Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет центр післядипломної освіти

(повна назва)

Кафедра програмної інженерії

(повна назва)

**КВАЛІФІКАЦІЙНА РОБОТА**

**Пояснювальна записка**

рівень вищої освіти перший (бакалаврський)

Веб-застосунок “Petly” для сприяння пошуку власників тварин

через розміщення оголошень.

(тема)

Виконав:

здобувач другого року навчання

групи ПЗПІп-23-1

Кузнєцов Максим

(Власне ім’я, ПРІЗВИЩЕ)

Спеціальність 121 – Інженерія програмного\_ \_\_

забезпечення

(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Програмна інженерія

(повна назва освітньої програми)

Керівник ст.викл. кафедри ПІ Дмитро МАТВЄЄВ

(посада, Власне ім’я, ПРІЗВИЩЕ)

Допускається до захисту

Зав. кафедри \_\_\_\_\_\_\_\_\_\_\_ Кирило СМЕЛЯКОВ

(підпис) (Власне ім’я, ПРІЗВИЩЕ)

2025 р.

Харківський національний університет радіоелектроніки

Факультет ЦПО

Кафедра програмної інженерії

Рівень вищої освіти перший (бакалаврський)

Спеціальність 121 – Інженерія програмного забезпечення

Тип програми Освітньо-професійна

Освітня програма Програмна Інженерія

(шифр і назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри \_\_\_\_\_\_\_\_\_

(підпис)

« » 2025 р.

**ЗАВДАННЯ**

НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Кузнєцову Максиму Олександровичу

(прізвище, ім’я, по батькові)

1. Тема роботи «Веб застосунок “Petly” для сприяння пошуку власників тварин через розміщення оголошень»

Затверджена наказом по університету від 19.05. 2025р. № 386 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 26.06.2025 \_

3. Вихідні дані до роботи Розробити вебзастосунок для взаємодії з оголошеннями щодо домашніх тварин, який дозволяє створювати, редагувати та переглядати публікації, додавати тварин до профілю, зберігати оголошення в обране, з використанням мов програмування Java Script (React, Node.js) та бази даних MongoDB.

4. Перелік питань, що потрібно опрацювати в роботі

Вступ, аналіз предметної галузі, формування вимог до програмної системи, архітектура та проектування програмного забезпечення, опис прийнятих програмних рішень, тестування розробленого програмного забезпечення, висновки, додатки.

**КАЛЕНДАРНИЙ ПЛАН**

|  |  |  |  |
| --- | --- | --- | --- |
| № | Назва етапів роботи | Термін виконання етапів роботи | Примітка |
| 1 | Аналіз предметної галузі | 24.04.2025 | *Виконано* |
| 2 | Створення специфікації ПЗ | 02.05.2025 | *виконано* |
| 3 | Проектування ПЗ | 16.05.2025 | *виконано* |
| 4 | Розробка ПЗ | 30.05.2025 | *виконано* |
| 5 | Тестування ПЗ | 06.06.2025 | *виконано* |
| 6 | Оформлення пояснювальної записки | 13.06.2025 | *виконано* |
| 7 | Підготовка презентації та доповіді | 20.06.2025 | *виконано* |
| 8 | Попередній захист | 21.06.2025 | *виконано* |
| 9 | Нормоконтроль, рецензування | 21.06.2025 | *виконано* |
| 10 | Здача роботи у електронний архів | 23.06.2025 | *виконано* |
| 11 | Допуск до захисту у зав. кафедри | 23.06.2025 | *виконано* |

Дата видачі завдання «07 » «квітня » 2025р.

Здобувач \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Кузнєцов М.О.

(підпис)

Керівник роботи ст.викл. кафедри ПІ Дмитро МАТВЄЄВ

(підпис) (посада, Власне ім’я, ПРІЗВИЩЕ)

**РЕФЕРАТ**

Пояснювальна записка до кваліфікаційної роботи бакалавра: 70 стор., 25 рис., 15 джерел.

ВЕБЗАСТОСУНОК, ДОМАШНІ ТВАРИНИ, JAVASCRIPT, MONGODB, NODE.JS REACT, RENDER, VERSEL

Об’єкт розробки – вебзастосунок для роботи з оголошеннями щодо домашніх тварин.

Мета розробки – створення вебзастосунку, який дозволяє реєструватися, додавати тварин, створювати та переглядати оголошення, додавати записи в обране.

Метод рішення – середовище веброзробки, фреймворки React і Node.js, база даних MongoDB, сервіси Cloudinary, SendGrid, хостинг Render і Vercel.

У результаті розробки створено вебзастосунок, що дозволяє взаємодіяти з оголошеннями про домашніх тварин, зручну авторизацію, роботу з профілем та обраним, а також підтримує адаптивність і масштабованість.

**ABSTRACT**

PET ADOPTION, WEB APPLICATION, JAVASCRIPT, REACT, NODE.JS, MONGODB, RENDER, VERSEL

The object of development is a web application for managing animal-related announcements.

The purpose of the work is to create a web application that allows users to register, manage pet profiles, create and browse announcements, and add posts to favorites.

Solution method – modern web development environment using React and Node.js frameworks, MongoDB database, Cloudinary and SendGrid services, Render and Vercel cloud hosting.

As a result of the development, a functional web application was created that enables interaction with pet announcements, convenient authorization, profile and favorites management, and provides adaptive and scalable architecture.

**ЗМІСТ**

**Вступ………………………………………………………………………………7**

**1. Аналіз предметної галузі……………………………………………………....8**1.1 Аналіз предметної галузі……………………………………..………..8  
1.2 Виявлення проблем та актуалізація ріше……………………..………9  
1.3 Постановка задачі…………………………………………………..…10

**2. Формування вимог до програмної системи**…………………………………12  
2.1 Функціональні вимоги……………………………………………..…12  
2.2 Нефункціональні вимоги…………………………………………..…13

**3. Архітектура та проєктування програмного забезпечення**………………….15  
3.1 UML-проєктування програмного забезпечення……………….....…15  
3.2 Проєктування структури зберігання даних………………………....17  
3.3 Приклади найцікавіших алгоритмів та методів………………….…20  
3.4 Проєктування інтерфейсу користувача…………………………...…22

**4. Опис прийнятих програмних рішень**……………………………………...…30

**5. Тестування розробленого програмного забезпечення**………...……………45  
Висновки ……………………………………………………………………...…49

Перелік джерел посилання …………………………………………….…….…51

Додаток А Звіт результатів перевірки на унікальність тексту в мережі інтернет та базі ХНУРЕ …………………………………………………...……53

Додаток Б Лістинг програмного коду …………………………………………54  
Додаток В Слайди презентації …………………………………………………63

**ВСТУП**

Інформаційні технології все глибше інтегруються у повсякденне життя суспільства, забезпечуючи автоматизацію, спрощення та прискорення різноманітних процесів. Однією з важливих тенденцій сьогодення є розробка цифрових інструментів, що дозволяють ефективно вирішувати соціально значущі завдання — зокрема, у сфері допомоги безпритульним тваринам.

Застосування вебтехнологій відкриває нові можливості для створення платформ, що об'єднують зусилля волонтерів, благодійних організацій та окремих користувачів. Такі системи можуть не лише автоматизувати пошук тварин для всиновлення чи повернення додому, а й підвищити рівень обізнаності, відкритості та довіри в процесі взаємодії між учасниками.

Розробка вебзастосунку з багатофункціональним інтерфейсом, інтеграцією із зовнішніми сервісами та хмарною інфраструктурою є актуальним напрямом підготовки фахівців у галузі інженерії програмного забезпечення.

Метою даної кваліфікаціїної роботи бакалавра є розробка повнофункціонального вебзастосунку Petly, який забезпечує:

* автентифікацію та авторизацію користувачів;
* додавання, редагування та фільтрацію оголошень про тварин;
* зручний інтерфейс з адаптивним дизайном;
* інтеграцію з хмарними сервісами (Cloudinary, SendGrid);
* взаємодію з базою даних MongoDB.

Об'єкт дослідження — процеси розробки вебзастосунків з відкритою архітектурою. Предмет дослідження — методи і засоби створення користувацьких сервісів із застосуванням сучасних вебтехнологій.

У роботі розглянуто ключові етапи створення програмного забезпечення: від аналізу предметної галузі до побудови архітектури, реалізації функціоналу, тестування та оцінки ефективності.

1. **АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ**
   1. Аналіз предметної галузі

Сьогодні тема захисту тварин і пошуку нових власників для безпритульних улюбленців є актуальною як ніколи. У багатьох країнах, зокрема й в Україні, залишається значна кількість тварин, які втратили домівку через економічні причини, воєнні дії або безвідповідальне ставлення людей. Волонтери, притулки та небайдужі громадяни активно створюють ініціативи допомоги, однак їхні зусилля часто є розрізненими та недостатньо ефективними.

У цьому контексті вебзастосунки відіграють ключову роль, адже дають змогу централізувати інформацію, автоматизувати процеси реєстрації тварин і забезпечити зручний інтерфейс взаємодії між волонтерами, притулками та потенційними власниками. Існують міжнародні сервіси на кшталт Petfinder [1] та Adopt a Pet [2], які дозволяють шукати тварин у притулках за різними критеріями. Втім, більшість із них орієнтовані на конкретні країни й лише частково підтримують українську мову, що обмежує їхню ефективність для локальної аудиторії.

У межах цієї кваліфікаційної роботи ставиться мета створення вебзастосунку Petly, який об’єднає найважливіші сценарії взаємодії користувачів. Зокрема, сервіс має:

* забезпечувати швидкий пошук тварин для всиновлення;
* надавати можливість розміщувати оголошення про знайдених або загублених улюбленців;
* публікувати дані про тварин у притулках;
* підтримувати авторизацію через Google та підтвердження реєстрації e‑mail‑повідомленням;
* дозволяти користувачам редагувати персональний профіль і фото.

У результаті Petly покликаний стати єдиною, зручною та україномовною платформою, що підвищить ефективність прилаштування тварин і полегшить координацію всіх залучених сторін.

* 1. Виявлення проблем та актуалізація рішень

Серед ключових бар’єрів, із якими щодня стикаються волонтери, притулки та майбутні власники тварин, можна виділити чотири взаємопов’язані проблеми.

По‑перше, інформація про тварин залишається розпорошеною: оголошення подаються у десятках груп соціальних мереж, на форумах або дрібних сайтах, тому людині складно швидко перевірити, чи актуальне конкретне повідомлення і чи не було тварину вже прилаштовано.

По‑друге, відсутній єдиний, зручний канал комунікації. Зазвичай потенційний власник мусить шукати контакти волонтера в особистих повідомленнях чи телефонувати до притулку, що призводить до затримок, дублювання запитів і навіть втрати зв’язку, якщо повідомлення губляться.

Третьою перешкодою є недовіра до анонімних записів. Якщо оголошення не містить перевірених контактних даних або чіткої дати створення, користувачі схильні ігнорувати його, остерігаючись шахрайства чи застарілої інформації.

Нарешті, більшість міжнародних платформ не мають повної україномовної локалізації або ж надають лише базовий інтерфейс, що ускладнює користування та зменшує залученість української аудиторії.

Саме сукупність невпорядкованої інформації, фрагментарної комунікації, низького рівня довіри та мовного бар’єра формує потребу у спеціалізованому, зручному й повністю локалізованому вебзастосунку, який пропонує Petly.

Таблиця 1.1 — Порівняння аналогічних платформ (таблицю виканано самостійно)

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  | | --- | | Критерії |  |  | | --- | |  | | |  | | --- | | Petfinder |  |  | | --- | |  | | |  | | --- | | Adopt a Pet |  |  | | --- | |  | | OLX «Тварини» | UAnimals Shelter |
| |  | | --- | | Україномовний інтерфейс |  |  | | --- | |  | | |  | | --- | |  |  |  | | --- | | – | | |  | | --- | |  |  |  | | --- | | – | | |  | | --- | |  |  |  | | --- | | + (частково) | | |  | | --- | |  |  |  | | --- | | + | |
| |  | | --- | | Перевірка контактів |  |  | | --- | |  | | |  | | --- | |  |  |  | | --- | | + | | |  | | --- | |  |  |  | | --- | | + | | – | |  | | --- | |  |  |  | | --- | | + | |
| Фільтри пошуку (порода, вік, місто) | + | + | + | |  | | --- | |  |  |  | | --- | | – | |
| Валідація актуальності оголошення | середня | висока | |  | | --- | |  |   низька | висока |
| Авторизація через соцмережі | |  | | --- | |  |  |  | | --- | | – | | |  | | --- | |  |  |  | | --- | | + | | |  | | --- | |  |  |  | | --- | | + | | – |

Таким чином, аналіз показує, що жодна з наявних платформ не охоплює повний спектр потреб українських користувачів. Запропонований застосунок Petly вирішує ці недоліки комплексно: він автоматично вилучає застарілі оголошення завдяки TTL механізму й регулярному підтвердженню актуальності волонтером; забезпечує надійну верифікацію контактів через e mail та Google OAuth 2.0; пропонує повністю україномовний, а також англомовний інтерфейс; крім того, вводить прозору систему рейтингу для волонтерів і притулків, що підвищує довіру користувачів до опублікованої інформації.

* 1. Постановка задачі

Метою кваліфікаційної роботи є розробка вебзастосунку Petly, що забезпечить комплексний сервіс для розміщення, пошуку та модерації оголошень про тварин. Щоб досягти цієї мети, необхідно:

* Спроєктувати клієнт‑серверну архітектуру з REST API та чітким розподілом ролей.
* Імплементувати модуль реєстрації та авторизації, включно з Google OAuth і JWT‑захистом.
* Розробити особистий кабінет, де користувач редагує профіль та керує власними оголошеннями.
* Створити механізм категоризованих оголошень (sell, lost‑found, for‑free), який підтримує фільтри, пошук і «вибране».
* Забезпечити завантаження медіафайлів у хмарне сховище (Cloudinary) з автоматичним масштабуванням.
* Налаштувати відправлення підтверджень та сповіщень через SendGrid.
* Реалізувати адаптивний інтерфейс із сучасним UI / UX для мобільних і десктопних пристроїв.
* Організувати сховище даних у MongoDB Atlas [3] з індексами за ключовими полями для швидкого доступу.

Виконання переліку завдань дозволить створити єдину, зручну та довірену платформу, яка сприятиме ефективнішому пристрою тварин і покращить взаємодію між волонтерами, притулками та майбутніми власниками.

1. **ФОРМУВАННЯ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ**

2.1 Функціональні вимоги

Модуль автентифікації та облікових записів:

* реєстрація за e‑mail / паролем із підтвердженням через SendGrid;
* авторизація за e‑mail / паролем або Google OAuth 2.0;
* видача та перевірка JWT‑токена для захищених запитів.

Ці вимоги гарантують безпечний і зручний вхід до системи для різних категорій користувачів.

Особистий кабінет:

* редагування імені, міста, телефону, аватара;
* миттєве оновлення інформації у базі після збереження.

Завдяки цьому користувач підтримує власні дані в актуальному стані.

Каталог оголошень:

* створення записів у категоріях «Для всиновлення», «Знайдені», «Загублені»;
* прикріплення фото тварини зберігається у Cloudinary;
* можливість редагувати й видаляти власні оголошення.

Функціонал забезпечує повний життєвий цикл публікації для зареєстрованого користувача.

Пошук і фільтрація:

* пошук за ключовими словами;
* фільтрація за категорією та датою створення;
* доступність результатів і для неавторизованих відвідувачів.

Опції пошуку дозволяють швидко знаходити релевантні записи навіть у великому масиві оголошень.

Система «Обране»:

* додавання й вилучення оголошень зі списку вподобань;
* відображення статусу («серце» активне / пасивне).

Механізм «обраного» підвищує залученість користувачів та спрощує повторний доступ до важливих записів.

Служба нотифікацій:

* інформування про успішні операції (реєстрація, оновлення профілю тощо);
* відображення помилок із описом причин.

Зворотний зв’язок допомагає користувачам чітко розуміти результат кожної дії.

2.2 Нефункціональні вимоги

Інтерфейс і юзабіліті:

* адаптивна верстка для смартфонів, планшетів і десктопів;
* навігація через верхнє меню або бургер‑панель (мобільна версія);
* україномовна та англомовна локалізація.

Такі вимоги роблять застосунок зрозумілим і доступним незалежно від типу пристрою та мови користувача.

Продуктивність:

* час відгуку будь‑якої сторінки не перевищує 2 сек;
* підтримка щонайменше 1 000 одночасних сесій;
* використання lazy‑loading для медіа‑контенту.

Дотримання цих показників забезпечує комфортний досвід навіть під час пікових навантажень.

Безпека:

* усі запити передаються через HTTPS;
* критичні маршрути захищено middleware, що перевіряє JWT;
* доступ до кластера MongoDB Atlas [3] обмежено списком IP‑адрес.

Комплекс заходів зводить до мінімуму ризики несанкціонованого доступу та витоку даних.

Масштабованість і підтримуваність:

* клієнт (React) і сервер (Node.js + Express) розділені за принципом front‑/back‑ end;
* можливість горизонтального масштабування контейнерів;
* CI/CD‑деплоймент: Render для API, Vercel (або Netlify) для клієнта.

Така архітектура дозволяє незалежно розширювати кожен шар і впроваджувати нові функції без простою сервісу.

Експлуатаційні умови:

* цілодобова доступність сервісу (SLA ≥ 99,5 %);
* резервне копіювання бази даних за розкладом;
* автоматичне видалення неактуальних оголошень за допомогою TTL‑полів.

Умови експлуатації визначають правила роботи системи в продакшен‑середовищі та політику резервування.

Надійність:

* перевірка актуальності кожного оголошення волонтером;
* логування критичних подій і помилок на сервері;
* можливість «гарячого» оновлення без втрати доступу.

Надійність гарантує сталість даних і безперервність сервісу навіть під час оновлень.

Сформульовані вимоги визначають технічні та дизайнерські рішення, які будуть використані на етапі архітектурного проєктування та подальшої реалізації Petly. Вони забезпечують баланс між функціональністю, зручністю, безпекою та потенціалом до масштабування.

1. **АРХІТЕКТУРА ТА ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ**

3.1 UML-проєктування програмного забезпечення

Для Petly центральною є Use Case‑діаграма, яка фіксує зовнішню поведінку системи: хто з нею взаємодіє та які сценарії запускає.   
Діаграма містить два актори, що позначають типові ролі відвідувачів сайту.

* User (авторизований) — зареєстрований власник акаунта, який має доступ до персональних модулів (профіль, тварини, оголошення, «обране»).
* Guest (гість) — анонімний відвідувач, що переглядає публічний контент і може ініціювати реєстрацію або вхід.

Розташування акторів на діаграмі не випадкове: користувача розміщено ліворуч як головного ініціатора більшості сценаріїв, а гостя — праворуч, щоб візуально відокремити обмежені можливості аноніма.

* Контент: Browse Notices, Search Notices, View News, View Friends.
* Аутентифікація: Register, Login, Logout.
* Профіль: Manage Profile.
* Тварини: Add Pet, Delete Pet.
* Оголошення: Create Notice, Edit/Delete Notice.
* Обране: View Favorites, Add to Favorites, Remove from Favorites.

Колір еліпсів у діаграмі підказує, кому доступний сценарій: жовтий — публічна дія, блакитний — лише після входу. Така візуальна кодова система спрощує читання та одразу розмежовує права користувачів.

Опис функціонального призначення веб-системи зображено за допомогою наступної USE-CASE діаграми на рисунку 3.1:

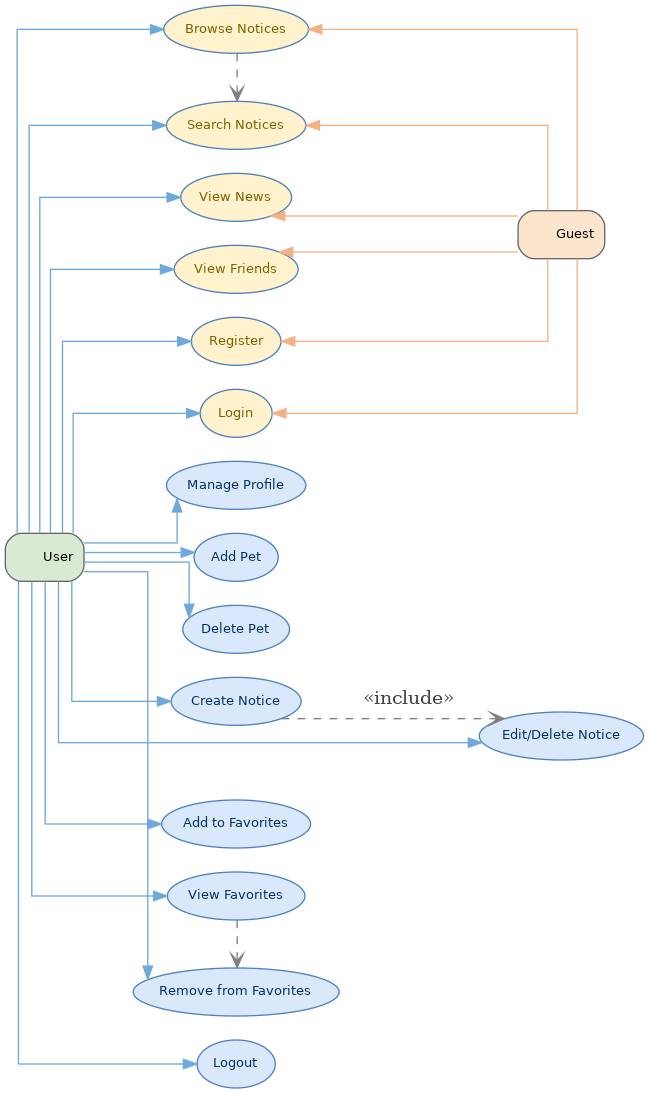


Рисунок 3.1 – USE-CASE діаграма (рисунок виканано самостійно)

Для зображення логічних залежностей між сценаріями використано пунктирні стрілки з підписами.

* «Browse Notices» extend «Search Notices» — пошук є розширенням базового перегляду: щойно вводиться ключове слово, активується додатковий фільтр.
* «View Favorites» extend «Remove from Favorites» — безпосередньо з екрану перегляду уподобань користувач може одразу прибрати запис.
* «Create Notice» include «Edit/Delete Notice» — форму створення оголошення повторно використовують для його редагування або видалення.

Такі зв’язки допомагають розробникам побачити місця повторного використання компонентів і точки розширення логіки.

Читаючи діаграму зліва направо, спершу бачимо повний спектр можливостей авторизованого користувача, а праворуч — обмежений набір дій гостя.

Dashed‑стрілки підказують, які сценарії є підфункціями або повторно використовують один і той самий інтерфейс. У підсумку діаграма чітко розмежовує відкритий і захищений функціонал та демонструє внутрішню структуру модулів «Оголошення» й «Обране».

3.2 Проєктування структури зберігання даних

Для вебзастосунку Petly обрано документ‑орієнтовану базу даних MongoDB Atlas.

Такий підхід природно відображає об’єктну сутність оголошень, тварин і користувачів та забезпечує горизонтальне масштабування, необхідне для зростання платформи. На рисунку 3.2 показано ER‑діаграму, що стала вихідною точкою для фізичного моделювання колекцій.

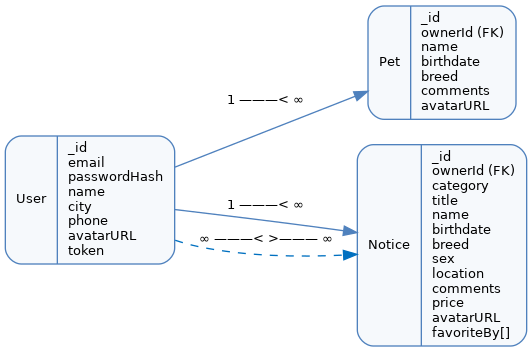


Рисунок 3.2 –ER-діаграма (рисунок виканано самостійно)

Сутність User  
Колекція users зберігає облікові записи. Кожен документ містить поля:

* \_id
* email
* passwordHash
* name
* city
* phone
* avatarURL
* token

Е‑mail проіндексовано як унікальний, що дає швидкий пошук при автентифікації та виключає дублювання облікових записів. Зберігання token у документі спрощує вихід із системи — під час логаута поле зану­лю­єть­ся, і JWT‑токен автоматично стає недійсним.

Сутність Pet. Колекція pets описує тварин, які належать користувачам. Атрибут ownerId посилається на \_id із users, утворюючи зв’язок *«один → багато»*. Такий референс, на відміну від вкладеного масиву, дозволяє легко вибирати тварин конкретного власника і, водночас, не «роздуває» документ користувача, якщо він додасть кілька десятків улюбленців.

 Сутність Notice. Оголошення розміщено в колекції notices. Обов’язковими є category, title, location та ownerId; додаткові поля — порода, опис, фото й ціна (актуальна лише для категорії «Для всиновлення», тобто для продажу). Масив favoriteBy[] зберігає ObjectId користувачів, які додали запис у «вибране». Таке вбудоване рішення спрацьовує швидше, ніж окрема таблиця‑зв’язка, доки кількість уподобань не перевищує декількох тисяч; на випадок подальшого зростання передбачено міграцію до колекції favorites.

* User ↔ Pet — *1 → ∞*. Користувач може зареєструвати необмежену кількість тварин.
* User ↔ Notice (owner) — *1 → ∞*. Лише автор має право змінювати або видаляти власне оголошення.
* User ↔ Notice (favorite) — *∞ ↔ ∞*. Багато користувачів можуть зберігати багато оголошень у списку вподобань; реалізовано через масив favoriteBy[].

Цілісність даних забезпечується комбінацією валідаційних схем MongoDB і серверних перевірок: наприклад, створити оголошення без завантаженого фото неможливо, а формат телефону верифікується регулярним виразом як на фронті, так і на бекенді.

* Гнучкість: JSON‑документи легко розширювати новими полями (наприклад, вакцинація тварини) без дорогих міграцій.
* Продуктивність: відсутність складних приєднань («join») пришвидшує читання популярних сторінок, а індекси мінімізують затримки під час пошуку.
* Масштабування: кластери MongoDB Atlas підтримують шардинг, а це дає змогу вертикально і горизонтально нарощувати потужності без зупинки сервісу.

Узгоджена із Use Case‑діаграмою ER‑модель формує цілісну основу, на якій будується решта архітектурних рішень — від API‑контракту до механізмів кешування та відновлення даних. Саме вона гарантує, що функціональні вимоги (створення, пошук, «обране», керування профілем) реалізуються швидко та безпечно, а запас гнучкості дозволяє розширювати сервіс у майбутньому без радикальної перебудови сховища.

3.3 Приклади найцікавіших алгоритмів та методів

У процесі реалізації системи Petly було застосовано низку алгоритмів і технік, спрямованих на оптимізацію користувацького досвіду, підвищення безпеки та ефективність пошуку. Нижче наведено приклади найцікавіших реалізацій.

3.3.1 Пошук оголошень за ключовими словами

Функція пошуку реалізована на бекенді через фільтрацію за заголовком оголошення (title) за допомогою регулярних виразів у MongoDB:

const regex = new RegExp(searchQuery, "i"); // нечутливий до регістру

const results = await Notice.find({ title: { $regex: regex } });

Такий підхід дозволяє гнучко шукати оголошення за частиною слова без урахування регістру.

3.3.2 Авторизація через Google (OAuth 2.0)

Алгоритм побудований за стандартною схемою OAuth[5]:

* Користувач клікає “Login with Google”
* Google перенаправляє назад із кодом
* Бекенд обмінює код на профіль користувача
* Якщо користувача не існує — створюється новий запис у базі
* Генерується JWT, який повертається на фронт

Це дозволяє швидко увійти без пароля та зменшує бар’єр входу в систему.

3.3.3 Валідація форм

На фронтенді форми реєстрації та логіну проходять обов’язкову валідацію перед відправкою:

* email — регулярний вираз для перевірки формату
* password — мінімум 7 символів, без пробілів
* phone — формат +380\*\*\*\*\*\*\*\*\*
* city — перевірка на формат Місто, Область

Це реалізовано через бібліотеку Formik + Yup, що дозволяє централізовано перевіряти поля та виводити повідомлення про помилки.

3.3.4 Завантаження зображень у Cloudinary

Зображення (аватари, фото тварин) завантажуються через API Cloudinary[6]:

const result = await cloudinary.uploader.upload(path, {

folder: folderName,

public\_id: userId,

transformation: [{ width: 600, crop: "limit" }]

});

Це дозволяє автоматично обробляти зображення перед збереженням та зменшувати розмір для прискорення завантаження сторінок.

Усі ці алгоритми не лише розширюють функціональність застосунку, а й забезпечують якісний користувацький досвід та стабільність системи в умовах реального використання.

3.4. Проєктування інтерфейсу користувача

3.4.1 Цілі та завдання створення програмного продукту

Проєкт Petly має на меті допомогти домашнім тваринам знайти чи повернутися до родини. Сервіс реалізує декілька ключових сценаріїв:

* Продаж тварин (якщо власник хоче знайти нового господаря, отримавши при цьому матеріальну вигоду).
* Віддання в добрі руки (безкоштовно передати вихованця тим, хто готовий надати йому належний догляд).
* Lost/Found (загубився/знайшовся): допомогти людям, які загубили свого улюбленця або навпаки знайшли чужу тварину, швидше з’єднатися з власником.

Додаткові завдання проєкту:

* Розміщення актуальних новин зі світу домашніх тварин (сторінка «Новини» / «News»).
* Список партнерів (ветклініки, магазини кормів, притулки, зоомагазини тощо) з контактною інформацією, які можуть бути корисними власникам тварин (сторінка «Партнери» / «Our Friends»).

Таким чином, основна ціль – створити зручний інформаційний майданчик, де будь-який користувач зможе:

* Опублікувати або переглянути оголошення стосовно тварин.
* Переглянути контакти корисних установ чи організацій.
* Дізнатись актуальні новини, пов’язані з домашніми улюбленцями.

Завдання програмного продукту:

* Забезпечити швидкий пошук потрібних оголошень за допомогою категорій («sell», «lost-found», «for-free») та ключового слова.
* Надати зручний інтерфейс створення та редагування власних оголошень (для зареєстрованих користувачів).
* Зробити процес реєстрації та авторизації максимально простим і захищеним.
* Реалізувати перегляд контактної інформації ветеринарних клінік, магазинів, притулків та інших корисних партнерів.
* Збирати та відображати новини, що будуть цікавими або корисними власникам тварин.
* Забезпечити зручне керування власним профілем (оновлення даних про себе, додавання інформації про власних тварин тощо).

Оскільки цільова аудиторія – це широке коло людей, котрі мають (або хочуть мати) домашніх тварин, серед яких є і досвідчені користувачі, і люди з мінімальним рівнем комп’ютерної грамотності, інтерфейс проєкту орієнтується на простоту та інтуїтивність.

* Обрано діалог на основі екранних форм та меню.
* Використовується помірна кількість полів введення та доступна колірна гама, щоб унаочнити процеси пошуку/реєстрації та не перевантажувати інтерфейс.
* Шрифти та кольори дібрані з урахуванням доступності для користувачів із можливими порушеннями зору (контрастна гама).
* Додані піктограми для категорій оголошень, кнопок дій, переходів у профіль тощо, аби користувачу було зручніше швидко орієнтуватися.

3.4.2 Прототипування набору необхідних сторінок/форм

Головна сторінка (Home) (Рисунок 3.3) :

* У шапці сайту розміщено логотип «Petly», меню з пунктами «News», «Notices», «Our Friends», а праворуч кнопка «Login» чи «Profile» (залежно від авторизації).
* Стандартні засоби навігації: основне меню згори, при переході курсором – підсвічування пунктів (стани hover).

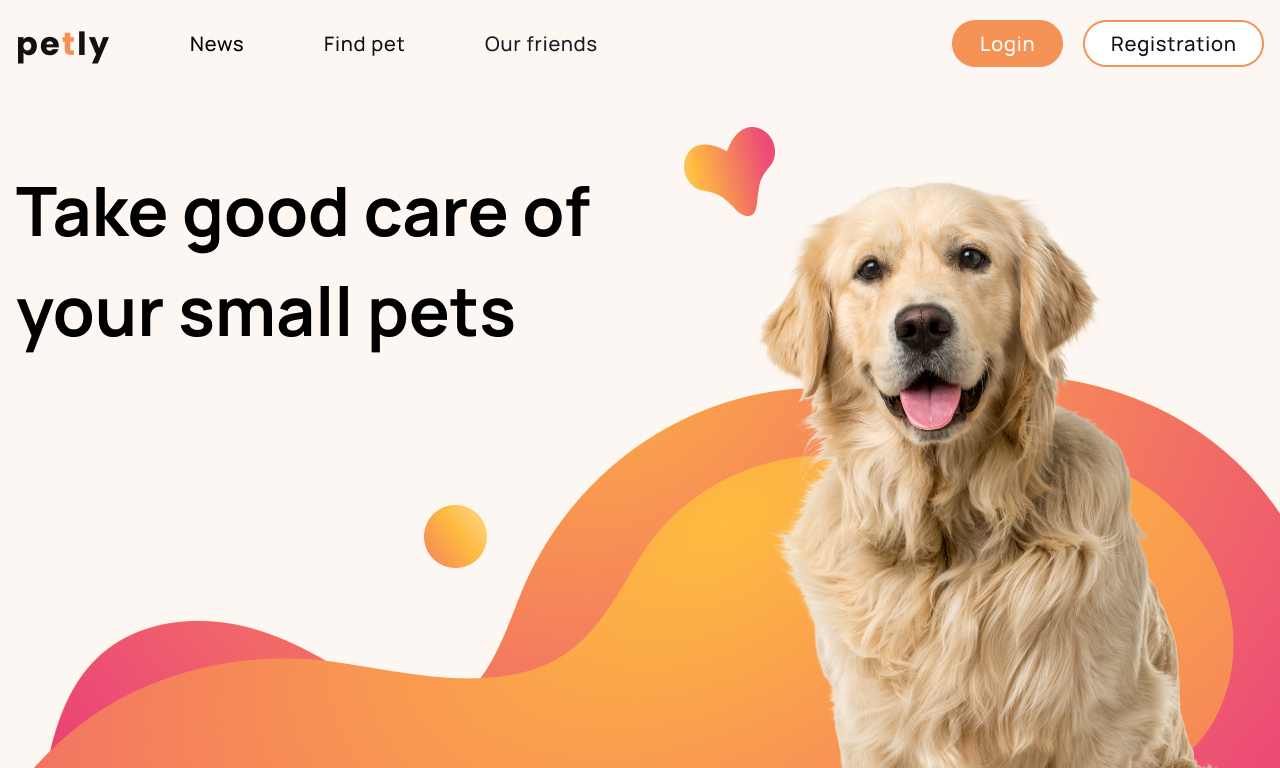


Рисунок 3.3 – Сторінка Home (рисунок виканано самостійно)

Сторінка «Новини» (News) (Рисунок 3.4):

* Відображає список новин, отриманих із сервера.
* Зверху є поле пошуку (додатковий елемент управління: текстове поле + кнопка “пошук” із іконкою лупи).
* Кожна новина відображається у вигляді картки з назвою, коротким описом, датою. Кнопка “Read more”
* Новини відображені блоками з датою, заголовком, коротким описом та кнопкою «Read More», котра відкриває повну версію в новій вкладці.. Найновіші – вгорі списку.

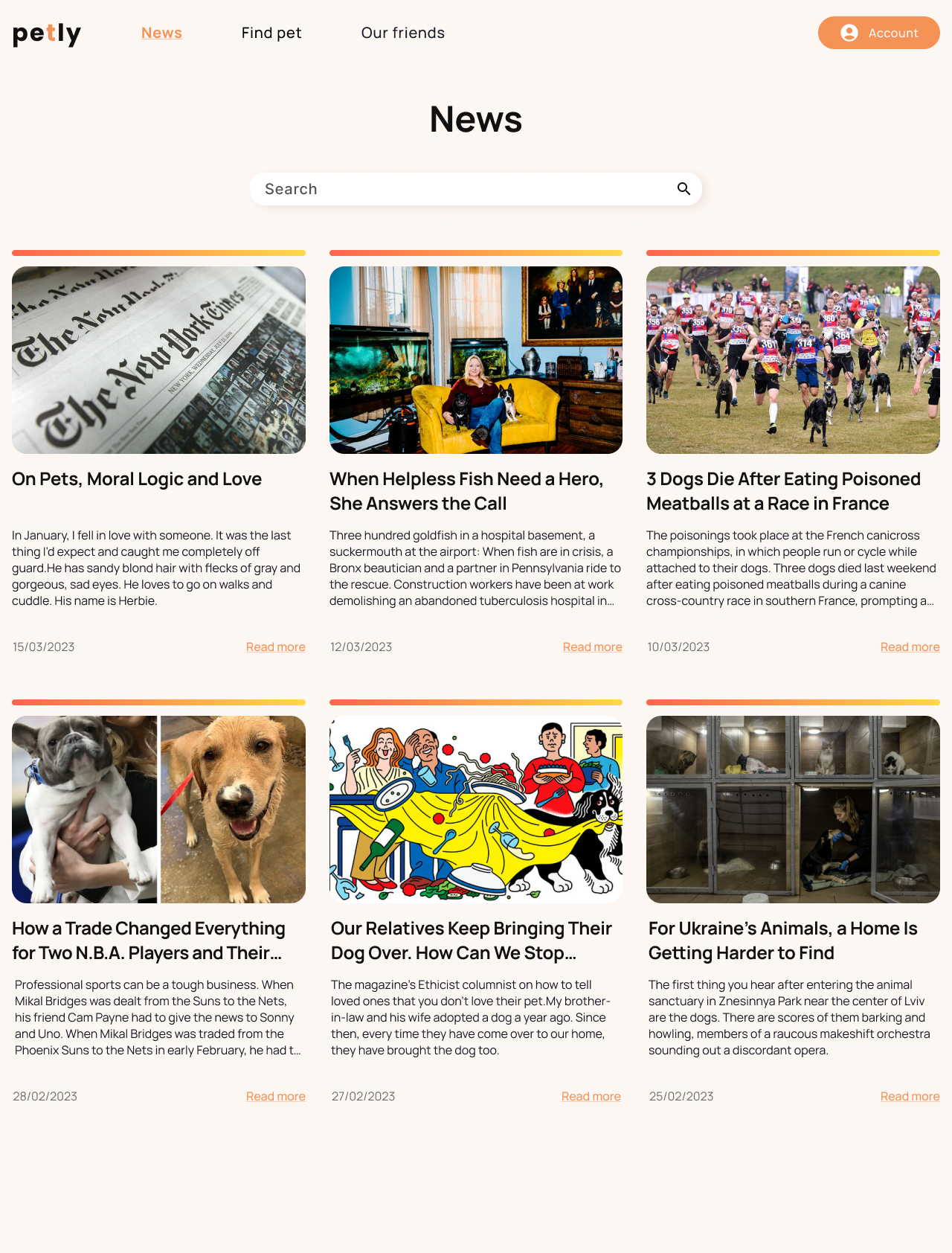


Рисунок 3.4 – Сторінка News (рисунок виканано самостійно)

Сторінка «Наші партнери» (Our Friends) (Рисунок 3.5):

Містить перелік партнерських організацій (ветклініки, притулки тощо).

* Кожна карточка може містити логотип, назву, адресу, контакти та розклад роботи.
* Контактні дані оформлені як гіперпосилання, які можна відкрити в окремій вкладці або.
* Показано, що при натисканні на «робочі години» може відкриватися додаткове вікно або випадаюча інформація.



Рисунок 3.5 – Our Friends (рисунок виканано самостійно)

Сторінка авторизації / логін (Login) (Рисунок 3.6):

* Включає форму введення логіну та пароля з кнопкою «Login».
* Розташування елементів відповідає стандартним патернам: поля введення одне під одним, кнопка відправки після них.
* Можливість перейти на реєстрацію (посилання «Register»).
* Підказки з’являються при некоректно введених даних (наприклад, “Пароль має містити не менше 8 символів”).

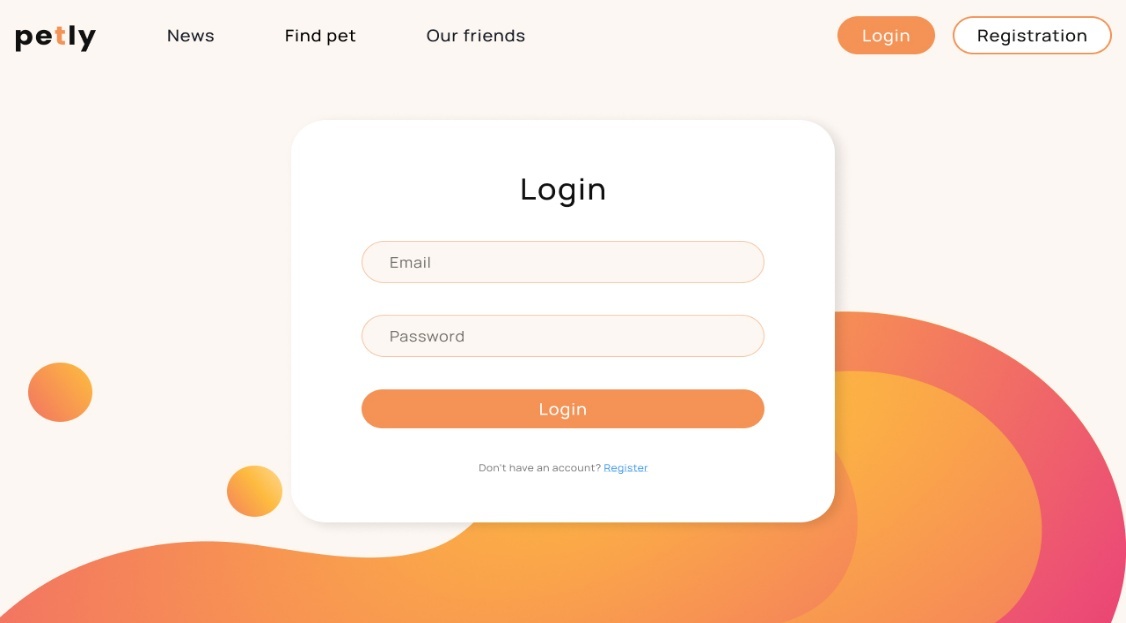


Рисунок 3.6 – Login (рисунок виканано самостійно)

Сторінка профілю (Profile / User Page) (Рисунок 3.7):

* Відображається інформація про користувача (ім’я, email тощо).
* Кнопка «Edit»/«Logout».
* Перелік тварин користувача: фото, ім’я, порода. Можна додати або видалити.

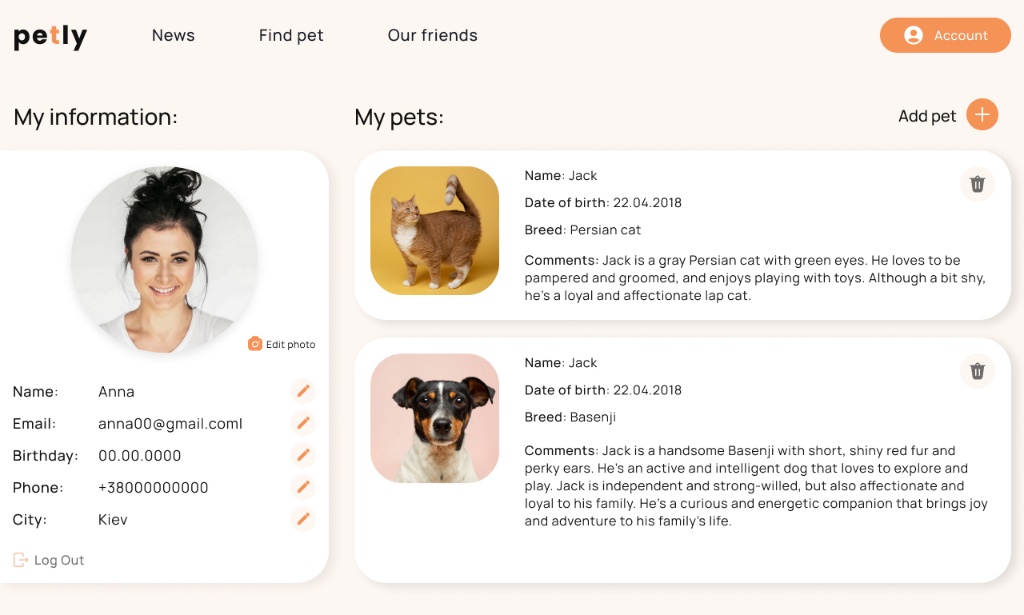


Рисунок 3.7 – Profile / User Page(рисунок виканано самостійно)

Сторінка з оголошеннями (Notices) (Рисунок 3.8) :

* Користувач може вибрати категорію (sell, lost-found, for-free) через меню або піктограми (іконки).
* Присутній пошук, можливі фільтри (радіокнопки/прапорці) для відбору певних характеристик оголошень.
* На сторінці передбачено кнопку «Створити оголошення», що відкриває модальне вікно або переносить на форму створення нового оголошення.

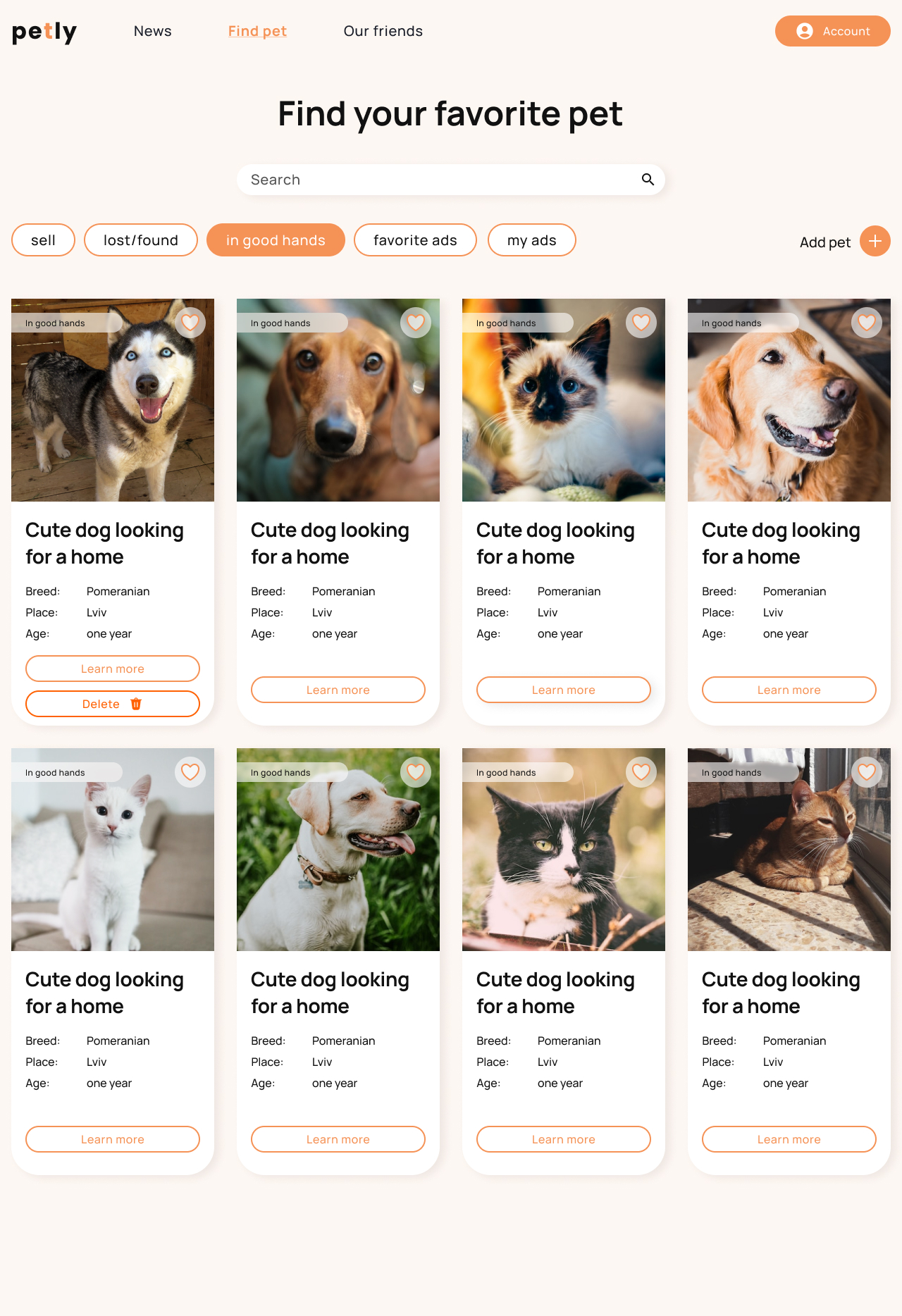


Рисунок 3.8 – Notices (рисунок виканано самостійно)

Усі сторінки створені в Figma з дотриманням єдиного візуального стилю:

* Використано спокійні пастельні кольори й контрастний текст (наприклад, темний текст на світлому фоні) для покращення зчитуваності.
* Розмір шрифту та відступи уніфіковані (щоб забезпечити зручне сканування контенту).

У **Petly** використовується єдина кольорова палітра та шрифти на всіх сторінках, головне меню фіксується вгорі з логотипом ліворуч і кнопкою авторизації чи профілю праворуч.

Для взаємодії з користувачем передбачено стандартні форми входу/реєстрації з полями email і пароль, а також кнопки «Login», «Register», «Submit», «Cancel» тощо. Кожен пункт текстового меню має відповідний піктограмічний аналог (наприклад, «лапка» – для головної, «газета» – для новин), а поточна активна сторінка підсвічується в шапці чи в окремому індикаторі стану (наприклад, для повідомлення про успішне збереження даних).

У вебреалізації можна налаштувати гарячі клавіші (скажімо, Ctrl+Enter для відправлення форми або Esc для закриття модалки). Також доступні додаткові елементи управління, як-от випадаючі списки в оголошеннях, лінійки прокрутки, що з’являються при великому обсязі контенту. При наведенні курсора на іконки або кнопки система виводить підказки (tooltip), а за некоректного введення даних формується відповідне попередження з описом помилки.

Для користувача забезпечується зрозумілий шлях від пошуку потрібної інформації чи оголошення до створення власних оголошень, а також дає змогу швидко ознайомитися з актуальними новинами та партнерами.

1. **ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ**

У розділі наведено детальний огляд ключових технічних рішень, архітектури та алгоритмів, що лежать в основі Petly. Опис спирається на реальні фрагменти коду із репозиторіїв petlyback‑main (бекенд) та petly‑project‑main (фронтенд).

Платформа побудована за схемою SPA + REST API (рис. 4.1): клієнтська частина — це React‑застосунок, який, завантажившись одного разу, далі обмінюється з сервером лише JSON‑повідомленнями.

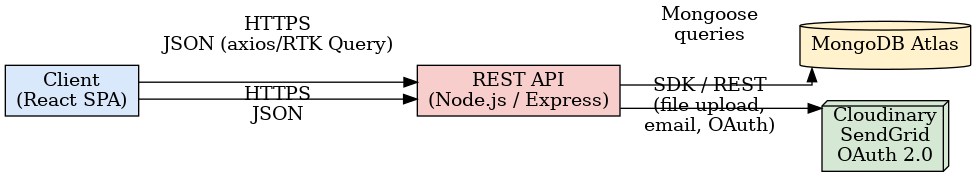


Рисунок 4.1 – SPA + REST API (рисунок виканано самостійно)

Сервер працює на Node.js + Express [8] і реалізує тонкий шар логіки: приймає запити, перевіряє права, викликає сервіси й повертає дані.

* Frontend (Netlify). React [9], Redux Toolkit, RTK Query для кешування запитів, styled‑components для стилів і Notiflix — для сповіщень.
* Backend (Render). Express‑сервер [10], модульні маршрути, Mongoose [4] як ORM до MongoDB Atlas.
* Зовнішні сервіси. Cloudinary (зображення), SendGrid [7] (e‑mail), Google OAuth 2.0 (соціальний вхід).
* CI/CD. GitHub Actions збирає тестовий білд, після успіху авто‑деплоїть на Netlify і Vercel.

Завдяки чіткому поділу зон відповідальності клієнт і сервер можна масштабувати окремо, а додавання мікросервісів у майбутньому не потребуватиме зміни базової топології.

*src/services/notices/getNoticesByCategory.js*

*const* { Notices } = require("../../db/noticesModel");

*const* { WrongParametersError } = require("../../helpers/errors");

*// Сервіс повертає оголошення потрібної категорії з пошуком і пагінацією*

*const* getNoticesByCategoryService = *async* (*category*, *search*, *page*, *limit*) => {

  limit = +limit > 8 ? 8 : +limit;

  page = +page;

*const* searchObj = {};

*// гілка для списку категорій (sell / lost-found / in-good-hands)*

  if (["sell", "lost-found", "in-good-hands"].includes(category)) {

    searchObj.categoryName = category;

    if (search) searchObj.title = { $regex: search, $options: "i" };

*// рахуємо одразу й total, і список сторінки (Promise не потрібен — два запити послідовно)*

*const* dataLength = await Notices.countDocuments(searchObj);

*const* data = await Notices.find(searchObj)

      .skip((page - 1) \* limit)

      .limit(limit)

      .sort({ updatedAt: -1 })

      .select(["-createdAt", "-updatedAt"]);

    return {

      data,

      perPage: limit,

      total: dataLength,

      noticesLeft:

        dataLength - page \* limit > 0 ? dataLength - page \* limit : 0,

      pageCount: Math.ceil(dataLength / +limit),

      currentPage: page,

      noticesOnPage: data.length,

    };

  }

*const* data = await Notices.findById(category);

  if (!data) throw new WrongParametersError("Not found");

  return {

    data,

  };

};

*module*.*exports* = { getNoticesByCategoryService };

Ця асинхронна функція інкапсулює всю бізнес‑логіку вибірки оголошень із колекції Notices. Алгоритм працює у двох режимах. Якщо значення параметра category збігається з однією з базових груп (*sell*, *lost‑found*, *in‑good‑hands*), формується об’єкт searchObj, що може містити фільтр за назвою ($regex, нечутливий до регістру) та номер категорії. Після цього виконується послідовна пара операцій: спершу countDocuments, аби дізнатися загальну кількість відповідних записів, далі find із skip, limit і sort.

Отримані значення упаковуються у відповідь, яка одразу містить і сторінкові метадані — *currentPage*, *pageCount*, *noticesLeft* тощо.

Якщо ж у category приходить \_id, сервіс переходить до «detail‑гілки» і виконує findById. У разі відсутності документа викидається виняток WrongParametersError("Not found"), що трансформується у HTTP 404 на рівні глобального обробника помилок. Подібне розділення логіки дозволяє одним і тим самим ендпоінтом обслуговувати як списки оголошень, так і детальні запити, не дублюючи код і не створюючи зайвих маршрутів.

petlyback-main/src/helpers/petPhotoUpload.js[6]

*const* fs = require("fs");

*const* cloudinary = require("cloudinary").v2;

cloudinary.config({

  cloud\_name: process.env.CLOUD\_NAME,

  api\_key: process.env.CLOUD\_KEY,

  api\_secret: process.env.CLOUD\_SECRET,

});

*const* petPhotoUpload = *async* (*file*, *folder*, *filename*) => {

*const* cloudFilePath = folder + "/" + filename;

*const* options = {

    public\_id: cloudFilePath + Math.floor(Math.random() \* 10000),

    overwrite: true,

  };

  console.log("public\_id:", options.public\_id);

  try {

*const* { url } = await cloudinary.uploader.upload(file, options);

    await fs.unlinkSync(file);

    return {

      message: "Upload success",

      url,

    };

  } catch (error) {

    fs.unlinkSync(file);

    return {

      message: "Upload fail",

      error,

    };

  }

};

*module*.*exports* = { petPhotoUpload };

* Модуль приймає локальний шлях file, папку folder (наприклад petly/pets) і filename — основу майбутнього public\_id.
* Через Math.random() додає до імені 4‑значний суфікс, щоб уникнути колізій.  
  Викликає cloudinary.uploader.upload, отримує url, одразу видаляє тимчасовий файл і повертає успішну відповідь.
* У разі помилки також видаляє tmp‑файл, але повертає об’єкт із error, не кидаючи виняток — це дозволяє контролеру самостійно вирішувати, як сповістити користувача.

Таким чином бінарні дані не залишаються на сервері, Cloudinary бере на себе ресайз і кешування, а контролер отримує вже готовий HTTPS‑URL для збереження в MongoDB.

Перевірка JWT‑токена

petlyback-main/src/middlewares/authMiddleware.js

*const* jsonwebtoken = require("jsonwebtoken");

*const* { NotAuthorizedError } = require("../helpers/errors");

*const* { User } = require("../db/userModel");

*const* authMiddleware = *async* (*req*, *res*, *next*) => {

  try {

    if (!req.headers.authorization)

      next(

        new NotAuthorizedError(

          "Please, provide a token in request authorization header"

        )

      );

*const* [, token] = req.headers.authorization.split(" ");

    if (!token || !jsonwebtoken.verify(token, process.env.JWT\_SECRET))

      next(new NotAuthorizedError("Please, provide a token"));

*const* user = jsonwebtoken.verify(token, process.env.JWT\_SECRET);

*const* findedUser = await User.findById(user.\_id);

    if (!findedUser) next(new NotAuthorizedError("Not authorized"));

    if (token !== findedUser.token)

      next(new NotAuthorizedError("Not authorized"));

    req.token = token;

    req.user = findedUser;

    next();

  } catch (error) {

    next(new NotAuthorizedError("Not authorized"));

  }

};

*module*.*exports* = { authMiddleware };

* Перевіряє наявність заголовка Authorization.
* Виділяє токен і двічі валідовує – спершу синтаксично (jsonwebtoken.verify), потім логічно — звіряє зі збереженим у користувача token.
* У разі будь‑якої невідповідності кидає NotAuthorizedError (401).
* Додає об’єкт findedUser у req для подальших контролерів.

Центральна логіка маршрутизації App.jsx

* Імпортуються React-хук, хуки Redux, компоненти маршрутизації та стилі.
* При монтуванні через useEffect відправляється getUser() для отримання даних користувача (isLoading контролює Loader).
* Головна структура маршрутизації створена через <Routes> з SharedLayout як основою.
* PublicRoute/PrivateRoute керують доступом до сторінок (реєстрація, логін, користувацька зона).
* Маршрут /notices має вкладені маршрути для категорій та елементів, а "\*" обробляє невідомі URL.
* Закоментований BrowserRouter вказує на попереднє використання іншого контейнера маршрутизації.

import { useEffect } from "react";

import { useDispatch, useSelector } from "react-redux";

import { Route, Routes } from "react-router-dom";

import { Loader } from "./components/Loader/Loder";

import PublicRoute from "./components/PublicRoute/PublicRoute";

import PrivateRoute from "./components/PrivateRoute/PrivateRoute";

import SharedLayout from "./components/SharedLayout/SharedLayout";

import { GlobalStyle } from "./GlobalStyles";

import HomePage from "./pages/HomePage/HomePage";

import LoginPage from "./pages/LoginPage/LoginPage";

import NewsPage from "./pages/NewsPage/NewsPage";

import NoticesPage from "./pages/NoticesPage/NoticesPage";

import OurFriendsPage from "./pages/OurFriendsPage/OurFriendsPage";

import RegisterPage from "./pages/RegisterPage/RegisterPage";

import UserPage from "./pages/UserPage/UserPage";

import { getUser } from "./redux/auth/auth-operations";

import { selectIsLoading } from "./redux/auth/auth-selectors";

import NotFound from "./components/NotFound/NotFound";

import NoticesPageContent from "./components/NoticesPageContent/NoiticesPageContent";

*const* App = () => {

*const* dispatch = useDispatch();

*const* isLoading = useSelector(selectIsLoading);

  useEffect(() => {

    dispatch(getUser());

  }, [dispatch]);

  return (

    <>

      <Loader *isLoading*={isLoading} />

      <GlobalStyle />

      <Routes>

        <Route *path*="/" *element*={<SharedLayout />}>

          <Route *index* *element*={<HomePage />} />

          <Route

*path*="/register"

*element*={

              <PublicRoute *redirectTo*="/user">

                <RegisterPage />

              </PublicRoute>

            }

          />

          <Route

*path*="/login"

*element*={

              <PublicRoute *redirectTo*="/user">

                <LoginPage />

              </PublicRoute>

            }

          />

          <Route

*path*="/user"

*element*={

              <PrivateRoute>

                <UserPage />

              </PrivateRoute>

            }

          />

          <Route *path*="/news" *element*={<NewsPage />} />

          <Route *path*="/friends" *element*={<OurFriendsPage />} />

          <Route *path*="/user" *element*={<UserPage />} />

          <Route *path*="/notices" *element*={<NoticesPage />}>

            <Route *index* *element*={<NoticesPageContent />} />

            <Route *path*=":category" *element*={<NoticesPageContent />}>

              <Route *path*=":id" />

            </Route>

          </Route>

          <Route *path*="\*" *element*={<NotFound />} />

        </Route>

      </Routes>

    </>

*// </BrowserRouter>*

  );

};

export default App;

Цей файл організовує центральну логіку маршрутизації, завантаження початкових даних (користувача) та встановлення глобальних стилів, що забезпечує єдність та узгодженість застосунку.

Компонент PetsList.jsx — відображення й керування картками тварин

import PropTypes from "prop-types";

import { DeletePetBtn } from "../../DeletePetBtn/DeletePetBtn";

import {

  WrapperPicDiv,

  Box,

  Description,

  InfoPet,

  BoxPetsList,

  BoxScrollbar,

  EditPetBtn,

} from "./PetsList.styled";

import defImage from "../../../img/defaultImg.jpeg";

import { ReactComponent as Edit } from "../../../assets/svg/penciNotices.svg";

export *const* PetsList = ({ *pets*, *onPostHandler*, *removeHandler* }) => {

  return (

    <BoxPetsList>

      <BoxScrollbar>

        {pets?.map((*el*) => {

*const* { avatarURL, birthday, breed, comments, name, \_id } = el;

          return (

            <Box *key*={\_id}>

              <WrapperPicDiv>

                <img *src*={avatarURL ? avatarURL : defImage} *alt*={name} />

              </WrapperPicDiv>

              <Description>

                <DeletePetBtn *name*={\_id} *onClick*={removeHandler} />

                <EditPetBtn *name*={\_id} *onClick*={onPostHandler}>

                  <Edit *width*={30} *height*={30} />

                </EditPetBtn>

                <InfoPet>

                  <b> Name:</b> {name ? name : "No Info"}

                </InfoPet>

                <InfoPet>

                  <b> Date of birth:</b> {birthday ? birthday : "No Info"}

                </InfoPet>

                <InfoPet>

                  <b>Breed:</b> {breed ? breed : "No Info"}

                </InfoPet>

                <InfoPet>

                  <b> Comments:</b> {comments ? comments : "No Info"}

                </InfoPet>

              </Description>

            </Box>

          );

        })}

      </BoxScrollbar>

    </BoxPetsList>

  );

};

PetsList.propTypes = {

  pets: PropTypes.array.isRequired,

};

PetsList.jsx відповідає за візуалізацію масиву даних про тварин поточного користувача. Компонент отримує три пропси: pets (масив документів), onPostHandler (колбек відкриття форми редагування) та removeHandler (видалення картки). Усередині блоку <BoxScrollbar> перелік перебирається методом map, а кожна тварина перетворюється на верстку з фото, кнопками‑діями та базовими атрибутами.

* Зображення**.** Якщо avatarURL порожній, підставляється дефолтне defImage, що запобігає зламаним посиланням.
* Кнопки.
* <DeletePetBtn> виносить логіку підтвердження та запиту DELETE /api/pets/:id;
* <EditPetBtn> запускає модальне вікно зі спрощеною формою редагування.
* Дані. Поля name, birthday, breed, comments виводяться з перевіркою на порожнє значення — замість undefined користувач бачить “No Info”, що покращує UX.
* Скролбар. Контейнер BoxScrollbar (custom CSS) залишає фіксовану висоту й малює власний тонкий скрол, аби довгий список не розтягував сторінку, але залишався доступним.

Таким чином компонент вирішує одразу три задачі: акуратно показує картки, дозволяє їх редагувати/видаляти одним кліком і гарантує, що навіть із порожніми даними інтерфейс лишається охайним та інформативним.

Хук useNoticesParams та контекст NoticesProvider — глобальний стан пагінації й пошуку оголошень

Файл UseMyContext.jsx інкапсулює React‑контекст для параметрів оголошень, щоб уникнути проп‑дрилінгу між сторінкою NoticesPage, пошуковою формою та списком категорій.

import { createContext, useContext, useState } from "react";

*const* NoticesContext = createContext();

export *const* useNoticesParams = () => useContext(NoticesContext);

export *const* NoticesProvider = ({ *children* }) => {

*const* [page, setPageParams] = useState(1);

*const* [search, setSearchParams] = useState("");

*const* setPage = (*value*) => {

    setPageParams(value);

  };

*const* setSearch = (*value*) => {

    setSearchParams(value);

  };

  return (

    <NoticesContext.Provider *value*={{ setPage, page, setSearch, search }}>

      {children}

    </NoticesContext.Provider>

  );

};

* NoticesContext зберігає рівно два параметри — page і search — та методи їх зміни; цього достатньо, щоб будь‑який дочірній компонент міг керувати пагінацією й фільтром без передавання пропсів униз по дереву.
* useNoticesParams() — кастомний хук, який повертає об’єкт контексту; споживання виглядає так:

*const { page, setPage, search, setSearch } = useNoticesParams();*

* Скидання стану. Коли користувач вводить нове ключове слово, компонент пошуку викликає setSearch(value) і одночасно setPage(1), щоб результати показувалися з першої сторінки.
* Поєднання з RTK Query. Хук передає page та search у useGetNoticesQuery({ category, page, search }); при зміні будь‑якого параметра RTK Query автоматично рефетчить дані.

У підсумку NoticesProvider виступає «єдиним джерелом істини» для навігаційних параметрів оголошень, спрощуючи логіку компонентів і запобігаючи дублюванню стану.

Компонент AddsPetForm.jsx — двокрокова форма створення / редагування картки тварини.

import { StepOne } from "./steps/StepOne";

import { StepTwo } from "./steps/StepTwo";

import { useState } from "react";

export *const* AddsPetForm = ({ *onClose*, *onEdit*, *post* }) => {

*const* [data, setData] = useState({

    name: post?.name || "",

    birthday: post?.birthday || "",

    breed: post?.breed || "",

    comments: post?.comments || "",

  });

*const* [currentStep, setCurrentStep] = useState(0);

*const* handleNextStep = (*newData*, *final* = false, *file* = null) => {

    setData((*prev*) => ({ ...prev, ...newData }));

    if (final && post?.\_id && file) {

*const* formData = new FormData();

      formData.append("avatar", file);

      onEdit([post.\_id, newData, formData]);

    } else if (final && post?.\_id && !file) {

      onEdit([post.\_id, newData, false]);

    } else if (final && !post?.\_id) {

*const* formData = new FormData();

      formData.append("avatar", file);

      formData.append("name", newData.name);

      formData.append("birthday", newData.birthday);

      formData.append("breed", newData.breed);

      formData.append("comments", newData.comments);

      onEdit(formData);

    }

    setCurrentStep((*prev*) => prev + 1);

  };

*const* handlePrevStep = (*newData*) => {

    setData((*prev*) => ({ ...prev, ...newData }));

    setCurrentStep((*prev*) => prev - 1);

  };

*const* steps = [

    <StepOne *next*={handleNextStep} *data*={data} *onClose*={onClose} />,

    <StepTwo

*avatar*={post?.avatarURL}

*prev*={handlePrevStep}

*next*={handleNextStep}

*data*={data}

*onClose*={onClose}

    />,

  ];

  return <div>{steps[currentStep]}</div>;

};

Компонент працює і як *додавання* (коли post відсутній), і як *редагування* (коли передано post.\_id).

* Два кроки: StepOne — текстові поля (ім’я, порода тощо). StepTwo — завантаження фото + підтвердження. Перемикачем слугує currentStep, що зберігається в useState.
* Збирання FormData:
* У режимі *create* формується FormData зі всіма полями й файлом аватара, потім передається в onEdit.
* У режимі *edit* компонент відправляє масив [id, newData, formDataOrFalse], де formDataOrFalse — або об’єкт із новим файлом, або false, якщо фото не змінювали. Це дозволяє бекенду оновлювати лише ті поля, що справді змінилися.
* Навігація: handlePrevStep повертає користувача на попередній крок і зберігає заповнені дані; handleNextStep лише після останнього кроку викликає onEdit.
* Закриття модалки: обидва степи приймають onClose, тож форму можна згорнути на будь‑якому етапі.

У підсумку AddsPetForm — це компактний «мастер‑крокер», який керує локальним станом, ховає від батьків складну роботу з FormData й надає єдиний колбек onEdit як для створення, так і для оновлення картки тварини.

1. **ТЕСТУВАННЯ РОЗРОБЛЕНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ**

Процес тестування є важливою частиною життєвого циклу програмного забезпечення, оскільки саме на цьому етапі перевіряється відповідність реалізованого функціоналу початковим вимогам, стійкість системи до помилок та її готовність до використання кінцевими користувачами.

У рамках кваліфікаційної роботи бакалавра проєкту Petly було проведено різнорівневе тестування, яке охоплювало:

* модулі фронтенду та бекенду;
* API-запити;
* валідацію форм;
* інтеграцію з зовнішніми сервісами (Cloudinary, Google OAuth, SendGrid);
* взаємодію з MongoDB.

5.1 Застосовані типи тестування

Модульне тестування (Unit Testing)

Застосовувалось для перевірки окремих функцій, таких як:

* валідація даних користувача;
* хешування пароля (bcrypt);
* генерація JWT токену;
* парсинг параметрів запиту.

Тестування проводилось з використанням бібліотеки **Jest**. Було створено тести для перевірки:

* створення користувача без обов'язкових полів;
* генерації токену з невалідним JWT\_SECRET;
* виводу помилок авторизації при неправильному паролі.

Інтеграційне тестування:

Було важливим для перевірки взаємодії між модулями, наприклад:

* логін ↔ генерація токена ↔ запис в LocalStorage;
* створення оголошення ↔ збереження в базі ↔ оновлення UI.

Тестування проводилось вручну через Postman та автоматизовано за допомогою Swagger UI, що дозволяло тестувати всі ключові API безпосередньо у браузері.

Таблиця 5.1 — Порівняння аналогічних платформ (таблицю виканано самостійно)

|  |  |
| --- | --- |
| Сценарій | Очікуваний результат |
| Реєстрація | Успішне створення користувача, редірект на сторінку профілю |
| Логін | Отримання JWT токена, доступ до захищених сторінок |
| Додавання питомця | Відображення нового запису в особистому кабінеті |
| Додавання оголошення | Оголошення з’являється на /notices/own |
| Завантаження аватара | Новий avatarURL в базі та змінене зображення в інтерфейсі |

Функціональне тестування застосовувалось для перевірки ключових сценаріїв з точки зору користувача.

5.2 Тестування безпеки

Захист маршрутів:

* Усі маршрути, що вимагають авторизації (/user, /api/users/current, /api/users/avatar, /api/notices/own тощо), були протестовані на предмет недопустимого доступу без токена.
* При використанні неправильного або простроченого JWT користувач отримує відповідь: 401 Unauthorized.

Обробка неправильного введення:

Система коректно обробляє такі ситуації:

* спроба створити акаунт із уже зареєстрованим email;
* логін із неправильним паролем;
* відправка порожньої форми.

5.3 Тестування інтеграції зі сторонніми сервісами

* Cloudinary: було протестовано завантаження зображень у різних форматах (.jpg, .png, .webp). Перевірено, що зображення завантажуються в потрібну папку, оптимізуються та доступні за посиланням.
* SendGrid: при реєстрації на електронну пошту надсилається лист підтвердження. Також протестовано функцію «Забув пароль».
* Google OAuth: перевірено, що логін через Google працює, при цьому створюється новий користувач або знаходиться існуючий, після чого генерується токен і повертається на фронт.

5.4 Загальні результати тестування

Система Petly була протестована за понад 20 сценаріями взаємодії, включно з рідкісними кейсами (видалення оголошень, повторна авторизація, обмежений доступ). Тестування показало, що:

* Функціонал працює відповідно до вимог технічного завдання;
* Усі ключові дії користувача завершуються коректним результатом;
* Система стабільно працює під час мережевих затримок і повторних запитів;
* Дані в базі залишаються консистентними.

5.5 Тестування користувацького інтерфейсу (UI/UX)

Адаптивність:

Перевірено, що сайт коректно відображається:

* на мобільних телефонах (360px);
* планшетах (768px);
* десктопах (1920px).

Усі компоненти адаптивні, меню згортається в бургер-меню, форма не виходить за межі контейнера.

Повідомлення про помилки:

Всі поля форм реєстрації, логіну, додавання тварини та оголошення супроводжуються валідаційними повідомленнями (Formik + Yup) [11, 12]. У випадку помилки від бекенду відображається toast-повідомлення з детальним описом.

**ВИСНОВКИ**

У межах даної кваліфікаційної роботи бакалавра було реалізовано повноцінний вебзастосунок Petly, призначений для підтримки власників та волонтерів, які опікуються домашніми тваринами. Система забезпечує публікацію оголошень про пошук нової родини для улюбленців, а також зручну взаємодію між користувачами, притулками та ініціативами з захисту тварин.

У процесі проєктування та реалізації системи було досягнуто наступних результатів:

* Розроблено клієнт-серверну архітектуру на основі React (frontend) та Node.js + Express (backend) з використанням MongoDB Atlas для зберігання даних.
* Реалізовано адаптивний інтерфейс із підтримкою мобільних, планшетних та десктопних пристроїв.
* Створено систему авторизації через email/пароль і Google OAuth 2.0.
* Реалізовано особистий профіль користувача з можливістю редагування, додавання аватара та улюбленців.
* Побудовано систему оголошень із категоріями, фільтрацією, додаванням у «обране» та можливістю створення, редагування й видалення власних постів.
* Забезпечено інтеграцію з хмарними сервісами — Cloudinary (для обробки зображень) та SendGrid (для надсилання email-підтверджень).
* Розроблено набір REST API із документацією у Swagger UI. [14]
* Проведено всебічне тестування системи (модульне, функціональне, UI/UX, безпекове), яке підтвердило відповідність програмного забезпечення поставленим вимогам.
* Здійснено підготовку системи до хостингу в хмарі (Render, Vercel), що дозволяє забезпечити постійну доступність застосунку.

Таким чином, розроблений застосунок Petly відповідає сучасним вимогам до вебсистем такого типу: є зручним, безпечним, масштабованим та готовим до реального використання. Його структура дозволяє легко розширювати функціональність і адаптувати систему під потреби конкретних організацій або користувачів.

### ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Petfinder – платформа для пошуку тварин [Електронний ресурс] – URL: <https://www.petfinder.com> (Дата звернення: 15.06.2025 р.).
2. Adopt a Pet – онлайн сервіс пошуку домашніх улюбленців [Електронний ресурс] – URL: <https://adoptapet.com> (Дата звернення: 26.06.2025 р.).
3. MongoDB Atlas Documentation [Електронний ресурс] – URL:  
   <https://www.mongodb.com/docs/atlas> (Дата звернення: 10.06.2025 р.).
4. Mongoose ODM Documentation [Електронний ресурс] – URL:  
   <https://mongoosejs.com> (Дата звернення: 13.06.2025 р.).
5. Google OAuth 2.0 Documentation [Електронний ресурс] – URL:  
   https://developers.google.com/identity/protocols/oauth2 (Дата звернення: 20.06.2025 р.).
6. Cloudinary Documentation [Електронний ресурс] – URL:  
   https://cloudinary.com/documentation (Дата звернення: 06.06.2025 р.).
7. SendGrid API Documentation [Електронний ресурс] – URL:  
   https://docs.sendgrid.com (Дата звернення: 13.06.2025 р.).
8. Node.js Documentation [Електронний ресурс] – URL:  
   https://nodejs.org/en/docs (Дата звернення: 04.06.2025 р.).
9. React Documentation [Електронний ресурс] – URL:  
   <https://reactjs.org> (Дата звернення: 05.06.2025 р.).
10. Express.js – Fast, unopinionated, minimalist web framework [Електронний ресурс] – URL: <https://expressjs.com> (Дата звернення: 20.06.2025 р.).
11. Yup – JavaScript schema validation [Електронний ресурс] – URL:  
    <https://github.com/jquense/yup> (Дата звернення: 15.06.2025 р.).
12. Formik – Forms in React, without the tears [Електронний ресурс] – URL: <https://formik.org> (Дата звернення: 20.06.2025 р.).
13. Redux Toolkit Documentation [Електронний ресурс] – URL:  
    https://redux-toolkit.js.org (Дата звернення: 11.06.2025 р.).
14. Swagger UI [Електронний ресурс] – URL:  
    https://swagger.io/tools/swagger-ui (Дата звернення: 02.06.2025 р.).
15. GitHub – Репозиторій проєкту “Petly” [Електронний ресурс] – URL: [https://github.com/kuznietsov-maksym-nure/2025\_B\_PI\_PZPI-23\_1\_Kuznietsov \_M\_O/](https://github.com/kuznietsov-maksym-nure/2025_B_PI_PZPI-23_1_Kuznietsov%20_M_O/) (Дата звернення: 21.06.2025 р.).

ДОДАТОК А

Звіт результатів перевірки на унікальність тексту в мережі інтернет та базі



Рисунок А.1 – Сторінка 1 звіту про унікальність тексту

ДОДАТОК Б

Лістинг програмного коду

*const* { Notices } = require("../../db/noticesModel");

*const* { WrongParametersError } = require("../../helpers/errors");

*const* getNoticesByCategoryService = *async* (*category*, *search*, *page*, *limit*) => {

  limit = +limit > 8 ? 8 : +limit;

  page = +page;

*const* searchObj = {};

  if (["sell", "lost-found", "in-good-hands"].includes(category)) {

    searchObj.categoryName = category;

    if (search) searchObj.title = { $regex: search, $options: "i" };

*const* dataLength = await Notices.countDocuments(searchObj);

*const* data = await Notices.find(searchObj)

      .skip((page - 1) \* limit)

      .limit(limit)

      .sort({ updatedAt: -1 })

      .select(["-createdAt", "-updatedAt"]);

    return {

      data,

      perPage: limit,

      total: dataLength,

      noticesLeft:

        dataLength - page \* limit > 0 ? dataLength - page \* limit : 0,

      pageCount: Math.ceil(dataLength / +limit),

      currentPage: page,

      noticesOnPage: data.length,

    };

  }

*const* data = await Notices.findById(category);

  if (!data) throw new WrongParametersError("Not found");

  return {

    data,

  };

};

*module*.*exports* = { getNoticesByCategoryService };

*const* fs = require("fs");

*const* cloudinary = require("cloudinary").v2;

cloudinary.config({

  cloud\_name: process.env.CLOUD\_NAME,

  api\_key: process.env.CLOUD\_KEY,

  api\_secret: process.env.CLOUD\_SECRET,

});

*const* petPhotoUpload = *async* (*file*, *folder*, *filename*) => {

*const* cloudFilePath = folder + "/" + filename;

*const* options = {

    public\_id: cloudFilePath + Math.floor(Math.random() \* 10000),

    overwrite: true,

  };

  console.log("public\_id:", options.public\_id);

  try {

*const* { url } = await cloudinary.uploader.upload(file, options);

    await fs.unlinkSync(file);

    return {

      message: "Upload success",

      url,

    };

  } catch (error) {

    fs.unlinkSync(file);

    return {

      message: "Upload fail",

      error,

    };

  }

};

*module*.*exports* = { petPhotoUpload };

*const* jsonwebtoken = require("jsonwebtoken");

*const* { NotAuthorizedError } = require("../helpers/errors");

*const* { User } = require("../db/userModel");

*const* authMiddleware = *async* (*req*, *res*, *next*) => {

  try {

    if (!req.headers.authorization)

      next(

        new NotAuthorizedError(

          "Please, provide a token in request authorization header"

        )

      );

*const* [, token] = req.headers.authorization.split(" ");

    if (!token || !jsonwebtoken.verify(token, process.env.JWT\_SECRET))

      next(new NotAuthorizedError("Please, provide a token"));

*const* user = jsonwebtoken.verify(token, process.env.JWT\_SECRET);

*const* findedUser = await User.findById(user.\_id);

    if (!findedUser) next(new NotAuthorizedError("Not authorized"));

    if (token !== findedUser.token)

      next(new NotAuthorizedError("Not authorized"));

    req.token = token;

    req.user = findedUser;

    next();

  } catch (error) {

    next(new NotAuthorizedError("Not authorized"));

  }

};

*module*.*exports* = { authMiddleware };

import { useEffect } from "react";

import { useDispatch, useSelector } from "react-redux";

import { Route, Routes } from "react-router-dom";

import { Loader } from "./components/Loader/Loder";

import PublicRoute from "./components/PublicRoute/PublicRoute";

import PrivateRoute from "./components/PrivateRoute/PrivateRoute";

import SharedLayout from "./components/SharedLayout/SharedLayout";

import { GlobalStyle } from "./GlobalStyles";

import HomePage from "./pages/HomePage/HomePage";

import LoginPage from "./pages/LoginPage/LoginPage";

import NewsPage from "./pages/NewsPage/NewsPage";

import NoticesPage from "./pages/NoticesPage/NoticesPage";

import OurFriendsPage from "./pages/OurFriendsPage/OurFriendsPage";

import RegisterPage from "./pages/RegisterPage/RegisterPage";

import UserPage from "./pages/UserPage/UserPage";

import { getUser } from "./redux/auth/auth-operations";

import { selectIsLoading } from "./redux/auth/auth-selectors";

import NotFound from "./components/NotFound/NotFound";

import NoticesPageContent from "./components/NoticesPageContent/NoiticesPageContent";

*const* App = () => {

*const* dispatch = useDispatch();

*const* isLoading = useSelector(selectIsLoading);

  useEffect(() => {

    dispatch(getUser());

  }, [dispatch]);

  return (

    <>

      <Loader *isLoading*={isLoading} />

      <GlobalStyle />

      <Routes>

        <Route *path*="/" *element*={<SharedLayout />}>

          <Route *index* *element*={<HomePage />} />

          <Route

*path*="/register"

*element*={

              <PublicRoute *redirectTo*="/user">

                <RegisterPage />

              </PublicRoute>

            }

          />

          <Route

*path*="/login"

*element*={

              <PublicRoute *redirectTo*="/user">

                <LoginPage />

              </PublicRoute>

            }

          />

          <Route

*path*="/user"

*element*={

              <PrivateRoute>

                <UserPage />

              </PrivateRoute>

            }

          />

          <Route *path*="/news" *element*={<NewsPage />} />

          <Route *path*="/friends" *element*={<OurFriendsPage />} />

          <Route *path*="/user" *element*={<UserPage />} />

          <Route *path*="/notices" *element*={<NoticesPage />}>

            <Route *index* *element*={<NoticesPageContent />} />

            <Route *path*=":category" *element*={<NoticesPageContent />}>

              <Route *path*=":id" />

            </Route>

          </Route>

          <Route *path*="\*" *element*={<NotFound />} />

        </Route>

      </Routes>

    </>

*// </BrowserRouter>*

  );

};

export default App;

import PropTypes from "prop-types";

import { DeletePetBtn } from "../../DeletePetBtn/DeletePetBtn";

import {

  WrapperPicDiv,

  Box,

  Description,

  InfoPet,

  BoxPetsList,

  BoxScrollbar,

  EditPetBtn,

} from "./PetsList.styled";

import defImage from "../../../img/defaultImg.jpeg";

import { ReactComponent as Edit } from "../../../assets/svg/penciNotices.svg";

export *const* PetsList = ({ *pets*, *onPostHandler*, *removeHandler* }) => {

  return (

    <BoxPetsList>

      <BoxScrollbar>

        {pets?.map((*el*) => {

*const* { avatarURL, birthday, breed, comments, name, \_id } = el;

          return (

            <Box *key*={\_id}>

              <WrapperPicDiv>

                <img *src*={avatarURL ? avatarURL : defImage} *alt*={name} />

              </WrapperPicDiv>

              <Description>

                <DeletePetBtn *name*={\_id} *onClick*={removeHandler} />

                <EditPetBtn *name*={\_id} *onClick*={onPostHandler}>

                  <Edit *width*={30} *height*={30} />

                </EditPetBtn>

                <InfoPet>

                  <b> Name:</b> {name ? name : "No Info"}

                </InfoPet>

                <InfoPet>

                  <b> Date of birth:</b> {birthday ? birthday : "No Info"}

                </InfoPet>

                <InfoPet>

                  <b>Breed:</b> {breed ? breed : "No Info"}

                </InfoPet>

                <InfoPet>

                  <b> Comments:</b> {comments ? comments : "No Info"}

                </InfoPet>

              </Description>

            </Box>

          );

        })}

      </BoxScrollbar>

    </BoxPetsList>

  );

};

PetsList.propTypes = {

  pets: PropTypes.array.isRequired,

};

import { createContext, useContext, useState } from "react";

*const* NoticesContext = createContext();

export *const* useNoticesParams = () => useContext(NoticesContext);

export *const* NoticesProvider = ({ *children* }) => {

*const* [page, setPageParams] = useState(1);

*const* [search, setSearchParams] = useState("");

*const* setPage = (*value*) => {

    setPageParams(value);

  };

*const* setSearch = (*value*) => {

    setSearchParams(value);

  };

  return (

    <NoticesContext.Provider *value*={{ setPage, page, setSearch, search }}>

      {children}

    </NoticesContext.Provider>

  );

};

ДОДАТОК В

Слайди презентації

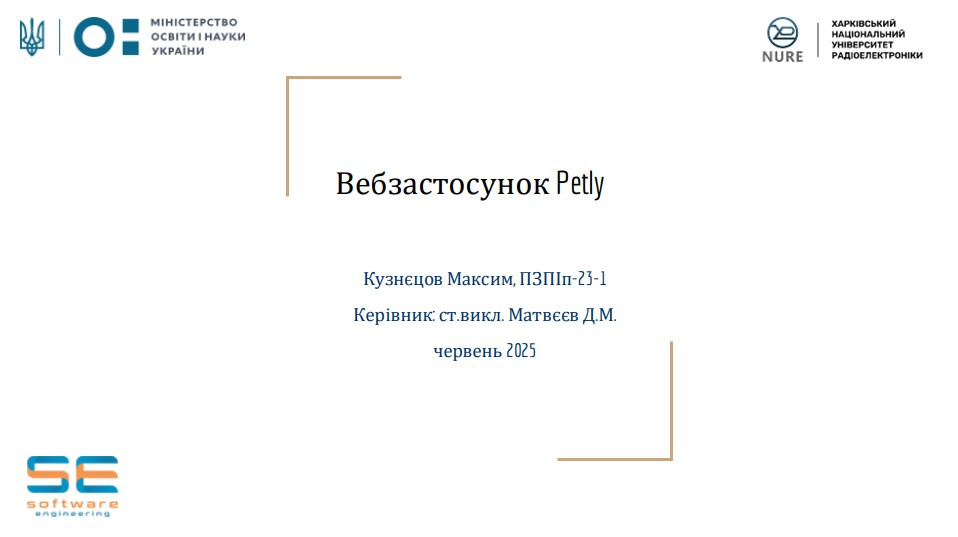


Рисунок В.1 – Слайд 1

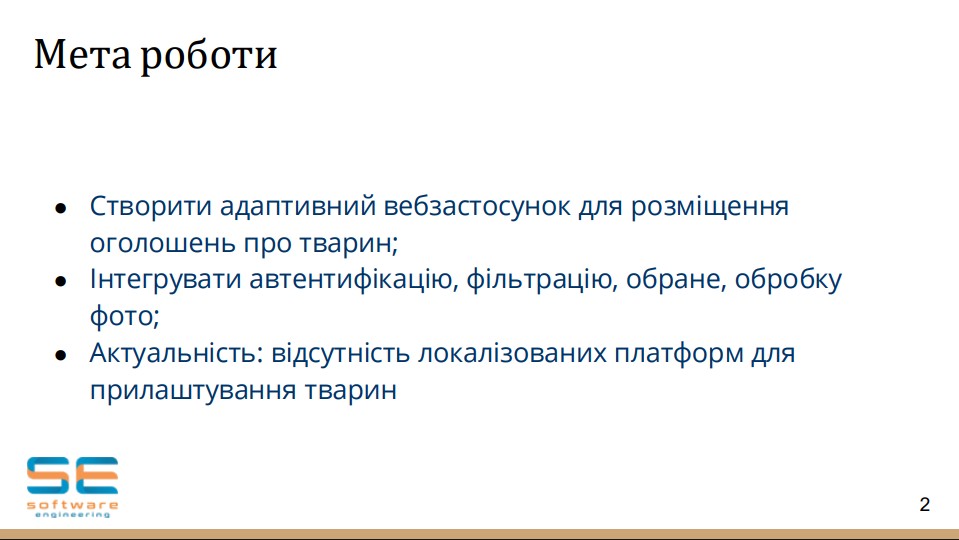


Рисунок В.2 – Слайд 2

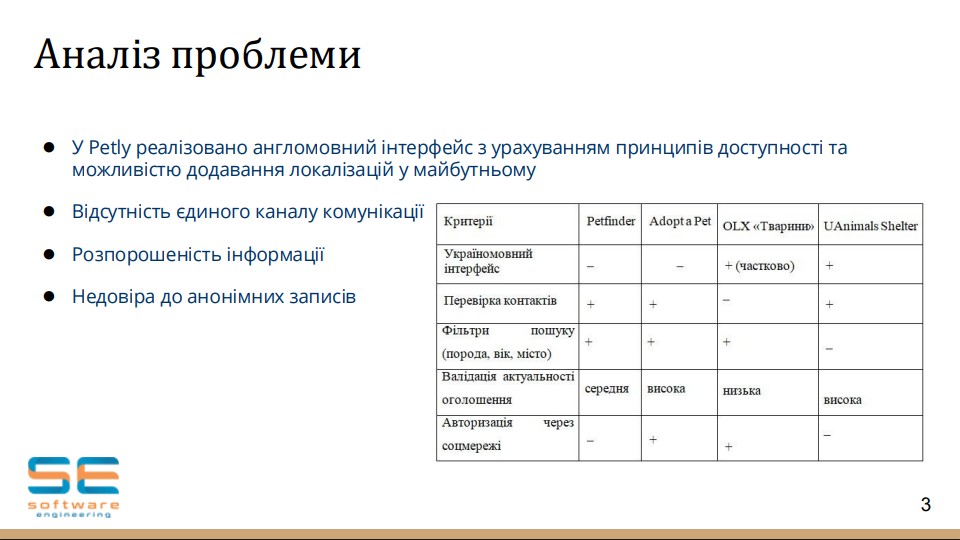


Рисунок В.3 – Слайд 3

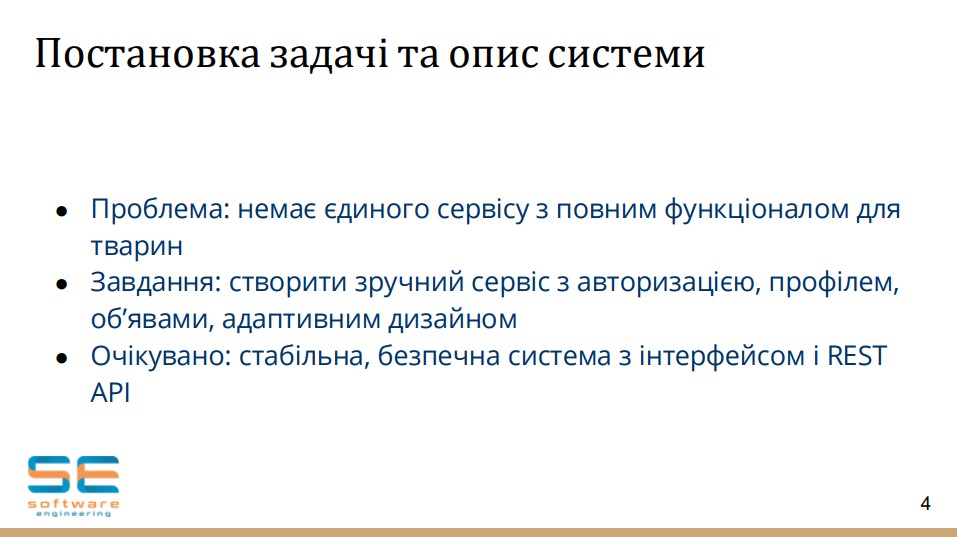


Рисунок В.4 – Слайд 4

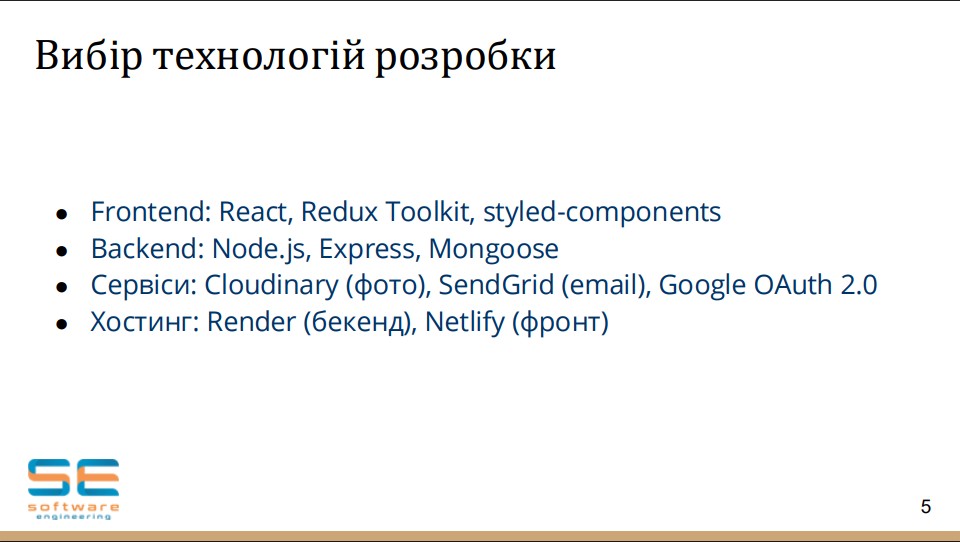


Рисунок В.5 – Слайд 5

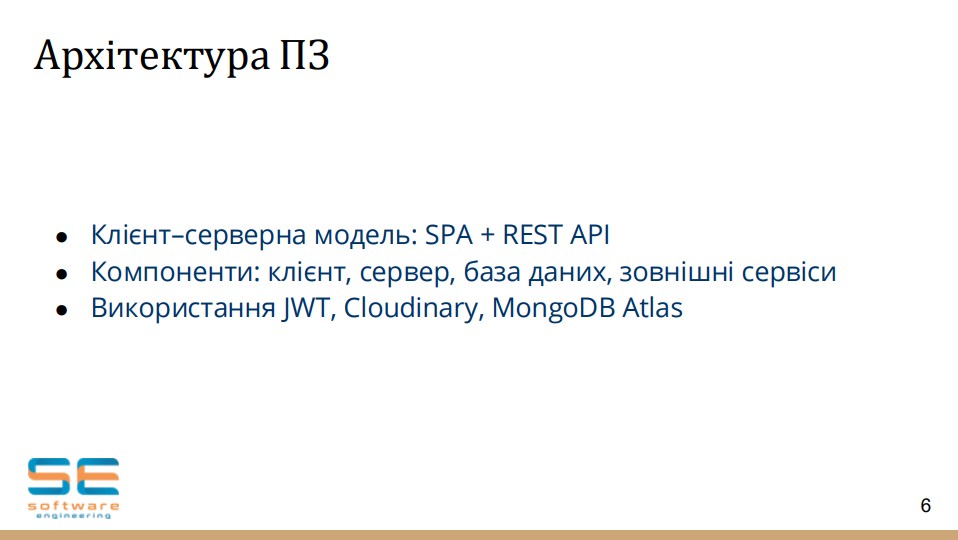


Рисунок В.6 – Слайд 6

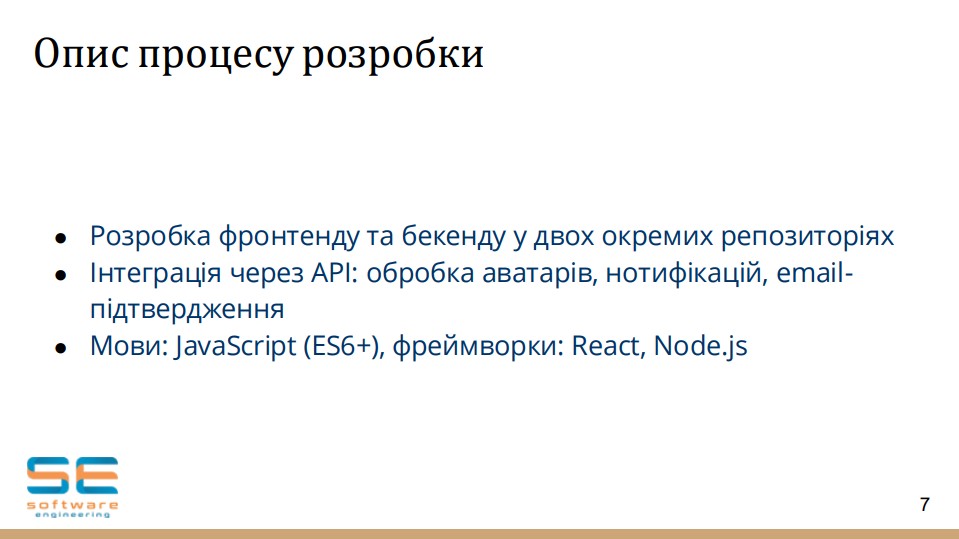


Рисунок В.7 – Слайд 7

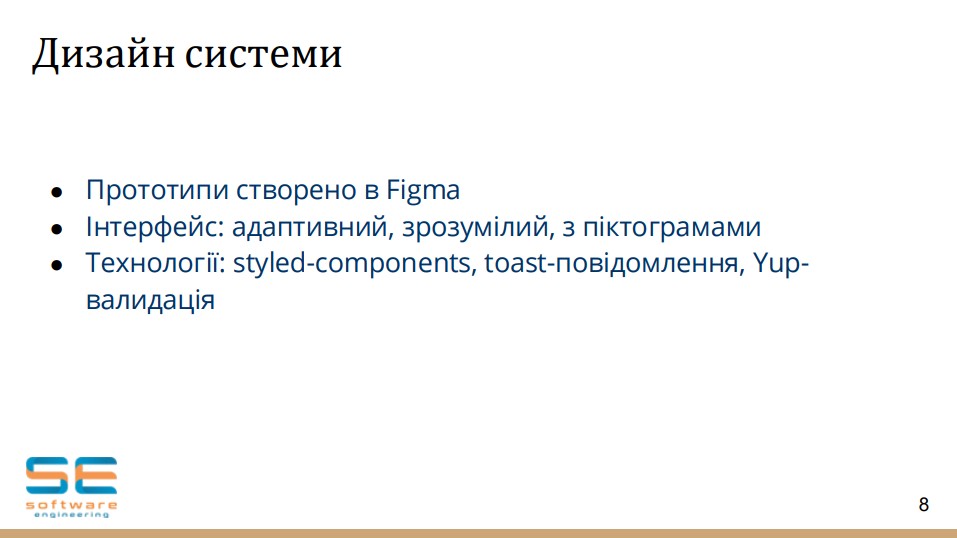


Рисунок В.8 – Слайд 8



Рисунок В.9 – Слайд 9



Рисунок В.10 – Слайд 10

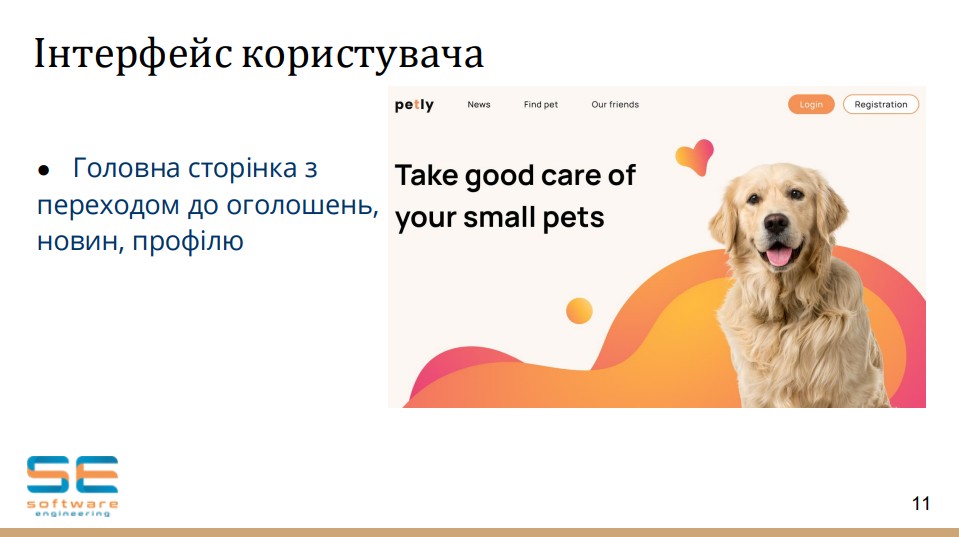


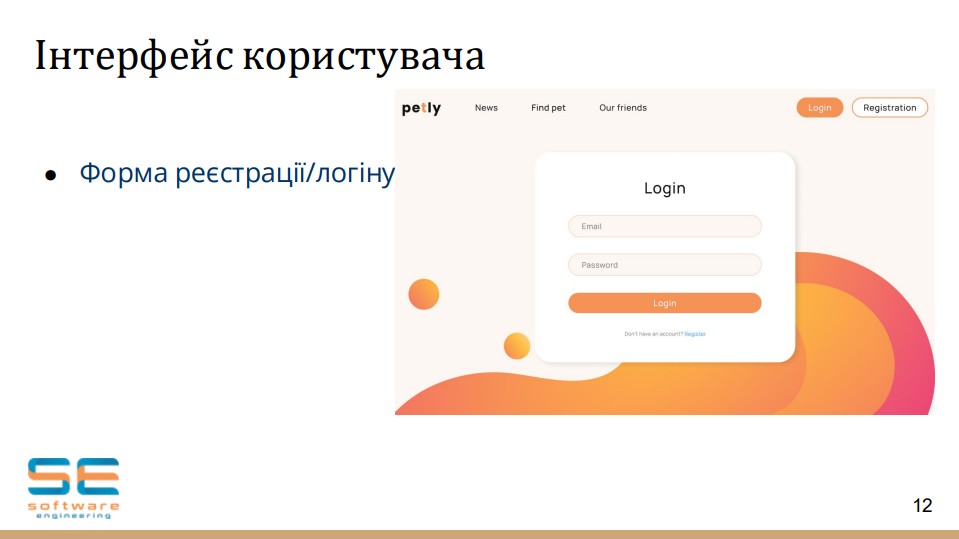
Рисунок В.11 – Слайд 11****

Рисунок В.12 – Слайд 12

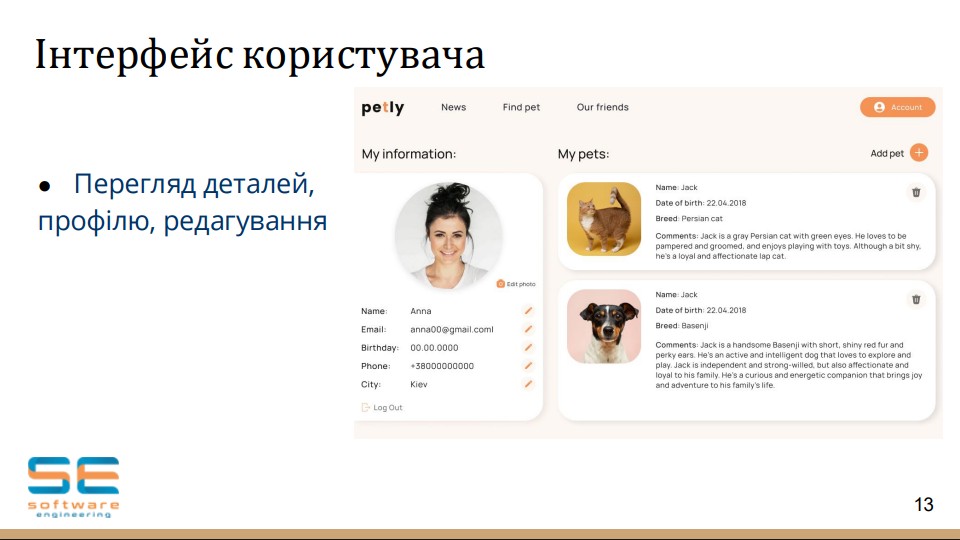


Рисунок В.13 – Слайд 13



Рисунок В.14 – Слайд 14

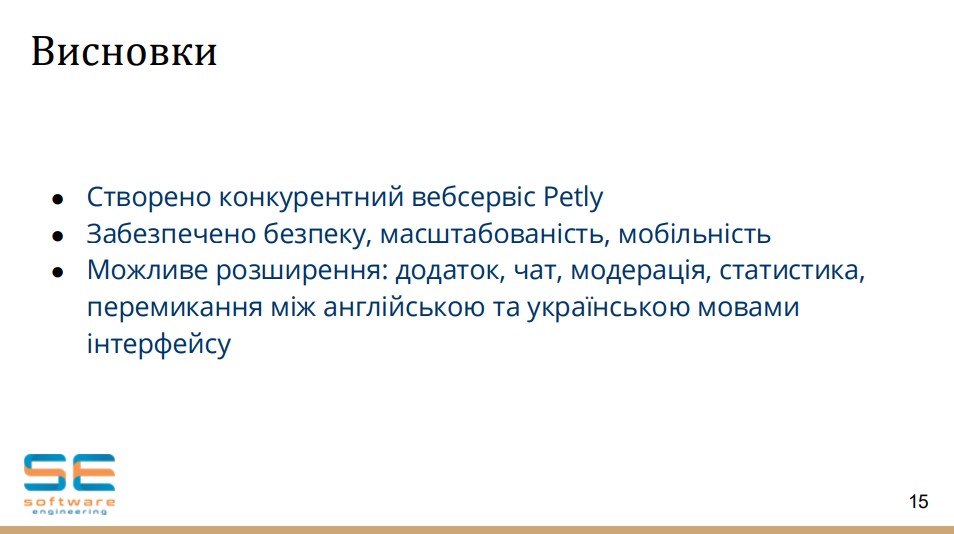


Рисунок В.15 – Слайд 15