ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка Web-застосунку для адопції тварин з притулку з використанням технологій Node Js, React»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Ігор ПИЛИПЧУК
(підпис)

Виконав: здобувач вищої освіти групи ПД-41

Ігор ПИЛИПЧУК

Керівник: Тимур ДОВЖЕНКО
к.т.н.

Рецензент: _____

Київ 2024

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
« ____ » _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Пилипчуку Ігорю Руслановичу

1. Тема кваліфікаційної роботи: «Розробка Web-застосунку для адопції тварин з притулку з використанням технологій Node js, React»
керівник кваліфікаційної роботи к.т.н., доцент кафедри ІПЗ Тимур ДОВЖЕНКО,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «27» лютого 2024 р. № 36.

2. Строк подання кваліфікаційної роботи «28» травня 2024 р.

3. Вихідні дані до кваліфікаційної роботи: науково-технічна література, пов'язана з розробкою веб-застосунків;мова програмування JavaScript;фреймворк для розробки серверної частини Express;бібліотека компонентів React; нереляційна база даних Mongo DB.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Здійснити аналіз предметної області у сфері веб-застосунків для адопції тварин.
2. Дослідити існуючі системи з адопції тварин, виявити їхні переваги та недоліки.
3. Визначити актуальні бібліотеки та фреймворки для створення веб-застосунку.
4. Створити детальний проект системи, включаючи архітектуру, дизайн інтерфейсу та моделі даних і розробити відповідне програмне забезпечення.
5. Провести тестування: серверної частини додатку end-to-end методом, клієнтської частини – мануальним способом.
6. Написати детальну документацію для back-end частини додатку.

5. Перелік графічного матеріалу: *презентація*

1. Аналіз аналогів.
2. Вимоги до додатку.
3. Програмні засоби реалізації.
4. Діаграма варіантів використання
5. Діаграма діяльності
6. Схема бази даних
7. Архітектуру застосунку
8. Мапа сайту
9. Екранні форми.
10. Апробація результатів дослідження

6. Дата видачі завдання «28» лютого 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	28.02.-06.03.2024	
2	Аналіз та дослідження існуючих аналогів веб-застосунків для адопції тварин	07.03-13.03.2024	
3	Аналіз та вибір інструментів для розробки веб-застосунку	14.03-17.03.2024	
4	Проектування та моделювання системи	18.03-19.03.2024	
5	Створення програмного рішення	19.03-27.04.2024	
6	Тестування застосунку	27.04-28.04.2024	
7	Оформлення роботи: вступ, висновки, реферат	29.04-05.05.2024	
8	Розробка демонстраційних матеріалів	06.05-12.05.2024	
9	Попередній захист роботи	13.05-31.05.2024	

Здобувач вищої освіти _____ Ігор ПИЛИПЧУК
(підпис)

Керівник
кваліфікаційної роботи _____ Тимур ДОВЖЕНКО
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра (магістра): 61стор., 44 рис., 1 табл., 13 джерел.

Мета роботи – автоматизація процесу адопції тварин з притулків шляхом створення веб-застосунку.

Об'єкт дослідження – процес адопції тварин з притулку

Предмет дослідження – програмне забезпечення для адопції тварин.

Короткий зміст роботи: в роботі було реалізований веб-застосунок, який складається з серверної та клієнтської частини. Для створення back-end частини використовувались: фреймворк Express js, нереляційна база даних Mongo DB, бібліотека для надсилання електронних листів Nodemailer, пакет jsonwebtoken для створення авторизаційних токенів для користувачів, інструмент для створення інтерактивної документації Swagger, інструменти для тестування: Jest, faker,supertest. Back-end частина має систему прав, яка розділяє користувачів і адміністраторів, що надає додатковий шар безпеки застосунку. Також існує система підтвердження акаунту по електронній пошті: після того, як користувач зареєструвався, йому на email надсилається унікальне посилання, за яким потрібно перейти, щоб активувати акаунт. Серверна частина протестувалась end-to-end тестами, клієнтська частина тестувалась мануальним способом. Для розробки front-end частини використовувались : бібліотека компонентів React, бібліотека UI елементів, у стилі Material Design, MUI, пакет для надсилання HTTP запитів axios, пакет react-router для створення маршрутизації між сторінками застосунку. У клієнтській частині усі сторінки складаються з компонентів, які можуть в свою чергу, бути динамічними. Токен

авторизації користувача зберігається в localstorage, на стороні браузера. Токен авторизації валідується, у випадку, якщо користувач намагається відвідати приватні сторінки, наприклад, такі як : особистий кабінет, розділ “favorites”. Авторизовані користувачі можуть надсилати запити на адопцію до тварин, редагувати інформацію про себе в особистому кабінеті і додавати тварин у розділ “favorites”.

КЛЮЧОВІ СЛОВА: ПРИТУЛОК, JAVASCRIPT, REACT, MONGO DB,АДОПЦІЯ, NODE JS

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.……11

ВСТУП…….....12

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ……...……...14

1.1 Актуальність проблеми безпритульних тварин…….……...14

1.2 Happy paw…….16

1.3 ЛКП «Лев……..18

1.4 Animal - ID……21

1.5 Таблиця порівнянь аналогів……..……23

2 ВИБІР ЗАСОБІВ ПРОГРАМНОЇ РЕАЛІЗАЦІЇ…….....25

2.1 Мова програмування Javascript……..25

2.2 Node js ……..27

2.3 Express js ……..32

2.4 Mongo DB ……33

2.5 React …….35

2.6 Додаткові інструменти та засоби розробки …….……36

2.6.1 MUI ……36

2.6.2 Nodemailer …….. ..37

2.6.3 Visual Studio Code ……...…….38

2.6.4 Postman ……...…...38

2.6.5 Swagger …….....40

2.6.6 Jest …….....41

3 ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ……42

3.1 Функціональні вимоги…….……...42

3.2 Нефункціональні вимоги……..……...42

3.3 Моделювання об’єкту проектування…….....43

4 РОЗРОБКА ТА ТЕСТУВАННЯ ЗАСТОСУНКУ…….....50
4.1 Реалізація API роутів …….…….50
4.2 Створення документації…….……..55
4.3 Реалізація інтерфейсу клієнтської частини…….....……...58
4.4 Тестування застосунку…….61
ВИСНОВКИ……64
ПЕРЕЛІК ПОСИЛАНЬ……..……65
ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛ (Презентація)…….…..66
ДОДАТОК Б. ОСНОВНІ БЛОКИ КОДУ……74

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API - Application Programming Interface

JSON – JavaScript Object Notation

BSON - Binary JavaScript Object Notation

UML - Unified Modeling Language

JSX - JavaScript Syntax eXtension

XML - EXtensible Markup Language

HTML - HyperText Markup Language

JS – JavaScript

ВСТУП

У сучасному світі, де галузь інформаційних технологій стрімко розвивається, все більше аспектів нашого життя переходять в онлайн середовище. З початком повномасштабного вторгнення Росії на територію України проблема безпритульних тварин стала особливо актуальною. Військовий конфлікт призвів до збільшення кількості тварин, які втратили дім. Евакуюючись від війни, багато громадян залишили домашніх улюбленців напризволяще. Звичні способи пошуку та адопції домашніх улюбленців часто вимагають значних часових і фізичних витрат. На цьому тлі виникає потреба у зручних та ефективних рішеннях, що могли б спростити цей процес і зробити його більш доступним для широкої аудиторії.

Адопція — це процес усиновлення тварини, яку попередній власник залишив у притулку.

Об'єкт дослідження – процес адопції тварин з притулків.

Предмет дослідження – програмне забезпечення для адопції тварин з притулків.

Мета роботи – автоматизація процесу адопції тварин з притулків, шляхом створення веб-застосунку.

Задачі дипломної роботи було сформульовано наступним чином:

1. Здійснити аналіз предметної області у сфері веб-застосунків для адопції тварин.
2. Дослідити існуючі системи з адопції тварин, виявити їхні переваги та недоліки.
3. Визначити актуальні бібліотеки та фреймворки для створення веб-застосунку.
4. Створити детальний проект системи, включаючи архітектуру, дизайн інтерфейсу та моделі даних і розробити відповідне програмне забезпечення.
5. Провести тестування: серверної частини додатку end-to-end методом, клієнтської частини – мануальним способом.
6. Написати детальну документацію для back-end частини додатку.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Актуальність проблеми безпритульних тварин

До повномасштабного вторгнення стан безпритульних тварин в Україні був складним, але поступово покращувався завдяки зусиллям волонтерів, громадських організацій та окремих активістів. Проблема безпритульних тварин існувала протягом багатьох років і охоплювала як великі міста, так і сільські місцевості. У містах, таких як Київ, Харків, Одеса та Львів, численні зоозахисні організації активно працювали над зменшенням кількості безпритульних тварин. Основними проблемами були неконтрольоване розмноження і недостатність ресурсів для стерилізації, вакцинації та догляду за тваринами. Програми стерилізації і вакцинації проводилися, але їхнє охоплення було недостатнім для вирішення проблеми на загальнодержавному рівні. Волонтери часто працювали на власному ентузіазмі та за власний рахунок, збираючи кошти через благодійні акції та онлайн-кампанії.

У великих містах було створено кілька муніципальних притулків, проте їхніх можливостей не вистачало для всіх потребуючих тварин. Більшість притулків фінансували за рахунок приватних пожертвувань, вони рідко отримували державну підтримку. Притулки надавали тимчасовий прихисток тваринам, лікування та допомогу в пошуку нових власників. Однією з основних проблем було ставлення до безпритульних тварин у суспільстві. Незважаючи на зростаючу свідомість щодо прав тварин, багато людей вважали їх обтяжливими та не брали на себе відповідальність за їхній догляд. На вулицях часто можна було побачити безпритульних собак і котів, які шукали їжу та прихисток.

У сільській місцевості ситуація була ще складнішою через відсутність організованих зоозахисних структур та обмежений доступ до ветеринарних послуг. Тварини часто залишалися без догляду, харчування і лікування, гостро стояла

проблема неконтрольованого розмноження. Попри ці труднощі, були й позитивні зрушення. Завдяки активній роботі зоозахисників і волонтерів, все більше людей починали усвідомлювати важливість стерилізації та відповідального ставлення до домашніх тварин. Проводилися освітні кампанії, спрямовані на підвищення обізнаності щодо прав тварин і необхідності їхнього захисту. Завдяки спільним зусиллям, повільно, але впевнено, ситуація з безпритульними тваринами в Україні до повномасштабного вторгнення покращувалася. Залучення міжнародних організацій та грантової підтримки також сприяло впровадженню більш ефективних методів боротьби з проблемою, але все ще залишалося багато роботи для досягнення стійких результатів.

Актуальність проблеми безпритульних тварин значно зросла під час війни. Конфлікт змушує людей покидати свої домівки, багато хто не має можливості взяти своїх улюбленців з собою. У результаті, кількість безпритульних тварин на вулицях стрімко зростає. Багато з цих тварин залишаються без їжі, води та медичної допомоги, що створює додаткові проблеми для гуманітарних організацій і місцевих волонтерів. Окрім того, безпритульні тварини можуть стати джерелом поширення хвороб, що особливо небезпечно в умовах війни. Організації з захисту тварин стикаються з великими труднощами, намагаючись забезпечити хоча б мінімальний рівень догляду за безпритульними тваринами. Вони організовують притулки, роздають їжу та надають медичну допомогу, але ресурсів часто не вистачає. Волонтери, які працюють у цій сфері, ризикують власним життям, допомагаючи тваринам у зонах бойових дій.

Проблема безпритульних тварин під час війни вимагає не лише місцевих, але й міжнародних зусиль. Крім того, важливим аспектом є інформування громадськості про ситуацію з безпритульними тваринами під час війни. Медійна підтримка може привернути увагу до проблеми та стимулювати збір коштів і ресурсів для допомоги. Важливо також проводити освітні кампанії, щоб люди знали, як вони можуть допомогти навіть на відстані. Психологічний аспект також не слід недооцінювати. Для багатьох людей домашні тварини є джерелом емоційної підтримки, особливо в такі

складні часи. Допомога безпритульним тваринам може мати позитивний вплив і на людей, які постраждали від війни, допомагаючи їм відчувати себе потрібними і корисними.

1.2 Happy Paw

Happy Paw – благодійний фонд, що активно займається проблемами безпритульних тварин в Україні. Місія фонду полягає у зменшенні кількості безпритульних тварин та поліпшенні їхнього становища. Одним із головних напрямків діяльності Happy Paw є підтримка притулків для тварин. Фонд забезпечує їх кормом, медичними препаратами та необхідним обладнанням. Він також організовує ветеринарні огляди, вакцинацію та стерилізацію тварин, щоб запобігти неконтрольованому розмноженню та поширенню хвороб. Фонд активно працює над пошуком нових домівок для безпритульних тварин. На своєму веб-сайті та у соціальних мережах він публікує інформацію про тварин, які потребують родини, а також проводить акції та заходи, спрямовані на підвищення рівня усвідомлення громадськості щодо проблем безпритульних тварин. Happy Paw також реалізує освітні програми для дітей та молоді, навчаючи їх відповідальному ставленню до тварин: проводить уроки доброти у школах, організовує конкурси та фестивалі, щоб виховувати нове покоління, яке буде дбайливо ставитися до тварин.

Фонд співпрацює з міжнародними організаціями та отримує підтримку від іноземних благодійників. Це дозволяє розширювати діяльність та впроваджувати нові проекти. Одним із таких проектів є програма «Вірний друг», яка допомагає людям похилого віку знайти собі домашнього улюбленця, що сприяє покращенню їхнього психологічного стану та зменшує кількість безпритульних тварин. Happy Paw також займається юридичною підтримкою та адвокацією прав тварин, зокрема працює над вдосконаленням законодавства у сфері захисту тварин та забезпечення його дотримання. Організація проводить кампанії проти жорстокого поводження з

тваринами та підтримує ініціативи, спрямовані на їхній захист. Фонд активно залучає волонтерів, які допомагають у догляді за тваринами, організації заходів та проведенні інформаційних кампаній. Волонтери є важливою складовою діяльності фонду, оскільки їхня допомога дозволяє досягати кращих результатів у боротьбі з проблемою безпритульних тварин.

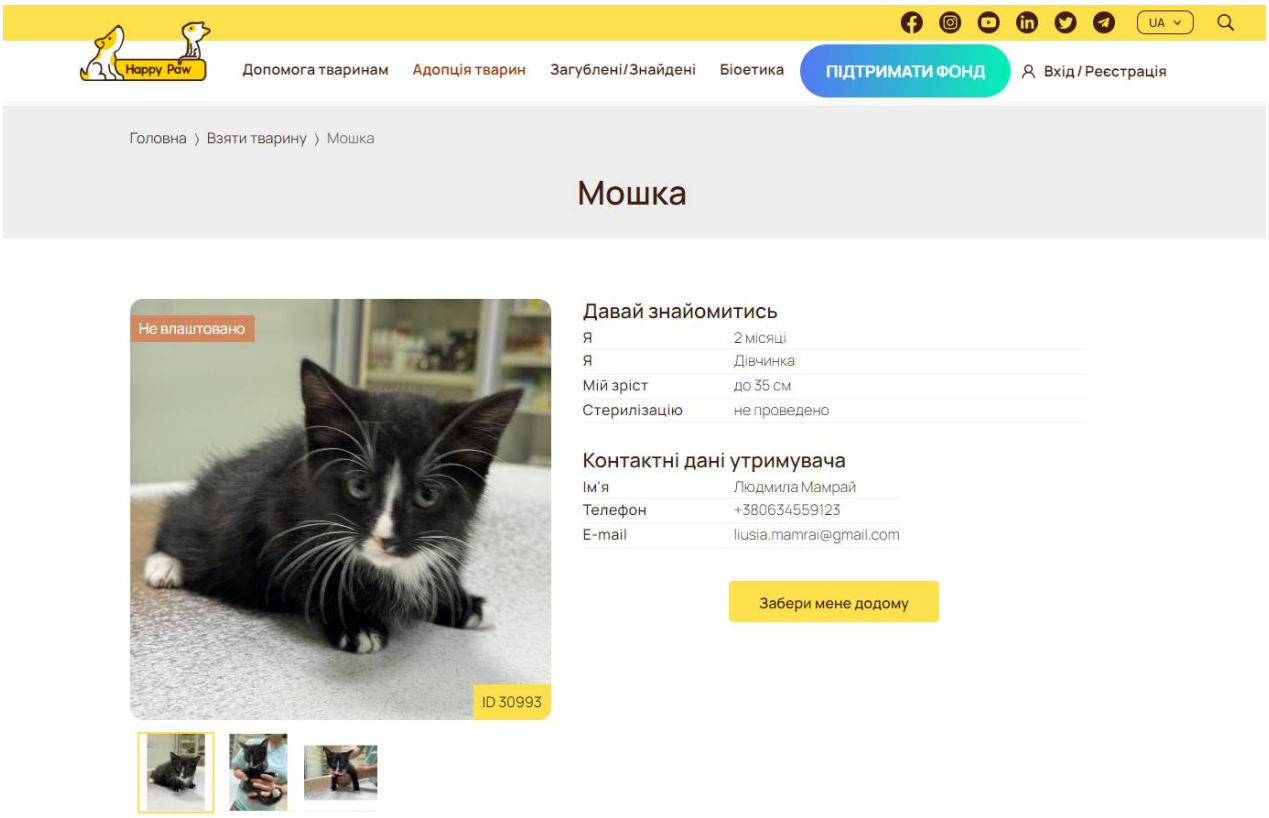


Рис. 1.1 Приклад перегляду картки тварини

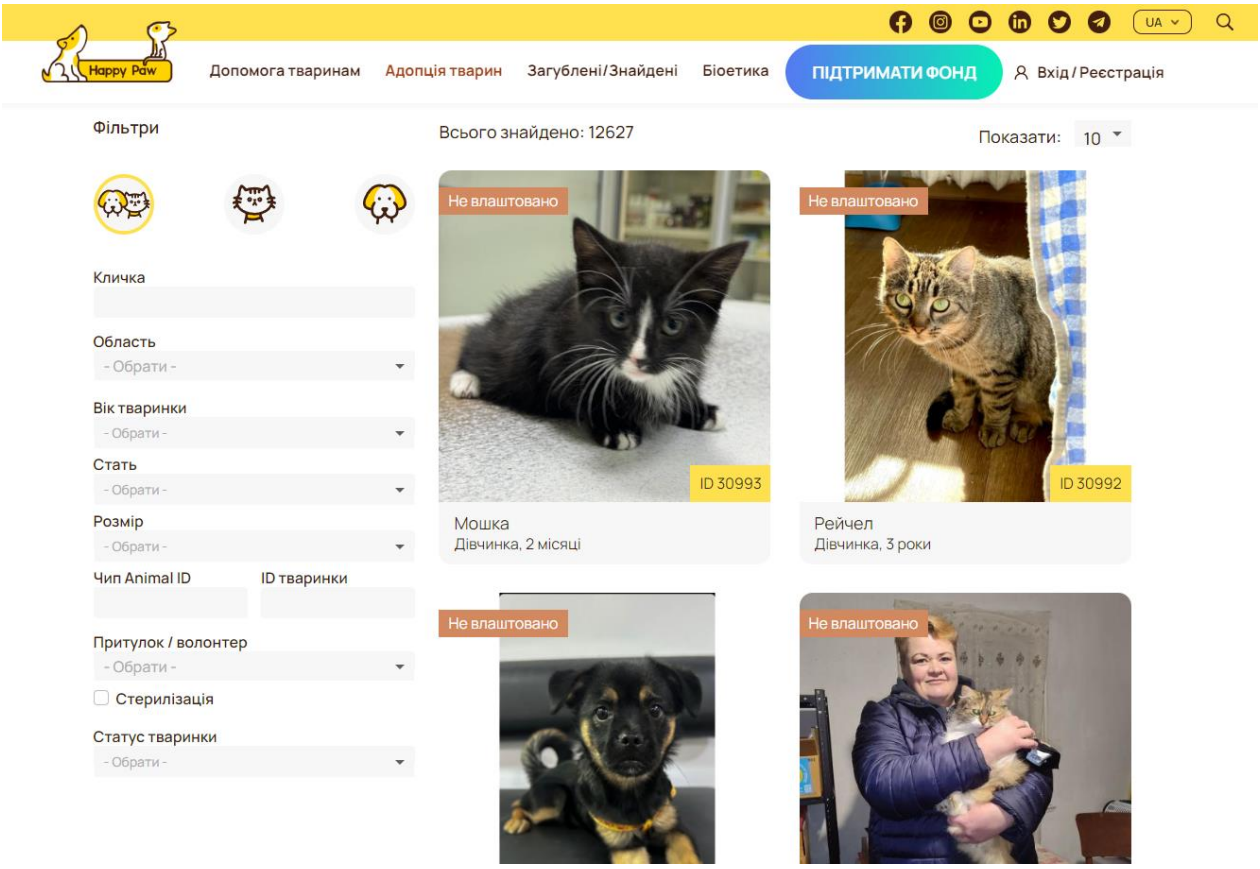


Рис. 1.2 Приклад перегляду сторінки з усіма тваринами

1.3 ЛКП «Лев»

ЛКП «Лев» – це благодійна організація, розташована у Львові, що займається проблемами безпритульних тварин. Заснована у 2003 році, ЛКП «Лев» є одним із найстаріших і найактивніших центрів допомоги тваринам в Україні. Місія організації полягає в забезпеченні гуманного ставлення до тварин, зменшенні кількості безпритульних тварин та пропаганді відповідального поводження з домашніми улюбленцями.

Основна діяльність ЛКП «Лев» включає роботу з безпритульними тваринами, зокрема їхнє відловлювання, лікування, стерилізацію та вакцинацію. Організація має власну ветеринарну клініку, де тварини отримують необхідну медичну допомогу.

Після реабілітації тварини передаються у нові родини або повертаються до притулку для подальшого утримання.

ЛКП «Лев» активно займається пошуком нових домівок для тварин. Вони проводять численні акції та кампанії, спрямовані на популяризацію адопції тварин. Організація співпрацює з місцевими та національними ЗМІ для поширення інформації про тварин, які шукають новий дім. Однією з важливих ініціатив ЛКП «Лев» є програма «Відповідальний власник», спрямована на виховання культури відповідального ставлення до домашніх улюбленців серед мешканців міста. Співробітники проводять освітні лекції, семінари та зустрічі для власників тварин, де розповідають про належний догляд, необхідність вакцинації та стерилізації. Організація також здійснює активну адвокаційну діяльність, спрямовану на вдосконалення законодавства у сфері захисту тварин. ЛКП «Лев» співпрацює з місцевою владою, поліцією та іншими організаціями для забезпечення дотримання законів щодо захисту тварин та запобігання жорстокому поводженню з ними. ЛКП «Лев» залучає волонтерів, які допомагають у догляді за тваринами, організації заходів та проведенні інформаційних кампаній. Волонтери є важливою частиною діяльності, оскільки їхня допомога дозволяє досягати кращих результатів у боротьбі з проблемою безпритульних тварин.

Організація також займається просвітницькою діяльністю серед молоді, проводячи уроки доброти у школах та реалізуючи освітні програми, спрямовані на виховання гуманного ставлення до тварин з раннього віку. ЛКП «Лев» співпрацює з міжнародними благодійними фондами та організаціями, отримуючи гранти та підтримку для реалізації своїх проектів. Це дозволяє їм розширювати свою діяльність та впроваджувати нові ініціативи.

Завдяки своїй діяльності ЛКП «Лев» здобув визнання як одна з провідних організацій у сфері захисту тварин в Україні. Вони продовжують працювати над тим, щоб забезпечити краще майбутнє для безпритульних тварин та виховувати у суспільстві культуру гуманного ставлення до всіх живих істот.

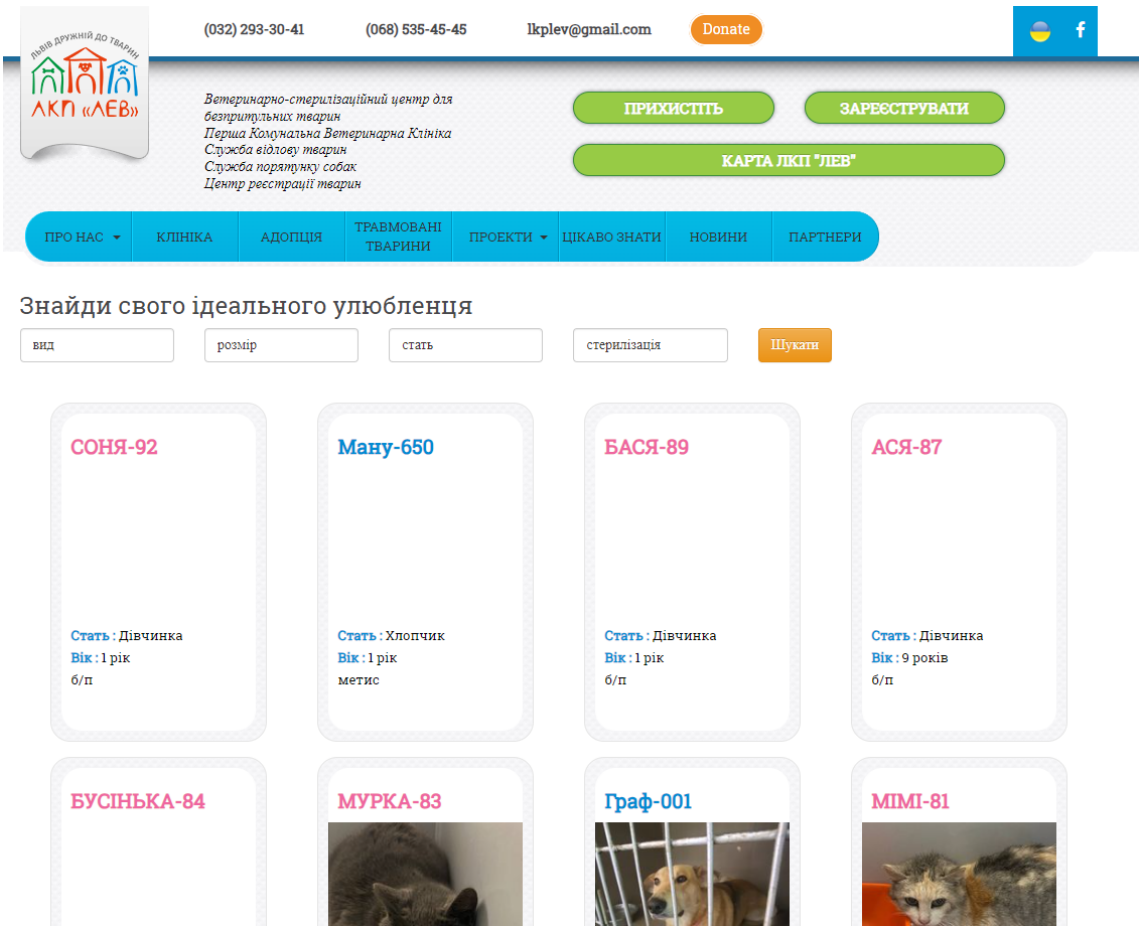


Рис. 1.3 Приклад перегляду сторінки з усіма тваринами

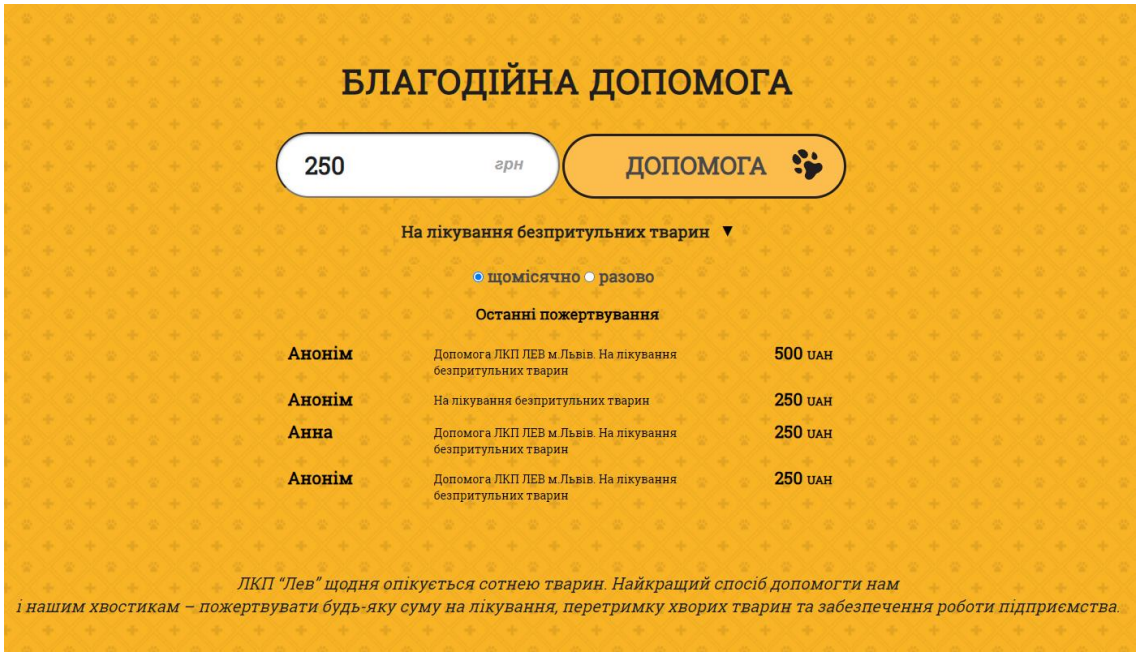


Рис. 1.4 Приклад форми з грошовою допомогою

1.4 Animal – ID

Animal ID – це міжнародна онлайн-платформа, що надає можливість знайти та усиновити домашнього улюбленця. Вона об’єднує різні притулки та волонтерські організації, роблячи процес пошуку тварини максимально зручним. Користувачі можуть переглядати профілі тварин, які потребують нового дому, з детальним описом їхнього характеру та особливостей догляду. Кожна тварина має свій унікальний ідентифікаційний номер, що полегшує процес комунікації з притулком. На платформі Animal ID можна знайти собак, котів і дрібних домашніх тварин, таких як кролики або морські свинки. Платформа також надає можливість залишити заявку на усиновлення або взяти тварину під тимчасову опіку. Це особливо корисно для людей, які хочуть допомогти, але не мають можливості взяти тварину назавжди.

Animal ID активно сприяє розвитку культури відповідального ставлення до домашніх тварин. Вона пропонує корисні поради щодо догляду та виховання тварин, а також інформацію про вакцинацію та стерилізацію. Платформа допомагає знаходити зниклих тварин за допомогою розділу з оголошеннями про зниклих та знайдених тварин.

Animal ID також співпрацює з ветеринарними клініками, пропонуючи спеціальні пропозиції та знижки для нових власників тварин. На сайті регулярно публікують блог успішних усиновлень, що позитивно впливає на мотивацію людей до активної участі у допомозі тваринам. Animal ID також організовує різні благодійні заходи та кампанії, метою яких є збір коштів для потреб тварин у притулках. Кожен

може долучитися до цих заходів, зробивши свій внесок у покращення життя бездомних тварин.

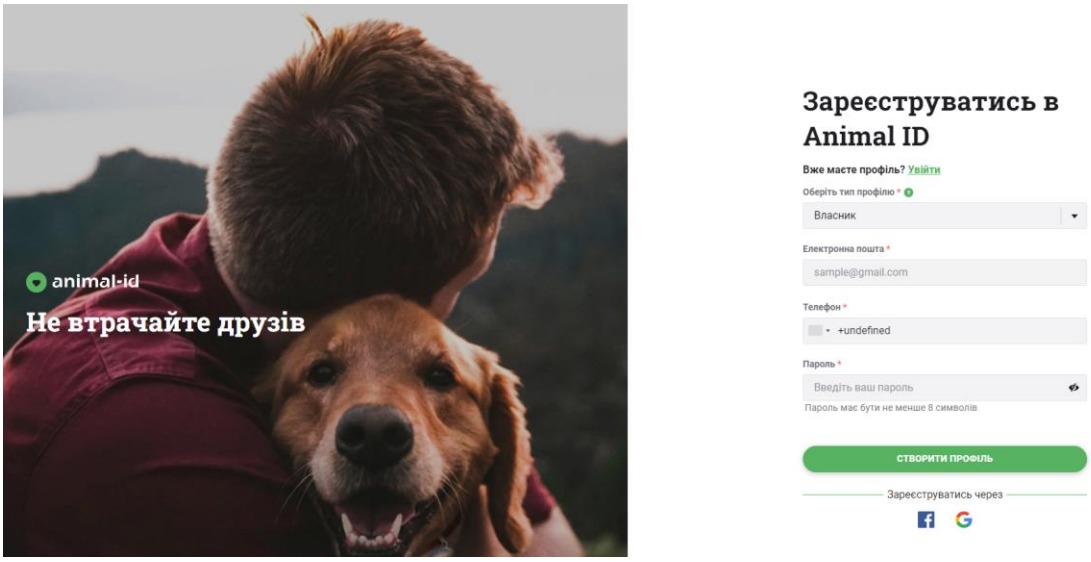


Рис. 1.5 Приклад форми реєстрації

Платформа має зручний інтерфейс та адаптована для використання на різних пристроях, включаючи смартфони та планшети. Це дозволяє швидко і зручно переглядати профілі тварин та подавати заявки на їх адопцію. Завдяки інтеграції з соціальними мережами, користувачі можуть ділитися інформацією про тварин, що потребують допомоги, зі своїми друзями та знайомими. Animal ID сприяє прозорості процесу усиновлення, надаючи всю необхідну інформацію про стан здоров’я та поведінку тварини. Це допомагає потенційним власникам зробити обґрунтований вибір і знайти саме ту тварину, яка стане їхнім справжнім другом. Платформа також підтримує зворотний зв’язок з користувачами, що дозволяє постійно покращувати її

роботу та додавати нові функції. Animal ID активно розвиває спільноту людей, які небайдужі до проблем бездомних тварин і готові допомагати їм знайти новий дім.

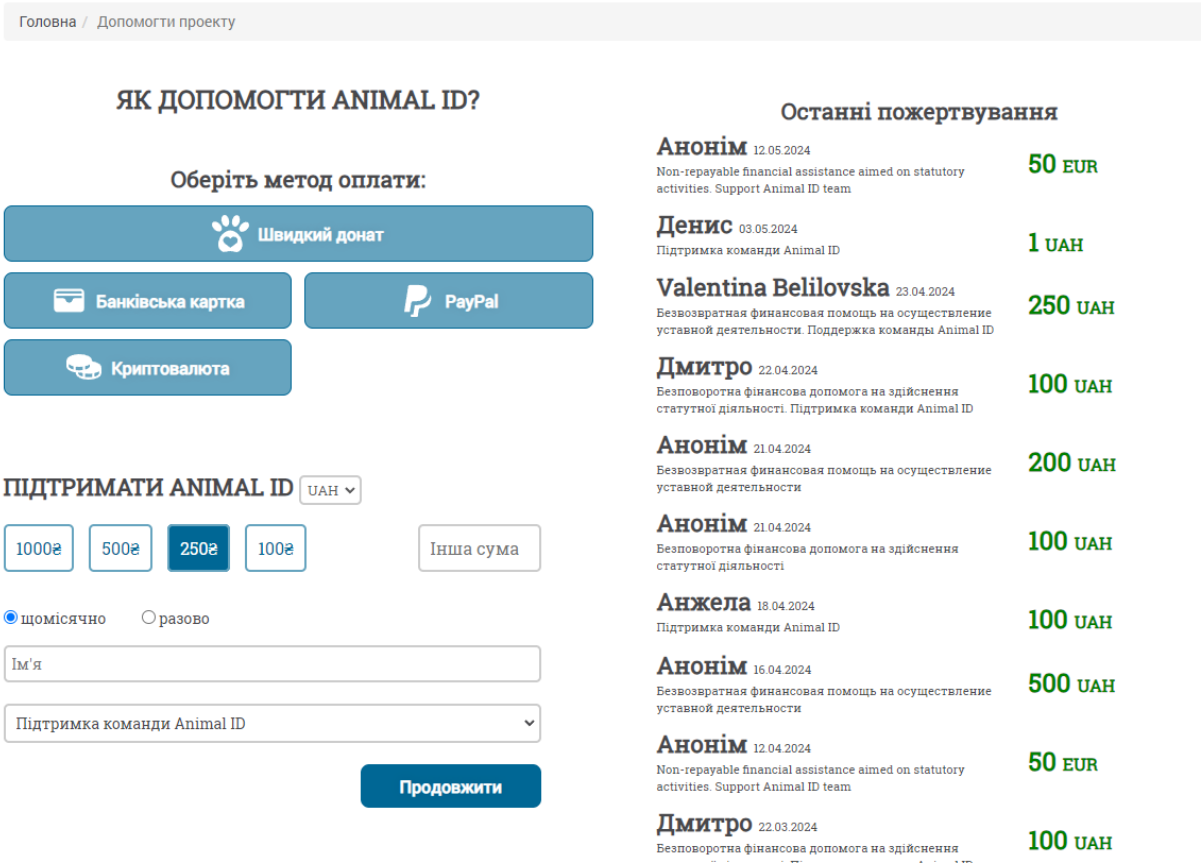


Рис. 1.6 Приклад сторінки з допомогою

1.5 Таблиця порівнянь аналогів

На основі аналізу аналогічних продуктів складемо таблицю для порівняння цих продуктів з нашим додатком, в якій “+” означає наявність відповідного функціоналу, а “-” – його відсутність.

Таблиця 1.1 ілюструє порівняння чотирьох онлайн сервісів для адопції тварин: Happy paw, «ЛКП» Лев, Animal-id та Adoptly. Кожен з них має різний набір функціональних можливостей.

Таблиця 1.1

Порівняння аналогів сервісів з онлайн адопції

Показник	**Happy paw**	**«ЛКП» Лев**	**Animal-id**	***Adoptly***
Наявність відкритої API документації	-	-	-	+
Можливість реєстрації та авторизації користувачів	+	-	+	+
Можливість додати тварину у розділ «Favorites»	+	-	+	+
Можливість надіслати донат на допомогу тварин	+	+	+	+
Можливість вибору випадкової тварини	-	-	-	+

Отже, за результатами дослідження предметної області було виявлено, що розроблюваний продукт не має прямих аналогів.

2 ВИБІР ЗАСОБІВ ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

2.1 Мова програмування JavaScript

JavaScript – це кросс-платформенна, динамічно-типізована мова програмування, яка є однією з найпопулярніших у світі. Мова дуже універсальна, оскільки, застосовуючи різні рушії, бібліотеки та фреймворки, її можна використовувати у:

- розробці front-end частині застосунку
- розробці back-end частині застосунку
- мобільній розробці
- розробці десктопних застосунків
- розробці консольних інструментів
- розробці телеграм-ботів
- розробці відеоігор

Однією з ключових особливостей JavaScript є підтримка асинхронного програмування за допомогою механізму колбеків, промісів і синтаксису async/await. Це дозволяє ефективно обробляти запити до серверів, налаштовувати роботу з таймерами, взаємодію з базами даних та іншими ресурсами, не блокуючи основний потік виконання коду. JavaScript має багату екосистему бібліотек та фреймворків, які значно спрощують розробку додатків.

Ще однією важливою перевагою JavaScript є його крос-платформеність. Код, написаний на JavaScript, може виконуватися на будь-якому пристрої, що підтримує веб-браузер. Це включає не тільки комп’ютери та ноутбуки, але й мобільні пристрої, планшети та навіть деякі телевізори.

Основною перевагою JavaScript на стороні клієнта є інтерактивність. Завдяки JavaScript можна реалізовувати такі функції, як валідація форм, динамічне оновлення

контенту, створення інтерактивних карт та графіків, анімація елементів та багато іншого.

JavaScript також може використовуватися на стороні сервера завдяки платформі Node.js. Це відкриває нові можливості для розробників, дозволяючи використовувати одну мову програмування для обох частин веб-додатка – клієнтської і серверної. Node.js забезпечує високу продуктивність завдяки неблокуючій моделі вводу-виводу, що робить його ідеальним для розробки масштабованих мережевих додатків.

Щодо основних парадигм, JavaScript підтримує об’єктно-орієнтоване програмування, функціональне програмування та імперативний стиль, що дозволяє розробникам вибирати найзручніший підхід до вирішення конкретних завдань. Не можна не згадати про механізм прототипного наслідування, який активно використовується у багатьох внутрішній методах чи класах. Все це робить мову дуже гнучкою і придатною для проектів різного рівня.

Серед інших переваг JavaScript варто відзначити його активну спільноту розробників і постійне вдосконалення мови. Веб-ресурси, такі як GitHub, Stack Overflow, та численні блоги і форуми надають величезну кількість прикладів коду, рішень проблем та рекомендацій. Це дозволяє розробникам швидко знаходити відповіді на свої запитання. ECMAScript, стандарт, на основі якого розробляється JavaScript, регулярно оновлюється, додаючи нові функції та можливості, що робить мову ще більш потужною і гнучкою. Останні версії включають такі корисні можливості, як деструктуризація об’єктів, стрілкові функції, шаблонні рядки та багато інших.

Завдяки своїй простоті у вивченні та використанні, JavaScript є ідеальним вибором для початківців у програмуванні, а його потужність і гнучкість роблять його незамінним інструментом для досвідчених розробників. Розширена підтримка з боку сучасних браузерів та активна розробка нових стандартів забезпечують JavaScript стабільне місце в майбутньому веб-технологій.

Підсумовуючи, JavaScript – це універсальна мова програмування, яка є невід'ємною частиною сучасної веб-розробки. Завдяки своїм можливостям та широкому спектру застосувань, вона залишається однією з найпопулярніших мов у світі програмування.

2.2 Node js

Node js – це відкрита серверна платформа, що використовує JavaScript для створення масштабованих веб-додатків з відмінною продуктивністю.[2] Вона була створена Раяном Далем, автором мови програмування JavaScript, і запущена в 2009 році. Node js використовує архітектуру, засновану на подіях, яка не блокує ввод-вивід, тому вона може обробляти численні користувацькі запити та активність даних.

Найважливішим серед усіх компонентів платформи є: opernsource двигун V8, який розробили у данському відділенні Google, та бібліотека Libuv, що містить механізм Event Loop, який, у свою чергу, відповідає за асинхронну поведінку. Також можна виділити декілька вбудованих модулів:

- fs (file system): робота з файлами та каталогами
- net: мережеві комунікації
- http: робота з HTTP-протоколом
- path: робота з шляхами до файлів
- crypto: криптографічні функції

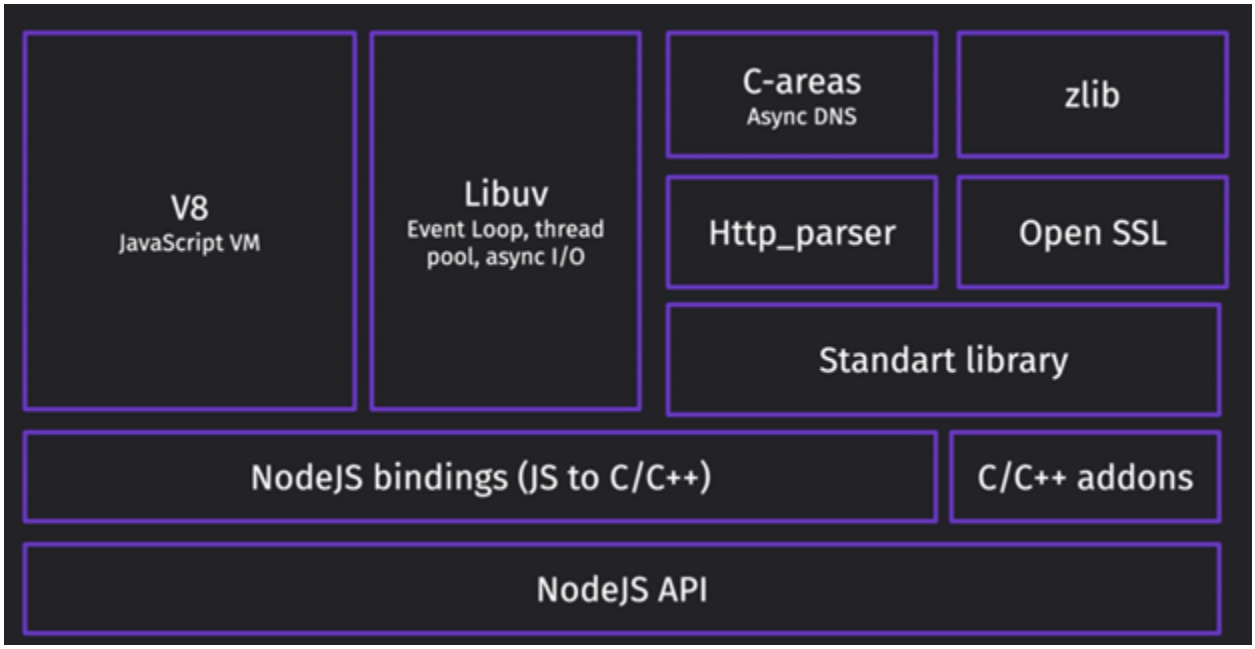


Рис. 2.1 Складові Node js

Загалом, рівні представлення коду у рушії V8 можна розділити на:

1. JavaScript code.
2. Abstract Syntax Tree.
3. Byte code.
4. Машинний код.

Abstract Syntax Tree – орієнтоване дерево, в якому внутрішні вершини співставлені з відповідними операторами мови програмування, а листя з відповідними операндами створене парсером.

ByteCode – машинонезалежний код низького рівня, більшість інструкцій байт-коду еквівалентні одній або кільком командам Асемблера. За створення ByteCode в Node.js відповідає інтерпретатор Ignition. Трансляція в байт-код займає проміжне положення між інтерпретацією та компіляцією в машинний код. Фактично, bytecode – це абстракція машинного кода. Байт-код називається так тому, що довжина кожного коду операції – один байт, але довжина коду команди різна. Кожна інструкція є

однобайтовим кодом операції від 0 до 255, за яким слідують такі параметри, як регістри або адреси пам’яті.

Шлях javascript коду всередині двигуна V8 починається з того, що скрипт потрапляє до парсеру, котрий створює Abstract Syntax Tree, потім по ньому інтерпретатор Ignition створює ByteCode.[1] Після чого, у певних випадках, наприклад, коли певна функція повторюється декілька разів, V8 припускає (за допомогою механізму hidden class), що код можна оптимізувати, направляє байт код до Turbofan компілятору, щоб отримати потім оптимізований машинний код. У випадку, якщо байт код не зможе бути оптимізований, байт код направляється до Sparkplug компілятора, що компілює у неоптимізований машинний код. Також можливий і процес деоптимізації, у випадку, коли байт код потрапив до Turbofan компілятора, проте код не можна оптимізувати, через, наприклад, зміни дій, котра постійно повторювалась. У такому випадку Turbofan делегуватиме компіляцію байт коду Sparkplug, котрий створить неоптимізований машинний код.

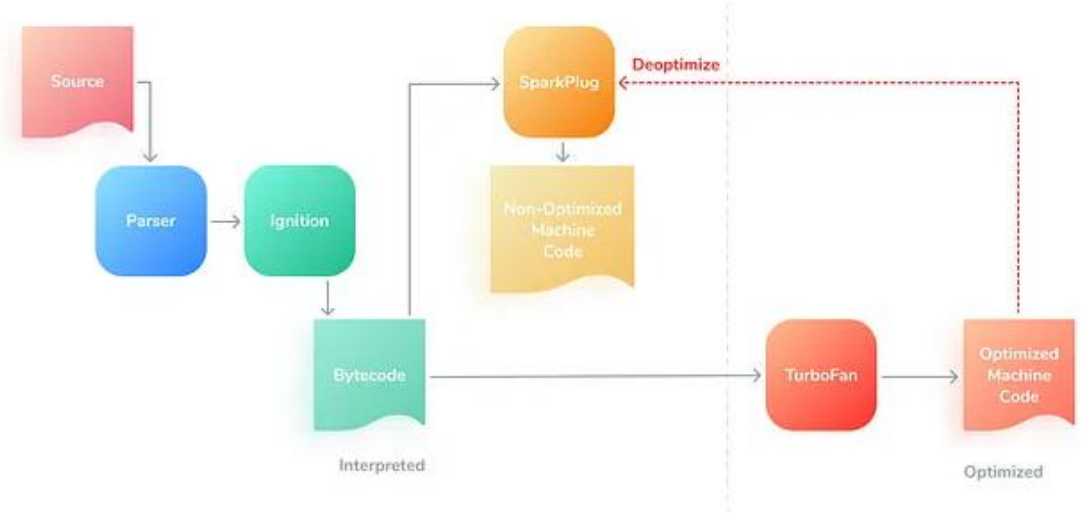


Рис. 2.2 Схема компіляції javascript коду в рушії V8

Не можна не згадати про роботу Garbage Collector. Node js розрізняє два простори, де існують об’єкти в памяті, нове (яке теж поділяється на два: nursery, intermediate) та старе, вони відрізняються різними алгоритмами, які до них використовують, щоб ефективно видаляти пам’ять. Спочатку всі об’єкти знаходяться у новому просторі, після певного часу вони переносяться у старий простір. Для нового простору використовується алгоритм parallel scavenger, котрий паралельно помічає, копіює об’єкт (тобто виконує дефрагментацію памяті, її зменшення) та обновляє вказівники. Для старого покоління використовується алгоритм mark and sweep, котрий рахує кількість посилань на певний об’єкт, масив, функцію, та з головного елементу намагається до них добратись та помітити. Потім алгоритм видаляє ті елементи, котрі не були помічені, тим самим, він вирішує проблему кругової залежності.

Three color mark – обновлена версія mark and sweep, алгоритм помічає по дереву вкладені елементи трьома кольорами в декількох ітераціях, оскільки mark and sweep default зупиняв додаток на певний час, щоб пройтись по всьому дереву, котрий міг стати критичним, а у нашому випадку алгоритм зупиняється не на довгий час, бо фарбує лише елементи, які є його нащадками.

Допомогти Garbage Collector працювати ефективніше не можна ніяк, можна лише не допускати витоку пам’яті . Найчастіші причини витоку пам’яті:

- Використання глобальних змінних
- Множинні посилання об’єктів
- Замикання

Рис. 2.3 Приклад роботи Garbage Collector під час роботи застосунку

Event Loop – механізм, котрий описаний в C бібліотеці libuv, який забезпечує Node js асинхронну, неблокуючу поведінку.[3] Node не працює за принципом виділення окремого потоку для кожного запиту, натомість, весь JavaScript код виконується в єдиному потоці, в якому опрацьовується цикл подій. При старті, Node ініціалізовує Event Loop, виконує початковий код, що запускає таймери, робить запити та інші дії, а тоді переходить до опрацювання циклу подій.

Двигун V8 виконує код line-by-line та поміщає його до стеку виклику (call stack), котрий працює за принципом Last-In-First-Out. Якщо в процесі виконання він натрапляє на таймери, асинхронні роботи з файлами чи інші асинхронні дії, то він звертається до бібліотеки libuv, яка звертається до демультиплексора подій (event demultiplexer), котрий реєструє запит та не блокує додаток. Після демультиплексора event loop перекладає подію у чергу подій (event queue), коли настає черга саме цієї події, event loop виділяє вільний потік з worker threads, котрий обробляє цю операцію та повертає колбек з результатом, котрий знову event loop кладе у call stack. Якщо у колбеці буде вкладена асинхронна операція, то event loop знову звернеться до демультиплексора подій і цикл повториться.

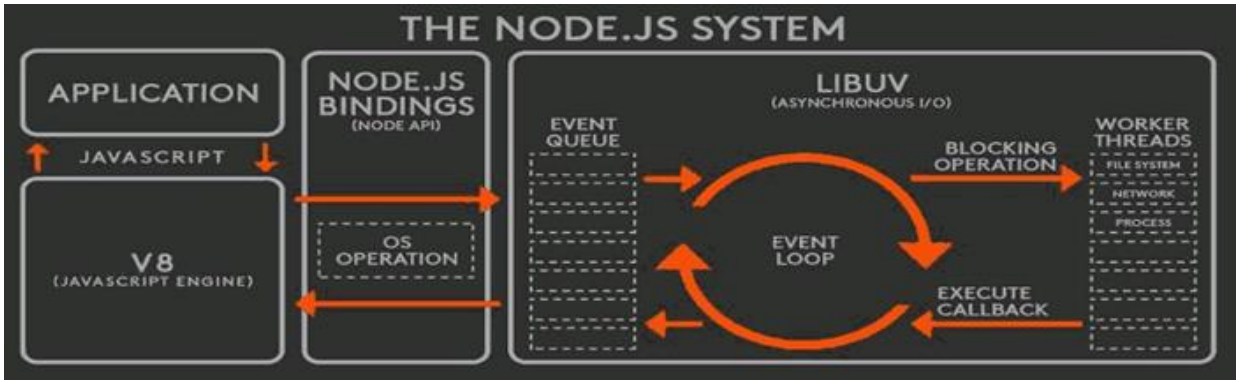


Рис. 2.4 Візуалізація роботи Event Loop

2.3 Express Js

Express js – це популярний веб-фреймворк для Node js, що дозволяє швидко і ефективно створювати гнучкі серверні додатки.[11] Завдяки своїй простоті і мінімалістичному підходу, Express js є ідеальним вибором як для новачків, так і для досвідчених розробників. Висока продуктивність фреймворку досягається завдяки легкій вазі і оптимізованому коду, що знижує затримки і забезпечує швидкий відгук серверу. Цей проект підтримується компанією StrongLoop, що є частиною IBM, і постійно оновлюється, забезпечуючи стабільність і безпеку.

Express js використовує систему middleware, що дозволяє розширювати функціональність додатків, додаючи обробники запитів, логування, автентифікацію, обробку помилок тощо. Це надає розробникам гнучкість і контроль над тим, як обробляються HTTP-запити і відповіді. Фреймворк інтегрується з великою кількістю шаблонізаторів, таких як Pug або EJS, що спрощує рендеринг динамічних HTML-сторінок. Також Express js добре працює з базами даних, включаючи MongoDB через Mongoose, що робить його відмінним вибором для розробки повноцінних веб-додатків.

Завдяки простому і інтуїтивно зрозумілому інтерфейсу для маршрутизації запитів, розробникам вдається створювати масштабовані та комплексні маршрути з мінімумом коду. Фреймворк підтримує як синхронні, так і асинхронні операції, що

робить його ідеальним для обробки великої кількості одночасних запитів від користувачів.

Суттєвою перевагою Express js є його активна спільнота та велика кількість доступних плагінів і модулів, які можна легко інтегрувати в проект, що значно прискорює процес розробки. Документація фреймворку детальна і добре структурована, що полегшує освоєння для розробників з мінімальним досвідом.

Підсумуємо, Express js є потужним інструментом для розробки серверних додатків. Він пропонує високу продуктивність, гнучкість та багатий набір функцій, що дозволяють створювати різноманітні веб-додатки – від простих до комплексних API рішень.

2.4 Mongo DB

Mongo DB – це документо-орієнтована система керування базами даних (СУБД) з відкритим кодом, яка набула широкої популярності завдяки своїй гнучкості, масштабованості та простоті використання. Вона належить до категорії NoSQL баз даних.[12]

Однією з ключових особливостей Mongo DB є відсутність структурованості, що дозволяє зберігати дані з різною будовою в одній і тій же колекції. Це надає розробникам велику свободу у проектуванні і зміні моделей даних без необхідності проектування та написання складних міграцій.

Mongo DB використовує BSON (Binary JSON) для зберігання документів, що дозволяє ефективно зберігати та обробляти великі обсяги даних. Це забезпечує високу продуктивність при читанні і запису даних, роблячи Mongo DB придатною для обробки великих даних та інших аналітичних завдань.

Реплікація в Mongo DB – це механізм, який дозволяє створювати копії даних для забезпечення високої доступності та стійкості до збоїв. Реплікаційні набори складаються з первинної копії даних та однієї або більше вторинних копій. Це

дозволяє швидко відновлюватися після відмови та забезпечує безперервність роботи додатків.

Mongo DB легко масштабувати горизонтально, варто лише додати більше серверів до кластера. Шардінг – це механізм для масштабування бази даних, який дозволяє розподіляти дані по кількох серверах. Це особливо важливо для веб-додатків з великою кількістю користувачів та постійно зростаючими обсягами даних.

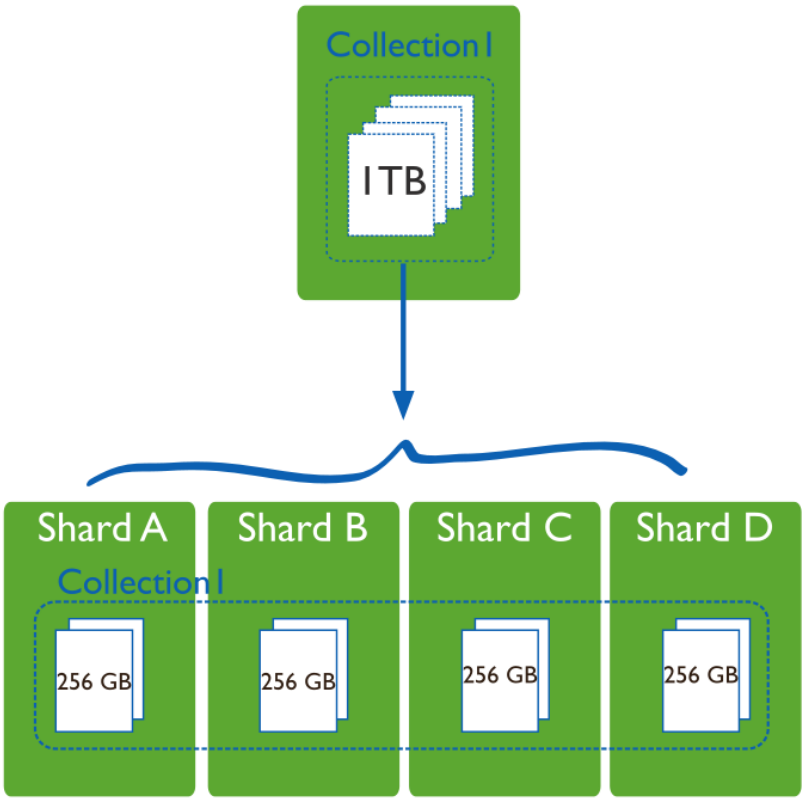


Рис. 2.5 Приклад шардингу в Mongo DB колекції Collection I

Mongo DB також має зручний API, який дозволяє виконувати складні запити, фільтрацію, сортування даних. Агрегаційний фреймворк Mongo DB надає можливість виконувати складні операції над даними, такі як обчислення підсумкових значень, групування і перетворення даних.

Mongo DB Atlas – це хмарний сервіс від Mongo DB, що спрощує розгортання, управління та масштабування баз даних. Він забезпечує автоматичне резервне

копіювання, моніторинг продуктивності та безпеку, дозволяючи зосередитися на розробці додатків.

Окрім цього, серед переваг Mongo DB варто виділити: активну спільноту користувачів і розробників, регулярні оновлення, детальну документацію, велику кількість офіційних драйверів для різних платформ .

2.5 React

React – безкоштовна opensource бібліотека для побудови користувацьких інтерфейсів, створена компанією Facebook. Вона була випущена у 2013 році і з того часу здобула широку популярність серед розробників завдяки своїй простоті та ефективності. [4]

Основою React є компоненти – незалежні, багаторазові частини коду, які представляють певні елементи інтерфейсу користувача. Кожен компонент може мати свій стан (state) та властивості (props). Створювати компоненти можна через класи або через функції.

Однією з ключових особливостей React є **віртуальний DOM (Virtual DOM)**. Він є копією реального DOM і використовується для відстеження змін та оптимізації оновлень інтерфейсу. Завдяки віртуальному DOM React може ефективно перерендерювати лише ті частини сторінки, які дійсно змінилися, замість оновлення всього DOM .

React також підтримує JSX – розширення синтаксису JavaScript, що дозволяє писати HTML-подібний код безпосередньо в JavaScript. Це спрощує створення інтерфейсів та робить код більш зрозумілим і зручним для читання. Також, завдяки впровадженому синтаксису JSX, React краще повідомляє розробників про наявність у коді певних помилок.

```jsx
function renderLogin({ prefilledEmail }) {
  return (
    <section>
      <input type="email" placeholder="email" value={prefilledEmail} />
      <input type="password" placeholder="password" />
      <button>Log In!</button>
    </section>
  );
}
```

Рис. 2.6 Приклад коду JSX, функціональний React компонент

React можна використовувати як для розробки односторінкових додатків (SPA), так і для більш комплексних, багатосторінкових рішень. Для цього часто використовують додаткові бібліотеки, такі як React Router для маршрутизації або Redux для управління станом компонентів.

Окрім веб-розробки, React також має версію для мобільних додатків – React Native, яка дозволяє створювати нативні мобільні додатки для iOS та Android з використанням адаптованої архітектури компонентів.

Таким чином, React пропонує потужний та гнучкий підхід до розробки інтерфейсів, який спрощує процес створення та підтримки складних додатків, забезпечуючи високу продуктивність і легкість у використанні.

2.6 Додаткові інструменти та засоби розробки

2.6.1 MUI

MUI – це бібліотека React з відкритим кодом, від Google. Вона пропонує широкий спектр компонентів, таких як кнопки, текстові поля, списки, таблиці та діалогові вікна, які відповідають принципам Material Design. Це полегшує розробникам створення красивих та зручних інтерфейсів користувача.[5]

Компоненти можна легко налаштовувати за допомогою тем, що дозволяє адаптувати зовнішній вигляд додатка до конкретних потреб проекту. Використовуючи систему тем, розробники можуть змінювати кольори, типографіку, відступи та інші стилістичні параметри.

MUI також підтримує адаптивний дизайн, що дозволяє створювати інтерфейси, які добре виглядають на різних пристроях і розмірах екрану. Компоненти бібліотеки вдало оптимізовані та мають невеликий розмір, що сприяє швидкому завантаженню сторінок.

Крім того, MUI має хорошу документацію та активну спільноту, що робить її легкою у вивченні та використанні. Документація містить багато прикладів використання компонентів, а також інструкції по налаштуванню та інтеграції бібліотеки в проекти.

2.6.2 Nodemailer

Nodemailer – це бібліотека для додатків, що функціонують на платформі Node js, яка дозволяє легко надсилати електронні листи. Nodemailer підтримує відправку як простих текстових листів, так і листів у форматі HTML. Крім того, бібліотека дозволяє додавати вкладення, наприклад, файли або зображення. Також можливе використання шаблонів для листів.[7]

Основною перевагою Nodemailer є її легкість у використанні. Для налаштування достатньо лише кількох рядків коду, щоб створити транспортер для відправки листів та визначити вміст самого листа.

Безпека є важливою складовою Nodemailer. Бібліотека підтримує шифрування SSL/TLS для безпечної передачі даних через поштовий сервер. Крім того, вона дозволяє використовувати OAuth2 для автентифікації, що підвищує безпеку, особливо при роботі з хмарними сервісами.

2.6.3 Visual Studio Code

Visual Studio Code (VS Code) – безкоштовний, кроссплатформений, opensource редактор коду, створений Microsoft у 2015 році. Він підтримує широкий спектр мов програмування, включаючи JavaScript, Python, Java, C++, PHP та багато інших, що дозволяє використовувати його для написання різних проектів.[9]

Основною перевагою Visual Studio Code є його здатність інтегруватися з системами керування версіями, наприклад, з Git. Це робить можливим для програмістів з легкістю управляти змінами в коді, відстежувати історію змін та здійснювати різноманітні дії прямо з інтерфейсу редактора. Користувацький інтерфейс VS Code є зрозумілим та гнучко конфігурованим, з можливістю вибору тем оформлення, налаштування гарячих клавіш та багато чого іншого.

Можливість доповнення функціоналу є ключовою характеристикою VS Code. Редактор пропонує обширний магазин плагінів, де можна знайти плагіни для підтримки нових мов програмування, інтеграції з іншими інструментами розробки, а також для додавання нових можливостей. Це надає розробникам можливість налаштувати редактор під особисті потреби та стиль роботи.

Інструменти для відладки, вбудовані в VS Code, дозволяють програмістам ефективно виявляти та усувати помилки в коді. Функції, такі як налаштування break points та моніторинг змінних в реальному часі, роблять процес відладки простим та ефективним. VS Code також підтримує роботу з контейнерами та віртуалізованими середовищами, як-от Docker , що дозволяє розробникам розробляти та тестувати додатки в ізольованих умовах.

2.6.4 Postman

Postman – це інструмент для тестування API. Завдяки своїй функціональності та зручності у використанні, Postman став незамінним інструментом для багатьох розробників і тестувальників.[10] Він дозволяє розробникам виконувати HTTP-запити до веб-сервісів, аналізувати відповіді й автоматизувати тести. Postman надає зручний

графічний інтерфейс, що дозволяє легко створювати і зберігати запити різних типів, таких як GET, POST, PUT, PATCH, DELETE та інші. Користувачі можуть налаштовувати заголовки, тіла запитів і параметри, а також переглядати відповіді сервера у різних форматах (JSON, XML, HTML).

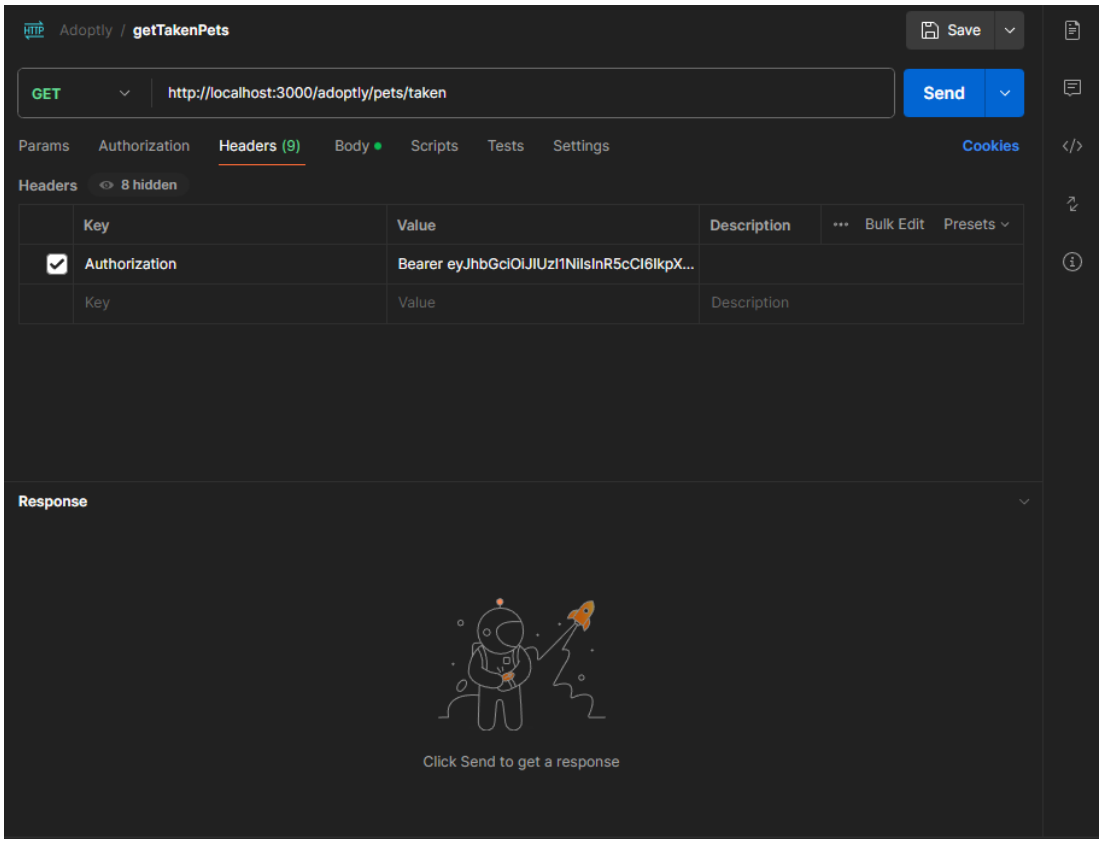


Рис. 2.7 Приклад GET HTTP запиту з Headers токеном в Postman

Однією з головних переваг Postman є його здатність створювати колекції запитів, які можна організувати в набори тестів. Це значно полегшує тестування складних API з великою кількістю маршрутів. Крім того, Postman надає можливості автоматизації за допомогою Newman – командного інтерфейсу, який дозволяє запускати колекції запитів з командного рядка. Це особливо корисно для інтеграції тестів у CI/CD процеси. Інструмент підтримує різні типи аутентифікації, включаючи OAuth, API ключі та інші методи.

2.6.5 Swagger

Swagger – це популярний інструмент для розробки, документування та тестування RESTful API.[13] Він дозволяє автоматично генерувати просту й інтерактивну документацію для розробників. Основною частиною Swagger є специфікація OpenAPI, що дозволяє розробникам описувати структуру свого API у форматі JSON або YAML. Це забезпечує стандартний спосіб документування та спрощує інтеграцію з іншими системами.

Ще один важливий компонент – Swagger Editor, що надає можливість редагувати та перевіряти OpenAPI специфікації в інтерактивному середовищі. Він підтримує автодоповнення та валідацію, що значно спрощує процес створення API документації.

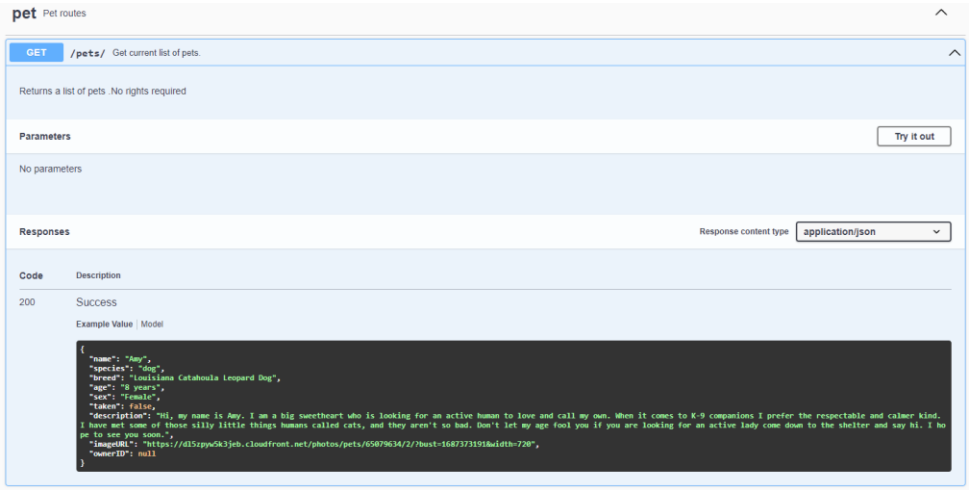


Рис. 2.8 Приклад задокументованого роуту через Swagger

Swagger також пропонує інструменти для генерації серверного та клієнтського коду на основі специфікації OpenAPI. Це означає, що розробники можуть швидко почати роботу над проектом, маючи вже готові шаблони для різних мов програмування та фреймворків.

Важливою перевагою Swagger є його підтримка великою спільнотою, а також інтеграція з багатьма популярними інструментами DevOps та CI/CD, що полегшує автоматизацію процесів розробки та тестування.

2.6.6 Jest

Jest – це інструмент для тестування JavaScript, що широко використовується у спільноті розробників для тестування додатків. Jest забезпечує просте налаштування, високу швидкість виконання тестів та гнучкість у роботі з різними проектами.[6]

Однією з ключових особливостей Jest є його здатність працювати без додаткової конфігурації. Він автоматично знаходить тестові файли, в яких закінчення назви містить `.test.js` або `.spec.js`, і запускає їх. Це дозволяє розробникам швидко інтегрувати Jest у свої проекти без зайвих налаштувань.

Jest підтримує модульне тестування, що дозволяє тестувати окремі частини програми незалежно від інших. Це досягається завдяки можливості створення моків (mocks) і шпигунів (spies), які імітують поведінку реальних залежностей, що значно спрощує тестування окремих модулів.

Крім того, Jest пропонує знімкове тестування (snapshot testing), яке дозволяє зберігати та порівнювати знімки вихідних даних або інтерфейсів користувача з попередніми версіями. Це особливо корисно для тестування інтерфейсів користувача, оскільки допомагає виявити ненавмисні зміни у відображенні компонентів.

Jest добре інтегрується з багатьма популярними бібліотеками, такими як React, Vue та Angular, що робить його ідеальним вибором для сучасних веб-додатків. Він також підтримує TypeScript, що розширює його можливості для більш типізованих проектів.

Інструмент забезпечує швидке виконання тестів завдяки паралельному запуску і повторному використанню попередніх результатів. Це особливо важливо для великих проектів, де швидкість виконання тестів може суттєво впливати на продуктивність розробників.

Jest має потужні можливості для дебагу коду, що дозволяє розробникам отримувати детальну інформацію про тестові сценарії та рівень покриття коду тестами. Це сприяє виявленню та виправленню помилок на ранніх етапах розробки.

3 ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Функціональні вимоги

Серед функціональних вимог до застосунку можна виділити наступні:

- Реєстрація та авторизація користувачів. Користувачі можуть створювати аккаунти, активувати їх через електронну пошту.
- Перегляд тварин. Користувачі можуть переглядати список тварин, які доступні для адопції.
- Пошук тварин. Користувачі можуть шукати тварин за різними критеріями.
- Адопція тварин. Авторизовані користувачі можуть подати запит на адопцію тварини, яку пізніше розгляне адміністратор.
- Права адміністратора. Адміністратор може додавати, видаляти та редагувати інформацію про тварин та користувачів, модерувати заявки на адопції від користувачів.
- Донат. Користувачі можуть надсилати донати на допомогу тваринам.
- Випадковий пошук. Користувачі можуть випадковим чином вибрати тварину та отримати детальну інформацію про неї.
- Документація. Користувачі можуть переглядати документацію для серверної частини.

3.2 Не функціональні вимоги

Серед нефункціональних вимог до застосунку можна виділити наступні:

- Адаптивність. Інтерфейс застосунку повинен бути адаптивним до різних пристроїв та екранів

- Стійкість. Система повинна бути стійкою до відмов, незважаючи на перепад трафіку від користувачів.
- Безпека. Чутливі данні, такі як, паролі, мають зберігатись у базі даних в зашифрованому вигляді.
- Тестування. Всі роути серверної частини повинні бути прокриті end-to-end тестами. Клієнтська частина повинна бути протестована мануально.

3.3 Моделювання об’єкту проектування

3.3.1 Проектування архітектури

Застосунок складається з двох частин : серверної та клієнтської. Користувачі взаємодіють з інтерактивним інтерфейсом front-end частини, яка в свою чергу, робить запити на API back-end частини. Інформація про : тварин, користувачів, притулки зберігається в базі даних, з якою може взаємодіяти лише серверна частина. Схема архітектури застосунку ілюстрована на рисунку 3.1.

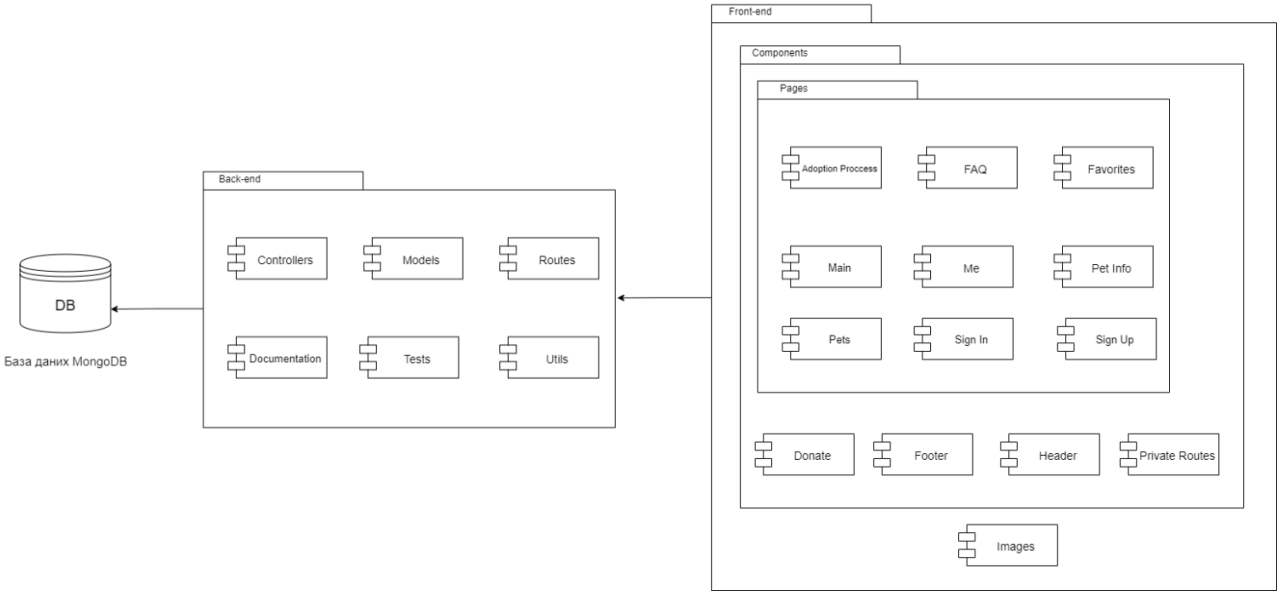


Рис. 3.1 Схема архітектури застосунку

Backend частина використовує Express.js для створення RESTful API і взаємодії з базою даних. Структура включає наступні модулі:

- Controllers : містять логіку обробки запитів, яка отримує дані з моделей, опрацьовує їх і повертає відповіді клієнту.
- Models : описують структуру даних і взаємодію з базою даних. Використовуються для створення, читання, оновлення та видалення (CRUD) записів у базі даних.
- Routes: визначають endpoints серверу та зв'язують їх з відповідними контролерами. Завдяки цьому модулю, конфігурується система маршутизації back-end застосунку. Додатково, в приватних роутах перевіряється токен доступу і права користувачів через механізм middlewares.
- Documentation: містить документацію API, написану через інструмент Swagger в форматі json, яка описує доступні маршрути, методи запитів та структури даних.
- Tests: містять end-to-end тести для перевірки коректності роботи контролерів, моделей та інших компонентів.
- Utils: включають допоміжні функції та утиліти, які використовуються в різних частинах застосунку.

Frontend частина побудована на React та використовує Material-UI для створення інтерфейсу користувача. Основним елементом, з яких складаються усі сторінки є компонент . В корені проекту існує папка images, в якій зберігаються логотипи та інші картинки. Структура сторінок наступна:

- Main: Головна сторінка застосунку.
- Adoption Process: Сторінка з розлогою інформацією про процес адопції.
- FAQ: Сторінка з часто задаваними питаннями.
- Sign In: Сторінка для входу користувачів.

- Sign Up: Сторінка для реєстрації користувачів.
- Me: Сторінка з інформацією про користувача.
- Favorites: Сторінка зі збереженими улюбленими тваринами.
- Pets: Сторінка зі списком доступних для адопції тварин.
- Pet Info: Сторінка з детальною інформацією про конкретну тварину.

З компонентів можна виділити :

- Private Routes: Компонент для маршрутизації, який забезпечує доступ до приватних сторінок лише для авторизованих користувачів, завдяки парсингу токену.
- Donate: Компонент для донату на допомогу тваринам.
- Footer: Компонент з додатковою інформацією, контактами та партнерами, який використовується на всіх сторінках, окрім авторизаційних. Розташований в самому низу сторінки.
- Header: Компонент з логотипом, та динамічними розділами , який використовується на всіх сторінках, окрім авторизаційних. Якщо користувач авторизований, то додатково на хедері розташовані кнопки для: виходу з облікового запису, перегляду favorites, перегляду особового запису. У випадку не авторизованого користувача, додаються кнопки для реєстрації та авторизації. Розташований вгорі сторінки.

3.3.2 Проектування бази даних

Загалом, база даних містить 3 колекції, в яких зберігається, відповідно, інформація про: користувачів, тварин, притулки. Оскільки базою даних є Mongo DB, яка є не реляційною за своїм типом, відповідно неможливо вказати первинні чи вторинні ключі між колекціями. Схема бази даних наведена на рисунку 3.2., де посилання на інші колекції зображені стрілками.

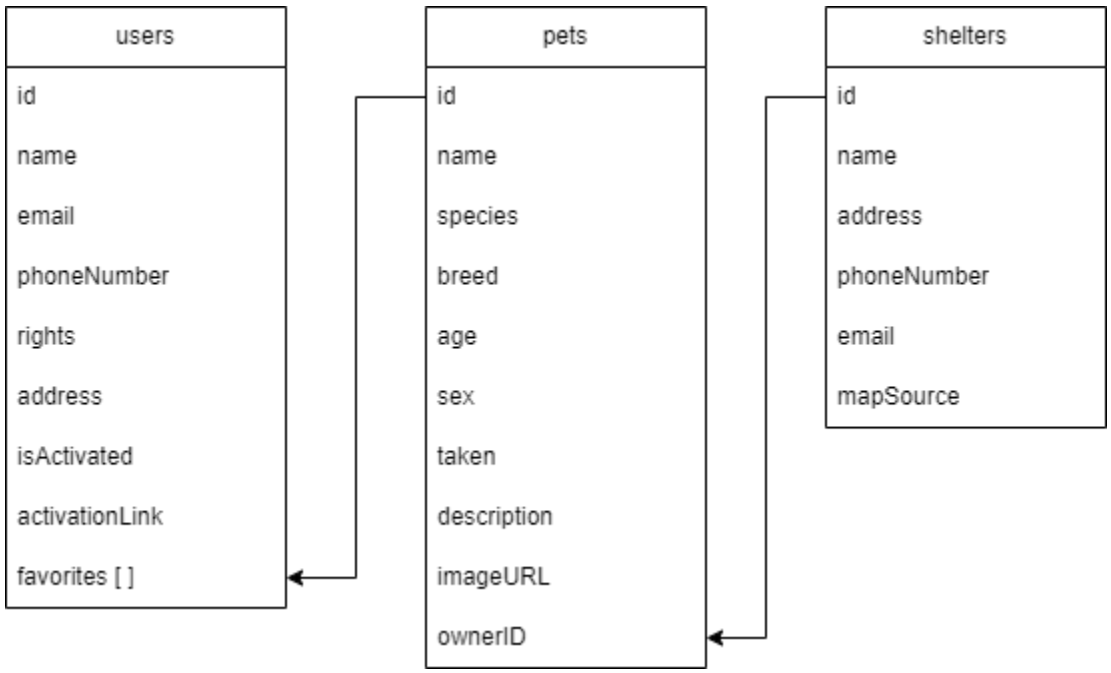


Рис. 3.2 Схема бази даних

В колекції users, поля: name,email,phoneNumber та address зберігають персональну інформацію про користувача. Поле rights відповідає за права юзера, а isActivated містить інформацію про те, чи був активований аккаунт через посилання в листі, яке було відправлено на електронну пошту. Самий же унікальний набір символів, який використовується для генерації роуту, зберігається в activationLink. У favorites зберігається масив з полями id колекції pets, тобто тих тварин, який користувач додав до улюблених.

В колекції pets, поля: name,species,breed,age,sex,description,imageURL містять інформацію про тварину. В полі taken зберігається булева інформація про те, чи тварина досі шукає власника,а відповідно в ownerID – id власника. Якщо taken має стан false, то в ownerID знаходиться id притулку, якщо ж true – то в ownerID знаходиться id користувача, який залишив запит на адопцію.

В колекції shelters, усі поля зберігають інформацію про конкретний притулок. Можна виділити mapSource, оскільки там містяться координати розташування притулку в Google Maps.

3.3.3 Діаграма діяльності

Діаграма діяльності використовується для опису логіки певних процесів в застосунку мовою UML. На рисунку 3.3 зображена діаграма діяльності для процесу адопції випадкової тварини.

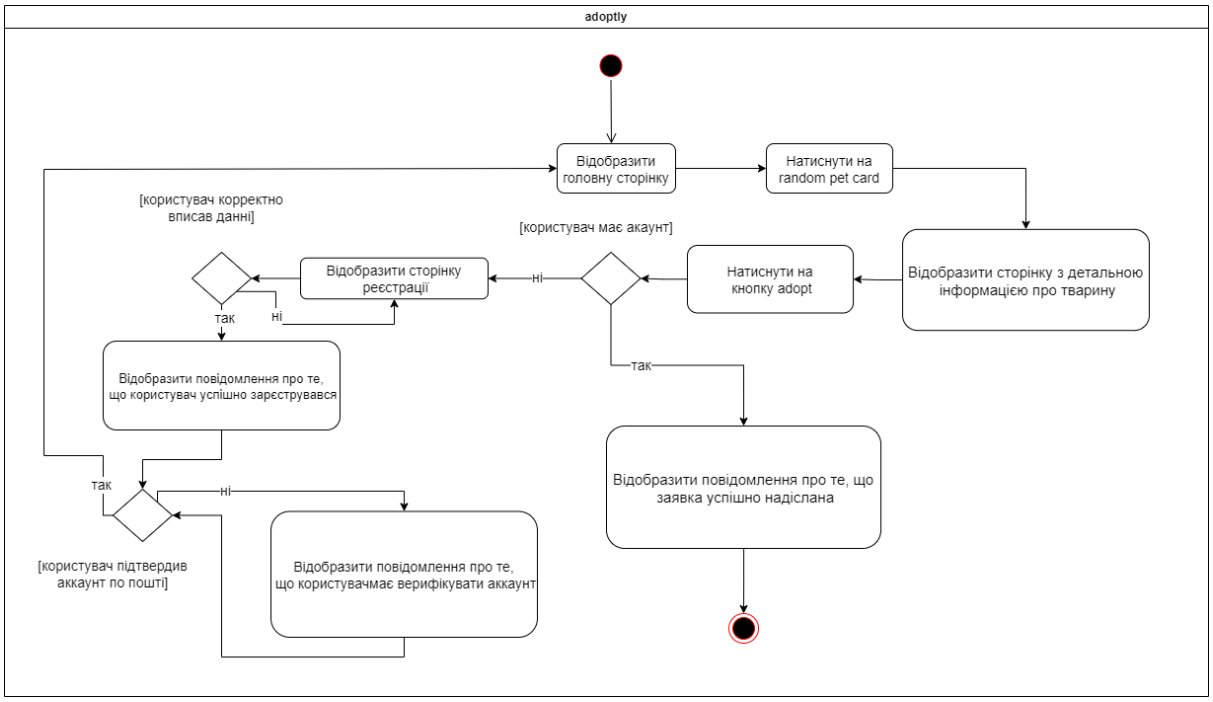


Рис. 3.3 Діаграма діяльності процесу адопції випадкової тварини

3.3.4 Діаграма використання

Діаграма використання використовується для візуалізації функціональних можливостей та вимог до системи мовою UML.

На рисунку 3.4 зображена діаграма використання застосунку

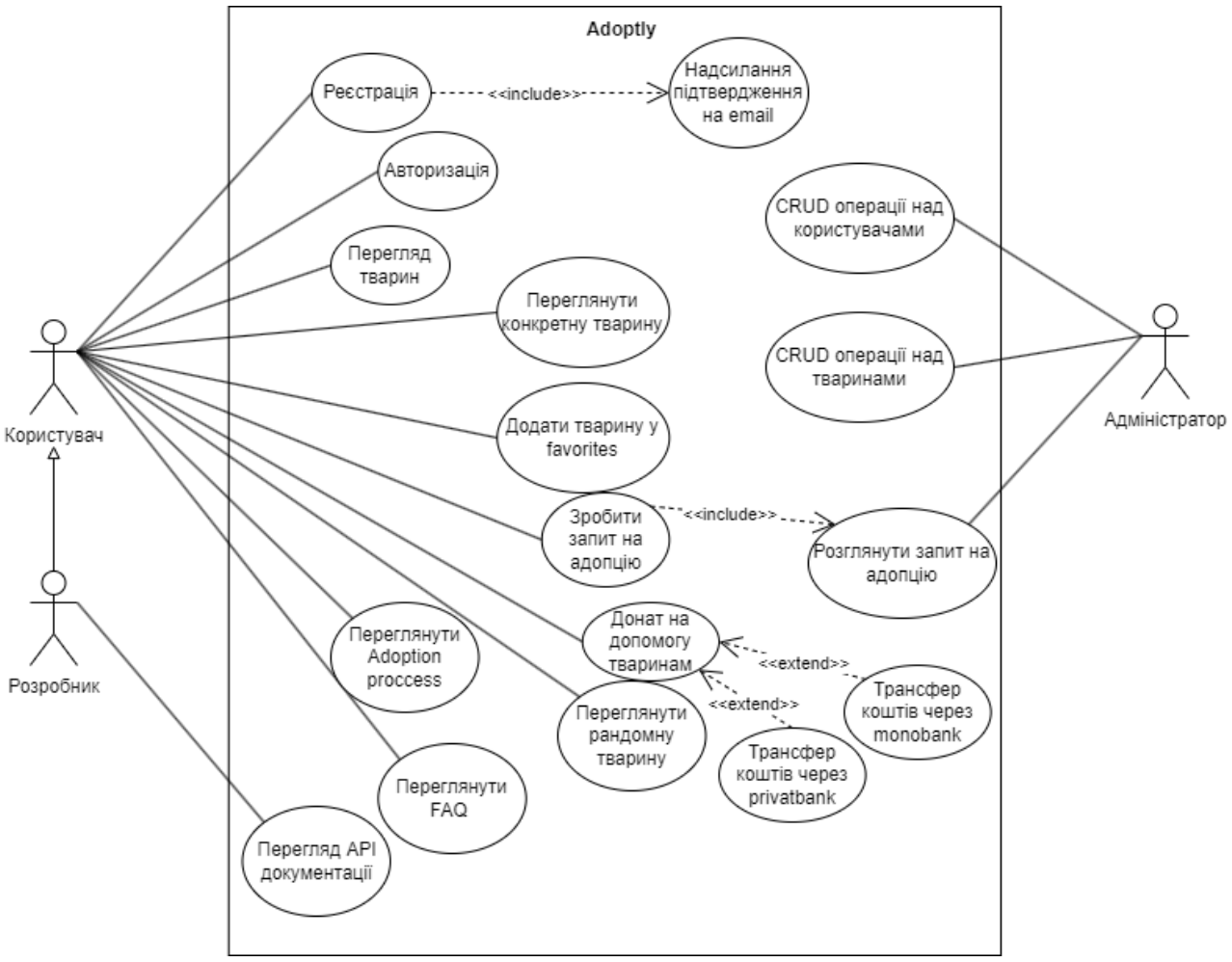


Рис. 3.4 Діаграма використання

3.3.5 Мапа сайту

Мапа сайту – схема, на якій зображена структура сторінок та звязки між ними, завдяки цьому, суттєво спрощується навігація на застосунку між доступними сторінками. На рисунку 3.5 проілюстрована мапа сайту.

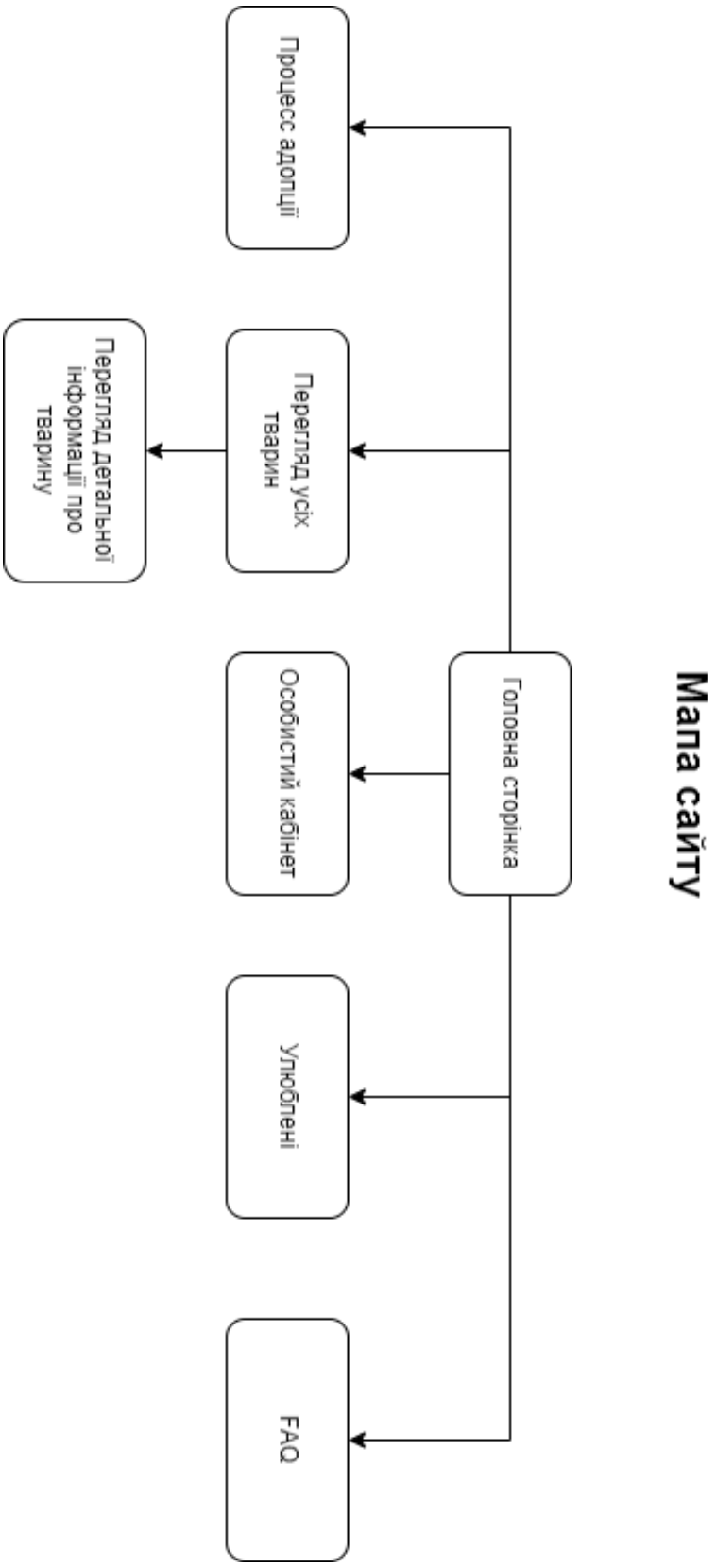


Рис. 3.5 Мапа сайту

4 РОЗРОБКА ТА ТЕСТУВАННЯ ЗАСТОСУНКУ

4.1 Реалізація API роутів

Першими, після базового налаштування API серверу, були розроблені схеми бібліотеки mongoose,які надали б мінімальну структуризацію колекціям(users,pets,shelters) для нереляційної бази даних .Вони в подальшому були використані в контроллерах, для операцій з Mongo DB.

Важливі дані, такі, наприклад, як порт, на який API сервер очікує отримання запитів від користувачів, або строка підключення бази даних, розміщені в окремому файлі конфігурації.

```
const mongoose= require("mongoose");
const app = require('./app').app
const PORT = process.env.PORT || 3000;
let MONGODB_URL = process.env.MONGODB_URL;

let server;
mongoose.connect(MONGODB_URL).then(() => {
  console.log('Connected to MongoDB');
  server = app.listen(PORT, () => {
    console.log(`Listening to port ${PORT}`);
  });
});
```

Рис. 4.1 Приклад створення серверу і підключення Mongo DB

```
var petSchema = new Schema({
  name: {
    type: String,
    required: [true, 'Name of the pet is required'],
  },
  species:{
    type: String,
    required: [true, 'Species of the pet is required']
  },
  age:{
    type: Number,
    required: [true, 'Age of the pet is required'],
    min:0,
    max:50
  },
  sex:{
    type: String,
    required: [true, 'Sex of the pet is required'],
    enum: ['male', 'female']
  },
  breed:{
    type: String
  },
  taken:{
    type: Boolean,
    default:false
  },
  description: {
    type: String,
    default:"",
  },
  imageURL:{
    type: String,
    default:"https://as1.ftcdn.net/v2/jpg/02/52/84/92/1000_F_252849218_Acdc6N696mDekuQvrCmqlFDMx4UYYF2Y.jpg"
  },
  ownerID:{
    type: mongoose.Schema.Types.ObjectId,
    default:null
  },
});
```

Рис. 4.2 Приклад mongoose схеми для колекції pets

Після цього, були створені роути та контролери для модулю авторизації користувачів, який використовує jsonwebtoken токен, серед яких, був один з динамічним параметром, оскільки він відповідає за активацію акаунту користувача по унікальному посиланню.

Jsonwebtoken — це стандарт безпеки, який використовується для передачі інформації між двома сторонами у форматі JSON, захищеної цифровими підписами.[8]

Для надсилання електронних листів, був розроблений клас MailService, який пізніше використовувався у контролері для реєстрації. Необхідні змінні для конфігурації класу були додані до .env файлу.

```
router.post('/register',authController.register)
router.post('/login',authController.login)
router.get('/logout', authController.logout)
router.get('/activate/:link',authController.activate)
```

Рис. 4.3 Приклад роутів для авторизації

```
class MailService{
    constructor()
    {
        this.transporter = nodemailer.createTransport({
            host: process.env.SMTP_HOST,
            port:process.env.SMTP_PORT,
            secure:false,
            auth:{
                user:process.env.SMTP_USER,
                pass:process.env.SMTP_PASSWORD,
            }
        })
    }
    async sendActivationMail(to,link)
    {
        await this.transporter.sendMail({
            from:process.env.SMTP_USER,
            to,
            subject:'Account activation on '+ process.env.PROJECT_NAME,
            text:'',
            html:`
                    <div>
                    <h1>Press on the link to activate your account</h1>
                    <a href="${link}">${link}</a>
                    </div>
                `
        })
    }

}
```

Рис. 4.4 Приклад класу для надсилання електронних листів

Для хешування паролю користувача використовувалась бібліотека bcrypt, а для створення унікального посилання для верифікації – бібліотека uuid. Хешовані дані

гарантують додаткову безпеку, у випадку, якщо, якимось чином, зловмисники отримають доступ до даних у базі.

```
const register = async (req, res, next) => {
  try {
    if (!req.body.password) {
      throw new apiError(400, "Bad request");
    }
    const salt = await bcryptjs.genSalt(10);
    const hashedPassword = await bcryptjs.hash(req.body.password, salt);
    const activationLink = uuid.v4();
    const newUser = new User({
      name: req.body.name,
      email: req.body.email,
      password: hashedPassword,
      phoneNumber: req.body.phoneNumber,
      address: req.body.address,
      rights: req.body.rights,
      favorites: req.body.favorites,
      isActivated:false,
      activationLink:activationLink
    });
    await mailSender.sendActivationMail(
      req.body.email,
      `http://localhost:${process.env.PORT}/${process.env.PROJECT_NAME}/auth/activate/${activationLink}`
    );
    const savedUser = await newUser.save();
    return res.status(201).json(savedUser);
  } catch (err) {
    return next(err);
  }
};
```

Рис. 4.5 Приклад контроллеру реєстрації

```
const activate = async (req, res, next) => {
  try {
    const activationLink = req.params.link;

    const user = await User.findOne({activationLink});
    if(!user)
    {
      throw new apiError(404, "User was not found");
    }
    user.isActivated=true;
    await user.save();
    return res.redirect(`http://localhost:${process.env.PORT}/${process.env.PROJECT_NAME}/pets`);
  } catch (err) {
    return next(err);
  }
};
```

Рис. 4.6 Приклад контроллеру активації акаунту

Далі був розроблений метод для парсингу токену, перевірки токену на актуальність і перевірки прав всередині токену. В подальшому, цей метод використовувався для обмеження осіб, які або не є авторизованими, або не мають прав для здійснення дії з певним роутом. Права типу “read” мають усі авторизовані користувачі, а права типу “right” мають лише адміністратори.

```
const checkRights = (requiredRight) => async (req, res, next) => {
  try {
    const token = req.get('Authorization').split(' ')[1]
    if (!token) {
      throw new apiError(401, "Unauthorized");
    }
    return jwt.verify(token, process.env.JWT_SECRET, (err, decoded) => {
      if (err) {
        throw new apiError(401, "Invalid Token");
      } else {
        if (requiredRight != decoded.rights && decoded.rights != "write") {
          throw new apiError(403, "Forbidden");
        }
        req.user = decoded;
        return next();
      }
    });
  } catch (err) {
    return next(err);
  }
};
```

Рис. 4.7 Приклад методу для парсингу та валідації токену

```
router.get('/',checkRights("write"),  userController.getUsers);
router.post('/',checkRights("write"),  userController.postUser);
router.get('/:id',checkRights("read"),  userController.getUser);
router.put('/:id',checkRights("read"),  userController.putUser)
router.delete('/:id',checkRights("write"),  userController.deleteUser);
```

Рис. 4.8 Приклад перевірки прав на users роутах

В подальшому були розроблені усі інші необхідні роути та контролери для CRUD операцій над колекціями.

```
router.get('/' , petController.getPets);
router.post('/',checkRights("write"),  petController.postPet);
router.get('/taken',checkRights("write"),  petController.getTakenPets);
router.get('/:id', petController.getPet);
router.put('/:id',checkRights("read"),  petController.putPet)
router.delete('/:id',checkRights("write"),  petController.deletePet);
```

Рис. 4.9 Приклад pets роутів

```
const deletePet = async(req,res,next)=>{
    try{
        const pet =await petModel.findById(req.params.id).exec();
        if(!pet)
        {
            throw new apiError(404, 'Pet was not found');
        }
        await petModel.findByIdAndDelete(req.params.id);
        return res.status(204).json("Pet deleted");
    }
    catch(err)
    {
        return next(err);
    }
};
```

Рис. 4.10 Приклад контролеру на видалення тварини

4.2 Створення документації

Документація була написана, використовуючи інструмент swagger в json форматі. По кожному роуту, була описана наступна інформація :

- Назва та опис роуту
- Шлях роуту

- Обовязкові параметри та їх тип
- Можливі відповіді та їх тип

Окрім опису роутів, були додатково описані і моделі серверної частини застосунку.

```
//docs routes
app.use(`/${PROJECT_NAME}/docs`,swaggerUI.serve,swaggerUI.setup(swaggerOutput))
```

Рис. 4.11 Приклад підключення документації


```
"swagger": "2.0",
"info": {
  "version": "1.0.0",
  "title": "Adoptly",
  "description": "Easy platform to adopt a furry friends 😸"
},
"host": "localhost:3000",
"basePath": "/adoptly",
"schemes": ["http"],
"tags": [···
],
"paths": {
  "/auth/register": {···
  },
  "/auth/login": {···
  },
  "/auth/logout": {
    "get": {
      "tags": ["auth"],
      "summary": "Log out of your account",
      "description": "Erases access_token from cookies",
      "produces": ["application/json"],
      "responses": {
        "200": {
          "description": "Logout success"
        }
      }
    }
  },
```


Рис. 4.12 Приклад написаної документації в форматі json

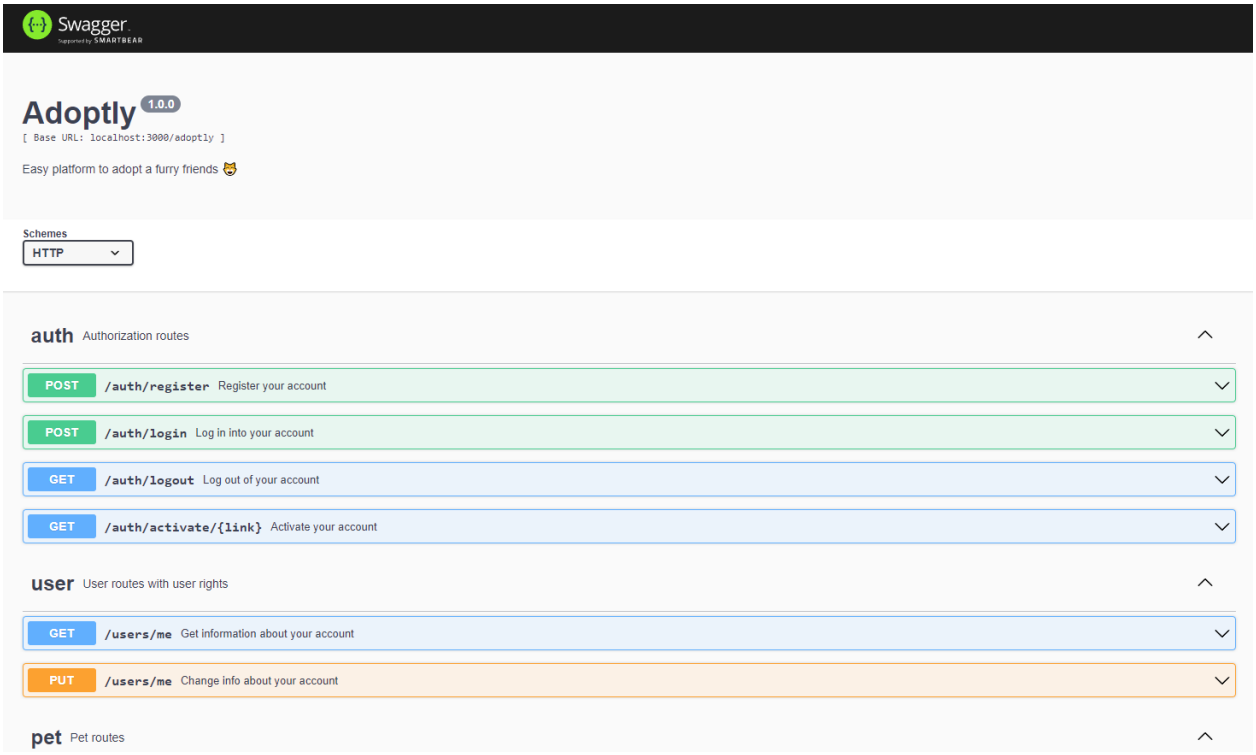


Рис. 4.13 Приклад згенерованої сторінки з документацією

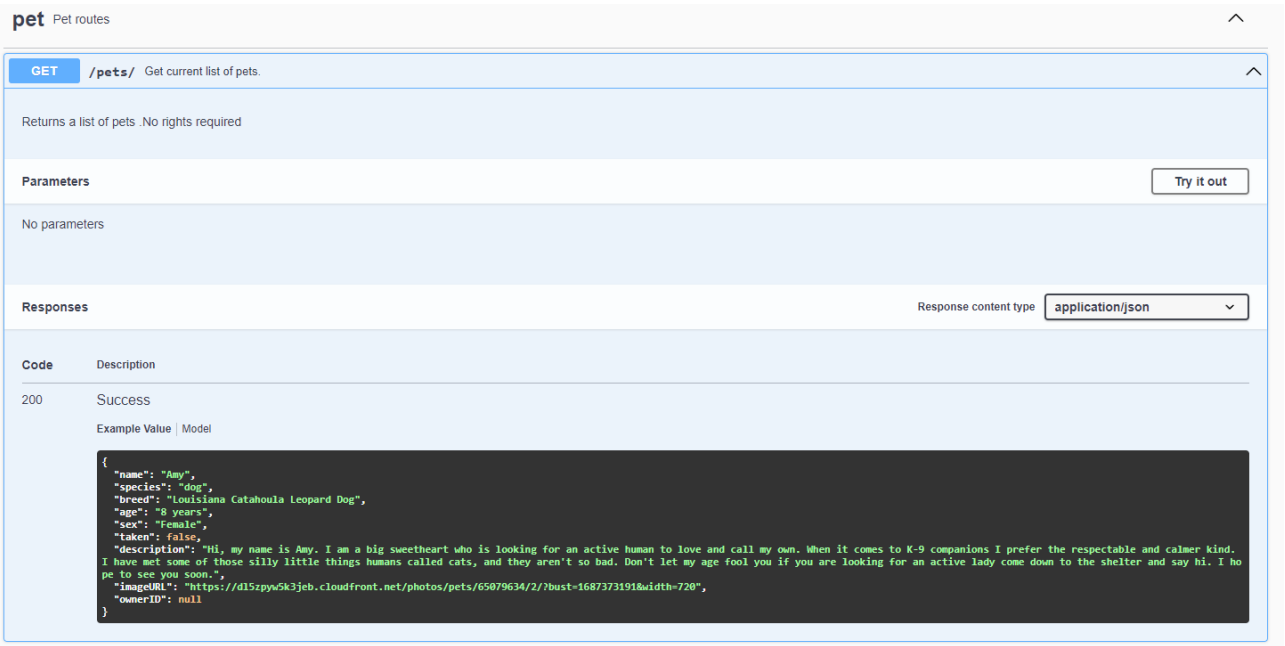


Рис. 4.14 Приклад описаного роуту

4.3 Реалізація інтерфейсу клієнтської частини

Оскільки в розробці front-end частини використовувалась бібліотека react – безпосередня розробка сторінок почалась зі створення react компонентів. В самих компонентах та сторінках багато використовувалась інша бібліотека – MUI, завдяки якій компоненти вдалося стилізувати в основних кольорах застосунку. Також з цієї бібліотеки було імпортовано іконки, які в подальшому були використані в різних компонентах.

```
function Header(props) {
  const navigate = useNavigate();
  function onExitClick(e)···
  }

  console.log(props.userId)
  return (
    <AppBar
      variant="outlined"
      position="fixed"
      elevation={0}
      id='header'
      sx={{
        borderRadius: "4vw",
        maxWidth: "60vw",
        minWidth: "50vw",
        top: "2vh",
        left: "19.6vw",
      }}
    >
      <Container fixed>
        <Toolbar sx={{ display: "flex", justifyContent: "space-between" }}>
          <Box sx={{ display: "flex", flexWrap: "wrap" }}>···
          </Box>
          <Box>···
          </Box>
        </Toolbar>
      </Container>
    </AppBar>
  )
}

export default Header
```

Рис. 4.15 Приклад структури компоненту Header

Рис. 4.16 Приклад відображення компоненту Header для неавторизованих користувачів

Рис. 4.17 Приклад відображення компоненту Header для авторизованих користувачів

Під час розробки клієнтської частини застосунку, активно використовувалась особливість розширення JSX, а саме – його універсальність. В одному файлі, можна використовувати, наприклад, теги з мови розмітки HTML, стилі з CSS та логіку дії і інтерактивність мови програмування JavaScript.

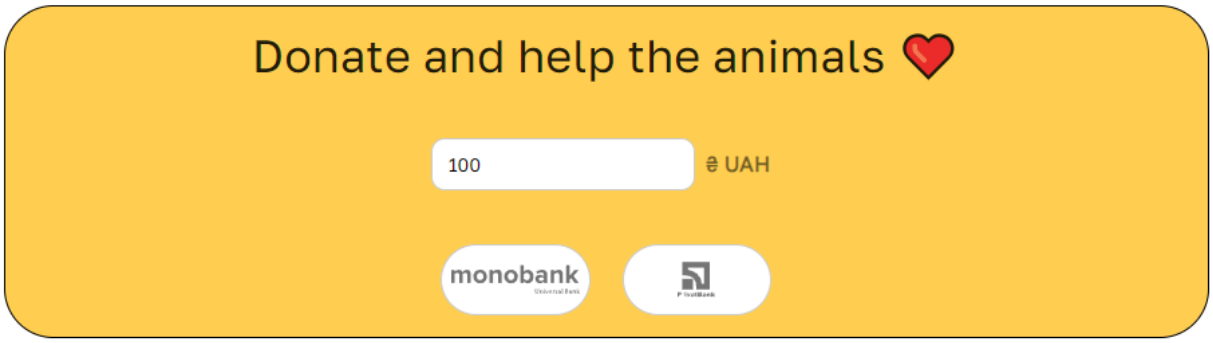


Рис. 4.18 Приклад відображення компоненту Donate

Авторизаційний токен користувача, після його отримання з API, зберігається у locastorage, на стороні браузера. При переході між сторінками, які є приватними, токен парситься і валідується, після цього користувачу надається доступ до контенту.

Завдяки цьому, неавторизований користувач не зможе отримати доступ до таких приватних сторінок, наприклад, як : особистий кабінет, розділ favorites.

```
function PrivateRoutes({onIdChange}) {

    const token = localStorage.getItem("access_token")
    if (token) {
        const decodedJwt = parseJwt(token);

        if (decodedJwt.exp * 1000 < Date.now()){
            localStorage.removeItem('access_token');
        }
        else{

            onIdChange(decodedJwt.id);
        }

    }
    return <Outlet />
}
```

Рис. 4.19 Приклад функції для парсингу та валідації токена

При реєстрації, від користувача достатньо всього лише ввести інформацію про його ім’я, email та пароль. Для авторизації – достатньо всього лише email та паролю. Додаткова інформація, така як : адреса, номер телефону може в подальшому бути додана в особистому кабінеті користувача.

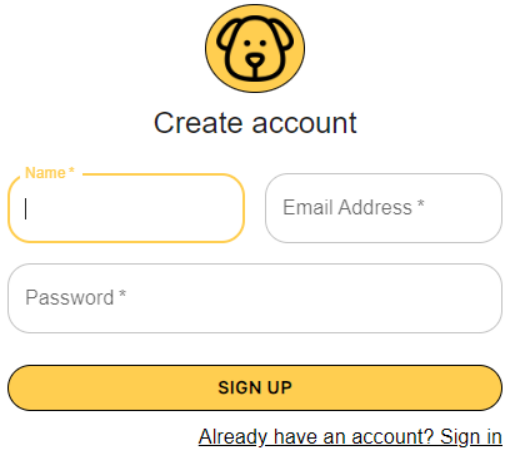


Рис. 4.20 Приклад сторінки реєстрації користувача

4.4 Тестування застосунку

Тестування застосунку – це оцінка програмного забезпечення для визначення його якості, відповідності вимогам і виявлення можливих дефектів. Воно може бути комплексним і складатись з декількох етапів, включати різні методики та типи тестування для покриття якумога більше випадків.

Для тестування front-end частини використовувався мануальний пошук багів, були протестовані усі компоненти та сторінки, їх можливі звязки між собою. Для серверної частини використовувались бібліотеки : jest, supertest, faker, завдяки яким, були покриті end-to-end тестами кожний роут.

```
test('should return a 201', async () => {
  const res = await supertest(app)
    .post(`/adoptly/auth/register`)
    .send(newUser)
    .expect('Content-Type', /json/)
    .expect(201);
  const expectedProperties = ['id','name', 'email', 'phoneNumber', 'password', 'rights', 'address', 'isActivated','activationLink','favorites'];
  for (const property of expectedProperties) {
    expect(res.body).toHaveProperty(property);
  }
  expect({
    name:res.body.name,
    email:res.body.email,
    phoneNumber:res.body.phoneNumber,
    password:res.body.password,
    rights:res.body.rights,
    address:res.body.address,
  }).toMatchObject({...newUser,password:res.body.password});
  expect(res.body.isActivated).toBeFalsy();
  expect(res.body.rights).toBe("write");
  expect(res.body.favorites).toStrictEqual([])
  expect(res.body.activationLink).toBeDefined()
});
```

Рис. 4.21 Приклад написаного end-to-end тесту

```
module.exports = {
    testEnvironment: 'node',
    testEnvironmentOptions: {
      NODE_ENV: 'test',
    }
  };
```

Рис. 4.22 Приклад конфігурації jest файлу

Тестування проводились на тестовій базі даних, оскільки небезпечно використовувати основну базу даних для таких дій. Сервер підключається до тестової бази, у випадку, якщо середовище серверу є тестовим, за це відповідає окрема змінна, яка зберігається в .env файлі. Після кожного тесту колекції очищаються, після усіх тестувань – тестова база від'єднується.

```
const setupTestDB = () => {
  beforeAll(async () => {
    let mongoDbUrl = process.env.MONGODB_URL;
    mongoDbUrl += process.env.NODE_ENV=='test'?'-test':'';
    await mongoose.connect(mongoDbUrl);
  });

  beforeEach(async () => {
    await Promise.all(Object.values(mongoose.connection.collections).map(async (collection) => collection.deleteMany()));
  });

  afterAll(async () => {
    await mongoose.disconnect();
  });
};
```

Рис. 4.23 Приклад методу для очистки бази даних після тестувань

```
let newUser;
beforeEach(async() => {
  newUser = {
    name: faker.person.firstName(),
    email:faker.internet.email(),
    phoneNumber:faker.phone.number(),
    password:faker.internet.password(),
    rights:'write',
    address:faker.location.streetAddress(),
  };
  userResponse= await supertest(app)
  .post(`/adoptly/auth/register`)
  .send(newUser)
  .expect('Content-Type', /json/)
  .expect(201);
  });
```

Рис. 4.24 Приклад створення фейкових даних після кожного тесту

```
PASS tests/auth.test.js (20.631 s)
  Authorization routes
    Register /adoptly/auth/register
      √ should return a 201 (2791 ms)
      √ should return a 400 if some of the required fields are missing (166 ms)
      √ should return a 400 if phoneNumber or email value is not unique (3161 ms)
      √ should return a 400 if rights field is not enum value (1417 ms)
    Login /adoptly/auth/login
      √ should return a 200 (1805 ms)
      √ should return a 400 if some of the required fields are missing (1617 ms)
      √ should return a 401 if password was not correct (1813 ms)
      √ should return a 404 if user was not found by email (1762 ms)
    Logout /adoptly/auth/logout
      √ should return a 200 (141 ms)
    Activate /adoptly/auth/:activationLink
      √ should return a 302 (1877 ms)
      √ should return a 404 if user was not found by activationLink (1856 ms)
```

Рис. 4.25 Приклад успішного завершення ряду тестів

За результатами тестування, було виявлено багато випадків некоректної поведінки модулів та компонентів на сторонах front-end та back-end. В подальшому, кожний знайдений баг був опрацьований і виправлений

ВИСНОВКИ

У результаті виконання дипломної роботи було розроблено веб-сервіс, що складається з серверної та клієнтської частини, для адопції тварин. Актуальність роботи полягає в тому, що наразі процес адопції тварин для більшості користувачів є мануальним.

1. Здійснено аналіз предметної області веб-застосунків для адопції тварин. В результаті аналізу виявлені основні вимоги користувачів.

2. Досліджені існуючи програмні засоби. Під час дослідження виявлені головні переваги та недоліки аналогів веб-застосунків для адопції тварин.

3. Визначено актуальні бібліотеки та фреймворки для створення застосунку. В результаті дослідження, були вибрані наступні програмні засоби реалізації: мова програмування Javascript, платформа для розробки front-end та back-end застосунків Node js, фреймворк для серверної частини Express js, бібліотека для клієнтської частини React,бібліотека UI компонентів MUI, нереляційна база даних Mongo DB, редактор коду Visual Studio Code та інші інструменти.

4.Створено детальний проект системи, включаючи архітектуру, дизайн інтерфейсу та моделі даних і розроблено відповідне програмне забезпечення.

5. Проведено тестування серверної частини додатку end-to-end методом, клієнтської частини – мануальним способом. Усі знайдені випадки некоректної поведінки в модулях та компонентах були виправлені.

6. Написано документацію для серверної частини застосунку.

ПЕРЕЛІК ПОСИЛАНЬ

1. V8 документація [Електронний ресурс] – Режим доступу до ресурсу : https://v8.dev/docs
2. Node js документація [Електронний ресурс] – Режим доступу до ресурсу : https://nodejs.org/en
3. Libuv документація [Електронний ресурс] – Режим доступу до ресурсу : https://docs.libuv.org/en/v1.x/
4. React документація [Електронний ресурс] – Режим доступу до ресурсу : https://react.dev/
5. MUI документація [Електронний ресурс] – Режим доступу до ресурсу : https://mui.com/
6. Jest документація [Електронний ресурс] – Режим доступу до ресурсу : https://jestjs.io/uk/
7. Nodemailer документація [Електронний ресурс] – Режим доступу до ресурсу : https://www.nodemailer.com/
8. JWT сертифікація і документація [Електронний ресурс] – Режим доступу до ресурсу : https://jwt.io/
9. Visual Studio Code документація [Електронний ресурс] – Режим доступу до ресурсу : https://code.visualstudio.com/docs
10. Postman документація [Електронний ресурс] – Режим доступу до ресурсу : https://www.postman.com/product/what-is-postman/
11. Express js документація [Електронний ресурс] – Режим доступу до ресурсу : https://expressjs.com/en/guide/routing.html
12. Mongo DB документація [Електронний ресурс] – Режим доступу до ресурсу : https://www.mongodb.com/
13. Swagger документація [Електронний ресурс] – Режим доступу до ресурсу : https://swagger.io/

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ
(Презентація)

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка веб-застосунку для адопції тварин з притулку з використанням технологій Node js, React

Виконав студент 4 курсу
Групи ПД-41
Пилипчук Ігор Русланович
Керівник роботи
К.т.н., доц., доцент кафедри ІПЗ Довженко Тимур Павлович

Київ – 2024

АНАЛІЗ АНАЛОГІВ

Показник	Happy paw	“ЛКП” Лев	Animal-id	Adoptly
Наявність відкритої API документації	-	-	-	+
Можливість реєстрації та авторизації користувачів	+	-	+	+
Можливість додати тварину у розділ «Favorites»	+	-	+	+
Можливість надіслати донат на допомогу тварин	+	+	+	+
Можливість вибору випадкової тварини	-	-	-	+

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги :

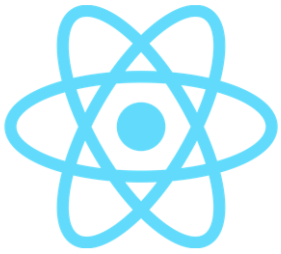
1. Реєстрація та авторизація користувачів. Користувачі можуть створювати аккаунти, активувати їх через електронну пошту.
2. Перегляд тварин. Користувачі можуть переглядати список тварин, які доступні для адопції. 3. Пошук тварин. Користувачі можуть шукати тварин за різними критеріями, такими як вид тварини, вік, стать тощо.
4. Адопція тварин. Авторизовані користувачі можуть подати запит на адопцію тварини, яку пізніше розгляне адміністратор.
5. Права адміністратора. Адміністратор може додавати, видаляти та редагувати інформацію про тварин та користувачів, модерувати заявки на адопції від користувачів.
6. Донат. Користувачі можуть надсилати донати на допомогу тваринам.
7. Випадковий пошук. Користувачі можуть випадковим чином вибрати тварину та отримати детальну інформацію про неї.
8. Документація. Користувачі можуть переглядати документацію для серверної частини.

Не функціональні вимоги :

1. Адаптивність. Веб додаток повинен бути адаптивним до різних пристроїв та екранів.
2. Стійкість. Система повинна бути стійкою до відмов, незважаючи на перепад трафіку від користувачів.
3. Безпека. Чутливі данні, такі як, паролі, мають зберігатись у базі даних в зашифрованому вигляді.
4. Тестування. Всі роути серверної частини повинні бути прокриті end-to-end тестами. Клієнтська частина повинна бути протестована мануально.

3

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

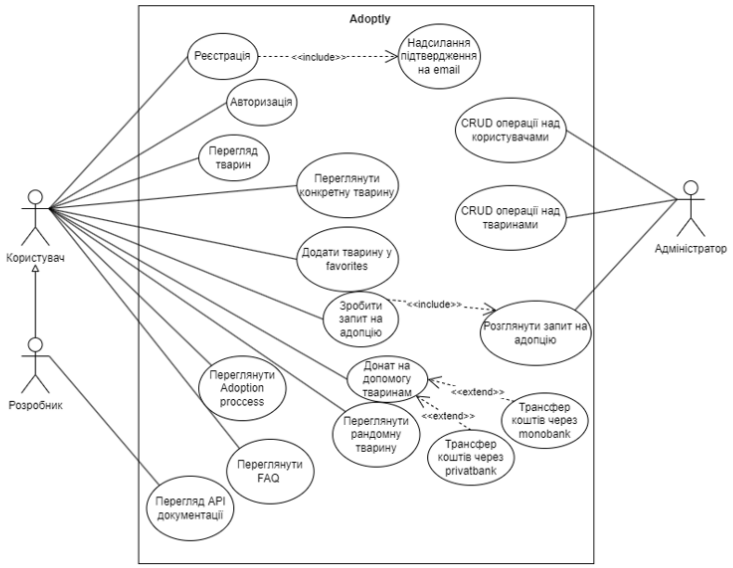


React

MUI

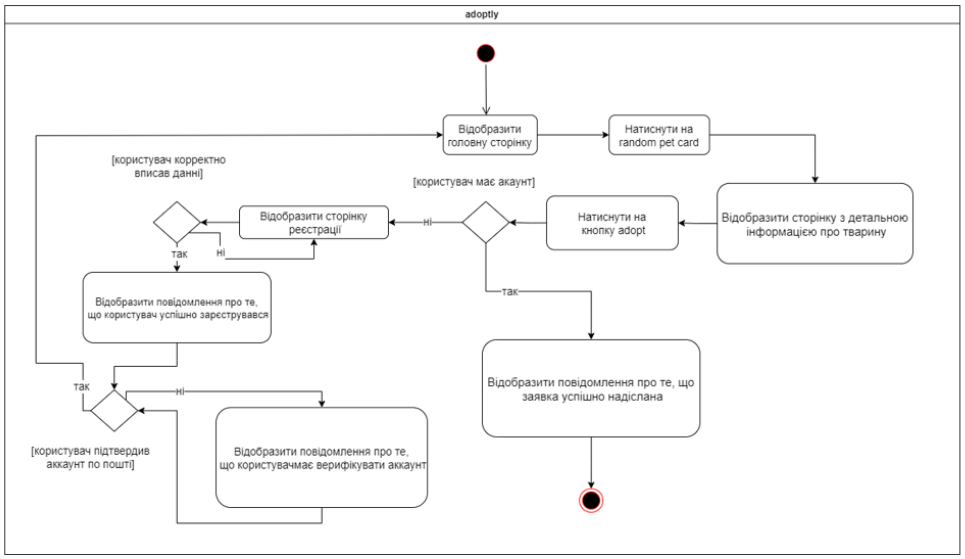
4

Діаграма варіантів використання



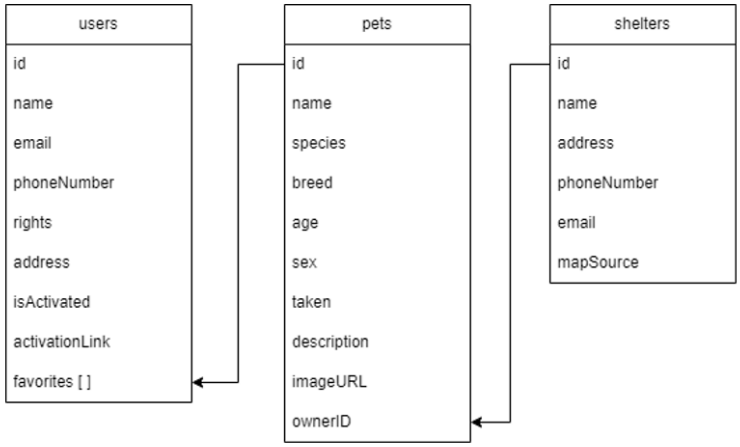
5

Діаграма діяльності : процес адопції випадкової тварини



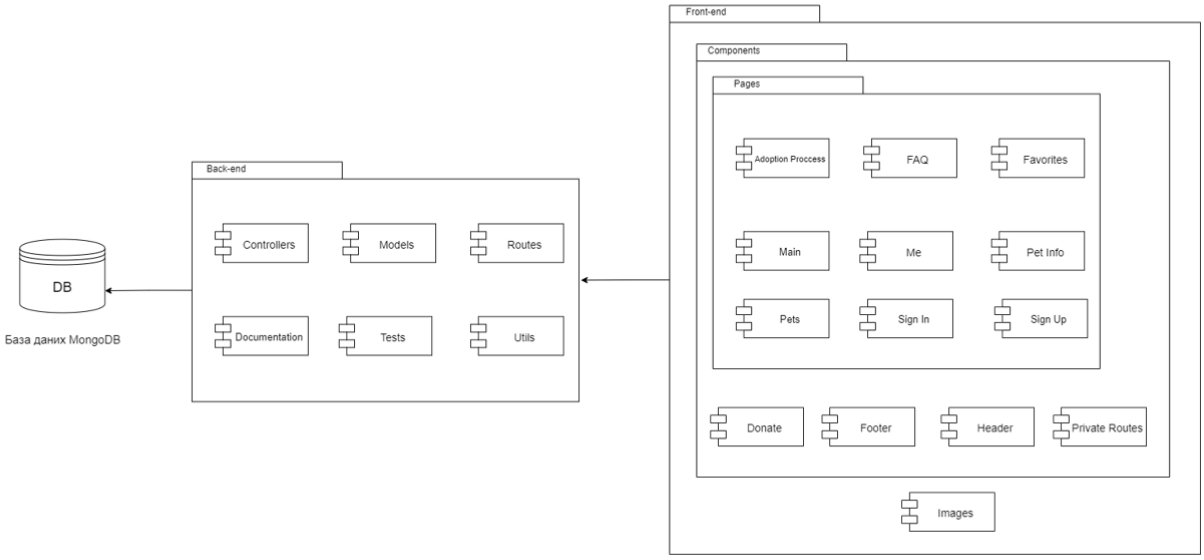
6

Схема бази даних



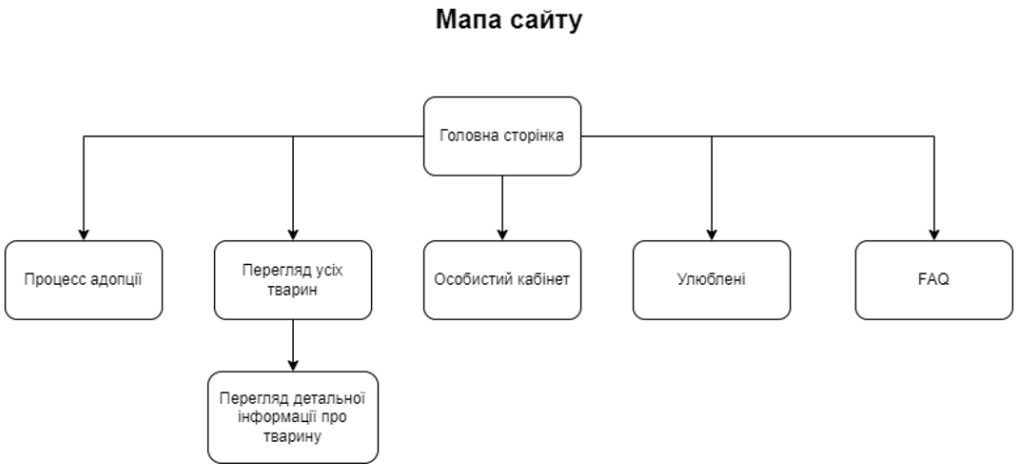
7

Архітектура застосунку



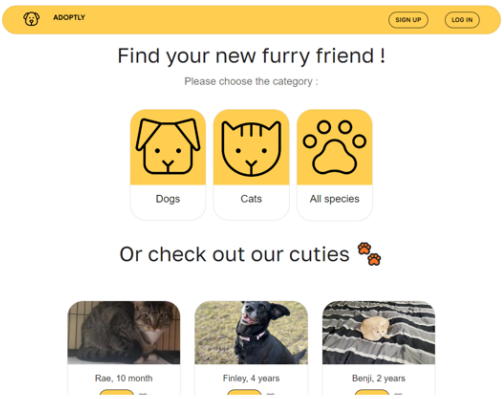
8

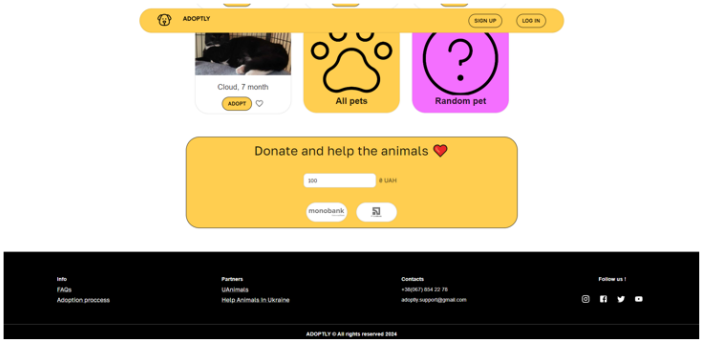
Мапа сайту



9

ЕКРАННІ ФОРМИ

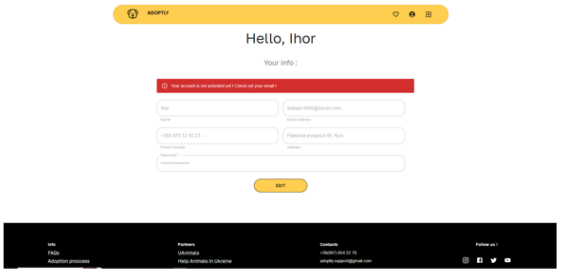




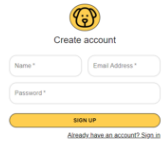
Головна сторінка для неавторизованого користувача

10

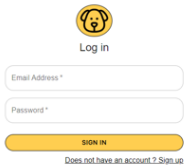
ЕКРАННІ ФОРМИ



Особистий акаунт не верифікованого користувача



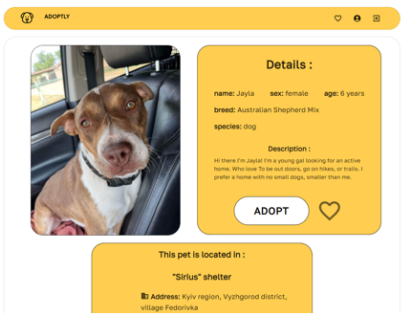
Сторінка створення акаунту

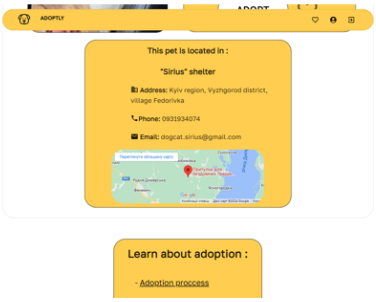


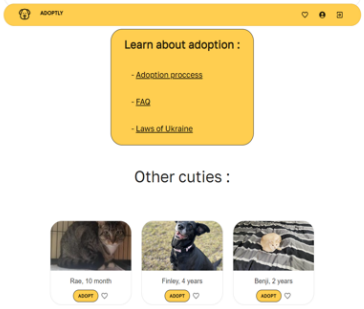
Сторінка авторизації акаунту

11

ЕКРАННІ ФОРМИ



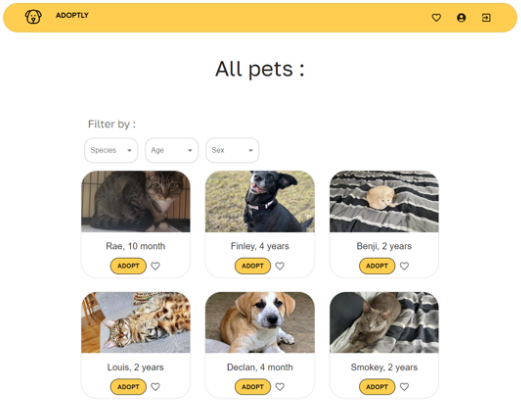


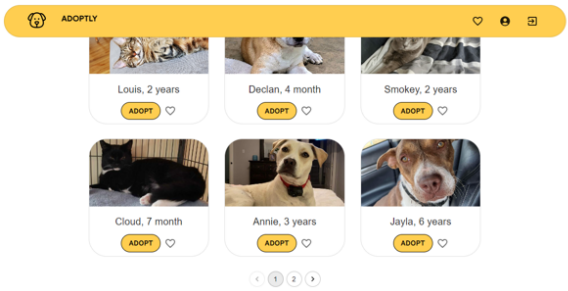


Сторінка тварини у авторизованого користувача

12

ЕКРАННІ ФОРМИ

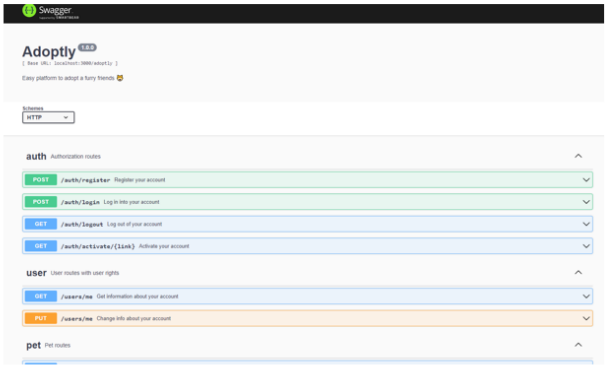


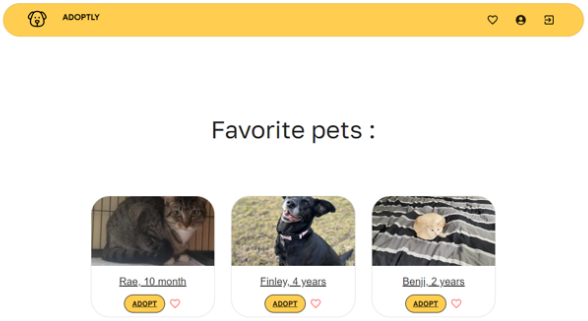


Сторінка усіх тварин у авторизованого користувача

13

ЕКРАННІ ФОРМИ





Сторінка зі swagger документацією back-end частини застосунку

Сторінка "favorites" у авторизованого користувача

14

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Пилипчук І.Р., Довженко Т.П. Визначення вимог до веб-додатку adoptly онлайн притулок тварин: Матеріали Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в ІКТ». Збірник тез. 24.04.2024, ДУІКТ, м. Київ. К.: ДУІКТ – С. 57.
2. Пилипчук І.Р., Довженко Т.П. Визначення додаткових функцій до веб-додатку adoptly онлайн притулок тварин: Матеріали Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в ІКТ». Збірник тез. 24.04.2024, ДУІКТ, м. Київ. К.: ДУІКТ – С. 58.

ДЯКУЮ ЗА УВАГУ!

ДОДАТОК Б
ОСНОВНІ БЛОКИ КОДУ

Pet controller:

```
const petModel = require('../models/index').Pet;
const userModel = require('../models/index').User;
const apiError = require('../utils/apiError');

const getPets =async(req,res,next)=>{
try{
    const pets = await petModel.find({}).select('name species taken breed age sex description
imageURL');
    return res.status(200).json(pets);
}
catch(err)
{
    return next(err);
}
};
const getPet = async(req,res,next)=>{
    try{
        const pet = await petModel.findById(req.params.id).exec();
        if (!pet) {
                throw new apiError(404, 'Pet was not found');
          }
          return  res.status(200).json(pet);
    }
    catch(err)
    {
        return next(err);
    }
};
const postPet = async(req,res,next)=>{
    try{
        const newPet= new petModel({
            name:req.body.name,
            species:req.body.species,
            breed:req.body.breed,
            age:req.body.age,
            sex:req.body.sex,
            taken:req.body.taken,
            description:req.body.description,
            imageURL:req.body.imageURL,
            ownerID:req.body.ownerID,
        });
        const savedPet= await newPet.save();
        return res.status(201).json(savedPet);
    }
    catch(err)
    {
        return next(err);
    }
};
const getTakenPets = async(req,res,next)=>{
    try{
        const pets = await petModel.find({ taken: true });
```

```
        return res.status(200).json(pets);

    }
    catch(err)
    {
        return next(err);
    }
};

const putPet = async(req,res,next)=>{
    try{
        if(req.user.rights == "read"&&req.user.isActivated==false)
        {
            throw new apiError(403, "Activate your account");
        }

        const pet =await petModel.findById(req.params.id).exec();
        if(!pet)
        {
            throw new apiError(404, 'Pet was not found');
        }
        const updatedPet = await petModel.findByIdAndUpdate(req.params.id,{
            name:req.body.name,
            species:req.body.species,
            breed:req.body.breed,
            age:req.body.age,
            sex:req.body.sex,
            taken:req.body.taken,
            description:req.body.description,
            imageURL:req.body.imageURL,
            ownerID:req.body.ownerID,
        },{new:true,
            //runValidators: true
        });
        return res.status(200).json(updatedPet);
    }
    catch(err)
    {
        return next(err);
    }
};
const deletePet = async(req,res,next)=>{
    try{
        const pet =await petModel.findById(req.params.id).exec();
        if(!pet)
        {
            throw new apiError(404, 'Pet was not found');
        }
        await petModel.findByIdAndDelete(req.params.id);
        return res.status(204).json("Pet deleted");
    }
    catch(err)
    {
        return next(err);
    }
};
module.exports={
```

```
    getPets,
    getPet,
    getTakenPets,
    postPet,
    putPet,
    deletePet
}
```

Pet model:

```
const mongoose = require('mongoose');
const toJSON = require('../utils/toJson');
var Schema = mongoose.Schema;

var petSchema = new Schema({
  name: {
    type: String,
    required: [true, 'Name of the pet is required'],
  },
  species:{
    type: String,
    required: [true, 'Species of the pet is required']
  },
  age:{
    type: Number,
    required: [true, 'Age of the pet is required'],
    min:0,
    max:50
  },
  sex:{
    type: String,
    required: [true, 'Sex of the pet is required'],
    enum: ['male', 'female']
  },
  breed:{
    type: String
  },
  taken:{
    type: Boolean,
    default:false
  },
  description: {
    type: String,
    default:"",
  },
  imageURL:{
    type: String,
    default:"https://as1.ftcdn.net/v2/jpg/02/52/84/92/1000_F_252849218_Acdc6N696mDekuQvrCmqlFDMx4UYYF2Y.jpg"
  },
  ownerID:{
    type: mongoose.Schema.Types.ObjectId,
    default:null
  },
});

toJSON(petSchema);
```

```
module.exports = mongoose.model('pets', petSchema);
```

Authorization tests:

```
const supertest = require('supertest');
const { faker } = require('@faker-js/faker');
const app = require('../src/app').app;
const setupTestDB = require('../src/utils/setupTestDB');

setupTestDB();

describe('Authorization routes', () => {
describe('Register /adoptly/auth/register', () => {
            let newUser;
            beforeEach(() => {
              newUser = {
                name: faker.person.firstName(),
                email:faker.internet.email(),
                phoneNumber:faker.phone.number(),
                password:faker.internet.password(),
                rights:'write',
                address:faker.location.streetAddress(),
              };
            });
            test('should return a 201', async () => {
              const res = await supertest(app)
                .post(`/adoptly/auth/register`)
                .send(newUser)
                .expect('Content-Type', /json/)
                .expect(201);
                 const expectedProperties = ['id','name', 'email', 'phoneNumber', 'password',
'rights', 'address', 'isActivated','activationLink','favorites'];
              for (const property of expectedProperties) {
                expect(res.body).toHaveProperty(property);
              }
              expect({
                name:res.body.name,
                email:res.body.email,
                phoneNumber:res.body.phoneNumber,
                password:res.body.password,
                rights:res.body.rights,
                address:res.body.address,
              }).toMatchObject({...newUser,password:res.body.password});
              expect(res.body.isActivated).toBeFalsy();
              expect(res.body.rights).toBe("write");
              expect(res.body.favorites).toStrictEqual([])
              expect(res.body.activationLink).toBeDefined()
            });
            test('should return a 400 if some of the required fields are missing', async () =>
{
                await supertest(app)
                   .post(`/adoptly/auth/register`)
                   .send({
                     name:newUser.name,
                     favorites:newUser.favorites,
                     address:newUser.address,
                   })
                   .expect(400);
```

```
            });
            test('should return a 400 if phoneNumber or email value is not unique', async ()
=> {
            await supertest(app)
                .post(`/adoptly/auth/register`)
                .send(newUser)
                .expect(201);
                await supertest(app)
                .post(`/adoptly/auth/register`)
                .send(newUser)
                .expect(400);
            });
            test('should return a 400 if rights field is not enum value', async () => {
            await supertest(app)
                .post(`/adoptly/auth/register`)
                .send({
                    ...newUser,
                    rights:'wrongRights',
                })
                .expect(400);
            });
    });
describe('Login /adoptly/auth/login', () => {
        let newUser;
        beforeEach(async() => {
          newUser = {
            name: faker.person.firstName(),
            email:faker.internet.email(),
            phoneNumber:faker.phone.number(),
            password:faker.internet.password(),
            rights:'write',
            address:faker.location.streetAddress(),
          };
          userResponse= await supertest(app)
          .post(`/adoptly/auth/register`)
          .send(newUser)
          .expect('Content-Type', /json/)
          .expect(201);
          });
        test('should return a 200', async () => {
          const res = await supertest(app)
            .post(`/adoptly/auth/login`)
            .send({email:newUser.email,password:newUser.password})
            .expect('Content-Type', /json/)
            .expect(200);
            expect(Object.keys(res.header)).toContain('set-cookie');
            expect(res.header['set-cookie'][0].split('=')[0]).toBe('access_token')
        });
        test('should return a 400 if some of the required fields are missing', async () => {
            await supertest(app)
               .post(`/adoptly/auth/login`)
               .send({email:newUser.email})
               .expect(400);
        });
        test('should return a 401 if password was not correct', async () => {
            await supertest(app)
               .post(`/adoptly/auth/login`)
```

```
                .send({email:newUser.email,password:newUser.password+'random'})
                .expect(401);
        });
        test('should return a 404 if user was not found by email', async () => {
            await supertest(app)
                .post(`/adoptly/auth/login`)
                .send({email:newUser.email+'random',password:newUser.password})
                .expect(404);
        });
});
describe('Logout /adoptly/auth/logout', () => {
    test('should return a 200', async () => {
      const res = await supertest(app)
        .get(`/adoptly/auth/logout`)
        .expect('Content-Type', /json/)
        .expect(200);
        expect(Object.keys(res.header)).toContain('set-cookie');
        expect(res.header['set-cookie'][0].split('access_token=')[1][0]).toBe(';')
    });

});
describe('Activate /adoptly/auth/:activationLink', () => {
    let newUser;
        beforeEach(async() => {
          newUser = {
            name: faker.person.firstName(),
            email:faker.internet.email(),
            phoneNumber:faker.phone.number(),
            password:faker.internet.password(),
            rights:'write',
            address:faker.location.streetAddress(),
          };
            userResponse= await supertest(app)
            .post(`/adoptly/auth/register`)
            .send(newUser)
            .expect('Content-Type', /json/)
            .expect(201);
            });
    test('should return a 302', async () => {
         await supertest(app)
        .get(`/adoptly/auth/activate/${userResponse.body.activationLink}`)
        .expect('Content-Type','text/plain; charset=utf-8')
        .expect(302);
    });
    test('should return a 404 if user was not found by activationLink', async () => {
          await supertest(app)
          .get(`/adoptly/auth/activate/${userResponse.body.activationLink}123`)
          .expect('Content-Type', /json/)
          .expect(404);
      });

});
});
```

Me page:

```
import {
    Box,
```

```
    Container,
    Typography,
    Stack,
    CardMedia,
    Card,
    CardContent,
    Grid,
    TextField,
    Link,
    Alert,
  } from "@mui/material";
  import React, { useEffect, useState } from "react";
  import { createTheme, ThemeProvider } from "@mui/material/styles";
  import Accordion from "@mui/material/Accordion";
  import AccordionActions from "@mui/material/AccordionActions";
  import AccordionSummary from "@mui/material/AccordionSummary";
  import AccordionDetails from "@mui/material/AccordionDetails";
  import ExpandMoreIcon from "@mui/icons-material/ExpandMore";
  import Button from "@mui/material/Button";
  import "../../App.css";

  import Header from "../Header";

  import Footer from "../Footer";
import axios from "axios";
import { useNavigate } from "react-router-dom";

  const theme = createTheme({
    palette: {
      primary: {
        main: "#ffce50",
      },

    },
  });

  function Me(props) {
    const navigate = useNavigate();
    const [user,setUser] = useState({})
    useEffect(() => {
        (
          async () => {
            try {
                if(!localStorage.getItem('access_token')){
                    navigate('/signin');
                }
                 let authUserData = await axios.get('http://localhost:3000/adoptly/users/me',
{
                   headers: { Authorization: `Bearer ${localStorage.getItem('access_token')}`
}
                 });
             setUser(authUserData.data);
             console.log(user)
             console.log(authUserData.data)
            } catch (err) {
              console.log(err);
            }
```

```
        }
      )();
    }, []);
  return (
    <ThemeProvider theme={theme}>
      <Header userId={props.userId}/>
      <main>
        <Container sx={{ pb: 12, pt: 13 }} maxWidth="md">
          <Typography
            component="h2"
            variant="h3"
            align="center"
            color="text.primary"
            fontFamily="Golos Text"
            gutterBottom
            sx={{ mb: 5 }}
          >
            Hello, {user.name}
          </Typography>
          <Typography
            component="h5"
            variant="h5"
            align="center"
            color="text.secondary"
            fontFamily="Golos Text"
            gutterBottom
            sx={{ mb: 5 }}
          >
           Your info :
          </Typography>
          {user.isActivated?(<></>):(<Alert

    severity="error"
    variant="filled"
    sx={{ width: '100%',height:'100%' }}
    >
Your account is not activated yet ! Check out your email !
  </Alert>)}
          <Box component="form" noValidate  sx={{ mt: 3 }}>
          <Grid container spacing={2}>
            <Grid item xs={12} sm={6}>
              <TextField
                autoComplete="name"
                name="name"
                disabled
                required
                value={user.name}
                helperText='Name'
                fullWidth
                id="name"
                autoFocus
                InputProps={{
                  style: {
                    borderRadius: "2vh",
                  }
                }}
              />
```

```
</Grid>
<Grid item xs={12} sm={6}>
<TextField
    required
    fullWidth
    id="email"
    helperText="Email Address"
    name="email"
    value={user.email}
    disabled
    autoComplete="email"
    InputProps={{
      style: {
        borderRadius: "2vh",
      }
    }}
  />
</Grid>
<Grid item xs={12} sm={6}>
<TextField
    required
    fullWidth
    id="phone"
    helperText="Phone Number"
    value={user.phoneNumber}
    name="phone"
    disabled
    autoComplete="phone"
    InputProps={{
      style: {
        borderRadius: "2vh",
      }
    }}
  />
</Grid>
<Grid item xs={12} sm={6}>
<TextField
    required
    fullWidth
    disabled
    id="address"
    helperText="Address"
    name="address"
    value={user.address}
    autoComplete="address"
    InputProps={{
      style: {
        borderRadius: "2vh",
      }
    }}
  />
</Grid>
<Grid item xs={12}>
  <TextField
    required
    fullWidth
    name="password"
```

```
                value='****************'
                disabled
                label="Password"
                type="password"
                id="password"
                autoComplete="new-password"
                InputProps={{
                  style: {
                    borderRadius: "2vh",
                  }
                }}
              />
            </Grid>
            <Grid item xs={12}>

            </Grid>
          </Grid>
          <Box sx={{
             display:'flex',
             alignItems:"center",
             alignContent:'center',
             justifyContent:'center'
          }}>
          <Button
            type="submit"
            variant="outlined"
            sx={{
              borderRadius: "4vw",
              borderColor: "black",
              color: "black",
              fontFamily: "Golos Text",
              fontWeight: "bold",
              backgroundColor:'#ffce50',
              my:'1vh',
              height:"5vh",
              width:'20vh',
              ':hover':{
                borderColor:'white',
                backgroundColor:'#ffce50',
              }
             }}
          >
            Edit
          </Button>
          </Box>

        </Box>
        </Container>
      </main>
      <footer>
        <Footer />
      </footer>
    </ThemeProvider>
  );
}

export default Me;
```

Footer component :

```
import Twitter from '@mui/icons-material/Twitter'
import { Box, IconButton, Link, Stack, Typography } from '@mui/material'
import FacebookIcon from "@mui/icons-material/Facebook";
import InputAdornment from "@mui/material/InputAdornment";

import InstagramIcon from "@mui/icons-material/Instagram";
import TwitterIcon from "@mui/icons-material/Twitter";
import YouTubeIcon from "@mui/icons-material/YouTube";
import LinkedInIcon from "@mui/icons-material/LinkedIn";
import React from 'react'

function Footer() {
  return (
    <Box
    sx={{
      backgroundColor: "black",
      margin: "0",

      display: "flex",
      flexDirection: "column",
      alignItems: "space-around",
      gap: { xs: 2, sm: 6 },
      pt: { xs: 6, sm: 8 },
      textAlign: { sm: "center", md: "left" },
    }}
  >
    <Box
      sx={{
        display: "flex",
        flexDirection: { xs: "column", sm: "row" },
        width: "100%",
        justifyContent: "space-around",
        fontFamily: "Golos Text",
      }}
    >
      <Box
        sx={{
          display: { xs: "none", sm: "flex" },
          flexDirection: "column",
          color: "#ffffff",
          gap: 1,
        }}
      >
        <Typography variant="body2" fontWeight={600}>
          Info
        </Typography>
        <Link color="#ffffff" href="/faq">
          FAQs
        </Link>
        <Link color="#ffffff" href="/adoption-proccess">
          Adoption proccess
        </Link>
      </Box>
      <Box
        sx={{
```

```
    display: { xs: "none", sm: "flex" },
    flexDirection: "column",
    color: "#ffffff",
    gap: 1,
  }}
>
  <Typography variant="body2" fontWeight={600}>
    Partners
  </Typography>
  <Link color="#ffffff" href="https://uanimals.org/">
    UAnimals
  </Link>
  <Link
    color="#ffffff"
    href="https://www.helpanimalsinukraine.com/etusivu"
  >
    Help Animals In Ukraine
  </Link>
</Box>
<Box
  sx={{
    display: { xs: "none", sm: "flex" },
    flexDirection: "column",
    color: "#ffffff",
    gap: 1,
  }}
>
  <Typography variant="body2" fontWeight={600}>
    Contacts
  </Typography>
  <Typography color="#ffffff" variant="body2">
    +38(067) 854 22 78
  </Typography>
  <Typography color="#ffffff" variant="body2">
    adoptly.support@gmail.com
  </Typography>
</Box>
<Box
  sx={{
    display: "flex",
    flexDirection: "column",
    gap: 3,
    color: "#ffffff",
    justifyContent: "space-between",
  }}
>
  <Typography align="center" variant="body2" fontWeight={600}>
    Follow us !
  </Typography>
  <Stack
    direction="row"
    justifyContent="center"
    spacing={1}
    useFlexGap
    sx={{
      color: "#ffffff",
    }}
```

```
    >
      <IconButton
        color="inherit"
        href="https://www.instagram.com/"
        aria-label="Instagram"
        sx={{
          alignSelf: "center",
          ":hover": {
            color: "#ff70ef",
          },
        }}
      >
        <InstagramIcon />
      </IconButton>
      <IconButton
        color="inherit"
        href="https://www.facebook.com/"
        aria-label="Facebook"
        sx={{
          alignSelf: "center",
          ":hover": {
            color: "#263d9c",
          },
        }}
      >
        <FacebookIcon />
      </IconButton>
      <IconButton
        color="inherit"
        href="https://twitter.com"
        aria-label="X"
        sx={{
          alignSelf: "center",
          ":hover": {
            color: "#35b0ff",
          },
        }}
      >
        <TwitterIcon />
      </IconButton>
      <IconButton
        color="inherit"
        href="https://www.youtube.com/watch?v=dQw4w9WgXcQ"
        aria-label="Youtube"
        sx={{
          alignSelf: "center",
          ":hover": {
            color: "#ff2e2e",
          },
        }}
      >
        <YouTubeIcon />
      </IconButton>
    </Stack>
  </Box>
</Box>
```

```
    <Box
      sx={{
        display: "flex",
        justifyContent: "center",

        p: 1,
        color: "#ffffff",
        borderTop: "1px solid",
        borderColor: "#5e5e5e",
      }}
    >
      <Typography
        fontWeight={"bold"}
        variant="body2"
        color="#ffffff"
        mt={1}
      >
        {` ADOPTLY © All rights reserved  `}
        {new Date().getFullYear()}
      </Typography>
    </Box>
  </Box>
  )
}

export default Footer
```