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 організації роботи ветеринарної клініки

Виконав студент IV курсу, групи ІП-12
(шифр групи)
Бондарчук Анастасія Олександрівна _____
(прізвище, ім'я, по батькові) (підпис)

Керівник ст. викл., Марченко О. І. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант ст. викл., д-р філософії, Головченко М.М. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент проф.д.т.н. каф. ІСТ Корнага Я.І. _____
(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

Бондарчук Анастасії Олександрівні
(прізвище, ім'я, по батькові)

1. Тема проєкту Вебзастосунок для організації роботи ветеринарної клініки
керівник проєкту Марченко Олена Іванівна, ст. викл.
(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) Загальні положення: основні визначення та терміни; предметна область вебзастосунку ветеринарної клініки; аналіз аналогів; постановка мети та задачі автоматизації.
2) Інформаційне забезпечення: вхідні дані; вихідні дані; структура бази даних та зв'язки між сутностями.
3) Математичне забезпечення: формалізація задачі підбору вільного слоту; розрахунок часу з урахуванням тривалості процедур, графіку роботи, наявних записів; обґрунтування вибору методу.
4) Програмне та технічне забезпечення: використані засоби розробки; вимоги до клієнтської та серверної частин; архітектура застосунку; структура модулів.
5) Технологічний розділ: керівництво користувача (адміністратор, лікар, клієнт); інтерфейси та сценарії взаємодії; методика ручного функціонального тестування; приклади тест-кейсів.

5. Перелік графічного матеріалу

1) Схема структурна варіантів використань
2) Схема структурна компонентів програмного забезпечення
3) Схема бази даних
4) Модель бізнес-процесів
5) Креслення вигляду екранних форм

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	18.03.2025	
3	Постановка та формалізація задачі	20.03.2025	
4	Розробка інформаційного забезпечення	29.03.2025	
5	Алгоритмізація задачі	04.04.2025	
6	Обґрунтування вибору використаних технічних засобів	08.04.2025	
7	Розробка програмного забезпечення	10.05.2025	
8	Налагодження програми	15.05.2025	
9	Виконання графічних документів	18.05.2025	
10	Оформлення пояснювальної записки	30.05.2025	
11	Подання ДП на попередній захист	02.06.2025	
12	Подання ДП рецензенту	10.06.2025	
13	Подання ДП на основний захист	16.06.2025	

Студент _____ Анастасія БОНДАРЧУК
(підпис) (ініціали, прізвище)

Керівник _____ Олена МАРЧЕНКО
(підпис) (ініціали, прізвище)

АНОТАЦІЯ

Пояснювальна записка дипломного проєкту складається з п’яти розділів, містить 32 таблиць, 40 рисунків та 22 джерела – загалом 85 сторінок.

Дипломний проєкт присвячений розробці вебзастосунку для організації роботи ветеринарної клініки.

Мета – створити зручну та функціональну систему для онлайн-запису, ведення електронних карток пацієнтів, управління процедурами, медичними призначеннями та ветаптекою.

У розділі 1 проаналізовано предметну область, оглянуто існуючі рішення, сформульовано вимоги до системи.

Розділ 2 присвячений проєктуванню архітектури вебзастосунку, моделюванню процесів за допомогою UML-діаграм та BPMN.

У розділі 3 наведено реалізацію серверної частини з використанням Node.js, Express та MongoDB Atlas, а також клієнтської частини на React.

Розділ 4 присвячено аналізу якості та тестуванню вебзастосунку. Основну увагу приділено мануальному тестуванню ключового функціоналу.

У розділі 5 писано особливості розгортання програмного забезпечення на сервері, організацію хостингу бази даних, заходи інформаційної безпеки, а також рекомендації щодо подальшого розвитку системи.

Програмне забезпечення впроваджено у тестовому середовищі з підтримкою основного робочого процесу ветеринарної клініки.

КЛЮЧОВІ СЛОВА: ВЕТЕРИНАРНА КЛІНІКА, ВЕБЗАСТОСУНОК, REACT, NODE.JS, MONGODB, ЗАПИС НА ПРИЙОМ, РЕЦЕПТ, CRM, АПТЕКА, МАНУАЛЬНЕ ТЕСТУВАННЯ.

ABSTRACT

The explanatory note of the diploma project consists of five sections, contains 32 tables, 40 figures and 22 sources – in total 85 pages.

The purpose of the diploma project is to develop a web application for organizing the work of a veterinary clinic.

The aim is to create a convenient and functional system for online appointment scheduling, maintaining electronic patient records, managing procedures, medical prescriptions, and the veterinary pharmacy.

Section 1 analyzes the subject area, reviews existing solutions, and formulates the system requirements.

Section 2 is devoted to designing the architecture of the web application and modeling processes using UML diagrams and BPMN.

Section 3 presents the implementation of the server side using Node.js, Express, and MongoDB Atlas, as well as the client side using React.

Section 4 is devoted to the analysis of quality and testing of the web application. The main focus is on manual testing of key functionality.

Section 5 describes the features of software deployment on the server, database hosting organization, information security measures, and recommendations for further system development.

The software was implemented in a test environment with support for the main workflow of the veterinary clinic.

KEYWORDS: VETERINARY CLINIC, WEB APPLICATION, REACT, NODE.JS, MONGODB, APPOINTMENT SCHEDULING, PRESCRIPTION, CRM, PHARMACY, MANUAL TESTING.

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	КПІ.ІП-1204.045440.01.91	Технічне завдання	27	
3	А4	КПІ.ІП-1204.045440.02.81	Пояснювальна записка	85	
4	А4	КПІ.ІП-1204.045440.03.12	Текст програми	50	
5	А4	КПІ.ІП-1204.045440.04.51	Програма та методика тестування	6	
6	А4	КПІ.ІП-1204.045440.05.34	Керівництво користувача	33	
7	А3	КПІ.ІП-1204.045440.06.99	Графічний матеріал	4	

Змін.	Арк.	№ докум.	Підп.	Дата	КПІ.ІП-1204.045440.00.90		
Розроб.		Бондарчук А.О.			Відомість дипломного проєкту	Літ.	Аркуш 1 / Аркушів 1
Перевір.		Марченко О.І.					
Н.контр.		Головченко М.М.				КПІ ім. Ігоря Сікорського ФІОТ каф. ІПІ гр. ІП-12	
Затв.		Жаріков Е.В.					

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”

Завідувач кафедри

_____ Едуард ЖАРІКОВ

“___” _____ 2025 р.

ВЕБЗАСТОСУНОК ДЛЯ ОРГАНІЗАЦІЇ РОБОТИ ВЕТЕРИНАРНОЇ КЛІНІКИ

Технічне завдання

КПІ.ІП-1204.045440.01.91

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Олена МАРЧЕНКО

Нормоконтроль:
_____ Максим ГОЛОВЧЕНКО

Виконавець:
_____ Анастасія БОНДАРЧУК

Київ – 2025

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: Вебзастосунок для організації роботи ветеринарної клініки.

Галузь застосування:

Наведене технічне завдання поширюється на розробку веб-застосунку для організації роботи ветеринарної клініки, котра використовується для автоматизації процесів, пов’язаних із документообігом, керуванням розкладами, записами тощо, які мають місце під час роботи ветеринарної клініки, та призначена для використання у ветеринарній сфері, а зокрема у вет-лікарнях для надання можливості електронного запису для клієнтів, створення та ведення електронних карток пацієнтів, редагування розкладу лікарів для адміністратора клініки.

2 ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки веб-застосунку для організації роботи ветеринарної клініки 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 автоматизації організаційних аспектів роботи ветеринарної клініки.

Метою розробки є підвищення якості та ефективності надання ветеринарних послуг шляхом автоматизації організаційних процесів клініки, забезпечення зручного управління записами, розкладом лікарів, базою пацієнтів, швидкого доступу до повної історії хвороб, призначень і результатів обстежень.

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вимоги до функціональних характеристик

Програмне забезпечення повинно забезпечувати виконання наступних основних функцій:

4.1.1 Користувацького інтерфейсу

– функція авторизації (рис. 4.1) та реєстрації (рис. 4.2): введення email (та номеру телефону), отримання коду підтвердження, вхід у систему;

Рисунок 4.1 – Авторизація користувача

Рисунок 4.2 – Реєстрація користувача

– функція перегляду та редагування особистого профілю користувача (рис. 4.3) та даних акаунту для входу (рис. 4.4);

Рисунок 4.3 – Редагування даних користувача

Рисунок 4.4 – Редагування даних акаунту для входу

– функція управління картками тварин: створення (рис. 4.7 та рис. 4.8), редагування (рис. 4.6 та рис.), перегляд даних тварини (рис. 4.5, рис. 4.8 та рис. 4.9), архівація/відновлення карток (у адміністраторів та лікарів) (рис. 4.10);

Рисунок 4.5 – Перегляд списку карток тварин клієнта

Рисунок 4.6 – Редагування даних про тварину клієнта

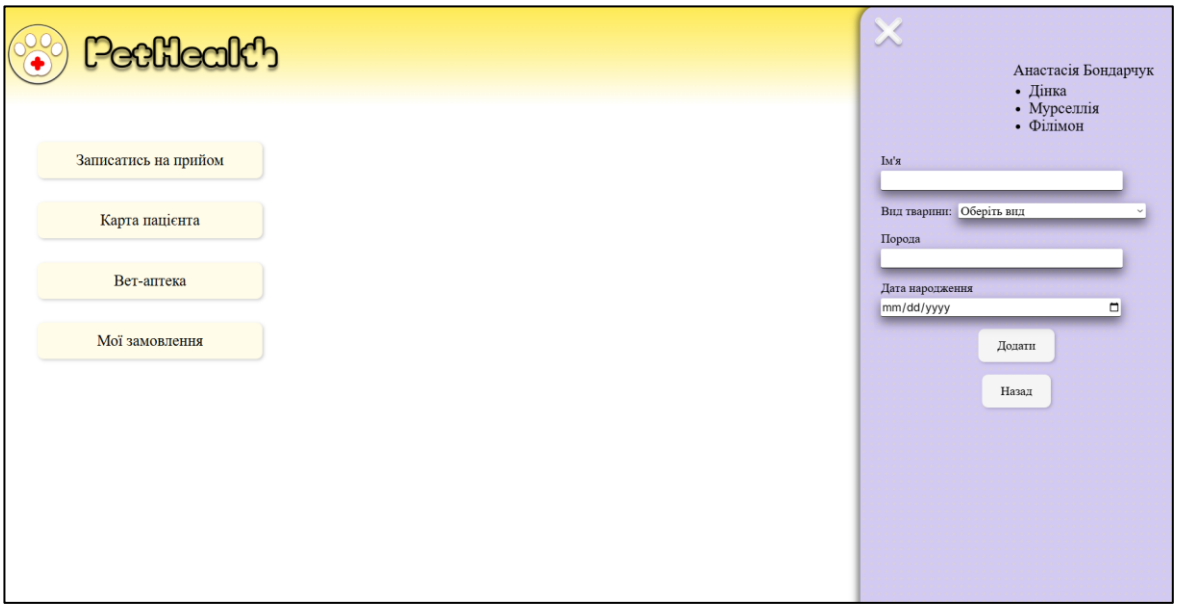


Рисунок 4.7 – Додавання нової картки тварини клієнта

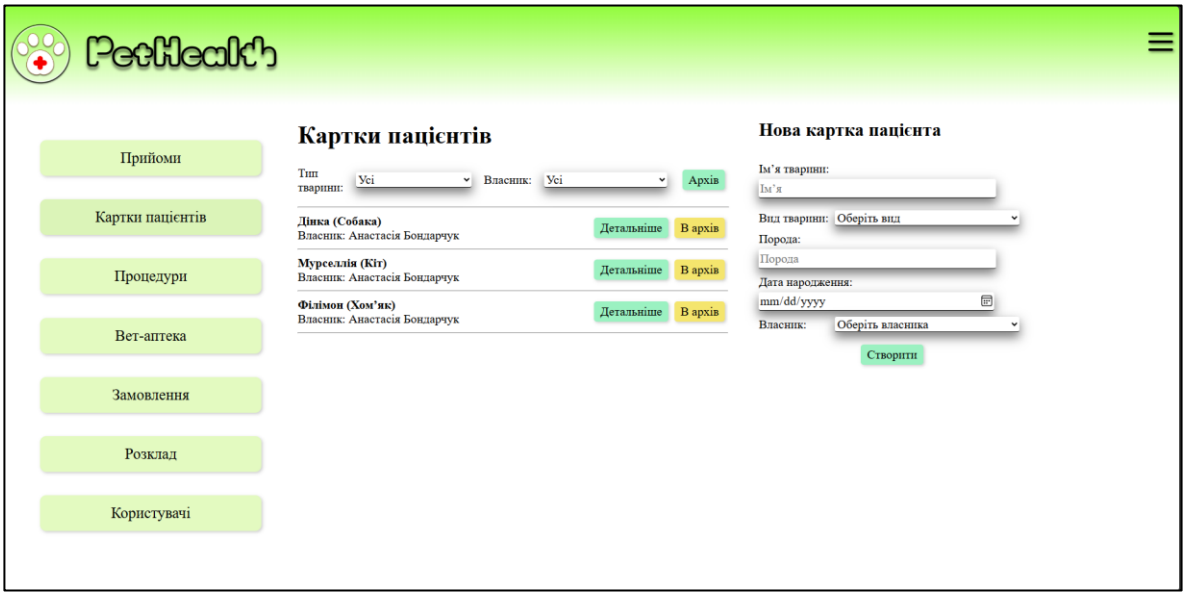


Рисунок 4.8 – Перегляд списку карток пацієнтів, створення нової картки

Рисунок 4.9 – Перегляд детальної інформації в картці пацієнта

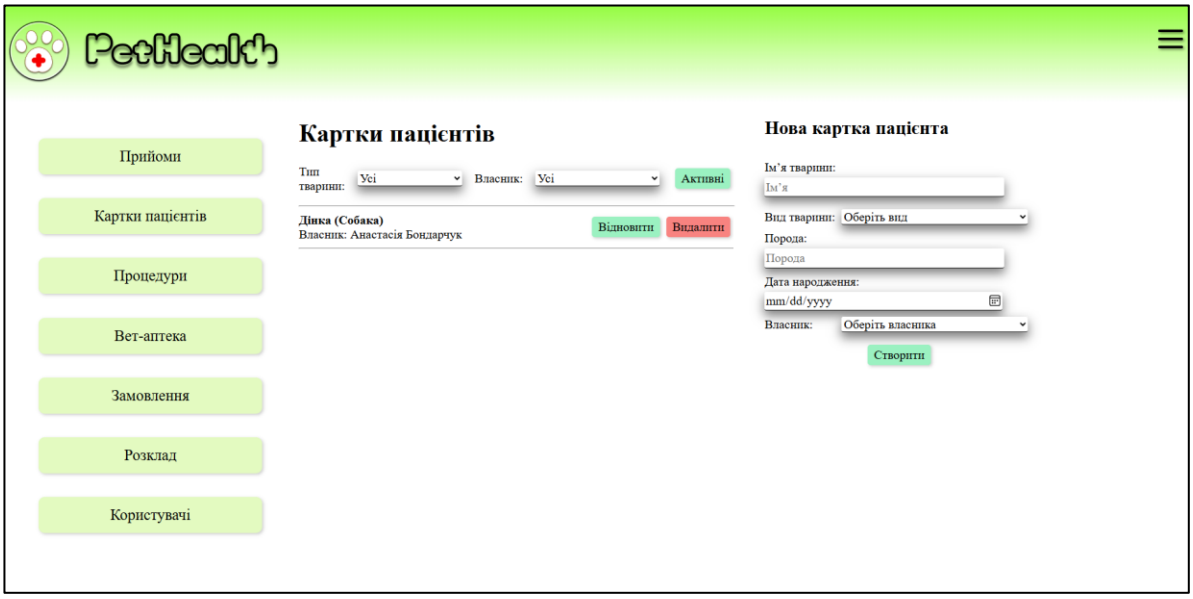


Рисунок 4.10 – Перегляд архівованих карток, відновлення або видалення архівованої картки пацієнта

– функція перегляду інформації про прийоми та записи: вивід списку прийомів та записів (рис. 4.11, рис. 4.12, рис. 4.14, рис. 4.15), доступ до детальної інформації (рис. 4.13, рис. 4.16);

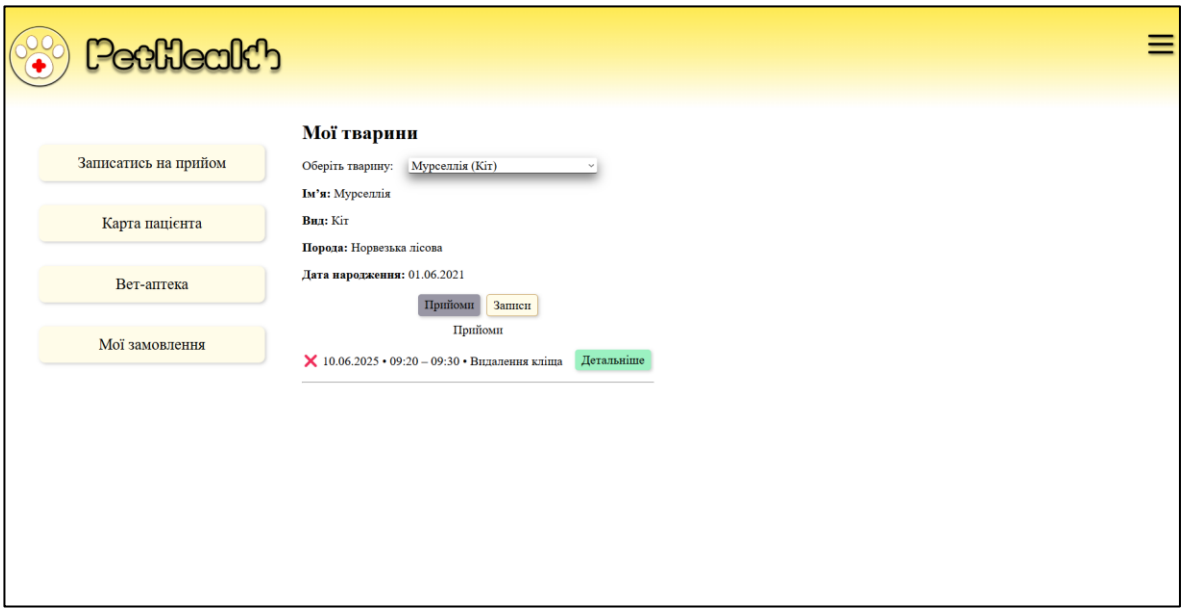


Рисунок 4.11 – Перегляд прийомів в картці пацієнта

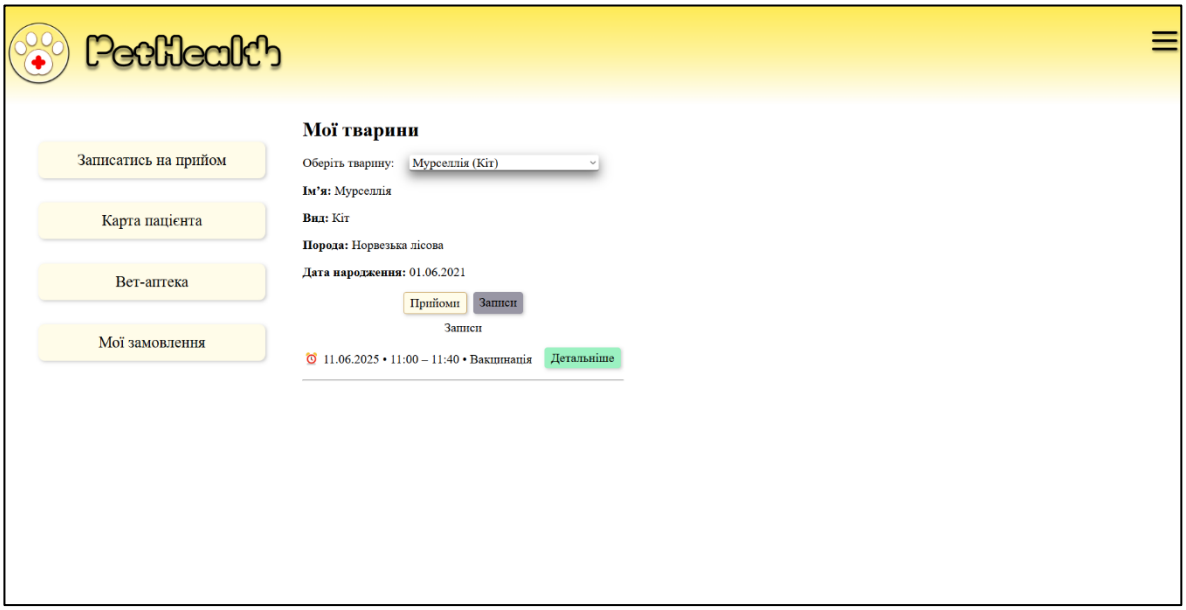


Рисунок 4.12 – Перегляд записів в картці пацієнта

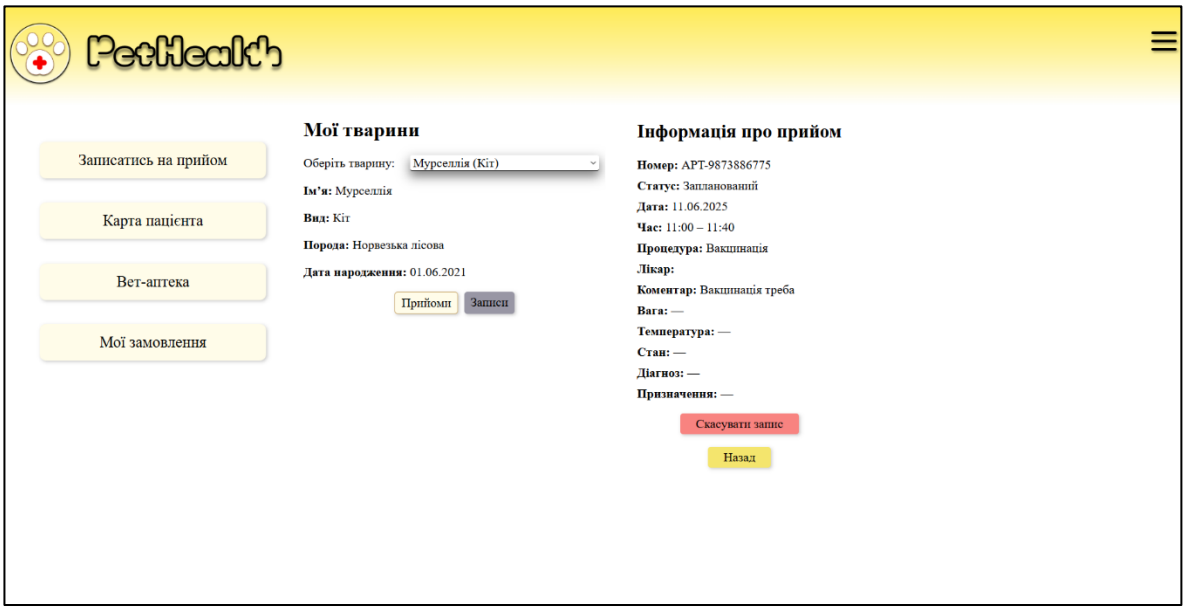


Рисунок 4.13 – Перегляд детальної інформації про прийом/запис

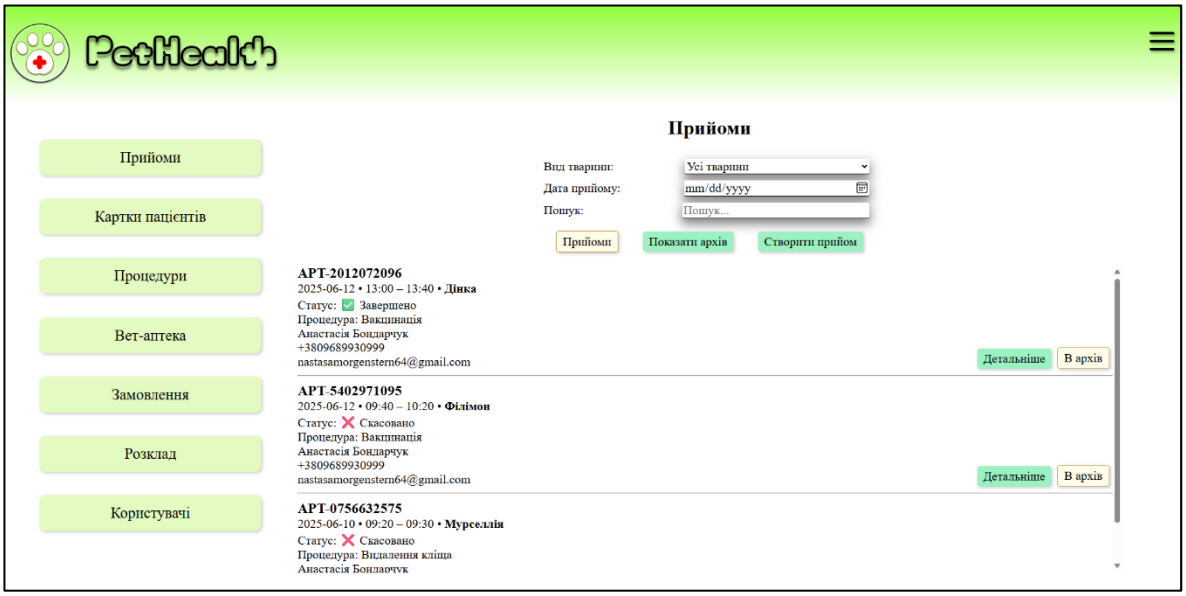


Рисунок 4.14 – Перелік прийомів з фільтрами та пошуком

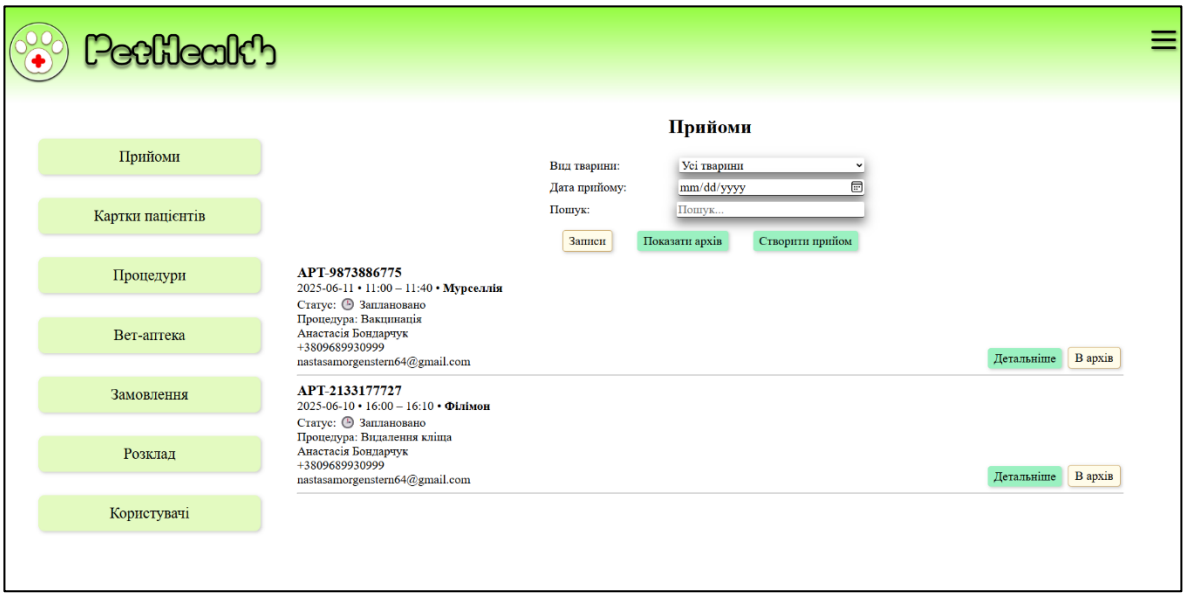


Рисунок 4.15 – Перелік записів з фільтрами та пошуком

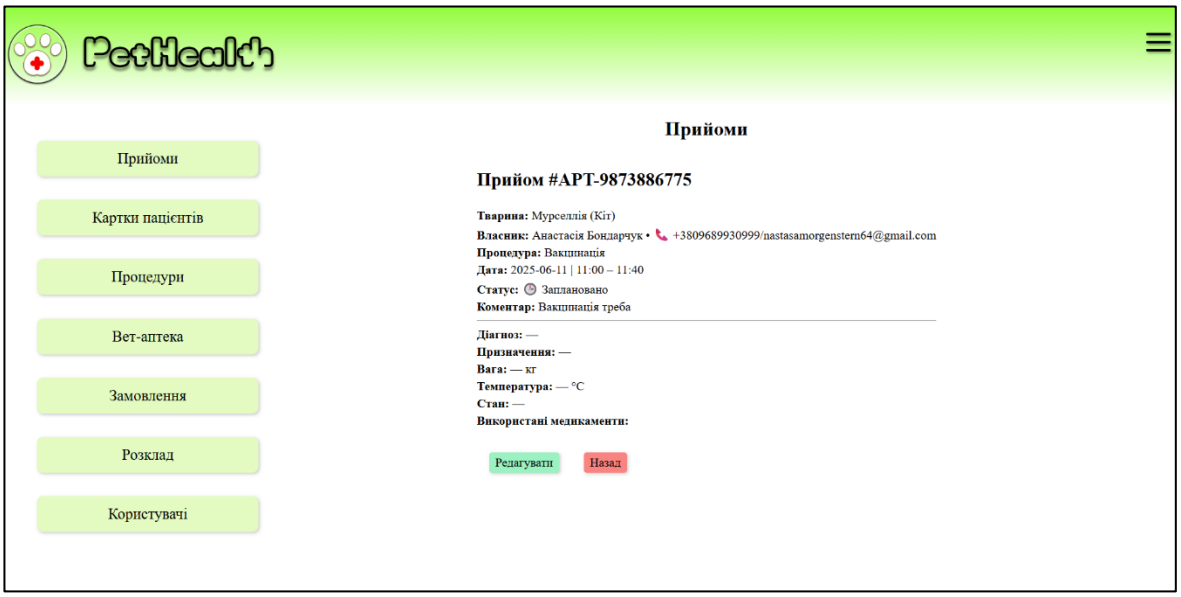


Рисунок 4.16 – Перегляд детальної інформації про запис/прийом

– функція запису на прийом (для клієнта): вибір тварини, процедури, дати, перегляд доступних слотів, підтвердження запису;

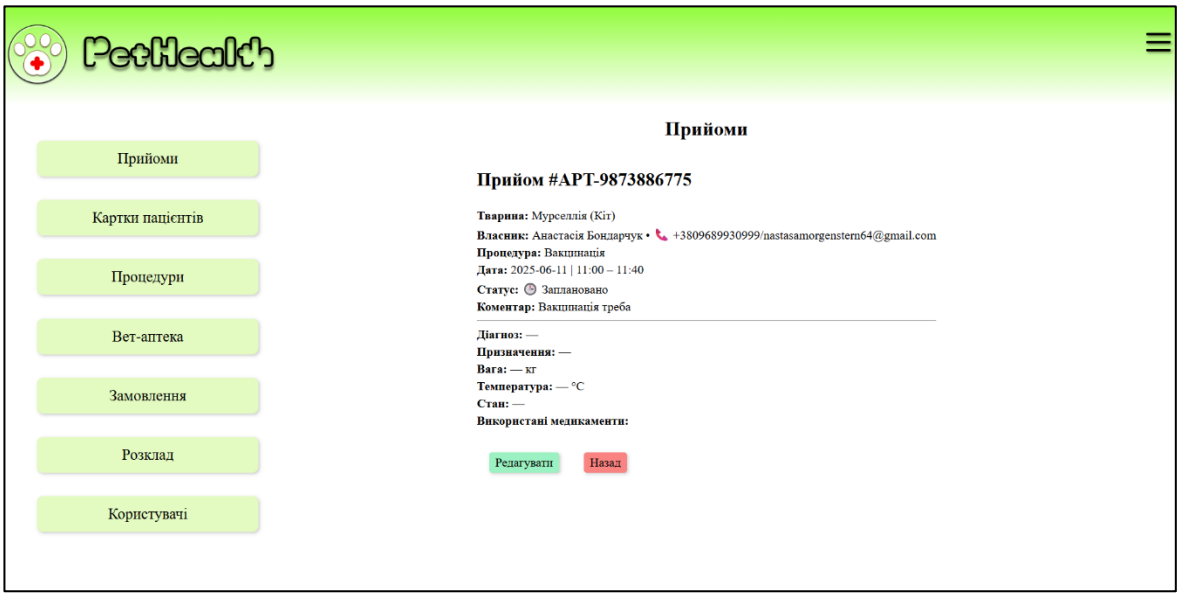


Рисунок 4.17 – Перегляд детальної інформації про запис/прийом

– функція запису на прийом (для адміністратора/лікаря): вибір тварини, процедури, дати, перегляд доступних слотів, підтвердження запису;

– функція проведення прийому (для лікаря/адміністратора): заповнення медичних полів (діагноз, вага, температура, стан, призначення, використані медикаменти, статус прийому, рецепт);

– функція створення та використання рецептів: вибір медикаментів, формування рецепта, прив’язка до прийому, подальше використання у замовленнях клієнтом;

– функція перегляду онлайн-аптеки: фільтрація товарів за категоріями, видами тварин, пошук, додавання в кошик;

– функція оформлення замовлення: вибір рецепта на товари в кошику (якщо потрібно), способу доставки (самовивіз, відділення, поштомат, адреса) та способу оплати;

– функція адміністрування: вкладки для керування користувачами, пацієнтами, прийомами, процедурами, товарами, графіком клініки та категоріями;

– функція повідомлень і сповіщень: використання вбудованих повідомлень браузера про успішні або помилкові дії, email-повідомлення про запис, скасування, нагадування.

4.1.2 Реєстрація та авторизація користувачів:

4.1.2.1 Для користувача:

– функція реєстрації з підтвердженням email;

– функція входу до системи через email-код;

– функція відновлення доступу по завершенні часу активної сесії.

4.1.2.2 Для адміністратора системи:

– функція переглядау всіх користувачів системи;

– функція зміни ролей користувачів;

– функції архівації, відновлення та видалення користувачів.

4.1.2.3 Додаткові вимоги:

– безпека обробки облікових даних (JWT, валідація); 1;

– унікальність email та номеру телефону.

4.1.3 Облік користувачів:

4.1.3.1 Для користувача:

– функція перегляду та редагування свого профілю;

– функція зміни телефону за підтвердженням на пошту та даних клієнта;

– функція перегляду пов'язаних карток тварин.

4.1.3.2 Для адміністратора системи:

– функція створення користувача вручну;

– функція редагування даних користувачів;

– функція перегляду контактної інформації користувачів.

4.1.3.3 Додаткові вимоги:

– функція підтвердження змін телефону та допуск до зміни email лише у адміна.

4.1.4 Управління картками пацієнтів:

4.1.4.1 Для користувача:

– функція створення картки для своєї тварини/тварин;

– функція перегляду основної інформації про тварину/тварин;

– функція редагування власної картки тварини/тварин.

4.1.4.2 Для адміністратора системи:

– функція створення користувача вручну;

– функція редагування картки пацієнта;

– функція архівації або видалення картки тварини.

4.1.5 Запис на прийом:

4.1.5.1 Для користувача:

– функція вибору тварини, процедури, дати та часу прийому;

– функція підтвердження запису з можливістю додавання коментаря;

– функція скасування запису, якщо до початку прийому залишається більше 12 годин.

4.1.5.2 Для адміністратора/лікаря системи:

– функція створення запису для клієнта вручну;

– функція перегляду, фільтрації та пошуку прийомів за різними параметрами.

4.1.5.3 Додаткові вимоги:

– функція перевірки на конфлікти при створенні запису;

– функція автоматичного надсилання нагадувань про запис за день і за годину до часу прийому.

4.1.6 Ведення прийому та облік процедур:

4.1.6.1 Для лікаря:

– функція зміни статусу прийому;

– функція внесення медичних даних: діагноз, температура, вага, стан;

– функція додавання використаних медикаментів, якщо їх вартість не закладена у вартість процедури;

– функція створення рецепта для рецептурних препаратів.

4.1.6.2 Для адміністратора системи:

– функція перегляду та редагування всіх прийомів;

– функція зміни статусу прийомів та архівація завершених записів.

4.1.6.3 Додаткові вимоги:

– функція обліку рецептурних медикаментів через окрему сутність «Рецепт»;

– функція автоматичного підрахунку загальної вартості прийому.

4.1.7 Ветаптека:

4.1.7.1 Для користувача:

– функція перегляду медикаментів із фільтрацією за видом тварини та категорією;

– функція перегляду детальної інформації про товар;

– функція додавання товару до кошика;

– функція оформлення замовлення з вибором доставки та способу оплати.

4.1.7.2 Для адміністратора системи:

– функція створення, редагування, архівації, видалення та відновлення товарів;

– функція позначення товару як рецептурного;

– функція створення та редагування категорій;

– функція автоматичного перенесення товарів у категорію «загальні» при видаленні категорії.

4.1.7.3 Додаткові вимоги:

– функція перевірки наявності рецепта при покупці рецептурного товару;

– функція фільтрації, пошуку та пагінації на стороні користувача.

4.1.8 Замовлення товарів:

4.1.8.1 Для користувача:

– функція перегляду історії власних замовлень;

– функція перегляду деталей замовлення (статус, доставка, товари, номер).

4.1.8.2 Для адміністратора системи:

– функція перегляду всіх замовлень;

– функція зміни статусу замовлень (в обробці, передано в доставку, скасовано, завершено);

– функція редагування способу доставки та оплати.

4.1.8.3 Додаткові вимоги:

– функція автоматичної генерації унікального номера замовлення;

– функція списання залишків товару при оформленні.

4.1.9 Управління графіком клініки:

4.1.9.1 Для користувача:

– функція запису активна лише у дозволені дні та години;

– функція недопущення запису у закриті дні або зайняті слоти.

4.1.9.2 Для адміністратора системи:

– функція встановлення робочих/вихідних днів, годин роботи, обіду, операційних днів та годин;

– функція налаштування графіка по замовчуванню та окремо по кожному дню;

– функція перегляду записів, що потрапляють у закриті дні при зміні графіку.

4.1.9.3 Додаткові вимоги:

– функція автоматичного скасування записів у закриті дні з надсиланням email-сповіщень;

– функція оновлення доступності слотів після збереження графіка.

4.1.10 Управління графіком клініки:

4.1.10.1 Для користувача:

– функція запису активна лише у дозволені дні та години;

– функція недопущення запису у закриті дні або зайняті слоти.

4.1.10.2 Для адміністратора системи:

– функція встановлення робочих/вихідних днів, годин роботи, обіду, операційних днів та годин;

– функція налаштування графіка по замовчуванню та окремо по кожному дню;

– функція перегляду записів, що потрапляють у закриті дні при зміні графіку.

4.1.10.3 Додаткові вимоги:

– функція автоматичного скасування записів у закриті дні з надсиланням email-сповіщень;

– функція оновлення доступності слотів після збереження графіка.

4.2 Вимоги до надійності

Для забезпечення надійності програмного забезпечення передбачено такі механізми:

Контроль введення інформації:

– функція валідації всіх полів форм на клієнтській та серверній стороні (наприклад, email, дата, числові значення);

– функція повідомлень про помилки користувача з підказками;

– функція обмеження доступу до захищених маршрутів на основі ролей (авторизація через токен).

Захист від некоректних дій користувача:

– функція перевірки прав доступу при спробі змінити дані, які користувач не має права змінювати;

– функція недопущення створення конфліктних записів та замовлень;

– функція автоматичної перевірки наявності рецепта перед купівлею рецептурного препарату.

Цілісність інформації в базі даних:

– функція транзакційного збереження записів, які містять кілька залежних сутностей (наприклад, створення рецепта з медикаментами);

– функція використання зовнішніх ключів (через Mongoose ref) для зв’язування документів;

– функція видалення або редагування записів лише з урахуванням цілісності (наприклад, не дозволено видалити лікаря, до якого прив’язані прийоми).

Відновлення після збоїв:

– функція періодичного резервного копіювання бази даних на рівні MongoDB Atlas;

– функція автоматичного перезапуску сервера у випадку помилки;

– функція логування критичних помилок на сервері для подальшого аналізу та усунення.

4.3 Умови експлуатації

Умови експлуатації згідно СанПін 2.2.2.542 – 96.

4.3.1 Вид обслуговування

«Вимоги до виду обслуговування не висуваються»

4.3.2 Обслуговуючий персонал

«Вимоги до обслуговуючого персоналу не висуваються»

4.4 Вимоги до складу і параметрів технічних засобів

Розроблене програмне забезпечення є клієнт-серверним вебзастосунком і повинно функціонувати на IBM-сумісних персональних комп’ютерах з підключенням до мережі Інтернет. Обидві частини – серверна (бекенд) та клієнтська (фронтенд) – можуть бути розгорнуті як локально, так і у хмарному середовищі.

Мінімальна конфігурація технічних засобів:

– тип процесору: Intel Core i3 (4-го покоління і вище) або AMD Athlon / Ryzen 3;

– об’єм ОЗП: 2-4 Гб;

– підключення до мережі Інтернет зі швидкістю від від 10 Мбіт/с;

– веббраузер: будь-який сучасний, що підтримує ES6+ (Chrome, Firefox, Edge, Opera тощо).

Рекомендована конфігурація технічних засобів:

– тип процесору: Intel Core i7 11th Gen;

– об’єм ОЗП: 16 ГБ DDR4;

– підключення до мережі Інтернет зі швидкістю від 100 мегабіт;

– накопичувач: SSD накопичувач об’ємом 500+ ГБ

4.5 Вимоги до інформаційної та програмної сумісності

Розроблене програмне забезпечення має бути кросплатформеним вебзастосунком, який не залежить від операційної системи користувача, оскільки функціонує через сучасний браузер.

Програмне забезпечення повинно працювати під управлінням операційних систем сімейства WIN32 (Windows 7/8/10/11), а також Unix-подібних систем (Linux, macOS), за умови наявності сучасного веббраузера (Google Chrome, Mozilla Firefox, Microsoft Edge, Opera тощо).

4.5.1 Вимоги до вхідних даних

Вхідні дані повинні бути представлені у форматі JSON, що є стандартом обміну даними між клієнтською та серверною частинами через REST API.

4.5.2 Вимоги до вихідних даних

Результати повинні бути представлені у форматі JSON, що забезпечує зручну обробку відповіді на стороні фронтенду. Дані можуть бути представлені у форматі HTML-інтерфейсу при відображенні у браузері або у вигляді email-повідомлень (формат MIME/HTML).

4.5.3 Вимоги до мови розробки

Розробку виконати мовами програмування:

– JavaScript (для клієнтської частини – React);
– JavaScript/Node.js (для серверної частини – Express.js);
– SCSS/CSS – для стилізації інтерфейсу.

4.5.4 Вимоги до середовища розробки

Розробку виконати за допомогою програмного середовища WebStorm та програмного забезпечення MongoDB Atlas. Для контролю версій коду використати Git та GitHub.

4.5.5 Вимоги до представленню вихідних кодів

Вихідний код програми має бути представлений у вигляді структурованих папок і файлів з дотриманням принципів модульності, розділення відповідальностей і найменування за стандартами (camelCase, kebab-case, PascalCase).

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

Необхідно згенерувати розгортальну версію програмного забезпечення. Розгортальна версія включає:

– повний вихідний код клієнтської та серверної частин;

– файл конфігурації .env.example з параметрами середовища;

– інструкцію з розгортання (файл README.md) з описом запуску за допомогою команд.

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;

– креслення вигляду екранних форм;

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

№	Назва етапу	Строк	Звітність
1.	Вивчення літератури за тематикою проєкту	15.03	
2.	Розробка технічного завдання	28.03	Технічне завдання
3.	Аналіз вимог та уточнення специфікацій	05.04	Специфікації програмного забезпечення
4.	Проєктування структури програмного забезпечення, проєктування компонентів	08.04	Схема структурна програмного забезпечення та специфікація компонентів
5.	Програмна реалізація програмного забезпечення	10.05	Тексти програмного забезпечення
6.	Тестування програмного забезпечення	15.05	Тести, результати тестування
7.	Розробка матеріалів текстової частини проєкту	18.05	Пояснювальна записка
8.	Розробка матеріалів графічної частини проєкту	18.05	Графічний матеріал проєкту
9.	Оформлення технічної документації проєкту	30.05	Технічна документація

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 організації роботи ветеринарної клініки**

КПІ.ІП-1204.045440.02.81

Київ – 2025

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

IDE	–	Integrated Development Environment – інтегроване середовище розробки.
API	–	Application programming interface, прикладний програмний Інтерфейс.
IT	–	Інформаційні технології.
БД	–	База даних.
ILF	–	Internal logical file, внутрішній логічний файл
EIF	–	External interface file, зовнішній інтерфейсний файл
EI	–	External input, зовнішній вхід
EO	–	External output, зовнішній вихід
EQ	–	External query, запит користувача
SPA	–	Single page application, односторінковий застосунок
JWT	–	JSON web token, JSON вебтокен

ВСТУП

Сьогодні значно зростає попит на сервіси онлайн-запису в різні структури: від салонів краси і лікарень до бронювання готелів чи квитків. Навіть здійснювати покупки можна не виходячи з дому. Саме тому актуальність розробки вебзастосунку для організації роботи ветеринарної клініки з можливістю онлайн-запису, перегляду інформації та розкладу є беззаперечною у сучасних умовах цифровізації як медичних послуг, так і послуг загалом.

Окрім зручності та витребуваності для потенційних клієнтів, застосунок має практичний сенс і для адміністрації клініки: основні організаційні функції будуть автоматизовані та доступні навіть з телефону. Електронне ведення карток пацієнтів, збереження інформації про прийоми, гнучке налаштування розкладу та контроль записів, ведення документообігу тощо може здійснюватись за допомогою зручного інструментарію вебзастосунку та практично повністю виключати з цього процесу необхідність комунікації по телефону та власноручне ведення окремих документів, значно зменшувати ймовірність виникнення помилок через людський фактор, забезпечувати цілісність та доступність даних.

Провідні розробки в цій та наближених сферах реалізовані у вигляді систем типу Helsi [1], PetDesk [2], VetClinicSoft [3], які частково або повністю автоматизують процеси обслуговування пацієнтів лікарні або вет-лікарні, проте більшість із них є платними, орієнтованими на західний ринок або не адаптованими до потреб малих і середніх клінік в Україні.

В рамках дипломного проєкту буде розроблено вебзастосунок, що дозволить керувати розкладом прийомів, вести електронні картки тварин, реєструвати власників, формувати історію процедур, а також забезпечить авторизацію для лікарів, адміністраторів та користувачів. Розробка надасть можливість власникам тварин переглядати та редагувати інформацію про тварин, просто і швидко здійснювати записи, отримувати доступ до свого акаунту без необхідності запам’ятовувати паролі, здійснювати онлайн-

замовлення ветеринарних товарів та лікарських засобів безпосередньо у ветлікарні тощо.

Призначення дипломного проєкту – розробка вебзастосунку, що автоматизує процеси роботи ветеринарної клініки. Метою є підвищення ефективності управління клінікою, покращення та пришвидшення обслуговування клієнтів і зменшення адміністративного навантаження на персонал.

Можливими сферами застосування є приватні та державні ветеринарні клініки, зоомедичні центри та мобільні ветеринарні сервіси.

1 ПЕРЕДПРОЄКТНЕ ОБСТЕЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Постановка завдання дипломного проєктування

Дипломне проєктування передбачає виконання наступних завдань:

– аналіз предметної області та опис її ключових бізнес-процесів, визначення загального завдання розробки у рамках ДП;

– аналіз існуючих рішень обраного завдання розробки у рамках ДП;

– аналіз та моделювання бізнес-процесів;

– розроблення функціональних, нефункціональних та системних вимог до програмного забезпечення;

– аналіз економічних показників програмного забезпечення ДП;

– постановка завдання на розробку програмного забезпечення ДП;

– розроблення архітектури програмного забезпечення;

– розроблення архітектурних рішень та обґрунтування вибору засобів розробки програмного забезпечення;

– конструювання та розроблення програмного забезпечення;

– аналіз безпеки даних програмного забезпечення;

– аналіз якості та тестування програмного забезпечення;

– розгортання та супровід програмного забезпечення;

– створення супроводжувальної документації до розробленого програмного забезпечення.

1.2 Аналіз предметної області

Ветеринарна медицина – це науково-практична сфера, яка охоплює вивчення фізіологічних процесів у тварин, дослідження хвороб, а також розробку методів їх профілактики, діагностики та лікування з метою підтримки здоров’я та продуктивності тварин.

Ветеринарне обслуговування передбачає професійну діяльність фахівців у галузі ветеринарії, що включає збір даних, проведення аналізів,

впровадження профілактичних заходів, лікування тварин та надання консультацій власникам.

Установа ветеринарної медицини – це організація, підприємство чи установа, де працює щонайменше один ветеринарний лікар і яка має право здійснювати лікувальні та ветеринарно-санітарні заходи. [4]

Зростання кількості домашніх улюбленців і підвищення рівня відповідальності власників тварин створює попит на якісні, швидкі та технологічно зручні послуги ветеринарного обслуговування.

На сучасному етапі розвитку IT-технологій цифровізація ветеринарної сфери розвивається у напрямках автоматизації прийомів, ведення електронних медичних карток тварин, формування графіків роботи персоналу, управління складом медикаментів та звітності. Програмне забезпечення, розроблене для ветеринарних клінік, часто має функціонал, схожий із системами для людської медицини, але адаптований до специфіки роботи з тваринами. [5]

Бізнес-процеси ветеринарної клініки:

– реєстрація та авторизація клієнтів за допомогою email або номеру телефону;

– створення графіку та контроль за розкладом прийомів;

– створення і ведення медичної картки тварини;

– онлайн-запис на прийом самостійно клієнтом та адміністратором;

– ведення історії прийомів та призначень пацієнтів;

– комунікація з клієнтами (нагадування, повідомлення);

– продаж лікарських засобів та ветеринарних товарів пацієнтам (опціонально);

– облік використаних під час прийому препаратів та проданих ветеринарних товарів, лікарських засобів тощо.

Процеси та їх реалізація у стані "as-is":

– Реєстрація клієнтів і тварин: виконується вручну в паперових журналах або в текстових редакторах. Дані дублюються або частково втрачаються, відсутня централізована база.

– Графік прийомів: лікарі ведуть графіки у паперових розкладах або окремих таблицях, що не синхронізуються між собою. Немає інтерактивної системи запису.

– Медичні картки тварин: часто відсутні або зберігаються у вигляді рукописних форм. Якщо і ведуться електронно, то без стандартизованої структури або пошуку.

– Запис на прийом: здійснюється лише телефоном або особисто при візиті. Клієнт не має доступу до запису через інтернет.

– Ведення історії процедур і призначень: дані фрагментарні, не пов’язані з карткою тварини, що ускладнює перегляд історії.

– Комунікація з клієнтами: здійснюється за телефоном, в кращому випадку за допомогою месенджерів.

– Продаж товарів клієнтам: якщо здійснюється, то фактично на місці без можливості онлайн-замовлення чи доставки.

– Облік товарів та медикаментів: підрахунок та визначення потреби ведеться вручну.

Таким чином, у стані *as-is* предметна область імплементована в ІТ лише частково, з використанням неінтегрованих, нефахових інструментів, що не забезпечують повноцінного управління клінікою.

На сьогодні дуже мало вет-клінік використовують існуючі в цій області застосунки. Причинами можуть бути складність використання, висока вартість підписки на сервіс, недостатня або ж навпаки надмірна кількість доступних функцій застосунку, замалий штат співробітників і як наслідок брак часу і персоналу для впровадження, вивчення та адміністрування онлайн-систем.

Можливими шляхами покращення є впровадження безкоштовної email-розсилки для входу в застосунок, надсилання нагадувань, електронних квитанцій про оплату та висновків лікарів; створення інтуїтивно зрозумілого інтерфейсу як для клієнта, так і для адміністратора системи; розміщення бази даних на хмарному сервері для економії фізичних ресурсів комп’ютера

користувача; розподіл ролей між адміністраторами для зручного та швидкого доступу до різних елементів та функцій програмного забезпечення. [6]

У рамках даного дипломного проєкту обрано шлях створення вебзастосунку з розподілом ролей (лікар, адміністратор, клієнт), можливістю створення та перегляду електронної картки тварини, системою запису на прийом, базою процедур, можливістю замовлення ветеринарних товарів та лікарських препаратів з використанням сучасного технологічного стеку: React, Node.js, Express, MongoDB.

1.3 Аналіз існуючих рішень

Для створення ефективного вебзастосунку для ветеринарної клініки доцільно розглянути існуючі алгоритмічні підходи та технічні рішення, що застосовуються у цій сфері. У цьому контексті буде проаналізовано актуальні програмні продукти, інструменти для розробки та допоміжне програмне забезпечення, яке може бути корисним під час реалізації системи.

1.3.1 Аналіз відомих програмних продуктів

Розглянемо існуючі програмні рішення у сфері розроблюваного продукту.

– Easyweek (рис. 1.1) – додаток для онлайн-бронювання. Це сайт, що кастомізується та має повний набір функціональних CRM-інструментів для управління бізнесом у сфері послуг. Має вигляд календаря з записами. [7]

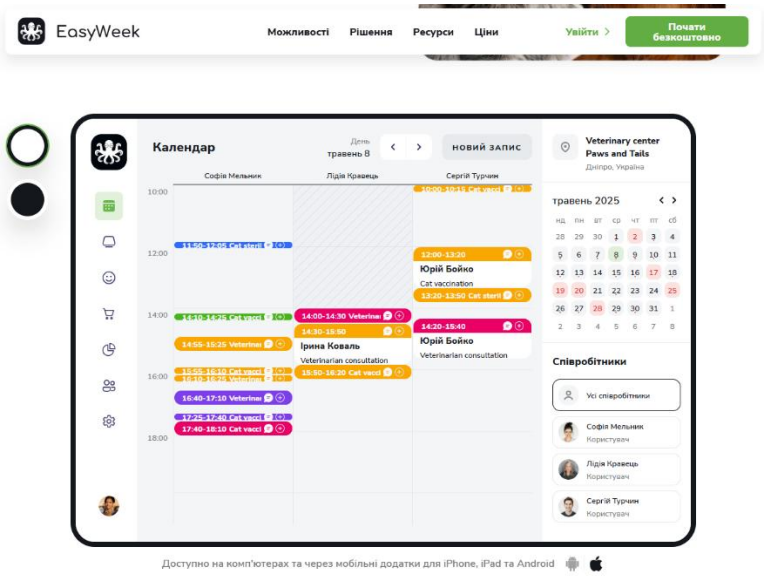


Рисунок 1.1 - Інтерфейс Easyweek

– Clinica Web (рис. 1.2-1.3) – медична інформаційна система, що є офіційним партнером реформи eHealth та підключена до центральної бази даних. Має окремий формат застосунку для ветеринарних клінік та дуже широкий спектр функцій.[8]

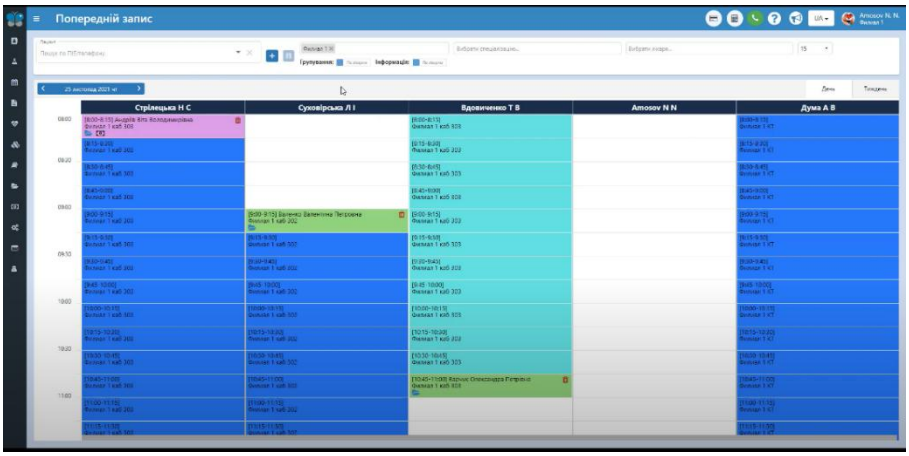


Рисунок 1.2 - Інтерфейс Clinica Web (частина 1)

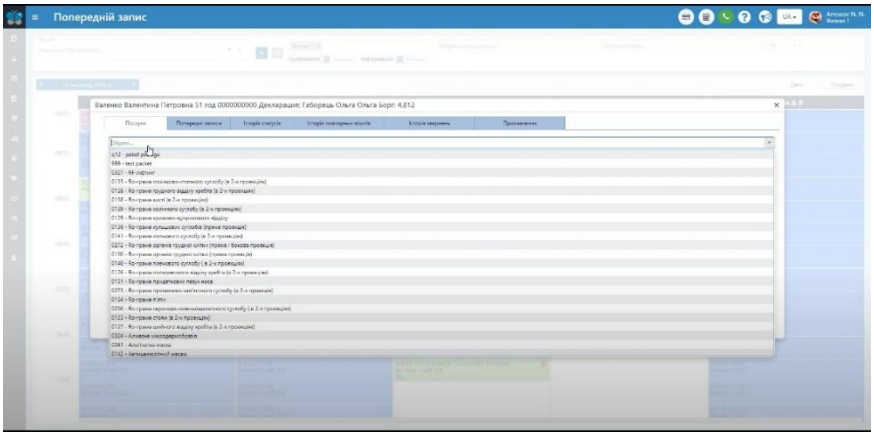


Рисунок 1.3 - Інтерфейс Clinica Web (частина 2)

– Appointer (рис. 1.4) – CRM система для середніх і малих підприємств, має окреме програмне забезпечення для ветеринарних клінік. [9]

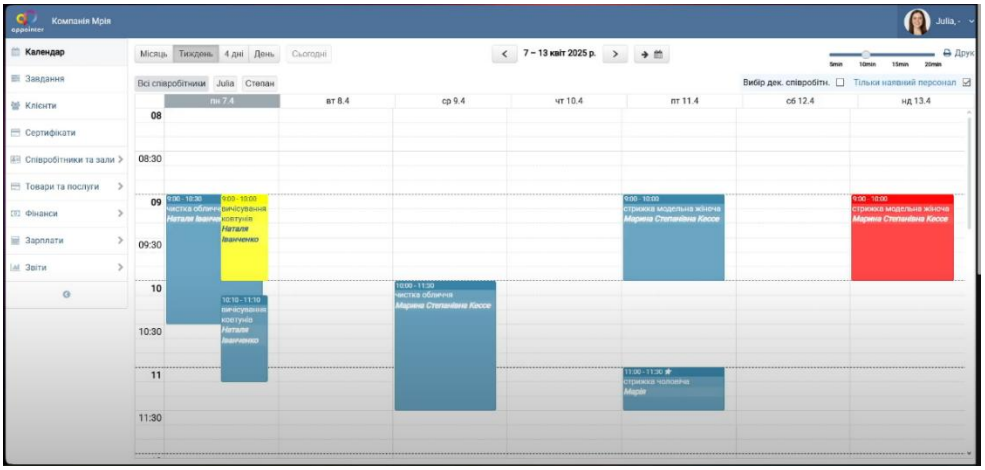


Рисунок 1.4 - Інтерфейс Appointer

– Jet Vet (рис. 1.5) – спеціалізоване програмне забезпечення для організації роботи вет-клініки. Має онлайн варіант для ПК чи ноутбука та додаток для смартфонів і планшетів. [10]

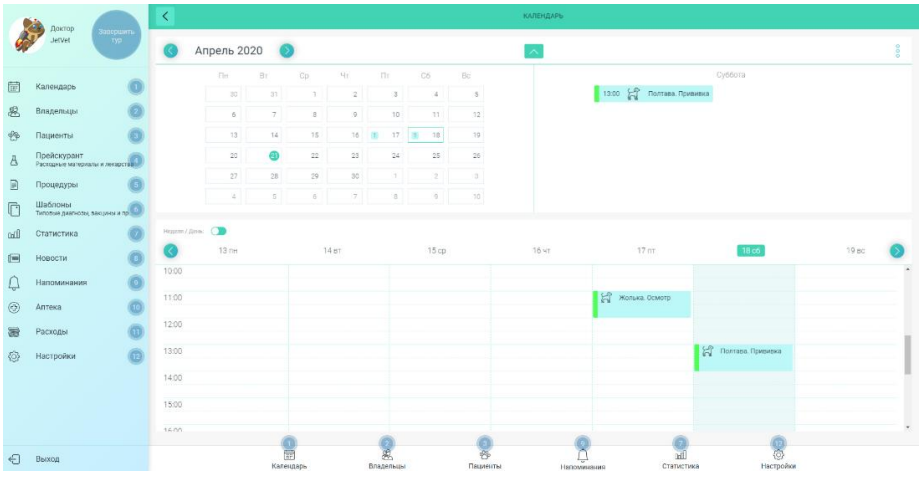


Рисунок 1.5 - Інтерфейс Jet Vet

Для порівняння проєкту з аналогами можна скористатись таблицею 1.1.

Таблиця 1.1 – Порівняння з аналогами

Критерій	PetHealth	Easyweek	Clinica Web	Appointer	Jet Vet	Пояснення
Доступ онлайн без встановлення сторонніх програм	+	+/-	+	+/-	+/-	Easyweek, Appointer та Jet Vet потребують встановлення додатку на телефон
Створення карток пацієнтів	+	+	+	+	+	
Система самостійного онлайн-запису клієнтами	+	+/-	+	+	-	Easyweek має відповідний модуль, що підключається за дод. кошти)
Підрахунок залишку кожного товару на складі	+	-	+	-	+/-	Jet Vet має підрахунок лише вручну
Управління правами користувачів	+	+	+	-	-	
Формування рецептів під час прийому	+	-	-	-	-	

Продовження таблиці 1.1

Критерій	PetHealth	Easyweek	Clinica Web	Appointer	Jet Vet	Пояснення
Онлайн-аптека	+	-	-	-	+/-	Jet Vet підтримує лише ручне внесення проданих товарів
Гнучке керування графіком роботи	+	-	-	-	-	

1.3.2 Аналіз відомих алгоритмічних та технічних рішень

Алгоритмічні рішення. У межах розробки вебзастосунку для ветеринарної клініки виникає потреба у реалізації наступних задач.

Пошук і фільтрація даних (тварини, власники, прийоми).

Задача: надати можливість зручно шукати тварин, власників, візити за різними параметрами (ПІБ, номер прийому та замовлення, вид тварини, дата прийому тощо).

Можливі підходи:

– Повнотекстовий пошук (Full-Text Search)

– Регулярні вирази ($regex)

– Комбіновані фільтри за кількома полями

Інструменти:

– MongoDB Aggregation Framework

– Індекси по ключовим полям (createIndex)

Обране рішення: використання MongoDB Aggregation Framework, зокрема оператори $match, $regex та комбіновані фільтри ($or, $and), що дозволяє здійснювати гнучке фільтрування за кількома полями та сортування.

Для підвищення продуктивності були застосовані індекси (unique, createIndex) на ключових полях. Забезпечує гнучкість і масштабованість. [11]

Облік та оновлення складу.

Задача: автоматично зменшувати залишки товарів та медикаментів після проведення процедур та продажу, контролювати наявність.

Можливі підходи:

– Таблиця залишків + логіка віднімання після процедури
– Облік за партіями (FIFO, FEFO)

Інструменти:

– MongoDB транзакції або атомарні оновлення ($inc, $set)
– Механізм подій або лог запису процедур

Обране рішення: реалізація простої моделі з оновленням залишку через $inc після кожної процедури/продажу.

Налаштування розкладу.

Формування розкладу базується на взаємодії двох ролей:

Адміністратор: встановлює робочі години для лікарів; обирає операційні дні; задає тривалість перерв (опціонально); формує загальний сітковий графік доступності.

Клієнт: обирає процедуру, яка має фіксовану тривалість; система підбирає вільний часовий слот згідно з графіком роботи; перевіряється, чи не перетинається вибраний час з уже запланованими прийомами.

Обране рішення:

– Реалізація власного механізму перевірки конфліктів за принципом:

start < existing_end && end > existing_start;

– Генерація вільних слотів на основі робочого графіку лікаря та тривалості процедури;

– Використання Redis pre-lock для тимчасового резервування слоту на час вибору клієнтом, що запобігає колізіям у разі одночасного вибору одного і того ж часу кількома користувачами. [12]

Технічні рішення. Порівняльний аналіз архітектур:

– Монолітна архітектура: простота реалізації, легше відлагоджувати, але менш масштабована.

– Клієнт-сервер: чіткий розподіл, гнучкість, незалежне оновлення, проте більша складність.

– Мікросервіси: масштабованість, незалежні модулі, але високий поріг входу, DevOps-навантаження.

Платформи та стек технологій:

Frontend:

Альтернативи: Angular, Vue, React.

Обрано: React (Висока гнучкість, підтримка, JSX, SPA). [13]

Backend:

Альтернативи: Django, Laravel, Express.js.

Обрано: Express.js (Простота, популярність, інтеграція з MongoDB). [14]

База даних:

Альтернативи: MySQL, PostgreSQL, MongoDB.

Обрано: MongoDB (Документно-орієнтована, гнучка структура даних).

Аутентифікація:

Альтернативи: Firebase Auth, Auth0, JWT.

Обрано: JWT (Безкоштовна, контроль з боку розробника). [15]

Хостинг:

Альтернативи: Heroku, Vercel, Render, VPS.

Обрано: Render (Простий CI/CD, безкоштовний план). [16]

Обране рішення: класична клієнт–сервер архітектура на основі React (frontend) + Express.js (backend) + MongoDB (база даних) – як найбільш відповідна для проєктів середньої складності, з чітким поділом ролей та масштабуванням у майбутньому.

1.4 Аналіз та моделювання бізнес-процесів

Наведемо опис моделей основних бізнес-процесів.

Відкриття картки пацієнта (адмін) (рис. 1.6):

– Адміністратор вводить ім’я (або інші атрибути) власника та виконує пошук.

– Якщо власник ще не зареєстрований, то відбувається створення нового власника, інакше - власника обрано.

– Після чого виконується пошук тварини цього власника.

– Якщо картки тварини ще немає, вона створюється, інакше - відкривається існуюча.

– Доступ до медкарти пацієнта отримано.

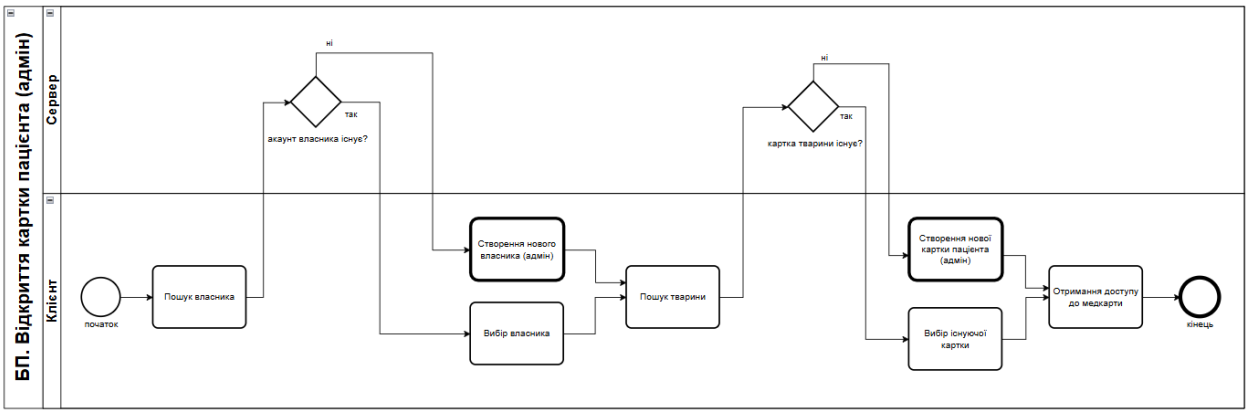


Рисунок 1.6 – Модель бізнес-процесу Відкриття картки пацієнта (адмін)

Створення запису на прийом (клієнт) (рис. 1.7):

– Власник тварини відкриває вкладку запису на прийом.

– У вікні бачить поля з відповідними даними для заповнення.

– У полях вибирає тварину (із зареєстрованих на цього власника), процедуру, дату та час прийому, додає опис проблеми за необхідності.

– Зберігає запис.

– Якщо усі дані введено і немає помилок, запис створюється та зберігається в БД, інакше - клієнт повинен перевірити форму та ввести дані або виправити помилки.

– Дані з серверу імпортуються на клієнтську частину, оновлення інтерфейсу.

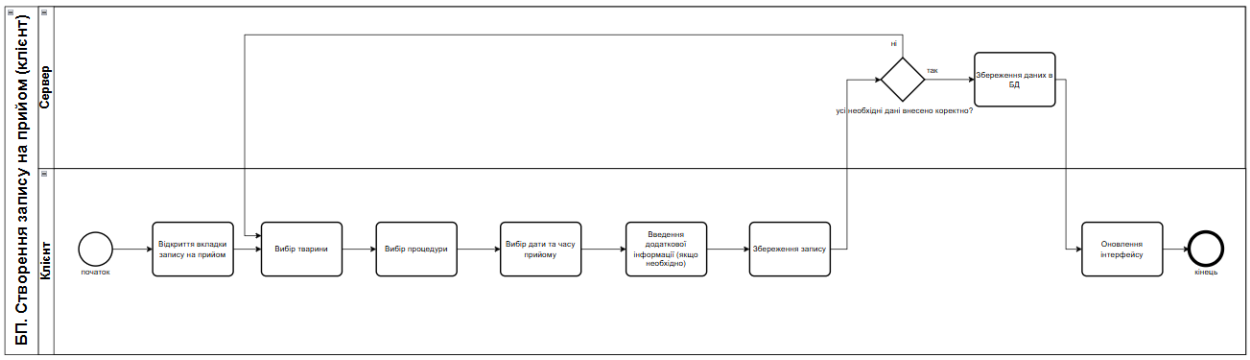


Рисунок 1.7 – Модель бізнес-процесу Створення запису на прийом (клієнт)

Створення запису на прийом (адмін) (рис. 1.8):

– Адміністратор відкриває картку пацієнта.

– Адміністратор вибирає процедуру, дату та час прийому, вносить додаткову інформацію, натискає зберегти запис.

– Якщо всі дані повні та введені коректно, то відбувається створення нового прийому та збереження його в БД, інакше - адміністратор повинен переглянути форму та виправити помилки або внести пропущені дані.

– Після успішного збереження дані імпортуються в інтерфейс користувача.

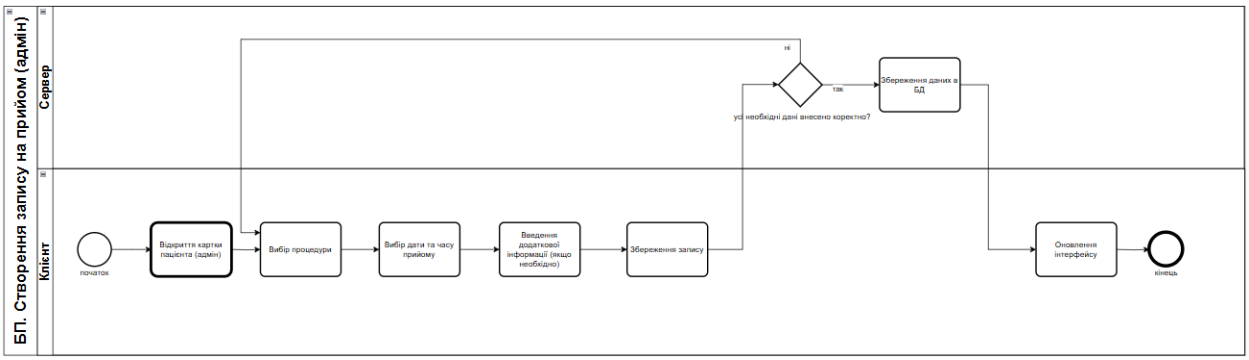


Рисунок 1.8 – Модель бізнес-процесу Створення запису на прийом (адмін)

Створення онлайн-замовлення у вет-аптеці (клієнт) (рис. 1.9):

– Клієнт відкриває вкладку вет-аптеки.

– Клієнт обирає категорію для пошуку, вводить слова у пошук.

– Відбувається пошук за введеним значенням, на екран виводяться доступні товари.

– Клієнт обирає товари, додає їх у кошик, обирає кількість.

– За необхідності покупки рецептурних препаратів клієнт обирає номер призначення на препарат.

– Вносяться дані доставки та оплати, клієнт натискає на кнопку підтвердження замовлення.

– Якщо в даних є помилки або вони неповні, то клієнт перевіряє форму та виправляє помилки, інакше - замовлення створюється та додається в БД, інформація в БД про кількість товарів, які були замовлені, оновлюється, інформація про замовлення надсилається на електронну пошту клієнту.

– Після успішного замовлення інтерфейс клієнта оновлюється.

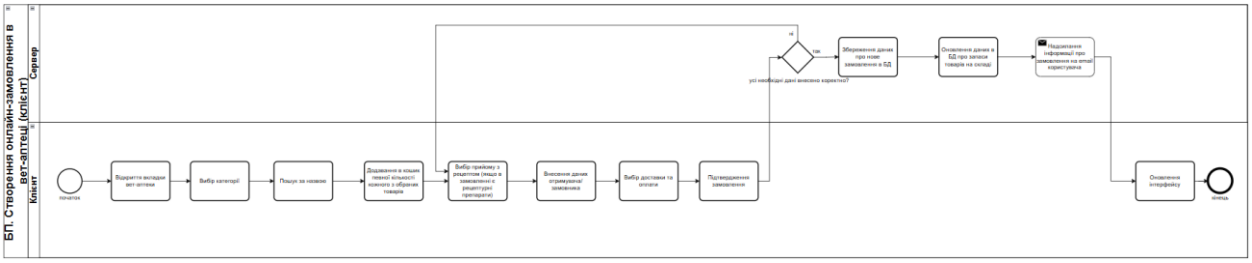


Рисунок 1.9 – Модель бізнес-процесу Створення онлайн-замовлення у вет-аптеці (клієнт)

Створення нової картки пацієнта (клієнт) (рис. 1.10):

– Клієнт додає нову картку пацієнта.

– Клієнт вносить необхідну інформацію про тварину та зберігає дані.

– Якщо дані внесено повністю та коректно, то нова картка зберігається в БД та оновлюється інтерфейс застосунку, інакше - клієнт має ввести правильні дані.

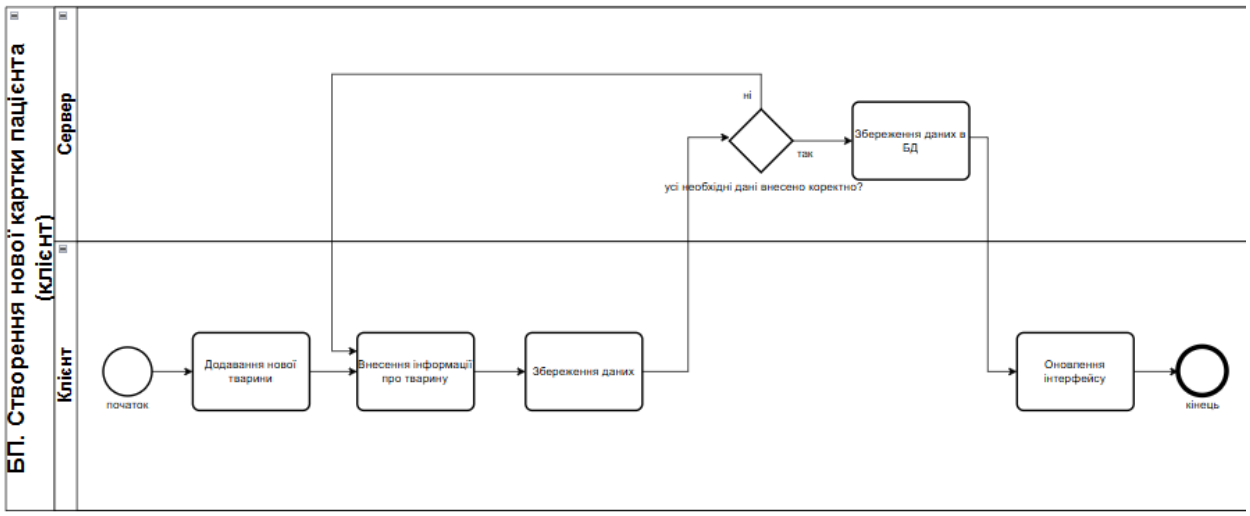


Рисунок 1.10 – Модель бізнес-процесу Створення нової картки пацієнта (клієнт)

Створення нової картки пацієнта (адмін) (рис. 1.11):

– Адміністратор обирає додавання нової картки.

– Адміністратор вводить дані про тварину, додає власника, який вже має бути зареєстрований, та зберігає дані.

– Якщо дані внесено не повністю або з помилками, то адміністратор повинен їх виправити, інакше - нова картка зберігається в БД та оновлюється інтерфейс системи.

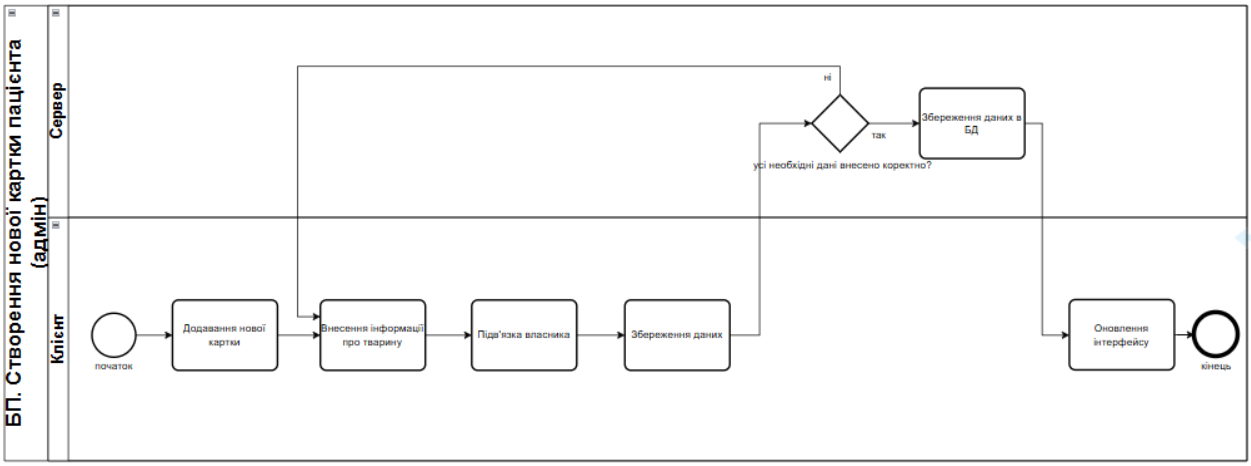


Рисунок 1.11 – Модель бізнес-процесу Створення нової картки пацієнта (адмін)

Закриття прийому (адмін) (рис. 1.12):

– Адміністратор відкриває картку пацієнта та обирає прийом.

– В формі прийому адміністратор змінює статус прийому, вносить дані про тварину, діагноз, призначення лікаря, вказує препарати та засоби, які були використані протягом прийому, додає рецепт за необхідності тощо.

– Якщо дані введено правильно, то інформація про прийом та кількість товарів, які були використані, оновлюється в БД, інакше - адміністратор повинен виправити помилки в формі прийому.

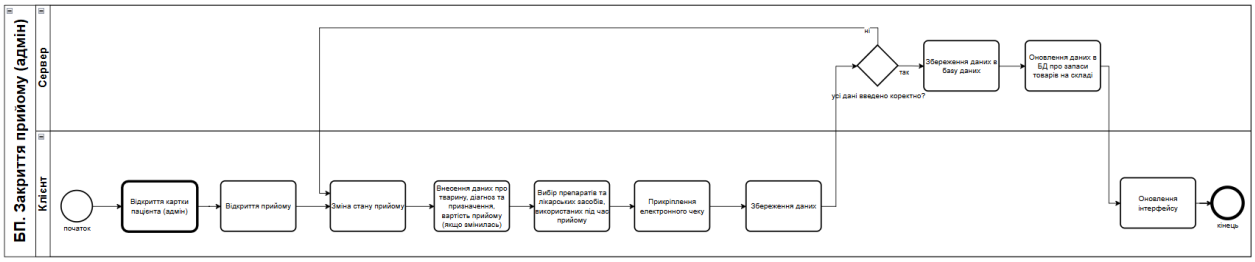


Рисунок 1.12 – Модель бізнес-процесу Закриття прийому (адмін)

Додавання товару (адмін) (рис. 1.13):

– Адміністратор відкриває вкладку створення товару.

– Адміністратор вводить дані про товар: ціну, опис , категорію, рецептурний він чи ні тощо, та зберігає дані.

– Якщо необхідні дані було введено коректно, то новий товар зберігається в БД, а інтерфейс системи оновлюється, інакше - адміністратор повинен виправити помилки або доповнити дані.

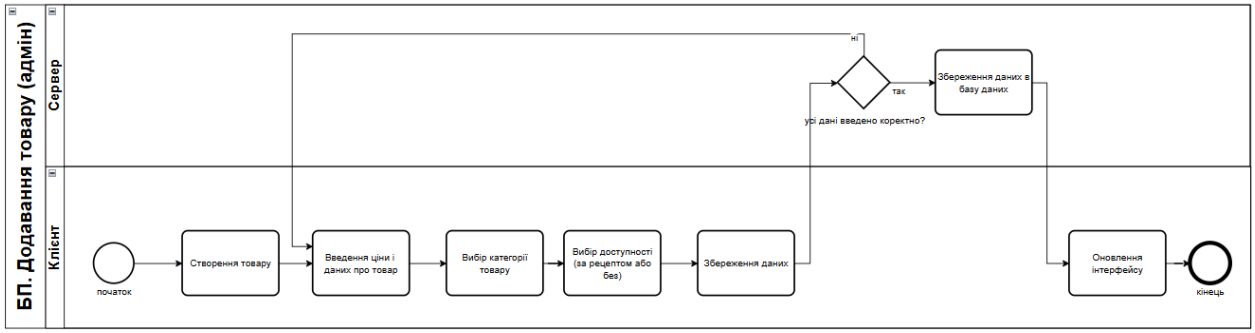


Рисунок 1.13 – Модель бізнес-процесу Додавання товару (адмін)

Редагування інформації про тварину (клієнт) (рис. 1.14):

– Клієнт відкриває вкладку редагування та обирає тварину для редагування.

– У відкритій формі клієнт редагує дані про тварину та зберігає їх.

– Якщо дані було введено коректно, то оновлена інформація про тварину зберігається в БД, а інтерфейс системи оновлюється, інакше - клієнт повинен виправити помилки.

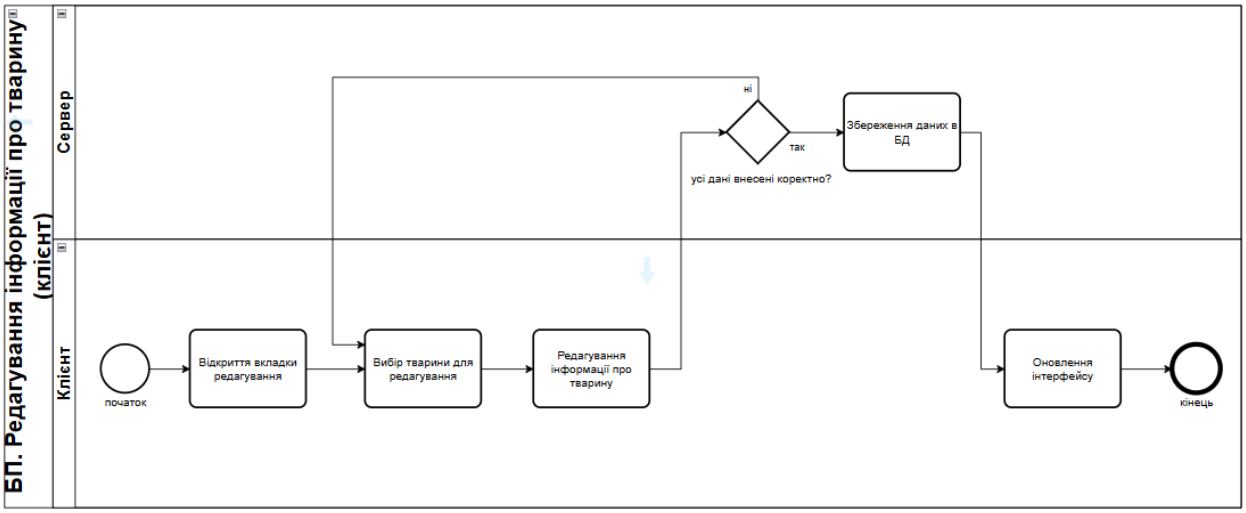


Рисунок 1.14 – Модель бізнес-процесу Редагування інформації про тварину (клієнт)

Створення графіку роботи (адмін) (рис. 1.15):

– Адміністратор відкриває вкладку редагування графіку роботи.

– Адміністратор може обрати вихідні дні, час роботи та перерв, операційні дні, після чого зберігає дані.

– Якщо всі дані було введено коректно, то графік оновлюється в БД та оновлюється інтерфейс системи, інакше - адміністратор повинен перевірити правильність та виправити помилки.

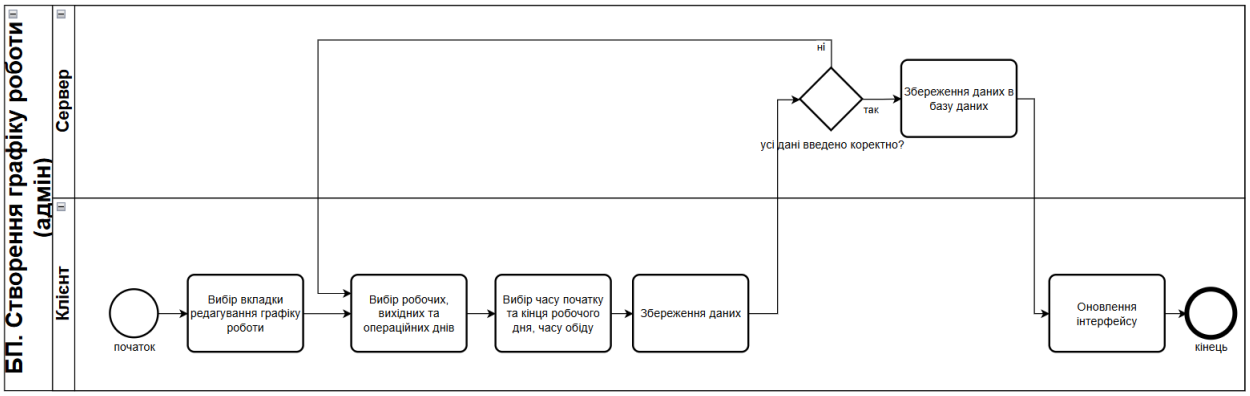


Рисунок 1.15 – Модель бізнес-процесу Створення графіку роботи (адмін)

Додавання процедури (адмін) (рис. 1.16):

– Адміністратор відкриває картку створення процедури.

– Адміністратор вводить дані про процедуру, тривалість, доступність запису тощо, та зберігає дані.

– Якщо всі необхідні дані було введено коректно, то графік оновлюється в БД та оновлюється інтерфейс системи, інакше - адміністратор повинен перевірити правильність та виправити помилки.

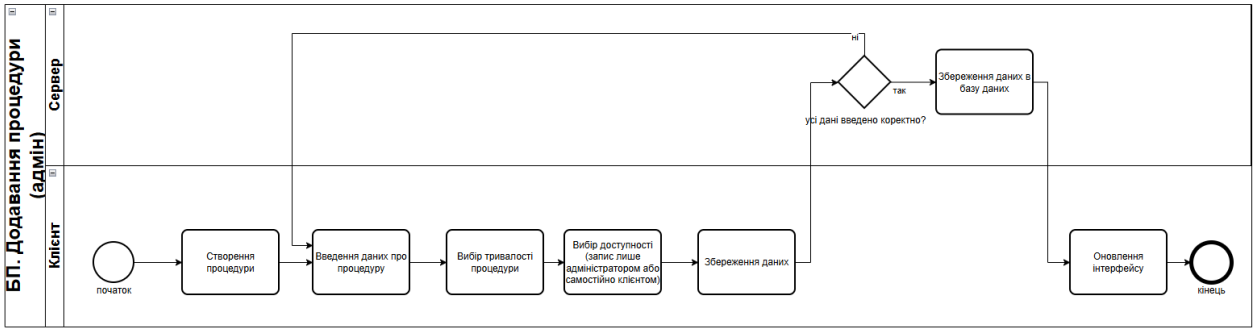


Рисунок 1.16 – Модель бізнес-процесу Додавання процедури (адмін)

1.5 Висновки до розділу

У даному розділі було проведено аналіз предметної області ветеринарної медицини з точки зору її цифровізації. Визначено основні особливості функціонування ветеринарних клінік, охарактеризовано бізнес-процеси, які є критично важливими для ефективної організації роботи таких закладів. Серед них: реєстрація користувачів, створення та ведення електронних медичних карток тварин, управління графіком роботи лікарів, запис клієнтів на прийом, облік препаратів та ветеринарних товарів, комунікація з клієнтами та внутрішній документообіг.

Проведено аналіз поточного стану впровадження IT-рішень у сфері ветеринарного обслуговування. Встановлено, що значна частина клінік і надалі використовує паперову або частково електронну, неінтегровану форму збереження інформації, що ускладнює управління і призводить до втрати або дублювання даних.

Проведено огляд та порівняльний аналіз існуючих програмних рішень у сфері автоматизації ветеринарної діяльності (Easyweek, Clinica Web, Appointer, Jet Vet). Встановлено, що розглянуті рішення є дороговартісними

для клінік з невеликим штатом, деякі з них мають надлишкову функціональність або ускладнюють роботу.

Розглянуто алгоритмічні та технічні підходи до реалізації ключових задач: пошуку та фільтрації даних, обліку залишків на складі, збереження інформації. Обрані рішення забезпечують баланс між функціональністю, ефективністю та простотою реалізації.

Виконано моделювання основних бізнес-процесів за допомогою BPMN, що дозволило чітко структурувати функціональні вимоги до програмного забезпечення, визначити точки взаємодії користувача та системи.

2 РОЗРОБЛЕННЯ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Варіанти використання програмного забезпечення

Діаграма варіантів використання з ключовими акторами та основними функціями показана на графічному матеріалі, креслення 1.

Опис варіантів використання програмного забезпечення наведено у Додатку Б.

2.2 Розроблення функціональних вимог

Програмне забезпечення розділене на модулі. Кожен модуль має свій певний набір функцій. В таблиці 2.1 наведено загальну модель вимог, а в таблиці 2.2 наведений опис функціональних вимог до програмного забезпечення. Матрицю трасування вимог можна побачити на рисунку 2.2.

Таблиця 2.1 – Загальна модель вимог

№	Назва	ID Вимоги	Пріоритети	Ризики
1	Система авторизації користувача	FR-1	Високий	Високий
2	Реєстрація користувача	FR-2	Високий	Високий
3	Авторизація користувача	FR-3	Високий	Високий
4	Вихід із акаунту	FR-4	Середній	Низький
5	Створення картки пацієнта	FR-5	Високий	Середній
6	Перегляд картки пацієнта	FR-6	Середній	Низький
7	Редагування картки пацієнта	FR-7	Високий	Середній
8	Видалення картки пацієнта	FR-8	Середній	Середній
9	Створення прийому	FR-9	Високий	Середній
10	Редагування прийому	FR-10	Середній	Середній
11	Видалення прийому	FR-11	Середній	Середній
12	Запис на прийом клієнтом	FR-12	Високий	Високий
13	Онлайн-замовлення у вет-аптеці	FR-13	Високий	Середній
14	Редагування графіку роботи	FR-14	Середній	Середній

Продовження таблиці 2.1:

№	Назва	ID Вимоги	Пріоритети	Ризики
16	Додавання товару	FR-15	Високий	Середній
17	Редагування товару	FR-16	Середній	Середній
18	Видалення товару	FR-17	Середній	Середній
19	Управління акаунтами власників	FR-18	Високий	Середній
20	Налаштування даних для входу	FR-19	Високий	Високий
21	Перегляд та редагування замовлень	FR-20	Середній	Середній

Таблиця 2.2 – Перелік функціональних вимог

Назва	Опис
FR-1	Система авторизації користувача Система повинна реалізовувати механізм перевірки користувача перед доступом до особистого кабінету. Авторизація відбувається через введення email та одноразового коду, надісланого на пошту.
FR-2	Реєстрація користувача Система дозволяє створення нового облікового запису через email і номер телефону. Користувач підтверджує реєстрацію кодом із пошти. Дублікати пошти/телефону заборонено.
FR-3	Авторизація користувача Після введення email і коду підтвердження користувач входить у систему. Успішна авторизація створює токен доступу, що зберігається локально.
FR-4	Вихід із акаунту Користувач має змогу завершити сесію, після чого токен доступу видаляється, і користувача перекидає на сторінку входу.

Продовження таблиці 2.2:

Назва	Опис
FR-5	Створення картки пацієнта Користувач може створити нову картку тварини, вказуючи кличку, вид, вік тощо. Адміністратор також обирає власника.
FR-6	Перегляд картки пацієнта Користувач може переглядати інформацію про тварину, її медичну історію, записи на прийом, призначення та процедури.
FR-7	Редагування картки пацієнта Редагування дозволяє оновити дані тварини. Зміни зберігаються лише після підтвердження.
FR-8	Видалення картки пацієнта Адміністратор може видалити картку пацієнта, лише якщо немає прив’язаних записів або процедур.
FR-9	Створення прийому Адміністратор може додати новий прийом до картки тварини, вказавши дату, час, лікаря, процедуру й симптоми.
FR-10	Редагування прийому Система дозволяє змінювати інформацію в записі про прийом — процедури, діагноз, препарати, дату або лікаря.
FR-11	Видалення прийому Адміністратор може видалити прийом, якщо він не завершений та не оплачений.
FR-12	Запис на прийом клієнтом Клієнт може самостійно обрати дату, процедуру, лікаря та час із доступного розкладу. Система не допускає конфліктів у розкладі.
FR-13	Онлайн-замовлення у вет-аптеці Користувач може додати товари до кошика, вибрати спосіб доставки/оплати та оформити онлайн-замовлення.

Продовження таблиці 2.2:

Назва	Опис
FR-14	Редагування графіку роботи Адміністратор змінює графік роботи: робочі дні, час початку/завершення зміни, обід. Ці дані використовуються у розкладі.
FR-15	Додавання товару Адміністратор може додати новий товар до вет-аптеки: назву, опис, категорію, фото, ціну та позначку «рецептурний».
FR-16	Редагування товару Існуючий товар можна редагувати: змінити опис, ціну, зображення, категорію, статус «рецептурний» або доступність.
FR-17	Видалення товару Адміністратор може видалити товар із аптеки, якщо він не пов’язаний з активним замовленням або історією покупок.
FR-18	Управління акаунтами власників Адміністратор може додавати, редагувати або видаляти акаунти власників. Видалення можливе лише без активних пацієнтів.
FR-19	Налаштування даних для входу Користувач може змінювати свої дані для входу в систему, для підвердження зміни необхідно отримати код на пошту та ввести його у відповідне поле.
FR-20	Перегляд та редагування замовлень Адміністратор може переглядати та редагувати створені клієнтами замовлення.

	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
UC-01	X	X																		
UC-02	X		X																	
UC-03				X																
UC-04					X															
UC-05							X													
UC-06						X														
UC-07																			x	
UC-08												X								
UC-09													X							
UC-10															X					
UC-11																X				
UC-12																	X			
UC-13																				
UC-14														X						
UC-15																		X		
UC-16																		X		
UC-17																		X		
UC-18					X															
UC-19							X													
UC-20								X												
UC-21									X											
UC-22										X										
UC-23											X									

Рисунок 2.1 – Матриця трасування вимог

2.3 Розроблення нефункціональних вимог

– Вимоги до продуктивності (Performance requirements).

Система повинна забезпечувати відкриття сторінок не довше ніж за 2 секунди при середньому навантаженні (до 100 користувачів одночасно). Обробка запису на прийом або замовлення не повинна перевищувати 1 секунди після натискання кнопки.

– Вимоги до надійності (Reliability).

Система повинна мати доступність не нижче 99% протягом робочого місяця. У разі збою з’єднання дані, що не були збережені, мають бути відновлені або користувач має бути повідомлений.

– Вимоги до безпеки (Security).

Всі API-запити мають бути захищені через HTTPS. Кожна сесія повинна мати JWT-токен, термін дії якого – до 30 хвилин без активності. Доступ до

персональних даних розмежовується за ролями: клієнт, адміністратор, лікар. Спроби авторизації з неправильним кодом більше 5 разів блокуються на 10 хвилин.

– Вимоги до зручності використання (Usability).

Інтерфейс повинен бути адаптивним (працювати на смартфонах, планшетах, ПК). Система має інтуїтивну навігацію, зрозумілі підказки при помилках.

– Вимоги до масштабованості (Scalability).

Система має бути готова до підключення множинних філій клінік без повного перероблення архітектури. База даних повинна підтримувати тисячі записів пацієнтів, прийомів і процедур без втрати продуктивності.

– Вимоги до портативності (Portability).

Вебзастосунок повинен працювати в сучасних браузерах: Opera, Google Chrome, Mozilla Firefox, Microsoft Edge, Safari. Підтримка різних операційних систем забезпечується через браузер (Windows, macOS, Linux, Android, iOS).

2.4 Аналіз системних вимог

– Клієнтська частина (браузер)

Тип доступу: через веббраузер (без встановлення)

Підтримувані браузери: Google Chrome 95+, Mozilla Firefox 90+, Microsoft Edge 90+, Safari 14+, Opera 80+.

Пристрої: комп’ютери, планшети, смартфони

Мінімальна роздільна здатність екрана: 360×640 (адаптивна верстка)

– Серверна частина Операційна система сервера: Linux (Ubuntu 20.04+ або сумісна)

Node.js: версія 18 або вище

СУБД: MongoDB 6.0+

Платформа розміщення: Render (або інша платформа з підтримкою Node.js та MongoDB Atlas)

RAM сервера: мінімум 512 МБ (рекомендовано 1 ГБ+ для продакшну)

CPU: мінімум 1 core (рекомендовано 2+)

– Браузерні технології Frontend: React.js SPA

Мова розмітки: HTML5 + CSS3

Інші вимоги: увімкнена підтримка JavaScript

2.5 Аналіз економічних показників програмного забезпечення

Використаємо метод функціональних точок - Function Point Analysis (FPA) [17] та розрахуємо кількість точок у таблиці 2.3.

Таблиця 2.3 – Перелік функціональних вимог

Компонент	Кількість	Складність	FP
ILF (картки пацієнтів, користувачі)	3	Середня (10)	30
EIF (інтеграція з поштою)	1	Середня (7)	7
EI (реєстрація, авторизація, редагування)	5	Середня (4)	20
EO (сповіщення)	1	Середня (5)	5
EQ (пошук, фільтрація)	2	Середня (4)	8
Разом			70

Середня зарплата розробника в Україні (2025) за даними Robota.ua [18] на посаді Junior Fullstack Developer ~30 000 грн.

Обрахуємо вартість години роботи: 30 000 / 160 год = 187,5 грн/год

За типовою продуктивністю розробника:

1 FP ≈ 10 годин розробки

Отже, 70 FP × 10 год = 700 год

Отримуємо вартість розробки: 700 год × 187,5 грн/год = 131 250 грн

Порівняємо з аналогами у таблиці 2.4:

Таблиця 2.4 – Порівняння економічної вартості з аналогами

Продукт	Вартість/місяць	Вартість/рік	Коментар
PetHealth	≈ 800 грн	≈ 9 600грн	Доступ до коду, контроль та налаштування функціоналу
Clinica Web	900 грн	10 800 грн	Ліцензія на одного користувача
JetVet	1 200 грн	14 400 грн	Без підтримки кастомізації
Appointer	500 грн	6 000 грн	Базовий тариф без API-доступу
EasyWeek	700 грн	8 400	Необмежені користувачі, API

Розробка власного вебзастосунку потребує вищих стартових вкладень, однак дозволяє уникнути абонплат, повністю контролювати функціональність, адаптувати систему під власні потреби. У довгостроковій перспективі (2+ роки) власне рішення є економічно вигіднішим, особливо якщо кількість користувачів зростатиме.

2.6 Постановка завдання на розробку програмного забезпечення

У межах дипломного проєкту поставлено задачу розробити вебзастосунок для автоматизації роботи ветеринарної клініки, який забезпечить зручну взаємодію між адміністрацією, лікарями та власниками тварин, дозволить ефективно керувати записами на прийом, електронними медичними картками, графіком роботи, аптекою та замовленнями, а також забезпечить безпечне зберігання та швидкий доступ до медичної інформації.

Метою розробки є підвищення якості організації роботи ветеринарної клініки шляхом забезпечення зручної взаємодії між адміністрацією, лікарями та власниками тварин, а також автоматизації обліку пацієнтів, прийомів, розкладу, товарів і замовлень.

Система має забезпечити авторизований доступ для адміністраторів, лікарів та клієнтів із чітким розмежуванням прав доступу. Повинна бути реалізована підтримка електронних медичних карток тварин. Застосунок має дозволяти запис на прийом з урахуванням графіка роботи лікарів, формувати

та обробляти прийоми, в тому числі з можливістю створення електронних рецептів і ведення обліку використаних лікарських засобів. Передбачається також функціональність для онлайн-замовлення товарів через модуль ветеринарної аптеки з підтримкою перевірки наявності рецептів. Інтерфейс системи повинен бути адаптивним і зрозумілим для користувачів різного рівня.

Висновки до розділу

В розділі було проведено аналіз предметної області – автоматизації роботи ветеринарної клініки. Визначено основні проблеми: відсутність єдиної електронної бази, ручне управління записами, слабка цифровізація обліку пацієнтів, товарів і розкладу.

Проаналізовано існуючі аналоги (Easyweek, Clinica Web, JetVet, Appointer), виокремлено їхні обмеження. Обґрунтовано доцільність створення власного рішення PetHealth.

Виконано постановку завдання, сформовано мету, задачі, функціональну та нефункціональну структуру ПЗ. Побудовано бізнес-процеси в нотації BPMN, діаграму варіантів використання та сценарії взаємодії користувачів із системою.

Здійснено техніко-економічне обґрунтування: оцінено трудомісткість розробки методом FPA , визначено вартість розробки для одного виконавця. Порівняння з SaaS-сервісами підтвердило вигідність власного застосунку в довгостроковій перспективі.

За результатами розділу сформовано технічне завдання на розробку програмного забезпечення.

3 КОНСТРУЮВАННЯ ТА РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Архітектура програмного забезпечення

Для розробки вебзастосунку було обрано архітектурний патерн «клієнт–сервер» у поєднанні з принципами трирівневої архітектури (Three-tier architecture), яка логічно розділяє систему на:

– Презентаційний рівень (Frontend / Client). Відповідає за взаємодію з користувачем. Реалізований як односторінковий застосунок за допомогою React.js. Усі дії (авторизація, перегляд картки, запис на прийом тощо) надсилаються через HTTP-запити до серверної частини.

– Серверна логіка (Backend / Application Layer). Реалізована за допомогою Node.js з фреймворком Express.js. Сервер приймає запити, виконує валідацію, обробляє бізнес-логіку та взаємодіє з базою даних. Також відповідає за авторизацію, надсилання email-кодів, обробку звітів тощо.

– Рівень зберігання даних (Data Layer). Представлений MongoDB – нереляційною документною базою даних. Усі сутності системи зберігаються у вигляді документів у відповідних колекціях.

Використаємо діаграму моделі C4 (1-2 рівні) [19], зображену на графічному матеріалі, креслення 3, аркуші 1-2.

3.2 Архітектурні рішення та обґрунтування вибору засобів розробки

Під час розробки вебзастосунку було обрано архітектурні рішення, спрямовані на забезпечення забезпечення масштабованості, безпеки, зручності використання та гнучкого розширення в майбутньому.

Використано клієнт–серверну архітектуру з чітким розділенням логіки на фронтенд, бекенд та базу даних. Цей підхід дозволяє окремо масштабувати фронтенд і бекенд, забезпечити високу швидкодію, легко оновлювати окремі компоненти системи без перезапуску всієї архітектури.

Для управління користувачами реалізована аутентифікація всередині системи, без використання сторонніх провайдерів. Метод логіну – email та одноразовий код підтвердження, що надсилається на пошту. Це забезпечує простоту для користувача, не потребує пароля, а також знижує ризики зберігання чутливих даних.

Шляхом інтеграції із зовнішньою email-системою забезпечується підтвердження входу, редагування даних акаунту користувача, надсилання нагадувань про прийом тощо. У рамках взаємодії із зовнішнім email-сервісом дані передаються у форматі JSON, а сформовані листи автоматично конвертуються сервісом у формат MIME, відповідно до специфікації SMTP-протоколу.

Для зберігання даних використано документно-орієнтовану базу даних типу NoSQL. Причинами вибору стали: зручна робота з вкладеними структурами, можливість вбудованих масивів.

Порівняємо можливі засоби розробки програмного забезпечення у таблиці 3.1:

Таблиця 3.1 – Порівняння засобів розробки

Засіб	Альтернативи	Обране рішення	Причина вибору
Мова програмування	JavaScript, Python, PHP	JavaScript	Універсальність для фронт+бекенд
Фреймворк для клієнта	Vue, Angular, React	React	Підтримка SPA, гнучкість
Фреймворк для сервера	Express.js, Koa, Nest	Express.js	Легка інтеграція з MongoDB, швидкодія
СУБД	PostgreSQL, MongoDB	MongoDB	NoSQL, документно-орієнтована, гнучка структура, хмарний хостинг

Продовження таблиці 3.1:

Засіб	Альтернативи	Обране рішення	Причина вибору
Авторизація	OAuth 2.0, Session ID, JWT	JWT	Простота, контроль на сервері
IDE	WebStorm, VS Code	WebStorm	Наявність студентської підписки, зручний інтерфейс, плагіни
Система email-сповіщень	Nodemailer, SendGrid	Nodemailer	Простота інтеграції з Express.js, підтримка MIME/HTML
Підтримка реального часу	Socket.IO, WebSockets	Socket.IO	Використовується для оновлення слотів у реальному часі
Кешування та блокування	Redis, Memcached	Redis	Для блокування слотів у реальному часі, для блокування товарів під час формування замовлень

Обрані архітектурні рішення дозволяють побудувати гнучку, модульну, безпечну і зручну систему для організації роботи ветеринарної клініки. Всі компоненти інтегруються між собою за чіткими протоколами, що спрощує розширення та супровід системи.

3.3 Конструювання програмного забезпечення

У розроблюваному застосунку використано прості алгоритми перевірки конфліктів при генерації доступного часу для запису, фільтрації та пошуку по БД, оновлення кількості товарів на складі.

Структури даних зберігаються у вигляді JSON-документів у MongoDB. Клієнтська частина реалізована з використанням React.js, розділена на модулі відповідно до основних функцій. Серверна частина (Node.js + Express) складається з окремих маршрутів, контролерів та сервісів, що забезпечують обробку даних та взаємодію з БД.

3.3.1 Опис структури бази даних

У розроблюваному вебзастосунку використовується документно-орієнтована база даних MongoDB, яка дозволяє гнучко зберігати об'єкти у форматі JSON-подібних документів. Основними сутностями системи є картки пацієнтів (тварин), користувачі (адміністратори, клієнти, лікарі), прийоми, процедури, замовлення, товари аптеки та декілька службових сутностей. В базі даних реалізовано вкладені масиви та зовнішні ключі для зв’язків. Опис сутностей БД наведено у таблицях 3.2 - 3.12.

Таблиця 3.2 – Опис таблиці User

Назва поля	Тип даних	Опис
id	ObjectId	ідентифікаційний номер користувача
email	string	електронна пошта користувача
phone	string	номер телефону користувача
createdAt	dateTime	дата створення екземпляру колекції
role	string	роль користувача (адмін, клієнт або лікар)
firstName	string	ім'я користувача
lastName	string	прізвище користувача
birthday	date	дата народження користувача
isArchived	boolean	статус в архіві, 0 - не в архіві, 1 - в архіві (не висвічується клієнту)

Таблиця 3.3 – Опис таблиці PetCard

Назва поля	Тип даних	Опис
id	ObjectId	ідентифікаційний номер картки пацієнта
name	string	ім'я пацієнта
speciesId	dateTime	ідентифікаційний номер типу тварини
breed	string	порода тварини
ownerId	ObjectId	ідентифікаційний номер власника (користувача-клієнта)
birthday	date	дата народження
createdAt	dateTime	дата та час створення екземпляру колекції
isArchived	boolean	статус в архіві, 0 - не в архіві, 1 - в архіві (не висвічується клієнту)

Таблиця 3.4 – Опис таблиці Species

Назва поля	Тип даних	Опис
id	ObjectId	ідентифікаційний номер породи
name	string	назва породи

Таблиця 3.5 – Опис таблиці Appointment

Назва поля	Тип даних	Опис
id	ObjectId	ідентифікаційний номер прийому
userId	ObjectId	ідентифікаційний номер власника тварини
petId	ObjectId	ідентифікаційний номер картки пацієнта, який записаний на прийом
procedureId	ObjectId	ідентифікаційний номер процедури, на яку записаний пацієнт
recipeId	ObjectId	ідентифікаційний номер рецепту, який виписується на прийомі
appointmentNumber	string	номер прийому

Продовження таблиці 3.5:

Назва поля	Тип даних	Опис
totalPrice	number	загальна вартість прийому разом з використаними медикаментами (якщо вони є)
date	date	дата прийому
startTime	time	час початку прийому
endTime	time	час завершення прийому
comment	string	коментар користувача при запису на прийом
status	string	ідентифікаційний номер статусу прийому (запланований, триває, скасований, завершений)
createdAt	dateTime	дата та час створення екземпляру колекції
animalInfo	embedded document	вбудований документ з полями: вага, температура, стан
diagnosis	string	діагноз
prescription	string	призначення лікаря
usedMedicines	array[embedded document]	список використаних медикаментів та ветеринарних товарів з кількістю кожного товару
isArchived	boolean	статус в архіві, 0 – не в архіві, 1 – в архіві (не висвічується клієнту)

Таблиця 3.6 – Опис таблиці Code

Назва поля	Тип даних	Опис
id	ObjectId	ідентифікаційний номер коду підтвердження
email	string	email користувача, якому надіслано код
code	string	код підтвердження
expiresAt	date	час спливання строку дії коду (автовидалення)

Таблиця 3.7 – Опис таблиці Procedure

Назва поля	Тип даних	Опис
id	ObjectId	ідентифікаційний номер процедури
name	string	назва процедури

Продовження таблиці 3.7:

Назва поля	Тип даних	Опис
speciesId	ObjectId	ідентифікаційний номер виду тварини, для якої ця процедура (якщо не вказано – універсальна)
doctorId	ObjectId	ідентифікаційний номер акаунту лікаря, який виконує процедуру
price	number	ціна процедури
selfBooking	boolean	доступність, 0 - може створити запис лише адмін, 1 - запис на процедуру доступний для всіх
duration	number	тривалість процедури у хвилинах
inArchived	boolean	статус в архіві, 0 - не в архіві, 1 - в архіві (не висвічується клієнту)

Таблиця 3.8 – Опис таблиці Product

Назва поля	Тип даних	Опис
id	ObjectId	ідентифікаційний номер товару
name	string	назва товару
shortDescription	string	короткий опис
longDescription	string	довгий опис
speciesId	ObjectId	ідентифікаційний номер виду тварини, для якої цей товар (якщо не вказано – універсальна)
isPrescriptionFree	boolean	продаж без рецепту, 0 - можна купити лише за рецептом лікаря, 1 - продається у вільному доступі.
categoryId	ObjectId	ідентифікаційний номер категорії товару
unit	string	одиниці вимірювання
stock	number	кількість одиниць товару на складі
price	number	ціна товару
isArchived	boolean	статус в архіві, 0 - не в архіві, 1 - в архіві (не висвічується клієнту)

Таблиця 3.9 – Опис таблиці Category

Назва поля	Тип даних	Опис
id	ObjectId	ідентифікаційний номер категорії товару
name	string	назва категорії

Таблиця 3.10 – Опис таблиці Order

Назва поля	Тип даних	Опис
id	ObjectId	ідентифікаційний номер товару
userId	ObjectId	ідентифікаційний номер акаунту замовника
items	array[embedded document]	список товарів із кількістю кожного товару у замовленні
orderNumber	string	номер замовлення
total	number	загальна вартість замовлення
delivery	embedded document	доставка: метод доставки, місто, відділення, поштомат, адреса
paymentMethod	string	спосіб оплати: накладений платіж, готівка чи картка
status	string	статус замовлення: в обробці, в доставці, завершений, скасований
createdAt	string	дата створення екземпляру колекції

Таблиця 3.11 – Опис таблиці Day

Назва поля	Тип даних	Опис
id	ObjectId	ідентифікаційний номер статусу замовлення
day	string	назва дня тижня
isOpen	boolean	статус відкритий: 0 – закритий в цей день, 1 – відкритий в цей день
workStart	string	час початку роботи у форматі «ГГ:ХХ»
workEnd	string	час завершення роботи у форматі «ГГ:ХХ»

Продовження таблиці 3.11:

Назва поля	Тип даних	Опис
hasLunchBreak	boolean	статус обіду: 0 – обіду немає в цей день, 1 – обід є в цей день
lunchStart	string	час початку обіду у форматі «ГГ:ХХ»
lunchEnd	string	час завершення обіду у форматі «ГГ:ХХ»
isSurgeryDay	boolean	статус операцій: 0 – не операційний день, 1 – операційний день
surgeryStart	string	час початку операцій у форматі «ГГ:ХХ»
surgeryEnd	string	час завершення операцій у форматі «ГГ:ХХ»

Таблиця 3.12 – Опис таблиці Schedule

Назва поля	Тип даних	Опис
id	ObjectId	ідентифікаційний номер розкладу
weekSchedule	array[ObjectId]	список днів в розкладі
updatedAt	date	дата останнього оновлення

Таблиця 3.13 – Опис таблиці Recipe

Назва поля	Тип даних	Опис
id	ObjectId	ідентифікаційний номер рецепту
recipeNumber	string	номер рецепту
userId	ObjectId	ідентифікаційний номер користувача, якому видано рецепт
petId	ObjectId	ідентифікаційний номер картки пацієнта, на яку виписано рецепт
doctorId	ObjectId	ідентифікаційний номер лікаря
products	array[embedded document]	товари, на які виписано рецепт, з кількістю кожного товару
issuedAt	date	дата видачі
validUntil	date	дата, до якої дійсний рецепт

Відповідно до опису сутностей бази даних наведено ER-діаграму сутностей БД у графічному матеріалі, креслення 2.

Опис утиліт, бібліотек та іншого стороннього програмного забезпечення, що використовується у розробці наведено в таблиці 3.13.

Таблиця 3.14 – Опис утиліт

№ п/п	Назва утиліти	Опис застосування
1	WebStorm	Головне середовище розробки програмного забезпечення клієнтської та серверної частин дипломної роботи.
2	Postman	Програмне забезпечення необхідне для тестування rest запитів. Використовувалось для тестування API інтерфейсів, та клієнтських запитів.
3	MongoDB Atlas	Вебзастосунок, який надає легкий графічний інтерфейс для доступу до бази даних.
4	Figma	Програмне забезпечення для створення прототипів та дизайнів інтерфейсу застосунку.
5	Render	Хмарна платформа для розгортання бекенд-серверу. Забезпечує автоматичний деплой та CI/CD для серверної частини.
6	npm	Менеджер пакетів Node.js, використовувався для встановлення бібліотек та залежностей проєкту.
7	Draw.io (diagrams.net)	Сервіс для створення UML-діаграм, BPMN-моделей та архітектурних діаграм за C4-моделлю. Використовувався для візуалізації структури системи.

Тексти програмного коду наведені в окремому документі «Текст програми КПІ.ІП-1204.045440.03.12».

3.4 Аналіз безпеки даних

У процесі розробки вебзастосунку PetHealth було враховано ключові аспекти безпеки для забезпечення конфіденційності, цілісності та доступності даних пацієнтів, користувачів та адміністраторів. Основна увага приділялася мінімізації ризиків, пов'язаних із витоком персональної інформації та несанкціонованим доступом до системи.

а) Управління вразливостями (Vulnerability Management)

Ідентифіковані ризики:

1) введення шкідливих даних (SQL/NoSQL-injection, XSS);
2) вразливі сторонні бібліотеки (npm-пакети);
3) відкрите API без авторизації.

Прийняті рішення:

1) валідація даних на боці клієнта та сервера;
2) регулярне оновлення залежностей (npm audit, npm-check-updates);
3) захист API через перевірку JWT-токена на кожному приватному маршруті.

б) Тестування безпеки (Security Testing).

1) перевірка роботи API-запитів, які потребують авторизації, щоб переконатися, що доступ мають лише авторизовані користувачі;
2) перевірка обхідних шляхів доступу до ресурсів без токену;
3) використання Postman для емуляції некоректних запитів та спроб доступу до інших ролей.

в) Безпека в хмарі (Security for Container and Cloud).

1) використання HTTPS для усіх запитів;
2) ротація та приховання ключів у .env, зберігання їх у секретах хмарного хостингу;
3) закриття неактивних портів; вхід лише через зазначений frontend.

Застосунок спроєктований з урахуванням сучасних вимог до безпеки даних. Основні принципи: мінімізація прав доступу, валідація даних, контроль автентичності та ізоляція середовища.

Висновки до розділу

В розділі було описано основні етапи реалізації програмного забезпечення PetHealth. Зокрема, реалізовано ключові алгоритми: пошук і фільтрація даних, формування розкладу без конфліктів, оновлення залишків на складі.

Представлено структуру бази даних MongoDB, в якій використано як посилання, так і вкладені документи, що забезпечує гнучку роботу з даними. Розглянуто логіку зв’язків між сутностями.

Використано сучасний стек технологій: React, Node.js, Express.js, MongoDB, а також сторонні утиліти: Postman, MongoDB Atlas, Render, Draw.io, які допомогли в побудові, тестуванні та візуалізації системи.

Розділ охоплює конструювання основної логіки системи, підготовку архітектури та інструментів, що забезпечують коректну, стабільну та масштабовану роботу вебзастосунку.

4 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Аналіз якості ПЗ

Метриками для оцінки якості ПЗ обрано наступні:

- надійність;
- продуктивність;
- зручність використання;
- безпека;
- модифікованість та масштабованість;
- кількість функціональних рядків коду;
- цикломатична складність.

Надійність (Reliability). У реалізованому вебзастосунку перевірено, що система не допускає створення конфліктних записів, коректно обробляє помилки введення, забезпечує валідацію форм як на клієнтській, так і на серверній стороні.

Продуктивність (Performance): Час відгуку інтерфейсу в межах < 200 мс для базових операцій (наприклад, відкриття картки пацієнта, додавання запису), що відповідає очікуваним нефункціональним вимогам. Також реалізовано пагінацію та фільтрацію для зменшення навантаження при роботі з великими масивами даних.

Зручність використання (Usability). Інтерфейс адаптивний і логічно структурований. Користувач може легко знайти потрібну інформацію та виконати дію (наприклад, записатися на прийом чи змінити дані тварини). Використання інтуїтивно зрозумілих елементів (селектори, вкладки, підтвердження дій) підвищує зручність роботи з системою.

Безпека (Security). Реалізовано механізми авторизації та автентифікації. Користувачі з різними ролями мають доступ лише до дозволеного функціоналу. Підтвердження email-коду при реєстрації та авторизації, а також

обмеження дій без авторизації дозволяють уникнути несанкціонованого доступу.

Модифікованість та масштабованість (Maintainability & Scalability). Архітектура застосунку побудована на основі REST API з розділенням логіки між фронтендом (React) і бекендом (Express, MongoDB). Код розділено на модулі, що спрощує тестування та подальшу розробку. Нові функціональні блоки (наприклад, онлайн-аптека) інтегруються без необхідності значної переробки існуючої системи.

Кількість функціональних рядків коду (LOC). Усі файли розробки було поділено на три основні частини: .js-файли, .jsx-файли та .scss-файли. За даними плагіну «Statistic» [20] кількість функціональних рядків коду кожного з типів становить:

.js: 2708 (всього рядків) – 2294 (LOC)

.jsx: 5128 (всього рядків) – 4696 (LOC)

.scss: 1983 (всього рядків) – 1815 (LOC)

Результати демонструють, що код розробленого застосунку є чітко структурованим, має високу складність та функціональність клієнтської частини та високий відсоток корисного коду.

Цикломатична складність. За допомогою програмного засобу Lizard [21] отримаємо значення середньої цикломатичної складності окремо для клієнтської (рис. 4.1) та серверної (рис. 4.2) частин застосунку:

Клієнт:

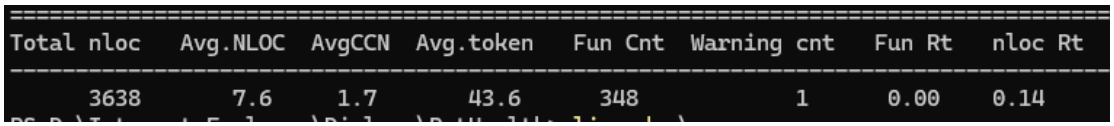
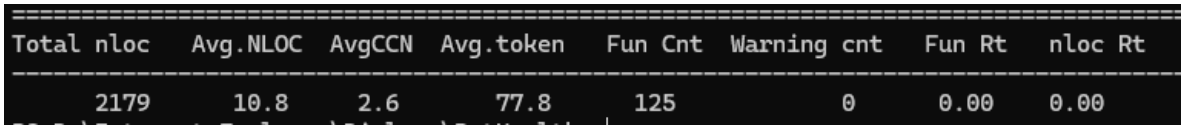


Рисунок 4.1 – Значення параметрів після аналізу коду клієнтської частини

Сервер:



Total nloc	Avg.NLOC	AvgCCN	Avg.token	Fun Cnt	Warning cnt	Fun Rt	nloc Rt
2179	10.8	2.6	77.8	125	0	0.00	0.00

Рисунок 4.2 - – Значення параметрів після аналізу коду серверної частини

З результатів можна зробити висновок, що код фронтенду має дуже низьку цикломатичну складність (≤2) та одну функцію з надмірною складністю; код бекенду має достатньо низьку цикломатичну складність (≈2), нормальну довжину функцій та не має жодної функції з підвищеною складністю.

4.2 Опис процесів тестування

Тестування виконується згідно документу «Програма та методика тестування КПІ.ІП-1204.045440.04.51».

Було виконане мануальне тестування програмного забезпечення. Цей метод полягає в послідовному виконанні сценаріїв користувача вручну (click-through testing) з метою перевірки відповідності поведінки системи очікуваним результатам. Було перевірено такі функції:

– Реєстрація та авторизація користувача через email + код;
– Створення карток пацієнтів;
– Запис на прийом, перевірка доступних слотів, блокування Redis;
– Робота з прийомами: редагування інформації, додавання рецептів;
– Створення замовлень, перевірка наявності рецептів;
– Управління товарами в аптеці;
– Перевірка конфліктів і доступності слотів (в тому числі після зміни графіку клініки).

Опис відповідних тестів наведено у таблицях 4.1 – 4.13:

Таблиця 4.1 – Тест 1.1 Реєстрація користувача

Початковий стан системи	Користувач знаходиться на сторінці реєстрації
Вхідні дані	Електронна пошта та номер телефону (коректні, ще не зареєстровані)
Опис проведення тесту	Користувач вводить email у форму та натискає кнопку «Отримати код». На пошту надсилається шестизначний код підтвердження. Користувач вводить отриманий код у відповідне поле та натискає кнопку підтвердження.
Очікуваний результат	Користувач реєструється, зберігається у базі з email, створюється сесія або токен авторизації, відбувається перенаправлення на головну сторінку
Фактичний результат	Збігається з очікуваним.

Таблиця 4.2 – Тест 1.2 Авторизація користувача

Початковий стан системи	Користувач знаходиться на сторінці авторизації
Вхідні дані	Коректний email та код, отриманий на пошту
Опис проведення тесту	У поля вводяться правильна електронна пошта та код підтвердження, який було надіслано на пошту користувача після запиту. Після цього натискається кнопка підтвердження авторизації.
Очікуваний результат	Авторизація проходить успішно, видається JWT-токен і користувача перенаправляє на головну сторінку.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.3 – Тест 1.3 Створення картки пацієнта

Початковий стан системи	Користувач має роль адміністратора і відкрив вкладку «Картки пацієнтів»

Продовження таблиці 4.3:

Вхідні дані	Ім’я тварини, вид тварини, порода, дата народження, власник
Опис проведення тесту	Користувач заповнює форму додавання картки пацієнта, обирає тварину, власника, натискає «Зберегти».
Очікуваний результат	Картку створено, вона з’являється у списку пацієнтів.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.4 – Тест 1.4 Запис на прийом

Початковий стан системи	Користувач авторизований, має щонайменше одну картку тварини
Вхідні дані	Обрана тварина, процедура, дата прийому, слот, опис проблеми (опціонально)
Опис проведення тесту	Користувач відкриває сторінку запису, обирає значення у всіх полях та натискає кнопку підтвердження запису.
Очікуваний результат	Запис зберігається, статус «запланований», дані доступні у вкладці «Записи».
Фактичний результат	Збігається з очікуваним.

Таблиця 4.5 – Тест 1.5 Створення рецепта

Початковий стан системи	Лікар редагує прийом і натискає «Створити рецепт»
Вхідні дані	Обрана тварина, список медикаментів (які потребують рецепта)
Опис проведення тесту	Лікар у формі додає медикаменти з відповідного списку, вводить їх кількість, натискає «Створити».

Продовження таблиці 4.5:

Очікуваний результат	Рецепт зберігається в базі, додається до прийому, присвоюється унікальний номер.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.6 – Тест 1.6 Оформлення замовлення

Початковий стан системи	Користувач має додані товари у кошику
Вхідні дані	Вміст кошика, обраний спосіб доставки, оплати, рецепт (якщо потрібен)
Опис проведення тесту	Користувач переходить до оформлення, обирає доставку (самовивіз, відділення НП, поштомат НП, доставка НП на адресу), тип оплати (при отриманні в клініці, накладений платіж, карткою чи готівкою кур’єру) додає рецепт за потреби, натискає «Оформити замовлення».
Очікуваний результат	Замовлення зберігається, створюється унікальний номер, кількість товарів у рецепті зменшується на кількість замовлених товарів, сторінка показує повідомлення про успішне оформлення, замовлення з’являється на сторінці замовлень зі статусом «в обробці».
Фактичний результат	Збігається з очікуваним.

Таблиця 4.7 – Тест 1.7 Редагування прийому адміністратором/лікарем

Початковий стан системи	Адміністратор/лікар відкрив картку пацієнта та обрав прийом або обрав прийом у загальному списку прийомів і натиснув «Детальніше» → «Редагувати»
Вхідні дані	Нові значення діагнозу, температури, ваги, стану, призначень, за потреби додаються використані під час прийому медикаменти чи ветеринарні товари.

Продовження таблиці 4.7:

Опис проведення тесту	Заповнюються медичні поля прийому, натискається кнопка «Зберегти»
Очікуваний результат	Дані прийому оновлено, статус змінено на обраний адміністратором/лікарем.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.8 – Тест 1.8 Відображення заблокованих слотів у реальному часі

Початковий стан системи	Користувач (будь-який авторизований) перебуває на сторінці запису
Вхідні дані	Обрана дата, процедура, інший користувач паралельно блокує слот
Опис проведення тесту	Система отримує оновлення через Socket або перезавантаження – слот стає недоступним (видаляється із доступних).
Очікуваний результат	Заблокований слот не можна обрати; якщо користувачі одночасно підтверджують запис на однаковий слот, то одному вдається це зробити, а іншим спливає повідомлення про неможливість бронювання обраного слоту.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.9 – Тест 1.9 Перевірка наявності рецепта при оформленні замовлення

Початковий стан системи	У кошику є медикамент, що вимагає рецепт
Вхідні дані	Вибраний рецепт, що містить відповідний препарат

Продовження таблиці 4.9:

Опис проведення тесту	Система перевіряє, чи співпадають препарати з рецепта і кошика, а також їх кількість
Очікуваний результат	Якщо медикамент є в рецепті у необхідній кількості, замовлення оформлюється; інакше користувач не може створити замовлення.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.10 – Тест 1.10 Перевірка скасування прийому при зміні графіка

Початковий стан системи	Є заплановані прийоми в день, який адміністратор робить неробочим
Вхідні дані	Графік змінено, день зроблено вихідним
Опис проведення тесту	Всі прийоми в цей день автоматично отримують статус «скасований»
Очікуваний результат	Кожен запис отримує статус «скасований», надсилається повідомлення на email користувачам про скасування прийому.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.11 – Тест 1.11 Видалення медикаменту з рецепта після покупки

Початковий стан системи	У рецепті є 2 медикаменти, замовляється лише 1
Вхідні дані	Оформлено замовлення на 1 медикамент із рецепта
Опис проведення тесту	Після оформлення цей медикамент видаляється з рецепта, другий лишається.

Продовження таблиці 4.11:

Очікуваний результат	В рецепті залишається лише невикористаний медикамент.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.12 – Тест 1.12 Невалідний код підтвердження при реєстрації/авторизації

Початковий стан системи	Користувач ввів email, отримав код
Вхідні дані	Введено неправильний або прострочений код
Опис проведення тесту	Користувач вводить не той код або вводить його через 5+ хв.
Очікуваний результат	Система показує помилку: «Невірний або прострочений код», реєстрація/авторизація не завершується
Фактичний результат	Збігається з очікуваним.

Таблиця 4.13 – Тест 1.13 Додавання або редагування товару

Початковий стан системи	Адміністратор знаходиться на вкладці «Аптека»
Вхідні дані	Назва, опис, категорія, вид тварини, одиниця, ціна, кількість на складі, чекбокс «за рецептом»
Опис проведення тесту	Адміністратор заповнює форму для нового товару або редагує існуючий товар за тією ж формою, натискає «Зберегти».
Очікуваний результат	Товар з’являється або оновлюється у списку, збережений у базі.
Фактичний результат	Збігається з очікуваним.

4.3 Опис контрольного прикладу

Контрольний приклад демонструє процес запису клієнта на прийом до ветеринарної клініки через вебзастосунок. Цей функціонал є ключовим, адже саме він забезпечує взаємодію клієнта з клінікою та доступ до медичних послуг. У прикладі враховано всі можливі варіанти вибору, перевірки та обробки даних, включно з особливими ситуаціями.

Крок 1. Вибір тварини

Користувач обирає тварину зі списку своїх активних пацієнтів (без архівних) (рис. 4.5).

Особливості:

Якщо у користувача немає жодної тварини, він повинен створити картку тварини у користувацькому меню (рис. 4.3-4.4).

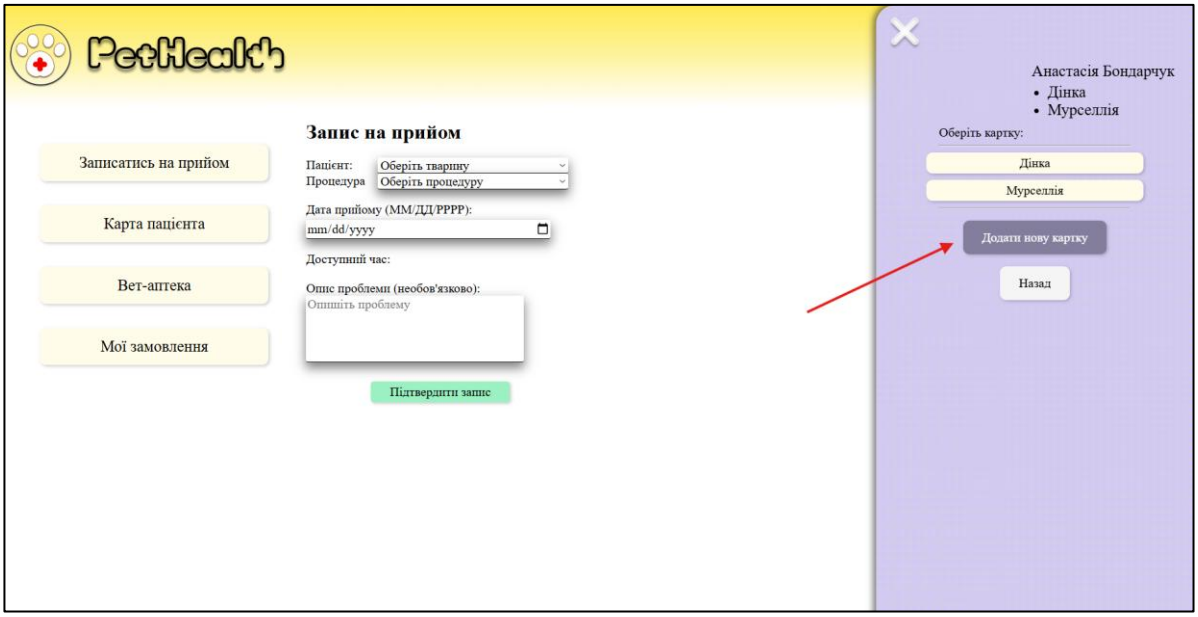


Рисунок 4.3 – Додавання картки пацієнта

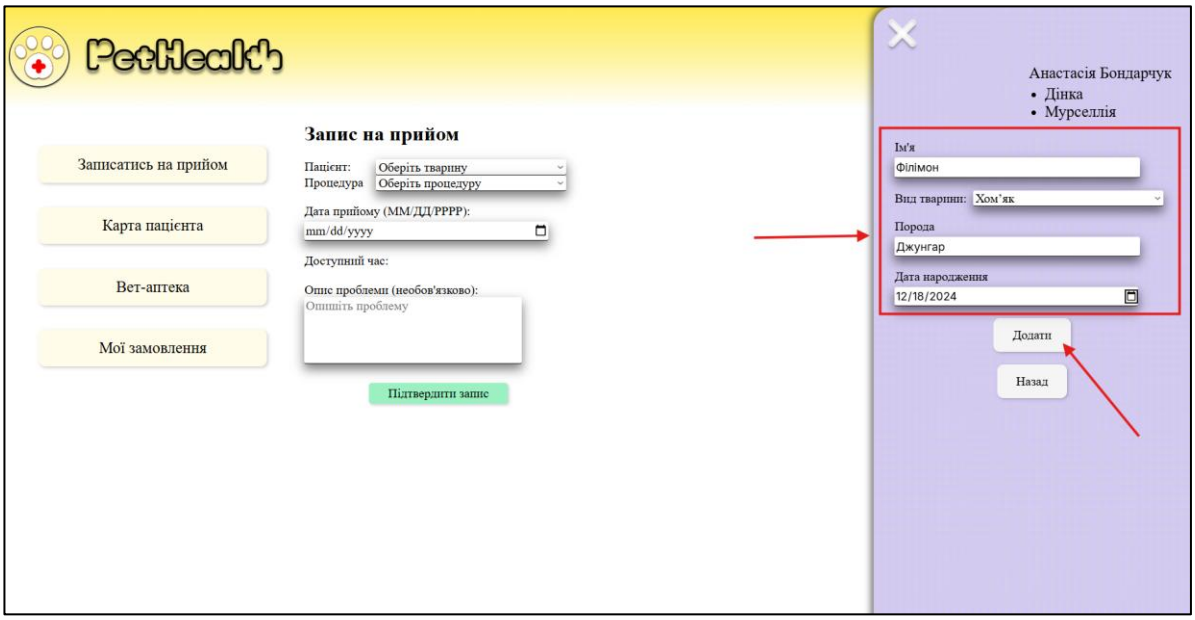


Рисунок 4.4 – Внесення даних про тварину

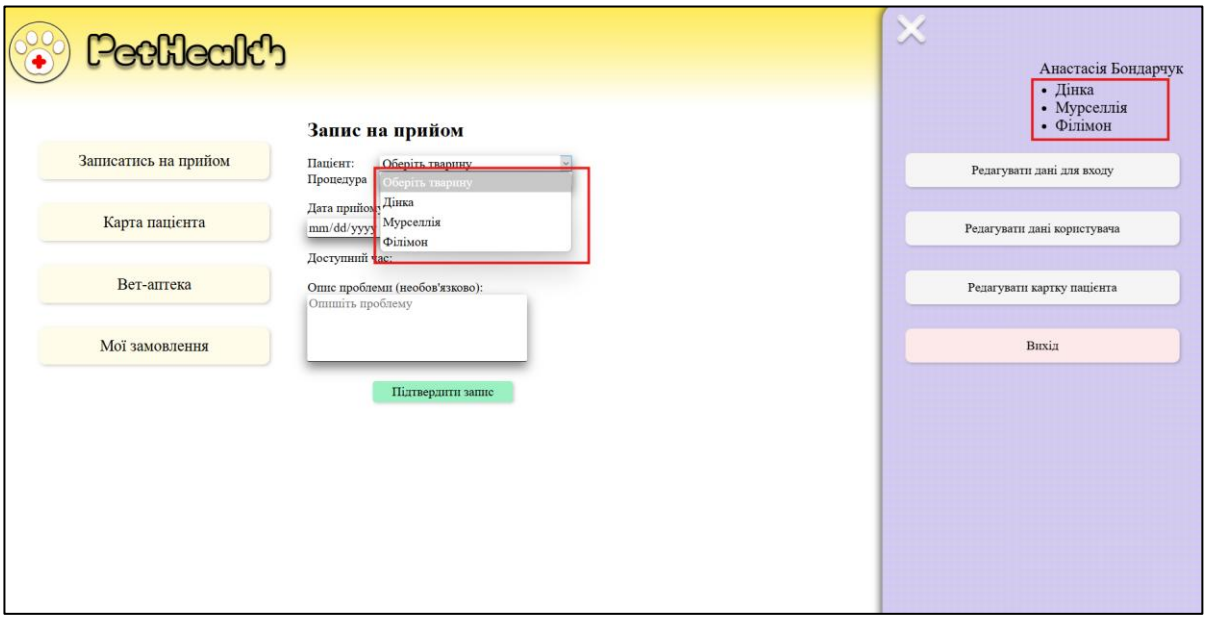


Рисунок 4.5 – Картка пацієнта додана

Крок 2. Вибір процедури

Користувач обирає процедуру із доступного списку (тільки ті, де дозволений самозапис) (рис. 4.6).

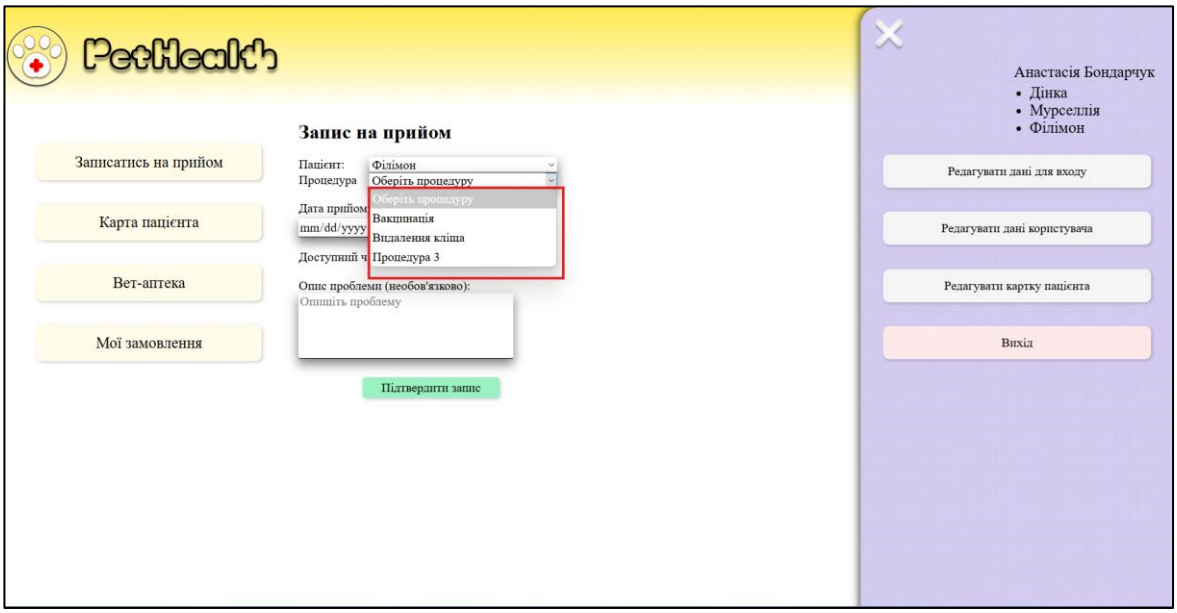


Рисунок 4.6 – Вибір процедури зі списку

Крок 3. Вибір дати

Календар дозволяє обрати дату прийому (рис. 4.7).

Особливості:

Вибір минулої дати заблокований.

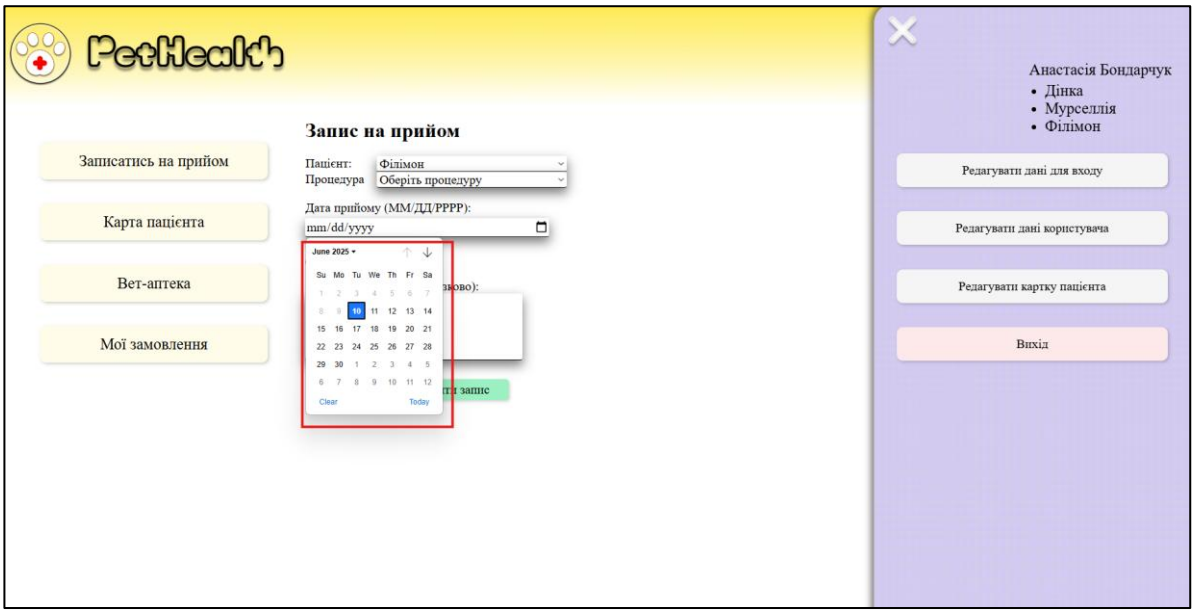


Рисунок 4.7 – Вибір дати прийому

Крок 4. Вибір часу

Система автоматично генерує список вільних слотів на обрану дату відповідно до тривалості процедури, вже зайнятого часу, графіку роботи клініки та поточного часу доби (рис. 4.10).

Особливості:

Якщо всі слоти зайняті або в обраний день клініка не працює – виводиться текст «Немає доступних слотів» (рис. 4.8).

Якщо обрано поточну дату, то генерація слотів відбувається від поточного часу (рис. 4.9).

Якщо одночасно кілька користувачів обирають один і той самий слот, застосовується механізм Redis pre-lock для уникнення конфліктів.

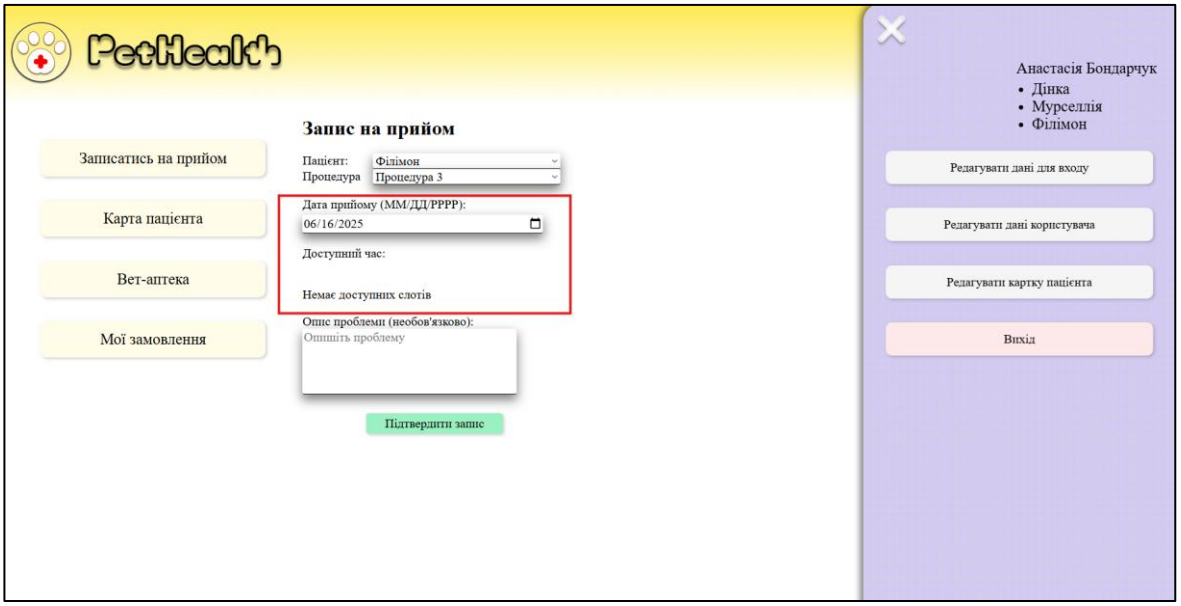


Рисунок 4.8 – Обрано вихідний день клініки

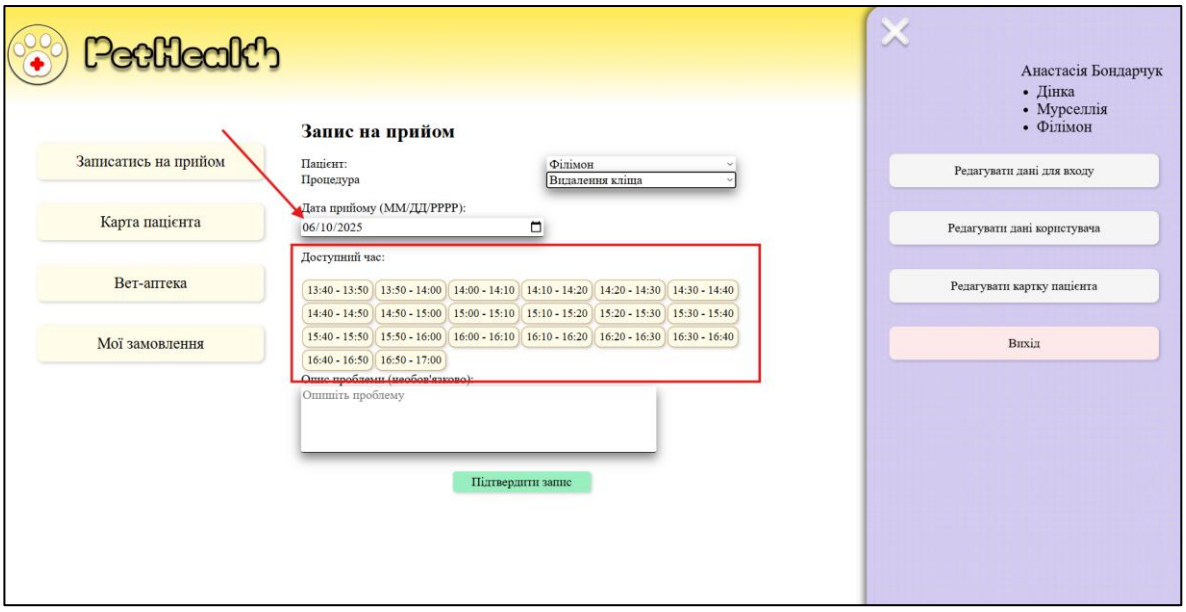


Рисунок 4.9 – Обрано поточну дату для прийому

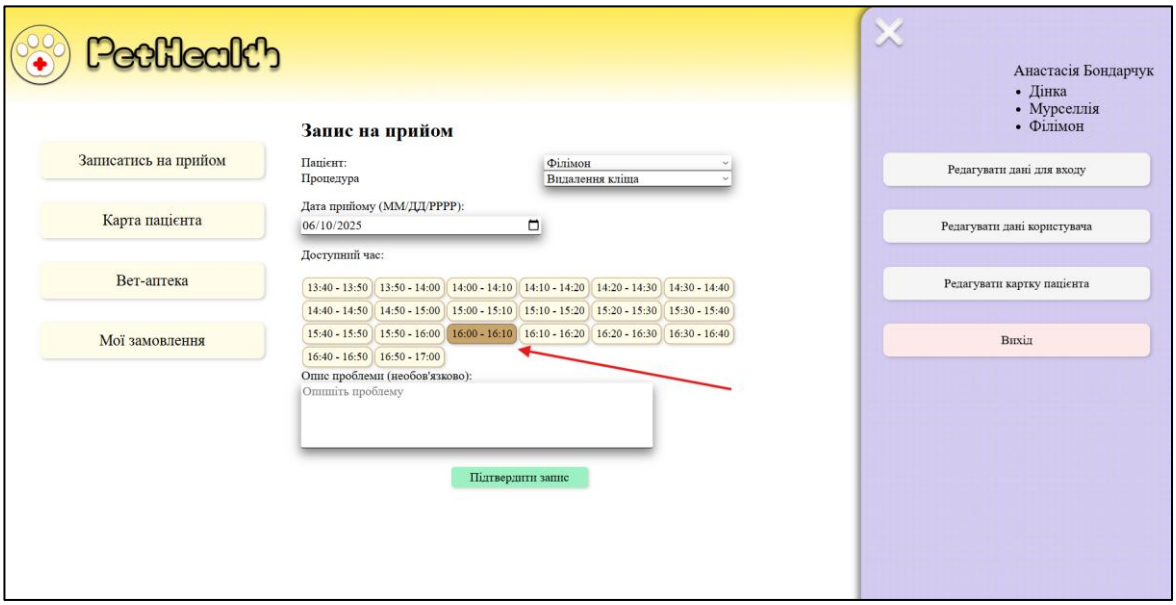


Рисунок 4.10 – Вибір доступного часу

Крок 5. Додавання коментаря

Користувач вводить опис проблеми (необов’язкове поле) (рис. 4.11).

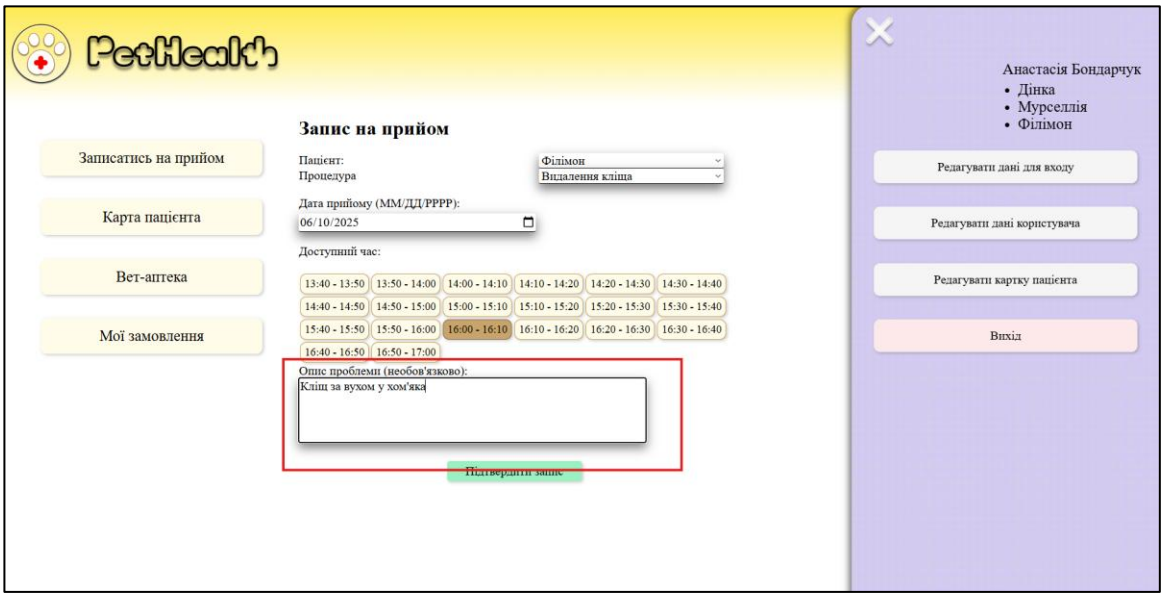


Рисунок 4.11 – Заповнення опису проблеми

Крок 6. Підтвердження запису

Після натискання кнопки «Підтвердити», система перевіряє:

4) актуальність обраного слоту;

5) автентифікацію користувача.

У разі успіху запис створюється, і користувач бачить підтвердження (рис. 4.12). Якщо слот уже зайнятий або виникла помилка – з’являється відповідне повідомлення.

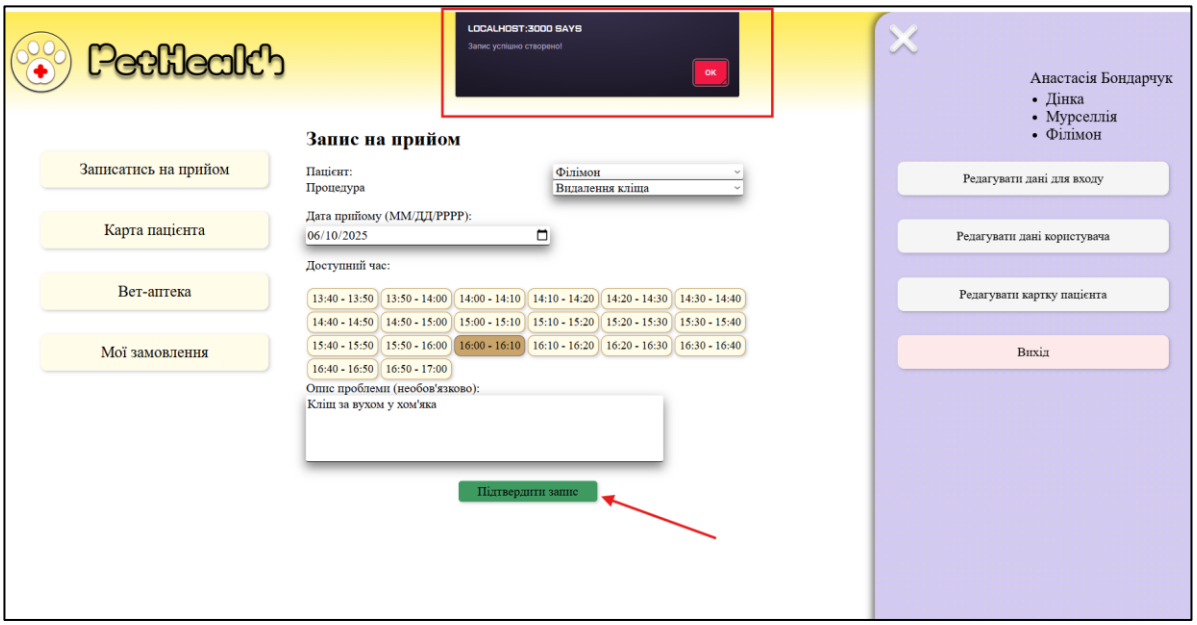


Рисунок 4.12 – Створення запису на прийом

Крок 7. Email-підтвердження

Після створення запису на електронну пошту клієнта надсилається підтвердження з деталями запису (дата, час, тварина, процедура) (рис. 4.13).

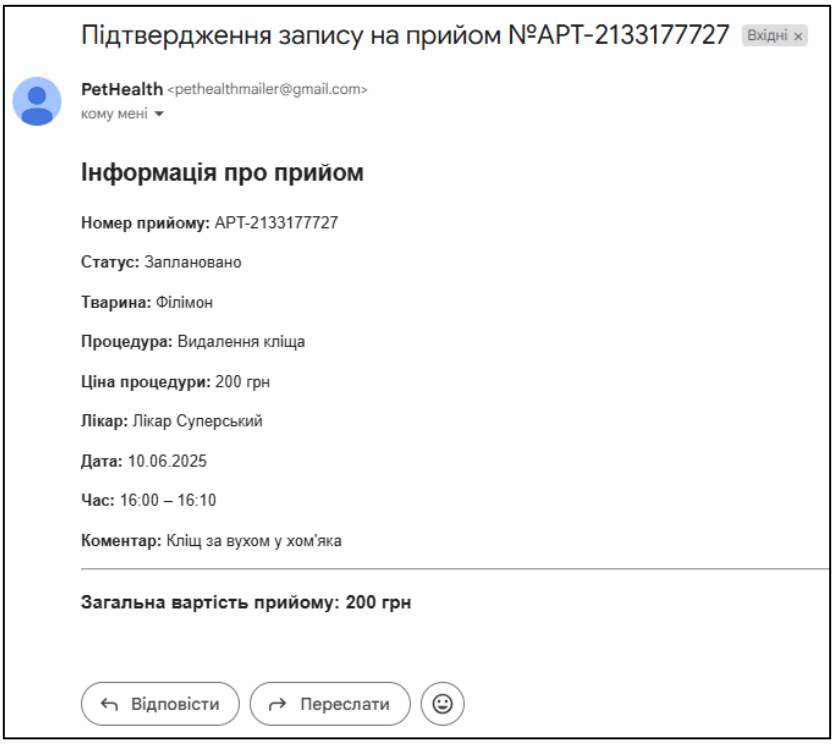


Рисунок 4.13 – Повідомлення на пошті про підтвердження прийому

Крок 8. Перегляд прийому в картці пацієнта

Після успішного створення прийому користувач може побачити прийом/запис у картці пацієнта (рис. 4.14) та переглянути деталі (рис. 4.17).

Особливості:

Якщо статус прийому «Запланований», то прийом висвічується у записах (рис. 4.14).

Якщо статус прийому інакший від «Запланований», то прийом висвічується у вкладці прийомів (рис. 4.15).

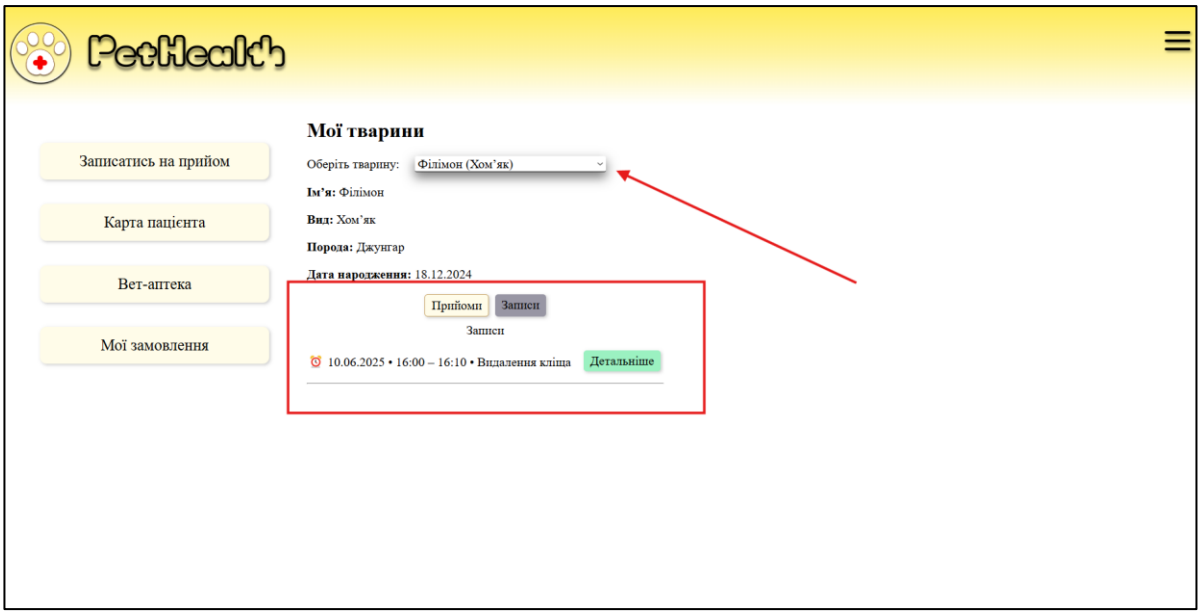


Рисунок 4.14 – Перегляд картки пацієнта (записи)

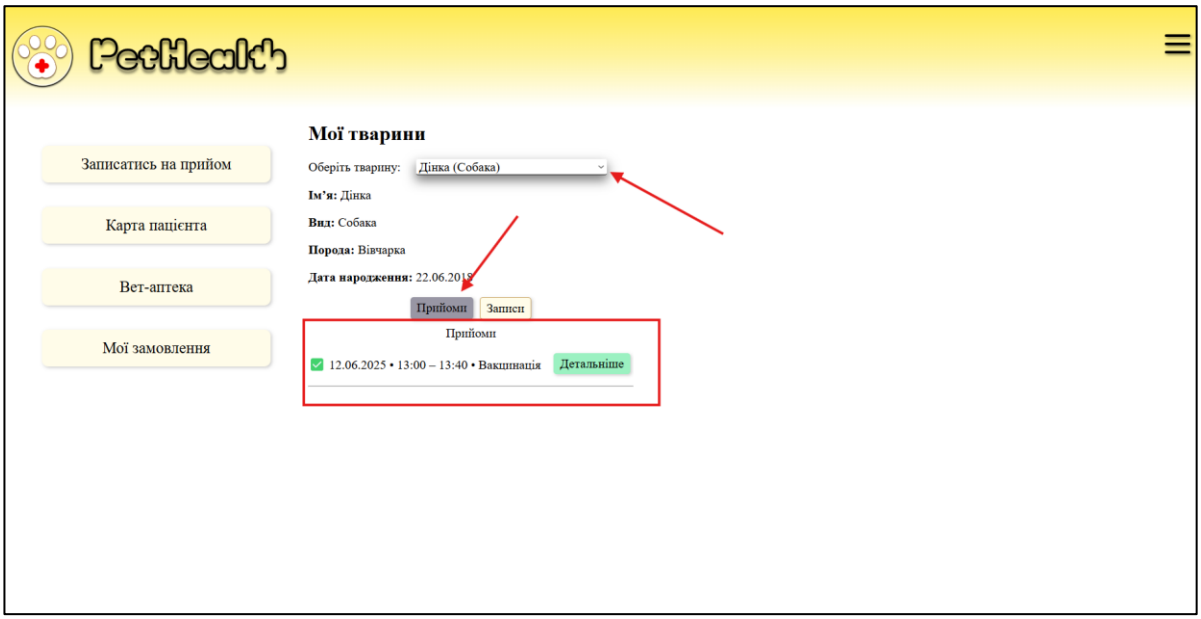


Рисунок 4.15 – Перегляд картки пацієнта (прийоми)

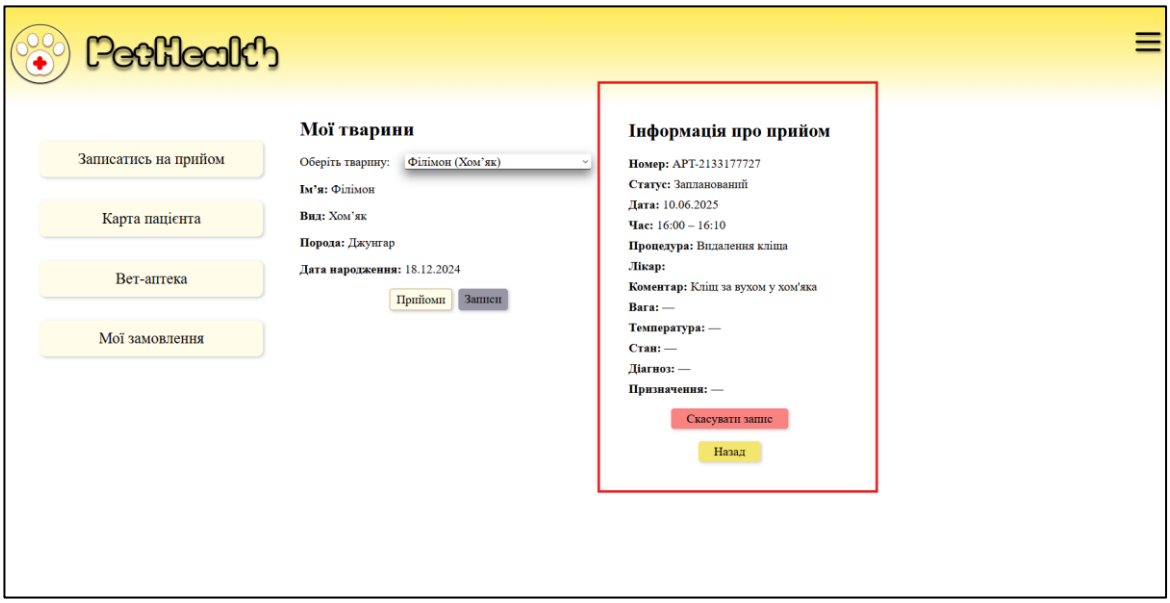


Рисунок 4.16 – Перегляд деталей запису/прийому

Крок 9. Скасування прийому

Користувач може скасувати прийом, якщо він ще запланований (рис. 4.19-4.21).

Особливості:

Скасувати прийом можна не пізніше, ніж за 12 годин до запису. В інакшому випадку користувач отримає повідомлення про неможливість скасування прийому (рис. 4.17).

Якщо прийом вже був розпочатий, скасований або завершений, доступу до скасування користувач не матиме (рис. 4.18).

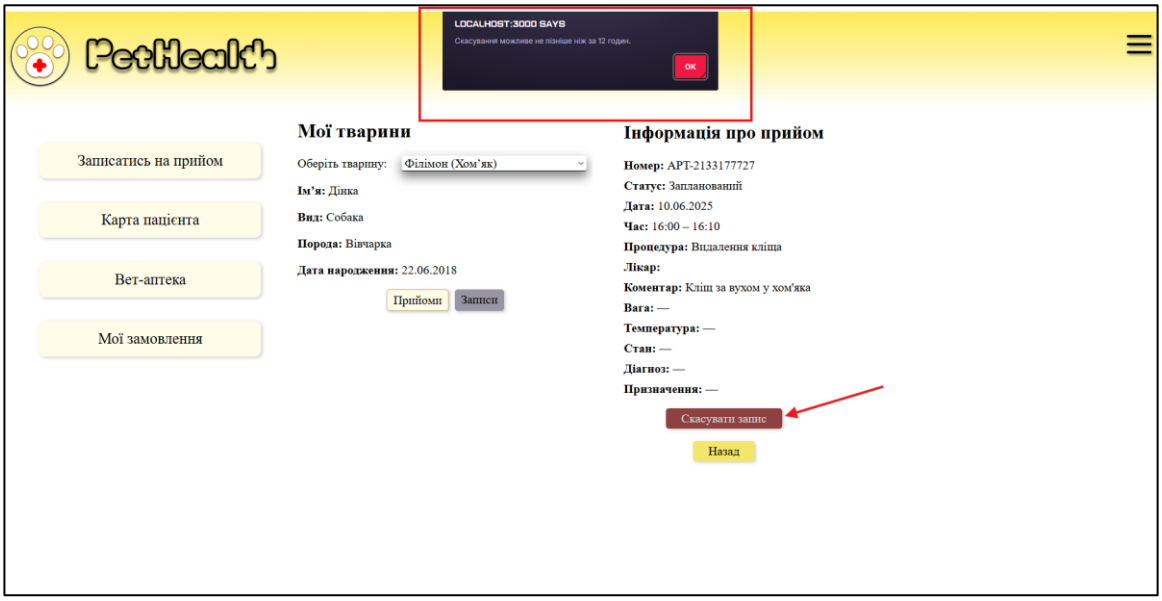


Рисунок 4.17 – Невдала спроба скасування прийому

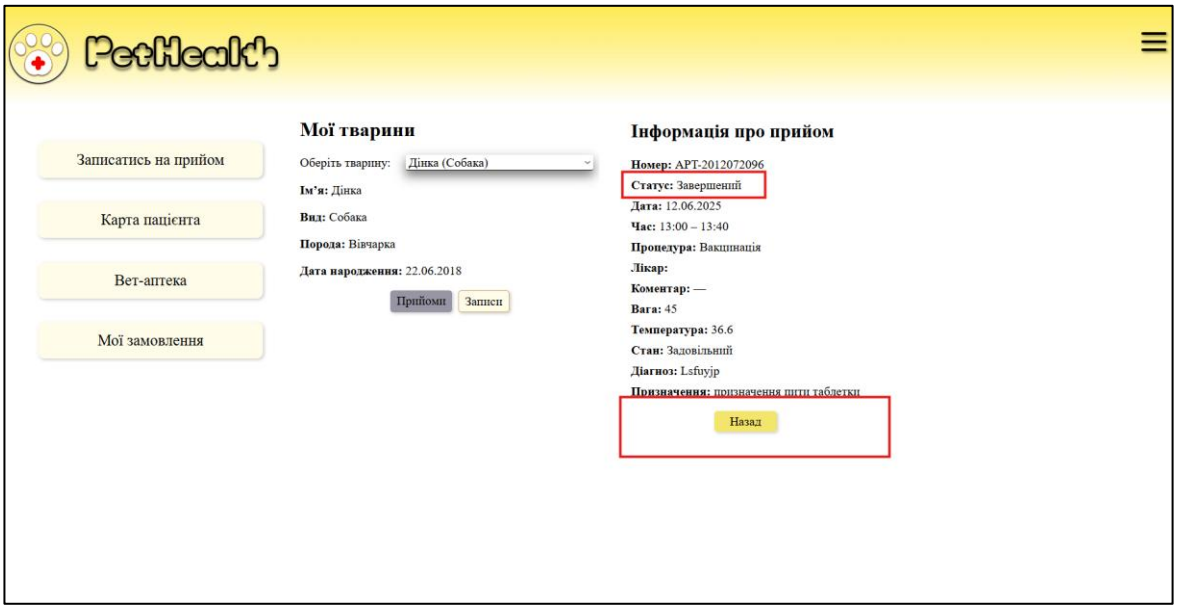


Рисунок 4.18 – Відсутність можливості скасування для прийому, що не є запланованим (записом)

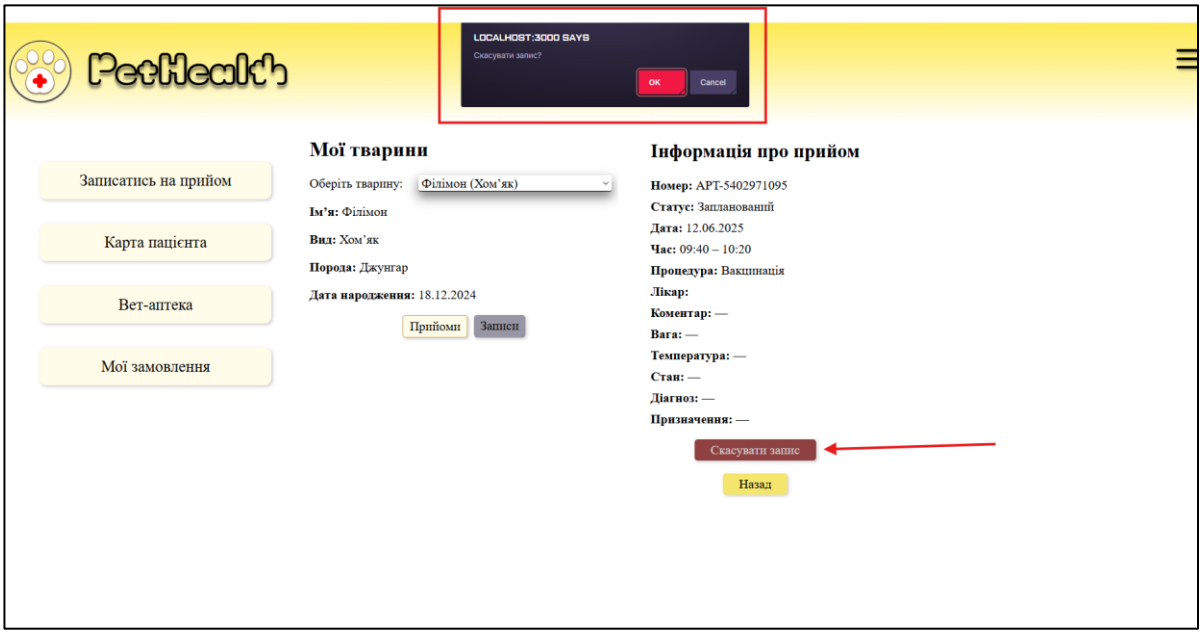


Рисунок 4.19 – Підтвердження скасування запису

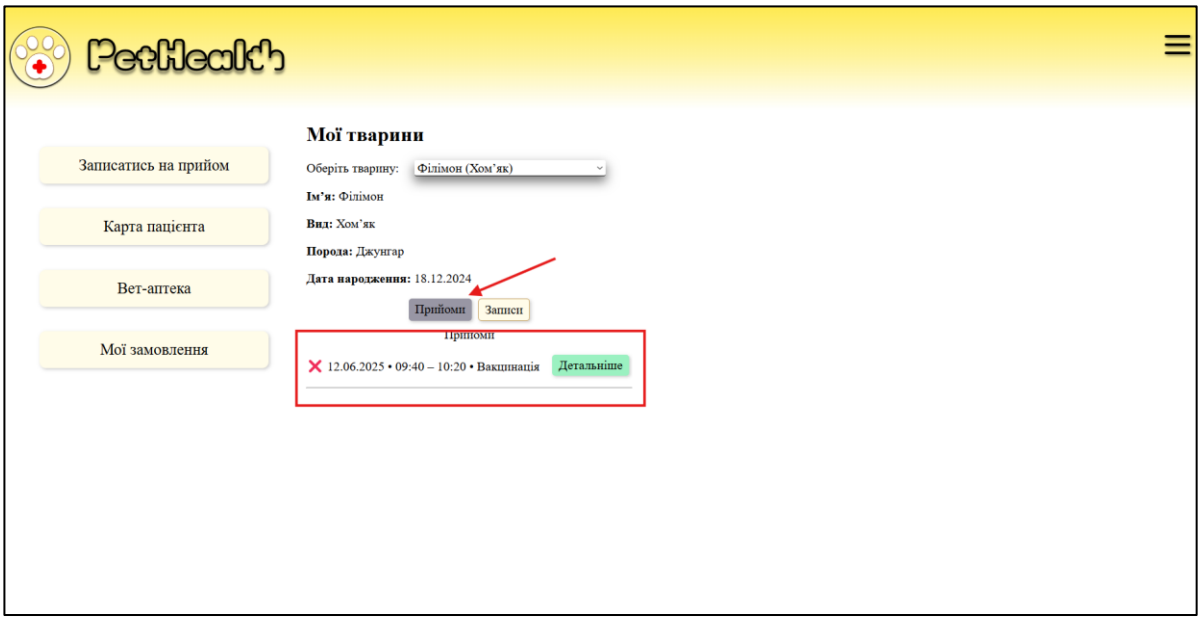


Рисунок 4.20 – Прийом скасовано (перегляд у списку)

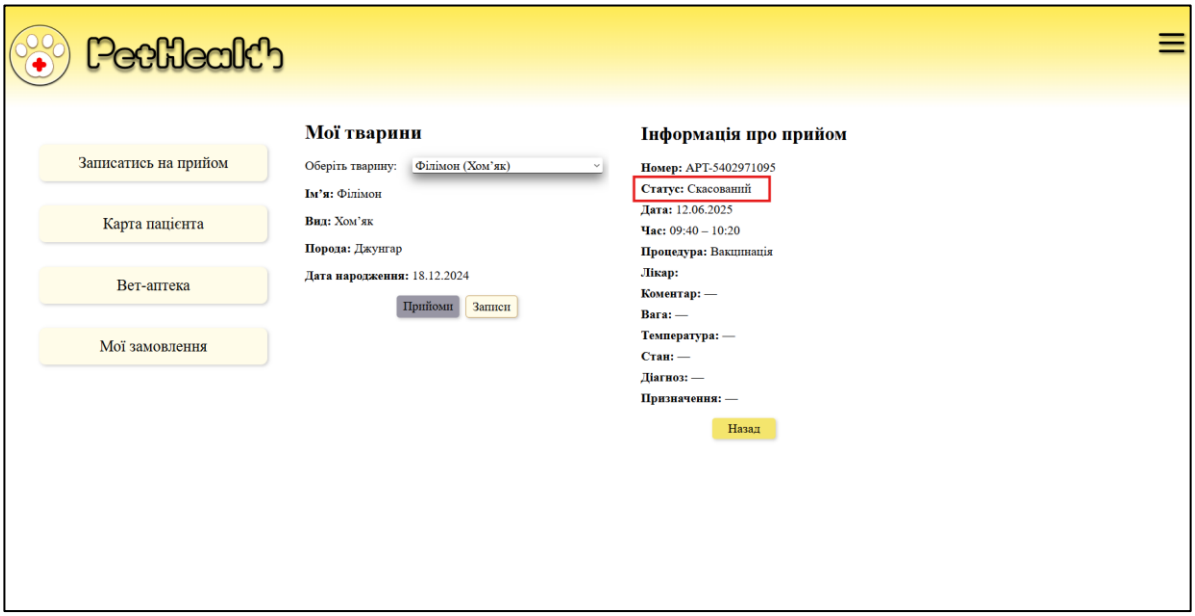


Рисунок 4.21 – Прийом скасовано (перегляд детальної інформації)

Висновки до розділу

В даному розділі було здійснено оцінку якості розробленого програмного забезпечення (вебзастосунку), проведено тестування функціоналу. Для оцінки якості було використано метрики відповідно до моделі ISO/IEC 25010:2023[22]: надійність, продуктивність, зручність використання, безпека, модифікованість та масштабованість. Також досліджено кількість функціональних рядків коду та визначено цикломатичну складність кожної частини застосунку.

Забезпечення описаних нефункціональних вимог було виконано під час розробки вебзастосунку, зокрема імплементовано вирішення конфліктів у записах та замовленнях, валідація даних, інтерфейс є адаптивним та інтуїтивно зрозумілим, безпека забезпечується підтвердженням по email, обмеженням доступу та перевіркою токену при ключових діях. Кількість функціональних рядків та показники цикломатичної складності вказують на значну масштабованість розробки та дотримання правильно структурованої клієнт-серверної архітектури, застосунок має прийнятну складність логіки та хорошу підтримуваність.

Тестування вебзастосунку проводилось шляхом мануального функціонального тестування, під час якого було перевірено коректність роботи розробки в основних функціональних блоках.

В контрольному прикладі наведено покроковий процес запису користувачем на прийом до ветклініки, який є ключовим для даної розробки. У прикладі описано можливі винятки та особливості їх обробки системою.

5 РОЗГОРТАННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

5.1 Розгортання програмного забезпечення

Для розгортання вебзастосунку ветеринарної клініки було обрано модель клієнт-сервер з розділенням на фронтенд та бекенд частини, що взаємодіють через REST API. У якості середовища розгортання використано:

– Серверну частину (бекенд): Node.js (Express), база даних MongoDB Atlas;

– Клієнтську частину (фронтенд): React;

– Середовище розміщення: Render.com;

– Сховище коду: GitHub.

Для локального розгортання застосунку:

а) Розгортання бази даних:

1) Створення акаунту на MongoDB Atlas;

2) Створення кластеру;

3) Створення бази даних vet-clinic та колекцій;

4) Налаштування доступу IP та створення користувача з паролем;

5) Отримання URI-посилання для підключення.

б) Розгортання бекенду:

1) Клонування репозиторію: git clone https://github.com/username/reponame.git

2) Перехід у директорію: cd reponame/server

3) Встановлення залежностей: npm install

4) Створення .env файлу

5) Запуск сервера у продакшн-режимі: npm run dev

в) Розгортання фронтенду:

1) Клонування репозиторію (якщо ще не був клонований): git clone https://github.com/username/reponame.git

2) Перехід у директорію: cd reponame/client

3) Встановлення залежностей: npm install

4) Створення .env файлу з вказанням бекенд-URL

5) Побудова проєкту: npm build

6) Запуск фронтенду: npm start

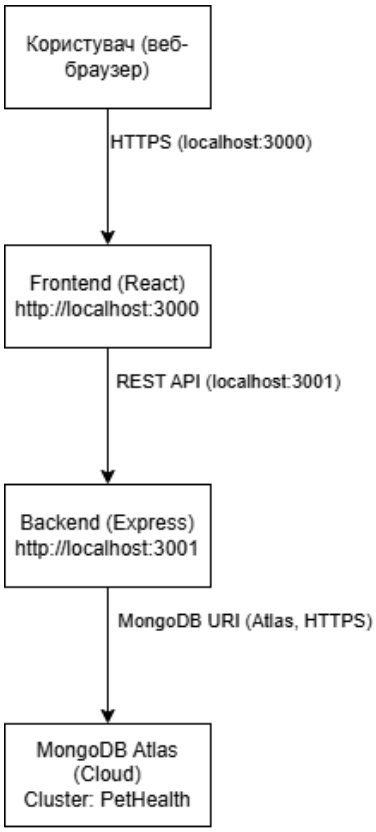


Рисунок 5.1 – Діаграма розгортання (локальна версія)

Для доступу через інтернет (інтеграція з Render.com):

а) Розгортання бази даних аналогічне до локальної версії.

б) Розгортання бекенду:

1) Реєстрація на https://render.com, створення нового вебсервісу, підключення серверної папки у репозиторії GitHub з кодом.

2) Налаштування сервісу (назва, гілка для деплою, команда для збірки npm install, команда для запуску npm run start, середовище Node).

3) Встановлення змінних .env.

4) Після завершення процесу отримуємо URL на кшталт https://name_server.onrender.com.

в) Розгортання фронтенду:

1) Створення нового вебсервісу, підключення клієнтської папки у репозиторії GitHub з кодом.

2) Налаштування сервісу (назва, команда для збірки npm run build, статична директорія файлів build).

3) Після деплою отримуємо публічну адресу https://name_client.onrender.com.

4) Встановити змінні в .env з отриманим URL.

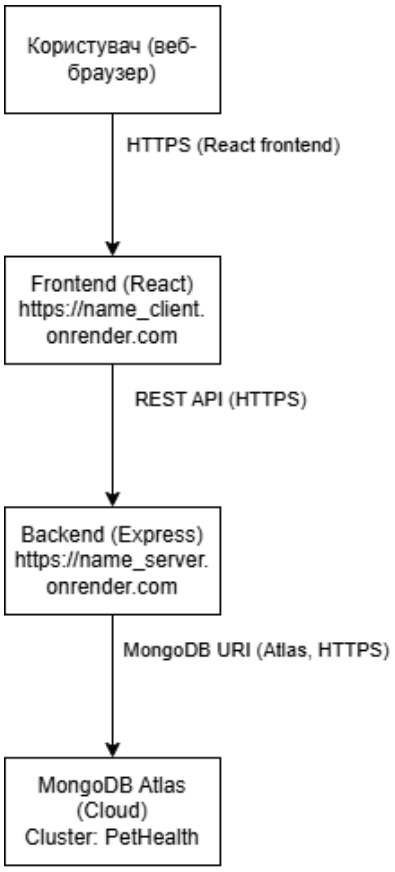


Рисунок 5.2 – Діаграма розгортання (хмарна версія)

5.2 Супровід програмного забезпечення

Інструкція користувача наведена в окремому документі «Керівництво користувача КПІ.ІП-1204.045440.05.34».

Архітектура застосунку побудована з урахуванням можливості масштабування, оновлення окремих модулів без зупинки всього сервісу та оперативного усунення помилок.

Технічна підтримка:

– Зовнішні сервіси: дані зберігаються в хмарній БД, що дозволяє керувати резервним копіюванням та відновленням; розгортання через Render забезпечує автоматичні деплої з GitHub після push, логування помилок та відновлення після збоїв.

– Оновлення коду: код зберігається на GitHub, при внесенні змін в певну гілку, Render автоматично виконує деплой.

– Обробка помилок: у бекенді реалізовано try-catch блоки, обробники помилок та відповіді з http-статусами; на фронтенді передбачено обробку помилок з відповідними повідомленнями користувачу.

– Підтримка інтерфейсу: структура фронтенду дозволяє оновлювати та додавати окремі компоненти без впливу на інші частини програми; структура SCSS-файлів дозволяє підтримувати ізольовані стилі для кожного елементу інтерфейсу.

Висновки до розділу

В розділі було розглянуто процес розгортання та супроводу вебзастосунку для ветеринарної клініки. Розгортання реалізовано за клієнт-серверною інфраструктурою, в якості середовища розміщення онлайн-хостингу використано хмарну платформу Render.com, а для зберігання даних – MongoDB Atlas. Код розміщено на GitHub, що дозволяє централізовано керувати версіями та спрощує оновлення.

Описано два основні способи розгортання: локальний та хмарний. Для кожного з них надано інструкція та наведено діаграми розгортання. Для супроводу ПЗ визначено основні підходи до технічної підтримки, зокрема оновлення системи, обробки помилок та підтримка інтерфейсу.

Розгортання і супровід описані та реалізовані із урахуванням вимог до масштабованості, простоти оновлення, підтримки та стійкості до помилок.

ВИСНОВКИ

У ході виконання дипломного проєкту було розроблено повнофункціональний вебзастосунок для організації роботи ветеринарної клініки. Система реалізує функціонал онлайн-запису на прийом, ведення електронних карток пацієнтів, управління процедурами та прийомами, створення рецептів, оформлення замовлень у вет-аптеці, управління аптекою та замовленнями, адміністрування користувачів та управління графіком роботи клініки.

Поставлена мета – створення сучасного інформаційного рішення для автоматизації основних процесів ветеринарної клініки – досягнута в повному обсязі. Розроблений застосунок забезпечує ефективну взаємодію між клієнтами, лікарями та адміністраторами, відповідає актуальним вимогам до безпеки, масштабованості та зручності використання.

Усі завдання, поставлені в рамках дипломного проєкту, виконано:

– спроєктовано структуру бази даних з урахуванням предметної області;

– реалізовано систему ролей та доступів;

– створено інтерфейс для пацієнтів, лікарів та адміністраторів;

– впроваджено механізми перевірки конфліктів при записі, Redis-блокування слотів;

– додано управління аптекою і товарами та створення замовлень з перевіркою наявності рецептів;

– реалізовано повну адміністративну панель для контролю записів та прийомів, користувачів, карток пацієнтів, процедур та графіку роботи;

– виконано тестування і аналіз якості програмного забезпечення;

– підготовлено систему до хмарного розгортання та обслуговування.

Результати виконання проєкту демонструють доцільність продовження розробки у вибраній предметній області. У майбутньому система може бути доповнена генерацією PDF та Excel звітів, підтримкою оплати онлайн,

аналітичними модулями для моніторингу навантаження клініки, а також багатомовною підтримкою для ширшого кола користувачів.

Таким чином, розроблене рішення є перспективною основою для впровадження сучасної цифрової екосистеми у ветеринарній сфері.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Про Helsi. helsi.me. URL: https://helsi.me/about.
2. Veterinary client communication software built to connect with clients. PetDesk. URL: https://petdesk.com.
3. Veterinary Software - GVET. URL: https://www.vetclinicsoft.com.
4. Про ветеринарну медицину: Закон України від 04.02.2021 №1206-IX: станом на 19.04.2025р. URL: https://zakon.rada.gov.ua/laws/show/1206-20#Text.
5. Автоматизація ветеринарного бізнесу: які програми допомагають покращити роботу клініки?. UKRPULSE. URL: https://ukrpulse.org.ua/2025/novini-kompanij/avtomatyzatsiia-veterynarnoho-biznesu-iaki-prohramy-dopomahaiut-pokrashchyty-robotu-kliniky/.
6. Veterinary management software development. Belitsoft. URL: https://belitsoft.com/veterinary-software.
7. Програма для ветклініки. Програма онлайн-запису та CRM управління бізнесом EasyWeek. URL: https://easyweek.com.ua/solutions/veterinary-clinic.
8. Інноваційна система для автоматизації ветеринарних клінік. CRM для ветеринарних клінік та лабораторій. URL: https://www.clinica-web.ua/vet-clinic/.
9. Програма для ветеринарної клініки – CRM Appointer. Appointer. URL: https://appointer.ua/programa-dlya-vet-kliniki/.
10. Jet Vet. Додаток і програма для ветклініки. URL: https://jet.vet/uk/.
11. MongoDB Documentation Team. Getting Started with MongoDB - Database Manual v8.0 - MongoDB Docs. MongoDB: The World's Leading Modern Database | MongoDB. URL: https://www.mongodb.com/docs/manual/tutorial/getting-started/.
12. Distributed locks with redis. Docs. URL: https://redis.io/docs/latest/develop/use/patterns/distributed-locks/
13. Quick Start – React. React. URL: https://react.dev/learn.

14. Express - Node.js web application framework. URL: https://expressjs.com.

15. JSON web tokens - jwt.io. URL: https://jwt.io.

16. Cloud application platform | Render. URL: https://render.com.

17. GeeksforGeeks. Functional point (FP) analysis - software engineering - GeeksforGeeks. URL: https://www.geeksforgeeks.org/software-engineering-functional-point-fp-analysis/.

18. Вакансії junior full stack developer. Robota.ua. URL: https://robota.ua/zapros/junior-full-stack-developer/ukraine.

19. Home. C4 model. URL: https://c4model.com.

20. Statistic – IntelliJ IDEs Plugin | Marketplace. JetBrains Marketplace. URL: https://plugins.jetbrains.com/plugin/4509-statistic

21. lizard. PyPI. URL: https://pypi.org/project/lizard/

22. ISO/IEC 25010:2023. ISO. URL: https://www.iso.org/standard/78176.html

ДОДАТКИ

ДОДАТОК А ЗВІТ ПОДІБНОСТІ

StrikePlagiarism.com

Дата звіту 6/11/2025
Дата редагування 6/11/2025

Документ прийнятий

Звіт подібності

метадані

Назва організації
National Technical University of Ukraine Igor Sikorskyi Kyiv Politech Institute
Заголовок
ІП-12_Бондарчук_ПЗ
Автор Науковий керівник / Експерт
ІП-12_БондарчукМарченко О.І.
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.

13.55%
13.55%
КП 1

10
Довжина фрази для коефіцієнта подібності 2

9337
Кількість слів

71882
Кількість символів

ДОДАТОК Б ВАРІАНТИ ВИКОРИСТАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Таблиця Б.1 – Варіант використання UC-01

Use case name	Реєстрація в системі
Use case ID	UC-01
Goals	Реєстрація нового користувача в системі
Actors	Гість (неавторизований користувач)
Trigger	Користувач бажає зареєструватися
Pre-conditions	-
Flow of Events	Користувач переходить на сторінку реєстрації. В поля для реєстрації вводяться відповідні дані: пошта користувача, номер телефону, код підтвердження із вказаної пошти. Після заповнення даних та підтвердження з кодом користувач натискає кнопку реєстрації. Після цього з'являється повідомлення про успішну реєстрацію, і користувач перенаправляється на головну сторінку застосунку.
Extension	В випадку введення не коректних даних, кнопка реєстрації стає неактивною. Якщо дані введено некоректно, то спливає повідомлення про це. Якщо користувач отримав код, він має натиснути на кнопку повторного надсилання та ввести повторно надісланий код.
Post-Condition	Створення сторінки користувача, перехід на головну сторінку

Таблиця Б.2 – Варіант використання UC-02

Use case name	Авторизація в системі
Use case ID	UC-02
Goals	Авторизація користувача в системі
Actors	Гість (неавторизований користувач)
Trigger	Користувач бажає авторизуватись
Pre-conditions	Користувач раніше був зареєстрований та має акаунт.
Flow of Events	Користувач переходить на сторінку авторизації. В поля для авторизації вводяться відповідні дані: пошта користувача та код підтвердження із вказаної пошти.

Продовження таблиці Б.2:

Flow of Events	Після заповнення даних та підтвердження з кодом користувач натискає кнопку входу.
Flow of Events	Після цього з’являється повідомлення про успішну авторизацію, і користувач перенаправляється на головну сторінку застосунку.
Extension	В випадку введення не коректних даних, кнопка авторизації стає неактивною. Якщо дані введено некоректно, то спливає повідомлення про це. Якщо користувач отримав код, він має натиснути на кнопку повторного надсилання та ввести повторно надісланий код.
Post-Condition	Отримання доступу до сторінки користувача, перехід на головну сторінку

Таблиця Б.3 – Варіант використання UC-03

Use case name	Вихід із системи
Use case ID	UC-03
Goals	Вихід користувача із системи
Actors	Авторизований користувач (адмін або клієнт)
Trigger	Користувач бажає вийти із системи
Pre-conditions	Користувач раніше увійшов в систему.
Flow of Events	Користувач відкриває бокове меню та обирає кнопку виходу.
Extension	-
Post-Condition	Вихід з акаунту користувача, перехід на сторінку авторизації.

Таблиця Б.4 – Варіант використання UC-04

Use case name	Додавання картки пацієнта
Use case ID	UC-04
Goals	Додавання картки пацієнта власником тварини
Actors	Авторизований користувач (клієнт)
Trigger	Користувач бажає додати картку пацієнта
Pre-conditions	Користувач раніше увійшов в систему.
Flow of Events	Користувач відкриває бокове меню, обирає кнопку редагування інформації про тварину.

Продовження таблиці Б.4:

Flow of Events	У відкритій формі користувач обирає додати тварину, після чого заповнює усі необхідні поля та зберігає дані.
Extension	В випадку введення не коректних даних, кнопка збереження стає неактивною. Якщо дані введено некоректно, то спливає повідомлення про це.
Post-Condition	Створення нової картки пацієнта

Таблиця Б.5 – Варіант використання UC-05

Use case name	Редагування картки пацієнта
Use case ID	UC-05
Goals	Редагування картки пацієнта власником тварини
Actors	Авторизований користувач (клієнт)
Trigger	Користувач бажає редагувати картку пацієнта
Pre-conditions	Картка пацієнта до цього була створена.
Flow of Events	Користувач відкриває бокове меню, обирає кнопку редагування інформації про тварину. У відкритій формі користувач обирає тварину для редагування, після чого редагує відповідні поля та зберігає дані.
Extension	В випадку введення не коректних даних, кнопка збереження стає неактивною. . Якщо дані введено некоректно, то спливає повідомлення про це.
Post-Condition	Оновлені дані картки пацієнта

Таблиця Б.6 – Варіант використання UC-06

Use case name	Перегляд картки пацієнта
Use case ID	UC-06
Goals	Перегляд картки пацієнта власником тварини
Actors	Авторизований користувач (клієнт)
Trigger	Користувач бажає переглянути картку пацієнта
Pre-conditions	Картка пацієнта до цього була створена.
Flow of Events	Користувач у лівому меню обирає вкладку картки пацієнта.

Продовження таблиці Б.6:

Flow of Events	У відкритій формі користувач обирає тварину, після чого відкривається картка пацієнта. Користувач може переглядати інформацію про тварину, прийоми, записи.
Extension	-
Post-Condition	-

Таблиця Б.7 – Варіант використання UC-07

Use case name	Редагування даних для входу
Use case ID	UC-07
Goals	Редагування даних користувача для входу в систему
Actors	Авторизований користувач (клієнт або адмін)
Trigger	Користувач бажає змінити дані для входу в систему
Pre-conditions	Користувач раніше увійшов в систему.
Flow of Events	Користувач відкриває бокове меню, обирає кнопку редагування даних для входу. У відкритій формі користувач змінює дані для входу в систему, після чого отримує підтвердження на пошту та вводить отриманий код в поле. Далі користувач натискає на кнопку збереження даних.
Extension	В випадку введення не коректних даних, кнопка збереження стає неактивною. . Якщо дані введено некоректно, то спливає повідомлення про це.
Post-Condition	Оновлені дані користувача для входу в систему.

Таблиця Б.8 – Варіант використання UC-08

Use case name	Запис на прийом
Use case ID	UC-08
Goals	Запис на прийом в клініку
Actors	Авторизований користувач (клієнт)
Trigger	Користувач бажає записатись на прийом до ветеринара
Pre-conditions	Користувач раніше увійшов в систему, картка пацієнта була створена.
Flow of Events	Користувач у лівому меню обирає вкладку запису на прийом.

Продовження таблиці Б.8:

Flow of Events	У відкритій формі користувач обирає тварину, процедуру, час та дату процедури, вносить додаткову інформацію та підтверджує запис. Після підтвердження та за день до прийому та за годину до прийому користувачу на пошту буде надіслано лист з інформацією про запис.
Extension	В випадку введення не коректних даних, кнопка підтвердження стає неактивною. . Якщо дані введено некоректно, то спливає повідомлення про це.
Post-Condition	Створений запис на прийом клієнтом.

Таблиця Б.9 – Варіант використання UC-09

Use case name	Здійснення онлайн-замовлень в вет-аптеці
Use case ID	UC-09
Goals	Придбати товари з вет-аптеки через інтерфейс застосунку
Actors	Авторизований користувач (клієнт)
Trigger	Користувач бажає замовити онлайн медикаменти або ветеринарні товари
Pre-conditions	Користувач авторизований, у системі є активні товари.
Flow of Events	Клієнт переходить у вкладку «Аптека», обирає категорії або вводить назву у пошук, переглядає список результатів, додає товари до кошика. У кошику користувач обирає спосіб оплати та доставки, підтверджує замовлення. Система створює запис у базі та надсилає підтвердження на email.
Extension	В випадку введення не коректних даних, кнопка підтвердження стає неактивною. Якщо дані введено некоректно, то спливає повідомлення про це.
Post-Condition	Замовлення створено, дані оновлено, клієнт отримує підтвердження.

Таблиця Б.10 – Варіант використання UC-10

Use case name	Додавання товарів
Use case ID	UC-10
Goals	Додати нову позицію товару у вет-аптеку
Actors	Авторизований користувач (адмін)
Trigger	Адміністратор бажає додати товар до бази

Продовження таблиці Б.10:

Pre-conditions	Адміністратор авторизований
Flow of Events	Адміністратор відкриває розділ «Аптека» і натискає кнопку «Додати товар». У формі введення він заповнює назву, опис, фото, ціну, категорію, а також вказує, чи товар є рецептурним. Після заповнення натискає «Зберегти», і система створює новий запис у базі.
Extension	В випадку введення не коректних даних, кнопка збереження стає неактивною. . Якщо дані введено некоректно, то спливає повідомлення про це.
Post-Condition	Новий товар збережено в базі, доступний для замовлення.

Таблиця Б.11 – Варіант використання UC-11

Use case name	Редагування товарів
Use case ID	UC-11
Goals	Оновити дані про вже існуючий товар
Actors	Авторизований користувач (адмін)
Trigger	Адміністратор бажає змінити деталі товару
Pre-conditions	Адміністратор авторизований, товар вже існує у базі
Flow of Events	Адміністратор знаходить товар у списку, натискає кнопку «Редагувати» і вносить необхідні зміни. Після оновлення інформації натискає «Зберегти», і зміни зберігаються у базі даних.
Extension	В випадку введення не коректних даних, кнопка збереження стає неактивною. Якщо якесь конкретне поле введено некоректно, то воно підсвічується червоним надписом.
Post-Condition	Інформацію про товар оновлено.

Таблиця Б.12 – Варіант використання UC-12

Use case name	Видалення товарів
Use case ID	UC-12
Goals	Видалити непотрібний або застарілий товар
Actors	Авторизований користувач (адмін)
Trigger	Адміністратор бажає видалити товар

Продовження таблиці Б.12:

Pre-conditions	Адміністратор авторизований, товар вже існує у базі
Flow of Events	Адміністратор знаходить товар у списку, натискає «В архів» та підтверджує дію. Після цього система переносить його в архів. Після перенесення в архів з’являється кнопка «Видалити», при натисканні на яку та на підтвердженні у спливаючому вікні товар видаляється з бази даних.
Extension	-
Post-Condition	Товар перенесено в архів та/або видалено або видалення скасовано.

Таблиця Б.13 – Варіант використання UC-13

Use case name	Перегляд та редагування замовлень
Use case ID	UC-13
Goals	Перегляд списку замовлень і оновлення їхнього статусу
Actors	Авторизований користувач (адмін)
Trigger	Адміністратор бажає переглянути чи змінити замовлення
Pre-conditions	Адміністратор авторизований, система містить хоча б одне створене замовлення
Flow of Events	Адміністратор відкриває вкладку «Замовлення», обирає потрібне зі списку та переглядає його зміст. За потреби змінює статус, додає коментар чи оновлює інформацію про доставку. Після змін натискає «Зберегти».
Extension	В випадку введення не коректних даних, кнопка збереження стає неактивною. Якщо дані введено некоректно, то спливає повідомлення про це.
Post-Condition	Замовлення оновлено та збережено у базі.

Таблиця Б.14 – Варіант використання UC-14

Use case name	Редагування графіку роботи
Use case ID	UC-14
Goals	Оновити графік роботи клініки
Actors	Авторизований користувач (адмін)
Trigger	Адміністратор бажає змінити графік
Pre-conditions	Адміністратор авторизований

Продовження таблиці Б.14:

Flow of Events	Адміністратор переходить у вкладку «Редагування графіку», встановлює робочі дні, години роботи, перерви та операційні дні. Після заповнення натискає кнопку «Зберегти». Система оновлює дані та використовує їх при формуванні вільних слотів для запису.
Extension	У випадку зміни розкладу таким чином, що дні, на які є записи, стають неробочими, записи скасовуються, а клієнтам надходять відповідні листи на пошту.
Post-Condition	Новий графік збережено, застосовується у системі.

Таблиця Б.15 – Варіант використання UC-15

Use case name	Додавання акаунту власника
Use case ID	UC-15
Goals	Створення нового облікового запису власника тварини
Actors	Авторизований користувач (адмін)
Trigger	Адміністратор бажає зареєструвати нового власника
Pre-conditions	Адміністратор авторизований
Flow of Events	Адміністратор переходить у розділ керування акаунтами, натискає кнопку «Додати власника», після чого заповнює форму з ПІБ, номером телефону, email-адресою. Після чого система надсилає код для підтвердження на введену пошту, а адміністратор повинен ввести його для продовження реєстрації клієнта. Далі натискає кнопку «Зберегти», і система створює новий обліковий запис, додаючи його до бази.
Extension	В випадку введення не коректних даних, кнопка збереження стає неактивною. Якщо дані введено некоректно, то спливає повідомлення про це.
Post-Condition	Новий акаунт власника збережено в базі даних.

Таблиця Б.16 – Варіант використання UC-16

Use case name	Редагування акаунту власника
Use case ID	UC-16
Goals	Внести зміни до наявного акаунту власника

Продовження таблиці Б.16:

Actors	Авторизований користувач (адмін)
Trigger	Адміністратор бажає оновити дані власника
Pre-conditions	Адміністратор авторизований, акаунт власника існує в системі
Flow of Events	Адміністратор відкриває список власників, знаходить потрібного, натискає кнопку «Редагувати». У формі змінює, наприклад, контактні дані або ПІБ, потім натискає «Зберегти». Система перевіряє зміни і оновлює запис у базі.
Extension	В випадку введення не коректних даних, кнопка збереження стає неактивною. Якщо якесь конкретне поле введено некоректно, то воно підсвічується червоним надписом.
Post-Condition	Дані акаунта успішно оновлено.

Таблиця Б.17 – Варіант використання UC-17

Use case name	Видалення акаунту власника
Use case ID	UC-17
Goals	Видалити обліковий запис власника тварини
Actors	Авторизований користувач (адмін)
Trigger	Адміністратор бажає видалити акаунт власника
Pre-conditions	Адміністратор авторизований, акаунт власника існує в системі і не має прив’язаних тварин
Flow of Events	Адміністратор обирає потрібного власника в списку, натискає кнопку «В архів». Система запитує підтвердження, після чого, у разі згоди, переносить обліковий запис в архів. В архіві адміністратор може видалити обліковий запис клієнта натиснувши на кнопку «Видалення».
Extension	-
Post-Condition	Акаунт власника додано в архів та/або видалено або видалення скасовано.

Таблиця Б.18 – Варіант використання UC-18

Use case name	Додавання картки пацієнта
Use case ID	UC-18
Goals	Створення нової картки пацієнта (тварини)
Actors	Авторизований користувач (адмін)
Trigger	Адміністратор бажає зареєструвати тварину

Продовження таблиці Б.18:

Pre-conditions	Адміністратор авторизований, акаунт власника вже існує
Flow of Events	Адміністратор переходить у розділ пацієнтів і натискає кнопку «Додати пацієнта». У формі заповнює поля: ім’я, вид, порода, дата народження, вказує власника з бази. Після заповнення форми натискає «Зберегти». Система перевіряє правильність введених даних і створює нову картку в базі.
Extension	В випадку введення не коректних даних, кнопка збереження стає неактивною. Якщо дані введено некоректно, то спливає повідомлення про це.
Post-Condition	Створено нову картку пацієнта, вона зберігається в базі й відображається у списку пацієнтів.

Таблиця Б.19 – Варіант використання UC-19

Use case name	Редагування картки пацієнта
Use case ID	UC-19
Goals	Оновити дані про тварину в наявній картці
Actors	Авторизований користувач (адмін)
Trigger	Адміністратор бажає змінити інформацію про пацієнта
Pre-conditions	Адміністратор авторизований, картка пацієнта вже існує
Flow of Events	Адміністратор відкриває картку потрібної тварини через список пацієнтів, натискає кнопку «Редагувати» і змінює відповідні поля (наприклад, вік, особливості, породу). Після внесення змін натискає кнопку «Зберегти». Система перевіряє введені дані і оновлює запис у базі.
Extension	В випадку введення не коректних даних, кнопка збереження стає неактивною. Якщо дані введено некоректно, то спливає повідомлення про це.
Post-Condition	Дані картки оновлено та збережено.

Таблиця Б.20 – Варіант використання UC-20

Use case name	Видалення картки пацієнта
Use case ID	UC-20
Goals	Видалити картку тварини

Продовження таблиці Б.20:

Actors	Авторизований користувач (адмін)
Trigger	Адміністратор бажає видалити пацієнта
Pre-conditions	Адміністратор авторизований, пацієнт зареєстрований, не має активних записів
Flow of Events	Адміністратор у списку пацієнтів натискає кнопку «В архів» навпроти потрібної картки. Дані пацієнта переносяться в архів. В архіві адміністратор може видалити картку пацієнта після натискання на кнопку «Видалити» та на підтвердження у спливаючому вікні.
Extension	-
Post-Condition	Картку пацієнта додано в архів та/або видалено або дію скасовано.

Таблиця Б.21 – Варіант використання UC-21

Use case name	Створення прийому
Use case ID	UC-21
Goals	Додати запис про майбутній або поточний прийом
Actors	Авторизований користувач (адмін)
Trigger	Адміністратор бажає зафіксувати новий прийом пацієнта
Pre-conditions	Адміністратор авторизований, пацієнт існує в системі
Flow of Events	Адміністратор відкриває сторінку додавання прийому, обирає власника, картку пацієнта, процедуру, дату та час, додатковий опис проблеми за необхідності. Після заповнення натискає «Зберегти». Система додає новий запис у медичну картку.
Extension	В випадку введення не коректних даних, кнопка збереження стає неактивною. Якщо дані введено некоректно, то спливає повідомлення про це.
Post-Condition	Новий прийом збережено та прикріплено до пацієнта.

Таблиця Б.22 – Варіант використання UC-22

Use case name	Редагування прийому
Use case ID	UC-22
Goals	Змінити інформацію у вже створеному прийомі
Actors	Авторизований користувач (адмін)
Trigger	Адміністратор бажає внести зміни до даних прийому

Продовження таблиці Б.22:

Pre-conditions	Адміністратор авторизований, прийом вже створено
Flow of Events	Адміністратор відкриває список прийомів пацієнта, натискає «Редагувати» і змінює потрібні поля (наприклад, діагноз, препарати, статус). Після редагування натискає «Зберегти». Система оновлює запис.
Extension	В випадку введення не коректних даних, кнопка збереження стає неактивною. Якщо дані введено некоректно, то спливає повідомлення про це.
Post-Condition	Прийом оновлено в медичній картці.

Таблиця Б.23 – Варіант використання UC-23

Use case name	Видалення прийому
Use case ID	UC-23
Goals	Видалити інформацію про прийом
Actors	Авторизований користувач (адмін)
Trigger	Адміністратор бажає видалити непотрібний прийом
Pre-conditions	Адміністратор авторизований, прийом існує
Flow of Events	Адміністратор у картці пацієнта натискає кнопку «В архів» біля запису про прийом. Після підтвердження прийом переноситься в архів. В архіві адміністратор може видалити прийом після натискання на кнопку «Видалити» та на підтвердження у спливаючому вікні.
Extension	-
Post-Condition	Прийом додано в архів та/або видалено або видалення скасовано.

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

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

Веб-застосунок для організації роботи ветеринарної клініки

Текст програми

КПІ.ІП-1204.045440.03.12

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Олена МАРЧЕНКО

Нормоконтроль:
_____ Максим ГОЛОВЧЕНКО

Виконавець:
_____ Анастасія БОНДАРЧУК

Київ – 2025

Посилання на репозиторій з повним текстом програмного коду
https://github.com/AndrushaNimbleHands/PetHealth.git

Файл adminOrders.js та adminOrderController.js
Реалізація функціональної задачі перегляду та редагування замовлень адміністратором.

```
const express = require('express');
const router = express.Router();
const auth = require('../middlewares/auth');
const adminOnly = require('../middlewares/adminOnly');
const adminOrderController = require('../controllers/adminOrderController');
router.get('/', auth, adminOnly, adminOrderController.getAllOrders);
router.patch('/:id', auth,  adminOnly, adminOrderController.updateOrder);

module.exports = router;

const Order = require('../models/Order');

exports.getAllOrders = async (req, res) => {
    try {
        const orders = await Order.find()
            .populate('userId', 'firstName lastName email')
            .populate('items.productId', 'name price')
            .populate('userId', 'firstName lastName email phone');

        res.json(orders);
    } catch (e) {
        res.status(500).json({ message: 'Помилка при отриманні замовлень',
error: e.message });
    }
};

exports.updateOrder = async (req, res) => {
    try {
        const { id } = req.params;
        const { status, paymentMethod, delivery } = req.body;

        const order = await Order.findById(id);
        if (!order) return res.status(404).json({ message: 'Замовлення не
знайдено' });

        if (status) order.status = status;
        if (paymentMethod) order.paymentMethod = paymentMethod;

        if (delivery) {
            if (!order.delivery) order.delivery = {};
            if (delivery.city !== undefined) order.delivery.city =
delivery.city;
            if (delivery.type !== undefined) order.delivery.type =
delivery.type;
            if (delivery.address !== undefined) order.delivery.address =
delivery.address;
        }

        await order.save();

        const populated = await Order.findById(order._id)
            .populate('userId', 'firstName lastName email')
            .populate('items.productId', 'name price');
```

```
        res.json(populated);
    } catch (e) {
        console.error('❌ updateOrder error:', e.message);
        res.status(500).json({ message: 'Помилка при оновленні замовлення',
error: e.message });
    }
};
```

Файл appointmentClientController.js

Реалізація функціональної задачі редагування та скасування прийому

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

exports.getByPetId = async (req, res) => {
    try {
        const petId = req.params.petId;
        const userId = req.user.id;

        const appointments = await Appointment.find({
            petId,
            status: { $in: ['scheduled', 'in_progress', 'completed',
'cancelled'] },
        })
            .populate('procedureId')
            .populate('procedureId.doctor', 'firstName lastName')
            .populate('petId', 'ownerId');


        const filtered = appointments.filter(appt =>
appt?.petId?.ownerId?.toString() === userId);

        res.json(filtered);
    } catch (err) {
        res.status(500).json({message: 'Помилка при отриманні прийомів'});
    }
};

exports.cancelAppointment = async (req, res) => {
    try {
        const appointment = await Appointment.findById(req.params.id);
        if (!appointment) return res.status(404).json({message: 'Прийом не
знайдено'});
        if (appointment.status !== 'scheduled') return
res.status(400).json({message: 'Прийом не можна скасувати'});
        if (appointment.userId.toString() !== req.user.id) return
res.status(403).json({message: 'Немає доступу'});

        const now = new Date();
        const appointmentDateTime = new
Date(`${appointment.date}T${appointment.startTime}`);

        const diffInMs = appointmentDateTime - now;
        const diffInHours = diffInMs / (1000 * 60 * 60);
        if (diffInHours < 12) return res.status(400).json({message: 'Скасування
можливе лише за 12 годин до прийому'});

        appointment.status = 'cancelled';
        await appointment.save();
        res.json({message: 'Прийом скасовано'});
    } catch (err) {
        res.status(500).json({message: 'Помилка при скасуванні прийому'});
    }
};
```

Файл appointmentController.js

Реалізація функціональної задачі створення прийому, що включає розрахунок вільних слотів часу, надсилання імейлів користувачу щодо прийомів, безпосередньо створення та редагування прийомів.

```
const Appointment = require('../models/Appointment');
const Procedure = require('../models/Procedure');
const Schedule = require('../models/Schedule');
const redis = require('../utils/redisClient');
const { io } = require("../index");
const mailer = require('../utils/mailer');

const sendAppointmentEmail = async ({ appointment, type = 'created' }) => {
    try {
        await appointment.populate([
            { path: 'userId', select: 'email firstName lastName' },
            { path: 'petId', select: 'name' },
            {
                path: 'procedureId',
                populate: {
                    path: 'doctor',
                    select: 'firstName lastName price'
                }
            },
            {
                path: 'usedMedicines.medicineId',
                select: 'name price'
            }
        ]);

        const formatDate = (dateStr) => {
            const date = new Date(dateStr);
            return date.toLocaleDateString('uk-UA', {
                day: '2-digit',
                month: '2-digit',
                year: 'numeric'
            });
        };

        // Мапа статусів для зручності
        const statusMap = {
            scheduled: 'Заплановано',
            in_progress: 'В процесі',
            completed: 'Завершено',
```

```
            cancelled: 'Скасовано'
        };

        if (appointment.status === 'cancelled') {
            const subject = `Скасування прийому
№${appointment.appointmentNumber}`;

            const html = `
                <h2>Прийом скасовано</h2>
                <p>Номер прийому:
<strong>${appointment.appointmentNumber}</strong></p>
                <p>Тварина: <strong>${appointment.petId?.name || '—
'}</strong></p>
                <p>Дата прийому:
<strong>${formatDate(appointment.date)}</strong></p>
                <p>Час: <strong>${appointment.startTime} –
${appointment.endTime}</strong></p>
                <p>Статус: <strong>Скасовано</strong></p>
                <p>Якщо у вас є питання, будь ласка, зв'яжіться з
клінікою.</p>
            `;

            await mailer.sendMail({
                from: `"PetHealth" <${process.env.EMAIL}>`,
                to: appointment.userId.email,
                subject,
                html
            });
            return;
        }
        const subject = type === 'created'
            ? `Підтвердження запису на прийом
№${appointment.appointmentNumber}`
            : `Оновлення прийому №${appointment.appointmentNumber}`;

        const animalInfo = appointment.animalInfo || {};
        const hasMedicalData =
            appointment.diagnosis ||
            appointment.prescription ||
            animalInfo.condition ||
            (animalInfo.temperature !== null && animalInfo.temperature !==
undefined) ||
            (animalInfo.weight !== null && animalInfo.weight !== undefined);
```

```
        const usedMedsTotal = appointment.usedMedicines?.reduce((sum, med) =>
{
            const price = med.medicineId?.price || 0;
            return sum + price * med.quantity;
        }, 0) || 0;

        const procedurePrice = appointment.procedureId?.price || 0;
        const totalPrice = procedurePrice + usedMedsTotal;

        let medicalHtml = '';
        if (hasMedicalData) {
            medicalHtml = `
                <hr>
                <h3>Медична інформація</h3>
                ${appointment.diagnosis ? `<p><strong>Діагноз:</strong>
${appointment.diagnosis}</p>` : ''}
                ${appointment.prescription ?
`<p><strong>Призначення:</strong> ${appointment.prescription}</p>` : ''}
                ${animalInfo.condition ? `<p><strong>Стан тварини:</strong>
${animalInfo.condition}</p>` : ''}
                ${(animalInfo.temperature !== null && animalInfo.temperature
!== undefined) ? `<p><strong>Температура:</strong> ${animalInfo.temperature}
°C</p>` : ''}
                ${(animalInfo.weight !== null && animalInfo.weight !==
undefined) ? `<p><strong>Вага:</strong> ${animalInfo.weight} кг</p>` : ''}
            `;
        }

        let usedMedsHtml = '';
        if (appointment.usedMedicines && appointment.usedMedicines.length >
0) {
            usedMedsHtml = `
                <hr>
                <h3>Використані медикаменти</h3>
                <table border="1" cellpadding="5" cellspacing="0">
                    <thead>
                        <tr>
                            <th>Назва</th>
                            <th>Кількість</th>
                            <th>Ціна за одиницю</th>
                            <th>Вартість</th>
                        </tr>
```

```
                    </thead>
                    <tbody>
                        ${appointment.usedMedicines.map(med => `
                            <tr>
                                <td>${med.medicineId?.name || '—'}</td>
                                <td>${med.quantity}</td>
                                <td>${med.medicineId?.price || '—'} грн</td>
                                <td>${med.quantity * (med.medicineId?.price
|| 0)} грн</td>
                            </tr>
                        `).join('')}
                    </tbody>
                </table>
            `;
        }

        const html = `
            <h2>Інформація про прийом</h2>
            <p><strong>Номер прийому:</strong>
${appointment.appointmentNumber}</p>
            <p><strong>Статус:</strong> ${statusMap[appointment.status] ||
appointment.status}</p>
            <p><strong>Тварина:</strong> ${appointment.petId?.name || '—
'}</p>
            <p><strong>Процедура:</strong> ${appointment.procedureId?.name ||
'—'}</p>
            <p><strong>Ціна процедури:</strong> ${procedurePrice} грн</p>
            <p><strong>Лікар:</strong>
${appointment.procedureId?.doctor?.firstName || ''}
${appointment.procedureId?.doctor?.lastName || ''}</p>
            <p><strong>Дата:</strong> ${formatDate(appointment.date)}</p>
            <p><strong>Час:</strong> ${appointment.startTime} -
${appointment.endTime}</p>
            ${appointment.comment ? `<p><strong>Коментар:</strong>
${appointment.comment}</p>` : ''}
            ${medicalHtml}
            ${usedMedsHtml}
            <hr>
            <h3>Загальна вартість прийому: ${totalPrice} грн</h3>
        `;

        await mailer.sendMail({
            from: `"PetHealth" <${process.env.EMAIL}>`,
```

```
            to: appointment.userId.email,
            subject,
            html
        });

    } catch (e) {
        console.error('Помилка при надсиланні email для прийому:',
e.message);
    }
};

exports.getAvailableSlots = async (req, res) => {
    const { date, procedureId } = req.query;
    const selectedDay = new Date(date).toLocaleString('en-US', { weekday:
'long' }).toLowerCase();

    const schedule = await Schedule.findOne().populate('weekSchedule');
    if (!schedule) return res.status(404).json({ message: 'Графік не
знайдено' });

    const daySettings = schedule.weekSchedule.find(d => d.day ===
selectedDay);
    if (!daySettings || !daySettings.isOpen) return res.json([]);

    const procedure = await Procedure.findById(procedureId);
    if (!procedure) return res.status(404).json({ message: 'Процедура не
знайдена' });

    const duration = procedure.duration;

    const existingAppointments = await Appointment.find({
        date,
        status: { $in: ['scheduled', 'in_progress', 'completed'] }
    });

    const busyTimes = existingAppointments.map(a => ({ start: a.startTime,
end: a.endTime }));

    const timeToMinutes = t => +t.split(":")[0] * 60 + +t.split(":")[1];
    const minutesToTime = m => `${String(Math.floor(m / 60)).padStart(2,
'0')}:${String(m % 60).padStart(2, '0')}`;

    const slots = [];
```

```
    let cursor = timeToMinutes(daySettings.workStart);
    const end = timeToMinutes(daySettings.workEnd);

    const now = new Date();
    const isToday = date === now.toISOString().split('T')[0];

    while (cursor + duration <= end) {
        const slotStart = minutesToTime(cursor);
        const slotEnd = minutesToTime(cursor + duration);

        // Якщо дата сьогодні, пропускаємо слоти в минулому
        if (isToday) {
            const [hour, minute] = slotStart.split(':').map(Number);
            const slotDateTime = new Date(date);
            slotDateTime.setHours(hour, minute, 0, 0);
            if (slotDateTime <= now) {
                cursor += duration;
                continue;
            }
        }

        const overlaps = busyTimes.some(b =>
            timeToMinutes(b.start) < cursor + duration &&
            timeToMinutes(b.end) > cursor
        );

        const isLocked = await redis.get(`lock:${date}:${slotStart}`);

        if (!overlaps &&
            (!daySettings.hasLunchBreak ||
                slotEnd <= daySettings.lunchStart ||
                slotStart >= daySettings.lunchEnd) &&
            !isLocked) {
            slots.push({ start: slotStart, end: slotEnd });
        }

        cursor += duration;
    }

    res.json(slots);
};
```

```
exports.createAppointment = async (req, res) => {
    const session = await Appointment.startSession();
    try {
        await session.withTransaction(async () => {
            const { petId, procedureId, date, startTime, comment } =
req.body;
            const userId = req.user.id;
            const lockKey = `lock:${date}:${startTime}`;

            const lock = await redis.set(lockKey, userId, { nx: true, px:
30000 });
            if (!lock) throw new Error("Цей слот уже заблокований іншим
користувачем");

            const procedure = await Procedure.findById(procedureId);
            const duration = procedure.duration;

            const [h, m] = startTime.split(':').map(Number);
            const endTime = `${String(Math.floor((h * 60 + m + duration) /
60)).padStart(2, '0')}:${String((h * 60 + m + duration) % 60).padStart(2,
'0')}`;

            const conflict = await Appointment.findOne({
                date,
                $or: [{ startTime: { $lt: endTime }, endTime: { $gt:
startTime } }]
            }).session(session);
            console.log('🔒 Lock result:', lock);
            console.log('⛔ Conflict result:', conflict);
            if (conflict) throw new Error("Слот уже зайнятий!");

            const appointmentNumber = await
generateUniqueAppointmentNumber();

            const newAppointment = new Appointment({
                userId,
                petId,
                procedureId,
                appointmentNumber,
                date,
                startTime,
                endTime,
                comment: comment || '',
```

```
                status: 'scheduled',
                animalInfo: { weight: null, temperature: null, condition: ''
},
                diagnosis: '',
                prescription: ''
            });

            await newAppointment.save({ session });

            if (!io) {
                console.error('❌ io is undefined — socket не під'єднаний');
            } else {
                io.emit('appointment:created', { date, startTime, endTime });
                console.log('📡 io.emit успішно виконано');
            }
            res.status(201).json(newAppointment);

            await sendAppointmentEmail({ appointment: newAppointment, type:
'created' });
        });
    } catch (err) {
        res.status(409).json({ message: err.message });
    } finally {
        try {
            const { date, startTime } = req.body || {};
            if (date && startTime) {
                await redis.del(`lock:${date}:${startTime}`);
                console.log('🔓 Lock released');
            }
        } catch (cleanupErr) {
            console.error('❌ Failed to release Redis lock:',
cleanupErr.message);
        }
    }
};

const generateUniqueAppointmentNumber = async () => {
    let unique = false;
    let number = '';
    while (!unique) {
        const timestamp = Date.now().toString().slice(-6);
        const random = Math.floor(1000 + Math.random() * 9000);
```

```
        number = `APT-${timestamp}${random}`;
        const exists = await Appointment.findOne({ appointmentNumber: number
});
        if (!exists) unique = true;
    }
    return number;
};

exports.getByPet = async (req, res) => {
    try {
        const appointments = await Appointment.find({ petId: req.params.petId
})
            .populate({
                path: 'procedureId',
                populate: {
                    path: 'doctor',
                    model: 'User',
                    select: 'firstName lastName price'
                }
            })
            .populate({
                path: 'userId',
                select: 'firstName lastName role'
            })
            .sort({ date: -1, startTime: -1 });

        const formatted = appointments.map(a => ({
            _id: a._id,
            date: a.date,
            startTime: a.startTime,
            endTime: a.endTime,
            status: a.status,
            comment: a.comment,
            isArchived: a.isArchived,
            diagnosis: a.diagnosis,
            prescription: a.prescription,
            animalInfo: a.animalInfo || {},

            procedureId: a.procedureId ? {
                _id: a.procedureId._id,
                name: a.procedureId.name,
                duration: a.procedureId.duration,
                doctor: a.procedureId.doctor ? {
```

```
                    _id: a.procedureId.doctor._id,
                    firstName: a.procedureId.doctor.firstName,
                    lastName: a.procedureId.doctor.lastName
                } : null
            } : null,

            userId: a.userId ? {
                _id: a.userId._id,
                firstName: a.userId.firstName,
                lastName: a.userId.lastName,
                role: a.userId.role
            } : null,
        }));

        res.json(formatted);
    } catch (err) {
        console.error(err);
        res.status(500).json({ message: err.message });
    }
};

exports.archive = async (req, res) => {
    try {
        await Appointment.findByIdAndUpdate(req.params.id, { isArchived: true
});
        res.sendStatus(204);
    } catch (err) {
        console.error('archive error:', err);
        res.status(500).json({ message: 'Не вдалося заархівувати' });
    }
};

exports.restore = async (req, res) => {
    try {
        await Appointment.findByIdAndUpdate(req.params.id, { isArchived:
false });
        res.sendStatus(204);
    } catch (err) {
        console.error('restore error:', err);
        res.status(500).json({ message: 'Не вдалося відновити' });
    }
};
```

```
exports.update = async (req, res) => {
    try {
        const { id } = req.params;
        const {
            status,
            animalInfo,
            diagnosis,
            prescription,
            usedMedicines,
            recipeId
        } = req.body;

        const updateFields = {};

        if (status) updateFields.status = status;
        if (diagnosis) updateFields.diagnosis = diagnosis;
        if (prescription) updateFields.prescription = prescription;
        if (recipeId !== undefined) updateFields.recipeId = recipeId;
        if (usedMedicines) updateFields.usedMedicines = usedMedicines;

        if (animalInfo) {
            if (animalInfo.weight !== undefined)
updateFields['animalInfo.weight'] = animalInfo.weight;
            if (animalInfo.temperature !== undefined)
updateFields['animalInfo.temperature'] = animalInfo.temperature;
            if (animalInfo.condition !== undefined)
updateFields['animalInfo.condition'] = animalInfo.condition;
        }

        const updated = await Appointment.findByIdAndUpdate(
            id,
            { $set: updateFields },
            { new: true }
        ).populate([
            {
                path: 'petId',
                populate: [
                    { path: 'ownerId', select: 'firstName lastName phone
email' },
                    { path: 'speciesId', select: 'name' }
                ]
            },
            { path: 'procedureId', select: 'name' },
```

```
            { path: 'usedMedicines.medicineId', select: 'name price' },
            { path: 'recipeId', select: 'name' }
        ]);

        if (!updated) return res.status(404).json({ message: "Appointment not
found" });

        res.json({ message: "Appointment updated", updated });

        await sendAppointmentEmail({ appointment: updated, type: 'updated'
});
    } catch (err) {
        console.error(err);
        res.status(500).json({ message: "Server error while updating
appointment" });
    }
};

exports.remove = async (req, res) => {
    try {
        await Appointment.findByIdAndDelete(req.params.id);
        res.sendStatus(204);
    } catch (err) {
        console.error('remove error:', err);
        res.status(500).json({ message: 'Не вдалося видалити прийом' });
    }
};

exports.getAllAdmin = async (req, res) => {
    try {
        const { type, species, date, search, archived } = req.query;
        const filter = {};

        if (type === 'appointments') filter.status = { $ne: 'scheduled' };
        if (type === 'bookings') filter.status = 'scheduled';
        if (archived !== undefined) filter.isArchived = archived === 'true';
        if (date) filter.date = date;

        let results = await Appointment.find(filter)
            .populate([
                {
                    path: 'petId',
                    populate: [
```

```
                        { path: 'ownerId', select: 'firstName lastName phone
email' },
                        { path: 'speciesId', select: 'name' }
                    ]
                },
                {
                    path: 'procedureId',
                    populate: { path: 'doctor', select: 'firstName lastName
email' }
                },
                {
                    path: 'userId',
                    select: 'firstName lastName email'
                },
                {
                    path: 'recipeId',
                    populate: {
                        path: 'products.productId',
                        select: 'name unit price'
                    }
                },
                {
                    path: 'usedMedicines.medicineId',
                    select: 'name'
                }
            ])
            .sort({ date: -1, startTime: -1 });

        if (species) {
            results = results.filter(r => r.petId?.speciesId?._id?.toString()
=== species);
        }

        if (search) {
            const lower = search.toLowerCase();
            results = results.filter(a =>
                a.petId?.name?.toLowerCase().includes(lower) ||
                a.appointmentNumber?.toLowerCase().includes(lower) ||
                a.petId?.ownerId?.phone?.includes(lower) ||
                a.petId?.ownerId?.email?.toLowerCase().includes(lower) ||
                a.petId?.ownerId?.firstName?.toLowerCase().includes(lower) ||
                a.petId?.ownerId?.lastName?.toLowerCase().includes(lower)
            );
```

```
        }

        res.json(results);
    } catch (err) {
        console.error('❌ getAllAdmin error:', err);
        res.status(500).json({ message: err.message });
    }
};

exports.createByAdmin = async (req, res) => {
    const session = await Appointment.startSession();
    try {
        await session.withTransaction(async () => {
            const {
                userId,
                petId,
                procedureId,
                doctorId,
                date,
                startTime,
                comment
            } = req.body;

            if (!userId || !petId || !procedureId || !doctorId || !date ||
!startTime) {
                return res.status(400).json({ message: 'Не всі обов’язкові
поля заповнені' });
            }

            const lockKey = `lock:${date}:${startTime}`;
            const lock = await redis.set(lockKey, userId, { nx: true, px:
30000 });
            if (!lock) throw new Error("Цей слот уже заблокований іншим
користувачем");

            const procedure = await Procedure.findById(procedureId);
            if (!procedure) throw new Error("Процедура не знайдена");

            const duration = procedure.duration;
            const [h, m] = startTime.split(':').map(Number);
            const endMinutes = h * 60 + m + duration;
            const endTime = `${String(Math.floor(endMinutes /
60)).padStart(2, '0')}:${String(endMinutes % 60).padStart(2, '0')}`;
```

```
            const conflict = await Appointment.findOne({
                date,
                $or: [{ startTime: { $lt: endTime }, endTime: { $gt:
startTime } }]
            }).session(session);
            if (conflict) throw new Error("Слот уже зайнятий!");

            const appointmentNumber = await
generateUniqueAppointmentNumber();

            const newAppointment = new Appointment({
                userId,
                petId,
                procedureId,
                appointmentNumber,
                date,
                startTime,
                endTime,
                comment: comment || '',
                status: 'scheduled',
                animalInfo: { weight: null, temperature: null, condition: ''
},
                diagnosis: '',
                prescription: ''
            });

            await newAppointment.save({ session });

            io.emit('appointment:created', { date, startTime, endTime });
            res.status(201).json(newAppointment);

            await sendAppointmentEmail({ appointment: newAppointment, type:
'created' });
        });
    } catch (err) {
        console.error('❌ createByAdmin error:', err.message);
        res.status(409).json({ message: err.message });
    } finally {
        try {
            const { date, startTime } = req.body || {};
            if (date && startTime) {
                await redis.del(`lock:${date}:${startTime}`);
```

```
            }
        } catch (e) {
            console.error('❌ Redis unlock error:', e.message);
        }
    }
};
```

Файл appointment.js

Реалізація керування роутами до попереднього файлу контролерів.

```
const express = require('express');
const router = express.Router();
const auth = require('../middlewares/auth');
const adminOrDoctor = require('../middlewares/adminOrDoctor');
const appointmentController =
require('../controllers/appointmentController');

router.get('/slots', auth, appointmentController.getAvailableSlots);
router.post('/', auth, appointmentController.createAppointment);

router.get('/admin', auth, adminOrDoctor, appointmentController.getAllAdmin);
router.post('/admin', auth, adminOrDoctor,
appointmentController.createByAdmin);
router.patch('/:id/archive', auth, adminOrDoctor,
appointmentController.archive);
router.get('/by-pet/:petId', auth, adminOrDoctor,
appointmentController.getByPet);
router.patch('/:id/restore', auth, adminOrDoctor,
appointmentController.restore);
router.patch('/:id', auth, adminOrDoctor, appointmentController.update);
router.delete('/:id', auth, adminOrDoctor, appointmentController.remove);
module.exports = router;
```

Файл category.js та categoryController.js

Реалізація функціоналу створення та редагування, видалення категорій товару вет-аптеки.

```
const express = require('express');
const router = express.Router();
const controller = require('../controllers/categoryController');
const auth = require('../middlewares/auth');
const adminOnly = require('../middlewares/adminOnly');

router.get('/', auth, adminOnly, controller.getAll);
router.post('/', auth, adminOnly, controller.create);
router.put('/:id', auth, adminOnly, controller.update);
router.delete('/:id', auth, adminOnly, controller.remove);
```

```
module.exports = router;

const Category = require('../models/Category');
const Product = require('../models/Product');

exports.getAll = async (req, res) => {
    try {
        const categories = await Category.find();
        res.json(categories);
    } catch (e) {
        res.status(500).json({ error: e.message });
    }
};

exports.create = async (req, res) => {
    try {
        const category = await Category.create({ name: req.body.name });
        res.status(201).json(category);
    } catch (e) {
        res.status(400).json({ error: e.message });
    }
};

exports.update = async (req, res) => {
    try {
        const updated = await Category.findByIdAndUpdate(req.params.id, {
name: req.body.name }, { new: true });
        if (!updated) return res.status(404).json({ error: 'Category not
found' });
        res.json(updated);
    } catch (e) {
        res.status(400).json({ error: e.message });
    }
};

exports.remove = async (req, res) => {
    try {
        const category = await Category.findByIdAndDelete(req.params.id);
        if (!category) return res.status(404).json({ error: 'Category not
found' });
```

```
        await Product.updateMany({ category: category.name }, { category:
'Загальні' });

        res.json({ message: 'Категорія видалена, товари оновлено' });
    } catch (e) {
        res.status(500).json({ error: e.message });
    }
};
```

Файл petCards.js та petCardsController.js

Реалізація функціоналу створення, редагування, архівації, видалення, отримання з бд карток пацієнтів.

```
const express = require('express');
const router = express.Router();
const petCardsController = require ("../controllers/petCardsController");
const auth = require('../middlewares/auth');
const adminOrDoctor = require('../middlewares/adminOrDoctor');
const adminOnly = require('../middlewares/adminOnly');

router.get('/users', auth, adminOrDoctor, petCardsController.getClients);
router.get('/mine', auth, petCardsController.getMine);
router.get('/', auth, adminOrDoctor, petCardsController.getAll);
router.post('/', auth, adminOnly, petCardsController.create);
router.patch('/:id/archive', auth, adminOrDoctor,
petCardsController.archive);
router.patch('/:id/restore', auth, adminOrDoctor,
petCardsController.restore);
router.delete('/:id', auth, adminOrDoctor, petCardsController.remove);
router.get('/:id', auth, adminOrDoctor, petCardsController.getOne);
router.patch('/:id', auth, adminOrDoctor, petCardsController.update);

module.exports = router;

const PetCard = require('../models/PetCard');
const auth = require("../middlewares/auth");
const User = require("../models/User");

exports.getAll = async (req, res) => {
    try {
        const {archived, species, owner} = req.query;
        const filter = {};
        if (archived !== undefined) filter.isArchived = archived === 'true';
        if (species) filter.speciesId = species;
        if (owner) filter.ownerId = owner;
```

```
        const cards = await PetCard.find(filter)
            .populate({path: 'ownerId', model: 'User', select: 'id firstName
lastName email phone'})
            .populate({path: 'speciesId', model: 'Species', select: 'id
name'});

        res.json(cards);
    } catch (e) {
        res.status(500).json({error: e.message});
    }
};

exports.getOne = async (req, res) => {
    try {
        const card = await PetCard.findById(req.params.id)
            .populate({path: 'ownerId', model: 'User', select: 'id firstName
lastName email phone'})
            .populate({path: 'speciesId', model: 'Species', select: 'id
name'});
        if (!card) return res.status(404).json({error: 'Картку не
знайдено'});
        res.json(card);
    } catch (e) {
        res.status(500).json({error: e.message});
    }
};

exports.archive = async (req, res) => {
    console.log('📩 archive запит на ID:', req.params.id);
    try {
        await PetCard.findByIdAndUpdate(req.params.id, {isArchived: true});
        res.json({success: true});
    } catch (e) {
        res.status(500).json({error: e.message});
    }
};

exports.restore = async (req, res) => {
    try {
        const result = await PetCard.findByIdAndUpdate(req.params.id,
{isArchived: false});
```

```
        if (!result) return res.status(404).json({error: 'Картку не
знайдено'});
        res.json({success: true});
    } catch (e) {
        res.status(500).json({error: e.message});
    }
};

exports.remove = async (req, res) => {
    try {
        const result = await PetCard.findByIdAndDelete(req.params.id);
        if (!result) return res.status(404).json({error: 'Картку не
знайдено'});
        res.json({success: true});
    } catch (e) {
        res.status(500).json({error: e.message});
    }
};


exports.update = async (req, res) => {
    try {
        const {name, speciesId, breed, ownerId, birthday} = req.body;

        const updated = await PetCard.findByIdAndUpdate(
            req.params.id,
            {name, speciesId, breed, ownerId, birthday},
            {new: true}
        );

        res.json(updated);
    } catch (e) {
        res.status(500).json({error: e.message});
    }
};

exports.getClients = async (req, res) => {
    try {
        // role: 'client',
        const clients = await User.find({isArchived: false}).select('_id
firstName lastName');
        console.log('Клієнти з бази:', clients);
        const formatted = clients.map(d => ({
```

```
            _id: d.id,
            name: `${d.firstName} ${d.lastName}`
        }));
        res.json(formatted);
    } catch (e) {
        console.error('GET /petcards/users error:', e);
        res.status(500).json({error: 'Internal server error'});
    }
};

exports.getMine = async (req, res) => {
    try {
        const pets = await PetCard.find({ownerId: req.user.id, isArchived:
false})
            .populate('speciesId', 'name'); // якщо потрібно
        res.json(pets);
    } catch (err) {
        res.status(500).json({message: 'Помилка при отриманні тварин'});
    }
};

exports.create = async (req, res) => {
    try {
        const {name, speciesId, breed, birthday, ownerId} = req.body;

        if (!name || !speciesId || !ownerId) {
            return res.status(400).json({message: 'Поля "name", "speciesId"
та "ownerId" є обов’язковими.'});
        }

        const newCard = new PetCard({
            name,
            speciesId,
            breed,
            birthday,
            ownerId
        });

        await newCard.save();
        res.status(201).json(newCard);
    } catch (err) {
        res.status(500).json({message: err.message});
    }
```

```
};
```

Файл petCardsClient.js та petCardsClientController.js

Реалізація функціоналу отримання карток пацієнтів їх власником.

```
const express = require('express');
const router = express.Router();
const auth = require('../middlewares/auth');
const petCardClientController =
require('../controllers/petCardClientController');
const appointmentClientController =
require('../controllers/appointmentClientController');

router.get('/petcards/mine', auth, petCardClientController.getMyPetCards);
router.get('/petcards/mine/:id', auth,
petCardClientController.getMyPetCardById);
router.get('/appointments/:petId', auth,
appointmentClientController.getByPetId);
router.patch('/appointments/:id/cancel', auth,
appointmentClientController.cancelAppointment);

module.exports = router;
const PetCard = require('../models/PetCard');

exports.getMyPetCards = async (req, res) => {
    try {
        const cards = await PetCard.find({
            ownerId: req.user.id,
            isArchived: false
        })
            .populate('speciesId', 'name')
            .populate('ownerId', 'firstName lastName');

        res.json(cards);
    } catch (err) {
        res.status(500).json({message: 'Помилка при завантаженні карток'});
    }
};

exports.getMyPetCardById = async (req, res) => {
    try {
        const card = await PetCard.findOne({
            _id: req.params.id,
            ownerId: req.user.id,
            isArchived: false
```

```
        })
            .populate('speciesId', 'name')
            .populate('ownerId', 'firstName lastName');

        if (!card) return res.status(404).json({message: 'Картка не
знайдена'});
        res.json(card);
    } catch (err) {
        res.status(500).json({message: 'Помилка при завантаженні картки'});
    }
};
```

Файл product.js та productController.js

Реалізація функціоналу створення та редагування, архівації та видалення товарів вет-аптеки.

```
const express = require('express');
const router = express.Router();
const controller = require('../controllers/productController');
const auth = require('../middlewares/auth');
const adminOnly = require('../middlewares/adminOnly');

router.get('/', auth, adminOnly, controller.getAll);
router.get('/:id', auth, adminOnly, controller.getById);
router.post('/', auth, adminOnly, controller.create);
router.put('/:id', auth, adminOnly, controller.update);
router.patch('/:id/archive', auth, adminOnly, controller.archive);
router.patch('/:id/restore', auth, adminOnly, controller.restore);
router.delete('/:id', auth, adminOnly, controller.remove);

module.exports = router;
const Product = require('../models/Product');

exports.getAll = async (req, res) => {
    try {
        const { archived, species, categoryId, search, prescription } =
req.query;
        const filter = {};

        if (archived !== undefined) {
            filter.isArchived = archived === 'true';
        }

        if (species) {
            filter.speciesId = species;
```

```
        }

        if (categoryId) {
            filter.categoryId = categoryId;
        }

        if (search) {
            filter.name = { $regex: search, $options: 'i' };
        }

        if (prescription === 'true') {
            filter.isPrescriptionFree = false;
        }

        const products = await Product.find(filter)
            .populate('speciesId', 'name')
            .populate('categoryId', 'name');

        res.json(products);
    } catch (e) {
        res.status(500).json({ error: e.message });
    }
};



exports.getById = async (req, res) => {
    try {
        const product = await
Product.findById(req.params.id).populate('speciesId', 'name');
        if (!product) return res.status(404).json({ error: 'Product not
found' });
        res.json(product);
    } catch (e) {
        res.status(500).json({ error: e.message });
    }
};

exports.create = async (req, res) => {
    try {
        console.log('BODY:', req.body); // <— подивись, що реально приходить

        const newProduct = new Product(req.body);
```

```
        const saved = await newProduct.save();
        res.status(201).json(saved);
    } catch (e) {
        console.error('❌ Error saving product:', e.message, e.errors);
        res.status(400).json({ error: e.message, details: e.errors });
    }
};


exports.update = async (req, res) => {
    try {
        const updated = await Product.findByIdAndUpdate(req.params.id,
req.body, { new: true });
        if (!updated) return res.status(404).json({ error: 'Product not
found' });
        res.json(updated);
    } catch (e) {
        res.status(400).json({ error: e.message });
    }
};

exports.archive = async (req, res) => {
    try {
        const updated = await Product.findByIdAndUpdate(
            req.params.id,
            { isArchived: true, archivedAt: new Date() },
            { new: true }
        );
        if (!updated) return res.status(404).json({ error: 'Product not
found' });
        res.json(updated);
    } catch (e) {
        res.status(500).json({ error: e.message });
    }
};

exports.restore = async (req, res) => {
    try {
        const updated = await Product.findByIdAndUpdate(
            req.params.id,
            { isArchived: false, archivedAt: null },
            { new: true }
        );
```

```
        if (!updated) return res.status(404).json({ error: 'Product not
found' });
        res.json(updated);
    } catch (e) {
        res.status(500).json({ error: e.message });
    }
};

exports.remove = async (req, res) => {
    try {
        const deleted = await Product.findByIdAndDelete(req.params.id);
        if (!deleted) return res.status(404).json({ error: 'Product not
found' });
        res.json({ message: 'Product deleted' });
    } catch (e) {
        res.status(500).json({ error: e.message });
    }
};
```

Файл procedure.js

Реалізація функціоналу створення та редагування, архівації та видалення процедур вет-клініки.

```
const express = require('express');
const router = express.Router();
const Procedure = require('../models/Procedure');
const auth = require('../middlewares/auth');
const adminOrDoctor = require('../middlewares/adminOrDoctor');
const User = require("../models/User");

router.get('/', auth, async (req, res) => {
    try {
        const list = await Procedure.find({ archived: false })
            .populate('doctor', 'name')
            .populate('species', 'name');
        res.json(list);
    } catch (err) {
        console.error('GET /procedures', err);
        res.status(500).json({ message: 'Помилка отримання процедур' });
    }
});

router.get('/archive', auth, adminOrDoctor, async (req, res) => {
    try {
        const list = await Procedure.find({ archived: true })
```

```
            .populate('doctor', 'name')
            .populate('species', 'name');
        res.json(list);
    } catch (err) {
        console.error('GET /procedures/archive', err);
        res.status(500).json({ message: 'Помилка отримання архіву' });
    }
});

router.post('/', auth, adminOrDoctor, async (req, res) => {
    try {
        const newProc = new Procedure(req.body);
        await newProc.save();
        res.status(201).json(newProc);
    } catch (err) {
        console.error('POST /procedures', err);
        res.status(400).json({ message: 'Не вдалося створити процедуру' });
    }
});

router.put('/:id', auth, adminOrDoctor, async (req, res) => {
    try {
        const updated = await Procedure.findByIdAndUpdate(req.params.id,
req.body, { new: true });
        res.json(updated);
    } catch (err) {
        console.error('PUT /procedures/:id', err);
        res.status(400).json({ message: 'Помилка при оновленні процедури' });
    }
});

router.post('/:id/archive', auth, adminOrDoctor, async (req, res) => {
    try {
        const updated = await Procedure.findByIdAndUpdate(req.params.id, {
            archived: true,
            archivedAt: new Date()
        }, { new: true });
        res.json(updated);
    } catch (err) {
        console.error('ARCHIVE /procedures/:id', err);
        res.status(400).json({ message: 'Помилка при архівації' });
    }
});
```

```
router.post('/:id/restore', auth, adminOrDoctor, async (req, res) => {
    try {
        const updated = await Procedure.findByIdAndUpdate(req.params.id, {
            archived: false,
            archivedAt: null
        }, { new: true });
        res.json(updated);
    } catch (err) {
        console.error('RESTORE /procedures/:id', err);
        res.status(400).json({ message: 'Помилка при відновленні процедури'
});
    }
});

router.delete('/:id', auth, adminOrDoctor, async (req, res) => {
    try {
        await Procedure.findByIdAndDelete(req.params.id);
        res.sendStatus(204);
    } catch (err) {
        console.error('DELETE /procedures/:id', err);
        res.status(400).json({ message: 'Помилка видалення процедури' });
    }
});

router.get('/doctors', auth, async (req, res) => {
    try {
        const doctors = await User.find({ role: 'doctor', isArchived: false
}).select('_id firstName lastName');
        const formatted = doctors.map(d => ({
            _id: d._id,
            name: `${d.firstName} ${d.lastName}`
        }));
        res.json(formatted);
    } catch (e) {
        console.error('GET /procedures/doctors error:', e);
        res.status(500).json({ error: 'Internal server error' });
    }
});
```

```
module.exports = router;
```

Файл recipe.js

Реалізація функціоналу створення та редагування рецептів, що виписуються під час прийому, а також надсилання їх на пошту клієнту.

```
const express = require('express');
const router = express.Router();
const auth = require('../middlewares/auth');
const Recipe = require('../models/Recipe');
const mailer = require('../utils/mailer');

const sendRecipeEmail = async ({ recipe }) => {
    try {
        await recipe.populate({
            path: 'products.productId',
            select: 'name unit price'
        });

        const subject = `Створено новий рецепт №${recipe._id}`;

        const html = `
            <h2>Інформація про рецепт №${recipe._id}</h2>
            <table border="1" cellpadding="5" cellspacing="0">
                <thead>
                    <tr>
                        <th>Назва медикаменту</th>
                        <th>Одиниця виміру</th>
                        <th>Кількість</th>
                        <th>Ціна за одиницю</th>
                        <th>Вартість</th>
                    </tr>
                </thead>
                <tbody>
                    ${recipe.products.map(prod => `
                        <tr>
                            <td>${prod.productId?.name || '—'}</td>
                            <td>${prod.productId?.unit || '—'}</td>
                            <td>${prod.quantity}</td>
                            <td>${prod.productId?.price || '—'} грн</td>
                            <td>${prod.quantity * (prod.productId?.price ||
0)} грн</td>
                        </tr>
                    `).join('')}
                </tbody>
```

```
            </table>
        `;

        await mailer.sendMail({
            from: `"PetHealth" <${process.env.EMAIL}>`,
            to: recipe.userId.email,
            subject,
            html
        });
    } catch (e) {
        console.error('Помилка при надсиланні email для рецепта:',
e.message);
    }
};

router.post('/', auth, async (req, res) => {
    try {
        const { userId, petId, doctorId, products, validUntil, comment } =
req.body;

        if (!Array.isArray(products) || products.length === 0) {
            return res.status(400).json({ message: 'Список медикаментів
порожній або некоректний' });
        }

        const recipe = new Recipe({
            userId,
            petId,
            doctorId,
            products,
            validUntil,
            comment: comment || ''
        });

        await recipe.save();


        sendRecipeEmail({ recipe }).catch(err => {
            console.error('Помилка при надсиланні email рецепта:',
err.message);
        });

        res.status(201).json(recipe);
```

```
    } catch (err) {
        console.error('Create recipe error:', err);
        res.status(500).json({
            message: 'Помилка при створенні рецепта',
            error: err.message
        });
    }
});

router.patch('/:id', auth, async (req, res) => {
    try {
        const { id } = req.params;
        const { products, comment, validUntil } = req.body;

        const updated = await Recipe.findByIdAndUpdate(
            id,
            {
                ...(products && { products }),
                ...(comment !== undefined && { comment }),
                ...(validUntil && { validUntil })
            },
            { new: true }
        ).populate([
            'products.productId',
            { path: 'userId', select: 'email firstName lastName' }
        ]);

        if (!updated) return res.status(404).json({ message: 'Recipe not
found' });

        sendRecipeEmail({ recipe: updated }).catch(err => {
            console.error('Помилка при надсиланні email рецепта:',
err.message);
        });

        res.json(updated);
    } catch (e) {
        res.status(500).json({ message: 'Error updating recipe', error:
e.message });
    }
});
```

```
router.get('/client', auth, async (req, res) => {
    try {
        const userId = req.user.id;
        const now = new Date();

        const recipes = await Recipe.find({
            userId,
            validUntil: { $gte: now },
            products: { $exists: true, $not: { $size: 0 } }
        }).populate([
            {
                path: 'products.productId',
                select: 'name unit price'
            },
            {
                path: 'userId',
                select: 'email firstName lastName'
            }
        ]);

        res.json(recipes);
    } catch (e) {
        res.status(500).json({
            message: 'Помилка при завантаженні рецептів',
            error: e.message
        });
    }
});

module.exports = router;
```

Файл schedule.js та scheduleController.js

Реалізація функціоналу редагування та отримання графіку роботи клініки, а також обробка випадків, коли при зміні графіку роботи виникають проблеми в записах на прийоми.

```
const express = require('express');
const router = express.Router();
const auth = require('../middlewares/auth');
const adminOnly = require('../middlewares/adminOnly');
const scheduleController = require('../controllers/ScheduleController');

router.get('/', auth, adminOnly, scheduleController.getSchedule);
router.put('/', auth, adminOnly, scheduleController.updateSchedule);
```

```
router.get('/appointments/in-closed-days', auth, adminOnly,
scheduleController.getAppointmentsInClosedDays);
router.post('/appointments/notify-in-closed-days', auth, adminOnly,
scheduleController.cancelAppointmentsInClosedDays);
module.exports = router;

const Schedule = require('../models/Schedule');
const Appointment = require('../models/Appointment');
const Day = require('../models/Day');
const User = require('../models/User');
const transporter = require("../utils/mailer");

exports.cancelAppointmentsInClosedDays = async (req, res) => {
    try {
        const schedule = await Schedule.findOne().populate('weekSchedule');
        if (!schedule) return res.json({ updated: 0 });

        const closedDays = schedule.weekSchedule
            .filter(day => !day.isOpen)
            .map(day => day.day);

        const appointments = await Appointment.find({ status: 'scheduled' })
            .populate('userId')
            .populate('procedureId')
            .populate('petId');

        const toCancel = appointments.filter(a => {
            const day = new Date(a.date).toLocaleString('en-US', { weekday:
'long' }).toLowerCase();
            return closedDays.includes(day);
        });

        for (const a of toCancel) {
            a.status = 'cancelled';
            await a.save();

            const petName = a.animalInfo?.petId?.name || 'ваш улюбленець';
            const userName = `${a.userId?.firstName || ''}
${a.userId?.lastName || ''}`.trim();
            const procedure = a.procedureId?.name || 'процедура';
            const dateStr = new Date(a.date).toLocaleDateString('uk-UA');

            if (a.userId?.email) {
```

```
                await transporter.sendMail({
                    from: `"PetHealth" <${process.env.EMAIL}>`,
                    to: a.userId.email,
                    subject: 'Ваш запис скасовано',
                    html: `
                        <p>Доброго дня${userName ? ', ' + userName : ''}!</p>
                        <p>Запис для <strong>${petName}</strong> на
<strong>${procedure}</strong>
                        <strong>${dateStr}</strong> з
<strong>${a.startTime}</strong> по <strong>${a.endTime}</strong>
                        <strong>скасовано</strong>, оскільки клініка не
працює в цей день.</p>
                        <p>Будь ласка, створіть новий запис або зв'яжіться з
клінікою для уточнення деталей.</p>
                        <br/>
                        <p>З повагою,<br/>PetHealth</p>
                    `
                });
            }
        }

        res.json(toCancel);;
    } catch (err) {
        console.error("Помилка при скасуванні записів:", err);
        res.status(500).json({ message: 'Помилка скасування записів' });
    }
};



exports.updateSchedule = async (req, res) => {
    try {
        const { weekSchedule } = req.body;

        // Видалити старі дні (щоб не було сміття)
        const oldSchedule = await Schedule.findOne();
        if (oldSchedule && oldSchedule.weekSchedule?.length) {
            await Day.deleteMany({ _id: { $in: oldSchedule.weekSchedule } });
        }

        // Створити нові дні
        const newDays = await Day.insertMany(weekSchedule);
        const newIds = newDays.map(day => day._id);
```

```
        // Записати ID нових днів у Schedule
        let schedule = await Schedule.findOne();
        if (!schedule) {
            schedule = new Schedule({ weekSchedule: newIds });
        } else {
            schedule.weekSchedule = newIds;
        }
        await schedule.save();

        res.sendStatus(200);
    } catch (err) {
        console.error("Помилка при оновленні графіку:", err);
        res.status(500).json({ message: 'Не вдалося оновити графік' });
    }
};


exports.getSchedule = async (req, res) => {
    try {
        const schedule = await Schedule.findOne().populate('weekSchedule');
        if (!schedule) return res.json({ weekSchedule: [] });
        res.json(schedule);
    } catch (err) {
        res.status(500).json({ error: 'Server error' });
    }
};

exports.getAppointmentsInClosedDays = async (req, res) => {
    try {
        const schedule = await Schedule.findOne().populate('weekSchedule');
        if (!schedule) return res.json([]);

        const closedDays = schedule.weekSchedule
            .filter(day => !day.isOpen)
            .map(day => day.day);

        const appointments = await Appointment.find()
            .populate({
                path: 'procedureId',
                select: 'name'
            });
```

```
        const badAppointments = appointments.filter(a => {
            const apptDay = new Date(a.date).toLocaleString('en-US', {
weekday: 'long' }).toLowerCase();
            return closedDays.includes(apptDay) && !a.isArchived;
        });

        res.json(badAppointments);
    } catch (err) {
        console.error("Помилка при перевірці записів у закриті дні:", err);
        res.status(500).json({ message: 'Не вдалося отримати записи' });
    }
};
```

Файл shop.js та shopController.js

Реалізація функціоналу отримання інформації про товари, створення замовлень у вет-аптеці та їх перегляд.

```
const express = require('express');
const router = express.Router();
const shopController = require('../controllers/shopController');
const auth = require('../middlewares/auth');

router.get('/products', shopController.getProducts);
router.get('/categories', shopController.getCategories);
router.get('/species', shopController.getSpecies);
router.post('/order', auth, shopController.createOrder);
router.get('/orders', auth, shopController.getUserOrders);
module.exports = router;
const Product = require('../models/Product');
const Category = require('../models/Category');
const Species = require('../models/Species');
const Order = require('../models/Order');
const Recipe = require('../models/Recipe');


exports.getProducts = async (req, res) => {
    try {
        const { search, speciesId, categoryId, page = 1, limit = 10 } =
req.query;
        const filter = { isArchived: false };

        if (search) filter.name = { $regex: search, $options: 'i' };
        if (speciesId) filter.speciesId = speciesId;
        if (categoryId) filter.categoryId = categoryId;
```

```
        const products = await Product.find(filter)
            .populate('categoryId', 'name')
            .populate('speciesId', 'name')
            .skip((page - 1) * limit)
            .limit(Number(limit));

        const count = await Product.countDocuments(filter);

        res.json({
            products,
            totalPages: Math.ceil(count / limit),
        });
    } catch (e) {
        res.status(500).json({ error: 'Помилка при завантаженні товарів' });
    }
};

exports.getCategories = async (req, res) => {
    try {
        const categories = await Category.find();
        res.json(categories);
    } catch (e) {
        res.status(500).json({ error: 'Помилка при завантаженні категорій'
});
    }
};

exports.getSpecies = async (req, res) => {
    try {
        const species = await Species.find();
        res.json(species);
    } catch (e) {
        res.status(500).json({ error: 'Помилка при завантаженні видів тварин'
});
    }
};

exports.createOrder = async (req, res) => {
    try {
        const { userId, items, delivery, paymentMethod, recipes } = req.body;

        if (!userId || !items?.length || !delivery || !paymentMethod) {
```

```
            return res.status(400).json({ error: 'Некоректні дані замовлення'
});
        }

        // Перевірка доставки
        switch (delivery.method) {
            case 'np_branch':
                if (!delivery.city || !delivery.branch) {
                    return res.status(400).json({ error: 'Місто та №
відділення обов’язкові' });
                }
                break;
            case 'np_postomat':
                if (!delivery.city || !delivery.postomat) {
                    return res.status(400).json({ error: 'Місто та ID
поштомату обов’язкові' });
                }
                break;
            case 'np_courier':
                if (!delivery.city || !delivery.address) {
                    return res.status(400).json({ error: 'Місто та адреса
обов’язкові' });
                }
                break;
            case 'pickup':
                break;
            default:
                return res.status(400).json({ error: 'Невідомий метод
доставки' });
        }

        // Валідація оплати
        const validMethods = {
            pickup: ['cash', 'card'],
            np_courier: ['cash', 'card'],
            np_branch: ['cod'],
            np_postomat: ['cod']
        };
        if (!validMethods[delivery.method]?.includes(paymentMethod)) {
            return res.status(400).json({ error: 'Недопустимий спосіб оплати
для цього типу доставки' });
        }
```

```
        // Перевірка залишків і підрахунок суми
        let total = 0;
        for (const item of items) {
            const product = await Product.findById(item.productId);
            if (!product || product.isArchived) {
                return res.status(400).json({ error: `Товар недоступний:
${item.productId}` });
            }

            if (product.stock < item.quantity) {
                return res.status(400).json({ error: `Недостатньо товару:
${product.name}` });
            }

            total += product.price * item.quantity;

            await Product.findByIdAndUpdate(item.productId, {
                $inc: { stock: -item.quantity }
            });
        }

        // 🔁 Зменшення кількості в рецептах
        if (recipes && typeof recipes === 'object') {
            for (const [productId, recipeId] of Object.entries(recipes)) {
                const recipe = await Recipe.findById(recipeId);
                if (!recipe) continue;

                const index = recipe.products.findIndex(p =>
p.productId.toString() === productId);
                if (index !== -1) {
                    const item = recipe.products[index];
                    const orderItem = items.find(i => i.productId ===
productId);
                    const orderedQty = orderItem?.quantity || 0;

                    if (item.quantity > orderedQty) {
                        item.quantity -= orderedQty;
                    } else {
                        recipe.products.splice(index, 1);
                    }

                    await recipe.save();
                }
```

```
            }
        }

        const orderNumber = await generateUniqueOrderNumber();

        const order = await Order.create({
            orderNumber,
            userId,
            items,
            total,
            delivery,
            paymentMethod,
            status: 'pending',
            createdAt: new Date()
        });

        res.status(201).json({ message: 'Замовлення створено', orderId:
order._id });
    } catch (e) {
        console.error('❌ ORDER ERROR', e);
        res.status(500).json({ error: 'Помилка при оформленні замовлення' });
    }
};

const generateUniqueOrderNumber = async () => {
    let unique = false;
    let number = '';

    while (!unique) {
        const timestamp = Date.now().toString().slice(-6);
        const random = Math.floor(100 + Math.random() * 900); // 3 цифри
        number = `ORD-${timestamp}${random}`;

        const exists = await Order.findOne({ orderNumber: number });
        if (!exists) unique = true;
    }

    return number;
};

exports.getUserOrders = async (req, res) => {
    try {
        const userId = req.user.id;
```

```
        console.log(userId);
        const orders = await Order.find({ userId })
            .sort({ createdAt: -1 })
            .populate('items.productId', 'name price unit')
            .lean();

        res.json(orders);
    } catch (e) {
        console.error('❌ getUserOrders error:', e);
        res.status(500).json({ error: 'Не вдалося завантажити замовлення' });
    }
};
```

Файл user.js

Реалізація функціоналу редагування даних акаунта, редагування даних поточного користувача, редагування даних про картки пацієнтів, для яких поточний користувач є власником.

```
const express = require('express');
const router = express.Router();
const User = require('../models/User');
const authMiddleware = require('../middlewares/auth');
const PetCard = require("../models/PetCard");
const Code = require("../models/Code");
const transporter = require("../utils/mailer");

router.put('/user/me', authMiddleware, async (req, res) => {
    const {firstName, lastName, birthday} = req.body;
    try {
        const updated = await User.findByIdAndUpdate(
            req.user.id,
            {firstName, lastName, birthday},
            {new: true}
        );
        res.json({success: true, updated});
    } catch (e) {
        console.error('Update error:', e);
        res.status(500).json({error: 'Internal server error'});
    }
});

router.get('/user/me', authMiddleware, async (req, res) => {
    try {
        const user = await User.findById(req.user.id).select('firstName
lastName birthday email phone');
```

```
        if (!user) return res.status(404).json({error: 'User not found'});
        res.json(user);
    } catch (e) {
        console.error('Get user error:', e);
        res.status(500).json({error: 'Internal server error'});
    }
});


router.put('/user/account', authMiddleware, async (req, res) => {
    const { email, phone } = req.body;
    if (email && req.user.role !== 'admin') {
        return res.status(403).json({ error: 'Зміна email дозволена лише
адміністраторам' });
    }

    try {
        const updateData = {};
        if (req.user.role === 'admin' && email) {
            updateData.email = email;
        }
        if (phone) {
            updateData.phone = phone;
        }

        const updated = await User.findByIdAndUpdate(req.user.id, updateData,
{ new: true });
        res.json({ success: true, updated });
    } catch (e) {
        console.error('Update error:', e);
        res.status(500).json({ error: 'Internal server error' });
    }
});
router.post('/user/send-phone-code', authMiddleware, async (req, res) => {
    const user = await User.findById(req.user.id);
    if (!user) return res.status(404).json({error: 'Користувача не
знайдено'});

    const {newPhone} = req.body;
    if (!newPhone) return res.status(400).json({error: 'Номер телефону не
вказано'});

    const code = Math.floor(100000 + Math.random() * 900000).toString();
```

```
    const expiresAt = new Date(Date.now() + 5 * 60 * 1000); // 5 хв

    await Code.create({email: user.email, code, expiresAt});

    await transporter.sendMail({
        from: `PetHealth <${process.env.EMAIL}>`,
        to: user.email,
        subject: 'Код підтвердження зміни номера телефону',
        text: `Ваш код підтвердження зміни номера телефону: ${code}. Дійсний
5 хвилин.`
    });

    res.json({message: 'Код підтвердження надіслано на email'});
});
router.post('/user/confirm-phone-change', authMiddleware, async (req, res) =>
{
    const {code, newPhone} = req.body;
    const user = await User.findById(req.user.id);
    if (!user) return res.status(404).json({error: 'Користувача не
знайдено'});

    const record = await Code.findOne({email: user.email, code, expiresAt:
{$gt: new Date()}});
    if (!record) return res.status(400).json({error: 'Невірний код або час
дії минув'});

    user.phone = newPhone;
    await user.save();
    await Code.deleteOne({_id: record._id});

    res.json({message: 'Номер телефону успішно змінено'});
});

router.get('/user/pets', authMiddleware, async (req, res) => {
    try {
        const pets = await PetCard.find({ownerId: req.user.id});
        res.json(pets);
    } catch (e) {
        console.error('Get pets error:', e);
        res.status(500).json({error: 'Internal server error'});
    }
});
```

```
router.put('/user/pet-card', authMiddleware, async (req, res) => {
    try {
        const {name, breed, birthday, speciesId} = req.body;

        const updated = await PetCard.findOneAndUpdate(
            {ownerId: req.user.id},
            {name, breed, birthday, speciesId: speciesId},
            {new: true}
        );

        if (!updated) {
            return res.status(404).json({error: 'Pet not found or not
yours'});
        }

        res.json({success: true, pet: updated});
    } catch (e) {
        console.error('PetCard update error:', e);
        res.status(500).json({error: 'Internal server error'});
    }
});
router.post('/user/pet-card', authMiddleware, async (req, res) => {
    try {
        const {name, breed, birthday, speciesId} = req.body;

        await PetCard.create(
            {name, breed, birthday, speciesId: speciesId, ownerId:
req.user.id},
        );
    } catch (e) {
        console.error('PetCard create error:', e);
        res.status(500).json({error: 'Internal server error'});
    }
    res.json({success: true, message: 'Pet created successfully'});
});

router.delete('/pet-card/:id', authMiddleware, async (req, res) => {
    const petId = req.params.id;

    try {
        console.log('Trying to delete pet with id:', petId, 'for user:',
req.user.id);
```

```
        const deleted = await PetCard.findOneAndDelete({
            _id: petId
        });

        if (!deleted) {
            return res.status(404).json({error: 'Pet not found or not
yours'});
        }

        res.json({success: true, message: 'Pet deleted successfully'});
    } catch (e) {
        console.error('PetCard delete error:', e);
        res.status(500).json({error: 'Internal server error'});
    }
});

module.exports = router;
```

Файл usersAdmin.js та userController.js

Реалізація функціоналу додавання, редагування архівування та видалення акаунтів користувачів адміністратором.

```
const express = require('express');
const router = express.Router();
const auth = require('../middlewares/auth');
const adminOnly = require('../middlewares/adminOnly');
const userController = require("../controllers/userController");

router.get('/', auth, adminOnly, userController.getAllUsers);
router.post('/', auth, adminOnly, userController.createUser);
router.patch('/:id/archive', auth, adminOnly, userController.archiveUser);
router.patch('/:id/restore', auth, adminOnly, userController.restoreUser);
router.patch('/:id', auth, adminOnly, userController.updateUser);
router.delete('/:id', auth, adminOnly, userController.deleteUser);

module.exports = router;
const User = require('../models/User');

exports.getAllUsers = async (req, res) => {
    try {
        const { role, archived } = req.query;
        const filter = {
            _id: { $ne: req.user.id }
        };
```

```
        if (role) filter.role = role;
        if (archived !== undefined) filter.isArchived = archived === 'true';
        const users = await User.find(filter);
        res.json(users);
    } catch (e) {
        res.status(500).json({ error: e.message });
    }
};

exports.updateUser = async (req, res) => {
    try {
        const { firstName, lastName, email, phone, birthday, role } =
req.body;
        await User.findByIdAndUpdate(req.params.id, {
            ...(firstName && { firstName }),
            ...(lastName && { lastName }),
            ...(email && { email }),
            ...(phone && { phone }),
            ...(birthday && { birthday }),
            ...(role && { role })
        });
        res.json({ success: true });
    } catch (e) {
        res.status(500).json({ error: e.message });
    }
};


exports.archiveUser = async (req, res) => {
    try {
        await User.findByIdAndUpdate(req.params.id, { isArchived: true });
        res.json({ success: true });
    } catch (e) {
        res.status(500).json({ error: e.message });
    }
};

exports.restoreUser = async (req, res) => {
    try {
        await User.findByIdAndUpdate(req.params.id, { isArchived: false });
        res.json({ success: true });
    } catch (e) {
        res.status(500).json({ error: e.message });
```

```
    }
};

exports.deleteUser = async (req, res) => {
    try {
        await User.findByIdAndDelete(req.params.id);
        res.json({ success: true });
    } catch (e) {
        res.status(500).json({ error: e.message });
    }
};
exports.createUser = async (req, res) => {
    try {
        const { firstName, lastName, email, phone, birthday, role } =
req.body;
        const existingEmail = await User.findOne({ email });
        if (existingEmail) {
            return res.status(400).json({ message: 'Користувач з таким email
вже існує' });
        }
        const existingPhone = await User.findOne({ phone });
        if (existingPhone) {
            return res.status(400).json({ message: 'Користувач з таким
номером телефону вже існує' });
        }

        const newUser = new User({
            firstName,
            lastName,
            email,
            phone,
            birthday,
            role
        });
        await newUser.save();

        res.status(201).json({ message: 'Користувача успішно створено' });
    } catch (err) {
        console.error('Помилка при створенні користувача:', err);
        res.status(500).json({ message: 'Помилка сервера при створенні
користувача' });
    }
};
```

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

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

ВЕБЗАСТОСУНОК ДЛЯ ОРГАНІЗАЦІЇ РОБОТИ ВЕТЕРИНАРНОЇ КЛІНІКИ

Програма та методика тестування

КПІ.ІП-1204.045440.04.51

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Олена МАРЧЕНКО

Нормоконтроль:
_____ Максим ГОЛОВЧЕНКО

Виконавець:
_____ Анастасія БОНДАРЧУК

Київ – 2025

ЗМІСТ

1 ОБ’ЄКТ ВИПРОБУВАНЬ

Об’єктом випробування є вебзастосунок для організації роботи ветеринарної клініки, розроблений у межах дипломного проєкту. Програмне забезпечення реалізовано з використанням технологій React (клієнтська частина), Node.js + Express (серверна частина) та MongoDB Atlas (віддалена база даних).

Система протестована в середовищі Opera GX, Google Chrome, та Microsoft Edge на операційних системах Windows 10/11.

Вебзастосунок призначений для використання трьома основними типами користувачів: адміністратором, лікарем та клієнтом. Усі основні сценарії використання були охоплені мануальним тестуванням.

2 МЕТА ТЕСТУВАННЯ

Метою тестування є наступне:

- перевірка правильності роботи програмного забезпечення відповідно до визначених функціональних вимог;
- перевірка коректного збереження, редагування та видалення даних у системі;
- перевірка сумісності вебзастосунку з останніми версіями сучасних браузерів (Google Chrome, Opera, Microsoft Edge);
- виявлення проблем, помилок і недоліків в інтерфейсі та логіці роботи системи для їх подальшого усунення;
- перевірка зручності та доступності графічного інтерфейсу користувача з точки зору логіки, структури й підказок;
- перевірка коректності поведінки застосунку у випадках помилкового введення даних, невалідних дій або порожніх форм;
- перевірка роботи основних користувацьких сценаріїв для різних ролей (адміністратор, лікар, клієнт) вручну, без застосування автоматизованих засобів.

3 МЕТОДИ ТЕСТУВАННЯ

Для тестування програмного забезпечення було застосовано мануальне тестування як основний метод контролю якості.

Мануальне тестування передбачає покрокове виконання заздалегідь визначених сценаріїв користувача з метою перевірки відповідності поведінки системи функціональним вимогам. Усі перевірки здійснювалися вручну, без використання автоматизованих тестових фреймворків.

У межах методики було застосовано наступні типи тестування:

- Функціональне тестування. Перевірка відповідності реалізованих функцій заявленим вимогам: створення пацієнтів, запис на прийом, управління прийомами, замовлення тощо;
- Системне тестування. Перевірка роботи вебзастосунку як єдиного цілого: взаємодія клієнтської частини, серверної логіки та бази даних;
- Динамічне тестування. Тестування в процесі реального виконання програмного забезпечення, у тому числі з використанням реальних браузерів та операційних систем;
- Тестування «чорної скриньки». Основний підхід, за якого оцінюється коректність результатів без аналізу внутрішньої реалізації функцій;
- Click-through testing. Покрокове проходження інтерфейсів для перевірки сценаріїв користувачів різних ролей (адміністратор, лікар, клієнт).

4 ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Під час проведення тестування будуть використовуватись наступні допоміжні засоби:

- стороннє програмне забезпечення (браузери, поштові клієнти);
- утиліти (вбудовані в браузер інструменти розробника, консольні логи та повідомлення помилок у браузері).

Порядок проведення тестування буде наступним:

- динамічне тестування на відповідність функціональним вимогам шляхом виконання покрокових сценаріїв користувача (click-through testing);;
- тестування інтерфейсу в різних браузерах, встановлених в ОС Windows 10/11;
- тестування зручності використання – перевірка логіки переходів між сторінками, читабельності інтерфейсу, наявності повідомлень про помилки;
- тестування поведінки при зміні графіка роботи клініки – перевірка автоматичного скасування записів у закриті дні та email-сповіщень;
- тестування взаємодії з Redis – перевірка блокування слотів запису в режимі реального часу при паралельних діях кількох користувачів;
- перевірка обробки невалідних сценаріїв, наприклад, неправильний код підтвердження, відсутність рецепта при оформленні замовлення, нестача медикаментів у рецепті.

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

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

ВЕБЗАСТОСУНОК ДЛЯ ОРГАНІЗАЦІЇ РОБОТИ ВЕТЕРИНАРНОЇ КЛІНІКИ

Керівництво користувача

КПІ.ІП-1204.045440.05.34

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Олена МАРЧЕНКО

Нормоконтроль:
_____ Максим ГОЛОВЧЕНКО

Виконавець:
_____ Анастасія БОНДАРЧУК

Київ – 2025

ЗМІСТ

1 ПРИЗНАЧЕННЯ ПРОГРАМИ

«PetHealth» – це вебзастосунок для організації роботи ветеринарної клініки, що забезпечує функціональність онлайн-запису на прийом, ведення електронних карток пацієнтів, управління медичними процедурами, призначеннями, аптечними товарами та рецептами.

Кінцева збірка програмного забезпечення включає:

а) клієнтську частину (frontend), зібрану за допомогою npm run build;

б) серверну частину (backend), підготовлену для розгортання на хостингу (Render або локальний Node-сервер);

в) файл .env з конфігурацією підключення до бази даних та Redis;

Вихідні коди збережено у Git-репозиторії з наступною структурою:

г) client/ – клієнтська частина вебзастосунку, розроблена на React:

1) public/ – статичні ресурси
2) src/ – основна логіка застосунку:
   – components/ – багаторазові UI-компоненти
   – layouts/ – шаблони елементів сторінок
   – pages/ – реалізація сторінок
   – context/ – контексти React
   – services/ – API-запити до бекенду
   – assets/styles/ – стилі
3) App.js, index.js – точка входу та ініціалізація React

д) server/ – серверна частина, реалізована на Node.js + Express:

1) config/ – конфігурації
2) controllers/ – логіка обробки запитів
3) middlewares/ – авторизація, валідація
4) models/ – Mongoose-схеми
5) routes/ – маршрути API
6) utils/ – допоміжні функції
7) .env – файл середовищних змінних (непублічний)
8) index.js – точка входу сервера

е) package.json, package-lock.json – для керування залежностями окремо у client і server.

2 ПІДГОТОВКА ДО РОБОТИ З ПРОГРАМНИМ ЗАБЕЗПЕЧЕННЯМ

2.1 Системні вимоги для коректної роботи

Для успішної роботи вебзастосунку необхідне виконання наступних вимог:

– наявність пристрою зі стабільним доступом до мережі Інтернет;

– сучасний веббраузер: Google Chrome, Mozilla Firefox, Microsoft Edge, Opera тощо (останні версії);

– підтримка JavaScript у браузері (увімкнена за замовчуванням);

– рекомендована швидкість інтернет-з’єднання не менше 10 Мбіт/с.

2.2 Завантаження застосунку

Оскільки застосунок є веборієнтованим, не потребується встановлення локального програмного забезпечення. Доступ до функціоналу здійснюється за допомогою переходу на відповідну URL-адресу в браузері.

Для завантаження локальної версії необхідно завантажити код з репозиторію GitHub та виконати дії, описані в README репозиторію.

2.3 Перевірка коректної роботи

Для перевірки коректності доступу до вебзастосунку необхідно:

а) Перейти за вказаною адресою через браузер.

б) Переконатися, що завантажилась головна сторінка застосунку.

в) Переконатися, що відображається інтерфейс реєстрації або входу.

г) Виконати спробу реєстрації нового користувача або авторизації через email та код:

1) якщо код надходить на пошту, то система сповіщень працює;

2) якщо після введення коду здійснюється вхід, то бекенд, база даних і токени працюють коректно.

д) У разі помилки – перевірити з’єднання з інтернетом, оновити сторінку або звернутись до адміністратора системи.

3 ВИКОНАННЯ ПРОГРАМИ

3.1 Інструкція для користувача (клієнта):

а) Відкрити вебзастосунок за посиланням.

б) Якщо користувач не зареєстрований (рис. 3.1-3.2):

1) перейти на форму реєстрації;

2) ввести адресу електронної пошти та номер телефону;

3) натиснути кнопку «Код»;

4) ввести отриманий на вказану пошту шестизначний код підтвердження у відповідне поле;

5) натиснути кнопку «Зареєструватись».

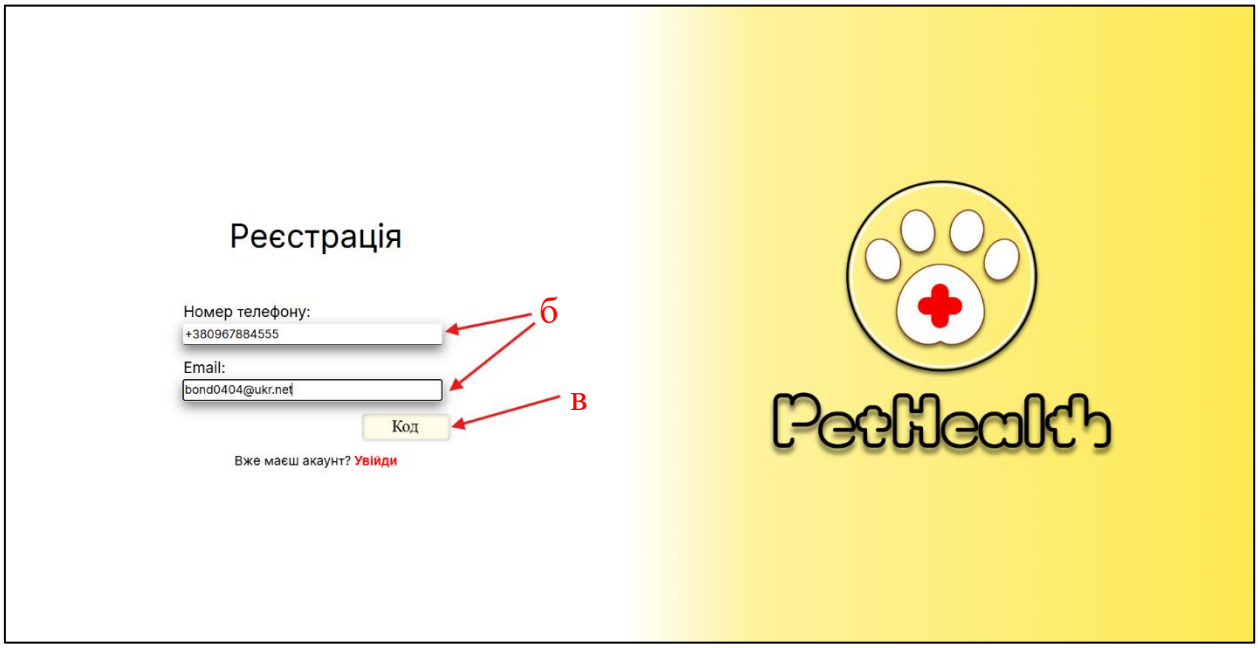


Рисунок 3.1 – Реєстрація користувача ч.1

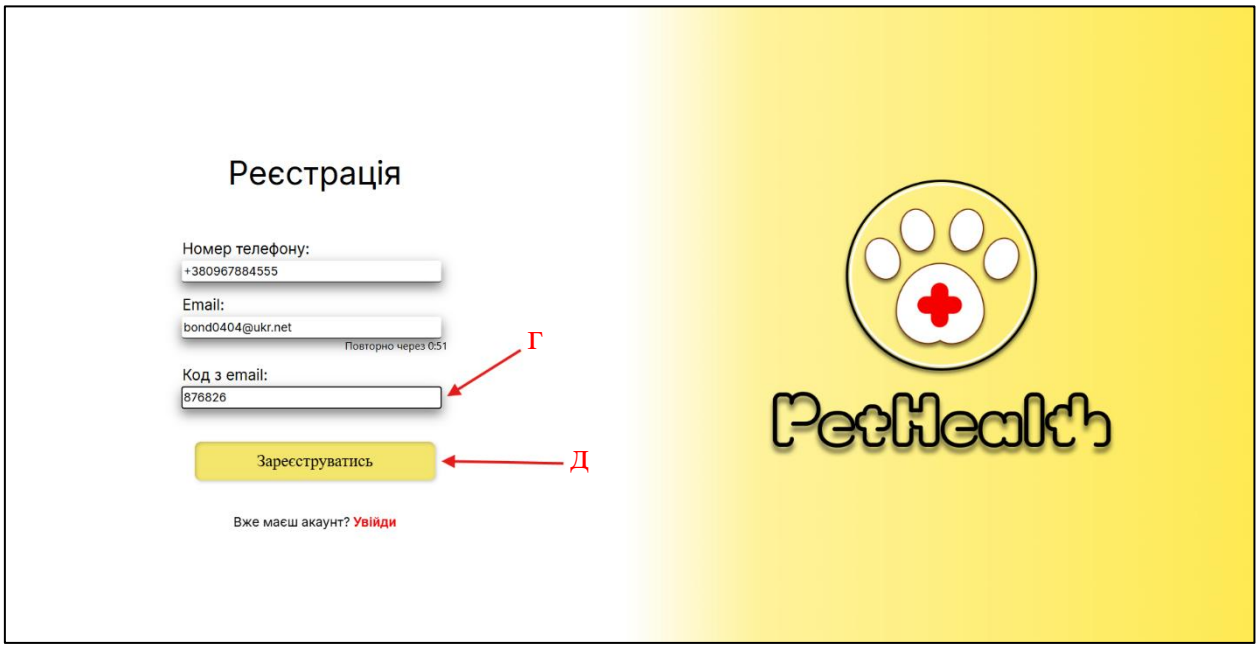


Рисунок 3.2 – Реєстрація користувача ч.2

в) Якщо користувач зареєстрований (рис. 3.3-3.4):

1) перейти на форму авторизації;

2) ввести адресу електронної пошти;

3) натиснути кнопку «Код»;

4) ввести отриманий на вказану пошту шестизначний код підтвердження у відповідне поле;

5) натиснути кнопку «Увійти».

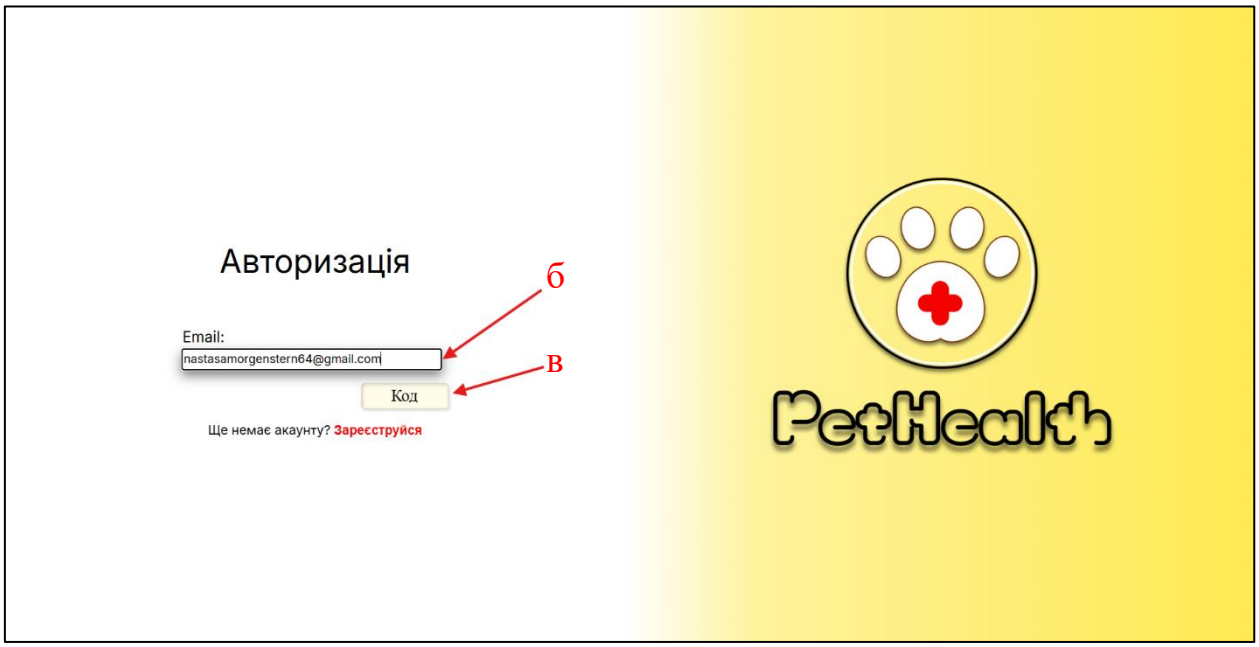


Рисунок 3.3 – Авторизація користувача ч.1

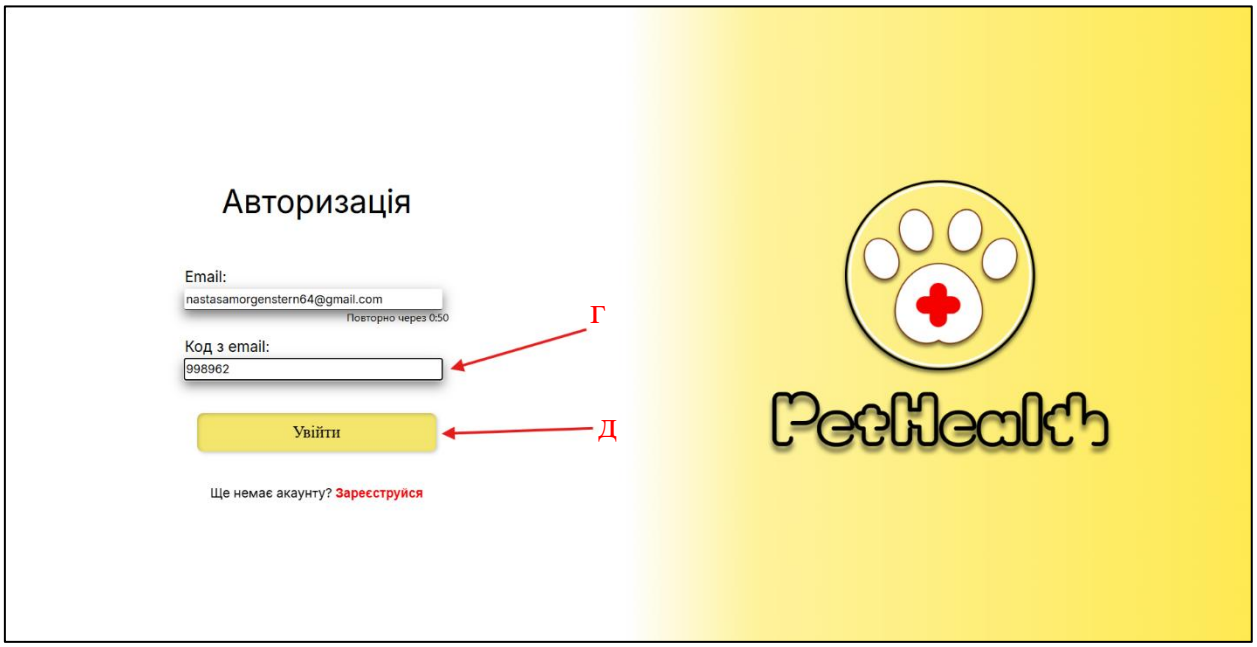


Рисунок 3.4 – Авторизація користувача ч.2

г) Після успішної авторизації відкривається головна сторінка застосунку (рис. 3.5).

Рисунок 3.5 – Головна сторінка

д) Для редагування інформації про користувача (рис.):

1) відкрити бокове меню та обрати вкладку «Редагувати дані користувача»;

2) внести зміни у необхідні поля (ім’я, прізвище, дата народження) та натиснути кнопку «Зберегти» або «Скасувати», якщо не потрібно зберігати зміни.

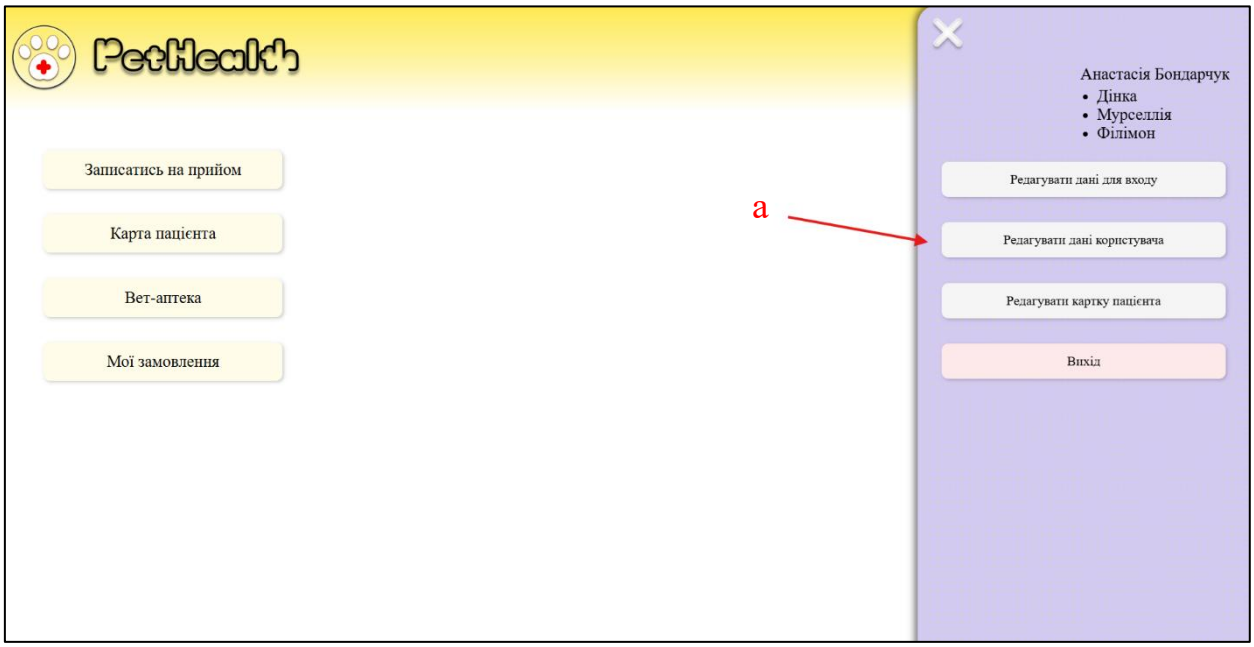


Рисунок 3.6 – Редагування даних користувача ч.2

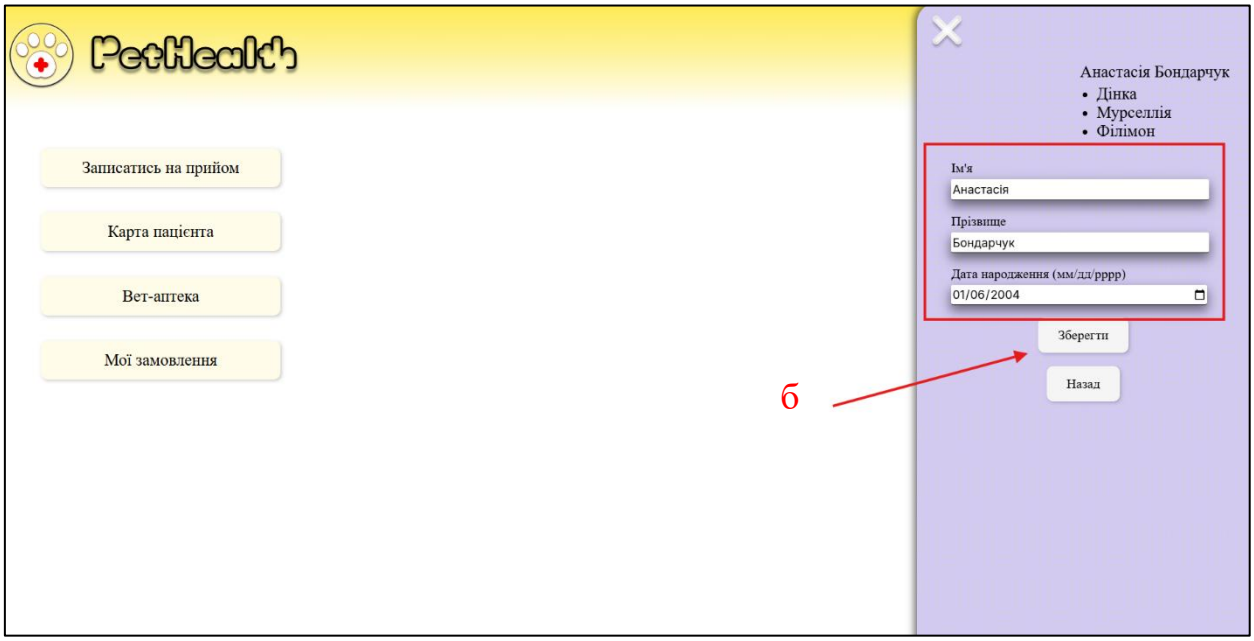


Рисунок 3.7 – Редагування даних користувача ч.2

е) Для редагування даних для входу (рис. 3.8-3.10):

1) відкрити бокове меню та обрати вкладку «Редагувати дані для входу»;

2) внести зміни у необхідні поля (номер телефону, адже зміна email доступна лише адміністратору) та натиснути кнопку «Зберегти» або «Скасувати»;

3) отримати код підтвердження на пошту та ввести його у відповідне поле для підтвердження зміни номеру телефону, натиснути кнопку п

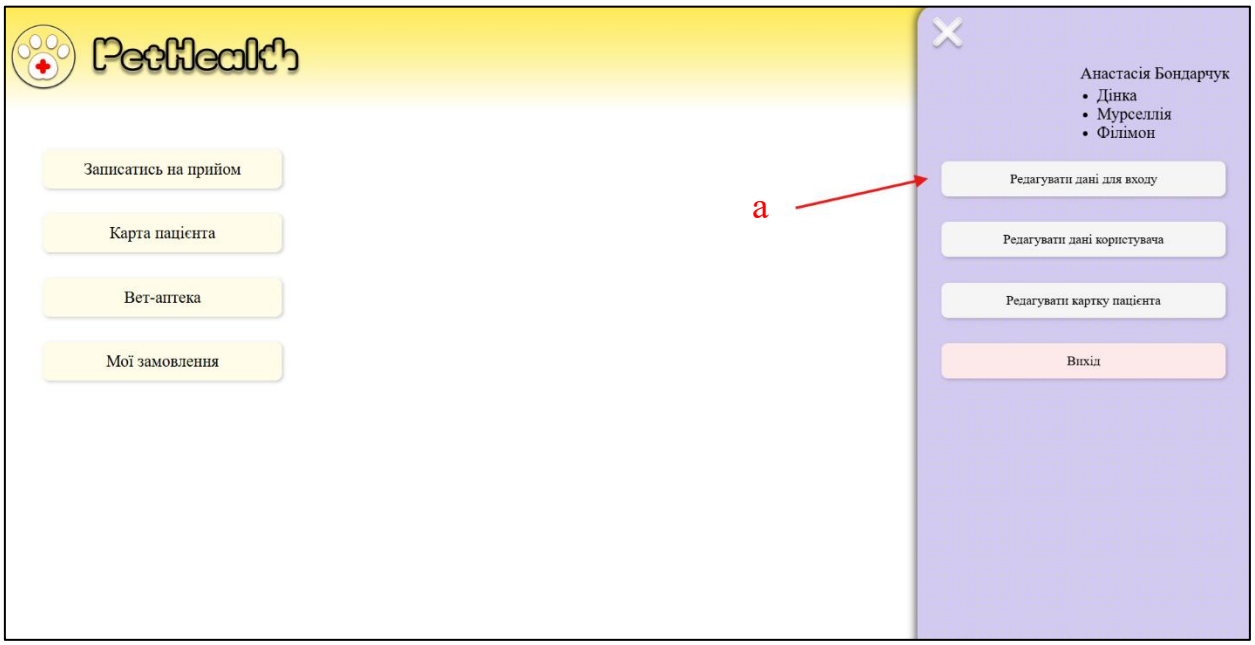


Рисунок 3.8 – Редагування даних для входу ч.1

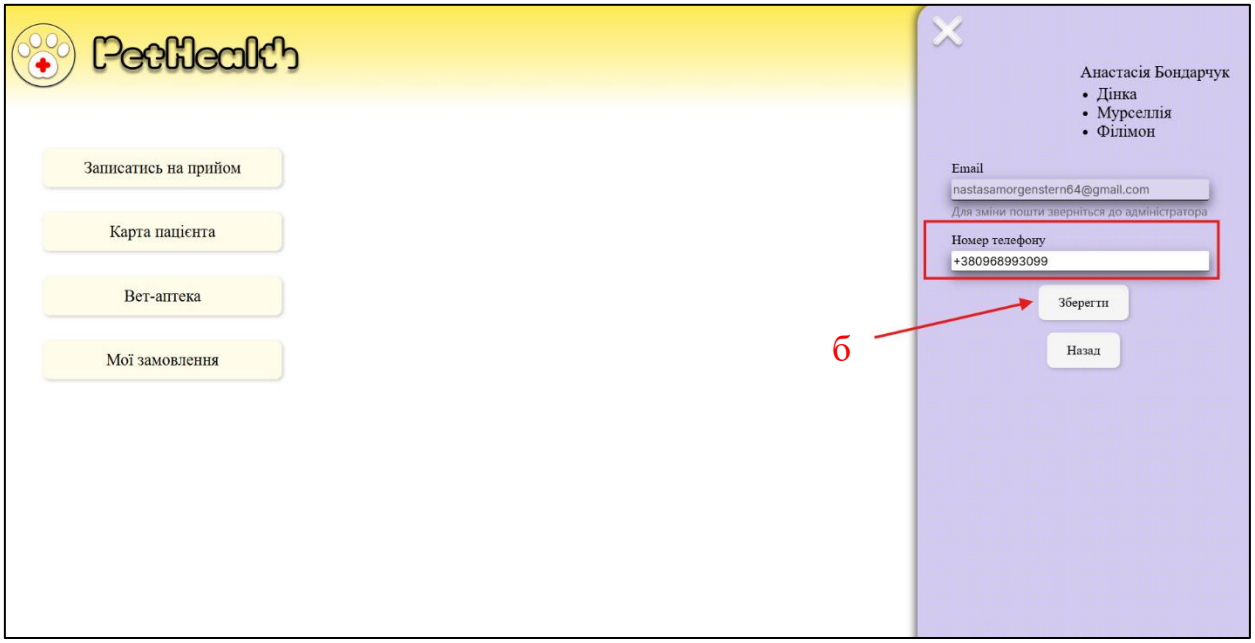


Рисунок 3.9 – Редагування даних для входу ч.2

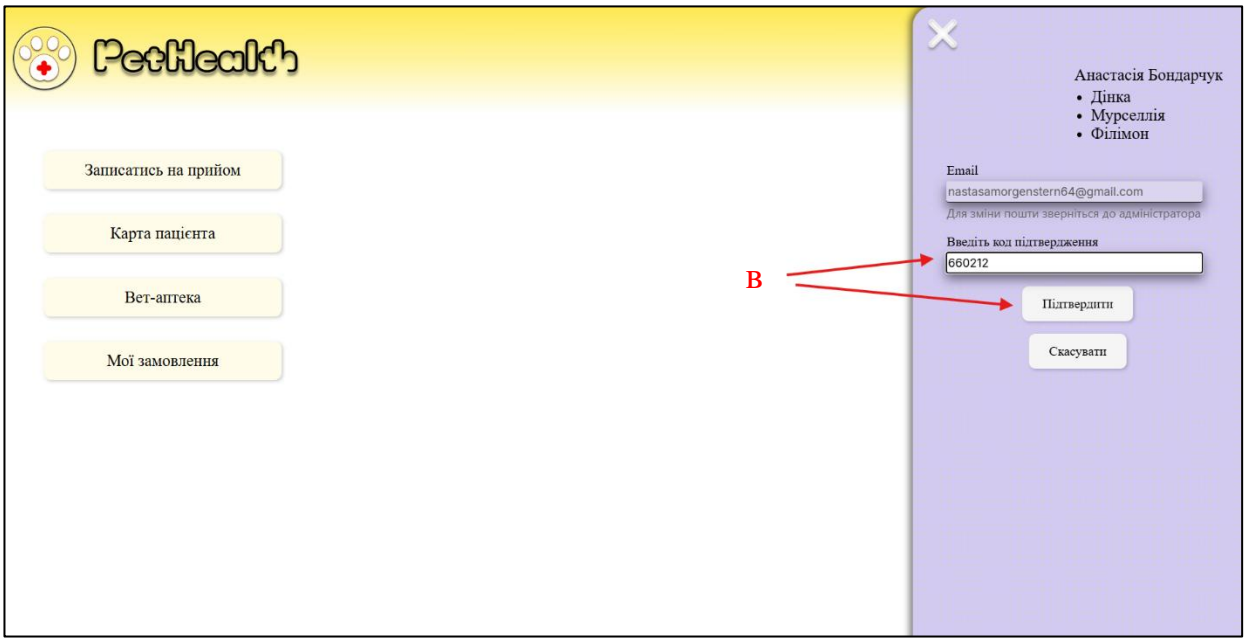


Рисунок 3.10 – Редагування даних для входу ч.3

ж) Для створення, редагування та видалення картки пацієнта:

1) відкрити бокове меню та обрати вкладку «Редагувати картку пацієнта»;

2) обрати картку для редагування або видалення, внести зміни у необхідні поля (ім’я, вид тварини, порода, дата народження) та натиснути кнопку «Зберегти» для збереження, кнопку «Скасувати», якщо не потрібно зберігати зміни, або кнопку «Видалити», якщо потрібно видалити картку тварини;

3) натиснути кнопку «Додати картку», внести дані у необхідні поля (ім’я, вид тварини, порода, дата народження) кнопку «Зберегти» або «Скасувати», якщо не потрібно зберігати зміни.

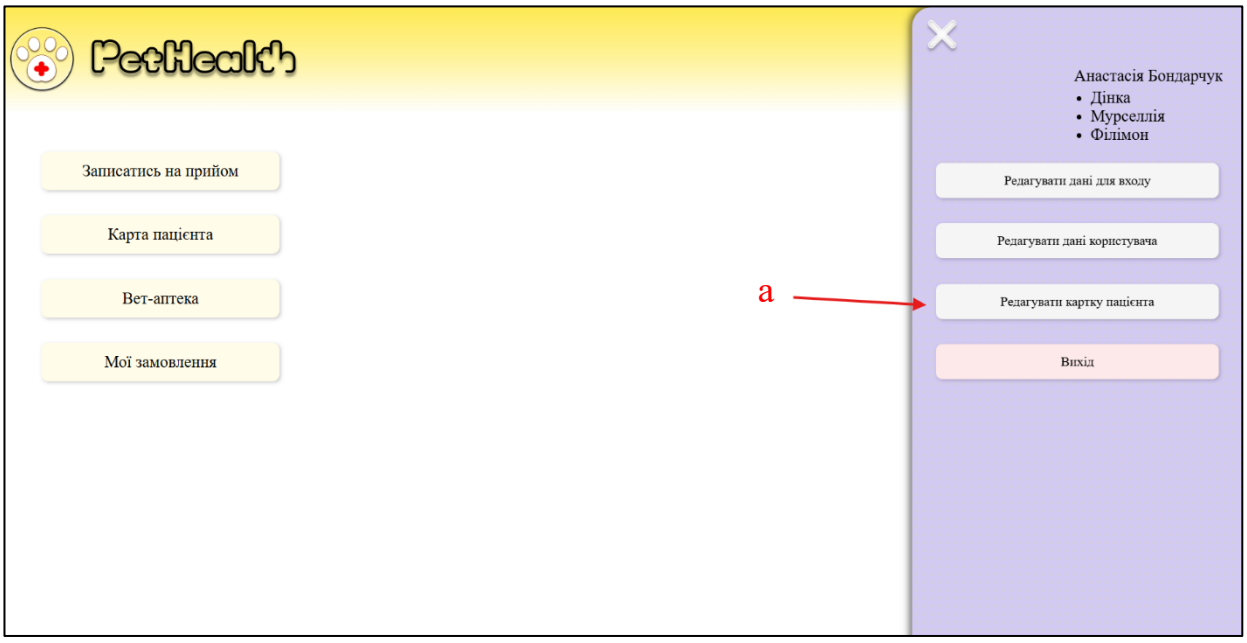


Рисунок 3.11 – Створення редагування та видалення даних пацієнта ч.1

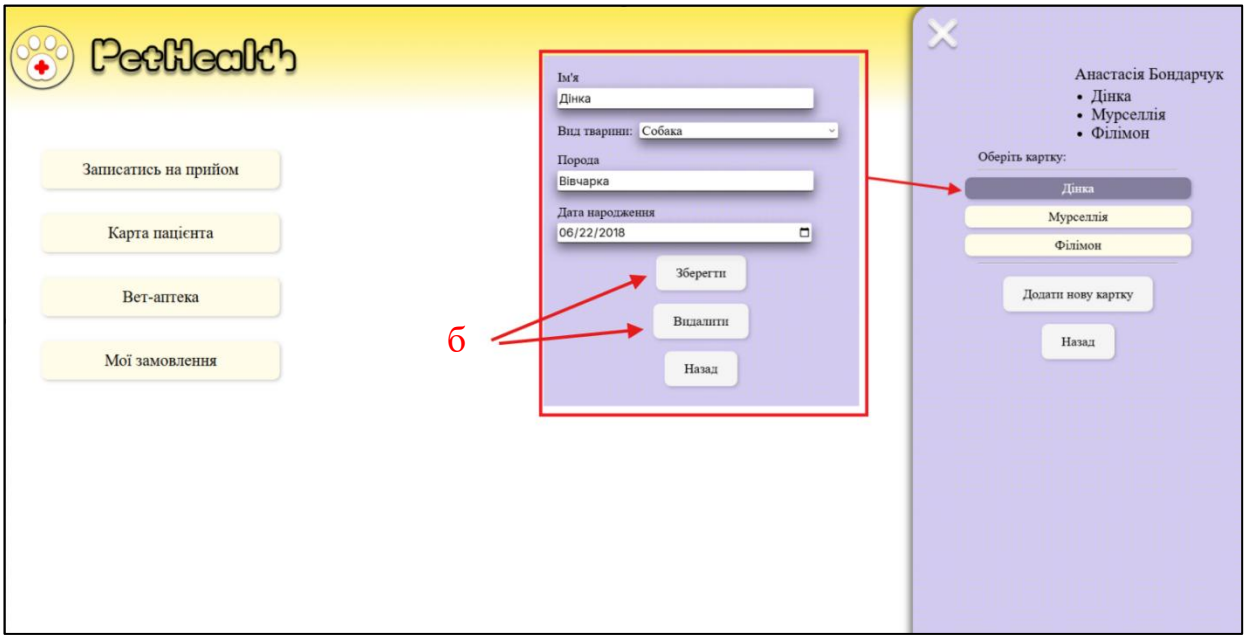


Рисунок 3.12 – Створення редагування та видалення даних пацієнта ч.2

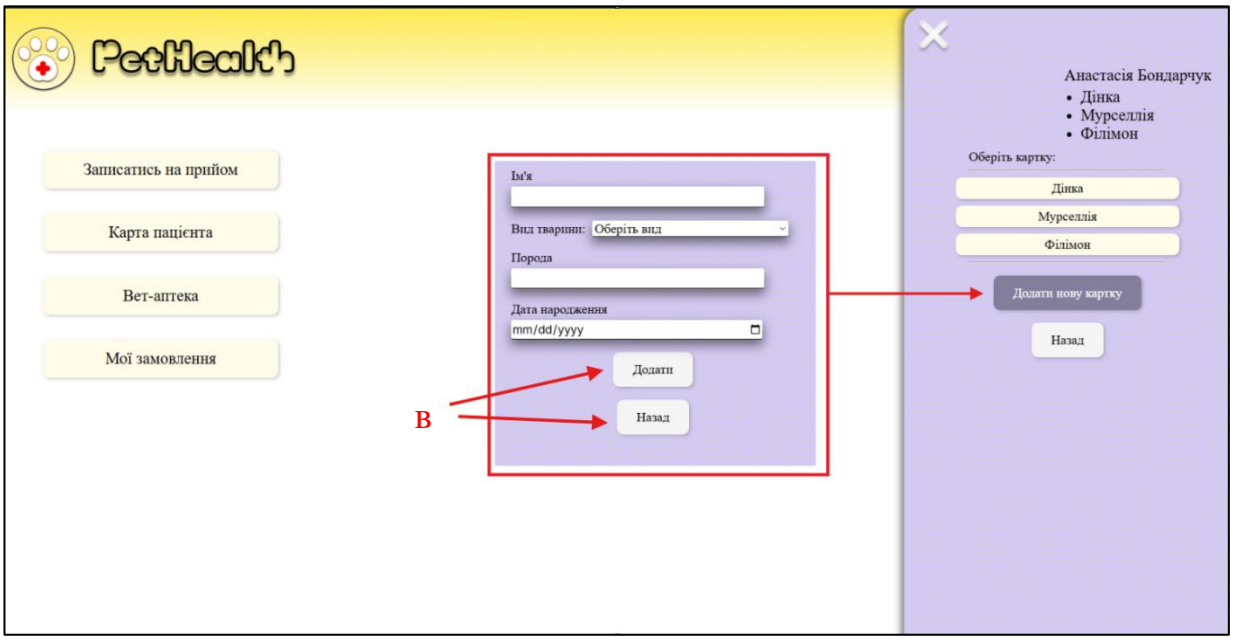


Рисунок 3.13 – Створення редагування та видалення даних пацієнта ч.3

з) Для створення запису на прийом (рис. 3.14-3.16):

1) перейти на вкладку «Записатись на прийом»;

2) обрати зі списку тварину, процедуру, дату прийому та обрати час із доступних згенерованих слотів;

3) написати коментар (за необхідності);

4) натиснути кнопку «Створити запис»;

5) отримати на пошту повідомлення про створення запису на прийом.

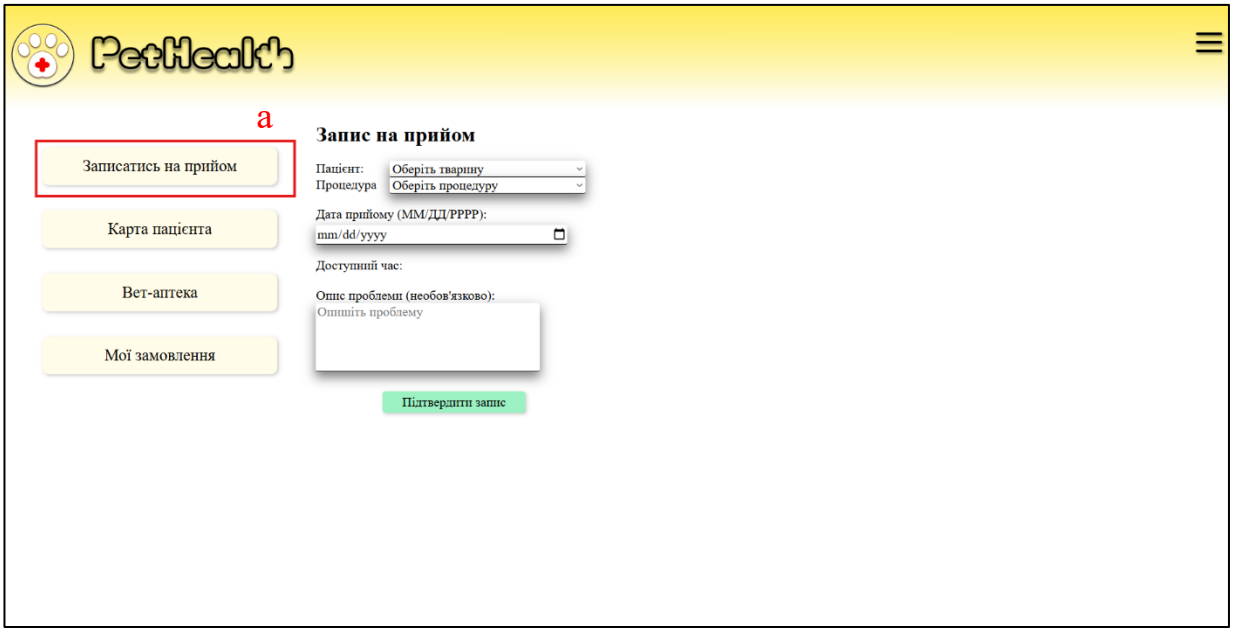


Рисунок 3.14 – Створення запису на прийом ч.1

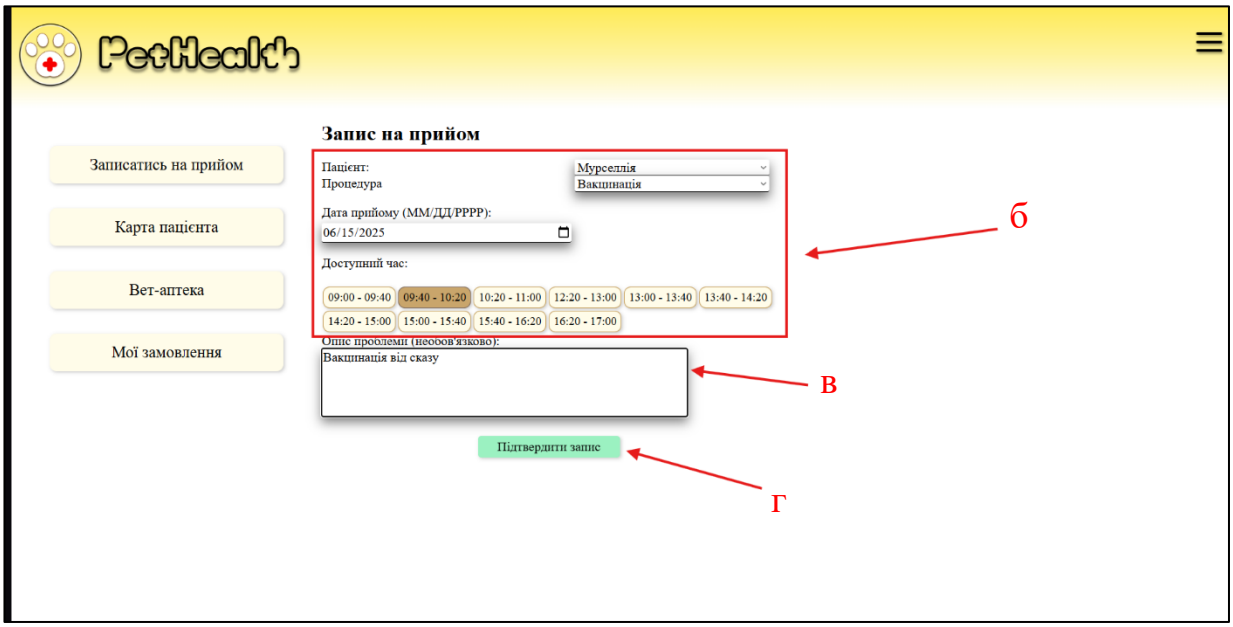


Рисунок 3.15 – Створення запису на прийом ч.2

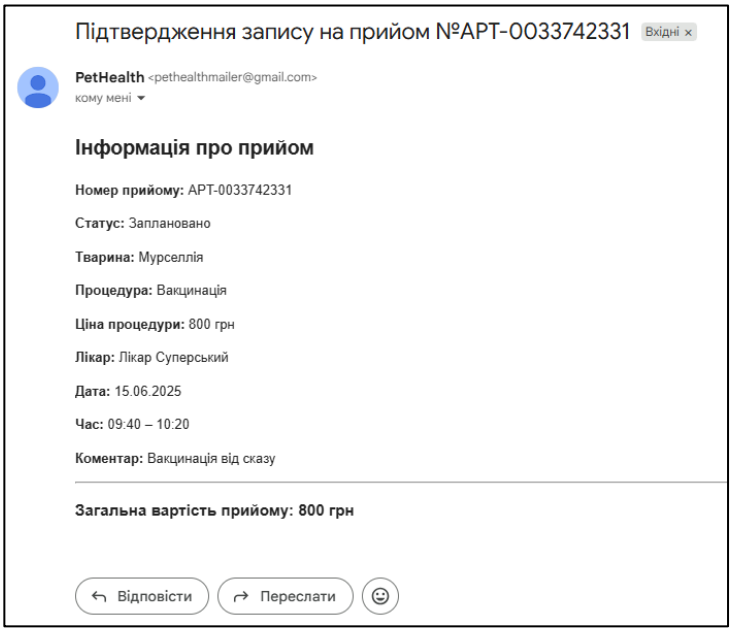


Рисунок 3.16 – Створення запису на прийом ч.3

и) Для перегляду картки пацієнта (рис. 3.16-3.17):

1) перейти на вкладку «Картка пацієнта»;

2) обрати зі списку тварину;

3) на екрані виведеться інформація про обрану тварину зі списком прийомів/записів. Порядок записів: спочатку найближчі за датою та часом. Порядок прийомів: спочатку той прийом, що триває, далі завершений, далі скасований, додатково упорядковані за спаданням дат.

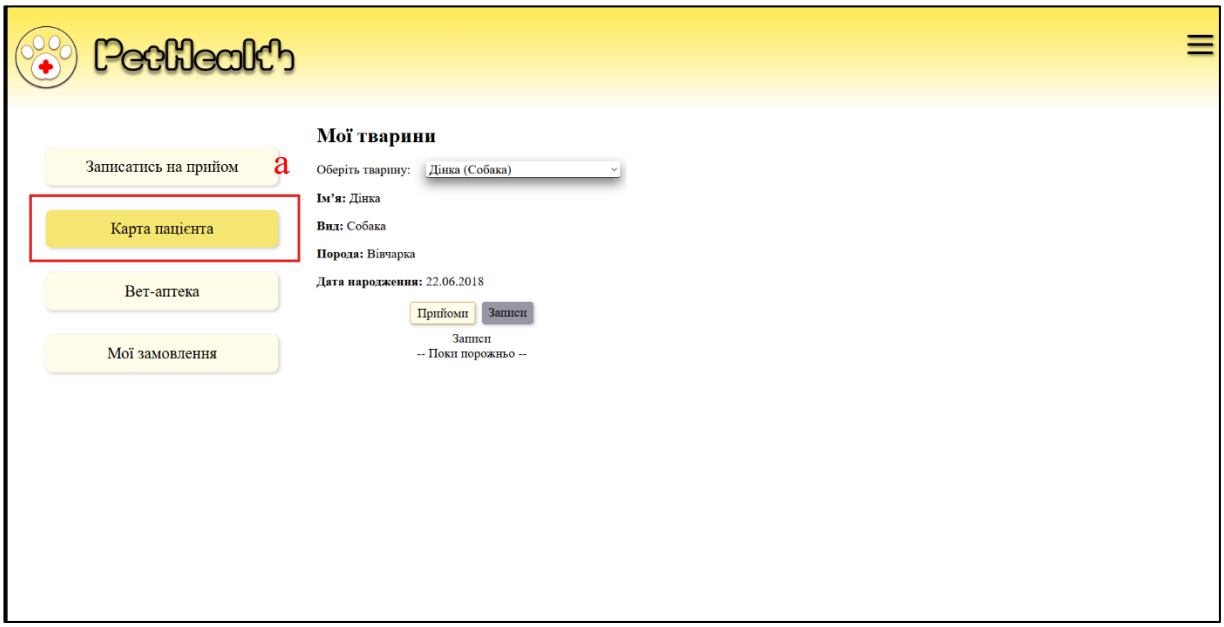


Рисунок 3.17 – Перегляд картки пацієнта ч.2

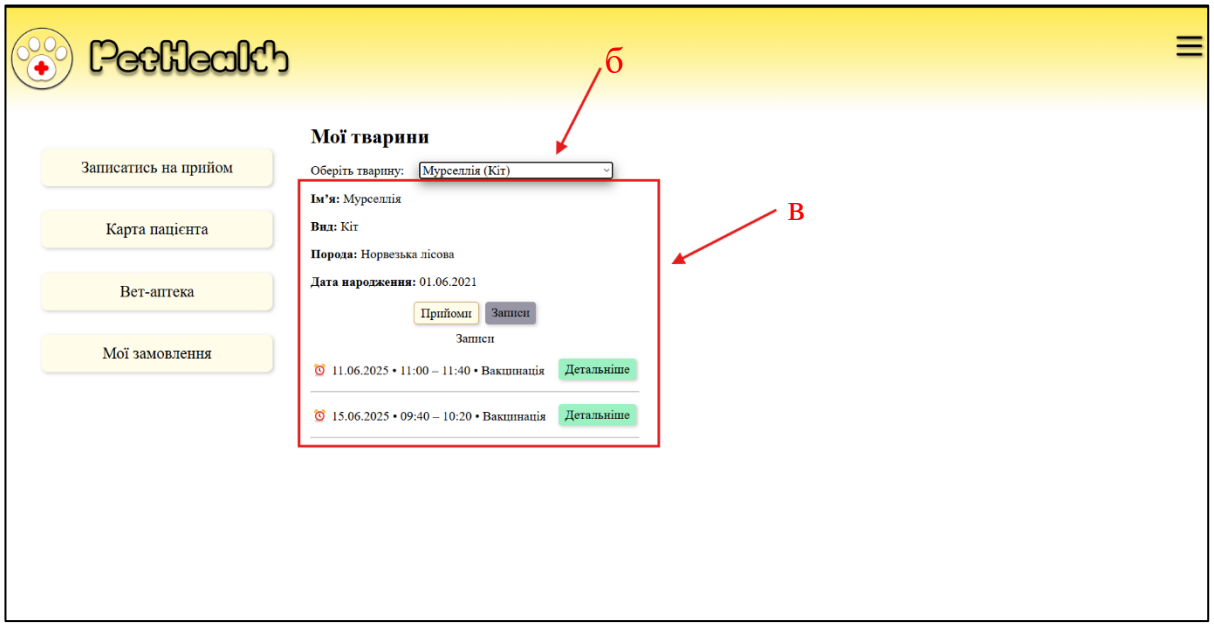


Рисунок 3.18 – Перегляд картки пацієнта ч.2

к) Для перегляду інформації про запис або прийом (рис. 3.19-3.20):

1) відкрити картку пацієнта;

2) знайти в списку необхідний прийом чи запис та натиснути на кнопку «Детальніше» біля нього;

3) на екран виведеться детальна інформація про обраний прийом чи запис;

4) натиснути кнопку «Назад», щоб повернутись до списку.

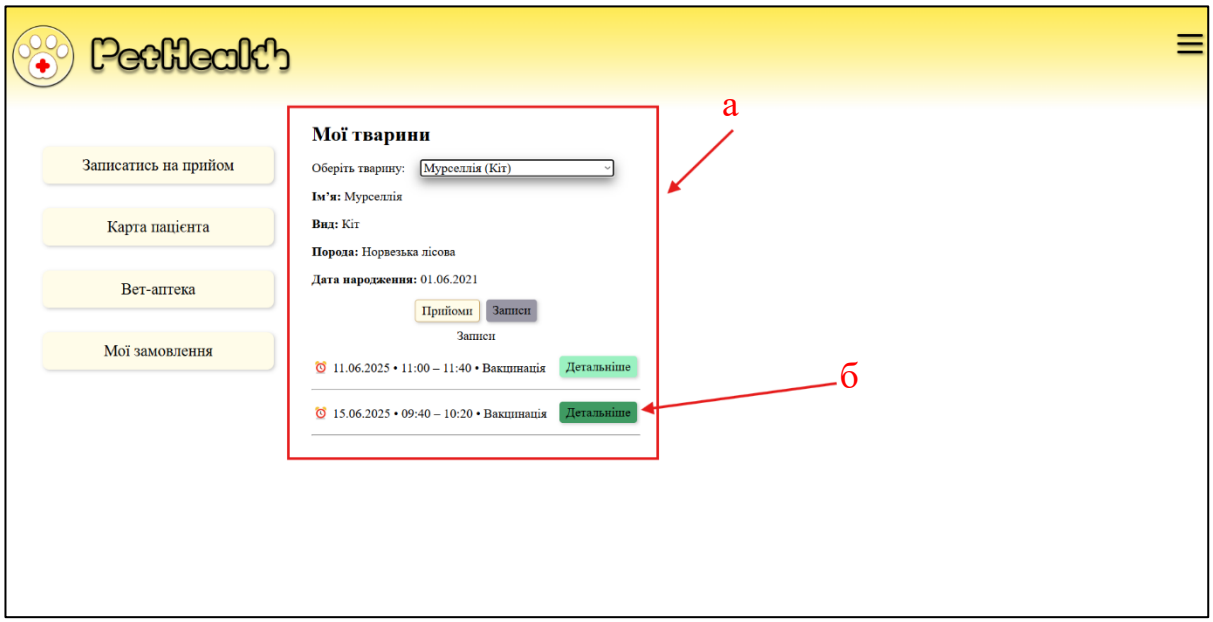


Рисунок 3.19 – Перегляд інформації про запис/прийом ч.1

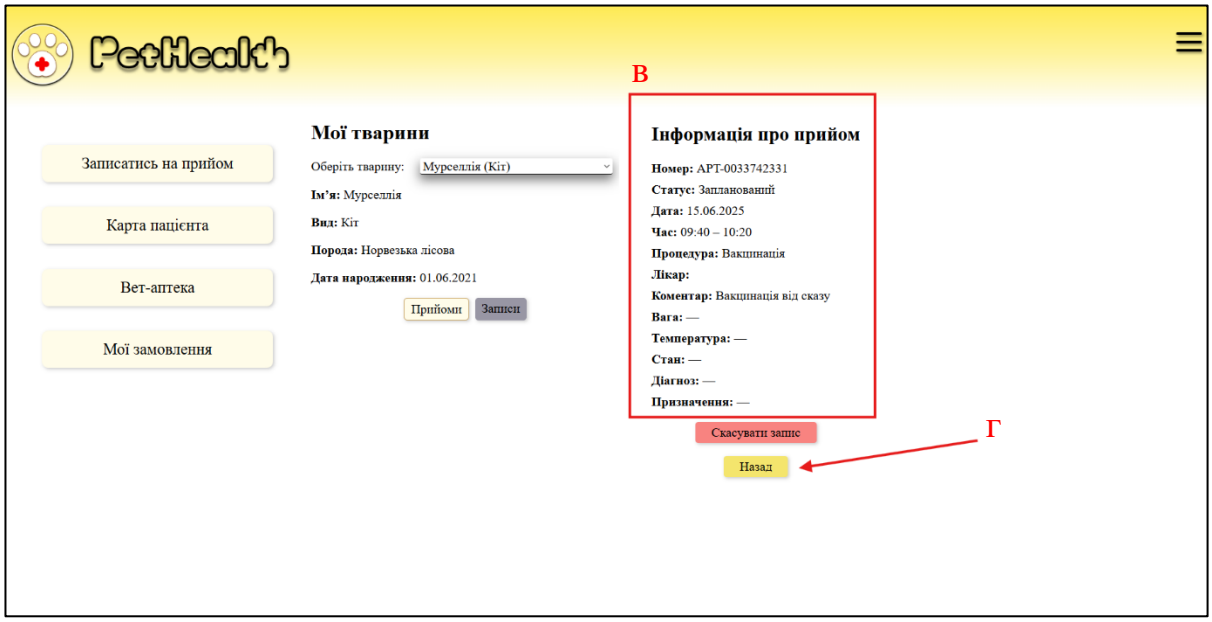


Рисунок 3.20 – Перегляд інформації про запис/прийом ч.2

л) Для скасування запису (рис. 3.21-3.22):

1) відкрити картку пацієнта;

2) знайти в списку необхідний запис та натиснути на кнопку «Детальніше» біля нього;

3) на екран виведеться детальна інформація про обраний прийом чи запис;

4) натиснути на кнопку «Скасувати запис»;

5) якщо до запланованого прийому лишилось більше 12 годин, запис буде успішно скасовано, інакше – виведено повідомлення про неможливість скасування запису.

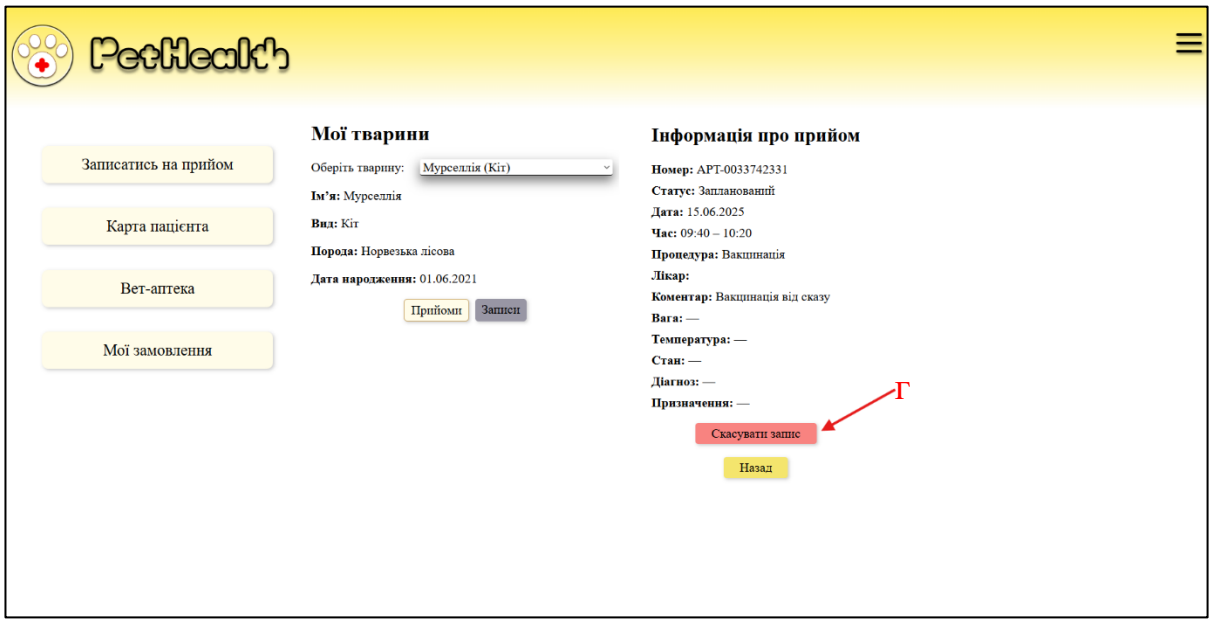


Рисунок 3.21 – Скасування запису ч.1

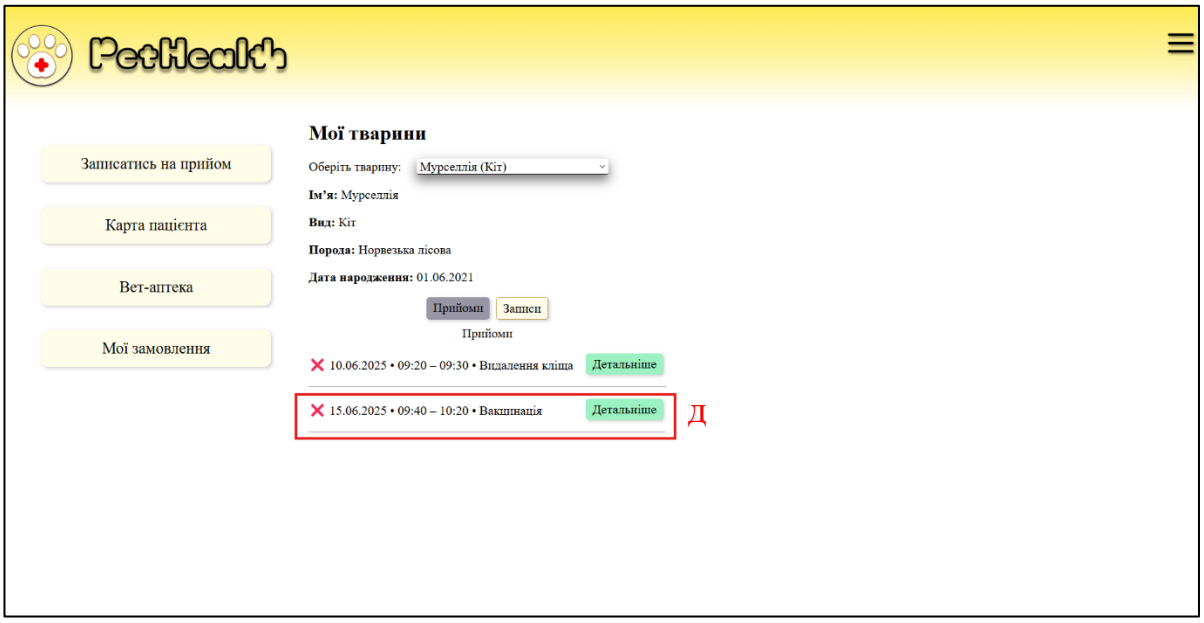


Рисунок 3.22 – Скасування запису ч.2

м) Для перегляду товарів ветаптеки (рис. 3.23):

1) перейти на вкладку «Ветаптека»;

2) ввести в пошук назву товару (за необхідності), обрати фільтри (за необхідності), переглянути список товарів;

3) за потреби натиснути на кнопку «Детальніше» для перегляду повної інформації про товар та кнопку «Сховати» для повернення до списку;

4) за потреби додати обрані товари у кошик.

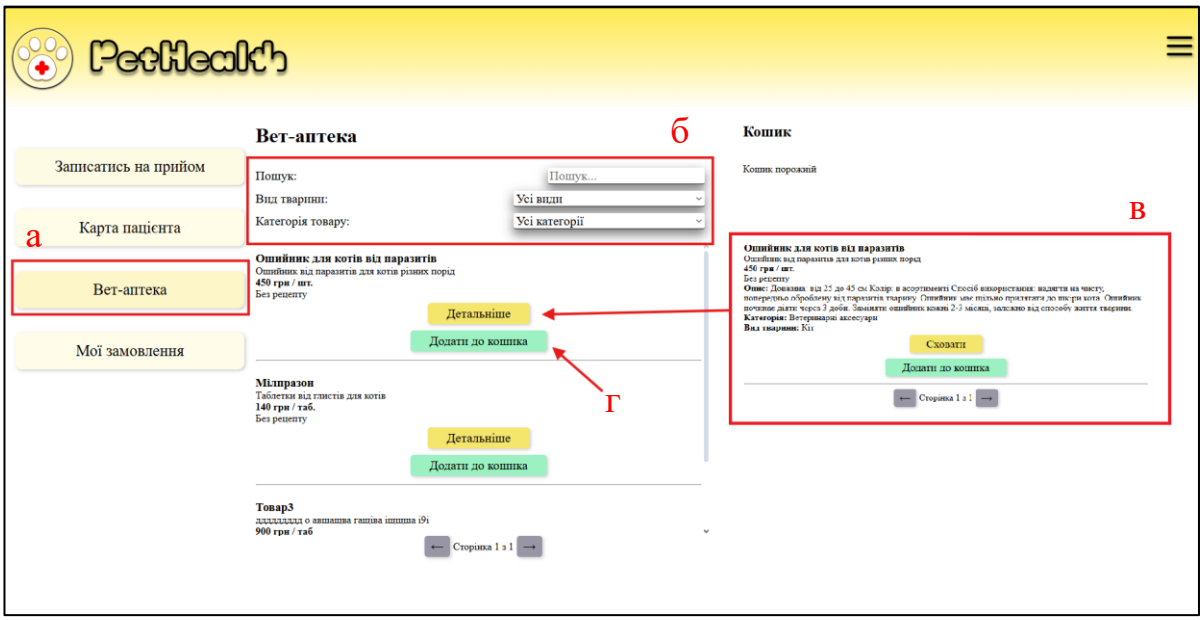


Рисунок 3.23 – Перегляд товарів ветаптеки ч.1

н) Для створення онлайн-замовлення у ветаптеці (рис. 3.24):

1) перейти на вкладку «Ветаптека»;

2) якщо кошик порожній, то додати товари;

3) на екрані з’явиться форма оформлення замовлення;

4) перевірити найменування та кількість товарів;

5) обрати спосіб доставки та оплати;

6) натиснути на кнопку «Оформити замовлення» та «Підтвердити замовлення»;

7) якщо у замовленні немає рецептурних препаратів, то воно створиться та на пошту буде відправлене повідомлення про створення нового замовлення;

8) якщо у замовленні є рецептурні препарати, то з’явиться поле для вибору рецепту на кожен препарат. Обрати рецепти, якщо вони є, та натиснути на кнопку «Підтвердити», та ще раз підтвердити формування замовлення, інакше – натиснути «Скасувати». При відсутності рецепту замовлення оформлене не буде.

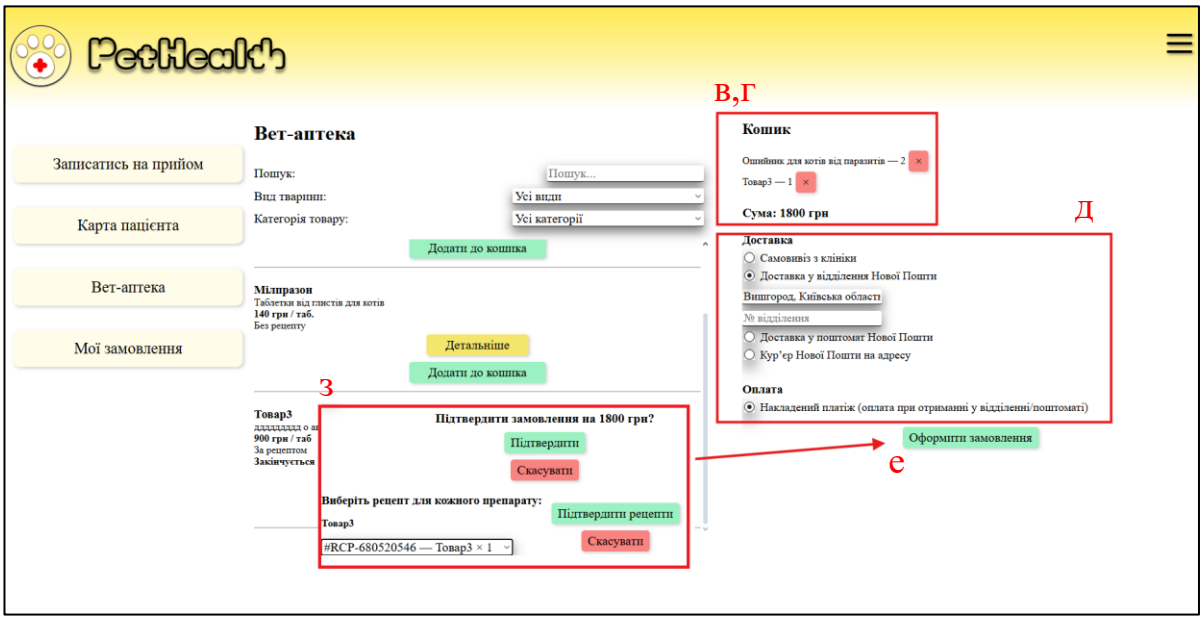


Рисунок 3.24 – Створення онлайн-замовлення у ветаптеці

о) Для перегляду замовлень (рис. 3.25):

1) перейти на вкладку «Мої замовлення»;

2) на екран виведеться список замовлень.

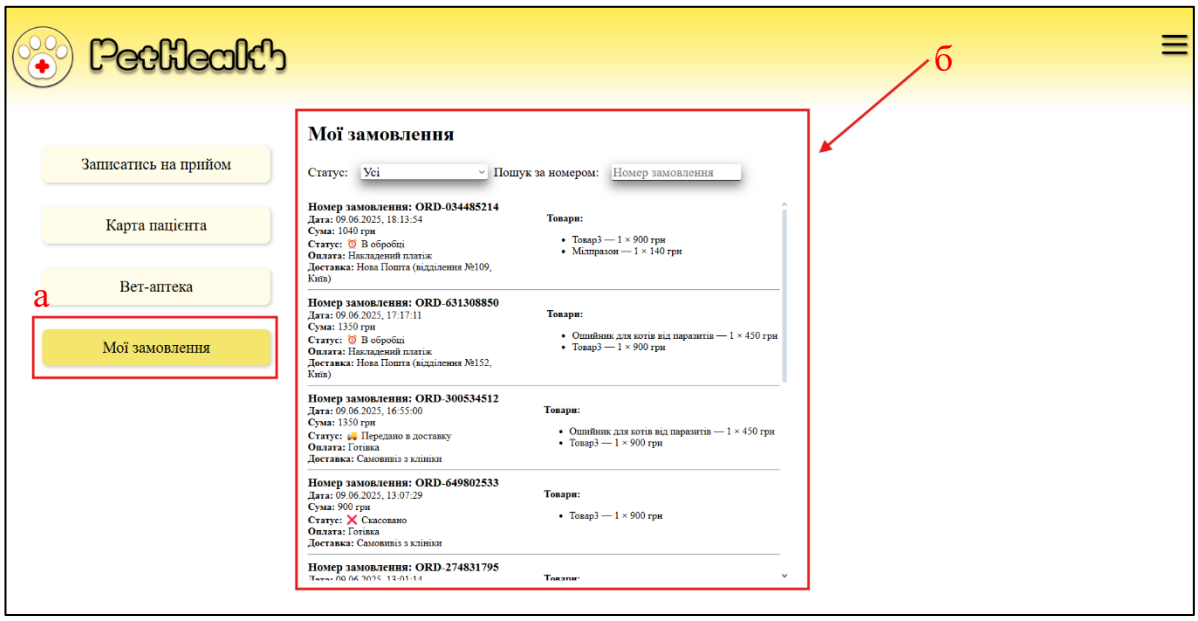


Рисунок 3.25 – Перегляд замовлень

3.2 Інструкція для адміністратора клініки:

1) Відкрити вебзастосунок за посиланням.

2) Увійти до облікового запису адміністратора (рис. 3.26):

а) перейти на форму авторизації;

б) ввести адресу електронної пошти;

в) натиснути кнопку «Код»;

г) ввести отриманий шестизначний код у відповідне поле та натиснути на кнопку «Увійти»;

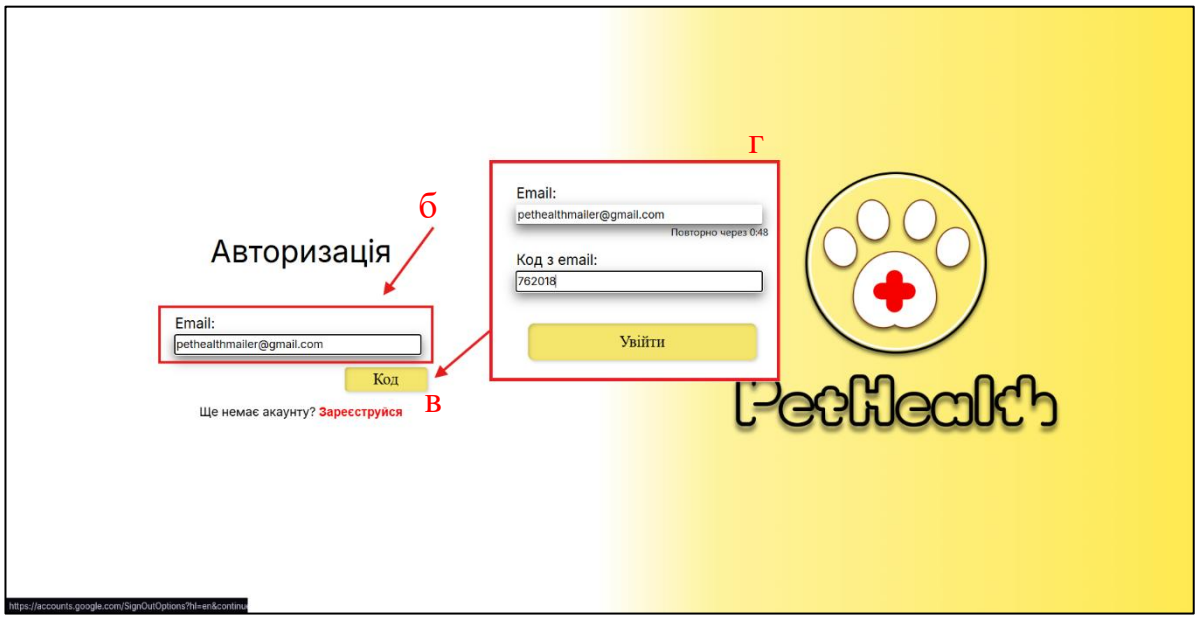


Рисунок 3.26 – Авторизація в застосунку

3) Для редагування особистих даних адміністратора (рис. 3.27):

а) відкрити бокове меню;

б) обрати вкладку «Редагувати дані користувача»;

в) внести зміни в необхідні поля (ім’я, прізвище, дата народження) та натиснути «Зберегти» або «Скасувати».

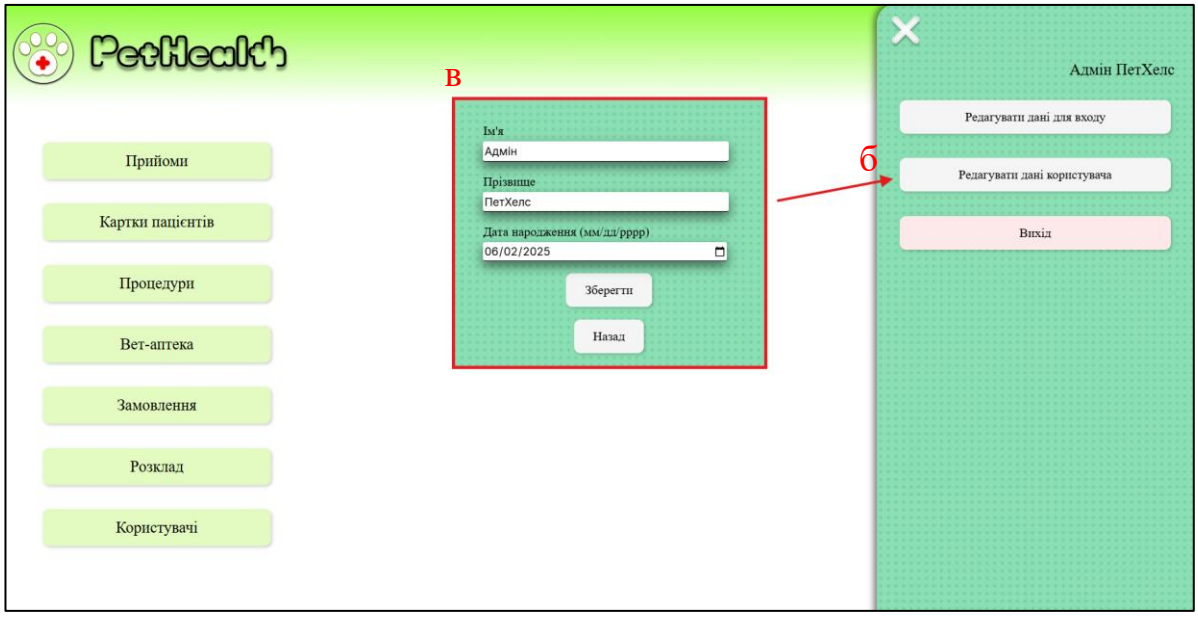


Рисунок 3.27 – Редагування особистих даних адміністратора

4) Для редагування даних для входу (рис. 3.28):

а) відкрити бокове меню;

б) обрати вкладку «Редагувати дані для входу»;

в) внести зміни в необхідні поля (номер телефону, email);

г) отримати на пошту шестизначний код підтвердження та ввести його у відповідне поле та натиснути «Зберегти» або «Скасувати».

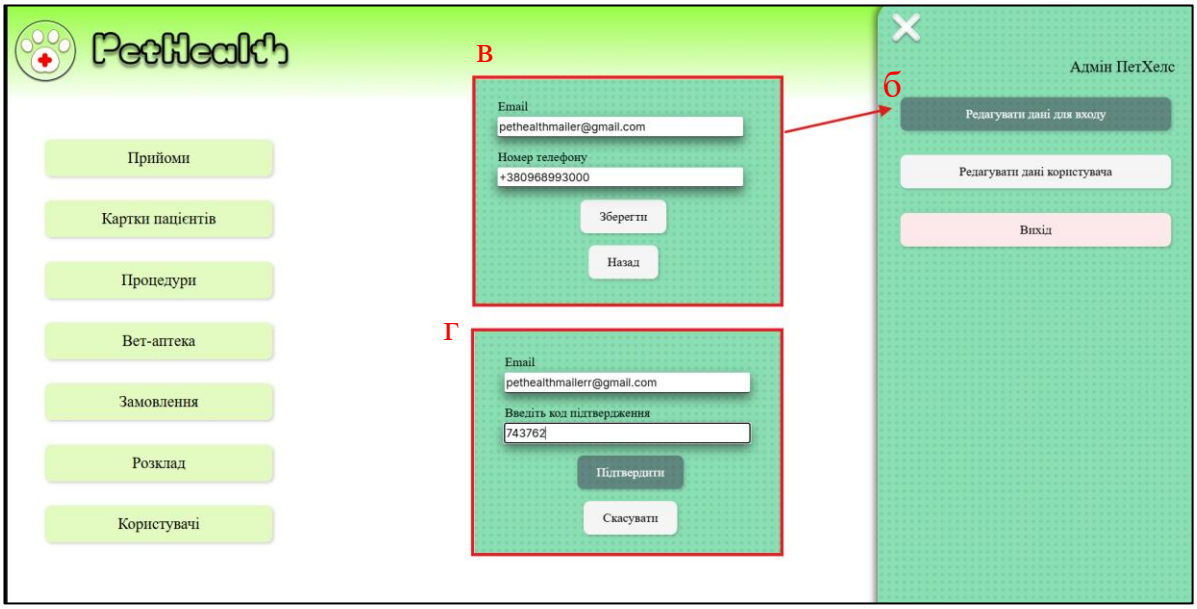


Рисунок 3.28 – Редагування даних для входу

5) Для управління картками пацієнтів (рис. 3.29-3.31):

а) перейти на вкладку «Картки пацієнтів»;

б) фільтрувати картки пацієнтів за відповідними полями (власник, вид тварини, архів/активні);

в) натиснути «Детальніше» біля обраної картки пацієнта для перегляду детальної інформації;

г) у режимі перегляду натиснути «Редагувати» та внести зміни у необхідні поля (ім’я, порода, вид тварини, дата народження) та натиснути «Зберегти» або «Скасувати»;

д) для архівації картки натиснути «В архів»;

е) для перегляду архівованих карток натиснути «Архів», де можна їх відновлювати натисканням на «Відновити» та видаляти натисканням на «Видалити»;

ж) для додавання нової картки заповнити необхідні поля (ім’я, вид тварини, порода, дата народження), обрати власника та натиснути кнопку «Додати картку».

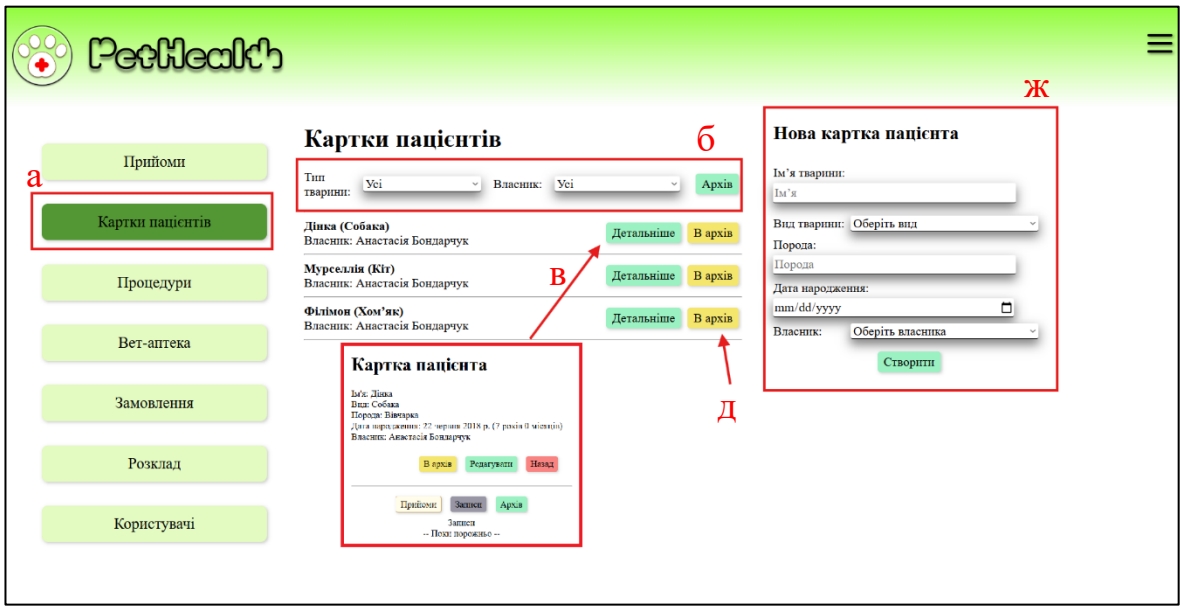


Рисунок 3.29 – Управління картками пацієнтів ч.1

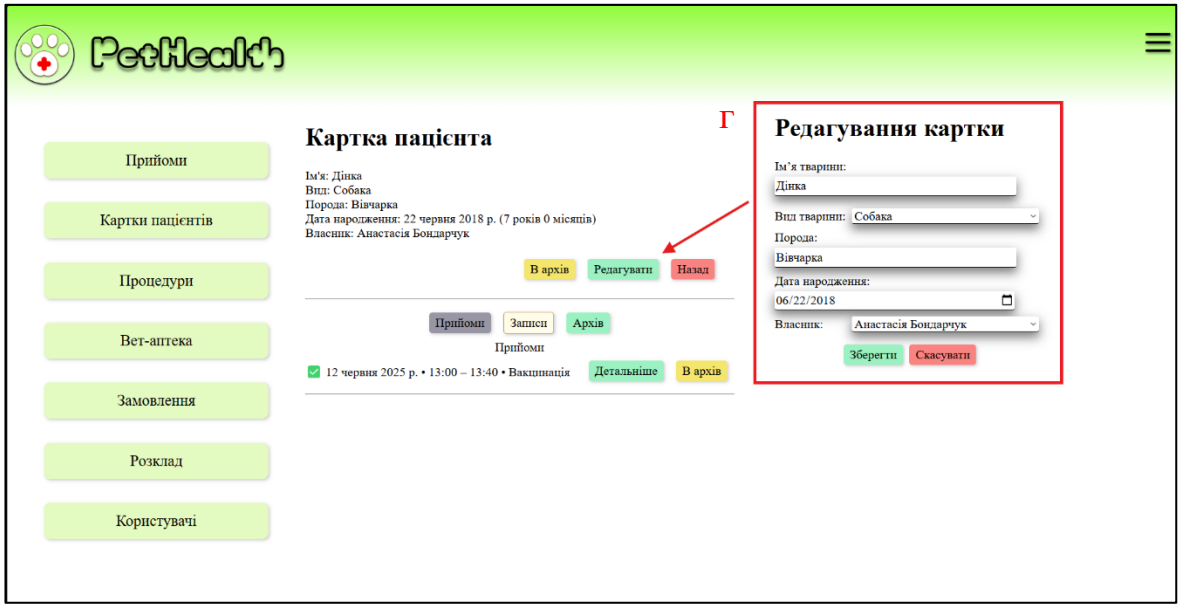


Рисунок 3.30 – Управління картками пацієнтів ч.2

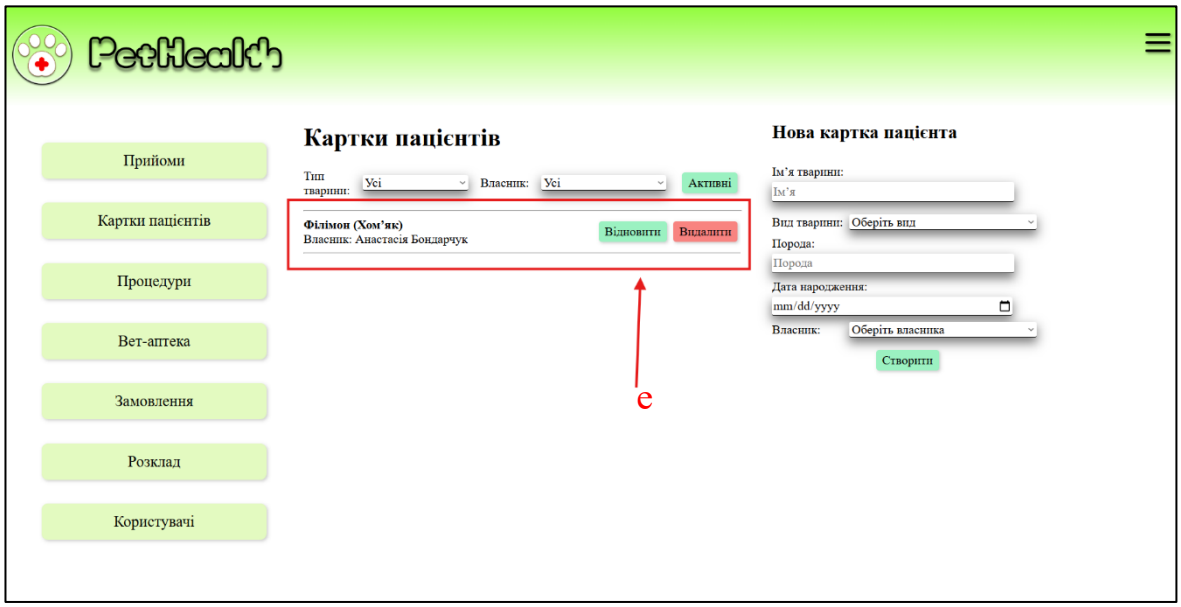


Рисунок 3.31 – Управління картками пацієнтів ч.3

6) Для управління прийомами (рис. 3.32-3.33):

а) перейти на вкладку «Прийоми»;

б) фільтрувати прийоми за необхідності за полями (прийоми/записи, дата, тип тварини, активні/архівні) та за пошуком по імені власника, тварини або за номером прийому;

в) у списку прийомів натиснути «Детальніше» для перегляду обраного прийому;

г) натиснути «Редагувати» для внесення медичних даних (статус, температура, вага та стан пацієнта, діагноз, призначення, список використаних медикаментів (обирається зі списку), рецепт (додається або редагується)) та натиснути «Зберегти» або «Скасувати».

д) для архівації прийому натиснути «В архів»;

е) для перегляду архівованих прийомів натиснути «Показати архів», де можна їх відновлювати натисканням на «Відновити» та видаляти натисканням на «Видалити»;

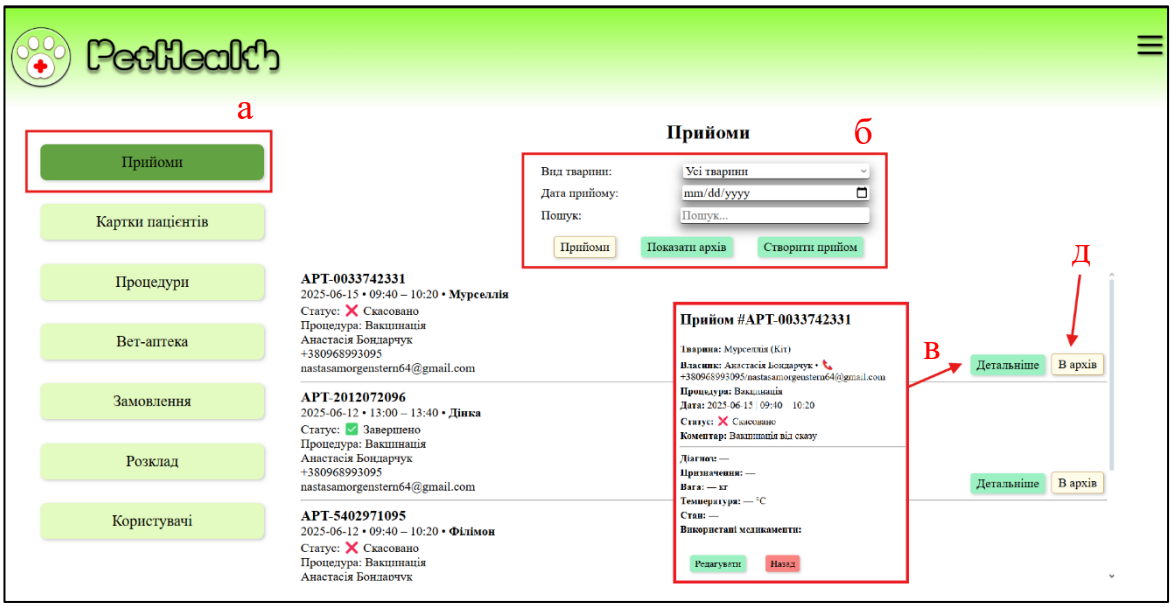


Рисунок 3.32 – Управління прийомами ч.1

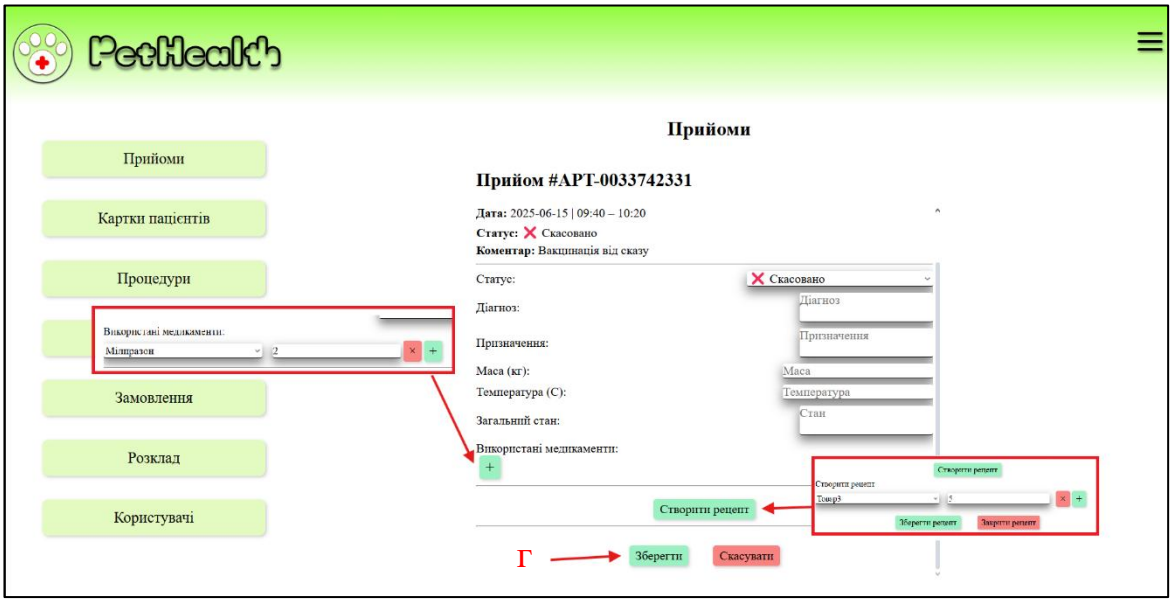


Рисунок 3.33 – Управління прийомами ч.2

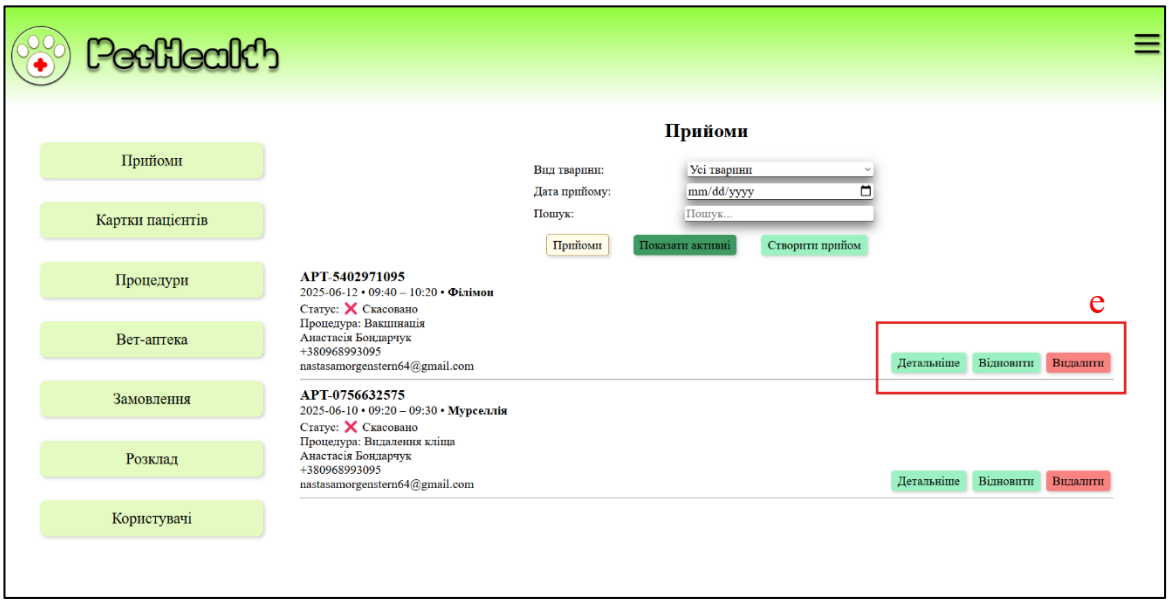


Рисунок 3.34 – Управління прийомами ч.3

7) Для створення нового запису (рис. 3.35):

а) перейти на вкладку «Прийоми»;

б) натиснути «Створити прийом»;

в) обрати у відповідних полях дані для створення прийому (власник, пацієнт, процедура, дата та вільний слот часу), за потреби ввести коментар та натиснути «Створити прийом» або «Назад».

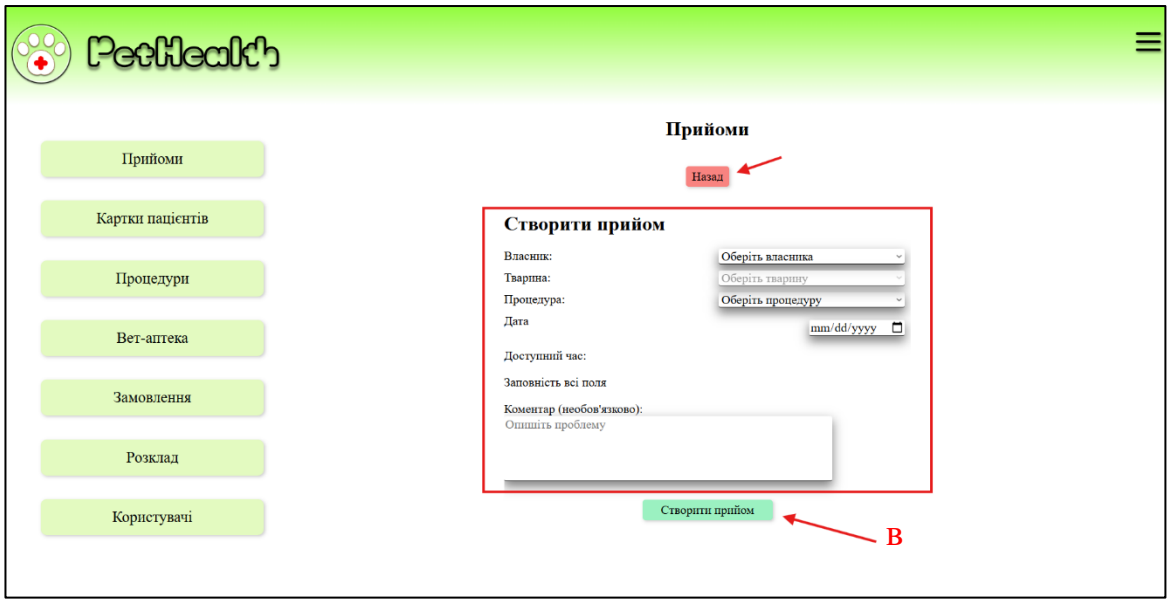


Рисунок 3.35 – Створення прийому

8) Для управління графіком роботи клініки (рис. 3.36):

а) перейти на вкладку «Розклад»;

б) для кожного дня тижня вказати необхідні дані (вихідний/робочий, години початку та кінця робочого дня, наявність обідньої перерви, години початку та кінця обідньої перерви, операційний/звичайний, години початку та кінця операцій в операційний день);

в) натиснути «Зберегти графік»;

г) якщо при зміні графіку лишились записи у закриті дні, користувачам надійде повідомлення про скасування записів, а відповідні записи будуть скасовані, на сторінці графіку з’явиться список конфліктних записів.

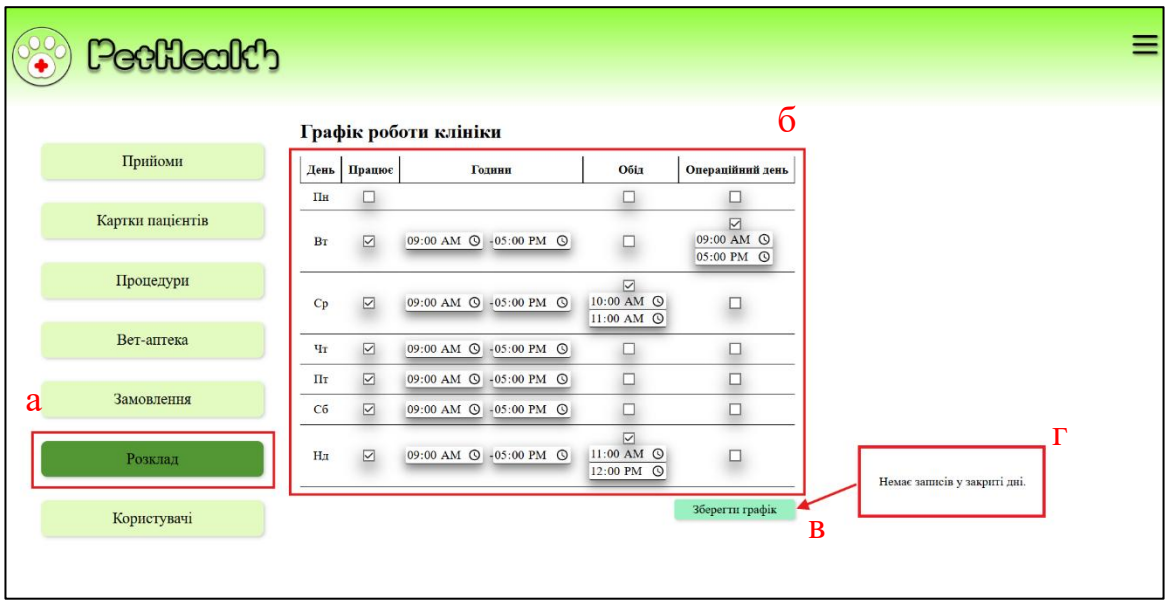


Рисунок 3.36 – Управління графіком роботи клініки

9) Для управління користувачами (рис. 3.37-3.38):

а) перейти на вкладку «Користувачі»;

б) фільтрувати список за роллю користувача (адміністратор, лікар, клієнт) та архівні/активні, виведеться список усіх користувачів, окрім активного користувача;

в) для редагування натиснути кнопку «Редагувати» біля обраного користувача та змінити необідні дані (ім’я, прізвище, email, номер телефону) та задати роль (клієнт, лікар, адмін) та натиснути кнопку «Зберегти» або «Скасувати»;

г) для створення нового користувача у формі ввести дані (ім’я, прізвище, email, номер телефону) та задати роль (клієнт, лікар, адмін) та натиснути кнопку «Зберегти»;

д) для архівації користувача натиснути кнопку «В архів»;

е) при натисканні кнопки «Показати архів» відкривається архів, де можна відновити користувача натисканням на «Відновити» та видалити натисканням на «Видалити»;

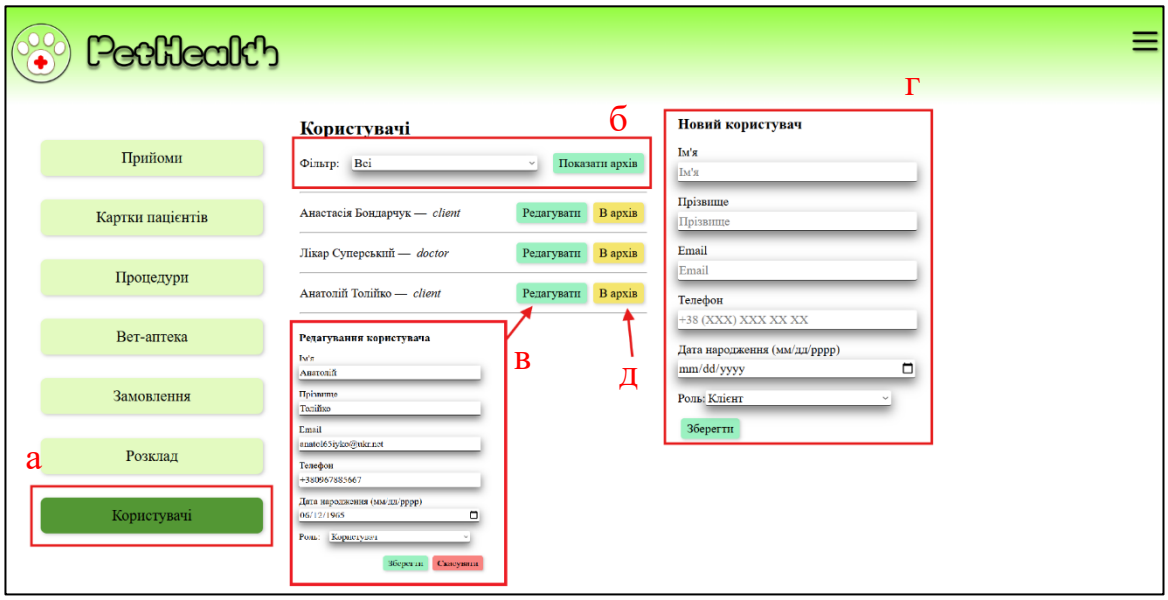


Рисунок 3.37 – Управління користувачами ч.1

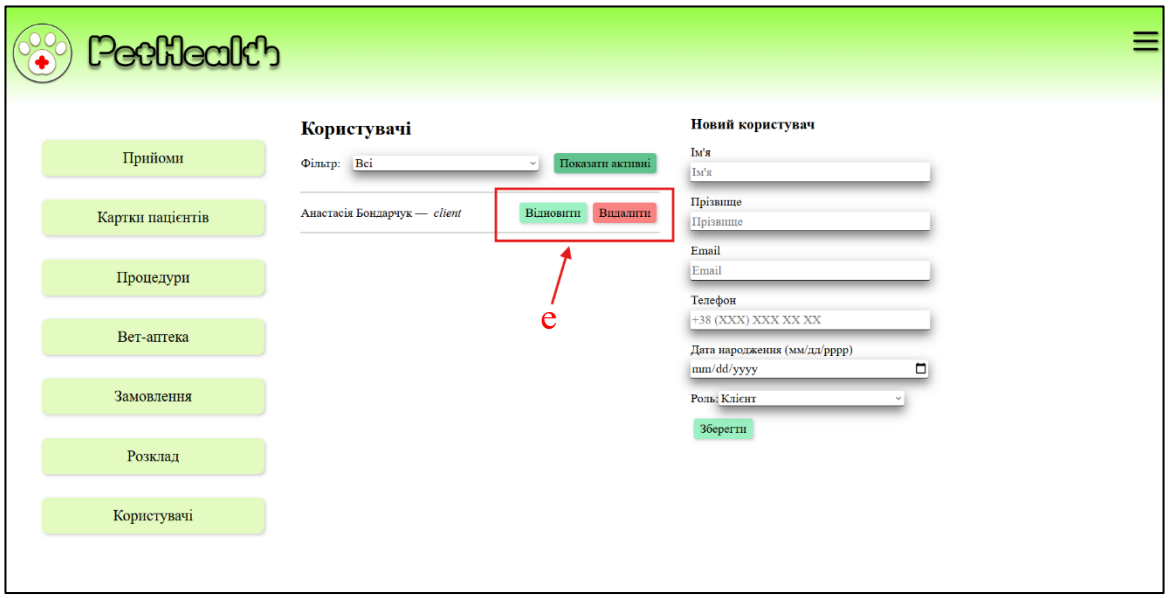


Рисунок 3.38 – Управління користувачами ч.2

10) Для управління процедурами (рис 3.39-3.40):

а) перейти на вкладку «Процедури»;

б) для редагування процедури натиснути кнопку «Редагувати» біля обраної процедури у списку, внести необхідні дані у поля (назва, вид тварини, лікар, вартість, тривалість, можливість самозапису) та натиснути кнопку «Зберегти» або «Скасувати»;

в) для створення нової процедури внести дані у поля в формі (назва, вид тварини, лікар, вартість, тривалість, можливість самозапису) та натиснути кнопку «Зберегти»;

г) для архівації процедури натиснути кнопку «В архів» біля обраної процедури;

д) переглянути архів натисканням на кнопку «Показати архівовані», де можна відновити процедуру натисканням на «Відновити» та видалити натисканням на «Видалити».

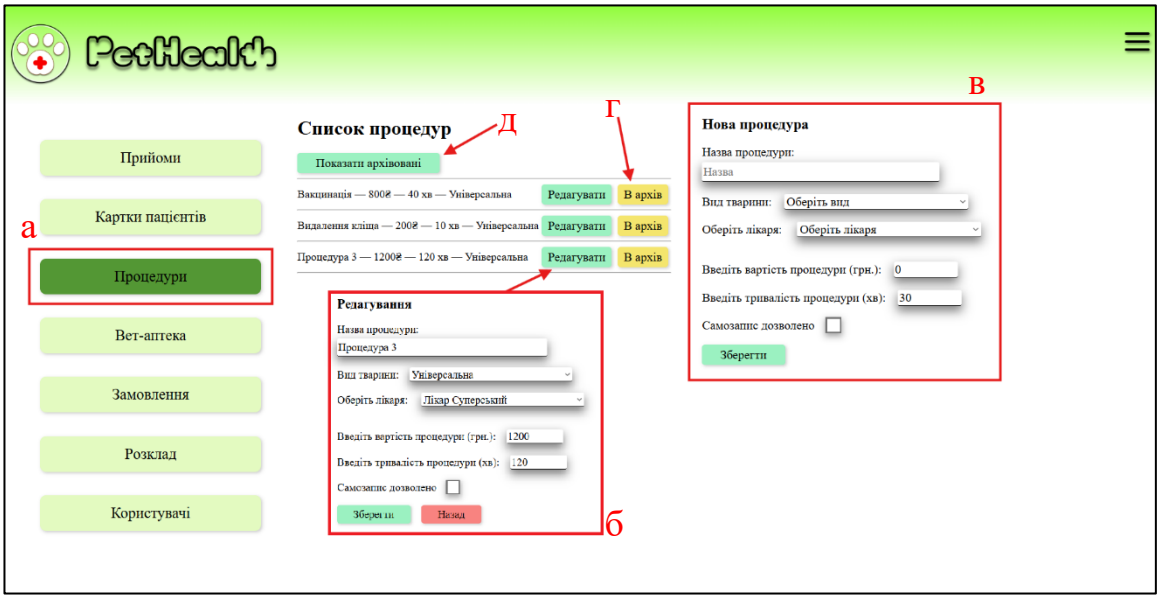


Рисунок 3.39 – Управління процедурами ч.1

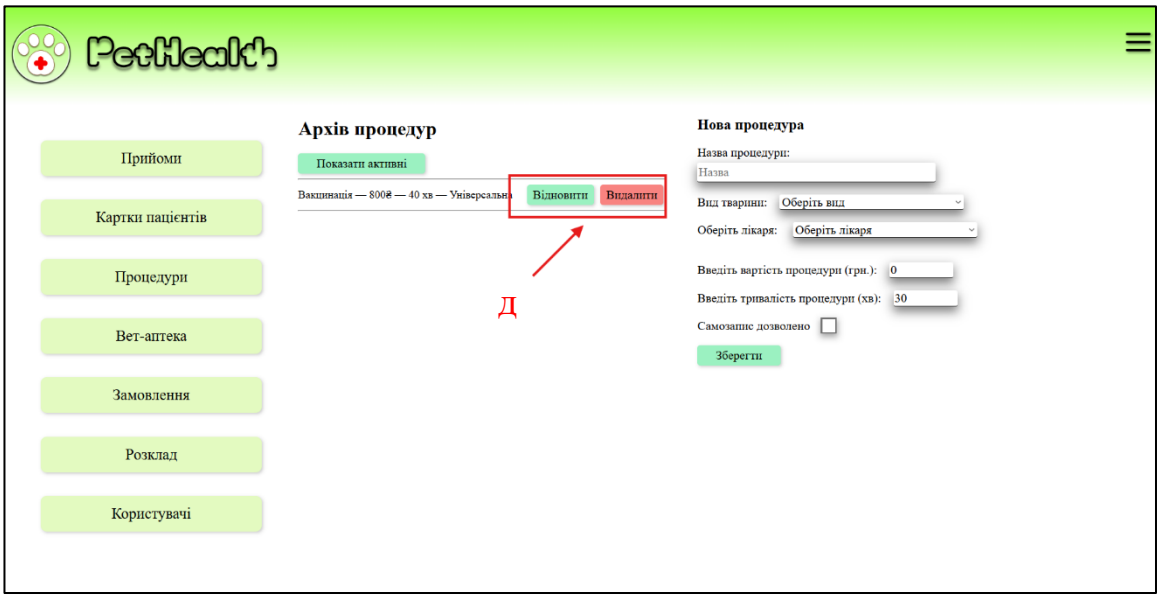


Рисунок 3.40 – Управління процедурами ч.2

11) Для управління товарами у ветаптеці (рис. 3.41-3.43):

а) перейти на вкладку «Ветаптека»;

б) фільтрувати товари за необхідними полями (вид тварини, категорія, архівні/активні) та за пошуком;

в) для редагування натиснути кнопку «Редагувати» біля обраного товару та ввести необхідні дані (назва, короткий опис, довгий опис, вид тварини, категорія, продаж з рецептом чи ні, одиниця виміру, кількість на складі, ціна за одиницю) та натиснути «Зберегти зміни»;

г) для додавання категорії ввести в поле назву нової категорії та натиснути кнопку «Додати»;

д) для редагування категорії натиснути на кнопку з олівцем, ввести нову назву та натиснути кнопку «Оновити»;

е) для видалення категорії натиснути кнопку з відром та підтвердити видалення;

ж) для додавання товару внести у форму дані (назва, короткий опис, довгий опис, вид тварини, категорія, продаж з рецептом чи ні, одиниця виміру, кількість на складі, ціна за одиницю) та натиснути «Додати товар»;

з) для архівації товару натиснути кнопку «В архів» біля обраного товару в списку;

и) для перегляду архівованих товарів натиснути «Показати архівні», де можна відновити товар натисканням на «Відновити» та видалити натисканням на «Видалити».

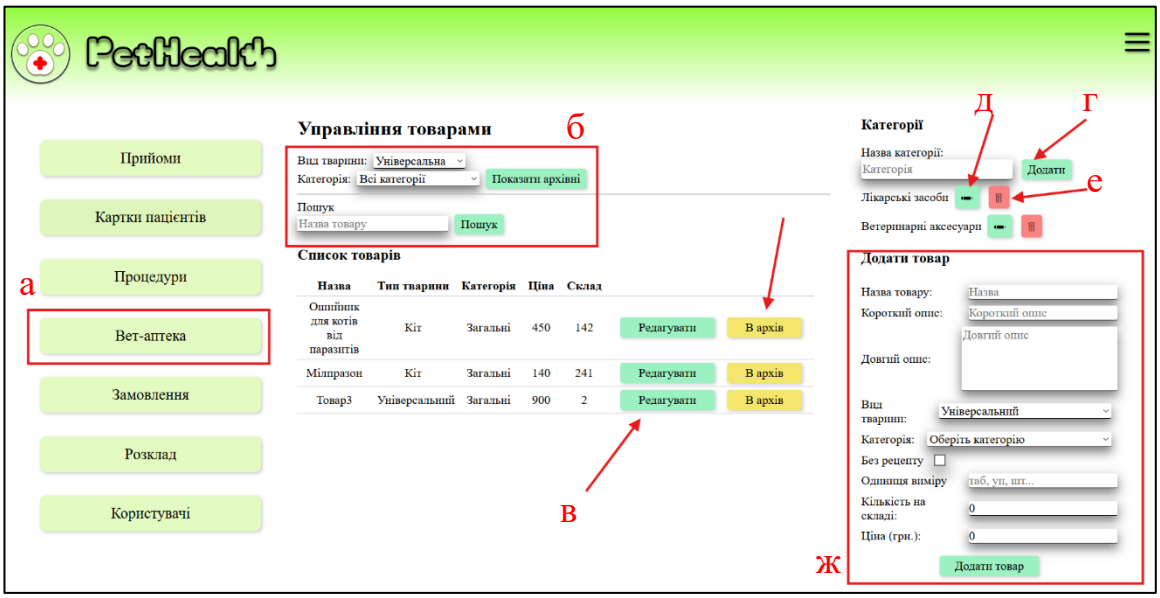


Рисунок 3.41 – Управління товарами у ветаптеці ч.1

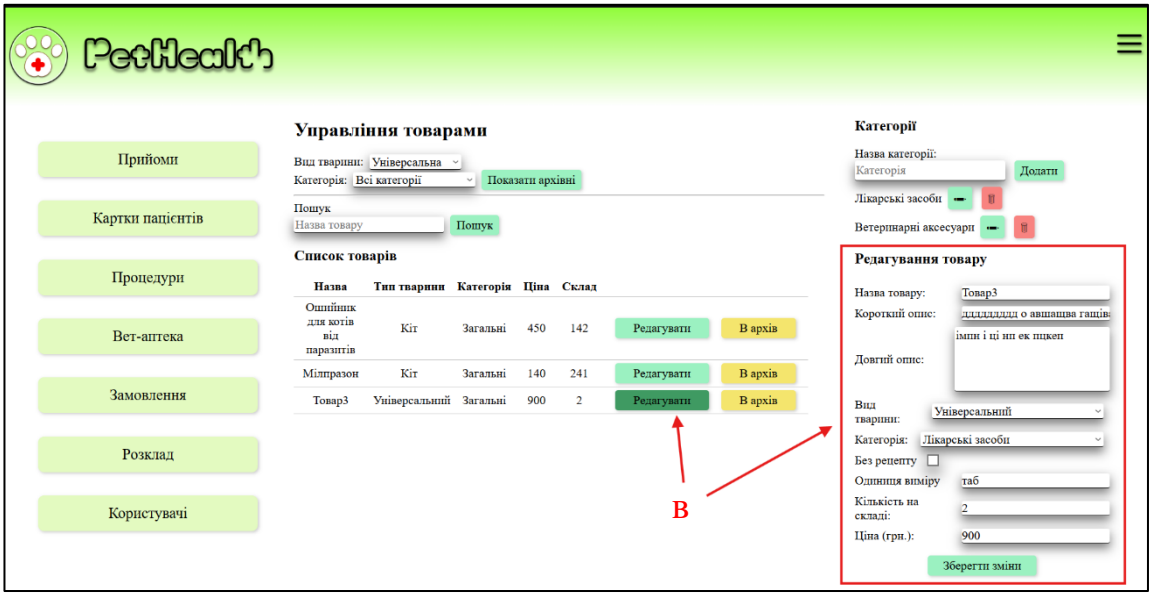


Рисунок 3.42 – Управління товарами у ветаптеці ч.2

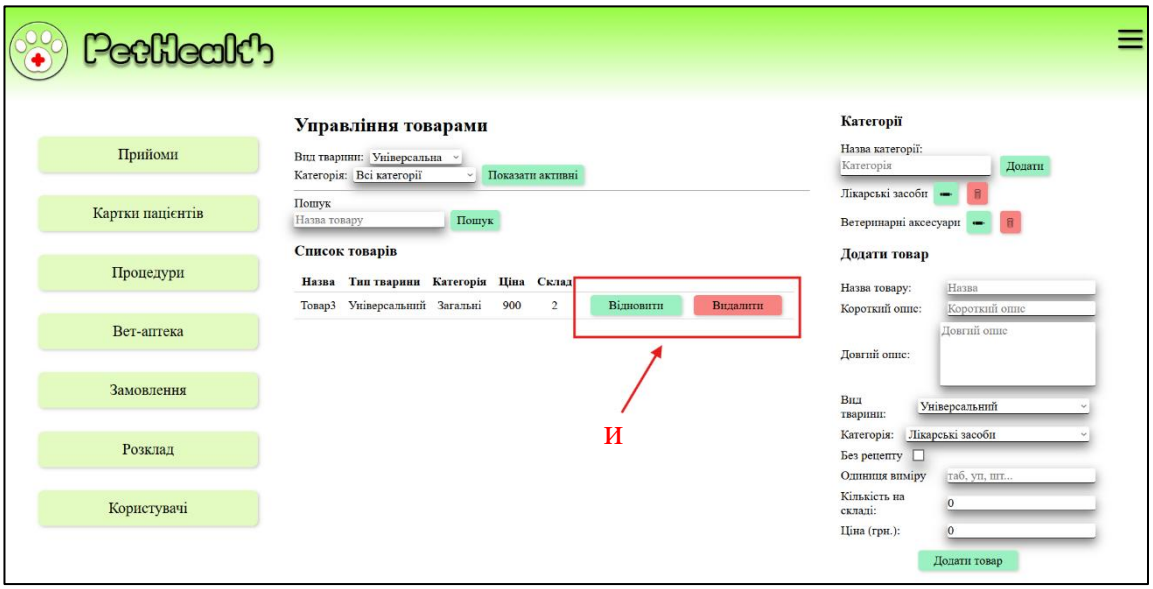


Рисунок 3.43 – Управління товарами у ветаптеці ч.3

12) Для перегляду та редагування замовлень (рис. 3.44-3.45):

а) перейти на вкладку «Замовлення»;

б) фільтрувати замовлення за статусом або шукати за номером;

в) для перегляду детальної інформації натиснути кнопку «Детальніше» біля обраного замовлення;

г) для редагування замовлення натиснути кнопку «Редагувати», у панелі редагування змінити необхідні поля (статус, тип оплати, тип доставки, місто, адреса/відділення/поштомат) та натиснути «Зберегти» або «Скасувати».

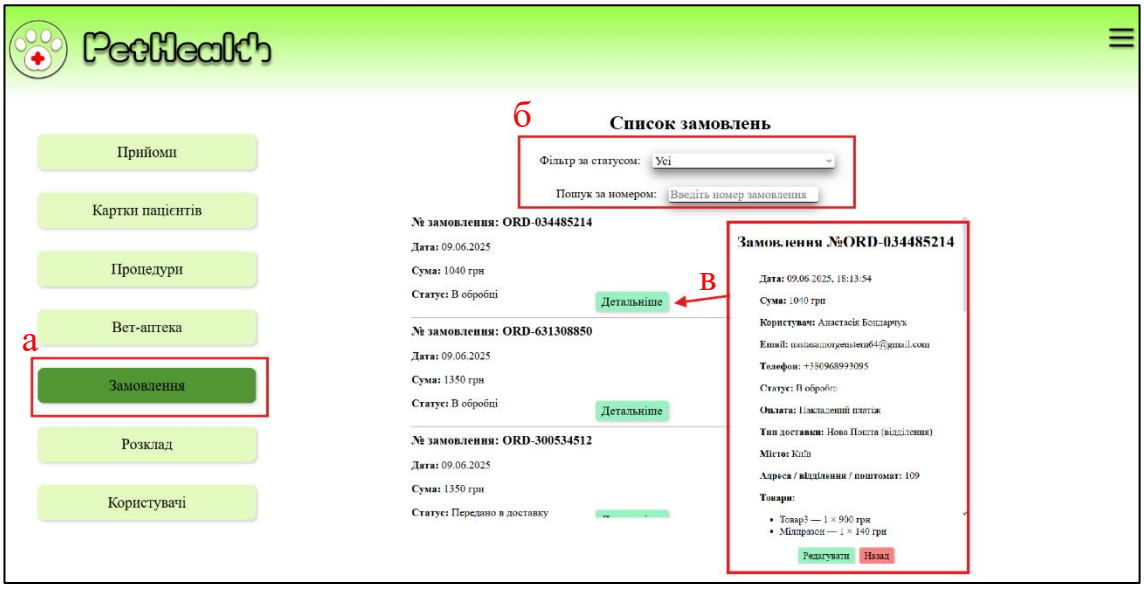


Рисунок 3.44 – Перегляд та редагування замовлень ч.1

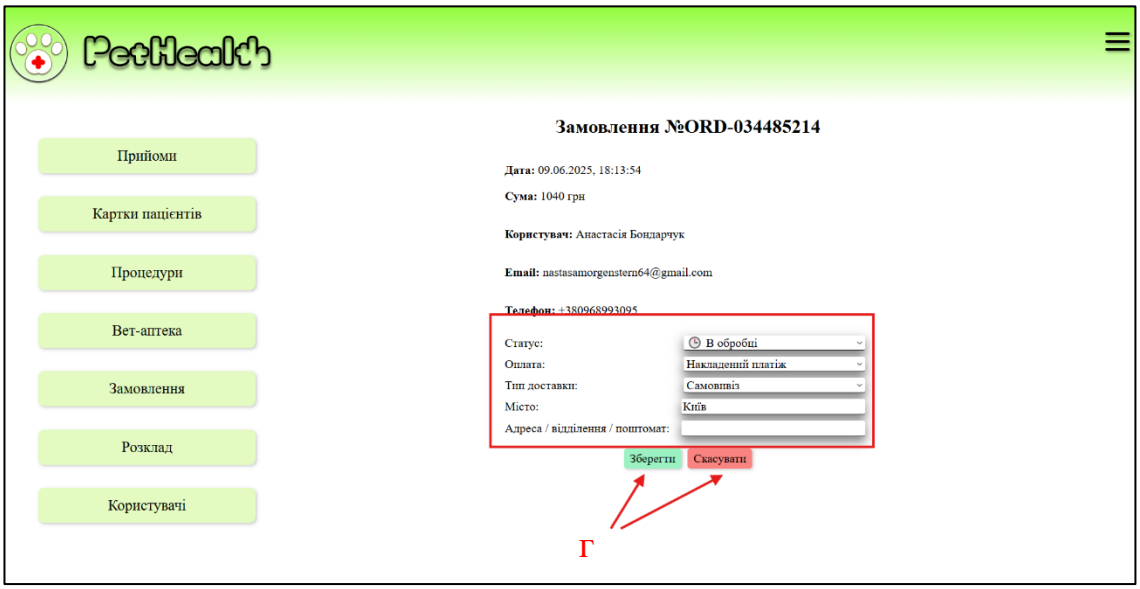


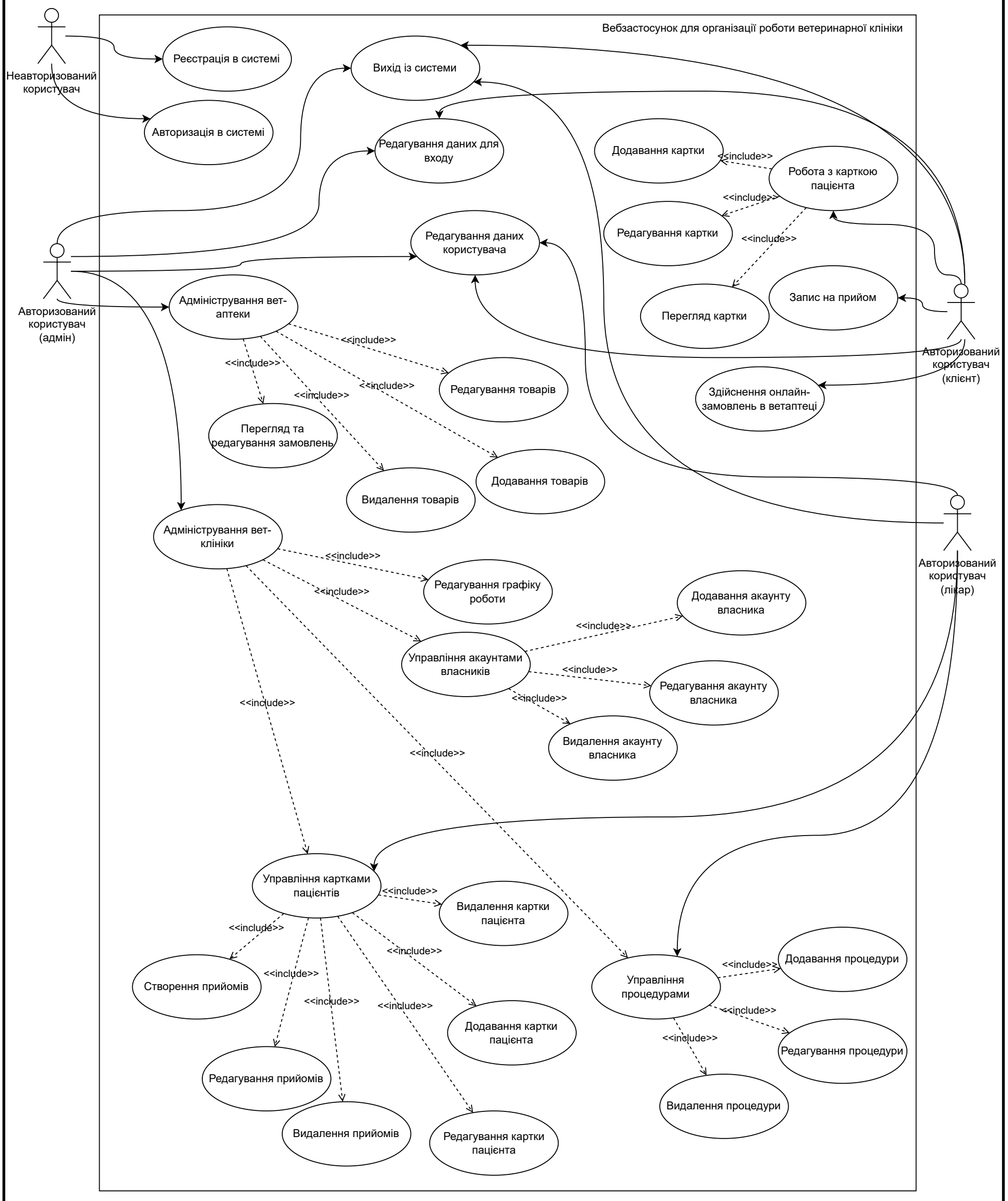
Рисунок 3.45 –Перегляд та редагування замовлень ч.2

3.3 Інструкція для лікаря клініки:

Інструкція для лікаря є аналогічною з інструкцією для адміністратора, проте лікарю доступні лише наступні функції системи:

– Реєстрація/вхід до облікового запису;
– Редагування особистих даних, даних для входу;
– Управління прийомами;
– Управління картками пацієнтів;
– Управління процедурами.

КПІ.ІП-1204.045440.06.99.ССВ



					КПІ.ІП-1204.045440.06.99.ССВ
Зм.	Арк.	№ докум.	Підп.	Дата	
Розробив		Бондарчук А.О.			Схема структурна варіантів використань
Перевірив		Марченко О.І.			
Т. контр.					
Н. контр.		Головченко М.М.			Вебзастосунок для організації роботи ветеринарної клініки
Затвердив		Жаріков Е.В.			

Лит.	Маса	Масштаб
Аркуш 1	Аркушів 1	

КПІ ім.Ігоря Сікорського
Кафедра ІПІ
гр. ІП-12

КПІ.ІП-1204.045440.06.99.СБД

User

PK	Id
	email
	phone
	createdAt
	role
	firstName
	lastName
	birthday
	isArchived

Procedure

PK	Id
	name
	duration
FK	speciesId
	price
	selfBooking
FK	doctorId
	inArchived

Day

PK	Id
	day
	isOpen
	workStart
	workEnd
	hasLunchBreak
	lunchStart
	lunchEnd
	isSurgeryDay
	surgeryStart
	surgeryEnd

embedded array of references

Schedule

PK	Id
	weekSchedule
	weekSchedule
	+ dayId (ref Day)
	updatedAt

Code

PK	Id
	email
	code
	expiresAt

Appointment

PK	Id
	appointmentNumber
FK	petId
FK	userId
FK	procedureId
FK	recipeId
	totalPrice
	date
	startTime
	endTime
	comment
	status
	createdAt
	animalInfo
	diagnosis
	prescription
	usedMedicines
	isArchived

animalInfo
+ weight
+ temperature
+ condition

usedMedicines[]
+ medicineId (ref Product)
+ quantity

PetCard

PK	Id
	name
FK	speciesId
	breed
FK	ownerId
	birthday
	createdAt
	isArchived

Species

PK	Id
	name

Product

PK	Id
	name
	shortDescription
	longDescription
FK	speciesId
	isPrescriptionFree
FK	categoryId
	unit
	stock
	price
	isArchived

Category

PK	Id
	name

Order

PK	Id
FK	userId
	items
	orderNumber
	total
	delivery
	paymentMethod
	status
	createdAt

items[]
+ productId (ref Product)
+ quantity

delivery
+ method
+ city
+ branch
+ postomat
+address

Recipe

PK	Id
	recipeNumber
FK	userId
FK	petId
FK	doctorId
	products
	issuedAt
	validUntil

products[]
+ productId (ref Product)
+ quantity

embedded array of references

embedded array of references

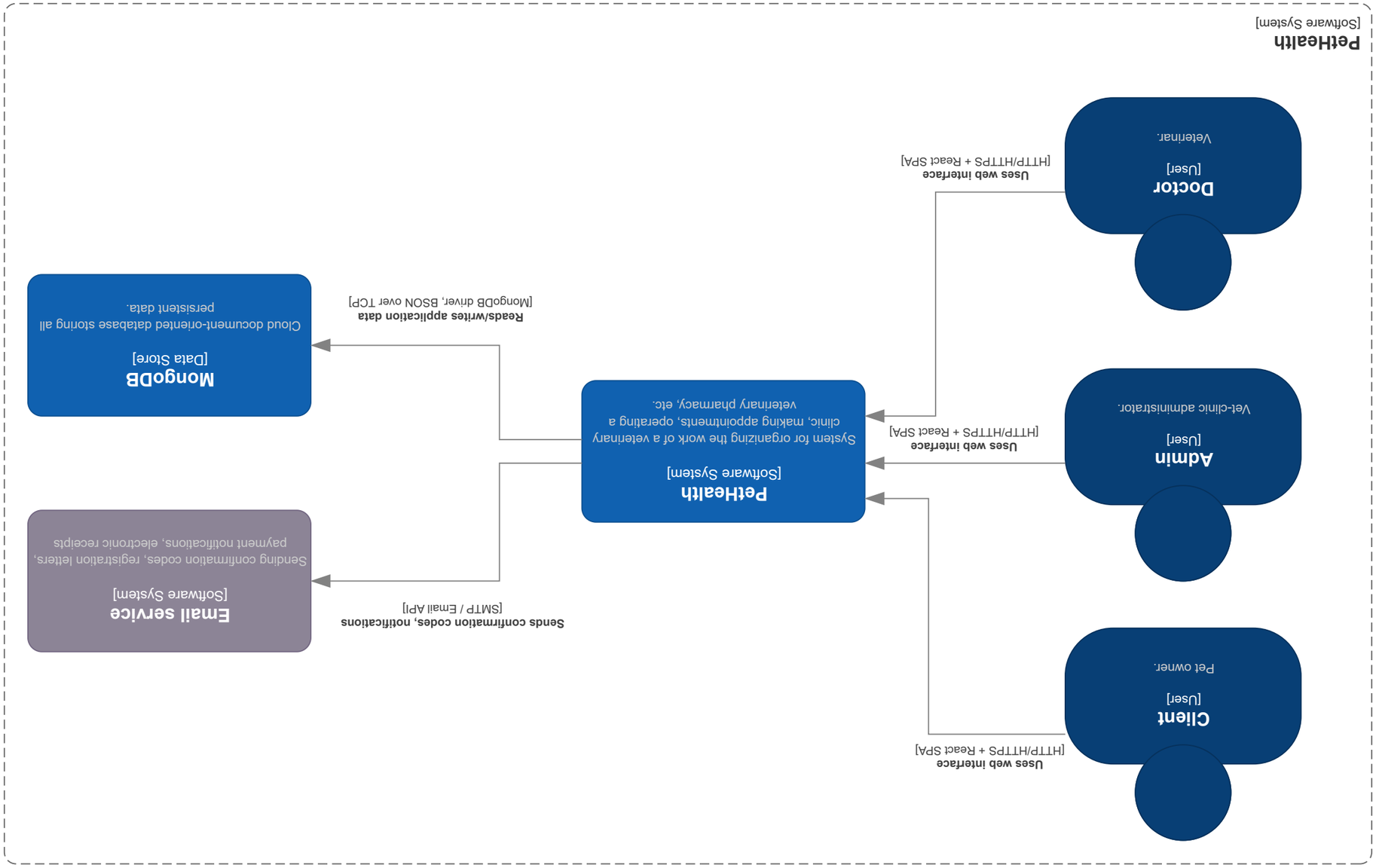
embedded array of references

					КПІ.ІП-1204.045440.06.99.СБД
Зм.	Арк.	№ документа	Підпис	Дата	
Розробив		Бондарчук А.О.			Схема бази даних
Перевірив		Марченко О.І.			
Т. контр.					
Н. контр.		Головченко М.М.			Вебзатосунок для організації роботи ветеринарної клініки
Затвердив		Жаріков Е.В.			

Літера	Маса	Масштаб

Аркуш 1	Аркушів 1

КПІ ім.Ігоря Сікорського
Кафедра ІПІ
гр. ІП-12

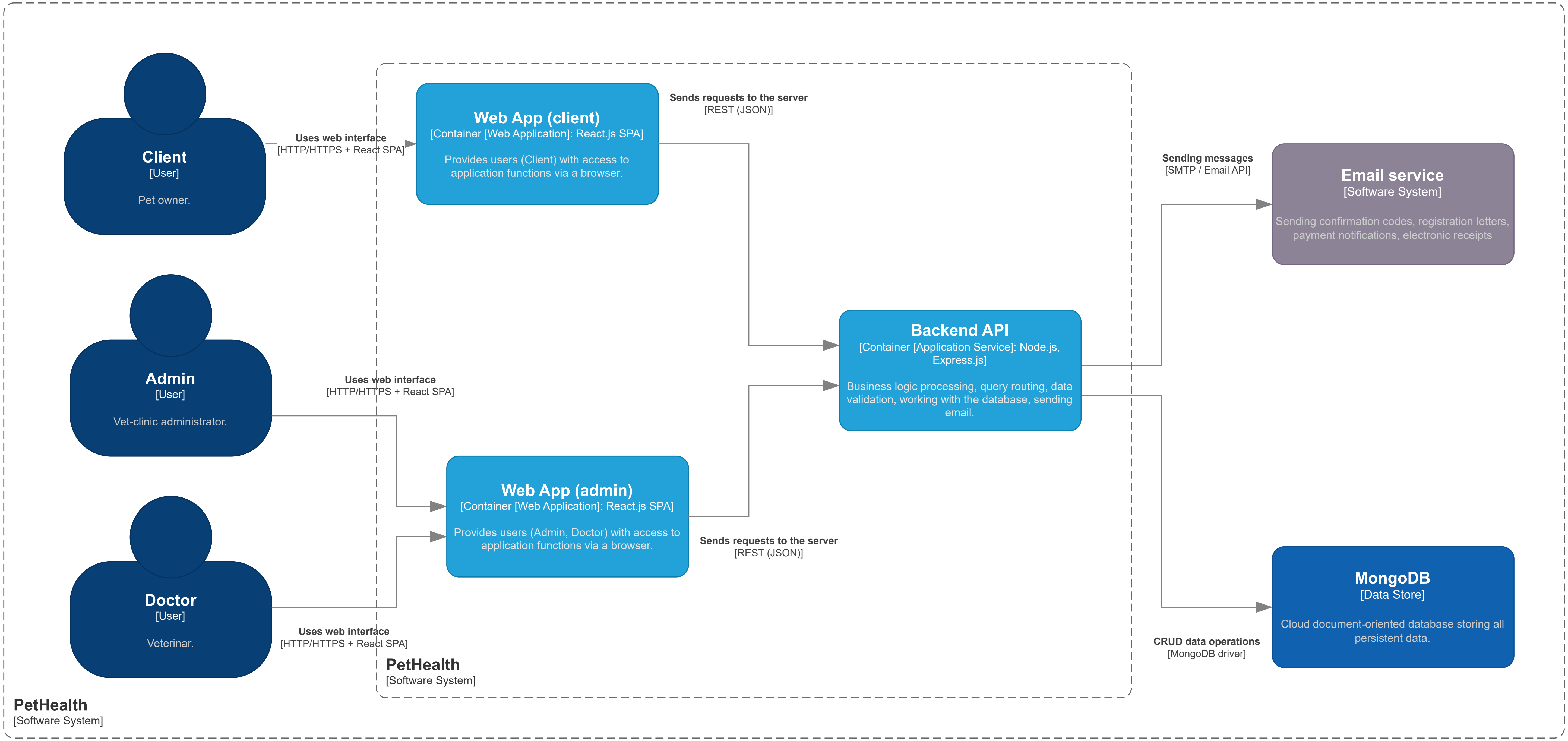


Зм.	Арк.	№ документа	Підпис	Дата	КПІ.ІП-1204.045440.06.99.ССМ
Розробив		Бондарчук А.О.			
Перевірив		Марченко О.І.			Схема структура компонентів програмного забезпечення
Т. контр.					
Н. контр.		Голованенко М.М.			Вебзастосунок для організації роботи ветеринарної клініки
Затвердив		Жарікова Е.В.			

Літера	Маса		Масштаб	
Аркуш 1		Аркушів 2		

КПІ ім.Ігоря Сікорського
Кафедра ІПІ
гр. ІП-12

КПІ.ІП-1204.045440.06.99.ССМ



Зм.	Арк.	№ документа	Підпис	Дата	КПІ.ІП-1204.045440.06.99.ССМ		
Розробив		Бондарчук А.О.			Схема структурна компонентів програмного забезпечення	Літера	Маса	Масштаб
Перевірив		Марченко О.І.				Аркуш 2	Аркушів 2
Т. контр.					Вебзастосунок для організації роботи ветеринарної клініки	КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІП-12	
Н. контр.		Головченко М.М.					
Затвердив		Жаріков Е.В.					