Полтавський університет економіки і торгівлі
Навчально-науковий інститут денної освіти
Форма навчання денна
Кафедра комп’ютерних наук та інформаційних технологій

Допускається до захисту
Завідувач кафедри
_____ Олена ОЛЬХОВСЬКА
(підпис)

«___» _____ 2026 р.

КВАЛІФІКАЦІЙНА РОБОТА

на тему
«РОЗРОБКА САЙТУ ДЛЯ ВЕТЕРИНАРНОЇ КЛІНІКИ»

зі спеціальності 122 Комп’ютерні науки
освітня програма «Комп’ютерні науки»
ступеня бакалавра

Виконавець роботи Кришталь Денис Михайлович
_____ «____»_____ 2026 р.
(підпис)

Науковий керівник к.пед.н., доцент, Кошова Оксана Петрівна
_____ «____»_____ 2026 р.
(підпис)

Рецензент

ПОЛТАВА 2026 р.

ЗАТВЕРДЖУЮ
Завідувач кафедри _____Олена ОЛЬХОВСЬКА
(підпис)
«___» _____ 2025 р.

ЗАВДАННЯ І КАЛЕНДАРНИЙ ГРАФІК
ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ

на тему «Розробка сайту для ветеринарної клініки»
зі спеціальності 122 Комп'ютерні науки
освітня програма «Комп'ютерні науки»
ступеня бакалавр
Прізвище, ім'я, по батькові Криштал Денис Михайлович
Затверджена наказом ректора № 213-Н від «01» жовтня 2025 р.
Термін подання студентом роботи «___»_____ 2026 р.
Вихідні дані до кваліфікаційної роботи: статті та документації з теми розробки веб застосунків та сайтів.
Зміст пояснювальної записки (перелік питань, які потрібно розробити)

ВСТУП
РОЗДІЛ 1. ПОСТАНОВКА ЗАДАЧІ
РОЗДІЛ 2. ОГЛЯД СУЧАСНИХ РІШЕНЬ
2.1. Аналіз існуючих вебресурсів аналогічного призначення
2.2. Порівняльна характеристика сучасних вебзастосунків у даній сфері
2.3. Переваги та недоліки існуючих програмних рішень
2.4. Сучасні підходи до розробки вебзастосунків
РОЗДІЛ 3. ТЕОРЕТИЧНА ЧАСТИНА
3.1. Характеристика використаних технологій і засобів розробки
3.2. Проєктування структури та основних модулів вебзастосунку
3.3. Опис архітектури системи та взаємодії її компонентів
РОЗДІЛ 4. ПРАКТИЧНА ЧАСТИНА
4.1. Розробка структури бази даних та основних сутностей системи
4.2. Реалізація серверної частини вебзастосунку
4.3. Розробка користувацького інтерфейсу та основних функціональних можливостей
ВИСНОВКИ
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ
ДОДАТОК А

Перелік графічного матеріалу: 1 аркуш блок-схем, 22 ілюстрації.

Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали, посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Постановка задачі	Кошова О.П.		
Інформаційний огляд	Кошова О.П.		
Теоретична частина	Кошова О.П.		
Практична частина	Кошова О.П.		

Календарний графік виконання кваліфікаційної роботи

Зміст роботи	Термін виконання	Фактичне виконання
Вступ		
Вивчення методичних рекомендацій та стандартів та звіт керівнику		
Постановка задачі		
Інформаційний огляд джерел бібліотек та інтернету		
Теоретична частина		
Практична частина		
Закінчення оформлення		
Доповідь студента на кафедрі		
Доробка (за необхідністю), рецензування		

Дата видачі завдання «__» _____ 2025 р.

Здобувач вищої освіти _____ Кришталь Денис Михайлович
(підпис)

Науковий керівник _____ к.пед.н., доц. Кошова О.П.
(підпис) (науковий ступінь, вчене звання, ініціали та прізвище)

Результати захисту кваліфікаційної роботи

Кваліфікаційна робота оцінена на _____
(балів, оцінка за національною шкалою, оцінка за ECTS)

Протокол засідання ЕК № ____ від «____» _____ 2026 р.

Секретар ЕК_____ _____
(підпис) (ініціали та прізвище)

Затверджую
Зав. кафедрою _____
к.ф.-м.н. Олена ОЛЬХОВСЬКА
«____» _____ 2025 р.

Погоджено
Науковий керівник _____
к.пед.н., Оксана КОШОВА
«____» _____ 2025 р.

План

дипломного проекту з фаху
спеціальності 122 Комп'ютерні науки
освітня програма 122 Комп'ютерні науки
на тему «Розробка сайту ветеринарної клініки»
Прізвище, ім'я, по батькові Кришталь Денис Михайлович

ВСТУП
РОЗДІЛ 1. ПОСТАНОВКА ЗАДАЧІ
РОЗДІЛ 2. ОГЛЯД СУЧАСНИХ РІШЕНЬ
2.1. Аналіз існуючих вебресурсів аналогічного призначення
2.2. Порівняльна характеристика сучасних вебзастосунків у даній сфері
2.3. Переваги та недоліки існуючих програмних рішень
2.4. Сучасні підходи до розробки вебзастосунків
РОЗДІЛ 3. ТЕОРЕТИЧНА ЧАСТИНА
3.1. Характеристика використаних технологій і засобів розробки
3.2. Проєктування структури та основних модулів вебзастосунку
3.3. Опис архітектури системи та взаємодії її компонентів
РОЗДІЛ 4. ПРАКТИЧНА ЧАСТИНА
4.1. Розробка структури бази даних та основних сутностей системи
4.2. Реалізація серверної частини вебзастосунку
4.3. Розробка користувацького інтерфейсу та основних функціональних можливостей
ВИСНОВКИ
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ
ДОДАТОК А

Здобувач вищої освіти _____ Д.М. Кришталь
(підпис)
«____» _____ 2025 р.

РЕФЕРАТ

Записка: 130 сторінок, 22 рисунки, 1 додаток, 21 літературне джерело.

ЕЛЕКТРОННИЙ ЗАПИС НА ПРИЙОМ, СИСТЕМА КЕРУВАННЯ КОНТЕНТОМ, КЛІЄНТСЬКА ЧАСТИНА, СЕРВЕРНА ЧАСТИНА, ВЕБТЕХНОЛОГІЇ.

Мета роботи є розробка вебзастосунку для ветеринарної клініки.

Об’єкт дослідження – процес автоматизації роботи ветеринарної клініки.

Предмет дослідження – методи, засоби та програмні технології розробки вебзастосунку для ветеринарної клініки.

Методи дослідження – у процесі виконання роботи були застосовані методи аналізу предметної області, порівняльного аналізу існуючих вебресурсів аналогічного призначення, методи об’єктно-орієнтованого проєктування, засоби побудови вебзастосунків на основі архітектури MVC, методи проєктування реляційних баз даних, а також сучасні підходи до реалізації серверної та клієнтської частин програмного забезпечення. Для створення вебзастосунку використано ASP.NET Core MVC, Entity Framework Core, LINQ, Razor, HTML5, CSS3, JavaScript, Bootstrap і Microsoft SQL Server. Організацію доступу до даних реалізовано на основі Repository Pattern. Для перевірки коректності введених даних застосовано DataAnnotations, для роботи із завантаженням і збереженням зображень використано FileService, а для обробки станів записів на прийом використано enum.

У межах дослідження було визначено основні вимоги до функціоналу вебзастосунку для ветеринарної клініки, зокрема вимоги до інформаційного наповнення, зручності користувацького інтерфейсу, організації запису на прийом, адміністрування даних і підтримки актуальності відомостей про працівників та послуги. Проведено аналіз існуючих вебресурсів аналогічного призначення, визначено їх переваги та недоліки, що дозволило обґрунтувати

вибір підходів до реалізації власного програмного рішення. Описано й обґрунтовано вибір архітектури MVC, структури серверної частини, бази даних, а також засобів реалізації інтерфейсу користувача.

У процесі виконання дипломної роботи спроєктовано структуру вебзастосунку, розроблено основні програмні модулі та реалізовано серверну частину системи. Створено базу даних для збереження інформації про працівників, послуги, записи на прийом, публікації, категорії, коментарі, теги, навігаційні елементи, параметри сайту та користувачів. Розроблено механізми доступу до даних, які забезпечують створення, отримання, редагування та видалення основних сутностей системи. Реалізовано публічну частину вебзастосунку, що забезпечує перегляд інформації про клініку, працівників, послуги, публікації та інші розділи сайту.

Окрему увагу приділено реалізації модуля запису на прийом, який забезпечує введення, перевірку, збереження та подальше адміністрування звернень користувачів. У системі передбачено обробку статусів записів, що дозволяє контролювати їх поточний стан у межах адміністративної частини. Розроблено адміністративний інтерфейс, який забезпечує керування працівниками, послугами, категоріями, публікаціями, коментарями, тегами, параметрами та іншими об’єктами системи. Для представлення архітектури та логіки роботи програмного рішення підготовлено діаграми класів, структури серверної частини, блок-схеми основних модулів і схеми бази даних.

У результаті виконання дипломної роботи створено функціональний вебзастосунок для ветеринарної клініки, який поєднує інформаційні, сервісні та адміністративні можливості в межах єдиної системи. Розроблене програмне рішення забезпечує зручну взаємодію користувачів із клінікою, підтримує централізоване керування даними та може бути використане як основа для подальшого розвитку і розширення функціональних можливостей.

ЗМІСТ

ВСТУП

Широке впровадження інформаційних систем у різні сфери діяльності зумовлює необхідність створення зручних, функціональних і надійних програмних рішень для автоматизації основних робочих процесів. Однією з актуальних сфер застосування таких рішень є діяльність ветеринарних клінік, у межах якої важливе значення мають оперативне опрацювання інформації, зручна взаємодія з клієнтами, впорядковане ведення відомостей про персонал, послуги та записи на прийом. Використання вебзастосунків у цій сфері дає змогу централізувати дані, спростити керування інформаційним наповненням сайту, покращити організацію роботи установи та підвищити якість обслуговування відвідувачів.

Актуальність теми зумовлена потребою в автоматизації основних процесів роботи ветеринарної клініки, зокрема публікації та оновлення контенту, ведення відомостей про працівників, надані послуги та організації запису на прийом. Традиційні способи ведення такої інформації часто є недостатньо зручними, вимагають значних витрат часу та не забезпечують належного рівня впорядкованості даних. У зв’язку з цим розробка вебзастосунку для ветеринарної клініки є актуальним завданням, спрямованим на підвищення ефективності інформаційного супроводу та спрощення взаємодії між адміністрацією закладу і клієнтами.

Метою роботи є розробка вебзастосунку для ветеринарної клініки.

Об’єктом дослідження є процес автоматизації роботи ветеринарної клініки.

Предметом дослідження є методи, засоби та програмні технології розробки вебзастосунку для ветеринарної клініки.

Для досягнення поставленої мети необхідно вирішити такі завдання: провести аналіз існуючих вебресурсів і програмних рішень аналогічного

призначення; визначити основні функціональні вимоги до вебзастосунку; обґрунтувати вибір технологій і засобів розробки; спроєктувати структуру системи; реалізувати серверну частину застосунку; організувати роботу з базою даних; реалізувати механізми доступу до даних; розробити користувацький інтерфейс; забезпечити валідацію введених даних; реалізувати допоміжні механізми роботи із файлами та зображеннями; передбачити обробку статусів записів на прийом; провести перевірку працездатності системи та відповідності її функціональних можливостей поставленим вимогам.

Методи дослідження включають аналіз предметної області, методи об’єктно-орієнтованого проєктування, принципи побудови інформаційних систем для автоматизації діяльності організації, а також підходи до розробки вебзастосунків із розподілом функцій між серверною і клієнтською частинами. Основною мовою програмування, використаною під час розробки вебзастосунку, є C#, на основі якої реалізовано серверну частину системи з використанням фреймворку ASP.NET Core, який забезпечив реалізацію логіки обробки запитів, маршрутизації, взаємодії між контролерами та представленнями. Архітектурною основою програмного рішення став підхід MVC, відповідно до якого структура застосунку поділяється на моделі, представлення та контролери, що сприяє логічному розмежуванню відповідальності між компонентами системи.

Для роботи з даними застосовано Entity Framework Core, який забезпечує взаємодію з базою даних на рівні програмних моделей і дає змогу виконувати операції створення, читання, оновлення та видалення записів. Для формування вибірок, фільтрації, сортування та обробки даних використано LINQ, що забезпечує зручний механізм роботи з колекціями та сутностями. Для збереження даних використано Microsoft SQL Server, який виконує роль основного сховища інформації про записи на прийом, працівників, послуги, публікації та інші об’єкти системи.

Клієнтська частина застосунку реалізована із застосуванням Razor, що дає змогу формувати динамічні вебсторінки на сервері та поєднувати HTML-розмітку з даними, отриманими з контролерів і моделей. Для побудови структури сторінок використано HTML5, для стилізації інтерфейсу застосовано CSS3, а для реалізації окремих елементів динамічної поведінки використано JavaScript. Для створення адаптивного інтерфейсу, оформлення таблиць, форм, кнопок, карток та інших елементів застосовано Bootstrap, що забезпечує зручність використання вебзастосунку на різних типах пристроїв.

У структурі програмного забезпечення також використано патерн Repository, який забезпечує відокремлення логіки доступу до даних від інших компонентів системи та спрощує супровід програмного коду. Для перевірки коректності введених користувачем даних застосовано DataAnnotations, що дає змогу реалізувати валідацію полів моделей. Для роботи із завантаженням і збереженням зображень та інших файлів використано FileService. Крім того, у системі передбачено використання enum для подання статусів записів на прийом, що забезпечує впорядкованість логіки їх обробки та однозначність визначення поточного стану кожного запису.

Пояснювальна записка складається зі вступу, чотирьох розділів, висновків, списку використаних джерел і додатків. У першому розділі сформульовано постановку задачі, мету, завдання та основні вимоги до вебзастосунку. У другому розділі проведено аналіз існуючих рішень аналогічного призначення та розглянуто сучасний стан проблеми. У третьому розділі висвітлено теоретичні аспекти проєктування та обґрунтовано вибір засобів розробки. У четвертому розділі подано практичну реалізацію вебзастосунку, описано структуру бази даних, серверну частину, інтерфейс користувача та основні функціональні можливості системи.

.

РОЗДІЛ 1. ПОСТАНОВКА ЗАДАЧІ

Розробка сучасних вебзастосунків для автоматизації діяльності організацій є важливим напрямом розвитку інформаційних технологій, оскільки такі програмні рішення забезпечують впорядковане зберігання даних, спрощують доступ до інформації та підвищують ефективність взаємодії між користувачами і адміністрацією системи. У сфері ветеринарного обслуговування потреба у впровадженні подібних засобів є особливо актуальною, оскільки діяльність клініки пов’язана з необхідністю організації запису на прийом, ведення відомостей про персонал, представлення переліку послуг, розміщення інформаційних матеріалів і забезпечення зручного доступу клієнтів до необхідної інформації. Використання вебзастосунку дає можливість поєднати зазначені функції в межах єдиного програмного середовища, що позитивно впливає на якість обслуговування, зменшує навантаження на працівників і сприяє впорядкуванню внутрішніх процесів.

Основне завдання дипломної роботи полягає у створенні вебзастосунку для ветеринарної клініки, який забезпечує інформаційну підтримку роботи закладу та реалізує основні функції, необхідні для його повсякденної діяльності. Розроблюване програмне рішення повинно поєднувати можливості керування контентом, ведення інформації про працівників і послуги, а також підтримку механізму запису на прийом. Вебзастосунок має бути зручним у використанні як для відвідувачів сайту, так і для адміністратора, який виконує оновлення вмісту, редагування відомостей і контроль даних у системі. Важливою умовою є забезпечення цілісності структури застосунку, логічної організації його компонентів та узгодженої взаємодії між серверною частиною, базою даних і користувацьким інтерфейсом [14-21].

Метою дипломної роботи є розробка вебзастосунку для ветеринарної клініки. Досягнення поставленої мети передбачає розв’язання комплексу взаємопов’язаних завдань, що охоплюють як аналітичні, так і проєктно-

реалізаційні етапи. Передусім необхідно дослідити особливості предметної області та проаналізувати існуючі вебресурси аналогічного призначення для виявлення їх сильних і слабких сторін. Це дає змогу визначити функціональні можливості, які мають бути реалізовані у власному програмному продукті, а також обґрунтувати доцільність створення такого вебзастосунку [14-20].

Наступним завданням є формування функціональних вимог до системи. До них належать забезпечення можливості перегляду інформації про клініку, послуги та персонал, підтримка публікації та редагування інформаційних матеріалів, організація запису на прийом, збереження та обробка відповідних даних, а також надання засобів адміністрування основних об’єктів системи. Окремого значення набувають вимоги до зручності інтерфейсу, логічності навігації, коректності введення даних і адаптивності сторінок для різних типів пристроїв. Функціональні можливості застосунку повинні бути достатніми для підтримки основних інформаційних і організаційних процесів клініки [14-20].

Важливим етапом є обґрунтування загальної структури програмного рішення. У межах роботи необхідно передбачити поділ системи на логічно пов’язані компоненти, що відповідають за обробку запитів, зберігання даних, формування представлень і взаємодію з користувачем. Такий підхід забезпечує чіткий розподіл функцій між частинами системи, підвищує зрозумілість програмної реалізації та створює передумови для подальшого супроводу і розвитку вебзастосунку. Крім того, структура системи має враховувати особливості предметної області, зокрема наявність різних типів даних, пов’язаних із записами на прийом, відомостями про персонал, послугами, публікаціями та іншими інформаційними об’єктами [14-20].

Оскільки основою функціонування вебзастосунку є робота з даними, окрему увагу необхідно приділити проєктуванню інформаційного наповнення системи та принципам організації бази даних. Потрібно визначити основні сутності, їхні атрибути та зв’язки між ними, забезпечити можливість збереження, оновлення, пошуку й відображення інформації у зручному для

користувача вигляді. Водночас система повинна підтримувати коректну обробку статусів записів на прийом, що є важливою складовою її практичного функціонування, оскільки дає змогу відображати поточний стан взаємодії між клієнтом і клінікою [14-20].

Щодо вимог до забезпечення коректності введення та опрацювання інформації, то усі дані, що надходять до системи через форми введення, повинні перевірятися на відповідність визначеним умовам, щоб запобігти помилкам, некоректним записам і порушенню цілісності інформації. Поряд із цим у вебзастосунку необхідно передбачити можливість роботи із файлами та зображеннями, що є важливим для представлення інформаційних матеріалів, відомостей про працівників або інших об’єктів, пов’язаних із діяльністю клініки. Реалізація таких функцій підвищує інформативність системи та покращує її практичну цінність [14-20].

У межах дипломної роботи важливо не лише реалізувати основні функціональні можливості системи, а й забезпечити її зручність, логічність побудови та відповідність сучасним вимогам до вебзастосунків. Це означає, що створюване програмне рішення повинно мати зрозумілий інтерфейс, послідовну структуру сторінок, коректну маршрутизацію та узгоджену взаємодію всіх елементів. Практична цінність вебзастосунку полягає в тому, що він може бути використаний як інструмент інформаційної підтримки діяльності ветеринарної клініки, а також як основа для подальшого розширення функціоналу відповідно до потреб конкретного закладу [14-21].

Отже, у дипломної роботи плануємо розробити цілісний вебзастосунок, орієнтований на автоматизацію основних процесів роботи ветеринарної клініки. Реалізація такого програмного рішення потребує аналізу предметної області, формування функціональних вимог, проєктування структури системи, організації бази даних, забезпечення коректної обробки інформації та створення зручного користувацького інтерфейсу. Усе це формує теоретичну та практичну основу для подальшого розгляду сучасних аналогів, вибору засобів

реалізації та безпосереднього створення програмного продукту в наступних розділах роботи.

РОЗДІЛ 2. ОГЛЯД СУЧАСНОГО СТАНУ ПРОБЛЕМИ

2.1. Аналіз існуючих вебресурсів аналогічного призначення

На етапі проєктування вебзастосунку для ветеринарної клініки важливим є аналіз існуючих вебресурсів аналогічного призначення, оскільки він дає змогу визначити поширені підходи до організації структури сайту, складу функціональних модулів та загальних технічних рішень. Для такого аналізу доцільно розглянути кілька прикладів сайтів, які відрізняються за наповненням, способом взаємодії з користувачем і рівнем цифрової інтеграції. Серед таких прикладів можна виокремити сайт ветеринарного центру 103Vet, сайт клініки «ВетНова», сервіс онлайн-консультацій Animal Clinic Live Expert, сторінку для пацієнтів English Creek Veterinary Clinic, а також сайти VetDomik, VetLik та VetPixel, які є показовими прикладами локальних вебресурсів ветеринарних клінік у місті Полтава. Зазначимо що ми проводили лише аналіз веб сайтів а не сторінок у соцмережах та інших видах організованого спілкування із клієнтами.

Сайт VetDomik (мережа ветклінік, розташованих у Полтаві, Харкові і Києві) містить структуроване представлення основної інформації про клініку, зокрема відомості про місце розташування в Полтаві, режим роботи, контакти, перелік послуг, ціни, відповіді на поширені запитання та відгуки. Окрему увагу привертає наявність функції запису на прийом, яка реалізована через перехід до зовнішнього сервісу, а також поділ сайту за мовними версіями і містами. Це свідчить про те, що ресурс поєднує інформаційну складову з сервісною та має ознаки масштабованої структури, орієнтованої на підтримку кількох локацій і розширення інформаційного наповнення. У технічному аспекті такий підхід є показовим, оскільки демонструє організацію вебресурсу як єдиного інформаційного простору з окремими функціональними блоками та винесенням частини інтерактивної логіки до спеціалізованої зовнішньої платформи запису.

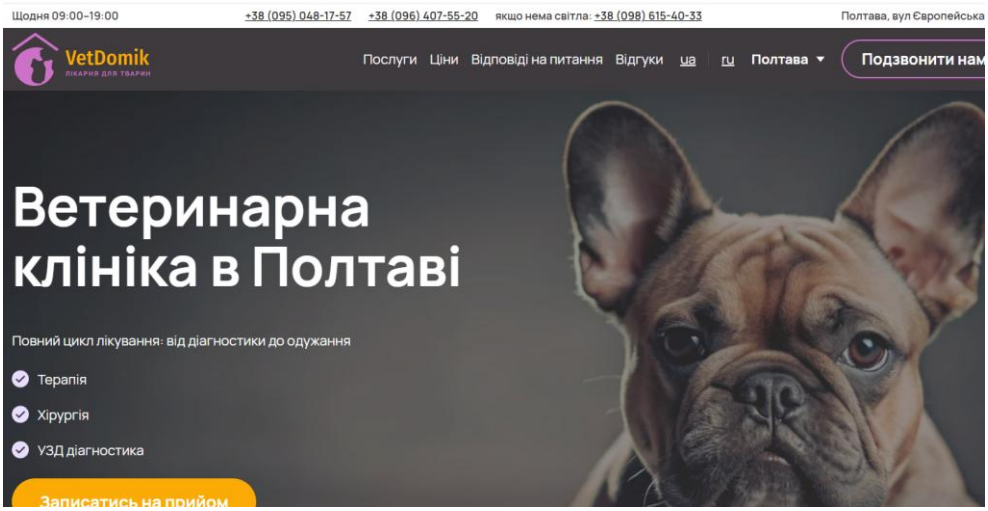


Рисунок 2.1. – Сайт клініки VetDomik

Порівняно з цим, сайт 103Vet також орієнтований на поєднання представницької та сервісної функцій, але його структура є більш компактною. На ньому акцент зроблено на швидкому доступі до основних послуг, інформації про спеціалістів, контактних даних та можливості онлайн-запису. Такий підхід характерний для вебресурсів, у яких головним є оперативне спрямування користувача до ключових дій без надмірного розгалуження структури. З технічної точки зору це свідчить про прагнення до мінімізації переходів між сторінками та концентрації основних функцій у межах декількох логічно пов’язаних розділів.

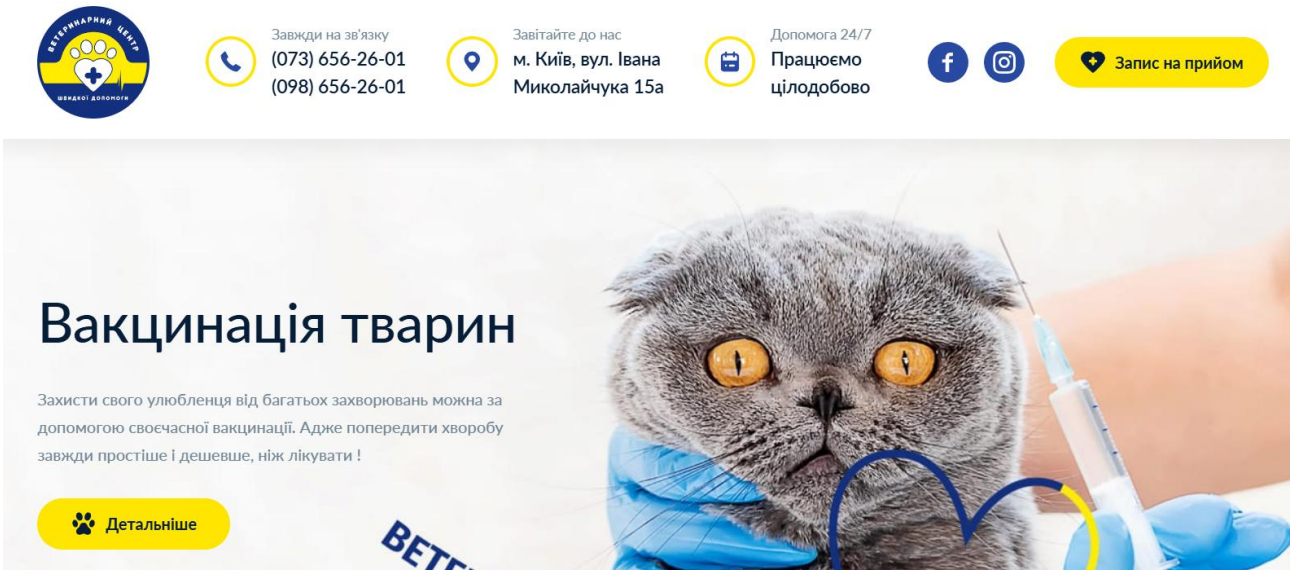


Рисунок 2.2. – Сайт клініки 103Vet

Сайт «ВетНова» демонструє інший підхід до побудови вебресурсу. Окрім стандартних розділів, пов’язаних із послугами, персоналом і контактами, на ньому передбачено тематичні сторінки з корисною інформацією, правилами прийому, кар’єрними можливостями та багатомовним представленням контенту. Така організація вказує на більш складну інформаційну архітектуру, у межах якої сайт виконує не лише функцію представлення клініки, а й функцію систематизованого інформаційного середовища. У загальному технічному розумінні це означає необхідність кращого структурування контенту, узгодженості між розділами та підтримки більшої кількості типів сторінок у межах єдиного ресурсу.

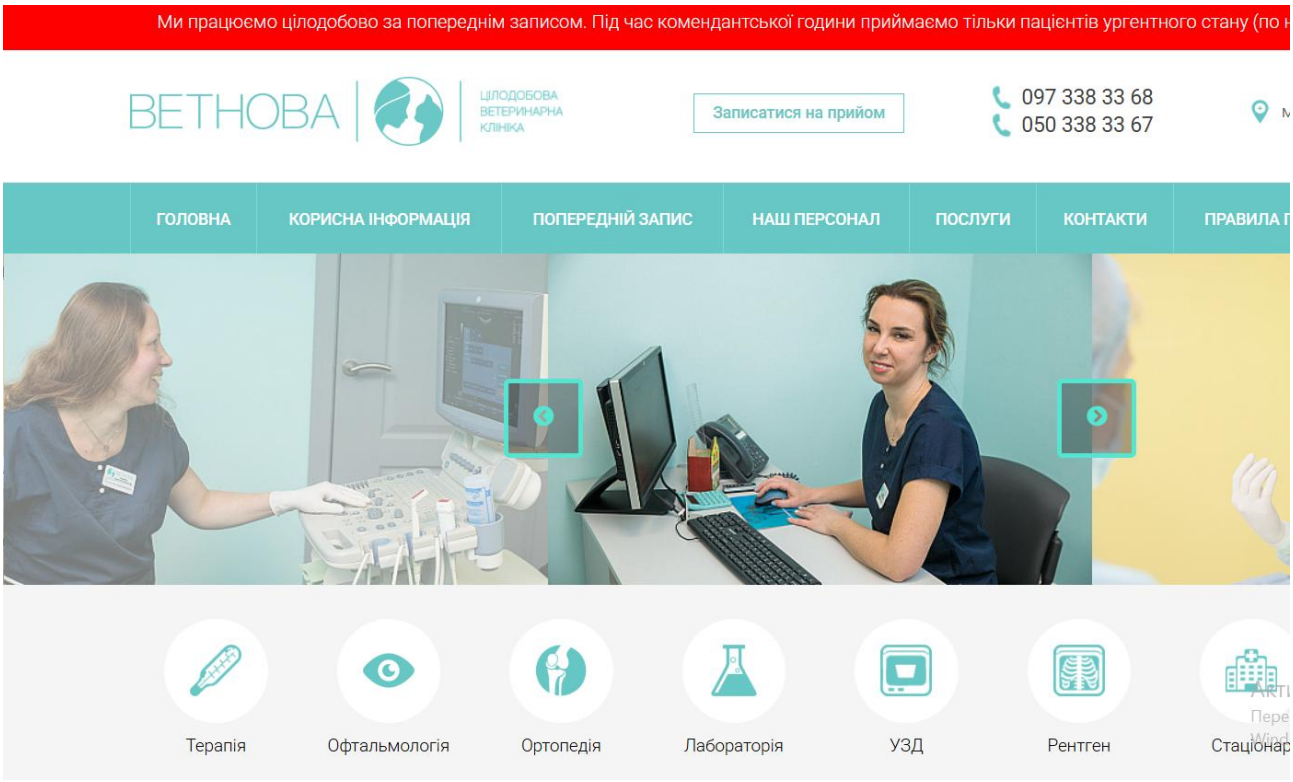


Рисунок 2.3. – Сайт клініки ВетНова

Сервіс Animal Clinic Live Expert відображає модель вебресурсу, у якій основний акцент зміщується у бік дистанційної взаємодії з користувачем. Консультації організовуються через зовнішні канали комунікації, зокрема месенджери та сервіси відеозв’язку, а сам сайт виступає як точка координації

звернення. Такий формат свідчить про інший тип технічної організації, коли вебресурс не повністю реалізує весь сервісний функціонал усередині власної структури, а інтегрується із зовнішніми засобами зв'язку. Це зменшує складність власної внутрішньої реалізації, але водночас підвищує вимоги до узгодженості користувацького сценарію та логіки перенаправлення між платформами.

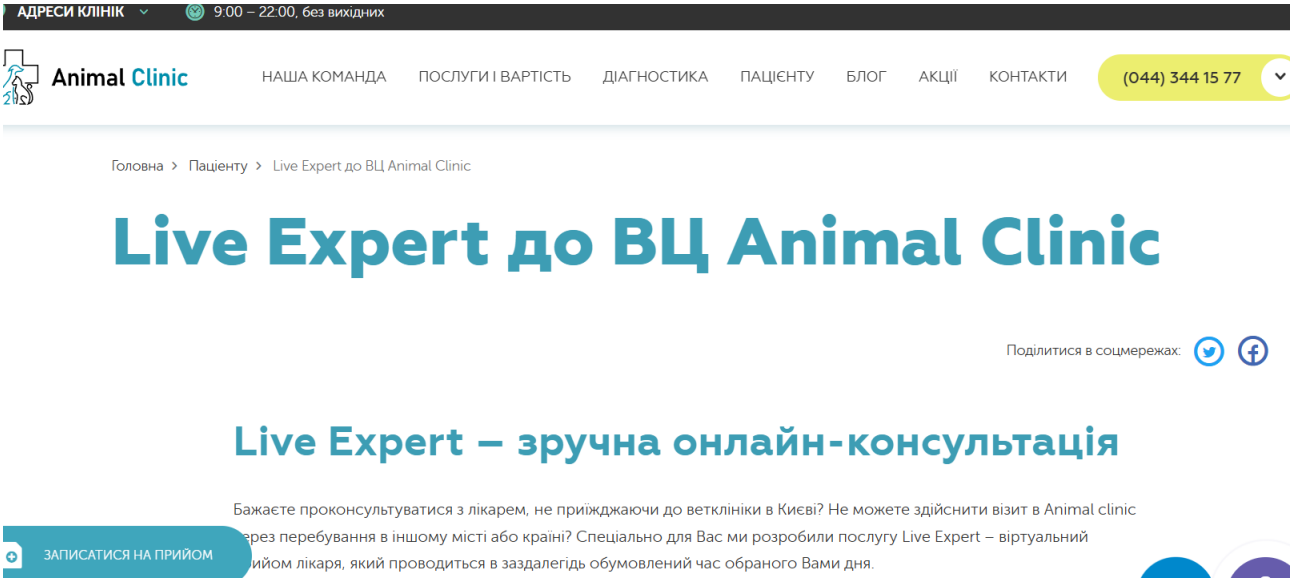


Рисунок 2.4. – Сайт Animal Clinic Live Expert

English Creek Veterinary Clinic демонструє більш високий рівень цифрової інтеграції, оскільки поряд із публічними сторінками для користувачів передбачено доступ до персоналізованого порталу, через який можна переглядати записи, медичну інформацію та інші дані, пов'язані з твариною. Такий підхід свідчить про перехід від традиційного інформаційного сайту до моделі повноцінної інформаційної системи з персоналізованим доступом. У технічному аспекті це вказує на складнішу організацію логіки зберігання та обробки даних, необхідність підтримки облікових записів і забезпечення взаємодії між відкритою та закритою частинами вебресурсу.

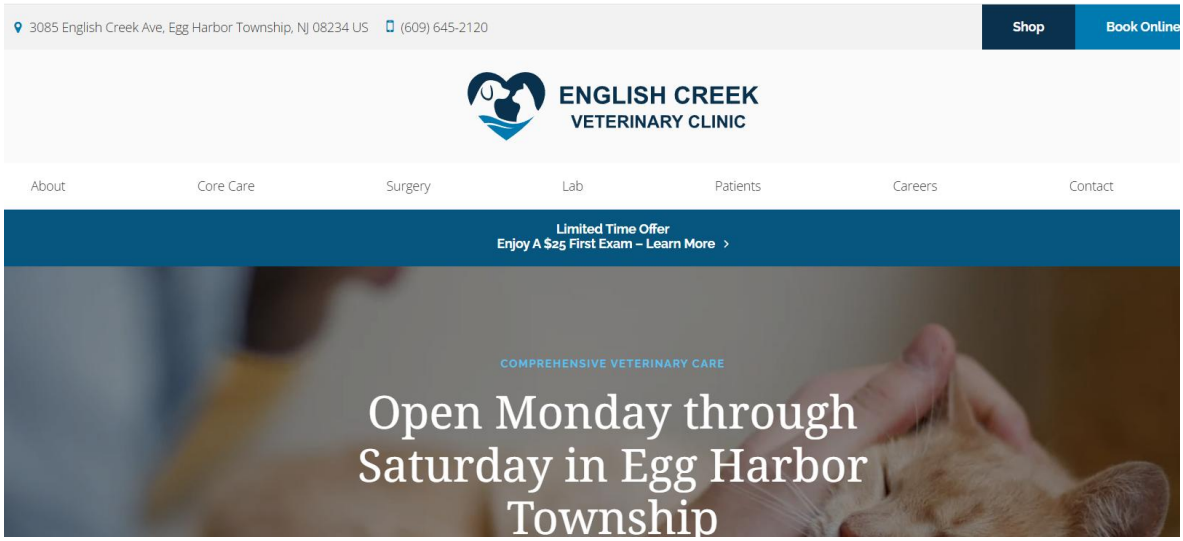


Рисунок 2.5. – Сайт клініки English Creek Veterinary Clinic

Додатково до аналізу доцільно розглянути ще два вебресурси, що функціонують у Полтаві, а саме VetLik та VetPixel. Сайт VetLik є прикладом більш структурованого інформаційно-сервісного ресурсу, на якому представлено окремі сторінки з контактами, персоналом, вакансіями та записом на прийом. Така побудова свідчить про наявність чіткішої організації контенту, поділу інформації за тематичними розділами та орієнтації не лише на зовнішнє представлення клініки, а й на систематизоване подання даних про її діяльність. З технічної точки зору це вказує на більш впорядковану структуру сайту та кращу основу для подальшого розширення функціональних можливостей.

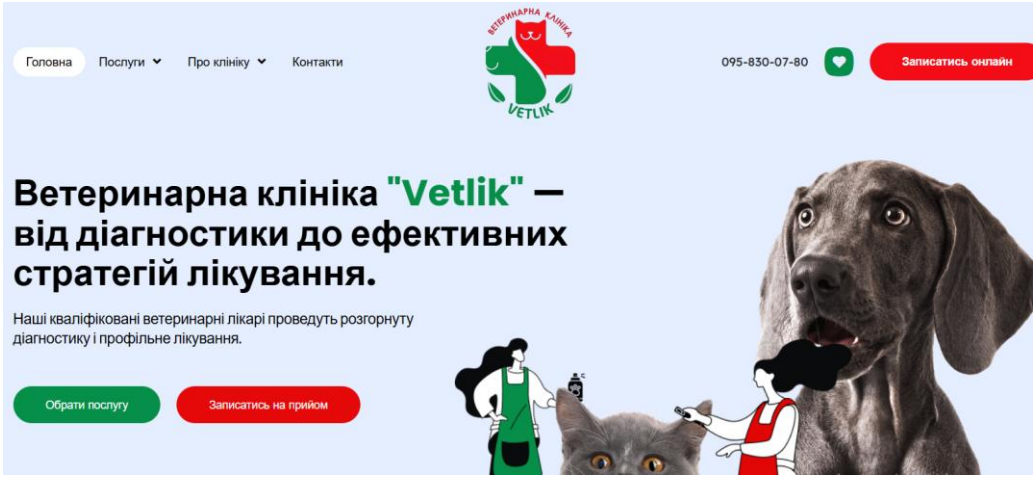


Рисунок 2.6. – Сайт VetLik

Сайт VetPixel також є повноцінним окремим вебресурсом ветеринарного закладу у Полтаві. На ньому подано перелік послуг, окремі сторінки конкретних процедур, контактну інформацію, відомості про графік роботи, а також розділ блогу. Особливістю цього ресурсу є детальніша подача окремих послуг із можливістю надсилання запиту, що свідчить про прагнення поєднати інформаційне представлення з елементами сервісної взаємодії. У загальному технічному аспекті це демонструє модульний підхід до побудови сайту, за якого інформація організовується у вигляді окремих функціональних сторінок, придатних для подальшого масштабування та адміністрування.

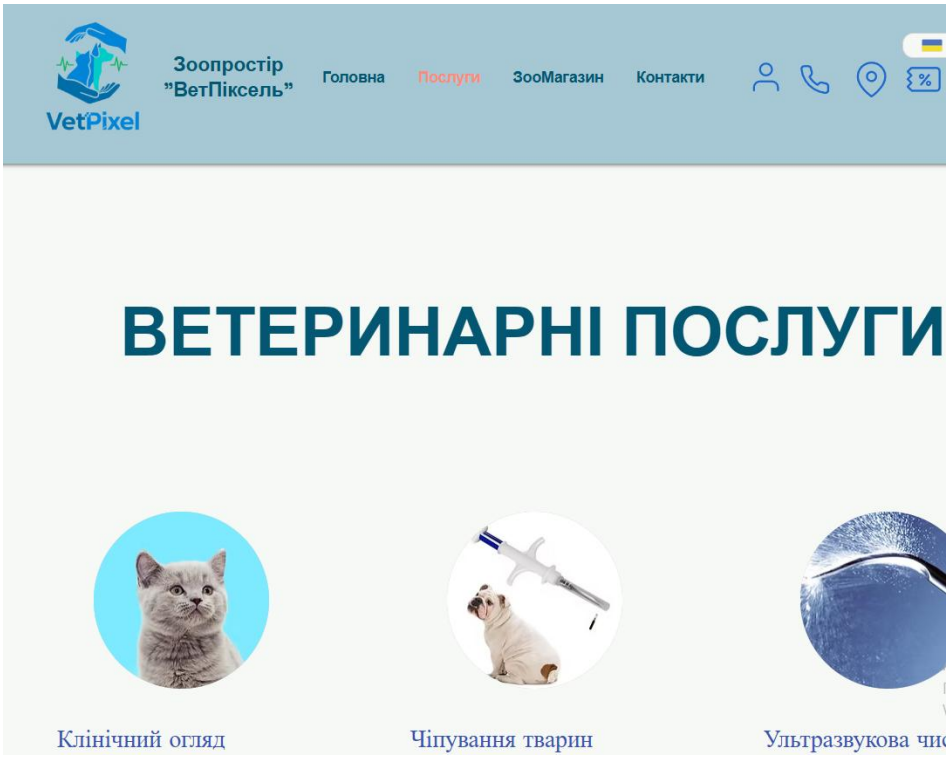


Рисунок 2.7. – Сайт VetPixel

Проведений аналіз показує, що в Полтаві кількість ветеринарних клінік, які мають власні окремо розроблені вебсайти, залишається відносно невеликою. Значна частина закладів представлена переважно сторінками у соціальних мережах, онлайн-каталогах або довідниках, тоді як повноцінні сайти з розвиненою структурою, сервісними можливостями та окремим доменом мають лише окремі клініки. Це свідчить про недостатній рівень поширення

комплексних вебрішень у цій сфері на локальному рівні та водночас підтверджує актуальність розробки власного вебзастосунку для ветеринарної клініки.

Отже, аналіз наведених вебресурсів дає змогу виділити кілька важливих загальних тенденцій. Сучасні сайти аналогічного призначення переважно поєднують інформаційні та сервісні функції, але відрізняються глибиною їх реалізації. Одні ресурси зосереджені на представленні основної інформації та базовому записі на прийом, інші мають складнішу структуру з багатомовністю, окремими тематичними розділами, зовнішніми сервісами взаємодії або персоналізованими кабінетами. У технічному аспекті це свідчить про важливість правильної структуризації сторінок, логічного поділу функціональних модулів, можливості масштабування ресурсу та підтримки інтеграції з додатковими сервісами. Саме ці особливості слід ураховувати під час розробки власного вебзастосунку для ветеринарної клініки.

2.2. Порівняльна характеристика сучасних вебзастосунків у даній сфері

Порівняльна характеристика сучасних вебзастосунків у сфері ветеринарного обслуговування дає змогу визначити не лише відмінності у функціональному наповненні, а й загальні підходи до технічної організації таких ресурсів, що буде важливо ураховувати для розробки нашого вебзастосунку. Для проведення порівняння використаємо ті самі вебресурси, які були розглянуті у попередньому розділі, а саме 103Vet, «ВетНова», Animal Clinic Live Expert, English Creek Veterinary Clinic, VetDomik, VetLik та VetPixel. Зіставлення саме цих прикладів дозволяє охопити кілька типів рішень: класичні інформаційні сайти, ресурси з онлайн-записом, сайти з винесенням частини функцій у зовнішні сервіси, а також більш комплексні системи з

персоналізованими елементами доступу.

Першою важливою ознакою для порівняння є структура функціональних можливостей. Сайт 103Vet забезпечує подання основної інформації про клініку, перелік послуг, контактні дані та можливість онлайн-запису, тобто реалізує компактну модель інформаційно-сервісного ресурсу. Сайт «ВетНова» має ширшу структуру, оскільки, окрім базових розділів, містить сторінки з корисною інформацією, правилами прийому, кар'єрними пропозиціями та двомовною навігацією. Animal Clinic Live Expert представляє окремий сервіс дистанційної консультації, де вебресурс використовується як засіб координації звернення та організації комунікації через зовнішні канали. English Creek Veterinary Clinic вирізняється наявністю PetPage Portal, що дає користувачеві доступ до записів, медичних даних і майбутніх вакцинацій, а це наближає ресурс до рівня повноцінної інформаційної системи. VetDomik поєднує інформацію про клініку, прайс, поширені запитання та запис на прийом. VetLik має окремі сторінки для послуг, команди, вакансій, блогу, контактів і запису. VetPixel, у свою чергу, містить каталог послуг із окремими картками, блог, сторінку про клініку та механізм подання запиту на послугу. Така різноманітність функціоналу свідчить про те, що сучасні вебзастосунки у даній сфері відрізняються не стільки наявністю базових інформаційних блоків, скільки рівнем глибини сервісної інтеграції.

Наступною важливою ознакою є організація користувацького сценарію запису на прийом. У 103Vet та «ВетНова» запис реалізовано через вбудовані форми на сторінках сайту, що забезпечує безперервність взаємодії користувача з ресурсом. У VetLik також передбачено окрему сторінку онлайн-запису, що вказує на інтеграцію сервісної функції в загальну структуру сайту. У VetDomik запис організовано через зовнішній віджет Vetmanager, тобто функція сервісної взаємодії частково винесена за межі основного сайту. У VetPixel замість класичного запису використано формат «запиту на послугу» для окремих процедур, що є спрощеним, але достатньо гнучким способом ініціювання

звернення. У випадку Animal Clinic Live Expert взаємодія ще більше зміщена у бік зовнішніх каналів, оскільки консультації проводяться через Skype, Viber, Telegram і WhatsApp. English Creek Veterinary Clinic демонструє найвищий рівень інтеграції, оскільки користувач отримує доступ до запису, рецептів та медичних даних через окремий портал пацієнта. Отже, за способом реалізації запису можна виділити три моделі: внутрішньосайтову, змішану із зовнішніми сервісами та порталізовану модель з персоналізованим доступом.

Значущою характеристикою є також організація контенту та модульність структури. У 103Vet і VetDomik інформаційні розділи мають більш лінійний і зрозумілий характер, орієнтований на швидкий доступ до ключових відомостей про послуги та контакти. У «ВетНова» та VetLik структура є більш розгалуженою, оскільки включає не лише базові сторінки, а й додаткові тематичні розділи, такі як правила прийому, вакансії, блог або персонал. У VetPixel спостерігається модульний підхід до представлення послуг, коли кожна процедура подається як окрема сторінка з власним описом, тривалістю та вартістю. У English Creek Veterinary Clinic модульність поєднується з багаторівневою навігацією та інтеграцією додаткових сервісів, наприклад онлайн-форми для нових пацієнтів, фінансових опцій та доступу до записів. Така відмінність у структурі показує, що вебзастосунки даної сфери можуть бути реалізовані як прості інформаційні сайти, як багатороздільні корпоративні ресурси або як системи з розширеними функціями самообслуговування користувача.

Порівняння технічних характеристик дає підстави для більш глибоких висновків. У частині сайтів можна встановити окремі деталі щодо використаних інструментів розробки. Зокрема, на сайті VetPixel у розділі політики конфіденційності прямо зазначено, що ресурс розміщено на платформі Wix.com, а дані користувачів зберігаються через інфраструктуру Wix. Це свідчить про використання конструкторної платформи, яка забезпечує швидке створення сайту, готові механізми форм, керування контентом і базові

засоби для реалізації сторінок послуг та блогу. На сайті VetLik у футері відкрито вказано, що розробку виконано iFish, тобто можна стверджувати, що цей ресурс створювався як окремий вебпроєкт, а не лише на базі стандартного шаблону без зовнішньої розробки. Для інших розглянутих сайтів точний технологічний стек не декларується у відкритій частині ресурсів, тому його коректно аналізувати лише за непрямими технічними ознаками, такими як наявність форм, порталу, багатомовності, окремих модулів або зовнішніх інтеграцій.

З технічної точки зору 103Vet і «ВетНова» можна віднести до категорії класичних багатосторінкових вебресурсів із формами зворотного зв'язку або попереднього запису. Для таких систем характерними є серверне або напівсерверне формування сторінок, логічно поділена навігація та орієнтація на інформаційно-сервісну модель взаємодії. VetDomik реалізує змішаний варіант, де основна інформаційна частина функціонує як цілісний сайт, але сервіс запису винесено до окремого спеціалізованого зовнішнього інструмента. VetPixel побудований за принципом сервісного сайту на платформі-конструкторі, що дозволяє швидко розгортати картки послуг, блогові матеріали та форми заявок, хоча така модель зазвичай має певні обмеження щодо гнучкості власної серверної логіки. VetLik демонструє ознаки індивідуально розробленого корпоративного сайту з окремими розділами і сторінками послуг. English Creek Veterinary Clinic вирізняється більш складною цифровою архітектурою, оскільки поєднує відкритий сайт, онлайн-форми, портал пацієнта, зовнішній застосунок та додаткові інструменти взаємодії з медичними записами. Animal Clinic Live Expert, у свою чергу, демонструє модель ресурсу, де частина функціоналу побудована на інтеграції з зовнішніми платформами комунікації, а не на повністю автономній вебреалізації.

Окремо слід відзначити питання масштабованості та перспектив розширення. Багатомовність «ВетНова» і VetDomik, наявність окремих сторінок вакансій і блогу у VetLik, каталогізована система послуг у VetPixel, а

також порталізована модель English Creek Veterinary Clinic свідчать про те, що сучасний ветеринарний вебресурс дедалі частіше виходить за межі простої візитівки. Масштабованість у таких умовах визначається не лише кількістю сторінок, а й можливістю нарощування функціональних модулів, інтеграції із зовнішніми сервісами, розмежування доступу до даних та підтримки актуальності контенту. Саме тому при розробці власного вебзастосунку важливо враховувати не тільки поточний перелік функцій, а й архітектурні передумови для подальшого розвитку системи.

Отже, проведене порівняння показало, що сучасні вебзастосунки у сфері ветеринарного обслуговування відрізняються за трьома основними напрямами: рівнем функціональної насиченості, способом технічної реалізації сервісних можливостей та ступенем структурної складності. Найпростішу модель формують інформаційні сайти з базовим записом на прийом, проміжну позицію займають ресурси зі змішаною організацією функціоналу та окремими сервісними інтеграціями, а найвищий рівень демонструють рішення з персоналізованими кабінетами і доступом до індивідуальних даних пацієнта. Водночас аналіз відкритих джерел показує, що точне визначення використаного програмного стеку можливе не для всіх сайтів, тому коректніше спиратися на встановлювані технічні ознаки, а не на непідтверджені припущення. Такий підхід створює підґрунтя для подальшого розгляду переваг і недоліків існуючих рішень у наступному підрозділі.

2.3. Переваги та недоліки існуючих програмних рішень

Однією з основних переваг більшості розглянутих рішень є наявність чітко структурованого інформаційного наповнення. На сайтах 103Vet, «ВетНова», VetDomik, VetLik і VetPixel користувачеві доступні окремі сторінки з описом послуг, контактною інформацією, даними про персонал або клініку,

що позитивно впливає на логіку навігації та зручність пошуку необхідних відомостей. Особливо важливо, що частина цих ресурсів містить додаткові тематичні розділи, зокрема правила прийому у «ВетНова», вакансії у VetLik, відповіді на запитання і прайс у VetDomik, а також блог і деталізовані картки процедур у VetPixel. Це свідчить про прагнення зробити сайт не лише засобом презентації закладу, а й повноцінним інформаційним середовищем.

Суттєвою перевагою окремих рішень є реалізація сервісної складової. На 103Vet і «ВетНова» передбачено онлайн-запис через форму на сайті, що забезпечує безперервну взаємодію користувача з вебресурсом. VetDomik також підтримує запис на прийом, хоча й через зовнішній інструмент, а Animal Clinic додатково пропонує Live-консультацію з вибором каналу зв’язку через Skype, Viber, Telegram або WhatsApp. Найвищий рівень сервісної цифровізації демонструє English Creek Veterinary Clinic, де через PetPage Portal користувач може записувати тварину на прийом, працювати з рецептами, переглядати медичні записи та майбутні вакцинації. Отже, перевага таких ресурсів полягає в тому, що вони забезпечують не лише доступ до загальної інформації, а й надають інструменти для реальної взаємодії з клінікою.

Ще однією важливою перевагою є наявність ознак масштабованості та функціонального розвитку. Наприклад, сайт «ВетНова» підтримує українську та російську мовні версії, що свідчить про розширення аудиторії користувачів і більш складну організацію контенту. VetDomik використовує структуру з окремими сторінками для різних міст, що демонструє можливість масштабування ресурсу на кілька локацій. VetLik і VetPixel мають додаткові розділи, не обмежені лише базовими сторінками клініки, а English Creek Veterinary Clinic інтегрує публічний сайт із персоналізованим порталом пацієнта. Такі ознаки дозволяють розглядати ці рішення як системи, придатні до подальшого розвитку, а не лише як статичні вебсторінки.

Разом із тим проведений аналіз дозволяє виявити і низку недоліків. Насамперед слід зазначити, що не всі ресурси однаково глибоко інтегрують

сервісні функції у власну структуру. У випадку VetDomik запис на прийом реалізовано через зовнішній сервіс, а в Animal Clinic частина взаємодії з користувачем винесена до сторонніх месенджерів і платіжних механізмів. Такий підхід має певні переваги з точки зору швидкості реалізації, однак у технічному відношенні він означає часткову залежність від зовнішніх платформ, обмеження щодо цілісного керування користувацьким сценарієм і менший контроль над усією сервісною логікою в межах одного вебзастосунку.

Іншим недоліком частини рішень є обмежена персоналізація взаємодії. Більшість розглянутих українських сайтів орієнтовані переважно на подання загальної інформації, форми запису або консультації, однак не надають користувачеві доступу до повноцінного персонального кабінету, історії звернень, електронних записів чи індивідуальних даних про тварину. На цьому тлі English Creek Veterinary Clinic вигідно вирізняється завдяки PetPage Portal. Відсутність подібного функціоналу на більшості локальних ресурсів свідчить про обмеженість їх цифрової зрілості та про те, що сервісна взаємодія в багатьох випадках залишається лише частково автоматизованою.

З технічної точки зору недоліком є також те, що частина ресурсів побудована на платформах або рішеннях, які хоча й спрощують створення сайту, але можуть обмежувати гнучкість подальшого розвитку. Зокрема, у VetPixel прямо зазначено, що сайт розміщено на платформі Wix.com. Такий підхід забезпечує швидке створення сторінок, форм і базових сервісів, але водночас може накладати певні обмеження на реалізацію власної серверної логіки, нетипових модулів та складної інтеграції з внутрішніми системами. Для невеликих інформаційних ресурсів це може бути прийнятним, однак для повноцінного вебзастосунку з розвиненою логікою керування записами, даними персоналу, послугами та статусами подібна модель не завжди є достатньо гнучкою.

Окремо слід звернути увагу на те, що у відкритому доступі далеко не завжди можна встановити точний стек технологій, на якому реалізовано

конкретний ресурс. Це само по собі не є недоліком для кінцевого користувача, однак з позиції аналізу програмних рішень ускладнює оцінювання архітектурних підходів, рівня захисту даних, способів обробки запитів і перспектив технічного супроводу. Тому для значної частини розглянутих сайтів технічні висновки доводиться робити переважно за непрямими ознаками, такими як багатомовність, наявність форм, зовнішніх інтеграцій, модульної структури чи персоналізованого порталу. Це підтверджує, що для власного дипломного проєкту важливо забезпечити не лише функціональність, а й внутрішню логічність архітектури та прозорість організації системи.

Якщо оцінювати розглянуті рішення в цілому, можна зробити висновок, що їх переваги полягають у наявності структурованого контенту, реалізації базових сервісних сценаріїв, зручному поданні інформації та поступовому переході до більш інтегрованих цифрових моделей взаємодії. Недоліки пов’язані з фрагментарністю сервісних функцій, залежністю від зовнішніх платформ, недостатнім рівнем персоналізації та не завжди достатньою технічною гнучкістю для реалізації складнішої внутрішньої логіки. Для умов Полтави додатково слід відзначити, що кількість ветеринарних клінік із власними повноцінними сайтами залишається відносно невеликою, а це ще раз підтверджує актуальність розробки вебзастосунку, який поєднуватиме інформаційні, сервісні та адміністративні функції в межах єдиної системи.

Таким чином, результати порівняльного аналізу, отримані із відкриих джерел, дають підстави стверджувати, що ефективне програмне рішення у даній сфері повинно не обмежуватися лише представленням інформації про клініку. Воно має забезпечувати цілісну організацію даних, зручні механізми запису на прийом, можливість адміністрування вмісту, гнучку структуру розширення функціоналу та технічно впорядковану архітектуру. Саме врахування виявлених переваг і недоліків існуючих рішень є важливим підґрунтям для обґрунтування підходів до розробки власного вебзастосунку в наступних розділах дипломної роботи.

2.4. Сучасні підходи до розробки вебзастосунків

Розвиток вебтехнологій зумовив формування нових підходів до створення програмних рішень, орієнтованих не лише на представлення інформації, а й на забезпечення повноцінної взаємодії користувача із системою. Сучасний вебзастосунок розглядається як багатокомпонентне програмне середовище, у якому поєднуються серверна логіка, засоби роботи з даними, механізми формування інтерфейсу та інструменти забезпечення зручності користування. Саме тому під час розробки таких систем особливого значення набувають питання архітектурної впорядкованості, масштабованості, підтримуваності та адаптивності інтерфейсу. Платформа ASP.NET Core позиціонується як високопродуктивний кросплатформний фреймворк для створення сучасних вебзастосунків, а MVC визначається як один із базових підходів до організації вебрішень на цій платформі [1, 2, 5-7, 9, 14-20].

Одним із найбільш поширених сучасних підходів є побудова вебзастосунку на основі архітектурного шаблону Model-View-Controller. Його сутність полягає у поділі програмного рішення на три логічні складові: моделі, що відповідають за дані й правила предметної області, представлення, які забезпечують відображення інформації, та контролери, що обробляють запити користувача і координують взаємодію між іншими компонентами. Такий підхід сприяє чіткому розмежуванню відповідальності, спрощує супровід коду та полегшує модифікацію окремих частин системи без істотного впливу на її загальну структуру. У сучасній практиці MVC залишається актуальним для створення вебзастосунків, де важливо поєднати серверну обробку запитів, роботу з даними та генерацію динамічних сторінок [5, 9].

Іншим важливим напрямом є орієнтація на сторінково-центровані підходи до побудови інтерфейсу. У межах екосистеми ASP.NET Core таку модель реалізує Razor, який дає змогу поєднувати HTML-розмітку з кодом

серверної логіки та формувати динамічні сторінки на основі переданих даних. Документація Microsoft підкреслює, що Razor підходить для сторінково-орієнтованих сценаріїв і забезпечує ефективну побудову вебінтерфейсу, зокрема в тих випадках, коли потрібне серверне рендерування вмісту. У сучасних вебзастосунках це має важливе значення, оскільки дозволяє забезпечити швидке формування початкової відповіді, логічну структуру сторінок і зручне поєднання візуального шаблону з даними, отриманими з контролерів або моделей [1-4, 6, 7].

Суттєвим сучасним підходом до розробки є використання об’єктно-реляційного відображення для роботи з базою даних. Застосування ORM дає змогу працювати з даними на рівні програмних об’єктів, зменшуючи обсяг низькорівневого коду доступу до бази даних і забезпечуючи більш зручну організацію логіки збереження та отримання інформації. Entity Framework Core позиціонується як легкий, розширюваний і кросплатформний ORM, який дозволяє працювати з базою даних через об’єкти .NET, підтримує LINQ-запити, відстеження змін і механізми міграцій. Такий підхід є важливим для сучасних вебзастосунків, оскільки дає можливість будувати більш структуровану модель даних, гнучко керувати сутностями та забезпечувати підтримуваність системи при її подальшому розширенні [3].

Окремої уваги заслуговує підхід, пов’язаний із побудовою адаптивного користувацького інтерфейсу. Сучасні вебзастосунки повинні коректно відображатися на різних типах пристроїв, тому важливою вимогою є підтримка responsive design. У цьому контексті широко застосовуються HTML5, CSS3, JavaScript і готові CSS-фреймворки, зокрема Bootstrap. Офіційна документація Bootstrap визначає його як один із найпоширеніших інструментів для створення responsive, mobile-first інтерфейсів, а система breakpoint-ів розглядається як базовий механізм адаптації макета до різних розмірів екрана. У сучасній практиці це означає, що інтерфейс повинен будуватися не лише з урахуванням візуальної привабливості, а й з урахуванням доступності, зручності навігації,

узгодженості елементів керування та коректного відображення на різних пристроях [20].

Ще одним важливим сучасним підходом є побудова вебзастосунку з орієнтацією на модульність і розширюваність. У сучасних архітектурних рекомендаціях для вебрішень підкреслюється важливість виокремлення окремих рівнів системи, зокрема рівня представлення, рівня бізнес-логіки та рівня доступу до даних. Такий поділ дає змогу локалізувати зміни, зменшує зв’язність між компонентами та створює основу для подальшого розвитку системи. У практичній площині це реалізується через виділення окремих сервісів, репозиторіїв, моделей представлення і допоміжних модулів, що забезпечують цілісність внутрішньої структури застосунку. Саме тому сучасний вебзастосунок розглядається не як набір окремих сторінок, а як ієрархічно організована система компонентів.

Поряд із цим сучасна розробка вебзастосунків орієнтується на спрощення супроводу та підвищення довготривалої життєздатності програмного продукту. Це означає, що вибір технологій і архітектурних підходів здійснюється не лише з позиції швидкості первинної реалізації, а й з урахуванням перспектив оновлення, масштабування та інтеграції з іншими підсистемами. Актуальні версії ASP.NET Core та EF Core підтримують довгострокові сценарії використання, розвиток функціональності та інтеграцію з іншими компонентами .NET-екосистеми, що робить їх придатними для створення не лише простих інформаційних сайтів, а й більш складних корпоративних вебзастосунків [1-7, 9].

Таким чином, сучасні підходи до розробки вебзастосунків ґрунтуються на поєднанні архітектурної впорядкованості, ефективної роботи з даними, адаптивного інтерфейсу та модульної організації системи. Використання MVC забезпечує логічне розділення відповідальності між компонентами, Razor дає змогу формувати динамічні сторінки, ORM-підхід у вигляді Entity Framework Core спрощує взаємодію з базою даних, а Bootstrap [11-13] і засоби клієнтської

розробки створюють основу для побудови сучасного адаптивного інтерфейсу. Сукупність цих підходів формує методологічне підґрунтя для створення вебзастосунку, який відповідає вимогам до функціональності, підтримуваності та подальшого розвитку.

РОЗДІЛ 3. ТЕОРЕТИЧНА ЧАСТИНА

3.1. Характеристика використаних технологій і засобів розробки

Під час розробки вебзастосунку для ветеринарної клініки важливо було забезпечити структуровану організацію серверної логіки, надійне зберігання даних, зручний інтерфейс користувача, коректну обробку записів на прийом, а також можливість адміністрування інформаційного наповнення. Для досягнення цих цілей було використано сучасні технології та підходи, які відповідають вимогам до побудови прикладних вебсистем. Нижче детально описано основні засоби, застосовані у проєкті, та їх роль у реалізації функціональних можливостей системи.

ASP.NET Core (MVC).

ASP.NET Core використано як основний фреймворк для побудови серверної частини вебзастосунку. Його застосування дало змогу реалізувати логіку обробки HTTP-запитів, налаштувати маршрутизацію, організувати взаємодію між контролерами та представленнями, а також забезпечити запуск і функціонування всього програмного рішення. У проєкті ця технологія становить основу для реалізації всіх основних функцій, пов’язаних із відображенням сторінок, обробкою форм, записом користувачів на прийом та керуванням адміністративною частиною системи [1-4, 6, 7].

Використання архітектури MVC забезпечило чіткий поділ між обробкою запитів, представленням даних і логікою роботи із сутностями предметної області. Завдяки цьому вдалося відокремити функціональну поведінку контролерів від інтерфейсної частини та моделей даних. Такий підхід підвищує зрозумілість коду, полегшує супровід вебзастосунку та створює передумови для подальшого розширення функціоналу [5, 9].

Entity Framework Core.

Для взаємодії з базою даних у проєкті використано Entity Framework Core. Ця технологія виступає як ORM-засіб, що забезпечує роботу з реляційною базою даних через об’єкти мови C#. Завдяки її використанню основні сутності системи, зокрема працівники, послуги, записи на прийом, категорії, публікації, коментарі, теги та інші інформаційні об’єкти, подано у вигляді класів, що значно спрощує доступ до даних і керування ними.

Entity Framework Core дозволив організувати збереження, оновлення, пошук і видалення записів без необхідності постійного ручного написання SQL-запитів. Крім того, цей засіб дає змогу задавати обмеження для полів, конфігурувати зв’язки між сутностями та підтримувати цілісність структури бази даних. Для проєкту ветеринарної клініки це має особливе значення, оскільки система оперує великою кількістю взаємопов’язаних даних, які мають бути впорядковано збережені й доступні для подальшого використання [3] .

LINQ.

Для формування запитів до бази даних і обробки колекцій у проєкті використано LINQ. Застосування цієї технології дало змогу реалізувати зручний і декларативний спосіб вибірки, фільтрації, сортування та групування даних. LINQ використовується під час отримання списків послуг, працівників, категорій, публікацій, записів на прийом, а також у процесі оновлення окремих сутностей та роботи із пов’язаними даними.

Використання LINQ підвищує читабельність програмного коду та робить логіку доступу до даних більш зрозумілою. Це особливо важливо у випадках, коли вебзастосунок повинен обробляти значну кількість інформаційних об’єктів і підтримувати різні сценарії їх подання у користувацькому та адміністративному інтерфейсах. Таким чином, LINQ є важливим інструментом реалізації внутрішньої логіки обробки даних у системі.

Microsoft SQL Server.

Як основне сховище даних у вебзастосунку використано Microsoft SQL Server. Вибір цієї системи керування базами даних зумовлений її надійністю,

підтримкою реляційної моделі, продуктивністю та широкими можливостями інтеграції із платформою .NET. У межах проєкту SQL Server забезпечує збереження структурованої інформації про всі ключові об’єкти системи, включаючи записи на прийом, працівників, послуги, контентні матеріали, коментарі, навігаційні елементи та користувачів [10].

Застосування Microsoft SQL Server є доцільним для вебзастосунку такого типу, оскільки система повинна підтримувати цілісність даних, швидкий доступ до них та коректну роботу зі зв’язаними таблицями. Для інформаційної системи ветеринарної клініки це є принципово важливим, оскільки база даних виступає основою функціонування як публічної, так і адміністративної частин застосунку.

Razor.

Для формування користувацького інтерфейсу у проєкті застосовано Razor. Ця технологія використовується для генерації динамічних вебсторінок на сервері та поєднання HTML-розмітки з даними, отриманими з контролерів. З її допомогою реалізовано представлення основних сторінок сайту, зокрема головної сторінки, сторінок із працівниками, послугами, публікаціями, інформаційними розділами та адміністративними формами.

Використання Razor є доцільним для серверного багатосторінкового вебзастосунку, оскільки цей шаблонізатор дозволяє логічно поєднувати відображення даних з елементами програмної логіки. У межах даного проєкту це забезпечує гнучке формування сторінок на основі моделей представлення, а також дає змогу реалізувати виведення структурованої інформації без переходу до складніших клієнтських SPA-підходів [4].

HTML5, CSS3, JavaScript.

Основою клієнтської частини вебзастосунку є HTML5, CSS3 та JavaScript. HTML5 використовується для формування структурної розмітки сторінок, побудови форм, таблиць, навігаційних блоків і секцій контенту. CSS3 забезпечує стилізацію елементів інтерфейсу, узгоджене оформлення сторінок і

візуальну цілісність вебресурсу. JavaScript використовується для реалізації окремих динамічних можливостей інтерфейсу та підтримки інтерактивної поведінки окремих елементів сторінки [20].

Поєднання цих технологій дозволило створити сучасний вебінтерфейс, який підтримує зручну навігацію, структуроване подання інформації та комфортну взаємодію користувача з функціональними елементами системи. У випадку ветеринарного вебзастосунку це особливо важливо, оскільки сайт повинен одночасно виконувати представницьку, інформаційну та сервісну функції.

Bootstrap.

Для побудови адаптивного інтерфейсу у проєкті використано Bootstrap. Цей CSS-фреймворк застосовано для оформлення форм, кнопок, таблиць, карток, навігаційних елементів та інших складових користувацького інтерфейсу. Його використання дозволило забезпечити єдиний візуальний стиль застосунку та спростити реалізацію responsive-дизайну [11, 12].

Bootstrap є доцільним вибором для подібного проєкту, оскільки він надає готові стилістичні компоненти, що значно скорочують час розробки інтерфейсу та водночас забезпечують його узгодженість. Для вебзастосунку ветеринарної клініки це дало можливість реалізувати зрозумілі форми запису на прийом, впорядковане подання довідкової інформації та зручне відображення контенту на різних типах пристроїв.

Repository Pattern.

У структурі проєкту використано патерн Repository, який застосовується для організації доступу до даних. Його сутність полягає у винесенні логіки роботи з окремими сутностями в спеціалізовані класи, що інкапсулюють операції створення, читання, оновлення та видалення записів. У межах цього проєкту окремі репозиторії відповідають за роботу з працівниками, послугами, категоріями, публікаціями, записами на прийом, навігаційними елементами та іншими об’єктами системи.

Застосування Repository Pattern дозволило зменшити залежність контролерів від безпосередньої роботи з контекстом бази даних, підвищити модульність коду та зробити систему більш придатною до супроводу. Це є важливою перевагою для дипломного проєкту, оскільки така побудова демонструє не лише працездатність програмного продукту, а й використання сучасних архітектурних підходів до розробки вебзастосунків.

DataAnnotations.

Для перевірки коректності введених користувачем даних у проєкті застосовано DataAnnotations. Цей механізм використовується у моделях і моделях представлення для задання обов’язкових полів, обмежень формату, валідації електронної пошти, телефонного номера, а також для опису правил введення текстових значень. Завдяки цьому система може автоматично перевіряти правильність заповнення форм перед надсиланням або збереженням даних.

Застосування DataAnnotations є важливим з точки зору забезпечення якості даних, оскільки вебзастосунок обробляє інформацію, що надходить від користувачів через форми запису на прийом, а також через адміністративні модулі створення і редагування сутностей. Використання цього механізму підвищує надійність системи та зменшує ймовірність появи помилкових або неповних записів у базі даних.

FileService.

Для роботи із завантаженням і збереженням зображень у проєкті використано FileService. Цей сервіс забезпечує обробку файлів, що додаються під час створення або редагування інформаційних об’єктів, зокрема працівників, категорій і послуг. Його використання дозволяє винести логіку файлової обробки в окремий програмний модуль та уникнути дублювання однакових дій у різних частинах системи.

Наявність такого сервісу є важливою для вебзастосунку ветеринарної клініки, оскільки візуальне представлення контенту має суттєве значення для

користувача. Додавання зображень до послуг, сторінок працівників або інформаційних розділів підвищує інформативність сайту та покращує його сприйняття. Тому FileService виступає важливим допоміжним компонентом практичної реалізації системи.

Обробка записів на прийом і enum-статуси.

Однією з ключових функціональних складових вебзастосунку є підсистема запису на прийом. Для її реалізації у проєкті передбачено окрему модель запису, що містить дані про користувача, дату звернення, контактну інформацію та додаткове повідомлення. Окрім збереження самого запису, система підтримує його подальшу обробку через статуси, що подані за допомогою enum.

Використання enum для подання статусів записів дозволяє забезпечити однозначність значень і впорядкованість логіки роботи із заявками. Такий підхід є зручним як для внутрішньої програмної обробки, так і для адміністративного контролю за станом записів. У результаті система підтримує не лише приймання заявок, а й їх подальше адміністрування, що підвищує практичну цінність розробленого вебзастосунку.

Вибір наведених технологій і підходів був зумовлений необхідністю створення повноцінного вебзастосунку, який поєднує представлення інформації, сервісну взаємодію з клієнтами та адміністративне керування вмістом. ASP.NET Core забезпечує надійну основу для побудови серверної частини системи. Entity Framework Core, LINQ і Microsoft SQL Server дозволяють ефективно організувати роботу з даними. MVC та Repository Pattern забезпечують структурованість програмного коду і логічне розмежування відповідальності між компонентами. Razor, HTML5, CSS3, JavaScript і Bootstrap створюють основу для побудови зручного та адаптивного інтерфейсу. DataAnnotations і FileService реалізують важливі допоміжні механізми валідації та файлової обробки, а enum-статуси забезпечують коректну обробку записів на прийом.

Отже, використані у проєкті технології та засоби розробки утворюють цілісну технічну основу для реалізації вебзастосунку ветеринарної клініки. Їх поєднання забезпечує функціональність, підтримуваність, зручність використання та можливість подальшого розвитку системи відповідно до потреб предметної області.

3.2. Проєктування структури та основних модулів вебзастосунку

Проєктування структури вебзастосунку ветеринарної клініки здійснювалося з урахуванням необхідності поєднання інформаційної, сервісної та адміністративної складових у межах єдиної системи. Архітектурною основою програмного рішення став підхід MVC, відповідно до якого структура застосунку поділяється на моделі, представлення та контролери, що сприяє логічному розмежуванню відповідальності між компонентами системи. Така організація дозволяє відокремити обробку запитів користувача від логіки роботи з даними та відображення результатів, завдяки чому система набуває впорядкованої внутрішньої структури, стає більш зрозумілою для супроводу та придатною до подальшого розширення [5, 9].

У межах цього підходу контролери виконують функцію приймання та обробки HTTP-запитів, моделі забезпечують подання та збереження даних предметної області, а представлення відповідають за формування користувацького інтерфейсу. Саме така побудова дала змогу реалізувати вебзастосунок не як набір окремих сторінок, а як цілісну інформаційну систему, у якій усі компоненти взаємодіють між собою за чітко визначеною логікою. Завдяки цьому стало можливим організувати як публічну частину сайту для відвідувачів, так і адміністративну частину для керування вмістом, послугами, персоналом і записами на прийом.

Загальна структура застосунку сформована як багаторівнева система. На верхньому рівні знаходиться інтерфейс користувача, через який здійснюється взаємодія з вебресурсом. До цього рівня належать сторінки перегляду інформації про клініку, працівників, послуги, публікації, а також форми введення даних, зокрема форма запису на прийом. Наступний рівень утворює логіка обробки запитів, де контролери забезпечують приймання введених даних, звернення до відповідних модулів системи та підготовку моделей для подальшого відображення. Далі функціонує рівень доступу до даних, у межах якого реалізовано спеціалізовані механізми роботи з основними сутностями системи. Основою ж усього застосунку є база даних, у якій централізовано зберігається інформація про працівників, послуги, записи на прийом, публікації, категорії, коментарі, теги, навігаційні елементи, параметри сайту та користувачів.

Важливим етапом проєктування була побудова моделі предметної області. У системі передбачено сукупність сутностей, які відображають основні об’єкти роботи ветеринарної клініки. До них належать працівники, послуги, записи на прийом, інформаційні публікації, категорії, коментарі, теги, елементи навігації, параметри конфігурації та користувачі адміністративної частини. Така модель дозволяє забезпечити не лише відображення інформації на сторінках сайту, а й підтримку адміністративних операцій, пов’язаних зі створенням, оновленням, видаленням і впорядкуванням даних. Отже, ще на етапі проєктування структура застосунку орієнтувалася на комплексне використання системи, а не лише на виконання представницької функції.

Одним із центральних модулів вебзастосунку є модуль публічної частини сайту. Його призначення полягає у наданні користувачеві структурованої інформації про діяльність клініки. У межах цього модуля реалізується відображення головної сторінки, сторінок із переліком послуг, відомостей про працівників, публікацій та інших інформаційних розділів. Цей модуль забезпечує первинну взаємодію користувача із системою, формує загальне

уявлення про клініку та надає доступ до її основних сервісів. Його побудова повинна бути зрозумілою, логічною та такою, що забезпечує швидкий доступ до ключової інформації.

Наступним важливим модулем є модуль запису на прийом. Його поява у структурі системи є принциповою, оскільки саме він переводить сайт із категорії звичайного інформаційного ресурсу до рівня прикладного вебзастосунку з сервісною складовою. У межах цього модуля користувач має можливість ввести персональні дані, контактну інформацію, бажану дату прийому та додаткове повідомлення. Після цього система виконує перевірку правильності введених значень і зберігає звернення у базі даних для подальшого опрацювання. Таким чином, модуль запису на прийом виконує роль проміжної ланки між клієнтом і клінікою та є одним із найбільш практично значущих компонентів усього застосунку.

Окреме місце у структурі вебзастосунку займає модуль керування контентом. Його функціональне призначення полягає у створенні, редагуванні, збереженні та відображенні інформаційних матеріалів. У межах цього модуля реалізується робота з публікаціями, категоріями, тегами, коментарями, а також елементами навігації та окремими параметрами сайту. Така побудова дозволяє підтримувати актуальність інформаційного наповнення вебресурсу, змінювати його структуру відповідно до потреб клініки та забезпечувати динамічне оновлення вмісту без втручання в основну логіку системи.

Ще одним важливим функціональним блоком є модуль керування довідковими сутностями. До нього належать операції з працівниками, послугами, категоріями послуг та іншими об’єктами, що формують основний зміст сайту. Саме через цей модуль підтримується актуальність відомостей про клініку, її спеціалістів та перелік послуг. Наявність такого окремого блоку у структурі застосунку є виправданою, оскільки дозволяє впорядкувати адміністративну роботу з базовими інформаційними сутностями та уникнути змішування її з іншими функціональними процесами.

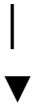
Важливим елементом проєктування системи є адміністративний модуль. Він забезпечує централізоване керування всіма ключовими об’єктами застосунку і виконує службову функцію. Через нього здійснюється робота з працівниками, послугами, записами на прийом, категоріями, публікаціями, коментарями, тегами, навігацією та іншими даними. Така організація дозволяє відокремити відкриту користувацьку частину від закритої частини адміністрування, що є важливою вимогою для системи подібного типу. Адміністративний модуль забезпечує підтримку цілісності структури сайту та надає інструменти для його повноцінного супроводу.

Окремо слід виділити модуль роботи з файлами та зображеннями. Його функція полягає у забезпеченні завантаження, перевірки та збереження графічних матеріалів, які використовуються в різних частинах сайту. Такий модуль має важливе значення для практичної реалізації застосунку, оскільки візуальне оформлення сторінок працівників, послуг чи категорій суттєво впливає на сприйняття ресурсу користувачем. Централізація логіки роботи з файлами дозволяє зробити систему більш упорядкованою та уникнути дублювання однакових операцій у різних частинах програми.

Логіка взаємодії між основними модулями вебзастосунку може бути відображена за допомогою блок-схеми, яка демонструє загальний рух даних і запитів усередині системи.

Загальну логіку взаємодії між модулями вебзастосунку доцільно подати у вигляді блок-схеми.

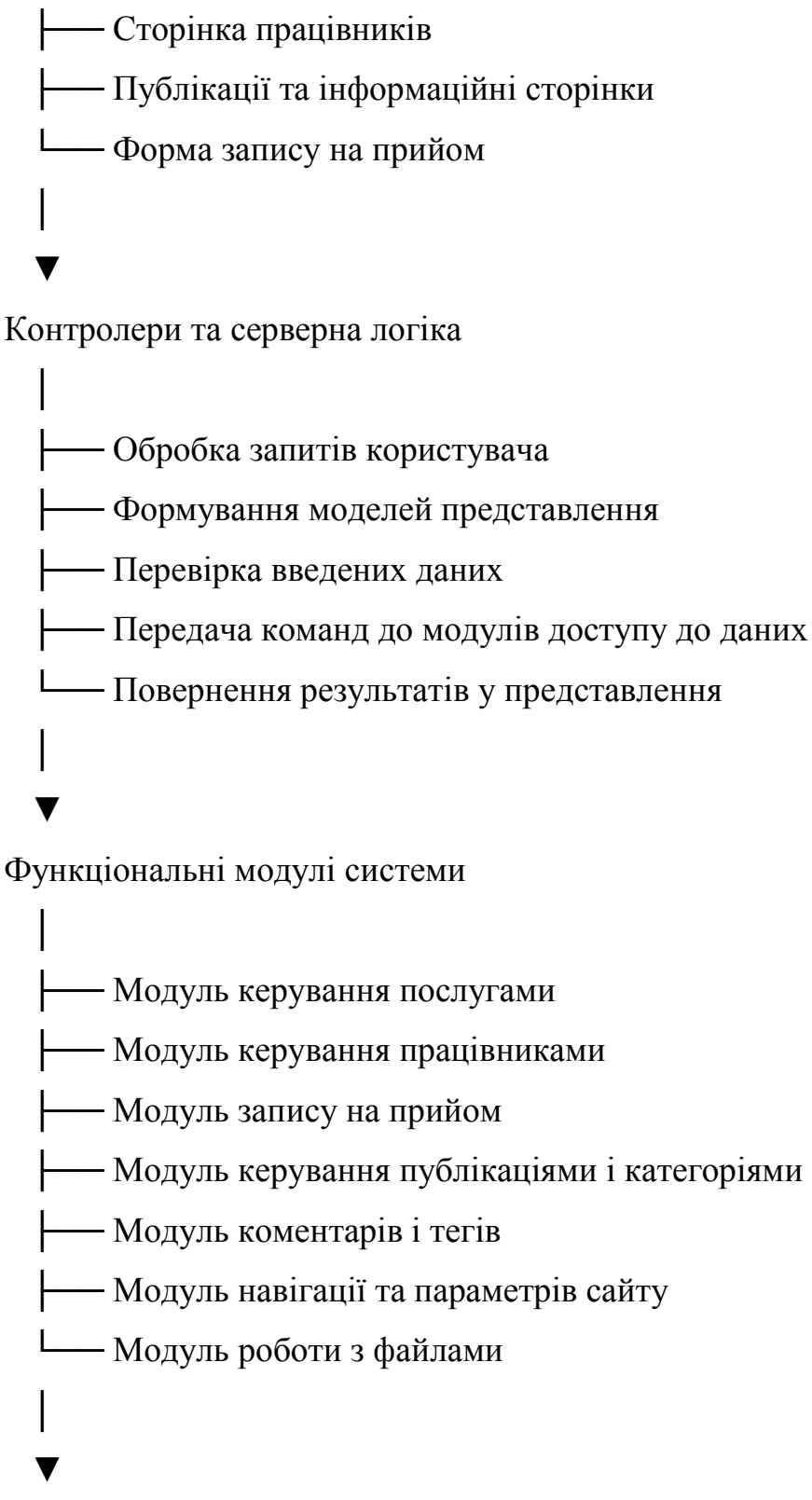
Користувач

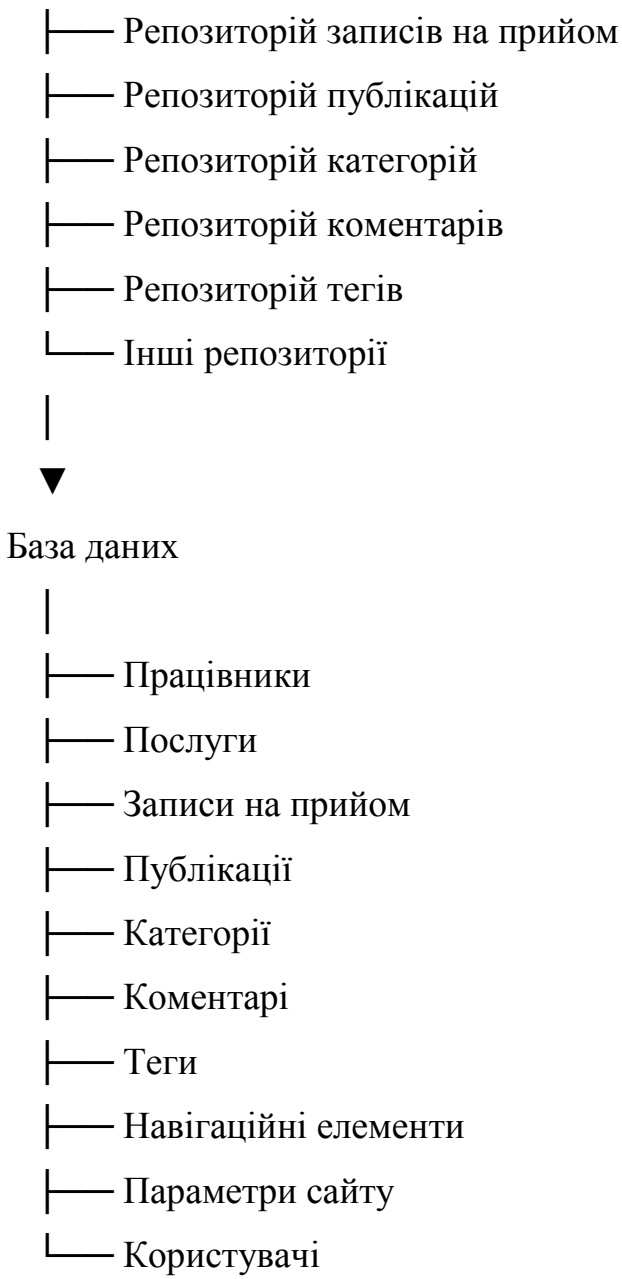


Користувацький інтерфейс вебзастосунку

├── Головна сторінка

├── Сторінка послуг


├── Сторінка працівників
├── Публікації та інформаційні сторінки
└── Форма запису на прийом
▼
Контролери та серверна логіка
├── Обробка запитів користувача
├── Формування моделей представлення
├── Перевірка введених даних
├── Передача команд до модулів доступу до даних
└── Повернення результатів у представлення
▼
Функціональні модулі системи
├── Модуль керування послугами
├── Модуль керування працівниками
├── Модуль запису на прийом
├── Модуль керування публікаціями і категоріями
├── Модуль коментарів і тегів
├── Модуль навігації та параметрів сайту
└── Модуль роботи з файлами
▼
Рівень доступу до даних
├── Репозиторій послуг
├── Репозиторій працівників



Крім загальної блок-схеми, доцільно окремо подати спрощену схему логіки запису на прийом, оскільки цей сценарій є одним із центральних для всього проєкту.

Рисунок 3. 1. – Блок–схема запису на прийом веб застосунку

Отже, проєктування структури вебзастосунку виконано на основі модульного підходу, за якого окремі частини системи відповідають за власний функціональний напрям, але працюють у межах єдиної архітектурної логіки. Така організація забезпечує зрозумілу побудову програмного рішення, підтримує розмежування функцій між рівнями системи та створює основу для ефективної реалізації як публічної, так і адміністративної частини вебзастосунку. Запропонована структура повністю відповідає завданням автоматизації роботи ветеринарної клініки та відображає логіку побудови розробленого програмного продукту.

3.3. Опис архітектури системи та взаємодії її компонентів

Архітектура розробленого вебзастосунку побудована на основі підходу MVC, у межах якого логіка обробки запитів, подання даних та їх відображення розмежовані між окремими компонентами системи. Така організація простежується як у способі формування користувацьких сценаріїв, так і в побудові адміністративної частини, механізмах доступу до даних і використанні моделей представлення. У застосунку поєднано контролери для обробки HTTP-запитів, репозиторії для роботи з даними, сутності бази даних, DTO для передавання значень у формах створення і редагування, а також ViewModel для підготовки даних до відображення на сторінках. Це забезпечує логічне розмежування відповідальності між компонентами системи та підвищує впорядкованість програмного рішення.

У межах даної архітектури контролери виконують роль центрального координаційного рівня. Саме вони приймають запити, викликають відповідні модулі доступу до даних, формують моделі для представлень і повертають результат користувачеві. Наприклад, HomeController об’єднує дані про послуги, працівників і останні публікації для головної сторінки, а також обробляє надсилання форми запису на прийом. AboutController відповідає за сторінку про клініку та обробку повідомлень з форми зворотного зв’язку. AccountController реалізує автентифікацію користувачів адміністративної частини, а AdminController забезпечує централізоване керування контентом, довідковими сутностями та записами на прийом. Наявність також CategoryController, PostController, OptionController, AjaxController і ErrorController свідчить про те, що система контролерів охоплює як публічну частину ресурсу, так і службові процеси керування даними.

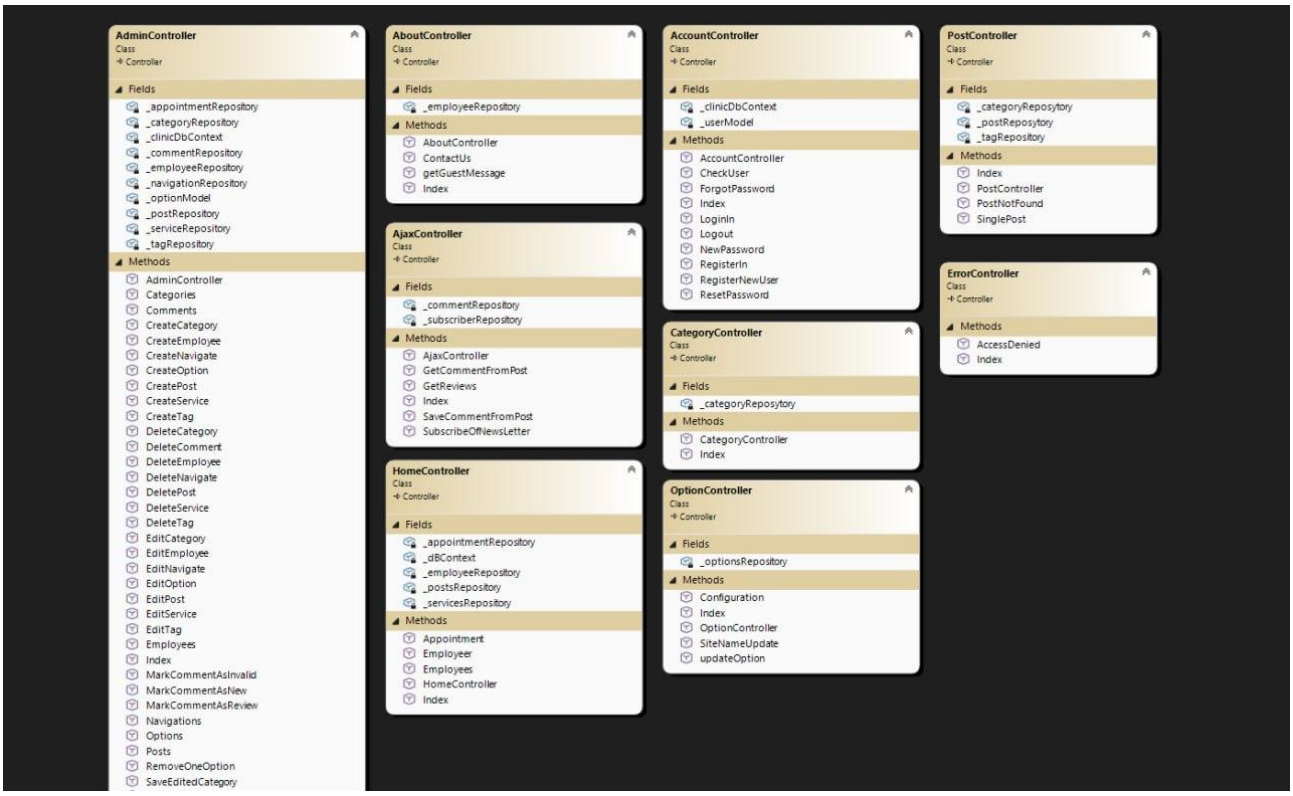


Рисунок 3.2. – Система контролерів вебзастосунку

Архітектурна логіка публічної частини добре простежується на прикладі головної сторінки, де контролер не працює безпосередньо з представленням окремих сутностей, а формує спеціальну модель представлення. У наведеному нижче фрагменті коду HomeController отримує дані через відповідні репозиторії та передає їх до HomePageViewModel без змішування з логікою інтерфейсу.

```
public IActionResult Index()
{
  HomePageViewModel homePageViewModel = new HomePageViewModel();
  homePageViewModel.Services = _servicesRepository.GetServices();
  homePageViewModel.Employees = _employeeRepository.GetEmployees();
  homePageViewModel.LatestPosts = _postsRepository.GetLatestPosts(3);

  return View(homePageViewModel);
}
```

Цей фрагмент підтверджує, що взаємодія між компонентами системи здійснюється послідовно: контролер звертається до модулів доступу до даних, далі формує модель представлення і лише після цього передає її до Razor-

представлення. Завдяки цьому рівень відображення не перевантажується логікою обробки даних, а контролер не містить безпосередньої розмітки інтерфейсу.

Важливим елементом архітектури є використання ViewModel. У системі вони застосовуються для підготовки агрегованих даних, необхідних для конкретних сторінок. Зокрема, HomePageViewModel об’єднує списки працівників, послуг, останніх публікацій і вкладену модель запису на прийом, AboutPageViewModel містить дані про працівників, DashboardViewModel використовується для групування термінових, поточних і майбутніх записів у панелі адміністратора, а CommentsAdminViewModel дозволяє розподілити коментарі за станами модерації. Наявність також BlogViewModel, ErrorViewModel і SiteInfoViewModel свідчить про те, що у проєкті системно застосовано окремий рівень моделей представлення для різних інтерфейсних сценаріїв.

Цю ідею ілюструє структура HomePageViewModel, де в межах одного класу поєднано дані кількох різних сутностей і окрему модель для форми запису.

```
        public class HomePageViewModel
{
   public List<Employee> Employees { get; set; }
   public List<Service> Services { get; set; }
   public List<Post> LatestPosts { get; set; }

    public AppointmentViewModel Appointment { get; set; } = new
AppointmentViewModel();
}
```

Наведений код демонструє, що ViewModel у застосунку не дублюють повністю структуру бази даних, а формуються відповідно до потреб конкретного представлення. Такий підхід є важливим для архітектури MVC, оскільки забезпечує адаптацію даних саме до інтерфейсного сценарію, а не до внутрішньої структури сховища.

Окремий рівень у взаємодії компонентів утворюють репозиторії. Вони

інкапсулюють доступ до бази даних і фактично виступають проміжною ланкою між контролерами та контекстом ApplicationDbContext. У проєкті реалізовано окремі класи для роботи з працівниками, категоріями, послугами, публікаціями, коментарями, тегами, навігацією, параметрами сайту, підписниками, користувачами та записами на прийом. Такий підхід відповідає патерну Repository і дозволяє уникнути перенесення всієї логіки доступу до даних у контролери. Наприклад, EmployeeRepository виконує отримання списку працівників, пошук за ідентифікатором, збереження, оновлення й видалення; ServicesRepository відповідає за впорядковане отримання та редагування послуг; PostRepository працює з дописами, категоріями та тегами; AppointmentRepository забезпечує збереження заявок і зміну їх статусу.

Архітектурна взаємодія між контролером і репозиторієм добре простежується на прикладі обробки форми запису на прийом. Спочатку контролер приймає AppointmentViewModel, далі виконує перевірку стану моделі, після чого створює об’єкт Appointment і передає його до репозиторію для збереження. Отже, на цьому сценарії чітко видно три послідовні рівні: інтерфейсна модель, сутність предметної області та модуль доступу до даних.

Важливу роль у внутрішній архітектурі виконує контекст даних ApplicationDbContext, який поєднує сутності прикладної частини із механізмами ідентифікації. Саме через нього в системі централізовано організовано роботу з таблицями працівників, послуг, категорій, коментарів, публікацій, тегів, записів на прийом і користувачів. Наявність успадкування від IdentityDbContext<IdentityUser> означає, що архітектура застосунку поєднує доменні сутності з підсистемою автентифікації та доступу. Додатково в контексті задаються індекси та обмеження для ключових полів, що робить його не лише технічним інструментом доступу до бази даних, а й важливим компонентом формалізації моделі системи.

Значущим елементом архітектури є також використання DTO. У проєкті вони застосовуються переважно в адміністративних сценаріях для створення та

редагування об’єктів. Зокрема, використано CategoryCreateDto, CategoryEditDto, EmployeeCreateDto, EmployeeEditDto, ServiceCreateDto, ServiceEditDto, PostCreateDto, PostEditDto, NavigateCreateDto, NavigateEditDto, а також GuestMessageDTO, JsonResponse і StatusResponce. Це свідчить про те, що в системі чітко розмежовано сутності бази даних і моделі передавання даних у формах та запитах, що є важливим для безпечної та контрольованої обробки введених значень.

Окремої уваги заслуговує архітектура адміністративної частини. Вона реалізована через AdminController, який позначений атрибутом авторизації, отже доступ до нього обмежений для неавтентифікованих користувачів. Усередині цього контролера координується робота з репозиторіями параметрів сайту, тегів, категорій, працівників, послуг, коментарів, навігації, дописів і записів на прийом. Панель керування використовує DashboardViewModel, де записи групуються за часовими характеристиками на термінові, поточні й майбутні. Таким чином, адміністративний модуль можна розглядати як окрему підсистему всередині загальної архітектури вебзастосунку.

Архітектурну побудову застосунку доцільно узагальнити за допомогою діаграми класів.

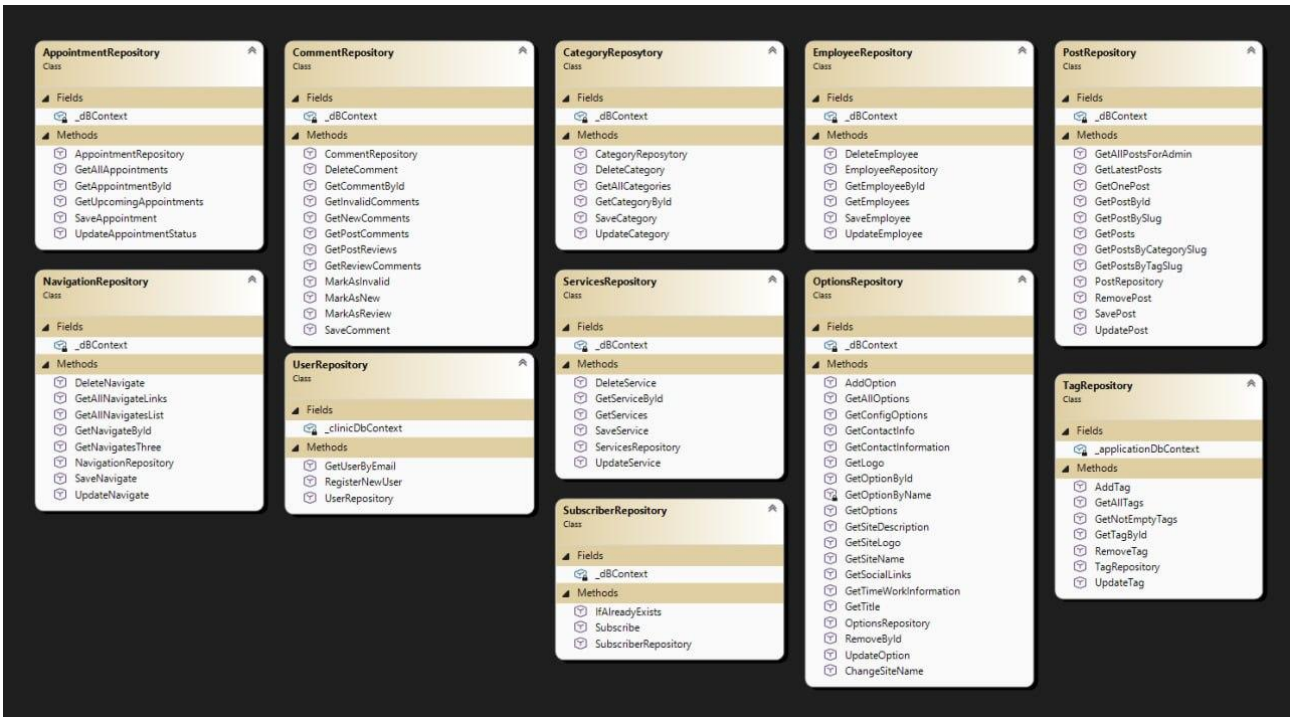


Рисунок 3.3. – Діаграма класів вебзастосунку

З позиції взаємодії компонентів програмне рішення функціонує за такою логікою. Користувач звертається до певного представлення через браузер. Запит надходить до контролера, який визначає необхідний сценарій обробки. Далі контролер звертається до одного або кількох репозиторіїв, які працюють із сутностями через ApplicationDbContext. Отримані або змінені дані перетворюються на ViewModel або DTO відповідно до поточного сценарію, після чого передаються до Razor-представлення або використовуються для виконання операції збереження. У випадку адміністративної частини цей ланцюг доповнюється перевіркою автентифікації користувача. Така послідовність свідчить про впорядковану архітектурну модель, де кожен рівень виконує власну функцію і не дублює відповідальність інших компонентів.

Отже, архітектура розробленого вебзастосунку ґрунтується на поєднанні підходу MVC, патерну Repository, використання сутностей бази даних, моделей представлення та DTO. Система контролерів забезпечує обробку запитів і координацію функціональних сценаріїв, репозиторії відповідають за доступ до даних, ApplicationDbContext формалізує взаємодію з базою даних, а ViewModel

і DTO забезпечують адаптацію даних до потреб інтерфейсу та форм введення. Така архітектурна організація відповідає вимогам до сучасного вебзастосунку сервісного типу, забезпечує підтримуваність коду і створює основу для подальшого аналізу практичної реалізації серверної частини системи.

РОЗДІЛ 4. ПРАКТИЧНА ЧАСТИНА

4.1. Розробка структури бази даних та основних сутностей системи

Розробка структури бази даних є одним із ключових етапів створення вебзастосунку ветеринарної клініки, оскільки саме база даних забезпечує централізоване зберігання інформації, необхідної для функціонування як публічної, так і адміністративної частин системи. У межах розробленого проєкту база даних використовується для збереження відомостей про працівників клініки, послуги, записи на прийом, публікації, категорії, коментарі, теги, навігаційні елементи, параметри сайту, підписників і користувачів. Така структура свідчить про те, що застосунок реалізовано не як простий інформаційний сайт, а як повноцінну прикладну систему, орієнтовану на підтримку кількох взаємопов’язаних функціональних процесів.

Основою роботи з даними в проєкті є контекст ApplicationDbContext, через який задається перелік основних сутностей і забезпечується їх відображення на таблиці бази даних. Саме в ньому визначено набір DbSet, що відображає загальну логіку побудови структури сховища даних.

```
public class ApplicationDbContext : IdentityDbContext<IdentityUser>
{
   public DbSet<Employee> Employees { get; set; }
   public DbSet<Service> Services { get; set; }
   public DbSet<Option> Options { get; set; }
   public DbSet<Navigate> Navigations { get; set; }
   public DbSet<Category> Categories { get; set; }
   public DbSet<Comment> Comments { get; set; }
   public DbSet<Subscribe> Subscribes { get; set; }
   public DbSet<Post> Posts { get; set; }
   public DbSet<Tag> Tags { get; set; }
   public DbSet<PostTags> PostTags { get; set; }
   public DbSet<Appointment> Appointments { get; set; }
   public DbSet<User> Users { get; set; }
```

Наведений фрагмент підтверджує, що структура бази даних охоплює як

інформаційні сутності, так і службові об’єкти системи. До першої групи належать працівники, послуги, публікації, категорії та коментарі, які безпосередньо впливають на наповнення публічної частини сайту. До другої групи можна віднести записи на прийом, навігаційні елементи, параметри конфігурації, підписників і користувачів, що забезпечують сервісне та адміністративне функціонування застосунку. Така організація є виправданою, оскільки вона дозволяє централізовано підтримувати як представницьку, так і прикладну логіку вебзастосунку.

Рисунок 4.1. – Діаграма класів сутностей бази даних]

Однією з центральних сутностей системи є запис на прийом. Саме вона забезпечує реалізацію сервісної взаємодії між користувачем і клінікою. У класі Appointment передбачено поля для збереження імені та прізвища клієнта, електронної пошти, номера телефону, повідомлення, дати прийому, дати створення запису та поточного статусу.

```
public class Appointment
{
```

```
    public int Id { get; set; }
    [Required]
    [MaxLength(100)]
    public string FirstName { get; set; } = null!;
    [Required]
    [MaxLength(100)]
    public string LastName { get; set; } = null!;
    [Required]
    [MaxLength(150)]
    public string Email { get; set; } = null!;
    [Required]
    [MaxLength(30)]
    public string Phone { get; set; } = null!;
    [Required]
    [MaxLength(2000)]
    public string Message { get; set; } = null!;
    [Required]
    public DateTime AppointmentDate { get; set; }
    public DateTime CreatedAt { get; set; } = DateTime.UtcNow;
    public AppointmentStatus Status { get; set; } = AppointmentStatus.New;
}
```

Структура цієї сутності свідчить про те, що система не лише приймає заявки, а й забезпечує їх подальше адміністрування. Особливу роль відіграє поле Status, яке використовує перелік AppointmentStatus і дозволяє фіксувати етап опрацювання звернення. У проєкті передбачено стани New, CalledBack, Confirmed, Completed, Cancelled, що робить підсистему запису на прийом більш впорядкованою та придатною до реального практичного використання.

Сутність Employee призначена для збереження інформації про працівників клініки. Вона містить ідентифікатор, шлях до зображення, альтернативний текст, прізвище та ім’я працівника, спеціалізацію і текстовий опис.

```
    public class Employee
{
    public int Id { get; set; }
    public string ImgSrc { get; set; } = String.Empty;
    public string ImgAlt { get; set; } = String.Empty;
    [Required(ErrorMessage = "Fio is required")]
    public string Fio { get; set; } = string.Empty;
    [Required(ErrorMessage = "Speciality is required")]
```

```
    public string Speciality { get; set; } = string.Empty;
    public string Context { get; set; } = String.Empty;
}
```

Наявність окремої сутності для працівників є важливою з погляду функціональної повноти системи, оскільки користувач повинен мати змогу переглядати відомості про спеціалістів клініки. Крім того, така структура забезпечує зручне адміністрування цих даних у межах службової частини вебзастосунку.

Сутність Service використовується для збереження інформації про послуги, які надає клініка. Вона містить назву, опис, графічні атрибути та поле Order, що дозволяє впорядковувати перелік послуг при відображенні.

```
        public class Service
{
    public int Id { get; set; }
    [Required(ErrorMessage = "Title is required")]
    public string Title { get; set; } = string.Empty;
    public string Description { get; set; } = String.Empty;
    public string ImgSrc { get; set; } = String.Empty;
    public string ImgAlt { get; set; } = String.Empty;
    public int Order { get; set; }
}
```

Сутність Service виконує роль одного з основних інформаційних об’єктів системи. Вона відображає перелік функціональних можливостей клініки та є важливою частиною як головної сторінки, так і окремих інформаційних блоків сайту. Поле порядку відображення свідчить про те, що розробники передбачили можливість адміністрування порядку подання цих даних без зміни програмної логіки.

Важливе місце у структурі бази даних займає блок, пов’язаний із контентом. Центральною сутністю тут є Post, яка призначена для збереження публікацій. Вона включає назву, короткий слоган, графічні поля, URL-ідентифікатор, текстовий вміст, дату публікації, ознаку оприлюднення та зовнішній ключ на категорію.

```
        public class Post
{    public int Id { get; set; }
```

```
    [Required(ErrorMessage = "Title is required")]
    public string Title { get; set; } = string.Empty;
    public string Slogan { get; set; } = String.Empty;
    public string ImgSrc { get; set; } = String.Empty;
    public string ImgAlt { get; set; } = String.Empty;
    [Required(ErrorMessage = "UrlSlug is required")]
    public string UrlSlug { get; set; } = string.Empty;
    [ForeignKey("Category")]
    public int CategoryId { get; set; }
    public string Content { get; set; } = String.Empty;
    [DefaultValue(false)]
    public bool PostedOn { get; set; }
    public DateTime DateOfPublished { get; set; }
    public Category Category { get; set; }
    public ICollection<Comment> Comments { get; set; } = new List<Comment>();
    public ICollection<PostTags> PostTags { get; set; } = new List<PostTags>();
}
```

Ця сутність є особливо показовою, оскільки демонструє наявність зв’язків між кількома таблицями. Публікація пов’язана з категорією через поле CategoryId, з коментарями через колекцію Comments, а з тегами через проміжну сутність PostTags. Отже, контентна частина сайту має не лінійну, а реляційну структуру, що дозволяє реалізувати більш гнучку модель організації матеріалів.

Для групування публікацій у системі використовується сутність Category. Вона містить назву категорії, унікальний символьний ідентифікатор Slug, короткий опис, графічні атрибути та колекцію публікацій, пов’язаних із цією категорією.

```
        public class Category
{
    public int Id { get; set; }
    [Required(ErrorMessage = "Title is required")]
    public string Title { get; set; } = String.Empty;
    [Required(ErrorMessage = "Slug is required")]
    public string Slug { get; set; } = String.Empty;
    public string Slogan { get; set; } = String.Empty;
    public string Description { get; set; } = String.Empty;
    public string ImgSrc { get; set; } = String.Empty;
    public string ImgAlt { get; set; } = String.Empty;
    public int Order { get; set; }
    public ICollection<Post> Posts { get; set; } = new List<Post>();
```

```
}
```

Наявність колекції Posts підтверджує зв’язок типу один-до-багатьох між категорією та публікаціями. Така організація структури даних є важливою для реалізації блогу або інформаційного розділу сайту, де матеріали повинні бути згруповані за тематичними напрямами.

Коментарі представлені окремою сутністю Comment, яка містить посилання на публікацію, текст повідомлення, дату створення, службові ознаки модерації, посилання на батьківський коментар та дані відвідувача.

```
        public class Comment
{
   public int Id { get; set; }
   public int PostId { get; set; }
   public string Message { get; set; } = String.Empty;
   public DateTime DateCreated{ get; set; }
   [DefaultValue(false)]
   public bool IsValid { get; set; }
   [DefaultValue(false)]
   public bool IsReview { get; set; }
   public int? Parent_Id { get; set; }
     public string VisitorName { get; set; } = String.Empty;
   public string VisitorEmail { get; set; } = String.Empty;
   public string VisitorAvatar { get; set; } = String.Empty;
   public Post Post { get; set; }
}
```

Ця сутність засвідчує, що в системі передбачено не лише виведення контенту, а й підтримку зворотного зв’язку з боку користувачів. Поле Parent_Id дозволяє реалізувати вкладені коментарі, а поля IsValid та IsReview — керувати станом модерації. Таким чином, структура бази даних підтримує не лише статичні інформаційні об’єкти, а й елементи інтерактивної взаємодії.

Для реалізації тематичного маркування публікацій використовується сутність Tag та проміжна таблиця PostTags. Саме вона забезпечує зв’язок багато-до-багатьох між публікаціями та тегами.

```
        public class Tag
{
   public int Id { get; set; }
   [Required(ErrorMessage = "Name is required")]
```

```
    public string Name { get; set; } = string.Empty;
    [Required(ErrorMessage = "Slug is required")]
    public string Slug { get; set; } = string.Empty;
}
        public class PostTags
{
    public int Id { get; set; }
    [ForeignKey("PostId")]
    public int PostId { get; set; }
    public virtual Post Post { get; set; }
    [ForeignKey("TagId")]
    public int TagId { get; set; }
    public virtual Tag Tag { get; set; }
}
```

Така побудова є типовою для реляційних систем, у яких одна публікація може мати кілька тегів, а один тег — належати до кількох публікацій. Це підвищує гнучкість контентної моделі та дозволяє організовувати додаткову класифікацію матеріалів.

Крім основних інформаційних сутностей, у системі передбачено і службові об’єкти. Так, Navigate використовується для побудови структури меню, Option - для збереження параметрів конфігурації сайту, Subscribe - для обліку підписників, а User - для локального збереження даних користувачів адміністративної частини.

```
        public class Navigate
{
    public int Id { get; set; }
    public string Title { get; set; } = String.Empty;
    public string Href { get; set; } = String.Empty;
    public int Order { get; set; }
    public int? Parent_Id { get; set; }
    public ICollection<Navigate> Childs { get; set; } = new List<Navigate>();
}
        public class Option
{
    public int Id { get; set; }
    [Required(ErrorMessage = "Name is required")]
    public string Name { get; set; } = string.Empty;
    public string? Key { get; set; }
    public string? Value { get; set; }
```

```
    public string? Relation { get; set; }
    public int Order { get; set; }
    public bool IsSystem { get; set; }
}
        public class Subscribe
{
    public int Id { get; set; }
    public string Name { get; set; } = String.Empty;
    public string Email { get; set; } = String.Empty;
}
        public class User
{
    public int Id { get; set; }
    public string Login { get; set; } = string.Empty;
    public string Email { get; set; } = string.Empty;
    public string PasswordHash { get; set; } = string.Empty;
    public DateTime DateOfCreated { get; set; } = DateTime.Now;
}
```

Наявність цих сутностей підтверджує, що база даних розроблялася з урахуванням не лише основної предметної області, а й завдань супроводу сайту. Зокрема, Navigate дозволяє підтримувати багаторівневу навігацію, Option — зберігати змінні параметри конфігурації без внесення змін у код, Subscribe — підтримувати взаємодію з користувачами, а User — забезпечувати основу для службового доступу до адміністративної частини.

Важливою особливістю розробленої структури є наявність обмежень і унікальних індексів, які задаються на рівні контексту даних. Це стосується, зокрема, унікальності Slug у категорій, Name у параметрів, UrlSlug у публікацій, Slug у тегів та Email у підписників.

```
        protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    base.OnModelCreating(modelBuilder);
    modelBuilder.Entity<Category>()
        .HasIndex(c => c.Slug)
        .IsUnique();
    modelBuilder.Entity<Option>()
        .HasIndex(o => o.Name)
        .IsUnique();
    modelBuilder.Entity<Post>()
        .HasIndex(p => p.UrlSlug)
```

```
    .IsUnique();
modelBuilder.Entity<Tag>()
    .HasIndex(t => t.Slug)
    .IsUnique();
modelBuilder.Entity<Subscribe>()
    .HasIndex(o => o.Email)
    .IsUnique();
```

Такі обмеження мають важливе значення з точки зору цілісності даних. Вони запобігають дублюванню ключових символьних ідентифікаторів, забезпечують однозначність посилань і спрощують подальшу обробку інформації на рівні програмної логіки.

Отже, структура бази даних розробленого вебзастосунку є цілісною та функціонально обґрунтованою. Вона охоплює як основні сутності предметної області, пов’язані з діяльністю ветеринарної клініки, так і службові компоненти, необхідні для підтримки адміністративного функціоналу, навігації та керування контентом. Використання окремих сутностей для записів на прийом, працівників, послуг, публікацій, категорій, коментарів, тегів, параметрів і користувачів забезпечує впорядкованість системи даних та створює надійну основу для функціонування всього вебзастосунку. Саме така побудова бази даних дозволяє реалізувати необхідні інформаційні, сервісні та адміністративні можливості в межах єдиного програмного рішення.

4.2. Реалізація серверної частини вебзастосунку

Реалізація серверної частини вебзастосунку ветеринарної клініки є одним із центральних етапів практичного створення системи, оскільки саме на цьому рівні забезпечуються обробка запитів користувача, взаємодія з базою даних, формування моделей для представлень, перевірка введених значень, керування адміністративними операціями та підтримка основної бізнес-логіки. У розробленому проєкті серверна частина побудована на основі ASP.NET Core MVC, що дозволило організувати чіткий розподіл функцій між контролерами,

моделями, DTO, ViewModel та репозиторіями. Такий підхід забезпечує не лише коректне функціонування застосунку, а й підтримуваність коду, зручність розширення системи та впорядковану взаємодію між її компонентами.

Загальна логіка серверної частини полягає в тому, що HTTP-запит надходить до відповідного контролера, де визначається сценарій його обробки. Далі контролер звертається до репозиторіїв, які виконують операції з базою даних через контекст ApplicationDbContext, після чого одержані результати перетворюються на моделі представлення або передаються безпосередньо до представлень. У випадку форм введення даних додатково застосовуються DTO або ViewModel, які використовуються для приймання, перевірки та подальшого перетворення даних у сутності предметної області. Таким чином, серверна частина виступає як проміжний рівень між інтерфейсом користувача та сховищем даних, забезпечуючи цілісну логіку роботи всієї системи.

У розробленому застосунку систему контролерів можна поділити на дві основні групи. Перша група охоплює контролери публічної частини, які відповідають за відображення сторінок сайту та взаємодію із відвідувачем. До них належать HomeController, AboutController, PostController, CategoryController, OptionController, AjaxController і ErrorController. Друга група представлена контролерами службової частини, серед яких основними є AdminController та AccountController. Перший забезпечує керування вмістом і довідковими сутностями, а другий відповідає за вхід користувачів до адміністративної частини системи. Такий розподіл є логічним, оскільки дозволяє відокремити публічні сценарії перегляду інформації від службових сценаріїв адміністрування та автентифікації.

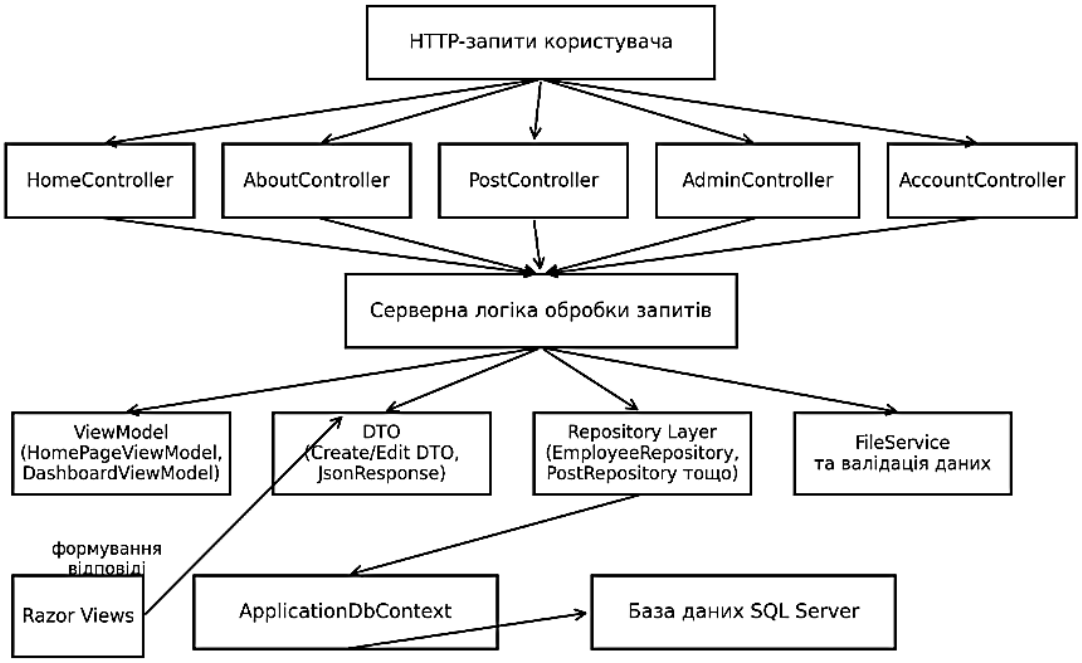


Рисунок 4.2. – Структура серверної частини вебзастосунку

Одним із найбільш показових прикладів реалізації серверної логіки є HomeController, який одночасно виконує функції формування головної сторінки та обробки запису на прийом. При завантаженні головної сторінки контролер звертається до репозиторіїв послуг, працівників і публікацій, формує HomePageViewModel та передає її до представлення. Завдяки цьому на одній сторінці відображаються одразу кілька інформаційних блоків, що об’єднуються єдиною моделлю представлення. Водночас саме цей контролер обробляє надсилання форми запису на прийом, перевіряє коректність введених даних та у разі успішної валідації створює новий об’єкт Appointment, який передається до репозиторію для збереження. Така реалізація свідчить про поєднання інформаційної та сервісної логіки в межах одного користувацького сценарію.

У цьому контексті важливу роль відіграє AppointmentViewModel, який використовується як проміжна модель введення даних. Він містить лише ті поля, які необхідні для заповнення форми, а також атрибути валідації. Саме через нього реалізується перевірка обов’язковості введення імені, прізвища, електронної пошти, номера телефону, дати прийому та повідомлення. Такий

підхід дозволяє не працювати безпосередньо із сутністю бази даних на рівні форми, а застосовувати спеціалізовану модель для приймання даних від користувача. Це підвищує безпечність і структурованість серверної логіки, оскільки відокремлює модель введення від моделі збереження.

Крім AppointmentViewModel, у системі використовуються й інші моделі представлення, що обслуговують різні інтерфейсні сценарії. Наприклад, HomePageViewModel об’єднує працівників, послуги, публікації та форму запису, AboutPageViewModel передає дані про працівників для сторінки про клініку, DashboardViewModel використовується для панелі адміністратора, а CommentsAdminViewModel дозволяє групувати коментарі за станом модерації. Таким чином, ViewModel у серверній частині не дублюють сутності бази даних, а формуються відповідно до потреб конкретної сторінки або функціонального сценарію.

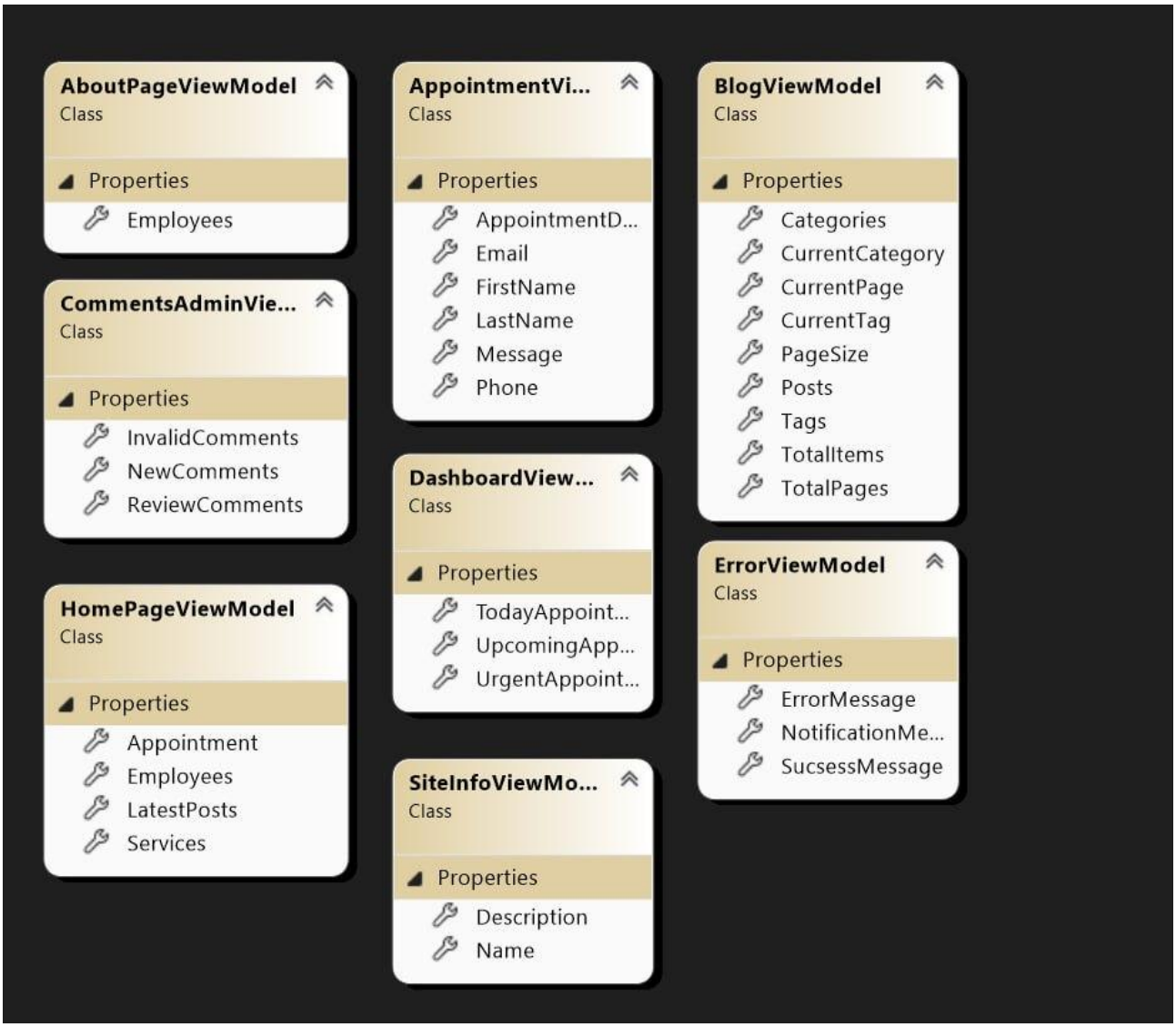


Рисунок 4.3. – Використання ViewModel у серверній частині

Окремого розгляду потребує застосування DTO. У проєкті вони використовуються насамперед у тих сценаріях, де необхідно створювати або редагувати сутності через адміністративні форми. Зокрема, наявні CategoryCreateDto, CategoryEditDto, EmployeeCreateDto, EmployeeEditDto, ServiceCreateDto, ServiceEditDto, PostCreateDto, PostEditDto, NavigateCreateDto, NavigateEditDto, а також GuestMessageDTO. Їх використання дозволяє обмежити набір полів, які надходять до серверної частини під час виконання певної операції. Це означає, що при створенні чи редагуванні об’єкта система приймає не повну сутність, а лише необхідний для поточного сценарію набір значень. Такий підхід є обґрунтованим, оскільки дозволяє зробити логіку

серверної обробки більш контрольованою та відокремити сутності бази даних від моделей передавання даних.

Особливо показовим є використання DTO при роботі з категоріями, працівниками, послугами та публікаціями в адміністративному модулі. Контролер приймає DTO з форми, виконує перевірку, у разі потреби обробляє завантажене зображення, а далі або створює нову сутність, або оновлює існуючу. Таким чином, DTO у серверній частині виступають як засіб формалізованого приймання значень, який дозволяє зменшити зв’язність між вебформами та моделями бази даних.

Рисунок 4.4. – Діаграма DTO серверної частини

Суттєве місце в реалізації серверної частини займає AdminController, який виконує роль центрального координатора адміністративного функціоналу. Через нього реалізовано керування параметрами сайту, тегами, категоріями, працівниками, послугами, коментарями, навігаційними елементами, публікаціями та записами на прийом. Завдяки такій концентрації службових сценаріїв в одному контролері вдалося забезпечити централізовану організацію керування основними об’єктами системи. Водночас логіка безпосередньої

роботи з даними не перенесена до самого контролера, а делегується репозиторіям, що відповідає принципу розмежування відповідальності.

Характерною ознакою серверної частини є використання репозиторіїв як окремого рівня взаємодії з базою даних. У проєкті реалізовано EmployeeRepository, ServicesRepository, CategoryReposytory, PostRepository, AppointmentRepository, CommentRepository, TagRepository, NavigationRepository, OptionsRepository, SubscriberRepository та інші класи доступу до даних. Саме вони інкапсулюють типові операції створення, читання, оновлення й видалення записів. Така організація дає змогу зробити контролери компактнішими, а логіку доступу до даних більш централізованою. Крім того, це спрощує подальший супровід системи, оскільки зміни у способі роботи з певною сутністю можуть бути локалізовані в межах одного репозиторію.

Серверна частина також забезпечує механізми авторизації та доступу до адміністративного модуля. Для цього у проєкті використано AccountController, де реалізовано логіку входу користувача до системи, перевірку облікових даних та створення набору claims для cookie-автентифікації. При реєстрації нового користувача пароль не зберігається у відкритому вигляді, а перетворюється на хеш, що підвищує рівень безпечності роботи адміністративної частини. Сам модуль адміністрування позначено атрибутом авторизації, тому доступ до нього можливий лише після успішного входу. Такий підхід відповідає базовим вимогам до захисту службової частини вебзастосунку.

Окремим напрямом реалізації серверної логіки є робота із файлами та зображеннями. У випадках створення або редагування категорій, працівників і послуг серверна частина не лише приймає текстові значення, а й обробляє завантажені графічні файли. Для цього використовується FileService, який забезпечує збереження файлів у відповідних директоріях, а також механізм перевірки коректності зображень. Це дозволяє централізувати файлову логіку та уникнути дублювання однакових дій у різних методах контролерів. Такий підхід є важливим для практичної реалізації системи, оскільки значна частина

інформаційних об’єктів у застосунку супроводжується графічними матеріалами.

Особливу увагу в серверній частині приділено записам на прийом. Після надсилання користувачем форми створюється новий об’єкт запису, якому автоматично присвоюється дата створення та початковий статус. Надалі ці записи відображаються в адміністративному модулі, де адміністратор має можливість змінювати їх поточний стан. Для цього використовується перелік AppointmentStatus, який упорядковує логіку опрацювання звернень і дозволяє розрізняти нові, підтверджені, завершені або скасовані записи. Отже, серверна частина не лише приймає звернення від користувача, а й підтримує повний цикл їх подальшого адміністрування.

Ще одним важливим аспектом є підтримка асинхронної або частково динамічної взаємодії. У проєкті для цього використовується AjaxController, а також DTO типу JsonResponse і StatusResponce, що дозволяють повертати структуровані відповіді для окремих клієнтських сценаріїв. Хоча основа застосунку є багатосторінковою і побудованою на серверному рендерингу, використання таких допоміжних елементів розширює можливості системи та забезпечує більш гнучку обробку окремих запитів.

Таким чином, серверна частина розробленого вебзастосунку реалізована як впорядкована багаторівнева система, у якій кожен компонент виконує чітко визначену функцію. Контролери обробляють запити користувачів і координують функціональні сценарії, репозиторії забезпечують доступ до даних, ApplicationDbContext формалізує взаємодію з базою даних, ViewModel адаптують інформацію до потреб представлень, а DTO використовуються для контрольованого приймання і передавання даних у межах адміністративних форм та окремих сервісних сценаріїв. Така реалізація повністю відповідає архітектурі MVC, забезпечує підтримуваність системи та створює надійну основу для функціонування як публічної, так і адміністративної частин вебзастосунку.

4.3. Розробка користувацького інтерфейсу та основних функціональних можливостей

Розробка користувацького інтерфейсу вебзастосунку ветеринарної клініки виконувалася з урахуванням необхідності поєднання інформаційної насиченості, зручної навігації, адаптивності та підтримки основних сервісних сценаріїв взаємодії з користувачем. У межах проєкту інтерфейс реалізовано засобами Razor, HTML5, CSS3, JavaScript і Bootstrap, що відповідає загальній архітектурі серверного багатосторінкового вебзастосунку. Такий підхід дозволив сформувати цілісне середовище, у якому публічна частина сайту забезпечує представлення клініки, її послуг, працівників, публікацій і форми запису на прийом, а адміністративна частина надає засоби керування внутрішнім наповненням системи.

Загальна побудова інтерфейсу публічної частини ґрунтується на використанні спільного макета сторінок, який задає однакову структуру для всіх основних розділів сайту. Саме через нього забезпечується відображення верхньої частини сторінки, логотипа, навігаційного меню, підписки на розсилку, інформаційного блоку в нижній частині та підключення спільних стилів і скриптів. Така організація є доцільною, оскільки дозволяє підтримувати єдиний візуальний стиль ресурсу та повторно використовувати однакові елементи без дублювання розмітки в кожному окремому представленні.

У спільному макеті видно, що для оформлення інтерфейсу застосовано бібліотеки стилів, шрифти, Bootstrap, власні CSS-файли, а також набір ViewComponent, через які формуються логотип, назва сайту, верхня панель, навігаційне меню, блоки з контактною інформацією та часом роботи. Це означає, що візуальна структура сайту проєктувалася не як набір статичних сторінок, а як система повторно використовуваних інтерфейсних компонентів.

```
        <header class="header-main">
    <div class="top-header container-fluid no-padding">
        <div class="col-md-4 col-sm-4 col-xs-4 no-padding color-red"></div>
        <div class="col-md-4 col-sm-4 col-xs-4 no-padding color-green"></div>
        <div class="col-md-4 col-sm-4 col-xs-4 no-padding color-yellow"></div>
        <div class="container">
            <div class="row">
                <div class="col-md-8 col-sm-8 col-xs-6">
                    @await Component.InvokeAsync("LeaveMessage")
                </div>
                <div class="social col-md-4 col-sm-4 col-xs-6 pull-right">
                    @await Component.InvokeAsync("TopHeader")
                </div>
            </div>
        </div>
    </div>
    <div class="middle-header container-fluid no-padding">
        <div class="container">
            <div class="row">
                <div class="col-md-5 logo-block" style="display:flex">
                    <a href="/">
                        @await Component.InvokeAsync("Logo")
                    </a>
                    @await Component.InvokeAsync("SiteName")
                </div>
                @await Component.InvokeAsync("TopBar")
            </div>
        </div>
    </div>
    @await Component.InvokeAsync("NavigationMenu");
</header>
```

Наведений фрагмент підтверджує, що інтерфейс побудовано за модульним принципом, де окремі візуальні блоки винесено до компонентів. Такий підхід є важливим для підтримки цілісності дизайну та спрощує супровід інтерфейсу, оскільки зміна спільного елемента не потребує редагування всіх сторінок вручну.

Для наочного подання загального зовнішнього вигляду публічної частини сайту доцільно вставити окремий рисунок.

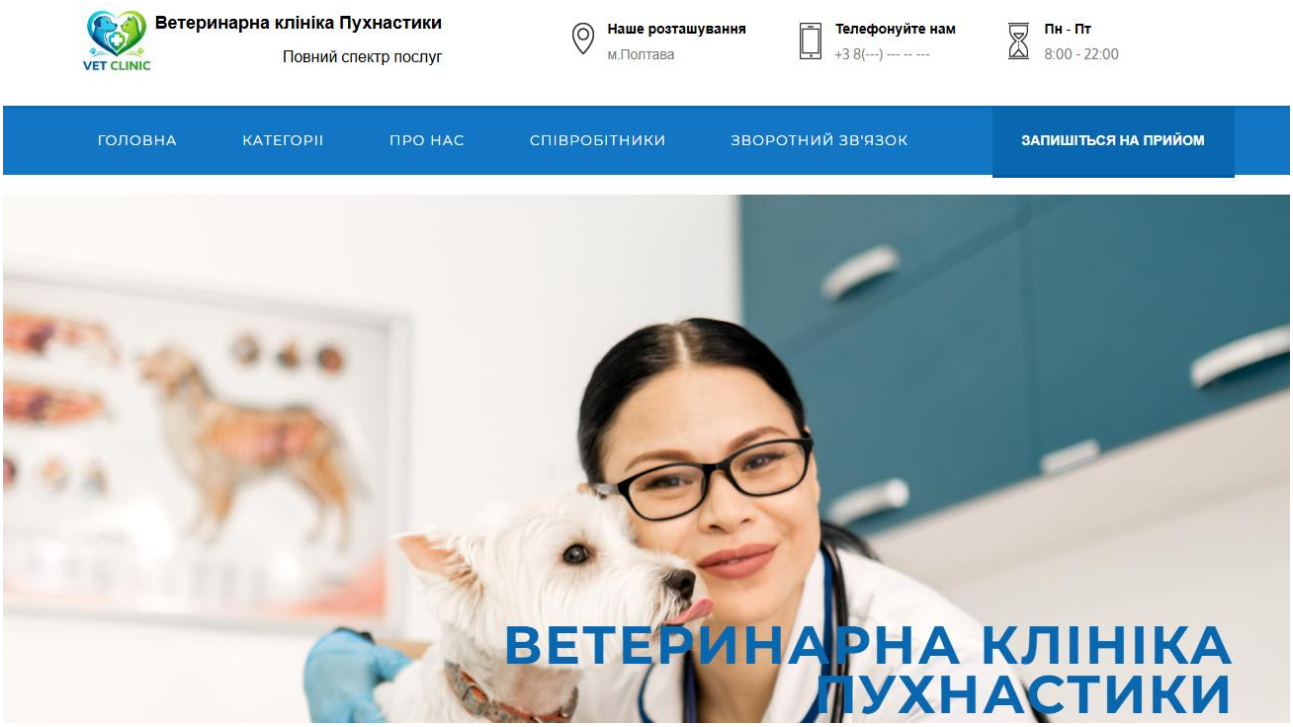


Рисунок 4.5. – Загальний вигляд головної сторінки вебзастосунку

Центральне місце у публічній частині займає головна сторінка, оскільки саме вона поєднує основні інформаційні та сервісні функції системи. Її структура охоплює фотослайдер, вступний інформаційний блок, форму запису на прийом, секцію послуг, блок переваг, відгуки клієнтів, а також інформацію про команду. Така побудова є обґрунтованою, оскільки дозволяє користувачеві вже з першої сторінки отримати загальне уявлення про клініку, ознайомитися з основними послугами та перейти до ключової дії, а саме до запису на прийом.

У коді представлення головної сторінки простежується, що важливим функціональним акцентом є інтеграція форми запису безпосередньо в загальну структуру сторінки. Вона підключається як окреме часткове представлення, що дозволяє відокремити логіку цього елемента від загальної розмітки сторінки.

```
    <div id="appointment" class="appointment container-fluid no-padding">
<div class="section-padding"></div>
<div class="container">
  <div class="row">
    <partial name="_AppointmentForm" model="Model.Appointment" />
  </div>
</div>
```

```
    <div class="section-padding"></div>
</div>
```

Таке рішення є виправданим як з точки зору структури інтерфейсу, так і з точки зору функціональності. Форма запису не ізолюється на окремій сторінці, а розташовується в межах головного інтерфейсного сценарію, що зменшує кількість переходів і спрощує взаємодію користувача із системою.

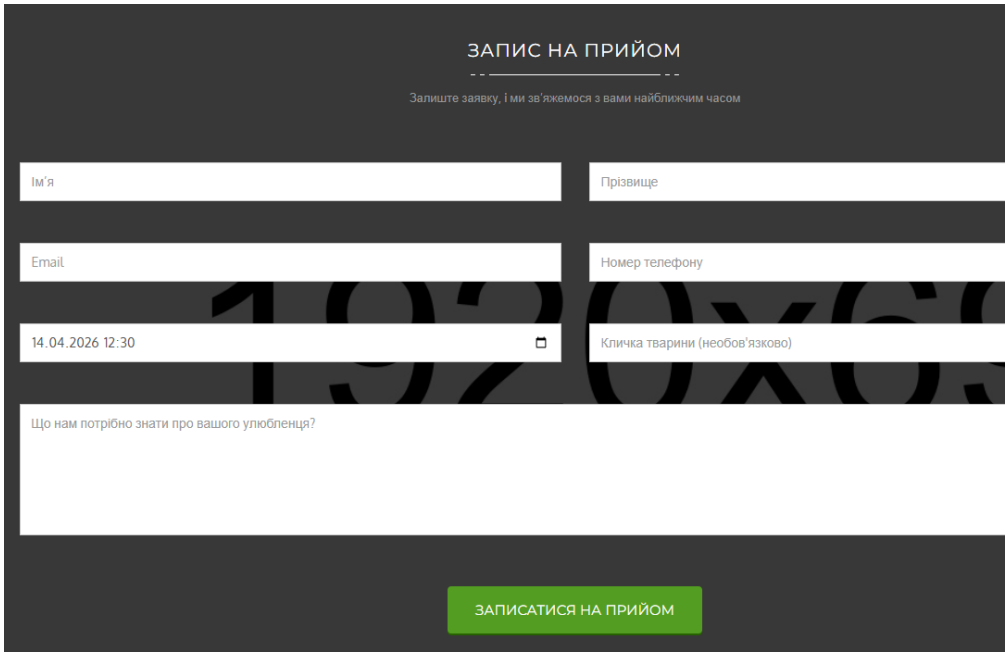


Рисунок 4.6. – Форма запису на прийом на головній сторінці

Однією з найважливіших інформаційних секцій є блок послуг. У межах головної сторінки він реалізований у вигляді вкладок, що дозволяє компактно подати перелік послуг та перемикатися між ними без перевантаження сторінки надмірним обсягом тексту. Кожна послуга подається разом із назвою, іконкою або зображенням та коротким описом. Такий спосіб представлення є вдалим, оскільки він поєднує наочність і структурованість.

```
        <div class="service-tab col-md-9 no-padding">
    <div class="col-md-4 col-sm-5 no-padding">
      <ul class="nav nav-tabs" role="tablist">
        @for (int i = 0; i < postVm.Services.Count; i++)
        {
            var service = postVm.Services[i];
            <li role="presentation" class="@(i == 0 ? "active" : "")">
```

```
            <a data-toggle="tab"
              role="tab"
              aria-controls="tab-@service.Id"
              href="#tab-@service.Id"
              aria-expanded="@(i == 0 ? "true" : "false")">
               <i>
                  <img src="@service.ImgSrc" alt="@service.ImgAlt">
               </i>
               @service.Title
            </a>
          </li>
        }
    </ul>
</div>
```

Наведений фрагмент показує, що функціональна можливість перегляду послуг реалізована динамічно, на основі даних, які передаються з серверної частини. Це означає, що візуальний інтерфейс прямо пов’язаний із наповненням бази даних, а адміністрування послуг автоматично впливає на вигляд відповідної секції сайту.

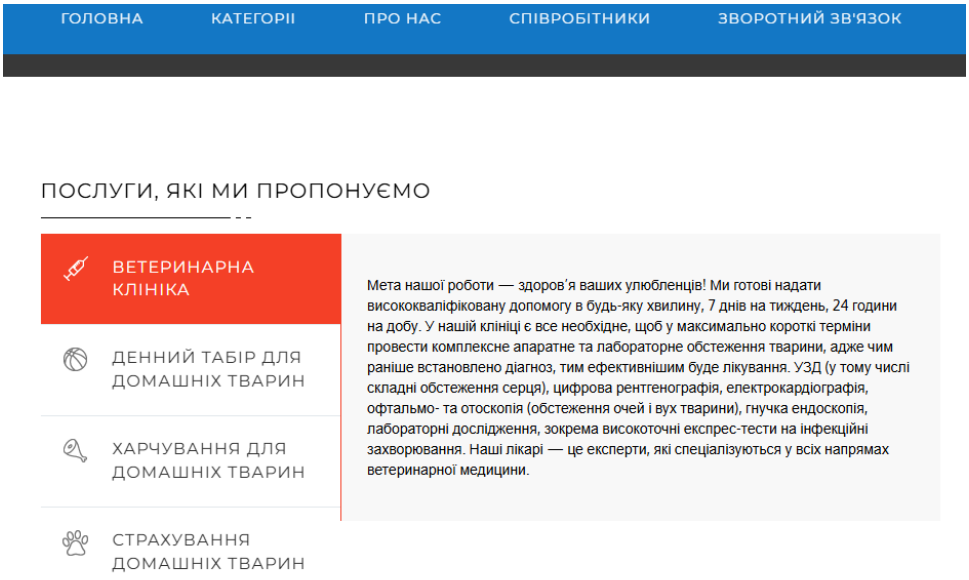


Рисунок 4.7. – Секція послуг вебзастосунку

Крім основних сервісних блоків, у публічній частині реалізовано секції, що підвищують довіру користувача до ресурсу. До них належать блок переваг клініки, відгуки клієнтів, інформація про працівників і підписка на розсилку.

Зокрема, у макеті сторінки передбачено окрему форму підписки з повідомленнями про успішне або неуспішне виконання дії. Це свідчить про наявність додаткових каналів взаємодії з користувачем та розширення функціоналу за межі базового представлення інформації.

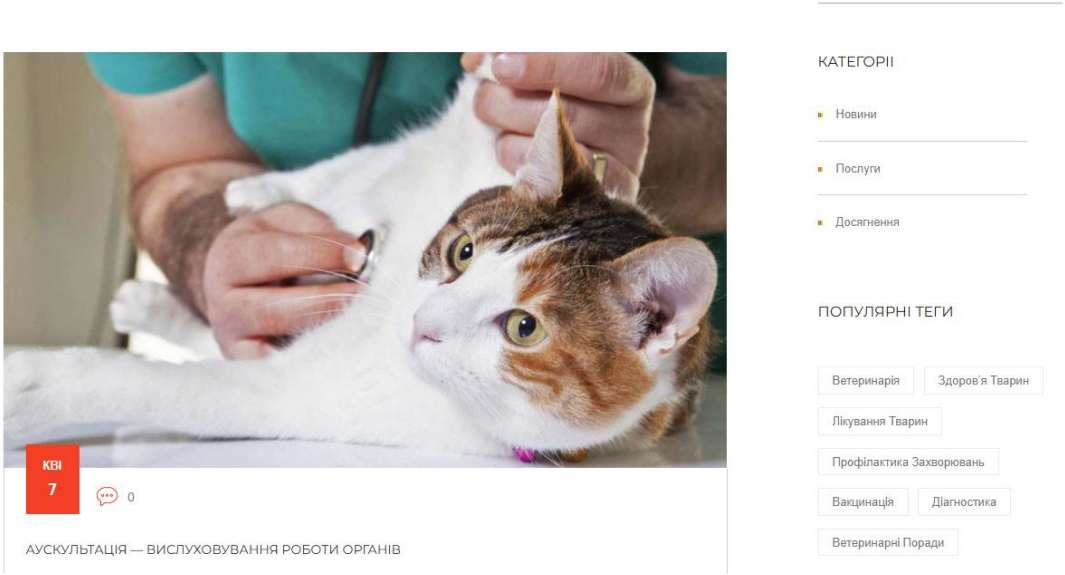


Рисунок 4.8. – Частина функціоналу сайту

Особливу увагу при реалізації інтерфейсу приділено використанню компонентного підходу. Наявність ViewComponent для логотипа, меню, назви сайту, контактної інформації, часу роботи та інших спільних елементів означає, що інтерфейс розроблявся з орієнтацією на повторне використання візуальних блоків. Це дозволяє зберігати узгоджений стиль сторінок, спрощує зміну структури сайту і є важливим з погляду підтримуваності проєкту.

Окремий інтерес становить адміністративна частина вебзастосунку, оскільки саме вона забезпечує реалізацію внутрішнього керування системою. Її інтерфейс має інше функціональне призначення, ніж публічна частина, тому орієнтований передусім на зручність адміністрування. В адміністративній панелі реалізовано сторінки для перегляду та редагування категорій, працівників, послуг, навігаційних елементів, публікацій, тегів, параметрів, коментарів і записів на прийом. Центральна сторінка адміністративного модуля

виконує роль інформаційної панелі та відображає термінові, поточні й найближчі записи на прийом.

```
        @if (Model.UrgentAppointments.Any())
{
   <div class="card mb-5 border border-danger">
      <div class="card-header">
         <h3 class="card-title text-danger">Термінові зустрічі</h3>
      </div>
      <div class="card-body">
         @await Html.PartialAsync("_AppointmentsTable",
Model.UrgentAppointments)
      </div>
   </div>
}

<div class="card mb-5">
   <div class="card-header">
      <h3 class="card-title">Сьогоднішні зустрічі</h3>
   </div>
   <div class="card-body">
      @await Html.PartialAsync("_AppointmentsTable", Model.TodayAppointments)
   </div>
</div>
```

Цей фрагмент демонструє, що інтерфейс адміністративної частини орієнтований на візуальне розмежування записів за ступенем терміновості та часом виконання. Такий підхід є особливо важливим для практичного використання системи, оскільки дозволяє адміністратору швидко оцінити поточний стан звернень і своєчасно реагувати на них.

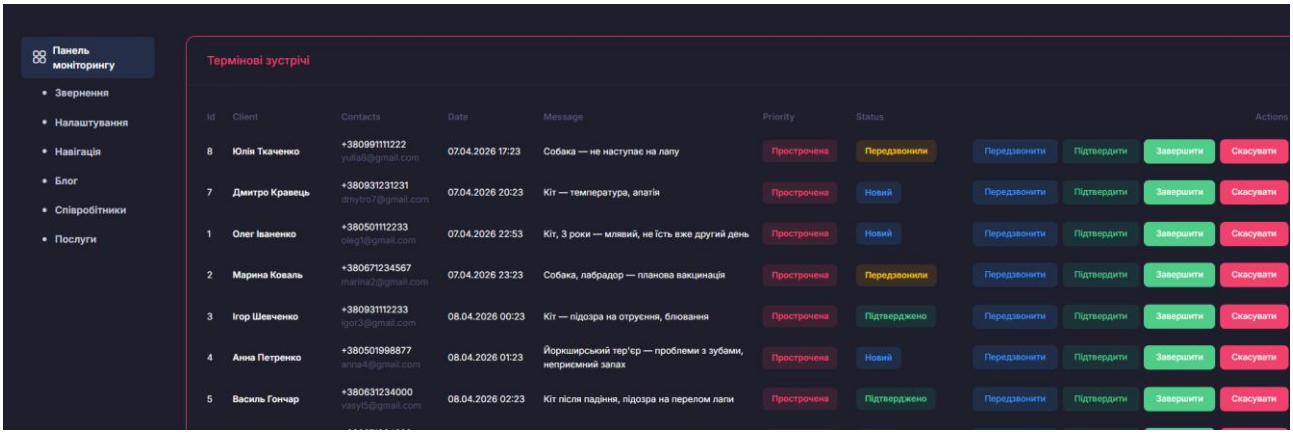


Рисунок 4.9. – Головна сторінка адміністративної панелі

Крім головної панелі, в адміністративній частині реалізовано форми створення та редагування сутностей. Це стосується працівників, послуг, категорій, публікацій, тегів та інших об’єктів. Такі сторінки є важливими з точки зору функціональних можливостей системи, оскільки саме через них забезпечується оновлення вмісту сайту. Наявність форм редагування означає, що вебзастосунок підтримує повний цикл керування даними без потреби у прямому втручанні в базу даних або програмний код.

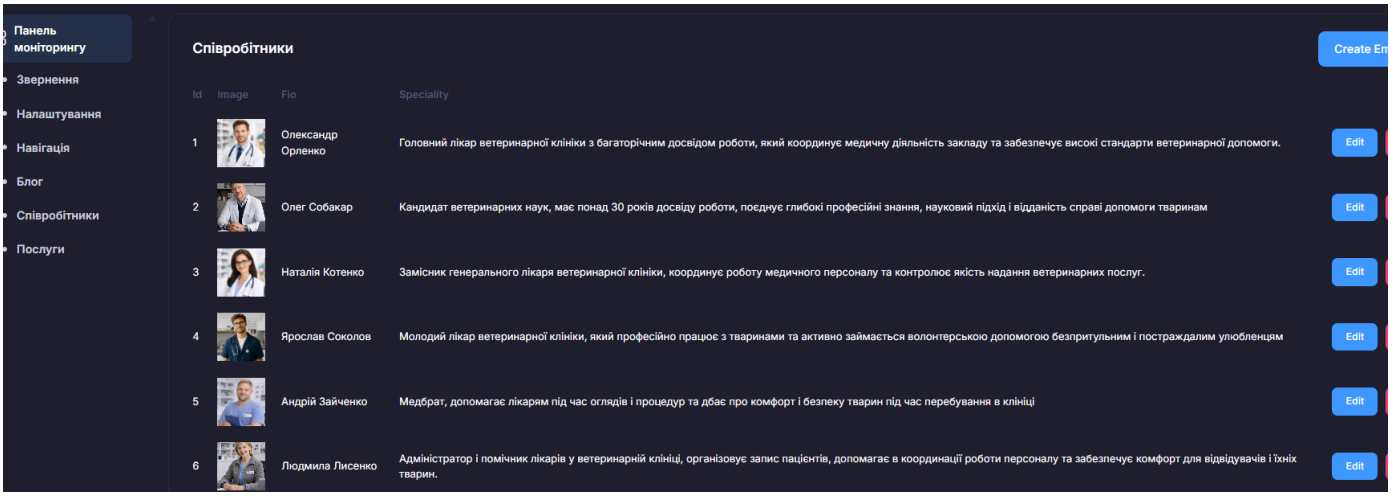


Рисунок 4.10. – Форма створення або редагування сутності в адміністративній частині

Інтерфейс вебзастосунку реалізовано з використанням Bootstrap, що забезпечує адаптивність сторінок, впорядкованість блоків і зручність роботи з формами, таблицями, кнопками та картками. Це особливо помітно як у публічній частині, де використовуються колонки, вкладки, секції та візуальні інформаційні блоки, так і в адміністративній частині, де застосовано табличні представлення, картки та панелі для організації даних. Використання Bootstrap у поєднанні з HTML5, CSS3 і JavaScript дозволило створити інтерфейс, який є достатньо гнучким для різних сценаріїв використання і водночас узгодженим за стилем.

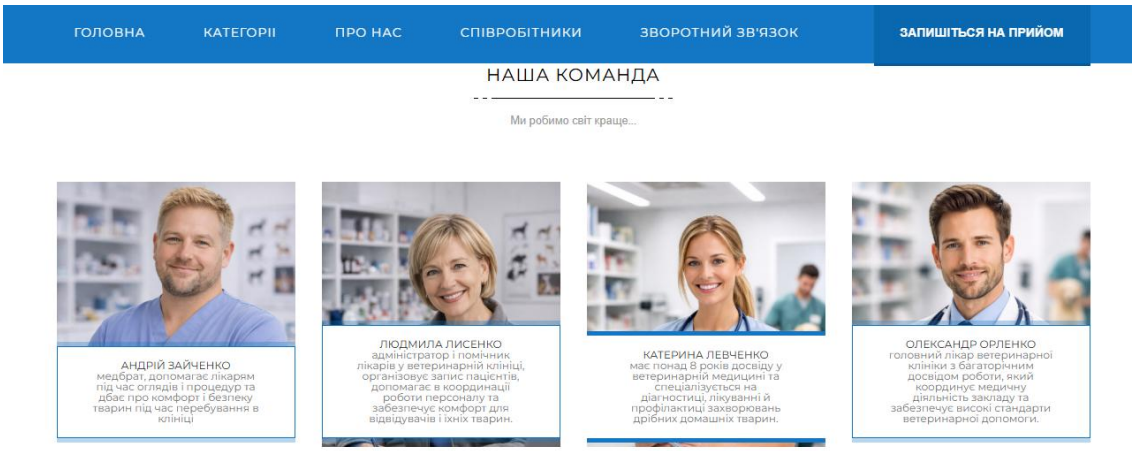


Рисунок 4.11. – Команда працівників клініки

Значущою складовою користувацького інтерфейсу є також підтримка динамічних можливостей за допомогою JavaScript. У проєкті використовуються окремі сценарії для роботи з коментарями, відгуками, підпискою та іншими інтерактивними елементами. Це означає, що система не обмежується лише серверним відображенням сторінок, а має додаткові механізми підвищення зручності взаємодії. Такі можливості покращують сприйняття інтерфейсу та роблять його більш сучасним.

Рисунок 4.12. – Частини функціоналу сайту

З точки зору основних функціональних можливостей вебзастосунок поєднує три ключові напрями. Перший полягає в інформаційному представленні клініки через сторінки про послуги, працівників, публікації та контактну інформацію. Другий пов’язаний із сервісною взаємодією через запис на прийом, форму зворотного зв’язку, підписку на розсилку та відгуки. Третій напрям реалізується в адміністративній частині, де забезпечується керування всіма основними об’єктами системи. Саме поєднання цих трьох напрямів у межах одного вебзастосунку визначає його практичну цінність і відрізняє його від звичайного сайту-візитівки.

Отже, користувацький інтерфейс розробленого вебзастосунку побудовано як цілісну багатосторінкову систему, у якій поєднуються публічна інформаційна частина, сервісні форми взаємодії та адміністративна панель керування. Реалізація інтерфейсу засобами Razor, HTML5, CSS3, JavaScript і Bootstrap забезпечила структурованість сторінок, адаптивність дизайну, повторне використання компонентів і зручність навігації. Основні функціональні можливості вебзастосунку охоплюють представлення діяльності клініки, запис на прийом, перегляд послуг і працівників, роботу з публікаціями, а також повноцінне адміністрування даних. Саме така організація інтерфейсу забезпечує практичну придатність системи до використання у діяльності ветеринарної клініки.

ВИСНОВКИ

У результаті виконання дипломної роботи досягнуто поставленої мети, яка полягала у розробці вебзастосунку для ветеринарної клініки.

У ході роботи було виконано основні завдання дослідження. Проведено аналіз існуючих вебресурсів і програмних рішень аналогічного призначення, що дозволило визначити їх сильні та слабкі сторони. На основі цього було сформовано основні функціональні вимоги до вебзастосунку.

Обґрунтовано вибір технологій і засобів розробки. Для реалізації проєкту використано ASP.NET Core MVC, Entity Framework Core, LINQ, Razor, HTML5, CSS3, JavaScript, Bootstrap і Microsoft SQL Server. Такий набір технологій забезпечив можливість створення структурованого, функціонального та зручного у використанні вебзастосунку.

У роботі спроєктовано структуру системи, розроблено архітектуру вебзастосунку та реалізовано основні програмні модулі. Створено серверну частину, яка забезпечує обробку запитів користувачів, взаємодію з базою даних і керування основними сутностями системи. Організовано роботу з базою даних, у якій реалізовано збереження інформації про працівників, послуги, записи на прийом, публікації, категорії, коментарі, теги, навігаційні елементи та параметри сайту.

Окрему увагу приділено реалізації механізмів доступу до даних на основі Repository Pattern, що дало змогу відокремити логіку роботи з даними від контролерів і зробити систему більш впорядкованою. Розроблено користувацький інтерфейс вебзастосунку, який забезпечує перегляд основної інформації про клініку, послуги, працівників, публікації та підтримує запис на прийом через вебформу.

У процесі реалізації застосунку забезпечено валідацію введених даних за допомогою DataAnnotations, реалізовано роботу із файлами та зображеннями

через FileService, а також передбачено обробку статусів записів на прийом із використанням enum. Проведено перевірку працездатності системи, що підтвердила коректність реалізації основних функціональних можливостей.

Отже, розроблений вебзастосунок може бути використаний як практичне програмне рішення для інформаційної підтримки та автоматизації окремих процесів роботи ветеринарної клініки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. ASP.NET documentation. Режим доступу URL: https://learn.microsoft.com/en-us/aspnet/core/?view=aspnetcore-10.0 – Назва з екрану.

2. ASP.NET Core MVC with EF Core. Режим доступу URL: https://learn.microsoft.com/en-us/aspnet/core/data/ef-mvc/?view=aspnetcore-10.0 – Назва з екрану.

3. Overview of Entity Framework Core - EF Core. Режим доступу URL: https://learn.microsoft.com/en-us/ef/core/ – Назва з екрану.

4. Razor syntax reference for ASP.NET Core. Режим доступу URL: https://learn.microsoft.com/en-us/aspnet/core/mvc/views/razor?view=aspnetcore-10.0 – Назва з екрану.

5. Model validation in ASP.NET Core MVC and Razor Pages. Режим доступу URL: https://learn.microsoft.com/en-us/aspnet/core/mvc/models/validation?view=aspnetcore-10.0 – Назва з екрану.

6. Upload files in ASP.NET Core. Режим доступу URL: https://learn.microsoft.com/en-us/aspnet/core/mvc/models/file-uploads?view=aspnetcore-10.0 – Назва з екрану.

7. ASP.NET Core fundamentals overview. Режим доступу URL: https://learn.microsoft.com/en-us/aspnet/core/fundamentals/?view=aspnetcore-10.0 – Назва з екрану.

8. Common web application architectures - .NET. Режим доступу URL: https://learn.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/common-web-application-architectures – Назва з екрану.

9. Implementing the Repository and Unit of Work Patterns in an ASP.NET MVC Application. Режим доступу URL: https://learn.microsoft.com/en-us/aspnet/mvc/overview/older-versions/getting-started-with-ef-5-using-mvc-

4/implementing-the-repository-and-unit-of-work-patterns-in-an-asp-net-mvc-application – Назва з екрану.

10. SQL Server technical documentation. Режим доступу URL: https://learn.microsoft.com/en-us/sql/sql-server/?view=sql-server-ver17 – Назва з екрану.

11. Introduction Bootstrap v5.3. Режим доступу URL: https://getbootstrap.com/docs/5.3/getting-started/introduction/ – Назва з екрану.

12. Forms Bootstrap v5.3. Режим доступу URL: https://getbootstrap.com/docs/5.3/forms/overview/ – Назва з екрану.

13. Tables Bootstrap v5.3. Режим доступу URL: https://getbootstrap.com/docs/5.3/content/tables/ – Назва з екрану.

14. Application of information technologies in veterinary medicine. Режим доступу URL: https://nvlvet.com.ua/index.php/journal/article/view/5462 – Назва з екрану.

15. An Online Scheduling Platform for Veterinary Appointments. Режим доступу URL: https://journal.uitm.edu.my/ojs/index.php/JI/article/view/6631 – Назва з екрану.

16. TerraVet: A Mobile and Web Application Framework for Pet Owners and Veterinary Clinic. Режим доступу URL: https://dl.acm.org/doi/10.1145/3568923.3568927 – Назва з екрану.

17. The Development of Veterinary Clinic Management Application System. Режим доступу URL: https://publisher.uthm.edu.my/periodicals/index.php/aitcs/article/download/2339/1370/28224 – Назва з екрану.

18. Veterinary Clinic Management Application System. Режим доступу URL: https://publisher.uthm.edu.my/periodicals/index.php/aitcs/article/view/2344 – Назва з екрану.

19. TerraVet: A Mobile and Web Application Framework for Pet Owners and Veterinary Clinic. Режим доступу URL: https://doi.org/10.1145/3568923.3568927 – Назва з екрану.

20. Кошова О. П. РОЗРОБКА ВЕБ-ДОДАТКІВ ТА СЕРВІСІВ НА ПЛАТФОРМІ NODE.JS / О. П. Кошова, О.В. Ольховська О.В., Д. С. Тацій, Ю.Ф. Олексійчук, О.О. Черненко // Таврійський науковий вісник. Серія: Технічні науки, 2023. Вип. 2. С. 78-89. Режим доступу: https://journals.ksauniv.ks.ua/index.php/tech/issue/view/26 <https://journals.ksauniv.ks.ua/index.php/tech/article/view/358/331> – Назва з екрану.

21. Ольховська О. В. Методичні рекомендації до виконання кваліфікаційної роботи для студентів спеціальності 122 Комп'ютерні науки освітня програма «Комп'ютерні науки» ступеня бакалавра / О. В. Ольховська, О. О. Черненко. – Полтава: ПУЕТ, 2024. – 67 с. Режим доступу: URL: http://dspace.puet.edu.ua/bitstream/123456789/14768/1/%D0%9C%D0%B5%D1%82%D0%BE%D0%B4%D0%B8%D1%87%D0%BA%D0%B0%20%D0%91%D0%A0%20%D0%B4%D0%BB%D1%8F%20%D0%9A%D0%9D%202024_%D0%B1%D0%B0%D0%BA%D0%B0%D0%BB%D0%B0%D0%B2%D1%80.pdf - Назва з екрану.

ДОДАТОК А
Код програми

FILE: Program.cs
==========

```
using Microsoft.AspNetCore.Authentication.Cookies;
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;
using WebApplicationVetClinic.Data;
using WebApplicationVetClinic.Models;

namespace WebApplicationVetClinic
{
   public class Program
   {
      public static void Main(string[] args)
      {
         var builder = WebApplication.CreateBuilder(args);

         // Add services to the container.
         var connectionString = builder.Configuration.GetConnectionString("DefaultConnection") ?? throw new
InvalidOperationException("Connection string 'DefaultConnection' not found.");
         builder.Services.AddDbContext<ApplicationDbContext>(options =>
            options.UseSqlServer(connectionString));
         builder.Services.AddDatabaseDeveloperPageExceptionFilter();

         builder.Services.AddAuthentication(CookieAuthenticationDefaults.AuthenticationScheme).AddCookie(
            option =>
            {
               option.LoginPath = new PathString("/Account/LoginIn");
               option.AccessDeniedPath = new PathString("/Error/AcssessDenied");
            }
         );

         builder.Services.AddScoped<CategoryReposytory>();
         builder.Services.AddScoped<CommentRepository>();
         builder.Services.AddScoped<EmployeeRepository>();
         builder.Services.AddScoped<NavigationRepository>();
         builder.Services.AddScoped<OptionsRepository>();
         builder.Services.AddScoped<PostRepository>();
         builder.Services.AddScoped<ServicesRepository>();
         builder.Services.AddScoped<SubscriberRepository>();
         builder.Services.AddScoped<TagRepository>();
         builder.Services.AddScoped<AppointmentRepository>();

         builder.Services.AddControllersWithViews();
         builder.Services.AddHttpContextAccessor();

         var app = builder.Build();

         using (var scope = app.Services.CreateScope())
         {
            var context = scope.ServiceProvider.GetRequiredService<ApplicationDbContext>();
            DbInitializer.Initialize(context);
         }
         // Configure the HTTP request pipeline.
         if (app.Environment.IsDevelopment())
         {
            app.UseMigrationsEndPoint();
         }
         else
         {
            app.UseExceptionHandler("/Home/Error");
            // The default HSTS value is 30 days. You may want to change this for production scenarios, see https://aka.ms/aspnetcore-hsts.
            app.UseHsts();
         }

         app.UseHttpsRedirection();
         app.UseStaticFiles();

         app.UseRouting();

         StaticHttpContextExtensions.UseStaticHttpContext(app);

         app.UseAuthentication();
```

```
        app.UseAuthorization();


        app.UseMiddleware<CustomMiddleWare>();

        app.MapControllerRoute(
            name: "default",
            pattern: "{controller=Home}/{action=Index}/{id?}");
        //app.MapRazorPages();

        app.Run();
    }
  }
}
```

==========

FILE: WebApplicationVetClinic.csproj

==========

```
<Project Sdk="Microsoft.NET.Sdk.Web">

  <PropertyGroup>
    <TargetFramework>net8.0</TargetFramework>
    <Nullable>enable</Nullable>
    <ImplicitUsings>enable</ImplicitUsings>
    <UserSecretsId>aspnet-WebApplicationVetClinic-d63041cb-e97a-4198-ab2f-d738d819b427</UserSecretsId>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="Microsoft.AspNetCore.Diagnostics.EntityFrameworkCore" Version="8.0.18" />
    <PackageReference Include="Microsoft.AspNetCore.Identity.EntityFrameworkCore" Version="8.0.18" />
    <PackageReference Include="Microsoft.AspNetCore.Identity.UI" Version="8.0.18" />
    <PackageReference Include="Microsoft.EntityFrameworkCore.SqlServer" Version="8.0.18" />
    <PackageReference Include="Microsoft.EntityFrameworkCore.Tools" Version="8.0.18" />
    <PackageReference Include="Newtonsoft.Json" Version="13.0.4" />
  </ItemGroup>

</Project>
```

==========

FILE: Data/ApplicationDbContext.cs

==========

```
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;
using WebApplicationVetClinic.Entities;

namespace WebApplicationVetClinic.Data
{
    public class ApplicationDbContext : IdentityDbContext<IdentityUser>
    {
        public DbSet<Employee> Employees { get; set; }
        public DbSet<Service> Services { get; set; }
        public DbSet<Option> Options { get; set; }
        public DbSet<Navigate> Navigations { get; set; }
        public DbSet<Category> Categories { get; set; }
        public DbSet<Comment> Comments { get; set; }
        public DbSet<Subscribe> Subscribes { get; set; }
        public DbSet<Post> Posts { get; set; }
        public DbSet<Tag> Tags { get; set; }
        public DbSet<PostTags> PostTags { get; set; }
        public DbSet<Appointment> Appointments { get; set; }
        public DbSet<User> Users { get; set; }

        public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
            : base(options)
        {
            Database.EnsureCreated();
        }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            base.OnModelCreating(modelBuilder);
```

```
            modelBuilder.Entity<Category>()
              .HasIndex(c => c.Slug)
              .IsUnique();

            modelBuilder.Entity<Option>()
               .HasIndex(o => o.Name)
               .IsUnique();

            modelBuilder.Entity<Post>()
               .HasIndex(p => p.UrlSlug)
               .IsUnique();


            modelBuilder.Entity<Tag>()
               .HasIndex(t => t.Slug)
               .IsUnique();


            modelBuilder.Entity<Subscribe>()
               .HasIndex(o => o.Email)
               .IsUnique();


            modelBuilder.Entity<Appointment>(entity =>
            {
               entity.HasKey(x => x.Id);

               entity.Property(x => x.FirstName)
                  .IsRequired()
                  .HasMaxLength(100);

               entity.Property(x => x.LastName)
                  .IsRequired()
                  .HasMaxLength(100);

               entity.Property(x => x.Email)
                  .IsRequired()
                  .HasMaxLength(150);

               entity.Property(x => x.Phone)
                  .IsRequired()
                  .HasMaxLength(30);

               entity.Property(x => x.Message)
                  .IsRequired()
                  .HasMaxLength(2000);

               entity.Property(x => x.CreatedAt)
                  .IsRequired();
            });
         }
      }

}

================================================================================================
FILE: Controllers/HomeController.cs
================================================================================================

using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using WebApplicationVetClinic.Data;
using WebApplicationVetClinic.Entities;
using WebApplicationVetClinic.Models;
using WebApplicationVetClinic.ViewModels;

namespace WebApplicationVetClinic.Controllers
{
   public class HomeController : Controller
   {
      private readonly ApplicationDbContext _dBContext;

      ServicesRepository _servicesRepository;
      EmployeeRepository _employeeRepository;
      PostRepository _postsRepository;
      AppointmentRepository _appointmentRepository;
```

```
    public HomeController(ServicesRepository servicesRepository, EmployeeRepository employeeRepository, PostRepository postsRepository, AppointmentRepository appointmentRepository)
    {
        _servicesRepository = servicesRepository;
        _employeeRepository = employeeRepository;
        _postsRepository = postsRepository;
        _appointmentRepository = appointmentRepository;
    }

    public IActionResult Index()
    {
        HomePageViewModel homePageViewModel = new HomePageViewModel();
        homePageViewModel.Services = _servicesRepository.GetServices();
        homePageViewModel.Employees = _employeeRepository.GetEmployees();
        homePageViewModel.LatestPosts = _postsRepository.GetLatestPosts(3);

        return View(homePageViewModel);
    }

    public IActionResult Employees()
    {

        return View(_employeeRepository.GetEmployees());
    }

    public IActionResult Employeer(int id)
    {
        return View(_employeeRepository.GetEmployeeById(id));
    }


    [HttpPost]
    public IActionResult Appointment(AppointmentViewModel model)
    {
        if (!ModelState.IsValid)
        {
            HomePageViewModel homePageViewModel = new HomePageViewModel();
            homePageViewModel.Services = _servicesRepository.GetServices();
            homePageViewModel.Employees = _employeeRepository.GetEmployees();
            homePageViewModel.LatestPosts = _postsRepository.GetLatestPosts(3);
            homePageViewModel.Appointment = model;

            return View("Index", homePageViewModel);
        }

        var appointment = new Appointment
        {
            FirstName = model.FirstName,
            LastName = model.LastName,
            Email = model.Email,
            Phone = model.Phone,
            Message = model.Message,
            CreatedAt = DateTime.UtcNow,
            AppointmentDate = model.AppointmentDate
        };

        _appointmentRepository.SaveAppointment(appointment);

        return RedirectToAction("Index");
    }
  }
}
```

==========
FILE: Controllers/AboutController.cs
==========

```
using Microsoft.AspNetCore.Mvc;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using WebApplicationVetClinic.Dto;
using WebApplicationVetClinic.Models;
using WebApplicationVetClinic.ViewModels;
```

```
namespace WebApplicationVetClinic.Controllers
{
   public class AboutController : Controller
   {
      EmployeeRepository _employeeRepository;

      public AboutController(EmployeeRepository employeeRepository)
      {
         _employeeRepository = employeeRepository;
      }

      [HttpGet]
      public IActionResult Index()
      {
         AboutPageViewModel aboutPageViewModel = new AboutPageViewModel();
         aboutPageViewModel.Employees = _employeeRepository.GetEmployees();
         return View(aboutPageViewModel);
      }

      [HttpGet]
      public IActionResult ContactUs()
      {
         return View();
      }
      [HttpPost]
      public ViewResult getGuestMessage(GuestMessageDTO guestMessage)
      {
         if (ModelState.IsValid)
         {
            return View("Thanks", guestMessage);
         }
         else
         {
            return View("ContactUs");
         }
      }
   }
}
```

==========

FILE: Controllers/AccountController.cs

==========

```
using Microsoft.AspNetCore.Authentication;
using Microsoft.AspNetCore.Authentication.Cookies;
using Microsoft.AspNetCore.Mvc;
using MyWebApplication.Models;
using System.Security.Claims;
using WebApp.Helpers;
using WebApplicationVetClinic.Data;
using WebApplicationVetClinic.Entities;
using WebApplicationVetClinic.ViewModels;

namespace WebApplicationVetClinic.Controllers
{
   public class AccountController : Controller
   {

      private ApplicationDbContext _clinicDbContext;

      private UserRepository _userModel;
      public AccountController(ApplicationDbContext clinicDbContext)
      {
         _clinicDbContext = clinicDbContext;
         _userModel = new UserRepository(clinicDbContext);
      }

      [HttpGet]
      public IActionResult Index()
      {
         return RedirectToAction("LoginIn");
      }

      [HttpGet]
      public IActionResult LoginIn()
      {
```

```
        return View();
    }

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

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

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

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

    [HttpPost]
    public IActionResult CheckUser(string email, string password)
    {
        User? currentUser = _userModel.GetUserByEmail(email);

        if (currentUser != null && SecurePasswordHasher.Verify(password, currentUser.PasswordHash))
        {

            var claims = new List<Claim>
            {
                new Claim(ClaimsIdentity.DefaultNameClaimType, currentUser.Login)
            };

            var identity = new ClaimsIdentity(claims, "CoockiesAuth", ClaimsIdentity.DefaultNameClaimType,
ClaimsIdentity.DefaultRoleClaimType);

            HttpContext.SignInAsync(CookieAuthenticationDefaults.AuthenticationScheme, new ClaimsPrincipal(identity));

            return RedirectToAction("Index", "Admin");
        }
        else
        {
            return View("LoginIn", new ErrorViewModel() { ErrorMessage = "Email or Password incorrect" });
        }
    }


    [HttpPost]
    public IActionResult RegisterNewUser(string Email, string Login, string Password, string ConfirmPassword)
    {
        if(Password == ConfirmPassword)
        {
            User newUser = new User()
            {
                Email = Email,
                Login = Login,
                PasswordHash = SecurePasswordHasher.Hash(Password)
            };

            if (_userModel.RegisterNewUser(newUser)) {

                return RedirectToAction("LoginIn", "Account");
            }
        }
        return View("RegisterIn", new ErrorViewModel() { ErrorMessage = "Registration of new user incorrect" });
    }

    [HttpGet]
    public IActionResult Logout()
    {
```

```
            HttpContext.SignOutAsync().Wait();
            return RedirectToAction("LoginIn");
        }
    }
}
```

```
================================================================================================
FILE: Controllers/AdminController.cs
================================================================================================
```

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using System.Text.Encodings.Web;
using System.Text.Json;
using WebApp.Helpers;
using WebApplicationVetClinic.Data;
using WebApplicationVetClinic.Dto;
using WebApplicationVetClinic.Entities;
using WebApplicationVetClinic.Models;
using WebApplicationVetClinic.ViewModels;

namespace WebApplicationVetClinic.Controllers
{
    [Authorize]
    public class AdminController : Controller
    {
        private ApplicationDbContext _clinicDbContext;

        private OptionsRepository _optionModel;
        private TagRepository _tagRepository;
        private CategoryReposytory _categoryRepository;
        private EmployeeRepository _employeeRepository;
        private ServicesRepository _serviceRepository;
        private CommentRepository _commentRepository;
        private NavigationRepository _navigationRepository;
        private PostRepository _postRepository;
        private AppointmentRepository _appointmentRepository;
        public AdminController(ApplicationDbContext clinicDbContext)
        {
            _clinicDbContext = clinicDbContext;
            _optionModel = new OptionsRepository(clinicDbContext);
            _tagRepository = new TagRepository(clinicDbContext);
            _categoryRepository = new CategoryReposytory(clinicDbContext);
            _employeeRepository = new EmployeeRepository(clinicDbContext);
            _serviceRepository = new ServicesRepository(clinicDbContext);
            _commentRepository = new CommentRepository(clinicDbContext);
            _navigationRepository = new NavigationRepository(clinicDbContext);
            _postRepository = new PostRepository(clinicDbContext);
            _appointmentRepository = new AppointmentRepository(clinicDbContext);
        }

        [HttpGet]
        public IActionResult Index()
        {
            var now = DateTime.Now;

            var appointments = _appointmentRepository.GetAllAppointments();

            var model = new DashboardViewModel
            {
                UrgentAppointments = appointments
                    .Where(x =>
                        x.AppointmentDate <= now.AddHours(2) &&
                        x.Status != AppointmentStatus.Completed &&
                        x.Status != AppointmentStatus.Cancelled)
                    .OrderBy(x => x.AppointmentDate)
                    .ToList(),

                TodayAppointments = appointments
                    .Where(x =>
                        x.AppointmentDate.Date == now.Date &&
                        x.Status != AppointmentStatus.Completed &&
                        x.Status != AppointmentStatus.Cancelled)
                    .OrderBy(x => x.AppointmentDate)
                    .ToList(),
```

```
        UpcomingAppointments = appointments
            .Where(x =>
                x.AppointmentDate > now.Date &&
                x.Status != AppointmentStatus.Completed &&
                x.Status != AppointmentStatus.Cancelled)
            .OrderBy(x => x.AppointmentDate)
            .ToList()
    };

    return View(model);
}

[HttpPost]
public IActionResult SetAppointmentStatus(int id, AppointmentStatus status)
{
    _appointmentRepository.UpdateAppointmentStatus(id, status);
    return RedirectToAction("Index", "Admin");
}


[HttpGet]
public IActionResult Options()
{
    return View(_optionModel.GetOptions());
}

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

[HttpPost]
public IActionResult SaveNewOption(Option option )
{
    if (option.Relation == null) option.Relation = "";
    if (option.Key == null) option.Key = "";
    if (option.Value == null) option.Value= "";

    ErrorViewModel errorViewModel = new ErrorViewModel();
    if (_optionModel.AddOption(option)) {

        errorViewModel.SucsessMessage = "New Option saved.";
    } else
    {
        errorViewModel.ErrorMessage = "New Option Not saved. Error";
    }
    return RedirectToAction("Options", "Admin");
}

[HttpGet]
public IActionResult EditOption(int id)
{
    var option = _optionModel.GetOptionById(id);

    if (option == null)
    {
        return NotFound();
    }

    return View(option);
}


[HttpPost]
public IActionResult SaveEditedOption(Option option)
{
    var existing = _optionModel.GetOptionById(option.Id);

    if (existing == null)
        return NotFound();

    // ! запрещаем менять имя
    if (existing.IsSystem)
    {
        option.Name = existing.Name;
    }
```

```
    var result = _optionModel.UpdateOption(option);

    if (!result)
    {
        ModelState.AddModelError("", "Update failed");
        return View("EditOption", option);
    }

    return RedirectToAction("Options", "Admin");
}

[HttpPost]
public string RemoveOneOption(int idToRemove)
{
    JsonResponse jsonResponse = new JsonResponse()
    {
        Code = 200,
        Status = StatusResponce.Error
    };
    JsonSerializerOptions jsonSerializerOptions = new JsonSerializerOptions();
    jsonSerializerOptions.Encoder = JavaScriptEncoder.UnsafeRelaxedJsonEscaping;

    string jsonStrResponce = string.Empty;

    try
    {
        if (idToRemove == 0)
        {
            jsonResponse.Message = "Option id => no valid data";
        }
        else
        {
            bool removeResult = _optionModel.RemoveById(idToRemove);
            if (removeResult == false)
            {
                jsonResponse.Message = "Remove option: status error";
            }
            else
            {
                jsonResponse.Status = StatusResponce.Success;
                jsonResponse.Message = "Option removed";
            }
        }
    }
    catch (Exception ex)
    {
        jsonResponse.Status = StatusResponce.Error;
        jsonResponse.Message = "Server error";
    }
    finally
    {
        jsonStrResponce = JsonSerializer.Serialize(jsonResponse, jsonSerializerOptions);
    }
    return jsonStrResponce;
}



[HttpGet]
public IActionResult Tags()
{
    var tags = _tagRepository.GetAllTags();
    return View(tags);
}

[HttpGet]
public IActionResult CreateTag()
{
    return View(new Tag());
}

[HttpPost]
public IActionResult SaveNewTag(Tag tag)
{
    if (tag.Name == null) tag.Name = "";
```

```
    if (tag.Slug == null) tag.Slug = "";

    if (string.IsNullOrWhiteSpace(tag.Name))
    {
        ModelState.AddModelError("Name", "Name is required");
    }

    if (string.IsNullOrWhiteSpace(tag.Slug))
    {
        ModelState.AddModelError("Slug", "Slug is required");
    }

    if (!ModelState.IsValid)
    {
        return View("CreateTag", tag);
    }

    var result = _tagRepository.AddTag(tag);

    if (!result)
    {
        ModelState.AddModelError("", "Tag was not saved");
        return View("CreateTag", tag);
    }

    return RedirectToAction("Tags", "Admin");
}

[HttpGet]
public IActionResult EditTag(int id)
{
    var tag = _tagRepository.GetTagById(id);

    if (tag == null)
    {
        return NotFound();
    }

    return View(tag);
}

[HttpPost]
public IActionResult SaveEditedTag(Tag tag)
{
    if (tag.Name == null) tag.Name = "";
    if (tag.Slug == null) tag.Slug = "";

    if (string.IsNullOrWhiteSpace(tag.Name))
    {
        ModelState.AddModelError("Name", "Name is required");
    }

    if (string.IsNullOrWhiteSpace(tag.Slug))
    {
        ModelState.AddModelError("Slug", "Slug is required");
    }

    if (!ModelState.IsValid)
    {
        return View("EditTag", tag);
    }

    var result = _tagRepository.UpdateTag(tag);

    if (!result)
    {
        ModelState.AddModelError("", "Tag update failed");
        return View("EditTag", tag);
    }

    return RedirectToAction("Tags", "Admin");
}

[HttpPost]
public IActionResult DeleteTag(int id)
{
    var tag = _tagRepository.GetTagById(id);
```

```
    if (tag == null)
    {
        return NotFound();
    }

    var result = _tagRepository.RemoveTag(tag);

    if (!result)
    {
        TempData["ErrorMessage"] = "Tag was not deleted";
    }
    else
    {
        TempData["SuccessMessage"] = "Tag deleted";
    }

    return RedirectToAction("Tags", "Admin");
}


[HttpGet]
public IActionResult Categories()
{
    return View(_categoryRepository.GetAllCategories());
}

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

[HttpPost]
public IActionResult SaveNewCategory(CategoryCreateDto categoryCreateDto)
{
    if (!ModelState.IsValid)
    {
        return View("CreateCategory", categoryCreateDto);
    }

    if (categoryCreateDto.Avatar == null || categoryCreateDto.Avatar.Length == 0)
    {
        ModelState.AddModelError("Avatar", "Avatar is required.");
        return View("CreateCategory", categoryCreateDto);
    }

    if (!ImageValidator.IsValidImage(categoryCreateDto.Avatar, out string error))
    {
        ModelState.AddModelError("Avatar", "Invalid image file.");
        return View("CreateCategory", categoryCreateDto);
    }

    var origExt = Path.GetExtension(categoryCreateDto.Avatar.FileName);

    string uploadFolder = Path.Combine(
        Directory.GetCurrentDirectory(),
        "wwwroot",
        "images",
        "categories"
    );

    if (!Directory.Exists(uploadFolder))
    {
        Directory.CreateDirectory(uploadFolder);
    }

    string fileName = $"{categoryCreateDto.Slug}{origExt}";
    string physicalPath = Path.Combine(uploadFolder, fileName);

    FileService.SaveImage(categoryCreateDto.Avatar, physicalPath);

    Category category = new Category
    {
        Title = categoryCreateDto.Title ?? string.Empty,
        Slug = categoryCreateDto.Slug ?? string.Empty,
        Slogan = categoryCreateDto.Slogan ?? string.Empty,
```

```
            Description = categoryCreateDto.Description ?? string.Empty,
            ImgAlt = categoryCreateDto.ImgAlt ?? string.Empty,
            ImgSrc = $"/images/categories/{fileName}",
            Order = categoryCreateDto.Order
        };

        if (_categoryRepository.SaveCategory(category))
        {
            return RedirectToAction("Categories", "Admin");
        }

        return RedirectToAction("Error");
    }

    [HttpGet]
    public IActionResult EditCategory(int id)
    {
        var category = _categoryRepository.GetCategoryById(id);

        if (category == null)
        {
            return NotFound();
        }

        var dto = new CategoryEditDto
        {
            Id = category.Id,
            Title = category.Title,
            Slug = category.Slug,
            Slogan = category.Slogan,
            Description = category.Description,
            ImgAlt = category.ImgAlt,
            CurrentImgSrc = category.ImgSrc,
            Order = category.Order
        };

        return View(dto);
    }
    [HttpPost]
    public IActionResult SaveEditedCategory(CategoryEditDto dto)
    {
        if (!ModelState.IsValid)
        {
            return View("EditCategory", dto);
        }

        var currentCategory = _categoryRepository.GetCategoryById(dto.Id);

        if (currentCategory == null)
        {
            return NotFound();
        }

        string imagePath = currentCategory.ImgSrc;

        if (dto.Avatar != null && dto.Avatar.Length > 0)
        {
            if (!ImageValidator.IsValidImage(dto.Avatar, out string error))
            {
                ModelState.AddModelError("Avatar", "Invalid image file.");
                return View("EditCategory", dto);
            }

            var origExt = Path.GetExtension(dto.Avatar.FileName);

            string uploadFolder = Path.Combine(
                Directory.GetCurrentDirectory(),
                "wwwroot",
                "images",
                "categories"
            );

            if (!Directory.Exists(uploadFolder))
            {
                Directory.CreateDirectory(uploadFolder);
            }
```

```
            string fileName = $"{dto.Slug}{origExt}";
            string physicalPath = Path.Combine(uploadFolder, fileName);

            FileService.SaveImage(dto.Avatar, physicalPath);

            imagePath = $"/images/categories/{fileName}";
        }

        currentCategory.Title = dto.Title ?? string.Empty;
        currentCategory.Slug = dto.Slug ?? string.Empty;
        currentCategory.Slogan = dto.Slogan ?? string.Empty;
        currentCategory.Description = dto.Description ?? string.Empty;
        currentCategory.ImgAlt = dto.ImgAlt ?? string.Empty;
        currentCategory.ImgSrc = imagePath;
        currentCategory.Order = dto.Order;

        if (_categoryRepository.UpdateCategory(currentCategory))
        {
            return RedirectToAction("Categories", "Admin");
        }

        ModelState.AddModelError("", "Category update failed.");
        return View("EditCategory", dto);
    }

    [HttpPost]
    public IActionResult DeleteCategory(int id)
    {
        var category = _categoryRepository.GetCategoryById(id);

        if (category == null)
        {
            return NotFound();
        }

        if (_categoryRepository.DeleteCategory(category))
        {
            return RedirectToAction("Categories", "Admin");
        }

        return RedirectToAction("Error");
    }



    [HttpGet]
    public IActionResult Employees()
    {
        return View(_employeeRepository.GetEmployees());
    }
    [HttpGet]
    public IActionResult CreateEmployee()
    {
        return View(new EmployeeCreateDto());
    }

    [HttpPost]
    public IActionResult SaveNewEmployee(EmployeeCreateDto dto)
    {
        if (!ModelState.IsValid)
        {
            return View("CreateEmployee", dto);
        }

        if (dto.Avatar == null || dto.Avatar.Length == 0)
        {
            ModelState.AddModelError("Avatar", "Avatar is required.");
            return View("CreateEmployee", dto);
        }

        if (!ImageValidator.IsValidImage(dto.Avatar, out string error))
        {
            ModelState.AddModelError("Avatar", error);
            return View("CreateEmployee", dto);
        }

        var origExt = Path.GetExtension(dto.Avatar.FileName);
```

```
    string uploadFolder = Path.Combine(
        Directory.GetCurrentDirectory(),
        "wwwroot",
        "images",
        "employees"
    );

    if (!Directory.Exists(uploadFolder))
    {
        Directory.CreateDirectory(uploadFolder);
    }

    string safeFileName = Guid.NewGuid().ToString();
    string fileName = $"{safeFileName}{origExt}";
    string physicalPath = Path.Combine(uploadFolder, fileName);

    FileService.SaveImage(dto.Avatar, physicalPath);

    var employee = new Employee
    {
        Fio = dto.Fio ?? string.Empty,
        Speciality = dto.Speciality ?? string.Empty,
        ImgAlt = dto.ImgAlt ?? string.Empty,
        ImgSrc = $"/images/employees/{fileName}"
    };

    if (_employeeRepository.SaveEmployee(employee))
    {
        return RedirectToAction("Employees", "Admin");
    }

    ModelState.AddModelError("", "Employee was not saved.");
    return View("CreateEmployee", dto);
}
[HttpGet]
public IActionResult EditEmployee(int id)
{
    var employee = _employeeRepository.GetEmployeeById(id);

    if (employee == null)
    {
        return NotFound();
    }

    var dto = new EmployeeEditDto
    {
        Id = employee.Id,
        Fio = employee.Fio,
        Speciality = employee.Speciality,
        ImgAlt = employee.ImgAlt,
        CurrentImgSrc = employee.ImgSrc
    };

    return View(dto);
}
[HttpPost]
public IActionResult SaveEditedEmployee(EmployeeEditDto dto)
{
    if (!ModelState.IsValid)
    {
        return View("EditEmployee", dto);
    }

    var currentEmployee = _employeeRepository.GetEmployeeById(dto.Id);

    if (currentEmployee == null)
    {
        return NotFound();
    }

    string imagePath = currentEmployee.ImgSrc;

    if (dto.Avatar != null && dto.Avatar.Length > 0)
    {
        if (!ImageValidator.IsValidImage(dto.Avatar, out string error))
        {
```

```
            ModelState.AddModelError("Avatar", "Invalid image file.");
            return View("EditEmployee", dto);
        }

        var origExt = Path.GetExtension(dto.Avatar.FileName);

        string uploadFolder = Path.Combine(
            Directory.GetCurrentDirectory(),
            "wwwroot",
            "images",
            "employees"
        );

        if (!Directory.Exists(uploadFolder))
        {
            Directory.CreateDirectory(uploadFolder);
        }

        string safeFileName = Guid.NewGuid().ToString();
        string fileName = $"{safeFileName}{origExt}";
        string physicalPath = Path.Combine(uploadFolder, fileName);

        FileService.SaveImage(dto.Avatar, physicalPath);

        imagePath = $"/images/employees/{fileName}";
    }

    currentEmployee.Fio = dto.Fio ?? string.Empty;
    currentEmployee.Speciality = dto.Speciality ?? string.Empty;
    currentEmployee.ImgAlt = dto.ImgAlt ?? string.Empty;
    currentEmployee.ImgSrc = imagePath;

    if (_employeeRepository.UpdateEmployee(currentEmployee))
    {
        return RedirectToAction("Employees", "Admin");
    }

    ModelState.AddModelError("", "Employee update failed.");
    return View("EditEmployee", dto);
}
[HttpPost]
public IActionResult DeleteEmployee(int id)
{
    var employee = _employeeRepository.GetEmployeeById(id);

    if (employee == null)
    {
        return NotFound();
    }

    if (_employeeRepository.DeleteEmployee(employee))
    {
        return RedirectToAction("Employees", "Admin");
    }

    return RedirectToAction("Error");
}


[HttpGet]
public IActionResult Services()
{
    return View(_serviceRepository.GetServices());
}
[HttpGet]
public IActionResult CreateService()
{
    return View(new ServiceCreateDto());
}
[HttpPost]
public IActionResult SaveNewService(ServiceCreateDto dto)
{
    if (!ModelState.IsValid)
    {
        return View("CreateService", dto);
    }
```

```
    if (dto.Avatar == null || dto.Avatar.Length == 0)
    {
        ModelState.AddModelError("Avatar", "Avatar is required.");
        return View("CreateService", dto);
    }

    if (!ImageValidator.IsValidImage(dto.Avatar, out string error))
    {
        ModelState.AddModelError("Avatar", error);
        return View("CreateService", dto);
    }

    var origExt = Path.GetExtension(dto.Avatar.FileName);

    string uploadFolder = Path.Combine(
        Directory.GetCurrentDirectory(),
        "wwwroot",
        "images",
        "services"
    );

    if (!Directory.Exists(uploadFolder))
    {
        Directory.CreateDirectory(uploadFolder);
    }

    string safeFileName = Guid.NewGuid().ToString();
    string fileName = $"{safeFileName}{origExt}";
    string physicalPath = Path.Combine(uploadFolder, fileName);

    FileService.SaveImage(dto.Avatar, physicalPath);

    var service = new Service
    {
        Title = dto.Title ?? string.Empty,
        Description = dto.Description ?? string.Empty,
        ImgAlt = dto.ImgAlt ?? string.Empty,
        ImgSrc = $"/images/services/{fileName}",
        Order = dto.Order
    };

    if (_serviceRepository.SaveService(service))
    {
        return RedirectToAction("Services", "Admin");
    }

    ModelState.AddModelError("", "Service was not saved.");
    return View("CreateService", dto);
}
[HttpGet]
public IActionResult EditService(int id)
{
    var service = _serviceRepository.GetServiceById(id);

    if (service == null)
    {
        return NotFound();
    }

    var dto = new ServiceEditDto
    {
        Id = service.Id,
        Title = service.Title,
        Description = service.Description,
        ImgAlt = service.ImgAlt,
        CurrentImgSrc = service.ImgSrc,
        Order = service.Order
    };

    return View(dto);
}
[HttpPost]
public IActionResult SaveEditedService(ServiceEditDto dto)
{
    if (!ModelState.IsValid)
    {
        return View("EditService", dto);
```

```
    }

    var currentService = _serviceRepository.GetServiceById(dto.Id);

    if (currentService == null)
    {
        return NotFound();
    }

    string imagePath = currentService.ImgSrc;

    if (dto.Avatar != null && dto.Avatar.Length > 0)
    {
        if (!ImageValidator.IsValidImage(dto.Avatar, out string error))
        {
            ModelState.AddModelError("Avatar", error);
            return View("EditService", dto);
        }

        var origExt = Path.GetExtension(dto.Avatar.FileName);

        string uploadFolder = Path.Combine(
            Directory.GetCurrentDirectory(),
            "wwwroot",
            "images",
            "services"
        );

        if (!Directory.Exists(uploadFolder))
        {
            Directory.CreateDirectory(uploadFolder);
        }

        string safeFileName = Guid.NewGuid().ToString();
        string fileName = $"{safeFileName}{origExt}";
        string physicalPath = Path.Combine(uploadFolder, fileName);

        FileService.SaveImage(dto.Avatar, physicalPath);

        imagePath = $"/images/services/{fileName}";
    }

    currentService.Title = dto.Title ?? string.Empty;
    currentService.Description = dto.Description ?? string.Empty;
    currentService.ImgAlt = dto.ImgAlt ?? string.Empty;
    currentService.ImgSrc = imagePath;
    currentService.Order = dto.Order;

    if (_serviceRepository.UpdateService(currentService))
    {
        return RedirectToAction("Services", "Admin");
    }

    ModelState.AddModelError("", "Service update failed.");
    return View("EditService", dto);
}
[HttpPost]
public IActionResult DeleteService(int id)
{
    var service = _serviceRepository.GetServiceById(id);

    if (service == null)
    {
        return NotFound();
    }

    if (_serviceRepository.DeleteService(service))
    {
        return RedirectToAction("Services", "Admin");
    }

    return RedirectToAction("Error");
}


[HttpGet]
public IActionResult Comments()
```

```
{
    var model = new CommentsAdminViewModel
    {
        NewComments = _commentRepository.GetNewComments(),
        ReviewComments = _commentRepository.GetReviewComments(),
        InvalidComments = _commentRepository.GetInvalidComments()
    };

    return View(model);
}
[HttpPost]
public IActionResult MarkCommentAsReview(int id)
{
    _commentRepository.MarkAsReview(id);
    return RedirectToAction("Comments", "Admin");
}
[HttpPost]
public IActionResult MarkCommentAsInvalid(int id)
{
    _commentRepository.MarkAsInvalid(id);
    return RedirectToAction("Comments", "Admin");
}
[HttpPost]
public IActionResult MarkCommentAsNew(int id)
{
    _commentRepository.MarkAsNew(id);
    return RedirectToAction("Comments", "Admin");
}
[HttpPost]
public IActionResult DeleteComment(int id)
{
    var comment = _commentRepository.GetCommentById(id);

    if (comment == null)
    {
        return NotFound();
    }

    _commentRepository.DeleteComment(comment);
    return RedirectToAction("Comments", "Admin");
}


[HttpGet]
public IActionResult Navigations()
{
    var items = _navigationRepository.GetNavigatesThree();
    return View(items);
}

[HttpGet]
public IActionResult CreateNavigate()
{
    ViewBag.AllNavigates = _navigationRepository
        .GetAllNavigatesList()
        .Where(x => x.Parent_Id == null)
        .ToList();

    return View(new NavigateCreateDto());
}
[HttpPost]
public IActionResult SaveNewNavigate(NavigateCreateDto dto)
{
    if (!ModelState.IsValid)
    {
        ViewBag.AllNavigates = _navigationRepository
            .GetAllNavigatesList()
            .Where(x => x.Parent_Id == null)
            .ToList();

        return View("CreateNavigate", dto);
    }

    var navigate = new Navigate
    {
        Title = dto.Title ?? string.Empty,
        Href = dto.Href ?? string.Empty,
```

```
            Order = dto.Order,
            Parent_Id = dto.ParentId
        };

        if (_navigationRepository.SaveNavigate(navigate))
        {
            return RedirectToAction("Navigations", "Admin");
        }

        ModelState.AddModelError("", "Navigate item was not saved.");

        ViewBag.AllNavigates = _navigationRepository
            .GetAllNavigatesList()
            .Where(x => x.Parent_Id == null)
            .ToList();

        return View("CreateNavigate", dto);
    }
    [HttpGet]
    public IActionResult EditNavigate(int id)
    {
        var navigate = _navigationRepository.GetNavigateById(id);

        if (navigate == null)
        {
            return NotFound();
        }

        var dto = new NavigateEditDto
        {
            Id = navigate.Id,
            Title = navigate.Title,
            Href = navigate.Href,
            Order = navigate.Order,
            ParentId = navigate.Parent_Id
        };

        ViewBag.AllNavigates = _navigationRepository
            .GetAllNavigatesList()
            .Where(x => x.Id != id && x.Parent_Id == null)
            .ToList();

        return View(dto);
    }
    [HttpPost]
    public IActionResult SaveEditedNavigate(NavigateEditDto dto)
    {
        if (!ModelState.IsValid)
        {
            ViewBag.AllNavigates = _navigationRepository
                .GetAllNavigatesList()
                .Where(x => x.Id != dto.Id && x.Parent_Id == null)
                .ToList();

            return View("EditNavigate", dto);
        }

        var currentNavigate = _navigationRepository.GetNavigateById(dto.Id);

        if (currentNavigate == null)
        {
            return NotFound();
        }

        if (dto.ParentId == dto.Id)
        {
            ModelState.AddModelError("ParentId", "Element cannot be its own parent.");

            ViewBag.AllNavigates = _navigationRepository
                .GetAllNavigatesList()
                .Where(x => x.Id != dto.Id && x.Parent_Id == null)
                .ToList();

            return View("EditNavigate", dto);
        }

        currentNavigate.Title = dto.Title ?? string.Empty;
```

```
    currentNavigate.Href = dto.Href ?? string.Empty;
    currentNavigate.Order = dto.Order;
    currentNavigate.Parent_Id = dto.ParentId;

    if (_navigationRepository.UpdateNavigate(currentNavigate))
    {
        return RedirectToAction("Navigations", "Admin");
    }

    ModelState.AddModelError("", "Navigate item was not updated.");

    ViewBag.AllNavigates = _navigationRepository
        .GetAllNavigatesList()
        .Where(x => x.Id != dto.Id && x.Parent_Id == null)
        .ToList();

    return View("EditNavigate", dto);
}
[HttpPost]
public IActionResult DeleteNavigate(int id)
{
    var navigate = _navigationRepository.GetNavigateById(id);

    if (navigate == null)
    {
        return NotFound();
    }

    if (_navigationRepository.DeleteNavigate(navigate))
    {
        return RedirectToAction("Navigations", "Admin");
    }

    return RedirectToAction("Error");
}

[HttpGet]
public IActionResult Posts()
{
    return View(_postRepository.GetAllPostsForAdmin());
}
[HttpGet]
public IActionResult CreatePost()
{
    ViewBag.Categories = _categoryRepository.GetAllCategories();
    ViewBag.Tags = _tagRepository.GetAllTags();

    return View(new PostCreateDto
    {
        DateOfPublished = DateTime.Now
    });
}
[HttpPost]
public IActionResult SaveNewPost(PostCreateDto dto)
{
    ViewBag.Categories = _categoryRepository.GetAllCategories();
    ViewBag.Tags = _tagRepository.GetAllTags();

    if (!ModelState.IsValid)
    {
        return View("CreatePost", dto);
    }

    if (dto.Avatar == null || dto.Avatar.Length == 0)
    {
        ModelState.AddModelError("Avatar", "Avatar is required.");
        return View("CreatePost", dto);
    }

    if (!ImageValidator.IsValidImage(dto.Avatar, out string error))
    {
        ModelState.AddModelError("Avatar", error);
        return View("CreatePost", dto);
    }

    var origExt = Path.GetExtension(dto.Avatar.FileName);
```

```
    string uploadFolder = Path.Combine(
        Directory.GetCurrentDirectory(),
        "wwwroot",
        "images",
        "posts"
    );

    if (!Directory.Exists(uploadFolder))
    {
        Directory.CreateDirectory(uploadFolder);
    }

    string safeFileName = Guid.NewGuid().ToString();
    string fileName = $"{safeFileName}{origExt}";
    string physicalPath = Path.Combine(uploadFolder, fileName);

    FileService.SaveImage(dto.Avatar, physicalPath);

    var post = new Post
    {
        Title = dto.Title ?? string.Empty,
        Slogan = dto.Slogan ?? string.Empty,
        ImgAlt = dto.ImgAlt ?? string.Empty,
        ImgSrc = $"/images/posts/{fileName}",
        UrlSlug = dto.UrlSlug ?? string.Empty,
        CategoryId = dto.CategoryId,
        Content = dto.Content ?? string.Empty,
        PostedOn = dto.PostedOn,
        DateOfPublished = dto.DateOfPublished ?? DateTime.Now
    };

    if (_postRepository.SavePost(post, dto.TagIds))
    {
        return RedirectToAction("Posts", "Admin");
    }

    ModelState.AddModelError("", "Post was not saved.");
    return View("CreatePost", dto);
}
[HttpGet]
public IActionResult EditPost(int id)
{
    var post = _postRepository.GetPostById(id);

    if (post == null)
    {
        return NotFound();
    }

    ViewBag.Categories = _categoryRepository.GetAllCategories();
    ViewBag.Tags = _tagRepository.GetAllTags();

    var dto = new PostEditDto
    {
        Id = post.Id,
        Title = post.Title,
        Slogan = post.Slogan,
        ImgAlt = post.ImgAlt,
        CurrentImgSrc = post.ImgSrc,
        UrlSlug = post.UrlSlug,
        CategoryId = post.CategoryId,
        Content = post.Content,
        PostedOn = post.PostedOn,
        DateOfPublished = post.DateOfPublished,
        TagIds = post.PostTags.Select(x => x.TagId).ToList()
    };

    return View(dto);
}
[HttpPost]
public IActionResult SaveEditedPost(PostEditDto dto)
{
    ViewBag.Categories = _categoryRepository.GetAllCategories();
    ViewBag.Tags = _tagRepository.GetAllTags();

    if (!ModelState.IsValid)
    {
```

```
            return View("EditPost", dto);
        }

        var currentPost = _postRepository.GetPostById(dto.Id);

        if (currentPost == null)
        {
            return NotFound();
        }

        string imagePath = currentPost.ImgSrc;

        if (dto.Avatar != null && dto.Avatar.Length > 0)
        {
            if (!ImageValidator.IsValidImage(dto.Avatar, out string error))
            {
                ModelState.AddModelError("Avatar", error);
                return View("EditPost", dto);
            }

            var origExt = Path.GetExtension(dto.Avatar.FileName);

            string uploadFolder = Path.Combine(
                Directory.GetCurrentDirectory(),
                "wwwroot",
                "images",
                "posts"
            );

            if (!Directory.Exists(uploadFolder))
            {
                Directory.CreateDirectory(uploadFolder);
            }

            string safeFileName = Guid.NewGuid().ToString();
            string fileName = $"{safeFileName}{origExt}";
            string physicalPath = Path.Combine(uploadFolder, fileName);

            FileService.SaveImage(dto.Avatar, physicalPath);

            imagePath = $"/images/posts/{fileName}";
        }

        currentPost.Title = dto.Title ?? string.Empty;
        currentPost.Slogan = dto.Slogan ?? string.Empty;
        currentPost.ImgAlt = dto.ImgAlt ?? string.Empty;
        currentPost.ImgSrc = imagePath;
        currentPost.UrlSlug = dto.UrlSlug ?? string.Empty;
        currentPost.CategoryId = dto.CategoryId;
        currentPost.Content = dto.Content ?? string.Empty;
        currentPost.PostedOn = dto.PostedOn;
        currentPost.DateOfPublished = dto.DateOfPublished ?? DateTime.Now;

        if (_postRepository.UpdatePost(currentPost, dto.TagIds))
        {
            return RedirectToAction("Posts", "Admin");
        }

        ModelState.AddModelError("", "Post update failed.");
        return View("EditPost", dto);
    }
    [HttpPost]
    public IActionResult DeletePost(int id)
    {
        var post = _postRepository.GetPostById(id);

        if (post == null)
        {
            return NotFound();
        }

        if (_postRepository.RemovePost(post))
        {
            return RedirectToAction("Posts", "Admin");
        }

        return RedirectToAction("Error");
```

```
        }
        [HttpPost]
        public IActionResult TogglePostPublished(int id)
        {
            var post = _postRepository.GetPostById(id);
            if (post == null) return NotFound();

            post.PostedOn = !post.PostedOn;

            _postRepository.UpdatePost(post, post.PostTags.Select(x => x.TagId).ToList());

            return RedirectToAction("Posts", "Admin");
        }
    }
}
```

```
================================================================================================================
FILE: Controllers/CategoryController.cs
================================================================================================================
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using WebApplicationVetClinic.Models;

namespace WebApplicationVetClinic.Controllers
{
    public class CategoryController : Controller
    {

        private CategoryRepository _categoryRepository;

        public CategoryController(CategoryRepository categoryRepository)
        {
            _categoryRepository = categoryRepository;
        }

        public IActionResult Index()
        {
            return View("Category", _categoryRepository.GetAllCategories());
        }
    }
}
```

```
================================================================================================================
FILE: Controllers/PostController.cs
================================================================================================================
```

```
using Microsoft.AspNetCore.Mvc;
using WebApplicationVetClinic.Entities;
using WebApplicationVetClinic.Models;
using WebApplicationVetClinic.ViewModels;

namespace WebApplicationVetClinic.Controllers
{
    public class PostController : Controller
    {

        private PostRepository _postRepository;
        private TagRepository _tagRepository;
        private CategoryRepository _categoryRepository;

        public PostController(PostRepository postRepository, TagRepository tagRepository, CategoryRepository categoryRepository )
        {
            _postRepository = postRepository;
            _tagRepository = tagRepository;
            _categoryRepository = categoryRepository;
        }

        public IActionResult Index(string? category, string? tag, int page = 1, int pageSize = 5)
        {
            if (page < 1)
```

```
            page = 1;

        var blogViewModel = new BlogViewModel
        {
            Tags = _tagRepository.GetNotEmptyTags(),
            Categories = _categoryReposytory.GetAllCategories(),
            CurrentCategory = category,
            CurrentTag = tag,
            CurrentPage = page,
            PageSize = pageSize
        };

        IEnumerable<Post> posts;

        if (!string.IsNullOrEmpty(category))
        {
            posts = _postReposytory.GetPostsByCategorySlug(category);
        }
        else if (!string.IsNullOrEmpty(tag))
        {
            posts = _postReposytory.GetPostsByTagSlug(tag);
        }
        else
        {
            posts = _postReposytory.GetPosts();
        }

        posts = posts.OrderByDescending(p => p.DateOfPublished);

        blogViewModel.TotalItems = posts.Count();

        blogViewModel.Posts = posts
            .Skip((page - 1) * pageSize)
            .Take(pageSize)
            .ToList();

        return View("AllPosts", blogViewModel);
    }

    public IActionResult PostNotFound()
    {
        return View("PostNotFound");
    }

    [HttpGet]
    public IActionResult SinglePost(string? slug)
    {
        if (slug == null)
        {
            return RedirectToAction("Index", "Error");
        }
        Post? onePost = _postReposytory.GetPostBySlug(slug);
        if (onePost == null)
        {
            return RedirectToAction("Index", "Error");
        }
        return View(onePost);
    }
  }
}


================================================================================================
FILE: Entities/Appointment.cs
================================================================================================

using System.ComponentModel.DataAnnotations;

namespace WebApplicationVetClinic.Entities
{
  public class Appointment
  {
    public int Id { get; set; }

    [Required]
    [MaxLength(100)]
    public string FirstName { get; set; } = null!;
```

```
        [Required]
        [MaxLength(100)]
        public string LastName { get; set; } = null!;

        [Required]
        [MaxLength(150)]
        public string Email { get; set; } = null!;

        [Required]
        [MaxLength(30)]
        public string Phone { get; set; } = null!;

        [Required]
        [MaxLength(2000)]
        public string Message { get; set; } = null!;

        [Required]
        public DateTime AppointmentDate { get; set; }

        public DateTime CreatedAt { get; set; } = DateTime.UtcNow;

        public AppointmentStatus Status { get; set; } = AppointmentStatus.New;
    }
}
```

```
================================================================================================
FILE: Entities/Category.cs
================================================================================================

using System.ComponentModel.DataAnnotations;

namespace WebApplicationVetClinic.Entities
{
    public class Category
    {
        public int Id { get; set; }

        [Required(ErrorMessage = "Title is required")]
        public string Title { get; set; } = String.Empty;

        [Required(ErrorMessage = "Slug is required")]
        public string Slug { get; set; } = String.Empty;

        public string Slogan { get; set; } = String.Empty;

        public string Description { get; set; } = String.Empty;

        public string ImgSrc { get; set; } = String.Empty;

        public string ImgAlt { get; set; } = String.Empty;

        public int Order { get; set; }

        public ICollection<Post> Posts { get; set; } = new List<Post>();
    }
}
```

```
================================================================================================
FILE: Entities/Employee.cs
================================================================================================

using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Threading.Tasks;

namespace WebApplicationVetClinic.Entities
{
    public class Employee
    {
        public int Id { get; set; }

        public string ImgSrc { get; set; } = String.Empty;
```

```
        public string ImgAlt { get; set; } = String.Empty;

        [Required(ErrorMessage = "Fio is required")]
        public string Fio { get; set; } = string.Empty;

        [Required(ErrorMessage = "Speciality is required")]
        public string Speciality { get; set; } = string.Empty;

        public string Context { get; set; } = String.Empty;
    }
}
```

==========

FILE: Entities/Post.cs

==========

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
using System.Linq;
using System.Threading.Tasks;

namespace WebApplicationVetClinic.Entities
{
    public class Post
    {
        public int Id { get; set; }

        [Required(ErrorMessage = "Title is required")]
        public string Title { get; set; } = string.Empty;

        public string Slogan { get; set; } = String.Empty;

        public string ImgSrc { get; set; } = String.Empty;

        public string ImgAlt { get; set; } = String.Empty;

        [Required(ErrorMessage = "UrlSlug is required")]
        public string UrlSlug { get; set; } = string.Empty;

        [ForeignKey("Category")]
        public int CategoryId { get; set; }

        public string Content { get; set; } = String.Empty;

        [DefaultValue(false)]
        public bool PostedOn { get; set; }

        public DateTime DateOfPublished { get; set; }

        public Category Category { get; set; }

        public ICollection<Comment> Comments { get; set; } = new List<Comment>();


        public ICollection<PostTags> PostTags { get; set; } = new List<PostTags>();
    }
}
```

==========

FILE: Entities/Service.cs

==========

```
using System.ComponentModel.DataAnnotations;

namespace WebApplicationVetClinic.Entities
{
    public class Service
    {
        public int Id { get; set; }

        [Required(ErrorMessage = "Title is required")]
```

```
        public string Title { get; set; } = string.Empty;

        public string Description { get; set; } = String.Empty;

        public string ImgSrc { get; set; } = String.Empty;

        public string ImgAlt { get; set; } = String.Empty;
        public int Order { get; set; }
    }
}
```

```
================================================================================================
FILE: Entities/User.cs
================================================================================================

namespace WebApplicationVetClinic.Entities
{
    public class User
    {
        public int Id { get; set; }
        public string Login { get; set; } = string.Empty;
        public string Email { get; set; } = string.Empty;
        public string PasswordHash { get; set; } = string.Empty;
        public DateTime DateOfCreated { get; set; } = DateTime.Now;
    }
}
```

```
================================================================================================
FILE: Models/AppointmentRepository.cs
================================================================================================

using Microsoft.EntityFrameworkCore;
using WebApplicationVetClinic.Data;
using WebApplicationVetClinic.Entities;

namespace WebApplicationVetClinic.Models
{
    public class AppointmentRepository
    {
        private readonly ApplicationDbContext _dBContext;
        public AppointmentRepository(ApplicationDbContext clinicDBContext)
        {
            _dBContext = clinicDBContext;
        }

        public bool SaveAppointment(Appointment appointment)
        {
            _dBContext.Appointments.Add(appointment);
            return _dBContext.SaveChanges() == 1 ? true : false;
        }

        public List<Appointment> GetAllAppointments()
        {
            return _dBContext.Appointments
                .OrderBy(x => x.AppointmentDate)
                .ToList();
        }

        public List<Appointment> GetUpcomingAppointments(DateTime now, DateTime to)
        {
            return _dBContext.Appointments
                .Where(x => x.AppointmentDate >= now && x.AppointmentDate <= to)
                .OrderBy(x => x.AppointmentDate)
                .ToList();
        }

        public Appointment? GetAppointmentById(int id)
        {
            return _dBContext.Appointments.FirstOrDefault(x => x.Id == id);
        }

        public bool UpdateAppointmentStatus(int id, AppointmentStatus status)
        {
            try
            {
```

```
            var appointment = _dBContext.Appointments.FirstOrDefault(x => x.Id == id);
            if (appointment == null) return false;

            appointment.Status = status;
            _dBContext.SaveChanges();

            return true;
        }
        catch
        {
            return false;
        }
    }
  }
}
```

==========
FILE: Models/CategoryReposytory.cs
==========

```
using WebApplicationVetClinic.Entities;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using WebApplicationVetClinic.Data;

namespace WebApplicationVetClinic.Models
{
  public class CategoryReposytory
  {
    private readonly ApplicationDbContext _dBContext;
    public CategoryReposytory(ApplicationDbContext clinicDBContext)
    {
      _dBContext = clinicDBContext;
    }
    public List<Category> GetAllCategories()
    {
      return _dBContext.Categories.OrderBy(x => x.Order).ToList();
    }

    public Category? GetCategoryById(int id)
    {
      return _dBContext.Categories.FirstOrDefault(x => x.Id == id);
    }

    public bool SaveCategory(Category category)
    {
      try
      {
        _dBContext.Categories.Add(category);
        _dBContext.SaveChanges();
        return true;
      }
      catch
      {
        return false;
      }
    }

    public bool UpdateCategory(Category category)
    {
      try
      {
        var currentCategory = _dBContext.Categories.FirstOrDefault(x => x.Id == category.Id);

        if (currentCategory == null)
          return false;

        currentCategory.Title = category.Title ?? string.Empty;
        currentCategory.Slug = category.Slug ?? string.Empty;
        currentCategory.Slogan = category.Slogan ?? string.Empty;
        currentCategory.Description = category.Description ?? string.Empty;
        currentCategory.ImgAlt = category.ImgAlt ?? string.Empty;
        currentCategory.ImgSrc = category.ImgSrc ?? string.Empty;
        currentCategory.Order = category.Order;
```

```
            _dBContext.SaveChanges();
            return true;
        }
        catch
        {
            return false;
        }
    }

    public bool DeleteCategory(Category category)
    {
        try
        {
            _dBContext.Categories.Remove(category);
            _dBContext.SaveChanges();
            return true;
        }
        catch
        {
            return false;
        }
    }
  }
}
```

==========
FILE: Models/EmployeeRepository.cs
==========

```
using WebApplicationVetClinic.Entities;
using WebApplicationVetClinic.Data;

namespace WebApplicationVetClinic.Models
{
  public class EmployeeRepository
  {
    private readonly ApplicationDbContext _dBContext;
    public EmployeeRepository(ApplicationDbContext clinicDBContext)
    {
        _dBContext = clinicDBContext;
    }
    public List<Employee> GetEmployees()
    {
        return _dBContext.Employees
            .OrderBy(x => x.Id)
            .ToList();
    }

    public Employee? GetEmployeeById(int id)
    {
        return _dBContext.Employees.FirstOrDefault(x => x.Id == id);
    }

    public bool SaveEmployee(Employee employee)
    {
        try
        {
            _dBContext.Employees.Add(employee);
            _dBContext.SaveChanges();
            return true;
        }
        catch
        {
            return false;
        }
    }

    public bool UpdateEmployee(Employee employee)
    {
        try
        {
            var currentEmployee = _dBContext.Employees.FirstOrDefault(x => x.Id == employee.Id);

            if (currentEmployee == null)
                return false;
```

```
            currentEmployee.Fio = employee.Fio ?? string.Empty;
            currentEmployee.Speciality = employee.Speciality ?? string.Empty;
            currentEmployee.ImgAlt = employee.ImgAlt ?? string.Empty;
            currentEmployee.ImgSrc = employee.ImgSrc ?? string.Empty;

            _dBContext.SaveChanges();
            return true;
        }
        catch
        {
            return false;
        }
    }

    public bool DeleteEmployee(Employee employee)
    {
        try
        {
            _dBContext.Employees.Remove(employee);
            _dBContext.SaveChanges();
            return true;
        }
        catch
        {
            return false;
        }
    }
  }
}
```

==========
FILE: Models/PostRepository.cs
==========

```
using Microsoft.EntityFrameworkCore;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using WebApplicationVetClinic.Data;
using WebApplicationVetClinic.Entities;

namespace WebApplicationVetClinic.Models
{
  public class PostRepository
  {
    private readonly ApplicationDbContext _dBContext;
    public PostRepository(ApplicationDbContext clinicDBContext)
    {
      _dBContext = clinicDBContext;
    }

    public IEnumerable<Post> GetPostsByCategorySlug(string categorySlug)
    {
      return _dBContext.Posts
        .AsNoTracking()
        .Include(p => p.Category)
        .Include(p => p.Comments)
        .Include(p => p.PostTags)
          .ThenInclude(pt => pt.Tag)
        .Where(p => p.PostedOn && p.Category != null && p.Category.Slug == categorySlug)
        .OrderByDescending(p => p.DateOfPublished)
        .ToList();
    }

    public IEnumerable<Post> GetPostsByTagSlug(string tagSlug)
    {
      return _dBContext.Posts
        .AsNoTracking()
        .Include(p => p.Category)
        .Include(p => p.Comments)
        .Include(p => p.PostTags)
          .ThenInclude(pt => pt.Tag)
        .Where(p => p.PostedOn && p.PostTags.Any(pt => pt.Tag.Slug == tagSlug))
        .OrderByDescending(p => p.DateOfPublished)
```

```
        .ToList();
}

public IEnumerable<Post> GetPosts()
{
    return _dBContext.Posts
        .AsNoTracking()
        .Include(p => p.Category)
        .Include(p => p.Comments)
        .Include(p => p.PostTags)
            .ThenInclude(pt => pt.Tag)
        .Where(p => p.PostedOn)
        .OrderByDescending(p => p.DateOfPublished)
        .ToList();
}

public List<Post> GetAllPostsForAdmin()
{
    return _dBContext.Posts
        .Include(p => p.Category)
        .Include(p => p.Comments)
        .Include(p => p.PostTags)
            .ThenInclude(pt => pt.Tag)
        .OrderByDescending(p => p.DateOfPublished)
        .ThenByDescending(p => p.Id)
        .ToList();
}

public Post? GetOnePost(string slug)
{
    return _dBContext.Posts
        .Include(p => p.Category)
        .Include(p => p.PostTags)
            .ThenInclude(pt => pt.Tag)
        .FirstOrDefault(x => x.UrlSlug.Contains(slug));
}

public List<Post> GetLatestPosts(int count)
{
    return _dBContext.Posts
        .AsNoTracking()
        .Where(p => p.PostedOn)
        .OrderByDescending(p => p.DateOfPublished)
        .Include(p => p.Comments)
        .Take(count)
        .ToList();
}

public Post? GetPostBySlug(string slug)
{
    return _dBContext.Posts
        .Include(p => p.Category)
        .Include(p => p.PostTags)
            .ThenInclude(pt => pt.Tag)
        .FirstOrDefault(p => p.UrlSlug.ToLower() == slug.ToLower());
}

public Post? GetPostById(int postId)
{
    return _dBContext.Posts
        .Include(p => p.Category)
        .Include(p => p.PostTags)
        .FirstOrDefault(p => p.Id == postId);
}

public bool SavePost(Post post, List<int> tagIds)
{
    try
    {
        _dBContext.Posts.Add(post);
        _dBContext.SaveChanges();

        if (tagIds.Any())
        {
            var postTags = tagIds
                .Distinct()
                .Select(tagId => new PostTags
```

```
                {
                    PostId = post.Id,
                    TagId = tagId
                })
                .ToList();

            _dBContext.PostTags.AddRange(postTags);
            _dBContext.SaveChanges();
        }

        return true;
    }
    catch
    {
        return false;
    }
}

public bool UpdatePost(Post post, List<int> tagIds)
{
    try
    {
        var currentPost = _dBContext.Posts
            .Include(x => x.PostTags)
            .FirstOrDefault(x => x.Id == post.Id);

        if (currentPost == null)
            return false;

        currentPost.Title = post.Title ?? string.Empty;
        currentPost.Slogan = post.Slogan ?? string.Empty;
        currentPost.ImgSrc = post.ImgSrc ?? string.Empty;
        currentPost.ImgAlt = post.ImgAlt ?? string.Empty;
        currentPost.UrlSlug = post.UrlSlug ?? string.Empty;
        currentPost.CategoryId = post.CategoryId;
        currentPost.Content = post.Content ?? string.Empty;
        currentPost.PostedOn = post.PostedOn;
        currentPost.DateOfPublished = post.DateOfPublished;

        var oldPostTags = _dBContext.PostTags.Where(x => x.PostId == post.Id).ToList();
        if (oldPostTags.Any())
        {
            _dBContext.PostTags.RemoveRange(oldPostTags);
        }

        if (tagIds.Any())
        {
            var newPostTags = tagIds
                .Distinct()
                .Select(tagId => new PostTags
                {
                    PostId = post.Id,
                    TagId = tagId
                })
                .ToList();

            _dBContext.PostTags.AddRange(newPostTags);
        }

        _dBContext.SaveChanges();
        return true;
    }
    catch
    {
        return false;
    }
}

public bool RemovePost(Post remove)
{
    try
    {
        var postTags = _dBContext.PostTags
            .Where(pt => pt.PostId == remove.Id)
            .ToList();

        _dBContext.PostTags.RemoveRange(postTags);
```

```
            _dBContext.Posts.Remove(remove);
            _dBContext.SaveChanges();

            return true;
         }
         catch
         {
            return false;
         }
      }
   }
}
```

==========
FILE: Models/ServicesRepository.cs
==========

```
using WebApplicationVetClinic.Entities;
using WebApplicationVetClinic.Data;

namespace WebApplicationVetClinic.Models
{
   public class ServicesRepository
   {
      private readonly ApplicationDbContext _dBContext;
      public ServicesRepository(ApplicationDbContext clinicDBContext)
      {
         _dBContext = clinicDBContext;
      }
      public List<Service> GetServices()
      {
         return _dBContext.Services
            .OrderBy(x => x.Order)
            .ThenBy(x => x.Id)
            .ToList();
      }

      public Service? GetServiceById(int id)
      {
         return _dBContext.Services.FirstOrDefault(x => x.Id == id);
      }

      public bool SaveService(Service service)
      {
         try
         {
            _dBContext.Services.Add(service);
            _dBContext.SaveChanges();
            return true;
         }
         catch
         {
            return false;
         }
      }

      public bool UpdateService(Service service)
      {
         try
         {
            var currentService = _dBContext.Services.FirstOrDefault(x => x.Id == service.Id);

            if (currentService == null)
               return false;

            currentService.Title = service.Title ?? string.Empty;
            currentService.Description = service.Description ?? string.Empty;
            currentService.ImgAlt = service.ImgAlt ?? string.Empty;
            currentService.ImgSrc = service.ImgSrc ?? string.Empty;
            currentService.Order = service.Order;

            _dBContext.SaveChanges();
            return true;
         }
         catch
         {
```

```
            return false;
        }
    }

    public bool DeleteService(Service service)
    {
        try
        {
            _dBContext.Services.Remove(service);
            _dBContext.SaveChanges();
            return true;
        }
        catch
        {
            return false;
        }
    }
  }
}
```

==========
FILE: ViewModels/HomePageViewModel.cs
==========

```
using WebApplicationVetClinic.Entities;

namespace WebApplicationVetClinic.ViewModels
{
  public class HomePageViewModel
  {
    public List<Employee> Employees { get; set; }
    public List<Service> Services { get; set; }
    public List<Post> LatestPosts { get; set; }

     public AppointmentViewModel Appointment { get; set; } = new AppointmentViewModel();
  }
}
```

==========
FILE: ViewModels/AboutPageViewModel.cs
==========

```
using WebApplicationVetClinic.Entities;

namespace WebApplicationVetClinic.ViewModels
{
  public class AboutPageViewModel
  {
    public List<Employee> Employees { get; set; }

  }
}
```

==========
FILE: ViewModels/AppointmentViewModel.cs
==========

```
using System.ComponentModel.DataAnnotations;

namespace WebApplicationVetClinic.ViewModels
{
  public class AppointmentViewModel
  {
    [Required]
    [Display(Name = "Ім'я")]
    public string FirstName { get; set; }

    [Required]
    [Display(Name = "Прізвище")]
    public string LastName { get; set; }

    [Required]
    [EmailAddress]
    [Display(Name = "Email")]
```

```
        public string Email { get; set; }

        [Required]
        [Phone]
        [Display(Name = "Телефон")]
        public string Phone { get; set; }

        [Required(ErrorMessage = "Оберіть дату прийому")]
        [DataType(DataType.DateTime)]
        public DateTime AppointmentDate { get; set; }

        [Required]
        [Display(Name = "Інформація про улюбленця")]
        public string Message { get; set; }
    }
}
```

```
=============================================================================================================
FILE: Views/Shared/_Layout.cshtml
=============================================================================================================

<!DOCTYPE html>
<html class="">
<head>
    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">

    <meta name="description" content="">
    <meta name="author" content="">

    <title>@ViewData["Title"]</title>

    <!-- Standard Favicon -->
    <link rel="icon" type="image/x-icon" href="~/favicon.ico" />

    <!-- For iPhone 4 Retina display: -->
    <link rel="apple-touch-icon-precomposed" sizes="114x114" href="~/images//apple-touch-icon-114x114-precomposed.png">

    <!-- For iPad: -->
    <link rel="apple-touch-icon-precomposed" sizes="72x72" href="~/images//apple-touch-icon-72x72-precomposed.png">

    <!-- For iPhone: -->
    <link rel="apple-touch-icon-precomposed" href="~/images//apple-touch-icon-57x57-precomposed.png">

    <!-- Library - Loader CSS -->
    <link rel="stylesheet" type="text/css" href="~/libraries/loader/loaders.min.css">

    <!-- Library - Google Font Familys -->
    <link href='https://fonts.googleapis.com/css?family=Open+Sans:400,300,300italic,400italic,600,600italic,700,700italic,800,800italic'
rel='stylesheet' type='text/css'>
    <link href='https://fonts.googleapis.com/css?family=Montserrat:400,700' rel='stylesheet' type='text/css'>
    <link href='https://fonts.googleapis.com/css?family=Oxygen:400,300,700' rel='stylesheet' type='text/css'>
    <link href='https://fonts.googleapis.com/css?family=Lato:400,100,100italic,300,300italic,400italic,700,700italic,900,900italic' rel='stylesheet'
type='text/css'>

    <!-- Library - Bootstrap v3.3.5 -->
    <link rel="stylesheet" type="text/css" href="~/libraries/bootstrap/bootstrap.min.css">

    <!-- Font Icons -->
    <link rel="stylesheet" type="text/css" href="~/libraries/fonts/font-awesome.min.css">

    <!-- Library - OWL Carousel V.2.0 beta -->
    <link rel="stylesheet" type="text/css" href="~/libraries/owl-carousel/owl.carousel.css">
    <link rel="stylesheet" type="text/css" href="~/libraries/owl-carousel/owl.theme.css">

    <!-- Library - Animate CSS -->
    <link rel="stylesheet" type="text/css" href="~/libraries/animate/animate.min.css">

    <!-- Library - Price Filter -->
    <link rel="stylesheet" type="text/css" href="~/libraries/price-filter/jquery-ui.min.css">

    <!-- Custom - Common CSS -->
    <link rel="stylesheet" type="text/css" href="~/css/plugins.css">
    <link rel="stylesheet" type="text/css" href="~/css/navigation-menu.css">
```

```
    <!-- Custom - Theme CSS -->
    <link rel="stylesheet" type="text/css" href="~/css/style.css">
    <link rel="stylesheet" type="text/css" href="~/css/shortcodes.css">

</head>
<body data-offset="200" data-spy="scroll" data-target=".ow-navigation">

    <div id="site-loader" class="load-complete">
        <div class="loader">
            <div class="loader-inner ball-clip-rotate">
                <div></div>
            </div>
        </div>
    </div>

    <a id="top"></a>
    <!-- Main Container -->
    <div class="main-container">
        <!-- Header -->
        <header class="header-main">
            <!-- Top Header -->
            <div class="top-header container-fluid no-padding">
                <div class="col-md-4 col-sm-4 col-xs-4 no-padding color-red"></div>
                <div class="col-md-4 col-sm-4 col-xs-4 no-padding color-green"></div>
                <div class="col-md-4 col-sm-4 col-xs-4 no-padding color-yellow"></div>
                <!-- Container -->
                <div class="container">
                    <div class="row">
                        <div class="col-md-8 col-sm-8 col-xs-6">
                            @await Component.InvokeAsync("LeaveMessage")
                        </div>
                        <div class="social col-md-4 col-sm-4 col-xs-6 pull-right">
                            @await Component.InvokeAsync("TopHeader")
                        </div>
                    </div>
                </div>
                <!-- container /- -->
            </div><!-- Top Header /- -->
            <!-- Logo Block -->
            <div class="middle-header container-fluid no-padding">
                <!-- Container -->
                <div class="container">
                    <div class="row">
                        <div class="col-md-5 logo-block" style="display:flex">
                            <a href="/">
                                @await Component.InvokeAsync("Logo")
                            </a>
                            @await Component.InvokeAsync("SiteName")
                        </div>
                        @await Component.InvokeAsync("TopBar")
                    </div>
                </div><!-- Container /- -->
            </div><!-- Logo Block /- -->
            @await Component.InvokeAsync("NavigationMenu");

        </header><!-- Header /- -->
        @RenderBody()

        <!-- Newsletter -->
        <div id="newsletter-section" class="newsletter-section container-fluid no-padding">
            <div class="container">
                <div class="row" style="display: flex; flex-direction: column; justify-content: space-between;">

                    <form class="subscribeOfNewsForm col-12">
                        <h3>Підписатися на розсилку</h3>

                        <div class="input-group mb-2">
                            <input type="text"
                                   class="form-control form-control-name"
                                   placeholder="Як Вас звуть ?">
                        </div>

                        <div class="input-group mb-2">
                            <input type="text"
                                   class="form-control form-control-email"
                                   placeholder="Ваш E-mail">
```

```
            </div>

            <input type="submit"
                   value="Вперед!"
                   class="form-control-subscribe-button" />
        </form>

        <!-- ERROR -->
        <div class="alert alert-danger alert-dismissible"
             role="alert"
             id="subscribe-alert-error"
             style="display:none; margin-top: 10px">
            <strong>Помилка.</strong>
            Не вдалося передплатити.
            <button type="button" class="close" data-dismiss="alert">
                <span>&times;</span>
            </button>
        </div>

        <!-- SUCCESS -->
        <div class="alert alert-success alert-dismissible"
             role="alert"
             id="subscribe-alert-success"
             style="display:none;  margin-top: 10px">
            <strong>Готово!</strong>
            Ви успішно передплатили.
            <button type="button" class="close" data-dismiss="alert">
                <span>&times;</span>
            </button>
        </div>

    </div>

    <script src="~/js/subscribe.js"></script>
  </div>
</div>

<footer class="footer-main">
  <!-- Container -->
  <div class="container">
    <div class="row">
      <div class="col-md-4 col-sm-6">
        <aside class="widget widget-about">
          <h3 class="widget-title">Про нас</h3>
          @await Component.InvokeAsync("FooterBar")

        </aside>
      </div>
      <div class="col-md-4 col-sm-6">
        <aside class="widget widget-links">
          <h3 class="widget-title">
            Час роботи
          </h3>
          @await Component.InvokeAsync("TimeWork")
        </aside>
      </div>
      <div class="col-md-4 col-sm-6">
        <h3 class="widget-title">Ми в соціальних мережах</h3>
        <aside class="widget widget-subscribe">
          @await Component.InvokeAsync("TopHeader")
        </aside>
      </div>
    </div>
  </div><!-- Container /- -->
  <div class="bottom-footer container-fluid no-padding">
    <div class="container">
      <div class="row">
        <ul class="col-md-5 pull-left">
          <li><a href="/Post?category=news">Новини</a></li>
          <li><a href="/post/index?category=services">Послуги</a></li>
        </ul>

        <div class="col-md-7 pull-right">
          <p class="copyright"> &copy; 2026 </p>
        </div>
      </div>
    </div>
```

```
        </div>
      </footer>

   </div><!-- Main Container -->
   <!-- JQuery v1.11.3 -->
   <script src="~/js/jquery.min.js"></script>

   <!-- Library - Modernizer -->
   <script src="~/libraries/modernizr/modernizr.js"></script>

   <!-- Library - Bootstrap v3.3.5 -->
   <script src="~/libraries/bootstrap/bootstrap.min.js"></script><!-- Bootstrap JS File v3.3.5 -->
   <!-- jQuery Easing v1.3 -->
   <script src="~/js/jquery.easing.min.js"></script>

   <!-- Library - jQuery.appear -->
   <script src="~/libraries/appear/jquery.appear.js"></script>

   <!-- Library - OWL Carousel V.2.0 beta -->
   <script src="~/libraries/owl-carousel/owl.carousel.min.js"></script>

   <!-- jQuery For Number Counter -->
   <script src="~/libraries/number/jquery.animateNumber.min.js"></script>

   <!-- Library - Price Filter -->
   <script src="~/libraries/price-filter/jquery-ui.min.js"></script>

   <!-- Library - Theme JS -->
   <script src="~/js/functions.js"></script>
</body>
</html>

================================================================================================
FILE: Views/Home/Index.cshtml
================================================================================================

@using WebApplicationVetClinic.Entities
@using WebApplicationVetClinic.ViewModels
@model WebApplicationVetClinic.ViewModels.HomePageViewModel

@{
   ViewData["Title"] = "Home Page";
}
@{
   var postVm = (HomePageViewModel)Model;
}

<!-- Photo Slider -->
<div id="photo-slider" class="photo-slider container-fluid no-padding">
   <!-- Main Carousel -->
   <div id="main-slider" class="carousel slide carousel-fade col-offset-2" data-ride="carousel">
      <div class="carousel-inner" role="listbox">
         <div class="item active">
            <img src="~/images/slider-image/slider-1.jpg" alt="slider-1" />
            <div class="slider-content">
               <div class="container">
                  <div class="slide-content slide-content-1">
                     <h2 style="line-height: 50px;">Ветеринарна клініка  <span>Пухнастики</span></h2>
                     <h3>Ми дбаємо про ваших милих домашніх улюбленців</h3>
                  </div>
               </div>
            </div>
         </div>
         <div class="item">
            <img src="~/images/slider-image/fon-veterinariia-2.webp" alt="slider-2" />
            <div class="slider-content">
               <div class="container">
                  <div class="slide-content slide-content-2">
                     <h3>Ми дбаємо про ваших милих домашніх улюбленців</h3>
                     <h2 style="line-height: 50px;">Ветеринарна клініка  <span>Пухнастики</span></h2>
                  </div>
               </div>
            </div>
         </div>
      </div>
   </div><!-- Main Carousel /-  -->
</div><!-- Photo Slider /- -->
```

```
<!-- Intro -->
<div id="intro" class="intro container-fluid no-padding">
    <div class="section-padding"></div>
    <!-- Container -->
    <div class="container">
        <!-- Section Header -->
        <div class="section-header">
            <h3>Що ми робимо</h3>
        </div><!-- Section Header /- -->
        <div class="row">
            <div class="col-md-4 col-sm-6">
                <div class="pet-images bgcolor-default">
                    <img src="~/images/intro/intro-1.jpg" alt="intro-1">
                    <div class="content-box">
                        <h3>Ветеринарна клініка <span>Ми поруч, коли вашому улюбленцю потрібна допомога.</span><i><img src="~/images/icon/intro-icon-1.png" alt="Intro Icon" /></i></h3>
                        <p> Ми лікуємо з любов’ю, турбуємось як про власних і повертаємо здоров’я вашим улюбленцям.</p>
                    </div>
                </div>
            </div>
            <div class="col-md-4 col-sm-6">
                <div class="pet-images bgcolor-green">
                    <img src="~/images/intro/intro-2.jpg" alt="intro-2">
                    <div class="content-box">
                        <h3>Ветеринарна клініка <span>Професійна ветеринарія з людським ставленням.</span><i><img src="~/images/icon/intro-icon-2.png" alt="Intro Icon" /></i></h3>
                        <p> Діагностика, лікування та профілактика — сучасна ветеринарна допомога для кожного хвостика.</p>
                    </div>
                </div>
            </div>
            <div class="col-md-4 col-sm-6">
                <div class="pet-images bgcolor-yellow">
                    <img src="~/images/intro/intro-3.jpg" alt="intro-3">
                    <div class="content-box">
                        <h3>Ветеринарна клініка <span>Бо кожен хвіст заслуговує на здорове життя.</span><i><img src="~/images/icon/intro-icon-3.png" alt="Intro Icon" /></i></h3>
                        <p> Здоров’я, турбота та безпека ваших улюбленців — наша щоденна робота.</p>
                    </div>
                </div>
            </div>
        </div>
    </div><!-- Container /- -->
    <div class="section-padding"></div>
</div><!-- Intro /- -->
<!-- Appointment -->
<div id="appointment" class="appointment container-fluid no-padding">
    <div class="section-padding"></div>
    <!-- Container -->
    <div class="container">
        <div class="row">
            <partial name="_AppointmentForm" model="Model.Appointment" />
        </div>
    </div><!-- Container /- -->
    <div class="section-padding"></div>
</div><!-- Appointment /- -->
<!-- Services -->
<div id="services" class="services container-fluid no-padding">
    <div class="section-padding"></div>
    <!-- Container -->
    <div class="container">
        <!-- Section Header -->
        <div class="section-header">
            <h3>
                Послуги, які ми пропонуємо
            </h3>
        </div><!-- Section Header /- -->
        <div class="service-tab col-md-9 no-padding">
            <div class="col-md-4 col-sm-5 no-padding">
                <ul class="nav nav-tabs" role="tablist">
                    @for (int i = 0; i < postVm.Services.Count; i++)
                    {
                        var service = postVm.Services[i];
                        <li role="presentation" class="@(i == 0 ? "active" : "")">
                            <a data-toggle="tab"
                               role="tab"
                               aria-controls="tab-@service.Id"
                               href="#tab-@service.Id"
```

```
                        aria-expanded="@(i == 0 ? "true" : "false")">
                        <i>
                            <img src="@service.ImgSrc" alt="@service.ImgAlt">
                        </i>
                        @service.Title
                    </a>
                </li>
            }
        </ul>
    </div>

    <div class="tab-content col-md-8 col-sm-7 no-padding">
        @for (int i = 0; i < postVm.Services.Count; i++)
        {
            var service = postVm.Services[i];
            <div id="tab-@service.Id"
                class="tab-pane @(i == 0 ? "active" : "")"
                role="tabpanel">
                <div class="tab-box">
                    @service.Description
                </div>
            </div>
        }
    </div>
  </div>
</div><!-- Container /- -->
<div class="section-padding"></div>
</div><!-- Services /- -->
<!-- Callout -->
<div class="callout container-fluid no-padding">
  <!-- Container -->
  <div class="container">
    <div class="callout-content row">
    </div>
  </div><!-- Container /- -->
</div><!-- Callout /- -->
<!-- Why Choose Us -->
<div class="why-choose-us container-fluid no-padding">
  <div class="section-padding"></div>
  <!-- Container -->
  <div class="container">
    <!-- Section Header -->
    <div class="section-header">
      <h3>Чому обирають нас?</h3>
      <p>Ми поєднуємо професіоналізм, турботу та любов до кожного улюбленця</p>
    </div><!-- Section Header /- -->
    <div class="row">
      <div class="col-md-3 col-sm-6">
        <div class="why-choose-content">
          <div class="image-box">
            <img src="~/images/why-choose/why-choose-1.jpg" alt="why choose" />
          </div>
          <div class="why-choose-hover">
            <i><img src="~/images/icon/medal.png" alt="medal" /></i>
            <h3>Досвід і професіоналізм</h3>
            <p>Наші лікарі — кваліфіковані фахівці з багаторічним досвідом, які постійно вдосконалюють свої знання та навички.</p>
          </div>
          <a href="#">Дізнатися більше</a>
        </div>
      </div>

      <div class="col-md-3 col-sm-6">
        <div class="why-choose-content">
          <div class="image-box">
            <img src="~/images/why-choose/why-choose-1.jpg" alt="why choose" />
          </div>
          <div class="why-choose-hover">
            <i><img src="~/images/icon/customer-support.png" alt="Customer Support" /></i>
            <h3>Підтримка 24/7</h3>
            <p>Ми завжди на зв’язку, коли вашому улюбленцю потрібна допомога — у будні, вихідні та святкові дні.</p>
          </div>
          <a href="#">Дізнатися більше</a>
        </div>
      </div>

      <div class="col-md-3 col-sm-6">
        <div class="why-choose-content">
```

```
                <div class="image-box">
                    <img src="~/images/why-choose/why-choose-1.jpg" alt="why choose" />
                </div>
                <div class="why-choose-hover">
                    <i><img src="~/images/icon/home.png" alt="home" /></i>
                    <h3>Індивідуальний підхід</h3>
                    <p>Кожна тварина для нас — унікальна. Ми підбираємо лікування з урахуванням віку, стану та потреб вашого улюбленця.</p>
                </div>
                <a href="#">Дізнатися більше</a>
            </div>
        </div>

        <div class="col-md-3 col-sm-6">
            <div class="why-choose-content">
                <div class="image-box">
                    <img src="~/images/why-choose/why-choose-1.jpg" alt="why choose" />
                </div>
                <div class="why-choose-hover">
                    <i><img src="~/images/icon/dogfeek.png" alt="dogfeek" /></i>
                    <h3>Якість і турбота</h3>
                    <p>Ми використовуємо сучасне обладнання, якісні препарати та дбаємо про комфорт тварин на кожному етапі лікування.</p>
                </div>
                <a href="#">Дізнатися більше</a>
            </div>
        </div>
    </div>
  </div><!-- Container /- -->
  <div class="section-padding"></div>
</div><!-- Why Choose Us /- -->
<!-- Testimonial -->
<div class="testimonial container-fluid no-padding">
  <div class="section-padding"></div>
  <!-- Container -->
  <!-- Container -->
  <div class="container">
    <div class="row">
      <div class="col-md-7 col-sm-12">
        <div class="section-header">
          <h3>Відгуки наших клієнтів:</h3>
        </div>
        <div id="main-carousel" class="main-carousel-reviews carousel slide carousel-fade col-offset-2" data-ride="carousel">
          <div class="carousel-inner" role="listbox">

          </div>
          <a class="left carousel-control" href="#main-carousel" role="button" data-slide="prev">
            <i class="fa fa-angle-left"></i>
            <span class="sr-only">Previous</span>
          </a>
          <a class="right carousel-control" href="#main-carousel" role="button" data-slide="next">
            <i class="fa fa-angle-right"></i>
            <span class="sr-only">Next</span>
          </a>
        </div><!-- Main Carousel /-  -->
        <script src="~/js/reviews.js"></script>
      </div>
      <div class="col-md-5 col-sm-12">

      </div>
    </div>
  </div><!-- Container /- -->
  <div class="section-padding"></div>
</div><!-- Testimonial /- -->

<!-- Team -->
<div class="team container-fluid no-padding">
  <div class="section-padding"></div>
  <!-- Container -->
  <div class="container">
    <div class="section-header">
      <h3>Наша команда</h3>
      <p>Ми робимо світ краще...</p>
    </div>
    <div class="row">
      <div class="team-thumb">
        @foreach (Employee oneEmp in postVm.Employees)
```

```
            {
                <div class="col-md-12">
                    <div class="team-member">
                        <div class="team-image-box">
                            <a asp-controller="home" asp-action="employeer" asp-route-id="@oneEmp.Id">
                                <img src="@oneEmp.ImgSrc" alt="@oneEmp.ImgAlt">
                                <div class="team-content">
                                    <h3 style="line-height:12px; font-size:12px">
                                        @oneEmp.Fio
                                        <span>@oneEmp.Speciality</span>
                                    </h3>
                                </div>
                            </a>
                        </div>
                    </div>
                </div>
            }
        </div>
    </div>
    </div><!-- Container /- -->
    <div class="section-padding"></div>
</div><!-- Team /- -->
<!-- Counter -->
<div class="counter container-fluid no-padding">
    <!-- Container -->
    <div class="container">
        <div class="row">
            <div class="col-md-3 col-sm-6 col-xs-12">
                <div class="counter-box">
                    <h3><img src="~/images/counter/counter-1.png" alt="counter-1"><span class="count" id="statistics_count-1" data-
statistics_percent="3540">  </span></h3>
                    <p>Виконаних послуг</p>
                </div>
            </div>
            <div class="col-md-3 col-sm-6 col-xs-12">
                <div class="counter-box">
                    <h3>
                        <img src="~/images/counter/counter-2.png" alt="counter-2">
                        <span class="count" id="statistics_count-2" data-statistics_percent="1234"> </span>
                    </h3>
                    <p>Наших щасливих клієнтівв</p>
                </div>
            </div>
            <div class="col-md-3 col-sm-6 col-xs-12">
                <div class="counter-box">
                    <h3>
                        <img src="~/images/counter/counter-3.png" alt="counter-3">
                        <span class="count" id="statistics_count-3" data-statistics_percent="2200"> </span>
                    </h3>
                    <p>Талановиті працівники</p>
                </div>
            </div>
            <div class="col-md-3 col-sm-6 col-xs-12">
                <div class="counter-box">
                    <h3>
                        <img src="~/images/counter/counter-4.png" alt="counter-4">
                        <span class="count" id="statistics_count-4" data-statistics_percent="1234"> </span>
                    </h3>
                    <p>Кількість відділень</p>
                </div>
            </div>
        </div>
    </div><!-- Container /- -->
</div><!-- Counter /- -->
<!-- Offer -->
<div class="offer container-fluid no-padding">
    <div class="offer-shape">
        <svg width="100%" height="100%">
            <clipPath id="clipPolygon1" clipPathUnits="objectBoundingBox">
                <polygon points="0 0, 32 100, 100 100, 100 0"></polygon>
            </clipPath>
        </svg>
    </div>

    <!-- Container -->
    <div class="container">
        <div class="row">
```

```
            <div class="col-md-5 col-sm-6">
                <div class="smart-price">
                    <h3>Щось особливе для ваших домашніх улюбленців</h3>
                    <h2>Скидки на лікування</h2>
                </div>
            </div>
            <div class="col-md-4 col-sm-6 pull-right offer-box">
                <h2>30%</h2><h3>Скидки<span>в день народження Вашого любимця</span></h3>
            </div>
        </div>
    </div><!-- Container /- -->
</div><!-- Offer / -->
<!-- Blog -->
<div id="blog-section" class="blog-section container-fluid no-padding">
    <div class="section-padding"></div>
    <!-- Container -->
    <div class="container">
        <!-- Section Header -->
        <div class="section-header">
            <h3>Останні новини</h3>
        </div><!-- Section Header /- -->
        <div class="row">
            @foreach (var post in postVm.LatestPosts)
            {
                <div class="col-md-4 col-sm-6">
                    <article>
                        <div class="entry-cover">
                            <a asp-controller="Post"
                              asp-action="SinglePost"
                              asp-route-slug="@post.UrlSlug">
                               <img src="@post.ImgSrc" alt="@post.Title" class="card-img" />
                            </a>

                            <a asp-controller="Post"
                              asp-action="SinglePost"
                              asp-route-slug="@post.UrlSlug"
                              class="read-more">
                               Read More
                            </a>
                        </div>

                        <!-- Post Content -->
                        <div class="post-content">
                            <div class="post-meta">
                                <div class="post-date">
                                    <span>@post.DateOfPublished.ToString("MMM")</span>
                                    <span>@post.DateOfPublished.Day</span>
                                </div>

                                <div class="post-comment">
                                    <i>
                                        <img src="~/images/icon/comment.png" alt="Comment" />
                                    </i>
                                    <a href="#">
                                        @post.Comments.Count(c => c.IsValid)
                                    </a>
                                </div>
                            </div>

                            <h3 class="entry-title">
                                <a asp-controller="Post"
                                  asp-action="SinglePost"
                                  asp-route-slug="@post.UrlSlug">
                                   @post.Title
                                </a>
                            </h3>

                            <div class="entry-content">
                                <p>@post.Slogan</p>
                            </div>
                        </div><!-- Post Content /- -->
                    </article>
                </div>
            }
        </div>
    </div><!-- Container -->
    <div class="section-padding"></div>
```

```
</div><!-- Blog /- -->


================================================================================================================
FILE: Views/About/Index.cshtml
================================================================================================================

@using WebApplicationVetClinic.Entities
@using WebApplicationVetClinic.ViewModels
@{
   var postVm = (AboutPageViewModel)Model;
}
<!-- Why Choose Us -->
<div class="why-choose-us container-fluid no-padding">
   <div class="section-padding"></div>
   <!-- Container -->
   <div class="container">
      <!-- Section Header -->
      <div class="section-header">
         <h3>Чому обирають нас?</h3>
         <p>Ми поєднуємо професіоналізм, турботу та любов до кожного улюбленця</p>
      </div><!-- Section Header /- -->
      <div class="row">
         <div class="col-md-3 col-sm-6">
            <div class="why-choose-content">
               <div class="image-box">
                  <img src="~/images/why-choose/why-choose-1.jpg" alt="why choose" />
               </div>
               <div class="why-choose-hover">
                  <i><img src="~/images/icon/medal.png" alt="medal" /></i>
                  <h3>Досвід і професіоналізм</h3>
                  <p>Наші лікарі — кваліфіковані фахівці з багаторічним досвідом, які постійно вдосконалюють свої знання та навички.</p>
               </div>
               <a href="#">Дізнатися більше</a>
            </div>
         </div>

         <div class="col-md-3 col-sm-6">
            <div class="why-choose-content">
               <div class="image-box">
                  <img src="~/images/why-choose/why-choose-1.jpg" alt="why choose" />
               </div>
               <div class="why-choose-hover">
                  <i><img src="~/images/icon/customer-support.png" alt="Customer Support" /></i>
                  <h3>Підтримка 24/7</h3>
                  <p>Ми завжди на зв’язку, коли вашому улюбленцю потрібна допомога — у будні, вихідні та святкові дні.</p>
               </div>
               <a href="#">Дізнатися більше</a>
            </div>
         </div>

         <div class="col-md-3 col-sm-6">
            <div class="why-choose-content">
               <div class="image-box">
                  <img src="~/images/why-choose/why-choose-1.jpg" alt="why choose" />
               </div>
               <div class="why-choose-hover">
                  <i><img src="~/images/icon/home.png" alt="home" /></i>
                  <h3>Індивідуальний підхід</h3>
                  <p>Кожна тварина для нас — унікальна. Ми підбираємо лікування з урахуванням віку, стану та потреб вашого
улюбленця.</p>
               </div>
               <a href="#">Дізнатися більше</a>
            </div>
         </div>

         <div class="col-md-3 col-sm-6">
            <div class="why-choose-content">
               <div class="image-box">
                  <img src="~/images/why-choose/why-choose-1.jpg" alt="why choose" />
               </div>
               <div class="why-choose-hover">
                  <i><img src="~/images/icon/dogfeek.png" alt="dogfeek" /></i>
                  <h3>Якість і турбота</h3>
                  <p>Ми використовуємо сучасне обладнання, якісні препарати та дбаємо про комфорт тварин на кожному етапі
лікування.</p>
               </div>
```

```
                <a href="#">Дізнатися більше</a>
              </div>
            </div>
          </div>
        </div><!-- Container /- -->
        <div class="section-padding"></div>
    </div><!-- Why Choose Us /- -->
    <!-- Callout -->
    <div class="callout container-fluid no-padding">
        <!-- Container -->
        <div class="container">
            <div class="callout-content row">

            </div>
        </div><!-- Container /- -->
    </div><!-- Callout /- -->



    <!-- Team -->
    <div class="team container-fluid no-padding">
        <div class="section-padding"></div>
        <!-- Container -->
        <div class="container">
            <div class="section-header">
                <h3>Наша команда</h3>
                <p>Ми робимо світ краще...</p>
            </div>
            <div class="row">
                <div class="team-thumb">
                    @foreach (Employee oneEmp in postVm.Employees)
                    {
                        <div class="col-md-12">
                            <div class="team-member">
                                <div class="team-image-box">
                                    <a asp-controller="home" asp-action="employeer" asp-route-id="@oneEmp.Id">
                                        <img src="@oneEmp.ImgSrc" alt="@oneEmp.ImgAlt">
                                        <div class="team-content">
                                            <h3 style="line-height:12px; font-size:12px">
                                                @oneEmp.Fio
                                                <span>@oneEmp.Speciality</span>
                                            </h3>
                                        </div>
                                    </a>
                                </div>
                            </div>
                        </div>
                    }
                </div>
            </div>
        </div><!-- Container /- -->
        <div class="section-padding"></div>
    </div><!-- Team /- -->

    <!-- Testimonial -->
    <div class="testimonial container-fluid no-padding">
        <div class="section-padding"></div>
        <!-- Container -->
        <!-- Container -->
        <div class="container">
            <div class="row">
                <div class="col-md-7 col-sm-12">
                    <div class="section-header">
                        <h3>Відгуки наших клієнтів:</h3>
                    </div>
                    <div id="main-carousel" class="main-carousel-reviews carousel slide carousel-fade col-offset-2" data-ride="carousel">
                        <div class="carousel-inner" role="listbox">
                        </div>
                        <a class="left carousel-control" href="#main-carousel" role="button" data-slide="prev">
                            <i class="fa fa-angle-left"></i>
                            <span class="sr-only">Previous</span>
                        </a>
                        <a class="right carousel-control" href="#main-carousel" role="button" data-slide="next">
                            <i class="fa fa-angle-right"></i>
                            <span class="sr-only">Next</span>
                        </a>
                    </div><!-- Main Carousel /-  -->
```

```
            <script src="~/js/reviews.js"></script>
        </div>
        <div class="col-md-5 col-sm-12">
        </div>
      </div>
   </div><!-- Container /- -->
   <div class="section-padding"></div>
</div><!-- Testimonial /- -->


================================================================================================
FILE: Views/About/ContactUs.cshtml
================================================================================================

@model WebApplicationVetClinic.Dto.GuestMessageDTO
@{
}
<!-- Page Banner -->
<div class="container-fluid no-padding page-banner">
   <!-- Container -->
   <div class="container">
      <!-- Banner Heading -->
      <div class="banner-heading">
         <h3>Зворотній зв'язок</h3>
         <ol class="breadcrumb">
            <li><a href="/">Головна</a></li>
            <li class="active">Зворотній зв'язок</li>
         </ol>
      </div><!-- Banner Heading /- -->
   </div><!-- Container /- -->
</div><!-- Page Banner /- -->
<!-- Contact Form -->
<div class="container-fluid no-padding contact-form">
   <div class="section-padding"></div>
   <!-- Container -->
   <div class="container">
      <div class="section-title">
         <h3>Зв'язатися з нами</h3>
         <p>Ваша думка дуже важлива для нас</p>
      </div>
      <div class="row">
         <form id="contact-form" method="post" asp-action="getGuestMessage" asp-controller="About">
            <div asp-validation-summary="All" id="alert-msg" class="alert-msg"></div>
            <div class="form-group col-md-4 col-sm-4 col-xs-12">
               <input type="text" asp-for="Name" id="first-name" name="Name" class="form-control" placeholder="Ім'я">
            </div>
            <div class="form-group col-md-4 col-sm-4 col-xs-12">
               <input type="text" asp-for="Surname" id="last-name" name="Surname" class="form-control" placeholder="Прізвище">
            </div>
            <div class="form-group col-md-6 col-sm-6 col-xs-12">
               <input type="email" asp-for="Email" id="input_email" name="Email" class="form-control" placeholder="E-Mail">
            </div>
            <div class="form-group col-md-6 col-sm-6 col-xs-12">
               <input type="text" asp-for="Subject" id="input_subject" name="Subject" class="form-control" placeholder="Тема повідомлення">
            </div>
            <div class="form-group col-md-12 col-sm-12 col-xs-12">
               <textarea rows="7" asp-for="Message" id="textarea_message" name="Message" class="form-control" placeholder="Ваше
повідомлення"></textarea>
            </div>
            <button type="submit" id="btn_submit" class="btn-submit">Надіслати</button>
         </form><!-- /.contact-form -->
      </div>
   </div>
   <div class="section-padding"></div>
</div><!-- Contact Form /- -->


================================================================================================
FILE: Views/Admin/Index.cshtml
================================================================================================

@using WebApplicationVetClinic.ViewModels
@model DashboardViewModel;


<div class="app-main flex-column flex-row-fluid" id="kt_app_main">
   <div class="d-flex flex-column flex-column-fluid">
```

```
    <div id="kt_app_content" class="app-content flex-column-fluid">

      @if (Model.UrgentAppointments.Any())
      {
        <div class="card mb-5 border border-danger">
          <div class="card-header">
            <h3 class="card-title text-danger">Термінові зустрічі</h3>
          </div>
          <div class="card-body">
            @await Html.PartialAsync("_AppointmentsTable", Model.UrgentAppointments)
          </div>
        </div>
      }

      <div class="card mb-5">
        <div class="card-header">
          <h3 class="card-title">Сьогоднішні зустрічі</h3>
        </div>
        <div class="card-body">
          @await Html.PartialAsync("_AppointmentsTable", Model.TodayAppointments)
        </div>
      </div>

      <div class="card mb-5">
        <div class="card-header">
          <h3 class="card-title">Найближчі зустрічі</h3>
        </div>
        <div class="card-body">
          @await Html.PartialAsync("_AppointmentsTable", Model.UpcomingAppointments)
        </div>
      </div>

    </div>
  </div>
</div>

================================================================================================
FILE: Views/Admin/Posts.cshtml
================================================================================================

@model IEnumerable<WebApplicationVetClinic.Entities.Post>

<div class="app-main flex-column flex-row-fluid" id="kt_app_main">
  <div class="d-flex flex-column flex-column-fluid">
    <div id="kt_app_content" class="app-content flex-column-fluid">

      <div class="card mb-5 mb-xl-8">
        <div class="card-header border-0 pt-5 d-flex justify-content-between align-items-center">
          <h3 class="card-title align-items-start flex-column">
            <span class="card-label fw-bold fs-3 mb-1">Пости</span>
          </h3>

          <a asp-controller="Admin" asp-action="CreatePost" class="btn btn-primary">
            Create Post
          </a>
        </div>

        <div class="card-body py-3">
          <div class="table-responsive">
            <table class="table table-row-bordered table-row-gray-100 align-middle gs-0 gy-3">
              <thead>
                <tr class="fw-bold text-muted">
                  <th>Id</th>
                  <th>Image</th>

                  <th>Title</th>
                  <th>Category</th>
                  <th>Published</th>
                  <th>Date</th>
                  <th class="text-end">Actions</th>
                </tr>
              </thead>
              <tbody>
                @foreach (var post in Model)
                {
                  <tr>
                    <td>@post.Id</td>
```

```
                        <td>
                            @if (!string.IsNullOrWhiteSpace(post.ImgSrc))
                            {
                                <img src="@post.ImgSrc" alt="@post.ImgAlt" style="width:60px;height:60px;object-fit:cover;" />
                            }
                        </td>
                        <td>@post.Title</td>
                        <td>@post.Category?.Title</td>
                        <td>@(post.PostedOn ? "Yes" : "No")</td>
                        <td>@post.DateOfPublished.ToString("dd.MM.yyyy HH:mm")</td>
                        <td class="text-end">
                            <a asp-controller="Admin"
                               asp-action="EditPost"
                               asp-route-id="@post.Id"
                               class="btn btn-sm btn-primary me-2">
                                Edit
                            </a>

                            <form asp-controller="Admin"
                                  asp-action="DeletePost"
                                  method="post"
                                  style="display:inline;">
                                <input type="hidden" name="id" value="@post.Id" />
                                <button type="submit"
                                        class="btn btn-sm btn-danger"
                                        onclick="return confirm('Delete this post?');">
                                    Delete
                                </button>
                            </form>
                        </td>
                    </tr>
                }
            </tbody>
        </table>
    </div>
</div>
</div>

</div>
</div>
</div>
```

```
================================================================================================================
FILE: Views/Post/SinglePost.cshtml
================================================================================================================

@using WebApplicationVetClinic.Entities;
@{

    var onePost = Model as Post;
}

<link href="~/css/comments.css" rel="stylesheet" />
<div class="container-fluid no-padding page-banner">
    <div class="container">
        <div class="banner-heading">
            <h3>@onePost.Title</h3>
            <h5 class="onePost-Id hidden">@onePost.Id</h5>
            <ol class="breadcrumb">
                <li><a href="/">Головна</a></li>
                <li><a href="/Post/index">Всі Пости</a></li>
                <li class="active">Перегляд поста</li>
            </ol>
        </div>
    </div>
</div>
<div class="container-fluid no-padding blog-list">
    <div class="section-padding"></div>
    <div class="container">
        <div class="row">
            <div class="col-md-12 col-sm-12 col-xs-12 blog-area single-post">
                <div class="section-title">
                    <h3>@onePost.Title</h3>
                    <p>@onePost.Slogan</p>
                </div>
                <article>
                    <div class="entry-cover">
```

```
                <img src="@onePost.ImgSrc" alt="@onePost.ImgAlt" />
            </div>
            <div class="post-content">
                <div class="post-meta">
                    <div class="post-date">
                        <span>@onePost.DateOfPublished.ToString("MMM")</span>
                        <span>@onePost.DateOfPublished.Day</span>
                    </div>
                </div>
                <h3 class="entry-title">@onePost.Slogan</h3>
                <div class="entry-content">
                    @onePost.Content
                </div>
            </div>
        </article>
        <div class="comment-section">
            <h3 class="comment-container-counter">Комментарі<span>()</span></h3>
            <div class="comments comments-container">

            </div>

            <form class="leave-comment leave-comment-form border border-primary" id="leave-comment-form" style="width:100%; margin-top:
80px">
                <h3 class="text-center">Залиште комментар</h3>
                <div class="alert alert-danger alert-dismissible hide" role="alert">
                    <strong>Виникла помилка, перевірте правильність заповнення поля</strong>
                </div>
                <div class="alert alert-success alert-dismissible hide" role="alert">
                    <strong>Спасибо!</strong> Ваш коментар був відправлений. Він буде відображатися після перевірки адміністратором сайту
                    <button type="button" class="close" data-dismiss="alert" aria-label="Close">
                        <span aria-hidden="true">&times;</span>
                    </button>
                </div>
                <div class="row">
                    <input type="hidden" name="Post_Id" value="@onePost.Id" />
                    <div class="form-group col-md-6 col-sm-6 col-xs-12">
                        <input type="text" id="name" class="form-control form-control-name" name="VisitorName" placeholder="Your Name*"
required>
                    </div>
                    <div class="form-group col-md-6 col-sm-6 col-xs-12">
                        <input type="email" id="email" class="form-control form-control-email" name="VisitorEmail" placeholder="Your E-Mail*"
required>
                    </div>
                    <div class="form-group col-md-12 col-sm-12 col-xs-12">
                        <textarea rows="7" class="form-control form-control-message" name="Message" placeholder="Message"></textarea>
                    </div>
                </div>
                <input type="button" id="saveComment" value="Комментувати">
            </form>

        </div>
    </div>
  </div>
 </div>
 <div class="section-padding"></div>
</div>

<script src="~/js/comments.js"></script>
```