Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)
Кафедра Інформатики
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

РОЗРОБКА КРОСПЛАТФОРМЕНОГО ЗАСТОСУНКУ ЩОДО ДОПОМОГИ ТВАРИНАМ
(тема)

Виконав:
студент 4 курсу, групи ІТІНФ-20-2
Бєгунова В.Д.
(прізвище, ініціали)

Спеціальності 122 Комп'ютерні науки
(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

Керівник доц. Кобилін О.А.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ Кобилін О.А.
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)

Кафедра Інформатики
(повна назва)

Рівень вищої освіти перший (бакалаврський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Бєгуновій Вероніці Дмитрівні
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка кросплатформеного застосунку щодо допомоги тваринам

затверджена наказом університету від 20 травня 2024 року № 464 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 27 травня 2024 р.

3. Вихідні дані до роботи науково-методична та науково-технічна література, матеріали конференцій, дані інтернет-мережі, редактор початкового коду і зневаджувач VS Code, векторний графічний редактор Figma.

4. Перелік питань, що потрібно опрацювати в роботі
1. Аналіз предметної області.
2. Аналіз завдання, технічних засобів.
3. Розробка дизайну та програмна реалізація.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) Актуальність проблеми безпритульних тварин, постановка задачі, схематичні зображення, графіки.

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	08.04.2024	
2	Аналіз завдання, підбір літератури	08.04.24-15.04.24	
3	Аналіз літератури з досліджуваної проблеми	16.04.24-18.04.24	
4	Аналіз технічних засобів	19.04.24-25.04.24	
5	Розробка дизайну	26.04.24-14.05.24	
6	Програмна реалізація	15.05.24-23.05.24	
7	Оформлення пояснювальної записки	24.05.24-26.05.24	
8	Перевірка на плагіат	27.05.24	
9	Рецензування	28.05.24	
10	Підготовка презентації та доповіді	29.05.24-05.06.24	
11	Занесення роботи в електронний архів	05.06.24	
12	Попередній захист кваліфікаційної роботи	05.06.24	

Дата видачі завдання 8 квітня 2024 р.

Студент _____
(підпис)

Керівник роботи _____ доц. Кобилін О.А.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ/ABSTRACT

Пояснювальна записка до кваліфікаційної роботи: 73 с., 22 рис., 32 джерела.

МІКРОСЕРВІСНА АРХІСТЕКТУРА, МЕТОДИ КЛАСТЕРИЗАЦІЇ, МЕТОД *K*-СЕРЕДНІХ, REACT.JS, REACT NATIVE, NODE.JS, MONGODB, FIGMA, КРОСПЛАТФОРМЕНІСТЬ.

Об’єктом роботи є процес надання допомоги безпритульним тваринам та всі аспекти, пов’язані з цим процесом, включаючи пошук, реєстрацію, медичний догляд, соціалізацію, адопцію та інші аспекти.

Мета цієї роботи є розробка та впровадження кросплатформеного застосунку для надання допомоги безпритульним тваринам. Розробка цієї програми спрямована на покращення координації та ефективності процесу допомоги безпритульним тваринам.

Під час розробки застосунку були використані сучасні технології та фреймворки, такі як React.js, Node.js, MongoDB та інші. Досліджено метод кластеризації *k*-середніх, який дозволяє ефективно сегментувати інформацію про тварин, що потребують допомоги. Розроблено мікросервісну архітектуру, яка забезпечує високу масштабованість та ефективність системи.

У результаті роботи здійснена програмна реалізація кросплатформеного застосунку щодо допомоги тваринам.

MICROSERVICE ARCHITECTURE, CLUSTERING METHODS, *K*-MEANS METHOD, REACT.JS, REACT NATIVE, NODE.JS, MONGODB, FIGMA, CROSS-PLATFORMITY.

The object of the work is the process of providing assistance to homeless animals and all aspects related to this process, including search, registration, medical care, socialization, adoption and other aspects.

The purpose of this work is to develop and implement a cross-platform application for providing assistance to homeless animals. The development of this program is aimed at improving the coordination and efficiency of the process of helping homeless animals.

During the development of the application, modern technologies and frameworks such as React.js, Node.js, MongoDB and others were used. The *k*-means clustering method was studied, which allows for efficient segmentation of information about animals in need of help. A microservice architecture has been developed, which ensures high scalability and efficiency of the system.

As a result of the work, a software implementation of a cross-platform application for helping animals was carried out.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

SQL – Structured Query Language (мова структурованих запитів)

API – Application Programming Interface (прикладний програмний інтерфейс)

CRUD – Create, Read, Update, Delete (створення, читання, оновлення, видалення)

HTTP – HyperText Transfer Protocol (протокол передачі гіпертекстових документів)

RPC – Remote Procedure Call (виклик віддалених процедур)

JS – JavaScript

HTML – HyperText Markup Language (мова розмітки гіпертексту)

CSS – Cascading Style Sheets (каскадні таблиці стилів)

UI – User Interface (інтерфейс користувача)

JSON – JavaScript Object Notation (запис об’єктів JavaScript)

JSX – JavaScript XML

БД – база даних

DOM – Document Object Model (об’єктна модель документа)

ВСТУП

Проблема безпритульних тварин є однією з найбільш нагальних соціальних викликів нашого часу. Щодня безліч тварин залишаються без догляду та захисту, стикаючись з жорстокими реаліями вуличного життя, що несе за собою серйозні ризики для їхнього здоров'я і благополуччя. Бездомність тварин веде до численних проблем, включно з переповненістю притулків, розповсюдженням захворювань, а також впливає на екологічну та санітарну безпеку урбанізованих територій. Втім, впровадження сучасних технологій може значно покращити ситуацію, пропонуючи нові способи вирішення цих проблем.

Ця робота орієнтована на створення кросплатформеного застосунку, який має на меті допомогу безпритульним тваринам. Застосунок стане незамінним інструментом у координації дій волонтерів, благодійних організацій та небайдужих громадян. Його основна задача – спростити процес надання допомоги, оптимізувати збір та розподіл ресурсів, а також забезпечити ефективне спілкування між усіма учасниками.

Значний акцент у дослідженні робиться на аналіз існуючих проблем у сфері допомоги безпритульним тваринам. Вивчення інформації про поточні умови життя цих тварин, статистичні дані про кількість бездомних тварин у різних регіонах. Ця робота включає комплексний аналіз існуючого стану проблеми бездомних тварин, виявлення основних викликів та потреб, які існують у цій галузі.

Розроблюваний застосунок має вирішити кілька основних завдань. Однією з найважливіших, яку покликаний вирішити розроблюваний застосунок, є ліквідація дефіциту в інструментах для ефективної взаємодії та співпраці між усіма учасниками процесу допомоги.

По-перше, це забезпечення можливості швидкого звернення за допомогою для тварин, які знаходяться в небезпеці. По-друге, створення цифрової платформи для ведення обліку та управління ресурсами, що значно

поліпшить логістику допомоги та підвищить її ефективність. Також передбачається інтеграція з соціальними мережами для кращого залучення громадськості та підвищення обізнаності про проблему бездомних тварин.

Окрім технічної сторони проєкту, буде важливим також розробка зручного та інтуїтивно зрозумілого інтерфейсу, що дозволить користувачам легко навігувати застосунком та ефективно використовувати його можливості незалежно від технічних навичок. Врахування побажань та потреб кінцевих користувачів забезпечить широке прийняття та активне використання застосунку у різних регіонах.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Актуальність проблеми

У сучасному світі проблема безпритульних тварин є однією з найбільш актуальних та соціально-значимих проблем [1]. Щодня тисячі бездомних тварин опиняються в складних життєвих обставинах, потребуючи негайної допомоги та підтримки від громадян та спеціалізованих організацій. Змінити цю ситуацію і покращити життя безпритульних тварин можна шляхом впровадження ефективних інструментів та технологій, які допомагали б координувати та оптимізувати процес надання допомоги.

Саме тому робота спрямована на розробку кросплатформеного застосунку, призначеного для допомоги безпритульним. Метою цього застосунку є створення зручного та ефективного інструменту, який би сприяв покращенню координації зусиль волонтерів, організацій та громадян, що зацікавлені у наданні допомоги безпритульним тваринам.

У рамках роботи буде проведено комплексне дослідження сучасного стану проблеми безпритульних тварин, визначені ключові проблемні аспекти та потреби у цій галузі. Дефіцит ефективних інструментів для координації дій та співпраці між різними учасниками процесу допомоги є однією з найбільш актуальних проблем, яку має вирішити розроблений застосунок.

1.2 Вплив на суспільство

Покращення ситуації з безпритульними тваринами, як показує досвід та наукові дослідження, має величезний позитивний вплив на суспільство в цілому.

По-перше, зменшення кількості тварин, які потрапляють на вулиці, сприяє стабілізації популяцій та зниженню негативних наслідків

перенаселеності. Переповненість притулків та безпритульні тварини на вулицях можуть створювати різні проблеми, включаючи агресію, вироблення токсичних звичок та загрозу для громадської безпеки. Зменшення цього явища сприятиме покращенню загального благополуччя в містах та селищах.

По-друге, зниження ризику поширення хвороб є іншим важливим аспектом. Безпритульні тварини, які живуть на вулицях, часто зазнають стресу та недоліків у догляді, що може підвищити їхню вразливість перед хворобами та інфекціями. Розробка кросплатформенного застосунку, який сприяє адопції та соціалізації тварин, допоможе зменшити ризик поширення захворювань серед тварин та людей, що живуть у спільноті.

Крім того, покращення ситуації з безпритульними тваринами також може позитивно вплинути на екологічну ситуацію у містах. Безпритульні тварини часто стають джерелом забруднення та руйнування довкілля через несанітарні умови, які вони створюють. Зменшення кількості безпритульних тварин сприятиме зниженню забруднення та покращенню якості довкілля, що вигідно вплине на здоров’я людей та всього екосистеми.

Від того, що велика кількість безпритульних тварин перебуває на вулицях, страждають як тварини, так і люди. Це також впливає на імідж населених пунктів, на комфортність проживання у них, на витрати місцевих бюджетів тощо. Усього в світі нараховується близько 600 млн. безпритульних тварин. Наприклад, у Великобританії за окремими даними нараховується понад 9 млн. безпритульних котів та, станом на 2017 рік, – понад 66 тисяч безпритульних собак. У США щодня народжується близько 70 тисяч безпритульних тварин, а їхня загальна кількість складає понад 70 мільйонів. Але кількість безпритульних тварин, які стали такими спадково є зазвичай меншою, аніж в Україні (рис. 1.1). Це пов’язано з тим, що для врегулювання їхньої кількості застосовують такі гуманні методи як стерилізація, прилаштування безпритульних тварин тощо [1].

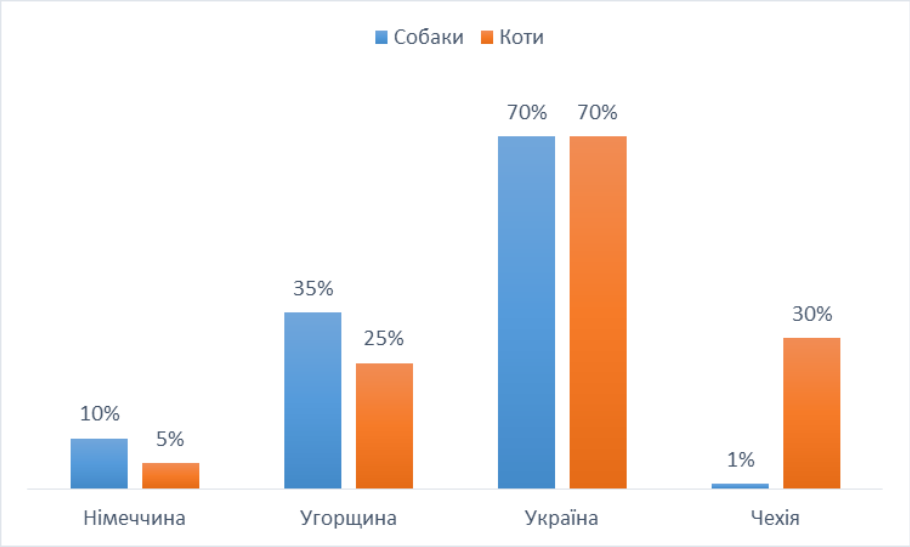

Рисунок 1. 1 – Спадково безпритульні собаки та коти, обсяг від загальної кількості безпритульних собак та котів

Нарешті, розвиток соціальної відповідальності серед громадян є ключовим аспектом у вирішенні проблеми безпритульних тварин. Розробка кросплатформенного застосунку, який сприяє взаємодії та допомозі тваринам, може спонукати людей до активної участі в допомозі та адопції безпритульних тварин. Це виховує почуття відповідальності та допомагає побудувати більш згуртовану та свідому громаду, яка турбується про добробут усіх її членів, включаючи тварин.

1.3 Роль кросплатформеного застосунку у вирішенні проблеми

Проблема безпритульних тварин є нагальною та потребує комплексного підходу для її вирішення. Розробка кросплатформеного застосунку спрямована на створення зручного та ефективного інструменту для координації допомоги та оптимізації зусиль волонтерів, організацій та громадян у цьому напрямку.

Основна мета застосунку полягає в створенні єдиної платформи, яка об’єднує в собі інформацію про безпритульних тварин, послуги притулків,

поточні потреби та можливості волонтерської допомоги. Він надає можливість швидкого та зручного доступу до інформації про тварин, які потребують допомоги, а також координації дій для їхнього врятування та адоптації.

Застосунок дозволяє переглядати тварин, які потребують допомоги, інформацію про їхні поточні потреби, медичний стан та інші важливі дані. Він також надає можливість організаціям та волонтерам пропонувати свою допомогу, реагуючи на актуальні потреби і розподіляючи завдання для оптимальної координації зусиль.

Один з ключових аспектів розробки застосунку – це створення бази даних з фотографіями та описами безпритульних тварин, що допоможе залучити увагу потенційних усиновлювачів та забезпечити їм можливість вибору та контакту з притулками.

Крім того, застосунок може включати функції спільноти, де користувачі можуть обмінюватися досвідом, порадами та історіями успіху у догляді та усиновленні тварин. Це сприяє формуванню підтримки та взаємодопомоги серед учасників спільноти.

У підсумку, розробка кросплатформеного застосунку для допомоги тваринам є кроком у напрямку полегшення та оптимізації процесів рятування та усиновлення безпритульних тварин, а також створення спільноти, яка об’єднує людей зі схожими цілями та цінностями.

1.4 Інтеграція функціональності та практичних рішень

Ефективне використання кросплатформеного застосунку для допомоги безпритульним тваринам залежить від інтеграції різноманітних функцій та можливостей, що спрямовані на полегшення процесу допомоги і підтримки цих тварин. Розробка такого застосунку спрямована на створення зручного та доступного інструменту для координації зусиль волонтерів, організацій та громадян у вирішенні проблеми безпритульних тварин.

Підхід до розробки застосунку передбачає поєднання теоретичних аспектів, які охоплюють питання етики, медицини та соціальної роботи у сфері тваринництва, з практичними функціями, спрямованими на полегшення взаємодії між користувачами, збір необхідної інформації та надання допомоги безпритульним тваринам.

Застосунок має створювати можливості для плавного переходу від базових функцій, таких як пошук та збір інформації про безпритульних тварин та волонтерські можливості, до більш складних завдань, наприклад, координація медичної допомоги, пошук притулків для тварин та організація процесу адопції.

Збір та аналіз даних про ефективність використання застосунку є важливою складовою розробки. Ця інформація дозволить оптимізувати функціонал застосунку, а також адаптувати його до змінних потреб користувачів та сучасних викликів у сфері захисту тварин.

Розробка кросплатформеного застосунку для допомоги безпритульним тваринам відображає необхідність комплексного підходу та взаємодії між різними сторонами, включаючи волонтерів, організації, місцеві влади та громадськість, для досягнення спільної мети – покращення умов життя безпритульних тварин.

На даний момент, у багатьох країнах, у тому числі і в Україні, проблема безпритульних тварин є складною і тривожною. Наприклад, за даними організації "Четвероногий Ангел", кількість безпритульних собак у нашій країні оцінюється десятками тисяч особин. В Україні не ведеться загальнонаціональний офіційний облік безпритульних тварин, через що їхня точна кількість залишається невідомою. Водночас, окремі міста здійснюють такий підрахунок для того, щоб в подальшому планувати видатки з бюджету та приймати рішення про регулювання чисельності безпритульних тварин. Так, у шести обласних містах згідно офіційних даних налічується від 24 до понад 26 тисяч безпритульних тварин (рис. 1.2) [1].

Рисунок 1.2 – Дані про кількість безпритульних тварин по містах України, 2016-2017 р.

Також, варто зазначити, що існуючі рішення в цій галузі мають свої обмеження та недоліки. Наприклад, багато застосунків та платформ спеціалізуються на конкретних аспектах проблеми (наприклад, пошук власників для безпритульних тварин), тоді як широкої координації та взаємодії між усіма учасниками процесу не вистачає.

Також, існує проблема із залученням нових волонтерів та осіб, які б хотіли допомогти безпритульним тваринам, але не знають, як саме їм це зробити. Для багатьох людей бракує зручного та доступного інструменту, який би дозволив їм знайти способи допомоги, що відповідали б їхнім можливостям та інтересам.

Отже, розробка кросплатформеного застосунку для надання допомоги безпритульним тваринам має велике соціальне значення. Він може стати зручним та ефективним інструментом для координації зусиль усіх учасників процесу та сприяти покращенню умов і якості життя безпритульних тварин. Такий застосунок може стати своєрідним мостом між тими, хто хоче

допомогти, і тими, хто потребує допомоги, та сприяти більш ефективній роботі притулків та організацій захисту тварин.

1.5 Необхідність зручного та універсального застосунку

Особлива увага буде приділена процесу проєктування архітектури та інтерфейсу користувача розроблюваного застосунку. Важливо створити зручне та інтуїтивно зрозуміле середовище для користувачів, що дозволить ефективно взаємодіяти з програмою та надавати необхідну допомогу безпритульним тваринам.

Буде проведено реалізацію програмного забезпечення розробленого застосунку, його тестування та вдосконалення з метою забезпечення високої якості та надійності продукту.

Розробка кросплатформенного застосунку для допомоги безпритульним тваринам не лише сприятиме полегшенню доступу до ресурсів та послуг для користувачів мобільних пристроїв, але й забезпечить універсальність використання для тих, хто віддає перевагу роботі на комп’ютерах. Користувачі зможуть безперервно користуватися застосунком, переходячи з комп’ютера на мобільний пристрій і навпаки, зберігаючи при цьому доступ до всіх функцій та даних. Це відкриває нові можливості для оптимізації робочого процесу та сприяє покращенню ефективності користувачів незалежно від їхнього вибору пристрою. Такий підхід до розробки застосунків допомоги безпритульним тваринам відповідає сучасним вимогам мобільності та гнучкості, роблячи допомогу тваринам доступною для широкого кола користувачів.

Кросплатформеність застосунку відкриває можливість його використання на різних операційних системах, таких як Windows, macOS, Linux, iOS та Android, без необхідності розробляти окремий код для кожної з цих платформ. Це означає, що один і той же код може бути використаний для

розгортання застосунку на будь-якій з підтримуваних платформ, що забезпечує значну економію часу та зусиль розробників.

Існують різні підходи до досягнення кросплатформеності, і одним з них є використання гібридних підходів, таких як розробка вебзастосунків. У цьому випадку застосунок буде написаний з використанням вебтехнологій, таких як HTML, CSS та JavaScript, що є стандартними для веброзробки, та він буде запускатися в спеціальному контейнері на кожній платформі. Це дозволить йому працювати на різних пристроях, таких як комп'ютери, планшети та смартфони, незалежно від операційної системи. Ключові аспекти цього підходу включають:

– використання вебтехнологій: розробники можуть використовувати відомі вебтехнології, такі як HTML, CSS та JavaScript, для створення інтерфейсу та логіки застосунку. Ці технології широко підтримуються на різних платформах;

– контейнеризація застосунку: застосунок запускається в спеціальному контейнері на кожній платформі, що забезпечує йому середовище для виконання. Це дозволяє йому працювати на різних пристроях без змін вихідного коду;

– кросплатформеність через веб: за допомогою цього підходу застосунок може бути доступним на різних операційних системах, таких як Windows, macOS, Linux, iOS та Android. Можна використовувати єдину базу коду для різних платформ;

– широкі можливості кастомізації: можна використовувати різні фреймворки та бібліотеки для розширення функціональності та кастомізації вебзастосунку. Це дозволяє створювати багатофункціональні та естетично привабливі застосунки для різних пристроях.

Ще одним підходом є використання кросплатформених мов програмування та фреймворків. Наприклад, Kotlin Multiplatform дозволяє розробникам писати код на Kotlin, який потім може бути компільований для різних платформ, включаючи Android, iOS та веб. Xamarin використовує мову

програмування C# та дозволяє розробляти кросплатформені застосунки для Android, iOS та Windows. Flutter, розроблений Google, дозволяє створювати кросплатформені мобільні застосунки, використовуючи один і той же код для Android та iOS, а також навіть для вебзастосунків.

Одним з важливих аспектів розробки кросплатформеного застосунку є розробка користувацького інтерфейсу, який може адаптуватися до різних розмірів екрану та роздільної здатності. Це забезпечує комфортне користування застосунком на будь-яких пристроях, незалежно від їхніх технічних характеристик. Розробка такого користувацького інтерфейсу є критично важливою для забезпечення комфортного користування кросплатформеним застосунком на різних пристроях:

– респонсивний дизайн – це підхід до розробки вебсайтів та вебзастосунків, коли макет автоматично змінюється відповідно до розміру екрану користувача. Наприклад, коли вебсторінка відображається на мобільному пристрої, елементи можуть розташовуватися вертикально один під одним, а на більшому екрані – горизонтально, з більшим розміщенням елементів на екрані;

– адаптивний дизайн: цей підхід дозволяє створювати різні макети для різних пристроїв, використовуючи один і той же код. Наприклад, на планшетах та десктопах можуть бути відображені більші панелі із великими кнопками, тоді як на мобільних пристроях ці елементи можуть бути скомпактовані для ефективного використання екранного простору;

– адаптація до роздільної здатності: різні пристрої можуть мати різну роздільну здатність екрану. Розробка інтерфейсу повинна враховувати цей аспект, щоб текст та зображення залишалися чіткими та читабельними незалежно від розміру екрану та його роздільної здатності;

– універсальний дизайн елементів керування: кнопки, поля введення, меню та інші елементи керування повинні бути розроблені таким чином, щоб вони були зручними для використання як на десктопах, так і на сенсорних пристроях. Наприклад, кнопки повинні мати достатньо великий розмір для

зручного натискання на сенсорних екранах, але вони також повинні бути зручними для використання за допомогою миші чи трекпаду.

1.6 Комплексний підхід для успішного впровадження програмного продукту

У контексті розробки кросплатформеного застосунку для допомоги безпритульним тваринам, важливо враховувати також потенційні перешкоди та виклики, з якими можуть зіткнутися розробники та користувачі в процесі впровадження та використання програмного продукту.

Однією з таких перешкод є фінансові обмеження. Розробка та підтримка кросплатформеного застосунку може вимагати значних витрат коштів на програмні ресурси, інфраструктуру, рекламу та інші витрати. Організації, що допомагають безпритульним тваринам, можуть мати обмежений бюджет, що ускладнює фінансування подібних проєктів.

Також слід враховувати технічні та юридичні аспекти. Наприклад, необхідно враховувати вимоги до захисту даних та конфіденційності інформації про користувачів та тварин, а також відповідати законодавству у сфері захисту персональних даних. Технічні обмеження та труднощі можуть виникнути при розробці кросплатформеного застосунку, особливо коли йдеться про забезпечення його сумісності з різними операційними системами та пристроями.

Додатково, важливо враховувати потреби та очікування різних категорій користувачів. Застосунок повинен бути зрозумілим та доступним як для досвідчених волонтерів та активістів, так і для новачків, які тільки починають займатися допомогою безпритульним тваринам. Також слід враховувати потреби людей з обмеженими можливостями та різних вікових груп.

Урахування цих аспектів допоможе створити більш повноцінний та ефективний кросплатформений застосунок для надання допомоги

безпритульним тваринам, який зможе успішно впровадитися та знайти своє місце у соціальному середовищі.

1.7 Розробка технологічного рішення для управління популяціями безпритульних тварин з застосуванням методу *k*-середніх

Для вирішення складнощів, пов’язаних з доглядом за безпритульними тваринами, потрібно розробити технологічне рішення, яке враховує унікальні характеристики кожної тварини та забезпечує адаптивність до змінних умов і обмежень даних.

Для вирішення цієї проблеми ми спочатку використовували аналіз часових рядів, зокрема час перебування тварин у притулку [2]. Однак, в ситуації з роботою з нестаціонарними часовими рядами, такими як тривалість перебування тварин в притулку, класичні методи аналізу часових рядів можуть бути непридатними. Тому ми звернулися до методу кластеризації, а саме алгоритму *k*-means, який є неієрархічним методом кластеризації [3-11].

Мета застосування кластеризації полягає в групуванні тварин залежно від тривалості їх перебування у притулку, щоб виявити патерни поведінки та особливості догляду за ними. Для цього спочатку визначаються часові інтервали, які включають тимчасове перебування (до 1 місяця), короткострокове перебування (1-6 місяців), середньострокове перебування (6-12 місяців), тривале перебування (12-24 місяців) та довгострокове перебування (понад 24 місяців). Потім можна використовувати алгоритм кластеризації, такий як *k*-середніх, щоб розділити тварин на групи на основі часу перебування в притулку [5]. Результати цієї кластеризації допоможуть ефективно управляти популяціями бездомних тварин і розробляти персоналізовані підходи до їхнього догляду та регулювання.

Алгоритм *k*-середніх виконується наступним чином:

Крок 1. Ініціалізація: визначається кількість кластерів (k), та випадковим чином обираються початкові центри кластерів.

Крок 2. Приналежність до кластера: кожен об’єкт призначається до кластера, центр якого є найближчим до нього.

Крок 3. Перерахунок центрів: для кожного кластера обчислюється новий центр як середнє арифметичне всіх об’єктів, які до нього належать.

Крок 4. Крок 2 і Крок 3 повторюються до досягнення збіжності, коли приналежність об’єктів до кластерів не змінюється або зміни відбуваються дуже малі.

Результати кластеризації дозволять виявити, наприклад, які категорії тварин більш схильні до тривалого перебування в притулку, що може вказувати на потребу особливої турботи та уваги. Також це допоможе керувати ресурсами притулку більш ефективно, передбачати завантаженість та вживати заходів для зменшення часу перебування тварин у притулку, наприклад, шляхом активізації програм з пошуку нових будинків та збільшення зусиль щодо залучення потенційних власників.

1.8 Постановка задачі

Таким чином, допомога безпритульним тваринам є актуальним завданням сьогодення. Тому ставиться завдання розробки, тестування та впровадження нового застосунку з метою покращення якості та ефективності допомоги тваринам.

Об’єктом роботи є процес надання допомоги безпритульним тваринам та всі аспекти, пов’язані з цим процесом, включаючи пошук, реєстрацію, медичний догляд, соціалізацію, адопцію та інші аспекти.

Метою цієї роботи є розробка та впровадження кросплатформеного застосунку для надання допомоги безпритульним тваринам. Розробка цієї програми спрямована на покращення координації та ефективності процесу

допомоги безпритульним тваринам, забезпечення зручного та доступного інструменту для волонтерів, організацій та осіб, що бажають надати допомогу цим тваринам.

Для досягнення мети необхідно вирішити такі завдання:

– проведення аналізу сучасного стану проблеми безпритульних тварин, включаючи обсяг проблеми, основні причини бездомності тварин та існуючі підходи до розв’язання цієї проблеми;

– огляд і аналіз існуючих платформ та застосунків, призначених для допомоги безпритульним тваринам, з визначенням їхніх переваг, недоліків та можливостей для вдосконалення;

– визначення потреб та вимог користувачів (волонтерів, організацій, потенційних власників тощо) щодо функціональності та інтерфейсу кросплатформеного застосунку;

– розробка архітектури та функціональності кросплатформеного застосунку з урахуванням визначених потреб та вимог користувачів;

– реалізація програмного забезпечення кросплатформеного застосунку з використанням сучасних методів та технологій розробки програмного забезпечення;

– проведення тестування розробленого застосунку з метою виявлення та виправлення помилок, а також оцінки його ефективності та користувальницької зручності;

– впровадження та підтримка кросплатформеного застосунку, включаючи розгортання на різних платформах;

– оцінка впливу та результативності розробленого застосунку на покращення ситуації з безпритульними тваринами, зокрема шляхом аналізу збільшення кількості адопцій, полегшення взаємодії між організаціями та волонтерами, зменшення кількості безпритульних тварин тощо.

2 ОБҐРУНТУВАННЯ ОБРАНИХ МЕТОДІВ

2.1 Мікросервісна архітектура

Мікросервісна архітектура – це спосіб побудови програмного забезпечення, де застосунок розбивається на невеликі, автономні сервіси, відомі як мікросервіси. Кожен мікросервіс відповідає за виконання певної обмеженої функціональності та має своє власне незалежне розгортання та масштабування (рис. 2.1) [12, 18].

Основна ідея полягає в тому, щоб розбити великий, складний застосунок на менші, більш керовані блоки, які легше розробляти, розгортати, масштабувати та підтримувати. Це протистоїть традиційним монолітним архітектурам, де весь застосунок представляє собою один великий блок коду, що може бути складним у розвитку та підтримці.

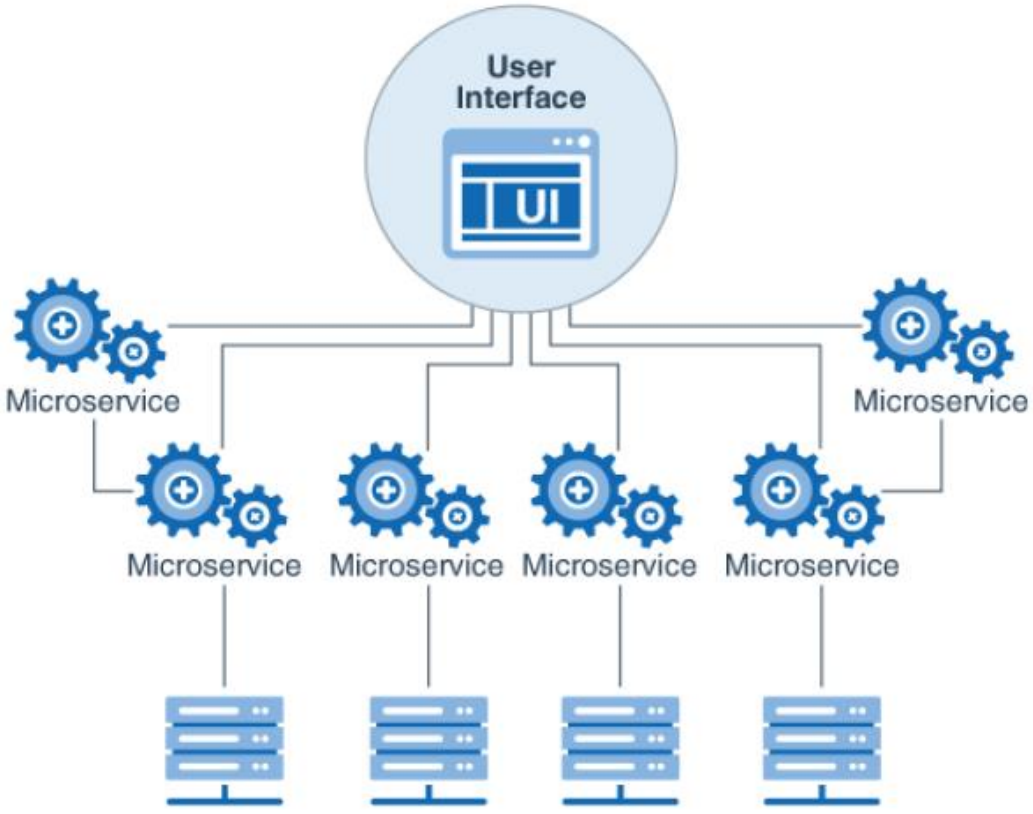

Рисунок 2.1 – Мікросервісна архітектура

Мікросервісна архітектура базується на ідеї розділення великого застосунку на менші, незалежні сервіси, кожен з яких відповідає за певну функціональність. Це розділення дозволяє кожному мікросервісу бути розгорнутим і масштабованим окремо від інших, забезпечуючи можливість розвитку та адаптації компонентів індивідуально, що сприяє гнучкості і швидкості внесення змін.

Мікросервіси взаємодіють один з одним через мережу, використовуючи різні методи комунікації, такі як HTTP, системи обміну повідомленнями або віддалені виклики процедур (RPC). Така взаємодія дозволяє сервісам ефективно спілкуватися, незважаючи на те, що кожен сервіс може мати власне середовище виконання і навіть свою базу даних.

Завдяки цій структурі, мікросервіси надають можливість масштабувати лише ті компоненти застосунку, які потребують цього, залежно від їх навантаження або інших вимог. Це забезпечує більш ефективне використання ресурсів і поліпшує загальну продуктивність застосунку.

Загалом, мікросервісна архітектура дозволяє створювати більш гнучкі, масштабовані та легко розвивати застосунки, що можуть краще відповідати змінюваним потребам бізнесу та користувачів. Однак вона також може вимагати більшої уваги до управління мережею та складнішої системи тестування та моніторингу.

2.2 Комунікація у вебзастосунках на основі React та Node.js

У вебзастосунках на основі React та Node.js мікропроцесорні комунікації можуть відбуватися через взаємодію клієнтської та серверної частин застосунку:

– HTTP/HTTPS протокол: клієнтська частина React може взаємодіяти з сервером, який працює на Node.js, за допомогою HTTP або HTTPS протоколів. Наприклад, React може виконувати запити до сервера для

отримання або відправлення даних, таких як дані форми або запити на отримання даних з бази даних;

– WebSocket комунікація: використання WebSocket дозволяє налагодити двонаправлену комунікацію між клієнтом і сервером. React може встановлювати підключення до сервера Node.js через WebSocket, щоб отримувати оновлення в реальному часі, такі як повідомлення чату або оновлення даних (рис. 2.2) [13];

– REST API: клієнтська частина React може використовувати REST API, яке надається сервером Node.js, для отримання або відправлення даних. Реакт може взаємодіяти з цим API, виконуючи HTTP запити, такі як GET, POST, PUT та DELETE, для роботи з ресурсами на сервері;

– GraphQL є альтернативою REST API, яка дозволяє клієнтам запитувати лише ті дані, які їм потрібні. Клієнт React може використовувати GraphQL запити до сервера Node.js для отримання точно визначених даних. Усі ці методи комунікації можуть бути використані для обміну даними між клієнтом React та серверною частиною, яка працює на Node.js, забезпечуючи ефективну та надійну взаємодію між ними.

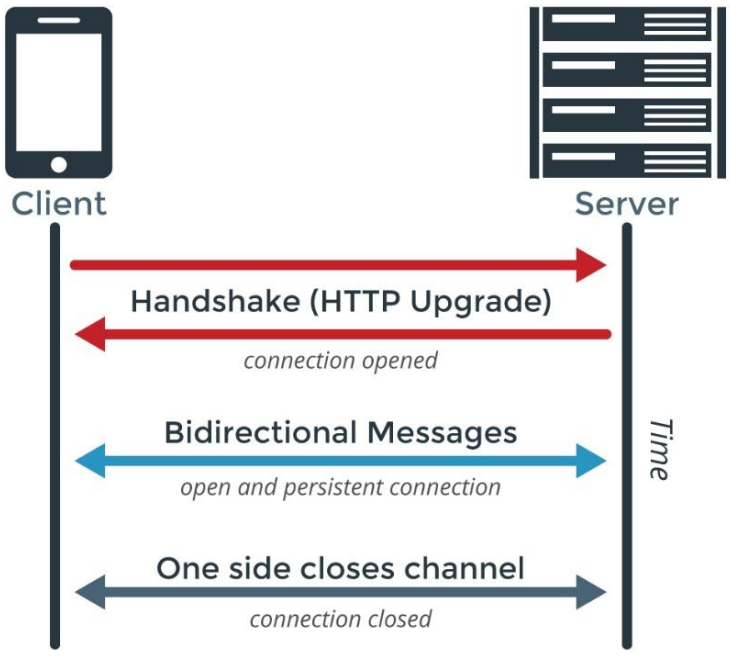

Рисунок 2.2 – WebSocket комунікація

2.3 REST API

REST (Representational State Transfer) – це стиль архітектури для розробки мережевих програмних систем, який розробив Рой Філдінг у своїй дисертації 2000 року. REST використовується для створення вебслужб, що надають доступ до даних та функціональності за допомогою стандартних протоколів HTTP [14].

Основні принципи REST:

– клієнт-серверна архітектура: REST розділяє клієнтів та сервери, що дозволяє їм розвиватися незалежно один від одного і зменшує зв’язаність між ними;

– безстатевість: кожний запит від клієнта містить всю необхідну інформацію для обробки цього запиту. Сервер не зберігає жодної інформації про стан клієнта між запитами;

– кешування: сервери можуть позначати відповіді як кешовані (cacheable) або некешовані. Клієнти можуть зберігати кешовані відповіді та використовувати їх пізніше, якщо це можливо;

– єдність інтерфейсу: уніфікований інтерфейс узгоджує спосіб взаємодії між клієнтом та сервером, що спрощує та робить систему більш зрозумілою;

– шари системи: система може бути розділена на рівні (layers), де кожен рівень може займатися своєю функціональністю, що сприяє масштабованості;

– код на запит: опційний принцип, який дозволяє серверу надсилати клієнту виконавчий код (наприклад, JavaScript) для розширення функціональності клієнта.

2.3.1 RESTful API

RESTful API – це вебслужба, яка використовує принципи REST для створення інтерфейсу, через який клієнти можуть взаємодіяти з сервером [15, 16].

Основні концепції RESTful API:

– у RESTful API дані представлені як ресурси, до яких можна звертатися через URL. Наприклад, ресурс «користувач» може бути доступний за адресою /users;

– управління ресурсами здійснюється за допомогою стандартних методів HTTP: GET (отримання), POST (створення), PUT (оновлення) та DELETE (видалення);

– ресурси мають представлення, які можуть бути відправлені та отримані клієнтом. Це може бути HTML, XML, JSON або будь-який інший формат даних;

– кожен ресурс має унікальний ідентифікатор URI, який дозволяє клієнту звертатися до нього;

– клієнт повинен надсилати всю необхідну інформацію для обробки запиту, а сервер не повинен зберігати жодної інформації про стан клієнта;

– гіпермедіа включає посилання на інші ресурси у відповіді сервера, що дозволяє клієнтам дізнатися про доступні дії та переходити до інших ресурсів.

RESTful API дозволяє створювати гнучкі, масштабовані та легко розширювані вебслужби, які можуть ефективно взаємодіяти з клієнтами через Інтернет. Він є стандартом для багатьох вебзастосунків і дозволяє створювати платформо-незалежні системи з великою кількістю клієнтів.

2.3.2 Стандартні методи HTTP

Управління ресурсами у RESTful API здійснюється за допомогою стандартних методів HTTP, які визначають тип операції, яку потрібно виконати з ресурсом:

– GET (отримання): цей метод використовується для отримання даних з ресурсу. Коли клієнт відправляє GET-запит на сервер, він отримує інформацію, що зберігається на сервері у вигляді відповіді. GET-запит не має побічних ефектів, тобто він не змінює стану сервера або ресурсу;

– POST (створення): цей метод використовується для створення нового ресурсу на сервері. Коли клієнт відправляє POST-запит на сервер, він передає дані, з яких створюється новий ресурс. POST-запит може мати побічний ефект у вигляді зміни стану сервера або створення нового ресурсу;

– PUT (оновлення): цей метод використовується для оновлення існуючого ресурсу на сервері. Коли клієнт відправляє PUT-запит на сервер, він передає дані, які повинні замінити вміст існуючого ресурсу. PUT-запит повинен бути ідемпотентним, що означає, що він може бути викликаний багато разів без зміни результату;

– DELETE (видалення): цей метод використовується для видалення існуючого ресурсу з сервера. Коли клієнт відправляє DELETE-запит на сервер, він передає ідентифікатор ресурсу, який повинен бути видалений. DELETE-запит також повинен бути ідемпотентним.

2.4 GraphQL

GraphQL – це мова запитів для API, розроблена компанією Facebook у 2012 році та випущена як відкритий стандарт у 2015 році. Вона дає можливість клієнтам запитувати лише ті дані, які їм потрібні, і нічого зайвого, що робить їхні запити більш ефективними та динамічними. Основна відмінність GraphQL

від REST полягає у тому, що ви не запитуєте окремі ресурси; замість цього, ви описуєте великі і складні запити до сервера, які виходять за межі можливостей REST (рис. 2.3) [15, 16].

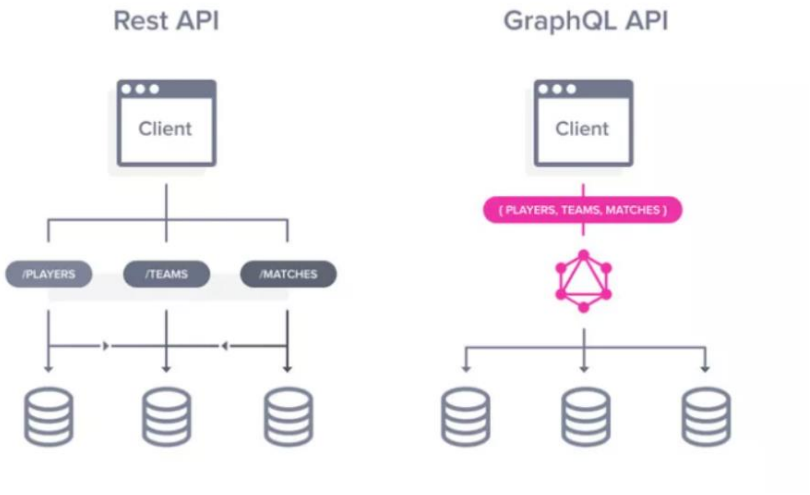

Рисунок 2.3 – Різниця між GraphQL та REST

В основі GraphQL лежить графова модель даних, яка дозволяє клієнтам отримувати дані з сервера, звертаючись до графа об’єктів та їх взаємозв’язків. Запити GraphQL описують структуру даних, яку клієнт хоче отримати, і сервер повертає лише ті дані, які відповідають цьому опису. Це дозволяє ефективно передавати дані між клієнтом і сервером, зменшуючи кількість необхідних запитів та об’єм переданих даних.

GraphQL також надає можливість використання фрагментів для перевикористання та організації запитів, підтримує вкладеність запитів та підтримує вибіркове включення та виключення полів відповіді. Крім того, він забезпечує інтуїтивний і масштабований підхід до розробки API, оскільки дозволяє визначати структуру даних та типи запитів на стороні сервера.

GraphQL широко використовується в різних сферах, включаючи веброзробку, мобільний розвиток та розробку API для IoT-пристроїв. Він дозволяє розробникам створювати потужні та ефективні API, які відповідають потребам сучасних застосунків та систем.

2.5 Взаємодія між клієнтом і сервером

Взаємодія між клієнтом і сервером є критично важливою складовою будь-якого сучасного програмного застосунку, особливо в контексті вебзастосунків. Ця взаємодія включає передачу даних, обробку запитів, відображення відповідей та багато іншого.

Клієнт-серверна архітектура – це модель взаємодії між різними комп'ютерами, де один з них, відомий як сервер, надає послуги, а інші, відомі як клієнти, звертаються до сервера за отриманням цих послуг. У вебзастосунках, браузер клієнта виступає як клієнт, а сервер вебзастосунка виступає як сервер.

Цей процес можна уявити як діалог між клієнтом (браузером) і сервером. Спочатку клієнт ініціює запит до сервера, надсилаючи HTTP-запит, який містить тип методу (GET, POST, PUT, DELETE), URL і, можливо, дані для передачі.

2.5.1 Переваги взаємодії клієнта і сервера

Перш за все, розділення обов'язків полегшує керування та розвиток системи. Клієнтська частина забезпечує інтерфейс користувача, взаємодіючи з ним і відображаючи результати, тоді як серверна частина відповідає за обробку запитів, доступ до бази даних та виконання бізнес-логіки. Це дозволяє спеціалізуватися на конкретних завданнях та ефективно розподіляти ресурси [12].

Крім того, така архітектура надає можливість масштабувати серверну частину окремо від клієнтської. Це означає, що в разі збільшення навантаження на сервер, можна додавати або розгортати нові сервери без впливу на роботу клієнтів. Такий підхід спрощує роботу з великими обсягами даних та підвищує масштабованість системи.

Гнучкість цієї архітектури полягає в тому, що зміни в клієнтській частині (наприклад, розробка мобільних застосунків) не вимагають змін в серверній частині, і навпаки [19]. Це дає можливість швидко адаптуватися до змін в вимогах користувачів та технологічному середовищі.

Нарешті, використання цієї архітектури сприяє підвищенню безпеки. Оскільки сервер зберігає дані та виконує обробку, це забезпечує більшу захищеність чутливих даних, оскільки вони не витікають на клієнтську сторону, де їх може бути важко контролювати. Такий підхід допомагає уникнути потенційних проблем із безпекою та конфіденційністю даних.

Загалом, взаємодія між клієнтом і сервером є ключовою для функціонування вебзастосунків, і вона дозволяє користувачам отримувати доступ до різноманітних послуг та інформації через Інтернет.

2.5.2 Синхронна взаємодія

Синхронна взаємодія, яка базується на прямій взаємодії між клієнтом і сервером, визначається передачею запитів від клієнта до сервера і блокуванням виконання клієнтом подальших дій до отримання відповіді від сервера (рис. 2.4). Цей метод є простим та зрозумілим, оскільки він створює послідовність дій, що спрощує розуміння та реалізацію взаємодії між клієнтом і сервером [12, 17].

Спочатку, коли клієнт ініціює синхронний запит до сервера, він блокує виконання подальших дій до отримання відповіді. Це означає, що після відправлення запиту клієнт не може продовжити свою діяльність до моменту отримання відповіді від сервера.

Після цього сервер приймає запит та розпочинає процес обробки. Це може включати аналіз маршруту, обробку даних у запиті, перевірку дозволів і взаємодію з базою даних. Цей етап є ключовим, оскільки сервер повинен правильно розуміти запит і підготувати відповідь.

Після успішної обробки запиту сервер генерує відповідь, яка відправляється назад клієнту. Це може бути HTML-сторінка, JSON-об’єкт, зображення або будь-який інший тип даних, залежно від характеру запиту та потреб клієнта.

Отримавши відповідь, клієнт обробляє її. Для вебзастосунків це може означати відображення сторінки, виконання JavaScript-коду або подальшу взаємодію з користувачем. Наприклад, клієнт може відображати сторінку для відповіді або взаємодіяти з нею через JavaScript.

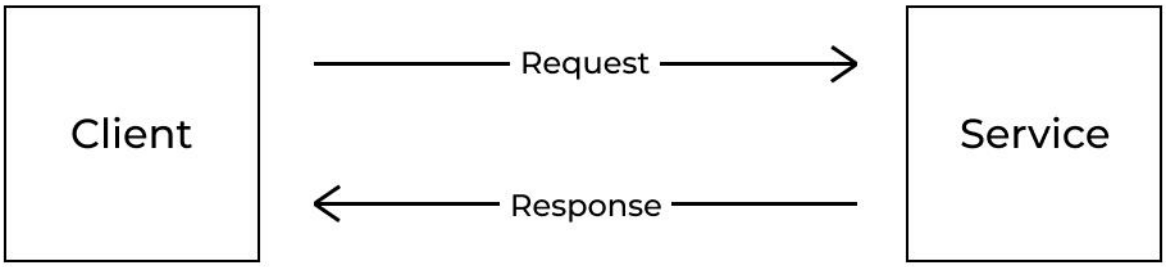

Рисунок 2.4 – Схема синхронної взаємодії

Однією з переваг синхронної взаємодії є його простота. Чітка послідовність виконання запитів дозволяє легко розуміти та відлагоджувати процес взаємодії між клієнтом і сервером. Крім того, він спрощує управління ресурсами та виконання операцій, оскільки передача запитів і отримання відповідей відбувається в чіткому порядку.

Проте синхронна взаємодія має деякі недоліки. Один з них – обмежена масштабованість через блокування ресурсів. У випадку, коли клієнт очікує відповіді від сервера, його ресурси залишаються неактивними, що може призвести до неефективного використання обчислювальних потужностей. Крім того, може виникнути проблема зі збоєм, якщо сервер не може швидко обробити всі запити через занадто велику кількість вхідних запитів.

Отже, хоча синхронна взаємодія є простою і зрозумілою, вона має обмежену масштабованість та може призводити до неефективного використання ресурсів в ситуаціях з великою кількістю запитів

Взаємодія може продовжуватися, наприклад, за допомогою асинхронних запитів (AJAX), вебсокетів для реального часу або обміну даними через API.

2.5.3 Асинхронна взаємодія

У підході асинхронної взаємодії між клієнтом і сервером клієнт не блокується під час очікування відповіді від сервера. Натомість, він може продовжувати виконувати інші дії або чекати відповіді в асинхронному режимі, отримуючи сповіщення про завершення операції. Цей метод відомий своєю здатністю забезпечувати вищу продуктивність та масштабованість, оскільки клієнт може продовжувати роботу під час очікування відповіді (рис. 2.5) [13].

Після ініціювання асинхронного запиту клієнт може продовжити свою діяльність, не очікуючи на відповідь від сервера. Це дозволяє клієнтові виконувати інші операції або обробляти інші запити, поки сервер обробляє початковий запит.

Після прийняття запиту сервер починає процес обробки, який може включати аналіз маршруту, обробку даних у запиті, перевірку дозволів та взаємодію з базою даних. Однак у цьому випадку сервер може приймати та обробляти інші запити незалежно від поточного асинхронного запиту.

Після успішної обробки запиту сервер генерує відповідь і відправляє її клієнту. Це може бути HTML-сторінка, JSON-об’єкт, зображення або будь-який інший тип даних. Однак у випадку асинхронної взаємодії, відправлення відповіді може відбутися пізніше, після того, як клієнт уже продовжив свою роботу.

Отримавши відповідь, клієнт може обробити її. Для вебзастосунків це може включати відображення сторінки, виконання JavaScript-коду або подальшу взаємодію з користувачем, проте ця обробка може відбутися в асинхронному режимі, не блокуючи інші дії клієнта.

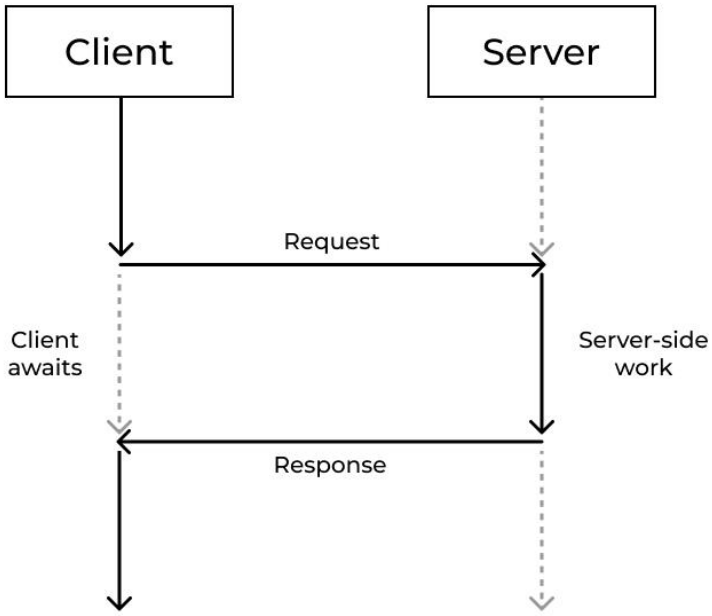

Рисунок 2.5 – Схема асинхронної комунікації

Однією з переваг асинхронної взаємодії є її здатність до покращення продуктивності. Клієнт може одночасно виконувати інші завдання або обробляти інші запити, не чекаючи завершення поточного запиту. Це дозволяє ефективніше використовувати обчислювальні ресурси та збільшує швидкість обробки запитів.

Крім того, асинхронний підхід сприяє покращенню масштабованості системи. Клієнти можуть надсилати запити без очікування відповідей, що дозволяє обробляти більшу кількість запитів одночасно. Це особливо важливо в сучасних динамічних системах з великим обсягом запитів.

Проте асинхронний підхід має деякі недоліки. Один з них – складність у реалізації та відлагодженні через його асинхронну природу. У порівнянні з

синхронним підходом, він вимагає більшої уваги до деталей, таких як управління потоками виконання та обробка помилок.

Крім того, можливі проблеми з управлінням станом даних та уникненням гонок. Оскільки запити можуть бути оброблені в довільному порядку, виникає ризик конфліктів при доступі до спільних ресурсів, що може призвести до некоректної поведінки системи.

Отже, асинхронний підхід відображає компроміс між покращенням продуктивності та складності у реалізації та управлінні системою.

2.6 Базові операції для управління даними CRUD

CRUD – це абревіатура, що означає чотири базові операції для управління даними: Create (створення), Read (читання), Update (оновлення) і Delete (видалення) [20]. Ці операції є фундаментальними для багатьох систем управління даними та застосунків, оскільки вони визначають способи взаємодії з даними (рис. 2.6).

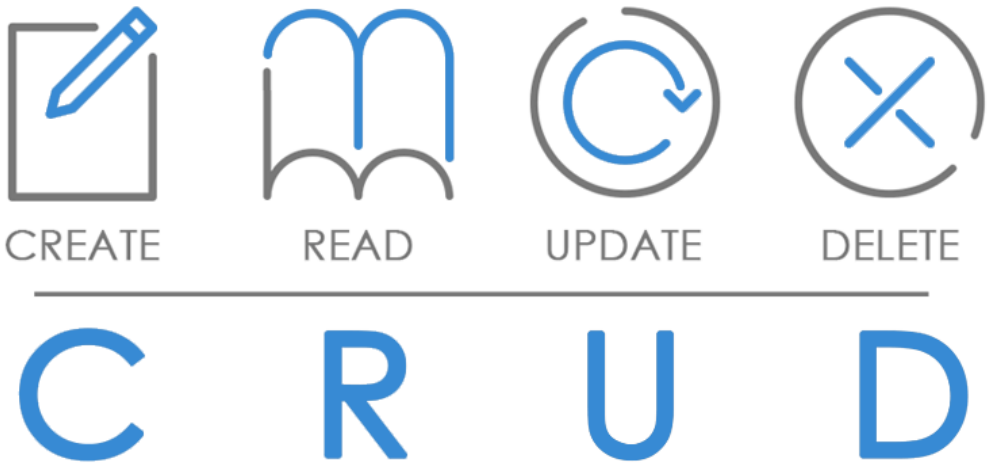

Рисунок 2.6 – Операції CRUD

2.6.1 Операція створення (create)

Операція створення використовується для додавання нових записів або об’єктів у базу даних або інше сховище даних. Цей процес може бути виконаний як користувачем, так і автоматично системою, зазвичай на основі попередньо встановлених правил або у відповідь на певні події або умови.

У контексті технологічного середовища, такого як веброзробка, операція створення може мати кілька складових елементів, які варто враховувати. Наприклад, використання форм для введення даних може стати вихідною точкою для операції створення. Користувач заповнює форму з необхідними даними, такими як ім’я, адреса електронної пошти тощо. Після відправки форми дані передаються на сервер для подальшої обробки.

На сервері, який може працювати на Node.js або іншому фреймворку, отримані дані проходять через процес валідації та обробки. Це включає перевірку правильності введених даних, зберігання їх у відповідному форматі та додавання до бази даних. Новий запис створюється у відповідній таблиці або колекції, ініціюючи тим самим операцію створення.

Операція створення є ключовим елементом в процесі управління даними, оскільки вона дозволяє системі динамічно збільшувати свій обсяг та розширювати функціональність шляхом додавання нової інформації.

2.6.2 Операція читання (read)

Операція читання є важливим процесом отримання інформації або даних з бази даних або іншого сховища даних без їх модифікації. Вона є фундаментальною для багатьох застосувань, оскільки дозволяє користувачам отримувати доступ до інформації, необхідної для подальшої обробки, аналізу або відображення.

У процесі операції читання зазвичай використовуються запити до бази даних за допомогою мови запитів, такої як SQL, або інші механізми отримання даних з сховища, такі як API. Ці запити можуть бути спрямовані на конкретні таблиці або колекції даних, іноді з використанням фільтрів, умов або сортування для отримання точних результатів.

У сучасному технологічному контексті, операція читання відіграє важливу роль у веброзробці, де інформація часто відображається на вебсторінках для користувачів. На вебрівні, операція читання зазвичай використовується для отримання даних з сервера та їх подальшого відображення на клієнтському пристрої.

Операція читання може бути використана для різних цілей. Наприклад:

– може включати відображення списків об’єктів або деталей конкретного об’єкта користувачам в інтерфейсі користувача вебзастосунка або мобільного застосунка;

– дозволяє аналізувати великі обсяги інформації для виявлення патернів, трендів або статистики, що може бути корисно для прийняття рішень у бізнесі або наукових дослідженнях;

– отримані дані можуть бути оброблені або перетворені перед подальшим використанням, наприклад, для використання у інших операціях чи взаємодії з іншими системами;

– може бути використана для надання доступу до даних через API для інших систем або сервісів, що дозволяє інтегрувати дані між різними застосунками.

Таким чином, операція читання в системах управління даними є ключовим елементом для забезпечення доступу до інформації та надає користувачам можливість отримати потрібні дані для подальшого використання або аналізу.

2.6.3 Операція оновлення (update)

Операція оновлення дозволяє змінювати існуючі дані в базі даних або іншому сховищі даних. Ця операція надає можливість користувачам або системі модифікувати інформацію відповідно до змінюваних потреб чи умов.

У контексті баз даних, операція оновлення зазвичай виконується за допомогою SQL (Structured Query Language) запитів, таких як UPDATE, що дозволяють змінювати значення конкретних полів у вже існуючих записах. Це може включати зміну значень певних полів, додавання нової інформації або видалення даних.

Операція оновлення відіграє важливу роль у великому спектрі сценаріїв та застосунків. Наприклад:

– користувачі можуть оновлювати свої дані, такі як паролі, адреси електронної пошти, інформацію про контакт тощо;

– адміністратори магазинів часто використовують операцію оновлення для зміни статусу товарів, їх кількості, ціни або інших атрибутів товарів;

– операція оновлення може бути використана для зміни статусу замовлень, вказання дати доставки, оновлення інформації про оплату тощо;

– в деяких випадках, дані можуть бути оновлені для вирішення питань аналізу даних або складання звітів.

В цілому, операція оновлення дозволяє системам управління даними підтримувати актуальність інформації та адаптуватися до змін в потребах користувачів чи умов середовища. Це є важливим елементом функціональності будь-якої системи, яка працює з даними.

2.6.4 Операція видалення (delete)

Операція видалення є важливою складовою управління даними в системах баз даних та інших сховищах даних. Ця операція передбачає

вилучення записів або даних з бази даних, що більше не використовуються або не є актуальними.

Ключовими аспектами операції видалення є забезпечення цілісності та консистентності даних. Видалення даних може бути проведене за різними причинами, такими як виправлення помилок, забезпечення дотримання правил конфіденційності або видалення застарілих або непотрібних даних для звільнення місця в сховищі.

При видаленні даних важливо враховувати можливі наслідки, такі як можливість втрати інформації та вплив на інші зв’язані дані. Також важливо враховувати права доступу та обмеження, щоб запобігти неправомірному видаленню даних.

Операція видалення може виконуватися як користувачем, так і адміністратором системи. Вона є важливою складовою процесу управління даними та підтримки ефективної роботи бази даних чи сховища даних.

CRUD – це основа для багатьох систем управління даними та застосунків, оскільки ці чотири операції надають користувачам зручний спосіб управління даними. Вони дозволяють створювати, читати, оновлювати та видаляти дані, що робить їх дуже потужними та універсальними в інформаційних системах.

2.7 Мікросервіс кластерного аналізу безпритульних тварин

Мікросервіс кластерного аналізу безпритульних тварин є складовою частиною системи, що забезпечує технологічне рішення для управління безпритульними тваринами. Основним завданням цього мікросервісу є аналіз та групування тварин залежно від тривалості їх перебування у притулку, що дозволяє розробляти стратегії догляду та управління, адаптовані до специфіки кожної групи.

Для аналізу часових рядів, особливо нестаціонарних, яким є тривалість перебування тварин у притулку, використовується алгоритм кластеризації *k*-means. Кластеризація спрямована на виявлення патернів та групування тварин зі схожою тривалістю перебування, що відображає їхню поведінку та потреби.

Основні функціональні можливості мікросервісу включають:

– мікросервіс проводить аналіз тривалості перебування тварин у притулку для виявлення залежностей та патернів у їх поведінці;

– застосовується алгоритм кластеризації *k*-means для групування тварин на основі тривалості перебування. Це дозволяє виділити різні групи тварин із схожими характеристиками;

– результати кластеризації використовуються для ефективного управління популяціями бездомних тварин, розробки персоналізованих підходів до догляду та регулювання.

Мікросервіс надає API для взаємодії з іншими компонентами системи, що дозволяє іншим сервісам використовувати дані про кластеризацію для прийняття рішень та планування дій з бездомними тваринами [29-31].

Технологічна структура мікросервісу може включати в себе застосування нейронних мереж та методів машинного навчання для поліпшення точності аналізу та прогнозування поведінки бездомних тварин.

На основі цього мікросервісу розроблено інноваційний підхід до скорочення часу перебування тварин у притулку. Шляхом використання алгоритму кластеризації *k*-means та нейронних мереж впроваджено систему, яка визначає та показує в списках тих тварин, які проводять у притулку найбільше часу. Це дозволяє стимулювати їхнє раніше прийняття та полегшує їхню адопцію, що в свою чергу сприяє скороченню часу перебування тварин у притулку та покращує їхні шанси на знаходження нового дому.

Усі ці функціональні аспекти спрямовані на покращення управління та догляду за безпритульними тваринами, що робить систему більш ефективною та спрощує процес прийняття стратегічних рішень у сфері тваринного добробуту.

3 РЕАЛІЗАЦІЯ ЗАСТОСУНКУ

3.1 Обґрунтування вибору середовища програмної реалізації

У рамках кваліфікаційної роботи був розроблений кросплатформений застосунок для допомоги безпритульним тваринам, використовуючи технології React.js, React Native, Node.js та MongoDB [21, 24-26]. Для реалізації було обране середовище Visual Studio Code (VS Code). Для розробки дизайну застосунку використовувалася Figma, яка забезпечує ефективне створення та управління інтерфейсами користувача.

3.1.1 Visual Studio Code

Visual Studio Code (рис. 3.1) надає розробникам потужні засоби для редагування коду, налагодження та тестування застосунків незалежно від обраної платформи. Важливою перевагою цього середовища розробки є його гнучкість і можливість налаштування, що дозволяє підлаштувати середовище під конкретні потреби проєкту. Інтеграція з Git забезпечує ефективне управління версіями коду, що є важливим аспектом у сучасній розробці програмного забезпечення.

VS Code бере на себе всю допоміжну роботу з управління проєктом, звільняючи від непродуктивного програмування та дозволяючи їм зосередитися на створенні функціоналу. Це не означає, що прямий обіг застосунків до інших інструментів або середовищ розробки забороняється – завжди при необхідності можна їх використовувати. Однак, інкапсуляція функціональних можливостей у зручному інтерфейсі та численні плагіни для різних мов програмування і технологій представляють більш перспективну методику розробки програмного забезпечення.

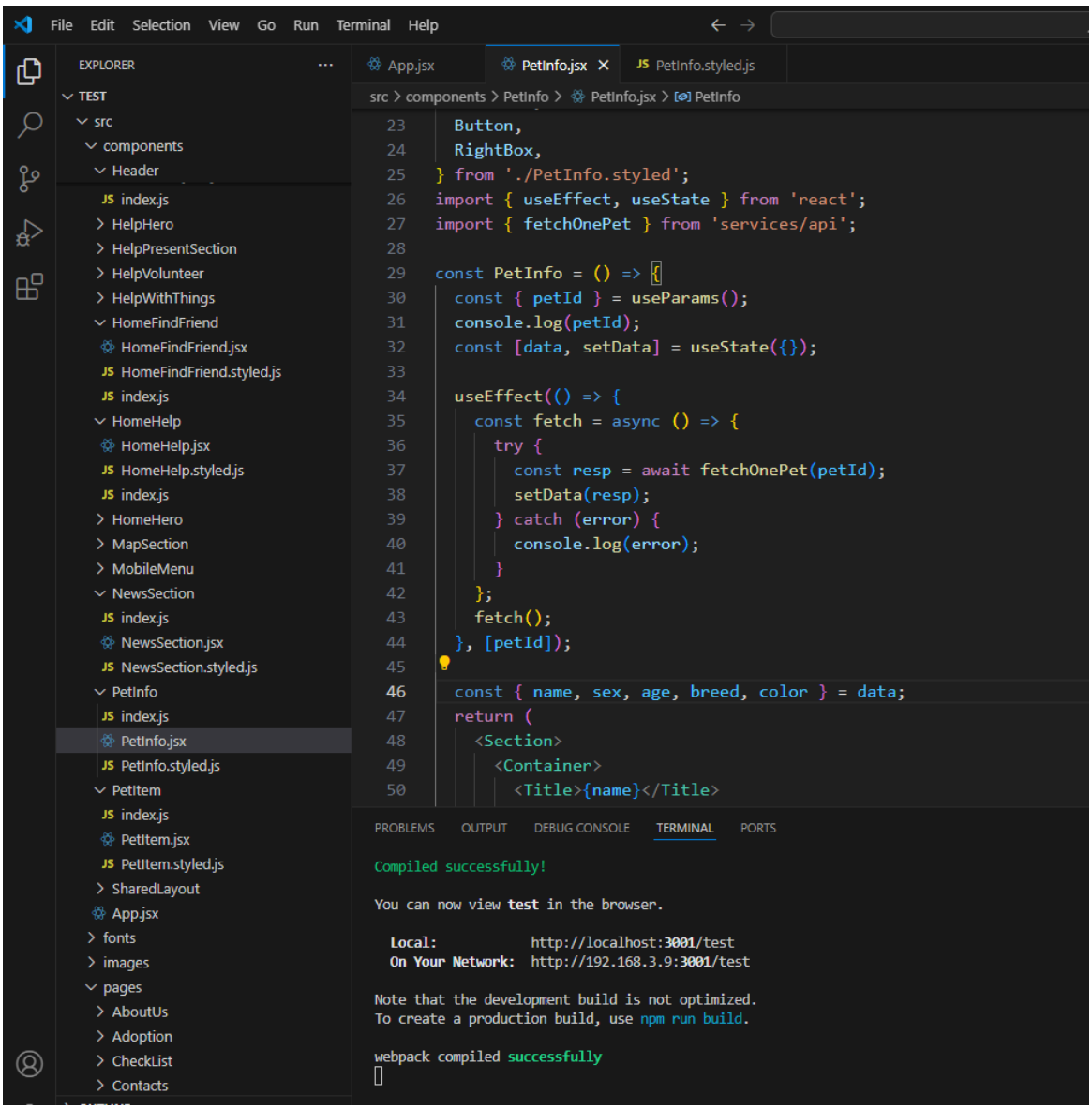

Рисунок 3.1 – Приклад інтерфейсу VS Code

Visual Studio Code інкапсулює різні інструменти та технології на різних рівнях. Найбільш важливим є спосіб, за допомогою якого редактор надає інтеграцію з різноманітними мовами програмування та фреймворками через розширення. При використанні цих розширень є доступ до таких функцій, як автодоповнення коду, рефакторинг, підсвічування синтаксису та інтегрована налагоджувальна консоль. Після завершення роботи над кодом, можна скористатися вбудованими інструментами для запуску та тестування застосунку, а також для його деплою на різні платформи.

Замість того щоб змушувати працювати з кодом на низькому рівні деталізації, Visual Studio Code надає простий і довершений інтерфейс через свою модульну архітектуру та розширення. Це дозволяє зосередитися на створенні якісного програмного забезпечення, використовуючи потужні інструменти у зручному та інтуїтивно зрозумілому середовищі розробки.

3.1.2 Інструмент для створення та управління інтерфейсами користувача Figma

Для розробки дизайну кросплатформеного застосунку використовувалася Figma (рис. 3.2), яка є сучасним інструментом для створення та управління інтерфейсами користувача. Вона надає розробникам і дизайнерам потужні засоби для розробки візуальних компонентів, створення прототипів і співпраці в режимі реального часу. Завдяки своїй гнучкості та можливостям налаштування, цей інструмент дозволяє ефективно створювати та редагувати інтерфейси, забезпечуючи високу продуктивність роботи над дизайном проєкту.

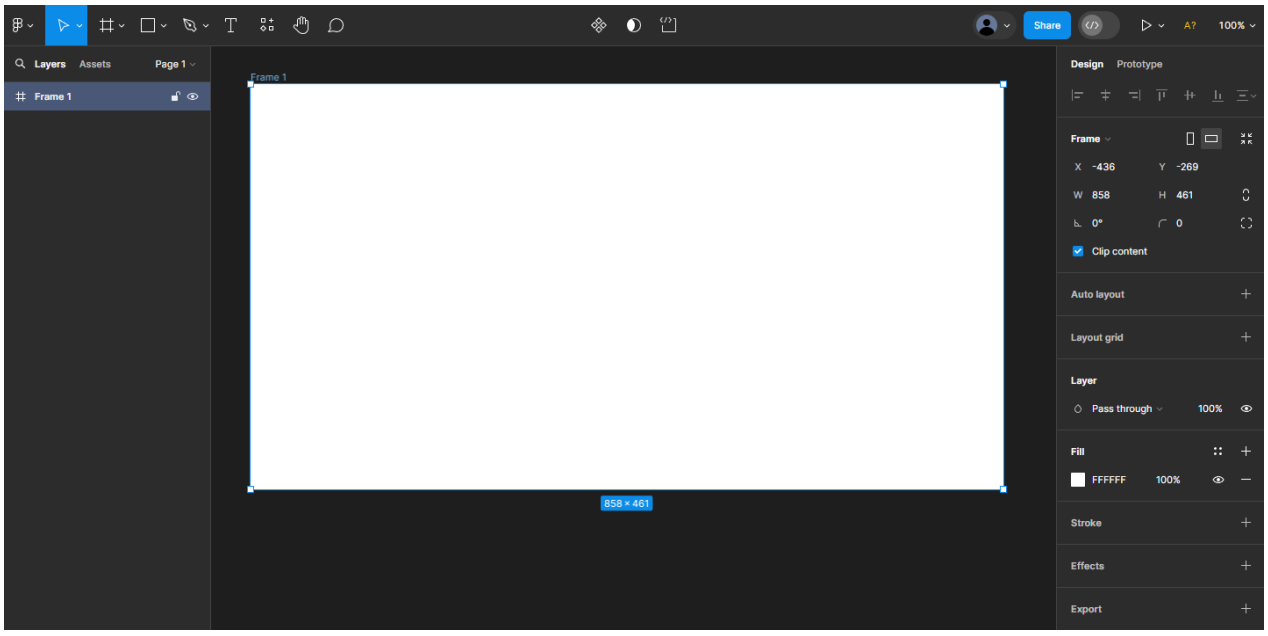

Рисунок 3.2 – Приклад інтерфейсу Figma

Однією з ключових переваг Figma є її кросплатформеність, що дозволяє використовувати інструмент на будь-якій операційній системі з доступом до Інтернету. Це особливо важливо для розподілених команд, де дизайнери можуть працювати над проєктом одночасно, переглядати зміни в режимі реального часу та легко обмінюватися ідеями. Інтеграція з іншими інструментами розробки та можливість експорту дизайнів у різні формати забезпечують зручність використання Figma на всіх етапах розробки.

Figma бере на себе всю допоміжну роботу з управління проєктом, звільняючи дизайнерів від необхідності використовувати кілька окремих інструментів для створення та редагування інтерфейсів. Це дозволяє зосередитися на створенні високоякісного дизайну, який відповідає потребам користувачів і вимогам проєкту. Крім того, Figma надає інструменти для побудови інтерактивних прототипів, що дозволяє тестувати інтерфейси перед їх реалізацією.

Замість того щоб змушувати працювати з графікою на низькому рівні деталізації, Figma надає простий і завершений інтерфейс через свої візуальні компоненти та інтерактивні інструменти. Це дозволяє дизайнерам зосередитися на створенні інтуїтивно зрозумілих та привабливих інтерфейсів, використовуючи потужні інструменти в зручному та інтуїтивно зрозумілому середовищі для дизайну.

3.2 Обґрунтування вибору технологій розробки

У сучасному світі розробки програмного забезпечення важливо використовувати технології, які забезпечують ефективність, гнучкість та високий рівень продуктивності. При виборі стека технологій для проєкту важливо знаходити рішення, що сприяють швидкому створенню якісних продуктів, які зручно підтримувати та розширювати. У цьому контексті,

React.js, React Native та Node.js є одними з найбільш популярних і потужних інструментів для створення сучасних веб та мобільних застосунків.

3.2.1 Бібліотека React

React (React.js) – це бібліотека JavaScript, створена компанією Facebook для розробки інтерфейсів користувача [21]. Вона дозволяє розробникам створювати швидкі, динамічні та інтерактивні вебзастосунки, використовуючи концепцію компонентів.

3.2.1.1 Основні концепції React

Основні концепції React полягають у його компонентах, JSX, властивостях і стані, Virtual DOM і хуках [22, 23]. У React є два основні типи компонентів: функціональні та класові. Функціональні компоненти – це JavaScript-функції, які приймають властивості (props) і повертають елементи React, що визначають вигляд частини інтерфейсу. Класові компоненти є ES6-класами, що розширюють React.Component і мають метод render, який повертає елементи React.

JSX, синтаксичне розширення для JavaScript, виглядає подібно до HTML і дозволяє писати HTML-подібний код у JavaScript, що робить код більш читабельним і зрозумілим.

Властивості (props) – це дані, які передаються компонентам і є незмінними всередині компоненту, забезпечуючи передбачувану поведінку. Стан (state) – це внутрішній стан компоненту, який може змінюватися. Зміна стану призводить до перерисовки компоненту, що дозволяє оновлювати інтерфейс користувача у відповідь на події.

Virtual DOM є ключовою концепцією React, що дозволяє ефективно маніпулювати реальним DOM браузера. Це абстракція, яку React використовує для мінімізації операцій з реальним DOM. Зміни в інтерфейсі спочатку застосовуються до Virtual DOM. Після цього React автоматично обчислює найефективніший спосіб оновлення реального DOM, що забезпечує високу продуктивність застосунків. Використання Virtual DOM дозволяє зберігати копію DOM у вигляді віртуального представлення, що дозволяє покращити продуктивність застосунків шляхом зменшення кількості маніпуляцій з реальним DOM.

Хуки (hooks), такі як useState і useEffect, дозволяють додавати функціональність до функціональних компонентів. Наприклад, useState додає стан, а useEffect дозволяє виконувати побічні ефекти, як-от запити до API або налаштування підписок. Існують також інші хуки, як-от useContext і useReducer, які розширюють можливості функціональних компонентів.

3.2.1.2 Екосистема React

Екосистема React – це об’єднання інструментів, бібліотек, спільноти розробників та ресурсів, які разом формують цілісне середовище для розробки застосунків. Це не просто бібліотека для створення інтерфейсів, а широка платформа, яка підтримується як основними компаніями, так і незалежними розробниками по всьому світу.

Однією з ключових особливостей екосистеми React є її модульність. Можна використовувати різноманітні бібліотеки та інструменти, які допомагають управляти станами, маршрутизацією, здійсненням запитів до сервера, тестуванням та іншими аспектами розробки. Наприклад, Redux як бібліотека для управління станом або React Router для маршрутизації в надають розробникам потужні інструменти для створення складних застосунків.

Запити до сервера можуть бути оптимізовані за допомогою таких бібліотек як Axios чи Fetch API, які інтегруються з React, забезпечуючи ефективне спілкування з бекендом. Для тестування компонентів і застосунків на React існують спеціалізовані інструменти, такі як Jest та Enzyme, які дозволяють розробникам проводити юніт-тестування і глибше інтеграційне тестування.

Також має широкий вибір бібліотек і інструментів, таких як Redux для управління станом, React Router для маршрутизації, Axios і Fetch API для обробки запитів до сервера, Jest і Enzyme для тестування, що дозволяє розробникам легко інтегрувати рішення для різних завдань у свої проекти.

Екосистема React не тільки сприяє технічному розвитку і професійному зростанню розробників, але й забезпечує розширений набір інструментів та можливостей для ефективної та інноваційної розробки вебзастосунків.

3.2.1.3 Переваги вебзастосунків виконаних на React

React володіє низкою суттєвих переваг, що роблять його популярним інструментом серед розробників для створення користувацьких інтерфейсів. Однією з головних переваг є декларативний підхід. Використовуючи React, розробник описує, як має виглядати інтерфейс у кожному стані програми, тоді як React автоматично керує всіма оновленнями DOM при зміні даних. Це значно спрощує розробку, робить код більш читабельним та легким для налагодження.

Крім того, React базується на компонентному підході, що передбачає створення самодостатніх, повторно використовуваних частин інтерфейсу користувача. Це дозволяє розробникам будувати складні інтерфейси шляхом поділу користувацького інтерфейсу на незалежні, керовані блоки, які можуть бути розроблені та протестовані окремо. Такий модульний дизайн сприяє легшій підтримці та масштабуванню застосунків.

JSX, що є розширенням синтаксису JavaScript, дозволяє писати HTML-структури у JavaScript коді. Це полегшує читання та підтримку коду, який створює користувацький інтерфейс, надаючи візуальну репрезентацію компонентів під час розробки.

Значною перевагою React є велика та активна спільнота розробників, що виникла завдяки популярності цієї бібліотеки. Це забезпечує велику кількість навчальних матеріалів, готових до використання компонентів, плагінів та інших ресурсів. Активна спільнота також сприяє швидкому вирішенню проблем та вдосконаленню інструментів.

React також має розвинуті інструменти для розробників, такі як React Developer Tools, які доступні у вигляді розширень для браузерів Chrome та Firefox. Ці інструменти дозволяють розробникам інспектувати компоненти React, спостерігати за їхніми поточними властивостями, станом тощо, що робить процес розробки більш інтуїтивним.

Нарешті, React відзначається великою гнучкістю. Він може бути інтегрований у великі масштабні проєкти, що використовують інші бібліотеки та фреймворки, або застосований для побудови невеликих застосунків чи функціональності на існуючих сторінках. React також підтримує серверний рендеринг, що робить його оптимальним вибором для проєктів, які потребують SEO-оптимізації.

3.2.2 Бібліотека React Native

React Native – це відкрита платформа для розробки мобільних застосунків, яка базується на JavaScript та React [24]. Вона дозволяє розробникам створювати нативні застосунки для iOS та Android за допомогою знайомих інструментів і технологій веброзробки.

React Native має низку основних особливостей, які роблять його потужним інструментом для розробки мобільних застосунків. Він

використовує ті ж основні концепції, що й React.js, такі як компоненти і Virtual DOM, що дозволяє розробникам використовувати свої знання з React для побудови мобільних застосунків. Однією з важливих особливостей React Native є можливість використання нативних компонентів для кожної платформи. Наприклад, розробники можуть використовувати <View> для контейнерів, <Text> для тексту, <Image> для зображень та інші компоненти, які мають нативний вигляд і поведінку на iOS та Android.

Концепція "Write Once, Run Anywhere" є ще однією важливою перевагою React Native. Вона дозволяє писати багатоплатформний код один раз і використовувати його для розгортання на обох платформах (iOS і Android). Це значно зменшує час розробки і спрощує підтримку двох окремих кодових баз. Крім того, можна створювати власні компоненти для повторного використання, що спрощує розробку і збільшує швидкість впровадження нових функцій та оновлень.

Використання JavaScript для логіки програми в React Native забезпечує високу продуктивність застосунків, а також ефективне використання пам'яті і ресурсів пристрою. Це робить застосунки, створені за допомогою React Native, швидкими та ефективними. Велику роль у цьому грає також велика і активна спільнота розробників, яка регулярно вносить нові ідеї, компоненти та інструменти для підтримки розробки. Це включає сторонні бібліотеки, плагіни, теми та навчальні матеріали.

React Native підтримує розробку для різних платформ, включаючи iOS, Android і веб (за допомогою проєкту React Native Web). Це робить його універсальним інструментом для розробників, які хочуть створювати застосунки для різних типів пристроїв.

React Native також відомий своєю інтеграцією з різноманітними сторонніми бібліотеками та сервісами, що значно розширює можливості розробників. Це включає підтримку популярних бібліотек для управління станом, таких як Redux або MobX, а також сервіси для роботи з мережею, базами даних, аналітикою та іншими функціональними можливостями. Такий

підхід дозволяє збільшити швидкість розробки та забезпечити більш гнучкість в реалізації різноманітних функцій у мобільних застосунках.

Загалом, React Native є потужним інструментом для розробки мобільних застосунків, який забезпечує ефективність, продуктивність і можливість використання нативних можливостей пристроїв. Його основні концепції, такі як використання React для компонентів і підтримка нативних інтерфейсів, роблять його привабливим вибором для розробників, які хочуть швидко створювати мобільні застосунки з високою якістю.

3.2.3 Сучасне рішення для серверної розробки Node.js

Node.js є середовищем виконання JavaScript, побудованим на движку V8 від Google Chrome [25]. Він дозволяє виконувати JavaScript на сервері, що відрізняє його від традиційного використання у браузері.

Однією з найбільш визначних характеристик є його асинхронний режим роботи, що дозволяє обробляти багатозадачні операції без блокування інших процесів завдяки використанню колбеків та подій. Важливою властивістю є також використання неблокуючих операцій вводу/виводу, що сприяє ефективному управлінню ресурсами сервера та дозволяє обробляти тисячі одночасних з’єднань без необхідності у великій кількості потоків.

Node.js також відомий своєю роллю платформи для створення сучасних серверних застосунків, таких як вебсервери, API та чат-сервери, з підтримкою різних протоколів, таких як HTTP, HTTPS, TCP та UDP. Разом з пакетним менеджером npm, що постачається разом з Node.js, розробники отримують зручний інструмент для установки, оновлення та видалення сторонніх пакетів JavaScript з репозиторію npm.

Крім того, Node.js має широкий вибір модулів, що значно спрощує використання готових рішень для різних завдань, таких як робота з базами даних, обробка зображень та автентифікація користувачів. Він також ідеально

підходить для фулстек розробки разом з React або іншими фронтенд технологіями, що дозволяє використовувати одну мову програмування (JavaScript) як на клієнтському, так і на серверному боці, спрощуючи розробку та підтримку застосунків [27].

Завдяки активній спільноті розробників, Node.js забезпечує швидке виправлення помилок, покращення і регулярний вихід нових версій, що забезпечує стабільність та подальший розвиток платформи.

Node.js є потужним інструментом для серверної розробки, який забезпечує швидкість, масштабованість і ефективність. Використання однієї мови програмування (JavaScript) на всьому стеку застосунків спрощує розробку та підтримку, роблячи Node.js популярним вибором для сучасних вебзастосунків.

3.2.4 Бібліотека Axios для HTTP запитів у JavaScript

Axios є однією з найпопулярніших бібліотек для здійснення HTTP запитів у JavaScript, і вона використовується як у веброзробці для браузера, так і на боці сервера за допомогою Node.js.

Axios надає простий та зрозумілий інтерфейс для виконання HTTP запитів. Він підтримує всі основні методи HTTP, такі як GET, POST, PUT, DELETE, PATCH і інші, що робить його дуже універсальним для різних вебпроєктів.

Ця бібліотека використовує обіцянки (promises) для зручної обробки результатів запитів. Завдяки цьому можна легко керувати послідовністю запитів і забезпечити асинхронну обробку даних без необхідності вкладати багато каскадних колбеків.

Однією з ключових переваг Axios є його міжплатформова підтримка. Він працює як у веббраузерах, так і у середовищі Node.js, що дозволяє

розробникам використовувати однаковий інструментарій для клієнтської і серверної сторін.

Axios дозволяє легко налаштовувати та перехоплювати HTTP запити та відповіді. Це включає можливість додавати заголовки до кожного запиту, обробляти помилки або логувати інформацію про запити для подальшого аналізу і налагодження.

Ще однією корисною функцією Axios є автоматична конвертація даних. Він автоматично конвертує дані у JSON формат, якщо вони передаються як об’єкти JavaScript, що спрощує взаємодію з багатьма вебсерверами, що підтримують JSON.

Крім того, Axios підтримує зручні методи для скачування і завантаження файлів, що робить його ідеальним інструментом для роботи з API, які передбачають роботу з файлами.

Інтерцептори в Axios дозволяють додавати спеціальні функції, які обробляють запити та відповіді перед їх відправленням або обробкою. Це дозволяє, наприклад, автоматично додавати токени аутентифікації до кожного запиту або обробляти загальні помилки для всього застосунку.

У підсумку, Axios є потужним інструментом для роботи з HTTP запитами у JavaScript, який пропонує чистий і зрозумілий API, різні корисні можливості і високу гнучкість для різних умов веброзробки.

3.2.5 Використання бібліотеки styled-components для стилізації вебзастосунків

Бібліотека styled-components для React перевертає звичний підхід до стилізації вебзастосунків, використовуючи концепцію «CSS в JavaScript». Це означає, що стилі компонентів описуються безпосередньо в JavaScript файлі, що дозволяє використовувати всі можливості мови, такі як змінні, умовні

оператори та ітерація, для генерації CSS. Такий підхід робить код більш структурованим та підвищує його переиспользуваність.

Оголошення стилів відбувається за допомогою спеціальних функцій-конструкторів, що створюють стилізовані компоненти. Це дозволяє описати стилі прямо поруч із визначенням компоненту, що підвищує читабельність коду та спрощує управління стилями.

Styled-components підтримує динамічні стилі, що дозволяє змінювати їх в залежності від пропсів компоненту. Це дає можливість створювати універсальні компоненти, які можуть змінювати свій вигляд в залежності від введених даних чи стану.

Крім стилізованих компонентів, бібліотека підтримує глобальні стилі, які дозволяють задавати стилі для всього застосунку або окремих елементів без прив’язки до конкретних компонентів.

Однією з великих переваг цієї бібліотеки є його продуктивність і оптимізація. Стилі компілюються лише один раз і кешуються, що знижує витрати на виконання і підвищує швидкість застосунку.

Також важливо, що вона легко інтегрується з іншими популярними бібліотеками і фреймворками React, що робить його універсальним інструментом для будь-яких вебзастосунків.

Загалом, styled-components забезпечує розробникам потужний інструмент для стилізації React компонентів, який поєднує в собі простоту використання, гнучкість управління стилями і високу продуктивність, що робить його пріоритетним вибором для багатьох веброзробників.

3.2.6 Використання Swiper для створення слайдерів

Бібліотека Swiper.js для React та React Native є потужним інструментом для створення сучасних, мобільно-дружніх карусельних слайдерів та галерей на вебсторінках та сторінках застосунку. Вона базується на оригінальній

Swiper.js, яка спеціалізується на різноманітних варіантах карусельного відображення вмісту, включаючи слайдери, галереї, карти тощо. Ця бібліотека надає розробникам можливість легко і ефективно інтегрувати ці каруселі і слайдери в їхні проєкти, забезпечуючи високу продуктивність і користувацький досвід.

Swiper.js підтримує багато функціональних можливостей, таких як безкінечне прокручування, автоматична прокрутка, перехід між слайдами за допомогою пальців або миші, анімації переходу, налаштування ефектів переходу, адаптивність до розміру вікна браузера та багато іншого. Бібліотека також підтримує додаткові функції, такі як горизонтальні і вертикальні каруселі, множинні галереї, пагінація, налаштування кернців, керування жестами і кастомізація інтерфейсу.

Swiper.js пропонує чистий API для інтеграції та керування каруселями і слайдерами, забезпечуючи зручний спосіб розробки і підтримки інтерактивних елементів на вебсайтах і застосунках.

3.3 Реалізація дизайну

Реалізація дизайну в Figma є структурованим процесом, що складається з кількох етапів, починаючи з розробки прототипу і закінчуючи фінальним дизайном [28].

На етапі прототипування розробляються важливі аспекти майбутнього програмного забезпечення (рис 3.3). Початковим завданням є збір усіх необхідних вимог до застосунку, що включає аналіз цільової аудиторії, їх потреб і основні проблеми, які необхідно вирішити. Також враховуються функціональні вимоги, які визначають функціонал програми.

На основі зібраних вимог розробляються вайрфрейми – спрощені чорно-білі схеми майбутнього інтерфейсу, які відображають розташування основних елементів на екрані. Вайрфрейми допомагають зрозуміти структуру

інтерфейсу та основні сценарії взаємодії користувачів, використовуючи мінімальну деталізацію для концептуалізації дизайну.

У програмному інструменті Figma реалізується створення інтерактивного прототипу, що дозволяє клієнтам та розробникам взаємодіяти з модельним зразком так само, як з готовим продуктом. Це сприяє виявленню можливих проблем у навігації та користувацькому досвіді на початкових етапах розробки, що дозволяє вчасно коригувати концепції і поліпшувати взаємодію з програмним інтерфейсом.

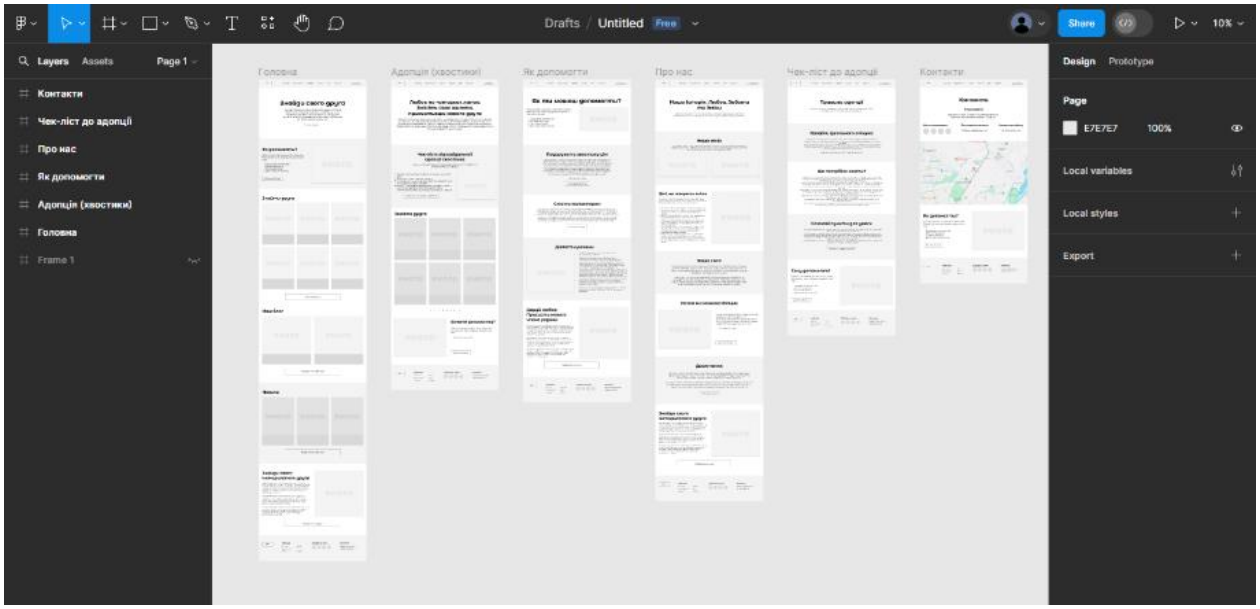

Рисунок 3.3 – Прототип для застосунку щодо допомоги безпритульним тваринам

На етапі візуального дизайну програмного забезпечення ретельно розробляються естетичні та функціональні аспекти інтерфейсу (рис 3.4). Починаючи з розроблення візуальних стилів, визначаються кольорові палітри, типографіка, іконки та інші графічні елементи. Використання бібліотек компонентів у Figma сприяє створенню єдинообразного дизайну та підвищує ефективність у розробці.

На основі вайрфреймів та візуальних стилів розробляються детальні макети кожного екрану програми. У цьому процесі всі елементи інтерфейсу

отримують свій фінальний вигляд, включаючи відповідність кольорів, шрифтів, іконок та інших графічних елементів з вимогами проєкту і з метою забезпечення комфортного користувацького досвіду.

Останнім етапом візуального дизайну є перевірка фінальних макетів за допомогою юзабіліті-тестування. Це дозволяє ідентифікувати можливі недоліки у дизайні та користувацькому досвіді ще до початку активної розробки. Інтерактивні прототипи в Figma сприяють ефективному проведенню тестування, підвищуючи шанси на виявлення й усунення проблем на ранніх стадіях розробки програмного продукту.

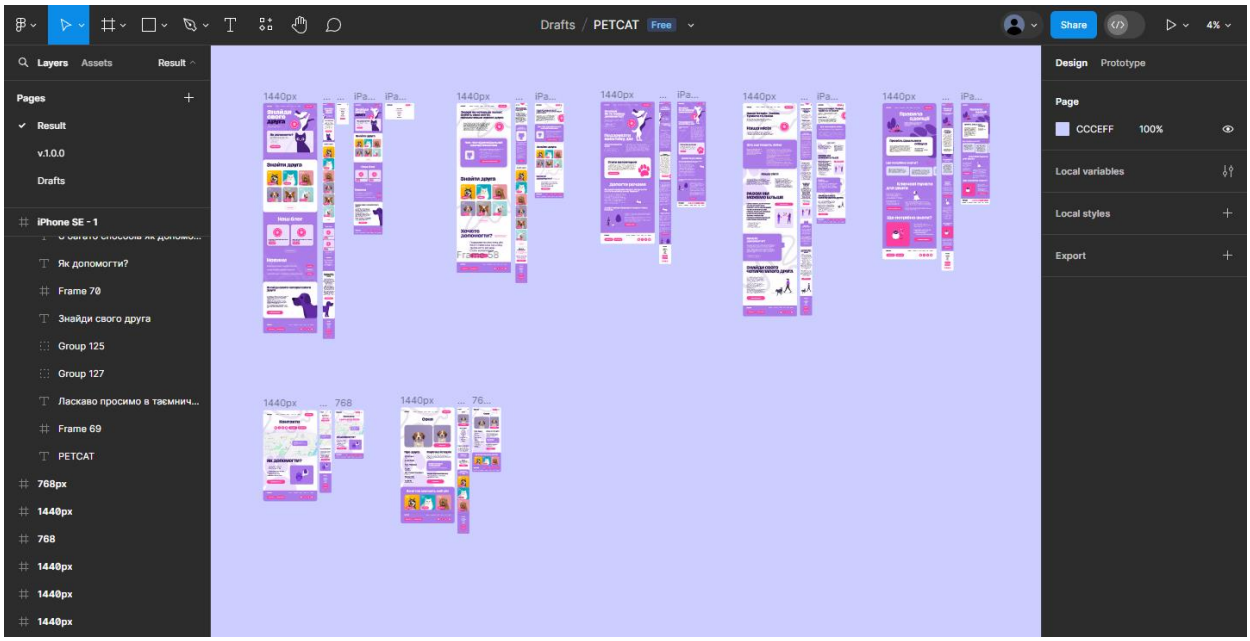

Рисунок 3.4 – Візуальний дизайн

На етапі фіналізації дизайну важливо ураховувати зворотний зв'язок від користувачів та зацікавлених сторін після юзабіліті-тестування. Отримані результати служать основою для внесення необхідних коректив у фінальні макети, зокрема узгодженість та завершеність всіх екранів інтерфейсу.

На основі оновлених макетів створюється дизайн-система, що включає всі компоненти, стилі та правила їх використання. Це сприяє забезпеченню узгодженості та ефективності в інтеграції дизайну у програмний код, що підтримується на всіх етапах проєкту.

Останнім кроком є експорт усіх необхідних графічних елементів та підготовка документації для розробників. Figma надає можливість експортувати елементи у різних форматах і забезпечує детальну інформацію про стилі та компоненти, що спрощує процес інтеграції дизайну в програмний продукт.

3.4 Програмна реалізація

При відкриті вебзастосунку відкривається за замовчуванням головна сторінка, де одразу можна розгледіти хедер, де є логотип сайту, меню навігації, кнопку, яка закликає до дії, та hero секцію, яка містить велике зображення разом із захоплюючою назвою та коротким описом, метою якого є залучити користувачів та передати основну цінність вебзастосунку (рис. 3.5).

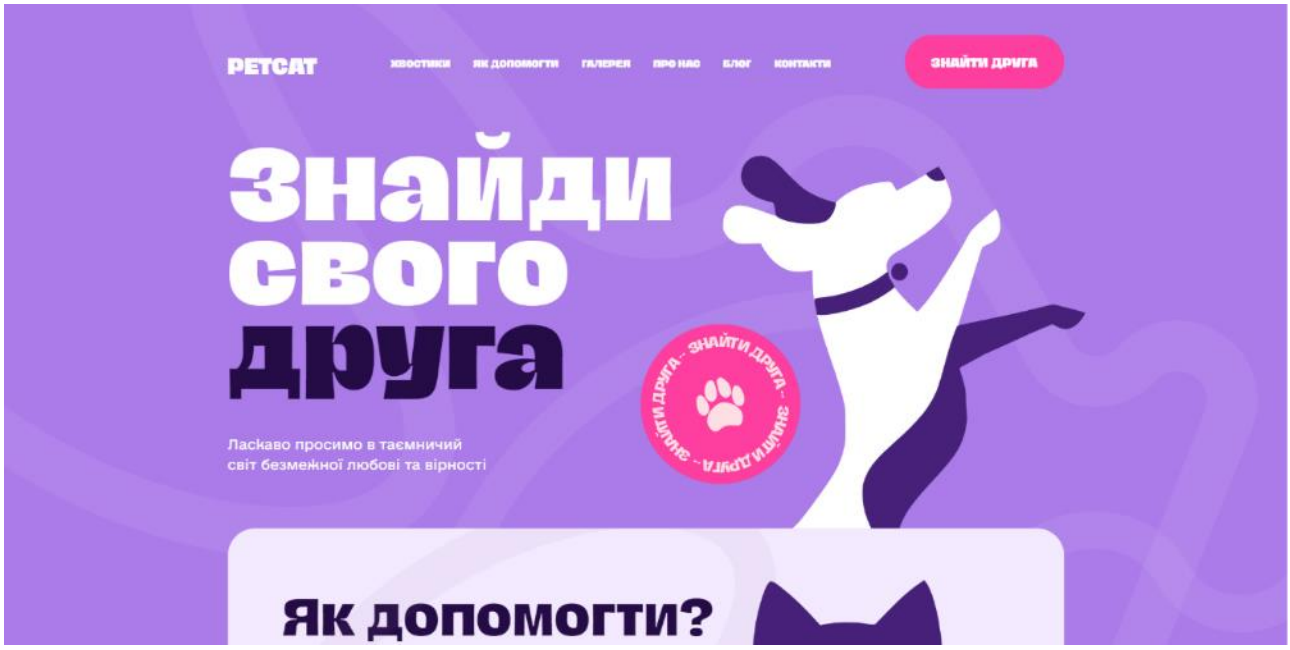

Рисунок 3.5 – Header та hero секція головного екрану

При відкритті цього ж самого сайту на планшетному (рис. 3.6) або мобільному (рис. 3.7) пристроях не буде навігаційного меню, бо ширини екрану недостатньо для відображення такого елементу, воно ховається в бічній

панелі (рис. 3.8), яку можно відкрити натисканням на іконку з відображенням лапи.

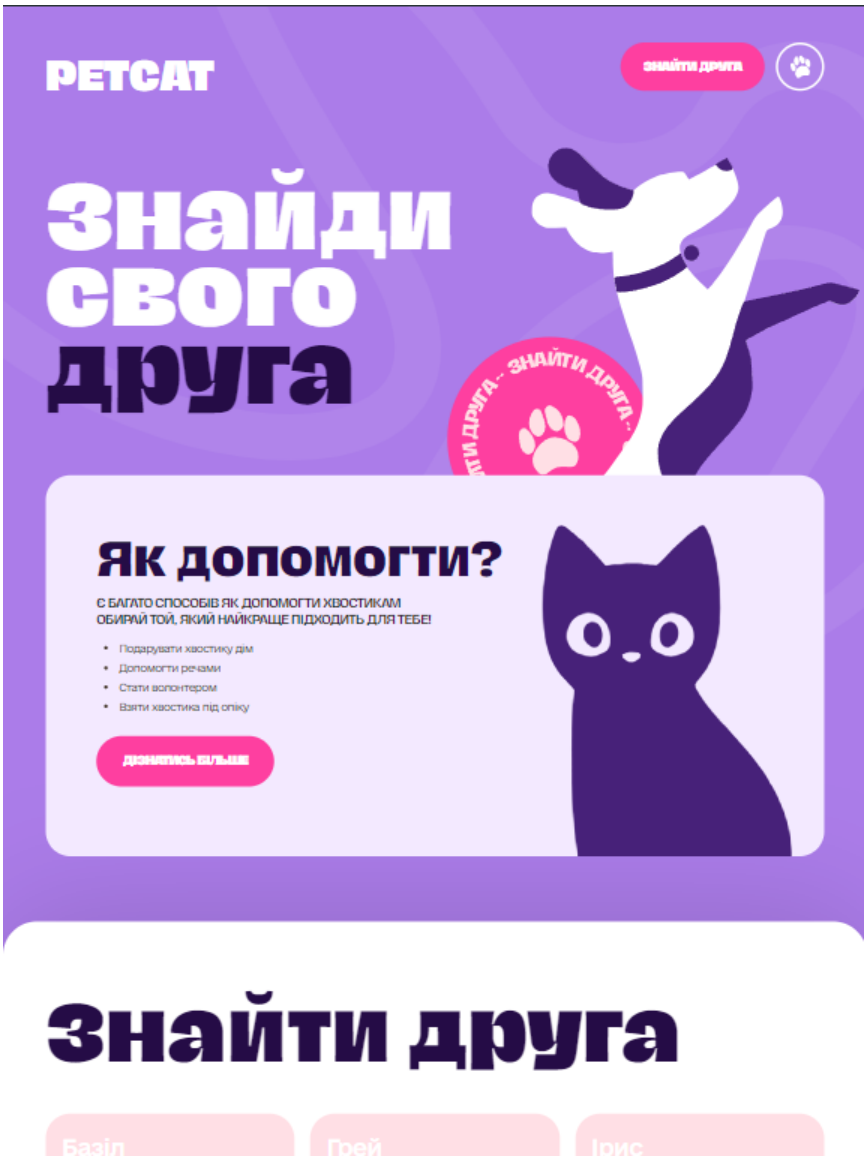

Рисунок 3.6 – Вигляд сайту на планшетному пристрої

На hero секції розташована кнопка «Знайти друга», яка привертає до себе увагу і веде на сторінку «Хвостики» (рис. 3.9), де користувач може продивитись тварин, які потребують сім’ї, познайомитись з ними та дізнатись про правила адопції.

Рисунок 3.7 – Вигляд сайту на мобільному пристрої

Також на мобільному та планшетному пристроях можна побачити наступну секцію, яка заохочує до допомоги безхатнім тваринам і веде на сторінку «Як допомогти» (рис. 3.10), де користувач може знайти інформацію та дізнатись про існуючі способи.

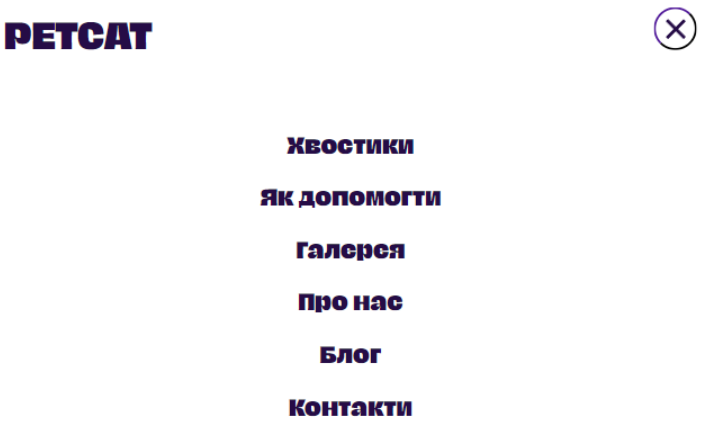

Рисунок 3.8 – Мобільне меню

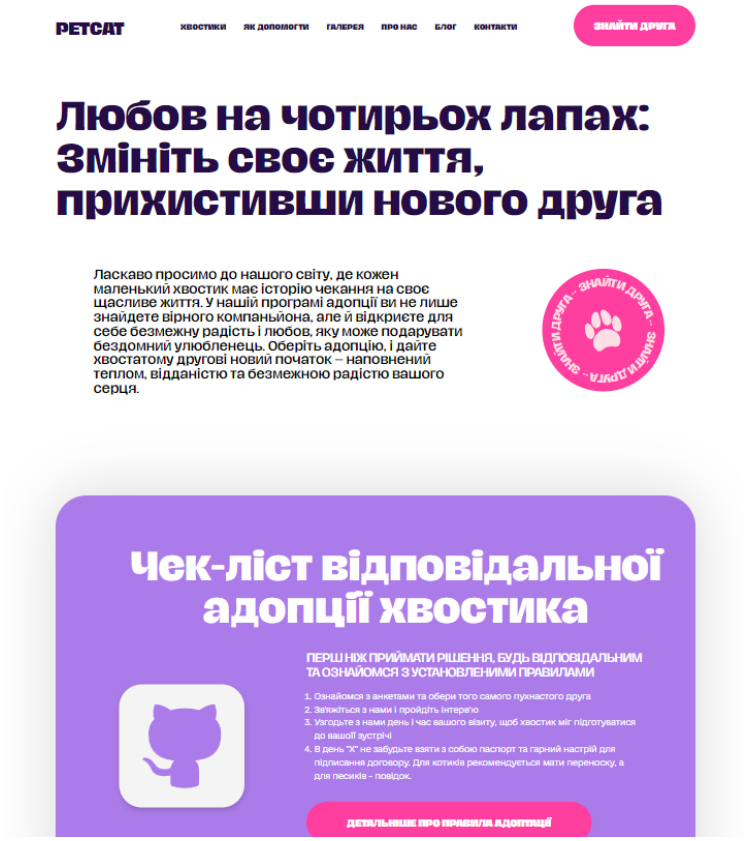

Рисунок 3.9 – Сторінка «Хвостики»

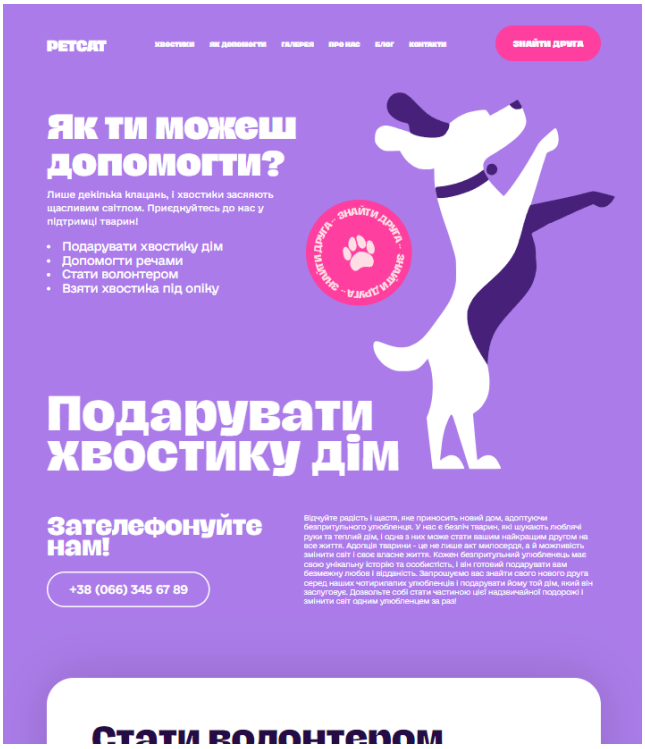

Рисунок 3.10 – Сторінка «Як допомогти»

Також на головній сторінці можна побачити секції блогу та новин (рис. 3.11).

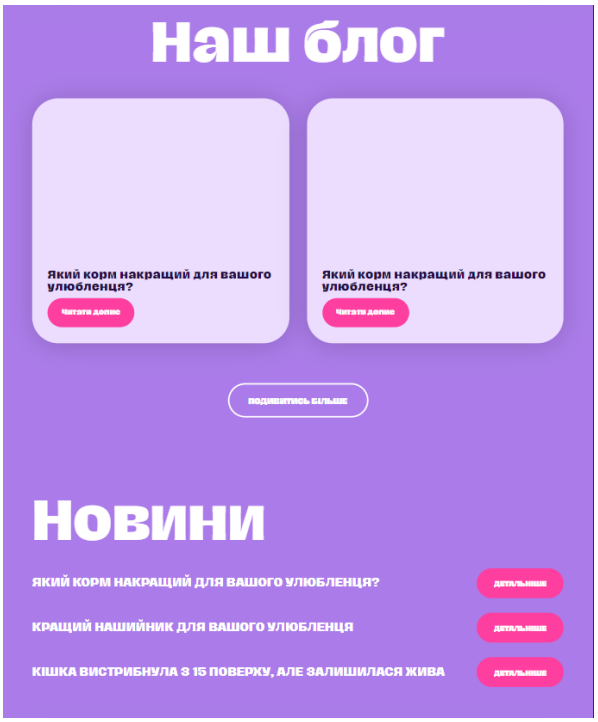

Рисунок 3.11 – Секції «Блог» та «Новини»

Для маршрутизації застосунку використовувався React Router (лістинг 3.1).

Лістинг 3.1 Маршрутизація застосунку:

```
import { lazy } from 'react';
import { Route, Routes } from 'react-router-dom';
import SharedLayout from './SharedLayout';

const Home = lazy(() => import('../pages/Home'));
const AboutUs = lazy(() => import('../pages/AboutUs'));
const Adoption = lazy(() => import('../pages/Adoption'));
const PetPage = lazy(() => import('../pages/PetPage'))
const CheckList = lazy(() => import('../pages/CheckList'));
const Help = lazy(() => import('../pages/Help'));
```

```
const Contacts = lazy(() => import('../pages/Contacts'));

export const App = () => {
  return (
    <div>
      <Routes>
        <Route path="/" element={<SharedLayout />}>
          <Route index element={<Home />} />
          <Route path="/about-us" element={<AboutUs />} />
          <Route path="/adoption" element={<Adoption />} />
          <Route path="/adoption/:petId" element={<PetPage />} />
          <Route path="/check-list" element={<CheckList />} />
          <Route path="/help" element={<Help />} />
          <Route path="/contacts" element={<Contacts />} />
          <Route path="*" element={<Home />} />
        </Route>
      </Routes>
    </div>
  );
};
```

На головній сторінці присутня секція «Знайти друга» (рис. 3.12), де відображається 3 картки тварин, на яких стисло відображається інформація про них і кнопка-посилання, яка веде на сторінку детальної інформації (рис. 3.13), і кнопка «Усі хвостики», натиснувши на яку можна перейти на сторінку «Хвостики», де є схожа секція, але вона підтримує пагінацію (лістинг 3.2) при розгортанні списку.

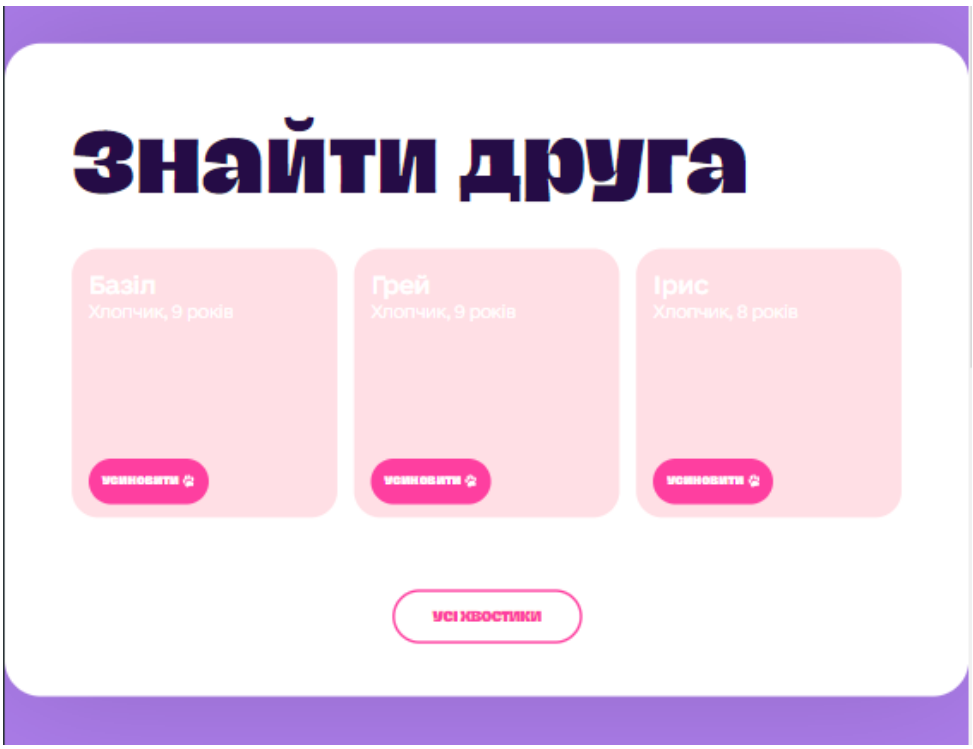

Рисунок 3.12 – Секція «Знайти друга»

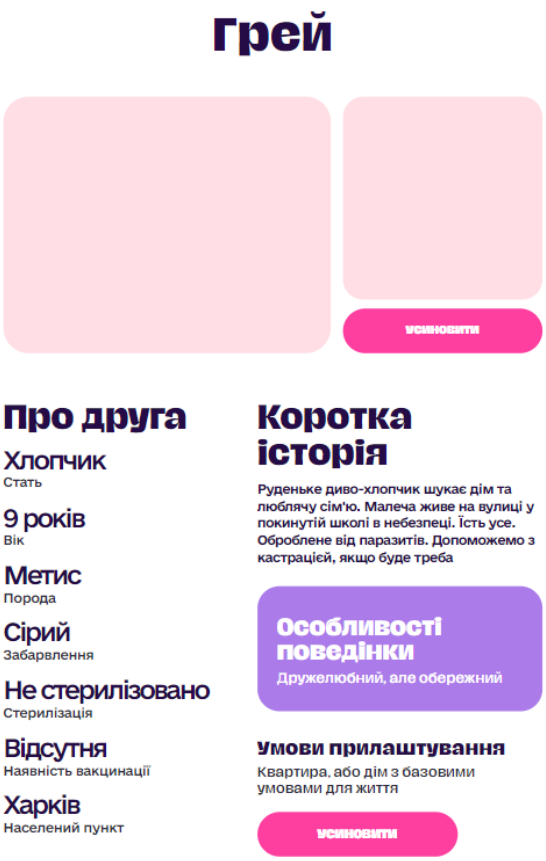

Рисунок 3.13 – Сторінка з детальною інформацією про тварину

Лістинг 3.2 Реалізація пагінації:

```
const AdoptionFindFriend = () => {
  ...
  const [currentPage, setCurrentPage] = useState(1);
  const [totalPages, setTotalPages] = useState(1);
  const [isPagination, setIsPagination] = useState(false);

  const handleNextPage = () => {
    if (currentPage < totalPages) {
      setCurrentPage(prevPage => prevPage + 1);
    }
  };

  const handlePreviousPage = () => {
    if (currentPage > 1) {
      setCurrentPage(prevPage => prevPage - 1);
    }
  };
 ...
  return (
    <Section>
      <Container>
        <Box>
          ...
        {!isPagination && (
          <MoreButton onClick={() => setIsPagination(true)}>
            Усі хвостики
          </MoreButton>
        )}
```

```
        {isPagination && (
          <PaginationWrapper>
            <PaginationButton
              onClick={handlePreviousPage}
              disabled={currentPage === 1}
            >
              <PaginationIcon src={prevButton} />
            </PaginationButton>
            <PaginationText>
              Сторінка {currentPage} з {totalPages}
            </PaginationText>
            <PaginationButton
              onClick={handleNextPage}
              disabled={currentPage === totalPages}
            >
              <PaginationIcon src={nextButton} />
            </PaginationButton>
          </PaginationWrapper>
        )}
      </Box>
    </Container>
  </Section>
 );
};
```

Був реалізований бекенд для того, щоб зберігати дані про тварин (лістинг 3.3), а на фронтенді – функції для отримання цих даних з БД з можливістю пагінації та конкретної тварини за її ідентифікатором (petId) (лістинг 3.4)

Лістинг 3.3 Реалізація бекенду:

```
const listPets = async (req, res, next) => {
  let { page = 1, limit = 10, sort = "desc" } = req.query;
  page = Number(page);
  limit = Number(limit);

  if (isNaN(page) || page < 1) {
    page = 1;
  }
  if (isNaN(limit) || limit < 1) {
    limit = 10;
  }

  if (sort !== "asc" && sort !== "desc") {
    sort = "desc";
  }

  const total = await Pet.countDocuments();
  const skip = (page - 1) * limit;
  const sortOption = sort === "desc" ? -1 : 1;
  const result = await Pet.find()
    .sort({ age: sortOption })
    .skip(skip)
    .limit(limit);

  res.status(200).json({
    page,
    limit,
    total,
```

```
    totalPages: Math.ceil(total / limit),
    data: result,
  });
};

const getPetById = async (req, res, next) => {
  const { petId } = req.params;
  const result = await Pet.findById(petId);
  if (!result) {
    throw HttpError(404, "Not Found");
  }
  res.status(200).json(result);
};
```

Лістинг 3.4 Реалізація функцій на фронтенді для отримання даних з БД:

```
export const fetchCats = async (limit, page = 1) => {
  const response = await axios.get(
    `${CATS_END_POINT}?page=${page}&limit=${limit}`
  );
  return response.data;
};

export const fetchOnePet = async petId => {
  const response = await axios.get(`${CATS_END_POINT}${petId}`);
  return response.data;
};
```

Також є сторінка «Про нас», де є інформація про притулок, про цілі і бажання (рис. 3.14).

Рисунок 3.14 – Сторінка «Про нас»

ВИСНОВКИ

У рамках кваліфікаційної роботи був розроблений кросплатформений застосунок щодо допомоги тваринам. Цей застосунок надає користувачам приємний та зручний інтерфейс та заохочує до допомоги тваринам.

Метою цієї роботи було створення та впровадження кросплатформеного застосунку для надання допомоги безпритульним тваринам. Розробка цієї програми була спрямована на покращення координації та ефективності процесу допомоги безпритульним тваринам.

Для досягнення цієї мети було використано сучасні технології, досліджено та впроваджено метод кластеризації *k*-середніх для ефективного управління інформацією про тварин, що потребують допомоги, що сприяє збільшенню ефективності та охоплення допомоги

За допомогою цього застосунку користувачі, які хочуть допомогти, дізнаються, як вони це можуть зробити, а ті, які шукають друга, матимуть під рукою зручний інструмент.

Для поліпшення цього застосунку можна буде додати додаткові модулі для відстеження медичного стану тварин, ведення інтерактивних журналів догляду, розширити географічне охоплення, тобто впровадити використання геолокації для знаходження тварин і підтримки допомоги у різних регіонах, можливість місцевого підтримання та співпраці з волонтерами, а також додати інструменти аналітики для збору статистичних даних про ефективність програми, вивчення трендів у проблемі безпритульних тварин та впливу допомоги на їхнє життя.

Результати роботи апробовано у вигляді тез доповідей під час Міжнародного молодіжного форуму «Радіоелектроніка і молодь у XXI столітті» [32].

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Безпритульні тварини: вбити неможливо прихистити? URL: https://epl.org.ua/eco-analytics/bezprytulni-tvaryny-vbyty-nemozhlyvo-pryhystyty/ (дата звернення 08.04.2024)

2. Кобилін, І. О. (2019). Нечітка кластеризація часових рядів в інтелектуальному аналізі потоків даних.

3. Bodyanskiy, Y., Kobylin, I., Rashkevych, Y., Vynokurova, O., & Peleshko, D. (2018, February). Hybrid fuzzy-clustering algorithm of unevenly and asynchronously spaced time series in computer engineering. In 2018 14th International Conference on Advanced Trends in Radioelecrtronics, Telecommunications and Computer Engineering (TCSET) (pp. 930-935). IEEE.

4. Bodyanskiy, Y., Vynokurova, O., Kobylin, I., & Kobylin, O. (2016). Adaptive fuzzy clustering of short time series with unevenly distributed observations in Data Stream Mining tasks. Information Technology and Management Science, 19(1), 23-28.

5. Kobylin, O., & Lyashenko, V. (2020). Time series clustering based on the k-means algorithm.

6. Setlak, G., Bodyanskiy, Y., Pliss, I., Vynokurova, O., Peleshko, D., & Kobylin, I. (2018). Adaptive fuzzy clustering of multivariate short time series with unevenly distributed observations based on matrix neuro-fuzzy self-organizing network. In Advances in Fuzzy Logic and Technology 2017: Proceedings of: EUSFLAT-2017–The 10th Conference of the European Society for Fuzzy Logic and Technology, September 11-15, 2017, Warsaw, Poland IWIFSGN’2017–The Sixteenth International Workshop on Intuitionistic Fuzzy Sets and Generalized Nets, September 13-15, 2017, Warsaw, Poland, Volume 3 10 (pp. 308-315). Springer International Publishing.

7. Ermshaus, A., Schäfer, P., & Leser, U. (2023). Window size selection in unsupervised time series analytics: A review and benchmark. In International

Workshop on Advanced Analytics and Learning on Temporal Data (pp. 83-101). Springer, Cham.

8. Keogh, E., Taposh, D. R., Naik, U., & Agrawal, A. (2021). Multi-dataset time-series anomaly detection competition. In ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.

9. Lu, Y., Srinivas, T. V. A., Nakamura, T., Imamura, M., & Keogh, E. (2023, December). Matrix Profile XXX: MADRID: A Hyper-Anytime and Parameter-Free Algorithm to Find Time Series Anomalies of all Lengths. In 2023 IEEE International Conference on Data Mining (ICDM) (pp. 1199-1204). IEEE.

10. Gorokhovatskyi, V., Tvoroshenko, I., Kobylin, O., & Vlasenko, N. (2023). Search for visual objects by request in the form of a cluster representation for the structural image description. Advances in Electrical and Electronic Engineering, 21(1), 19-27.

11. Bodyanskiy, Y. V., Shafronenko, A., & Klymova, I. (2021, April). Adaptive Recovery of Distorted Data Based on Credibilistic Fuzzy Clustering Approach. In COLINS (pp. 6-15).

12. Newman, S. (2021). Building microservices. " O'Reilly Media, Inc.".

13. Realtime notifications using Websocket, Angular and SpringBoot. URL: https://medium.com/@shuklendu.ayachit/realtime-notifications-using-websocket-angular-and-springboot-37c4da8c03c3 (дата звернення 14.04.2024).

14. Фреймворк gRPC: як із ним працювати та чим він кращий за REST API. URL: https://proit.org.ua/frieimvork-grpc-iak-iz-nim-pratsiuvati-ta-chim-vin-krashchii-za-rest-api/ (дата звернення 15.04.2024).

15. GraphQL: only what you need. URL: https://medium.com/@jdelamettrie/graphql-only-what-you-need-5e1656c21e6c (дата звернення 15.04.2024).

16. Introduction to GraphQL. URL: https://medium.com/swlh/introduction-to-graphql-e887ccfbb92a (дата звернення 15.04.2024).

17. How the Web Works Part II: Client-Server Model & the Structure of a Web Application. URL: https://www.linkedin.com/pulse/how-web-works-part-ii-client-server-model-structure-preethi-kasireddy/ (дата звернення 26.04.2024).

18. Wolff, E. (2019). Microservices: A Practical Guide: Principles, Concepts, and Recipes. Manning.

19. Mezzalira, L. (2021). Building Micro-Frontends. " O'Reilly Media, Inc.".

20. Truica, C. O., Radulescu, F., Boicea, A., & Bucur, I. (2015, May). Performance evaluation for CRUD operations in asynchronously replicated document oriented database. In 2015 20th International Conference on Control Systems and Computer Science (pp. 191-196). IEEE.

21. React.js. URL: https://react.dev/ (дата звернення 16.04.2024).

22. Танянський, О., & Руденко, Д. (2018). Порівняльний аналіз популярних JavaScript-фреймворків та бібліотек для front-end розробки.

23. Похваленна, О. Д., & Штих, І. А. (2023). Порівняльний аналіз технічних характеристик найпопулярніших фреймворків та бібліотек JS (Doctoral dissertation, ХНУРЕ).

24. React Native. URL: https://reactnative.dev/ (дата звернення 20.04.2024).

25. Node.js. URL: https://nodejs.org/en (дата звернення 20.04.2024).

26. MongoDB. URL: https://www.mongodb.com/ (дата звернення 20.04.2024).

27. Кириченко, І. В., Назаренко, А. В., & Попов, Р. О. (2020). Оптимізація та масштабування Node .js додатків.

28. Figma. URL: https://help.figma.com/hc/en-us (дата звернення 08.04.2024).

29. Mashtalir, S., Mashtalir, V., & Stolbovyi, M. (2018, August). Representative based clustering of long multivariate sequences with different lengths. In 2018 IEEE second international conference on Data Stream Mining & Processing (DSMP) (pp. 545-548). IEEE.

30. Shafronenko, A. Y., Bodyanskiy, Y. V., & Pliss, I. P. (2019, September). The Fast Modification of Evolutionary Bioinspired Cat Swarm Optimization

Method. In 2019 IEEE 8th International Conference on Advanced Optoelectronics and Lasers (CAOL) (pp. 548-552). IEEE.

31. Kobylin, O. A., Gorokhovatskyi, V. O., Tvoroshenko, I. S., & Peredrii, O. O. (2020). The application of non-parametric statistics methods in image classifiers based on structural description components. Telecommunications and Radio Engineering, 79(10).

32. Бєгунова В.Д. (2024). Застосування кластеризації на основі машинного навчання для вдосконалення управління популяціями бездомних тварин.