НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«До захисту допущено»
Завідувач кафедри
_____ Євгенія СУЛЕМА
«___» _____ 2023 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інженерія програмного забезпечення мультимедійних та інформаційно-пошукових систем»

спеціальності 121 Інженерія програмного забезпечення

на тему: «Вебзастосунок для управління наданням ветеринарних послуг»

Виконала:
студентка IV курсу, групи КП-93
Григор Олена Миколаївна _____

Керівник:
Доцент кафедри ПЗКС, к.т.н., доцент,
Люшенко Леся Анатоліївна _____

Консультант з нормоконтролю:
Доцент кафедри ПЗКС, к.т.н., доцент,
Онай Микола Володимирович _____

Рецензент:
Доцент кафедри СПСКС, к.т.н., доцент,
Тарасенко-Клятченко Оксана Володимирівна _____

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

Студентка _____

Київ – 2023 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет прикладної математики
Кафедра програмного забезпечення комп'ютерних систем

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення мультимедійних та інформаційно-пошукових систем»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Євгенія СУЛЕМА
«___» _____ 2022 р.

ЗАВДАННЯ

на дипломний проєкт студенту

Григор Олені Миколаївні

1. Тема проєкту: «Вебзастосунок для управління наданням ветеринарних послуг», керівник проекту Люшенко Леся Анатоліївна, доцент кафедри ПЗКС, к.т.н. доцент, затверджені наказом по університету від

«31» травня 2023 р. № 2107-с

2. Термін подання студентом проєкту: «16» червня 2023 р.

3. Вихідні дані до проєкту: див. Технічне завдання.

4. Зміст пояснювальної записки:
   - аналіз існуючих рішень поставленої задачі;
   - обґрунтування вибору засобів реалізації;
   - реалізація вебзастосунку "HappyPet";
   - аналіз розробленого вебзастосунку.

5. Перелік обов'язкового графічного матеріалу:
   - Структура бази даних (креслення).
   - Алгоритм роботи вебзастосунку (креслення).
   - Структура вебзастосунку (плакат).
   - Діаграма використання вебзастосунку (плакат).

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Онай М.В., доцент		

7. Дата видачі завдання «31» жовтня 2022 р.

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	Вивчення літератури за тематикою проєкту	13.11.2022	
2.	Розроблення та узгодження технічного завдання	25.11.2022	
3.	Розроблення структури вебзастосунку	15.12.2022	
4.	Підготовка матеріалів першого розділу дипломного проєкту	30.12.2022	
5.	Розроблення дизайну сторінок та графічних елементів	03.02.2023	
6.	Підготовка матеріалів другого розділу дипломного проєкту	19.02.2023	
7.	Програмна реалізація вебзастосунку	10.03.2023	
8.	Тестування вебзастосунку	17.03.2023	
9.	Підготовка матеріалів третього розділу дипломного проєкту	30.03.2023	
10.	Підготовка матеріалів четвертого розділу дипломного проєкту	12.04.2023	
11.	Підготовка графічної частини дипломного проєкту	21.04.2023	
12.	Оформлення документації дипломного проєкту	26.05.2023	

Студентка Олена ГРИГОР

Керівник проєкту Леся ЛЮШЕНКО

АНОТАЦІЯ

Даний дипломний проєкт присвячено розробці програмного забезпечення для управління наданням ветеринарних послуг.

Розроблене програмне забезпечення являє собою вебзастосунок, який дозволяє власникам тварин записуватись онлайн на прийом до ветеринара, дозволяє зберігати медичну інформацію тварини, доступом до якої керує лише власник тварини, дозволяє переглядати на скасовувати заплановані прийоми.

Метою даного дипломного проєкту є створення програмного продукту, який спростив би комунікацію між власником тварини та ветеринаром, а також допоміг би автоматизувати процес запису на прийом до ветеринара, що в свою чергу, зменшить кількість адміністративної роботи для ветеринара та покращить процес запису на прийом для власника тварини.

У даному вебзастосунку розроблено архітектуру проєктування MVC (Model-View-Controller), який дозволяє розділити систему на три компоненти: Модель – відповідає за управління даними та бізнес-логікою, Представлення – відповідає за відображення даних на вебсторінках та Контролер – відповідає за керування потоком даних та взаємодіє з користувачем, було розроблено структуру бази даних, а також створені графічні елементи та дизайн інтерфейсів клієнтської частини.

ABSTRACT

This diploma project is devoted to the development of software for managing the provision of veterinary services.

The software developed is a web-based application that allows pet owners to book vet appointments online, allows for storage of pet medical information that only the pet owner can control, and allows for viewing and canceling scheduled appointments.

The purpose of this thesis is to create a software product that would simplify communication between the pet owner and the veterinarian, as well as help automate the process of making an appointment with a veterinarian, which in turn will reduce the amount of administrative work for the veterinarian. veterinarian and improve the process of registering animal owners

In this web application, the MVC (Model-View-Controller) design architecture is developed, which allows you to divide the system into three components: Model – responsible for managing data and business logic, Presentation – responsible for displaying data on web pages, Controller – responsible manages data flow and interacts with the user, developed the database structure, and also created graphic elements and design of client-side interfaces.

ДП.045440-01-90 Вебзастосунок для управління наданням ветеринарних послуг. Відомість проєкту

Позначення	Найменування	Кіл-ть	Примітка
	Документація проєкту		
ДП.045440-02-91	Вебзастосунок для	5	
	управління наданням		
	ветеринарних послуг.		
	Технічне завдання		
ДП.045440-03-81	Вебзастосунок для	81	
	управління наданням		
	ветеринарних послуг.		
	Пояснювальна записка		
ДП.045440-04-51	Вебзастосунок для	4	
	управління наданням		
	ветеринарних послуг.		
	Програма та методика		
	тестування		
ДП.045440-05-34	Вебзастосунок для	13	
	управління наданням		
	ветеринарних послуг.		
	Керівництво користувача		
ДП.045440-06-99	Вебзастосунок для	1	
	управління наданням		
	ветеринарних послуг.		
	Алгоритм роботи		
	вебзастосунку. Діаграма		
	діяльності		

Позначення	Найменування	Кіл-ть	Примітка
	Документація проєкту		
ДП.045440-07-99	Вебзастосунок для	1	
	управління наданням		
	ветеринарних послуг.		
	Структура бази даних.		
	ERD-діаграма		
ДП.045440-08-98	Вебзастосунок для	1	
	управління наданням		
	ветеринарних послуг.		
	Компакт-диск		

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

"ЗАТВЕРДЖЕНО"

Завідувач кафедри

_____ Євгенія СУЛЕМА

"___" _____ 2022 р.

ВЕБЗАСТОСУНОК ДЛЯ УПРАВЛІННЯ НАДАННЯМ ВЕТЕРИНАРНИХ ПОСЛУГ

Технічне завдання

ДП.045440-02-91

"ПОГОДЖЕНО"

Керівник проєкту:

_____ Леся ЛЮШЕНКО

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Олена ГРИГОР

2022

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: Вебзастосунок для управління наданням ветеринарних послуг.

Галузь застосування: інформаційні технології.

2. ПІДСТАВА ДЛЯ РОЗРОБЛЕННЯ

Підставою для розроблення є завдання на дипломне проектування, затверджене кафедрою програмного забезпечення комп'ютерних систем Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського» (КПІ ім. Ігоря Сікорського).

3. ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для використання особами в якості інформаційного забезпечення в сфері ветеринарних послуг з метою надання користувачеві можливості записуватись на прийом до ветеринара, а також зберігати та керувати медичною інформацією про тварину, а для ветеринарів надається можливість керувати власними записами на прийом.

4. ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ

Вебзастосунок повинен забезпечувати такі основні функції:

- можливість переглядати список ветеринарів, а також здійснювати пошук за фільтрами;
- можливість переглядати інформацію про ветеринара, доступні часові слоти для запису, а також читати відгуки про нього;
- можливість запису на прийом до ветеринара;

- для власника тварини є можливість зберігати медичну інформацію про тварину та керувати доступом до неї;
- можливість переглядати та скасовувати заплановані прийоми;
- після завершення прийому, є можливість написати відгук про ветеринара.

Розробку виконати на платформі .NET (мова програмування C#) з використанням фреймворку ASP.NET Core MVC.

Додаткові вимоги:

- наявність логотипу та назви програмного додатку;
- адаптивний користувацький інтерфейс;
- дизайн сторінок з використанням в якості базових білого, зеленого та червоного кольорів.

5. ВИМОГИ ДО ПРОЄКТНОЇ ДОКУМЕНТАЦІЇ

У процесі виконання проекту повинна бути розроблена наступна документація:

1) пояснювальна записка;
2) програма та методика тестування;
3) керівництво користувача;
4) креслення:
    - «Алгоритм роботи вебзастосунку. Діаграма діяльності».
    - «Структура бази даних. ERD-діаграма».

6. ЕТАПИ ПРОЄКТУВАННЯ

Вивчення літератури за тематикою роботи.....16.11.2022

Розроблення та узгодження технічного завдання.....27.11.2022

Розроблення структури вебзастосунку.....15.12.2022

Розроблення дизайну сторінок та графічних елементів.....03.02.2023

Програмна реалізація вебзастосунку.....15.03.2023

Тестування вебзастосунку.....07.04.2023

Підготовка матеріалів текстової частини проєкту.....28.04.2023

Підготовка матеріалів графічної частини проєкту.....12.05.2023

Оформлення технічної документації проєкту.....25.05.2023

7. ПОРЯДОК ТЕСТУВАННЯ РОЗРОБКИ

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

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

"ЗАТВЕРДЖЕНО"

Завідувач кафедри

_____ Євгенія СУЛЕМА

"___" _____ 2023 р.

ВЕБЗАСТОСУНОК ДЛЯ УПРАВЛІННЯ НАДАННЯМ ВЕТЕРИНАРНИХ ПОСЛУГ

Пояснювальна записка

ДП.045440-03-81

"ПОГОДЖЕНО"

Керівник проєкту:

_____ Леся ЛЮШЕНКО

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Олена ГРИГОР

2023

ЗМІСТ

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

Ветеринарна клініка – це медичний заклад, який надає ветеринарні послуги, включаючи профілактику, діагностику та лікування захворювань тварин.

Власник тварини – фізична або юридична особа, яка здійснює догляд за твариною, що належить їй на праві власності, і несе відповідальність згідно із законом за стан та дії тварини.

DOM (document object model) – це об’єктна модель документа в браузері для відтворення розмітки. Це деревоподібне представлення сторінки, починаючи з тегу <«html»>, яке зберігається в пам’яті браузера й показує те, що користувач бачить на сторінці. Якщо реальний DOM змінюється, браузер перемальовує й перекомпоновує частини сторінки або її повністю. Це ресурсовитратна операція.

ПЗ – програмне забезпечення.

Common Language Runtime (CLR) – це віртуальна машина, яка виконує програми, написані для платформи .NET Framework. CLR забезпечує середовище виконання для програм, що розроблені на різних мовах програмування, таких як C#, VB.NET, F#, і т.д.

Garbage Collector (GC) – це компонент, який входить до складу Common Language Runtime (CLR) в платформі .NET. GC виконує автоматичне управління пам'яттю в програмах, написаних для .NET Framework.

СКБД (Система Керування Базами Даних або англ. Relational Database Management System або RDBMS) – набір взаємопов'язаних даних (база даних) і програм для доступу до цих даних. Надає можливості створення, збереження, оновлення та пошуку інформації в базах даних з контролем доступу до даних.

SQL (Structured Query Language) – декларативна мова програмування для взаємодії користувача з базами даних, що застосовується для формування запитів, оновлення і керування реляційними БД, створення схеми бази даних та її модифікації, системи контролю за доступом до бази даних.

Авторизація – надання певній особі або групі осіб прав на виконання певних дій; а також процес перевірки (підтвердження) даних прав при спробі виконання цих дій.

Аутентифікація – процедура встановлення належності користувачеві інформації в системі через перевірку пред'явленого ним ідентифікатора.

JS – JavaScript.

БД – база даних.

UI – інтерфейс користувача.

ОС – операційна система.

Модель–вигляд–контролер (Model-view-controller, MVC) – архітектурний шаблон, який використовується під час проєктування та розроблення програмного забезпечення.

Модульне тестування (Unit testing) – це метод тестування програмного забезпечення, який полягає в осібному тестуванні кожного модуля коду програми.

ВСТУП

У сучасному світі інформаційні технології стали невід'ємною частиною повсякденного життя кожної особи. Згідно зі соціологічними дослідженнями, проведеними компанією "Washington Post Group", Україна займає місце серед десяти країн з найбільшою кількістю домашніх тварин. За дослідженнями компанії "Worldatlas", Україна займає друге місце у світі за кількістю котів на одну особу, зі статистикою 17 котів на 100 осіб, при цьому 8 з кожних 100 осіб мають собаку. Тому збільшуються попит на медичні послуги, косметичні засоби, одяг, харчування, ліки для домашніх тварин[1].

На даний момент для запису у ветеринарну клініку необхідно здійснити пошук, знайти контактні дані, погодити час прийому, і в більшості випадків це відбувається за телефоном, що не досить зручно у сучасному автоматизованому світі. До того ж, компаніям, що надають послуги необхідно постійно слідкувати за графіком. Інколи, при зміні ветеринара може частково втратитись важлива інформація, така як дані про щеплення, історія хвороб тварини.

Метою дипломного проєкту є створення вебзастосунку, який оптимізує комунікацію між власником тварини та ветеринаром: спрощує процес пошуку та запису до ветеринара; дозволяє переглядати на скасовувати заплановані прийоми, а також оцінювати якість наданих ветеринарних послуг.

1. АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ПОСТАВЛЕНОЇ ЗАДАЧІ

1.1. Загальні положення та аналіз предметної області

Наявність домашніх тварини є зростаючим трендом сьогодення. Згідно з результатами соціологічного опитування, проведеного компанією "Research & Branding Group", близько 57% українців мають домашніх тварин [2]. А за даними видання “Worldatlas.com”, Україна займає дев'яте місце в рейтингу країн, жителі яких утримують найбільше котів [1]. Для більшості власників тварин їхні домашні улюбленці як члени сім'ї, тому їхнє здоров'я та добробут є одним із головних пріоритетів.

Зі зростанням популярності домашніх тварин, зростає і кількість проблем, з якими стикаються їхні власники. Однією з них є потреба в наданні якісної ветереренарної допомоги, оскільки власники тварин прагнуть надати своїм улюбленцям максимальний догляд та комфорт.

Відвідування ветеринарних клінік часто буває трудомістким завданням, оскільки у процесі запису до ветеринара власники тварин стикаються з низкою проблем:

- складний пошук ветеринара, через наявність великої кількості різних вебзастосунків;
- необхідність перевіряти власноруч актуальність інформації про ветеринара, оскільки на час війни багато ветеринарів припинили роботу;
- необхідність телефонувати та узгоджувати час прийому;
- відсутність вебзастосунку з можливістю запису на прийом до ветеринара онлайн.

У той же час, ветеринар також має певні незручності:

- необхідно відповідати на телефонні дзвінки, повідомлення в месенджерах від власників домашніх тварин, записувати їх дані

вручну та розміщувати у зручний для себе час, що може призвести до затримок у записі та неефективного використання часу;

- при записі інформації вручну, існує ризик запису не правильної інформації;
- якщо власник домашньої тварини не прийде на запланований прийом, ветеринар втрачає час;
- може бути обмежений в доступі до повної інформації про стан тварини, що може ускладнити процес діагностики та лікування.

У зв'язку з цим, виникає потреба у наявності вебзастосунку, який допоміг би налогодити комунікацію в мережі інтернет між власниками домашніх тварин та ветеринарами. А також допоміг би ветеринарам зменшити кількість адміністративної роботи.

1.2. Аналіз аналогічних додатків

Провівши аналіз різних джерел інформації, а також існуючих програмних рішень, які пов’язані з ветеринарними клініками, були зібрані дані про переваги та недоліки окремо взятих продуктів, які будуть використовуватись на етапі збору вимог до створюваного вебзастосунку.

Нижче наведені декілька основних ПЗ.

1.2.1. *“PetAdvisor”*

“PetAdvisor” – це вебзастосунок для пошуку ветеринарних клінік, грумінг-салонів та закладів для перетримки домашніх тварин в найбільших містах України, який також містить рейтинги на основі відгуків клієнтів для клінік, грумінг-салонів, закладів для перетримки тварин. Головна сторінка вебзастосунку зображена на рис. 1.1.

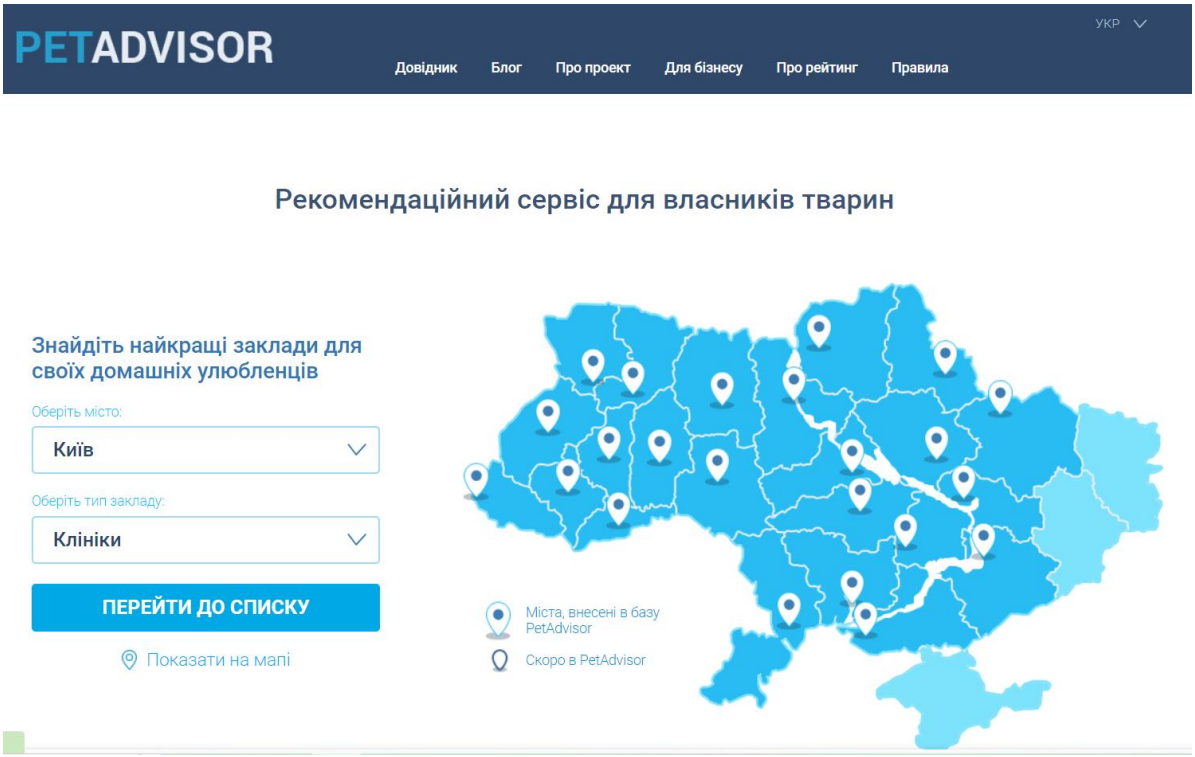


Рис. 1.1. Вигляд інтерфейсу вебзастосунку “PetAdvisor”

Переваги вебзастосунку:

- рейтинг ветеринарних клінік з урахуванням позитивних відгуків та скарг;
- пошук різних типів закладів: ветеринарні клініки, грумінг-салони, заклади для перетримки домашніх тварин;
- користувачі за допомогою декількох кліків можуть знайти на карті, або в рейтингу ветеринарних клінік, контактну інформацію про ветеринарну клініку;
- доступний для використання в будь-який час та в будь-якому місці при доступі до Інтернету;
- наявна велика кількість ветеринарних клінік;
- перевірена на актуальність інформація стосовно номерів телефонів, адреси та години роботи ветеринарних клінік;
- наявність блогу та довідника частих хвороб домашніх тварин.

Недоліки вебзастосунку:

- оцінки та відгуки, залишені користувачами вебзастосунку, можуть бути необ'єктивними або спотвореними, що може призвести до неправильного вибору ветеринарної клініки власником тварини;
- надає лише контактну інформацію ветеринарної клініки та не містить даних про наявність вільних часових слотів для запису на прийом;
- немає можливості зробити онлайн-запис на прийом до ветеринара, тому власникам тварин потрібно зв’язуватись з клінікою самостійно, щоб дізнатися про можливість запису та наявність вільних часових слотів;
- не містить інформації про стан здоров’я, історію візитів до ветеринара конкретної тварини;
- особисті дані власника та тварини зберігаються на серверах компанії, тому компанія має повний доступ до неї, що може бути проблемою для тих, хто пильно стежить за своєю конфіденційністю та безпекою використанні даних в Інтернеті;
- наявність великої кількості реклами, що відволікає власника тварини від перегляду основного контенту сайту, а також збільшує час завантаження сторінок вебзастосунку.

1.2.2. *“PetLive”*

“PetLive” – це вебзастосунок, створений з метою об'єднання інтересів власників тварин та надавачів різних послуг, пов'язаних з доглядом за домашніми улюбленцями. Вебзастосунок надає широкий вибір послуг, включаючи перетримку та вигул тварин, грумінг, консультації ветеринарів та інші послуги, які стосуються догляду за тваринами. Крім того, “PetLive” включає в себе різноманітні бізнеси, пов'язані з доглядом за тваринами, такі як ветеринарні клініки, зооаптеки, зоомагазини, ресторани і готелі, в яких

можна перебувати з тваринами, та інші. Сторінка з ветеринарними клініками зображена на рис. 1.2.

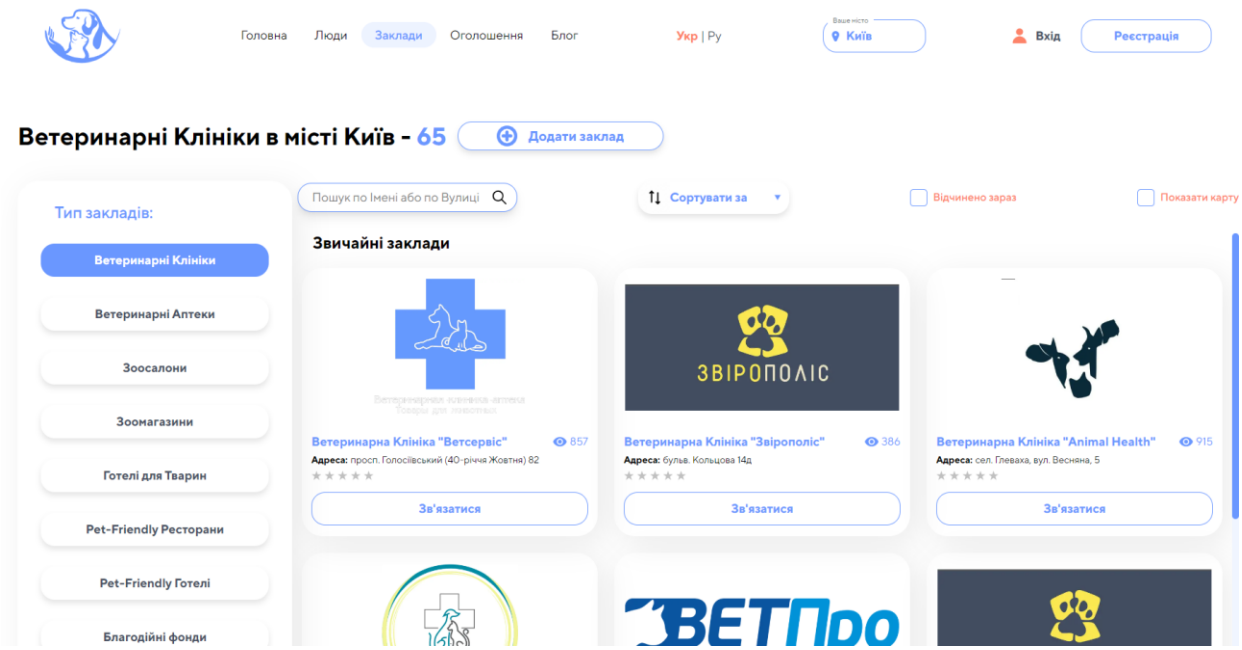


Рис. 1.2. Вигляд інтерфейсу вебзастосунку “PetLive”

Переваги вебзастосунку:

- представлені різні типи закладів пов’язаних з доглядом за тваринами – ветеринарні клініки, ветеринарні аптеки, зоосалони, готелі для тварин, зоомагазини, ресторани та готелі, в яких на перебувати з тваринами та інші;
- надавачі послуг можуть зареєструватись на сайті та запропонувати свої послуги власникам тварин;
- має інтуїтивно зрозумілий та простий інтерфейс, який дозволяє власникам тварин швидко та легко знайти послуги, які вони потребують для своїх домашніх улюбленців;
- наявність особистого кабінету, де можна редагувати особисту інформацію, керувати власними повідомленнями та заявками на бронювання послуг;

- можливість сортувати заклади по популярності, по рейтингу на основі відгуків та по даті додавання закладу на сайт.

Недоліки вебзастосунку:

- відсутність онлайн-запису на прийом до ветеринара з можливістю забронювати зручний для власника тварини час візиту;
- користувач вебзастосунку може додати не існуючу ветеринарну клініку у загальний список клінік представлених на сайті, що може ускладнити пошук власникам тварин при виборі клініки;
- представлена невелика кількість клінік та ветеринарів, що може бути недостатньою для деяких власників тварин;
- немає можливості збереження медичних даних тварин;
- відсутність перегляду історії відвідування ветеринара.

1.2.3. *“Vetconsult.gladpet.org”*

“Vetconsult.gladpet.org” – це безкоштовний вебзастосунок, де можна отримати онлайн-консультацію для домашніх тварин від міжнародних ветеринарів, створений при сприянні “Veterinarians Without Borders”. Модерують та адмініструють вебзастосунок партнери “GladPet” та ветеринарна клініка “Пес і Кіт” [3].

Вебзастосунок надає можливість власникам тварин поставити питання стосовно здоров’я улюбленця ветеринарам із Канади, США, Великої Британії й інших країн. Наприклад, як знайти альтернативи звичайному корму в умовах війни, що робити в разі відсутності потрібних ліків, як знизити стрес тварини, урятувати її в екстремальних умовах чи як доглядати за екзотичними тваринами.

Головна сторінка вебзастосунку зображена на рис. 1.3.

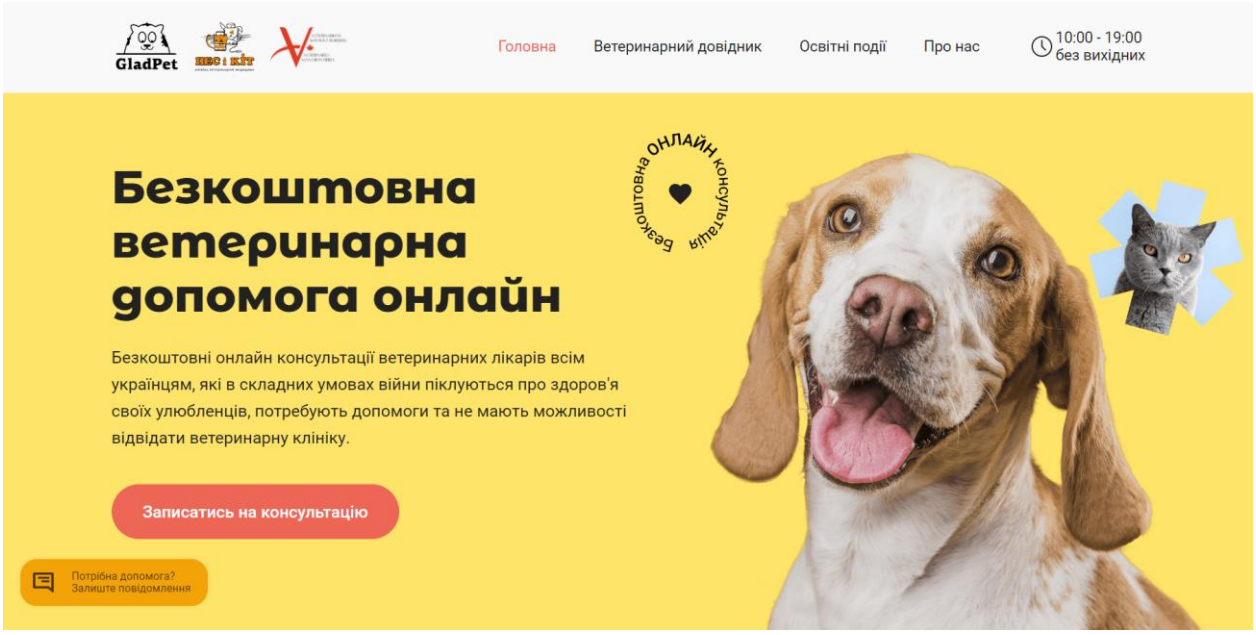


Рис. 1.3. Вигляд інтерфейсу вебзастосунку “ Vetconsult.gladpet ”

Переваги вебзастосунку:

- власники домашніх тварин можуть отримати консультацію від ветеринарів у режимі онлайн;
- можна обирати час онлайн-консультації;
- власники домашніх тварин можуть скористатися вебзастосунком в будь-який час, що забезпечує максимальну зручність;
- на вебзастосунку можна знайти інформацію про освітні заходи, такі як семінари, тренінги, конференції, які проводяться для ветеринарів, власників тварин та всіх, хто цікавиться тваринами;
- вебзастосунок має ветеринарний довідник, де розміщено корисну інформацію про догляд за тваринами, їх харчування, лікування та профілактику захворювань.

Недоліки вебзастосунку:

- не має можливості створити обліковий запис та зберігати медичну інформацію про тварину;

- відсутня можливість перегляду доступних ветеринарів та вибір ветеринара залежно від потреб власника тварини;
- відсутність рейтингу ветеринарів на основі відгуків користувачів вебзастосунку, що може бути проблемою для тих, хто хоче дізнатись думку інших перед зверненням до конкретного ветеринара;
- відсутність блогу або форуму для обговорення проблем власників тварин.

1.3. Актуальність розробки вебзастосунку “HappyPet”

Проаналізувавши наведені вебзастосунки за такими критеріями як зручність пошуку ветеринара, можливість онлайн запису на прийом, можливість скасовувати прийом, інтуїтивно зрозумілий інтерфейс – можна зробити висновок, що існуючі вебзастосунки, не задовольняють потребу власників тварин у наявності вебзастосунку, який поєднував би ці функції. Результати порівняння досліджуваних вебзастосунків за вказаними критеріями наведено у табл. 1.1.

Таблиця 1.1

Порівняння функціональності наведених вище вебзастосунків та запропонованого вебзастосунку “HappyPet”

Показник	PetAdvisor	PetLive	Vetconsult.gladpet.org
Зручність пошуку ветеринара	+	+	–
Можливість зареєструватись на сайті	–	+	–
Зручний та інтуїтивно зрозумілий інтерфейс	+	+	–

Продовження табл. 1.1

Наявність відгуків до ветеринарів	+	+	–
Керування прийомами	–	–	–
Сповіщення про прийом	–	–	–
Онлайн запис на прийом до ветеринара	–	–	+

Отже, відсутність вебзастосунку, який надає можливість записуватись онлайн на прийом до ветеринара, обирати ветеринара за відгуками з великої вибірки ветеринарів та мати можливість скасовувати прийом є актуальною проблемою. Вебзастосунок “HappyPet” може стати рішенням, яке допоможе власникам тварин швидше та простіше записуватись до ветеринарного лікаря з можливістю керувати прийомами.

1.4. Вимоги до вебзастосунку “HappyPet”

На підставі зазначених вище проблем було сформульовано список загальних вимог до вебзастосунку, розробленого в рамках даного дипломного проєкту:

- можливість створення облікового запису з вказанням особистих даних власника тварини;
- можливість записуватися онлайн на прийом до ветеринара, вибираючи зручний час;
- керування прийомами з боку власника тварини та ветеринара;
- сповіщення власнику тварини про запланований прийом;
- підтримка шифрування даних для забезпечення інформаційної безпеки та конфіденційності даних;

- надання різних рівнів доступу та прав користувачам – власник тварини та ветеринар;
- наявність фільтрів для пошуку та сортування ветеринарів;
- інтуїтивно зрозумілий та зручний у користуванні інтерфейс.

1.4.1. *Вимоги до безпеки додатка*

Перед розробкою програмного забезпечення важливо знизити його вразливість до потенційних загроз. Один з основних методів для визначення потенційних загроз і ризиків в інформаційній безпеці є метод STRIDE. Цей метод був розроблений у 1999 році в рамках проєкту “Microsoft Research” і відтоді став широко використовуваним у галузі інформаційної безпеки [4].

Метод STRIDE базується на визначенні шести основних загроз для інформаційної системи:

- Spoofing (фальсифікування) – це загроза, при якій зловмисник може підробити ідентифікацію користувача або системи, щоб отримати незаконний доступ до системи або даних.
- Tampering (підробка даних) – це загроза, коли зловмисник може змінювати, вносити або вилучати дані в системі, щоб отримати несанкціонований доступ до інформації або вплинути на її роботу.
- Repudiation (відмова від відповідальності) – це загроза, коли зловмисник може заперечувати факт виконання певної дії в системі, що ускладнює виявлення та реагування на атаки.
- Information Disclosure (розголошення інформації) – це загроза, при якій зловмисник може проникнути в систему та незаконно отримати доступ до конфіденційної інформації з метою її недозволеного використання у своїх особистих цілях.

- Denial of Service (відмова в обслуговуванні) – це загроза, коли зловмисник може переповнити систему запитами, що призводить до зниження її продуктивності або навіть повного зупину.
- Elevation of Privilege (підвищення привілеїв) – це загроза, коли зловмисник може отримати більше привілеїв або доступу до системи або даних, ніж має.

З використанням цього методу були визначені можливі загрози фальсифікування, підробки даних, відмови, розголошення інформації, відмови в обслуговуванні, підвищення привілеїв, які наведені у табл. 1.2.

До відповідних загроз також встановлені вимоги безпеки.

Таблиця 1.2

Загрози обраного програмного забезпечення визначені методом STRIDE

Загроза	Властивість, що порушується	Визначення загрози	Вимога безпеки
Spoofing	Автентифікованість	Видавання за серверну частину (спуфінг DNS)	Розмістити серверну частину на захищеній сторонній хмарі, зберігати значення доменного імені на клієнті у прихованих файлах і не використовувати явно у коді
Spoofing	Автентифікованість	Зміна коду серверної частини	
Spoofing	Автентифікованість	Видавання за адміністратора бази даних	Забезпечити мультифакторову автентифікацію у СУБД, регулярне оновлювати пароль адміністратора

Продовження табл. 1.2

Tampering	Цілісність	Зміна даних у базі даних	Вести журнал активності користувачів, запитувати підтвердження дій користувача перед внесенням змін
Tampering	Цілісність	Втручання в мережу	Забезпечити з’єднання за протоколом HTTPS
Repudiation	Безвідмовність	Внесення змін до системних даних	Забезпечити окремі рівні доступу до бази даних для різних ролей
Information Disclosure	Конфіденційність	Отримання повідомлень помилки серверного та клієнтського коду	Обробляти помилки у коді як на клієнтській, так і на серверній частині, та виводити користувачу оброблену інформацію про помилку
Information Disclosure	Конфіденційність	Перехоплення запитів через мережу	Забезпечити з’єднання за протоколом HTTPS, шифрувати чутливі дані
Denial of Service	Доступність	Унеможливлення доступу до серверної частини застосунку	Використовувати сервіси захисту серверної частини від DoS та DDoS атак (наприклад, AWS Shield)
Elevation of Privilege	Авторизованість	Діяння користувача понад встановлених прав	Забезпечити окремі рівні доступу до бази даних з різними обліковими даними

Виявлені вимоги до безпеки допомогли визначити потенційно слабкі місця при реалізації спроєктованої архітектури та прийняти відповідне рішення по усуненню їх.

1.5. Висновки до розділу

Більше половини українців тримають домашніх тварин, яких часто сприймають як членів своєї сім'ї і піклуються про їхнє здоров'я та добробут. Тому при виборі ветеринара обирають кращого за своїми потребами. Однак, при пошуку ветеренарних послуг часто виникають проблеми – наявність великої кількості ветеринарів, розміщених у мережі Інтернет, необхідність перевіряти актуальність знайденої про ветеринара інформації, відсутність запису на прийом онлайн та інші. Тому метою даної роботи є створення вебзастосунку “HappyPet”, який вирішуватиме згадані вище проблеми.

Проаналізовані схожі за функціональністю вебзастосунки – “PetAdvisor”, “PetLive” та “ Vetconsult.gladpet.org” та встановлено, що у цих вебзастосунків відсутня можливість обирати ветеринара за відгуками про нього, записуватись на прийом до ветеринара онлайн, керувати власними прийомами.

Оскільки не існує вебзастосунку, який задовольняв би описані проблеми, то вебзастосунок “HappyPet” може стати важливим рішенням, яке допоможе власникам тварин швидше та простіше записуватись до ветеринарного лікаря та керувати власними прийомами.

Вебзастосунок повинен мати можливість створення облікового запису з вказанням особистих даних власника тварини, записуватись на прийом до ветеринара у режимі онлайн, мати можливість скасовувати прийом, залишати відгук про ветеринара, зебезпечувати інформаційну безпеку даних та мати інтуїтивно зрозумілий інтерфейс.

Оскільки вебзастосунок працює з важливими даними, тому має бути забезпечений належний рівень безпеки. Вебзастосунок має бути стійким до загрози фальсифікування даних, загрози викрадення даних, загрози перехоплення запитів у мережі Інтернет, бути стійким до DoS та DDoS атак.

2. ОБҐРУНТУВАННЯ ВИБОРУ ЗАСОБІВ РЕАЛІЗАЦІЇ

Наступним етапом після аналізу існуючих рішень та опису функціональних вимог до вебзастосунку є вибір засобів реалізації. Вебзастосунок розділяється на дві частини: клієнтську та серверну.

Клієнтська частина відповідає за функції представлення та включає в себе інтерфейс користувача. В цій частині збирається інформація від користувача, формуються запити до сервера за допомогою методів HTTP, таких як GET, POST, PUT та DELETE та передаються до сервера, обробляються та відображаються результати.

В свою чергу, серверна частина відповідає за оброблення та відправлення запитів клієнтській частині, забезпечує керування даними з БД та бізнес-логікою вебзастосунку. Зазвичай, серверна частина вебзастосунку містить більше логіки та обчислень, ніж клієнтська частина, оскільки вона має більше ресурсів та може обробляти більше даних за один раз.

У цьому розділі будуть розглянуті найпопулярніші мови програмування, фреймворки та інші засоби розробки, що можуть бути використанні для розроблення вебзастосунку. Аналізуючи їх переваги та недоліки, будуть вибрані інструменти для створення даного вебзастосунку.

2.1. Вибір мови програмування для розроблення серверної частини

Виходячи з вимог до вебзастосунку було складено перелік критеріїв для вибору мови програмування для серверної частини:

- бути масштабованою для забезпечення потрібної продуктивності;
- мати певну кількість доступних інструментів для розробки, тестування та налагодження;
- мати підтримку від спільноти, для швидкого вирішення проблеми;
- підтримувати поліморфізм;

- виконувати парадигми об’єктно-орієнтованого програмування;
- бути високорівневою;
- підтримувати мультиплатформеність;
- бути статично типізованою;
- підтримувати перевантаження операторів;
- підтримувати асинхронне виконання;
- мати фреймворк для вебзастосунку.

2.1.1. *Мова програмування C#*

C# є мовою програмування з об'єктно-орієнтованою парадигмою, що використовується для розробки програмного забезпечення на платформі .NET. Синтаксис C# подібний до мов програмування C++ і Java. Ця мова має строгу систему статичної типізації, що дозволяє використовувати поліморфізм, перевантаження операторів, атрибути, події, винятки, а також можливість додавання коментарів у форматі XML [5].

C# був спеціально створений для використання на платформі Microsoft .NET Framework, яка забезпечує повну екосистему для розробки, компіляції та виконання програмного коду. .NET Framework включає в себе багатий набір класових бібліотек і середовище виконання, відоме як Common Language Runtime (CLR). Ці компоненти є ключовими складовими для розробки програм на C# та інших мовах програмування, таких як F# і Visual Basic. CLR відповідає за виконання коду програми і надає різноманітні сервіси, що сприяють розробці додатків і забезпечують міжплатформову сумісність. Крім того, він забезпечує розширену підтримку мов програмування високого рівня, що спрощує написання складних програм для розробників.

Також C# має хорошу підтримку для розробки вебзастосунків, зокрема, за допомогою фреймворків ASP.NET та ASP.NET Core. які дозволяють

розробляти високоякісні та безпечні вебзастосунки. ASP.NET Core може бути використано для розгортання вебзастосунків на різних операційних системах, таких як Linux, MacOS і Windows. Це дозволяє розробляти крос-платформні вебзастосунки.

Переваги C#:

- наявність Garbage Collector, який керує розподілом і звільненням програмної пам’яті, усуваючи потребу в ручному керуванні пам’яттю;
- має можливість писати асинхронний код, який спрощує виконання коду на основі зовнішніх ресурсів і завдань;
- підтримує паралельне програмування, що дозволяє виконувати кілька потоків одночасно та ефективно використовувати ресурси обробки системи;
- можливість писати атрибути, які дозволяють розробникам вставляти додаткову описову інформацію в метадані, які можуть бути отримані службами відображення під час виконання;
- належність до суворо типізованих мов – через що код є зрозумілим і дозволяє виявити більшість помилок на етап компіляції;
- можливість використовувати делегати, які визначають типи, які представляють посилання на методи з конкретними списками параметрів та типами повернення;
- наявність подій, що дозволяють отримувати сповіщення про виконання дії об’єктом;
- наявність великої спільноти розробників – спільнота Microsoft для розробників C# і .NET;
- наявність технології ASP.NET для веброзробки;
- має підтримку інтегрованого запиту LINQ.

Недоліки C#:

- тривалий час підтримувалась лише пристроями з ОС Windows;
- при першому запуску можливі затримки виконання коду;
- не найкраща продуктивність, яка визначається виміром часу компіляції та фактичною продуктивністю програми, то в порівнянні з C++ значно поступається;
- залежність від .Net платформи, оскільки сама по собі мова не така гнучка;
- складніша у вивченні, у порівнянні з такими мовами як Python, Perl, PHP або Ruby;
- не має функції низькорівневого програмування, тому відсутня можливість взаємодіяти безпосередньо з апаратним забезпеченням, створювати драйвери або працювати з вбудованим програмним забезпеченням.

2.1.2. *Мова програмування Kotlin*

У 2010 році розробники з JetBrains, компанії, що займається розробкою інструментів програмування, вирішили створити нову мову програмування на базі Java. Вони мали потребу в мові, яка була більш компактною та містила нові конструкції, наприклад, функції високого рівня. На сьогоднішній день Kotlin є однією з найпопулярніших мов, що працюють на віртуальній машині Java (JVM), після самої Java. У 2019 році компанія Google оголосила Kotlin пріоритетною мовою для розробки додатків для платформи Android [6].

Kotlin – це статично типізована мова програмування, яка працює на платформі JVM та розробляється компанією JetBrains. Вона також може бути скомпільована в JavaScript. Назва мови походить від назви острова Котлін, розташованого у Фінській затоці, де знаходиться частина Кронштадту.

Kotlin використовується для розробки різних платформ, включаючи серверну, клієнтську, вебзастосунки та Android-додатки. З'явлення Kotlin/Native дозволило підтримку вбудованих систем macOS та iOS. Мову Kotlin використовують для створення мобільних, серверних та клієнтських додатків з використанням JavaScript або JavaFX, а також для розвитку галузі Data Science [7].

Переваги Kotlin:

- повністю сумісний із Java – усі фреймворки та бібліотеки Java сумісні з Kotlin, тому вони можуть співіснувати;
- є автоматична конвертація Java-коду в Kotlin та навпаки, яка доступна у середовищі розробки IntelliJ IDEA;
- працює на різних платформах (Windows, Mac, Linux, Raspberry Pi);
- наявність спеціальних класів, призначених для зберігання даних, а також для генерації різноманітних шаблонів;
- підтримує функціональне програмування, наприклад дозволяє писати функції вищого порядку, лямбда-вирази та функції розширення;
- легка у вивченні, особливо якщо вже знаєте Java;
- велика зростаюча спільнота/підтримка;
- відкритість вихідного коду;
- null-безпечність мови, яка запобігає виникненню виключень нульового вказівника, що дозволяє писати більш надійний код і підвищує безпеку програм.

Недоліки Kotlin:

- обмежена кількість сторонніх бібліотек, що значно впливає на швидкість розроблення;
- нестабільність інструменту побудови проєкту;

- більш складний обчислювальний процес додатків, створених з використанням Kotlin, у порівнянні з C#;
- іноді Kotlin може працювати ефективніше за Java, зокрема при використанні інкрементних конструкцій.

2.1.3. *Мова програмування JavaScript*

JavaScript є мовою програмування, яка працює у вебсередовищі та забезпечує можливість взаємодії користувача з вебсторінками. Вона має динамічну природу та базується на прототипному стилі програмування. JavaScript є одним з основних інструментів для розробки функціональності клієнтської частини вебсторінок, разом з мовою розмітки HTML та мовою опису стилів CSS. Ця мова програмування дозволяє створювати динамічні та інтерактивні вебелементи [8].

JavaScript початково була створена компанією Netscape як засіб для розробки динамічних та інтерактивних елементів для вебзастосунків. У своєму синтаксисі JavaScript подібна до мови програмування C. Її розробка базується на стандартах ECMAScript, які були розроблені компанією Sun Microsystems [9].

JavaScript, так само як серверні мови PHP і ASP, можна вставляти в будь-яке місце HTML-коду вебсторінки. Після генерації сервером вихідних даних у форматі HTML, код залишається видимим у вихідному коді сторінки. Цей код також можна зберегти як окремий файл з розширенням ".js", який можна відкрити в браузері. JavaScript відрізняється від інших мов програмування завдяки своїй особливій асинхронній моделі виконання коду, яку реалізують браузерні двигуни.

Переваги JavaScript:

- є «інтерпретованою» мовою, що зменшує час, необхідний іншим мовам програмування, для компіляції;

- всі сучасні браузери підтримують JavaScript;
- наявність фреймворку Node.js для написання вебзастосунків;
- простота мови – JavaScript легко зрозуміти та вивчити;
- має високий рівень інтероперабельності, що дозволяє використовувати її разом з різними технологіями та інструментами;
- надає різноманітні можливості для маніпулювання елементами сторінки, створення анімацій, динамічних ефектів та інтерактивності;
- наявність сторонніх додатків, які дозволяють розробникам додавати у свій код фрагменти попередньо визначеного коду, заощаджуючи час при розробленні вебзастосунку;
- допомагає покращити продуктивність вебзастосунків і програм, зменшуючи довжину коду, за рахунок використання вбудованих функцій для циклів, доступу до DOM та інших операцій.

Недоліки JavaScript:

- відсутність засобів налагодження, що сповільнює знаходження помилки розробником;
- підтримує лише одиночне успадкування, для деяких програм може знадобитися ця характеристика об’єктно-орієнтованої мови;
- зберігає число як 64-розрядне число з плаваючою комою, а оператори працюють з 32-розрядними операндами, перетворення з 64-розрядного в 32-розрядне збільшують час виконання сценарію;
- зупинена віртуалізація, наявність однієї помилки в коді може зупинити відтворення всього JavaScript коду на вебзастосунку.

В результаті була обрана мова C# з фреймворком ASP.NET Core MVC, оскільки вона відповідає потребам проекту, забезпечує високу продуктивність, підтримує поліморфізм є високорівневою та строго типізовано та широкі можливості для розробки вебзастосунків. C# є

потужною та ефективною мовою програмування, яка має багато функцій та бібліотек. Фреймворк ASP.NET Core MVC забезпечує розширені можливості для побудови вебзастосунків з модель-представлення-контролер архітектурою, роботою з маршрутизацією, обробкою запитів та багато іншого. Обрана комбінація мови C# з фреймворком ASP.NET Core MVC дозволяє ефективно розробляти потужні вебзастосунки з високою швидкодією та надійністю.

2.2. Вибір технології для розроблення клієнтської частини

На сьогоднішній день існує багато інструментів для розроблення клієнтської частини, але основними технологіями є HTML, CSS і JavaScript. Звичайно, у сучасному світі для пришвидшення розробки використовують різні бібліотеки та фреймворки. Одними з найпопулярніших є Bootstrap, React.js та Angular.js, які і будуть проаналізовані при виборі технології для написання клієнтської частини.

Основні вимоги до технології або фреймворку для розробки клієнтської частини є:

- наявність великої спільноти розробників;
- добре написана документація;
- висока швидкість оновлення стану вебсторінки;
- наявність великої кількості модулів або бібліотек;
- простота та лаконічність коду.

2.2.1. *Фреймворк Bootstrap*

Bootstrap є безкоштовним відкритим фреймворком CSS, що дозволяє зручно створювати елегантні та адаптивні вебінтерфейси. Він має багатий набір готових шаблонів та компонентів, які спрощують розміщення контенту, стилізацію тексту, створення форм, кнопок, навігаційних панелей та багатьох

інших елементів інтерфейсу. Крім базових CSS-стилів, Bootstrap також надає можливість додаткового налаштування за допомогою Sass і має зручні JavaScript-плагіни для динамічної взаємодії з елементами інтерфейсу. Також він забезпечує інструменти для створення власних тем та адаптації фреймворка до потреб конкретного проекту. Завдяки своїй популярності та активній спільноті розробників, Bootstrap є одним з найуспішніших фреймворків для швидкої розробки стильних та сучасних вебінтерфейсів [10].

Bootstrap був створений розробниками Марком Отто і Джейкобом Торнтоном, які працювали у Twitter. Початкова назва фреймворку була "Twitter Blueprint", і він був використаний усередині Twitter для забезпечення єдності та ефективності в їхніх процесах дизайну та розробки. Пізніше, у 2011 році, проект був випущений як відкритий фреймворк під назвою "Bootstrap", що відображає його головну мету – допомогти розробникам швидко та легко почати свої проекти з адаптивним та стильним дизайном. Зараз Bootstrap є одним з найпопулярніших інтерфейсних фреймворків, широко використовуваних розробниками по всьому світу [11].

Переваги Bootstrap:

- документація, де є пояснення, зразки коду для базової реалізації;
- менше кросбраузерних помилок;
- послідовна структура, яка підтримує більшість усіх браузерів і виправлення сумісності з CSS;
- має власну адаптивну сітку, яка складається з рядків і стовпців, що дозволяє створити сітку всередині існуючої замість введення медіа-запитів у файлі CSS;
- дозволяє автоматично змінювати розмір зображення залежно від розміру екрана;
- має кілька плагінів JavaScript, що використовують jQuery;

- наявність великої спільноти дизайнерів та розробників, що дозволяє вирішувати різні технічні проблеми;
- безліч безкоштовних і професійних шаблонів, тем і плагінів, що прискорює процес веброзробки.

Недоліки Bootstrap:

- стилі є багатослівними і можуть призвести до великої кількості виводу в HTML, який не потрібен;
- JavaScript прив’язаний до jQuery і є однією з найпоширеніших бібліотек, тому більшість плагінів не використовуються;
- файли Bootstrap є досить великими, що може сповільнити виконання програми під час першого завантаження;
- заплутаний синтаксис і може бути не зрозумілим для новачків;
- надмірна надлишковість шаблонів – не всі шаблони, що описані у фрейворку, будуть використовуватися.

2.2.2. *Бібліотека React.js*

React (також відомий як React.js або ReactJS) – це відкрита JavaScript бібліотека, створена для розробки користувацьких інтерфейсів. Вона призначена для вирішення проблем, пов'язаних з частковим оновленням вмісту вебсторінки, зокрема при розробці односторінкових додатків. React розробляється компанією Meta та активною спільнотою розробників, яка вносить свій внесок у покращення та розширення функціональності бібліотеки [12].

За допомогою фреймворку React можна створювати великі вебзастосунки, які змінюють дані з часом, без необхідності перезавантаження вебсторінки. Його головними цілями є швидкість, простота та масштабованість. React являє собою компонент вид у шаблоні MVC, оскільки обробляє тільки користувацький інтерфейс у вебзастосунках, і може

бути поєднаним у використанні з іншими JavaScript бібліотеками або в великих фреймворках MVC.

Переваги React.js:

- віртуальний DOM, що дозволяє оновлювати лише певні елементи вебсторінки;
- модульність та декларативність, дозволяє додавати нові елементи, не впливаючи на інші;
- наявність кросплатформного фреймворку React Native;
- односпрямованість потоку даних, що зменшує кількість помилок і полегшує процес налагодження;
- дозволяє швидко писати код без великого шаблону та багатьох складних бібліотек;
- після кожного оновлення версії, React API залишається майже незмінним, що дозволяє легко оновлювати проєкт;
- дозволяє розробникам створювати вебзастосунки, які можуть відтворюватися на сервері, або генерувати файли HTML з коду React: фреймворк Next.js дозволяє створювати вебсторінки, які можна відтворювати на сервері з використанням React, що полегшує розробку динамічних та зручних для SEO вебзастосунків.

Недоліки React.js:

- невпорядкована документація ускладнює вивчення та оволодіння структурою;
- для вивчення всіх можливостей фреймворку потрібен тривалий час;
- немає офіційних бібліотек для обробки загальних функцій інтерфейсних вебзастосунків, таких як маршрутизація, HTTP-запити тощо;

- недостатня документованість, оскільки немає часу робити належну документацію через високі темпи розвитку технології;
- ReactJS використовує JSX, що може бути перешкодою для нових розробників;
- немає роздільної логіки компонента та перегляду за замовчуванням.

2.2.3. *Фреймворк Angular.js*

Angular.js, розроблений Google та відкритим фреймворком JavaScript. Він спеціально призначений для створення односторінкових вебзастосунків, які складаються з одного HTML-файлу, де використовуються CSS та JavaScript. Основною метою Angular.js є розширення можливостей браузерних додатків, використовуючи шаблон модель-вид-контролер (MVC). Крім того, Angular.js надає зручні інструменти для тестування, що допомагають забезпечити якість коду та впевненість в його працездатності [13].

Для забезпечення зв'язку між областями вводу або виводу на вебсторінці та моделлю, яка представляється звичайними JavaScript-змінними, фреймворк використовує HTML разом з додатковими атрибутами.

Переваги Angular.js:

- використання MVC (Model-View-Controller) шаблону, що дозволяє розробникам окремо працювати в одному розділі програми з використанням одного і того ж набору даних;
- можливість двостороннього зв'язування, що дозволяє будь-які зміни в інтерфейсі користувача відображатися в об'єктах програми миттєво і навпаки;
- має вбудовану підтримку відкладеного завантаження компонентів, тому модулі завантажуються з сервера, коли користувач переміщується в програмі;

- має величезне ком’юніті – підтримується Google і величезною спільнотою розробників у всьому світі;
- використовує декларативну парадигму програмування, що полегшує підтримку і читання коду;
- надає готові рішення, що дозволяють виконувати різні завдання шляхом використання вже існуючих модулів;
- просто тестувати, оскільки частини програми знаходяться у відокремлених модулях, що дозволяє завантажувати лише необхідні служби та автоматизувати процес тестування;
- легкість прототипування та підтримка ітераційної розробки, що дозволяє написати мінімально життєвоздатний застосунок в короткий термін.

Недоліки Angular.js:

- складність освоєння, особливо для тих, хто використовув бібліотеку jQuery, адже вона вона базується на маніпулюванні деревом DOM;
- використовує концепцію впровадження залежностей та інверсії керування, що може викликати труднощі для розробників;
- уповільнення роботи з використанням понад 2000 вотчерів;
- відсутність зворотної сумісності із другою версією;
- забагато версій фреймворку, через що розробники повинні вирішувати конфлікти версій і проблеми сумісності;
- складність перенесення застарілого коду на основі js/jquery на архітектуру angular;
- має проблему з оптимізацією для пошукових систем;
- побудований на двох нових мовах TypeScript та RXjs, який потребує досить багато часу для розуміння та освоєння.

Для створення клієнтської частини вебзастосунку, було вибрано поєднання фреймворка Bootstrap та мови програмування JavaScript тому що:

- Bootstrap: Фреймворк Bootstrap надає готові шаблони, компоненти та стилізацію, що дозволяє зручно та ефективно будувати стильні та адаптивні інтерфейси. Він має широкий набір можливостей для розміщення контенту, стилізації елементів, організації навігації та інших компонентів. Використання Bootstrap сприяє створенню єдиної стилістики та прискорює процес розробки.
- JavaScript: Мова програмування JavaScript надає можливості для динамічного та інтерактивного функціоналу вебзастосунку. Використання JavaScript дозволяє створювати взаємодію з користувачем, обробляти події, валідувати форми та реалізовувати інші функції. Вона надає гнучкість та можливість налаштування функціоналу вебзастосунку під потреби вебзастосунку.

Комбінація Bootstrap та JavaScript надає розробникам потужні та зручні інструменти для створення привабливих, адаптивних та функціональних інтерфейсів вебзастосунків. Вони співпрацюють разом, доповнюючи один одного, що дозволяє створити вебзастосунок з професійним дизайном та багатим функціоналом.

2.3. Вибір системи управління базами даних

СУБД (система управління базою даних) – це система програмного забезпечення, яка дозволяє обробляти звернення до бази даних, які надходять від прикладних програм кінцевих користувачів. Іншими словами, СУБД є інтерфейсом між базою даних і прикладними задачами. Системи управління базами даних дозволяють об’єднувати великі обсяги інформації, обробляти, сортувати їх, робити вибірки за певними критеріями і т. д. Основними

функціями СУБД є: визначення даних, оброблення даних та управління ними [14].

Відомо, що існує два основні види БД: реляційні (SQL) та нереляційні (NoSQL). Реляційні бази даних зберігають дані в таблицях, тоді як нереляційні бази даних використовують інші методи, такі як ключ-значення або документи, для зберігання даних, і не використовують таблиці, як це робить реляційна база даних. Реляційні бази даних надають можливість складних запитів, гнучкість і допомагають аналізувати дані. З іншого боку, нереляційні бази даних ефективно працюють з великими обсягами даних, знижують затримки і поліпшують пропускну здатність. Оскільки взаємодія з БД відбувається за допомогою SQL (Structured Query Language), база даних має відповідати вимогам ACID (атомарність, узгодженість, ізольованість, довговічність) та має бути підтримка написання складних запитів з можливістю отримання зв’язних даних – було надано перевагу використанню реляційної бази даних.

Керування даними у вебзастосунку відіграє важливу роль, оскільки при повільній роботі з даними вебзастосунок буде відображати дані з затримкою, що вплине на враження від користування вебзастосунком. До того ж, керування даними має бути захищеним, щоб усунути можливість потрапляння важливих даних до зловмисників.

В результаті аналізу початкових вимог були визначені наступні вимоги до системи управління базами даних:

- безоплатне розповсюдження;
- реляційна модель даних;
- можливість розгортання на віддаленому сервері;
- відповідність вимогам ACID, що захищають цілісність, оскільки відомо як саме запити взаємодіють із базою даних;

- висока продуктивність та швидкість відповіді на запити користувачів;
- можливість роботи з базою даних з різних платформ та оперативних систем;
- підтримка індексів;
- забезпечення безпеки даних, включаючи автентифікацію користувачів, авторизацію доступу до даних та шифрування даних в покої;
- підтримка повнотекстового пошуку;
- можливість створення резервних копій та відновлення бази даних.

2.3.1. *MS SQL Server*

Microsoft SQL Server є розробленим корпорацією Microsoft системою управління базами даних. Вона виконує роль сервера даних, забезпечуючи збереження і надання даних відповідно до запитів інших застосунків. Ці запити можуть виконуватися як на тому ж сервері, так і через мережу. SQL Server дозволяє ефективно керувати базами даних та забезпечує надійність, масштабованість та безпеку для додатків, які використовують ці дані [15].

Для взаємодії з MS SQL Server використовується SQL Server Management Studio (SSMS). MS SQL Server використовує стандартну мову структурованих запитів ANSI SQL. Однак, наряду з ANSI SQL, він також має власну мову програмування – Transact-SQL (T-SQL). T-SQL надає додаткові можливості, такі як визначення збережених процедур, обробка винятків, використання змінних та інших.

У 1987 році Microsoft об'єдналася з Sybase Solutions для розробки системи управління базами даних, яка могла б конкурувати з існуючими гігантами, такими як IBM і Oracle. Початкова версія сервера баз даних, відома як SQL Server 1.0, була випущена приблизно у 1989 році і був практично

ідентичним Sybase SQL Server 3.0, який був доступний для платформ Unix, VMS та інших. Пізніше була випущена версія 4.2, яка уже мала графічний інтерфейс та повністю належала компанії Microsoft. На сьогоднішній день Microsoft SQL Server є однією з найкращих підтримуваних і найпопулярніших RDBMS на ринку [16].

Переваги MS SQL Server:

- дотримання принципів ACID;
- висока продуктивність;
- легкість навчання та використання, синтаксис MySQL мало відрізняється від SQL, а також він підтримує всі наявні типи даних;
- підтримка індексів;
- легке встановлення – в основному для завантаження MS SQL Server вам потрібен мережевий фреймворк, мінімум 1 ГБ пам’яті та система NTFS;
- надання шифрування даних;
- можливість створювати реплікації;
- використовує клієнт-серверну архітектуру, де вона виступає у ролі сервера, до якого з'єднуються клієнтські програми для взаємодії з базою даних;
- можливість відновлювати пошкоджені та втрачені дані;
- наявність тригерів та представлень;
- має здатність ефективно обробляти значні обсяги даних і легко масштабується.

Недоліки MS SQL Server:

- призначений виключно для використання на операційних системах Windows серверного типу;
- вільне розповсюдження лише у середовищі для розробників;

- висока вартість розповсюдження для виробничого середовища;
- не надає повного контролю над базами даних своїм користувачам. Це пов’язано з деякими прихованими правилами бізнесу;
- оптимізація запитів і налаштування продуктивності можуть бути складними для спеціалістів із даних, які не мають глибоких спеціальних знань;
- БД у SQL постійно знаходяться під загрозою, оскільки містять величезні обсяги конфіденційних даних.

2.3.2. *PostgreSQL*

PostgreSQL, також відомий як Postgres, є системою управління базами даних з відкритим кодом, спрямованою на задоволення корпоративних потреб, яка підтримує мову SQL, формат JSON, що дозволяє виконувати реляційні та нереляційні запити, надаючи гнучкість і сумісність зі стандартом SQL. PostgreSQL відрізняється широким спектром потужних функцій, таких як розширені типи даних і оптимізація продуктивності, які зазвичай притаманні тільки комерційним системам управління базами даних, наприклад, Oracle і SQL Server. Ця система має високу репутацію завдяки своїй надійності та здатності ефективно обробляти великі обсяги даних, що робить її популярним вибором для розробників та адміністраторів баз даних [17].

PostgreSQL виник як результат проекту POSTGRES, розпочатого в Каліфорнійському університеті в Берклі в 1986 році і спонсорованого різними організаціями, включаючи DARPA та NSF, під керівництвом Майкла Стоунбрейкера. У 1994 році студенти Ендрю Ю та Джоллі Чен внесли зміни до коду, додаючи інтерпретатор SQL, що зробило модифікацію приблизно на 30-50% швидшою. Вони випустили цю модифікацію як відкрите рішення під назвою Postgres95, розповсюджуючи його за ліцензією, схожою на ліцензії

BSD і MIT. У 1996 році, з випуском версії 6.0, додаток бази даних був офіційно перейменований на PostgreSQL, і ця назва залишається актуальною й досі [18].

PostgreSQL використовує стандартну модель клієнт-сервер, де центральний компонент сервера з назвою "postmaster" керує всіма файлами бази даних та з'єднаннями, що встановлюються для зв'язку з сервером бази даних. Для встановлення з'єднання користувачам потрібна відповідна клієнтська програма і пакет програмного забезпечення PostgreSQL, який містить вбудоване рішення для роботи через командний рядок за допомогою psql. Крім того, можна використовувати різні програми з графічним інтерфейсом користувача, такі як pgAdmin або phpPgAdmin.

PostgreSQL – гнучка та надійна система управління базами даних, яка добре підходить для багатьох галузей та програм. PostgreSQL часто використовується у поєднанні з програмами для аналізу баз даних, що дозволяє досягти високої ефективності аналітичних процесів. Крім цього, PostgreSQL прекрасно поєднується з розширенням PostGIS, яке надає широкий спектр функцій для роботи з геоданими. Часто ця комбінація використовується вебзастосунками, оскільки PostgreSQL сумісний з популярними фреймворками, такими як Node.js, Ruby on Rails або Django, і підтримує класичні вебмови. Також забезпечує синхронну та асинхронну реплікацію, що дозволяє розподілити збережені дані між кількома серверами для забезпечення надійності та мінімізації часу доступу до критично важливої інформації [19].

Переваги PostgreSQL:

- може запускати динамічні вебзастосунки як варіант стеку LAMP;
- є реляційною;
- має ліцензію відкритого коду, що дозволяє використовувати, змінювати та впроваджувати його у вебзастосунок вільно;

- має підтримку географічних об'єктів, що дозволяє використовувати його для створення служб на основі розташування та систем географічної інформації;
- підтримує ACID;
- підтримка індексів;
- асинхронна реплікація;
- можливість відновлення даних;
- безпечне зберігання даних;
- дозволяє лишати коментарі до коду, що полегшує розуміння коду розробником;
- легко розширювати;
- повнотекстовий пошук;
- підтримка тригерів, представлень, функцій і збережуваних процедур.

Недоліки PostgreSQL:

- підвищення швидкості може бути більш трудоємким порівняно з MySQL, оскільки PostgreSQL зосереджується на забезпеченні високої сумісності;
- за показниками продуктивності він повільніший, ніж MySQL;
- здійснення реплікації складніше;
- багато програм з відкритим кодом не підтримують PostgreSQL;
- складність оновлення версії БД;
- зменшення продуктивності під час простих обчислень.

Було обрано систему управління MySQL, оскільки вона має ряд переваг і відповідає вимогам проекту. По-перше, MySQL є відкритою системою з вільною ліцензією, що дозволяє безкоштовне використання і зміну коду, що є важливим для економічної ефективності проекту. По-друге, MySQL є дуже швидким та ефективним в роботі з великими обсягами даних, що важливо для

вебзастосунків з великою кількістю користувачів. Крім того, MySQL підтримує багато мов програмування і має велику спільноту розробників, що забезпечує доступ до розширень та підтримку. Також MySQL відомий своєю надійністю і стабільністю, що робить його привабливим вибором для критичних застосунків.

2.4. Висновки

При виборі мови програграмування для розробки серверної частини вебзастосунку було розглянуто мови C#, Kotlin та JavaScript. В результаті було обрано мову програмування C# з фреймворком ASP.NET Core MVC, оскільки вона є гнучкою, підтримує сучасні технології, є статично типізованою та має високу швиткодію. Фреймворк ASP.NET MVC забезпечує високу продуктивність та масштабованість, що дозволяє розробляти вебзастосунки будь-якої складності. Також цей фреймворк підтримує шаблон проєктування Model-View-Controller (MVC), який дозволяє ефективно організовувати код і забезпечує його легку підтримку та розширення. Крім того, ASP.NET MVC має вбудовану підтримку AJAX, що полегшує розробку вебзастосунків з динамічним змістом.

Для вибору фреймворку для розроблення клієнтської частини вебзастосунку було проаналізовано фреймворки: Bootstrap, Angular.js та бібліотеку React.js. В результаті було обрано фреймворк Bootstrap, оскільки він має хорошо написану документацію, підтримується більшістю браузерів, дозволяє розробляти адаптивний до розміру екрану інтерфейс, а також має безліч безкоштовних і професійних шаблонів, тем і плагінів, що прискорює процес веброзробки.

Для вибору СУБД для вебзастосунку “HappyPet” були розглянуті переваги та недоліки двох популярних СУБД: Microsoft SQL Server та PostgreSQL. В результаті було обрано Microsoft SQL Server, оскільки дана

СУБД була обрана, оскільки вона є реляційною, є легко розширюваною, має велику спільноту розробників, дозволяє безпечно зберігати дані, має можливість відновлювати пошкоджені дані, а також підтримує ACID, індекси, тригери, представлення. Крім того, Microsoft SQL Server є добре інтегрованим з .NET Framework, що спрощує розробку та підтримку вебзастосунку.

3. РЕАЛІЗАЦІЯ ВЕБЗАСТОСУНКА “HAPPYPET”

3.1. Загальний опис системи

Вебзастосунок було розроблено з виконанням усіх вимог до інтерфейсу, безпеки та функціональним можливостям, що були описані у підрозділі 1.4. У вебзастосунку передбачено 3 типа користувачів: «Власник тварини, Ветеринар» та «Адміністратор» та неавторизований користувач.

Власник тварини. Має можливість переглядати усі свої прийоми, а також скасовувати заплановані. Має можливість переглядати усіх своїх пацієнтів. Може редагувати особисті дані.

Ветеринар. Має можливість обирати час та дату і записатись на прийом до ветеринара. Наявна можливість переглядати заплановані прийоми та є можливість скасувати прийом. Може редагувати особисті дані.

Адміністратор. Може видаляти зареєстрованих користувачів та змінювати роль користувача у системі, редагувати особусті дані користувача.

Авторизовані та неавторизовані користувачі можуть переглядати усіх зареєстрованих у системі ветеринарів, проводити пошук за їх спеціалізацією та по імені. А також переглядати детальнішу інформацію про ветеринара.

На початку роботи користувач потрапляє на головну сторінку системи, де описана загальна інформація про вебзастосунок “HappyPet”.

Користувач може авторизуватись або зареєструватись у системі. Для авторизації необхідно ввести логін та пароль. Для реєстрації: прізвище, ім’я, логін, номер телефону, електронна пошта. Для ролі «Ветеринар» передбачено додаткове поле, де вказується спеціалізація ветеринара.

На рис. 3.1 наведено діаграму використання для вебзастосунку.

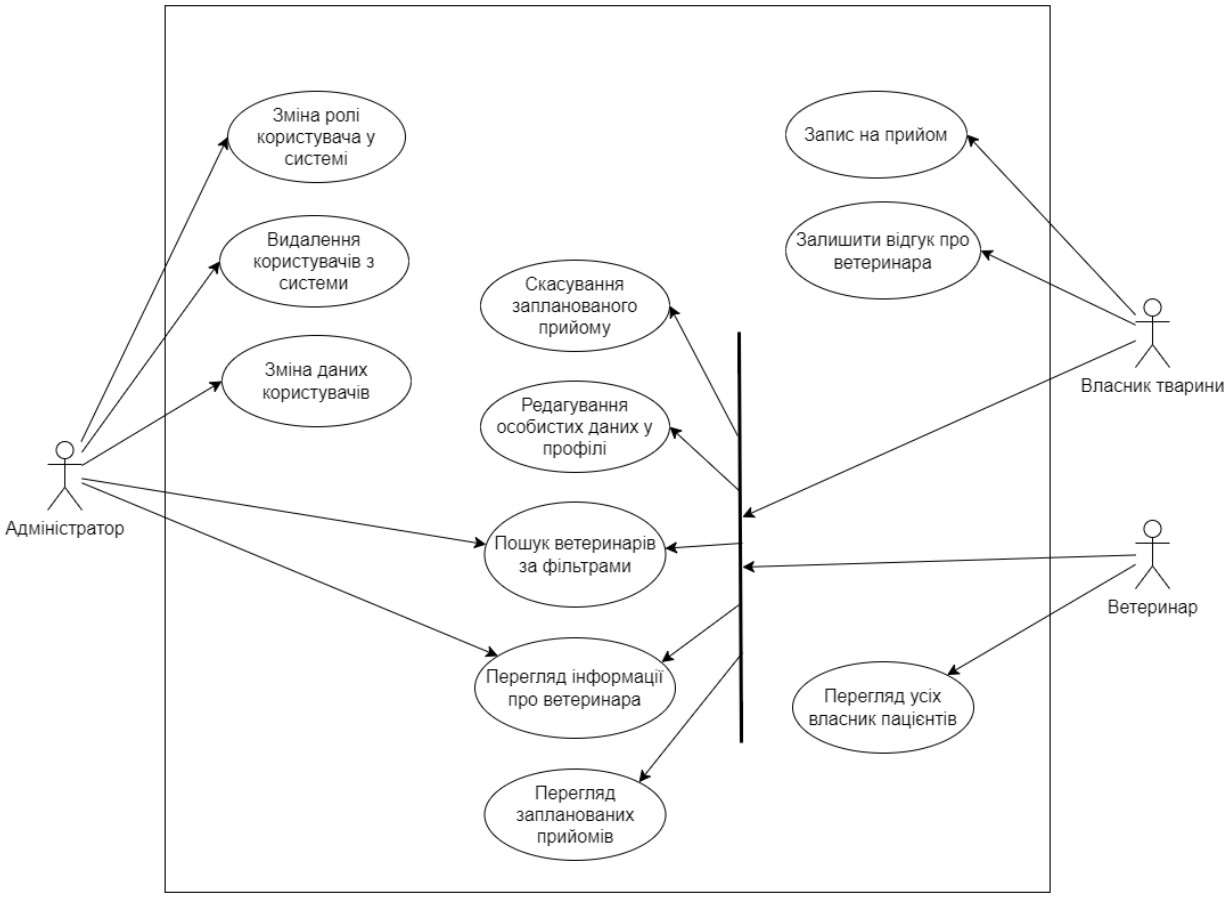


Рис. 3.1. Діаграма використання вебзастосунку “HappyPet”

3.2. Визначення вимог та їх реалізація у вебзастосунку

На першому етапі розробки вебзастосунку проводиться збір, аналіз та документування. вимог Це дозволяє чітко визначити межі розроблюваного вебзастосунку, зафіксувати поставлені цілі та спланувати процес розроблення вебзастосунку. Неправильно або неповно сформульовані вимоги можуть призвести до розробки вебзастосунку, який не відповідає потребам користувачів та потреби у внесенні змін після завершення розробки, що вимагає додаткових зусиль і ресурсів.

Для визначення функціональних вимог були описані можливі сценарії взаємодії користувачів з вебзастосунком у форматі таблиць use cases. Такий підхід структуровано відображає послідовну інтеракцію користувача з системою та описує поведінку системи під час отримання запитів або

відповідей. Нижче наведено усі можливі сценарії використання вебзастосунку.

Таблиця 3.1

UC-1: «Створення облікового запису і/або авторизація у системі»

UC-1	Створення облікового запису і/або авторизація у системі
Актори	Користувач вебзастосунку “HappyPet”
Передумови	• Посилання на вебзастосунок “HappyPet”. • Наявність електронної пошти у користувача.
Постумови	Користувач має особисту сторінку у вебзастосунку, може її переглядати та використовувати всі функції системи.
Нормальна послідовність дій	1. Користувач натискає на поле «Реєстрація». 2. Система відображає вікно для реєстрації. 3. Користувач обирає роль у системі – «Власник тварини» або «Ветеринар». 4. Користувач вводить свою електронну пошту, логін, пароль. 5. Користувач підтверджує введену інформацію. 6. Система перевіряє коректність введеної інформації. 7. Система створює новий обліковий запис. 8. Система відкриває сторінку авторизації вебзастосунку “HappyPet”.
Альтернативний хід	1а. Відвідувач натискає на поле «Вхід». 1а-1. Система відображає вікно для авторизації. 1а-2. Користувач вводить свій логін та пароль. 1а-3. Користувач натискає на клавішу «Вхід». 1а-4. Система перевіряє коректність введеної інформації. 1а-5. Система відкриває головну сторінку вебзастосунку “HappyPet”.

Результат виконання тестового сценарію UC-1: «Створення облікового запису і/або авторизація у системі» за нормальною послідовністю дій показано на рис. 3.2.

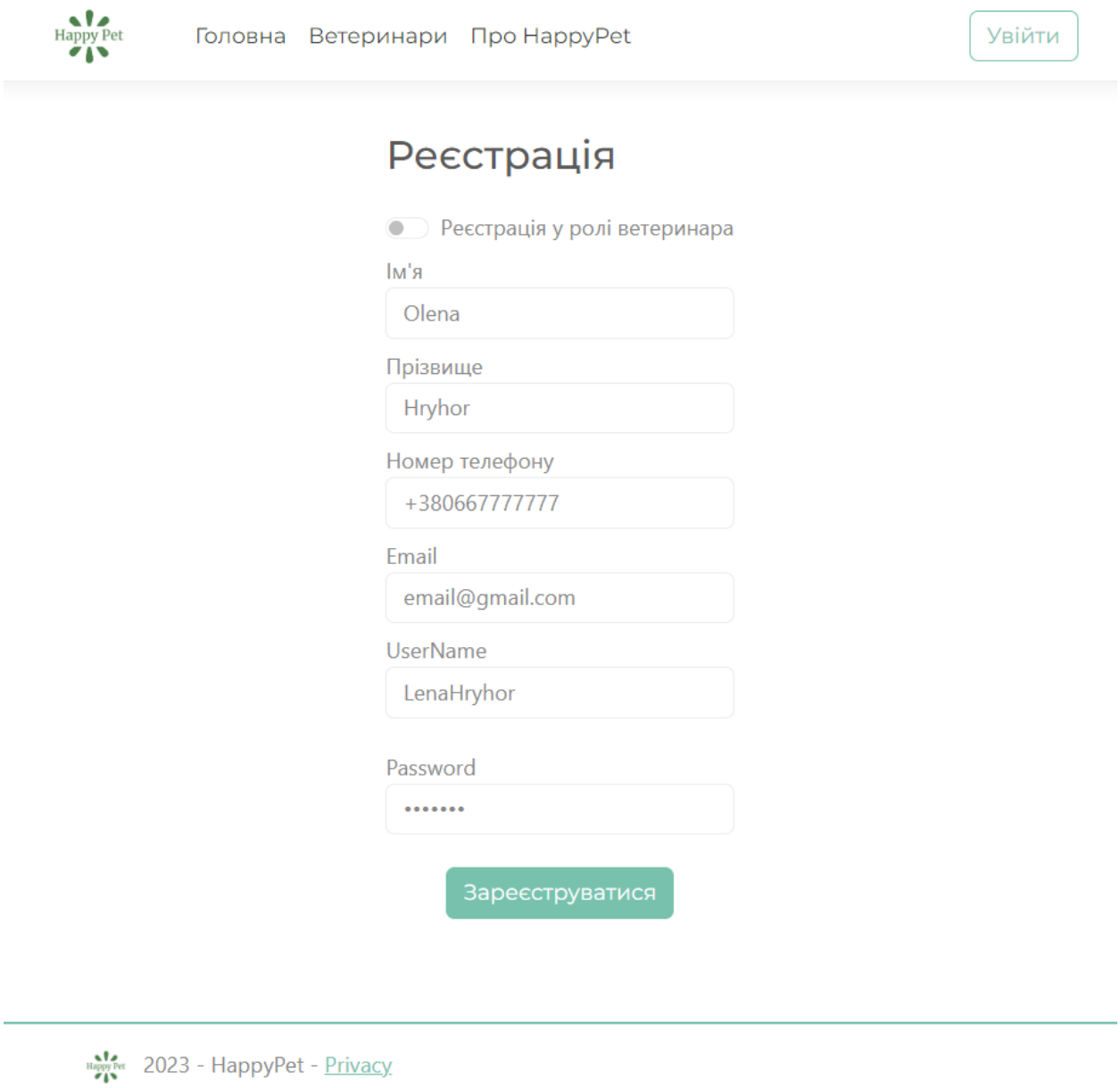


Рис. 3.2. Створення облікового запису у вебзастосунку “HappyPet”

Таблиця 3.2

UC-2: «Запис на прийом до ветеринара»

UC-2	Запис на прийом до ветеринара
Актори	Користувач вебзастосунку “HappyPet”
Передумови	• Користувач авторизувався у вебзастосунку у ролі «Власник тварини». • Користувач натиснув на перегляд детальної інформації про ветеринара.
Постумови	Користувач має запланований прийом до ветеринара. Ветеринар має запланований прийом власника тварини.
Нормальна послідовність дій	1. Користувач обирає спеціалізацію ветеринара, за якою він звертається. 2. Користувач обирає бажаний час і дату для прийому. 3. Користувач натискає «Записатись». 4. Система додає новий прийом в список запланованих прийомів.

Таблиця 3.3

UC-3: «Перегляд інформації про ветеринара»

UC-3	Перегляд інформації про ветеринара
Актори	Користувач вебзастосунку “HappyPet”
Передумови	Користувач натиснув на ветеринара, якого обрав у списку ветеринарів.
Постумови	Користувач має змогу переглянути детальну інформацію про ветеринара та вільний час для запису на прийом.
Нормальна послідовність дій	1. Користувач натискає на ветеринара, якого попередньо обрав зі списку. 2. Система відображає інформацію про ветеринара та вільні для запису на прийом часові слоти.

Результат виконання тестових сценаріїв UC-2: «Запис на прийом до ветеринара» та UC-3: «Перегляд інформації про ветеринара» за нормальною послідовністю дій показано на рис. 3.3.

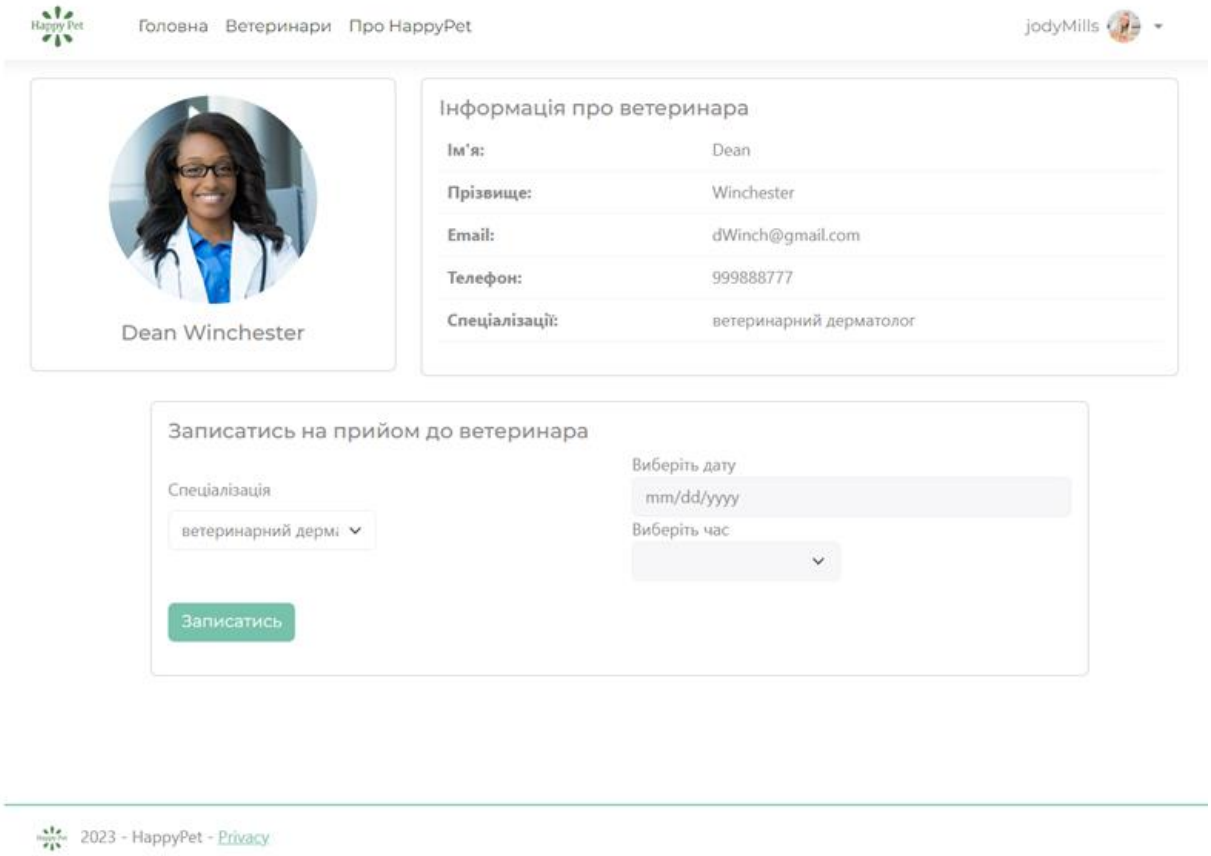


Рис. 3.3. Запис на прийом до ветеринара

Таблиця 3.4

UC-4: «Пошук ветеринара»

UC-4	Пошук ветеринара
Актори	Користувач вебзастосунку “HappyPet”
Передумови	Посилання на вебзастосунок “HappyPet”.
Постумови	Користувач знайшов необхідного ветеринара за допомогою сортування за різними параметрами і/або введення інформації у поле пошуку.

Продовження табл. 3.4

Нормальна послідовність дій	1. Користувач натискає на поле «Ветеринари». 2. Система відображає список усіх доступних ветеринарів. 3. Користувач має змогу ввести прізвище ветеринара у рядок пошуку та натиснути «Пошук». 4. Система відображає список ветеринарів, які задовільняють запит користувача. 5. Система відображає список задач, відсортованих за обраними категоріями.
Альтернативний хід	3а. Користувач може фільтрувати список ветеринарів за спеціалізацією.

Результат виконання тестового сценарію UC-4: «Пошук ветеринара» за нормальною послідовністю дій показано на рис. 3.4.

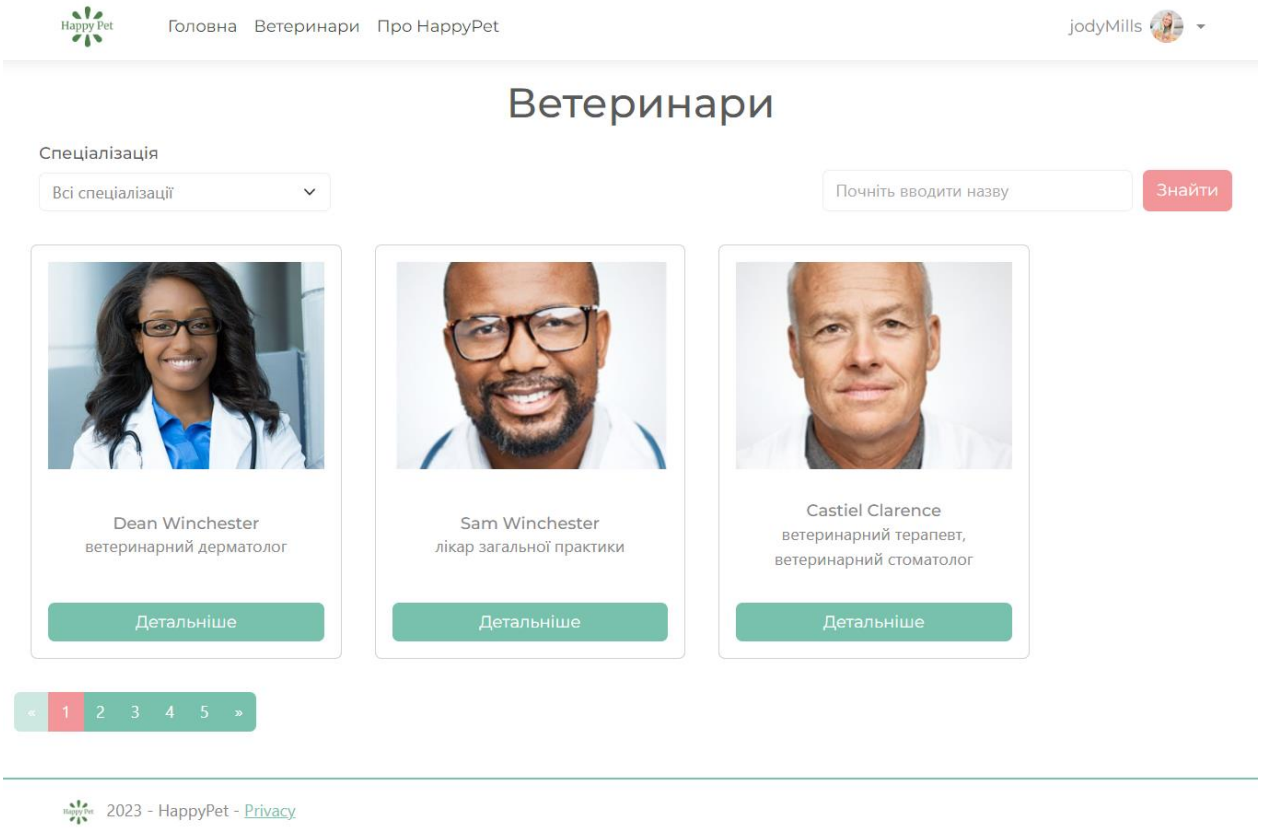


Рис. 3.4. Пошук ветеринара

Таблиця 3.5

UC-5: «Перегляд та скасування запланованих прийомів до ветеринара»

UC-5	Перегляд запланованих прийомів до ветеринара
Актори	Користувач вебзастосунку “HappyPet”
Передумови	Користувач авторизувався у вебзастосунку у ролі «Ветеринар» або «Власник тварини».
Постумови	Користувач має змогу переглядати заплановані прийоми.
Нормальна послідовність дій	1. Користувач натискає на «Прийоми». 2. Система відображає сторінку з усіма запланованими прийомами до ветеринара. 3. Користувач може переглянути детальну інформацію про прийом, натиснувши кнопку «Детальніше». 4. Система відображає детальну інформацію про запланований прийом до ветеринара.
Альтернативний хід	4a. Користувач може скасувати запланований прийом натиснувши на кнопку «Скасувати прийом». 4a-1. Користувач має підтвердити скасування прийому. 4a-2. Система відображає повідомлення про успішне видалення та повідомляє ветеринара/власника тварини, в залежності від того хто скасував прийом.

Результат виконання тестового сценарію UC-5: «Перегляд та скасування запланованих прийомів до ветеринара» за нормальною послідовністю дій показано на рис. 3.5.

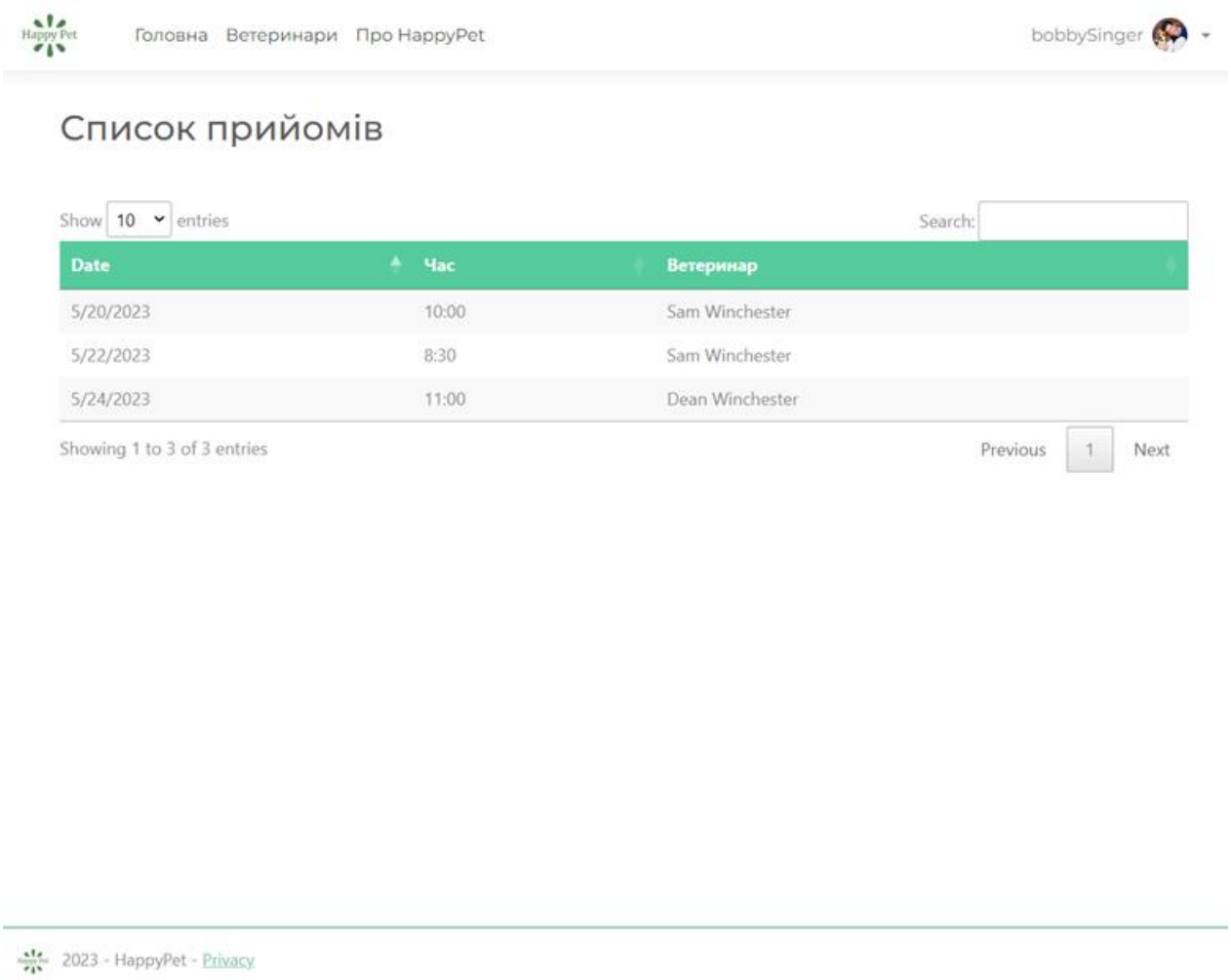


Рис. 3.5. Перегляд запланованих прийомів до ветеринара

Таблиця 3.6

UC-6: «Перегляд усіх пацієнтів ветеринара»

UC-6	Перегляд усіх пацієнтів ветеринара
Актори	Користувач вебзастосунку “HappyPet”
Передумови	Користувач авторизувався у вебзастосунку у ролі «Ветеринар».
Постумови	Користувач переглядає усіх пацієнтів.
Нормальна послідовність дій	1. Користувач натискає на «Пацієнти». 2. Система відображає сторінку з усіма пацієнтами ветеринара.

Результат виконання тестового сценарію UC-6: «Перегляд усіх пацієнтів ветеринара» за нормальною послідовністю дій показано на рис. 3.6.

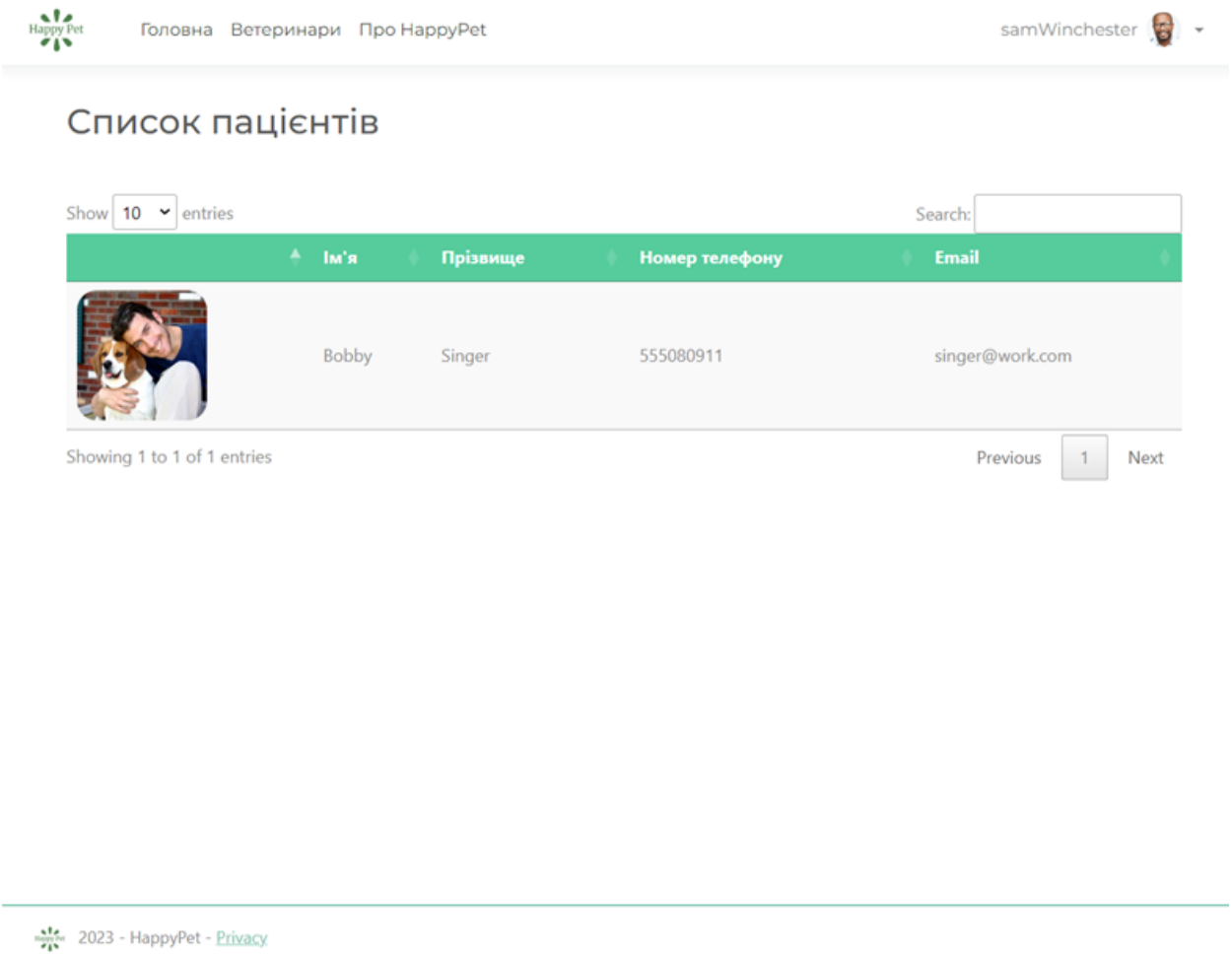


Рис. 3.6. Перегляд усіх пацієнтів ветеринара

3.2.1. *Алгоритм роботи вебзастосунку “HappyPet”*

Усі користувачі, незалежно від авторизації, мають доступ до загального функціоналу вебзастосунку, який включає наступні можливості:

- перегляд головної сторінки вебзастосунку “HappyPet”;
- перегляд списку всіх ветеринарів;
- можливість використовувати фільтри для пошуку ветеринарів за певними критеріями.

Далі описаний алгоритм роботи вебзастосунку “HappyPet”:

1. Користувач відкриває вебзастосунок “HappyPet”.
2. Визначається режим доступу користувача: авторизований або неавторизований.
3. Якщо користувач у режимі неавторизованого доступу, він може скористатись функціями реєстрації та авторизації.
4. Незалежно від режиму доступу, користувач має можливість переглядати головну сторінку вебзастосунку “HappyPet”, список всіх ветеринарів та використовувати фільтри для пошуку ветеринарів за певними критеріями.
5. Якщо користувач авторизований, йому стає доступним додатковий функціонал залежно від його ролі в системі.
6. Користувач з роллю «Власник тварини» може записатись на прийом до ветеринара. Для цього він здійснює пошук ветеринара за фільтрами, обирає ветеринара, переглядає сторінку обраного ветеринара та вибирає вільну дату та час для прийому.
7. Користувач з роллю «Власник тварини» або «Ветеринар» може скасувати заплановані прийоми. Для цього він переходить до списку запланованих прийомів, обирає прийом, який потрібно скасувати, і виконує відповідну дію.
8. Користувач з роллю «Адміністратор» може редагувати дані користувачів, змінювати їх ролі в системі та видаляти користувачів з системи. Для цього він переглядає список усіх користувачів системи, обирає необхідного користувача та здійснює відповідну дію.
9. У будь-який момент користувач може вийти зі свого облікового запису, завершивши сеанс роботи з вебзастосунком “HappyPet”.

На рис. 3.7 наведений алгоритм роботи вебзастосунку “HappyPet”.

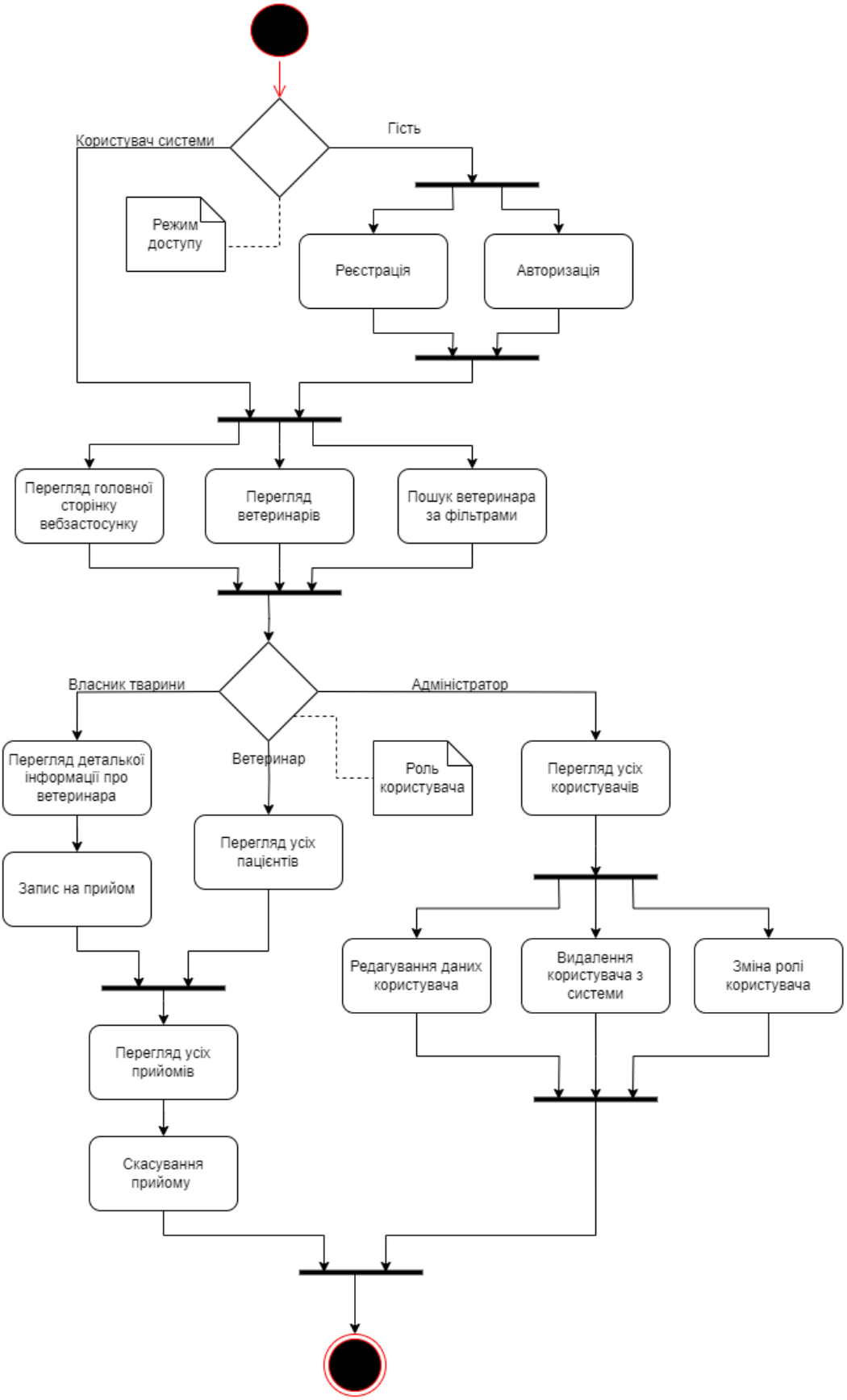


Рис. 3.7. Алгоритм роботи вебзастосунку. Діаграма діяльності

3.3. Опис архітектури вебзастосунку

Архітектура вебзастосунку визначає структуру, організацію та взаємозв'язки компонентів системи, що забезпечують її взаємодію з користувачами та іншими системами, може бути побудована на основі різних підходів та технологій, таких як клієнт-серверна архітектура, модель MVC (Model-View-Controller), мікросервісна архітектура та інші. Архітектура визначає, як компоненти, такі як користувацький інтерфейс, серверна логіка та бази даних, взаємодіють між собою та як здійснюється передача даних між ними.

Найпоширенішим типом архітектури для вебзастосунків є клієнт-серверна архітектура, тому при розробленні вебзастосунку було обрано саме її. У цій моделі, клієнти (зазвичай веббраузери) взаємодіють з сервером за допомогою протоколу HTTP (або іншого протоколу). Сервер обробляє запити клієнтів, виконує логіку бізнес-процесів та надсилає відповідь. Ця архітектура дозволяє розділити функціональність між клієнтом і сервером, забезпечуючи масштабованість та ефективність системи.

Оскільки для реалізації був обраний фреймворк ASP.NET MVC, то архітектура вебзастосунку реалізує шаблон Model-View-Controller (MVC). При такому підході система розділяється на три частини: модель (model), представлення (view) та контролер (controller). Модель забезпечує доступ до даних та їхню маніпуляцію відповідно до правил бізнес-логіки. Представлення відповідає за створення графічного інтерфейсу та відображення інформації для користувача. Контролер обробляє вхідні дані від користувача, виконує відповідні дії та взаємодіє з моделлю та представленням для забезпечення потрібної функціональності. Такий підхід забезпечує розділення відповідальностей, яке полегшує розробку та підтримку вебзастосунків.

На рис. 3.8 наведена структура вебзастосунку “HappyPet”.

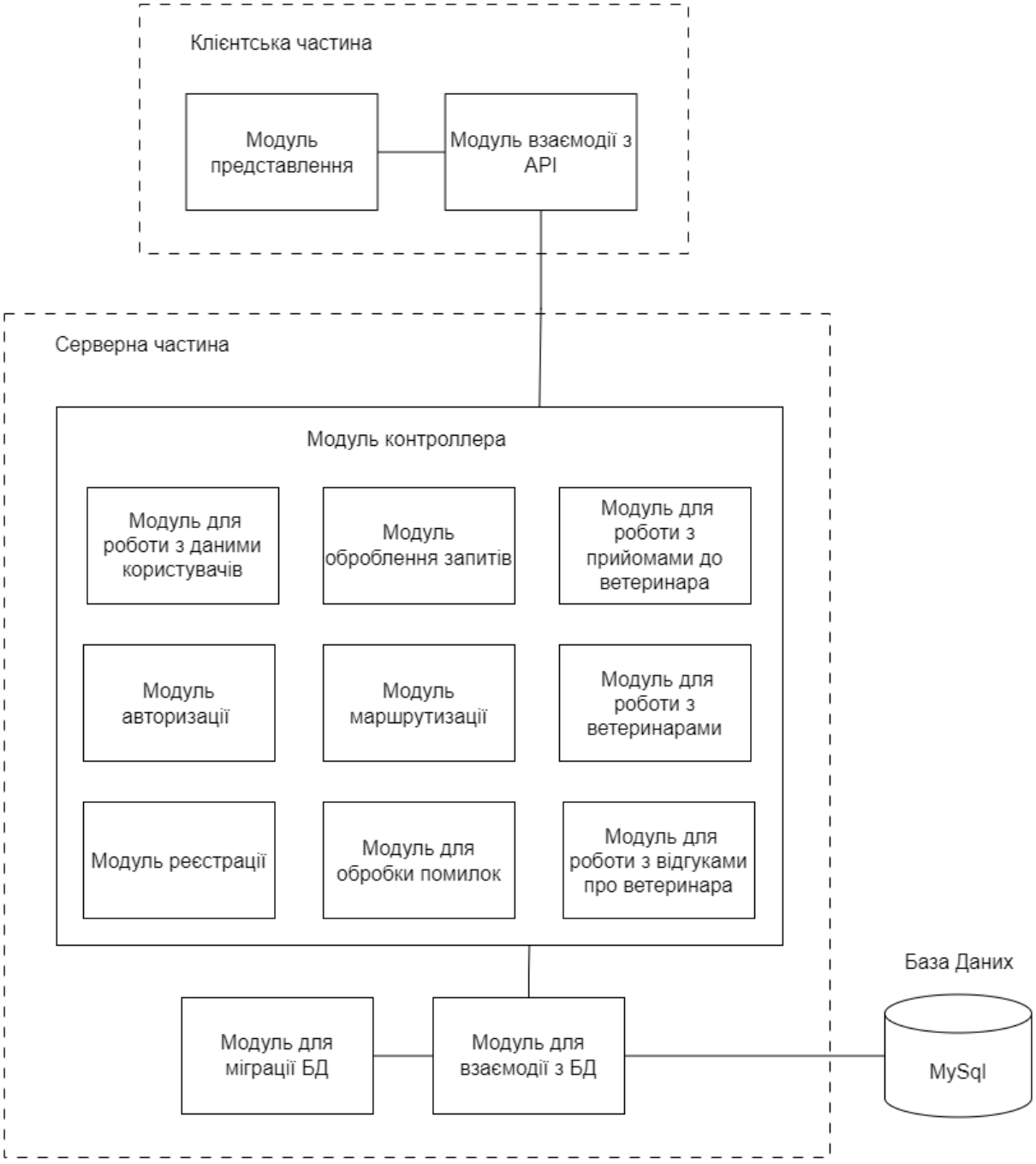


Рис. 3.8. Структура вебзастосунку “HappyPet”

Діаграма взаємодії компонентів Model, View Controller представлена на рис. 3.9.

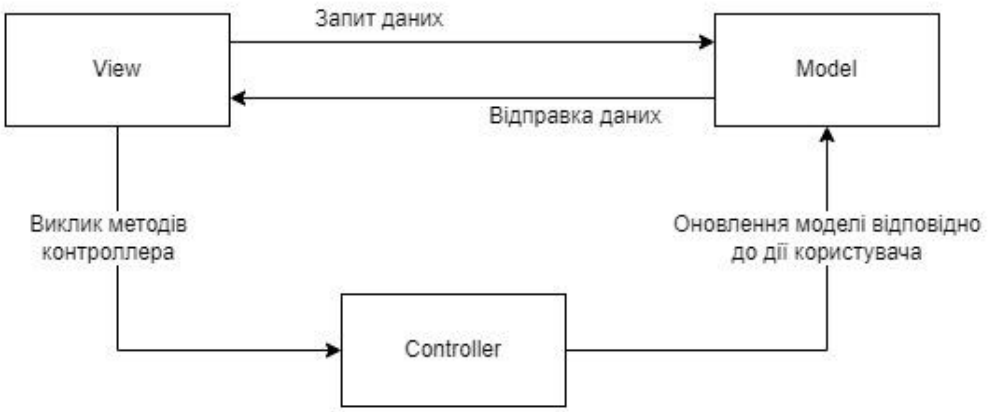


Рис. 3.9. Діаграма взаємодії між компонентами шаблону MVC

Нижче наведено переваги та недоліки використання шаблону MVC.

Переваги:

- Дозволяє чітко розділити логіку вебзастосунку на окермі компоненти: Модель, Представлення та Контролер, що полегшує розробку, підтримку та розширення вебзастосунку.
- Наявність модульності дозволяє розробляти та тестувати кожен компонент окремо. Це призводить до більшої гнучкості і зручності в розробці.
- Наявність окремих модулів дозволяє перевикористовувати раніше розроблені компоненти.
- Наявність окремих компонентів сприяє паралельній розробці, а тому розробники можуть одночасно працювати над різними частинами вебзастосунку, що дозволяє зменшити час розробки вебзастосунку.

Недоліки:

- Для вебзастосунків з невеликим обсягом функціональності, використання архітектури MVC може бути надмірним і призводити до збільшення складності розробки та зайвих витрат ресурсів.

- Іноді використання архітектури MVC може призвести до зайвого перенавантаження, оскільки вебзастосунок потребує додаткових процесорних ресурсів для обробки логіки переключення між компонентами.
- Може бути складна для розуміння для початківців, оскільки потребує глибокого розуміння концепції.

Модулі вебзастосунку “HappyPet”:

1. Models: Моделі представляють логіку вебзастосунку та даних, включаючи взаємодію з базою даних або зовнішніми сервісами. Вона містить класи, які визначають сутності вебзастосунку, такі як користувачі, прийоми, пацієнти та інші.
2. Views: Представлення відповідають за відображення даних користувачеві та обробку його взаємодії з вебзастосунком. Представлення отримують дані від контролера та відображають їх відповідно до вимог користувача.
3. Controllers: Контролери обробляють запити користувача та взаємодіє з моделлю та представленням. Контролер отримує запит від користувача, виконує необхідні дії з даними, які передаються моделі, та повертає результат у відповідь на запит. Він також відповідає за маршрутизацію запитів до відповідних методів обробки.
4. Configurations: Містить класи, які дозволяють керувати відповідними сутностями в БД та встановлюють залежності між таблицями в БД. Файл “AutoMapperProfile.cs” встановлює відповідність між сутностями БД та сутностями вебзастосунку.
5. Routing: Маршрутизація визначає, який контролер та його метод будуть викликані для обробки запиту користувача. У ASP.NET Core MVC маршрутизація використовує шаблони маршрутів, які вказують на шлях до певного контролера та його методу. Конфігурація

маршрутів здійснюється у файлі "Startup.cs", де визначаються шаблони URL та зв'язок із відповідними контролерами.

6. Middleware: Middleware у ASP.NET Core дозволяє додавати компоненти обробки запиту між сервером та додатком, такі як аутентифікація, авторизація, логування, компресія даних тощо. Застосовується до запиту перед його обробкою контролером.
7. База даних: Модуль для зберігання та отримання інформації з БД. Для взаємодії з базою даних MySQL використовується фреймворк Entity Framework Core.

3.4. Опис бази даних вебзастосунку “HappyPet”

Для створення бази даних була розроблена схема бази даних, яка зображена на рис. 3.10.

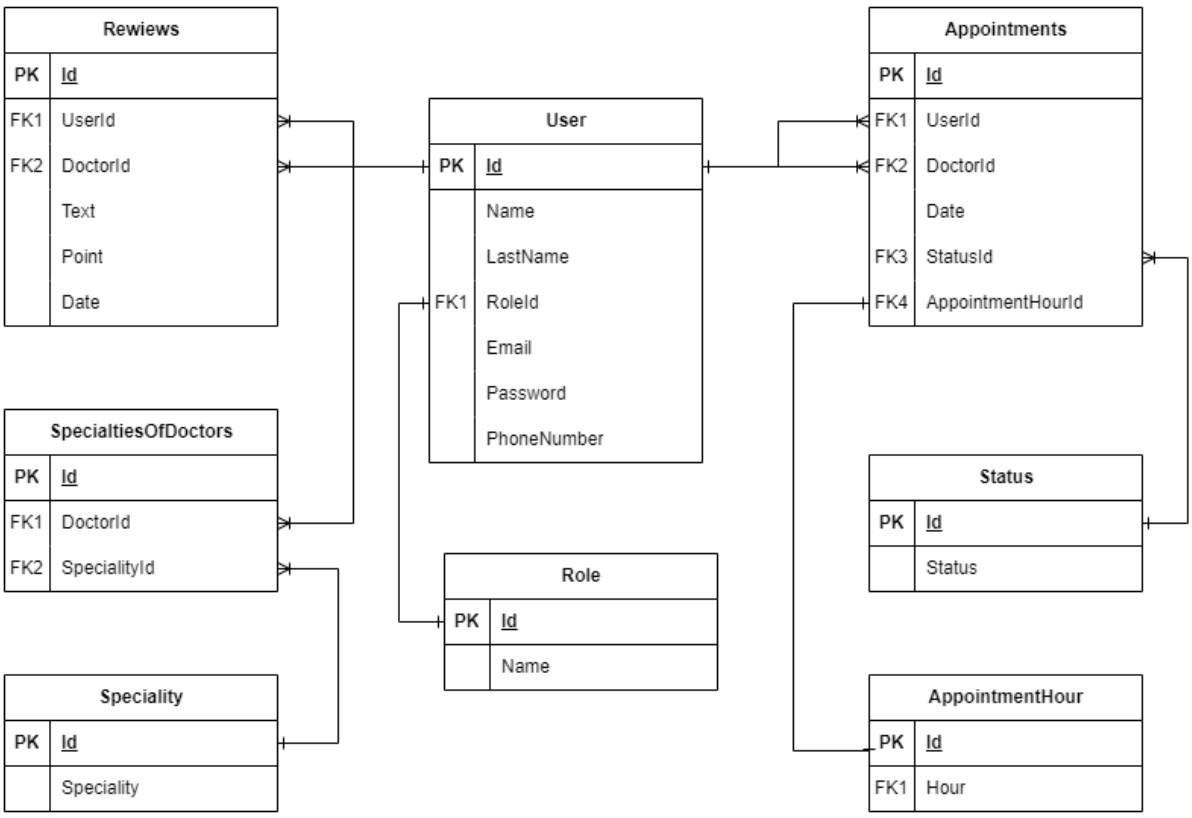


Рис. 3.10. ERD структура бази даних вебзастосунку

В результаті аналізу інформаційних потоків та використання даних у застосунку було створено 8 таблиць, що зберігають дані вебзастосунку. Нижче описано ці таблиці.

User зберігає дані про зареєстрованих користувачів вебзастосунку.

Таблиця 3.7

Структура таблиці User

Назва поля	Тип даних
id	AUTOINCREMENT
Name	string
LastName	string
Email	string
PasswordHash	string
Email	string
PhoneNumber	string
RoleId	int

Також для таблиці User створені пов’язані таблиці Specialization, SpecializationOfDoctors та Roles.

SpecializationOfDoctors містить спеціалізацію ветеринарів.

Таблиця 3.8

Структура таблиці SpecializationOfDoctors

Назва поля	Тип даних
id	AUTOINCREMENT

Продовження табл. 3.8

SpecializationId	int
DoctorId	int

Roles – зберігає типи користувачів вебзастосунку.

Всього передбачено три типи користувачів:

- «Власник тварини»;
- «Ветеринар»;
- «Адміністратор».

Таблиця 3.9

Структура таблиці Roles

Назва поля	Тип даних
id	AUTOINCREMENT
RoleName	string

Specialization зберігає назви спеціалізації ветеринарів.

Таблиця 3.10

Структура таблиці Specialization

Назва поля	Тип даних
id	AUTOINCREMENT
Speciality	string

Appointments зберігає усі прийоми власників тварин до ветеринара.

Таблиця 3.11

Структура таблиці Appointments

Назва поля	Тип даних
id	AUTOINCREMENT
UserId	int
DoctorId	int
Date	datetime
StatusId	string
AppointmentHourId	int

AppointmentHour зберігає доступні часові слоти для прийому.

Таблиця 3.12

Структура таблиці AppointmentHour

Назва поля	Тип даних
id	AUTOINCREMENT
Hour	string

Reviews зберігає відгуки про ветеринара.

Таблиця 3.13

Структура таблиці Reviews

Назва поля	Тип даних
id	AUTOINCREMENT
UserId	int
DoctorId	int

Продовження табл. 3.13

Date	Datetime
Text	string
Point	int

StatusTypes визначає статуси прийомів.

Всього передбачено три типи статусу: запланований, скасований та завершений.

Таблиця 3.14

Структура таблиці StatusTypes

Назва поля	Тип даних
id	AUTOINCREMENT
Status	string

Нижче розглянуто основні запити до БД, що використовуються у вебзастосунку “HappyPet”.

Для роботи з таблицею Users використовуються наступні запити:

1. Створення таблиці Users (лістинг 1).

Лістинг 1. Створеннння таблиці Users

```
CREATE TABLE [HappyPet].[dbo].[Users] (
    [id] INT PRIMARY KEY,
    [Name] VARCHAR(50),
    [LastName] VARCHAR(50),
    [Email] VARCHAR(100),
    [Password] VARCHAR(100),
    [PhoneNumber] VARCHAR(20)
);
```

2. Додавання нового користувача (лістинг 2).

Лістинг 2. Додавання нового користувача у таблицю Users

```
INSERT INTO [HappyPet].[dbo].[Users]
([id], [Name], [LastName], [Email], [Password], [PhoneNumber])
VALUES (1, 'John', 'Doe', 'johndoe@example.com', 'password123',
'1234567890');
```

3. Зміна полів існуючого користувача (лістинг 3).

Лістинг 3. Зміна полів користувача у таблиці Users

```
UPDATE [HappyPet].[dbo].[Users]
SET [Name] = 'Jane', [LastName] = 'Smith',
[Email] = 'janesmith@example.com'
WHERE [id] = 1;
```

4. Видалення користувача з таблиці (лістинг 4).

Лістинг 4. Видалення користувача з таблиці Users

```
DELETE FROM [HappyPet].[dbo].[Users] WHERE [id] = 1;
```

Для роботи з прийомами до ветеринара використовуються запити:

1. Додавання запису на прийом (лістинг 5).

Лістинг 5. Додавання нового запису у таблицю Appointments

```
INSERT INTO [HappyPet].[dbo].[Appointments]
([id], [UserId], [DoctorId], [Date], [Status],
[AppointmentHourId])
VALUES (1, 1, 1, '2023-06-04', 'Scheduled', 1);
```

2. Створення таблиці Appointments (лістинг 6).

Лістинг 6. Створеннння таблиці Appointments

```
CREATE TABLE [HappyPet].[dbo].[Appointments](
    [id] INT PRIMARY KEY,
    [UserId] INT,
    [DoctorId] INT,
    [Date] DATE,
    [Status] VARCHAR(50),
    [AppointmentHourId] INT,
FOREIGN KEY (UserId) REFERENCES [HappyPet].[dbo].[Users]([id]),
FOREIGN KEY (DoctorId) REFERENCES
[HappyPet].[dbo].[Doctors]([id]),
FOREIGN KEY (AppointmentHourId) REFERENCES
[HappyPet].[dbo].[AppointmentHours]([id])
);
```

3. Видалення запису з таблиці Appointments (лістинг 7).

Лістинг 7. Видалення запису з таблиці Appointments

```
DELETE FROM [HappyPet].[dbo].[Appointments]
WHERE [id] = 1;
```

3.5. Висновок

Розроблено вебзастосунок “HappyPet” з виконанням усіх вимог до інтерфейсу, безпеки та функціональним можливостям описаних у підрозілі 1.4. У вебзастосунку передбачено 3 типа користувачів: «Власник тварини, Ветеринар» та «Адміністратор» та неавторизований користувач.

Було представлено сценарії взаємодії користувачів з вебзастосунком та описано їх у формі таблиць використання (use cases), що дозволяє структуровано відобразити послідовну взаємодію користувача з системою і

надати опис поведінки системи під час обробки запитів і надання відповідей. Також було описано загальний алгоритм роботи вебзастосунку “HappyPet”.

Для архітектури вебзастосунку був обраний шаблон проєктування MVC (Model-View-Controller), який дозволяє розділити логіку системи на три основні компоненти: модель, представлення та контролер. Такий підхід дозволяє чітко розподілити відповідальності між компонентами системи, що спрощує розробку, підтримку та тестування вебзастосунку, а також забезпечує гнучкість архітектури. Вебзастосунок складається з таких модулів: Models, Views, Controllers, Configurations, Routing, Middleware та модуль для роботи з базою даних:

Після аналізу вимог до вебзастосунку були визначені дані, які потрібно зберігати у БД в результаті було створено 8 таблиць для зберігання даних: User, Specialization, SpecializationOfDoctors, Roles, Appointments, AppointmentHour, StatusTypes та Reviews. Було розглянуто основні типи запитів до БД мовою SQL у вебзастосунку “HappyPet” – таких як створення таблиці, додавання нового користувача у таблицю Users, редагування даних користувача, видалення користувача, створення та видалення прийому з таблиці Appointments, пошук ветеринара по імені.

4. АНАЛІЗ РОЗРОБЛЕНОГО ВЕБЗАСТОСУНКУ

4.1. Особливості тестування вебзастосунку

Тестування є важливою складовою процесу розробки програмного забезпечення, основною метою якого є збір інформації про якість програмного продукту. Також тестування ПЗ запобігає низці ризиків, що виникають при використанні вебзастосунку користувачем.

Для перевірки відповідності розробленого програмного забезпечення поставленим вимогам був застосований метод димового тестування.

Димове тестування – це тип тестування програмного забезпечення, який спрямований на перевірку основних функцій програмного забезпечення та виявленні очевидних помилок та проблем. Цей метод передбачає проведення обмеженого обсягу тестів з мінімальними навантаженнями для перевірки основного функціоналу програми та її стабільності. Після проведення димового тестування можна отримати початкову оцінку проекту та виявити можливі дефекти, які потребують подальшого виправлення. Такий підхід дозволяє забезпечити початкову якість програмного забезпечення та виявити проблеми на ранніх етапах розробки [19].

Для проведення димового тестування було складено тестові сценарії. Ці сценарії були структуровані у вигляді таблиці, що дозволяє систематично оцінити, чи відповідає система вимогам та чи функціонує належним чином.

Тестування проведено на машині з ОС Windows 10 Pro, у браузері Google Chrome 113.0. Тестові сценарії описано нижче та наведено результат виконання програми.

Таблиця 4.1

Створення облікового запису у системі

Передумова	Користувач не має облікового запису у системі.
Сценарій дій	1. Користувач відкрив вебзастосунок “HappyPet”. 2. Користувач натискає на поле «Реєстрація». 3. Користувач вводить обирає роль (власник тварини або ветеринар), заповнює поля з даними про ПІБ, email та пароль. Для ролі ветеринар необхідно обрати спеціалізацію. 4. Користувач підтверджує введену інформацію, натиснувши на клавішу «Реєстрація».
Очікуваний результат	Користувач успішно зареєстрований у системі.

Результат виконання тестового сценарію описаного у табл. 4.1 наведений на рис. 4.1.

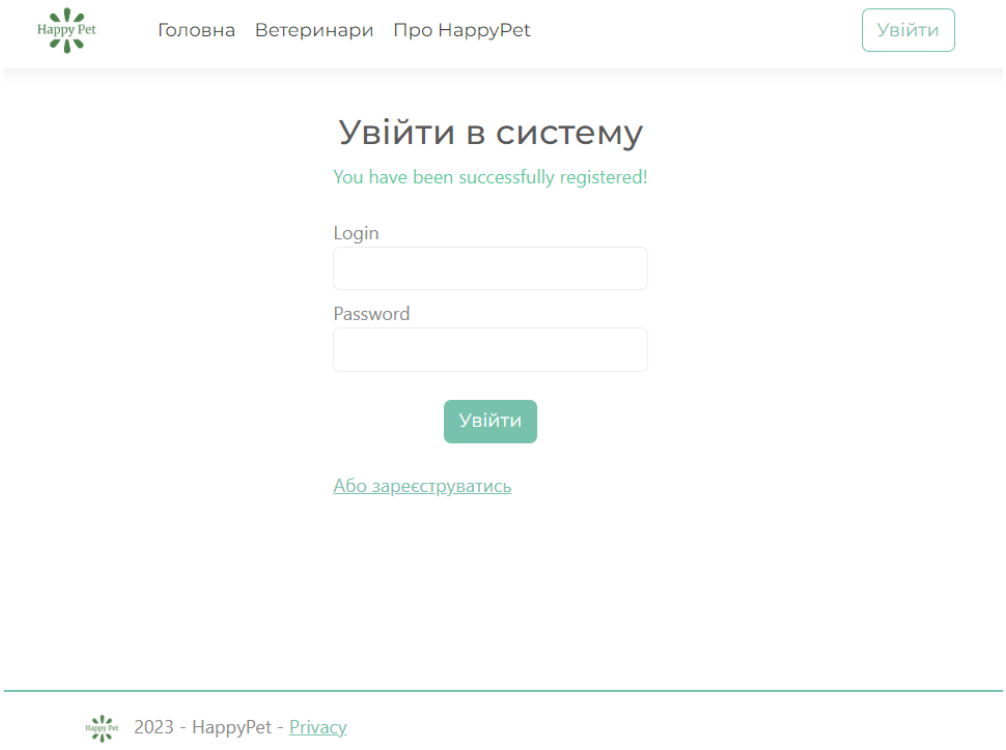


Рис. 4.1. Створення облікового запису у системі

Таблиця 4.2

Авторизація у системі

Передумова	Користувач є зареєстрованим у системі.
Сценарій дій	1. Користувач відкрив вебзастосунок “HappyPet”. 2. Користувач натискає на поле «Увійти». 3. Користувач вводить свій логін та пароль, з якими він реєструвався у системі. 4. Користувач натискає на клавішу «Увійти».
Очікуваний результат	Користувач успішно авторизувався у системі; система відкриває головну сторінку вебзастосунку.

Результат виконання тестового сценарію описаного у табл. 4.2 наведений на рис. 4.2.

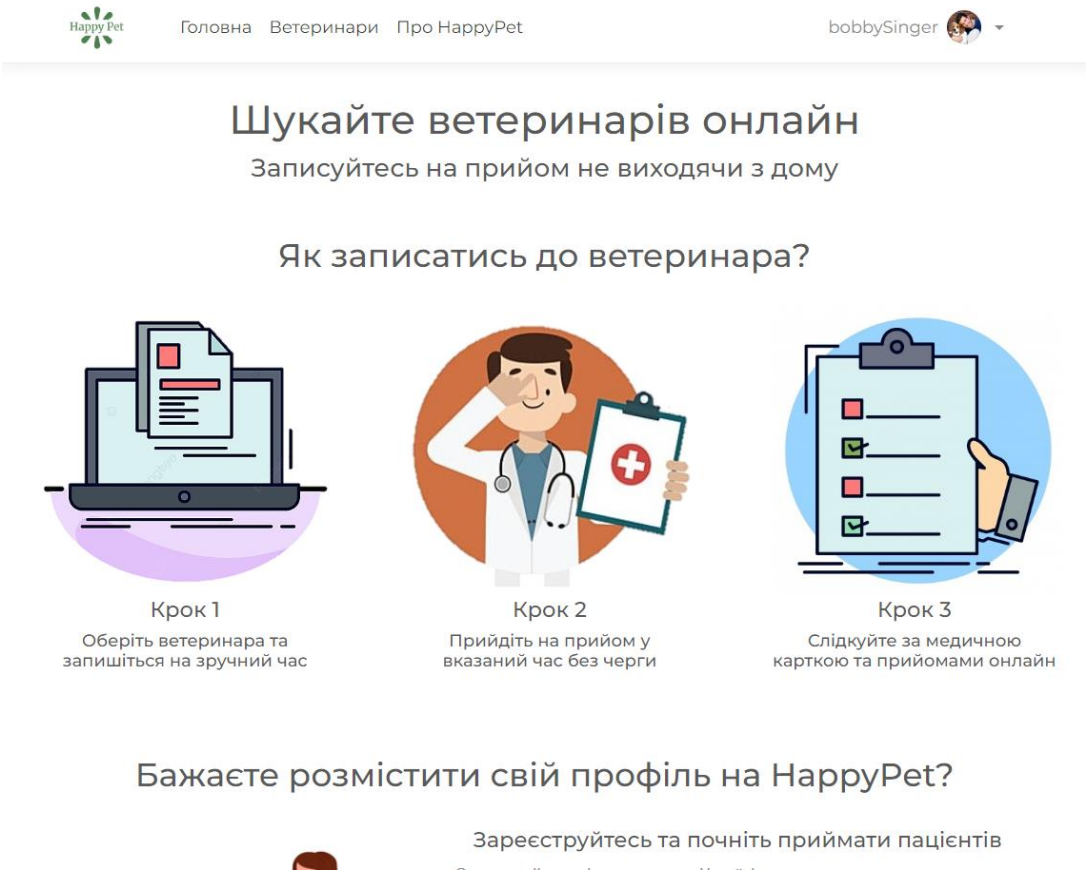


Рис. 4.2. Результат авторизації у системі

Таблиця 4.3

Пошук ветеринарів

Передумова	Користувач є авторизованим у системі.
Сценарій дій	1. Користувач натискає на поле «Ветеринари». 2. Користувач вводить у рядку пошуку прізвище ветеринара або у фільтрах обирає спеціальність чи регіон міста та натискає клавішу «Пошук». 3. Користувач фільтрує та сортує список знайдених ветеринарів.
Очікуваний результат	Користувач отримує відфільтровані та відсортовані результати у вигляді списку відповідно до свого запиту.

Результат виконання тестового сценарію описаного у табл. 4.3 наведений на рис. 4.3.

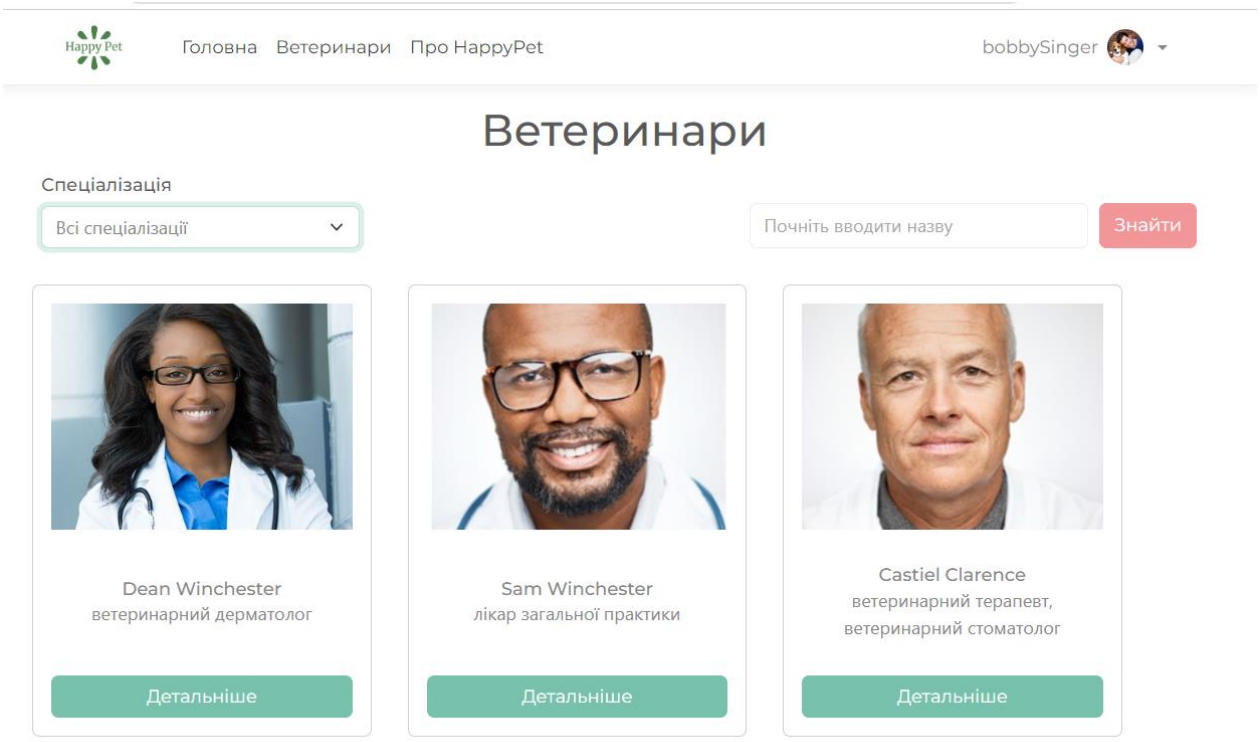


Рис. 4.3. Пошук ветеринарів

Таблиця 4.4

Перегляд інформації про обраного ветеринара

Передумова	Користувач є авторизованим у системі та здійснив пошук ветеринарів.
Сценарій дій	Користувач натиснув на ветеринара із запропонованого списку.
Очікуваний результат	Система відображає повну інформацію про ветеринара, список відгуків про нього та часові слоти для запису на прийом.

Результат виконання тестового сценарію описаного у табл. 4.4 наведений на рис. 4.4.

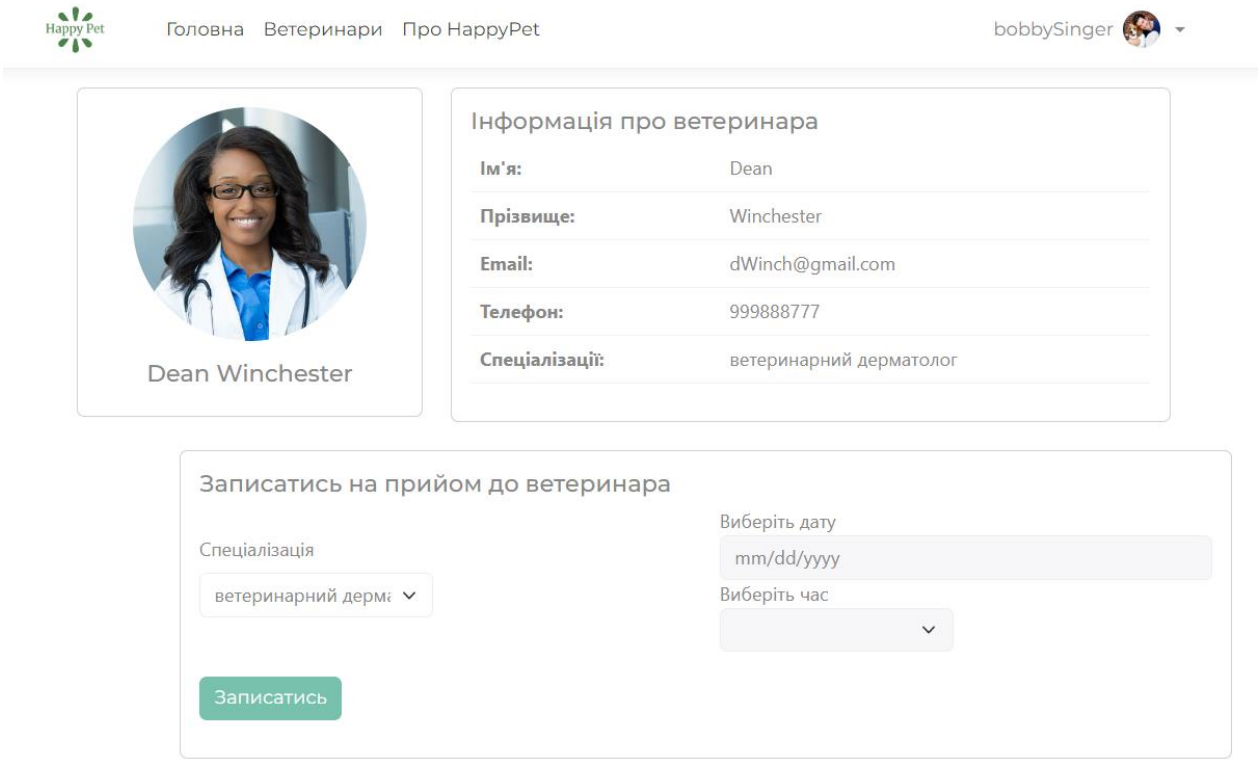


Рис. 4.4. Перегляд інформації про обраного ветеринара

Таблиця 4.5

Скасування прийому

Передумова	Користувач є авторизованим у системі.
Сценарій дій	1. Користувач натискає на поле «Прийоми». 2. Користувач обирає запланований прийом та натискає «Скасувати».
Очікуваний результат	Система відміняє запланований прийом; у ветеринара додається для запису часовий слот з скасованим прийомом.

Результат виконання тестового сценарію описаного у табл. 4.5 наведений на рис. 4.5.

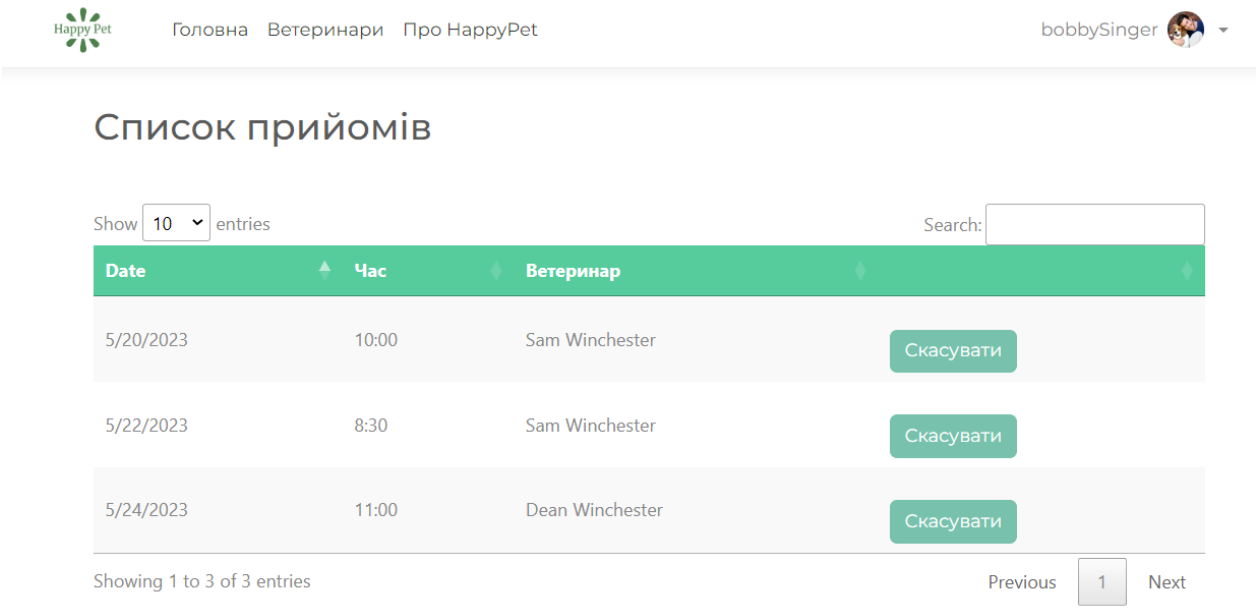


Рис. 4.5. Скасування запланованого прийому

Таблиця 4.6

Видалення користувача з системи

Передумова	Користувач є авторизованим у системі у ролі «Адміністратор».
Сценарій дій	1. Користувач натискає на поле «Керувати користувачами» та переходить на сторінку з усіма користувачами. 2. Користувач натискає на поле «Редагувати». 3. Користувач вибирає «Видалити». 4. Підтверджує видалення.
Очікуваний результат	Користувач видалений з системи.

Результат виконання тестового сценарію описаного у табл. 4.6 наведений на рис. 4.6.

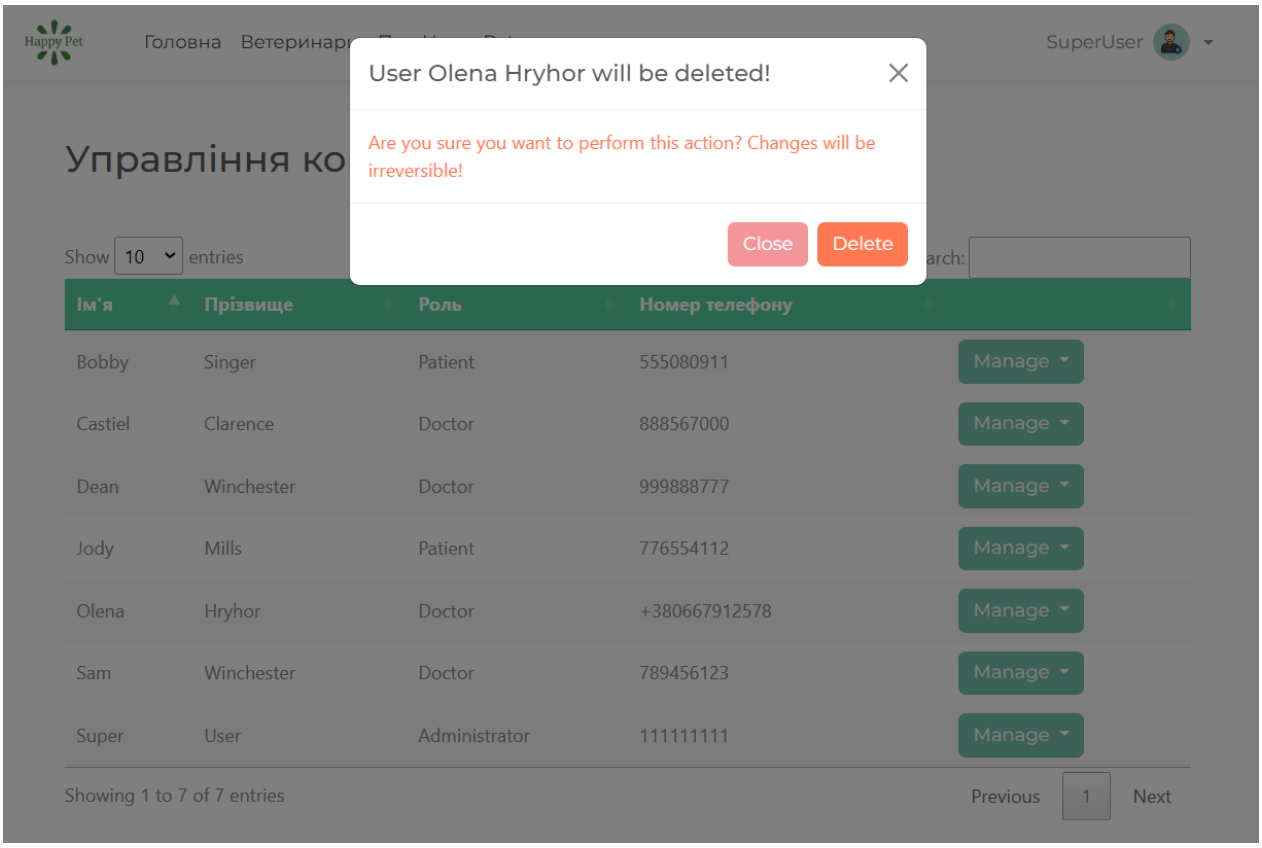


Рис. 4.6. Видалення користувача з системи

В результаті було знайдено знайдено та опрацьовано дві помилки.

Перша помилка стосувалась некоректного відображення підменю для авторизованого користувача (рис. 4.7).

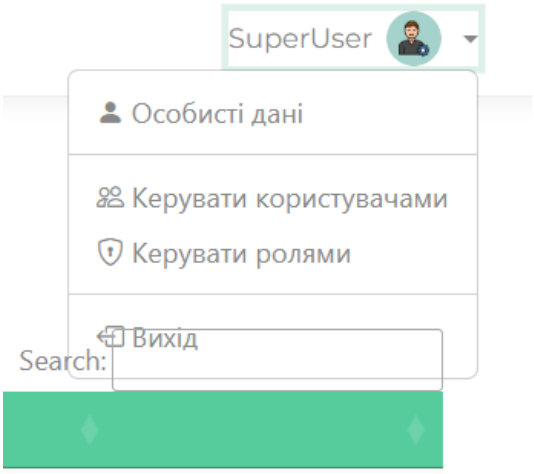


Рис. 4.7. Помилка відображення підменю користувача

Друга помилка – це не доступні для вибору вікно дати та випадаючий список для вибору часу (рис. 4.8).

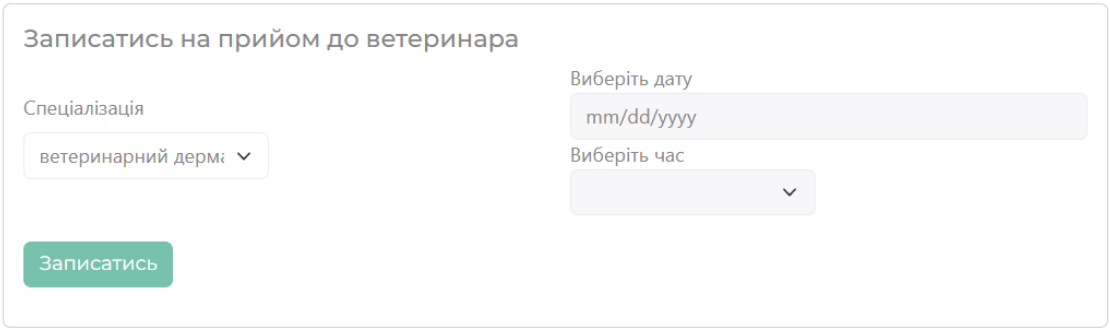


Рис. 4.8. Заблоковано елементи «Виберіть дату» та «Виберіть час»

Першу помилку було виправлено шляхом зміни стилізації підменю та головного наповнення вебсторінки. Друга помилка також стосувалась стилів,

тому її також було виправлено шляхом зміни стилю для компонентів «Виберіть дату» та «Виберіть час».

4.2. Порівняння розробки з існуючими аналогами

У підрозділі 1.2 першого розділу були досліджені схожі за функціональністю вебзастосунки та розглянути їх особливості, переваги та недоліки. Враховуючи вимоги, визначені в технічному завданні проєкту, були встановлені критерії оцінки для порівняння характеристик аналогічних рішень. Результат дослідження було представлено у вигляді порівняльної таблиці (табл. 1.1). Після аналізу цих критеріїв було виявлено, що жодне з аналогічних рішень не відповідає більше ніж 60% необхідних критеріїв.

Вебзастосунок “HappyPet” відповідає висунутим критеріям:

- Зручність пошуку ветеринара – вебзастосунок має можливість пошуку ветеринара за прізвищем, наявна фільтрація за спеціалізацією ветеринара, а також є можливість сортувати за популярністю, яка формується на основі відгуків про ветеринара.
- Можливість зареєструватись у системі – вебзастусонок підтримує можливість реєстрації у системі.
- Зручний та інтуїтивно зрозумілий інтерфейс – при розробці інтерфейсу користувача були враховані стандарти інтерфейсів сучасних вебзастосунків, зокрема, GUI було спроектовано з урахуванням усунення непотрібних елементів, які можуть відволікати увагу користувача.
- Наявність відгуків до ветеринарів – після завершення прийому власник тварини може залишити відгук.
- Керування прийомами – для обох ролей (власник тварини та ветеринар) є можливість запланувати та скасувати прийом.

- Онлайн запис на прийом до ветеринара – для авторизованого у системі користувача є можливість запису на прийом до ветеринара.

Після аналізу розробленого вебзастосунку можна зробити висновок, що розроблений вебзастосунок “HappyPet” виходить на передній план серед інших розглянутих рішень і стає кращим вибором для тих, хто шукає надійну та функціональну систему для своїх потреб у ветеринарній допомозі та догляді за тваринами.

4.3. Рекомендації для подальшого вдосконалення

Розроблений вебзастосунок є програмним продуктом, що задовольняє базові функціональні вимоги, які були висунуті. Тому запропоновано ряд покращень, які можна додати у наступній версії вебзастосунку “HappyPet”:

- можливість онлайн зв'язку ветеринара та власника тварини через чат або відео-зв'язок;
- можливість реєструватись та авторизуватись за допомогою Google акаунту, Facebook акаунту;
- можливість отримувати сповіщення про заплановані прийоми;
- підтримка мобільної версії вебзастосунку;
- наявність блогу у системі;
- додати медичну інформацію про тварин для власників тварин;
- можливість додавати фото матеріали до медичної інформації тварини;
- можливість зберігати медичну інформацію тварини у файл формату PDF.

4.4. Висновок

Для тестування розробленого вебзастосунку був обраний метод димового тестування. Для проведення димового тестування було складено

тестові сценарії та наведено результати виконання. У результаті тестування були виявлені помилки у графічному інтерфейсі: не правильно відображалось підменю користувача та були не доступні поля при записі на прийом – «Виберіть дату» та «Виберіть час». Обидві помилки були виправлені шляхом зміни стилів для цих компонентів.

Вебзастосунок “HappyPet” кращий порівняно з іншими розглянутими аналогічними рішеннями, оскільки він надає можливість зареєструватись у системі, здійснювати пошук ветеринара за прізвище та фільрацією по спеціалізації ветеринара, має можливість записуватись на прийом та скасовувати запланований прийом, крім того вебзастосунок має зручний та інтуїтивно зрозумілий інтерфейс.

Завершальним етапом розробки вебзастосунку є оцінка можливих покращень системи у майбутньому. В зв'язку з цим було розроблено ряд покращень, які можна внести у наступну версію вебзастосунку “HappyPet”, таких як розроблення мобільної версії вебзастосунку, додати можливість зберігати медичну інформацію тварини, можливість проводити онлайн-консультацію.

ВИСНОВКИ

Метою дипломного проєкту було створення вебзастосунку, який оптимізує комунікацію між власником тварини та ветеринаром: спрощує процес пошуку та запису до ветеринара; дозволяє переглядати на скасовувати заплановані прийоми, а також оцінювати якість наданих ветеринарних послуг.

Був проведений аналіз схожих за функціональністю вебзастосунків: “PetAdvisor”, “PetLive” та “Vetconsult.gladpet.org” та встановлено, що описані вебзастосунки не повністю задовільняють потреби користувачів. На основі цього аналізу були визначені функціональні вимоги до вебзастосунку “HappyPet”: можливість обирати ветеринара за спеціалізацією, записуватись онлайн на прийом до ветеринара, керувати власними прийомами.

Після аналізу різних програмних технологій, які можуть бути використані для розроблення вебзастосунку, зокрема фреймворків та систем керування базами даних (СКБД), та урахуванням переваг та недоліків кожної з них – було прийнято рішення розробляти клієнтську частину вебзастосунку з використанням HTML та CSS, базуючись на фреймворк Bootstrap. Щодо серверної частини вебзастосунку, було обрано використання фреймворку ASP.NET Core MVC та мови програмування C# з використанням СКБД MySQL Lite.

Для архітектури вебзастосунку був обраний шаблон проєктування MVC (Model-View-Controller), що дозволило чітко розділити логіку системи на модель, представлення та контролер. Цей підхід спрощує розробку, підтримку та тестування вебзастосунку, забезпечуючи гнучкість архітектури. Вебзастосунок включає 3 типи користувачів та неавторизованих користувачів. Для зберігання даних було створено 8 таблиць. Результатом є функціональний, безпечний та зручний вебзастосунок, який задовольняє

потреби користувачів та у повному обсязі відповідає висунутим до нього вимогам.

Розроблений вебзастосунок був протестований методом димового тестування, у результаті чого були знайдені помилки, які були вправлені у кінцевому вебзастосунку. У порівнянні з розглянутими аналогічними рішеннями, вебзастосунок “HappyPet” надає можливість зареєструватись у системі, здійснювати пошук ветеринара за прізвище та фільрацією по спеціалізації ветеринара, має можливість записуватись на прийом та скасовувати запланований прийом, крім того вебзастосунок має зручний та інтуїтивно зрозумілий інтерфейс, що робить його конкурентноспроможним. Також була проведена оцінка можливих покращень для майбутніх версій вебзастосунку. В результаті чого були виявлені подальші напрямки покращення вебзастосунку: можливість зберігати медичну інформацію тварини, розробка мобільної версії вебзастосунку, можливість проводити онлайн-консультацію .

СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

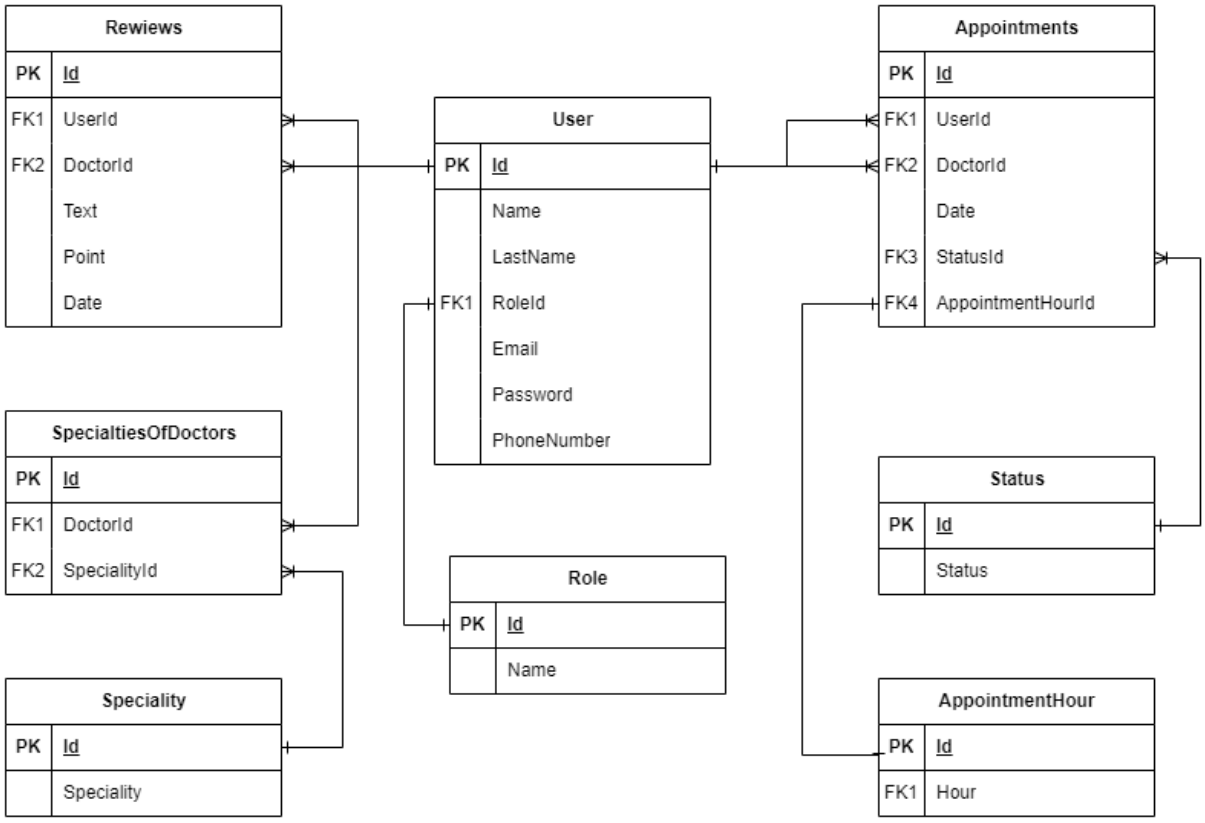
1. Countries With The Most Pet Cats Globally [Електронний ресурс]. — Режим доступу: https://www.worldatlas.com/articles/countries-with-the-most-pet-cats-globally.html.
2. Українці вважають за краще тримати вдома собак і котів. Опитування [Електронний ресурс]. — Режим доступу: https://www.ukrinform.ua/rubric-society/1464296-ukraiintsi_vvagayut_za_krashche_trimati_vdoma_sobak_i_kotiv___opituvannya_1802595.html.
3. Безкоштовна ветеринарна допомога онлайн [Електронний ресурс]. — Режим доступу: https://vetconsult.gladpet.org/.
4. Shostack, A. Threat Modeling: Designing for Security. Indianapolis: Wiley, 2014. 304 p.
5. de Kort W. Programming in C#: Exam Ref 70: Microsoft Press, 2018. 483 c.
6. Гріффітс Д. Head First Kotlin. Сан-Франциско: O'Reilly Media, 2019. 480 p.
7. Jemerov D., Isakova S. Kotlin in Action: Manning Publications, 2017. 360 p.
8. Haverbeke, M. Eloquent JavaScript: A Modern Introduction to Programming: No Starch Press, 2018. 472 p.
9. Javascript Engine & Performance Comparison (V8, Chakra, Chakra Core): [Електронний ресурс]. Режим доступу: https://developers.redhat.com/blog/2016/05/31/javascript-engine-performance-comparison-v8-charkra-chakra-core-2/.
10. Shaw P., Bootstrap 4 Succinctly: Syncfusion, 2018. 115 p.
11. Bootstrap (front-end framework) [Електронний ресурс]. — Режим доступу: https://en.wikipedia.org/wiki/Bootstrap_(front-end_framework).
12. Stefanov S. React Up and Running: O'Reilly Media, 2016. 298 p.

13. Ruebbelke L. AngularJS in Action: Manning Publications, 2015. 192 p.
14. Garcia-Molina H., Ullman J. D., Widom, J. Database Systems: The Complete Book: Pearson, 2008. 1296 p.
15. Petkovic D. Microsoft SQL Server 2019: A Beginner's Guide: McGraw Hill, 2020. 864 p.
16. Microsoft SQL Server [Електронний ресурс]. — Режим доступу: https://uk.wikipedia.org/wiki/Microsoft_SQL_Server.
17. What is PostgreSQL? Introduction, Advantages & Disadvantages [Електронний ресурс]. — Режим доступу: https://www.guru99.com/introduction-postgresql.html.
18. PostgreSQL [Електронний ресурс]. Режим доступу: https://uk.wikipedia.org/wiki/PostgreSQL.
19. Сучасні проблеми наукового забезпечення енергетики. Том 2: Матеріали XIX Міжнар. наук.-практ. конф. молод. вчених і студ., м. Київ: «Політехніка», 2021. Т. 2. 303 с.
20. Effective Software Test Automation: Developing an Automated Software Testing Tool / Li K., Wu M., Li T., Chen D. : CRC Press, 2017. 400 p.
21. What Is the Cost of a Requirement Error: [Електронний ресурс]. Режим доступу: https://www.stickyminds.com/article/what-cost-requirement-error.
22. Регламент Європейського Парламенту і Ради (ЄС) 2016/679 від 27 квітня 2016 року про захист фізичних осіб [Електронний ресурс]. — Режим доступу: https://zakon.rada.gov.ua/laws/show/984_008-16#Text.

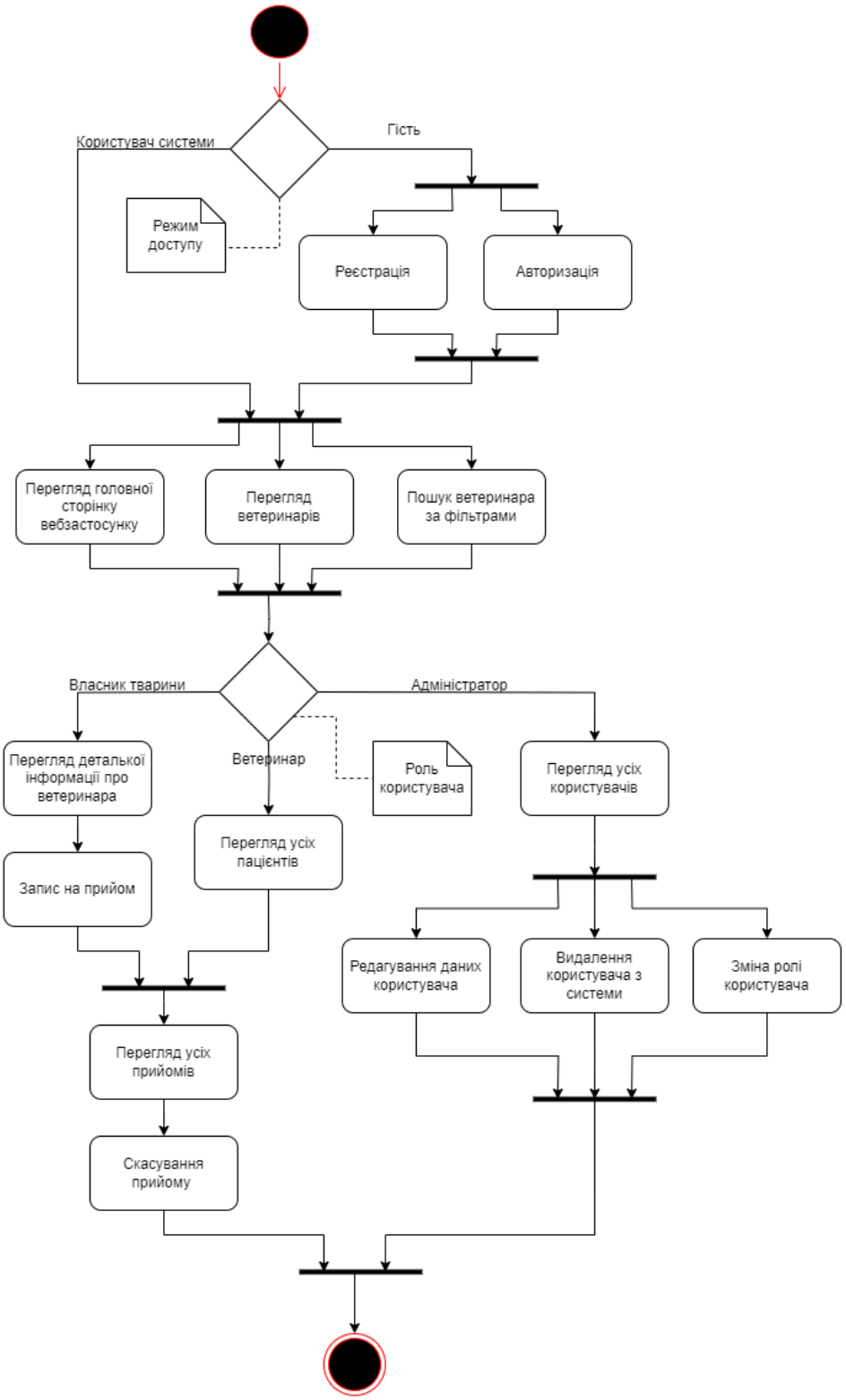
ДОДАТКИ

Додаток 1

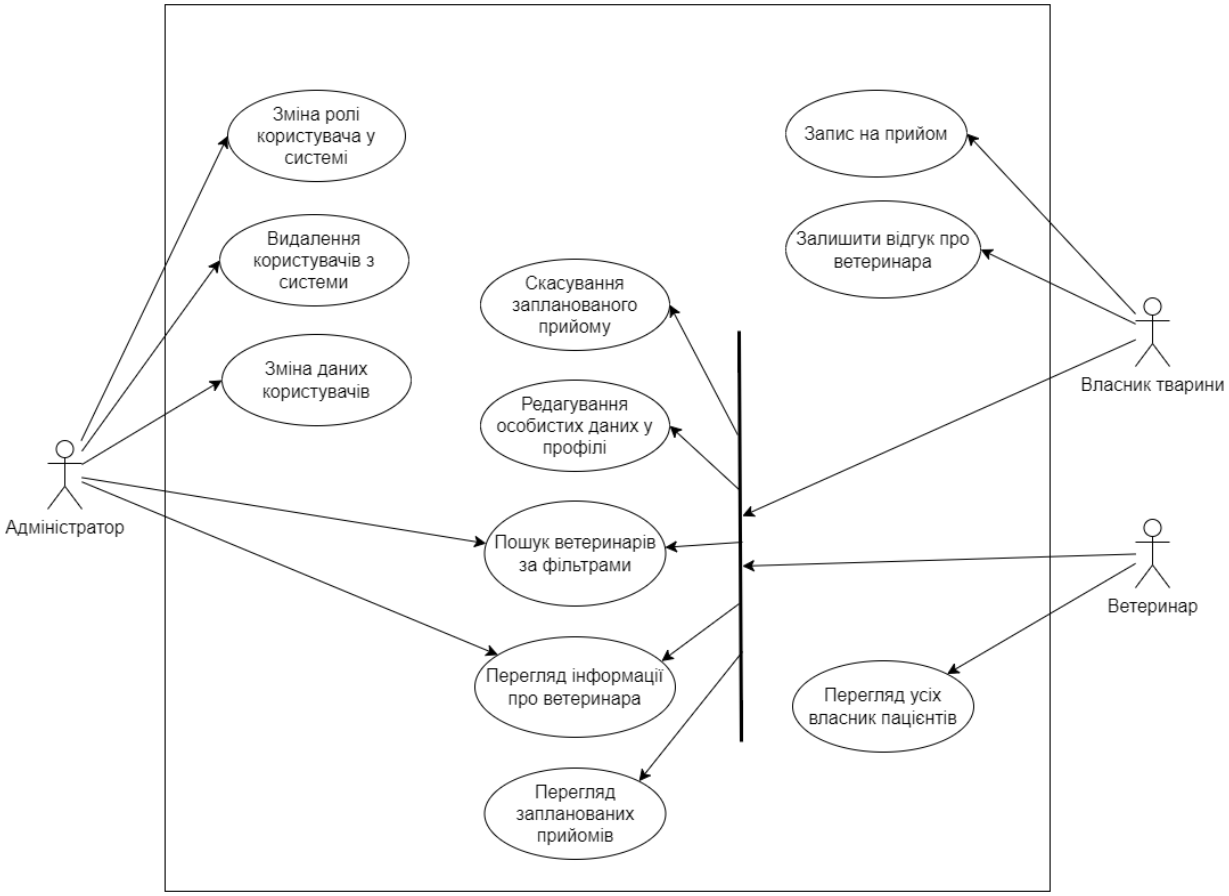
Копії графічних матеріалів



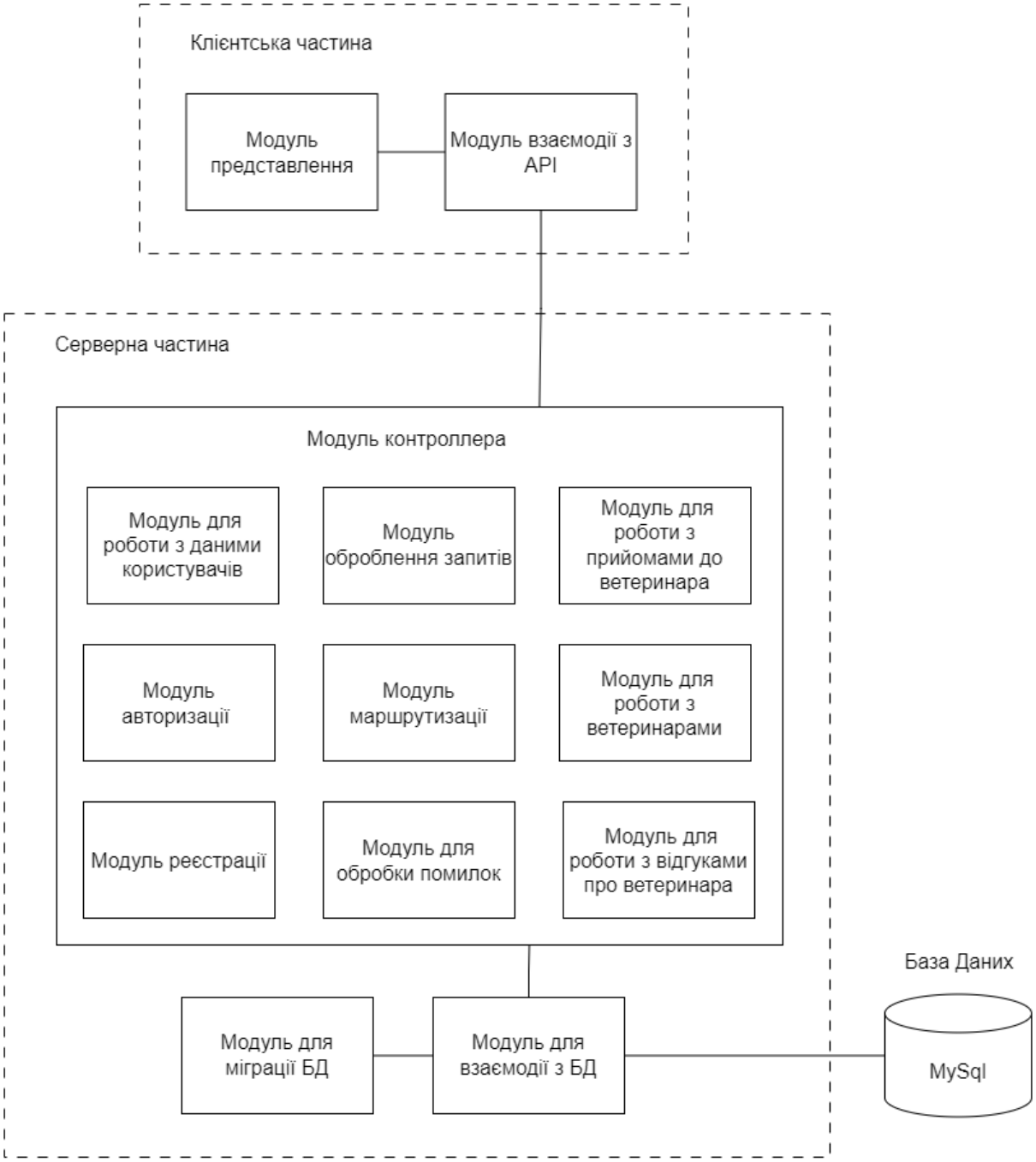
ДП.045440-07-99
Вебзастосунок для управління наданням ветеринарних послуг. Структура таблиць БД. ERD діаграма



ДП.045440-06-99
Вебзастосунок для управління наданням ветеринарних послуг. Алгоритм роботи вебзастосунку. Діаграма діяльності



Діаграма використання вебзастосунку
Григор Олена, група КП-93



Структура вебзастосунку “HappyPet”
Григор Олена, група КП-93

Додаток 2

Лістинг програми

UserController.cs

```
namespace HappyPet.Controllers;

[Authorize]
public class UserController : Controller
{
    private readonly UserManager<User> _userManager;
    private readonly RoleManager<Role> _roleManager;
    private readonly IMapper _mapper;
    private readonly IHostingEnvironment _hostEnvironment;

    public UserController(UserManager<User> userManager, RoleManager<Role>
    roleManager, IMapper mapper, IHostingEnvironment hostEnvironment)
    {
        _userManager = userManager;
        _roleManager = roleManager;
        _mapper = mapper;
        _hostEnvironment = hostEnvironment;
    }

    public IActionResult Index()
    {
        return View();
    }

    [HttpGet]
    [Authorize(Roles = "Administrator,Doctor,Patient")]
    public async Task<IActionResult> Profile()
    {
        var currentUser = await _userManager.GetUserAsync(HttpContext.User);

        var user = _mapper.Map<UserProfileViewModel>(currentUser);
        return View(user);
    }

    [HttpGet]
    [Authorize(Roles = "Administrator,Patient")]
    public async Task<IActionResult> Doctors()
    {
        var users = await _userManager.GetUsersInRoleAsync("Doctor");
        var doctors = _mapper.Map<IEnumerable<DoctorViewModel>>(users)
        return View(doctors);
    }

    [HttpGet]
    [Authorize(Roles = "Administrator,Doctor")]
    public async Task<IActionResult> Patients()
    {
        var currentUser = await _userManager.GetUserAsync(HttpContext.User);

        var     patients     =     currentUser.AppointmentsDoc.Select(a     =>
        a.Patient).Distinct();
        var                            patientsVM                            =
        mapper.Map<IEnumerable<PatientViewModel>>(patients);

        _hostEnvironment.WebRootPath                                          =
        Path.Combine(Directory.GetCurrentDirectory(), "wwwroot");
```

```
      foreach (var patient in patientsVM)
      {
          string    path    =    string.Format("{0}/img/profiles/{1}.jpg",
          _hostEnvironment.WebRootPath, patient.UserName);

          patient.PhotoUploaded  =  System.IO.File.Exists(path)  ?  true  :
          false;
      }
      return View(patientsVM);
}

[HttpGet]
[Authorize(Roles = "Administrator")]
public async Task<IActionResult> Manage()
{
    var usersDb = await _userManager.Users.ToListAsync();
    var users = _mapper.Map<IEnumerable<UserViewModel>>(usersDb);
    return View(users);
}

[HttpGet]
[Authorize(Roles = "Administrator, Doctor")]
public async Task<IActionResult> Edit(string userId)
{
    var userDb = await _userManager.FindByIdAsync(userId);
    var user = _mapper.Map<UserEditViewModel>(userDb);
    return View(user);
}

[HttpPost]
[Authorize(Roles = "Administrator, Doctor")]
public async Task<IActionResult> Edit(UserEditViewModel userViewModel)
{
    if (ModelState.IsValid)
    {
        var                 userDb                 =                 await
        _userManager.FindByIdAsync(userViewModel.UserId);
        if (userDb != null)
        {
            userDb.UserName = userViewModel.UserName;
            userDb.FirstName = userViewModel.FirstName;
            userDb.LastName = userViewModel.LastName;
            userDb.PhoneNumber = userViewModel.PhoneNumber;
            var result = await _userManager.UpdateAsync(userDb);

            if (result.Succeeded)
                return RedirectToAction("Manage");
            else
                foreach (var error in result.Errors)
                   ModelState.AddModelError(error.Code,
                      error.Description);
        }
    }

    return View(userViewModel);
}

[HttpGet]
[Authorize(Roles = "Administrator")]
```

```
public async Task<IActionResult> Delete(string userId)
{
    var user = await _userManager.FindByIdAsync(userId);
    if (await _userManager.IsInRoleAsync(user, "Administrator"))
        return RedirectToAction("Manage");

    await _userManager.DeleteAsync(user);
    return RedirectToAction("Manage");
}

[HttpGet]
[Authorize(Roles = "Administrator")]
public async Task<IActionResult> ManageRoles()
{
    var usersDb = await _userManager.Users.ToListAsync();
    var model = new UserEditRoleViewModel();
    model.Users = _mapper.Map<IEnumerable<UserViewModel>>(usersDb);
    var roles = await _roleManager.Roles.ToListAsync();
    var rolesVM = _mapper.Map<IEnumerable<RoleViewModel>>(roles);
    model.Roles = new SelectList(rolesVM, "RoleId", "Name");
    return View(model);
}

[HttpPost]
[Authorize(Roles = "Administrator")]
public async Task<IActionResult> ManageRoles(string userId, string roleId)
{
    var usersDb = await _userManager.Users.ToListAsync();
    var      x      =      await      _userManager.Users.Include(u      =>
    u.UserRoles).ThenInclude(ur => ur.Role).ToListAsync();
    var model = new UserEditRoleViewModel();
    model.Users = _mapper.Map<IEnumerable<UserViewModel>>(usersDb);
    var roles = await _roleManager.Roles.ToListAsync();
    var rolesVM = _mapper.Map<IEnumerable<RoleViewModel>>(roles);
    model.Roles = new SelectList(rolesVM, "RoleId", "Name");
    if (ModelState.IsValid)
    {
        var user = await _userManager.FindByIdAsync(userId);
        if (user is null)
        {
            ModelState.AddModelError(string.Empty, "User not found");
            return View(model);
        }

        var role = await _roleManager.FindByIdAsync(roleId);
        if (role is null)
        {
            ModelState.AddModelError(string.Empty, "Role not found");
            return View(model);
        }

        var userRoles = await _userManager.GetRolesAsync(user);
        var   result   =   await   _userManager.RemoveFromRolesAsync(user,
         userRoles.ToArray());

        if (!result.Succeeded)
        {
            foreach (var error in result.Errors)
                ModelState.AddModelError(error.Code, error.Description);
```

```
                return View(model);
            }
            result = await _userManager.AddToRoleAsync(user, role.Name);
            if (!result.Succeeded)
            {
                foreach (var error in result.Errors)
                    ModelState.AddModelError(error.Code, error.Description);
                return View(model);
            }

            var userVM = _mapper.Map<UserViewModel>(user);
            model.Users = model.Users.Select(u => u.UserId.Equals(user.Id) ?
            userVM : u);

            TempData["roleChanged"] = true;
        }

        return View(model);
    }
}
```

VeterinariansController.cs

```
   public class VeterinariansController : Controller
   {
       private readonly IUnitOfWork _unitOfWork;
       private readonly ILogger<HomeController> _logger;
       private readonly UserManager<User> _userManager;
       private readonly IMapper _mapper;

       public VeterinariansController(IUnitOfWork unitOfWork,
       UserManager<User> userManager, IMapper mapper)
       {
           _unitOfWork = unitOfWork;
           _userManager = userManager;
           _mapper = mapper;
       }


       public async Task<IActionResult> Index()
       {
           var users = await _userManager.GetUsersInRoleAsync("Doctor");
           var doctors = _mapper.Map<IEnumerable<DoctorViewModel>>(users);
           return View(doctors);
       }

       [HttpGet]
       public async Task<IActionResult> Details(string userId)
       {
           var userDb = await _userManager.FindByIdAsync(userId);
           var viewModel = new DetailsViewModel()
           {
               DoctorId = userDb.Id,
               FirstName = userDb.FirstName,
               LastName = userDb.LastName,
               Email = userDb.Email,
               PhoneNumber = userDb.PhoneNumber
           };
```

```
    viewModel.Specialties = new SelectList(userDb.UserSpecialties,
    "SpecialtyId", "Specialty.Name");
    return View(viewModel);
}

[HttpPost]
[Authorize(Roles = "Administrator,Patient")]
public async Task<IActionResult> Details(DetailsViewModel viewModel)
{
    if (ModelState.IsValid)
    {
        var doctor = await
        _userManager.FindByIdAsync(viewModel.DoctorId);
        if (doctor is null)
        {
            ModelState.AddModelError("DoctorId", "Doctor 404");
            return View(viewModel);
        }

        if (doctor.AppointmentsDoc.Any(a => a.Date == viewModel.Date
        && a.AppointmentHourId == viewModel.HourId))
        {
            ModelState.AddModelError("HourId", "Цей час уже зайнятий");
            return View(viewModel);
        }

        var specialty = await
        _unitOfWork.Specialties.GetById(viewModel.SpecialtyId);
        if (specialty is null)
        {
            ModelState.AddModelError("SpecialtyId", "Specialty 404");
            return View(viewModel);
        }

       var hour = await _unitOfWork.AppointmentHours.GetById((long)
        viewModel.HourId);
        if (hour is null)
        {
            ModelState.AddModelError("HourId", "Hour 404");
            return View(viewModel);
        }

        var newAppointment = new Appointment()
        {
            AppointmentHourId = (long) viewModel.HourId,
            CreateDate = DateTime.Now,
            Date = (DateTime) viewModel.Date,
            Hour = hour,
            DoctorId = viewModel.DoctorId,
            Doctor = doctor
        };


        newAppointment.Patient = await
        _userManager.GetUserAsync(HttpContext.User);

        _unitOfWork.Appointments.Insert(newAppointment);
        if (await _unitOfWork.SaveChangesAsync() < 1)
        {
```

```
            ModelState.AddModelError("", "Error while creating
              appointment");
                  return View(viewModel);
              }

              viewModel.Doctor = _mapper.Map<DoctorViewModel>(doctor);
               viewModel.Patient                                          =
               _mapper.Map<PatientViewModel>(newAppointment.Patient);
              viewModel.Specialty                                         =
             _mapper.Map<SpecialtyViewModel>(specialty);
              viewModel.Hour = _mapper.Map<AvailableHourViewModel>(hour);
              return View("SuccessCreateAppointment", viewModel);
          }

          return View(viewModel);
      }

     [ResponseCache(Duration = 0, Location = ResponseCacheLocation.None,
      NoStore = true)]
      public IActionResult Error()
      {
          return View(new ErrorViewModel {RequestId = Activity.Current?.Id ??
         HttpContext.TraceIdentifier});
      }
  }
```

AppointmentController.cs

```
  public class AppointmentController : Controller
  {
      private readonly IUnitOfWork _unitOfWork;
      private readonly IMapper _mapper;
      private readonly UserManager<User> _userManager;

       public AppointmentController(IUnitOfWork unitOfWork, UserManager<User>
      userManager, IMapper mapper)
      {
          _unitOfWork = unitOfWork;
          _userManager = userManager;
          _mapper = mapper;
      }

      [HttpGet]
      public async Task<IActionResult> Index()
      {
         var                currentUser                =                await
          _userManager.GetUserAsync(HttpContext.User);
           var              userAppointments              =              await
        _userManager.IsInRoleAsync(currentUser, "Doctor")
              ? currentUser.AppointmentsDoc
              : currentUser.AppointmentsPat;

         var                           appointments                          =
          _mapper.Map<IEnumerable<AppointmentViewModel>>(userAppointments);
          appointments = appointments.OrderByDescending(a => a.Date);
          return View(appointments);
      }
```

```
[HttpGet]
public async Task<IActionResult> Create()
{
    var specialties = await _unitOfWork.Specialties.GetAll();
    var                          specialtiesVM                          =
    _mapper.Map<IEnumerable<SpecialtyViewModel>>(specialties);
    var model = new AppointmentCreateViewModel();
    model.Specialties  =  new  SelectList(specialtiesVM,  "SpecialtyId",
    "Name");
    return View(model);
}

[HttpPost]
public async Task<IActionResult> Create(
     [Bind("SpecialtyId",      "DoctorId",      "Date",      "HourId")]
     AppointmentCreateViewModel model)
{
    var specialties = await _unitOfWork.Specialties.GetAll();
     var                          specialtiesVM                          =
    _mapper.Map<IEnumerable<SpecialtyViewModel>>(specialties);
   model.Specialties  =  new  SelectList(specialtiesVM,  "SpecialtyId",
    "Name");

    if (ModelState.IsValid)
    {
        var doctor = await _userManager.FindByIdAsync(model.DoctorId);
        if (doctor is null)
        {
            ModelState.AddModelError("DoctorId", "Doctor 404");
            return View(model);
        }

       if  (doctor.AppointmentsDoc.Any(a  =>  a.Date  ==  model.Date  &&
        a.AppointmentHourId == model.HourId))
        {
            ModelState.AddModelError("HourId", "This hour is taken");
            return View(model);
        }

        var               specialty               =               await
       _unitOfWork.Specialties.GetById(model.SpecialtyId);
        if (specialty is null)
        {
            ModelState.AddModelError("SpecialtyId", "Specialty 404");
            return View(model);
        }

       var  hour  =  await  _unitOfWork.AppointmentHours.GetById((long)
        model.HourId);
        if (hour is null)
        {
            ModelState.AddModelError("HourId", "Hour 404");
            return View(model);
        }

        var newAppointment = _mapper.Map<Appointment>(model);
        newAppointment.Patient                  =                  await
        _userManager.GetUserAsync(HttpContext.User);
        _unitOfWork.Appointments.Insert(newAppointment);
```

```
            if (await _unitOfWork.SaveChangesAsync() < 1)
            {
                ModelState.AddModelError("", "Error while creating
            appointment");
                return View(model);
            }

            model.Doctor = _mapper.Map<DoctorViewModel>(doctor);
            model.Patient =
            _mapper.Map<PatientViewModel>(newAppointment.Patient);
            model.Specialty = _mapper.Map<SpecialtyViewModel>(specialty);
            model.Hour = _mapper.Map<AvailableHourViewModel>(hour);
            return View("AppointmentCreated", model);
        }
        return View(model);
    }

    public async Task<JsonResult> AllSpecialties()
    {
        var specialties = await _unitOfWork.Specialties.GetAll();
        var specialtiesVM =
        _mapper.Map<IEnumerable<SpecialtyViewModel>>(specialties);
        return Json(specialtiesVM);
    }

    public async Task<JsonResult> AllDoctorsInSpecialty(int specialtyId)
    {
        var doctors = await _userManager.Users.Where(u =>
        u.UserSpecialties.Any(us => us.SpecialtyId == specialtyId))
            .ToListAsync();
        var doctorsVM = _mapper.Map<IEnumerable<DoctorViewModel>>(doctors);
        return Json(doctorsVM);
    }

    public async Task<JsonResult> HoursAvailability(string doctorId,
    DateTime date)
    {
        var appointmentHours = await _unitOfWork.AppointmentHours.GetAll();

        var appointmentHoursVM =
        _mapper.Map<IEnumerable<AvailableHourViewModel>>(appointmentHours);

        var doctorAppointments = await _unitOfWork.Appointments.FindAsync(a
        => a.DoctorId.Equals(doctorId) && a.Date.Date.CompareTo(date.Date)
        == 0);

        var doctorAppointmentsHoursId = doctorAppointments.Select(a =>
        a.AppointmentHourId);

        for (int i = 0; i < appointmentHoursVM.Count(); i++)
            if
               (!doctorAppointmentsHoursId.Contains(appointmentHoursVM.Elem
               entAt(i).HourId))
                appointmentHoursVM.ElementAt(i).Available = true;
        return Json(appointmentHoursVM);
    }
}
```

AccountController.cs

```
[Authorize]
public class AccountController : Controller
{
    private readonly IUnitOfWork _unitOfWork;
    private readonly UserManager<User> _userManager;
    private readonly SignInManager<User> _signInManager;
    private readonly IMapper _mapper;

    public AccountController(IUnitOfWork unitOfWork, UserManager<User>
    userManager, SignInManager<User> signInManager, IMapper mapper)
    {
        _unitOfWork = unitOfWork;
        _userManager = userManager;
        _signInManager = signInManager;
        _mapper = mapper;
    }

    [HttpGet]
    public IActionResult Index()
    {
        return View();
    }

    [HttpGet]
    [AllowAnonymous]
    public IActionResult Login()
    {
        if (User.Identity.IsAuthenticated)
            return RedirectToAction("Index", "Home");
        var model = new UserLoginViewModel();
        if (TempData.ContainsKey("RegisterSuccess"))
                model.RegisterSuccess                                    =
         Convert.ToBoolean(TempData["RegisterSuccess"]);
        return View(model);
    }

    [HttpPost]
    [AllowAnonymous]
    public async Task<IActionResult> Login(UserLoginViewModel user)
    {
        if (ModelState.IsValid)
        {
            var               result               =               await
             _signInManager.PasswordSignInAsync(user.UserName,
             user.Password, false, false);
            if (result.Succeeded)
                return RedirectToAction("Index", "Home");

            ModelState.AddModelError("LoginError",      "Invalid      login
             attempt");
        }
        return View(user);
    }
```

```
[HttpGet]
[AllowAnonymous]
public async Task<IActionResult> Register()
{
    if (User.Identity.IsAuthenticated)
        return RedirectToAction("Index", "Home");
    var specialties = await _unitOfWork.Specialties.GetAll();

    var specialtiesVM =
    mapper.Map<IEnumerable<SpecialtyViewModel>>(specialties)

    ViewBag.SpecialtiesNames     =     new     SelectList(specialtiesVM,
    "SpecialtyId", "Name");
    return View();
}

[HttpPost]
[AllowAnonymous]
public async Task<IActionResult> Register(UserRegisterViewModel model)
{
    if (ModelState.IsValid)
    {
        var newUser = _mapper.Map<User>(model);
        var    result    =    await    _userManager.CreateAsync(newUser,
        model.Password);

        if (!result.Succeeded)
        {
            foreach (var error in result.Errors)

                ModelState.AddModelError(string.Empty,
                error.Description);

            return View(model);
        }

        if (!model.IsVeterinarian)
            await _userManager.AddToRoleAsync(newUser, "Patient");
        else
        {
            await _userManager.AddToRoleAsync(newUser, "Doctor");
            foreach (var specialtyId in model.SelectedSpecialties)
            {
                if(long.TryParse(specialtyId, out var id))

                    _unitOfWork.UserSpecialties.Insert(new
                    UserSpecialty(){UserId = newUser.Id, SpecialtyId =
                    id});
            }

            await _unitOfWork.SaveChangesAsync();

        }

        TempData["registerSuccess"] = true;
        return RedirectToAction("Login", "Account");
    }
    return View(model);
}
```

Додаток 3

Копія презентації

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

КАФЕДРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ КОМП’ЮТЕРНИХ СИСТЕМ

ВЕБЗАСТОСУНОК ДЛЯ УПРАВЛІННЯ НАДАННЯМ ВЕТЕРИНАРНИХ ПОСЛУГ

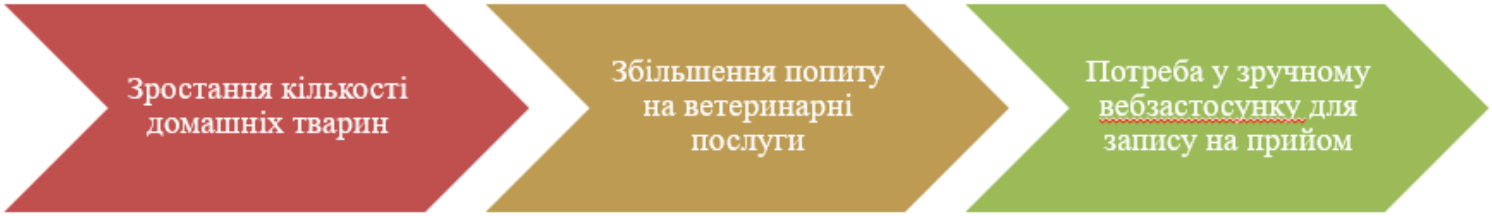
Виконала: Григор Олена Миколаївна

Керівник: доц. кафедри ПЗКС, к.т.н., доц. Люшенко Леся Анатоліївна

Київ – 2023

АКТУАЛЬНІСТЬ

За даними соціологічних досліджень компанії “Washington Post Group”, Україна знаходиться в десятці країн за кількістю домашніх тварин на громадянина.



Happy Pet

ПОСТАНОВКА ЗАДАЧІ

Мета проєкту: розробити вебзастосунок, який оптимізує комунікацію між власником тварини та ветеринаром: спрощує процес пошуку та запису до ветеринара; дозволяє переглядати на скасовувати заплановані прийоми, а також оцінювати якість наданих ветеринарних послуг.

Завдання:

1. Проаналізувати предметну область, переваги та недоліки існуючих програмних рішень
2. Визначити вимоги до розроблюваного ПЗ.
3. Розробити вебзастосунок з можливостю записуватись онлайн на прийом до ветеринара.
4. Протестувати коректність роботи вебзастосунку та проаналізувати його на відповідність до вимог.
5. Визначити напрямки для подальшого розвитку.

ІСНУЮЧІ РІШЕННЯ

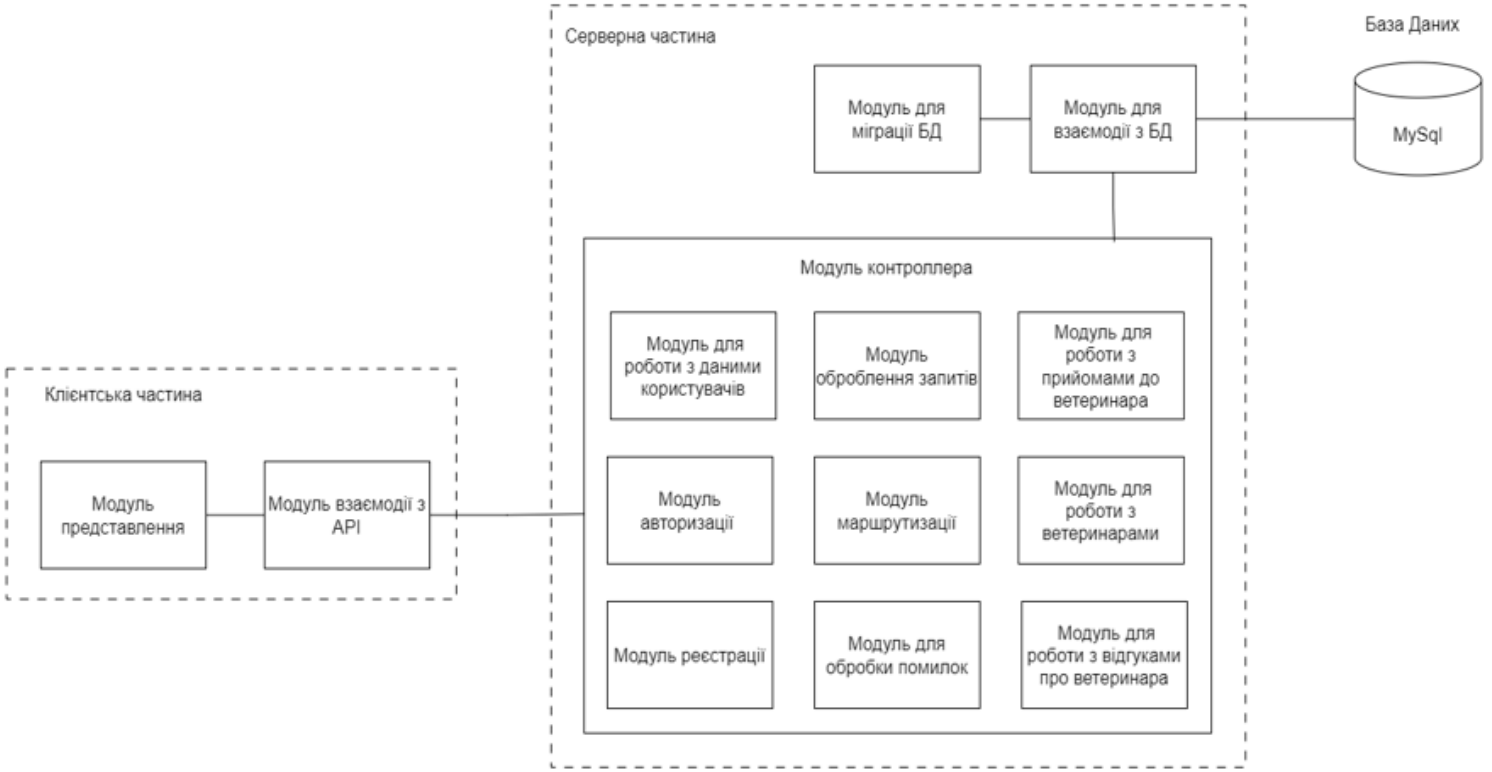
PetLive

Vetconsult.gladpet

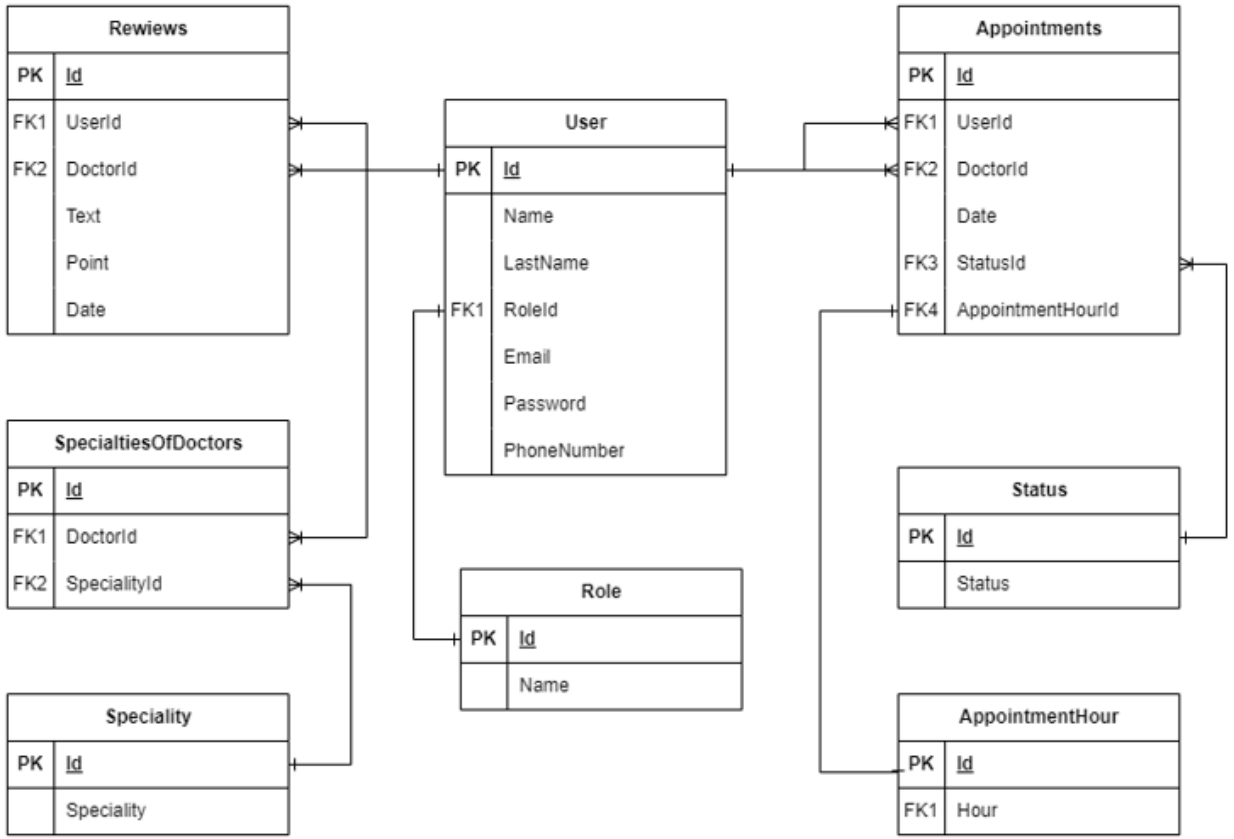
PetAdvisor

ВИБІР ТЕХНОЛОГІЙ РОЗРОБЛЕННЯ

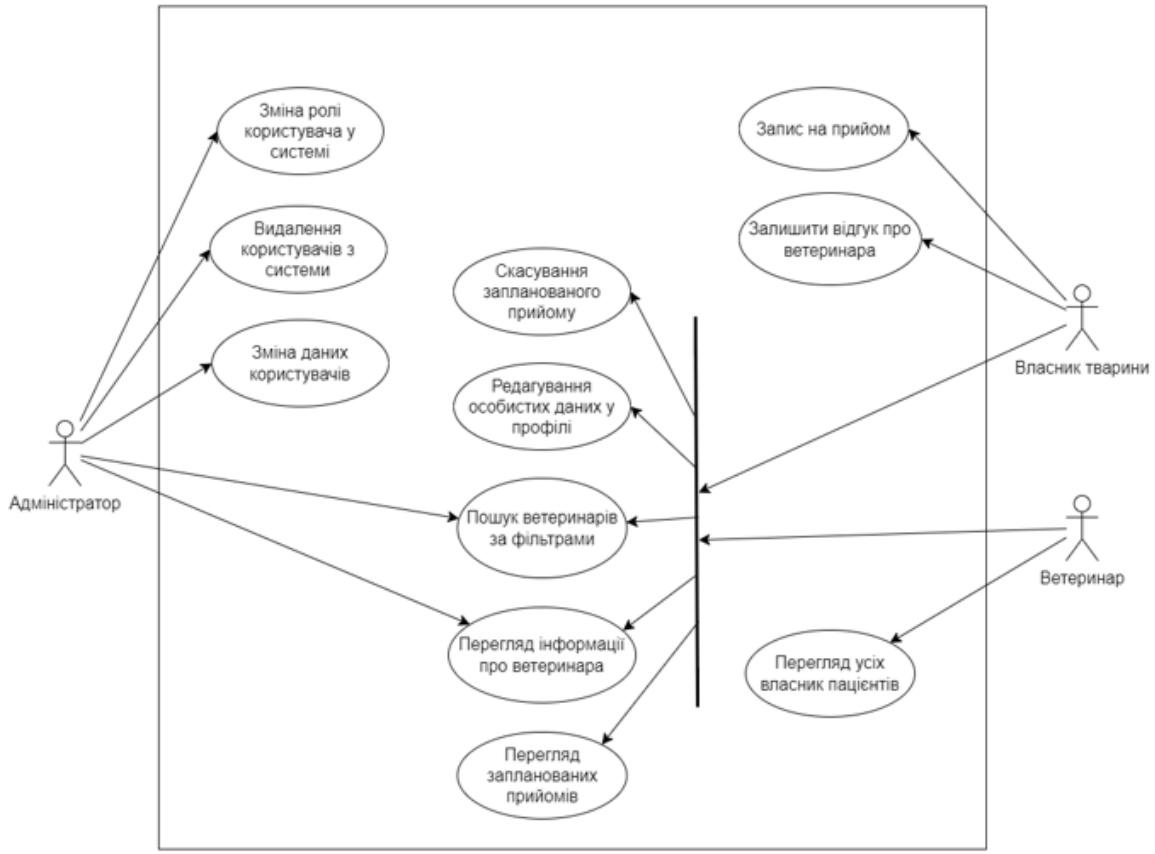
АРХІТЕКТУРА СИСТЕМИ



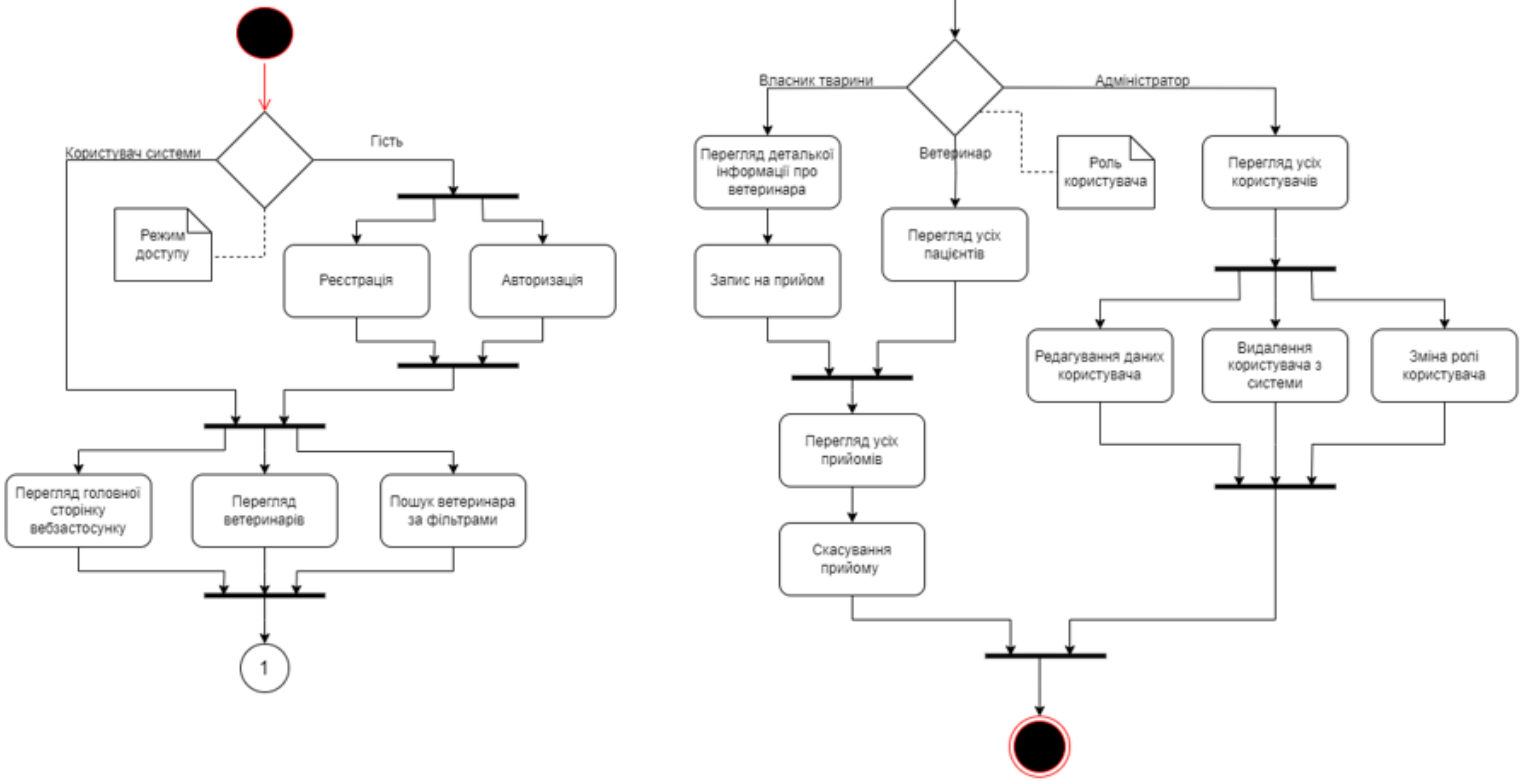
СТРУКТУРА БД



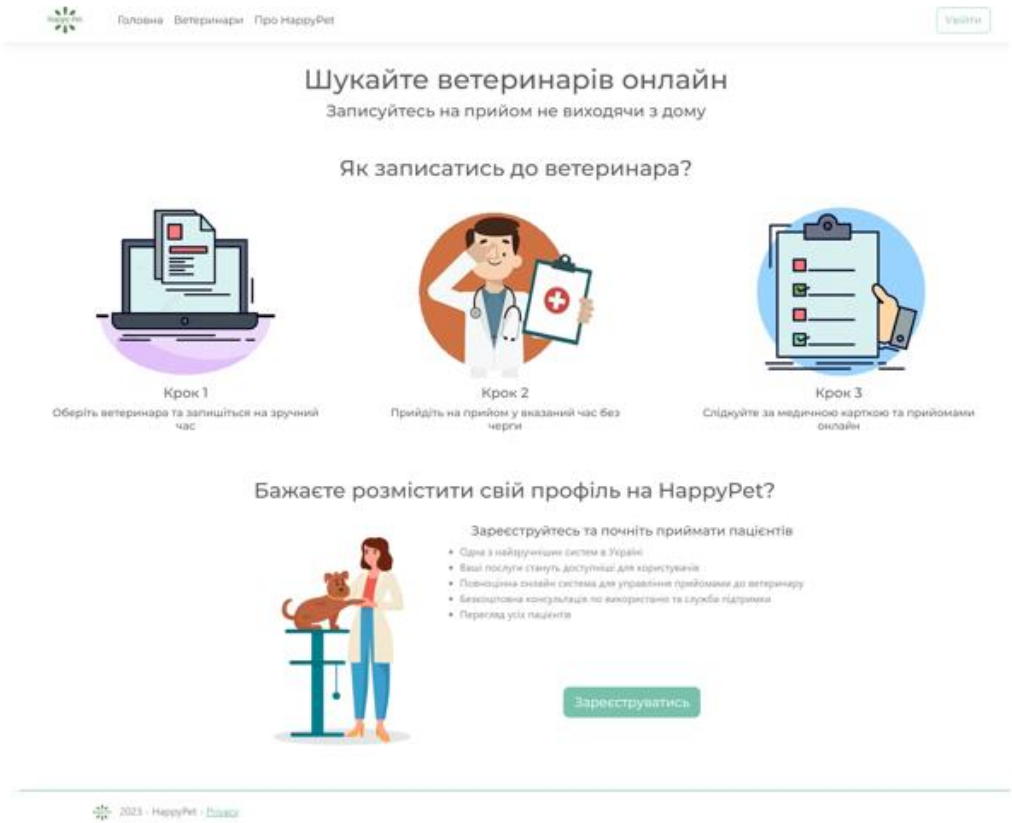
ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ



ДІАГРАМА ДІЯЛЬНОСТІ. АЛГОРИТМ РОБОТИ ПРОГРАМИ

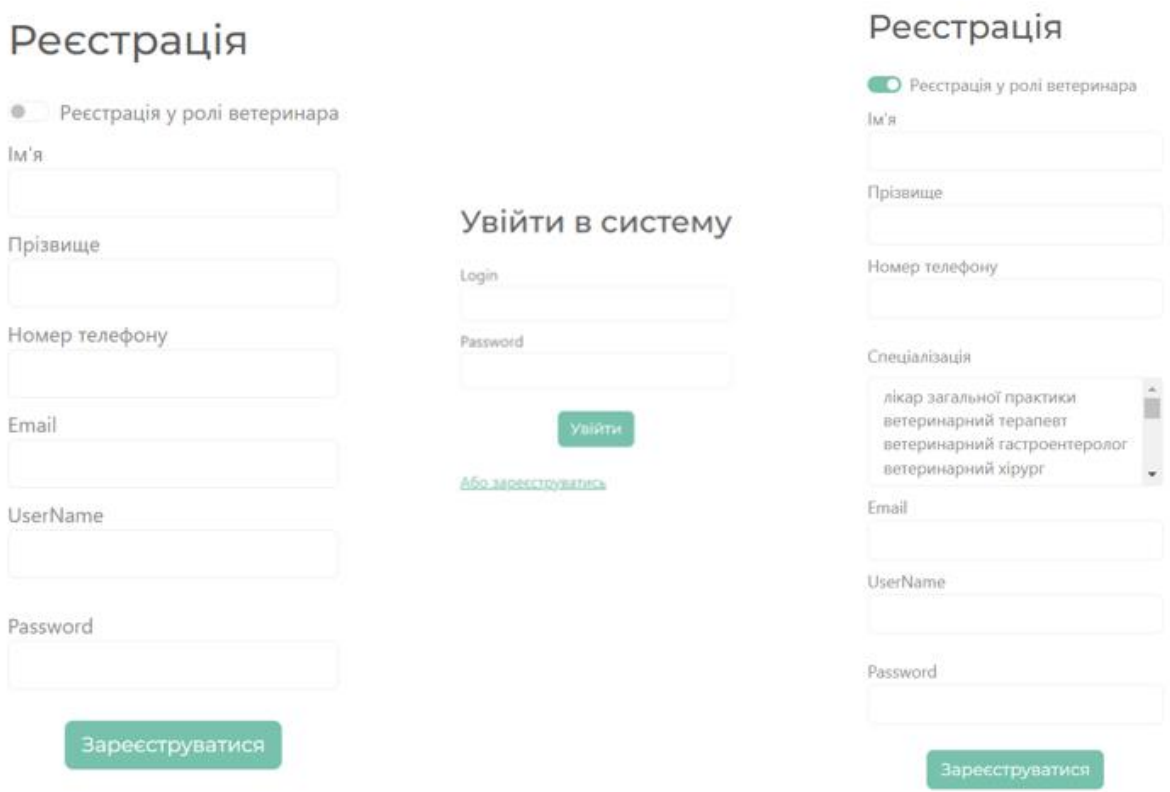


ПРИКЛАД РОБОТИ ВЕБЗАСТОСУНКУ



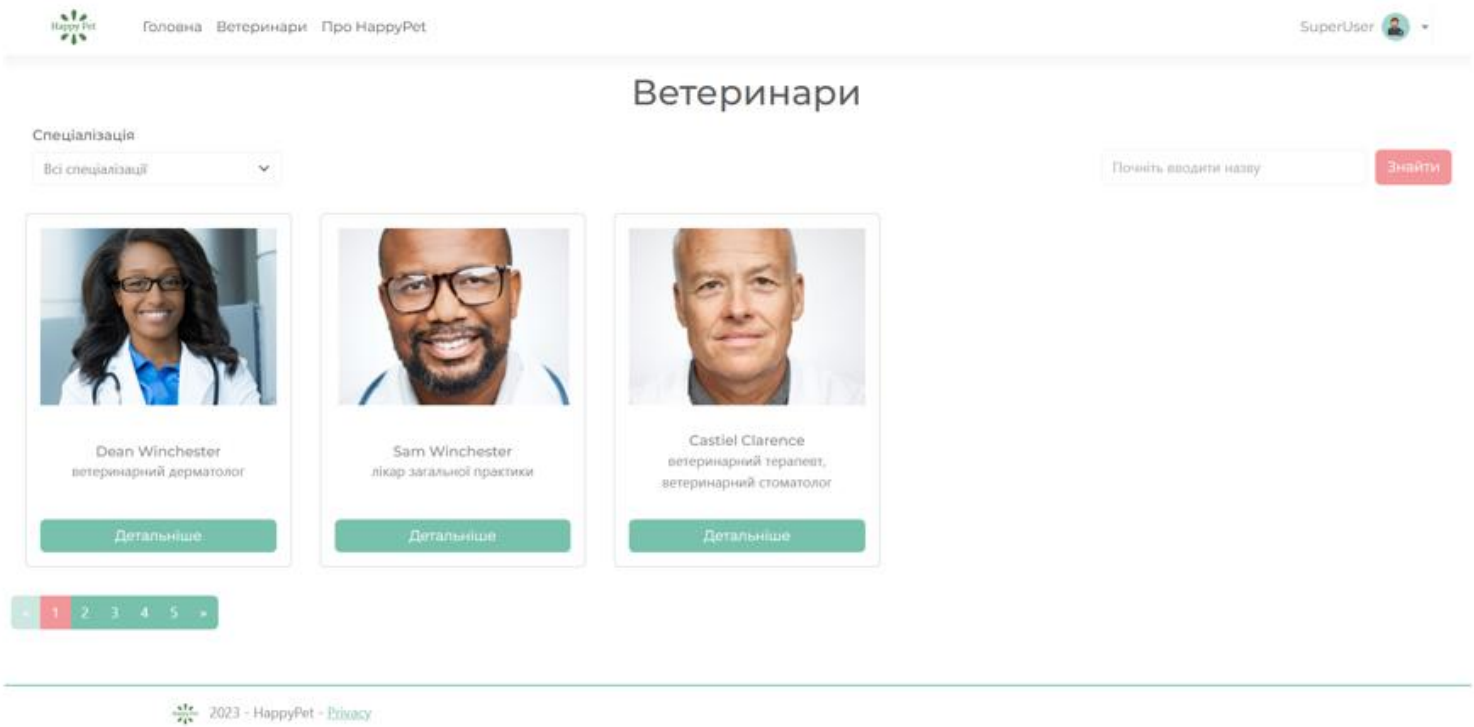
Головна сторінка

ПРИКЛАД РОБОТИ ВЕБЗАСТОСУНКУ



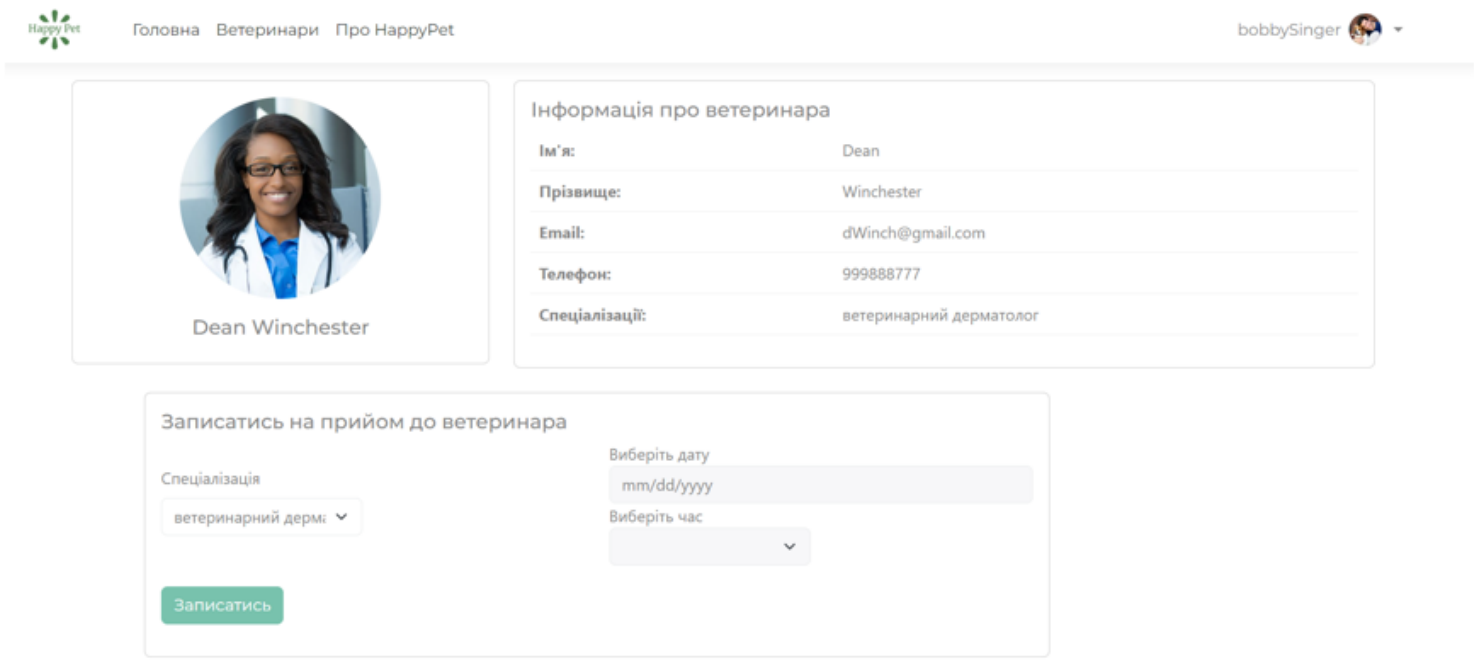
Реєстрація у системі для власника тварини та ветеринара. Вхід у систему

ПРИКЛАД РОБОТИ ВЕБЗАСТОСУНКУ



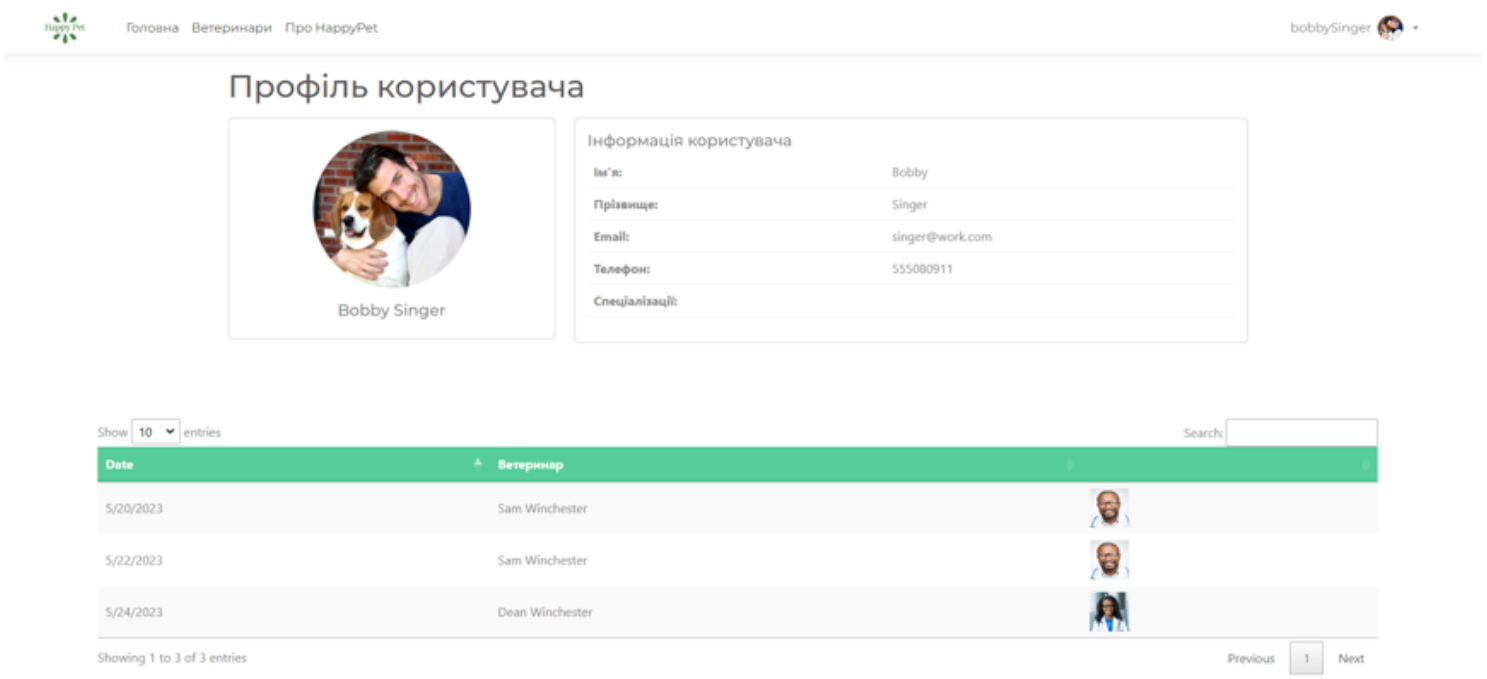
Сторінка усіх ветеринарів системи

ПРИКЛАД РОБОТИ ВЕБЗАСТОСУНКУ. ВЛАСНИК ТВАРИНИ



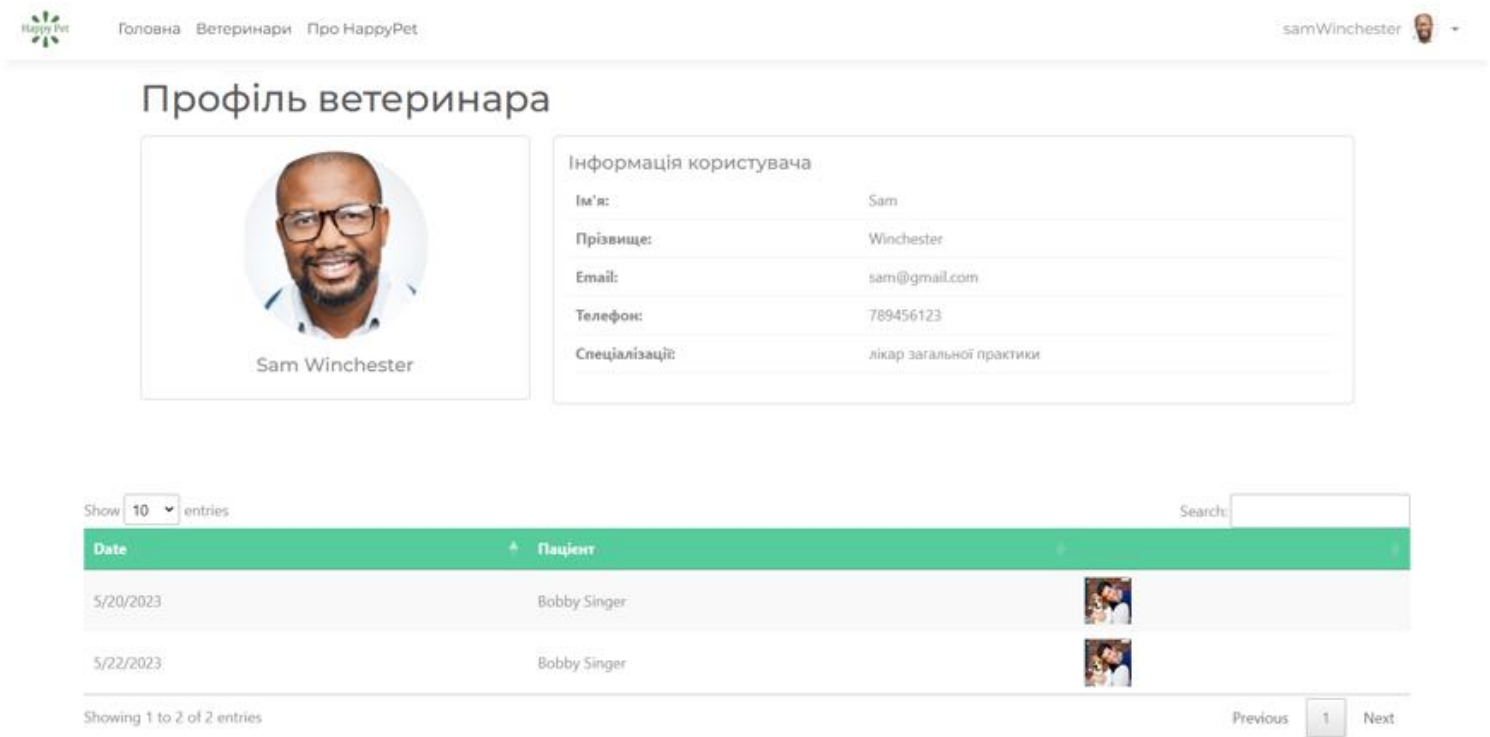
Інформація про ветеринара. Запис на прийом

ПРИКЛАД РОБОТИ ВЕБЗАСТОСУНКУ. ВЛАСНИК ТВАРИНИ



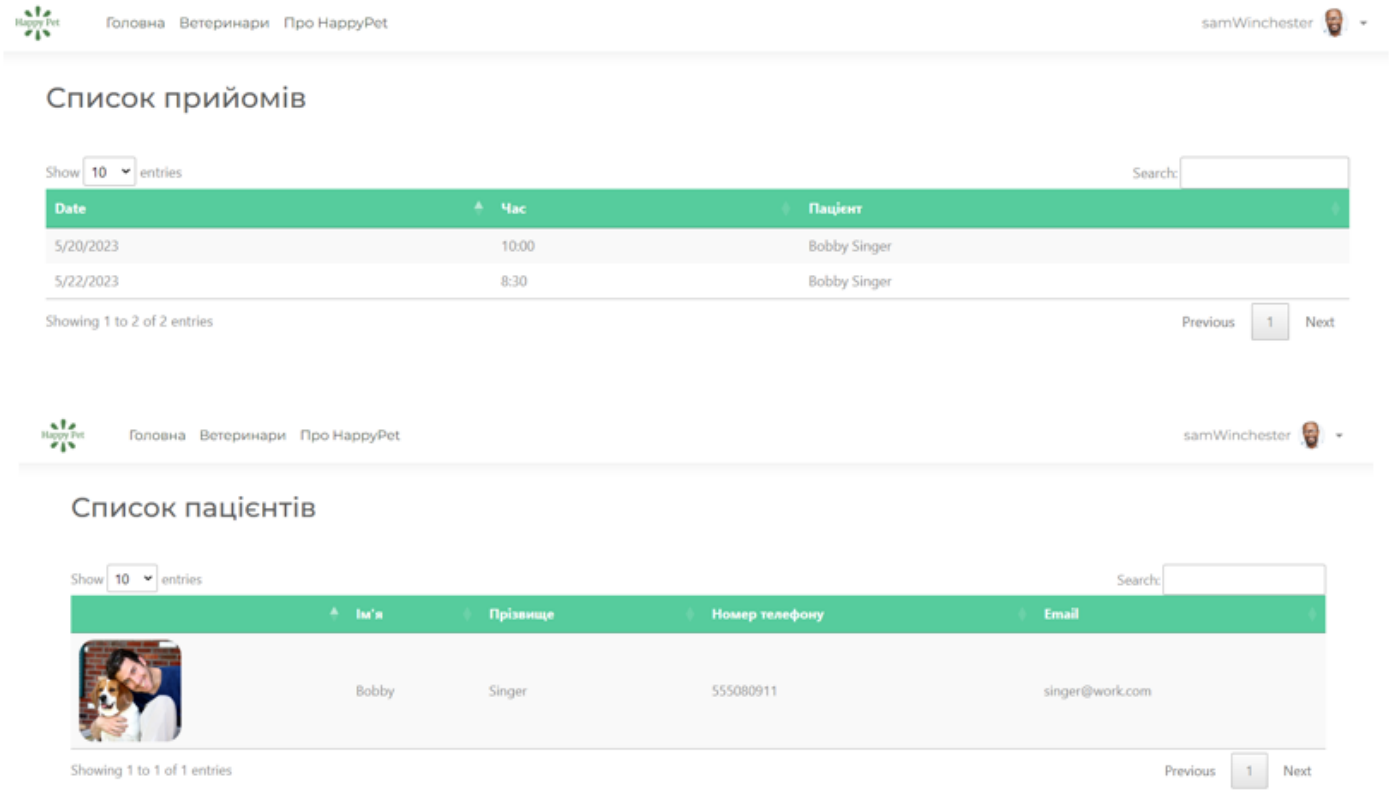
Сторінка профілю власника тварини

ПРИКЛАД РОБОТИ ВЕБЗАСТОСУНКУ. ВЕТЕРИНАР



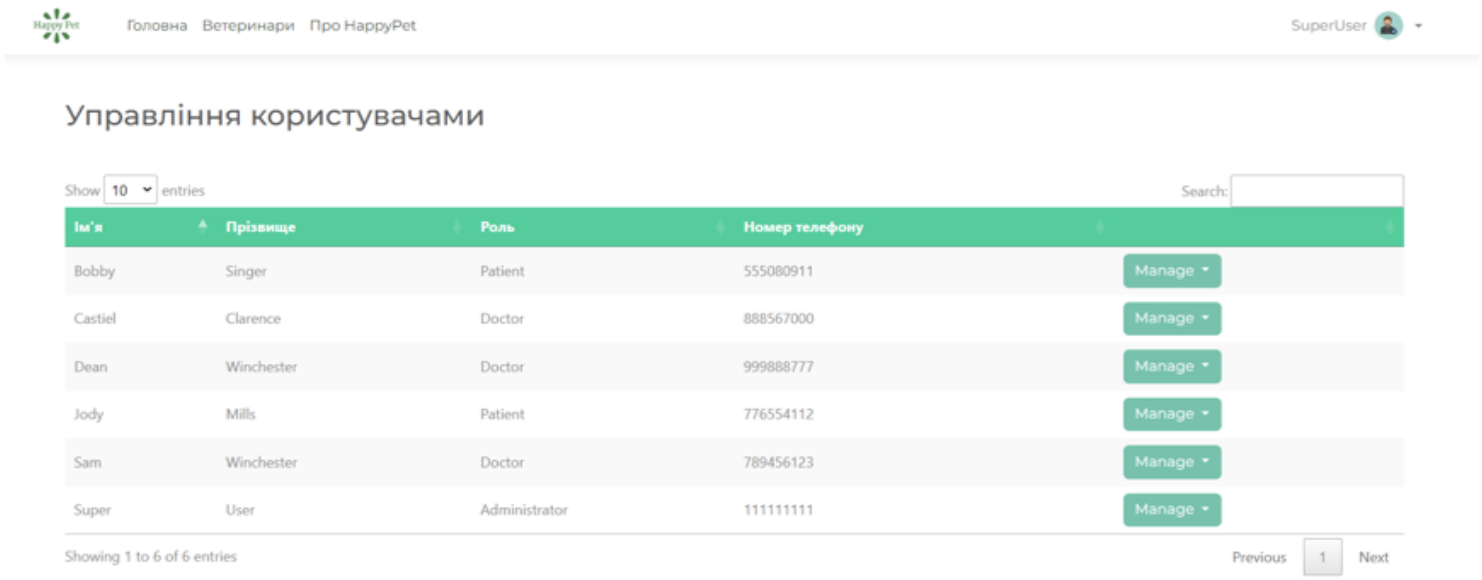
Сторінка профілю власника тварини

ПРИКЛАД РОБОТИ ВЕБЗАСТОСУНКУ. ВЕТЕРИНАР



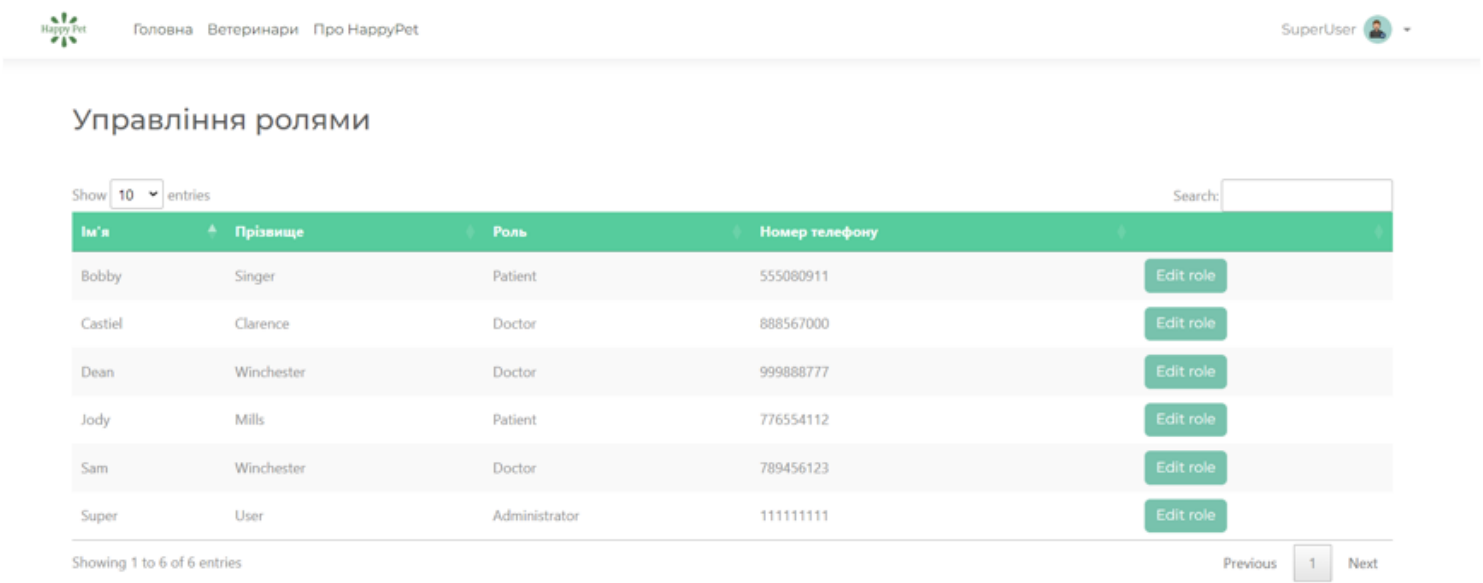
Сторінки зі списками усіх прийомів та пацієнтів ветеринара

ПРИКЛАД РОБОТИ ВЕБЗАСТОСУНКУ. АДМІНІСТРАТОР



Сторінка для керування користувачами системи

ПРИКЛАД РОБОТИ ВЕБЗАСТОСУНКУ. АДМІНІСТРАТОР



Сторінка для керування ролями користувачів

ОЦІНЮВАННЯ ЯКОСТІ РОЗРОБЛЕНОГО ПЗ

Функціональна працездатність елементів сторінок вебзастосунку

Зручність пошуку ветеринара

Онлайн запис на прийом до ветеринара

Можливість скасовувати прийом

Зручний та інтуїтивно зрозумілий інтерфейс

ПОРІВНЯННЯ РОЗРОБЛЕНОГО ПЗ ТА АНАЛОГІВ. ТЕСТУВАННЯ

Критерій	PetAdvisor	PetLive	Vetconsult.gladpet	HappyPet
Функціональна працездатність елементів сторінок вебдодатку	+	+	+	+
Зручність пошуку ветеринара	+	+	–	+
Онлайн запис на прийом до ветеринара	–	–	–	+
Можливість скасовувати прийом	–	–	–	+
Зручний та інтуїтивно зрозумілий інтерфейс	–	+	+/–	+

НАПРЯМКИ ПОДАЛЬШОГО РОЗВИТКУ ПРОЄКТУ

Онлайн-прийом у ветеринара через чат або відео-зв'язок

Наявність блогу з статтями пов’язаними з доглядом за тваринами

Зберігання медичної інформаціїї тварини

Підтримка мобільної версії вебзастосунку

ВИСНОВКИ


Проаналізовано предметну область та існуючі програмні рішення, виділено переваги та недоліки кожного з них.


Визначено та описано вимоги до розроблюваного вебзастосунку “HappyPet”.


Запропоновано архітектуру ПЗ та структуру БД.

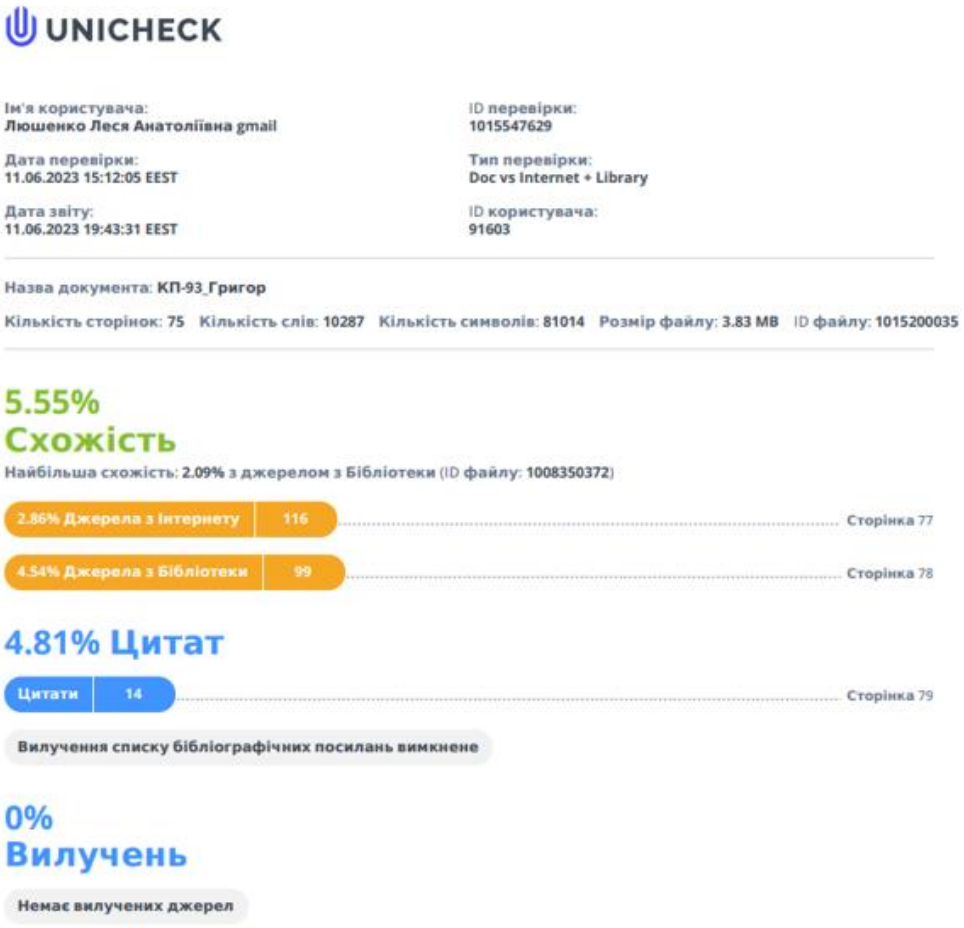

Розроблено алгоритм роботи вебзастосунку “HappyPet”.


Розроблено ПЗ, яке дозволяє шукати ветеринара та записуватись на прийом, керувати власними прийомами.


Протестовано та порівняно ПЗ з аналогами за основними критеріями та встановлено, що розробка повністю відповідає вимогам.


Запропоновано напрямки для подальшого розвитку вебзастосунку “HappyPet”.

УНІКАЛЬНІСТЬ



UNICHECK

Ім'я користувача:
Люшенко Леся Анатоліївна gmail

ID перевірки:
1015547629

Дата перевірки:
11.06.2023 15:12:05 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
11.06.2023 19:43:31 EEST

ID користувача:
91603

Назва документа: КП-93_Григор

Кількість сторінок: 75 Кількість слів: 10287 Кількість символів: 81014 Розмір файлу: 3.83 MB ID файлу: 1015200035

5.55%
Схожість

Найбільша схожість: 2.09% з джерелом з Бібліотеки (ID файлу: 1008350372)

2.86% Джерела з Інтернету 116 Сторінка 77

4.54% Джерела з Бібліотеки 99 Сторінка 78

4.81% Цитат

Цитати 14 Сторінка 79

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Дякую за увагу!

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

"ЗАТВЕРДЖЕНО"

Завідувач кафедри

_____ Євгенія СУЛЕМА

"___" _____ 2022 р.

ВЕБЗАСТОСУНОК ДЛЯ УПРАВЛІННЯ НАДАННЯМ ВЕТЕРИНАРНИХ ПОСЛУГ

Програма та методика тестування

ДП.045440-04-51

"ПОГОДЖЕНО"

Керівник проєкту:

_____ Леся ЛЮШЕНКО

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Олена ГРИГОР

2022

ЗМІСТ

1. ОБ’ЄКТ ВИПРОБУВАНЬ

Вебзастосунок для управління наданням ветеринарних послуг, створений на платформі .Net з використанням фреймворку ASP.NET Core та технології Bootstrap.

2. МЕТА ТЕСТУВАННЯ

У процесі тестування має бути перевірено наступне:

1. Функціональна працездатність елементів сторінок вебзастосунку.
2. Наявність доступу до бази ветеринарів.
3. Можливість переглядати заплановані прийоми до ветеринара.
4. Можливість записуватись на прийом до ветеринара.
5. Забезпечення належного рівня безпеки даних.
6. Зручність роботи з вебзастосунком.
7. Відповідність дизайну вимогам Технічного завдання.

3. МЕТОДИ ТЕСТУВАННЯ

Тестування виконується методом Gray Box Testing. Перевіряється як код, так і безпосередньо програмний продукт на відповідність функціональним вимогам. Тестування відбувається на рівні «системного тестування».

Використовуються наступні методи:

1) функціональне тестування, зокрема на рівні Critical path test (базове тестування);
2) тестування продуктивності програмного забезпечення, зокрема Stability testing (тестування стабільності) та Load testing (навантажувальне тестування);
3) тестування інтерфейсу.

4. ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Працездатність вебзастосунку перевіряється шляхом:

1) динамічного ручного тестування – введенням граничних та недопустимих значень в поля, які можна редагувати;
2) динамічного ручного тестування на відповідність функціональним вимогам;
3) статичного тестування коду;
4) тестування вебзастосунку в різних веббраузерах;
5) тестування при максимальному навантаженні;
6) тестування стабільності роботи при різних умовах;
7) тестування зручності використання;
8) тестування інтерфейсу.

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

"ЗАТВЕРДЖЕНО"

Завідувач кафедри

_____ Євгенія СУЛЕМА

"___" _____ 2023 р.

ВЕБЗАСТОСУНОК ДЛЯ УПРАВЛІННЯ НАДАННЯМ ВЕТЕРИНАРНИХ ПОСЛУГ

Керівництво користувача

ДП.045440-05-34

"ПОГОДЖЕНО"

Керівник проєкту:

_____ Леся ЛЮШЕНКО

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Олена ГРИГОР

2023

ЗМІСТ

1. Опис структури вебзастосунку

Користувацький інтерфейс вебзастосунку для управління наданням ветеринарних послуг містить наступні сторінки:

- Головна сторінка.
- Сторінка реєстрації.
- Сторінка авторизації.
- Сторінка для перегляду списку та пошуку ветеринарів.
- Сторінка з детальною інформацією про окремого ветеринара та можливістю запису на прийом.
- Сторінка зі списком запланованих прийомів.
- Сторінка зі списком пацієнтів.
- Сторінка з усіма користувачами системи.
- Сторінка для управління ролями користувачів.
- Особиста сторінка авторизованого користувача.

2. Головна сторінка вебзастосунку

При першому відвідуванні вебзастосунку неавторизований користувач потрапляє на головну сторінку (рис. 1), яка надає інформацію про призначення та головну ідею цього вебзастосунку.

У верхній частині міститься навігаційне меню. Клавіша «Ветеринари» перейде на сторінку з усіма ветеринарами, «Про HappyPet» – перейде на сторінку з інформацією про вебзастосунок. Кнопка «Увійти» відкриває сторінку авторизації у вебзастосунку.

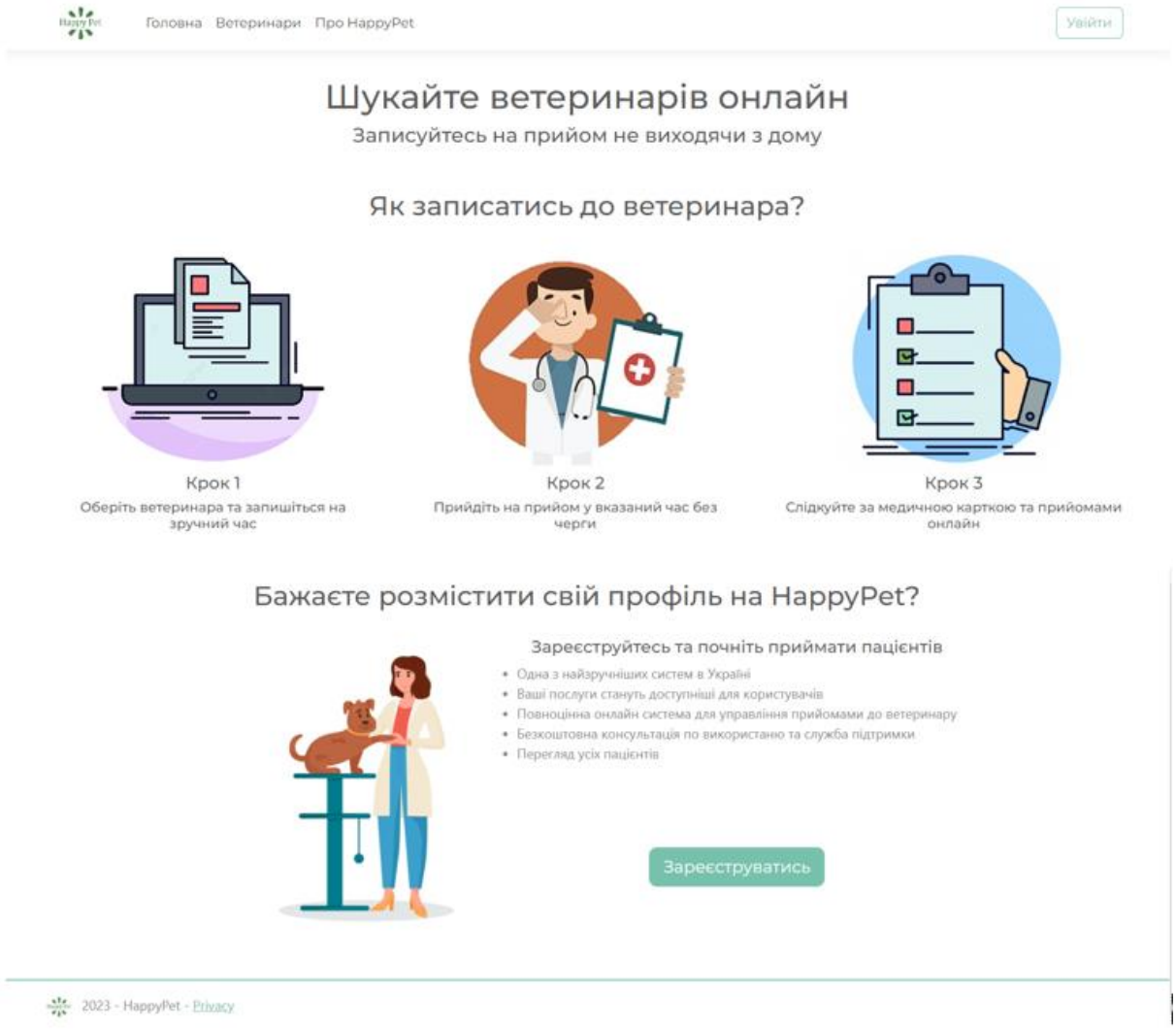


Рис. 1. Головна сторінка вебзастосунку

3. Процедура авторизації користувача

Користувачі мають змогу увійти у систему, відкривши сторінку авторизації (рис. 2). Деякі сторінки доступні лише авторизованим користувачам. Якщо користувач не зареєстрований у системі він може перейти на сторінку реєстрації натиснувши «Або зареєструватись».

У формі авторизації наявні два обов'язкових для заповнення поля: «Login» – для вводу електронної адреси або логіну та поле «Password» з паролем.

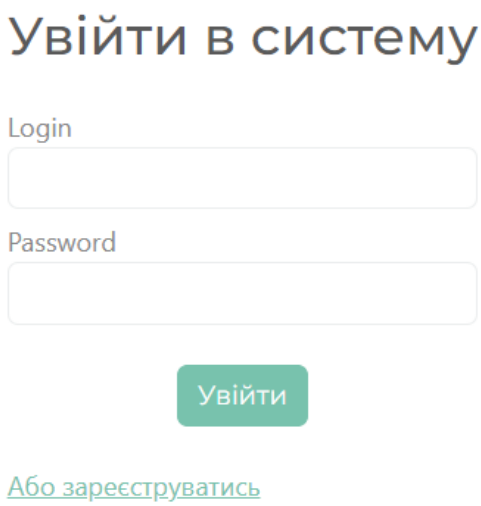


Рис. 2. Форма для авторизації користувача

4. Процедура реєстрації користувача

Користувачі мають змогу зареєструватись у системі у двох ролях – «Власник тварини» та «Ветеринар». Для обох ролей форма реєстрації містить обов’язкові поля для вводу – ім’я та прізвище користувача, номер телефону, електронна адреса, логін у системі та пароль (рис. 3). Для ролі «Ветеринар» наявне додаткове обов’язкове для заповнення поле для вказання спеціалізації ветеринара.

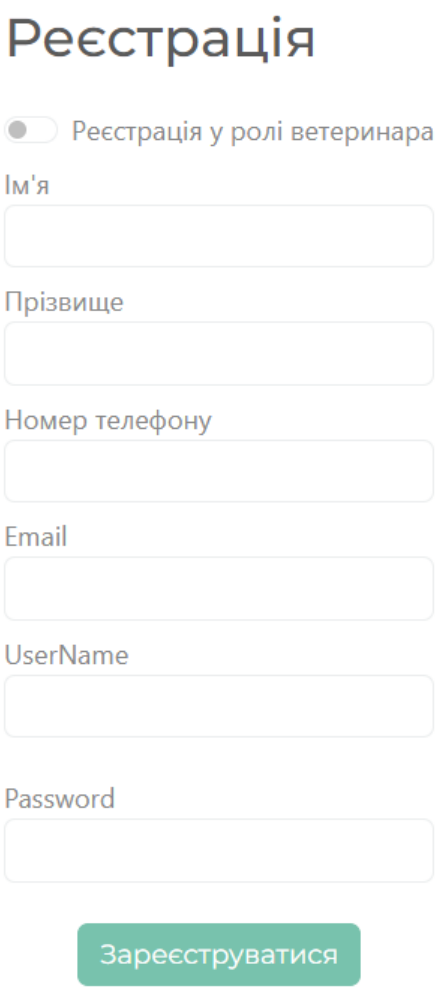


Рис. 3. Форма реєстрації користувача

5. Сторінка зі списком ветеринарів

При переході на вкладку «Ветеринари» відображається сторінка зі списком усіх існуючих у системі ветеринарів (рис. 4). На даній сторінці можна фільтрувати ветеринарів за спеціалізацією. А також здійснювати пошук ветеринара за прізвищем. Після натискання на кнопку «Детальніше», користувач направиться на сторінку з переглядом детальної інформації про ветеринара.

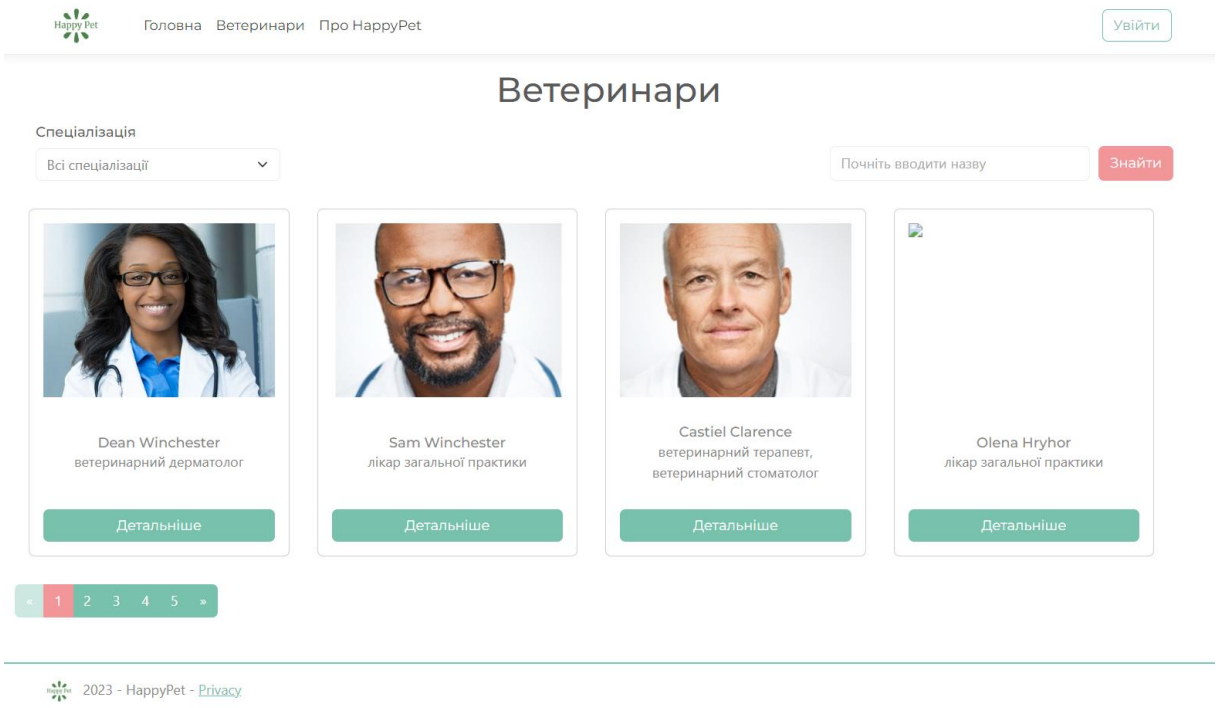


Рис. 4. Сторінка зі списком ветеринарів

6. Сторінка з детальною інформацією про ветеринара

Дана сторінка містить у собі повну інформацію про ветеринара – прізвище та ім’я ветеринара, електронна адреса, номер телефону та спеціалізації (рис. 5).

У нижній частині сторінки є спеціальна форма для запису на прийом до цього ветеринара, яка доступна для авторизованих користувачів у ролі «Власник тварини». Дана форма містить поля для вибору спеціалізації за якою планується прийом, а також дати та часу. Для запису на прийом необхідно заповнити ці поля та натиснути на кнопку «Записатись».

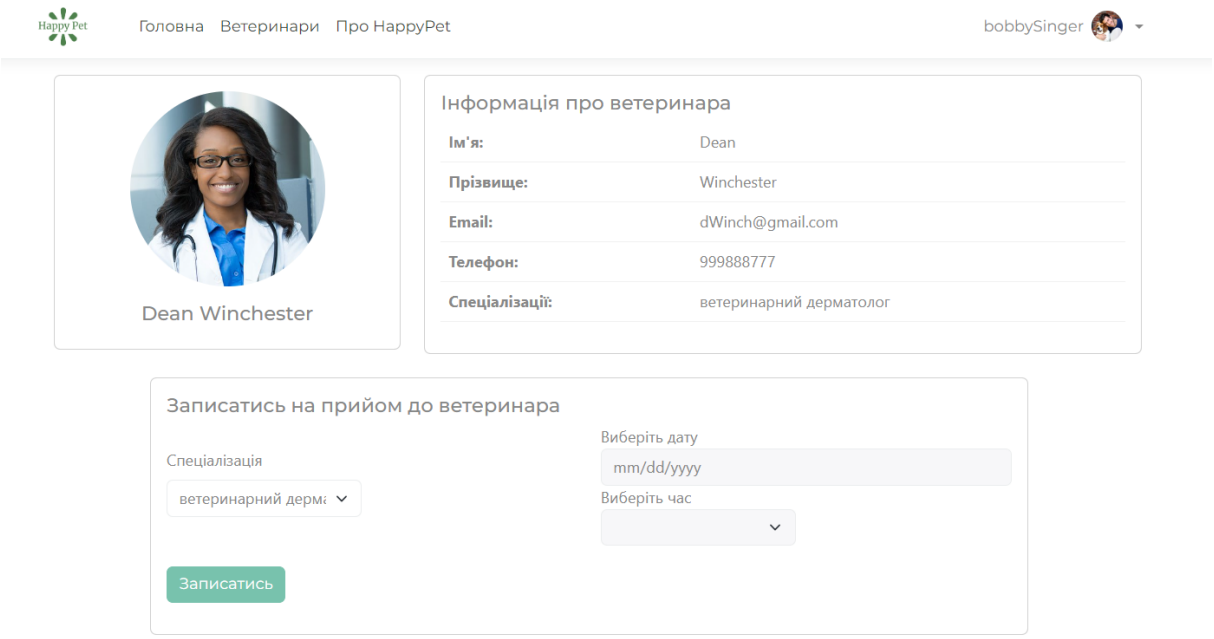


Рис. 5. Сторінка з детальною інформацією про ветеринара

7. Сторінка з переліком запланованих прийомів

При переході на «Прийоми» у підменю авторизованого користувача, яке розташоване справа на головному навігаційному меню, відображається сторінка з пагінованим переліком усіх запланованих прийомів користувача (рис. 6). Пагінація здійснюється за допомогою навігаційної панелі, що знаходиться у нижній частині сторінки. У цьому переліку відображена загальна інформація про прийоми, а саме: дата, час та ім’я і прізвище ветеринара. Для ролі «Ветеринар» відображаються наступні поля: дата, час та ім’я і прізвище пацієнта.

На цій же сторінці навпроти кожного прийому знаходиться кнопка «Скасувати», яка відповідає за скасування прийому. При натисканні на неї у центрі екрану з’являється модальне вікно, у якому необхідно підтвердити скасування прийому.

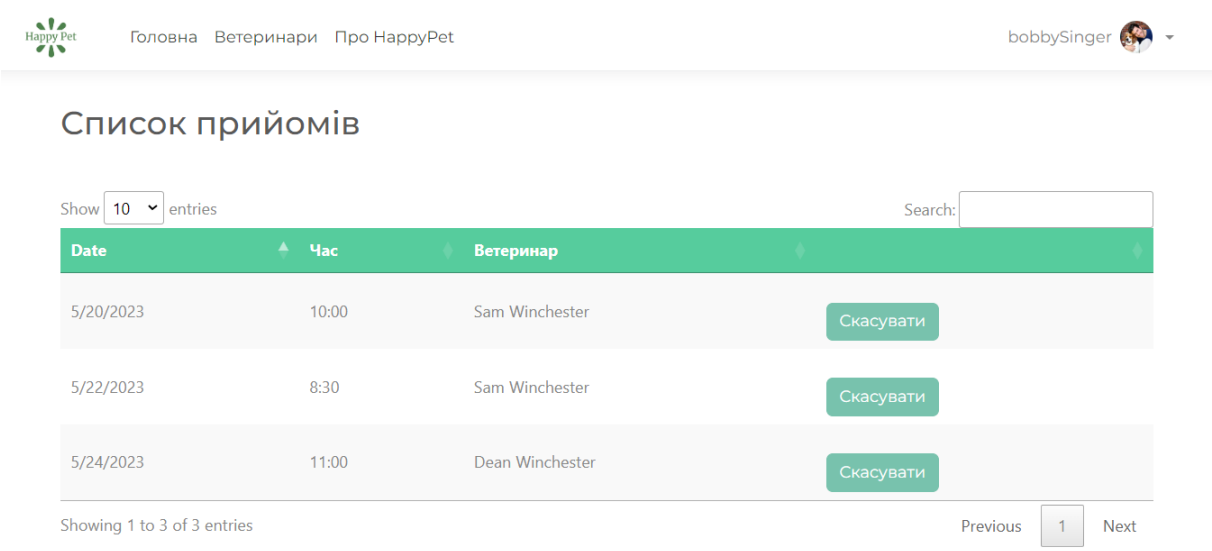


Рис. 6. Сторінка з переліком запланованих прийомів

8. Сторінка з переліком пацієнтів ветеринара

При переході на «Пацієнти» у підменю авторизованого користувача, яке розташоване справа на головному навігаційному меню, відображається сторінка з пагінованим переліком усіх запланованих прийомів користувача (рис. 7). Пагінація здійснюється за допомогою навігаційної панелі, що знаходиться у нижній частині сторінки. У цьому переліку відображена інформація про пацієтів, а саме: прізвище та ім’я, номер телефону та електронна адреса.

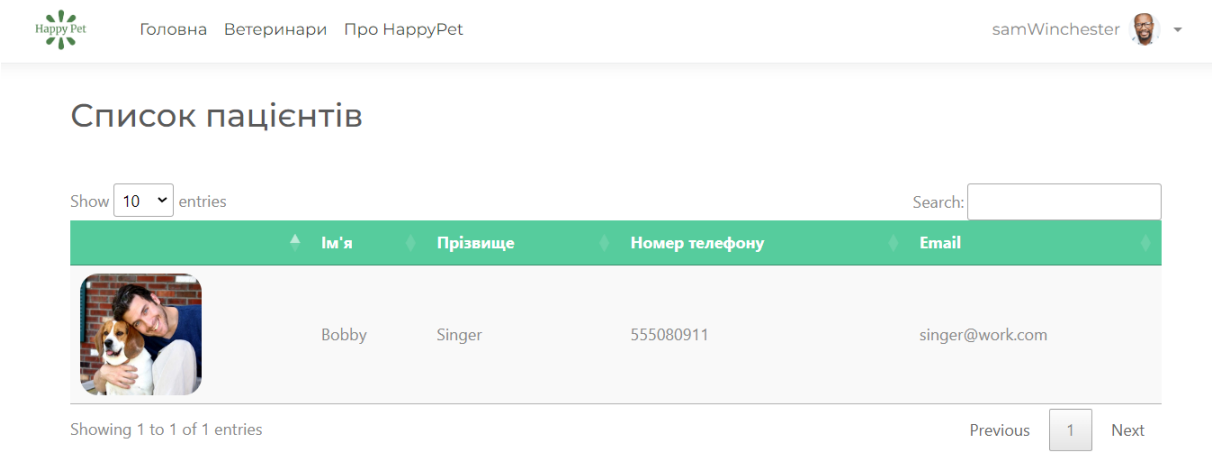


Рис. 7. Сторінка з переліком пацієнтів

9. Сторінка для управління користувачами

При переході на «Керувати користувачами» у підменю авторизованого у ролі «Адміністратор» користувача, яке розташоване справа на головному навігаційному меню, відображається сторінка з усіма зареєстрованими у системі користувачами (рис. 8). Пагінація здійснюється за допомогою навігаційної панелі, що знаходиться у нижній частині сторінки. У цьому переліку відображена інформація про користувачів, а саме: прізвище та ім’я, номер телефону та роль користувача у системі.

Натиснувши кнопку «Manage» з обраного підменю можна обрати «Delete» для видалення користувача з системи та «Edit» для редагування даних користувача.

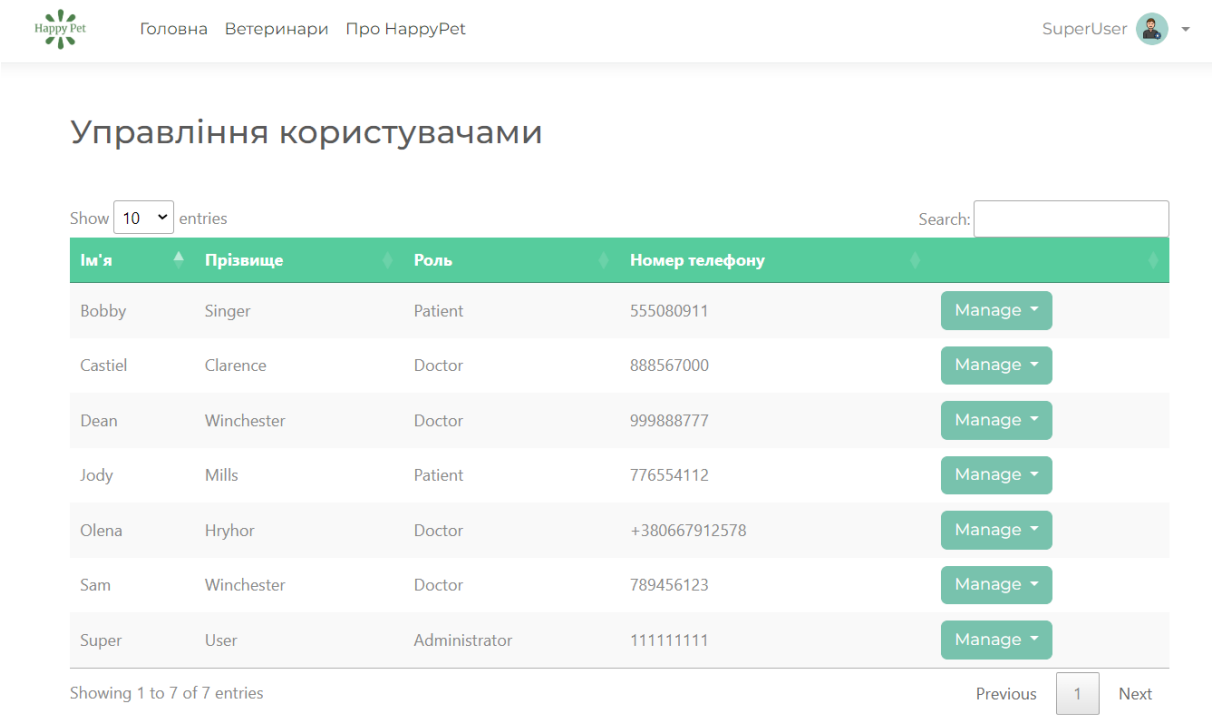


Рис. 8. Сторінка з усіма користувачами вебзастосунку

10. Сторінка для управління ролями

При переході на «Керувати ролями» у підменю авторизованого у ролі «Адміністратор» користувача, яке розташоване справа на головному навігаційному меню, відображається сторінка з усіма зареєстрованими у системі користувачами (рис. 9). Пагінація здійснюється за допомогою навігаційної панелі, що знаходиться у нижній частині сторінки. У цьому переліку відображена інформація про користувачів, а саме: прізвище та ім’я, номер телефону та роль користувача у системі.

Натиснувши кнопку «Edit role» з’являється модальне вікно з можливістю обрати нову роль для користувача.

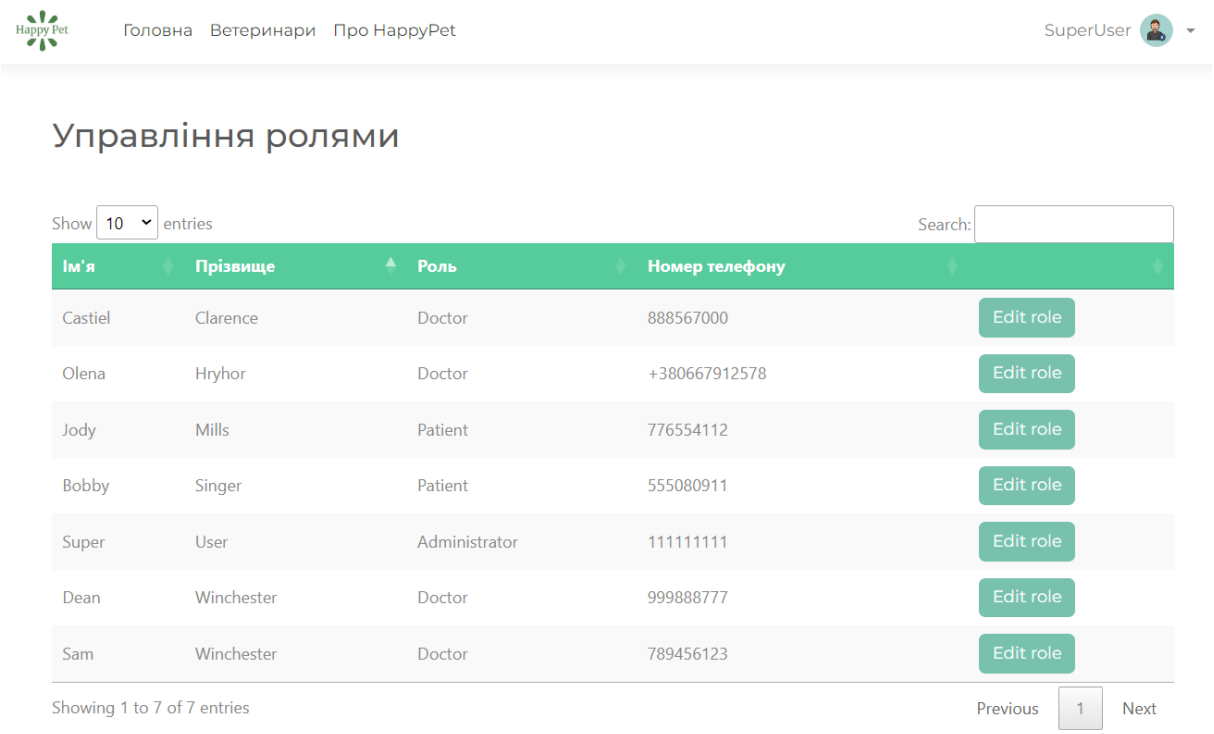


Рис. 9. Сторінка для управління ролями користувачів

11. Особиста сторінка зареєстрованого користувача

При переході на «Особисті дані» у підменю авторизованого користувача, яке розташоване справа на головному навігаційному меню, відображається сторінка з інформацією про користувача (рис. 10). Для перегляду доступні наступні дані: фотокартка користувача, ім’я та прізвище, номер телефону, електронна адреса, а також внизу сторінки відображається перелік запланованих прийомів користувача з пагінацією.

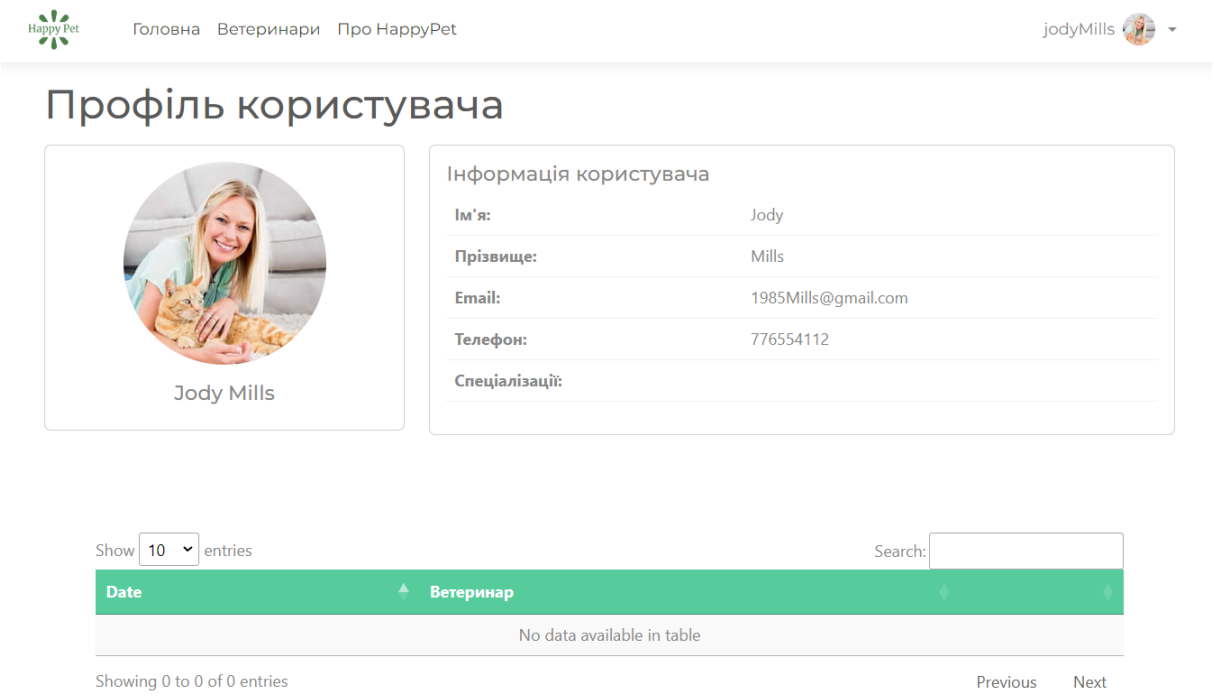


Рис. 10. Особиста сторінка зареєстрованого користувача