Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационных технологий

Кафедра программной инженерии

Специальность 1-40 01 01 «Программное обеспечение информационных технологий»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**КУРСОВОГО ПРОЕКТА**

по дисциплине «Объектно-ориентированное программирование» Тема «Программное средство – Учет питомцев»

|  |  |  |  |
| --- | --- | --- | --- |
| Исполнитель  студентка 2 курса группы 10 Косовец Полина Олеговна  (Ф.И.О.)  Руководитель работы ассистент Сапотько Р.А.  (учен. степень, звание, должность, подпись, Ф.И.О.)  Курсовой проект защищен с оценкой  Председатель Пацей Н.В.  (подпись) |  |  |  |

Минск 2019

**СОДЕРЖАНИЕ**

[Введение](#page3) [6](#page3)

1. [Постановка задачи](#page4) [7](#page4)

[2. Обзор](#page8) аналогов 7

[2.1 DogLogBook](#page8) 7

2.2 PawTracks ..8

2.3 CatBuddy и DogBuddy ..9

[3. Проектирование](#page10) программного средства ..9

[3.1 Проектирование базы данных](#page10) ..9

[3.2 Структура](#page11) 11

[3.3](#page11) Диаграмма UML, взаимосвязь всех компонентов 12

[4. Реализация](#page10) программного средства 14

[4.1 Вспомогательные](#page10) классы 14

[4.2 Регистрация](#page11) и авторизация 17

[4.3](#page11) Создание питомца 20

[4.4](#page11) Статистика здоровья питомца 20

[5. Руководство пользователя](#page15) [21](#page15)

[Заключение](#page23) 25

[Список используемых источников](#page24) 26

[Приложение А](#page25) 27

[Приложение Б](#page28) 30

[Приложение](#page28) С 36

**ВВЕДЕНИЕ**

Практически в каждом доме есть питомцы, будь это кот или собака. Уход за ними крайне важен, в особенности самые капризные породы. Делать пометки в бумажном варианте не всегда удобно, так как мы можем потерять и случайно выбросить. Крайне удобно пользоваться электронным учетом здоровья и состояния своего питомца.

Электронные заметки применяются в качестве различных приложений: учет веса, питания, планов, питомцев. Что касается последнего, это такие приложения как DogLogBook, Paw Tracks, iKibble, Cat/Dog Buddy. Такие приложения позволяют следить за состоянием питомца, ставить напоминания о вакцинации, делать заметки каких-либо событий, произошедших с питомцем в определенную дату, получать информацию о правильном уходе при определенных проблемах их здоровья.

Целью данного курсового проекта является создание программного средства «Учет питомцев», которое будет предоставлять пользователю следующий функционал:

– Создание своего питомца;

– Оставлять заметки о состоянии питомца в определенную дату;

– Следить за весом и здоровьем питомца;

– Получать рекомендацию по уходу за питомцем в случае проблем с шерстью, зубами и другими;

– Загрузка фотографий.

# **Постановка задачи**

Основной задачей курсового проекта является разработка десктопного приложения, позволяющего пользователю хранить информацию в централизованной базе данных о своих питомцах, а также следить за изменениями состояния здоровья каждого из них.

Функционально ПС должно выполнять следующие задачи:

– создавать питомцев (кот или собака);

– выполнять регистрацию и авторизацию;

– сохранять информацию о питомцах в централизованной базе данных;

– загружать фотографии питомца;

– оставлять заметки о состоянии питомца в определенную дату;

– предоставлять статистику о изменении веса питомца;

– смотреть информацию о каждом из питомцев.

В разработке приложения были использованы нижеперечисленные технологии:

– C# — объектно-ориентированный язык программирования;

– Entity Framework;

* Microsoft SQL Server;
* WPF.

Entity Framework предоставляет специальную объектно-ориентированную технологию на базе .NET для работы с данными. Более высокий уровень абстракции позволяет абстрагироваться от самой базы данных и работать с данными независимо от типа хранилища. Если на физическом уровне мы оперируем таблица, индексами, первичными и вторичными ключами, то на концептуальном уровне, который нам предлагает данная технология, мы уже работаем с объектами. Центральной концепцией Entity Framework является понятие сущности или entity. Сущность представляет набор данных, ассоциированных с определенным объектом. Поэтому данная технология предполагает работу не с таблицами, а с объектами и их наборами. Способы взаимодействия с БД: Database first, Model first, Code first.

Microsoft SQL Server — система управления реляционными БД. Основной используемый язык запросов — Transact-SQL — реализован на структурированном языке запросов (SQL) с расширениями.

Windows Presentation Foundation (WPF) — система для построения клиентских приложений Windows с визуально привлекательными возможностями взаимодействия с пользователем, графическая (презентационная) подсистема в составе .NET Framework (начиная с версии 3.0), использующая язык XAML.

В основе WPF лежит векторная система визуализации, не зависящая от разрешения устройства вывода и созданная с учётом возможностей современного графического оборудования. WPF предоставляет средства для создания визуального интерфейса, включая язык XAML (eXtensible Application Markup Language), элементы управления, привязку данных, макеты, двухмерную и трёхмерную графику, анимацию, стили, шаблоны, документы, текст, мультимедиа и оформление.

Графической технологией, лежащей в основе WPF, является DirectX, в отличие от Windows Forms, где используется GDI/GDI+. Производительность WPF выше, чем у GDI+ за счёт использования аппаратного ускорения графики через DirectX.

Также существует урезанная версия CLR, называющаяся WPF/E, она же известна как Silverlight.

## **Обзор аналогов**

## **2.1 DogLogBook**

Разработанное сотрудниками ветеринарного факультета Университета Сиднея приложение можно назвать Swarm для собак. Предлагает создание своих питомцев. В нем можно вести учет всего, что происходит с вашей собакой: прогулок, тренировок, приема лекарств и тому подобных вещей (рисунок 1.1). Все это может помочь ветеринару поставить грамотный диагноз.

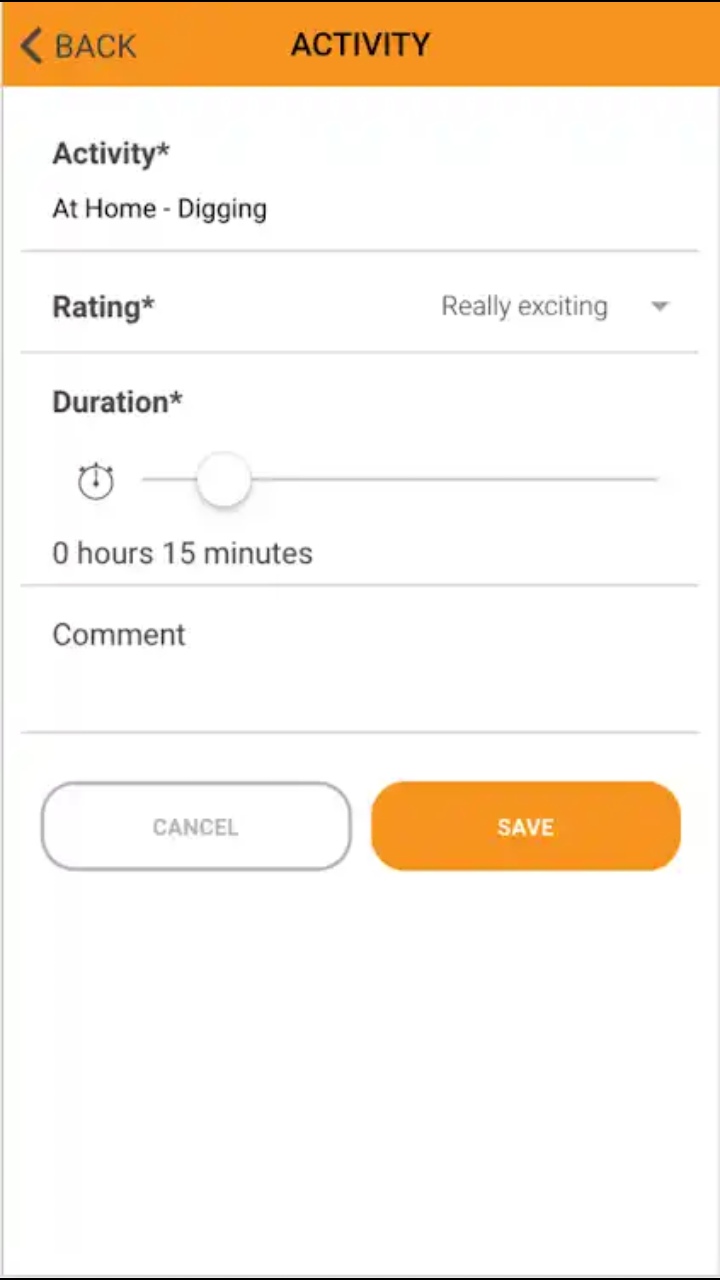
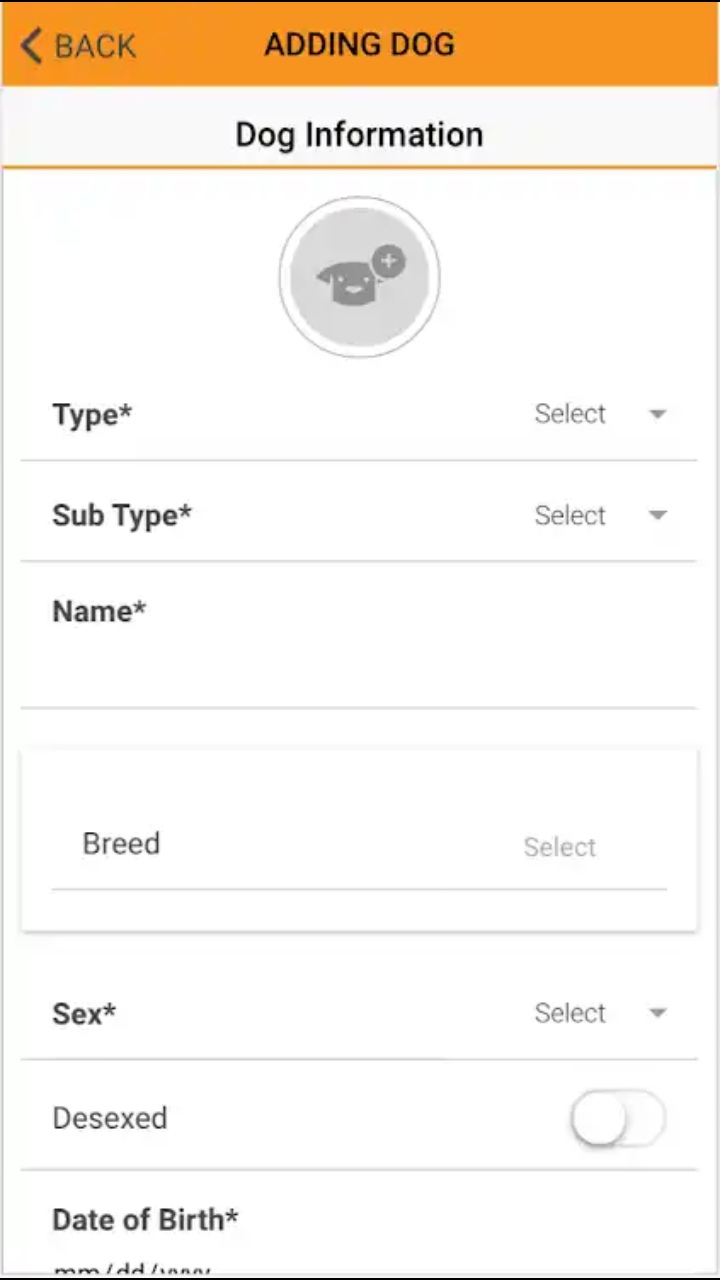
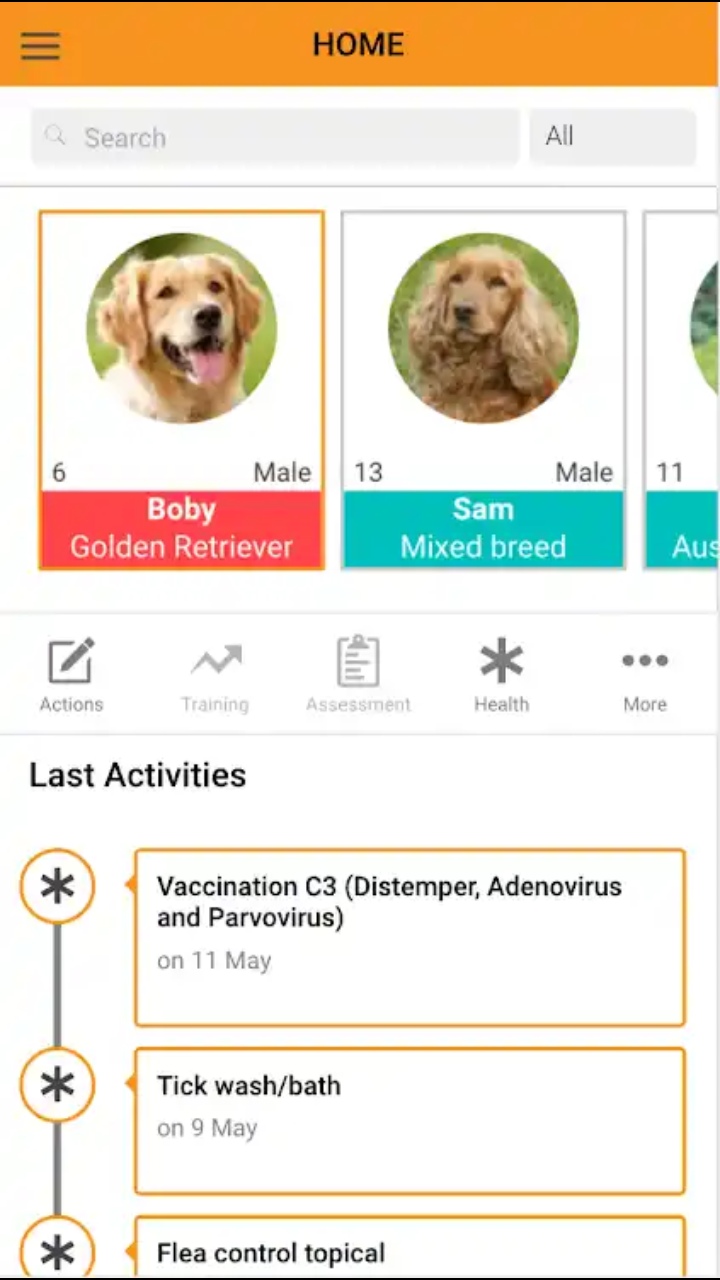


Рисунок 1.1 – Окна приложения «DogLogBook»

## **2.2 Paw Tracks**

Аналог Doglogbook. Он больше подойдет тем, кто ухаживает за животным вместе с друзьями, соседями или семьей. В приложение можно записывать информацию о том, когда и сколько собака ела и гуляла, сколько времени прошло с тех пор, кто этим занимался и другие данные (рисунок 1.2).

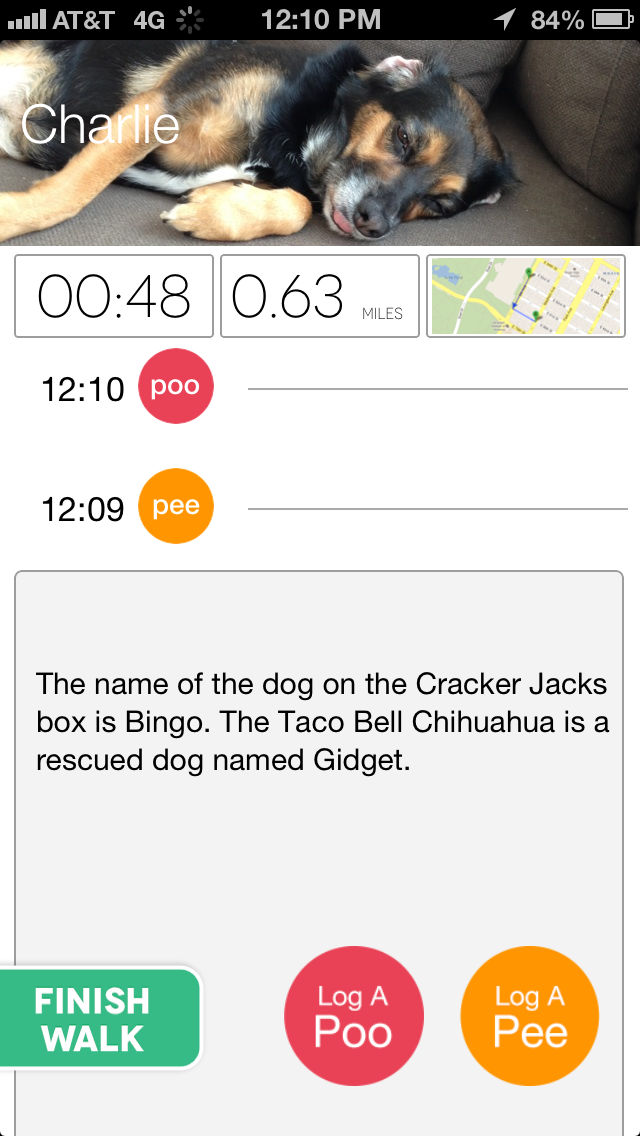


Рисунок 1.2 – Главная экран приложения «Paw Tracks»

**2.3 Cat Buddy и Dog Buddy**

Еще одно приложение для мониторинга здоровья животных. Напоминает про прием лекарств, следит за весом, дает советы по оказанию первой помощи кошке или собаке.

1. **Проектирование программного средства**

# **Проектирование базы данных**

Для реализации поставленной задачи была создана база данных MyPets. Для её создания использовалась система управления реляционными базами данных Microsoft SQL Server 2012.

База данных состоит из 6 таблиц, перечисленных ниже. Скрипты для работы с базой данных представлены в ПРИЛОЖЕНИИ Б.

Диаграмма на рисунке 3.1 иллюстрирует структуру таблицы USERS, которая содержит информацию о пользователе:

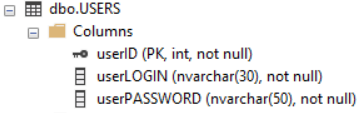


Рисунок 3.1.1 – Структура таблицы «USERS»

Для хранения питомцев создана таблица PETS представленная на рисунке 3.1.2:

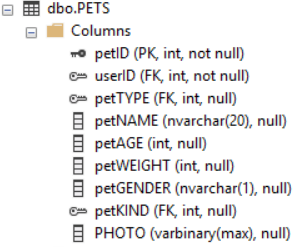


Рисунок 3.1.2 – Структура таблицы «PETS»

Сведения о типах питомцев можно посмотреть в таблице PETTYPES (рисунок 3.3):



Рисунок 3.1.3 – Структура таблицы «PETTYPES»

Также была создана и наполнена таблица KINDS с породами котов и собак:

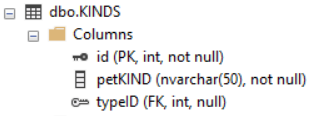


Рисунок 3.1.4 – Структура таблицы KINDS

В таблице PETPROGRESS храниться информация о состоянии питомца в соответствующую дату (рисунок 3.5).



Рисунок 3.1.5 – Структура таблицы PETPROGRESS

Таблица CARE наполнена информацией о болезнях, вакцинации и уходе за питомцем (рисунок 3.6).

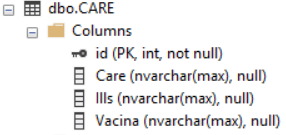


Рисунок 3.1.6 – Структура таблицы CARE

Взаимосвязь таблиц базы данных представлена на рисунке 3.7.

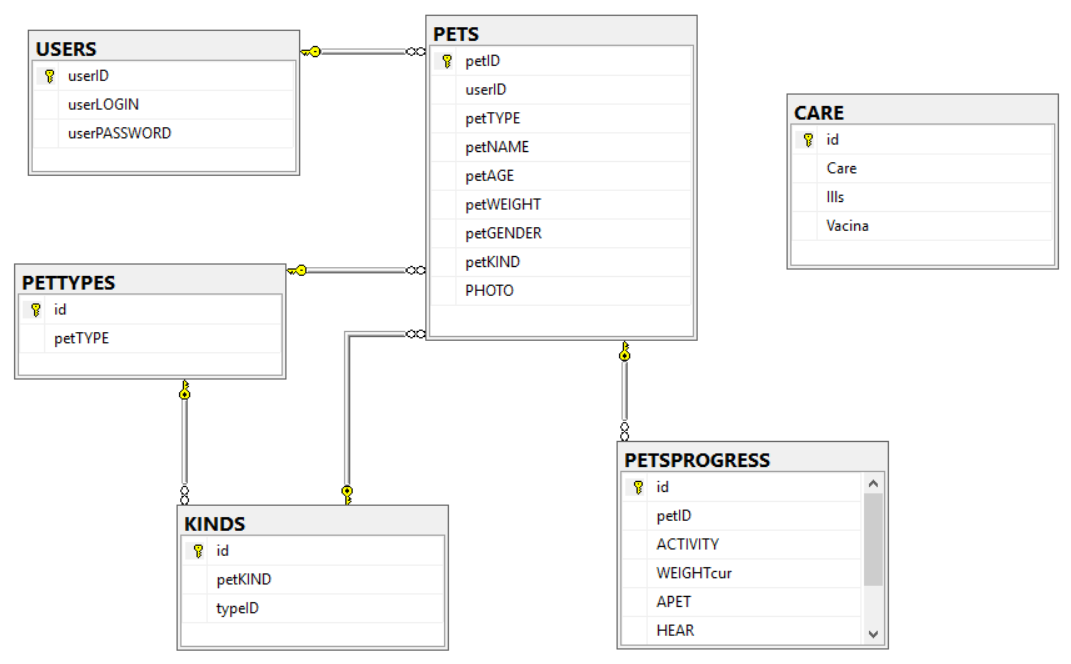


Рисунок 3.1.7 – Взаимосвязь таблиц базы данных

**3.2 Структура**

Структура проекта представлена на рисунке 3.2.1.

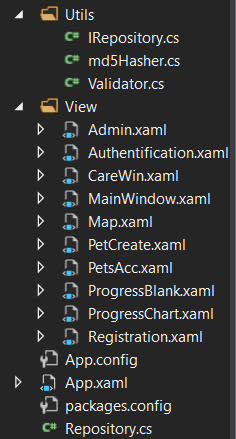
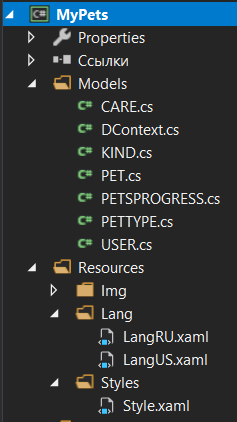


Рисунок 3.2.1 – Структура проекта

– Properties **–** свойства проекта, содержит информацию о сборке, используемых ресурсах и настройках;

– Ссылки **–** перечень сборок, используемых в проекте;

– Resources – папка с ресурсами проекта (локализация, изображения и стили);

– App.config **–** файл с параметрами проекта;

– App.xaml – класс Application;

– Utils – папка с вспомогательными классами (валидатор, хэшер и интерфейс IRepository);

– View – папка с пользовательскими элементами управления и окнами приложения;

– Models – папка с классами и контекстной моделью.

**3.3 Диаграммы UML, взаимосвязь всех компонентов**

Диаграмма в UML — это графическое представление набора элементов, изображаемое чаще всего в виде связанного графа с вершинами (сущностями) и ребрами (отношениями). Диаграммы рисуют для визуализации. Основная цель диаграмм — визуализация разрабатываемой системы с разных точек зрения.

Приложение спроектировано таким образом, что каждый класс выполняет свои функции и практически не зависит от других. Диаграмма классов представлена в ПРИЛОЖЕНИИ С.

Диаграмма использования представлена на рисунке 3.3.1. Это наиболее общее представление функционального назначения системы. Каждый вариант использования определяет последовательность действий, которые должны быть выполнены проектируемой системой при взаимодействии ее с соответствующим актером. В данном курсовом проекте пользователь может выполнять функции, показанные на диаграмме.



Рисунок 3.3.1 – Диаграмма использования

На рисунке 3.3.2 представлена блок-схема порядка использования.



Рисунок 3.3.2 – Блок-схема порядка использования

На рисунке 3.3.3 представлена диаграмма последовательности. Она представляет собой порядок действий, который пользователь должен совершить для того, чтобы создать нового питомца.

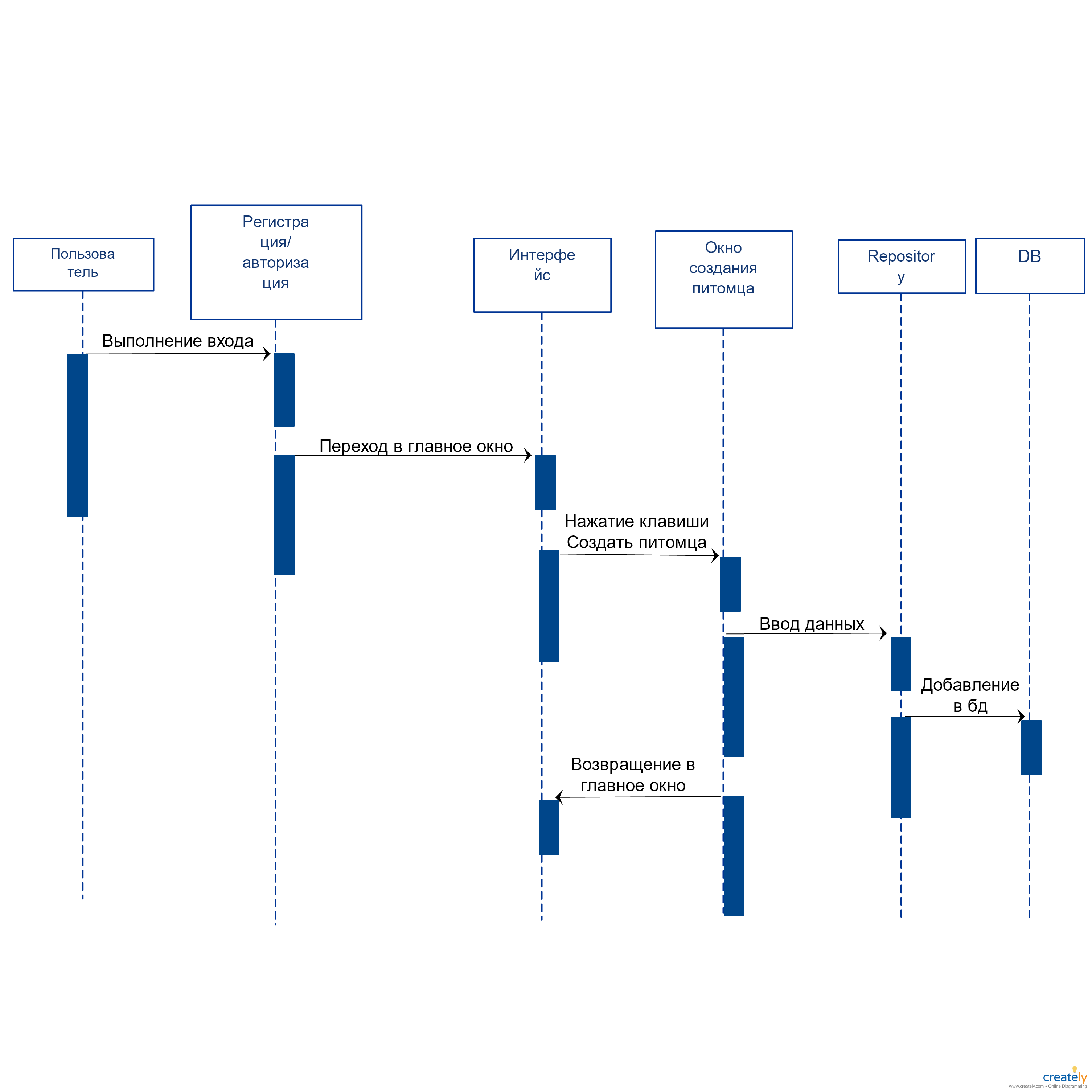


Рисунок 3.3.3 – Диаграмма последовательности

**4 Реализация программного средства**

**4.1 Вспомогательные классы**

Данное приложение предоставляет пользователю ряд функций, таких как:

* регистрация с последующей авторизацией пользователя;
* добавление питомцев;
* возможность оставлять заметки о состоянии питомца;
* отображение статистики;
* информация об уходе;
* локализацию языка.

Паттерн Repository позволяет абстрагироваться от конкретных подключений к источникам данных, с которыми работает программа, и является промежуточным звеном между классами, непосредственно взаимодействующими с данными, и остальной программой. Сперва мы должны определить интерфейс репозитория (рисунок 4.1). На случай, если контекст данных будет подразумевать освобождение или закрытие подключений, интерфейс репозитория применяет интерфейс IDisposable. Далее была создана непосредственная реализация для работы с MS SQL Server в самом классе Repository, представленная в ПРИЛОЖЕНИИ А.

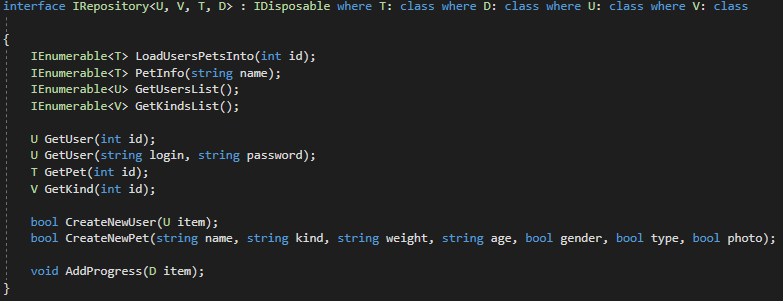


Рисунок 4.1.1 – Интерфейс IRepository

Класс Validator содержит статические методы, осуществляющие проверку логина, пароля, а также имени, возраста и веса питомца при его создании или заметке состояния здоровья.

Валидный логин представляет собой запись @mail адреса пользователя, следовательно, в методе ValidLogin осуществляется проверка строки на наличие «@[адрес]». Результатом функции является строка, содержащая логин, введенный пользователем. В случае не выполнения условия валидности результатом функции является строка «Укажите логин в форме [х@х.х»](mailto:х@х.х). Реализация данного метода представлена на рисунке 4.1.2.

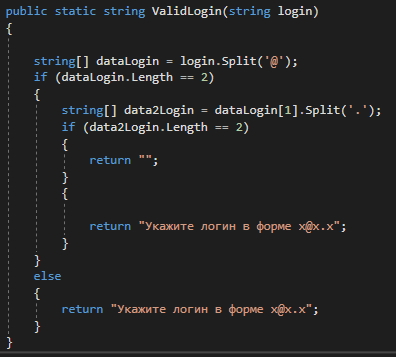


Рисунок 4.1.2 – Функция проверки валидности логина

Следующая функция проверяет валидность пароля и называется ValidPass (рисунок 4.3). В данном случае, пароль должен содержать цифры, строчные и прописные буквы, а также исключать пробелы. Были использованы регулярные выражения. Результат функции – булевое значение true или false.

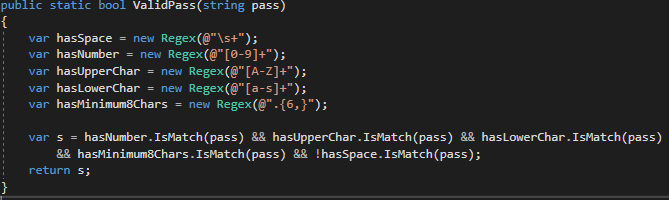


Рисунок 4.1.3 – Функция проверки валидности пароля

Так как имена питомцев могут содержать цифры, строчные и прописные буквы, следует убедиться, что данные условия выполняются. На рисунке 4.5 представлена функция ValidName, позволяющая проверить выполнение ранее перечисленных условий.

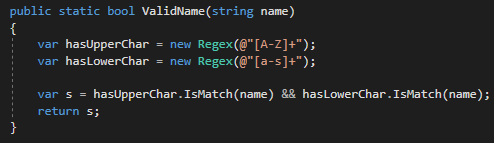


Рисунок 4.1.5 – Функция проверки валидности имени

Две функции ValidAgeWeight типа bool с разными параметрами проверяют поля возраста и веса питомца (рисунок 4.6). В первом случае функция принимает вес и возраст и проверяет, чтобы значения не были отрицательными. Во втором случае функция проверяет строку на наличие запятой в весе для большей точности.

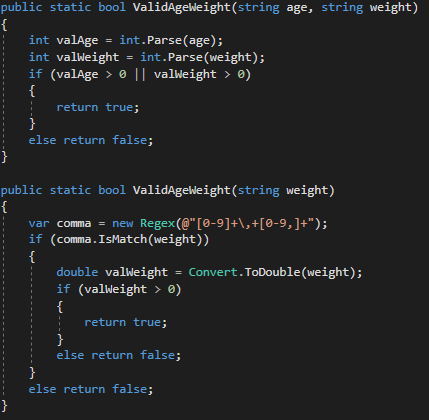


Рисунок 4.1.6 – Функции проверки валидности веса и возраста

Следующий вспомогательный класс имеет название md5Hasher. Алгоритм md5 – это алгоритм хэширования.

Функция GetMd5Hash получает строку и шифрует полученные данные в форме 128-bit hash (рисунок 4.7).

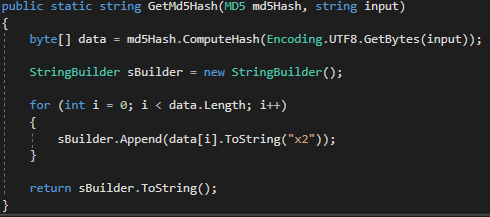


Рисунок 4.1.7 – Функция хеширования строки

### 4.2 Регистрация и авторизация

Регистрация пользователя происходит в окне Registration. Обработчик события клика на кнопку Зарегистрироваться (рисунок 4.2.1) осуществляет проверку валидации, обращаясь к статическим методам класса Validator, и, в случае прохождения проверки, хэширует пароль, и полученные данные нового пользователя добавляет в базу данных, обращаясь к функции CreateNewUser класса Repository (рисунок 4.2.2).

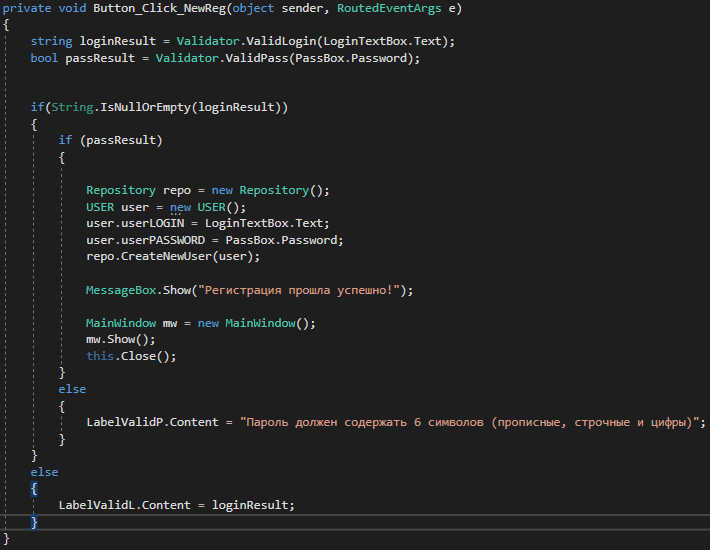


Рисунок 4.2.1 – Функция обработчик клика на кнопку Зарегистрироваться

Если логин не является валидным, переменная loginResult принимает строку с условиями, которые должны выполняться, и передает в скрытый label красного цвета, которые пользователь увидит под TextBox-ом. То же самое происходит при не выполнении условия валидности пароля.

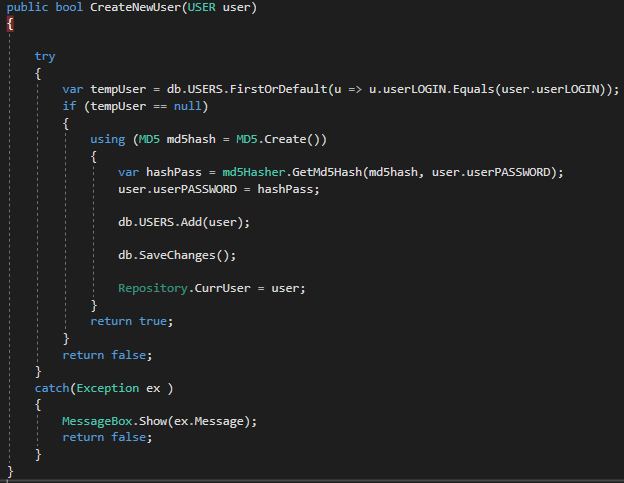


Рисунок 4.2.2 – Функция создания нового пользователя

Авторизация пользователя происходит в окне Autentification. Кнопка войти имеет обработчик клика, представленный на рисунке 4.2.3. В случае, если данный пользователь не найден в базе данных, выводится сообщение с просьбой зарегистрироваться.

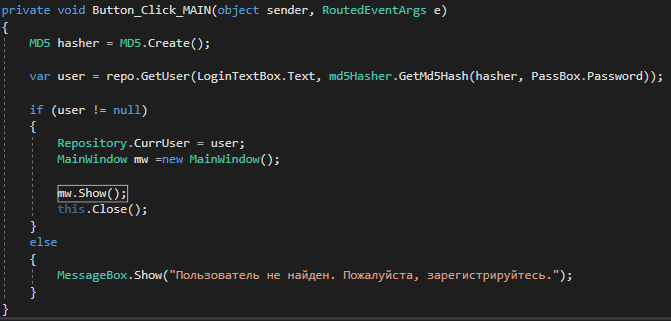


Рисунок 4.2.3 – Функция обработчик клика кнопки входа

Функция содержит проверку на соответствие введенных данных с данными в бд. Поиск пользователя в базе данных осуществляется функцией GetUser, принимающей строки логина и пароля, класса Repository (рисунок 4.2.4).

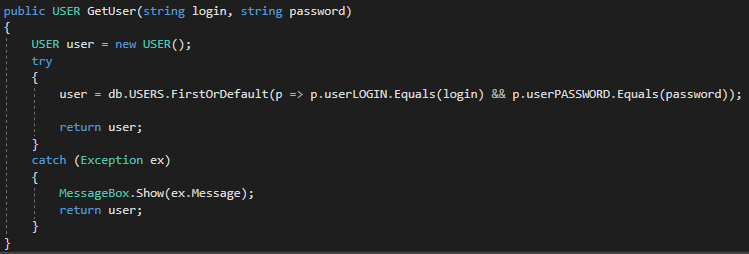


Рисунок 4.2.4 – Функция поиска пользователя в БД

При создании питомца на почту приходит сообщение о том, что питомец создан и добавлен в базу данных. Функция MailService представлена на рисунке 4.2.5.



Рисунок 4.2.5 – Функция отправки email

### 4.3 Создание питомца.

Создание питомца происходит при открытии соответствующего окна с последующим вводом необходимой информации. Каждое поле проверяется на валидность введенных данных.

На рисунке 4.3.1 представлена структура класса окна PetCreate:

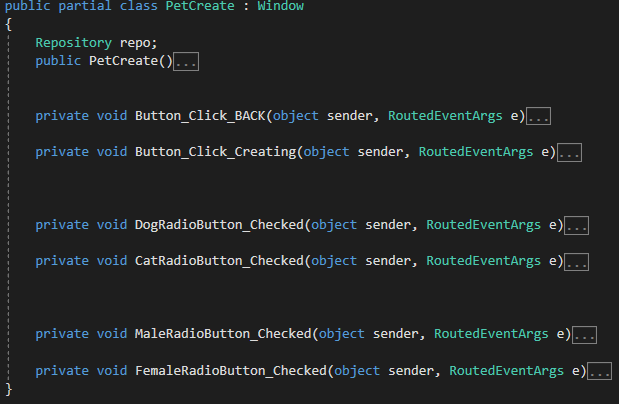


Рисунок 4.3.1 – Структура класса PetCreate

**4.4 Статистика здоровья питомца**

Возможность оставить заметки о здоровье питомца предоставляется классом ProgressBlank, структура которого представлена на рисунке 4.4.1, а просмотр статистики в классе ProgressChart (рисунок 4.4.2).



Рисунок 4.4.1– Структура класса ProgressBlank

Кнопка сохранения заметки о здоровье проверяет, чтобы поле Вес было заполнена в форме х,х, что позволяет с большей точностью строить график, показывающий изменение веса питомца за определенный промежуток времени.

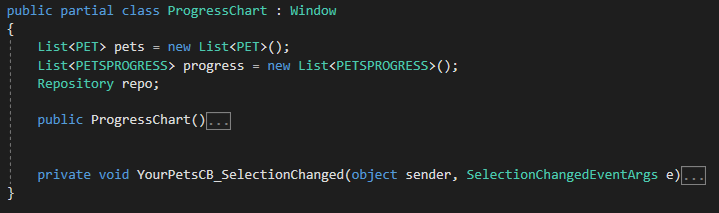


Рисунок 4.4.2 – Структура класса ProgressChart

**5. Руководство пользователя**

При запуске приложения открывается окно авторизации (рисунок 5.1), на котором имеются поля для инициализации пользователя, кнопка для регистрации и кнопка изменения языка.

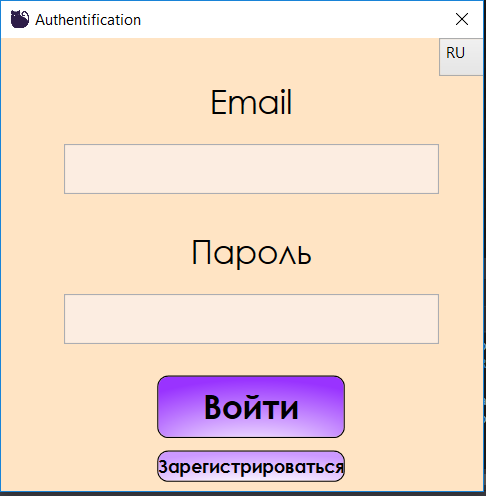


Рисунок 5.1 – Окно авторизации

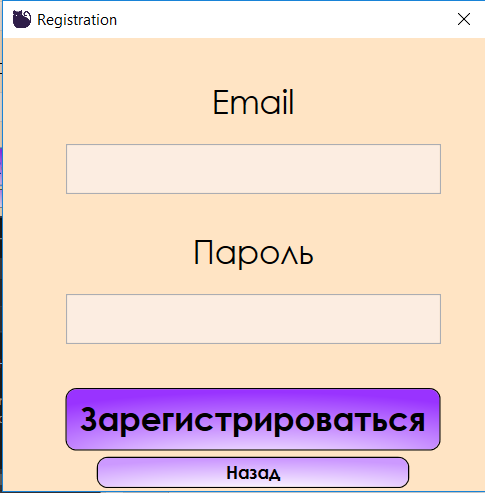


Рисунок 5.2 – Окно регистрации

После успешного прохождения авторизации пользователю будет представлено главное окно приложения, представленное на рисунке 5.3.

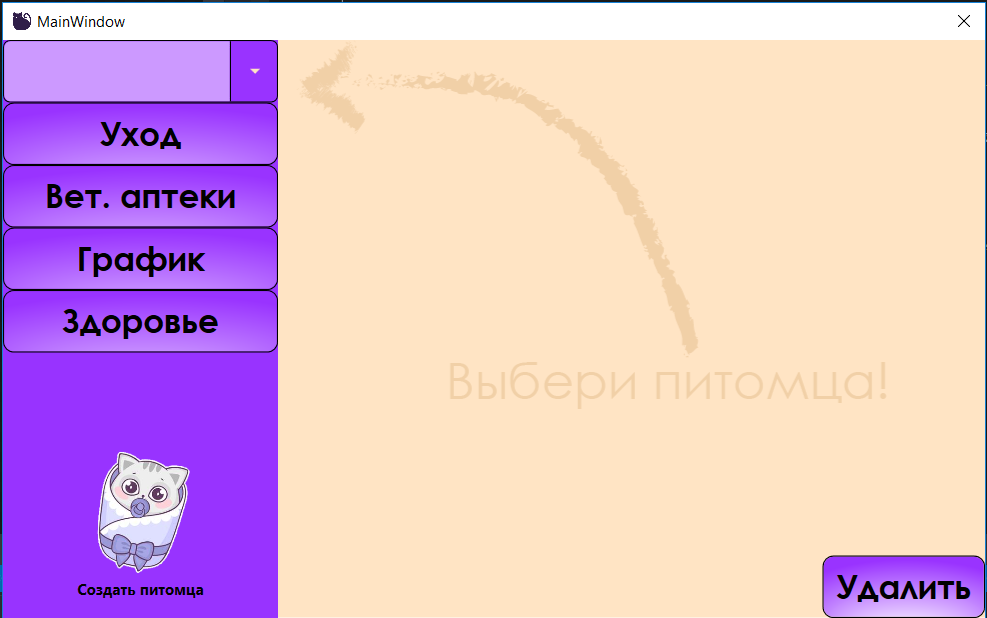


Рисунок 5.3 – Главное окно приложения

На главном окне расположено 6 кнопок для вызова соответствующих методов.

При нажатии на кнопку «Уход» появится окно, содержащее информацию о уходе за питомцем, вакцинациях и заболеваниях (рисунок 5.4).

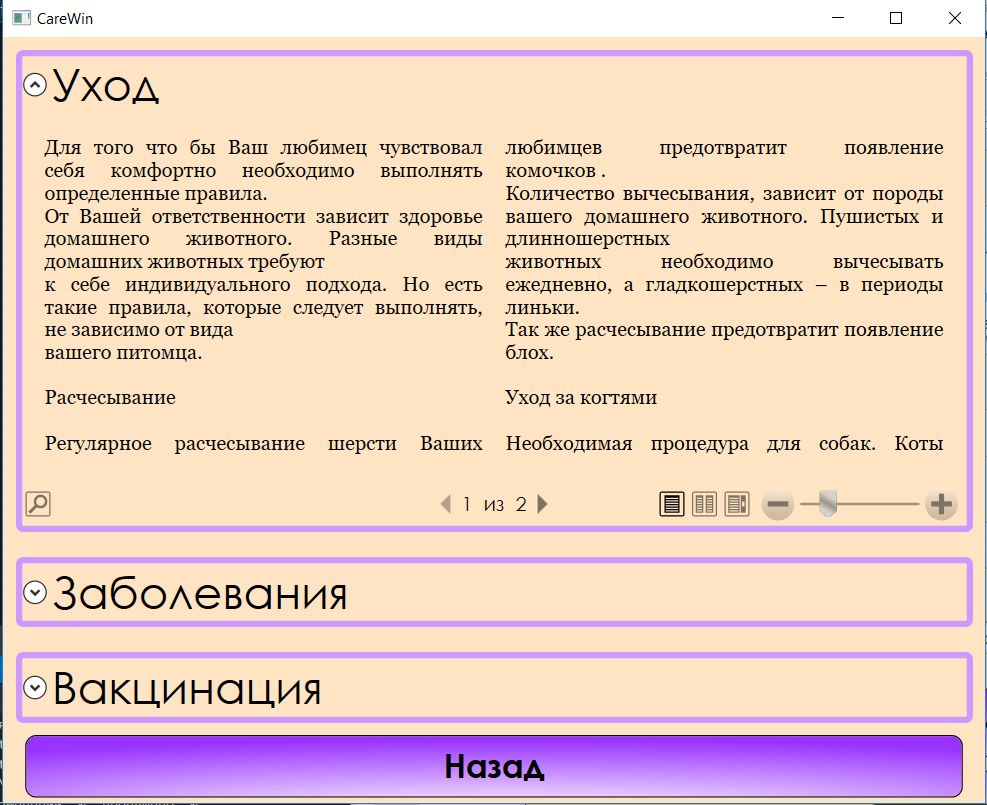


Рисунок 5.4 – Окно информации о уходе за питомцем

Создать нового питомца пользователь может, нажать на кнопку Создать питомца. Откроется окно для заполнения информацией. Пользователю будет представлен список пород, соответствующий выбранному типу питомца (Кот или Собака). Также есть функция добавления фотографии питомца. После нажатия на кнопку Создать питомца, открывается окно для загрузки фотографии (рисунок 5.5).

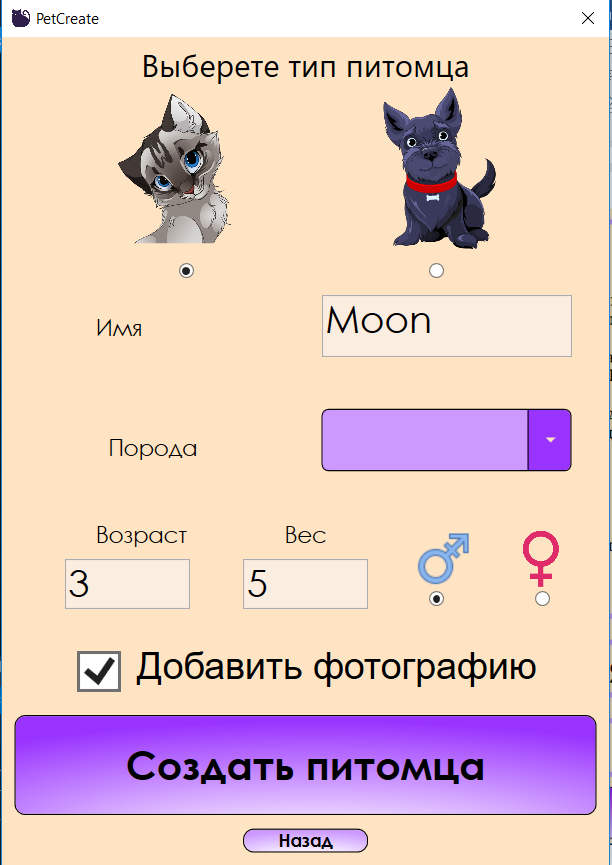


Рисунок 5.5 – Окно создания питомца

При выборе питомца в списке, открывается информация о нем и фотография (рисунок 5.6).

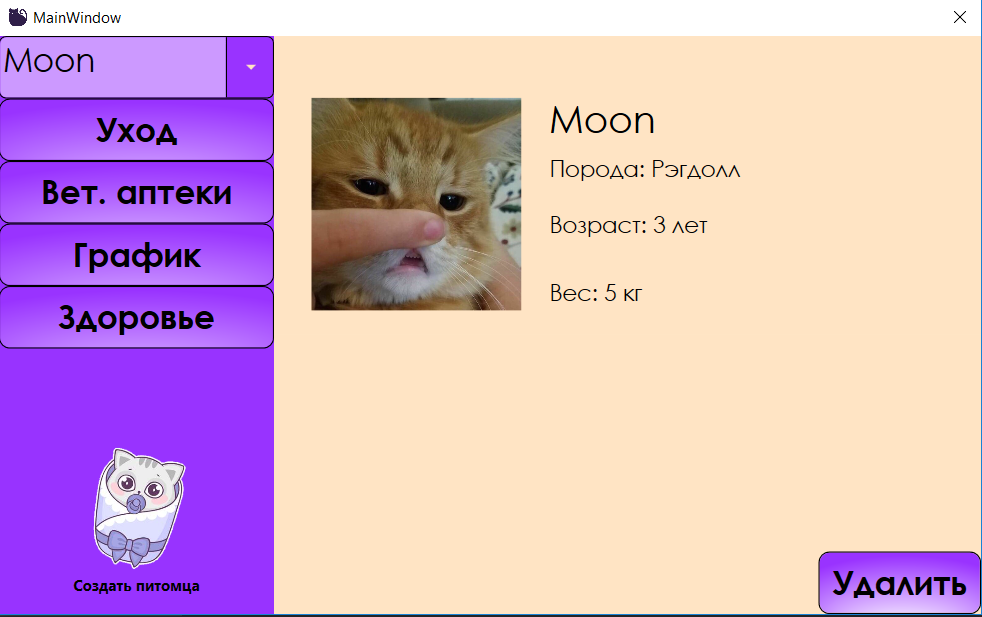


Рисунок 5.6 – информация о питомце

Нажав кнопку Удалить. Выбранный в списке питомец будет удален.

При нажатии на кнопку «Здоровье» пользователю будет представлено окно, позволяющее оставить заметки о состоянии здоровья питомца в определенную дату (рисунок 5.7).

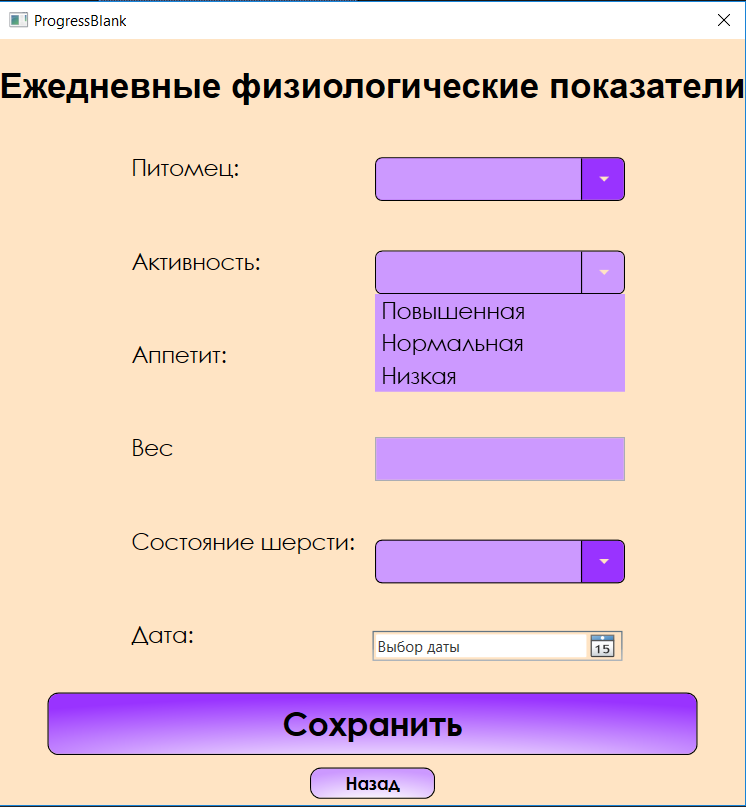


Рисунок 5.7– Окно заметок

Просмотр статистики изменения веса питомца осуществляется в окне, открывающемся нажатием кнопки График (рисунок 5.8). При выборе питомца в списке, будет построена линейная диаграмма, а также максимальный и минимальный вес.

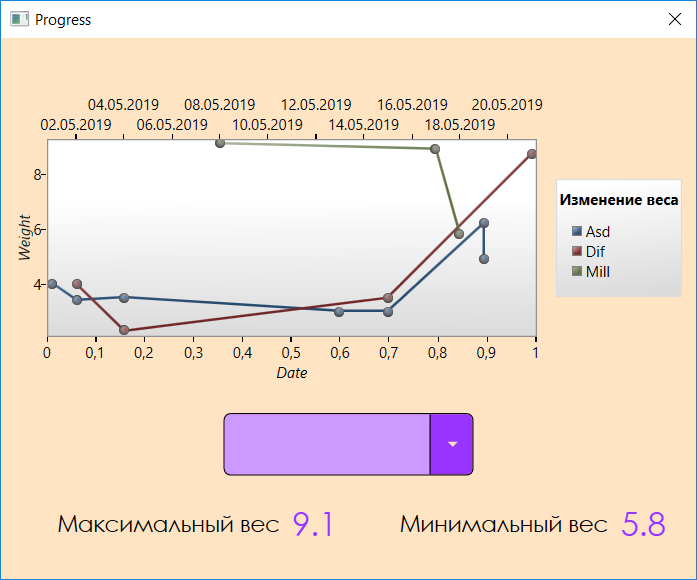


Рисунок 5.8 – График

**ЗАКЛЮЧЕНИЕ**

В курсовой работе описана и реализована программа, написанная на языке C#, с использованием технологий Entity Framework, WPF.

В ходе выполнения курсовой работы было создано приложение учета питомцев, которое выполняет следующие функции:

* Создание своего питомца;
* Оставлять заметки о состоянии питомца в определенную дату;
* Следить за весом и здоровьем питомца;
* Получать рекомендацию по уходу за питомцем в случае проблем с шерстью, зубами и другими;
* Загрузка фотографий.
* локализацию языка.

Данное программное средство имеет удобный и понятный интерфейс, реагирует на ошибочный ввод данных (например, ввод данных не того типа, проверка на пустоту вводимых значений).

В соответствии с полученным результатом работы программы можно сделать вывод, что разработанная программа работает верно, а требования технического задания выполнены в полном объеме.

**СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ**

1. MSDN сеть разработчиков в Microsoft [Электронный ресурс] — http://msdn.microsoft.com/library/rus/
2. Язык C# и .NET Framework [Электронный ресурс] — http://professorweb.ru/my/csharp/charp\_theory/level1/infonet.php
3. Пацей, Н.В. Курс лекций по языку программирования C# / Н.В. Пацей. – Минск: БГТУ, 2016. – 175 с.
4. Мэтью Макдональд — WPF: Windows Presentation Foundation в .NET 4.0 с примерами на C#

**ПРИЛОЖЕНИЕ А**

Скрипт базы данных MyPet:

--create database PetsApp;

use PetsApp;

drop table PETS;

drop table USERS;

drop table KINDS;

drop table PETTYPES;

create table USERS

(

userID int identity(1,1) primary key,

userLOGIN nvarchar(30) not null,

userPASSWORD nvarchar(50) not null

)

create table PETTYPES

(

id int primary key identity(1,1),

petTYPE nvarchar(30) not null

)

insert into PETTYPES(petTYPE)

values

('Кот'),

('Собака');

select \* from PETTYPES;

create table KINDS

(

id int primary key identity(1,1),

petKIND nvarchar(50) not null,

typeID int foreign key (typeID)

references PETTYPES(id)

)

create table PETS

(

petID int primary key identity(1,1),

userID int foreign key(userID)

references USERS(userID) not null,

petTYPE int foreign key(petTYPE)

references PETTYPES(id),

petNAME nvarchar(20) default('Name'),

petAGE int check(petAGE > 0),

petWEIGHT int ,

petGENDER nvarchar(1) default('м'),

petKIND int foreign key (petKIND)

references KINDS(id),

PHOTO varbinary(MAX)

)

create table PETSPROGRESS

(

id int primary key identity(1,1),

petID int foreign key(petID)

references PETS(petID),

ACTIVITY nvarchar(30) ,

WEIGHTcur double precision,

APET nvarchar(30) ,

HEAR nvarchar(30) ,

DATEcurr date,

)

drop table CARE;

create table CARE

(

id int primary key identity(1,1),

Care nvarchar(MAX),

Ills nvarchar(MAX),

Vacina nvarchar(MAX)

)

insert into KINDS(petKIND, typeID)

values

('Абиссинская',1),

('Австралийская дымчатая',1),

('Азиатская короткошерстная',1),

('Американская длинношерстная',1),

('Американская жесткошерстная',1),

('Американская короткошерстная',1),

('Американский бобтейл',1),

('Американский керл',1),

('Баленезийская',1),

('Бамбино',1),

('Бенгальская',1),

('Бомбейская',1),

('Британская короткошерстная',1),

('Британская короткошерстная',1),

('Бурманская',1),

('Бурмилла',1),

('Гавана-браун',1),

('Девон-рекс',1),

('Донской сфинкс',1),

('Европейская короткошерстная',1),

('Египетская мау',1),

('Йоркская шоколадная',1),

('Канадский сфинкс',1),

('Као-мани',1),

('Кимрик',1),

('Корат',1),

('Корниш-рекс',1),

('Курильский Бобрейл',1),

('Лаперм',1),

('Манчкин',1),

('Мейн-кун',1),

('Мэнская кошка',1),

('Невская маскарадная',1),

('Немецкий рекс',1),

('Нибелунг',1),

('Норвежская лесная',1),

('Ориентальная',1),

('Оцикет',1),

('Персидская',1),

('Петербургский сфинкс',1),

('Пикси-боб',1),

('Русская голубая',1),

('Рэгдолл',1),

('Саванна',1),

('Сиамская',1),

('Сибирская',1),

('Австралийская овчарка',2),

('Австралийский терьер',2),

('Австралийской хилер',2),

('Акина ину',2),

('Стаффордширский терьер',2),

('Английский бульдог',2),

('Спаниель',2),

('Английский сеттер',2),

('Аргентинский дог',2),

('Афганская борзая',2),

('Бассенджи',2),

('Бельгийская овчарка',2),

('Бельгийский гриффон',2),

('Боксер',2),

('Бордер колли',2),

('Бультерьер',2),

('Корги',2),

('Шпиц',2),

('Далматинец',2),

('Доберман',2),

('Ретривер',2),

('Ирландский волкодав',2),

('Йоркширский терьер',2),

('Кавказская овчарка',2),

('Лабрадор ретривер',2),

('Мопс',2),

('Мастиф',2),

('Немецкая овчарка',2),

('Немецкий дог',2),

('Папийон',2),

('Пекинес',2),

('Пудель',2),

('Самоедская лайка',2),

('Ротвейлер',2),

('Сенбернар',2),

('Такса',2),

('Чау Чау',2),

('Чихуахуа',2),

('Хаски',2),

('Японский хин',2);

insert into CARE(Care, Ills, Vacina)

values (…………….)

**ПРИЛОЖЕНИЕ Б**

Листинг класса Repository:

public class Repository : IRepository<USER,KIND,PET, PETSPROGRESS>

{

static DContext db;

static private USER currUser;

public static string lang = "US";

public static USER CurrUser { get => currUser; set => currUser = value; }

public Repository()

{

db = new DContext();

}

public IEnumerable<USER> GetUsersList()

{

return db.USERS.ToList();

}

public IEnumerable<KIND> GetKindsList()

{

return db.KINDS.ToList();

}

public IEnumerable<PETSPROGRESS> GetProgressList()

{

return db.PETSPROGRESSes.ToList();

}

public USER GetUser(int id)

{

USER user = new USER();

try

{

user = db.USERS.Find(id);

return user;

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

return user;

}

}

public IEnumerable<KIND> GetKinds(int id)

{

return db.KINDS.ToList().Where(p => p.typeID == id);

}

public IEnumerable<PET> LoadUsersPetsInto(int userID)

{

List<PET> pets = new List<PET>();

try

{

pets = db.PETS.Where(p => p.userID == userID).ToList();

return pets;

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

return pets;

}

}

public IEnumerable<PET> PetInfo(string petName)

{

List<PET> pets = new List<PET>();

try

{

pets = db.PETS.Where(p => p.petNAME == petName).ToList();

return pets;

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

return pets;

}

}

public USER GetUser(string login, string password)

{

USER user = new USER();

try

{

user = db.USERS.FirstOrDefault(p => p.userLOGIN.Equals(login) && p.userPASSWORD.Equals(password));

return user;

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

return user;

}

}

public PET GetPet(int id)

{

PET pet = new PET();

try

{

pet = db.PETS.Find(id);

return pet;

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

return pet;

}

}

public KIND GetKind(int id)

{

KIND kind = new KIND();

try

{

kind = db.KINDS.Find(id);

return kind;

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

return kind;

}

}

public bool CreateNewUser(USER user)

{

try

{

var tempUser = db.USERS.FirstOrDefault(u => u.userLOGIN.Equals(user.userLOGIN));

if (tempUser == null)

{

using (MD5 md5hash = MD5.Create())

{

var hashPass = md5Hasher.GetMd5Hash(md5hash, user.userPASSWORD);

user.userPASSWORD = hashPass;

db.USERS.Add(user);

db.SaveChanges();

Repository.CurrUser = user;

}

return true;

}

return false;

}

catch(Exception ex )

{

MessageBox.Show(ex.Message);

return false;

}

}

public bool CreateNewPet(string name, string kind, string weight, string age, bool gender, bool type, bool photo)

{

try

{

if (type == false)

{

PET cat = new PET();

cat.userID = CurrUser.userID;

cat.petNAME = name;

cat.petKIND = int.Parse(kind);

cat.petWEIGHT = int.Parse(weight);

cat.petAGE = int.Parse(age);

cat.petTYPE = 1;

if (gender == false)

{

cat.petGENDER = "м";

}

else

{

cat.petGENDER = "ж";

}

if (photo)

{

OpenFileDialog dlg = new OpenFileDialog();

dlg.InitialDirectory = "";

dlg.Filter = "Image files (\*.jpg,\*.png,\*.bmp)|\*.jpg;\*.png;\*.bmp|All Files (\*.\*)|\*.\*";

if (dlg.ShowDialog() == true)

{

byte[] imageData;

string selectedFileName = dlg.FileName;

using (FileStream fs = new FileStream(selectedFileName, FileMode.Open))

{

imageData = new byte[fs.Length];

fs.Read(imageData, 0, imageData.Length);

}

cat.PHOTO = imageData;

}

}

db.PETS.Add(cat);

db.SaveChanges();

}

else

{

PET dog = new PET();

dog.userID = CurrUser.userID;

dog.petNAME = name;

dog.petKIND = int.Parse(kind);

dog.petWEIGHT = int.Parse(weight);

dog.petAGE = int.Parse(age);

dog.petTYPE = 2;

if (gender == false)

{

dog.petGENDER = "м";

}

else

{

dog.petGENDER = "ж";

}

if (photo)

{

OpenFileDialog dlg = new OpenFileDialog();

dlg.InitialDirectory = "";

dlg.Filter = "Image files (\*.jpg,\*.png,\*.bmp)|\*.jpg;\*.png;\*.bmp|All Files (\*.\*)|\*.\*";

if (dlg.ShowDialog() == true)

{

byte[] imageData;

string selectedFileName = dlg.FileName;

using (FileStream fs = new FileStream(selectedFileName, FileMode.Open))

{

imageData = new byte[fs.Length];

fs.Read(imageData, 0, imageData.Length);

}

dog.PHOTO = imageData;

}

}

db.PETS.Add(dog);

db.SaveChanges();

}

return true;

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

return false;

}

}

public BitmapImage ConvertByteArrayToImage(byte[] array)

{

if (array != null)

{

using (var ms = new MemoryStream(array, 0, array.Length))

{

var image = new BitmapImage();

image.BeginInit();

image.CacheOption = BitmapCacheOption.OnLoad;

image.StreamSource = ms;

image.EndInit();

return image;

}

}

return null;

}

//private byte[] ConvertImageToByteArray(string fileName)

//{

// Bitmap bitMap = new Bitmap(fileName);

// ImageFormat bmpFormat = bitMap.RawFormat;

// var imageToConvert = Image.FromFile(fileName);

// using (MemoryStream ms = new MemoryStream())

// {

// imageToConvert.Save(ms, bmpFormat);

// return ms.ToArray();

// }

//}

public void AddProgress(PETSPROGRESS progress)

{

try

{

db.PETSPROGRESSes.Add(progress);

db.SaveChanges();

}

catch(Exception ex)

{

MessageBox.Show(ex.Message);

}

}

public void Dispose() { }

public List<string> GetCare()

{

return db.CAREs.Select(c => c.Care1).ToList();

}

public List<string> GetIlls()

{

return db.CAREs.Select(c => c.Ills).ToList();

}

public List<string> GetVacina()

{

return db.CAREs.Select(c => c.Vacina).ToList();

}

public void DeletePet(string name)

{

PET pet = PetInfo(name).Single();

List<PETSPROGRESS> progress = new List<PETSPROGRESS>();

progress = db.PETSPROGRESSes.Where(p => p.petID == pet.petID).ToList();

if (progress.Count > 0)

{

foreach (PETSPROGRESS p in progress)

{

db.PETSPROGRESSes.Remove(p);

}

}

db.PETS.Remove(db.PETS.Where(p => p.petID == pet.petID).Single());

db.SaveChanges();

}

}