НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
(повна назва інституту/факультету)

Кафедра інформатики та програмної інженерії
(повна назва кафедри)

«До захисту допущено»
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2025 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інженерія програмного забезпечення інформаційних систем»

спеціальності «121 Інженерія програмного забезпечення»

на тему: Вебзастосунок для раннього виявлення хвороб домашніх тварин за допомогою ШІ та інтерактивного консультування з ветеринаром

Виконав студент IV курсу, групи ІП-15
(шифр групи)
Дацьо Іван Іванович _____
(прізвище, ім'я, по батькові) (підпис)

Керівник асистент, Очеретяний О.К. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант асистент, Шулькевич Т.В. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент старший викладач, Алещенко О. В. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.

Студент _____
(підпис)

Київ – 2025

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 Інженерія програмного забезпечення

Освітньо-професійна програма – Інженерія програмного забезпечення інформаційних систем

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2025 р.

ЗАВДАННЯ
на дипломний проєкт студенту

Дацю Івану Івановичу
(прізвище, ім'я, по батькові)

1. Тема проєкту Вебзастосунок для раннього виявлення хвороб домашніх тварин за допомогою ШІ та інтерактивного консультування з ветеринаром

керівник проєкту Очеретяний Олександр Костянтинович, асистент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «23» травня 2025 р. №1705-с

2. Термін подання студентом проєкту «16» червня 2025 року

3. Вихідні дані до проєкту: технічне завдання

4. Зміст пояснювальної записки
1) Передпроєктне обстеження предметної області: постановка завдання дипломного проєктування, аналіз предметної області, аналіз існуючих рішень (включаючи відомі програмні продукти, алгоритмічні та технічні рішення), опис бізнес-процесів.
2) Розроблення вимог до програмного забезпечення: варіанти використання програмного забезпечення, розроблення функціональних та нефункціональних вимог, аналіз системних вимог, аналіз економічних показників програмного забезпечення, постановка завдання на розробку програмного забезпечення.
3) Конструювання та розроблення програмного забезпечення: архітектура програмного забезпечення, архітектурні рішення та обґрунтування вибору засобів розробки, конструювання програмного забезпечення (включаючи опис структури бази даних), аналіз безпеки даних.

4) Аналіз якості та тестування програмного забезпечення: аналіз якості ПЗ, опис процесів тестування, опис контрольного прикладу.
5) Розгортання та супровід програмного забезпечення: розгортання програмного забезпечення, супровід програмного забезпечення.

5. Перелік графічного матеріалу
1) Схема структурна варіантів використань
2) Схема структурна компонентів програмного забезпечення
3) Схема бази даних
4) BPMN-діаграма проведення онлайн-консультації

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання «15» березня 2025 року

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Вивчення рекомендованої літератури	15.03.2025	
2	Аналіз існуючих методів розв’язання задачі	05.04.2025	
3	Постановка та формалізація задачі	11.04.2025	
4	Розробка інформаційного забезпечення	14.04.2025	
5	Алгоритмізація задачі	21.04.2025	
6	Обґрунтування вибору використаних технічних засобів	22.04.2025	
7	Розробка програмного забезпечення	10.05.2025	
8	Налагодження програми	15.05.2025	
9	Виконання графічних документів	16.05.2025	
10	Оформлення пояснювальної записки	27.05.2025	
11	Подання ДП на попередній захист	04.06.2025	
12	Подання ДП рецензенту	10.06.2025	
13	Подання ДП на основний захист	16.06.2025	

Студент _____ Іван ДАЦЬО
(підпис) (ініціали, прізвище)

Керівник _____ Олександр ОЧЕРЕТЯНИЙ
(підпис) (ініціали, прізвище)

АНОТАЦІЯ

Пояснювальна записка дипломного проєкту складається з п’яти розділів, містить 48 таблиць, 28 рисунків та 36 джерел – загалом 112 сторінки.

Дипломний проєкт присвячений розробці вебзастосунку для раннього виявлення хвороб домашніх тварин за допомогою штучного інтелекту та інтерактивного консультування з ветеринаром.

Метою розробки є підвищення якості та доступності ветеринарної допомоги для власників домашніх тварин шляхом створення інструменту для попередньої діагностики стану здоров'я їхніх улюбленців та забезпечення зручного онлайн-зв'язку з кваліфікованими ветеринарними фахівцями.

У розділі 1 здійснено передпроєктне обстеження предметної області, включаючи аналіз актуальності, існуючих програмних продуктів, алгоритмічних та технічних рішень, а також описано ключові бізнес-процеси та сформульовано постановку завдання проєктування.

У розділі 2 розглянуто розроблення вимог до програмного забезпечення: визначено варіанти використання, розроблено функціональні та нефункціональні вимоги, проаналізовано системні вимоги, економічні показники та сформульовано постановку завдання на розробку.

Розділ 3 присвячений конструюванню та розробленню програмного забезпечення. Представлено архітектуру програмного забезпечення, обґрунтовано вибір засобів розробки, детально описано структуру бази даних та проведено аналіз безпеки даних.

У розділі 4 проаналізовано якість програмного забезпечення за допомогою статичного аналізу коду, описано процеси мануального тестування ключових функцій та наведено контрольний приклад роботи вебзастосунку.

У розділі 5 описано процеси розгортання програмного забезпечення на хмарних платформах та основні аспекти його подальшого супроводу, включаючи оновлення, моніторинг та виправлення помилок.

Програмне забезпечення впроваджено з використанням хмарних сервісів Vercel для основного застосунку, Render для сервісу ШІ та Supabase для бази даних PostgreSQL і файлового сховища.

КЛЮЧОВІ СЛОВА: ВЕБЗАСТОСУНОК, ТЕЛЕВЕТЕРИНАРІЯ, ШТУЧНИЙ ІНТЕЛЕКТ, ДІАГНОСТИКА ХВОРОБ ТВАРИН, ОНЛАЙН-КОНСУЛЬТАЦІЇ, NEXT.JS, TYPESCRIPT, PYTHON, FLASK, POSTGRESQL, SOCKET.IO, SUPABASE, VERCEL, PRISMA.

ABSTRACT

The explanatory note of the diploma project consists of five sections, contains 48 tables, 28 figures and 36 sources – in total 112 pages.

The diploma project is dedicated to the development of a web application for early detection of pet diseases using artificial intelligence and interactive consultation with a veterinarian.

The purpose of the development is to improve the quality and accessibility of veterinary care for pet owners by creating a tool for preliminary health diagnostics of their pets and providing convenient online communication with qualified veterinary professionals.

In chapter 1, the analysis of the subject area is carried out, including the analysis of relevance, existing software products, algorithmic and technical solutions; business processes are described, and the task statement is formulated.

Chapter 2 discusses the development of software requirements. Use cases are defined, functional and non-functional requirements are developed, system requirements and economic indicators are analyzed, and the development task is formulated.

Chapter 3 focuses on software design and development. The software architecture is presented, the choice of development tools is justified, the database structure is described in detail, and data security analysis is carried out.

Chapter 4 analyzes the quality of the software using static code analysis, describes the process of manual testing of key features, and provides a control example of the web application in operation.

Chapter 5 describes the deployment of the software on cloud platforms and the main aspects of its further maintenance, including updates, monitoring, and bug fixing.

The software is implemented using Vercel for the main application, Render for the AI service, and Supabase for the PostgreSQL database and file storage.

KEYWORDS: WEB APPLICATION, TELE-VETERINARY, ARTIFICIAL INTELLIGENCE, PET DISEASE DIAGNOSTICS, ONLINE CONSULTATIONS, NEXT.JS, TYPESCRIPT, PYTHON, FLASK, POSTGRESQL, SOCKET.IO, SUPABASE, VERCEL, PRISMA.

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	КПІ.ІП-1507.045440.01.91	Технічне завдання	21	
3	А4	КПІ.ІП-1507.045440.02.81	Пояснювальна записка	112	
4	А4	КПІ.ІП-1507.045440.03.12	Текст програми	30	
5	А4	КПІ.ІП-1507.045440.04.51	Програма та методика тестування	6	
6	А4	КПІ.ІП-1507.045440.05.34	Керівництво користувача	21	
7	А3	КПІ.ІП-1507.045440.06.99	Графічний матеріал	4	

Змін.	Арк.	№ докум.	Підп.	Дата	КПІ.ІП-1507.045440.00.90		
Розроб.		Дацьо І.І.			Відомість дипломного проєкту	Літ.	Аркуш 1 / Аркушів 1
Перевір.		Очеретяний О.К.					
Н.контр.		Шулькевич Т.В.				КПІ ім. Ігоря Сікорського ФІОТ каф. ІПІ гр. ІП-15	
Затв.		Жаріков Е.В.					

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”

Завідувач кафедри

_____ Едуард ЖАРІКОВ

“___” _____ 2025 р.

ВЕБЗАСТОСУНОК ДЛЯ РАННЬОГО ВИЯВЛЕННЯ ХВОРОБ ДОМАШНІХ ТВАРИН ЗА ДОПОМОГОЮ ШІ ТА ІНТЕРАКТИВНОГО КОНСУЛЬТУВАННЯ З ВЕТЕРИНАРОМ

Технічне завдання

КПІ.ІП-1507.045440.01.91

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Олександр ОЧЕРЕТЯНИЙ

Нормоконтроль:
_____ Тетяна ШУЛЬКЕВИЧ

Виконавець:
_____ Іван ДАЦЬО

Київ – 2025

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: Вебзастосунок для раннього виявлення хвороб домашніх тварин за допомогою ШІ та інтерактивного консультування з ветеринаром.

Галузь застосування:

Наведене технічне завдання поширюється на розробку вебзастосунку «Vai» [045440], котра використовується для раннього виявлення захворювань домашніх тварин за допомогою штучного інтелекту та забезпечення інтерактивного онлайн-консультування з ветеринарними фахівцями. Програмне забезпечення призначене для використання власниками домашніх тварин для попередньої оцінки стану здоров’я своїх улюбленців та ветеринарними практиками для оптимізації процесу прийому пацієнтів і надання дистанційних консультаційних послуг, що сприятиме цифровізації ветеринарної медицини та підвищенню оперативності діагностики.

2 ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки вебзастосунку для раннього виявлення хвороб домашніх тварин за допомогою ШІ та інтерактивного консультування з ветеринаром є завдання на дипломне проєктування, затверджене кафедрою інформатики та програмної інженерії Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

3 ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для забезпечення можливості власникам домашніх тварин проводити попередню діагностику стану здоров'я своїх улюбленців за допомогою аналізу симптомів засобами штучного інтелекту, а також отримувати інтерактивні онлайн-консультації з кваліфікованими ветеринарними спеціалістами.

Метою розробки є підвищення якості та доступності ветеринарної допомоги через раннє виявлення захворювань за допомогою штучного інтелекту та спрощення процесу отримання онлайн-консультацій, сприяючи цифровізації ветеринарної медицини та загального рівня догляду за домашніми тваринами.

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вимоги до функціональних характеристик

Програмне забезпечення повинно забезпечувати виконання наступних основних функції:

4.1.1 Користувацького інтерфейсу

– відображення форм реєстрації та авторизації користувачів;

– надання персональної панелі керування (дашборду) з оглядом ключової інформації, адаптованої до ролі користувача (власник тварини або ветеринар)

– забезпечення інтерфейсу для управління персональними даними профілю користувача;

– реалізація інтерфейсу для додавання, перегляду, редагування та видалення інформації про домашніх тварин;

– надання інтерактивного інтерфейсу для вибору симптомів тварини та відображення результатів їх аналізу за допомогою модуля штучного інтелекту;

– забезпечення інтерфейсу для пошуку ветеринарних фахівців з можливістю застосування фільтрів та перегляду їхніх детальних професійних профілів;

– реалізація інтерфейсу для бронювання та перегляду доступних часових слотів для онлайн-консультацій;

– надання інтерактивного інтерфейсу чату для проведення онлайн-консультацій у реальному часі з можливістю обміну текстовими повідомленнями та зображеннями;

– забезпечення інтерфейсу для керування робочим розкладом ветеринарів;

– відображення системних сповіщень та нагадувань про важливі події.

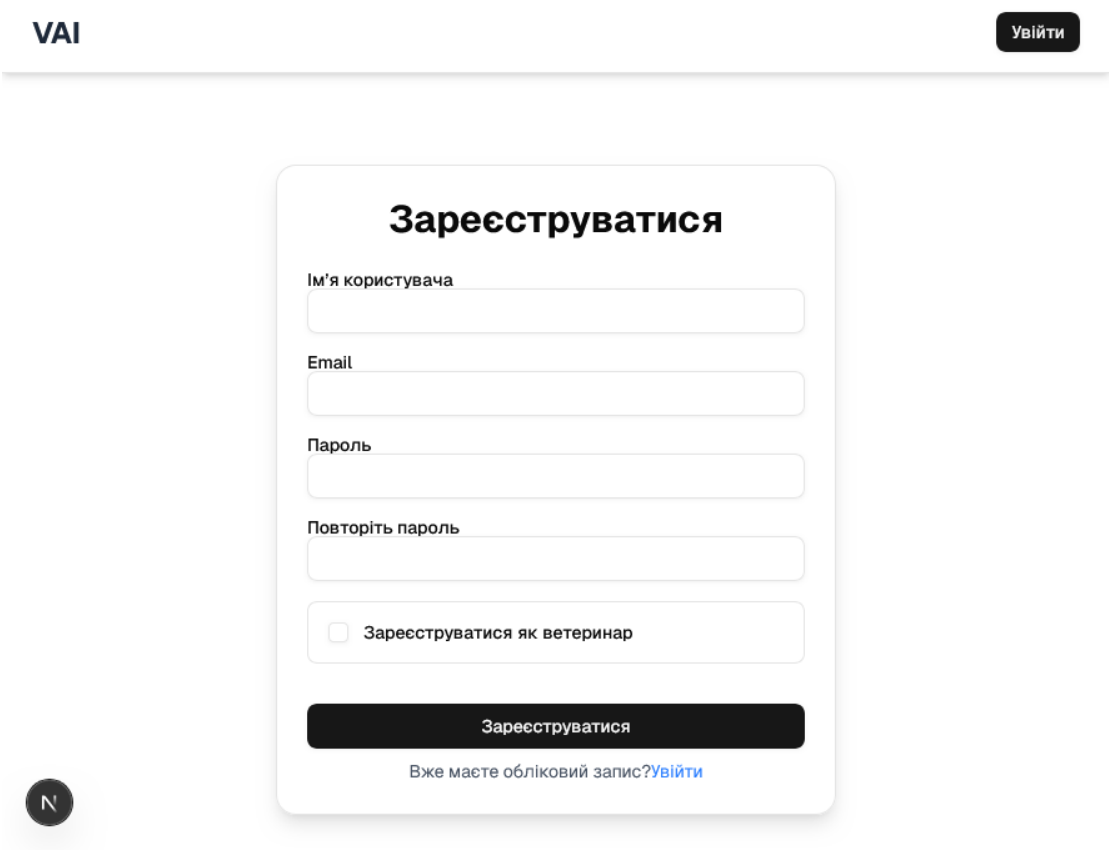


Рисунок 4.1 – Екранна форма сторінки реєстрації

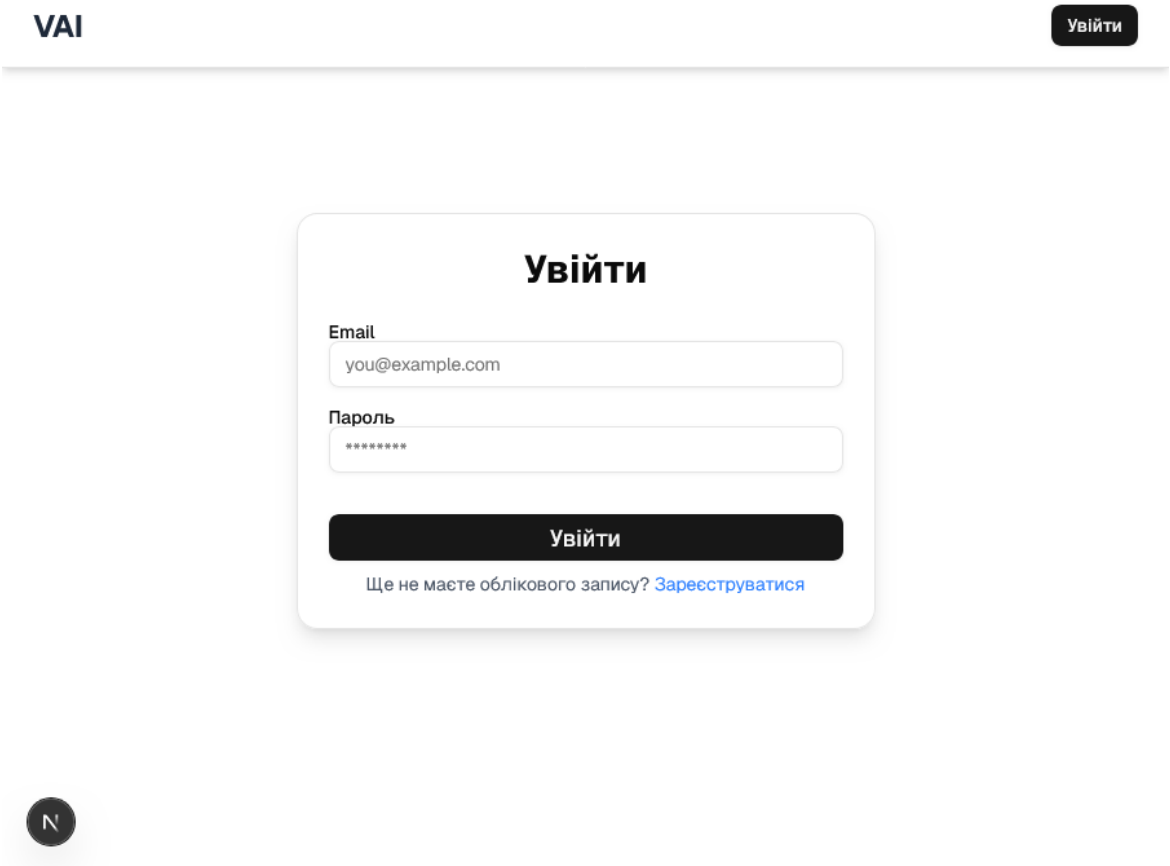


Рисунок 4.2 – Екранна форма сторінки авторизації

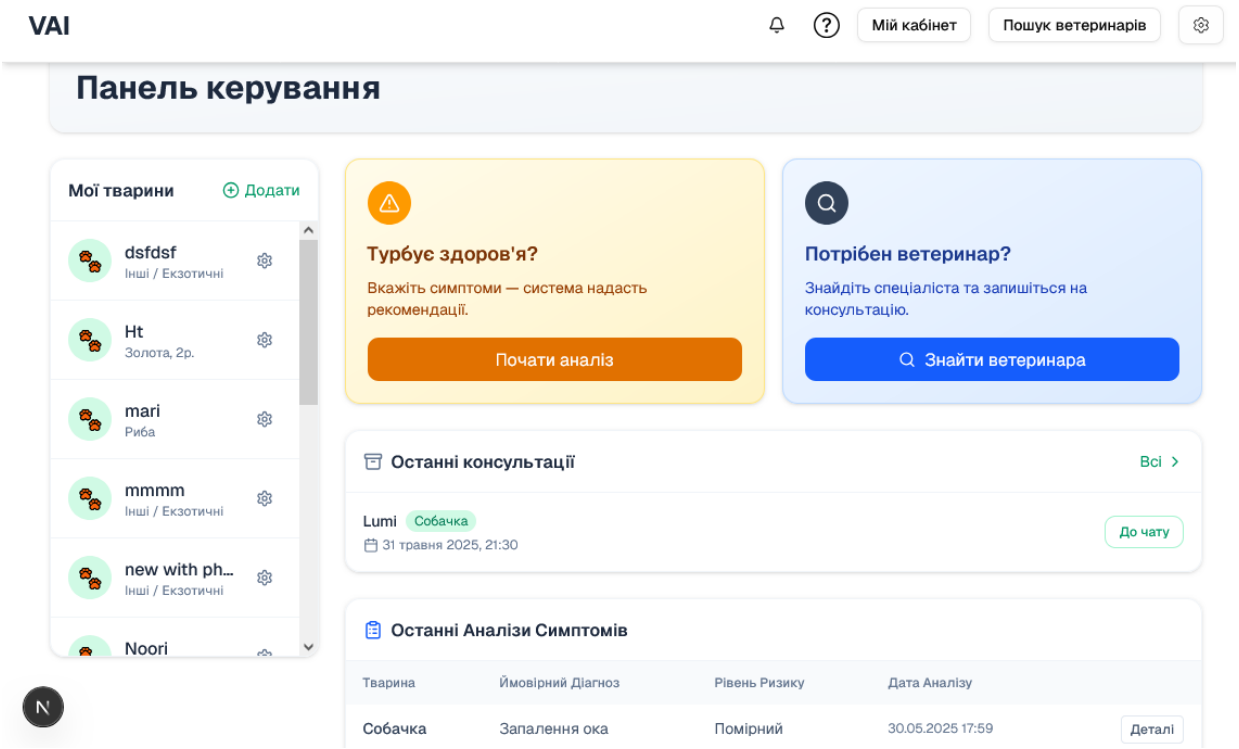


Рисунок 4.3 – Екранна форма панелі керування власника тварини

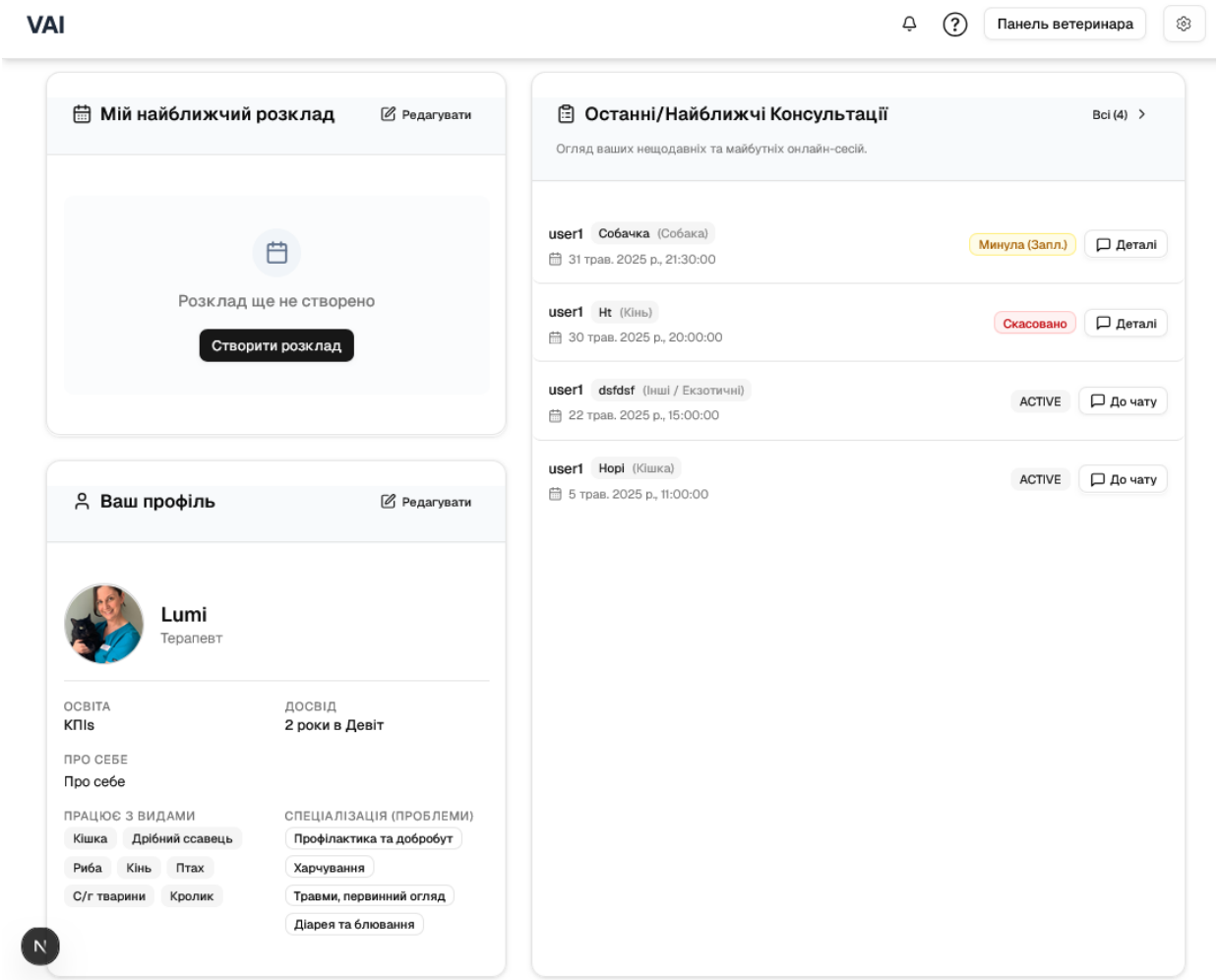


Рисунок 4.4 – Екранна форма панелі керування ветеринара

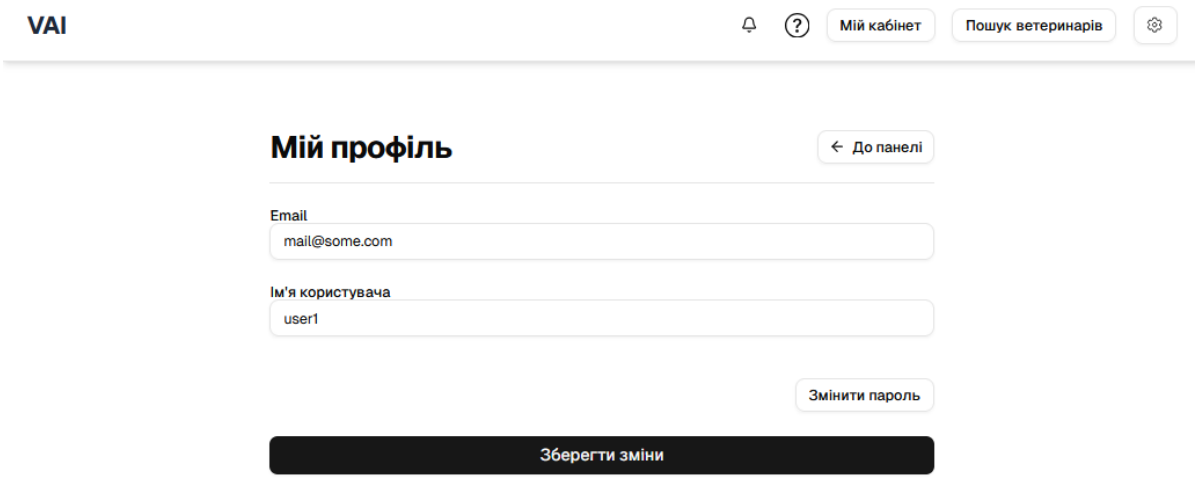


Рисунок 4.5 – Екранна форма сторінки керування профілем користувача

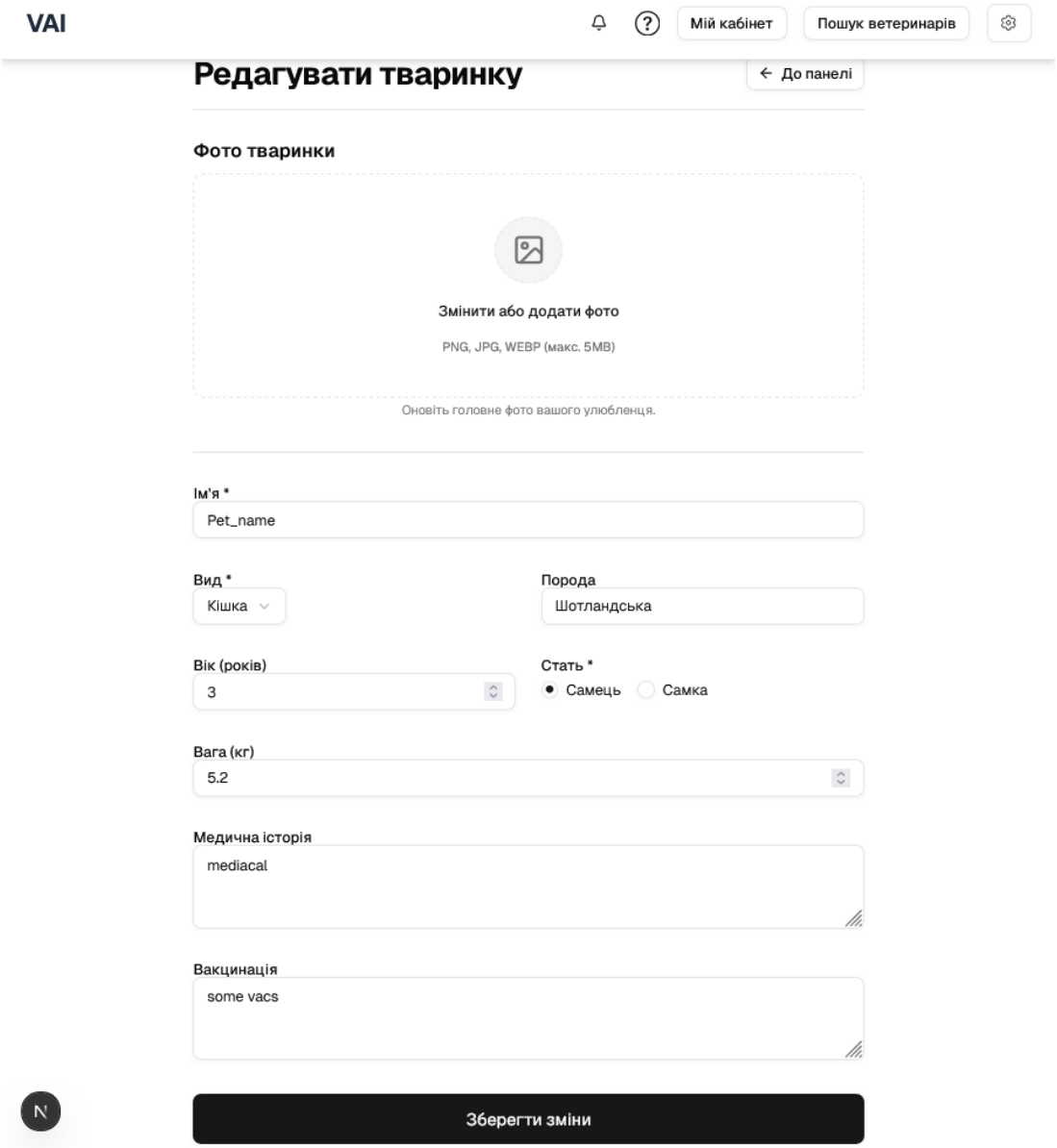


Рисунок 4.6 – Екранна форма сторінки управління домашніми тваринами

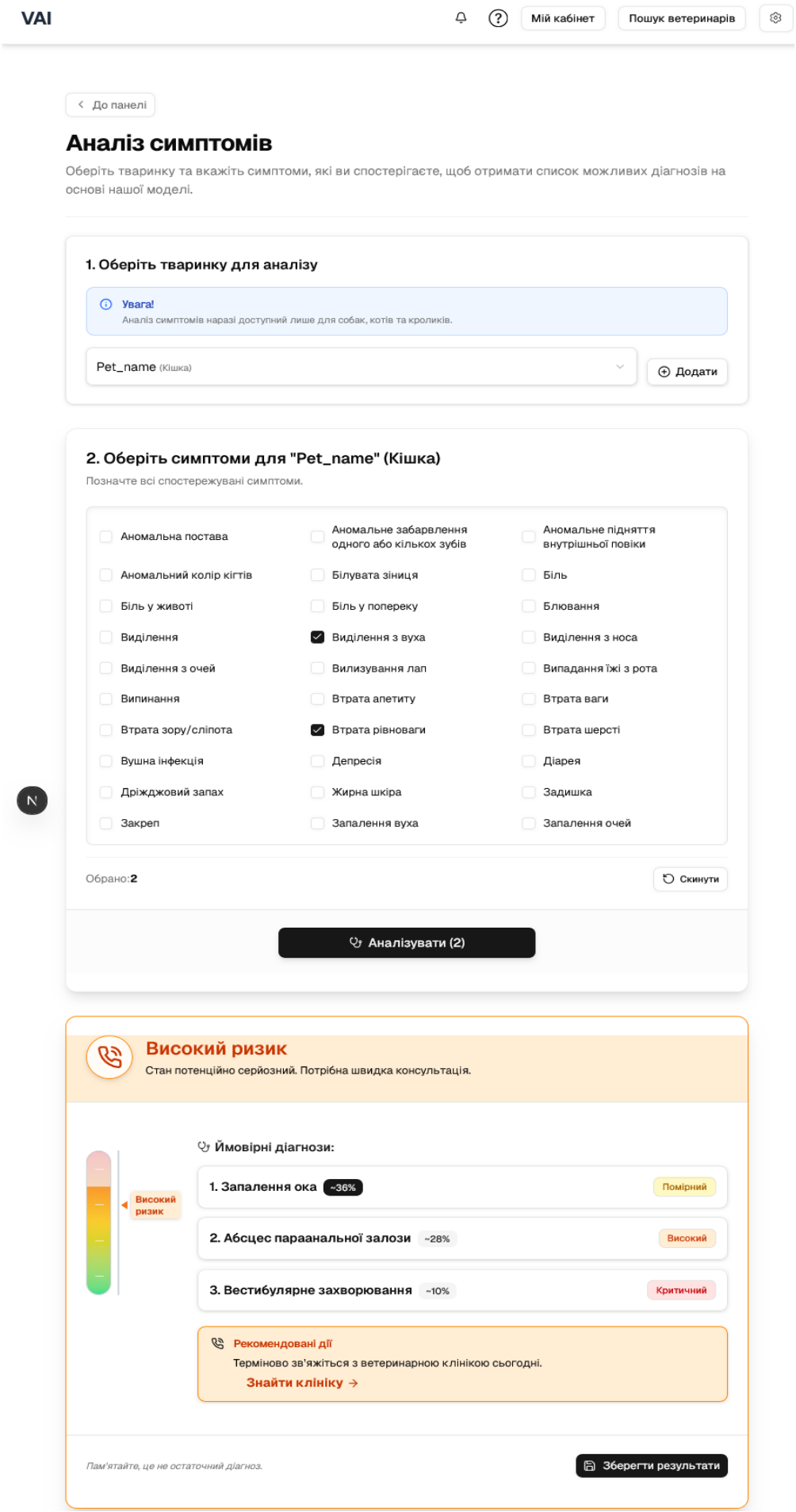


Рисунок 4.7 – Екранна форма сторінки аналізу симптомів тварини

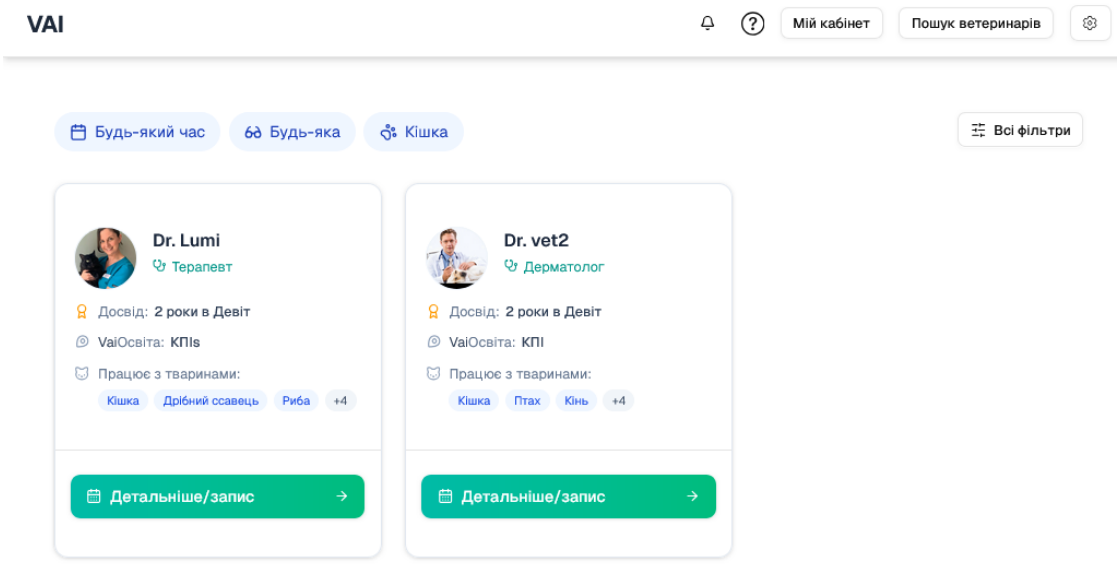


Рисунок 4.8 – Екранна форма сторінки пошуку ветеринарів

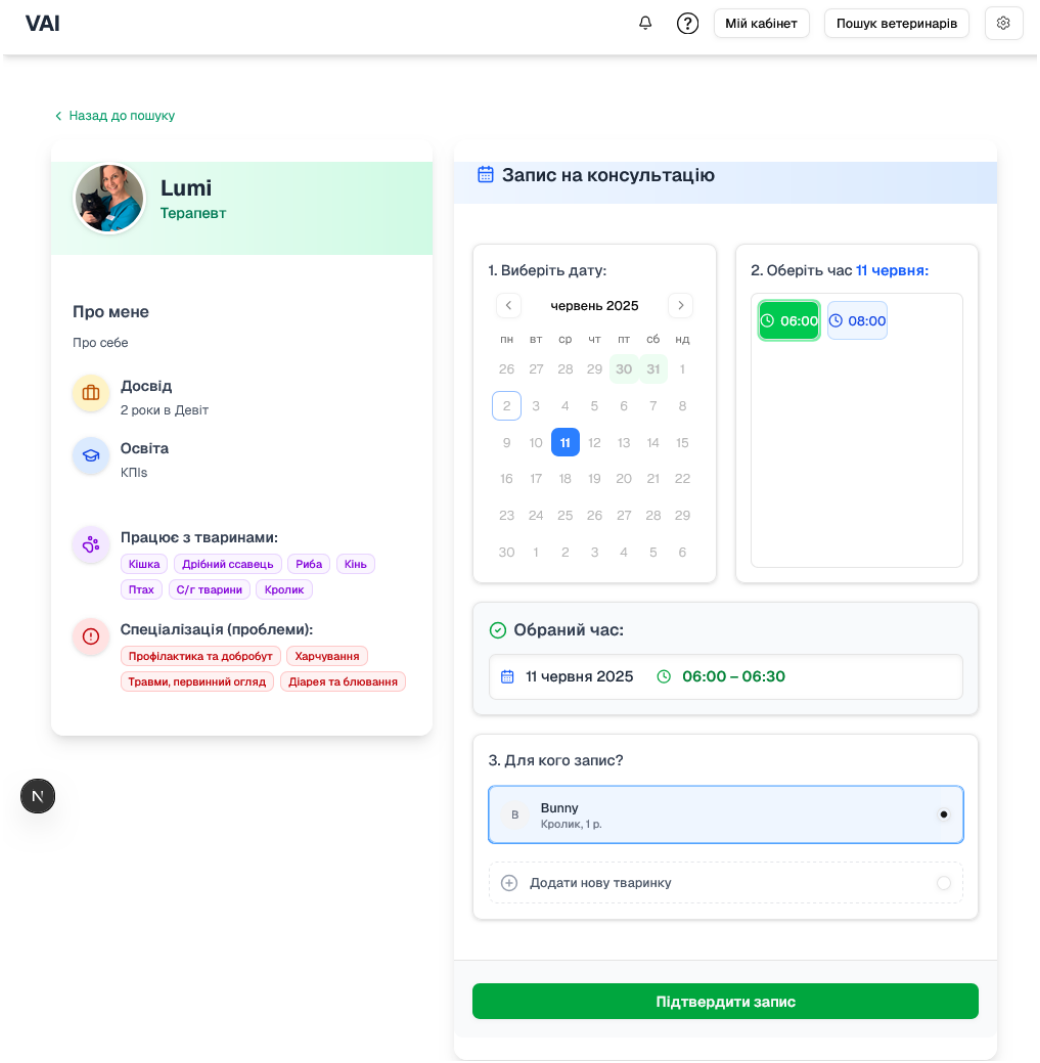


Рисунок 4.9 – Екранна форма сторінки профілю ветеринара та запису на консультацію

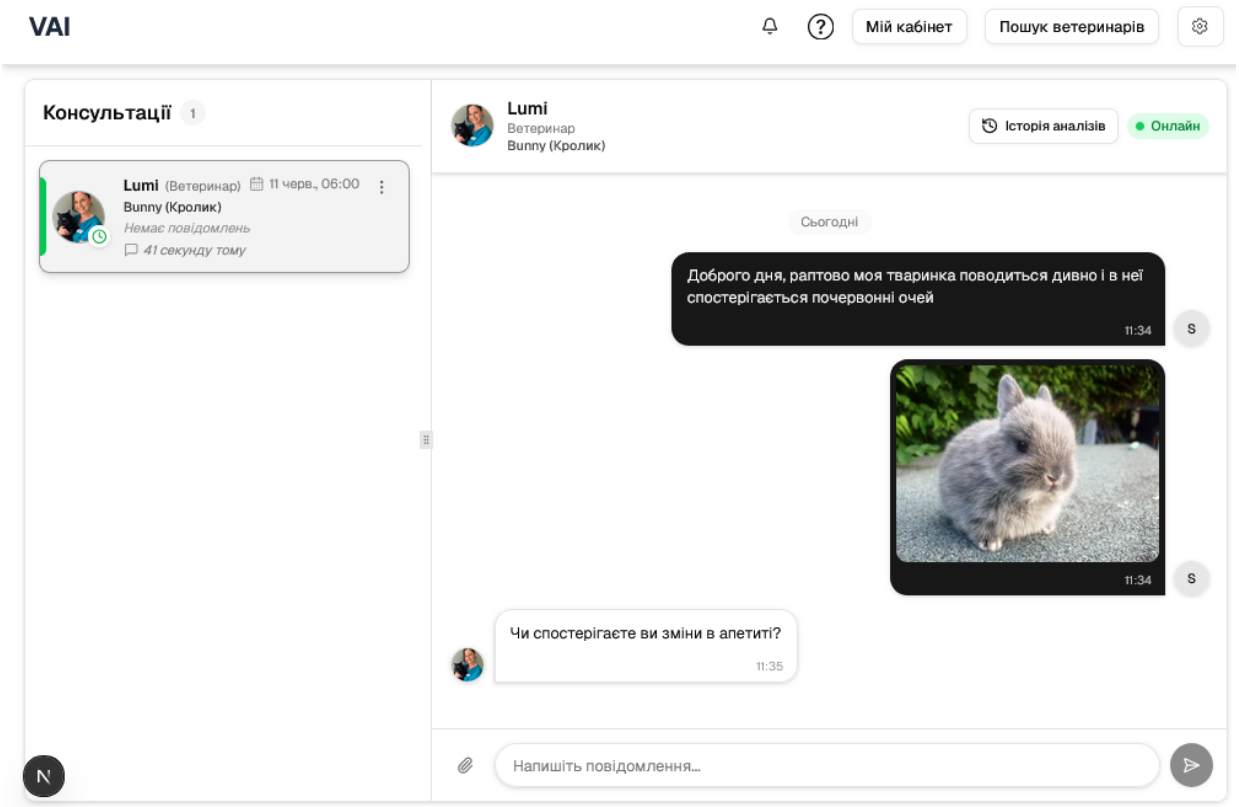


Рисунок 4.10 – Екранна форма сторінки онлайн-консультації (чат)

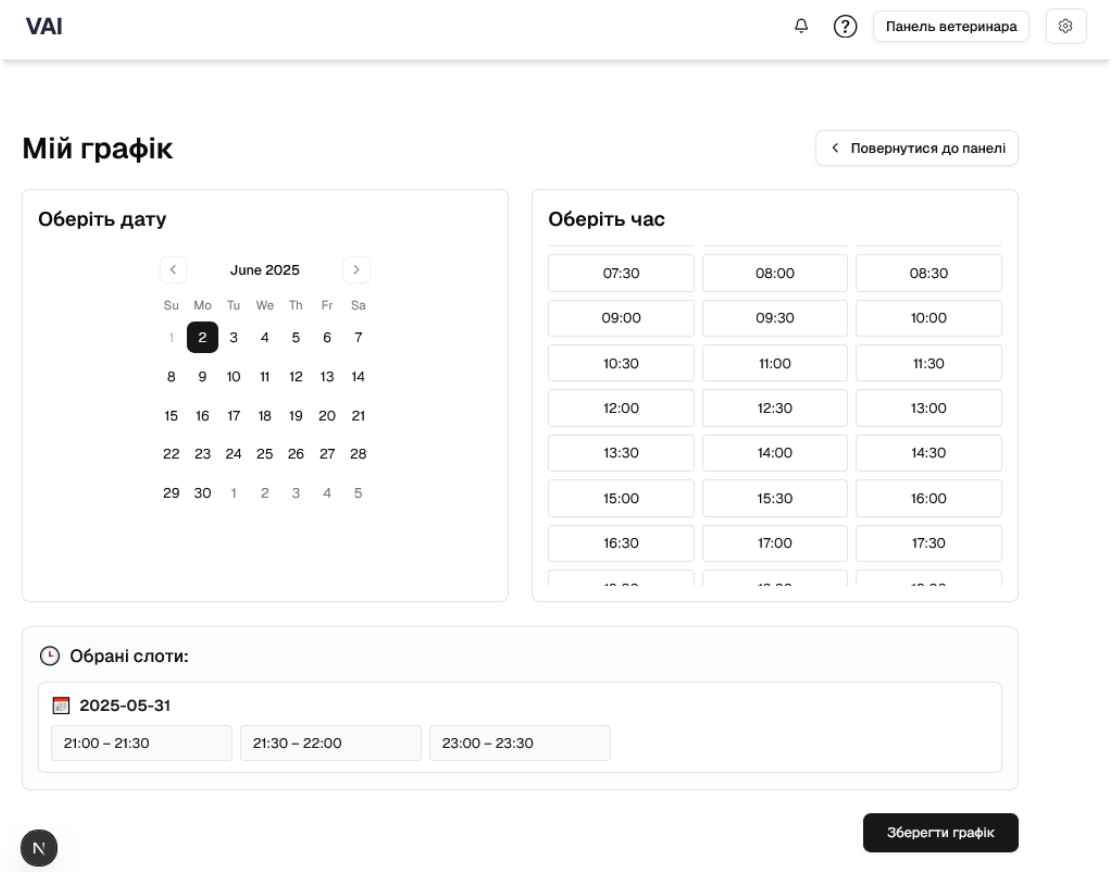


Рисунок 4.11 – Екранна форма сторінки керування розкладом ветеринара

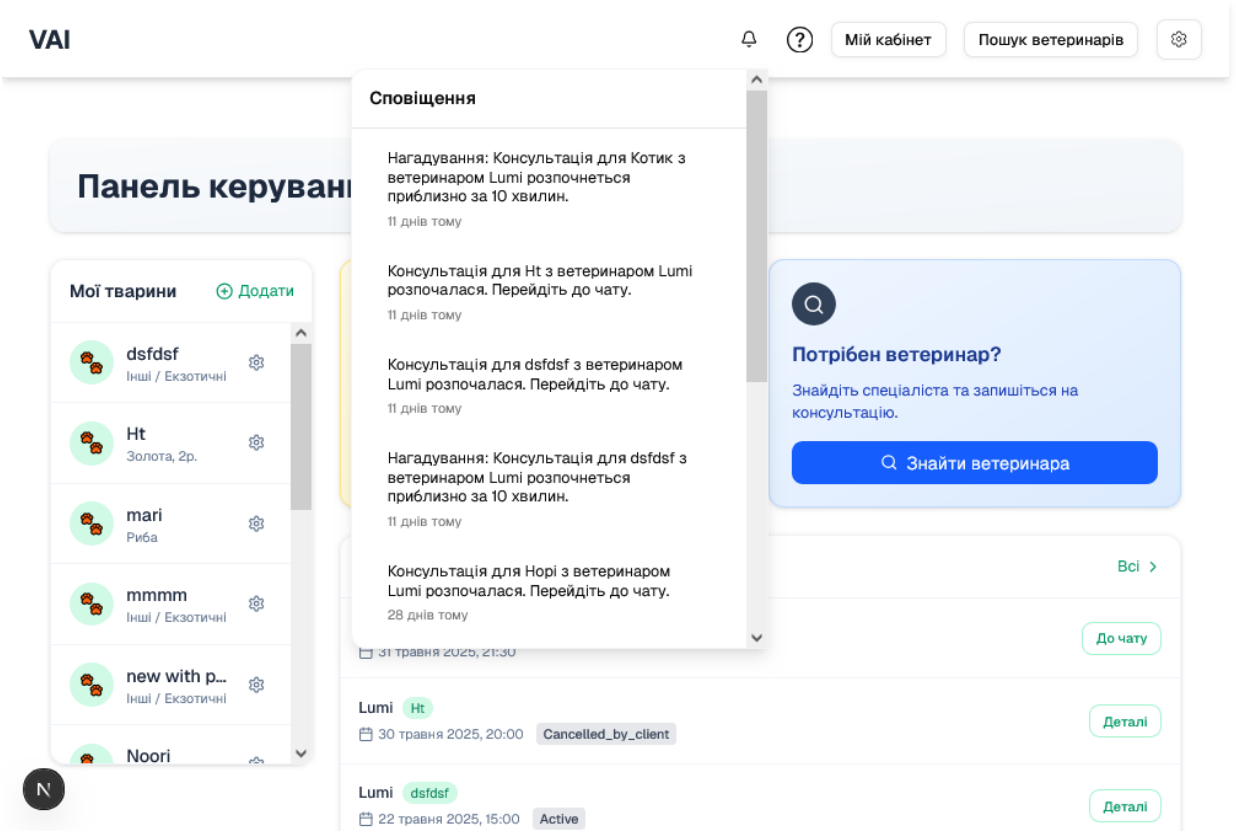


Рисунок 4.12 – Екранна форма панелі системних сповіщень

4.1.2 Облік користувачів:

4.1.2.1 Для користувача (загальні функції, доступні для Власника тварини та Ветеринара):

– можливість зареєструвати новий обліковий запис у системі;

– можливість здійснити авторизацію та увійти до існуючого облікового запису;

– можливість безпечно вийти зі свого облікового запису;

– можливість переглядати та редагувати свої загальні дані профілю (ім'я користувача, електронна пошта) ;

– можливість змінити пароль свого облікового запису;

– можливість скасувати заплановану онлайн-консультацію;

– можливість переглядати список своїх запланованих та минулих онлайн-консультацій;

– можливість відкривати чат для проведення онлайн-консультації;

– можливість надсилати текстові повідомлення під час онлайн-консультації;

– можливість надсилати зображення під час онлайн-консультації;

– можливість отримувати та переглядати текстові повідомлення та зображення в чаті в реальному часі;

– можливість отримувати та відображати системні сповіщення про важливі події;

4.1.2.2 Для Власника тварини:

– можливість додавати нову інформацію про домашню тварину (кличка, вид, порода, вік, стать, вага, медична історія, вакцинація, фото);

– можливість переглядати список усіх своїх доданих тварин з коротким описом;

– можливість редагувати раніше введені дані про домашню тварину;

– можливість видаляти запис про домашню тварину зі свого профілю;

– можливість здійснювати пошук ветеринарів за визначеними критеріями;

– можливість переглядати детальний профіль ветеринара;

– можливість записатися на онлайн-консультацію з ветеринаром, обравши доступний час та прикріпивши тваринку;

– можливість провести аналіз симптомів домашньої тварини за допомогою модуля штучного інтелекту та отримати попередній список імовірних діагнозів, рівень їх серйозності та загальні рекомендації;

– можливість зберігати отримані результати аналізу симптомів для подальшого ознайомлення;

– можливість переглядати історію раніше збережених аналізів симптомів для своїх тварин.

4.1.2.3 Для Власника тварини:

– можливість переглядати та редагувати свій професійний профіль (фото, опис, спеціалізація, досвід, освіта, види тварин, з якими працює, та проблеми, які він лікує) ;

– можливість створювати та додавати нові доступні часові слоти до свого робочого розкладу для онлайн-консультацій;

– можливість редагувати та вилучати наявні вільні часові вікна в особистому календарі;

– можливість переглядати власний робочий розклад та записи на консультації.

4.1.2.4 Додаткові вимоги:

– забезпечення коректного розмежування доступу до функціоналу та даних на основі ролей авторизованих користувачів;

– надання адаптивного дизайну інтерфейсу, що забезпечує коректне відображення та зручне використання на різних типах пристроїв (персональні комп'ютери, планшети, мобільні телефони) та в популярних веббраузерах.

4.2 Вимоги до надійності

Передбачити контроль введення інформації та захист від некоректних дій користувача. Забезпечити цілісність інформації в базі даних.

Вебзастосунок має гарантувати збереження цілісності й узгодженості інформації, здійснюючи перевірку введених даних як на стороні клієнта, так і на сервері. Система повинна правильно реагувати на програмні та мережеві збої, повідомляючи про них у зрозумілій формі, не перериваючи виконання критично важливих функцій. Допустимий час недоступності через технічні проблеми має бути мінімальним (не менше 99,5% часу протягом календарного року), а відновлення ключових функцій — не перевищувати однієї години з моменту виявлення збою.

4.3 Умови експлуатації

Умови експлуатації згідно СанПін 2.2.2.542 – 96.

4.3.1 Вид обслуговування

Розробка потребує постійного технічного обслуговування та супроводу. Це включає безперервний моніторинг стану та продуктивності серверів на хостинг-платформах, своєчасну оплату платних послуг для забезпечення безперебійної роботи, а також регулярне оновлення версій фреймворків та бібліотек для підвищення безпеки та сумісності.

4.3.2 Обслуговуючий персонал

Розробка потребує обслуговуючого персоналу для моніторингу, оновлень, виправлення помилок та підтримки. Необхідний кваліфікований IT-фахівець зі знанням веб-технологій та машинного навчання. Допускається залучення фахівців на тимчасовій основі.

4.4 Вимоги до складу і параметрів технічних засобів

Програмне забезпечення повинно функціонувати на персональних комп'ютерах через основні браузери, такі як Google Chrome, Mozilla Firefox, Microsoft Edge, Apple Safari, а також на смартфонах та планшетах (iOS та Android) через відповідні мобільні браузери.

Мінімальна конфігурація технічних засобів:

– тип процесору: Intel Core i3 (або аналог);
– об’єм ОЗП: 4 ГБ;
– підключення до мережі Інтернет зі швидкістю від 1 Мбіт/с;

Рекомендована конфігурація технічних засобів:

– тип процесору: Intel Core i5/i7 (або аналог);
– об’єм ОЗП: 8 ГБ;
– підключення до мережі Інтернет зі швидкістю від 10 Мбіт/с.

4.5 Вимоги до інформаційної та програмної сумісності

Програмне забезпечення повинно працювати під управлінням операційних систем Windows, macOS, Linux, Android та iOS. Функціональність забезпечується через основні веббраузери, такі як Google Chrome, Mozilla Firefox, Microsoft Edge та Apple Safari, які підтримують останні версії стандартів HTML5, CSS3 та JavaScript.

4.5.1 Вимоги до вхідних даних

Вимоги до вхідних даних не висуваються.

4.5.2 Вимоги до вихідних даних

Вимоги до вихідних даних не висуваються.

4.5.3 Вимоги до мови розробки

Розробку основного вебзастосунку (клієнтська частина та серверні API-маршрути) виконати на мовах програмування TypeScript (з використанням JavaScript як основи, а також HTML та CSS для користувацького інтерфейсу). Розробку модуля штучного інтелекту (ML API) виконати на мові програмування Python.

4.5.4 Вимоги до середовища розробки

Розробку основного вебзастосунку (клієнтська та серверна частини) виконати за допомогою інтегрованого середовища розробки Visual Studio Code, з використанням фреймворку Next.js (на базі бібліотеки React.js) на платформі Node.js. Розробку модуля штучного інтелекту виконати у PyCharm на мові Python з використанням фреймворку Flask та бібліотек для машинного навчання, таких як TensorFlow/Keras та Scikit-learn.

4.5.5 Вимоги до представленню вихідних кодів

Вихідний код програми має бути представлений у вигляді репозиторію

на GitHub.

4.6 Вимоги до маркування та пакування

Вимоги до маркування та пакування не висуваються.

4.7 Вимоги до транспортування та зберігання

Вимоги до транспортування та зберігання не висуваються.

4.8 Спеціальні вимоги

Спеціальні вимоги не висуваються.

5 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

5.1 Попередній склад програмної документації

У склад супроводжувальної документації повинні входити наступні документи на аркушах формату А4:

– пояснювальна записка;
– технічне завдання;
– текст програми;
– програма та методика тестування;
– керівництво користувача;

Графічна частина повинна бути виконана на аркушах формату А3 та містити наступні документи:

– схема структурна варіантів використання;
– схема структурна компонентів програмного забезпечення;
– схема бази даних;
– BPMN-діаграма проведення онлайн-консультації.

5.2 Спеціальні вимоги до програмної документації

Програмні модулі, котрі розробляються, повинні бути задокументовані, тобто тексти програм повинні містити всі необхідні коментарі.

6 СТАДІЇ І ЕТАПИ РОЗРОБКИ

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Вивчення рекомендованої літератури	15.03.2025	
2	Аналіз існуючих методів розв’язання задачі	05.04.2025	
3	Постановка та формалізація задачі	11.04.2025	
4	Розробка інформаційного забезпечення	14.04.2025	
5	Алгоритмізація задачі	21.04.2025	
6	Обґрунтування вибору використаних технічних засобів	22.04.2025	
7	Розробка програмного забезпечення	10.05.2025	
8	Налагодження програми	15.05.2025	
9	Виконання графічних документів	16.05.2025	
10	Оформлення пояснювальної записки	27.05.2025	
11	Подання ДП на попередній захист	04.06.2025	
12	Подання ДП рецензенту	10.06.2025	
13	Подання ДП на основний захист	16.06.2025	

7 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

Тестування розробленого програмного продукту виконується відповідно до “Програми та методики тестування”.

Пояснювальна записка
до дипломного проєкту

на тему: **Вебзастосунок для раннього виявлення хвороб домашніх тварин за допомогою ШІ та інтерактивного консультування з ветеринаром**

КПІ.ІП-1507.045440.02.81

Київ – 2025

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

IDE	–	Integrated Development Environment – інтегроване середовище розробки.
API	–	Application programming interface, прикладний програмний Інтерфейс.
SPA	–	Single Page Application, односторінковий застосунок.
IT	–	Інформаційні технології.
ER	–	Entity-Relation diagram.
HTTPS	–	Hypertext Transfer Protocol Secure, безпечний протокол передачі гіпертексту.
БД	–	База даних.
JSON	–	JavaScript Object Notation, текстовий формат обміну даними.
JWT	–	JSON Web Token, веб-токен JSON.
ML	–	Machine Learning, машинне навчання.
ORM	–	Object-Relational Mapping, об'єктно-реляційне відображення.
ОС	–	Операційна система.
SQL	–	Structured Query Language, структурована мова запитів.
СУБД	–	Система управління базами даних.
UCP	–	Use Case Points, точки варіантів використання.
UI	–	User Interface, інтерфейс користувача.
ШІ	–	Штучний інтелект.

ВСТУП

Здоров'я домашніх улюбленців є важливим аспектом життя їхніх власників, однак доступ до якісної та своєчасної ветеринарної допомоги часто ускладнений через географічні, фінансові чи логістичні фактори. Тому актуальність розробки програмного забезпечення, що використовує сучасні цифрові технології для підтримки здоров'я тварин, є очевидною.

Провідною тенденцією у вирішенні цих проблем є активне впровадження технологій штучного інтелекту (ШІ) для аналізу медичних даних та розвиток телеветеринарії. Оцінюючи сучасний стан розробок за тематикою дипломного проєкту, слід зазначити, що на ринку вже існують цифрові інструменти. Проте більшість з них або надають лише загальні рекомендації без глибокого індивідуального аналізу, або пропонують консультаційні послуги без інтегрованих інтелектуальних засобів попередньої оцінки стану тварини. Хоча провідні наукові установи та дослідницькі групи активно демонструють успіхи у застосуванні ШІ для діагностики окремих захворювань у тварин, комплексних вебзастосунків, які б ефективно поєднували можливості ШІ для раннього виявлення широкого спектра проблем зі здоров'ям та забезпечували зручний інтерактивний зв'язок з ветеринаром, наразі недостатньо. Це визначає потребу у створенні інноваційного продукту, здатного комплексно підійти до окреслених завдань.

Метою даного дипломного проєкту є підвищення якості та доступності ветеринарної допомоги для власників домашніх тварин шляхом розробки вебзастосунку, що дозволяє виявляти захворювання на ранніх стадіях та отримувати онлайн-консультації з ветеринарами.

Для досягнення поставленої мети, в рамках даного дипломного проєкту буде розроблено та реалізовано наступні ключові програмні модулі:

- Модуль керування обліковими записами користувачів, що забезпечить реєстрацію, автентифікацію, авторизацію та управління профілями Власників тварин і Ветеринарів.

- Модуль пошуку ветеринарів та перегляду їх профілів, який надає можливість знаходження фахівців за різними критеріями та ознайомлення з їхньою детальною інформацією.
- Модуль онлайн-консультування, який включатиме функціонал запису до ветеринарів, планування розкладу та проведення інтерактивного чату в реальному часі.
- Модуль діагностування із використанням штучного інтелекту, призначений для аналізу симптомів тварин та надання попередніх діагностичних припущень.
- Модуль управління даними про домашніх тварин, що дозволить власникам зберігати та редагувати інформацію про своїх улюбленців.
- Модуль сповіщень та нагадувань, для інформування користувачів про важливі події.

Можливі сфери застосування розробленого вебзастосунку включають використання власниками домашніх тварин для попередньої оцінки стану здоров’я своїх улюбленців та ветеринарними практиками для оптимізації процесу прийому пацієнтів і надання дистанційних консультаційних послуг. Таким чином, реалізація проєкту сприятиме цифровізації ветеринарної медицини, підвищенню оперативності діагностики та загального рівня догляду за домашніми тваринами.

1 ПЕРЕДПРОЄКТНЕ ОБСТЕЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Постановка завдання дипломного проєктування

Успішна розробка будь-якого програмного продукту, зокрема такого масштабного, як вебзастосунок для раннього виявлення хвороб домашніх тварин з використанням штучного інтелекту, неможлива без ґрунтовного попереднього аналізу. Етап передпроєктного обстеження є фундаментальним, оскільки він дозволяє глибоко зануритися у специфіку проблеми, яку покликаний вирішити майбутній застосунок, вивчити існуючий досвід та сформулювати чіткі орієнтири для подальшого проєктування та розробки.

Метою даного етапу дипломного проєктування полягає у комплексному вивченні предметної області, пов'язаної з діагностикою захворювань домашніх тварин, аналізом сучасних підходів до використання штучного інтелекту в цій сфері та інтерактивним консультуванням з ветеринарними фахівцями. Отримані результати будуть використані як підґрунтя для визначення ключових функцій системи та формування відповідних вимог до її реалізації.

У рамках поставленої мети передбачено виконання таких основних завдань:

1) Детальний аналіз предметної області: до цього пункту входить вивчення специфіки ранньої діагностики захворювань у домашніх тварин, аналіз симптоматики поширених патологій, а також визначення ролі сучасних цифрових технологій — зокрема, засобів штучного інтелекту й рішень телемедицини — у покращенні якості ветеринарної допомоги. Крім того, слід описати ключові діагностичні та консультаційні процеси, що дозволить чітко окреслити загальне технічне завдання для дипломної розробки.
2) Аналіз наявних рішень: необхідно здійснити аналіз існуючих IT-продуктів (вебдодатків, мобільних сервісів), які реалізують подібну або суміжну функціональність. Особливу увагу буде зосереджено на вивченні технічних і алгоритмічних підходів, застосовуваних для аналізу

симптомів, обробки медичних даних і організації онлайн-взаємодії між фахівцем та користувачем. Це дозволить виявити переваги та недоліки конкурентів, а також визначити унікальні характеристики запропонованого рішення.

3) Дослідження та моделювання бізнес-процесів: на основі аналізу предметної галузі буде виокремлено основні бізнес-процеси, які система повинна підтримувати або автоматизувати. Це стосується як дій зі сторони власника тварини (внесення симптомів, отримання первинних порад, запис на онлайн-прийом), так і ветеринарного фахівця (аналіз вхідної інформації, надання консультацій та рекомендацій). За потреби, ці процеси будуть змодельовані для точнішого розуміння логіки взаємодії між користувачами та системою
4) Формулювання вимог до програмного забезпечення: завершальним кроком передпроєктного етапу стане формулювання початкових вимог — функціональних, нефункціональних і загальносистемних — які стануть основою технічного дизайну та реалізації вебзастосунку.

Виконання цих завдань дозволить системно підійти до розробки, забезпечити відповідність майбутнього продукту реальним потребам користувачів та створити конкурентоспроможне рішення на ринку цифрових ветеринарних послуг.

1.2 Аналіз предметної області

Предметною областю даного дипломного проєкту визначено процес раннього виявлення захворювань домашніх тварин та надання ветеринарних консультацій з використанням сучасних інформаційних технологій. Своєчасна діагностика є критично важливою для здоров'я домашніх улюбленців, оскільки традиційний підхід звернення до фахівця при появі явних симптомів часто затримує початок лікування.

Розвиток цифрових технологій, зокрема штучного інтелекту (ШІ), відкриває значні перспективи для ветеринарної медицини. Застосування ШІ

для аналізу медичних даних тварини, таких як симптоми та анамнез, може сприяти ранньому виявленню потенційних проблем зі здоров'ям. Успішний досвід застосування подібних технологій у медицині для людей [1] підтверджує потенціал цього напрямку. Дослідження також вказують на активне вивчення застосування штучного інтелекту та машинного навчання у ветеринарії для аналізу показників здоров'я та формування відповідних рекомендацій [2]. Алгоритми, навчені на великих масивах даних, здатні ідентифікувати приховані закономірності, що є важливим для попередження розвитку захворювань.

Процес використання знань предметної області у програмному забезпеченні на поточний момент розвитку IT-технологій (стан «as-is») характеризується наявністю кількох основних категорій цифрових рішень. Ринок цифрових ветеринарних послуг активно розвивається. Серед основних напрямків імплементації IT у ветеринарії можна виділити:

- Телеветеринарія (онлайн-консультації): ці сервіси дозволяють власникам тварин дистанційно консультуватися з ветеринарами. Хоча це забезпечує оперативність, діагностичні можливості обмежені через суб'єктивність опису симптомів власником та неможливість фізичного огляду.
- Мобільні додатки для моніторингу та догляду: існують додатки для відстеження вакцинацій, харчування та активності тварин. Вони корисні для організації догляду, але зазвичай не мають функцій автоматизованого аналізу симптомів.
- Системи підтримки прийняття діагностичних рішень на основі ШІ: розробляються системи, що аналізують медичні зображення, лабораторні дані або текстові описи симптомів. ШІ може сприяти точнішій діагностиці. Однак, багато таких систем є вузькоспеціалізованими, не враховують індивідуальні особливості тварин і рідко інтегровані з консультаційними сервісами.

Загальний опис бізнес-процесів предметної області, пов'язаних із взаємодією власника тварини та ветеринарної служби, включає етапи від спостереження симптомів до отримання лікування. Інформаційні системи можуть оптимізувати ці процеси, особливо на етапах первинного звернення та консультацій. Зростаючий попит на онлайн-консультації підтверджується досвідом українських компаній, таких як Mr.Snoopy та VetOnline, які відзначають збільшення активності у цьому сегменті [3].

Недоліки поточного стану речей з предметною областю у сфері IT полягають у фрагментарності існуючих рішень. Часто відсутня єдина платформа, що поєднувала б інтелектуальний аналіз симптомів з можливістю онлайн-консультації. Це обмежує ефективне використання ШІ для раннього виявлення хвороб, незважаючи на те, що такі технології могли б стати альтернативою традиційним візитам у певних ситуаціях.

Глобальний ринок ветеринарної телемедицини демонструє значне зростання. Згідно з дослідженням Precedence Research, обсяг світового ринку телеветеринарії у 2024 році оцінювався у 306,75 мільйонів доларів США, з прогнозом зростання до 1 955,44 мільйонів доларів США до 2034 року, при сукупному середньорічному темпі зростання (CAGR) 20,35% [4]. Динаміка зростання ринку телеветеринарії представлена на рисунку 1.1. Інші аналітичні дані, зокрема від Univdatos Market Insights, також підтверджують цю тенденцію, прогнозуючи, що ринок може досягти 450 мільйонів доларів США до 2030 року, зростаючи із середньорічним темпом близько 18% [2]. Одним з рушійних факторів є збільшення витрат власників на здоров'я своїх улюбленців. Український зооринок також показує позитивну динаміку: у 2023 році він збільшився на 27%, з прогнозованим зростанням щонайменше на 15% у 2024 році [5]. Збільшення витрат українців на домашніх тварин та зростання частки онлайн-продажів зоотоварів до 11,2% у 2023 році також свідчать про актуальність цифрових рішень [5].

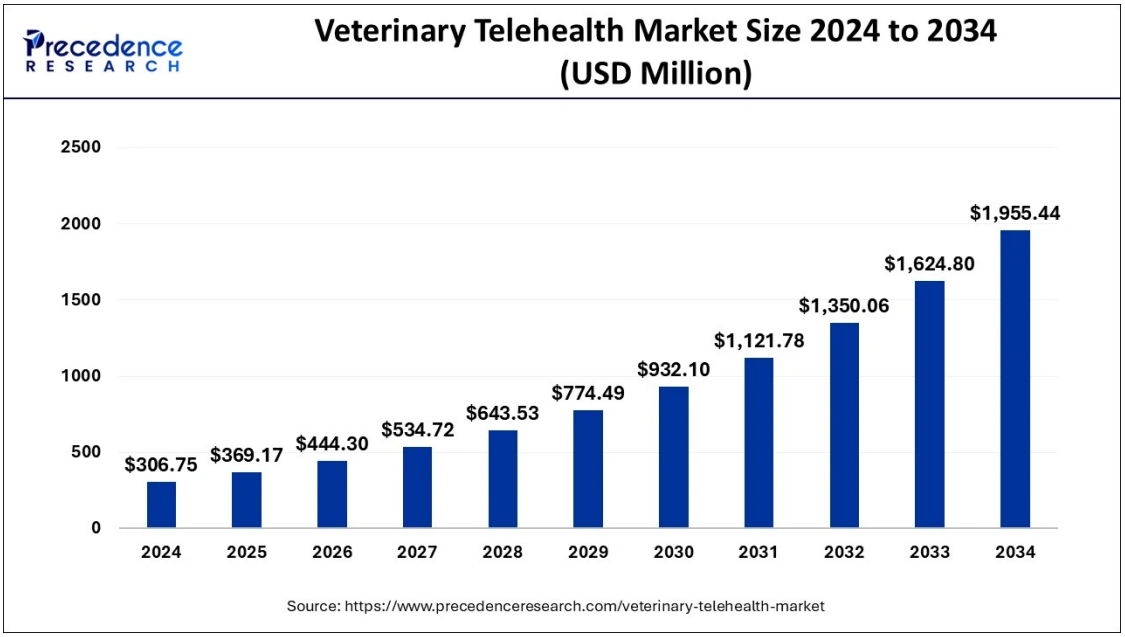


Рисунок 1.1 – Динаміка зростання ринку телеветеринарії

Можливі шляхи покращення ситуації з розробками у сфері IT за даною предметною областю полягають у створенні комплексних інтегрованих платформ. Такі платформи мають забезпечувати:

- Інтерфейс для введення даних про тварину та її симптоми.
- ШІ-модуль для попереднього аналізу наданої інформації.
- Функціонал для інтерактивного консультування з ветеринаром.

У рамках даного дипломного проєкту було обрано шлях розробки саме такого інтегрованого вебзастосунку. Передбачається, що система дозволить власникам тварин отримувати попередню оцінку стану здоров'я своїх улюбленців за допомогою ШІ та доступ до онлайн-консультацій з ветеринарами. Це спрямовано на підвищення доступності та ефективності ветеринарної допомоги, особливо на етапі раннього виявлення захворювань.

1.3 Аналіз існуючих рішень

Проаналізуємо відоме на сьогодні алгоритмічне забезпечення у даній області та технічні рішення, що допоможуть у реалізації вебзастосунку для раннього виявлення хвороб домашніх тварин за допомогою ШІ та інтерактивного консультування з ветеринаром. Далі будуть розглянуті готові програмні рішення, допоміжні програмні засоби та засоби розробки.

1.3.1 Аналіз відомих програмних продуктів

Для визначення конкурентних переваг та оптимального напрямку розробки вебзастосунку для раннього виявлення хвороб домашніх тварин було проведено аналіз ключових програмних продуктів, що існують на ринку цифрових ветеринарних послуг. Увагу було зосереджено на тих платформах, функціонал яких частково або повністю перетинається із запланованим у даному дипломному проєкті, зокрема, на можливостях онлайн-консультування з ветеринарами та/або використанні елементів діагностики на основі технологій штучного інтелекту.

1) Vetster: Vetster (vetster.com) позиціонується як канадська платформа телеветеринарії, що забезпечує зв'язок між власниками домашніх тварин та ліцензованими ветеринарними фахівцями. Комунікація відбувається через відеозв'язок, текстовий чат або телефонні дзвінки. Платформа надає користувачам можливість вибору ветеринара за різними критеріями, такими як спеціалізація, рейтинг та вартість послуг. Vetster пропонує консультації з широкого кола питань, включаючи здоров'я, харчування та поведінку тварин, а також можливість отримання альтернативної думки щодо раніше встановленого діагнозу [6].

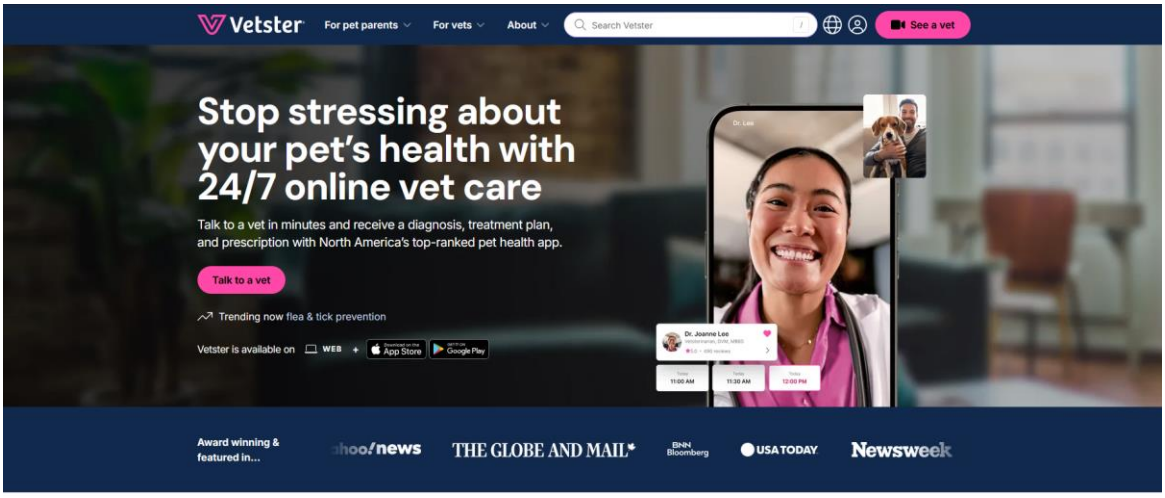


Рисунок 1.2 – Домашня сторінка Vetsler [6]

Ключовий функціонал Vetster охоплює відео- та текстові консультації з ветеринарами, можливість вибору спеціаліста за різноманітними критеріями, інструменти для планування віртуальних візитів, а також опцію зберігання медичної історії тварини та, у деяких регіонах, отримання електронних рецептів. Щодо переваг Vetster, варто відзначити гнучкість у виборі ветеринара, широкий спектр надаваних консультаційних послуг та доступність платформи у різних країнах. Серед недоліків можна відзначити відсутність інструментів для попередньої автоматизованої оцінки стану тварини на основі симптомів до початку консультації, що могло б оптимізувати час спілкування з лікарем. Діагностичні висновки формуються ветеринаром виключно під час онлайн-сесії на основі візуального огляду та інформації, наданої власником тварини.

2) IDEXX VetConnect PLUS: IDEXX VetConnect PLUS (idexx.com) є інтегрованою діагностичною платформою, розробленою компанією IDEXX. Цей програмний продукт призначений переважно для використання у ветеринарних клініках та дозволяє агрегувати діагностичні дані з різних джерел, таких як лабораторні аналізатори (аналізи крові, сечі), системи візуалізації (рентген, УЗД) та монітори пацієнтів [7]. Платформа використовує алгоритми машинного навчання для аналізу цих даних та надання підтримки ветеринарним лікарям у процесі постановки діагнозу.

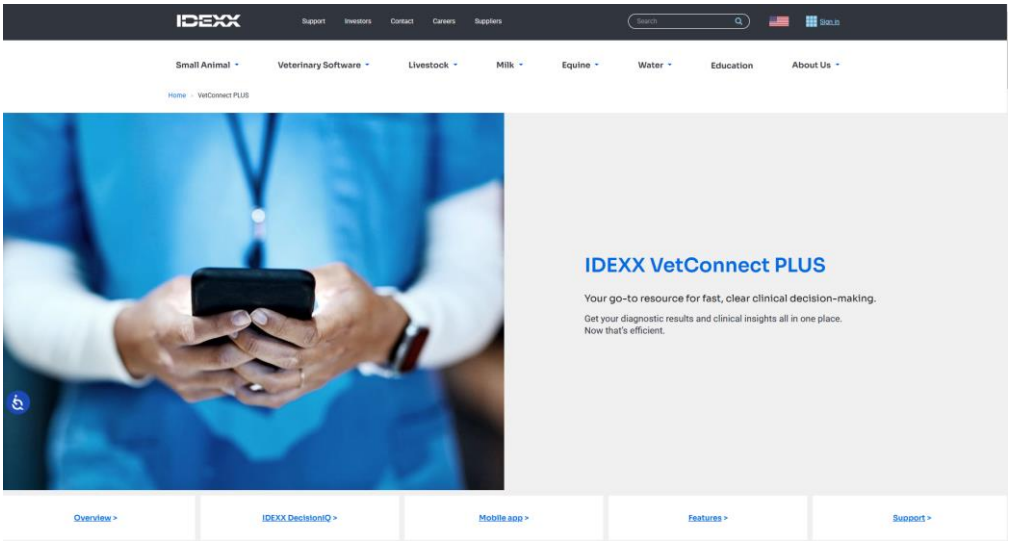


Рисунок 1.3 – Домашня сторінка IDEXX VetConnect PLUS [7]

Основний функціонал IDEXX VetConnect PLUS включає інтеграцію діагностичних даних з різноманітних джерел, автоматизований аналіз лабораторних показників, аналіз медичних зображень із застосуванням технологій штучного інтелекту, візуалізацію отриманих даних та трендів, а також доступ до експертних висновків та інтеграцію з системами управління ветеринарною практикою (PIMS).

Перевагами даної платформи є комплексний підхід до діагностики, використання ШІ для підвищення точності та оперативності діагностичних процедур, а також ефективна підтримка ветеринарних фахівців у прийнятті клінічних рішень.

Основними недоліками для широкого кола користувачів є те, що платформа орієнтована виключно на професійних ветеринарних лікарів та використання в умовах клініки, що робить її недоступною для безпосереднього використання власниками тварин для самостійної попередньої оцінки стану улюбленця. Також, висока вартість та складність інтеграції обмежують її доступність поза межами спеціалізованих ветеринарних закладів.

3) Animo (від Sure Petcare): Animo (surepetcare.com/animo) – це система моніторингу активності та поведінки собак, яка складається з невеликого носимаого пристрою, що кріпиться до нашийника, та відповідного мобільного додатку. Пристрій збирає дані про рівень фізичної активності тварини, якість та тривалість сну, а також фіксує специфічні поведінкові патерни, такі як гавкіт, чухання та тремтіння. Зібрана інформація передається до мобільного додатку, де вона аналізується за допомогою алгоритмів, що вивчають індивідуальні норми поведінки конкретної собаки. Система сповіщає власника про значущі та тривалі відхилення від звичних показників, що може свідчити про потенційні проблеми зі здоров'ям, стрес або поведінкові розлади [8,9]. Приклад візуалізації даних про активність та поведінку в мобільному додатку Animo представлено на рисунку 1.4.

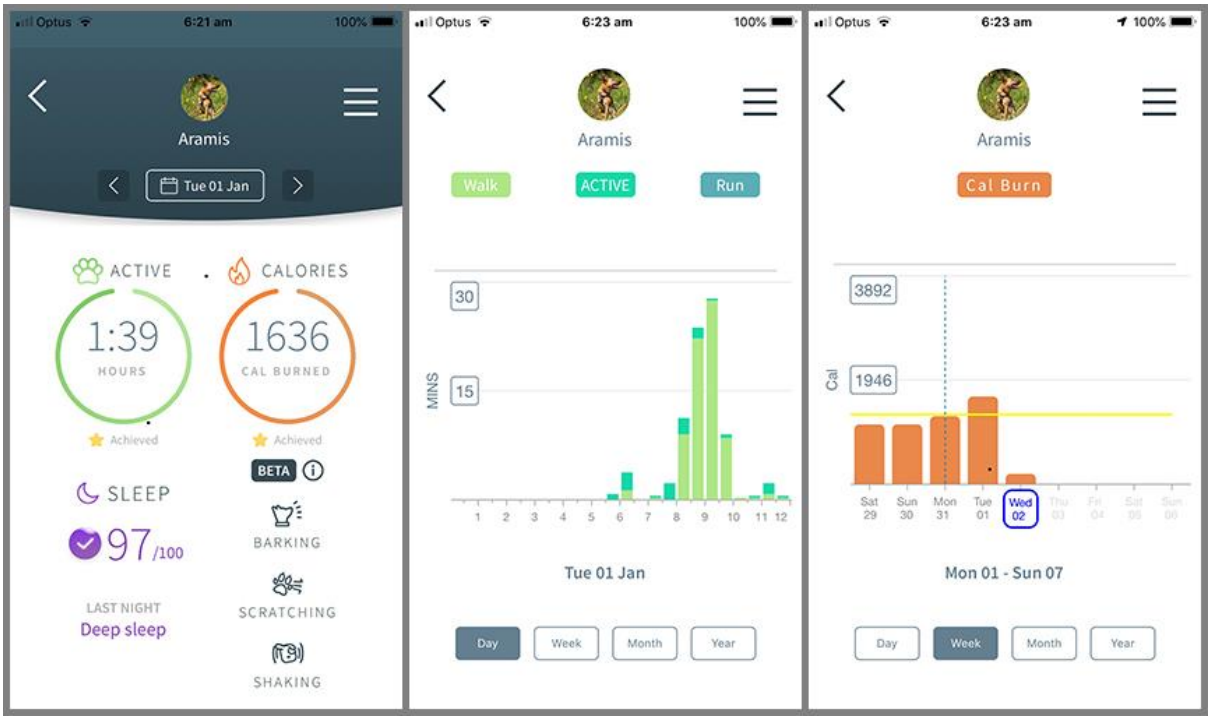


Рисунок 1.4 – Інтерфейс додатку Animo [10]

Ключовий функціонал Animo полягає у довгостроковому моніторингу активності та поведінки тварини, аналізі сну, детекції таких проявів як гавкіт, чухання та тремтіння, встановленні індивідуальних поведінкових норм для конкретної собаки та наданні сповіщень власнику про суттєві зміни.

Серед переваг системи варто виділити можливість виявлення поступових змін у стані тварини, які можуть бути непомітними для власника при щоденному спостереженні, об'єктивізацію даних про поведінку завдяки постійному збору даних, а також фокус на проактивному моніторингу стану здоров'я.

Недоліками системи є її обмеженість виключно моніторингом фізичної активності та деяких поведінкових проявів, без можливості врахування інших важливих симптомів, які власник міг би ввести вручну (наприклад, зміни апетиту, характеру випорожнень тощо). Система не надає користувачеві припущень щодо можливих причин виявлених відхилень або конкретних захворювань, а лише сигналізує про наявність змін. Також відсутня можливість безпосередньо з додатку зв'язатися з ветеринаром для обговорення даних моніторингу.

Для наочного порівняння функціональних можливостей та особливостей розглянутих програмних продуктів із дипломний проектом було складено порівняльну таблицю 1.1.

Таблиця 1.1 – Порівняння з аналогами

Функціонал /Критерій	Дипломний проєкт(Vai)	Vetster	IDEXX VetConnect PLUS	Animo (Sure Petcare)	Пояснення
Цільова аудиторія: Власники тварин	+	+	-	+	IDEXX орієнтований на ветклініки. Animo – на власників собак.
Онлайн-консультації з ветеринаром	+	+	-	-	IDEXX та Animo не мають вбудованої функції телеконсультацій.
Попередня діагностика на основі ШІ	+	_	+	+/-	Vetster не має ШІ. IDEXX має ШІ для лікарів. Animo аналізує поведінку, але не ставить діагноз

Продовження таблиці 1.1

Аналіз симптомів (введення власником)	+	-	-	-	Vetster – лише в рамках збору анамнезу лікарем. IDEXX та Animo не передбачають ручного введення симптомів
Інтеграція з ШІ для допомоги власнику	+	-	-	+	Vetster та IDEXX не надають ШІ-інструментів безпосередньо власнику. Animo надає сповіщення на основі ШІ
Доступність для індивідуального користувача	+	+	-	+	IDEXX потребує підписки клініки. Animo – купівлі пристрою
Безкоштовний базовий функціонал	+	-	-	-	Vetster – платні консультації. IDEXX – частина платних послуг клініки. Animo – купівля пристрою

Продовження таблиці 1.1

Україномовний інтерфейс	+	+/-	+/-	-	Для Vetster та IDEXX залежить від локалізації. Animo зазвичай не має
Надання рекомендацій щодо звернення до спеціаліста/ клініки	+	+	-	+	Vetster – через консультацію. Animo – через сповіщення. IDEXX не для власників

1.3.2 Аналіз відомих алгоритмічних та технічних рішень

Під час розробки вебзастосунку, що поєднує функціональність штучного інтелекту та інтерактивне консультування, ключовим є вибір оптимальних алгоритмічних і технологічних рішень для забезпечення надійності, масштабованості та зручності користування [11].

Щодо серверної частини (Backend), існує кілька підходів до архітектури та добору інструментів. Один із варіантів — створення системи на базі готових систем управління контентом (CMS), таких як WordPress, що зазвичай використовують мову PHP у зв’язці з базами даних MySQL [12]. Перевага такого підходу — швидке розгортання для типових завдань, однак він суттєво обмежує можливості налаштування, продуктивність при високих навантаженнях та інтеграцію складних ШІ-модулів [13].

Більш гнучкі та продуктивні рішення зазвичай створюються з використанням сучасних мов програмування та фреймворків. Python разом із Django (який надає комплексне рішення «з коробки» з ORM, адмін-панеллю

тощо) або Flask (мінімалістичний і зручний мікрофреймворк) є популярними виборами для проєктів зі штучним інтелектом завдяки великому набору бібліотек і простоті синтаксису [14]. Node.js з Express.js дозволяє використовувати JavaScript на серверній стороні, що полегшує розробку для full-stack JavaScript-команд і добре підходить для створення API чи реального часу додатків, як-от чати, завдяки асинхронній моделі введення-виведення [15]. Для побудови масштабованих, продуктивних корпоративних систем часто обирають Java (зокрема фреймворк Spring) або C# із платформою .NET від Microsoft, яка забезпечує високу продуктивність та сумісність з екосистемою Microsoft [16].

Щодо архітектури, традиційна монолітна модель, де весь функціонал реалізовано в одному застосунку, є простішою на стартових етапах, проте обмежує масштабованість та оновлення окремих компонентів. Натомість мікросервісна архітектура, яка передбачає поділ системи на окремі незалежні сервіси з API-взаємодією, забезпечує кращу масштабованість, відмовостійкість та технологічну гнучкість, хоч і ускладнює управління та розгортання [17].

Клієнтська частина (Frontend), яка забезпечує взаємодію з користувачем через веббраузер, базується на технологіях HTML, CSS та JavaScript. Хоча можна створювати інтерфейс за допомогою «чистих» вебтехнологій, це часто призводить до високих затрат часу та складності підтримки коду. Тому широко використовуються фреймворки. React.js, створений Facebook, є популярною бібліотекою для побудови UI, яка базується на компонентному підході — що сприяє повторному використанню коду й спрощує управління інтерфейсом [18]. Серед переваг React — висока продуктивність через використання віртуального DOM, велика спільнота та гнучкість інтеграції. Недолік — потреба у додаткових бібліотеках для маршрутизації (React Router) та керування станом (Redux, Context API) [19]. Angular від Google — повноцінний фреймворк на TypeScript з вбудованими модулями для форм, HTTP, маршрутизації, ін'єкції залежностей тощо. Переваги Angular —

структурованість і придатність для Enterprise-рівня проєктів. Його недоліком є високий поріг входу [20]. Vue.js — прогресивний JavaScript-фреймворк з низьким порогом входу, гнучкістю та чіткою документацією. Проте, порівняно з React або Angular, має дещо менш розвинену екосистему [21]. Згадуючи про React, варто також зазначити що доволі поширеним рішенням є використання Next.js — фреймворку на основі React.js із підтримкою серверного рендерингу та сучасних розробницьких підходів [22].

Вибір системи управління базами даних (СУБД) залежить від типу даних. Для структурованої інформації (профілі, історії консультацій тощо) доцільно використовувати реляційні СУБД, такі як PostgreSQL (ACID-сумісна, підтримує складні запити) або MySQL (проста в освоєнні, популярна) [23]. Для зберігання гнучко структурованих або неструктурованих даних (логи, текстові описи) підходять NoSQL-рішення: MongoDB — документоорієнтована БД для JSON-даних, Redis — key-value сховище для кешу й сесій [24].

Щодо модуля ШІ, для аналізу текстових описів симптомів застосовуються методи обробки природної мови (NLP). Найсучасніші з них використовують архітектуру Transformer (BERT, GPT) з можливістю донавчання на специфічних даних. Існують потужні бібліотеки: Transformers (Hugging Face), spaCy, NLTK [25]. Для аналізу зображень застосовуються згорткові нейронні мережі (CNN), реалізовані через TensorFlow або PyTorch, часто з transfer learning [26]. Для класифікації також використовуються модулі Scikit-learn (логістична регресія, дерева рішень, random forest, SVM) [27].

Для реалізації чату в реальному часі застосовуються технології WebSocket, що підтримують постійне двостороннє з'єднання. Популярною бібліотекою є Socket.IO, яка пропонує зручний API, підтримку «кімнат» та fallback на інші протоколи за потреби [28].

1.4 Опис бізнес-процесів

Для графічного зображення основних бізнес-процесів вебзастосунку використовується нотація BPMN (Business Process Model and Notation) версії 2.0. Відповідні діаграми наведено на рисунках 1.5 – 1.10.

Послідовність реєстрації нового користувача:

- Відвідувач відкриває сторінку створення акаунта.
- Вносить персональні дані у форму (ім’я, електронна пошта, пароль).
- Система здійснює первинну перевірку введених значень, помилки підсвічуються.
- Користувач визначає тип облікового запису («Власник тварини» або «Ветеринар»).
- У разі вибору «Ветеринар» необхідно додатково заповнити професійні відомості.
- Дані надсилаються на сервер, де проходять фінальну перевірку та перевірку на унікальність.
- У разі виявлення помилок, система надсилає повідомлення з описом помилки.
- Якщо перевірка проходить успішно, у базі даних створюється новий обліковий запис (а також профіль ветеринара, якщо було обрано «Зареєструватися як ветеринар»).
- Користувач отримує підтвердження про реєстрацію та перенаправляється до форми входу.

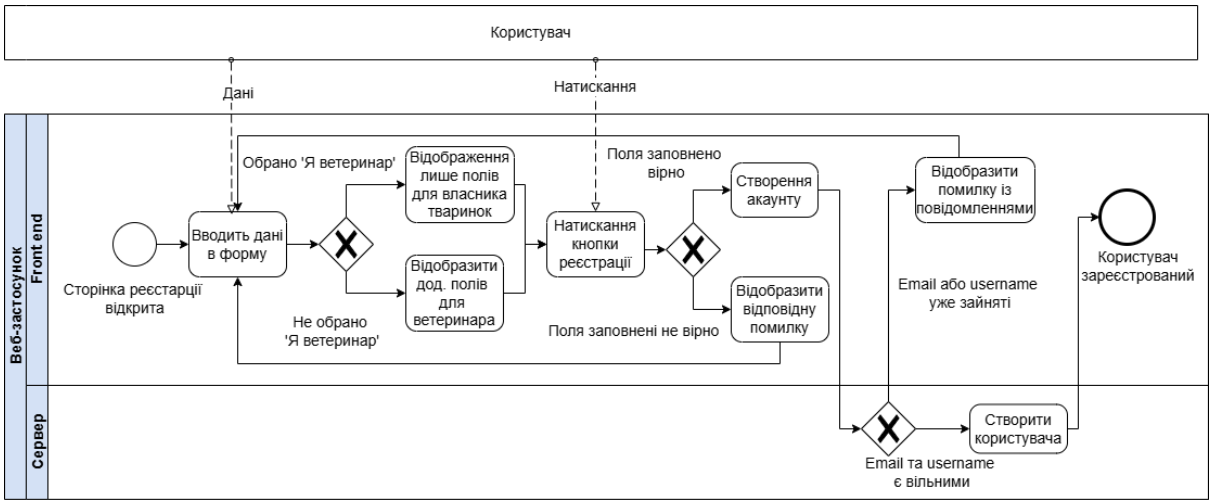


Рисунок 1.5 – BPMN-діаграма процесу реєстрації

Послідовність входу до системи:

- Користувач відкриває сторінку авторизації.
- Вводить логін (електронну адресу або ім’я) і пароль.
- Дані передаються на сервер для автентифікації.
- Сервер звіряє отриману інформацію з наявними записами.
- Якщо облікові дані правильні — ініціюється сесія, після чого користувача перенаправляють на панель для відповідного користувача.
- У випадку помилки, система інформує про неправильні дані.

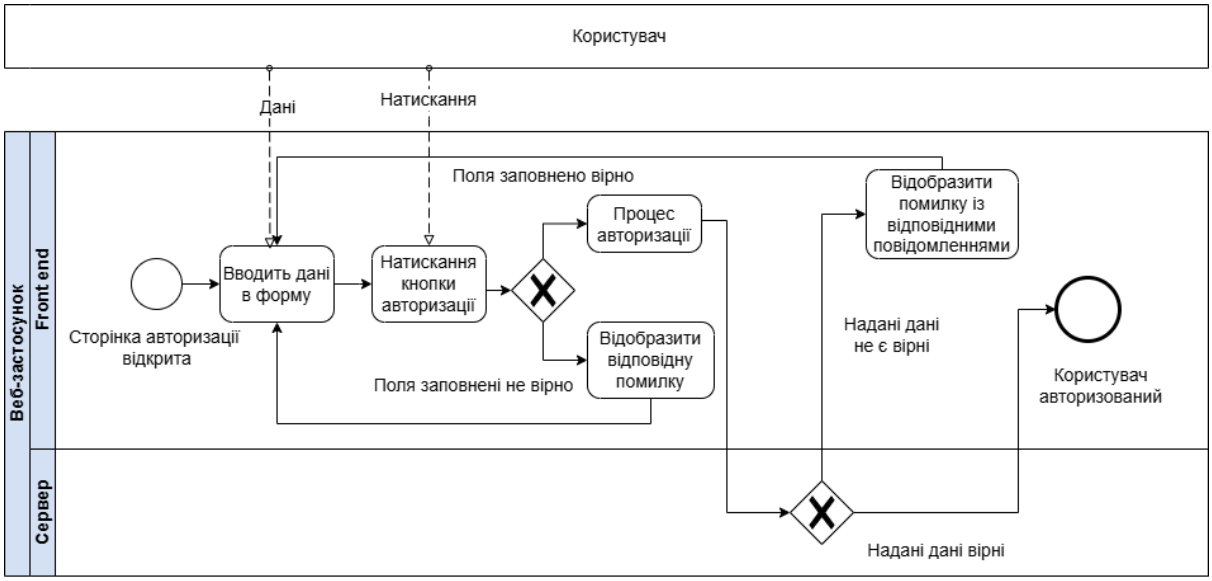


Рисунок 1.6 – BPMN-діаграма авторизації користувача

Послідовність додавання тварини:

- Авторизований користувач відкриває форму для внесення нової тварини.
- Заповнює поля (ім’я, вид, порода, вік тощо).
- За бажанням додає фото тварини.
- Відбувається перевірка правильності введених даних на стороні клієнта.
- Якщо інформація коректна — її надсилають на сервер.
- На сервері створюється новий запис про тварину, який пов’язується з користувачем.
- Користувач отримує підтвердження про успішне додавання.

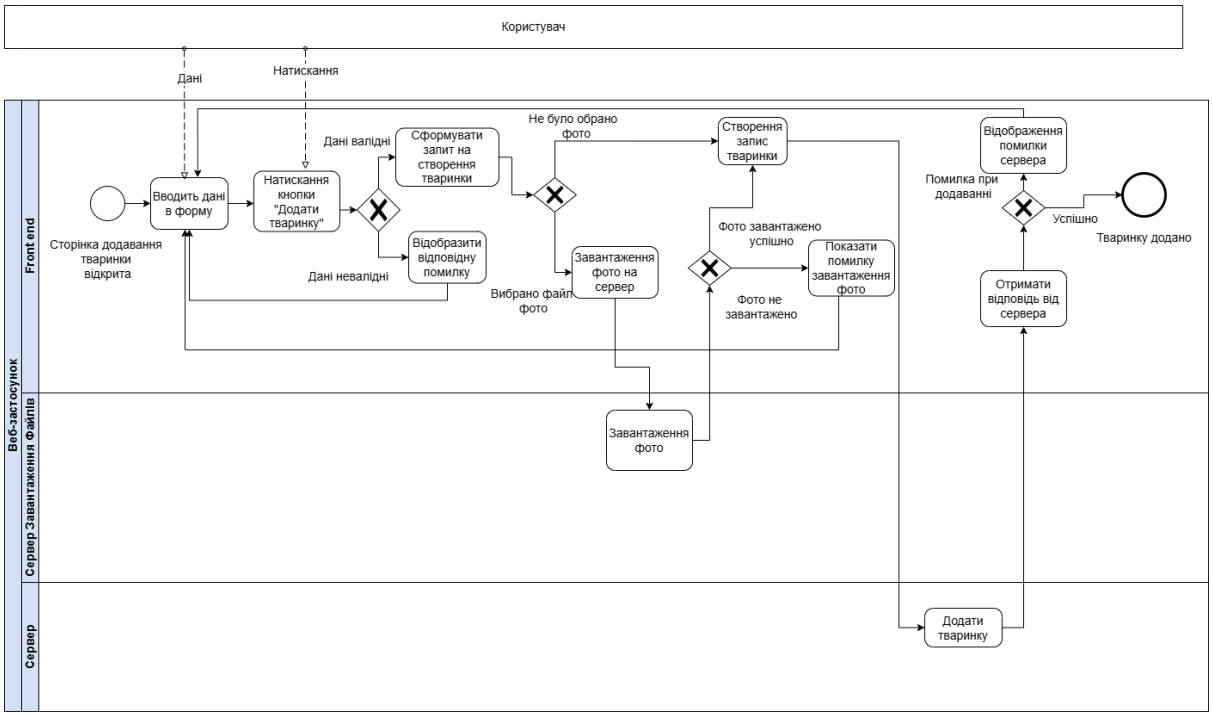


Рисунок 1.7 – BPMN-діаграма додавання тварини

Послідовність пошуку ветеринара:

- Користувач переходить до інтерфейсу пошуку спеціалістів.
- Вводить ПІБ ветеринара або застосовує фільтри (напрям спеціалізації, типи тварин).
- Параметри пошуку надсилаються на сервер.
- Відбувається пошук відповідних профілів у базі.
- Користувач бачить результати або повідомлення про їхню відсутність.

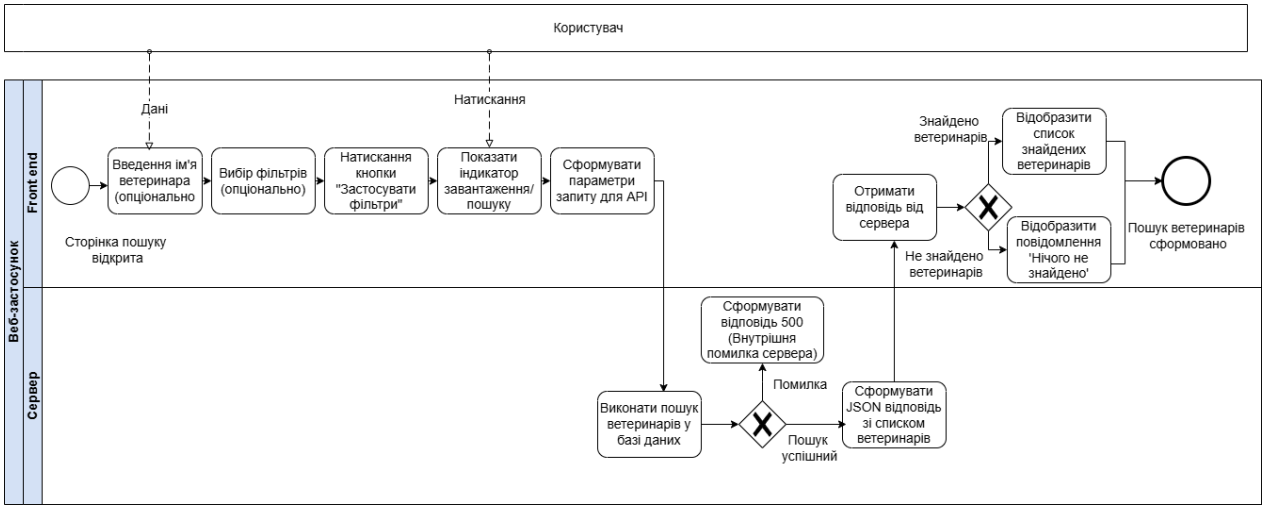


Рисунок 1.8 – BPMN-діаграма пошуку ветеринара

Послідовність запису на консультацію:

- Користувач обирає дату та час консультації на сторінці профілю ветеринара.
- Вказує одну з власних тваринок для запису.
- Підтверджує бажання забронювати обраний слот.
- Запит на бронювання надсилається серверу.
- Система перевіряє доступність слоту та правильність введених даних, після чого створює запис у базі.
- У випадку успіху — користувач отримує підтвердження.
- Якщо слот вже зайнятий або виявлено помилку, надходить відповідне повідомлення.

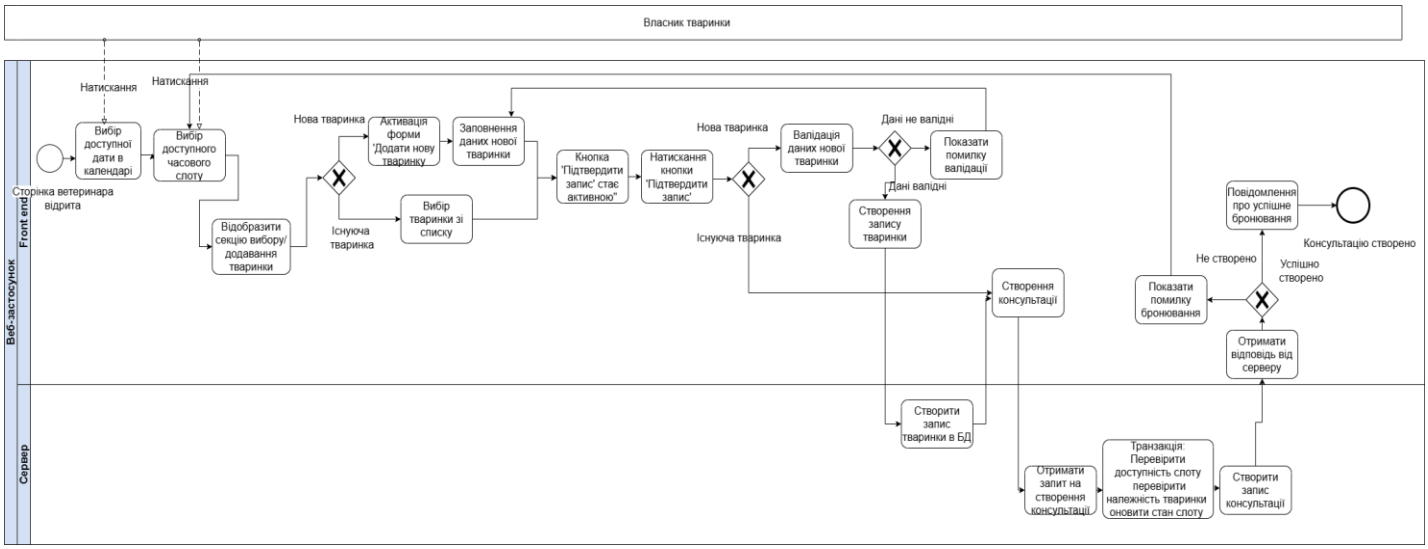


Рисунок 1.9 – BPMN-діаграма бронювання консультації

Послідовність проведення текстової онлайн-консультації:

- Власник тварини та ветеринар переходять на сторінку запланованої сесії.
- Відображається історія повідомлень, встановлюється канал зв’язку в реальному часі.
- Учасники спілкуються через текстові повідомлення з можливістю прикріплення зображень.
- Кожне повідомлення фіксується в базі та відображається для обох сторін.
- Чат триває до завершення консультації.

BPMN-діаграма, що ілюструє цей процес, наведена у графічному матеріалі, плакат «BPMN-діаграма проведення онлайн-консультації».

Послідовність аналізу симптомів за допомогою штучного інтелекту:

- Користувач заходить до розділу аналізу стану тварини.
- Обирає відповідну тваринку зі списку.
- Підтверджує ознайомлення з попередженням про попередній характер діагностики.
- Вибирає симптоми з доступного переліку.
- Ініціює запуск аналізу.
- Інформація надсилається на сервер із ШІ-моделлю.
- ШІ генерує прогноз: можливі діагнози, ймовірність, ступінь серйозності.
- Результати передаються користувачу разом з порадами.

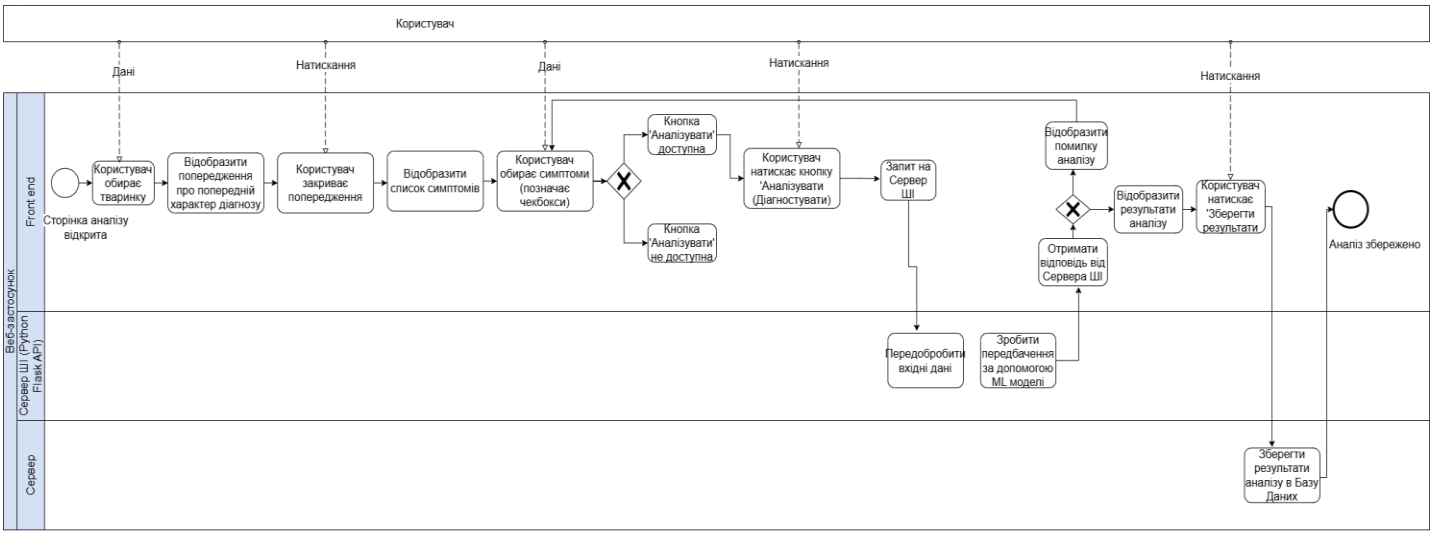


Рисунок 1.10 – BPMN-діаграма аналізу симптомів з використанням ШІ

Висновки до розділу

У першому розділі дипломної роботи проведено попереднє дослідження, що дало змогу сформувати цілісне бачення предметної області та визначити основні передумови для створення вебсервісу з раннього виявлення захворювань домашніх тварин із використанням ШІ та дистанційного консультування ветеринарів.

Аналіз галузі підтвердив актуальність проблеми своєчасної діагностики. Встановлено, що сучасні IT-рішення, зокрема штучний інтелект і телеветеринарія, мають значний потенціал для вдосконалення існуючих підходів. Ринок цифрових ветеринарних послуг демонструє зростання як у світі, так і в Україні, проте досі відчувається нестача інтегрованих рішень, які поєднували б AI-діагностику з можливістю швидкого консультування. Саме це визначає перспективність пропонованої розробки.

Порівняльний аналіз наявних програмних рішень — Vetster, IDEXX VetConnect PLUS, Animo — дозволив виявити їхні сильні та слабкі сторони, що допомогло чітко окреслити унікальні функції майбутнього продукту. Також проаналізовано основні алгоритмічні методи та технічні підходи, що використовуються у подібних вебзастосунках, що дало змогу сформувати технологічну базу для подальшої реалізації.

На основі отриманих даних визначено ключові бізнес-процеси, які має підтримувати система: реєстрація та вхід користувачів, управління профілями тварин, AI-діагностика на основі симптомів, пошук ветеринарів, запис та проведення онлайн-консультацій. Для візуалізації логіки роботи було застосовано BPMN-нотацію.

Таким чином, результати підготовчого етапу забезпечили глибоке розуміння проблематики, потреб ринку та існуючих рішень, що дозволило сформулювати обґрунтовані вимоги для подальшого проєктування вебзастосунку, спрямованого на підвищення доступності та якості ветеринарної допомоги.

2 РОЗРОБЛЕННЯ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Варіанти використання програмного забезпечення

Головними функціями програмного забезпечення є забезпечення можливості для власників тварин проводити попередню діагностику стану здоров'я своїх улюбленців за допомогою аналізу симптомів засобами штучного інтелекту, а також отримання інтерактивних онлайн-консультацій з кваліфікованими ветеринарними спеціалістами. Більш детальний перелік функцій та взаємодію з ними акторів системи можна побачити на діаграмі варіантів використання, яка наведена на рисунку 2.1.

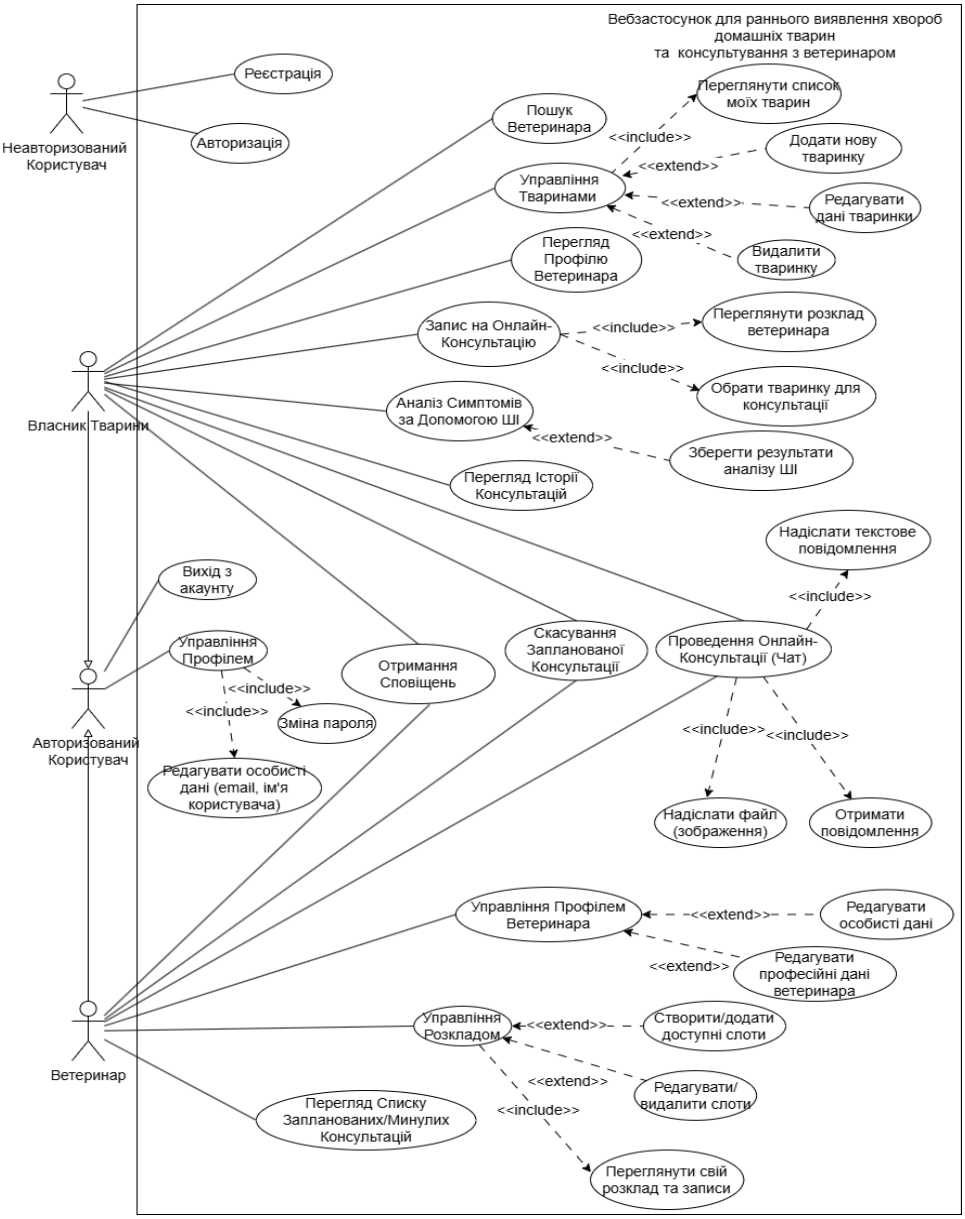


Рисунок 2.1 – Діаграма варіантів використання

В таблицях 2.1 – 2.15 наведені варіанти використання програмного забезпечення.

Таблиця 2.1 - Варіант використання UC-1

Use case name	Реєстрація користувача
Use case ID	UC-01
Goals	Створення нового облікового запису користувача в системі, дозволяючи доступ до функціоналу для авторизованих користувачів.
Actors	Гість (незареєстрований користувач)
Trigger	Користувач бажає створити новий обліковий запис та переходить на сторінку реєстрації.
Pre-conditions	—
Flow of Events	Користувач переходить на сторінку реєстрації. В поля для реєстрації вводяться відповідні дані: ім'я користувача, електронна пошта, пароль та його повтор для підтвердження. За бажанням, встановлюється прапорець «Я ветеринар», після чого з'являються додаткові поля для профілю ветеринара (інформація про себе, спеціалізація, освіта, досвід, види тварин, з якими працює, та проблеми, з якими допомагає), які також заповнюються. Після заповнення даних користувач натискає кнопку «Зареєструватися». З'являється повідомлення про успішну реєстрацію, і користувач перенаправляється на сторінку входу.

Продовження таблиці 2.1

Extension	– У випадку введення некоректних даних або незаповнення обов'язкових полів, кнопка реєстрації може бути неактивною, а відповідні поля підсвічуються з повідомленням про помилку. – Якщо електронна пошта або ім'я користувача вже зайняті, після спроби реєстрації відображається відповідне повідомлення про помилку.
Post-Condition	Створено обліковий запис користувача. Здійснено перехід на сторінку входу. Або: Відображено повідомлення про помилку.

Таблиця 2.2 - Варіант використання UC-2

Use case name	Авторизація користувача
Use case ID	UC-02
Goals	Надання доступу зареєстрованому користувачеві до його облікового запису та функціоналу системи, призначеного для авторизованих користувачів.
Actors	Гість (незареєстрований користувач)
Trigger	Користувач бажає увійти до системи та переходить на сторінку авторизації.
Pre-conditions	Користувач має існуючий зареєстрований обліковий запис в системі.

Продовження таблиці 2.2

Flow of Events	Користувач переходить на сторінку входу. В поля для авторизації вводяться відповідні дані: електронна пошта (або ім'я користувача) та пароль. Після заповнення даних користувач натискає кнопку «Увійти» та перенаправляється на свою особисту панель керування (дашборд).
Extension	– У випадку введення некоректних даних або незаповнених полів, відповідні поля підсвічуються з повідомленням про помилку. – Якщо дані користувача введені невірно, або такого користувача не існує – відображається повідомлення про помилку авторизації.
Post-Condition	Користувача авторизовано. Здійснено перехід на панель керування. Або: Відображено повідомлення про помилку.

Таблиця 2.3 - Варіант використання UC-03

Use case name	Пошук ветеринара
Use case ID	UC-03
Goals	Надати користувачам можливість знаходити ветеринарів за іменем та/або фільтрами.
Actors	Власник тварини, Ветеринар
Trigger	Користувач переходить на сторінку пошуку ветеринарів.
Pre-conditions	—

Продовження таблиці 2.3

Flow of Events	Користувач відкриває сторінку пошуку ветеринарів. Відображається поле для введення імені та/або набір фільтрів (наприклад, за спеціалізацією, видами тварин). Користувач вводить ім'я та/або обирає потрібні фільтри. Після введення критеріїв пошуку (або автоматично при зміні фільтрів) оновлюється список ветеринарів, що відповідають запиту. Користувач може переглядати список та переходити на сторінки профілів окремих ветеринарів.
Extension	– Якщо за вказаними критеріями ветеринарів не знайдено, відображається відповідне повідомлення – Якщо виникає помилка під час пошуку, відображається повідомлення про помилку
Post-Condition	Відображено список ветеринарів, що відповідають критеріям пошуку, або повідомлення про відсутність результатів

Таблиця 2.4 - Варіант використання UC-04

Use case name	Перегляд профілю ветеринара
Use case ID	UC-04
Goals	Отримати детальну інформацію про конкретного ветеринара, його спеціалізацію, досвід, послуги та доступний розклад для можливого запису на консультацію
Actors	Власник тварини
Trigger	Користувач обирає ветеринара зі списку результатів пошуку або переходить за прямим посиланням на профіль ветеринара
Pre-conditions	Користувач авторизований

Продовження таблиці 2.4

Flow of Events	Відкривається сторінка профілю обраного ветеринара. На сторінці відображається інформація про ветеринара: фото (якщо є), ім'я, спеціалізація, опис, досвід роботи, освіта, види тварин, з якими працює, та проблеми, за якими можна звернутися. Також відображається календар з доступними для запису часовими слотами. Власник тварини може бачити кнопку для запису на консультацію.
Extension	– Якщо профіль ветеринара не знайдено або він неактивний, відображається відповідне повідомлення
Post-Condition	Користувач ознайомлений з інформацією про ветеринара та його розкладом. Власник тварини може перейти до процесу запису на консультацію

Таблиця 2.5 - Варіант використання UC-5

Use case name	Керування профілем користувача
Use case ID	UC-05
Goals	Надати можливість авторизованому користувачеві переглядати та змінювати свої особисті дані (ім'я, email, пароль).
Actors	Авторизований користувач (Власник тварини або Ветеринар)
Trigger	Користувач переходить до розділу налаштувань свого профілю/акаунту.
Pre-conditions	Користувач авторизований в системі.

Продовження таблиці 2.5

Flow of Events	Користувач відкриває сторінку редагування свого профілю. Відображаються поточні дані: ім'я користувача, адреса електронної пошти та поля для зміни пароля. Користувач вносить бажані зміни в поля імені та/або email. Для зміни пароля користувач вводить поточний пароль (якщо вимагається), новий пароль та його підтвердження. Після внесення змін користувач натискає кнопку «Зберегти зміни». У разі успішного збереження з'являється відповідне повідомлення
Extension	– При введенні некоректних даних (наприклад, невірний формат email, короткий пароль, паролі не співпадають) відповідні поля підсвічуються з повідомленням про помилку. – Якщо новий email або ім'я користувача вже зайняті, відображається відповідна помилка – При невдалій спробі змінити пароль (наприклад, невірний старий пароль) відображається помилка
Post-Condition	Особисті дані користувача оновлені. Або: Відображено повідомлення про помилку збереження змін.

Таблиця 2.6 - Варіант використання UC-06

Use case name	Вихід з акаунту
Use case ID	UC-06
Goals	Завершити поточну сесію авторизованого користувача в системі
Actors	Авторизований користувач (Власник тварини, Ветеринар)
Trigger	Користувач натискає кнопку «Вихід» в меню навігації
Pre-conditions	Користувач авторизований в системі

Продовження таблиці 2.6

Flow of Events	Користувач натискає кнопку «Вийти». Після цього його поточна сесія завершується, і він перенаправляється на головну сторінку
Extension	—
Post-Condition	Сесія користувача завершена. Користувач більше не авторизований і бачить контент для неавторизованих користувачів

Таблиця 2.7 - Варіант використання UC-7

Use case name	Керування тваринами
Use case ID	UC-07
Goals	Надати Власнику тварини можливість додавати, переглядати, редагувати та видаляти інформацію про своїх тварин.
Actors	Власник тварини
Trigger	Власник тварини переходить до розділу «Мої тварини» на панелі керування.
Pre-conditions	Користувач авторизований як Власник тварини.

Продовження таблиці 2.7

Flow of Events	(Перегляд списку): Власник тварини відкриває розділ «Мої тварини», де відображається список його доданих тварин. (Додавання): Власник тварини натискає кнопку «Додати тваринку». Відкривається форма для введення даних нової тварини (ім'я, вид, порода, вік, стать, вага, мед. історія, вакцинація, фото). Після заповнення та збереження форми нова тваринка з'являється у списку. (Редагування): Власник тварини обирає тваринку зі списку та натискає опцію «Редагувати». Відкривається форма з поточними даними тварини. Власник вносить зміни та зберігає їх. (Видалення): Власник тварини обирає тваринку зі списку та натискає опцію «Видалити». З'являється запит на підтвердження видалення. Після підтвердження тваринка видаляється зі списку.
Extension	– При некоректному введенні даних під час додавання або редагування тварини, поля підсвічуються з повідомленнями про помилки. – Якщо виникає помилка під час збереження або видалення даних на сервері, відображається відповідне повідомлення.
Post-Condition	Список тварин користувача оновлено відповідно до виконаних дій (додано, відредаговано або видалено тваринку). Або: Відображено повідомлення про помилку.

Таблиця 2.8 - Варіант використання UC-8

Use case name	Аналіз симптомів за допомогою ШІ
Use case ID	UC-08

Продовження таблиці 2.8

Goals	Отримання попереднього списку імовірних діагнозів.
Actors	Авторизований користувач
Trigger	Власник тварини переходить на сторінку аналізу.
Pre-conditions	Користувач авторизований.
Flow of Events	Відкривається сторінка аналізу симптомів. З'являється запит на вибір тварини (Кіт, Собака, Кролик) або додавання нової. Після вибору тварини відображається попередження про попередній характер діагнозу, яке підтверджується. Далі відображається список симптомів. Обирається один або декілька симптомів, і кнопка «Аналізувати» стає активною. При натисканні кнопки «Аналізувати» з'являється індикатор обробки. Після отримання результатів аналізу, індикатор зникає, і відображається список імовірних діагнозів з відсотками, рівень важкості, рекомендації (включаючи пораду звернутися до ветеринара) та кнопки «Знайти ветеринара» і (опціонально) «Зберегти результати аналізу».
Extension	– Якщо не обрано симптомів, кнопка «Аналізувати» неактивна або з'являється підказка. – У випадку помилки запиту до сервісу ШІ, відображається повідомлення – При спробі покинути сторінку з незбереженими результатами, з'являється попередження.
Post-Condition	Надано інформацію про можливі діагнози.

Таблиця 2.9 - Варіант використання UC-9

Use case name	Запис на онлайн-консультацію
Use case ID	UC-9
Goals	Забронювати час для онлайн-спілкування з ветеринаром.
Actors	Власник тварини
Trigger	Власник тварини знаходиться на сторінці профілю ветеринара та обирає записатися.
Pre-conditions	Користувач авторизований. Ветеринар має доступні часові слоти.
Flow of Events	На сторінці ветеринара Власник тварини бачить календар та обирає вільну дату. Після цього відображається список доступних часових проміжків для цієї дати, і Власник тварини обирає один з них. Далі з'являється можливість обрати тваринку зі свого списку або, якщо потрібно, заповнити форму для додавання нової. Після вибору слоту та тварини активується кнопка «Підтвердити запис». При її натисканні, у разі успіху, з'являється повідомлення про успішне бронювання.
Extension	– Якщо на обрану дату немає вільних слотів, відображається відповідна інформація. – При некоректному заповненні даних для нової тварини, поля підсвічуються з помилками. – Якщо обраний слот стає недоступним під час процесу, з'являється повідомлення про помилку.
Post-Condition	Консультацію заплановано. Або: Відображено повідомлення про помилку бронювання.

Таблиця 2.10 - Варіант використання UC-10

Use case name	Перегляд списку своїх консультацій
Use case ID	UC-10
Goals	Надати авторизованому користувачеві (Власнику тварини або Ветеринару) можливість переглядати список своїх запланованих та минулих онлайн-консультацій
Actors	Власник тварини, Ветеринар
Trigger	Користувач переходить до розділу «Всі консультації» зі своєї панелі керування.
Pre-conditions	Користувач авторизований в системі
Flow of Events	Користувач відкриває сторінку зі списком своїх консультацій. Відображається список, що містить інформацію про кожну консультацію: дата та час, ім'я іншого учасника (ветеринара для власника, або власника та ім'я тварини для ветеринара), статус консультації (наприклад, «Заплановано», «Завершено», «Скасовано»). Для кожної консультації може бути доступна опція переходу до чату або (для запланованих) опція скасування. Список може бути розділений на майбутні та минулі консультації.
Extension	– Якщо у користувача немає жодної консультації, відображається відповідне повідомлення
Post-Condition	Користувач бачить список своїх консультацій з основною інформацією по кожній.

Таблиця 2.11 - Варіант використання UC-11

Use case name	Скасування запланованої консультації
Use case ID	UC-11
Goals	Надати можливість Власнику тварини або Ветеринару скасувати раніше заплановану онлайн-консультацію
Actors	Власник тварини, Ветеринар
Trigger	Користувач (Власник або Ветеринар) переглядає список своїх консультацій та вирішує скасувати одну з них
Pre-conditions	Користувач авторизований. Існує запланована консультація, до якої користувач має відношення і яку дозволено скасувати (час консультації ще не настав)
Flow of Events	Користувач знаходить потрібну консультацію у списку та обирає опцію «Скасувати». З'являється діалогове вікно з проханням підтвердити скасування та попередженням про незворотність дії. Користувач підтверджує скасування. Після успішного скасування (обробки на сервері) консультація позначається як скасована, і слот, якщо він був заброньований, звільняється
Extension	– Якщо консультацію вже неможливо скасувати (наприклад, вона вже відбулася або до неї залишилося занадто мало часу згідно з політикою скасування), опція скасування може бути неактивною або відображається повідомлення про неможливість скасування – Якщо виникає помилка під час обробки запиту на скасування, відображається відповідне повідомлення

Продовження таблиці 2.11

Post-Condition	Заплановану консультацію скасовано, відповідний часовий слот звільнено. Обидва учасники (якщо можливо) сповіщені про скасування. Або: Відображено повідомлення про помилку

Таблиця 2.12 - Варіант використання UC-12

Use case name	Проведення онлайн-консультації (Чат)
Use case ID	UC-12
Goals	Спілкування між Власником тварини та Ветеринаром в реальному часі
Actors	Власник тварини, Ветеринар
Trigger	Учасник відкриває сторінку консультації
Pre-conditions	Учасники авторизовані. Консультація існує
Flow of Events	Відкривається сторінка чату консультації, де відображається історія повідомлень (якщо є) та інтерфейс для спілкування. Учасник вводить текст у поле повідомлення та натискає кнопку «Надіслати». Повідомлення з'являється в його інтерфейсі та в інтерфейсі іншого учасника. Для надсилання зображення, учасник натискає кнопку «Прикріпити зображення», обирає файл, який перевіряється на відповідність типу та розміру. У разі успіху, повідомлення із зображенням з'являється в чаті для обох учасників. Обмін повідомленнями та зображеннями може тривати до завершення консультації.

Продовження таблиці 2.12

Extension	– Якщо виникає помилка підключення до сервісу чату, відображається відповідне повідомлення – При помилці надсилання повідомлення відправник бачить помилку – Якщо файл зображення невалідний або виникає помилка його завантаження, відображається помилка
Post-Condition	Інформація передана між учасниками. Історія чату оновлена

Таблиця 2.13 - Варіант використання UC-13

Use case name	Керування профілем ветеринара
Use case ID	UC-13
Goals	Надати Ветеринару можливість переглядати та оновлювати свій професійний профіль.
Actors	Ветеринар
Trigger	Ветеринар переходить до розділу керування своїм профілем.
Pre-conditions	Користувач авторизований як Ветеринар.
Flow of Events	Ветеринар відкриває сторінку свого професійного профілю. Відображаються поточні дані: фото (якщо є), інформація про себе, спеціалізація, освіта, досвід, обрані види тварин та проблеми, з якими він працює. Ветеринар може редагувати ці поля, завантажувати або змінювати фото профілю. Після внесення змін Ветеринар натискає кнопку «Зберегти». У разі успішного збереження дані профілю оновлюються

Продовження таблиці 2.13

Extension	– При введенні некоректних даних або незаповненні обов'язкових полів для професійного профілю, відповідні поля підсвічуються з повідомленнями про помилки. – Якщо виникає помилка під час завантаження фото або збереження даних профілю на сервері, відображається відповідне повідомлення
Post-Condition	Професійний профіль Ветеринара оновлено. Або: Відображено повідомлення про помилку збереження змін

Таблиця 2.14 - Варіант використання UC-14

Use case name	Керування розкладом ветеринара
Use case ID	UC-14
Goals	Надати Ветеринару інструменти для створення, перегляду, редагування та видалення доступних часових слотів для онлайн-консультацій
Actors	Ветеринар
Trigger	Ветеринар переходить до розділу керування своїм розкладом
Pre-conditions	Користувач авторизований як Ветеринар

Продовження таблиці 2.14

Flow of Events	Ветеринар відкриває сторінку керування розкладом. Відображається календар та список існуючих часових слотів. Для додавання нового слоту, Ветеринар обирає дату та слот/слоти. Для видалення, Ветеринар обирає дату та слот, який бажає видалити, здійснює натискання на нього. Або, у переліку усіх обраних слотів знаходить потрібний та натискає іконку хрестика в даному часовому слоті. Всі зміни зберігаються по натисненню кнопки «Зберегти графік». Список доступних слотів та відображення в календарі оновлюються.
Extension	– При спробі створити слот, що перекривається з існуючим, або при введенні некоректного часу, відображається помилка – Якщо слот, який намагаються видалити або редагувати, вже заброньований, відображатися попередження (про необхідність спочатку скасувати пов'язану консультацію) – Якщо виникає помилка при збереженні змін розкладу на сервері, відображається відповідне повідомлення.
Post-Condition	Розклад Ветеринара оновлено відповідно до виконаних дій. Або: Відображено повідомлення про помилку.

Таблиця 2.15 - Варіант використання UC-15

Use case name	Отримання/Перегляд сповіщень
Use case ID	UC-15

Продовження таблиці 2.15

Goals	Інформувати авторизованого користувача про важливі події в системі, такі настання 10 хвилин до запланованої консультації, настання часу консультації, скасування консультації.
Actors	Авторизований користувач (Власник тварини, Ветеринар)
Trigger	Система генерує сповіщення на основі подій; користувач може відкрити спеціальний розділ сповіщень або бачити індикатори нових сповіщень
Pre-conditions	Користувач авторизований. Відбулася подія, що генерує сповіщення
Flow of Events	При виникненні релевантної події (наприклад нагадування про консультацію за 10 хв до початку, скасування консультації іншим учасником), користувач отримує сповіщення. Це може бути індикатор нових сповіщень (червона крапка на іконці дзвіночка) в інтерфейсі. При переході до розділу сповіщень відображається список отриманих сповіщень, кожне з яких містить текст та посилання на відповідну сторінку (наприклад, на чат консультації). Користувач може позначити сповіщення як прочитані
Extension	—
Post-Condition	Користувач проінформований про важливі події. Список сповіщень оновлено.

2.2 Розроблення функціональних вимог

Програмне забезпечення розділене на модулі. Кожен модуль має свій певний набір функцій. В таблиці 2.16 наведено загальну модель вимог, а в таблиці 2.17 наведений опис функціональних вимог до програмного забезпечення. Матрицю трасування вимог можна побачити на рисунку 2.2.

Таблиця 2.16 – Загальна модель вимог

№	Назва вимоги	ID вимоги	Пріоритет	Ризики
1	Реєстрація нового користувача	FR-1	Високий	Середній
2	Авторизація існуючого користувача	FR-2	Високий	Середній
3	Вихід з облікового запису	FR-3	Високий	Низький
4	Редагування загальних даних профілю користувача	FR-4	Середній	Низький
5	Зміна пароля користувача	FR-5	Середній	Середній
6	Додавання нової тварини	FR-6	Високий	Низький
7	Перегляд списку своїх тварин	FR-7	Високий	Низький
8	Редагування даних тварини	FR-8	Середній	Низький
9	Видалення тварини	FR-9	Середній	Середній
10	Пошук ветеринарів за критеріями	FR-10	Високий	Низький
11	Перегляд детального профілю ветеринара	FR-11	Високий	Низький
12	Редагування професійного профілю	FR-12	Середній	Середній
13	Створення/додавання доступних слотів розкладу	FR-13	Високий	Середній
14	Редагування/видалення слотів розкладу	FR-14	Середній	Середній
15	Перегляд власного розкладу та записів	FR-15	Високий	Низький
16	Запис на онлайн-консультацію (Власник)	FR-16	Високий	Середній
17	Скасування запланованої консультації	FR-17	Середній	Середній
18	Перегляд списку своїх консультацій (майбутніх/минулих)	FR-18	Високий	Низький
19	Проведення онлайн-консультації (відкриття чату)	FR-19	Високий	Середній
20	Надсилання текстових повідомлень у чаті	FR-20	Високий	Низький
21	Надсилання зображень у чаті	FR-21	Середній	Середній

Продовження таблиці 2.16

22	Отримання та відображення повідомлень/зображень у чаті	FR-22	Високий	Низький
23	Аналіз симптомів тварини за допомогою ШІ	FR-23	Високий	Високий
24	Збереження результатів аналізу симптомів	FR-24	Середній	Низький
25	Перегляд історії аналізів симптомів	FR-25	Середній	Низький
26	Отримання та відображення системних сповіщень	FR-26	Середній	Низький

Таблиця 2.17 – Перелік функціональних вимог до програмного забезпечення

ID вимоги	Назва та опис
FR-1	Реєстрація нового користувача. Неавторизований користувач має можливість створити новий обліковий запис, надавши необхідні дані (ім'я, email, пароль) та, за бажанням, вказавши тип акаунту «Ветеринар» із заповненням відповідних професійних даних.
FR-2	Авторизація існуючого користувача. Користувач із зареєстрованим обліковим записом може увійти в систему, зазначивши свої ідентифікаційні дані (email і пароль)
FR-3	Вихід з облікового запису. Авторизований користувач має можливість безпечно завершити свій сеанс в системі.
FR-4	Редагування загальних даних профілю користувача. У користувача є опція оновити основну інформацію профілю, включно з ім’ям та електронною поштою.
FR-5	Зміна пароля користувача. Авторизований користувач може оновити свій пароль, підтверджуючи чинний пароль.

Продовження таблиці 2.17

FR-6	Додавання нової тварини (Власник). Власник може внести дані про свою тварину: кличку, вид, породу, вік, стать та іншу важливу інформацію, включно з фото.
FR-7	Перегляд списку своїх тварин (Власник). Власник має змогу бачити перелік усіх зареєстрованих у профілі тварин з коротким описом.
FR-8	Редагування даних тварини (Власник). Власник тварини може змінити раніше введену інформацію про домашню тварину.
FR-9	Видалення тварини (Власник). Власник тварини має змогу видалити запис про тваринку зі свого профілю.
FR-10	Пошук ветеринарів за критеріями. Користувач може здійснювати пошук ветеринарів за іменем або фільтрувати результати за спеціалізацією, видом тварин тощо. Для звуження результатів пошуку.
FR-11	Перегляд детального профілю ветеринара. Користувач має змогу ознайомитися з повною інформацією про фахівця: опис, досвід, освіта, напрямки діяльності, доступний графік.
FR-12	Редагування професійного профілю (Ветеринар). Ветеринар може оновлювати специфічну для його професії дані: фото, опис, спеціалізацію, досвід, освіту, види тварин, з якими працює, та проблеми, які він лікує.
FR-13	Створення/додавання доступних слотів розкладу (Ветеринар). Ветеринар може додавати нові доступні часові вікна до свого розкладу для запису на онлайн-консультації, вказуючи дату та час.
FR-14	Редагування/видалення слотів розкладу (Ветеринар). Ветеринар може змінювати або вилучати наявні вільні часові вікна в особистому календарі.

Продовження таблиці 2.17

FR-15	Перегляд власного розкладу та записів (Ветеринар). Користувач-ветеринар має доступ до свого графіку, в якому відображаються доступні години та записи на консультацію.
FR-16	Запис на онлайн-консультацію (Власник). Власник тварини, переглядаючи профіль ветеринара, може обрати доступний час, прикріпити тваринку та підтвердити запис на консультацію.
FR-17	Скасування запланованої консультації. Будь-хто з учасників — Власник або Ветеринар — може відмінити консультацію відповідно до правил скасування.
FR-18	Перегляд списку своїх консультацій (майбутніх/минулих). Авторизований користувач переглядає перелік консультацій, поділений на завершені та заплановані, з детальною інформацією.
FR-19	Проведення онлайн-консультації (відкриття чату). Учасники запланованої консультації можуть відкрити чат для обміну повідомленнями в режимі реального часу.
FR-20	Надсилання текстових повідомлень у чаті. Користувачі під час онлайн-сесії можуть вводити та передавати текстові повідомлення через чат.
FR-21	Надсилання зображень у чаті. Учасники онлайн-консультації мають змогу прикріплювати та надсилати файли зображень у чаті.
FR-22	Отримання та відображення повідомлень/зображень у чаті. Інтерфейс чату має динамічно оновлюватися, відображаючи нові текстові повідомлення та зображення
FR-23	Аналіз симптомів тварини за допомогою ШІ (Власник). Власник тварини має можливість вибрати тваринку, вказати спостережувані симптоми та отримати від системи попередній список імовірних діагнозів, рівень їх серйозності та загальні рекомендації.

Продовження таблиці 2.17

FR-24	Збереження результатів аналізу симптомів (Власник). Отримані висновки про стан тварини можуть бути збережені для подальшого ознайомлення або консультації з ветеринаром.
FR-25	Перегляд історії аналізів симптомів (Власник). Власник тварини має доступ до архіву раніше збережених результатів аналізу симптомів для своїх тварин.
FR-26	Отримання та відображення системних сповіщень. Авторизований користувач має отримувати та бачити сповіщення про важливі події (нагадування, скасування, тощо).

ID вимоги	UC-1	UC-2	UC-3	UC-4	UC-5	UC-6	UC-7	UC-8	UC-9	UC-10	UC-11	UC-12	UC-13	UC-14	UC-15
FR-1	+														
FR-2		+													
FR-3						+									
FR-4					+										
FR-5					+										
FR-6							+								
FR-7							+								
FR-8							+								
FR-9							+								
FR-10			+												
FR-11				+											
FR-12													+		
FR-13														+	
FR-14														+	
FR-15														+	
FR-16									+						
FR-17											+				
FR-18										+					
FR-19												+			
FR-20												+			
FR-21												+			
FR-22												+			
FR-23								+							
FR-24								+							
FR-25								+							
FR-26															+

Рисунок 2.2 – Матриця трасування вимог

2.3 Розроблення нефункціональних вимог

Окрім реалізації основних функцій, стабільна та якісна робота вебзастосунку значною мірою обумовлюється дотриманням

нефункціональних вимог, які визначають його експлуатаційні характеристики. У межах цього дипломного проєкту ключовими серед них є такі вимоги:

– Продуктивність. Застосунок має забезпечувати зручну та швидку взаємодію з користувачем. Завантаження основних сторінок повинно відбуватися протягом не більше ніж 3 секунд, а реакція на базові дії користувача (наприклад, фільтрація чи збереження інформації) — у межах 1–2 секунд. Затримка передачі повідомлень не повинна перевищувати 1-2 секунди. Для компонента діагностики симптомів на основі штучного інтелекту очікується, що отримання результатів триватиме не довше 10–15 секунд. На початковому етапі впровадження система повинна стабільно обслуговувати мінімум 100 одночасно активних відвідувачів.

– Надійність. Оскільки система оперує медичними даними, високий рівень надійності є надзвичайно важливим. Платформа має гарантувати збереження цілісності й узгодженості інформації, здійснюючи перевірку введених даних як на стороні клієнта, так і на сервері. Це зменшує ймовірність виникнення помилок через некоректні дії користувачів. Також система повинна правильно реагувати на програмні та мережеві збої, повідомляючи про них у зрозумілій формі, не перериваючи виконання критично важливих функцій.

– Захищеність. Безпека персональних відомостей користувачів і інформації про їхніх домашніх улюбленців є пріоритетною. Доступ до функціональних можливостей та даних повинен здійснюватися через перевірку ідентичності та визначення прав доступу відповідно до ролі (наприклад «Власник тварини», «Ветеринар»). Паролі користувачів мають зберігатися у вигляді хешованих значень у зашифрованому форматі.

– Зручність взаємодії. Інтерфейс користувача має бути інтуїтивно зрозумілим, логічно організованим та простим у користуванні навіть для нових відвідувачів. Вебзастосунок повинен мати адаптивний вигляд, що гарантує його коректне відображення на різних пристроях (персональні комп’ютери, планшети, мобільні телефони) та у найпопулярніших браузерах. Користувачі

мають отримувати чіткий зворотний зв’язок на свої дії, а доступ до основних можливостей повинен бути легким і не вимагати зайвих зусиль.

– Масштабованість. Архітектурне рішення системи має передбачати можливість поступового розширення обчислювальних можливостей серверної частини, що дозволить підтримувати стабільну роботу при зростанні числа користувачів або навантаження.

– Доступність. Сервіс має бути працездатним не менше ніж 99,5% часу протягом календарного року. Це означає, що допустимий час недоступності через технічні проблеми має бути мінімальним, а відновлення ключових функцій — не перевищувати однієї години з моменту виявлення збою.

2.4 Аналіз системних вимог

Для належної роботи клієнтської частини вебзастосунку важливою умовою є її підтримка сучасними інтернет-оглядачами, такими як Google Chrome, Mozilla Firefox, Apple Safari та Microsoft Edge (на базі Chromium), які відповідають актуальним стандартам вебтехнологій — HTML5, CSS3 і JavaScript (версія ES6 або новіша). Це гарантує коректне відображення інтерфейсних елементів, стабільне функціонування інтерактивних компонентів і комфортне користування на різноманітних пристроях.

Тип операційної системи користувача (Windows, macOS, Linux для десктопів; Android або iOS для мобільних пристроїв) не є визначальним фактором, однак передбачається, що застосунок буде стабільно функціонувати на всіх сучасних та оновлених версіях ОС, які підтримують зазначені браузери.

Наявність постійного та надійного з’єднання з Інтернетом є критично важливою умовою, оскільки вебзастосунок працює у режимі онлайн і потребує безперервного обміну інформацією з серверною частиною. Це стосується процесів авторизації, завантаження даних, обміну повідомленнями у чаті та використання інструменту штучного інтелекту для оцінки симптомів. Для базової функціональності достатньо швидкості з’єднання на рівні 1 Мбіт/с,

однак для повноцінної взаємодії з мультимедійним контентом та проведення онлайн-консультацій бажано мати інтернет зі швидкістю від 10 Мбіт/с.

2.5 Аналіз економічних показників програмного забезпечення

Оцінка економічних характеристик є важливою частиною процесу розроблення ПЗ, що дозволяє визначити трудомісткість і фінансові витрати. У даному проєкті застосовано методологію Use Case Points (UCP), яка ґрунтується на оцінюванні функціональної складності системи.

1) Ідентифікація та класифікація акторів (UAW)

   У системі визначено 4 основні актори:

   - Неавторизований користувач (Гість): Простий актор (взаємодіє через елементарний графічний інтерфейс без АР) - 1 бал.
   - Власник тварини: Середній актор (використовує GUI та виконує низку транзакцій) - 2 бали.
   - Ветеринар: Середній/Складний актор (керує розкладом, здійснює консультації, редагує профіль через інтерфейс) 3 бали.
   - Сервер ШІ (ML API): Розглядатися як простий актор (взаємодія здійснюється виключно через API) - 1 бал.

   Результат обчислення UAW: UAW = 1 + 2 + 3 + 1 = 7 UAW.

2) Класифікація варіантів використання (UUCW):

Таблиця 2.18 — Класифікація варіантів використання

ID Use case	Назва	Орієнтовна К-сть Транзакцій	Складність	Бали
UC-01	Реєстрація користувача	5-7	Середній	10
UC-02	Авторизація користувача	3-4	Простий	5

Продовження таблиці 2.18

UC-03-05	Пошук, перегляд, керування профілем	4-6	Середній	10
UC-06	Вихід з акаунту	1-2	Простий	5
UC-07	Керування тваринами (Власник)	8-12	Складний	15
UC-08-11	Аналіз ШІ, запис, перегляд консультації, скасування	5-8	Середній	10
UC-12	Проведення онлайн-консультації (Чат)	>10	Складний	15
UC-13-14	Профіль, розклад ветеринара	5-8	Середній/Складний	10/15
UC-15	Отримання/Перегляд сповіщень	2-4	Простий	5
Разом UUCW				150

3) Обчислення нескоригованої кількості точок варіантів використання (UUCP):

$$\mathrm{UUCP} = \mathrm{UAW} + \mathrm{UUCW} = 7 + 150 = 157$$

4) Визначення фактора технічної складності (TCF):

Таблиця 2.19 – Оцінка технічних факторів (T1-T13)

ID	Фактор технічної складності	Оцінка (0-5)	Обґрунтування Оцінки
T1	Розподілена система	3	Вебзастосунок, із окремим ML API, базою даних і сокет-сервером

Продовження таблиці 2.19

T2	Продуктивність / час відгуку	4	Висока швидкодія чату, прийнятний час обробки AI, оперативне завантаження сторінок
T3	Ефективність користувача	4	Зручна онлайн-взаємодія
T4	Внутрішня обробка	3	Бронювання, розклад, ШІ, чат мають певну складність
T5	Повторне використання	3	Компонентний підхід
T6	Легкість встановлення	2	Простий деплоймент із налаштуванням
T7	Легкість експлуатації (операційна підтримка)	2	Хмарні платформи, але потрібен моніторинг
T8	Множинне використання	1	Один основний вебзастосунок
T9	Гнучкість	3	Гнучка база, складність зміни логіки ШІ
T10	Паралельність	3	Одночасна взаємодія із API, сокети
T11	Безпека	4	Конфіденційність даних
T12	Залежність від сторонніх бібліотек/сервісів	2	—
T13	Навчання користувачів	2	Інтуїтивний, потреба в ознайомленні з новим функціоналом
TFactor - сума 36			

$$TCF = 0.6 + (0.01 * TFactor)\ 0.6 + (0.01 * 36) = 0.6 + 0.3 = 0.96$$

5) Визначення фактора складності середовища (ECF):

Фактори середовища оцінюються з урахуванням наявного досвіду у веб розробці, високого рівня зацікавленості, притаманного дипломним роботам, а

також використання інноваційних і добре задокументованих технологічних рішень. Водночас, певні труднощі створюються через впровадження новітнього компонента штучного інтелекту, можливу варіативність вимог на початкових стадіях та обмеження, пов’язані з освітнім характером проєкту. Сукупність цих факторів формує оціночний коефіцієнт середовища EFactor = 24.

$$ECF = 1.4 + (-0.03 * EFactor)$$

$$ECF = 1.4 + (-0.03 * 24) = 1.4 - 0.72 = 0.68$$

6) Розрахунок скоригованої кількісті точок варіантів використання (UCP):

$$UCP = UUCP * TCF * ECF$$

$$UCP = 157 * 0.96 * 0.68 = 141.3 * 0.68 \approx 102$$

Для визначення приблизної кількості людино-годин зазвичай використовується множник продуктивності, типовий для подібних проєктів. Це число було запропоновано Густавом Карнером. Тому припустимо, значення продуктивності становить 20 годин / 1 UCP [29].

Загальна трудомісткість:

$$Effort_H = UCP * PF = 102\,UCP * 20\frac{\text{год}}{UCP} = 2040\ \text{людино} - \text{годин}$$

Для оцінки фінансових витрат використаємо умовну середню погодинну заробітну плату програміста, яка становить 8 USD/год.

Фінансові витрати на розроблення програмного забезпечення:

$$\text{Вартість розробки} = 2040\ \text{год} \times 8\ USD = 16320\ USD$$

Отже, орієнтовна вартість реалізації проєкту складає 16 320 доларів США, а обсяг трудових витрат — 2040 людино-годин.

2.6 Постановка завдання на розробку програмного забезпечення

Метою розробки є підвищення якості та доступності ветеринарної допомоги для власників домашніх тварин шляхом створення інструменту для попередньої діагностики стану здоров'я їхніх улюбленців та забезпечення зручного онлайн-зв'язку з кваліфікованими ветеринарними фахівцями.

Для реалізації поставленої мети визначено такі основні цілі, що відповідають функціональним вимогам першого рівня:

- Забезпечення реєстрації та авторизації користувачів: Реалізувати надійний механізм для реєстрації нових користувачів (з розрізненням ролей Власника тварини та Ветеринара) та захищеної авторизації існуючих користувачів в системі.
- Впровадження керування обліковими записами: Надати змогу користувачам (Власникам та Ветеринарам) створювати, переглядати та редагувати свої персональні та професійні дані профілю.
- Розробка системи керування даними про тварин: Дозволити Власникам тварин додавати, переглядати, редагувати та видаляти інформацію про своїх домашніх улюбленців.
- Створення інструменту пошуку: Реалізувати функціонал пошуку ветеринарів за різними критеріями (ПІБ,, спеціалізація, види тварин) з можливістю ознайомлення з розширеною інформацією їх профілів.
- Інтеграція системи розкладів для ветеринарів: Надати Ветеринарам інструменти для формування, редагування та управління своїм робочим розкладом та доступними часовими вікнами для консультацій.
- Розробка модуля запису на онлайн-консультації: Забезпечити Власникам тварин можливість переглядати розклад ветеринарів та резервувати зручні часові вікна для онлайн-консультацій.
- Створення системи онлайн-консультацій (текстовий чат): Реалізувати функціонал чату в реальному часі для проведення онлайн-консультацій між Власником тварини та Ветеринаром, з можливістю обміну текстовими повідомленнями та зображеннями.
- Інтеграція модуля аналізу симптомів на основі AI: Реалізувати систему, що дозволяє Власникам тварин вводити спостережувані симптоми своєї тварини та отримувати попередній список імовірних діагнозів, рівень їх серйозності та загальні рекомендації.

– Розробка системи сповіщень: Передбачити механізм повідомлення користувачів про ключові події — нагадування, скасування консультацій.

Для реалізації вищенаведених цілей необхідно вирішити такі технічні завдання:

- Спроектувати та реалізувати базу даних для зберігання інформації про користувачів, тварин, ветеринарів, їхні профілі, розклади, консультації, повідомлення чату, а також результати роботи AI-модуля.
- Розробити інтерфейс користувача (Frontend веб застосунку з інтуїтивно зрозумілим та адаптивним користувацьким інтерфейсом.
- Розробити серверну частину (Backend) за допомогою API-маршрутів для обробки бізнес-логіки, взаємодії з базою даних та автентифікації/авторизації користувачів.
- Розробити та інтегрувати окремий серверний модуль для аналізу симптомів та надання попередніх діагностичних гіпотез.
- Інтегрувати функціонал обміну повідомленнями в реальному часі.
- Здійснити заходи щодо захисту персональних даних і забезпечення конфіденційності користувацької інформації.
- Провести всебічне тестування застосунку для виявлення технічних недоліків та усунення помилок.
- Підготувати застосунок до розміщення на хостинг-платформі для подальшого розгортання.

Реалізація цих завдань дозволить створити повноцінний та функціональний вебзастосунок «Vai», що відповідатиме поставленій меті та цілям дипломної роботи.

Висновки до розділу

У цьому розділі здійснено аналіз вимог до вебзастосунку «Vai», що є ключовим підготовчим етапом перед початком проєктування системи. Було сформульовано функціональні, нефункціональні та системні вимоги, а також здійснено первинну оцінку обсягу робіт, необхідних для реалізації проєкту.

Ретельно досліджено предметну область і визначено основних акторів: «Неавторизований користувач», «Власник тварини», «Ветеринар». Для наочного відображення сценаріїв взаємодії між учасниками системи створено діаграму варіантів використання, що охоплює реєстрацію, керування профілями, пошук ветеринарів, онлайн-консультації з чатом та зображеннями, а також модуль аналізу симптомів на основі ШІ.

Кожен варіант використання був формалізований у вигляді таблиць з метою, передумовами, потоками подій та постумовами. Це дозволило структурувати функціональну очікувану поведінку системи. На основі варіантів використання розроблено функціональні вимоги з ідентифікаторами, пріоритетами та ризиками, а також побудовано матрицю трасування.

Нефункціональні характеристики включають продуктивність, надійність, захищеність, зручність інтерфейсу, масштабованість і доступність сервісу (99.5%). Окремо визначено технічні вимоги до клієнтської частини (вебінтерфейс) і серверної частини.

За допомогою методу UCP було розраховано обсяг витрат часу та фінансових ресурсів, необхідних для реалізації проєкту, що дозволяє зробити висновки щодо його економічної ефективності як для кінцевих користувачів, так і для розробника.

За результатами розділу сформовано технічне завдання на розробку програмного забезпечення.

3 КОНСТРУЮВАННЯ ТА РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Архітектура програмного забезпечення

Для реалізації програмного рішення, спрямованого на раннє виявлення захворювань у домашніх тварин за участю ШІ та інтерактивного консультування, було обрано поширену й перевірену практикою архітектурну модель типу «клієнт-сервер». Такий підхід передбачає розмежування функцій між фронтендом, який відповідає за візуалізацію інтерфейсу та взаємодію з кінцевим користувачем, і бекендом, що забезпечує виконання бізнес-логіки, опрацювання даних, а також взаємодію з базою даних і зовнішніми сервісами. Клієнтська та серверна частини функціонують як відносно незалежні компонентищо обмінюються інформацією через мережу згідно із заздалегідь визначеними протоколами.

Клієнтська частина вебзастосунку «Vai» реалізована як односторінковий застосунок (SPA) який доступний користувачам через стандартні веббраузери. Вона забезпечує динамічний та інтерактивний користувацький інтерфейс, надсилаючи запити до серверної частини для отримання, оновлення чи видалення даних, і відображає відповідну інформацію користувачеві.

Серверна частина відповідає за опрацювання запитів, реалізацію прикладної логіки, забезпечення інформаційної безпеки, а також автентифікацію та авторизацію користувачів. Вона також підтримує взаємодію з системою керування базами даних та інтеграцію з іншими підключеними сервісами, зокрема з модулем штучного інтелекту.

Передача даних між клієнтом і сервером відбувається в режимі «запит–відповідь» (request–response), із використанням безпечного протоколу HTTPS та формату обміну JSON. Для реалізації обміну повідомленнями в реальному часі, зокрема в межах функціоналу чату, застосовується протокол WebSocket, реалізований за допомогою бібліотеки Socket.IO.

З метою візуального представлення архітектурної моделі застосовується підхід C4 Model діаграми [30]. Цей підхід дозволяє послідовно декомпозитувати систему на різних рівнях абстрактного представлення, починаючи від загального контексту і завершуючи деталізацією окремих компонентів, що робить архітектуру зрозумілою для розробки. Далі будуть представлені діаграми першого та другого рівнів моделі C4.

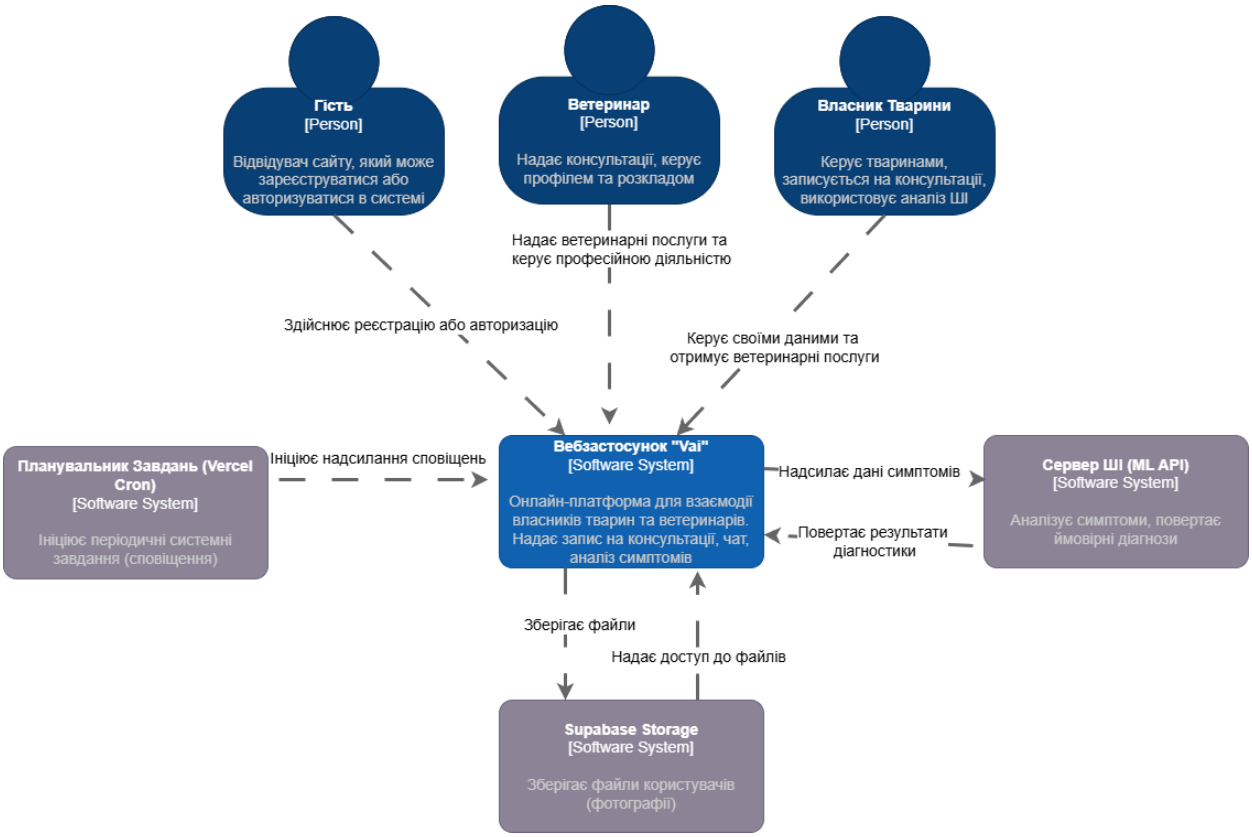


Рисунок 3.1 – 1 рівень. Діаграма контексту системи

На рисунку 3.1 наведено контекстну діаграму, яка представляє систему як єдиний логічний блок та демонструє її взаємозв’язки з користувачами й зовнішніми сервісами. Основними акторами, які взаємодіють з вебзастосунком, є Гість (незареєстрований користувач), Власник тварини та Ветеринар. Гість має можливість переглядати загальнодоступну інформацію та зареєструватися/авторизуватися. Власник тварини може управляти профілями своїх улюбленців, записуватись на консультації, вести чат і скористатися функцією первинного аналізу симптомів. Ветеринар, у свою

чергу, має змогу адмініструвати професійний профіль, формувати графік прийому та проводити онлайн-консультації.

До зовнішніх компонентів належать:

- Сервер Штучного Інтелекту (ML API) — забезпечує функціонал первинної діагностики;
- Supabase Storage — відповідає за зберігання мультимедійних файлів;
- Планувальник Завдань (Vercel Cron) — виконує регулярні фонові дії, наприклад, надсилання повідомлень користувачам.

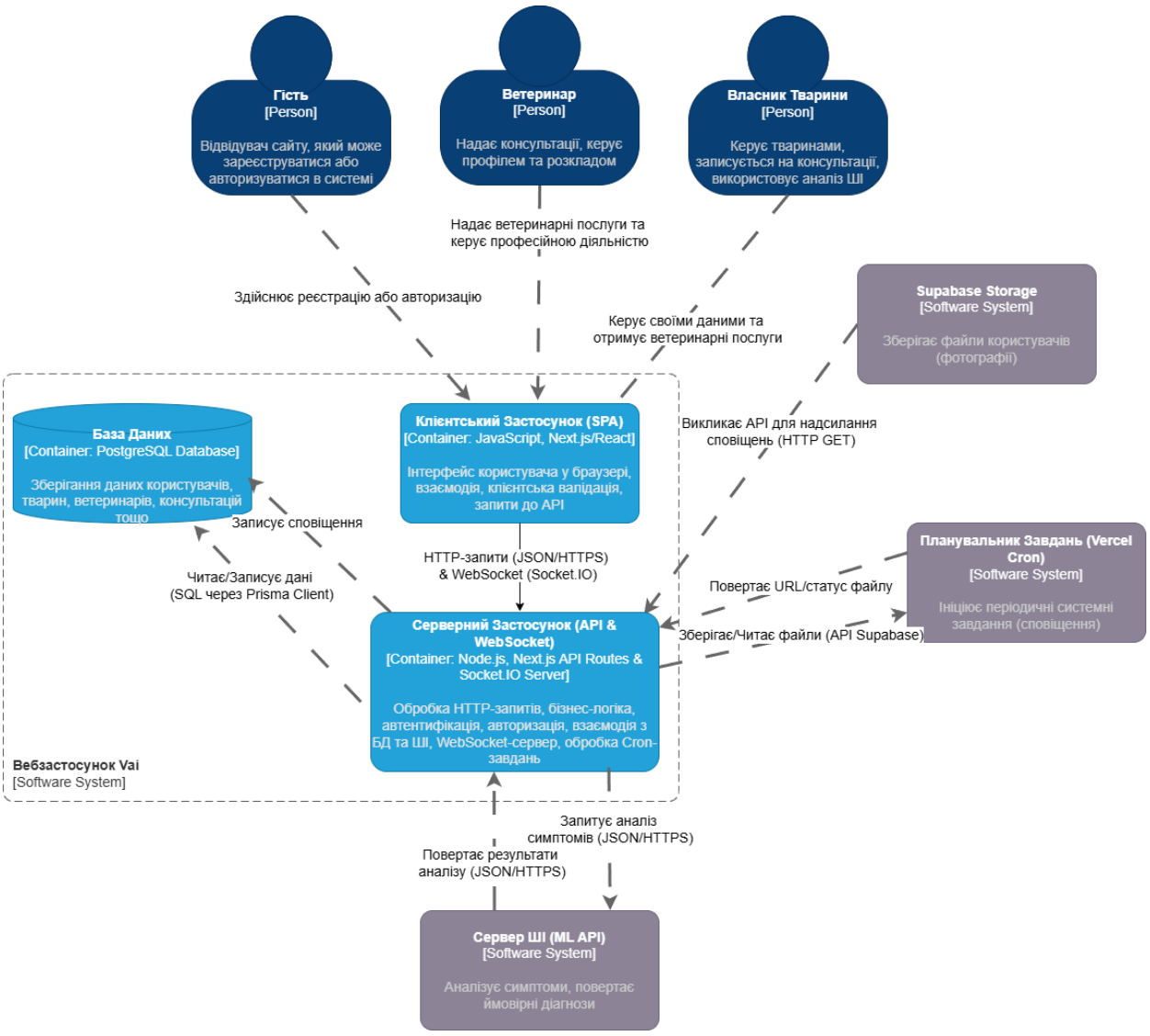


Рисунок 3.2 – 2 рівень. Діаграма контейнерів системи

На рисунку 3.2 представлено діаграму контейнерів, яка деталізує загальну структуру внутрішніх компонентів системи. Вона відображає ключові логічні блоки (контейнери) й способи їхньої взаємодії.

Актори системи взаємодіють з клієнтським застосунком (SPA), розробленим за допомогою фреймворку Next.js (на основі React.js) з використанням мови TypeScript. Цей компонент відповідає за обробку подій, формування інтерфейсу й комунікацію із серверною частиною.

Серверний застосунок, що також побудований з використанням Next.js (API Routes, WebSocket Server) і TypeScript, виконує ключові серверні функції: реалізацію бізнес-логіки, автентифікацію та авторизацію (на базі NextAuth.js), обробку з'єднань WebSocket, зв'язок із реляційною Базою Даних PostgreSQL через Prisma ORM [31], а також взаємодію з іншими контейнерами, такими як ML-сервер і файлове сховище.

Модуль ШІ, реалізований з використанням Python/Flask, опрацьовує запити на аналіз симптомів. Зображення зберігаються у Supabase Storage (Buckets). Фонові задачі виконуються за допомогою Планувальника, який активує відповідні API-ендпоінти для запуску періодичних процесів, надсилання сповіщень.

3.2 Архітектурні рішення та обґрунтування вибору засобів розробки

У процесі створення веб застосунку основною метою було побудувати зручну, надійну та масштабовану платформу, яка б відповідала як технічним, так і практичним потребам користувачів. Для досягнення цього була обрана класична клієнт-серверна архітектура, що дозволяє відокремити логіку взаємодії з інтерфейсом (на клієнтській стороні) та обробку запитів, бізнес-логіку й доступ до бази даних (на серверній частині). На відміну від підходів, які покладаються винятково на клієнтську сторону, це рішення покращує керованість кодової бази, підвищує захищеність системи та дозволяє масштабувати окремі її компоненти незалежно.

Окремий акцент зроблено на керуванні користувачами та їх автентифікації. Замість використання сторонніх провайдерів «як є» для всіх аспектів, було прийнято рішення реалізувати власний механізм реєстрації та авторизації. Це дозволило зберегти повний контроль над даними користувачів

та чітко реалізувати розмежування ролей: «Власник тварини» та «Ветеринар». Безпечне зберігання облікових даних забезпечується завдяки хешуванню паролів з використанням алгоритму bcrypt. Для організації сесій та захисту API-ендпоінтів інтегровано бібліотеку NextAuth.js, яка, порівняно з ручною реалізацією JWT-логіки, надає готовий, перевірений часом набір інструментів та дозволяє зручно реалізувати OAuth-автентифікацію (наприклад, через Google), що розглядається як реалізація в майбутньому.

Додатковий функціонал реалізовано завдяки інтеграції з окремими сервісами. Модуль первинного аналізу симптомів реалізований у вигляді мікросервісу на Flask (Python) що виступає у ролі сервера машинного навчання (ML API). Вибір Python для цього завдання був очевидним через його домінування в галузі Data Science та наявність широкого вибору бібліотек, таких як TensorFlow та Keras [32], на відміну від JavaScript, де екосистема для складного ML менш розвинена. Для трансформації категоріальних характеристик, таких як вид тварини та встановлені діагнози, у числовий вигляд, сумісний із нейронною мережею, було застосовано відповідні кодувальники (LabelEncoder і OneHotEncoder), що входять до складу бібліотеки Scikit-learn. Процес навчання моделі багаторівневого персептрона (MLP) виконувався з використанням алгоритму оптимізації Adam. З метою уникнення перенавчання та забезпечення автоматичної зупинки тренування при відсутності покращення результатів на валідаційній вибірці, було впроваджено механізм дострокового завершення навчання (EarlyStopping). Винесення модуля в окремий сервіс дозволяє масштабувати його незалежно від основного застосунку. Передача інформації реалізована за допомогою HTTP POST-запитів у форматі JSON. Для збереження файлів користувачів (фото) використано Supabase Storage, яке було обрано як інтегроване рішення в екосистемі Supabase, що робить його привабливішим порівняно з конфігуруванням сторонніх рішень на кшталт AWS S3 або аналогічного сервісу з нуля. Регулярні процеси (наприклад, нагадування або сповіщення) автоматизовані за допомогою Vercel Cron.

Технологічний стек було обрано з урахуванням ефективності розробки та якості кінцевого рішення. Для хостингу клієнтської частини використовуються серверniless-платформи Vercel (Next.js), а для Python-сервісів — Render, які забезпечують автоматичне масштабування та високу доступність без потреби в ручному адмініструванні. Передача даних зашифрована через HTTPS-протокол. Для реалізації обміну повідомленнями в реальному часі застосовується Socket.IO, який забезпечує додаткові переваги (автоматичне перепідключення, fallback-механізми), на відміну від базового WebSocket.

Для обробки структурованих даних було обрано реляційну СУБД PostgreSQL, яка доступна через інтерфейс Supabase. У порівнянні з нереляційними альтернативами (наприклад, MongoDB), це рішення краще гарантує цілісність інформації завдяки строгому дотриманню схем і підтримці ACID-властивостей, що критично для роботи з конфіденційними даними користувачів та їхніх тварин.

Основний застосунок реалізовано з використанням TypeScript у поєднанні з Next.js (React). Розглядалися також інші фреймворки — Angular, Vue.js, однак перевагу надано Next.js через його високу гнучкість, широку підтримку спільноти та наявний досвід роботи з ним, що дозволяє зменшити витрати часу на розробку. TypeScript, завдяки статичній типізації, дозволяє уникати багатьох помилок на ранніх етапах, що особливо цінно у великих проєктах, на відміну від чистого JavaScript. Як ORM-систему обрано Prisma, яка, порівняно з TypeORM чи Sequelize, пропонує вищий рівень типобезпеки, генерацію типів з бази даних і більш зрозумілий API.

Фронтенд побудовано на основі shadcn/ui, що поєднує у собі переваги Radix UI та Tailwind CSS [33]. На відміну від громіздких бібліотек інтерфейсів (таких як Material UI чи Ant Design), це рішення забезпечує максимальну адаптивність та легку кастомізацію, зберігаючи при цьому акцент на доступність.

У ролі основного інструменту розробки використовується Visual Studio Code (VS Code), який, порівняно з важчими середовищами на кшталт WebStorm або IntelliJ IDEA, надає оптимальний баланс між швидкістю, гнучкістю й можливостями налаштування завдяки розгалуженій екосистемі розширень.

3.3 Конструювання програмного забезпечення

3.3.1 Опис структури бази даних

Після проведеного аналізу вимог у попередніх розділах було розроблено логічну та фізичну структуру бази даних. Як і раніше зазначалося, для зберігання даних обрано реляційну систему управління базами даних PostgreSQL. ER-діаграма розробленої бази даних, яка візуалізує таблиці та зв'язки між ними, наведена у графічному матеріалі, креслення 3.

Для управління базою даних було обрано систему керування реляційними базами даних PostgreSQL у поєднанні з Prisma ORM. Усі моделі даних були декларативно описані у файлі schema.prisma, де визначено назви таблиць, типи полів та типи зв’язків між сутностями (один-до-одного, один-до-багатьох, багато-до-багатьох). Інструмент Prisma дозволив забезпечити типізовану взаємодію з базою даних на серверній частині застосунку, що розроблялась мовою TypeScript у середовищі Next.js. Створення та оновлення фізичної структури бази виконувалося за допомогою команди prisma migrate dev, яка автоматично генерує SQL-міграції на основі змін у схемі та застосовує їх до PostgreSQL, забезпечуючи узгодженість між описаними моделями та реальною структурою даних.

Детальний опис кожної таблиці бази даних, її полів, типів даних та обмежень наведено у таблицях 3.1 - 3.11

Таблиця 3.1 – Опис таблиці User

Назва поля	Тип даних	Опис	Обмеження
id	uuid	Унікальний ідентифікатор користувача	primary key, unique
email	text	Електронна пошта користувача	unique, not null
password	text	Хешований пароль користувача	not null
username	text	Унікальне ім'я користувача в системі	unique, not null
type	enum	Тип користувача (OWNER, VET)	not null
createdAt	timestamp	Дата та час створення облікового запису	not null, default now()

Таблиця 3.2 – Опис таблиці VetProfile

Назва поля	Тип даних	Опис	Обмеження
id	uuid	Унікальний ідентифікатор профілю ветеринара	primary key, unique
userId	uuid	Ідентифікатор користувача (User), якому належить цей профіль	unique, not null, foreign key (user)
photoUrl	text	URL фотографії профілю ветеринара	
about	enum	Текстовий опис ветеринара «про себе»	

Продовження таблиці 3.2

specialization	text	Основна спеціалізація ветеринара	
experience	text	Опис досвіду роботи ветеринара	
education	text	Інформація про освіту ветеринара	
createdAt	timestamp	Дата та час створення профілю	not null, default now()

Таблиця 3.3 – Опис таблиці Pet

Назва поля	Тип даних	Опис	Обмеження
id	uuid	Унікальний ідентифікатор профілю ветеринара	primary key, unique
ownerId	uuid	Ідентифікатор користувача (User), який є власником тварини (зовнішній ключ)	unique, not null, foreign key (user)
name	text	Ім'я (кличка) тварини	not null
type	text	Вид тварини	not null
breed	text	Порода тварини	
age	integer	Вік тварини в роках	
gender	text	Стать тварини	
weight	double precision	Вага тварини в кілограмах	
history	text	Медична історія тварини	
vaccination	text	Інформація про вакцинацію тварини	
photoUrls	text[]	Масив URL фотографій тварини	not null, default []

Таблиця 3.4 – Опис таблиці Schedule

Назва поля	Тип даних	Опис	Обмеження
id	uuid	Унікальний ідентифікатор запису часового вікна	primary key, unique
vetId	uuid	Ідентифікатор профілю ветеринара (VetProfile), якому належить цей слот (зовнішній ключ)	not null, foreign key (vetprofile)
date	date	Дата, на яку заплановано слот (тільки дата, без часу)	not null
from	text	Час початку слоту (наприклад, «09:00»)	not null
to	text	Час закінчення слоту (наприклад, «09:30»)	not null
isBooked	boolean	Прапорець, що вказує, чи заброньовано цей слот	not null, default false

Таблиця 3.5 – Опис таблиці Consultation

Назва поля	Тип даних	Опис	Обмеження
id	uuid	Унікальний ідентифікатор користувача	primary key, unique
userId	uuid	Ідентифікатор власника тварини (User)	not null, foreign key (user)
vetId	uuid	Ідентифікатор профілю ветеринара (VetProfile)	not null, foreign key (vetprofile)

Продовження таблиці 3.5

petId	uuid	Ідентифікатор тварини (Pet), для якої консультація	not null, foreign key (pet)
scheduleId	uuid	Ідентифікатор слоту розкладу (Schedule), до якого прив'язана консультація	foreign key (schedule)
scheduledAt	timestamp	Заплановані дата та час початку консультації	not null
status	text	Поточний статус консультації (наприклад, «SCHEDULED», «COMPLETED», «CANCELLED_BY_CLIENT»)	not null
notificationSentAt	text	Час, коли було надіслано нагадування про консультацію	
createdAt	text	Дата та час створення запису про консультацію	not null, default now()

Таблиця 3.6 – Опис таблиці Message

Назва поля	Тип даних	Опис	Обмеження
id	uuid	Унікальний ідентифікатор користувача	primary key, unique
consultationId	uuid	Ідентифікатор консультації (Consultation), до якої належить повідомлення (зовнішній ключ)	not null, foreign key (consultation)

Продовження таблиці 3.6

senderId	uuid	Ідентифікатор користувача (User), який надіслав повідомлення (зовнішній ключ)	not null, foreign key (user)
text	text	Текстовий вміст повідомлення	
imageUrl	imageurl	URL зображення, прикріпленого до повідомлення	
imageMeta	jsonb	Метадані зображення (розмір, ім'я файлу, тип файлу)	
timestamp	varchar	Дата та час надсилання повідомлення	not null, default now()

Таблиця 3.7 – Опис таблиці SymptomAnalysis

Назва поля	Тип даних	Опис	Обмеження
id	uuid	Унікальний ідентифікатор результату аналізу	primary key, unique
petId	uuid	Ідентифікатор тварини (Pet), для якої проводився аналіз	not null, foreign key (pet)
symptomsProvided	text[]	Масив симптомів, введених користувачем	not null
unknownSymptoms	text[]	Масив симптомів, які не були розпізнані моделлю ШІ	not null, default []

Продовження таблиці 3.7

overallSeverity	text	Загальний визначений рівень важкості (наприклад, «Легкий», «Критичний»)	not null
triggeredWarning	boolean	Прапорець, чи було показано додаткове попередження через крихітну ймовірність (>15%) серйозного діагнозу	not null, default false
triggeredExplanation	text	Діагноз, який спричинив пояснення про підвищення загального рівня важкості	not null, default now()
createdAt	timestamp	Дата та час проведення аналізу	not null, default now()

Таблиця 3.8 – Опис таблиці SymptomPrediction

Назва поля	Тип даних	Опис	Обмеження
id	uuid	Унікальний ідентифікатор передбачення	primary key, unique
analysisId	uuid	Ідентифікатор аналізу симптомів (SymptomAnalysis), до якого належить це передбачення	not null, foreign key (symptomanalysis)
diagnosis	text[]	Назва передбаченого діагнозу	not null

Продовження таблиці 3.8

probability	text[]	Ймовірність даного діагнозу (від 0 до 1)	not null
severity	text	Рівень важкості, асоційований з цим конкретним діагнозом	not null

Таблиця 3.9 – Опис таблиці Notification

Назва поля	Тип даних	Опис	Обмеження
id	uuid	Унікальний ідентифікатор передбачення	primary key, unique
userId	uuid	Ідентифікатор користувача (User), якому адресовано сповіщення	not null, foreign key (user)
message	text	Текст сповіщення	not null
link	text	Посилання, пов'язане зі сповіщенням (наприклад, на сторінку консультації)	
isRead	boolean	Прапорець, чи прочитано сповіщення користувачем	not null, default false
timestamp	timestamp	Дата та час створення сповіщення	not null, default now()
consultationId	uuid	Рівень важкості, асоційований з цим конкретним діагнозом	foreign key (consultation)

Таблиця 3.10 – Опис таблиці VetConcern

Назва поля	Тип даних	Опис	Обмеження
id	uuid	Унікальний ідентифікатор запису проблеми/напрямку з яким працює ветеринар	primary key, unique
vetId	uuid	Ідентифікатор профілю ветеринара (VetProfile), з яким пов'язана ця проблема/напрямок	not null, foreign key (vetprofile)
name	text	Назва медичної проблеми або напрямку, з яким працює ветеринар (наприклад, «Дерматологія», «Кардіологія»)	not null

Таблиця 3.11 – Опис таблиці VetSpecies

Назва поля	Тип даних	Опис	Обмеження
id	uuid	Унікальний ідентифікатор запису виду тварин, з яким працює ветеринар	primary key, unique
vetId	uuid	Ідентифікатор профілю ветеринара (VetProfile), з яким пов'язаний цей вид тварин	not null, foreign key (vetprofile)
name	text	Назва виду тварин, з яким працює ветеринар (наприклад, «Собаки», «Коти», «Гризуни», «Птахи», тощо)	not null

Опис ключових утиліт, бібліотек та іншого стороннього програмного забезпечення, що використовується у розробці вебзастосунку, наведено в таблиці 3.12.

Таблиця 3.12 – Опис утиліт та бібліотек

№	Назва утиліти/бібліотеки	Опис застосування
1	Visual Studio Code	Основне середовище розробки з підтримкою TypeScript, JavaScript та Python, що забезпечує розширені функції для відлагодження, зміни структури коду (рефакторингу) та взаємодії з системами контролю версій, зокрема Git
2	Next.js	Фреймворк на базі React для побудови повноцінних вебрішень, що охоплюють як серверну, так і клієнтську логіку, включаючи маршрутизацію, серверний рендеринг та обробку API-запитів
3	React.js	Бібліотека JavaScript для створення інтерфейсів користувача з використанням компонентного підходу, що забезпечує інтерактивність і гнучкість
4	Node.js	Серверна платформа виконання JavaScript-коду, яка є середовищем для запуску Next.js та інших серверних компонентів
5	Prisma ORM	Інструмент об'єктно-реляційного відображення (ORM) для роботи з PostgreSQL, що забезпечує типобезпечну взаємодію з базою даних і спрощує керування її структурою.
6	PostgreSQL	Обрана реляційна СУБД

Продовження таблиці 3.12

7	NextAuth.js	Рішення для впровадження автентифікації та контролю доступу у застосунках, розроблених на Next.js
8	Zod	Бібліотека для типобезпечної перевірки та валідації структур даних як на фронтенді, так і на бекенді
9	bcrypt	Засіб для шифрування паролів за допомогою стійкого хешування, що підвищує безпеку облікових записів
10	Socket.IO	Бібліотека для організації комунікації в реальному часі між сервером і клієнтом, базується на WebSocket з fallback-механізмами
11	shadcn/ui	Колекція інтерфейсних компонентів, заснована на Radix UI та Tailwind CSS, орієнтована на швидку розробку сучасного UI
12	Sonner	Інструмент для створення повідомлень-сповіщень (toast notifications), що з’являються на інтерфейсі користувача
13	date-fns	Бібліотека для зручної обробки дат і часу, включаючи форматування та обчислення
14	TensorFlow/Keras	Інструменти для побудови, тренування та використання моделей глибинного навчання
15	Scikit-learn	Бібліотека машинного навчання, яка використовується для попередньої обробки даних та оцінювання моделей

Продовження таблиці 3.12

16	NumPy/Pandas	Інструменти на Python для аналітики даних та роботи з табличними структурами під час підготовки датасету
17	Joblib	Засіб для серіалізації Python-об’єктів, що забезпечує їх збереження і завантаження
18	Postman	Засіб для ручного тестування API-запитів під час налагодження серверної логіки

3.4 Аналіз безпеки даних

Забезпечення захисту даних користувачів та стійкості системи до кіберзагроз є пріоритетом у розробці вебзастосунку. З огляду на чутливість оброблюваної інформації — зокрема персональних даних та відомостей про здоров’я домашніх тварин — було реалізовано багаторівневу систему безпеки.

Механізм аутентифікації користувачів побудований на основі криптографічного алгоритму bcrypt, який застосовує «сіль» до кожного пароля перед хешуванням. Такий підхід робить непридатним застосування таблиць зі збереженими хешами (так званих rainbow tables) і значно ускладнює зловмисний підбір. Отримані хеші зберігаються в базі PostgreSQL, а при вході пароль повторно хешується для порівняння з наявним значенням.

Для авторизації та керування сесіями використовується бібліотека NextAuth.js, яка працює з JWT-токенами. Після успішної аутентифікації генерується підписаний токен з ID користувача та його роллю («Власник тварини», «Ветеринар»), що дозволяє реалізувати гнучку систему контролю доступу. Токен зберігається на клієнтській стороні та надсилається з кожним запитом через заголовок Authorization, забезпечуючи безперервну перевірку автентичності.

Комунікація між клієнтом і сервером, включно з інтеграцією модуля штучного інтелекту через API, здійснюється виключно через HTTPS, що

забезпечує шифрування передаваних даних і захист від їх перехоплення або підміни.

З боку сервера безпека зберігання реалізована через платформу Supabase, де база даних має обмеження доступу за унікальними обліковими записами та IP-адресами. Це запобігає несанкціонованому підключенню до сховища.

Для мінімізації ризиків, пов’язаних із поширеними вразливостями вебзастосунків, таких як XSS (міжсайтове впровадження скриптів) та CSRF (підробка міжсайтових запитів), застосовуються вбудовані захисні механізми Next.js. Додатково реалізовано ретельну перевірку вхідних даних на клієнтському і серверному рівнях за допомогою бібліотеки Zod, яка забезпечує валідацію та санітизацію введеної інформації.

Конфіденційні параметри, зокрема токени доступу, ключі для підпису JWT і облікові дані до бази, розміщені в змінних середовища, що дозволяє уникнути їхнього розголошення у вихідному коді та знижує ризик витоку інформації під час розгортання.

Висновки до розділу

У цьому розділі було докладно проаналізовано процес проєктування та реалізації програмного рішення. Центральним архітектурним рішенням стала клієнт-серверна модель, що забезпечила чітке розмежування функціональних блоків: фронтенд відповідає за користувацьку взаємодію, тоді як серверна логіка керує обробкою запитів та доступом до даних. Обмін даними реалізовано через захищений протокол HTTPS для стандартних запитів та технологію WebSocket для забезпечення динамічної взаємодії в режимі реального часу, зокрема для чату.

Архітектурна структура системи була візуалізована за допомогою діаграм C4 Model (рівні 1 та 2), що демонструють загальний контекст функціонування застосунку, основні складові (контейнери) та способи їх взаємодії. Також відображено інтеграцію з окремим мікросервісом модуля

штучного інтелекту, системою збереження файлів та сервісом для планування завдань (повідомлень).

Для створення основного застосунку (як інтерфейсної, так і серверної API частини) було використано мову TypeScript у поєднанні з фреймворком Next.js. Компонент модуля ШІ реалізовано мовою Python із використанням вебфреймворку Flask та бібліотек TensorFlow/Keras. У ролі системи керування базами даних виступає PostgreSQL, до якої здійснюється доступ через хмарний сервіс Supabase та ORM-інструмент Prisma. Користувацький інтерфейс реалізований за допомогою бібліотеки UI-компонентів shadcn/ui.

У межах технічного проєктування докладно розглянуто побудову алгоритму первинної діагностики, заснованого на нейронній мережі. Висвітлено етапи підготовки, обрання архітектури моделі, тренування та збереження відповідних артефактів. Також представлено структуру реляційної бази даних із поясненням ключових таблиць і зв’язків між ними.

Аспекти інформаційної безпеки були інтегровані на всіх рівнях розробки: від захисту аутентифікаційних даних за допомогою NextAuth.js, JWT-токенів та надійного хешування паролів алгоритмом bcrypt, до шифрування всього трафіку даних через HTTPS та впровадження диференційованого доступу до функціональних можливостей системи на основі ролей користувачів.

4 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Аналіз якості ПЗ

Для забезпечення високої якості розробленого вебзастосунку було проведено статичний аналіз коду з використанням інструменту SonarQube [34]. Даний інструмент надає можливість оцінювати код за сукупністю важливих характеристик, виявити потенційні проблеми, вразливості та області для покращення. Аналіз проводився для основного коду проєкту – як інтерфейсну частину, так і серверну логіку, реалізовану за допомогою Next.js і мови TypeScript.

Результати статичного аналізу коду проєкту представлені на рисунку 4.1.

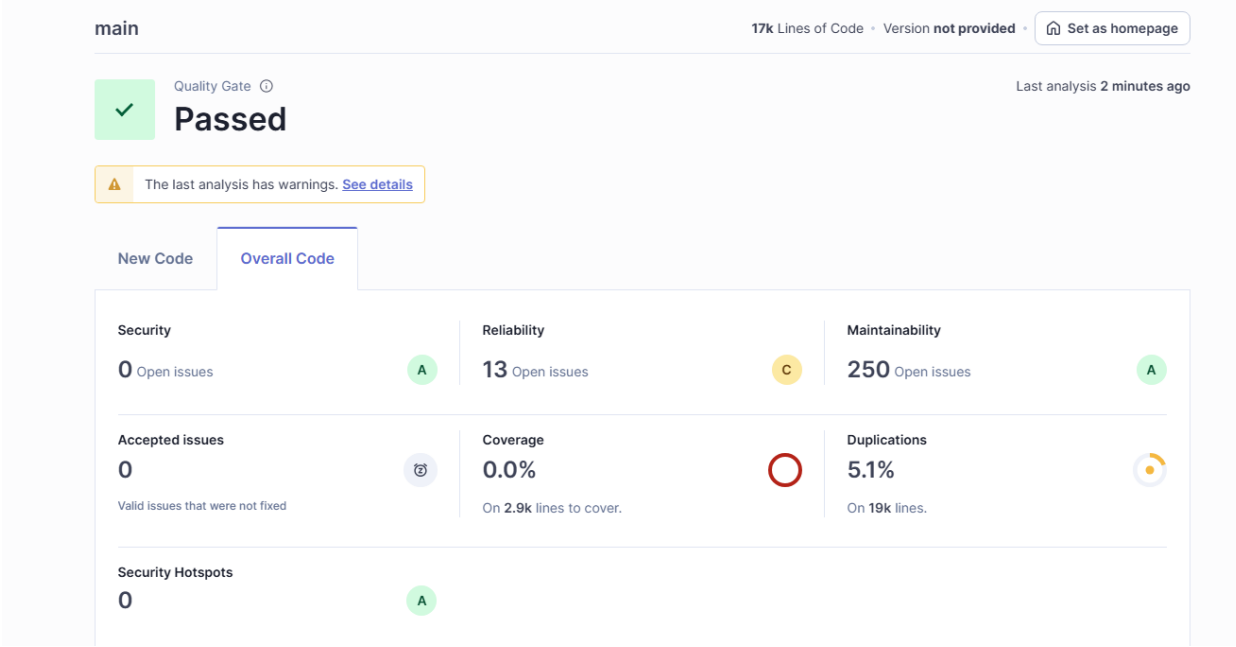


Рисунок 4.1 – Загальні результати статичного аналізу коду в SonarQube

Загальний статус проєкту в SonarQube – «Passed», що свідчить про належну якість програмного коду на поточному етапі реалізації. Детальний аналіз за основними метриками наведено в таблиці 4.1.

Таблиця 4.1 – Результати статичного аналізу коду проєкт

Параметр	Результат	Оцінка SonarQube
Безпека (Security)	0 відкритих проблем, 0 гарячих точок безпеки. Високий рівень безпеки підтверджує відсутність відомих вразливостей та дотримання практик безпечного кодування, таких як уникнення жорстко закодованих секретів.	A
Надійність (Reliability)	13 відкритих проблем. Загальна оцінка свідчить про високу надійність коду. Виявлені 13 незначних проблем надають можливість для подальшого точкового покращення та підвищення відмовостійкості системи.	A
Підтримуваність (Maintainability)	250 відкритих проблем. Оцінка «C» сигналізує про потребу в оптимізації структури коду та зменшенні технічної заборгованості. Виявлені проблеми є типовими для проєктів на активній стадії розробки та можуть бути адресовані під час рефакторингу для полегшення подальшої підтримки та розвитку.	C
Покриття тестами (Coverage)	0.0% (на 2.9 тис. рядків). На поточному етапі автоматизовані тести ще не були реалізовані. Це є плановим наступним кроком для забезпечення додаткової стабільності та полегшення регресійного тестування.	0.0%

Продовження таблиці 4.1

Дуплікація коду (Duplications)	5.1% (на 19 тис. рядків). Поточний рівень повторень знаходиться в допустимих межах.	5.1%
Цикломатична складність	1,960	1,960

4.2 Опис процесів тестування

Тестування програмного забезпечення є невід'ємним етапом розробки, спрямованим на забезпечення відповідності продукту визначеним вимогам та виявлення потенційних дефектів. Було виконане мануальне тестування програмного забезпечення. Основний акцент зроблено на перевірці працездатності інтерфейсу користувача, коректності обміну даними з бекендом, а також відповідності реакцій системи очікуваним сценаріям поведінки.

Опис відповідних тестів, що охоплюють ключові аспекти функціонування системи, наведено в таблицях 4.2 – 4.16.

Таблиця 4.2 – Тест-кейс 1: Реєстрація нового користувача (Власника тварини)

Початковий стан системи	Користувач знаходиться на сторінці реєстрації вебзастосунку «Vai»
Вхідні дані	Ім'я користувача (валідний рядок), електронна пошта (унікальна, валідний формат), пароль (відповідає вимогам до складності), підтвердження паролю (збігається), прапорець «Я ветеринар» не встановлено
Опис проведення тесту	У відповідні поля форми реєстрації вводяться коректні дані для власника тварини. Прапорець «Я ветеринар» залишається неактивним. Натискається кнопка «Зареєструватися»

Продовження таблиці 4.2

Очікуваний результат	Реєстрація проходить успішно. Обліковий запис типу «OWNER» створено, виводиться повідомлення про успішну реєстрацію, користувача перенаправлено на сторінку входу
Фактичний результат	Збігається з очікуваним.

Таблиця 4.3 – Тест-кейс 2: Реєстрація нового користувача (Ветеринара)

Початковий стан системи	Користувач знаходиться на сторінці реєстрації
Вхідні дані	Ім'я користувача (валідний рядок), електронна пошта (унікальна, валідний формат), пароль (відповідає вимогам складності), підтвердження паролю (співпадає з паролем), прапорець «Я ветеринар» встановлено. Додаткові поля: Спеціалізація (валідний рядок), Про себе (текст)
Опис проведення тесту	Вводяться дані для реєстрації. Встановлюється прапорець «Я ветеринар». Заповнюються обов'язкові та деякі опціональні поля профілю ветеринара. Натискається кнопка «Зареєструватися»
Очікуваний результат	Реєстрація проходить успішно. Створюється обліковий запис типу «VET» та пов'язаний профіль ветеринара. Користувач перенаправляється на сторінку входу
Фактичний результат	Збігається з очікуваним.

Таблиця 4.4 – Тест-кейс 3: Авторизація існуючого користувача

Початковий стан системи	Користувач знаходиться на сторінці входу. В системі існує обліковий запис з попередньо зареєстрованими даними
Вхідні дані	Електронна пошта (зареєстрована в системі), пароль (коректний для даного облікового запису)
Опис проведення тесту	У поля форми авторизації вводяться валідні електронна пошта та пароль. Натискається кнопка «Увійти»
Очікуваний результат	Авторизація проходить успішно. Користувач перенаправляється на відповідну панель керування (дашборд власника або ветеринара, залежно від типу акаунта)
Фактичний результат	Збігається з очікуваним.

Таблиця 4.5 – Тест-кейс 4: Додавання нової тварини Власником

Початковий стан системи	Авторизований користувач (Власник тварини) знаходиться на сторінці додавання нової тварини
Вхідні дані	Ім'я тварини (валідний рядок), вид (обрано зі списку), порода (опціонально), вік (опціонально, число), стать (обрано), вага (опціонально, число), медична історія (опціонально), вакцинація (опціонально). Фото не додається
Опис проведення тесту	Заповнюються відповідні поля форми. Натискається кнопка «Додати тваринку»
Очікуваний результат	Нова тварина успішно додається до профілю користувача. Користувач бачить оновлений список своїх тварин або отримує повідомлення про успішне додавання

Продовження таблиці 4.5

Фактичний результат	Збігається з очікуваним.

Таблиця 4.6 – Тест-кейс 5: Редагування даних тварини

Початковий стан системи	Авторизований користувач (Власник тварини). В його профілі існує принаймні одна тварина. Користувач знаходиться на сторінці редагування даних цієї тварини
Вхідні дані	Нове ім'я тварини, новий вік. Інші дані залишаються без змін
Опис проведення тесту	Змінюються поля «Ім'я» та «Вік» у формі редагування. Натискається кнопка «Зберегти зміни»
Очікуваний результат	Дані тварини успішно оновлені. На сторінці профілю тварини (або в списку тварин) відображаються оновлені ім'я та вік
Фактичний результат	Збігається з очікуваним.

Таблиця 4.7 – Тест-кейс 6: Пошук ветеринара за ім'ям

Початковий стан системи	Користувач авторизований. В системі існує ветеринар з відомим ім'ям. Користувач на сторінці пошуку ветеринарів
Вхідні дані	В поле пошуку за ім'ям введено ім'я існуючого ветеринара. Фільтри за спеціалізацією та видом тварин не обрані
Опис проведення тесту	Вводиться ім'я ветеринара. Натискається кнопка пошуку (або пошук відбувається динамічно)
Очікуваний результат	У результатах пошуку відображається профіль ветеринара з відповідним ім'ям

Продовження таблиці 4.7

Фактичний результат	Збігається з очікуваним.

Таблиця 4.8 – Тест-кейс 7: Перегляд профілю ветеринара та доступних слотів

Початковий стан системи	Авторизований користувач (Власник тварини). Обраний ветеринар має створений профіль та додані вільні слоти у своєму розкладі на майбутні дати
Вхідні дані	Користувач переходить на сторінку профілю обраного ветеринара
Опис проведення тесту	Відкривається сторінка профілю ветеринара. Користувач переглядає інформацію про ветеринара та календар з його розкладом
Очікуваний результат	На сторінці відображається детальна інформація про ветеринара. Календар показує дні з доступними слотами. При виборі дати з доступними слотами відображаються вільні часи
Фактичний результат	Збігається з очікуваним.

Таблиця 4.9 – Тест-кейс 8: Запис на онлайн-консультацію до ветеринара

Початковий стан системи	Авторизований користувач (Власник тварини) знаходиться на сторінці профілю ветеринара. У ветеринара є вільний слот на майбутню дату. Власник має принаймні одну додану тварину
Вхідні дані	Обрана дата та вільний часовий слот у розкладі ветеринара. Обрана тварина зі списку власника

Продовження таблиці 4.9

Опис проведення тесту	Користувач обирає дату, час та тварину. Натискає кнопку «Підтвердити запис»
Очікуваний результат	Запис на консультацію успішно створено. Обраний слот стає забронюваним. Власник тварини та ветеринар отримують сповіщення про заплановану консультацію. Консультація з'являється у списку запланованих консультацій обох користувачів
Фактичний результат	Збігається з очікуваним.

Таблиця 4.10 – Тест-кейс 9: Скасування запланованої консультації Власником

Початковий стан системи	Авторизований користувач (Власник тварини). Існує запланована консультація для його тварини з ветеринаром на майбутню дату, час якої ще не настав
Вхідні дані	Намір скасувати консультацію
Опис проведення тесту	Власник тварини переходить до списку своїх консультацій, знаходить заплановану консультацію та обирає опцію «Скасувати». Підтверджує дію у діалоговому вікні
Очікуваний результат	Консультація позначається як «CANCELLED_BY_CLIENT». Відповідний часовий слот у розкладі ветеринара стає вільним. Ветеринар отримує сповіщення про скасування
Фактичний результат	Збігається з очікуваним.

Таблиця 4.11 – Тест-кейс 10: Проведення онлайн-консультації (чат, надсилання тексту та зображення)

Початковий стан системи	Авторизовані Власник тварини та Ветеринар. Настав час їхньої запланованої консультації. Обидва відкрили сторінку чату консультації
Вхідні дані	Текстове повідомлення від власника. Файл зображення від власника (валідний формат та розмір). Текстове повідомлення від ветеринара
Опис проведення тесту	Власник тварини вводить текстове повідомлення та надсилає його. Потім прикріплює та надсилає файл зображення. Ветеринар бачить повідомлення та зображення, відповідає текстовим повідомленням
Очікуваний результат	Усі текстові повідомлення та зображення коректно відображаються в інтерфейсі чату обох учасників в реальному часі. Повідомлення зберігаються в історії. Кнопка надсилання активна тільки якщо є текст або вибрано файл
Фактичний результат	Збігається з очікуваним.

Таблиця 4.12 – Тест-кейс 11: Аналіз симптомів за допомогою ШІ для обраної тварини

Початковий стан системи	Авторизований користувач (Власник тварини) знаходиться на сторінці аналізу симптомів. Обрано тварину відповідного типу (наприклад, Собака) для аналізу. Користувач погодився з попередженням про попередній характер діагностики
Вхідні дані	Набір обраних симптомів зі списку, релевантних для обраного типу тварини

Продовження таблиці 4.12

Опис проведення тесту	Користувач обирає кілька симптомів зі списку та натискає кнопку «Аналізувати»
Очікуваний результат	Після короткого очікування система відображає результат аналізу: список імовірних діагнозів з відсотками ймовірності, загальний розрахований рівень серйозності та відповідні рекомендації, включаючи пораду звернутися до ветеринара
Фактичний результат	Збігається з очікуваним.

Таблиця 4.13 – Тест-кейс 12: Збереження результатів аналізу симптомів

Початковий стан системи	Авторизований користувач (Власник тварини). Щойно отримано результати аналізу симптомів для його тварини
Вхідні дані	Результати аналізу з попереднього кроку
Опис проведення тесту	Користувач натискає кнопку «Зберегти результати аналізу»
Очікуваний результат	Результати аналізу (введені симптоми, отримані прогнози, загальна серйозність) зберігаються в базі даних, пов'язані з відповідною твариною. Користувач отримує повідомлення про успішне збереження. Кнопка «Зберегти результати» стає неактивною або змінює вигляд на «Збережено»
Фактичний результат	Збігається з очікуваним.

Таблиця 4.14 – Тест-кейс 13: Перегляд історії аналізів симптомів для тварини

Початковий стан системи	Авторизований користувач (Власник тварини). Для однієї з його тварин було збережено принаймні один результат аналізу симптомів. Користувач відкриває діалогове вікно «Історія аналізів» для цієї тварини
Вхідні дані	Відсутні (дія користувача)
Опис проведення тесту	Користувач відкриває історію аналізів для конкретної тварини
Очікуваний результат	Відображається список попередніх аналізів для обраної тварини, відсортований за датою. Кожен запис містить дату аналізу, основний прогноз та загальний рівень серйозності. Є можливість перейти до деталей аналізу
Фактичний результат	Збігається з очікуваним.

Таблиця 4.15 – Тест-кейс 14: Керування розкладом ветеринара (додавання слотів)

Початковий стан системи	Авторизований користувач (Ветеринар) знаходиться на сторінці керування своїм розкладом
Вхідні дані	Обрано майбутню дату. Обрано декілька вільних часових слотів на цю дату
Опис проведення тесту	Ветеринар обирає дату на календарі. Потім обирає декілька часових слотів зі списку доступних. Натискає кнопку «Зберегти графік»

Продовження таблиці 4.15

Очікуваний результат	Обрані слоти успішно додаються до розкладу ветеринара на вказану дату та позначаються як вільні. На календарі ця дата підсвічується як така, що має слоти. Користувач отримує повідомлення про успішне збереження
Фактичний результат	Збігається з очікуваним.

Таблиця 4.16 – Тест-кейс 15: Отримання сповіщення про нагадування за 10 хвилин до консультації

Початковий стан системи	Авторизований користувач (Власник тварини або Ветеринар). Має заплановану консультацію через 10 хвилин. Cron-завдання для надсилання сповіщень налаштоване та працює
Вхідні дані	Час наближається до запланованої консультації
Опис проведення тесту	Система автоматично перевіряє заплановані консультації. За 10 хвилин до початку консультації генерується та надсилається сповіщення учаснику(ам). Користувач перевіряє свої сповіщення в інтерфейсі застосунку
Очікуваний результат	Учасники консультації отримують сповіщення про те, що консультація розпочнеться приблизно за 10 хвилин. Сповіщення містить посилання на сторінку консультації. Поле notificationSentAt для консультації оновлюється
Фактичний результат	Збігається з очікуваним.

4.3 Опис контрольного прикладу

Для демонстрації роботи ключового функціоналу вебзастосунку «Vai» розглянемо послідовний сценарій взаємодії власника тварини з системою. Користувач, будучи авторизованим, переходить до розділу «Аналіз симптомів» з метою отримання попередньої оцінки стану здоров'я свого улюбленця.

На сторінці (рисунок 4.2) він обирає конкретну тварину зі списку раніше доданих.

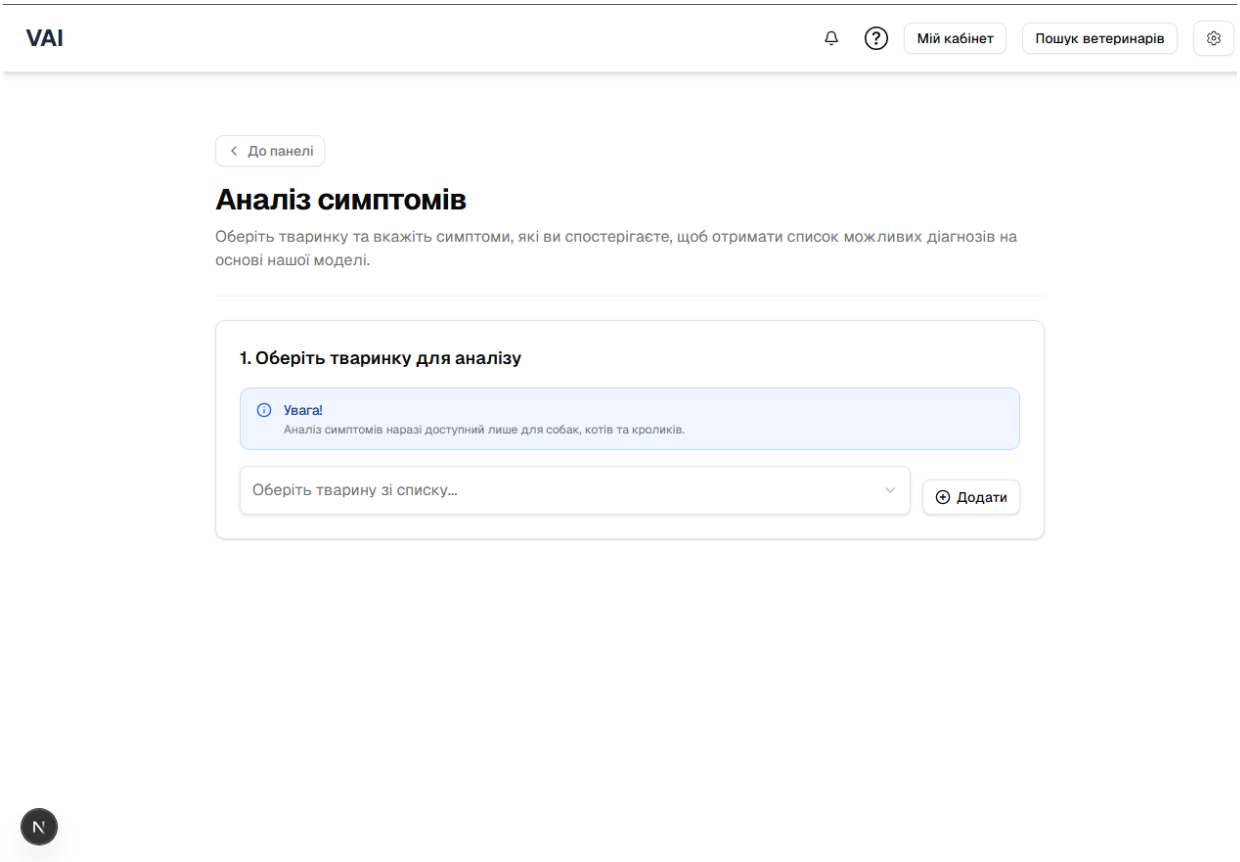


Рисунок 4.2 – Сторінка вибору тварини для аналізу симптомів

Перед початком аналізу система виводить попередження про імовірнісний характер результатів, яке користувач підтверджує (рисунок 4.3).

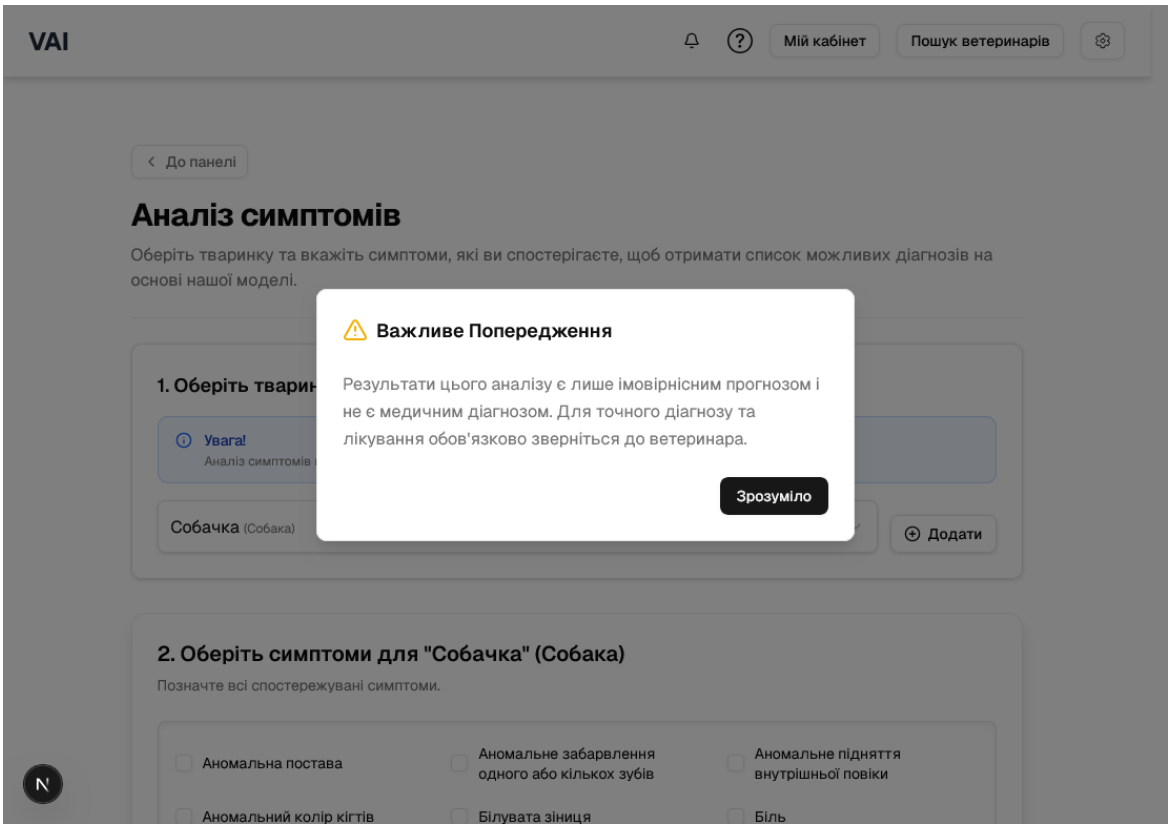


Рисунок 4.3 – Вікно попередження перед аналізом симптомів

Далі відкривається інтерфейс (рисунок 4.4), де власник відмічає спостережувані симптоми.

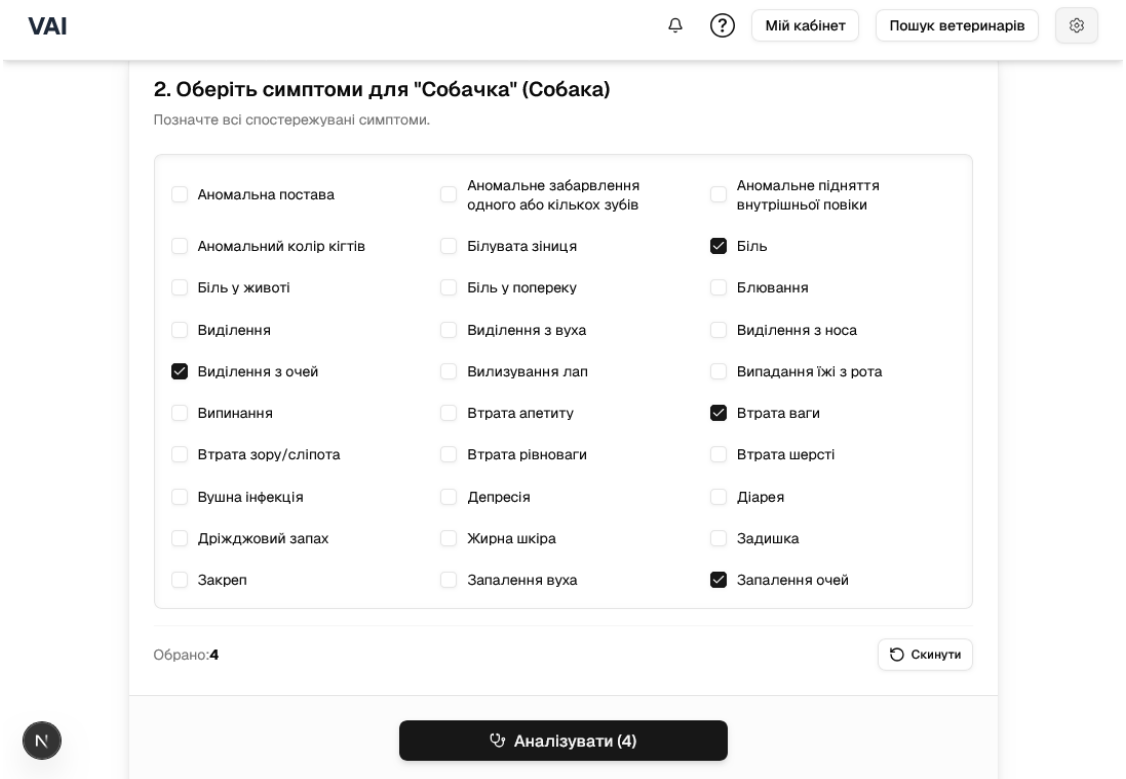


Рисунок 4.4 – Інтерфейс вибору симптомів тварини

Після вибору симптомів користувач натискає кнопку «Аналізувати», ініціюючи запит до ШІ-моделі. Під час обробки відображається відповідний індикатор (рисунок 4.5).

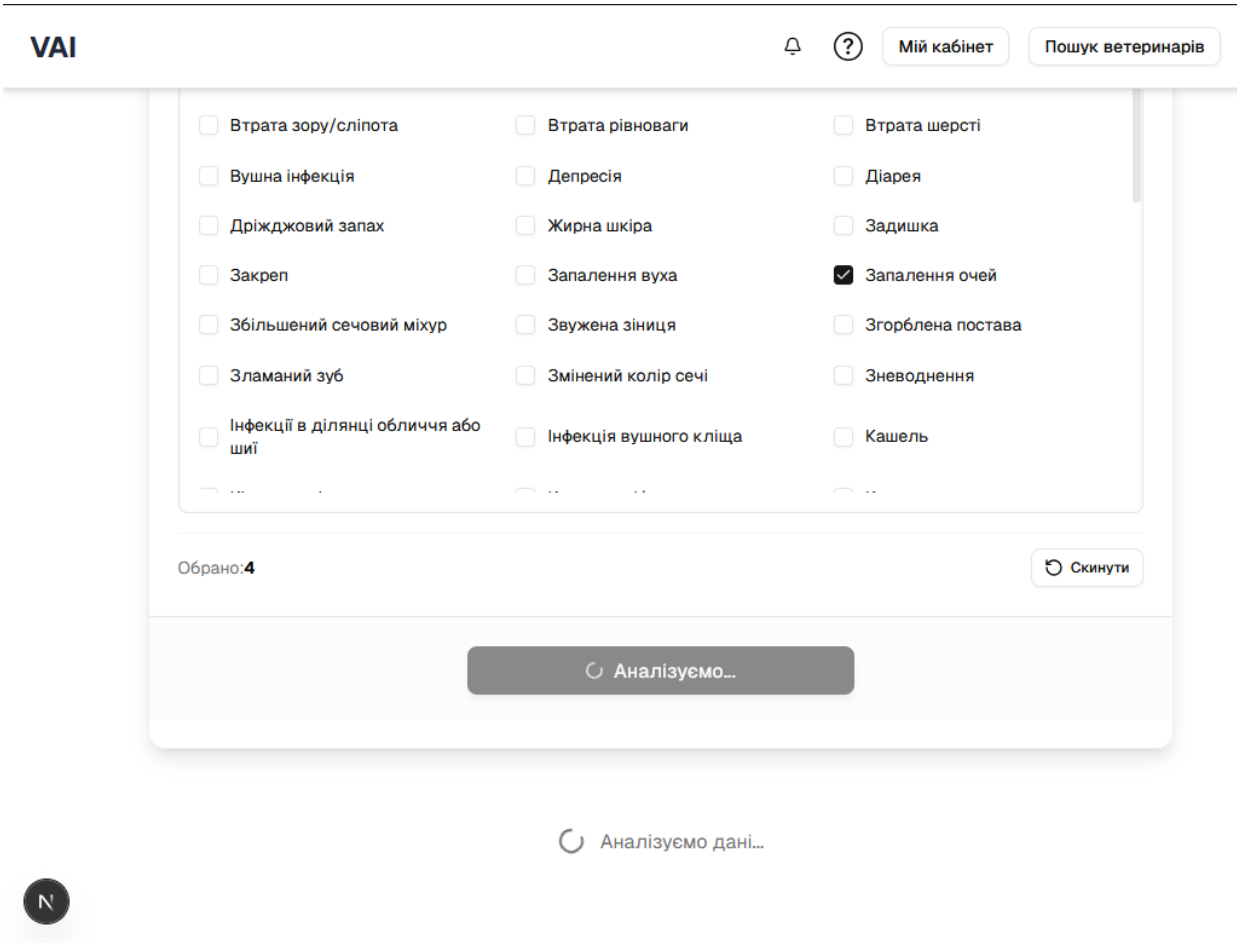


Рисунок 4.5 – Процес аналізу симптомів

По завершенні на сторінці з'являються результати: перелік імовірних діагнозів з відсотками, загальний рівень серйозності та рекомендації. Користувачеві також пропонуються опції для подальших дій (рисунок 4.6).

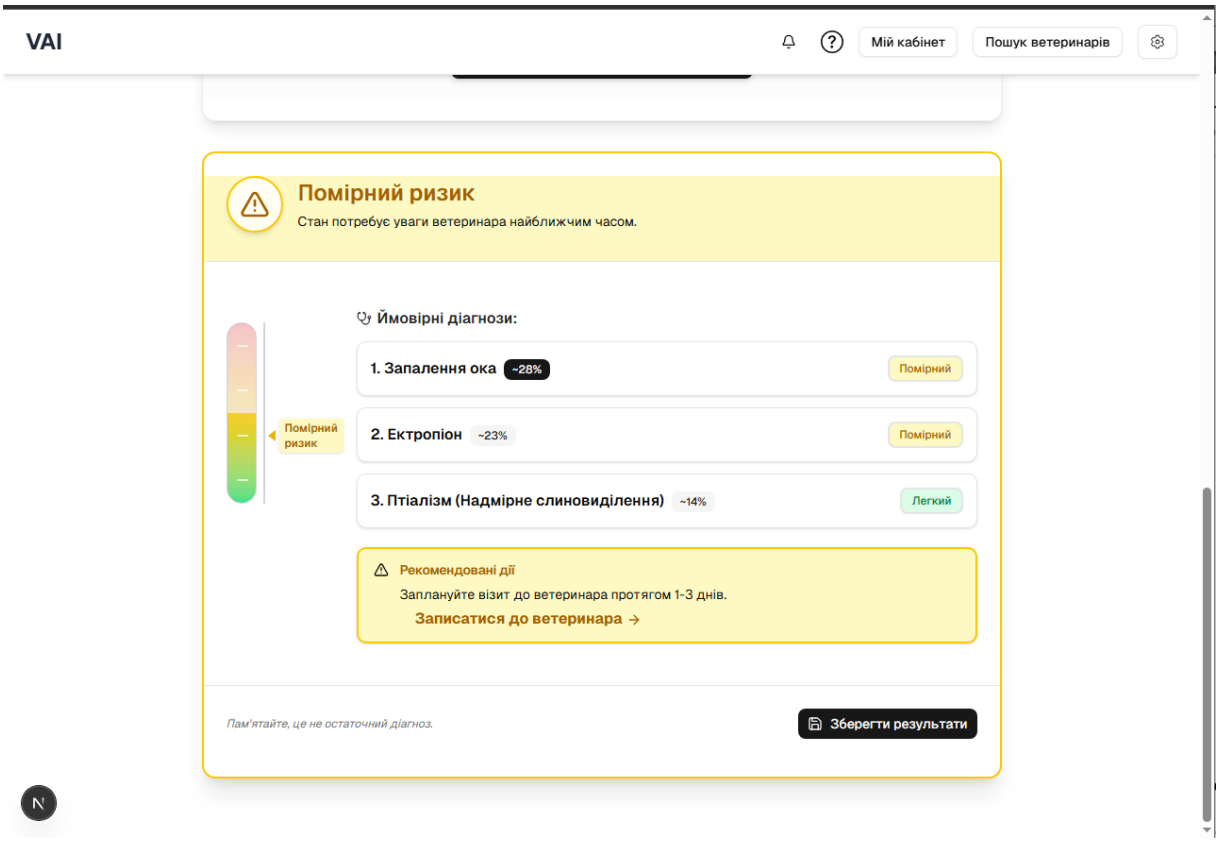


Рисунок 4.6 – Відображення результатів аналізу симптомів

Припустимо, на основі результатів аналізу або за власним бажанням, користувач вирішує знайти ветеринара. Він переходить до розділу пошуку, де може застосувати фільтри, наприклад, за спеціалізацією, що може бути пов'язано з отриманими діагностичними припущеннями (рисунок 4.7).

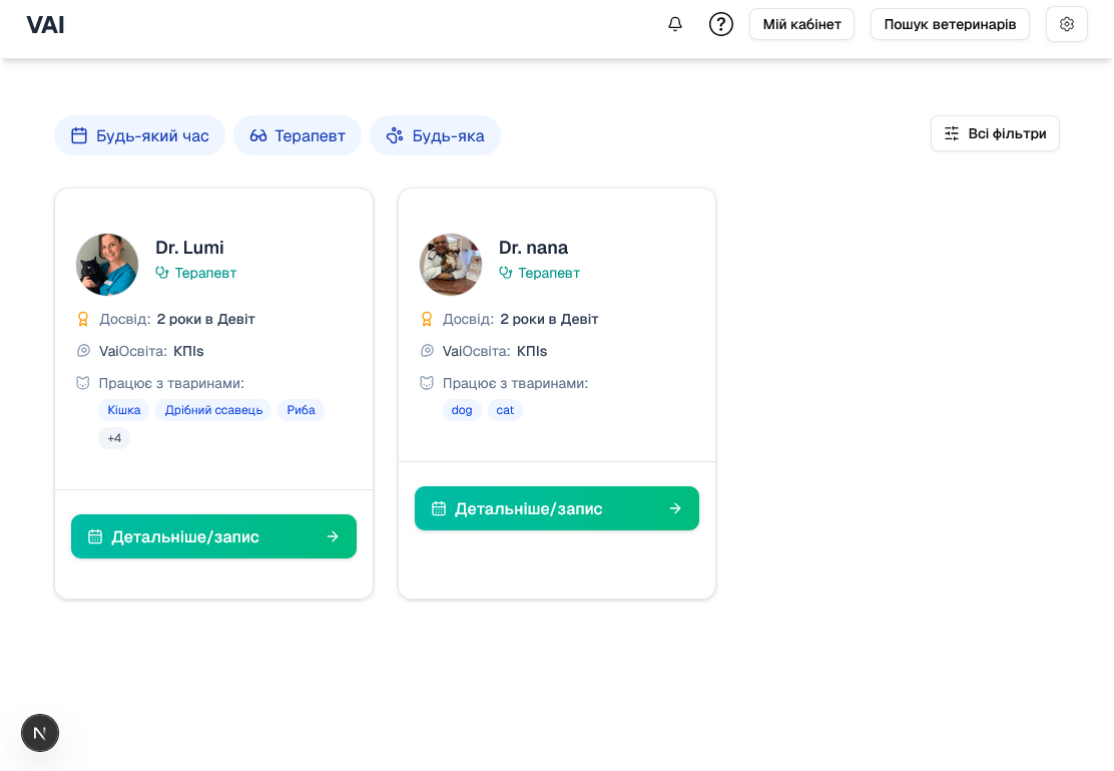


Рисунок 4.7 – Сторінка пошуку ветеринара з використанням фільтрів

Зі списку знайдених фахівців користувач обирає одного та переглядає його детальний профіль, включаючи інформацію про досвід, освіту та доступний розклад для запису (рисунок 4.8).

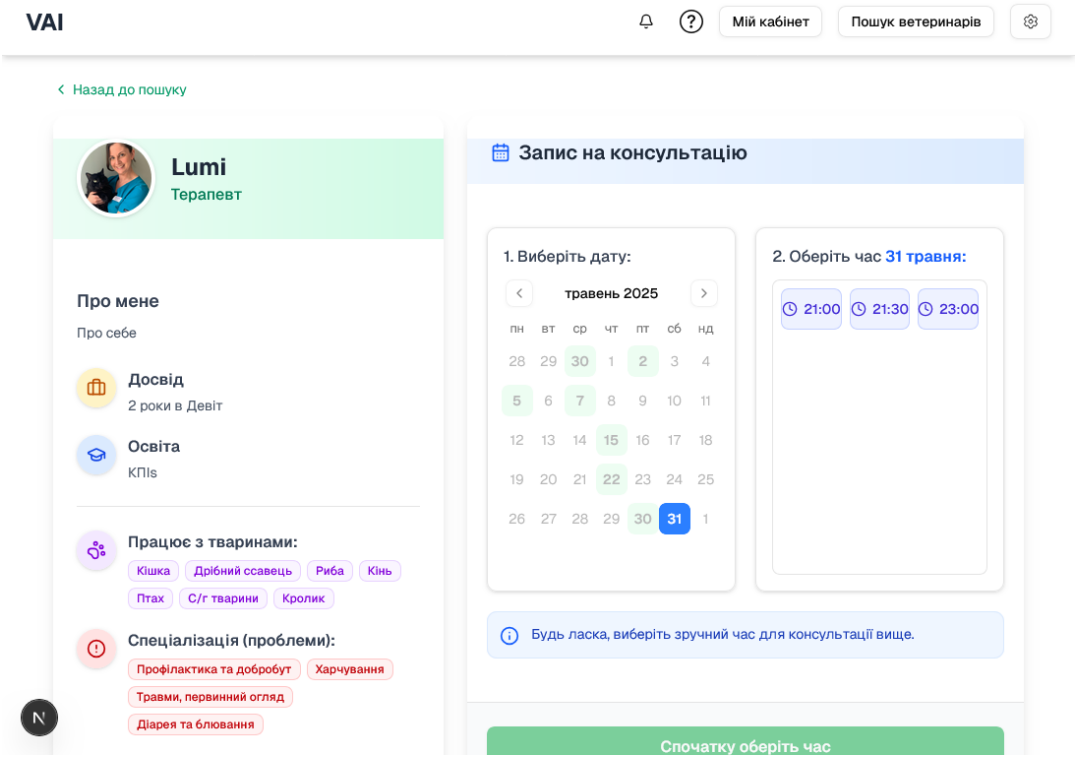


Рисунок 4.8 – Сторінка профілю ветеринара з календарем доступних слотів

На сторінці профілю ветеринара користувач обирає зручну дату та вільний часовий слот на календарі (рисунок 4.9).

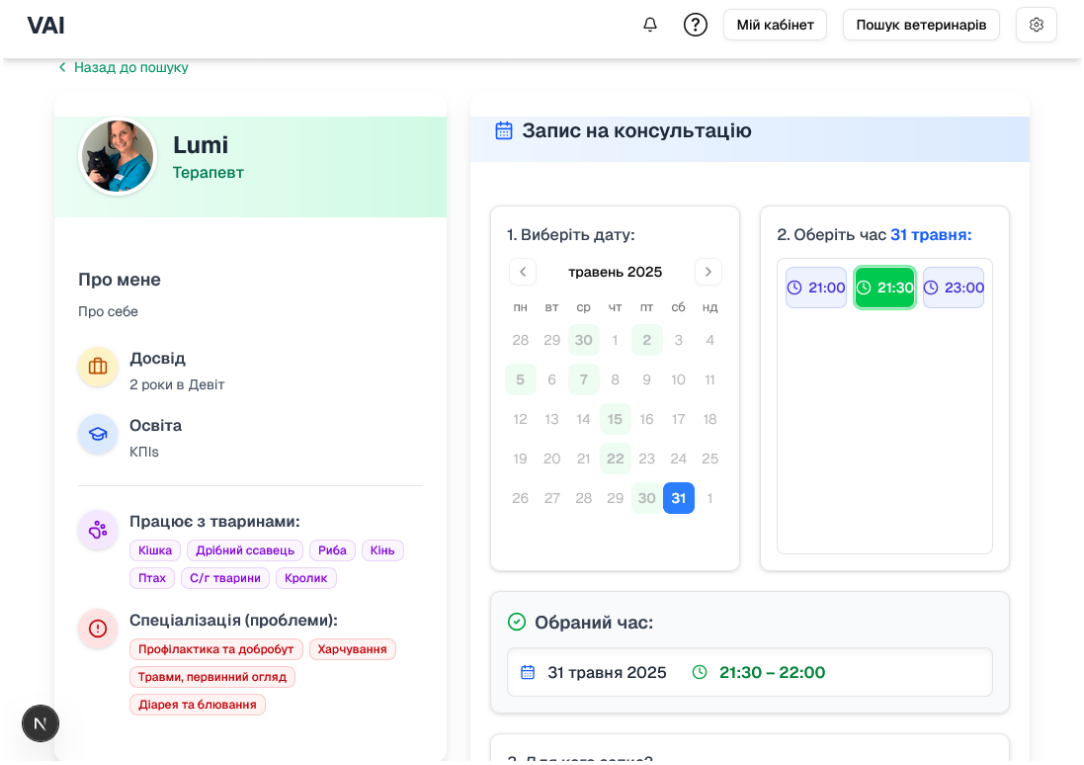


Рисунок 4.9 – Вибір дати та часу для запису на консультацію

Наступним кроком є вибір тварини, для якої потрібна консультація, зі списку своїх улюбленців (рисунок 4.10).

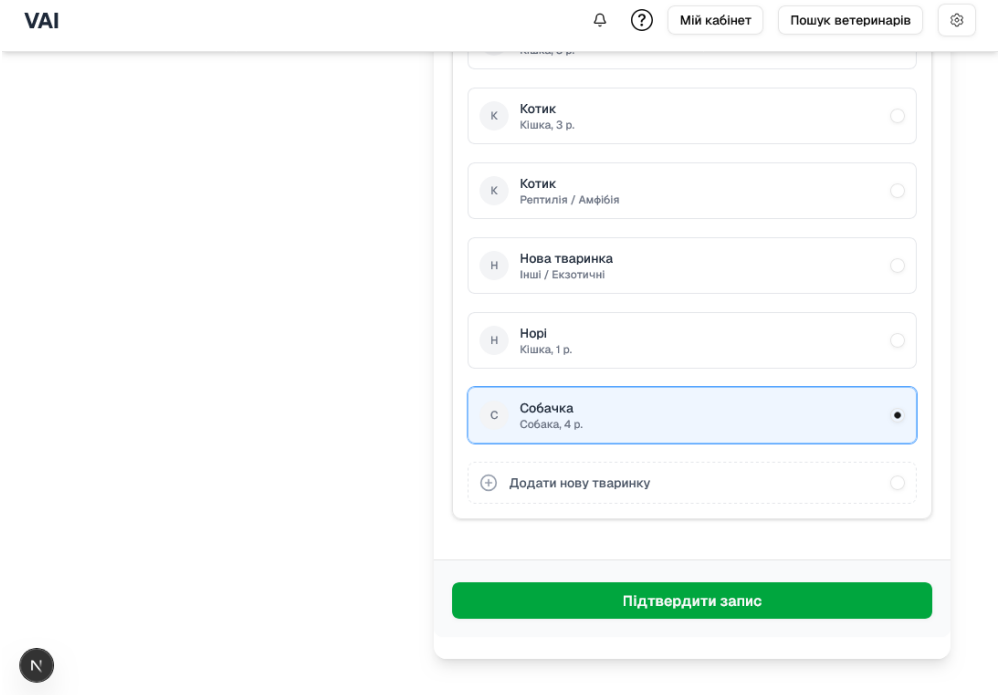


Рисунок 4.10 – Вибір тварини для запису на консультацію

Після підтвердження всіх даних натискається кнопка «Підтвердити запис». У разі успішного бронювання система відображає відповідне повідомлення, а запланована консультація додається до розкладів обох учасників (рисунок 4.11).

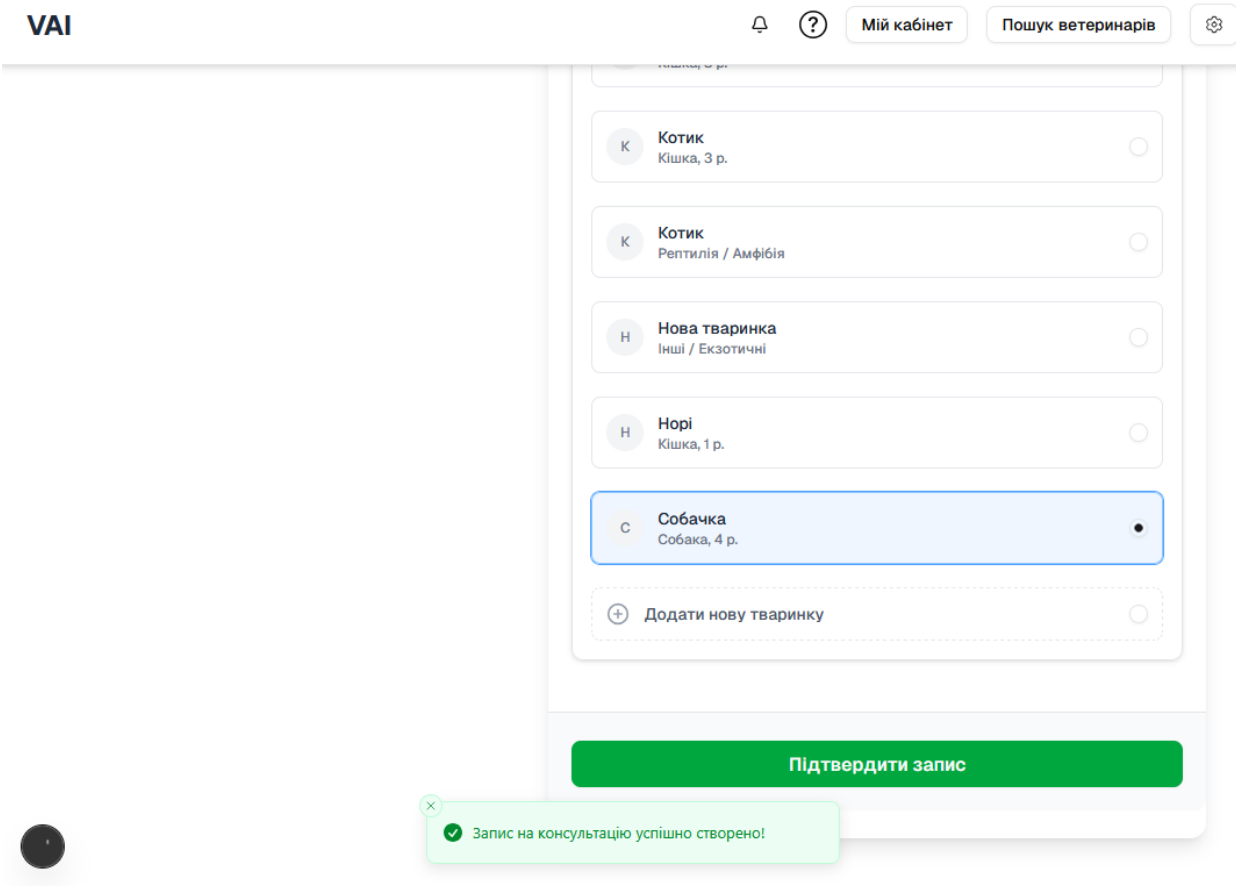


Рисунок 4.11 – Повідомлення про успішне бронювання консультації

У призначений час власник тварини та ветеринар входять до системи та відкривають сторінку консультації, де активується чат. Вони можуть обмінюватися текстовими повідомленнями та файлами зображень для детального обговорення стану тварини. Всі повідомлення відображаються в реальному часі та зберігаються в історії (рисунок 4.12).

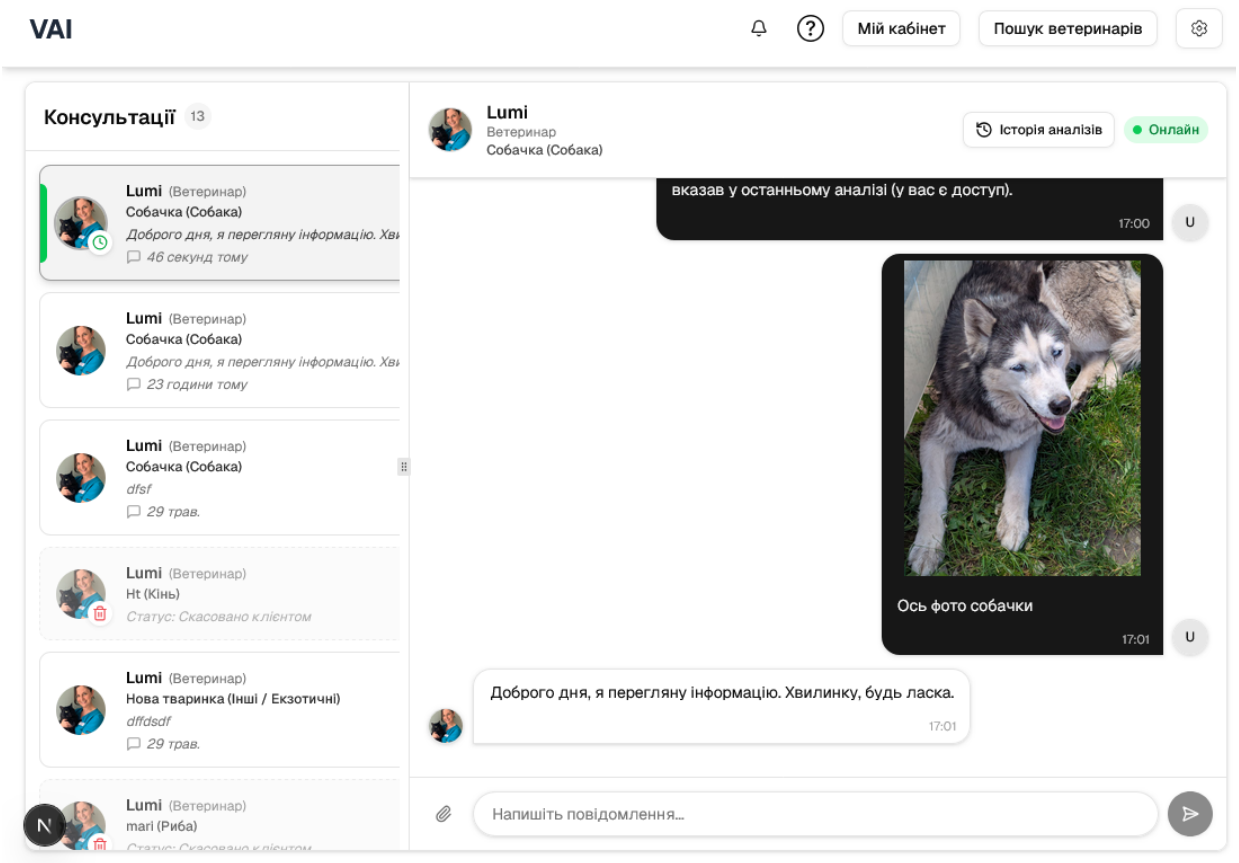


Рисунок 4.12 – Інтерфейс чату онлайн-консультації

Висновки до розділу

У цьому розділі дипломного проєкту було проведено комплексний аналіз якості розробленого програмного забезпечення «Vai» та детально описано процеси його тестування. Основною завданням цього етапу було підтвердження відповідності вебзастосунку як функціональним, так і нефункціональним вимогам, а також виявлення можливих помилок і їхнє усунення.

Для оцінки якості кодової бази було використано інструмент статичного аналізу SonarQube. Отримані результати показали, що система загалом відповідає встановленим стандартам якості, зокрема в аспектах безпечності та надійної роботи. Водночас було виявлено аспекти, пов'язані з підтримуваністю коду, які потребують подальшої оптимізації та рефакторингу. Важливим напрямком подальшої роботи визначено впровадження автоматизованого тестування для підвищення стабільності системи.

У межах тестування було сформовано набір тестових сценаріїв, призначених для перевірки ключового функціоналу вебзастосунку. Ці тест-кейси охоплюють сценарії реєстрації та авторизації користувачів різних типів, керування даними тварин, пошук ветеринарів, запис на онлайн-консультацію, проведення самої консультації в чаті з обміном текстовими повідомленнями та зображеннями, а також використання модуля аналізу симптомів за допомогою штучного інтелекту та збереження отриманих результатів. Кожен сценарій супроводжується описом початкових умов, вхідних параметрів, послідовності дій, а також очікуваних і фактичних результатів.

Окрім того, у розділі наведено опис контрольного прикладу, який ілюструє наскрізний сценарій взаємодії користувача (власника тварини) з системою, починаючи від аналізу симптомів улюбленця за допомогою ШІ, пошуку відповідного ветеринарного спеціаліста, запису на онлайн-прийом і завершуючи проведенням самої консультації. Цей приклад, доповнений відповідними ілюстраціями (скріншотами інтерфейсу), що наочно демонструють практичне застосування основного функціоналу вебзастосунку та його здатність вирішувати поставлені завдання.

Проведені заходи з аналізу якості та тестування підтвердили працездатність ключових модулів системи й відповідність розробленого програмного забезпечення ключовим вимогам. Отримані результати стали основою для подальших рекомендацій щодо вдосконалення та еволюціонування програмного продукту.

5 РОЗГОРТАННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

5.1 Розгортання програмного забезпечення

Для забезпечення доступності та надійної роботи вебзастосунку «Vai» було обрано сучасні хмарні платформи, кожна з яких оптимізована для певного типу компонентів системи. Основний застосунок, розроблений на Next.js, розгорнуто на платформі Vercel [35], відомій своєю тісною інтеграцією з цим фреймворком. Сервіс штучного інтелекту, реалізований на Python з використанням Flask, розміщено на платформі Render [36], яка надає гнучкі можливості для розгортання вебсервісів. База даних PostgreSQL та сховище для файлів (фотографій тварин та профілів ветеринарів) функціонують на базі інфраструктури Supabase [37].

Процес розгортання розпочався з налаштування інфраструктури на Supabase. Було створено новий проєкт, в рамках якого активовано базу даних PostgreSQL. Схема бази даних, попередньо розроблена та описана у файлі schema.prisma, була застосована до хмарної бази даних за допомогою механізму міграцій Prisma. Отримані облікові дані для підключення до бази (хост, порт, ім'я бази, користувач, пароль) були конфігуровані як змінні середовища для основного застосунку. Паралельно, в Supabase Storage було створено окремий бакет під назвою pet-photos та налаштовано відповідні політики безпеки доступу (RLS) для забезпечення контрольованого завантаження та отримання файлів користувачами.

Наступним кроком було розгортання основного Next.js застосунку на платформі Vercel. Репозиторій проєкту, розміщений на GitHub, було підключено до Vercel, що дозволило налаштувати автоматичне розгортання при кожному оновленні коду в основній гілці. У налаштуваннях проєкту на Vercel було вказано всі необхідні змінні середовища, включаючи DATABASE_URL для з'єднання з базою даних Supabase, ключі для NextAuth.js (AUTH_SECRET), секретний ключ для Supabase JWT

(SUPABASE_JWT_SECRET), а також публічні ключі для доступу до Supabase з клієнтської частини (NEXT_PUBLIC_SUPABASE_URL, NEXT_PUBLIC_SUPABASE_ANON_KEY) та секрет для захисту cron-завдань (CRON_SECRET).

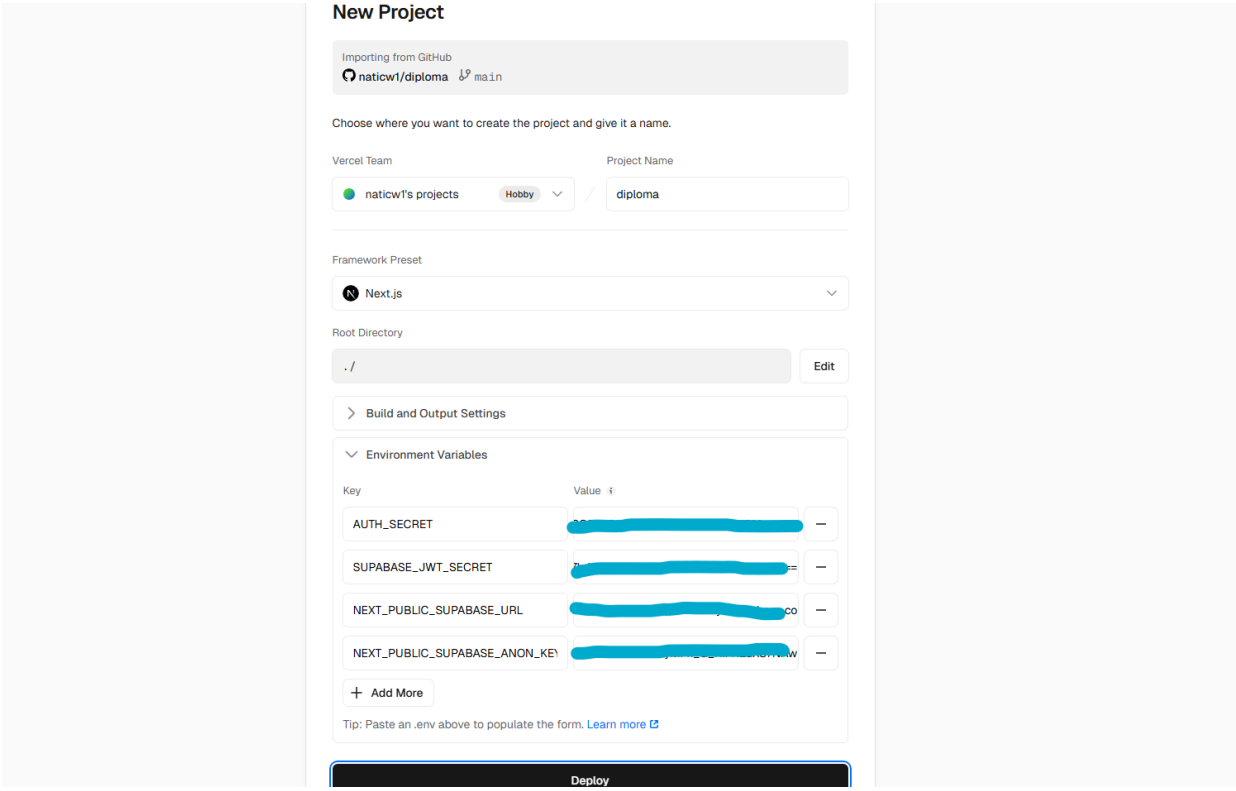


Рисунок 5.1 – Налаштування змінних середовища для Next.js проєкту на Vercel

Після успішного розгортання застосунок став доступним за публічною URL-адресою, а також було налаштовано Vercel Cron для періодичного виконання завдання надсилання сповіщень.

Для розгортання сервісу штучного інтелекту на Flask було використано платформу Render. Код ML-сервісу, розміщений в окремому репозиторії GitHub, було підключено до нового вебсервісу на Render. У налаштуваннях сервісу було вказано середовище виконання Python, команду для встановлення залежностей з файлу requirements.txt та команду для запуску Flask-застосунку. Після завершення процесу розгортання, ML-сервіс отримав унікальну URL-адресу, яка була прописана як змінна середовища NEXT_PUBLIC_DIAGNOSIS_API_URL в основному Next.js застосунку,

забезпечуючи таким чином взаємодію між фронтендом та модулем аналізу симптомів.

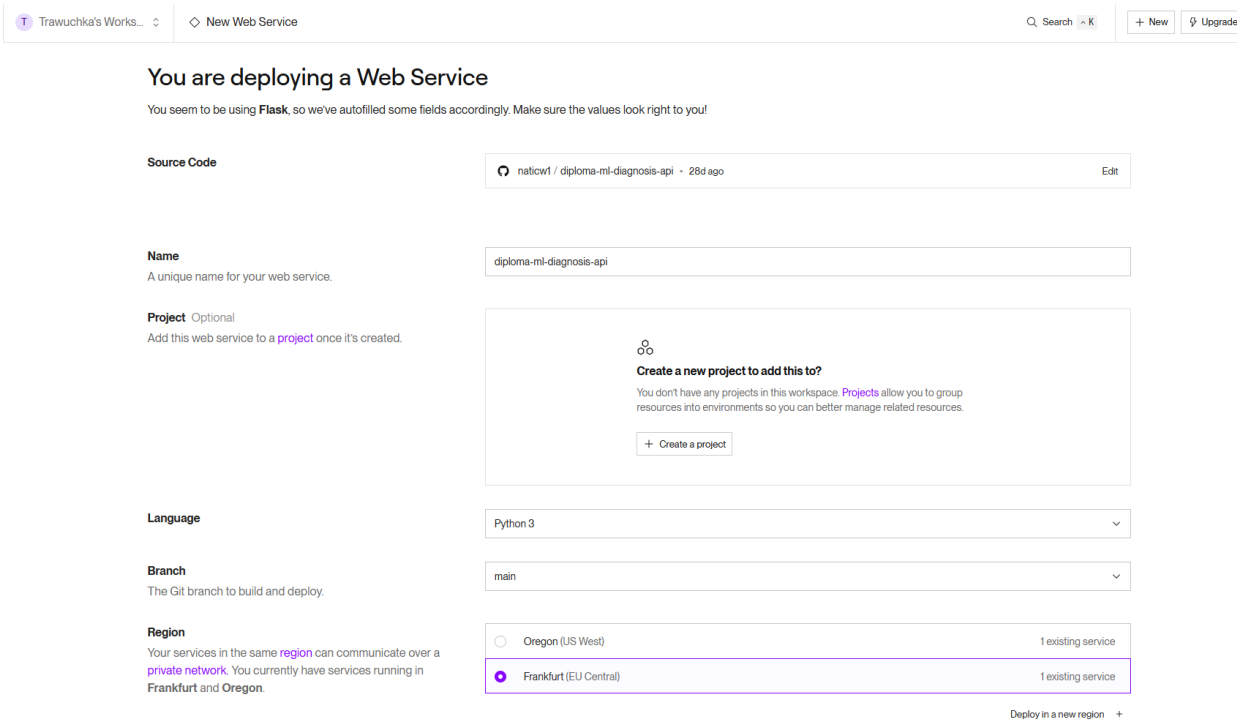


Рисунок 5.2 – Конфігурація розгортання Python/Flask сервісу на Render

5.2 Супровід програмного забезпечення

Інструкція користувача наведена в окремому документі. Забезпечення довготривалої та надійної роботи розгорнутої системи «Vai» реалізується через супровід, що включає оновлення, моніторинг, виправлення помилок та підтримку користувачів.

Оновлення програмного забезпечення спрямовані на впровадження нового функціоналу, покращення продуктивності та безпеки. Процес починається з аналізу вимог та відгуків, після чого розробляються або модифікуються програмні модулі. Зміни тестуються на локальних середовищах та в окремих Git-гілках (develop, feature-гілки) перед інтеграцією. Розгортання оновлень Next.js застосунку на Vercel та Flask ML-сервісу на Render відбувається автоматично після злиття коду в основну гілку (main) відповідних репозиторіїв. Оновлення схеми бази даних Supabase PostgreSQL здійснюються за допомогою міграцій Prisma: розробник модифікує schema.prisma, генерує міграцію (npx prisma migrate dev --name

<назва_міграції>) для тестування та застосовує її до робочої БД (npx prisma migrate deploy) після перевірки.

Моніторинг працездатності «Vai» здійснюється за допомогою інструментів хостинг-платформ Vercel, Render та Supabase. Vercel надає дані щодо часу відповіді, помилок (HTTP 5xx, 4xx), використання ресурсів (CPU, пам'ять) Next.js застосунку, а також відстежує трафік і статус розгортань. Render забезпечує моніторинг доступності ML-сервісу, навантаження на його ресурси (CPU, пам'ять) та аналіз логів. Supabase дозволяє контролювати використання дискового простору PostgreSQL, кількість з'єднань, навантаження на CPU бази даних та виявляти повільні SQL-запити. Додатково, для детального аналізу нештатних ситуацій, передбачено інтеграцію системи логування помилок безпосередньо на рівні Next.js та Flask компонентів застосунку «Vai».

Виявлення помилок відбувається через дані моніторингу, системні логи або звернення користувачів. Процес виправлення включає збір інформації про інцидент (повідомлення, кроки відтворення), діагностику причини, розробку та тестування виправлення. Критичні помилки, що впливають на основний функціонал або безпеку, усуваються невідкладно шляхом позачергового оновлення. Менш пріоритетні виправлення включаються до планових релізів.

Висновки до розділу

У цьому розділі продемонстровано процеси розгортання та подальшого супроводу програмного забезпечення «Vai». Описано конфігурацію інфраструктури на хмарних платформах: Vercel (Next.js застосунок), Render (Flask ML-сервіс), а також Supabase (PostgreSQL база даних, файлове сховище, включаючи налаштування бакету pet-photos та застосування схеми БД через міграції Prisma). Процес розгортання усіх основних модулів автоматизовано за допомогою інтеграції з GitHub, що забезпечує оперативне внесення та впровадження змін до коду. Особливу увагу приділено налаштуванню середовищ виконання, зокрема конфігурації змінних середовища для коректної

взаємодії між компонентами, а також використанню Vercel Cron для автоматичного запуску регулярних задач (системи сповіщень).

З метою забезпечення стабільної та безперервної роботи, визначено заходи із супроводу, що охоплюють регулярні оновлення для впровадження нового функціоналу та посилення безпеки. Ці оновлення тестуються в окремих Git-гілках і розгортаються автоматично для веб-компонентів на Vercel/Render, та контрольовано для схеми бази даних Supabase (команди npx prisma migrate dev та npx prisma migrate deploy). Система моніторингу забезпечує спостереження за технічним станом програмного забезпечення, використовуючи вбудовані засоби Vercel, Render та Supabase. Вони дозволяють відстежувати продуктивність, активність запитів та стабільність бази даних. Додатково застосовується логування на рівні коду для діагностики несправностей. Процес виправлення помилок включає їх ідентифікацію, діагностику та оперативне усунення, з пріоритетом для критичних несправностей. Запропоновані методи розгортання та технічного обслуговування формують надійну платформу для стабільного функціонування вебзастосунку, регулярного вдосконалення його можливостей і масштабування в майбутньому.

ВИСНОВКИ

В ході виконання дипломного проєкту розроблено вебзастосунок «Vai», призначений для раннього виявлення захворювань домашніх тварин та надання онлайн-консультацій з ветеринарними фахівцями. Система включає модуль попередньої діагностики на основі штучного інтелекту, функціонал управління профілями користувачів (власників тварин та ветеринарів), керування даними про тварин, пошук ветеринарів, планування та проведення текстових онлайн-консультацій.

Основна мета проєкту – підвищення доступності та якості ветеринарної допомоги шляхом використання сучасних цифрових технологій – була досягнута. Розроблений вебзастосунок «Vai» надає власникам тварин інструмент для первинної оцінки стану здоров’я улюбленців та зручний спосіб зв’язку з ветеринарами, що сприяє своєчасному реагуванню на потенційні проблеми.

У рамках проєкту було вирішено ключові завдання: реалізовано безпечну реєстрацію та авторизацію користувачів з розрізненням ролей; надано можливість користувачам керувати своїми особистими та професійними профілями; власники тварин отримали інструменти для додавання, редагування та перегляду інформації про своїх улюбленців; розроблено функціонал пошуку ветеринарів за критеріями з переглядом їхніх детальних профілів; ветеринари можуть формувати та керувати своїм робочим розкладом для онлайн-консультацій; власники тварин можуть бронювати доступні часові вікна для консультацій з обраними фахівцями; створено систему текстового чату в реальному часі для проведення онлайн-консультацій з можливістю обміну зображеннями; інтегровано модуль штучного інтелекту для аналізу симптомів тварин та надання попередніх діагностичних припущень; впроваджено систему сповіщень для інформування користувачів про важливі події.

Предметна область телеветеринарії та застосування ШІ в діагностиці має значний потенціал для подальшого розвитку. Розроблений вебзастосунок «Vai» може бути розширений шляхом додавання відеоконсультацій, інтеграції з системами управління ветеринарними клініками, розробки мобільної версії, а також вдосконаленням діагностичного ШІ-модуля. Це підтверджує доцільність продовження робіт у даному напрямку для створення ще більш комплексних та ефективних рішень у сфері цифрової ветеринарної медицини. Вебзастосунок розроблено за клієнт-серверною архітектурою з використанням TypeScript та фреймворку Next.js для основного застосунку, Python та Flask для ШІ-сервісу, і PostgreSQL як системи управління базами даних. Розгортання здійснено на хмарних платформах Vercel, Render та Supabase. Якість програмного забезпечення перевірялася шляхом статичного аналізу коду та мануального функціонального тестування.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1) NAUKA.UA. Алгоритм розпізнав ранню стадію хвороби Альцгеймера з точністю 99% [Електронний ресурс]. – 2024. – Режим доступу: https://nauka.ua/news/algoritm-rozpiznav-rannyu-stadiyu-hvorobi-alcgejmera-z-tochnistyu-99 (дата звернення: 28.03.2025).

2) UnivDatos Market Insights. Veterinary Telehealth Market [Електронний ресурс]. – Режим доступу: https://univdatos.com/news/veterinary-telehealth-market (дата звернення: 28.03.2025).

3) Інтерфакс-Україна. Обсяг ринку телемедицини для тварин досягне $546,8 млн до 2030 р. – дослідження [Електронний ресурс]. – 2022. – Режим доступу: https://interfax.com.ua/news/economic/866105.html (дата звернення: 28.03.2025).

4) Precedence Research. Veterinary Telehealth Market Size, Share, Trends [Електронний ресурс]. – Режим доступу: https://www.precedenceresearch.com/veterinary-telehealth-market (дата звернення: 28.03.2025).

5) Suziria Group. Зростання зооіндустрії в Україні: ринок збільшився на 27% у 2023 році [Електронний ресурс]. – 2024. – Режим доступу: https://suziria-group.com/zrostannia-zooindustrii-v-ukraini-rynok-zbilshyvsia-na-27-u-2023-rotsi (дата звернення: 28.03.2025).

6) Vetster. What is Vetster? [Електронний ресурс]. – Режим доступу: https://support.vetster.com/en/what-is-vetster (дата звернення: 08.04.2025).

7) IDEXX. VetConnect PLUS [Електронний ресурс]. – Режим доступу: https://www.idexx.com/en/veterinary/software-services/vetconnect-plus (дата звернення: 08.04.2025).

8) Sure Petcare. Animo [Електронний ресурс]. – Режим доступу: https://www.surepetcare.com/en-us/animo (дата звернення: 08.04.2025).

9) Australian Dog Lover. Animo: Dog Activity & Behaviour Monitor [Електронний ресурс]. – 2019. – Режим доступу:

https://www.australiandoglover.com/2019/01/animo-dog-activity-behaviour-monitor.html (дата звернення: 08.04.2025).

10) Sure Petcare. Animo Activity Tracker – Behaviour Monitor: Dashboard [Зображення : електронний ресурс]. – 2019. – Режим доступу: https://blogger.googleusercontent.com/img/b/R29vZ2xl/AVvXsEjSFwxyYxUitP8_fNn3nBJfg8z5_FeioclNit24re475anK_JV3MOzqg-KUuzlz5j0I2iOGMKDGvWKv66esBiBAmUt02u_-fMJb0FZGZ_nH4a3WZ7yTL5kzEedPUJVEqfd4EIRP2DgriyU/s1600/Sure-Petcare-Animo-Activity-Tracker-Behaviour-Monitor-Review-Aramis-Dashboard-January-2019.jpg (дата звернення: 08.04.2025).

11) Mozilla Developer Network [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org (дата звернення: 27.04.2025).

12) WordPress.org [Електронний ресурс]. – Режим доступу: https://wordpress.org (дата звернення: 27.04.2025).

13) W3Techs. WordPress Usage Statistics [Електронний ресурс]. – Режим доступу: https://w3techs.com/technologies/details/cm-wordpress (дата звернення: 27.04.2025).

14) Flask Documentation [Електронний ресурс]. – Режим доступу: https://flask.palletsprojects.com (дата звернення: 27.04.2025).

15) Express.js [Електронний ресурс]. – Режим доступу: https://expressjs.com (дата звернення: 27.04.2025).

16) Microsoft Learn. .NET Documentation [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com/en-us/dotnet (дата звернення: 27.04.2025).

17) Microservices.io [Електронний ресурс]. – Режим доступу: https://microservices.io (дата звернення: 27.04.2025).

18) React Documentation [Електронний ресурс]. – Режим доступу: https://react.dev (дата звернення: 27.04.2025).

19) Redux Documentation [Електронний ресурс]. – Режим доступу: https://redux.js.org (дата звернення: 29.04.2025).

20) Angular Documentation [Електронний ресурс]. – Режим доступу: https://angular.io/docs (дата звернення: 29.04.2025).

21) Vue.js [Електронний ресурс]. – Режим доступу: https://vuejs.org (дата звернення: 29.04.2025).

22) Next.js Documentation [Електронний ресурс]. – Режим доступу: https://nextjs.org/docs (дата звернення: 27.04.2025).

23) PostgreSQL Documentation [Електронний ресурс]. – Режим доступу: https://www.postgresql.org/docs/ (дата звернення: 29.04.2025).

24) MongoDB Documentation [Електронний ресурс]. – Режим доступу: https://www.mongodb.com/docs/ (дата звернення: 29.04.2025).

25) Hugging Face. Transformers Documentation [Електронний ресурс]. – Режим доступу: https://huggingface.co/docs/transformers (дата звернення: 01.05.2025).

26) TensorFlow [Електронний ресурс]. – Режим доступу: https://www.tensorflow.org (дата звернення: 01.05.2025).

27) Scikit-learn [Електронний ресурс]. – Режим доступу: https://scikit-learn.org/stable/ (дата звернення: 01.05.2025).

28) Socket.IO Documentation [Електронний ресурс]. – Режим доступу: https://socket.io/docs (дата звернення: 01.05.2025).

29) Cohn M. Estimating with Use Case Points [Електронний ресурс]. – Режим доступу: https://www.mountaingoatsoftware.com/articles/estimating-with-use-case-points (дата звернення: 01.05.2025).

30) C4 Model. Context, Containers, Components, and Code [Електронний ресурс]. – Режим доступу: https://c4model.com/ (дата звернення: 01.05.2025).

31) Prisma Documentation [Електронний ресурс]. – Режим доступу: https://www.prisma.io/docs (дата звернення: 01.05.2025).

32) Keras API Documentation [Електронний ресурс]. – Режим доступу: https://keras.io/api (дата звернення: 01.05.2025).

33) shadcn/ui [Електронний ресурс]. – Режим доступу: https://ui.shadcn.com (дата звернення: 01.05.2025).

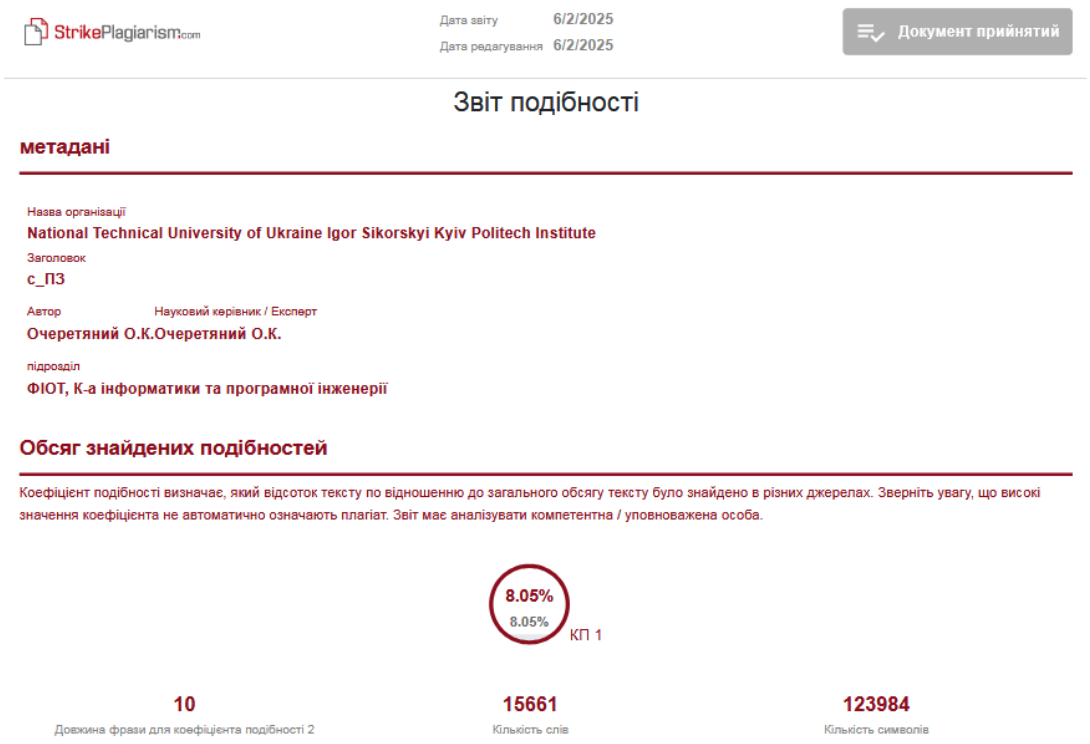
34) SonarSource. SonarQube [Електронний ресурс]. – Режим доступу: https://www.sonarsource.com/products/sonarqube/ (дата звернення: 16.05.2025).

35) Vercel [Електронний ресурс]. – Режим доступу: https://vercel.com/ (дата звернення: 17.05.2025).

36) Render. Deploys Documentation [Електронний ресурс]. – Режим доступу: https://render.com/docs/deploys (дата звернення: 17.05.2025).

37) Supabase. Documentation [Електронний ресурс]. – Режим доступу: https://supabase.com/docs (дата звернення: 17.05.2025).

ДОДАТОК А ЗВІТ ПОДІБНОСТІ



StrikePlagiarism.com

Дата звіту 6/2/2025
Дата редагування 6/2/2025

Документ прийнятий

Звіт подібності

метадані

Назва організації
National Technical University of Ukraine Igor Sikorskyi Kyiv Politech Institute
Заголовок
c_ПЗ
Автор Науковий керівник / Експерт
Очеретяний О.К.Очеретяний О.К.
підрозділ
ФІОТ, К-а інформатики та програмної інженерії

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.

8.05%
8.05%
КП 1

10	15661	123984
Довжина фрази для коефіцієнта подібності 2	Кількість слів	Кількість символів

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2025 р.

ВЕБЗАСТОСУНОК ДЛЯ РАННЬОГО ВИЯВЛЕННЯ ХВОРОБ ДОМАШНІХ ТВАРИН ЗА ДОПОМОГОЮ ШІ ТА ІНТЕРАКТИВНОГО КОНСУЛЬТУВАННЯ З ВЕТЕРИНАРОМ

Текст програми

КПІ.ІП-1507. 045440.03.12

“ПОГОДЖЕНО”
Керівник проєкту:
_____Олександр ОЧЕРЕТЯНИЙ

Нормоконтроль:
_____ Тетяна ШУЛЬКЕВИЧ

Виконавець:
_____ Іван ДАЦЬО

Київ – 2025

Посилання на репозиторій з повним текстом програмного коду

https://github.com/naticw1/diploma

Файл /app/api/pets/route.ts

Реалізація API для списку тварин: отримання тварин користувача та створення нових записів про тварин.

```
const petSchema = z.object({
  name: z.string().min(1, "Ім’я обов’язкове"),
  type: z.string().min(1, "Вид обов’язковий"),
  breed: z.string().optional(),
  age: z.number().min(0).optional().nullable(),
  gender: z.enum(["Чоловіча", "Жіноча"]),
  weight: z.number().min(0).optional().nullable(),
  history: z.string().optional(),
  vaccination: z.string().optional(),
  photoUrls: z
    .array(z.string().url("Некоректний URL фото"))
    .optional()
    .default([]),
});

export async function GET() {
  try {
    const session = await auth();
    if (!session?.user?.id) {
      return NextResponse.json(
        { message: "Неавторизований запит" },
        { status: 401 }
      );
    }

    const pets = await db.pet.findMany({
      where: { ownerId: session.user.id },
      select: { id: true, name: true, type: true, age: true, photoUrls: true },
      orderBy: { name: "asc" },
    });

    return NextResponse.json(pets);
  } catch (error) {
    console.error("Помилка отримання тваринок:", error);
    return NextResponse.json(
      { message: "Не вдалося отримати тваринок" },
      { status: 500 }
    );
  }
}

export async function POST(req: Request) {
  try {
    const session = await auth();
    if (!session?.user?.id) {
      return NextResponse.json(
        { message: "Неавторизований запит" },
        { status: 401 }
      );
    }

    const body = await req.json();
    const validationResult = petSchema.safeParse(body);
```

```
    if (!validationResult.success) {
      console.error(
        "Pet API Validation Error:",
        validationResult.error.flatten()
      );
      return NextResponse.json(
        {
          message: "Помилка валідації даних",
          errors: validationResult.error.flatten().fieldErrors,
        },
        { status: 400 }
      );
    }
    const data = validationResult.data;

    const newPet = await db.pet.create({
      data: {
        ownerId: session.user.id,
        name: data.name,
        type: data.type,
        breed: data.breed,
        age: data.age,
        gender: data.gender,
        weight: data.weight,
        history: data.history,
        vaccination: data.vaccination,
        photoUrls: data.photoUrls,
      },
    });

    console.log("New pet created:", newPet);
    return NextResponse.json(newPet, { status: 201 });
  } catch (error: unknown) {
    console.error("Помилка при додаванні тваринки:", error);
    let message = "Не вдалося створити запис про тваринку";
    if (error instanceof z.ZodError) {
      message = "Помилка валідації даних при збереженні.";
    } else if (error instanceof Prisma.PrismaClientKnownRequestError) {
      message = `Помилка бази даних (${error.code})`;
    } else if (error instanceof Error) {
      message = error.message;
    }
    return NextResponse.json({ message }, { status: 500 });
  }
}
```

Файл /app/api/pets/[id]/route.ts

Реалізація API для конкретної тварини: отримання, оновлення та видалення даних тварини з перевіркою прав доступу.

```
const PetUpdateSchema = z.object({
  name: z.string().min(1, "Ім’я обов’язкове"),
  type: z.string().min(1, "Оберіть вид тварини"),
  breed: z.string().optional().nullable(),
  age: z.preprocess(
    (val) =>
      val === "" || val === null || val === undefined ? null : Number(val),
    z
      .number()
      .min(0, "Вік не може бути від'ємним")
      .max(30, "Нереальний вік")
```

```
      .optional()
      .nullable()
  ),
  gender: z.enum(["Чоловіча", "Жіноча"]),
  weight: z.preprocess(
    (val) =>
      val === "" || val === null || val === undefined ? null : Number(val),
    z
      .number()
      .min(0, "Вага не може бути від'ємною")
      .max(150, "Нереальна вага")
      .optional()
      .nullable()
  ),
  history: z.string().optional().nullable(),
  vaccination: z.string().optional().nullable(),
  photoUrls: z
    .array(z.string().url("Некоректний URL фото"))
    .optional()
    .default([]),
});

const err = (message: string, status = 500) =>
  NextResponse.json({ message }, { status });

export async function GET(
  _req: NextRequest,
  { params }: { params: { id: string } }
) {
  try {
    const session = await auth();
    if (!session?.user?.id || !session?.user?.type) {
      return err("Не авторизовано", 401);
    }
    const currentUserId = session.user.id;
    const currentUserType = session.user.type as UserType;

    const { id: petId } = params;
    if (!petId) return err("ID тваринки не надано", 400);

    const pet = await db.pet.findUnique({
      where: { id: petId },
      include: {
        owner: { select: { id: true } },
        consultations: {
          where: {
            vet: {
              userId:
                currentUserType === UserType.VET ? currentUserId : undefined,
            },
          },
          select: { id: true },
        },
      },
    });

    if (!pet) return err("Тваринку не знайдено", 404);

    const isOwner = pet.ownerId === currentUserId;
    const isRelatedVet =
      currentUserType === UserType.VET && pet.consultations.length > 0;

    if (!isOwner && !isRelatedVet) {
```

```
      console.warn(
        `User ${currentUserId} (type: ${currentUserType}) attempted to access pet ${petId} without permission.`
      );
      return err("Доступ до даних тваринки заборонено", 403);
    }

    const {
      // eslint-disable-next-line @typescript-eslint/no-unused-vars
      owner: _owner,
      // eslint-disable-next-line @typescript-eslint/no-unused-vars
      consultations: _consultations,
      ...petDataToSend
    } = pet;

    return NextResponse.json(petDataToSend);
  } catch (error) {
    const { id: petIdForError } = params;
    console.error(`[PETS_GET_BY_ID_ERROR] for ID ${petIdForError}:`, error);
    return err("Помилка сервера при отриманні даних тваринки");
  }
}

export async function PUT(
  req: NextRequest,
  { params }: { params: { id: string } }
) {
  try {
    const session = await auth();
    if (!session?.user?.id) {
      return err("Не авторизовано", 401);
    }
    const currentUserId = session.user.id;

    const { id } = params;
    if (!id) return err("ID тваринки не надано для оновлення", 400);

    const petToUpdate = await db.pet.findUnique({
      where: { id },
      select: { ownerId: true },
    });

    if (!petToUpdate) {
      return err("Тваринку для оновлення не знайдено", 404);
    }
    if (petToUpdate.ownerId !== currentUserId) {
      console.warn(
        `User ${currentUserId} attempted to update pet ${id} owned by ${petToUpdate.ownerId}`
      );
      return err("Доступ для оновлення заборонено", 403);
    }

    const body = await req.json();
    const validationResult = PetUpdateSchema.safeParse(body);

    if (!validationResult.success) {
      console.error(
        "Pet Update Validation Error:",
        validationResult.error.flatten()
      );
      return NextResponse.json(
        {
          message: "Помилка валідації даних",
          errors: validationResult.error.flatten().fieldErrors,
```

```
        },
        { status: 400 }
      );
    }
    const dataToUpdate = validationResult.data;

    const updatedPet = await db.pet.update({
      where: { id: id },
      data: {
        name: dataToUpdate.name,
        type: dataToUpdate.type,
        breed: dataToUpdate.breed,
        age: dataToUpdate.age,
        gender: dataToUpdate.gender,
        weight: dataToUpdate.weight,
        history: dataToUpdate.history,
        vaccination: dataToUpdate.vaccination,
        photoUrls: dataToUpdate.photoUrls,
      },
    });

    console.log(`Pet ${id} updated successfully by user ${currentUserId}`);
    return NextResponse.json(updatedPet);
  } catch (error: unknown) {
    const { id } = params;
    console.error(`[PETS_PUT_ERROR] for ID ${id}:`, error);
    let message = "Не вдалося оновити тваринку";
    let status = 500;

    if (error instanceof z.ZodError) {
      message = "Помилка валідації даних";
      status = 400;
    } else if (error instanceof Prisma.PrismaClientKnownRequestError) {
      if (error.code === "P2025") {
        message = "Тваринку для оновлення не знайдено";
        status = 404;
      } else {
        message = `Помилка бази даних (${error.code})`;
      }
    } else if (error instanceof Error) {
      message = error.message;
    }
    return err(message, status);
  }
}

export async function DELETE(
  _req: NextRequest,
  { params }: { params: { id: string } }
) {
  try {
    const session = await auth();
    if (!session?.user?.id) {
      return err("Не авторизовано", 401);
    }
    const currentUserId = session.user.id;

    const { id } = params;
    if (!id) return err("ID тваринки не надано для видалення", 400);

    const petToDelete = await db.pet.findUnique({
      where: { id },
      select: { ownerId: true },
```

```
    });

    if (!petToDelete) {
      return err("Тваринку для видалення не знайдено", 404);
    }
    if (petToDelete.ownerId !== currentUserId) {
      console.warn(
        `User ${currentUserId} attempted to delete pet ${id} owned by ${petToDelete.ownerId}`
      );
      return err("Доступ для видалення заборонено", 403);
    }

    await db.pet.delete({ where: { id } });
    console.log(`Pet ${id} deleted successfully by user ${currentUserId}`);
    return NextResponse.json({ message: "Тваринку видалено" });
  } catch (error: unknown) {
    const { id } = params;
    console.error(`[PETS_DELETE_ERROR] for ID ${id}:`, error);
    let message = "Не вдалося видалити тваринку";
    let status = 500;

    if (error instanceof Prisma.PrismaClientKnownRequestError) {
      if (error.code === "P2025") {
        message = "Тваринку для видалення не знайдено";
        status = 404;
      } else {
        message = `Помилка бази даних (${error.code})`;
      }
    } else if (error instanceof Error) {
      message = error.message;
    }
    return err(message, status);
  }
}
```

Файл /app/pets/add/_actions/upload-photo.ts (Server Action)
Реалізація серверної логіки завантаження фото тварини на Supabase Storage з валідацією та поверненням URL.

```
const fileSchema = z
  .instanceof(File)
  .refine((file) => file.size <= 5 * 1024 * 1024, {
    message: `Розмір файлу не повинен перевищувати 5MB.`,
  })
  .refine(
    (file) => ["image/png", "image/jpeg", "image/webp"].includes(file.type),
    { message: "Непідтримуваний тип файлу." }
  );

interface UploadResult {
  publicUrl?: string;
  error?: string;
}

const BUCKET_NAME = "pet-photos";
const supabaseUrl = process.env.NEXT_PUBLIC_SUPABASE_URL!;
const supabaseAnonKey = process.env.NEXT_PUBLIC_SUPABASE_ANON_KEY!;

export async function uploadPetPhotoAction(
  formData: FormData
): Promise<UploadResult> {
  const session = await auth();
```

```
if (!session?.user?.id || !session.supabaseAccessToken) {
  console.error("UploadAction: Auth failed or Supabase token missing.");
  return {
    error:
      "Помилка: Користувач не авторизований або відсутній токен доступу.",
  };
}
const userId = session.user.id;
const userSupabaseToken = session.supabaseAccessToken;

const supabase = createClient(supabaseUrl, supabaseAnonKey, {
  global: { headers: { Authorization: `Bearer ${userSupabaseToken}` } },
  auth: {
    persistSession: false,
    autoRefreshToken: false,
    detectSessionInUrl: false,
  },
});

const file = formData.get("photo") as File | null;
if (!file) {
  return { error: "Файл не знайдено у запиті." };
}

const validationResult = fileSchema.safeParse(file);
if (!validationResult.success) {
  const errMsg =
    validationResult.error.errors[0]?.message || "Некоректний файл.";
  console.error("UploadAction: File validation failed:", errMsg);
  return { error: errMsg };
}
const validatedFile = validationResult.data;

const fileExt = validatedFile.name.split(".").pop();
const fileName = `${crypto.randomUUID()}.${fileExt}`;
const filePath = `${userId}/${fileName}`;

try {
  const { error: uploadError } = await supabase.storage
    .from(BUCKET_NAME)
    .upload(filePath, validatedFile, { cacheControl: "3600", upsert: false });

  if (uploadError) {
    console.error(
      "UploadAction: Storage upload failed:",
      JSON.stringify(uploadError, null, 2)
    );
    if (
      uploadError.message.toLowerCase().includes("policy") ||
      (uploadError as { statusCode?: number })?.statusCode?.toString() ===
        "403"
    ) {
      return { error: "Помилка доступу при завантаженні фото (RLS)." };
    }
    return { error: `Помилка завантаження Supabase: ${uploadError.message}` };
  }

  const { data: urlData } = supabase.storage
    .from(BUCKET_NAME)
    .getPublicUrl(filePath);

  if (!urlData?.publicUrl) {
```

```
      console.error("UploadAction: Could not get public URL for:", filePath);
      return { error: "Не вдалося отримати посилання на фото." };
    }
    return { publicUrl: urlData.publicUrl };
  } catch (error: unknown) {
    const message =
      error instanceof Error ? error.message : "Невідома помилка.";
    console.error("UploadAction: Overall process failed.", message);
    return { error: `Загальна помилка: ${message}` };
  }
}
```

Файл /app/dashboard/_actions/pet-actions.ts (Server Action)
Реалізація серверної логіки видалення тварини: перевірка прав, залежностей (консультації, аналізи) та видалення з ревалідацією кешу.

```
export async function deletePetAction(
  petId: string
): Promise<{ success?: boolean; error?: string }> {
  const session = await auth();
  if (!session?.user?.id) {
    return { error: "Не авторизовано." };
  }
  const currentUserId = session.user.id;

  if (!petId || typeof petId !== "string") {
    return { error: "Некоректний ID тваринки." };
  }

  try {
    const pet = await db.pet.findUnique({
      where: { id: petId },
      select: { ownerId: true, name: true },
    });

    if (!pet) {
      return { error: "Тваринку не знайдено." };
    }

    if (pet.ownerId !== currentUserId) {
      console.warn(
        `User ${currentUserId} attempted to delete pet ${petId} owned by ${pet.ownerId}`
      );
      return { error: "У вас немає прав для видалення цієї тваринки." };
    }

    const relatedConsultationsCount = await db.consultation.count({
      where: { petId: petId },
    });

    if (relatedConsultationsCount > 0) {
      return {
        error: `Неможливо видалити тваринку "${pet.name}", оскільки вона має ${relatedConsultationsCount} пов'язаних консультацій. Будь ласка, спочатку видаліть або змініть ці консультації.`,
      };
    }

    const relatedSymptomAnalysesCount = await db.symptomAnalysis.count({
      where: { petId: petId },
    });

    if (relatedSymptomAnalysesCount > 0) {
```

```
      return {
        error: `Неможливо видалити тваринку "${pet.name}", оскільки вона має ${relatedSymptomAnalysesCount} збережених аналізів симптомів. Будь ласка, спочатку видаліть ці аналізи.`,
      };
    }

    await db.pet.delete({
      where: { id: petId },
    });

    console.log(
      `Pet ${petId} (${pet.name}) deleted successfully by user ${currentUserId}`
    );

    revalidatePath("/dashboard");
    revalidatePath("/pets");
    revalidatePath("/diagnosis");

    return { success: true };
  } catch (error: unknown) {
    console.error(`Error deleting pet ${petId} (action):`, error);

    if (error instanceof Prisma.PrismaClientKnownRequestError) {
      if (error.code === "P2003") {
        return {
          error:
            "Не вдалося видалити тваринку через непередбачені пов'язані записи. Зверніться до адміністратора.",
        };
      }
      if (error.code === "P2025") {
        return {
          error:
            "Тваринку для видалення не вдалося знайти (можливо, її вже видалено).",
        };
      }
      return { error: `Помилка бази даних при видаленні (${error.code}).` };
    }
    if (error instanceof Error) {
      return { error: error.message };
    }
    return {
      error: "Сталася невідома помилка під час спроби видалення тваринки.",
    };
  }
}
```

Файл /app/api/diagnosis/route.ts

Реалізація API збереження аналізу симптомів: валідація, перевірка власності тварини, атомарне збереження аналізу та прогнозів.

```
const analysisSaveSchema = z.object({
  petId: z.string().uuid("Некоректний ID тварини"),
  symptomsProvided: z.array(z.string()).min(1, "Потрібен хоча б один симптом"),
  unknownSymptoms: z.array(z.string()).optional().default([]),
  overallSeverity: z.string().min(1, "Не вказано загальну важкість"),
  triggeredWarning: z.boolean().optional().default(false),
  triggeredExplanation: z.string().nullable().optional().default(null),
  predictions: z
    .array(
      z.object({
        diagnosis: z.string().min(1, "Не вказано назву діагнозу"),
        probability: z
```

```
        .number()
        .min(0)
        .max(1, "Імовірність має бути між 0 та 1"),
      severity: z.string().min(1, "Не вказано важкість для діагнозу"),
    })
  )
  .min(1, "Потрібен хоча б один прогноз")
  .max(5, "Забагато прогнозів (макс 5)"),
});

export async function POST(req: NextRequest) {
 try {
  const session = await auth();
  if (!session?.user?.id) {
   return NextResponse.json({ message: "Не авторизовано" }, { status: 401 });
  }
  if (session.user.type !== UserType.OWNER) {
   return NextResponse.json(
    { message: "Тільки власники можуть зберігати аналіз" },
    { status: 403 }
   );
  }
  const currentUserId = session.user.id;

  const body = await req.json();
  const validationResult = analysisSaveSchema.safeParse(body);
  if (!validationResult.success) {
   console.error("Validation Error:", validationResult.error.flatten());
   return NextResponse.json(
    {
     message: "Помилка валідації даних",
     errors: validationResult.error.flatten().fieldErrors,
    },
    { status: 400 }
   );
  }
  const data = validationResult.data;

  const pet = await db.pet.findUnique({
   where: { id: data.petId },
   select: { ownerId: true },
  });
  if (!pet) {
   return NextResponse.json(
    { message: "Тварину не знайдено" },
    { status: 404 }
   );
  }
  if (pet.ownerId !== currentUserId) {
   console.warn(
    `User ${currentUserId} attempted to save analysis for pet ${data.petId} owned by ${pet.ownerId}`
   );
   return NextResponse.json(
    { message: "Доступ до тварини заборонено" },
    { status: 403 }
   );
  }

  const savedAnalysis = await db.$transaction(async (tx) => {
   const analysis = await tx.symptomAnalysis.create({
    data: {
     petId: data.petId,
     symptomsProvided: data.symptomsProvided,
```

```
          unknownSymptoms: data.unknownSymptoms,
          overallSeverity: data.overallSeverity,
          triggeredWarning: data.triggeredWarning,
          triggeredExplanation: data.triggeredExplanation,
        },
        select: { id: true },
      });

      await tx.symptomPrediction.createMany({
        data: data.predictions.map((p) => ({
          analysisId: analysis.id,
          diagnosis: p.diagnosis,
          probability: p.probability,
          severity: p.severity,
        })),
      });
      return analysis;
    });

    console.log(
      `Analysis ${savedAnalysis.id} saved successfully for pet ${data.petId}`
    );
    return NextResponse.json(
      { message: "Результати аналізу збережено", analysisId: savedAnalysis.id },
      { status: 201 }
    );
  } catch (error: unknown) {
    console.error("[SAVE_SYMPTOM_ANALYSIS_ERROR]", error);
    let message = "Внутрішня помилка сервера при збереженні аналізу";
    if (error instanceof Prisma.PrismaClientKnownRequestError) {
      message = `Помилка бази даних (${error.code})`;
    } else if (error instanceof Error) {
      message = error.message;
    }
    return NextResponse.json({ message }, { status: 500 });
  }
}
```

Файл /app/api/pets/[id]/analyses/route.ts

Реалізація API історії аналізів: авторизований доступ та вибірка структурованих даних аналізів для тварини, включаючи топ-1 прогноз.

```
type AnalysisHistoryResponseItem = {
  id: string;
  createdAt: Date;
  overallSeverity: string;
  symptomsProvided: string[];
  predictions: {
    diagnosis: string;
    probability: number;
  }[];
};

interface RouteParams {
  id?: string;
}

interface RouteHandlerContext {
  params: RouteParams;
}

export async function GET(req: NextRequest, { params }: RouteHandlerContext) {
```

```
  try {
    const session = await auth();
    if (!session?.user?.id || !session?.user?.type) {
      return NextResponse.json({ message: "Не авторизовано" }, { status: 401 });
    }
    const currentUserId = session.user.id;
    const currentUserType = session.user.type as UserType;

    const petId = params?.id;
    if (!petId || typeof petId !== "string" || petId.length === 0) {
      console.error("Invalid or missing pet ID in API route params:", params);
      return NextResponse.json(
        { message: "Некоректний ID тварини" },
        { status: 400 }
      );
    }

    const pet = await db.pet.findUnique({
      where: { id: petId },
      select: {
        ownerId: true,
        consultations: {
          where: {
            vet: {
              userId:
                currentUserType === UserType.VET ? currentUserId : undefined,
            },
          },
          select: { id: true },
        },
      },
    });

    if (!pet) {
      return NextResponse.json(
        { message: "Тварину не знайдено" },
        { status: 404 }
      );
    }

    const isOwner = pet.ownerId === currentUserId;
    const isRelatedVet =
      currentUserType === UserType.VET && pet.consultations.length > 0;

    if (!isOwner && !isRelatedVet) {
      console.warn(
        `User ${currentUserId} (type: ${currentUserType}) attempted to access analyses for pet ${petId} without
permission.`
      );
      return NextResponse.json(
        { message: "Доступ до історії аналізів заборонено" },
        { status: 403 }
      );
    }

    const analyses = await db.symptomAnalysis.findMany({
      where: { petId: petId },
      select: {
        id: true,
        createdAt: true,
        overallSeverity: true,
        symptomsProvided: true,
        predictions: {
```

```
        orderBy: { probability: "desc" },
        take: 1,
        select: { diagnosis: true, probability: true },
       },
     },
     orderBy: { createdAt: "desc" },
    });

    return NextResponse.json(analyses as AnalysisHistoryResponseItem[]);
  } catch (error: unknown) {
    const petIdForError = params?.id || "unknown";
    console.error(
      `[API GET_PET_ANALYSES_ERROR] for Pet ID ${petIdForError}:`,
      error
    );
    let message = "Внутрішня помилка сервера при отриманні історії аналізів";
    if (error instanceof Error) message = error.message;
    return NextResponse.json({ message }, { status: 500 });
  }
}
```

Файл /app/api/consultations/route.ts

Реалізація API створення консультацій: валідація, перевірка доступності слоту, належності тварини, атомарне створення консультації та оновлення статусу слоту. Також отримання списку консультацій.

```
const consultationPostSchema = z.object({
  slotId: z.string().uuid("Некоректний формат ID слоту"),
  petId: z.string().uuid("Некоректний формат ID тваринки"),
});

// eslint-disable-next-line @typescript-eslint/no-unused-vars
export async function GET(_req: NextRequest) {
  try {
    const session = await auth();
    if (!session?.user?.id || !session?.user?.type) {
      return NextResponse.json({ message: "Не авторизовано" }, { status: 401 });
    }
    const currentUserId = session.user.id;
    const currentUserType = session.user.type as UserType;

    const consultationsFromDb = await db.consultation.findMany({
      where:
        currentUserType === UserType.OWNER
          ? { userId: currentUserId }
          : { vet: { userId: currentUserId } },
      select: {
        id: true,
        scheduledAt: true,
        status: true,
        pet: { select: { name: true, type: true } },
        user: { select: { id: true, username: true, type: true } },
        vet: {
          select: {
            photoUrl: true,
            user: { select: { id: true, username: true, type: true } },
          },
        },
        messages: {
          orderBy: { timestamp: "desc" },
          take: 1,
          select: { text: true, timestamp: true, imageUrl: true },
```

```
        },
      },
      orderBy: { scheduledAt: "desc" },
    });

    const result: ConsultationListItemDTO[] = consultationsFromDb.map((c) => {
      let otherParticipantData: ConsultationListItemDTO["otherParticipant"];

      if (currentUserType === UserType.OWNER) {
        if (!c.vet || !c.vet.user) {
          otherParticipantData = {
            id: "unknown_vet",
            username: "Невідомий ветеринар",
            type: UserType.VET,
            avatarUrl: null,
          };
        } else {
          otherParticipantData = {
            id: c.vet.user.id,
            username: c.vet.user.username,
            type: UserType.VET,
            avatarUrl: c.vet.photoUrl || null,
          };
        }
      } else {
        if (!c.user) {
          otherParticipantData = {
            id: "unknown_client",
            username: "Невідомий клієнт",
            type: UserType.OWNER,
            avatarUrl: null,
          };
        } else {
          otherParticipantData = {
            id: c.user.id,
            username: c.user.username,
            type: UserType.OWNER,
            avatarUrl: null,
          };
        }
      }

      return {
        id: c.id,
        scheduledAt: c.scheduledAt,
        status: c.status,
        pet: c.pet
          ? { name: c.pet.name, type: c.pet.type || "N/A" }
          : { name: "N/A", type: "N/A" },
        otherParticipant: otherParticipantData,
        lastMessage: c.messages.length > 0 ? c.messages[0] : null,
      };
    });
    return NextResponse.json(result);
  } catch (error) {
    console.error("[GET_CONSULTATIONS_LIST_ERROR]", error);
    return NextResponse.json(
      { message: "Помилка сервера при отриманні списку консультацій" },
      { status: 500 }
    );
  }
}
```

```
export async function POST(req: NextRequest) {
  try {
    const session = await auth();
    if (!session?.user?.id || !session?.user?.type) {
      return NextResponse.json({ message: "Не авторизовано" }, { status: 401 });
    }
    const userType = session.user.type as UserType;
    if (userType !== UserType.OWNER) {
      return NextResponse.json(
        { message: "Тільки клієнти можуть створювати записи" },
        { status: 403 }
      );
    }
    const clientUserId = session.user.id;

    const body = await req.json();
    const validationResult = consultationPostSchema.safeParse(body);
    if (!validationResult.success) {
      return NextResponse.json(
        {
          message: "Помилка валідації даних",
          errors: validationResult.error.flatten().fieldErrors,
        },
        { status: 400 }
      );
    }
    const { slotId, petId } = validationResult.data;

    const newConsultation = await db.$transaction(async (tx) => {
      const slot = await tx.schedule.findUnique({
        where: { id: slotId },
        select: {
          id: true,
          date: true,
          from: true,
          vetId: true,
          isBooked: true,
        },
      });
      if (!slot) throw new Error("SLOT_NOT_FOUND");
      if (slot.isBooked) throw new Error("SLOT_ALREADY_BOOKED");
      if (!slot.vetId) throw new Error("VET_ID_MISSING_IN_SLOT");

      let scheduledAt: Date;
      try {
        const scheduleDatePart = slot.date.toISOString().split("T")[0];
        const scheduledDateTimeString = `${scheduleDatePart}T${slot.from}:00`;
        scheduledAt = new Date(scheduledDateTimeString);
        if (isNaN(scheduledAt.getTime()))
          throw new Error("INVALID_DATE_PARSING");
      } catch {
        throw new Error("INVALID_SLOT_TIME_FORMAT");
      }

      const now = new Date();
      const bufferMinutes = 1;
      const minBookingTime = new Date(
        now.getTime() - bufferMinutes * 60 * 1000
      );
      if (scheduledAt < minBookingTime) throw new Error("SLOT_IN_PAST");

      const pet = await tx.pet.findUnique({
        where: { id: petId },
```

```
      select: { ownerId: true },
    });
    if (!pet) throw new Error("PET_NOT_FOUND");
    if (pet.ownerId !== clientUserId) throw new Error("PET_FORBIDDEN");

    await tx.schedule.update({
      where: { id: slotId, isBooked: false },
      data: { isBooked: true },
    });

    const consultation = await tx.consultation.create({
      data: {
        userId: clientUserId,
        vetId: slot.vetId,
        petId: petId,
        scheduledAt: scheduledAt,
        status: "SCHEDULED",
        scheduleId: slot.id,
      },
      include: {
        user: { select: { id: true, username: true } },
        vet: { include: { user: { select: { id: true, username: true } } } },
        pet: { select: { id: true, name: true } },
        schedule: { select: { id: true, date: true, from: true } },
      },
    });
    return consultation;
  });

  return NextResponse.json(newConsultation, { status: 201 });
 } catch (error: unknown) {
  console.error("[CONSULTATION_POST_ERROR]", error);
  let errorMessageText = "Сталася помилка під час бронювання.";
  let statusCodeValue = 500;
  let errorCode: string | undefined = undefined;

  if (error instanceof Error) {
    errorMessageText = error.message;
    const errorAsAny = error as any;
    if ("code" in errorAsAny && typeof errorAsAny.code === "string")
      errorCode = errorAsAny.code;
  } else {
    errorMessageText = "Сталася невідома помилка сервера.";
  }
  let messageHandled = true;
  switch (errorMessageText) {
    case "SLOT_NOT_FOUND":
      statusCodeValue = 404;
      break;
    case "SLOT_ALREADY_BOOKED":
      statusCodeValue = 409;
      break;
    case "SLOT_IN_PAST":
      statusCodeValue = 400;
      break;
    case "INVALID_DATE_PARSING":
    case "INVALID_SLOT_TIME_FORMAT":
    case "VET_ID_MISSING_IN_SLOT":
      statusCodeValue = 500;
      break;
    case "PET_NOT_FOUND":
      statusCodeValue = 404;
      break;
```

```
      case "PET_FORBIDDEN":
        statusCodeValue = 403;
        break;
      default:
        messageHandled = false;
        break;
    }
    if (!messageHandled) {
      if (errorCode === "P2025") {
        errorMessageText =
          "Не вдалося оновити запис. Можливо, він вже змінений або видалений.";
        statusCodeValue = 409;
      } else if (error instanceof Error && error.message) {
        errorMessageText = error.message;
      } else {
      }
    }
    return NextResponse.json(
      { message: errorMessageText },
      { status: statusCodeValue }
    );
  }
}
```

Файл /app/api/consultations/[consultationId]/cancel/route.ts

Реалізація API скасування консультації: перевірка прав та умов, оновлення статусу, звільнення слоту, створення сповіщень.

```
const ValidConsultationStatuses = {
  SCHEDULED: "SCHEDULED",
  COMPLETED: "COMPLETED",
  CANCELLED_BY_CLIENT: "CANCELLED_BY_CLIENT",
  CANCELLED_BY_VET: "CANCELLED_BY_VET",
  ACTIVE: "ACTIVE",
} as const;

type ConsultationStatus =
  | (typeof ValidConsultationStatuses)[keyof typeof ValidConsultationStatuses]
  | string;

export async function PATCH(
  _req: NextRequest,
  { params }: { params: { consultationId: string } }
) {
  try {
    const session = await auth();
    if (
      !session?.user?.id ||
      !session?.user?.type ||
      !session?.user?.username
    ) {
      return NextResponse.json(
        { message: "Не авторизовано або неповні дані сесії" },
        { status: 401 }
      );
    }
    const currentUserId = session.user.id;
    const currentUserType = session.user.type as UserType;
    const currentUsername = session.user.username;

    const consultationId = params.consultationId;
    if (
```

```
  !consultationId ||
  typeof consultationId !== "string" ||
  consultationId.length < 10
) {
  return NextResponse.json(
    { message: "Некоректний ID консультації" },
    { status: 400 }
  );
}

await db.$transaction(async (tx) => {
  const consultation = await tx.consultation.findUnique({
    where: { id: consultationId },
    select: {
      id: true,
      userId: true,
      vetId: true,
      user: { select: { id: true, username: true } },
      vet: {
        select: {
          id: true,
          userId: true,
          user: { select: { id: true, username: true } },
        },
      },
      pet: { select: { name: true } },
      scheduledAt: true,
      status: true,
      scheduleId: true,
    },
  });

  if (!consultation) throw new Error("NOT_FOUND");

  const ownerUserId = consultation.userId;
  const vetUserId = consultation.vet?.user?.id;
  const isOwner = ownerUserId === currentUserId;
  const isVet = vetUserId === currentUserId;

  if (!isOwner && !isVet) throw new Error("FORBIDDEN");
  if (consultation.status !== ValidConsultationStatuses.SCHEDULED)
    throw new Error("INVALID_STATUS");
  if (isPast(new Date(consultation.scheduledAt)))
    throw new Error("ALREADY_PAST");

  const newStatus: ConsultationStatus =
    currentUserType === UserType.OWNER
      ? ValidConsultationStatuses.CANCELLED_BY_CLIENT
      : ValidConsultationStatuses.CANCELLED_BY_VET;

  await tx.consultation.update({
    where: { id: consultationId },
    data: { status: newStatus },
  });

  if (consultation.scheduleId) {
    await tx.schedule
      .update({
        where: { id: consultation.scheduleId },
        data: { isBooked: false },
      })
      .catch((error) => {
        if (
```

```
            error instanceof Prisma.PrismaClientKnownRequestError &&
            error.code === "P2025"
          ) {
            console.warn(
              `Schedule slot ${consultation.scheduleId} not found while cancelling consultation ${consultationId}. Ignoring.`
            );
          } else {
            console.error(
              `Error updating schedule slot ${consultation.scheduleId} during cancellation:`,
              error
            );
            throw error;
          }
        });
      } else {
        console.warn(
          `Consultation ${consultationId} cannot update schedule: scheduleId is null.`
        );
      }

      const recipientUserId = isOwner ? vetUserId : ownerUserId;
      const petName = consultation.pet?.name || "вашої тваринки";
      const scheduledTimeFormatted = format(
        new Date(consultation.scheduledAt),
        "d MMM HH:mm",
        { locale: uk }
      );

      if (recipientUserId) {
        const notificationMessage =
          currentUserType === UserType.OWNER
            ? `Клієнт ${currentUsername} скасував(ла) консультацію для "${petName}" (${scheduledTimeFormatted}).`
            : `Ветеринар ${currentUsername} скасував(ла) Вашу консультацію для "${petName}" (${scheduledTimeFormatted}).`;
        try {
          await tx.notification.create({
            data: {
              userId: recipientUserId,
              message: notificationMessage,
              consultationId: consultation.id,
              link: "/dashboard/consultations",
            },
          });
        } catch (notificationError) {
          console.error(
            `Failed to create notification for user ${recipientUserId} about consultation ${consultationId}:`,
            notificationError
          );
        }
      } else {
        console.warn(
          `Could not determine recipient user ID for cancellation notification of ${consultationId}. VetUserID: ${vetUserId}, OwnerUserID: ${ownerUserId}`
        );
      }
    });

    return NextResponse.json(
      { message: "Консультацію скасовано" },
      { status: 200 }
    );
```

```
  } catch (error: unknown) {
    console.error("[CANCEL_CONSULTATION_ERROR]", error);
    const defaultErrorMessage = "Внутрішня помилка сервера при скасуванні";
    const defaultStatusCode = 500;

    let errorMessage: string = defaultErrorMessage;
    let statusCode: number = defaultStatusCode;

    if (error instanceof Error) {
      switch (error.message) {
        case "NOT_FOUND":
          errorMessage = "Консультацію не знайдено";
          statusCode = 404;
          break;
        case "FORBIDDEN":
          errorMessage = "У вас немає прав для скасування цієї консультації";
          statusCode = 403;
          break;
        case "INVALID_STATUS":
          errorMessage =
            "Консультацію не можна скасувати, оскільки вона вже не запланована";
          statusCode = 400;
          break;
        case "ALREADY_PAST":
          errorMessage =
            "Консультацію не можна скасувати, оскільки її час вже минув";
          statusCode = 400;
          break;
        default:
          if (error.message && error.message !== defaultErrorMessage) {
            errorMessage = error.message;
          }
          break;
      }
    } else if (typeof error === "string") {
      errorMessage = error;
    }

    if (error instanceof Prisma.PrismaClientKnownRequestError) {
      errorMessage = `Помилка бази даних при скасуванні (${error.code})`;
    }

    return NextResponse.json({ message: errorMessage }, { status: statusCode });
  }
}
```

Файл pages/api/socket/io.ts

Реалізація Socket.IO сервера: автентифікація, управління кімнатами чатів, отримання, збереження та трансляція повідомлень чату (текст, зображення) в реальному часі.

```
const NEXTAUTH_SECRET = process.env.NEXTAUTH_SECRET;
if (!NEXTAUTH_SECRET) {
  console.error(
    "FATAL ERROR: NEXTAUTH_SECRET is not set in environment variables!"
  );
}

export const config = { api: { bodyParser: false } };

const ioHandler = async (req: NextApiRequest, res: NextApiResponseServerIO) => {
  if (req.url?.startsWith("/api/socket/init") && !req.url.includes("?EIO=4")) {
    if (!res.headersSent && !res.writableEnded) {
```

```
    res
      .status(404)
      .send(
        "This specific path /api/socket/init is handled by its own API route for warmups."
      );
  }
  return;
}

if (!res.socket.server.io) {
  console.log(
    "[Socket.IO] Initializing Socket.IO server for /api/socket/io..."
  );
  const httpServer: NetServer = res.socket.server as any;
  const io = new SocketIOServer<
    ClientToServerEvents,
    ServerToClientEvents,
    InterServerEvents,
    SocketData
  >(httpServer, {
    path: "/api/socket/io",
    addTrailingSlash: false,
    cors: {
      origin: process.env.NEXT_PUBLIC_APP_URL || "http://localhost:3000",
      methods: ["GET", "POST"],
    },
  });

  io.on("connection", async (socket) => {
    console.log(`[Socket.IO] Client connected: ${socket.id}`);
    const cookie = socket.handshake.headers.cookie;
    if (!cookie) {
      console.warn(
        `[Socket.IO] Socket ${socket.id} no cookie, disconnecting.`
      );
      socket.disconnect(true);
      return;
    }

    let token;
    try {
      token = await getToken({
        req: { headers: { cookie } } as any,
        secret: NEXTAUTH_SECRET,
      });
    } catch (e: any) {
      console.error(
        `[Socket.IO] Error calling getToken for ${socket.id}:`,
        e.message
      );
      socket.disconnect(true);
      return;
    }

    if (!token) {
      console.warn(
        `[Socket.IO] Socket ${socket.id} - getToken returned null. Disconnecting.`
      );
      socket.disconnect(true);
      return;
    }

    const userId = token?.sub;
```

```
if (!userId) {
  console.warn(
    `[Socket.IO] Socket ${socket.id} not authorized (no userId). Disconnecting.`
  );
  socket.disconnect(true);
  return;
}
socket.data.userId = userId;
console.log(
  `[Socket.IO] Socket authenticated: ${socket.id}, User: ${userId}`
);

socket.on("join_room", async (consultationId: string) => {
  console.log(
    `[Socket.IO] User ${userId} joining room: ${consultationId}`
  );
  try {
    const access = await db.consultation.findFirst({
      where: {
        id: consultationId,
        OR: [{ userId }, { vet: { user: { id: userId } } }],
      },
      select: { id: true },
    });
    if (!access) {
      console.warn(
        `[Socket.IO] User ${userId} NO ACCESS to room: ${consultationId}`
      );
      socket.emit("message_error", {
        consultationId,
        error: "Доступ до цієї кімнати заборонено.",
      });
      return;
    }
    await socket.join(consultationId);
    console.log(
      `[Socket.IO] User ${userId} JOINED room: ${consultationId}`
    );
  } catch (error) {
    console.error(
      `[Socket.IO] Error join_room for ${consultationId} by ${userId}:`,
      error
    );
    socket.emit("message_error", {
      consultationId,
      error: "Помилка приєднання до кімнати.",
    });
  }
});

socket.on("leave_room", async (consultationId: string) => {
  await socket.leave(consultationId);
  console.log(`[Socket.IO] User ${userId} LEFT room: ${consultationId}`);
});

socket.on(
  "send_message",
  async (
    data: SendMessagePayload,
    callback?: (response: SendMessageAckResponse) => void
  ) => {
    const { consultationId, senderId, text, imageUrl, imageMeta } = data;
    console.log(
```

```
  `[Socket.IO] Received 'send_message' from User: ${
    socket.data.userId
  } for CId: ${consultationId}. Callback provided: ${
    typeof callback === "function"
  }`
);

if ((!text || !text.trim()) && !imageUrl) {
  if (typeof callback === "function")
    callback({
      status: "error",
      error: "Порожнє повідомлення не дозволено.",
    });
  return;
}
if (!consultationId || !senderId) {
  if (typeof callback === "function")
    callback({
      status: "error",
      error: "Неправильні дані повідомлення (відсутній ID).",
    });
  return;
}
if (senderId !== socket.data.userId) {
  if (typeof callback === "function")
    callback({ status: "error", error: "Недійсний відправник." });
  return;
}

try {
  const access = await db.consultation.findFirst({
    where: {
      id: consultationId,
      OR: [{ userId: senderId }, { vet: { user: { id: senderId } } }],
    },
    select: { id: true },
  });
  if (!access) {
    if (typeof callback === "function")
      callback({
        status: "error",
        error:
          "Немає доступу до надсилання повідомлень в цій консультації.",
      });
    return;
  }

  const newMessage = await db.message.create({
    data: {
      consultationId,
      senderId,
      text: text ? text.trim() : null,
      imageUrl: imageUrl || null,
      imageMeta: imageMeta
        ? (imageMeta as Prisma.JsonValue)
        : Prisma.JsonNull,
    },
    include: {
      sender: { select: { id: true, username: true, type: true } },
    },
  });

  const senderData = newMessage.sender;
```

```
        let senderEffectiveAvatarUrl: string | null = null;
        if (senderData.type === UserType.VET) {
          const vetProfile = await db.vetProfile.findUnique({
            where: { userId: senderData.id },
            select: { photoUrl: true },
          });
          senderEffectiveAvatarUrl = vetProfile?.photoUrl || null;
        }

        const messageToSend: MessageWithSender = {
          ...newMessage,
          sender: {
            id: senderData.id,
            username: senderData.username,
            type: senderData.type,
            avatarUrl: senderEffectiveAvatarUrl,
          },
        };

        io.to(consultationId).emit("receive_message", messageToSend);
        console.log(
          `[Socket.IO] Message (ID: ${newMessage.id}) broadcasted to room ${consultationId}`
        );

        if (typeof callback === "function") {
          callback({ status: "ok", messageId: newMessage.id });
        }
      } catch (err: any) {
        console.error(
          `[Socket.IO] Error send_message for ${consultationId} by ${senderId}:`,
          err.message,
          err.stack
        );
        if (typeof callback === "function") {
          callback({
            status: "error",
            error: "Не вдалося зберегти або надіслати повідомлення.",
          });
        }
      }
    }
  );

  socket.on("disconnect", (reason) => {
    console.log(
      `[Socket.IO] Client disconnected: ${socket.id}, User: ${socket.data.userId}, Reason: ${reason}`
    );
  });
});

 global.io = io;
 res.socket.server.io = io;
 console.log("[Socket.IO] Server instance created and attached.");
} else {
 // console.log("[Socket.IO] Server already running.");
}

if (!req.url?.includes("?EIO=4") && !res.writableEnded && !res.headersSent) {
 res
  .status(400)
  .send(
    "This endpoint is primarily for Socket.IO connections. Please use a Socket.IO client."
  );
```

```
  }
};
export default ioHandler;
```

Файл /app/api/vet-schedule/route.ts

Реалізація API управління розкладом: отримання та збереження масиву робочих слотів ветеринара.

```
export async function GET(req: NextRequest) {
  const session = await auth();

  if (!session?.user?.id) {
    return NextResponse.json({ message: "Unauthorized" }, { status: 401 });
  }

  const vetId = await db.vetProfile.findUnique({
    where: { userId: session.user.id },
    select: { id: true },
  });

  if (!vetId) {
    return NextResponse.json(
      { message: "Vet profile not found" },
      { status: 404 }
    );
  }

  const slots = await db.schedule.findMany({
    where: { vetId: vetId.id },
    select: {
      date: true,
      from: true,
      to: true,
    },
    orderBy: { date: "asc" },
  });

  return NextResponse.json(slots);
}

export async function POST(req: NextRequest) {
  const session = await auth();
  if (!session?.user?.id) {
    return NextResponse.json({ message: "Unauthorized" }, { status: 401 });
  }

  const vet = await db.vetProfile.findUnique({
    where: { userId: session.user.id },
  });

  if (!vet) {
    return NextResponse.json(
      { message: "Vet profile not found" },
      { status: 404 }
    );
  }

  const body = await req.json();

  // Очистка старого розкладу
  await db.schedule.deleteMany({ where: { vetId: vet.id } });
```

```
  const validSlots = body
    // eslint-disable-next-line @typescript-eslint/no-explicit-any
    .filter((s: any) => s.date && s.from && s.to)
    // eslint-disable-next-line @typescript-eslint/no-explicit-any
    .map((s: any) => ({
      vetId: vet.id,
      date: new Date(s.date),
      from: s.from,
      to: s.to,
    }));

  await db.schedule.createMany({ data: validSlots });

  return NextResponse.json({ message: "Schedule saved" });
}
```

Файл /app/api/cron/send-notifications/route.ts

Реалізація CRON-задачі для періодичного пошуку запланованих консультацій та автоматичного створення сповіщень про нагадування або початок для учасників.

```
const CRON_SECRET = process.env.CRON_SECRET;

export async function GET(req: NextRequest) {
  const authorization = req.headers.get("authorization");

  if (process.env.NODE_ENV === "production") {
    if (!CRON_SECRET) {
      console.error("CRON_SECRET is not defined.");
      return NextResponse.json(
        { error: "Internal configuration error" },
        { status: 500 }
      );
    }
    if (`Bearer ${CRON_SECRET}` !== authorization) {
      console.warn("Unauthorized Cron Job access attempt.");
      return NextResponse.json({ error: "Unauthorized" }, { status: 401 });
    }
  }
  console.log("Running Cron Job: Send Consultation Notifications (UTC Based)");

  try {
    const nowUtc = new Date();

    const reminderWindowStartUtc = addMinutes(nowUtc, 9);
    const reminderWindowEndUtc = addMinutes(nowUtc, 11);

    // console.log(`[CRON DEBUG] Now UTC: ${nowUtc.toISOString()}`);
    // console.log(`[CRON DEBUG] Reminder Window UTC: ${reminderWindowStartUtc.toISOString()} to
${reminderWindowEndUtc.toISOString()}`);

    const consultationsToRemind = await db.consultation.findMany({
      where: {
        status: "SCHEDULED",
        notificationSentAt: null,
        scheduledAt: {
          gte: reminderWindowStartUtc,
          lte: reminderWindowEndUtc,
        },
      },
      include: {
        user: { select: { id: true, username: true } },
        vet: { include: { user: { select: { id: true, username: true } } } },
```

```
      pet: { select: { name: true } },
    },
  });

  console.log(
    `Reminders (10 min): Found ${consultationsToRemind.length} consultations.`
  );

  const reminderNotificationsData = [];
  for (const consult of consultationsToRemind) {
    if (
      !consult.user?.id ||
      !consult.vet?.user?.id ||
      !consult.pet?.name ||
      !consult.user?.username ||
      !consult.vet?.user?.username
    ) {
      console.warn(
        `Skipping reminder for consultation ${consult.id} due to missing data.`
      );
      continue;
    }
    const link = `/dashboard/consultations/${consult.id}`;
    reminderNotificationsData.push({
      userId: consult.user.id,
      consultationId: consult.id,
      message: `Нагадування: Консультація для ${consult.pet.name} з ветеринаром ${consult.vet.user.username} розпочнеться приблизно за 10 хвилин.`,
      link,
      isRead: false,
    });
    reminderNotificationsData.push({
      userId: consult.vet.user.id,
      consultationId: consult.id,
      message: `Нагадування: Консультація для ${consult.pet.name} (${consult.user.username}) розпочнеться приблизно за 10 хвилин.`,
      link,
      isRead: false,
    });
  }

  if (reminderNotificationsData.length > 0) {
    const createdNotifications = await db.notification.createMany({
      data: reminderNotificationsData,
      skipDuplicates: true,
    });
    console.log(
      `Created ${createdNotifications.count} reminder notifications.`
    );
    const remindedConsultationIds = consultationsToRemind.map((c) => c.id);
    await db.consultation.updateMany({
      where: { id: { in: remindedConsultationIds } },
      data: { notificationSentAt: nowUtc },
    });
    console.log(
      `Marked ${remindedConsultationIds.length} consultations as reminded.`
    );
  }

  const startWindowStartUtc = subMinutes(nowUtc, 1);
  const startWindowEndUtc = addMinutes(nowUtc, 1);
```

```
    // console.log(`[CRON DEBUG] Start Window UTC: ${startWindowStartUtc.toISOString()} to ${startWindowEndUtc.toISOString()}`);

    const consultationsToStart = await db.consultation.findMany({
      where: {
        status: "SCHEDULED",
        scheduledAt: {
          gte: startWindowStartUtc,
          lte: startWindowEndUtc,
        },
      },
      include: {
        user: { select: { id: true, username: true } },
        vet: { include: { user: { select: { id: true, username: true } } } },
        pet: { select: { name: true } },
      },
    });

    console.log(
      `Starting now: Found ${consultationsToStart.length} consultations.`
    );

    const startNotificationsData = [];
    const startedConsultationIds = [];
    for (const consult of consultationsToStart) {
      if (
        !consult.user?.id ||
        !consult.vet?.user?.id ||
        !consult.pet?.name ||
        !consult.user?.username ||
        !consult.vet?.user?.username
      ) {
        console.warn(
          `Skipping start notification for consultation ${consult.id} due to missing data.`
        );
        continue;
      }
      const link = `/dashboard/consultations/${consult.id}`;
      startNotificationsData.push({
        userId: consult.user.id,
        consultationId: consult.id,
        message: `Консультація для ${consult.pet.name} з ветеринаром ${consult.vet.user.username} розпочалася. Перейдіть до чату.`,
        link,
        isRead: false,
      });
      startNotificationsData.push({
        userId: consult.vet.user.id,
        consultationId: consult.id,
        message: `Консультація для ${consult.pet.name} (${consult.user.username}) розпочалася. Перейдіть до чату.`,
        link,
        isRead: false,
      });
      startedConsultationIds.push(consult.id);
    }

    if (startNotificationsData.length > 0) {
      await db.$transaction([
        db.notification.createMany({
          data: startNotificationsData,
          skipDuplicates: true,
        }),
```

```
        db.consultation.updateMany({
          where: { id: { in: startedConsultationIds }, status: "SCHEDULED" },
          data: { status: "ACTIVE" }, // Оновлюємо статус на ACTIVE
        }),
      ]);
      console.log(
        `Created ${startNotificationsData.length} start notifications and updated status for
${startedConsultationIds.length} consultations.`
      );
    }

    return NextResponse.json({
      success: true,
      reminded: consultationsToRemind.length,
      started: consultationsToStart.length,
    });
  } catch (error: unknown) {
    console.error("Cron Job Error:", error);
    let errorMessage = "Internal Server Error";
    if (error instanceof Error) errorMessage = error.message;
    return NextResponse.json(
      { success: false, message: errorMessage },
      { status: 500 }
    );
  }
}
```

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2025 р.

ВЕБЗАСТОСУНОК ДЛЯ РАННЬОГО ВИЯВЛЕННЯ ХВОРОБ ДОМАШНІХ ТВАРИН ЗА ДОПОМОГОЮ ШІ ТА ІНТЕРАКТИВНОГО КОНСУЛЬТУВАННЯ З ВЕТЕРИНАРОМ

Програма та методика тестування

КПІ.ІП-1507.045440.04.51

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Олександр ОЧЕРЕТЯНИЙ

Нормоконтроль:
_____ Тетяна ШУЛЬКЕВИЧ

Виконавець:
_____ Іван ДАЦЬО

Київ – 2025

ЗМІСТ

1 ОБ’ЄКТ ВИПРОБУВАНЬ

Об’єктом випробування є розроблений вебзастосунок «Vai», призначений для раннього виявлення хвороб домашніх тварин за допомогою штучного інтелекту та інтерактивного консультування з ветеринаром. Тестуванню підлягають функціональні вимоги, користувацький інтерфейс та коректність роботи інтегрованого модуля штучного інтелекту. Потрібно переконатися що вебзастосунок працює на основних браузерах на комп’ютерів та мобільних телефонах.

2 МЕТА ТЕСТУВАННЯ

Метою тестування є наступне:

- перевірка правильності роботи програмного забезпечення відповідно до функціональних вимог;
- перевірка збереження даних;
- перевірка сумісності вебзастосунка з останніми версіями сучасних браузерів (Google Chrome, Mozilla Firefox, Apple Safari, Microsoft Edge (на базі Chromium));
- перевірка сумісності застосунку з різними операційними системами (Windows, macOS, Linux, Android та iOS);
- знаходження проблем, помилок і недоліків з метою їх усунення;
- перевірка зручності графічного інтерфейсу;

3 МЕТОДИ ТЕСТУВАННЯ

Для тестування програмного забезпечення використовуються такі методи:

- статичне тестування – перевіряється програма разом з усією документацією, яка аналізується на предмет дотримання стандартів програмування;
- динамічне тестування – застосовується в процесі виконання програми. Коректність програмного засобу перевіряється на певній кількості тестів. При прогоні кожного з них збираються та аналізуються дані про проблеми та помилки в роботі програми;
- функціональне тестування – полягає у перевірці відповідності реальної поведінки програмного забезпечення очікуваній;
- системне тестування – перевіряється усе програмне забезпечення в цілому;
- мануальне тестування – тестування без використання автоматизації, тест-кейси пише особа, що тестує програмне забезпечення;
- тестування «чорної скриньки» – об’єктом тестування тут є функції присутні у програмі. Перевіряється коректність вихідних даних при заданих вхідних;
- тестування «білої скриньки» – об’єктом тестування тут є внутрішня поведінка програми. Перевіряється коректність побудови всіх елементів програми та правильність їхньої взаємодії один з одним;
- тестування «сірої скриньки» – об’єктом тестування тут є деякі особливості внутрішньої поведінки програми. Перевіряється коректність вихідних даних при заданих вхідних, застосовується для тестування окремих алгоритмів (функцій).

4 ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Під час проведення тестування будуть використовуватись наступні допоміжні засоби:

- сучасні веббраузери для виконання мануального тестування інтерфейсу та функціоналу;
- вбудовані інструменти розробника у веббраузерах для аналізу елементів сторінки, мережевих запитів та відлагодження клієнтського коду
- система статичного аналізу коду SonarQube для оцінки якості кодової бази та виявлення потенційних вразливостей;
- утиліта Postman для ручного тестування API-ендпоінтів серверної частини;
- інтегроване середовище розробки Visual Studio Code для можливості відлагодження програмного коду на етапах тестування.

Порядок проведення тестування буде наступним:

- проведення мануального динамічного тестування для перевірки відповідності функціональним вимогам за розробленими тест-кейсами;
- тестування користувацького інтерфейсу на коректність відображення, адаптивність та взаємодію на настільних комп'ютерах;
- тестування адаптивності інтерфейсу та сумісності на мобільних пристроях (смартфонах, планшетах) з різними роздільними здатностями екранів та операційними системами (Android, iOS);
- оцінка зручності використання та інтуїтивності навігації;
- перевірка коректності обміну даними між усіма компонентами системи, включаючи модуль штучного інтелекту.

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2025 р.

ВЕБЗАСТОСУНОК ДЛЯ РАННЬОГО ВИЯВЛЕННЯ ХВОРОБ ДОМАШНІХ ТВАРИН ЗА ДОПОМОГОЮ ШІ ТА ІНТЕРАКТИВНОГО КОНСУЛЬТУВАННЯ З ВЕТЕРИНАРОМ

Керівництво користувача

КПІ.ІП-1507.045440.05.34

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Олександр ОЧЕРЕТЯНИЙ

Нормоконтроль:
_____ Тетяна ШУЛЬКЕВИЧ

Виконавець:
_____ Іван ДАЦЬО

Київ – 2025

ЗМІСТ

1 ПРИЗНАЧЕННЯ ПРОГРАМИ

«Vai» – це вебзастосунок, призначений для раннього виявлення хвороб домашніх тварин за допомогою інтегрованого модуля штучного інтелекту та для забезпечення інтерактивного консультування з ветеринарними фахівцями. Застосунок розроблено для власників домашніх тварин, які прагнуть отримати попередню оцінку стану здоров'я своїх улюбленців та зручний доступ до онлайн-консультацій, а також для ветеринарів, які можуть надавати дистанційні консультаційні послуги та ефективно керувати своїм робочим процесом.

Вебзастосунок «Vai» функціонує як онлайн-сервіс, доступний користувачам через стандартні веббраузери на персональних комп’ютерах, планшетах та мобільних телефонах, і не потребує встановлення окремої інсталяційної версії.

Основний функціонал вебзастосунку включає:

– реєстрацію, автентифікацію та керування обліковими записами користувачів (власників тварин та ветеринарів;

– створення та управління профілями домашніх тварин, дозволяючи власникам зберігати та редагувати інформацію про своїх улюбленців;

– аналіз симптомів тварини за допомогою модуля штучного інтелекту для надання попередніх діагностичних припущень та рекомендацій;

– пошук ветеринарних фахівців за різними критеріями (наприклад, спеціалізація) та перегляд їхніх детальних профілів;

– запис на онлайн-консультацію до обраного ветеринара, виходячи з його доступного розкладу;

– проведення текстових онлайн-консультацій в режимі реального часу між власником тварини та ветеринаром з можливістю обміну повідомленнями та зображеннями;

– можливість для ветеринарів формувати та керувати своїм робочим розкладом та доступними часовими слотами для консультацій.

2 ПІДГОТОВКА ДО РОБОТИ З ПРОГРАМНИМ ЗАБЕЗПЕЧЕННЯМ

2.1 Системні вимоги для коректної роботи

Для успішної роботи даного застосунку необхідне виконання наступних вимог.

Мінімальні системні вимоги для персональних комп'ютерів (ПК) та ноутбуків:

– Операційна система: сучасні оновлені версії Windows, macOS або Linux;

– Веббраузер: актуальна версія Google Chrome, Mozilla Firefox, Apple Safari або Microsoft Edge (на базі Chromium) з підтримкою HTML5, CSS3 та JavaScript (ES6+);

– Процесор: двоядерний процесор (наприклад, Intel Core i3 або аналогічний);

– Оперативна пам'ять (ОЗП): 4 ГБ;

– Підключення до мережі Інтернет: стабільне з'єднання зі швидкістю від 1 Мбіт/с.

Рекомендовані системні вимоги для персональних комп'ютерів (ПК) та ноутбуків:

– Операційна система: сучасні оновлені версії Windows, macOS або Linux;

– Веббраузер: остання стабільна версія Google Chrome, Mozilla Firefox, Apple Safari або Microsoft Edge (на базі Chromium);

– Процесор: чотирьохядерний процесор (наприклад, Intel Core i5 або аналогічний) або краще;

– Оперативна пам'ять (ОЗП): 8 ГБ або більше;

– Підключення до мережі Інтернет: стабільне з'єднання зі швидкістю від 10 Мбіт/с (особливо для онлайн-консультацій з обміном зображеннями).

Мінімальні системні вимоги для мобільних пристроїв:

– Операційна система: остання стабільна версія Android або iOS;

– Веббраузер: актуальна версія системного браузера (наприклад, Chrome для Android, Safari для iOS) з підтримкою сучасних вебтехнологій;

– Процесор: сучасний мобільний процесор (наприклад, рівня Snapdragon 4xx серії / Apple A9 або аналогічний);

– Оперативна пам'ять (ОЗП): 2 ГБ;

– Підключення до мережі Інтернет: стабільне з'єднання зі швидкістю від 1 Мбіт/с.

Рекомендовані системні вимоги для мобільних пристроїв:

– Операційна система: остання стабільна версія Android або iOS;

– Веббраузер: остання стабільна версія системного браузера;

– Процесор: продуктивний сучасний мобільний процесор (наприклад, рівня Snapdragon 6xx серії / Apple A11 Bionic або аналогічний) або краще;

– Оперативна пам'ять (ОЗП): 4 ГБ або більше;

– Підключення до мережі Інтернет: стабільне з'єднання зі швидкістю від 10 Мбіт/с.

2.2 Завантаження застосунку

Вебзастосунок не потребує завантаження програмних файлів.

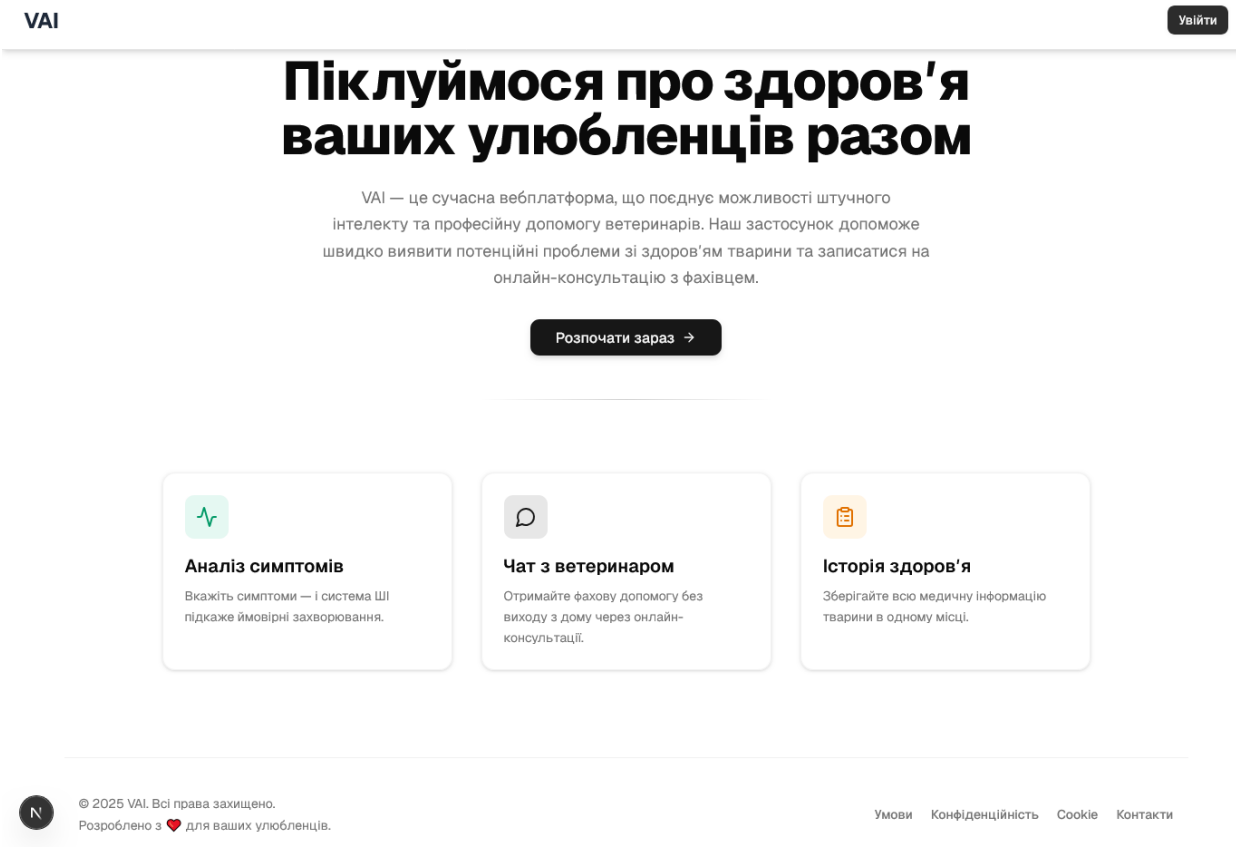
2.3 Перевірка коректної роботи

Оскільки вебзастосунок «Vai» є онлайн-сервісом і не потребує встановлення на пристрій користувача, перевірка його коректної роботи полягає у верифікації успішного доступу до нього через веббраузер після його розгортання на сервері.

Для перевірки коректної роботи вебзастосунку «Vai», користувачу необхідно відкрити сумісний веббраузер на своєму пристрої та ввести у адресний рядок надану URL-адресу застосунку.

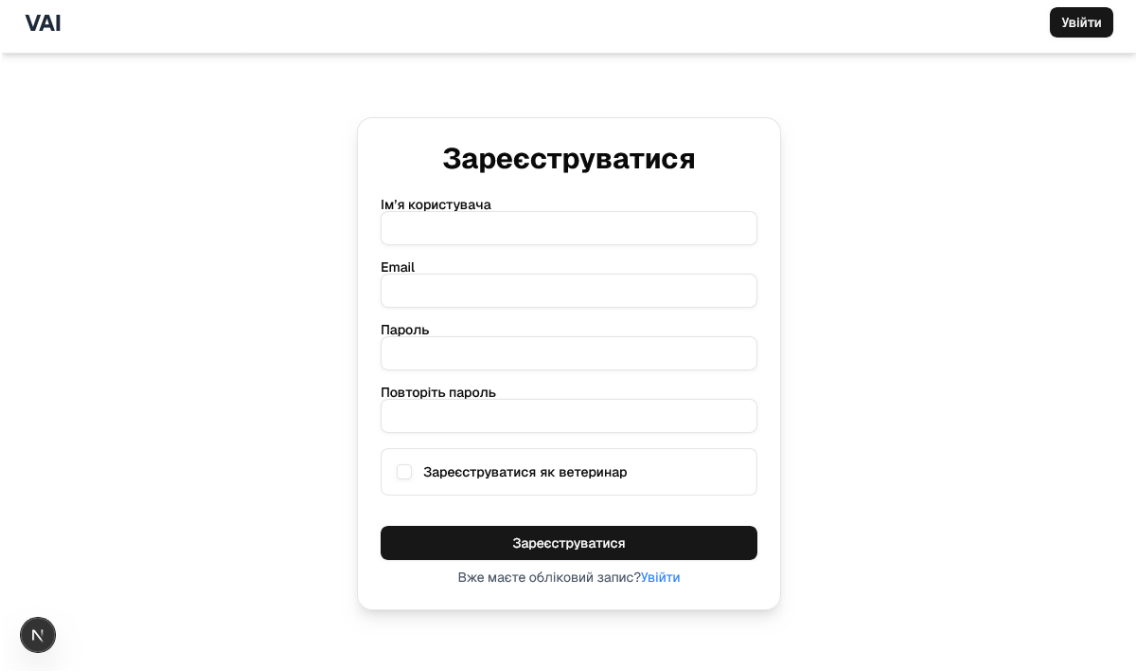
3 ВИКОНАННЯ ПРОГРАМИ

Для початку роботи з вебзастосунком «Vai» користувач відкриває веббраузер та переходить за URL-адресою застосунку. На головній сторінці неавторизованому користувачеві пропонуються варіанти «Увійти» або «Розпочати зараз».

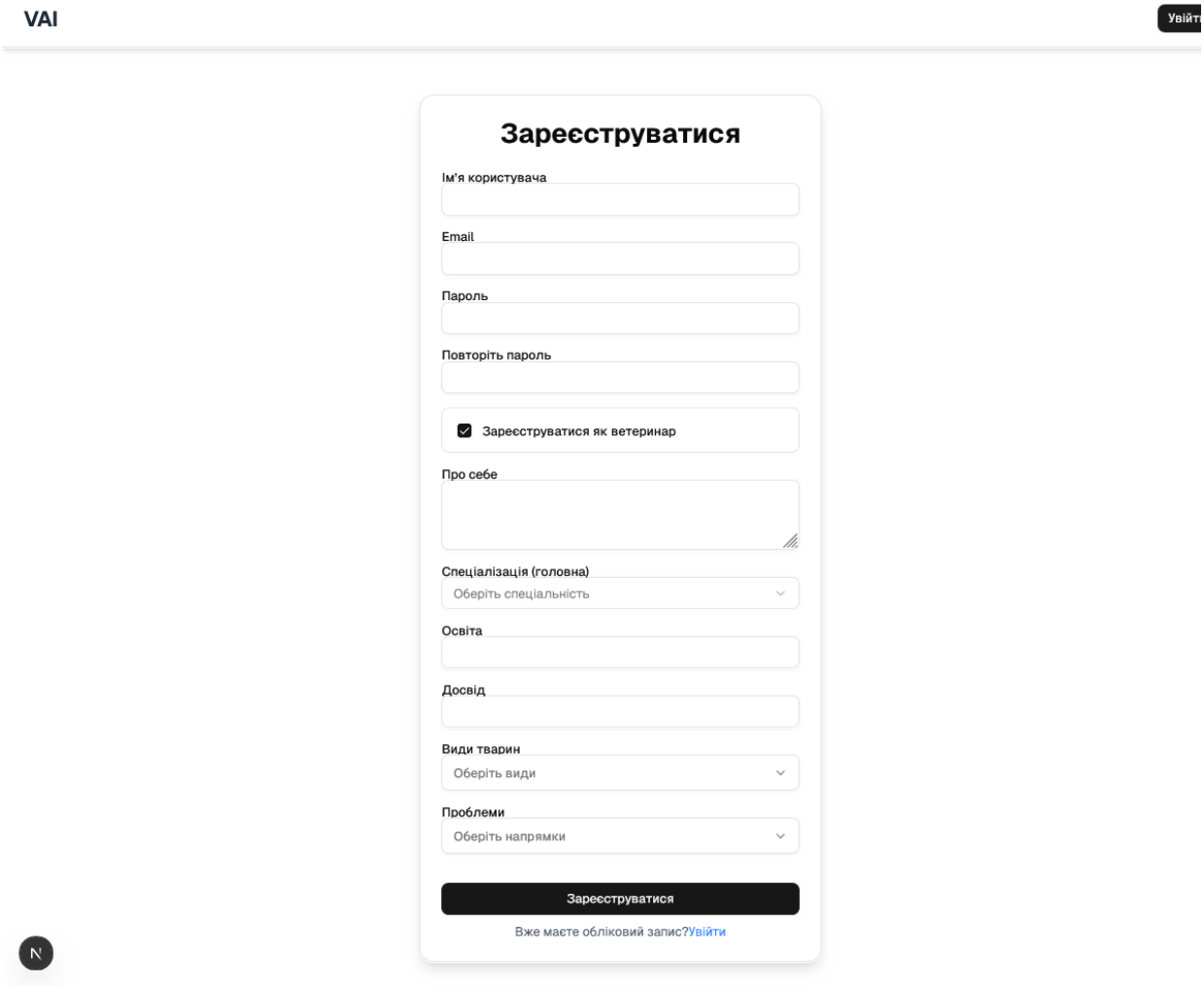


Рисунку 3.1 – Головна сторінка вебзастосунку "Vai" для неавторизованого користувача

Для створення нового облікового запису користувач обирає опцію «Розпочати зараз». На сторінці реєстрації необхідно заповнити поля: ім'я користувача, адреса електронної пошти, пароль та його підтвердження. Якщо користувач є ветеринаром, він встановлює прапорець «Я ветеринар», що відкриває додаткові поля для введення професійної інформації (спеціалізація, досвід тощо). Після заповнення форми та натискання кнопки «Зареєструватися», у разі успіху, користувача буде перенаправлено на сторінку входу.

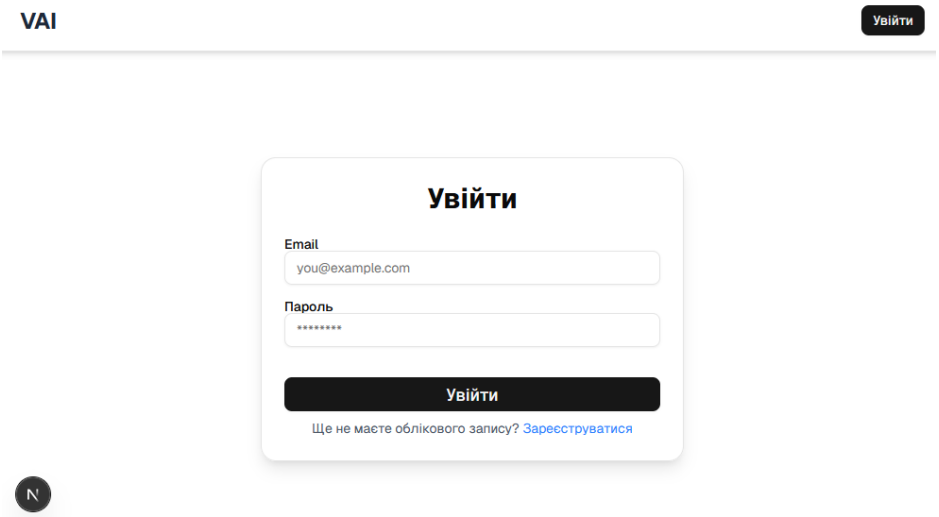


Рисунку 3.2 – Сторінка реєстрації нового користувача

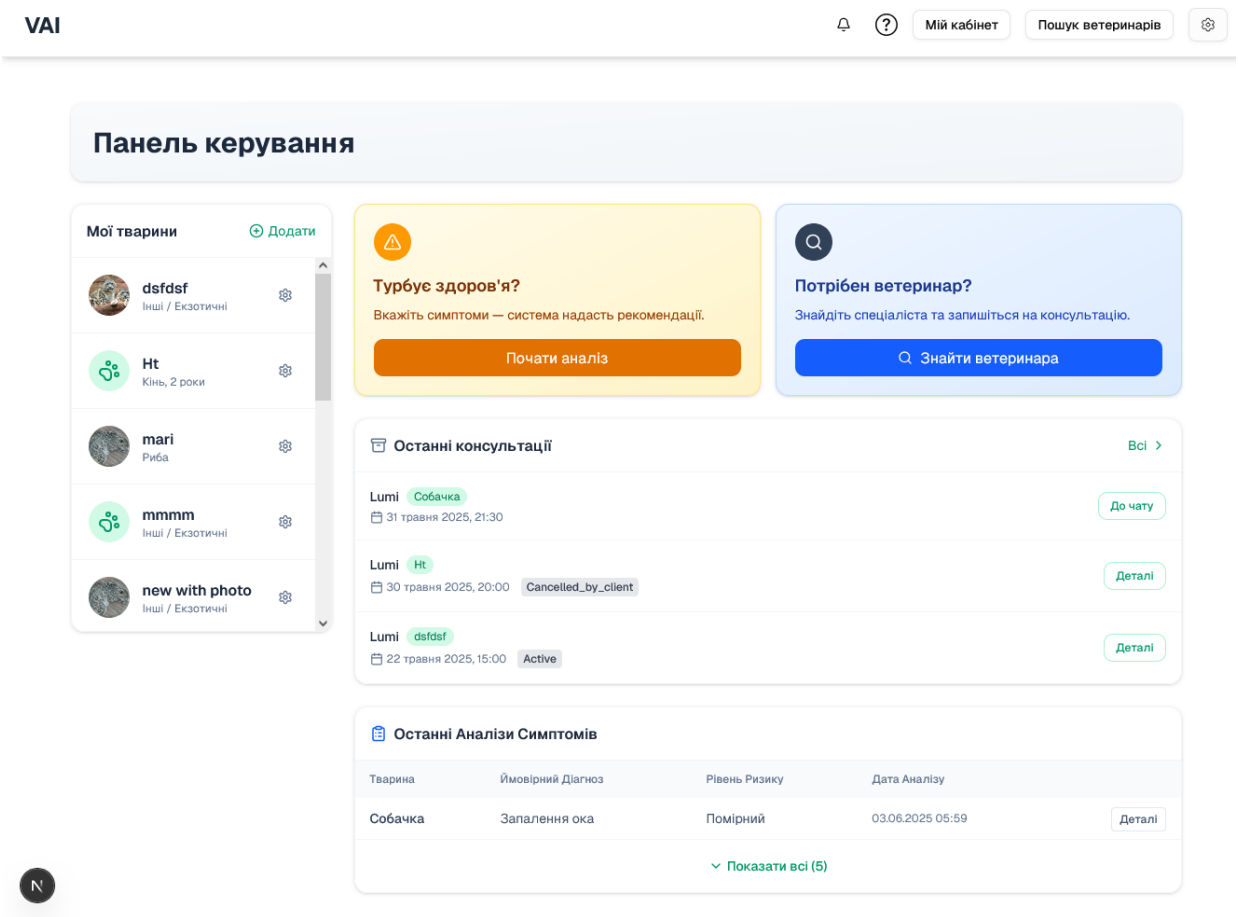


Рисунку 3.3 – Сторінка реєстрації з додатковими полями для ветеринара

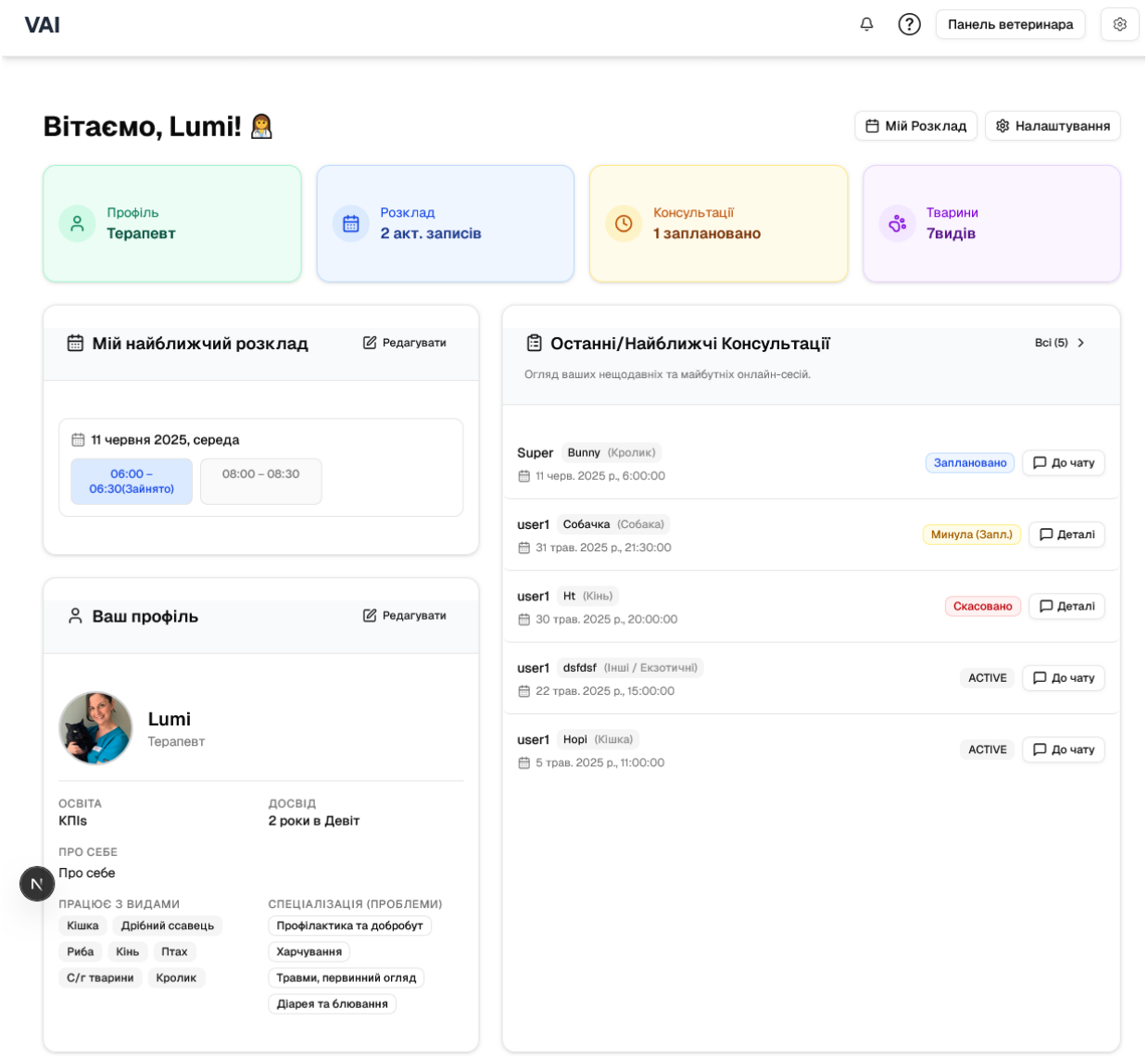
Для входу в існуючий обліковий запис користувач обирає опцію «Увійти» та вводить свою електронну пошту і пароль. Після успішної авторизації користувач потрапляє на свою персональну панель керування.



Рисунку 3.4 – Сторінка авторизації (входу) користувача

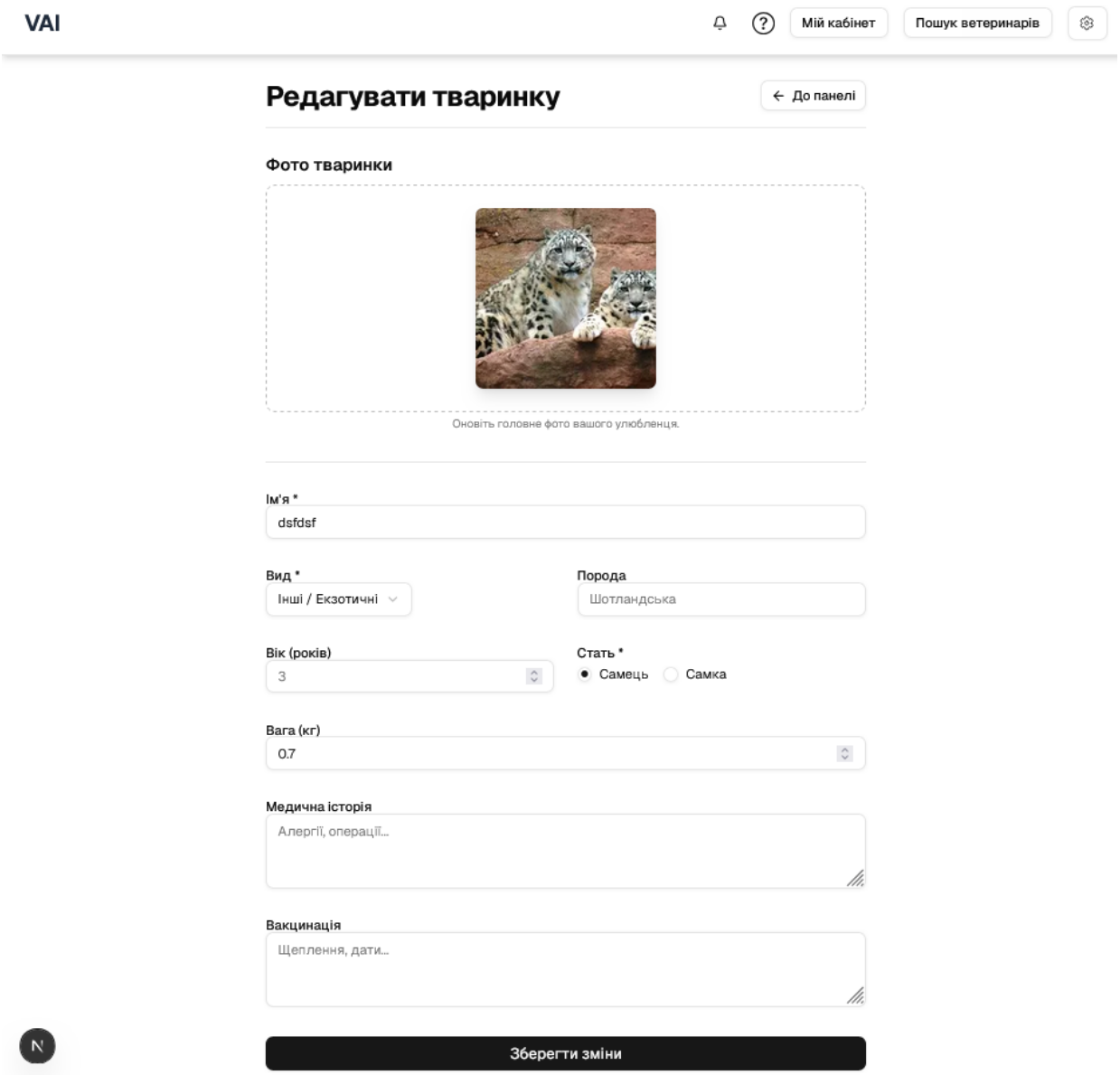


Рисунку 3.5 – Панель керування Власника тварини після авторизації



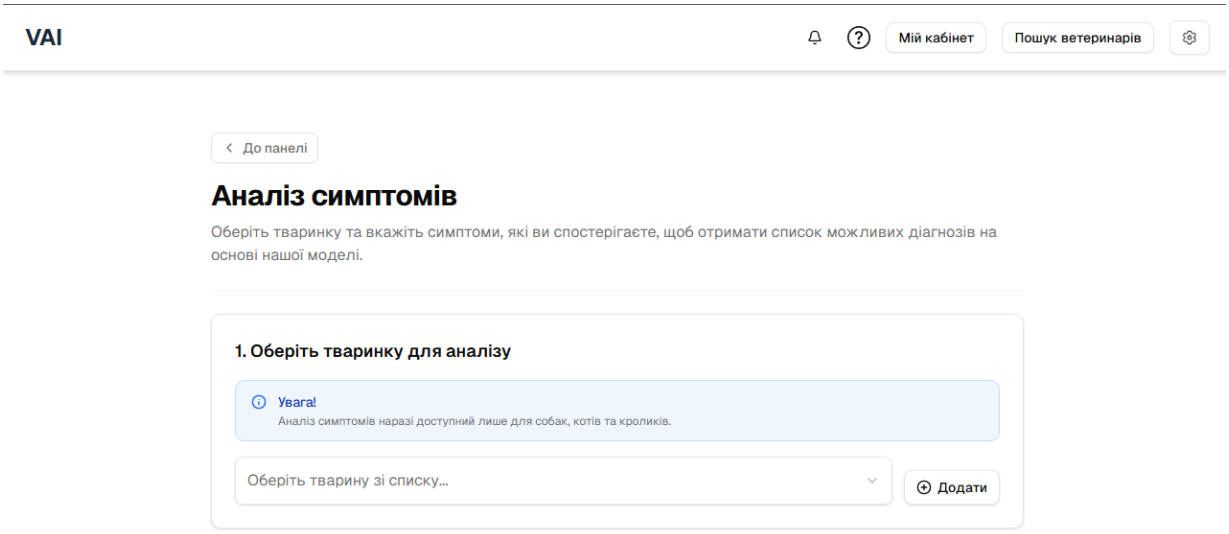
Рисунку 3.6 – Панель керування Ветеринара після авторизації

Власник тварини на своїй панелі керування може перейти до розділу «Мої тварини» для управління інформацією про своїх улюбленців. Тут він може додати нову тваринку, заповнивши форму з її даними (кличка, вид, порода, вік, фото тощо), редагувати інформацію про вже доданих тварин або видалити їхні профілі.



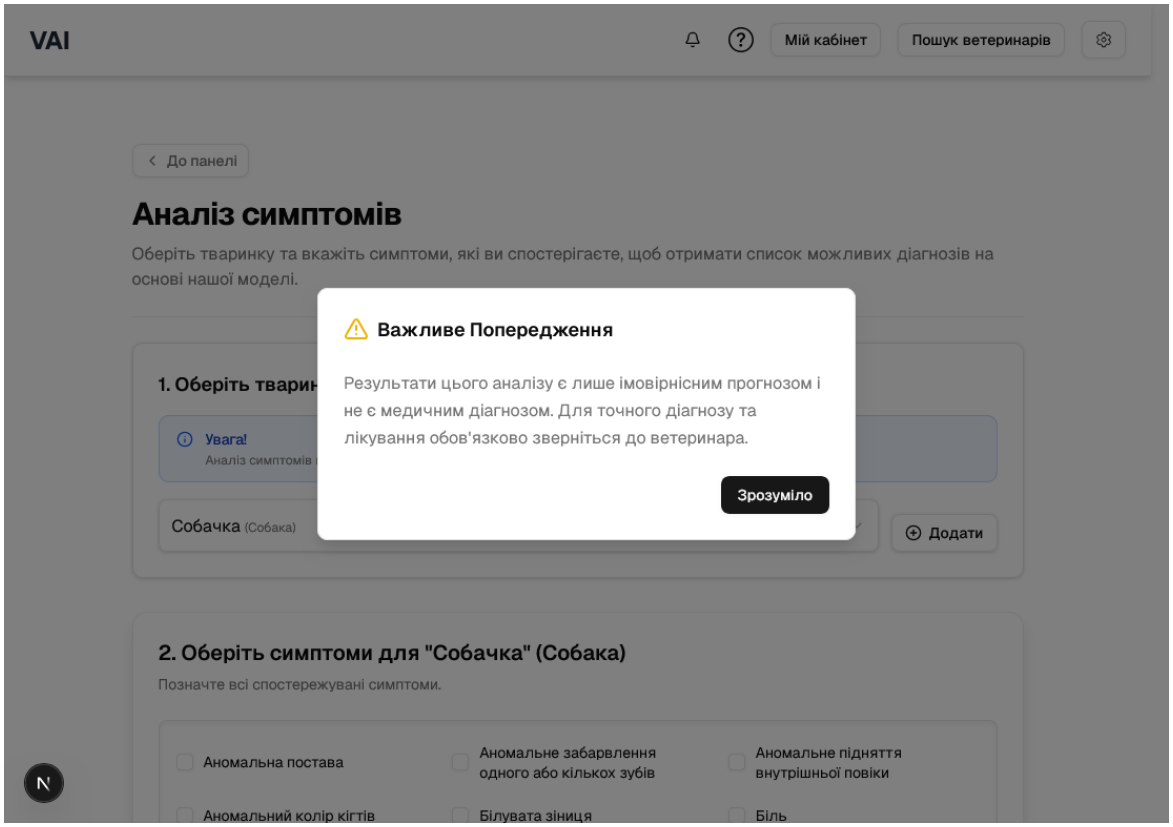
Рисунку 3.7 – Форма додавання/редагування даних про тварину

Для отримання попередньої оцінки стану здоров'я свого улюбленця користувач (Власник тварини) переходить до розділу "Аналіз симптомів". На сторінці він обирає конкретну тварину зі списку раніше доданих.



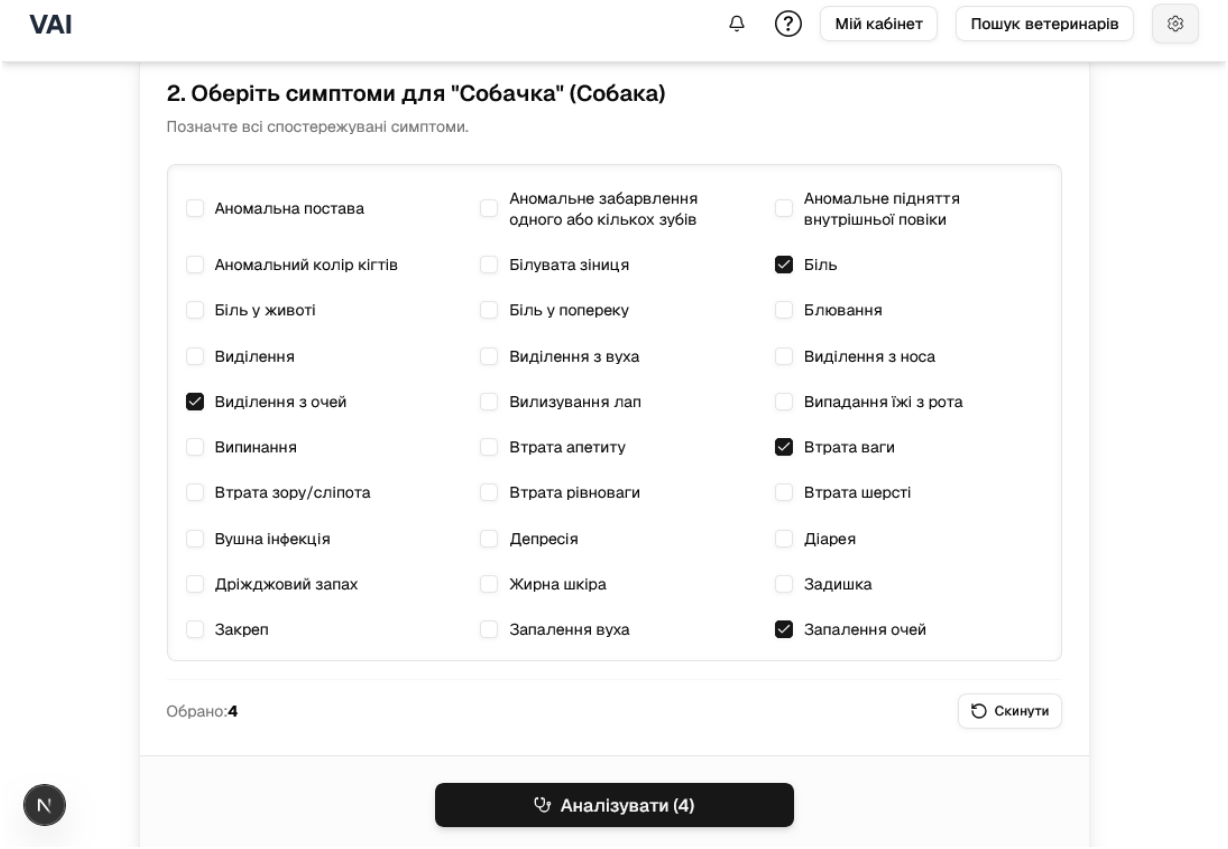
Рисунку 3.8 – Сторінка вибору тварини для аналізу симптомів

Перед початком аналізу система виводить попередження про імовірнісний характер результатів, яке користувач підтверджує.



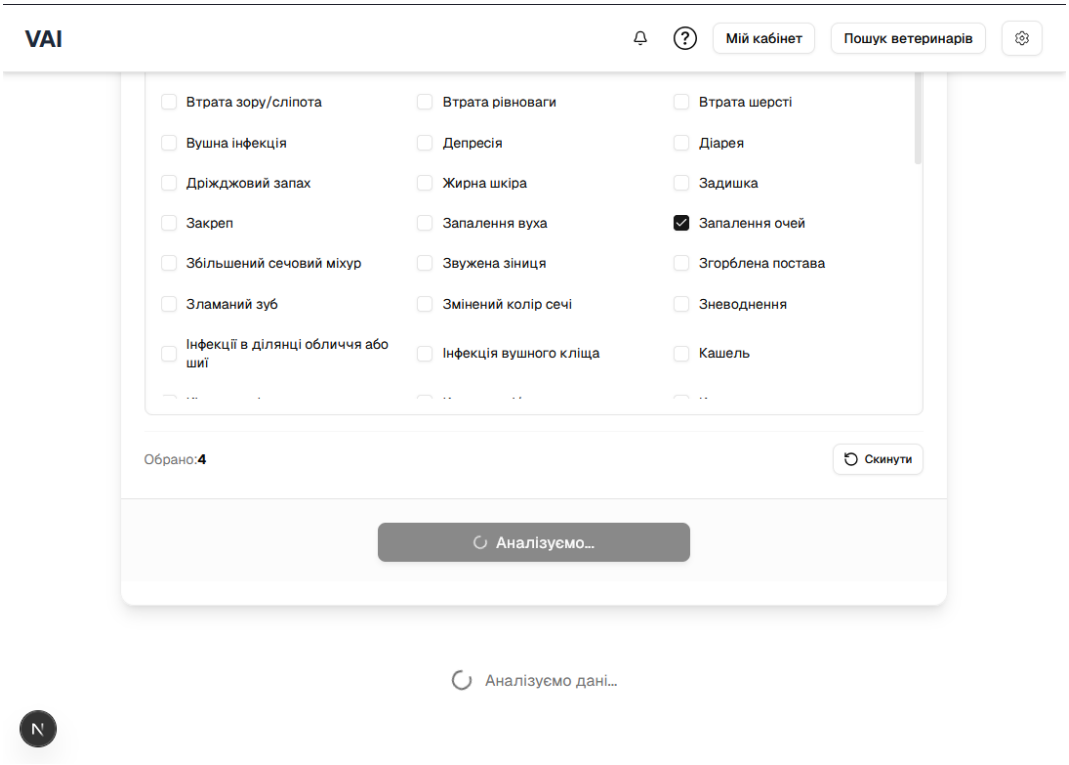
Рисунку 3.9 – Вікно попередження перед аналізом симптомів

Далі відкривається інтерфейс, де власник відмічає спостережувані симптоми.



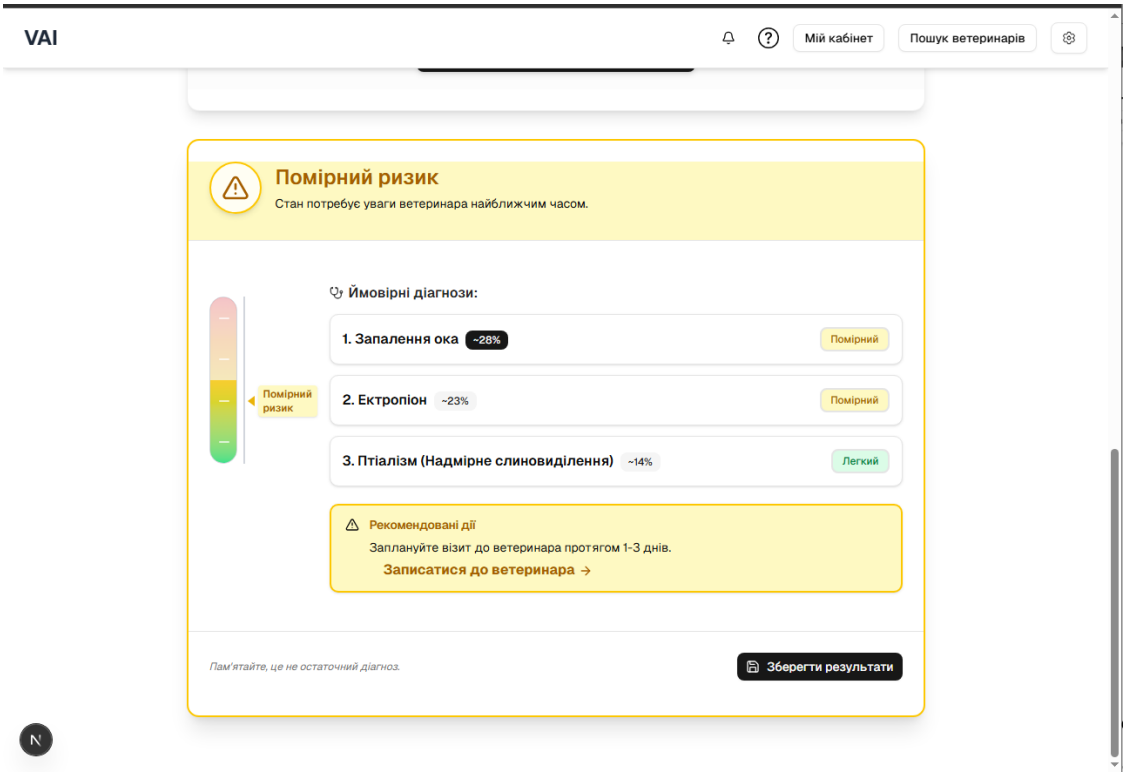
Рисунку 3.10 – Інтерфейс вибору симптомів тварини

Після вибору симптомів користувач натискає кнопку "Аналізувати", ініціюючи запит до ШІ-моделі. Під час обробки відображається відповідний індикатор.

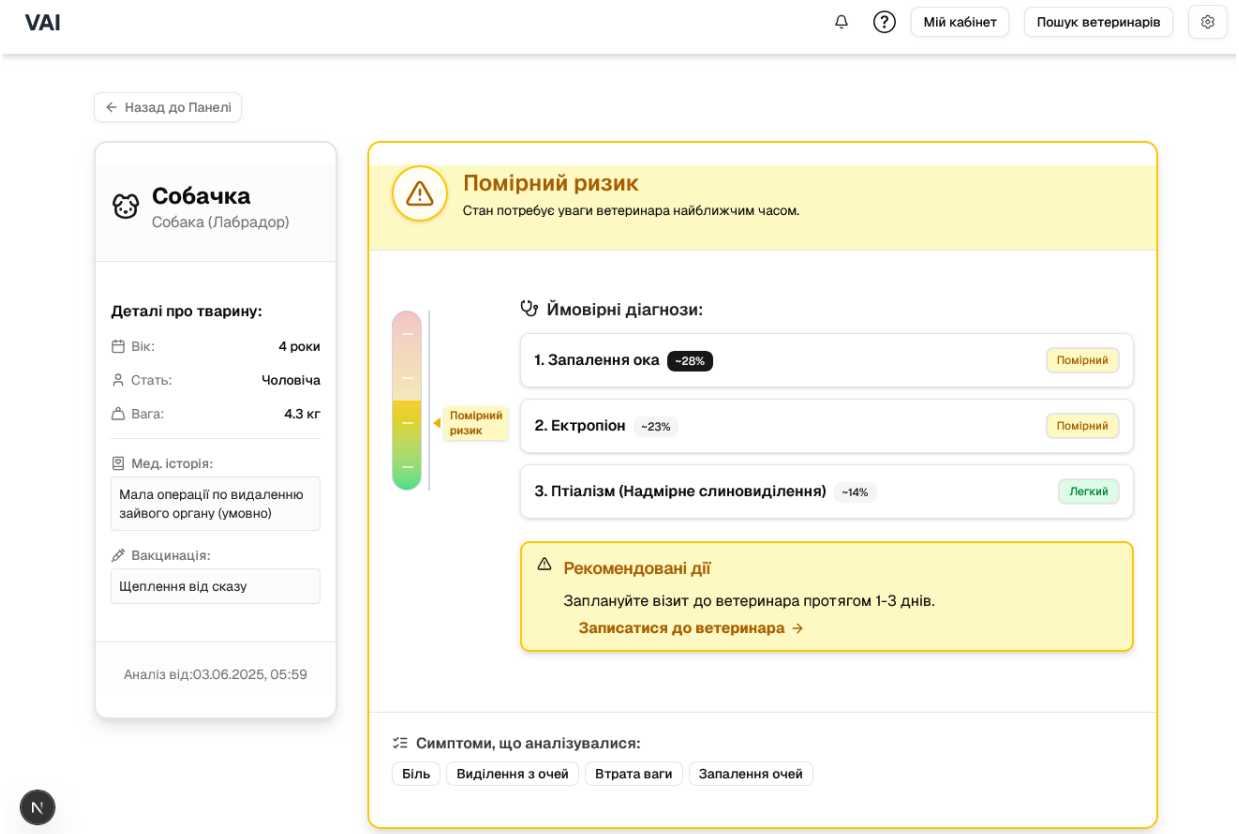


Рисунку 3.11 – Процес аналізу симптомів

По завершенні на сторінці з'являються результати: перелік імовірних діагнозів з відсотками, загальний рівень серйозності та рекомендації. Користувачеві також пропонуються опції для подальших дій. Результати аналізу можна зберегти.

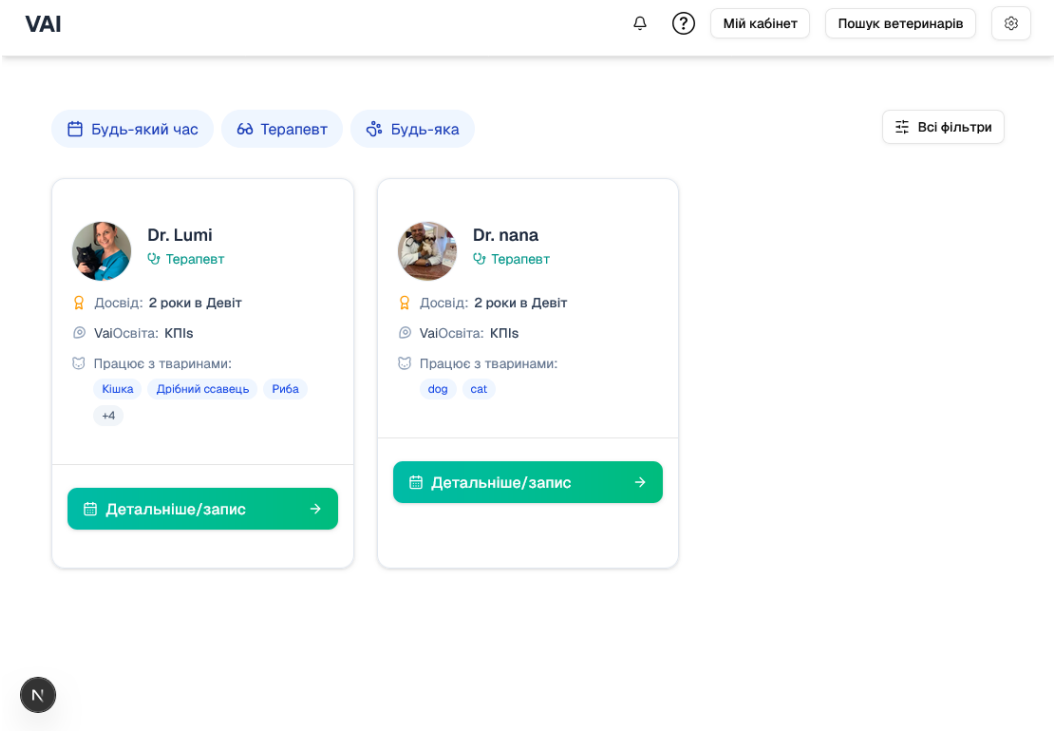


Рисунку 3.12 – Відображення результатів аналізу симптомів



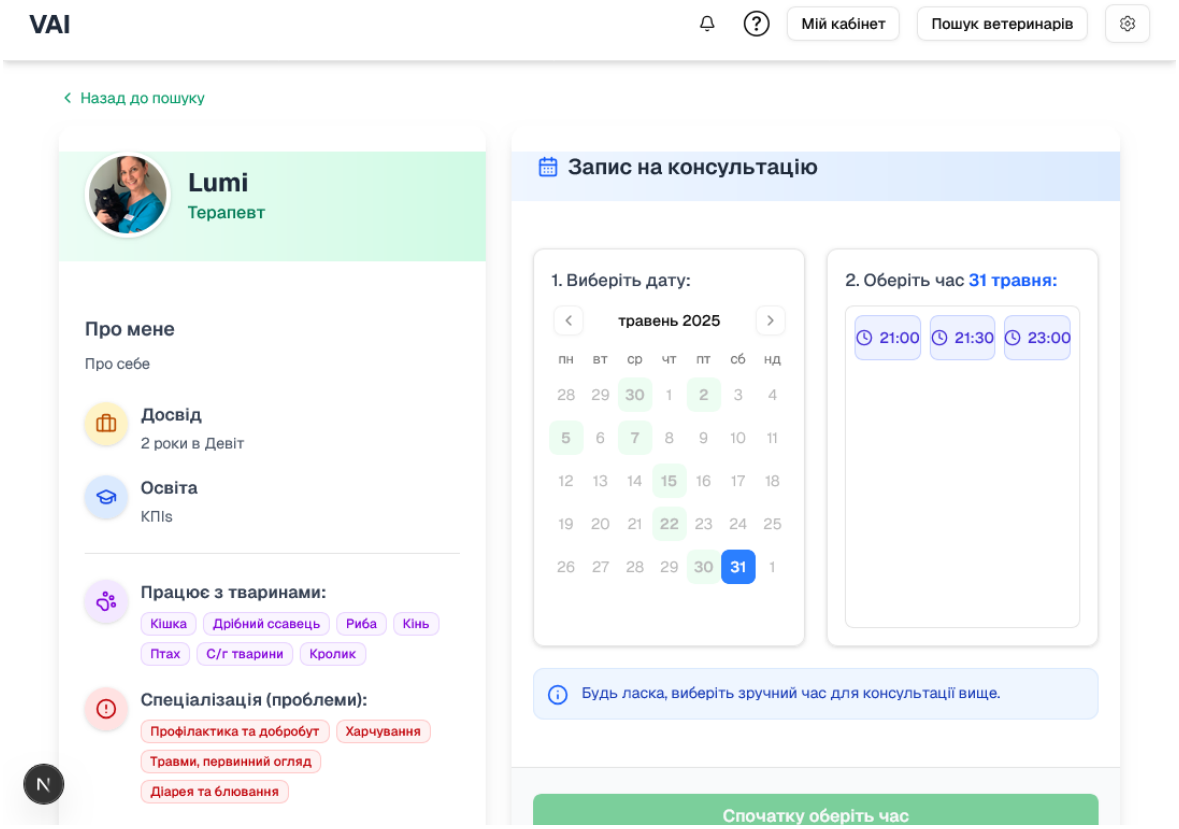
Рисунку 3.13 – Інтерфейс збережених результатів аналізу

На основі результатів аналізу або за власним бажанням, користувач вирішує знайти ветеринара. Він переходить до розділу пошуку, де може застосувати фільтри, наприклад, за спеціалізацією.



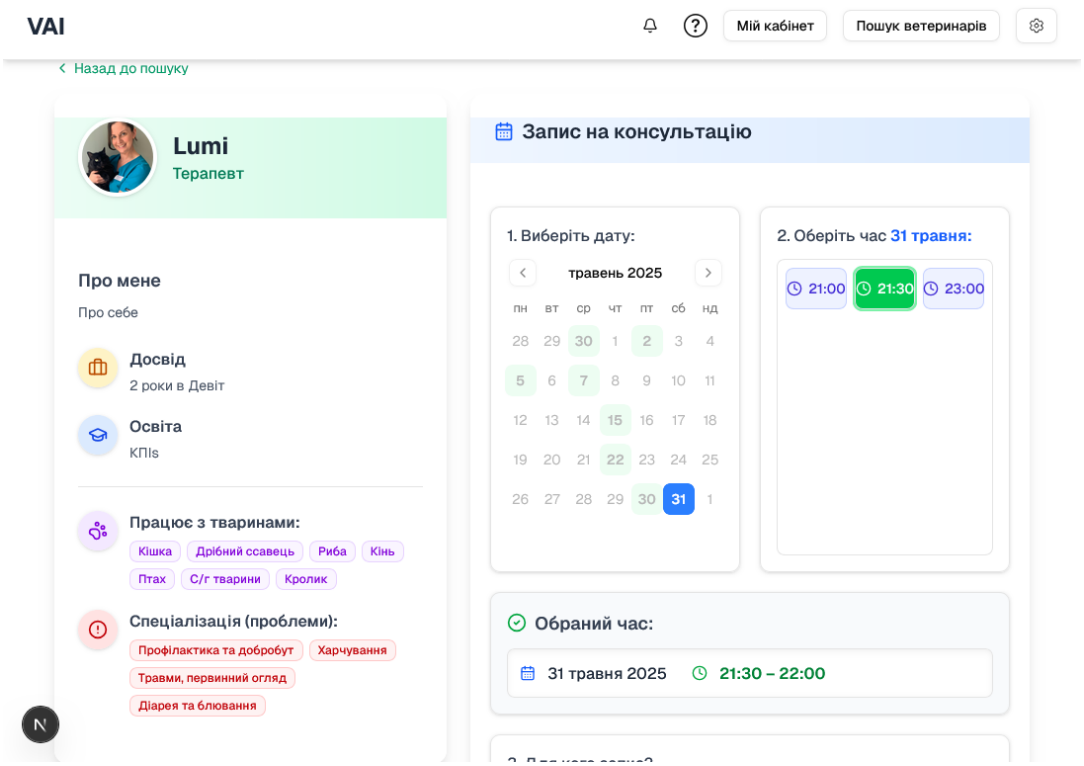
Рисунку 3.14 – Сторінка пошуку ветеринара з використанням фільтрів

Зі списку знайдених фахівців користувач обирає одного та переглядає його детальний профіль, включаючи інформацію про досвід, освіту та доступний розклад для запису.



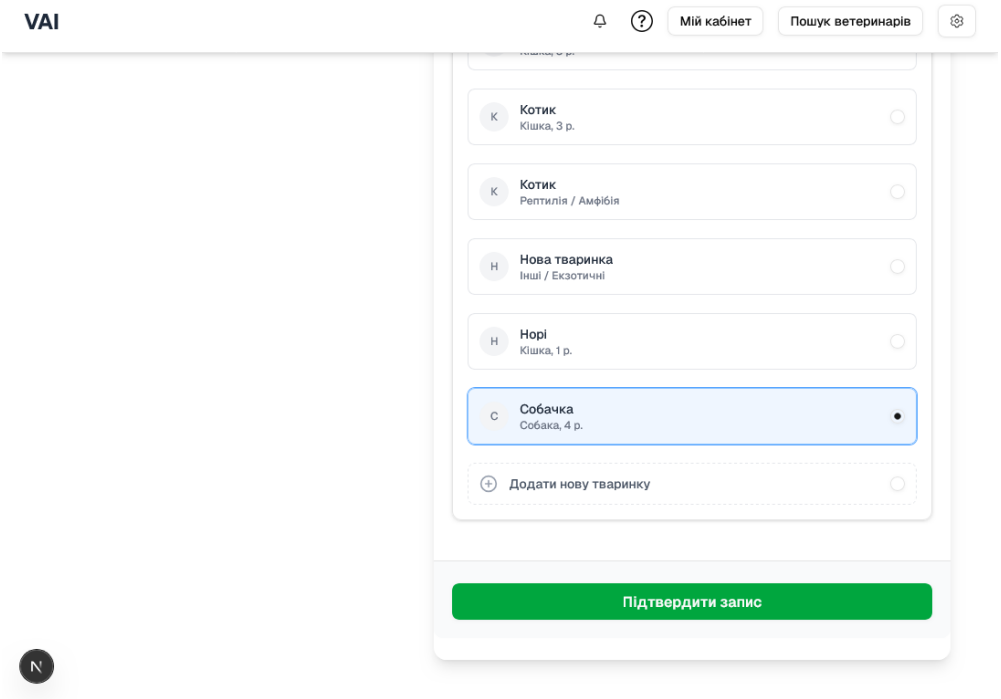
Рисунку 3.15 – Сторінка профілю ветеринара з календарем доступних слотів

На сторінці профілю ветеринара користувач обирає зручну дату та вільний часовий слот на календарі.



Рисунку 3.16 – Вибір дати та часу для запису на консультацію

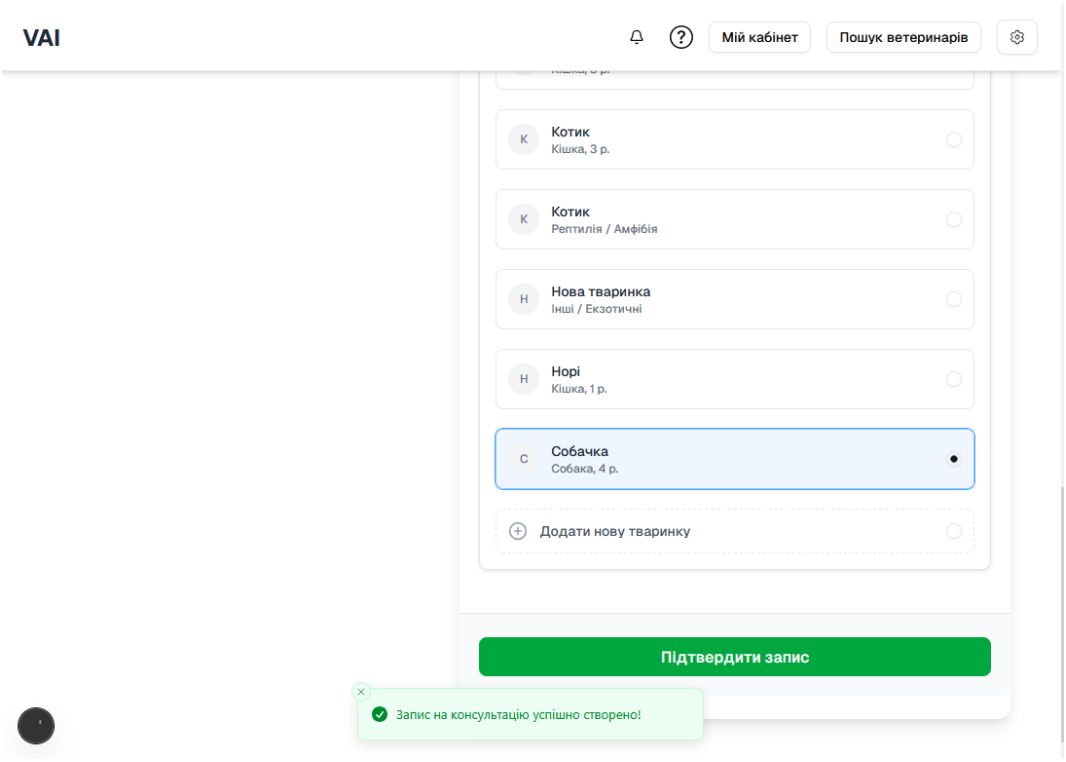
Наступним кроком є вибір тварини, для якої потрібна консультація, зі списку своїх улюбленців.



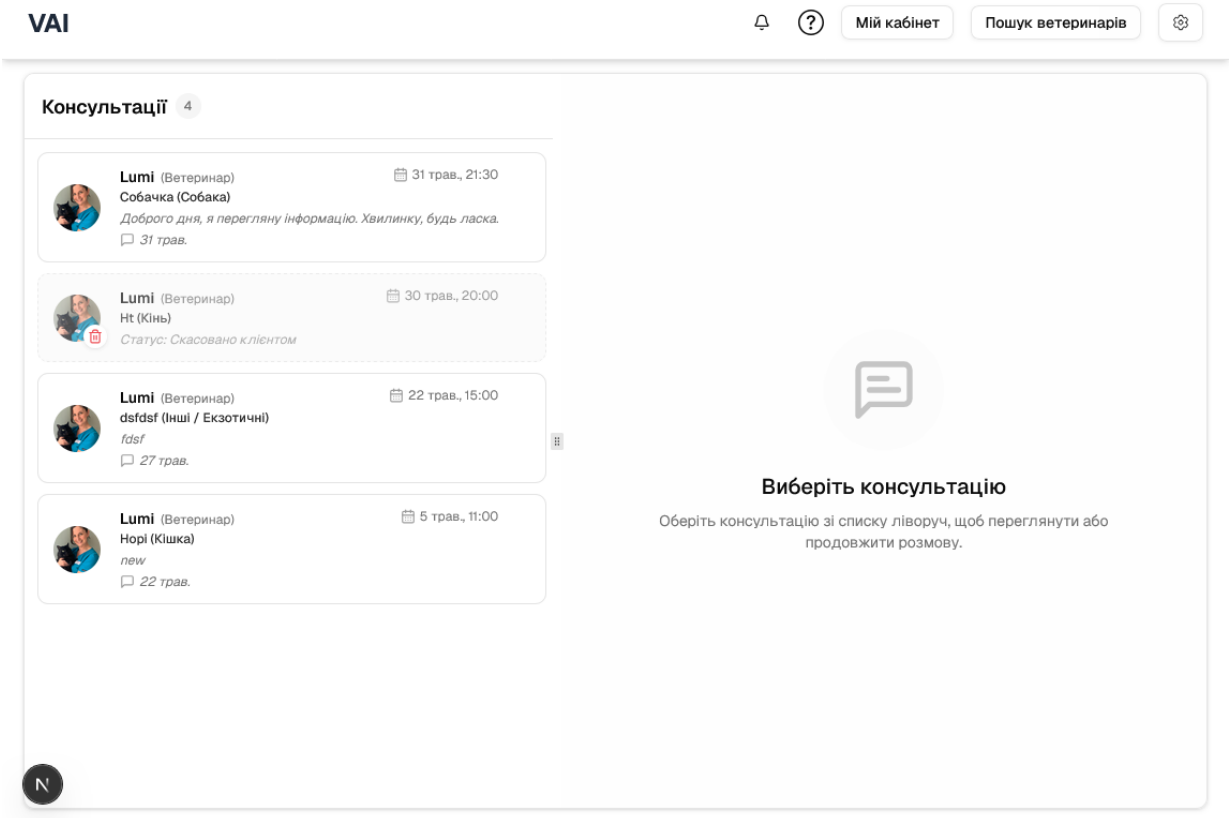
Рисунку 3.17 – Вибір тварини для запису на консультацію

Після підтвердження всіх даних натискається кнопка "Підтвердити запис". У разі успішного бронювання система відображає відповідне

повідомлення, а запланована консультація додається до розкладів обох учасників.

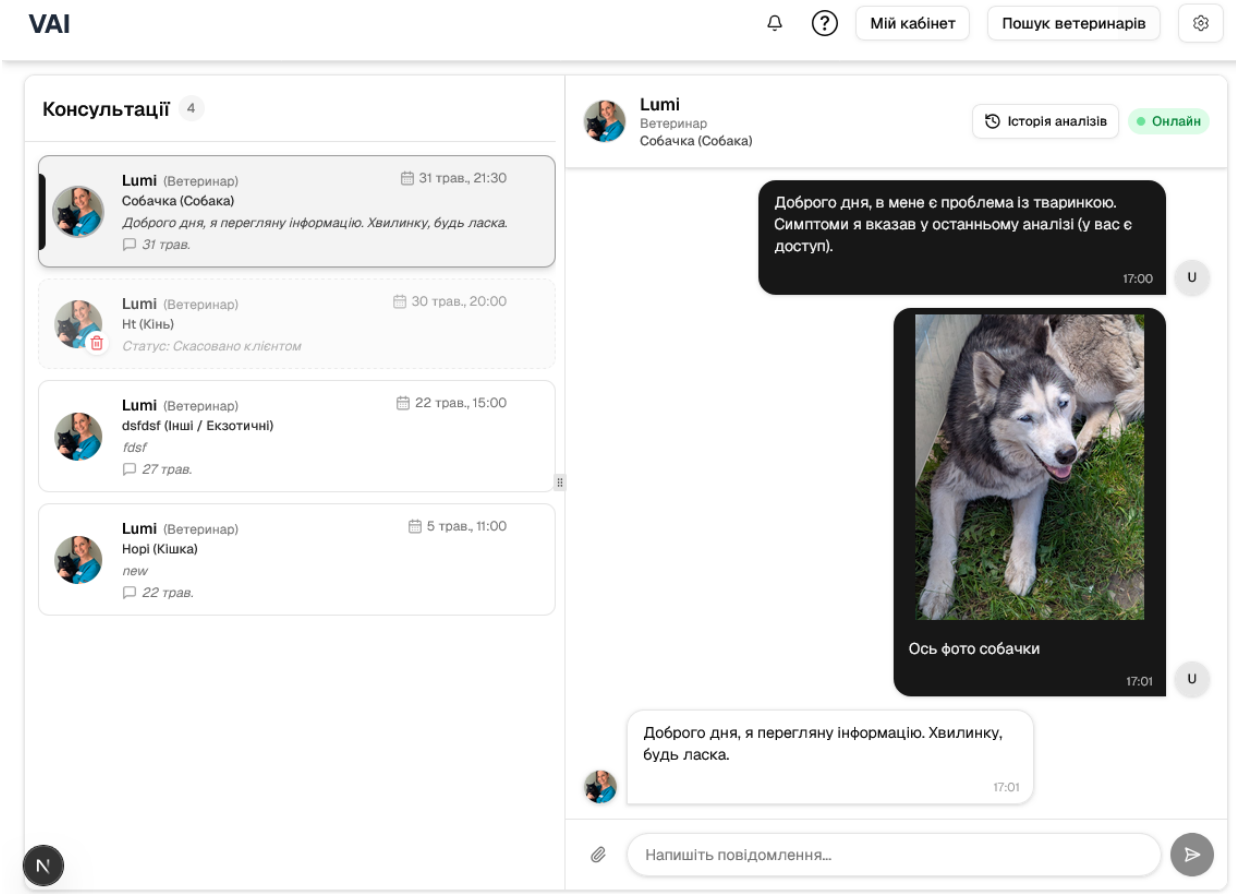


Рисунку 3.18 – Повідомлення про успішне бронювання консультації



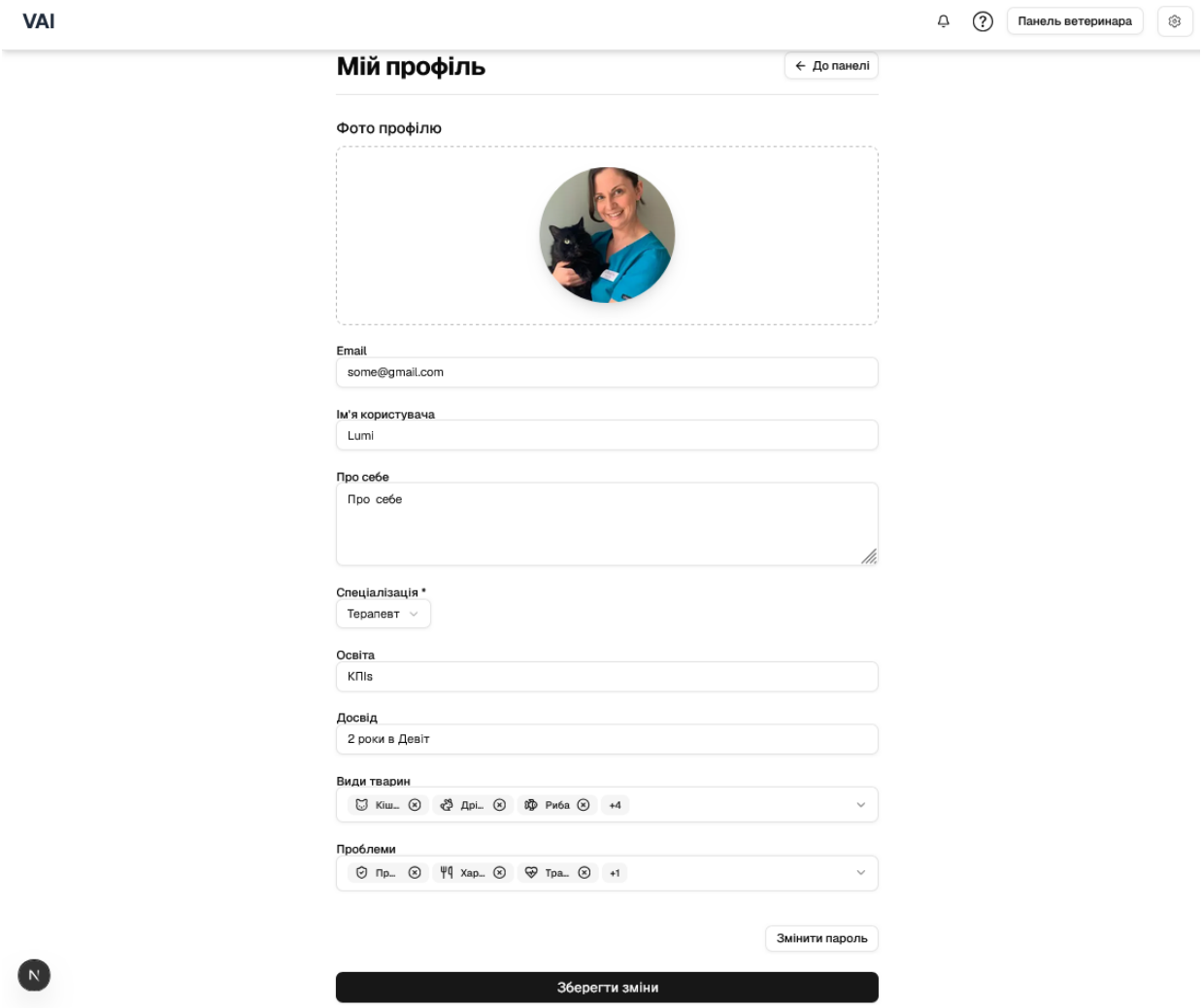
Рисунку 3.19 – Сторінка «Мої консультації» з переліком запланованих консультацій

У призначений час власник тварини та ветеринар входять до системи та відкривають сторінку консультації, де активується чат. Вони можуть обмінюватися текстовими повідомленнями та файлами зображень для детального обговорення стану тварини. Всі повідомлення відображаються в реальному часі та зберігаються в історії.

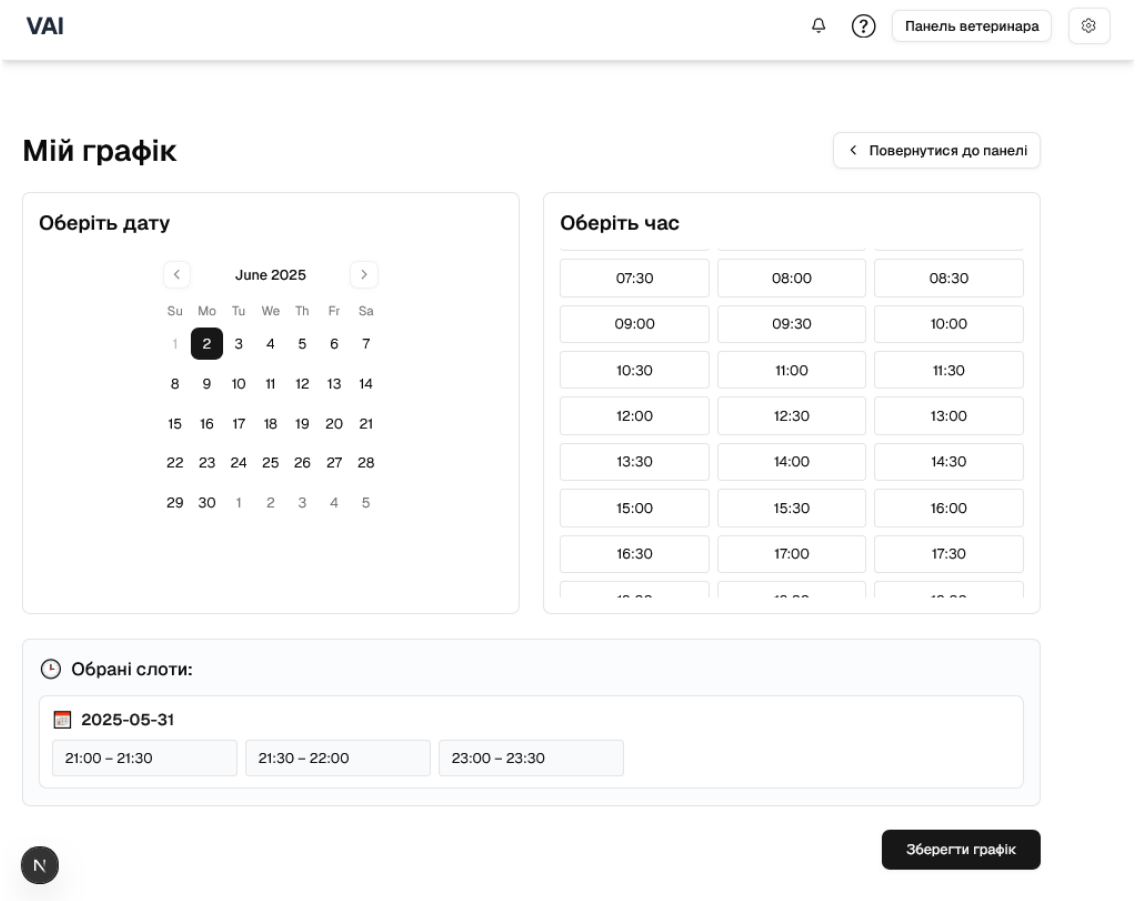


Рисунку 3.20 – Інтерфейс чату онлайн-консультації

Ветеринар у своєму кабінеті може керувати своїм професійним профілем, оновлюючи інформацію про себе та свою спеціалізацію. Також він має доступ до керування своїм розкладом, де може додавати або видаляти доступні часові слоти для консультацій.

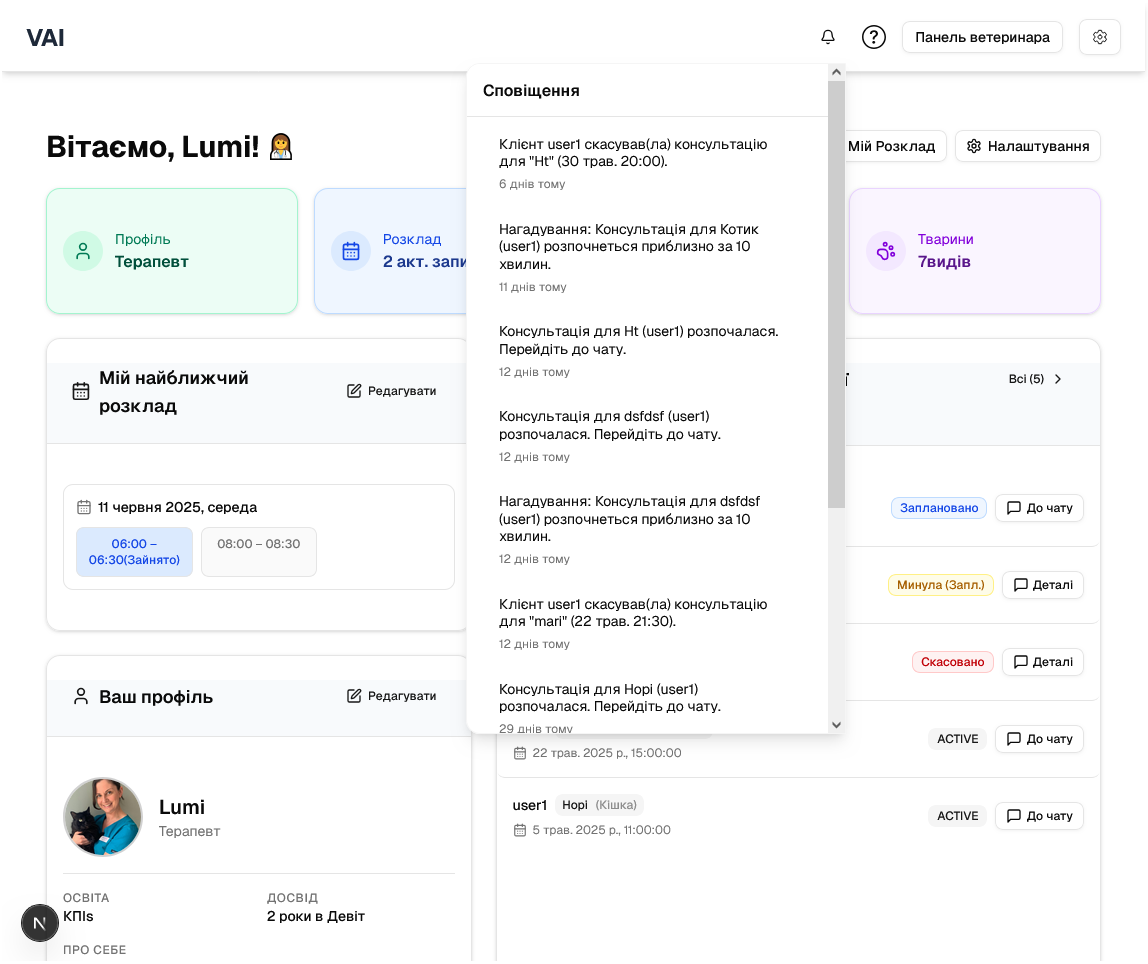


Рисунку 3.21 – Сторінка редагування профілю ветеринара



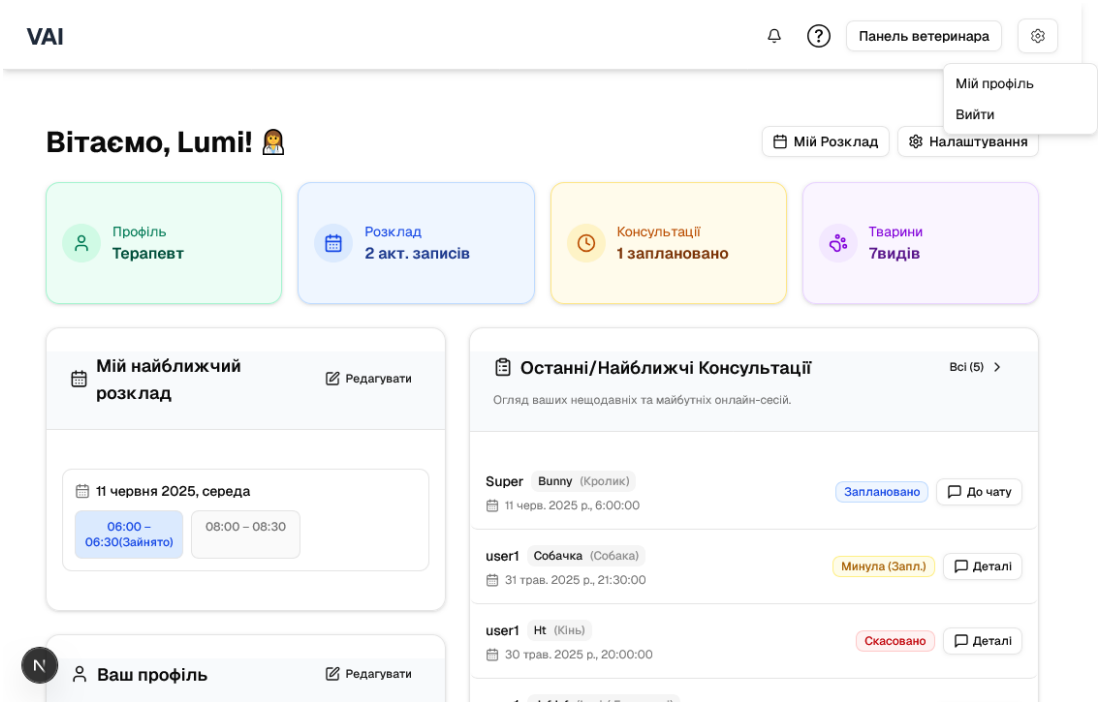
Рисунку 3.22 – Інтерфейс керування розкладом ветеринара

Користувачі (як Власники, так і Ветеринари) можуть переглядати системні сповіщення про важливі події, наприклад, нагадування про консультацію, у спеціальному розділі або за допомогою індикаторів.



Рисунку 3.23 – Розділ сповіщень користувача

Для виходу із системи користувач натискає кнопку «Вийти», що завершує його сесію.



Рисунку 3.24 – Кнопка/меню для виходу із системи

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2025 р.

ВЕБЗАСТОСУНОК ДЛЯ РАННЬОГО ВИЯВЛЕННЯ ХВОРОБ ДОМАШНІХ ТВАРИН ЗА ДОПОМОГОЮ ШІ ТА ІНТЕРАКТИВНОГО КОНСУЛЬТУВАННЯ З ВЕТЕРИНАРОМ

Графічний матеріал

КПІ.ІП-1507.045440.06.99

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Олександр ОЧЕРЕТЯНИЙ

Нормоконтроль:
_____ Тетяна ШУЛЬКЕВИЧ

Виконавець:
_____ Іван ДАЦЬО

Київ – 2025

КПІ.ІП-1507.045440.06.99.ССВ

Вебзастосунок для раннього виявлення хвороб домашніх тварин
за допомогою ШІ та інтерактивного консультування з ветеринаром

Неавторизований Користувач

Реєстрація

Авторизація

Власник Тварини

Пошук Ветеринара

Управління Тваринами

<<include>>

Переглянути список моїх тварин

<<extend>>

Додати нову тваринку

<<extend>>

Редагувати дані тваринки

<<extend>>

Видалити тваринку

Перегляд Профілю Ветеринара

Запис на Онлайн-Консультацію

<<include>>

Переглянути розклад ветеринара

<<include>>

Обрати тваринку для консультації

Аналіз Симптомів за Допомогою ШІ

<<extend>>

Зберегти результати аналізу ШІ

Перегляд Історії Консультацій

Надіслати текстове повідомлення

<<include>>

Вихід з акаунту

Управління Профілем

<<include>>

<<include>>

Зміна пароля

Редагувати особисті дані (email, ім'я користувача)

Авторизований Користувач

Отримання Сповіщень

Скасування Запланованої Консультації

Проведення Онлайн-Консультації (Чат)

<<include>>

<<include>>

Надіслати файл (зображення)

Отримати повідомлення

Управління Профілем Ветеринара

<<extend>>

Редагувати особисті дані

<<extend>>

Редагувати професійні дані ветеринара

Ветеринар

Управління Розкладом

<<extend>>

Створити/додати доступні слоти

<<extend>>

Редагувати/ видалити слоти

<<include>>

Перегляд Списку Запланованих/ Минулих Консультацій

Переглянути свій розклад та записи

КПІ.ІП-1507.045440.06.99.ССВ

Зм.	Арк.	№ документа	Підпис	Дата
Розробив		Дацьо І.І.		
Перевірив		Очеретяний О.К.		
Т. контр.				
Н. контр.		Шулькевич Т.В.		
Затвердив		Жаріков Е.В.		

Схема структурна варіантів використань

Літера	Маса	Масштаб

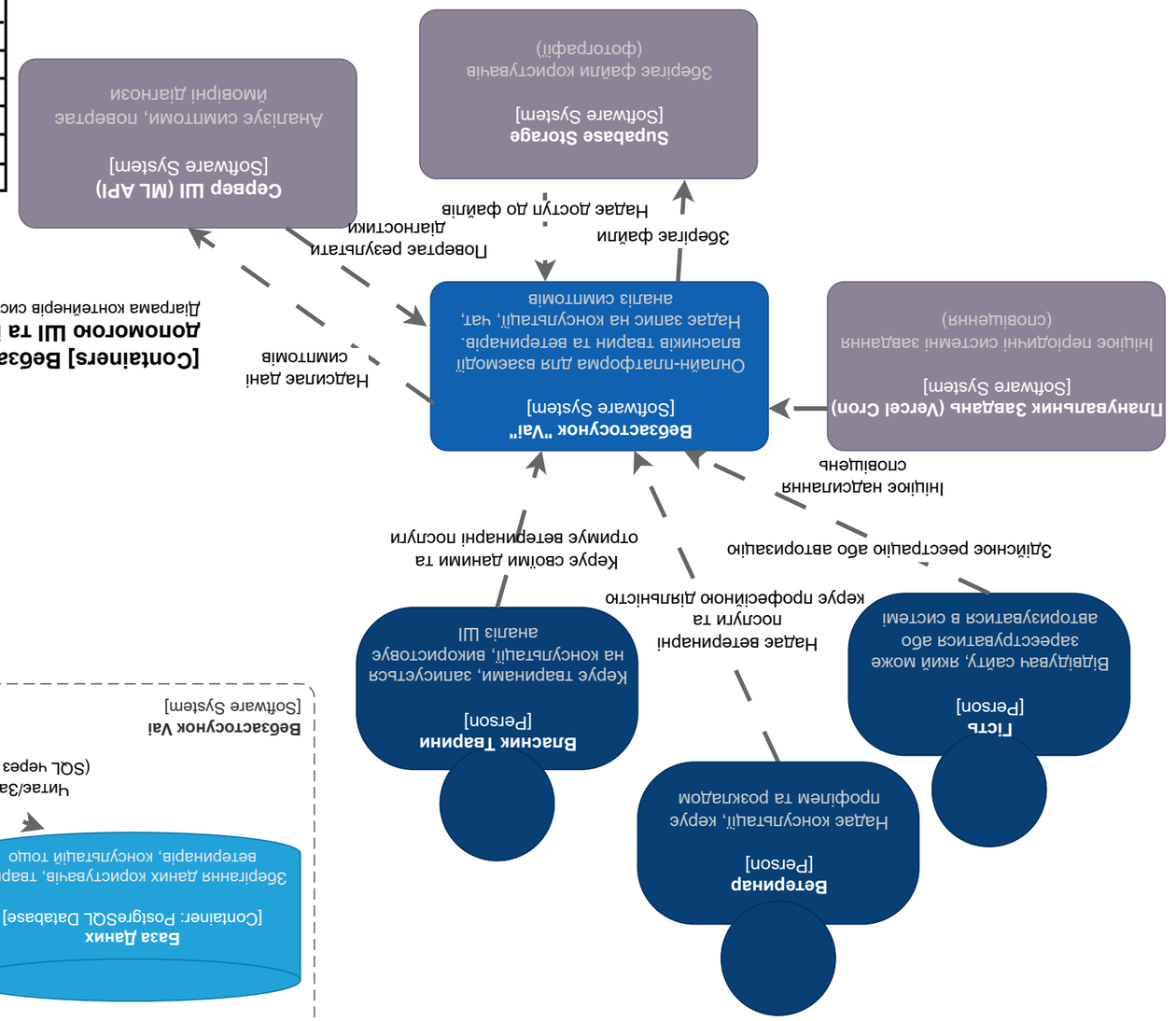
Аркуш 1	Аркушів 1

Вебзастосунок для раннього виявлення хвороб домашніх тварин за допомогою ШІ та інтерактивного консультування з ветеринаром

КПІ ім.Ігоря Сікорського
Кафедра ІПІ
гр. ІП-15

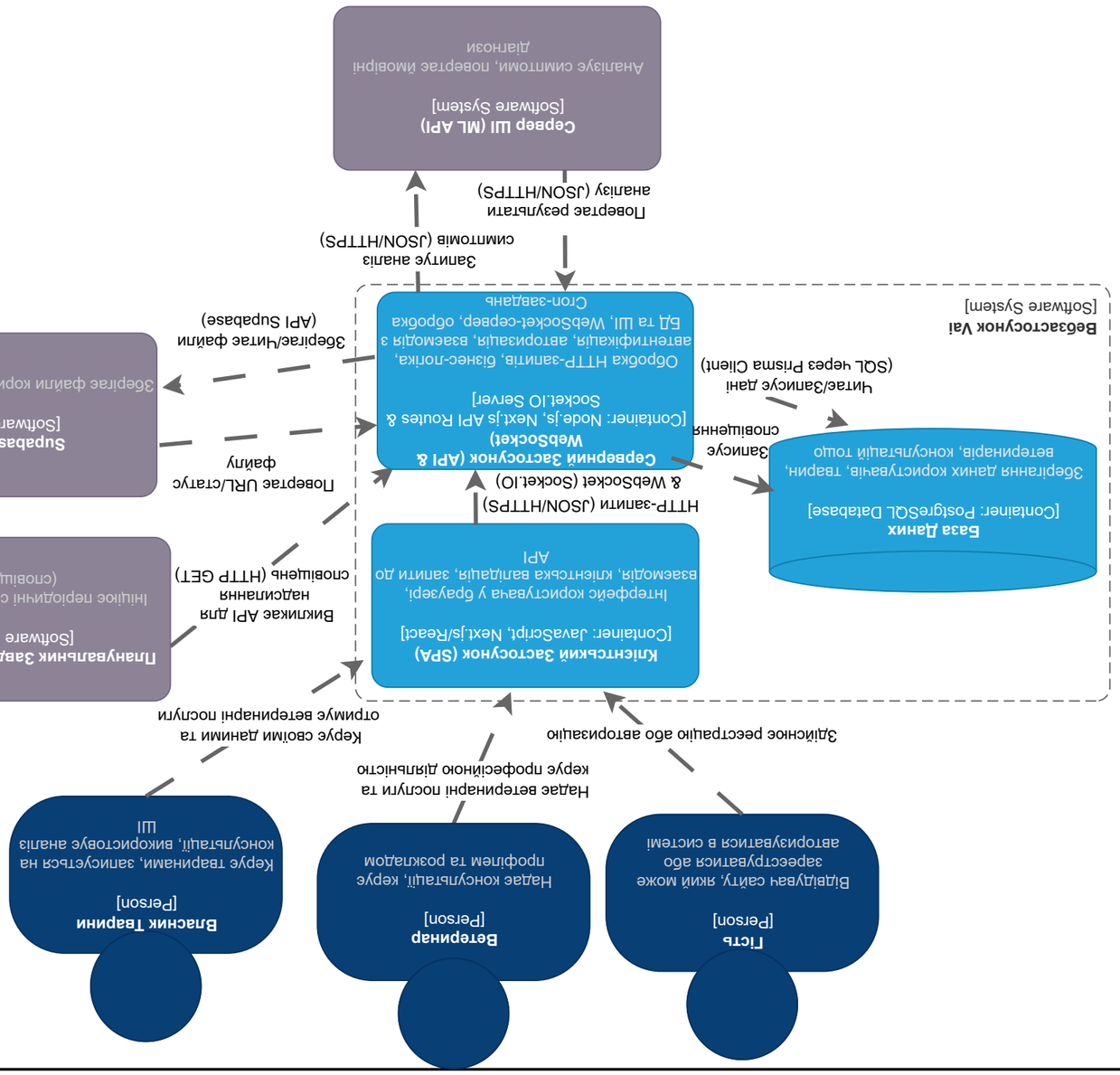
[System Context] Вебзастосунок для раннього виявлення хворób домашніх тварин за допомогою ШІ та інтерактивного консультування з ветеринаром

Діаграма контексту системи

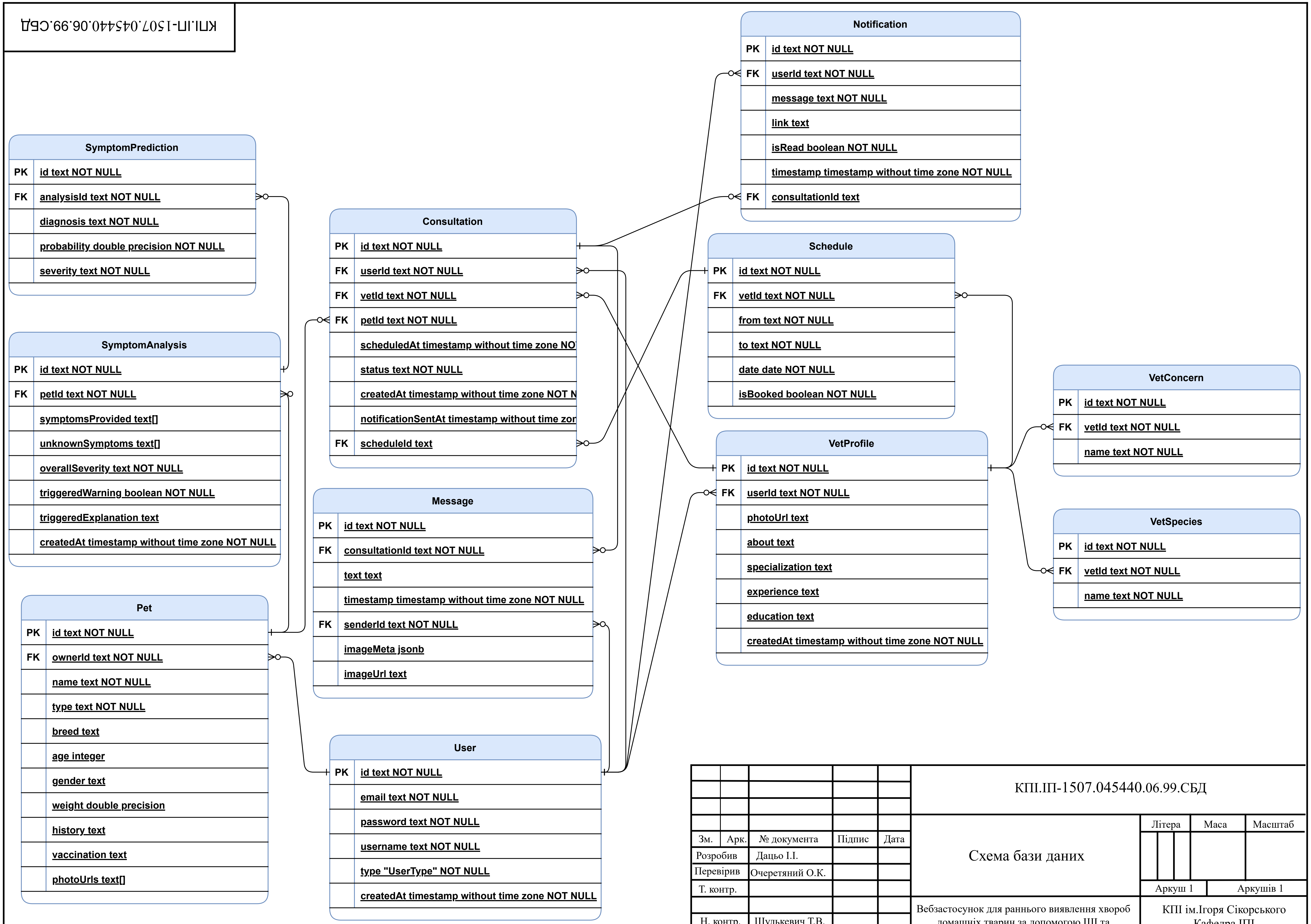


[Containers] Вебзастосунок для раннього виявлення хворób домашніх тварин за допомогою ШІ та інтерактивного консультування з ветеринаром

Діаграма контейнерів системи

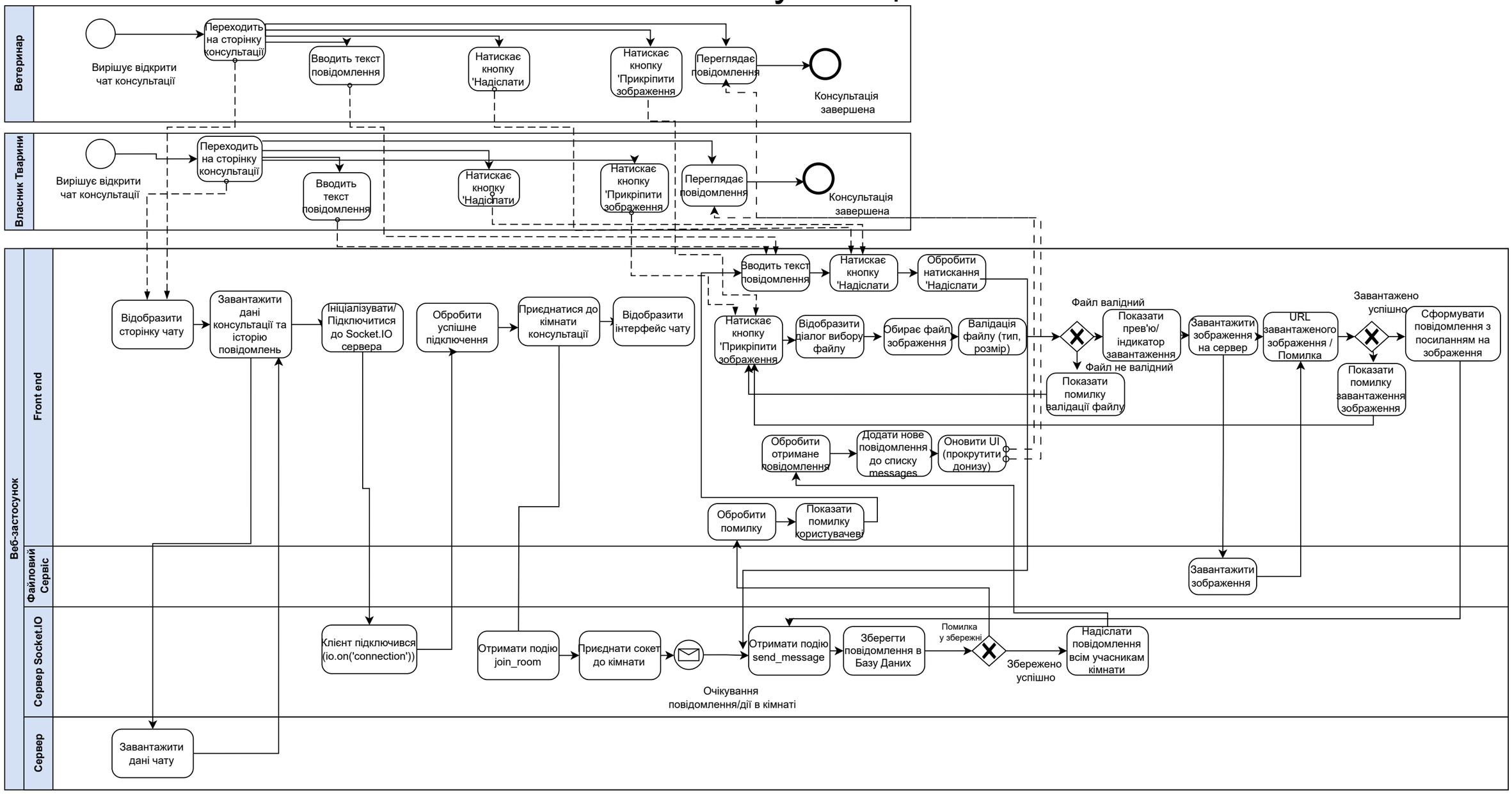


								КПІ.ІП-1507.045440.06.99.ССМ	
Зм.	Арк.	№ документа	Підпис	Дата	Схема структура компонентів програмного забезпечення			Літера	Маса	Масштаб
Розробив		Дашьо І.І.								
Перевірив		Очеретний О.К.								
Т. контр.						Аркуш 1	Аркушів 1			
Н. контр.		Шупикевич Т.В.			Вебзастосунок для раннього виявлення хворób домашніх тварин за допомогою ШІ та інтерактивного консультування з ветеринаром			КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІП-15		
Затвердив		Жаріков Е.В.								



					КПІ.ІП-1507.045440.06.99.СБД		
Зм.	Арк.	№ документа	Підпис	Дата	Схема бази даних	Літера	Маса	Масштаб
Розробив		Дацьо І.І.					
Перевірив		Очеретяний О.К.				Аркуш 1	Аркушів 1
Т. контр.					Вебзастосунок для раннього виявлення хвороб домашніх тварин за допомогою ШІ та інтерактивного консультування з ветеринаром	КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІП-15	
Н. контр.		Шулькевич Т.В.					
Затвердив		Жаріков Е.В.					

BPMN-діаграма проведення онлайн-консультації



Демонстраційний плакат до дипломного проекту

Вебзастосунок для раннього виявлення хвороб домашніх тварин за допомогою ШІ та інтерактивного консультування з ветеринаром

Виконав студент гр. ІП-15 Дацьо І.І.

Керівник Очеретяний О.К.