Documentație

Sistem de gestiune clinică veterinară – VetLife

Realizat de:

Buzan Alexandra-Maria (1311A)

Enache Maria (1311A)

Moisoiu Andreea-Nicola (1311A)

Putină Iustina-Elena (1312A)

Universitatea Tehnică „Gheorghe Asachi ” Iași

Facultatea de Automatică și Calculatoare

Mai, 2025

1. Software Requirements Specification

Pentru

Sistem de gestiune clinică veterinară – VetLife

1. Introducere
   1. Scop

Scopul acestui proiect este de a defini cerințele software pentru aplicația VetLife, o soluție desktop destinată gestionării activităților unei clinici veterinare. Documentul este adresat dezvoltatorilor, oferind o bază clară pentru proiectare, implementare și verificare. Aplicația urmărește să permită introducerea și organizarea eficientă a datelor despre proprietari, animale și consultații medicale, într-un mediu accesibil.

* 1. Domeniul de aplicare

VetLife este o aplicție desktop dezvoltată în C# cu Windows Forms, având o arhitectură modulară formată din trei componente: VetLife.Core, VetLife.DataAccess și VetLifeApp. Aplicația oferă funcționalități precum: adăugarea de proprietari, introducerea datelor despre animale, înregistrarea consultațiilor veterinare și vizualizarea acestora. Sistemul este destinat exclusiv personalului medical și administrativ al clinicii și este gândit pentru utilizare locală, fără conexiune la internet.

* 1. Definiții, acronime și abrevieri
* SRS – Specificația cerințelor software
* UI – Interfață cu Utilizatorul
* CRUD – Create, Read, Update, Delete (operații de bază asupra datelor)
* DLL – Dynamic-Link Library (biblioteci externe .NET)
* SQLite – Sistem de gestiune a bazelor de date locale, folosit pentru stocarea datelor
* VetLifeApp – Proiectul care conține interfața grafică (Windows Forms)
  1. Referințe
* IEEE 830- 1998 – Standardul pentru documentația SRS
* Exemple și resurse de la cursul de Ingineria Programării
* Documentația oficială .NET ( <https://learn.microsoft.com/ro-ro/> )
  1. Prezentare generală a documentului

Documentul este structurat conform standardului IEEE și oferă o prezentare detaliată a aplicației VetLife. După această introducere, capitolul 2 descrie arhitectura sistemului și funcționalitățile generale, iar capitolul 3 prezintă cerințele specifice ale sistemului, inclusiv cerințele funcționale pentru module precum gestionarea proprietarilor, animalelor și consultațiilor.

1. Descriere generală
   1. Perspectiva asupra produsului

VetLife este o aplicație software independentă, dezvoltată pentru a sprijini activitatea unei clinici veterinare. Ea nu este parte a unui sistem mai mare, dar este structurată modular, respectând principiile separării logicii aplicației pe mai multe niveluri.

Aplicația este împărțită în trei componente (proiecte) distincte:

* VetLife.Core – conține clasele care definesc modelul de date (exemple: Proprietar, Animal, Consultație);
* VetLife.DataAccess – se ocupă de comunicarea cu baza de date SQLite (repository-uri);
* VetLifeApp – conține interfața grafică și logica de prezentare, implementată în Windows Forms
* Această arhitectură permite o mai bună organizare a codului, reutilizare și mentenanță pe termen lung.
  1. Funcțiile produsului

Aplicația VetLife oferă următoarele funcționalități principale:

* Autentificare utilizatori pe baza numelui și parolei;
* Administrarea utilizatorilor (adăugare, ștergere, roluri diferite), posibilă doar de către administrator;
* Introducerea datelor unui nou proprietar (nume, prenume, telefon, email) cu validări și verificări duplicate;
* Introducerea datelor despre un animal de companie asociat unui proprietar ( nume, specie, rasă, vârsta, greutatea și numele proprietarului);
* Înregistrarea unei consultații veterinare, cu detalii despre data efectuării consultului, simptomele observate, diagnosticul stabilit și tratamentul recomandat;
* Vizualizarea listei complete a consultațiilor efectuate;
* Atribuirea unor roluri diferite utilizatorilor (administrator, medic veterinar, asistent, operator) cu acces diferențiat la funcționalități.
  1. Caracteristicile utilizatorilor

Sistemul permite autentificarea utilizatorilor cu roluri bine definite. După autentificare, funcționalitățile disponibile sunt controlate în funcție de rol:

* Administrator – are acces complet la toate modulele: proprietari, animale, consultații, personal;
* Medic veterinar – poate vizualiza consultațiile pacienților;
* Asistent – poate adăuga proprietari, animale și consultații;
* Operator (recepționer) – poate introduce și vizualiza consultațiile;

Fiecare utilizator are o experiență minimă de bază în utilizarea calculatorului. Interfața a fost gândită să fie intuitivă și ușor de folosit.

* 1. Constrângeri generale
* Aplicația trebuie să ruleze pe sistem de operare Windows (minim Windows 7), cu .NET Framework 4.7.2 instalat;
* Baza de date utilizată este SQLite, cu fișier local (clinica.db) care se modifică în timpul rulării;
* Nu este permisă inserarea de date duplicate în tabelele critice (exemplu: doi proprietari cu același telefon sau email);
* Interacțiunea se face doar prin interfața grafică locală (Windows Forms).
  1. Presupuneri și dependențe
* Se presupune că utilizatorul are un calculator cu drepturi de administartor și poate rula aplicații .NET;
* Nu se folosesc servicii externe (API-uri sau internet);
* Toate datele se salvează local, iar aplicația nu are nevoie de conexiune permanentă;
* Fișierul de bază de date clinica.db este distribuit odată cu aplicația și poate fi modificat doar prin acestea.

1. Cerințe specifice
   1. Cerințe privind interfețele externe
      1. Interfețe cu utilizatorul

Aplicația oferă o interfață grafică simplă și intuitivă, construită cu ajutorul tehnologiei Windows Forms. Utilizatorii pot interacționa cu aplicația prin intermediul ferestrelor, butoanelor, câmpurilor de introducere text și meniurilor. Fiecare secțiune a aplicației este grupată logic: datele pacientului, consultațiile și datele proprietarului sunt separate vizual și funcțional.

* + 1. Interfețe hardware

Aplicația nu are cerințe hardware speciale. Ea necesită doar un sistem compatibil cu Windows care suportă .NET Framework 4.7.2 sau o versiune ulterioară.

* + 1. Interfețe software

Aplicația este construită pe .NET Framework și utilizează biblioteca System.Data.SQLite pentru interacțiunea cu baza de date. Nu sunt necesare alte aplicații sau servicii externe pentru funcționarea sistemului.

* + 1. Interfețele de comunicație

VetLife este o aplicație locală și nu necesită conexiune la internet sau interacțiuni în rețea. Toate datele sunt salvate în fișierul clinica.db pe sistemul local.

* 1. Cerințe funcționale
     1. Modul: Gestionare proprietari
        1. Adăugare proprietar
* Descriere: Utilizatorul poate introduce un nou proprietar în baza de date a clinicii veterinare.
* Date introduse: Nume, Prenume, Telefon, Email.
* Rezultat așteptat: Proprietarul este salvat în baza de date dacă toate datele sunt valide și nu există deja un proprietar identic.
* Precondiții:
* Toate câmpurile sunt completate.
* Numele și prenumele conțin cel puțin 2 caractere.
* Telefonul are minimum 7 cifre, fără litere sau caractere speciale.
* Emailul este într-un format valid.
* Postcondiții:
* Proprietarul este adăugat în tabela Proprietari.
* Un mesaj de confirmare este afișat și formularul se resetează.
* Dacă datele sunt duplicat (aceeași combinație de nume+prenume, telefon sau email), salvarea este blocată.
  + - 1. Validarea datelor introduse
* Cerință funcțională: Verificarea completitudinii și corectitudinii câmpurilor.
* Descriere: Aplicația validează fiecare câmp înainte de a salva datele în baza de date.
* Condiții de validare:
* Niciun câmp nu trebuie să fie gol sau format doar din spații.
* „Nume” și „Prenume” trebuie să aibă minimum 2 caractere.
* „Telefon” trebuie să fie format doar din cifre și să aibă cel puțin 7 caractere.
* „Emailul” trebuie să respecte formatul standard (ex: [user@domeniu.com](mailto:user@domeniu.com))
* Nu trebuie să existe deja un alt proprietar cu aceleași informații.
* Rezultat așteptat: Dacă datele nu sunt valide sau sunt duplicate, utilizatorul primește un mesaj de eroare și salvarea este oprită.
  + 1. Modul: Gestionare animale

3.2.2.1 Adăugare animal

* Descriere: Utilizatorul poate introduce un nou animal în sistemul clinicii veterinare.
* Date introduse: Nume, Specie, Rasă, Vârstă, Greutate, Proprietar(selectat din listă).
* Rezultat așteptat: Animalul este salvat în baza de date și este asociat cu un proprietar valid.
* Precondiții:
* Toate câmpurile sunt completate.
* Există cel puțin un proprietar în baza de date.

Postcondiții:

* Animalul este adăugat în tabela Animale cu ProprietarId asociat.
* Un mesaj de confirmare este afișat.

3.2.2.2 Validarea datelor introduse

* Cerință funcțională: Verificarea câmpurilor completate
* Descriere: Aplicația validează corectitudinea datelor introduse înainte de salvare.
* Condiții de validare:
* Niciun câmp nu trebuie să fie gol.
* Câmpul „Vârstă” trebuie să conțină un umăr întreg pozitiv.
* Câmpul „Greutate” trebuie să fie un număr real pozitiv.

Rezultat așteptat: Dacă datele sunt invalide, utilizatorul primește un mesaj de eroare și salvarea este blocată.

* + - 1. Asocierea unui proprietar

Cerință funcțională: Alegerea unui proprietar existent

Descriere: Utilizatorul selectează un proprietar existent dintr-o listă derulantă(ComboBox).

Precondiții: Lista de proprietari este încărcată automat la pornirea aplicației.

Rezultat așteptat: La salvarea animalului, ProprietarId este inclus în înregistrare.

* + 1. Modul: Gestionare consultații

3.2.3.1 Adăugare consultație

* Descriere: Utilizatorul poate înregistra o nouă consultație medicală pentru un animal existent în sistem.
* Date introduse:
  + Animal (selectat din listă)
  + Data consultației (selectată automat ca data curentă, editabilă)
  + Simptome (textbox, obligatoriu)
  + Diagnostic (textbox)
  + Tratament (textbox)
* Rezultat așteptat: Consultația este salvată în baza de date și asociată cu animalul selectat.
* Precondiții:
* Există cel puțin un animal înregistrat în sistem
* Câmpul "Simptome" este completat
* Animalul este selectat din listă
* Postcondiții:
* Consultația este adăugată în tabela Consultatii cu AnimalId asociat
* Un mesaj de confirmare este afișat
* Formularul este resetat pentru o nouă introducere

3.2.3.2 Validarea datelor introduse

* Cerință funcțională: Verificarea corectitudinii datelor consultației
* Descriere: Sistemul validează datele înainte de salvare
* Condiții de validare:
  + Animalul trebuie selectat din listă
  + Câmpul "Simptome" nu poate fi gol
  + Data consultației nu poate fi din viitor (opțional)
  + Dacă sunt completate, Diagnostic și Tratament trebuie să aibă minimum 2 caractere
* Rezultat așteptat: La date invalide, se afișează mesaj de eroare și salvarea este blocată

3.2.3.3 Asocierea cu un animal

* Cerință funcțională: Legătura consultației cu un animal existent
* Descriere: Utilizatorul selectează un animal din listă derulantă care afișează informații combinate (nume, specie, rasă)
* Precondiții:
  + Lista de animale este încărcată la deschiderea formularului
  + Animalele sunt afișate în format: "Nume (Specie, Rasă, Vârstă ani)"
* Rezultat așteptat: La salvare, AnimalId este inclus în înregistrarea consultației

3.2.3.4 Vizualizare consultații

* Descriere: Afișarea tuturor consultațiilor înregistrate
* Date afișate:
  + Data consultației
  + Numele animalului
  + Simptome
  + Diagnostic
  + Tratament
* Funcționalități:
  + Sortare după dată (descrescător)
  + Filtrare după animal (opțional)
  + Actualizare automată la adăugare nouă consultație

3.2.3.5 Feedback pentru utilizator

* La salvare reușită:
  + Mesaj "Consultație înregistrată cu succes!"
  + Golire automată a câmpurilor (exceptând data curentă)
* La erori:
  + Mesaje specifice pentru fiecare caz de validare eșuată
  + Mesaj de eroare pentru probleme la salvarea în baza de date
  + Captură de excepții cu detalii pentru depanare

3.2.3.6 Secvență operații

1. Selectare animal din ComboBox
2. Completare simptome (obligatoriu)
3. Completare diagnostic și tratament (opțional)
4. Validare dată (implicit data curentă)
5. Apăsare buton "Salvează consultația"
6. Validare câmpuri
7. Verificare precondiții
8. Salvare în baza de date
9. Afișare rezultat operatie

3.2.3.7 Excepții

* Animal negăsit: Dacă animalul selectat nu există în baza de date
* Date invalide: Când validările eșuează
* Eroare SQL: La probleme de conexiune sau interogare
* Acces neautorizat: Pentru utilizatori fără drepturi de medic/administrator

3.2.3.8 Reguli de afișare

* Lista consultațiilor este ordonată cronologic (cele mai recente primele)
* Data este afișată în format "dd.MM.yyyy"
* Animalele sunt grupate după proprietar în listă (opțional)
* Consultațiile fără diagnostic completat sunt marcate vizual
  + 1. Modul: Gestionare utilizatori

3.2.4.1 Autentificare

* Descriere: La pornirea aplicației, utilizatorul este întâmpinat de o fereastră de autentificare.
* Date introduse: Nume utilizator și parolă.
* Precondiții:
* Utilizatorul trebuie să existe în baza de date.
* Utilizatorul trebuie să aibă o parolă corectă.

Postcondiții:

* Dacă autentificarea este validă, utilizatorul accesează aplicația în funcție de rolul său. În caz contrar, este afișat un mesaj de eroare.

3.2.4.2 Administrarea utilizatorilor

* Descriere: Administratorul are acces la un formular special unde poate:
* Vizualiza lista utilizatorilor
* Adăuga un utilizator nou (nume, parolă, rol)
* Șterge un utilizator
* Precondiții: Numai administratorul are acces la această funcționalitate.
* Postcondiții: Modificările se salvează în baza de date.

* + - 1. Asocierea unui proprietar

Cerință funcțională: Alegerea unui proprietar existent

Descriere: Utilizatorul selectează un proprietar existent dintr-o listă derulantă(ComboBox).

Precondiții: Lista de proprietari este încărcată automat la pornirea aplicației.

Rezultat așteptat: La salvarea animalului, ProprietarId este inclus în înregistrare.

* 1. Cerințe de performanță

Operațiile de salvare și afișare trebuie să aibă loc instantaneu (maxim 1-2 secunde). Lista proprietarilor și consultațiilor trebuie să se încarce complet fără întârzieri perceptibile pentru utilizator.

* 1. Constrângeri de proiectare

Aplicația trebuie să respecte structura pe layere: Core, DataAccess, UI. Totodată, nu este permis accesul direct din UI către baza de date; toate operațiile trec prin repository-uri. Baza de date SQLite trebuie să fie inclusă în soluție și accesată prin DatabaseHelper.

* 1. Atribute

Aplicația este fiabilă, cu tratament pentru erori (excepții capturate în try-catch). Validările sunt făcute atât la nivel de interfață, cât și in codul sursă (repository). În același timp, interfața este ușor de folosit, organizată logic și accesibilă.

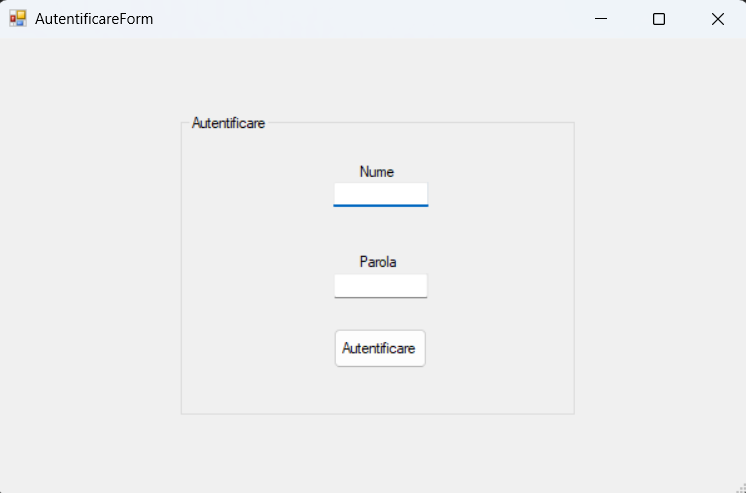
* 1. Alte cerințe

Aplicația trebuie să poată rula portabil, fără instalare complexă (doar cu fișiere binare). Baza de date trebuie să fie inclusă și să permită testarea rapidă de către evaluator.

Modul de utilizare a programului

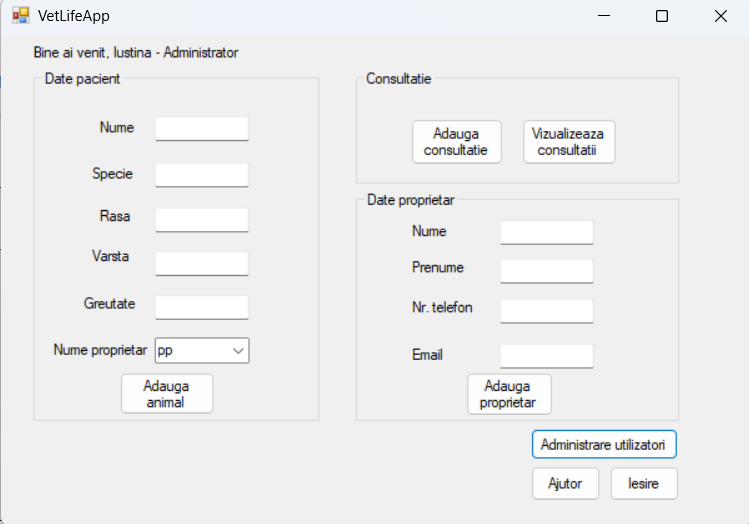
La deschiderea aplicației, utilizatorul este întâmpinat de o fereastră de autentificare, unde trebuie să introducă numele de utilizator și parola aferentă. Acest formular este esențial pentru a controla accesul în aplicație și pentru a încărca interfața corespunzătoare rolului avut. Astfel:

* Administratorul
* Poate adăuga noi utilizatori (nume, rol, parolă)
* Poate șterge utilizatori existenți
* Are acces complet la toate funcționalitățile aplicației.
* Medic veterinar
* Nu poate gestiona utilizatorii
* Are acces limitat, poate vizualiza doar consultațiile
* Asistentul
* Nu poate gestiona utilizatorii
* Poate adăuga animale sau consultații
* Recepționer
* Nu poate gestiona utilizatorii
* Poate adăuga animale și consultații, dar nu poate adăuga proprietari sau personal



Pentru a accesa funcționalitățile aplicației, se completează câmpul „Nume” cu numele de utilizator și câmpul „Parolă” asociată contului și mai apoi se apasă butonul „Autentificare” pentru validare. Dacă datele introduse sunt corecte, utilizatorul este redirecționat către interfața principală. În caz contrar, va fi afișat un mesaj de eroare.

După autentificare, utilizatorul este redirecționat către fereastra principală a aplicației VetLife, unde sunt disponibile funcționalități specifice rolului său. În colțul din stânga sus este afișat mesajul de întâmpinare, cu numele și rolul utiilizatorului curent.



Interfața este împărțită în patru secțiuni funcționale:

* 1. Date pacient

Această secțiune permite introducerea unui nou animal în baza de date. Utilizatorul trebuie să completeze: numele animalului, specia (ex: câine, pisică), rasa, vârsta (număr întreg), greutatea (număr zecimal), proprietarul (selectat dintr-o listă derulantă oferită de baza de date). După completarea câmpurilor, se apasă butonul „Adaugă animal”, iar datele sunt validate și salvate în baza de date.

* 1. Date proprietar

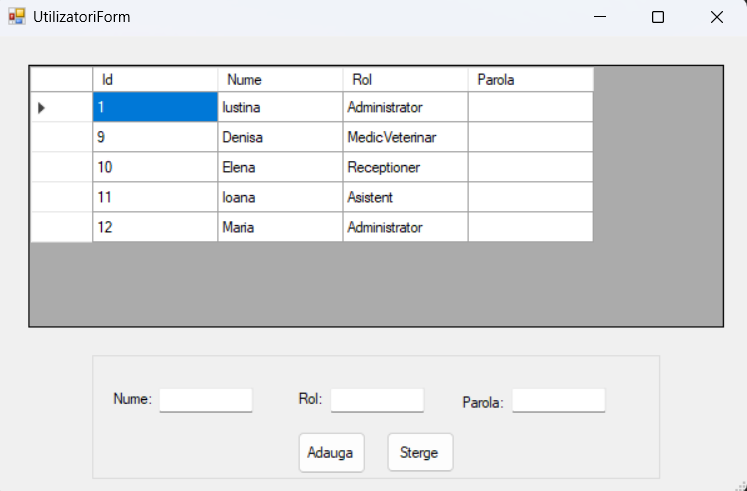
Permite înregistrarea unui nou proprietar. Se introduc: nume, prenume, număr de telefon, email. După apăsarea butonului „Adaugă proprietar”, aplicația verifică dacă există deja un proprietar cu aceleași date (nume+prenume, telefon, email). În caz contrar, datele sunt salvate și un mesaj de confirmare este afișat.

* 1. Consultații

În această zonă se regăsesc două opțiuni:

* „Adaugă consultație” – deschide o fereastră unde se poate înregistra o consultație pentru un animal existent.
* „Vizualizează consultații” – afișează toate consultațiile existente din sistem, într-un format ușor de citit.
  1. Administrare utilizatori

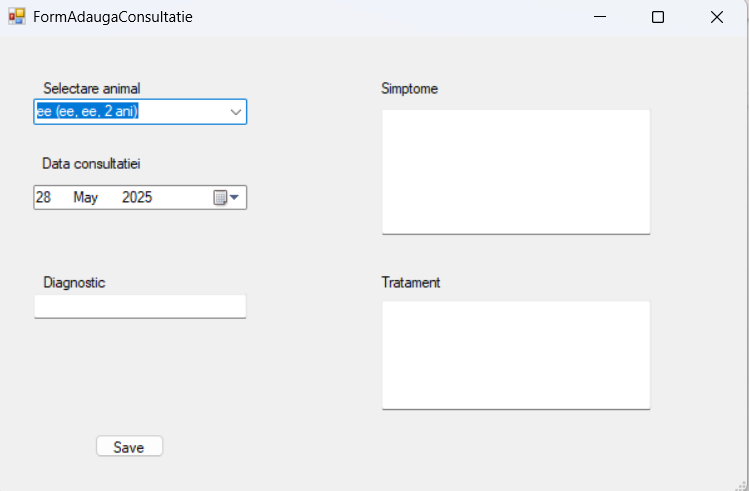
Butonul „Administrare utilizatori” este disponibil doar pentru utilizatorii cu rol de administrator. Apăsarea lui deschide o interfață unde se pot: vizualiza toți utilizatorii înregistrați, adăuga un utilizator nou, șterge un utilizator existent, atribui roluri și parole.



Alte opțiuni:

* Ajutor – deschide fereastra de Help (VetLifeHelp.chm), unde utilizatorul poate găsi informații detaliate despre fiecare modul al aplicației
* Ieșire – închide aplicația

Adăugare consultație



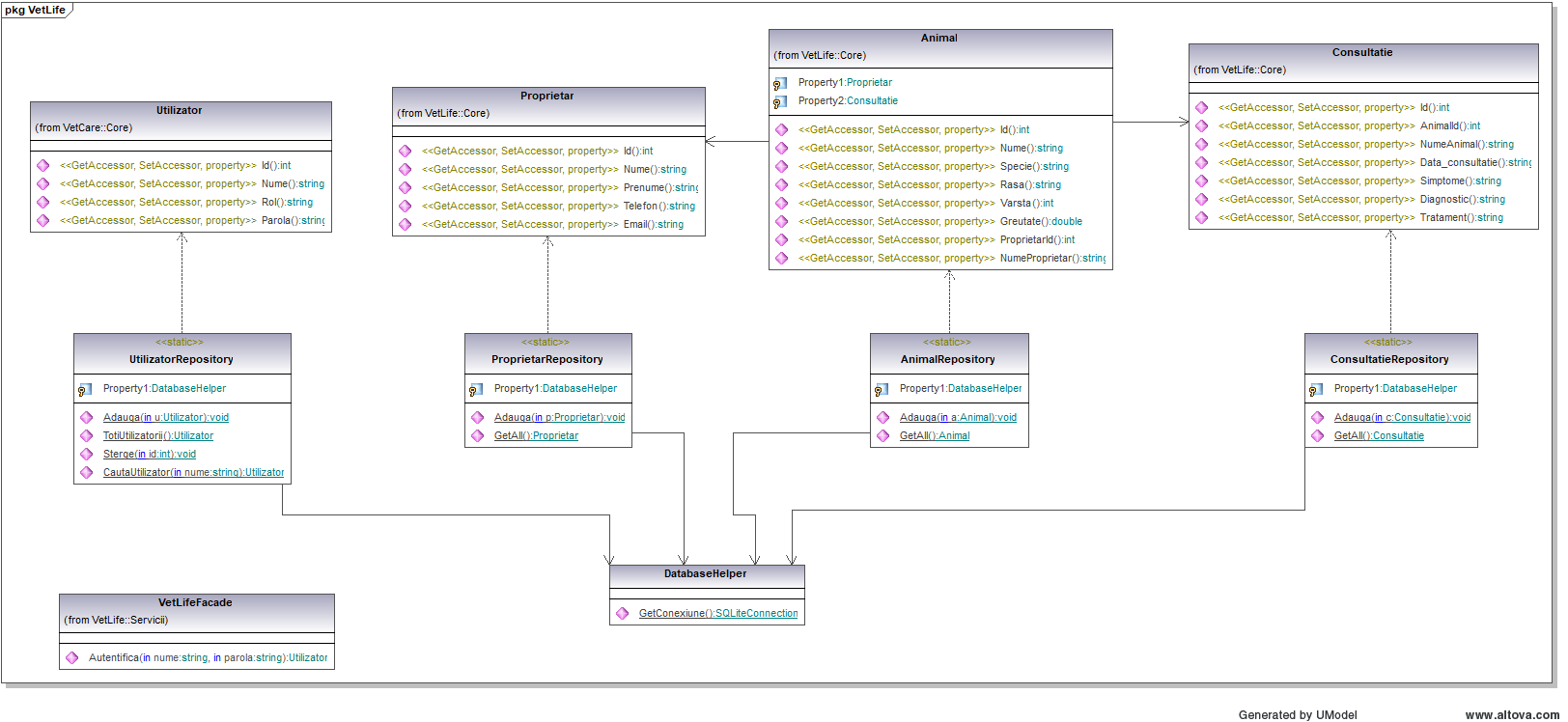
Această interfață permite înregistrarea unei consultații medicale pentru un animal deja existent. Utilizatorul selectează animalul dintr-o listă, introduce data consultației, simptomele observate, diagnosticul stabilit și tratamentul recomandat. Toate câmpurile sunt obligatorii. La apăsarea butonului „Save”, aplicația validează completitudinea și salvează consultația în baza de date, asociind-o corect cu animalul selectat. Un mesaj confirmă salvarea cu succes.

4. Diagrame

**4.1 Diagrama de clase**

### 4.1.1 Prezentare Generală

Diagrama de clase prezentată face parte dintr-o aplicație pentru gestionarea activităților unei clinici veterinare, numită "VetLife". Aceasta ilustrează principalele entități ale sistemului, cum ar fi Utilizatori, Proprietari de animale, Animale, Consultații, precum și Repozitoarele pentru accesarea datelor. Există și un Facade pentru simplificarea autentificării.



### 4.1.2 Descrierea Pachetelor și Claselor

#### 1. Clasa Utilizator

* Atribute (proprietăți):
  + Id: Identificatorul unic al utilizatorului.
  + Nume: Numele utilizatorului.
  + Prenume: Prenumele utilizatorului.
  + Rol: Rolul utilizatorului în aplicație (ex. medicVeterinat, asistent).
  + Parola: Parola utilizatorului.
* Funcționalitate: Reprezintă datele unui utilizator al aplicației.

#### 2. Clasa Proprietar

* Atribute (proprietăți):
  + Id: Identificatorul unic al proprietarului.
  + Nume, Prenume: Numele și prenumele proprietarului.
  + Telefon, Email: Date de contact.
* Funcționalitate: Reprezintă stăpânii animalelor.

#### 3. Clasa Animal

* Atribute (proprietăți):
  + Id: Identificatorul animalului.
  + Nume, Specie, Rasa, Varsta, Greutate: Detalii despre animal.
  + ProprietarId, NumeProprietar: Legătura cu proprietarul.
* Proprietăți de referință:
  + Proprietar: Legătura cu clasa Proprietar.
  + Consultatie: Consultațiile aferente animalului.
* Funcționalitate: Reprezintă datele de bază despre animale.

#### 4. Clasa Consultatie

* Atribute (proprietăți):
  + Id, AnimalId, NumeAnimal: Informații despre animalul consultat.
  + Data\_consultatiei, Simptome, Diagnostic, Tratament: Detalii despre consultație.
* Funcționalitate: Reprezintă o consultație veterinară.

#### 5. Clasa DatabaseHelper

* Funcționalitate:
  + Se ocupă cu conexiunea la baza de date (GetConexiune → SQLiteConnection).
  + Este utilizată de toate Repozitoarele.

### 4.1.3. Relații între Clase

* Proprietar ↔ Animal: Relație de asociatie (un proprietar are unul sau mai multe animale).
* Animal ↔ Consultatie: Relație de asociatie (un animal poate avea mai multe consultații).
* Repozitoare (Repository) au dependențe către DatabaseHelper și sunt responsabile cu gestionarea datelor claselor pe care le servesc.

### 4.1.4. Repozitoare și Rolul Lor

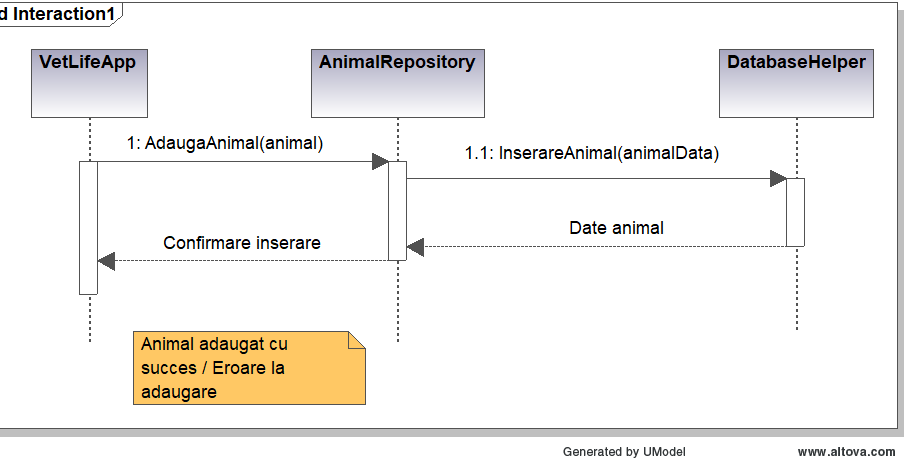
* UtilizatorRepository:
  + Operații: Adauga, TotiUtilizatorii, Sterge, CautaUtilizator.
* ProprietarRepository:
  + Operații: Adauga, GetAll.
* AnimalRepository:
  + Operații: Adauga, GetAll.
* ConsultatieRepository:
  + Operații: Adauga, GetAll.
* Funcționalitate comună: Aceste clase facilitează manipularea și stocarea datelor în baza de date.

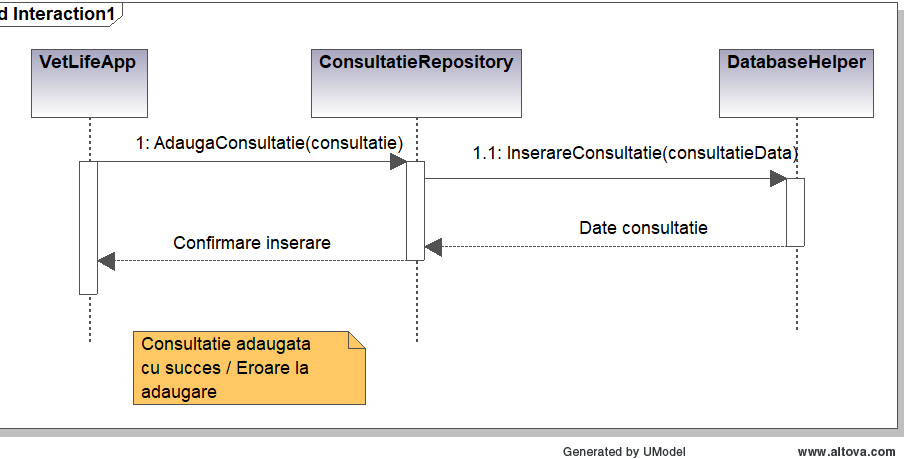
### 4.1.5. VetLifeFacade

* Operații:
  + Autentifica: Permite autentificarea utilizatorului pe baza numelui și parolei.
* Rol: Oferă un punct de acces simplificat pentru operația de autentificare.

**4.2 Diagramele de secvente**

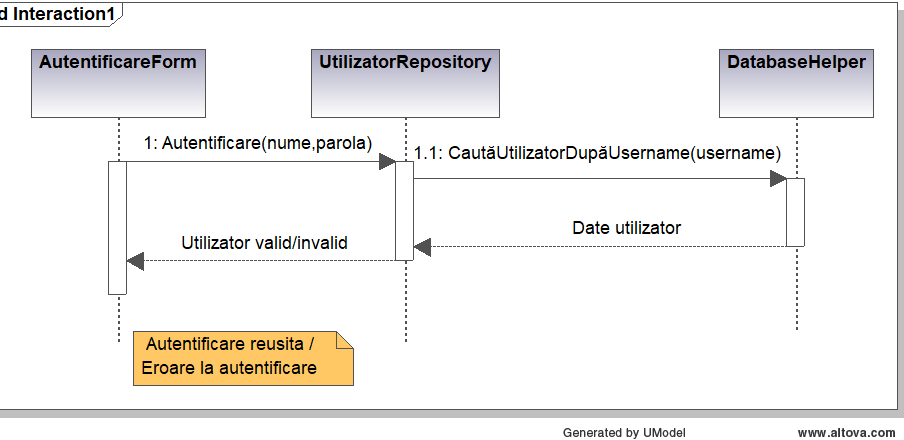
**4.2.1. Adaugarea unui animal**

 Atunci când un utilizator vrea să adauge un animal în aplicația VetLife, introduce datele și apasă butonul de salvare. Apoi, aplicația trimite datele la un modul (AnimalRepository) care le pregătește și le validează, iar apoi le transmite către baza de date printr-un alt modul (DatabaseHelper). Acesta din urmă încearcă să le salveze și trimite înapoi un rezultat – dacă a mers totul bine sau dacă a fost o eroare. În funcție de acest rezultat, aplicația îi arată utilizatorului un mesaj care confirmă succesul sau semnalează eroarea.

**4.2.2 Adaugarea unei consultatii**

Când un utilizator introduce o consultație și apasă butonul de „Adaugă”, aplicația trimite aceste date către un modul specializat (ConsultatieRepository) care le pregătește și le validează, iar apoi le transmite mai departe către un alt modul (DatabaseHelper) care se ocupă de salvarea efectivă în baza de date. Acesta încearcă să le adauge și trimite înapoi un rezultat care indică succesul sau o eventuală eroare. În funcție de acest rezultat, aplicația afișează un mesaj de confirmare pentru utilizator.

**4.2.3 Adaugarea unui utilizator**



Când un utilizator își introduce datele de autentificare (nume de utilizator și parolă) și apasă butonul de logare, aplicația (VetLifeApp) trimite aceste date către un modul specializat (UtilizatorRepository) care le verifică și le validează. Dacă datele sunt corecte și utilizatorul nu există deja, modulul trimite mai departe datele către un alt modul (DatabaseHelper) care le salvează în baza de date. După ce datele sunt salvate, aplicația primește un răspuns care confirmă dacă procesul a fost reușit sau nu și afișează un mesaj corespunzător pentru utilizator.

**4.3 Diagrama cazurilor de utilizare**

A diagram of a company

AI-generated content may be incorrect.

Inițial, actorul este un utilizator neautentificat care trebuie să treacă de pasul de autentificare, după care actorul poate avea 4 roluri: administrator, medic veterinar, asistent și recepționer. Administratorul are acces la toate activitățile programului nostru: gestionare utilizatori, gestionare animale, gestionare consultații și gestionare proprietari. Medicul veterinar și asistentul au aceleași posibilități, exceptând gestionarea utilizatorilor, iar recepționerul poate doar să gestioneze consultații.

**4.4 Diagrama de activitate**

A diagram of a flowchart

AI-generated content may be incorrect.

Diagrama de activitate e compusă din 4 swimlane-uri, cate unul pentru fiecare rol pe care îl poate avea un utilizator. Nodul inițial e mereu punctul ulterior autentificării în care utilizatorul își preia rolul și ajunge în meniul principal, care e marcat cu un bloc decizional. În funcție de posibilitățile oferite de fiecare rol, utilizatorul poate desfășura o activitate, după care alege dacă părăsește aplicația sau dacă se întoarce în meniul principal. Pentru interfața FormAdaugaConsultatie, ieșirea se poate realiza prin butonul Save.

ANEXĂ

Coduri esențiale:

* Model de bază – Proprietar

public class Proprietar

{

//Identificator unic

public int Id { get; set; }

//Numele proprietarului

public string Nume { get; set; }

//Prenumele proprietarului

public string Prenume { get; set; }

//Numărul de telefon al proprietarului

public string Telefon { get; set; }

//Emailul proprietarului

public string Email { get; set; }

}

Reprezintă un exemplu standard de entitate. S-a efectuat similar și în cazul claselor Animal, Consultație, Utilizator, respectându-se proprietățile entităților de fiecare fel.

* Metoda Adaugă din ProprietarRepository

public static void Adauga(Proprietar proprietar)

{

try

{

//Obține conexiunea la baza de date SQLite

using (var conexiune = DatabaseHelper.GetConexiune())

{

conexiune.Open();

//Creează comanda SQL cu parametri pentru inserare

var comanda = new SQLiteCommand("INSERT INTO Proprietari (Nume,Prenume,Telefon,Email)" +

"VALUES(@nume, @prenume, @tel, @email)", conexiune);

//Parametrii protejează împotriva SQL injection

comanda.Parameters.AddWithValue("@nume", proprietar.Nume);

comanda.Parameters.AddWithValue("@prenume", proprietar.Prenume);

comanda.Parameters.AddWithValue("@tel", proprietar.Telefon);

comanda.Parameters.AddWithValue("@email", proprietar.Email);

//Execută comanda de inserare

comanda.ExecuteNonQuery();

}

}

catch (Exception exceptie)

{

//Prinde orice eroare legată de salvarea în baza de date

throw new Exception("Eroare la adăugarea proprietarului:" + exceptie.Message);

}

}

Arată cum se salvează datele în baza de date SQLite. S-a efectuat similar și în cazul claselor AnimalRepository, ConsultațieRepository, UtilizatorRepository.

* Logica UI – butonul de adăugare Proprietar

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

{

var p = new Proprietar

{

Nume = txtNumeProprietar.Text.Trim(),

Prenume = txtPrenumeProprietar.Text.Trim(),

Telefon = txtTelefonProprietar.Text.Trim(),

Email = txtEmailProprietar.Text.Trim()

};

try

{

if (string.IsNullOrWhiteSpace(p.Nume) || p.Nume.Length < 2)

{

MessageBox.Show("Introduceti un nume valid (minim 2 caractere)");

return;

}

if (string.IsNullOrWhiteSpace(p.Prenume) || p.Prenume.Length < 2)

{

MessageBox.Show("Introduceti un prenume valid (minim 2 caractere)");

return;

}

if (string.IsNullOrWhiteSpace(p.Telefon) || !p.Telefon.All(char.IsDigit) || p.Telefon.Length < 7)

{

MessageBox.Show("Introduceti un nr de telefon valid (minim 7 cifre)!");

return;

}

try

{

var emailCheck = new System.Net.Mail.MailAddress(p.Email);

}

catch

{

MessageBox.Show("Introduceti un email valid!");

return;

}

var toti = ProprietarRepository.GetAll();

bool exista = toti.Any(ex =>

(ex.Nume.Equals(p.Nume, StringComparison.OrdinalIgnoreCase) &&

ex.Prenume.Equals(p.Prenume, StringComparison.OrdinalIgnoreCase)) ||

ex.Telefon == p.Telefon || ex.Email.Equals(p.Email, StringComparison.OrdinalIgnoreCase));

if (exista)

{

MessageBox.Show("Un proprietar cu acest nume, prenume, telefon sau email existent");

return;

}

ProprietarRepository.Adauga(p);

MessageBox.Show("Proprietar adăugat cu succes!");

txtNumeProprietar.Clear();

txtPrenumeProprietar.Clear();

txtTelefonProprietar.Clear();

txtEmailProprietar.Clear();

}

catch (Exception ex)

{

MessageBox.Show("Eroare la adaugare proprietar:" + ex.Message);

}

}

S-a efectuat similar și în cazul butoanelor Adauga animal, Adauga consultație și Adauga din UtilizatoriForm.cs.

* Permisiune pe roluri

private void AplicarePermisiuni()

{

btnAnimal.Enabled = false;

btnConsultatie.Enabled = false;

btnConsultatiiV.Enabled = false;

btnPersonal.Enabled = false;

bool poateAdaugaAnimal = \_rolCurent == "Administrator" || \_rolCurent == "Receptioner";

switch (\_rolCurent)

{

case "Administrator":

txtNumeAnimal.Enabled = poateAdaugaAnimal;

txtSpecie.Enabled = poateAdaugaAnimal;

txtRasa.Enabled = poateAdaugaAnimal;

txtVarsta.Enabled = poateAdaugaAnimal;

txtGreutate.Enabled = poateAdaugaAnimal;

btnAnimal.Enabled = true;

btnConsultatie.Enabled = true;

btnConsultatiiV.Enabled=true;

btnPersonal.Enabled=true;

break;

case "MedicVeterinar":

btnProprietar.Enabled = false;

btnAnimal.Enabled=false;

btnConsultatie.Enabled=false;

btnPersonal.Enabled=false;

btnConsultatiiV.Enabled = true;

break;

case "Asistent":

btnConsultatiiV.Enabled=false;

btnPersonal.Enabled=false;

btnConsultatie.Enabled = true;

btnAnimal.Enabled = true;

break;

case "Receptioner":

txtNumeAnimal.Enabled = poateAdaugaAnimal;

txtSpecie.Enabled = poateAdaugaAnimal;

txtRasa.Enabled = poateAdaugaAnimal;

txtVarsta.Enabled = poateAdaugaAnimal;

txtGreutate.Enabled = poateAdaugaAnimal;

btnConsultatiiV.Enabled = true;

btnPersonal.Enabled=false;

btnConsultatie.Enabled = true;

btnAnimal.Enabled = false;

btnProprietar.Enabled=false;

break;

}

}

Arată diferențierea de acces pe roluri.

* Test de verificare al parametrilor de tip Proprietar

[TestMethod]

public void VerificareInitializareProprietar()

{

//Se creează un obiect Proprietar și se setează manual proprietățile

var u = new Proprietar

{

Id = 1,

Nume = "Popescu",

Prenume = "Ana",

Telefon = "0765348271",

Email = "popescuana@gmail.com"

};

//Se verifică dacă fiecare proprietate a fost setată corect

Assert.AreEqual(1, u.Id);//Test pentru ID

Assert.AreEqual("Popescu", u.Nume);//Test pentru Nume

Assert.AreEqual("Ana", u.Prenume);//Test pentru Prenume

Assert.AreEqual("0765348271", u.Telefon);//Test pentru Telefon

Assert.AreEqual("popescuana@gmail.com", u.Email);//Test pentru Email

}

Demonstrează că aplicația este stabilă. S-a efectuat similar și în cazul claselor Animal, Consultație și Utilizator.