Міністерство освіти і науки України
Український державний університет науки і технологій

Факультет «Комп'ютерні технології і системи»

Кафедра «Комп'ютерні інформаційні технології»

Пояснювальна записка
до кваліфікаційної роботи бакалавра

на тему: «Розробка сайту ветеринарної клініки»
за освітньою програмою: «Інженерія програмного забезпечення»
зі спеціальності: «121 Інженерія програмного забезпечення»
Виконав: студент групи «ПЗ19130»

_____ /Микита КРОЛЬ/
(підпис студента) (Ім'я ПРІЗВИЩЕ)

Керівник: _____ /доц. Олександр ІВАНОВ/
(підпис) (посада, Ім'я ПРІЗВИЩЕ)

Нормоконтролер: _____ /доц. Олена КУРОП'ЯТНИК/
(підпис) (посада, Ім'я ПРІЗВИЩЕ)

Засвідчую, що у цій роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент _____
(підпис)

Дніпро – 2022 рік

Ministry of Education and Science of Ukraine
Ukrainian State University of Science and Technologies

Faculty «Computer technologies and systems»

Department «Computer information technology»

Explanatory Note
to Bachelor's Thesis

on the topic: «Development of the site of the veterinary clinic»
according to educational curriculum «Software engineering»
in the Speciality: «121 Software engineering»

Done by the student of the group PZ19130: /Mykyta KROL/

Scientific Supervisor: /Oleksandr IVANOV/

Normative controller: /Olena KUROPIATNYK/

Dnipro – 2022

Міністерство освіти і науки України
Український державний університет науки і технологій

Факультет: «Комп'ютерні технології і системи»
Кафедра: «Комп'ютерні інформаційні технології»
Рівень вищої освіти: бакалавр
Освітня програма: «Інженерія програмного забезпечення»
Спеціальність: «121 Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри КІТ
_____ /Вадим ГОРЯЧКІН/
(підпис)
Дата 22.12.2021

З А В Д А Н Н Я

на кваліфікаційну роботу бакалавра
студенту Кролю Микиті Євгеновичу
1. Тема роботи: «Розробка сайту ветеринарної клініки»
Керівник роботи: Іванов Олександр Петрович, доцент
затверджені наказом № 77 ст від 08.12.2021
2. Строк подання студентом роботи: 29.05.2022 р.
3. Вихідні дані до роботи: _____

4. Зміст пояснювальної записки (перелік питань, які потрібно опрацювати):
Збір та аналіз вимог, зовнішнє проєктування, внутрішнє проєктування, проєктування архітектури системи, розробка програми, тестування та налагодження, висновки та рекомендації

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень):

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Постановка задачі. Збір інформації. Визначення та обґрунтування критеріїв розробки додатку.	До 18.01.22	
2	Визначення методів вирішення задачі. Визначення вимог до технічних та програмних засобів.	До 18.02.22	
3	Програмування та відладка програми.	До 01.05.22	
4	Тестування програми.	До 24.05.22	
5	Розробка, програмної документації.	До 03.06.22	
6	Узгодження і затвердження програмної документації	До 07.06.22	
7	Подання кваліфікаційної роботи до кафедри	08.06.22	
8	Захист кваліфікаційної роботи на засіданні Екзаменаційної комісії	23.06.22	

Студент _____ Микита КРОЛЬ
(підпис) (Ім'я ПРІЗВИЩЕ)

Керівник роботи _____ доц. Олександр ІВАНОВ
(підпис) (Ім'я ПРІЗВИЩЕ)

РЕФЕРАТ

Мета дипломного проекту – створення сайту, що являє собою систему керування записами про пацієнтів ветеринарної клініки для окремо взятого лікаря.

Об’єкт дипломної роботи – швидка розробка сайтів на мові C#, використовуючи її можливості для роботи з базою даних через вбудовані можливості, такі як:

- міграція – спеціальна можливість, завдяки якій таблиці бази даних будуються відносно створених класів у програмі;
- спеціальне – API ASP.Net Identity, яке виконує дії по керуванню користувачами, паролями, даними профілю, ролями, підтвердженням електронною поштою, авторизації через сторонні сервіси.

Предмет дипломної роботи – сайт, що використовується для ведення інформації про пацієнтів клініки та створення розкладу їх прийомів.

Пояснювальна записка складається зі вступу, 6 розділів, висновків, бібліографічного списку та 4 додатків:

- вступ описує актуальність побудови сайту та засоби його розробки мовою C#, містить 1 сторінка;
- перший розділ – описує постановку задачі та загальні вимоги до сайту, містить 1 сторінку;
- другий розділ – описує призначення сайту та опис функціональних вимог, містить 2 сторінки;
- третій розділ – описує архітектуру системи, містить 11 сторінок;
- четвертий розділ – де описані внутрішня будова сайту, містить 7 сторінок;
- п’ятий розділ – описує алгоритм роботи модулів та їх рівні, містить 3 сторінки;
- шостий розділ – описує процес тестування та налагодження програми, містить 2 сторінки;
- додатки – технічне завдання, специфікацію й опис програми.

Усього робота містить: таблиць – 9, рисунків – 26, бібліографія – 4 джерела.

Ключові слова: SQL, MS SQL, C#, ASP.NET, VETERINARY CLINIC.

ЗМІСТ

ВСТУП

Веб-застосунок «Сайт ветеринарної клініки», що розробляється призначений для керування записами про пацієнтів ветеринарної клініки для окремо взятого лікаря.

Малі клініки, які не є частинами великих мереж, зазвичай не мають своїх сайтів, а подекуди навіть записують дані про відвідування пацієнтів та їх лікування в спеціальні журнали, розробка такого сайту дозволяє підвищити ефективність роботи таких закладів, оскільки програма надає широкі можливості по роботі з даними, наприклад, очевидною перевагою є швидкий пошук пацієнта та його історії лікування, який може виконуватись навіть за допомогою браузера.

Область застосувань: користувачі – лікарі клініки, які ведуть обстеження та облік пацієнтів. Лікарі зможуть заносити дані про обстеження пацієнта та матимуть можливість створювати записи про майбутні візити пацієнтів.

Експлуатаційне призначення – завдяки роботі на сайті користувачі (ветеринари), матимуть можливість зберігати свої дані в зручному вигляді, відпаде необхідність в постійному зберіганні даних на папері й з’явиться можливість швидкого пошуку необхідної інформації записаної до медичної картки пацієнта.

1 ЗБІР ТА АНАЛІЗ ВИМОГ

Постановка задачі:

- розробити програмний продукт, що представляє собою сайт – особистий кабінет ветеринара;
- сайт повинен забезпечити можливість зберігати дані про пацієнтів ветеринара у зручному вигляді.

За результатом збору та аналізу вимог було сформовано функціональні вимоги до системи.

Сайт повинен мати можливості реєстрації та авторизації.

Авторизований користувач має доступ до свого особистого простору пацієнтів, в якому він керує записами, що містять інформацію про:

- господарів тварин;
- тварин;
- стан тварини на поточному прийомі.

Ветеринар може змінювати пароль свого облікового запису та планувати наступні візити для пацієнтів. Також він повинен мати можливість фільтрації даних медичних карток, за такими критеріями, як кличка тварина, діагноз, симптоми та вказувати діапазон записів у вигляді початкової та кінцевої дати.

Елементи вводу даних повинні контролювати інформацію, що вводить користувач.

Також один ветеринар не повинен мати доступ до господарів, тварин та розкладів інших ветеринарів, тобто забезпечити розмежування доступу до даних для ветеринарів.

Висновки до розділу 1

Цей розділ містить збір та аналіз вимог до сайту, опис основних можливостей для початкової ітерації проекту, без яких проект є непрацездатним.

На наступних ітераціях можливо додати вимоги щодо фільтрації даних на інших сторінках та можливість сортування.

2 ЗОВНІШНЄ ПРОЄКТУВАННЯ

Функціональне призначення – організація роботи ветеринарів за допомогою інтернет-сайту.

Експлуатаційне призначення – завдяки роботі на сайті ветеринари, мають можливість зберігати свої дані в зручному вигляді, документація перейде у електронний вигляд, з’явиться можливість швидкого пошуку необхідної інформації записаної до медичної картки пацієнта.

Функціональне вимоги будуть описані у вигляді діаграм прецедентів (рис. 2.1 – 2.7).

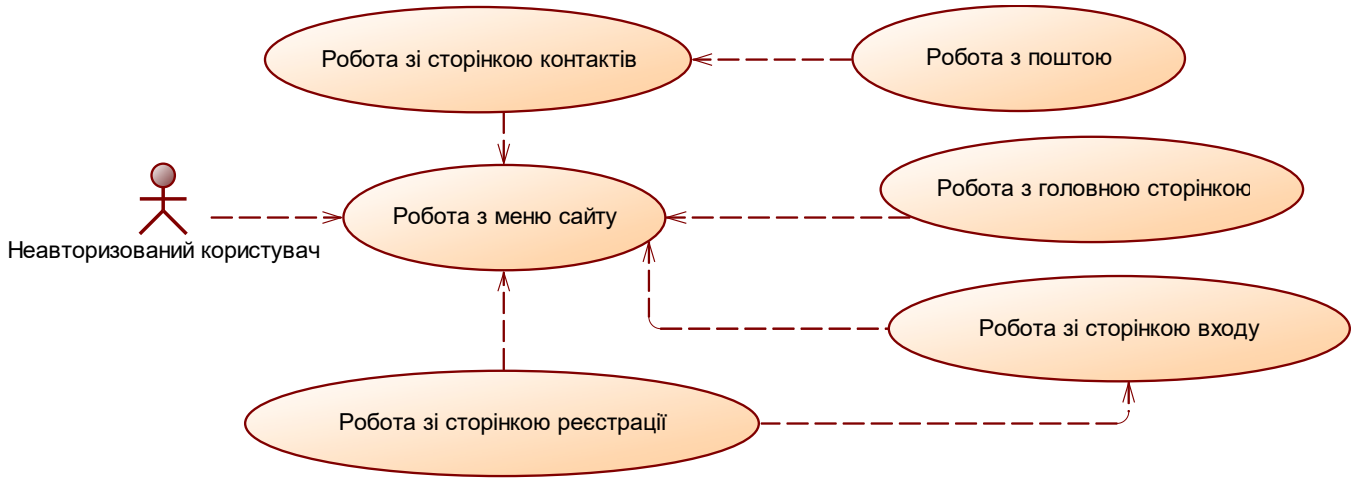

Рисунок 2.1 – Діаграма прецедентів неавторизованого користувача

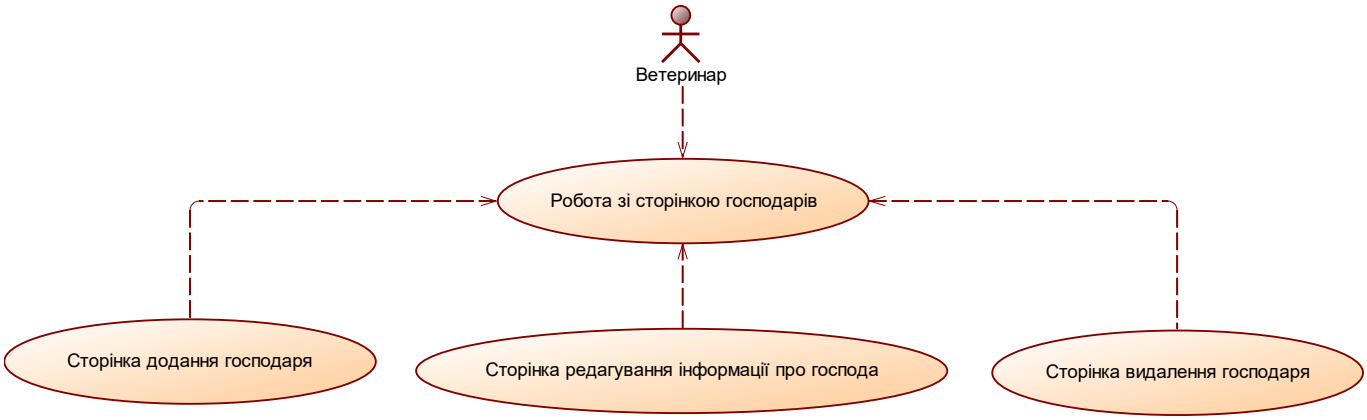

Рисунок 2.2 – Діаграма прецедентів для ветеринара при роботі зі сторінкою господарів

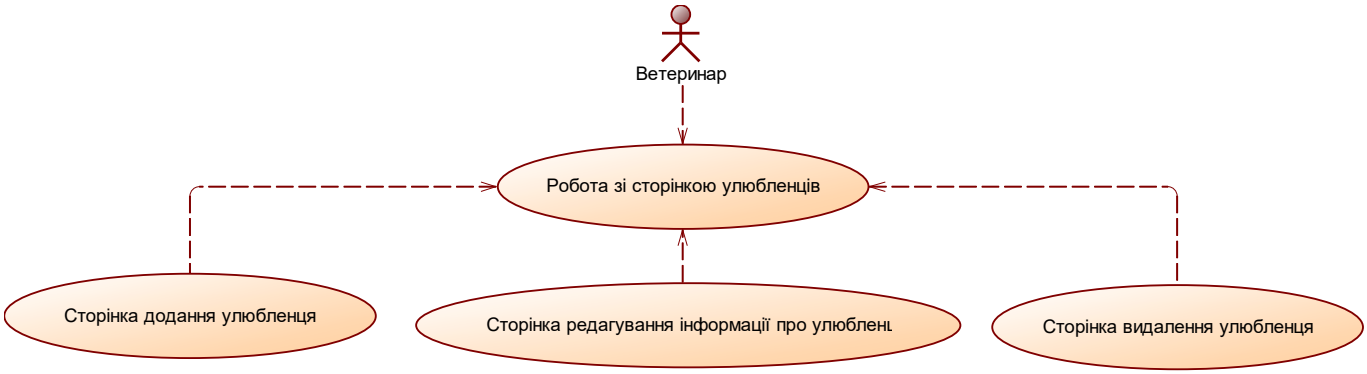

Рисунок 2.3 – Діаграма прецедентів для ветеринара при роботі зі сторінкою улюбленців

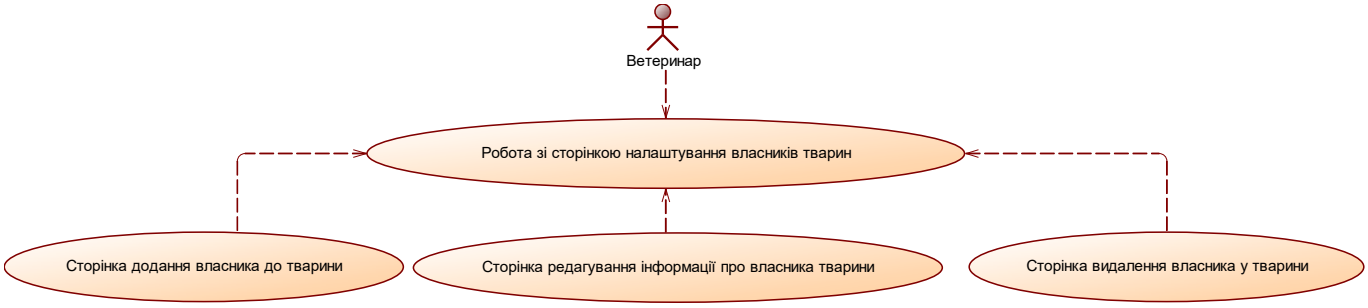

Рисунок 2.4 – Діаграма прецедентів для ветеринара при роботі зі сторінкою власників тварин

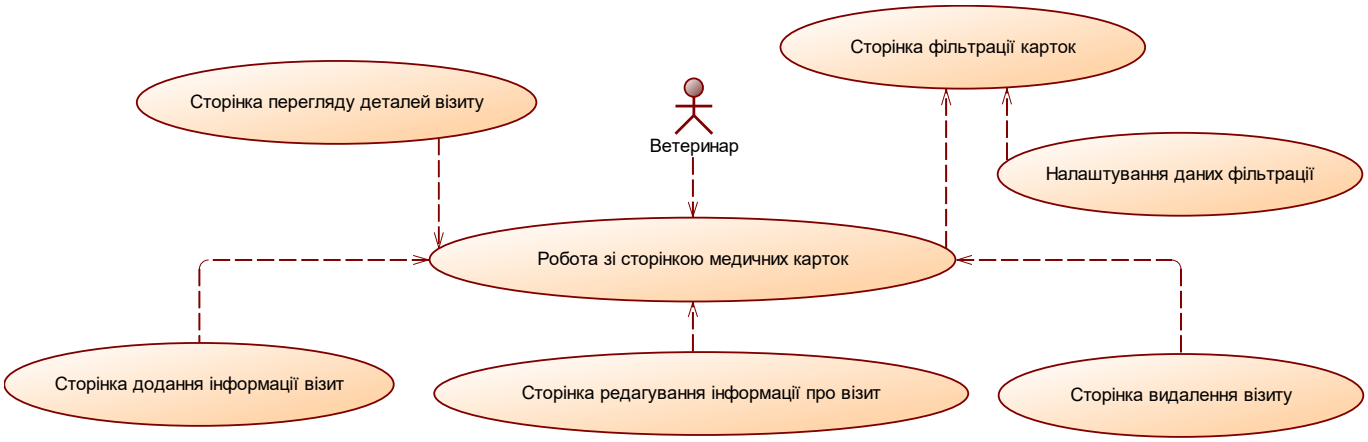

Рисунок 2.5 – Діаграма прецедентів для ветеринара при роботі зі сторінкою медичних карток

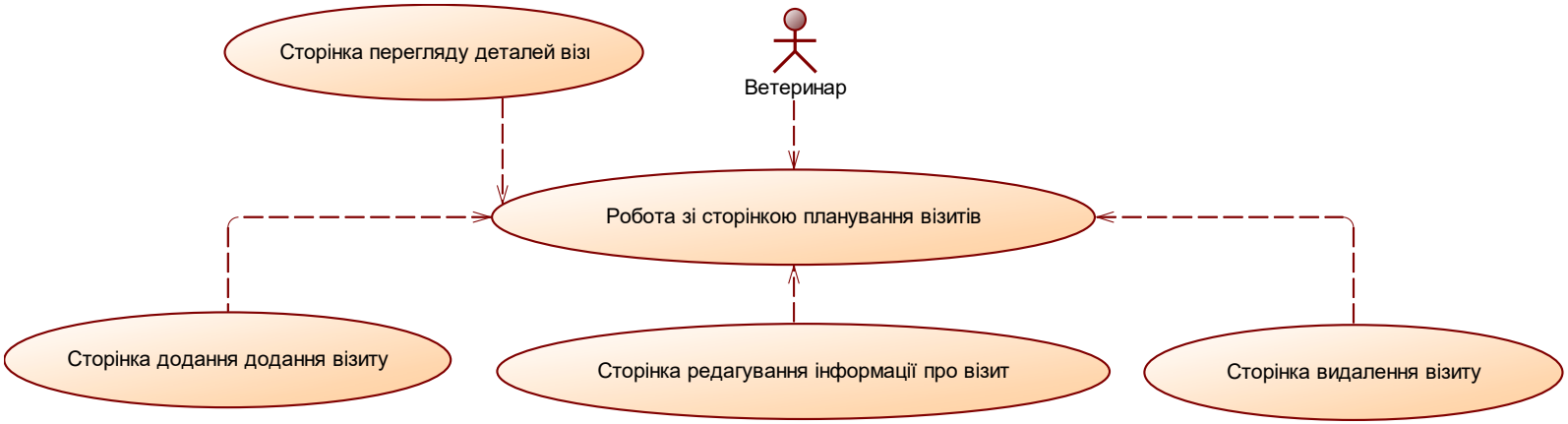

Рисунок 2.6 – Діаграма прецедентів для ветеринара при роботі зі сторінкою планування візитів

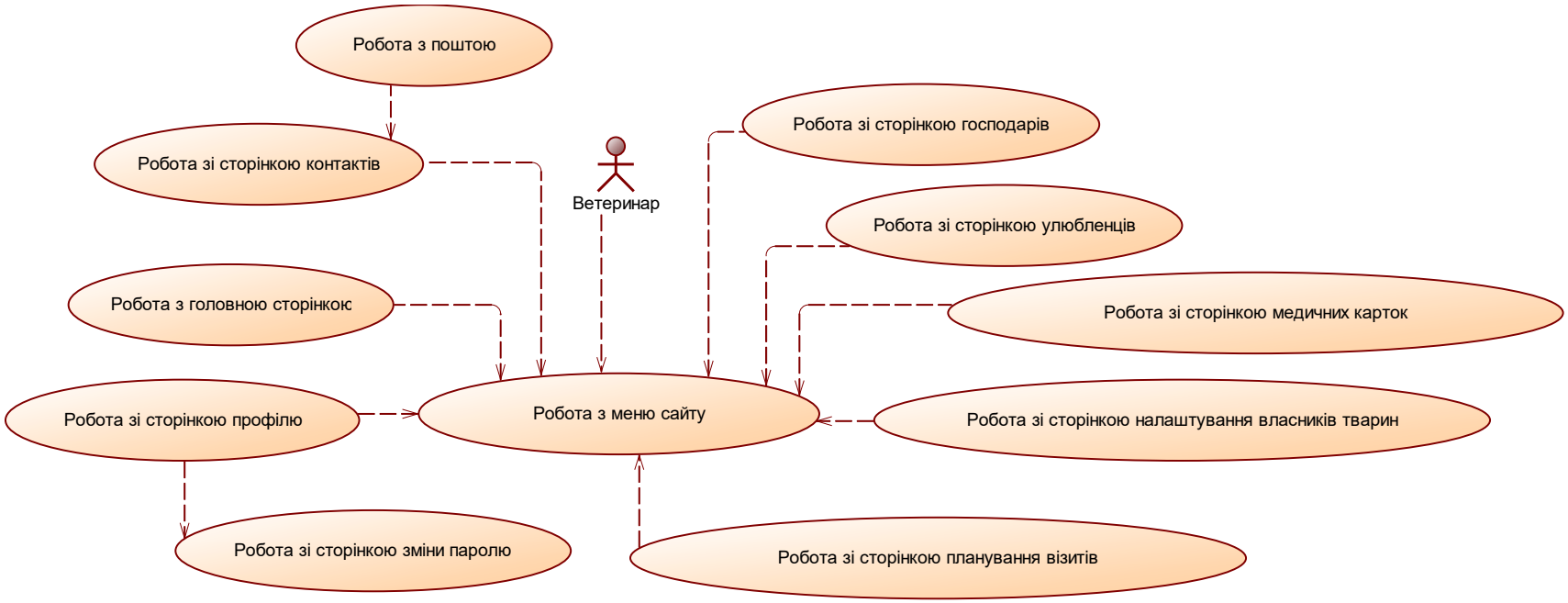

Рисунок 2.7 – Діаграма прецедентів авторизованого користувача

Висновки до розділу 2

Цей розділ містить опис функціональних вимог сайту, його експлуатаційне та функціональне призначення. Також в розділі відображено діаграми прецедентів, що описують всі можливості користувача в найбільш повному вигляді.

3 ВНУТРІШНЄ ПРОЄКТУВАННЯ

Проекти, які представляють собою сайти, що програмуються на мові C# з використанням фреймворку ASP.Net, здебільшого використовують архітектурний патерн MVC.

Існує три рівні проекту, кожен з яких виконує тільки свою функцію.

Представлення (View) – відображають дані користувача, та забезпечують їх введення й за можливістю перевірку коректності введених даних.

Моделі (Model) – представляють собою контейнери даних, які використовуються для зв’язку представлення з контролером та контролера з базою даних.

Контролер (Controller) – виконують роль посередника між іншими рівнями проекту, вони призначені для обробки подій на сторінці користувача, відправки даних до бази даних, додаткові перевірки даних, також контролери можуть використовуватись для обмеження доступу користувача до певних функцій сайту. Завдяки вбудованим функціям C# Identity ASP.Net, програміст може легко обмежувати доступ до даних для користувачів, оскільки існують спеціальні атрибути доступу (наприклад атрибут [Authorize], що виконує перевірку, чи авторизувався користувач в системі, якщо він не авторизований, а метод має такий атрибут, то користувач буде повернений на сторінку вказану за замовченням).

Структуру проекту буде відображено за допомогою діаграми класів на рис. 3.1 – 3.9.

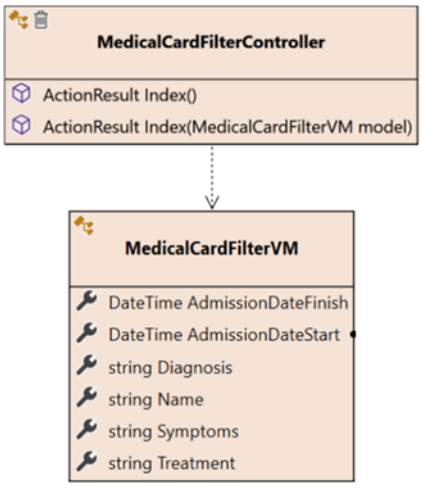

Рисунок 3.1 – Діаграма класів, зв’язок контролеру фільтрації даних медичних карт

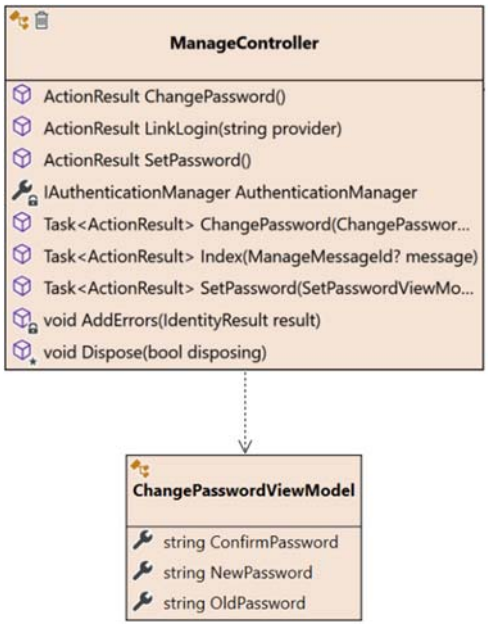

Рисунок 3.2 – Діаграма класів, зв’язки контролера роботи зі сторінкою зміни паролю

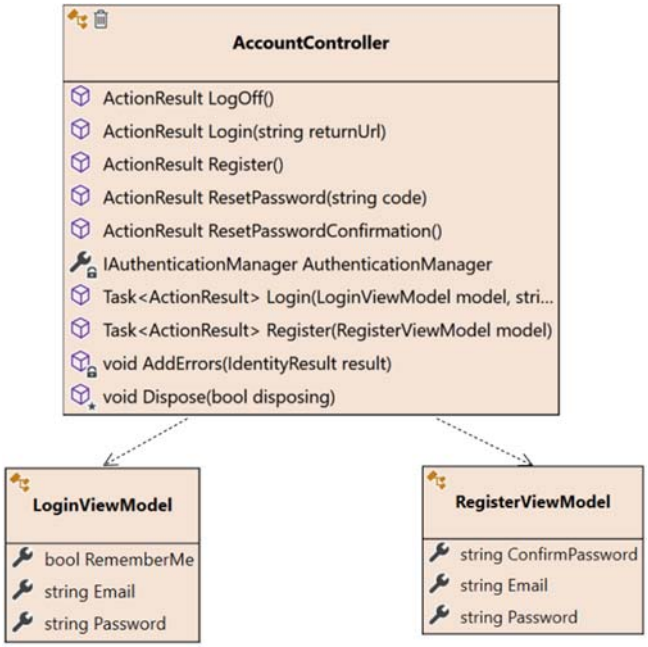

Рисунок 3.3 – Діаграма класів, зв’язки контролера роботи зі сторінками авторизації та реєстрації

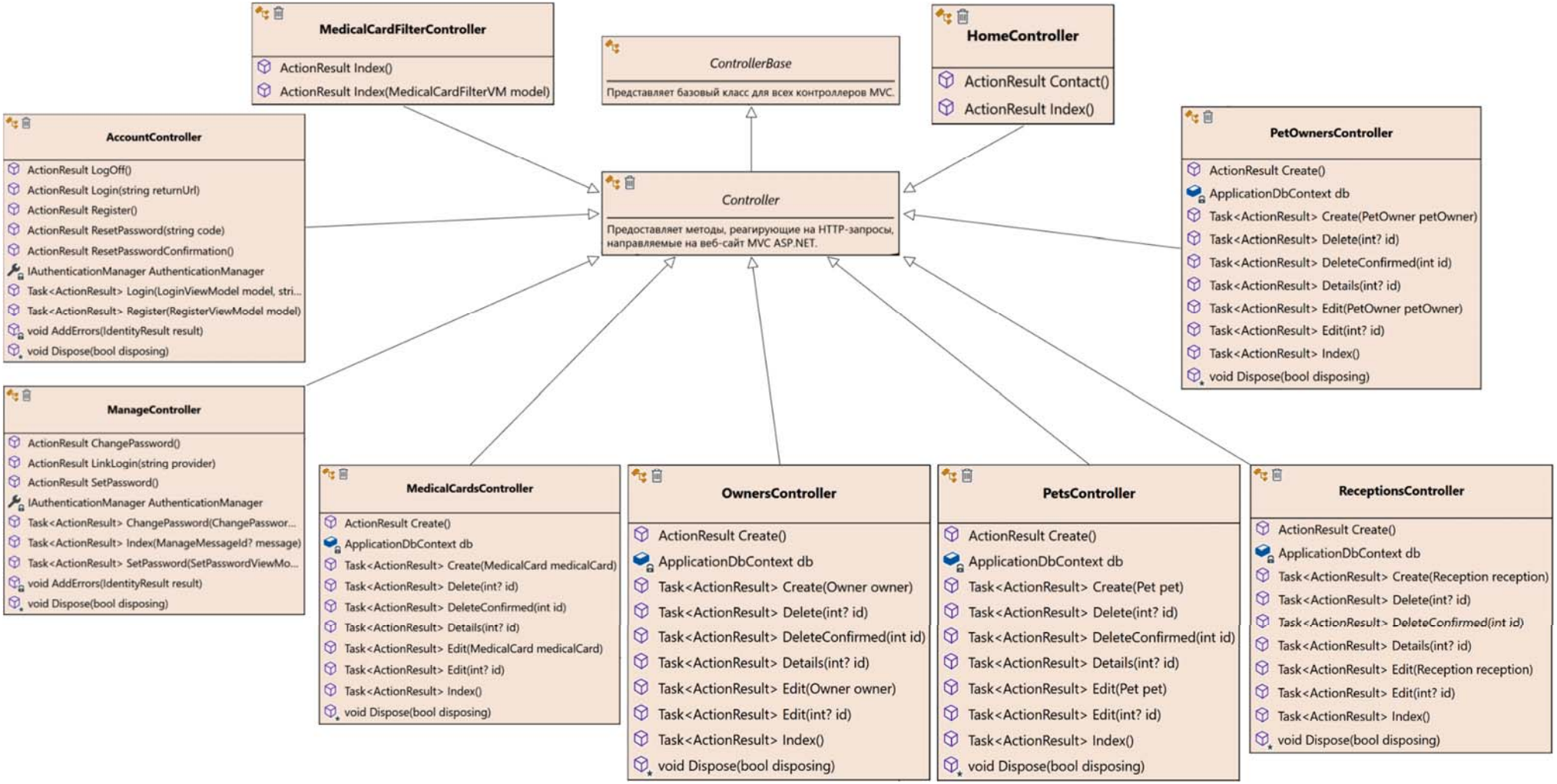

Рисунок 3.4 – Діаграма класів, зв’язки контролерів з їх базовим класом

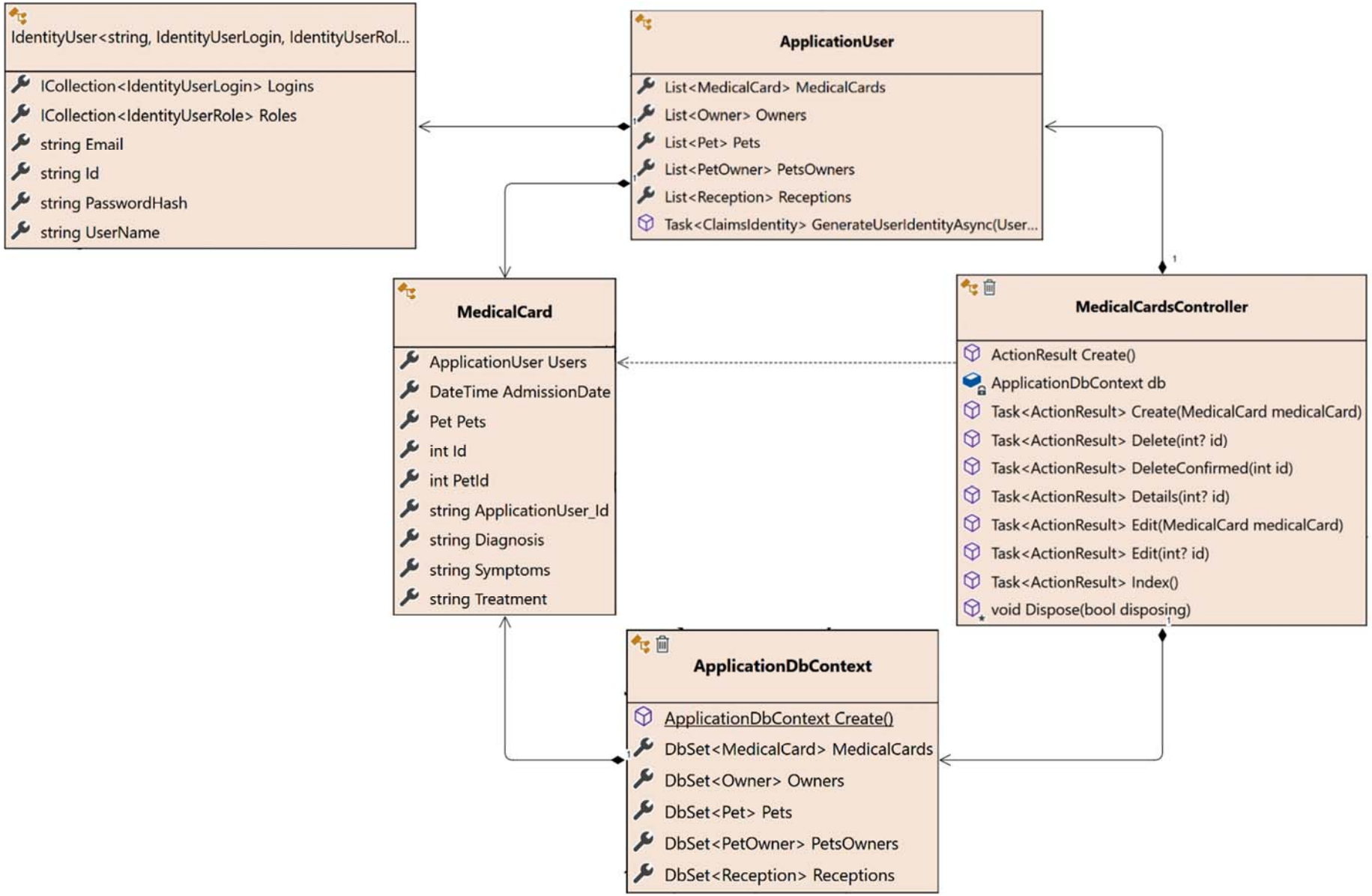

Рисунок 3.5 – Діаграма класів, зв’язки контролера роботи з даними медичних карт

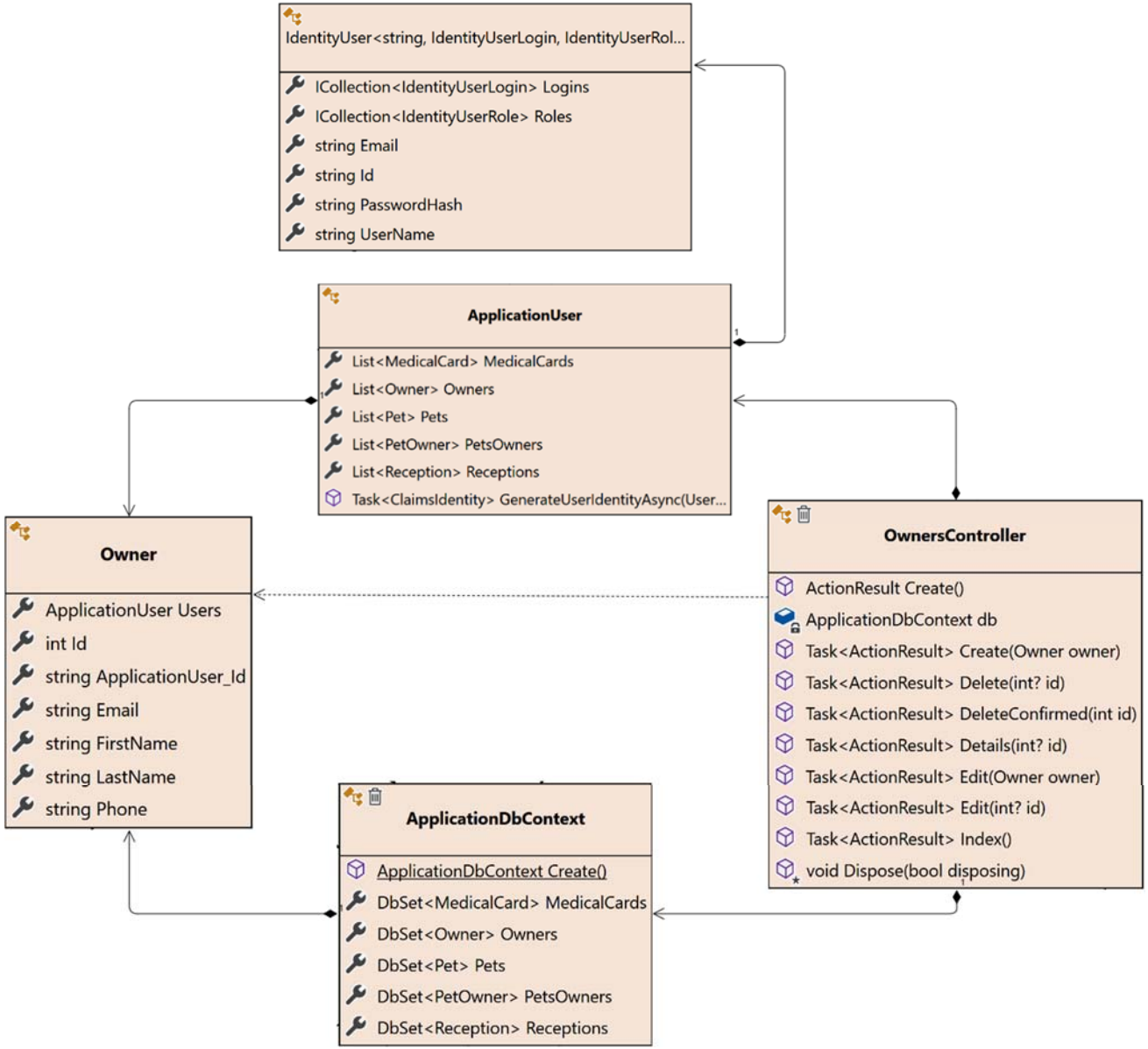

Рисунок 3.6 – Діаграма класів, зв’язки контролера для роботи з даними господарів

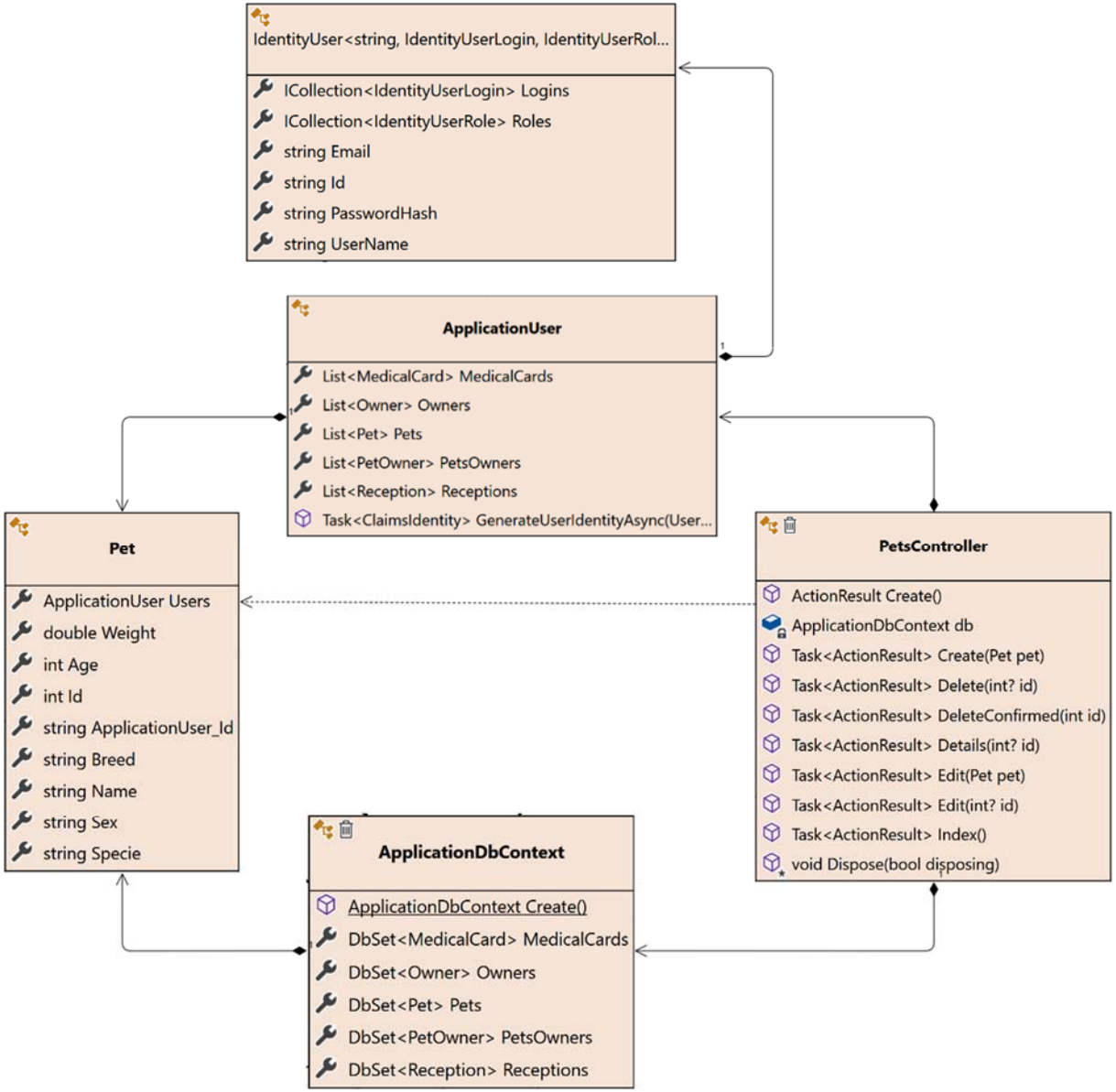

Рисунок 3.7 – Діаграма класів, зв’язки контролера для роботи з даними тварин

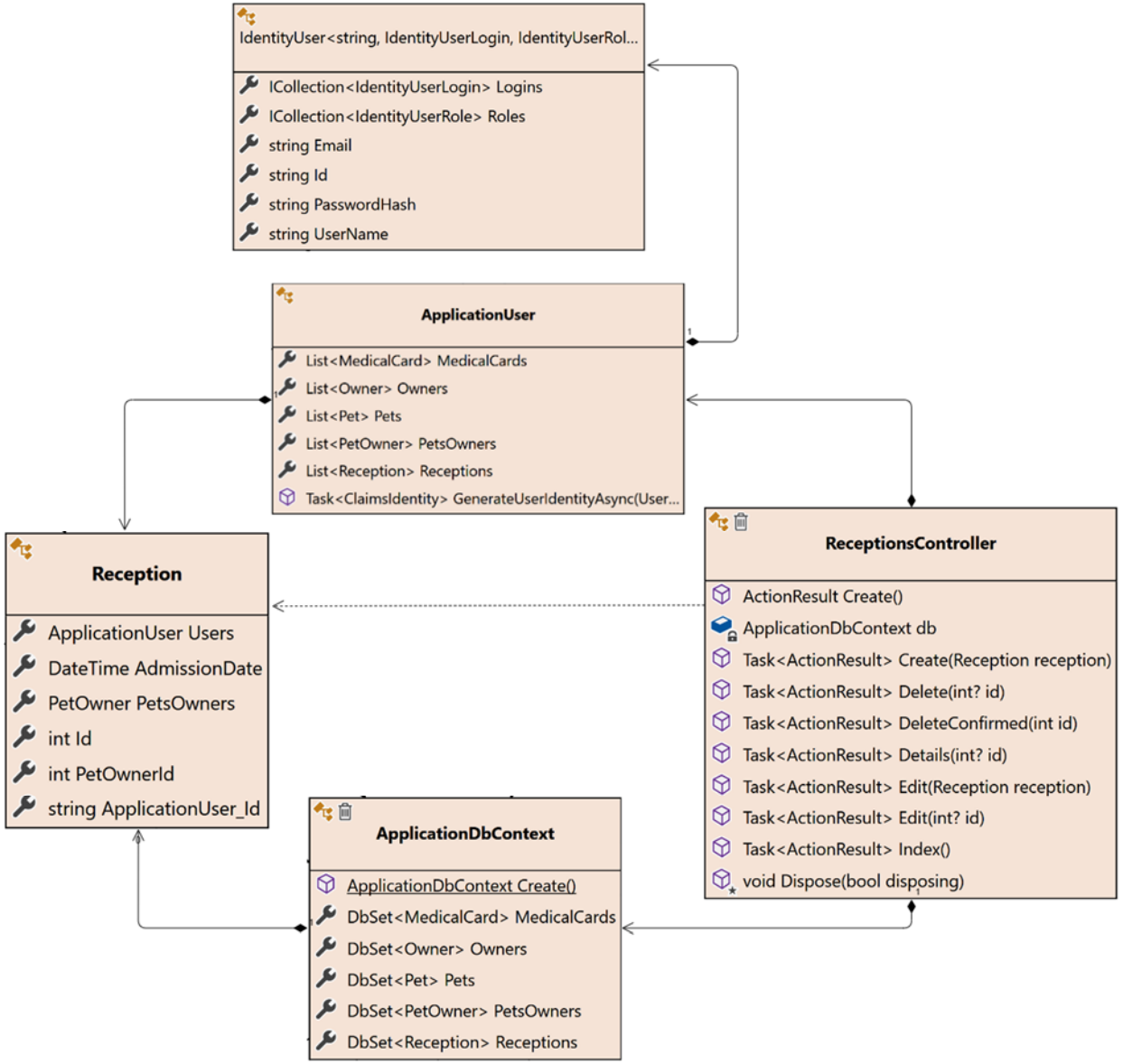

Рисунок 3.8 – Діаграма класів, зв’язки контролера для роботи з візитів

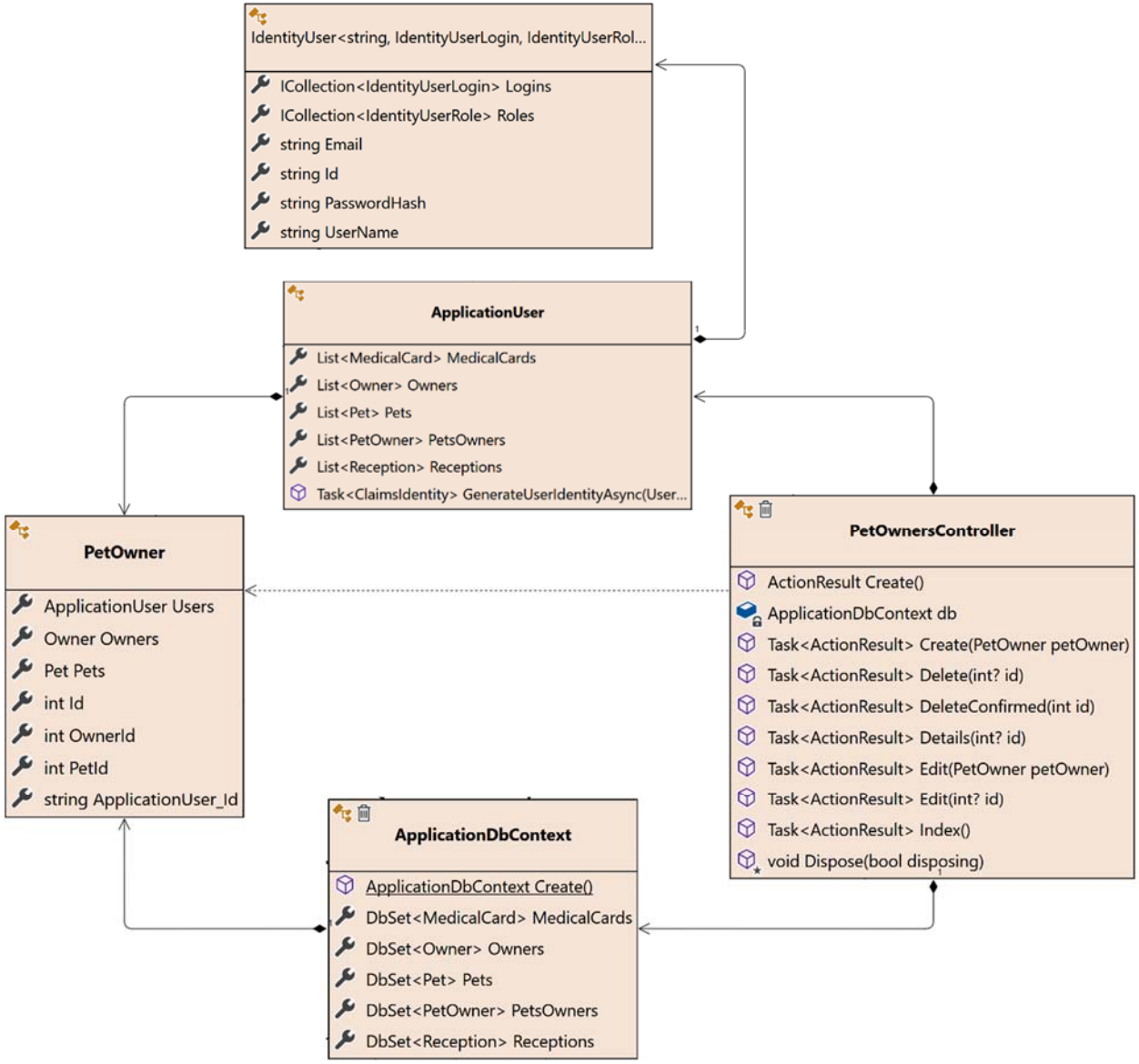

Рисунок 3.9 – Діаграма класів, зв’язки контролера для роботи з даними про зв'язок господарів з тваринами

Дизайн сайту буде представлений шаблонними кольорами, що пропонує Visual Studio 2019 та бібліотека bootstrap 4.

У табл. 3.1 буде відображено кольори для окремих елементів сайту.

Таблиця 3.1 – Кольори елементів сайту

Елемент	Номер кольору
Меню	#101010
Текст меню	#9d9d9d
Основний текст сторінки	#333333
Текст головного меню при виборі елемента меню	#ffffff
Посилання на пошту	#337ab7
Фон кнопок «обрання зони роботи» та «додання запису»	#5cb85c
Текст кнопок «обрання зони роботи» та «додання запису»	#fff
Фон кнопки «редагувати»	#337ab7
Фон кнопки «переглянути» та «фільтри»	#5bc0de
Фон кнопки «видалити»	#d9534f
Колір тексту для кнопок редагувати, переглянути, видалити, фільтри	#fff
Текст інших кнопки	#333
Фон інших кнопок	#fff
Кольори текстів повідомлень про неправильно заповнені поля	#d9534f

Діалог з користувачем буде відбуватись у вигляді екранних форм, оскільки це найбільш зручний вид діалогу для роботи з сайтом, що проектується.

Основними засобами привернення уваги на сайтах є їх кольорова палітра, тому кнопки, які відіграють важливу роль при роботі з програмою мають кольорову палітру, яка найбільш доречно відображає їх призначення.

Перелік обов’язкових браузерів на яких повинні правильно працювати усі функції сайту:

- Google Chrome v.98+;
- Microsoft Edge v.98+.

В табл. 3.2 представлені повідомлення системи, що можуть виникнути при роботі з нею.

Таблиця 3.2 – Повідомлення системи

Текст	Адресат	Ситуація	Рекомендовані дії
Невдала спроба увійти	Неавторизований користувач	Користувач ввів неправильні дані для входу до профілю	Перевірити правильність введених даних та спробувати увійти ще раз
Паролі не співпадають	Неавторизований / Авторизований користувач	Користувач ввів різні паролі у поля «пароль» та «підтвердити пароль»	Перевірити правильність паролів, ввести однакові дані в обох полях та спробувати ще раз
Поле Пароль є обов'язковим	Неавторизований користувач	Користувач не заповнив поле паролю	Заповнити поле паролю
Поле Адреса електронної пошти не є дійсною адресою електронної пошти	Неавторизований користувач	Користувач неправильно заповнив поле електронної пошти	Введена пошта не відповідає формату написання електронної пошти
Поле Адреса електронної пошти є обов'язковим	Неавторизований користувач	Користувач не заповнив поле електронної пошти	Заповнити поле
Значення Пароль має містити не менше 6 символів	Неавторизований / Авторизований користувач	Користувач неправильно заповнив поле паролю	Ввести новий пароль
Ім'я вже зайняте. Електронна пошта вже зайнята	Неавторизований користувач	Введена пошта вже зареєстрована в системі	Ввести нове ім'я
Паролі повинні містити принаймні один символ, який не є літерою або цифрою	Неавторизований / Авторизований користувач	Користувач неправильно заповнив поле паролю	Ввести новий пароль
Паролі повинні мати принаймні один символ верхнього регістру	Неавторизований / Авторизований користувач	Користувач неправильно заповнив поле паролю	Ввести новий пароль
Паролі повинні містити принаймні один символ, який не є літерою або цифрою	Неавторизований / Авторизований користувач	Користувач неправильно заповнив поле паролю	Ввести новий пароль

Для описання динаміки системи буде описано загальний принцип роботи системи на всіх її рівнях.

При переході на обрану користувачем сторінку, він викликає контролер, що повертає користувачу список моделей у представлення або саме представлення. Після відображення сторінки кожна подія, яку викликає користувач виконує запит до контролера, який реагує на неї. При необхідності контролер виконує запит до бази даних для формування вибірки за певними критеріями (наприклад показати лише ті записи, які відносяться до поточного користувача системи). База даних повертає колекцію записів, над якою контролер проводить певні маніпуляції, після чого повертає користувачу нове представлення. Цей принцип роботи притаманний всім модулям програми, тому детального опису вони не потребують.

Висновки до розділу 3

Розділ містить опис повідомлень системи для користувача в залежності від його рівня доступу й кольорову палітру сайту в залежності від призначення компонентів. Також в ньому пояснюється загальна динаміка роботи системи та відображено класову структуру проекту.

4 ПРОЄКТУВАННЯ АРХІТЕКТУРИ СИСТЕМИ

Фізичну структуру проекту буде відображено на діаграмах компонентів рис. 4.1 – 4.9.

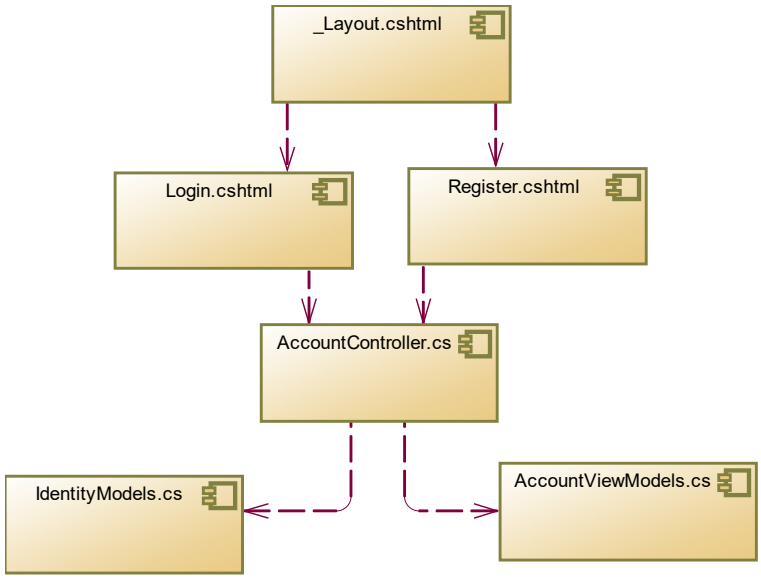

Рисунок 4.1 – Діаграма компонентів, залежності модулів відносно Account контролеру

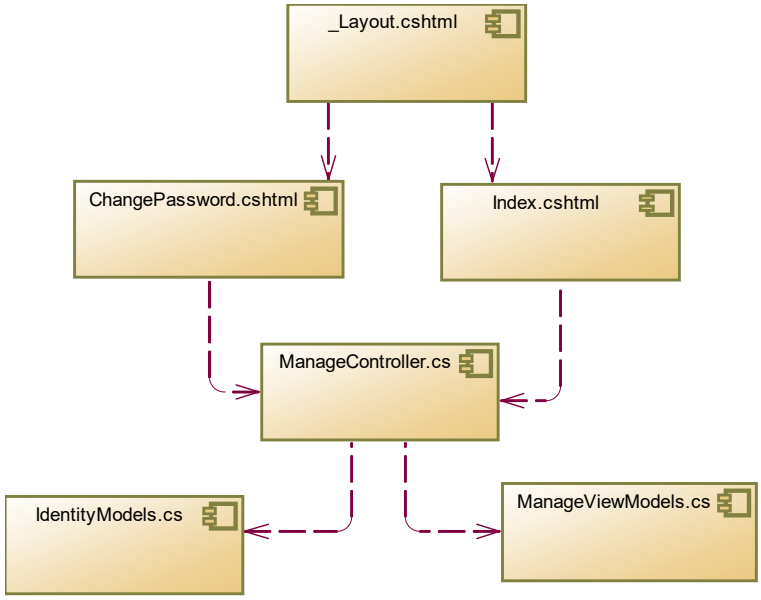

Рисунок 4.2 – Діаграма компонентів, залежності модулів відносно Manage контролеру

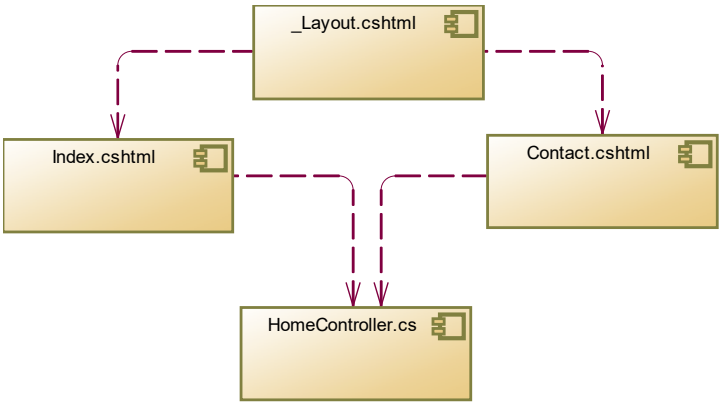

Рисунок 4.3 – Діаграма компонентів, залежності модулів відносно Home контролеру

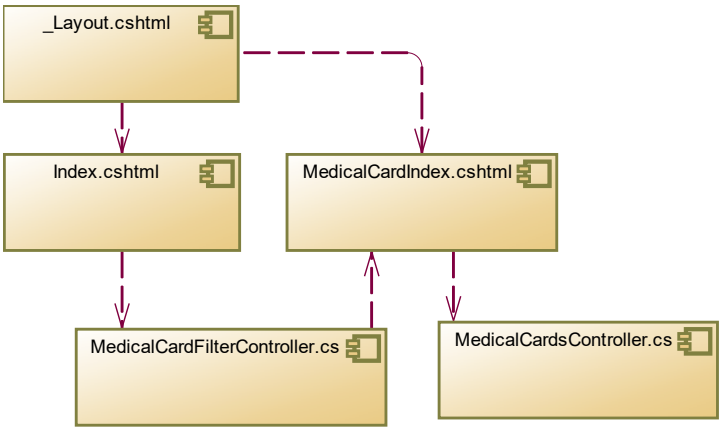

Рисунок 4.4 – Діаграма компонентів, залежності модулів відносно MedicalCardFilter контролеру

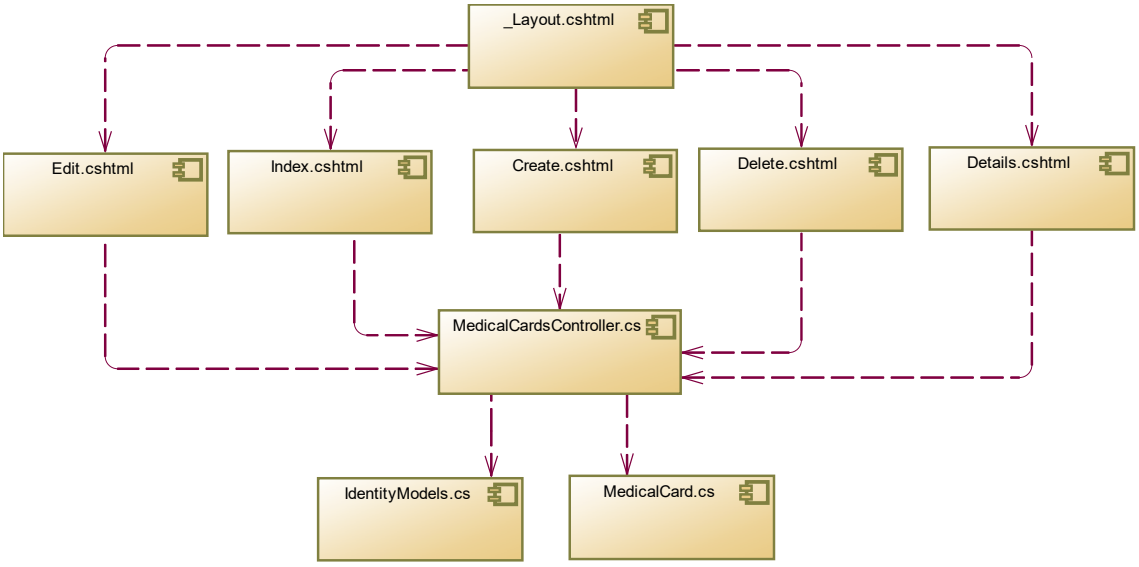

Рисунок 4.5 – Діаграма компонентів, залежності модулів відносно MedicalCards контролеру

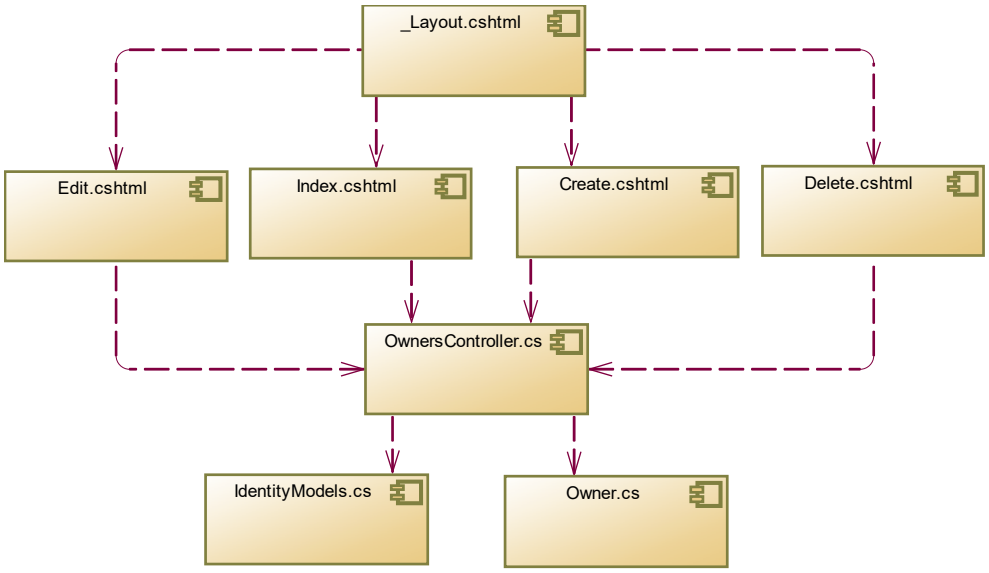

Рисунок 4.6 – Діаграма компонентів, залежності модулів відносно Owners контролеру

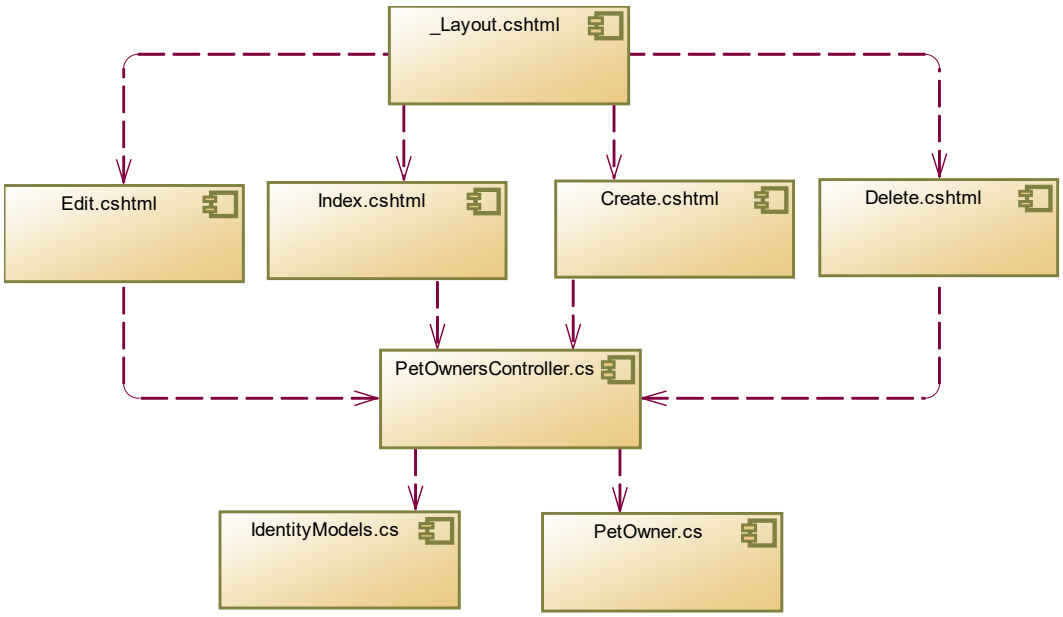

Рисунок 4.7 – Діаграма компонентів, залежності модулів відносно PetOwners контролеру

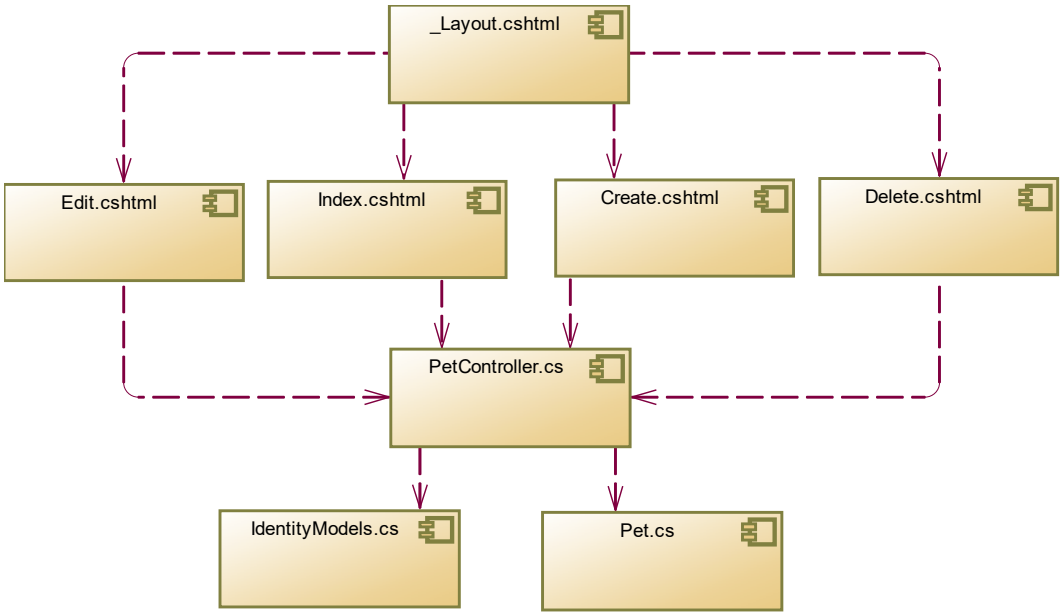

Рисунок 4.8 – Діаграма компонентів, залежності модулів відносно Pet контролеру

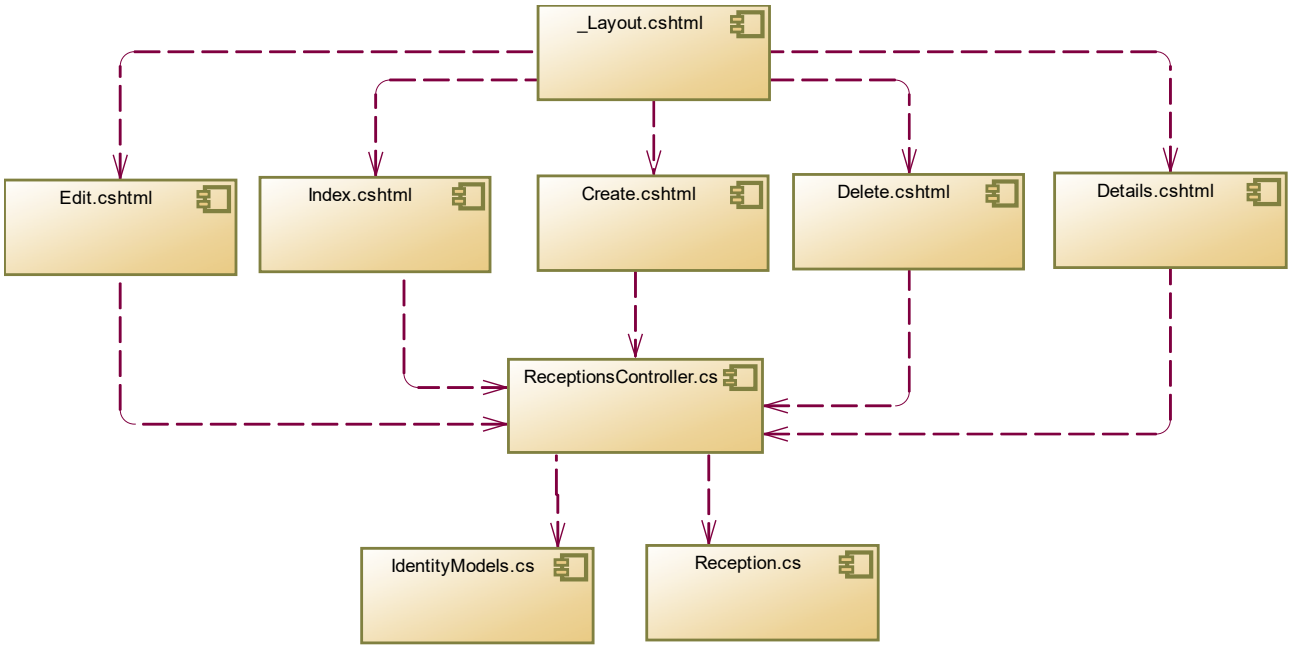

Рисунок 4.9 – Діаграма компонентів, залежності модулів відносно Receptions контролеру

Для опису сутностей, що будуть занесені до бази даних виконано побудову фізичної ER-моделі вказаної на рис. 4.10. Додатково варто зазначити, що дані про користувача заповнює сама Visual Studio й для збереження цілісності проекту зміни до таблиці AspNetUsers не вносились.

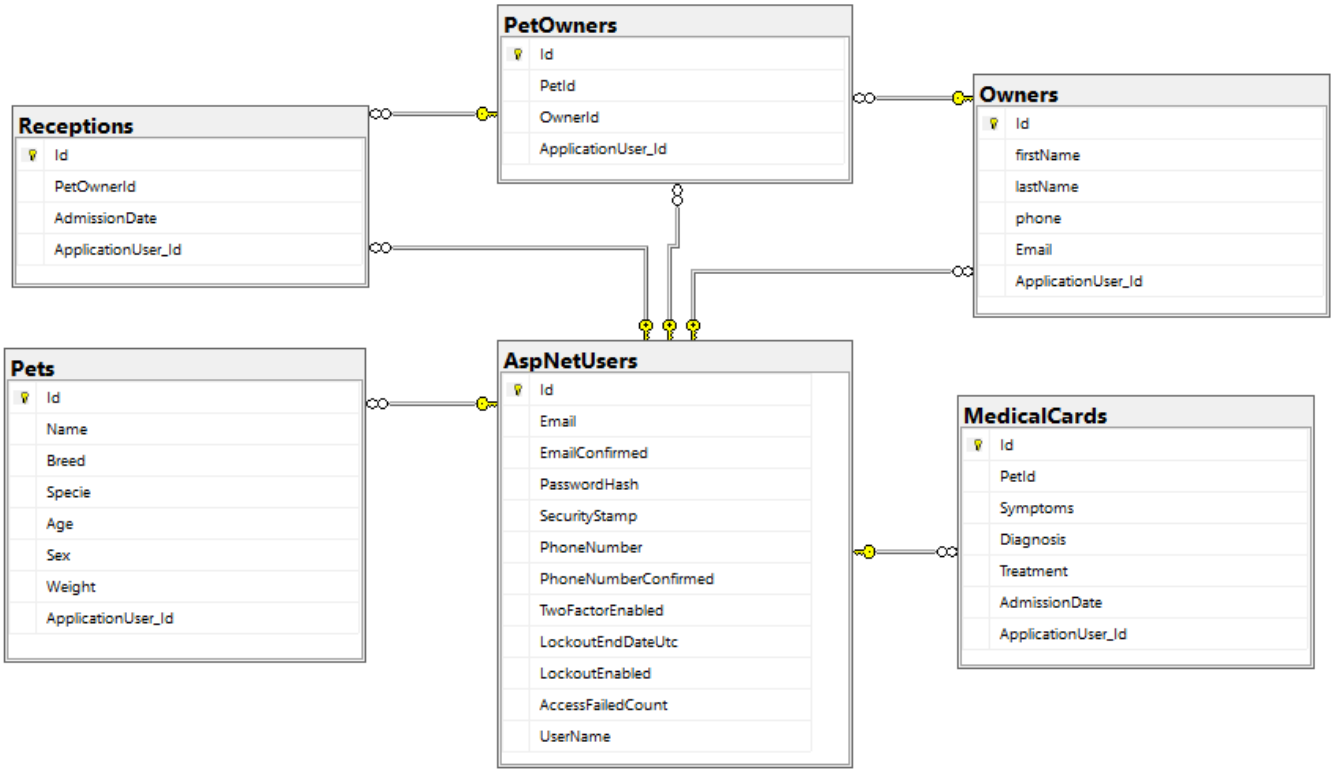

Рисунок 4.10 – ER-модель предметної області

Завдяки потужним можливостям Visual Studio було використано механізм міграцій, який дозволяє не писати самостійно запити на формування бази даних, а лише створити відповідні моделі цих таблиць. Для міграції таблиць до бази даних було використано команди: Enable-Migrations, Add-Migration «Назва міграції» та Update-database.

Згідно створеної ER-моделі були сформовані моделі відношень предметної області, які надалі були занесені до бази даних через механізм міграцій (табл. 4.1 – 4.6).

Таблиця 4.1 – Відношення Owners

№	Стовпець	Опис поля	Тип даних	Розмір	Ключ
1.	Id	Унікальний код	int identity	4	+
2.	firstName	Імя	ntext	2147483646	-
3.	lastName	Прізвище	ntext	2147483646	-
4.	phone	Телефон	ntext	2147483646	-
5.	Email	Електронна пошта	ntext	2147483646	-
6.	Application User_Id	Унікальний код власника запису (користувача)	nvarchar	256	-

Таблиця 4.2 – Відношення AspNetUsers

№	Стовпець	Опис поля	Тип даних	Розмір	Ключ
1.	Id	Унікальний код	nvarchar	256	+
2.	Email	Електронна пошта	nvarchar	512	-
3.	EmailConfirmed	Статус підтвердження пошти	bit	1	-
4.	PasswordHash	Пароль	ntext	2147483646	-
5.	SecurityStamp	Поточний стан користувача для автоматичного входу у систему через сторонні сервіси	ntext	2147483646	-
6.	PhoneNumber	Телефон	ntext	2147483646	-
7.	PhoneNumberConfirmed	Статус підтвердження телефону	bit	1	-

Таблиця 4.3 – Відношення MedicalCards

№	Стовпець	Опис поля	Тип даних	Розмір	Ключ
1.	Id	Унікальний код	int identity	4	+
2.	PetId	Код улюбленця	int	4	-
3.	Symptoms	Опис симптомів	ntext	2147483646	-
4.	Diagnosis	Опис діагнозів	ntext	2147483646	-
5.	Treatment	Опис лікування	ntext	2147483646	-
6.	AdmissionDate	Дата візиту	datetime	16	-
7.	ApplicationUser_Id	Унікальний код власника запису (користувача)	nvarchar	256	-

Таблиця 4.4 – Відношення PetOwners

№	Стовпець	Опис поля	Тип даних	Розмір	Ключ
1.	Id	Унікальний код	int identity	4	+
2.	PetId	Код улюбленця	int	4	-
3.	OwnerId	Код власника	int	4	-
4.	ApplicationUser_Id	Унікальний код власника запису (користувача)	nvarchar	256	-

Таблиця 4.5 – Відношення Receptions

№	Стовпець	Опис поля	Тип даних	Розмір	Ключ
1.	Id	Унікальний код	int identity	4	+
2.	PetOwnerId	Код зв’язку улюбленця та власника	int	4	-
3.	Admission Date	Дата візиту	datetime	16	-
4.	Application User_Id	Унікальний код користувача (власника запису)	nvarchar	256	-

Таблиця 4.6 – Відношення Pets

№	Стовпець	Опис поля	Тип даних	Розмір	Ключ
1.	Id	Унікальний код	int identity	4	+
2.	Name	Кличка	ntext	2147483646	-
3.	Breed	Порода	ntext	2147483646	-
4.	Specie	Вид	ntext	2147483646	-
5.	Age	Вік	int	4	-
6.	Sex	Стать	ntext	2147483646	-
7.	Weight	Вага	float	8	-
8.	Application User_Id	Унікальний код користувача (власника запису)	nvarchar	256	-

Висновки до розділу 4

Розділ описує компоненту структуру проекту, структуру базу даних та відношення, шо в ній знаходяться.

Для максимально точного опису компонентів проекту було використано діаграми компонентів, які найбільш повно відображають всі зв’язки компонентів між собою.

Також у розділі було описано ER-модель предметної області та сформовані відношення бази даних.

У відношення було описано призначення полів та їх типи даних й розміри.

5 РОЗРОБКА ПРОГРАМИ

Існує багато мов для розробки сайтів, кожна з них має свої переваги та недоліки. Деякі мови використовуються для створення серверної частини сайту через простоту розробки на них в цій галузі або через швидкість обробки даних, інші ж мови використовуються для створення інтерфейсу сайту. Існують також універсальні мови такі як JavaScript та C#. Перевагами цих мов є те, що вони мають багато готових рішень серед бібліотек для роботи з сайтом.

Аналізуючи можливості JavaScript можливо сказати, що ця мова потребує постійного контролю даних, й більшість готових рішень вимагають додаткових перевірок, при програмуванні.

На відміну від JavaScript мова C# має здебільшого визначенні стандартні типи та рішення, що вбудовані в середовище розробки Visual Studio. Варто також зазначити, що мова широко використовується в продуктах Microsoft, тому якість бібліотек написаних для неї надає програмісту впевненості у справності коду, який він використовуватиме.

Варто зазначити, що готові рішення на мові C# для фреймворку ASP.Net дозволять легко розширювати проекти, коли постає проблема їх росту. Так, наприклад, якщо користувачу необхідно буде авторизуватись за допомогою інших соціальних мереж, фреймворк ASP.Net Identity дозволяє це зробити в лічені хвилини.

Зважаючи на переваги та недоліки, описані для кожної з зазначених мов, мовою розробки було обрано C#.

Використовуючи метод покрокової деталізації зверху вниз, поетапно створимо структуру програми в залежності від необхідних дій роботи сайту.

1) Авторизований користувач.

1.1) Меню.

1.1.1) Домашня сторінка.

1.1.1.1) Інформація про сайт.

1.1.2) Сторінка контактів.

1.1.2.1) Телефон та електронна пошта.

1.1.3) Робочі зони.

1.1.3.1) Господарі.

1.1.3.2) Улюбленці.

1.1.3.3) Приналежність улюбленців господарям.

1.1.3.4) Медичні карти.

1.1.3.5) Візити.

1.1.4) Сторінка профілю.

1.1.4.1) Зміна паролю.

1.1.5) Вихід з системи.

1.2) Робота з даними тварин.

1.2.1) Додання.

1.2.2) Редагування.

1.2.3) Видалення.

1.3) Робота з даними господарів.

1.3.1) Додання.

1.3.2) Редагування.

1.3.3) Видалення.

1.4) Налаштування зв'язку господаря з тваринами.

1.4.1) Додання.

1.4.2) Редагування.

1.4.3) Видалення.

1.5) Робота з медичними картами.

1.5.1) Додання.

1.5.2) Редагування.

1.5.3) Перегляд.

1.5.4) Фільтрація.

1.5.4.1) за кличкою.

1.5.4.2) за симптомами.

1.5.4.3) за діагнозами.

1.5.4.4) за лікуванням.

1.5.4.5) за датою візиту.

1.5.4.5.1) Діапазон дат.

1.5.4.5.1.1) Початкова дата.

1.5.4.5.1.2) Кінцева дата.

1.5.5) Видалення.

1.6) Робота з плануванням візитів.

1.6.1) Додання.

1.6.1.1) Забезпечити неможливість встановити дату та час, що вже минули.

1.6.2) Редагування.

1.6.2.1) Забезпечити неможливість встановити дату та час, що вже минули.

1.6.3) Перегляд.

1.6.3.1) Перегляд візитів починаючи з поточної дати.

1.6.4) Видалення.

2) Неавторизований користувач.

2.1) Авторизація.

2.2) Реєстрація.

2.3) Домашня сторінка.

2.3.1) Інформація про сайт.

2.4) Сторінка контактів.

2.4.1) Телефон та електронна пошта.

Опису алгоритму програми можливо звести до виконання таких дій:

- користувач викликає подію на сторінці;
- подія сторінки викликає обробку цієї події контролером;
- контролер аналізує отриману модель або видає модель в залежності від типу методу (get – отримати дані, post – відправити);
- аналізуючи модель або формуючи відповідь на подію контролер зв’язується з базою даних, отримує необхідні дані, або відправляє їх;
- по завершенню роботи з даними, контролер повертає відповідь представленню у вигляді посилання на іншу подію, або заповненого списку моделей (моделі).

Висновки до розділу 5

Розділ містить опис рівнів деталізації роботи сайту, які можуть доповнюватись іншими етапами на наступних ітераціях проекту. Так, наприклад, на наступних ітераціях можуть додатись додаткові елементи фільтрації даних. Також розділ містить обґрунтування вибору мови програмування C# порівняно до мови JavaScript.

6 ТЕСТУВАННЯ ТА НАЛАГОДЖЕННЯ

Серед безлічі методів тестування найбільш простими в розумінні є методи білої та чорної скриньки.

Оскільки код проекту складається з простих операцій, що здебільшого не містять циклів та складних розгалужень, помилку методами білої скрині буде вкрай важко віднайти, тому цей спосіб тестування буде недоцільним. Отже, якщо метод білої скриньки не дасть змогу найбільш повно відшукати помилку, буде використано методи чорної скриньки.

Серед методів чорної скриньки було обрано метод граничних умов, оскільки в програмі існують такі поля як дата та час візиту, вік тварини, вага тварини, що потребують перевірки на своїх граничних значеннях.

Також для виявлення помилок буде наведено припущення, внаслідок яких дій користувача могла б виникнути помилка роботи програми.

Було розроблено такі граничні умови:

1) поле вік тварини: менше 0, 0 та більше 0;
2) поле вага: менше 0, 0 та більше 0;
3) поле дати та часу візиту при плануванні візитів: менше поточних дати та часу, рівні поточній даті та часу, більше за поточну дату та час.

Припущення про помилку:

1) що буде при введені некоректних даних у поле для вводу?
2) що буде якщо в числові поля ввести літери?
3) чи зберігається стан текстових полів фільтрації після виконання пошуку?
4) що буде якщо користувач буде додавати записи, які пов’язані з господарем чи твариною, в яких ці поля будуть порожні?
5) чи може користувач звернутись до запису, який йому недоступний при його редагуванні?

Виконавши всі зазначені тести було знайдено помилки, дані про які записані до табл. 6.1.

Зазначимо тести, що були пройдені успішно:

- тести граничних умов під номером 1 та 2. Помилка не виникла при введені даних 0 та більше 0;
- тести граничних умов під номером 3. Всі тести було пройдено успішно;
- припущення про помилку 1 та 2 – було виведено відповідне повідомлення;
- припущення про помилку 3 – дані зберігаються;
- припущення про помилку 5 – користувача переадресує на сторінку «404»;

Таблиця 6.1 – Протокол налагодження програми

Опис помилки	Опис ситуації	Способи усунення	Дії, що були застосовані для усунення
Некоректно працює збереження запису	Поле вік тварини менше 0	Додати анотацію до моделі, або створити перевірку в контролері	Було додано анотацію в моделі. Range(1, int.MaxValue, ErrorMessage = " Значення повинно бути 0 або більше ")
Некоректно працює збереження запису	Поле вага менше 0	Додати анотацію до моделі, або створити перевірку в контролері	Було додано анотацію в моделі. Range(1, int.MaxValue, ErrorMessage = "Значення повинно бути 0 або більше")
Некоректно працює збереження запису	Користувач при роботі з плануванням візитів додав візит в якому відсутні дані тварини та її господаря	Перевірити подію збереження даних та додати умову на перевірку введеної інформації, або додати анотацію «Required» для відповідного поля	Було додано перевірку введених даних

Висновки до розділу 6

В розділі було обґрунтовано обрання методів тестування програми, виконано опис можливих ситуацій де може виникнути помилка, а також наведено протокол налагодження програми. Найбільш вагомою помилкою було створення записів з порожніми даними. Тому було проведено перевірку всіх контролерів на наявність такої помилки й виявлено та виправлено її у контролері медичних карток та приналежності тварин господарям.

ВИСНОВКИ ТА РЕКОМЕНДАЦІЇ

Виконуючи дипломну роботу було проаналізовано інформацію про розробку сайтів з використанням різних мов програмування та патернів проектування. Також було проаналізовано існуючі бібліотеки та фреймворки.

На основі отриманої інформації було обрано мову C#, яка надає найбільше документації, щодо роботи з фреймворками, що використовуються при розробці сайтів. Також дана мова містить багато готових рішень, які пришвидшують розробку сайтів не корпоративної архітектури. Окремо варто відзначити фреймоврк, що забезпечує доступ користувача до ресурсів сайту – ASP. Net Identity.

Мова програмування C# також надає потужні можливості для публікації розробленого сайту в інтернеті, публікація проекту виконується за лічені хвилини, необхідно лише взяти файл профілю, ввести логін та пароль від хостингу й сайт опублікований. Варто також зазначити простоту створення таблиць бази даних за допомогою вбудованої функції міграції даних мови C#. Міграції надають можливість швидко створити всі необхідні таблиці в пустій базі даних всього лише запустивши сайт.

Результатом виконання дипломної роботи є сайт для організації особистого кабінету лікаря ветеринарної клініки й ведення обліку інформації про пацієнтів.

Аналізуючи зручність використання сайту та його можливості рекомендовано на наступних ітераціях проекту додати фільтрацію даних для всіх зон роботи ветеринара, також рекомендується додати можливість сортування даних.

Додатково рекомендовано поєднати зони ведення інформації про улюбленця та приналежності улюбленця до його господаря. Також для зручності можливо створити список улюбленців, поруч з якими буде перехід на набір їх медичних візитів.

Всі рекомендації загалом направленні на покращення зручності користування сайтом.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Fagerberg, J. ASP.NET MVC 5 - Building a Website with Visual Studio 2015 and C Sharp: The Tactical Guidebook. – NY.: Springer Science, 2016. – 492 c.
2. Esposito, D. Programming Microsoft ASP.NET MVC (Developer Reference) 3rd Edition. – NY.: Microsoft Press, 2014. – 528 c.
3. Freeman. A. Pro ASP.NET MVC 5 (Expert's Voice in ASP.Net). – NY.: Apress, 2014. – 832 c.
4. Galloway, J. Professional ASP.NET MVC 5. – NY.: Wiley, 2014. – 624 c.

ДОДАТКИ

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАТВЕРДЖУЮ
Перший проректор Українського державного університету науки і технологій
Анатолій РАДКЕВИЧ
18.02.22

САЙТ ВЕТЕРИНАРНОЇ КЛІНІКИ

Технічне завдання
ЛИСТ ЗАТВЕРДЖЕННЯ
44165850.01254-01-ЛЗ

Представники
підприємства-розробника
Завідувач кафедри КІТ
Вадим ГОРЯЧКІН
18.02.22

Керівник розробки
Олександр ІВАНОВ
18.02.22

Виконавець
Микита КРОЛЬ
18.02.22

Нормоконтролер
Олена КУРОП'ЯТНИК
18.02.22

ЗАТВЕРДЖЕНО
44165850.01254-01-ЛЗ

САЙТ ВЕТЕРИНАРНОЇ КЛІНІКИ

Технічне завдання

44165850.01254-01

Листів 16

2022

АНОТАЦІЯ

Документ 44165850.01254-01 «Розробка сайту ветеринарної клініки. Технічне завдання» входить до складу програмної документації на програму, яка представляє собою сайт особистого кабінету лікаря.

У даному документі представлені призначення та область застосування програмного продукту, основні вимоги, стадії та строки виконання проекту.

ЗМІСТ

ВСТУП

Веб-застосунок «Сайт ветеринарної клініки», що розробляється призначений для керування записами про пацієнтів ветеринарної клініки для окремо взятого лікаря.

Малі клініки, які не є частинами великих мереж, зазвичай не мають своїх сайтів, а подекуди навіть записують дані про відвідування пацієнтів та їх лікування в спеціальні журнали, розробка такого сайту дозволяє підвищити ефективність роботи таких закладів, оскільки програма надає широкі можливості по роботі з даними, наприклад, очевидною перевагою є швидкий пошук пацієнта та його історії лікування, який може виконуватись навіть за допомогою браузера.

Область застосувань: користувачі - лікарі клініки, які ведуть обстеження та облік пацієнтів. Лікарі зможуть заносити дані про обстеження пацієнта та матимуть можливість створювати записи про майбутні візити пацієнтів.

Експлуатаційне призначення – завдяки роботі на сайті користувачі (ветеринари), матимуть можливість зберігати свої дані в зручному вигляді, відпаде необхідність в постійному зберіганні даних на папері й з’явиться можливість швидкого пошуку необхідної інформації записаної до медичної картки пацієнта.

1 ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки є наказ «Про призначення керівників та затвердження бакалаврських робіт» від 08.12.2021 р., затверджений ректором Українського державного університету науки і технологій проф. Пшіньком О. М.

Тема дипломної роботи – «Розробка сайту ветеринарної клініки». Керівник – доцент Іванов О. П.

2 ПРИЗНАЧЕННЯ РОЗРОБКИ

Функціональне призначення – організація роботи ветеринарів за допомогою інтернет-сайту.

Експлуатаційне призначення – завдяки роботі на сайті ветеринари, мають можливість зберігати свої дані в зручному вигляді, документація перейде у електронний вигляд, з’явиться можливість швидкого пошуку необхідної інформації записаної до медичної картки пацієнта.

3 ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ

3.1 Вимоги до функціональних характеристик

Функціональне вимоги будуть описані у вигляді діаграм прецедентів (рис. 3.1 – 3.7).

Рисунок 3.1 – Діаграма прецедентів неавторизованого користувача

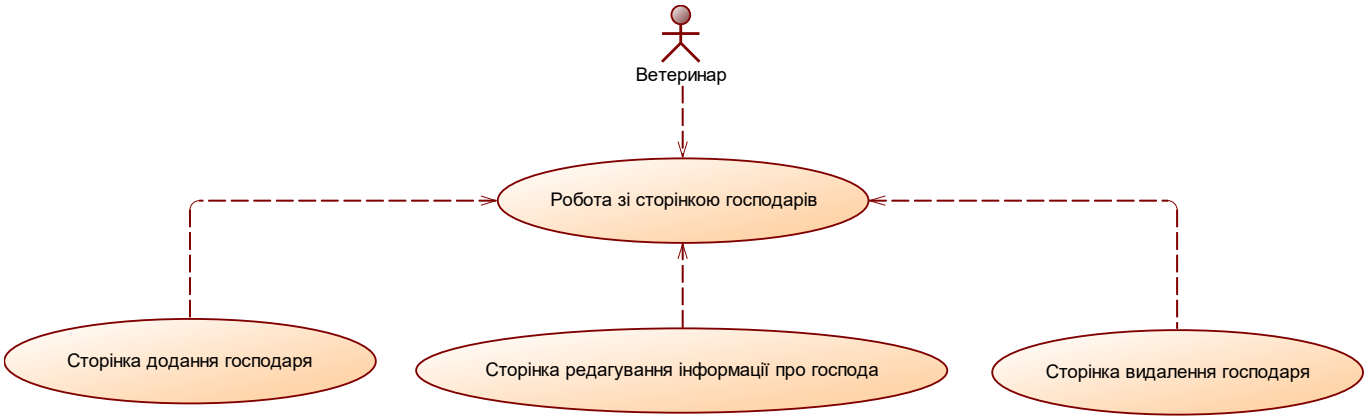

Рисунок 3.2 – Діаграма прецедентів для ветеринара при роботі зі сторінкою господарів

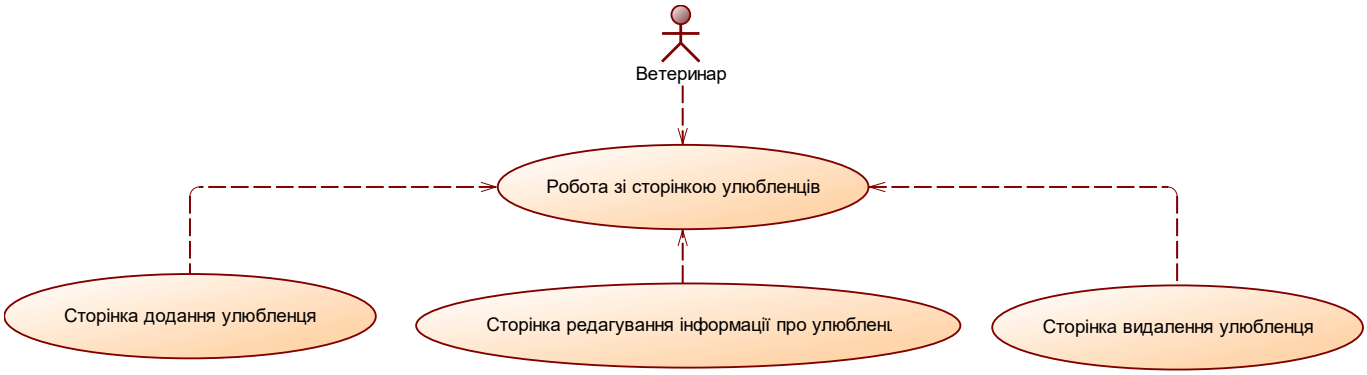

Рисунок 3.3 – Діаграма прецедентів для ветеринара при роботі зі сторінкою улюбленців

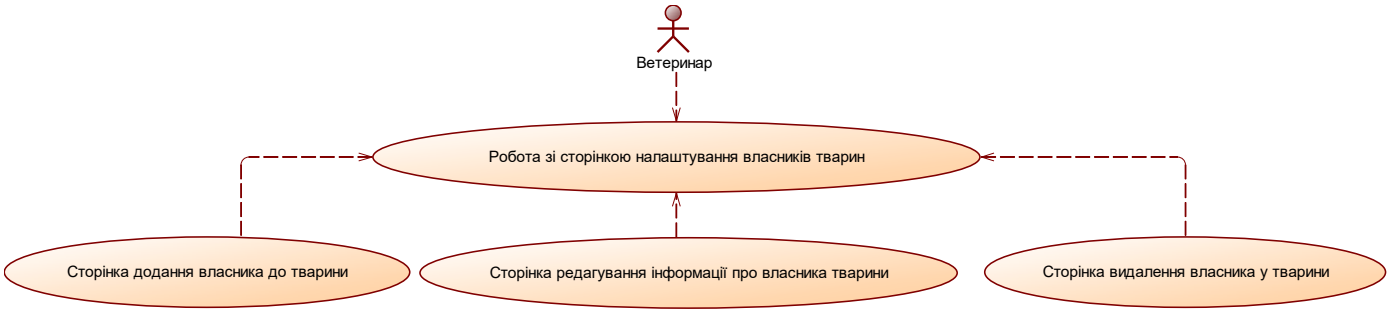

Рисунок 3.4 – Діаграма прецедентів для ветеринара при роботі зі сторінкою власників тварин

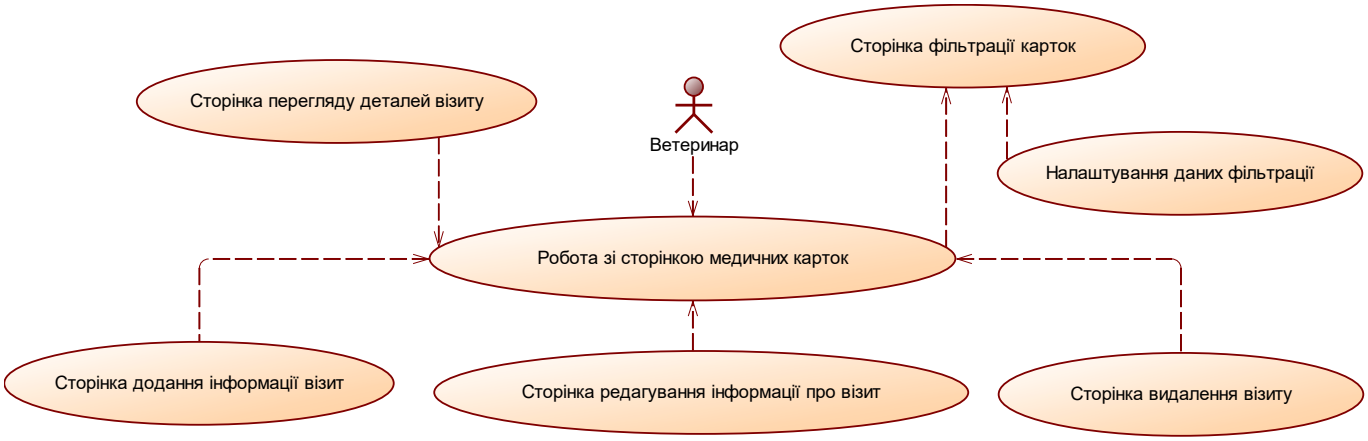

Рисунок 3.5 – Діаграма прецедентів для ветеринара при роботі зі сторінкою медичних карток

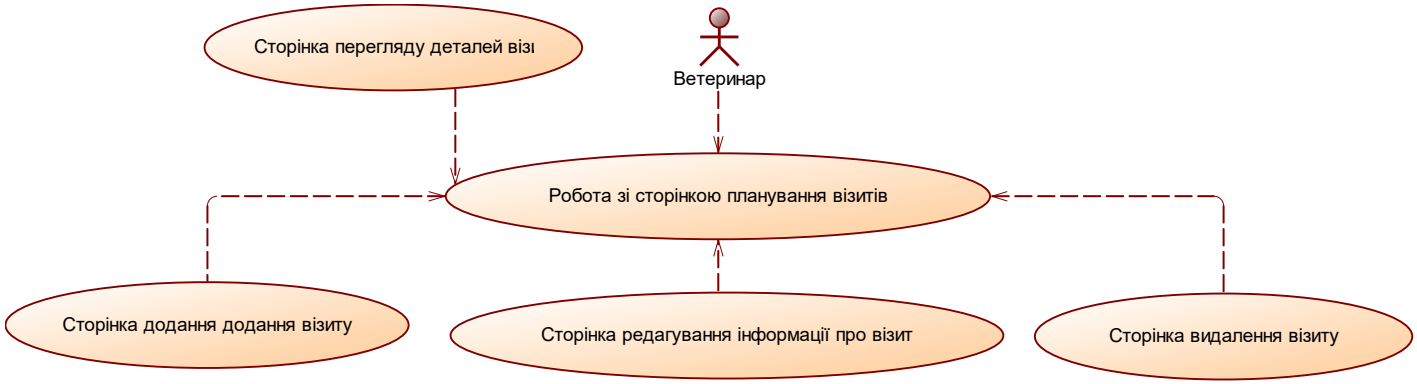

Рисунок 3.6 – Діаграма прецедентів для ветеринара при роботі зі сторінкою планування візитів

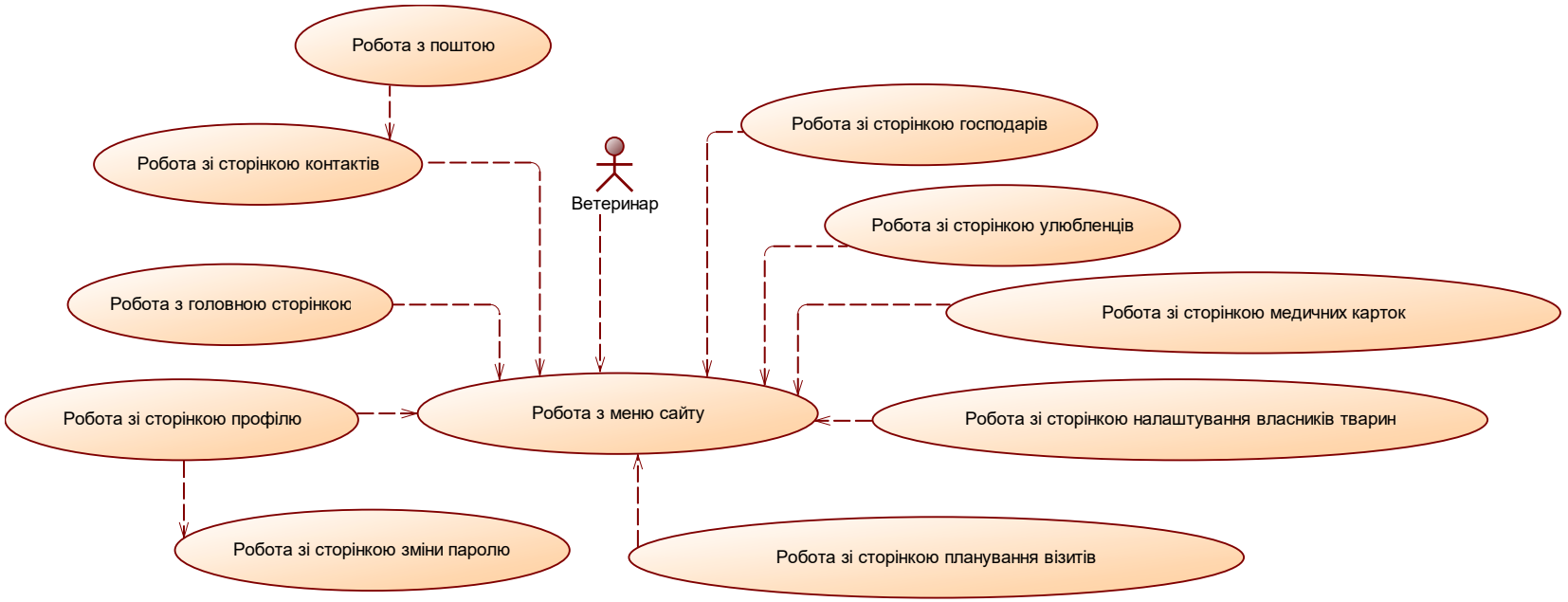

Рисунок 3.7 – Діаграма прецедентів авторизованого користувача

3.2 Вимоги до надійності

Вимоги до надійності програми:

- контроль введення користувачем інформації в інформаційних полях;
- хешування паролю користувача використовуючи стандартний метод фреймворку ASP.Net Identity;
- відгук про стан програми після виконання певних дій на сайті;
- наявність архівної копії тексту програми;
- наявність резервної копії бази даних на зовнішньому носії (сервері де розміщена база даних).

3.3 Умови експлуатації

Програмний продукт повинен використовуватись у приміщеннях, які відповідають умовам роботи ЕОМ, а саме мають такі кліматичні, санітарні та гігієнічні умови, які відповідають ДНАОП 0.00-1.31-99 (див. табл. 3.3.1).

Таблиця 3.3.1 – Кліматичні умови

Пора року	Категорія робіт згідно з ГОСТ 12.01-005-88	Температура повітря, град.С	Відносна вологість повітря, %	Швидкість руху повітря, м/с
		Оптимальна	Оптимальна	Оптимальна
Холодна	легка-1-а	22-24	40-60	0,1
	легка-1-б	21-23	40-60	0,1
Тепла	легка-1-а	23-25	40-60	0,1
	легка-1-б	22-24	40-60	0,2

Працювати з програмою може людина, що має навички роботи з персональним комп’ютером та ознайомлена з керівництвом користувача.

3.4 Вимоги до складу та параметрів технічних засобів

Розроблюваний програмний продукт повинен використовуватись на ЕОМ, що мають браузер.

Та мобільних пристроях, що мають:

- 2 гігабайти оперативної пам’яті;
- 16 гігабайт доступної пам’яті;
- операційну систему android v.10+.

Сервер повинен мати:

- 2 гігабайти оперативної пам’яті;
- процесор: 32 або 64 розрядний процесор із тактовою частотою 1 ГГц або вище з набором інструкцій SSE2;
- 100 гігабайт простору на жорсткому диску.

3.5 Вимоги до інформаційної та програмної сумісності

Програмний продукт розробляється для всіх видів операційних систем Android версії 10.0 та вище й для сімейства операційних систем Windows 95 та наступних версій. Середовище розробки – Visual Studio 2022. Система керування базами даних –MS SQL Server 2017 [1, 2], інтернет-браузер.

3.6 Вимоги до маркування і упаковки

Упаковка програмного продукту, включаючи документацію повинна бути захищена від пошкоджень різного роду (механічних, кліматичних). На упаковці повинно бути вказана назва продукту, номер версії (якщо вона змінювалась), мінімальні системні вимоги. На зворотній стороні упаковки вказується розробник та його юридична адреса.

На рис. 3.7 представлено приклад маркування.

Програмний додаток «САЙТ ВЕТЕРИНАРНОЇ КЛІНІКИ» Мінімальні системні вимоги: ЕОМ, що мають: – 4 гігабайти оперативної пам’яті. Мобільні пристрої, що мають: – 2 гігабайти оперативної пам’яті; – 16 гігабайт доступної пам’яті. Сервер повинен мати: – 2 гігабайти оперативної пам’яті; – 100 гігабайт простору на жорсткому диску.	Розробник: Кроль М. Є. Кафедра «КІТ», УДУНТ м. Дніпро, вул. Лазаряна 2 2022

Рисунок 3.7 – маркування

3.7 Вимоги до транспортування і зберігання

Транспортування повинне забезпечувати збереження програмного продукту, його цілісність і запобігання несанкціонованого доступу до нього. Програмний виріб міститься на оптичному носії даних типу CD-R і повинно мати відповідну упаковку для захисту від механічних ушкоджень та атмосферного впливу (пластиковий футляр або паперовий конверт).

4 ВИМОГИ ДО ПРОГРМАНОЇ ДОКУМЕНТАЦІЇ

Вся документація до програмного додатку повинна задовольняти вимоги до програмної документації.

До складу програмної документації має входити технічне завдання та робочий проект.

До складу робочого проекту мають входити:

- специфікація;
- текст програми;
- опис програми.

Вся документація до програми повинна задовольняти вимогам державного стандарту до оформлення програмних документів [1].

5 СТАДІЇ І ЕТАПИ РОЗРОБКИ

В табл. 5.1 приведені стадії та етапи розробки програмного продукту.

Таблиця 5.1 – Стадії та етапи розробки

Стадія	Зміст	Строки виконання
Технічне завдання	Постановка задачі, збір інформації, виріб та обґрунтування критеріїв розробки. Попередній вибір методів рішення задач. Визначення вимог до технічних засобів. Узгодження і затвердження технічного завдання.	31.01.22 - 20.02.22
Робочий проект	Програмування та відладка програми.	21.02.22 - 05.05.22
	Тестування програми	06.05.22 - 26.05.22
	Розробка, узгодження і затвердження програмної документації.	27.05.22 - 08.06.22

6 ПОРЯДОК КОНТРОЛЮ І ПРИЙМАННЯ

Контроль здійснюється за допомогою виконання набору тестів з метою знаходження помилок в програмному продукті та його специфікації. Контроль виконання роботи забезпечується головним керівником розробки.

Прийом програмного продукту здійснюється уповноваженою комісією.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Основи стандартизації програмних систем [Текст]: методичні вказівки до дипломного проектування та лабораторних робіт / уклад.: Ю.М.Івченко, В.І.Шинкаренко, В. Г. Івченко; Дніпропетр. нац. ун-т залізн. трансп. ім. акад. В.Лазаряна. – Д.: Вид-во Дніпропетр. нац. ун-ту залізн. трансп. ім. акад. В.Лазаряна, 2009. – 38 с.
2. Petkovic, D. Microsoft SQL Server 2019: A Beginner's Guide, Seventh Edition. – NY.: McGraw Hill, 2020. – 864 c.
3. Kellenberger, K. Beginning T-SQL: A Step-by-Step Approach. – NY.: Apress, 2020. – 918 c.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАТВЕРДЖУЮ
Перший проректор Українського
державного університету
науки і технологій
Анатолій РАДКЕВИЧ
10.06.2022

САЙТ ВЕТЕРИНАРНОЇ КЛІНІКИ

Специфікація
ЛИСТ ЗАТВЕРДЖЕННЯ
44165850.01254-01-ЛЗ

Представники
підприємства-розробника
Завідувач кафедри КІТ
Вадим ГОРЯЧКІН
10.06.2022

Керівник розробки
Олександр ІВАНОВ
10.06.2022

Виконавець
Микита КРОЛЬ
10.06.2022

Нормоконтроллер
Олена КУРОП’ЯТНИК
10.06.2022

2022

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАТВЕРДЖУЮ
Перший проректор Українського
державного університету
науки і технологій
Анатолій РАДКЕВИЧ
10.06.2022

САЙТ ВЕТЕРИНАРНОЇ КЛІНІКИ

Специфікація
ЛИСТ ЗАТВЕРДЖЕННЯ
44165850.01254-01-ЛЗ

Представники
підприємства-розробника
Завідувач кафедри КІТ
Вадим ГОРЯЧКІН
10.06.2022

Керівник розробки
Олександр ІВАНОВ
10.06.2022

Виконавець
Микита КРОЛЬ
10.06.2022

Нормоконтроллер
Олена КУРОП'ЯТНИК
10.06.2022

Позначення	Найменування	Примітка
	Документація	
44165850.01254-01-ЛЗ	Лист затвердження	
44165850.01254-01 12 01-ЛЗ	Лист затвердження	
44165850.01254-01 12 01	Текст програми	
44165850.01254-01 13 01-ЛЗ	Лист затвердження	
44165850.01254-01 13 01	Опис програми	

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАТВЕРДЖУЮ
Перший проректор Українського
державного університету
науки і технологій
Анатолій РАДКЕВИЧ
10.06.22

САЙТ ВЕТЕРИНАРНОЇ КЛІНІКИ

Текст програми
ЛИСТ ЗАТВЕРДЖЕННЯ
44165850.01254-01 12 01-ЛЗ

Представники
підприємства-розробника
Завідувач кафедри КІТ
Вадим ГОРЯЧКІН
10.06.22

Керівник розробки
Олександр ІВАНОВ
10.06.22

Виконавець
Микита КРОЛЬ
10.06.22

Нормоконтроллер
Олена КУРОП'ЯТНИК
10.06.22

ЗАТВЕРДЖЕНО
44165850.01254-01 12 01-ЛЗ

САЙТ ВЕТЕРИНАРНОЇ КЛІНІКИ

Текст програми

44165850.01254-01 12 01

Листів 22

2022

АНОТАЦІЯ

Документ 44165850.01254-01 12 01 «Розробка сайту ветеринарної клініки. Текст програми» входить до складу програмної документації на програму, яка представляє собою сайт особистого кабінету лікаря.

У даному документі представлений текст програми та схема взаємодії модулів програми. Для розробки проекту було обрано мову програмування C#.

ЗМІСТ

1 СХЕМА ВЗАЄМОДІЇ МОДУЛІВ

На рис. 1.1 – 1.9 приведена схема взаємодії модулів програми.

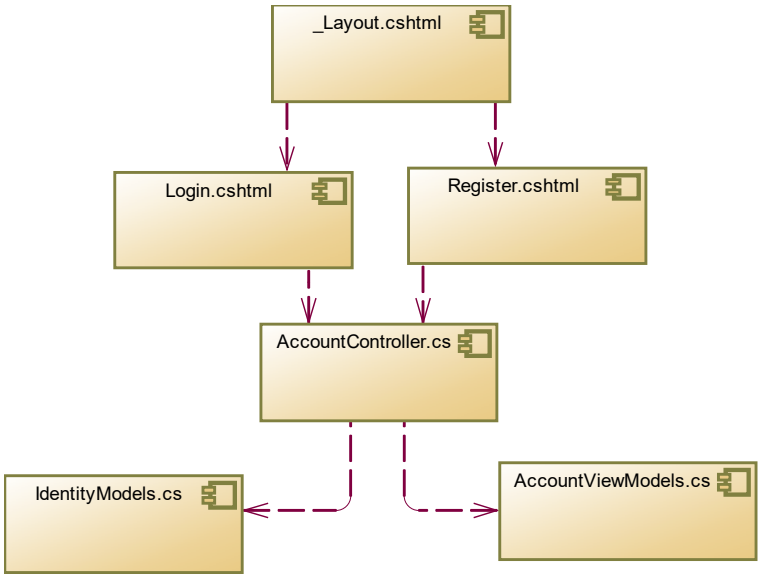

Рисунок 1.1 – Діаграма компонентів, залежності модулів відносно Account контролеру

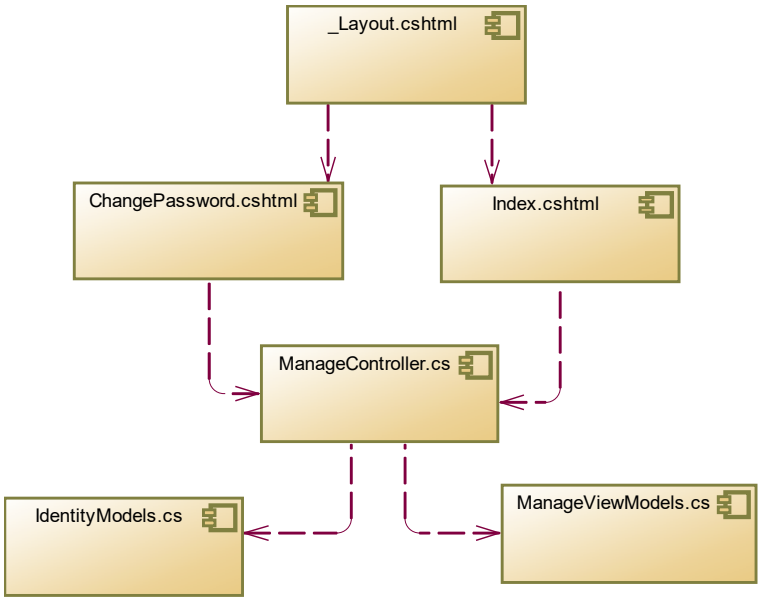

Рисунок 1.2 – Діаграма компонентів, залежності модулів відносно Manage контролеру

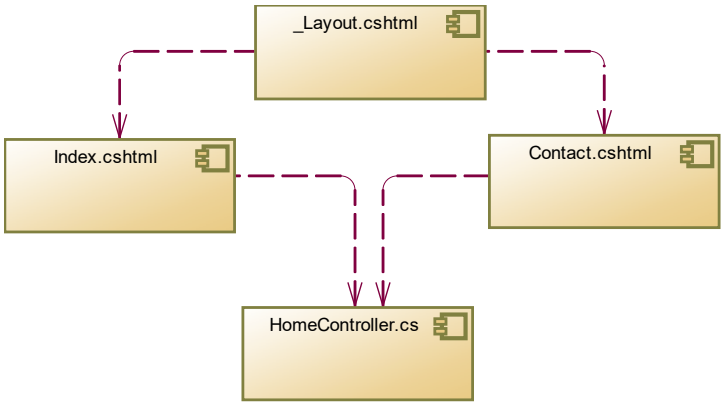

Рисунок 1.3 – Діаграма компонентів, залежності модулів відносно Home контролеру

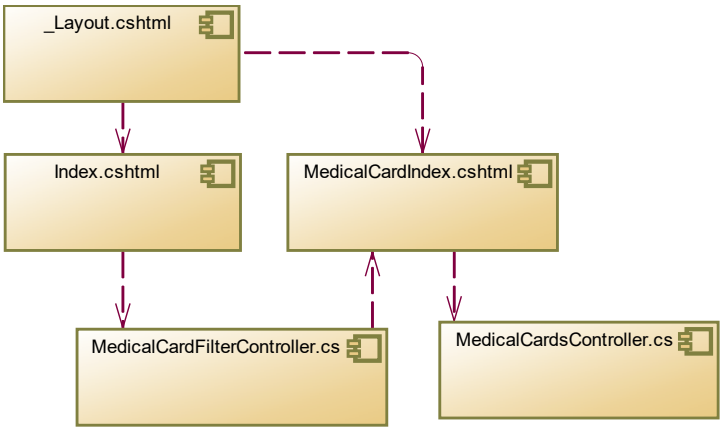

Рисунок 1.4 – Діаграма компонентів, залежності модулів відносно MedicalCardFilter контролеру

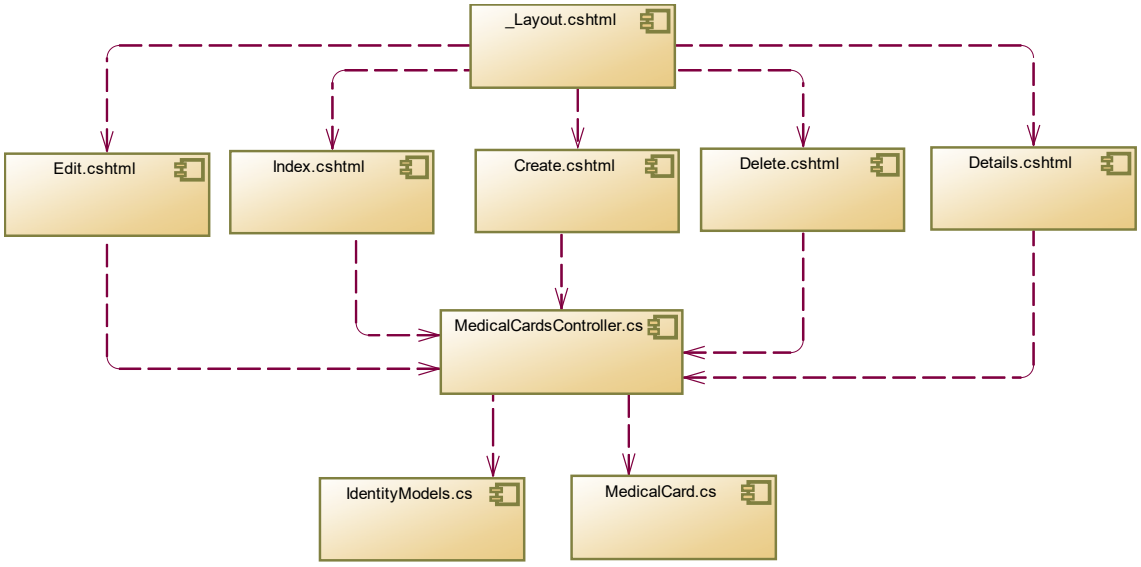

Рисунок 1.5 – Діаграма компонентів, залежності модулів відносно MedicalCards контролеру

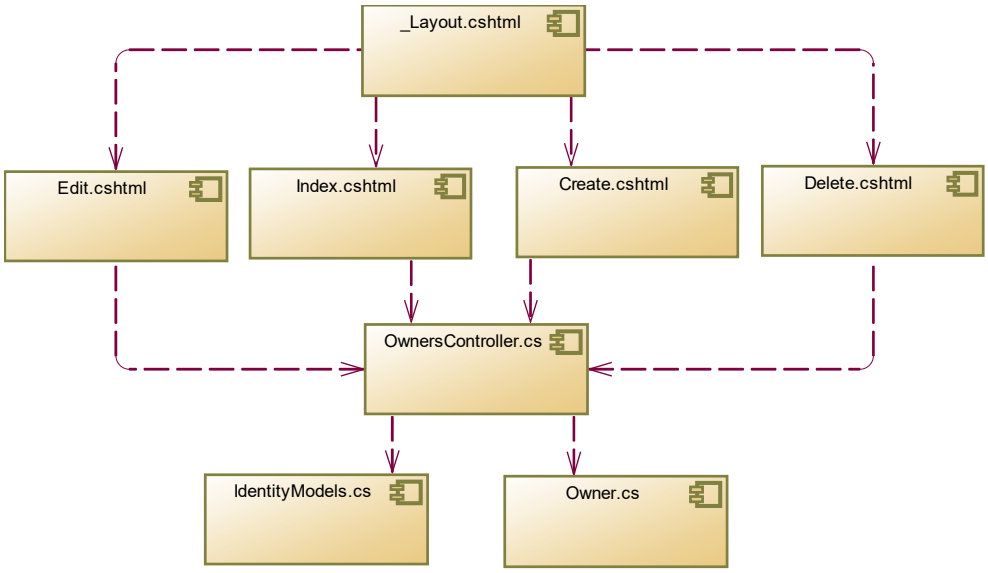

Рисунок 1.6 – Діаграма компонентів, залежності модулів відносно Owners контролеру

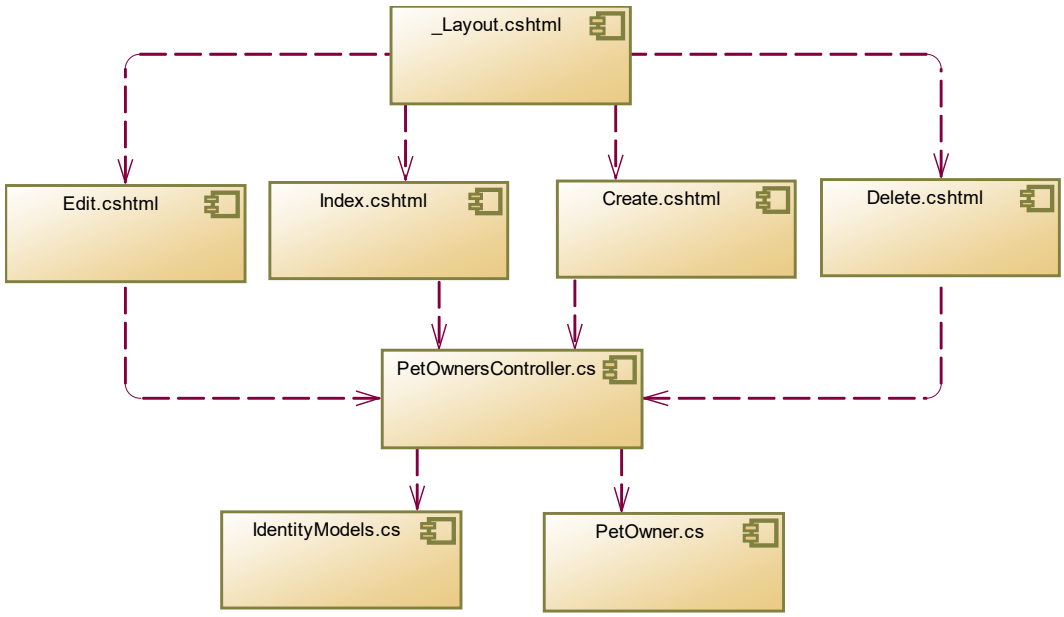

Рисунок 1.7 – Діаграма компонентів, залежності модулів відносно PetOwners контролеру

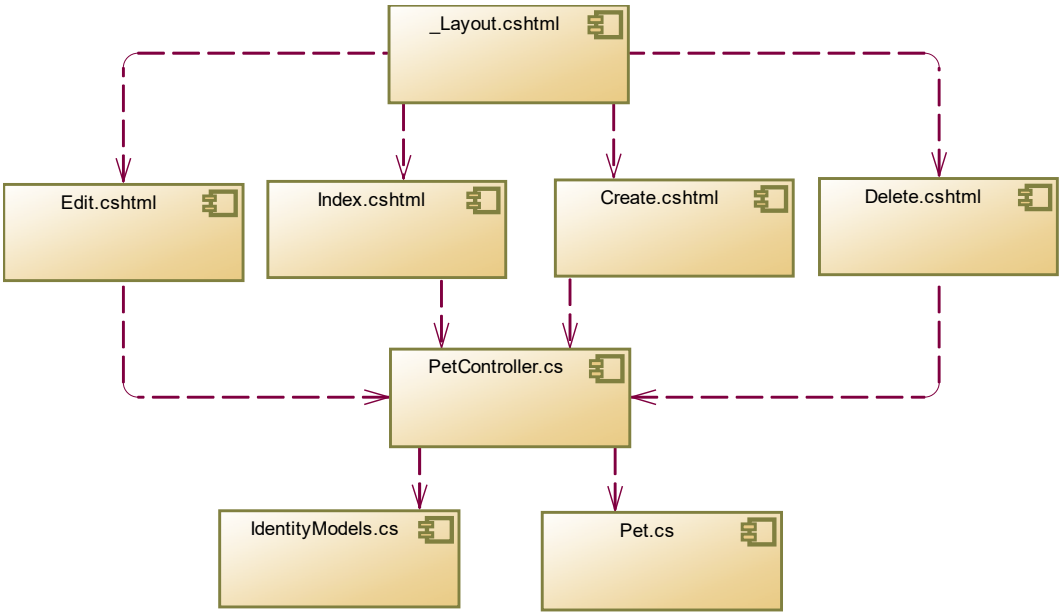

Рисунок 1.8 – Діаграма компонентів, залежності модулів відносно Pet контролеру

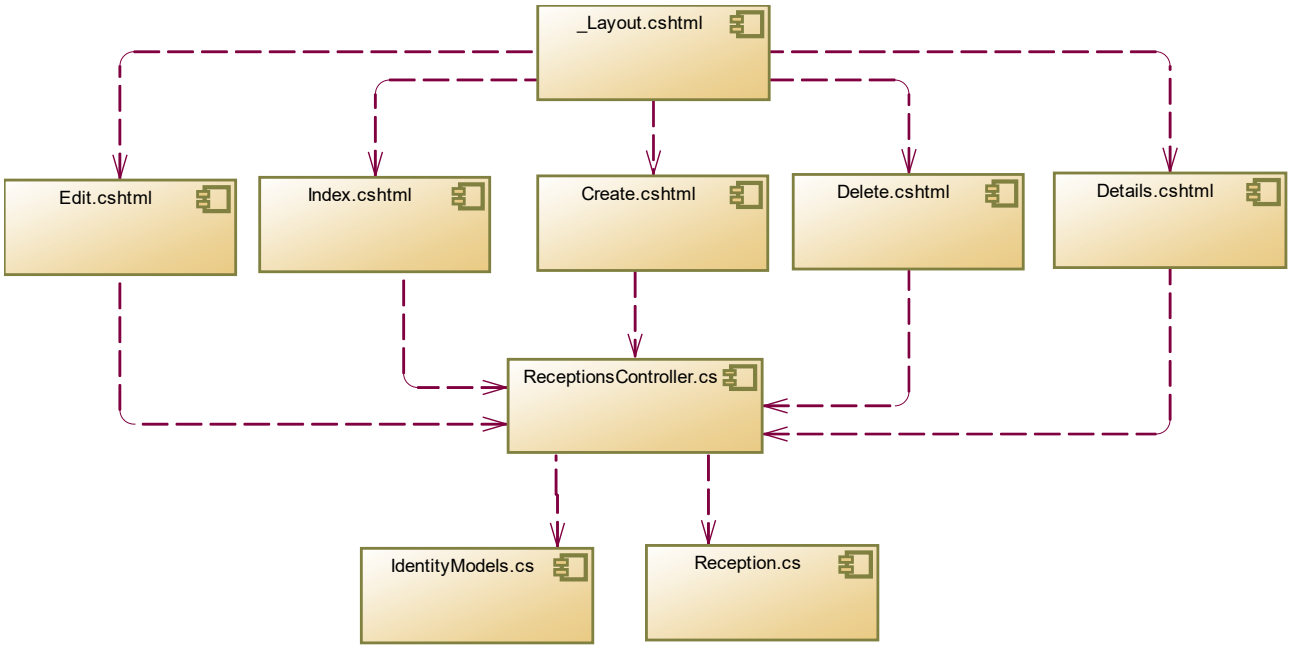

Рисунок 1.9 – Діаграма компонентів, залежності модулів відносно Receptions контролеру

2 ТЕКСТ ПРОГРАМИ

2.1 Модуль MedicalCardFilterController.cs

```
using System.Web.Mvc;
using
VeterinaryClinic.Models.View_models;

namespace VeterinaryClinic.Controllers
{
    public class
MedicalCardFilterController :
Controller
    {
        // GET: MedicalCardFilter
        [Authorize]
        public ActionResult Index()
        {
            if
(Session["MedicalCardFilter"] != null)
            {
                return
PartialView((MedicalCardFilterVM)Sessio
n["MedicalCardFilter"]);
            }
            return PartialView();
        }
        // POST: MedicalCardFilter
        [Authorize]
        [HttpPost]
        public ActionResult
Index(MedicalCardFilterVM model)
        {

Session["MedicalCardFilter"] = model;
            return
RedirectToAction("Index",
"MedicalCards");
        }
    }
}
```

2.2 Модуль MedicalCardsController.cs

```
using System;
using System.Data;
using System.Data.Entity;
using System.Linq;
using System.Net;
using System.Threading.Tasks;
using System.Web.Mvc;
using VeterinaryClinic.Models;
using
VeterinaryClinic.Models.Data_models;
using
VeterinaryClinic.Models.View_models;

namespace VeterinaryClinic.Controllers
{
    public class MedicalCardsController
: Controller
    {
        private ApplicationDbContext db
= new ApplicationDbContext();

        // GET: MedicalCards
        [Authorize]
        public async Task<ActionResult>
Index()
        {
            var currUser =
db.Users.ToArrayAsync().Result.Where(x
=>
x.Email.Equals(User.Identity.Name)).ToL
ist().FirstOrDefault();
            var medicalCards =
db.MedicalCards.Include(m =>
m.Pets).Where(card =>
card.ApplicationUser_Id.Equals(currUser
.Id)).ToArray();
            if
(Session["MedicalCardFilter"] != null)
            {
                var filtersData =
(MedicalCardFilterVM)Session["MedicalCa
rdFilter"];
                if (filtersData.Name !=
null && !filtersData.Name.Equals(""))
                {
                    medicalCards =
medicalCards.Where(x =>
x.Pets.Name.Contains(filtersData.Name))
.ToArray();
                }
                if
(filtersData.Symptoms != null &&
!filtersData.Symptoms.Equals(""))
                {
                    medicalCards =
medicalCards.Where(x =>
x.Symptoms.Contains(filtersData.Symptom
s)).ToArray();
                }
                if
(filtersData.Diagnosis != null &&
!filtersData.Diagnosis.Equals(""))
```

```
                {
                    medicalCards =
medicalCards.Where(x =>
x.Diagnosis.Contains(filtersData.Diagno
sis)).ToArray();
                }
                if
(filtersData.Treatment != null &&
!filtersData.Treatment.Equals(""))
                {
                    medicalCards =
medicalCards.Where(x =>
x.Treatment.Contains(filtersData.Treatm
ent)).ToArray();
                }
                if
(filtersData.AdmissionDateStart != null
&&
!filtersData.AdmissionDateStart.Equals(
""))
                {
                    medicalCards =
medicalCards.Where(
                        x =>

x.AdmissionDate.Day >=
filtersData.AdmissionDateStart.Day
                        &&
x.AdmissionDate.Month >=
filtersData.AdmissionDateStart.Month
                        &&
x.AdmissionDate.Year >=
filtersData.AdmissionDateStart.Day
                    ).ToArray();
                }
                if
(filtersData.AdmissionDateFinish !=
null &&
!filtersData.AdmissionDateFinish.Equals
("""))
                {
                    medicalCards =
medicalCards.Where(
                        x =>

x.AdmissionDate.Day <=
filtersData.AdmissionDateFinish.Day
                        &&
x.AdmissionDate.Month <=
filtersData.AdmissionDateFinish.Month
                        &&
x.AdmissionDate.Year <=
filtersData.AdmissionDateFinish.Year
                    ).ToArray();
                }
            }
            return
View(medicalCards.ToList());
        }

        // GET: MedicalCards/Details/5
        [Authorize]
        public async Task<ActionResult>
Details(int? id)
        {
            if (id == null)
            {
                return new
HttpStatusCodeResult(HttpStatusCode.Bad
Request);
            }
            MedicalCard medicalCard =
await db.MedicalCards.FindAsync(id);
            if (medicalCard == null)
            {
                return HttpNotFound();
            }

            var currUser =
db.Users.ToArrayAsync().Result.Where(x
=>
x.Email.Equals(User.Identity.Name)).ToL
ist().FirstOrDefault();
            if
(medicalCard.ApplicationUser_Id.Equals(
currUser.Id))
            {
                return
View(medicalCard);
            }
            return HttpNotFound();

        }

        // GET: MedicalCards/Create
        [Authorize]
        public ActionResult Create()
        {
            var currUser =
db.Users.ToArrayAsync().Result.Where(x
=>
x.Email.Equals(User.Identity.Name)).ToL
ist().FirstOrDefault();
            var pets =
db.Pets.Where(own =>
own.ApplicationUser_Id.Equals(currUser.
Id))
                        .Select(s =>
new
                        {
                            Text =
s.Name + " " + s.Specie + " " + s.Breed
+ " вагою " + s.Weight,
                            Value =
s.Id
                        })
                        .ToList();
            MedicalCard model = new
MedicalCard();
            model.AdmissionDate =
DateTime.Now;
            ViewBag.PetId = new
SelectList(pets, "Value", "Text");
            return View(model);
        }
```

```
        // POST: MedicalCards/Create
        // Чтобы защититься от атак
чрезмерной передачи данных, включите
определенные свойства, для которых
следует установить привязку.
Дополнительные
        // сведения см. в разделе
https://go.microsoft.com/fwlink/?LinkId
=317598.
        [Authorize]
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<ActionResult>
Create([Bind(Include =
"Id,PetId,Symptoms,Diagnosis,Treatment,
AdmissionDate")] MedicalCard
medicalCard)
        {
            var currUser =
db.Users.ToArrayAsync().Result.Where(x
=>
x.Email.Equals(User.Identity.Name)).ToL
ist().FirstOrDefault();
            if (ModelState.IsValid)
            {
                if (await
db.Pets.FindAsync(medicalCard.PetId) !=
null)
                {

medicalCard.ApplicationUser_Id =
currUser.Id;

db.MedicalCards.Add(medicalCard);
                    await
db.SaveChangesAsync();
                    return
RedirectToAction("Index");
                }
            }

            var pets =
db.Pets.Where(own =>
own.ApplicationUser_Id.Equals(currUser.
Id))
                        .Select(s =>
new
                        {
                            Text =
s.Name + " " + s.Specie + " " + s.Breed
+ " вагою " + s.Weight,
                            Value =
s.Id
                        })
                        .ToList();

            ViewBag.PetId = new
SelectList(pets, "Value", "Text");
            return View(medicalCard);
        }

        // GET: MedicalCards/Edit/5
        [Authorize]
        public async Task<ActionResult>
Edit(int? id)
        {
            if (id == null)
            {
                return new
HttpStatusCodeResult(HttpStatusCode.Bad
Request);
            }
            MedicalCard medicalCard =
await db.MedicalCards.FindAsync(id);
            if (medicalCard == null)
            {
                return HttpNotFound();
            }

            var currUser =
db.Users.ToArrayAsync().Result.Where(x
=>
x.Email.Equals(User.Identity.Name)).ToL
ist().FirstOrDefault();
            if
(medicalCard.ApplicationUser_Id.Equals(
currUser.Id))
            {
                var pets =
db.Pets.Where(own =>
own.ApplicationUser_Id.Equals(currUser.
Id))
                                .Select(s
=> new
                                {
                                    Text =
s.Name + " " + s.Specie + " " + s.Breed
+ " вагою " + s.Weight,
                                    Value =
s.Id
                                })
                                .ToList();

                ViewBag.PetId = new
SelectList(pets, "Value", "Text");
                return
View(medicalCard);
            }
            return HttpNotFound();
        }

        // POST: MedicalCards/Edit/5
        // Чтобы защититься от атак
чрезмерной передачи данных, включите
определенные свойства, для которых
следует установить привязку.
Дополнительные
        // сведения см. в разделе
https://go.microsoft.com/fwlink/?LinkId
=317598.
        [Authorize]
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<ActionResult>
Edit([Bind(Include =
```

```
"Id,PetId,Symptoms,Diagnosis,Treatment,
AdmissionDate,ApplicationUser_Id")]
MedicalCard medicalCard)
        {
            var currUser =
db.Users.ToArrayAsync().Result.Where(x
=>
x.Email.Equals(User.Identity.Name)).ToL
ist().FirstOrDefault();
            if (ModelState.IsValid)
            {
                if (await
db.Pets.FindAsync(medicalCard.PetId) !=
null)
                {

medicalCard.ApplicationUser_Id =
currUser.Id;

db.Entry(medicalCard).State =
EntityState.Modified;
                    await
db.SaveChangesAsync();
                    return
RedirectToAction("Index");
                }
            }
            var pets =
db.Pets.Where(own =>
own.ApplicationUser_Id.Equals(currUser.
Id))
                            .Select(s
=> new
                            {
                                Text =
s.Name + " " + s.Specie + " " + s.Breed
+ " вагою " + s.Weight,
                                Value =
s.Id
                            })
                            .ToList();
            ViewBag.PetId = new
SelectList(pets, "Value", "Text");
            return View(medicalCard);
        }

        // GET: MedicalCards/Delete/5
        [Authorize]
        public async Task<ActionResult>
Delete(int? id)
        {
            if (id == null)
            {
                return new
HttpStatusCodeResult(HttpStatusCode.Bad
Request);
            }
            MedicalCard medicalCard =
await db.MedicalCards.FindAsync(id);
            if (medicalCard == null)
            {
                return HttpNotFound();
            }
            return View(medicalCard);
        }

        // POST: MedicalCards/Delete/5
        [Authorize]
        [HttpPost,
ActionName("Delete")]
        [ValidateAntiForgeryToken]
        public async Task<ActionResult>
DeleteConfirmed(int id)
        {
            MedicalCard medicalCard =
await db.MedicalCards.FindAsync(id);
            var currUser =
db.Users.ToArrayAsync().Result.Where(x
=>
x.Email.Equals(User.Identity.Name)).ToL
ist().FirstOrDefault();
            if
(medicalCard.ApplicationUser_Id.Equals(
currUser.Id))
            {

db.MedicalCards.Remove(medicalCard);
                await
db.SaveChangesAsync();
            }
            return
RedirectToAction("Index");
        }

        [Authorize]
        protected override void
Dispose(bool disposing)
        {
            if (disposing)
            {
                db.Dispose();
            }
            base.Dispose(disposing);
        }
    }
}
```

2.3 Модуль OwnersController.cs

```
using System.Data;
using System.Data.Entity;
using System.Linq;
using System.Net;
using System.Threading.Tasks;
using System.Web.Mvc;
using VeterinaryClinic.Models;

namespace VeterinaryClinic.Controllers
{
    [Authorize]
    public class OwnersController :
Controller
    {
        private ApplicationDbContext db
= new ApplicationDbContext();

        // GET: Owners

        [Authorize]
        public async Task<ActionResult>
Index()
        {
            var currUser =
db.Users.ToArrayAsync().Result.Where(x
=>
x.Email.Equals(User.Identity.Name)).ToL
ist().FirstOrDefault();
            return
View(db.Owners.ToArrayAsync().Result.Wh
ere(x =>
x.ApplicationUser_Id.Equals(currUser.Id
)).ToList());
        }

        [Authorize]
        // GET: Owners/Details/5
        public async Task<ActionResult>
Details(int? id)
        {
            if (id == null)
            {
                return new
HttpStatusCodeResult(HttpStatusCode.Bad
Request);
            }
            Owner owner = await
db.Owners.FindAsync(id);
            if (owner == null)
            {
                return HttpNotFound();
            }
            var currUser =
db.Users.ToArrayAsync().Result.Where(x
=>
x.Email.Equals(User.Identity.Name)).ToL
ist().FirstOrDefault();
            if
(!owner.ApplicationUser_Id.Equals(currU
ser.Id))
            {
                return HttpNotFound();
            }
            return View(owner);
        }

        [Authorize]
        // GET: Owners/Create
        public ActionResult Create()
        {
            return View();
        }

        // POST: Owners/Create
        [Authorize]
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<ActionResult>
Create([Bind(Include =
"Id,FirstName,LastName,Phone,Email")]
Owner owner)
        {
            var currUser =
db.Users.ToArrayAsync().Result.Where(x
=>
x.Email.Equals(User.Identity.Name)).ToL
ist().FirstOrDefault();
            if (ModelState.IsValid)
            {
                if
(db.Owners.ToArrayAsync().Result.Where(
x => x.Phone.Equals(owner.Phone) &&
x.ApplicationUser_Id.Equals(currUser.Id
)).Any())
                {

ModelState.AddModelError("Phone",
"Господар з таким номером телефону вже
існує");
                    return
View("Create", owner);
                }
                else
                {

owner.ApplicationUser_Id = currUser.Id;

db.Owners.Add(owner);
                    await
db.SaveChangesAsync();
                }
                return
RedirectToAction("Index");
            }

            return View(owner);
        }

        // GET: Owners/Edit/5

        [Authorize]
```

```
        public async Task<ActionResult>
Edit(int? id)
        {
            if (id == null)
            {
                return new
HttpStatusCodeResult(HttpStatusCode.Bad
Request);
            }
            Owner owner = await
db.Owners.FindAsync(id);
            if (owner == null)
            {
                return HttpNotFound();
            }
            var currUser =
db.Users.ToArrayAsync().Result.Where(x
=>
x.Email.Equals(User.Identity.Name)).ToL
ist().FirstOrDefault();
            if
(owner.ApplicationUser_Id.Equals(currUs
er.Id))
            {
                return View(owner);
            }
            return
RedirectToAction("Index");
        }

        // POST: Owners/Edit/5
        [Authorize]
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<ActionResult>
Edit([Bind(Include =
"Id,FirstName,LastName,Phone,Email")]
Owner owner)
        {
            if (ModelState.IsValid)
            {
                var currUser =
db.Users.ToArrayAsync().Result.Where(x
=>
x.Email.Equals(User.Identity.Name)).ToL
ist().FirstOrDefault();
                if
(db.Owners.ToArrayAsync().Result.Where(
x => x.Phone.Equals(owner.Phone) &&
x.ApplicationUser_Id.Equals(currUser.Id
)).Any())
                {

ModelState.AddModelError("Phone",
"Господар з таким номером телефону вже
існує");
                    return View(owner);
                }
                else
                {
                    var item =
db.Owners.Find(owner.Id);
                    item.FirstName =
owner.FirstName;
                    item.LastName =
owner.LastName;
                    item.Email =
owner.Email;
                    item.Phone =
owner.Phone;

item.ApplicationUser_Id = currUser.Id;


db.Entry(item).State =
EntityState.Modified;
                    await
db.SaveChangesAsync();
                }
                return
RedirectToAction("Index");
            }
            return View(owner);
        }

        // GET: Owners/Delete/5
        [Authorize]
        public async Task<ActionResult>
Delete(int? id)
        {
            if (id == null)
            {
                return new
HttpStatusCodeResult(HttpStatusCode.Bad
Request);
            }
            Owner owner = await
db.Owners.FindAsync(id);
            if (owner == null)
            {
                return HttpNotFound();
            }

            var currUser =
db.Users.ToArrayAsync().Result.Where(x
=>
x.Email.Equals(User.Identity.Name)).ToL
ist().FirstOrDefault();
            if
(owner.ApplicationUser_Id.Equals(currUs
er.Id))
            {
                return View(owner);
            }
            return
RedirectToAction("Index");
        }

        // POST: Owners/Delete/5
        [Authorize]
        [HttpPost,
ActionName("Delete")]
        [ValidateAntiForgeryToken]
        public async Task<ActionResult>
DeleteConfirmed(int id)
```

```
        {
            Owner owner = await
db.Owners.FindAsync(id);
            var currUser =
db.Users.ToArrayAsync().Result.Where(x
=>
x.Email.Equals(User.Identity.Name)).ToL
ist().FirstOrDefault();
            if
(owner.ApplicationUser_Id.Equals(currUs
er.Id))
            {

db.Owners.Remove(owner);
                await
db.SaveChangesAsync();
            }
            return
RedirectToAction("Index");
        }

        [Authorize]
        protected override void
Dispose(bool disposing)
        {
            if (disposing)
            {
                db.Dispose();
            }
            base.Dispose(disposing);
        }
    }
}
```

2.4 Модуль PetOwnersController.cs

```
using System.Data;
using System.Data.Entity;
using System.Linq;
using System.Net;
using System.Threading.Tasks;
using System.Web.Mvc;
using VeterinaryClinic.Models;
using
VeterinaryClinic.Models.Data_models;

namespace VeterinaryClinic.Controllers
{
    public class PetOwnersController :
Controller
    {
        private ApplicationDbContext db
= new ApplicationDbContext();

        // GET: PetOwners

        [Authorize]
        public async Task<ActionResult>
Index()
        {
            var currUser =
db.Users.ToArrayAsync().Result.Where(x
=>
x.Email.Equals(User.Identity.Name)).ToL
ist().FirstOrDefault();
            var petsOwners =
db.PetsOwners.Include(p =>
p.Owners).Include(p => p.Pets).Where(p
=>
p.ApplicationUser_Id.Equals(currUser.Id
));
            return View(await
petsOwners.ToListAsync());
        }

        // GET: PetOwners/Details/5

        [Authorize]
        public async Task<ActionResult>
Details(int? id)
        {
            if (id == null)
            {
                return new
HttpStatusCodeResult(HttpStatusCode.Bad
Request);
            }
            PetOwner petOwner = await
db.PetsOwners.FindAsync(id);
            if (petOwner == null)
            {
                return HttpNotFound();
            }
            var currUser =
db.Users.ToArrayAsync().Result.Where(x
=>
x.Email.Equals(User.Identity.Name)).ToL
ist().FirstOrDefault();
            if
(petOwner.ApplicationUser_Id.Equals(cur
rUser.Id))
                return View(petOwner);
            return HttpNotFound();
        }

        // GET: PetOwners/Create

        [Authorize]
        public ActionResult Create()
        {
            var currUser =
db.Users.ToArrayAsync().Result.Where(x
=>
x.Email.Equals(User.Identity.Name)).ToL
ist().FirstOrDefault();
            var owners =
db.Owners.Where(own =>
```

```
                        {
                            Text =
s.FirstName + " " + s.LastName + " " +
s.Phone,
                            Value =
s.Id
                        })
                        .ToList();

            ViewBag.OwnerId = new
SelectList(owners, "Value", "Text");

            var pets =
db.Pets.Where(own =>
own.ApplicationUser_Id.Equals(currUser.
Id))
                        .Select(s =>
new
                        {
                            Text =
s.Name + " " + s.Specie + " " + s.Breed
+ " вагою " + s.Weight,
                            Value =
s.Id
                        })
                        .ToList();

            ViewBag.PetId = new
SelectList(pets, "Value", "Text");

            return View();
        }

        // POST: PetOwners/Create
        // Чтобы защититься от атак
чрезмерной передачи данных, включите
определенные свойства, для которых
следует установить привязку.
Дополнительные
        // сведения см. в разделе
https://go.microsoft.com/fwlink/?LinkId
=317598.

        [Authorize]
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<ActionResult>
Create([Bind(Include =
"Id,PetId,OwnerId")] PetOwner petOwner)
        {
            var currUser =
db.Users.ToArrayAsync().Result.Where(x
=>
x.Email.Equals(User.Identity.Name)).ToL
ist().FirstOrDefault();

            if (ModelState.IsValid)
            {
                if (await
db.Pets.FindAsync(petOwner.PetId) !=
null)
                {
                    db.PetsOwners.Add(petOwner);

                    petOwner.ApplicationUser_Id =
currUser.Id;
                    await
db.SaveChangesAsync();
                    return
RedirectToAction("Index");
                }
            }

            var owners =
db.Owners.Where(own =>
own.ApplicationUser_Id.Equals(currUser.
Id))
                        .Select(s =>
new
                        {
                            Text =
s.FirstName + " " + s.LastName + " " +
s.Phone,
                            Value =
s.Id
                        })
                        .ToList();

            ViewBag.OwnerId = new
SelectList(owners, "Value", "Text");

            var pets =
db.Pets.Where(own =>
own.ApplicationUser_Id.Equals(currUser.
Id))
                        .Select(s =>
new
                        {
                            Text =
s.Name + " " + s.Specie + " " + s.Breed
+ " вагою " + s.Weight,
                            Value =
s.Id
                        })
                        .ToList();

            ViewBag.PetId = new
SelectList(pets, "Value", "Text");

            return View(petOwner);
        }

        // GET: PetOwners/Edit/5

        [Authorize]
        public async Task<ActionResult>
Edit(int? id)
        {
            if (id == null)
            {
                return new
HttpStatusCodeResult(HttpStatusCode.Bad
Request);
            }
```

```
            PetOwner petOwner = await
db.PetsOwners.FindAsync(id);
            if (petOwner == null)
            {
                return HttpNotFound();
            }

            var currUser =
db.Users.ToArrayAsync().Result.Where(x
=>
x.Email.Equals(User.Identity.Name)).ToL
ist().FirstOrDefault();

            if
(petOwner.ApplicationUser_Id.Equals(cur
rUser.Id))
            {

                var owners =
db.Owners.Where(own =>
own.ApplicationUser_Id.Equals(currUser.
Id))
                                .Select(s
=> new
                                {
                                    Text =
s.FirstName + " " + s.LastName + " " +
s.Phone,
                                    Value =
s.Id
                                })
                                .ToList();

                ViewBag.OwnerId = new
SelectList(owners, "Value", "Text");

                var pets =
db.Pets.Where(own =>
own.ApplicationUser_Id.Equals(currUser.
Id))
                                .Select(s
=> new
                                {
                                    Text =
s.Name + " " + s.Specie + " " + s.Breed
+ " вагою " + s.Weight,
                                    Value =
s.Id
                                })
                                .ToList();

                ViewBag.PetId = new
SelectList(pets, "Value", "Text");
                return View(petOwner);
            }
            return HttpNotFound();

        }

        // POST: PetOwners/Edit/5
        // Чтобы защититься от атак
чрезмерной передачи данных, включите
определенные свойства, для которых
следует установить привязку.
Дополнительные
        // сведения см. в разделе
https://go.microsoft.com/fwlink/?LinkId
=317598.

        [Authorize]
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<ActionResult>
Edit([Bind(Include =
"Id,PetId,OwnerId")] PetOwner petOwner)
        {
            var currUser =
db.Users.ToArrayAsync().Result.Where(x
=>
x.Email.Equals(User.Identity.Name)).ToL
ist().FirstOrDefault();
            if (ModelState.IsValid)
            {
                if (await
db.Pets.FindAsync(petOwner.PetId) !=
null)
                {

petOwner.ApplicationUser_Id =
currUser.Id;

db.Entry(petOwner).State =
EntityState.Modified;
                    await
db.SaveChangesAsync();
                    return
RedirectToAction("Index");
                }
            }

            var owners =
db.Owners.Where(own =>
own.ApplicationUser_Id.Equals(currUser.
Id))
                        .Select(s =>
new
                        {
                            Text =
s.FirstName + " " + s.LastName + " " +
s.Phone,
                            Value =
s.Id
                        })
                        .ToList();

            ViewBag.OwnerId = new
SelectList(owners, "Value", "Text");

            var pets =
db.Pets.Where(own =>
own.ApplicationUser_Id.Equals(currUser.
Id))
                        .Select(s =>
new
                        {
```

```
                                Text =
s.Name + " " + s.Specie + " " + s.Breed
+ " вагою " + s.Weight,
                                Value =
s.Id
                            })
                            .ToList();

            ViewBag.PetId = new
SelectList(pets, "Value", "Text");
            return View(petOwner);
        }

        // GET: PetOwners/Delete/5

        [Authorize]
        public async Task<ActionResult>
Delete(int? id)
        {
            if (id == null)
            {
                return new
HttpStatusCodeResult(HttpStatusCode.Bad
Request);
            }
            PetOwner petOwner = await
db.PetsOwners.FindAsync(id);
            if (petOwner == null)
            {
                return HttpNotFound();
            }
            var currUser =
db.Users.ToArrayAsync().Result.Where(x
=>
x.Email.Equals(User.Identity.Name)).ToL
ist().FirstOrDefault();
            if
(petOwner.ApplicationUser_Id.Equals(cur
rUser.Id))
            {
                return View(petOwner);
            }
            return
RedirectToAction("Index");
        }

        // POST: PetOwners/Delete/5

        [Authorize]
        [HttpPost,
ActionName("Delete")]
        [ValidateAntiForgeryToken]
        public async Task<ActionResult>
DeleteConfirmed(int id)
        {
            PetOwner petOwner = await
db.PetsOwners.FindAsync(id);

            var currUser =
db.Users.ToArrayAsync().Result.Where(x
=>
x.Email.Equals(User.Identity.Name)).ToL
ist().FirstOrDefault();
            if (petOwner != null &&
petOwner.ApplicationUser_Id.Equals(curr
User.Id))
            {

db.PetsOwners.Remove(petOwner);
                await
db.SaveChangesAsync();
            }
            return
RedirectToAction("Index");
        }


        [Authorize]
        protected override void
Dispose(bool disposing)
        {
            if (disposing)
            {
                db.Dispose();
            }
            base.Dispose(disposing);
        }
    }
}
```

2.5 Модуль PetsController.cs

```
using System.Data;
using System.Data.Entity;
using System.Linq;
using System.Net;
using System.Threading.Tasks;
using System.Web.Mvc;

namespace
VeterinaryClinic.Models.Data_models
{
    public class PetsController :
Controller
    {
        private ApplicationDbContext db
= new ApplicationDbContext();

        // GET: Pets

        [Authorize]
        public async Task<ActionResult>
Index()
        {
            var currUser =
db.Users.ToArrayAsync().Result.Where(x
=>
x.Email.Equals(User.Identity.Name)).ToL
ist().FirstOrDefault();
```

```
            return
View(db.Pets.ToArrayAsync().Result.Wher
e(x => x.ApplicationUser_Id == null ||
x.ApplicationUser_Id.Equals(currUser.Id
)).ToList());
        }

        // GET: Pets/Details/5

        [Authorize]
        public async Task<ActionResult>
Details(int? id)
        {
            if (id == null)
            {
                return new
HttpStatusCodeResult(HttpStatusCode.Bad
Request);
            }
            Pet pet = await
db.Pets.FindAsync(id);
            if (pet == null)
            {
                return HttpNotFound();
            }
            var currUser =
db.Users.ToArrayAsync().Result.Where(x
=>
x.Email.Equals(User.Identity.Name)).ToL
ist().FirstOrDefault();
            if
(!pet.ApplicationUser_Id.Equals(currUse
r.Id))
            {
                return HttpNotFound();
            }
            return View(pet);
        }

        // GET: Pets/Create

        [Authorize]
        public ActionResult Create()
        {
            return View();
        }

        // POST: Pets/Create
        // Чтобы защититься от атак
чрезмерной передачи данных, включите
определенные свойства, для которых
следует установить привязку.
Дополнительные
        // сведения см. в разделе
https://go.microsoft.com/fwlink/?LinkId
=317598.

        [Authorize]
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<ActionResult>
Create([Bind(Include =
"Id,Name,Breed,Specie,Age,Sex,Weight")]
Pet pet)
        {
            var currUser =
db.Users.ToArrayAsync().Result.Where(x
=>
x.Email.Equals(User.Identity.Name)).ToL
ist().FirstOrDefault();
            if (ModelState.IsValid)
            {
                pet.ApplicationUser_Id
= currUser.Id;
                db.Pets.Add(pet);
                await
db.SaveChangesAsync();
                return
RedirectToAction("Index");
            }

            return View(pet);
        }

        // GET: Pets/Edit/5

        [Authorize]
        public async Task<ActionResult>
Edit(int? id)
        {
            if (id == null)
            {
                return new
HttpStatusCodeResult(HttpStatusCode.Bad
Request);
            }
            Pet pet = await
db.Pets.FindAsync(id);
            if (pet == null)
            {
                return HttpNotFound();
            }
            var currUser =
db.Users.ToArrayAsync().Result.Where(x
=>
x.Email.Equals(User.Identity.Name)).ToL
ist().FirstOrDefault();
            if
(pet.ApplicationUser_Id.Equals(currUser
.Id))
            {
                return View(pet);
            }
            return
RedirectToAction("Index");
        }

        // POST: Pets/Edit/5
        // Чтобы защититься от атак
чрезмерной передачи данных, включите
определенные свойства, для которых
следует установить привязку.
Дополнительные
```

```
        // сведения см. в разделе
https://go.microsoft.com/fwlink/?LinkId
=317598.

        [Authorize]
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<ActionResult>
Edit([Bind(Include =
"Id,Name,Breed,Specie,Age,Sex,Weight")]
Pet pet)
        {
            if (ModelState.IsValid)
            {
                var currUser =
db.Users.ToArrayAsync().Result.Where(x
=>
x.Email.Equals(User.Identity.Name)).ToL
ist().FirstOrDefault();
                pet.ApplicationUser_Id
= currUser.Id;
                db.Entry(pet).State =
EntityState.Modified;
                await
db.SaveChangesAsync();
                return
RedirectToAction("Index");
            }
            return View(pet);
        }

        // GET: Pets/Delete/5

        [Authorize]
        public async Task<ActionResult>
Delete(int? id)
        {
            if (id == null)
            {
                return new
HttpStatusCodeResult(HttpStatusCode.Bad
Request);
            }
            Pet pet = await
db.Pets.FindAsync(id);
            if (pet == null)
            {
                return HttpNotFound();
            }
            var currUser =
db.Users.ToArrayAsync().Result.Where(x
=>
x.Email.Equals(User.Identity.Name)).ToL
ist().FirstOrDefault();
            if
(pet.ApplicationUser_Id.Equals(currUser
.Id))
            {
                return View(pet);
            }
            return
RedirectToAction("Index");
        }

        // POST: Pets/Delete/5
        [HttpPost,
ActionName("Delete")]
        [ValidateAntiForgeryToken]

        [Authorize]
        public async Task<ActionResult>
DeleteConfirmed(int id)
        {
            Pet pet = await
db.Pets.FindAsync(id);

            var currUser =
db.Users.ToArrayAsync().Result.Where(x
=>
x.Email.Equals(User.Identity.Name)).ToL
ist().FirstOrDefault();
            if
(pet.ApplicationUser_Id.Equals(currUser
.Id))
            {
                db.Pets.Remove(pet);
                await
db.SaveChangesAsync();
            }
            return
RedirectToAction("Index");
        }


        [Authorize]
        protected override void
Dispose(bool disposing)
        {
            if (disposing)
            {
                db.Dispose();
            }
            base.Dispose(disposing);
        }
    }
}
```

2.6 Модуль ReceptionsController.cs

```
using System;
using System.Data;
using System.Data.Entity;
using System.Linq;
using System.Net;
using System.Threading.Tasks;
using System.Web.Mvc;
using VeterinaryClinic.Models;
using
VeterinaryClinic.Models.Data_models;
```

```
namespace VeterinaryClinic.Controllers
{
    public class ReceptionsController :
Controller
    {
        private ApplicationDbContext db
= new ApplicationDbContext();

        // GET: Receptions
        [Authorize]
        public async Task<ActionResult>
Index()
        {
            var currUser =
db.Users.ToArrayAsync().Result.Where(x
=>
x.Email.Equals(User.Identity.Name)).ToL
ist().FirstOrDefault();
            var receptions =
db.Receptions.Include(r =>
r.PetsOwners).Where(x =>
x.ApplicationUser_Id.Equals(currUser.Id
) && x.AdmissionDate >= DateTime.Now);
            return View(await
receptions.ToListAsync());
        }

        // GET: Receptions/Details/5
        [Authorize]
        public async Task<ActionResult>
Details(int? id)
        {
            if (id == null)
            {
                return new
HttpStatusCodeResult(HttpStatusCode.Bad
Request);
            }
            Reception reception = await
db.Receptions.FindAsync(id);
            if (reception == null)
            {
                return HttpNotFound();
            }

            var currUser =
db.Users.ToArrayAsync().Result.Where(x
=>
x.Email.Equals(User.Identity.Name)).ToL
ist().FirstOrDefault();
            if
(reception.ApplicationUser_Id.Equals(cu
rrUser.Id))
            {
                var petList =
db.Receptions.Where(x =>
x.ApplicationUser_Id.Equals(currUser.Id
) && x.Id == id)
                        .Select(s => new
                        {
                            Text =
s.PetsOwners.Owners.FirstName + " " +
s.PetsOwners.Owners.LastName + " " +
s.PetsOwners.Owners.Phone
                                + " " +
s.PetsOwners.Pets.Name + " " +
s.PetsOwners.Pets.Specie + " " +
s.PetsOwners.Pets.Breed + " вагою " +
s.PetsOwners.Pets.Weight,
                            Value = s.Id
                        })
                        .ToList();

                ViewBag.PetOwnerId =
petList.FirstOrDefault().Text;
                return View(reception);
            }
            return
RedirectToAction("Index");
        }

        // GET: Receptions/Create
        [Authorize]
        public ActionResult Create()
        {
            var currUser =
db.Users.ToArrayAsync().Result.Where(x
=>
x.Email.Equals(User.Identity.Name)).ToL
ist().FirstOrDefault();
            var petList =
db.PetsOwners.Where(x =>
x.ApplicationUser_Id.Equals(currUser.Id
))
                .Select(s => new
                {
                    Text =
s.Owners.FirstName + " " +
s.Owners.LastName + " " +
s.Owners.Phone
                        + " " +
s.Pets.Name + " " + s.Pets.Specie + " "
+ s.Pets.Breed + " вагою " +
s.Pets.Weight,
                    Value = s.Id
                })
                .ToList();

            ViewBag.PetOwnerId = new
SelectList(petList, "Value", "Text");
            return View();
        }

        // POST: Receptions/Create
        // Чтобы защититься от атак
чрезмерной передачи данных, включите
определенные свойства, для которых
следует установить привязку.
Дополнительные
        // сведения см. в разделе
https://go.microsoft.com/fwlink/?LinkId
=317598.
        [Authorize]
        [HttpPost]
        [ValidateAntiForgeryToken]
```

```
        public async Task<ActionResult>
Create([Bind(Include =
"Id,PetOwnerId,AdmissionDate")]
Reception reception)
        {
            var currUser =
db.Users.ToArrayAsync().Result.Where(x
=>
x.Email.Equals(User.Identity.Name)).ToL
ist().FirstOrDefault();

            if (reception.AdmissionDate
< DateTime.Now)
            {

ModelState.AddModelError("AdmissionDate
", "Дата та час наступного візиту
повинні бути більше ніж поточна дата та
час");
            }


            if (await
db.Owners.FindAsync(reception.PetOwnerI
d) != null)
                if (ModelState.IsValid)
                {

reception.ApplicationUser_Id =
currUser.Id;

db.Receptions.Add(reception);
                    await
db.SaveChangesAsync();
                    return
RedirectToAction("Index");
                }

            var petList =
db.PetsOwners.Where(x =>
x.ApplicationUser_Id.Equals(currUser.Id
))
                .Select(s => new
                {
                    Text =
s.Owners.FirstName + " " +
s.Owners.LastName + " " +
s.Owners.Phone
                        + " " +
s.Pets.Name + " " + s.Pets.Specie + " "
+ s.Pets.Breed + " вагою " +
s.Pets.Weight,
                    Value = s.Id
                })
                .ToList();

            ViewBag.PetOwnerId = new
SelectList(petList, "Value", "Text");
            return View(reception);
        }

        // GET: Receptions/Edit/5
        [Authorize]
        public async Task<ActionResult>
Edit(int? id)
        {
            if (id == null)
            {
                return new
HttpStatusCodeResult(HttpStatusCode.Bad
Request);
            }
            Reception reception = await
db.Receptions.FindAsync(id);
            if (reception == null)
            {
                return HttpNotFound();
            }
            var currUser =
db.Users.ToArrayAsync().Result.Where(x
=>
x.Email.Equals(User.Identity.Name)).ToL
ist().FirstOrDefault();
            if
(reception.ApplicationUser_Id.Equals(cu
rrUser.Id))
            {
                var petList =
db.PetsOwners.Where(x =>
x.ApplicationUser_Id.Equals(currUser.Id
))
                    .Select(s => new
                    {
                        Text =
s.Owners.FirstName + " " +
s.Owners.LastName + " " +
s.Owners.Phone
                            + " " +
s.Pets.Name + " " + s.Pets.Specie + " "
+ s.Pets.Breed + " вагою " +
s.Pets.Weight,
                        Value = s.Id
                    })
                    .ToList();

                ViewBag.PetOwnerId =
new SelectList(petList, "Value",
"Text");
                return View(reception);
            }
            return HttpNotFound();

        }

        // POST: Receptions/Edit/5
        // Чтобы защититься от атак
чрезмерной передачи данных, включите
определенные свойства, для которых
следует установить привязку.
Дополнительные
        // сведения см. в разделе
https://go.microsoft.com/fwlink/?LinkId
=317598.
        [HttpPost]
        [ValidateAntiForgeryToken]
        [Authorize]
```

```
        public async Task<ActionResult>
Edit([Bind(Include =
"Id,PetOwnerId,AdmissionDate")]
Reception reception)
        {

reception.ApplicationUser_Id =
currUser.Id;

db.Entry(reception).State =
EntityState.Modified;
                    await
db.SaveChangesAsync();
                    return
RedirectToAction("Index");
                }
            var petList =
db.PetsOwners.Where(x =>
x.ApplicationUser_Id.Equals(currUser.Id
))
                    .Select(s => new
                    {
                        Text =
s.Owners.FirstName + " " +
s.Owners.LastName + " " +
s.Owners.Phone
                                + " " +
s.Pets.Name + " " + s.Pets.Specie + " "
+ s.Pets.Breed + " вагою " +
s.Pets.Weight,
                        Value = s.Id
                    })
                    .ToList();

            ViewBag.PetOwnerId = new
SelectList(petList, "Value", "Text");
            return View(reception);
        }

        // GET: Receptions/Delete/5
        [Authorize]
        public async Task<ActionResult>
Delete(int? id)
        {
            var currUser =
db.Users.ToArrayAsync().Result.Where(x
=>
x.Email.Equals(User.Identity.Name)).ToL
ist().FirstOrDefault();
            if (id == null)
            {
                return new
HttpStatusCodeResult(HttpStatusCode.Bad
Request);
            }
            Reception reception = await
db.Receptions.FindAsync(id);
            if (reception == null)
            {
                return HttpNotFound();
            }
            if
(reception.ApplicationUser_Id.Equals(cu
rrUser.Id))
            {
                var petList =
db.Receptions.Where(x =>
x.ApplicationUser_Id.Equals(currUser.Id
) && x.Id == id)
                        .Select(s => new
                        {
                            Text =
s.PetsOwners.Owners.FirstName + " " +
s.PetsOwners.Owners.LastName + " " +
s.PetsOwners.Owners.Phone
                                    + " " +
s.PetsOwners.Pets.Name + " " +
s.PetsOwners.Pets.Specie + " " +
s.PetsOwners.Pets.Breed + " вагою " +
s.PetsOwners.Pets.Weight,
                            Value = s.Id
                        })
                        .ToList();
        // POST: Receptions/Delete/5
        [Authorize]
        [HttpPost,
ActionName("Delete")]
        [ValidateAntiForgeryToken]
        public async Task<ActionResult>
DeleteConfirmed(int id)
        {
            var currUser =
db.Users.ToArrayAsync().Result.Where(x
=>
x.Email.Equals(User.Identity.Name)).ToL
ist().FirstOrDefault();
            Reception reception = await
db.Receptions.FindAsync(id);
            if
(reception.ApplicationUser_Id.Equals(cu
rrUser.Id))
            {

db.Receptions.Remove(reception);
                await
db.SaveChangesAsync();
            }
            return
RedirectToAction("Index");
        }

        [Authorize]
        protected override void
Dispose(bool disposing)
        {
            if (disposing)
            {
                db.Dispose();
            }
            base.Dispose(disposing);
        }
    }
}
```

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАТВЕРДЖУЮ
Перший проректор Українського
державного університету
науки і технологій
Анатолій РАДКЕВИЧ
10.06.22

САЙТ ВЕТЕРИНАРНОЇ КЛІНІКИ

Опис програми
ЛИСТ ЗАТВЕРДЖЕННЯ
44165850.01254-01 13 01-ЛЗ

Представники
підприємства-розробника
Завідувач кафедри КІТ
Вадим ГОРЯЧКІН
10.06.22

Керівник розробки
Олександр ІВАНОВ
10.06.22

Виконавець
Микита КРОЛЬ
10.06.22

Нормоконтроллер
Олена КУРОП'ЯТНИК
10.06.22

ЗАТВЕРДЖЕНО
44165850.01254-01 13 01-ЛЗ

САЙТ ВЕТЕРИНАРНОЇ КЛІНІКИ

Опис програми

44165850.01254-01 13 01

Листів 40

2022

АНОТАЦІЯ

Документ 44165850.01254-01 13 01 «РОЗРОБКА САЙТУ ВЕТЕРИНАРНОЇ КЛІНІКИ. Опис програми» входить до складу програмної документації на програму, яка представляє собою сайт особистого кабінету лікаря.

У даному документі представленні опис програми та її функціональних можливостей.

ЗМІСТ

1 ЗАГАЛЬНІ ВІДОМОСТІ

Програмний продукт «Особистий кабінет ветеринара», що розробляється призначений для керування записами про пацієнтів ветеринарної клініки для окремо взятого лікаря.

Програмне забезпечення необхідне для функціонування:

- інтернет-браузер;
- бібліотека для мови C# – .Net framework v.4+;
- фреймворк для мови C# – Entity Framework v.6+.

Мови програмування:

- C#;
- javascript.

2 ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ

Функціональне призначення – організація роботи ветеринарів за допомогою інтернет-сайту.

Функціональні обмеження: сайт працює без помилок у браузерах Google Chrome v.98+ та Microsoft Edge v.98+, робота у інших браузерах потребує детальнішого тестування.

3 ОПИС ЛОГІЧНОЇ СТРУКТУРИ

3.1 Алгоритм програми

Опису алгоритму програми можливо звести до виконання таких дій:

- користувач викликає подію на сторінці;
- подія сторінки викликає обробку цієї події контролером;
- контролер аналізує отриману модель або видає модель в залежності від типу методу (get – отримати дані, post – відправити);
- аналізуючи модель або формуючи відповідь на подію контролер зв’язується з базою даних, отримує необхідні дані, або відправляє їх;
- по завершенню роботи з даними, контролер повертає відповідь представленню у вигляді посилання на іншу подію, або заповненого списку моделей (моделі).

3.2 Використані методи

Архітектура системи побудована згідно методології архітектурного патерну Model View Controller. Описуючи дану методологію можливо виділити те, що в програмі існує три логічні рівні, кожен з яких виконує тільки свою функцію.

Представлення (View) – відображають дані користувача, та забезпечують їх введення й за можливістю перевірку коректності введених даних.

Моделі (Model) – представляють собою контейнери даних, які використовуються для зв’язку представлення з контролером та контролера з базою даних.

Контролер (Controller) – виконують роль посередника між іншими рівнями проекту, вони призначені для обробки подій на сторінці користувача, відправки даних до бази даних, додаткові перевірки даних, також контролери можуть використовуватись для обмеження доступу користувача до певних функцій сайту. Завдяки вбудованим функціям C# Identity ASP.Net, програміст може легко обмежувати доступ до даних для користувачів, оскільки існують спеціальні атрибути доступу (наприклад атрибут [Authorize], що виконує перевірку, чи авторизувався

користувач в системі, якщо він не авторизований, а метод має такий атрибут, то користувач буде повернений на сторінку вказану за замовченням).

3.3 Структура програми

Структуру проєкту буде відображено за допомогою діаграми класів на рис. 3.1 – 3.9.

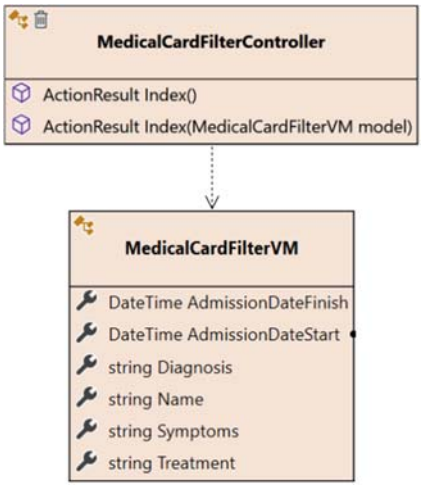

Рисунок 3.1 – Діаграма класів, зв’язок контролеру фільтрації даних медичних карт

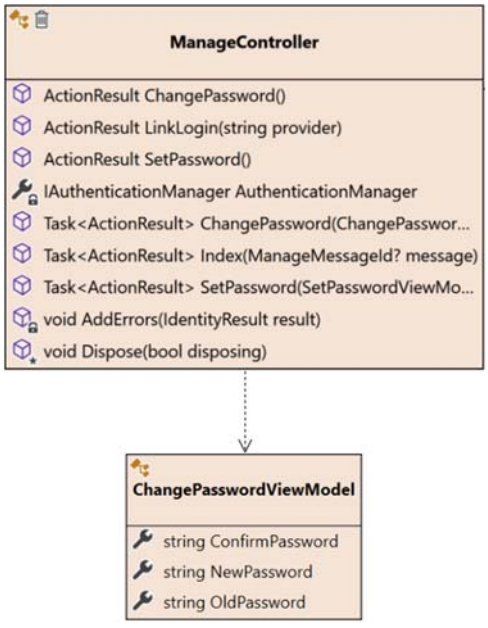

Рисунок 3.2 – Діаграма класів, зв’язки контролера роботи зі сторінкою зміни паролю

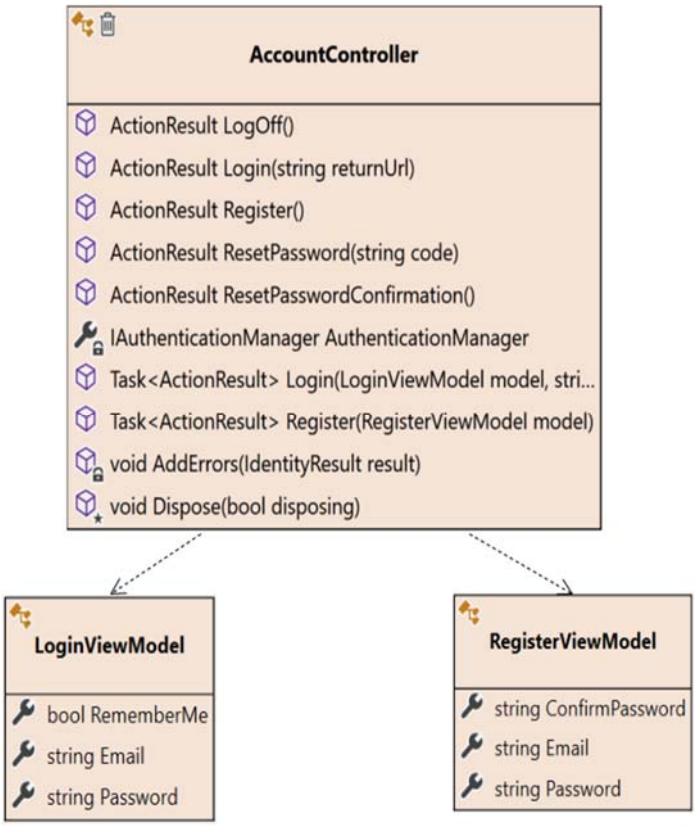

Рисунок 3.3 – Діаграма класів, зв’язки контролера роботи зі сторінками авторизації та реєстрації

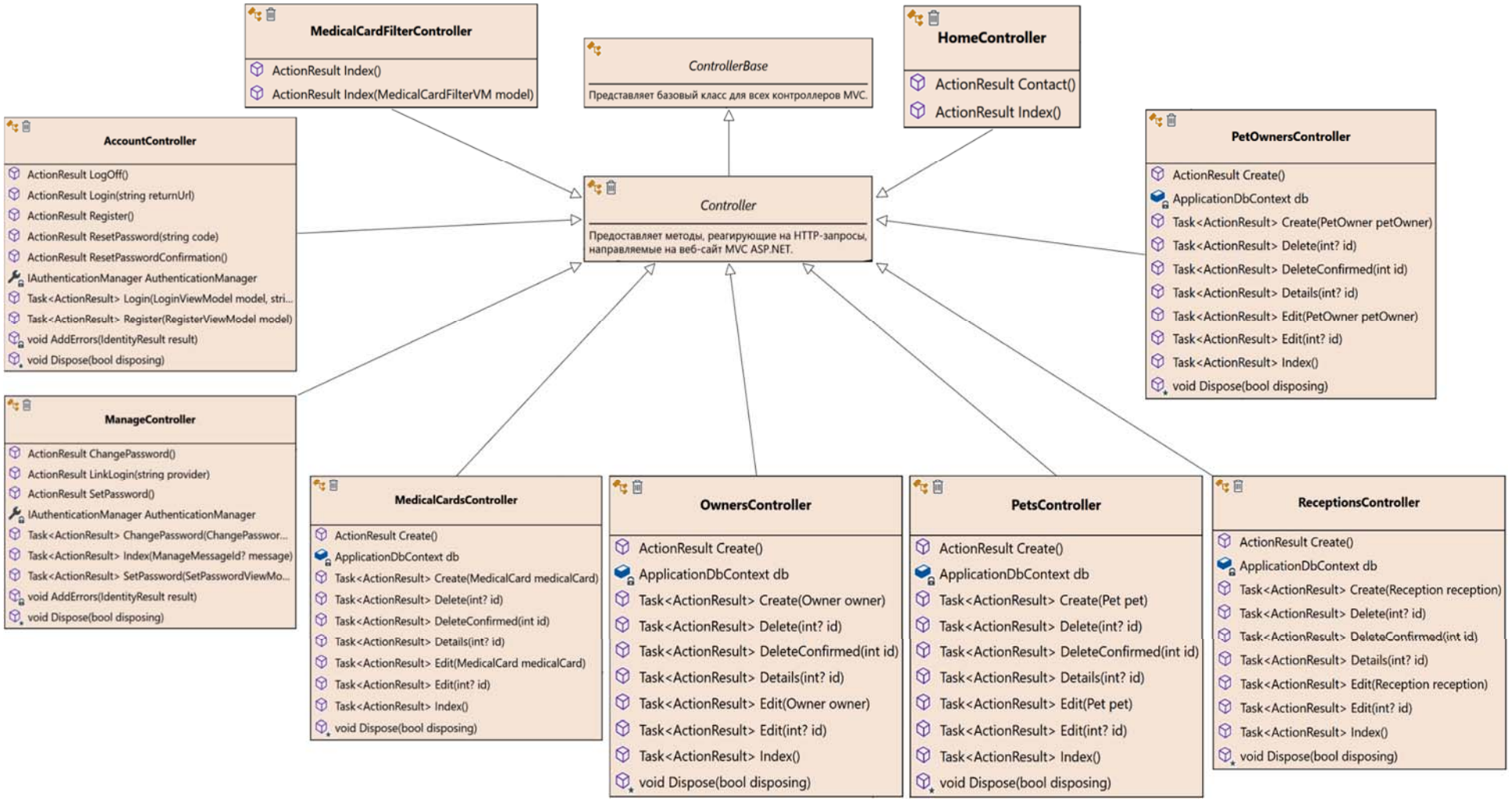

Рисунок 3.4 – Діаграма класів, зв’язки контролерів з їх базовим класом

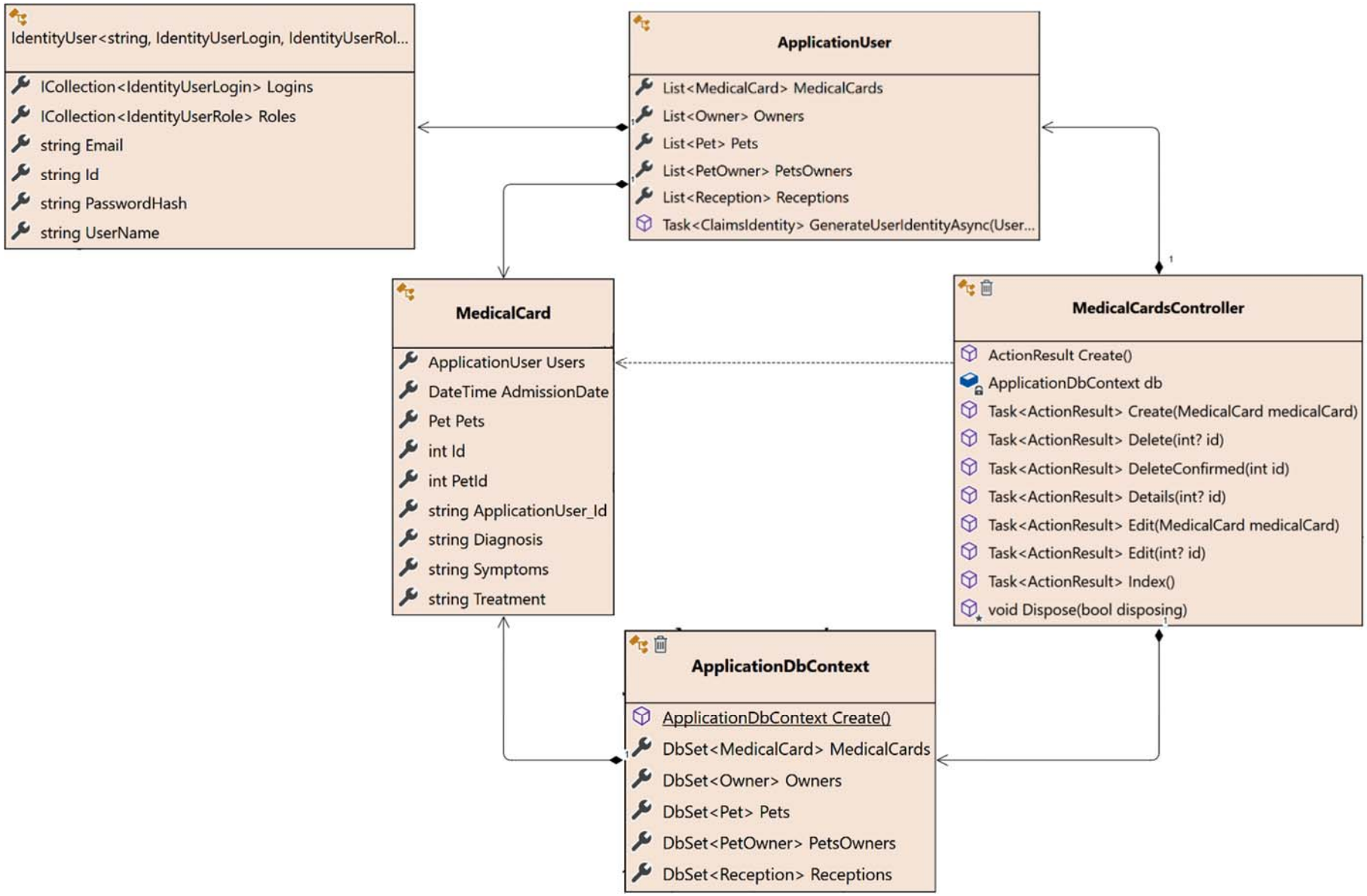

Рисунок 3.5 – Діаграма класів, зв’язки контролера роботи з даними медичних карт

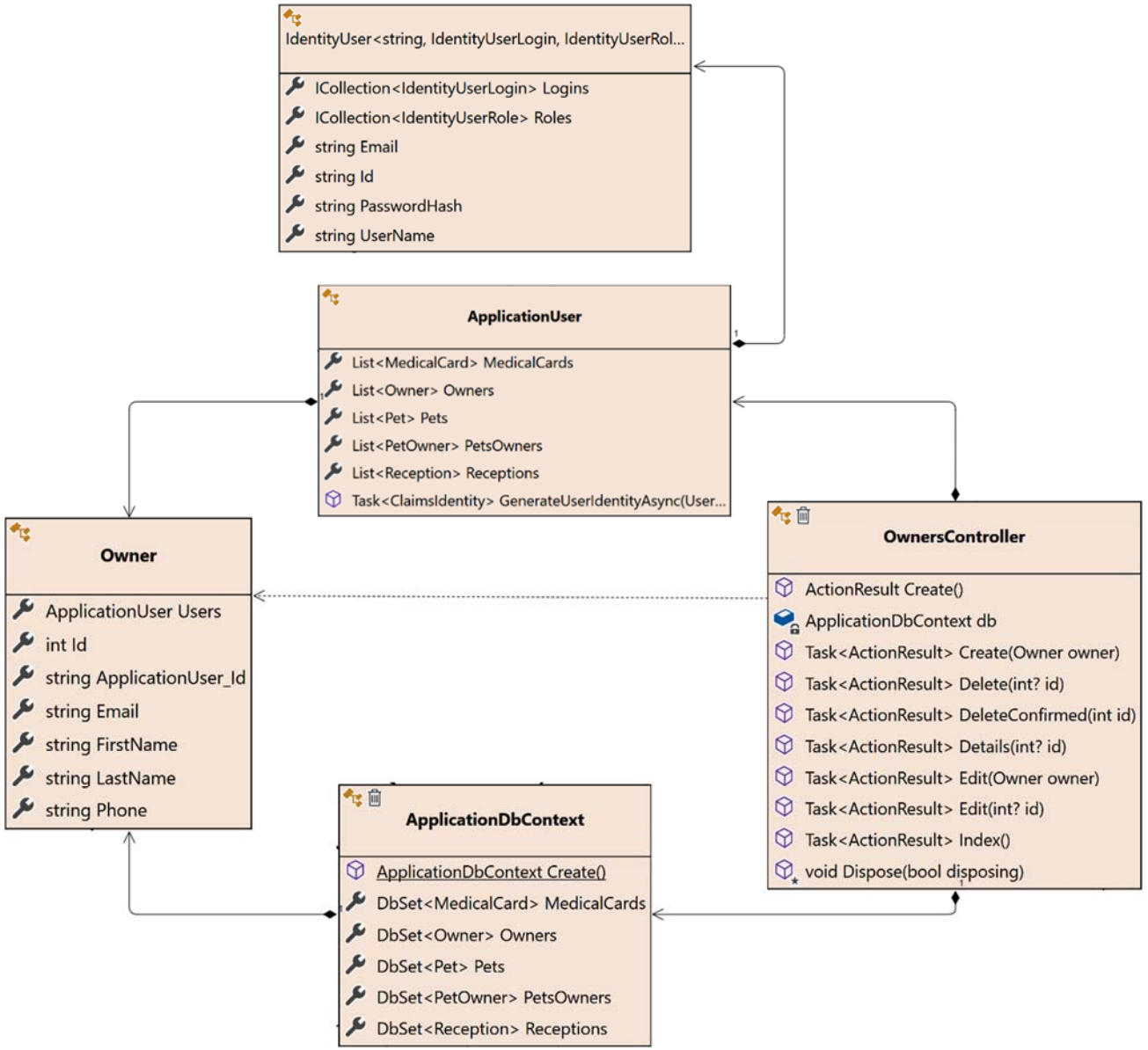

Рисунок 3.6 – Діаграма класів, зв’язки контролера для роботи з даними господарів

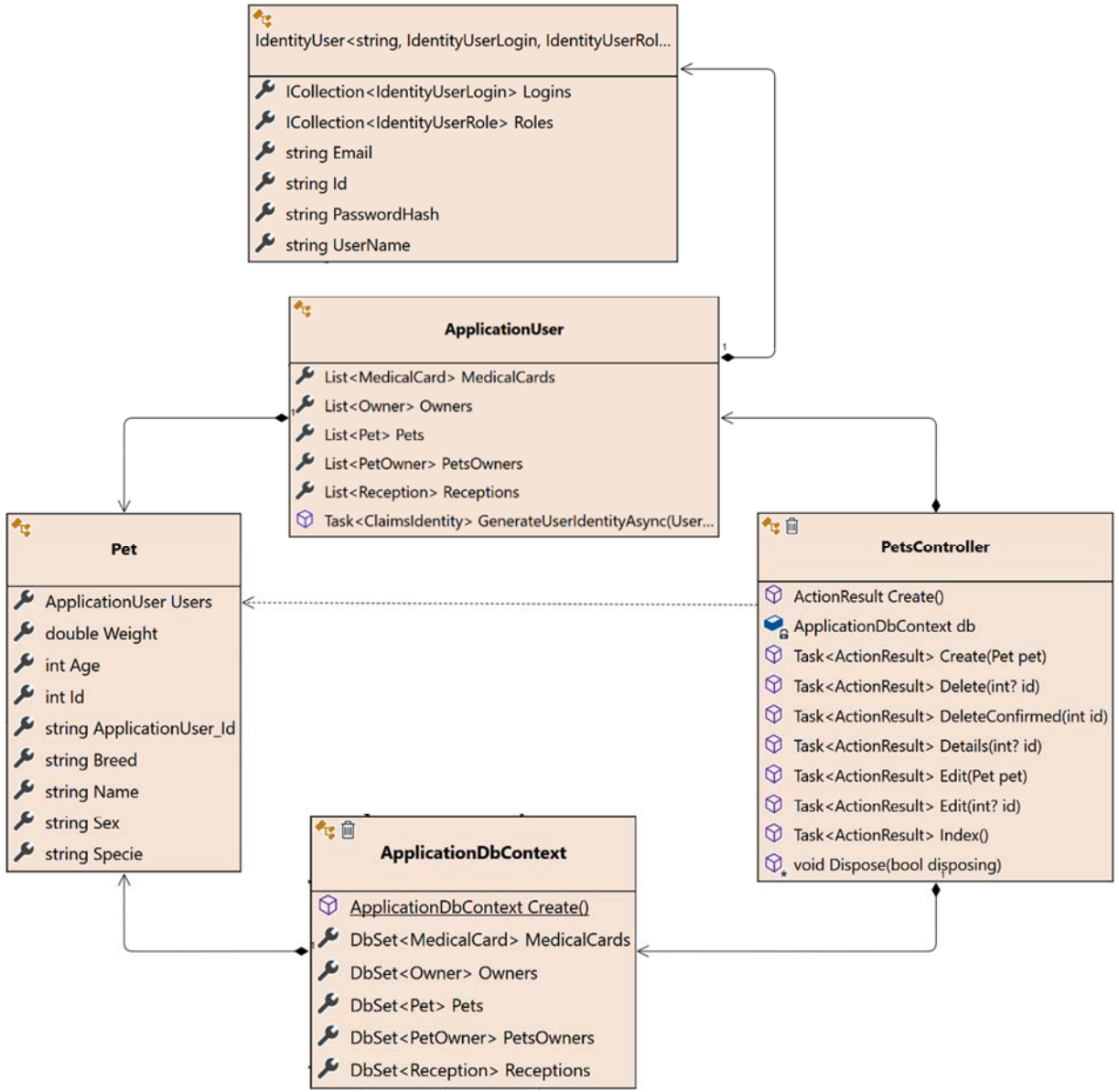

Рисунок 3.7 – Діаграма класів, зв’язки контролера для роботи з даними тварин

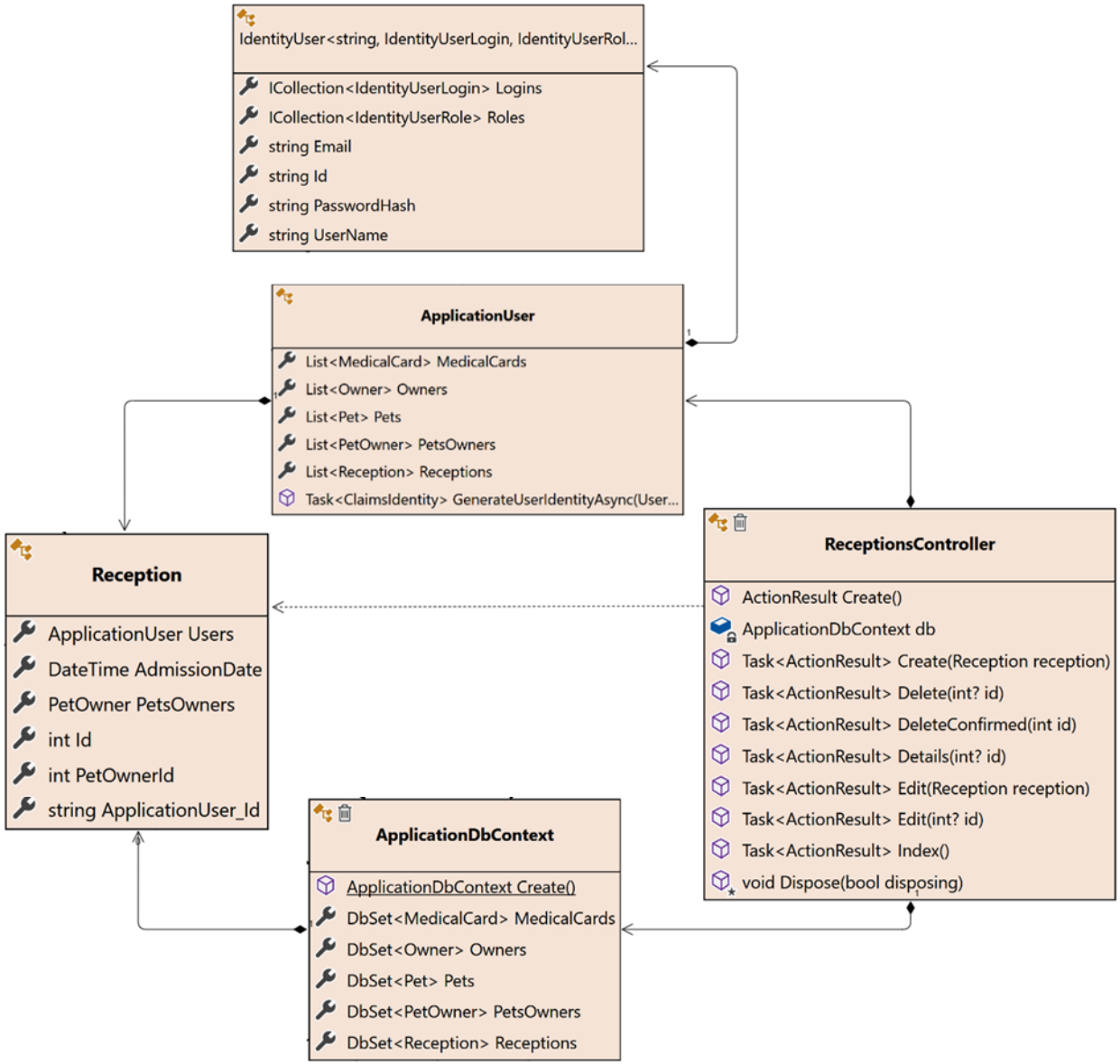

Рисунок 3.8 – Діаграма класів, зв’язки контролера для роботи з візитів

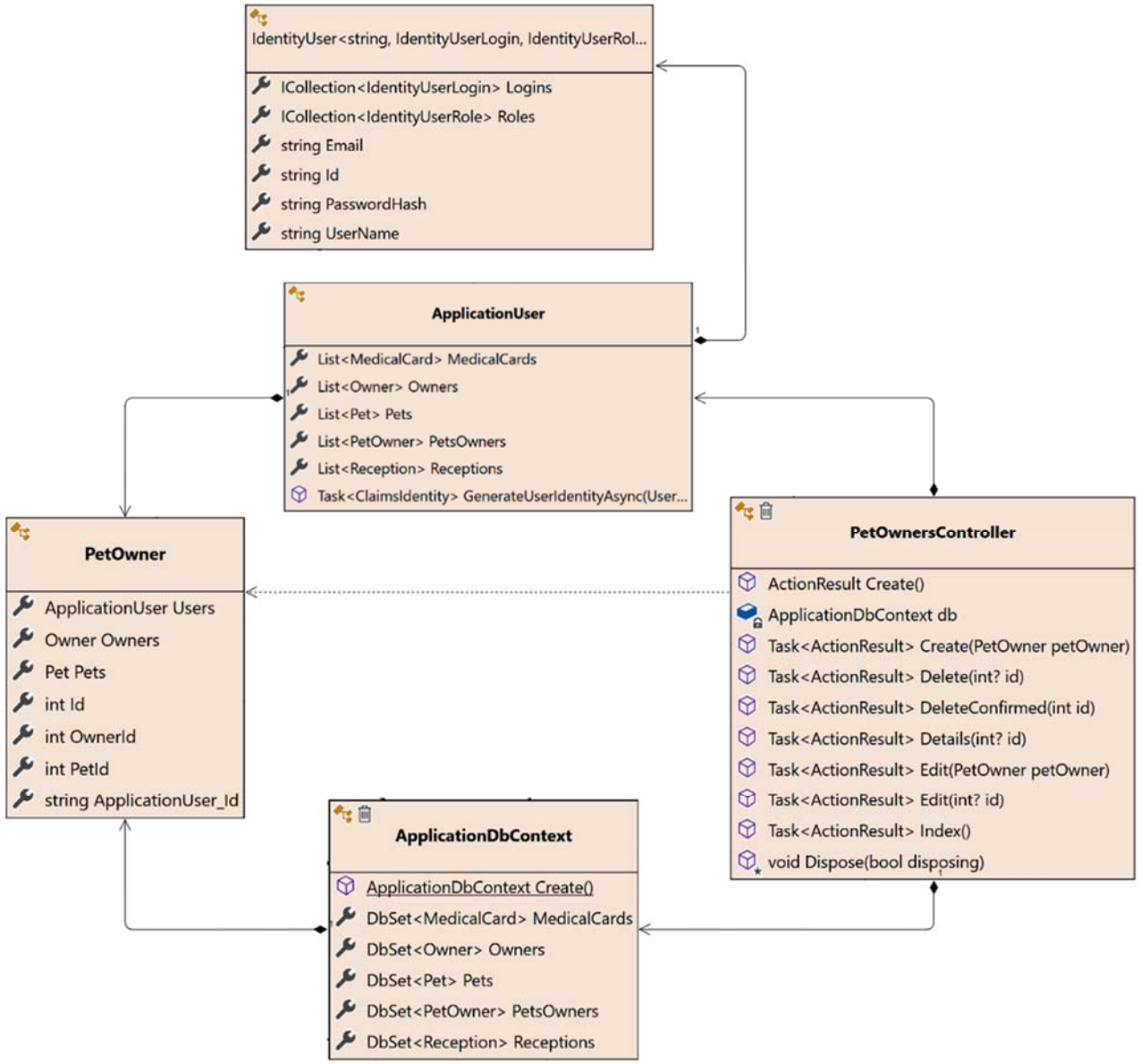

Рисунок 3.9 – Діаграма класів, зв’язки контролера для роботи з даними про зв'язок госпосодарів з тваринами

Призначення класів:

- ApplicationDbContext – звернення до бази даних для отримання результатів відправлених запитів;
- ApplicationUser – модель збереження даних бази даних;
- IdentityUser – контейнер для збереження даних про користувача системи;
- AccountController – керування подіями авторизації та реєстрації;
- HomeController – керування подіями на головній сторінці;
- ManageController – керування зміною паролю;
- MedicalCardFilterController – керування фільтрацією даних;
- MedicalCardsController – керування подіями зони медичних карт;
- OwnersController – керування подіями зони господарів;
- PetOwnersController – керування подіями зони приналежності улюбленця господарю;
- PetsController – керування подіями зони улюбленців;
- ReceptionsController – керування подіями зони планування візитів;
- MedicalCard – контейнер для збереження даних про медичну карту;
- Owner – контейнер для збереження даних про господарів;
- Pet – контейнер для збереження даних про улюбленців;
- PetOwner – контейнер для збереження даних про приналежність улюбленця господарю;
- Reception – контейнер для збереження даних про майбутній візит;
- MedicalCardFilterVM – контейнер для збереження даних про фільтри медичних карт;
- AccountViewModels – контейнер для збереження даних про користувача системи.

3.4 Зв’язки програми з іншими програмами

Програма при роботі розгортається на спеціальному веб-сервері – IIS (Internet Information Services), який дозволяє розміщувати в інтернеті сайти.

4 ВИКОРИСТАНІ ТЕХНІЧНІ ЗАСОБИ

Програмний продукт використовується на ЕОМ, у складі програмного забезпечення яких є браузер.

Також програмний продукт використовується на мобільних пристроях, що мають такі характеристики:

- 2 гігабайти оперативної пам’яті;
- 16 гігабайт доступної пам’яті;
- операційну систему android v.10+.

Сервер, на якому розгортається сайт, повинен мати такі характеристики:

- 2 гігабайти оперативної пам’яті;
- процесор: 32 або 64 розрядний процесор із тактовою частотою 1 ГГц або вище з набором інструкцій SSE2;
- 100 гігабайт простору на жорсткому диску.

5 ВИКЛИК ЗАВАНТАЖЕННЯ

Програмний продукт було розміщено на хостингу, й отримати доступ до сайту можна за посиланням: http://nekitkroi-001-site1.gtempurl.com/.

6 ВХІДНІ ДАНІ

Вхідні дані проекту представлено у вигляді таблиць відношень бази даних (табл. 6.1 – 6.6).

Таблиця 6.1 – Відношення Owners

№	Стовпець	Опис поля	Тип даних	Розмір	Ключ
1.	Id	Унікальний код	int identity	4	+
2.	firstName	Імя	ntext	2147483646	-
3.	lastName	Прізвище	ntext	2147483646	-
4.	phone	Телефон	ntext	2147483646	-
5.	Email	Електронна пошта	ntext	2147483646	-
6.	ApplicationUser_Id	Унікальний код власника запису (користувача)	nvarchar	256	-

Таблиця 6.2 – Відношення AspNetUsers

№	Стовпець	Опис поля	Тип даних	Розмір	Ключ
1.	Id	Унікальний код	nvarchar	256	+
2.	Email	Електронна пошта	nvarchar	512	-
3.	EmailConfirmed	Статус підтвердження пошти	bit	1	-
4.	PasswordHash	Пароль	ntext	2147483646	-
5.	SecurityStamp	Поточний стан користувача для автоматичного входу у систему через сторонні сервіси	ntext	2147483646	-
6.	PhoneNumber	Телефон	ntext	2147483646	-
7.	PhoneNumberConfirmed	Статус підтвердження телефону	bit	1	-

Таблиця 6.3 – Відношення MedicalCards

№	Стовпець	Опис поля	Тип даних	Розмір	Ключ
1.	Id	Унікальний код	int identity	4	+
2.	PetId	Код улюбленця	int	4	-
3.	Symptoms	Опис симптомів	ntext	2147483646	-
4.	Diagnosis	Опис діагнозів	ntext	2147483646	-
5.	Treatment	Опис лікування	ntext	2147483646	-
6.	Admission Date	Дата візиту	datetime	16	-
7.	Application User_Id	Унікальний код власника запису (користувача)	nvarchar	256	-

Таблиця 6.4 – Відношення PetOwners

№	Стовпець	Опис поля	Тип даних	Розмір	Ключ
1.	Id	Унікальний код	int identity	4	+
2.	PetId	Код улюбленця	int	4	-
3.	OwnerId	Код власника	int	4	-
4.	Application User_Id	Унікальний код власника запису (користувача)	nvarchar	256	-

Таблиця 6.5 – Відношення Pets

№	Стовпець	Опис поля	Тип даних	Розмір	Ключ
1.	Id	Унікальний код	int identity	4	+
2.	Name	Кличка	ntext	2147483646	-
3.	Breed	Порода	ntext	2147483646	-
4.	Specie	Вид	ntext	2147483646	-
5.	Age	Вік	int	4	-
6.	Sex	Стать	ntext	2147483646	-
7.	Weight	Вага	float	8	-
8.	Application User_Id	Унікальний код користувача (власника запису)	nvarchar	256	-

Таблиця 6.6 – Відношення Receptions

№	Стовпець	Опис поля	Тип даних	Розмір	Ключ
1.	Id	Унікальний код	int identity	4	+
2.	PetOwnerId	Код зв’язку улюбленця та власника	int	4	-
3.	Admission Date	Дата візиту	datetime	16	-
4.	Application User_Id	Унікальний код користувача (власника запису)	nvarchar	256	-

7 ВИХІДНІ ДАНІ

Вихідні дані програмного продукту:

- сформовані представлення для користувача;
- повідомлення системи;
- дані бази даних.

8 ОПИС ПРИЗНАЧЕНОГО ДЛЯ КОРИСТУВАЧА ІНТЕРФЕЙСУ

8.1 Опис станів програми

Стани, в яких може знаходитись програма наведено у табл. 8.1.

Таблиця 8.1 – Стани програми

№	Стан	Опис стану	Рекомендовані дії
1	2	3	4
1	Завантаження сторінки	Майже кожна дія користувача призводить до перезавантаження сторінки	Почекати завантаження сторінки
2	Відправка запиту (подія на сайті)	Виконуючи дії на сайті користувач створює запити, які відправляються до контролеру	Почекати завантаження сторінки
3	Обробка запиту	Після отримання запиту сервер аналізує події, що викликав користувач й реагує на неї виконуючи певні маніпуляції з даними, внаслідок обробки яких, відбувається формування результатів	Почекати завантаження сторінки
4	Формування результатів запиту	Сформувавши відповідь серверна частина надсилає користувачу її у вигляді списку моделей, або нового представлення, або посилання на іншу подію	Почекати завантаження сторінки
5	Помилка завантаження сторінки (404)	При переході за вказаним URL було отримано помилку про неможливість віднайти таку сторінку	Повторити виконані дії, або написати до підтримки
6	Помилка виконання запиту	При виконанні запиту було виявлено помилку, тому користувач отримав відповідну сторінку з результатом про помилку	Повторити виконані дії, або написати до підтримки

8.2 Опис керування діалогом

Заходячи на сайт користувач бачить початкову сторінку сайту з інформацією про нього. В меню сайту користувач може переглянути контакти, за якими він може звернутись, кнопки авторизації та реєстрації.

Якщо користувач не має облікового запису, він реєструє себе в системі та має можливість авторизації на сайті.

Після входу до особистого кабінету, лікарю відкривається доступ до редагування даних профілю (на поточній ітерації можливо редагувати тільки пароль) та до робочих зон.

Загальний алгоритм роботи з сайтом передбачає, що лікар спочатку створює облікові записи господаря та улюбленця далі переходить до зони приналежності улюбленця до господаря й формує зв'язок між ними.

Після того, як ветеринар створив облікові записи улюбленців та господарів, він може перейти до зони медичних карт, де створити запис про візит господаря з улюбленцем. Також на цій сторінці він може перейти до сторінки фільтрів, де вказати всі необхідні дані для фільтрації медичних карток. Фільтрація виконує пошук даних відповідних до обраних атрибутів, при введені частини слова або окремого речення, результатами фільтрації будуть дані, які містять цей текст. Також при фільтрації обов’язково необхідно вказати початкову та кінцеву дати візитів.

За необхідності лікар створює запис про візит у зоні планування візитів. Важливо, що візит неможливо зберегти, якщо обраний час вже минув. На сторінці візитів відображаються лише майбутні візити та візити за поточний день.

8.3 Формування екранів

Після опису всіх можливостей сайту відобразимо на рисунках 8.1 – 8.31 спроектовані сторінки сайту.

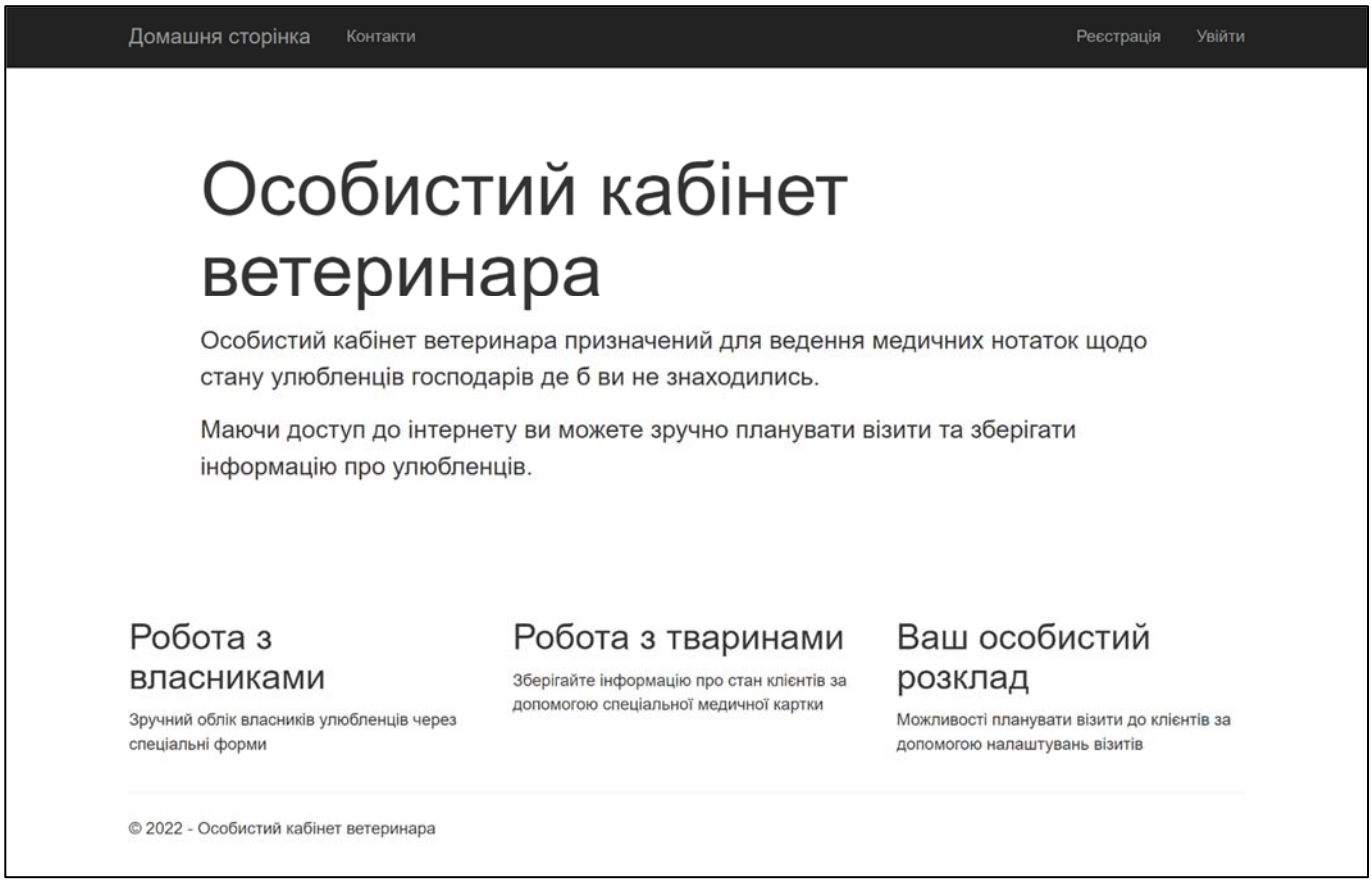

Рисунок 8.1 – Початкова сторінка сайту

Рисунок 8.2 – Сторінка контактів

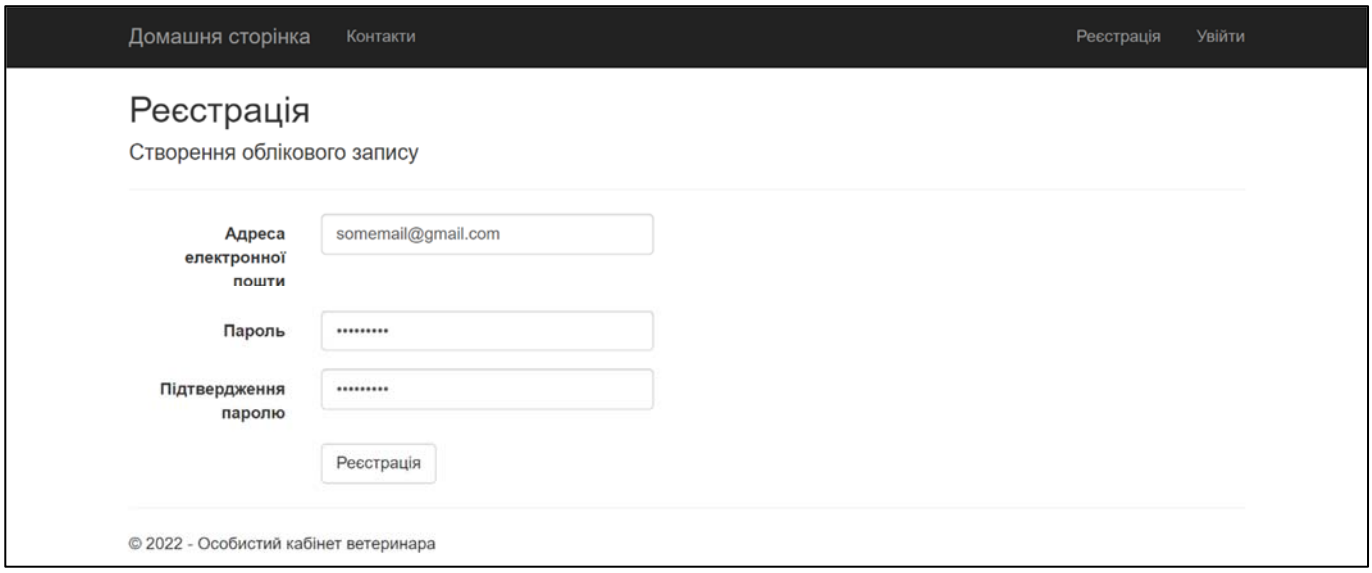

Рисунок 8.3 – Сторінка реєстрації

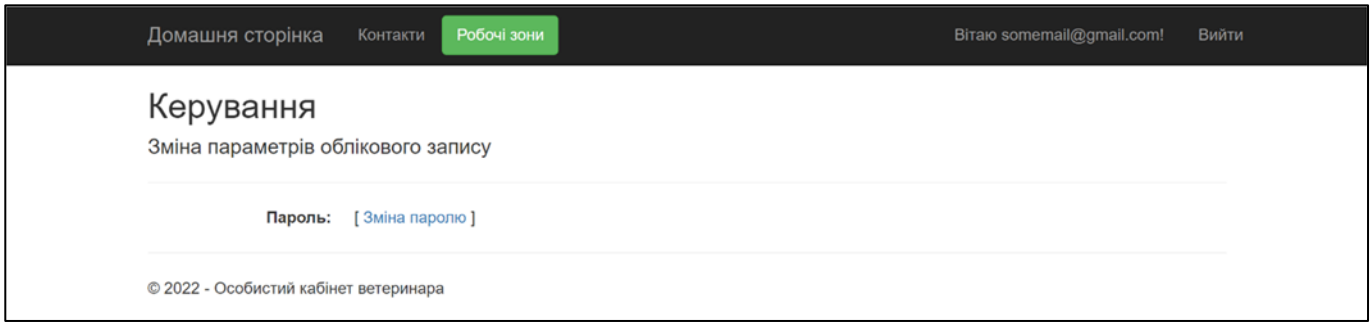

Рисунок 8.4 – Сторінка керування обліковим записом

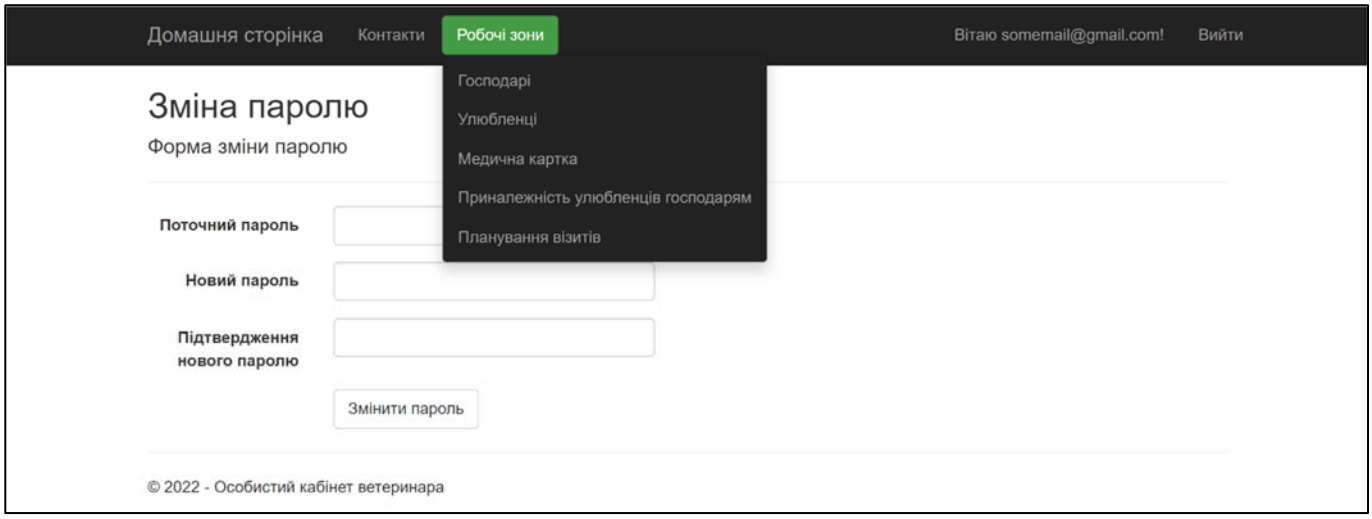

Рисунок 8.5 – Робочі зони

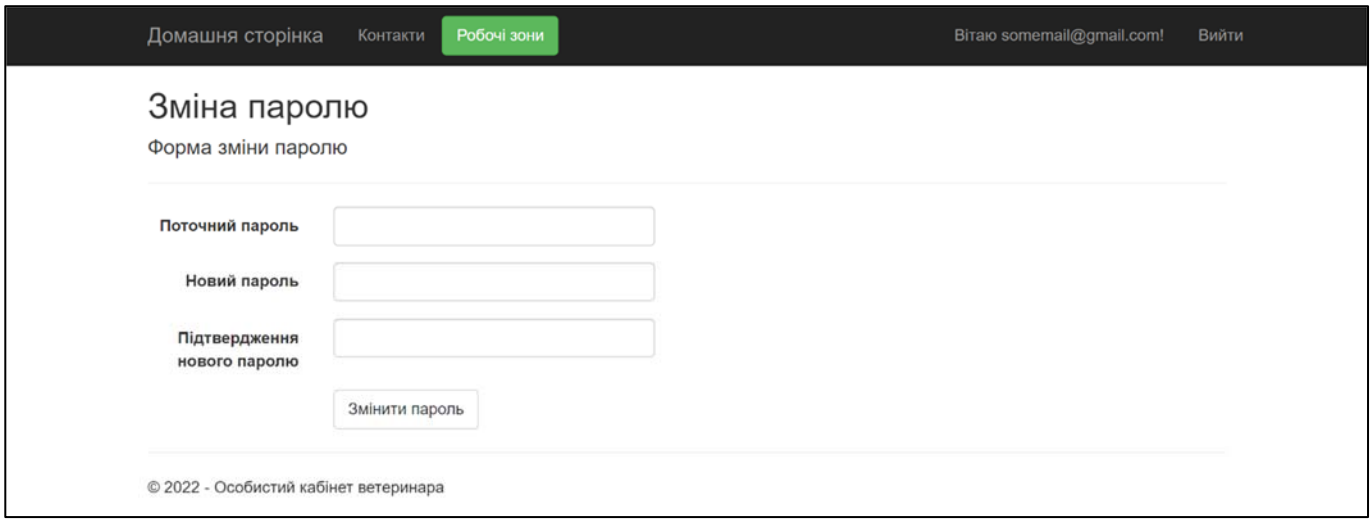

Рисунок 8.6 – Сторінка зміни паролю

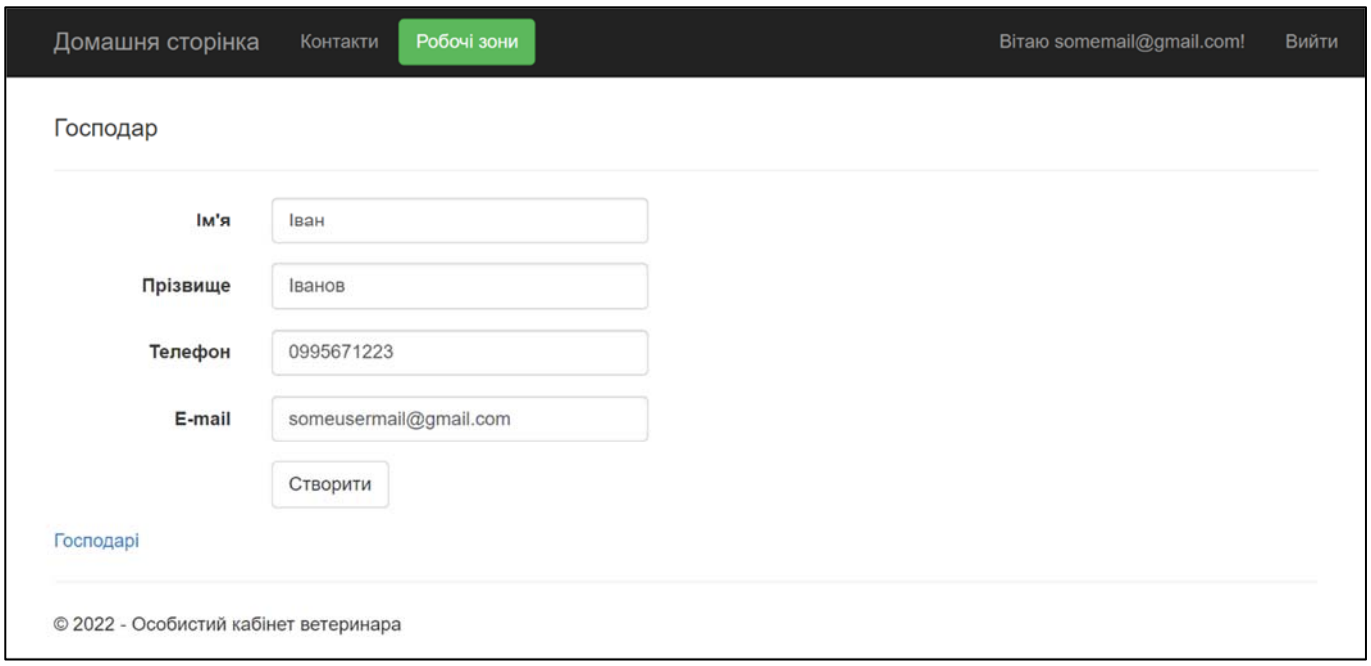

Рисунок 8.7 – Сторінка створення господаря

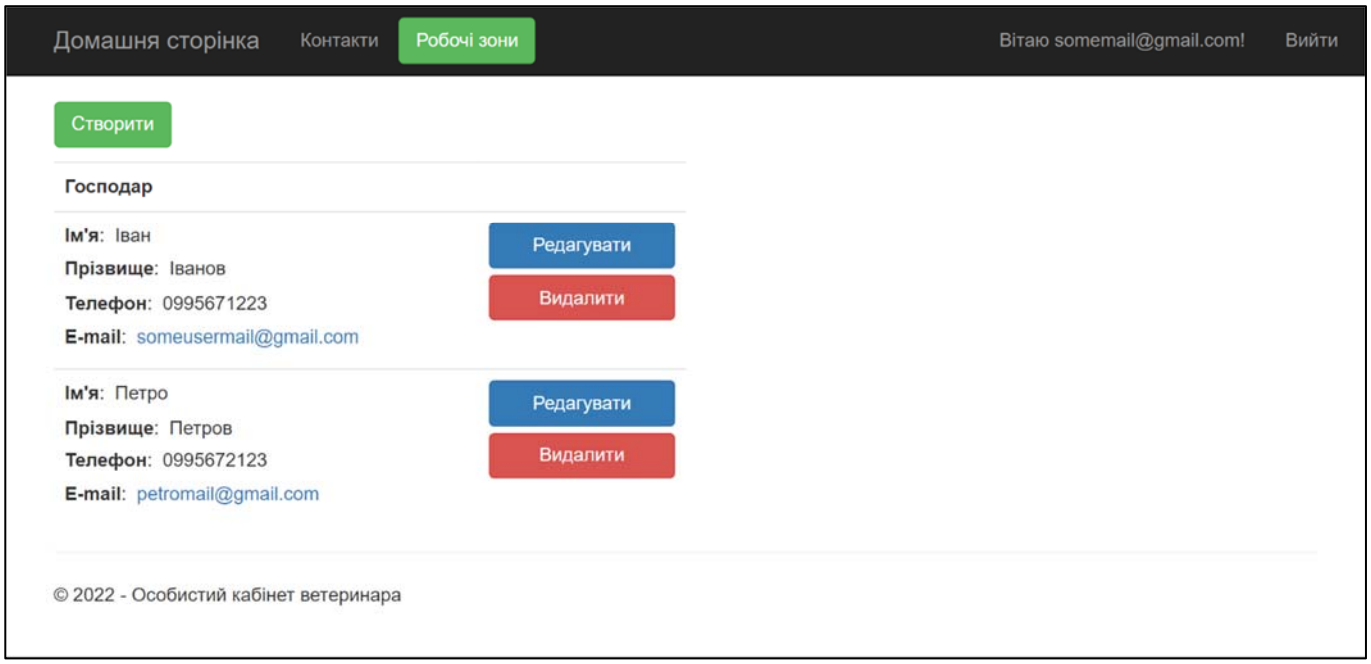

Рисунок 8.8 – Сторінка господарів

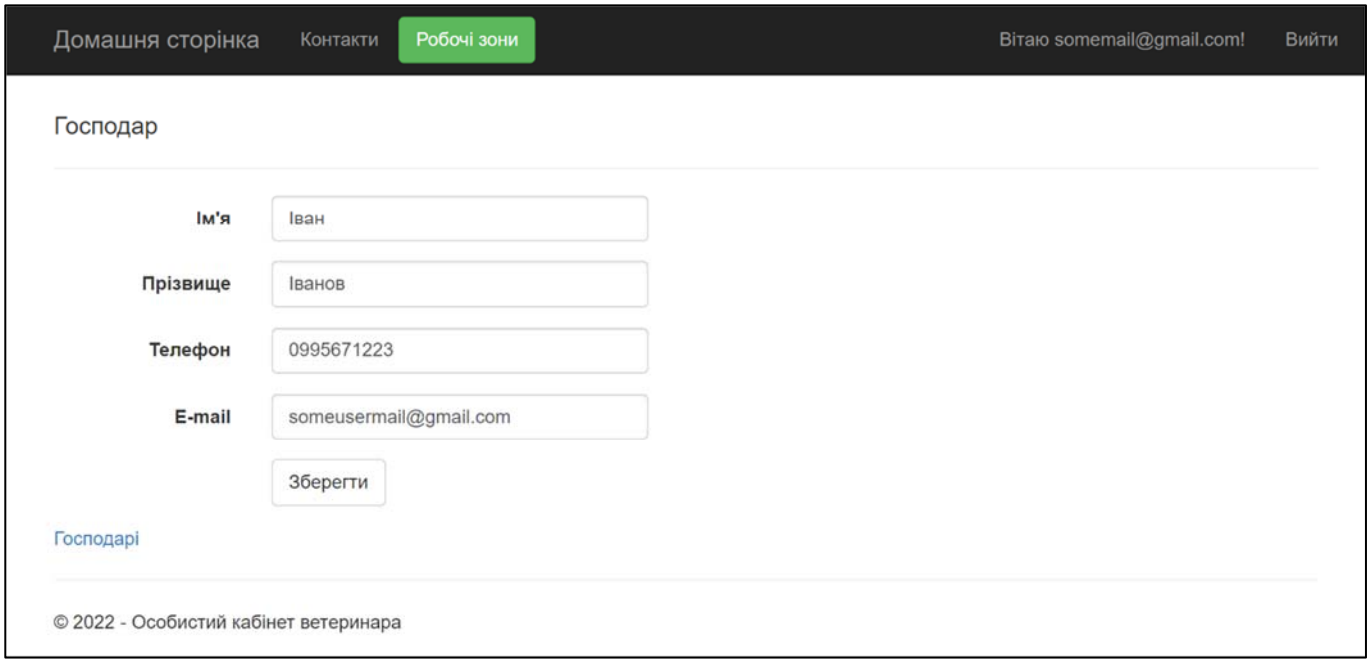

Рисунок 8.9 – Сторінка редагування господаря

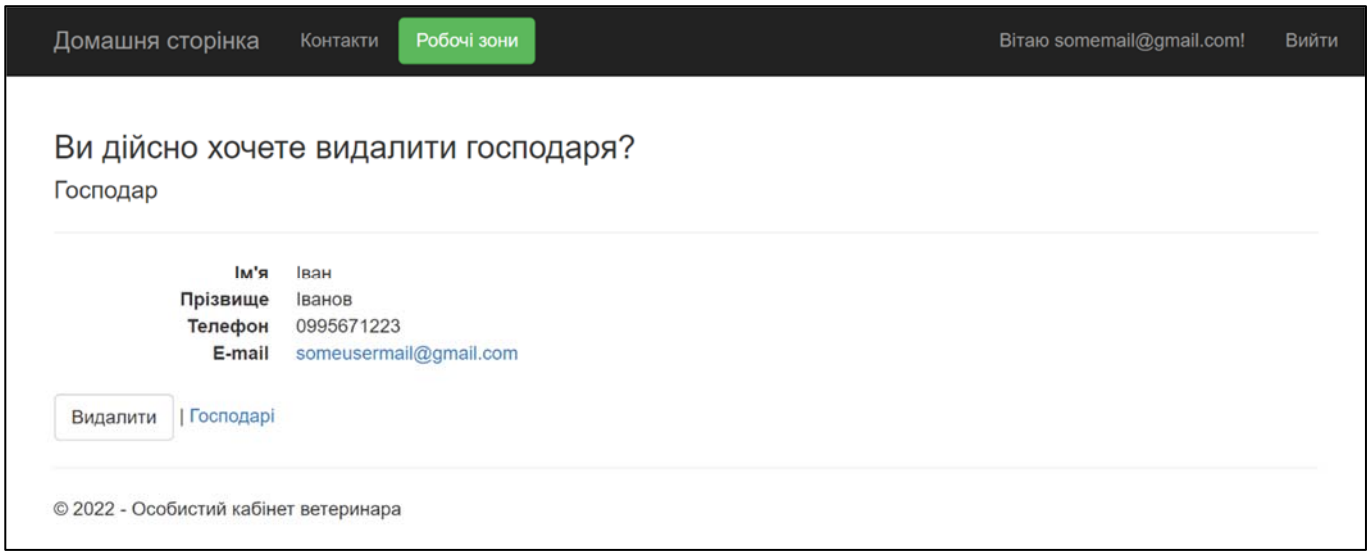

Рисунок 8.10 – Сторінка видалення господаря

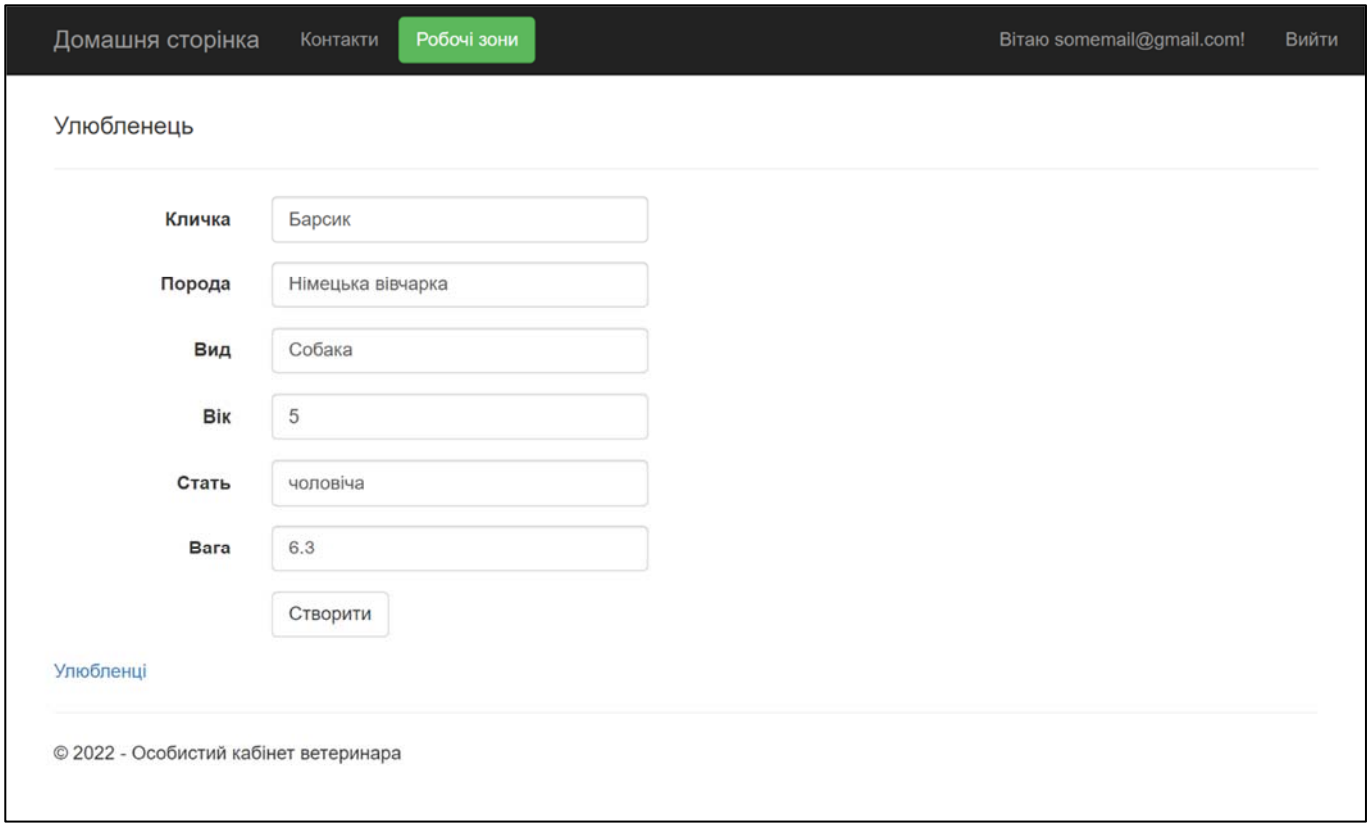

Рисунок 8.11 – Сторінка створення улюбленця

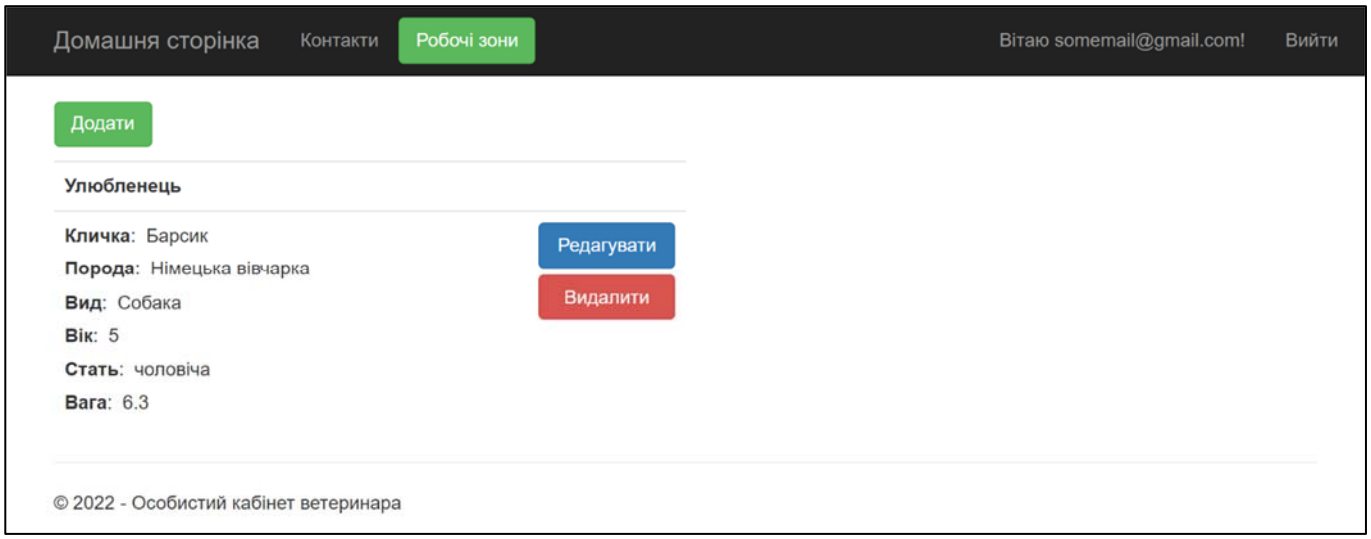

Рисунок 8.12 – Сторінка улюбленців

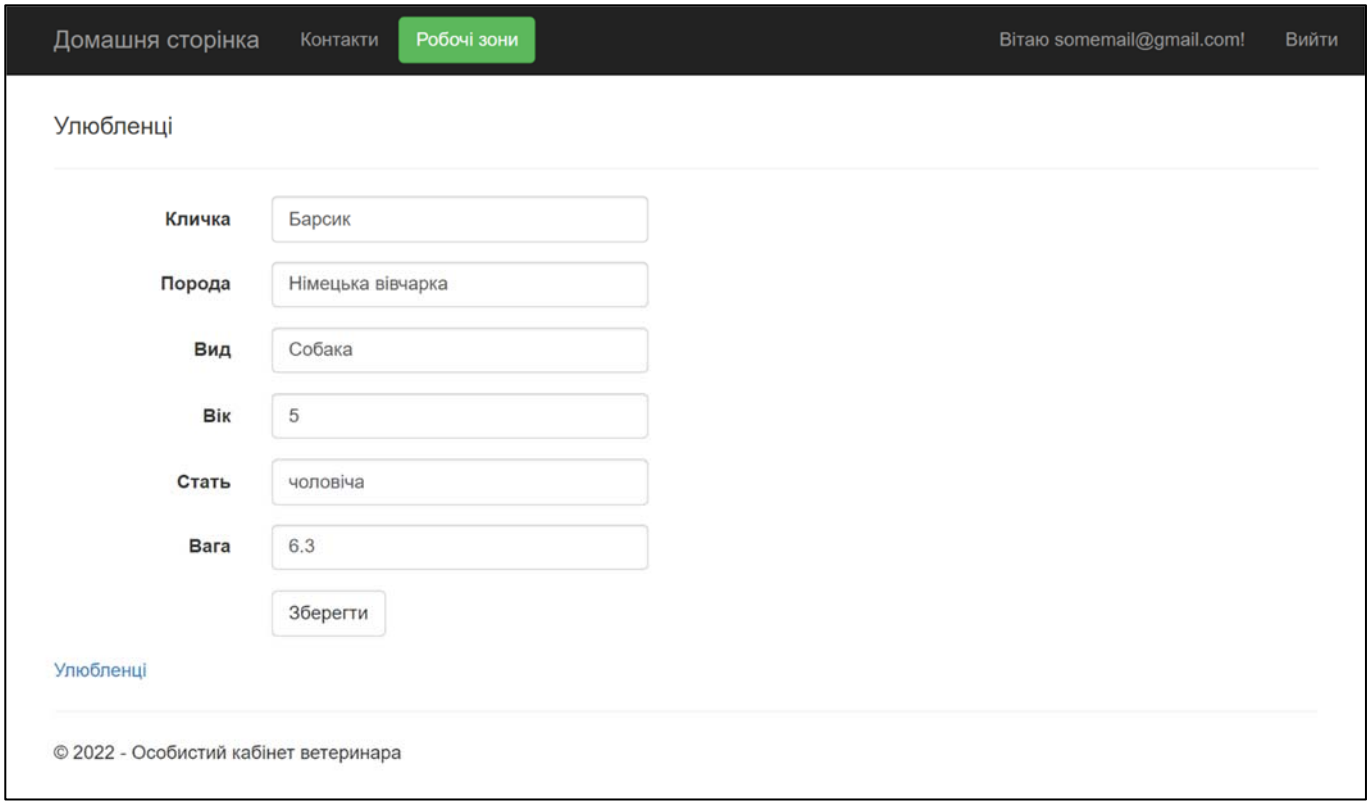

Рисунок 8.13 – Сторінка редагування улюбленця

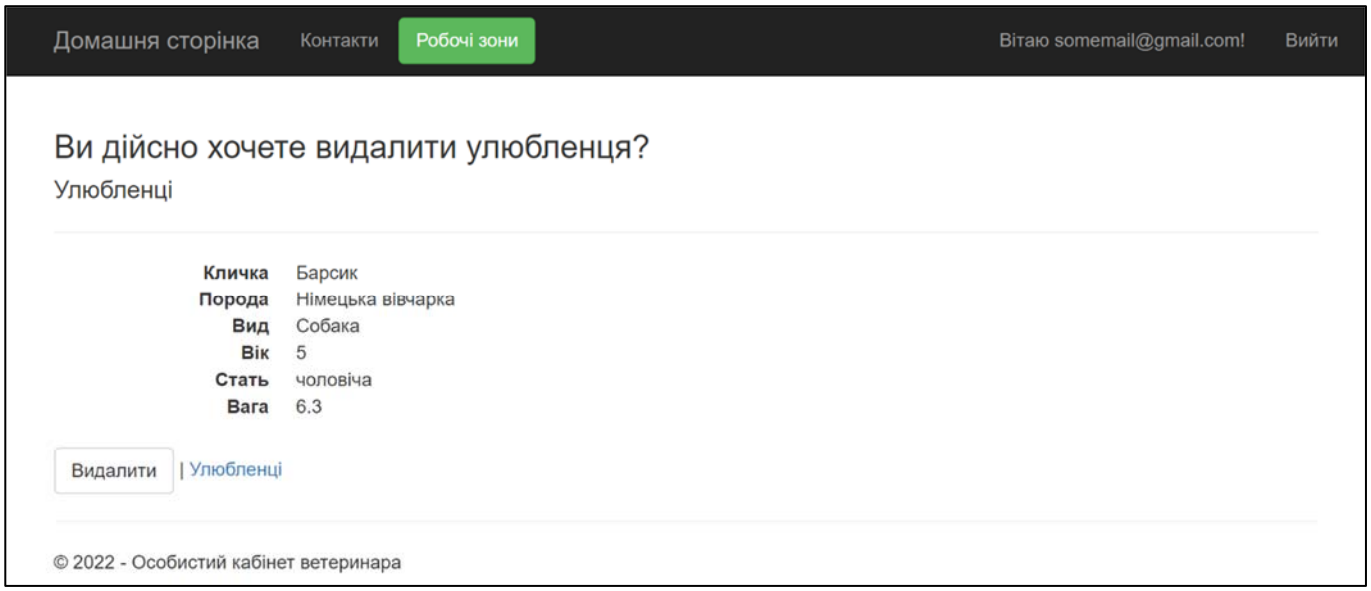

Рисунок 8.14 – Сторінка видалення улюбленця

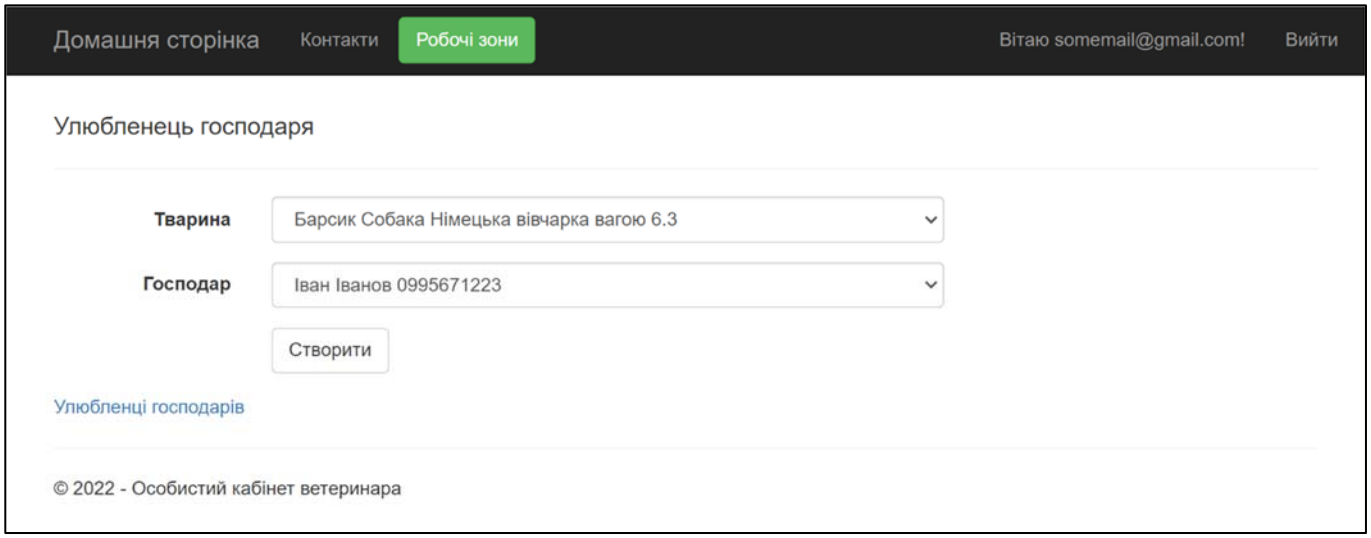

Рисунок 8.15 – Сторінка створення прив’язки тварини до господаря

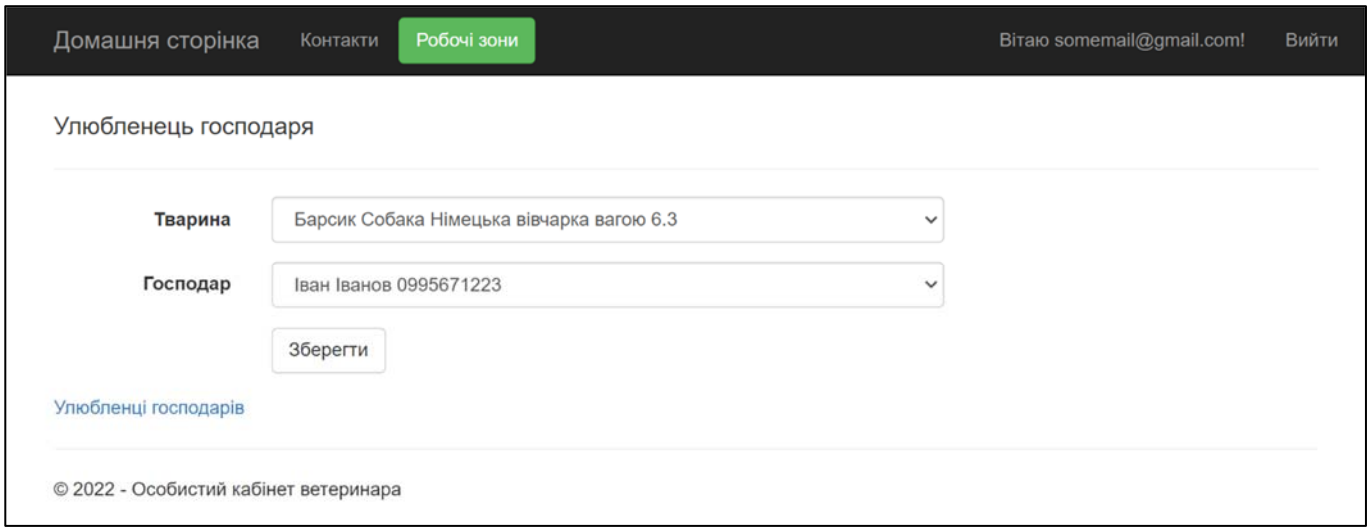

Рисунок 8.16 – Сторінка редагування прив’язки тварини до господаря

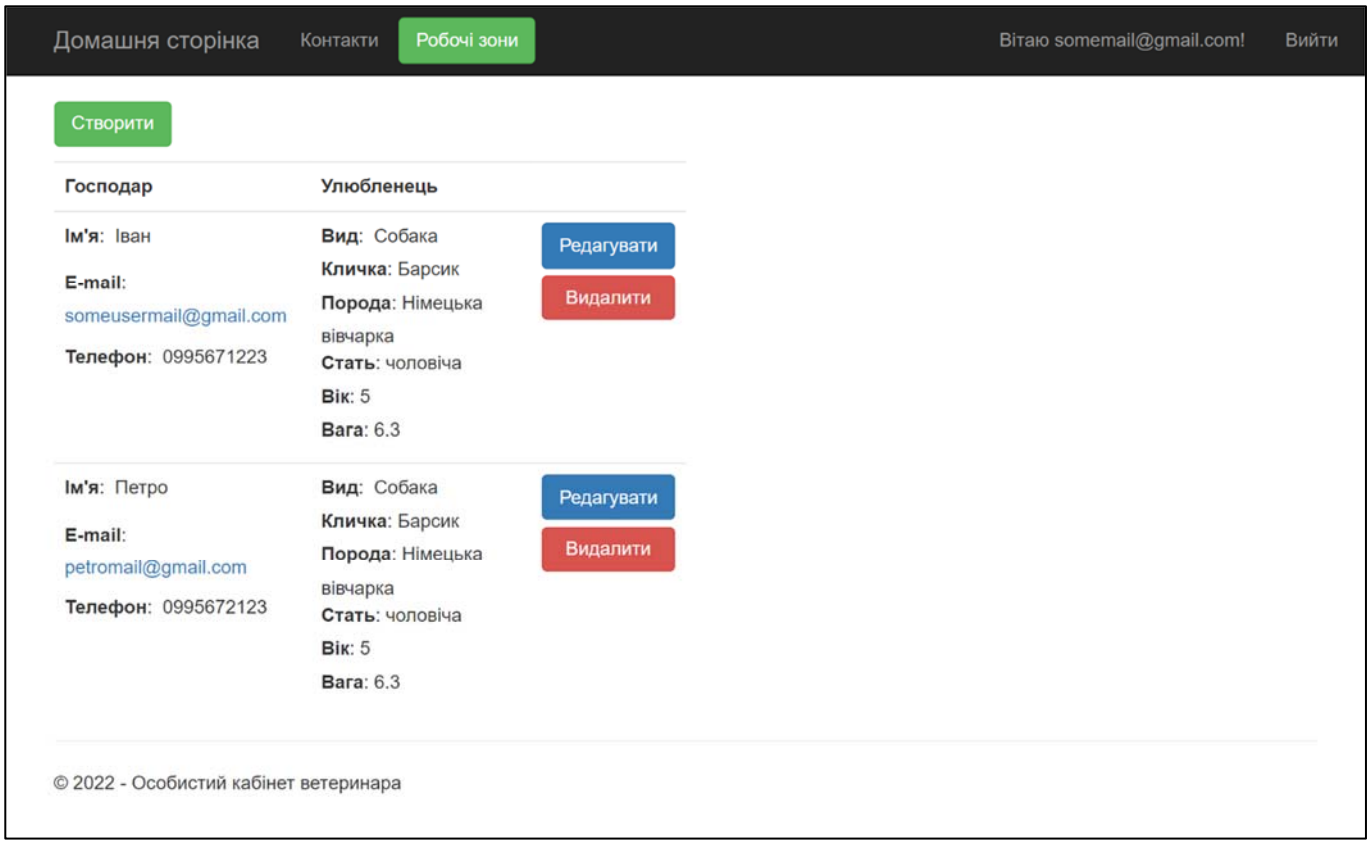

Рисунок 8.17 – Сторінка господарів та їх тварин

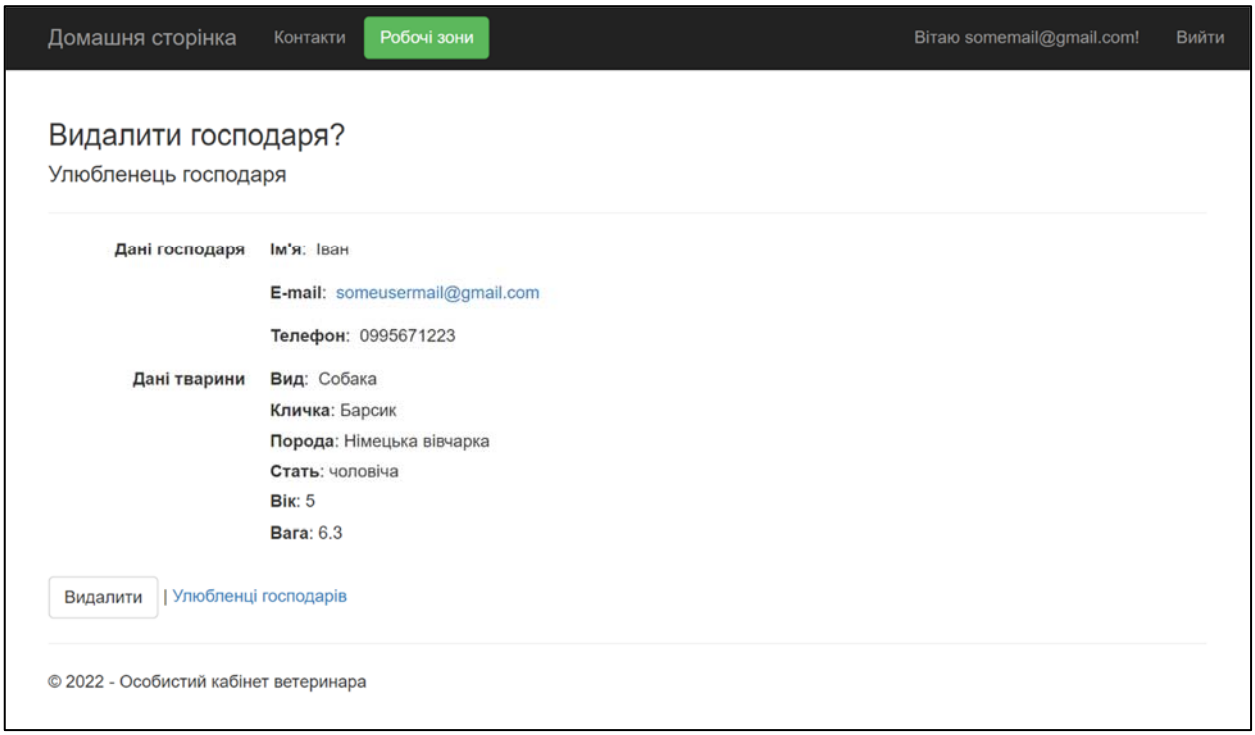

Рисунок 8.18 – Сторінка видалення прив’язки

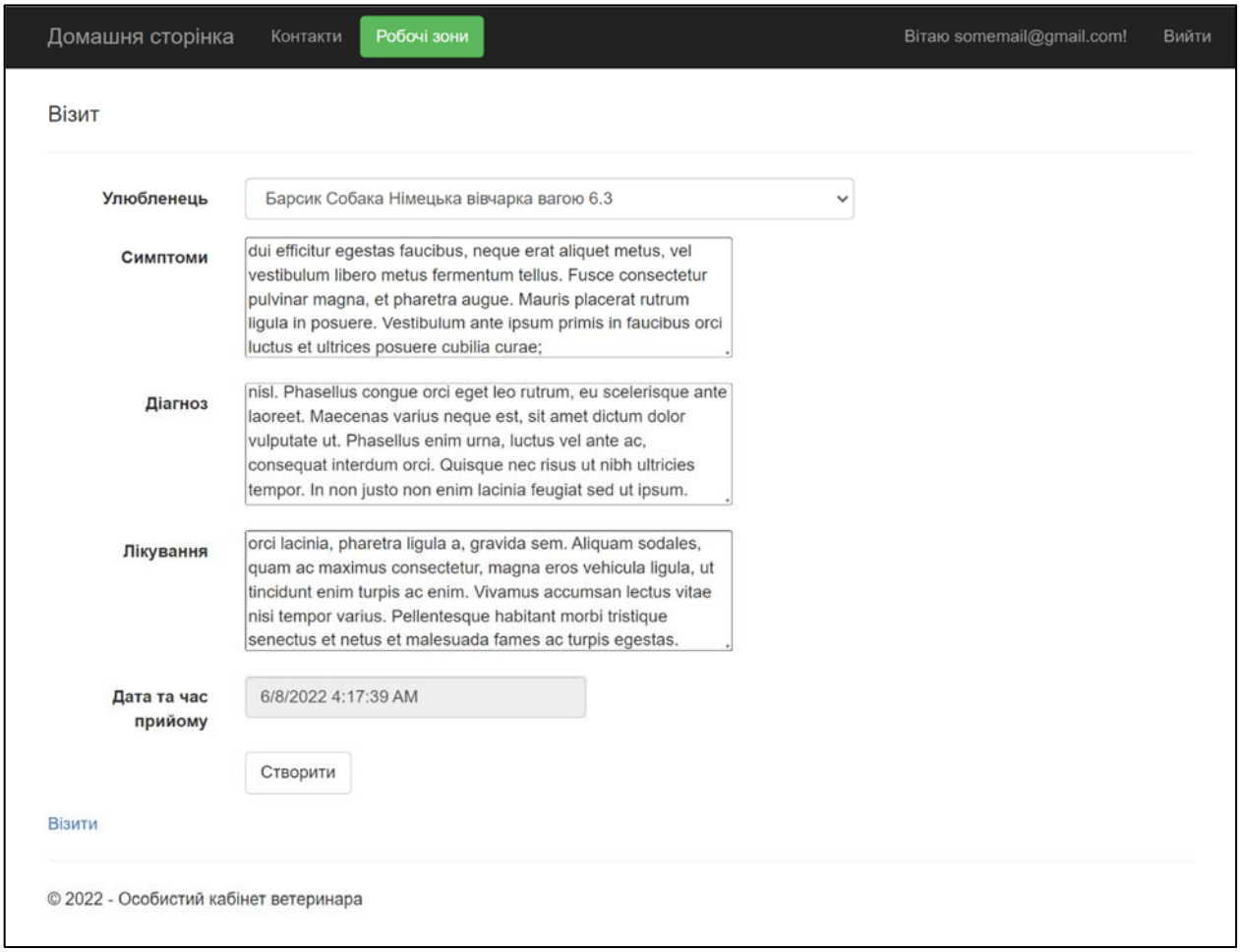

Рисунок 8.19 – Сторінка створення візиту

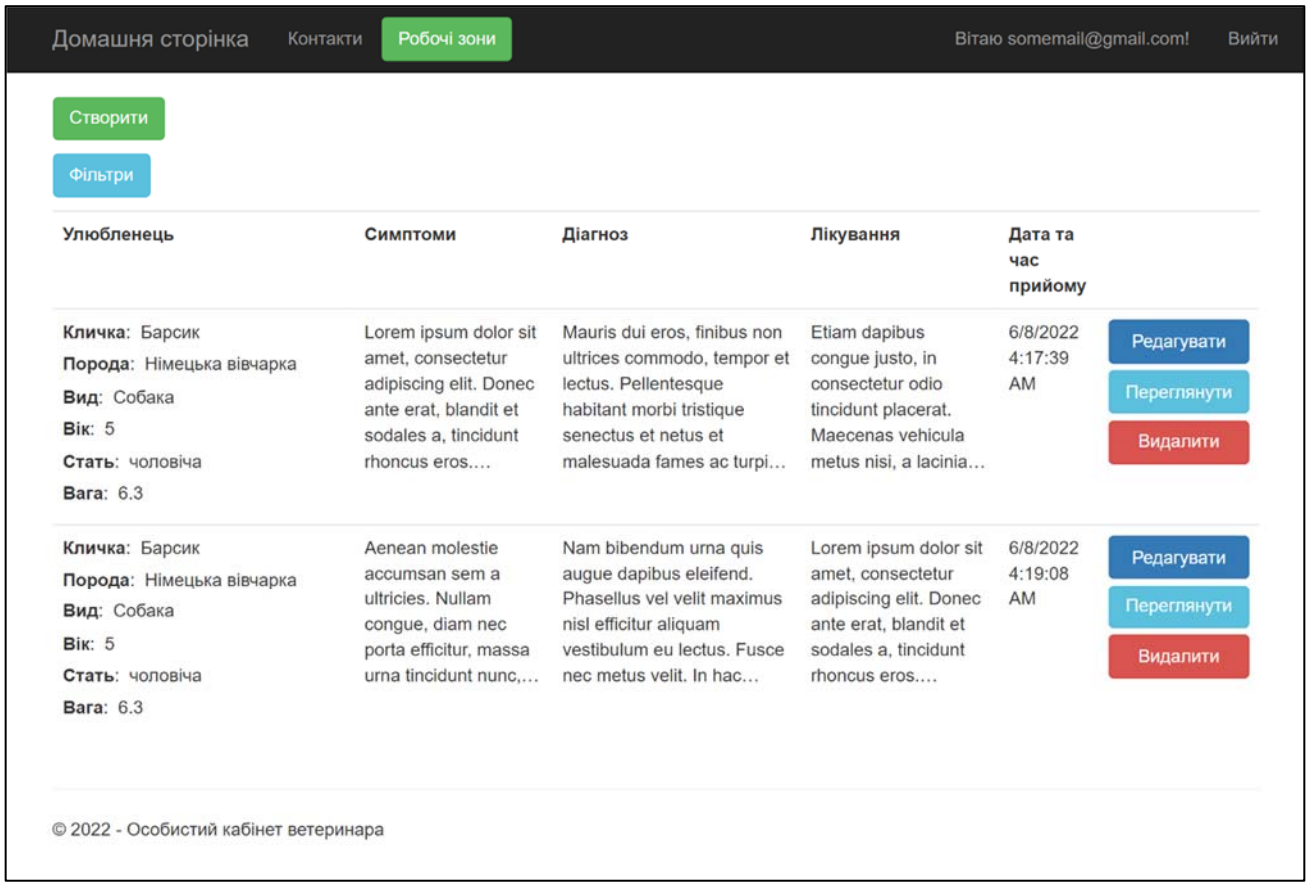

Рисунок 8.20 – Сторінка візитів

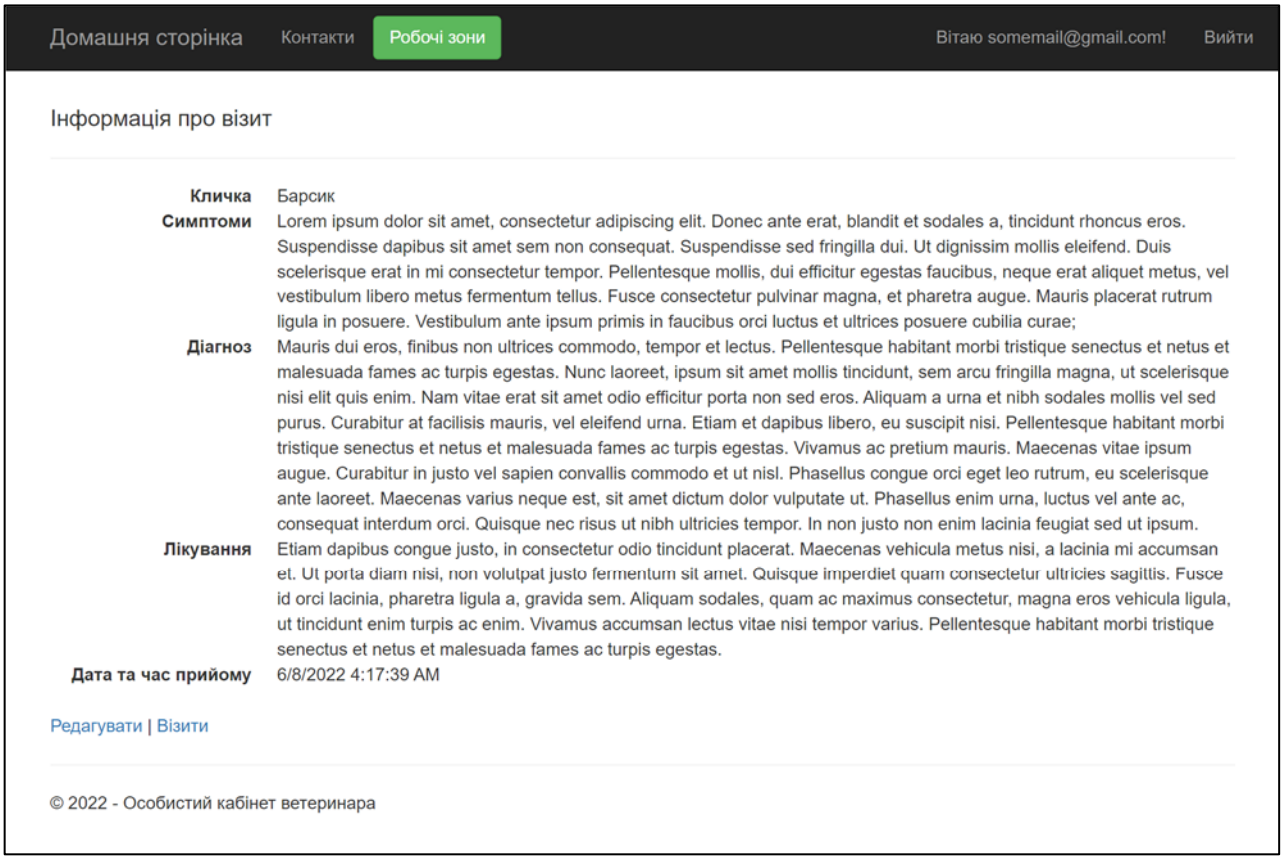

Рисунок 8.21 – Сторінка перегляду візиту

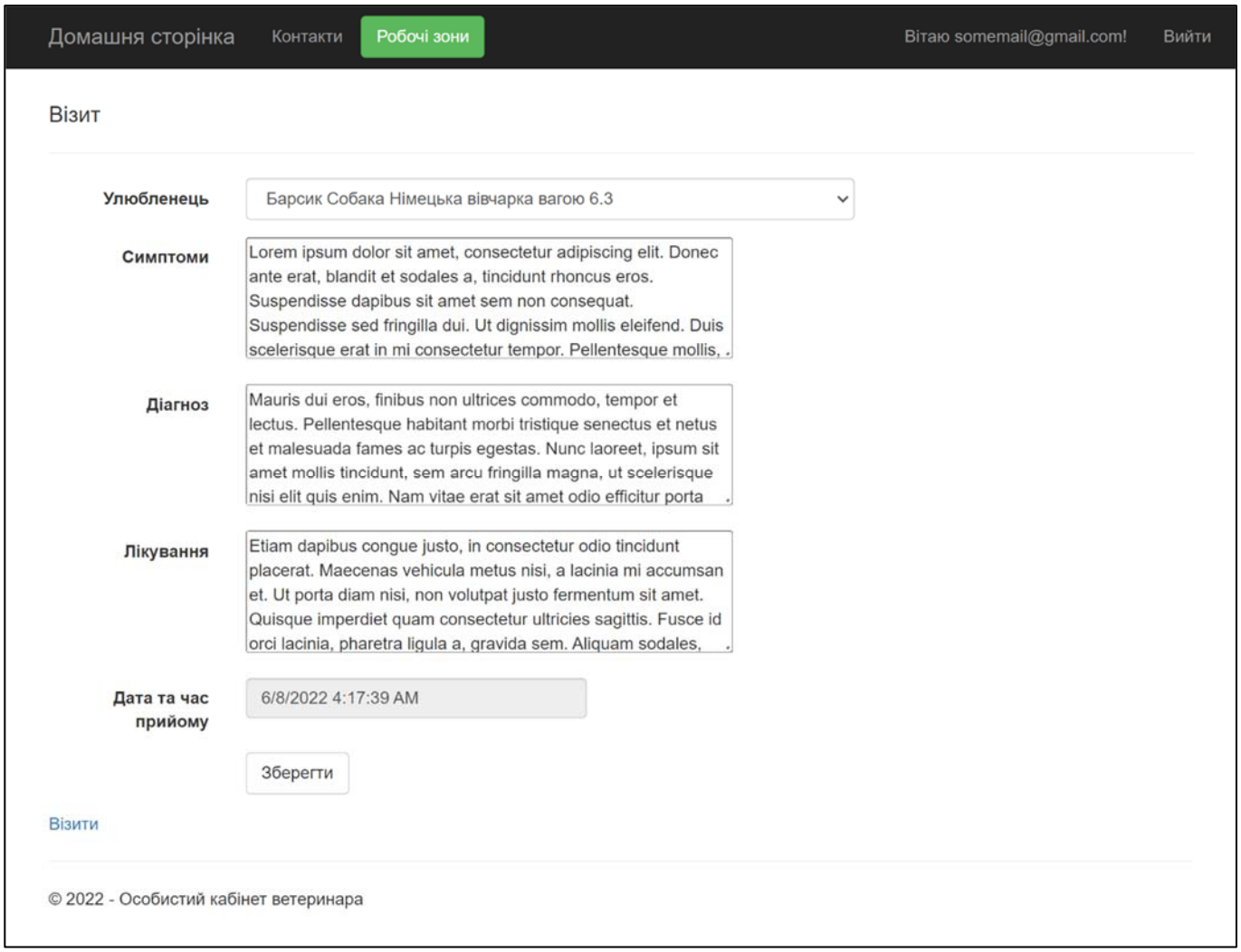

Рисунок 8.22 – Сторінка редагування візиту

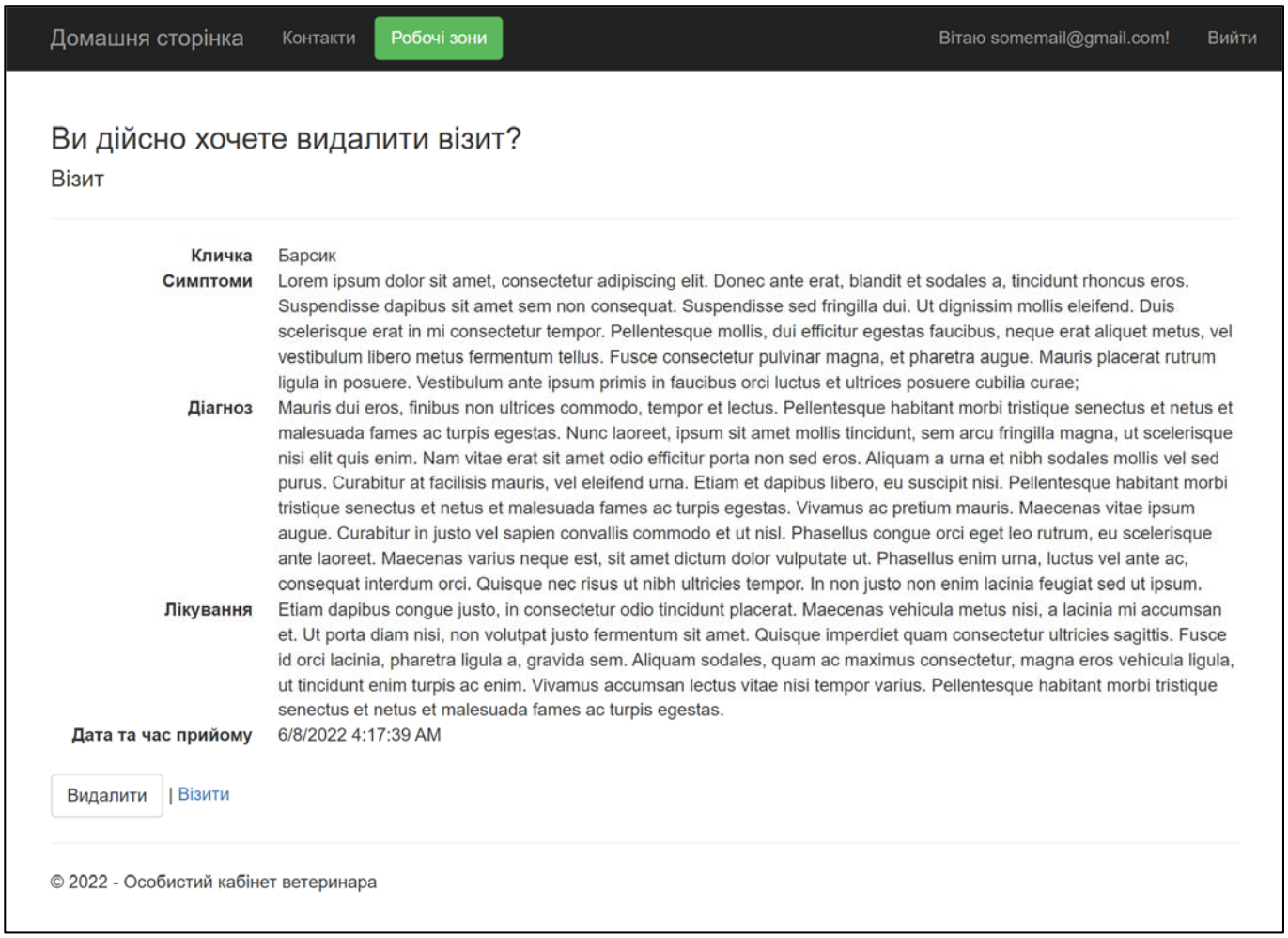

Рисунок 8.23 – Сторінка видалення візиту

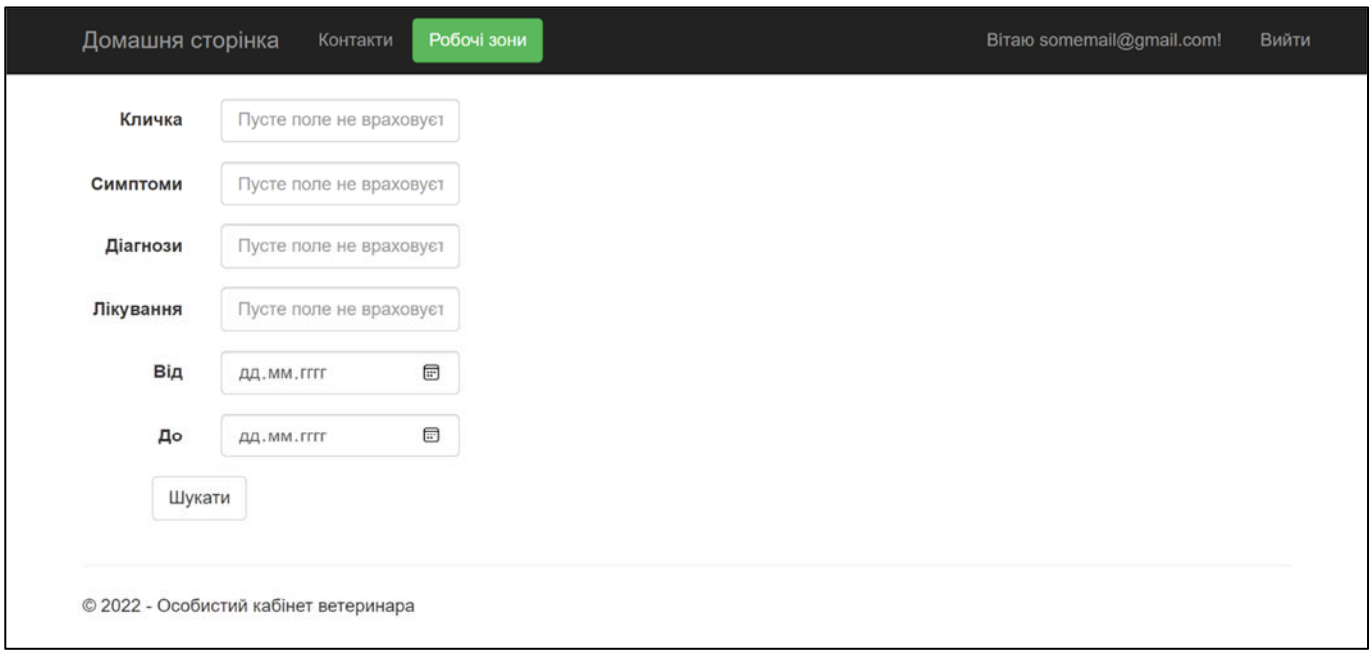

Рисунок 8.24 – Сторінка фільтрації медичних карт

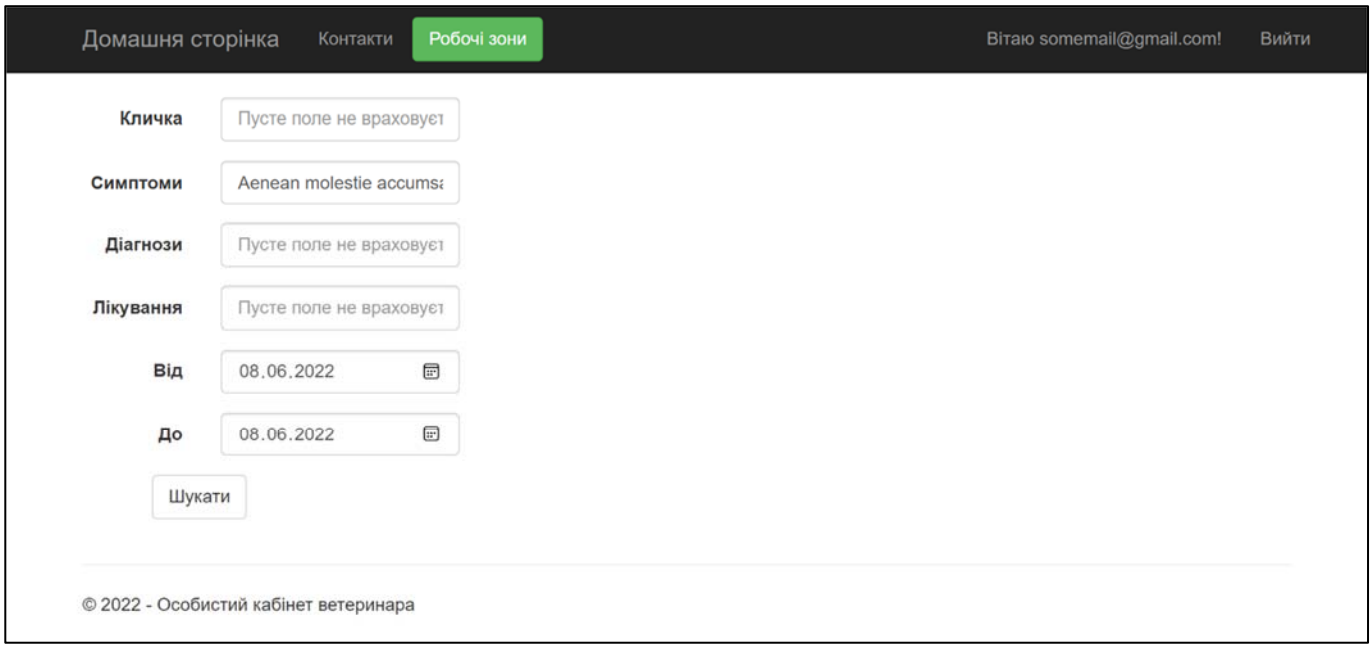

Рисунок 8.25 – Сторінка фільтрації медичних карт. Приклад введених даних

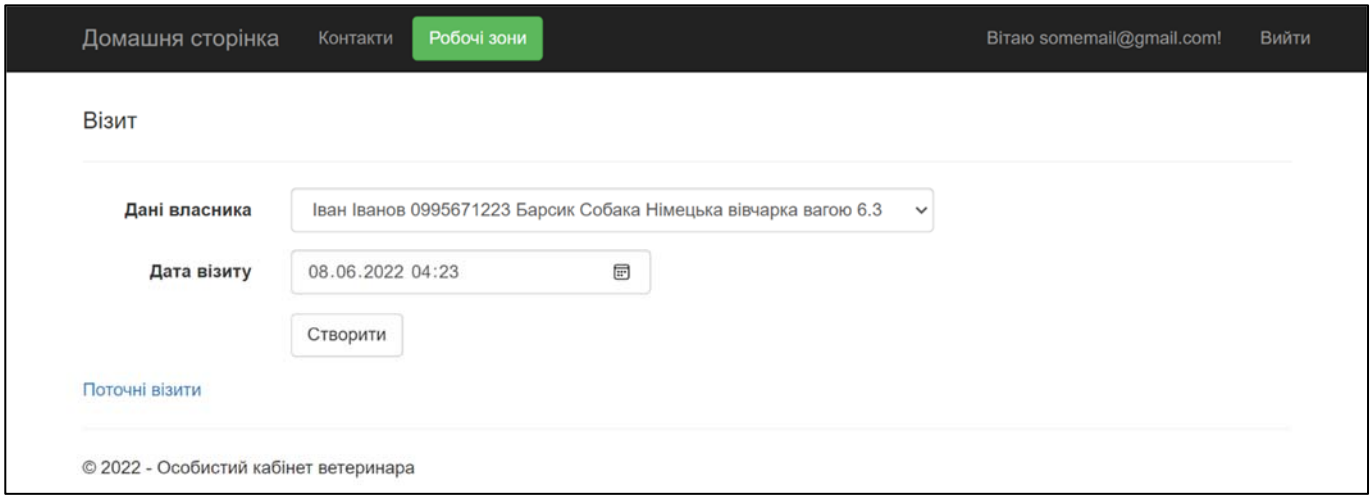

Рисунок 8.26 – Сторінка планування візиту

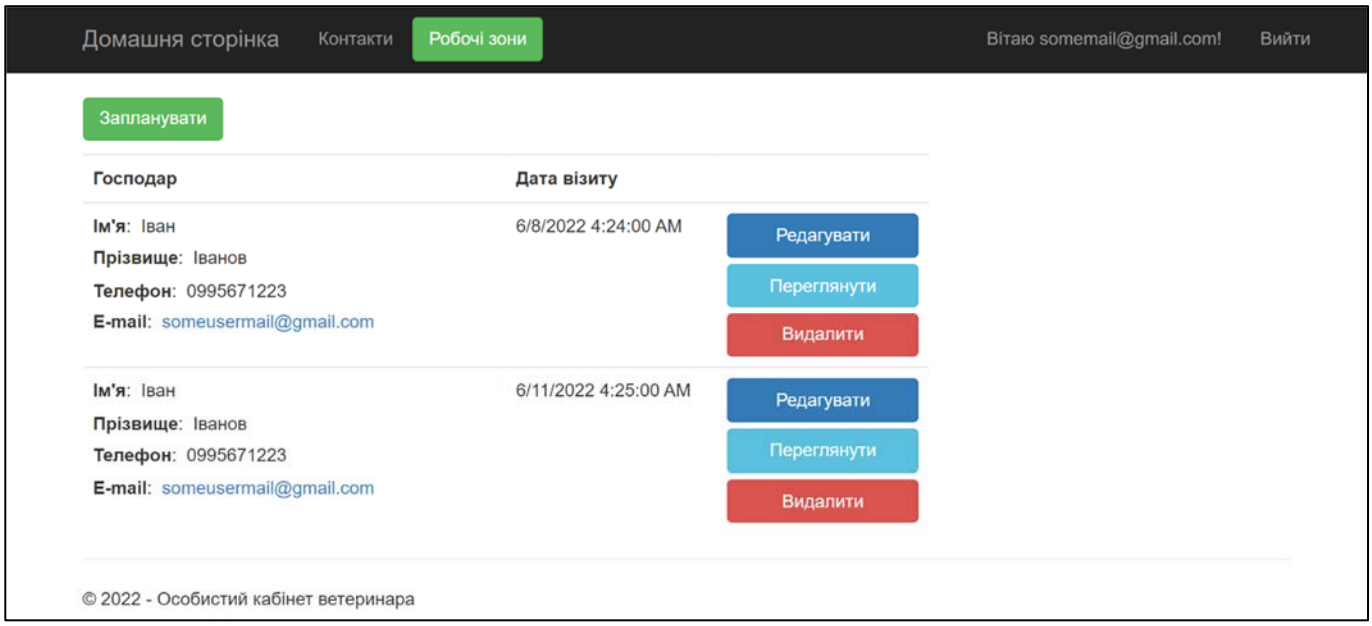

Рисунок 8.27 – Сторінка візитів

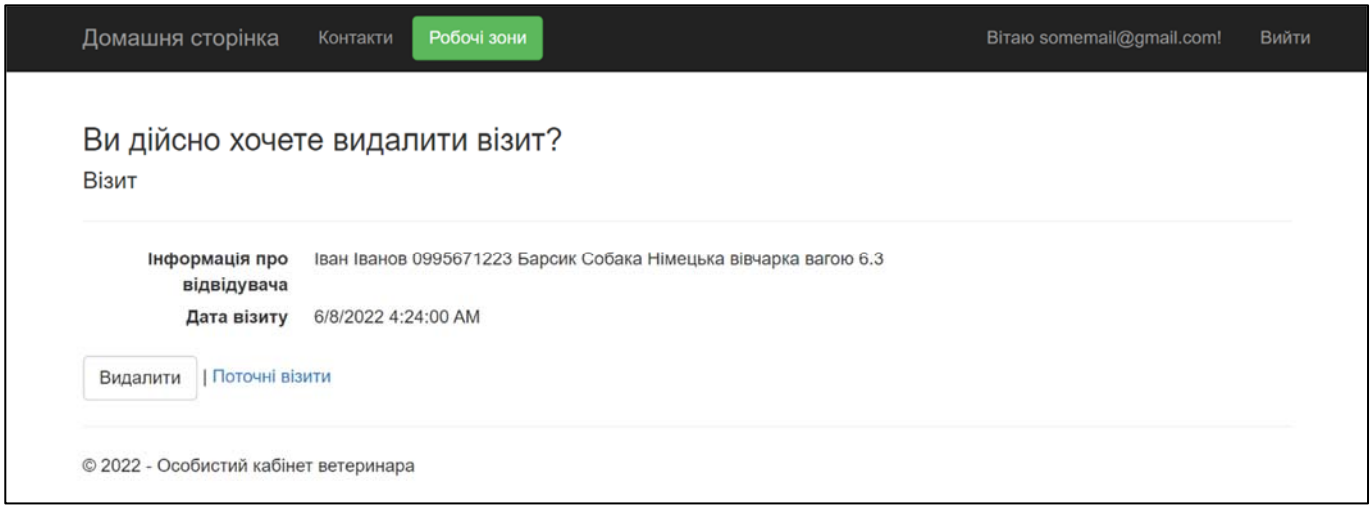

Рисунок 8.29 – Сторінка видалення візиту

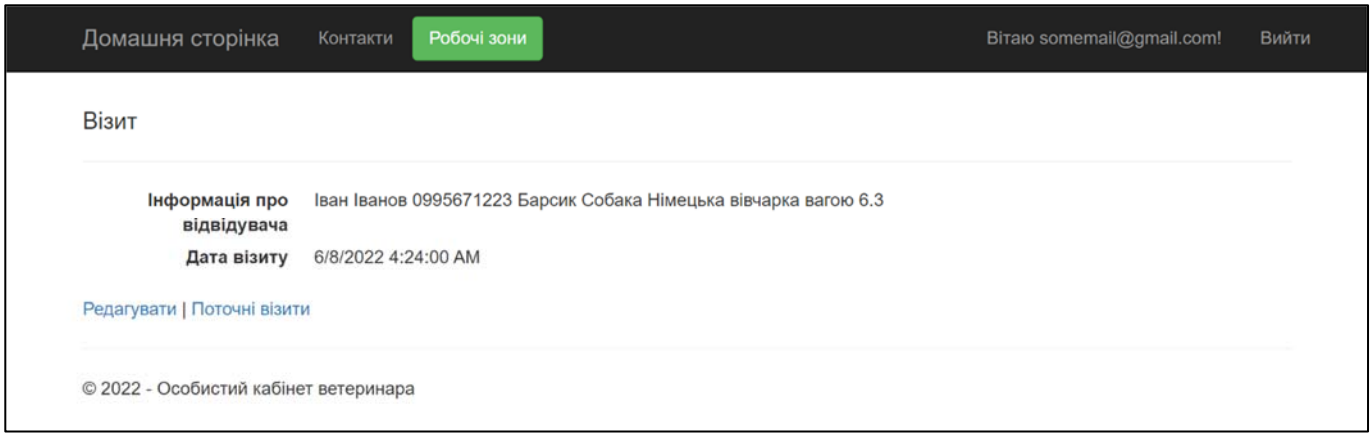

Рисунок 8.30 – Сторінка перегляду інформації про візит

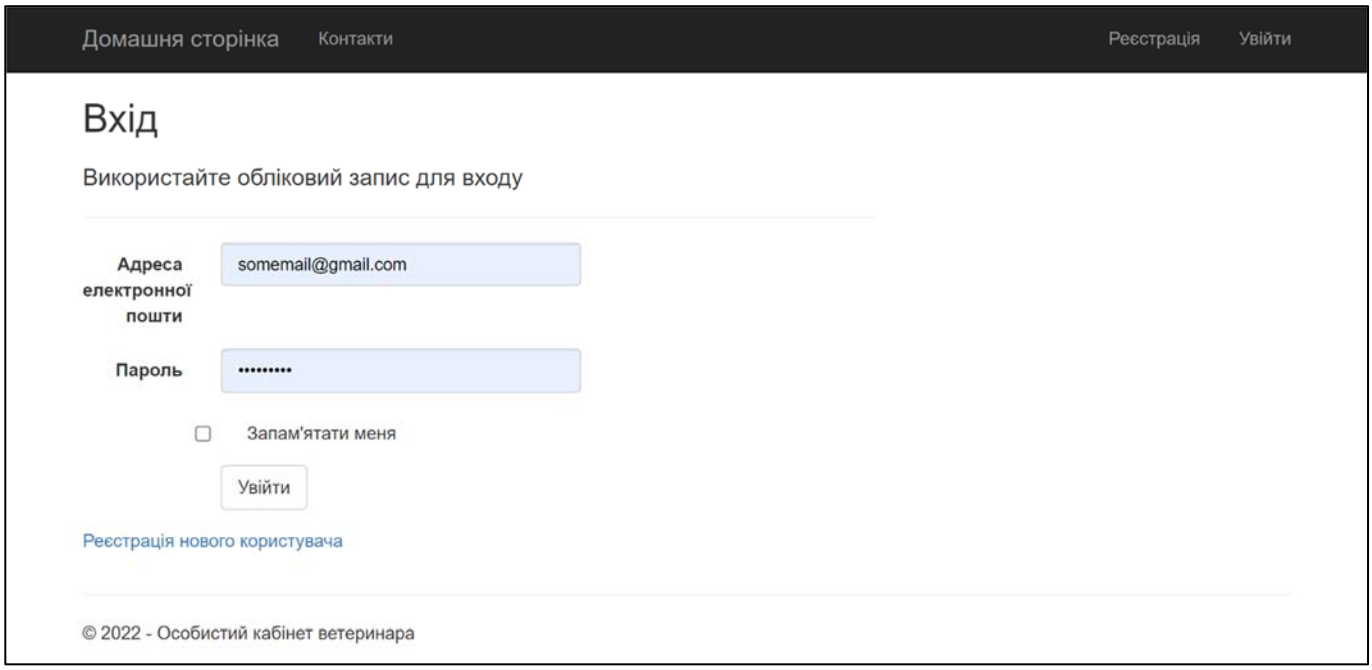

Рисунок 8.31 – Сторінка авторизації після виходу з запису

9 ПОРЯДОК РОБОТИ З ПРОГРАМОЮ

Щодо питань, пов’язаних з продуктом, відправте лист за електронною адресою: nekitkroi@gmail.com. Підтримка з 9:00 до 18:00 по буднях.

10 ПОВІДОМЛЕННЯ

В табл. 10.1 представлені повідомлення системи, що можуть виникнути при роботі з нею.

Таблиця 10.1 – Повідомлення системи

Текст	Адресат	Ситуація	Рекомендовані дії
Невдала спроба увійти	Неавторизований користувач	Користувач ввів неправильні дані для входу до профілю	Перевірити правильність введених даних та спробувати увійти ще раз
Паролі не співпадають	Неавторизований / Авторизований користувач	Користувач ввів різні паролі у поля «пароль» та «підтвердити пароль»	Перевірити правильність паролів, ввести однакові дані в обох полях та спробувати ще раз
Поле Пароль є обов'язковим	Неавторизований користувач	Користувач не заповнив поле паролю	Заповнити поле паролю
Поле Адреса електронної пошти не є дійсною адресою електронної пошти	Неавторизований користувач	Користувач неправильно заповнив поле електронної пошти	Введена пошта не відповідає формату написання електронної пошти
Поле Адреса електронної пошти є обов'язковим	Неавторизований користувач	Користувач не заповнив поле електронної пошти	Заповнити поле
Значення Пароль має містити не менше 6 символів	Неавторизований / Авторизований користувач	Користувач неправильно заповнив поле паролю	Ввести новий пароль
Ім'я вже зайняте. Електронна пошта вже зайнята	Неавторизований користувач	Введена пошта вже зареєстрована в системі	Ввести нове ім'я
Паролі повинні містити принаймні один символ, який не є літерою або цифрою	Неавторизований / Авторизований користувач	Користувач неправильно заповнив поле паролю	Ввести новий пароль
Паролі повинні мати принаймні один символ верхнього регістру	Неавторизований / Авторизований користувач	Користувач неправильно заповнив поле паролю	Ввести новий пароль