



ДИПЛОМЕН ПРОЕКТ

НА ТЕМА

ПЛАТФОРМА ЗА ДОБРОВОЛНА ВЗАИМНОПОМОЩ ЗА ДОМАШНИ ЛЮБИМЦИ

“Pet Exchange”

(“Нов дом за щастливи опашки”)

Ученик:

Виктор Божидаков Зашев

Професия: Системен програмист

Специалност: Системно програмиране

Ръководител консултант:

д-р. Янислав Картелов

гр. Пловдив, 2025



Съдържание

I.	Увод	4
II.	Основна част.....	5
1.	Цел на ДП.....	5
2.	Проучване на съществуващи приложения.	6
3.	Обхват на проекта	7
4.	Технологии.....	8
4.1	.NET 8	8
4.2	Visual Studio 2022.....	9
4.3	C#.....	9
4.4	ASP.NET CORE	9
4.5	MSSQL Server.....	9
4.6	MSSQL Server Management Studio	10
4.7	Entity Framework Core	10
4.8	LINQ.....	10
4.9	ASP.NET Core Identity.....	10
4.10	Уеб браузър.....	11
4.11	HTML	11
4.12	CSS	11
4.13	JavaScript.....	11
4.14	Razor.....	12
4.15	Bootstrap Framework	12
4.16	NUnit	12
5.	Реализация	13
5.1	Функционалност	13
5.1.1	База Данни	14
5.2	Системни изисквания	16



5.2.1. Хардуерни изисквания.....	16
5.2.1. Софтуерни изисквания	16
5.3 Експлоатация.....	16
5.4 Архитектурен дизайн	17
5.4.1 Data Layer.....	17
5.4.2 Business Layer	42
5.4.3 ASP.NET MVC (Presentation Layer)	44
5.4.4 Test Layer (Компонентно тестване)	53
5.4.5 Осигуряване на сигурност	55
6. Авторски права	55
III. Заключение	56
IV. Списък на използвана литература	58
V. Приложения	60



I. Увод

Домашните любимци заемат специално място в живота на хората. Те не само предлагат компания и безусловна обич, но също така допринасят за психическото и физическото благосъстояние на своите стопани. Изследвания показват, че притежаването на домашен любимец може да намали нивата на стрес, да подобри настроението и дори да намали риска от сърдечносъдови заболявания. Освен това, животните често играят роля в терапевтични процеси, подпомагайки хора с различни физически и психически състояния.

Обстоятелствата, при които стопаните на животните понякога изпитват промени в личния или професионалния си живот, могат да ги поставят в ситуация, в която вече не могат да осигуряват необходимата грижа за тях. В такива случаи намирането на подходящ нов дом за животното става приоритет. Често хората прибегват до социални мрежи или приюти, но тези методи не винаги гарантират най-добрия резултат както за животното, така и за новите му стопани. Ето защо е оправдана нуждата от специализирана платформа, която да улесни доброволната размяна на домашни любимци между хора, които могат да предложат адекватна грижа и подходяща среда.

Настоящият дипломен проект предлага разработването на платформа за доброволна размяна на домашни любимци, която ще свързва собственици на животни, желаещи да ги предадат на друг стопанин, с хора, търсещи нов любимец. Чрез използване на съвременни технологии платформата ще улесни процеса на намиране на най-подходящото място за всяко животно. Основната цел на проекта е да осигури по-безопасна, удобна и ефективна алтернатива за доброволна размяна на домашни любимци, която да минимизира стреса за животните и да осигури най-добрите възможни условия за тяхното отглеждане.

II. Основна част

1. Цел на ДП.

Основната цел на проекта е да създаде интуитивна и ефективна платформа (с адаптивен дизайн), която да улесни доброволната размяна на домашни любимци, като осигури безопасен и удобен процес както за настоящите, така и за бъдещите стопани. Архитектурата на системата ще бъде многослойна и графичният дизайн ще се бъде създаден чрез ASP.NET MVC уеб приложение. Проектът ще включва слой за компонентно тестване.

Ще се проведе проучване на вече съществуващи решения на българския пазар за взаимнапомощ на домашни любимци. Ще бъде направено сравнение между тях и ще бъдат подчертани техните главни функционалности и недостатъци.

Чрез платформата потребителите ще могат да регистрират своите животни за осиновяване, предоставяйки подробна информация за тях, включително снимки и описание. След създаването на профил, потребителят ще има достъп до база данни с всички налични животни, търсещи нов дом. Той ще може да търси и сортира предложенията по различни критерии, като тип животно, порода, възраст, дали е включена с клетка и др., за да намери най-подходящия домашен любимец според своите предпочитания.

Когато потребителят намери животно, което го интересува, той ще може да изпрати запитване до настоящия стопанин. Стопанинът от своя страна ще може да прегледа потенциалния нов собственик, да отговори на въпросите му и да му предостави детайли за контакт. По този начин ще се създаде директна връзка между настоящия и бъдещия стопанин, което ще увеличи шансовете за намиране на най-подходящия дом за всяко животно.

Тази система ще улесни процеса на намиране на нов дом за домашни любимци, като гарантира, че те попадат в ръцете на отговорни и грижовни стопани. Освен това, платформата ще съкрати времето за осиновяване и ще намали натоварването върху приютите за животни, като предлага алтернатива за директна размяна между собственици.

Потребител без регистрация ще има достъп само до основната информация за няколко животни търсещи си дом.

2. Проучване на съществуващи приложения.

В България съществуват няколко платформи, насочени към осиновяване и дарения на домашни любимци.

Например, сайтът **"Осинови Ме България"** предлага възможност за осиновяване на бездомни животни, като публикува обяви с базова информация за животните, но липсват по-сложни функционалности като филтриране по град, порода и специфични нужди на животните, както и директна комуникация между потребителите. [1]

Друга иновативна инициатива е платформата **"Pet Buddy"**, която се фокусира върху персонализирана грижа за домашни любимци чрез месечни "кутии" с продукти за отглеждане. Въпреки че тя успешно обслужва сегмента на потребителите, които искат да осигурят качествени продукти за своите животни, платформата не предоставя функция за доброволна размяна на животни, което ограничава нейния обхват за тези, които се нуждаят от нов дом за своите любимци. [2]

Платформата **"My-PetPal"** е още един пример, който помага на стопани да открият изгубените си животни и да свържат собственици с нови стопани. Въпреки това, тя е предимно насочена към намиране на изгубени животни, а не към целенасочен обмен или осиновяване поради промяна в обстоятелствата на стопанина. [3]

Предложената платформа за доброволна размяна на домашни любимци има за цел да запълни пропуските в съществуващите системи. Докато настоящите платформи се фокусират основно върху осиновяването на бездомни животни или предлагането на продукти за грижа, новата система ще предлага възможност за регистрация както на животни, търсещи нов дом, така и на потенциални стопани, които могат да ги намерят лесно чрез детайлно филтриране.

Допълнително, платформата ще осигури директна комуникация¹ между потребителите, което значително ще ускори и улесни процеса на осиновяване. Така се

¹ Комуникацията ще е в опростен вид – съобщение за питане и отговор. "Чат" функционалността ще бъде изградена в бъдещо развитие.

постигат две основни цели: подобряване на ефективността на процеса по намиране на нов дом за животните и минимизиране на стреса за тях в преходния период.

3. Обхват на проекта

Проектът цели разработването на интуитивна и функционална онлайн платформа, която да улесни доброволната размяна на домашни любимци в България. Системата ще бъде достъпна чрез интернет и ще функционира като уебсайт, предоставящ централизирано място за публикуване на обяви за животни, които търсят нов дом.

Основните функционалности ще са:

1. Потребителски профили и автентикация

Потребителите ще могат да се регистрират и управляват своите профили чрез въвеждане на потребителско име и парола. Автентикацията е осъществена изцяло чрез вътрешни механизми и не включва възможност за влизане чрез Google или други профили от трети страни.

2. Административен контрол

Платформата ще разполага с отделни администраторски профили, които ще осъществяват контрол върху съдържанието, публикувано от потребителите, както и върху цялостната поддръжка и сигурност на системата.

3. Географски обхват

Проектът обхваща всички областни градове в България.

4. Поддръжка на всички видове домашни любимци

Системата ще поддържа обяви за всички видове животни, включително кучета, котки, малки животни и дори коне. Това осигурява възможност за разнообразен избор на животни според нуждите и предпочитанията на потребителите.

Ограниченията на системата са²:

1. Липса на директна комуникация

² Тези ограничения са предвидени да се премахнат в бъдещото развитие на проекта.

Чат функционалности и възможност за директен контакт между потребителите няма да бъдат интегрирани в системата. Това означава, че комуникацията относно обявите ще се осъществява чрез предоставяне на основни данни за контакт, като по-нататъшната комуникация ще бъде осъществена чрез други социални платформи или директно между потребителите.

2. Географска информация само на ниво областен град

Системата няма опция за избиране на точно географско положение или GPS координати на животното. При публикуване на обява, потребителят трябва да посочи само областния град, в който се намира животното. Това ограничава възможността за по-прецизно филтриране на местоположението, но отразява практическото предположение, че потребителите били могли да посетят най-близкия до тях областен град за осиновяване.

4. Технологии

В този раздел ще бъдат разгледани основните технологии и инструменти, използвани при разработката на платформата. Ще се фокусираме върху програмните езици, рамки(“frameworks”) и технологии, които осигуряват функционалността и ефективността на системата.

4.1 .NET 8

.NET 8 е последната версия на универсалната платформа на Microsoft с дългосрочна поддръжка (LTS), която предлага значителни подобрения в производителността, паметта и сигурността. Тя позволява създаването на бързи, кросплатформени приложения с подобрена интеграция с облачни услуги и поддръжка на ARM64. Благодарение на усъвършенстваните минимални API, оптимизираната работа с данни чрез LINQ и съвместимостта с Visual Studio, .NET 8 е идеален избор за разработчици, които търсят стабилна и ефективна среда за съвременна разработка на софтуер. [4]

4.2 Visual Studio 2022

Visual Studio 2022 е мощна среда за разработка (IDE) от Microsoft, която поддържа над 36 програмни езика и предлага инструменти като IntelliCode, Git интеграция и усъвършенстван дебъгер. Тя е идеална за създаване на настолни, уеб, мобилни и облачни приложения, като предоставя както безплатна версия (Community), така и професионални издания. Разработчиците я избират заради удобния интерфейс, интелигентните инструменти за кодиране и широката ѝ екосистема. [5]

4.3 C#

C# е модерен, обектно-ориентиран език за програмиране, разработен от Microsoft, широко използван за настолни, уеб, мобилни и игрови приложения. Той е силно типизиран и разполага с автоматично управление на паметта (garbage collector), което улеснява разработчиците. Благодарение на поддръжката на асинхронно програмиране (широко използвано в проекта), C# позволява бърза и ефективна работа на приложенията, подобрявайки потребителското изживяване. [6]

4.4 ASP.NET CORE

ASP.NET е мощен уеб “framework” от Microsoft, който позволява разработката на сигурни, мащабируеми и динамични уеб приложения. Той поддържа различни архитектурни модели като MVC, Web API и Blazor, както и интеграция с технологии като SignalR и Identity Core(технология, която се ползва в проекта за автентикация и оторизация). Благодарение на своята гъвкавост и висока производителност, ASP.NET е подходящ избор за съвременни уеб приложения с разнообразни функционалности. [7]

4.5 MSSQL Server

MSSQL Server е релационна база данни, осигуряваща висока достъпност, мащабируемост, сигурност и производителност. Той поддържа различни типове данни и сложни заявки, улеснявайки управлението на транзакции и съхранението на информация. В комбинация с Entity Framework (Вж. [4.7 EntityFramework](#)), MSSQL позволява ефективно управление на данните без необходимост от ръчно писане на SQL заявки. [8]

4.6 MSSQL Server Management Studio

SSMS (SQL Server Management Studio) е инструмент с графичен интерфейс за управление и администриране на MSSQL Server. Той позволява създаване и конфигуриране на бази данни, изпълнение на SQL заявки и мониторинг на сървърите. SSMS предлага широк набор от инструменти за ефективно управление, мониторинг и поддръжка на бази данни, улеснявайки работата на администраторите и разработчиците. [9]

4.7 Entity Framework Core

Entity Framework Core (EF Core) е ORM³ решение от Microsoft, позволяващо работа с базата данни чрез обекти и LINQ заявки. EF Core поддържа различни бази данни и предлага два подхода за създаване на приложения – Code First и Database First. В проекта е използван Code First подход, при който се дефинират класовете, след което EF Core генерира базата данни с необходимите връзки и ключове. [10]

4.8 LINQ

Language-Integrated Query (LINQ) е набор от технологии, които интегрират възможности за заявки директно в езика C#. Вместо да използвате различни езици за заявки за различни източници на данни, LINQ позволява да пишете заявки срещу типизирани обекти, използвайки познати езикови ключови думи и оператори. LINQ се използва заради улесненото писане на сложни алгоритми с работа на данни, което тази технология предлага. [11]

4.9 ASP.NET Core Identity

ASP.NET Core Identity е система за управление на потребители, роли и удостоверяване в ASP.NET уеб приложения. Тя осигурява оторизация за контрол на достъпа и персонализиране на процесите на автентикация и оторизация, за да отговори на специфичните изисквания на приложението. Поддържа различни методи за удостоверяване, включително локални акаунти, социални влизания и двуфакторно

³ ORM (Object-Relational Mapping) технологиите позволяват на разработчиците да взаимодействат с базата данни чрез обектно-ориентирани езици като C# или Java, без да се налага да пишат SQL заявки, което освобождава времето им да се съсредоточат върху бизнес логиката.

удостоверяване, както и хеширане и валидиране на пароли за защита срещу уязвимости. В проекта е използван ASP.NET Core Identity за лесно управление на потребителски роли и профили. [12]

4.10 Уеб браузър

Уеб браузърът е приложение, което позволява на потребителите да достъпват и разглеждат уеб страници в интернет. Той предоставя графичен интерфейс за взаимодействие с уеб съдържание, като текст, изображения, видеоклипове и други мултимедийни елементи. По време на разработката на проекта беше използван уеб браузърът Mozilla Firefox. [13]

4.11 HTML

HTML (Hypertext Markup Language) е език, използван за създаване и структуриране на съдържание за уеб. HTML използва тагове за определяне на структурата на уеб страницата, като създава елементи като заглавия, параграфи, списъци, връзки, изображения и форми. HTML е в основата на интернет и често се комбинира с други технологии като CSS и JavaScript, за да се създадат динамични и интерактивни уеб приложения. HTML5, най-новата версия на HTML, е използвана в проекта. [14]

4.12 CSS

CSS (Cascading Style Sheets) е език за описание на представянето на документ, написан в HTML или XML. Той определя как елементите на HTML да се показват на екрана, в печата или в други медии. Използва се за създаване на адаптивни и визуално привлекателни уебсайтове, като позволява прилагането на стилове към конкретни елементи или групи от елементи. CSS също така поддържа адаптивни дизайни, които променят стила на страницата в зависимост от размера на екрана и устройството (платформата използва такъв адаптивен дизайн). CSS се състои от правила, които включват селектори и декларации за стилове и стойности, като може да се записва в HTML документа, в отделен стилев файл или вградено в HTML. [15]

4.13 JavaScript

JavaScript е популярен език за програмиране, използван за създаване на динамични и интерактивни уеб страници. Създаден през 1995 г. от Брендън Айх в

Netscape Communications, JavaScript е „client-side“ скриптов език, който се изпълнява в уеб браузъра на потребителя, като често се комбинира с HTML и CSS за добавяне на интерактивност. JavaScript е обектно-ориентиран и поддържа парадигми на функционалното програмиране, предлагаща масиви, низове и регулярни изрази. Използва се в платформата при избиране и премахване на използваните филтри в страницата “Любимци” на платформата. [16]

4.14 Razor

Razor е синтаксис за създаване на динамични уеб страници в ASP.NET, който комбинира HTML и код от страна на сървъра. В платформата се използват Razor компоненти, които позволяват изграждане на многократно използваем и модулен код. Синтаксисът използва символа "@" за вграждане на C# изрази и оператори в HTML. Razor улеснява създаването на динамични страници, като елиминира нуждата от допълнителни файлове с код и предлага силна проверка на типовете за повишена безопасност и качество на кода. [17]

4.15 Bootstrap Framework

Bootstrap е популярна рамка с отворен код за създаване на адаптивни уеб приложения, ориентирани към мобилни устройства. Създадена през 2011 г. от разработчиците на Twitter⁴, тя предлага CSS стилове и JavaScript компоненти за бързо изграждане на уеб страници. В платформата се използват Bootstrap стилове и икони, които осигуряват гъвкавост и адаптивност. Рамката включва решетъчна система и компоненти като навигационни менюта, бутони и модални прозорци, които улесняват създаването на визуално привлекателни интерфейси, използвани в проекта. [18]

4.16 NUnit

NUnit е популярен “framework” за UNIT тестване, който се използва за изолирано тестване на отделни функционални единици от кода. Той предоставя възможност за автоматизация на тестовете и интеграция в CI/CD процеси. NUnit предлага мощни функции като параметризирани тестове, подробни отчети за грешки и възможност за използване на mocking библиотеки за симулация на зависимости. В платформата се

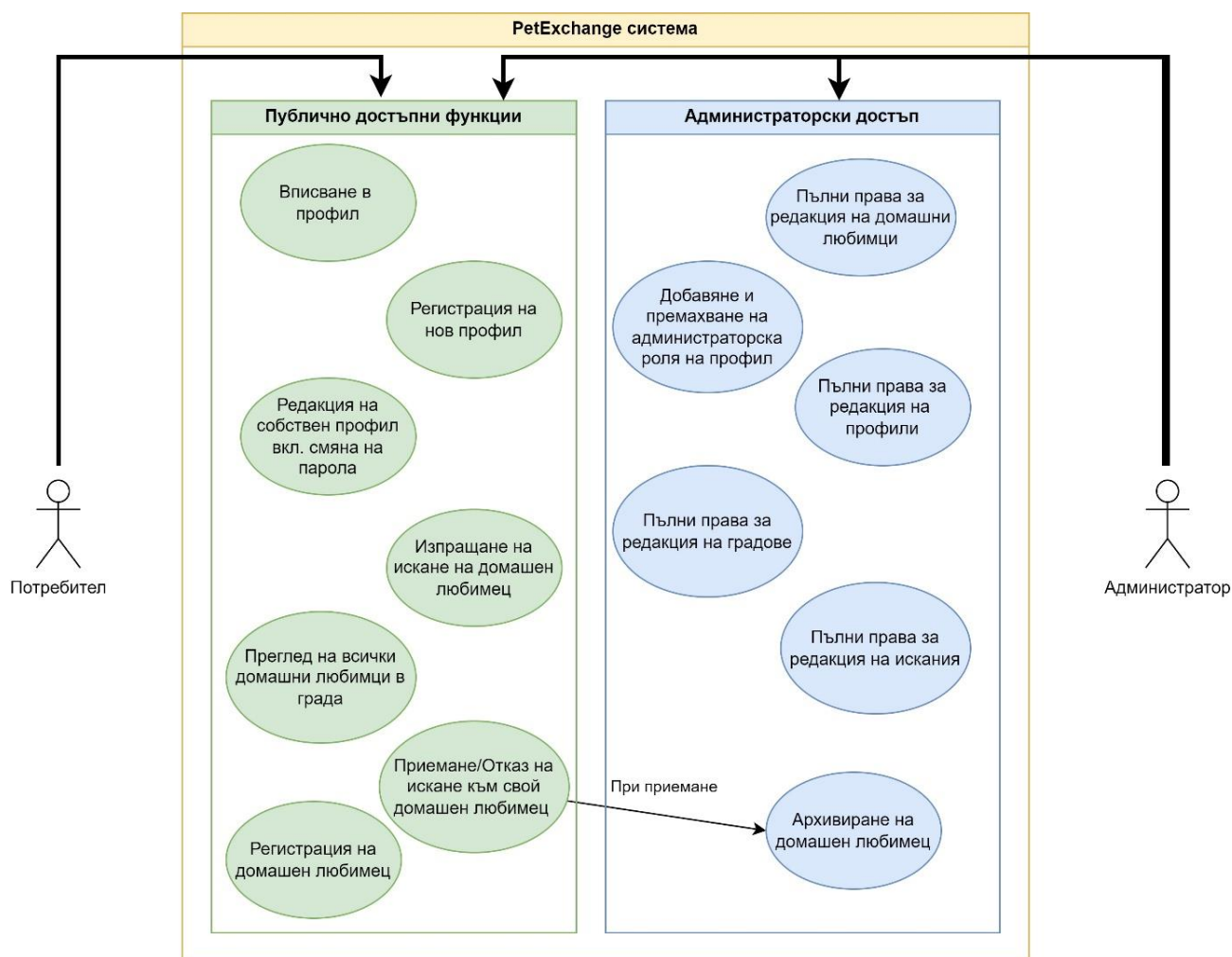
⁴ Сегашната социална мрежа “X”

използва NUnit за тестване на различни компоненти на платформата, което подобри качеството на кода и намали необходимостта от поддръжка. [19]

5. Реализация

В този раздел се описват основните етапи и технически решения, свързани с разработката на софтуерния продукт, както и процесът на изграждане и интегриране на всички функционалности в рамките на проекта.

5.1 Функционалност



Фигура 1. “Use Case” диаграма на платформата

Платформата “Нов дом за щастливи опашки” предоставя различни функционалности както за обикновените потребители, така и за администраторите (Фигура 1.). Те са разделени в две основни категории:

1. Публично достъпни функции

- **Вписване в профил;**
- **Регистрация на нов профил;**
- **Регистрация на домашен любимец;**
- **Редактиране на собствен профил (включително смяна на парола);**
- **Редактиране на собствен домашен любимец;**
- **Преглед на всички домашни любимци, търсещи си дом;**
- **Изпращане на искане на домашен любимец;**
- **Отхвърляне на направено собствено искане;**
- **Приемане/Отказ на чуждо искане към свой домашен любимец** - Ако потребител притежава регистриран домашен любимец в системата, той може да приема или отказва заявки от други потребители. При приемане, регистрираното животно се архивира и спира да бъде показвано публично.

2. Администраторски достъп

- **Пълни права за редактиране на домашни любимци;**
- **Добавяне и премахване на администраторска роля на профил;**
- **Пълни права за редактиране на профили;**
- **Пълни права за преглед на искания;**
- **Архивиране на домашен любимец** – Администраторът може да архивира домашния любимец (“псевдо-изтриване”⁵), дори и без одобрено искане за него.

5.1.1 База Данни

Проектът използва MSSQL Server за базата данни в комбинация с Entity Framework Core, който спомага за работата с нея. Базата данни работи, като съхранява данните в отделни таблици с типизирани колони. Всяка таблица разполага с първичен ключ (primary key), който служи за идентификация и връзка с другите таблици в базата данни. Връзките в нея се осъществяват чрез т.нар. чуждестранен ключ (foreign key).

За създаването на базата данни е използван модела Code First, при който първоначално разработчикът създава класовете на приложението, които отговарят на

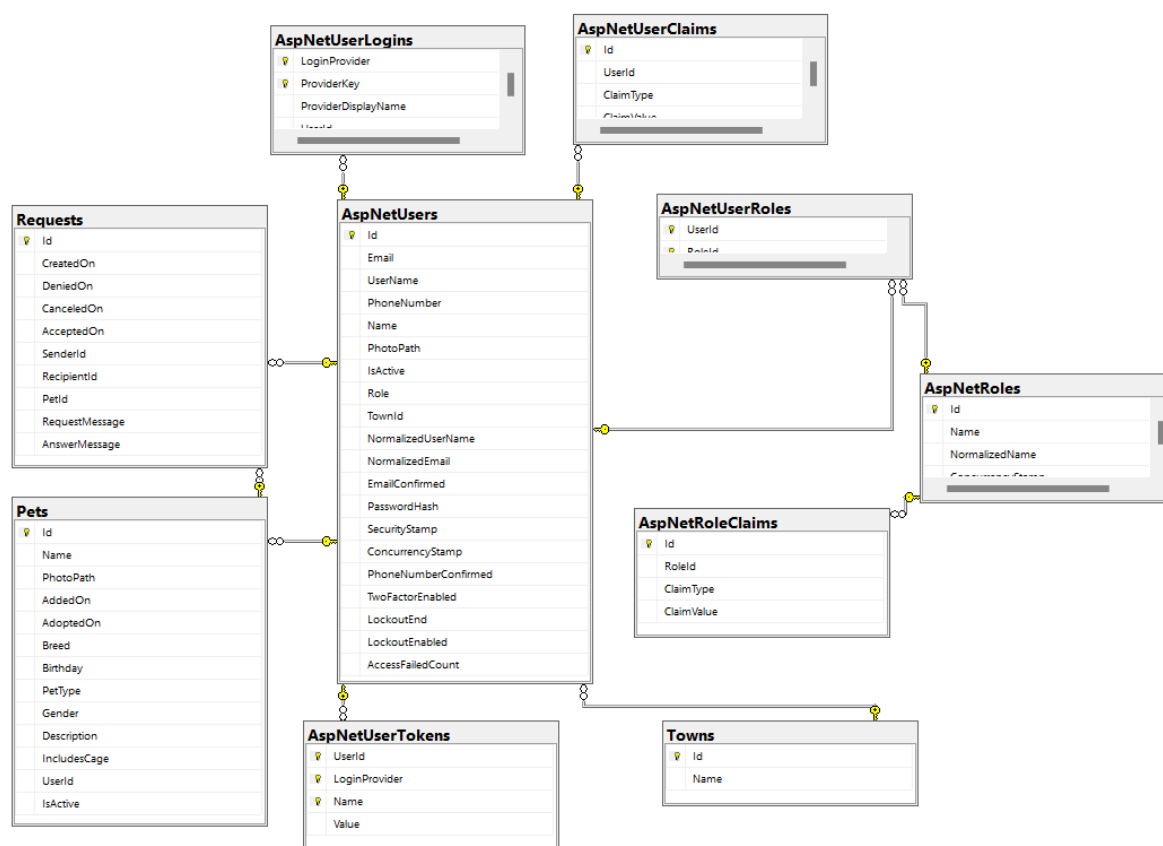
⁵ Повече информация за “псевдо-изтриването” се намира в бележката на “а) Модели” на точка [5.4.1 DataLayer](#).

модела, и техните атрибути и индекси, след което EF Core създава базата данни и връзките между таблиците на база тези класове.

В приложението са използвани следните таблици в базата данни⁶:

1. Users (“AspNetUsers”);
2. Pets;
3. UserRequests;
4. Towns.

На Фигура 2. е показана схемата на базата данни. Моделите на тези таблици и връзките между тях ще бъдат разгледани подробно в [а\) Модели](#) на раздел [5.4.1 Data Layer](#).



Фигура 2. “UML” схема на базата данни

⁶ Не се споменават таблиците, използвани от ASP.NET Core Identity за управлението на ролите на потребителите, автентикация и оторизирането им.

5.2 Системни изисквания

За успешното внедряване и функциониране на разработената платформа, е необходимо да се спазят определени системни изисквания, свързани с хардуерната и софтуерната среда.

5.2.1. Хардуерни изисквания

- **Процесор:** Двухъдрен процесор (Intel Core i3 или AMD еквивалент)
- **RAM:** 2 GB
- **Дисково пространство:** 50 GB SSD
- **Графична карта:** Вградена графика
- **Мрежова връзка:** Интернет връзка с минимална скорост 10 Mbps

5.2.2. Софтуерни изисквания

- **Операционна система** - Windows 2019 Server
- **Необходими софтуерни компоненти:**
 - **ASP.NET Core 8** – за стартиране и разработка на платформата
 - **IIS (Internet Information Services)** – за хостване на уеб сървъра

Съобразяването с горепосочените системни изисквания ще осигури стабилната и надеждна работа на платформата, както и възможност за нейното бъдещо разширяване и поддръжка.

5.3 Експлоатация

Платформата ще може да бъде достъпна чрез интернет връзка. Всеки потребител с уеб браузър, ще може да достъпи и ползва функционалностите ѝ.

Ще бъдат назначени администратори, които да модерират дейностите на потребителите, осигуряващи безопасността на платформата.

Проектът няма да изисква такса за ползване от потребителите. Основният източник на финансиране ще бъде чрез рекламни приходи, които ще покриват разходите за хостинг, поддръжка и бъдещо развитие на платформата. Този модел осигурява

безплатен достъп до услугите на сайта за всички потребители, като същевременно гарантира неговата устойчивост и развитие.

Създадени са тестови акаунти за ползване на платформата, заредени с примерни данни:

1. Администраторски профил:
 - a. Потребителско име: vbzashev
 - b. Парола: 4_sQYgeyu:Cx5-@"TT.e
2. Потребителски профил:
 - a. Потребителско име: goshoUser
 - b. Парола: u-k.KwKn3;LEN:QMf/

5.4 Архитектурен дизайн

В рамките на този раздел ще бъдат разгледани основните компоненти на многослойната архитектура, включително:

- **Слой за данни (DataLayer)** – отговаря за управлението на данните, комуникацията с базата данни и изпълнението на заявки;
- **Бизнес слой (BusinessLayer)** – осигурява посредничество между слоя за данните и потребителския интерфейс;
- **Презентационен слой (ASP.Net MVC)** – отговаря за графическия потребителски интерфейс;
- **Слой за тестване** – гарантира коректността и надеждността на разработеното приложение чрез компонентни тестове.

Чрез анализа на тези слоеве ще бъде демонстрирана организацията на системата, принципите на разработка и използваните технологии. Това ще даде ясна представа за начина, по който различните компоненти комуникират помежду си, и ще покаже предимствата на многослойния архитектурен подход.

5.4.1 Data Layer

Основната част от софтуера, който е ангажиран с организацията и управление на данните, необходими за изпълнение на логиката на приложението.

а) Модели

Класовете “модели” в приложението служат за сформирането на таблици на база на техните полета. Чрез подхода “code-first” и EntityFramework, се създава ORM (Object-Relational Mapping) връзка между моделите и създадените таблици. Имплементирана е валидация на данни чрез анотации в дефиницията на моделите. Допълнителна валидация на данни се прави в слоя ASP.NET MVC (Вж. [5.4.3 ASP.NET MVC](#))

Съществуват 4 модела в приложението:

- Pet;
- Town;
- User;
- UserRequest.

Бележка:

Pet и User, имат поле PhotoPath пазецо пътят към снимката на съответно домашното животно или потребителя. Изображението се пази в директорията wwwroot в проекта в папки pet или user.

Също така Pet и User имат поле IsActive, което определя тяхната видимост в много функции на платформата. При изтриване на модел, той не се премахва от базата данни, но неговата видимост се изключва. Това “псевдо-изтриване” се прави с оглед на мерки за предпазването от случайно изтриване на цялата или голяма част от базата данни при каскада на операцията “изтрий”, както и за да се съхранят данните за бъдещо активиране. Чрез този подход информация никога няма да се изгуби.

Полето IsActive на UserRequest, няма такава “псевдо-изтрий” функция. То служи за визуално определяне на статуса в интернет страницата на приложението и за скриване/показване на бутони за изтрий/приеми.

i. *Pet*

```
public class Pet
{
    [Key]
    99+ references | 32/34 passing
    public Guid Id { get; set; } = Guid.NewGuid();

    [Required]
    [DisplayName("Име")]
    99+ references | 38/38 passing
    public string Name { get; set; } = string.Empty;

    [Required]
    57 references
    public string PhotoPath { get; set; } = string.Empty;

    [Required]
    [DisplayName("Добавен")]
    61 references | 7/7 passing
    public DateTime AddedOn { get; set; }

    [DisplayName("Осиновен")]
    52 references
    public DateTime? AdoptedOn { get; set; }
```

Фигура 3. Модела *Pet*

Моделът *Pet* (Фигура 3.) служи за описването на един домашен любимец в базата данни.

Той има следните полета:

1. *Id*, *Guid* – уникален първичен ключ за таблицата;
2. *Name*, *String* – текстово поле за името на домашния любимец;
3. *PhotoPath*, *String* – текстово поле пазещо пътят към снимката на домашния любимец;
4. *AddedOn*, *DateTime* – поле за дата, служещо за запазването на датата на регистриране на животното в системата;
5. *AdoptedOn*, *DateTime* – поле за дата, служещо за запазването на датата на осиновяване на животното в системата. Първоначално стойността на полето е нулевата стойност;
6. *Breed*, *String* – текстово поле съдържащо името на породата на животното;



7. Birthday, DateTime – поле за дата, служещо за запазване на рождената дата на животното;
8. AgeDays, Int – поле, което присвоява възрастта на животното в дни чрез пресмятането на разликата между текущата дата и рождението;
9. AgeEnum, PetAgeEnum – поле от изброим тип данни, което пази дали животното се определя като възрастно или младо, според това дали е навършило възраст от 90 дни или не; (Вж. [vii. PetAgeEnum](#))
10. PetType, PetTypeEnum – поле от изброим тип данни, което пази типа на животното от множеството поддържани типове за домашен любимец; (Вж. [v. PetTypeEnum](#))
11. Gender, GenderEnum – поле от изброим тип данни, което пази пола на животното; (Вж. [viii. GenderEnum](#))
12. Description, String – текстово поле за кратко описание на животното, написано от неговия стопанин;
13. IncludesCage, Bool – поле пазещо логическа стойност дали животното при осиновяване включва и клетка, или е само то;
14. UserId, Guid – поле съдържащо външният ключ към потребител - стопанина на животното;
15. User, User – поле съдържащо обект от класа User, пазещ информация за стопанина; (Вж. [iii. User](#))
16. IsActive, Bool – поле от тип логическа стойност, определящо дали записът е активен;
17. Town, Town – поле съдържащо обект от класа Town, пазещо града, в който се намира животното. Не е външен ключ, а присвоява данните от обекта User;
18. TownId – пази ключа на града. Не е външен ключ, а присвоява данни от обекта User;
19. UserRequests, List<UserRequest> - списък, осъществяващ връзка “един към много” с таблицата от модел UserRequest и пазещ всички искания за животното от потребители на платформата. (Вж. [iv. UserRequest](#))

ii. *Town*

```
public class Town
{
    [Key]
    48 references | 12/12 passing
    public Guid Id { get; set; } = Guid.NewGuid();

    [Required]
    [DisplayName("Име")]
    54 references | 15/15 passing
    public string Name { get; set; } = string.Empty;

    25 references | 12/12 passing
    public Town() { }

    2 references
    public Town(string name)
    {
        Id = Guid.NewGuid();
        Name = name;
    }

    27 references
    public Town(Guid id, string name)
    {
        Id = id;
        Name = name;
    }
}
```

Фигура 4. Модела *Town*

Моделът *Town* (Фигура 4.) служи за описването на поддържан град в базата данни. На този етап само всички областни градове в България се поддържат.

Той има следните полета:

1. *Id*, *Guid* – уникален първичен ключ за таблицата;
2. *Name*, *String* – текстово поле за името на града.

iii. *User*

```
public class User : IdentityUser<Guid>
{
    [Required(ErrorMessage = "задължително")]
    [DisplayName("Имейл")]
    33 references | 2/2 passing
    public override string? Email { get; set; } = null;

    [Required(ErrorMessage = "задължително")]
    [DisplayName("Потребителско име")]
    42 references | 9/9 passing
    public override string? UserName { get; set; } = null;

    [Required(ErrorMessage = "задължително")]
    [DisplayName("Телефон")]
    30 references | 2/2 passing
    public override string? PhoneNumber { get; set; } = null;

    [Required(ErrorMessage = "задължително")]
    [DisplayName("Име")]
    50 references | 7/7 passing
    public string Name { get; set; } = string.Empty;

    25 references
    public string PhotoPath { get; set; } = string.Empty;
}
```

Фигура 5. Модела User

Моделът User (Фигура 5.) служи за запазването на потребителски, както и администраторски профил в системата. Той наследява класа IdentityUser, използвайки неговите полета и функции за автентикация и оторизация. Също така включва полета за първичен ключ и хеширана парола, които класът User автоматично присвоява. Другите полета на IdentityUser са извън обхвата на този проект и не се използват активно.

Класът User има следните полета:

1. Id, Guid – уникален първичен ключ за таблицата;
2. Email, String – текстово поле за адрес на електронна поща на потребителя;
3. UserName, String – текстово поле за потребителското име на потребителя, което служи за автентикация;
4. PhoneNumber, String – текстово поле за телефонния номер на потребителя;
5. Name, String – текстово поле за името на потребителя;
6. PhotoPath, String – текстово поле, пазещо пътя към файла на профилната снимка, която се показва в платформата;

7. IsActive, Bool – поле от тип логическа стойност, определящо дали записът е активен;
8. Role, RoleEnum - поле от изброим тип, определящо ролята на акаунта; (Вж. [vi. RoleEnum](#))
9. TownId, Guid - поле за външният ключ към таблицата за градовете;
10. Town, Town - поле, пажещо обекта от тип град свързан с профила; (Вж. [ii. Town](#))
11. Pets, List<Pet> - списък, осъществяващ връзка “един към много” с таблицата от модел Pet и пажещ всички регистрирани животни от потребителя; (Вж. [i. Pet](#))
12. RequestOutbox, List<UserRequest> - списък, осъществяващ връзка “един към много” с таблицата от модел UserRequest и пажещ всички изпратени искания към чужди животни от потребителя; (Вж. [iv. UserRequest](#))
13. RequestInbox, List<UserRequest> - списък, осъществяващ връзка “един към много” с таблицата от модел UserRequest и пажещ всички получени искания, изпратени към животни, на които потребителят е стопанин.

iv. *UserRequest*

```
public class UserRequest
{
    [Key]
    83 references | 24/27 passing
    public Guid Id { get; set; } = Guid.NewGuid();

    [Required]
    39 references | 16/16 passing
    public DateTime CreatedOn { get; set; }

    12 references | 2/2 passing
    public DateTime? DeniedOn { get; set; }
    11 references | 2/2 passing
    public DateTime? CanceledOn { get; set; }
    13 references | 2/2 passing
    public DateTime? AcceptedOn { get; set; }
    [Required]
    41 references | 16/16 passing
    public Guid SenderId { get; set; }
```

Фигура 6. Модела UserRequest

Моделът UserRequest (Фигура 6.) служи за запазване на искане на домашен любимец в базата данни.

Той има следните полета:

1. Id, Guid – уникален първичен ключ за таблицата;
2. CreatedOn, DateTime – поле за дата, пазещо кога искането е било създадено;
3. DeniedOn, DateTime – поле за дата, пазещо кога искането е било отказано от потребителя получател. Първоначалната му стойност е нулевата стойност(“null”);
4. CanceledOn, DateTime – поле за дата, пазещо кога искането е било отхвърлено от потребителя, който първоначално го е създал. Първоначалната му стойност е null;
5. AcceptedOn, DateTime – поле за дата, пазещо кога искането е било прието от потребителя получател. Първоначалната му стойност е null;
6. SenderId, Guid – поле за външен ключ към таблицата на модела User. То пази връзката към потребителя, който е създал искането;
7. Sender, User – поле пазещо обект от тип User – това е потребителят “изпращач”; (Вж. [iii. User](#))
8. RecipientId, Guid – поле за външен ключ към таблица на модела User – това е потребителят “получател” на искането;
9. Recipient, User – поле пазещо обект от тип User – това е получателя;
10. PetId, Guid – поле за външен ключ към таблицата на модела Pet – това е домашният любимец, за когото е направено искането;
11. Pet, Pet – поле пазещо обект от тип Pet – това е исканият домашен любимец;
12. RequestMessage, String – текстово поле, пазещо първоначалното съобщение, което получателят ще получи относно искането му на домашния любимец;
13. AnswerMessage, String – текстово поле, пазещо съобщението, което получателят ще изпрати на изпращача. Показва се както при отказ, така и при приемане на искането;
14. IsActive, Bool – логическо поле, което определя дали искането е активно още или не. При получаване на отказ или прием от получателя или отхвърляне от изпращача, искането става неактивно.

б) Помощни класове

В този раздел са разгледани помощни класове, които покриват различни функционалности, допълващи главните модели.

v. *PetTypeEnum*

```
public enum PetTypeEnum
{
    [Description("котки")]
    Cat = 0,
    [Description("кучета")]
    Dog = 1,
    [Description("риби")]
    Fish = 2,
    [Description("малки бозайници")]
    SmallMammal = 3,
    [Description("птици")]
    Bird = 4,
    [Description("влечуги")]
    Reptile = 5,
    [Description("земноводни")]
    Amphibian = 6,
    [Description("коне")]
    Horse = 7,
    [Description("други")]
    Other = 8,
}
```

Фигура 7. Изброимият тип *PetTypeEnum*

Изброимият тип *PetTypeEnum* (Фигура 7.) съдържа всички типове домашни любимци, които платформата поддържа. Те са: котки, кучета, риби, малки бозайници, птици, влечуги, земноводни, коне и други.

vi. *RoleEnum*

```
public enum RoleEnum
{
    [Description("Потребител")]
    User = 0,
    [Description("Администратор")]
    Admin = 1
}
```

Фигура 8. Изброимият тип *RoleEnum*

Изброимият тип *RoleEnum* (Фигура 8.) съдържа всички роли, които платформата поддържа за оторизация – потребител и администратор.

vii. PetAgeEnum

```
public enum PetAgeEnum
{
    [Description("млад")]
    Young = 0,
    [Description("възрастен")]
    Adult = 1,
}
```

Фигура 9. Изброимият тип PetAgeEnum

Изброимият тип PetAgeEnum (Фигура 9.) съдържа двете възможности за възраст на животното, с които то е категоризирано – младо или възрастно. (Вж. [i.Pet](#))

viii. GenderEnum

```
public enum GenderEnum
{
    [Description("мъжки")]
    Male = 0,
    [Description("женски")]
    Female = 1,
    [Description("не се знае")]
    Other = 2,
}
```

Фигура 10. Изброимият тип GenderEnum

Този изброим тип данни (Фигура 10.) служи за правилното задаване на пола на всеки домашен любимец. Вариантите са: мъжки, женски или “не се знае”, поради невъзможно определяне на пола при новородени животни или други причини.

ix. SelectOption

```
public class SelectOption
{
    4 references | 1/1 passing
    public string Label { get; set; } = string.Empty;
    1 reference
    public string Value { get; set; }
    0 references
    public bool Selected { get; set; } = false;

    0 references
    public SelectOption() { }
    1 reference
    public SelectOption(string label, string value)
    {
        Label = label;
        Value = value;
    }
}
```

Фигура 11. Класа SelectOption

Класът SelectOption (Фигура 11.) служи за изграждането на опциите в множество избирателни списъци, които се използват в интернет страницата на приложението.

Той има следните полета:

1. Label, String – текстово поле за видимият етикет на опцията;
2. Value, String – текстово поле за стойността, която ще се подаде, ако опцията е избрана;
3. Selected, Bool – логическо поле за определяне, дали опцията е избрана по-подразбиране. Това става често при зареждане на стара информация, която вече потребителя е попълнил.

х. EnumExtensions

```
public static string ToDescriptionString<TEnum>(this TEnum e) where TEnum : IConvertible
{
    string description = "";

    if (e is Enum)
    {
        Type type = e.GetType();
        var memInfo = type.GetMember(type.GetEnumName(e.ToInt32(CultureInfo.InvariantCulture)));
        var soAttributes = memInfo[0].GetCustomAttributes(typeof(DescriptionAttribute), false);
        if (soAttributes.Length > 0)
        {
            // Достъпваме само първото поле за описание
            // Ще игнорираме всички гръзи
            description = ((DescriptionAttribute)soAttributes[0]).Description;
        }
    }

    return description;
}
```

Фигура 12. Метода ToDescriptionString в класа EnumExtensions

Класът EnumExtensions (Фигура 12.) цели да увеличи функционалностите на изброимия тип данни. Чрез метода ToDescriptionString(), който се съдържа в него, се достъпва атрибута за описание на всяка опция. По този начин лесно може да се използва българският превод на всяка опция във всеки дефиниран изброим тип в графическият потребителски интерфейс на платформата.

с) DbContext класовете и интерфейси

Класовете “DbContexts” отговарят за работата с данните в приложението и записите в базата данни. Всяка таблица си има направен отделен dbContext. В тези класове се реализират асинхронно функциите на CRUD (Create, Read, Update, Delete) – създай, прочети, актуализирай, изтрий. Асинхронното изпълнение на тези методи е от изключителна важност, позволяващо множество едновременни заявки от много потребители към базата данни. Това се гарантира чрез 2 интерфейса, които също ще бъдат разгледани в този раздел.

В тези класове има и помощни функции, които служат за извличането на специфични данни от базата данни чрез филтри или други критерии.

Бележка:

DbContext класовете могат и да ползват навигационни свойства при извличането на данни. Тези “свойства” определят дали да се зареди и информацията

във външните обекти, свързани с текущия. Чрез логически параметър се определя дали това допълнително зареждане ще се случи или обектите да си останат с нулева стойност.

xi. *IdbWithNav*

```
public interface IdbWithNav<T, K>
{
    // CRUD Operations
    31 references | 12/12 passing
    Task CreateAsync(T entity);

    16 references | 4/4 passing
    Task CreateAsync(List<T> entities);

    35 references | 8/8 passing
    Task<T>? ReadAsync(K id, bool useNavigationalProperties = false, bool isReadOnly = true);

    21 references | 7/7 passing
    Task<List<T>>? ReadAllAsync(bool useNavigationalProperties = false, bool isReadOnly = true);

    19 references | 6/6 passing
    Task UpdateAsync(T entity, bool useNavigationalProperties = false);

    16 references | 6/6 passing
    Task DeleteAsync(K id);
}
```

Фигура 13. Интерфейса *IdbWithNav*

Интерфейсът *IdbWithNav* (Фигура 13.) служи за определянето на стандарт за функционалностите на един *dbContext* клас, който има навигационни свойства и може да ги достъпи при извличане на данни.

Интерфейсът е шаблонен клас с два параметъра от неизвестен тип, които ще се инициализират в последствие. Те са:

1. T – типа данни на модела, с който ще се работи;
2. K – типа данни на първичния ключ на таблицата на модела. (В нашият случай това винаги ще е Guid)

Интерфейсът гарантира, че следните методи ще бъдат създадени:

1. *CreateAsync* (T entity), Task – запазване на обект T в базата данни;
2. *CreateAsync* (List<T> entities), Task – запазване на списък от обекти T в базата данни;
3. *ReadAsync* (K id, bool useNavigationalProperties = false, bool isReadOnly = true), Task<T> - извличане от базата данни запис с ключ K. Съществуват опциите да

се извличат и свързаните със записа навигационни свойства, както и дали данните да само за четене. Тези опции са обозначени с два логически параметъра;

4. `ReadAllAsync` (`bool useNavigationalProperties = false`, `bool isReadOnly = true`), `Task<List<T>>` - наподобява метода `ReadAsync`, но извличане всички записи от конкретна таблица в базата данни;
5. `UpdateAsync` (`T entity`, `bool useNavigationalProperties = false`), `Task` - актуализиране на данните на един запис. Има опция за актуализиране и на навигационните свойства, свързани със записа;
6. `DeleteAsync` (`K id`) – премахване на запис от таблицата според неговия първичен ключ `K`. В повечето `DbContext` класове имплементираното изтриване не е окончателно, а е “псевдо-изтриване”. (Вж. [бележката след а\) Модели](#))

xii. *IdbWithoutNav*

```
public interface IdbWithoutNav<T,K>
{
    // CRUD Operations
    7 references | 3/3 passing
    Task CreateAsync(T entity);

    6 references | 2/2 passing
    Task CreateAsync(List<T> entities);

    9 references | 3/3 passing
    Task<T>? ReadAsync(K id, bool isReadOnly = true);

    11 references | 3/3 passing
    Task<List<T>>? ReadAllAsync(bool isReadOnly = true);

    5 references | 2/2 passing
    Task UpdateAsync(T entity);

    6 references | 3/3 passing
    Task DeleteAsync(K id);
}
```

Фигура 14. Интерфейса *IdbWithoutNav*

Интерфейсът `IdbWithoutNav` (Фигура 14.) наподобява `IdbWithNav` с разликата, че функционалността да се извличат навигационните свойства не е зададена. Този интерфейс е направен за `DbContext` класа на модела `Town`, защото той няма външни ключове. (Вж. [xi. IdbWithNav](#))

xiii. *PetExchangeDbContext*

```
public class PetExchangeDbContext : IdentityDbContext<User, IdentityRole<Guid>, Guid>
{
    1 reference
    public PetExchangeDbContext()
    {
    }
    2 references
    public PetExchangeDbContext(DbContextOptions<PetExchangeDbContext> options) : base(options)
    {
    }
    0 references
    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        if (!optionsBuilder.IsConfigured)
        {
            optionsBuilder.UseSqlServer(ConnectionString.Value);
        }
    }
}
```

Фигура 15. Класа *PetExchangeDbContext*

Класът *PetExchangeDbContext* (Фигура 15.) служи за връзката с базата данни и определянето на таблиците, които платформата ще използва. Това се случва с дефиницията на 4 *DbSet* полета (съответно за 4-те модели). Низът за връзка с базата данни е изведен в отделен клас *ConnectionString* (Вж. [xiv. ConnectionString](#)).

Също така този клас съдържа и метод за попълване на примерни записи в базата данни, когато тя е празна. Тези записи ще се използват при представянето на дипломната работа, за да се покажат всички функционалности на проекта.

xiv. *ConnectionString*

```
public static class ConnectionString
{
    2 references
    public static string Value { get; } = "Server=localhost\\SQLEXPRESS;Database=PetExchange;";
}
```

Фигура 16. Класа *ConnectionString*

Низът за връзка с базата данни се отделя в друг клас (Фигура 16.) с оглед мерки на сигурност и лесна четимост. Това също позволява кооперативна работа на много програмисти, като всеки използва свой различен низ за връзка.

xv. *PetDbContext*

```
public class PetDbContext : IDbWithNav<Pet, Guid>
{
    private PetExchangeDbContext _dbContext;
    public PetDbContext(PetExchangeDbContext context)
    {
        _dbContext = context;
    }

    public async Task<List<Pet>> ReadAllWithFilterAsync(string name, string petBreed, string petType,
        string gender, string ownerName, int page, int pageSize, bool useNavigationalProperties = true,
        bool isReadOnly = true)
    {
        var allPets = await ReadAllAsync(useNavigationalProperties, isReadOnly);
        // прилагане на филтри
        var filteredPets = allPets.Where(x =>
            (String.IsNullOrEmpty(name) || x.Name.ToLower().Contains(name.ToLower()))
            && (String.IsNullOrEmpty(petBreed) || x.Breed.ToLower().Contains(petBreed.ToLower()))
            && (String.IsNullOrEmpty(petType) || x.PetType.ToDescriptionString().ToLower().Contains(petType.ToLower()))
            && (String.IsNullOrEmpty(gender) || x.Gender.ToDescriptionString().ToLower().Contains(gender.ToLower()))
            && (String.IsNullOrEmpty(ownerName) || x.User.Name.ToLower().Contains(ownerName.ToLower()))
        ).ToList();
        // странициране
        filteredPets = filteredPets.Skip((page - 1) * pageSize).Take(pageSize).ToList();
        return filteredPets;
    }
}
```

Фигура 17. Класа *PetDbContext*

Класът *PetDbContext* (Фигура 17.) служи за работа със записите на домашни любимци в платформата. Той имплементира интерфейса *IdbWithNav* и има няколко метода за филтрирано извличане на данни. Тези методи използват LINQ заявки за извличане на нужната информация.

Първият метод за извличане е *ReadAllWithFilterAsync* (Фигура 17.). Служи за филтриране на домашните любимци при работа с таблици в графичния потребителски интерфейс (Вж. [lvii. Таблица](#)). Приемайки няколко параметъра, обозначаващи различните критерии за филтриране (име, порода, тип на животното, пол, име на стопанина), както и параметри свързани със странирането, навигационните свойства и режима “само четене”, методът връща списък с обектите, които отговарят на критериите. Алгоритъмът първо извлича всички записи с метода *ReadAllAsync()* и след това чрез LINQ заявки проверя всеки критерий първо дали е зададен и после го прилага към списъка. Страницирането идва след това - според размера на страниците и номерът на текущата страница се взимат данните за текущата страница.


```
public async Task<List<Pet>> ReadAllWithFilterAsyncOfUser(Guid userId, string name, string petBreed,
string petType, string gender, int page, int pageSize, bool useNavigationalProperties = true, bool isReadOnly = true)
{
    var allPets = await ReadAllAsync(useNavigationalProperties, isReadOnly);
    // прилагане на филтри
    var filteredPets = allPets.Where(x =>
        (String.IsNullOrEmpty(name) || x.Name.ToLower().Contains(name.ToLower()))
        && (String.IsNullOrEmpty(petBreed) || x.Breed.ToLower().Contains(petBreed.ToLower()))
        && (String.IsNullOrEmpty(petType) || x.PetType.ToDescriptionString().ToLower().Contains(petType.ToLower()))
        && (String.IsNullOrEmpty(gender) || x.Gender.ToDescriptionString().ToLower().Contains(gender.ToLower()))
        && x.UserId == userId
        && x.IsActive == true
    ).ToList();
    // странициране
    filteredPets = filteredPets.Skip((page - 1) * pageSize).Take(pageSize).ToList();
    return filteredPets;
}
```

Фигура 18. Метода ReadAllWithFilterAsyncOfUser

Вторият метод за извличане е ReadAllWithFilterAsyncOfUser (Фигура 18.). Той е аналогичен с предишния, но добавя още 2 критерия – стопанин и активност (“видимост”) на записа. Той се използва за извличането само на домашни любимци, на които потребителят е стопанин и не са “псевдо-изтрити”.

```
public async Task<List<Pet>> ReadWithFiltersAsync(List<PetTypeEnum> types, List<GenderEnum> genders,
List<PetAgeEnum> ages,
bool? withCage, int page = 1, int pageSize = 8)
{
    IQueryable<Pet> query = _dbcontext.Pets;
    query = query.AsNoTrackingWithIdentityResolution();
    query = query.Include(e => e.User).ThenInclude(e => e.Town);
    query = query.Where(e => e.AdoptedOn == null && e.IsActive);
    if (types != null && types.Count > 0)
    {
        query = query.Where(e => types.Contains(e.PetType));
    }
    if (genders != null && genders.Count > 0)
    {
        query = query.Where(e => genders.Contains(e.Gender));
    }
    //Необходимо е само, ако егун филтър за възраст е избран
    if (ages != null && ages.Count == 1)
    {
        var adultDate = DateTime.Now.AddDays(-90);
        if (ages[0] == PetAgeEnum.Young)
        {

```

Фигура 19. Метода ReadWithFiltersAsync

Третият метод за извличане е ReadWithFiltersAsync (Фигура 19.). Той се използва в страница “Любимци” за извличане на всички активни неосиновени животни според множество филтри. Параметрите тук на филтрите се задават с листове от дефинираните изброими типове.

```
public async Task<List<Pet>> Read4NewestAsync()
{
    IQueryable<Pet> query = _dbcontext.Pets;

    query = query.AsNoTrackingWithIdentityResolution();
    query = query.Include(e => e.User).ThenInclude(e => e.Town);
    query = query.Where(e => e.AdoptedOn == null && e.IsActive);
    query = query.OrderByDescending(e => e.AddedOn);
    query = query.Take(4);
    return await query.ToListAsync();
}

public async Task<List<Pet>> Read4OldestAsync()
{
    IQueryable<Pet> query = _dbcontext.Pets;

    query = query.AsNoTrackingWithIdentityResolution();
    query = query.Include(e => e.User).ThenInclude(e => e.Town);
    query = query.Where(e => e.AdoptedOn == null && e.IsActive);
    query = query.OrderBy(e => e.AddedOn);
    query = query.Take(4);
    return await query.ToListAsync();
}
```

Фигура 20. Методите *Read4NewestAsync()* и *Read4OldestAsync()*

Следват още 2 метода, които са сходни помежду си – *Read4NewestAsync* и *Read4OldestAsync* (Фигура 20.). Те извличат от базата данни съответно 4-те най-скорошно регистрирани животни и 4-те животни, които чакат осиновяване от най-много време. Методите са използвани за извличане на животни, които ще се покажат на главната страница на платформата.

Следват методи, свързани с имплементацията на интерфейса за CRUD.

1. Запазване

```
public async Task CreateAsync(Pet entity)
{
    await _dbcontext.Pets.AddAsync(entity);
    await _dbcontext.SaveChangesAsync();
}

public async Task CreateAsync(List<Pet> pets)
{
    foreach (var pet in pets)
    {
        await CreateAsync(pet);
    }
}
```

Фигура 21. Методите за запазване в PetDbContext

Съществуват 2 метода за създаване на записи в базата данни (Фигура 21.) – CreateAsync(Pet entity) и CreateAsync(List<Pet> pets). Вторият извиква първият за всеки елемент в списъка с обекти, който получава като параметър.

Запазването се случва, като се достъпи DbSet Pets от инстанция на PetExchangeDbContext (Виж [xii. PetExchangeDbContext](#)), добавя се елемент към него и се запазят промените.

2. Четене

```
public async Task<Pet>? ReadAsync(Guid id, bool useNavigationalProperties = true, bool isReadOnly = true)
{
    IQueryable<Pet> query = _dbcontext.Pets;
    if (isReadOnly)
    {
        query = query.AsNoTrackingWithIdentityResolution();
    }
    if (useNavigationalProperties)
    {
        query = query.Include(e => e.User).ThenInclude(e => e.Town);
    }
    return await query.SingleOrDefaultAsync(e => e.Id == id);
}

public async Task<List<Pet>> ReadAllAsync(bool useNavigationalProperties = true, bool isReadOnly = true)
{
    IQueryable<Pet> query = _dbcontext.Pets;

    if (isReadOnly)
    {
        query = query.AsNoTrackingWithIdentityResolution();
    }
    if (useNavigationalProperties)
    {

```

Фигура 22. Методите за четене в PetDbContext

Методите за четене `ReadAsync` и `ReadAllAsync` (Фигура 22.) съдържат параметри за навигационни свойства и за режим “само четене”.

Първата стъпка на алгоритъма е достъпване на пълния списък с животни чрез `DbSet Pets`. Втората е, ако е зададено, да се настройват данните да не се следят за промени от `EntityFramework`, поставяйки ги в режим “само четене”. Това става чрез метода `AsNoTrackingWithIdentityResolution()`.

Третата стъпка е зареждането на навигационните свойства, отново при зададена логическа стойност “истина“ на параметъра `useNavigationalProperties`. Това става чрез множество LINQ заявки (`Include` и `ThenInclude`). В зависимост от това колко навигационни свойства съществуват тези заявки може да бъдат повторени няколко пъти.

Накрая в случая на `ReadAsync()` се връща елемента от списъка с избран първичен ключ или в случая на `ReadAllAsync()` – целият списък.

3. Актуализация

```
public async Task UpdateAsync(Pet pet, bool useNavigationalProperties = true)
{
    Pet petFromDb = await ReadAsync(pet.Id, useNavigationalProperties, false);
    if (petFromDb is null)
    {
        throw new ArgumentException("Pet with id = " + pet.Id + "does not exist!");
    }
    if (useNavigationalProperties) _dbContext.Pets.Update(pet); // актуализира всички навигационни свойства
    else
    {
        _dbContext.Pets.Entry(petFromDb).CurrentValues.SetValues(pet); // актуализира само текущият запис
    }
    await _dbContext.SaveChangesAsync();
}
```

Фигура 23. Метода за актуализация в `PetDbContext`

Методът за актуализация `UpdateAsync` (Фигура 23.) използва метода `ReadAsync`, за да достъпи нужният запис от базата данни. След това, ако се използват навигационни свойства, се актуализират тези записи чрез метода `Update()`. Иначе, се актуализира само достопеният запис.

4. Изтриване

```
public async Task DeleteAsync(Guid key)
{
    Pet pet = await ReadAsync(key, false, false);

    if (pet is null)
    {
        throw new ArgumentException("Pet with id = " + key + " does not exist!");
    }

    pet.IsActive = false;
    await _dbcontext.SaveChangesAsync();
}
```

Фигура 24. Метода за изтриване в *PetDbContext*

Методът за изтриване `DeleteAsync` (Фигура 24.) “псевдо-изтрива” запис по зададен първичен ключ, като първо го достъпва чрез `ReadAsync`.

xvi. *TownDbContext*

```
public class TownDbContext : IDbWithoutNav<Town, Guid>
{
    private readonly PetExchangeDbContext _dbcontext;
    3 references
    public TownDbContext(PetExchangeDbContext context)
    {
        _dbcontext = context;
    }

    5 references | 2/2 passing
    public async Task CreateAsync(Town entity)
    {
        if (_dbcontext.Towns.Count(x => x.Name == entity.Name) > 0)
        {
            throw new DuplicateNameException("There is already a town with the same name.");
        }
        await _dbcontext.Towns.AddAsync(entity);
        await _dbcontext.SaveChangesAsync();
    }

    4 references | 1/1 passing
    public async Task CreateAsync(List<Town> towns)
    {
        foreach (var town in towns)
        {
            await CreateAsync(town);
        }
    }

    #region CRUD
    6 references | 2/2 passing
    public async Task<Town>? ReadAsync(Guid id, bool isReadOnly = true)
    {
        IQueryable<Town> query = _dbcontext.Towns;
```

Фигура 25. Класа *TownDbContext*

Класът TownDbContext (Фигура 25.) наследява IDbWithoutNav, защото класа Town няма външни ключове. По аналогичен начин както PetDbContext, имплементира CRUD, като пропуска функционалностите за навигационните свойства. (Виж [xv. PetDbContext](#))

```
public async Task DeleteAsync(Guid key)
{
    Town town = await ReadAsync(key, false);

    if (town is null)
    {
        throw new ArgumentException("Town with id = " + key + " does not exist!");
    }

    _dbContext.Towns.Remove(town);
    await _dbContext.SaveChangesAsync();
}
```

Фигура 26. Метода за изтриване в TownDbContext

Единствената разлика е, че при метода за изтриване (Фигура 26.) окончателно се изтрива записът от базата данни - липсва функция “псевдо-изтриване”.

xvii. UserDbContext

```
public class UserDbContext : IDbWithNav<User, Guid>
{
    private readonly PetExchangeDbContext _dbContext;
    private readonly UserManager<User> userManager;
    private readonly RoleEnum adminRole = RoleEnum.Admin;
    private readonly RoleEnum userRole = RoleEnum.User;

    public UserDbContext(PetExchangeDbContext petExchangeDbContext, UserManager<User> userManager)
    {
        _dbContext = petExchangeDbContext;
        this.userManager = userManager;
    }

    public UserDbContext(PetExchangeDbContext petExchangeDbContext)
    {
        _dbContext = petExchangeDbContext;
    }

    public async Task ChangePassWord(User entity, string newPassWord)
    {
        var userFromDb = await userManager.FindByNameAsync(entity.UserName);
        var token = await userManager.GeneratePasswordResetTokenAsync(userFromDb);
        var result = await userManager.ResetPasswordAsync(userFromDb, token, newPassWord);
        await _dbContext.SaveChangesAsync();
    }
}
```

Фигура 27. Класа UserDbContext

Класът UserDbContext (Фигура 27.) имплементира интерфейса IDbWithNav и аналогично като PetDbContext изгражда методи за филтрирано извличане на записи и CRUD. (Виж [xv. PetDbContext](#))

Разликите между двата класа са следните:

1. UserDbContext имплементира метода ChangePassWord (Фигура 27.), който, използвайки вграденият клас userManager<User>, променя паролата на потребител с нова, която отново е хеширана и само нейният хеш е запазен в базата данни.
2. User моделът разчита на задаването на роли за оторизация. Те се слагат и премахват чрез методите на класа userManager - AddToRoleAsync и RemoveFromRoleAsync. (Фигура 28.)

```
if (user.Role == adminRole && user.Role != userFromDb.Role)
{
    await userManager.RemoveFromRoleAsync(user, userRole.ToString());
    await userManager.AddToRoleAsync(user, adminRole.ToString());
}
else if (user.Role == userRole && user.Role != userFromDb.Role)
{
    await userManager.RemoveFromRoleAsync(user, adminRole.ToString());
    await userManager.AddToRoleAsync(user, userRole.ToString());
}
await _dbContext.SaveChangesAsync();
```

Фигура 28. Добавяне и премахване на роли с UserManager при метода за актуализация на UserDbContext

xviii. UserRequestsDbContext

```
public class UserRequestsDbContext : IDbWithNav<UserRequest, Guid>
{
    private readonly PetExchangeDbContext _dbContext;

    public UserRequestsDbContext(PetExchangeDbContext context)
    {
        _dbContext = context;
    }

    public async Task<List<UserRequest>> ReadAllWithFilterAsync(string petName, string petBreed,
        string senderName, string receiverName,
        int page, int pageSize, bool useNavigationalProperties = true, bool isReadOnly = true)
    {
        var allRequests = await ReadAllAsync(useNavigationalProperties, isReadOnly);
        // filtering
        var filteredRequests = allRequests.Where(x =>
            (String.IsNullOrEmpty(petName) || x.Pet.Name.ToLower().Contains(petName.ToLower()))
            && (String.IsNullOrEmpty(petBreed) || x.Pet.Breed.ToLower().Contains(petBreed.ToLower()))
            && (String.IsNullOrEmpty(senderName) || x.Sender.Name.ToLower().Contains(senderName.ToLower()))
            && (String.IsNullOrEmpty(receiverName) || x.Recipient.Name.ToLower().Contains(receiverName.ToLower()))
        ).ToList();
        // paging
        filteredRequests = filteredRequests.Skip((page - 1) * pageSize).Take(pageSize).ToList();
        return filteredRequests;
    }
}
```

Фигура 29. Класа UserRequestsDbContext

Класът `UserRequestsDbContext` (Фигура 29.) имплементира интерфейса `IdbWithNav` и аналогично като `PetDbContext` изгражда методи за филтрирано извличане на записи и CRUD. (Виж [xv. PetDbContext](#))

Класът съдържа и още няколко специализирани методи за работа с искания:

1. Извличане на входящите и изходящите искания за даден потребител

```
public async Task<List<UserRequest>> ReadUserRequestOutboxAsync(Guid userId)
{
    IQueryable<UserRequest> query = _dbContext.Requests;

    query = query.AsNoTrackingWithIdentityResolution();
    query = query.Include(e => e.Pet);
    query = query.Include(e => e.Recipient).ThenInclude(r => r.Town);
    query = query.Include(e => e.Sender);
    query = query.Where(e => e.SenderId == userId).OrderByDescending(x => x.CreatedOn);
    return await query.ToListAsync();
}

public async Task<List<UserRequest>> ReadUserRequestInboxAsync(Guid userId)
{
    IQueryable<UserRequest> query = _dbContext.Requests;

    query = query.AsNoTrackingWithIdentityResolution();
    query = query.Include(e => e.Pet);
    query = query.Include(e => e.Recipient).ThenInclude(r => r.Town);
    query = query.Include(e => e.Sender);
    query = query.Where(e => e.RecipientId == userId).OrderByDescending(x => x.CreatedOn);
    return await query.ToListAsync();
}
```

Фигура 30. Методите за Outbox и Inbox в `UserRequestsDbContext`

Методите `ReadUserRequestOutboxAsync` и `ReadUserRequestInboxAsync` (Фигура 30.) извличат всички записи за изходящи или входящи искания на даден потребител. Филтрирането се извършва с помощта на LINQ заявки, които се прилагат към пълният списък искания от базата данни

2. Методи за прием/отказ/отхвърляне на искане

```
public async Task CancelAsync(Guid requestId)
{
    UserRequest requestFromDb = await ReadAsync(requestId, false, false);

    if (requestFromDb is null)
    {
        throw new ArgumentException("User request with id = " + requestId + "does not exist!");
    }
    requestFromDb.CanceledOn = DateTime.Now;

    await _dbContext.SaveChangesAsync();
}
```

Фигура 31. Метода `CancelAsync` на класа `UserRequestsDbContext`

Методът `CancelAsync` (Фигура 31.) служи за отхвърляне на искане към домашен любимец от потребителя, който първоначално го е направил. Това се случва с извличане от на запис чрез първичния му ключ, задаване на дата на отхвърляне и запазване на промените.

```
public async Task AcceptAsync(Guid requestId, string? message)
{
    UserRequest requestFromDb = await ReadAsync(requestId, true, false);

    if (requestFromDb is null)
    {
        throw new ArgumentException("User request with id = " + requestId + "does not exist!");
    }
    requestFromDb.AcceptedOn = DateTime.Now;
    requestFromDb.AnswerMessage = message;
    requestFromDb.Pet.AdoptedOn = DateTime.Now;

    await _dbContext.SaveChangesAsync();
}
```

Фигура 32. Метода `AcceptAsync` на класа `UserRequestsDbContext`

Методът `AcceptAsync` (Фигура 32.) служи за приемане на искане към домашен любимец от потребителя получател. Това се случва с извличане на запис чрез първичния му ключ, задаване на дата на приемане, както и съобщение за отговор и дата на осиновяване, и запазване на промените.

```
public async Task DenyAsync(Guid requestId, string? message)
{
    UserRequest requestFromDb = await ReadAsync(requestId, true, false);

    if (requestFromDb is null)
    {
        throw new ArgumentException("User request with id = " + requestId + "does not exist!");
    }
    requestFromDb.DeniedOn = DateTime.Now;
    requestFromDb.AnswerMessage = message;

    await _dbContext.SaveChangesAsync();
}
```

Фигура 33. Метода `DenyAsync` на класа `UserRequestsDbContext`

Методът `DenyAsync` (Фигура 33.) служи за отказ на искане към домашен любимец от потребителя получател. Това се случва с извличане на запис чрез първичния му ключ, задаване на дата на отказване, както и съобщение за отговор, и запазване на промените.

5.4.2 Business Layer

Бизнес слойът е основна част от архитектурата на приложението, която осигурява връзката между слоя за данни (Data Layer) и презентационния слой (Presentation Layer). Неговата роля е:

1. Да осигурява абстракция, като скрива детайлите на достъпа до база данни от потребителския интерфейс;
2. Да следва структурата на методите, дефинирани в DbContext класовете, като извиква методите, създадени в слоя за данни.

Тази архитектура осигурява модулност, лесна поддръжка и повторно използване на кода. Слойът съдържа четири класа – един за всеки дефиниран DbContext клас.

xix. *PetService*

```
public class PetService : IDbWithNav<Pet, Guid>
{
    public PetDbContext _PetContext;

    public PetService(PetExchangeDbContext _ProjectContext)
    {
        _PetContext = new PetDbContext(_ProjectContext);
    }

    public async Task<List<Pet>> ReadAllWithFilterAsync(string name, string petBreed,
        string petType, string gender, string ownerName, int page, int pageSize,
        bool useNavigationalProperties = true, bool isReadOnly = true)
    {
        return await _PetContext.ReadAllWithFilterAsync(name, petBreed, petType,
            gender, ownerName, page, pageSize, useNavigationalProperties, isReadOnly);
    }

    public async Task<List<Pet>> ReadAllWithFilterAsyncOfUser(Guid userId, string name,
        string petBreed, string petType, string gender, int page, int pageSize,
        bool useNavigationalProperties = true, bool isReadOnly = true)
    {
        return await _PetContext.ReadAllWithFilterAsyncOfUser(userId, name,
            petBreed, petType, gender, page, pageSize, useNavigationalProperties, isReadOnly);
    }
}

#region CRUD
```

Фигура 34. Класа *PetService*

Класът *PetService* (Фигура 34.) съдържа методи-дубликати на тези в *PetDbContext* и ги извиква. (Виж [xv. PetDbContext](#))

xx. *TownService*

```
public class TownService : IDbWithoutNav<Town, Guid>
{
    public TownDbContext _TownContext;

    public TownService(PetExchangeDbContext _ProjectContext)
    {
        _TownContext = new TownDbContext(_ProjectContext);
    }

    public async Task<List<SelectOption>> GetTownOptions()
    {
        var result = new List<SelectOption>();
        foreach (var town in await ReadAllAsync())
        {
            result.Add(new SelectOption(label: town.Name, value: town.Id.ToString()));
        }
        result = result.OrderBy(x => x.Label).ToList();
        return result;
    }
}
```

CRUD

Фигура 35. Класа *TownService*

Класът *TownService* (Фигура 35.) съдържа методи-дубликати на тези в *TownDbContext* и ги извиква. (Вж. [xvi. TownDbContext](#))

Също така съдържа и метод *GetTownOptions* (Фигура 35.), който извлича всички регистрирани до момента градове и ги предава под формата на списък от тип *SelectOption*. Този списък след това се използва в регистрационната форма на платформата. (Вж. [ix. SelectOption](#))

xxi. *UserService*

```
public class UserService : IDbWithNav<User, Guid>
{
    public UserDbContext _UserContext;
    public UserService(PetExchangeDbContext _ProjectContext, UserManager<User> userManager)
    {
        _UserContext = new UserDbContext(_ProjectContext, userManager);
    }
    public UserService(PetExchangeDbContext _ProjectContext)
    {
        _UserContext = new UserDbContext(_ProjectContext);
    }

    public async Task<List<User>> ReadAllWithFilterAsync(string username, string name,
string email, string town, string role, int page = 1, int pageSize = 10,
bool useNavigationalProperties = true, bool isReadOnly = true)...
```

CRUD

Фигура 36. Класа *UserService*

Класът UserService (Фигура 36.) съдържа методи-дубликати на тези в UserDbContext и ги извиква. (Виж [xvii. UserDbContext](#))

xxii. UserRequestsService

```
public class UserRequestsService : IDbWithNav<UserRequest, Guid>
{
    public UserRequestsDbContext _UserRequestsContext;
    public UserRequestsService(PetExchangeDbContext _ProjectContext)
    {
        _UserRequestsContext = new UserRequestsDbContext(_ProjectContext);
    }

    public async Task<List<UserRequest>> ReadAllWithFilterAsync(string petName, string petBreed,
        string senderName, string receiverName, int page, int pageSize,
        bool useNavigationalProperties = true, bool isReadOnly = true)
    {
        return await _UserRequestsContext.ReadAllWithFilterAsync(petName, petBreed,
            senderName, receiverName, page, pageSize, useNavigationalProperties, isReadOnly);
    }
}
```

Фигура 37. Класа UserRequestsService

Класът UserRequestsService (Фигура 37.) съдържа методи-дубликати на тези в UserRequestsDbContext и ги извиква. (Виж [xviii. UserRequestsDbContext](#))

5.4.3 ASP.NET MVC (Presentation Layer)

Този слой отговаря за графичния интерфейс на платформата и начина, по който потребителите взаимодействат с приложението. Той е изграден с технологията ASP.NET MVC, която следва Model-View-Controller (MVC) архитектурния шаблон.

MVC разделя слоя на три основни компонента:

1. Модели (Models) – представят данните и бизнес логиката на приложението;
2. Изгледи (Views) – визуализират информацията и оформят потребителския интерфейс;
3. Контролери (Controllers) – управляват заявките на потребителя, обработват входните данни и координират взаимодействието между моделите и изгледите.

Тази структура улеснява поддръжката, подобрява организацията на кода и позволява по-ясно разделение на отговорностите.

В този раздел ще разгледаме как тези компоненти работят заедно.

Бележка:

Повечето документация на този раздел ще се съдържа в кода на проекта, поради мерки за намаляване на дължината на документационния файл.

а) Модели

xxiii. ChangePasswordModel

ChangePasswordModel е клас, който служи за дефиниране и валидация на полетата, които ще се показват за попълване при променянето на парола в графичния интерфейс. Съдържа две текстови полета за парола и потвърждение на парола.

xxiv. PetManage

PetManage е клас, който служи за дефиниране и валидация на полетата, които ще се показват за попълване при редакция или създаване на домашен любимец.

xxv. UserManage

UserManage е клас, който служи за дефиниране и валидация на полетата, които ще се показват за попълване при редакция или създаване на потребителски профил.

xxvi. UserRequestAction

UserRequestAction е клас, който служи за дефиниране и валидация на полетата, които ще се показват за изпращане на искане към домашен любимец

xxvii. Filter

Filter е клас, определящ единна структура за представянето на филтрите в графичния потребителски интерфейс.

xxviii. FilterSelection

FilterSelection е клас, определящ сложната структура на филтри, които се ползват в главната страница за домашни любимци.

xxix. MenuItem

MenuItem е клас, дефиниращ характеристиките на навигационно меню.

xxx. *MenuItemType*

MenuItemType е изброим тип, която дефинира типа на менюто, което от своя страна влияе на начина, по който менюто е изобразено на екрана.

xxxi. *FileService*

FileService е помощен клас, който служи за създаването и изтриването на файлове (изображения), използвани в платформата.

xxxii. *AppErrorDescriber*

AppErrorDescriber е помощен клас, който служи за превеждането на грешки при валидацията, които могат да излязат на графичния интерфейс, на български език.

xxxiii. *FilterUtility*

FilterUtility е помощен клас с методи за инициализация на филтри от URL.

xxxiv. *ViewUtility*

ViewUtility е помощен клас с методи, нужни за презентация.

б) Изгледи

Повечето изгледи са изградени върху компоненти (Razor Components), които са отделни “модули” HTML, CSS и логически операции, събрани заедно. Това позволява широка използваемост на един и същ код на много различни места и лесна поддръжка на кода.

Графическият потребителски интерфейс (“ГПТ”) поддържа следните изгледи:

xxxv. *Вход*

xxxvi. *Регистрация*

xxxvii. *Начална страница*

xxxviii. *Всички домашни любимци (Изгледа “Любимци”)*

xxxix. *Редактиране на потребителски профил*

xl. *Смяна на потребителска парола*

xli. *Преглед на моите домашни любимци*

xl.ii. *Редактиране на моите домашни любимци*

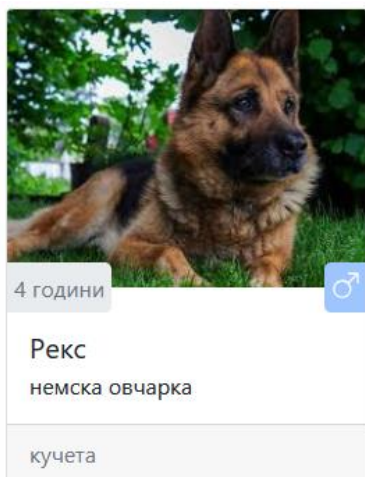
- xliv. Изпращане на искане*
- xliv. Преглед и администрация на моите искания*
- xlvi. Преглед и администрация на получените искания*
- xlvi. “За нас”*
- xlvi. Изглед за контакти с администратори*
- xlvi. Политиката и условията за ползване*
- xlvi. Поверителност на данните*
 - i. Страница за грешка 404*
 - ii. Страница за грешка 500*

Административни изгледи

- lii. Редактиране на всички домашни любимци*
- liii. Редактиране на всички потребители*
- liv. Преглед на всички изпратени искания*

В ГПТ се използват следните главни функционалности:


- lvi. Компонентът PetCard*



Фигура 38. Пример за PetCard

Компонентът PetCard (Фигура 38.) съдържа цялата информация за един домашен любимец. При натискане върху него се показва модал, който служи за подробно показване на информацията и съдържа бутон за изпращане на искане. (Фигура 39.)

Любимец
×






Име	Рекс
Порода	немска овчарка
Възраст	4 години, 0 месеца, 6 дни
Пол	мъжки
Описание	Рекс е смел и лоялен пазач, който винаги защитава своето семейство. Обича дългите разходки и тренировките. Отличава се с интелигентност и лесно се поддава на обучение.
Клетка	без клетка
Собственик	Георги Иванов
Град	Пловдив
Обявен на	31.1.2025 г.

Изпрати искане
Затвори

Фигура 39. Модалът на PetCard

lvii. Таблица

	снимка	име	добавен на	осиновен на	порода	тип животно	пол
<input type="text" value=""/>		<input type="text" value=""/>			<input type="text" value=""/>	всички ▾	всички ▾
<input type="text" value=""/>		Голди	13.3.2025 г.	не е осиновен	златна рибка	риби	♀ Женски
<input type="text" value=""/>		Майлс	20.2.2025 г.	не е осиновен	джербил	малки бозайници	♂ Мъжки
<input type="text" value=""/>		Тропчо	22.11.2024 г.	не е осиновен	джербил	малки бозайници	♂ Мъжки
Назад	Напред						

Фигура 40. Пример за таблица

Таблицата (Фигура 40.) се среща многократно в платформата за визуализиране на налични домашни любимци, потребители или искания. На вторият ред винаги съдържа наличните възможности за филтри под формата на текстови полета или SelectOption

елементи. След това почват записите, като първата колона е резервирана за функциите “редактирай”, “преглед” или “изтрий” (не всички функции са поддържани за всяка таблица и за всяка роля).

Накрая на таблицата има опции за странициране.

За стиловете и иконите е използвана библиотеката за стилове и икони Bootstrap.

lviii. Layout

За да се поддържа еднаквост на стила и наличието на header и footer, независимо от избрания изглед, се използват оформлени (Layouts). Съществуват 3 оформления:

1. LayoutSite

```
<!DOCTYPE html>
<html lang="bg">
<head>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>@ViewData["Title"] - PetExchange</title>
  <link rel="icon" href="~/favicon.ico" type="image/x-icon">
  <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.min.css" />
  <link rel="stylesheet" href="~/lib/bootstrap-icons/bootstrap-icons.css">
  <link rel="stylesheet" href="~/css/site.css" asp-append-version="true" />
  <link rel="stylesheet" href="~/WebPresentationLayer.styles.css" asp-append-version="true" />
</head>
<body>
  @await Component.InvokeAsync("Header")
  @await Component.InvokeAsync("Hero")
  <main role="main" class="pb-3">
    @RenderBody()
  </main>
  @await Component.InvokeAsync("Footer")
  <script src="~/lib/bootstrap/dist/js/bootstrap.bundle.min.js"></script>
  <script src="~/js/site.js" asp-append-version="true"></script>
  @await RenderSectionAsync("Scripts", required: false)
</body>
</html>
```

Фигура 41. Оформлението LayoutSite

LayoutSite е мастър оформление (Фигура 41.), което другите оформлени извикват. Чрез извикването на Component.InvokeAsync(<име на компонента>) се зареждат компонентите на конкретните места в кода. В нашия случай зареждаме компонента за Header, Hero⁷ и Footer.

HTML кодът, който ще наследи това оформление, ще се визуализира на мястото на метода @RenderBody ().

⁷ Компонент, който или зарежда снимка на заглавната страница или прави “breadcrumb” пътечка с пътя на текущата страница. Тази пътечка може да се наблюдава във всяка страница освен началната.

2. LayoutAccount

```
@{
    Layout = "/Views/Shared/_LayoutSite.cshtml";
}
<div class="container pt-3">
    <div class="row">
        <aside class="sidebar col-12 col-md-5 col-lg-4 col-xl-3 col-xxl-2">
            @await Component.InvokeAsync("UserNav")
        </aside>
        <section class="content col-12 col-md-7 col-lg-8 col-xl-9 col-xxl-10">
            @RenderBody()
        </section>
    </div>
</div>
```

Фигура 42. Оформлението *LayoutAccount*

LayoutAccount (Фигура 42.) е оформлението за потребителски достъп, което се появява, когато потребителят е бил вписан в профила.

Чрез задаване оформление чрез полето Layout = "...", LayoutAccount използва оформлението LayoutSite. По този начин, подобно на куклите матрьошки, няколко оформления могат да използват сами себе си (с отсъствието на порочен кръг, разбира се).

Това оформление извиква компонента UserNav, което е менюто отстрани, съдържащо потребителските функционалности на системата.

3. LayoutAdmin

Оформление за администраторски достъп, аналогично на LayoutAccount с разликата, че извиква компонента AdminNav, който съдържа администраторските функционалности, вместо UserNav.

с) Контролери

Функционалността на платформата е поделена на няколко контролера.

lx. AccountController

AccountController отговаря за всички изгледи, които потребителя може да достъпи.

lxi. AdminController

AdminController отговаря за всички изгледи, които администратора може да достъпи.

lxii. PetsController

PetsController отговаря за изгледа “Любимци”.

lxiii. SiteController

SiteController отговаря за 4-те изгледа, които съдържат допълнителна информация за платформата.

lxiv. HomeController

HomeController отговаря за началната страница и пренасочването към страниците за грешките.

d) Program.cs

```
public static async Task Main(string[] args)
{
    var builder = WebApplication.CreateBuilder(args);

    // Инжектиране на сървиси в приложението
    var connectionString = ConnectionString.Value;
    builder.Services.AddDbContext<PetExchangeDbContext>(options =>
    {
        options.UseSqlServer(connectionString);
        options.EnableSensitiveDataLogging();
    });

    builder.Services.AddIdentity<User, IdentityRole<Guid>>()
        .AddRoles<IdentityRole<Guid>>()
        .AddEntityFrameworkStores<PetExchangeDbContext>()
        .AddDefaultTokenProviders()
        .AddErrorDescriber<AppErrorDescriber>();

    builder.Services.AddHttpContextAccessor();
    builder.Services.AddRazorPages();
    builder.Services.AddControllersWithViews();
}
```

Фигура 45. Класа Program.cs

Program.cs (Фигура 45.) е класът, който се извиква при стартиране на ASP.NET MVC проекта. В него се декларира DbContext класа, който ще се използва в платформата, инжектират се нужните методи⁸ под формата на “services” и се настройват още допълнителни настройки като изисквания за паролите и максимална големина на файловете с изображения.

5.4.4 Test Layer (Компонентно тестване)

Тестването на платформата е извършено чрез сто теста, използвайки технологията за тестване NUnit. Автоматичното тестване в този слой покрива повече от деветдесет процента от DataLayer и BusinessLayer. Повечето функционалности на презентационният слой са тествани ръчно, но автоматичното тестване покрива петнадесет процента от кода на този слой.

lxv. Пример за тестов мениджър

```
public partial class DataLayerTestsManagement
{
    public static PetExchangeDbContext db;
    public static PetDbContext petContext;
    public static TownDbContext townContext;
    public static UserDbContext userContext;
    public static UserRequestsDbContext userRequestsContext;

    1 reference
    private static PetExchangeDbContext GetMemoryContext()
    {
        var options = new DbContextOptionsBuilder<PetExchangeDbContext>()
            .UseInMemoryDatabase(databaseName: "InMemoryDatabase")
            .Options;
        return new PetExchangeDbContext(options);
    }

    1 reference
    protected static void DeleteAllEntriesInDb()
    {
        foreach (var entity in db.Pets)
        {
            db.Pets.Remove(entity);
        }
    }
}
```

Фигура 46. Пример за тестов мениджър

⁸ При инжектиране (Dependency Injection) на класове в Program.cs те стават достъпни за всички други класове в MVC слоя.

Платформата се тества с помощта на тестов мениджър клас⁹ (Фигура 46.), който се компилира и извършва преди всеки тест. Служи за зареждането на тестовата база данни (в паметта) и други методи за приготвяне.

lxvi. *Пример за тест*

```
public class PetDbContextTests : DataLayerTestsManagement
{
    [Test]
    public async Task CreateAsync_SinglePet_Succeeds()
    {
        // Arrange
        var user = await GetExampleUser(true);

        var pet = new Pet
        {
            Name = "Buddy",
            Breed = "Golden Retriever",
            PetTypeEnum = PetTypeEnum.Dog,
            Gender = GenderEnum.Male,
            UserId = user.Id,
            IsActive = true
        };

        // Act
        await petContext.CreateAsync(pet);

        // Assert
        var createdPet = db.Pets.FirstOrDefault(p => p.Name == "Buddy");
        Assert.NotNull(createdPet);
    }
}
```

Фигура 47. Пример за компонентен тест

Един компонентен тест (Фигура 47.) се състои от 3 части:

1. Arrange – подготвяне на данните за тестване;
2. Act – извикване на метода, който е тестван;
3. Assert – проверка на резултата.

По този начин се осигурява, че този метод (в изолация) работи правилно.

⁹ Съществува мениджър за всеки слой и класът се наследява от класовете с компонентните тестове.

5.4.5 Осигуряване на сигурност

За да гарантира защитата на данните и предотвратяването на злонамерени атаки, платформата прилага цялостен набор от мерки за сигурност.

Системата внедрява следните механизми за осигуряване на защита:

1. **Сигурност на данните** – Защитата на данните се осигурява чрез криптиране на чувствителна информация, съхранявана в базата данни;
2. **Валидация на данни** – Всички входни данни се валидират и филтрират с цел предотвратяване на SQL инжекции, XSS и други атаки, използващи злонамерени входове;
3. **Сигурност на контролерите и извеждането на данни** – Защитата на контролерите се осигурява чрез механизми за автентикация, ограничаване на достъпа, защита срещу открадване на токени за сесия;
4. **Потребителски роли** – Платформата прилага ролево-базирана система за контрол на достъпа, като осигурява различни нива на права в зависимост от ролята на потребителя;
1. **Хеширане на пароли** – Потребителските пароли се съхраняват в хеширан вид, като се използват надеждни алгоритми, предоставени от технологията Identity.

6. Авторски права

Настоящата дипломна работа и свързаният с нея софтуерен код са защитени с авторски права Виктор Зашев ©, 2025 . Всички права запазени.

Библиотеките използвани в проекта са или “отворен код” или се разпространяват с MIT лиценз. [4] [12] [18] [19]

III. Заключение

В настоящия дипломен проект беше разработена онлайн платформа за доброволна размяна на домашни любимци в България. Проектът постигна основната си цел – да създаде интуитивна и ефективна система, която улеснява процеса на осиновяване и намиране на подходящ дом за животните. Чрез внимателно проектиране и използване на съвременни технологии бе изградена сигурна и достъпна среда, която позволява на потребителите да публикуват обяви, да намират животни и да осъществяват контакт с други стопани.

Предварително проучване беше проведено, за да се преценят функционалностите, които платформата трябва да притежава и да се избегнат недостатъците на други подобни платформи, съществуващи на пазара.

Успешно беше реализирана многослойна архитектура, която играе важна роля за функционалността и стабилността на платформата:

- **DataLayer** – Отговаря за управлението и съхранението на данните в базата. Този слой гарантира ефективно обработване на заявки и сигурност на информацията, включително профилите на потребителите и обявите за животни;
- **BusinessLayer** – Осигурява посредничество между слоят за данните и потребителския интерфейс;
- **ASP.NET MVC** – Презентационният слой, който осигурява интерфейс за взаимодействие с потребителите. Благодарение на този слой системата предлага адаптивен дизайн, позволяващ достъпност както от настолни компютри, така и от мобилни устройства;
- **TestLayer** – Проведени бяха тестове за правилно функциониране на всички компоненти от кода. Тестването включваше цялостно тестване на функционалността на CRUD операциите, както и правилното маршрутизиране чрез контролерите на приложението.

По време на разработката се сблъсках с някои предизвикателства - имплементацията на търсенето по различни критерии, управлението на профилите, съхранението на изображения и проблеми при изграждането на адаптивния дизайн. Чрез

прилагане на добри практики в програмирането и много упоритост от моя страна, тези предизвикателства бяха успешно преодолені.

Въпреки, че проектът изпълнява своята основна цел, съществуват възможности за разширяване и подобрене на платформата. Някои от ключовите предложения за бъдещо развитие включват:

- **Интеграция на чат функционалност** – Възможност за директна комуникация между потребителите в рамките на платформата, без необходимост от външни социални мрежи;
- **По-точно географско филтриране** – Добавяне на GPS координати или по-прецизна локализация за по-добро съвпадение между потребителите;
- **Разширяване на административния контрол** – Разработване на по-детайлни инструменти за управление и модериране на съдържанието от администраторите;
- **Мобилно приложение** – Създаване на специализирано мобилно приложение, което ще подобри достъпността и удобството при използване на услугата;
- **Интеграция с ветеринарни услуги** – Добавяне на възможност за проверка на здравословното състояние на животните и консултации с ветеринарни специалисти в платформата.

Разработената система предоставя сигурен, удобен и ефективен начин за доброволна размяна на домашни любимци. Внедрените технологии и архитектурни решения позволяват гъвкавост и бъдещо мащабиране. Платформата има потенциал да бъде успешно разширена и адаптирана към нуждите на потребителите, като допринася за намаляване на изоставянето на животни и за подобряване на процеса на осиновяване.

Настоящият проект не само осигурява практично решение за осиновяване на животни, но също така насърчава отговорното отглеждане и предоставя надежден метод за намиране на подходящи стопани. С бъдещи подобрения и разширяване на функционалността, системата може да стане водеща платформа в България за доброволна размяна на домашни любимци.

IV. Списък на използвана литература

- [1] М. Стоянова, „Осинови Ме България”,“ 2025. [Онлайн]. Available: adoptmebg.com.
- [2] „Пет Бъди“ ООД, „petbuddy.bg”,“ 2025. [Онлайн].
- [3] Инвестор.БГ АД, „Нова платформа помага на стопани да открият домашните си любимци”,“ 26 2 2024. [Онлайн]. Available: <https://www.investor.bg/a/571-it-i-telekomunikatsii/389919-nova-platforma-pomaga-na-stopani-da-otkriyat-domashnite-si-lyubimtsi>.
- [4] Microsoft , „.NET 8”,“ 11 3 2025. [Онлайн]. Available: dotnet.microsoft.com.
- [5] Microsoft, „Visual Studio 2022”,“ 2022. [Онлайн]. Available: visualstudio.microsoft.com.
- [6] Microsoft, „learn.Microsoft.com”,“ 2025. [Онлайн]. Available: <https://learn.microsoft.com/en-us/dotnet/csharp/>.
- [7] Microsoft, „ASP.NET Core”,“ 2024. [Онлайн]. Available: <https://dotnet.microsoft.com/en-us/apps/aspnet>.
- [8] Microsoft, „MSSQL Server”,“ 2025. [Онлайн]. Available: <https://www.microsoft.com/en-us/sql-server/sql-server-downloads>.
- [9] Microsoft, „Download SQL Server Management Studio”,“ 2025. [Онлайн]. Available: <https://learn.microsoft.com/en-us/ssms/download-sql-server-management-studio-ssms>.
- [10] Microsoft, „Entity Framework Core”,“ 2024. [Онлайн]. Available: <https://learn.microsoft.com/en-us/ef/core/>.
- [11] Wikipedia, „Language Integrated Query”,“ 2024. [Онлайн]. Available: https://en.wikipedia.org/wiki/Language_Integrated_Query.
- [12] Microsoft, „Introduction to Identity on ASP.NET Core”,“ 2024. [Онлайн].
- [13] Wikipedia, „Браузър”,“ 2017. [Онлайн].



- [14] w3schools, „HTML Tutorial,“ 2024. [Онлайн].
- [15] w3schools, „W3.CSS Tutorial,“ 2024. [Онлайн].
- [16] w3schools.com, „JavaScript Tutorial,“ 2023. [Онлайн].
- [17] w3schools.com, „ASP.NET Razor - Markup,“ 2024. [Онлайн].
- [18] BootStrap, „BootStrap,“ 2024. [Онлайн]. Available: <https://getbootstrap.com/>.
- [20] И. Първанов, Учебник по Информатика за 12. клас – профилирана подготовка, издателство Изкуства, 2023.

V. Приложения

ФИГУРА 1. “USE CASE” ДИАГРАМА НА ПЛАТФОРМАТА	13
ФИГУРА 2. “UML” СХЕМА НА БАЗАТА ДАННИ	15
ФИГУРА 3. МОДЕЛА PET.....	19
ФИГУРА 4. МОДЕЛА TOWN.....	21
ФИГУРА 5. МОДЕЛА USER	22
ФИГУРА 6. МОДЕЛА USERREQUEST	23
ФИГУРА 7. ИЗБРОИМИЯТ ТИП PETTypeEnum.....	25
ФИГУРА 8. ИЗБРОИМИЯТ ТИП RoleEnum	25
ФИГУРА 9. ИЗБРОИМИЯТ ТИП PETAgeEnum.....	26
ФИГУРА 10. ИЗБРОИМИЯТ ТИП GenderEnum.....	26
ФИГУРА 11. КЛАСА SELECTOPTION.....	27
ФИГУРА 12. МЕТОДА ToDescriptionString В КЛАСА EnumExtensions	28
ФИГУРА 13. ИНТЕРФЕЙСА IDbWithNav	29
ФИГУРА 14. ИНТЕРФЕЙСА IDbWithoutNav	30
ФИГУРА 15. КЛАСА PetExchangeDbContext	31
ФИГУРА 16. КЛАСА ConnectionString.....	31
ФИГУРА 17. КЛАСА PetDbContext	32
ФИГУРА 18. МЕТОДА ReadAllWithFilterAsyncOfUser	33
ФИГУРА 19. МЕТОДА ReadWithFiltersAsync.....	33
ФИГУРА 20. МЕТОДИТЕ Read4NewestAsync() И Read4OldestAsync().....	34
ФИГУРА 21. МЕТОДИТЕ ЗА ЗАПАЗВАНЕ В PetDbContext	35
ФИГУРА 22. МЕТОДИТЕ ЗА ЧЕТЕНЕ В PetDbContext	35
ФИГУРА 23. МЕТОДА ЗА АКТУАЛИЗАЦИЯ В PetDbContext	36
ФИГУРА 24. МЕТОДА ЗА ИЗТРИВАНЕ В PetDbContext	37
ФИГУРА 25. КЛАСА TownDbContext	37
ФИГУРА 26. МЕТОДА ЗА ИЗТРИВАНЕ В TownDbContext	38
ФИГУРА 27. КЛАСА UserDbContext.....	38
ФИГУРА 28. ДОБАВЯНЕ И ПРЕМАХВАНЕ НА РОЛИ С UserManager ПРИ МЕТОДА ЗА АКТУАЛИЗАЦИЯ НА UserDbContext	39
ФИГУРА 29. КЛАСА UserRequestsDbContext	39
ФИГУРА 30. МЕТОДИТЕ ЗА OUTBOX И INBOX В UserRequestDbContext	40



ФИГУРА 31. МЕТОДА CANCELASYNC НА КЛАСА USERREQUESTSDbCONTEXT	40
ФИГУРА 32. МЕТОДА ACCEPTASYNC НА КЛАСА USERREQUESTSDbCONTEXT	41
ФИГУРА 33. МЕТОДА DENYASYNC НА КЛАСА USERREQUESTSDbCONTEXT	41
ФИГУРА 34. КЛАСА PetSERVICE	42
ФИГУРА 35. КЛАСА TownSERVICE	43
ФИГУРА 36. КЛАСА UserService	43
ФИГУРА 37. КЛАСА UserREQUESTSService	44
ФИГУРА 38. ПРИМЕР ЗА PetCARD	47
ФИГУРА 39. МОДАЛЪТ НА PetCARD	48
ФИГУРА 40. ПРИМЕР ЗА ТАБЛИЦА	48
ФИГУРА 41. ОФОРМЛЕНИЕТО LayoutSite	49
ФИГУРА 42. ОФОРМЛЕНИЕТО LayoutAccount	50
ФИГУРА 43. ФИЛТРИРАНЕ В ГЛАВНАТА СТРАНИЦА ЗА ДОМАШНИ ЛЮБИМЦИ	51
ФИГУРА 44. ВИЗУАЛИЗАЦИЯ НА ФИЛТРИ ЧРЕЗ FilterGroup КОМПОНЕНТ	51
ФИГУРА 45. КЛАСА Program.cs	52
ФИГУРА 46. ПРИМЕР ЗА ТЕСТОВ МЕНИДЖЪР	53
ФИГУРА 47. ПРИМЕР ЗА КОМПОНЕНТЕН ТЕСТ	54