

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

бакалавра

(ступінь вищої освіти)

на тему ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДОПОМОГИ

(назва теми)

ДЛЯ ДОГЛЯДУ ЗА ТВАРИНАМИ

PET CARE ASSISTANCE SOFTWARE

Виконав(ла): студент(ка) 4 курсу, групи КНТ-220сп

Спеціальності 122 Комп'ютерні науки

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Комп'ютерні науки

БЄЛЯКІНА К.Ю.

(ПРІЗВИЩЕ та ініціали)

Керівник ЛЬОВКІН В.М.

(ПРІЗВИЩЕ та ініціали)

Рецензент ТІМЕНКО А.В.

(ПРІЗВИЩЕ та ініціали)

2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет ФКНТ
Кафедра програмних засобів
Ступінь вищої освіти бакалавр
Спеціальність 122 Комп'ютерні науки
(код і найменування)
Освітня програма (спеціалізація) Комп'ютерні науки
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
« » 2023 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

БЄЛЯКІНОЇ Катерини Юріївни

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Програмне забезпечення допомоги для догляду за тваринами. Pet Care Assistance Software

керівник проєкту (роботи) к.т.н., доцент ЛЬОВКІН Валерій Миколайович,
(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від «04» квітня 2023 року № 87

2. Строк подання студентом проєкту (роботи) 1 червня 2023 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Проєктування програми. 3. Розробка програми. 4. Експлуатація та тестування програми.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів)

Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	ЛЬОВКІН В.М., доцент		
Нормоконтроль	БЄЛОВА А.В., асистент		

7. Дата видачі завдання « 4 » квітня 2023 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Срок виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області	2 тиждень	Розділ 1
3	Розробка архітектури програми	3 тиждень	Розділ 2
4	Розробка програми	4-5 тижні	Розділ 3
5	Тестування та експериментальне дослідження програми	6 тиждень	Розділ 4
6	Оформлення пояснювальної записки та документів до неї. Нормоконтроль та рецензування	7 тиждень	Додатки
7	Захист роботи	8 тиждень	

Студент(ка)

_____ Катерина БЄЛЯКІНА
(підпис) (Ім'я ПРИЗВИЩЕ)

Керівник проєкту (роботи)

_____ Валерій ЛЬОВКІН
(підпис) (Ім'я ПРИЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 94 с., 27 рис., 3 табл., 3 дод., 12 джерел.

ДОМАШНІ ТВАРИНИ, ПОМІЧНИК З ДОГЛЯДУ, ВЛАСНИК ТВАРИНИ, БРОНЮВАННЯ, ВЕБДОДАТОК.

Об'єкт роботи – процес догляду за тваринами. Предмет роботи – програмне забезпечення допомоги для догляду за тваринами. Мета роботи – розробити програмне забезпечення допомоги для догляду за тваринами для надання можливості взаємовигідної взаємодії між власниками тварин і помічниками з догляду за тваринами.

Матеріали, методи та технічні засоби: мови Python, HTML, CSS, JavaScript, фреймворк Django, система керування базами даних PostgreSQL.

Результати. Результати роботи включають програмне забезпечення, розроблене в підсумку, та комплекс результатів окремих етапів, що в підсумку дозволили створити вебдодаток, який дозволяє власникам тварин шукати помічників з догляду як для вигулювання тварин, коли вони не здатні це зробити, так і для відвідування тварин у момент їх відсутності вдома, а також для реалізації тривалого догляду на території власника або помічника, а помічникам дозволяє виставляти власні пропозиції, визначаючи гнучко доступний робочий час, тарифи на обслуговування, аналізуючи ефективність.

Висновки. Визначений комплекс етапів і процедур дозволив розробити програму, підсумкове тестування якої встановило відповідність вимогам загалом і функціональним вимогам до програми зокрема, визначеним на початкових етапах.

Галузь використання. Програма придатна для використання як окрема платформа, на якій беруть участь власники домашніх тварин та люди, які готові працювати в ролі помічників з догляду за тваринами.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 94 pages, 27 figures, 3 tables, 3 appendixes, 12 sources.

PETS, CARE ASSISTANT, PET OWNER, BOOKING, WEB APP.

The object of work is the process of caring for animals. The subject of the work is pet care assistance software. The goal of the work is to develop pet care assistance software to enable mutual interaction between pet owners and pet care assistants.

Materials, methods and technical means: Python, HTML, CSS, JavaScript languages, Django framework, PostgreSQL database management system.

The results. The results of the work include the software developed in the end and the set of results of the individual stages, which ultimately allowed the creation of a web application that allows pet owners to search for care assistants both to walk their animals when they are unable to do so, and to visit their animals in the moment of their absence at home, as well as for the implementation of long-term care on the territory of the owner or assistant, and it allows assistants to issue their own offers, determining flexibly available working hours, service tariffs, analyzing efficiency.

Conclusions. The defined set of stages and procedures made it possible to develop a program, the final testing of which established compliance with the requirements in general and the functional requirements for the program in particular, determined at the initial stages.

Field of use. The program is suitable for use as a stand-alone platform for pet owners and people who are willing to work as pet care assistants.

ЗМІСТ

	С.
Перелік скорочень та умовних познач.....	7
Вступ.....	8
1 Аналіз предметної області.....	9
1.1 Огляд програмного забезпечення допомоги для догляду за тваринами .	9
1.2 Порівняння аналогів	12
1.3 Висновки за розділом 1	14
2 Проєктування програми.....	15
2.1 Вибір засобів проєктування і розробки	15
2.2 Визначення функціональних вимог до програмного забезпечення	16
2.3 Проєктування структури бази даних.....	23
2.4 Висновки за розділом 2	37
3 Розробка програми	38
3.1 Порядок функціонування програми.....	38
3.2 Рішення з реалізації одиниць додатку	39
3.3 Висновки за розділом 3	44
4 Експлуатація та тестування програми	45
4.1 Опис програми.....	45
4.2 Експлуатація програми.....	45
4.3 Тестування програми.....	52
4.4 Висновки за розділом 4	55
Висновки	56
Перелік джерел посилання	57
Додаток А Технічне завдання	59
Додаток Б Текст програми	64
Додаток В Слайди презентації.....	87

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

БД – база даних;

ПО – предметна область.

ВСТУП

Після завершення пандемії COVID-19 багато власників домашніх тварин стикається з проблемою: вони більше не можуть приділяти стільки часу, вигулювати своїх домашніх тварин стільки, скільки робили це під час пандемії [1]. З подібною проблемою зіштовхуються і люди, які традиційно багато працюють: для них мати домашню тварину стає проблемним, адже вони не кожен день щонайменше можуть приділити достатню увагу своїм улюбленицям. Однак кожна з описаних сторін потребує вирішення проблеми.

Загалом тварини потребують постійного догляду. Кожен день потрібно їх годувати, з певною періодичністю вигулювати, прибирати за ними. Ще однією стороною проблеми є ситуація, коли власники їдуть на відпочинок, мають відрядження. Щонайменше не завжди вони можуть взяти з собою домашніх тварин. Тому вони в такий момент потребують знайти людину, яка здатна доглядати за їхніми тваринами. Кількість домашніх тварин у світі загалом і у кожній країні зокрема є величезною: наприклад, у Великобританії 12 мільйонів собак і 7,75 мільйонів котів, в Україні 7,5 мільйонів котів, а в США їх кількість є найбільшою у світі і досягає 76,5 мільйонів, що додатково підкреслює актуальність проблеми [1]-[2].

Люди за певну плату готові допомогти іншим людям, а тому важливо дозволити їм знайти один одного, при цьому надаючи можливість отримати інформацію один про одного, яка визначається тими людьми, які вже скористалися їхніми послугами. За наявності такого зручного програмного забезпечення ці користувачі зможуть взаємодіяти і допоможуть один одному вирішити достатньо буденну, але це одночасно значить і важливу проблему.

У статті [1] підкреслюється, що догляд за тваринами можна розділити на вигулювання тварин та щоденний догляд. Звичайно, що кожен з цих варіантів має свої особливості, а тому при розробці програми потрібно врахувати такі особливості, щоб зробити створений інструмент зручним для використання всіма сторонами для розв'язання всіх основних завдань.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд програмного забезпечення допомоги для догляду за тваринами

Цілий ряд додатків, переважно вебдодатків, дозволяє на певному рівні вирішити проблему допомоги для догляду за тваринами, якою вона розглядається в роботі. Проте важливо підкреслити, що це розв'язання є абсолютно локальним. Для всіх існуючих додатків так чи інакше використовується обмеження на географію послуг. Тому скористатися цими інструментами в Україні не можна. І поточний стан проблеми в Україні виглядає так, що помічника з догляду за тваринами потрібно шукати через соціальні мережі, що є незручним і до того ж не дозволяє адекватно оцінити цього помічника, щоб розуміти, кому саме власник віддасть свою тварину.

Частина вебдодатків взагалі є недоступною на території України, тобто ознайомитися з їх послугами, знаходячись на території України взагалі неможливо. Тому в огляді акцент зроблений на функціональні можливості, на особливості предметної області (ПО), сутності та функції цієї ПО.

Rover.com [3] є як раз прикладом тих вебдодатків, які не діють на території України. Це вебдодаток, який має велику кількість важливих для цієї ПО функцій, тому він був розглянутий, але для власників з України та багатьох інших країн він фактично є недоступним. Це американська компанія, тому послуги надаються на території США. Користувачам надається можливість бронювати послуги як з вигулювання собак і котів (це єдині домашні тварини, які можуть обслуговуватися), так і відвідування тварин вдома, догляду за тваринами вдома у власника та у помічника з догляду. Окремо виділяється ночівля у помічника з догляду вдома, а окремо просто догляд у помічника вдома в інший час.

Вебсайт Petcloud [4] (рис. 1.1) надає можливість працювати к різних країнах, включаючи Австралію, Нову Зеландію, США, Канаду, Південну Корею, Австрію, Бельгію, Чехію, Данію, Фінляндію, Німеччину, Угорщину,

Ірландію, Італію, Нідерланди, Норвегію, Польщу, Португалію, Іспанію, Швецію, Швейцарію, Великобританію. Тобто кількість варіантів більша, проте Україна, наприклад, відсутня.

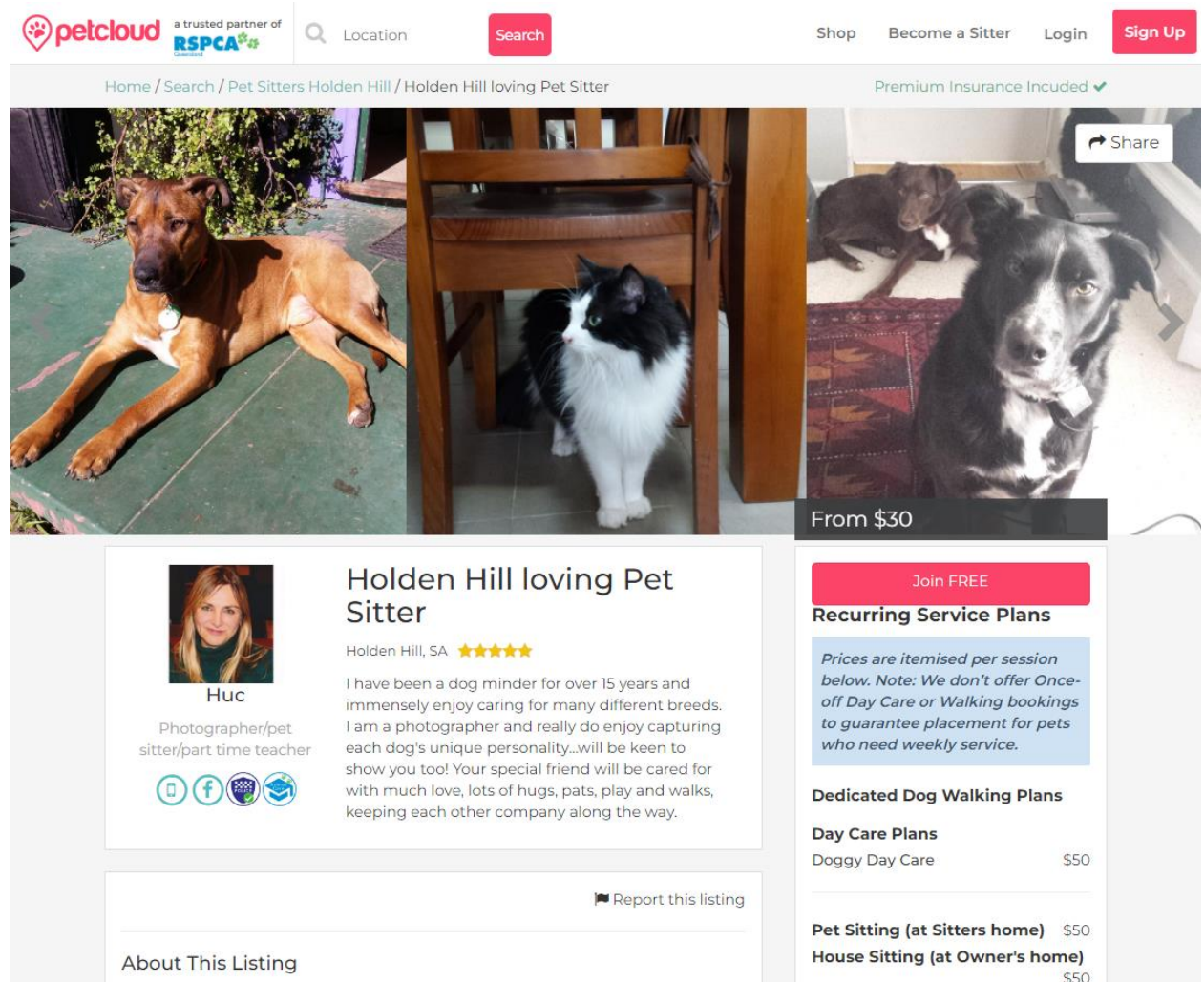
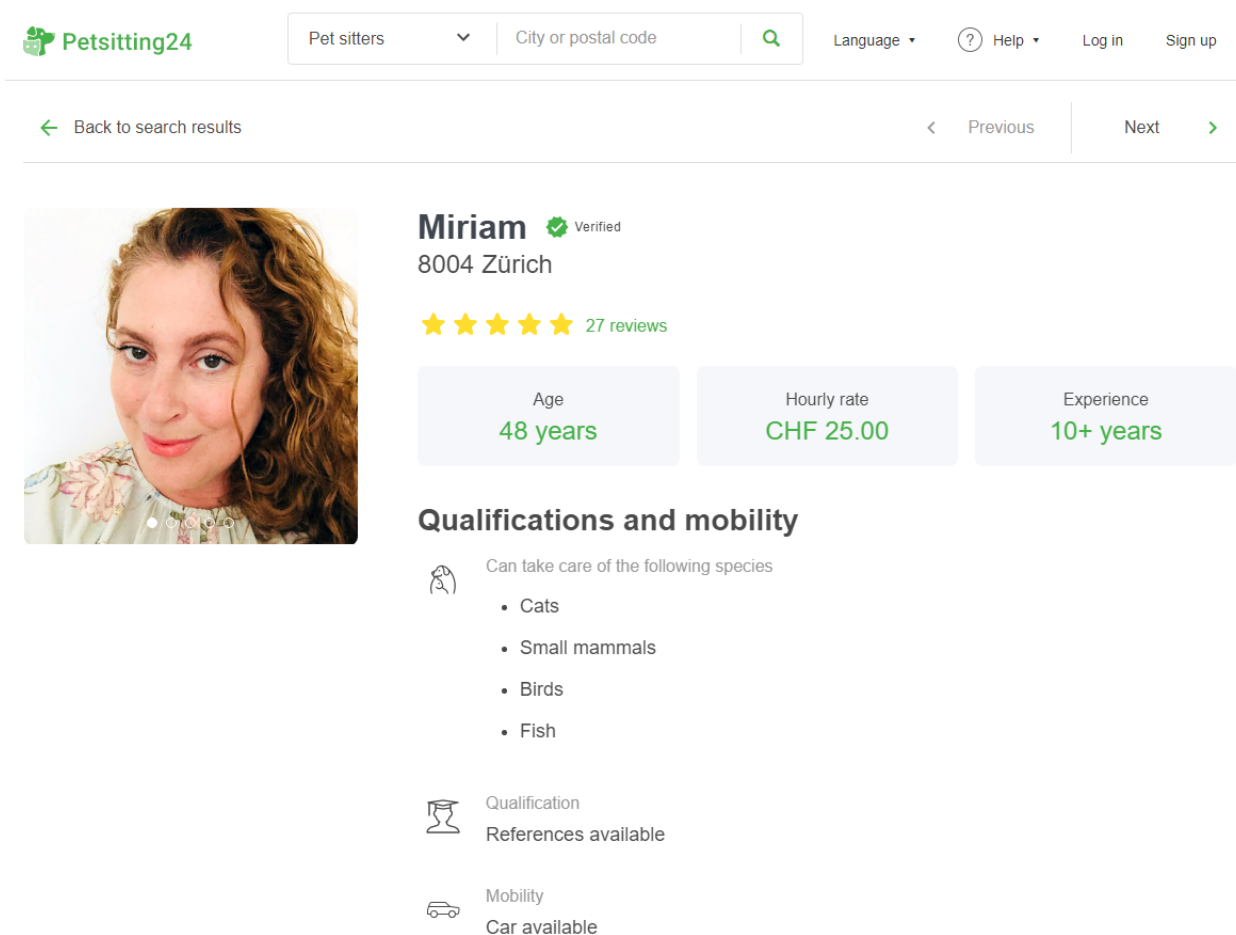


Рисунок 1.1 – Вебсайт Petcloud

Petcloud дозволяє визначити під час пошуку адресу, тип тварини (серед варіантів присутній і інші), тип сервісу (догляд у власника тварини або помічника, відвідування, вихулювання, таксі, тренування, прибирання, дезінфекція та інші) та період догляду. Визначається саме день в якості дати, тобто вказати час не можна. Тарифи встановлюються на кожну конкретну послугу, тобто задати різні тарифи на різний час обслуговування, на різну тривалість неможливо. Помічник вказує розмір тварин, які він хотів би доглядати в першу чергу, але не має можливість вказати породу.

Petsitting24 [5] (рис. 1.2) діє тільки в Швейцарії, що значно обмежує використання. Однак це один з представників подібних вебсайтів. Такі ж системи наявні у Бельгії, Угорщини і т.д. Пошук відбувається за містом, визначається вид тварини, прийнятна вартість за годину, вік помічника, стать. Більше налаштувань недоступно. Тарифи тільки погодинні, інших варіантів недоступно. Тварини можуть обслуговуватися різні, користувачі можуть залишати відгуки, в результаті накопичується оцінка. При цьому все базується на погодинному обслуговуванні, тому не виділяється окремо вигулювання, відвідування, догляд на тривалий період.



The screenshot shows the profile of a pet sitter named Miriam on the Petsitting24 website. The profile includes a photo of Miriam, a verified status, and her location in Zurich. It also displays her age (48 years), hourly rate (CHF 25.00), and experience (10+ years). A section titled 'Qualifications and mobility' lists the species she can care for (Cats, Small mammals, Birds, Fish), her qualifications (References available), and her mobility (Car available).

Petsitting24 Pet sitters City or postal code Language ? Help Log in Sign up

← Back to search results Previous Next →

Miriam  Verified
8004 Zürich

★★★★★ 27 reviews

Age 48 years	Hourly rate CHF 25.00	Experience 10+ years
------------------------	---------------------------------	--------------------------------

Qualifications and mobility

 Can take care of the following species

- Cats
- Small mammals
- Birds
- Fish

 Qualification
References available

 Mobility
Car available

Рисунок 1.2 – Вебсайт Petsitting24

Wag [6] (рис. 1.3) надає подібну функціональність, діє тільки в США і тільки для собак і котів. На вебсайті доступні статті про догляд, але вони створюються на помічниках, а власниками вебсайту загалом. За

помічниками обчислюються оцінки за відгуками, надаються самі відгуки та рекомендації. Помічник може змінити персонально вартість обслуговування, але загалом тариф встановлюється на конкретну процедуру загалом.

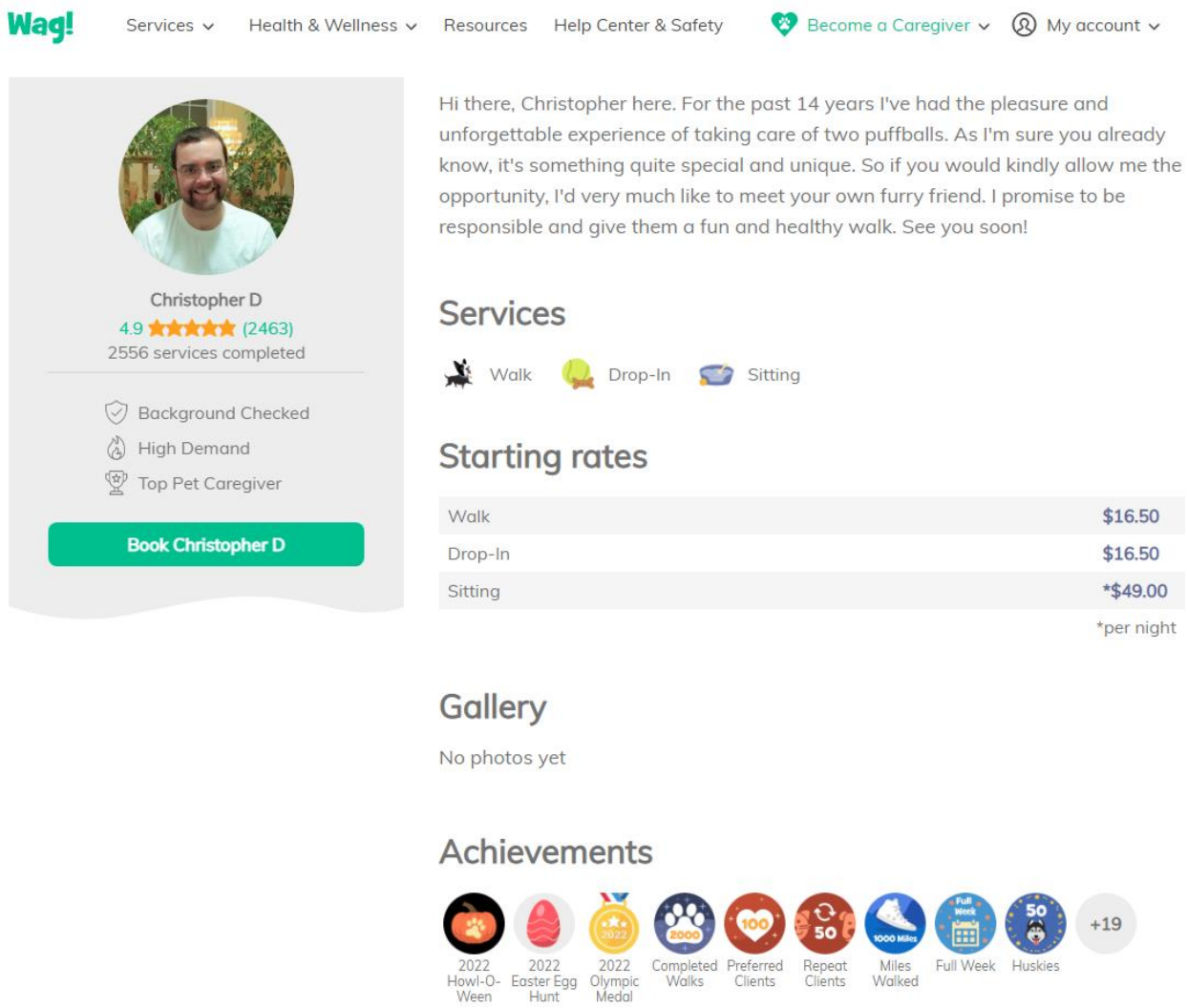


Рисунок 1.3 – Вебсайт Wag

1.2 Порівняння аналогів

Виконаний огляд у першу чергу дозволив проаналізувати ПО і визначити основні сутності та функції, які мають забезпечуватися в цій ПО.

Порівняння доступних вебсайтів дозволило виділити їх основні відмінності і спільні риси, визначити основні для розробки та сформулювати відмінності розробки (табл. 1.1).

Таблиця 1.1 – Порівняння вебдодатків за результатами огляду

Характеристика	Rover.com	Petcloud	Petsitting24	Wag	Розробка
Територія дії	США	23 країни, Україна відсутня	Швейцарія	США	Потенційно будь-яка
Види тварин	Собаки, коти	Усі	Усі	Собаки, коти	Усі
Гнучка система тарифів	Ні	Ні	Ні, погодинна	Ні	Так
Налаштування послуг на рівні визначення порід тварин	Ні	Досвід помічника з різними породами	Ні	Ні	Так
Статті про догляд	Ні	Ні	Ні	Тільки загальні	Так
Типи сервісів	Вигулювання, відвідування, догляд у власника, у помічника	Вигулювання, відвідування, догляд у власника, у помічника та інші	Не деталізовано	Вигулювання, відвідування, догляд у власника, у помічника	Вигулювання, відвідування, догляд у власника, у помічника

По-перше, розробка дозволить охопити користувачів в Україні. По-друге, вона дозволить помічникам самим створювати статті про догляд

тварин, тому власники зможуть отримувати про них і від них більше інформації. По-третє, передбачається гнучка система тарифів, тобто помічник зможе встановити різну вартість послуг на різний період дня, а також визначити, що послуга може тривати різний час, змінюючи таким чином вартість послуги. По-четверте, розробка дозволить помічнику визначити, що з певними породами тварин він взагалі не бажає працювати.

1.3 Висновки за розділом 1

Мета роботи має бути виконана через успішне виконання таких завдань:

- аналіз існуючого програмного забезпечення, що підтримує догляд за тваринами;
- проектування програмного забезпечення допомоги для догляду за тваринами;
- реалізація і тестування програми.

У розділі 1 виконано аналіз існуючого програмного забезпечення, що підтримує догляд за тваринами, визначено основні функції таких вебдодатків та основні характеристики, в тому числі відмінності, розробки.

2 ПРОЄКТУВАННЯ ПРОГРАМИ

2.1 Вибір засобів проєктування і розробки

Розробку програми має бути виконано, використовуючи мову Python [7]-[9], адже вона дозволяє розробляти вебдодатки достатньо високого рівня, але при цьому ще й робить менш витратним процес розробки за рахунок повторного використання. На подальшому кроці виконував вибір фреймворку, адже без зручного фреймворку веброботка може бути проблемною, а тоді і вибір мови може бути вимушено зміненим, тому цей вибір був важливим. Порівняння виконувалось для фреймворків Django [10]-[11] і одного з нових фреймворків Falcon. Другий надає багато можливостей налаштування, однак для розробки програми в роботі потрібно забезпечити ті вимоги, які винесені в основні показники (табл. 2.1). Відповідно Falcon вимагає додаткової роботи з підключення, налаштування цих інструментів. Тому вибрано Django.

Таблиця 2.1 – Порівняння фреймворків Django і Falcon для розробки вебдодатку

Показник	Django	Falcon
Тип фреймворку	Фреймворк за принципом «все включено»	Мінімалістичний бекенд-фреймворк
Фреймворк для роботи з БД	Вбудований	Можливе підключення
Панель керування для адміністратора	Вбудована	Можливе підключення
Бібліотека шаблонів	Вбудована	Можливе підключення
Забезпечення безпеки	Вбудоване, багатоцільове	Вбудовані засоби майже повністю відсутні

2.2 Визначення функціональних вимог до програмного забезпечення

Робота користувача (рис. 2.1) повинна підтримуватися на основі наступних сценаріїв, що відповідають фактично неавторизованим користувачам.

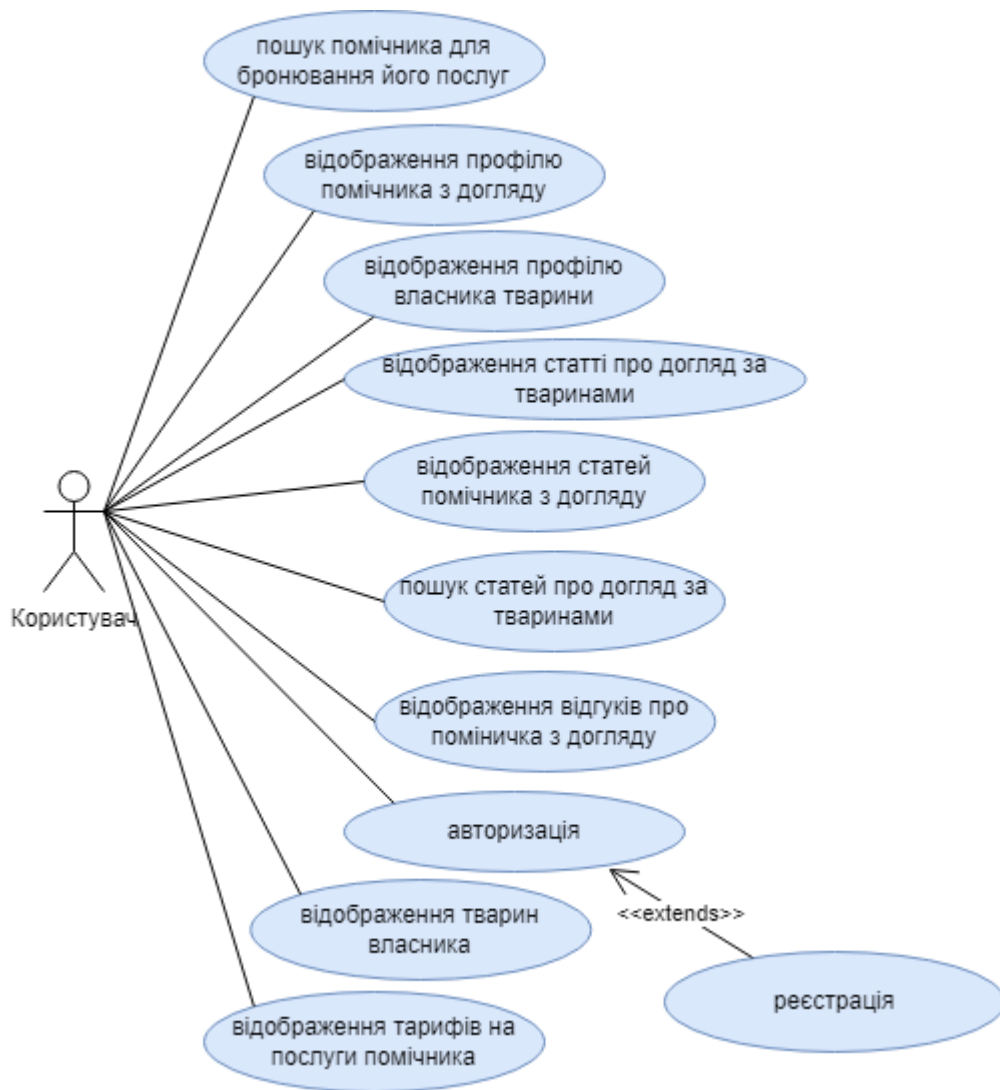


Рисунок 2.1 – Діаграма прецедентів користувача

Сценарії:

- пошук помічника для бронювання його послуг;
- відображення профілю помічника з догляду;
- відображення профілю власника тварини;

- відображення статей помічника з догляду;
- відображення статті про догляд за тваринами;
- пошук статей про догляд за тваринами;
- відображення відгуків про помічника з догляду;
- авторизація;
- відображення тварин власника;
- відображення тарифів на послуги помічника.

Альтернативний сценарій передбачено один – реєстрація. Він реалізується у тому випадку, якщо користувач не був зареєстрований раніше або не має доступу до власного облікового запису. Він реалізується як альтернатива сценарію авторизації.

Робота власника тварини повинна підтримуватися на основі наступних сценаріїв:

- пошук помічника для бронювання його послуг;
- відображення профілю помічника з догляду;
- відображення профілю власника тварини;
- відображення статей помічника з догляду;
- пошук статей про догляд за тваринами;
- відображення відгуків про помічника з догляду;
- бронювання послуг з догляду;
- відображення доступних для відгуку помічників з догляду;
- створення відгуку;
- відображення тварин власника;
- додавання тварини власника;
- створення групи тварин;
- додавання тварини до групи;
- відображення статті про догляд за тваринами;
- підтвердження додавання тварини до групи;
- відображення груп, до яких внесено тварину власника;
- відображення груп, якими керує власник;

- коментування статті;
- відображення бронювань за власною твариною;
- відображення бронювань за всіма тваринами, з якими пов'язаний користувач;
- підтвердження бронювання;
- відображення всіх користувачів, яким писав користувач;
- відправлення повідомлення;
- читання повідомлень;
- внесення до переліку власника помічника;
- відображення помічників з переліку;
- відображення тарифів на послуги помічника;
- відображення помічників, послуги яких бронював власник тварини;
- відображення бронювань послуг помічника власником тварин.

Сценарій бронювання послуг з догляду узагальнює сценарії бронювання, кожен з яких має певну специфіку взаємодії і визначення необхідних даних:

- бронювання прогулянки тварини з помічником з догляду;
- бронювання відвідування помічником тварини;
- бронювання догляду тварини помічником на території помічника;
- бронювання догляду тварини помічником у приміщенні власника.

Створення відгуку узагальнює сценарій створення рекомендації, який є його специфічною реалізацією. Фактично рекомендація є тим же самим відгуком, але по-перше, створеним власником тварини про помічника з догляду за тваринами, а по-друге, такий специфічний відгук власник тварини може створити тільки 1 на місяць.

Моделювання сценаріїв роботи власника тварини з програмою на основі описаної множини сценаріїв виконано за допомогою діаграми прецедентів (рис. 2.2).

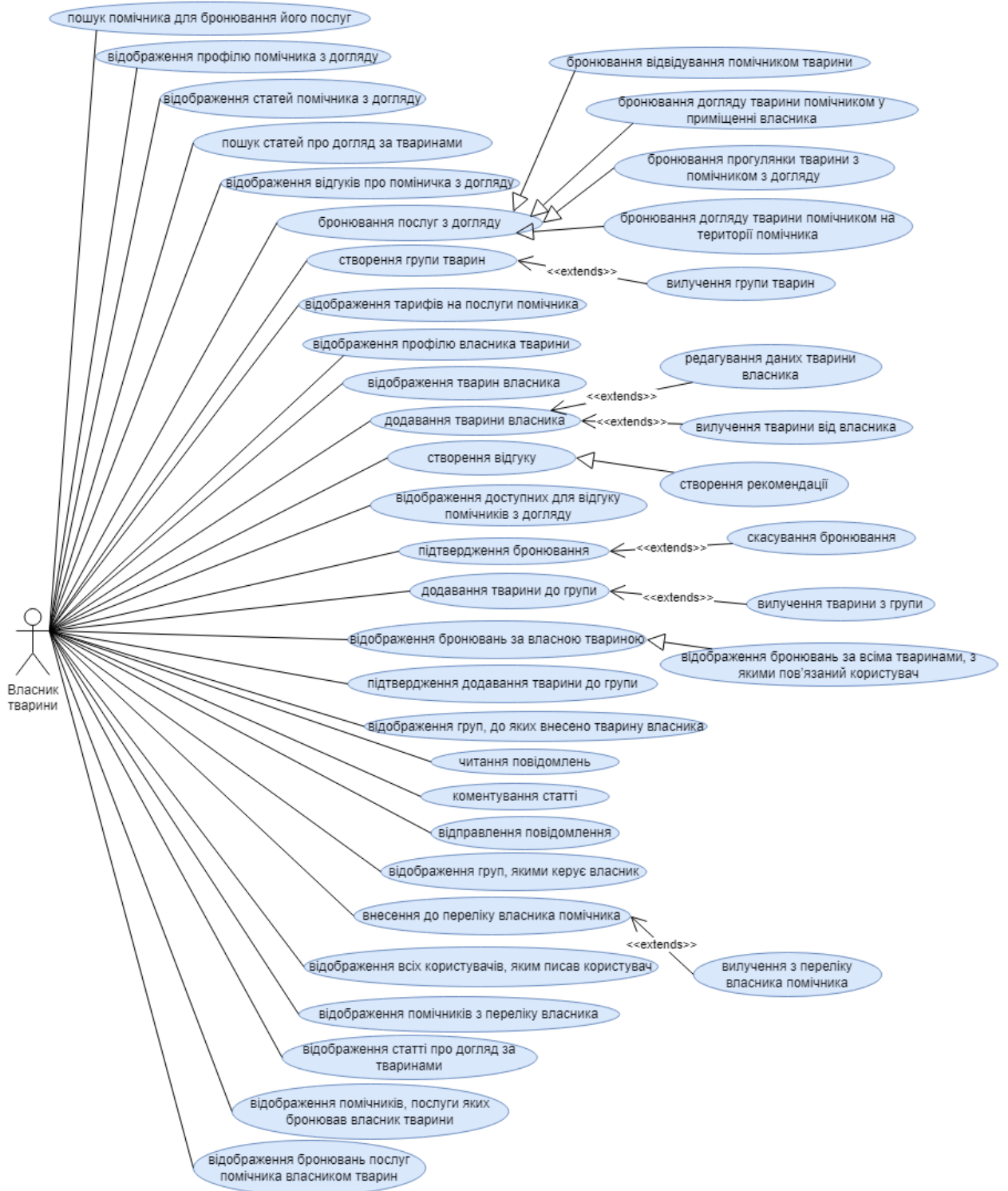


Рисунок 2.2 – Діаграма прецедентів власника тварини

Для власника тварин доступний наступний набір альтернативних сценаріїв:

- скасування бронювання, що є альтернативним сценарієм для підтвердження бронювання у випадку, коли власник тварини змінив плани і хоче відмовитися від заброньованого обслуговування;
- вилучення групи тварин, що є альтернативним сценарієм для створення групи тварин і що задіюється у тому випадку, якщо група тварин вже створена, але власник більше не потребує роботи з нею;
- редагування даних тварини власника, що є альтернативним сценарієм для додавання тварини власника і що задіюється у тому випадку, якщо дані про тварину вже внесені, але вони не є актуальними;
- вилучення тварини від власника, що є альтернативним сценарієм для додавання тварини власника і задіюється у тому випадку, якщо тварину власник вже вніс, але після того ці дані стали з тієї чи іншої причини некоректними, адже тварина перестала належати даному власнику;
- вилучення тварини з групи, що є альтернативним сценарієм для додавання тварини до групи і задіюється у тому випадку, якщо власник цієї тварини більше не бажає, щоб вона входила до певної групи або у тому випадку, коли користувач, який керує цією групою не включає більше цю тварину в цю групу;
- вилучення з переліку власника помічника, що є альтернативним сценарієм для внесення до переліку власника помічника і задіюється у тому випадку, коли користувач більше не бажає, щоб цей помічник відображався у його переліку помічників.

Робота помічника з догляду за тваринами повинна підтримуватися на основі наступних сценаріїв:

- пошук помічника для бронювання його послуг;
- відображення профілю помічника з догляду;
- відображення профілю власника тварини;
- відображення статей помічника з догляду;
- пошук статей про догляд за тваринами;
- відображення статті про догляд за тваринами;

- відображення відгуків про помічника з догляду;
 - відображення особистого відкритого для роботи часу;
 - додавання особистого відкритого для роботи часу;
 - відображення доступних для відгуку власників;
 - створення відгуку;
 - відображення тварин власника;
 - коментування статті;
 - визначення успішності помічника з різними видами і породами тварин;
 - відображення всіх бронювань у помічника з догляду;
 - підтвердження бронювання;
 - створення статті про догляд за тваринами;
 - відображення всіх користувачів, яким писав користувач;
 - відправлення повідомлення;
 - читання повідомлень;
 - відображення видів і пород тварин, з якими працює помічник;
 - додавання породи тварин, з якими працює помічник;
 - відображення тарифів на послуги помічника;
 - додавання тарифу на послуги помічника;
 - відображення власників тварин, які бронювали послуги помічника;
 - відображення всіх обслуговувань у помічника з догляду за датою
- (рис. 2.3).

Сценарій відображення всіх бронювань у помічника з догляду узагальнює сценарії відображення всіх обслуговувань у помічника з догляду за датою та відображення бронювань послуг помічника власником тварин, що фактично є специфічними сценаріями, які деталізують певну частину даних, з якими відбувається роботи в узагальненому сценарії.

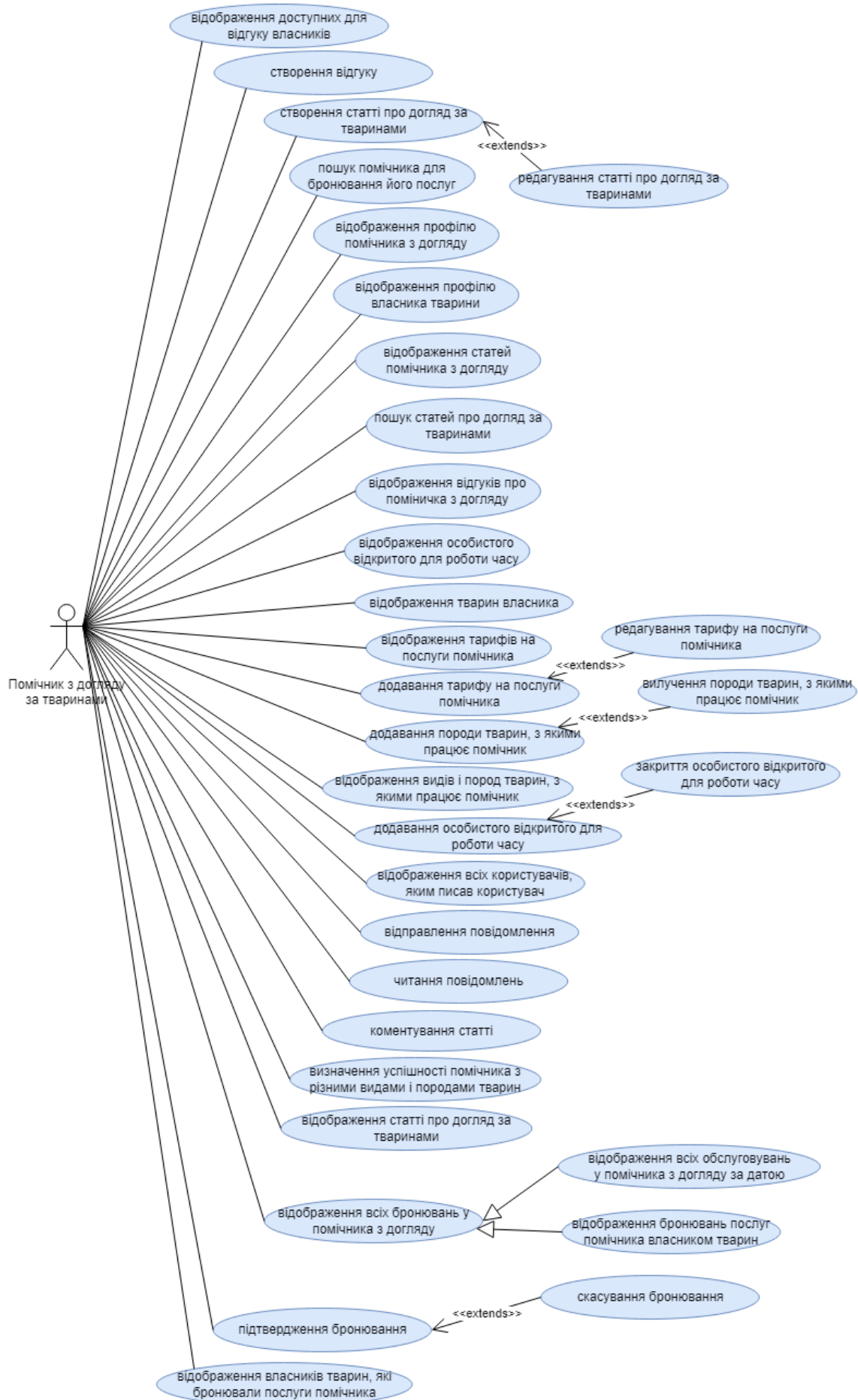


Рисунок 2.3 – Діаграма прецедентів помічника з догляду за тваринами

Для помічника з догляду за тваринами доступний наступний набір альтернативних сценаріїв:

- редагування статті про догляд за тваринами, що є альтернативним сценарієм для створення статті про догляд за тваринами і здійснюється у тому випадку, коли стаття вже створена, але до неї автор бажає внести зміни;
- редагування тарифу на послуги помічника, що є альтернативним сценарієм для додавання тарифу на послуги помічника і здійснюється у тому випадку, коли помічник з догляду за тваринами вже створив тариф, але бажає внести до нього зміни;
- вилучення виду тварин, з якими працює помічник, що є альтернативним сценарієм для додавання виду тварин, з якими працює помічник і здійснюється у тому випадку, коли помічник з догляду за тваринами перестав працювати з певним видом тварин;
- закриття особистого відкритого для роботи часу, що є альтернативним сценарієм для додавання особистого відкритого для роботи часу і здійснюється у тому випадку, коли певний часовий проміжок у роботі помічника з догляду за тваринами перестає бути доступним для бронювань власників тварин;
- скасування бронювання, що є альтернативним сценарієм для підтвердження бронювання і здійснюється у тому випадку, коли бронювання вже зареєстровано, але помічник з догляду за тваринами, якого воно стосується, бажає відхилити це бронювання.

2.3 Проєктування структури бази даних

Надійне збереження даних і виконання доступу до них є критичним для такого типу додатків, тому спершу було обрано систему для підтримки керування базами даних (БД) PostgreSQL, адже її характеристиками є відкрита ліцензія, надійність, безпека даних і роботи з ними, масштабованість.

Домашні тварини можуть вимагати різних процедур для догляду за ними. І ці процедури залежать від того, яка саме це тварина. Собак потрібно виховувати, а для котів така процедура може бути щонайменше нестандартною. Тому перш за все потрібно виділити серед сутностей ПО сутність, що визначає вид тварини. Дані про нього зберігаються в таблиці Animal. Вона має дуже просту структуру, що включає первинний ключ id та назву name.

Однак кожна тварина має конкретну породу. Порода може впливати на те, чи погодиться помічник з нею працювати. Це є важливою інформацією для його прийняття рішень. І взагалі важливою характеристикою тварини, тому в програмі потрібен каталог порід тварин. Дані про породи тварин зберігаються в таблиці БД Breed. Вона має структуру, визначену наступними полями:

- id – ідентифікатор породи тварин;
- name – назва породи тварини;
- animal – тварина, яку характеризує ця порода;
- photo – фотографія тварини цієї породи.

Помічники з догляду за тваринами можуть не тільки доглядати за ними і таким чином отримувати певний рейтинг, але і розвивати певні середовища навколо себе, привертаючи увагу. Для цього вони можуть публікувати певні інформаційні матеріали – статті про породи тварин. Вони визначаються в таблиці Breedpost. Вона має структуру, визначену наступними полями:

- id – ідентифікатор статті про породу тварин;
- keeper – помічник з догляду, який опублікував цю статтю;
- breed – порода тварин, якої стосується ця стаття;
- animalshared – позначка про те, що ця стаття характеризує і відповідну тварину загалом, а не тільки породу;
- posttitle – назва статті;
- posttext – текст статті;

- postphoto – фотографія до цієї статті;
- creationtime – час, коли була опублікована стаття.

Статті привертають увагу інших користувачів, а тому можуть містити коментарі від них. Дані про коментарі до статей зберігаються в таблиці BreedpostComment. Вона має структуру, визначену наступними полями:

- id – ідентифікатор коментаря до статті;
- breedpost – стаття, якої стосується коментар;
- user – користувач, який створив цей коментар (зовнішній ключ до таблиці User);
- commenttext – текст коментаря;
- creationtime – час, коли був створений коментар.

Безпосередньо конкретні тварини, якими володіють власники, з якими взаємодіють помічники, представляю ще одну сутність ПО. Ці дані зберігаються в таблиці БД Pet. Вона має структуру, визначену наступними полями:

- id – ідентифікатор тварини;
- alias – ім'я (кличка) тварини;
- photo – фотографія тварини;
- breed – порода тварини (зовнішній ключ до таблиці Breed);
- owner – власник тварини (зовнішній ключ до таблиці Owner);
- feature – опис особливостей тварини;
- since – час, з якого цей власник має цю тварину.

Власники тварин керують багатьма процесами, пов'язаними з тваринами, а дані про них зберігаються в таблиці Owner. Вона має структуру, визначену наступними полями:

- id – ідентифікатор власника тварини;
- user – обліковий запис користувача, який використовує цей власник (зовнішній ключ до таблиці User);
- gender – стать власника;
- photo – фотографія власника;

- birthdate – дата народження власника;
- about – опис власника себе;
- rating – середня оцінка власника від помічників;
- city – місто, в якому живе власник;
- zipcode – індекс (код), який визначає основну локацію проживання власника.

Тварини об'єднуються в групи, щоб потім ними таким чином керувати. Це дозволяє сусідам об'єднати власних тварин разом. Це дозволяє власнику декількох тварин об'єднати їх у групу і зручно потім ними керувати, не вводячи окремо дані кожної з тварин. Тому для визначення тварин саме група виділена як базова одиниця. Дані про групи тварин зберігаються в таблиці Group. Вона має структуру, визначену наступними полями:

- id – ідентифікатор групи тварин;
- name – назва групи тварин;
- owner – користувач-власник, який керує даною групою (зовнішній ключ до таблиці Owner);
- state – стан підтвердження групи іншими власниками: необхідний для тих ситуацій, коли в групі тварини різних власників, тоді вони всі мають підтвердити дозвіл на участь у групі перед тим, як нею можна буде керувати.

Групи об'єднують різних тварин, а тварини можуть входити до різних груп, тому реалізовано цей зв'язок типу багато-до-багатьох через створення таблиці GroupPet. Вона має стандартну структуру для такої таблиці, включаючи номер групи group з однойменної таблиці та номер тварини pet з відповідної ще однієї однойменної таблиці. Додатковим полем є поле activation, яке визначає, чи підтвердив участь тварини у групі її власник.

Самі помічники з догляду за тваринами визначаються даними, які зберігаються в таблиці БД Кеерер. Вона має структуру, визначену наступними полями:

- id – ідентифікатор помічника з догляду за тваринами;

- user – обліковий запис користувача, який використовує цей помічник (зовнішній ключ до таблиці User);
- gender – стать помічника з догляду за тваринами;
- photo – фотографія помічника;
- license – номер ліцензії, якщо в країні відбувається ліцензування;
- rating – середня оцінка помічника від власників;
- recommendations – кількість рекомендацій власника;
- startoperation – час, коли почав працювати помічник з догляду за тваринами у цій програмі;
- lastoperation – час, коли виконав останній на даний момент догляд за тваринами даний помічник;
- mobility – можливість переміщення помічника з догляду за тваринами поза межами району, визначеного індексом;
- about – опис помічником з догляду за тваринами себе;
- city – місто, в якому працює помічник з догляду за тваринами (зовнішній ключ до таблиці City);
- zipcode – індекс (код), який визначає основну локацію проживання або роботи помічника з догляду за тваринами;
- birthdate – дата народження помічника з догляду за тваринами;
- withpets – наявність власних тварин у помічника з догляду за тваринами;
- waiting – середній час очікування відповіді від помічника з догляду за тваринами;
- yardtype – тип подвір'я у місці проживання помічника з догляду за тваринами: дозволяє вказати, що помічник має власне подвір'я, наприклад, огорожене, де може вигулювати тварин, якщо забирає їх до себе додому, або що це подвір'я неогорожене, або що його взагалі немає;
- park – наявність парку поблизу з місцем проживання помічника з догляду за тваринами;

- visit – надання послуг з відвідування тварин у власника цим помічником з догляду за тваринами;
- walking – надання послуг з вигулювання тварин цим помічником з догляду за тваринами;
- boarding – надання послуг з догляду за твариною у власному приміщенні (у себе вдома) цим помічником з догляду за тваринами;
- sitting – надання послуг з догляду за твариною власника на території власника цим помічником з догляду за тваринами.

Міста є базовою ознакою в процесі пошуку, тому їх було для зручності внесення даних у програму винесено у окрему таблицю City. Вона має структуру, визначену наступними полями:

- id – ідентифікатор міста;
- name – назва міста;
- country – країна міста (визначається програмно заданим набором на основі перелічення).

Помічники з догляду за тваринами працюють зазвичай не з будь-якими тваринами, а з тими, з якими вміють поводитися. Тому кожен з них має визначити перелік порід тварин, з якими він може допомогти власникам. Але кожна порода тварин звичайно може стосуватися різних помічників. Тому довелося реалізовувати зв'язок багато-до-багатьох між цими таблицями, а це потребувало введення проміжної таблиці KeeperBreed, яка відповідно містить як зовнішній ключ ідентифікатор помічника з догляду за тваринами keeper та ідентифікатор породи тварин breed.

Власники тварин для зручної роботи повинні мати можливість створити переліки помічників. Це може допомогти, коли власнику потрібно знайти іншого помічника, тоді працюючи з програмою, він може поступово зберігати кандидатури. Ці дані накопичуються через таблицю KeeperList. Вона має структуру, визначену наступними полями:

- id – ідентифікатор входження помічника до переліків власників;

- owner – власник тварин, який зберіг цього помічника з догляду за тваринами (зовнішній ключ до таблиці Owner);

- keeper – помічник з догляду за тваринами, який був збережений у переліку власника (зовнішній ключ до таблиці Keeper).

Публікація статей помічниками може бути частиною привертання уваги, але основою у подібних додатках звичайно є відгуки, для збереження яких було створено таблицю Review. Це можуть бути відгуки двосторонні, тобто такий запис передбачає як відгук власника про помічника, так і помічника про власника. Таблиця має структуру, визначену наступними полями:

- id – ідентифікатор відгуку;

- owner – власник тварин, який створив цей відгук або якого стосується цей відгук (зовнішній ключ до таблиці Owner);

- keeper – помічник з догляду за тваринами, який створив цей відгук або якого стосується цей відгук (зовнішній ключ до таблиці Keeper);

- direction – напрямок відгуку: 0 – відгук власника тварин про помічника з догляду за тваринами, 1 – відгук помічника з догляду за тваринами про власника тварин;

- creationtime – час, коли було створено цей відгук;

- opinion – текст думки, яку було висловлено у відгуку;

- mark – оцінка за 5-бальною шкалою, яка була виставлена у відгуку;

- recommendation – визначення відгуку як рекомендації: використовується тільки для відгуків з найвищою оцінкою, при цьому кількість таких відгуків кожним власником тварин обмежена.

Для помічників з догляду за тваринами дана робота може бути не основною. Також помічник може мати обмежений робочий час, потребувати особистого часу. Усе це призводить до того, що помічник повинен явно вказати, коли він готовий працювати. Дані про це зберігаються в таблиці KeeperFree. Вона має структуру, визначену наступними полями:

- id – ідентифікатор вільного для роботи часу помічника з догляду за тваринами;
- keeper – помічник з догляду за тваринами, якого стосується цей час (зовнішній ключ до таблиці Keeper);
- begintime – час початку роботи (вільного часу для роботи);
- finishtime – час закінчення роботи (вільного часу для роботи);
- boardingpossible – можливість приймання в цей час тварин на своїй території.

Усі тарифи розділені на дві групи. Це тарифи, що стосуються головним чином певної кількості дій, і тарифи, що стосуються певної тривалості дій.

Тарифи, що стосуються певної кількості дій, зберігаються в таблиці ShortTariff. Далі вони називаються кількісними тарифами. Ці тарифи визначають тарифи на відвідування тварин на території власника, тарифи на прогулянки з тваринами. Таблиця має структуру, визначену наступними полями:

- id – ідентифікатор кількісного тарифу;
- keeper – помічник з догляду за тваринами, який визначив цей тариф (зовнішній ключ до таблиці Keeper);
- duration – тривалість часу, за який виконуються ці дії;
- daytimes – кількість таких дій на день: наприклад, кількість прогулянок з собакою або кількість відвідувань kota;
- extrapet – додаткова вартість за виконання дій над ще однією твариною: наприклад, якщо прогулянка має бути з 2 собаками, то передбачається, що за другу доведеться доплатити вказану суму;
- fromdays – мінімальна кількість днів, для якої діє такий тариф: тобто щоб сплачувати саме таку суму, потрібно, щоб бронювання охоплювало хоча би вказану кількість днів поспіль;
- dayprice – вартість послуги на день: наприклад, якщо передбачаються 2 прогулянки на день, то це вартість одразу 2 прогулянок;

– visitwalking – тип дії, якої стосується цей тариф: відвідування або прогулянка.

Друга група тарифів фактично характеризується як часова, тобто потребує певного часу для сплати певної суми. Для цього визначена таблиця KeepingTariff, що має структуру, визначену наступними полями:

- id – ідентифікатор часового тарифу;
- keeper – помічник з догляду за тваринами, який визначив цей тариф (зовнішній ключ до таблиці Keeper);
- begintime – час, з якого діє цей тариф;
- finishtime – час, до якого діє цей тариф: begintime і finishtime визначають інтервал часу протягом дня, коли може бути задіяно цей тариф;
- duration – мінімальна тривалість бронювання, з якої починає діяти цей тариф;
- unit – одиниця виміру (доба, година, хвилина);
- multiples – кількість разів, за якої починає діяти цей тариф: тобто якщо власник тварини виконує бронювання і визначає, що з собакою потрібно посидіти 3 дні поспіль по 1 годині, то буде задіяно такий тариф;
- timeprice – вартість послуги за один раз: тобто для вказаного вище випадку це вартість, яку потрібно сплатити за кожен день, тоді загальна вартість буде дорівнювати даній вартості, помноженій на 3;
- sittingboarding – дія послуги на території власника або на території помічника з догляду за тваринами: використання власного приміщення помічником може призводити до визначення більшої вартості, а тому незважаючи на однакову загальну структуру тарифу потрібно фактично конкретизувати, чи буде використовуватися будинок помічника.

Безпосередньо бронювання відвідувань, прогулянок, догляду за твариною протягом певного періоду часу в себе або на території власника передбачає врахування різних характеристик, не зважаючи на подібність тарифів.

Для даних про бронювання відвідувань визначена таблиця Visitbooking, що має структуру, визначену наступними полями:

- id – ідентифікатор бронювання відвідування помічником з догляду за тваринами тварини;
- keeper – помічник з догляду за тваринами, який має виконати це відвідування (зовнішній ключ до таблиці Keeper);
- group – група тварин (мінімально складається з однієї тварини), яку має відвідати помічник з догляду за тваринами (зовнішній ключ до таблиці Group);
- city – місто, в якому відбувається відвідування (зовнішній ключ до таблиці City);
- address – адреса, яку має відвідати помічник з догляду за тваринами;
- startdate – дата, коли має відбутися перше відвідування;
- enddate – дата, коли має відбутися останнє відвідування;
- daytimes – кількість разів на день, яку має відвідати помічник з догляду за тваринами тварину;
- state – стан бронювання: кожне бронювання обов'язково має бути підтверджено помічником з догляду за тваринами, тобто має бути узгоджено взаємодію не тільки з боку власника тварин;
- createtime – час, коли було створено бронювання;
- payment – інформація про платіж за бронюванням;
- price – вартість бронювання;
- instructions – інструкції зі сторони власника тварин при відвідуванні: можуть міститися потрібні коди чи дії, які необхідно виконати, щоб дістатися тварини у житлі власника.

Бронювання відвідування може включати тільки одне окреме відвідування, а може включати цілу послідовність, наприклад, коли власник поїхав на відпочинок, а помічник піклується про його тварину. Тому кожне бронювання відвідування може включати не менше ніж одне відвідування.

Дані про відвідування знаходяться в таблиці Visit. Вона має структуру, визначену наступними полями:

- id – ідентифікатор відвідування помічником з догляду за тваринами тварини;
- visitbooking – бронювання, якого стосується це відвідування (зовнішній ключ до таблиці Visitbooking);
- visittime – конкретний час, коли потрібно відвідати тварину;
- state – стан відвідування.

Бронювання прогулянки з твариною зберігаються в таблиці Walkingbooking. Вона має структуру, визначену наступними полями:

- id – ідентифікатор бронювання прогулянки з твариною помічника з догляду за тваринами;
- city – місто, в якому відбувається прогулянка (зовнішній ключ до таблиці City);
- beginaddress – адреса, за якою розпочинається і/або завершується прогулянка;
- finishaddress – адреса, за якою розпочинається і/або завершується прогулянка;
- keeper – помічник з догляду за тваринами, який здійснює цю прогулянку (зовнішній ключ до таблиці Keeper);
- group – група тварин (мінімально складається з однієї тварини), з якою має виконати прогулянку помічник з догляду за тваринами (зовнішній ключ до таблиці Group);
- createtime – час, коли було створено бронювання;
- payment – інформація про платіж за бронюванням;
- checkin – час, з якого починається прогулянка;
- checkout – час, коли закінчується прогулянка;
- price – вартість бронювання;
- state – стан бронювання прогулянки;
- instructions – інструкції щодо прогулянки.

Бронювання догляду за твариною протягом певного періоду часу на території помічника зберігаються в таблиці Boardingbooking. Вона має структуру, визначену наступними полями:

- id – ідентифікатор бронювання догляду за твариною на території помічника;
- keeper – помічник з догляду за тваринами, який здійснює цей догляд (зовнішній ключ до таблиці Keeper);
- group – група тварин (мінімально складається з однієї тварини), яку має доглядати помічник з догляду за тваринами (зовнішній ключ до таблиці Group);
- checkin – час, з якого починається догляд;
- checkout – час, коли закінчується догляд;
- createtime – час, коли було створено бронювання;
- payment – інформація про платіж за бронюванням;
- price – вартість бронювання догляду за твариною протягом певного періоду часу на території помічника;
- state – стан бронювання догляду за твариною протягом певного періоду часу на території помічника;
- instructions – інструкції щодо догляду за твариною.

Бронювання догляду за твариною протягом певного періоду часу на території власника зберігаються в таблиці Sittingbooking. Вона має структуру, визначену наступними полями:

- id – ідентифікатор бронювання догляду за твариною на території власника тварини;
- keeper – помічник з догляду за тваринами, який здійснює цей догляд (зовнішній ключ до таблиці Keeper);
- group – група тварин (мінімально складається з однієї тварини), яку має доглядати помічник з догляду за тваринами (зовнішній ключ до таблиці Group);
- city – місто, де відбувається догляд;

- address – адреса, за якою відбувається догляд (адреса власника тварини або якогось іншого запропонованого ним приміщення);
- createtime – час, коли було створено бронювання;
- payment – інформація про платіж за бронюванням;
- checkin – час, з якого починається догляд;
- checkout – час, коли закінчується догляд;
- price – вартість бронювання догляду за твариною протягом певного періоду часу на території власника;
- state – стан бронювання догляду за твариною протягом певного періоду часу на території власника;
- instructions – інструкції щодо догляду за твариною.

Під час відвідувань власник може попросити виконати певні завдання. Наприклад, погодувати тварину, погратися з нею і т.д. Безпосередньо самі процедури визначаються централізовано на рівні програми. Вони не можуть бути розширені самими власниками. Дані про ці процедури з догляду зберігаються в таблиці Assistanceprocedure, до складу якої входить ідентифікатор та назва процедури name.

Дані про визначення конкретних процедур, які застосовуються до конкретного відвідування тварини, зберігаються в таблиці VisitProcedure (рис. 2.4). Вона має структуру, визначену наступними полями:

- id – ідентифікатор процедури, яку потрібно виконати під час відвідування;
- visit – відвідування, якого стосується ця процедура (зовнішній ключ до таблиці Visit);
- fullbooking – необхідність застосувати цю процедуру до кожного відвідування в межах бронювання, до якого це відвідування відноситься, якщо це потрібно;
- assistanceprocedure – конкретна процедура, яка має бути виконана (зовнішній ключ до таблиці Assistanceprocedure).

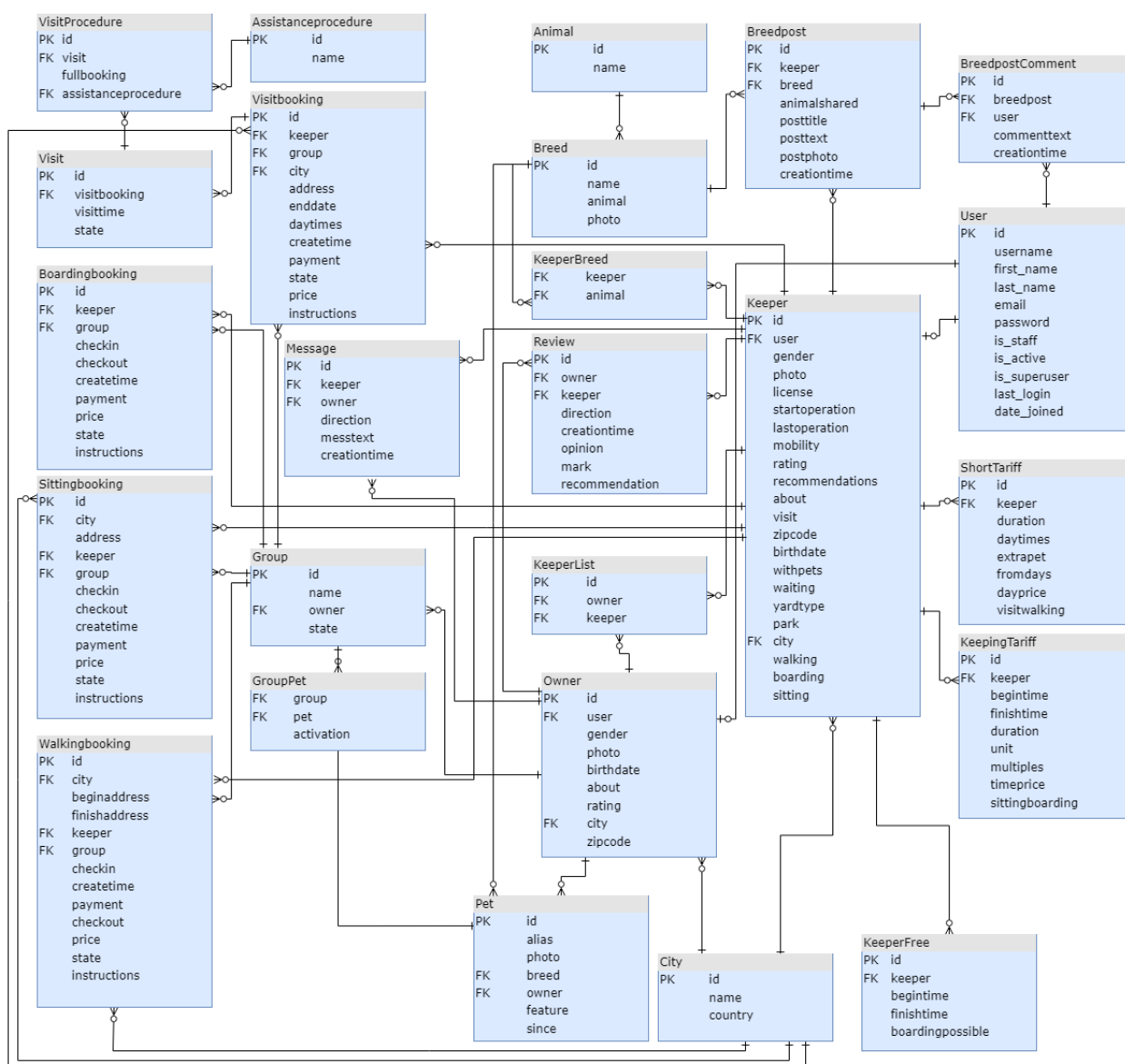


Рисунок 2.4 – Схема БД

Важливо для власників тварин і помічників з догляду за ними обмінюватися повідомленнями, щоб дійти спільного розуміння або просто обмінятися інформацією. Це може здійснюватися і через месенджери або інші засоби, але на початку щонайменше потрібно взаємодіяти через програму. Дані повідомлень містяться в таблиці Message (рис. 2.4). Вона має структуру, визначену наступними полями:

- id – ідентифікатор повідомлення;
- owner – власник тварини, якого стосується або який створив повідомлення (зовнішній ключ до таблиці Owner);

- direction – напрямок повідомлення: 0 – від власника до помічника, 1 – від помічника до власника;
- keeper – помічник з догляду за твариною, якого стосується або який створив повідомлення (зовнішній ключ до таблиці Keeper);
- messtext – текст повідомлення;
- creationtime – час, коли було відправлено повідомлення.

2.4 Висновки за розділом 2

Другий розділ є результатом виконання проєктування програмного забезпечення, включаючи вибір стеку розробки на основі мови Python, фреймворку Django, моделювання сценаріїв роботи користувача, власника тварини, помічника з догляду за тваринами з програмою, проєктування роботи з даними на основі структуризації реляційної БД з визначенням відповідних таблиць і зв'язків між ними.

3 РОЗРОБКА ПРОГРАМИ

3.1 Порядок функціонування програми

Порядок функціонування програми базується на меню, яке залежить від того, чи авторизований користувач, а якщо авторизований, то він є помічником з догляду за тваринами чи власником тварин (рис. 3.1).

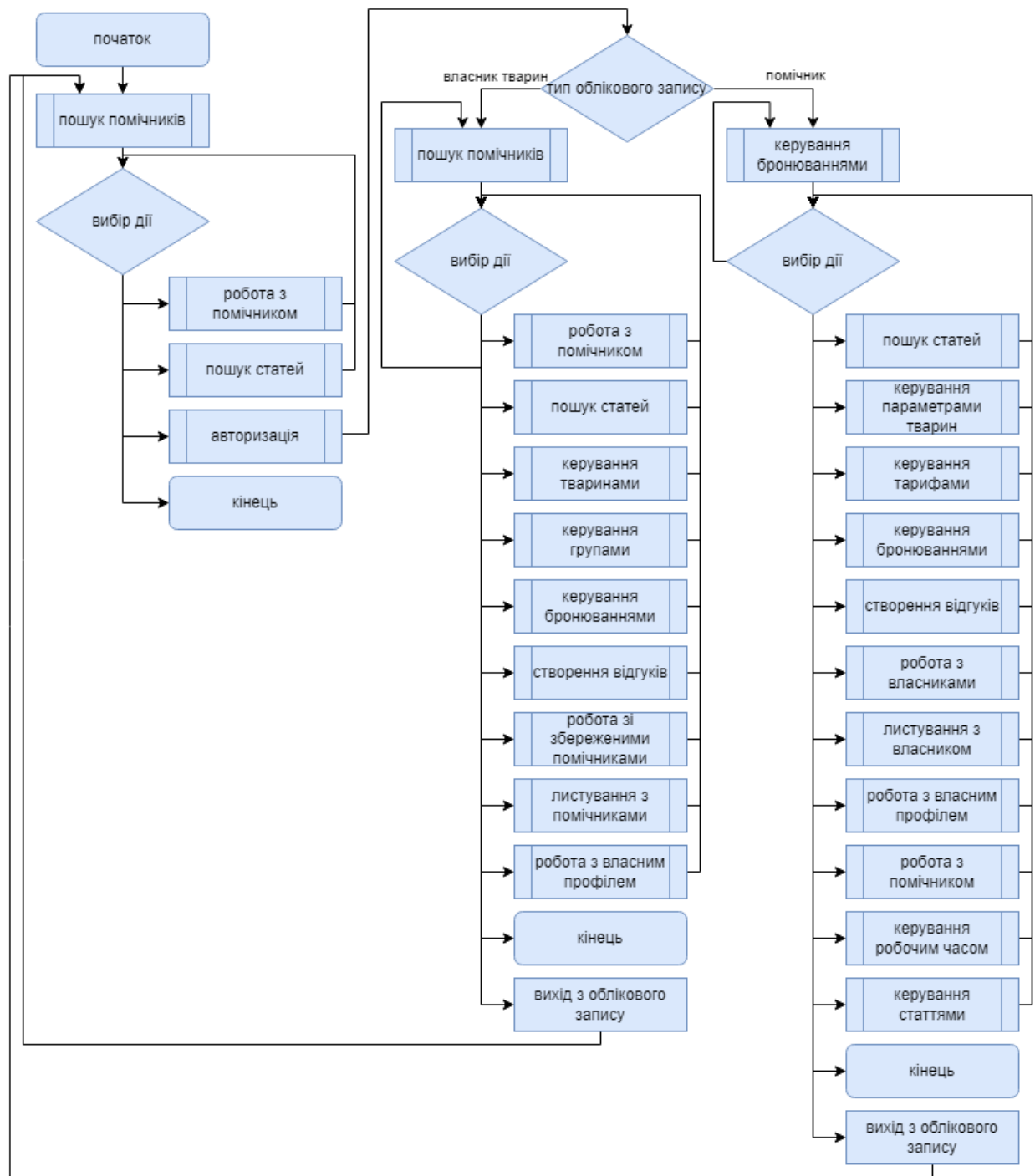


Рисунок 3.1 – Схема функціонування програми

3.2 Рішення з реалізації одиниць додатку

Розробка вебдодатку, базуючись на фреймворку Django, визначила загальну структуру. Відповідно основними складовими були моделі, представлення, шаблони вебсторінок, форми і т.д. Шаблони вебсторінок базуються на розмітці, визначеній шаблоном [12]. Моделі створені у повній відповідності з детально описаною в розділі 2 структурою БД.

Представлення реалізують підготовку всіх необхідних даних для вебсторінок, тому фактично реалізують функції на основі сценаріїв, описаних у розділі 2, представлення. Тому логіка їх побудови відповідає логіці розділення на основі ролей користувачів. За цими ролями представлення розділені на окремі файли: `keeper_views`, `owner_views`, `user_views`. І окремо виділені ті представлення, які доступні декільком ролям користувачів, – `common_views`.

Представлення пошуку помічників з догляду за тваринами реалізовано за допомогою функції `search_keeper(request)`. Усі параметри пошуку передаються через GET-запит і визначаються наступним чином:

- рік, місяць, день, година і хвилина необхідного початку виконання послуги через параметри `startyear`, `startmonth`, `startday`, `starthour`, `startminute` відповідно;

- рік, місяць, день, година і хвилина необхідного завершення виконання послуги через параметри `endyear`, `endmonth`, `startday`, `endhour`, `endminute` відповідно: як було описано в розділі 2, не всі послуги є тривалими, деякі можуть передбачати виконання деякої дії (прогулянка, відвідування) у певний момент часу, а тому ці параметри можуть бути не задані, хоча і для коротких дій (прогулянка, відвідування) також можуть визначатися, щоб вказати на потрібну тривалість, стандартна тривалість дорівнює 30 хвилинам;

- послуга помічника з догляду за тваринами, яка потрібна власнику тварини визначається параметром `service`: значення 0 – відвідування, 1 –

прогулянка, 2 – догляд тварини у приміщенні власника, 3 – догляд тварини на території помічника;

- кількість повторень обслуговування визначається параметром `times`: використовується тільки для відвідувань та прогулянок, тобто не для догляду за часом;

- кількість повторень обслуговування на один день визначається параметром `daytimes`: використовується тільки для відвідувань і прогулянок;

- мінімальна і максимальна допустима для власника тварини вартість догляду визначається параметрами `pricemin` і `pricemax`;

- порода тварини, яку потрібно доглядати, визначається ідентифікатором і встановлюється параметром `type`;

- кількість тварин, догляд яких потрібно виконати, визначається параметром `quantity`;

- місто, в якому потрібно доглядати тварину, і індекс розташування в цьому місті визначаються параметром `city`, який представляє ідентифікатор міста, і `zipcode` відповідно;

- напрямок сортування результатів визначається через параметр `order`.

Після встановлення цих значень виконується відбір помічників, які доступні за вказаними параметрами. Для цього спочатку виконується зміна тривалості догляду, додаючи до початку і до кінця по 10 хвилин. Це необхідно для визначення паузи на те, щоб дістатися між адресами, між подіями. Тобто якщо виявляється, що цей період часу хоч в якійсь частині у помічника зайнятий, то цей помічник не включається в результат.

Виконавши цю підготовку, за кожним помічником спочатку перевіряється, чи відкритий у нього на цей період робочий час. Реалізується через звернення до об'єктів моделі `KeeperFree`, встановлюючи фільтр з визначенням тривалості події, а також перевіряючи одразу, що порода тварин ця може обслуговуватися цим помічником, а він знаходиться в потрібному місті. Це базова перевірка, вона в подальшому розширюється і уточнюється. Так в подальшому перевіряється, що помічник або знаходиться за тим же

самим індексом, що потрібен у пошуку, або що він є мобільним, тобто може переїхати по місту, або що відстань між індексами є не більшою ніж 700 метрів, що вважається достатнім для подолання за 10 хвилин.

Окрім того виконується перевірка вартості догляду. Для цього спочатку перевіряється, яка саме послуга планується до бронювання.

Якщо це відвідування або прогулянка, то використовуються тарифи на основі моделі ShortTariff. Виконується на основі фільтрування пошук таких тарифів, які в даного помічника включають не більшу кількість разів, ніж задано в параметрі times, на день повторюється догляд day_times разів, кількість додаткових тварин дорівнює заданій під час пошуку кількості, зменшеній на 1, а також відповідає саме прогулянці або відвідуванню, як це задано в пошуку. Зі всіх заданих тарифів виконується їх сортування за зменшенням кількості разів повторення, а найбільший з них обирається для підрахунків. Денна вартість, вказана в даному тарифі, множиться на кількість повторень, а отримане значення перевіряється на те, що воно не виходить за заданий інтервал вартості. Якщо відповідність повна, то цей помічник вноситься в результуючий перелік. Усе це повторюється для кожного помічника, для якого попередні перевірки дали позитивну відповідь.

Якщо це тривалий догляд, то виконується перевірка вартості за іншим алгоритмом, адже такий варіант передбачає більше особливих випадків. Ці випадки включають добові, погодинні та похвилинні тарифи. Для кожного варіанта з цих тарифів визначається вартість послуги для кожного помічника. Для кожного помічника в результаті обирається найменше значення з розрахованих для цих тарифів у підсумку. Вже воно перевіряється на входження в заданий інтервал, а якщо входить, то включається в результат. Тобто фактично виконуються розрахунки для кожного тарифу, який може бути використаний, а далі

Для використання добових тарифів обчислюється кількість діб, яку має тривати догляд. Якщо час початку або закінчення виходить хоч на хвилину за межі часу, зазначеного за тарифом, то додається додаткова доба.

Для використання погодинних тарифів застосовується принцип розбиття, коли вся тривалість догляду розбивається на частини у відповідності з наявними тарифами. Тарифи задаються помічником обов'язково такими, щоб вони не перетинались. Тобто часові проміжки між двома тарифами для одного й того самого помічника не можуть перетинатися. Це дозволяє спочатку сформувати перелік з доступних у тарифах інтервалів часу, а потім для кожного такого інтервалу переглянути всі доступні тарифи і обчислити вартість обслуговування за даним проміжком. У підсумку обирається найменше значення для кожного інтервалу і розраховується сума значень. Таким чином отримується вартість обслуговування за погодинними тарифами.

Для похвилинних тарифів знову ж таки визначаються частини, а для кожної частини визначається перетин заданого в пошук часового інтервалу з інтервалом тарифу. Отриманий перетин перетворюється на кількість хвилин у цьому перетині. Ця кількість хвилин перетворюється на вартість за кожним тарифом. У підсумку подібно до погодинних тарифів знаходиться сума з найменших вартостей за всіма інтервалами.

Найменше значення з 3 варіантів порівнюється з інтервалом вартості і якщо відповідає, то заноситься в результуючий перелік. Після цього результуючий перелік упорядковується за заданим напрямком: 0 – за рейтингом, 1 – за зменшенням вартості, 2 – за збільшенням вартості.

Подібно до даного пошуку організовано пошук статей про догляд за тваринами через функцію `search_breedpost`.

Представлення відображення статті з догляду за тваринами підтримується функцією `breedpost_detalize`, яка окрім традиційного параметру `request` має також параметр `id`, що визначає ідентифікатор статті. Але дане представлення підтримує роботу не тільки з відображенням самої статті, але і з відображенням переліку коментарів, а також створенням нових коментарів. Для кожного коментаря при створенні програмно вноситься автор за авторизованих на даний момент записом. Якщо ж користувач не

авторизований або некоректно виконано звернення до сторінки, то форма для внесення коментаря не створюється взагалі. Якщо використано метод звернення GET, то повертається порожня форма для внесення коментаря `BreedpostCommentFrm`, але при цьому надається перелік коментарів та об'єкт статті про догляд за тваринами.

Функція `keeper_profile` підтримує роботу представлення відображення профілю помічника з догляду за тваринами. Функція має в якості параметра ідентифікатор помічника `id`. Якщо значення не задано, то передбачається, що представлення використовується для самого помічника, а тому використовуються дані про його авторизований обліковий запис через параметр `request` і витягання з нього даних запису як `request.user`. Для виділеного таким чином помічника визначаються тарифи на прогулянки і відвідування окремо, окремо тарифи на догляд протягом періоду як у цього помічника, так і у власника тварини. Також виділяються 3 останні опубліковані статті цього помічника та 3 останні опубліковані відгуки про цього помічника. Для відображення всіх відгуків, всіх статей помічника реалізовані окремі функції представлень `keeper_reviews` і `keeper_breedposts`.

Функція `add_review` реалізує представлення створення відгуку. Оскільки це передбачає внесення даних, то спосіб організації в основі подібний до створення коментаря. Проте це представлення відрізняється деякими особливостями. Зокрема функція підтримує створення відгуків і помічником про власника, і власником про помічника. Ідентифікатор користувача, про який створюється відгук, передається через параметр `id`. Дані запису автора визначаються через дані про його авторизацію. У функції визначається, ким може бути авторизований користувач, і кому відповідає заданий ідентифікатор. Так встановлюється напрямок, який відзначається у відгуку і який дозволяє створювати за допомогою цієї функції і відгуки власників тварин про помічників з догляду за тваринами, і навпаки. Відгук не просто створюється, а виконується перевірка, що цей користувач має можливість створити відгук про іншого користувача. Для цього вибираються

всі бронювання, у яких брали участь дані користувачі, а стан їх відповідає повністю виконаним. З них обираються останні за часом публікації. Також виділяється останній опублікований відгук між цими двома користувачами у необхідному напрямку. Якщо виявляється, що бронювання було завершено пізніше ніж був створений останній такий відгук, то виконується створення відгуку або надається форма для створення відгуку.

Усі бронювання знаходяться в одному зі станів: 0 – створено і сплачено, 1 – виконано (підтверджено власником), 2 – виконано (підтверджено помічником з догляду), 3 – виконано і підтверджено обома сторонами, 4 – скасовано.

Підтвердження бронювання помічником з догляду виконується на основі функції представлення `submit_booking`. Вона в якості параметра має ідентифікатор `id` з номером бронювання, тип бронювання `type`. У залежності від заданого типу виділяється відповідний об'єкт на основі потрібної моделі. Якщо стан бронювання дорівнює 0, то воно переводиться до стану 2, а якщо дорівнює 1, то переводиться до стану 3. Подібно організовано представлення для власника тварини, а також функції скасування бронювання.

За подібною на створення відгуку логікою визначено представлення відтворення переліку користувачів, яким писав авторизований користувач. Для цього реалізовано функцію `chattered`.

Ці функції для підтримки представлень описують основні рішення, які застосовувались.

3.3 Висновки за розділом 3

Програмне забезпечення допомоги для догляду за тваринами повністю реалізовано у відповідності з функціональними вимогами і всіма іншими вимогами, визначеними технічним завданням: використано фреймворк Django, реалізовано структуру вебдодатку за заданими сценаріями роботи користувачів усіх ролей.

4 ЕКСПЛУАТАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМИ

4.1 Опис програми

Розроблена програма рівноцінно призначена для використання власниками тварин і помічниками з догляду за тваринами. Власники тварин мають можливість за допомогою програми знаходити помічників, бронювати обслуговування, керувати власними бронюваннями. Помічники з догляду за тваринами можуть керувати бронюваннями від власників, визначати свій робочий графік, створювати статті про догляд за тваринами, відслідковувати власну успішність на основі відгуків щодо роботи з різними породами тварин.

Сервер повинен підтримувати наступні мінімальні апаратні вимоги: процесор не менше ніж Xeon сімейства 3000 або Core 2 Duo, оперативна пам'ять обсягом у 4 Гб, простір на жорсткому диску з доступним вільним обсягом не менше ніж 3 Гб, але для можливості збереження файлів зображень бажано не менше 20 Гб, мережна карта і підключення до мереж Інтернет.

4.2 Експлуатація програми

Програма має універсальну систему меню, яка структурується у відповідності з тим, хто саме взаємодіє з нею. Існує 3 типи користувачів для програми. Якщо користувач зайшов на вебсайт, ввівши базову адресу вебсервера, і не виконав авторизацію, то йому пропонується мінімальне меню, що включає тільки можливості пошуку помічників з догляду (пункт «Пошук») і пошуку статей (пункт «Статті»). Авторизація дозволяє користувачу перейти до іншого типу взаємодії.

Пошук помічника буде описаний нижче для власника тварин. Після вибору одного з результатів користувач опиняється на сторінці помічника з догляду за тваринами (рис. 4.1).

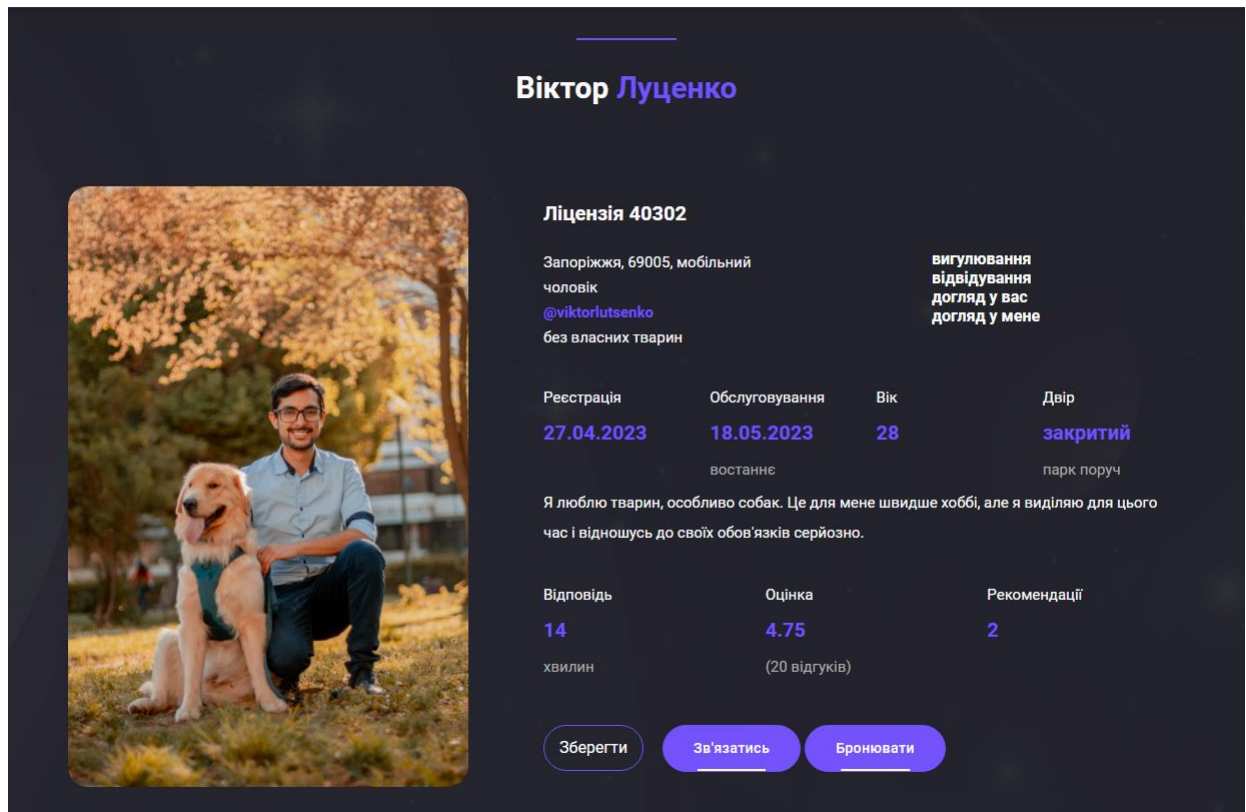


Рисунок 4.1 – Верхня частина сторінки помічника з догляду за тваринами

На цій сторінці міститься вся інформація про даного помічника. Зокрема це його повне ім'я, його користувацький запис, номер ліцензії, якщо така є, місто, індекс, інформація про те, чи може він переміщатися транспортом (це дозволяє пропонувати його послуги всім власникам у межах всього міста), стать, інформація про те, чи має він власних тварин, послуги з догляду тварин, які він надає (правий стовпчик), дата його реєстрації на вебсайті, дата останнього обслуговування, яке він виконав, вік, тип двору, який в нього вдома та наявність парку поруч з будинком. Окремо наводиться його інформація про себе, а нижче також середній час, який доводиться очікувати його відповіді, середня оцінка за відгуками, кількість рекомендацій. Кнопка «Зберегти» дозволяє зберегти помічника до власного повторного перегляду. До авторизації вона є неактивною. Щоб написати помічнику, потрібно натиснути на кнопку «Зв'язатись». Кнопка «Бронювати» дозволяє перейти до сторінки бронювання, де потрібно вказати повністю дані

про бронювання послуги з догляду, зокрема включаючи групу тварин, для яких виконується бронювання. Тому до авторизації виконати бронювання неможливо.

Нижче на сторінці знаходиться перелік з 3 останніх статей, які вніс на вебсайт помічник (рис. 4.2), або меншої кількості, якщо помічник ще не створив достатню кількість. Кнопка «Статті» над цим переліком дозволяє перейти до сторінки перегляду всіх статей помічника. Ще нижче знаходиться перелік відгуків про помічника. Також приводяться 3 останні, а кнопка «Відгуки» дозволяє перейти до сторінки зі всіма відгуками. Кожен відгук окрім оцінки містить дату публікації, натискаючи як на неї, так і на оцінку, можна відобразити у спливаючому віконці тексту відгуку.

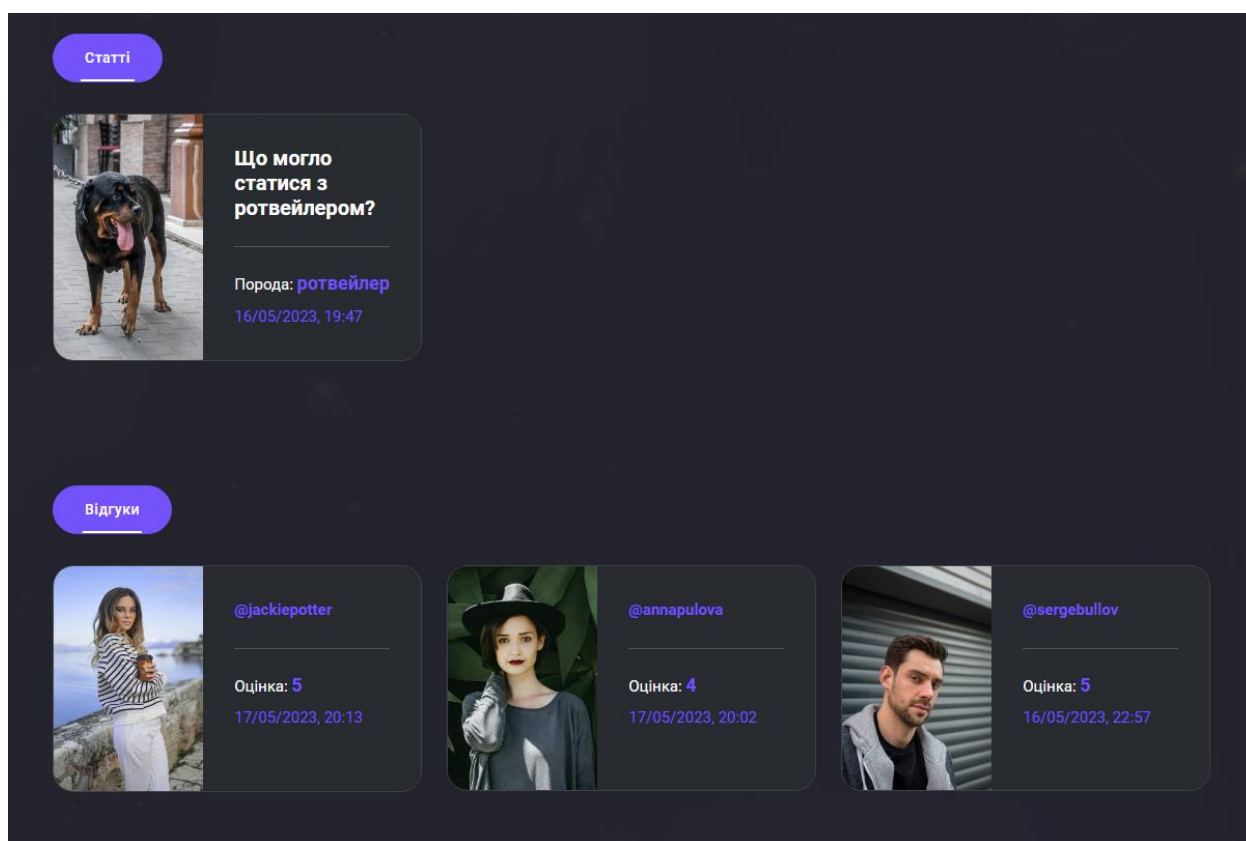


Рисунок 4.2 – Друга частина сторінки помічника з догляду за тваринами

На сторінці також є повний перелік усіх тарифів на відвідування та вихулювання тварин даного помічника, а ще нижче – перелік тарифів на

догляд на території власника тварини або на території помічника. Останню частину сторінки займає перелік порід тварин, з якими працює помічник.

Пошук статей включає текст з назви або тексту статті, період публікації та породу тварини, про яку має бути стаття.

Власники тварин мають розширене і дещо переструктуроване меню. Пункт меню «Пошук» одразу дозволяє виконувати пошук помічників з догляду за тваринами (рис. 4.3), хоча у першому рядку містить кнопку для переходу до пошуку статей.

Пошук Помічника Для Догляду За Твариною

Тип догляду: догляд у власника (dropdown), Обрати

Дата початку: 23.05.2023 (calendar icon), Дата завершення: 25.05.2023 (calendar icon), Місто: Дніпро (dropdown)

Час початку: 14:00 (clock icon), Час завершення: 14:00 (clock icon), Індекс: 49000

Мінімальна вартість: 0, Максимальна вартість: 1000

Вид тварини: Собака (dropdown), Порода тварини: Лабрадор-ретривер (dropdown), Додати

Кількість: 1 (dropdown)

Додані породи: ротвейлер (собака)

Шукати

Рисунок 4.3 – Пошук помічників з догляду за тваринами

Форма пошуку залежить від обраного згори типу догляду. Це може змінювати склад полів у формі. За тривалого догляду потрібно вказати дату і час початку та завершення догляду. Можна обмежити вартість догляду. Вказується обов'язково місто пошуку та індекс, де розташовується тварина. За твариною потрібно також зазначити її породу, а якщо їх буде декілька, то вказати кількість і додати всі породи, які є в цій групі. Результати пошуку (рис. 4.4) розташовані нижче на сторінці. Вони містять дані за кожним помічником. Це основні дані, а перейти до повної сторінки можна за допомогою кнопки «Детально». Для кожного помічника вказується вартість догляду за вказаними параметрами пошуку в його виконанні. Кнопка «Бронювати» дозволяє одразу перейти до бронювання послуги за вказаними параметрами.

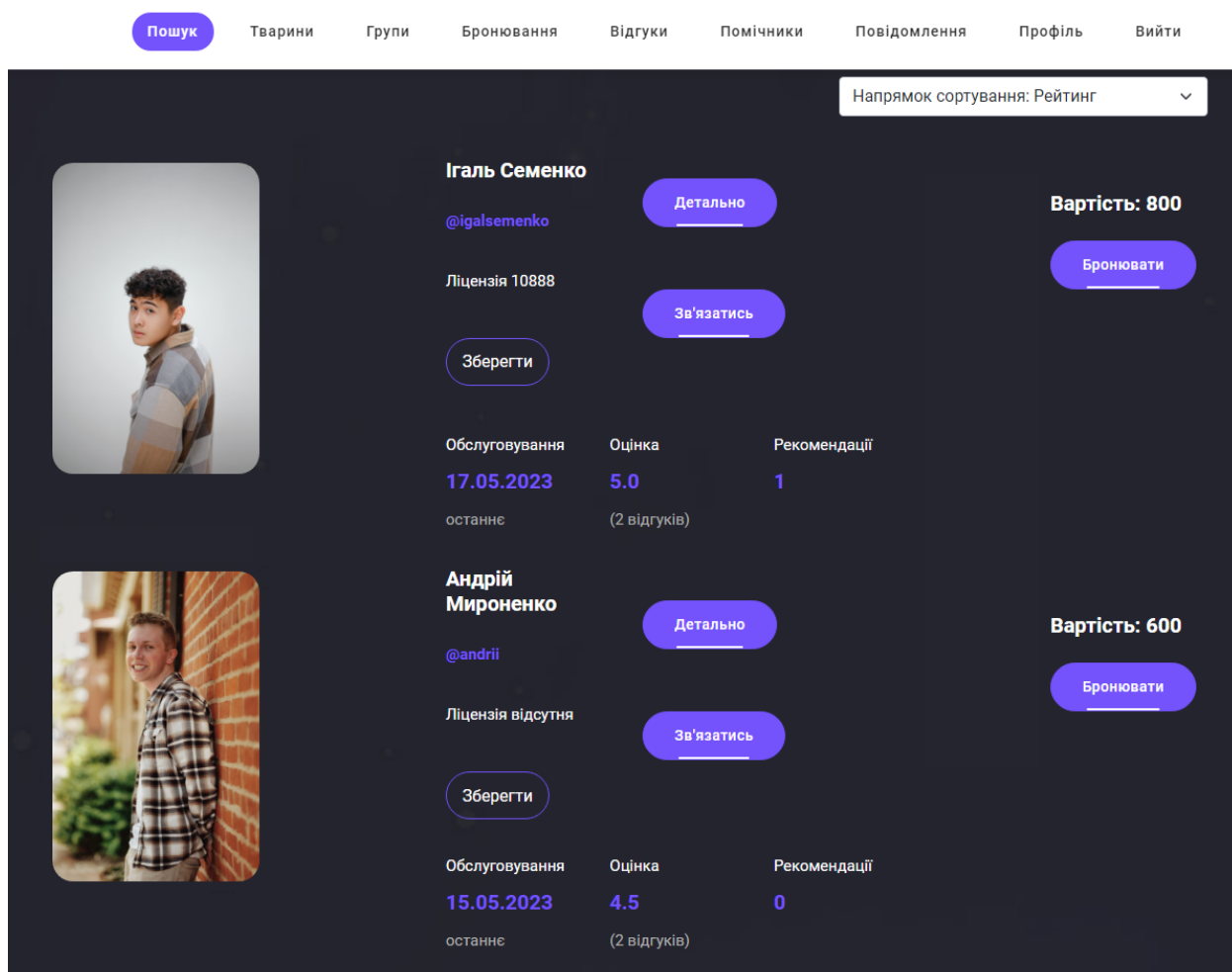


Рисунок 4.4 – Результати пошуку помічників з догляду за тваринами

Пункт меню «Бронювання» дозволяє переглядати всі власні бронювання власника, при цьому за допомогою кнопок згори визначаючи, чи це бронювання стосовно власної тварини (за замовчуванням), чи це бронювання стосовно всіх тварин, які знаходяться в групах, якими керує власник. Визначення власних тварин відбувається через пункт меню «Тварини», а визначення груп відбувається через пункт меню «Групи».

Отримати перелік помічників, про які можна написати відгук, можна через пункт «Відгуки». Пункт «Помічники» дозволяє перейти до переліку збережених через кнопку «Зберегти» помічників з догляду. Звідти їх можна також вилучити. Відкрити перелік помічників, яким писав власник, можна через пункт «Повідомлення». Обираючи помічника, можна перейти до переписки з ним.

Для помічника з догляду доступні дещо інші пункти меню, що включають «Пошук», аналогічний власнику тварин, «Статті», звідки можна бачити власні статті та редагувати їх за потреби, «Відгуки», звідки за аналогією з власником можна створювати відгуки, але вже про власників. Пункт «Власники» дозволяє переглядати власників, з якими взаємодівав помічник з догляду.

Пункт «Бронювання» дозволяє керувати бронюваннями послуг авторизованого помічника з догляду за тваринами. У верхньому рядку доступний варіант роботи з бронюваннями за допомогою відповідних кнопок. Кнопка «Усі» дозволяє переглянути всі бронювання. Кнопка «Дата» дозволяє при натисканні на неї вказати конкретну дату, за якою цікавлять бронювання помічника. Кнопка «Власник» дозволяє вибрати власника з тих, з якими має бронювання помічник, та відобразити тільки бронювання цим власником. Після цього вибору нижче доступний вибір за допомогою кнопок або нових бронювань, які ще не було виконано (кнопка «Нові»), або бронювань, які вже було реалізовано (кнопка «Архіви»). Також нижче можна за допомогою кнопок обрати, чи відображаються всі відповідні бронювання, чи тільки відповідний тип (прогулянки, відвідування, догляд у власника,

догляду у помічника). Пункт «Тварини» містить перелік всіх порід тварин, з якими працює помічник. Помічник може додавати нові породи або вилучати їх. За допомогою кнопки «Успішність» помічник може відобразити сторінку, на якій після вибору виду тварин відображаються всі породи цього виду тварин, яких стосувалися бронювання у цього помічника (рис. 4.5).

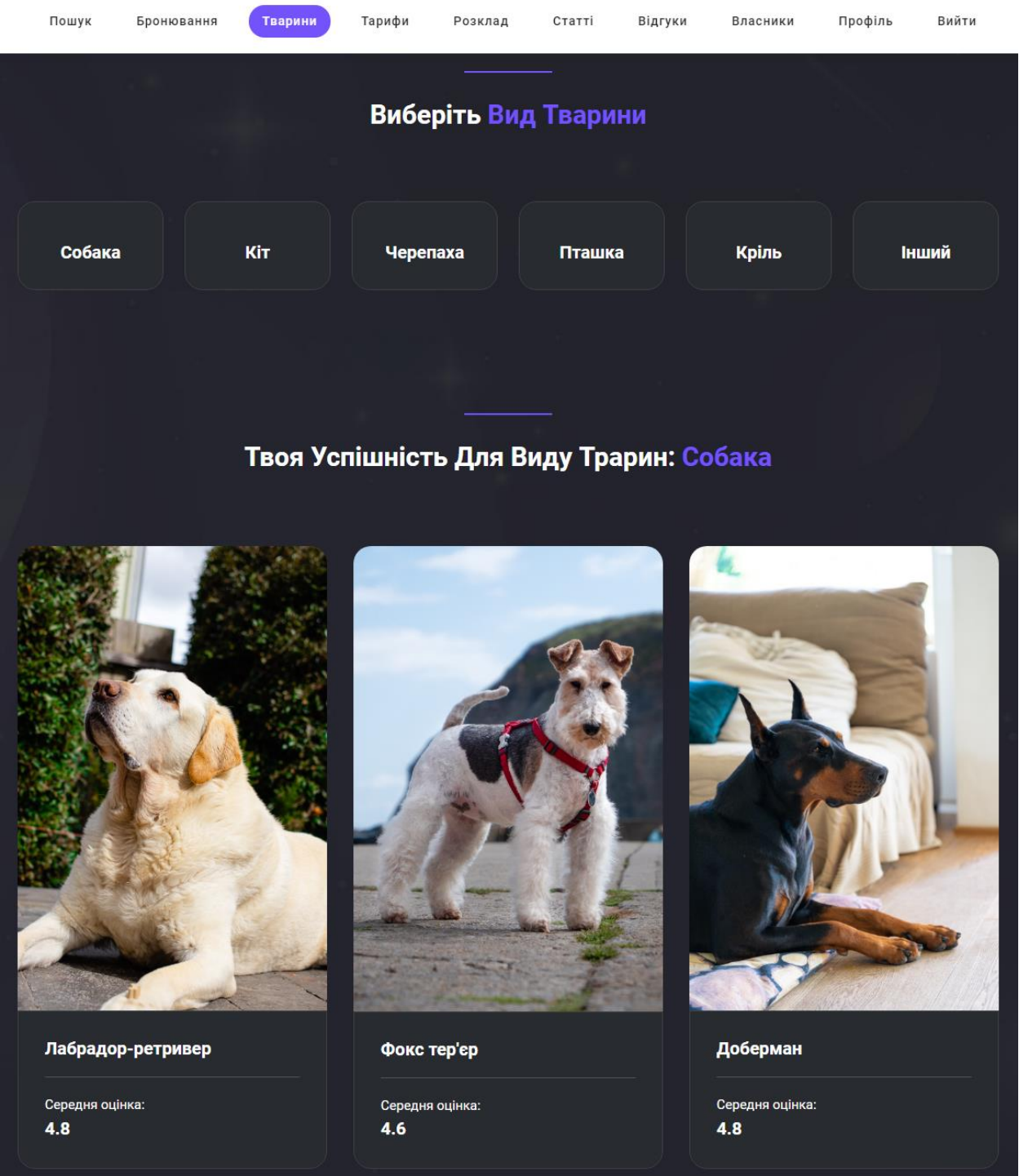


Рисунок 4.5 – Сторінка успішності помічника з різними породами тварин

За кожною породою відображається середній бал з відгуків, створених після цих бронювань. Так помічник може вирішити перестати працювати з певною породою.

Пункти «Тарифи» і «Розклад» дозволяють керувати власними тарифами і доступним робочим часом.

4.3 Тестування програми

Для перевірки працездатності програми було у підсумку створено таблицю з переліком всіх функцій, які мала за вимогами реалізувати програма, та виконано перевірку коректності їхньої роботи (табл. 4.1), що виявилась успішною для всіх функцій.

Таблиця 4.1 – Тестування програми

Функція	Корис- тувач	Власник тварини	Помічник з догляду
1	2	3	4
Пошук помічника для бронювання його послуг	+	+	+
Відображення профілю помічника з догляду	+	+	+
Відображення профілю власника тварини	+	+	+
Відображення статей помічника з догляду	+	+	+
Пошук статей про догляд за тваринами	+	+	+
Відображення відгуків про помічника з догляду	+	+	+
Авторизація	+		
Реєстрація	+		

Продовження таблиці 4.1

1	2	3	4
Бронювання прогулянки тварини з помічником з догляду		+	
Бронювання відвідування помічником тварини		+	
Бронювання догляду тварини помічником на території помічника		+	
Бронювання догляду тварини помічником у приміщенні власника		+	
Відображення особистого відкритого для роботи часу			+
Додавання особистого відкритого для роботи часу			+
Закриття особистого відкритого для роботи часу			+
Відображення доступних для відгуку власників			+
Відображення доступних для відгуку помічників з догляду		+	
Створення відгуку		+	+
Створення рекомендації		+	
Відображення тварин власника	+	+	+
Додавання тварини власника		+	
Вилучення тварини від власника		+	
Редагування даних тварини власника		+	
Створення групи тварин		+	
Додавання тварини до групи		+	
Вилучення тварини з групи		+	

Продовження таблиці 4.1

1	2	3	4
Підтвердження додавання тварини до групи		+	
Відображення груп, до яких внесено тварину власника		+	
Відображення груп, якими керує власник		+	
Вилучення групи тварин		+	
Коментування статті		+	+
Визначення успішності помічника з різними видами і породами тварин			+
Відображення бронювань за власною твариною		+	
Відображення бронювань за всіма тваринами, з якими пов'язаний користувач		+	
Відображення всіх бронювань у помічника з догляду			+
Скасування бронювання		+	+
Підтвердження бронювання		+	+
Створення статті про догляд за тваринами			+
Редагування статті про догляд за тваринами			+
Відображення всіх користувачів, яким писав користувач		+	+
Відправлення повідомлення		+	+
Читання повідомлень		+	+
Внесення до переліку власника помічника		+	
Вилучення з переліку власника помічника		+	
Відображення помічників з переліку		+	

Кінець таблиці 4.1

1	2	3	4
Відображення статті про догляд за тваринами	+	+	+
Відображення видів і порід тварин, з якими працює помічник			+
Додавання порід тварин, з якими працює помічник			+
Вилучення породи тварин, з якими працює помічник			+
Додавання тарифу на послуги помічника			+
Відображення помічників, послуги яких бронював власник тварини		+	
Редагування тарифу на послуги помічника			+
Відображення власників тварин, які бронювали послуги помічника			+
Відображення бронювань послуг помічника власником тварин		+	+
Відображення всіх бронювань помічника з догляду за датою			+

4.4 Висновки за розділом 4

Програму реалізовано повністю: усі заплановані функції як для помічника з догляду за тваринами, так і для власника тварин успішно реалізовано.

ВИСНОВКИ

Розробка програмного забезпечення допомоги для догляду за тваринами для надання можливості взаємовигідної взаємодії між власниками тварин і помічниками з догляду за тваринами була визначена метою дипломної кваліфікаційної роботи і була досягнута в роботі.

В основу виконаної розробки було покладено аналіз ПО, що дозволило встановити необхідні вимоги до програми, виділити потрібні особливості, що покращують зручність сторін у програмі. Це забезпечило ресурс для моделювання програми, проєктування її структури, реалізації програми. Для розробки було сукупно використано мови Python, HTML, CSS, JavaScript, фреймворк Django, систему керування базами даних PostgreSQL.

Результати роботи включають програмне забезпечення, розроблене в підсумку, та комплекс результатів окремих етапів, що в підсумку дозволили створити вебдодаток, який дозволяє власникам тварин шукати помічників з догляду як для виховування тварин, коли вони не здатні це зробити, так і для відвідування тварин у момент їх відсутності вдома, а також для реалізації тривалого догляду на території власника або помічника, а помічникам дозволяє виставляти власні пропозиції, визначаючи гнучко доступний робочий час, тарифи на обслуговування, створюючи статті про догляд за тваринами, так надаючи власникам тварин додаткову інформацію про свою обізнаність у процесах догляду, аналізуючи ефективність власної роботи з різними породами тварин на основі відгуків власників тварин.

Визначений комплекс етапів і процедур дозволив розробити програму, підсумкове тестування якої встановило відповідність вимогам загалом і функціональним вимогам до програми зокрема, визначеним на початкових етапах.

Програма придатна для використання як окрема платформа, на якій беруть участь власники домашніх тварин та люди, які готові працювати в ролі помічників з догляду за тваринами.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Haurant, S. Dog daycare [Electronic resource] : who will look after your pet when you go in to work? / S. Haurant. – Access mode : <https://www.theguardian.com/lifeandstyle/2021/jun/12/dog-daycare-who-will-care-for-your-pet-when-you-go-in-to-work>.
2. Україна увійшла в топ-10 країн з найбільшою кількістю домашніх котів [Електрон. ресурс]. – Режим доступу : https://lb.ua/society/2016/03/21/330809_ukraine_voshla_top10_stran.html.
3. Rover.com [Electronic resource] : Book Dog Boarding, Dog Walking and More. – Access mode : <https://www.rover.com/>.
4. #1 Accredited Pet Sitting & Dog Boarding + Minding. PetCloud [Electronic resource]. – Access mode : <https://www.petcloud.com.au/>.
5. Affordable Dog sitters, Catsitters and Animal Care – Petsitting24 [Electronic resource]. – Access mode : <https://petsitting24.ch/>.
6. Wag! Find Best Local Dog Walkers, Boarders, and Trainers [Electronic resource]. – Access mode : <https://wagwalking.com/>.
7. Bernard, J. Python Recipes Handbook [Text] : A Problem-Solution Approach / J. Bernard. – New York : Apress, 2016. – 155 p.
8. Nguyen, Q. Advanced Python Programming [Text] : Accelerate your Python programs using proven techniques and design patterns / Wuan Nguyen. – Birmingham : Packt Publishing, 2022. – 606 p.
9. Lubanovic, B. Introducing Python [Text] : Modern Computing in Simple Packages / B. Lubanovic. – Sebastopol, USA : O'Reilly Media, 2019. – 630 p.
10. Web Development with Django [Text] : Learn to build modern web applications with a Python-based framework / Ben Shaw, Saurabh Badhwar, Andrew Bird et al. – Birmingham : Packt Publishing, 2021. – 826 p.
11. Chan, J. Python API Development Fundamentals [Text] : Develop a full-stack web application with Python and Flask / Jack Chan, Ray Chung, Jack

Huang. – Birmingham : Packt Publishing, 2019. – 372 p.

12. Liberty Market Template [Electronic resource]. – Access mode :
<https://templatemo.com/tm-577-liberty-market>

ДОДАТОК А
Технічне завдання

Вступ

Тварини потребують постійного догляду. Кожен день потрібно їх годувати, з певною періодичністю вигулювати, прибирати за ними. Однак власники можуть захотіти поїхати на відпочинок, мати робоче відраядження. Щонайменше не завжди вони можуть взяти з собою домашніх тварин. Тому вони в такий момент потребують знайти людину, яка здатна доглядати за їхніми тваринами. Люди за певну плату готові допомогти іншим людям, а тому важливо дозволити їм знайти один одного, при цьому надаючи можливість отримати інформацію один про одного, яка визначається тими людьми, які вже скористалися їхніми послугами. За наявності такого зручного програмного забезпечення ці користувачі зможуть взаємодіяти і допоможуть один одному вирішити достатньо буденну, але це одночасно значить і важливу проблему.

A.1 Підстави для розробки

Завдання на дипломну кваліфікаційну роботу сформульовано у відповідності з наказом № 87 від 4 квітня 2023 р. за Національним університетом «Запорізька політехніка», за яким було визначено тему «Програмне забезпечення допомоги для догляду за тваринами».

A.2 Призначення розробки

Програма рівноцінно призначена для використання власниками тварин і помічниками з догляду за тваринами. Власники тварин повинні мати можливість за допомогою програми знаходити помічників, бронювати обслуговування, керувати власними бронюваннями. Помічники з догляду за тваринами можуть керувати бронюваннями від власників, визначати свій робочий графік, створювати статті про догляд за тваринами.

А.3 Основні вимоги до програми

А.3.1 Вимоги до функціональних характеристик

Забезпечення призначення програми виконується через реалізацію сценаріїв використання програми на основі наступних функцій:

- бронювання прогулянки тварини з помічником з догляду;
- бронювання відвідування помічником тварини;
- бронювання догляду помічником тварини у власника;
- бронювання догляду тварини помічником;
- визначення відкритого для роботи часу помічником з догляду;
- авторизація і реєстрація;
- створення відгуку власником про помічника з догляду за тваринами;
- створення відгуку помічника з догляду за тваринами про власника;
- визначення тварин власника;
- визначення груп тварин;
- пошук помічника для бронювання його послуг;
- пошук статей про догляд за тваринами;
- визначення тарифів на послуги помічника;
- визначення видів тварин, з якими працює помічник;
- створення, редагування статей про тварин;
- коментування статті;
- визначення успішності помічника з різними видами і породами тварин;
- внесення до переліку власника помічника або вилучення його;
- відображення статей помічника з догляду;
- відображення профілю помічника з догляду;
- відображення відгуків про помічника з догляду;
- відображення профілю власника тварини;
- відображення бронювань за власною твариною;

- відображення бронювань за всіма тваринами, з якими пов'язаний користувач;
- відображення помічників, послуги яких бронював власник тварини;
- відображення власників тварин, які бронювали послуги помічника;
- відображення бронювань послуг помічника власником тварин;
- відправлення і читання повідомлень;
- відображення всіх помічників, яким писав власник тварини;
- відображення всіх власників тварин, яким писав помічник з догляду;
- відображення всіх запитів на бронювання до помічника з догляду;
- відображення всіх обслуговувань у помічника з догляду;
- відображення всіх обслуговувань і запитів на бронювання до помічника з догляду за датою;
- відображення груп, до яких внесено тварину власника.

A.3.2 Умови експлуатації

Експлуатація програми на сервері потребує підтримки, що включає навички роботи з БД, зокрема PostgreSQL, виконання міграцій для Django-додатків та внесення даних через адміністративний інтерфейс при необхідності розширення або зміни каталогів стандартних даних.

A.3.3 Вимоги до складу та параметрів технічних засобів

Набір мінімальних технічних засобів, що мають забезпечуватися на сервері для підтримки роботи вебдодатку, включає:

- процесор не менше ніж Xeon сімейства 3000 або Core 2 Duo;
- оперативна пам'ять обсягом у 4 Гб;
- простір на жорсткому диску з доступним вільним обсягом не менше ніж 3 Гб, але для можливості збереження файлів зображень бажано не менше 20 Гб, мережна карта і підключення до мереж Інтернет.

А.4 Порядок контролю та приймання

Для виконання дипломної кваліфікаційної роботи критичним є виконання визначеного завдання, що включає безпосередньо завдання на роботу і технічне завдання, за темою, яка зафіксована наказом, у строки, які визначені завданням на роботу.

У підсумку виконується захист роботи за оприлюдненим наперед графіком перед екзаменаційною комісією.

ДОДАТОК Б
Текст програми

Б.1 Текст файла keeper_views.py

```

from .models import Review, Visitbooking, Walkingbooking,
Boardingbooking, Sittingbooking, Keeper, KeeperFree, Breedpost, Animal
from django.shortcuts import render
from pethelperapp.forms import ShortTariffFrm, KeepingTariffFrm,
KeeperFreeFrm, KeeperbreedFrm, BreedpostFrm
from pethelperapp.payments import CancelPayment
from django.db.models import Q
from django.shortcuts import get_object_or_404

def for_reviewing(request):
    keeper = Keeper.object.get(user = request.user, None)
    if not keeper:
        return render(request,
"pethelperapp/for_reviewing.html", {"bookings": [] })
        visitbookings = Visitbooking.objects.filter(keeper =
keeper).order_by('-createtime')
        walkingbookings = Walkingbooking.objects.filter(keeper =
keeper).order_by('-createtime')
        boardingbookings = Boardingbooking.objects.filter(keeper =
keeper).order_by('-createtime')
        sittingbookings = Sittingbooking.objects.filter(keeper =
keeper).order_by('-createtime')
        reviews = Review.objects.filter(keeper = keeper, direction =
1)

        owners = []
        owners_analysed = []
        bookings = []
        for visitbooking in visitbookings:
            if visitbooking.group.owner not in owners_analysed:
                owners_analysed.append(visitbooking.group.owner)
                if visitbooking.group.owner not in owners:
                    keeper_review = reviews.filter(owner =
visitbooking.group.owner).order_by('-createtime')[0]
                    if visitbooking.enddate >
keeper_review.creationtime:
                        owners.append(visitbooking.group.owner)
                        bookings.append(visitbooking)
        for walkingbooking in walkingbookings:
            if walkingbooking.group.owner not in owners_analysed:
                owners_analysed.append(walkingbooking.group.owner)
                if walkingbooking.group.owner not in owners:
                    keeper_review = reviews.filter(owner =
visitbooking.group.owner).order_by('-createtime')[0]
                    if walkingbooking.enddate >
keeper_review.creationtime:

```

```

        owners.append(walkingbooking.group.owner)
        bookings.append(walkingbooking)
    for boardingbooking in boardingbookings:
        if boardingbooking.group.owner not in owners_analysed:
            owners_analysed.append(boardingbooking.group.owner)
        if boardingbooking.group.owner not in owners:
            keeper_review = reviews.filter(owner =
boardingbooking.group.owner).order_by('-createtime')[0]
            if boardingbooking.enddate >
keeper_review.creationtime:
                owners.append(boardingbooking.group.owner)
                bookings.append(boardingbooking)
    for sittingbooking in sittingbookings:
        if sittingbooking.group.owner not in owners_analysed:
            owners_analysed.append(sittingbooking.group.owner)
        if sittingbooking.group.owner not in owners:
            keeper_review = reviews.filter(owner =
sittingbooking.group.owner).order_by('-createtime')[0]
            if sittingbooking.enddate >
keeper_review.creationtime:
                owners.append(sittingbooking.group.owner)
                bookings.append(sittingbooking)
    return render(request, "pethelperapp/for_reviewing.html",
{"owners": owners, "bookings": bookings })

def keeper_owners(request):
    keeper = Keeper.object.get(user = request.user, None)
    if not keeper:
        return render(request,
"pethelperapp/keeper_owners.html", {"owners": [] })
    visitbookings = Visitbooking.objects.filter(keeper =
keeper).order_by('-createtime')
    walkingbookings = Walkingbooking.objects.filter(keeper =
keeper).order_by('-createtime')
    boardingbookings = Boardingbooking.objects.filter(keeper =
keeper).order_by('-createtime')
    sittingbookings = Sittingbooking.objects.filter(keeper =
keeper).order_by('-createtime')
    owners = []
    for visitbooking in visitbookings:
        if visitbooking.group.owner not in owners:
            owners.append(visitbooking.group.owner)
    for walkingbooking in walkingbookings:
        if walkingbooking.group.owner not in owners:
            owners.append(walkingbooking.group.owner)
    for boardingbooking in boardingbookings:
        if boardingbooking.group.owner not in owners:
            owners.append(boardingbooking.group.owner)

```

```

        for sittingbooking in sittingbookings:
            if sittingbooking.group.owner not in owners:
                owners.append(sittingbooking.group.owner)
        return render(request, "pethelperapp/keeper_owners.html",
{"owners": owners})

def add_free(request):
    keeper = Keeper.objects.get(user = request.user)
    if request.user.is_active and keeper:
        if request.method == "POST":
            form_keeperfree = KeeperFreeFrm(request.POST)
            if form_keeperfree.is_valid():
                keeperfree = form_keeperfree.save(commit =
False)

                keeperfree.keeper = keeper
                keeperfree.save()
                return redirect(request.path)
            else:
                form_keeperfree = KeeperFreeFrm()
        else:
            form_keeperfree = None
    return render(request, "pethelperapp/add_free.html",
{"form_keeperfree": form_keeperfree })

def add_short_tariff(request):
    keeper = Keeper.objects.get(user = request.user)
    if request.user.is_active and keeper:
        if request.method == "POST":
            form_shorttariff = ShortTariffFrm(request.POST)
            if form_shorttariff.is_valid():
                shorttariff = form_shorttariff.save(commit =
False)

                shorttariff.keeper = keeper
                shorttariff.save()
                return redirect(request.path)
            else:
                form_shorttariff = ShortTariffFrm()
        else:
            form_shorttariff = None
    return render(request, "pethelperapp/add_short_tariff.html",
{"form_tariff": form_shorttariff })

def edit_short_tariff(request, id):
    keeper = Keeper.objects.get(user = request.user)
    short_tariff = get_object_or_404(ShortTariff, id = id)
    if request.user.is_active and keeper and short_tariff.keeper
== keeper:

```

```

        form_shorttariff = ShortTariffFrm(request.POST or None,
instance = short_tariff)
        if request.method == "POST" and
form_shorttariff.is_valid():
            form_shorttariff.save()
            return redirect('keeper', id = keeper.id)
        else:
            form_shorttariff = None
            return render(request,
"pethelperapp/edit_short_tariff.html", {"form_shorttariff":
form_shorttariff })

def add_keeping_tariff(request):
    keeper = Keeper.objects.get(user = request.user)
    if request.user.is_active and keeper:
        if request.method == "POST":
            form_keepingtariff = KeepingTariffFrm(request.POST)
            if form_keepingtariff.is_valid():
                keepingtariff = form_keepingtariff.save(commit =
False)

                tariffs = KeepingTariff.objects.filter(keeper =
keeper, unit = keepingtariff.unit).values('begintime',
'finishtime').distinct()

                intersection = False
                for tariff in tariffs:
                    if keepingtariff.begintime >
tariff.finishtime or tariff.begintime > keepingtariff.finishtime:
                        intersection = True
                if not intersection:
                    keepingtariff.keeper = keeper
                    keepingtariff.save()
                    return redirect(request.path)
                else:
                    form_keepingtariff = KeepingTariffFrm()
            else:
                form_keepingtariff = KeepingTariffFrm()
        else:
            form_keepingtariff = None
            return render(request,
"pethelperapp/add_keeping_tariff.html", {"form_tariff":
form_keepingtariff })

def edit_keeping_tariff(request, id):
    keeper = Keeper.objects.get(user = request.user)
    keeping_tariff = get_object_or_404(KeepingTariff, id = id)
    if request.user.is_active and keeper and
keeping_tariff.keeper == keeper:

```

```

        form_keepingtariff = KeepingTariffFrm(request.POST or
None, instance = keeping_tariff)
        if request.method == "POST" and
form_keepingtariff.is_valid():
            form_keepingtariff.save()
            return redirect('keeper', id = keeper.id)
        else:
            form_keepingtariff = None
            return render(request,
"pethelperapp/edit_keeping_tariff.html", {"form_keepingtariff":
form_keepingtariff })

def add_breedpost(request):
    keeper = Keeper.objects.get(user = request.user)
    if request.user.is_active and keeper:
        if request.method == "POST":
            form_breedpost = BreedpostFrm(request.POST)
            if form_breedpost.is_valid():
                breedpost = form_breedpost.save(commit = False)
                breedpost.keeper = keeper
                breedpost.save()
                return redirect(request.path)
            else:
                form_breedpost = BreedpostFrm()
        else:
            form_breedpost = None
            return render(request, "pethelperapp/add_breedpost.html",
{"form_breedpost": form_breedpost })

def edit_breedpost(request, id):
    keeper = Keeper.objects.get(user = request.user)
    breedpost = get_object_or_404(Breedpost, id = id)
    if request.user.is_active and keeper and breedpost.keeper ==
keeper:
        form_breedpost = BreedpostFrm(request.POST or None,
instance = breedpost)
        if request.method == "POST" and
form_breedpost.is_valid():
            form_breedpost.save()
            return redirect('breedpost', id = breedpost.id)
        else:
            form_breedpost = None
            return render(request, "pethelperapp/edit_breedpost.html",
{"form_breedpost": form_breedpost })

def submit_booking(request, id, type = 0):
    keeper = Keeper.objects.get(user = request.user)
    if type == 4:

```

```

        booking = Sittingbooking.objects.get(keeper = keeper, id
= id)
    elif type == 3:
        booking = Boardingbooking.objects.get(keeper = keeper,
id = id)
    elif type == 2:
        booking = Walkingbooking.objects.get(keeper = keeper, id
= id)
    elif type == 1:
        booking = Visit.objects.get(visitbooking__keeper =
keeper, id = id)
    else:
        booking = Visitbooking.objects.get(keeper = keeper, id =
id)

    if request.user.is_active and keeper:
        if request.method == "POST":
            if booking.state == 0:
                booking.state = 2
            elif booking.state == 1:
                booking.state = 3
                keeper.lastoperation = datetime.now()
                keeper.save()
            booking.save()
            if type == 1 and booking.state == 3:
                visits = Visit.objects.filter(visitbooking =
booking.id)

                visits = visits.filter(~Q(state = 3))
                if not visits:
                    visitbooking = Visitbooking.objects.get(id =
booking.visitbooking.id)

                    visitbooking.state = 3
                    visitbooking.save()
                return redirect('keeper-bookings', id = keeper.id)
            else:
                return redirect('keeper-bookings', id =
request.user.id)
        else:
            return redirect('keeper-bookings', id = request.user.id)

def cancel_booking(request, id, type = 0):
    keeper = Keeper.objects.get(user = request.user)
    if type == 4:
        booking = Sittingbooking.objects.get(keeper = keeper, id
= id)
    elif type == 3:
        booking = Boardingbooking.objects.get(keeper = keeper,
id = id)
    elif type == 2:

```

```

        booking = Walkingbooking.objects.get(keeper = keeper, id
= id)
    elif type == 1:
        booking = Visit.objects.get(visitbooking__keeper =
keeper, id = id)
    else:
        booking = Visitbooking.objects.get(keeper = keeper, id =
id)

    if request.user.is_active and keeper:
        if request.method == "POST":
            if booking.state == 0 or booking.state == 1 or
booking.state == 2:
                booking.state = 4
                if type != 1:
                    res = CancelPayment(booking = booking, owner =
booking.group.owner, keeper = keeper)
                    if res:
                        booking.save()
                else:
                    booking.save()
                return redirect('keeper-bookings', id = keeper.id)
            else:
                return redirect('keeper-bookings', id =
request.user.id)
        else:
            return redirect('keeper-bookings', id = request.user.id)

def keeper_breeds(request):
    keeper = Keeper.objects.get(user = request.user)
    if request.user.is_active and keeper:
        breeds = keeper.breeds
        if request.method == "POST":
            form_keeperbreed = KeeperbreedFrm(request.POST)
            if form_keeperbreed.is_valid():
                keeper = form_keeperbreed.save(commit = False)
                keeper.save()
                return redirect(request.path)
        else:
            form_keeperbreed = KeeperbreedFrm()
    else:
        form_keeperbreed = None
    return render(request, "pethelperapp/keeper_breeds.html",
{"form_keepersbreeds": form_keeperbreed, "breeds": breeds})

def cancel_keeperfree(request, id):
    kf = KeeperFree.objects.get(id = id).delete()

def keeper_frees(request):

```

```

        keeper = Keeper.objects.get(user = request.user)
        older = datetime.now()
        keeperfress = KeeperFree.objects.filter(keeper = keeper,
        finishtime__gte = older).order_by('begintime')
        return render(request,
        "pethelperapp/keeper_breedposts.html", {"frees": keeperfress })

def keeper_bookings(request):
    mode = int(self.request.GET.get('mode', 0))
    keeper = Keeper.objects.get(user = request.user)
    older = datetime.now()
    if keeper:
        if mode == 0 or mode == 1:
            visitbookings_new =
Visitbooking.objects.filter(keeper = keeper, createtime__gte =
older).order_by('createtime')
            walkingbookings_new =
Walkingbooking.objects.filter(keeper = keeper, createtime__gte =
older).order_by('createtime')
            boardingbookings_new =
Boardingbooking.objects.filter(keeper = keeper, createtime__gte =
older).order_by('createtime')
            sittingbookings_new =
Sittingbooking.objects.filter(keeper = keeper, createtime__gte =
older).order_by('createtime')
            if mode == 1:
                return render(request,
                "pethelperapp/keeper_bookings.html", {"visitbookings":
visitbookings_new, "walkingbookings" : walkingbookings_new,
                "boardingbookings" : boardingbookings_new, "sittingbookings" :
sittingbookings_new })
            else:
                bookings = sorted(chain(visitbookings_new,
walkingbookings_new, boardingbookings_new, sittingbookings_new), key =
attrgetter('createtime'))
                return render(request,
                "pethelperapp/keeper_bookings.html", {"bookings": bookings })
            if mode == 2 or mode == 4:
                visitbookings_old =
Visitbooking.objects.filter(keeper = keeper, createtime__lt =
older).order_by('-createtime')
                walkingbookings_old =
Walkingbooking.objects.filter(keeper = keeper, createtime__lt =
older).order_by('-createtime')
                boardingbookings_old =
Boardingbooking.objects.filter(keeper = keeper, createtime__lt =
older).order_by('-createtime')

```

```

        sittingbookings_old =
Sittingbooking.objects.filter(keeper = keeper, createtime__lt =
older).order_by('-createtime')
        if mode == 3:
            return render(request,
"pethelperapp/keeper_bookings.html", {"visitbookings":
visitbookings_old, "walkingbookings" : walkingbookings_old,
"boardingbookings" : boardingbookings_old, "sittingbookings" :
sittingbookings_old })
        else:
            bookings = sorted(chain(visitbookings_old,
walkingbookings_old, boardingbookings_old, sittingbookings_old), key =
attrgetter('-createtime'))
            return render(request,
"pethelperapp/keeper_bookings.html", {"bookings": bookings })
        else:
            return render(request,
"pethelperapp/keeper_bookings.html", {"bookings": []})

def keeper_plan(request):
    today = datetime.now()
    year = int(self.request.GET.get('year', today.year))
    month = int(self.request.GET.get('month', today.month))
    day = int(self.request.GET.get('day', today.day))
    search_day = datetime.datetime(year, month, day, 0, 0)
    search_day_end = datetime.datetime(year, month, day, 23, 59)
    keeper = Keeper.objects.get(user = request.user)
    if keeper:
        visitbookings_new = Visitbooking.objects.filter(keeper =
keeper, createtime__gte = search_day, createtime__lte =
search_day_end, state__in = [0,1]).order_by('createtime')
        walkingbookings_new =
Walkingbooking.objects.filter(keeper = keeper, createtime__gte =
search_day, createtime__lte = search_day_end, state__in =
[0,1]).order_by('createtime')
        boardingbookings_new =
Boardingbooking.objects.filter(keeper = keeper, createtime__gte =
search_day, createtime__lte = search_day_end, state__in =
[0,1]).order_by('createtime')
        sittingbookings_new =
Sittingbooking.objects.filter(keeper = keeper, createtime__gte =
search_day, createtime__lte = search_day_end, state__in =
[0,1]).order_by('createtime')
        bookings = sorted(chain(visitbookings_new,
walkingbookings_new, boardingbookings_new, sittingbookings_new), key =
attrgetter('createtime'))
        return render(request, "pethelperapp/keeper_plan.html",
{"bookings": bookings })

```

```

        else:
            return render(request, "pethelperapp/keeper_plan.html",
{"bookings": []})

def keeper_success(request):
    keeper = Keeper.objects.get(user = request.user)
    if not keeper:
        return render(request,
"pethelperapp/keeper_success.html", {"results": []})
    animals = Animal.objects.all()
    results = {}
    quantity = {}
    for animal in animals:
        breeds = animal.breeds
        results[animal.name] = {}
        quantity[animal.name] = {}
        for breed in breeds:
            results[animal.name][breed.name] = 0
            quantity[animal.name][breed.name] = 0
    reviews = Review.objects.filter(keeper = keeper, direction =
0)

    for review in reviews:
        time = review.creationtime
        visitbooking = Visitbooking.objects.filter(keeper =
keeper, group__owner = review.owner, createtime__lte =
review.creationtime).order_by('-createtime')[0]
        group = visitbooking.group
        for pet in group.pets:
            breed = pet.breed
            animal = breed.animal
            results[animal.name][breed.name] =
results[animal.name][breed.name] + review.mark
            quantity[animal.name][breed.name] =
quantity[animal.name][breed.name] + 1
        for animal in animals:
            for breed in breeds:
                results[animal.name][breed.name] =
results[animal.name][breed.name] / quantity[animal.name][breed.name]
    return render(request, "pethelperapp/keeper_success.html",
{"results": results})

```

Б.2 Текст файла common_views.py

```

from .models import KeeperFree, Animal, City, ShortTariff,
KeepingTariff, Breedpost, BreedpostComment, Breed, Owner, Pet, Keeper,
Review, Visitbooking, Walkingbooking, Boardingbooking, Sittingbooking,
Message

```

```

import datetime
from pethelperapp.forms import BreedpostCommentFrm, ReviewFrm,
MessageFrm
from django.db.models import Q
from statistics import mean
import pgeocode

def compare(start_time, end_time, begintime, finishtime):
    if start_time >= begintime:
        left = start_time
    else:
        left = begintime
    if end_time <= finishtime:
        right = end_time
    else:
        right = finishtime
    if left != start_time and right != end_time:
        return None
    else:
        return (left, right)

def search_keeper(request):
    start_year = self.request.GET.get('startyear', None)
    start_month = self.request.GET.get('startmonth', None)
    start_day = self.request.GET.get('startday', None)
    start_hour = self.request.GET.get('starthour', 0)
    start_minute = self.request.GET.get('startminute', 0)
    if start_year and start_month and start_day:
        start_year = int(start_year)
        start_month = int(start_month)
        start_day = int(start_day)
        start_hour = int(start_hour)
        start_minute = int(start_minute)
        start_date = datetime.datetime(start_year, start_month,
start_day, start_hour, start_minute)
    end_year = self.request.GET.get('endyear', None)
    end_month = self.request.GET.get('endmonth', None)
    end_day = self.request.GET.get('endday', 0)
    end_hour = self.request.GET.get('endhour', 0)
    end_minute = self.request.GET.get('endminute', 0)
    if end_year and end_month and end_day:
        end_year = int(end_year)
        end_month = int(end_month)
        end_day = int(end_day)
        end_hour = int(end_hour)
        end_minute = int(end_minute)
        end_date = datetime.datetime(end_year, end_month,
end_day, end_hour, end_minute)

```

```

else:
    end_date = None
    service = self.request.GET.get('service', 0)
    if service:
        service = int(service)
    times = int(self.request.GET.get('times', 1))
    day_times = int(self.request.GET.get('daytimes', 1))
    pricemin = int(self.request.GET.get('pricemin', 0))
    pricemax = self.request.GET.get('pricemax', None)
    if pricemax:
        pricemax = int(pricemax)
    breed_type = int(self.request.GET.get('breed', 0))
    breed_type = Breed.objects.get(id = breed_type)
    animal_quantity = int(self.request.GET.get('quantity', 1))
    extra_pets = animal_quantity - 1
    search_city = int(self.request.GET.get('city', 0))
    search_city = City.object.get(id = search_city)
    order_direction = int(self.request.GET.get('order', 0))
    search_zipcode = self.request.GET.get('zipcode', None)
    if search_zipcode:
        search_zipcode = int(search_zipcode)
    if (service == 0 or service == 1) and not end_date:
        end_date = start_date + datetime.timedelta(minutes = 30)
        start_date = datetime.datetime(start_date.year,
start_date.month, start_date.day, start_date.hour, start_date.minute -
10)

        end_date = datetime.datetime(end_date.year, end_date.month,
end_date.day, end_date.hour, end_date.minute + 10)
        if service == 0 or service == 1:
            delta = end_date - start_date
            delta_minutes = delta.minutes
            delta_hours = delta.hours
            add_minutes = delta_minutes - delta_hours * 60
            keeper_frees = KeeperFree.objects.filter(begintime__lte =
start_date, finishtime__gte = end_date, keeper__breeds = breed_type,
keeper__city = search_city).distinct('keeper')
            keepers = []
            calc_service = service - 2
            days = delta.days
            add_hours = delta_hours - days * 60 * 24
            start_time = datetime.time(start_date.hour,
start_date.minute)
            end_time = datetime.time(end_date.hour, end_date.minute)
            for onekeeper in keeper_frees:
                keeper = onekeeper.keeper
                add = False
                if search_zipcode == keeper.zipcode:
                    add = True

```

```

elif keeper.mobility == True:
    add = True
else:
    dist = pgeocode.GeoDistance(keeper.city.country)
    if dist.query_postal_code(search_zipcode,
keeper.zipcode) * 1000 <= 700:
        add = True
    if service == 0 or service == 1:
        tariffs = ShortTariff.objects.filter(keeper =
keeper, fromdays__lte = times, duration__gte = delta_minutes, extrapet
= extra_pets, daytimes = day_times, visitwalking = service)
        tariff = tariffs.order_by('-fromdays')[0]
        price = tariff.dayprice * times
        if pricemax:
            if price >= pricemin and price <= pricemax and
add:
                data = {}
                data['keeper'] = keeper
                data['price'] = price
                data['rating'] = keeper.rating
                keepers.append(data)
            elif price >= pricemin and add:
                data = {}
                data['keeper'] = keeper
                data['price'] = price
                data['rating'] = keeper.rating
                keepers.append(data)
    if service == 2 or service == 3:
        tariffs = KeepingTariff.objects.filter(keeper =
keeper, sittingboarding = calc_service)
        price = None
        parts = []
        prices = []
        prices_minutes = []
        for tariff in tariffs:
            if tariff.unit == 1 or tariff.unit == 0:
                t = (tariff.begintime, tariff.finishtime)
                if t not in parts:
                    parts.append(t)
                    prices.append(0)
                    prices_minutes.append(0)
        for tariff in tariffs:
            if tariff.unit == 2:
                calc_days = days
                if tariff.begintime > start_time:
                    calc_days = calc_days + 1
                if tariff.finishtime < end_time:
                    calc_days = calc_days + 1

```

```

        if calc_days == 0:
            calc_days = 1
        if tariff.multiples <= calc_days:
            tariff_value = tariff.timeprice *
calc_days
            elif tariff.unit == 1 and delta_hours >= 1:
                value24 = False
                if tariff.beginntime.hour >=
tariff.finishtime.hour:
                    tariff.finishtime =
datetime.time(tariff.finishtime.hour + 24, tariff.finishtime.minute)
                    value24 = True
                    pos = parts.index((tariff.beginntime,
tariff.finishtime))
                    if start_time.hour >= end_time.hour and
value24:
                        res = datetime.time(end_time.hour + 24,
end_time.minute)
                        limits = compare(start_time, res,
tariff.beginntime, tariff.finishtime)
                        elif start_time.hour >= end_time.hour and
not value24:
                            limits = compare(start_time,
datetime.time(24, 0), tariff.beginntime, tariff.finishtime)
                            limits = limits or
compare(datetime.time(0, 0), end_time, tariff.beginntime,
tariff.finishtime)
                        else:
                            limits = compare(start_time, end_time,
tariff.beginntime, tariff.finishtime)
                            if limits:
                                hours = limits[1].hour - limits[0].hour
                                if limits[1].hour == limits[0].hour and
limits[1].minute != 0:
                                    hours = hours + 1
                                if limits[1].hour > limits[0].hour and
limits[1].minute != 0:
                                    hours = hours + 1
                                d = tariff.finishtime - tariff.beginntime
                                if tariff.duration <= hours or
tariff.duration == d.hours:
                                    value = tariff.timeprice * hours
                                    if prices[pos] > value:
                                        prices[pos] = value
                                elif tariff.unit == 0 and delta_minutes >= 1:
                                    value24 = False
                                    if tariff.beginntime.hour >=
tariff.finishtime.hour:

```

```

tariff.finishtime =
datetime.time(tariff.finishtime.hour + 24, tariff.finishtime.minute)
value24 = True
pos = parts.index((tariff.begintime,
tariff.finishtime))
if start_time.hour >= end_time.hour and
value24:
    res = datetime.time(end_time.hour + 24,
end_time.minute)
    limits = compare(start_time, res,
tariff.begintime, tariff.finishtime)
    elif start_time.hour >= end_time.hour and
not value24:
        limits = compare(start_time,
datetime.time(24, 0), tariff.begintime, tariff.finishtime)
        limits = limits or
compare(datetime.time(0, 0), end_time, tariff.begintime,
tariff.finishtime)
    else:
        limits = compare(start_time, end_time,
tariff.begintime, tariff.finishtime)
    if limits:
        minut = limits[1] - limits[0]
        minutes = minut.minutes
        if tariff.duration <= minutes:
            value = tariff.timeprice * minutes
            if prices_minutes[pos] > value:
                prices_minutes[pos] = value
        if not price or price > tariff_value:
            price = tariff_value
value = sum(prices)
if value < price:
    price = value
value = sum(prices_minutes)
if value < price:
    price = value
if price >= pricemin and price <= pricemax:
    data = {}
    data['keeper'] = keeper
    data['price'] = price
    data['rating'] = keeper.rating
    keepers.append(data)
if order_direction == 0:
    keepers_sorted = sorted(keepers, key = lambda d:
d['rating'])
elif order_direction == 1:
    keepers_sorted = sorted(keepers, key = lambda d:
d['price'])

```

```

        elif order_direction == 2:
            keepers_sorted = sorted(keepers, key = lambda d: -
d['price'])
            return render(request, "pethelperapp/search_keeper.html",
{"keepers": keepers_sorted})
        def breedpost_detalize(request, id):
            breedpost = get_object_or_404(Breedpost, id = id)
            comments = BreedpostComment.objects.filter(breedpost =
breedpost).order_by('-creationtime')
            if request.user.is_active:
                if request.method == "POST":
                    form_breedpostcomment =
BreedpostCommentFrm(request.POST)
                    if form_breedpostcomment.is_valid():
                        breedpostcomment =
form_breedpostcomment.save(commit = False)
                        breedpostcomment.breedpost = breedpost
                        breedpostcomment.user = request.user
                        breedpostcomment.save()
                        return redirect(request.path)
                    else:
                        form_breedpostcomment = BreedpostCommentFrm()
                else:
                    form_breedpostcomment = None
            return render(request, "pethelperapp/breedpost.html",
{"breedpost": breedpost, "comments": comments,
"breedpostcomment_form": form_breedpostcomment})

        def search_breedpost(request):
            search_text = self.request.GET.get('text', None)
            breedposts = Breedpost.objects.filter(Q(posttitle__icontains
= search_text) | Q(posttext__icontains = search_text))
            breed = self.request.GET.get('breed', None)
            if breed:
                breed = Breed.object.get(id = int(breed))
                breedposts = breedposts.filter(breed = breed)
            else:
                animal = self.request.GET.get('animal', None)
                if animal:
                    animal = Animal.object.get(id = int(animal))
                    breedposts = breedposts.filter(breed__animal =
animal, animalshared = True)
            start_year = self.request.GET.get('startyear', 0)
            end_year = self.request.GET.get('endyear', None)
            start_month = self.request.GET.get('startmonth', 1)
            end_month = self.request.GET.get('endmonth', None)
            start_day = self.request.GET.get('startday', 1)
            end_day = self.request.GET.get('startday', None)

```

```

        start_date = datetime.datetime(int(start_year),
int(start_month), int(start_day))
        breedposts = breedposts.filter(creationtime__gte =
start_date)
        if end_year:
            if not end_month:
                end_date = datetime.datetime(int(end_year), 12, 31)
            else:
                if not end_day:
                    if month == 12:
                        end_date = datetime.datetime(int(end_year),
12, 31)
                    else:
                        end_date = datetime.datetime(int(end_year),
int(end_month) + 1, 1) - datetime.timedelta(days = 1)
                breedposts = breedposts.filter(creationtime__lte =
end_date)
        return render(request,
"pethelperapp/search_breedposts.html", {"breedposts": breedposts })

def owner_animals(request, id = None):
    if id:
        pet_owner = get_object_or_404(Owner, id = id)
    else:
        pet_owner = Owner.objects.get(user = request.user)
    pets = Pet.objects.filter(owner = pet_owner)
    return render(request, "pethelperapp/owner_pets.html",
{"pets": pets})

def keeper_profile(request, id = None):
    if id:
        keeper = get_object_or_404(Keeper, id = id)
    else:
        keeper = Keeper.objects.get(user = request.user)
    short_tariffs = ShortTariff.objects.filter(keeper = keeper)
    keeping_tariffs = KeepingTariff.objects.filter(keeper =
keeper)
    breedposts = Breedpost.objects.filter(keeper =
keeper).order_by('-creationtime')[:3]
    reviews = Review.objects.filter(keeper = keeper, direction =
0).order_by('-creationtime')[:3]
    return render(request, "pethelperapp/keeper_profile.html",
{"keeper": keeper, "short_tariffs": short_tariffs, "keeping_tariffs":
keeping_tariffs, "breedposts": breedposts, "reviews": reviews})

def keeper_reviews(request, id = None):
    if id:
        keeper = get_object_or_404(Keeper, id = id)

```

```

else:
    keeper = Keeper.objects.get(user = request.user)
    reviews = Review.objects.filter(keeper = keeper, direction =
0).order_by('-creationtime')
    return render(request, "pethelperapp/keeper_reviews.html",
{"reviews": reviews })

def add_review(request, id):
    keeper = Keeper.objects.get(user = request.user)
    if not keeper:
        direction = 0
        owner = Owner.objects.get(user = request.user)
        keeper = get_object_or_404(Keeper, id = id)
    else:
        direction = 1
        owner = get_object_or_404(Owner, id = id)
    if request.user.is_active:
        possible = False
        visitbooking = Visitbooking.objects.filter(keeper =
keeper, group__owner = owner, state = 3).order_by('-createtime')[0]
        walkingbooking = Walkingbooking.objects.filter(keeper =
keeper, group__owner = owner, state = 3).order_by('-createtime')[0]
        boardingbooking = Boardingbooking.objects.filter(keeper
= keeper, group__owner = owner, state = 3).order_by('-createtime')[0]
        sittingbooking = Sittingbooking.objects.filter(keeper =
keeper, group__owner = owner, state = 3).order_by('-createtime')[0]
        review = Review.objects.filter(keeper = keeper, owner =
owner, direction = 1)[0]
        if visitbooking.enddate > review.creationtime or
walkingbooking.enddate > review.creationtime or
boardingbooking.enddate > review.creationtime or
sittingbooking.enddate > review.creationtime:
            possible = True
        if not possible:
            form_review = None
            return render(request,
"pethelperapp/add_review.html", {"form_review": form_review })
        if request.method == "POST":
            form_review = ReviewFrm(request.POST)
            if form_review.is_valid():
                review = form_review.save(commit = False)
                review.direction = direction
                review.owner = owner
                review.keeper = keeper
                if review.recommendation and direction == 0:
                    reviews = Review.objects.filter(owner =
owner, direction = 0).order_by('-creationtime')[0]
                    diff = reviews.creationtime - datetime.now()

```

```

        if diff.months >= 1:
            keeper.recommendations =
keeper.recommendations + 1
        else:
            review.recommendation = False
            review.save()
            reviews = Review.objects.filter(owner = owner,
keeper = keeper, direction = direction).values('mark')
            avg_mark = mean(reviews)
            if direction == 0:
                keeper.rating = avg_mark
                keeper.save()
            else:
                owner.rating = avg_mark
                owner.save()
            return redirect(request.path)
    else:
        form_review = ReviewFrm()
    else:
        form_review = None
    return render(request, "pethelperapp/add_review.html",
{"form_review": form_review })

def keeper_breedposts(request, id = None):
    if id:
        keeper = get_object_or_404(Keeper, id = id)
    else:
        keeper = Keeper.objects.get(user = request.user)
    breedposts = Breedpost.objects.filter(keeper =
keeper).order_by('-creationtime')
    return render(request,
"pethelperapp/keeper_breedposts.html", {"breedposts": breedposts})

def owner_profile(request, id = None):
    if id:
        owner = get_object_or_404(Owner, id = id)
    else:
        owner = Owner.objects.get(user = request.user)
    reviews = Review.objects.filter(owner = owner, direction =
1).order_by('-creationtime')
    return render(request, "pethelperapp/owner_profile.html",
{"owner": owner, "reviews": reviews })

def chattered(request):
    keeper = Keeper.objects.get(user = request.user)
    if not keeper:
        direction = 0
        owner = Owner.objects.get(user = request.user)

```



```

        mess = mess + 1
        keeper.waiting = waiting / mess
        keeper.save()
        return redirect(request.path)
    else:
        form_message = MessageFrm()
    else:
        form_message = None
    return render(request, "pethelperapp/messages.html",
{"messages": messages, "direction": direction, "form": form_message })

def keeper_owner_bookings(request):
    username = self.request.GET.get('user', None)
    us = False
    if username:
        owner = Owner.objects.get(user__username = username)
        if not owner:
            owner = Owner.objects.get(user = request.user)
            us = True
    else:
        return render(request,
"pethelperapp/keeper_owner_bookings.html", {"bookings": []})
    mode = int(self.request.GET.get('mode', 0))
    if not us:
        keeper = Keeper.objects.get(user = request.user)
    else:
        keeper = Keeper.objects.get(user__username = username)
    if not keeper:
        return render(request,
"pethelperapp/keeper_owner_bookings.html", {"bookings": []})
    if mode == 0 or mode == 1:
        visitbookings = Visitbooking.objects.filter(keeper =
keeper, group__owner = owner).order_by('-createtime')
        walkingbookings_new =
Walkingbooking.objects.filter(keeper = keeper, group__owner =
owner).order_by('-createtime')
        boardingbookings_new =
Boardingbooking.objects.filter(keeper = keeper, group__owner =
owner).order_by('-createtime')
        sittingbookings_new =
Sittingbooking.objects.filter(keeper = keeper, group__owner =
owner).order_by('-createtime')
        if mode == 1:
            return render(request, "pethelperapp/
keeper_owner_bookings.html", {"visitbookings": visitbookings_new,
"walkingbookings" : walkingbookings_new, "boardingbookings" :
boardingbookings_new, "sittingbookings" : sittingbookings_new })
    else:

```

```
        bookings = sorted(chain(visitbookings_new,
walkingbookings_new, boardingbookings_new, sittingbookings_new), key =
attrgetter('-createtime'))
        return render(request,
"pethelperapp/keeper_owner_bookings.html", {"bookings": bookings })
```

ДОДАТОК В
Слайди презентації



Рисунок В.1 – Початковий слайд

- Об'єкт роботи – процес догляду за тваринами.
- Предмет роботи – програмне забезпечення допомоги для догляду за тваринами.
- Мета роботи – розробити програмне забезпечення допомоги для догляду за тваринами для надання можливості взаємовигідної взаємодії між власниками тварин і помічниками з догляду за тваринами.

Рисунок В.2 – Мета роботи

Завдання роботи

- аналіз існуючого програмного забезпечення, що підтримує догляд за тваринами;
- проектування програмного забезпечення допомоги для догляду за тваринами;
- реалізація і тестування програми.

3

Рисунок В.3 – Завдання роботи

Програмне забезпечення допомоги для догляду за тваринами

Характеристика	Rover.com	Petcloud	Petsitting24	Wag	Розробка
Територія дії	США	23 країни, Україна відсутня	Швейцарія	США	Потенційно будь- яка
Види тварин	Собаки, коти	Усі	Усі	Собаки, коти	Усі
Гнучка система тарифів	Ні	Ні	Ні, погодинна	Ні	Так
Налаштування послуг на рівні визначення порід тварин	Ні	Досвід помічника з різними породами	Ні	Ні	Так
Статті про догляд	Ні	Ні	Ні	Тільки загальні	Так
Типи сервісів	Вигулювання, відвідування, догляд у власника, у помічника	Вигулювання, відвідування, догляд у власника, у помічника та інші	Не деталізовано	Вигулювання, відвідування, догляд у власника, у помічника	Вигулювання, відвідування, догляд у власника, у помічника ⁴

Рисунок В.4 – Програмне забезпечення допомоги для догляду за тваринами

Порівняння фреймворків Django і Falcon для розробки вебдодатку

Показник	Django	Falcon
Тип фреймворку	Фреймворк за принципом «все включено»	Мінімалістичний бекенд-фреймворк
Фреймворк для роботи з БД	Вбудований	Можливе підключення
Панель керування для адміністратора	Вбудована	Можливе підключення
Бібліотека шаблонів	Вбудована	Можливе підключення
Забезпечення безпеки	Вбудоване, багатоцільове	Вбудовані засоби майже повністю відсутні

Рисунок В.5 – Порівняння фреймворків Django і Falcon для розробки вебдодатку

Діаграми прецедентів користувача і помічника

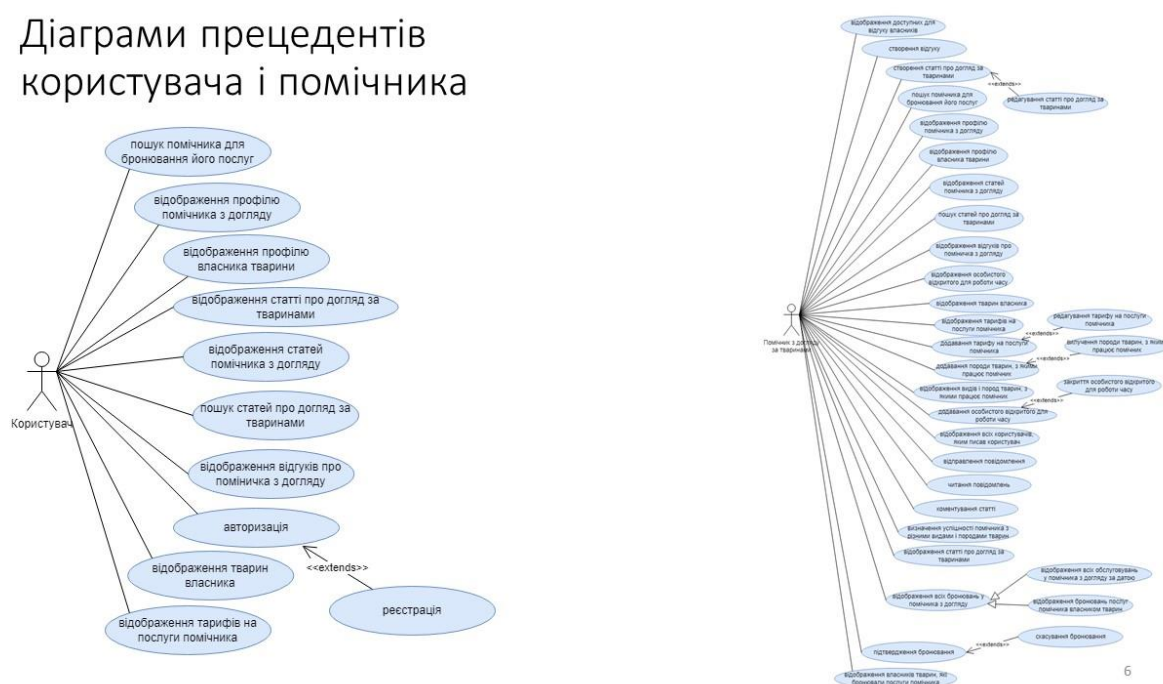


Рисунок В.6 – Діаграми прецедентів користувача і помічника

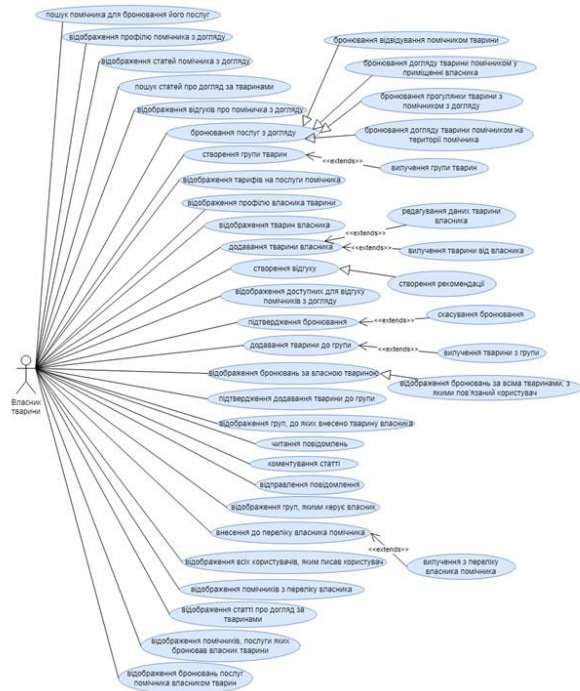


Рисунок В.7 – Діаграма прецедентів власника тварин

Схема бази даних

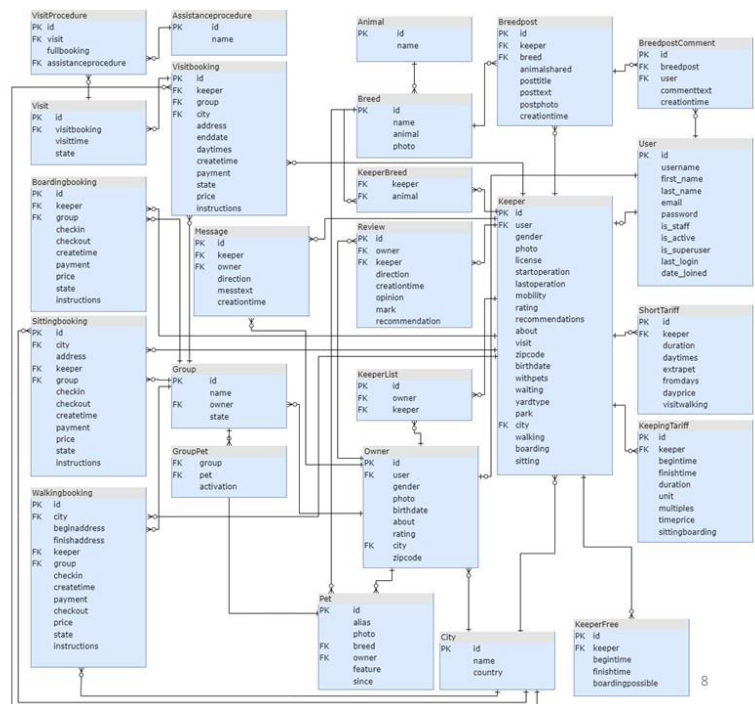
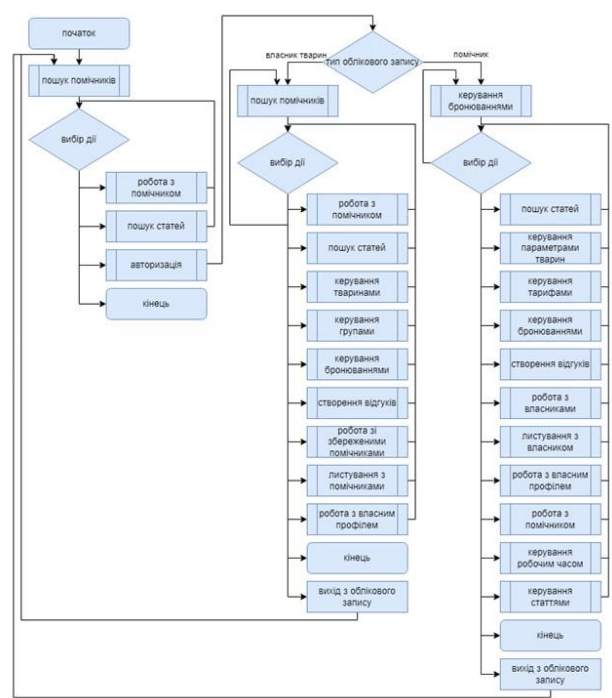


Рисунок В.8 – Схема бази даних

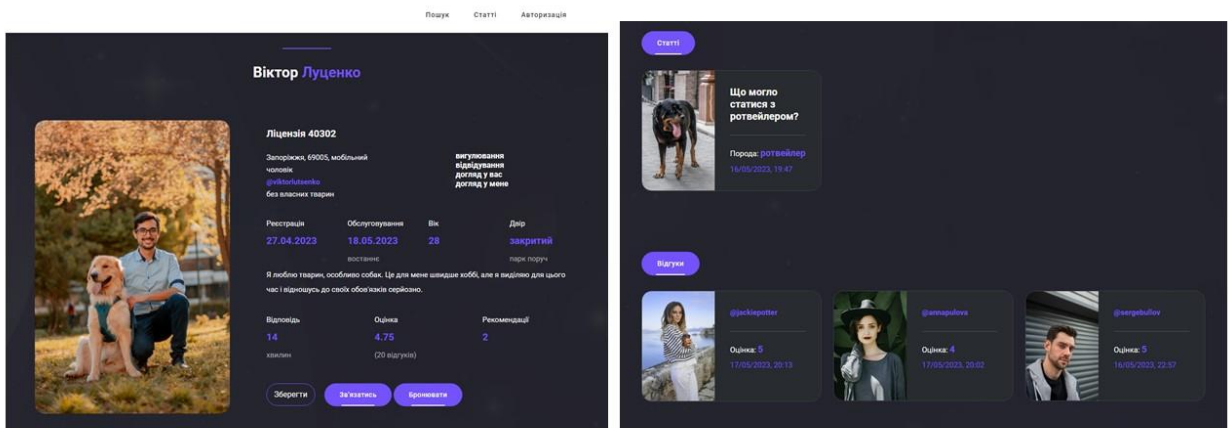
Схема функціонування програми



9

Рисунок В.9 – Схема функціонування програми

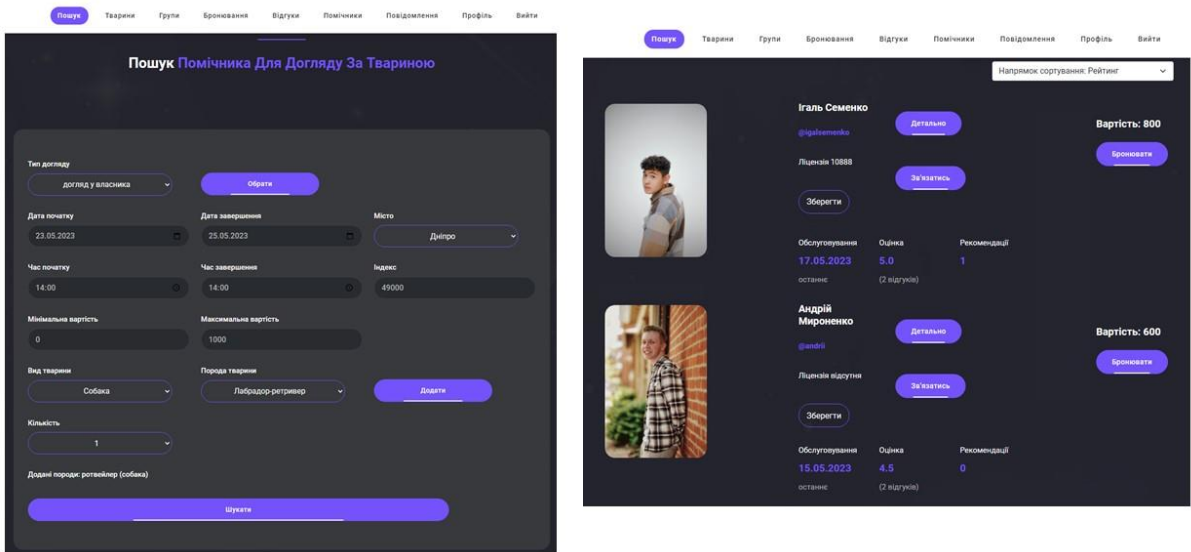
Вебдодаток для користувача



10

Рисунок В.10 – Вебдодаток для користувача

Вебдодаток для власника тварин



11

Рисунок В.11 – Вебдодаток для власника тварин

Вебдодаток для помічника з догляду за тваринами

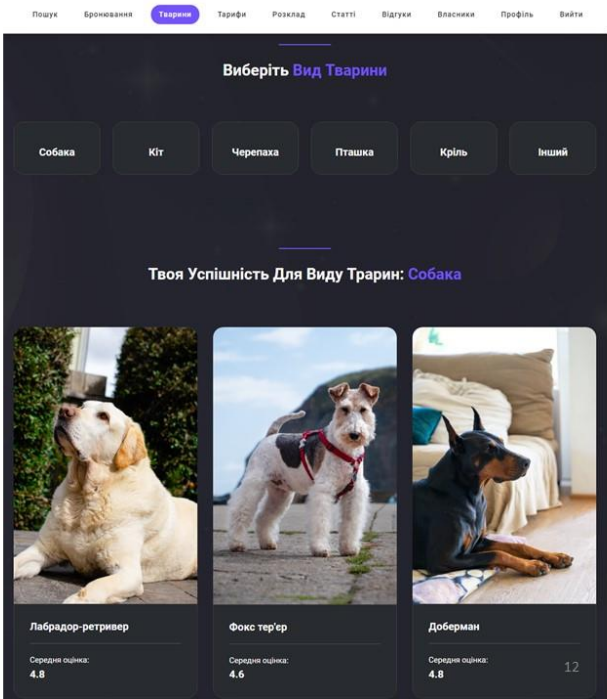


Рисунок В.12 – Вебдодаток для помічника з догляду за тваринами

Тестування програми

Функція	Користувач	Власник тварини	Помічник з догляду
1	2	3	4
Пошук помічника для бронювання його послуг	+	+	+
Відображення профілю помічника з догляду	+	+	+
Відображення профілю власника тварини	+	+	+
Відображення статей помічника з догляду	+	+	+
Пошук статей про догляд за тваринами	+	+	+
Відображення відгуків про помічника з догляду	+	+	+
Авторизація	+		
Відображення статті про догляд за тваринами	+	+	+
Відображення видів і порід тварин, з якими працює помічник			+
Додавання порід тварин, з якими працює помічник			+
Вилучення породи тварин, з якими працює помічник			+
Додавання тарифу на послуги помічника			+
Відображення помічників, послуги яких бронював власник тварини		+	
Редагування тарифу на послуги помічника			+
Відображення власників тварин, які бронювали послуги помічника			+
Відображення бронювань послуг помічника власником тварин		+	+

1	2	3	4
Підтвердження додавання тварини до групи	+		
Відображення груп, до яких внесено тварину власника	+		
Відображення груп, якими керує власник	+		
Вилучення групи тварин	+		
Коментування статті	+	+	
Визначення успішності помічника з різними видами і породами тварин		+	
Відображення бронювань за власною твариною	+		
Відображення бронювань за всіма тваринами, з якими пов'язаний користувач	+		
Відображення всіх бронювань у помічника з догляду		+	
Скасування бронювання	+	+	
Підтвердження бронювання	+	+	
Створення статті про догляд за тваринами		+	
Редагування статті про догляд за тваринами		+	
Відображення всіх користувачів, яким писав користувач	+	+	
Відправлення повідомлення	+	+	
Читання повідомлень	+	+	
Внесення до переліку власника помічника	+		
Відображення всіх бронювань помічника з догляду за датою		+	
Ресстрація	+		
Вилучення з переліку власника помічника	+		

1	2	3	4
Бронювання прогулянки тварини з помічником з догляду		+	
Бронювання відвідування помічником тварини		+	
Бронювання догляду тварини помічником на території помічника		+	
Бронювання догляду тварини помічником у приміщенні власника		+	
Відображення особистого відкритого для роботи часу			+
Додавання особистого відкритого для роботи часу			+
Закриття особистого відкритого для роботи часу			+
Відображення доступних для відгуків власників			+
Відображення доступних для відгуків помічників з догляду		+	
Створення відгуку		+	+
Створення рекомендації		+	
Відображення тварин власника	+	+	+
Додавання тварини власника		+	
Вилучення тварини від власника		+	
Редагування даних тварини власника		+	
Створення групи тварин		+	
Додавання тварини до групи		+	
Вилучення тварини з групи		+	
Відображення помічників з переліку		+	

13

Рисунок В.13 – Тестування програми

Висновки

- розробка програмного забезпечення допомоги для догляду за тваринами для надання можливості взаємовигідної взаємодії між власниками тварин і помічниками з догляду за тваринами була визначена метою дипломної кваліфікаційної роботи і була досягнута в роботі;
- в основу виконаної розробки було покладено аналіз предметної області, що дозволило встановити необхідні вимоги до програми, виділити потрібні особливості, що покращують зручність сторін у програмі, забезпечило ресурс для моделювання програми, проектування її структури, реалізації програми;
- для розробки було сукупно використано мови Python, HTML, CSS, JavaScript, фреймворк Django, систему керування базами даних PostgreSQL;
- програма придатна для використання як окрема платформа, на якій беруть участь власники домашніх тварин та люди, які готові працювати в ролі помічників з догляду за тваринами.

14

Рисунок В.14 – Висновки за роботою