction | Reprezintă incapsularea unei tranzacții |

În dependență de furnizorul de date elementele au denumiri specifice, dar implementeză și derivă de la aceleași clase de baze specificate mai sus (ex. SqlConnection, OracleConnection, OdbcConnection, MySqlConnection) acest lucru permite utilizarea diferitor furnizori utilizînd aceleași metode de utilizare.

Cum sa specificat mai sus ADO.NET este un set de spații de denumiri care ne permite lucrul cu bazele de date. În continuare se vor analiza doar cele mai importante spații de denumiri și rolul acestora

Tabel 1.4 *Principalele spatii de denumiri din ADO.NET*

|  |  |
| --- | --- |
| Spațiul de denumiri | Rolul spațiului de denumiri |
| Microsoft.SqlServer.Server | Acest spațiu de denumiri furnizează tipurile care facilitează CLR și Microsoft SQL Server |
| System.Data | Acest spațiu de denumiri definește tipurile de bază ADO.NET utilizate de către toți furnizorii de date, inclusiv interfețe comune și numeroase tipuri care reprezintă stratul deconectat (de exemplu, DataSet și DataTable). |
| System.Data.Common | Acest spațiu de denumiri conține tipuri partajate între toți furnizorii de date ADO.NET, inclusiv clasele abstracte de bază comune. |
| System.Data.SqlTypes | Acest spațiu de denumiri conține tipuri de date native folosite de Microsoft SQL Server. Se pot utiliza întotdeauna tipurile de date CLR corespunzătoare, dar sunt SQLTYPE-uri optimizate pentru a lucra cu SQL Server (de exemplu, în cazul în care baza de date SQL Server conține o valoare întreagă, aveți posibilitatea să o reprezentați folosind fie int sau SqlTypes.SqlInt32). |

Dintre toate spațiile de denumiri din ADO.NET, spațiul de denimuri *System.Data* este numitorul comun pentru toate funcționalitățile oferite de ADO.NET. Nu se pot construi aplicațiile utilizînd ADO.NET fără a specifica acest spațiu de denumiri. Acest spațiu de denumire conține tipurile care sunt partajate între toate sistemele de baze de date care se utilizează cu ajutorul tehnologiei ADO.NET.

În plus acest spațiu de denumiri conține și un set de excepții specifice în utilizarea bazelor de date (de exemplu, NoNullAllowedException, RowNotInTableException). System.Data conține tipuri care reprezintă componente a bazei de date (de exemplu tabele, rânduri, coloane și constrângeri), precum și interfețe comune implementate de clasele specifice ficărui sistem de date implementat în ADO.NET. În Tabelul 1.5 sunt reprezentate principalele componente ale acestui spațiu de denumiri System.Data

Tabel 1.5 *Componentele principale ale System.Data*

|  |  |
| --- | --- |
| Tipul | Rolul |
| DataSet | Reprezintă baza de date construită din un oarecare număr de obiecte de tipul DataTable ce reprezintă tabelele, toate acestea sunt păstrate în memoria cache. |
| DataTable | Reprezintă un tabel din baza de date |
| DataColumn | Reprezintă o coloană a unui tabel de tip DataTable |
| DataRow | Reprezintă o linie dintrun obiect DataTabel |
| Constraint | Reprezintă o constringere asupra unei obiect de tip DataColumn |
| DataRelation | Reprezintă relația copil părinte dintre daouă obiecte de tip DataTable |
| IdataAdapter | Definește funcționalitățile unui obiect de tipul DataAdapter |
| IdataReader | Definește funcționalitățile unui obiect de tipul DdataReader |
| IdbCommnad | Definește funcționalitățile unui obiect de tipul Command |

Pentru a putea utiliza toate aceste componente oferite de System.Data primul pas ce trebuie făcut este conectarea la baza de date, pentru a face acest lucru fiecare sistem de baze de date are implementat o clasa specifică Connection în dependență de furnizor ea are denumire difereită (ex. SqlConnection, OracleConnection, MySqlConnection ), funcționalitățile de bază ale acestei clase care sunt comune sunt definite de interfața:

*public interface IDbConnection : IDisposable*

*{*

*string ConnectionString { get; set; }*

*void Close();*

*void Open();*

*}*

Acestea sunt doar cîteva din cele mai importante proprietăți și funcționalități ale acestei interfețe, metoda Open() și Close() , deschid și respectiv închid conexiunea la baza de date, specificare bazei de date se face cu ajutorul proprietății ConnectionString care reprezintă un șir de caracte unde sunt specificate numele bazei de date și numele serverului. În exemplu urmator este realizat conectarea la un anumit tip de sistem de baze de date și anume Microsoft SQL Server

*SqlConnection connection = new SqlConnection (@"Data Source = (local)\SQLEXPRESS; Initial Catalog = USM; Integrated Security = True");*

Această metodă de conectare face aplicația construită dependenată de un anumit sistem de baze de date, în cazul în care este necesar schimbarea altui sistem de date va fi nevoie de schimbat o mare parte din cod în dependență de noul sistem de date, pentru a ușura acest lucru se paote folosi spațiul de denumiri System.Data.Common ce conține un set de clase și interfețe ce ne permite să tratăm unele componente polimorfic, în cazul abordat se va proceda în modul următor, în fișierul \*.config se poate specifica tipul sistemului de baze de date și stringul de conecatere.

*<appSettings>*

*<add key="provider" value="System.Data.SqlClient" />*

*<add key="cnStr" value= "Data Source=(local)\SQLEXPRESS;*

*Initial Catalog=USM; Integrated Security=True"/>*

*</appSettings>*

Acesta este un fragment de fișier de tip XML ce conține unele configurații ale aplicației, cu ajutorul unor clase specifice se pot extrage unele date în momentul cînd deja aplicația este pornită.

*string dataProvider = ConfigurationManager.AppSettings["provider"];*

*string connectionString = ConfigurationManager.AppSettings["cnStr"];*

*DbProviderFactory dataFactory = DbProviderFactories.GetFactory(dataProvider);*

*using (DbConnection connection = dataFactory.CreateConnection())*

*{*

*connection.ConnectionString = connectionString;*

*}*

În acest fragment de cod C# se extrag din fisierul \*.config cu ajutorul clasei ConfigurationManager numele sistemului de date și stringul de conectare după care se crează obiectul DbConnection care oferă toate funcționalitățile pentru a lucra cu baza de date.

### ADO.NET metoda conectării permanente

În cele descrise mai sus au fost atinse daor componentele comune ale celor doua metode de utilizare a bazelor de date utilizînd tehnologia ADO.NET, prima fiind conectarea permanentă la baza de date iar a doua metoda disconecătrii, care constă în conectare doar în cazul cînd sunt extrase date din baza de date și salvate în memoria cashe și atunci cînd se modifică datele din baza de date propriuzisă.

Pentru utilizare metodei conectării permanentă mai întii este nevoie de a crea o conexiune utilizînd una din metodele descrie mai sus, principalul component pentru utilizarea acestei metode este clasa DataReader care se obține cu ajutorul clasei Command care oferă poșibilitatea de a executa cereri SQL asupra bazei noastre de date și în urma execuției se obține un obiect de tip DataReader ce ne permite să accesăm rezultatul cererii SQL executată. În figura de mai jos Fig. 1.1 este reprezentat arhitectura acestei metode:

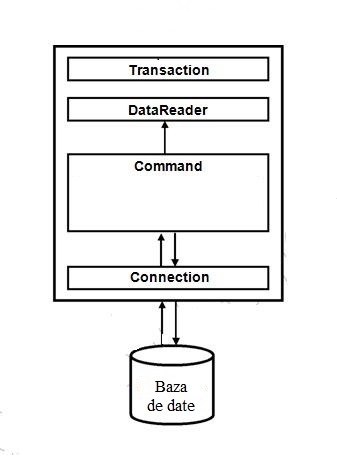


Fig. 1.1 Arhitectura ADO.Net Conectat [6]

Această metodă ne permite să avem toate tiprile de comenzi SQL precum de selecție, înserare, actualizare și stergere, doar în urma selecției vom obține un obiect DataReader, care accesează datele direct din baza de date, utilizînd metoda Read() pe care o are acestă clasă, vom obține cîte un rînd din toate care sau obținut în urma unei cereri SQL.

Această metodă de utilizare a bazei de date este eficientă în cazuri în care este nevoie de iterat rapid asupra unui set mare de date și nu este nevoie de a păstra toate datele în memoria aplicației, un aspect negativ este că trebuie să avem o conectare permanentă la baza de date plus trebuie să deschidem și să închidem conecțiunea, ceeace în metoda desconectării acest lucru este efectuat de către sistem.

### ADO.NET metoda desconectării

Conceptul care stă la baza acestei metode și de unde vine denumirea ei este de a optimiza numărul de cereri asupra bazei de date, dacă în metoda conectării permanente conexiunea este deschisă și inchisă de către programator, atunci în metoda desconectării aceste acțiuni sunt efectuate automat de către sistem. Inițial datele necesare sunt copiate în memoria aplicației și apoi de fiecare dată cînd este nevoie de a reactualiza datele din baza de date sau din aplicație, se deschide o conexiune, se efectuiează toate acțiunile necesare după care conexiunea se închide, acesta metoda înlătură necesitatea de a fi conectați permanent la baza de date.

În tabelul 1.5 sau enumerat cîteva tipuri de date din spațiul de denumiri System.Data, o mare parte din ele sunt utilizate în acestă metodă, aceste tipuri sunt DataSet, DataTable, DataColumn, DataRow, Constrints, DataRelation etc. Toate aceste tipuri sunt reprezentarea bazei de date în aplicație, DataSet reprezintă baza de date propriuzisă, iar DataTable tabelele din baza de date și respectiv celelalte clase sunt alte componente ale baze de date, cu ajutorul acestui set de clase se pot efectua diferite acțiuni asupra bazei de date. În figura 1.2 reprezintă arhitectura acestei metode.

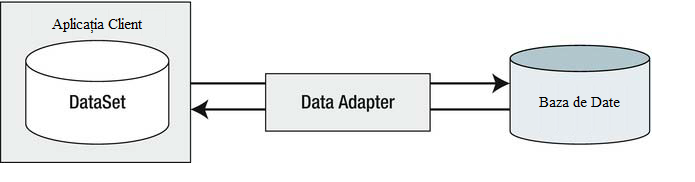


Figura 1.2 *Arhitectura metodei desconectării*

Clasa care este responsabilă de sincronizarea datelor din aplicație și a celor din baza de date este DataAdapter, în Tabelul 1.6 sunte reprezentate principalele metode care sunt necesare pentru efectuarea sincronizării.

Tabel 1.6 *Principalele metode și propreietăți ale clasei DataReader*

|  |  |
| --- | --- |
| Metoda | Rolul |
| Fill() | Această metodă execută o comandă SQL SELECT (așa cum este specificat de către proprietatea SelectCommand) pentru a interoga baza de date și a înscri datele într-un obiect DataTable. |
| SelectCommand | Interogare de tipul SQL SELECT care se execută cu ajutorul metodei Fill() |
| InsertCommand | Interogare SQL INSERT care se execută cu ajutorul metodei Update() |
| UpdateCommand | Interogare SQL UPDATE care se execută cu ajutorul metodei Update() |
| DeleteCommand | Interogare SQL DELETE care se execută cu ajutorul metodei Update() |
| Update() | Execută comenzile specificate de proprietățile  InsertCommant, UpdateCommand și DeleteCommand |

Următorul fragment arată cum pot fi utilizate metodele enumerate în Tabelul 1.6, crearea unui obiect DataSet ce reprezintă baza de date și apoi popularea acesteia, utilizînd un obiect DataReader

*// Crearea unui obiect DataSet*

*DataSet dataSet = new DataSet("USM");*

*// Crearea unui obiect DataAdapter*

*SqlDataAdapter dataAdapter = new SqlDataAdapter("Select \* From USM", ConnectionString);*

*// Apelarea metodei Fill*

*dataAdapt.Fill(dataSet, "USM");*

## Entity Framework

Majoritate aplicațiilor elaborate folosesc baze de date în care sunt stocate, prelucrate datele necesare pentru aplicație, de fiecare dată cînd este necesar adaugarea sau prelucrarea dateleor este nevoiea de a elabora metode care vor realiza aceste funcționalități, pentru realizarea acestor metode se consumă foarte mult timp, pe lînga elaborarea funcționalității aplicațiilor care este defapt scopul principal în elaboararea unei aplicații. Utilizînd tehnologia ADO.NET mentionată mai sus va fi nevoie de a scris foarte mult cod care de multe ori este foarte dificil si costisitor de mentinut. Pentru a simplifica modalitatea de utilizare a bazelor de date, se paote utiliza arhitectura reprezentata în figura 1.3

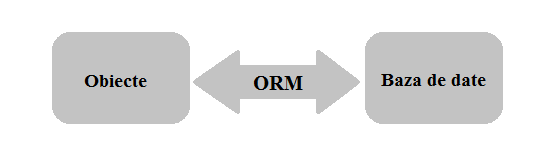


Figura 1.3 ORM

Această arhitectură stabilește relația dintre obiectele și clasele care sunt similare cu tabelele și colaonele din baza de date, ceia ce ne ofera posibilitatea de a lucra cu clase și obiecte fara a utiliza o sumedenie de clase și interfețe adiționale pentru a ne conecta și apoi desconecta de la baza de date, plus de a scri interogari SQL direct în codul aplicației. În anul 2008 Microsoft lansează ORM-ul Entity Framework care are rolul de a simplifica utilizarea bazelor de date, acestă noua tehnologie are un succes foarte mare și a reusit să înlocuiască utilizarea tehnologiei ADO.NET, ultima versiune EF 6 a reușit să adune pîna în prezent peste 11 milioane de descărcări, iar ultima subversiune EF 6.1.3 peste 750 000 de descărcări doar în primele 3 luni. În tabelul 1.7 sunt reprezentate toate versiunile Entity Framework lansate de către Microsoft.

Tabel 1.7 *Versiuni Entity Framework [8]*

|  |  |  |
| --- | --- | --- |
| Virsiunea Entity Faramework | Versiunea .Net Framework | Anul lansarii |
| Enity Framework v1.0 | .Net Framework 3.5 | 2008 |
| Enity Framework v4.0 | .Net Framework 4.0 | 2010 |
| Enity Framework v5.0 | .Net Framework 5.5 | 2012 |
| Enity Framework v6.0 | .Net Framework 4.6 | 2013 |

Pe parcursul dezvoltării acestei tehnologii, pîna în prezent sa ajuns la trei metode de utilizare a bazelor de date folosind Entity Framework, prima posibilitate oferită în versiununea v1.0, bineînțeles care a evoluat foarte mult pîna la versiunea curentă v.6.0, este crearea mai întii a bazei de date și apoi integrarea acesteia.

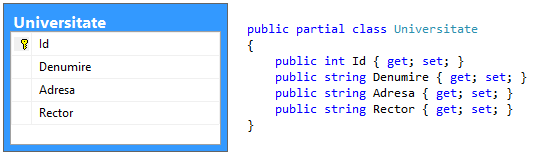


Figura 1.4

Întegrarea consta in maparea bazei de date cu clasele și obiectele aplicației, în figura 1.4 este reprezentată maparea unui tabel din baza de date cu o clasa care este identică cu tabelul respectiv, denumirile coloanelor fiind proprietățile clasei create, iar denumira tabelului numele clasei.

Cea mai utilizată metoda însă este crearea claselor în aplicație și apoi generarea bazei de date de la clasele create. Acest lucru se face cu ajutorul unei clase abstracte *DbContext* în înteriorul carea se efectuiază conxiunea la baza de date și generarea bazei de date propriuzise. Această metoda ne permite ca aplicația noastră sa nu conțină clase elaborate pentru a putea lucra cu baza de date, doar sa creem clase ce vor reprezenta entitatile aplicației.

A treia metoda care are la baza metoda crearii claselor și apoi generarea bazei de date este crearea modelului grafic și apoi generarea bazei de date. Aceasta metoda ne permite crearea bazei de date fara a scri vreo careva clasa doar de a genera tabel și crea legaturi dintre acestea, în spatele acestei metoda defapt se genereaza un cod XML, din care apoi se genereaza clasele și se revine la metoda a doua din clasele create se genereaza cod SQL care și crează baza de date mapata cu clasele din aplicație.

Toate tabelele care sunt generate se genereaza conform unor convenții stabilite, pentru schimbarea acestor conventii sunt folosite doua tehnologii Data Anotation și Fluent API.

Pentru ultimile doua metode mentionate apare notiunea de migratie care reprezintă defapt schimabrea structurii claselor ce reprezintă tabelele din baza de date.

# CAPITOLUL

Acest capitol reprezintă partea practică a tezei care consta în realizarea unie aplicații care folosește tehnologiile menționate in primul capitol. Aplicatia a fost realizata in urma consultarii si intelegerii cu utilizatorii acesteia care sunt angajatii Direcției Raională pentru Siguranta Alimentelor Cimișlia (DRSA Cimișlia) care nu dispunea de careva produse soft pentru inregistrarea autorizatiilor sanitar veterinare de functionare (Anexa 1).

## Proiectarea Aplicației

Sa gasit necesar implemntarea unui sistem informatic deoarece in urma eliberarii acestor autorizatii datele sunt pastrate in forma scrisa, in caiete sau alte metode de pastrare a datelor, dar nicidecum cel electronic.

Evident toate aceste informatii devin vulnerabile si pot fi foarte usor de pierdut, deasemenea angajatii au obligatia de a face rapoarte, rapoartele includ in sine si informatii despre acesta autorizatie.

Pentru solutionarea acestei probleme cea mai eficienta solutie este bineinteles realizarea unui produs soft ce ar perminte angajatilor realizarea acetui proces la calculator si salvarea datelor in format electoronic. Salvarea datelor in format electornic reduce riscul pierderii acestor informatii, deasemenea permite cu usurinta gasirea lor dupa anumite criterii precum sortarea si generearea rapoartelor in diferite formate fie documente sau excel.

## Structura Organizatiei

Agentie regionala DRSA Cimislia o responsabila de reglementarea si control pentru sigutanta alimentelor si in domeniul sanitar-veterinar, zootehnic, al protectiei plantelor si carantinei fitosanitare, controlul semincer, calitatii produselor primare, produselor alimentare si a hranei pentru animale.Agentia dispune de o serie de drepturi care sunt puse practica prin angajatii sai.

* *Sa dispuna accesul* inspectorilor aflati in exercitiu, conform legislaţiei în vigoare, în incintele şi pe teritoriul staţiilor de cale ferată, aeroporturilor, staţiilor auto, oficiilor poştale, elevatoarelor, depozitelor, în vagoanele de marfă şi de pasageri, autobuze, pe aeronavele civile şi pe teritoriul întreprinderilor, instituţiilor şi organizaţiilor cu orice tip de proprietate, care produc, achiziţionează, prelucrează şi comercializează producţie şi materiale supuse supravegherii şi controlului în domeniile vizate;
* *Sa interzica*

1. importul, comercializarea şi utilizarea produselor de uz fitosanitar şi a fertilizanţilor, produselor farmaceutice de uz veterinar care nu corespund standardelor naţionale, reglementărilor tehnice şi altor acte tehnico-normative;
2. prelucrarea şi comercializarea produselor agroalimentare şi folosirea apei în cazul depistării de reziduuri de produse de uz fitosanitar, fertilizanţi, reziduuri de hormoni şi produse farmaceutice de uz veterinar ce depăşesc limitele maxim admisibile;
3. utilizarea produselor genetic modificate etc.

* *Sa solicite eliberarea din functie a persoanelor* antrenate in activitati cu produse de us fitosanitar si cu fertilizanti care nu au fost supuse examenului medical si nu au fost instruite in privinta securitatii muncii in cadrul acestor activitati etc.



Figura 2.1 *Organigrama Directiei sanitar veterinare [10]* In figura 2.1 este reprezentata organigrama institutiei ce reprezinta o prezentare mai ampla structurii orgnizatorice.

Seful subdiviziunii are trei sefi adjuncti, consacrati activitatilor dupa cum urmeaza

* Pentru problem sanitar-veterinare.
* Pentru problem fitosanitare si controlul calitatii.
* Pentru problemele sigurantei alimentelor.

### 2.1.2 Cerintele fata de sistemul informatic

Destinatia acestui sistem informatic este organizatia DRSA Cimislia, pentru care va fi necesar dezvoltarea unui soft, care va avea o serie de functionalitati accesibile utilizatorilor:

* Colectarea, stocare datelor introduse de catre utilizator.
* Viziualizarea datelor.
* Filtrarea datelor.
* Generarea rapoartelor.

Exista doar un tip de utilizatori care va avea o singura interfata de acces:

* Angajatii.

Dezvoltarea acestui SI solutioneaza urmatoarele probleme:

* Siguranta datelor.
* Vizualizarea rapida a datelor.
* Cautarea dupa anumite criterii.
* Generarea rapoartelor.

Tabel 2.2 *Cerinte functionale*

|  |  |  |
| --- | --- | --- |
| Nr. | Descrierea Cerintei | Prioritatea |
|  | Posibilitatea de a introduce si salva datele autorizatiei | inalta |
|  | Avertizarea utilizatorului cind datele pentru salvare nu sunt complete | inalta |
|  | Posibilitatea de asterge toate datele inserate, dar nesalvate | medie |
|  | Vizualizarea datelor salvate | inalta |
|  | Cautarea autorizatiilor pe anumite perioade de timp | inalta |
|  | Cautarea autorizatiilor dupa numele subdiviziunii teritoriale | medie |
|  | Cautarea dupa alte criterii | medie |

*Cerinte nefunctionale:*

1. Cerinte fata de confidentialitate:

* Datele din baza de date nu trebuie sa fie disponibile utlizatorilor doar administratorului bazei de date.
* Aplicatia nu trebuie sa de datele persoanei care a alibrat autorizatia.

1. Cerinte fata de interfata utilizator:

* Casetele pentru introducerea aplicatiei sa nu aiba fonturi stridente.
* Textul sa fie de dimensiuni citibile.
* Dimensiunea ferestri sa fie suficinet de mare pentru a putea introduce, vizualiza datele.

1. Cerinte fata de operabilitate:

* Orice actiune sa nu dureze mai mult de 3 secunde.
* Intorducerea datelor si vizualizarea datelor sa fie separate.
* In cazul generarii unui raport numele fisierului trebui sa fie ales de utilizator.
* Adresa de salvare a raportului trebuie sa fie ales de utilizator.

1. Cerinte fata de fiabiliate:

* In cazul deconectarii datele intorduse sa nu fie salvate, se salveaza doar in momentul cind utilizatorul va da acordul.

## Realizarea Aplicatiei

In urma analizei cerinteleor inaintate de catre utlizatorii aplicatiei sa decis pentru realizarea acestui soft sa se utilizeze urmatoarele limbaje si tehnologii, limbajul de programare C#, tehnologia Windows Forms deoarece toate aceste mecanisme permit realizarea softului cerut foarte rapid si calitativ.

Crearea si interactiunea cu baza de date a fost realizata cu ajutorul tehnologiei Entity Framework si anumecode first care ne permite generarea bazei de date de la o structura de clase care sunt create in limbaj, ceia ce ne permite flexibilitatea bazei de date in conformitate cu modificarile aplicatiei, pentru aceasta aplicatie a fost utilizata versiunea 6.1.3 care este ultima versiune lansata, instalarea acestei tehnologii este farte simpla, mediul de dezvoltare utilizat Visul Studio dispune de un set foarte larg de instrumente pentre care NuGet, acest instrument penrmite instalarea foarte rapida a tuturor teh ologiilor existente pentru un anumit limbaj sau platforma de dezvoltare a aplicatiilor.

Solutia aplicatiei este formata din doua proiecte deoarece separarea realizarii aplicatiei in mai multe compartimente permite dezvoltarea si mentinerea aplicatiei mai eficient fara a implica toate compartimentele aplicatiei. Primul compartiment este *Authorization.Data* in care sunt realizatea toate actiunile ce tin de datele eplicatiei inclusiv interactiunea cu baza de date. Proiuectul al doilea este *Authorization.WinForms* care este proiectl in care sa realizat logica aplicatiei si prelucrearea datelor si interfatea utilizatorului.

Proiectul ce tine de datele aplicatiei este conpus din 4 clase, dintre care 5 dintre ele reprezinta tabelele din baza de date iar una din clase reprezinta baza de date sau in termentii tehnologiei date contextual bazei de date. Clasele respective sunt *ActivityProfile, Agent, Authrization, Subdivision.* Un element foarte important este realizarea legaturilor corecte dintre tabelele unei baze de date, in structura creata se intilnesc doua tipuri de legaturi 1-n si m-n lse sunt reprezentate in figura 2.2, pentru realizarea acestor legaturi sunt respectate urmatoarelel reguli. Pentru legatura 1-n, de exemplu clasele *Authorization* si *Agent*, in care *Agent* este legatura 1 iar *Authorization* n.

|  |  |
| --- | --- |
| 1 | n |
| *public class Authorization*  *{*  *[ForeignKey(nameof(Agent))]*  *public int AgentId {get;set;}*  *public Agent Agent {get;set;}* | *public class Agent*  *{*  *public ICollection<Authorization> Authorizations {get;set;}*  *}* |

Clasa Authorization are doua proprietati care sunt strict legatea de realizarea legaturii cu clasa Agent, proprietatea Agent este referinta obiectului cu care se face legatura si respectiv proprietatea AgentId este id-ul acestui obiect, aceste doua proprietati fac legatura de tip 1, in clasa Agent pentru realizarea legaturii de tip n este o colectie de obiecte de tip Authrization.

Pentru realizarea legaturii de tip n-m nu este nevoie de a crea careva clase intermediare, in baza de date automat se va genera un table ce va realiza legatura de tip n-m, in aplicatia realizata legatura n-m este intre clasele ActivityProfile si Authorization.

|  |
| --- |
| n |
| *public class Authorization*  *{*  *public virtual ICollection<ActivityProfile> ActivityProfiles { get; set; }*  *}* |
| m |
| *public class ActivityProfile*  *{*  *public virtual ICollection<Authorization> Authorizations { get; set; }*  *}* |

In exemplul dat legatura este realizata prin proprietati ce reprezinta cite o colectie de obiecte de tipul respectiv, in bza de date va fi creat un tabel intermediar cu ajutorul caruia se va efectua legatura. Clasa care zeprezinta baza de date in aplicatie este clasa *AuthorizationDbContext* care mosteneste clasa abstracta *DbContext* din spatiul de denumire System.Data.Entity care contine un set de metode care permit manipularea bazei de date. Proprietatile acestei clase sunt respectiv tabelele din baza de date care sunt reprezentate de un set de colectii de tipul DbSet care reprezinta un tabel.

*public class AuthorizationDbContext : DbContext*

*{*

*public DbSet<Authorization> Authorizations { get; set; }*

*public DbSet<Subdivision> Subdivisions { get; set; }*

*public DbSet<Agent> Agents { get; set; }*

*public DbSet<ActivityProfile> ActivityProfiles { get; set; }*

*}*

In exemplul de mai sus este reprezentata clasa *AuthorizationDbContext* care are 4 proprieteti de tipul DbSet care sunt tabelele din baza de date. Un alt element foarte important pentru aceasta tehnologie sunt migratiile ce reprezinta schimbarile bazei de date care sunt realizate prin modificarea contextului, modificari precum adaugarea, stergerea claselor si proprietatilor. Pentru a utilizarea migratiilor sunt utilizate trei comenzi:

1. Enable-Migrations, permite utilizarea migratiilor si se utilizeaza doar o singura data in proiect.
2. Add-Migration, de fiecare data dupa efectuarea unui modificari aceasta comanda va adauga schimbarea respectiva.
3. Update-Database, aseasta comanda sincronizeaza modificarile din aplicatie si baza de date.

In momentul crearii unei migratii se genereaza o clasa ce mosteneste clasa abstracta DbMigration

*public partial class InitialCreate : DbMigration*

*{*

*public override void Up()*

*{*

*CreateTable("dbo.ActivityProfiles", c => new*

*{*

*Id = c.Int(nullable: false, identity: true),*

*Activity = c.String(),*

*})*

*.PrimaryKey(t => t.Id);*

*}*

*public override void Down()*

*{*

*DropTable("dbo.ActivityProfiles");*

*}*

*}*

Aceasta clasa contine doua metode, in prima metoda Up() se genereaza tot codul ce va efectua modificarile in baza de date conform schimbarilor aparute in contextul dat, metoda Down() este codul ce anuleaza toate schimbarile din metoda Up(), metoda Down() se aplica in momentul cind este nevoia de a reveni la o anumita stare a bazei de date, respectiv toate migratiile aplicate vor putea fi anulate cu ajutorul acestei metode

In figura 2.2 este reprezentata structura bazei de date a acestei aplicatii care a fost generata in urma crearii claselor in proiectul Authorization.Data tabelele date au denumirile claselor la plural

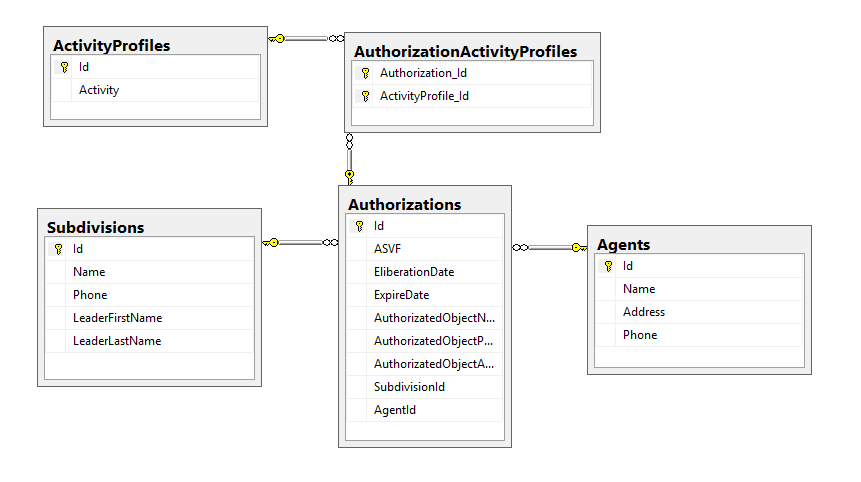


Figura 2.2 Structura bazei de date a aplicatiei

In proiectul Authorization.WinForms interactiunea cu baza de de date se realizeaza prin intermediul unui obiect de tipul *AuthorizationDbContext,* prelucrarea datelor din baza de date se paote realiza utilizind 3 metode:

1. SQL.
2. LINQ.
3. Metode LINQ.

In aplicatia data a fost utilizate metode LINQ care permit interogarea bazei de date cu ajutorul unor metode specifice limbajului, iar in spatele lor se genereaza cereri SQL care sunt optimizate si permit obtinerea rezultatului cu eficienta si performanta maxima.

# BIBLIOGRAFIE

1. Angel CAŢARON, (2004), Baze de Date, extras 3 mai 2016 din sursa

<http://vega.unitbv.ro/~cataron/Courses/BD/BD_Cap_1.pdf>

1. <http://bd.ase.ro/uploads/bd_curs/Curs_BD_Lungu,Botha.pdf>
2. Pro C# 5.0 and The .NET 4.5 Framework, Andrew Troelsen, p.807
3. <https://www.safaribooksonline.com/library/view/ado-activex-data/1565924150/ch01.html>
4. <https://www.pluralsight.com/courses/entity-framework5-getting-started>

<http://programcall.com/4/adonet/connected-architecture-of-adonet.aspx>

1. <https://www.udemy.com/entity-framework-tutorial/learn/v4/overview>
2. <https://www.pluralsight.com/courses/entity-framework-6-getting-started>
3. http://www.seap.usv.ro/~valeriul/lupu/cafec/capitolul1.pdf
4. <http://www.ansa.gov.md/ro/structuri-teritoriale.html>
5. http://www.ansa.gov.md/uploads/files/Siguranta/Autorizatie%20San-Vet\_Functionare.pdf

**Anexa 1. Autorizație Sanitară Veterinară de functionare**

