МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

Кваліфікаційна робота бакалавра

ВЕБ-СЕРВІС ДЛЯ ПІДТРИМКИ ДІЯЛЬНОСТІ ПРИТУЛКУ ДЛЯ ТВАРИН З ВИКОРИСТАННЯМ КАРТОГРАФІЧНОГО API

Здобувачка освіти гр. ІН-82/2 Марія СЕМЕНЕНКО

Наукова керівниця,
к.ф.-м.н., доцентка Олена ПРОЦЕНКО

Завідувач кафедри,
д.т.н., професор Анатолій ДОВБИШ

СУМИ 2022

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

Затверджую_____
зав. кафедрою Довбиш А.С.
"_____" _____2022 р.

ЗАВДАННЯ
до випускної роботи

Студентки четвертого курсу, групи ІН-82/2 спеціальності "122 – Комп'ютерні науки" денної форми навчання Семененко Марії Андріївни.

Тема: " Веб-сервіс для підтримки діяльності притулку для тварин з використанням картографічного API "

Затверджено наказом по СумДУ
№_____ від _____2022 р.

Зміст пояснювальної записки: 1) Огляд теми та існуючих рішень. Огляд проблемної області. Огляд побідних рішень. Актуальність створення веб-сервісу. Постановка задачі. 2) Методика вирішення поставлених задач. Вибір засобів програмування. Паттерн проектування. Система керування базою даних. Проектування інформаційної системи. Макети. 3) Практична реалізація. Проектування бази даних. Реалізація карти. Зберігання зображень. Адміністративна панель. Використання веб-сервісу.

Дата видачі завдання "___"_____2022 р.
Керівниця випускної роботи _____Проценко О.Б.
Завдання прийняла до виконання _____Семененко М.А.

РЕФЕРАТ

Записка: 150 стор., 36 рис., 3 табл., 1 додаток, 30 джерел.

Об’єкт дослідження — проблема безпритульності тварин в місті Суми, алгоритм та методи систематизації наявних даних, організація ведення обліку, процес пошуку та прилаштування.

Мета роботи — розробка веб-сервісу для пришвидшення процесу пошуку тварин та їх господарів з використанням картографічного API та сучасних інформаційних технологій.

Методи дослідження — аналіз і вивчення літератури, розгляд тематичних джерел, порівняльний аналіз, узагальнення, спостереження, моделювання, опис.

Результати — розроблено метод оптимізації збору даних про безпритульних тварин за допомогою онлайн-форми для отримання інформації, інструмент швидкого пошуку тварин та їх господарів з використанням картографічного API. Інструмент реалізовано у вигляді веб-сервісу за допомогою мови розмітки HTML, мови опису зовнішнього вигляду CSS, формату передачі даних JSON, програмної платформи Node.JS, мови програмування JavaScript, фреймворку Express.js та модулю Mongoose. Розроблюваний веб-сервіс сприяє боротьбі з явищем безпритульності тварин і оптимізує діяльність Сумського товариства захисту тварин.

БЕЗПРИТУЛЬНІ ТВАРИНИ, ЗАХИСТ ТВАРИН, ПОШУК ТВАРИН, ВЕБ-СЕРВІС, ІНФОРМАЦІЙНА СИСТЕМА, ІНСТРУМЕНТИ ПОШУКУ, ОБРОБКА ДАНИХ, НЕРЕЛЯЦІЙНІ БАЗИ ДАНИХ, MONGODB, КАРТОГРАФІЧНИЙ API, JAVASCRIPT, NODE.JS, MVC.

ЗМІСТ

ВСТУП

На сьогодні в місті Суми існує серйозна проблема безпритульних тварин. Останні декілька років збільшилась кількість звернень громадян щодо її подолання на території нашого міста. Для успішного вирішення будь-якого гострого питання варто, перш за все, усвідомити причини його виникнення. А причин такого явища як «безпритульність» можна перелічити багато: невідповідальні господарі, які позбавляються набридлих улюбленців, не зваживши ретельно свої можливості перед їх заведенням, нерегулюване розмноження тварин через відсутність культури стерилізації в суспільстві, культ і статусність породистих тварин, що спонукає людей витрачати гроші на придбання замість допомоги звірям з переповнених притулків, неуважність та необачність під час прогулянок і недотримання правил вигулу домашніх тварин. Але не завжди улюбленці вступають в ряди безпритульних через людський фактор, адже є обставини, які нам не підвладні. Важливим аспектом даної проблеми є той факт, що часто тварини губляться випадково. Цьому також є ряд своїх причин, наприклад відсутність обов'язкового чипування в нашій країні на законодавчому рівні, яке вже довело свою ефективність у боротьбі з безпритульністю в країнах Європи.

Якщо тварина все ж таки загубилася з якоїсь з перелічених вище причин, то зазвичай пошук – досить довгий і мало контрольований процес, в якому все залежить здебільше від волі випадку, ніж від конкретної послідовності дій чи величезного бажання господарів нарешті знайти чотирилапого друга. Серед місцевих мешканців найбільшою популярністю користується такий спосіб пошуку, як розклеювання паперових об'яв, що насправді є застарілим та малоефективним явищем. Єдиною надією знайти свою домашню тварину наразі є вірогідність, що хтось із знайомих або сусідів випадково бачив знайому морду десь неподалік від місця втрати. Проте, звісно, таке співпадіння має досить невеликий шанс в реальному житті.

В нашому місті існує лише одна організація, чия діяльність спрямована на боротьбу з явищем безпритульності тварин, захист людей від її наслідків і, перш за все, захист прав самих тварин, і це – Сумське товариство захисту тварин. Всі інші організації, про які можна почути, займаються або короткостроковою перетримкою тварин або частковою допомогою у вигляді фінансової підтримки або підгодовування бездомних собак і котів. Сумське товариство захисту тварин існує виключно за рахунок благодійних внесків і вкрай потребує будь-якої можливої допомоги, будь то пожертвування грошей чи непотрібних речей, які можуть стати в нагоді, розроблення інструментів для полегшення і пришвидшення його роботи або ж просто розповсюдження інформації про діяльність на сторінках в соціальних мережах. Насправді допомога єдиній організації, яка стоїть на захисті наших менших друзів і нас самих є головним аспектом у вирішенні даного питання.

Враховуючи все вищесказане, можна зробити висновок, що питання захисту і оптимізації пошуку безпритульних тварин в Сумах на сьогоднішній день стоїть досить гостро. З огляду на це й обрана тема даної бакалаврської роботи, в процесі виконання якої розроблений і створений веб-додаток для підтримки діяльності притулку для тварин зі зручним онлайн-інструментом пошуку з використанням картографічного API-інтерфейсу.

1. ОГЛЯД ТЕМИ ТА ІСНУЮЧИХ РІШЕНЬ

1.1 Огляд проблемної області

Проблема безпритульних тварин є набагато серйознішою і глибшою, ніж здається на перший погляд. Люди звикли до бродячих котів та собак на вулицях міста, деякі з них навіть підгодовують або запускають тварин в під'їзди під час холодної пори року або на ніч. Хтось може вважати бездомного звіря своєю домашньою твариною, беручи його під власний частковий нагляд. Але це може лише погіршити проблему з часом.

Перенаселення людей і хаотичні умови проживання в містах призвели до великої популяції безпритульних тварин не лише в місті Суми, а й по всій Україні, що спричиняє немало проблем. Собаки або коти, які перебувають на вулиці, можуть легко спричинити аварію, коли перетинають проїзджу частину вулиці (адже тварини не дотримуються правил дорожнього руху), а це призводить до травм не лише їх самих, а й людей, які можуть мимоволі стати учасниками ДТП.

Безпритульні тварини зазвичай псують місця перебування та самих себе, адже за ними ніхто не доглядає. Вони можуть потрапити у сад, клумбу, сарай, гараж. Вони можуть зіпсувати газон, особливо під час вологих періодів. Бродячі собаки, наприклад, часто є більш агресивними, непередбачуваними і жорстокими в порівнянні з домашніми, бо такими їх зробили умови вуличного життя. Якщо у бродячих звірів з'явиться бажання полювати, щоб дістати собі їжу, і вони випадково натраплять на город чи домашнє господарство, що належить людям, вони можуть завдати шкоди домашнім тваринам, зокрема кролям, свиням, курям, вівцям або навіть коровам. Якщо бродячий собака натрапляє на сільськогосподарську землю і завдає шкоди худобі, існує ймовірність, що фермери стрілятимуть в нього, оскільки їм це дозволено на законодавчому рівні.

В пошуках їжі безпритульні тварини змушені відкривати пакети зі сміттям, що призводить до того, що сміття розкидається по всьому навколишньому середовищу та вздовж вулиць, а сміттєві баки, в свою чергу, досить легко пошкоджуються, розбиваються або перегортаються.

Також слід згадати, що вуличні звірі найчастіше живуть в підвалах, під'їздах, під навісами та будівлями, намагаючись триматися ближче до людей (бо ті їх, наприклад, годують і пестять), і можуть бути носіями бліх, кліщів та інших паразитів, а також різноманітних хвороб. Окремо хотілося б зазначити проблему поширення сказу, який є вкрай небезпечним захворюванням і при цьому легко передається людям. Сказ з кожним роком збільшує смертність людей, і цей показник продовжує збільшуватися щороку, незважаючи на кроки, які вживаються, зокрема вакцинації пацієнтів та надання первинної медичної допомоги відразу після укусу.

Звісно, не всі тварини дружелюбні до людей, особливо це стосується тих, хто провів усе або більшість свого життя на вулиці в жахливих умовах, і, як наслідок, існує загроза нападу на людину неконтрольованого бездомного собаки. Також бродячі тварини можуть становити загрозу й для домашніх улюбленців під час прогулянок. Безпритульні тварини часто живуть зграями, що може викликати у людей відчуття загрози, а також призвести до неконтрольованого розмноження і небажаних цуценят, що лише поширює і поглиблює проблему. Вони мають тенденцію вити та гавкати, особливо коли відбувається бійка з іншим собакою, що є звичайним явищем. Це може заважати мешканцям навколо, дратувати і навіть призводити до безсоння.

Відповідно до вищесказаного хочеться нагадати, як часто можна побачити безпритульних тварин у місті Суми. Відповідь очевидна: кожного разу при перебуванні на вулиці; кожного дня, а іноді навіть й по декілька разів на день. Багато з них мають нашийники, що є ознакою приналежності до домашніх тварин в минулому, тобто колись ця тварина мала свого власника, проте зараз з якоїсь причини вони втратили контакт. Можливо це створіння взагалі мешкало в іншому районі міста. І звісно, навіть якщо спробувати

допомогти – знайти господаря власноруч, найвірогідніше ця трудомістка робота не принесе ніяких плодів, адже складно знайти потрібну людину з понад 250 тисяч мешканців міста Суми без відповідного інструменту для такого пошуку.

Збір даних від звичайних пересічних громадян, які вони отримують просто подорожуючи вулицями міста, займаючись звичайними справами і просто проживаючи в місті, грає ключову роль в формуванні величезної бази даних загублених і знайдених тварин, і ними, безсумнівно, не можна нехтувати. З їх допомогою з'являється можливість зібрати все, що бачить кожен з мешканців міста, щодня, в конкретні проміжки часу, в кожному районі і вулиці, об'єднати в одному місці, структурувати цю інформацію і оптимізувати її для зручності пошуку. Саме це є запорукою конкретних результатів в вирішенні питання безпритульних тварин.

Деякі сторінки в соціальних мережах намагаються організувати щось схоже, проте це не набирає широкого розповсюдження через відсутність врахування усіх факторів даної проблеми. Наприклад, звичайний перелік об'яв від мешканців без будь-якої фільтрації і сортування незручний для звичайного користувача, який шукає свого чотирилапого друга. В багатьох об'явах не вказаний час, місце, а якщо й вказане, то досить приблизно і з великим радіусом, що ніяк не сприяє швидкому пошуку. Також часто відсутня фотографія тварини або її опис є досить абстрактним, під який попадає більшість собак та котів. Враховуючи всі ці аспекти, знайдену тварину з об'яви майже неможливо ідентифікувати і визначити її конкретне місцезнаходження, і, як наслідок, зусилля людини, яка помітила її на вулиці і вирішила про це повідомити, виявляються марними.

До того ж, єдиною організацією в нашому місті, яка наразі займається вирішенням проблем безпритульних тварин, в тому числі і допомагає з пошуком старих або нових господарів, є Сумське товариство захисту тварин (СТЗТ). Це неприбуткова організація, що існує виключно за рахунок благодійних внесків та не фінансується жодною з державних установ.

Суттєвою допомогою з боку товариства у вирішенні даного питання є розповсюдження інформації на своїх сторінках в соціальних мережах, ведення та оприлюднення обліку загублених та знайдених тварин на веб-сайті. Це особливо важливо в сучасному світі ІТ-технологій, де віртуальна інформація сприймається і розповсюджується набагато швидше, ніж будь-яка інша. Проте це лише перший етап з багатьох на шляху подолання проблеми, адже зручний інструмент, який пришвидшив би процес пошуку, досі не розроблений.

1.2 Огляд подібних рішень

Метою даного дослідження є розкриття потенціалу діджиталізації інструментів збору інформації, які спрямовані на оптимізацію пошуку загублених та знайдених тварин в місті, а також розуміння переваг та недоліків цього процесу; визначення практичного застосування цифрових інструментів в процесах генерації та обміну даними мешканців для подальшого прискорення вирішення питання безпритульності.

Задля цього вирішено порівняти максимально подібне і максимально схоже рішення до розроблюваного веб-сервісу, а саме українській проект HappyPaw. Цей інструмент включає в себе веб-платформу для розміщення та пошуку інформації про загублених і знайдених тварин з можливістю її фільтрації за різноманітними критеріями. На рисунках 1.1-1.3 представлений вигляд головної сторінки веб-сайту, сторінки з пошуком і фільтрацією і картки з анкетою тварини відповідно і зберігаючи порядок переліку.

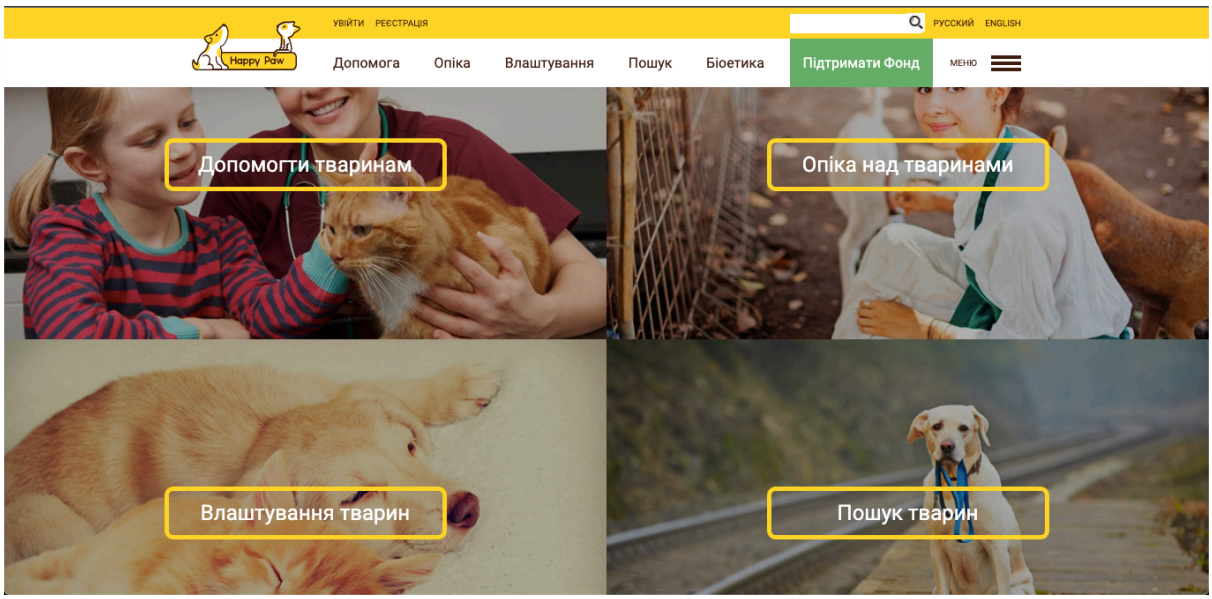

Рисунок 1.1 – Головна сторінка сайту (happypaw.ua)

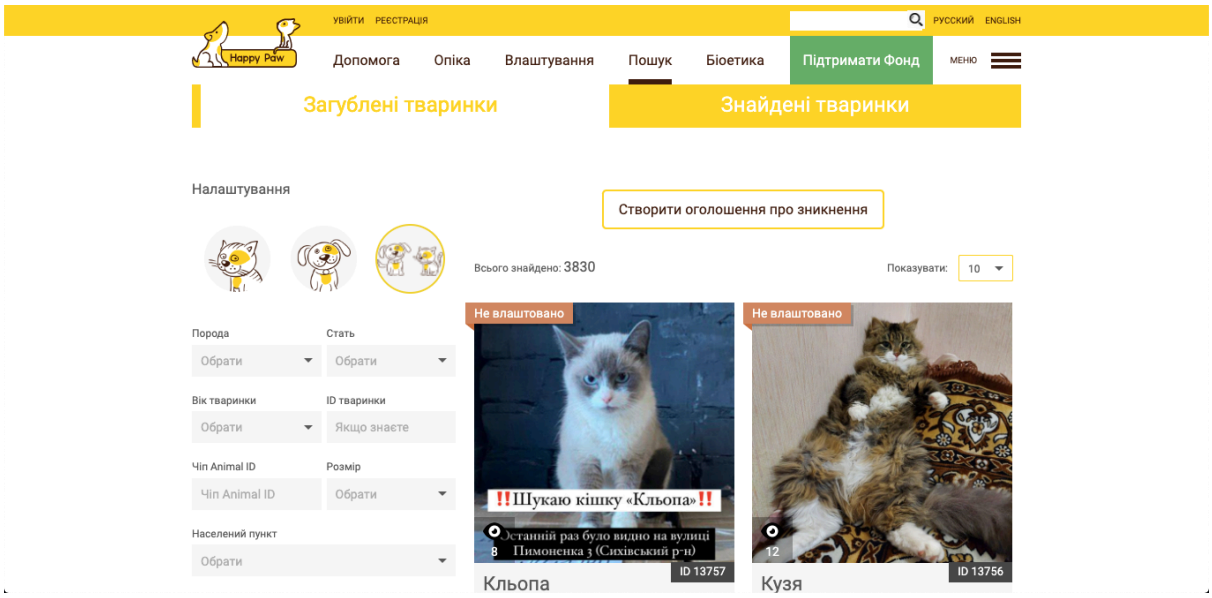

Рисунок 1.2 – Сторінка пошуку з фільтрацією (happypaw.ua)

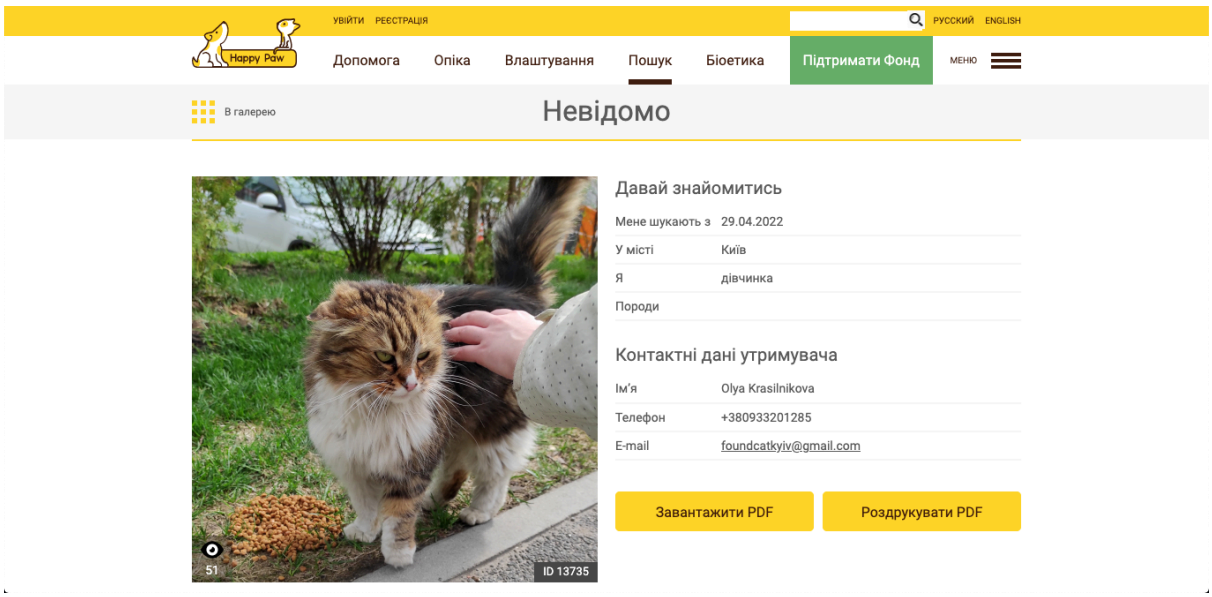

Рисунок 1.3 – Вигляд картки тварини (happypaw.ua)

Цей веб-сайт є чудовим прикладом того, як застосування інформаційних технологій може спростити процес вирішення соціальних проблем і того, що в сучасному світі багато людей віддають перевагу використанню онлайн-засобів задля досягнення своєї мети, а саме нарешті знайти свого зниклого друга або допомогти з пошуком господаря тому, кого знайдено на вулиці. Головна ідея проекту схожа з темою даної бакалаврської роботи: залучити якомога більше громадян до спільного вирішення проблеми безпритульності шляхом збору і розповсюдження інформації за допомогою веб-сервісу, розробленого саме для досягнення цієї мети.

В ході експлуатації проекту Happy Paw проведений детальний аналіз і виділено переваги і недоліки такого вирішення питання оптимізації пошуку безпритульних тварин. Серед переваг хотілося б відмітити наступне:

- додавання власної анкети знайденої або загубленої тварини з детальною інформацією та фотографіями;
- фільтрація анкет за кількома критеріями, а саме: вид тварини, стать, вік, розмір, порода, населений пункт, дата;
- можливість завантажити і роздрукувати об’яву у вигляді PDF-документу всього в один клік;

- оформлення заявки на прихисток або опіку над твариною онлайн без зайвих клопотань;
- надання кожній тварині в базі власного ID, що значно спрощує пошук за умови знання цього номеру;
- ведення та відображення статистики вилікуваних тварин, яка дані якої оновлюються раз на місяць;
- додавання таких статусів, як «потребує допомоги», «під опікою», «повернено», «не влаштовано», що значно спрощує ведення статистики.

Але, як і в будь-якого явища в цьому світі, в проекті Happy Paw можна також помітити й деякі недоліки, а саме:

- відсутність інтерактивної карти, де можна було б наочно побачити місцезнаходження тварини на момент подачі заявки;
- відсутність будь-якої адреси, прив’язаної до кожної анкети взагалі, вказане лише місто і в деяких випадках район, що збільшує радіус пошуку і ускладнює цей і так досить повільний процес;
- застарілий дизайн, а саме: недоречна різнокольоровість сайту, присутність більше ніж трьох шрифтів, відсутність анімації і т. і.
- незначні баги в користуванні: неробочі посилання, неадаптованість сайту під різні роздільні здатності екранів, граматичні помилки і т. і.
- наявність кнопки «забери мене додому», яка не має практичного застосування і грає лише роль переадресації на розділ з номером телефону;
- використання пошуку на сайті за ключовими словами призводить до появи реклами з інших джерел перед виведенням посилань на блоки з необхідними результатами;
- необхідність реєстрації або авторизації для того, щоб подати заявку про знайдену або загублену тварину, що уповільнює процес і призводить до зниження коефіцієнту зацікавленості користувача в експлуатації сайту (адже не всі виявлять бажання в реєстрації акаунту лише для того, щоб повідомити про знайденого песика);

- сайт інтуїтивно не зрозумілий, елементи розташовані досить хаотично, без чіткої стандартизованої структури, наприклад наявність двох стрічок меню, одне з яких – горизонтальне, інше – так зване «гамбургер» або «трирядкове», що ускладнює пошук потрібного розділу.

Аналіз вже існуючих подібних інструментів полегшує формування вимог до проекту і формулювання завдання на розробку, адже можна наочно побачити, які з рішень є зручними для користувача, і покращують досвід користування продуктом, а яких краще оминати, щоб не припуститися тих самих помилок, що й конкуренти. Найбільшою перевагою проекту Happy Paw є спрощення збору інформації від громадян та її структуризація, а найсуттєвішими недоліками, в свою чергу, є витрачання великої кількості часу на реєстрацію, авторизацію і відправку заявки, а також відсутність точної адреси в анкетах тварин (що і є ключовим фактором пришвидшення процесу пошуку) і будь-якої інтерактивності у вигляді мап, анімації тощо.

В процесі створення проекту для бакалаврської роботи враховані і збережені всі переваги, викладені вище, і усунені всі недоліки за допомогою сучасних дизайнерських рішень та використання картографічного API-інтерфейсу. Також прибрано процес авторизації для подання заявки і оптимізовано форму для її заповнення задля максимально швидкого і зручного збору інформації від користувача і отримання тих самих результатів за менший проміжок часу.

1.3 Актуальність створення веб-сервісу

В сучасному світі стрімкого розвитку інформаційних технологій, особливо що стосується публікації, розповсюдження і швидкої обробки даних будь-якого характеру людьми за допомогою використання мережі Інтернет, кожному бізнесу просто необхідно змістити фокус уваги на просування своїх послуг у віртуальному середовищі. На сьогоднішній день людина схильна надавати перевагу пошуку рішення своїх побутових та інших проблем онлайн

ніж звертатися до застарілих способів, адже це значно пришвидшує процес, надаючи більший спектр вибору рішень і заощаджуючи час (адже уся інформація зібрана в одному середовищі, і нею можна скористатися за допомогою лише одного пристрою з доступом в Інтернет) і тим самим забезпечуючи вищі показники ефективності та результативності.

Відомою цитатою Білла Гейтса є наступне висловлювання: «If your business is not on the Internet then your business will be out of business», що в перекладі означає «Якщо ваш бізнес не існує в інтернеті, ваш бізнес не існує взагалі», що безсумнівно, є вірною істиною нашого часу. [1] Сайт Beam Local для реклами своїх послуг використовує статистичні дані: 81% людей досліджують бізнес чи послугу в Інтернеті, перш ніж прийняти рішення про покупку. [2] Тож якщо у певного бізнесу немає веб-сайту або хоч якоїсь інформації в мережі Інтернет, то немає й шансів захопити частку цього ринку. Наявність веб-сайту неодмінно залучить нових клієнтів до бізнесу та, в свою чергу, принесе більший дохід. Веб-сайт може бути доступний 24/7/365, навіть в той час, коли ніхто з представників компанії не може бути залучений. Веб-сайт діє як «завжди активний» діловий адвокат у робочий час і поза ним.

Споживач 21 століття – скептик. За даними статистичного опитування сайту Beam Local 56% людей насправді заявили, що вони не довірятимуть бізнесу без веб-сайту. [2] Наявність інтуїтивно зрозумілого й інформаційно наповненого веб-сайту миттєво підвищує довіру до компанії як законного бізнесу, адже таким чином стає зрозумілим, що власники і співробітники не намагаються нічого приховати. Чим більш розгорнутою і повною буде інформація про діяльність і послуги компанії, тим, перш за все, користувачу буде простіше орієнтуватися і буде більш зрозумілим, на що орієнтований бізнес і чому треба обрати саме його, а по-друге, більшим буде рівень довіри, адже насьогодні важко щось приховати, якщо інформація вже існує в мережі Інтернет.

На відміну від загальноприйнятної точки зору, веб-сайти насправді є досить економічно вигідними. Вони пропонують кращу рентабельність

інвестицій, ніж будь-яка інша форма реклами. Якщо бізнес планує рекламувати свою діяльність, створення веб-сайту має бути найпершим пріоритетом. Немає необхідності володіти спеціальною технікою, щоб отримати веб-сайт. Багато компаній з веб-розробки мають прості процеси реєстрації, які роблять процес швидким і безболісним. З часом відносно низькі інвестиції в створення бізнес-сайту принесуть дивіденди, залучивши нових клієнтів і створивши бренд. Крім того, це найбільш законні витрати бізнесу та списання податків.

Перші враження користувача враховуються і мають вплив на його подальші дії більше, ніж будь-які інші, і саме веб-сайти приходять на допомогу в цьому процесі і дозволяють справити найточніше і найвигідніше для бізнесу враження завдяки добре продуманій і розробленій структурі, інтерфейсу і дизайну. Веб-сайт може допомогти продемонструвати досвід компанії і краще позиціонувати бізнес, висвітлюючи його під найкращим кутом. Саме він може бути інструментом номер один, щоб виділитися з натовпу конкурентів.

Наявність веб-сайту суттєво заощаджує час. Незалежно від того, чи комунікація відбувається через електронну пошту, приймання дзвінків чи розсилку пропозицій – спілкування з потенційними клієнтами вимагає часу. Веб-сайт може зіграти ключову роль в його економії, надаючи відповіді на поширені запитання та запити клієнтів, а отже співробітники і власники бізнесу можуть зосередитися і витрачати свій час на більш цінні справи.

Веб-сайт діє як платформа для обміну повідомленнями та формує сприйняття бізнесу в Інтернеті таким чином, який не можуть продемонструвати і надати інші канали або сторінки соціальних мереж. Саме платформа дозволяє позиціонувати себе на ринку, щоб отримати саме той тип клієнтів, на яких націлений бізнес і в процесі пошуку яких він знаходиться. Багато власників отримують зворотній зв’язок та відгуки користувачів в Інтернеті, іноді навіть несвідомо. За даними сайту Beam Local 79% користувачів, які читають огляди в Інтернеті, вірять їм. Тож створення веб-

сайту є гарним рішенням в боротьбі з великою кількістю негативу або критики. [2]

Веб-сервіси – це концепція створення таких додатків, які можна використовувати за допомогою стандартних протоколів Інтернет. В сучасному світі цю концепцію застосовують і розвивають багато провідних компаній в IT-області. Концепція веб-сервісів реалізується за допомогою ряду технологій, які стандартизовані World Wide Web Consortium (W3C). [3] Якщо ж надавати визначення поняттю «веб-сервіс», то найточнішим стислим варіантом є таке формулювання: сервер, що працює на комп'ютерному пристрої, прослуховуючи запити на певному порті через мережу і обслуговує веб-документи, такі як HTML, JSON, XML і зображення. Веб-сервіси, на відміну від веб-сайтів, не залежать від платформи, оскільки вони використовують відкриті протоколи. Веб-сайти, як відомо, є кросплатформними, оскільки вимагають певних налаштувань для роботи в різних браузерах, операційних системах тощо. Тож це є суттєвою перевагою веб-сервісів.

Взагалі, використання приставки «веб» в понятті «веб-сервер» насправді є не досить коректним, але розповсюдженим явищем. Веб-сервер, в свою чергу, не використовує людський інтерфейс користувача, що працює в Інтернеті, (так звану Всесвітню павутину (WWW)), а скоріше міжмашинний сервер, який працює за допомогою протоколів WWW.

Веб-сервери для передачі машиночитаних форматів файлів, таких як XML і JSON використовують таку веб-технологію, як HTTP. На практиці веб-сервіс зазвичай надає об'єктно-орієнтований веб-інтерфейс серверу бази даних, який використовується, наприклад, іншим веб-сервером або мобільним додатком, який, в свою чергу, надає кінцевому користувачеві зручний і зрозумілий інтерфейс. Багато організацій, які надають дані у вигляді форматованих HTML-сторінок, також нададуть ці дані своєму серверу у форматі XML або JSON (насьогодні частіше використовується другий варіант), часто саме через веб-сервер, щоб дозволити синдикацію. Іншим варіантом цього процесу може бути mashup, де сервер використовує кілька

веб-сервісів на різних машинах і компілює вміст в один кінцевий інтерфейс користувача. [4]

Розробники концепції веб-сервісів пропонують такі сценарії їх застосування на практиці:

1. Веб-сервіси як реалізація логіки програми (бізнес-логіки). Тобто створення нового додатка, бізнес-логіка якого реалізується у веб-сервісі. (Рис. 1.4)

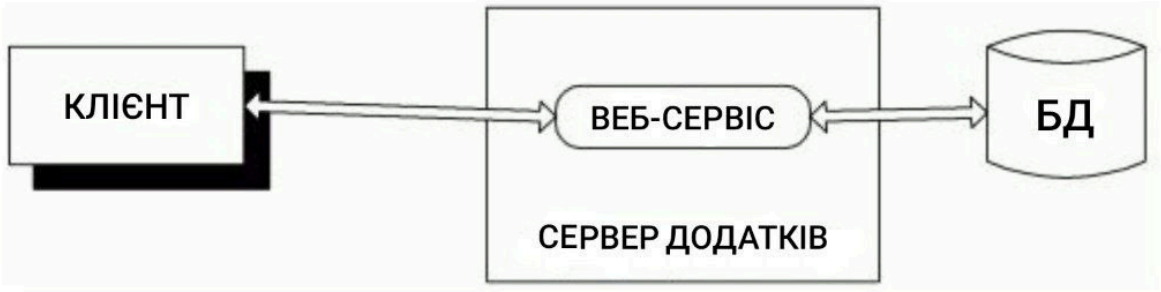

Рисунок 1.4 – Концепція роботи веб-сервісу як реалізації логіки

2. Веб-сервіси як інтеграції. Тобто використання веб-сервісу як способу доступу віддалених клієнтів до внутрішньої ІС компанії, або для організації взаємодії компонента (наприклад, EJB, COM-компонента) з різними віддаленими клієнтами. (Рис. 1.5)

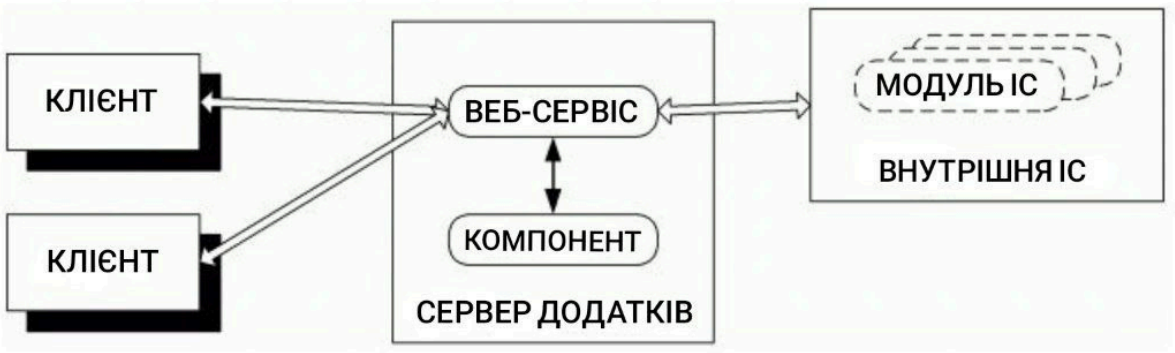

Рисунок 1.5 – Концепція роботи веб-сервісу як інтеграції

3. Використання веб-сервісу як будівельного блоку під час створення програми. Програма може використовувати веб-сервіси як віддалені компоненти, які надають певну функціональність. Існують різні сервіси, які надають якісне вирішення таких завдань як

автентифікація, ведення календаря, надсилання повідомлень, пошук, переклад тощо. (Рис. 1.6)

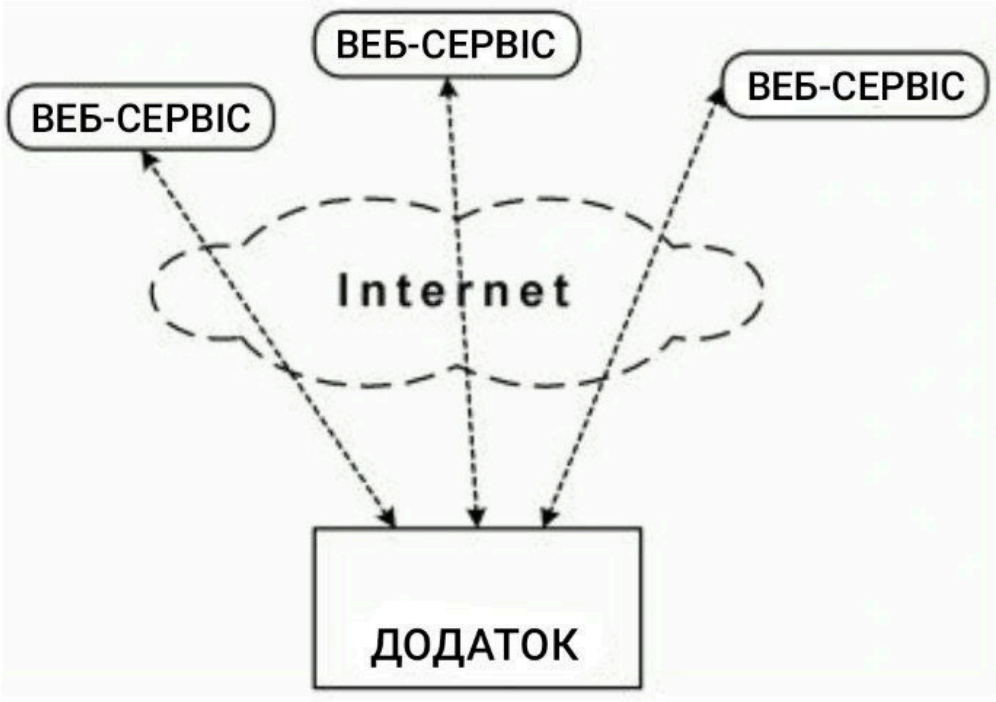

Рисунок 1.6 – Концепція роботи веб-сервісів як віддалених компонентів

Крім вищерозглянутих концепцій, можливі також й інші варіанти використання веб-сервісів. Наприклад, існують дослідження з використання веб-сервісів для побудови розподілених обчислювальних та інформаційних систем, однорангових та зі складною ієрархічною структурою. [5]

Сучасні бізнес-додатки використовують різноманітні платформи програмування для розробки веб-додатків. Деякі програми можуть бути розроблені на Java, інші в .Net, а деякі інші на Angular JS, Node.js тощо. Найчастіше ці різнорідні програми потребують певного зв’язку між ними. Оскільки вони створені з використанням різних мов розробки, стає дуже важко забезпечити точне спілкування між додатками. [6]

Тут на допомогу приходять веб-сервіси. Веб-сервіси забезпечують загальну платформу, яка дозволяє багатьом додаткам, створеним на різних мовах програмування, мати можливість спілкуватися один з одним. Оскільки вся комунікація здійснюється в XML, веб-сервіси не прив’язані до жодної

операційної системи чи мови програмування – Java може взаємодіяти з Perl; програми Windows можуть взаємодіяти з додатками Unix. [7]

Веб-сервіси використовують стандартизований промисловий стандартний протокол для зв'язку. Усі чотири рівні (Service Transport, XML Messaging, Service Description та Service Discovery) використовують чітко визначені протоколи в стеку протоколів веб-сервісів. Така стандартизація стеку протоколів дає бізнесу багато переваг, таких як широкий вибір, зниження вартості через конкуренцію та підвищення якості.

Також слід зазначити, що веб-сервіси використовують протокол SOAP через HTTP, тому є можливість використовувати наявний недорогий Інтернет для впровадження. Це рішення набагато дешевше в порівнянні з власними рішеннями, такими як EDI/B2B. Крім SOAP через HTTP, веб-сервіси також можуть бути реалізовані й на інших надійних транспортних механізмах, таких як FTP. [8]

Отже, узагальнюючи все вище викладене, можна зробити висновок, що кожному бізнесу сьогодення варто попіклуватися про розміщення інформації про свою діяльність в онлайн-просторі і гарним помічником в цьому процесі може стати веб-сервіс. А для такої організації, як неприбуткове громадське формування, яке фінансується виключно благодійними внесками небайдужих людей і взагалі не підтримується жодною з державних установ, це взагалі перша необхідність, адже наявність і просування структурованого і функціонального веб-сайту у віртуальному інформаційному просторі приверне значно більше уваги потенційних спонсорів і сформує навколо діяльності товариства скупчення зацікавлених у допомозі людей. Саме це і є метою створення даного бакалаврського проекту.

1.4 Постановка задачі

Метою виконання бакалаврської роботи є проектування і розробка веб-сервісу для підтримки діяльності притулку для тварин (Сумського товариства

захисту тварин), який міститиме візуальну (Front-end) та серверну (Back-end) частини і матиме наступний функціонал:

- перегляд на карті усіх тварин з бази даних товариства з виведенням їх місцезнаходження у вигляді маркерів;
- перегляд розгорнутої анкети кожної окремої тварини при натисканні на відповідний маркер на карті;
- створення нової заявки про загублену або знайдену тварину шляхом заповнення форми і надання інформації про звіря;
- панель адміністрування з реалізацією редагування і керування доданими заявками в базі даних.

В веб-частині мають бути присутніми наступні елементи:

- хедер з логотипом, назвою товариства і фіксованим горизонтальним головним меню з можливістю перемикання між розділами;
- «Головна» – лендінг-розділ зі стислим описом діяльності Сумського товариства захисту тварин;
- «Про нас» – перелік послуг товариства у вигляді карток з коротким описом;
- «Пошук тварин» – розділ реалізації головного функціоналу, який поділяється не декілька блоків:
    - інструкція і опис роботи сервісу для пошуку із спонуканням заповнити форму і створити нову заявку;
    - «Нова заявка» – вкладка, яка містить мапу, на якій необхідно обрати місцезнаходження тварини і форму для створення нової заявки про загублену чи знайдену тварину;
    - «Тварини» – вкладка, яка містить мапу з відображенням усіх тварин з бази даних з їх місцезнаходженням у вигляді маркерів та спливаючою анкетою кожної окремої тварини при натисканні на відповідний маркер.
- футер з адресою розташування, контактами і посиланнями на соціальні мережі товариства.

- панель адміністратора з авторизацією та можливістю перегляду, редагування і видалення існуючих анкет і роботою з базою даних.

Створити програмний інтерфейс додатку для взаємодії веб-частини з сервером через HTTP-запити (POST та GET). Використати картографічний API для створення інтерактивної карти перегляду всіх наявних анкет і створення нової заявки.

2. МЕТОДИКА ВИРІШЕННЯ ПОСТАВЛЕНИХ ЗАДАЧ

2.1 Вибір засобів програмування

В ході навчання в Сумському державному університеті за спеціальністю «Інформатика» розглянуто, вивчено і застосовано на практиці ряд різноманітних сучасних інформаційних технологій і методів їх створення. Ці знання та навички значно спростили процес вибору засобів створення проекту бакалаврської роботи.

Мова програмування JavaScript.

Вибір мови програмування є одним з перших та ключових етапів в процесі розробки та створення веб-сервісу. Хтось обирає мову лише з особистих уподобань, хтось тому, що знає лише її, а хтось навіть не замислюється над цим. Проте, не слід забувати, що даний етап розробки є дуже важливим, тому що саме через це в майбутньому процесі реалізації можуть виникнути проблеми, а можуть і не виникнути, дивлячись як підійти до питання. В сучасному світі існує безліч різних мов програмування, які відрізняються між собою способом компіляції, екосистемою і спільнотою, платформою, попитом, і перш за все, призначенням. Тож, першочерговим завданням є чітке формулювання мети і завдання, яке ставить замовник. З огляду на специфіку сфери розробки веб-сервісів і того, що даний проект має містити в собі як візуальну (клієнтську) частину, так і серверну, вирішено зупинити свій вибір на такій мові програмування, як JavaScript.

Коли людина в чомусь новачок, вона зазвичай шукаєте визначення і потрапляє у Вікіпедію. У випадку з JavaScript після прочитання статті може виникнути більше запитань, ніж було до цього. Можна побачити багато слів про те, що це мова програмування на стороні клієнта, і це дійсно так. [9] Але це, в той самий час, доволі зменшено відносно до того, чим насправді JavaScript є сьогодні. Наразі скажімо просто, що JavaScript є інструментом для

розробників і використовується для того, щоб додати веб-сайтам інтерактивність.

З появою JavaScript стало можливим задовольнити різну аудиторію: як авторів компонентів і професіоналів корпоративного рівня, так й сценаристів і дизайнерів. Цю другу групу можна також назвати, кажучи сучасними термінами, розробниками інтерфейсу.

JavaScript – це мова сценаріїв, яка вставляється безпосередньо в HTML-сторінки. Це єдина мова програмування такого роду, яку можуть зрозуміти веб-браузери. Браузери можуть з легкістю читати, інтерпретувати її, а потім запускати програму. Саме вона дозволяє користувачам взаємодіяти з веб-сторінками. Поява цієї мови принесла собою динамічні функції в Інтернет. Зараз практично немає обмежень для того, що можна робити за допомогою JavaScript на веб-сторінці – ось лише кілька прикладів, які реалізовані в даному бакалаврському проекті:

- ефекти прокрутки, фіксовані або випадаючі меню;
- анімація елементів сторінки, таких як зміна розміру або переміщення;
- валідація форми (перевірка введення даних в поля і їх відповідність);
- зміна коліру кнопки, коли миша наводить на неї курсор;
- відображення тостерів (спливаючих повідомлень про помилки);
- відображення та приховання додаткової інформації одним натисканням кнопки;
- завантаження нового вмісту або даних на сторінку без її перезавантаження;
- та багато іншого, що розглянуте в наступних розділах.

Тож наразі стає зрозумілим, що JavaScript дозволяє створювати високочутливі інтерфейси, які покращують користувацький досвід та забезпечують динамічну функціональність, не чекаючи, поки сервер відреагує та покаже іншу сторінку. [10]

Звісно, сьогодні JavaScript не може існувати окремо від мови розмітки для перегляду веб-сторінок HTML та її невід’ємного доповнення – мови опису

зовнішнього вигляду документу CSS та фрейморків, таких як, наприклад, React, Angular, Vue, JQuery або Node.

Мова розмітки HTML.

Звичайно, мови розмітки відрізняються від мов програмування. У той час, як мови програмування допомагають нам змінювати дані, мови розмітки допомагають визначити, як елементи повинні відображаються на веб-сторінці. Мало хто знає, що HTML – це насправді абревіатура, яка розшифровується як HyperText Markup Language. HTML-код забезпечує правильне форматування тексту та зображень для Інтернет-браузера. Без цієї мови розмітки браузер просто не знав би, як відображати текст, елементи або як завантажувати зображення чи інший вміст. HTML також забезпечує базову структуру сторінки, на яку вже згодом накладаються каскадні таблиці стилів, щоб змінити і вдосконалити зовнішній вигляд. HTML має просту текстову структуру, яку легко вивчити та зрозуміти початківцям. [11]

Мова опису зовнішнього вигляду CSS.

В теорії, веб-сайти, звісно, можуть працювати і без CSS (Cascading Style Sheets), але на практиці, це, перш за все, просто некрасиво, а по-друге, складно для сприймання користувачем. CSS робить інтерфейс веб-сайту яскравим і формує чудовий досвід від експлуатації. Без CSS веб-сайти були б менш приємними для ока і набагато складнішими для навігації. Окрім макету та формату, CSS відповідає за колір, розмір шрифтів, відступи, відображення елементів та багато іншого. Хотілося б розглянути переваги використання каскадних стилів більш детально:

1. Підвищення швидкості обробки сторінок. Як відомо, чим більше коду, тим повільніша швидкість обробки сторінки інтерпретатором. CSS, в свою чергу, дозволяє використовувати менше коду і використовувати лише одне правило CSS, застосовуючи його до всіх входжень певного тегу в HTML-документ.
2. Кращий користувальницький досвід. CSS не тільки робить веб-сторінки зручними і привабливими для людського ока, але також

дозволяє зручне форматування. Коли кнопки та текст розташовані в логічних місцях і добре організовані, покращується взаємодія веб-сайту з користувачем.

3. Швидший час розробки. За допомогою CSS можна застосувати певні правила форматування та стилі до кількох сторінок за допомогою всього лише одного рядка коду. Одну каскадну таблицю стилів можна відтворити на кількох сторінках веб-сайту. Якщо, наприклад, є сторінки продуктів, які мають однакове форматування, зовнішній вигляд і сприйняття, написання правил CSS для однієї сторінки буде достатнім для всіх сторінок того ж самого типу.
4. Прості зміни форматування. Якщо потрібно змінити формат певного набору сторінок, це легко зробити за допомогою CSS. Немає необхідності виправляти кожну окрему сторінку. Просто треба відредагувати відповідну таблицю стилів CSS, і можна побачити, що внесені зміни будуть застосовані до всіх сторінок, які використовують цю таблицю стилів.
5. Сумісність між пристроями. Адаптивний веб-дизайн вкрай важливий у час стрімкого розвитку і створення електронних пристроїв різного формату. Веб-сторінки мають бути повністю видимими та легко навігаційними на всіх видах пристроїв, будь то мобільний телефон чи планшет, настільний комп’ютер або навіть смарт-телевізор. CSS поєднується з HTML, щоб зробити адаптивний дизайн можливим і легким в розробці. [12]

Формат передачі даних JSON.

Для передачі даних в розроблюваному веб-сервісі використаний формат JSON (скорочено від JavaScript Object Notation) – відкритий стандартний формат файлу та формат обміну даними, який використовує зрозумілий людині текст для зберігання та передачі об’єктів даних, що складаються з пар атрибут-значення та типів даних масиву. Він широко використовується в

Інтернеті майже для кожного доступного на сьогодні API, а також для файлів конфігурації і роботи таких систем, як ігри та текстові редактори.

Можна сміливо сказати, що нині JSON заволодів світом. Сьогодні, коли будь-які дві програми спілкуються одна з одною через Інтернет, велика ймовірність, що вони роблять це за допомогою JSON. Його прийняли всі великі корпорації: з десяти найпопулярніших API, що пропонуються такими великими компаніями, як Google, Facebook і Twitter, лише один API працює з даними в форматі XML, тоді як всі інші мають справу з JSON. Twitter, як ілюстративний приклад з цього списку, підтримував XML до 2013 року, коли він випустив нову версію свого API, який відмовився від XML на користь використання виключно JSON. Історія та еволюційний шлях Інтернету зіграли значну роль у популяризації JSON. Згідно зі Stack Overflow, зараз про JSON задається більше питань, ніж про будь-які інші формати обміну даними. [13]

Проте, слід зазначити, що XML не є єдиною альтернативою JSON – деякі люди зараз використовують такі технології, як YAML або буфери протоколу Google. Але вони навіть і близько не так популярні, як JSON. На даний момент JSON, здається, є основним форматом для спілкування з іншими програмами через Інтернет.

В даному проекті також використаний JSON, з огляду на те, що він надзвичайно легкий для надсилання і отримання назад даних у HTTP-запитах і відповідях через невеликий розмір файлу. Його легко читати в порівнянні з чимось на кшталт XML, оскільки він набагато простіший, зрозумілий навіть новачкам і не має так багато відкриваючих і закриваючих тегів, які можуть збити з пантелику.

JSON також дуже добре інтегрується з JavaScript, оскільки JSON – це лише його підмножина, а це означає, що все, що написано в JSON, є дійсним в JavaScript. Майже кожна основна мова має певну форму бібліотеки або вбудовану функціональність для розбору рядків JSON на об’єкти або класи цієї мови. Це робить роботу з даними JSON надзвичайно легкою в інтеграціях майже з будь-якою мовою програмування. [14]

Програмна платформа NodeJS.

Джефф Харрелл, технічний директор PayPal сказав наступне: «Node.js powers our web applications and has allowed our teams to move much faster in bringing their designs to life», що в перекладі означає: «Node.js підтримує наші веб-додатки і дає змогу нашим командам рухатися набагато швидше у втіленні своїх проектів в життя». [15]

Node.js за своєю суттю є програмною платформою з відкритим кодом для виконання коду JavaScript за межами веб-браузера, яке працює на движку V8. Раніше JavaScript використовувався лише для обробки даних в браузері користувача, тоді як з появою Node.js з'явилася можливість виконувати скрипти вже на сервері та відправляти користувачеві результат їхнього виконання. Платформа Node.js перетворила JavaScript на мову загального використання. Отже, Node.js являє собою парадигму «JavaScript всюди», що об'єднує розробку веб-додатків навколо однієї мови програмування, а не різних мов для сценаріїв окремо на стороні сервера та клієнта. [16] Слід відмітити, що поширеною помилкою серед розробників є уявлення про те, що Node.js – це бекенд-фреймворк і використовується лише для створення серверів. Це неправда: Node.js можна використовувати як на інтерфейсі, так і на сервері.

Node.js швидко набрав популярності за останні кілька років. Це завдяки широкому переліку переваг, якими він володіє:

1. Легкість – це найкращий вибір для початківців у сфері веб-розробки, адже з великою кількістю навчальних посібників і великою спільнотою розпочати роботу досить легко.
2. Масштабованість – ця платформа забезпечує широку масштабованість для додатків, адже, будучи однопотоковим, здатна обробляти величезну кількість одночасних з'єднань з високою пропускною здатністю.
3. Швидкість – виконання потоку без блокування робить Node.js ще швидшим і ефективнішим.

4. Вибір пакетів – Node.js має доступний великий набір пакетів з відкритим кодом, які можуть значно спростити роботу розробників. Сьогодні в екосистемі NPM є понад мільйон пакетів.
5. Потужний бекенд – Node.js написаний на C і C++, що робить його швидким і додає такі функції, як, наприклад, підтримка мережі.
6. Кросплатформенна підтримка дозволяє створювати веб-сайти SaaS, десктопні програми і навіть мобільні додатки, використовуючи Node.js.
7. Зручність – Node.js є простим вибором для розробників, оскільки інтерфейсом і бекендом можна керувати за допомогою лише однієї мови – JavaScript.

За останні 2 десятиліття відбулося величезне зростання веб-сайтів, і, як очікувалося, Node.js також швидко зростає. Популярне середовище виконання вже перевищило поріг завантаження в 1 мільярд ще у 2018 році, і, за даними W3Techs, Node.js використовується 1,2% усіх веб-сайтів у всьому світі. Це понад 20 мільйонів сайтів в Інтернеті. [17] Не дивно, що це також популярний вибір багатьох компаній. Ось кілька популярних з них, які сьогодні використовують Node.js: Twitter, Spotify, eBay, Reddit, LinkedIn, PayPal, Yahoo, Netflix, The Mail Online та Walmart.

Хотілося б розібрати детальніше, в чому саме полягає перевага Node.js перед іншими платформами і розібратися з тим, як саме організований процес його роботи. Перш за все, Node.js підтримує обмежений пул потоків для обслуговування запитів. Щоразу, коли надходить запит, Node.js поміщує його в чергу. Тепер з’являється однопоточний «цикл подій» – основний компонент. Цей цикл подій чекає на запити необмежено довго. Коли надходить запит, цикл забирає його з черги і перевіряє, чи потрібна для нього операція блокування введення/виводу (I/O). Якщо ні, він обробляє запит і надсилає відповідь. Якщо запит має виконати операцію блокування, цикл подій призначає потік із внутрішнього пулу потоків для обробки запиту. Цикл подій

відстежує запити блокування та розміщує їх у черзі після обробки завдання блокування. Таким чином він зберігає свою неблокуючу природу.

Оскільки Node.js використовує менше потоків, він використовує менше ресурсів/пам’яті, що призводить до швидшого виконання завдань. Отже, для цілей даного проекту така однопоточна архітектура еквівалентна багатопоточній архітектурі. Коли потрібно обробляти завдання з інтенсивним використанням даних, використання багатопоточних мов має набагато більше сенсу. Але для додатків реального часу Node.js є очевидним вибором. [18]

Фреймворк Express.js.

Фреймворки JS є різноманітними бібліотеками програмування JavaScript, в яких є попередньо написаний код для використання у стандартних функціях та завданнях програмування. Це основа для створення веб-сайтів або веб-додатків. Теоретично, звісно процес кодування можливий і без їх використання, але правильно підібране середовище може значно полегшити роботу. Крім того, всі фреймворки безкоштовні і з відкритим вихідним кодом, тому розробник не має ніяких ризиків.

Насамперед це підвищує продуктивність. Можна розглядати фреймворки як своєрідний обхідний шлях: вони зменшать кількість необхідного для написання вручну коду, адже мають вже заздалегідь написані і готові до використання функції та шаблони. Деякі компоненти веб-сайту не мають бути виготовлені індивідуально за замовленням, тому можна створювати та розширювати вже заздалегідь створені компоненти. Фреймворки більш адаптовані під дизайн веб-сайтів, і тому більшість розробників віддають перевагу саме їм.

Node.js, як вже зазначено раніше, – це швидке середовище виконання JavaScript, яке використовується для створення додатків на стороні сервера, але воно не знає, як виконувати обслуговування файлів, обробляти запити та обробляти методи HTTP, тому тут на допомогу приходить один з найпопулярніших на сьогоднішній день фреймворків Express.js. Цей фреймворк розроблений для швидкого створення кросплатформних

мобільних додатків, API для веб-програм і полегшення роботи вузлів JavaScript. Він використовується для створення односторінкових, багатосторінкових та гібридних веб-додатків. Це шар, надбудований над Node.js, який допомагає керувати серверами та маршрутами і який використовується для полегшення створення API та веб-додатків. Це заощаджує багато часу на кодування і в той самий час робить веб і мобільні додатки більш ефективними. Найочевиднішими перевагами використання Express.js є:

1. Швидка розробка на стороні сервера.
2. Проміжне програмне забезпечення – обробник запитів, який має доступ до циклу запиту-відповіді програми.
3. Маршрутизація. Даний пункт стосується того, як URL-адреси кінцевої точки програми відповідають на запити клієнта.
4. Шаблонування. Express.js надає механізми шаблонів для створення динамічного вмісту на веб-сторінках шляхом створення шаблонів HTML на сервері.
5. Налагодження. Express.js робить процес дебагу простішим, оскільки визначає точну частину, в якій є помилки.
6. Express.js швидко зв’язує веб-сайт з базами даних, як реляційними, так і нереляційними (наприклад, MongoDB).
7. Express дозволяє динамічно відображати HTML-сторінки на основі передачі аргументів шаблонам. [19]

Модуль Mongoose.

Більшість інтерпретують Mongoose як фреймворк, але більш коректним формулюванням визначення Mongoose є бібліотека моделювання об’єктних даних (ODM) для MongoDB і Node.js. Він керує зв’язками між даними, забезпечує перевірку схеми та використовується для обміну між об’єктами в коді та представлення цих об’єктів у MongoDB (база даних документів NoSQL без схем і з можливістю зберігати документи в раніше зазначеному форматі JSON). Слід вказати декілька переваг використання модуля Mongoose:

1. Можна легко зробити перевірку колекції бази даних MongoDB.
2. Попередньо визначену структуру можна реалізувати безпосередньо в колекції.
3. Можна застосувати обмеження до документів колекцій.
4. Модуль побудований на основі драйвера MongoDB і забезпечує легку реалізацію необхідних запитів.

Багато розробників, які звикли використовувати SQL, відчувають себе незручно під час роботи з MongoDB, оскільки це нереляційна база даних і вона має гнучку структуру. Тож Mongoose відіграє важливу роль і робить схему збору схожою на бази даних SQL і тим самим робить процес переходу від одного типу бази даних до іншого менш складним. Після визначення схеми Mongoose дозволяє створити модель на основі певної схеми. Потім модель Mongoose зіставляється з документом MongoDB за допомогою визначення схеми моделі. Після того, як визначено схеми та моделі, можна використовувати безліч різних функцій, які дозволяють перевіряти, зберігати, видаляти та запитувати дані за допомогою звичайних функцій MongoDB. [20]

Для наочного розуміння процесу роботи усіх вищевикладених в цьому розділі інструментів, на рисунку 2.1 відображено концепцію взаємодії між Node.js та MongoDB за допомогою використання модулю Mongoose.

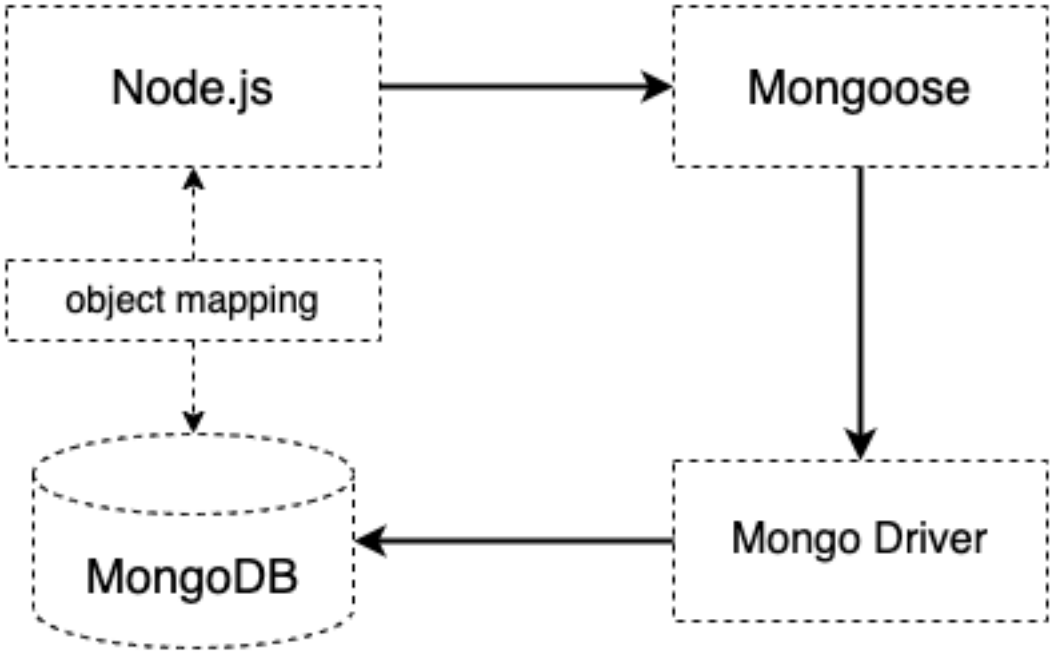

Рисунок 2.1 – Взаємодія Node.js і MongoDB через модуль Mongoose

Картографічна бібліотека Leaflet.

Головним інструментом створення даного бакалаврського проекту є використання картографічного API, що відображено навіть в темі роботи, тож важливою задачею є вибір бібліотеки для досягнення цієї мети. В сфері відображення мап на веб-сайтах безумовними лідерами в Україні є Google Maps та Leaflet, тож вибір стояв саме між цими варіантами. Після порівняльного аналізу, результати якого відображені в таблиці 2.1, вибір на користь другої бібліотеки стає доволі очевидним.

Бібліотеку для відображення мап на веб-сайтах Leaflet 11 років тому створив українець Володимир Агафонкін. Вона розроблена з урахуванням простоти, продуктивності та зручності використання й ефективно працює на всіх основних платформах для десктопних і мобільних пристроїв. Також її можна розширити за допомогою великої кількості плагінів. Leaflet має привабливий, простий у використанні та добре документований API, а також простий, читабельний вихідний код.

Таблиця 2.1 – Порівняльний аналіз бібліотек

Критерій/Бібліотека	**Leaflet**	**Google Maps**
Безкоштовність	+	-
Простота встановлення	+	-
Вибір налаштувань	+	-
Розповсюдженість	-	+

Leaflet досить легко інтегрується в проект шляхом звичайного додавання посилання на CSS-документ в заголовок і посилання на JavaScript-файл у кінець. Це абсолютно безкоштовна, на відміну від Google Maps, де багато функцій і розширень є платними, бібліотека, яка без проблем працює на багатьох платформах, будь то комп’ютер чи мобільний пристрій. Також вона має широкий вибір різноманітних плагінів для підключення і налаштування під конкретні цілі кожного окремого проекту.

Середа розробки Visual Studio Code.

Після того, як обрано мову програмування, програмну платформу, необхідні фреймворки та додаткові бібліотеки, настає етап вибору IDE (Integrated Development Environment – інтегроване середовище розробки), яка б мала зручний і зрозумілий користувачеві інтерфейс, надавала змогу працювати з усіма технологіями одразу і коректно інтерпретувала написаний програмний код. В ході написання даної роботи використаний Visual Studio Code. Далі хотілося б розповісти детальніше про цю середу розробки і обґрунтувати такий вибір.

У 2016 році Visual Studio Code зайняв 13-е місце серед найпопулярніших інструментів розробки на Stackoverflow. Маленькому редактору коду не знадобилося багато часу, щоб досягти місця №1 згідно з опитуванням розробників 2019 року, коли ним користувалися 50% з 87 317 респондентів. [21]

Visual Studio Code виглядає як звичайний редактор коду. Однак є кілька відмінних функцій, які віддають перевагу розробника саме йому. Дана середа розробки є безкоштовною, має відкритий вихідний код і є кросплатформною. Це означає, що вона працює як на Windows, так і на Linux та macOS.

На відміну від багатьох інших редакторів коду, Visual Studio Code має вбудований налагоджувач, що робить процес розробки менш «клацаючим» і підтримує єдине представлення коду. Це значно полегшує та пришвидшує процес відстеження помилок. Більше не потрібно мати кілька екранів, щоб запускати різні консолі та переставляти їх щоразу, коли потрібно щось мінімізувати. Він вже вбудований в дизайн і бажаний робочий простір вже налаштований.

Ще одна важлива особливість ринку розширень для Visual Studio Code полягає в тому, що він не обмежується лише кодом. Є різноманітні користувацькі налагоджувачі, лінтери коду, ssh-термінали та конвеєрні з’єднання DevOps. І хоча головною особливістю Visual Studio Code є редактор коду, з додаванням розширень він стає більше, ніж просто редактором. За

допомогою правильних налаштувань, розширень і налаштованих комбінацій клавіш він може легко стати повноцінною робочою станцією.

Від перших кроків до встановлення нових розширень все в VS Code виглядає просто та інтуїтивно зрозуміло. VS Code створено за допомогою Electron – платформи для створення додатків із JavaScript за допомогою Chromium та Node.js. З іншого боку, VS Code використовує свою вражаючу інтеграцію TypeScript для автозаповнення та інших корисних функцій редагування як для JS, так і для TS.

Також хотілося б відмітити дизайн, що, як правило, є досить суб’єктивною річчю, але навіть якщо користувачеві не подобається базовий вигляд інтерфейсу, запропонований за замовчуванням, можна створювати власні налаштовані теми з високою гнучкістю, що дозволяє підлаштувати під себе майже всі елементи інтерфейсу редактора. Якщо ж немає часу на самостійне створення теми, є можливість обрати тему з тисяч, доступних на ринку VS Code.

Там же можна знайти й буквально тисячі розширень, в той час як нові з’являються щодня. Розширення можуть служити багатьом цілям. Починаючи від вже згаданих тем інтерфейсу користувача, до підтримки мов програмування, налагодження, інтеграції з Git і навіть програвачів Spotify. Також можна легко створити і своє власне розширення за допомогою JavaScript або TypeScript. Завдяки підтримці сотень мов VS Code допомагає миттєво працювати з наочним підсвічуванням синтаксису, узгодженням у дужках, зручним автоматичним відступом, виділенням полів, фрагментів тощо.

Отже, коли визначено всі необхідні засоби програмування і середу розробки, наступними етапами є визначення паттерну, за яким працюватиме веб-сервіс і проектування майбутньої бази даних.

2.2 Паттерн проектування

Здається, кожен, хто сьогодні займається веб-додатками, прагне використовувати MVC паттерн. Загальна ідея полягає в тому, щоб відокремити логіку бекенду від користувацького інтерфейсу, який представляє програму. MVC (скорочено від Model, View і Controller) є методологією або архітектурним шаблоном, який використовується для ефективного зв'язку інтерфейсу користувача з базовими моделями даних і організації коду програми. MVC використовується для поділу програми на три основні компоненти: модель, представлення і контролер. Далі кожен з цих компонентів розглянуто детальніше.

1. Модель. Цей рівень вважається найнижчим у порівнянні з представленням та контролером. Вона насамперед представляє дані для користувача та визначає зберігання всіх даних об'єктів програми.
2. Представлення. Цей рівень в основному пов'язаний з користувацьким інтерфейсом (UI) і використовується для надання візуального представлення моделі MVC. Простіше кажучи, цей рівень має справу з відображенням фактичного результату для користувача. Він також обробляє комунікацію між користувачем (введення, запити тощо) і контролером.
3. Контролер. Цей рівень займається обробником запитів. Його часто розглядають як мозок системи MVC – так би мовити зв'язок між користувачем і системою. Контролер завершує цикл отримання результатів користувача, перетворення їх у потрібні повідомлення та передачі їх у представлення (UI).

Звісно, свою популярність даний паттерн заслугував завдяки багатьом перевагам використання:

1. Організація великих веб-додатки. Оскільки існує сегрегація коду між трьома рівнями, стає надзвичайно легко розділити й організувати

логіку великомасштабних веб-додатків на менші (якими повинні керувати окремі групи розробників). Основна перевага використання таких методів кодування полягає в тому, що вони допомагають швидко знаходити певні частини коду та дозволяють легко редагувати старі або додавати нові функції.

2. Підтримка асинхронного виклику методів (AMI). Оскільки архітектура MVC добре працює з JavaScript та його фреймворками, не дивно, що вона також підтримує використання асинхронного виклику методів (AMI), що дозволяє розробникам створювати веб-додатки, які швидко завантажуються. Це означає, що програми MVC можна змусити працювати навіть з файлами PDF, веб-переглядачами, а також використовувати для віджетів на робочому столі.
3. Легке підлаштування. Використання методології MVC дозволяє легко модифікувати всю програму. Додавання/оновлення нового типу представлень спрощено у даному шаблоні (оскільки один розділ не залежить від інших). Отже, будь-які зміни в певному розділі програми ніколи не вплинуть на всю архітектуру. Це, у свою чергу, допоможе підвищити гнучкість та масштабованість програми.
4. Швидший процес розробки. Оскільки існує сегрегація коду між трьома рівнями, розробка веб-додатків за допомогою моделі MVC дозволяє одному розробнику працювати над певним розділом (скажімо, представленням), а інший може працювати над будь-яким іншим розділом (скажімо, контролером) одночасно. Це дозволяє легко реалізувати бізнес-логіку, а також допомагає прискорити процес розробки в декілька разів. Помічено, що в порівнянні з іншими моделями розробки, модель MVC в кінцевому підсумку демонструє вищу швидкість розробки.
5. Легке планування та обслуговування. Парадигма MVC корисна на початковому етапі планування програми, оскільки вона дає розробнику схему того, як упорядкувати свої ідеї в фактичний код.

Це також чудовий інструмент, який допомагає обмежити дублювання коду та полегшує обслуговування програми.

6. Повернення даних без форматування. Повертаючи невідформатовані дані, фреймворк MVC дає можливість створити власний механізм перегляду. Наприклад, будь-який тип даних можна відформатувати за допомогою HTML, але з MVC також є можливість відформатувати дані за допомогою програми Macromedia Flash або Dream Viewer. Це корисно для розробників, оскільки одні й ті самі компоненти можна повторно використовувати з будь-яким інтерфейсом.
7. Підтримка TTD (тест-керована розробка). Основною перевагою шаблону MVC є те, що він значно спрощує процес тестування. Це полегшує налагодження великомасштабних програм, оскільки всі рівні структурно визначені та правильно написані.
8. Декілька представлень. В архітектурі MVC легко досяжна розробка різних компонентів представлень для компонента моделі. Це дає можливість розробляти різні компоненти представлення, таким чином обмежуючи дублювання коду, оскільки він розділяє дані та бізнес-логіку.
9. SEO-дружня платформа. Платформа MVC підтримує розробку веб-додатків, зручних для SEO. Щоб збільшити кількість відвідувань з певної програми, MVC надає простий спосіб розробити оптимізовані для SEO URL-адреси RESTful.

Таким чином, підсумовуючи все вище викладене, можна зробити висновок, що паттерн проектування MVC є одним з найкращих рішень, які можна інтегрувати в проект для досягнення ефективних результатів. [22]

2.3 Система керування базою даних

Бази даних SQL, також відомі як реляційні бази даних, розроблені для зберігання інформації, яка має структуровану схему. У структурованій схемі

дані зберігаються у форматі рядків-стовпців, часто їх ще називають таблицями. Їх можна отримати за допомогою запитів, написаних мовою структурованих запитів (SQL).

Реляційні бази даних SQL були єдиним життєздатним комерційним рішенням для зберігання даних до 2000-х років, коли Інтернет та Web 2.0 почали генерувати велику кількість неструктурованої інформації. Такі дані не могли бути належним чином зіставлені з табличними схемами, тому виникла потреба в іншому класі баз для підтримки таких неструктурованих даних.

Саме тоді почали набирати популярність бази даних NoSQL, або, як їх ще називають, нереляційні. Ці нові бази даних могли забезпечити підтримку іншого типу даних, неструктурованих і не придатних для концепції схем; такі дані, як пари «ключ-значення», документи, текст, графіки та широкі стовпці. MongoDB, наприклад, переважно підтримує неструктуровані документи.

Накопичення неструктурованих даних було великим кроком у напрямку епохи Big Data, але з іншого боку, оскільки дані, що зберігалися, були неструктурованими, неможливо запитати ці дані за допомогою SQL. SQL до того моменту був єдиним стандартом для запитів і був добре відомий абсолютно всім розробникам.

У MySQL, наприклад, дані зберігаються в таблицях, де стовпець позначає атрибут, а рядок – певний запис. Ці таблиці, у свою чергу, знаходяться всередині баз даних. У MongoDB дані зберігаються в колекціях, аналогічних таблицям MySQL. Колекція може складатися з багатьох документів, дані в яких зберігаються у форматі JSON ключ-значення. У базі даних MongoDB можуть бути сотні таких колекцій.

Бази даних SQL мають таку реляційну властивість, коли різні таблиці пов’язані одна з одною зовнішніми (первинними) ключами, тоді як у MongoDB не можливо встановити такий зв’язок між неструктурованими даними колекцій. Також їх загалом можна масштабувати лише по вертикалі за рахунок збільшення розміру пам’яті, дискового простору або обчислювальної потужності сервера. Вертикальне масштабування є досить дорогим, оскільки

витрати швидко зростають для великих баз даних із великим обсягом запитів. Бази даних NoSQL, такі як MongoDB, підтримують горизонтальне масштабування, також відоме як шардінг. У цьому випадку замість збільшення конфігурації сервера додається новий сервер з метою масштабованості. Цей підхід зазвичай значно дешевший.

Бази даних SQL мають попередньо визначену схему, якій повинні відповідати дані. Наприклад, під час створення таблиці необхідно визначити кількість стовпців у таблиці та її тип даних. Будь-які дані, збережені в таблиці, повинні відповідати структурі таблиці, інакше це призведе до помилки. В той час як у MongoDB немає необхідності заздалегідь визначати будь-яку схему. Колекція може без проблем зберігати різні типи документів. Нема про що турбуватися, якщо надійде новий тип документа, його можна легко зберегти.

Динамічна природа схеми MongoDB корисна, оскільки більшість даних, які генеруються інтернет-додатками та пристроями IoT, неструктуровані, і їх неможливо зберегти в традиційній базі даних SQL. Крім того, багато компаній зберігають дані, перш ніж дізнаються, як вони будуть використані пізніше. Це характерно для мобільних додатків, які зберігають дані журналів і активність користувачів. Коли компанія виводить свої програми на ринок, вони збирають дані без кінцевої мети. Пізніше вони можуть виявити, що ці дані дають їм цінну інформацію про те, які функції необхідно додати. З неструктурованими базами даних простіше здійснювати подібний незапланований збір даних, оскільки немає необхідності визначати схему заздалегідь.

Підводячи підсумки, слід зазначити, що MongoDB обрана з огляду на її гнучкість, підтримку різних типів неструктурованих даних, що є важливим при роботі з даними користувача. Також MongoDB повсюдно використовується для збереження неструктурованих даних у форматі JSON, що є частиною даного бакалаврського проекту. [23]

2.4 Проектування інформаційної системи. Макети

Для наочного представлення роботи інформаційної системи в даному бакалаврському проекті на рисунку 2.2 наведено діаграму варіантів використання (Use Case Diagram), що є графічним зображенням можливої взаємодії користувача з системою. Діаграма варіантів використання, як стає зрозумілим з її назви, відображає різні варіанти використання та різні типи користувачів, яких має система. Варіанти використання представлені колами або еліпсами. Актори зображені у вигляді фігурок. Взаємозв’язки представлені у вигляді двох типів: «include», що означає обов’язкове виконання дії, яке в багатьох випадках відбувається автоматично і навіть без участі суб’єктів системи; «extend», що означає необов’язкове виконання дії, а наприклад лише за умови виконання якихось дій з боку користувача або зовнішніх обставин. [24]

Завдяки своїй спрощеній природі діаграми варіантів використання можуть бути ефективним інструментом комунікації для зацікавлених сторін. [25]

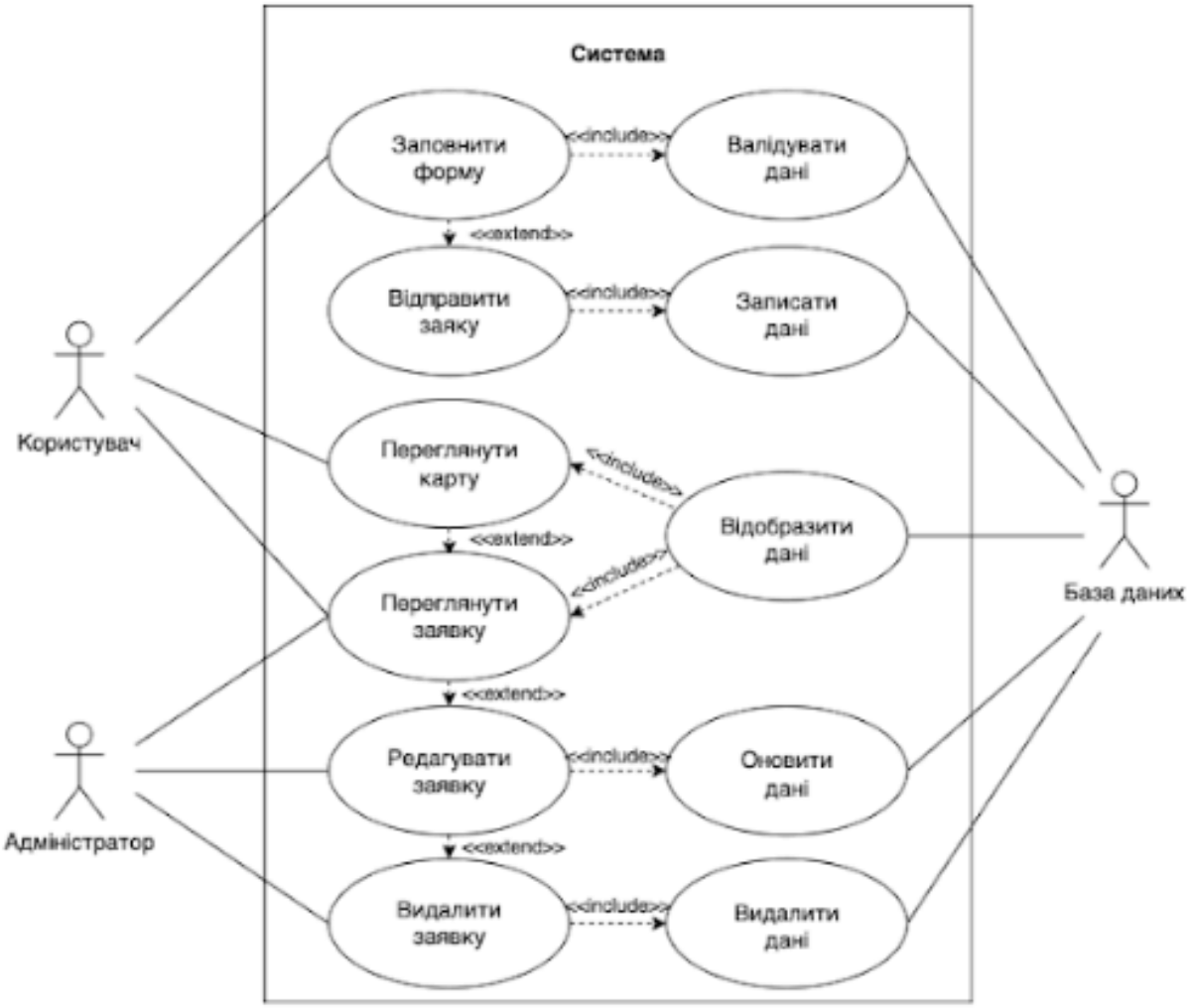

Рисунок 2.2 – Діаграма варіантів використання інформаційної системи

Після визначення функціонування інформаційної системи можна перейти до наступного етапу проектування – прототипування і створення макетів майбутнього веб-сервісу. Для їх розробки використаний інструмент Figma, а точніше його офлайн-версія для Windows, адже вона є повноформатною, швидшою і зручнішою за онлайн.

Колірна схема майбутнього веб-сервісу обрана компліментарна і складається з двох основних кольорів: темно-синього і жовтого, що з точки зору психології поєднує в собі доброту, вірність та постійність (синій колір) й тепло, сонце і довіру (жовтий колір), а також патріотичний характер, що є вкрай актуальним зараз. Допоміжними кольорами в інтерфейсі є темно-червоний і хакі (болотний).

Дизайн веб-сервісу мінімалістичний, головною метою якого є залишити на увазі у користувача тільки важливий контент. Додаткові ефекти, непотрібна

анімація перевантажують веб-сайт і тим самим відволікають користувача від кінцевої мети. Відповідні ілюстрації, якими збагачено прототип, одразу дадуть зрозуміти, до якої тематики відноситься і які цілі має веб-сторінка.

На рисунках 2.3-2.6 наведено ймовірний зовнішній вигляд лендінг-блоку, переліку послуг товариства, пошуку тварин з мапою і формою для заповнення і футеру (зберігаючи порядок переліку).

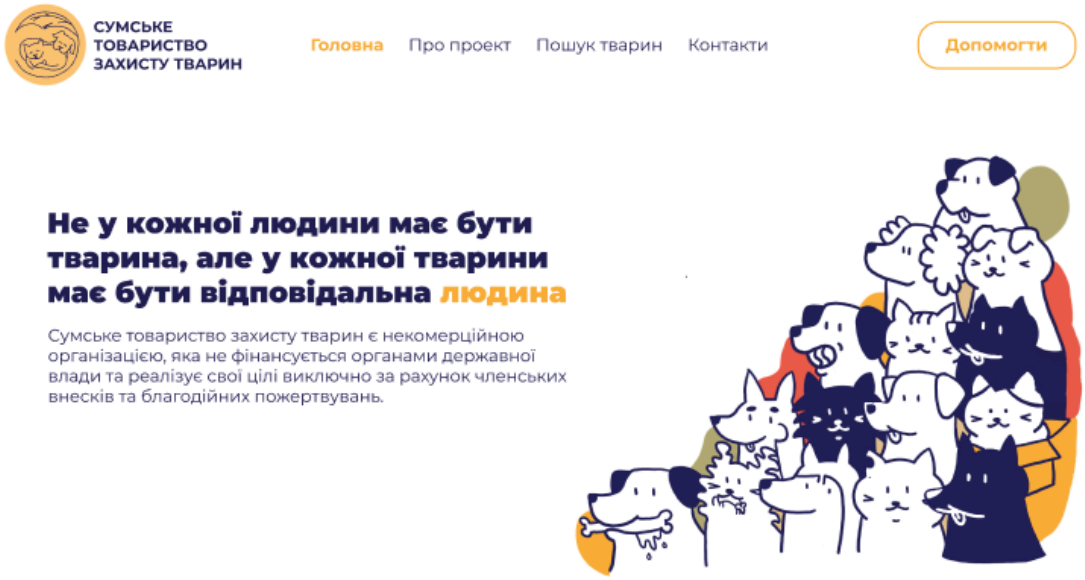

Рисунок 2.3 – Прототип лендінг-блоку

Рисунок 2.4 – Прототип переліку послуг товариства

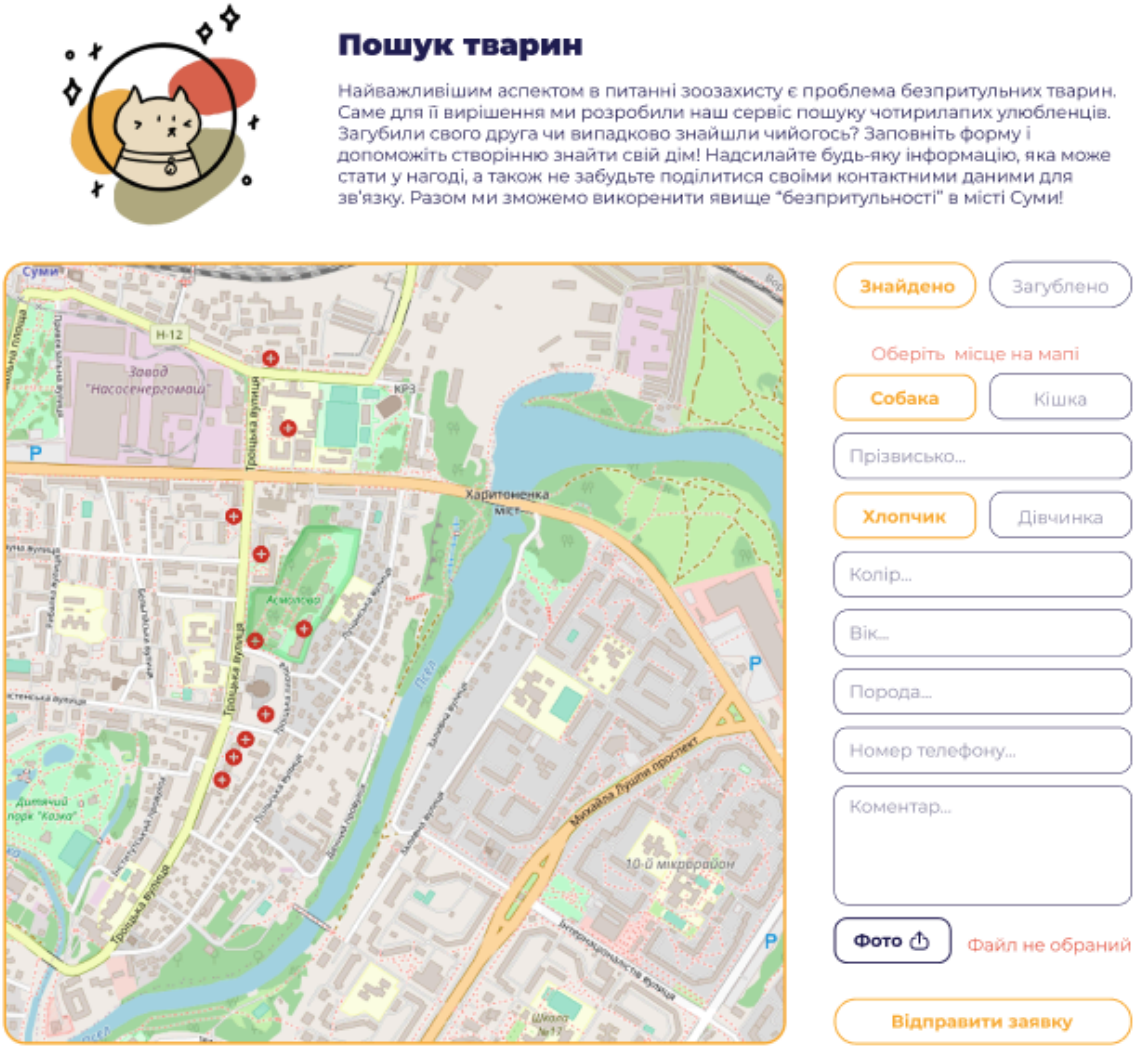

Рисунок 2.5 – Прототип пошуку тварин

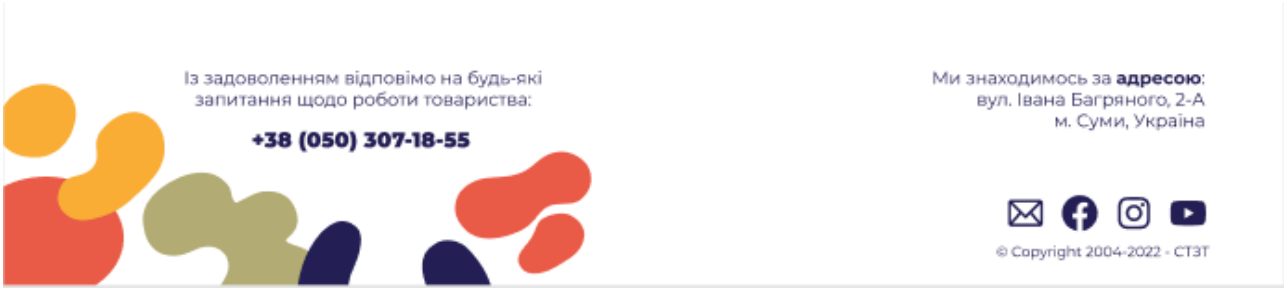

Рисунок 2.6 – Прототип футеру

Після того, як обраний дизайн майбутнього веб-сервісу, створені прототип і макети, етап проектування можна вважати завершеним і переходити до наступної стадії розробки – практичної реалізації веб-сервісу.

3. ПРАКТИЧНА РЕАЛІЗАЦІЯ

3.1 Проектування бази даних

Першим і одним з найважливіших етапів в програмній реалізації є проектування бази даних для наочного представлення даних, які будуть отримані від користувача, впорядковані і збережені для можливості подальшої роботи з ними, а також для легшого формулювання запитів до бази даних в коді програми в майбутньому. Дані для підключення зберігаються в файлі Config.js, де визначена відповідна однойменна константа:

```
const Config = {
    DBConnectUri: 'mongodb+srv://msmnnk:-9-
ifCZm6azXQHp@cluster0.zt4qc.mongodb.net/SumyPets?retryWrites=true
&w=majority',
    DBName: 'SumyPets',
```

Там же визначені й дані для входу в адміністративний акаунт:

```
    AdminLogin: 'admin',
    AdminPass: 'admin',
    AdminToken: '123456789',
}
```

На рисунку 3.1 наведена ER-діаграма бази даних SumyPets.

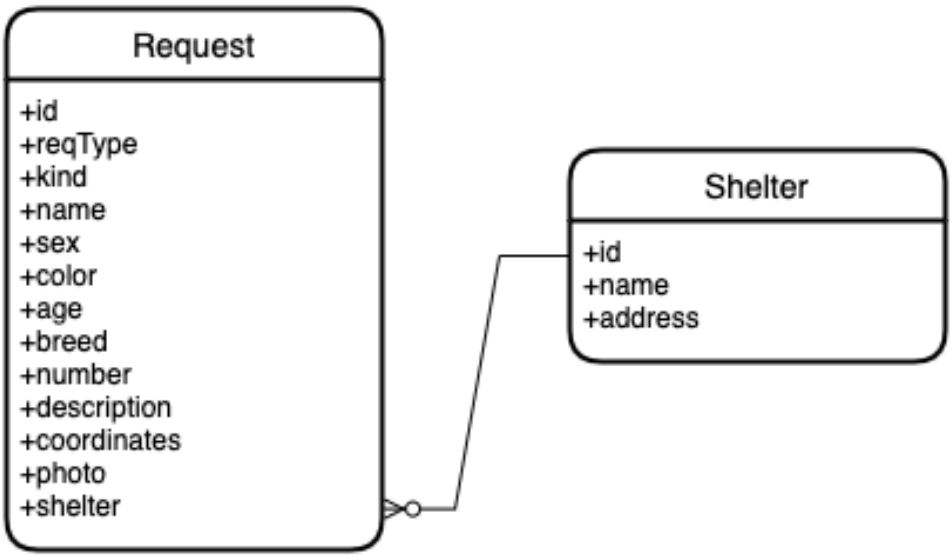

Рисунок 3.1 – ER-діаграма бази даних SumyPets

В базі даних проекту визначені дві таблиці: таблиця Request, яка містить в собі дані, введені користувачем в поля форми при оформленні заявки про знайдену або загублену тварину, або, іншими словами, відповідає за зберігання унікальних анкет тварин і ведення обліку; таблиця Shelter, яка містить дані про тимчасове місце перебування тварини (перетримки/прихистку/притулку), а саме його назву, адресу, а також унікальний ідентифікатор, який присвоюється тварині через акаунт з правами адміністратора. Одна тварина може перебувати лише в одному місці одночасно, але в одному місці можуть перебувати одночасно декілька тварин, тож зв’язок від таблиці Shelter до таблиці Request «один к багатьом». Також зв’язок є необов’язковим, адже тварина може не знайти свій тимчасовий прихисток (наприклад, одразу знайдений старий або новий господар), що відображено на ER-діаграмі.

В таблицях 3.1 і 3.2 детально описані всі поля обох таблиць, їх тип даних, джерело отримання інформації та обов’язковість.

Таблиця 3.1 – Опис полів таблиці Request

Атрибут	Обов’язково	Тип даних	Джерело даних	Пояснення
id	+	String	Створюється автоматично	Унікальний ідентифікатор заявки
reqType	+	String	Обирається користувачем	Тип заявки (загублено/знайдено)
kind	+	String	Обирається користувачем	Тип тварини (собака/кішка)
name	-	String	Вводиться користувачем	Прізвисько (ім’я) тварини
sex	+	String	Обирається користувачем	Стать тварини
color	-	String	Вводиться користувачем	Колір тварини
age	-	String	Вводиться користувачем	Вік тварини
breed	-	String	Вводиться користувачем	Порода тварини
number	+	String	Вводиться користувачем	Контактний номер телефону заповнювача
description	-	String	Вводиться користувачем	Коментар до заявки, опис, додаткові деталі
coordinates	+	lat:Number, lng:Number	Обирається користувачем	Координати місця знаходження тварини
photo	+	String	Завантажується користувачем	Фото тварини
shelter	-	Schema.Types.ObjectID	Присвоюється адміністратором	Тимчасове місце перебування тварини

Таблиця 3.2 – Опис полів таблиці Shelter

Атрибут	**Обов’язково**	**Тип даних**	**Джерело даних**	**Пояснення**
id	+	String	Створюється автоматично	Унікальний ідентифікатор місця перетримки
name	+	String	Вводиться адміністратором	Назва місця перетримки/прихистку
address	+	String	Вводиться адміністратором	Адреса місця перетримки/прихистку

Оскільки використана база даних MongoDB є нереляційною, створення таблиць відрізняється за синтаксисом і принципом від створення таблиць в SQL. За створення таблиць відповідають файли Request.js та Shelter.js відповідно, обидва з них містять однакові рядки коду на початку:

```
const mongoose = require("mongoose");
const Schema = mongoose.Schema;
```

Саме цей фрагмент коду задає модель або схему бази даних.

Унікальні ідентифікатори заявок і унікальні ідентифікатори можливих місць перетримки тварини створюються автоматично базою даних, без участі користувача або адміністратора. Саме цей ідентифікатор є так званим «первинним ключем», за яким можна відрізнити один запис в таблиці від іншого, адже інші дані можуть співпасти.

Ось так виглядає створення таблиці Request в однойменному файлі:

```
let RequestSchema = new Schema({
reqType: {
        type: String,
        enum : ['Загублено','Знайдено'],
        required: true
    },
    kind: {
        type: String,
        enum : ['Собака','Кішка'],
        required: true
    },
    sex: {
        type: String,
        enum : ['Хлопчик','Дівчинка'],
        required: true
    },
    name: {
        type: String,
        default: null
```

```
},
color: {
    type: String,
    default: null
},
age: {
    type: String,
    default: null
},
breed: {
    type: String,
    default: null
},
number: {
    type: String,
    default: null
},
description: {
    type: String,
    default: null
},
coordinates: {
    lat: Number,
    lng: Number
},
photo: {
    type: String,
    required: true
},
shelter: {
    type: Schema.Types.ObjectId,
    ref: 'Shelter',
    default: null
}
```

```
},
{ versionKey: false });
```

Створення таблиці Shelter відбувається аналогічним чином в файлі Shelter.js відповідно:

```
let RequestSchema = new Schema({
    reqType: {
        type: String,
        enum : ['Загублено','Знайдено'],
        required: true
    },
    kind: {
        type: String,
        enum : ['Собака','Кішка'],
        required: true
    },
    sex: {
        type: String,
        enum : ['Хлопчик','Дівчинка'],
        required: true
    },
    name: {
        type: String,
        default: null
    },
    color: {
        type: String,
        default: null
    },
    age: {
        type: String,
        default: null
    },
    breed: {
        type: String,
```

```
            default: null
        },
        number: {
            type: String,
            default: null
        },
        description: {
            type: String,
            default: null
        },
        coordinates: {
            lat: Number,
            lng: Number
        },
        photo: {
            type: String,
            required: true
        },
        shelter: {
            type: Schema.Types.ObjectId,
            ref: 'Shelter',
            default: null
        }
    },
    { versionKey: false });
```

Вигляд бази даних з відповідними таблицями в адміністративній панелі та приклад вигляду таблиці з необхідними полями і даними відображено на рисунках 3.2 і 3.3 відповідно.

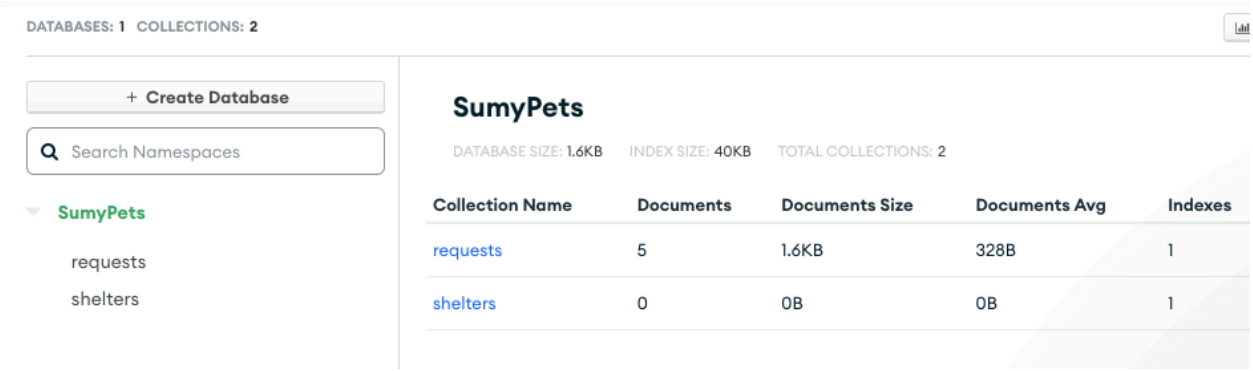

Рисунок 3.2 – Вигляд бази даних в адміністративній панелі

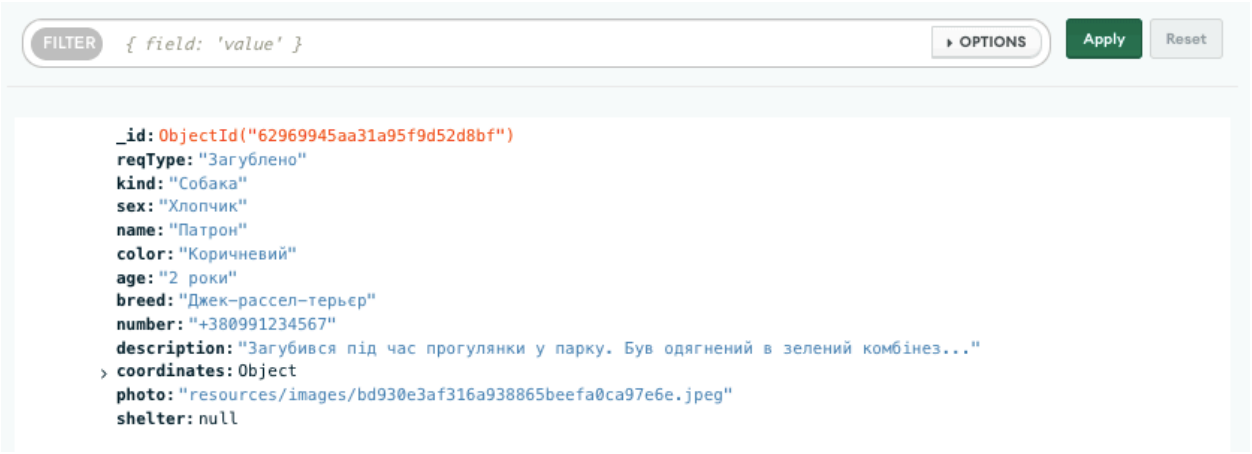

Рисунок 3.3 – Вигляд таблиці Request з відповідними полями

Незамінним візуальним представленням всіх процесів з точки зору даних є DF-діаграми, які є незамінними під час розробки інформаційної системи. Діаграма дозволяє візуалізувати рух даних між об’єктами системи, так і перетворення даних, які можуть застосовуватися на різних кроках процесу. [26] На малюнках 3.4 і 3.5 представлені верхній рівень DF-діаграми для інформаційної системи проекту і декомпозиція її головного елементу відповідно.

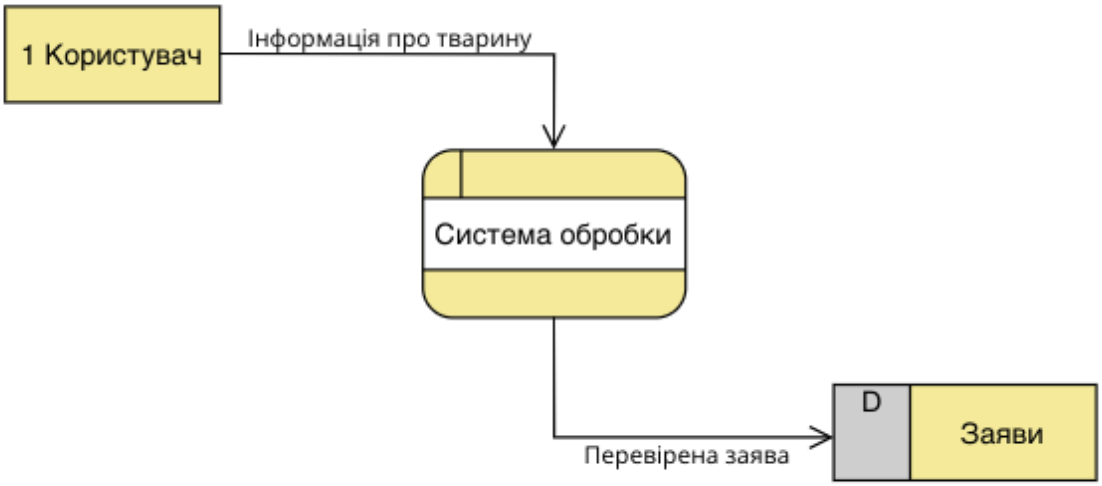

Рисунок 3.4 – Верхній рівень DF-діаграми

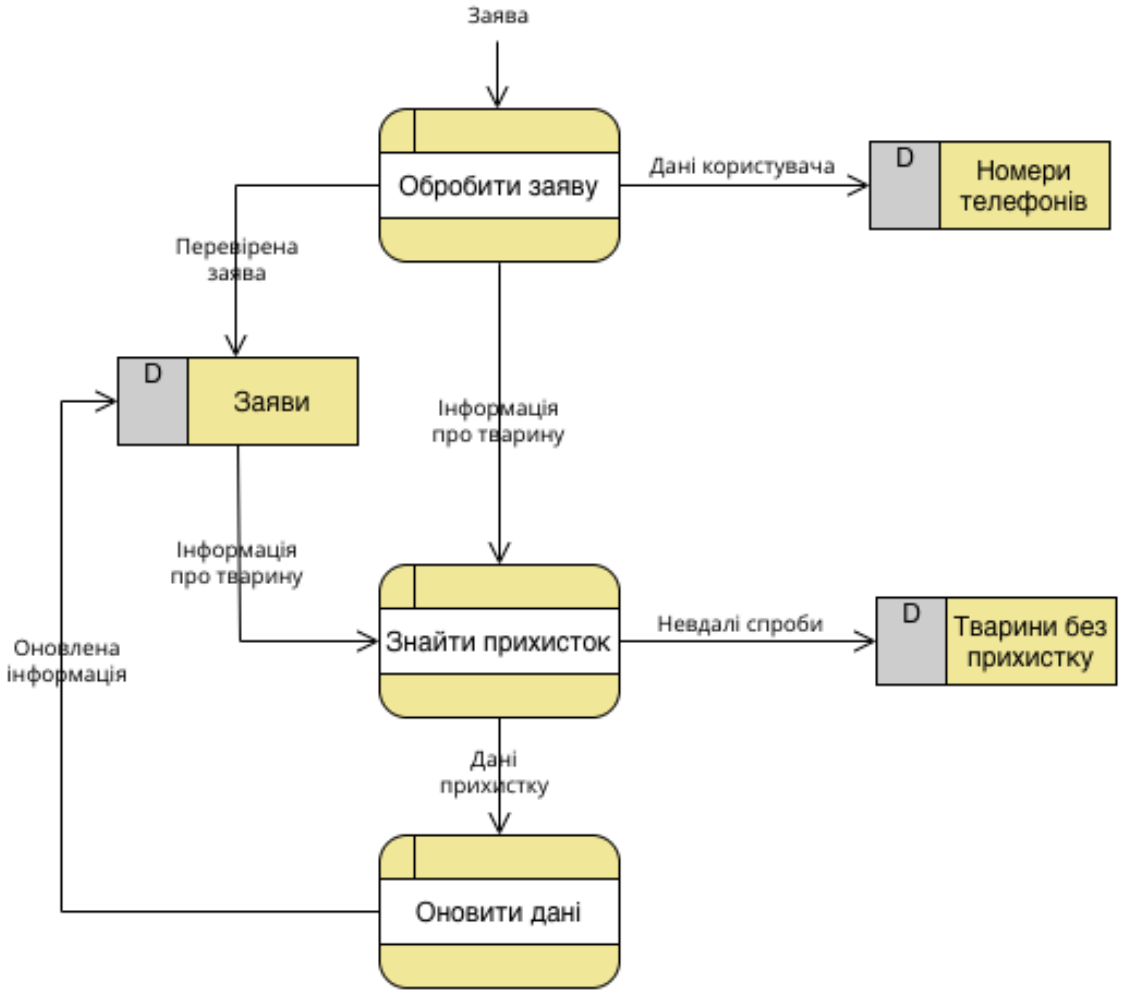

Рисунок 3.5 – Декомпозиція головного елемента DF-діаграми

Отже, коли базу даних визначено і спроектовано, можна приступити до наступних кроків практичної реалізації, а саме до програмування основного функціоналу веб-сервісу.

3.2 Реалізація карти

Для реалізації картографічного інтерфейсу в даному бакалаврському проекті використовується JavaScript бібліотека Leaflet. Вона дозволяє реалізувати підтримку різноманітних шарів мапи, які побудовані за такими технологіями, як, наприклад, WMS, GeoJSON або векторного відображення поверхні. Бібліотека Leaflet допомагає працювати з різними типами даних, основними з яких є:

- растрові типи (TileLayer і ImageOverlay);
- векторні типи (Path, Polygon і специфічні типи, такі як Circle);
- групові типи (LayerGroup, FeatureGroup і GeoJSON);
- керуючі елементи (Zoom, Layers тощо).

Також існують допоміжні класи для керування проекціями та трансформаціями і взаємодії з об’єктною моделлю документу (DOM).

В html-документі місце, де має відображатися мапа, визначено за допомогою елементу div. Для області відображення визначаються параметри поведінки мапи, а саме джерело отримання, масштаб відображення, точки, з яких починати відлік, вигляд маркерів тощо. Оскільки додаток розробляється для Сумського товариства захисту тварин, чия діяльність є локальною і обмежується лише містом Суми, радіус можливого місцезнаходження тварини на карті обмежений 5 км (5000 в системі SI), аби користувач не обирав координати і маркер за межами міста. Активна область підсвічується світло-жовтим кольором, аби користувач міг наочно побачити, чи розповсюджується діяльність товариства на той чи інший район:

```
let circle = L.circle([50.907688, 34.796716], {
    color: 'yellow',
    fillColor: 'yellow',
    fillOpacity: 0.08,
    radius: 5000
}).addTo(map);
```

Наступний фрагмент коду відповідає за завантаження головного шару карти. У tileLayer записаний токен, який й посилається на API мапи:

```
window.onload = function() {
    map = L.map('map').setView([centerLat, centerLong], 13);
    L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png', {
    attribution: '&copy; <a
href="http://osm.org/copyright">OpenStreetMap</a> contributors'
    }).addTo(map);
```

Наведений далі фрагмент коду відповідає за можливість відмітити на мапі координати місцезнаходження тварини, формування маркеру або його видалення, якщо користувач захоче змінити місцезнаходження:

```
map.on('click', function(e){
    let center = {
        lat: centerLat,
        lng: centerLong,
    }
    let point = {
        lat: e.latlng.lat,
        lng: e.latlng.lng,
    }
    if(getDistance(center, point) <= 5000) {
        if (marker) {
            marker.remove()
        }
        marker = L.marker(e.latlng).addTo(map);
        selectedPoint = point;
    }
});
```

Якщо заявка заповнена згідно всіх вимог і валідація виконана вдало, координати місцезнаходження разом з іншими даними запишуться в базу даних:

```
function addMarker(request) {
    let icon = getIcon(request.reqType, request.kind);
```

```
        let marker = L.marker([request.coordinates.lat,
    request.coordinates.lng], {icon: icon});
        let description = request.description;
        let popupText = '<div class="popup"><img class="popup-image"
    src="'+ request.photo +'"></br><p>'+ description +'</p></div>';
        marker.bindPopup(popupText);
        marker.on('click', function(e) {
            openAnimalForm(request);
        });
        animalsMarkers.push(marker);
        return marker;
    }
```

Функція clearMap, в свою чергу, відповідає за очищення мапи після того, як дії користувача завершені, заявку прийнято і додано до бази даних:

```
    function clearMap() {
        for (let i = 0; i < animalsMarkers.length; i++) {
            animalsMarkers[i].remove();
        }
        animalsMarkers = [];
    }
```

Відповідно до типу заявки (загублено тварину чи знайдено) та типу тварини (собака чи кішка) виведення маркеру відбувається за допомогою різних іконок, щоб можна було відрізнити заявки одну від одної і швидко знайти потрібну. Якщо тварину знайдено, кішка чи собака відображатимуться в зеленому колі, а якщо загублено – в червоному (рис. 3.6).

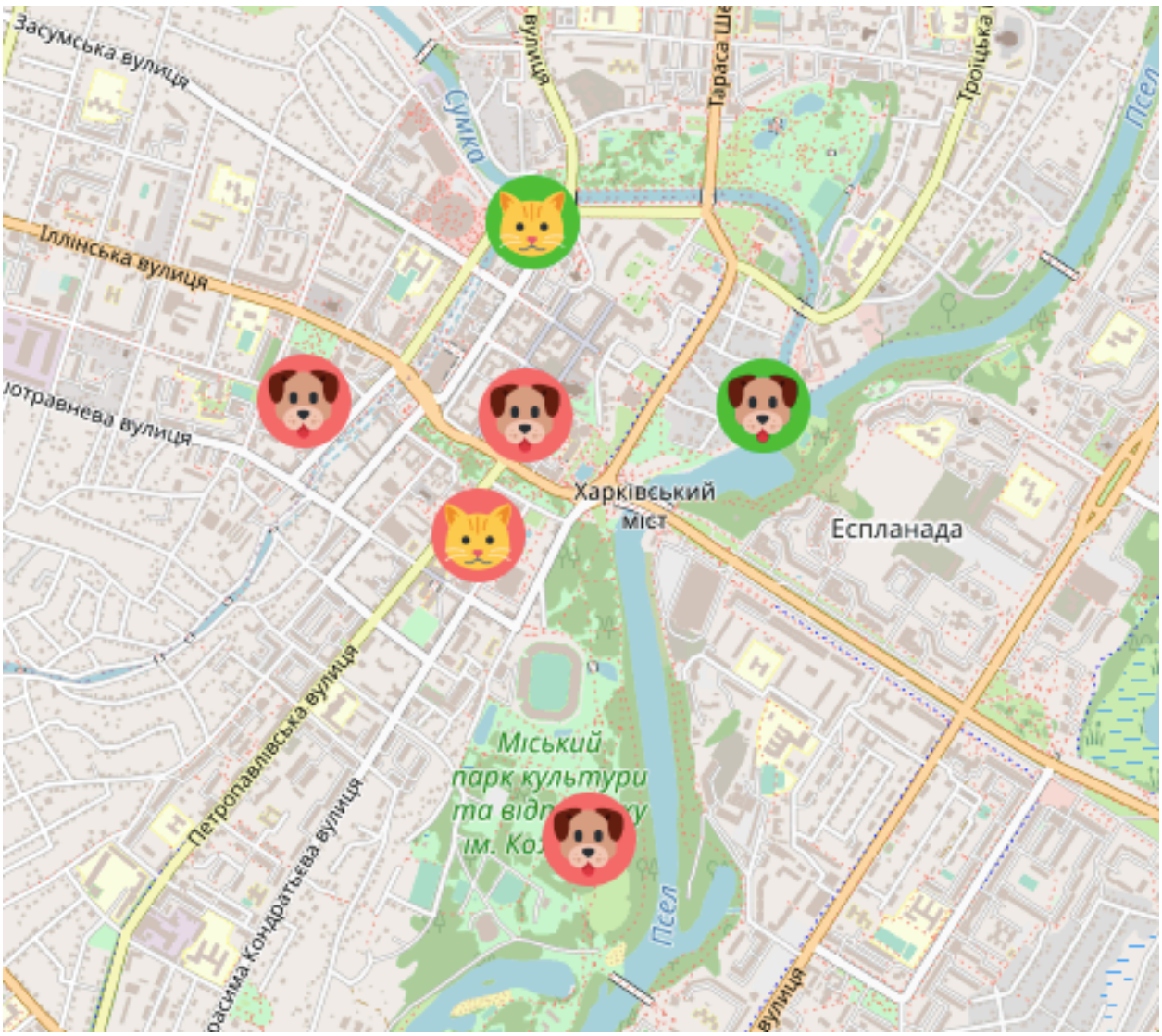

Рисунок 3.6 – Приклад вигляду маркерів залежно від типу заявки

За присвоєння відповідних маркерів і виведення їх на мапі відповідає наступна функція:

```
function getIcon(reqType, kind) {
    let iconName = '';
    if (reqType === 'Загублено') {
        iconName = 'lost';
    }   else if (reqType === 'Знайдено') {
        iconName = 'found';
    }
    if (kind === 'Кішка') {
        iconName += 'Cat';
    } else if (kind === 'Собака') {
        iconName += 'Dog';
```

```
    }
    return L.icon({
        iconUrl: 'resources/' + iconName + '.png',
        iconSize: [40, 40],
        iconAnchor: [20, 20],
    });
}
```

Всі функції, які відповідають за будь-які маніпуляції з картою, в тому числі й описані вище, знаходяться в файлі map.js.

3.3 Зберігання зображень

Для того, щоб була можливість ідентифікувати і розпізнати тварину, при заповненні заявки додавання фотографії є обов’язковою умовою (тобто поле має властивість required). З огляду на цей факт, постає питання, як саме реалізувати зберігання зображень найбільш оптимальним і раціональним способом.

Можна, звичайно, зберігати зображення прямо в базі даних, але це не продуктивний спосіб, якщо немає великого кластеру, який компенсує знижену швидкість доступу за рахунок розпаралелювання. Найкращим рішенням з точки зору як ефективності, так і безпеки в майбутньому для подальшої розробки та міграції є збереження зображень в місцезнаходженні X (наприклад, на окремому сервері), а потім додавання посилання або ключа до цього зображення в базу даних. Саме зображення при цьому не зберігається в базі, не займає багато обсягу пам’яті і не знижує продуктивність. Ось як це реалізовано в даному проекті:

```
app.post('/api/loadPhoto', upload.single('file'), (req, res) => {
    const file = req.file;
    let fileExtension = req.file.originalname.split('.').pop();
```

```
        fs.rename(imagesPath + '/' + file.filename, imagesPath + '/'
    + file.filename + '.' + fileExtension, (err) => {
            res.status(400).send();
        });
        res.status(200).send({status: 'ok', fileUrl:
    'resources/images/' + file.filename + '.' + fileExtension});
    });
```

Користувач отримає повідомлення про помилку, якщо фото важить більше, ніж 5 Мб. Таке обмеження накладене для економії простору, адже в подальшому на сервері зберігатиметься велика кількість зображень.

```
function fileHandler(event) {
    let file = event.target.files[0];
    if (file.size / 1024 / 1024 < 5) {
        document.getElementById('file-name').innerHTML =
file.name;
    } else {
        formData.photo = document.getElementById('form-
photo').value = null;
        Toastify({
            text: '⚠ Фото завелике, виберіть фото менше 5 Мб',
            duration: 3000,
            gravity: 'bottom',
            position: 'center',
            stopOnFocus: true,
            className: 'toast-error'
        }).showToast();
    }
}
```

Оскільки наявність фото в заявці є обов’язковою умовою, спочатку необхідно відправити його на сервер, перевірити наявність коректного шляху до нього, і лише після цього можна записати всі дані, введені користувачем у відповідні поля форми, в базу даних:

```
if (validateForm(formData)) {
    let form = new FormData()
    form.append('file', formData.photo);
    fetch('api/loadPhoto', {
        method: 'POST',
        body: form
    })
    .then(response => response.json())
    .then(data => {
        formData.photo = data.fileUrl;
        fetch('api/submitRequest', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json'
            },
            body: JSON.stringify(formData)
        })
```

На рисунку 3.7 можна побачити, як саме виглядатиме отримана від користувача інформація після її обробки і запису в базу даних. Як видно, атрибут photo дійсно містить в собі шлях до файлу замість того, щоб зберігати саме зображення.

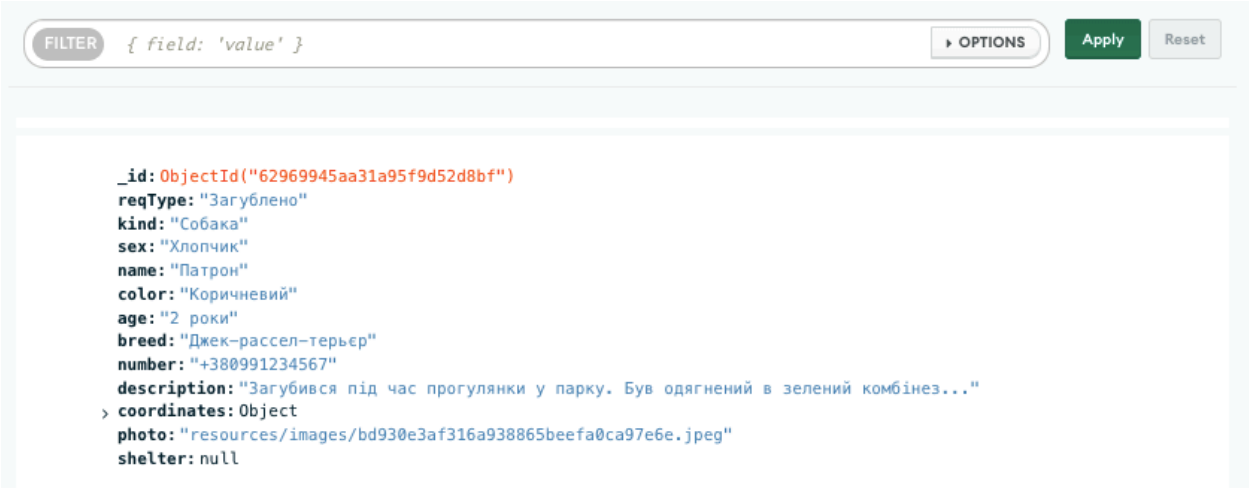

Рисунок 3.7 – Приклад вигляду запису заявки в базі даних

Звичайно, якщо файли невеликого розміру (наприклад, іконки або зменшені аватари профілю) та їх кількість невелика, то можна

використовувати прямий запис зображень в базу даних. Плюсом такого рішення є наявність бекапу, який завжди зберігає файли, а також можливість застосовувати каскадні операції. Але важливим мінусом є той факт, що база даних стає великого розміру, а отже бекап та відновлення можуть тривати годинами. У випадку розроблюваного веб-сервісу такий варіант не є оптимальним, адже кожне зображення може досягати 5 Мб, а їх кількість, скоріше за все, буде великою через великий обсяг заявок.

3.4 Адміністративна панель

Для того, щоб додати можливість керування заявками і додавання тимчасового прихистку для тварин, реалізовано адміністративну панель сайту. Найрозповсюдженішим методом авторизації насьогодні є введення логіну і паролю. Саме цей процес й використаний в даному веб-сервісі: якщо додати /admin до посилання, користувач потрапить на форму авторизації для адміністраторів сайту. За умови введення коректного логіну та паролю і після перевірки токену, користувач буде переадресований в адміністративну панель:

```
app.post('/api/login', (req, res) => {
    if (req.body.username === Config.AdminLogin &&
req.body.password === Config.AdminPass) {
        res.status(200).send({status: 'ok', token:
Config.AdminToken});
    } else {
        res.status(400).send({status: 'error'});
    }
});

app.post('/api/checkToken', (req, res) => {
    if (req.body.token === Config.AdminToken) {
        res.status(200).send({status: 'ok'});
    } else {
        res.status(401).send({status: 'unauthorized'});
```

```
    }
});

app.get('/admin', (req, res) => {
    res.sendFile(path.join(__dirname, '../views/admin.html'));
});
```

Далі з’являється інтерфейс зі списком усіх заявок з бази даних (рис. 3.8). При натисканні на будь-яку з них відкривається форма для редагування з усіма даними про тварину і можливістю додати місце перетримки (тимчасовий прихисток) з адресою та ім’ям людини або назвою товариства, де саме знаходиться тварина (рис. 3.9).

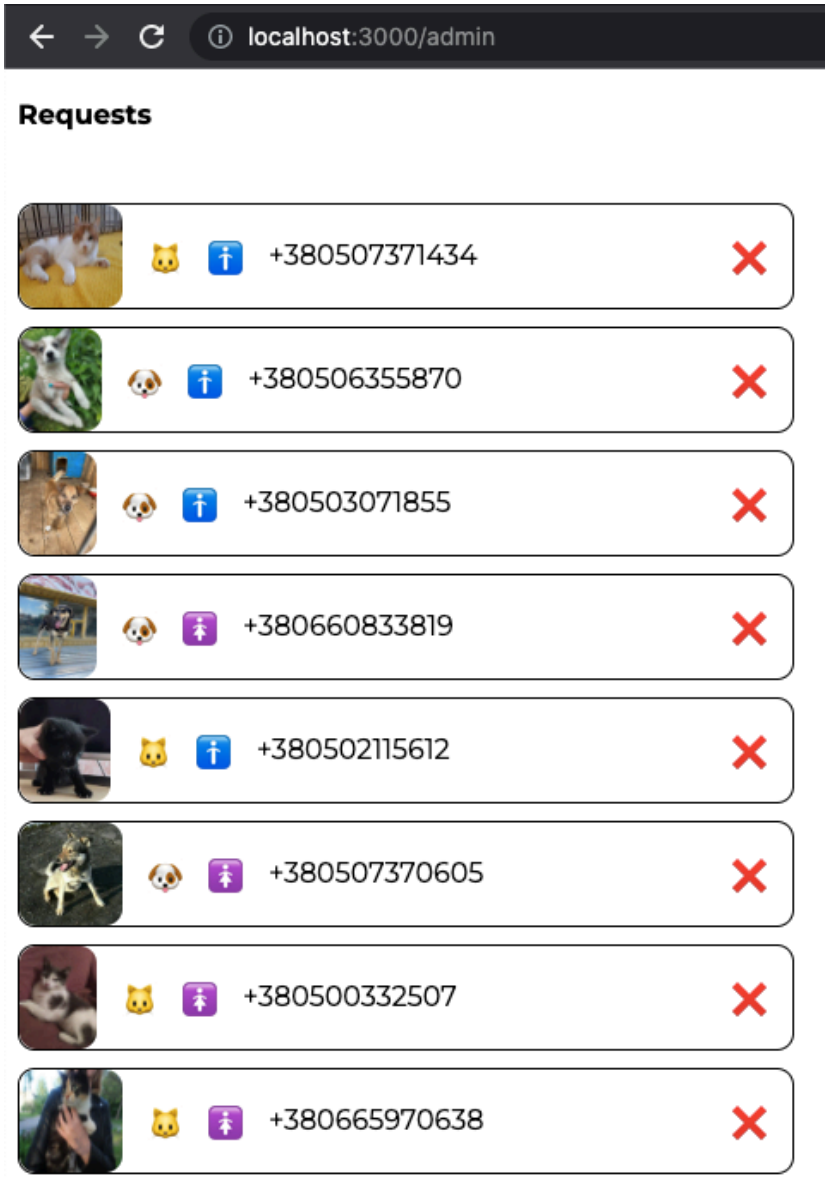

Рисунок 3.8 – Перелік заявок в адміністративній панелі

Загублено

Кішка

Дівчинка

Соня

2 роки

Немає

Різнольорова, білий з чорним і рижим

+380665970638

Втекла з рулеткою. Свідки бачили її збоку оз. Чеха, серед чотирнадцяти

Перетримки

Наталія

вул. Герасима Кондратьєва, 132/1

Додати

Зберегти

Рисунок 3.9 – Вигляд форми редагування заявки в адміністративній панелі

Кожна дія в адміністративній панелі супроводжується перевіркою відповідного токену, наприклад при додаванні тимчасового місця перебування тварини:

```
app.post('/api/addShelter', (req, res) => {
```

```
    if (req.headers.authorization === Config.AdminToken) {
        shelterController.createNew(req.body).then((shelter) => {
            res.status(200).send({status: 'ok', shelter:
shelter});
        }).catch((err) => {
            res.status(400).send({status: 'error', error: err});
        });
    } else {
        res.status(400).send({status: 'error', error: 'Не
авторизовано'});
    }
});
```

Або під час видалення тимчасового місця перебування тварини:

```
app.post('/api/deleteShelter', (req, res) => {
    if (req.headers.authorization === Config.AdminToken) {
        shelterController.deleteShelter(req.body.id).then(() => {
            res.status(200).send({status: 'ok'});
        }).catch((err) => {
            res.status(400).send({status: 'error', error: err});
        });
    } else {
        res.status(400).send({status: 'error', error: 'Не
авторизовано'});
    }
});
```

Всі функції, які відповідають за редагування, видалення, додавання або будь-яку іншу дію в адміністративній панелі, описані у файлі admin.js. Логіку відображення додано в окремий html-файл з назвою admin.html, куди й відбувається переадресація при коректній авторизації. Усі зображення і номери телефонів для прикладів взяті і сторінок соціальних мереж Сумського товариства захисту тварин.

3.5 Використання веб-сервісу

Використання веб-сервісу, звичайно, починається з головної сторінки. В головному розділі сайту знаходиться хедер з логотипом Сумського товариства захисту тварин, головним меню та кнопкою «допомогти», а також ілюстрація, слоган і короткий опис діяльності товариства. Цей розділ виглядає як звичайна лендінг-сторінка сайту. Все це відображено на рисунку 3.10. Кнопка допомогти, в свою чергу, переадресовує користувача на форму для переказу коштів за вказаними реквізитами через Приват (рис. 3.11). Необхідно лише ввести дані своєї картки, обрати відповідну суму і натиснути «переказати».

Наступний розділ «Про проект» зроблений у вигляді карток, які ілюструють і дають короткий опис кожної послуги товариства, а саме: пошук тварин, прилаштування, стерилізація, розміщення інформації, юридична допомога та лекції в закладах освіти (рис 3.12).

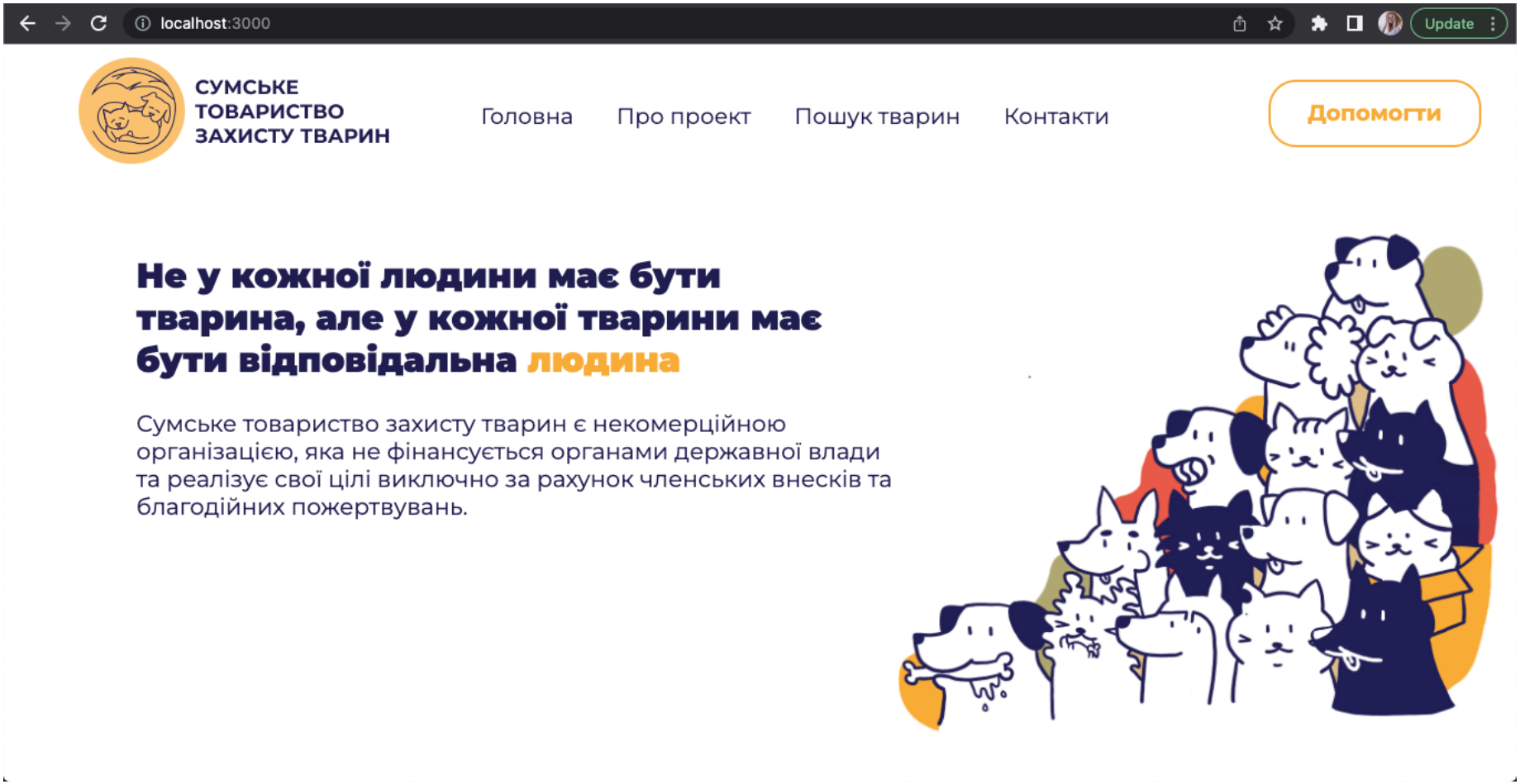

Рисунок 3.10 – Головна сторінка веб-сервісу

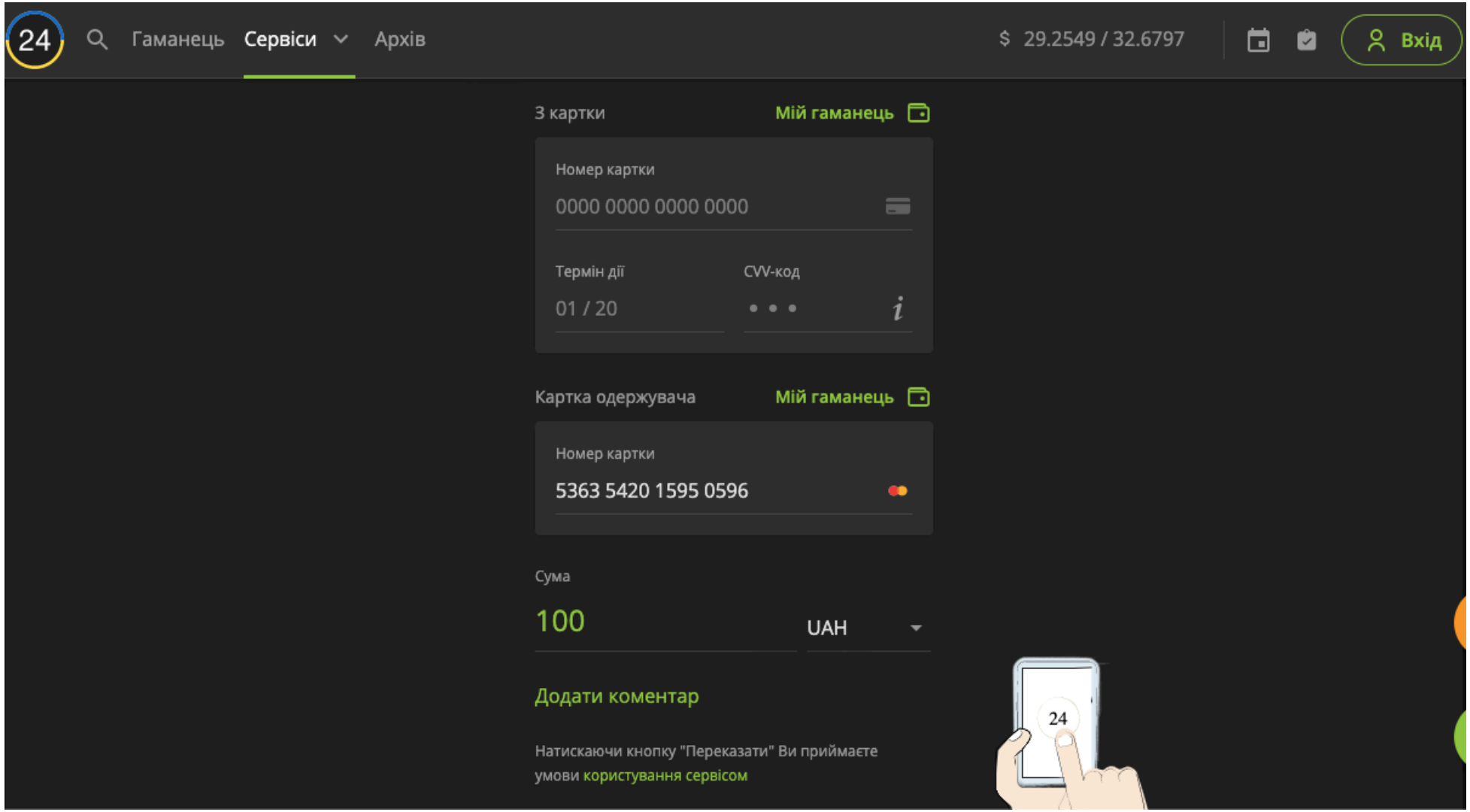

Рисунок 3.11 – Переказ коштів після натискання кнопки «допомогти»

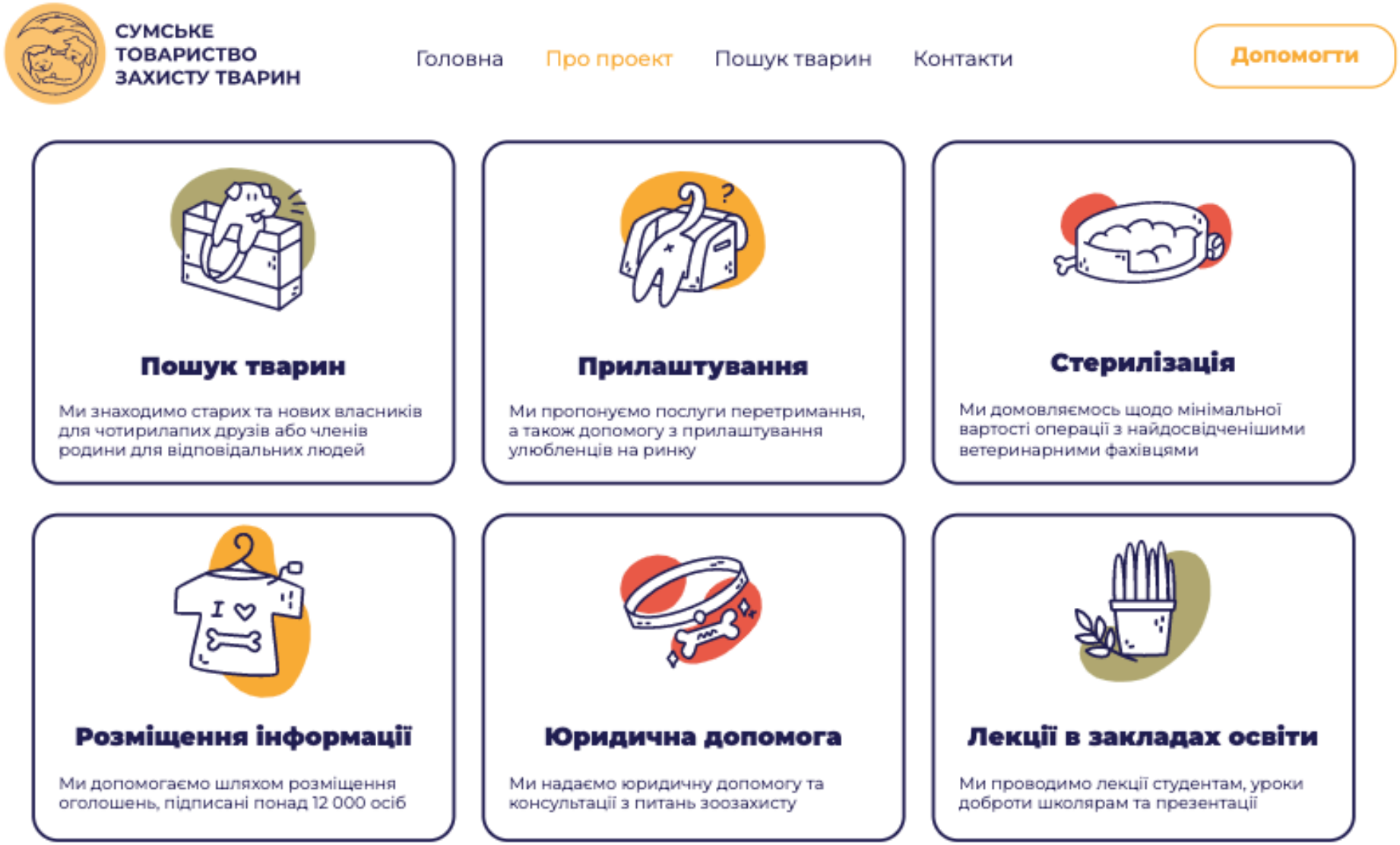

Рисунок 3.12 – Вигляд розділу «Про проект»

Головним функціоналом проекту є реалізація інструменту пошуку тварин з використанням картографічного API-інтерфейсу. Саме цей функціонал знаходиться в однойменному розділі «Пошук тварин». Він починається з опису даного інструменту і заклику ним скористуватися (рис. 3.14). Розділ має дві вкладки: «Нова заявка» і «Тварини». На першій вкладці, яка відкривається за замовчуванням, користувач має змогу відправити нову заявку. Як було зазначено раніше, з огляду на те, що діяльність товариства розповсюджується лише на місто Суми і заявки за його територією не можуть бути оброблені, на карті відображено територію, на якій заявки будуть розглянуті, у вигляді жовтого кола з радіусом 5 км (рис. 3.13).

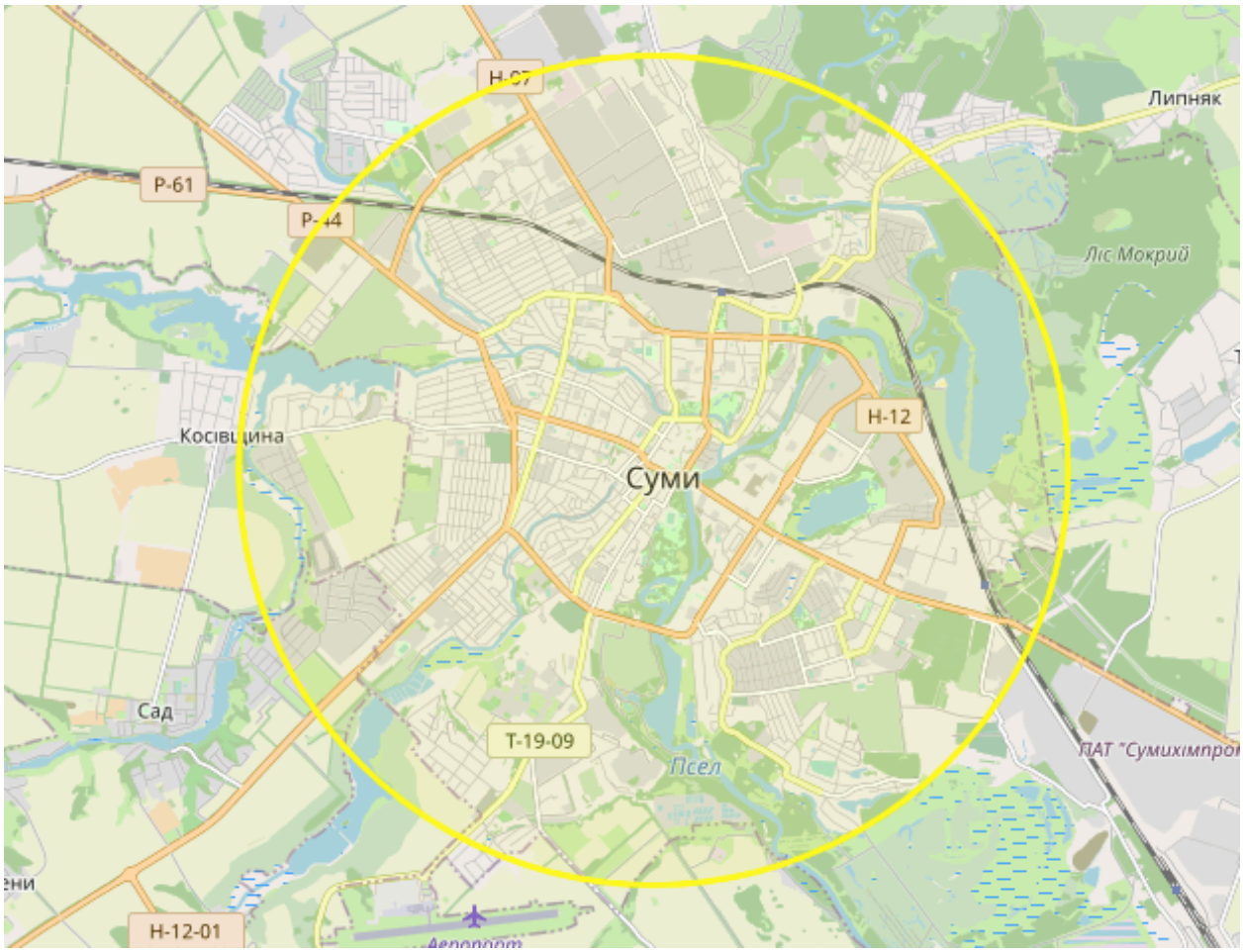

Рисунок 3.13 – Активна територія діяльності товариства на карті

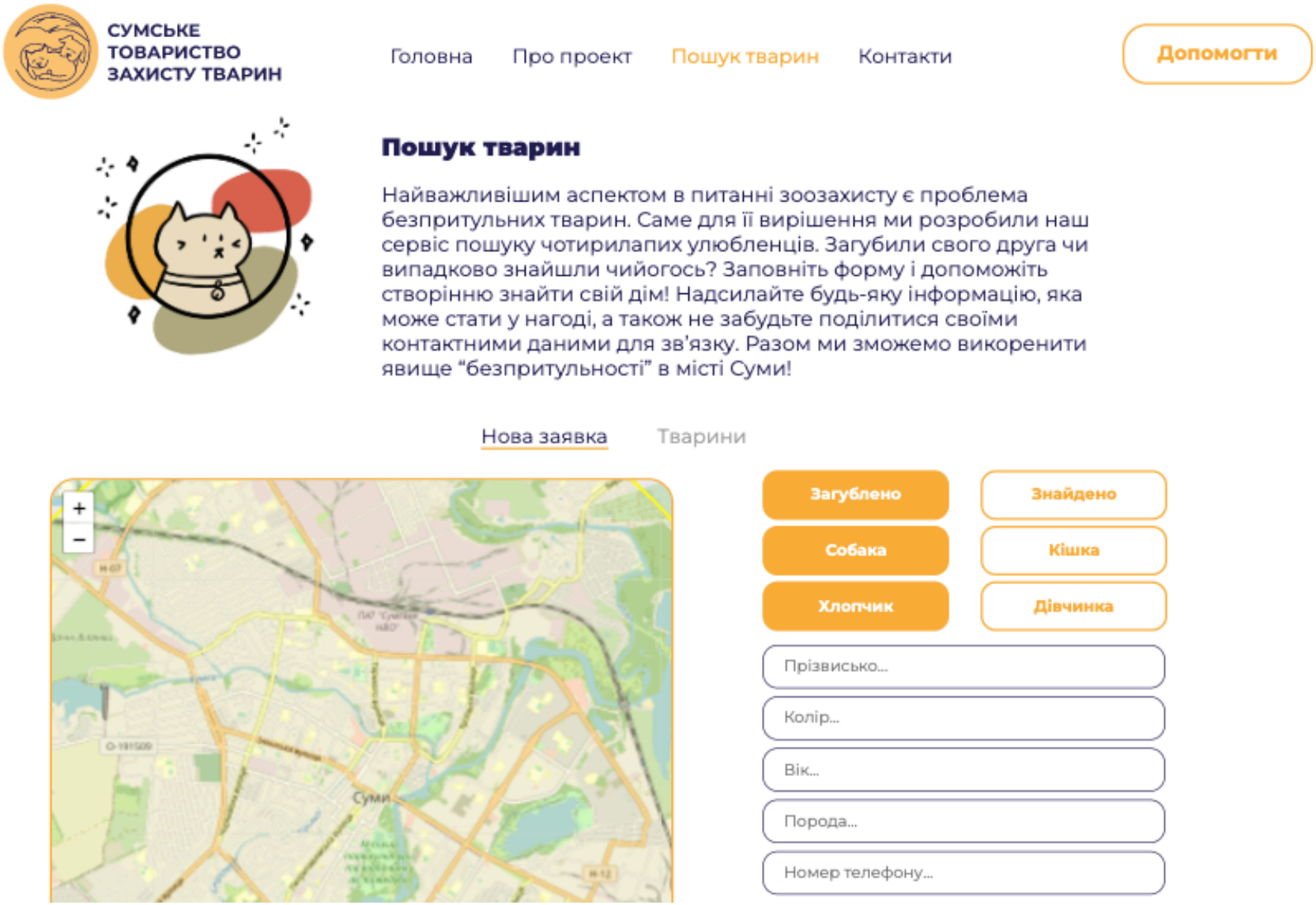

Рисунок 3.14 – Вигляд розділу «Про проект»

Вкладка «Нова заявка» містить в собі форму для заповнення і карту, щоб відмітити місцезнаходження тварини. Перш за все, користувач має обрати тип заявки «Загублено» або «Знайдено», адже від цього залежить вигляд форми і наявність полів для введення даних. Різницю показано на рисунках 3.20 і 3.21 відповідно. Якщо обрано тип заявки «Знайдено», відсутні такі поля, як «Прізвисько» та «Вік», адже для незнайомої тварини визначити це неможливо. Якщо заявка заповнена згідно усіх вимог (приклад на рис. 3.22), користувач отримає повідомлення про те, що заявку відправлено (рис. 3.15).

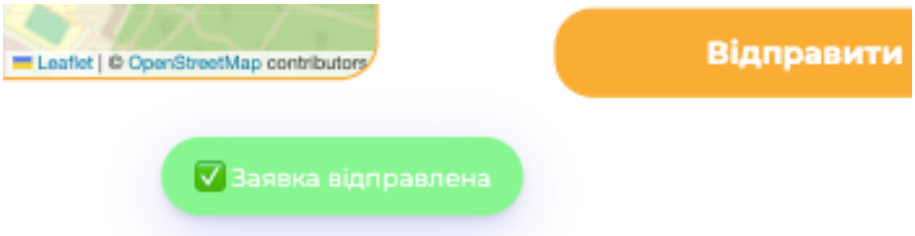

Рисунок 3.15 – Повідомлення «Заява відправлена»

Обов’язковим для заповнення полем для обох типів заявки є контактний номер телефону людини, яка заповнює анкету. Номер має формат +380991231212. Якщо номер телефону відсутній або не відповідає необхідному формату, користувач отримає повідомлення про помилку (рис. 3.16).

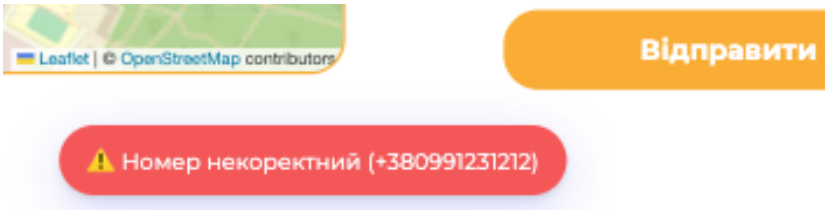

Рисунок 3.16 – Повідомлення «Номер некоректний»

Ще одним обов’язковим елементом для обох типів є наявність мітки на карті, яка вказує місцерозташування загубленої або знайденої тварини. Якщо мітка відсутня, користувач отримає повідомлення про помилку (рис. 3.17).

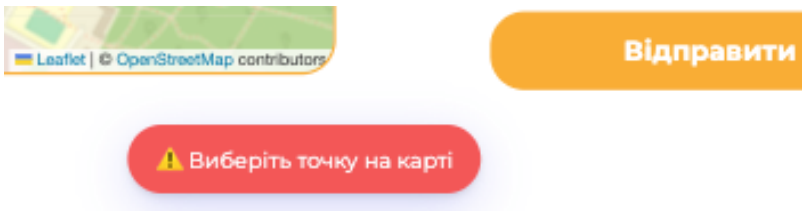

Рисунок 3.17 – Повідомлення «Виберіть точку на карті»

Також незалежно від обраного типу заявки користувач має завантажити фото тварини, яке не перебільшує 5 Мб. Якщо фото перебільшує 5 Мб, воно не буде завантажене. Якщо фото відсутнє, користувач отримає повідомлення про помилку (рис. 3.18).

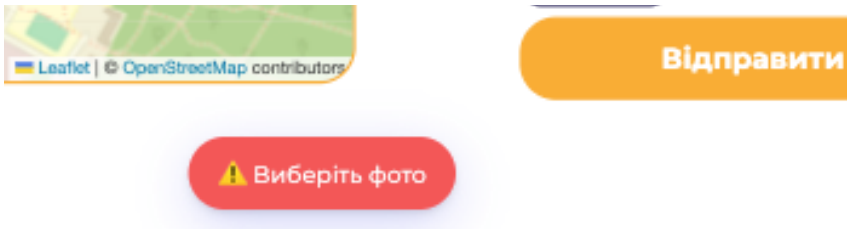

Рисунок 3.18 – Повідомлення «Виберіть фото»

Для типу заявки «Загублено» обов’язковим полем для заповнення є «Прізвисько», яке має містити від 3 до 25 символів. Якщо прізвисько відсутнє або не відповідає необхідному формату, користувач отримає повідомлення про помилку (рис. 3.19).

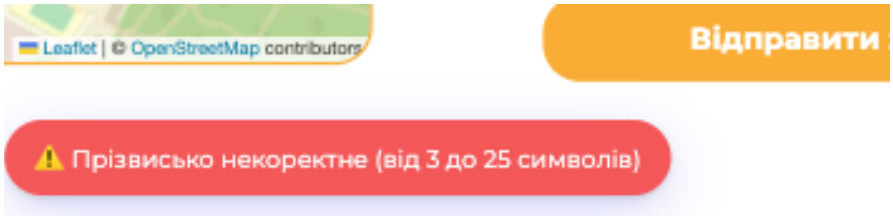

Рисунок 3.19 – Повідомлення «Прізвисько некоректне»

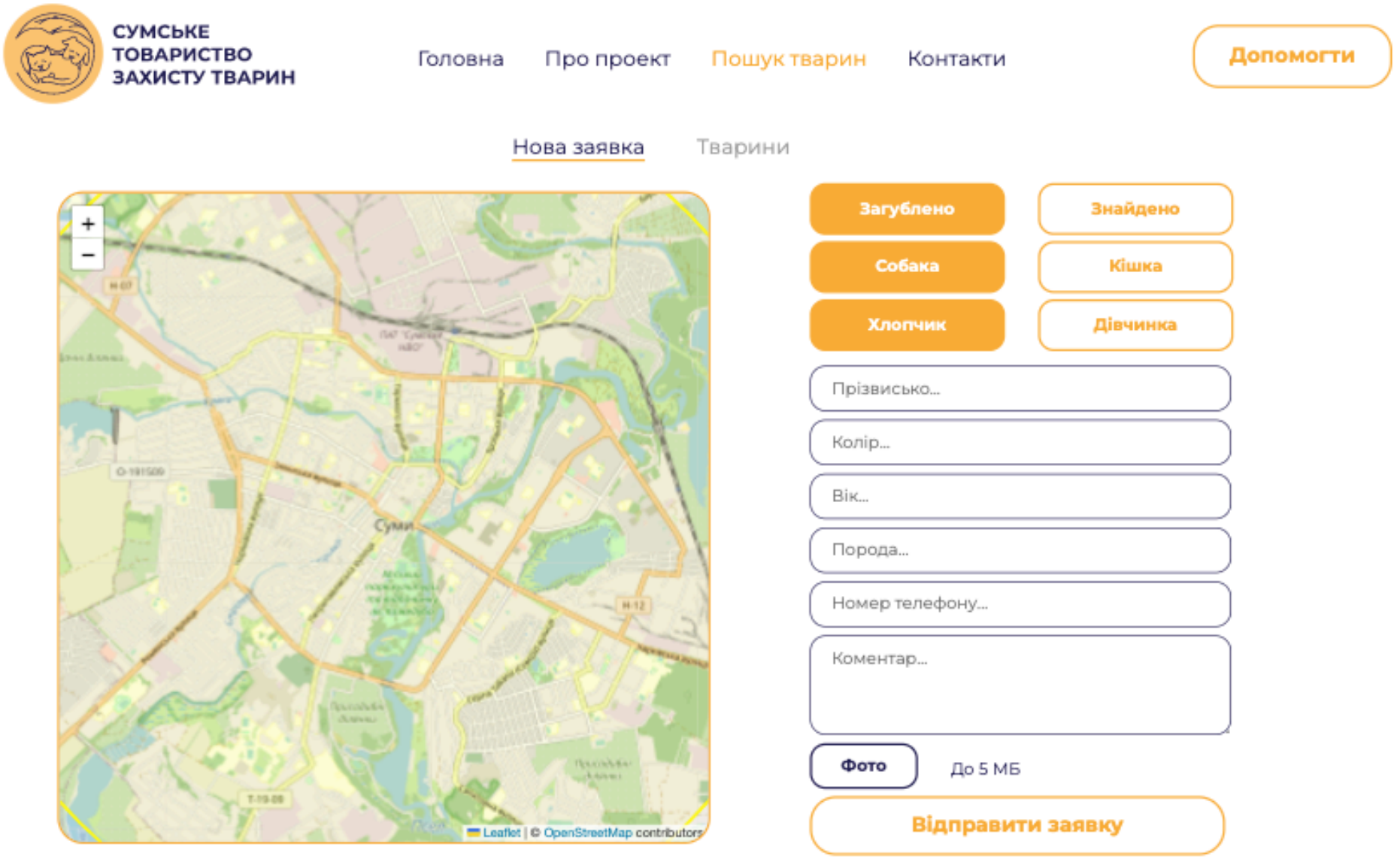

Рисунок 3.20 – Вигляд вкладки «Нова заявка» для типу «Загублено»

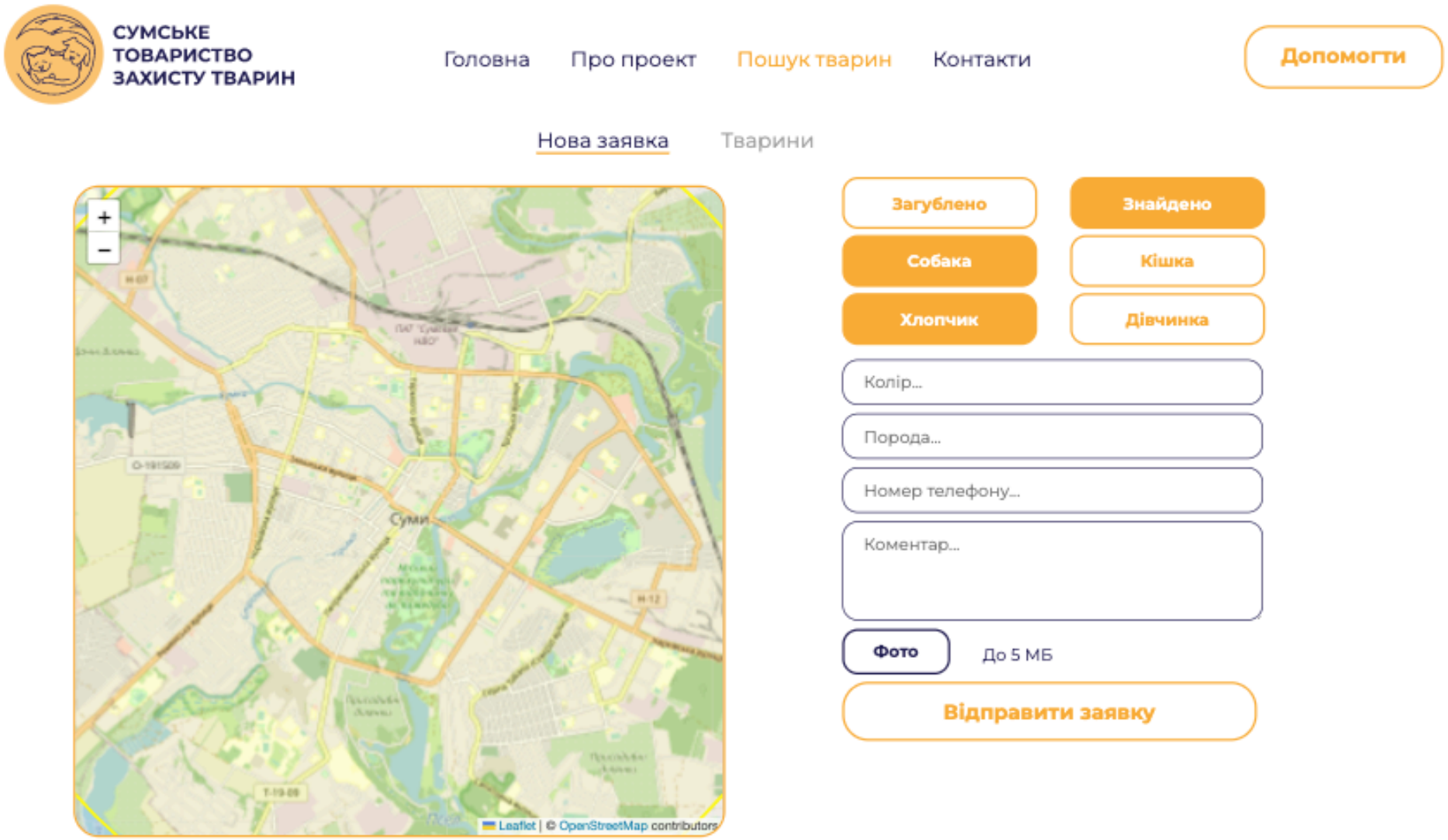

Рисунок 3.21 – Вигляд вкладки «Нова заявка» для типу «Знайдено»

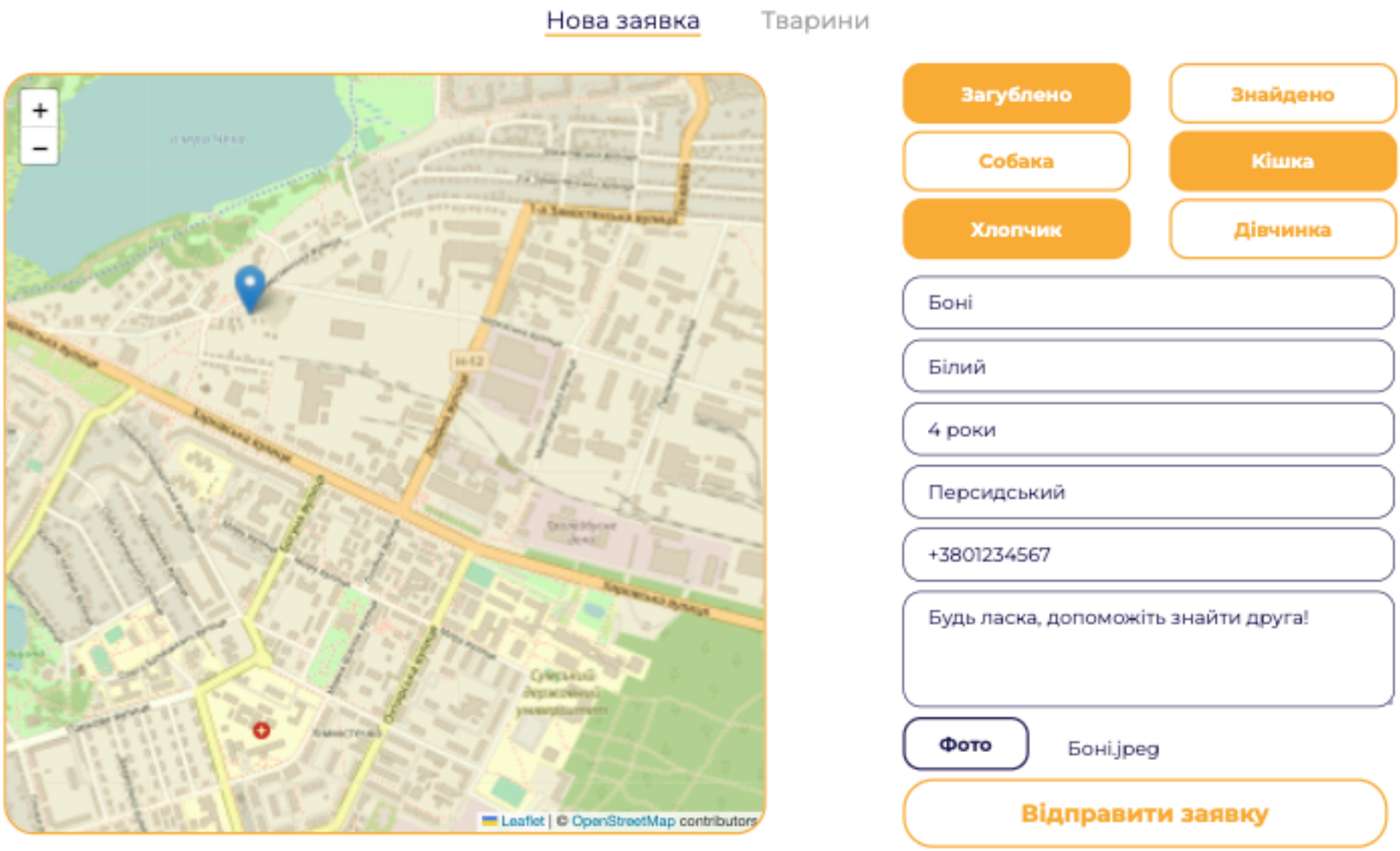

Рисунок 3.22 – Приклад коректного заповнення форми

Вкладка «Тварини» містить в собі карту з маркерами місцезнаходження тварин. При натисканні на маркер на карті відображається стисла фотографія і коментар до заявки, а справа з’являється повна анкета тварини з її даними та повноформатною фотографією (рис. 3.23). Як зазначено раніше, заявки типу «Загублено» відображаються у вигляді червоних маркерів, в той час як заявки типу «Знайдено» мають зелені маркери. Також маркери відрізняються між собою типом тварин, маючи відповідні іконки залежно від того, анкета належить кішці чи собаці.

Розділ «Контакти» одночасно є футером сайту, який містить в собі номер телефону, адресу електронної пошти, фізичну адресу товариства, посилання на соціальні мережі та копірайт (рис. 3.24). Соціальні мережі оформлені у вигляді іконок і при натисканні на одну з них користувач переадресовується на відповідну сторінку.

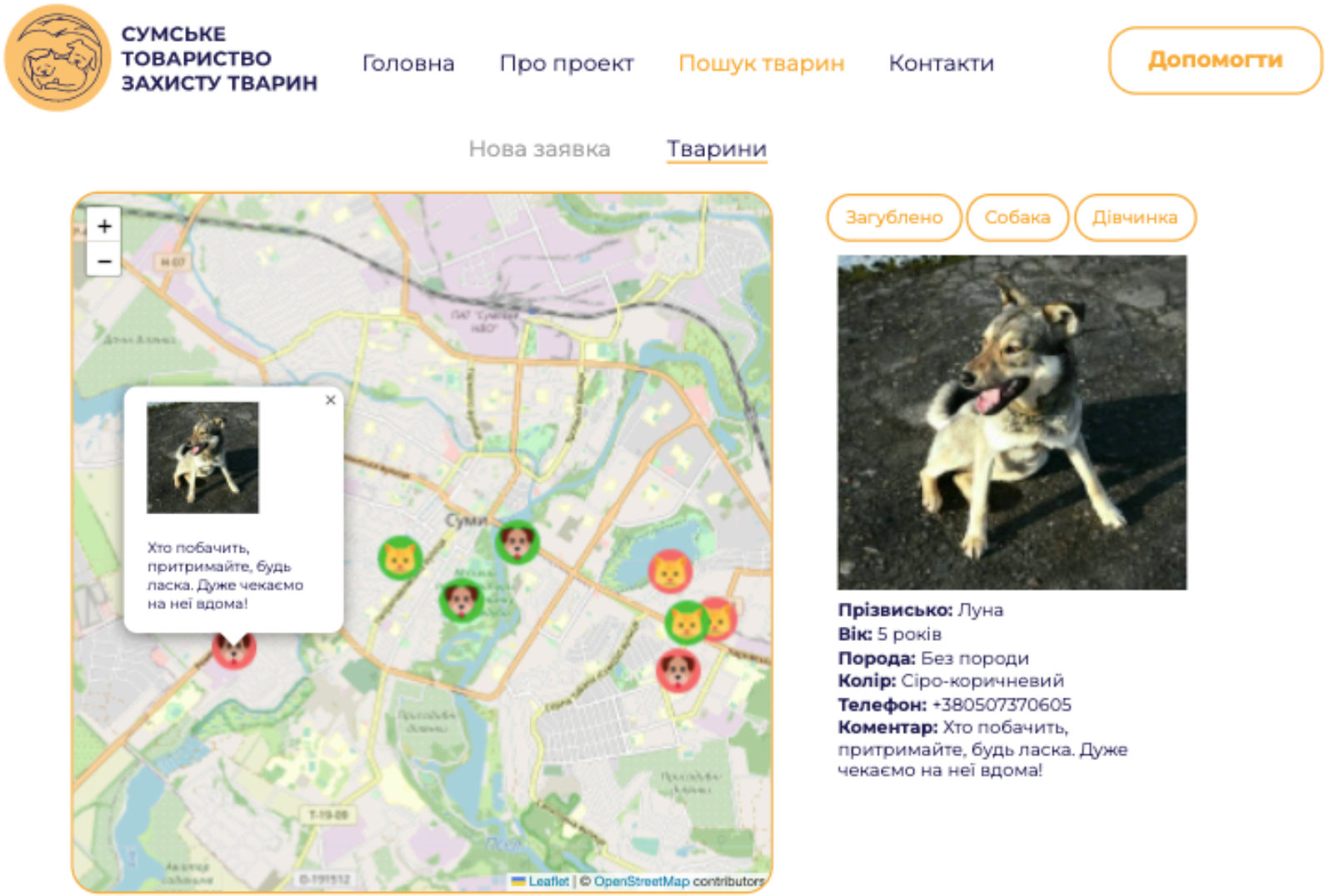

Рисунок 3.23 – Приклад відображення анкети тварини

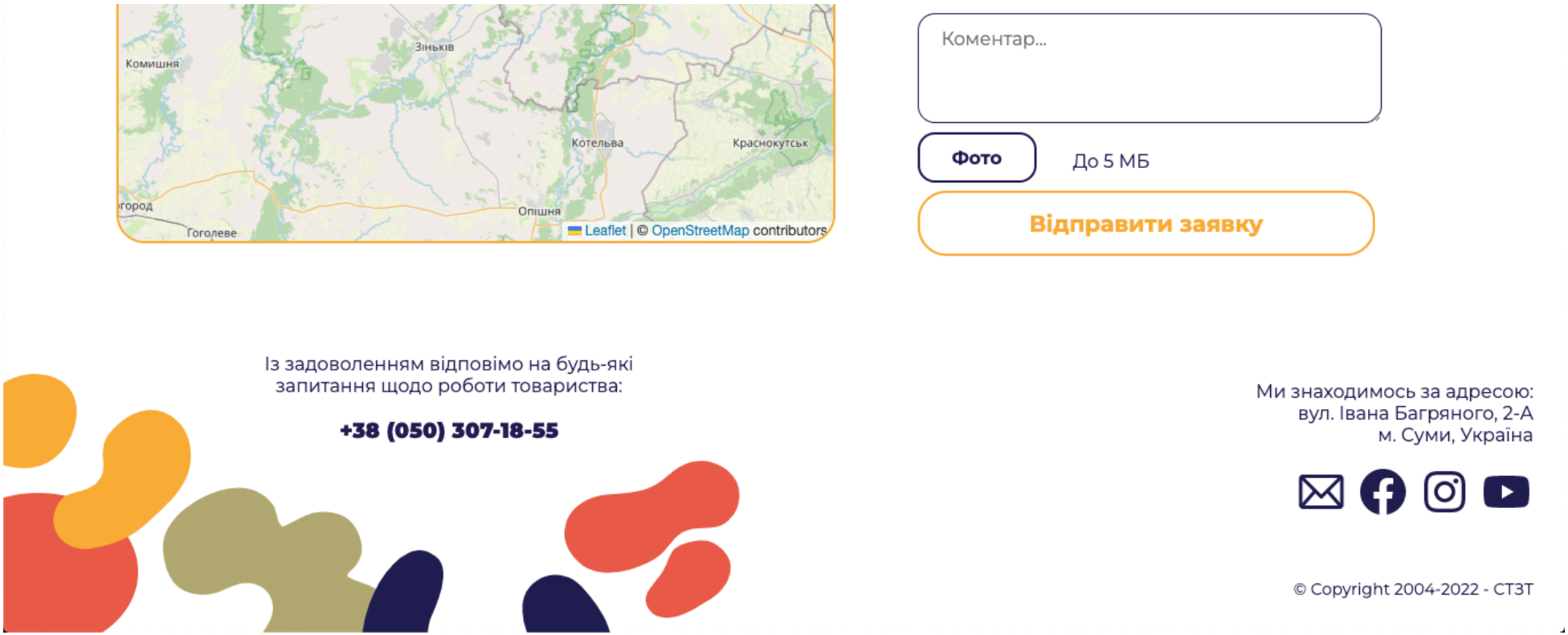

Рисунок 3.24 – Розділ «Контакти» (футер) веб-сайту

Як видно з описаного вище, розроблений інтерфейс є інтуїтивно зрозумілим та легким у використанні. Весь сайт зроблений таким чином, щоб користувач не витрачав багато часу на заповнення заявки або перегляд існуючих анкет. За допомогою картографічного інтерфейсу досягнуто максимальної наочності у відображенні якомога точнішого місцезнаходження тварин.

ВИСНОВКИ

Підсумовуючи хід роботи і процес виконання бакалаврської роботи, в якості отриманих результатів можна перелічити наступне:

1. Доведено актуальність проблеми і вибору теми, виконано детальний аналіз всіх її аспектів і можливих рішень.
2. Проведено літературний огляд сучасних джерел за тематикою розробки веб-сервісів, на основі чого сформовано мету роботи і задачу для виконання.
3. Визначено поняття «веб-сервіс», розглянуто структуру його роботи, можливості і варіанти використання, функції і призначення в бакалаврському проекті і в цілому.
4. Розглянуто доступні і обрано оптимальні інструменти для розробки веб-сервісів, а саме: мову розмітки HTML, мову опису зовнішнього вигляду CSS, мову програмування JavaScript, формат передачі даних JSON, програмну платформу Node.js, фреймворк Express.js, модуль Mongoose, картографічну бібліотеку Leaflet та середу розробки Visual Studio Code, а також проведений їх порівняльний аналіз з існуючими альтернативними рішеннями.
5. Сформульоване технічне завдання і вимоги до веб-сервісу, який має:

- Відображати на карті усіх тварин з бази товариства з виведенням їх місцезнаходження у вигляді маркерів;
- Відображати розгорнуту анкету кожної тварини за умови натискання на відповідний маркер на карті;
- Створювати і зберігати нову заявку після заповнення відповідної форми і натискання кнопки «відправити заявку»;
- Накопичувати і зберігати базу даних, ефективно її систематизувати з можливістю фільтрації за відповідними критеріями;
- Надавати можливість маніпулювати даними з бази, а саме виконувати редагування, видалення і керування заявками.

6. Розроблено сучасний дизайн інтерфейсу, прототипи і макети з урахуванням необхідного функціоналу і потреб користувача.
7. Відповідно до сформульованого технічного завдання програмно реалізовано і впроваджено проект для підтримки діяльності притулку для тварин з використанням картографічного API-інтерфейсу та інших сучасних IT-технологій.

З урахуванням усього вище зазначеного, можна сміливо сказати, що розроблений веб-сервіс значно спрощує процес пошуку старих або нових господарів тварин, сприяє накопиченню нової інформації, оптимізує ведення обліку та зберігання вже існуючих анкет у базі даних і має потенціал до розвитку, а отже стане в нагоді як потенціальним користувачам – мешканцям міста, так і Сумському товариству захисту тварин для досягнення спільної мети – подолання явища безпритульності тварин у місті Суми.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. 19 Bill Gates Quotes About Business And The Real World [Електронний ресурс] – Режим доступу до ресурсу: https://www.inscribd.com/19-bill-gates-quotes-about-business-real-world/.
2. 5 Reasons Why Every Business Needs a Website [Електронний ресурс] – Режим доступу до ресурсу: https://www.beamlocal.com/5-reasons-every-business-needs-a-website/.
3. World Wide Web Consortium [Електронний ресурс] – Режим доступу до ресурсу: https://www.w3.org/.
4. Web Service [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Web_service/.
5. Шелякін П. Концепція веб-сервісів. Реалізація в Java-технологіях [Електронний ресурс] / Павло Шелякін – Режим доступу до ресурсу: http://xmlhack.ru/texts/converted/ws.in.java.tech/WebServiceinJava_axis10.html/.
6. What are Web Services? Architecture, Types, Example [Електронний ресурс] – Режим доступу до ресурсу: https://www.guru99.com/web-service-architecture.html/.
7. What are Web Services? [Електронний ресурс] – Режим доступу до ресурсу: https://www.tutorialspoint.com/webservices/what_are_web_services.htm.
8. Why Web Services? [Електронний ресурс] – Режим доступу до ресурсу: https://www.tutorialspoint.com/webservices/why_web_services.htm/.
9. JavaScript [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/JavaScript/.
10. Why is JavaScript so Important? [Електронний ресурс] – Режим доступу до ресурсу: https://snipcart.com/blog/why-javascript-benefits/.

11. What is HTML: Common uses and defining features [Електронний ресурс] – Режим доступу до ресурсу: https://www.codecademy.com/resources/blog/what-is-html/.
12. What is CSS and Why Should You Use It? [Електронний ресурс] – Режим доступу до ресурсу: https://devmountain.com/blog/what-is-css-and-why-use-it/.
13. Stack Overflow Developer Survey [Електронний ресурс] – Режим доступу до ресурсу: https://insights.stackoverflow.com/survey/2020/.
14. The Reason Why JSON is so Popular [Електронний ресурс] – Режим доступу до ресурсу: https://www.prokurainnovations.com/json/.
15. Node.js at PayPal [Електронний ресурс] – Режим доступу до ресурсу: https://www.joyent.com/blog/node-js-on-the-road-sf-node-js-at-paypal/.
16. Node.js [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Node.js/.
17. W3Techs [Електронний ресурс] – Режим доступу до ресурсу: https://w3techs.com/.
18. Why and When to Use Node.js: A Complete Guide for 2021 [Електронний ресурс] – Режим доступу до ресурсу: https://relevant.software/blog/why-and-when-to-use-node-js/.
19. What is Express JS in Node JS [Електронний ресурс] – Режим доступу до ресурсу: https://www.simplilearn.com/tutorials/nodejs-tutorial/what-is-express-js/.
20. What are the Advantages of Using Mongoose Module? [Електронний ресурс] – Режим доступу до ресурсу: https://www.geeksforgeeks.org/what-are-the-advantages-of-using-mongoose-module/.
21. Stack Overflow [Електронний ресурс] – Режим доступу до ресурсу: https://stackoverflow.com/.
22. Benefit of Using MVC [Електронний ресурс] – Режим доступу до ресурсу: https://www.geeksforgeeks.org/benefit-of-using-mvc/.

23. MongoDB VS SQL [Електронний ресурс] – Режим доступу до ресурсу: https://www.knowi.com/blog/mongodb-vs-sql/.
24. Digging Deeper: Use Case Diagrams [Електронний ресурс] – Режим доступу до ресурсу: http://www.csun.edu/~andrzej/UML/usecase.html/.
25. Use Case Diagram [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Use_case_diagram/.
26. DFD – навіщо вони потрібні і які бувають [Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/ru/post/668684/.
27. Цеслів О.В. Web-програмування : навч. посібнник / О.В. Цеслів ; М-во освіти і науки, молоді та спорту України, Нац. техн. ун-т України «Київ. політехн. ін-т». – Київ : НТУУ «КПІ», 2011. – 296, с.
28. Куленко М.Я. Основи графічного дизайну : підручник для студентів вищих навч. закладів / Михайло Куленко; МОНУ; Київський нац. ун-т будівництва і архітектури. – 2-ге вид., виправл. та доп. – Київ : Кондор, 2007. – 492 с.
29. Бернерс-Лі Заснування павутини = Weaving the web. The original design and ultimate destiny of the world wide web : З чого починалася і до чого прийде Всесвітня мережа / Тім Бернерс-Лі разом з Марком Фічетті; пер. з англ. А. Іщенка. – Київ : Києво-Могилянська академія, 2007. – 208 с.
30. Шмідт Я. Нова мережа: ознаки, практики і наслідки веб 2.0 = Das Neue Nets Markmale, Praktiken und Folgen des Web 2.0 : посібник для вузів / Ян Шмідт ; [пер. з нім. В. Климченко ; за заг. ред. В. Іванова]. – Київ : Академія Української Преси, Центр Вільної Преси, 2013. – 283 с.

ДОДАТОК А

package.json

```
{
    "name": "sspa",
    "version": "1.0.0",
    "description": "Sumy Society Protectional Animal",
    "main": "index.js",
    "scripts": {
        "app": "cd server && node app.js",
        "test": "echo \"Error: no test specified\" && exit 1"
    },
    "author": "semenenko.masha@gmail.com",
    "license": "ISC",
    "dependencies": {
        "body-parser": "^1.20.0",
        "express": "^4.18.1",
        "mongoose": "^6.3.3",
        "multer": "^1.4.4"
    }
}
```

A.1 Server

app.js

```
const express = require('express');
const path = require('path');
const bodyParser = require('body-parser');
const fs = require('fs');
const app = express();
const port = 3000;
const imagesPath = '../views/resources/images';
const multer  = require('multer');
```

```
const upload = multer({ dest: imagesPath });

const mongoose = require('mongoose');
const Config = require('./Config');
const requestController = require('./controllers/requestController');
const shelterController = require('./controllers/shelterController');

app.use('/resources', express.static('../views/resources'));
app.use('/styles', express.static('../views/styles'));
app.use('/scripts', express.static('../views/scripts'));
app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());

mongoose.connect(Config.DBConnectUri);
mongoose.Promise = global.Promise;
let db = mongoose.connection;
db.on('error',    console.error.bind(console,    'MongoDB    connection
error:'));

app.post('/api/loadPhoto', upload.single('file'), (req, res) => {
    const file = req.file;
    let fileExtension = req.file.originalname.split('.').pop();
    fs.rename(imagesPath  +  '/'  +  file.filename,  imagesPath  +  '/'  +
file.filename + '.' + fileExtension, (err) => {
        res.status(400).send();
    });
    res.status(200).send({status: 'ok', fileUrl: 'resources/images/' +
file.filename + '.' + fileExtension});
});

app.post('/api/deleteShelter', (req, res) => {
    if (req.headers.authorization === Config.AdminToken) {
        shelterController.deleteShelter(req.body.id).then(() => {
            res.status(200).send({status: 'ok'});
```

```
        }).catch((err) => {
            res.status(400).send({status: 'error', error: err});
        });
    } else {
        res.status(400).send({status:     'error',     error:     'Не
авторизовано'});
    }
});

app.post('/api/addShelter', (req, res) => {
    if (req.headers.authorization === Config.AdminToken) {
        shelterController.createNew(req.body).then((shelter) => {
            res.status(200).send({status: 'ok', shelter: shelter});
        }).catch((err) => {
            res.status(400).send({status: 'error', error: err});
        });
    } else {
        res.status(400).send({status:     'error',     error:     'Не
авторизовано'});
    }
});

app.get('/api/getShelters', (req, res) => {
    if (req.headers.authorization === Config.AdminToken) {
        shelterController.getAll().then(data => {
            res.status(200).send({status: 'ok', shelters: data});
        }).catch(err => {
            res.status(400).send({status: 'error'});
        });
    } else {
        res.status(400).send({status: 'error'});
    }
});
```

```
app.get('/api/getShelterById', (req, res) => {
    shelterController.getById(req.query.id).then(data => {
        res.status(200).send({status: 'ok', shelter: data});
    }).catch(err => {
        res.status(400).send({status: 'error'});
    });
});

app.post('/api/updateRequest', (req, res) => {
    if (req.headers.authorization === Config.AdminToken) {
        requestController.updateRequest(req.body)
        .then(data => {
            res.status(200).send({status: 'ok'});
        })
        .catch(err => {
            res.status(400).send({status: 'error'});
        });
    } else {
        res.status(401).send({status: 'error'});
    }
});

app.post('/api/deleteRequest', (req, res) => {
    if (req.headers.authorization === Config.AdminToken) {
        requestController.deleteRequest(req.body.id)
        .then(() => {
            res.status(200).send({status: 'ok'});
        })
        .catch((err) => {
            res.status(400).send({status: 'error'});
        });
    } else {
        res.status(401).send({status: 'error'});
    }
```

```
});

app.post('/api/submitRequest', (req, res) => {
    if (req.body) {
        requestController.createNew(req.body).then(() => {
            res.status(200).send({status: 'ok'});
        }).catch((err) => {
            res.status(400).send({err: err});
        });
    } else {
        res.status(400).send({status: 'error'});
    }
});

app.get('/api/getAllRequests', (req, res) => {
    requestController.getAll().then((requests) => {
        res.status(200).send({status: 'ok', requests: requests});
    }).catch((err) => {
        res.status(400).send({status: 'error'});
    });
});

app.post('/api/login', (req, res) => {
    if (req.body.username === Config.AdminLogin && req.body.password ===
Config.AdminPass) {
        res.status(200).send({status: 'ok', token: Config.AdminToken});
    } else {
        res.status(400).send({status: 'error'});
    }
});

app.post('/api/checkToken', (req, res) => {
    if (req.body.token === Config.AdminToken) {
        res.status(200).send({status: 'ok'});
```

```
    } else {
        res.status(401).send({status: 'unauthorized'});
    }
});

app.get('/admin', (req, res) => {
    res.sendFile(path.join(__dirname, '../views/admin.html'));
});

app.get('/', (req, res) => {
    res.sendFile(path.join(__dirname, '../views/index.html'));
});

app.listen(port, () => {
    console.log(`App listening on port ${port}`)
})
```

Config.js

```
const Config = {
    //db data
    DBConnectUri:                                'mongodb+srv://msmnnk:-9-
ifCZm6azXQHp@cluster0.zt4qc.mongodb.net/SumyPets?retryWrites=true&w=ma
jority',
    DBName: 'SumyPets',
    //admin credentials
    AdminLogin: 'admin',
    AdminPass: 'admin',
    AdminToken: '123456789',
}
module.exports = Config;
```

/controllers

requestController.js

```
const Request = require('../models/Request');
```

```
const RequestController = {
    createNew(requestData) {
        let request = new Request(requestData);
        return request.save();
    },
    getAll() {
        return Request.find({});
    },
    updateRequest(requestData) {
        return Request.findByIdAndUpdate(requestData._id, requestData);
    },
    deleteRequest(id) {
        return Request.findByIdAndRemove(id);
    }
}

module.exports = RequestController;
```

shelterController.js

```
const Shelter = require('../models/Shelter');
const ShelterController = {
    getById(id) {
        return Shelter.findById(id);
    },
    createNew(shelterData) {
        let shelter = new Shelter(shelterData);
        return shelter.save();
    },
    getAll() {
        return Shelter.find({});
    },
    deleteShelter(id) {
        return Shelter.findByIdAndRemove(id);
    }
}
```

```
module.exports = ShelterController;
```

/models

Request.js

```
const mongoose = require("mongoose");
const Schema = mongoose.Schema;

let RequestSchema = new Schema({
    reqType: {
        type: String,
        enum : ['Загублено','Знайдено'],
        required: true
    },
    kind: {
        type: String,
        enum : ['Собака','Кішка'],
        required: true
    },
    sex: {
        type: String,
        enum : ['Хлопчик','Дівчинка'],
        required: true
    },
    name: {
        type: String,
        default: null
    },
    color: {
        type: String,
        default: null
    },
    age: {
        type: String,
```

```
        default: null
    },
    breed: {
        type: String,
        default: null
    },
    number: {
        type: String,
        default: null
    },
    description: {
        type: String,
        default: null
    },
    coordinates: {
        lat: Number,
        lng: Number
    },
    photo: {
        type: String,
        required: true
    },
    shelter: {
        type: Schema.Types.ObjectId,
        ref: 'Shelter',
        default: null
    }
},{ versionKey: false });

module.exports = mongoose.model('Request', RequestSchema);
```

Shelter.js

```
const mongoose = require("mongoose");
const Schema = mongoose.Schema;
```

```
let ShelterSchema = new Schema({
    name: {
        type: String,
        required: true
    },
    address: {
        type: String,
        required: true
    }
},{ versionKey: false });

module.exports = mongoose.model('Shelter', ShelterSchema);
```

A.2 Views

admin.html

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <title>SAPS Admin</title>
        <link rel="stylesheet" href="styles/admin.css">
        <link          rel="apple-touch-icon"          sizes="180x180"
href="resources/favicons/apple-touch-icon.png">
        <link      rel="icon"      type="image/png"      sizes="32x32"
href="resources/favicons/favicon-32x32.png">
        <link      rel="icon"      type="image/png"      sizes="16x16"
href="resources/favicons/favicon-16x16.png">
    </head>
    <body>
        <div id="loginForm" style="display: none">
            <form onsubmit="login(event)" class="login-form">
                <h4>LOGIN</h4>
                <input           type="text"           name="username"
placeholder="Username" required>
```

```
                <input          type="password"          name="password"
placeholder="Password" required>
                <input type="submit" value="Login">
            </form>
        </div>
        <div id="content" style="display: none">
            <div class="head">
                <div class="head-label">SAPS Admin Panel</div>
                <button onclick="logout()">Logout</button>
            </div>
            <h4>Requests</h4>
            <div class="body">
                <div id="requestsList">
                </div>
                <div id="infoForm">
                </div>
            </div>
        </div>
    </body>
    <script src="/scripts/admin.js" type="text/javascript"></script>
</html>
```

index.html

```
<!DOCTYPE html>
<html lang="en">

<head>
    <title>Сумське товариство захисту тварин</title>
    <link                                                rel="stylesheet"
href="https://unpkg.com/leaflet@1.8.0/dist/leaflet.css"
        integrity="sha512-
hoalWLoI8r4UszCkZ5kL8vayOGVae1oxXe/2A4AO6J9+580uKHDO3JdHb7NzwwzK5xr/Fs
0W40kiNHxM9vyTtQ=="
        crossorigin="" />
```

```
    <link               rel="stylesheet"               type="text/css"
href="https://cdn.jsdelivr.net/npm/toastify-js/src/toastify.min.css">
    <link rel="stylesheet" href="styles/index.css">
    <link rel="stylesheet" href="styles/header.css">
    <link rel="stylesheet" href="styles/main.css">
    <link rel="stylesheet" href="styles/searchPet.css">
    <link rel="stylesheet" href="styles/cards.css">
    <link rel="stylesheet" href="styles/footer.css">
    <link rel="stylesheet" href="styles/infoForm.css">
    <link            rel="apple-touch-icon"            sizes="180x180"
href="resources/favicons/apple-touch-icon.png">
    <link       rel="icon"       type="image/png"       sizes="32x32"
href="resources/favicons/favicon-32x32.png">
    <link       rel="icon"       type="image/png"       sizes="16x16"
href="resources/favicons/favicon-16x16.png">
</head>

<body>
    <header id="stickyHeader">
        <div class="logo-container">
            <img src="resources/logo.svg">
        </div>
        <div class="menu-container">
            <ul id="menu">
                <li><a href="#main">Головна</a></li>
                <li><a href="#about">Про проект</a></li>
                <li><a href="#search">Пошук тварин</a></li>
                <li><a href="#contact">Контакти</a></li>
            </ul>
        </div>
        <div class="button-container">
            <a   class="button"   href="https://next.privat24.ua/money-
transfer/form/%7B%22form%22:%7B%22receiver%22:%7B%22source%22:%22manua
```

```
l%22,%22number%22:%2253635420159505 96%22%7D,%22amount%22:%220%22,%22cu
rrency%22:%22UAH%22%7D%7D">Допомогти</a>
        </div>
    </header>

    <div id="main">
        <div id="main-text">
            <h2>Не у кожної людини має бути тварина, але у кожної тварини
має бути відповідальна <span
                    class="colortext">людина</span></h2>
            <p>Сумське товариство захисту тварин є некомерційною
організацією, яка не фінансується органами державної
                влади
                та реалізує свої цілі виключно за рахунок членських
внесків та благодійних пожертвувань.</p>
        </div>
        <div id="illustration">
            <img                                class="main-illustration"
src="resources/illustration.svg">
        </div>
    </div>

    <div id="about">
        <div class='card-container'>
            <div class="card">
                <img class="cardimage" src="resources/card_1.svg">
                <h2 class="cardheader">Пошук тварин</h2>
                <p  class="cardtext">Ми  знаходимо  старих  та  нових
власників для чотирилапих друзів або членів родини
                    для відповідальних людей
                </p>
            </div>
            <div class="card">
                <img class="cardimage" src="resources/card_2.svg">
```

```
                <h2 class="cardheader">Прилаштування</h2>
                <p class="cardtext">Ми пропонуємо послуги перетримання,
а також допомогу з
                    прилаштування улюбленців на ринку</p>
            </div>
            <div class="card">
                <img class="cardimage" src="resources/card_3.svg">
                <h2 class="cardheader">Стерилізація</h2>
                <p  class="cardtext">Ми  домовляємось  щодо  мінімальної
вартості операції з
                    найдосвідченішими ветеринарними фахівцями</p>
            </div>
        </div>
        <div class='card-container'>
            <div class="card">
                <img class="cardimage" src="resources/card_4.svg">
                <h2 class="cardheader">Розміщення інформації</h2>
                <p  class="cardtext">Ми  допомогаємо  шляхом  розміщення
оголошень, підписані понад 12 000 осіб</p>
            </div>
            <div class="card">
                <img class="cardimage" src="resources/card_5.svg">
                <h2 class="cardheader">Юридична допомога</h2>
                <p  class="cardtext">Ми  надаємо  юридичну  допомогу  та
консультації з питань зоозахисту</p>
            </div>
            <div class="card">
                <img class="cardimage" src="resources/card_6.svg">
                <h2 class="cardheader">Лекції в закладах освіти</h2>
                <p class="cardtext">Ми проводимо лекції студентам, уроки
доброти школярам та презентації</p>
            </div>
        </div>
    </div>
```

```
    <div id="search" class="searchDescription">
        <img id="gifCat" src="resources/cat.gif">
        <div id="searchText">
            <h2>Пошук тварин</h2>
            <p>Найважливішим аспектом в питанні зоозахисту є проблема безпритульних тварин. Саме для її вирішення ми
                розробили наш сервіс пошуку чотирилапих улюбленців. Загубили свого друга чи випадково знайшли
                чийогось?
                Заповніть форму і допоможіть створінню знайти свій дім! Надсилайте будь-яку інформацію, яка може
                стати у
                нагоді, а також не забудьте поділитися своїми контактними даними для зв’язку. Разом ми зможемо
                викоренити
                явище “безпритульності” в місті Суми!</p>
        </div>
    </div>
    <div class="search">
        <div id="tabBar" onclick="tabHandler(event)">
            <div id="newRequest" class="tab active">Нова заявка</div>
            <div id="animals" class="tab">Тварини</div>
        </div>
        <div id="newRequestTab">
            <div id="map"></div>
            <form class="form" onsubmit="submitHandler(event)">
                <div class="selector" id="form-status" onclick="selectorHandler(event)">
                    <input class="formButton selected" type="button" value="Загублено">
                    <input class="formButton" type="button" value="Знайдено">
                </div>
```

```
                <div          class="selector"          id="form-kind"
onclick="selectorHandler(event)">
                    <input  class="formButton  selected"  type="button"
value="Собака">
                    <input       class="formButton"       type="button"
value="Кішка">
                </div>
                <div           class="selector"           id="form-sex"
onclick="selectorHandler(event)">
                    <input  class="formButton  selected"  type="button"
value="Хлопчик">
                    <input       class="formButton"       type="button"
value="Дівчинка">
                </div>
                <input            class="field"            type="text"
placeholder="Прізвисько..." id="form-name">
                <input class="field" type="text" placeholder="Колір..."
id="form-color">
                <input  class="field"  type="text"  placeholder="Вік..."
id="form-age">
                <input            class="field"            type="text"
placeholder="Порода..." id="form-breed">
                <input   class="field"   type="tel"   placeholder="Номер
телефону..." id="form-number">
                <textarea    class="field"    placeholder="Коментар..."
id="form-comment" rows="5" cols="33"></textarea>
                <div class="file-selector">
                    <input         type="file"         id="form-photo"
accept="image/png,          image/jpeg"          style="display:none;"
onchange="fileHandler(event)">
                    <label     for="form-photo"     class="custom-file-
input"></label>
                    <p id="file-name">До 5 МБ</p>
                </div>
```

```
                <input    id="submit"    class="button"    type="submit"
value="Відправити заявку">
            </form>
        </div>
        <div id="animalsTab">
            <div id="animalMap"></div>
            <div id="animalForm"></div>
        </div>
    </div>

    <footer id="contact">
        <div id="left-container">
            <div id="phone">
                <p>Із задоволенням відповімо на будь-які<br>запитання
щодо роботи товариства:</p>
                <h3>+38 (050) 307-18-55</h3>
            </div>
            <img id="footerImage" src="resources/footer.svg">
        </div>
        <div id="right-container">
            <p>Ми знаходимось за адресою: <br> вул. Івана Багряного, 2-
А <br> м. Суми, Україна</p>
            <ul id="socialmedia">
                <li><a      href="mailto:info@animal-home.sumy.ua"><img
src="resources/email.svg"></a></li>
                <li><a
href="https://www.facebook.com/sumy.animal.home/"><img
src="resources/facebook.svg"></a></li>
                <li><a
href="https://www.instagram.com/helping_animals_in_sumy/"><img
src="resources/instagram.svg"></a></li>
                <li><a
href="https://www.youtube.com/channel/UCO4_oflHZ_WfCexAtDKSBzw"><img
src="resources/youtube.svg"></i></a></li>
```

```
            </ul>
            <br>
            <p id="copyright">© Copyright 2004-2022 - CT3T</p>
        </div>
    </footer>
</body>

<script src="https://unpkg.com/leaflet@1.8.0/dist/leaflet.js"
    integrity="sha512-
BB3hKbKWOc9Ez/TAwyWxNXeoV9c1v6FIeYiBieIWkpLjauysF18NzgR1MBNBXf8/KABdlk
X68nAhlwcDFLGPCQ=="
    crossorigin=""></script>
<script                                          type="text/javascript"
src="https://cdn.jsdelivr.net/npm/toastify-js"></script>
<script src="/scripts/requests.js" type="text/javascript"></script>
<script src="/scripts/tab.js" type="text/javascript"></script>
<script src="/scripts/helper.js" type="text/javascript"></script>
<script src="/scripts/map.js" type="text/javascript"></script>
<script src="/scripts/form.js" type="text/javascript"></script>
</html>
```

/scripts

admin.js

```
let requests = [];
let shelters = [];
let currentRequest = null;

window.onload = function() {
    checkToken();
    fillList();
    getShelters();
}

function getShelters() {
```

```
    fetch('api/getShelters', {
        method: 'GET',
        headers: {
            'Content-Type': 'application/json',
            'authorization': localStorage.getItem('token')
            }
        })
        .then(response => response.json())
        .then(data => {
            shelters = data.shelters;
            if (currentRequest !== null) {
                fillForm(currentRequest);
            }
        })
        .catch(error => console.error(error)
    );
}

function fillList() {
    fetch('/api/getAllRequests', {
        method: 'GET',
        headers: {
            'Content-Type': 'application/json'
        }
    }).then(response => response.json())
    .then(data => {
        if (data.status === 'ok') {
            requests = data.requests;
            let list = document.getElementById('requestsList');
            list.innerHTML = '';
            requests.forEach(request => {
                addListItem(list, request);
            });
        }
```

```
    }).catch((err) => {
        alert('Connection error');
    });
}

function addListItem(list, request) {
    let listItem = document.createElement('div');
    listItem.classList.add('list-item');
    listItem.id = request._id;

    let itemImage = document.createElement('img');
    itemImage.src = request.photo;
    itemImage.classList.add('list-item-image');

    let itemKind = document.createElement('div');
    itemKind.classList.add('list-item-kind');
    itemKind.innerText = request.kind === 'Собака' ? '🐶' : '🐱';

    let itemSex = document.createElement('div');
    itemSex.classList.add('list-item-sex');
    itemSex.innerText = request.sex === 'Хлопчик' ? '🚹' : '🚺';

    let itemPhone = document.createElement('div');
    itemPhone.classList.add('list-item-phone');
    itemPhone.innerText = request.number;

    let deleteItem = document.createElement('div');
    deleteItem.classList.add('list-item-delete-btn');
    deleteItem.innerText = '❌';
    deleteItem.onclick = (event) => {
        deleteRequest(event.target.parentElement.id);
    }

    let infoContainer = document.createElement('div');
```

```
    infoContainer.classList.add('list-item-info-container');
    infoContainer.onclick = (event) => {
        openRequest(event.target);
    }

    infoContainer.appendChild(itemImage);
    infoContainer.appendChild(itemKind);
    infoContainer.appendChild(itemSex);
    infoContainer.appendChild(itemPhone);
    listItem.appendChild(infoContainer);
    listItem.appendChild(deleteItem);
    list.appendChild(listItem);
}

function openRequest(target) {
    let requestId;
    if (target.classList.contains('list-item-info-container')) {
        requestId = target.parentElement.id;
    } else {
        requestId = target.parentElement.parentElement.id;
    }
    let request = requests.find(request => request._id === requestId);
    currentRequest = request;
    fillForm(request);
}

function deleteRequest(id) {
    if (confirm('delete item?')) {
        fetch('/api/deleteRequest', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json',
                'authorization': localStorage.getItem('token')
            },
```

```
            body: JSON.stringify({
                id: id
            })
        }).then(response => response.json())
        .then(data => {
            if (data.status === 'ok') {
                fillList();
            }
        }).catch((err) => {
            alert('Connection error');
        });
    }
}

function updateRequest() {
    let request = requests.find(request => request._id ===
document.getElementById('infoForm').getAttribute('data-id'));
    request.reqType = document.getElementsByClassName('form-
type')[0].value;
    request.kind = document.getElementsByClassName('form-
kind')[0].value;
    request.sex = document.getElementsByClassName('form-sex')[0].value;
    request.name = document.getElementsByClassName('form-
name')[0].value;
    request.age = document.getElementsByClassName('form-age')[0].value;
    request.breed = document.getElementsByClassName('form-
breed')[0].value;
    request.color = document.getElementsByClassName('form-
color')[0].value;
    request.number = document.getElementsByClassName('form-
phone')[0].value;
    request.description = document.getElementsByClassName('form-
description')[0].value;
```

```
    request.shelter      =      document.getElementsByClassName('form-
shelter')[0].value;

    fetch('/api/updateRequest', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json',
            'authorization': localStorage.getItem('token')
        },
        body: JSON.stringify(request)
    }).then(response => response.json())
    .then(data => {
        if (data.status === 'ok') {
            fillList();
        }
    }).catch((err) => {
        alert('Connection error');
    });
}

function fillForm(request) {
    let form = document.getElementById('infoForm');
    form.setAttribute('data-id', request._id);
    form.innerHTML = '';
    let formTitle = document.createElement('h2');
    formTitle.innerText = 'Заявка № ' + request._id;
    form.appendChild(formTitle);

    let formPhoto = document.createElement('img');
    formPhoto.src = request.photo;
    formPhoto.classList.add('form-photo');
    form.appendChild(formPhoto);

    let formType = document.createElement('select');
```

```
    formType.classList.add('form-type');
    let lostOption = document.createElement('option');
    lostOption.value = 'Загублено';
    lostOption.innerText = 'Загублено';
    let foundOption = document.createElement('option');
    foundOption.value = 'Знайдено';
    foundOption.innerText = 'Знайдено';
    formType.appendChild(lostOption);
    formType.appendChild(foundOption);
    form.appendChild(formType);
    formType.value = request.reqType;

    let formKind = document.createElement('select');
    formKind.classList.add('form-kind');
    let DogOption = document.createElement('option');
    DogOption.value = 'Собака';
    DogOption.innerText = 'Собака';
    let CatOption = document.createElement('option');
    CatOption.value = 'Кішка';
    CatOption.innerText = 'Кішка';
    formKind.appendChild(DogOption);
    formKind.appendChild(CatOption);
    form.appendChild(formKind);
    formKind.value = request.kind;

    let formSex = document.createElement('select');
    formSex.classList.add('form-sex');
    let maleOption = document.createElement('option');
    maleOption.value = 'Хлопчик';
    maleOption.innerText = 'Хлопчик';
    let femaleOption = document.createElement('option');
    femaleOption.value = 'Дівчинка';
    femaleOption.innerText = 'Дівчинка';
    formSex.appendChild(maleOption);
```

```
    formSex.appendChild(femaleOption);
    form.appendChild(formSex);
    formSex.value = request.sex

    let formName = document.createElement('input');
    formName.classList.add('form-name');
    formName.name = 'name';
    formName.value = request.name;
    formName.placeholder = 'name'
    form.appendChild(formName);

    let formAge = document.createElement('input');
    formAge.classList.add('form-age');
    formAge.name = 'age';
    formAge.value = request.age;
    formAge.placeholder = 'age'
    form.appendChild(formAge);

    let formBreed = document.createElement('input');
    formBreed.classList.add('form-breed');
    formBreed.name = 'breed';
    formBreed.value = request.breed;
    formBreed.placeholder = 'breed'
    form.appendChild(formBreed);

    let formColor = document.createElement('input');
    formColor.classList.add('form-color');
    formColor.name = 'color';
    formColor.value = request.color;
    formColor.placeholder = 'color'
    form.appendChild(formColor);

    let formPhone = document.createElement('input');
    formPhone.classList.add('form-phone');
```

```
    formPhone.name = 'phone';
    formPhone.value = request.number;
    formPhone.placeholder = 'phone'
    form.appendChild(formPhone);

    let formDescription = document.createElement('textarea');
    formDescription.classList.add('form-description');
    formDescription.name = 'description';
    formDescription.value = request.description;
    formDescription.placeholder = 'description'
    form.appendChild(formDescription);

    let formShelter = document.createElement('select');
    formShelter.classList.add('form-shelter');
    for (let shelter of shelters) {
        let option = document.createElement('option');
        option.value = shelter._id;
        option.innerText = shelter.name + ' ' + shelter.address;
        formShelter.appendChild(option);
    }
    formShelter.value = request.shelter;
    let formShelterButton = document.createElement('button');
    formShelterButton.classList.add('form-shelter-button');
    formShelterButton.innerText = 'Перетримки';
    formShelterButton.addEventListener('click', () => {
        openShelters();
    });

    let formSheltercontainer = document.createElement('div');
    formSheltercontainer.classList.add('form-sheltercontainer');

    formSheltercontainer.appendChild(formShelter);
    formSheltercontainer.appendChild(formShelterButton);
    form.appendChild(formSheltercontainer);
```

```
    let sheltersList = document.createElement('div');
    sheltersList.id = 'sheltersList';
    sheltersList.style.display = 'none';
    form.appendChild(sheltersList);

    let formSubmit = document.createElement('input');
    formSubmit.type = 'submit';
    formSubmit.value = 'Зберегти';
    formSubmit.classList.add('form-submit');
    formSubmit.onclick = (event) => {
        event.preventDefault();
        updateRequest();
    };
    form.appendChild(formSubmit);
}

function openShelters() {
    let sheltersList = document.getElementById('sheltersList');
    if (sheltersList.style.display === 'none') {
        sheltersList.style.display = 'flex';
        sheltersList.innerHTML = '';
        let list = document.createElement('div');
        list.classList.add('shelters-list');
        for (let shelter of shelters) {
            let shelterItem = document.createElement('div');
            shelterItem.classList.add('shelter-item');
            shelterItem.innerText = shelter.name + ' / ' +
shelter.address;
            let deleteButton = document.createElement('button');
            deleteButton.classList.add('delete-button');
            deleteButton.innerText = '❌';
            deleteButton.addEventListener('click', (event) => {
                deleteShelter(shelter._id, event.target.parentElement);
```

```
        });
        shelterItem.appendChild(deleteButton);
        list.appendChild(shelterItem);
        sheltersList.appendChild(list);
    }

    let form = document.createElement('div')
    form.classList.add('shelters-form');

    let formName = document.createElement('input');
    formName.classList.add('form-name');
    formName.id = 'shelterName';
    formName.name = 'name';
    formName.placeholder = 'name'
    form.appendChild(formName);

    let formAddress = document.createElement('input');
    formAddress.classList.add('form-address');
    formAddress.id = 'formAddress';
    formAddress.name = 'address';
    formAddress.placeholder = 'address'
    form.appendChild(formAddress);

    let formSubmit = document.createElement('input');
    formSubmit.type = 'submit';
    formSubmit.value = 'Додати';
    formSubmit.classList.add('form-submit');
    formSubmit.onclick = (event) => {
        event.preventDefault();
        addShelter();
    };
    form.appendChild(formSubmit);

    sheltersList.appendChild(form);
```

```
    } else {
        sheltersList.style.display = 'none';
    }
}

function addShelter() {
    let name = document.getElementById('shelterName').value;
    let address = document.getElementById('formAddress').value;
    let shelter = {
        name: name,
        address: address
    };
    fetch('/api/addShelter', {
        method: 'POST',
        body: JSON.stringify(shelter),
        headers: {
            'Content-Type': 'application/json',
            'authorization': localStorage.getItem('token')
        }
    }).then(() => {
        getShelters();
    }).catch(error => {
        alert(error);
    });
}

function deleteShelter(id, element) {
    if (confirm('Ви впевнені, що хочете видалити перетримку?')) {
        fetch('/api/deleteShelter', {
            method: 'POST',
            body: JSON.stringify({
                id: id
            }),
            headers: {
```

```
                'Content-Type': 'application/json',
                'authorization': localStorage.getItem('token')
            }
        }).then(() => {
            element.remove();
            let    select    =    document.getElementsByClassName('form-
shelter')[0].options;
            for (let i = 0; i < select.length; i++) {
                if (select[i].value === id) {
                    select[i].remove();
                }
            }
        }).catch(error => {
            alert(error);
        });
    }
}

function login(event) {
    event.preventDefault();
    let form = new FormData(event.target);
    fetch('/api/login', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify({
                username: form.get('username'),
                password: form.get('password')
            })
    }).then(response => response.json())
    .then(data => {
        if (data.status === 'ok') {
            localStorage.setItem('token', data.token);
```

```
            showContent();
        } else {
            alert('Wrong username or password');
        }
    }).catch((err) => {
        alert('Connection error');
    });
}

function logout() {
    localStorage.removeItem('token');
    showLoginForm();
}

function checkToken(){
    let token = localStorage.getItem('token');
    if (token) {
        fetch('/api/checkToken', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json'
            },
            body: JSON.stringify({token: token})
        }).then(response => response.json())
        .then(data => {
            if (data.status === 'ok') {
                showContent();
            } else {
                showLoginForm();
            }
        }).catch((err) => {
            showLoginForm();
        });
    } else {
```

```
        showLoginForm();
    }
}

function showLoginForm() {
    document.getElementById('loginForm').style.display = 'flex';
    document.getElementById('content').style.display = 'none';
}

function showContent() {
    document.getElementById('loginForm').style.display = 'none';
    document.getElementById('content').style.display = 'flex';
}
```

form.js

```
function selectorHandler(event) {
    if (event.target.tagName === 'INPUT') {
        let activeElem =
event.target.parentElement.getElementsByClassName("selected")[0];
        activeElem.classList.remove('selected');
        event.target.classList.add('selected');
        if (event.target.value === 'Знайдено') {
            document.getElementById('form-name').hidden = true;
            document.getElementById('form-age').hidden = true;
        } else if (event.target.value === 'Загублено') {
            document.getElementById('form-name').hidden = false;
            document.getElementById('form-age').hidden = false;
            document.getElementById('form-name').value = '';
            document.getElementById('form-color').value = '';
        }
    }
}

function getSelectedValue(id) {
```

```
    let selected =
document.getElementById(id).getElementsByClassName("selected")[0];
    return selected.value;
}

function getFormData() {
    let formData = {};
    formData.reqType = getSelectedValue('form-status');
    formData.kind = getSelectedValue('form-kind');
    formData.sex = getSelectedValue('form-sex');
    formData.name = document.getElementById('form-name').value;
    formData.color = document.getElementById('form-color').value;
    formData.age = document.getElementById('form-age').value;
    formData.breed = document.getElementById('form-breed').value;
    formData.number = document.getElementById('form-number').value;
    formData.description = document.getElementById('form-
comment').value;
    formData.coordinates = getSelectedCoordinates();
    formData.photo = document.getElementById('form-photo').files[0] ||
null;
    return formData;
}

function clearForm() {
    document.getElementById('form-name').value = '';
    document.getElementById('form-color').value = '';
    document.getElementById('form-age').value = '';
    document.getElementById('form-breed').value = '';
    document.getElementById('form-number').value = '';
    document.getElementById('form-comment').value = '';
    removeSelectedCoordinates();
    document.getElementById('form-photo').value = null;
    document.getElementById('file-name').innerHTML = 'До 5 МБ';
}
```

```
function fileHandler(event) {
    let file = event.target.files[0];
    if (file.size / 1024 / 1024 < 5) {
        document.getElementById('file-name').innerHTML = file.name;
    } else {
        formData.photo = document.getElementById('form-photo').value =
null;
        Toastify({
            text: '⚠ Фото завелике, виберіть фото менше 5 Мб',
            duration: 3000,
            gravity: 'bottom',
            position: 'center',
            stopOnFocus: true,
            className: 'toast-error'
        }).showToast();
    }
}

function submitHandler(event) {
    event.preventDefault();
    let formData = getFormData();
    if (validateForm(formData)) {
        let form = new FormData()
        form.append('file', formData.photo);
        fetch('api/loadPhoto', {
            method: 'POST',
            body: form
        })
        .then(response => response.json())
        .then(data => {
            formData.photo = data.fileUrl;
            fetch('api/submitRequest', {
                method: 'POST',
```

```
                headers: {
                    'Content-Type': 'application/json'
                },
                body: JSON.stringify(formData)
            })
            .then(response => response.json())
            .then(data => {
                clearForm();
                Toastify({
                    text: '✅ Заявка відправлена',
                    duration: 2000,
                    gravity: 'bottom',
                    position: 'center',
                    stopOnFocus: true,
                    className: 'toast-success'
                }).showToast();
            })
            .catch(error => {
                showValidationError('Щось пішло не так :(');
            });
        })
        .catch(error => {
            showValidationError('Щось пішло не так :(');
        });
    }
}

function validateForm(formData) {
    if (validateField(formData.name, true, 3, 25) == false &&
formData.reqType === 'Загублено') {
        showValidationError('Прізвисько некоректне (від 3 до 25
символів)');
        return false;
    }
```

```
    if (validateField(formData.color, false, 3, 50) == false) {
        showValidationError('Колір некоректний (від 3 до 50 символів)');
        return false;
    }
    if (validateField(formData.age, false, 1, 25) == false) {
        showValidationError('Вік некоректний (від 1 до 25 символів)');
        return false;
    }
    if (validateField(formData.breed, false, 3, 50) == false) {
        showValidationError('Порода некоректна (від 3 до 50 символів)');
        return false;
    }
    if (validateField(formData.number, true, 9, 13) == false ||
validatePhone(formData.number) == false) {
        showValidationError('Номер некоректний (+380991231212)');
        return false;
    }
    if (validateField(formData.description, false, 3, 500) == false) {
        showValidationError('Опис некоректний (від 3 до 500 символів)');
        return false;
    }
    if (formData.photo == null) {
        showValidationError('Виберіть фото');
        return false;
    }
    if (formData.coordinates == null) {
        showValidationError('Виберіть точку на карті');
        return false;
    }
    return true;
}

function validatePhone(phone) {
```

```
    let phoneRegex = /^[\+]?[(]?[0-9]{3}[)]?[-\s\.]?[0-9]{3}[-\s\.]?[0-
9]{4,6}$/im;
    return phoneRegex.test(phone);
}

function showValidationError(message) {
    Toastify({
        text: '⚠ ' + message,
        duration: 2000,
        gravity: 'bottom',
        position: 'center',
        stopOnFocus: true,
        className: 'toast-error'
    }).showToast();
}

function validateField(value, required, minLen, maxLen) {
    let isValid = true;
    if (value.length === 0) {
        isValid = !required;
    } else {
        if (value.length < minLen || value.length > maxLen) {
            isValid = false;
        }
        if (value.includes('<') || value.includes('>')) {
            isValid = false;
        }
    }
    return isValid;
}
```

helper.js

```
//convert degrees to radians
function rad(x) {
    return x * Math.PI / 180;
```

```
}

// get distance between two points in meters
function getDistance(p1, p2) {
    let R = 6378137;
    let dLat = rad(p2.lat - p1.lat);
    let dLong = rad(p2.lng - p1.lng);
    let a = Math.sin(dLat / 2) * Math.sin(dLat / 2) +
        Math.cos(rad(p1.lat)) * Math.cos(rad(p2.lat)) *
        Math.sin(dLong / 2) * Math.sin(dLong / 2);
    let c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));
    let d = R * c;
    return d;
}
```

map.js

```
let centerLat = 50.907688;
let centerLong = 34.796716;
let selectedPoint = {};
let marker;
let map;
let animalMap;
let animalsMarkers = [];

window.onload = function() {
    map = L.map('map').setView([centerLat, centerLong], 13);
    L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png', {
    attribution:                        '&copy;                        <a
href="http://osm.org/copyright">OpenStreetMap</a> contributors'
    }).addTo(map);

    let circle = L.circle([50.907688, 34.796716], {
        color: 'yellow',
        fillColor: 'yellow',
        fillOpacity: 0.08,
```

```
        radius: 5000
    }).addTo(map);

    map.on('click', function(e){
        let center = {
            lat: centerLat,
            lng: centerLong,
        }
        let point = {
            lat: e.latlng.lat,
            lng: e.latlng.lng,
        }

        if(getDistance(center, point) <= 5000) {
            if (marker) {
                marker.remove()
            }
            marker = L.marker(e.latlng).addTo(map);
            selectedPoint = point;
        }
    });

    animalMap  =  L.map('animalMap').setView([centerLat,  centerLong],
13);
    L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png', {
        attribution:                    '&copy;                    <a
href="http://osm.org/copyright">OpenStreetMap</a> contributors'
    }).addTo(animalMap);
    document.getElementById('animalsTab').style.display = 'none';// fix
for map loading
}

function addMarker(request) {
    let icon = getIcon(request.reqType, request.kind);
```

```
    let marker = L.marker([request.coordinates.lat,
request.coordinates.lng], {icon: icon});
    let description = request.description;
    let popupText = '<div class="popup"><img class="popup-image" src="'+
request.photo +'"></br><p>'+ description +'</p></div>';
    marker.bindPopup(popupText);
    marker.on('click', function(e) {
        openAnimalForm(request);
    });
    animalsMarkers.push(marker);
    return marker;
}

function clearMap() {
    for (let i = 0; i < animalsMarkers.length; i++) {
        animalsMarkers[i].remove();
    }
    animalsMarkers = [];
}

function getIcon(reqType, kind) {
    let iconName = '';
    if (reqType === 'Загублено') {
        iconName = 'lost';
    }   else if (reqType === 'Знайдено') {
        iconName = 'found';
    }
    if (kind === 'Кішка') {
        iconName += 'Cat';
    } else if (kind === 'Собака') {
        iconName += 'Dog';
    }
    return L.icon({
        iconUrl: 'resources/' + iconName + '.png',
```

```
        iconSize: [40, 40],
        iconAnchor: [20, 20],
    });
}

function getSelectedCoordinates() {
    if (selectedPoint.lat && selectedPoint.lng) {
        return selectedPoint;
    } else {
        return null;
    }
}

function removeSelectedCoordinates() {
    if (marker) {
        marker.remove();
        selectedPoint = {}
    }
}
```

requests.js

```
let animals = [];
let defaultForm = document.createElement('p');
defaultForm.innerHTML = 'Оберіть маркер на карті'
defaultForm.id = 'defaultForm';
document.getElementById('animalForm').appendChild(defaultForm);

function updateRequests() {
    fetch('/api/getAllRequests')
        .then((response) => {
            return response.json();
        })
        .then((data) => {
            clearMap();
            animals = data.requests;
```

```
            for (let i = 0; i < data.requests.length; i++) {
                let marker = addMarker(data.requests[i]);
                marker.addTo(animalMap);
            }
        }).catch((err) => {
            Toastify({
                text: '⚠️ Щось пішло не так :(',
                duration: 2000,
                gravity: 'bottom',
                position: 'center',
                stopOnFocus: true,
                className: 'toast-error'
            }).showToast();
        });
}

function openAnimalForm(request) {
    document.getElementById('animalForm').innerHTML = '';
    let form = document.createElement('div');
    form.id = 'populatedForm';

    let name = document.createElement('div');
    name.innerHTML = request.name;
    name.classList.add('info-name', 'info-div');

    let type = document.createElement('div');
    type.innerHTML = request.reqType;
    type.classList.add('info-type', 'info-div');

    let age = document.createElement('div');
    if (request.age === '') {
        age.innerHTML = 'не вказано';
    } else {
        age.innerHTML = request.age;
```

```
    }
    age.classList.add('info-age', 'info-div');

    let breed = document.createElement('div');
    if (request.breed === '') {
        breed.innerHTML = 'не вказано';
    } else {
        breed.innerHTML = request.breed;
    }
    breed.classList.add('info-breed', 'info-div');

    let color = document.createElement('div');
    if (request.color === '') {
        color.innerHTML = 'не вказано';
    } else {
        color.innerHTML = request.color;
    }
    color.classList.add('info-color', 'info-div');

    let description = document.createElement('div');
    if (request.description === '') {
        description.innerHTML = 'немає';
    } else {
        description.innerHTML = request.description;
    }
    description.classList.add('info-description', 'info-div');

    let kind = document.createElement('div');
    kind.innerHTML = request.kind;
    kind.classList.add('info-kind', 'info-div');

    let number = document.createElement('div');
    number.innerHTML = request.number;
    number.classList.add('info-number', 'info-div');
```

```
let photo = document.createElement('img');
photo.setAttribute('src', request.photo);
photo.classList.add('info-photo', 'info-div');

let sex = document.createElement('div');
sex.innerHTML = request.sex;
sex.classList.add('info-sex', 'info-div');

let tagContainer = document.createElement('div');
tagContainer.classList.add('info-tags');
tagContainer.appendChild(type);
tagContainer.appendChild(kind);
tagContainer.appendChild(sex);

form.appendChild(tagContainer);
form.appendChild(photo);
if (request.reqType === 'Загублено') {
    form.appendChild(name);
    form.appendChild(age);
}
form.appendChild(breed);
form.appendChild(color);
form.appendChild(number);
form.appendChild(description);

if (request.shelter) {
    fetch('/api/getShelterById/?id=' + request.shelter, {
        method: 'GET',
        headers: {
            'Content-Type': 'application/json'
            }
        })
        .then((response) => {return response.json();})
```

```
            .then((data) => {
                let shelter = document.createElement('div');
                shelter.innerHTML = data.shelter.name + ' / ' +
data.shelter.address;
                shelter.classList.add('info-shelter', 'info-div');
                form.appendChild(shelter);
            }
            ).catch((err) => {
                Toastify({
                    text: '⚠ Щось пішло не так :(',
                    duration: 2000,
                    gravity: 'bottom',
                    position: 'center',
                    stopOnFocus: true,
                    className: 'toast-error'
                }).showToast();
            }
            );

    }

    document.getElementById('animalForm').appendChild(form);
}
```

tab.js

```
function tabHandler(event) {
    if (event.target.classList.contains('tab')) {
        let activeTab =
event.target.parentElement.getElementsByClassName('active')[0];
        activeTab.classList.remove('active');
        event.target.classList.add('active');
        if (event.target.id === 'animals') {
            document.getElementById('newRequestTab').style.display =
'none';
```

```
            document.getElementById('animalsTab').style.display       =
'flex';
            updateRequests();
        }
        if (event.target.id === 'newRequest') {
            document.getElementById('animalsTab').style.display       =
'none';
            document.getElementById('newRequestTab').style.display    =
'flex';
            if (document.getElementById('populatedForm')) {
                document.getElementById('populatedForm').remove();
            }

document.getElementById('animalForm').appendChild(defaultForm);
        }
    }
}
```

/styles

admin.css

```
@import
url('https://fonts.googleapis.com/css2?family=Montserrat:wght@400;700;
900&display=swap');

select,
input,
textarea {
    margin-bottom: 10px;
    min-width: 350px;
    min-height: 18px;
}

#loginForm {
```

```
    justify-content: center;
    font-family: 'Montserrat', sans-serif;
}

#content {
    font-family: 'Montserrat', sans-serif;
    flex-direction: column;
}

#infoForm {
    display: flex;
    flex-direction: column;
    margin-left: 20px;
}

.shelters-form {
    display: flex;
    flex-direction: column;
    margin-left: 10px;
    margin-right: 10px;
    margin-top: 10px;
}

#sheltersList {
    border: 1px solid black;
    height: 300px;
    margin-bottom: 10px;
}

.shelter-item {
    display: flex;
    justify-content: space-between;
    font-family: monospace;
    border: 1px solid black;
```

```
    max-width: 250px;
}

.shelters-list {
    height: 300px;
    width: 250px;
    overflow-y: scroll;
    overflow-x: hidden;
}

.delete-button {
    border: none;
    background: none;
    cursor: pointer;
}

.form-shelter-button {
    margin-left: 10px;
}

.form-photo {
    width: 250px;
    margin-bottom: 10px;
}

.login-form {
    display: flex;
    flex-direction: column;
    align-items: center;
}

.login-form input {
    margin-bottom: 10px;
}
```

```
.head {
    width: 100%;
    display: flex;
    justify-content: space-between;
    align-items: center;
}

.body {
    margin-top: 20px;
    height: 93vh;
    display: flex;
}

.list-item {
    display: flex;
    justify-content: space-between;
    align-items: center;
    border: 1px solid black;
    border-radius: 10px;
    width: 450px;
    margin-bottom: 10px;
}

#requestsList {
    height: 94%;
    overflow-y: auto;
    width: fit-content;
}

.list-item-image {
    height: 60px;
    max-width: 60px;
    border-radius: 10px;
```

```
}

.list-item-phone {
    padding-left: 15px;
}

.list-item-delete-btn {
    padding-right: 15px;
    cursor: pointer;
}

.list-item-sex,
.list-item-place,
.list-item-kind {
    padding-left: 15px;
}

.list-item-info-container {
    display: flex;
    align-items: center;
    cursor: pointer;
}
```

cards.css

```
#about {
    display: flex;
    justify-content: flex-start;
    height: 90vh;
    flex-direction: column;
    padding-top: 12%;
    margin-top: -12%;
}

.card-container {
    display: flex;
```

```
    justify-content: center;
    align-content: space-around;
    align-items: center;
}

.card {
    padding: 5px 20px 32px 20px;
    text-align: left;
    border: 3px solid #231E54;
    border-radius: 25px;
    width: 25%;
    height: 80%;
    margin-left: 25px;
}

p.cardtext {
    margin-left: auto;
    margin-right: auto;
    width: 350px;
}

h2.cardheader {
    text-align: center;
}

img.cardimage {
    display: block;
    margin-left: auto;
    margin-right: auto;
    width: 48%;
}
```

footer.css

```
#contact {
    display: flex;
```

```
    justify-content:space-between;
}

#left-container {
    text-align: center;
}

#right-container {
    text-align: right;
    margin-top: 1.8%;
    margin-right: 2%;
}

#footerImage {
    margin-left: -5%;
    margin-bottom: -52% !important;
}

#phone {
    margin-left: 18%;
    margin-bottom: -32%;
}

#socialmedia {
    display: flex;
}

#copyright {
    font-size: 14px;
}

footer p {
    font-size: 16px;
}
```

```
footer li {
    list-style-type: none;
    margin-left: 6px;
}
```

header.css

```
header {
    display: flex;
    align-items: center;
    justify-content: space-between;
    background: #FFFFFF;
    width: 100%;
    padding-top: 12px;
    position: fixed;
    z-index: 100000;
}

#menu,
#menu li {
    margin: 0;
    padding: 0;
}

#menu {
    text-align: center;
}

#menu li {
    display: inline-block;
    text-align: center;
    height: 24px;
}

#menu a {
```

```
    color: #231E54;
    font-family: 'Montserrat', sans-serif;
    font-size: 20px;
    text-decoration: none;
    display: block;
    padding: 5px 15px;
    margin-right: 5px;
}

#menu a:hover {
    color: #FBAD36;
}

.logo-container,
.button-container {
    width: 300px;
}

.logo-container {
    padding-left: 5%;
}
```

index.css

```
@import
url('https://fonts.googleapis.com/css2?family=Montserrat:wght@400;700;
900&display=swap');

h2 {
    color: #231E54;
    font-family: 'Montserrat', sans-serif;
    font-weight: 900;
}

h3 {
    color: #231E54;
```

```
    font-family: 'Montserrat', sans-serif;
    font-weight: 900;
}

p {
    color: #231E54;
    font-family: 'Montserrat', sans-serif;
}

.colortext {
    color: #FBAD36;
}

#contact {
    padding-top: 17%;
    margin-top: -12%;
}
```

infoForm.css

```
.info-div {
    font-family: 'Montserrat', sans-serif;
    font-size: 16px;
    color: #231E54;
    margin-left: 1%;
    margin-top: 2px;
}

.info-name::before {
    content: 'Прізвисько: ';
    font-weight: 600;
}

.info-age::before {
    content: 'Вік: ';
    font-weight: 600;
```

```
}

.info-breed::before {
    content: 'Порода: ';
    font-weight: 600;
}

.info-color::before {
    content: 'Колір: ';
    font-weight: 600;
}

.info-number::before {
    content: 'Телефон: ';
    font-weight: 600;
}

.info-shelter::before {
    content: 'Перетримка: ';
    font-weight: 600;
}

.info-description::before {
    content: 'Коментар: ';
    font-weight: 600;
}

.info-photo {
    height: 300px;
    width: auto;
    max-width: 400px;
    margin-bottom: 7px;
}
```

```
.info-description {
    max-width: 300px;
}

.info-tags {
    display: flex;
    flex-direction: row;
    margin-bottom: 10px;
    margin-left: -10px;
}

.info-type,
.info-kind,
.info-sex {
    border: 2px solid #FBAD36;
    border-radius: 25px;
    padding: 10px 15px 10px 15px;
    color: #FBAD36 !important;
}

#populatedForm {
    display: flex;
    flex-direction: column;
    align-items: stretch;
}
```

main.css

```
body {
    margin: 0px !important;
}

#main {
    height: 89vh;
    display: flex;
    padding-top: 12%;
```

```
}

#main-text {
    margin-left: 120px;
}

#illustration {
    -webkit-transform: scaleX(-1);
    transform: scaleX(-1);
}

.main-illustration {
    height: 80%;
}


.button-container {
    display: flex;
    justify-content: flex-end;
}

.button {
    color: #FBAD36;
    background-color: #FFFFFF;
    font-family: 'Montserrat', sans-serif;
    font-size: 20px;
    font-weight: 600;
    text-decoration: none;
    border: 2px solid #FBAD36;
    height: 57px;
    width: 188px;
    transition: all 0.4s ease 0s;
    border-radius: 25px;
    display: flex;
    align-items: center;
```

```
    justify-content: center;
    margin-right: 32px;
}

.button:hover {
    color: #FFFFFF;
    background: #FBAD36;
    border-color: #FBAD36;
    transition: all 0.4s ease 0s;
}

.selected {
    color: #FFFFFF;
    background: #FBAD36;
    border-color: #FBAD36;
}

.toast-error {
    background: #ff5959;
    color: #FFFFFF;
    border-radius: 25px;
    font-family: 'Montserrat', sans-serif;
}

.toast-success {
    background: #8aff90;
    color: #FFFFFF;
    border-radius: 25px;
    font-family: 'Montserrat', sans-serif;
}

@media only screen
and (min-width: 1224px) {
    #main-text {
```

```
        font-size: 2.6ch;
    }
}

@media only screen
and (min-width: 1824px) {
    #main-text {
        margin-left: 10%;
        margin-top: 5%;
        font-size: 3ch;
    }
    p.cardtext {
        margin-bottom: 40px;
    }
    #map {
        margin-left: 10% !important;
    }
}
```

searchPet.css

```
#search {
    padding-top: 12%;
    margin-top: -5%;
}

.search {
    margin-top: 2%;
}

.searchDescription {
    margin-left: 10%;
    margin-top: 10%;
    display: flex;
    align-items: center;
}
```

```
#searchText {
    margin-top: 35px;
    margin-left: 35px;
    width: 725px;
}

#gifCat {
    width: 275px;
    height: 275px;
}

.form {
    margin-left: 3%;
    margin-right: 7%;
    display: flex;
    flex-direction: column;
}

.selector {
    display: flex;
    margin-bottom: 22px;
}

.field {
    color: #231E54;
    font-family: 'Montserrat', sans-serif;
    font-size: 16px;
    text-decoration: none;
    border: 1px solid #231E54;
    height: 40px;
    width: 400px;
    transition: all 0.4s ease 0s;
    border-radius: 15px;
```

```
    align-items: center;
    justify-content: center;
    margin-right: 32px;
    margin-bottom: 8px;
    text-indent: 18px;
}

#map, #animalMap {
    width: 625px;
    height: 625px;
    border: 2px solid #FBAD36;
    border-radius: 25px;
    margin-left: 5%;
}

#animalMap {
    margin-right: -100px;
    margin-left: -30px;
}

.popup {
    max-width: 150px;
}

.popup-image {
    max-width: 100px;
}

#defaultForm {
    min-width: 350px;
    margin-top: 40%;
}

#submit {
```

```
    height: 57px;
    width: 400px;
    cursor: pointer;
}

#newRequestTab {
    margin-top: 2%;
    display: flex;
    justify-content: space-evenly;
    align-items: flex-start;
}

#animalsTab {
    margin-top: 2%;
    display: flex;
    justify-content: space-evenly;
    align-items: flex-start;
    height: 625px;
}

#tabBar {
    display: flex;
    justify-content: center;
}

.tab {
    margin: 0px 50px 0px 0px;
    font-family: 'Montserrat', sans-serif;
    font-size: 20px;
    cursor: pointer;
    color: #a4a4a4;
}

.tab:hover {
```

```
    color: #231e54;
}

.active {
    color: #231e54;
    border-bottom: 2px solid #fbad36;
}

.custom-file-input::before {
    content: 'Фото';
    font-weight: 600;
    display: flex;
    outline: none;
    white-space: nowrap;
    cursor: pointer;
    color: #231E54;
    background-color: #FFFFFF;
    font-family: 'Montserrat', sans-serif;
    font-size: 16px;
    text-decoration: none;
    border: 2px solid #231E54;
    height: 40px;
    width: 100px;
    border-radius: 15px;
    align-items: center;
    justify-content: center;
    margin-right: 32px;
}

.custom-file-input:hover::before {
    border-color: #231E54;
    background-color: #231E54;
    color: #FFFFFF;
    transition: all 0.4s ease 0s;
```

```
}

.formButton {
    color: #FBAD36;
    background-color: #FFFFFF;
    cursor: pointer;
    font-family: 'Montserrat', sans-serif;
    font-size: 16px;
    font-weight: 600;
    text-decoration: none;
    border: 2px solid #FBAD36;
    height: 50px;
    width: 188px;
    transition: all 0.4s ease 0s;
    border-radius: 15px;
    display: flex;
    align-items: center;
    justify-content: center;
    margin-right: 32px;
    margin-top: -8px;
    margin-bottom: -8px;
}

.selected {
    color: #FFFFFF;
    background: #FBAD36;
    border-color: #FBAD36;
}

#form-comment {
    height: 80px;
    padding-top: 12px;
}
```

```
.file-selector {
    display: flex;
    justify-content: flex-start;
    align-items: flex-start;
}
```