

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)

магістра

(ступінь вищої освіти)

на тему ДОСЛІДЖЕННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ МЕТОДІВ

(назва теми)

ПІДТРИМКИ ПЛАНУВАННЯ ДІЯЛЬНОСТІ ПРИТУЛКУ ДЛЯ

ТВАРИН. RESEARCH AND SOFTWARE IMPLEMENTATION OF

METHODS FOR PLANNING THE ACTIVITY OF

AN ANIMAL SHELTER

Виконав(ла): студент(ка) 2 курсу, групи КНТ-212м

Спеціальності 122 Комп'ютерні науки

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Системи штучного інтелекту

АУБАКІРОВА М.Б.

(ПРІЗВИЩЕ та ініціали)

Керівник ЛЬОВКІН В.М.

(ПРІЗВИЩЕ та ініціали)

Рецензент СКРУПСЬКИЙ С.Ю.

(ПРІЗВИЩЕ та ініціали)

2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет _____ ФКНТ _____
Кафедра _____ програмних засобів _____
Ступінь вищої освіти _____ магістр _____
Спеціальність _____ 122 Комп'ютерні науки _____
(код і найменування)
Освітня програма (спеціалізація) _____ Системи штучного інтелекту _____
(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)

АУБАКІРОВОЇ Марії Булатівни

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Дослідження та програмна реалізація методів підтримки планування діяльності притулку для тварин. Research and Software Implementation of Methods for Planning the Activity of an Animal Shelter

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 « 14 » листопада 2023 року № 437

2. Строк подання студентом проєкту (роботи) 01 грудня 2023 року
3. Вихідні дані до проєкту (роботи) _____ рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз проблеми та постановка завдань дослідження. 2. Матеріали і методи. 3. Проєктування програмного забезпечення. 4. Основні рішення щодо реалізації програмного забезпечення. 5. Експлуатація, тестування та експериментальне дослідження програми.

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

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-5 Основна частина	ЛЬОВКІН В.М., доцент		
Нормоконтроль	КАМІНСЬКА Ж.К., асистент		

7. Дата видачі завдання « 04 » вересня 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	Дослідження проблеми	3-5 тижні	Розділ 2
4	Розробка архітектури програми	6 тиждень	Розділ 3
5	Розробка програми	7-8 тижні	Розділ 4
6	Тестування та експериментальне дослідження програми	9-10 тижні	Розділ 5
7	Оформлення пояснювальної записки та документів до неї. Нормоконтроль та рецензування	11 тиждень	Додатки
8	Захист роботи	12 тиждень	

Студент(ка)

_____ Марія АУБАКІРОВА
(підпис) (Ім'я ПРІЗВИЩЕ)

Керівник проєкту (роботи)

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

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи магістра:
103 с., 35 рис., 4 табл., 3 дод., 12 джерел.

ТВАРИНА, ПОРОДА, ПРИТУЛОК, РЕЄСТРАЦІЯ, ЗАДАЧА РЕГРЕСІЇ, XGBOOST.

Об'єкт роботи – процес діяльності притулку для тварин. Предмет роботи – методи підтримки планування діяльності притулку для тварин. Мета роботи – розробити програмне забезпечення підтримки діяльності притулку для тварин для забезпечення можливостей планування діяльності на основі визначення очікуваної тривалості перебування тварини у притулку.

Матеріали, методи та технічні засоби: мова Python, бібліотека XGBoost, система MySQL, фреймворк Django.

Результати. Створена програма підтримує діяльність притулку для тварин, забезпечує реєстрацію дій, які виконуються в притулку для тварин, внесення даних про самих тварин у притулку і їхніх господарів, створення повідомлень господарями тварин про їх втрату, пошук серед повідомлень і реєстрацій, визначення відповідностей між повідомленнями господарів та реєстраціями тварин у притулках для тварин, планування діяльності притулків для тварин на основі визначення очікуваної кількості місяців перебування кожної тварини у притулку після її прибуття.

Висновки. Наукова новизна роботи полягає в методі прогнозування періоду перебування тварини у притулку, який застосовує для прогнозування модель, побудовану за допомогою ансамблю дерев рішень на основі алгоритму градієнтного бустингу, виконуючи визначення періоду перебування на основі кількості днів та встановлюючи порядок формування даних для навчання моделі.

Галузь використання. Підтримка роботи притулків для тварин.

ABSTRACT

Explanatory note to the diploma qualifying work of the master: 103 pages, 35 figures, 4 tables, 3 appendixes, 12 sources.

ANIMAL, BREED, SHELTER, REGISTRATION, REGRESSION PROBLEM, XGBOOST.

The object of the work is the activity process of the animal shelter. The subject of the work is methods of supporting the planning of animal shelter activities. The purpose of the work is to develop software to support the activity of an animal shelter to provide the ability to plan activities based on the determination of the expected duration of an animal's stay in the shelter.

Materials, methods and technical means: Python language, XGBoost library, MySQL system, Django framework.

The results. The created program supports the activity of the shelter for animals, provides registration of actions performed in the shelter for animals, entering data about the animals themselves in the shelter and by their owners, creating reports by owners of animals about their loss, searching among reports and registrations, determining correspondences between reports of owners and registrations of animals in shelters for animals, planning activities of shelters for animals based on the determination of the expected number of months of stay of every animal in the shelter after its arrival.

Conclusions. The scientific novelty of the work is in the method of forecasting the period of the animal's stay in the shelter, which uses a model built using an ensemble of decision trees based on the gradient boosting algorithm for forecasting, determining the period of stay based on the number of days and establishing the procedure of data generation for model training.

Field of use. Support for animal shelters.

ЗМІСТ

	С.
Перелік скорочень та умовних познач.....	8
Вступ.....	9
1 Аналіз проблеми та постановка завдань дослідження	10
1.1 Огляд програмного забезпечення підтримки діяльності притулку для тварин.....	10
1.2 Порівняння аналогів	14
1.3 Висновки за розділом 1	15
2 Матеріали і методи.....	16
2.1 Задача прогнозування періоду перебування тварини у притулку	16
2.2 Метод прогнозування періоду перебування тварини у притулку.....	19
2.3 Висновки за розділом 2	20
3 Проєктування програмного забезпечення	22
3.1 Вибір рішень для веброзробки.....	22
3.2 Моделювання програмного забезпечення на основі функціональних вимог	23
3.3 Проєктування структури бази даних.....	29
3.4 Висновки за розділом 3	33
4 Основні рішення щодо реалізації програмного забезпечення	35
4.1 Порядок роботи програмного забезпечення	35
4.2 Рішення з реалізації вебзастосунку	37
4.3 Висновки за розділом 4	52
5 Експлуатація, тестування та експериментальне дослідження програми	53
5.1 Опис програми.....	53
5.2 Експлуатація програми.....	53
5.3 Експериментальне дослідження	58
5.4 Висновки за розділом 5	61
Висновки	63
Перелік джерел посилання	64

Додаток А Технічне завдання	65
Додаток Б Текст програми	71
Додаток В Слайди презентації.....	95

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

CART – Classification And Regression Tree;

XGBoost – eXtreme Gradient Boosting;

БД – база даних.

ВСТУП

Притулки для тварин діють у постійному циклі, адже коли тварина знаходить свого старого або нового господаря, притулок може розмістити в себе іншу тварину. Тому знання того, коли можна очікувати вибуття тварини з притулку, є важливим для планування діяльності притулку для тварин, адже дозволяє розуміти, з якою кількістю тварин вдасться попрацювати притулку за деякий період. Окрім самого планування при цьому потрібно забезпечити внесення всіх пов'язаних даних, зручний пошук, надати певні можливості не тільки працівникам притулків для тварин, але і господарям тварин. Програму, яка буде підтримувати ці сторони роботи притулків для тварин, потрібно створити в роботі. Основною її направленістю є підтримка роботи притулку для тварин, однак шляхом надання як додаткових засобів можливостей для звичайних користувачів можна привернути увагу господарів тварин, таким чином сприяючи результативному у оперативному поверненню тварин власникам.

Мета роботи – розробити програмне забезпечення підтримки діяльності притулку для тварин для забезпечення можливостей планування діяльності на основі визначення очікуваної тривалості перебування тварини у притулку. Щоб виконати поставлену мету роботи, визначено для виконання такі завдання:

- проаналізувати аналоги програми підтримки діяльності притулку для тварин;
- розробити метод прогнозування періоду перебування тварини у притулку;
- виконати проєктування програмного забезпечення підтримки діяльності притулку для тварин;
- реалізувати програмне забезпечення підтримки діяльності притулку для тварин.

1 АНАЛІЗ ПРОБЛЕМИ ТА ПОСТАНОВКА ЗАВДАНЬ ДОСЛІДЖЕННЯ

1.1 Огляд програмного забезпечення підтримки діяльності притулку для тварин

Аналізуючи вебзасоби, які доступні користувачам в Україні, в першу чергу було виділено вебсайт притулку для тварин «Сіріус» [1], зокрема зважаючи на те, що це найбільший притулок для тварин в Україні (рис. 1.1). Вебсайт направлений на надання інформації користувачам про притулок, щоб стимулювати з одного боку зібрати кошти для роботи притулку, а з іншого – знайти нових господарів тваринам. Надаються також можливість пошуку за тваринами з наявністю ряду фільтрів, збереження тварин у переліку, щоби обрати, яку з них користувач бажає всиновити. Тобто даний вебсайт направлений на надання базових засобів саме для звичайних людей, які можуть певним чином комунікувати, допомагати притулку, при цьому робота самого притулку через нього окрім як отримання даних про тварин у притулку не виконується.

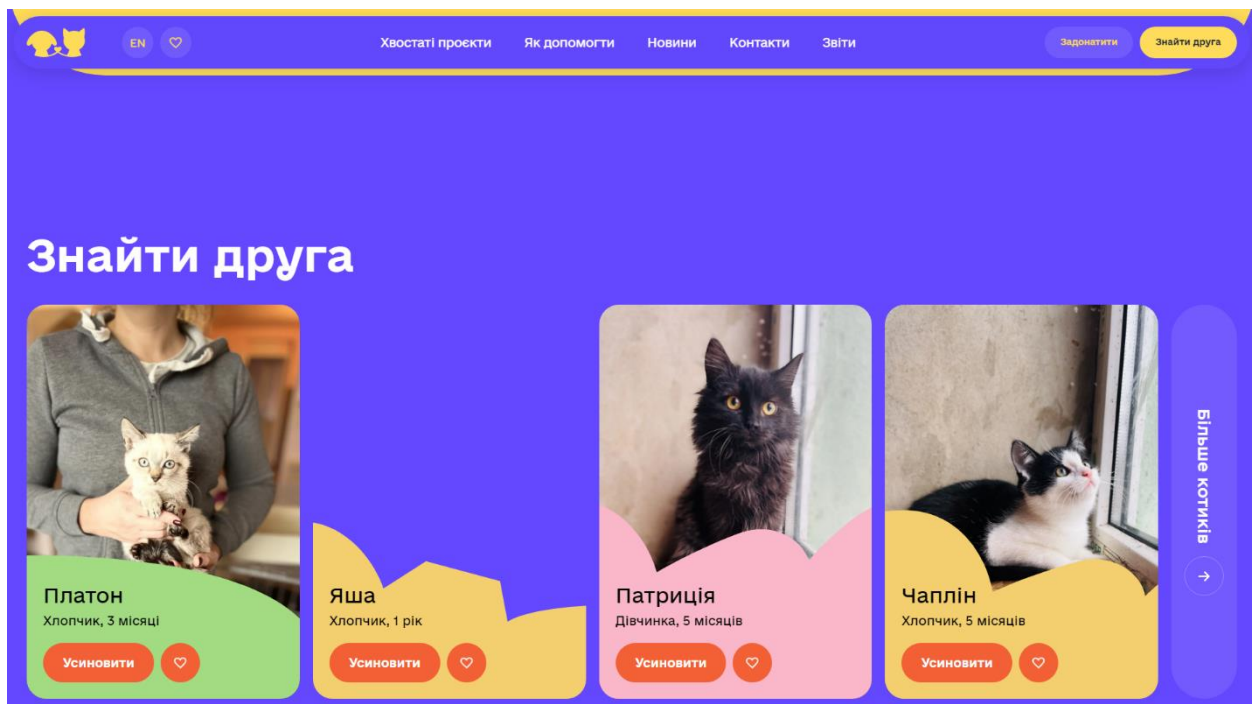


Рисунок 1.1 – Вебсайт притулку для тварин «Сіріус»

Притулок для тварин «В добрі руки» [2] організував роботу свого вебсайту подібним чином до попереднього (рис. 1.2), але наявні декілька важливих відмінностей. Вебсайт не дозволяє створити список своїх улюбленців, проте він дозволяє значно детальніше ознайомитись з кожною твариною. Так вказується окремо порода, характер тварини, надається її опис, тобто кожна тварина представлена власною вебсторінкою.

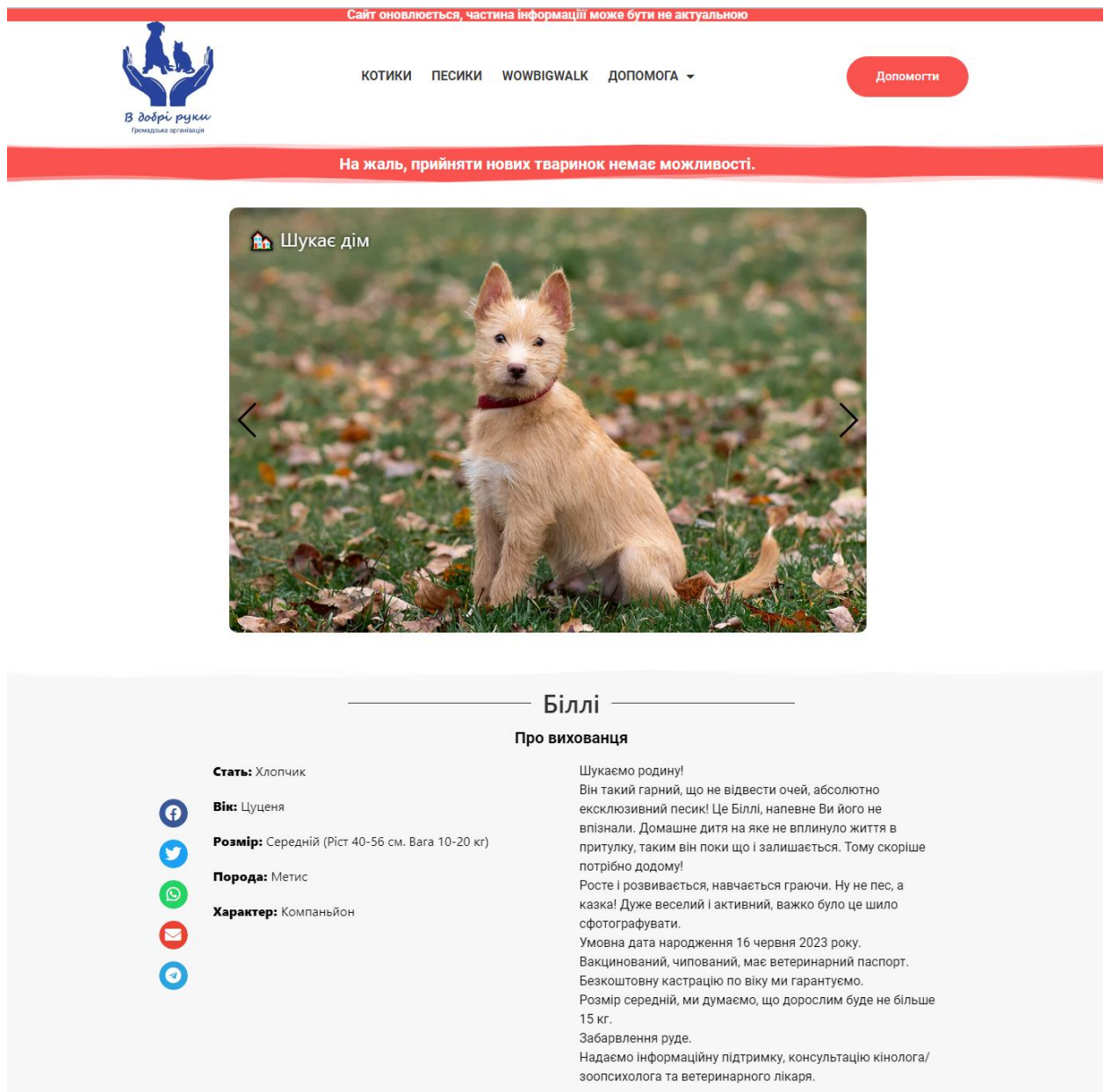


Рисунок 1.2 – Вебсайт притулку для тварин «В добрі руки»

Вебсайт також має пошук тварин, який здійснюється за обраною категорією тварин, але кількість фільтрів є невеликою порівняно з попереднім вебсайтом. Недоліком є те, що вказано в шапці вебсайту: частина інформації на вебсайті може бути не актуальною, відповідно можливості отримати інформацію про тварину одразу після її приймання в притулок, виконати швидко пошук власної тварини немає, тобто інформація з'являється на вебсайт з певною затримкою, що загалом можна вважати в розрізі пошуку власної тварини, яка загубилась, негативною характеристикою.

Існує також вебсайт для пошуку тварин з притулків PetHelp [3] (рис 1.3).

[Всеукраїнський он-лайн каталог тварин, що проживають в притулках](#) > Пошук тварин

1

Знайдіть

Бронза	Гор	Ванкувер	Арон
Хлопчик • Щуця	Хлопчик • Щуця	Хлопчик • Щуця	Хлопчик • Щуця
Розмір: Середній	Розмір: Середній	Розмір: Середній	Розмір: Середній
Область: Київська	Область: Київська	Область: Київська	Область: Київська
Притулок: Центр захисту тварин	Притулок: Центр захисту тварин	Притулок: КП "Київська Міська Лікарня Ветеринарної Медицини"	Притулок: КП "Київська Міська Лікарня Ветеринарної Медицини"

Рисунок 1.3 – Вебсайт для пошуку тварин з притулків PetHelp

PetHelp відрізняється тим, що він дозволяє авторизуватися представникам притулків для тварин, внести дані про тварин, які у них наявні, а звичайні користувачі тоді можуть шукати тварин, які було внесено, шукати притулки.

Аналізуючи іноземні рішення, було виокремлено вебзастосунок Petfinder [4], який спеціалізується на пошуку тварин для подальшого всиновлення (рис. 1.4). Він надає можливість пошуку тварин, за кожною твариною надає окрему інформацію, окрім того надає дані про вартість всіх процедур, що характерно для інших країн, адже приймання, утримання тварини у себе там пов'язано зі сплатою певних податків. Вебсайт надає ряд матеріалів, які можуть бути корисними для людей, які збираються всиновити тварину.

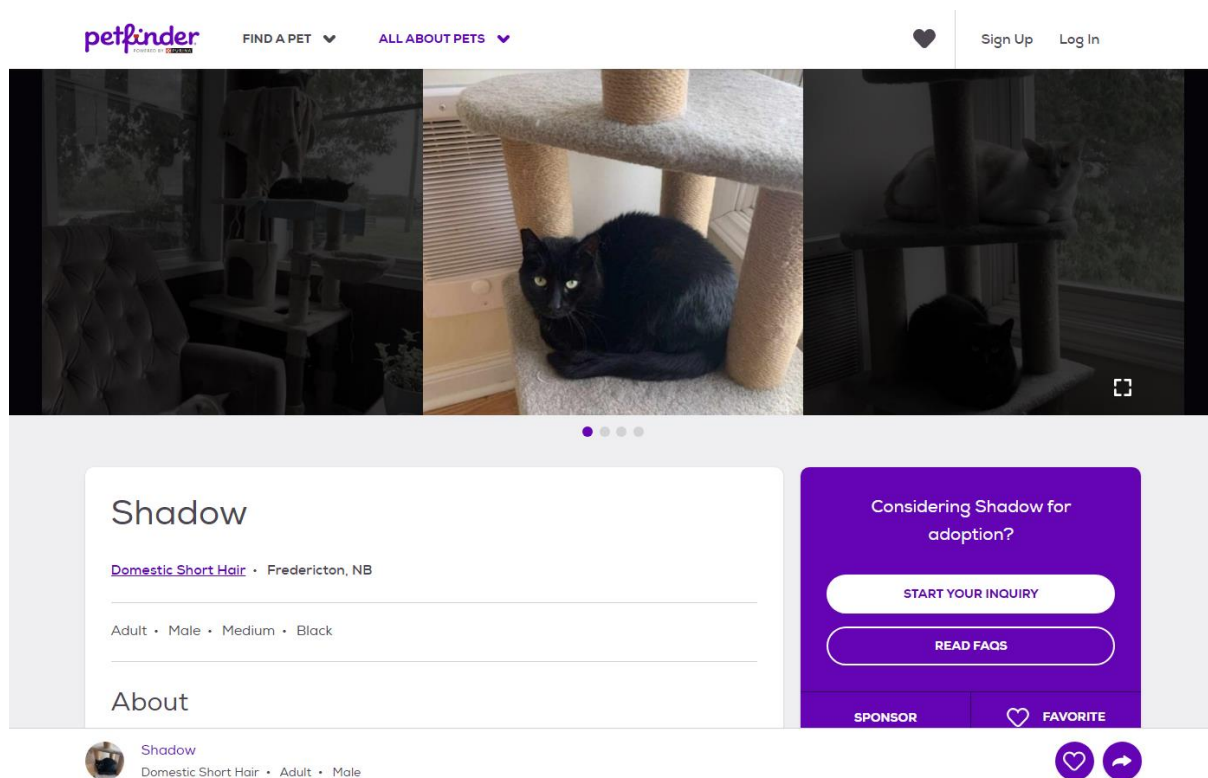


Рисунок 1.4 – Вебзастосунок Petfinder

Тож загалом існуючі програми мають достатньо привабливий інтерфейс, базові засоби роботи, проте для них характерно те, що навіть за наявності можливості роботи представників різних притулків з програмою,

потрібно спочатку в своєму притулку вести певну реєстрацію тварин, а тоді вносити цих тварин окремо у програму, що призводить до зменшення активності використання таких програм, збільшення періоду на внесення даних у програму, а відповідно і не можливості шукати власну тварину одразу через програму, якщо вона загубилась.

1.2 Порівняння аналогів

Основні особливості і можливості аналогів були розписані вище, загальне порівняння ж продемонструвало відмінності програми, яка створюється в роботі, від аналогів (табл. 1.1).

Таблиця 1.1 – Порівняння програмного забезпечення підтримки діяльності притулку для тварин

Критерій	Cіріус	PetHelp	Petfinder	Anisheltersoft
Пошук тварин з притулків	+	+	+	+
Докладна інформація про тварину	—	+	+	+
Доступ різних притулків до програми	—	+	+	+
Реєстрація тварин притулками одразу в програмі	—	—	—	+
Реєстрація процедур лікування тварин	—	—	—	+
Прогнозування періоду перебування тварини у притулку	—	—	—	+

Аналоги не мають можливостей безпосередньої підтримки базової діяльності притулків через програму. Інтеграція цих можливостей має дозволити підвищити зручність роботи представників притулків через уникнення дублювання. Одночасно має підвищитись оперативність отримання інформації про зареєстрованих у притулках тварин. Це в свою чергу має збільшити простір використання програми до пошуку господарями власних тварин, які могли загубитись і потрапити у притулок.

1.3 Висновки за розділом 1

Виконаний огляд і аналіз аналогів, включав вебсайти притулків для тварин «Сіріус» та «В добрі руки», вебсайт для пошуку тварин з притулків PetHelp, вебзастосунок Petfinder. Аналіз вказав на наявну неоперативність оновлення інформації, необхідність дублювання інформації у таких програмах представниками притулків, тому програма, що створюється, за рахунок підтримки базових процесів роботи притулку для тварин, має підвищити зручність цієї роботи і досягти вищої оперативності.

2 МАТЕРІАЛИ І МЕТОДИ

2.1 Задача прогнозування періоду перебування тварини у притулку

Притулки для тварин не є прибутковими організаціями, а тому планування діяльності притулків для тварин передбачає не визначення прибутку, доходу, витрат організації, як це буває у випадку комерційних організацій. Витрати звичайно, що є важливими, але вони більшою мірою залежить від кількості і типів місць у притулку, що в свою чергу визначає необхідну кількість обслуговуючого персоналу. Тож це в підсумку дозволяє розрахувати стандартні витрати. До нестандартних витрат належать витрати на лікування, можливе збільшення необхідного обсягу корму тощо. Однак ці причини пов'язані з конкретними тваринами, які перебувають у притулку. Наприклад, якщо нові тварини з'являються достатньо часто, то витрати на лікування можуть зростати, адже кожна нова тварина може потребувати певних процедур. Тож у підсумку визначення часу, протягом якого тварина перебуває у притулку, є критичним фактором, адже дозволяє чіткіше планувати витрати, дозволяє розуміти кількість вільних місць на певний період, планувати відповідно взаємодію з іншими притулками, які можуть брати до себе тварин, якщо не вистачає місць у притулку, з людьми, які готові взяти до себе тварин. Тому в даній роботі було вирішено розв'язати саме задачу прогнозування періоду перебування тварини у притулку.

Задача прогнозування періоду перебування тварини у притулку полягає в тому, що визначити кількість днів, протягом якої тварина буде перебувати у притулку до того моменту, як вона буде віддана новому господарю на постійне або тимчасове проживання. Для прогнозування кількості днів перебування тварини в притулку використовується певний набір ознак, які характеризують цю тварину або її потрапляння у притулок. Серед них необхідно для розв'язання задачі визначити ті вхідні ознаки, які дозволять сформулювати модель прогнозування періоду перебування тварини у

притулку. До таких потенційних ознак, вивчивши відповідні вибірки даних, можна віднести:

- дата потрапляння тварини у притулок;
- причина потрапляння тварини у притулок;
- порода тварини;
- категорія тварини;
- вік тварини на момент потрапляння у притулок;
- колір шерсті тварини.

При цьому додатково потрібно визначити, яким чином представити дату потрапляння тварини у притулок, адже дата потребує певного перетворення.

Дана задача прогнозування періоду перебування тварини у притулку є задачею, яка в машинному навчанні називається задачею регресії. У такому випадку виконується прогнозування кількості днів, протягом якої тварина буде перебувати у притулку, на основі даних, отриманих під час потрапляння тварини у притулок. Розв'язання задачі призводить до створення моделі прогнозування. Модель має бути створена на основі даних, які є історичними і представляють дані про тварин, які раніше потрапляли в ті самі притулки і за якими відома кількість днів перебування тварини у притулку та значення характеристик, визначених під час їхньої реєстрації в притулку. Ці історичні дані називаються навчальною вибіркою даних. Для розв'язання задачі необхідно спочатку сформувати вибірку даних.

Розв'язання задач регресії може виконуватися на основі дерев рішень, які можуть застосовуватися не тільки для класифікації, але і для регресії, такими деревами є Classification And Regression Tree (CART) – техніка машинного навчання, що будує бінарні дерева, щоб розділити наявний простір ознак на сегменти відносно цільової змінної, створюючи деревоподібну структуру на основі реалізації рекурсивного алгоритму [5].

Створення складніших моделей потребує організації ансамблів дерев рішень, одним з варіантів створення яких є eXtreme Gradient Boosting (XGBoost). Він побудований на основі градієнтного бустингу.

У бустингу дерева будуються, що і є його головною відмінністю, послідовно таким чином, що кожне наступне дерево спрямоване на зменшення помилок попереднього дерева, тобто кожне дерево навчається на основі результатів своїх попередників, оновлюючи залишкові похибки, тому дерево, яке з'являється у цій послідовності навчання наступним, навчається з оновленого результату навчання, відповідно базові дерева лише трохи кращі, ніж випадкове вгадування, проте найважливішим є те, що кожне з цих початкових дерев надає важливу інформацію для прогнозування [6]-[7].

Алгоритм градієнтного бустингу під час реалізації методу XGBoost серед всіх параметрів роботи зокрема дозволяє встановлювати наступні:

- швидкість навчання (learning rate) дозволяє скоротити внесок кожного дерева, при цьому впливаючи зокрема на кількість ітерацій під час реалізації навчання, а менші значення вимагають більше ітерацій, при цьому загалом можуть покращити узагальнення створених моделей;

- кількість оцінювачів або кількість раундів бустингу, що фактично визначає кількість дерев у ансамблі дерев рішень: збільшення кількості дерев рішень у ансамблі може підвищити продуктивність створеної моделі, але одночасно і збільшує час навчання, тому потрібно встановити те значення, яке відповідає найкращому варіанту моделі в підсумку;

- максимальна глибина кожного дерева в ансамблі дерев: глибші дерева можуть моделювати складніші взаємодії між вхідними та вихідними ознаками, але можуть бути надмірними, оптимальне значення залежить від складності проблеми, наявних даних, зазвичай значення встановлюються в інтервалі від 3 до 10 [8].

2.2 Метод прогнозування періоду перебування тварини у притулку

Метод прогнозування періоду перебування тварини у притулку розпочинається зі збору даних, включаючи наступні властивості:

- дата потрапляння тварини у притулок;
- порода тварини;
- категорія тварини;
- дата реєстрації потрапляння тварини у притулок;
- дата реєстрації вибуття тварини з притулку.

З вибраних даних усі дублікати мають бути вилучені. Однак, до дублікатів відносяться тільки повністю однакові екземпляри даних з вибірки. Тобто це екземпляри, які стосуються однакових тварин, а також того самого їхнього потрапляння в той самий притулок. Якщо це різні потрапляння тієї самої тварини у притулки, які відбулись в різні періоди часу, то усунення таких дублювань не має відбуватися.

Усі реєстрації тварин у притулках, які не містять значення хоча би однієї з наведених вище характеристик, мають бути вилучені, окрім ситуацій, коли ці значення можна уточнити.

Дата потрапляння тварини у притулок має бути перетворена на кількість днів, яка пройшла з найпершого зареєстрованого потрапляння тварин у притулок у отриманій вибірці даних. Для цього відповідно всі реєстрації потрапляння тварин у притулки мають бути впорядковані за часом, а найперша реєстрація виділена окремо.

Тривалість перебування тварини у притулку має бути обчислена на основі визначення кількості днів, які пройшли між датою реєстрації потрапляння тварини у притулок та датою вибуття тварини з притулку (усиновлення тварини іншим господарем, взяття тварини для тимчасового утримання).

Створення моделі прогнозування періоду перебування тварини у притулку має бути виконано на основі створення ансамблю дерев рішень на основі методу XGBoost. За результатами експериментального дослідження модель створена на основі використання 40 дерев та створення дерев з максимальною глибиною 6.

Після створення моделі навчання її виконується на основі навчальної вибірки з визначенням тривалості перебування тварини у притулку для кожного екземпляра в якості вихідної ознаки та наступних вхідних ознак моделі:

- дати потрапляння тварини у притулок у вигляді кількості днів, що пройшла з першої реєстрації тварини у притулку, що було зареєстровано;
- порода тварини;
- категорія тварини.

На наступному етапі визначаються нові зареєстровані випадки реєстрації тварин у притулку зі встановленням дати потрапляння тварини у притулок описаним вище способом, визначенням категорії та породи тварин. Для кожного такого випадку подаються на вхід моделі прогнозування періоду перебування тварини у притулку отримані значення нових реєстрацій, а на виході отримується кількість днів, через яку тварина вибуде з притулку для тварин.

2.3 Висновки за розділом 2

Задачу прогнозування періоду перебування тварини у притулку представлено як задачу регресії під час машинного навчання. Для її розв'язання було проаналізовано можливості створення дерев рішень, включаючи окремі дерева на основі моделей CART та ансамблі дерев рішень, побудовані на основі методу XGBoost.

Наукова новизна роботи полягає в методі прогнозування періоду перебування тварини у притулку, який застосовує для прогнозування модель,

побудовану за допомогою ансамблю дерев рішень на основі алгоритму градієнтного бустингу, виконуючи визначення періоду перебування на основі кількості днів та встановлюючи порядок формування даних для навчання моделі.

3 ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Вибір рішень для веброзробки

Розробку програми виконано як веброзробку, а порівняння основних технологій зведено до порівняння Django [9] і Joomla [10] (табл. 3.1) як засобів для веброзробки, використовуючи для порівняння наступні критерії:

- ліцензія: визначає права подальшого використання;
- рівень забезпечення безпеки: для розподілення прав, збереження даних користувачів важливим є високий рівень безпеки, який в Django є одразу після створення проєкту;
- складність підтримки великих програм: програма має широку функціональність, яка з часом як і кількість користувачів має збільшуватись, тому цей критерій є важливим, а Django дозволяє таке розширення виконувати простіше;
- рівень можливостей інтегрування моделей прогнозування: мова Python має багато готових бібліотек для створення моделей прогнозування, що без додаткових проблем інтегруються в проєкти Django.

Таблиця 3.1 – Порівняння Django і Joomla для веброзробки

Критерій	Django	Joomla
Використовувана мова програмування	Python	PHP
Ліцензія	Відкрита	Відкрита
Рівень забезпечення безпеки	Високий	Низький
Складність підтримки великих програм	Нижча	Вища
Рівень можливостей інтегрування моделей прогнозування	Високий	Вбудований низький

3.2 Моделювання програмного забезпечення на основі функціональних вимог

Функціональні вимоги, які програма має забезпечити для роботи неавторизованого користувача (рис. 3.1), який не має акаунта в програмі або не авторизований за ним у цей момент:

- пошук серед повідомлень про втрату тварини;
- пошук серед реєстрацій потрапляння тварин в притулок;
- демонстрування картки тварини;
- демонстрування даних притулку для тварин;
- відображення зареєстрованих у програмі притулків для тварин;
- демонстрування останніх зареєстрованих повідомлень про втрату тварин у місті;
- авторизація, до якого є альтернативний сценарій – реєстрація, що застосовується тоді, коли користувач взагалі не має власного акаунту в програмі;
- демонстрування повідомлення про втрату тварини.
- демонстрування реєстрації потрапляння тварини в притулок (деталізоване представлення).

Господар тварини використовує програму після авторизації в ній, відповідно початково він повинен мати акаунт, а тоді він має всі функціональні вимоги неавторизованого користувача та наступні додаткові вимоги:

- внесення даних про власну тварину користувачем у програму, включаючи додавання картки, до якого є два альтернативні сценарії – вилучення існуючої картки та її редагування;
- визначення відповідності між твариною з повідомлення та твариною з притулку, до якого є альтернативний сценарій – вилучення відповідності між твариною з повідомлення та твариною з притулку;
- демонстрування картки лікування тварини;

- демонстрування відповідностей за карткою тварини;
- демонстрування створених користувачем, який авторизувався, відповідностей;
- створення повідомлення про втрату тварини, до якого є два альтернативні сценарії – редагування повідомлення про втрату тварини, коли повідомлення було створено раніше, та видалення повідомлення про втрату тварини;
- демонстрування власних повідомлень про втрату тварин господарем, який авторизувався;
- демонстрування всіх зареєстрованих дій щодо тварини;
- демонстрування карток власних тварин господаря (рис. 3.2).

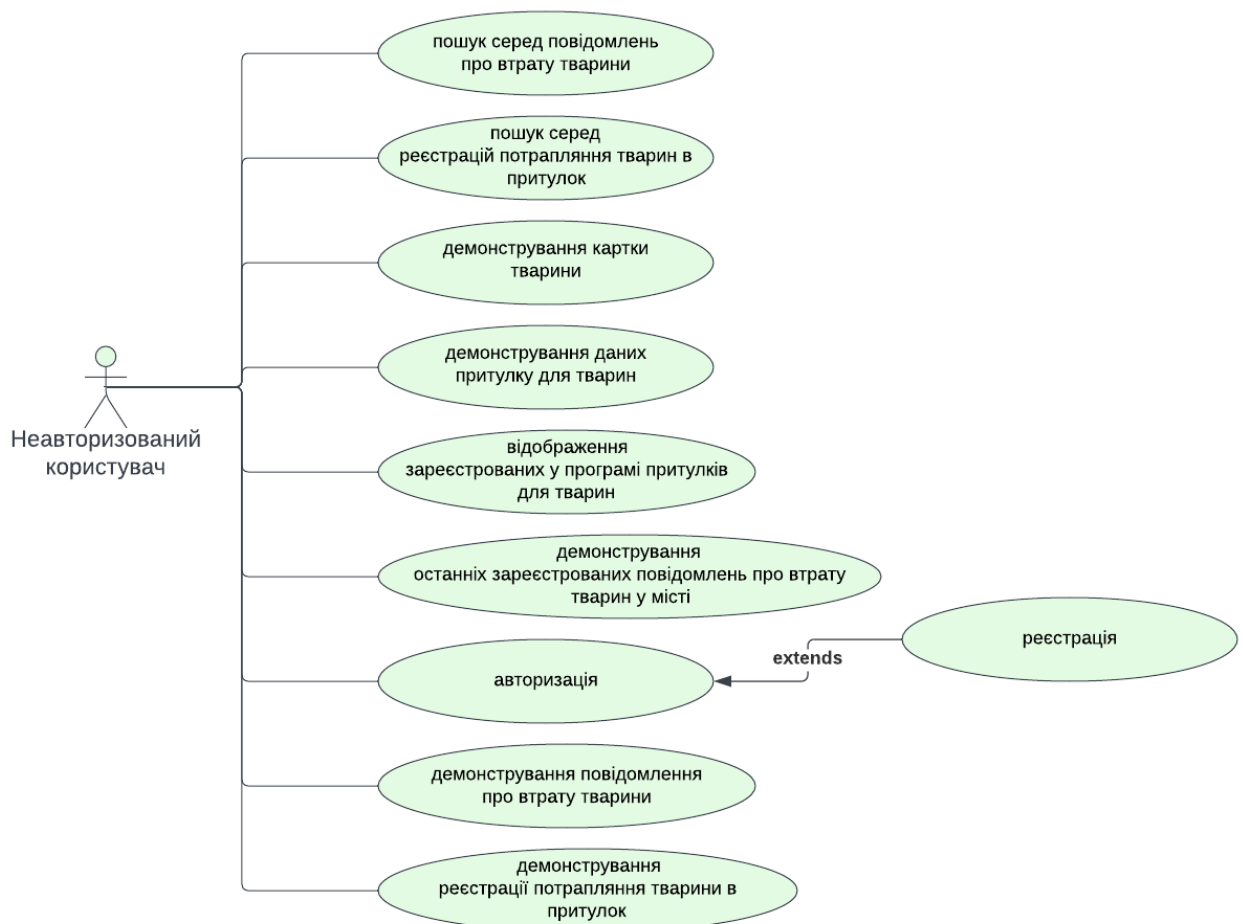


Рисунок 3.1 – Діаграма прецедентів для неавторизованого користувача

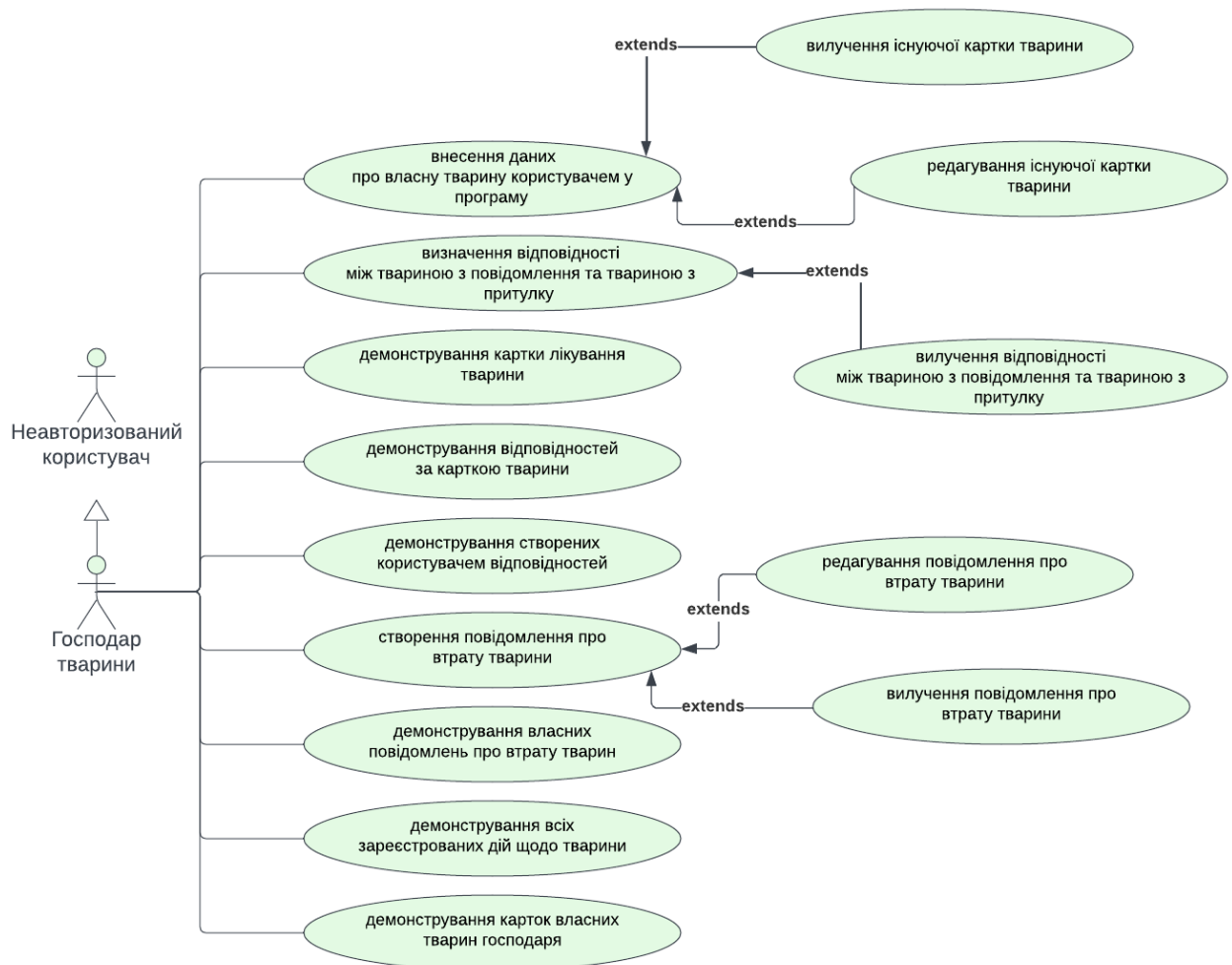


Рисунок 3.2 – Діаграма прецедентів для господаря тварини

Притулки для тварин у програмі можуть представлятися менеджерами притулків для тварин, що є узагальнюючою роллю, адже в залежності від конкретної ролі це може бути людина, яка відповідає за реєстрацію в притулку, лікар, який вносить сам лікування в програму, або представник (менеджер) лікаря. Функціональні вимоги, які програма має забезпечити для роботи менеджера притулку для тварини включають функціональні вимоги для неавторизованого користувача та наступні додаткові функціональні вимоги:

- реєстрація потрапляння тварини в притулок, включаючи прогнозування часу перебування тварини в притулку, до якого є два альтернативні сценарії – редагування даних реєстрації потрапляння тварини в притулок, вилучення даних незафіксованої реєстрації потрапляння тварини

в притулок (вилучити реєстрацію можна до того моменту, поки вона не буде підтверджена для остаточного збереження);

- внесення даних про тварину, яка потрапила в притулок, включаючи додавання картки, до якого є два альтернативні сценарії – вилучення існуючої картки та її редагування;

- демонстрування менеджерів, що працюють у притулку для тварин;

- визначення відповідності між картками про потенційно однакових тварин, до якого є альтернативний сценарій – вилучення відповідності між картками про потенційно однакових тварин;

- фіксація лікування тварини, до якого є альтернативний сценарій – редагування даних лікування тварини;

- демонстрування картки лікування тварини;

- демонстрування останніх зареєстрованих тварин за притулком або за менеджером;

- визначення відповідності між твариною з повідомлення та твариною з притулку, до якого є альтернативний сценарій – вилучення відповідності між твариною з повідомлення та твариною з притулку;

- демонстрування відповідностей за карткою тварини;

- демонстрування створених користувачем відповідностей;

- пошук серед реєстрацій потрапляння тварин в притулок;

- пошук серед реєстрацій повернення тварин у притулок;

- пошук серед реєстрацій переведень тварин між притулками;

- пошук серед реєстрацій вибуття тварин з притулку;

- пошук серед реєстрацій за притулком;

- пошук серед фіксацій лікувань тварин;

- реєстрування повернення тварини в притулок, до якого є альтернативний сценарій – редагування даних повернення тварини в притулок;

- реєстрація переведення тварини в інший притулок, до якого є альтернативний сценарій – редагування даних реєстрації переведення тварини в інший притулок;
- демонстрування часу перебування тварин у притулку за категоріями;
- демонстрування кількості тварин у притулку;
- демонстрування реєстрацій за притулком;
- демонстрування лікувань тварин за менеджером;
- демонстрування останніх зареєстрованих повернутих тварин за притулком або за менеджером;
- демонстрування останніх зареєстрованих тварин за вибуттям за притулком або за менеджером;
- демонстрування останніх зареєстрованих тварин за переведенням з притулку;
- демонстрування останніх зареєстрованих тварин за переведенням до притулку;
- демонстрування останніх зареєстрованих тварин за переведеннями між притулками за менеджером;
- демонстрування всіх зареєстрованих дій щодо заданої менеджером притулку тварини;
- реєстрація вибуття тварини з притулку, до якого є альтернативний сценарій – редагування даних реєстрації вибуття тварини з притулку, коли реєстрація вже відбулась;
- додавання даних про місця для категорій тварин у притулку для тварин, до якого є альтернативний сценарій – редагування даних про місця для категорій тварин у притулку для тварин;
- редагування даних акаунту (рис. 3.3).

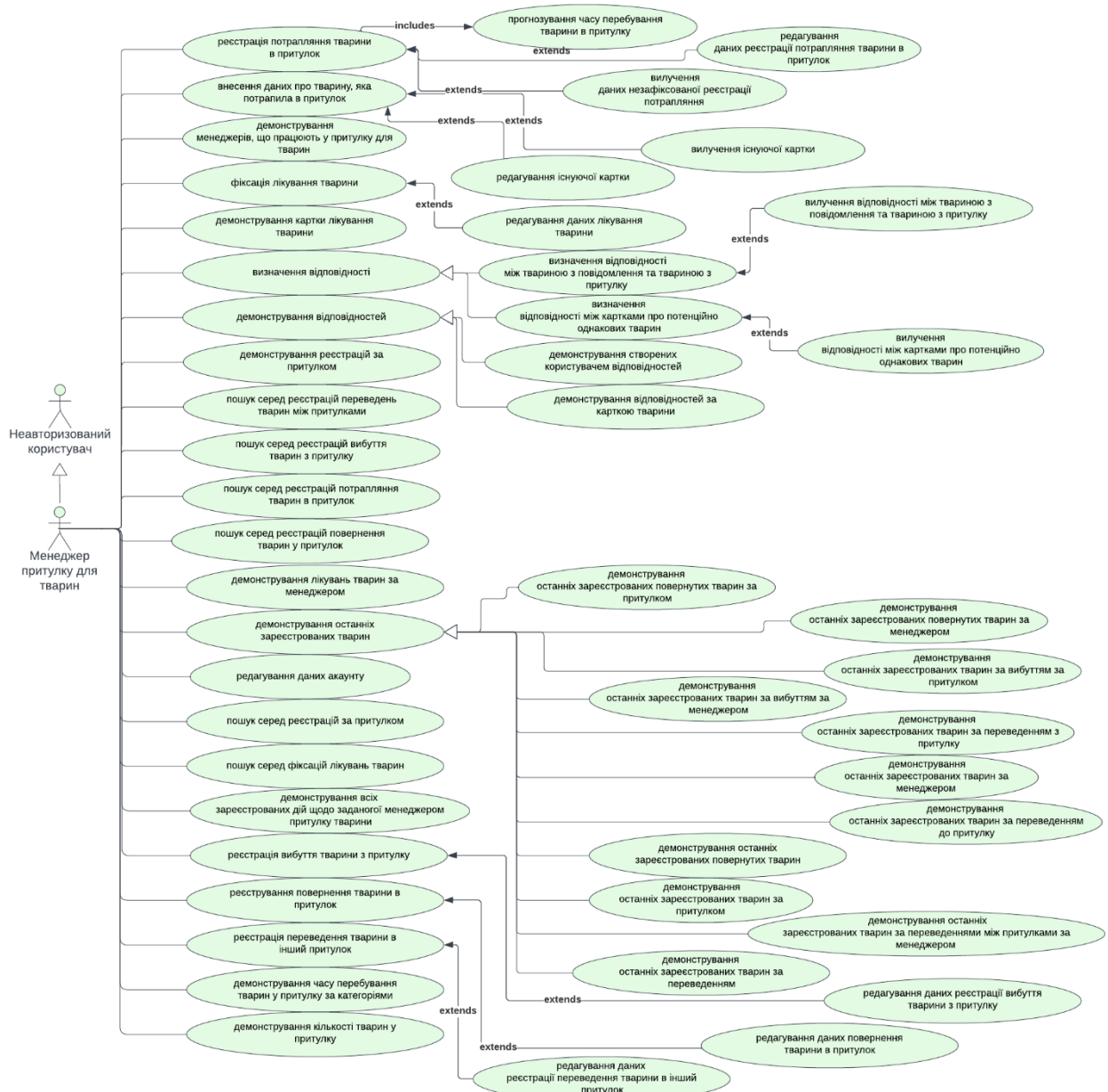


Рисунок 3.3 – Діаграма прецедентів для менеджера притулку для тварин

Адміністратору відповідають всі функціональні вимоги менеджера притулку для тварин та господаря тварини та наступні додаткові функціональні вимоги (рис. 3.4):

- реєстрація менеджера для роботи в притулку;
- реєстрація нового притулку для тварин в програмі, до якого є альтернативний сценарій – редагування даних притулку для тварин.

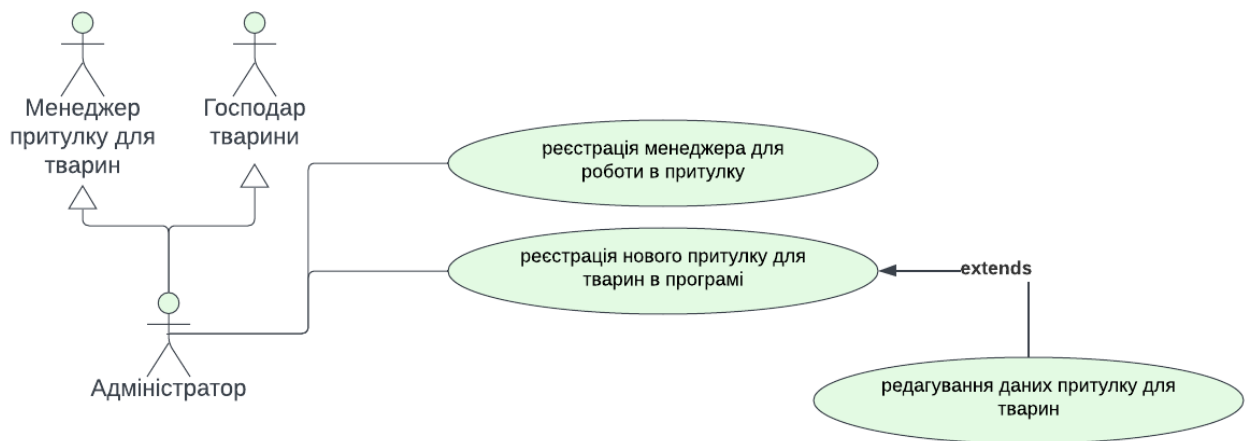


Рисунок 3.4 – Діаграма прецедентів для адміністратора

3.3 Проєктування структури бази даних

Проєктування бази даних (БД) потребувало створення ряду сутностей, які в подальшому в реляційній структурі були виділені у відповідні таблиці. Звичайно, що логічним було виділення за цією предметною областю спочатку наступних сутностей:

- притулок для тварин визначає об’єднання всіх наданих даних щодо тварин за місцями їхнього концентрування, саме притулки є основною одиницею функціонування програми, адже на планування їхньої діяльності направлена програма, відповідно основним джерелом внесення даних є самі притулки, тож відповідна таблиця БД Shelter містить саме ці дані;

- тварина є фактично конкретним об’єктом, на який направлена діяльність програми, адже обслуговування тварин є основною направленістю роботи притулків для тварин, а додатковою є пошук через програму господарями своїх загублених тварин, відповідно саме внесення інформації про тварин у такому випадку потребує господар, тож у підсумку внесення даних про тварину зведено до таблиці Pet, а інформація в таблиці про конкретну тварину далі називається карткою тварини;

- господар тварини є одним з користувачів програми: він може вносити дані про свою тварину, коли вона загубилась або за бажання і за відсутності такого факту, окрім того він може знаходити власну тварину в

реєстраціях в притулках для тварин, тож він фактично є одним з джерел впливу на внесені в картку тварини дані;

– менеджер притулку для тварин представляє притулки для тварин у програмі, адже самі притулки є фактично місцями концентрації тварин, але вони самі не можуть внести дані про тварин, про виконувані дії в притулку до програми, тому менеджери представляють їх у програмі та забезпечують виконання цих функцій, при цьому виконуючи певну роль, яка може бути або загальною, або специфічною (реєстрація, лікування тощо), у БД представлена таблицю Manager.

Для ідентифікації тварини окрім номеру мікрочіпу помічною інформацією може бути фотографія. Саме фотографії зазвичай і використовуються в оголошеннях про втрату тварин. Таких фотографій може бути надано декілька, тому для збереження самих фотографій потрібна окрема сутність Petphoto, в якій знаходяться фотографії тварини з таблиці Pet.

Для підтримки роботи основних сутностей і функцій, які забезпечуються для роботи цих функцій з програмою, створено певний довідник, який містить основні типи, категорії, які використовуються під час роботи програми для впорядкування інформації. До таких сутностей і відповідних таблиць відносяться наступні:

– категорія тварини, тобто кіт, пес або більш рідкісна категорія, що віднесена до таблиці Petcategory;

– порода тварини, що відповідає конкретній категорії тварини, віднесена до таблиці Animalbreed;

– стандартна причина потрапляння тварини до притулку, що віднесена до таблиці Registerreasontype і може бути представлена, наприклад, відмовою господаря, хворобою або смертю господаря, рішенням суду, втечею від господаря тощо;

– тип дій, за допомогою яких тварина потрапила у притулок як реакція на виявлену причину потрапляння тварини до притулку, віднесений до

таблиці Taketype і може бути, наприклад, бродячою твариною, конфіскацією, дії власника тощо;

- тип дій, за допомогою яких тварина покинула притулок після перебування в ньому, віднесений до таблиці Movementtype і може бути, наприклад, адопцією (усиновленням) тварини, прийманням тварини (тимчасова дія) або передачею в інший притулок;

- можливий тип лікування, який застосовується до тварини, віднесений до таблиці Therapytype, тобто це певна процедура, яка застосована до тварини;

- можлива причина повернення тварини, яка була в притулку, віднесена до таблиці Returnreason і може бути, наприклад, несумісністю проживання з цією твариною, відсутністю умов проживання тощо;

- місто, з яким можуть співвідноситись розташування притулку, знаходження тварини, віднесене до таблиці Locationcity;

- країна, з містом якої може відбуватися співвіднесення в програмі розташування притулку, знаходження тварини, віднесена до таблиці Locationcountry;

- можливий стан тварини, віднесений до таблиці Petcondition і може бути представлений, наприклад, значеннями здоровий, виліковний, не піддається лікуванню тощо.

Визначення відповідності між різними картками про тварин, які виявились або можуть виявитися однією і тією самою твариною, необхідно для того, щоб встановити, коли це можливо, що це однакова тварина, якщо тварина знову потрапляє до притулку, наприклад, після того, як була знайдена на вулиці, для уникнення дублювання, для отримання додаткової інформації про тварину, зокрема проведене лікування тощо. Такій сутності відповідає таблиця БД Petpossibility.

Визначення відповідності між створеним користувачем повідомленням про втрату тварини та реєстрацією потрапляння тварини в притулок відповідає таблиці БД Petlostadpossibility. Такі дії можуть

виконуватися менеджерами притулків, коли вони знайшли повідомлення стосовно тварини, яка знаходиться в притулку. Подібні дії можуть виконуватися звичайно і господарями тварин, коли вони знайшли інформацію про реєстрацію її в притулку. Такий варіант називається визначенням відповідності між власною твариною користувача та твариною в притулку, відповідає таблиці БД Shelterregisterpossibility.

Лікувальні процедури, які застосовуються до тварин, в притулках знаходяться в таблиці БД Pettherapy.

Притулки для тварин не завжди працюють зі всіма можливим тваринами, окрім того можуть мати певну кількість місць під певні категорії тварин, тому такі дані потрібно зберігати окремо, зокрема за допомогою таблиці Shelterpetcategory.

Дії, які в притулку виконуються щодо тварин, включають наступні варіанти:

- реєстрація потрапляння тварини в притулок у вигляді таблиці Shelterregister, коли нова тварина з'являється в притулку та виконується її приймання, визначення інформації про неї;

- реєстрація переведення тварини в інший притулок у вигляді таблиці Transfer, коли тварина вже перебуває в притулку, але притулок не має місць у подальшому для її знаходження, а за наявності місць у іншому притулку та наявності домовленостей тварина переводиться в інший притулок;

- реєстрація вибуття тварини з притулку у вигляді таблиці Movementregister, коли тварина забирається (усиновлюється) новим господарем або передається людині для тимчасового проживання, щоб потім хтось її усиновив;

- реєстрація повернення тварини людиною, яка взяла її до себе до того ж притулку у вигляді таблиці Movementregisterreturn, коли людина всиновила тварину, але вирішила через певний час від неї відмовитись і повернути її до притулку.

Окремою функцією є внесення даних користувачем про тварину, коли він її втратив, щоб знайти її. Це створення користувачем повідомлення про втрату тварини знаходиться в підсумку в таблиці БД Petlostad, а вже ці оголошення можуть, наприклад, визначатися подібними господарем тварини до реєстрації в притулку для тварин. Реалізацію схеми реляційної БД (рис. 3.5) виконано через MySQL.

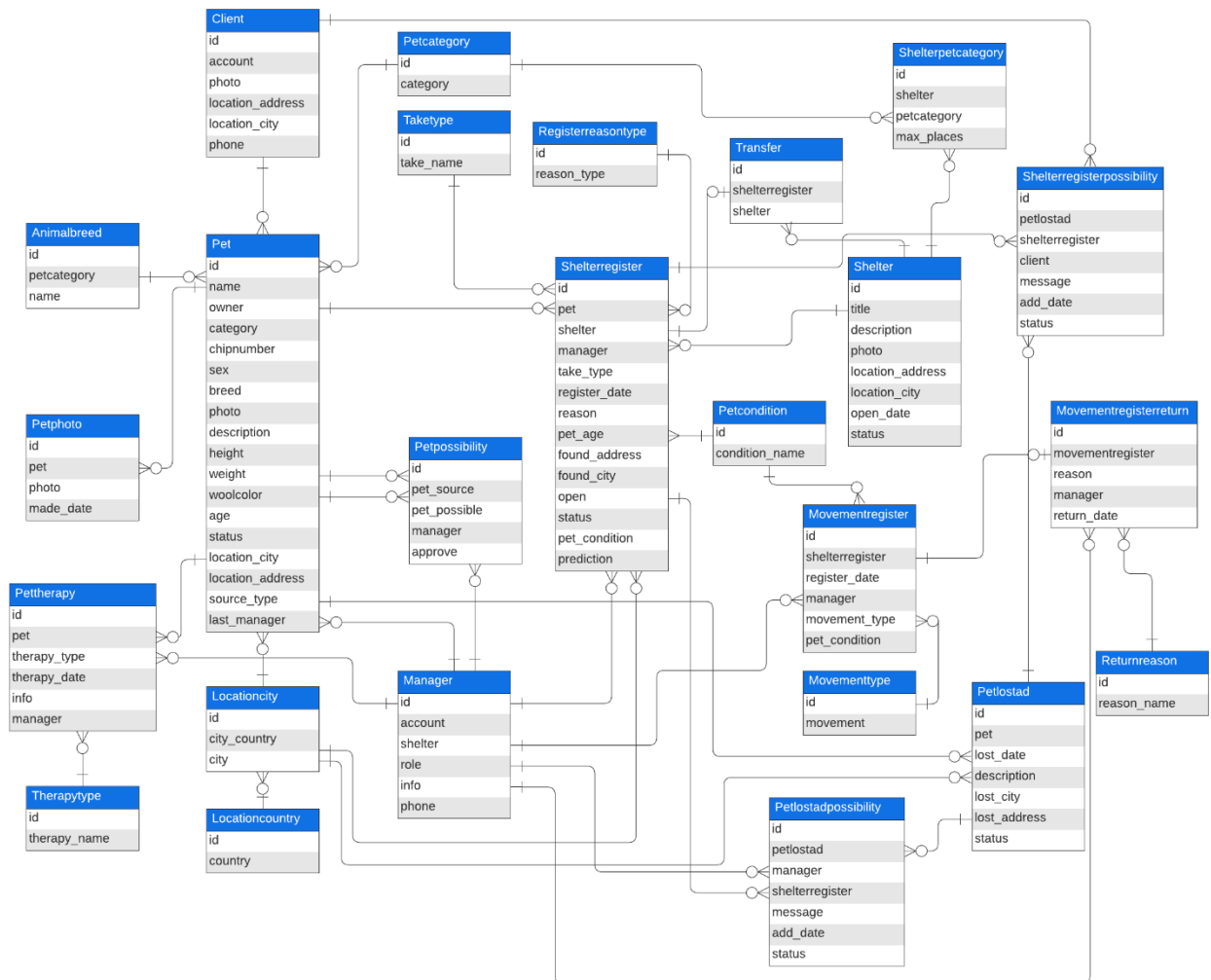


Рисунок 3.5 – Схема реляційної БД

3.4 Висновки за розділом 3

Забезпечено вибір стеку технологій для реалізації веброзробки, включаючи Python як мову програмування, яка підтримує і веброзробку, що вирішено виконувати з застосуванням фреймворку Django, і машинне

навчання, розширюючи його системою MySQL для організації роботи з даними на основі БД та визначаючи повну схему впорядкування даних у БД. Представлено ряд сценаріїв, у межах яких відбувається робота груп користувачів, включаючи менеджерів притулків для тварин, неавторизованих користувачів, господарів тварин, адміністраторів.

4 ОСНОВНІ РІШЕННЯ ЩОДО РЕАЛІЗАЦІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Порядок роботи програмного забезпечення

Сценарії з розділу 3 були впорядковані як послідовність роботи користувачів з програмою. Робота з програмою починається з пошуку серед реєстрацій тварин у притулках (рис. 4.1). Після відображення цієї сторінки можна перейти до інших варіантів роботи, але можна авторизуватися і тоді в залежності від типу акаунту виконувати доступні дії для господаря тварини (рис. 4.2) або для менеджера притулку для тварин (рис. 4.3).

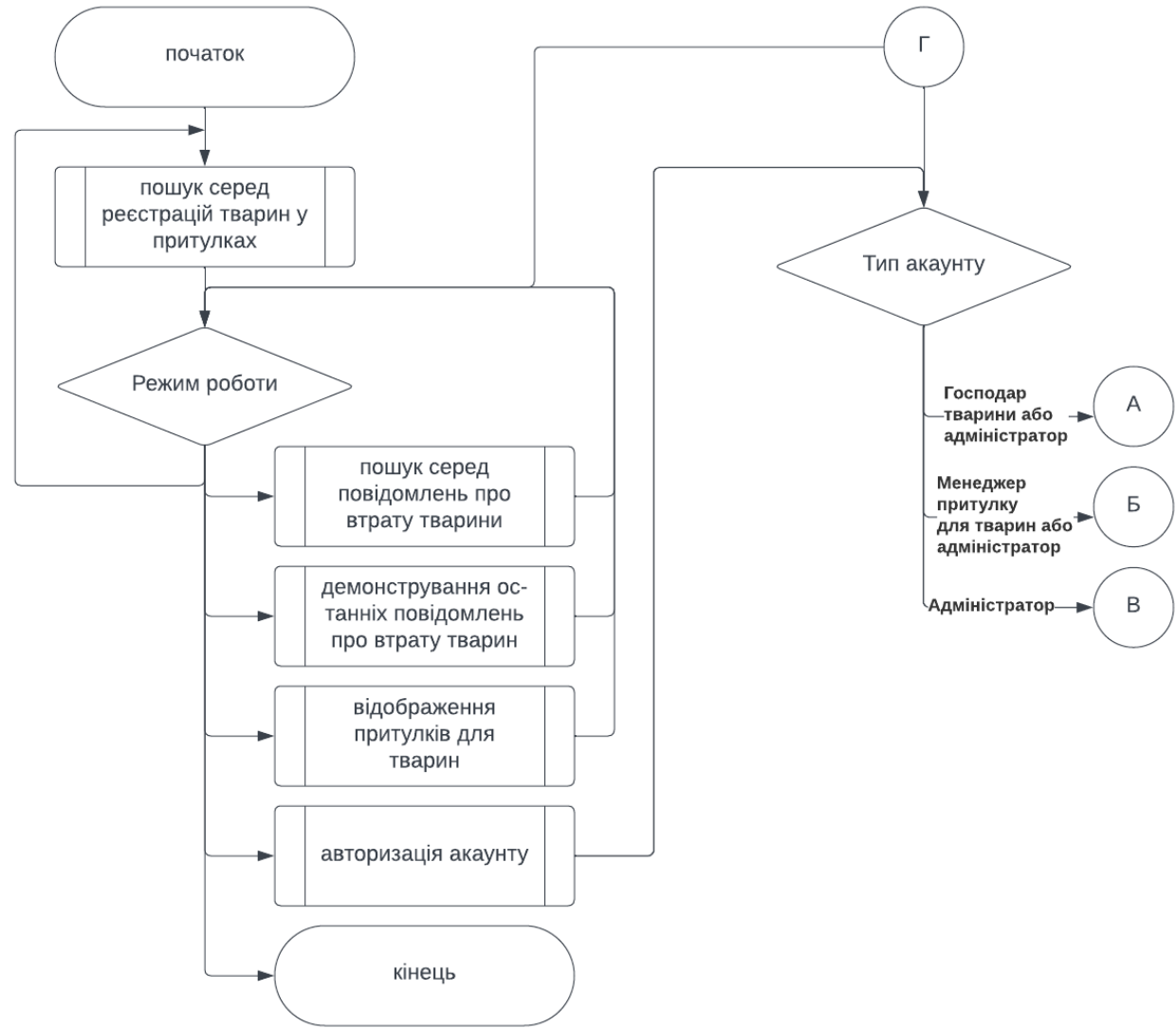


Рисунок 4.1 – Функціональна схема програми підтримки діяльності притулку для тварин



Рисунок 4.2 – Фрагмент функціональної схеми програми для господаря тварини



Рисунок 4.3 – Фрагмент функціональної схеми програми для менеджера притулку для тварин

Існує також декілька функцій доступних тільки для адміністратора (рис. 4.4).

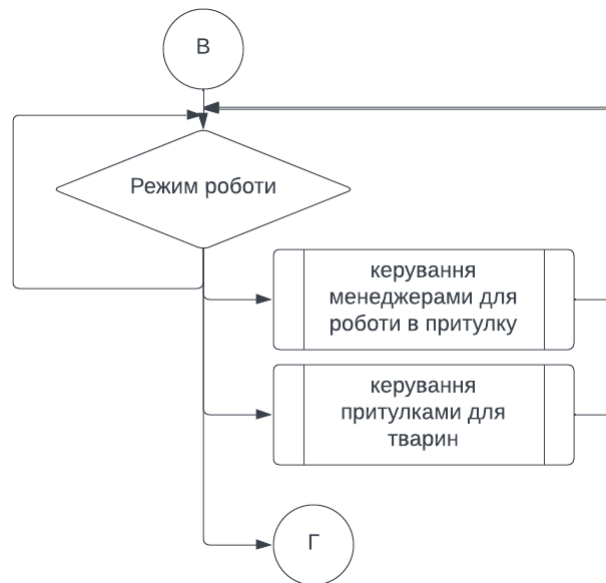


Рисунок 4.4 – Фрагмент функціональної схеми програми для адміністратора

4.2 Рішення з реалізації вебзастосунку

Вебзастосунок створено як проєкт вебфреймворку Django, виконуючи відповідно реалізацію всіх необхідних складових. Усі представлення були реалізовані на основі створення функцій під кожен функціональний вимогу.

Пошук серед повідомлень про втрату тварини реалізовано функцією `search_pet_lost_ad`. Функція не має окремих параметрів, але значення передаються через запит і виділяються потім, включаючи ім'я тварини (`petname`), породу (`petbreed`), категорію тварини (`petcategory`), стать тварини (`petsex`), номер мікрочіпу (`petchip`), зріст тварини (зріст має відповідати значенню `petheight` в інтервалі 5 сантиметрів від `petheight`), вага тварини (зріст має відповідати значенню `petweight` в інтервалі 10 грамів від `petweight`), колір шерсті (`woolcolor`), вік тварини та можливий інтервал віку (вік має відповідати значенню `petage` в інтервалі `petinterval` років від `petage` або в інтервалі 5 років, якщо значення інтервалу не задано), місто, де загублено тварину (`petcity`), країна, в якій загублено тварину (`petcountry`), інтервал дат,

коли було загублено тварину, що визначається можливою лівою та правою межами. Ліва межа дат визначається датою, заданою днем (daypetdate1), місяцем (monthpetdate1), роком (yearpetdate1). Права межа дат визначається датою, заданою днем (daypetdate2), місяцем (monthpetdate2), роком (yearpetdate2).

Пошук серед реєстрацій потрапляння тварин в притулок реалізовано функцією search_shelter_register. Додатково до попереднього пошуку в даному випадку також може додаватися притулок (petshelter). Якщо це і попереднє представлення не мають переданих значень параметрів, то повертається перелік значень за параметрами, які обираються з таблиці (міста, країни, породи, категорії).

Демонстрування картки тварини створено функцією pet_card, сама тварина визначається номером pet_id, що є параметром функції. Повертається не тільки вся інформація з БД про цю тварину, але і у випадку, якщо результат повертається користувачу, який є менеджером або господарем тварини, то і останні 2 записи про лікування цієї тварини, а також останні 2 записи про відповідність цієї картки іншим карткам або повідомленням.

Демонстрування даних притулку для тварин створено функцією shelter_info, сам притулок визначається номером shelter_id, що є параметром функції. Окрім даних притулку для тварин повертаються також останні 5 зареєстрованих у цьому притулку тварин.

Відображення зареєстрованих у програмі притулків для тварин відповідає функції shelters_list.

Демонстрування останніх зареєстрованих повідомлень про втрату тварин у місті створено через last_ads, номер міста задається параметром запиту petcity, а повідомлення впорядковуються за часом створення.

Додавання картки тварини створено функцією pet_card_create через забезпечення для цього форми PetCardForm, доступ до якої може отримати тільки авторизований господар тварини або менеджер. Редагування існуючої

картки тварини з номером `pet_card_id` виконано функцією `pet_card_change` з використанням тієї ж форми.

Демонстрування картки лікування тварини створено функцією `pet_therapies`, при цьому доступ до переліку всіх виконаних процедур з лікування надається тільки, якщо це менеджер одного з притулків, або якщо це господар цієї тварини з номером `pet_id`.

Демонстрування всіх відповідностей за картою тварини з номером `pet_id` реалізовано функцією `pet_possibilities`, при цьому доступ до переліку всіх виконаних процедур з лікування надається тільки, якщо це менеджер одного з притулків, або якщо це господар цієї тварини з номером `pet_id`.

Демонстрування створених користувачем відповідностей створено функцією `user_possibilities` за даними акаунту, визначаючи відповідності в залежності від того, чи цей користувач є господарем тварини, чи він є менеджером притулку для тварин.

Демонстрування карток власних тварин господаря реалізовано функцією `user_pet_cards`. Демонстрування власних повідомлень про втрату тварин реалізовано функцією `user_pet_ads`. Демонстрування всіх зареєстрованих дій щодо тварини з номером `pet_id` реалізовано функцією `pet_actions`, визначаючи всі реєстрації тварин у притулку, вибуття, переведення у інші притулки, повернення тварини до притулку.

Визначення відповідності між твариною з повідомлення та твариною з притулку реалізовано функцією `shelter_registration_possible` з визначенням реєстрації прибуття тварини в притулок за номером `sheltreg_id` та повідомлення за номером `ad_id`, номер якого може бути не заданий (тоді повертається перелік всіх повідомлень авторизованого користувача). Вилучення відповідності між твариною з повідомлення та твариною з притулку за номером `poss_id` реалізовано функцією `shelter_registration_possible_delete`.

Створення повідомлення про втрату тварини реалізовано функцією `pet_ad_create` через забезпечення для цього форми `PetlostadForm`, доступ до

якої може отримати тільки авторизований господар тварини, щодо якої створюється оголошення. Редагування існуючого повідомлення з номером `petad_id` про втрату тварини виконано функцією `pet_ad_change` з використанням тієї ж форми. За вилучення повідомлення відповідає функція `pet_ad_delete`.

Демонстрування повідомлення про втрату тварини з номером `petad_id` реалізовано функцією `pet_ad`. Демонстрування реєстрації з номером `register_id` потрапляння тварини в притулок реалізовано функцією `shelter_register_info`.

Реєстрація потрапляння тварини в притулок виконана функцією `shelter_register_create` з використанням форми `ShelterRegisterForm` з доступом до неї тільки менеджера притулку для тварин. При збереженні з використанням моделі повертається прогнозована кількість днів, протягом якої тварина буде перебувати в притулку. Разом з загальною інформацією про реєстрацію прогнозоване значення в кінці відображається на сторінці. Редагування даних реєстрації потрапляння тварини в притулок реалізовано функцією `shelter_register_edit` з використанням цієї ж форми за номером реєстрації `register_id`. Сама ж модель прогнозування створена на основі класу `XGBRegressor` з пакету `xgboost`.

Пошук серед реєстрацій потрапляння тварин в притулок реалізовано функцією `shelter_register_search` з визначенням притулку за тим, в якому притулку працює менеджер. Якщо це не менеджер притулку для тварин, то йому результати не надаються. Пошук реалізується з визначенням у розширеному вигляді параметрів, які вище були визначені для пошуку серед реєстрацій потрапляння тварин в притулок, а також з можливістю задати номер реєстрації (`register_id`), стан (`status_number`), причину потрапляння (`reason_id`), адресу, за якою знайдено тварину (`address`), тип дій з потрапляння у притулок (`take_type_id`) та позначка про те, що реєстрацію мав виконувати сам авторизований менеджер. Для пошуку серед реєстрацій повернення тварин у притулок створено функцію `movement_register_return_search` з

можливістю задати номер реєстрації (`register_id`), причину повернення (`reason_id`) та позначкою про те, що реєстрацію мав виконувати сам авторизований менеджер. Пошук серед реєстрацій переведень тварин між притулками реалізовано функцією `transfer_search` тільки з додатковою позначкою. Пошук серед реєстрацій вибуття тварин з притулку реалізовано функцією `movement_register_search` з типом вибуття (`movement_type_id`), станом тварини (`condition_id`), позначкою про авторство. Пошук серед реєстрацій за притулком реалізовано функцією `shelter_to_register_search` з додатковим визначенням притулку (`shelter_id`), до якого було зареєстровано тварину. Пошук серед фіксацій лікувань тварин реалізовано функцією `pet_therapy_search` з визначенням додаткового параметра у вигляді позначки про авторство.

Реалізований клас `Petcategory` формує модель категорії тварин, що фактично є довідниковими даними, які використовуються в програмі для визначення можливих категорій (типів) тварин. Цей клас включає атрибути стандартного для Django ідентифікатора в даному випадку категорії тварин, що має назву `id` та тип `AutoField`, та атрибут з назвою категорії тварин, що має назву `category` і тип `CharField` (це і є пес, кіт тощо).

Демонстрування часу перебування тварин у притулку з номером `shelter_id` за категорією з номером `category_id` або за всіма категоріями, якщо номер не задано, реалізовано функцією `pet_in_shelter_time` за час з `start` до `end`. Демонстрування кількості тварин у притулку з номером `shelter_id` реалізовано функцією `pet_in_shelter_number` за час з `start` до `end`. Демонстрування реєстрацій за притулком з номером `shelter_id` реалізовано `shelter_pet_registers` за час з `start` до `end`. Демонстрування лікувань тварин за менеджером, який авторизувався, реалізовано функцією `manager_pet_therapies`. Демонстрування останніх зареєстрованих повернутих тварин за притулком з номером `shelter_id` реалізовано функцією `last_shelter_returns`. Демонстрування останніх зареєстрованих повернутих тварин за менеджером реалізовано функцією `last_shelter_returns`.

Демонстрування останніх зареєстрованих тварин за вибуттям за притулком з номером `shelter_id` реалізовано функцією `last_shelter_movements`. Демонстрування останніх зареєстрованих тварин за вибуттям за менеджером реалізовано функцією `last_manager_movements`. Демонстрування останніх зареєстрованих тварин за переведенням з притулку з номером `shelter_id` реалізовано функцією `last_shelter_transfers`. Демонстрування останніх зареєстрованих тварин за переведенням до притулку з номером `shelter_id` реалізовано функцією `last_to_shelter_transfers`. Демонстрування останніх зареєстрованих тварин за переведеннями між притулками за менеджером реалізовано функцією `last_between_shelter_transfers`.

Вебсторінки шаблонів для описаних функцій представлень створені на основі макетів, які забезпечені за допомогою стилів шаблону [11].

Реалізований клас `Animalbreed` формує модель породи тварини, до якої потім має бути віднесено конкретну тварину, зареєстровану в системі, та модель якої містить поля:

- `id` – ідентифікатор відомої породи тварин, тип `AutoField` (первинний ключ моделі);
- `petcategory` – категорія тварин, до якої відноситься ця порода, адже породи не відносяться до всіх тварин одночасно, тип `ForeignKey`;
- `name` – назва цієї відомої породи тварин, тип `CharField`.

Реалізований клас `Registerreasontype` формує модель стандартної причини потрапляння тварини до притулку та знову ж як і попередня модель є довідковими даними. Одна з причин встановлюється за кожної реєстрації потрапляння тварини в притулок. Цей клас включає атрибути стандартного для Django ідентифікатора в даному випадку причини потрапляння тварини до притулку, що має назву `id` та тип `AutoField`, та атрибут з назвою причини потрапляння тварини до притулку, що має назву `reason_type` і тип `CharField`.

Реалізований клас `Taketype` формує модель типу дій, за допомогою яких тварина потрапила у притулок як реакція на виявлену причину потрапляння тварини до притулку. Цей клас включає атрибути стандартного

для Django ідентифікатора в даному випадку типу дій, за допомогою яких тварина може потрапити у притулок, що має назву `id` та тип `AutoField`, та атрибут з назвою типу дій, за допомогою яких тварина може потрапити у притулок, що має назву `take_name` і тип `CharField`.

Реалізований клас `Movementtype` формує модель типу дій, за допомогою яких тварина покинула притулок після перебування в ньому. Цей клас включає атрибути стандартного для Django ідентифікатора в даному випадку типу дій, за допомогою яких тварина покинула притулок, що має назву `id` та тип `AutoField`, та атрибут з назвою типу дій, за допомогою яких тварина покинула притулок, що має назву `movement` і тип `CharField`.

Реалізований клас `Therapytype` формує модель можливого типу лікування, який застосовується до тварин. Цей клас включає атрибути стандартного для Django ідентифікатора в даному випадку типу лікування, який застосовується до тварин, що має назву `id` та тип `AutoField`, та атрибут з назвою типу лікування, який застосовується до тварин, що має назву `therapy_name` і тип `CharField`.

Реалізований клас `Returnreason` формує модель можливої причини повернення тварини, яка була в притулку, до нього та містить поля:

- `id` – можливої причини повернення тварини, яка була в притулку, тип `AutoField` (первинний ключ моделі);
- `reason_name` – назва можливої причини повернення тварини, яка була в притулку, має тип `CharField`.

Реалізований клас `Locationcity` формує модель міста, в якому можуть відбуватися події (співвіднесення розташування притулку, знаходження тварини) та яка складається з полів:

- `id` – ідентифікатор міста, з яким може відбуватися співвіднесення, має тип `AutoField` (первинний ключ моделі);
- `city_country` – країна, в якій таке місто знаходиться у відповідності з міжнародно визнаними територіями, тип `ForeignKey`, тобто є зовнішнім ключем;

– city – назва міста, з яким може відбуватися співвіднесення, має тип CharField.

Реалізований клас Locationcountry формує модель країни, з містом якої може відбуватися співвіднесення розташування притулку, знаходження тварини та яка складається з полів:

– id – ідентифікатор країни, з яким може відбуватися співвіднесення, має тип AutoField (первинний ключ моделі);

– country – назва країни з містом якого може відбуватися співвіднесення, має тип ForeignKey, тобто є зовнішнім ключем.

Реалізований клас Petcondition формує можливий стан тварин, складається з полів:

– id – ідентифікатор можливого стану тварини, має тип AutoField (первинний ключ моделі);

– condition_name – назва можливого стану тварини, має тип CharField.

Реалізований клас Pet формує модель картки конкретної тварини, яка може або потрапити у притулок, або бути загублена, складається з полів:

– id – ідентифікатор тварини, яку внесено в програму, має тип AutoField (первинний ключ моделі);

– name – ім'я тварини (кличка), має тип CharField;

– sex – стать тварини;

– owner – господар тварини, має тип ForeignKey, тобто є зовнішнім ключем;

– chipnumber – номер, який зафіксовано в мікрочіпі тварини, якщо мікрочіп наявний, має тип CharField;

– breed – порода тварини, має тип ForeignKey, тобто є зовнішнім ключем;

– photo – фотографія тварини, має тип ImageField;

– description – інформація про тварину як її опис, має тип CharField;

– height – висота тварини, має тип SmallIntegerField;

– weight – вага тварини, має тип SmallIntegerField;

- `woolcolor` – головний колір шерсті тварини, має тип `CharField`;
- `age` – вік тварини, має тип `SmallIntegerField`;
- `status` – відомий на поточний момент стан тварини відносно її перебування в притулку або у господаря, тип `SmallIntegerField`;
- `location_city` – місто, в якому на даний момент знаходиться тварина, має тип `ForeignKey`, тобто є зовнішнім ключем;
- `location_address` – повна адреса, за якою знаходиться тварина (якщо вона знаходиться у господаря, інакше це адреса притулку, а вона співвідноситься з притулком, а для тварини реєструється прибуття в притулок), має тип `CharField`;
- `source_type` – джерело інформації (картку зареєстровано при потраплянні тварини в притулок або самим господарем), має тип `BooleanField`;
- `last_manager` – менеджер, який останнім вносив зміни до картки тварини, має тип `ForeignKey`, тобто є зовнішнім ключем.

Реалізований клас `Petpossibility` формує модель визначення відповідності між різними картками про тварин, які виявились або можуть виявитися однією і тією самою твариною, складається з полів:

- `id` – ідентифікатор тварини, яку внесено в програму, має тип `AutoField` (первинний ключ моделі);
- `pet_source` – перша картка про тварину з відповідності, має тип `ForeignKey`, тобто є зовнішнім ключем;
- `pet_possible` – друга картка про тварину з відповідності, має тип `ForeignKey`, тобто є зовнішнім ключем;
- `approve` – підтвердження факту відповідності, тобто що дві картки стосуються однієї і тієї самої тварини, має тип `BooleanField`;
- `manager` – менеджер, який створив цю відповідність між різними картками про тварин, які виявились або можуть виявитися однією і тією самою твариною, має тип `ForeignKey`, тобто є зовнішнім ключем.

Реалізований клас Petphoto формує модель фотографії, де відображена тварина, складається з полів:

- id – ідентифікатор фотографії тварини, яку внесено в програму, має тип AutoField (первинний ключ моделі);
- pet – тварина, яку відображено на цій фотографії, має тип ForeignKey, тобто є зовнішнім ключем;
- photo – вміст фотографії тварини, має тип ImageField;
- made_date – час, коли цю фотографію тварини було створено, має тип DateTimeField.

Реалізований клас Shelter формує модель притулку, складається з полів:

- id – ідентифікатор притулку для тварин, має тип AutoField (первинний ключ моделі);
- title – назва притулку, має тип CharField;
- description – інформація з опису про притулок, має тип CharField;
- photo – вміст фотографії притулку, має тип ImageField;
- location_city – місто, в якому знаходиться притулок для тварин, має тип ForeignKey, тобто є зовнішнім ключем;
- location_address – повна адреса, за якою знаходиться притулок для тварин, має тип CharField;
- open_date – дата, коли було цей притулок для тварин відкрито, має тип DateField;
- status – стан притулку (діючий або закритий), має тип BooleanField.

Реалізований клас Shelterpetcategory формує модель співвідношення між притулком для тварин і категорією тварин, що визначає роботу з такими тваринами у притулку, складається з полів:

- id – ідентифікатор встановленого співвідношення між притулком для тварин і категорією тварин, має тип AutoField (первинний ключ моделі);
- shelter – притулок, в якому працюють з цією категорією тварин, має тип ForeignKey, тобто є зовнішнім ключем;

- petcategory – категорія тварин, з якою ведеться робота в притулку, має тип ForeignKey, тобто є зовнішнім ключем;

- max_places – кількість місць у притулку для цієї категорії тварин, має тип SmallIntegerField.

Реалізований клас Shelterregister формує реєстрацію потрапляння тварини в притулок, складається з полів:

- id – ідентифікатор реєстрації потрапляння тварини в притулок, має тип AutoField (первинний ключ моделі);

- pet – тварина, яку зареєстровано в притулку, має тип ForeignKey, тобто є зовнішнім ключем;

- shelter – притулок, в якому зареєстровано тварину, має тип ForeignKey, тобто є зовнішнім ключем;

- manager – менеджер, який зареєстрував цю тварину в притулку, має тип ForeignKey, тобто є зовнішнім ключем;

- take_type – тип дій, за допомогою яких тварина потрапила у притулок як реакція на виявлену причину потрапляння тварини до притулку, має тип ForeignKey, тобто є зовнішнім ключем;

- register_date – час реєстрації потрапляння тварини в притулок, має тип DateTimeField;

- reason – причина потрапляння тварини до притулку, має тип ForeignKey, тобто є зовнішнім ключем;

- pet_age – вік тварини, який було визначено під час реєстрації потрапляння тварини в притулок, має тип SmallIntegerField;

- found_city – місто, в якому було знайдено тварину перед її привезенням у притулок, має тип ForeignKey, тобто є зовнішнім ключем;

- found_address – повна адреса, за якою було знайдено тварину перед її привезенням у притулок, має тип CharField;

- pet_condition – стан, який був співставлений з твариною під час реєстрації її потрапляння в притулок, має тип ForeignKey, тобто є зовнішнім ключем;

- open – відкрита реєстрація або закрита, тобто чи вона доступна всім користувачам для перегляду, має тип BooleanField;
- status – стан реєстрації, тобто чи вона ще не закрита, тобто тварина в притулку, чи ні, має тип BooleanField;
- prediction – прогнозована кількість днів, протягом якої тварина буде перебувати в притулку, має тип SmallIntegerField.

Реалізований клас Transfer формує модель переведення тварини в інший притулок, складається з полів:

- id – ідентифікатор переведення тварини в інший притулок, має тип AutoField (первинний ключ моделі);
- shelterregister – реєстрація потрапляння тварини в притулок, має тип ForeignKey, тобто є зовнішнім ключем;
- shelter – притулок, до якого було переведено тварину, має тип ForeignKey, тобто є зовнішнім ключем.

Реалізований клас Movementregister формує модель реєстрації вибуття тварини з притулку (не переведення до іншого притулку), складається з полів:

- id – ідентифікатор реєстрації вибуття тварини з притулку, має тип AutoField (первинний ключ моделі);
- shelterregister – реєстрація вибуття тварини в притулок, має тип ForeignKey, тобто є зовнішнім ключем;
- register_date – час реєстрації вибуття тварини в притулок, має тип DateTimeField;
- manager – менеджер, який зареєстрував вибуття цієї тварини з притулку, має тип ForeignKey, тобто є зовнішнім ключем;
- movement_type – тип дій, за допомогою яких тварина покинула притулок після перебування в ньому, має тип ForeignKey, тобто є зовнішнім ключем;

- `pet_condition` – стан, який був співставлений з твариною під час реєстрації її вибуття з притулку, має тип `ForeignKey`, тобто є зовнішнім ключем;

- `manager` – менеджер, який зареєстрував вибуття тварини з притулку, має тип `ForeignKey`, тобто є зовнішнім ключем.

Реалізований клас `Movementregisterreturn` формує модель повернення тварини людиною, яка взяла її до себе до того ж притулку (тобто людина змінила своє рішення), складається з полів:

- `id` – ідентифікатор повернення тварини людиною, яка взяла її до себе до того ж притулку, має тип `AutoField` (первинний ключ моделі);

- `movementregister` – реєстрації вибуття тварини з притулку, коли ця людина взяла цю тварину до себе, має тип `ForeignKey`, тобто є зовнішнім ключем;

- `reason` – причина повернення тварини до притулку, має тип `ForeignKey`, тобто є зовнішнім ключем;

- `return_date` – час повернення тварини до притулку, має тип `DateTimeField`.

Реалізований клас `Manager` формує модель акаунту менеджера притулку для тварин, складається з полів:

- `id` – ідентифікатор акаунту менеджера притулку для тварин, має тип `AutoField` (первинний ключ моделі);

- `account` – акаунт користувача в Django, має тип `ForeignKey`, тобто є зовнішнім ключем;

- `shelter` – притулок, для тварин, в якому працює цей менеджер, має тип `ForeignKey`, тобто є зовнішнім ключем;

- `role` – менеджерська роль за цим акаунтом, має тип `SmallIntegerField`;

- `info` – інформація з опису, виконаного менеджером, має тип `CharField`;

- `phone` – номер телефону, за яким можна зв'язатися з цим менеджером притулку для тварин, має тип `CharField`.

Реалізований клас `Client` формує модель акаунту користувача програми підтримки діяльності притулку для тварин, складається з полів:

- `id` – ідентифікатор акаунту користувача програми підтримки діяльності притулку для тварин, має тип `AutoField` (первинний ключ моделі);
- `account` – акаунт користувача в Django, має тип `ForeignKey`, тобто є зовнішнім ключем;
- `photo` – вміст фотографії притулку, має тип `ImageField`;
- `location_city` – місто, в якому знаходиться цей користувач програми підтримки діяльності притулку для тварин, має тип `ForeignKey`, тобто є зовнішнім ключем;
- `location_address` – повна адреса, за якою знаходиться цей користувач програми підтримки діяльності притулку для тварин, має тип `CharField`;
- `phone` – номер телефону цього користувача програми підтримки діяльності притулку для тварин, має тип `CharField`.

Реалізований клас `Petlostad` формує модель створеного користувачем повідомлення про втрату тварини, складається з полів:

- `id` – ідентифікатор створеного користувачем повідомлення про втрату тварини, має тип `AutoField` (первинний ключ моделі);
- `pet` – тварина, яку було загублено та якої стосується повідомлення, має тип `ForeignKey`, тобто є зовнішнім ключем;
- `lost_date` – орієнтований час, коли було загублено цю тварину, має тип `DateTimeField`;
- `description` – повідомлення щодо загублення тварини, яке описує деталі, має тип `CharField`;
- `location_city` – місто, в якому було загублено цю тварину, має тип `ForeignKey`, тобто є зовнішнім ключем;
- `location_address` – повна адреса, за якою було загублено цю тварину, має тип `CharField`;
- `status` – стан повідомлення щодо загублення тварини, має тип `BooleanField`.

Реалізований клас `Petlostadpossibility` формує модель визначення відповідності між створеним користувачем повідомленням про втрату тварини та реєстрацією потрапляння тварини в притулок, складається з полів:

- `id` – ідентифікатор визначення відповідності між створеним користувачем повідомленням про втрату тварини та реєстрацією потрапляння тварини в притулок, має тип `AutoField` (первинний ключ моделі);

- `petlostad` – створене користувачем повідомлення про втрату тварини, має тип `ForeignKey`, тобто є зовнішнім ключем;

- `manager` – менеджер, який вважає, що між створеним користувачем повідомленням про втрату тварини та реєстрацією потрапляння тварини в притулок є відповідність, має тип `ForeignKey`, тобто є зовнішнім ключем;

- `shelterregister` – реєстрація потрапляння тварини в притулок, має тип `ForeignKey`, тобто є зовнішнім ключем;

- `message` – повідомлення, яке визначено стосовно цієї відповідності, має тип `CharField`;

- `add_date` – час визначення відповідності між створеним користувачем повідомленням про втрату тварини та реєстрацією потрапляння тварини в притулок, має тип `DateTimeField`;

- `status` – стан оброблення відповідності між створеним користувачем повідомленням про втрату тварини та реєстрацією потрапляння тварини в притулок, має тип `SmallIntegerField`.

Реалізований клас `Shelterregisterpossibility` формує модель визначення відповідності між власною твариною користувача та твариною, потрапляння якої було зареєстровано в притулку, складається з полів:

- `id` – ідентифікатор визначення відповідності між власною твариною користувача та твариною, потрапляння якої було зареєстровано в притулку, має тип `AutoField` (первинний ключ моделі);

- `shelterregister` – реєстрація потрапляння тварини в притулок, має тип `ForeignKey`, тобто є зовнішнім ключем;

- client – користувач програми підтримки діяльності притулку для тварин
- message – повідомлення, яке визначено стосовно цієї відповідності, має тип CharField;
- add_date – час визначення відповідності між власною твариною користувача та твариною, потрапляння якої було зареєстровано в притулку, має тип DateTimeField;
- status – стан оброблення відповідності між власною твариною користувача та твариною, потрапляння якої було зареєстровано в притулку, має тип SmallIntegerField.

Реалізований клас Pettherapy формує модель лікування, яке застосовується до тварини, складається з полів:

- id – ідентифікатор лікування, яке застосовується до тварини, має тип AutoField (первинний ключ моделі);
- pet – тварина, яку було проліковано, має тип ForeignKey, тобто є зовнішнім ключем;
- therapy_type – ти лікування, яке застосовується до цієї тварини, має тип ForeignKey, тобто є зовнішнім ключем;
- therapy_date – час лікування, має тип DateTimeField;
- info – інформація про лікування, має тип CharField;
- manager – менеджер, який зареєстрував це лікування, має тип ForeignKey, тобто є зовнішнім ключем.

4.3 Висновки за розділом 4

Створено функціональну схему програми та за нею програмно реалізовані всі варіанти взаємодії менеджера притулку для тварин, господарів тварин, неавторизованих користувачів, адміністраторів з описом функцій представлень, класів моделей.

5 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

5.1 Опис програми

Програма підтримує діяльність притулку для тварин, забезпечуючи реєстрацію дій, які виконуються в притулку для тварин, внесення даних про самі тварини в притулку і їхніми господарями, створення повідомлень господарями тварин про їх втрату, пошук серед повідомлень і реєстрацій, визначення відповідностей між повідомленнями господарів та реєстраціями тварин у притулках для тварин, планування діяльності притулків для тварин.

5.2 Експлуатація програми

У процесі експлуатації програми перевірено виконання тих сценаріїв, які мали бути забезпечені у відповідності з наведеними в розділі 3 діаграмами. Усі функції реалізовано. Нижче описано загальні правила роботи, хоча загалом вони відповідають тому порядку, який запланований і описаний у підрозділі 4.1.

Неавторизований користувач може управляти роботою через головне меню, показане згори, яке включає зокрема пункт Увійти, щоб авторизуватися або зареєструватися, якщо користувач не має акаунту (рис. 5.1).

Робота розпочинається з виведення сторінки пошуку тварин за реєстрацією в притулках, що за пунктом меню Пошук відображається в додатковому меню (зліва на сторінці) як пункт Притулки. Після введення у формі пошуку параметрів тварин відображаються результати пошуку. Ці результати містять базові дані картки тварини, включаючи фотографію тварини, її породу, стать, притулок, в якому знаходиться тварина та її орієнтовний вік. Якщо натиснути на породу тварини, то відкриється сторінка з картою тварини з повною інформацією.

Результати пошуку тварин серед реєстрацій у притулках

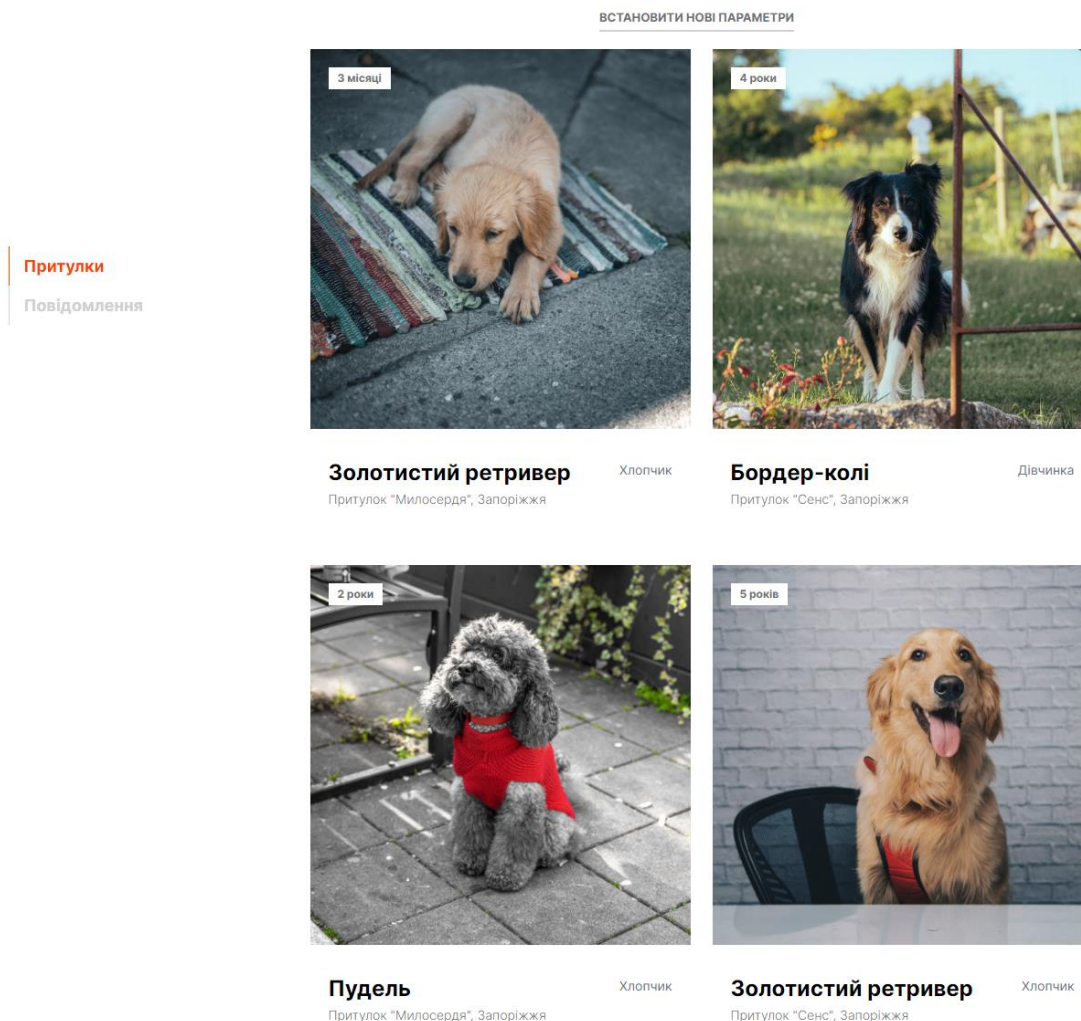


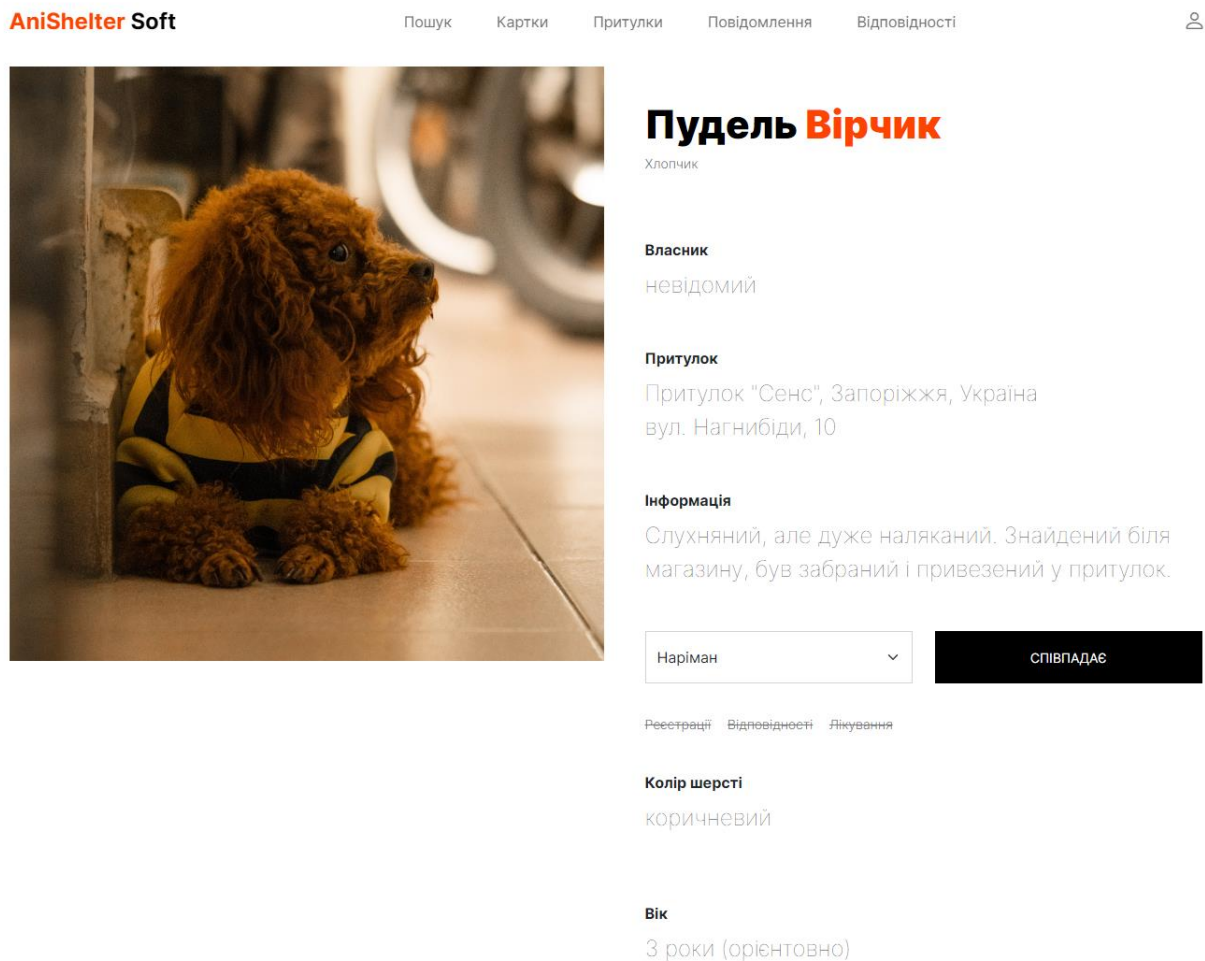
Рисунок 5.1 – Пошук тварин серед реєстрацій у притулках

Пункт додаткового меню Повідомлення дозволяє шукати серед повідомлень про втрату тварин, створених самостійно звичайними користувачами (господарями тварин) на вебсайті.

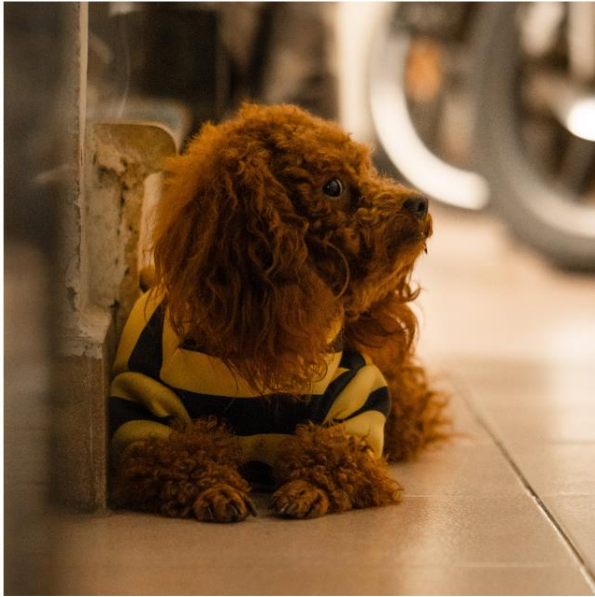
Пункт головного меню Притулки надає можливість подивитися наявні притулки для тварин у програмі. Пункт Повідомлення дозволяє відобразити останні створені на вебсайті повідомлення про втрату тварин, задавши у рядку пошуку конкретне місто..

Успішна авторизація змінює головне меню користувача. Якщо користувач є господарем тварини, то у нього додається пункт Картки, який

дозволяє працювати з даними власних тварин, пункт Повідомлення у цьому випадку має в додатковому меню ще пункт Власні, що дозволяє додавати, переглядати та редагувати власні оголошення господаря. Також господар має пункт меню Відповідності, що надає перелік створених відповідностей між карткою тварини господаря та зареєстрованими тваринами у притулках (рис. 5.2).



AniShelter Soft Пошук Картки Притулки Повідомлення Відповідності 



Пудель Вірчик

Хлопчик

Власник
невідомий

Притулок
Притулок "Сенс", Запоріжжя, Україна
вул. Нагнибіди, 10

Інформація
Слухняний, але дуже наляканий. Знайдений біля магазину, був забраний і привезений у притулок.

Реєстрації Відповідності Лікування

Колір шерсті
коричневий

Вік
3 роки (орієнтовно)

Рисунок 5.2 – Картка тварини у програмі

Для того щоб таку відповідність встановити, господар має знайти картку потрібної тварини, для чого спочатку потрібно перейти на сторінку реєстрації тварини у притулку, а далі до картки тварини, а тоді з випадającego переліку обрати свою тварину, з якою потрібно встановити відповідність. Картка містить опис тварини, внизу зокрема фотографії (окрім

основної), а також пункти для отримання додаткових даних, зокрема Реєстрації, Відповідності, Лікування, але ці пункти доступні тільки господарю цієї тварини, для всіх інших вони перекреслені і недоступні. Ці ж пункти доступні також і для менеджера. Їм же відображаються останні знайдені відповідності.

Менеджер притулку для тварин отримує в головному меню пункт Реєстрації, через який йому доступна можливість доступу до додаткового меню з пунктами Всі дії, Потрапляння, Повернення, Переведення, Вибуття, Лікування для роботи з відповідними реєстраціями (рис. 5.3).

AniShelter Soft
Пошук
Реєстрації
Притулки
Повідомлення
Відповідності

Реєстрації вибуття тварин з притулку Сонечко

Всі дії
Потрапляння
Повернення
Переведення
Вибуття
Лікування

Вибуття з
Вибуття до
Результат

дд.мм.rrrr

дд.мм.rrrr

всі

Стан

всі

ШУКАТИ

РЕЄСТРУВАТИ ПОТРАПЛЯННЯ

03.12.2023

Бордер-колі

Усиновлення

Редагувати

Здоровий

03.12.2023

Лабрадор

Усиновлення

Редагувати

Здоровий

Рисунок 5.3 – Демонстрування останніх реєстрацій вибуття тварин з притулку

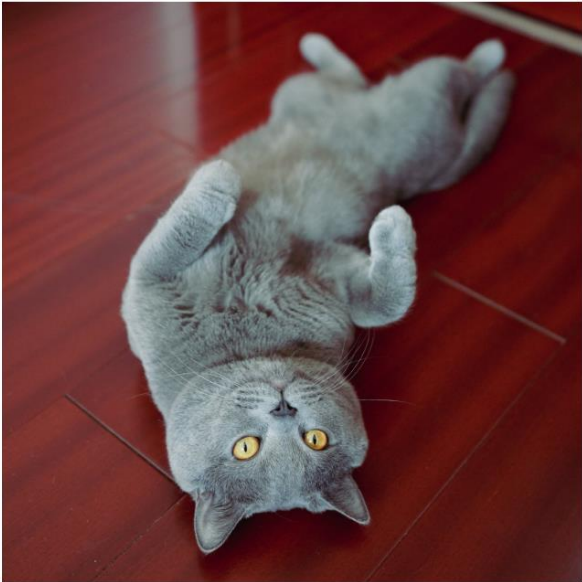
Такі сторінки реєстрації відображають останні реєстрації за менеджером або за притулком, з яким він працює (посилання Притулок тоді змінює назву на Менеджер і назад). Але також вони містять зазвичай форму пошуку, в якій одразу можна задати потрібні параметри та отримати результат пошуку, який реалізується за допомогою окремого представлення.

Після внесення в форму даних реєстрації потрапляння тварини в притулок менеджером, збереження цих даних відображається сторінка з внесеними даними, яка дозволяє одразу перейти до редагування, якщо щось було внесено некоректно, а також показує прогнозоване значення, коли тварина залишить притулок (рис. 5.4).

Адміністратор має об'єднані можливості господаря тварини та менеджера притулку, а також містить у цих розділах додаткові можливості реєстрації нових менеджерів та внесенні даних нових притулків для тварин.

AniShelter Soft
Пошук
Реєстрації
Притулки
Повідомлення
Відповідності

Реєстрація потрапляння тварини у притулок



Кіт

Родріго

Британський короткошерстний

Хлопчик

Здоровий

КАРТКА

Реєстрація

Очікуваний вік: 3 роки

Втеча від господаря

Бродяча тварина

Зареєстровано 03.12.2023 12:45

Менеджер Аубакірова Марія

Притулок "Сенс", Запоріжжя

Адреса знаходження: Запоріжжя, проспект Соборний, 123

Відкрита для пошуку

Стан: створений

РЕДАГУВАТИ

Прогноз

Перебування в притулку до 05.12.2023

Рисунок 5.4 – Реєстрація потрапляння тварини у притулок

5.3 Експериментальне дослідження

Вибірка даних [12], яку було застосовано для експериментального дослідження, зібрана за допомогою даних притулків для тварин у Блумінгтоні, що в штаті Техас у США. Ця вибірка складається з 10290 записів. Це тварини, які потрапляли в притулки Блумінгтона протягом 2017-2020 років. Серед них найбільше собак та котів, а всі інші тварини представлені у незначній кількості (рис. 5.5).

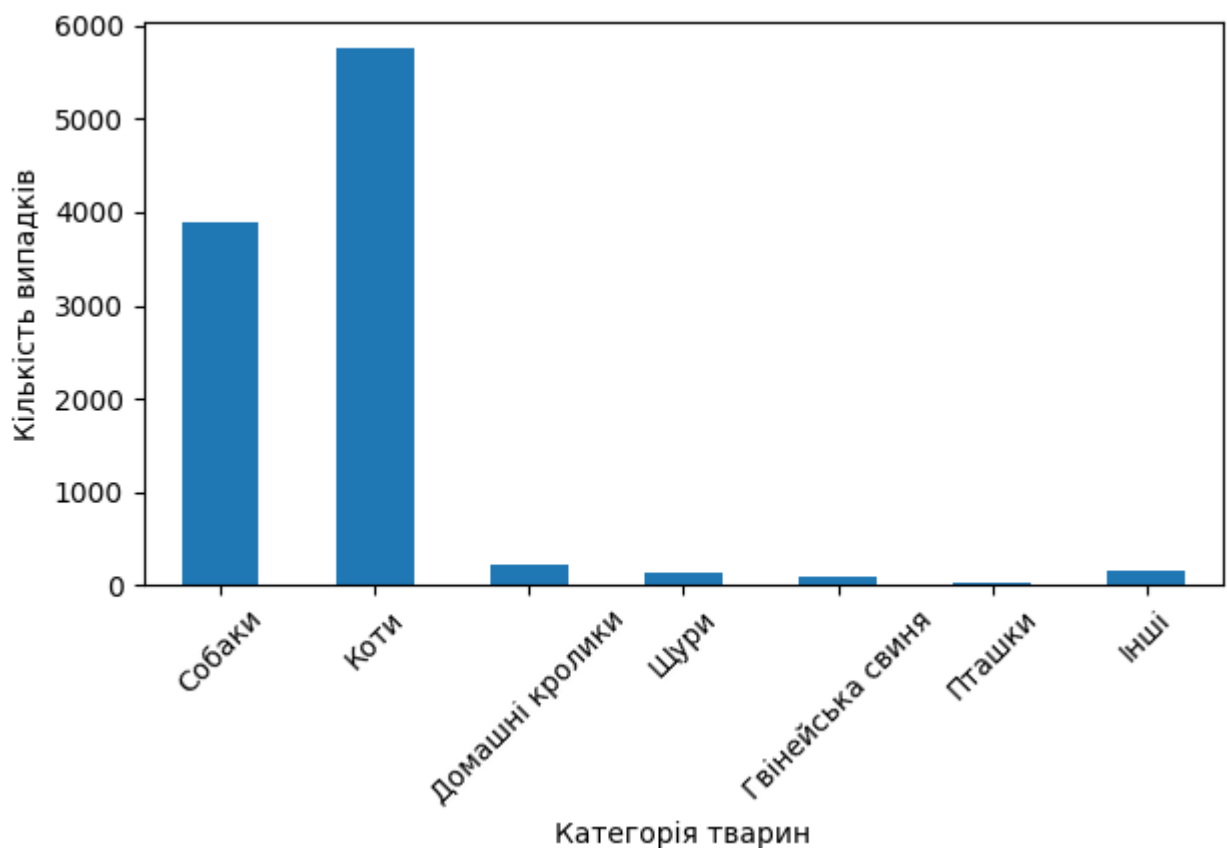


Рисунок 5.5 – Кількість випадків потрапляння різних категорій тварин у притулок

Результати вивчення періодів, протягом яких тварина знаходилась в притулку, показали, що основна частина випадків потрапляння тварин у притулок тривала протягом 3 місяців, більша тривалість знаходження в притулку є достатньо незначною (рис. 5.6). Також під час вивчення було

виявлено, що серед випадків у вибірці даних є такі, в яких дата вибуття з притулку вказана ранішою за дату потрапляння. Такі випадки були вилучені з вибірки даних. Також були вилучені випадки, в яких дані дубльовані, які також були знайдені у вибірці.

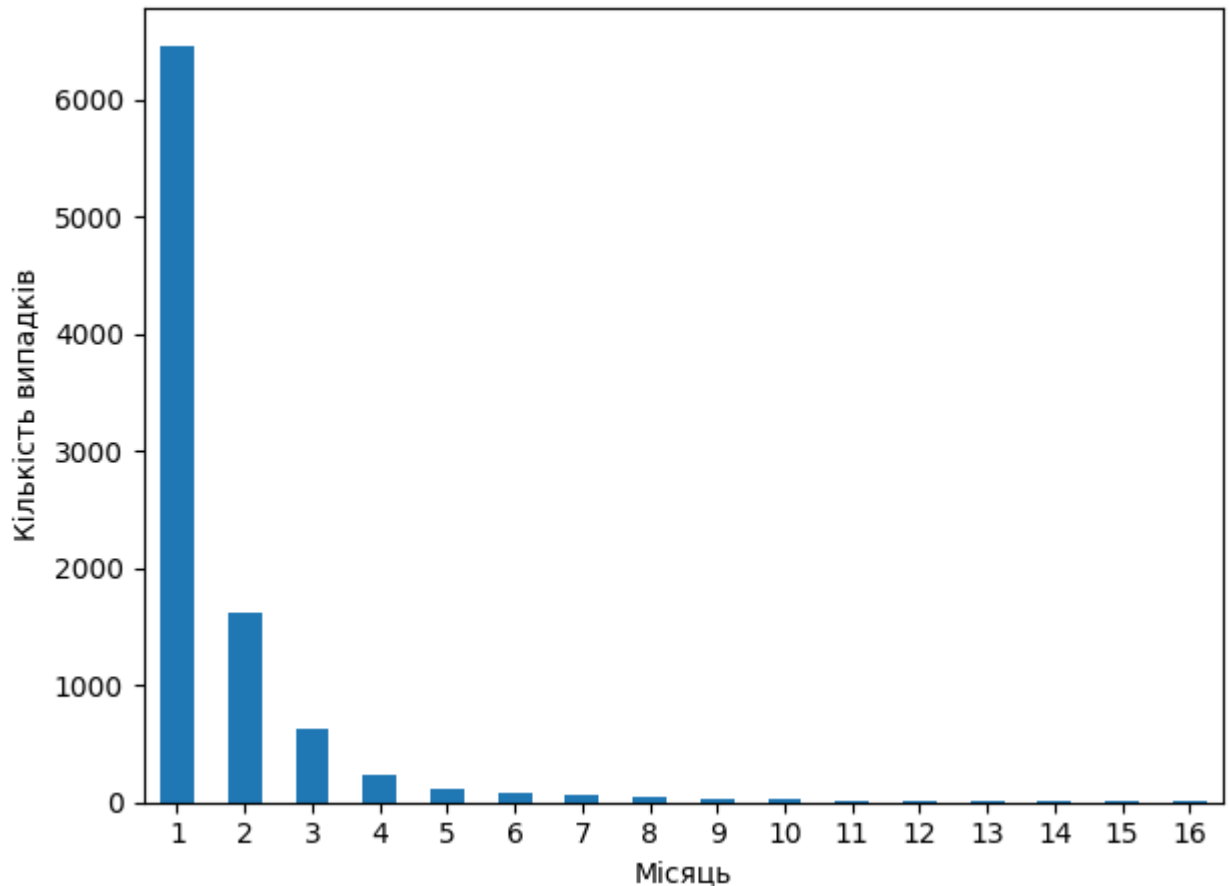


Рисунок 5.6 – Кількість випадків завершення перебування тварин у притулку за різну кількість місяців

Оскільки велика частка випадків перебування в притулку тривала перші 3 місяці, то окремо було розглянуто розподіл за тижнями (рис. 5.7). Більше третини випадків тривали менше тижня.

Після аналізу і оброблення даних було створено модель на основі XGBoost, для якої перевірено різні варіанти з однаковими іншими параметрами, але різними вхідними даними (табл. 5.1). Результати оцінено за середньою абсолютною похибкою. Запропоновані у методі, створеному в роботі, вхідні дані мають найменшу похибку в 0,01331958.

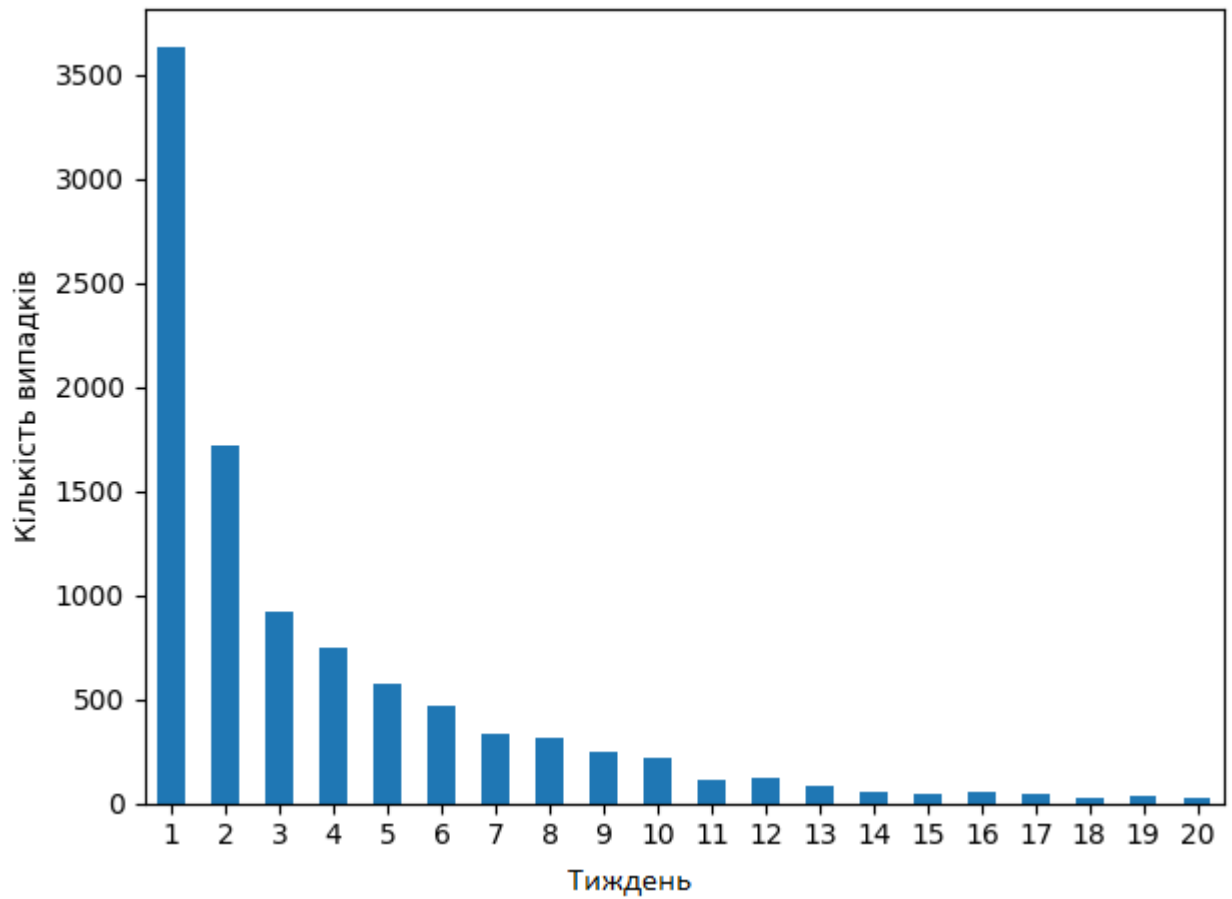


Рисунок 5.7 – Кількість випадків завершення перебування тварин у притулку за різну кількість тижнів (для перших 20 тижнів)

Таблиця 5.1 – Дослідження параметрів моделі XGBoost

Модель	Дата потрапляння	Причина	Порода	Колір	Категорія	Вік	Середня абсолютна похибка
XGBoost	+	—	+	—	—	—	0,01338571
XGBoost	+	—	+	—	+	+	0,01364843
XGBoost	+	—	+	+	+	—	0,01382266
XGBoost	+	—	+	+	+	+	0,01411962
XGBoost	+	+	+	+	+	+	0,01387665
XGBoost	+	+	+	—	+	—	0,01356873
XGBoost	+	—	+	—	+	—	0,01331958

Для перевірки даної моделі порівняно з іншими моделями було також виконано прогнозування періоду перебування тварин у притулку за допомогою моделі CART та різних інших параметрів моделі XGBoost (табл. 5.2). Найкращий результат з описаними вхідними даними, 40 раундами бустингу, максимальною глибиною дерева в 6 отримано з середньою абсолютною похибкою у 0,01331958 за методом, який запропоновано. Це на 11,7 % кращий результат ніж у моделі CART. Найкраща модель має серед вхідних даних значення дати потрапляння тварини у притулок, породи тварини та категорії тварини.

Таблиця 5.2 – Дослідження різних моделей для прогнозування періоду перебування тварини у притулку

Модель	Середня абсолютна похибка
CART	0,01508935
XGBoost (10 раундів бустингу, глибина 3)	0,01353547
XGBoost (40 раундів бустингу, глибина 8)	0,01345309
XGBoost (40 раундів бустингу, глибина 3)	0,01349237
XGBoost (1000 раундів бустингу, глибина 6)	0,01381536
XGBoost (100 раундів бустингу, глибина 6)	0,01348275
XGBoost (40 раундів бустингу, глибина 6)	0,01331958

5.4 Висновки за розділом 5

За функціональною схемою програми реалізовані всі варіанти взаємодії менеджера притулку для тварин, господарів тварин, неавторизованих користувачів, адміністраторів.

Експериментальне дослідження виконано за допомогою даних притулків для тварин у Блумінгтоні, у які тварини потрапляли протягом

2017-2020 років. Дані було оброблено, виключаючи некоректні. Створено і застосовано моделі CART і XGBoost, перевірено параметри моделі XGBoost. Створена за методом, розробленим у роботі, модель мала на 11,7 % кращий результат ніж у моделі CART.

ВИСНОВКИ

Усі завдання, а відповідно і мету роботи, було виконано і досягнуто, включаючи аналіз аналогів програми, розробку методу прогнозування періоду перебування тварини у притулку, проєктування і реалізацію власного програмного забезпечення підтримки діяльності притулку для тварин.

За функціональною схемою програми реалізовані всі варіанти взаємодії менеджера притулку для тварин, господарів тварин, неавторизованих користувачів, адміністраторів. Створена програма підтримує діяльність притулку для тварин, забезпечує реєстрацію дій, які виконуються в притулку для тварин, внесення даних про самих тварин у притулку і їхніми господарями, створення повідомлень господарями тварин про їх втрату, пошук серед повідомлень і реєстрацій, визначення відповідностей між повідомленнями господарів та реєстраціями тварин у притулках для тварин, планування діяльності притулків для тварин на основі визначення очікуваної кількості місяців перебування кожної тварини у притулку після її прибуття.

Наукова новизна роботи полягає в методі прогнозування періоду перебування тварини у притулку, який застосовує для прогнозування модель, побудовану за допомогою ансамблю дерев рішень на основі алгоритму градієнтного бустингу, виконуючи визначення періоду перебування на основі кількості днів та встановлюючи порядок формування даних для навчання моделі.

Експериментальне дослідження виконано за допомогою даних притулків для тварин у Блумінгтоні, у які тварини потрапляли протягом 2017-2020 років. Дані було оброблено, виключаючи некоректні. Створено і застосовано моделі CART і XGBoost, перевірено параметри моделі XGBoost. Створена за методом, розробленим у роботі, модель мала на 11,7 % кращий результат ніж у моделі CART.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Притулок для тварин Сіріус. Головна [Електрон. ресурс]. – Режим доступу : <https://dogcat.com.ua/>.
2. Притулок для безхатніх тварин “В добрі руки” [Електрон. ресурс]. – Режим доступу : <https://dobri-ruky.com.ua/>.
3. Взяти тварину з притулку [Електрон. ресурс] : PetHelp – всеукраїнська база притулків. – Режим доступу : <https://pethelp.com.ua/>.
4. Urgent Need for Pet Adoption – Find Dogs & Cats & More. Petfinder [Electronic resource]. – Access mode : <https://www.petfinder.com/>.
5. Yang, A. [Electronic resource] / A. Yang. – Access mode : <https://medium.com/@chenghanyang728/decision-trees-classification-and-regression-tree-cart-algorithm-c1c4ac10f826>.
6. What is XGBoost? H2O.ai [Electronic resource]. – Access mode : <https://h2o.ai/wiki/xgboost/>.
7. What is XGBoost? Introduction to XGBoost Algorithm in ML [Electronic resource]. – Access mode : <https://www.analyticsvidhya.com/blog/2018/09/an-end-to-end-guide-to-understand-the-math-behind-xgboost/>.
8. Sen, V. Gradient Boosting and XGBoost explained in detail [Electronic resource] / V. Sen. – Access mode : <https://www.linkedin.com/pulse/gradient-boosting-xgboost-explained-detail-vishwajit-sen>.
9. Django [Electronic resource]. – Access mode : <https://github.com/django>.
10. Joomla! Content Management System to build websites & apps [Electronic resource]. – Access mode : <https://www.joomla.org/about-joomla.html>.
11. Little Fashion HTML Template by Tooplate [Electronic resource]. – Access mode : <https://www.tooplate.com/view/2127-little-fashion>.
12. Bonnie, J. Animal Care and Control Adopted Animals [Electronic resource] / J. Bonnie. – Access mode : <https://www.kaggle.com/datasets/jinbonnie/animal-data/data>.

ДОДАТОК А
Технічне завдання

Вступ

Притулки для тварин діють у постійному циклі, адже коли тварина знаходить свого старого або нового господаря, притулок може розмістити в себе іншу тварину. Тому знання того, коли можна очікувати вибуття тварини з притулку, є важливим для планування діяльності притулку для тварин, адже дозволяє розуміти, з якою кількістю тварин вдасться попрацювати притулку за деякий період. Окрім самого планування при цьому потрібно забезпечити внесення всіх пов'язаних даних, зручний пошук, надати певні можливості не тільки працівникам притулків для тварин, але і господарям тварин. Програму, яка буде підтримувати ці сторони роботи притулків для тварин, потрібно створити в роботі.

A.1 Підстави для розробки

Наказ № 437 від 14 листопада 2023 р. за Національним університетом «Запорізька політехніка» з зазначенням теми роботи «Дослідження та програмна реалізація методів підтримки планування діяльності притулку для тварин» та завдання на роботу є підставами для розробки.

A.2 Призначення розробки

Програма має підтримувати діяльність притулку для тварин, забезпечуючи реєстрацію дій, які виконуються в притулку для тварин, внесення даних про самі тварини в притулку і їхніми господарями, створення повідомлень господарями тварин про їх втрату, пошук серед повідомлень і реєстрацій, визначення відповідностей між повідомленнями господарів та реєстраціями тварин у притулках для тварин, планування діяльності притулків для тварин.

A.3 Основні вимоги до програми

A.3.1 Вимоги до функціональних характеристик

Забезпечення функціональних характеристик повинно вкладатися в наступний набір функцій:

- внесення даних про власну тварину користувачем у програму, включаючи додавання картки, її редагування за наявності та вилучення існуючої картки;
- реєстрація потрапляння тварини в притулок;
- редагування даних реєстрації потрапляння тварини в притулок;
- реєстрація менеджерів для роботи в притулку;
- вилучення даних незафіксованої реєстрації потрапляння тварини в притулок;
- пошук серед повідомлень про втрату тварини;
- пошук серед реєстрацій потрапляння тварин в притулок;
- визначення відповідності між картками про потенційно однакових тварин;
- вилучення відповідності між картками про потенційно однакових тварин;
- демонстрування картки тварини;
- демонстрування даних притулку для тварин;
- реєстрація нового притулку для тварин в програмі;
- редагування даних притулку для тварин;
- відображення зареєстрованих у програмі притулків для тварин;
- демонстрування останніх зареєстрованих тварин за притулком або за менеджером;
- демонстрування останніх зареєстрованих повідомлень про втрату тварин у місті;
- демонстрування картки лікування тварини;
- фіксація лікування тварини;

- редагування даних лікування тварини;
- визначення відповідності між твариною з повідомлення та твариною з притулку;
- вилучення відповідності між твариною з повідомлення та твариною з притулку;
- демонстрування відповідностей за карткою тварини;
- демонстрування створених користувачем відповідностей;
- створення повідомлення про втрату тварини;
- редагування повідомлення про втрату тварини;
- вилучення повідомлення про втрату тварини;
- демонстрування власних повідомлень про втрату тварин;
- авторизація менеджерів притулків для тварин і користувачів програми;
- реєстрування повернення тварини в притулок;
- редагування даних повернення тварини в притулок;
- демонстрування всіх зареєстрованих дій щодо тварини;
- реєстрація вибуття тварини з притулку;
- редагування даних реєстрації вибуття тварини з притулку;
- реєстрація переведення тварини в інший притулок;
- редагування даних реєстрації переведення тварини в інший притулок;
- демонстрування останніх зареєстрованих повернутих тварин за притулком або за менеджером;
- демонстрування останніх зареєстрованих тварин за вибуттям за притулком або за менеджером;
- демонстрування останніх зареєстрованих тварин за переведенням з притулку;
- демонстрування останніх зареєстрованих тварин за переведенням до притулку;

- демонстрування останніх зареєстрованих тварин за переведеннями між притулками за менеджером;
- додавання даних про місця для категорій тварин у притулку для тварин;
- редагування даних про місця для категорій тварин у притулку для тварин;
- прогнозування часу перебування тварини в притулку;
- демонстрування часу перебування тварин у притулку за категоріями;
- демонстрування кількості тварин у притулку;
- пошук серед реєстрацій потрапляння тварин в притулок;
- пошук серед реєстрацій повернення тварин у притулок;
- пошук серед реєстрацій переведень тварин між притулками;
- пошук серед реєстрацій вибуття тварин з притулку;
- пошук серед реєстрацій за притулком;
- пошук серед фіксацій лікувань тварин;
- демонстрування лікувань тварин за менеджером;
- демонстрування реєстрацій за притулком;
- демонстрування карток власних тварин господаря;
- демонстрування менеджерів, що працюють у притулку для тварин.

A.3.2 Умови експлуатації

Програмне забезпечення в підсумку має підтримуватись під час експлуатації через:

- організацію хостингу для роботи з сервером;
- розміщення на сервері вебзастосунку з базовим налаштуванням Django;
- наявність інтерпретатора python 3.

Або у складі цього серверу, або як окремий сервер БД має бути

використано і налаштовано систему MySQL.

А.3.3 Вимоги до складу та параметрів технічних засобів

Вимоги до організації технічних засобів сервера включають мінімально процесор Dual Intel Xeon, який має частоту 3,2 ГГц, 4 Гб доступної оперативної пам'яті, 10 Гб доступної пам'яті на жорсткому диску.

А.4 Порядок контролю та приймання

Строки виконання дипломної кваліфікаційної роботи магістра визначені в завданні на роботу безпосередньо, включаючи строк подання роботи, календарний план виконання роботи з зазначеними етапами, результатами та строками. Захист роботи відбувається за оголошеним завчасно графіком захисту дипломних кваліфікаційних робіт і відповідно з графіком роботи екзаменаційної комісії.

ДОДАТОК Б
Текст програми

```

from .models import Petlostad
import datetime
from .models import Locationcity
from .models import Locationcountry
from django.shortcuts import render
from .models import Animalbreed
from .models import Petcategory
from django.db.models import Q
from .models import Shelter
from .models import Shelterregister
from .models import Movementregister
from django.http import HttpResponse

def search_pet_lost_ad(request):
    pet_lost_ads = { }
    set_parameters = False
    pet_ads = Petlostad.objects.filter(status = True)
    pet_ads = pet_ads.select_related('lost_city')
    pet_ads = pet_ads.select_related('lost_city__city_country')
    pet_ads = pet_ads.select_related('pet')
    pet_ads = pet_ads.select_related('pet_owner')
    pet_ads = pet_ads.select_related('pet_breed')
    if request.method == 'GET' and 'petname' in request.GET:
        pet_name = self.request.GET.get('petname')
        pet_ads = pet_ads.filter(pet__name__icontains =
pet_name)
        set_parameters = True
    if request.method == 'GET' and 'petbreed' in request.GET:
        pet_breed = self.request.GET.get('petbreed')
        breed = Animalbreed.objects.get(id = int(pet_breed))
        pet_ads = pet_ads.filter(pet__breed = breed)
        set_parameters = True
    if request.method == 'GET' and 'petcategory' in request.GET:
        pet_category = self.request.GET.get('petcategory')
        pet_category = Petcategory.objects.get(id =
int(pet_category))
        pet_ads = pet_ads.filter(pet__breed__petcategory =
pet_category)
        set_parameters = True
    if request.method == 'GET' and 'petsex' in request.GET:
        pet_sex = self.request.GET.get('petsex')
        pet_ads = pet_ads.filter(pet__sex = int(pet_sex))
        set_parameters = True
    if request.method == 'GET' and 'petchip' in request.GET:
        pet_chip = self.request.GET.get('petchip')
        pet_ads = pet_ads.filter(pet__chipnumber = pet_chip)
        set_parameters = True
    if request.method == 'GET' and 'petheight' in request.GET:
        pet_height = self.request.GET.get('petheight')

```

```

        pet_ads = pet_ads.filter(pet__height__gte =
int(pet_height) - 5)
        pet_ads = pet_ads.filter(pet__height__lte =
int(pet_height) + 5)
        set_parameters = True
        if request.method == 'GET' and 'petweight' in request.GET:
            pet_weight = self.request.GET.get('petweight')
            pet_ads = pet_ads.filter(pet__weight__gte =
int(pet_weight) - 10)
            pet_ads = pet_ads.filter(pet__weight__lte =
int(pet_weight) + 10)
            set_parameters = True
        if request.method == 'GET' and 'woolcolor' in request.GET:
            pet_wool = self.request.GET.get('woolcolor')
            pet_ads = pet_ads.filter(pet__woolcolor = int(pet_wool))
            set_parameters = True
        if request.method == 'GET' and 'petage' in request.GET and
'petinterval' in request.GET:
            pet_age = self.request.GET.get('petage')
            pet_age_interval = self.request.GET.get('petinterval')
            pet_ads = pet_ads.filter(pet__age__gte = int(pet_age) -
int(pet_age_interval))
            pet_ads = pet_ads.filter(pet__age__lte = int(pet_age) +
int(pet_age_interval))
            set_parameters = True
        elif request.method == 'GET' and 'petage' in request.GET:
            pet_age = self.request.GET.get('petage')
            pet_age_interval = self.request.GET.get('petinterval')
            pet_ads = pet_ads.filter(pet__age__gte = int(pet_age) -
5)
            pet_ads = pet_ads.filter(pet__age__lte = int(pet_age) +
5)
            set_parameters = True
        if request.method == 'GET' and 'petcity' in request.GET:
            pet_lost_city = self.request.GET.get('petcity')
            location_city = Locationcity.objects.get(id =
int(pet_lost_city))
            pet_ads = pet_ads.filter(lost_city = location_city)
            set_parameters = True
        if request.method == 'GET' and 'petcountry' in request.GET:
            pet_lost_country = self.request.GET.get('petcountry')
            location_country = Locationcountry.objects.get(id =
int(pet_lost_country))
            pet_ads = pet_ads.filter(lost_city__city__country =
location_country)
            set_parameters = True
        if request.method == 'GET' and 'daypetdate1' in request.GET
and 'monthpetdate1' in request.GET and 'yearpetdate1' in request.GET:

```

```

        pet_lost_date_from_day =
self.request.GET.get('daypetdate1')
        pet_lost_date_from_month =
self.request.GET.get('monthpetdate1')
        pet_lost_date_from_year =
self.request.GET.get('yearpetdate1')
        pet_lost_date_from =
datetime.datetime(int(pet_lost_date_from_day),
int(pet_lost_date_from_month), int(pet_lost_date_from_year))
        pet_ads = pet_ads.filter(lost_date__gte =
pet_lost_date_from)
        set_parameters = True
        elif request.method == 'GET' and 'monthpetdate1' in
request.GET and 'yearpetdate1' in request.GET:
            pet_lost_date_from_month =
self.request.GET.get('monthpetdate1')
            pet_lost_date_from_year =
self.request.GET.get('yearpetdate1')
            pet_lost_date_from = datetime.datetime(1,
int(pet_lost_date_from_month), int(pet_lost_date_from_year))
            pet_ads = pet_ads.filter(lost_date__gte =
pet_lost_date_from)
            set_parameters = True
            if request.method == 'GET' and 'daypetdate2' in request.GET
and 'monthpetdate2' in request.GET and 'yearpetdate2' in request.GET:
                pet_lost_date_to_day =
self.request.GET.get('daypetdate2')
                pet_lost_date_to_month =
self.request.GET.get('monthpetdate2')
                pet_lost_date_to_year =
self.request.GET.get('yearpetdate2')
                pet_lost_date_to =
datetime.datetime(int(pet_lost_date_to_day),
int(pet_lost_date_to_month), int(pet_lost_date_to_year))
                pet_ads = pet_ads.filter(lost_date__lte =
pet_lost_date_to)
                set_parameters = True
                elif request.method == 'GET' and 'monthpetdate2' in
request.GET and 'yearpetdate2' in request.GET:
                    pet_lost_date_to_month =
self.request.GET.get('monthpetdate2')
                    pet_lost_date_to_year =
self.request.GET.get('yearpetdate2')
                    if pet_lost_date_to_month == 12:
                        pet_lost_date_to = datetime.datetime(1, 1,
int(pet_lost_date_to_year) + 1)
                    else:
                        pet_lost_date_to = datetime.datetime(1,
int(pet_lost_date_to_month) + 1, int(pet_lost_date_to_year))

```

```

        pet_ads = pet_ads.filter(lost_date__lt =
pet_lost_date_to)
        set_parameters = True
        if set_parameters:
            pet_lost_ads["ads"] = pet_ads
            if request.method == 'GET' and 'petbreed' in
request.GET:
                pet_breed = self.request.GET.get('petbreed')

                pet_lost_ads["breed"] = pet_breed
            if request.method == 'GET' and 'petcategory' in
request.GET:
                pet_category = self.request.GET.get('petcategory')
                pet_lost_ads["category"] = pet_category
            if request.method == 'GET' and 'petcity' in request.GET:
                pet_lost_city = self.request.GET.get('petcity')
                pet_lost_ads["city"] = pet_lost_city
            if request.method == 'GET' and 'petcountry' in
request.GET:
                pet_lost_country =
self.request.GET.get('petcountry')
                pet_lost_ads["country"] = pet_lost_country
                pet_lost_ads["breeds"] = Animalbreed.objects.all()
                pet_lost_ads["categories"] = Petcategory.objects.all()
                pet_lost_ads["cities"] = Locationcity.objects.all()
                pet_lost_ads["countries"] =
Locationcountry.objects.all()
            else:
                pet_lost_ads["breeds"] = Animalbreed.objects.all()
                pet_lost_ads["categories"] = Petcategory.objects.all()
                pet_lost_ads["cities"] = Locationcity.objects.all()
                pet_lost_ads["countries"] =
Locationcountry.objects.all()
        return render(request, "anisheltersoft/search_ads.html",
pet_lost_ads)

from .models import Shelterregister

def search_shelter_register(request):
    pet_register_data = { }
    set_parameters = False
    pet_registers = Shelterregister.objects.filter(status =
True, open = True)
    pet_registers =
pet_registers.select_related("pet_condition")
    pet_registers = pet_registers.select_related("shelter")
    pet_registers = pet_registers.select_related("pet")
    pet_registers = pet_registers.select_related("reason")
    pet_registers = pet_registers.select_related("take_type")

```

```

pet_registers = pet_registers.select_related("manager")
if request.method == 'GET' and 'petname' in request.GET:
    pet_name = self.request.GET.get('petname')
    pet_registers =
pet_registers.filter(pet__name__icontains = pet_name)
    set_parameters = True
if request.method == 'GET' and 'petbreed' in request.GET:
    pet_breed = self.request.GET.get('petbreed')
    breed = Animalbreed.objects.get(id = int(pet_breed))
    pet_registers = pet_registers.filter(pet__breed = breed)
    set_parameters = True
if request.method == 'GET' and 'petcategory' in request.GET:
    pet_category = self.request.GET.get('petcategory')
    pet_category = Petcategory.objects.get(id =
int(pet_category))
    pet_registers =
pet_registers.filter(pet__breed__petcategory = pet_category)
    set_parameters = True
if request.method == 'GET' and 'shelter' in request.GET:
    pet_shelter = self.request.GET.get('petsshelter')
    pet_shelter = Shelter.objects.get(id = int(pet_shelter))
    pet_registers = pet_registers.filter(shelter =
pet_shelter)
    set_parameters = True
if request.method == 'GET' and 'petsex' in request.GET:
    pet_sex = self.request.GET.get('petsex')
    pet_registers = pet_registers.filter(pet__sex =
int(pet_sex))
    set_parameters = True
if request.method == 'GET' and 'petchip' in request.GET:
    pet_chip = self.request.GET.get('petchip')
    pet_registers = pet_registers.filter(pet__chipnumber =
pet_chip)
    set_parameters = True
if request.method == 'GET' and 'petheight' in request.GET:
    pet_height = self.request.GET.get('petheight')
    pet_registers = pet_registers.filter(pet__height__gte =
int(pet_height) - 5)
    pet_registers = pet_registers.filter(pet__height__lte =
int(pet_height) + 5)
    set_parameters = True
if request.method == 'GET' and 'petweight' in request.GET:
    pet_weight = self.request.GET.get('petweight')
    pet_registers = pet_registers.filter(pet__weight__gte =
int(pet_weight) - 10)
    pet_registers = pet_registers.filter(pet__weight__lte =
int(pet_weight) + 10)
    set_parameters = True
if request.method == 'GET' and 'woolcolor' in request.GET:

```

```

        pet_wool = self.request.GET.get('woolcolor')
        pet_registers = pet_registers.filter(pet__woolcolor =
int(pet_wool))
        set_parameters = True
        if request.method == 'GET' and 'petage' in request.GET and
'petinterval' in request.GET:
            pet_age = self.request.GET.get('petage')
            pet_age_interval = self.request.GET.get('petinterval')
            pet_registers = pet_registers.filter(Q(pet__age__gte =
int(pet_age) - int(pet_age_interval)) | Q(pet__age__gte = int(pet_age)
- int(pet_age_interval)))
            pet_registers = pet_registers.filter(Q(pet__age__lte =
int(pet_age) + int(pet_age_interval)) | Q(pet__age__lte = int(pet_age)
+ int(pet_age_interval)))
            set_parameters = True
        elif request.method == 'GET' and 'petage' in request.GET:
            pet_age = self.request.GET.get('petage')
            pet_age_interval = self.request.GET.get('petinterval')
            pet_registers = pet_registers.filter(Q(pet__age__gte =
int(pet_age) - 5) | Q(pet__age__gte = int(pet_age) - 5))
            pet_registers = pet_registers.filter(Q(pet__age__lte =
int(pet_age) + 5) | Q(pet__age__lte = int(pet_age) + 5))
            set_parameters = True
        if request.method == 'GET' and 'petcity' in request.GET:
            pet_lost_city = self.request.GET.get('petcity')
            location_city = Locationcity.objects.get(id =
int(pet_lost_city))
            pet_registers = pet_registers.filter(found_city =
location_city)
            set_parameters = True
        if request.method == 'GET' and 'petcountry' in request.GET:
            pet_lost_country = self.request.GET.get('petcountry')
            location_country = Locationcountry.objects.get(id =
int(pet_lost_country))
            pet_registers =
pet_registers.filter(found_city__city_country = location_country)
            set_parameters = True
        if request.method == 'GET' and 'daypetdate1' in request.GET
and 'monthpetdate1' in request.GET and 'yearpetdate1' in request.GET:
            pet_lost_date_from_day =
self.request.GET.get('daypetdate1')
            pet_lost_date_from_month =
self.request.GET.get('monthpetdate1')
            pet_lost_date_from_year =
self.request.GET.get('yearpetdate1')
            pet_lost_date_from =
datetime.datetime(int(pet_lost_date_from_day),
int(pet_lost_date_from_month), int(pet_lost_date_from_year))

```



```

        if request.method == 'GET' and 'petcategory' in
request.GET:
            pet_category = self.request.GET.get('petcategory')
            petregisters["category"] = pet_category
        if request.method == 'GET' and 'petcity' in request.GET:
            pet_lost_city = self.request.GET.get('petcity')
            petregisters["city"] = pet_lost_city
        if request.method == 'GET' and 'petcountry' in
request.GET:
            pet_lost_country =
self.request.GET.get('petcountry')
            petregisters["country"] = pet_lost_country
        if request.method == 'GET' and 'petsshelter' in
request.GET:
            pet_shelter = self.request.GET.get('petsshelter')
            petregisters["shelter"] = pet_shelter
            petregisters["breeds"] = Animalbreed.objects.all()
            petregisters["categories"] = Petcategory.objects.all()
            petregisters["cities"] = Locationcity.objects.all()
            petregisters["countries"] =
Locationcountry.objects.all()
            petregisters["shelters"] = Shelter.objects.all()
        else:
            petregisters["breeds"] = Animalbreed.objects.all()
            petregisters["categories"] = Petcategory.objects.all()
            petregisters["cities"] = Locationcity.objects.all()
            petregisters["countries"] =
Locationcountry.objects.all()
            petregisters["shelters"] = Shelter.objects.all()
        return render(request,
"anisheltersoft/search_registers.html", petregisters)

from .models import Pet
from .models import Client
from .models import Manager
from .models import Pettherapy
from .models import Shelterregisterpossibility
from .models import Petlostadpossibility

def pet_card(request, pet_id):
    pet_card_info = Pet.objects.filter(id = pet_id)
    pet_card_info = pet_card_info.select_related("owner")
    pet_card_info = pet_card_info.select_related("breed")
    pet_card_info = pet_card_info.prefetch_related("photos")
    pet_card_info =
pet_card_info.select_related("location_city")
    pet_card_info =
pet_card_info.select_related("location_city__city_country")

```

```

        client_account = Client.objects.filter(account =
request.user)
        manager_account = Manager.objects.filter(account =
request.user)
        register = None
        if not pet_card_info.owner:
            register = Shelterregister.objects.filter(pet =
pet_card_info).select_related("shelter").order_by("-
register_date").values[0]
        pets = []
        if client_account and client_account.count() > 0:
            pets = Pet.objects.filter(owner = client_account)
        therapies = []
        possibilities = []
        if client_account.values()[0] == pet_card_info.owner or
manager_account.count() > 0:
            therapies = Pettherapy.objects.filter(pet =
pet_card_info)
            therapies = therapies.select_related("therapy_type")
            therapies = therapies.select_related("manager")
            therapies = therapies.order_by("-
therapy_date").values()[0:2]
            shelter_possibilities =
Shelterregisterpossibility.objects.filter(shelterregister__pet =
pet_card_info)
            shelter_possibilities =
shelter_possibilities.select_related("client")
            shelter_possibilities =
shelter_possibilities.select_related("petlostad")
            shelter_possibilities =
shelter_possibilities.select_related("shelterregister")
            shelter_possibilities =
shelter_possibilities.select_related("shelterregister__pet")
            shelter_possibilities =
shelter_possibilities.select_related("shelterregister__shelter")
            shelter_possibilities =
shelter_possibilities.select_related("shelterregister__manager")
            shelter_possibilities =
shelter_possibilities.select_related("shelterregister__take_type")
            shelter_possibilities =
shelter_possibilities.select_related("shelterregister__reason")
            shelter_possibilities =
shelter_possibilities.select_related("shelterregister__found_city")
            shelter_possibilities =
shelter_possibilities.select_related("shelterregister__pet_condition")
            shelter_possibilities =
shelter_possibilities.order_by("-add_date").values()[0:2]
            ad_possibilities =
Petlostadpossibility.objects.filter(petlostad__pet = pet_card_info)

```

```

        ad_possibilities =
ad_possibilities.select_related("petlostad")
        ad_possibilities =
ad_possibilities.select_related("petlostad__pet")
        ad_possibilities =
ad_possibilities.select_related("petlostad__lost_city")
        ad_possibilities =
ad_possibilities.select_related("manager")
        ad_possibilities =
ad_possibilities.select_related("shelterregister")
        ad_possibilities =
ad_possibilities.select_related("shelterregister__pet")
        ad_possibilities =
ad_possibilities.select_related("shelterregister__shelter")
        ad_possibilities =
ad_possibilities.select_related("shelterregister__manager")
        ad_possibilities =
ad_possibilities.select_related("shelterregister__take_type")
        ad_possibilities =
ad_possibilities.select_related("shelterregister__reason")
        ad_possibilities =
ad_possibilities.select_related("shelterregister__found_city")
        ad_possibilities =
ad_possibilities.select_related("shelterregister__pet_condition")
        ad_possibilities = ad_possibilities.order_by("-
add_date").values()[:2]
        pet_possibilities =
Petpossibility.objects.filter(Q(pet_source = pet_card_info) |
Q(pet_possible = pet_card_info))
        pet_possibilities =
pet_possibilities.select_related("manager")
        pet_possibilities =
pet_possibilities.select_related("pet_source__owner")
        pet_possibilities =
pet_possibilities.select_related("pet_source__breed")
        pet_possibilities =
pet_possibilities.select_related("pet_source__location_city")
        pet_possibilities =
pet_possibilities.select_related("pet_source__location_city__city_count
ry")
        pet_possibilities =
pet_possibilities.select_related("pet_possible__owner")
        pet_possibilities =
pet_possibilities.select_related("pet_possible__breed")
        pet_possibilities =
pet_possibilities.select_related("pet_possible__location_city")
        pet_possibilities =
pet_possibilities.select_related("pet_possible__location_city__city_cou
ntry")

```

```

        pet_possibilities = pet_possibilities.order_by("-
id").values()[:2]
        card = {}
        card["card"] = pet_card_info
        card["therapies"] = therapies
        possibilities = [shelter_possibilities, ad_possibilities,
pet_possibilities]
        card["possibilities"] = possibilities
        card["register"] = register
        card["pets"] = pets
        return render(request, "anisheltersoft/card.html", card)

    def shelter_info(request, shelter_id):
        shelter_data = { }
        shelter_information = Shelter.objects.filter(id =
shelter_id)
        shelter_information =
shelter_information.select_related("location_city")
        shelter_information =
shelter_information.select_related("location_city__city_country")
        shelter_data ["shelter"] = shelter_information
        pet_registers = Shelterregister.objects.filter(status =
True, open = True, shelter = shelter_information)
        pet_registers =
pet_registers.select_related("pet_condition")
        pet_registers = pet_registers.select_related("shelter")
        pet_registers = pet_registers.select_related("pet")
        pet_registers = pet_registers.select_related("reason")
        pet_registers = pet_registers.select_related("take_type")
        pet_registers = pet_registers.select_related("manager")
        pet_registers = pet_registers.prefetch_related("categories")
        pet_registers = pet_registers.order_by("register_date")[:5]
        shelter_data["last_registers"] = pet_registers
        return render(request, "anisheltersoft/shelter.html",
shelter_data)

    def shelters_list(request):
        shelter_data = { }
        shelters = Shelter.objects.all()
        shelters = shelters.select_related("location_city")
        shelters =
shelters.select_related("location_city__city_country")
        shelter_data["shelters"] = shelters
        return render(request, "anisheltersoft/shelters.html",
shelter_data)

    def last_ads(request):
        ads_data = { }
        ads = Petlostad.objects.all()

```

```

ads = ads.select_related("location_city")
ads = ads.select_related("location_city__city__country")
ads = ads.order_by("-lost_date").values()[:50]
if request.method == 'GET' and 'petcity' in request.GET:
    pet_lost_city = self.request.GET.get('petcity')
    location_city = Locationcity.objects.get(id =
int(pet_lost_city))
    pet_ads = pet_ads.filter(lost_city = location_city)
    set_parameters = True
if set_parameters:
    ads_data["ads"] = ads
    if request.method == 'GET' and 'petcity' in request.GET:
        pet_lost_city = self.request.GET.get('petcity')
        ads_data["city"] = pet_lost_city
    ads_data["cities"] = Locationcity.objects.all()
else:
    ads_data["cities"] = Locationcity.objects.all()
return render(request, "anisheltersoft/last_ads.html",
ads_data)

from .forms import PetCardForm
from django.shortcuts import redirect

def pet_card_create(request):
    if request.user:
        proprietor = User.objects.filter(account = request.user)
        manager = Manager.objects.filter(account = request.user)
        if proprietor and proprietor.count() > 0:
            userType = 0
        elif manager and manager.count() > 0:
            userType = 1
        if request.method == "POST":
            pet_data_from_form = PetCardForm(request.POST)
            if pet_data_from_form.is_valid():
                pet = pet_data_from_form.save(commit = False)
                if userType == 0:
                    pet.owner = proprietor
                elif userType == 1:
                    pet.last_manager = manager
                pet.save()
                return redirect('petcardview', pet_id = pet.id)
            else:
                pet_data_from_form = None
        return render(request,
"anisheltersoft/pet_card_add.html", {"form": pet_data_from_form })
    else:
        return render(request,
"anisheltersoft/pet_card_add.html", {"form": None })

```

```

def pet_card_change(request, pet_card_id):
    if request.user:
        pet_card_info = Pet.objects.filter(id = pet_id)
        pet_card_info = pet_card_info.select_related("owner")
        pet_card_info = pet_card_info.select_related("breed")
        pet_card_info =
pet_card_info.select_related("location_city")
        pet_card_info =
pet_card_info.select_related("location_city__city_country")
        proprietor = User.objects.filter(account = request.user)
        proprietorFlag = False
        manager = Manager.objects.filter(account = request.user)
        managerFlag = False
        if request.method == "POST" and
pet_card_info.source_type == True and pet_card_info.owner ==
proprietor.values()[0]:
            proprietorFlag = True
            if request.method == "POST" and
pet_card_info.source_type == False and manager.count() >= 0:
                managerFlag = True
            if proprietorFlag or managerFlag:
                pet_data_from_form = PetCardForm(request.POST or
None, instance = pet_card_info)
                if pet_data_from_form.is_valid() and manager and
manager.count() > 0:
                    pet_data_from_form.last_manager = manager
                    pet_data_from_form.save()
                    return redirect('petcardview', pet_id =
pet_card_info.id)
                else:
                    pet_data_from_form = None
            return render(request,
"anisheltersoft/pet_card_change.html", {"form": pet_data_from_form })

def pet_therapies(request, pet_id):
    pet_card_info = Pet.objects.filter(id = pet_id)
    pet_card_info = pet_card_info.select_related("owner")
    pet_card_info = pet_card_info.select_related("breed")
    pet_card_info =
pet_card_info.select_related("location_city")
    pet_card_info =
pet_card_info.select_related("location_city__city_country")
    client_account = Client.objects.filter(account =
request.user)
    manager_account = Manager.objects.filter(account =
request.user)
    therapies = []
    if client_account.values()[0] == pet_card_info.owner or
manager_account.count() > 0:

```

```

        therapies = Pettherapy.objects.filter(pet =
pet_card_info)
        therapies = therapies.select_related("therapy_type")
        therapies = therapies.select_related("manager")
        therapies = therapies.order_by("-therapy_date")
        card = {}
        card["card"] = pet_card_info
        card["therapies"] = therapies
        return render(request, "anisheltersoft/therapies.html",
card)

def pet_possibilities(request, pet_id):
    pet_card_info = Pet.objects.filter(id = pet_id)
    pet_card_info = pet_card_info.select_related("owner")
    pet_card_info = pet_card_info.select_related("breed")
    pet_card_info =
pet_card_info.select_related("location_city")
    pet_card_info =
pet_card_info.select_related("location_city__city_country")
    client_account = Client.objects.filter(account =
request.user)
    manager_account = Manager.objects.filter(account =
request.user)
    therapies = []
    possibilities = []
    if client_account.values()[0] == pet_card_info.owner or
manager_account.count() > 0:
        shelter_possibilities =
Shelterregisterpossibility.objects.filter(shelterregister__pet =
pet_card_info)
        shelter_possibilities =
shelter_possibilities.select_related("client")
        shelter_possibilities =
shelter_possibilities.select_related("petlostad")
        shelter_possibilities =
shelter_possibilities.select_related("shelterregister")
        shelter_possibilities =
shelter_possibilities.select_related("shelterregister__pet")
        shelter_possibilities =
shelter_possibilities.select_related("shelterregister__shelter")
        shelter_possibilities =
shelter_possibilities.select_related("shelterregister__manager")
        shelter_possibilities =
shelter_possibilities.select_related("shelterregister__take_type")
        shelter_possibilities =
shelter_possibilities.select_related("shelterregister__reason")
        shelter_possibilities =
shelter_possibilities.select_related("shelterregister__found_city")

```

```

        shelter_possibilities =
shelter_possibilities.select_related("shelterregister__pet_condition")
        shelter_possibilities =
shelter_possibilities.order_by("-add_date").values()
        ad_possibilities =
Petlostadpossibility.objects.filter(petlostad__pet = pet_card_info)
        ad_possibilities =
ad_possibilities.select_related("petlostad")
        ad_possibilities =
ad_possibilities.select_related("petlostad__pet")
        ad_possibilities =
ad_possibilities.select_related("petlostad__lost_city")
        ad_possibilities =
ad_possibilities.select_related("manager")
        ad_possibilities =
ad_possibilities.select_related("shelterregister")
        ad_possibilities =
ad_possibilities.select_related("shelterregister__pet")
        ad_possibilities =
ad_possibilities.select_related("shelterregister__shelter")
        ad_possibilities =
ad_possibilities.select_related("shelterregister__manager")
        ad_possibilities =
ad_possibilities.select_related("shelterregister__take_type")
        ad_possibilities =
ad_possibilities.select_related("shelterregister__reason")
        ad_possibilities =
ad_possibilities.select_related("shelterregister__found_city")
        ad_possibilities =
ad_possibilities.select_related("shelterregister__pet_condition")
        ad_possibilities = ad_possibilities.order_by("-
add_date").values()
        pet_possibilities =
Petpossibility.objects.filter(Q(pet_source = pet_card_info) |
Q(pet_possible = pet_card_info))
        pet_possibilities =
pet_possibilities.select_related("manager")
        pet_possibilities =
pet_possibilities.select_related("pet_source__owner")
        pet_possibilities =
pet_possibilities.select_related("pet_source__breed")
        pet_possibilities =
pet_possibilities.select_related("pet_source__location_city")
        pet_possibilities =
pet_possibilities.select_related("pet_source__location_city__city_count
ry")
        pet_possibilities =
pet_possibilities.select_related("pet_possible__owner")

```

```

        pet_possibilities =
pet_possibilities.select_related("pet_possible__breed")
        pet_possibilities =
pet_possibilities.select_related("pet_possible__location_city")
        pet_possibilities =
pet_possibilities.select_related("pet_possible__location_city__city_cou
ntry")
        pet_possibilities = pet_possibilities.order_by("-
id").values()
        card = {}
        card["card"] = pet_card_info
        possibilities = [shelter_possibilities, ad_possibilities,
pet_possibilities]
        card["possibilities"] = possibilities
        return render(request, "anisheltersoft/possibilities.html",
card)

def user_possibilities(request):
    client_account = Client.objects.filter(account =
request.user)
    manager_account = Manager.objects.filter(account =
request.user)
    possibilities = []
    if client_account.count() > 0 or manager_account.count() >
0:
        client_account = False
        if client_account.count() > 0:
            client_account = client_account.values()[0]
            client_account = True
        manager_account = False
        if manager_account.count() > 0:
            manager_account = manager_account.values()[0]
            manager_account = True
        possibilities = []
        if client_account:
            shelter_possibilities =
Shelterregisterpossibility.objects.filter(client = client_account)
            shelter_possibilities =
shelter_possibilities.select_related("client")
            shelter_possibilities =
shelter_possibilities.select_related("petlostad")
            shelter_possibilities =
shelter_possibilities.select_related("shelterregister")
            shelter_possibilities =
shelter_possibilities.select_related("shelterregister__pet")
            shelter_possibilities =
shelter_possibilities.select_related("shelterregister__shelter")
            shelter_possibilities =
shelter_possibilities.select_related("shelterregister__manager")

```

```

        shelter_possibilities =
shelter_possibilities.select_related("shelterregister__take_type")
        shelter_possibilities =
shelter_possibilities.select_related("shelterregister__reason")
        shelter_possibilities =
shelter_possibilities.select_related("shelterregister__found_city")
        shelter_possibilities =
shelter_possibilities.select_related("shelterregister__pet_condition")
        shelter_possibilities =
shelter_possibilities.order_by("-add_date").values()
        possibilities = shelter_possibilities
    if manager_account:
        ad_possibilities =
Petlostadpossibility.objects.filter(manager = manager_account)
        ad_possibilities =
ad_possibilities.select_related("petlostad")
        ad_possibilities =
ad_possibilities.select_related("petlostad__pet")
        ad_possibilities =
ad_possibilities.select_related("petlostad__lost_city")
        ad_possibilities =
ad_possibilities.select_related("manager")
        ad_possibilities =
ad_possibilities.select_related("shelterregister")
        ad_possibilities =
ad_possibilities.select_related("shelterregister__pet")
        ad_possibilities =
ad_possibilities.select_related("shelterregister__shelter")
        ad_possibilities =
ad_possibilities.select_related("shelterregister__manager")
        ad_possibilities =
ad_possibilities.select_related("shelterregister__take_type")
        ad_possibilities =
ad_possibilities.select_related("shelterregister__reason")
        ad_possibilities =
ad_possibilities.select_related("shelterregister__found_city")
        ad_possibilities =
ad_possibilities.select_related("shelterregister__pet_condition")
        ad_possibilities = ad_possibilities.order_by("-
add_date").values()
        pet_possibilities =
Petpossibility.objects.filter(manager = manager_account)
        pet_possibilities =
pet_possibilities.select_related("manager")
        pet_possibilities =
pet_possibilities.select_related("pet_source__owner")
        pet_possibilities =
pet_possibilities.select_related("pet_source__breed")

```

```

        pet_possibilities =
pet_possibilities.select_related("pet_source__location_city")
        pet_possibilities =
pet_possibilities.select_related("pet_source__location_city__city_coun
try")

        pet_possibilities =
pet_possibilities.select_related("pet_possible__owner")
        pet_possibilities =
pet_possibilities.select_related("pet_possible__breed")
        pet_possibilities =
pet_possibilities.select_related("pet_possible__location_city")
        pet_possibilities =
pet_possibilities.select_related("pet_possible__location_city__city_co
untry")

        pet_possibilities = pet_possibilities.order_by("-
id").values()

        possibilities = [ad_possibilities,
pet_possibilities]
        return render(request,
"anisheltersoft/user_possibilities.html", possibilities)

def user_pet_cards(request):
    proprietor = User.objects.filter(account = request.user)
    pet_cards = []
    if proprietor and proprietor.count() > 0:
        pet_cards = Pet.objects.filter(owner =
proprietor.values[0])
        pet_cards = pet_cards.select_related("owner")
        pet_cards = pet_cards.select_related("breed")
        pet_cards = pet_cards.select_related("location_city")
        pet_cards =
pet_cards.select_related("location_city__city_country")
        pet_cards = pet_cards.order_by("-id")
        pet_cards_data = {}
        pet_cards_data["cards"] = pet_cards
        return render(request, "anisheltersoft/user_pet_cards.html",
pet_cards_data)

def user_pet_ads(request):
    proprietor = User.objects.filter(account = request.user)
    pet_ads = []
    if proprietor and proprietor.count() > 0:
        pet_ads = Petlostad.objects.filter(pet__owner =
proprietor.values[0])
        pet_ads = pet_ads.select_related("pet")
        pet_ads = pet_ads.select_related("pet__breed")
        pet_ads =
pet_ads.select_related("pet__breed__petcategory")
        pet_ads = pet_ads.select_related("location_city")

```

```

        pet_ads =
pet_ads.select_related("location_city__city_country")
        pet_ads = pet_ads.order_by("-id")
        pet_ads_data = {}
        pet_ads_data["ads"] = pet_ads
        return render(request, "anisheltersoft/user_pet_ads.html",
pet_ads_data)

from .models import Transfer
from .models import Movementregisterreturn

def pet_actions(request, pet_id):
    proprietor = User.objects.filter(account = request.user)
    if proprietor and proprietor.count() > 0:
        pet_card_info = Pet.objects.filter(id = pet_id)
        pet_card_info = pet_card_info.select_related("owner")
        pet_card_info = pet_card_info.select_related("breed")
        pet_card_info =
pet_card_info.select_related("location_city")
        pet_card_info =
pet_card_info.select_related("location_city__city_country")
        manager_account = Manager.objects.filter(account =
request.user)
        shelter_registers = []
        movement_registers = []
        returns = []
        transfers = []
        if client_account.values()[0] == pet_card_info.owner or
manager_account.count() > 0:
            shelter_registers = Shelterregister.objects.filter(pet =
pet_card_info)
            shelter_registers =
shelter_registers.select_related("pet_condition")
            shelter_registers =
shelter_registers.select_related("shelter")
            shelter_registers =
shelter_registers.select_related("pet")
            shelter_registers =
shelter_registers.select_related("reason")
            shelter_registers =
shelter_registers.select_related("take_type")
            shelter_registers =
shelter_registers.select_related("manager")
            shelter_registers = shelter_registers.order_by("-
register_date")
            transfers = Transfer.objects.filter(shelterregister__in
= shelter_registers)
            transfers = transfers.select_related("shelter")
            transfers = transfers.order_by("-id")

```

```

        movement_registers =
Movementregister.objects.filter(shelterregister__in =
shelter_registers)
        movement_registers =
movement_registers.select_related("pet_condition")
        movement_registers =
movement_registers.select_related("movement_type")
        movement_registers =
movement_registers.select_related("manager")
        movement_registers =
movement_registers.select_related("shelterregister")
        movement_registers = movement_registers.order_by("-
register_date")
        returns =
Movementregisterreturn.objects.filter(movementregister__in =
movement_registers)
        returns = returns.select_related("manager")
        returns = returns.select_related("movementregister")
        returns = returns.select_related("reason")
        returns = returns.order_by("-return_date")
        pet_actions = {}
        pet_actions["shelter_registers"] = shelter_registers
        pet_actions["movement_registers"] = movement_registers
        pet_actions["transfers"] = transfers
        pet_actions["returns"] = returns
        return render(request, "anisheltersoft/actions.html",
pet_actions)

def shelter_registration_possible_delete(request, sheltreg_id):
    proprietor = User.objects.filter(account = request.user)
    shelter_register = Shelterregisterpossibility.objects.get(id
= sheltreg_id)
    if shelter_register.client == proprietor.values()[0]:
        shelter_register.delete()
        return HttpResponse(status = 200)

def shelter_registration_possible(request, sheltreg_id, ad_id =
None):
    proprietor = User.objects.filter(account = request.user)
    shelter_register = Shelterregister.objects.get(id =
sheltreg_id)
    pet_card_info = pet_card_info.select_related("owner")
    pet_card_info = pet_card_info.select_related("breed")
    pet_card_info =
pet_card_info.select_related("location_city")
    pet_card_info =
pet_card_info.select_related("location_city__city_country")
    if ad_id:
        pet_ad = Petlostad.objects.get(id = ad_id)

```

```

        pet_message = self.request.GET.get('message')
        possible = Shelterregisterpossibility(petlostad =
pet_ad, shelterregister = shelter_register, client = client_account,
message = pet_message, add_date = datetime.datetime.now(), status = 0)
        possible.save()
        shelter_registration_data = {}
        shelter_registration_data["ads"] =
Petlostad.objects.filter(pet__owner = proprietor)
        shelter_registration_data["possible"] = possible
    else:
        shelter_registration_data = {}
        shelter_registration_data["ads"] =
Petlostad.objects.filter(pet__owner = proprietor)
        return render(request,
"anisheltersoft/shelter_possible.html", shelter_registration_data)

from .forms import PetlostadForm

def pet_ad_create(request):
    if request.user:
        proprietor = User.objects.get(account = request.user)
        if request.method == "POST":
            pet_data_from_form = PetlostadForm(request.POST)
            if pet_data_from_form.is_valid() and
pet_data_from_form.pet.owner == proprietor:
                pet_data_from_form.save()
                return redirect('petadview', petad_id =
pet_data_from_form.id)
            else:
                pet_data_from_form = None
                return render(request, "anisheltersoft/pet_ad_add.html",
{"form": pet_data_from_form })
        else:
            return render(request, "anisheltersoft/pet_ad_add.html",
{"form": None })

def pet_ad_change(request, petad_id):
    if request.user:
        pet_ad_info = Petlostad.objects.filter(id = petad_id)
        pet_ad_info = pet_ad_info.select_related("pet")
        pet_ad_info = pet_ad_info.select_related("lost_city")
        pet_ad_info = pet_ad_info.select_related("pet__owner")
        pet_ad_info = pet_ad_info.select_related("pet__breed")
        pet_ad_info =
pet_ad_info.select_related("lost_city__city_country")
        proprietor = User.objects.filter(account = request.user)
        if request.method == "POST" and pet_ad_info.pet.owner ==
proprietor.values()[0]:

```

```

        pet_data_from_form = PetlostadForm(request.POST or
None, instance = pet_ad_info)
        if pet_data_from_form.is_valid():
            pet_data_from_form.save()
            return redirect('petadview', petad_id =
pet_ad_info.id)
        else:
            pet_data_from_form = None
            return render(request,
"anisheltersoft/pet_ad_change.html", {"form": pet_data_from_form })

def pet_ad_delete(request, petad_id):
    if request.user:
        pet_ad_info = Petlostad.objects.filter(id =
petad_id).delete()
        return HttpResponse(status = 200)

def pet_ad(request, petad_id):
    pet_ad_info = Petlostad.objects.get(id = petad_id)
    pet_ad_info = pet_ad_info.select_related("pet")
    pet_ad_info = pet_ad_info.select_related("pet__owner")
    pet_ad_info = pet_ad_info.select_related("pet__breed")
    pet_ad_info = pet_ad_info.prefetch_related("pet__photos")
    pet_ad_info =
pet_ad_info.select_related("pet__location_city")
    pet_ad_info =
pet_ad_info.select_related("pet__location_city__city_country")
    pet_ad_info = pet_ad_info.select_related("lost_city")
    card = {}
    card["card"] = pet_ad_info
    return render(request, "anisheltersoft/ad.html", card)

def shelter_register_info(request, register_id):
    shelter_register = Shelterregister.objects.get(id =
register_id, status = True, open = True)
    shelter_register = shelter_register.select_related("pet")
    shelter_register =
shelter_register.select_related("pet__owner")
    shelter_register =
shelter_register.select_related("pet__breed")
    shelter_register =
shelter_register.prefetch_related("pet__photos")
    shelter_register =
shelter_register.select_related("pet__location_city")
    shelter_register =
shelter_register.select_related("pet__location_city__city_country")
    shelter_register =
shelter_register.select_related("shelter")

```

```
        shelter_register =
shelter_register.select_related("manager")
        shelter_register =
shelter_register.select_related("take_type")
        shelter_register = shelter_register.select_related("reason")
        shelter_register =
shelter_register.select_related("found_city")
        shelter_register =
shelter_register.select_related("pet_condition")
        card = {}
        card["card"] = shelter_register
        return render(request,
"anisheltersoft/shelter_register.html", card)
```

ДОДАТОК В
Слайди презентації

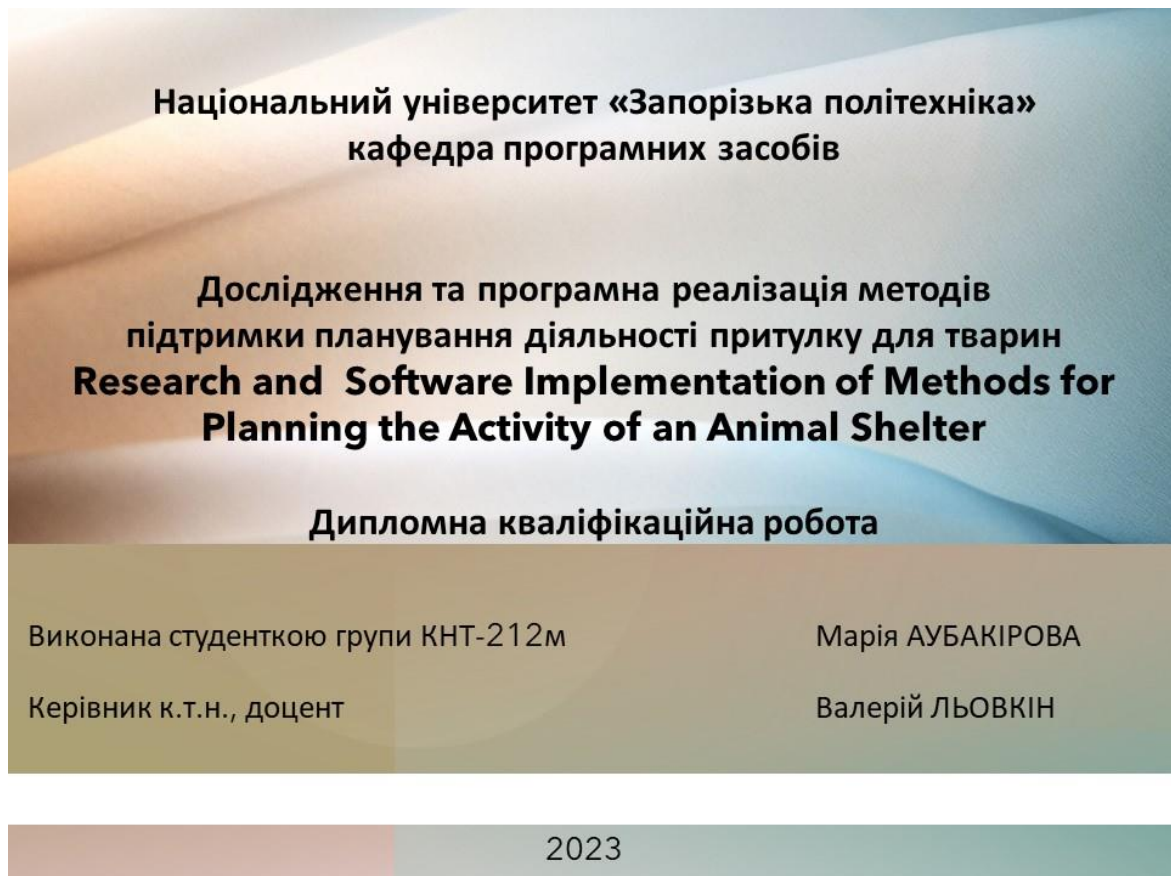


Рисунок В.1 – Слайд № 1 (Титульний слайд)

ОБ'ЄКТ, ПРЕДМЕТ ТА МЕТА РОБОТИ

- Об'єкт роботи – процес діяльності притулку для тварин.
- Предмет роботи – методи підтримки планування діяльності притулку для тварин.
- Мета роботи – розробити програмне забезпечення підтримки діяльності притулку для тварин для забезпечення можливостей планування діяльності на основі визначення очікуваної тривалості перебування тварини у притулку.



Рисунок В.2 – Слайд № 2 (Об'єкт, предмет та мета роботи)

ЗАВДАННЯ РОБОТИ

- проаналізувати аналоги програми підтримки діяльності притулку для тварин;
- розробити метод прогнозування періоду перебування тварини у притулку;
- виконати проектування програмного забезпечення підтримки діяльності притулку для тварин;
- реалізувати програмне забезпечення підтримки діяльності притулку для тварин.

3

Рисунок В.3 – Слайд № 3 (Завдання роботи)

ПОРІВНЯННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ПІДТРИМКИ ДІЯЛЬНОСТІ ПРИТУЛКУ ДЛЯ ТВАРИН

Критерій	Сіпіс	PetHelp	Petfinder	Anisheltersoft
Пошук тварин з притулків	+	+	+	+
Докладна інформація про тварину	-	+	+	+
Доступ різних притулків до програми	-	+	+	+
Реєстрація тварин притулками одразу в програмі	-	-	-	+
Реєстрація процедур лікування тварин	-	-	-	+
Прогнозування періоду перебування тварини у притулку	-	-	-	+

4

Рисунок В.4 – Слайд № 4 (Порівняння програмного забезпечення підтримки діяльності притулку для тварин)

МЕТОД ПРОГНОЗУВАННЯ ПЕРІОДУ ПЕРЕБУВАННЯ ТВАРИНИ У ПРИТУЛКУ

- збір даних, включаючи наступні властивості: порода тварини, категорія тварини, дата реєстрації потрапляння тварини у притулок, дата реєстрації вибуття тварини з притулку та дата потрапляння тварини у притулок, виконуючи перетворення на кількість днів;
- вилучення записів з пропущеними значеннями властивостей та дублікатів;
- визначення тривалості перебування тварини у притулку;
- створення моделі прогнозування періоду перебування тварини у притулку на основі створення ансамблю дерев рішень на основі методу XGBoost;
- навчання моделі на основі навчальної вибірки з визначенням тривалості перебування тварини у притулку для кожного екземпляра в якості вихідної ознаки;
- визначення нових зареєстрованих випадків реєстрації тварин у притулку та прогнозування кількості днів перебування тварини в притулку.

5

Рисунок В.5 – Слайд № 5 (Метод прогнозування періоду перебування тварини у притулку)

ПОРІВНЯННЯ DJANGO І JOOMLA ДЛЯ ВЕБРОЗРОБКИ

Критерій	Django	Joomla
Використовувана мова програмування	Python	PHP
Ліцензія	Відкрита	Відкрита
Рівень забезпечення безпеки	Високий	Низький
Складність підтримки великих програм	Нижча	Вища
Рівень можливостей інтегрування моделей прогнозування	Високий	Вбудований низький

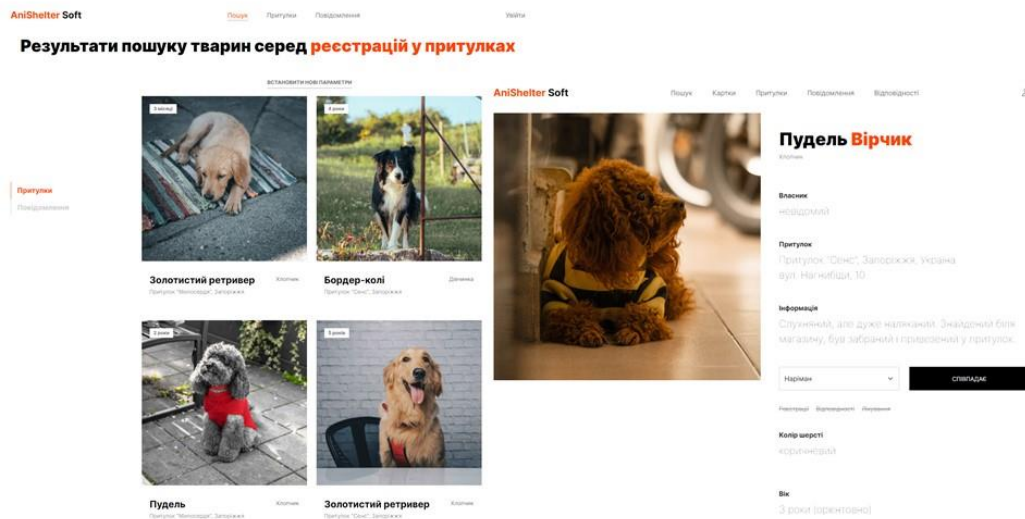
6

Рисунок В.6 – Слайд № 6 (Порівняння Django і Joomla для веброзробки)

ДІАГРАМИ ПРЕЦЕДЕНТІВ ДЛЯ МЕНЕДЖЕРА ПРИТУЛКУ ДЛЯ ТВАРИН ТА АДМІНІСТРАТОРА

Рисунок В.8 – Слайд № 8 (Діаграми прецедентів для менеджера притулку для тварин та адміністратора)

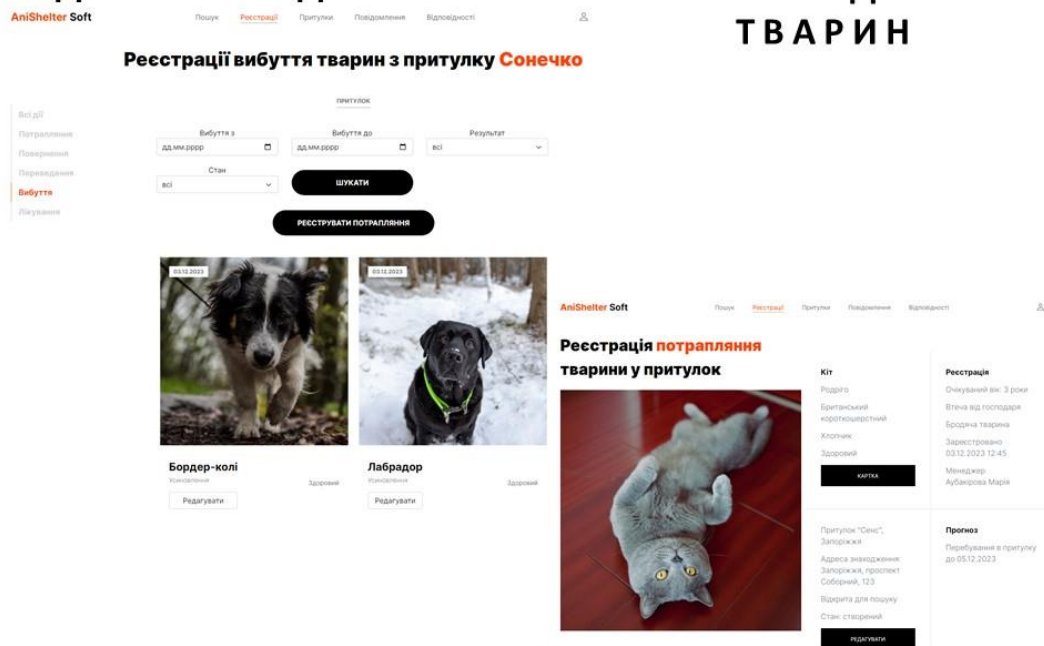
ВЕБСТОРІНКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ПІДТРИМКИ ДІЯЛЬНОСТІ ПРИТУЛКУ ДЛЯ ТВАРИН



11

Рисунок В.11 – Слайд № 11 (Вебсторінки програмного забезпечення підтримки діяльності притулку для тварин)

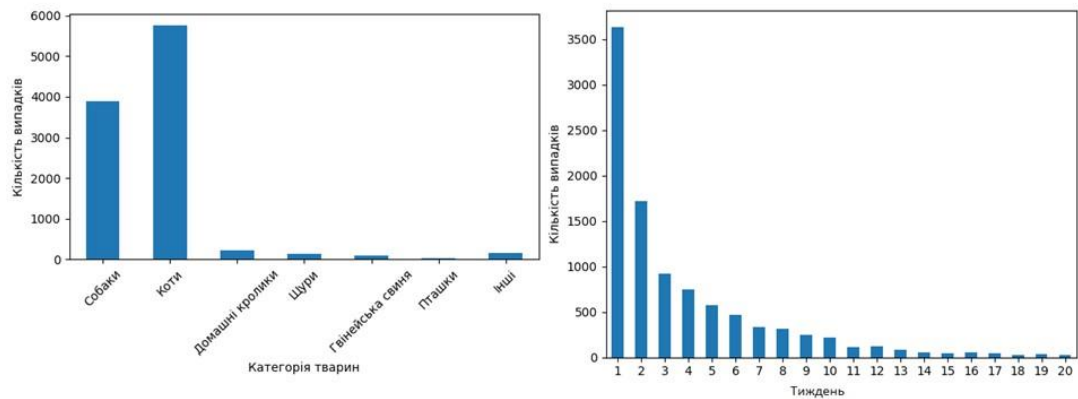
ВЕБСТОРІНКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ПІДТРИМКИ ДІЯЛЬНОСТІ ПРИТУЛКУ ДЛЯ ТВАРИН



12

Рисунок В.12 – Слайд № 12 (Вебсторінки програмного забезпечення підтримки діяльності притулку для тварин)

ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ



13

Рисунок В.13 – Слайд № 13 (Експериментальне дослідження)

РЕЗУЛЬТАТИ ЕКСПЕРИМЕНТАЛЬНОГО ДОСЛІДЖЕННЯ

Модель	Дата потрапляння	Причина	Порода	Колір	Категорія	Вік	Середня абсолютна похибка
XGBoost	+	-	+	-	-	-	0,01338571
XGBoost	+	-	+	-	+	+	0,01364843
XGBoost	+	-	+	+	+	-	0,01382266
XGBoost	+	-	+	+	+	+	0,01411962
XGBoost	+	+	+	+	+	+	0,01387665
XGBoost	+	+	+	-	+	-	0,01356873
XGBoost	+	-	+	-	+	-	0,01331958

Модель	Середня абсолютна похибка
CART	0,01508935
XGBoost (10 раундів бустингу, глибина 3)	0,01353547
XGBoost (40 раундів бустингу, глибина 8)	0,01345309
XGBoost (40 раундів бустингу, глибина 3)	0,01349237
XGBoost (1000 раундів бустингу, глибина 6)	0,01381536
XGBoost (100 раундів бустингу, глибина 6)	0,01348275
XGBoost (40 раундів бустингу, глибина 6)	0,01331958

14

Рисунок В.14 – Слайд № 14 (Результати експериментального дослідження)

ВИСНОВКИ

- Усі завдання, а відповідно і мету роботи, було виконано і досягнуто, включаючи аналіз аналогів програми, розробку методу прогнозування періоду перебування тварини у притулку, проєктування і реалізацію власного програмного забезпечення підтримки діяльності притулку для тварин.
- Створена програма підтримує діяльність притулку для тварин, забезпечує реєстрацію дій, які виконуються в притулку для тварин, внесення даних про самих тварин у притулку і їхніми господарями, створення повідомлень господарями тварин про їх втрату, пошук серед повідомлень і реєстрацій, визначення відповідностей між повідомленнями господарів та реєстраціями тварин у притулках для тварин, планування діяльності притулків для тварин на основі визначення очікуваної кількості місяців перебування кожної тварини у притулку після її прибуття.
- Наукова новизна роботи полягає в методі прогнозування періоду перебування тварини у притулку, який застосовує для прогнозування модель, побудовану за допомогою ансамблю дерев рішень на основі алгоритму градієнтного бустингу, виконуючи визначення періоду перебування на основі кількості днів та встановлюючи порядок формування даних для навчання моделі.

Рисунок В.15 – Слайд № 15 (Висновки)