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

Кафедра інформатики та програмної інженерії
(повна назва кафедри)

«До захисту допущено»
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2023 р.

Дипломний проєкт

на здобуття ступеня бакалавра

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

спеціальності «121 Інженерія програмного забезпечення»

на тему: Веб-застосунок з надання та отримання послуг для власників домашніх тварин

Виконав студент IV курсу, групи ІТ-92
(шифр групи)

Гриник Валентина Олександрівна _____
(прізвище, ім'я, по батькові) (підпис)

Керівник доцент, к.т.н., доц., Крамар Ю. М. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант з графічної документації ст.викл. Вітковська І.І. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент доцент, к.т.н., доц., Ткач М.М _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ –2023

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

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 Інженерія програмного забезпечення

Освітньо-професійна програма – Інженерія програмного забезпечення інформаційних систем

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2023 р.

ЗАВДАННЯ
на дипломний проєкт студенту

Гриник Валентині Олександрівні
(прізвище, ім'я, по батькові)

1. Тема проєкту Веб-застосунок цифрової платформи надання та отримання послуг для власників домашніх тварин

керівник проєкту Крамар Юлія Михайлівна, к.т.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «31»квітня 2023 р. №2101-с

2. Термін подання студентом проєкту « 17 » червня 2023 року

3. Вихідні дані до проєкту: технічне завдання

4. Зміст пояснювальної записки
1) Аналіз вимог до програмного забезпечення: загальні положення, основні визначення та терміни, опис предметного середовища, огляд ринку програмних продуктів, постановка задачі.
2) Моделювання та конструювання програмного забезпечення: опис структури бази даних, опис архітектури застосунку та його моделювання.
3) Аналіз якості та тестування програмного забезпечення.
4) Впровадження та супровід програмного забезпечення: розгортання та підтримка програмного забезпечення.

5. Перелік графічного матеріалу

1) Схема структурна варіантів використань _____

3) Схема бази даних _____

6) Креслення вигляду екранних форм _____

6. Консультанти розділів проєкту

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

7. Дата видачі завдання «10» березня 2023 року _____

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Вивчення рекомендованої літератури	10.03.2023	
2	Розробка технічного завдання	03.03.2023	
3	Аналіз існуючих методів розв’язання задачі	19.03.2023	
4	Проектування структури та компонентів програмного забезпечення	30.03.2023	
5	Розробка програмного забезпечення	05.04.2023	
6	Тестування програмного забезпечення	10.04.2023	
7	Написання матеріалів текстової частини розробки	14.04.2023	
8	Написання матеріалів графічної частини розробки	20.04.2023	
9	Оформлення технічної документації проєкту	29.04.2023	
10	Подання ДП на попередній захист	02.06.2023	
11	Подання ДП рецензенту	12.06.2023	
12	Подання ДП на основний захист	17.06.2023	

Студент _____ Валентина ГРИНИК

(підпис) (ініціали, прізвище)

Керівник _____ Юлія КРАМАР

(підпис) (ініціали, прізвище)

АНОТАЦІЯ

Пояснювальна записка дипломного проєкту складається з чотирьох розділів, містить 58 таблиць, 35 рисунків та 9 джерел – загалом 79 сторінок.

Дипломний проєкт присвячений розробці веб-застосунку з надання та отримання послуг для власників домашніх тварин, котрий використовується для легкого та швидкого отримання та надання послуг в сфері догляду за домашніми тваринами та призначений для того, щоб власники домашніх тварин могли у віддаленому форматі отримати потрібні їм послуги максимально швидко і зручно, а виконавці цих послуг могли знайти собі клієнтів.

Метою розробки є надання користувачам можливості легко та швидко отримувати та надавати послуги в сфері догляду за домашніми тваринами шляхом створення заявок на отримання чи надання послуг, а також можливості підтримувати зв’язок із замовником чи виконавцем послуг.

Об'єкт дослідження:

Предмет дослідження: веб-застосунок з надання та отримання послуг для власників домашніх тварин.

У першому розділі "Аналіз вимог до програмного забезпечення" проведено детальний опис предметної області, пов'язаної з розробкою, і визначено основні терміни та поняття, використовувані у проєкті. Також були проаналізовані наявні технічні рішення, засоби розробки та програмні продукти, і обрано найбільш ефективні і корисні для даного проекту.

У другому розділі "Моделювання та конструювання програмного забезпечення" проведено детальний аналіз і дослідження різних аспектів програмного забезпечення. Розглянуто моделювання та аналіз програмного забезпечення, архітектуру програмного забезпечення, принципи роботи та конструювання програмного забезпечення.

У третьому розділі "Аналіз якості та тестування програмного забезпечення" було проведено аналіз якості та тестування веб-застосунку з надання та отримання послуг для власників домашніх тварин.

У четвертому розділі "Впровадження та супровід програмного забезпечення" описано процес розгортання та підтримки програмного забезпечення.

КЛЮЧОВІ СЛОВА: ВЕБ-ЗАСТОСУНОК, БАЗА ДАНИХ, ANDROID, ПОСТ.

ABSTRACT

The explanatory note of the diploma project consists of four chapters, contains 58 tables, 35 figures and 9 sources - a total of 79 pages.

The diploma project is dedicated to the development of a web application for providing and receiving services for pet owners, which is used to easily and quickly receive and provide services in the field of pet care and is designed to enable pet owners to receive the services they need as quickly and conveniently as possible in a remote format, and for the providers of these services to find clients.

The purpose of the development is to provide users with the ability to easily and quickly receive and provide services in the field of pet care by creating applications for the receipt or provision of services, as well as the ability to maintain communication with the customer or service provider.

Object of research:

The subject of the study: a web application for providing and receiving services for pet owners.

In the first section, "Software Requirements Analysis," a detailed description of the development-related subject area was made and the main terms and concepts used in the project were defined. The available technical solutions, development tools, and software products were also analyzed, and the most effective and useful ones for this project were selected.

The second chapter, "Software Modeling and Design," provides a detailed analysis and study of various aspects of software. The chapter covers software modeling and analysis, software architecture, principles of software operation and design.

The third chapter, "Software Quality Analysis and Testing," analyzes the quality and testing of the web application for providing and receiving services for pet owners.

The fourth section, "Implementation and Maintenance of the Software", describes the process of software deployment and support.

KEYWORDS: WEB APPLICATION, DATABASE, ANDROID, POST.

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	КПІ.ІТ-9208.045440.01.91	Технічне завдання	20	
3	А4	КПІ.ІТ-9208.045440.02.81	Пояснювальна записка	79	
4	А4	КПІ.ІТ-9208.045440.03.12	Текст програми	18	
5	А4	КПІ.ІТ-9208.045440.04.51	Програма та методика тестування	6	
6	А4	КПІ.ІТ-9208.045440.05.34	Керівництво користувача	19	
7	А3	КПІ.ІТ-9208.045440.06.99	Схема структурна варіантів	1	
			використань		
9	А3	КПІ.ІТ-9208.045440.07.99	Схема бази даних	1	
11	А3	КПІ.ІТ-9208.045440.08.99	Креслення вигляду екранних форм	1	
12	А4		Звіт Unicheck	1	

Змін.	Арк.	№ докум.	Підп.	Дата	КПІ.ІТ-9208.045440.00.90		
Розроб.		Гриник В.О..			Відомість дипломного проєкту	Літ.	Аркуш 1 / Аркушів
Перевір.		Крамар Ю.М.					
Н.контр.		Вітковська І.І.				КПІ ім. Ігоря Сікорського ФІОТ каф. ІПІ гр. ІТ-92	
Затв.		Жаріков Е.В.					

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”

Завідувач кафедри

_____ Едуард ЖАРІКОВ

“___” _____ 2023 р.

ВЕБ-ЗАСТОСУНОК З НАДАННЯ ТА ОТРИМАННЯ ПОСЛУГ ДЛЯ ВЛАСНИКІВ ДОМАШНІХ ТВАРИН

Технічне завдання

КПІ.ІТ-9208.045440.01.91

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Юлія КРАМАР

Нормоконтроль:

_____ Ірина ВІТКОВСЬКА

Виконавець:

_____ Валентина ГРИНИК

Київ – 2023

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: ВЕБ-ЗАСТОСУНОК З НАДАННЯ ТА ОТРИМАННЯ ПОСЛУГ ДЛЯ ВЛАСНИКІВ ДОМАШНІХ ТВАРИН.

Галузь застосування:

Наведене технічне завдання поширюється на розробку веб-застосунку з надання та отримання послуг для власників домашніх тварин «PawsitivelyCare», котрий використовується для легкого та швидкого отримання та надання послуг в сфері догляду за домашніми тваринами та призначений для того, щоб власники домашніх тварин могли у віддаленому форматі отримати потрібні їм послуги максимально швидко і зручно, а виконавці цих послуг могли знайти собі клієнтів.

2 ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки веб-застосунку з надання та отримання послуг для власників домашніх тварин є завдання на дипломне проєктування, затверджене кафедрою інформатики та програмної інженерії Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

3 ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для полегшенню отримання послуг у сфері догляду за домашніми тваринами.

Метою розробки є надання користувачам можливості легко та швидко отримувати та надавати послуги в сфері догляду за домашніми тваринами шляхом створення заявок на отримання чи надання послуг, а також можливості підтримувати зв’язок із замовником чи виконавцем послуг.

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вимоги до функціональних характеристик

Програмне забезпечення повинно забезпечувати виконання наступних основних функції:

4.1.1 Користувацького інтерфейсу

– відкриття сторінки входу до системи при першому відкритті веб-застосунку – рисунок 4.1. Щоб увійти до системи, потрібно ввести пошту та пароль у відповідні поля та натиснути на кнопку входу. Щоб перейти до сторінки реєстрації, потрібно натиснути на кнопку реєстрації – рисунок 4.2. Щоб зареєструватися у системі, потрібно ввести потрібні дані у відповідні поля і натиснути кнопку реєстрації;

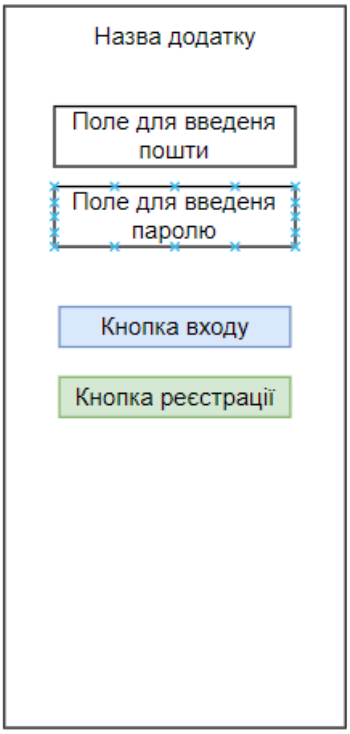


Рисунок 4.1 – Сторінка входу до системи

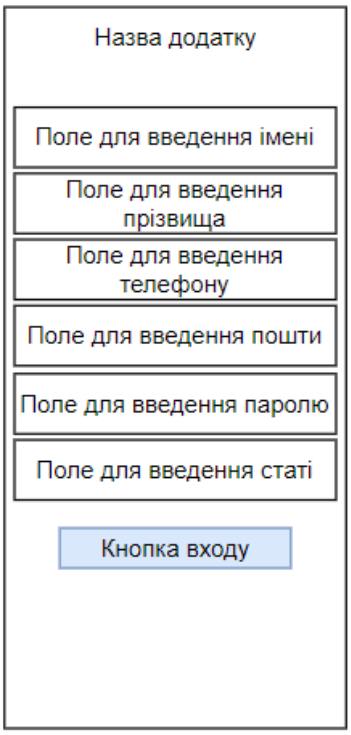


Рисунок 4.2 – Сторінка реєстрації у системі

– виведення повідомлення про успіх при успішній реєстрації у системі. Виведення повідомлення про помилку при неуспішній реєстрації у системі. Виведення повідомлення про помилку при неуспішній спробі увійти до акаунту користувача – рисунок 4.3;

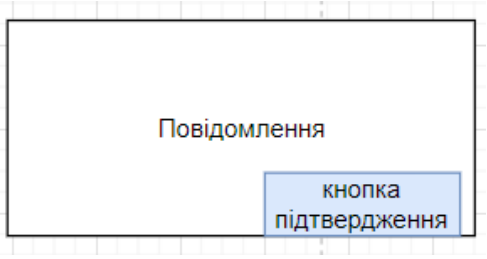


Рисунок 4.3 – Шаблон повідомлення

– відкриття сторінки з постами від інших користувачів при успішному вході до веб-застосунку – рисунок 4.4;

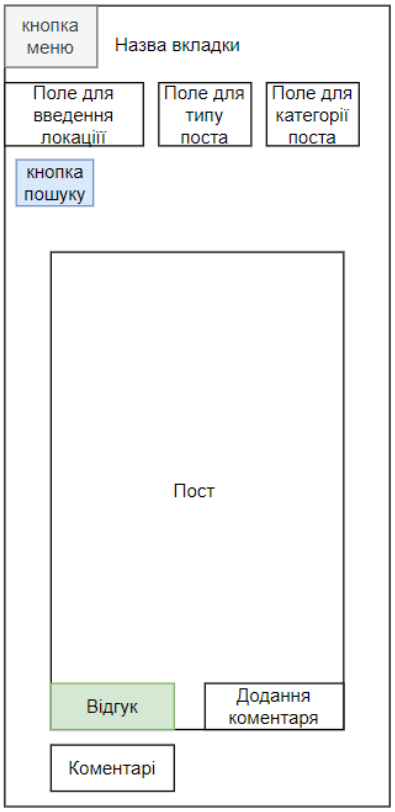


Рисунок 4.4 – Сторінка з постами від інших користувачів

– відкриття меню при свайпі вліво чи натисканні на кнопку меню – рисунок 4.5. Щоб перейти на відповідну сторінку, потрібно натиснути на неї;

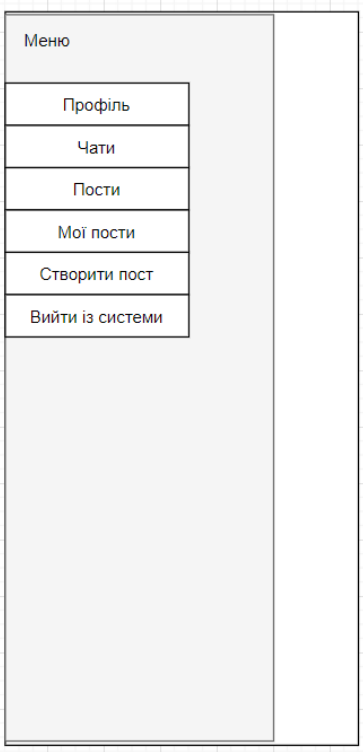


Рисунок 4.5 – Меню додатку

– відкриття сторінки с даними користувача при натисканні кнопки «Профіль» у меню додатку – рисунок 4.6. Щоб відредагувати дані профілю, потрібно натиснути на кнопку редагування профілю;

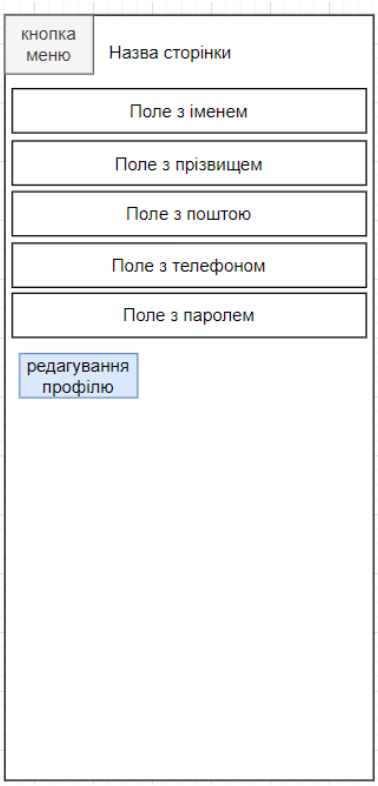


Рисунок 4.6 – Сторінка з даними користувача

– відкриття сторінки с чатами користувача при натисканні кнопки «Чати» у меню додатку – рисунок 4.7. Відкриття чату після натискання на нього – рисунок 4.8. Для написання повідомлення у чат потрібно заповнити текстом поле знизу вікна чату та надіслати його при натисканні на кнопку надсилання повідомлення. Щоб дізнатися деталі поста, до якого був створений чат, треба натиснути на кнопку деталей поста у вікні чату;

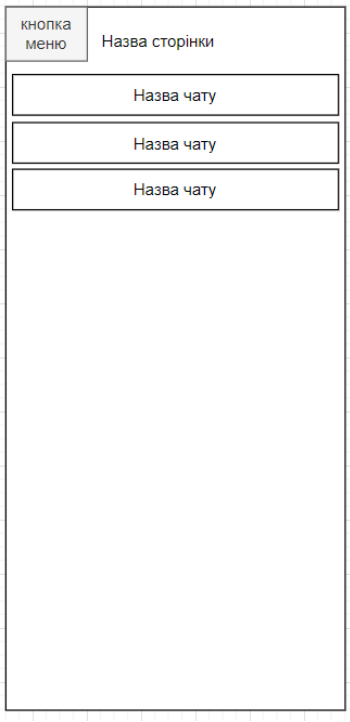


Рисунок 4.7 – Сторінка з чатами

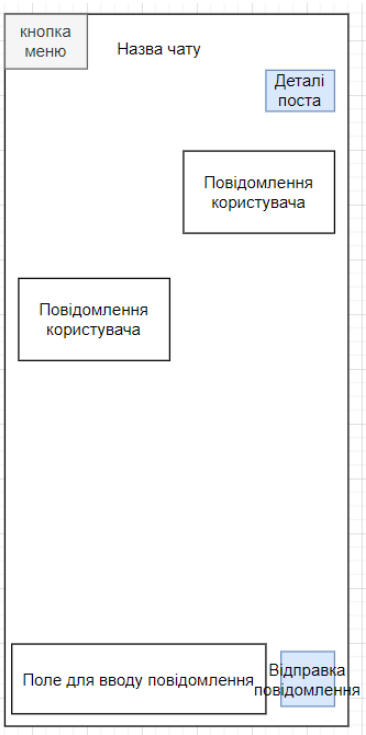


Рисунок 4.8 – Вікно чату

– відкриття сторінки с постами від інших користувачів при натисканні кнопки «Пости» у меню додатку – рисунок 4.9. Щоб відфільтрувати пости за локацією, типом та категорією поста, потрібно обрати відповідні дані зверху вікна. Щоб відгукнутися на пост, потрібно

натиснути на кнопку відгука. Щоб переглянути коментарі під постом, потрібно натиснути на кнопку коментарів під постом. Щоб написати коментар під постом, потрібно натиснути на кнопку додавання коментаря – рисунок 4.10;

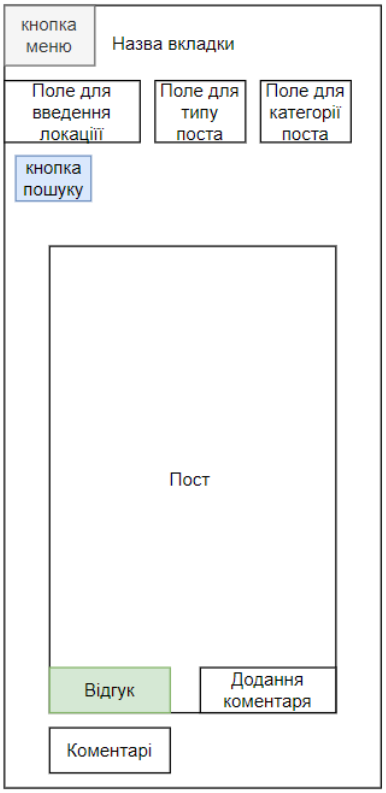


Рисунок 4.9 – Сторінка з чатами від інших користувачів

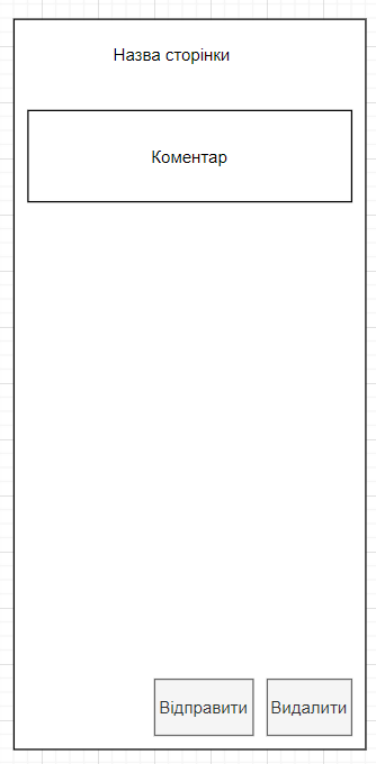


Рисунок 4.10 – Сторінка написання коментаря

– відкриття сторінки с постами користувача при натисканні кнопки «Мої пости» у меню додатку – рисунок 4.11. Відкриття сторінки для створення поста при натисканні кнопку додавання поста у вікні з постами користувача. Щоб відредагувати пост, потрібно натиснути на кнопку редагування поста. Щоб видалити пост, потрібно натиснути кнопку видалення поста;

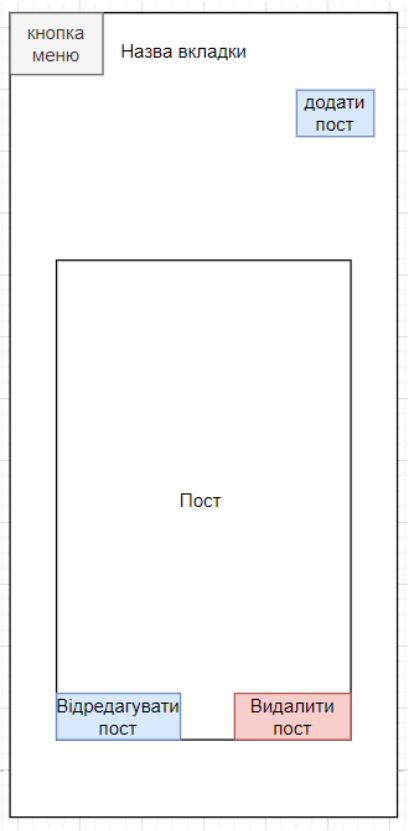


Рисунок 4.11 – Сторінка з постами користувача

– відкриття сторінки для створення поста при натисканні кнопки «Створити пост» у меню додатку – рисунок 4.12. Щоб додати зображення, потрібно натиснути кнопку додавання зображення. Щоб створити пост, потрібно натиснути на кнопку створення зображення;

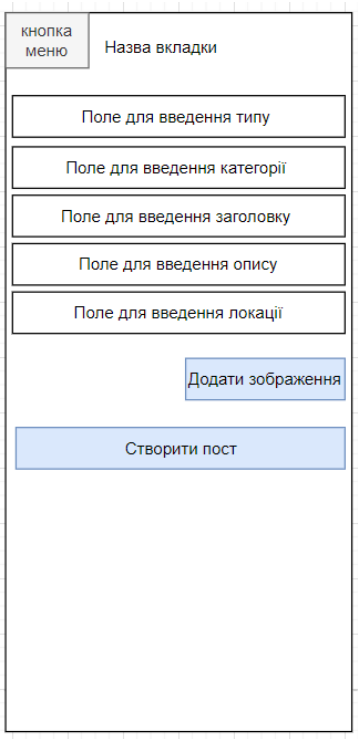


Рисунок 4.12 – Сторінка створення поста

– відкриття сторінки входу до системи при натисканні кнопки виходу із системи у меню додатку;

4.1.2 Для користувача:

– реєстрація;
– аутентифікація;
– вихід із системи;
– перегляд постів від інших користувачів;
– фільтрація постів від інших користувачів;
– відгук на пост;
– перегляд коментарів під постом;
– написання коментаря під постом;
– перегляд даних профілю користувача;
– редагування даних профілю користувача;
– перегляд чатів;
– написання повідомлення до чату;

– перегляд деталей поста, до якого був створений чат;
– створення поста;
– редагування поста;
– видалення поста;

4.2 Вимоги до надійності

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

4.3 Умови експлуатації

Умови експлуатації згідно СанПін 2.2.2.542 – 96.

4.3.1 Вид обслуговування

Вимоги до виду обслуговування не висуваються.

4.3.2 Обслуговуючий персонал

Вимоги до обслуговуючого персоналу не висуваються.

4.4 Вимоги до складу і параметрів технічних засобів

Програмне забезпечення повинно функціонувати на персональних комп'ютерах та мобільних пристроях під операційною системою Android.

Мінімальна конфігурація технічних засобів:
– версія Android: Android 5.0 (Lollipop) або вище;
– процесор: ARM або Intel з архітектурою ARMv7 або вище;
– об'єм оперативної пам'яті (ОЗП): 1 ГБ та більше;
– вільне місце на пристрої: 2ГБ та більше;
– підключення до Інтернету зі швидкістю від 20 мегабіт;

Рекомендована конфігурація технічних засобів:
– Версія Android: Android 7.0 (Nougat) або вище;
– процесор: ARM або Intel з архітектурою ARMv8 або вище;

– об'єм оперативної пам'яті (ОЗП): 4 ГБ або більше;

– вільне місце на пристрої: рекомендується не менше 2 ГБ для додатку та зберігання даних;

– об‘єм ОЗП: 32 Гб;

– підключення до мережі Інтернет зі швидкістю від 1 гігабіт;

4.5 Вимоги до інформаційної та програмної сумісності

Веб-застосунок повинен працювати під управлінням усіх операційних систем які підтримують підключення до мережі інтернет та можуть надати доступ до веб-додатку через браузер.

4.5.1 Вимоги до вхідних даних

Вхідні дані вводяться користувачем за допомогою веб-застосунку.

4.5.2 Вимоги до вихідних даних

Результати повинні бути графічно представлені на інтерфейсі користувача.

4.5.3 Вимоги до мови розробки

Розробку виконати на мові програмування C# та TypeScript.

4.5.4 Вимоги до середовища розробки

Розробку виконати на за допомогою середовищ Microsoft Visual Studio, Microsoft Visual Studio Code та Microsoft SQL Server Management Studio

4.5.5 Вимоги до представленню вихідних кодів

Вихідний код програми має бути представлений у вигляді файлів із розширеннями cs, ts, html, scss.

4.6 Вимоги до маркування та пакування

Вимоги до маркування та пакування не висуваються.

4.7 Вимоги до транспортування та зберігання

Вимоги до транспортування та зберігання не висуваються.

4.8 Спеціальні вимоги

Спеціальні вимоги не висуваються.

5 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

5.1 Попередній склад програмної документації

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

– пояснювальна записка;

– технічне завдання;

– текст програми;

– програма та методика тестування;

– керівництво користувача;

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

– схема структурна варіантів використання;

– схема бази даних;

– креслення вигляду екранних форм;

5.2 Спеціальні вимоги до програмної документації

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

6 СТАДІЇ І ЕТАПИ РОЗРОБКИ

№	Назва етапу	Строк	Звітність
1.	Вивчення літератури за тематикою проекту	21.02	
2.	Розробка технічного завдання	03.03	Технічне завдання
3.	Аналіз вимог та уточнення специфікацій	19.04	Специфікації програмного забезпечення
4.	Проектування структури програмного забезпечення, проектування компонентів	30.04	Схема структурна програмного забезпечення та специфікація компонентів (діаграма класів, схема алгоритму)
5.	Програмна реалізація програмного забезпечення	05.05	Тексти програмного забезпечення
6.	Тестування програмного забезпечення	10.05	Тести, результати тестування
7.	Розробка матеріалів текстової частини проекту	14.05	Пояснювальна записка
8.	Розробка матеріалів графічної частини проекту	20.05	Графічний матеріал проекту
9.	Оформлення технічної документації проекту	29.05	Технічна документація

7 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

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

Пояснювальна записка
до дипломного проєкту

на тему:

Веб-застосунок з надання та отримання послуг
для власників домашніх тварин

КПІ.ІТ-9208.045440.02.81

Київ – 2023

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

IDE	–	Integrated Development Environment – інтегроване середовище розробки.
API	–	Application programming interface, прикладний програмний Інтерфейс
ER	–	Entity-Relation diagram
MSSQL	–	Microsoft SQL Server
БД	–	База даних

ВСТУП

Актуальність даної роботи обумовлена зростаючим попитом на послуги для домашніх тварин у воєнний період та потребою у зручному способі їх надання та отримання. Велика кількість господарів залишили своїх улюбленців вдома або у інших людей та виїхали і безпечне місце. Тепер господарі потребують можливості легко та швидко отримувати доступ до професійної допомоги ветеринарів, кінологів, нянь та інших спеціалістів у галузі надання послуг для домашніх тварин. Це означає, що тема є надзвичайно актуально у нас час і потребує більш детального розгляду.

Оскільки ми живемо у час технологій та інтернету, можна відслідкувати тенденцію на зростання популярності веб-застосунків для надання послуг у різних галузях, включаючи тваринництво. Веб-платформи забезпечують легкий та зручний доступ до інформації та послуг, оскільки потребують мінімуму: наявності комп’ютера чи телефона та інтернету.

Сучасний стан об’єкта розробки, тобто веб-застосунку цифрової платформи надання та отримання послуг для власників домашніх тварин, ще не повністю розвинутий. Проте, провідні наукові установи та організації, такі як PetDesk та Rover, пропонують веб-застосунки, які надають доступ до інформації про тварин, графіків візитів до ветеринарів, замовлення послуг та товарів для тварин.

Можливі сфери застосування веб-застосунку цифрової платформи надання та отримання послуг для власників домашніх тварин можуть включати підтримку здоров'я та догляду за тваринами, замовлення товарів для тварин, бронювання послуг ветеринарних клінік для тварин, а також отримання порад та рекомендацій від фахівців у цій галузі.

1 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Загальні положення

Предметна область веб-застосунка цифрової платформи надання та отримання послуг для власників домашніх тварин пов'язана зі здоров'ям, доглядом та життям домашніх тварин. Це включає у себе ветеринарну медицину, догляд за тваринами, харчування, а також інші товари та послуги, пов'язані з домашніми тваринами.

Основні користувачі включають в себе власників домашніх тварин, які потребують надання послуг для своїх тварин, а також виконавців, які є спеціалістами або навіть любителями у певній сфері, пов’язаній з тваринами.

Напрямки розробок веб-застосунків цифрової платформи надання та отримання послуг для власників домашніх тварин можуть бути різними: мобільні додатки та веб-сайти.

1.2 Змістовний опис і аналіз предметної області

Рівень знань в предметній області надання послуг для власників домашніх тварин є надзвичайно важливим у програмному забезпеченні. Розробники повинні розуміти основні принципи догляду за домашніми тваринами, їх поведінку та потреби. Від рівня підготовки розробників залежить якість та функціональність веб-застосунку. Найкращий випадок – коли розробники самі є власниками домашніх тварин.

Недоліками поточного стану речей з предметною областю у сфері ІТ можна вважати недостатню увагу до деталей догляду за домашніми тваринами, а також обмежену кількість функцій, які можуть бути надані користувачам. Також зазвичай люди можуть замовити та отримати послуги для своїх тварин не на вузькоспеціалізованій платформі з послугами тільки для тварин, а на платформах, які охоплюють різноманітні варіанти послуг.

Шляхи покращення ситуації з розробками у сфері ІТ для власників домашніх тварин, в першу чергу, можуть включати більш глибоке

дослідження потреб та проблем власників домашніх тварин, а також виділення окремої платформи, яка буде надавати послуги лише для власників тварин.

1.3 Аналіз існуючих технологій та успішних ІТ-проєктів

Проаналізуємо відоме на сьогодні алгоритмічне забезпечення у даній області та технічні рішення, що допоможуть у реалізації веб-застосунку цифрової платформи надання та отримання послуг для власників домашніх тварин. Далі будуть розглянуті допоміжні програмні засоби, засоби розробки та готові програмні рішення.

1.3.1 Аналіз відомих алгоритмічних та технічних рішень

У наш час замовники мають доступ до спеціалізованих платформ з різноманітними видами послуг. Один з мінусів цих платформ – занадто велика кількість різноманітних послуг, що може збити користувача і створити видимість платформи, яка переповнена не потрібними фічами. Як приклад таких платформ можна привести платформи OLX і Kabanchik.

Основна ціль – створити зручний та інтуітивно зрозумілий застосунок, який дозволить як замовникам послуг, так і виконавцям легко і швидко знайти людину для своїх конкретних цілей.

Один з головних алгоритмів цього дипломного проекту – це алгоритм пошуку виконавців або замовників. Спочатку людина реєструється на платформі, після чого вона може переглянути існуючи пости замовників та виконавців, а також створити свої власні пости. У кожного користувача є можливість відгукнутися на будь-який пост.

На даний час існує багато відомих архітектур програмного забезпечення, серед яких модель-вид-контролер (Model-View-Controller, MVC), клієнт-сервер (Client-Server), архітектура з шаруванням (Layered Architecture) та інші. Одним з найкращих варіантів є поєднання декількох архітектур задля можливості сумістити їх сильні сторони. Наприклад, можна

обрати клієнт-серверну архітектуру, але на стороні серверу використати схему послідовностей.

Патерни, які можна використати у розробці, можна обирати серед патернів банди чотирьох. це набір патернів проектування, описаних у книзі «Design Patterns: Elements of Reusable Object-Oriented Software», написаній Еріхом Гаммою, Річардом Хелмом, Ральфом Джонсоном і Джоном Вліссідесом. Ці патерни є шаблонами для розв'язання поширених проблем проектування програмного забезпечення.

1.3.2 Аналіз допоміжних програмних засобів та засобів розробки

Існує достатня велика кількість IDE, за допомогою яких можна виконати розробку, що розглядається у даній дипломній роботі. Серед варіантів є Visual Studio, Visual Studio Code, Rider, MonoDevelop та інші. Вибір IDE залежить від зручності IDE та функціональності, яку вона пропонує. Проте якщо розробник вже має досвід роботи з IDE, є сенс використати ту IDE, з якою він вже звик працювати і яка є зручною конкретно для нього.

Стосовно мови програмування, наразі існує доволі велика кількість мов, які можуть покрити потреби майбутнього веб-застосунку. Доцільно зупинити вибір на популярній мові програмування, по якій створена велика кількість навчальних матеріалів, і яку буде легко підтримувати. Серед таких мов можна виділити C#, Java, Python, PHP, Ruby та інші. Для клієнтської частини можна обирати між такими популярними мовами, як Java Script та Type Script.

Оскільки у проекті буде відбуватися взаємодія з базою даних, доцільно підібрати фреймворк, який це дозволить робити. Одними з найпопулярніших фрейморків для взаємодії з базами даних є Entity Framework, Hibernate, Eloquent ORM, що вбудований у Laravel та інші. Кожен фреймворк пов’язаний з певною мовою програмування, тому потрібно враховувати цей факт при його виборі.

Для створення веб-застосунку, який розглядається у даному дипломному проекті, було обрано мову програмування C# [1] та платформу ASP.NET Core. У якості фреймворку для роботи з базою даних було взято Entity Framework [2]. C# є сучасною потужною мовою програмування, яка має велику кількість переваг, особливо коли використовується разом з платформою ASP.NET Core.

C# має чітку синтаксичну структуру та об'єктно-орієнтований підхід, що спрощує розробку як простих, так і більш складних додатків. Платформа ASP.NET Core [3], яка побудована на основі мови C#, є потужним інструментом для розробки веб-додатків. Серед переваг даної платформи є висока продуктивність, масштабованість та безпека. Також ASP.NET Core має вбудовану підтримку для розробки API, що дозволяє розробникам легко створювати RESTful API для взаємодії з клієнтськими додатками та іншими службами.

1.3.3 Аналіз відомих програмних продуктів

Як було зазначено вище, на ринку вже існують платформи, на яких люди можуть знайти замовників та виконавців для своїх послуг. Більшість платформ загальні, у яких послуги для домашніх тварин є лише однією із багатьох інших варіантів послуг. Наведемо одні з найпопулярніших платформ.

OLX: це популярна онлайн-платформа для розміщення оголошень про купівлю, продаж та обмін різних товарів та послуг. Вона дозволяє користувачам взаємодіяти безпосередньо між собою, без посередництва компанії. OLX дозволяє користувачам розміщувати оголошення у різних категоріях, а також вибирати категорії, регіони, цінові діапазони та інші параметри для точного пошуку. Головну сторінку платформи можна побачити на рисунку 1.1.

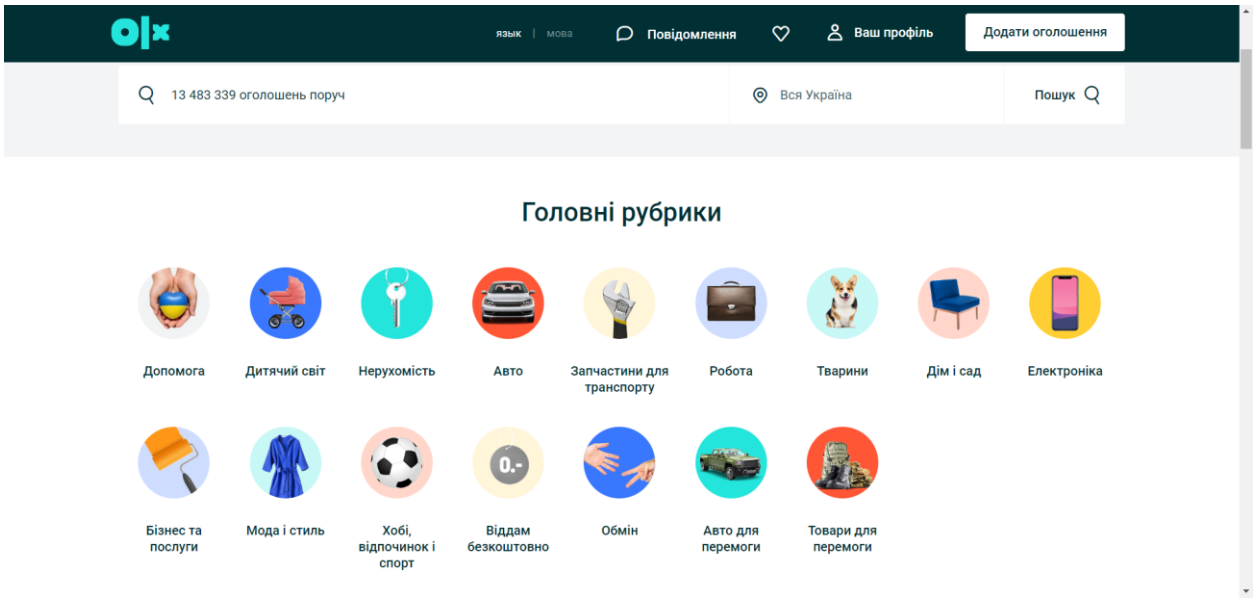


Рисунок 1.1 – Головна сторінка платформи OLX

Kabanchik: онлайн-платформа, яка забезпечує послуги зі замовлення та надання різноманітних послуг. Платформа пропонує широкий спектр послуг, користувачі можуть замовляти різні послуги залежно від своїх потреб. Користувачі можуть залишати відгуки та оцінки про фахівців, з якими вони співпрацювали раніше. Також Kabanchik надає можливість забезпечити попередню оплату, яка буде зберігатися на спеціальному рахунку до завершення робіт. Головну сторінку платформи можна побачити на рисунку 1.2.

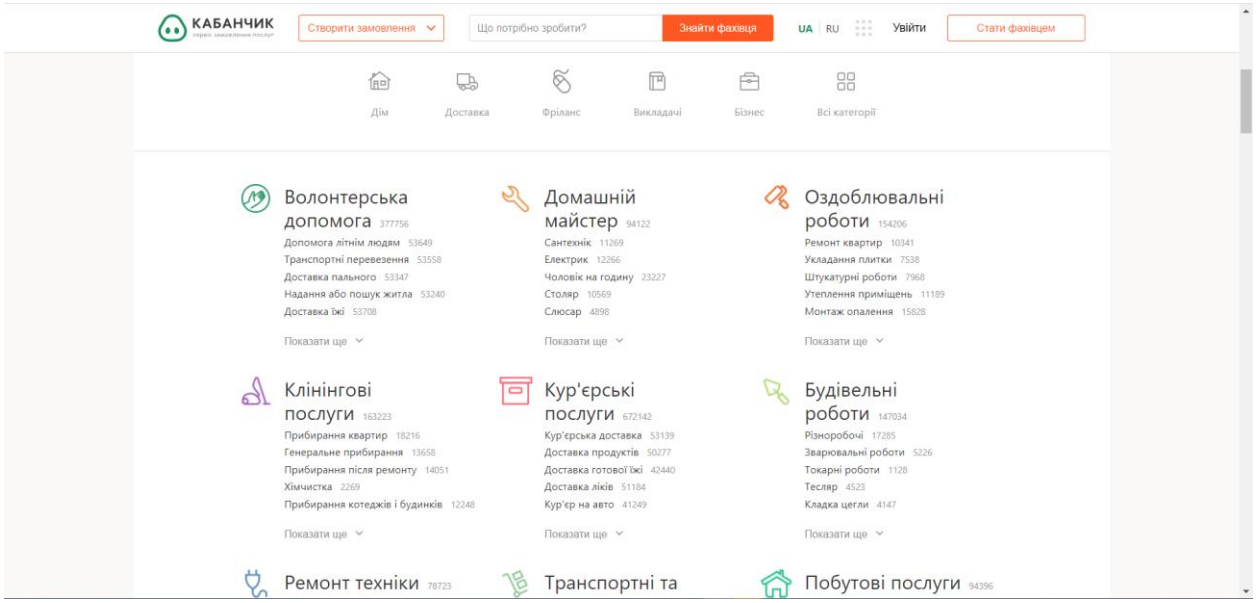


Рисунок 1.2 – Головна сторінка платформи Kabanchik

Основна відмінність між веб-застосунком, який розглядається у даній дипломній роботі, і платформами, наведеними вище, в тому, що веб-застосунок буде вузькоспеціалізованим – послуги, які можна буде отримати або замовити, будуть стосуватися лише домашніх тварин. Платформ з такою самою вузькою спеціалізацією серед українських платформ знайдено не було. Проте є платформи, які пропонують послуги лише з однієї сфери для домашніх тварин, наприклад, ветеринарна допомога або магазини з різноманітними продуктами для домашніх улюбленців.

1.4 Аналіз вимог до програмного забезпечення

Головною функцією програмного забезпечення є надання людям можливості легко знайти виконавців або замовників для послуг, які стосуються домашніх тварин. Більше функцій можна побачити на рисунку 1.3.

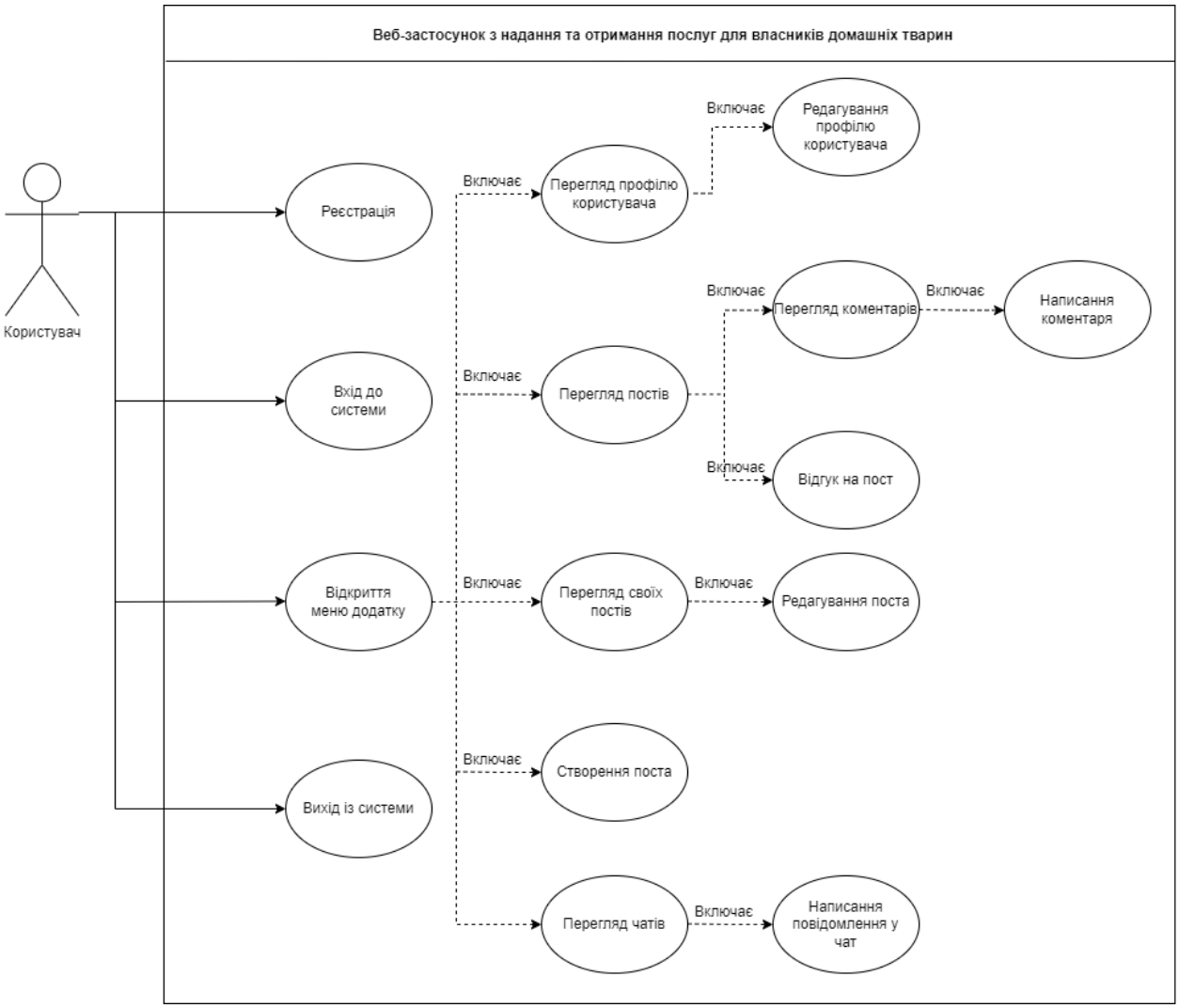


Рисунок 1.3 – Діаграма варіантів використання

В таблицях 1.1 - 1.15 наведені варіанти використання програмного забезпечення.

Таблиця 1.1 - Варіант використання UC-1

Use case name	Реєстрація користувача
Use case ID	UC-01
Goals	Реєстрація нового користувача в системі
Actors	Незареєстрований користувач
Trigger	Користувач бажає зареєструватися
Pre-conditions	-
Flow of Events	Користувач переходить на сторінку логіну. Користувач бажає зареєструватися на тисне кнопку «Registration», після чого переходить на сторінку реєстрації. В поля для реєстрації вводяться відповідні дані: пошта, ім’я, прізвище користувача, пароль в системі. Після заповнення даних користувача натискає кнопку реєстрації. Після цього з’являється повідомлення про успішну реєстрацію і користувач перенаправляється на сторінку входу
Extension	У випадку введення не коректних даних, кнопка реєстрації стає неактивною
Post-Condition	Створення акаунту користувача, перехід на сторінку входу

Таблиця 1.2 - Варіант використання UC-2

Use case name	Вхід до системи
Use case ID	UC-02
Goals	Вхід існуючого користувача до системи
Actors	Зареєстрований користувач
Trigger	Користувач бажає увійти до системи
Pre-conditions	-
Flow of Events	Користувач переходить на сторінку логіну. Користувач вводить в відповідні поля свої електронну пошту та пароль, після чого натискає кнопку «Login»
Extension	У випадку введення не коректних даних, користувачу буде показане повідомлення
Post-Condition	Користувач входить до системи

Таблиця 1.3 - Варіант використання UC-3

Use case name	Відкриття меню додатку
Use case ID	UC-03
Goals	Надати користувачу можливість перейти на іншу вкладку додатка
Actors	Користувач
Trigger	Користувач робить свайп вліво або натискає на кнопку на навігаційній панелі затосунку
Pre-conditions	-
Flow of Events	Користувач робить свайп вліво або нажимає на кнопку та переходить до меню, де знаходяться назви усіх вкладок застосунку та кнопка для виходу із акаунту
Extension	-
Post-Condition	Користувач відкрив меню з перерахованими вкладками додатку

Таблиця 1.4 - Варіант використання UC-4

Use case name	Перегляд профілю користувача
Use case ID	UC-04
Goals	Надати користувачу можливість переглянути дані свого профілю, де показані його ім’я, прізвище, електронна пошта та номер телефону. Поле для паролю показане пустим.
Actors	Користувач
Trigger	Користувач тисне на вкладку «Profile» в боковому меню
Pre-conditions	-
Flow of Events	Користувач переходить на вкладку «Profile»
Extension	-
Post-Condition	-

Таблиця 1.5 - Варіант використання UC-5

Use case name	Редагування профілю користувача
Use case ID	UC-05
Goals	Надати користувачу можливість редагувати профільні дані
Actors	Користувач
Trigger	Користувач робить тисне на кнопку «Edit»
Pre-conditions	-
Flow of Events	Користувач переходить на вкладку «Profile». Користувач натискає на кнопку «Edit» під своїми даними. Після натискання на кнопку «Edit» відкривається вкладка з полями для вводу імені, прізвища, пошти, номера телефону та пароля користувача. Після введення даних користувач натискає «Save» для збереження даних.
Extension	У випадку введення не коректних даних, користувачу буде показане повідомлення про помилку
Post-Condition	Користувач змінив деякі або усі свої профільні дані

Таблиця 1.6 - Варіант використання UC-6

Use case name	Перегляд постів
Use case ID	UC-06
Goals	Надати користувачу можливість переглянути пости від інших користувачів
Actors	Користувач
Trigger	Користувач тисне на вкладку «Posts» в боковому меню
Pre-conditions	-
Flow of Events	Користувач переходить на вкладку «Posts». Користувач бачить пости від інших користувачів
Extension	Користувач може виконати фільтрацію постів за локацією, типом поста та категорією поста
Post-Condition	-

Таблиця 1.7 - Варіант використання UC-7

Use case name	Перегляд коментарів до поста
Use case ID	UC-07
Goals	Надати користувачу можливість переглянути коментарі до поста
Actors	Користувач
Trigger	Користувач тисне на кнопку «Comments» під постом
Pre-conditions	-
Flow of Events	Користувач переходить до вкладки «Posts». Користувач тисне на кнопку «Comments» під постом, після чого розгортаються коментарі
Extension	При повторному натисканні на кнопку «Comments» коментарі згорнуться назад
Post-Condition	-

Таблиця 1.8 - Варіант використання UC-8

Use case name	Написання коментаря
Use case ID	UC-08
Goals	Надати користувачу можливість написати коментар під постом іншого користувача
Actors	Користувач
Trigger	Користувач вводить дані в відповідне поле та тисне кнопку «Send»
Pre-conditions	-
Flow of Events	Користувач переходить до вкладки «Posts». Користувач тисне на кнопку «Comments» під постом, після чого розгортаються коментарі. Користувач натискає на кнопку «Add Comment», після чого вводить текст у відповідне поле та тисне кнопку «Send»
Extension	-
Post-Condition	Створення коментаря під постом іншого користувача

Таблиця 1.9 - Варіант використання UC-9

Use case name	Відгук на пост
Use case ID	UC-09
Goals	Надати користувачу можливість відгукнутися на пост
Actors	Користувач
Trigger	Користувач тисне на кнопку «Respond» під постом іншого користувача
Pre-conditions	-
Flow of Events	Користувач переходить до вкладки «Posts». Користувач тисне на кнопку «Respond» під постом. Після натискання кнопки користувач переходить до чату з користувачем, який створив відповідний пост
Extension	-
Post-Condition	Створення та відкриття чату з користувачем, який створив пост

Таблиця 1.10 - Варіант використання UC-10

Use case name	Перегляд своїх постів
Use case ID	UC-10
Goals	Надати користувачу можливість переглянути свої пости
Actors	Користувач
Trigger	Користувач тисне на вкладку «My Posts» в боковому меню
Pre-conditions	-
Flow of Events	Користувач переходить до вкладки «My Posts». Користувач бачить свої створені пости
Extension	Користувач може створити пост при натисканні на кнопку зі знаком «+»
Post-Condition	-

Таблиця 1.11 - Варіант використання UC-11

Use case name	Редагування поста
Use case ID	UC-11
Goals	Надати користувачу можливість редагувати пост, який він створив
Actors	Користувач
Trigger	Користувач тисне на кнопку із зображенням олівця під відповідним постом
Pre-conditions	-
Flow of Events	Користувач переходить до вкладки «My Posts». Користувач бачить свої створені пости. Користувач тисне на кнопку із зображенням олівця під відповідним постом, після чого відкривається вікно з полями для редагування. Щоб зберегти пост, користувач тисне на кнопку «Save»
Extension	-
Post-Condition	-

Таблиця 1.12 - Варіант використання UC-12

Use case name	Створення поста
Use case ID	UC-12
Goals	Надати користувачу можливість створити пост
Actors	Користувач
Trigger	Користувач тисне на вкладку «Create Post» в боковому меню
Pre-conditions	-
Flow of Events	Користувач тисне на вкладку «Create Post», після чого відкривається вікно з полями для вводу даних про тип поста, категорію поста, назву, опис та локацію. Користувач тисне на кнопку «Create» щоб зберегти пост
Extension	-
Post-Condition	Створений пост відображається у вкладці «My Posts»

Таблиця 1.13 - Варіант використання UC-13

Use case name	Перегляд чатів
Use case ID	UC-13
Goals	Надати користувачу можливість переглянути свої чати
Actors	Користувач
Trigger	Користувач тисне на вкладку «Chats» в боковому меню
Pre-conditions	-
Flow of Events	Користувач тисне на вкладку «Chats», після чого відкривається вікно з назвами чатів, в яких приймає участь користувач
Extension	-
Post-Condition	-

Таблиця 1.14 - Варіант використання UC-14

Use case name	Написання повідомлення у чат
Use case ID	UC-14
Goals	Надати користувачу можливість написати повідомлення іншому користувачу
Actors	Користувач
Trigger	Користувач тисне на кнопку з зображенням стрілки після написання тексту у поле для повідомлення
Pre-conditions	-
Flow of Events	Користувач тисне на вкладку «Chats», після чого відкривається вікно з назвами чатів, в яких приймає участь користувач. Користувач тисне на назву чату, після чого відкривається відповідний чат. Користувач вводить текст у поле для повідомлення і тисне кнопку з зображенням стрілки справа від поля
Extension	-
Post-Condition	Відправлення повідомлення до іншого користувача

Таблиця 1.15 - Варіант використання UC-15

Use case name	Вихід користувача із системи
Use case ID	UC-15
Goals	Надати користувачу можливість вийти із системи
Actors	Користувач
Trigger	Користувач тисне на вкладку «Logout» в боковому меню
Pre-conditions	-
Flow of Events	Користувач тисне на вкладку «Logout,» після чого відбувається вихід користувача із системи. Відкривається вікно для входу користувача до системи
Extension	-
Post-Condition	-

1.4.1 Розроблення функціональних вимог

Програмне забезпечення розділене на модулі. Кожен модуль має свій певний набір функцій. У таблиці 1.16 наведена модель вимог у загальному вигляді. На рисунку 1.4 наведено матрицю трасування вимог, а в таблицях 1.17 – 1.33 наведений опис функціональних вимог до програмного забезпечення.

Таблиця 1.16 – Модель вимог у загальному вигляді

Title ID	Full Description	Code	Priority	Risk	Status
1.	Реєстрація користувача	REQ_1	1	Undefined	Draft
2.	Вхід користувача до системи	REQ_2	Undefined	Undefined	Draft
3.	Перехід до меню додатку	REQ_3	Undefined	Undefined	Draft
3.1	Вибір вкладки	REQ_4	Undefined	Undefined	Draft
4.	Перегляд даних профілю користувача	REQ_5	Undefined	Undefined	Draft
5.	Редагування профілю користувача	REQ_6	Undefined	Undefined	Draft

Продовження таблиці 1.16

6.	Перегляд постів інших користувачів	REQ_7	Undefined	Undefined	Draft
7.	Фільтрація постів інших користувачів	REQ_8	Undefined	Undefined	Draft
8.	Перегляд коментарів до постів	REQ_9	Undefined	Undefined	Draft
9.	Написання коментарів до постів	REQ_10	Undefined	Undefined	Draft
10	Відгук на пост іншого користувача	REQ_11	Undefined	Undefined	Draft
11.	Перегляд своїх постів	REQ_12	Undefined	Undefined	Draft
12.	Редагування своїх постів	REQ_13	Undefined	Undefined	Draft
13.	Створення постів	REQ_14	Undefined	Undefined	Draft
14.	Перегляд чатів	REQ_15	Undefined	Undefined	Draft
15.	Написання повідомлення у чат	REQ_16	Undefined	Undefined	Draft
16.	Вихід із системи	REQ_17	Undefined	Undefined	Draft

Таблиця 1.17 – Функціональна вимога FR-1

Назва	Реєстрація користувача
Опис	Користувач повинен мати можливість реєстрації користувачеві шляхом введення імені, прізвища, номера телефону, пошти, паролю.

Таблиця 1.18 – Функціональна вимога FR-2

Назва	Вхід користувача до системи
Опис	Користувач повинен мати можливість користувачеві увійти в систему, для чого потрібно ввести пошту і пароль

Таблиця 1.19 – Функціональна вимога FR-3

Назва	Перехід до меню додатку
Опис	Користувач повинен мати можливість користувачеві перейти до меню додатку, у якому показані назви вкладок, на які користувач може перейти.

Таблиця 1.20 – Функціональна вимога FR-4

Назва	Вибір вкладки
Опис	Користувач повинен мати можливість користувачеві обрати вкладку, на яку він бажає перейти, шляхом натискання на неї.

Таблиця 1.21 – Функціональна вимога 6

Назва	Перегляд даних профілю користувача
Опис	Користувач повинен мати можливість користувачеві переглядати дані його профілю, які раніше були введені користувачем.

Таблиця 1.22 – Функціональна вимога FR-6

Назва	Редагування профілю користувача
Опис	Користувач повинен мати можливість користувачеві редагувати дані свого профілю, такі як ім’я, прізвище, пошту, номер телефону та пароль.

Таблиця 1.23 – Функціональна вимога FR-7

Назва	Перегляд постів інших користувачів
Опис	Користувач повинен мати можливість користувачеві переглядати пости, створені іншими користувачами.

Таблиця 1.24 – Функціональна вимога FR-8

Назва	Фільтрація постів інших користувачів
Опис	Користувач повинен мати можливість користувачеві фільтрувати пости інших користувачів за локацією, типом та категорією.

Таблиця 1.25 – Функціональна вимога FR-9

Назва	Перегляд коментарів до постів
Опис	Користувач повинен мати можливість користувачеві переглядати коментарі до постів інших користувачів.

Таблиця 1.26 – Функціональна вимога FR-10

Назва	Написання коментарів до постів
Опис	Користувач повинен мати можливість користувачеві писати коментарі до постів інших користувачів.

Таблиця 1.27 – Функціональна вимога FR-11

Назва	Відгук на пост іншого користувача
Опис	Користувач повинен мати можливість користувачеві відгукатися на пости інших користувачів. Після відгука буде створено чат з користувачем, який опублікував пост.

Таблиця 1.28 – Функціональна вимога FR-12

Назва	Перегляд своїх постів
Опис	Користувач повинен мати можливість користувачеві переглядати пости, які були ним створені.

Таблиця 1.29 – Функціональна вимога FR-13

Назва	Редагування своїх постів
Опис	Користувач повинен мати можливість користувачеві редагувати пости, які були ним створені.

Таблиця 1.30 – Функціональна вимога FR-14

Назва	Створення постів
Опис	Користувач повинен мати можливість користувачеві створювати пости з вказанням типу, категорії, локації, назви та опису.

Таблиця 1.31 – Функціональна вимога FR-15

Назва	Перегляд чатів
Опис	Користувач повинен мати можливість користувачеві переглядати чати, в яких він приймає участь.

Таблиця 1.32 – Функціональна вимога FR-16

Назва	Написання повідомлення у чат
Опис	Користувач повинен мати можливість користувачеві писати повідомлення у чати, в яких він приймає участь.

Таблиця 1.33 – Функціональна вимога FR-17

Назва	Вихід із системи
Опис	Користувач повинен мати можливість користувачеві вийти із системи.

	FR-1	FR-2	FR-3	FR-4	FR-5	FR-6	FR-7	FR-8	FR-9	FR-10	FR-11	FR-12	FR-13	FR-14	FR-15	FR-16	FR-17
UC-1	+																
UC-2		+															
UC-3			+	+													
UC-4					+												
UC-5						+											
UC-6							+	+									
UC-7									+								
UC-8										+							
UC-9											+						
UC-10												+					
UC-11													+				
UC-12														+			
UC-13															+		
UC-14																+	
UC-15																	+

Рисунок 1.4 – Матриця трасування вимог

1.4.2 Розроблення нефункціональних вимог

Мета даної дипломної роботи – створення веб-застосунку з надання та отримання послуг для власників домашніх тварин з наступними нефункціональними вимогами:

– застосунок повинен мати єдиний дизайн: усі вікна, кнопки, шрифт повинні бути оформлені в схожому стилі;

– застосунок повинен бути швидким, моментально реагувати на дії користувача;

– застосунок повинен мати простий та зручний користувацький інтерфейс;

– застосунок повинен бути адаптивним до екранів різноманітних мобільних пристроїв.

1.5 Постановка задачі

Мета даного дипломного проекту – створити веб-застосунок з надання та отримання послуг для власників домашніх тварин. Застосунок повинен дозволити користувачам швидко та легко знаходити виконавців та замовників для своїх послуг.

Користувач зможе переглядати пости від інших користувачів та фільтрувати їх задля більш продуктивного пошуку. Під кожним постом користувачі зможуть залишати коментарі, що дозволить людям залишати свої думки на рахунок постів інших користувачів.

Для комунікації між користувачами в застосунку буде реалізовано чати, чат стане доступним після того, як користувач відгукнувся на пост іншого користувача.

Висновки до розділу

У даному розділі було проведено аналіз предметної області і здійснено змістовний опис веб-застосунку з надання та отримання послуг для власників домашніх тварин. У розділі було проаналізовано та описано технології, які можливо використати для реалізації застосунку, а також було показано та описано вже існуючі програмні рішення. Після аналізу існуючих програмних рішень було зроблено висновок, що на українському ринку відсутні вузькоспеціалізовані застосунки, які легким да доступним способам дозволили б користувачам замовляти та надавати різноманітні послуги у сфері домашніх тварин.

Також у розділі були проаналізовані та описані функціональні та нефункціональні вимоги до застосунку. Була створена діаграма варіантів використання та матриця трасування вимог.

Як результат, була сформульована та оформлена постановка задачі для даного дипломного проекту.

2 МОДЕЛЮВАННЯ ТА КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Моделювання та аналіз програмного забезпечення

Задля опису бізнес процесів програмного забезпечення було створено декілька BPMN моделей для кожного з бізнес-процесів веб-застосунку.

Бізнес процеси, для опису яких були створені BPMN моделі:

– реєстрація та вхід користувача до системи (рисунок 2.1);

– редагування профілю користувача (рисунок 2.2);

– перегляд постів від інших користувачів (рисунок 2.3);

– створення поста (рисунок 2.4);

– редагування поста (рисунок 2.5);

– написання повідомлення у чат (рисунок 2.6).

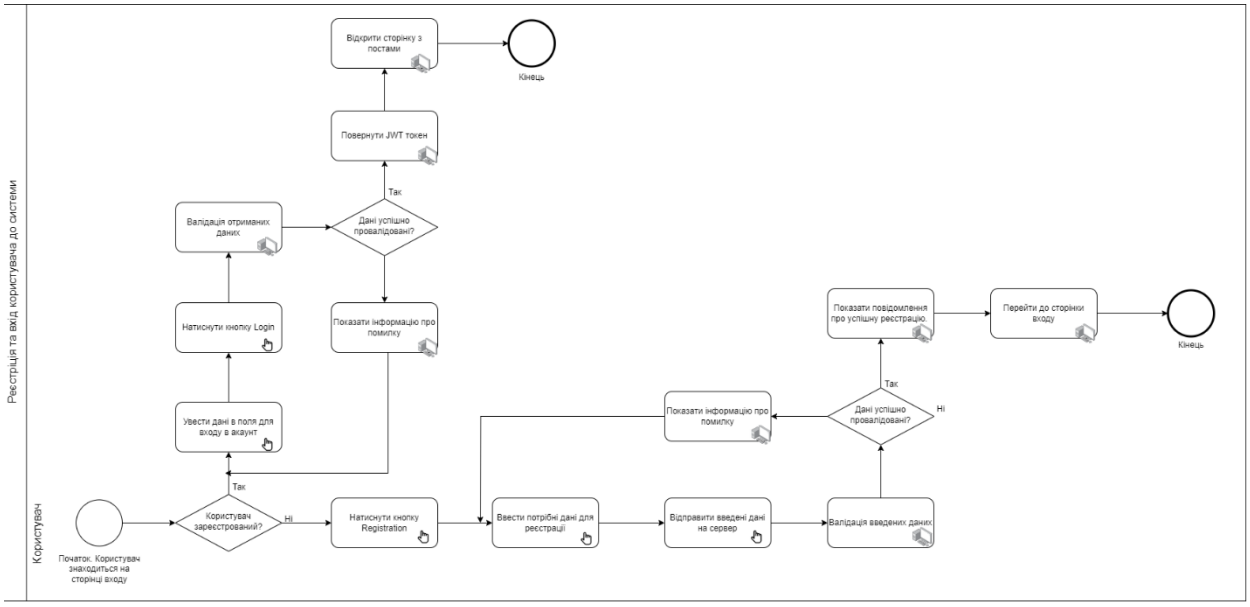


Рисунок 2.1 – Схема бізнес-процесу реєстрації та входу користувача до системи

Опис реєстрації та входу користувача до системи:

– користувач переходить на сторінку для входу в застосунок;

– якщо користувач зареєстрований, він вводить свої дані для входу у відповідні поля. Дані відправляються на сервер;

– якщо поля провалідовані, користувач успішно входить до системи. Відкривається сторінка з постами;

– якщо користувач не зареєстрований, він переходить на сторінку реєстрації;

– користувач вводить свої дані для реєстрації у відповідні поля. Дані відправляються на сервер;

– якщо поля провалідовані, користувач вважається успішно зареєстрованим у системі. Відкривається сторінка для входу в застосунок.

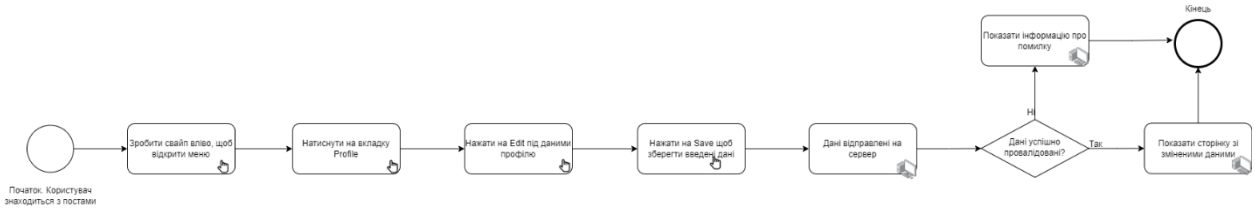


Рисунок 2.2 – Схема бізнес-процесу редагування профілю користувача

Опис редагування профілю користувача:

– користувач переходить на головну сторінку з постами;

– користувач робить свайп вліво, щоб відкрити меню;

– користувач натискає на вкладку «Profile»;

– користувач натискає на «Edit» під даними профілю;

– користувач вводить нові дані у відповідні поля.

– користувач тисне на кноку «Save». Дані відправляються на сервер;

– якщо поля провалідовані, дані вважаються успішно зміненими. Показуються змінені дані;

– якщо поля не провалідовані, показується помилка.

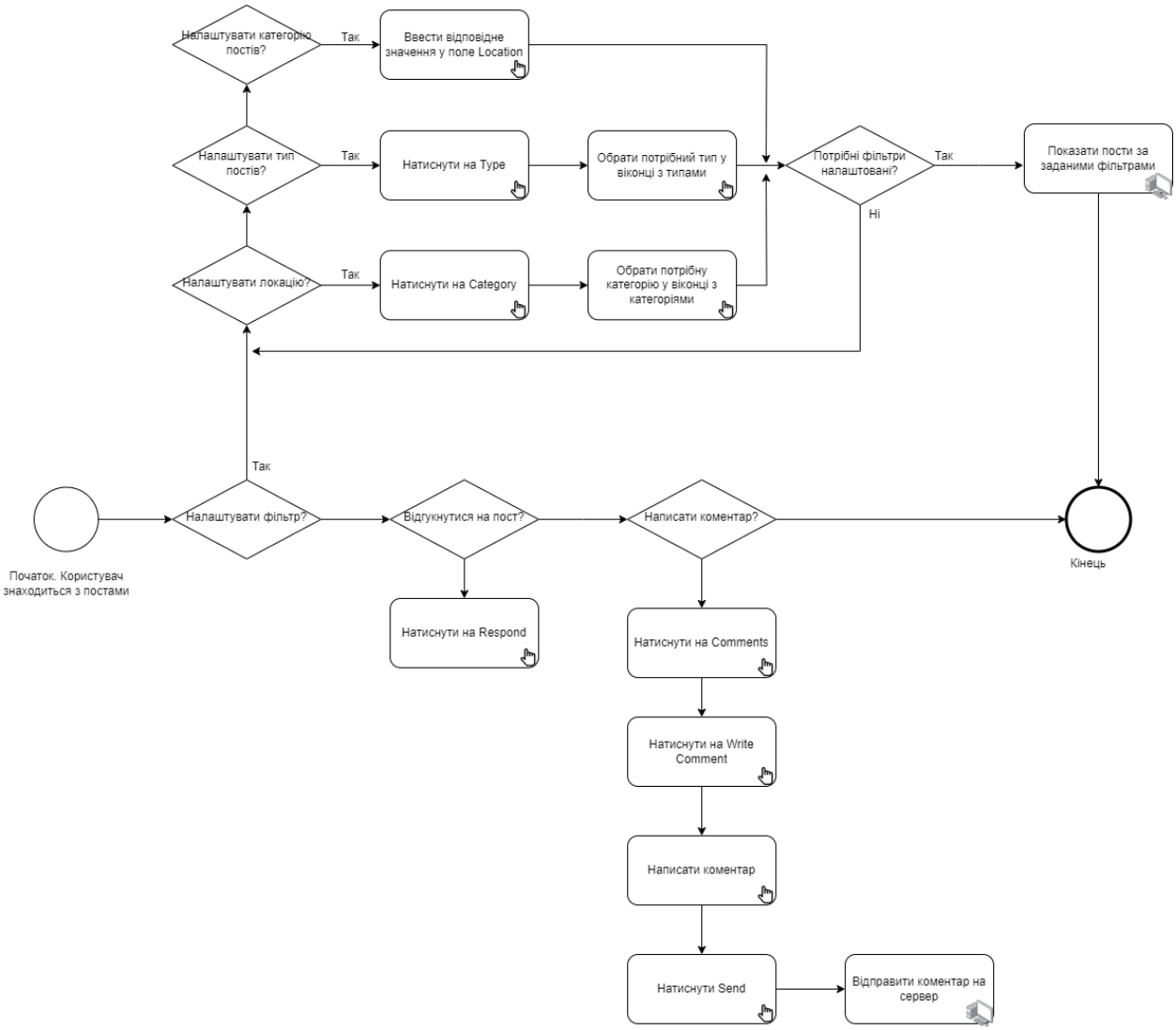


Рисунок 2.3 – Схема бізнес-процесу перегляду постів постів інших користувачів

Опис перегляду постів постів інших користувачів:

– користувач переходить на головну сторінку з постами;

– якщо є потреба, користувач може налаштувати фільтр за локацією, типом та категорією поста;

– якщо користувач бажає, він може відгукнутися на пост при натисканні на «Respond»;

– якщо користувач бажає, він може написати коментар під постом.

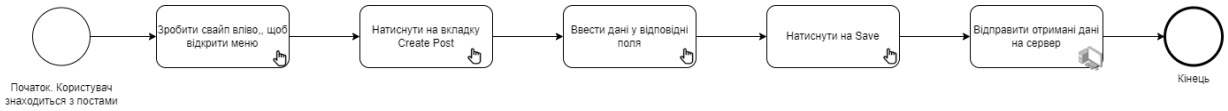


Рисунок 2.4 – Схема бізнес-процесу створення поста

Опис створення поста:

– користувач переходить на головну сторінку з постами;

– користувач робить свайп вліво, щоб відкрити меню;

– користувач натискає на вкладку «Create Post»;

– користувач вводить потрібні дані у відповідні поля;

– користувач тисне на кнопку «Create». Дані відправляються на сервер. Пост вважається створеним.

Рисунок 2.5 – Схема бізнес-процесу редагування поста

Опис редагування поста:

– користувач переходить на головну сторінку з постами;

– користувач робить свайп вліво, щоб відкрити меню;

– користувач натискає на вкладку «My Posts»;

– користувач натискає на іконку із зображенням олівця під потрібним постом;

– користувач вводить дані у відповідні поля;

– користувач тисне на «Save». Дані відправляються на сервер.

Рисунок 2.6 – Схема бізнес-процесу написання повідомлення у чат

Опис написання повідомлення у чат:

– користувач переходить на головну сторінку з постами;
– користувач робить свайп вліво, щоб відкрити меню;
– користувач натискає на вкладку «Chats»;
– користувач натискає на потрібний чат;
– користувач повідомлення у поле знизу вікна з повідомленнями;
– користувач тисне на кнопку у вигляді стрілки. Дані відправляються на сервер.

2.2 Архітектура програмного забезпечення

Архітектуру для веб-застосунку з надання та отримання послуг для власників домашніх тварин можна обрати серед таких відомих архітектурних підходів: модель-вид-контролер (Model-View-Controller, MVC), клієнт-сервер (Client-Server) та архітектура з шаруванням (Layered Architecture).

Архітектура MVC (Model-View-Controller) [4] – це шаблон архітектури програмного забезпечення, основна ідея якого полягає в тому, щоб розділити логічні компоненти програми на основі їх відповідальностей. MVC дозволяє чітко розділити компоненти системи на модель (Model), представлення (View) та контролер (Controller). Це сприяє покращенню керованості і облегшує розробку, тестування та підтримку коду. Компоненти MVC можуть бути розроблені та розгорнуті незалежно один від одного, що дозволяє збільшувати модульність системи, спрощує розширення функціональності і полегшує співпрацю в команді розробників. Також, через чітке розділення ролей, код, написаний для однієї частини MVC, може бути перевикористаний в інших частинах або проектах. Архітектура MVC надає гнучкість для зміни або заміни окремих компонентів без впливу на інші.

Архітектура «клієнт-сервер» [5] є моделлю взаємодії між клієнтськими пристроями (наприклад, веб-браузером) та сервером, де клієнти та сервери взаємодіють через мережеві запити та відповіді. Дана архітектура включає

такі основні компоненти: клієнт, сервер і канал зв’язку. Клієнт – це пристрій або додаток, який ініціює запити до сервера для отримання даних або виконання операцій. Це може бути, наприклад, веб-браузери, мобільні додатки або настільні додатки. Сервер – це комп'ютер або пристрій, який обробляє запити від клієнтів і надає дані або виконує запитані операції. Сервери можуть бути фізичними або віртуальними та зазвичай працюють в неперервному режимі, очікуючи запитів від клієнтів. У якості канала зв'язку між клієнтом та сервером може виступати HTTP (Hypertext Transfer Protocol), WebSocket, TCP (Transmission Control Protocol) та інші. Переваги архітектури «клієнт-сервер» включають розподіл відповідальності між клієнтами та серверами, масштабованість, можливість оновлення або зміни серверної частини без впливу на клієнтів, а також централізоване управління даними та бізнес-логікою на сервері.

Layered Architecture (архітектура з шаруванням) [6] – це популярний підхід до організації програмного забезпечення, в якому компоненти системи розбиваються на окремі шари згідно з їхніми відповідальностями. Ідея за архітектурою з шаруванням полягає в тому, щоб розділити функціональність системи на логічно зв'язані шари, де кожен шар виконує конкретні завдання і має чітко визначені межі інтерфейсів. Кожен шар залежить лише від шарів, які знаходяться нижче в ієрархії. Зазвичай використовуються такі шари: Presentation Layer – цей шар відповідає за відображення даних користувачу та обробку його взаємодії з системою. Application Layer – ей шар містить бізнес-логіку та правила обробки даних. Domain Layer – цей шар представляє основний функціонал системи та включає моделі домену, бізнес-правила та логіку. Infrastructure Layer – цей шар забезпечує різноманітні технічні аспекти, такі як доступ до баз даних, зовнішні служби, комунікація по мережі, безпека тощо. Загалом, архітектура з шаруванням може мати різну кількість слоїв, яку обирають згідно з потребами конкретного програмного продукту.

Можна побачити, що у всіх трьох архітектурах використовується розділення відповідальностей між компонентами. Це дозволяє краще організувати роботу системи і спрощує розробку, тестування та підтримку коду. Всі три архітектури прагнуть до модульної структури, де різні компоненти можуть бути розроблені та підтримувані окремо. Це сприяє повторному використанню коду, зміні частин системи без впливу на інші компоненти та полегшує розподіл роботи між командами розробників. Усі три архітектури передбачають способи взаємодії між компонентами. Це може бути через виклики методів, обмін повідомленнями, передачу даних тощо.У всіх трьох архітектурах основний фокус знаходиться на відокремленні бізнес-логіки від інших аспектів програмного забезпечення, таких як інтерфейс користувача або доступ до даних. Незважаючи на ці спільні концепції, кожна з архітектур має свої особливості і призначення. Підсумуємо:

– клієнт-серверна архітектура: клієнти взаємодіють з серверами за допомогою мережевого з'єднання (рисунок 2.7);

– архітектура MVC: Розділення програми на Модель (бізнес-логіка), Представлення (візуалізація) та Контролер (керування взаємодією) (рисунок 2.8);

– архітектура з шаруванням (Layered Architecture): Розділення системи на логічні шари з різними відповідальностями (рисунок 2.9).

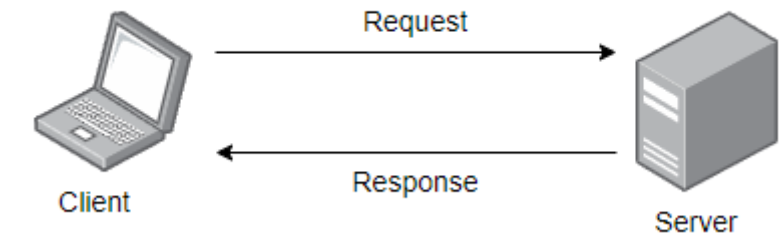


Рисунок 2.7 – Візуалізація клієнт-серверної архітектура

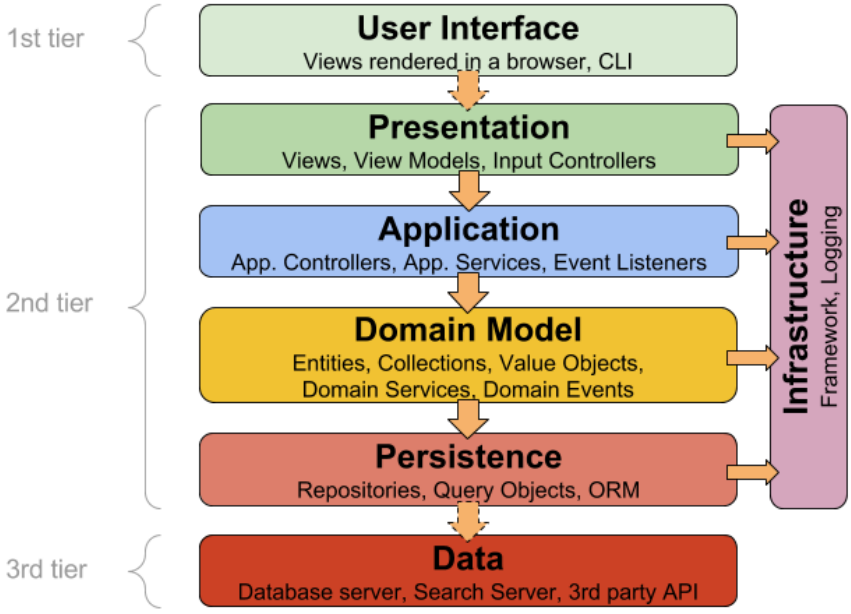


Рисунок 2.8 – Візуалізація архітектури з шаруванням

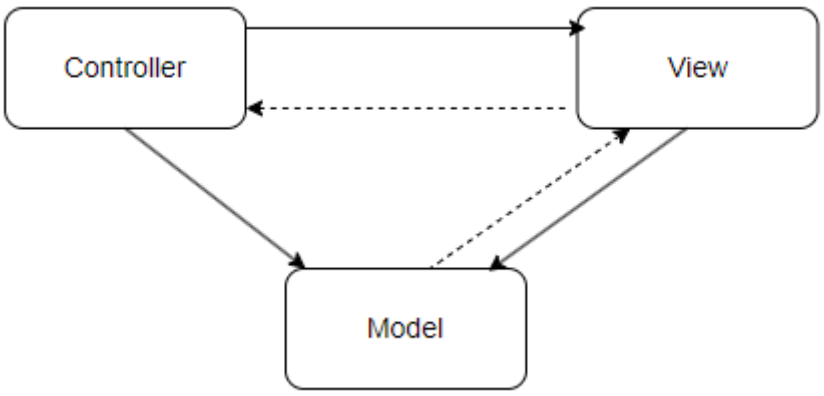


Рисунок 2.9 – Візуалізація архітектури MVC

Для веб-застосунку, який розглядається у даному дипломному проекті, було обрано клієнт-серверну архітектуру, а також архітектуру з шаруванням для реалізації серверної архітектури (Рисунок 2.10).

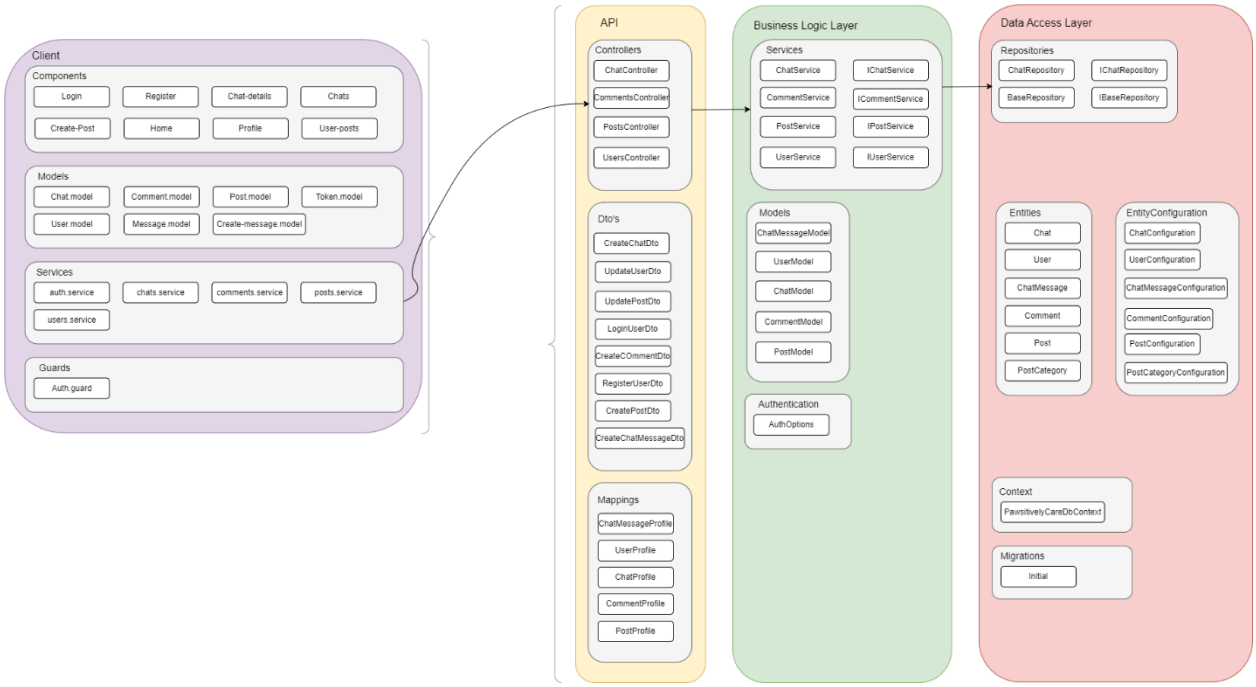


Рисунок 2.10 – Архітектура веб-застосунку з надання та отримання послуг для власників домашніх тварин

Веб-застосунок має серверну та клієнтську частини. Серверна частина організована з трьох шарів:

– Data Access Layer – це шар, в якому реалізовані операції доступу до даних, такі як збереження, отримання, оновлення або видалення даних з бази даних або іншого джерела даних. DAL відповідає за взаємодію з базою даних.

– Business Logic Layer – це шар, в якому реалізована бізнес-логіка програми. Він включає правила та логіку, які визначають, як програма обробляє дані, здійснює розрахунки та виконує бізнес-операції. BLL відповідає за забезпечення коректності та цілісності даних, а також за виконання бізнес-правил та процесів.

– API – це шар, який визначає інтерфейси та контракти для взаємодії між різними компонентами програми і між програмою та зовнішніми системами.

Клієнтська частина не має окремих слоїв, вона одночасно зберігає в собі і логіку клієнтської частини, її представлення і зв’язок з серверною частиною.

Для кращого розуміння структури проекту перед початком розробки і написання коду потрібно створити ER діаграму сутностей (Рисунок 2.11). ER-діаграма допомагає візуалізувати структуру даних та взаємозв'язки в інформаційній системі. З її допомогою можна легко розуміти структуру системи та взаємозв'язки між сутностями, що допомагає покращити процес розробки і управління даними.

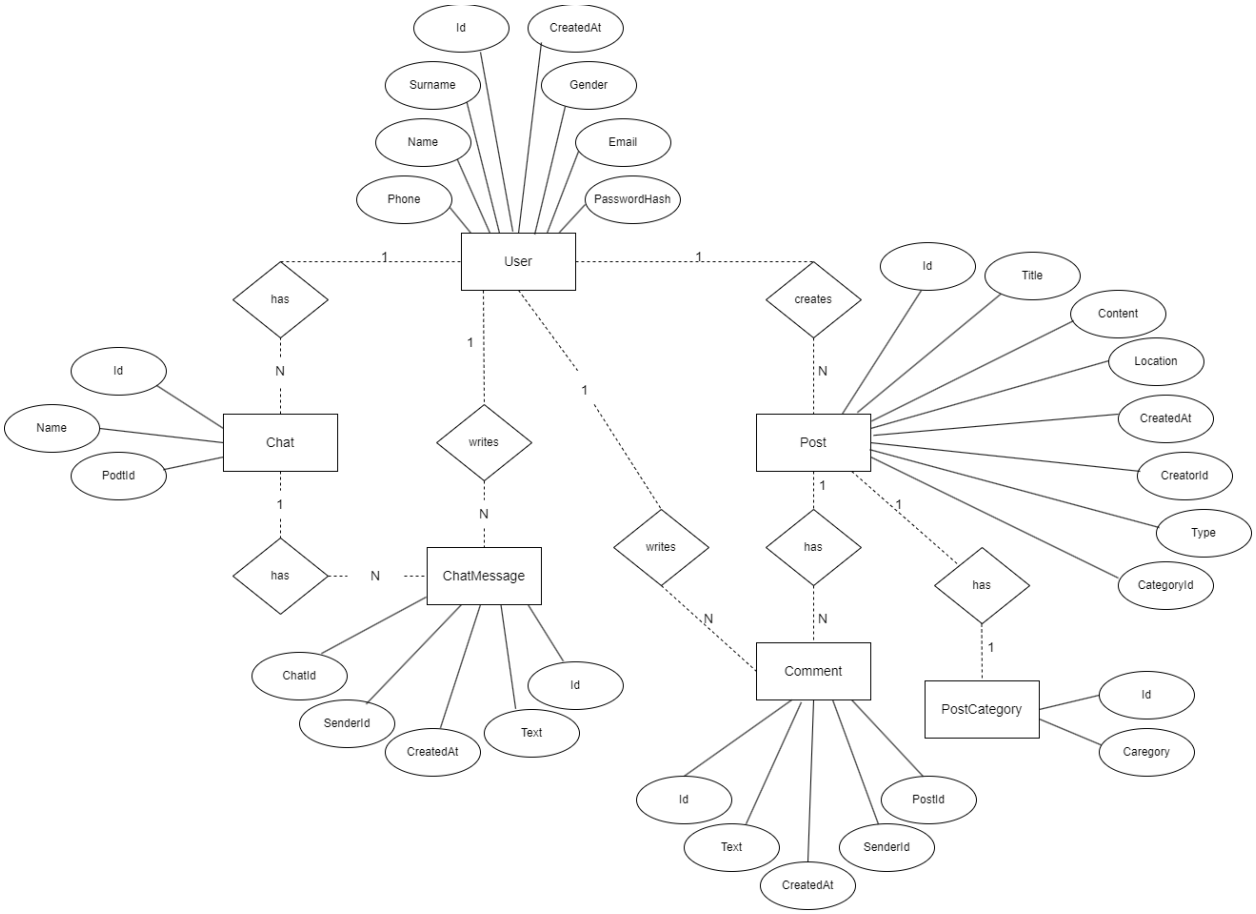


Рисунок 2.11 – ER діаграма сутностей User, Chats, ChatMessages, Posts, PostCategories, Comments

Проведемо опис усіх наведених сутностей, атрибутів та зв’язків.

Сутність User – представляє користувача додатку. У систему одночасно може увійти лише один користувач. Атрибути сутності User наведені в таблиці 2.1.

Таблиця 2.1 – Опис атрибутів сутності User

Назва	Опис
Id	Унікальний ідентифікатор сутності.
Name	Ім’я користувача.
Surname	Прізвище користувача.
Email	Електронна пошта користувача.
Phone	Номер телефону користувача.
Gender	Стать користувача.
PasswordHash	Захешований пароль користувача.
CreatedAt	Дата створення користувача.

Сутність Chat – представляє чат. Користувач може мати багато чатів. Чат може мати тільки два користувачі. Атрибути сутності Chat наведені в таблиці 2.2.

Таблиця 2.2 – Опис атрибутів сутності Chat

Назва	Опис
Id	Унікальний ідентифікатор сутності.
Name	Назва чату. Назва співзвучна з назвою поста, до якого був створений чат.
PostId	Ідентифікатор поста, до якого був створений чат.

Сутність ChatMessage – представляє повідомлення у чаті. Одне повідомлення може знаходитися лише в одному чаті, один чат може мати багато повідомлень. Атрибути сутності ChatMessage наведені в таблиці 2.3.

Таблиця 2.3 – Опис атрибутів сутності ChatMessage

Назва	Опис
Id	Унікальний ідентифікатор сутності.
Text	Текст повідомлення.
ChatId	Ідентифікатор чату, якому належить повідомлення.
SenderId	Ідентифікатор користувача, який надіслав повідомлення.
CreatedAt	Дата відправлення повідомлення.

Сутність Post – представляє пости. Один пост може мати лише одного користувача-творця. Один користувач може мати багато створених ним постів. Атрибути сутності Post наведені в таблиці 2.4.

Таблиця 2.4 – Опис атрибутів сутності Post

Назва	Опис
Id	Унікальний ідентифікатор сутності.
Title	Заголовок поста.
Content	Опис поста.
Location	Локація, в якій пост актуальний.
CreatedAt	Дата створення поста.
Type	Тип поста. Може бути лише два типа поста: від замовника та від виконавця.
CreatorId	Ідентифікатор користувача, який створив пост.
CategoryId	Ідентифікатор категорії, якій належить пост.

Сутність PostCategory – представляє категорії постів. Один пост може мати лише одну категорію, проте одна категорія може бути привласнена багатьма постами. Атрибути сутності PostCategory наведені в таблиці 2.5.

Таблиця 2.5 – Опис атрибутів сутності PostCategory

Назва	Опис
Id	Унікальний ідентифікатор сутності.
Category	Назва категорії.

Сутність Comment – представляє коментарі, які користувачі пишуть під постами. Один коментар належить лише одному посту. Один піст може мати багато коментарів. . Атрибути сутності Comment наведені в таблиці 2.6.

Таблиця 2.6 – Опис атрибутів сутності Comment

Назва	Опис
Id	Унікальний ідентифікатор сутності.
Text	Текст коментаря.
CreatedAt	Дата створення коментаря.
SenderId	Ідентифікатор користувача, який створив коментар.
PostId	Ідентифікатор поста, до якого був написаний комент.

2.3 Конструювання програмного забезпечення

Для реалізації серверної частини було обрано архітектуру з шаруванням (Layered Architecture). Опис основних інтерфейсів серверної частини веб-застосунку наведений у таблиці 2.7. Опис основних класів серверної частини веб-застосунку наведений у таблиці 2.8.

Таблиця 2.7 – Основні інтерфейси серверної частини

Назва Інтерфейсу	Опис
IChatService, ICommentService, IPostService, IUserService	Інтерфейси, які вказують контракт, який мають реалізувати відповідні сервіси, які реалізують один з інтерфейсів.

Продовження таблиці 2.7

IBaseRepository, IChatRepository	Інтерфейси, які вказують контракт, який мають реалізувати відповідні репозиторії, які реалізують один з інтерфейсів.

Таблиця 2.8 – Основні класи серверної частини

Назва Інтерфейсу	Опис
ChatsController	Клас, який представляє контролер, який відповідає за обробку запитів, пов'язаних із функціональністю чатів у веб-додатку.
CommentsController	Клас, який представляє контролер, який відповідає за обробку запитів, пов'язаних із функціональністю коментарів у веб-додатку.
PostsController	Клас, який представляє контролер, який відповідає за обробку запитів, пов'язаних із функціональністю постів у веб-додатку.
UsersController	Клас, який представляє контролер, який відповідає за обробку запитів, пов'язаних із функціональністю користувачів у веб-додатку.
Dto’s classes	Сукупність класів, які використовуються для отримання даних у контролерах з клієнтської частини.
Entities classes	Сукупність класів, які використовуються для відображення сутностей бази даних.
Mapping classes	Сукупність класів, які використовуються для перетворення даних з однієї структури чи типу на іншу.

Продовження таблиці 2.8

Program	Клас, який містить точку входу та визначає конфігурацію та налаштування програми.
AuthOptions	Клас, який налаштовує параметри аутентифікації.
ChatService	Клас, який виконує функціонал, пов'язаний з обробкою чатів і зв'язаної з ними бізнес-логіки.
CommentService	Клас, який виконує функціонал, пов'язаний з обробкою коментарів і зв'язаної з ними бізнес-логіки.
PostService	Клас, який виконує функціонал, пов'язаний з обробкою постів і зв'язаної з ними бізнес-логіки.
UserService	Клас, який виконує функціонал, пов'язаний з обробкою користувачів і зв'язаної з ними бізнес-логіки.
PawsitivelyCareDbContext	Клас, який відповідає за зв'язок з БД і виконання операцій збереження, отримання та модифікації даних у контексті веб-застосунку.
Entity configurations classes	Сукупність класів, які представляють конфігурації та налаштування сутностей у БД.
Migrations classes	Сукупність класів, ікі використовуються для керування версіями та структурою БД.
BaseRepository	Клас, який надає базові загальні методи та функціональність для доступу до даних у БД для усіх сутностей.

Продовження таблиці 2.8

ChatRepository	Клас, який надає базові загальні методи та функціональність для доступу до даних у БД для сутностей чатів.

База даних серверу призначена для зберігання користувачів, постів, категорій постів коментарів, чатів та повідомлень у чатах.

Microsoft SQL Server (MSSQL) [7] – це реляційна база даних, розроблена компанією Microsoft. З переваг даної бази можна виділити надійність, вона може обробляти великі обсяги даних і забезпечувати високу доступність та надійність сервісів бази даних. Також MSSQL має вбудовані механізми безпеки, такі як автентифікація, авторизація, шифрування даних і контроль доступу. MSSQL підтримує горизонтальне та вертикальне масштабування. Вона може обробляти великі обсяги даних та підтримувати високу продуктивність при зростанні навантаження.

Обрана база даних має багатий набір функціональних можливостей, таких як тригери, збережені процедури, функції, реплікація даних, кластеризація та інші. Це дає розробникам широкі можливості для реалізації бізнес-логіки та оптимізації роботи з даними. З недоліків можна виділити ліцензійні витрати: MSSQL є комерційним продуктом, тому для його використання потрібні ліцензії. Також MSSQL розроблена для платформи Windows і має обмежену підтримку для інших операційних систем, таких як Linux і macOS. Часом MSSQL може вимагати значних обчислювальних ресурсів і пам'яті для ефективної роботи з великими обсягами даних та складними запитами. Це може призвести до необхідності масштабування апаратного забезпечення для забезпечення оптимальної продуктивності.

Опис таблиць бази даних наведено у таблицях 2.9 - 2.15. Модель бази даних наведена на рисунку 2.12.

Таблиця 2.9 – Опис таблиці Users

Таблиця	Назва поля	Тип даних	Опис
Users	Id	uniqueidentifier	Ідентифікаційний номер користувача
	Name	nvarchar(20)	Ім’я користувача
	Surname	nvarchar(20)	Прізвище користувача
	Email	nvarchar(254)	Електронна пошта користувача
	Phone	nvarchar	Номер телефону користувача
	PasswordHash	nvarchar	Захешований пароль користувача
	CeatedAt	datetime	Дата реєстрації користувача у системі
	Gender	int	Стать користувача

Таблиця 2.10 – Опис таблиці Posts

Таблиця	Назва поля	Тип даних	Опис
Posts	Id	uniqueidentifier	Ідентифікаційний номер поста
	Title	nvarchar(100)	Заголовок поста
	Content	nvarchar	Опис поста

Продовження таблиці 2.10

Location	datetime	Локація, в якій пост є актуальним
CreatedAt	uniqueidentifier	Дата створення поста
CreatorId	uniqueidentifier	Ідентифікаційний номер користувача, який створив пост
Type	int	Тип поста
PostCategoryId	int	Ідентифікаційний номер категорії поста

Таблиця 2.11 – Опис таблиці PostCategories

Таблиця	Назва поля	Тип даних	Опис
PostCategories	Id	int	Ідентифікаційний номер категорії
	Category	nvarchar(50)	Назва категорії

Таблиця 2.12 – Опис таблиці Comments

Таблиця	Назва поля	Тип даних	Опис
Comments	Id	uniqueidentifier	Ідентифікаційний номер коментаря
	Text	nvarchar	Текст коментаря
	CreatedAt	nvarchar	Дата написання коментаря

Продовження таблиці 2.12

PostId	datetime	Ідентифікаційний номер поста, до якого написаний коментар
SenderId	uniqueidentifier	Ідентифікаційний номер користувача, який написав коментар

Таблиця 2.13 – Опис таблиці Chats

Таблиця	Назва поля	Тип даних	Опис
Chats	Id	uniqueidentifier	Ідентифікаційний номер чату
	Name	nvarchar	Назва чату
	PostId	nvarchar	Ідентифікаційний номер поста, до якого був створений чат

Таблиця 2.14 – Опис таблиці ChatMessages

Таблиця	Назва поля	Тип даних	Опис
ChatMessages	Id	uniqueidentifier	Ідентифікаційний номер повідомлення
	Text	nvarchar	Текст повідомлення
	CreatedAt	nvarchar	Дата відправлення повідомлення

Продовження таблиці 2.14

ChatId	datetime	Ідентифікаційний номер чату, в якому написане повідомлення
SenderId	uniqueidentifier	Ідентифікаційний номер користувача, який надіслав повідомлення

Таблиця 2.15 – Опис таблиці ChatUser

Таблиця	Назва поля	Тип даних	Опис
ChatUser	ChatId	uniqueidentifier	Ідентифікаційний номер чату
	UserId	uniqueidentifier	Ідентифікаційний номер користувача

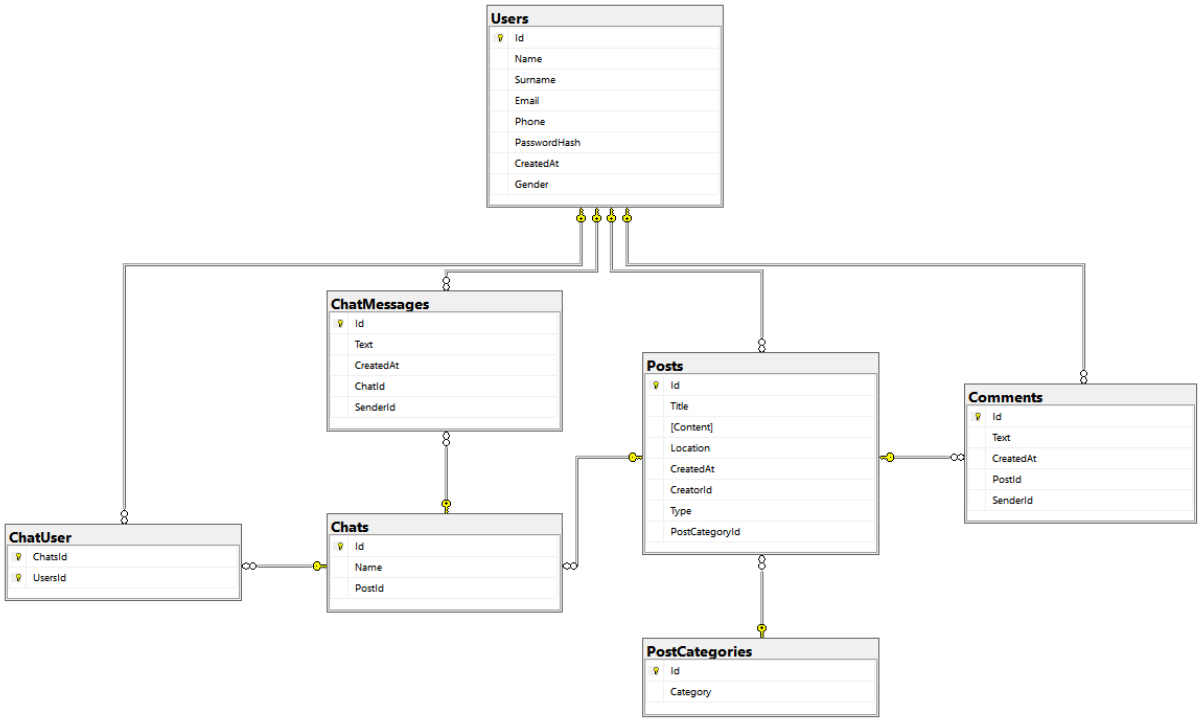


Рисунок 2.12 – Модель бази даних

Модель бази даних відрізняється від ER діаграми на рисунку 2.2 відсутністю таблиці ChatUser. Дана таблиця використовується для зв’язування таблиці Chats та Users зв’язком багато до багатьох.

Опис утиліт, бібліотек та іншого стороннього програмного забезпечення, що використовується у розробці наведено в таблиці 2.16.

Таблиця 2.16 – Опис утиліт

№ п/п	Назва утиліти	Опис застосування
1	Visual Studio	Головне середовище розробки програмного забезпечення серверної частини курсової роботи.
2	Visual Studio Code	Головне середовище розробки програмного забезпечення клієнтської частини курсової роботи.
3	ASP.NET Core	Фреймворк веб-розробки, надає розширені можливості для розробки сучасних веб-додатків, включаючи веб-сайти, веб-служби, API та додатки в режимі реального часу.
4	AndroidStudio	Інтегроване середовище розробки (IDE) для розробки мобільних додатків на платформі Android. Використовується для емулювання мобільного пристрою.
5	Postman	Програмне забезпечення необхідне для тестування REST запитів. Використовувалось для тестування API інтерфейсів, та клієнтських запитів.

Продовження таблиці 2.16

6	Swagger	Набір відкритих стандартів і інструментів, які допомагають описати, створити, візуалізувати та спілкуватися з веб-сервісами RESTful API.
7	Entity Framework	Об'єктно-реляційний маппер (ORM) для платформи .NET.
8	AutoMapper	Бібліотека для маппінгу об'єктів в платформі .NET. Вона дозволяє автоматично виконувати перетворення даних між об'єктами різних типів.
9	BCrypt-Net-Next	Бібліотека для хешування паролів в платформі .NET. Вона надає можливість застосовувати алгоритм хешування BCrypt до паролів користувачів.
10	Angular	Фреймворк для розробки веб-додатків, базується на мові програмування TypeScript і використовує HTML, CSS і JavaScript для створення динамічних і масштабованих додатків.
11	Capacitor	Веб-фреймворк, який дозволяє розробляти переносні мобільні додатки, використовуючи веб-технології, такі як HTML, CSS і JavaScript.
12	Ionic	Фреймворк для розробки мобільних додатків та гібридних веб-додатків, що працюють на різних мобільних платформах, таких як iOS та Android.

2.4 Аналіз безпеки даних

Безпека даних є важливим аспектом при створенні будь-якого застосунку. Безпека даних забезпечує конфіденційність чутливої інформації, такої як особисті дані користувачів, фінансові дані, медична інформація

тощо. Це важливо для збереження довіри користувачів і запобігання несанкціонованому доступу до цих даних. Безпека даних допомагає запобігти втраті даних через системні збої, помилки, атаки зловмисників.

Користувачі, які реєструються у системі, повинні мати пароль, який має зберігатися у базі даних не в оригінальному, а в захешованому вигляді [8]. Хешування паролів забезпечує захист від несанкціонованого доступу до паролів, якщо база даних або файл з хешами паролів стають доступними зловмисникам. Також хешування паролів ускладнює завдання перебору паролів зловмисниками. Хешування паролів також допомагає забезпечити цілісність даних: при перевірці введеного пароля, система порівнює хеш, отриманий з введеного пароля, збереженим хешем у базі даних. Якщо хеші співпадають, то пароль вважається правильним. Один з найголовніших плюсів хешування – воно є незворотним процесом, що означає, що неможливо отримати початковий пароль з хешу.

Для безпечної передачі інформації між сервером та клієнтом у застосунку будуть використовуватися JWT токени. JWT токени широко використовуються в сучасних веб-застосунках і мобільних додатках.

JWT токени використовуються для аутентифікації користувачів. При успішній аутентифікації сервер видаватиме JWT токен, який містить потрібні дані про користувача, такі як ідентифікатор користувача, роль та інші. Цей токен можна передавати з кожним запитом на сервер для підтвердження ідентифікації користувача. У JWT токені можна включити інформацію про ролі та права доступу користувача. JWT токени можуть бути підписані цифровим підписом, що дозволяє перевірити їх цілісність та автентичність. Це забезпечує, що дані в токені не були змінені під час передачі.

Висновки до розділу

У другому розділі дипломного проекту було виконано моделювання та аналіз програмного забезпечення. У розділі були описані бізнес процеси застосунку, для кожного з яких були створені BPMN моделі. Було проведено

аналіз популярних архітектур програмного забезпечення і обрано аріхтектуру, яка буде застосована в веб-застосунку, який розглядається у даній дипломній роботі.

Для кращого розуміння структури проекту перед початком розробки і написання коду було створена ER діаграма сутностей, після чого було проведено опис усіх наведених у діаграмі сутностей, атрибутів та зв’язків.

У розділі було описано обрану базу даних, наведено її плюси та недоліки. Була надана модель бази даних.

Також у розділі було наведено опис утиліт, бібліотек та іншого стороннього програмного забезпечення, що буде використовуватися у розробці.

Була розглянута безпека даних, яка повинна бути реалізована у веб-застосунку з надання та отримання послуг для власників домашніх тварин.

3 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Аналіз якості ПЗ

Аналіз якості ПЗ є важливою складовою процесу розробки програмного продукту. Метою аналізу якості ПЗ є оцінка відповідності ПЗ визначеним вимогам, виявлення дефектів і недоліків, а також вдосконалення якості продукту.

Аналіз якості допомагає виявити дефекти в програмному продукті, такі як помилки, неочікувана поведінка, використання антипатернів і т.д. Виявлення дефектів на ранніх етапах розробки дозволяє їх виправити перед випуском продукту, що знижує ризик негативних наслідків для користувачів.

Також аналіз якості допомагає визначити, наскільки продукт відповідає вимогам і потребам користувачів. Виявлення можливих недоліків у функціональності дозволяє розробникам внести зміни та вдосконалити продукт для забезпечення кращого досвіду користувача.

Аналіз якості включає оцінку продуктивності програмного продукту, таку як швидкодія, реакція на велику кількість запитів і обробка даних. Це дозволяє виявити можливі проблеми з продуктивністю і провести оптимізацію для забезпечення ефективної роботи програмного продукту.

Крім того, аналіз якості включає перевірку стійкості програмного продукту до непередбачених ситуацій і помилок. Це дозволяє виявити можливі проблеми, які можуть призвести до збоїв або втрати даних.

Загалом, аналіз якості ПЗ допомагає забезпечити, що програмний продукт відповідає вимогам, функціонує коректно, є надійним і надає задоволення користувачам.

Щоб проаналізувати якість коду веб-застосунку даної дипломної роботи було використано сервіс Embold. Даний сервіс надає інструменти для аналізу якості програмного коду. Embold використовує штучний інтелект та

аналітичні алгоритми для виявлення проблем і вдосконалення якості програмного забезпечення.

Після аналізу коду маємо результати– рисунок 3.1.

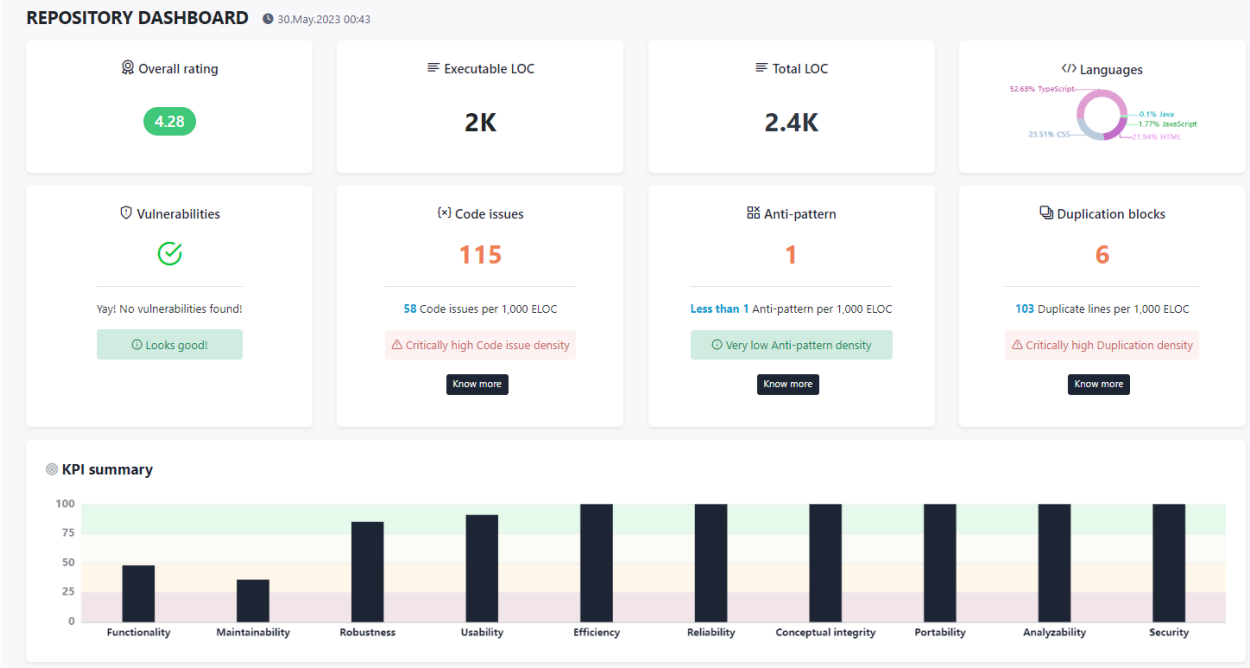


Рисунок 3.1 – Результати аналізу коду проекту

Код проекту було проаналізовано за декількома критеріями: функціональність, ремонтопридатність, надійність, зручність використання, ефективність, надійність, концептуальна цілісність, портативність, можливість аналізу і безпека.

Аналізатор коду виконує детальний аналіз кожного файлу у проекті для оцінки якості його коду шляхом перевірки коду на наявність потенційних проблем, таких як безпекові вразливості, дублікація коду, наявність багів та інше. Після того, як аналізатор коду провів аналіз усіх файлів, вираховується середня оцінка якості коду. Код проекту отримав оцінку у 4.28 балів. Підводячи підсумок, код проекту можна вважати якісним.

3.2 Опис процесів тестування

Існує багато видів тестування ПЗ, які використовуються для перевірки різних аспектів його якості і функціональності. Мануальне тестування [9] - це

процес перевірки програмного забезпечення людським тестувальником без використання автоматизованих інструментів чи скриптів. Воно включає в себе ручне виконання тестових сценаріїв, введення даних та оцінку результатів. Мануальне тестування може бути легко адаптоване до змін у вимогах та функціоналу програмного забезпечення, тестувальники можуть змінювати сценарії тестування та виконувати непередбачувані дії, що дозволяє виявляти непередбачувані проблеми.

Мануальне тестування дозволяє тестувальникам виконувати тестування, коли вони активно досліджують програмне забезпечення, виявляють проблеми, які не були задокументовані та передбачені.

Серед мінусів мануального тестування можна виділити часові затрати, таке тестування вимагає багато часу та зусиль для виконання повторних тестів та перевірки різних сценаріїв. Також, оскільки мануальне тестування залежить від людського фактору, існує підвищений ризик помилок. Деякі види тестування, особливо ті, що потребують великої кількості даних або довгих процесів, можуть бути важкими або неможливими для виконання вручну. Це обмежує можливості мануального тестування в повній автоматизації процесу тестування. Результати мануального тестування можуть бути суб'єктивними, оскільки вони залежать від індивідуального рівня знань, досвіду та експертизи тестувальника. Різні тестувальники можуть мати різні підходи та інтерпретацію результатів, що може призвести до непродуктивного суперечливого процесу.

Для тестування програмного забезпечення, яке розглядається у даній дипломній роботі, було обрано мануальне тестування. Опис опис відповідних тестів наведено у таблицях 3.1 – 3.9.

Таблиця 3.1 – Тест 1.1

Тест	Реєстрація користувача
Модуль	Реєстрація користувача
Номер тесту	1.1
Початковий стан системи	Користувач знаходиться на сторінці реєстрації
Вхідні данні	Ім’я, прізвище, номер телефону, електронна пошта, пароль
Опис проведення тесту	Користувач вводить ім’я, прізвище, номер телефону, коректна електронна пошта, яка до цього не була зареєстрована в системі, пароль у відповідні поля. Після цього користувач натискає кнопку підтвердження реєстрації
Очікуваний результат	Висвічується повідомлення про успішну реєстрацію користувача у системі. Відкривається сторінка входу у систему
Фактичний результат	Висвічується повідомлення про успішну реєстрацію користувача у системі. Відкривається сторінка входу у систему

Таблиця 3.2 – Тест 2.1

Тест	Вхід до застосунку
Модуль	Вхід до застосунку
Номер тесту	2.1
Початковий стан системи	Користувач знаходиться на сторінці входу до системи
Вхідні данні	Електронна пошта, пароль

Продовження таблиці 3.2

Опис проведення тесту	У відповідні поля вводяться: електронна пошта і пароль Після цього натискається кнопка підтвердження входу до системи
Очікуваний результат	Відкривається сторінка з постами від інших користувачів
Фактичний результат	Відкривається сторінка з постами від інших користувачів

Таблиця 3.3 – Тест 3.1

Тест	Відгук на пост
Модуль	Пости
Номер тесту	2.1
Початковий стан системи	Користувач знаходиться на сторінці з постами від інших користувачів
Вхідні данні	-
Опис проведення тесту	Користувач натискає на кнопку «Respond» під потрібним постом
Очікуваний результат	Користувач переходить до чату з користувачем, який створив пост, під яким було натиснуто кнопку «Respond»
Фактичний результат	Користувач переходить до чату з користувачем, який створив пост, під яким було натиснуто кнопку «Respond»

Таблиця 3.4 – Тест 3.2

Тест	Написання коментаря до поста
Модуль	Пости
Номер тесту	2.1
Початковий стан системи	Користувач знаходиться на сторінці з постами від інших користувачів
Вхідні данні	Коментар до поста
Опис проведення тесту	Користувач натискає на кнопку «Add Comment» під потрібним постом. Відкривається сторінка для введеня коментаря. Користувач вводить текст у відповідне поле і тисне на кнопку «Save»
Очікуваний результат	Користувачу показується повідомлення про успішно доданий коментар. Коментар відображається у коментарях під відповідним постом
Фактичний результат	Користувачу показується повідомлення про успішно доданий коментар. Коментар відображається у коментарях під відповідним постом

Таблиця 3.5 – Тест 3.3

Тест	Перегляд коментарів
Модуль	Пости
Номер тесту	2.1
Початковий стан системи	Користувач знаходиться на сторінці з постами від інших користувачів
Вхідні данні	-
Опис проведення тесту	Користувач натискає на кнопку «Comments» під потрібним постом

Продовження таблиці 3.5

Очікуваний результат	Під постом розгортаються коментарі, які були написані до поста
Фактичний результат	Під постом розгортаються коментарі, які були написані до поста

Таблиця 3.6 – Тест 4.1

Тест	Написання повідомлення до чату
Модуль	Чати
Номер тесту	2.1
Початковий стан системи	Користувач знаходиться на сторінці з постами від інших користувачів
Вхідні данні	Повідомлення
Опис проведення тесту	Користувач робить свайп вліво, після чого натискає на «Chats» у меню. Відкривається сторінка з чатами, які має користувач. Користувач натискає на чат у списку чатів. Відкривається вікно чату. Користувач вводить текст у поле знизу вікна і тисне на кнопку із зображенням стрілки справа від поля
Очікуваний результат	Написане повідомлення показано у чаті зліва. Повідомлення має у собі текст, який написав користувач, та дату написання
Фактичний результат	Написане повідомлення показано у чаті зліва. Повідомлення має у собі текст, який написав користувач, та дату написання

Таблиця 3.7 – Тест 5.1

Тест	Створення поста
Модуль	Пости користувача, який залогінений у системі
Номер тесту	2.1
Початковий стан системи	Користувач знаходиться на сторінці з постами від інших користувачів
Вхідні данні	Тип поста, категорія поста, заголовок, опис, локація, зображення
Опис проведення тесту	Користувач робить свайп вліво, після чого натискає на «Create Post» у меню. Відкривається сторінка для створення поста. Користувач вводить дані у відповідні поля. Користувач може додати зображення до поста при натисканні на кнопку «Add Image». Після введення усіх даних користувач тисне кнопку «Create».
Очікуваний результат	Створений пост відображається у вкладці «My Posts»
Фактичний результат	Створений пост відображається у вкладці «My Posts»

Таблиця 3.8 – Тест 5.2

Тест	Видалення поста
Модуль	Пости користувача, який залогінений у системі
Номер тесту	2.1
Початковий стан системи	Користувач знаходиться на сторінці з постами від інших користувачів
Вхідні данні	-

Продовження таблиці 3.8

Опис проведення тесту	Користувач робить свайп вліво, після чого натискає на «My Posts» у меню. Користувач переходить на сторінку з постами, які були ним створені. Користувач тисне під постом на червону кнопку із зображенням смітника
Очікуваний результат	Пост, під яким була натиснута кнопка із зображенням смітника, зникає із вкладки «My Posts».
Фактичний результат	Пост, під яким була натиснута кнопка із зображенням смітника, зникає із вкладки «My Posts».

Таблиця 3.9 – Тест 6.1

Тест	Редагування даних профілю користувача
Модуль	Профіль користувача
Номер тесту	2.1
Початковий стан системи	Користувач знаходиться на сторінці з постами від інших користувачів
Вхідні данні	Ім’я користувача, прізвище, електронна пошта, номер телефону, пароль
Опис проведення тесту	Користувач робить свайп вліво, після чого натискає на «Profile» у меню. Користувач переходить на сторінку з даними профілю. Користувач тисне на кнопку «Edit». Користувач вводить потрібні дані у відповідні поля і тисне «Save».
Очікуваний результат	Дані користувача успішно змінені. Нові дані відображаються у профілі користувача
Фактичний результат	Дані користувача успішно змінені. Нові дані відображаються у профілі користувача

Можна зробити висновок, що усі функції веб-застосунку були успішно протестовані. Усі тести показали очікувані результати.

3.3 Опис контрольного прикладу

Проведемо опис детального контрольного прикладу, у якому відобразимо всі можливості та особливості веб-застосунку.

Спочатку користувач знаходиться на сторінці входу в застосунок. Щоб зареєструватися, користувач повинен натиснути на кнопку «Register». У вікні реєстрації користувач повинен ввести відповідні дані та натиснути на «Register» – рисунок 3.2. Якщо при спробі реєстрації виникла помилка, наприклад, якщо у базі даних вже існує користувач з введеною поштою, показується повідомлення про помилку – рисунок 3.3. Якщо реєстрація пройшла успішно, користувач повертається на сторінку входу в застосунок, користувачу показується повідомлення про успішну реєстрацію в застосунку – рисунок 3.4.

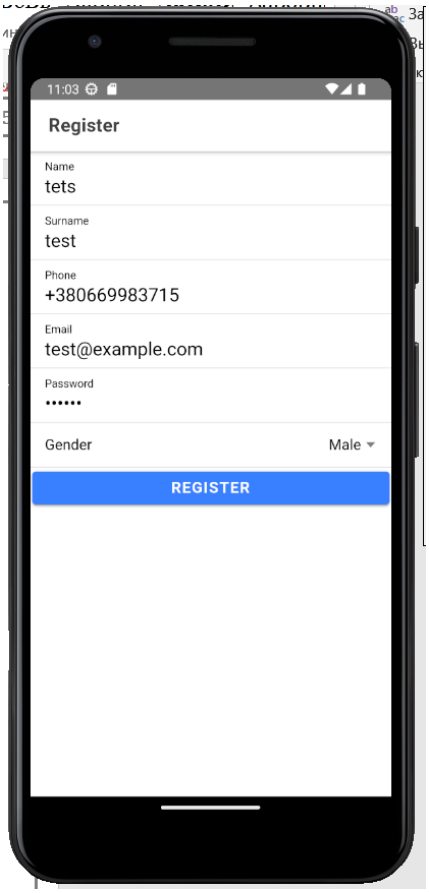


Рисунок 3.2 – Сторінка реєстрації

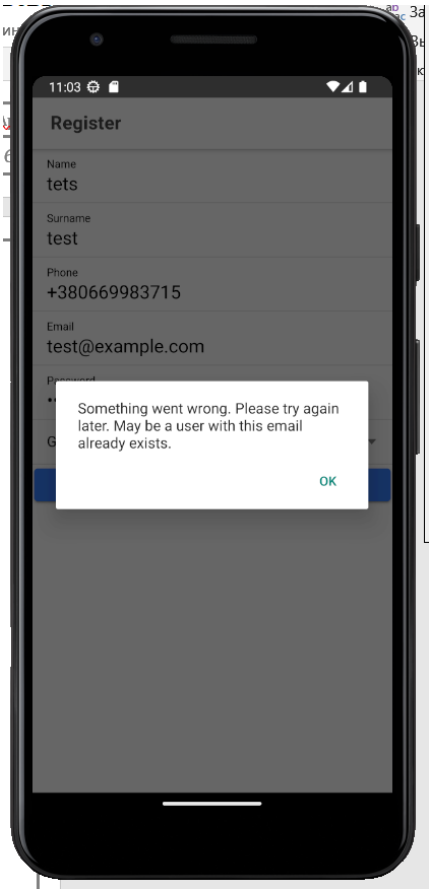


Рисунок 3.3 – Помилка при спробі зареєструватися у застосунку

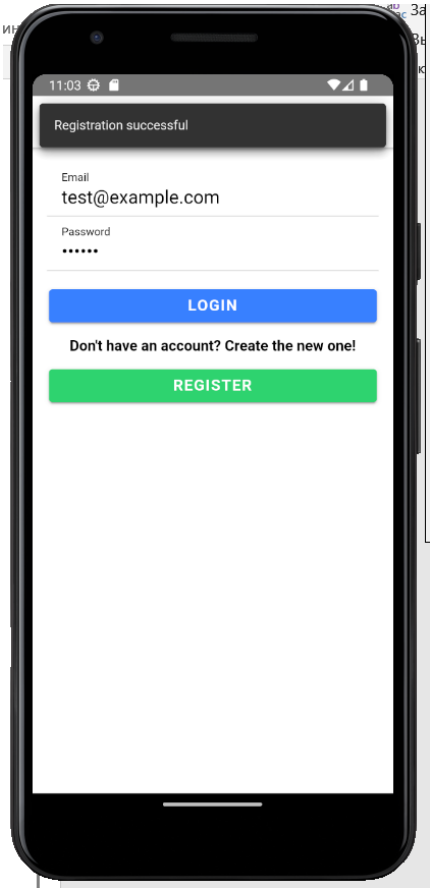


Рисунок 3.4 – Повідомлення про успішну реєстрацію у застосунку

Щоб увійти у застосунок, користувач повинен ввести свою електронну пошту па пароль – рисунок 3.5. Якщо користувач ввів не коректні дані, показується повідомлення – рисунок 3.6. Після успішного входу в застосунок відкривається сторінка з постами від інших користувачів – рисунок 3.7.

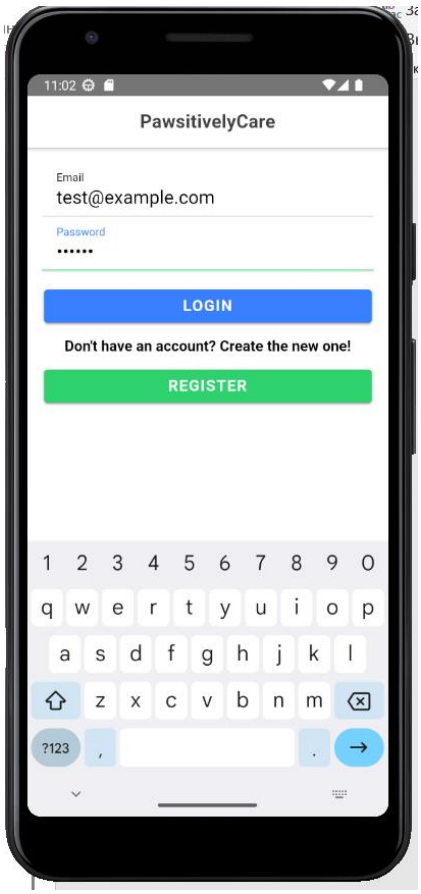


Рисунок 3.5 – Сторінка входу в застосунок

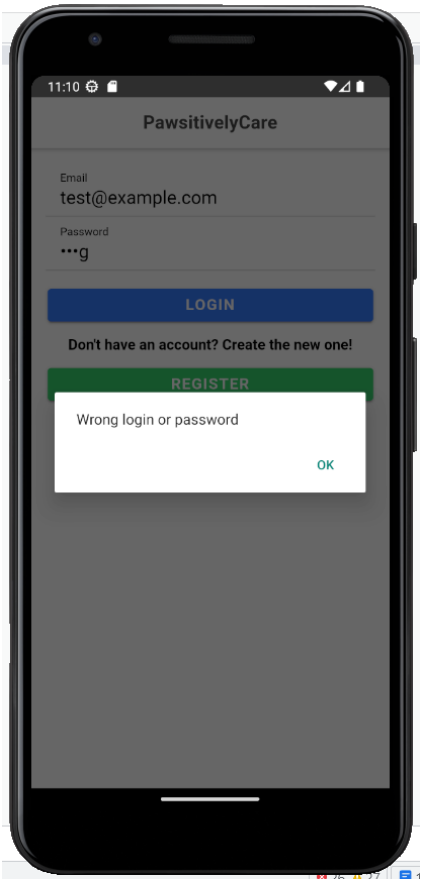


Рисунок 3.6 – Помилка при введенні некоректних даних

Рисунок 3.7 – Успішний вхід в систему. Відкривається сторінка з постами

Користувач може відгукнутися на пост натиснувши на кнопку «Respond», після чого перейде до чату з користувачем, який створив відповідний пост – рисунок 3.8.

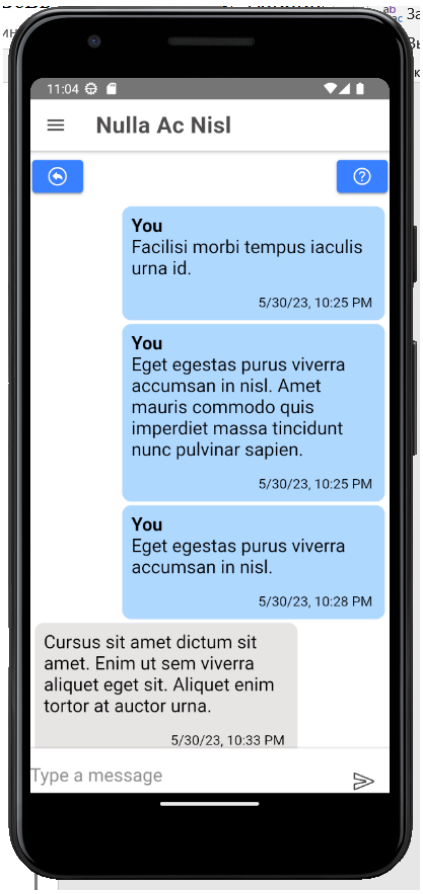


Рисунок 3.8 – Чат, який відкрився при натисканні на кнопку «Respond»

Користувач може подивитися коментарі під постом, натиснувши на «Comments» пост – рисунок 3.9. Щоб написати коментар, користувач повинен натиснути на «Add Comment», після чого відкриється сторінка, де користувач може написати коментар. Щоб скасувати коментар, користувач повинен натиснути на «Cancel». Щоб надіслати коментар, користувач повинен натиснути на «Save» – рисунок 3.10.

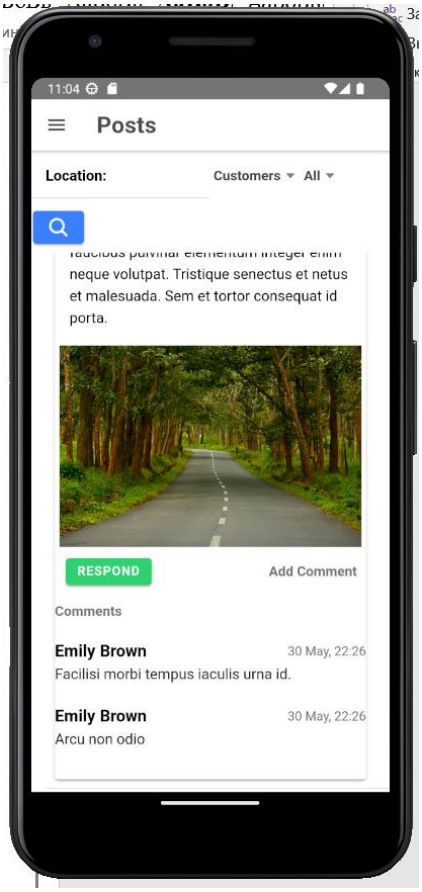


Рисунок 3.9 – Відкриті коментарі під постом

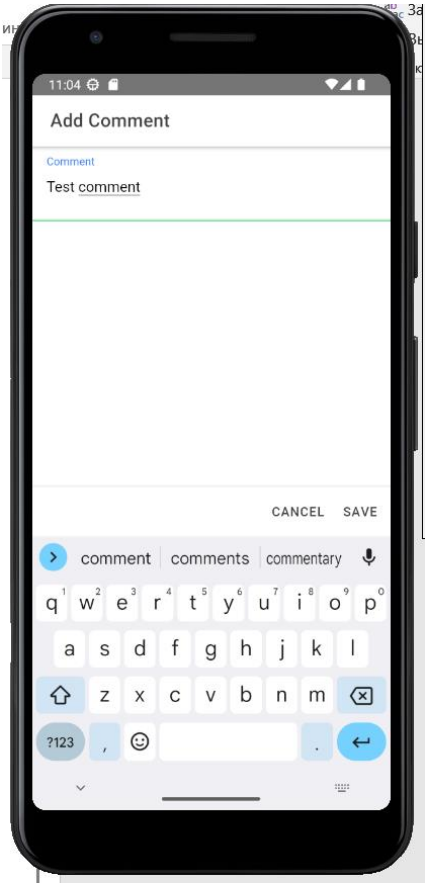


Рисунок 3.10 – Сторінка, де користувач може написати коментар

Користувач може відкрити меню при натисканні на кнопку з трьома рисками або при свайпі вліво – рисунок 3.11. Щоб перейти на потрібну сторінку, користувач повинен натиснути на відповідний пункт в меню.

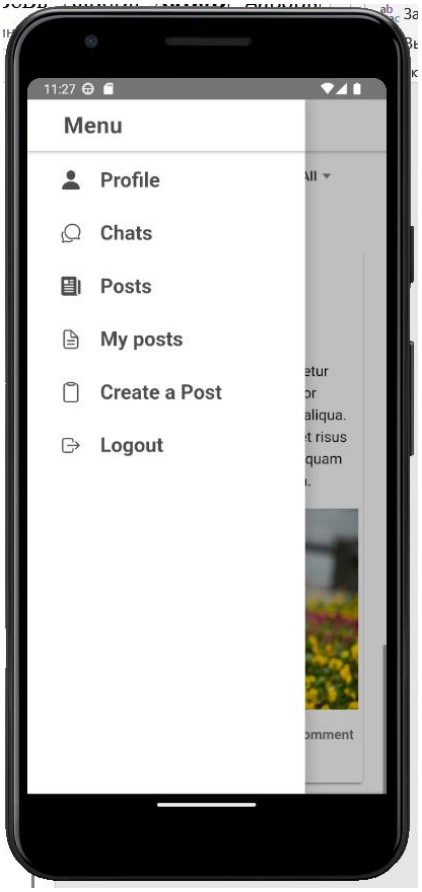


Рисунок 3.11 – Відкрите меню додатку

Користувач може переглянути свій профіль при натисканні на «Profile» у меню додатку – рисунок 3.12. Щоб відредагувати дані профілю, користувач повинен натиснути на «Edit», після чого відкриється сторінка редагування профілю – рисунок 3.13. Щоб зберегти відредаговані дані, треба натиснути на «Save».

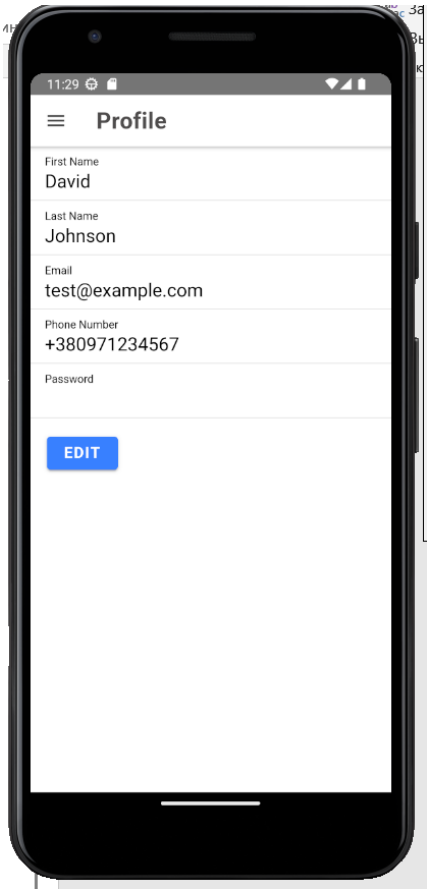


Рисунок 3.12 – Профіль користувача

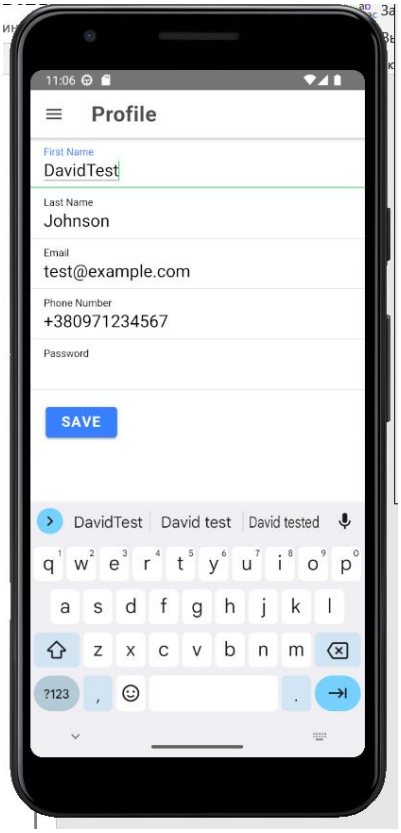


Рисунок 3.13 – Редагування профілю користувача

Користувач може переглянути свій чати при натисканні на «Chats» у меню додатку – рисунок 3.14. При натисканні на чат відкривається вікно з повідомленнями у чаті. Користувач може написати своє повідомлення у поле знизу та відправити його при натисканні на кнопку у вигляді стрілки – рисунок 3.15. Користувач може подивитися деталі поста, до якого був створений чат, при натисканні на кнопку зі знаком «?» – рисунок 3.16.

Рисунок 3.14 – Сторінка з чатами користувача

Рисунок 3.15 – Вікно з чатом користувача

Рисунок 3.16 – Деталі поста, до якого був створений чат

Користувач може переглянути свій пости при натисканні на «My Posts» у меню додатку – рисунок 3.17. Користувач може відредагувати пост при натисканні на кнопку із зображенням олівця – рисунок 3.18. Щоб видалити пост, користувач повинен натиснути на червону кнопку із зображенням смітника.

Користувач може створити новий пост при натисканні на кнопку зі знаком «+» – рисунок 3.19. Така сама сторінка відкривається при натисканні на «Create Post» у меню додатку.

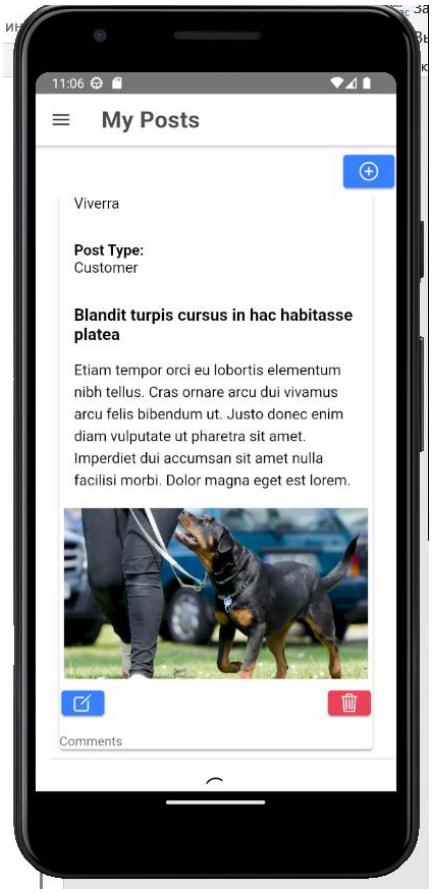


Рисунок 3.17 – Сторінка з постами користувача

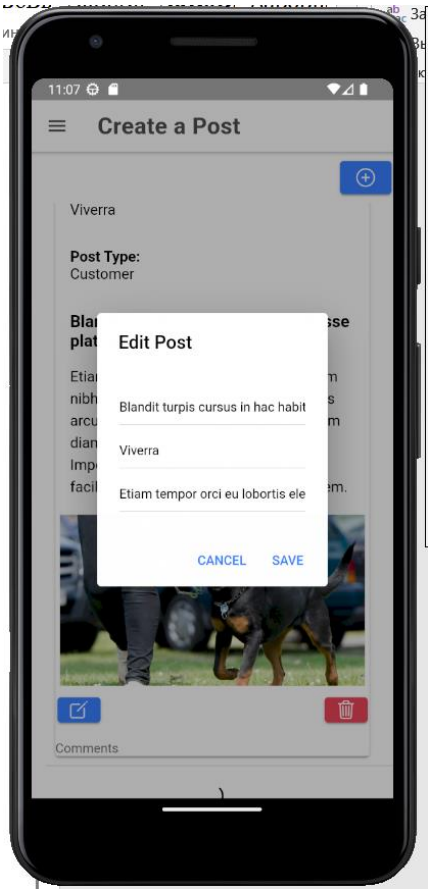


Рисунок 3.18 – Віконце редагування поста

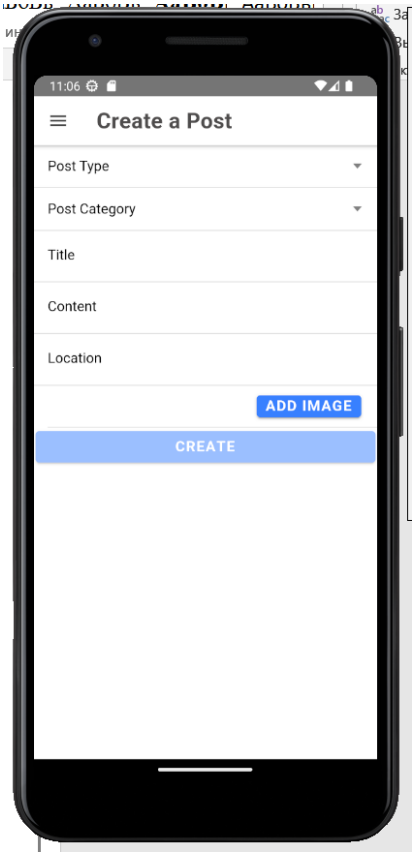


Рисунок 3.19 – Сторінка створення поста

Висновки до розділу

У даному розділі дипломного проекту було здійснено оцінку якості коду серверної та клієнтської частин, а також здійснено мануальне тестування веб-застосунку.

Для оцінки якості коду було використано сервіс Embold. Даний сервіс надає інструменти для аналізу якості програмного коду. Embold використовує штучний інтелект та аналітичні алгоритми для виявлення проблем і вдосконалення якості програмного забезпечення.

Під час мануального тестування було перевірено функціональність та коректність роботи всіх функцій програмного застосунку. Метод мануального тестування був обраний, оскільки, по-перше, функції застосунку тестувалися під час написання коду. По-друге, цей спосіб тестування дозволить уявити себе на місці користувача, що дозволить змоделювати поведінку справжнього юзера і виявити слабкості та баги застосунку. Таким чином, мануальне тестування вимагало перевірки функцій та їх взаємодії з метою підтвердження їх коректності та оптимальності роботи. В результаті аналізу коду та мануального тестування було забезпечено перевірку якості та функціональності програмного застосунку, що є важливими кроками у забезпеченні його успішного розгортання та використання.

4 ВПРОВАДЖЕННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Розгортання програмного забезпечення

Для хостингу серверної частини було обрано Heroku – це хмарна платформа, яка надає можливість розгортання та керування додатками. Вона спрощує процес розгортання та масштабування серверних додатків. Після створення облікового запису на heroku.com, потрібно створити новий додаток на платформі Heroku. Після цього буде отримано унікальне ім'я та URL для веб-застосунку. Після цього потрібно завантажити код серверної частини на Heroku, використовуючи інтеграцію з відповідним репозиторієм. Heroku автоматично забезпечує розгортання та запуск додатку. Також потрібно налаштувати необхідні змінні середовища на Heroku, такі як з'єднання з базою даних, секретні ключі або параметри конфігурації.

Клієнтська частина буде розгорнута в Google Play Store – це найбільша платформа для розповсюдження мобільних додатків для Android-пристроїв. Після реєстрації облікового запису розробника на Google Play Console потрібно створити новий проект для веб-застосунку. Потім потрібно завантажити скомпільований веб-застосунку, написаного на Angular Ionic, на Google Play Console. Потребується надання необхідних метаданих, таких як назва, опис, зображення та інші деталі, що стосуються застосунку. Також перед розміщенням додатку потрібно перевірити його на відповідність вимогам Google Play, зокрема стосовно дизайну, функціональності та політики конфіденційності. Після цього потрібно надіслати додаток на перевірку до Google. Процес перевірки займає певний час, під час якого виконується оцінка відповідності вимогам Google Play, а також перевіряється наявність будь-яких проблем або порушень політики. Якщо застосунок пройшов перевірку без проблем, його можна опублікувати в Google Play Store. Потрібно обрати налаштування для додатку, такі як категорія та країни, в яких додаток має бути доступним.

Після публікації застосунку його можна знайти в Google Play Store. Після розгортання вашого додатку в Google Play Store можна продовжувати вносити зміни та оновлювати його, щоб поліпшити функціональність, виправити помилки або додати нові функції.

4.2 Підтримка програмного забезпечення

Користувачі повинні мати можливість завантажити нову та актуальну версію мобільного застосунку або оновити свою поточну версію до останньої.

Щоб оновити мобільний застосунок, потрібно внести необхідні зміни та підготувати нову версію застосунку. Після цього потрібно завантажити оновлену версію застосунку у магазин Google Play. Для цього потрібно увійти до консолі розробника Google Play та обрати свій застосунок. У розділі «Випуски» обрати «Новий випуск» і завантажити новий APK-файл, який містить оновлену версію мобільного застосунку. Після завантаження потрібно надати необхідну інформацію про нову версію, таку як номер версії, опис внесених змін та змінений список функцій. Після введення всієї необхідної інформації, обрати «Опублікувати», щоб надіслати оновлену версію мобільного застосунку на перевірку в Google Play. Після успішної перевірки нова версія буде доступна користувачам через магазин Google Play, де вони зможуть завантажити або оновити застосунок до найновішої версії.

Висновки до розділу

У розділі було розглянуто процес поширення веб-застосунку через магазин додатків Google Play Store. Поширення застосунку вимагає кількох кроків для розгортання і оновлення як серверної, так і клієнтської частини.

Для серверної частини, було обрано Heroku як хмарну платформу, яка спрощує процес розгортання та керування додатками. Щоб розгорнути серверний додаток на Heroku, потрібно створити обліковий запис, створити

новий додаток, завантажити код на платформу та налаштувати необхідні змінні середовища.

У разі клієнтської частини, Google Play Store є найбільшою платформою для розповсюдження мобільних додатків для Android-пристроїв. Щоб розгорнути клієнтський застосунок в Google Play Store, потрібно створити обліковий запис розробника, створити новий проект, завантажити скомпільований код на платформу та надати необхідну метаінформацію про додаток. Після перевірки та відповідності вимогам Google Play, додаток можна опублікувати в магазині.

Також було розглянуто шлях оновлення версії мобільного застосунку в Google Play Store. Для оновлення застосунку потрібно підготувати нову версію з внесеними змінами, надіслати її на перевірку та після успішної перевірки опублікувати для користувачів. В процесі оновлення необхідно надати інформацію про зміни, виправлення помилок та нові функції. В результаті, процес розгортання та оновлення мобільного застосунку в магазині Google Play Store вимагає кількох кроків, включаючи створення облікових записів розробника, завантаження коду та перевірку відповідності вимогам та публікацію додатку для користувачів.

Завдяки процесу розгортання та оновлення через магазини додатків, розробники отримують можливість ефективно поширювати свої застосунки серед широкої аудиторії користувачів. Користувачі, у свою чергу, мають зручний спосіб завантажувати та оновлювати застосунки безпосередньо з офіційних магазинів, що забезпечує їм безпеку та надійність.

ВИСНОВКИ

В результаті виконання дипломного проєкту було спроєктовано веб-застосунок з надання та отримання послуг для власників домашніх тварин. Цей застосунок має на меті надати користувачам можливість легко та швидко отримувати та надавати послуги в сфері догляду за домашніми тваринами. За допомогою цього додатку, користувачі зможуть публікувати послуги чи заявки на отримання послуги, переглядати послуги і заявки на отримання послуг, фільтрувати послуги за критеріями, такими як локація, типи, категорія та іншими, додавати та переглядати коментарі до послуг, відгукуватися на пост с подальшим контактом с замовником або виконавцем, підтримувати контакт з замовником або виконавцем, редагувати дані свого профілю.

У якості середовища розробки було обрано Visual Studio для написання серверної частини та Visual Studio Code для написання клієнтської частини.

У якості БД було використано MSSQL – це реляційна база даних, розроблена компанією Microsoft. З переваг даної бази можна виділити надійність, вона може обробляти великі обсяги даних і забезпечувати високу доступність та надійність сервісів бази даних.

Після реалізації застосунку він було проведено мануальне тестування – це процес перевірки програмного забезпечення людським тестувальником без використання автоматизованих інструментів чи скриптів. Воно включає в себе ручне виконання тестових сценаріїв, введення даних та оцінку результатів.

Новизна роботи полягає в унікальній вузькоспеціалізованості розробленого веб-застосунку, оскільки наразі в українському сегменті не існує окремого застосунку, який був би спеціалізований під отримання і надання послуг, пов’язаних з домашніми тваринами. Зазвичай ці послуги можна знайти лише як частину багатьох інших послуг. Окремими є лише

ветеринарні застосунки та застосунки та магазини, в яких можна купити речі чи харчування для тварин.

У процесі розробки та тестування застосунку вдалося успішно досягти всіх поставлених перед проектом цілей.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1) Мова програмування C#: Microsoft Developer Network (MSDN) [Електронний ресурс] — Режим доступу: https://learn.microsoft.com/en-us/dotnet/csharp/

2) Entity Framework: Microsoft Entity Framework Documentation - [Електронний ресурс] — Режим доступу: https://learn.microsoft.com/en-us/ef/

3) Платформа ASP.NET Core: Microsoft Documentation [Електронний ресурс] — Режим доступу: https://learn.microsoft.com/en-us/aspnet/core/?view=aspnetcore-7.0

4) Microsoft Documentation: ASP.NET MVC - Model-View-Controller (MVC) [Електронний ресурс] — Режим доступу: https://learn.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-7.0

5) Techopedia: Client-Server Architecture [Електронний ресурс] — Режим доступу: https://www.techopedia.com/definition/438/clientserver-architecture

6) Layered Architecture [Електронний ресурс] — Режим доступу: https://www.baeldung.com/cs/layered-architecture

7) Microsoft Corporation: Microsoft SQL Server [Електронний ресурс] — Режим доступу: https://www.microsoft.com/en-us/sql-server/

8) NIST Special Publication 800-63B: Digital Identity Guidelines - Authentication and Lifecycle Management [Електронний ресурс] — Режим доступу: https://pages.nist.gov/800-63-3/sp800-63b.html

9) Software Testing Help: Pros and Cons of Manual Testing [Електронний ресурс] — Режим доступу: https://www.softwaretestinghelp.com/manual-testing-tutorial-1/

ДОДАТОК А ЗВІТ ПОДІБНОСТІ

UNICHECK

Ім'я користувача:
Лісовиченко Олег Іванович

ID перевірки:
1015375227

Дата перевірки:
01.06.2023 18:54:46 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
01.06.2023 19:08:38 EEST

ID користувача:
76913

Назва документа: **IT-92_Гриник_ПЗ**

Кількість сторінок: **81** Кількість слів: **10385** Кількість символів: **81405** Розмір файлу: **3.57 MB** ID файлу: **1015041071**

11.9%
Схожість

Найбільша схожість: **4.64%** з джерелом з Бібліотеки (ID файлу: **1015013680**)

2.88% Джерела з Інтернету	121	Сторінка 83
11.5% Джерела з Бібліотеки	180	Сторінка 84

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0.76%
Вилучень

Деякі джерела вилучено автоматично (фільтри вилучення: кількість знайдених слів є меншою за **8 слів** та **0%**)

0% Вилучення з Інтернету	25	Сторінка 85
0.76% Вилученого тексту з Бібліотеки	37	Сторінка 85

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2023 р.

ВЕБ-ЗАСТОСУНОК З НАДАННЯ ТА ОТРИМАННЯ ПОСЛУГ ДЛЯ ВЛАСНИКІВ ДОМАШНІХ ТВАРИН

Текст програми

КПІ.ІТ-9208.045440.03.12

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Юлія КРАМАР

Нормоконтроль:
_____ Ірина ВІТКОВСЬКА

Виконавець:
_____ Валентина ГРИНИК

Київ – 2023

Файл Program.cs

```
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.EntityFrameworkCore;
using Microsoft.OpenApi.Models;
using PawsitivelyCare.BLL.Common.Auth;
using PawsitivelyCare.BLL.Services.Interfaces;
using PawsitivelyCare.BLL.Services.Realizations;
using PawsitivelyCare.DAL.Contexts;
using PawsitivelyCare.DAL.Entities;
using PawsitivelyCare.DAL.Repositories.Interfaces;
using PawsitivelyCare.DAL.Repositories.Realizations;
using PawsitivelyCare.Mappings;
using static CSharpFunctionalExtensions.Result;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddDbContextFactory<PawsitivelyCareDbContext>(
        options => options.UseSqlServer(builder.Configuration.GetConnectionString("PawsitivelyCareDBConnection")));

#region "Services"

builder.Services.AddScoped<IUserService, UserService>();
builder.Services.AddScoped<IPostService, PostService>();
builder.Services.AddScoped<ICommentService, CommentService>();
builder.Services.AddScoped<IChatService, ChatService>();

#endregion Region


#region "Repositories"

builder.Services.AddScoped<IBaseRepository<User, Guid>, BaseRepository<User, Guid>>();
builder.Services.AddScoped<IBaseRepository<Post, Guid>, BaseRepository<Post, Guid>>();
builder.Services.AddScoped<IBaseRepository<Comment, Guid>, BaseRepository<Comment, Guid>>();
builder.Services.AddScoped<IBaseRepository<Chat, Guid>, BaseRepository<Chat, Guid>>();
builder.Services.AddScoped<IBaseRepository<ChatMessage, Guid>, BaseRepository<ChatMessage, Guid>>();
builder.Services.AddScoped<IBaseRepository<Image, Guid>, BaseRepository<Image, Guid>>();
builder.Services.AddScoped<IChatRepository, ChatRepository>();


#endregion Region


#region "MapperProfiles"

builder.Services.AddAutoMapper(AppDomain.CurrentDomain.GetAssemblies());

builder.Services.AddAutoMapper(typeof(UserProfile));
builder.Services.AddAutoMapper(typeof(PostProfile));
builder.Services.AddAutoMapper(typeof(CommentProfile));
builder.Services.AddAutoMapper(typeof(ChatProfile));
builder.Services.AddAutoMapper(typeof(ChatMessageProfile));

#endregion Region


#region "Authentication"

builder.Services.Configure<AuthOptions>(builder.Configuration.GetSection("Auth"));
var authOptions = builder.Configuration.GetSection("Auth").Get<AuthOptions>();

builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
   .AddJwtBearer(options =>
   {
      options.RequireHttpsMetadata = false;
      options.TokenValidationParameters = new Microsoft.IdentityModel.Tokens.TokenValidationParameters
      {
         ValidateIssuer = true,
         ValidIssuer = authOptions.Issuer,

         ValidateAudience = true,
         ValidAudience = authOptions.Audience,

         ValidateLifetime = true,

         IssuerSigningKey = authOptions.GetSymmetricSecurityKey(), //HS256
         ValidateIssuerSigningKey = true,
      };
```

```
    });

#endregion Region

#region "Swagger"

builder.Services.AddSwaggerGen(c=>
{
    c.SwaggerDoc("v1", new OpenApiInfo
    {
        Title = "PawsitivalyCare_Api",
        Version = "v1"
    });

    c.AddSecurityDefinition("Bearer", new OpenApiSecurityScheme
    {
        Name = "Authorization",
        Type = SecuritySchemeType.ApiKey,
        Scheme = "Bearer",
        BearerFormat = "JWT",
        In = ParameterLocation.Header,
        Description = "Enter JWT Token with Bearer format"
    });

    c.AddSecurityRequirement(new OpenApiSecurityRequirement
    {
        {
        new OpenApiSecurityScheme
        {
            Reference = new OpenApiReference
            {
                Type = ReferenceType.SecurityScheme,
                Id = "Bearer"
            }
        },
        new string[] { }
        }
    });
});

#endregion Region


#region "Cors"

builder.Services.AddAuthorization();

builder.Services.AddCors(options =>
{
    options.AddPolicy("CorsPolicy", builder =>
    {
        builder.AllowAnyOrigin()
               .AllowAnyMethod()
               .AllowAnyHeader()
               .WithMethods("PUT");
    });
});

#endregion Region


builder.Services.AddControllers();
builder.Services.AddEndpointsApiExplorer();

var app = builder.Build();

if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Error");
    app.UseHsts();
}

app.UseSwaggerUI();
app.UseSwagger();

app.UseCors("CorsPolicy");

app.UseHttpsRedirection();
```

```
app.MapControllers();

app.UseAuthentication();
app.UseAuthorization();

app.Run();
```

Файл ChatService.cs

```
using AutoMapper;
using PawsitivelyCare.BLL.Models;
using PawsitivelyCare.BLL.Services.Interfaces;
using PawsitivelyCare.DAL.Entities;
using PawsitivelyCare.DAL.Repositories.Interfaces;

namespace PawsitivelyCare.BLL.Services.Realizations
{
   public class ChatService : IChatService
   {
      protected readonly IMapper _mapper;
      private readonly IChatRepository _chatRepository;
      private readonly IBaseRepository<ChatMessage, Guid> _chatMessageRepository;
      public ChatService(IChatRepository chatRepository, IBaseRepository<ChatMessage, Guid> chatMessageRepository, IMapper mapper)
      {
         _mapper = mapper;
         _chatRepository = chatRepository;
         _chatMessageRepository = chatMessageRepository;
      }

      public async Task<List<ChatModel>> GetChats(Guid currentUserId)
      {
         var chats = await _chatRepository.GetChats(currentUserId);

         return _mapper.Map<List<ChatModel>>(chats);
      }

      public async Task<ChatModel> GetChat(Guid id)
      {
         var User = await _chatRepository.GetAsync(id);

         return _mapper.Map<ChatModel>(User);
      }


      public async Task<ChatModel> CreateChat(ChatModel chatModel, Guid currentUserId)
      {
         var chatEntity = _mapper.Map<Chat>(chatModel);
         var chat = await _chatRepository.AddChat(chatEntity, chatModel.PostCreatorId, currentUserId);

         return _mapper.Map<ChatModel>(chat);
      }

      public async Task<ChatMessageModel> CreateMessage(ChatMessageModel chatMessage)
      {
         var messageEntity = _mapper.Map<ChatMessage>(chatMessage);
         messageEntity.CreatedAt = DateTime.Now;

         var message = await _chatMessageRepository.AddAsync(messageEntity);

         return _mapper.Map<ChatMessageModel>(message);
      }

      public async Task<List<ChatMessageModel>> GetMessages(Guid chatId)
      {
         var messages = await _chatMessageRepository.Query(m => m.ChatId == chatId);
         return _mapper.Map<List<ChatMessageModel>>(messages.OrderBy(m => m.CreatedAt));
      }
   }
}
```

Файл CommentService.cs

```
using AutoMapper;
using PawsitivelyCare.BLL.Models;
using PawsitivelyCare.BLL.Services.Interfaces;
using PawsitivelyCare.DAL.Entities;
using PawsitivelyCare.DAL.Repositories.Interfaces;

namespace PawsitivelyCare.BLL.Services.Realizations
```

```
{
    public class CommentService : ICommentService
    {
        private readonly IBaseRepository<Comment, Guid> _commentRepository;
        private readonly IBaseRepository<Post, Guid> _postRepository;
        private readonly IBaseRepository<User, Guid> _userRepository;
        protected readonly IMapper _mapper;

        public CommentService(IBaseRepository<Comment, Guid> commentRepository, IBaseRepository<Post, Guid> postRepository,
            IBaseRepository<User, Guid> userRepository, IMapper mapper)
        {
            _commentRepository = commentRepository;
            _postRepository = postRepository;
            _userRepository = userRepository;
            _mapper = mapper;
        }

        public async Task<CommentModel> CreateComment(CommentModel commentModel)
        {
            var commentEntity = _mapper.Map<Comment>(commentModel);
            commentEntity.CreatedAt = DateTime.Now;
            var createdComment = await _commentRepository.AddAsync(commentEntity);

            return _mapper.Map<CommentModel>(createdComment);
        }

        public async Task<List<CommentUserModel>> GetComments(Guid postId)
        {
            var comments = await _commentRepository.Query(p => p.PostId == postId);
            var commentsModels = _mapper.Map<List<CommentUserModel>>(comments);

            foreach ( var comment in commentsModels)
            {
                var creator = await _userRepository.QueryFirst(p => p.Id == comment.Comment.SenderId);
                comment.Creator = _mapper.Map<UserModel>(creator);
            }

            return commentsModels;
        }

        public async Task DeleteComment(Guid id)
        {
            var commentEntity = await _commentRepository.GetAsync(id);

            if (commentEntity == null)
                throw new ArgumentException($"Chosen comment could not be found.");

            await _commentRepository.DeleteAsync(commentEntity);
        }
    }
}
```

Файл PostService.cs

```
using AutoMapper;
using PawsitivelyCare.BLL.Models;
using PawsitivelyCare.BLL.Services.Interfaces;
using PawsitivelyCare.DAL.Entities;
using PawsitivelyCare.DAL.Repositories.Interfaces;
using static PawsitivelyCare.DAL.Entities.Post;

namespace PawsitivelyCare.BLL.Services.Realizations
{
    public class PostService : IPostService
    {
        private readonly IBaseRepository<Post, Guid> _postRepository;
        private readonly IBaseRepository<Image, Guid> _imageRepository;
        protected readonly IMapper _mapper;

        public PostService(IBaseRepository<Post, Guid> postRepository, IBaseRepository<Image, Guid> imageRepository, IMapper mapper)
        {
            _postRepository = postRepository;
            _imageRepository = imageRepository;
            _mapper = mapper;
        }

        public async Task<PostModel> CreatePost(PostModel postModel)
        {
            var postEntity = _mapper.Map<Post>(postModel);
```

```
            postEntity.CreatedAt = DateTime.Now;
            var createdPost = await _postRepository.AddAsync(postEntity);

            return _mapper.Map<PostModel>(createdPost);
        }

        public async Task<PostModel> GetPost(Guid id)
        {
            var User = await _postRepository.GetAsync(id);

            return _mapper.Map<PostModel>(User);
        }

        public async Task<List<PostModel>> GetUserPosts(Guid userId)
        {
            return _mapper.Map<List<PostModel>>(await _postRepository.Query(
                p => p.CreatorId == userId,
                orderBy: p=>p.OrderByDescending(d=>d.CreatedAt)));
        }

        public async Task<List<PostModel>> GetPosts(PostType type, int? category, string? location, Guid userId)
        {
            return _mapper.Map<List<PostModel>>(await _postRepository.Query(
                p => p.Type == type &&
                (category == 0 || (p.PostCategoryId == category)) &&
                (location == null || (p.Location == location)) &&
                (p.CreatorId != userId),
                orderBy: p => p.OrderByDescending(d => d.CreatedAt)));
        }

        public async Task<List<string>> GetPostImages(Guid postId)
        {
            var postImages = _mapper.Map<List<ImageModel>>(await _imageRepository.Query(i => i.PostId == postId));

            var imagePaths = new List<string>();
            foreach (var image in postImages)
            {
                imagePaths.Add(image.FileName);
            }

            return imagePaths;
        }

        public async Task UploadImages(List<string> images, Guid postId)
        {
            foreach (var image in images)
            {
                var imageEntity = new Image
                {
                    FileName = image,
                    PostId = postId
                };

                await _imageRepository.AddAsync(imageEntity);
            }
        }

        public async Task UpdatePost(PostModel postModel)
        {
            var postEntity = _mapper.Map<Post>(postModel);

            await _postRepository.UpdateAsync(postEntity);
        }

        public async Task DeletePost(Guid id)
        {
            var postEntity = await _postRepository.GetAsync(id);

            if (postEntity == null)
                throw new ArgumentException($"User with the id '{id}' could not be found.");

            await _postRepository.DeleteAsync(postEntity);
        }
    }
}
```

Файл UserService.cs

```
using BCrypt.Net;
```

```
using AutoMapper;
using Microsoft.EntityFrameworkCore;
using PawsitivelyCare.BLL.Models;
using PawsitivelyCare.BLL.Services.Interfaces;
using PawsitivelyCare.DAL.Entities;
using PawsitivelyCare.DAL.Repositories.Interfaces;
using System.Security.Claims;
using Microsoft.IdentityModel.Tokens;
using Microsoft.Extensions.Configuration;
using System.Text;
using System.IdentityModel.Tokens.Jwt;
using Microsoft.Extensions.Options;
using PawsitivelyCare.BLL.Common.Auth;

namespace PawsitivelyCare.BLL.Services.Realizations
{
   public class UserService : IUserService
   {
      private readonly IBaseRepository<User, Guid> _userRepository;
      protected readonly IMapper _mapper;
      private readonly IConfiguration _configuration;
      private readonly IOptions<AuthOptions> _authOptions;

      public UserService(IBaseRepository<User, Guid> userRepository, IMapper mapper, IConfiguration configuration, IOptions<AuthOptions>
authOptions)
      {
         _userRepository = userRepository;
         _mapper = mapper;
         _configuration = configuration;
         _authOptions = authOptions;
      }

      public async Task<UserModel?> AuthenticateUser(UserModel userModel)
      {
         var userEntity = await _userRepository.QueryFirst(u => u.Email == userModel.Email);

         if(userEntity == null || !(await PasswordVerify(userModel.Password, userEntity.PasswordHash)))
         {
            return null;
         }

         return _mapper.Map<UserModel>(userEntity);
      }

      public async Task<UserModel> Register(UserModel model)
      {
         if (await _userRepository.QueryFirst(u=>u.Email == model.Email) != null)
            throw new ArgumentException("Username with email " + model.Email + " already exist");

         var userEntity = _mapper.Map<User>(model);

         userEntity.PasswordHash = await HashPasswordAsync(model.Password);
         userEntity.CreatedAt= DateTime.Now;

         var createdEntity = await _userRepository.AddAsync(userEntity);

         return _mapper.Map<UserModel>(createdEntity);
      }

      public async Task<UserModel> Get(Guid id)
      {
         var user = await _userRepository.GetAsync(id);

         return _mapper.Map<UserModel>(user);
      }

      public async Task<UserModel> GetByEmail(string email)
      {
         return _mapper.Map<UserModel>(await _userRepository.QueryFirst(u => u.Email == email));
      }

      public async Task Update(UserModel userModel)
         {
         var userEntityFromDb = await _userRepository.QueryFirst(u => u.Email == userModel.Email);

         if (userEntityFromDb != null && userEntityFromDb.Id != userModel.Id)
            throw new ArgumentException("Email '" + userModel.Email + "'  already exist");

         var userEntity = _mapper.Map<User>(userModel);
```

```
            if (!string.IsNullOrEmpty(userModel.Password))
                userEntity.PasswordHash = await HashPasswordAsync(userModel.Password);
            else
                userEntity.PasswordHash = userEntityFromDb.PasswordHash;

            await _userRepository.UpdateAsync(userEntity);
        }

        public async Task Delete(Guid id)
        {
            var userEntity = await _userRepository.GetAsync(id);

            if (userEntity == null)
                throw new ArgumentException($"User with the id '{id}' could not be found.");

            await _userRepository.DeleteAsync(userEntity);
        }

        public string GenerateJwtToken(UserModel userModel)
        {
            var authParams = _authOptions.Value;

            var securityKey = authParams.GetSymmetricSecurityKey();
            var credentials =  new SigningCredentials(securityKey, SecurityAlgorithms.HmacSha256);

            var claims = new List<Claim>()
            {
                new Claim(JwtRegisteredClaimNames.Email, userModel.Email),
                new Claim(JwtRegisteredClaimNames.Sub, userModel.Id.ToString())
            };

            var token = new JwtSecurityToken(
                authParams.Issuer,
                authParams.Audience,
                claims,
                expires: DateTime.Now.AddDays(authParams.TokenLifeTime),
                signingCredentials: credentials);

            return new JwtSecurityTokenHandler().WriteToken(token);
        }

        private async Task<string> HashPasswordAsync(string password)
        {
            return await Task.Run(() => BCrypt.Net.BCrypt.HashPassword(password));
        }

        private async Task<bool> PasswordVerify(string password, string passwordHash)
        {
            return await Task.Run(() => BCrypt.Net.BCrypt.Verify(password, passwordHash));
        }
    }
}
```

Файл BaseRepository.cs

```
using Microsoft.EntityFrameworkCore;
using PawsitivelyCare.DAL.Contexts;
using PawsitivelyCare.DAL.Repositories.Interfaces;
using System.Linq.Expressions;

namespace PawsitivelyCare.DAL.Repositories.Realizations
{
    public class BaseRepository<TEntity, TKey> : IBaseRepository<TEntity, TKey> where TEntity : class //abstract??
    {
        protected readonly IDbContextFactory<PawsitivelyCareDbContext> _contextFactory;

        public BaseRepository(IDbContextFactory<PawsitivelyCareDbContext> contextFactory)
        {
            _contextFactory = contextFactory;
        }

        public async Task<TEntity> AddAsync(TEntity entity)
        {
            using var context = _contextFactory.CreateDbContext();
            GetDbSet(context).Add(entity);
            await context.SaveChangesAsync();

            return entity;
```

```
    }

    public async Task<TEntity?> GetAsync(TKey id)
    {
        using var context = _contextFactory.CreateDbContext();
        return await GetDbSet(context).FindAsync(id);
    }

    public async Task UpdateAsync(TEntity entity)
    {
        using var context = _contextFactory.CreateDbContext();
        context.Entry(entity).State = EntityState.Modified;
        await context.SaveChangesAsync();
    }

    public async Task DeleteAsync(TEntity entity)
    {
        using var context = _contextFactory.CreateDbContext();
        GetDbSet(context).Remove(entity);
        await context.SaveChangesAsync();
    }

    public async Task<IEnumerable<TEntity>> AddRangeAsync(IEnumerable<TEntity> entities)
    {
        using var context = _contextFactory.CreateDbContext();

        var entitiesList = entities.ToList();
        GetDbSet(context).AddRange(entitiesList);

        await context.SaveChangesAsync();

        return entitiesList;
    }

    public async Task DeleteRangeAsync(IEnumerable<TEntity> entities)
    {
        using var context = _contextFactory.CreateDbContext();
        GetDbSet(context).RemoveRange(entities);
        await context.SaveChangesAsync();
    }

    public async Task<TEntity?> QueryFirst(Expression<Func<TEntity, bool>>? filter = null, Func<IQueryable<TEntity>,
IOrderedQueryable<TEntity>>? orderBy = null, params Expression<Func<TEntity, object>>[] includes)
    {
        var entities = await Query(filter, orderBy, 1, includes);
        return entities.FirstOrDefault();
    }

    public async Task<List<TEntity>> Query(Expression<Func<TEntity, bool>>? filter = null, Func<IQueryable<TEntity>,
IOrderedQueryable<TEntity>>? orderBy = null, int count = 0, params Expression<Func<TEntity, object>>[] includes)
    {
        using var context = _contextFactory.CreateDbContext();
        IQueryable<TEntity> query = GetDbSet(context).AsNoTracking();

        foreach (Expression<Func<TEntity, object>> include in includes)
            query = query.Include(include);

        if (filter != null)
            query = query.Where(filter);

        if (count > 0)
            query = query.Take(count);

        if (orderBy is { })
            query = orderBy(query);

        return await query.ToListAsync();
    }

    public async Task<int> QueryCount(Expression<Func<TEntity, bool>> filter)
    {
        using var context = _contextFactory.CreateDbContext();
        return await GetDbSet(context).AsNoTracking().Where(filter).CountAsync();
    }

    private DbSet<TEntity> GetDbSet(DbContext dbContext) => dbContext.Set<TEntity>();
  }
}
```

Файл PawsitivelyCareDbContext.cs

```
using Microsoft.EntityFrameworkCore;
using PawsitivelyCare.DAL.Entities;
using PawsitivelyCare.DAL.EntityConfiguration;

namespace PawsitivelyCare.DAL.Contexts
{
   public class PawsitivelyCareDbContext : DbContext
   {
      public PawsitivelyCareDbContext(DbContextOptions<PawsitivelyCareDbContext> options) : base(options)
      {
      }

      public DbSet<User> Users { get; set; }
      public DbSet<Chat> Chats { get; set; }
      public DbSet<ChatMessage> ChatMessages { get; set; }
      public DbSet<Post> Posts { get; set; }
      public DbSet<PostCategory> PostCategories { get; set; }
      public DbSet<Comment> Comments { get; set; }
      public DbSet<Image> Images { get; set; }

      protected override void OnModelCreating(ModelBuilder modelBuilder)
      {
         modelBuilder.ApplyConfiguration(new ChatConfiguration());
         modelBuilder.ApplyConfiguration(new ChatMessageConfiguration());
         modelBuilder.ApplyConfiguration(new PostConfiguration());
         modelBuilder.ApplyConfiguration(new UserConfiguration());
         modelBuilder.ApplyConfiguration(new PostCategoryConfiguration());
         modelBuilder.ApplyConfiguration(new CommentConfiguration());
         modelBuilder.ApplyConfiguration(new ImageConfiguration());

         modelBuilder.Entity<User>()
            .HasMany(f => f.Chats)
            .WithMany(g => g.Users)
            .UsingEntity<Dictionary<string, object>>(
               "ChatUser",
                  j => j.HasOne<Chat>().WithMany().OnDelete(DeleteBehavior.Cascade),
                  j => j.HasOne<User>().WithMany().OnDelete(DeleteBehavior.NoAction));

         modelBuilder.Entity<User>()
            .HasMany(u => u.Chats)
            .WithMany(r => r.Users);

         base.OnModelCreating(modelBuilder);
      }
   }
}
```

Файл home.page.ts

```
import{ Component, OnInit }from '@angular/core';
import{ Post }from 'src/app/models/post.model';
import{ AuthenticationService }from 'src/app/services/auth.service';
import{ PostsService }from 'src/app/services/posts.service';
import{ PostType }from 'src/app/models/post.model';
import{ PostCategory }from 'src/app/models/post.model';
import{ ChatsService }from 'src/app/services/chats.service';
import{ Chat }from 'src/app/models/chat.model';
import{ Router }from '@angular/router';
import{ AppComponent }from 'src/app/app.component';
import{ CommentsService }from 'src/app/services/comments.service';
import{ Comment }from 'src/app/models/comment.model';
import{ CommentUser }from 'src/app/models/comment-user.model';
import{ ModalController, ToastController }from '@ionic/angular';
import{ CreateCommentComponent }from '../comment/create-comment/create-comment.component';

@Component({
selector: 'app-home',
  templateUrl: 'home.page.html',
  styleUrls: ['home.page.scss'],
})
export class HomePage implements OnInit
{
   posts: Post [] = [];
   selectedType: PostType = PostType.Customer;
   selectedCategory: number = 0;
   selectedLocation: string = '';
   postTypes: typeof PostType = PostType;
```

```
    postCategories: typeof PostCategory = PostCategory;
    imagesLoaded: boolean = false;

    constructor(
    private appComp: AppComponent,
    private postService: PostsService,
    private chatService: ChatsService,
    private commentsService: CommentsService,
    private modalController: ModalController,
    private toastController: ToastController,
    private router: Router,
  ) { }

ngOnInit() {
    this.appComp.changeNavbarTitle("Posts");
    this.getPosts();
    this.imagesLoaded = true;
}

ionViewDidEnter() {
    this.appComp.changeNavbarTitle("Posts");
}


async getPosts()
{
    this.posts = await this.postService.getPosts(this.selectedType, this.selectedCategory, this.selectedLocation);
}

async showComments(post: Post) {
    const currentPostIndex = this.posts.findIndex((p) => p.id === post.id);

    if (currentPostIndex !== -1)
    {
        const currentPost = this.posts[currentPostIndex];
        this.posts[currentPostIndex].showComments = !this.posts[currentPostIndex].showComments;

        if (currentPost.showComments)
        {
            this.commentsService
              .getComments(currentPost.id)
              .subscribe(
                (data: CommentUser[]) => {
                 this.posts[currentPostIndex].comments = data;
            },
            (error) => {
                console.log(error);
            }
            );
        }
    }
}

async createChat(post: Post) {
    const chat: Chat = {
    id: '',
      name: post.title,
      postId: post.id,
      postCreatorId: post.creatorId,
    }

    this.chatService.createChat(chat).subscribe(
      res => {
      const id: string = res.id;
      this.router.navigateByUrl(`/ chat /${ id}`);
},
      error => {
         alert('Something went wrong. Please try again later.');
      }
    );
  }

  async openCommentModal(postId: string) {
    const modal = await this.modalController.create({
    component: CreateCommentComponent
    });
    await modal.present();
    const { data } = await modal.onDidDismiss();
```

```
    if (data)
    {
       const comment: Comment = {
       text: data.comment,
       postId: postId,
      };

       this.commentsService.createComment(comment).subscribe(
        (res) => {
           this.presentToast('Comment created successfully');
        },
        (error) => {
           alert('Something went wrong. Please try again later.');
        }
       );
    }
}

async presentToast(message: string) {
    const toast = await this.toastController.create({
    message: message,
      duration: 3000,
      position: 'top',
    });
    toast.present();
}
}
```

Файл user-posts.component.ts

```
import{ Component, OnInit }from '@angular/core';
import{ Router }from '@angular/router';
import{ AlertController, ToastController }from '@ionic/angular';
import{ AppComponent }from 'src/app/app.component';
import{ Post, PostCategory, PostType }from 'src/app/models/post.model';
import{ CommentsService }from 'src/app/services/comments.service';
import{ PostsService }from 'src/app/services/posts.service';

@Component({
selector: 'app-user-posts',
  templateUrl: './user-posts.component.html',
  styleUrls: ['./user-posts.component.scss'],
})
export class UserPostsComponent implements OnInit
{
    posts: Post [] = [];
    newPost: Post;
    postCategories: typeof PostCategory = PostCategory;
    postTypes: typeof PostType = PostType;

    constructor(
    private appComp: AppComponent,
    private postService: PostsService,
    private commentService: CommentsService,
    private alertCtrl: AlertController,
    private router: Router,
    private toastController: ToastController,
  ) { }

async ngOnInit()
{
    this.appComp.changeNavbarTitle("My Posts")
    await this.getUserPosts();
}

async getUserPosts()
{
    this.posts = await this.postService.getUserPosts();
}

async editPost(post: any) {
    const alert = await this.alertCtrl.create({
    header: 'Edit Post',
      inputs:
        [
        {
        name: 'title',
          type: 'text',
          value: post.title,
```

```
          placeholder: 'Title',
        },
        {
        name: 'location',
          type: 'text',
          value: post.location,
          placeholder: 'Location',
        },
        {
        name: 'content',
          type: 'text',
          value: post.content,
          placeholder: 'Description',
        },
      ],
      buttons:
        [
        {
        text: 'Cancel',
          role: 'cancel',
        },
        {
        text: 'Save',
          handler: (data) => {
            post.title = data.title;
            post.location = data.location;
            post.content = data.content;

            this.sendPost(post)
          },
        },
      ],
    });
    await alert.present();
}

async createPost()
{
    this.router.navigate(['/home/createPost']);
  }

  async showComments(post: Post) {
    const foundPost = this.posts.find((p) => p.id === post.id);

    if (foundPost)
    {
        foundPost.showComments = !foundPost.showComments;

        if (foundPost.showComments)
        {
            this.commentService.getComments(foundPost.id);
        }
    }
}

async sendPost(post: any) {
    this.postService.editPost(post).subscribe(
    (response) => {
        console.log('Post edited:', response);
    },
      (error) => {
        console.log(error);
      }
    );
}

async deletePost(postId: string) {
    this.postService.deletePost(postId).subscribe(
      (response) => {
        this.presentToast('Post deleted successfully');
        this.getUserPosts();
      },
      (error) => {
        alert('Something went wrong. Please try again later.');
      }
    );

    this.getUserPosts();
}
```

```
async presentToast(message: string) {
   const toast = await this.toastController.create({
   message: message,
     duration: 3000,
     position: 'top',
   });
   toast.present();
}
}
```

Файл post.service.ts

```
import{ HttpClient, HttpParams }from '@angular/common/http';
import{ Inject, Injectable }from '@angular/core';
import{ Post, PostCategory, PostType }from '../models/post.model';
import{ BACKEND_API_URL }from '../app-injection-tokens';
import{ Observable }from 'rxjs';

@Injectable({
providedIn: 'root',
})
export class PostsService
{
   constructor(
   private http: HttpClient,
   @Inject(BACKEND_API_URL) private apiUrl: string
 ) {}

  async getPosts(
   type: PostType,
   category: number,
   location: string
 ): Promise<Post[]> {
   let params = new HttpParams()
     .set('type', type.toString())
     .set('category', category)
     .set('location', location);

   const posts = await this.http
     .get<Post[]>(`${this.apiUrl}
api / posts /`, { params })
     .toPromise();

if (posts === undefined)
{
   return [];
}

const finalPosts = await this.test(posts)

   return finalPosts;
  }

  async getUserPosts()
{
   const posts = await this.http.get<Post[]>(`${ this.apiUrl}
   api / posts / myPosts`, { }).toPromise();

if (posts === undefined)
{
   return [];
}

const finalPosts = await this.test(posts)

   return finalPosts;
  }

  getPost(postId: string): Observable<Post> {
   return this.http.get<Post>(`${ this.apiUrl}
   api / posts /${ postId}`);
}

editPost(post: Post): Observable<Post> {
   return this.http.put<Post>(`${ this.apiUrl}
   api / posts /` +post.id, post);
}
```

```
createPost(post: Post): Observable<Post> {
   return this.http.post<Post>(`${ this.apiUrl}
   api / posts`, post);
}

deletePost(postId: string): Observable<boolean> {
   return this.http.delete<boolean>(`${ this.apiUrl}
   api / posts /${ postId}`);
}

async getPostImages(postId: string): Promise<string[]> {
   const images = await this.http
   .get<string[]>(`${ this.apiUrl}
   api / posts /${ postId}/ images`)
     .toPromise();

   if (images === undefined)
   {
      return [];
   }
   return images;
}

uploadImages(images: string[], postId: string): Observable<boolean> {
   return this.http.post<boolean>(
     `${ this.apiUrl}
   api / posts /${ postId}/ images`,
     images
   );
}

private async test(posts: Post[]) {
   posts.forEach(async(post: Post) => {
      post.images = await this.getPostImages(post.id);

      if (post.images)
      {
         for (let i = 0; i < post.images.length; i++)
         {
            const image = post.images[i];
            const photoBlob = await this.base64toBlob(image);
            const photoFile = new File([photoBlob], 'img.jpeg', {
            type: 'image/jpeg',
        });
         post.images[i] = URL.createObjectURL(photoFile);
      }
   }
});

return posts;
 }

 private async base64toBlob(base64: string) {
   const byteString = window.atob(base64);
   const arrayBuffer = new ArrayBuffer(byteString.length);
   const int8Array = new Uint8Array(arrayBuffer);
   for (let i = 0; i < byteString.length; i++)
   {
      int8Array[i] = byteString.charCodeAt(i);
   }

   const blob = new Blob([int8Array], { type: 'image/png' });
   return blob;
}
}
```

Файл login.component.ts

```
import{ Component, ViewChild }from '@angular/core';
import{ NgForm }from '@angular/forms';
import{ Router }from '@angular/router';
import{ AuthenticationService }from 'src/app/services/auth.service';

@Component({
selector: 'app-home',
 templateUrl: 'login.component.html',
 styleUrls: ['login.component.scss'],
})
export class LoginComponent
```

```
{
   constructor(private authService: AuthenticationService, private router: Router) { }

 @ViewChild('loginForm') loginForm: NgForm;

 email: string;
 password: string;

 ngOnInit() : void {
  this.authService.setIsLoginPage(true);
 }

public get isLogginIn(): boolean
{
   return this.authService.isAuthenticated();
}

login(email: string, password: string) {
   this.authService.login(email, password).subscribe(res => {
      this.authService.setIsLoginPage(false);
      this.loginForm.reset();
      this.router.navigateByUrl('home/posts');
   }, error => {
      alert("Wrong login or password")
   })
 }

logout() {
   this.authService.logout;
}

navigateToRegister() {
   this.router.navigate(['/auth/register']);
}
}
```

Файл register.component.ts

```
import{ Component, OnInit }from '@angular/core';
import{ FormBuilder, FormGroup, Validators }from '@angular/forms';
import{ Route, Router }from '@angular/router';
import{ ToastController }from '@ionic/angular';
import{ Gender, User }from 'src/app/models/user.model';
import{ AuthenticationService }from 'src/app/services/auth.service';

@Component({
selector: 'app-register',
 templateUrl: './register.component.html',
 styleUrls: ['./register.component.scss'],
})
export class RegisterComponent implements OnInit
{
   registrationForm: FormGroup;
   gender: typeof Gender = Gender;

   constructor(
   private formBuilder: FormBuilder,
   private authService: AuthenticationService,
   private toastController: ToastController,
   private router: Router
 ) { }

ngOnInit() {
   this.registrationForm = this.formBuilder.group({
   Name: ['', Validators.required],
     Surname: ['', Validators.required],
     Phone: ['', Validators.required],
     Email: ['', [Validators.required, Validators.email]],
     Password: ['', Validators.required],
     Gender: ['', Validators.required],
   });
}

onSubmit() {
   const user: User = this.registrationForm.value;
   this.authService.register(user).subscribe(
     res => {
     this.presentToast('Registration successful');
     this.router.navigate(['/auth/login']);
```

```
},
    error => {
      alert('Something went wrong. Please try again later. May be a user with this email already exists.');
    }
  );
 }

 async presentToast(message: string) {
  const toast = await this.toastController.create({
  message: message,
   duration: 3000,
   position: 'top',
  });
  toast.present();
}
}
```

Файл app.component.ts

```
import{ Component }from '@angular/core';
import{ AuthenticationService, TOKEN }from './services/auth.service';
import{ Route, Router }from '@angular/router';

@Component({
selector: 'app-root',
 templateUrl: 'app.component.html',
})
export class AppComponent
{
   public isLoginPage: boolean = false;
 navbarTitle: string = 'PawsitivelyCare';

 constructor(private authService: AuthenticationService, private router: Router) {
  this.authService.isLoginPage$.subscribe(isLoginPage => {
     this.isLoginPage = isLoginPage;
  });
 }

logout() {
   this.authService.logout()
 }

changeNavbarTitle(title: string) {
   this.navbarTitle = title;
}
}
```

Файл home.page.html

```
< ion - toolbar style = "width: 100%;" >
 < div class= "toolbar-container" >
  < ion - item >
   < ion - label class= "lable" style = "font-size: medium" > Location:</ ion - label >
   < ion - input[(ngModel)] = "selectedLocation" ></ ion - input >
  </ ion - item >
  < ion - select class= "lable"style = "width: 40%;"[(ngModel)] = "selectedType" placeholder = "Type" >
   < ion - select - option[value] = "postTypes.Customer" > Customers </ ion - select - option >
   < ion - select - option[value] = "postTypes.Executor" > Executors </ ion - select - option >
  </ ion - select >
  < ion - select class= "lable" style = "width: 40%;"[(ngModel)] = "selectedCategory" placeholder = "Category" >
   < ion - select - option[value] = "0" > All </ ion - select - option >
   < ion - select - option[value] = "postCategories.Medicine" > Medicine </ ion - select - option >
   < ion - select - option[value] = "postCategories.Walking" > Walking </ ion - select - option >
   < ion - select - option[value] = "postCategories.Grooming" > Grooming </ ion - select - option >
   < ion - select - option[value] = "postCategories.Training" > Training </ ion - select - option >
   < ion - select - option[value] = "postCategories.PetSitting" > PetSitting </ ion - select - option >
   < ion - select - option[value] = "postCategories.VeterinaryCare" > VeterinaryCare </ ion - select - option >
   < ion - select - option[value] = "postCategories.Other" > Other </ ion - select - option >
  </ ion - select >
 </ div >
</ ion - toolbar >
< ion - toolbar >
 < ion - button(click) = "getPosts()" >
  < ion - icon slot = "icon-only" name = "search" ></ ion - icon >
 </ ion - button >
</ ion - toolbar >

< p * ngIf = "posts.length == 0" class= "no-content-text" >
 Posts not found
```

```
</p>

<ion-content* ngIf = "posts.length != 0" >
 < ion - list >
  < ion - item * ngFor = "let post of posts" >
   < ion - card style= "width: 100%; display: flex; flex-direction: column;" >
    < ion - card - header class= "item-description" >
     < div class= "items" >
      < div class= "item-name" > Category:</ div >
      < div class= "item-value" >{ { postCategories[post.postCategoryId]} }</ div >
     </ div >
     < div class= "items" >
      < div class= "item-name" > Location:</ div >
      < div class= "item-value" >{ { post.location} }</ div >
     </ div >
    </ ion - card - header >
    < ion - card - header class= "post-title" style = "font-size: 18px; font-weight: bold; color: black" >
     { { post.title} }
    </ ion - card - header >
    < ion - card - content style = "font-size: 16px; color: black" >
     { { post.content} }
    </ ion - card - content >
    < div style = "display: flex; flex-wrap: wrap;" >
     < div * ngFor = "let image of post.images" style = "flex-basis: 33.33%; padding: 5px; flex-grow: 1;" >
      < img[src] = "image" style = "max-width: 100%; max-height: 100%; width: auto; height: auto;" >
     </ div >
    </ div >
    < div style = "display: flex; justify-content: space-between; align-items: center; padding-left: 10px; padding-right: 10px;" >
     < ion - button color = "success"(click) = "createChat(post)" > Respond </ ion - button >
     < ion - label class= "lable"(click) = "openCommentModal(post.id)" > Add Comment </ ion - label >
    </ div >
    < br >
    < ion - label class= "lable"(click) = "showComments(post)" > Comments </ ion - label >
     < ion - list * ngIf = "post.showComments" >
      < br >
      < div style = "margin-bottom: 10px;" * ngFor = "let c of post.comments" >
       < div style = "display: flex; justify-content: space-between; align-items: center; margin-bottom: 5px;" >
        < div style = "display: flex; align-items: center;" >
         < div style = "font-weight: bold; font-size: 18px; color: black; margin-right: 5px;" >{ { c.creator.name } }</ div >
         < div style = "font-weight: bold; font-size: 18px; color: black;" >{ { c.creator.surname } }</ div >
        </ div >
        < div class= "time other-time" >{ { c.comment.createdAt | date:'d MMMM, HH:mm' } }</ div >
       </ div >
       < div style = "font-size: 16px; color: rgb(36, 35, 35)" >{ { c.comment.text } }</ div >
       < br >
      </ div >
     </ ion - list >
   </ ion - card >
  </ ion - item >
 </ ion - list >
 < ion - infinite - scroll >
  < ion - infinite - scroll - content ></ ion - infinite - scroll - content >
 </ ion - infinite - scroll >
 </ ion - content >
```

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2023 р.

ВЕБ-ЗАСТОСУНОК З НАДАННЯ ТА ОТРИМАННЯ ПОСЛУГ ДЛЯ ВЛАСНИКІВ ДОМАШНІХ ТВАРИН

Програма та методика тестування

КПІ.ІТ-9208.045440.04.51

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Юлія КРАМАР

Нормоконтроль:
_____ Ірина ВІТКОВСЬКА

Виконавець:
_____ Валентина ГРИНИК

Київ – 2023

ЗМІСТ

1 ОБ’ЄКТ ВИПРОБУВАНЬ

Об’єктом випробування є веб-застосунок застосунку з надання та отримання послуг для власників домашніх тварин.

2 МЕТА ТЕСТУВАННЯ

Метою тестування є наступне:

– перевірка правильності роботи програмного забезпечення відповідно до функціональних вимог;

– перевірка збереження даних;

– перевірка надання даних;

– перевірка сумісності застосунку з різними операційною системою Android;

– знаходження проблем, помилок і недоліків з метою їх усунення;

– перевірка зручності графічного інтерфейсу.

3 МЕТОДИ ТЕСТУВАННЯ

Для тестування програмного забезпечення використовуються такі методи:

– статичне тестування – перевіряється програма разом з усією документацією, яка аналізується на предмет дотримання стандартів програмування;

– функціональне тестування – полягає у перевірці відповідності реальної поведінки програмного забезпечення очікуваній;

– системне тестування – перевіряється усе програмне забезпечення в цілому;

– мануальне тестування – тестування без використання автоматизації, тест-кейси пише особа, що тестує програмне забезпечення.

4 ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Тестування виконується мануально з метою знаходження помилок та недоліків як у функціональній частині програмного забезпечення так і в зручності користування. Для того, щоб перевірити працездатність та відмовостійкість застосунку, необхідно провести наступні тестування:

– динамічне тестування на відповідність функціональним вимогам;

– тестування на мобільних пристроях з різною версією операційної системи;

– тестування на виведення повідомлень про помилку, коли це необхідно;

– тестування зміни орієнтації екрану;

– тестування інтерфейсу користувача;

– тестування зручності використання.

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2023 р.

ВЕБ-ЗАСТОСУНОК З НАДАННЯ ТА ОТРИМАННЯ ПОСЛУГ ДЛЯ ВЛАСНИКІВ ДОМАШНІХ ТВАРИН

Керівництво користувача

КПІ.ІТ-9208.045440.05.34

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Юлія КРАМАР

Нормоконтроль:
_____ Ірина ВІТКОВСЬКА

Виконавець:
_____ Валентина ГРИНИК

Київ – 2023

ЗМІСТ

1 ПРИЗНАЧЕННЯ ПРОГРАМИ

Веб-застосунок з надання та отримання послуг для власників домашніх тварин – це застосунок, який надає користувачам можливості легко та швидко отримувати та надавати послуги в сфері догляду за домашніми тваринами шляхом створення заявок на отримання чи надання послуг, а також можливості підтримувати зв’язок із замовником чи виконавцем послуг. Основними функціями застосунку є публікація послуги чи заявки на отримання послуги, перегляд послуг і заявок на отримання послуг, фільтрація послуг за критеріями, збір коментарів до послуг, відгук на послугу з подальшим контактом з замовником і подальше підтримання контакту з замовником.

2 ПІДГОТОВКА ДО РОБОТИ З ПРОГРАМНИМ ЗАБЕЗПЕЧЕННЯМ

2.1 Системні вимоги для коректної роботи

Для успішної роботи даного застосунку необхідне виконання наступних вимог:

– наявність мобільного пристрою з операційною системою на базі Android: Android 5.0 (Lollipop) або вище;

– процесор: ARM або Intel з архітектурою ARMv7 або вище;

– наявність доступу до Інтернету;

– об'єм оперативної пам'яті (ОЗП): 1 ГБ та більше;

– для встановлення додатку на мобільному пристрої повинно бути не менше 1 ГБ вільної пам’яті;

2.2 Завантаження застосунку

На даний момент застосунок можна встановити власноруч, використовуючи відповідний apk-файл. Для цього спершу необхідно завантажити його на мобільний пристрій, а потім, використовуючи який-небудь інсталятор, виконати встановлення даного додатку.

2.3 Перевірка коректної роботи

По завершенню встановлення додатка на робочому столі мобільного пристрою повинна відобразитись іконка даного застосунку. У разі, якщо дана іконка не з’явилась, то встановлення відбулось не успішно. Інакше користувач має змогу запустити додаток, клацнувши на його іконку. Після натискання повинна відобразитись початкова сторінка застосунку.

3 ВИКОНАННЯ ПРОГРАМИ

При першому запуску веб-застосунку користувачу буде відображено сторінку входу до системи (рисунок 3.1).

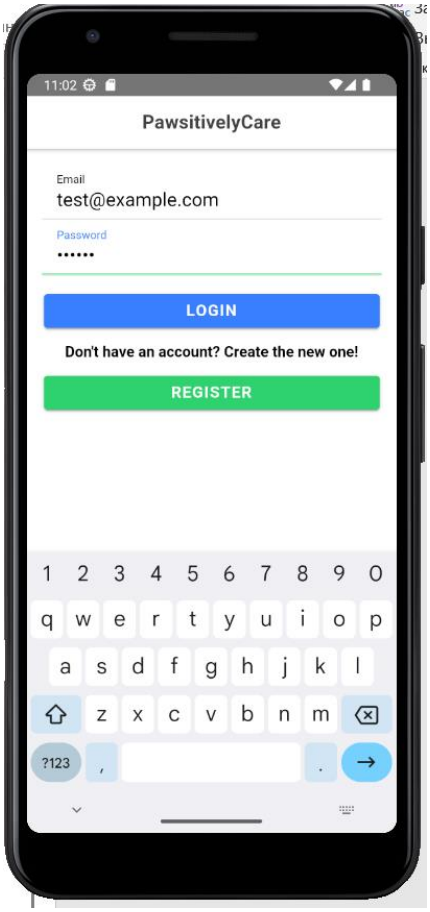


Рисунок 3.1 – Сторінка входу до системи

Якщо користувач зареєстрований у системі, для успішного входу до системи він повинен ввести коректні пошту та пароль, які він використовував при реєстрації у застосунку. Якщо користувач введе некоректні дані, буде показано повідомлення про помилку (рисунок 3.2).

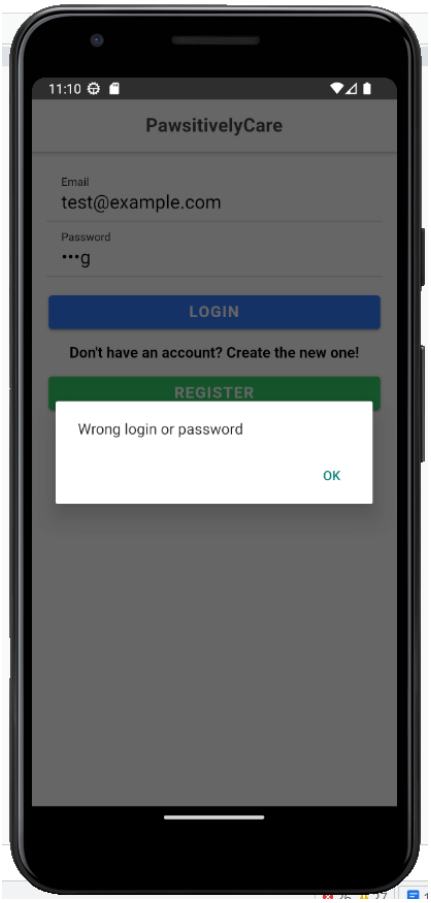


Рисунок 3.2 – Повідомлення про помилку при спробі увійти до системи

Якщо користувач не зареєстрований у системі, він повинен натиснути на «Register», після чого відкриється сторінка для реєстрації у застосунку (рисунок 3.3).

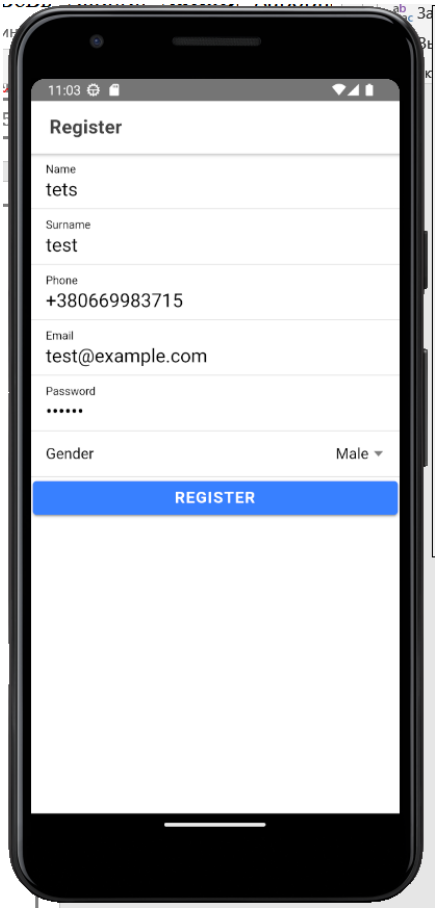


Рисунок 3.3 – Сторінка для реєстрації у застосунку

Користувач повинен ввести потрібні дані у відповідні поля, після чого натиснути на «Register». Якщо при спробі реєстрації виникне помилка, користувачу буде показане повідомлення про помилку (рисунок 3.4). Після успішної реєстрації відкривається сторінка входу до системи.

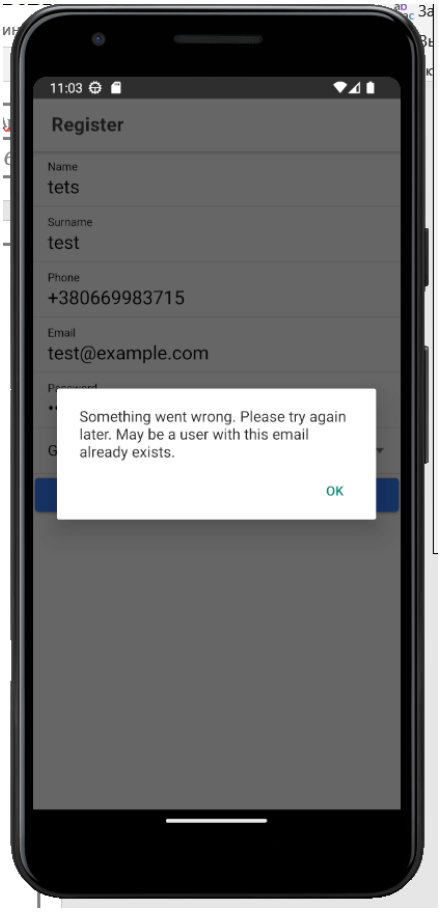


Рисунок 3.4 – Повідомлення про помилку при спробі реєстрації у системі

Після успішного входу до системи відкривається сторінка з постами від інших користувачів (рисунок 3.5). Така ж сторінка відкривається при натисканні на «Posts» у меню додатку.

Рисунок 3.5 – Сторінка з постами від інших користувачів

Пости можна відфільтрувати за локацією, типом та категорією, для цього потрібно увести потрібну локацію у відповідне поле і обрати тип постів (рисунок 3.6) та категорію постів (рисунок 3.7).

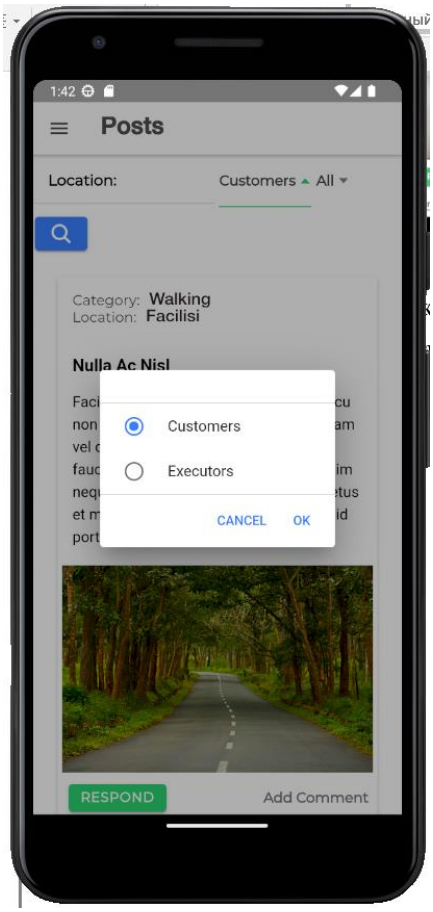


Рисунок 3.6 – Віконце для вибору типу постів

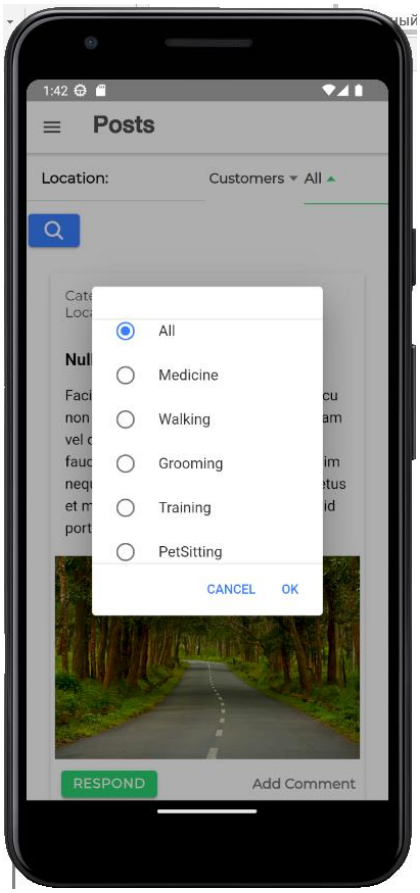


Рисунок 3.7 – Віконце для вибору категорії постів

Користувач може відгукнутися на пост при натисканні на Respond, після чого відкриється чат з користувачем, який створив відповідний пост (рисунок 3.8).

Рисунок 3.8 – Вікно чату з користувачем

Користувач може переглянути коментарі до поста при натисканні на «Comments» під постом (рисунок 3.9). Щоб сховати коментарі, потрібно повторно натиснути на «Comments».

Щоб написати коментар до поста, користувач повинен натиснути на «Add Comment», після чого відкриється сторінка для написання коментаря (рисунок 3.10). Потрібно ввести коментар у відповідне поле, після чого зберегти його при натисканні на «Save» або відмінити його створення при натисканні на «Cancel».

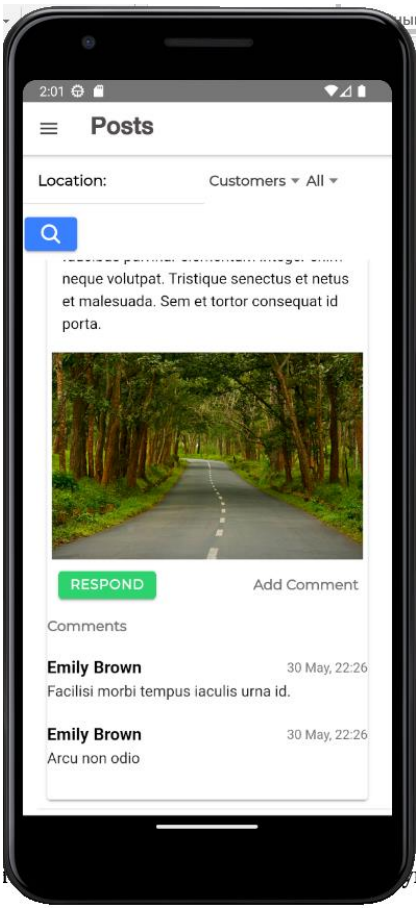


Рисунок 3.9 – Відкриті коментарі під постом

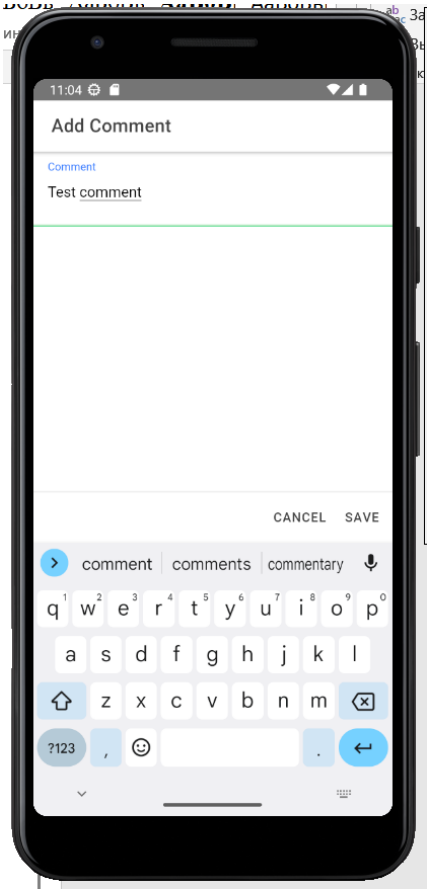


Рисунок 3.10 – Сторінка для написання коментаря

Щоб відкрити меню додатку, користувач повинен натиснути на кнопку з трьома рисками у верхньому лівому кутку веб-застосунку чи зробити свайп вліво (рисунок 3.11). Щоб перейти на певну сторінку, користувач повинен натиснути на відповідний пункт меню.

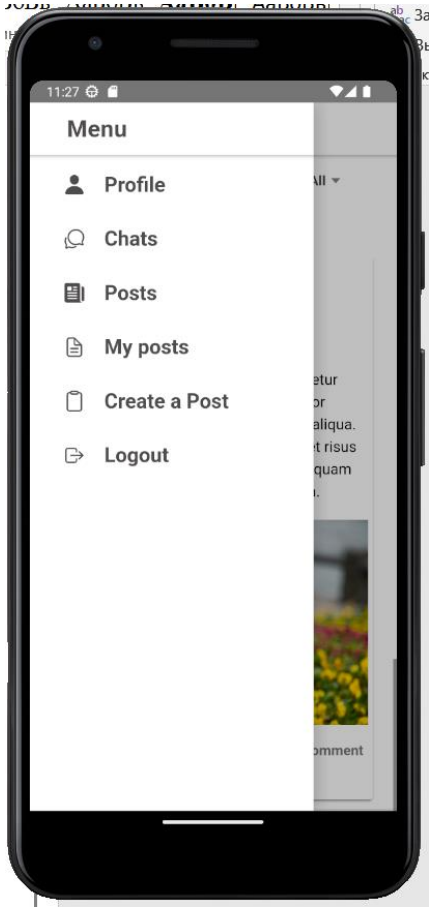


Рисунок 3.11 – Меню додатку

Щоб переглянути дані профілю, користувач повинен натиснути на «Profile» у меню додатку (рисунок 3.12). Щоб відредагувати дані профілю, потрібно натиснути на «Edit» (рисунок 3.13). Після зміни даних для їх збереження потрібно натиснути на «Save».

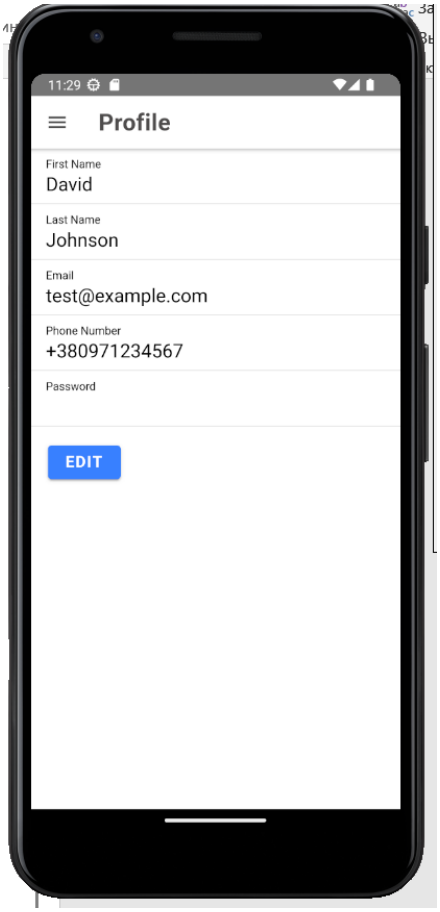


Рисунок 3.12 – Сторінка з даними профілю користувача

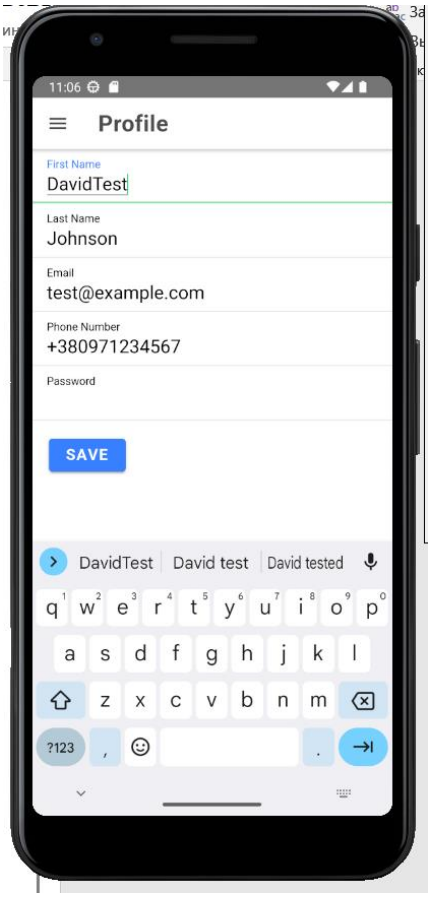


Рисунок 3.13 – Сторінка редагування даних профілю користувача

Щоб переглянути свої чати, користувач повинен натиснути на «Chats» у меню додатку (рисунок 3.14).

Рисунок 3.14 – Сторінка з чатами користувача

Щоб відкрити чат, користувач повинен на нього натиснути (рисунок 3.15). Щоб написати повідомлення до чату, користувач повинен ввести повідомлення у поле знизу та відправити його при натисканні на кнопку у вигляді стрілки. Щоб переглянути деталі поста, до якого був створений чат, потрібно натиснути на кнопку зі знаком «?» (рисунок 3.16).

Рисунок 3.15 – Чат користувача

Рисунок 3.16 – Деталі поста, до якого створений чат

Щоб переглянути свої пости, користувач повинен натиснути на «My Posts» у меню додатку (рисунок 3.17).

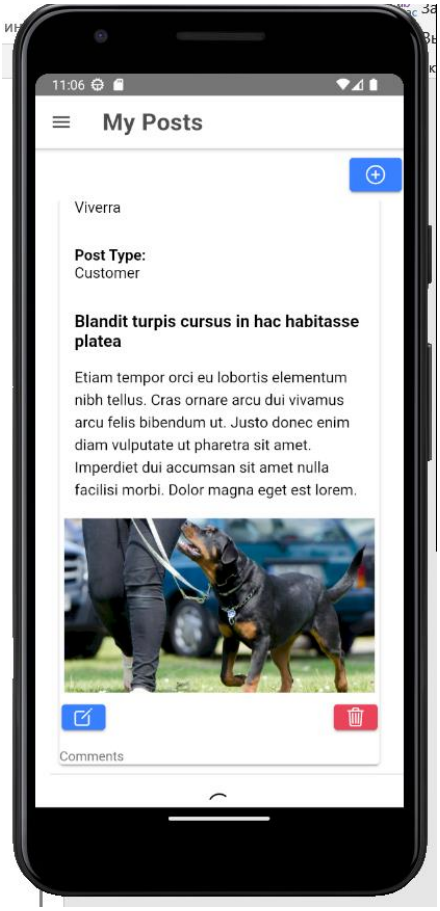


Рисунок 3.17 – Сторінка з постами користувача

Користувач може відредагувати пост при натисканні на блакитну кнопку із зображенням олівця (рисунок 3.18). Щоб видалити пост, користувач повинен натиснути на червону кнопку із зображенням смітника.

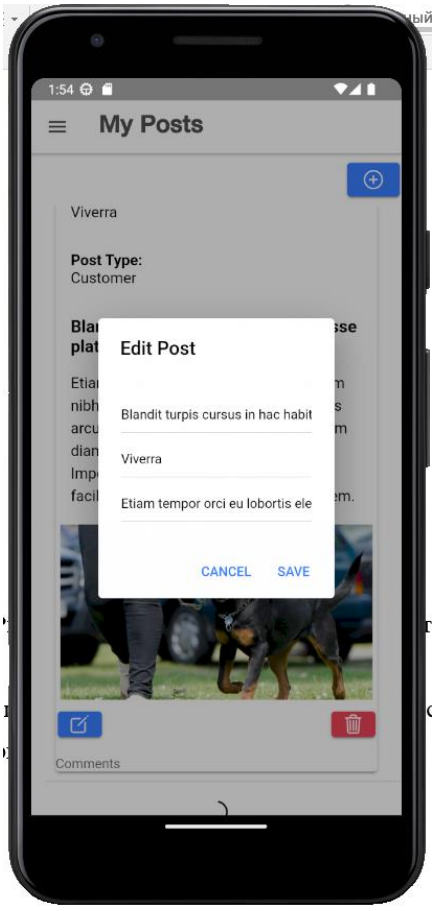


Рисунок 3.18 – Віконце для редагування поста

При натисканні на блакитну кнопку зі знаком «+» відкриється сторінка для створення поста (рисунок 3.19). Така ж сторінка відкривається при натисканні на «Create Post» у меню додатку. Щоб створити пост, користувач повинен ввести потрібні дані у відповідні поля та натиснути на кнопку «Create». Користувач може прикріпити зображення до поста при натисканні на «Add Image», що не є обов’язковим.

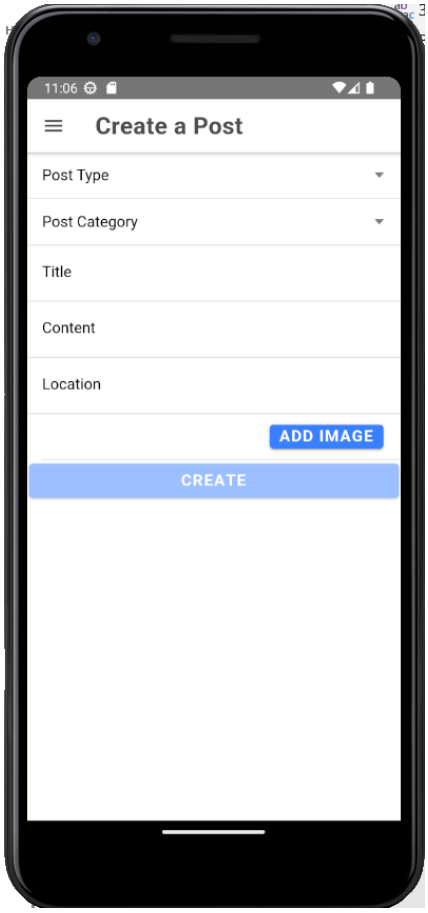


Рисунок 3.19 – Сторінка для створення поста

Щоб вийти із акаунту, користувач повинен натиснути на «Logout» у меню додатку, після чого відкриється сторінка входу до системи.

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2023 р.

ВЕБ-ЗАСТОСУНОК З НАДАННЯ ТА ОТРИМАННЯ ПОСЛУГ ДЛЯ ВЛАСНИКІВ ДОМАШНІХ ТВАРИН

Графічний матеріал

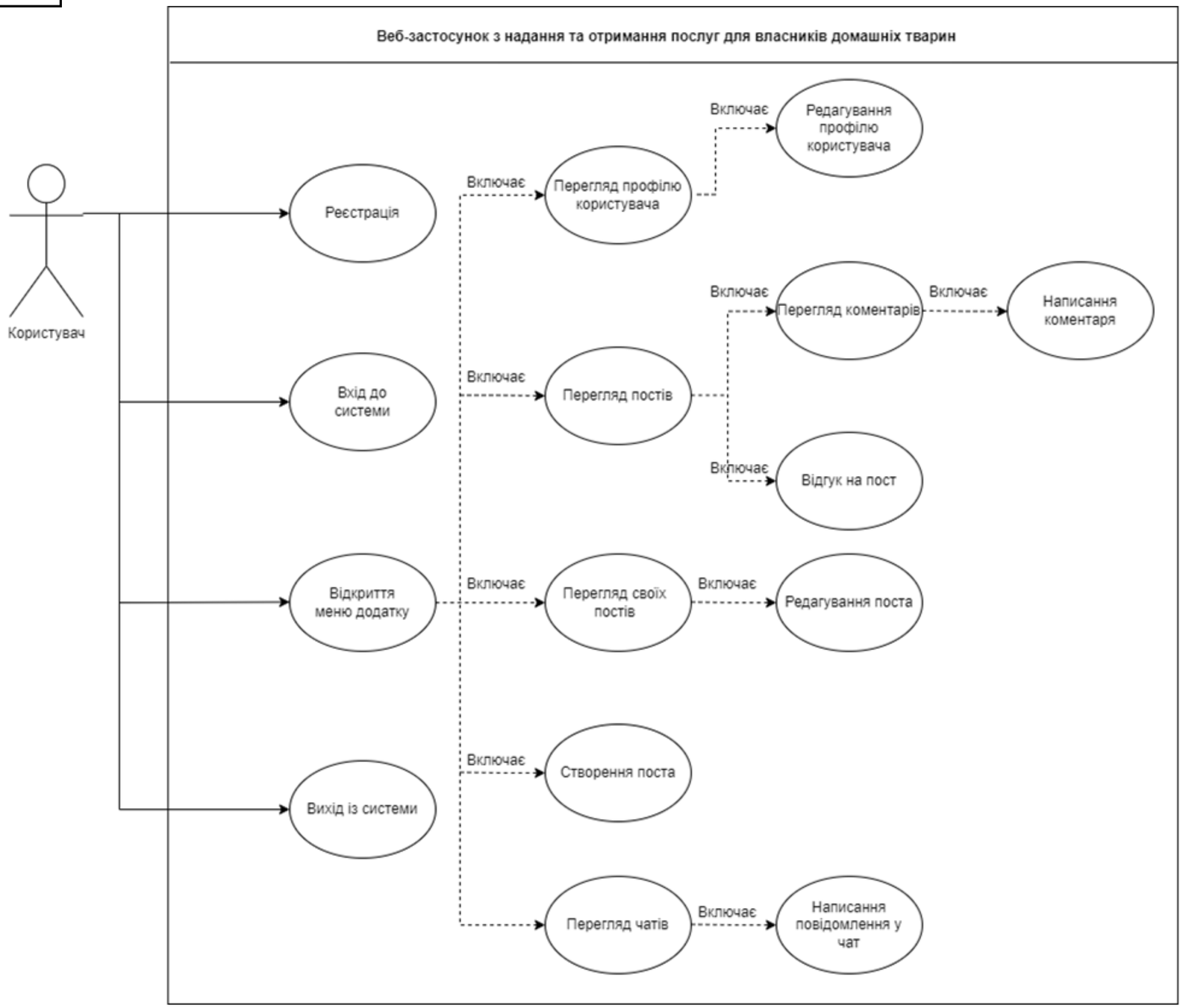
КПІ.ІТ-9208.045440.06.99

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Юлія КРАМАР

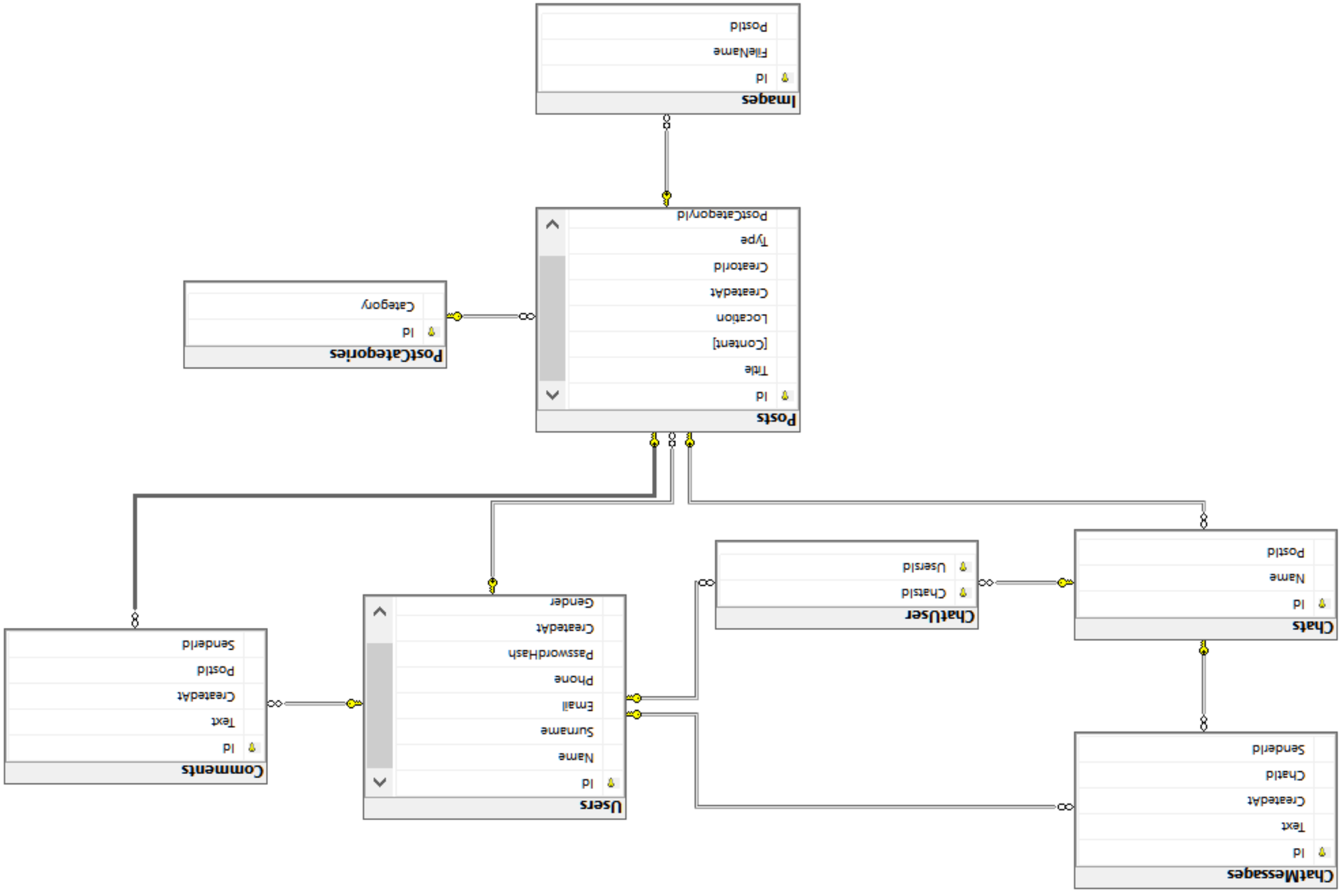
Нормоконтроль:
_____ Ірина ВІТКОВСЬКА

Виконавець:
_____ Валентина ГРИНИК

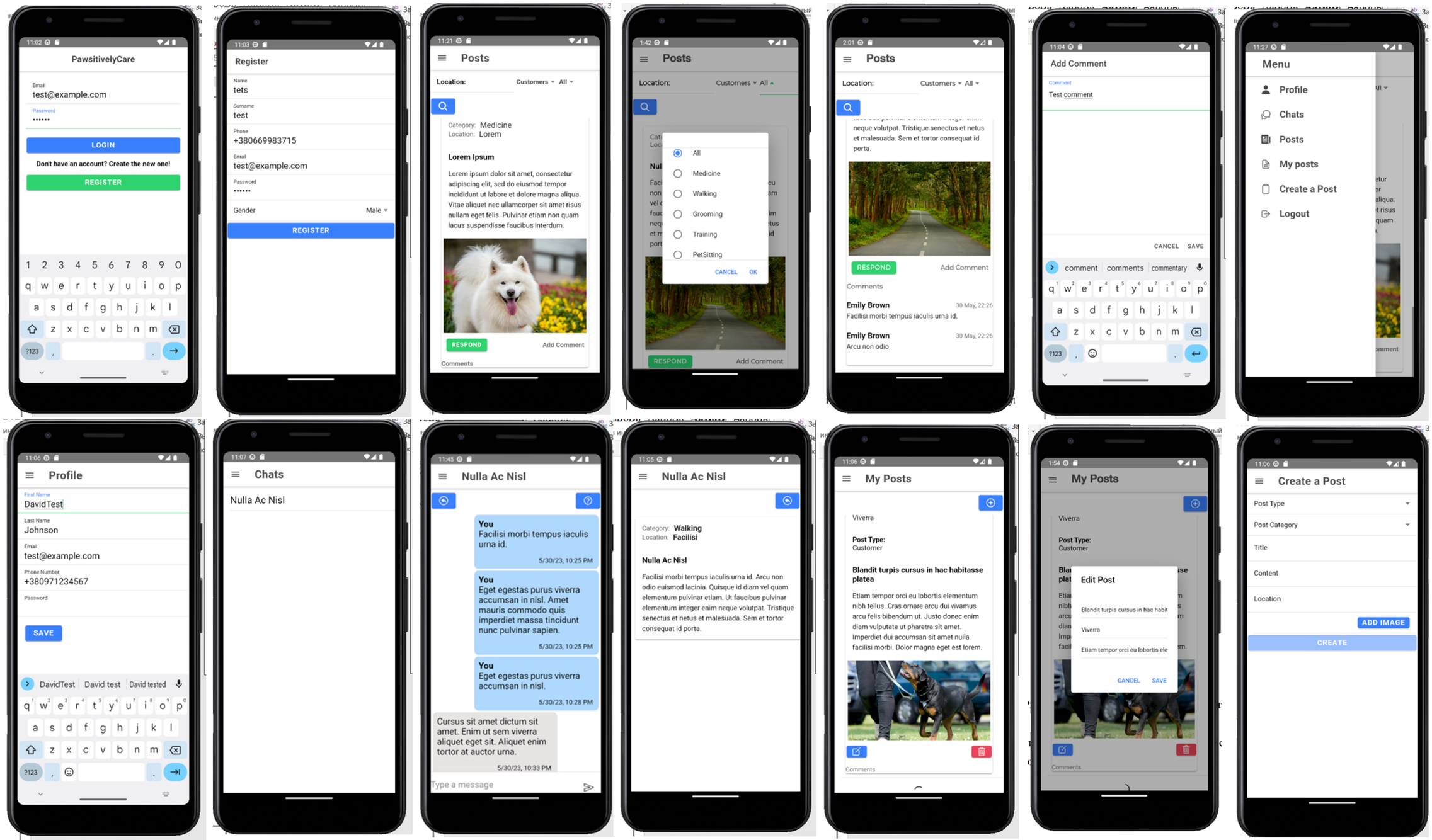
Київ – 2023



					КПІ.ІТ-9208.045440.06.99.СС	
Зм.	Арк.	№ документа	Підпис	Дата	Схема структурна варіантів використань	Літера / Маса / Масштаб
Розробив		Гриник В.О.				
Перевірив		Крамар Ю.М.				Аркуш / Аркушів
Т. кон.						
Н. кон.		Вітковська І.І.			Веб-застосунок з надання та отримання послуг для власників домашніх тварин	КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІТ-92
Затвердив		Жаріков Е.В.				



Зм.	Арк.	№ докумeнта	Підпис	Дата	КПІ.ІТ-9208.045440.07.99.СБД				
Розробив		Гринік В.О.							
Перевірив		Крамар Ю.М.			Схема бази даних		Літера	Маса	Масштаб
Т. кон.									
Н. кон.		Бітковська І.І.							
Затвердив		Жаріков Е.В.			Веб-застосунок з надання та отримання послуг для власників домашніх тварин		Аркуш	Аркушів	
							Аркуш		
					КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІТ-92				



					КПІ.ІТ-9208.045440.08.99.КЕ	
Зм.	Арк.	№ документа	Підпис	Дата		
Розробив		Гриник В.О.			Креслення вигляду екранних форм	Літера / Маса / Масштаб
Перевірив		Крамар Ю.М.				
Т. кон.						Аркуш / Аркушів
Н. кон.		Вітковська І.І.			Веб-застосунок з надання та отримання послуг для власників домашніх тварин	КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІТ-92
Затвердив		Жаріков Е.В.				

UNICHECK

Ім'я користувача:
Лісовиченко Олег Іванович

Дата перевірки:
01.06.2023 18:54:46 EEST

Дата звіту:
01.06.2023 19:08:38 EEST

ID перевірки:
1015375227

Тип перевірки:
Doc vs Internet + Library

ID користувача:
76913

Назва документа: **ІТ-92_Гриник_ПЗ**

Кількість сторінок: **81** Кількість слів: **10385** Кількість символів: **81405** Розмір файлу: **3.57 MB** ID файлу: **1015041071**

11.9%
Схожість

Найбільша схожість: **4.64%** з джерелом з Бібліотеки (ID файлу: **1015013680**)

2.88% Джерела з Інтернету	121	Сторінка 83
11.5% Джерела з Бібліотеки	180	Сторінка 84

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0.76%
Вилучень

Деякі джерела вилучено автоматично (фільтри вилучення: кількість знайдених слів є меншою за **8 слів** та **0%**)

0% Вилучення з Інтернету	25	Сторінка 85
0.76% Вилученого тексту з Бібліотеки	37	Сторінка 85