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)
"___"_____2024 р.

Дипломний проєкт

на здобуття ступеня бакалавра

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

спеціальності «121 Інженерія програмного забезпечення»

на тему: Мобільний застосунок для пошуку виконавців та замовників послуг на прикладі догляду за тваринами

Виконав студент IV курсу, групи ІП-01
(шифр групи)

Смислов Даніл Юрійович _____
(прізвище, ім'я, по батькові) (підпис)

Керівник професор кафедри ІПІ, д.т.н., проф. Павлов О.А. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент професор кафедри ІСТ, д.т.н., проф., Корнага Я.І. _____
(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 –2024

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)
"___"_____2024 р.

ЗАВДАННЯ
на дипломний проєкт студенту

Смислову Данілу Юрійовичу
(прізвище, ім'я, по батькові)

1. Тема проєкту Мобільний застосунок для пошуку виконавців та замовників послуг на прикладі догляду за тваринами

керівник проєкту д.т.н., проф. Павлов Олександр Анатолійович
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від « 27 » травня 2024 р. №2112-с

2. Термін подання студентом проєкту « 17 » червня 2024 року

3. Вихідні дані до проєкту: технічне завдання

4. Зміст пояснювальної записки
1) Передпроєктне обстеження предметної області: аналіз предметної області, аналіз існуючих рішень, опис бізнес-процесів, постановка задачі.
2) Розроблення вимог до програмного забезпечення: варіанти використання програмного забезпечення, аналіз системних вимог, розроблення функціональних вимог, розроблення нефункціональних вимог.
3) Конструювання та розроблення програмного забезпечення: архітектура програмного забезпечення, обґрунтування вибору засобів розробки, конструювання програмного забезпечення, аналіз безпеки даних.
4) Аналіз якості та тестування програмного забезпечення: аналіз якості ПЗ, опис процесів тестування, опис контрольного прикладу.
5) Розгортання та супровід програмного забезпечення: розгортання програмного забезпечення, супровід програмного забезпечення.

5. Перелік графічного матеріалу

1) Креслення вигляду екранних форм

2) Схема структурна компонентів програмного забезпечення

3) Схема бази даних

6. Консультанти розділів проєкту

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

7. Дата видачі завдання «11» березня 2024 року

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Вивчення рекомендованої літератури	11.03.2024	
2	Аналіз існуючих методів розв’язання задачі	21.03.2024	
3	Постановка та формалізація задачі	29.03.2024	
4	Розробка інформаційного забезпечення	04.04.2024	
5	Алгоритмізація задачі	09.04.2024	
6	Обґрунтування вибору використаних технічних засобів	10.04.2024	
7	Розробка програмного забезпечення	13.05.2024	
8	Налагодження програми	17.05.2024	
9	Виконання графічних документів	25.05.2024	
10	Оформлення пояснювальної записки	01.06.2024	
11	Подання ДП на попередній захист	03.06.2024	
12	Подання ДП рецензенту	10.06.2024	
13	Подання ДП на основний захист	14.06.2024	

Студент _____ Даніл СМИСЛОВ
(підпис) (ініціали, прізвище)

Керівник _____ Олександр ПАВЛОВ
(підпис) (ініціали, прізвище)

АНОТАЦІЯ

Пояснювальна записка дипломного проєкту складається з п’ятьох розділів, містить 44 таблиці, 40 рисунків та 39 джерел – загалом 96 сторінок.

Дипломний проєкт присвячений розробці мобільного застосунку для пошуку виконавців та замовників послуг на прикладі догляду за тваринами.

Метою дипломного проекту є полегшення процесу пошуку виконавців та замовників з формуванням конкретних релевантних пропозицій, враховуючи задані критерії, та часткова автоматизація процесу підбору можливих виконавців.

Об'єкт дослідження: пошук виконавців та замовників послуг.

Предмет дослідження: мобільний застосунок для пошуку виконавців та замовників послуг.

У розділі передпроєктного дослідження предметної області було проведено аналіз ринку перетримки тварин, оглянуто існуючі рішення, виділено їх сильні сторони та можливості для покращення. Описано основні бізнес-процеси. Сформовано основні функціональні задачі.

Розділ розроблення вимог до програмного забезпечення присвячений формуванню основних функціональних та нефункціональних вимог. Побудовано діаграму варіантів використання, описано основний функціонал. Аргументовано системні вимоги програмного забезепечення.

У розділі конструювання вимог до програмного забезпечення було описано архітектуру, засоби розробки, описано проект, стороннє програмне забезпечення, що використовується при розробці. Проаналізовано рівень безпеки даних в застосунку.

Розділ аналізу якості та тестування програмного забезпечення присвячений аналізу коду, опису процесів тестування та контрольного прикладу.

У розділі розгортання та супроводу програмного забезпечення описано розгортання серверної частини застосунку та збірка файлів, що можуть використовуватись для публікації клієнтської частини застосунку в крамницю.

КЛЮЧОВІ СЛОВА: МОБІЛЬНИЙ ЗАСТОСУНОК, IOS, REACT NATIVE, ASP.NET, DOCKER, БАЗА ДАНИХ, CLIENT-SERVER

ABSTRACT

The explanatory note of the diploma project consists of five sections, contains 44 tables, 40 figures and 39 sources – in total 96 pages.

The diploma project is dedicated to the development of a mobile application for searching for performers and customers of services on the example of pet care. The goal of the diploma project is to make easier the process of finding providers and customers by forming specific relevant proposals, taking into account given criteria, and partial automation of the process of selecting possible performers.

Object of research: search for performers and customers of services.

Subject of research: a mobile application for searching for performers and customers of services.

In the pre-project research section of the subject area, an analysis of the pet boarding market was conducted, existing solutions were reviewed, their strengths and opportunities for improvement were highlighted. The main business processes are described. The main functional tasks are formed.

The section on developing software requirements is devoted to the formation of the main functional and non-functional requirements. A use case diagram was built, the main functionality was described. The system requirements of the software are substantiated.

The section on designing software requirements described the architecture, development tools, described the project, and third-party software used in development.

The section on software quality analysis and testing is devoted to the analysis of the code base, description of testing processes, and description of a control example.

The section on software deployment and maintenance describes the deployment of the server part of the application and the assembly of files that can be used to publish the client part of the application to the store.

KEYWORDS: MOBILE APP, IOS, REACT NATIVE, ASP.NET, DOCKER, DATABASE, CLIENT-SERVER.

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	КПІ.ІП-0127.045490.01.91	Технічне завдання	20	
3	А4	КПІ.ІП-0127.045490.02.81	Пояснювальна записка	96	
4	А4	КПІ.ІП-0127.045490.03.12	Текст програми	43	
5	А4	КПІ.ІП-0127.045490.04.51	Програма та методика тестування	6	
6	А4	КПІ.ІП-0127.045490.05.34	Керівництво користувача	29	
7	А3	КПІ.ІП-0127.045490.06.99	Графічний матеріал	4	

КПІ.ІП-0127.045490.00.90

Змін.	Арк.	№ докум.	Підп.	Дата
Розроб.		Смислов Д.Ю.		
Перевір.		Павлов О.А.		
Н.контр.		Головченко М.М.		
Затв.		Жаріков Е.В.		

Відомість дипломного проєкту

Літ.	Аркуш	Аркушів
	1	

КПІ ім. Ігоря Сікорського
ФІОТ каф. ІПІ гр. ІП-01

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”

Завідувач кафедри

_____ Едуард ЖАРІКОВ

“___” _____ 2024 р.

Мобільний застосунок для пошуку виконавців та замовників послуг на прикладі догляду за тваринами

Технічне завдання

КПІ.ІП-0127.045490.01.91

“ПОГОДЖЕНО”

Керівник проєкту:

Олександр ПАВЛОВ

Нормоконтроль:

Максим ГОЛОВЧЕНКО

Виконавець:

Даніл СМИСЛОВ

Київ – 2024

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: мобільний застосунок для пошуку виконавців та замовників послуг на прикладі догляду за тваринами.

Галузь застосування:

Наведене технічне завдання поширюється на розробку прикладного програмного забезпечення, а саме мобільного застосунку для пошуку та підбору виконавців, що зможуть посидіти з домашнім улюбленцем замовників [PetHome], котре використовується для пошуку, підбору та комунікації між виконавцем та замовником та призначена для широкого кола користувачів.

2 ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки мобільного застосунку для пошуку виконавців та замовників послуг на прикладі догляду за тваринами 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 пошуку виконавців, що зможуть подбати про тварин замовників протягом зазначеного часу за певну грошову винагороду.

Метою розробки є полегшення процесу пошуку виконавців та замовників з формуванням конкретних релевантних пропозицій, враховуючи задані критерії, та часткова автоматизація процесу підбору можливих виконавців.

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вимоги до функціональних характеристик

Програмне забезпечення повинно забезпечувати виконання наступних основних функцій:

4.1.1 Користувацького інтерфейсу

– мобільний застосунок підтримує лише вертикальне розташування пристрою;

– відображення профілю користувача з його фото, ім’ям, календарем зайнятості, переліком контактів (рисунок 4.1) ;

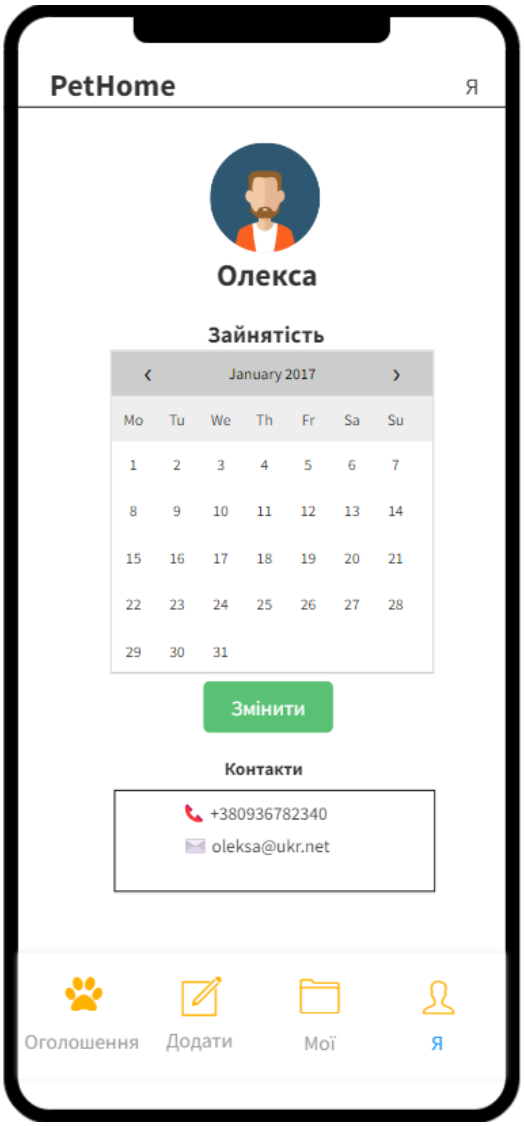


Рисунок 4.1 – Вікно профілю користувача

– відображення конкретного оголошення з умовами виконання(дата початку, дата закінчення, сума винагороди, опис тварини, фото тварини, ім’я тварини) ;

– відображення вікна аутентифікації за логіном та паролем;

– відображення вікна реєстрації;

– відображення вікна додавання оголошення (рисунок 4.2) ;

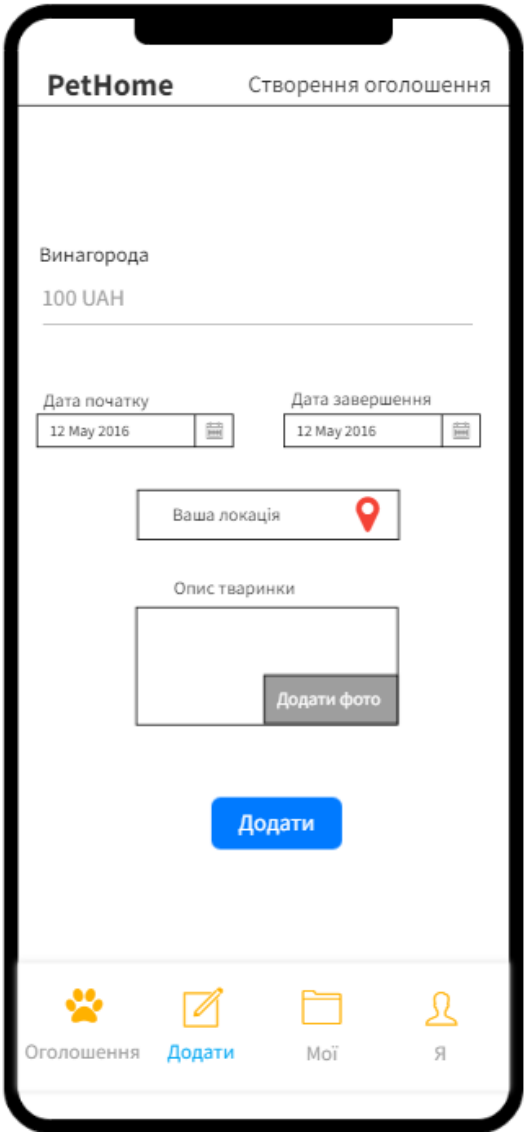


Рисунок 4.2 – Вікно додавання оголошення

– відображення списку поданих оголошень (рисунок 4.3) ;

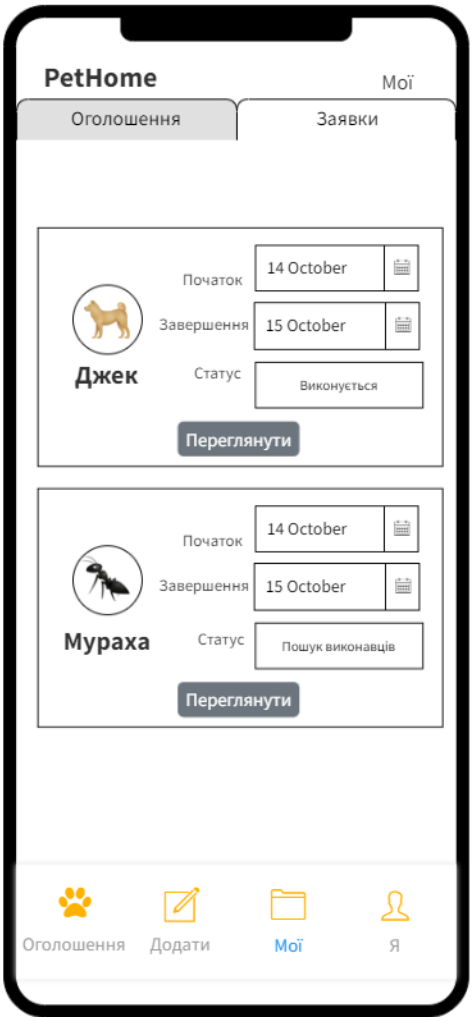


Рисунок 4.3 – Вікно списку поданих оголошень

– відображення списку виконавців, що відгукнулись на конкретне оголошення з можливістю приймати та відміняти заявки (рисунок 4.4) ;

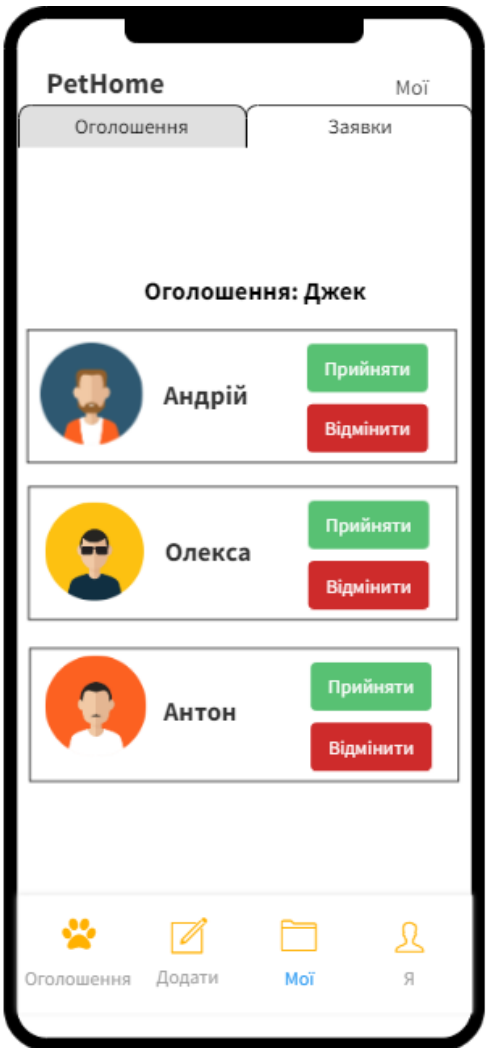


Рисунок 4.4– Вікно списку виконавців, що відгукнулись на конкретне оголошення

– відображення списку оголошень з фільтрами, що можна застосувати (рисунок 4.5) ;

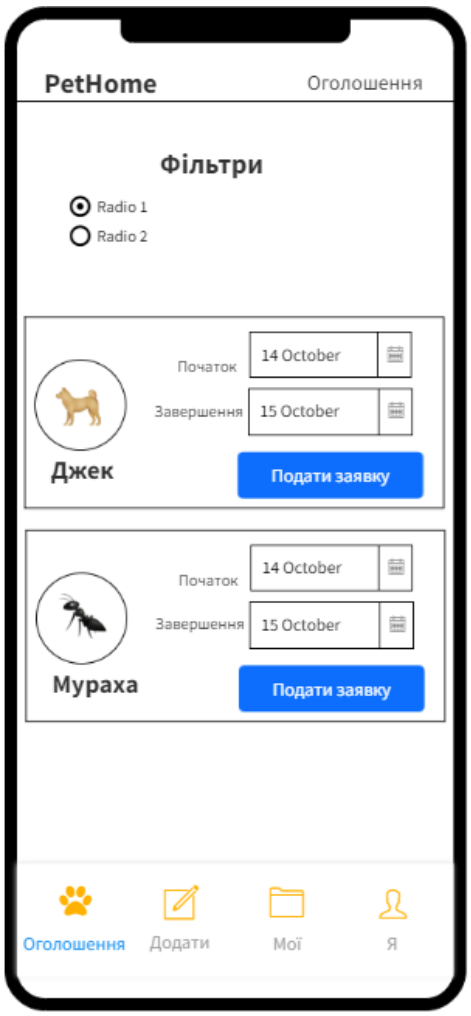


Рисунок 4.5 – Вікно списку оголошень

– відображення списку заявок, які подав виконавець або автоматично отримав пропозиції від системи (рисунок 4.6) ;

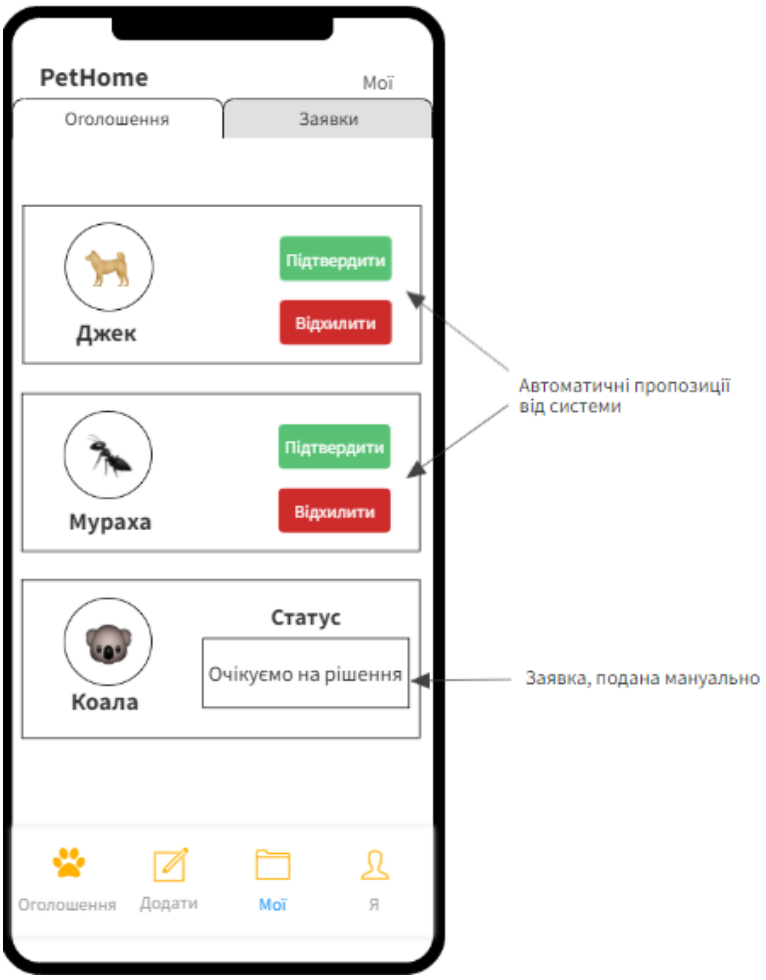


Рисунок 4.6 – Вікно поданих заявок

4.1.2 Для користувача:

4.1.2.1 Можливість авторизуватися за логіном та паролем

4.1.2.1.1 Якщо логін та пароль не є пустими – надіслати форму

4.1.2.1.2 Якщо логін або пароль є пустими – візуально відобразити незаповнені поля

4.1.2.2 Можливість перегляду списку оголошень (рис. 4.5)

4.1.2.3 Можливість перегляду конкретного оголошення

4.1.2.3.1 Можливість перегляду періоду виконання оголошення, суми винагороди, фото тварини, короткого опису тварини

4.1.2.4 Можливість застосування певних фільтрів до оголошень, що показуються (рис. 4.5)

4.1.2.4.1 Можливість застосування фільтрів щодо суми винагороди за виконання оголошення

4.1.2.4.2 Можливість застосування фільтрів щодо статусу оголошення

4.1.2.4.3 Можливість застосування фільтрів щодо терміну виконання оголошення, що не суперечить вільним датам користувача

4.1.2.5 Можливість перегляду профілів інших користувачів

4.1.2.5.1 Можливість перегляду персональної інформації користувача та його фото

4.1.2.5.2 Можливість перегляду календаря з графіком зайнятості користувача

4.1.2.6 Можливість додавання оголошень

4.1.2.6.1 Можливість вказувати суму винагороди за виконання оголошення, що має бути числовим значенням, більшим за 0 та меншим за 100000

4.1.2.6.2 Можливість вказувати час виконання оголошення, дата закінчення виконання має бути пізніше за дату початку виконання

4.1.2.6.3 Можливість автоматично або мануально вказувати локацію виконання оголошення

4.1.2.6.4 Можливість введення опису тварини, довжиною від 10 до 500 символів

4.1.2.6.5 Можливість прикріпляти фото тварини, що задовольняють умовам, зазначеним в пункті 4.5.1

4.1.2.6.6 За умови некоректного введення даних – підсвічування неправильно заповнених полів

4.1.2.6.7 Якщо поля заповнено правильно – надсилання форми

4.1.2.7 Можливість автоматичного підбору та відправки пропозицій <= 10 виконавцям, що вільні в зазначені дати одразу після додавання оголошення замовником

4.1.2.8 Можливість подавання заявки на виконання

4.1.2.9 Можливість підтверджувати або відхиляти пропозиції, запропоновані виконавцям системою (рис. 4.6)

4.1.2.10 Можливість замовнику обирати конкретного виконавця серед тих, хто відгукнулись (рис 4.4)

4.1.2.11 Можливість замовнику відмічати оголошення як «виконане» після виконання виконавцем заявки.

4.1.2.12 Можливість користувачу створювати новий профіль

4.1.2.12.1 Можливість введення ім’я та прізвища, з довжиною більше або дорівнює 2 символи, зазначення статі

4.1.2.12.2 Можливість прикріплення фото, що задовольняє умовам, описаним в пункті 4.5.1

4.1.2.12.3 Можливість введення email-адреси та номеру телефону зазначених форматів

4.1.2.12.4 Можливість вибору унікального логіну

4.1.2.12.5 Можливість введення та підтвердження пароля з довжиною більше або дорівнює 8 символів, хоча б з однією великою латинською літерою та спеціальним символом

4.1.2.12.6 Якщо введені дані не задовільняють умови – візуальне пісвічування неправильно заповнених полів

4.1.2.12.7 Якщо дані введено коректно – відправка форми

4.1.2.13 Можливість користувачу редагувати дані свого профіля

4.1.2.13.1 Оновлені дані мають задовольняти правила реєстрації нового профіля, що зазначені в підкунктах пункту 4.1.2.11

4.1.2.14 Можливість замовнику редагувати дані поданого оголошення

4.1.2.14.1 Оновлені дані мають задовольняти правила створення нового оголошення, що зазначені в пінпунктах пункту 4.1.2.6

4.1.2.15 Можливість користувачу налаштовувати свої вільні дати в профілі, що будуть враховуватись при автоматичному підборі

4.1.2.15.1 Можливість відмічення певних дат як «зайняті» та відміна такого позначення, щоб дати відображались як «вільні»

4.1.2.16 Можливість користувачу виходити зі свого облікового запису

4.1.2.17 Можливість користувачу видаляти свій профіль

4.1.3 Для адміністратора системи:

– можливість перегляду профілей користувачів;
– можливість перегляду оголошень;
– видалення оголешень користувачів;
– видалення профілей користувачів.

4.1.4 Додаткові вимоги:

– додаткові вимоги не висуваються.

4.2 Вимоги до надійності

Передбачити контроль введення інформації та захист від некоректних дій користувача. Забезпечити цілісність інформації в базі даних, зберігання захешованих паролів. Забезпечити можливість повного видалення особистих даних користувача. Реалізувати безпечний обмін даними між клієнтом та сервером, використовуючи протокол HTTPS, що передає дані у зашифрованому виді.

Для авторизації та аутентифікації використовувати токен доступу стандарту JWT. Використовувати пару токенів – access та refresh. Забезпечити невелику тривалість актуальності access-токену та за наявності актуального refresh-токену оновлювати пару токенів користувача. Задля підвищення рівня безпеки та зменшення ризиків XSS-атак, токени зберігати у вигляді HttpOnly cookie.

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 сучасних пристроях з операційною системою iOS починаючи з версії 13.4.

Мінімальна рекомендована конфігурація технічних засобів:

– версія операційної системи iOS: 13.4;
– підключення до мережі Інтернет зі швидкістю від 5 мегабіт;
– об’єм вільної пам’яті: 50 Мб;

– об’єм оперативної пам’яті: можливість пристрою підтримки iOS версії 13.4;

– характеристики процесора: можливість пристрою підтримки iOS версії 13.4;

– роздільна здатність екрану: 750x1334 пікселів.

4.5 Вимоги до інформаційної та програмної сумісності

Програмне забезпечення повинно працювати під управлінням операційних систем iOS версій, починаючи з 13.4. Застосунок підтримує лише вертикальне розрашування пристрою. Роздільна здатність екрану від 750x1334 пікселів. Співвідношення сторін 9:16, або 9:19.5.

4.5.1 Вимоги до вхідних даних

Власник тварини під час додавання оголошення може завантажувати фото у форматах .jpg/png/jpeg. Також при реєстрації користувач може прикріпляти своє фото у форматах .jpg/png/jpeg. Максимальний розмір завантажуваних зображень – 5 Мб. Співвідношення сторін не має перевищувати 16:9, або бути меншим за 9:16.

4.5.2 Вимоги до вихідних даних

Вимоги до вихідних даних не висуваються.

4.5.3 Вимоги до мови розробки

Розробку виконати на мовах програмування C#, Javascript.

4.5.4 Вимоги до середовища розробки

Розробку виконати в середовищах Visual Studio та Visual Studio Code.

4.5.5 Вимоги до представленню вихідних кодів

Вихідний код програми має бути представлений у вигляді репозиторію на GitHub.

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

Згенерувати інсталяційну версію клієнтської частини програмного забезпечення.

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 компонентів програмного забезпечення;
– схема бази даних;
– креслення вигляду екранних форм;

5.2 Спеціальні вимоги до програмної документації

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

6 СТАДІЇ І ЕТАПИ РОЗРОБКИ

№	Назва етапу	Строк	Звітність
1.	Вивчення літератури за тематикою проекту	11.03.2024	
2.	Розробка технічного завдання	29.03.2024	Технічне завдання
3.	Аналіз вимог та уточнення специфікацій	05.04.2024	Специфікації програмного забезпечення
4.	Проектування структури програмного забезпечення, проектування компонентів	15.04.2024	Схема структурна програмного забезпечення та специфікація компонентів (діаграма класів, схема алгоритму)
5.	Програмна реалізація програмного забезпечення	13.05.2024	Тексти програмного забезпечення
6.	Тестування програмного забезпечення	17.05.2024	Тести, результати тестування
7.	Розробка матеріалів текстової частини проекту	22.05.2024	Пояснювальна записка
8.	Розробка матеріалів графічної частини проекту	25.05.2024	Графічний матеріал проекту
9.	Оформлення технічної документації проекту	01.06.2024	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: Мобільний застосунок для пошуку виконавців та замовників послуг на прикладі догляду за тваринами

КПІ.ІП-0127.045490.02.81

Київ – 2024

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

IDE	–	Integrated Development Environment – інтегроване середовище розробки.
API	–	Application programming interface, прикладний програмний Інтерфейс
IT	–	Інформаційні технології
JWT	–	JSON Web Token
ПЗ	–	Програмне забезпечення
RN	–	React Native

ВСТУП

З кожним роком домашні тварини стають все більш популярнішими. Усе більше й більше людей заводять собі меншого друга. Також люди все частіше подорожують, що може викликати складнощі у догляді за домашньою твариною.

У деяких країнах за кордоном вже працюють застосунки, що допомагають налаштувати комунікацію між власниками домашніх тварин та виконавцями, що можуть перетримати їх протягом певного часу.

У нашій країні ця проблема частково вирішується за допомогою груп та чатів, створених людьми. Є також кілька сервісів, що пропонують пошук виконавців послуг (далі виконавців) для перетримки, проте вони мають можливості для покращення та, здебільшого, реалізовані як WEB-застосунки.

Є проблеми, що сповна не вирішуються жодним з наявних програмних продуктів. У більшості розроблених застосунків підбір виконавців відбувається лише мануальним шляхом, що негативно впливає на час пошуку. Через брак гнучкого налаштування графіку завантаженості виконавців, можуть виникати непорозуміння при комунікації з замовниками. Відсутність сповіщень в режимі реального часу негативно впливає на швидкість, з якою користувачі реагують на пропозиції.

Розв’язанню цих проблем присвячений даний дипломний проект. Розроблений мобільний застосунок має на меті полегшити пошук та частково автоматизувати його.

Вибір мобільної платформи також має позитивно вплинути на актуальність розробки, оскільки щорічно зростає кількість завантажуваних мобільних застосунків.

1 ПЕРЕДПРОЄКТНЕ ОБСТЕЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз предметної області

Тема перетримки домашніх тварин стає щорічно більш популярною. На це впливає загальна тенденція зміна способу життя людей на більш активний та загальне збільшення кількості домашніх тварин.

Словосполучення пет сіттінг походить із англійської мови та передбачає тимчасовий догляд за домашніми тваринами. Це може відбуватись у власника вдома або у спеціаліста, який надає такі послуги. Послуга може бути оплачуваною. Зазвичай цей період триває від кількох днів до тижня, що відповідає середній тривалості відпустки або короткого відрядження.

Згідно з дослідженням глобального ринку перетримки тварин у світі, що було проведено індійською компанією BrainyInsights у 2022 році, ринок було оцінено приблизно в 2.48 мільярди доларів та очікується сукупний середньорічний темп росту 10.63% [1]. Регіоном з найбільш високим рівнем поширеності послуги пет сіттінгу було визначено Північну Америку. Серед факторів, що вплинули на цей показник, було виділено стиль життя населення, що часто включає завантажений графік, часті подорожі, розвиток культури сприйняття тварини як повноцінного члена родини.

Щодо України. Зараз проблему перетримки тварин вирішують кількома способами:

– готелі для тварин;
– оголошення на дошках оголошень;
– спільноти в соціальних мережах;
– WEB-застосунки для пошуку пет сіттерів;
– мобільні застосунки для пошуку пет сіттерів.

У рамках даного дипломного проекту було розроблено саме мобільний застосунок. Це різновид програмного забезпечення, призначеного для роботи на портативних пристроях. До прикладу, на смартфонах, або планшетах [2].

Ринок мобільних застосунків щорічно зростає. Кількість завантажень сягає кількох сотень мільярдів кожного року, діаграму представлено на рисунку 1.1 [3].

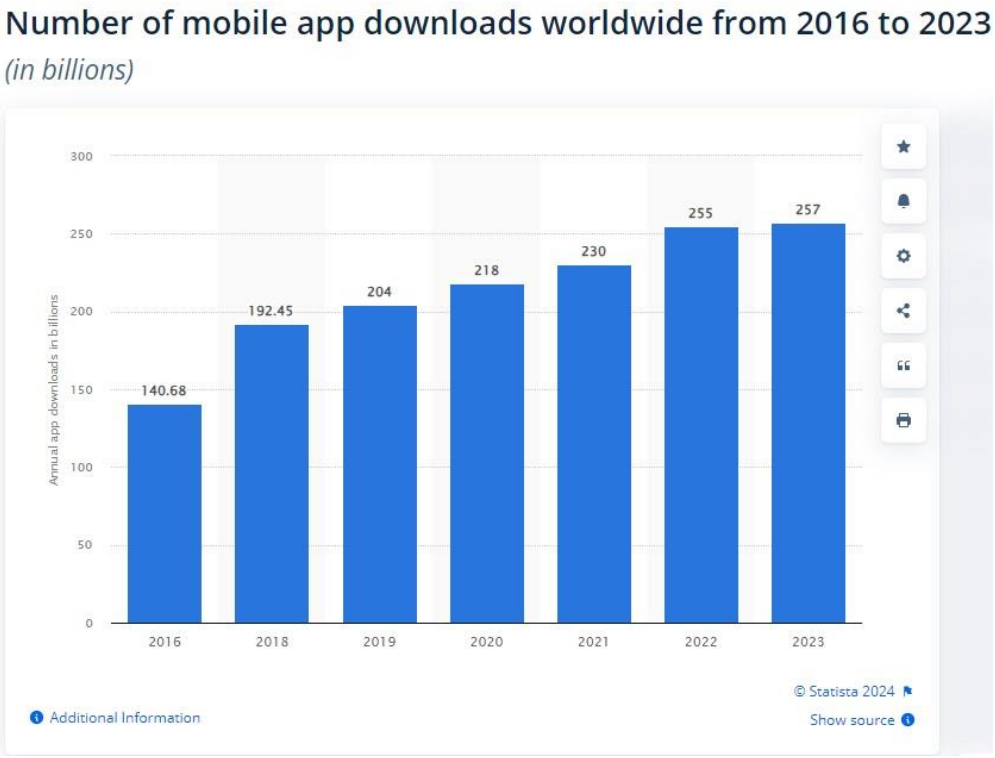


Рисунок 1.1 – Діаграма кількості завантажень мобільних застосунків за роками [3]

Задля зменшення часу підбору виконавців необхідно враховувати певні критерії, такі, як: відстань між користувачами, графік завантаженості виконавців, винагорода за перетримку. Також слід частково автоматизувати процес підбору виконавців, щоб система надавала рекомендації та генерувала заявки для можливих виконавців, що задовольняють критеріям. Користувачам мають приходити сповіщення у режимі реального часу, щоб вони могли швидше відгукуватись на нові пропозиції. Завдяки цьому частина рутинних процесів буде відбуватись автоматично при створенні оголошення та час пошуку виконавців або замовників послуг зменшиться.

Реалізовані мобільні застосунки в даній предметній області не реалізовують часткову автоматизацію процесу підбору та лише один застосунок частково працює з організацією графіку завантаженості виконавця.

Тому в рамках даного дипломного було реалізовано мобільний застосунок для пошуку виконавців та замовників послуг на прикладі пет сіттінгу, що частково автоматизує процес пошуку, враховуючи задані критерії, реалізовує гнучкий менеджмент графіку завантаженості виконавців [PetHome].

1.2 Аналіз існуючих рішень

Проаналізуємо відоме на сьогодні алгоритмічне забезпечення у даній області та технічні рішення, що допоможуть у реалізації мобільного застосунку для пошуку виконавців та замовників послуг на прикладі пет сіттінгу. Далі будуть розглянуті готові програмні рішення, допоміжні програмні засоби та засоби розробки.

1.2.1 Аналіз відомих програмних продуктів

Задачу перетримки частково вирішують готелі для тварин. Однак вони мають декілька фундаментальних відмінностей у порівнянні з підбором виконавців для тимчасового догляду за тваринами. Загалом, середня ціна за добу у готелі для тварин буде трохи вищою, ніж у приватних осіб. Також необхідно враховувати той факт, що в готелі для тварин улюбленцю не будуть приділяти достатньо часу на гру та повноцінний догляд. Ймовірно, його будуть утримувати у клітці і випускати на невеликі проміжки часу для фізичних вправ.

Щодо готових мобільних застосунків з пошуку виконавців для перетримки, в Україні в App Store було знайдено 2 рішення: PetLive та hipaw[4]. Слід зазначити, що PetLive також має WEB-версію застосунку [5].

У рамках порівняння та аналізу цих програмних рішень не слід зважати на інтерфейс користувача та деталі реалізації поза основним функціоналом. На рисунку 1.2 представлено сторінку профілю користувача в застосунку PetLive.

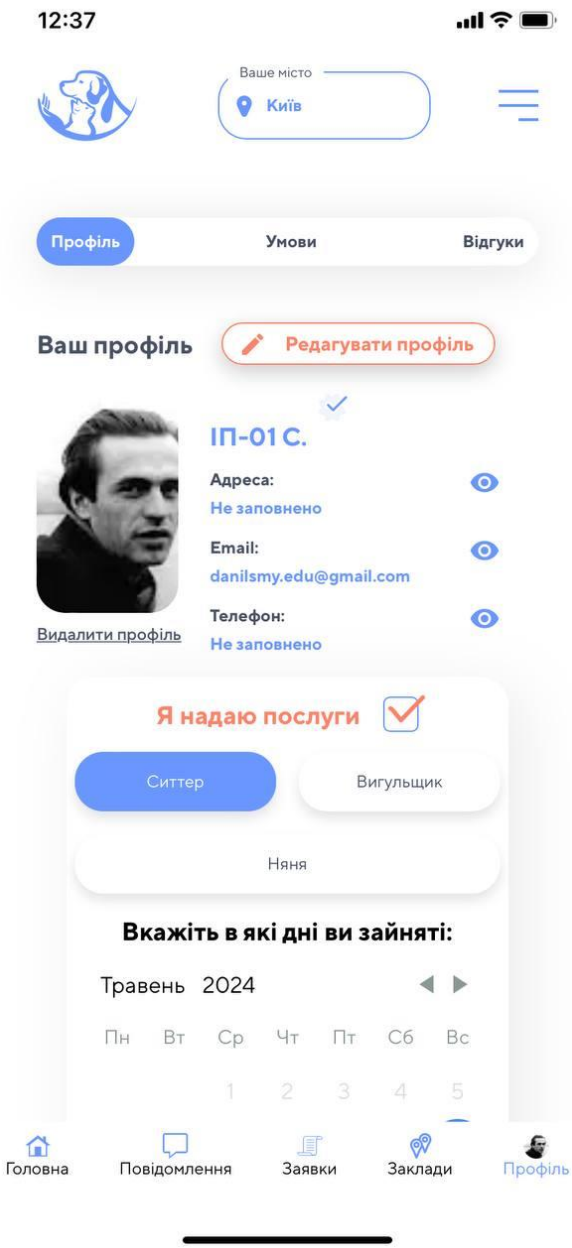


Рисунок 1.2 – Сторінка профілю застосунку PetLive [5]

Слід звернути увагу на ключові моменти: місцеположення обирається лише з наданого переліку міст. У випадку великих міст-мільйонників було б добре уточнювати місцеположення. До прикладу, райони міста, або точну адресу. Адже відстань між користувачами може бути значною і співпраця може бути некомфортною навіть якщо вони знаходяться в одному місці.

Позитивним є те, що користувач може деталізувати які саме послуги він надає, з якими тваринами готовий працювати, яка вартість його послуг. У випадку з застосунком, що розроблено в рамках даного дипломного проекту, можливість гнучких налаштувань послуг, що надає користувач, є відсутньою.

Логіка оплати перетримки відрізняється, оскільки в PetHome передбачається створення оголошень власниками з зазначенням грошової винагороди за перетримку, а не вибір вартості перетримки виконавцем. На рисунку 1.3 представлено сторінку з відгуками.

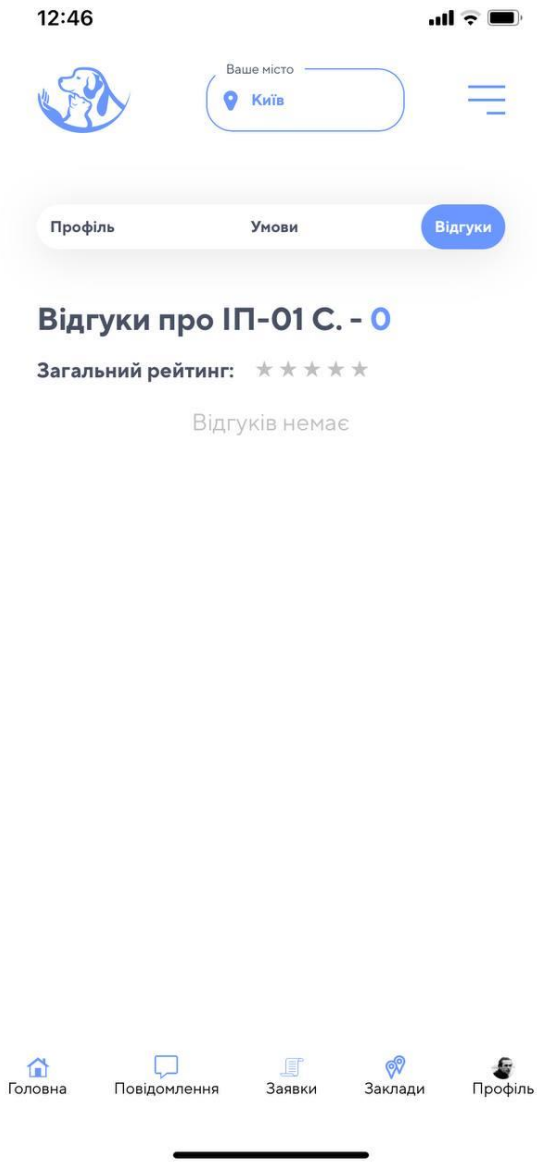


Рисунок 1.3 – Відгуки про користувача в застосунку PetLive [5]

Перевагою також є відгуки про користувача, оскільки це впливає на вибір виконавців замовниками та спонукає виконавців якісніше надавати послуги. На рисунку 1.4 представлено головну сторінку застосунку PetLive.

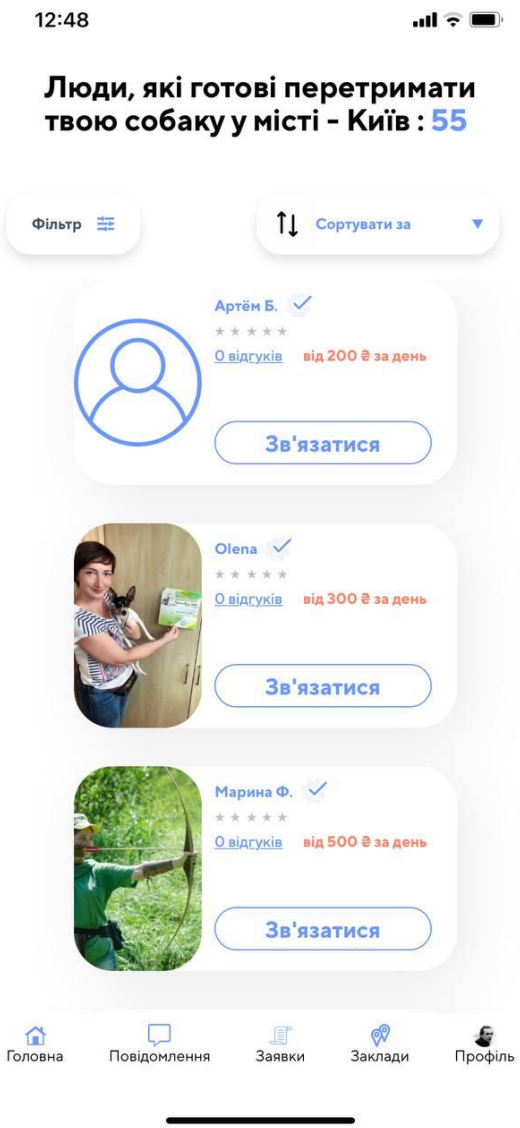


Рисунок 1.4 – Головна сторінка застосунку PetLive [5]

Основний акцент в застосунку зроблено саме на виконавців. Тобто вони налаштовують свої профілі, а замовники обирають з-поміж них, бронюють їх на певні дати та комунікують з ними.

Щодо реалізованого застосунку PetHome, замовники створюють оголошення, вказуючи грошову винагороду, обираючи конкретні дати, на котрі потрібно перетримати тварину, обирають свою локацію з точністю до номеру будинку, завантажують фото тваринки (рисунок 1.5).

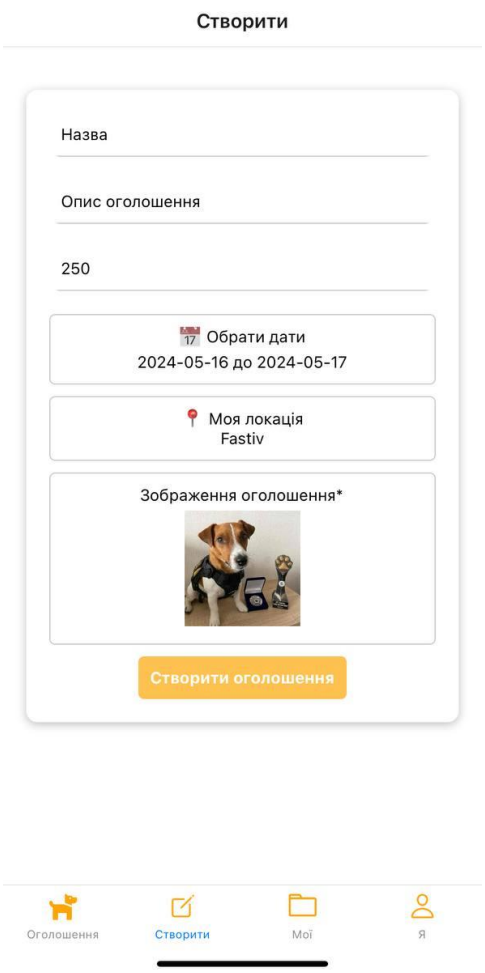


Рисунок 1.5 – Створення оголошення в застосунку PetHome

Одразу після створення нового оголошення, система автоматично підбирає можливих виконавців, враховуючи їх вільні дати та локацію. Вони мають бути не більше, ніж за 30 км від замовника. Генеруються заявки на виконання оголошення для можливих виконавців (рисунок 1.6). Вони отримують сповіщення про це та мають підтвердити або відхилити заявку. Якщо вони підтверджують заявку – власник оголошення отримує сповіщення про це та обирає виконавця з-поміж тих, хто відгукнулись на це оголошення власноруч, або підтвердили заявку, згенеровану системою.

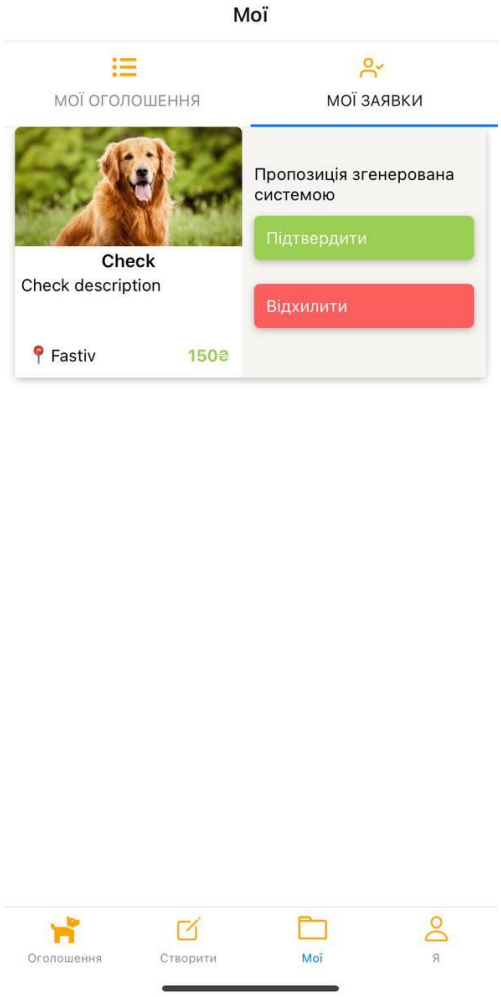


Рисунок 1.6 – Згенерована заявка в застосунку PetHome

Основною сторінкою є перелік оголошень з можливістю застосувань фільтрів, що допоможуть виконавцям швидше підібрати оголошення, на яке слід відгукнутись. Основну увагу слід звернути на фільтр «У Ваші вільні дати». Завдяки цьому фільтру, виконавцю буде представлено лише ті оголошення, дати яких не пересікаються з датами, коли виконавець зайнятий. (рисунок 1.7).

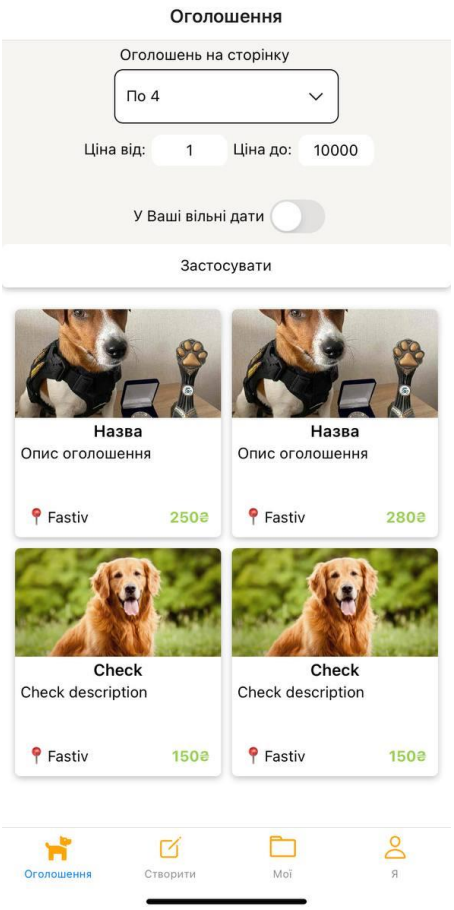


Рисунок 1.7 – Сторінка оголошень в застосунку PetHome

Користувач може додавати або прибирати дати, у які він зайнятий на сторінці свого профілю (рисунок 1.8). Якщо замовник обирає користувача як виконавця певного замовлення, певні дати буде автоматично відмічено як «забронювані».

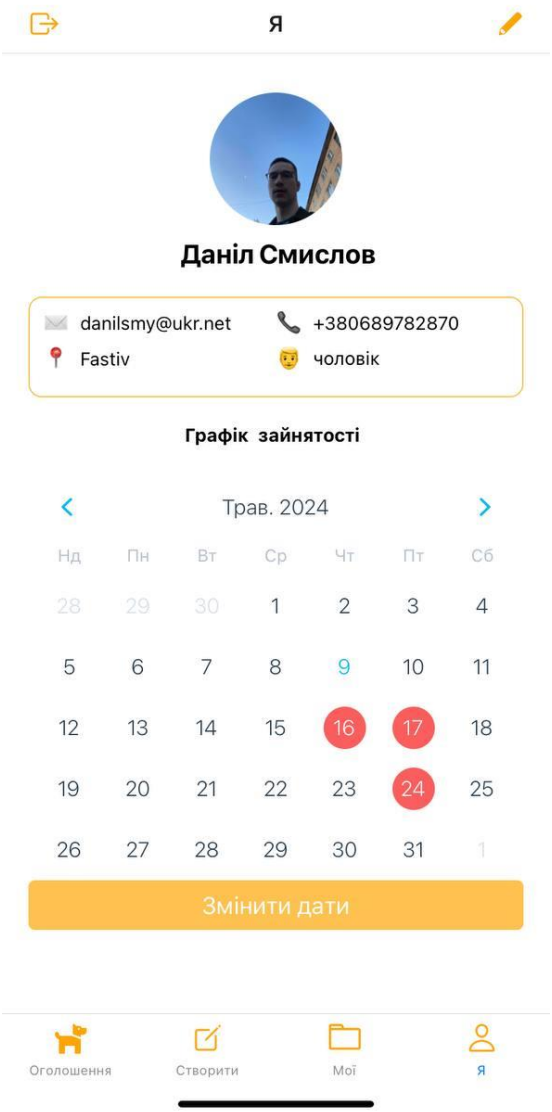


Рисунок 1.8 – Сторінка профілю користувача в застосунку PetHome

Другим мобільним застосунком у App Store є hipaw. У ньому також акцент зроблено на виконавцях. Замовник має шукати та обирати виконавця (рисунок 1.9). Менеджменту дат не реалізовано. Тобто виконавець не може налаштувати дати, у які він зайнятий та не може доглядати за тваринами.

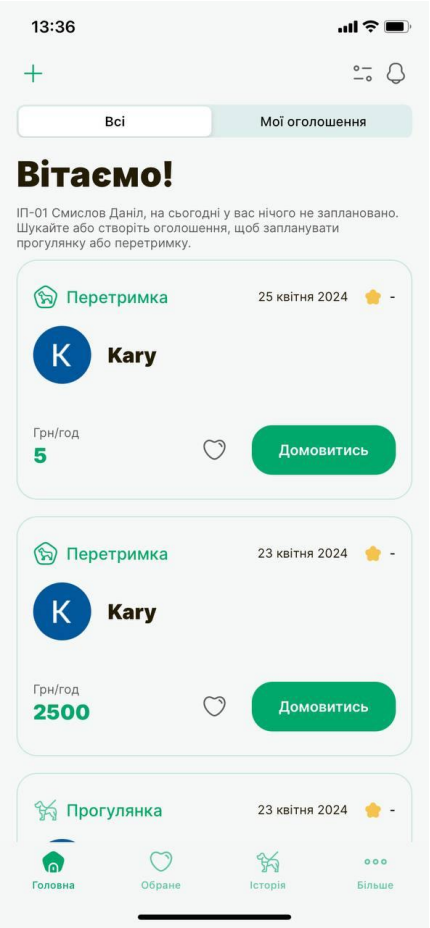


Рисунок 1.9 – Головна сторінка в застосунку hipaw [4]

Оголошення створюють саме виконавці про надання своїх послуг. Про час можна вказати лише текстом в описі (рисунок 1.10).

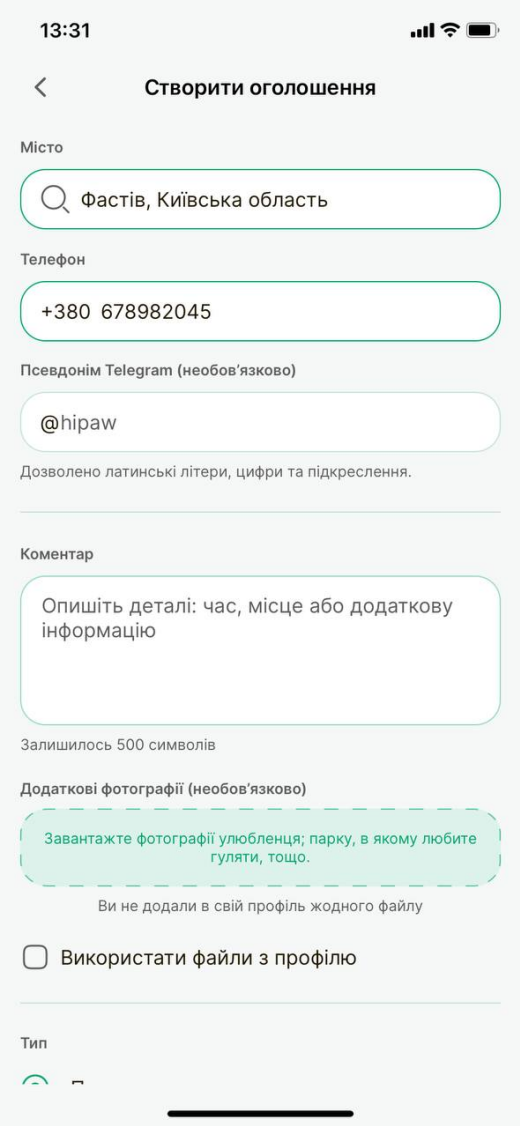


Рисунок 1.10 – Сторінка створення оголошення виконавцем в застосунку hipaw [4]

Система не генерує пропозицій. Робота з локацією користувачів також реалізовано недостатньо гнучко. Немає можливості обирати місцеположення з деталізацією вулиці або номеру будинку. Позитивним є те, що можна залишати відгуки на виконавців та авторизуватись через Google-акаунт.

Для порівняння проєкту з аналогом можна скористатись таблицею 1.1.

Таблиця 1.1 – Порівняння з аналогом

Функціонал	Дипломний проєкт [PetHome]	PetLive	Hipaw
Відгуки на користувачів	-	+	+
Онлайн-чат з користувачем	-	+	-
Сповіщення користувача на всіх етапах пошуку	+	-	-
Менеджмент зайнятих дат користувача	+	+	-
Детальне налаштування послуг, що надає виконавець	-	+	-
Автоматичний підбір можливих виконавців при створенні оголошення	+	-	-

1.2.2 Аналіз відомих алгоритмічних та технічних рішень

Дане програмне забезпечення не включає в себе складної алгоритмічної складової. Усі алгоритми та формули, що було використано, є загально відомими.

Однією з основних складових розробки програмного забезпечення є архітектура. Слід детально розглянути відомі підходи до розробки мобільних

застосунків. Під час вибору з-поміж них слід звертати увагу на вимоги до їх швидкодії, безпеки, можливості масштабування. Також важливим фактором є можливість крос-платформеності.

Для початку слід визначитись із загальною моделлю взаємодії. Вибір потрібно зробити між архітектурними шаблонами Peer-to-peer та Client-Server. Peer-to-peer архітектура складається з рівноправних пов’язаних пристроїв, кожен з яких є, водночас, сервером. Client-Server підхід передбачає централізований сервер, що обслуговується та підтримується постачальником послуг [6]. Клієнти надсилають запити на сервер та отримують відповіді.

Виходячи з особливості реалізації, Peer-to-peer архітектура зазвичай використовується для більш вузькоспеціалізованих задач. До прикладу, офлайн-чатів, розподілених обчислень, розповсюдження файлів. Основними перевагами є відсутність необхідності постачальникам послуг інвестувати в сервери, стійкість до помилок, оскільки у випадку відмови одного сервера, у ролі сервера може виступати інший користувач. Недоліками є відсутність централізованого контролю, складна схема взаємодії, менший рівень безпеки.

У випадку розробки в рамках даного дипломного проекту, краще підходить саме модель Client-Server взаємодії. Дана модель дозволяє забезпечити централізовану обробку запитів, захистити користувачів від несанкціонованого доступу до їх даних. Також надає можливість масштабування застосунку.

Щодо клієнтської частини застосунку, потрібно обрати один з підходів розробки з-поміж: native-застосунки, hybrid-застосунки, PWA-застосунки, cross-platform-застосунки [7].

Відносно новими є PWA-застосунки, тобто Progressive Web App[8]. Це застосунки, що створені за допомогою WEB-технологій та є гібридом звичайного WEB-сайту, доступ до якого здійснюється через браузер. Мобільний браузер пропонує встановити окрему іконку на робочий стіл при відвідуванні такого застосунку в браузері. Ззовні під час роботи такі застосунки схожі на нативні додатки. Завдяки цьому вони імітують досвід

роботи з native-застосунками. Даний підхід має значний перелік переваг. Користувачу не потрібно встановлювати застунок безпосередньо на свій пристрій, оскільки він виконується в браузері. Також ціна розробки є нижчою, оскільки дане рішення є крос-платформеним і працюватиме на всіх пристроях, на яких працюватиме браузер. Серед недоліків можна виділити відсутність доступу до певних можливостей пристроїв, на яких запускається PWA-застосунок. На iOS працює лише в браузері Safari. Менші можливості впровадження безпеки.

Native-застосунки мають найбільше переваг з точки зору швидкодії, функціоналу без підключення до мережі, якості взаємодії користувача та доступу до hardware-частини пристроїв, на яких вони запускаються [9]. Native-застосунок розробляється спеціально для певної платформи. Завдяки цьому він використовує її можливості найбільш оптимально. Проте серед недоліків можна визначити високу вартість розробки, оскільки під кожну платформу потрібно мати окрему команду розробників. Також недоліком для користувача є необхідність завантажувати застосунок на свій пристрій. І складнішим є поширення застосунку, у порівнянні з PWA-застосунком, який знаходиться за url-посиланням.

Hybrid-застосунки використовують WEB-технології такі, як HTML, CSS, Javascript, а потім «загортають» їх в native-оболонку [9]. Розробка таких застосунків є досить швидкою, дешевшою за розробку native-застосунків, проте гіршу швидкодію, безпеку. Також відсутня можливітьс роботи без підключення до мережі.

Cross-platform-застосунки побудовані за допомогою однієї спільної кодової бази та можуть запускатись на багатьох платформах. Одним з прикладів фреймворків, що забезпечують cross-platform розробку є React Native. Даний інструмент має загальний набір складових та компонентів, що використовуються при розробці та компілюються в native-елементи кожної платформи [10]. Тобто в результаті застосунок, що запускається на різних платформах, використовує native-складові кожної платформи. Проте

порівняно з native-застосунками, застосунки, розроблені за допомогою RN, мають трішки гіршу швидкодію.

Для реалізації клієнтської частини програмного забезпечення в рамках даного дипломного проекту було обрано cross-platform підхід, оскільки він є балансом між швидкодією native-застосунків та швидкістю розробки.

У рамках серверної частини застосунку потрібно обрати з-поміж підходів: Monolith, Serverless, Microservices [11].

Serverless розробка займає менше часу, для різних частин функціоналу використовувати різні мови програмування, що найкраще задовольняють вимогам. Проте даний підхід більше підходить у випадку, якщо на сервері має обчислюватись лише певний набір функцій. Також за умови реалізації даного підходу розробник не має повного контролю над серверною частиною застосунку, можуть бути труднощі з відлагодженням та трасування помилок.

Слід обирати між Monolith та Microservices. Якщо загально дати визначення то у випадку Monolith весь застосунок побудовано як один суцільний елемент, у випадку ж другого підходу застосунок розділяється на менші складові, що не залежні між собою. Кожна з них виконує конкретний функціонал та взаємодіє з іншими.

Основною перевагою Microservices архітектури є те, що застосунки з такою архітектурою значно простіше підтримувати та масштабувати, адже вони не мають такої сильної зв’язності, як у випадку Monolith.

Але, все ж, є й значний мінус – складність розгортання проекту з такою архітектурою, адже в такому випадку програмне забезпечення складається з багатьо складових. Також складніше тестувати таку систему. Цей архітектурний шаблон слід використовувати на великих проектах, над якими працює велика кількість розробників та є взаємодія між цілими різними складовими, які можна відділити. До прикладу, сервіс нотифікації, сервіс авторизації та сервіс взаємодії з локацією.

Підсумовуючи, оптимальним рішенням в рамках даного дипломного проекту є Monolith. Через порівняно невелике навантаження в рамках даної

предметної області, підхід Microservices не буде виправданим. У майбутньому за умови використання програмного забезпечення великою кількістю користувачів та, відповідно, збільшення навантаження на сервер, Monolith може бути розділено на кілька основних сервісів, завдяки чому буде забезпечено більшу стійкість до відмов та можливість більш гнучкого масштабування.

Загалом Monolith підхід добре задовольняє потреби у рамках даного проекту. Достатній рівень підтримки, відносна простота розгортання та тестування.

Також потрібно визначитись з архітектурним шаблоном всередині Monolith. Найбільш поширені підходи: Clean та 3-Layered, що є підвидом N-layered архітектури [12].

Clean є більш сучасною архітектурою, приблизно на 25 років молодшою за просте розбиття на 3 шари та вирішує проблему сильної залежності між шарами. Уся архітектура будується на ідеї ізоляції внутрішніх її слоїв від зовнішніх (рисунок 1.11) [13].

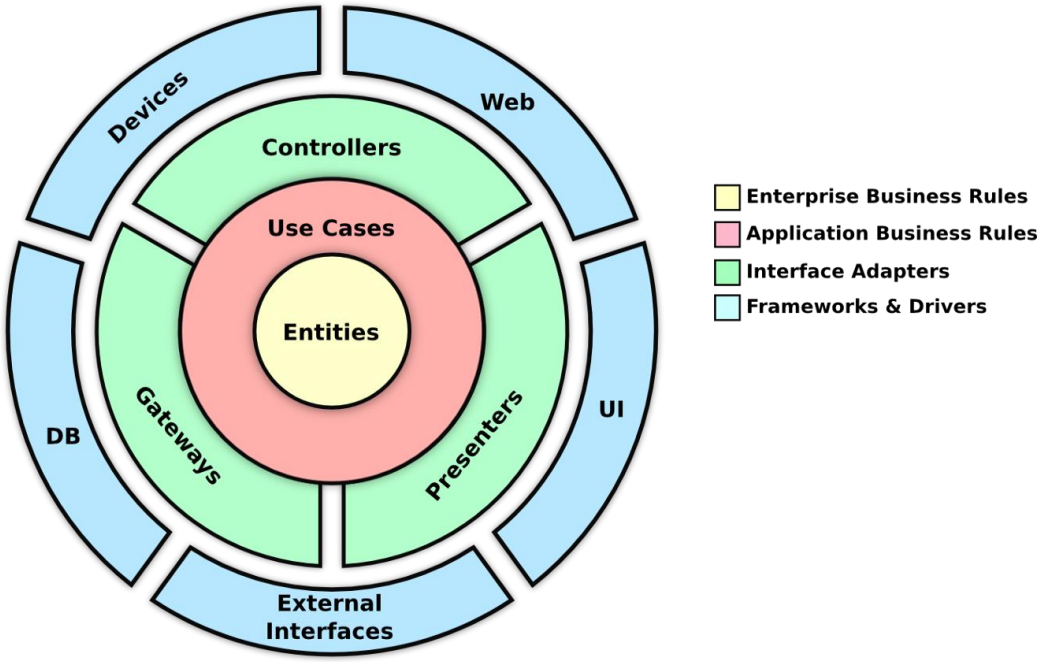


Рисунок 1.11 – Схематичне зображення Clean Architecture [13]

Завдяки цьому досягається слабка зв’язність між шарами, зменшена кількість повторюваного коду, легкість масштабування та модифікації ПЗ. Але моделювання та реалізація такої архітектури вимагає більше ресурсів та є не завжди виправданою, якщо не використовуються всі її плюси.

Щодо 3-Layered архітектури. Хоча й даний підхід не є новим, але все ж багато проектів й сьогодні існують з такою архітектурою, нові розробки й досі обирають цю архітектуру. Основна її ідея в розбитті на шари, що відповідають за конкретні свої задачі та є слабко пов’язаними між собою, логіка інкапсульована всередині них (рисунок 1.12) [14].

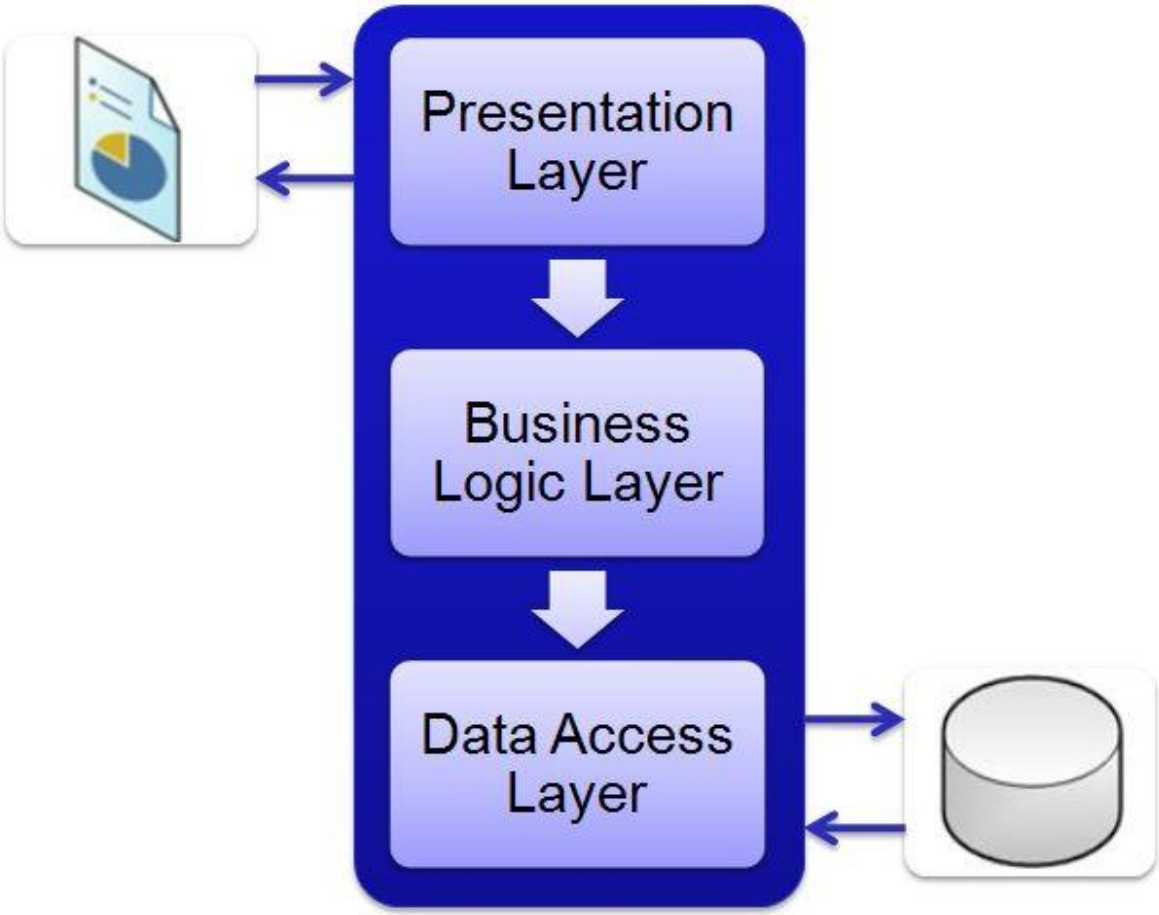


Рисунок 1.12 – Схематичне зображення 3-Layered Architecture [14]

Використання шарів є виправданим у більшості випадків, крім, можливо, демонстраційних та нескладних проектів з кількома функціями, для яких шари лише б додали складності сприйняття. Також мінусом є невеликий негативний вплив на швидкодію, оскільки після надходження запиту користувача, необхідно виконати код кількох шарів та ініціалізувати більшу кількість об’єктів.

Вибір 3-Layered архітектури забезпечує можливість зміни окремих складових програмного забезпечення без безпосереднього впливу на його інші частини. До прикладу, замінити частину шару бізнес логіки. При цьому логіка Presentation Layer та логіка роботи з даними в сховищі залишиться тією ж самою.

Більш детально архітектуру реалізованого програмного забезпечення розглянуто в одному з наступних розділів.

1.3 Опис бізнес-процесів

Для опису бізнес процесу програмного забезпечення використовується BPMN модель. Завдяки цій моделі можна представити бізнес процеси у вигляді діаграм, що є зрозумілими як розробнику, так і звичайному користувачу.

Побудовано BPMN діаграму процесу створення нового оголошення користувачем (рисунок 1.13).

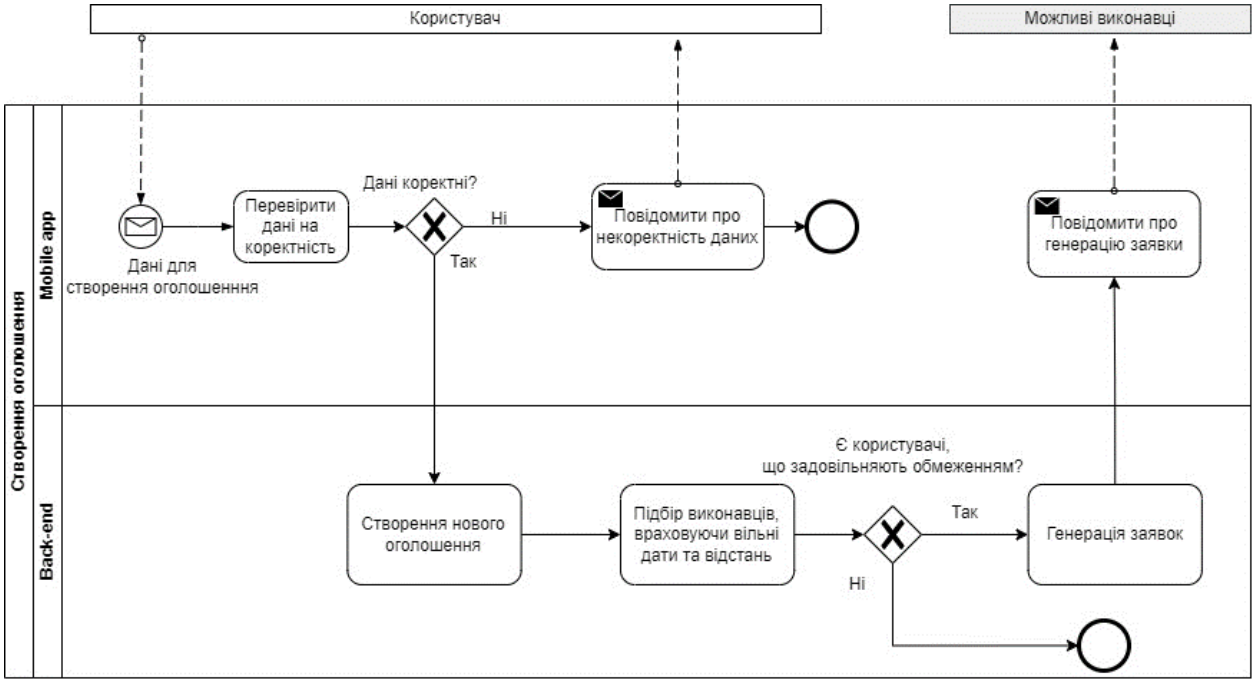


Рисунок 1.13 – BPMN модель створення оголошення

Опис послідовності створення оголошення:

– на сторінці створення оголошення користувач вводить дані оголошення та натискає кнопку створення;

– якщо введені дані не є коректними, то поля підсвічуються червоним, очікується введення оновлених даних та повторна відправка даних, щоб процес почався знову, у іншому випадку з клієнтської частини на серверну частину застосунку передаються ці дані та створюється нове оголошення;

– після цього здійснюється підбір можливих виконавців, якщо немає користувачів, що задовольняють критеріям підбору, то закінчується процес, якщо є то генеруються заявки;

– згенеровані заявки надсилаються можливим виконавцям, вони отримують про це сповіщення.

Також створено BPMN модель підтвердження власником оголошення однієї з поданих заявок (рисунок 1.14).

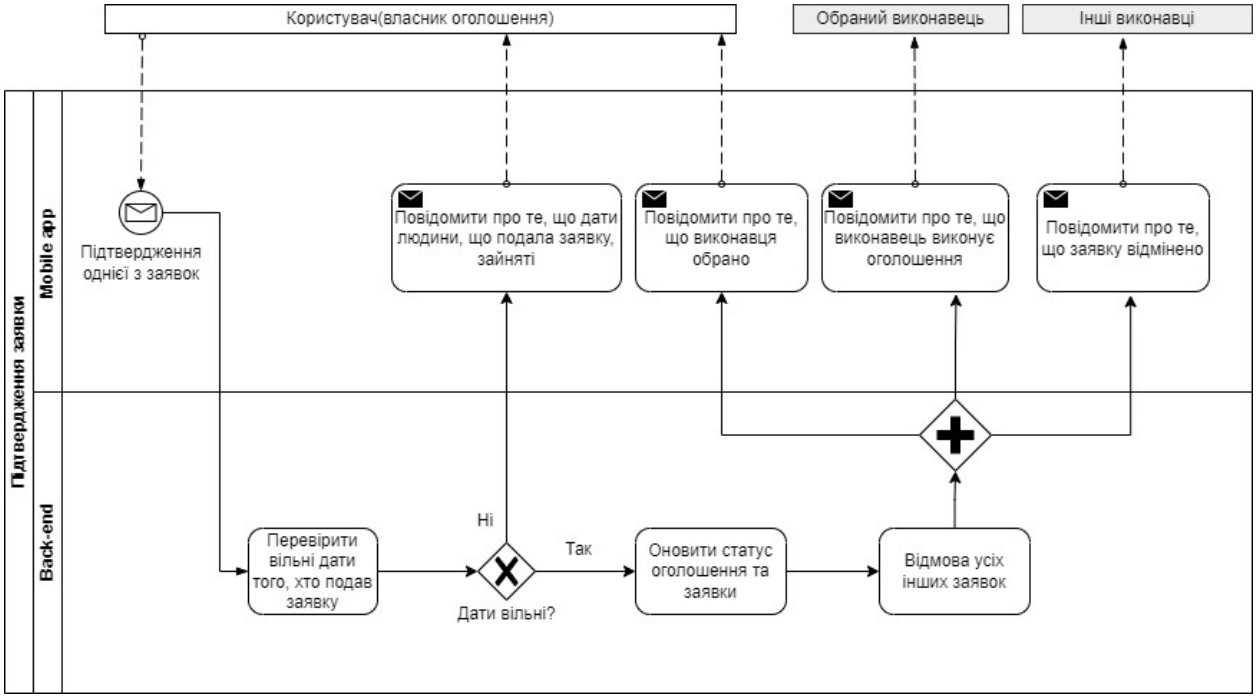


Рисунок 1.14 – BPMN модель підтвердження заявки

Опис послідовності підтвердження однієї з заявок власником оголошення:

– на сторінці створеного власником оголошення власник обирає одного з виконавців, що подали заявку та підтверджує цю заявку;

– далі back-end частина перевіряє чи вільний це виконавець у дати, зазначені в оголошенні, якщо ні – власник отримує повідомлення про те, що виконавець вже не вільний в дані дати, якщо дати вільні то оновлюємо статус оголошення та заявки, що було прийнято;

– автоматичне встановлення статусу інших заявок як «відхилено»;

– далі відбується сповіщення власника оголошення про те, що він обрав виконавця, також сповіщується обраний виконавець про те, що його було обрано до виконання, усі інші виконавці, що подали заявку, отримують сповіщення про те, що їх заявки було відхилено.

Створено BPMN модель подання користувачем заявки на виконання оголошення (рисунок 1.15).

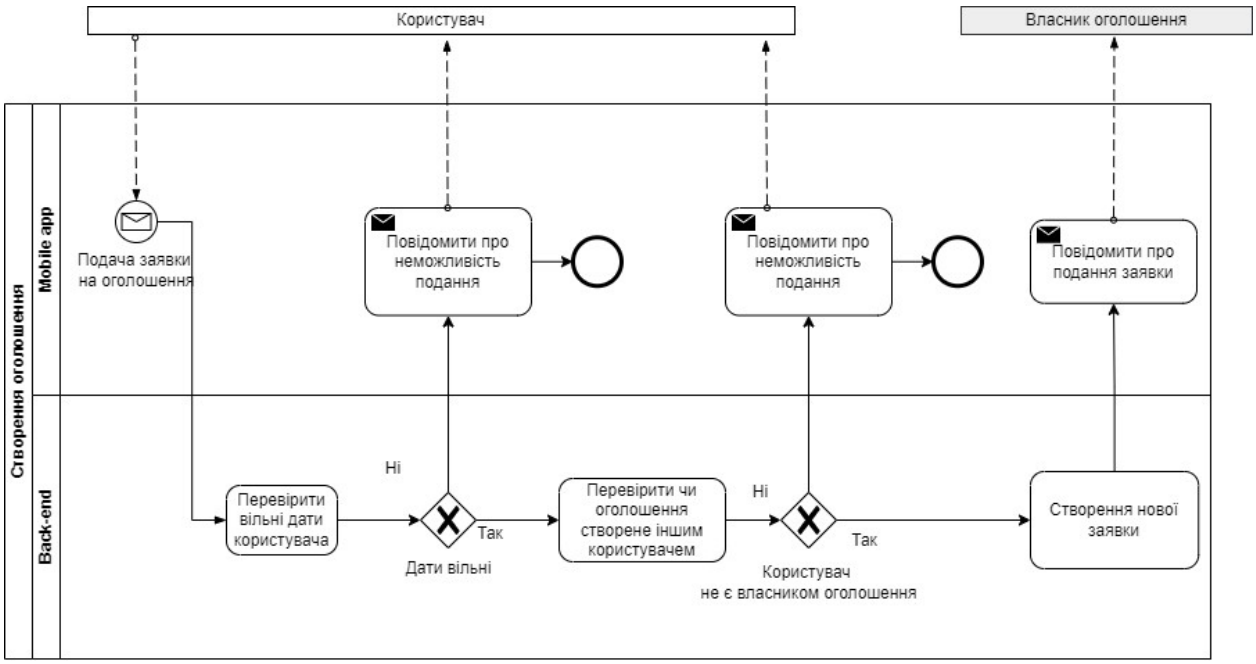


Рисунок 1.15 – BPMN модель подання заявки

Опис послідовності подання заявки на виконання оголошення:

– на сторінці конкретного оголошення користувач (можливий виконавець) натискає кнопку надсилання заявки на виконання;

– серверна частина застосунку оброблює запит, перевіряє вільні дати користувача, що подає заявку. Якщо дати виконання оголошення

збігаються з датами, в які користувач зайнятий, система надсилає відповідне повідомлення користувачу;

– якщо в дати, що вказані в оголошенні, користувач вільний, перевіряється чи це оголошення не створено самим користувачем, що надсилає заявку. Якщо оголошення створено тим самим користувачем, система надсилає відповідне повідомлення про неможливість подання заявки;

– якщо оголошення створено іншим користувачем – створюється нова заявка, власнику оголошення надсилається сповіщення про подання нової заявки на виконання його оголошення.

1.4 Постановка задачі

У рамках даного дипломного проекту потрібно реалізувати мобільний застосунок, основною функцією якого є підбір виконавців, що можуть перетримати тварину, згідно з умовами, що зазначені в оголошенні.

Метою розробки є полегшення процесу пошуку виконавців та замовників з формуванням конкретних релевантних пропозицій, враховуючи задані критерії, та часткова автоматизація процесу підбору можливих виконавців.

Для досягнення мети потрібно реалізувати наступні задачі:

– часткова автоматизація процесу пошуку виконавців, шляхом генерації пропозицій можливим виконавцям, що задовольняють обмеженням;

– фільтрація оголошень, що при підборі може враховувати графік завантаженості виконавця;

– сповіщення в режимі реального часу;

– функціонал адміністрування для модерації профілів та оголошень;

– робота з графіком завантаженості.

Висновки до розділу

У даному розділі на початку було більш детально досліджено предметну область та доведено, що розробка мобільного застосунку PetHome є

актуальною на сьогодні. Було досліджено існуючі рішення, описано можливі шляхи покращення.

Було проаналізовано 2 існуючі реалізації мобільних застосунків в рамках предметної області. Визначено їх сильні сторони, виділено функціонал, який відсутній, проте може позитивно вплинути на швидкість та спростити процес підбору виконавців.

Детально описано вибір архітектурних рішень та підходів щодо реалізації мобільного застосунку. Було обрано загальну модель взаємодії Client-Server та деталізовано архітектурні підходи в рамках клієнтської та серверної частини застосунку.

Визначено основні бізнес-процеси та побудовано BPMN діаграми, що описують ці процеси.

Сформовано постановку задачі, що визначає завдяки якому функціоналу досягається мета розробки.

2 РОЗРОБЛЕННЯ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Варіанти використання програмного забезпечення

Головною функцією програмного забезпечення є частково автоматизований релевантний підбір виконавців, що можуть посидіти з тваринами замовників. При цьому має враховуватись місцеположення виконавців та дати, в які вони є вільними та можуть доглядати за тваринами. Більше функцій можна побачити на рисунку 2.1.

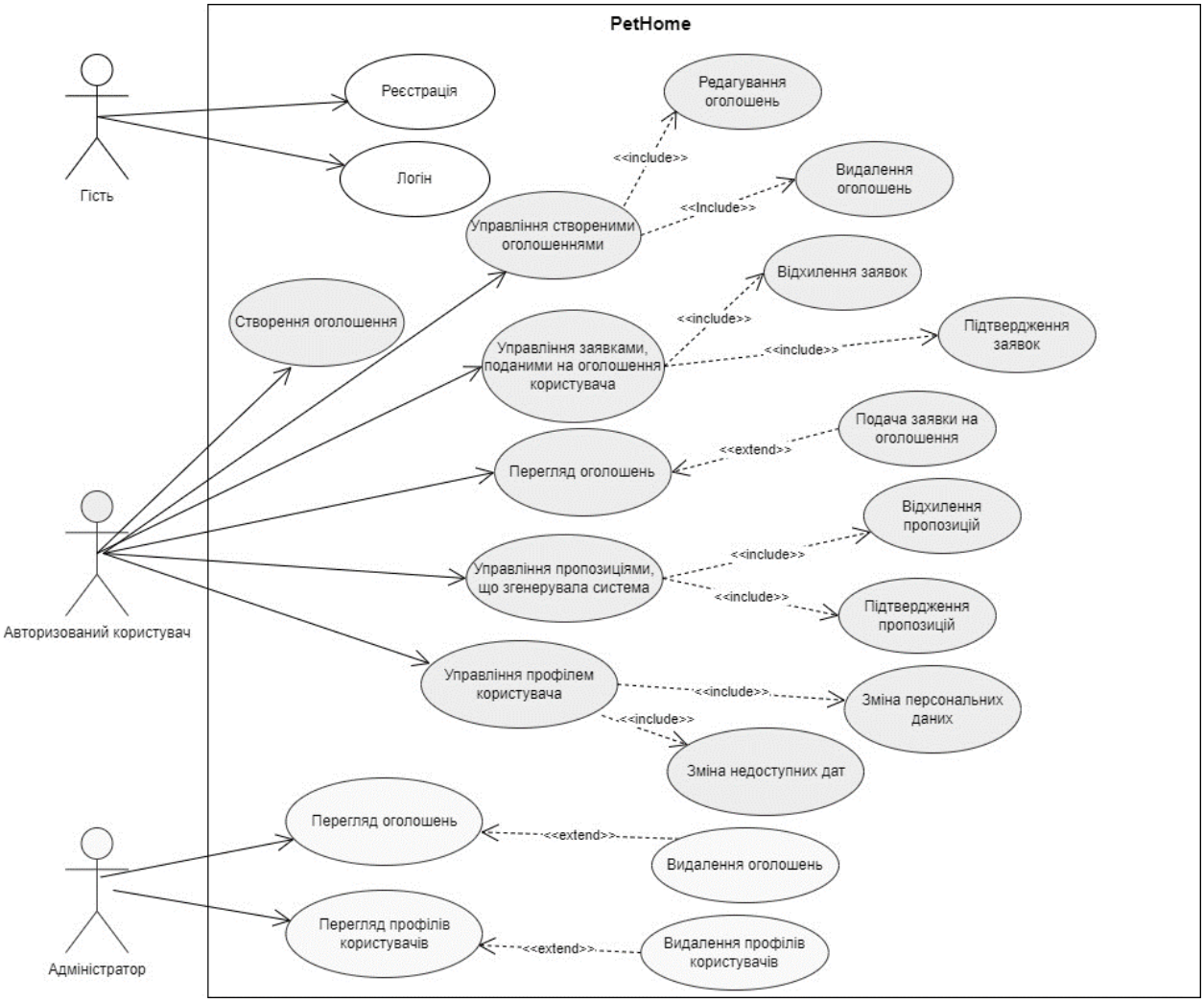


Рисунок 2.1 – Діаграма варіантів використання

В таблицях 2.1 - 2.18 наведені варіанти використання програмного забезпечення.

Таблиця 2.1 - Варіант використання UC-1

Use case name	Реєстрація нового користувача
Use case ID	UC-01
Goals	Реєстрація нового користувача в системі
Actors	Гість (незареєстрований користувач)
Trigger	Користувач бажає зареєструватися
Pre-conditions	-
Flow of Events	Користувач переходить на сторінку реєстрації. В поля для реєстрації вводяться відповідні дані: прізвище та ім’я, стать, обирається фото, адреса електронної пошти користувача, номер телефону, його логін, пароль в системі, та його повтор для підтвердження, а також користувач може ввести свій населений пункт та обрати його з випадаючого списку, або ж надати доступ до свого місцеположення та, таким чином, автоматично заповнити адресу. Після заповнення даних користувач натискає кнопку реєстрації. Після цього користувач перенаправляється на сторінку входу.
Extension	У випадку введення не коректних даних поля, що не задовольняють умові, підсвічуються червоним кольором та користувач не має змоги зареєструватись, поки не виправить значення, введені в ці поля.
Post-Condition	Створення сторінки користувача, перехід на сторінку входу.

Таблиця 2.2 - Варіант використання UC-2

Use case name	Вхід користувача в профіль
Use case ID	UC-02
Goals	Вхід користувача у свій раніше зареєстрований профіль
Actors	Зареєстрований користувач
Trigger	Користувач бажає увійти в свій профіль
Pre-conditions	Користувач має зареєстрований акаунт в застосунку

Продовження таблиці 2.2

Flow of Events	Користувач переходить на сторінку логіну. У відповідні поля він вводить свій логін та пароль. Натискає кнопку логіну. Після цього користувач перенаправляється на сторінку з оголошеннями.
Extension	У випадку введення не коректних даних користувач отримає повідомлення про те, що дані для входу не є валідними.
Post-Condition	Перехід на сторінку з оголошеннями, користувач стає авторизованим.

Таблиця 2.3 - Варіант використання UC-3

Use case name	Створення оголошення
Use case ID	UC-03
Goals	Створення нового оголошення про пошук виконавця, щоб він доглянув за твариною.
Actors	Авторизований користувач
Trigger	Користувач бажає створити оголошення
Pre-conditions	Користувач має бути авторизованим в застосунку
Flow of Events	Користувач переходить на сторінку створення оголошення. У відповідні поля він вводить назву, опис, вартість, обирає фото тварини та може автоматично обрати місцеположення, надавши доступ до геолокації, або ввести населений пункт вручну та обрати локацію з випадаючого списку. Користувач натискає кнопку створення оголошення. Після цього користувач переходить на сторінку з оголошеннями. Система автоматично підбирає можливих виконавців, генерує нові заявки та надсилає сповіщення можливим виконавцям.
Extension	У випадку введення не коректних даних, поля з невалідними даними буде підсвічено червоним кольором та користувач не зможе створити оголошення.
Post-Condition	Створилось нове оголошення. За умови, що є користувачі, що задовольняють умовам виконання оголошення, згенеровано нові заявки та надіслано сповіщення можливим виконавцям.

Таблиця 2.4 - Варіант використання UC-4

Use case name	Подача заявки на виконання оголошення
Use case ID	UC-04
Goals	Подача нової заявки на оголошення, що обрав користувач
Actors	Авторизований користувач
Trigger	Користувач бажає надіслати заявку на оголошення
Pre-conditions	Користувач має бути авторизованим в застосунку. Іншим користувачем має бути створено оголошення.
Flow of Events	Користувач переходить на сторінку конкретного оголошення. Ознайомившись з умовами виконання та датами оголошення, користувач натискає кнопку подачі заявки. Після цього створюється нова заявка та власник оголошення отримує сповіщення про нову заявку на його оголошення.
Extension	У випадку наявності у користувача заброньованих дат, що заважають виконанню оголошення, користувач отримує повідомлення про прохання видалити заброньовані дати, через які відбулась колізія. У випадку якщо це оголошення створив сам користувач, кнопка подачі заявки буде неактивною.
Post-Condition	Нова заявка створена. Власник оголошення отримав сповіщення про нову заявку на його оголошення.

Таблиця 2.5 - Варіант використання UC-5

Use case name	Підтвердження виконавцем згенерованої системою пропозиції
Use case ID	UC-05
Goals	Підтвердити пропозицію, яку згенерувала система
Actors	Авторизований користувач
Trigger	Користувач бажає підтвердити згенеровану пропозицію
Pre-conditions	Користувач має бути авторизованим в застосунку. Система мала згенерувати пропозицію на виконання оголошення, що може бути цікавим цьому користувачу.

Продовження таблиці 2.5

Flow of Events	Якщо користувач був в мережі у момент генерації заявки, то він отримає сповіщення про це. У іншому випадку користувач має перейти на сторінку з його заявками. Далі потрібно обрати конкретну заявку, можна переглянути оголошення. Користувач натискає кнопку підтвердження заявки. Власнику оголошення приходить сповіщення про подачу заявки користувачем.
Extension	У випадку наявності у користувача заброньованих дат, що заважають виконанню оголошення, користувач отримує повідомлення про прохання видалити заброньовані дати, через які відбулась колізія.
Post-Condition	Нова заявка створена. Власник оголошення отримав сповіщення про нову заявку на його оголошення.

Таблиця 2.6 - Варіант використання UC-6

Use case name	Відхилення згенерованої системою пропозиції
Use case ID	UC-06
Goals	Відхилити пропозицію, яку згенерувала система
Actors	Авторизований користувач
Trigger	Користувач бажає відхилити згенеровану пропозицію
Pre-conditions	Користувач має бути авторизованим в застосунку. Система мала згенерувати пропозицію на виконання оголошення, що може бути цікавим цьому користувачу.
Flow of Events	Якщо користувач був в мережі у момент генерації заявки, то він отримає сповіщення про це. У іншому випадку користувач має перейти на сторінку з його заявками. Далі потрібно обрати конкретну заявку, можна переглянути оголошення. Користувач натискає кнопку відхилення заявки. Власник оголошення не отримує сповіщення про відхилення користувачем заявки, адже ця заявка була згенерована автоматично та власник про неї навіть не знав.
Extension	-
Post-Condition	Заявка видалена.

Таблиця 2.7 - Варіант використання UC-7

Use case name	Підтвердження заявки власником оголошення
Use case ID	UC-07
Goals	Підтвердити заявку, яку подав виконавець
Actors	Авторизований користувач
Trigger	Користувач бажає прийняти заявку, що було подано на його оголошення
Pre-conditions	Користувач має бути авторизованим в застосунку. Виконавець подав заявку на оголошення, що створив користувач.
Flow of Events	Якщо користувач в мережі, то він отримає сповіщення про це. У іншому випадку користувач має перейти на сторінку з його оголошеннями. Далі на сторінку конкретного оголошення та обрати одного з виконавців, що подали заявки. Потрібно натиснути на кнопку підтвердження заявки. Обраному виконавцю прийде сповіщення про те, що він виконує це оголошення, також автоматично у нього з’являться нові заброньовані дати, а усім іншим виконавцям, що подали заявку, повідомлення про те, що їх заявки було відхилено.
Extension	У випадку якщо у виконавця, що подав заяку, з’явились нові незаброньовані дати, що пересікаються з датами оголошення, тоді власник оголошення отримає повідомлення про те, що користувач вже зайнятий у ці дати.
Post-Condition	Заявка прийнята. Виконавці, що подавали заявки, отримали відповідні сповіщення.

Таблиця 2.8 - Варіант використання UC-8

Use case name	Відхилення заявки власником оголошення
Use case ID	UC-08
Goals	Відхилити заявку, яку подав виконавець
Actors	Авторизований користувач
Trigger	Користувач бажає відхилити заявку, що було подано на його оголошення

Продовження таблиці 2.8

Pre-conditions	Користувач має бути авторизованим в застосунку. Виконавець подав заявку на оголошення, що створив користувач.
Flow of Events	Якщо користувач був в мережі у момент подачі заявки, то він отримає сповіщення про це. У іншому випадку користувач має перейти на сторінку з його оголошеннями. Далі потрібно перейти на сторінку конкретного оголошення та обрати одного з виконавців, що подали заявки. Слід натиснути на кнопку відхилення заявки. Після цього цьому виконавцю прийде сповіщення про те, що його заявку було відхилено.
Extension	-
Post-Condition	Заявка відхилена.

Таблиця 2.9 - Варіант використання UC-9

Use case name	Додавання недоступних дат
Use case ID	UC-9
Goals	Додати недоступні дати в профілі користувача
Actors	Авторизований користувач
Trigger	Користувач бажає додати недоступні дати у себе в профілі
Pre-conditions	Користувач має бути авторизованим в застосунку.
Flow of Events	Користувач має перейти на сторінку свого профіля. Далі слід натиснути кнопку додавання недоступних дат та в календарі, що з’явиться, потрібно обрати дати для бронювання та натиснути на кнопку їх додавання.
Extension	-
Post-Condition	Нові недоступні дати користувача додано, вони відображаються в його профілі.

Таблиця 2.10 - Варіант використання UC-10

Use case name	Видалення недоступних дат
Use case ID	UC-10

Продовження таблиці 2.10

Goals	Видалити недоступні дати в профілі користувача
Actors	Авторизований користувач
Trigger	Користувач бажає видалити недоступні дати у себе в профілі
Pre-conditions	Користувач має бути авторизованим в застосунку.
Flow of Events	Користувач має перейти на сторінку свого профіля. Далі слід натиснути кнопку видалення недоступних дат та в календарі, що з’явиться, потрібно обрати недоступні дати для видалення та натиснути кнопку їх видалення.
Extension	-
Post-Condition	Певні недоступні дати видалено, тепер ці дати є вільними для користувача та відображаються в його профілі.

Таблиця 2.11 - Варіант використання UC-11

Use case name	Редагування профіля користувача
Use case ID	UC-11
Goals	Редагувати персональну інформацію користувача
Actors	Авторизований користувач
Trigger	Користувач бажає змінити свої дані в профілі
Pre-conditions	Користувач має бути авторизованим в застосунку.
Flow of Events	Користувач має перейти на сторінку свого профіля. Далі слід натиснути кнопку редагування профілю та у вікні, що з’явиться, слід за потреби замінити дані, що містяться в полях: прізвище, ім’я, email, номер телефону. Також можна додати нове фото та змінити місцеположення. Далі слід натиснути на кнопку редагування. Після цього вікно редагування зникне та дані буде оновлено.
Extension	За умови введення некоректних даних, відповідні поля буде підсвічено червоним кольором та до їх виправлення користувач не зможе редагувати свій профіль.
Post-Condition	Дані профіля користувача змінено.

Таблиця 2.12 - Варіант використання UC-12

Use case name	Редагування оголошення
Use case ID	UC-12
Goals	Редагувати оголошення, яке створив користувач
Actors	Авторизований користувач
Trigger	Користувач бажає змінити дані оголошення
Pre-conditions	Користувач має бути авторизованим в застосунку. Він створив оголошення про пошуку виконавця.
Flow of Events	Користувач має перейти на сторінку з своїми оголошеннями. Далі слід обрати конкретне оголошення та перейти на його сторінку, натиснути кнопку редагування оголошення та у вікні, що з’явиться, слід за потреби замінити дані, що містяться в полях: назва, опис, вартість. Також можна додати нове фото, змінити місцеположення оголошення та змінити дати оголошення. Далі слід натиснути на кнопку редагування. Після цього вікно редагування зникне та дані оголошення буде оновлено.
Extension	За умови введення некоректних даних, відповідні поля буде підсвічено червоним кольором та до їх виправлення користувач не зможе редагувати своє оголошення.
Post-Condition	Дані оголошення користувача змінено.

Таблиця 2.13 - Варіант використання UC-13

Use case name	Позначення оголошення як «виконане»
Use case ID	UC-13
Goals	Позначення оголошення як «виконане»
Actors	Власник оголошення
Trigger	Користувач бажає відмітити оголошення як «виконане» коли перетримка відбулась.
Pre-conditions	Користувач має бути авторизованим в застосунку. Він створив оголошення, підібрав виконавця, виконавець перетримав тварину.

Продовження таблиці 2.13

Flow of Events	Користувач має перейти на сторінку створеного ним оголошення, обрати кнопку позначення оголошення як «виконане».
Extension	-
Post-Condition	Оголошення позначено як «виконане». Власник може видалити оголошення.

Таблиця 2.14 - Варіант використання UC-14

Use case name	Перегляд оголошень користувача з фільтрами
Use case ID	UC-14
Goals	Переглянути певним чином відфільтровані оголошення
Actors	Авторизований користувач
Trigger	Користувач має бажання переглянути оголошення, застосувавши фільтр
Pre-conditions	Користувач має бути авторизованим в застосунку. Мають бути опубліковані оголошення.
Flow of Events	Користувач має перейти на сторінку з оголошеннями. Він може обрати діапазон вартостей оголошень. Також можна обрати кількість оголошень на сторінці та номер сторінки. Є можливість обрати налаштування «У Ваші вільні дати». У такому випадку буде виведено лише оголошення, дати яких не перетинаються з зайнятими датами користувача.
Extension	У випадку якщо це оголошення користувача – можна обрати ще й статус оголошень.
Post-Condition	Користувач бачить перелік оголошень, що задовольняють фільтрам.

Таблиця 2.15 - Варіант використання UC-15

Use case name	Вихід з профілю користувача
Use case ID	UC-15
Goals	Вийти з профілю на сторінку авторизації
Actors	Авторизований користувач
Trigger	Користувач має бажання вийти зі свого профілю

Продовження таблиці 2.15

Pre-conditions	Користувач має бути авторизованим в застосунку.
Flow of Events	Користувач має перейти на сторінку профілю та натиснути кнопку виходу з профілю.
Extension	-
Post-Condition	Користувач вийшов зі свого профілю та перейшов на сторінку логіну в застосунок.

Таблиця 2.16 - Варіант використання UC-16

Use case name	Видалення оголошення адміністратором
Use case ID	UC-16
Goals	Видалити оголошення з неприйнятним або некоректним вмістом
Actors	Адміністратор
Trigger	Адміністратор знайшов оголошення, що містить неприйнятний або некоректний вміст.
Pre-conditions	Адміністратор має бути авторизованим в застосунку
Flow of Events	Адміністратор переходить на сторінку оголошення та натискає кнопку видалення.
Extension	-
Post-Condition	Оголошення видалено

Таблиця 2.17 - Варіант використання UC-17

Use case name	Видалення профіля користувача адміністратором
Use case ID	UC-16
Goals	Видалити профіль користувача з неприйнятним або некоректним вмістом
Actors	Адміністратор
Trigger	Адміністратор знайшов профіль користувача, що містить неприйнятний або некоректний вміст.

Продовження таблиці 2.17

Pre-conditions	Адміністратор має бути авторизованим в застосунку
Flow of Events	Адміністратор переходить на сторінку профілю користувача та натискає кнопку видалення.
Extension	-
Post-Condition	Профіль користувача видалено

Таблиця 2.18 - Варіант використання UC-18

Use case name	Видалення власного профіля користувачем
Use case ID	UC-18
Goals	Видалити власний профіль
Actors	Авторизований користувач
Trigger	Користувач бажає видалити свій профіль з персональною інформацією.
Pre-conditions	Користувач має бути авторизованим в застосунку.
Flow of Events	Користувач переходить на сторінку профілю та натискає кнопку видалення профілю.
Extension	-
Post-Condition	Профіль користувача видалено. Користувача перенаправлено на сторінку реєстрації.

2.2 Аналіз системних вимог

Програмне забезпечення повинно функціонувати на сучасних пристроях з операційною системою iOS, починаючи з версії 13.4. Саме дана версія iOS вказана як мінімальна для запуску RN-застосунків в офіційному репозиторії React Native [15]. Тому й інші мінімальні рекомендовані системні вимоги в основному виходять з можливості пристрою підтримки iOS 13.4 та новіше. Крайнім телефоном, що підтримує iOS 13.4 є Iphone 6s, тому вказана його роздільна здатність екрану у якості мінімальної рекомендованої.

У застосунку відсутнє завантаження великих об’ємів інформації. Також відсутня постійна взаємодія між користувачами в режимі реального часу, тому у якості мінімальної рекомендованої швидкості підключення до мережі Інтернет вказано 5 мегабіт/с, як мінімальну швидкість, яку надають всі постачальники телекомунікаційних послуг.

Мінімальна рекомендована конфігурація технічних засобів:

– версія операційної системи iOS: 13.4;

– підключення до мережі Інтернет зі швидкістю від 5 мегабіт/с;

– об’єм вільної пам’яті: 50 Мб;

– об’єм оперативної пам’яті: можливість пристрою підтримки iOS версії 13.4;

– характеристики процесора: можливість пристрою підтримки iOS версії 13.4;

– роздільна здатність екрану: 750x1334 пікселів.

Застосунок підтримує лише вертикальне розташування пристрою. Співвідношення сторін 9:16, або 9:19.5.

2.3 Розроблення функціональних вимог

Програмне забезпечення розділене на модулі. Кожен модуль має свій набір функцій. На рисунках 2.2 та 2.3 наведено загальну модель вимог, а в таблиці 2.19 наведений опис функціональних вимог до програмного забезпечення. Матрицю трасування вимог можна побачити на рисунку 2.4.

ID	Full decription	Code	Priority	Risk
1.	**Авторизація користувача**	REQ_1	Високий	Високий
1.1	Валідація введених даних			
1.1.1	Перевірка мінімальної довжини логіну та паролю			
1.2	Перевірка існуючого профілю за введеними даними			
2.	**Перегляд списку оголошень**	REQ_2	Високий	Низький
2.1	Застосування фільтрів			
2.1.1	Вибір діапазону сум виплат			
2.1.2	Вибір фільтру "У мої вільні дати"			
2.2	Вибір кількості оголошень на сторінці та номеру сторінки			
3.	**Створення оголошень**	REQ_3	Високий	Високий
3.1	Валідація введених даних			
3.1.1	Валідація суми винагороди (>0 && <=100000)			
3.1.2	Валідація дат часу виконання(startTime <= endTime)			
3.1.3	Валідація локації(має бути існуючою)			
3.1.4	Валідація довжини назви та опису			
3.1.5	Валідація формату, розміру, співвідношення сторін прикріпленого зображення			
4.	**Автоматична генерація та відправка пропозицій виконавцям**	REQ_4	Середній	Середній
4.1	Сповіщення підібраних можливих виконавців про генерацію пропозиції			
5.	**Подача заявки на оголошення**	REQ_5	Середній	Середній
5.1	Врахування вільних дат користувача			
6.	**Підтвердження власником однієї з поданих заявок**	REQ_6	Середній	Середній
6.1	Сповіщення користувача, заявку якого було підтверджено			
6.2	Сповіщення користувача, заявку якого було відхилено			
7.	**Відхилення власником однієї з поданих заявок**	REQ_7	Середній	Низький
7.1	Сповіщення користувача, заявку якого було відхилено			

Рисунок 2.2 – Модель вимог в загальному вигляді

8.	**Підтвердження виконавцями згенерованих заявок**	REQ_8	Середній	Середній
8.1	Сповіщення власника про подачу заявки-			
9.	**Відхилення виконавцями згенерованих заявок**	REQ_9	Середній	Середній
10.	**Можливість замовнику відмічати оголошення як "виконане"**	REQ_10	Низький	Низький
11.	**Реєстрація нового користувача**	REQ_11	Високий	Високий
11.1	Валідація введених даних			
11.1.1	Валідація довжини ім'я та прізвища			
11.1.2	Валідація зображення			
11.1.3	Валідація email-адреси			
11.1.4	Перевірка унікальності логіну			
11.1.5	Перевірка складності паролю			
11.1.6	Валідація локації(має бути існуючою)			
12.	**Редагування профілю користувача**	REQ_12	Середній	Високий
12.1	Валідація введених даних			
13.	**Редагування оголошення**	REQ_13	Низький	Середній
13.1	Валідація введених даних			
14.	**Зміна недоступних дат в профілі**	REQ_14	Середній	Високий
15.	**Вихід з профілю користувача**	REQ_15	Низький	Середній
16.	**Видалення оголошення адміністратором**	REQ_16	Середній	Середній
17.	**Видалення профіля адміністратором**	REQ_17	Середній	Середній
18.	**Видалення профіля користувачем**	REQ_18	Низький	Високий

Рисунок 2.3 – Модель вимог в загальному вигляді

Таблиця 2.19 – Перелік функціональних вимог

Назва	Опис
REQ-1	Система повинна надавати можливість авторизації користувачеві за умови що він має акаунт та ввів коректні логін та пароль. Пріоритет є високим, оскільки для виконання практично усіх запитів потрібно бути авторизованим.

Продовження таблиці 2.19

Назва	Опис
REQ-2	Система повинна надавати можливість перегляду списку оголошень та застосування певних фільтрів до цього списку. До прикладу, вибір діапазону вартості, статусу, кількості оголошень на сторінці та номеру сторінки, умови «У мої вільні дати». Пріоритет є високим, оскільки без можливості перегляду оголошень користувачі не зможуть надсилати заявки та, відповідно, не буде можливості підбору виконавців.
REQ-3	Система повинна надавати можливість створення нових оголошень, коректно ввівши назву, опис, вартість оголошення. Назва має бути більше 5 символів, опис більше 10 символів, вартість додатньою. А також додавши файл розширення .jpg/png/jpeg, співвідношення сторін якого не має перевищувати 16:9 або 9:16, розміром не більше 5 Мб, обравши локацію, що існує та дати. Пріоритет є високим, оскільки якщо не буде можливості створення оголошень – перелік оголошень буде пустим.
REQ-4	Після створення нового оголошення система повинна автоматично генерувати та надсилати заявки виконавцям, що вільні в конкретні дати та знаходяться в радіусі 30 км. Цим виконавцям мають приходити сповіщення в режимі реального часу. Пріоритет є середнім, оскільки хоча даний функціонал і є таким, що зменшує час пошуку виконавців, без нього замовники все одно зможуть знаходити виконавців.
REQ-5	Система повинна надавати можливість користувачеві подавати заявки на виконання оголошення. Пріоритет середній, оскільки користувачі можуть скористатись контактами та домовитись про перетримку без даного функціоналу.

Продовження таблиці 2.19

Назва	Опис
REQ-6	Система повинна надавати можливість власнику оголошення обирати одну серед заявок, що було подано на його оголошення та підтверджувати її. Виконавці, що подавали заявки мають отримувати відповідні сповіщення. Той, кого було обрано, має сповіщуватись про те, що він тепер виконує це оголошення. Усі інші про те, що їх заявки відхилено. Пріоритет є середнім.
REQ-7	Система повинна надавати можливість власнику оголошення відхиляти заявки, що було подано на його оголошення. Виконавці, що подавали заявки мають отримувати сповіщення про відхилення їх заявок. Пріоритет є середнім.
REQ-8	Система повинна надавати можливість виконавцям підтверджувати заявки, що система згенерувала для них. Після цього власник оголошення має отримувати сповіщення про це. Пріоритет є середнім.
REQ-9	Система повинна надавати можливість виконавцям відхиляти заявки, що система згенерувала для них. Пріоритет є середнім.
REQ-10	Замовник має мати змогу відмітити оголошення як «виконане». Після цього його можна буде видалити. Пріоритет є низьким, оскільки процес перетримки вже відбувся на момент коли даний функціонал використовується користувачами.

Продовження таблиці 2.19

Назва	Опис
REQ-11	Система повинна надавати можливість реєстрації користувачеві. Для цього йому потрібно ввести прізвище та ім’я, стать, прикріпити фото, ввести email, номер телефону, унікальний логін, пароль, повторити пароль та обрати місцеположення. Місцеположення можна або ввести самому та обрати з випадаючого списку, або надати доступ до геолокації та натиснути на кнопку автоматичного заповнення. Пароль має містити як мінімум 8 символів, спеціальний символ, як мінімум 1 велику та маленькі літери. Номер телефону має бути відповідного українського формату. Email має задовольняти формату адреси поштової скриньки. Пріоритет є високим, оскільки без зареєстрованих профілів, користувачі не зможуть авторизуватись та користуватись функціоналом застосунку.
REQ-12	Система повинна надавати можливість користувачеві редагувати його профіль, за умови, що було введено коректні оновлені дані. Пріоритет середній.
REQ-13	Система повинна надавати можливість користувачеві редагувати оголошення, за умови, що було введено коректні оновлені дані. Пріоритет низький.
REQ-14	Система повинна надавати можливість користувачу змінювати його недоступні дані вручну у себе в профілі. Пріоритет середній.
REQ-15	Користувач має мати можливість вийти зі свого профілю. Пріоритет низький.
REQ-16	Адміністратор має мати можливість видалити оголошення користувача, що містить непристойну або некоректну інформацію. Пріоритет середній.

Продовження таблиці 2.19

Назва	Опис
REQ-17	Адміністратор має мати можливість видалити профіль користувача, що містить непристойну або некоректну інформацію. Пріоритет середній.
REQ-18	Користувач має мати можливість видалити власний профіль. Пріоритет низький.

	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	UC-16	UC-17	UC-18
REQ_1		+																
REQ_2														+				
REQ_3			+															
REQ_4			+															
REQ_5				+														
REQ_6							+											
REQ_7								+										
REQ_8					+													
REQ_9						+												
REQ_10													+					
REQ_11	+																	
REQ_12											+							
REQ_13												+						
REQ_14									+	+								
REQ_15															+			
REQ_16																+		
REQ_17																	+	
REQ_18																		+

Рисунок 2.4 – Матриця трасування вимог

2.4 Розроблення нефункціональних вимог

Було виділено такі нефункціональні вимоги до ПЗ:

– українська локалізація, тобто в основному інтерфейс має бути українською, відстань вимірюватись в кілометрах, відповідний формат дати, валюта гривня;

– система має мати зручний та інтуїтивно зрозумілий інтерфейс користувача задля більш приємного досвіду користування.

Висновки до розділу

У рамках даного розділу було побудовано загальну діаграму варіантів використання з відображенням акторів різних ролей та основними варіантами використання, що мають бути реалізовані для кожної ролі користувачів. Було деталізовано більшість варіантів використання.

Було визначено мінімальні системні вимоги мобільних пристроїв для коректної роботи застосунку.

Побудовано модель вимог в загальному вигляді. Виділено 18 основних функціональних вимог. В моделі також було деталізовано деякі з них. Також було описано більш детально кожну з вимог та аргументовано пріоритети.

Було побудовано матрицю трасування, що відображає покриття функціональних вимог варіантами використання.

Виділено основні нефункціональні вимоги до програмного забезпечення.

3 КОНСТРУЮВАННЯ ТА РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Архітектура програмного забезпечення

У загальному обрані архітектурні підходи було описано в одному з попередніх розділів. У цьому розділі деталізовано окремі компоненти, з яких складається архітектура застосунку. Побудовано діаграму компонентів, на яку винесено основні складові та взаємозв’язок між ними представлено у графічному матеріалі, креслення 2.

Основний підхід взаємодії це Client-Server. Взаємодія між клієнтом та сервером відбуваєтьсяз завдяки двом протоколам. Базовий функціонал застосунку працює за HTTP request/response. Частина функціоналу, що відповідає за сповіщення в режимі реального часу, працює за WebSocket з’єднанням. Для цього на клієнтській та серверній частинах створені окремі модулі HUB, що відповідають за цю взаємодію.

Клієнтська частина це мобільний застосунок, розроблений за допомогою React Native. React Native за своїм загальним підходом досить схожий на звичайний React, він не має конкретної архітектури, яку можна було б зобразити на діаграмі, як, до прикладу MVC архітектура в Angular.

Проте можна виділити основні складові більшості застосунків на React Native. Service, у випадку даної розробки це axios, відповідає за взаємодію з серверною частиною застосунку. Service надсилає запити та отримує відповіді.

Дані, отримані у відповідях, відображаються користувачу, використовуючи компоненти, тобто Components. Практично вся візуальна складова застосунку складається з власнописних, або готових RN компонентів. Оскільки за допомогою RN розробляються cross-platform застосунки, для розробки використовуються компоненти RN. Під час виконання ці компоненти транслюються в native-компоненти, тобто окремі для кожної платформи, за допомогою механізму моста, тобто Bridge. Міст

забезпечує комунікацію Javascript та RN компонентів з native-кодом та native-модулями (рисунок 3.1) [16].

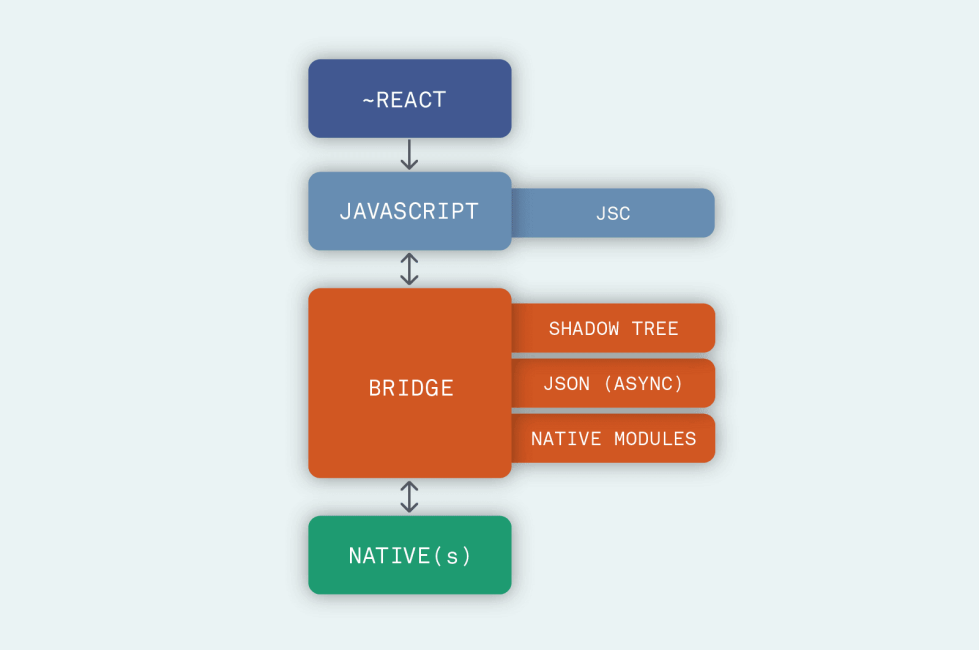


Рисунок 3.1 – Схематичне представлення архітектури RN застосунку [16]

Щодо серверної частини, реалізовано 3-layered архітектуру. Кожен шар архітектури представлений як окремий компонент [17].

Presentation layer це контролери. Кожен контролер відповідає за конкретну логіку. До прикладу, окремий контролер відповідає за дії з оголошеннями, окремий відповідає за дії з заявками. Вони отримують запити ззовні та віддають певні відповіді назовні, тобто рівень представлення. Цей шар не може взаємодіяти напряму з сутностями та базою даних. Він може лише звертатись до шару з бізнес логікою та викликати його методи.

Наступний шар це Business logic layer. Цей шар відповідає за бізнес логіку застосунку. Всередині цього шару є певні сервіси. Також кожен сервіс відповідає за свою частину функціоналу. Цей шар може взаємодіяти з сутностями рівня бази даних та з базою даних, та віддавати назовні, тобто в шар представлення, певні дані.

Найнижчий шар це Data access layer, тобто шар доступу до даних. Він відповідає за взаємодію з базою даних та віддає певні дані в шар з бізнес логікою. Всередині цього шару реалізовано 3 патерни, що дозволяють зробити

архітектуру проекту кращою та зменшити рівень залежності між рівнями. Це Unit of Work, Repository, Specification патерни.

Repository патерн полягає в тому, щоб винести логіку доступу до конкретних даних в окремі репозиторії [18]. Завдяки цьому за потреби можливо підміняти репозиторії та працювати з іншими постачальниками даних. Для деяких задач можна робити один інтерфейс шаблонного репозиторію для всіх сутностей та його реалізацію, проте в реалізації в рамках даного дипломного проекту було створено різні інтерфейси та реалізації для кожного репозиторія, адже кількість методів, що повторюється у всіх репозиторіях, є невеликою.

Наступний патерн це Unit of Work [18]. Він дозволяє певним чином систематизувати усю роботу з репозиторіями та об’єднати роботу з даними в цілому. Реалізація цього патерну гарантує, що всі репозиторії використовують один і той самий контекст даних. Практично є одне місце, в якому відбувається вся робота з даними та можна зручно підключати Unit of Work у сервіси в шарі бізнес логіки. Задля кращого розуміння було схематично зображено взаємодію між сервісами та даними (рисунок 3.2).

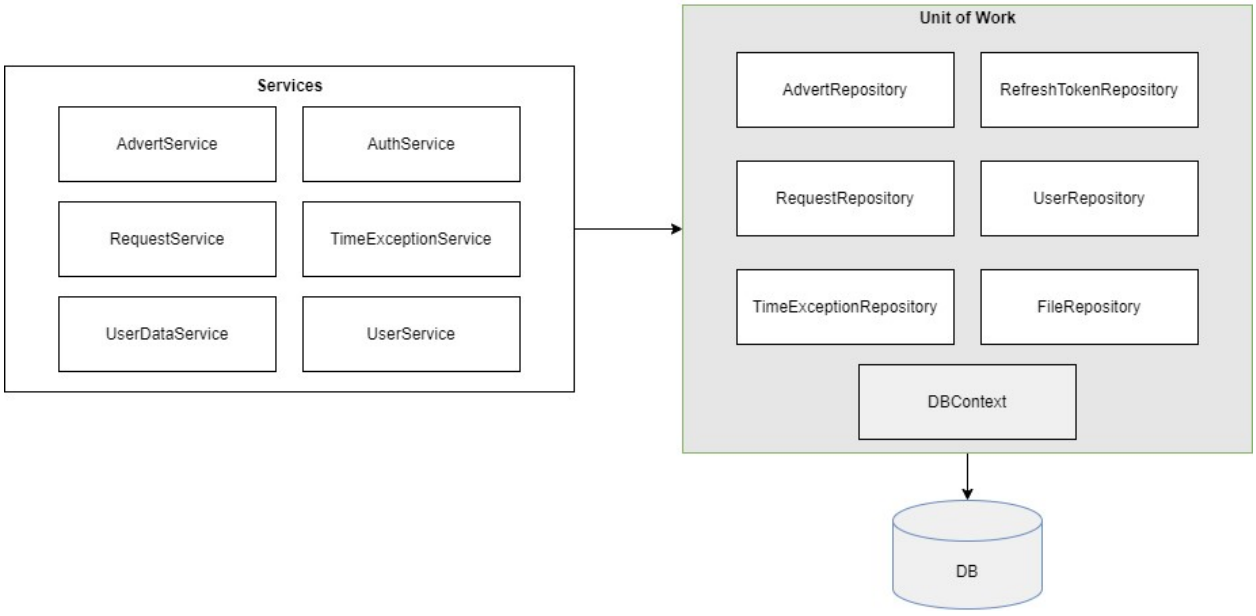


Рисунок 3.2 – Схематичне представлення Unit of Work патерну

Щодо Specification патерну [19]. Його реалізовано задля того, щоб інкапсулювати частину логіку запитів даних в одне місце. Таким чином, можна створювати певні класи, що називаються специфікаціями та перевикористовувати у багатьох місцях застосунку, де потрібні саме ці набори даних. Ідея в тому, що методи в репозиторіях можуть приймати ці специфікації з сервісів та знають як їх можна обробляти. В сервіси вони повертають вже необхідні дані. Таким чином, в сервісі необхідно з потрібного репозиторію, що міститься в Unit of Work, викликати метод, що поверне певні сутності за специфікацією, яка передається в цей метод. Завдяки такій реалізації, репозиторії не «розростаються» від багатьох різних методів, щоб діставати певні вибірки сутностей, а лише мають методи, що вміють працювати з специфікаціями. Таким чином, коли в шарі з бізнес логікою необхідно отримати певну нову вибірку, необхідно лише описати нову специфікацію для неї та викликати необхідний метод репозиторію, що обробить її та поверне необхідний набір сутностей.

Для сервісів, репозиторіїв, одиниці роботи, реалізовано інтерфейси. Це зроблено для того, щоб зменшити залежність між шарами та складовими цих шарів. Це забезпечує можливість впровадження залежностей через конструктор та дозволяє легко підміняти їх реалізації за необхідності. Це робить ПЗ більш гнучким до масштабування та певною мірою спрощує написання тестів.

Дані в застосунку є чітко структурованими, зв’язки між ними є визначеними. Предметна область не вимагає дуже високої швидкості збереження та отримання даних. Об’єми даних та кількість транзакцій також не є великими, типи записів не змінюються динамічно. Тому було обрано SQL реляційну базу даних. NoSQL, тобто нереляційні бази даних, краще підходять в застосунках з високою ступінню навантаженості та у випадку, якщо типи даних можуть змінюватись динамічно. До прикладу, система збору та аналізу даних в режимі реального часу.

3.2 Обґрунтування вибору засобів розробки

Слід обґрунтувати вибір засобів розробки.

Для розробки клієнтської частини застосунку було обрано React Native. Даний фреймворк дозволяє розробляти cross-platform рішення, використовуючи мову Javascript, або її типізований варіант – TypeScript. Як було зазначено, cross-platform застосунки більш раціонально використовують бюджет замовників та ресурс розробників, при цьому забезпечуючи достатню швидкодію та можливості native-застосунків. З-поміж основних аналогів React Native, можна виділити Flutter, проте є кілька ключових відмінностей між даними фреймворками [20].

Flutter використовує мову програмування Dart, що розроблена Google, для побудови користувацьких інтерфейсів. Dart є компільованою мовою, що дозволяє збільшити швидкодію. В той час, як React Native використовує Javascript, що є більш популярним та широко використовується у WEB-розробці. RN дозволяє використовувати нативні компоненти кожної платформи, що забезпечує більш звичний вигляд застосунку для користувачів певної платформи. RN є трішки старшим фреймворком та має більшу спільноту розробників, більшу кількість допоміжних бібліотек та документації.

Враховуючи предметну область, висока швидкодія не є критичною вимогою до програмного забезпечення. Тому для розробки клієнтської частини даного дипломного проекту було обрано фреймворк React Native та мову програмування Javascript, оскільки вона є трішки простішою для розробки [21] та популярнішою на даний момент часу. Завдяки цьому існує більше прикладів написання коду, досвідчених розробників.

Щодо серверної частини застосунку. Вибір з-поміж мов програмувань та фреймворків є досить широким. До прикладу, з мов програмування це може бути PHP, Java, Javascript, .NET, Python. Для реалізації даного застосунку можна було використати будь-яку з цих мов програмування. Проте було обрано саме мову C#, а саме технологію ASP.NET [22], що використовується

для створення back-end частини застосунків. На рішення вплинула наявність досвіду використання технології, якісна документація від Microsoft, велика кількість розширень та бібліотек. З-поміж яких SignalR [23].

Ця бібліотека допомагає створювати додатки, що працюють в режимі реального часу. Відбувається двонаправлений зв'язок для обміну між клієнтом та севрером, завдяки чому сервер може надсилати певні дані клієнту асинхронно в режимі реального часу. У застосунку PetHome завдяки цій бібліотеці реалізовано функціонал відправки користувачу сповіщень та оновлення вмісту деяких сторінок в режимі реального часу. Представлено схему як SignalR Hub надсилає сповіщення користувачам (рисунок 3.3) [24].

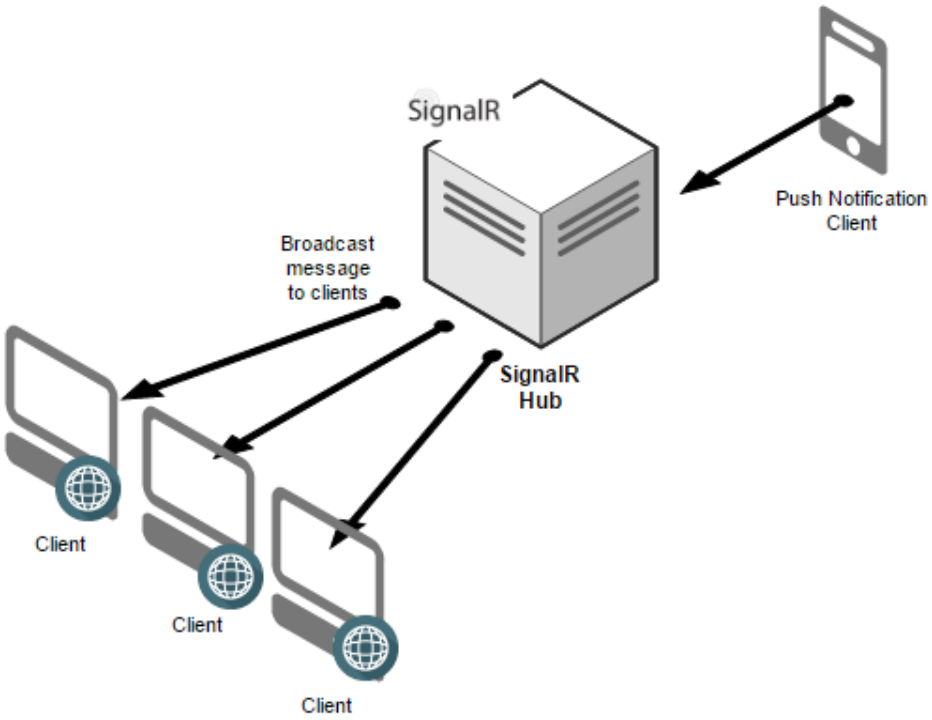


Рисунок 3.3 – Схематичне представлення роботи SignalR хабу [24]

SignalR підтримує кілька видів з'єднання. Вони відрізняються тим, яким чином підтримується зв'язок клієнта з сервером та відслідковуються оновлення. Найбільш оптимальним є WebSockets, його використано в реалізації в рамках даного дипломного проекту.

Систему керування базами даних, тобто СКБД, було обрано базуючись на досвіді роботи, можливості генерування схеми бази даних, наявності провайдера для Entity Framework [25]. Було обрано саме MS SQL, оскільки вона задовольняє цим критеріям, є рішенням від Microsoft, завдяки чому добре поєднується з ASP.NET технологією та середовищем розробки Visual Studio.

Задля взаємодії ASP.NET об’єктів з записами у базі даних, потрібно обрати ORM(object-relational mapping) технологію [26]. Найбільш популярними інструментами є Entity Framework та Dapper. Загалом Dapper має менший набір готового функціоналу, проте має кращу швидкодію. Dapper є більш низькорівневим, дозволяє писати SQL код. Щодо Entity Framework, дана технологія дозволяє перетворювати створені класи сутностей в таблиці. Також дана технологія дозволяє не писати власноруч SQL-запити, а використовувати більш високорівневі технології, такі, як LINQ-запити безпосередньо в коді, написаному мовою C#. Недоліком Entity Framework є нижча швидкодія. Проте завдяки вищому рівню абстракції, дозволяє розробляти програмне забезпечення швидше. Враховуючи предметну область, висока швидкодія не є критично важливою, тому було обрано більш високорівневий інструмент – Entity Framework.

У якості IDE було обрано Visual Studio та Visual Studio Code. Visual Studio Code для розробки клієнтської частини було обрано через високу популярність, наявність пакетів та розширень, що допомагають покращити зручність написання коду. До прикладу, налаштування лінтерів, гарячих клавіш, сніпетів. Для серверної частини застосунку було обрано Visual Studio, оскільки в цьому IDE одразу наявний менеджер пакетів, можливість інтеграції з Docker, вбудована можливість роботи з MS SQL базою даних. Також при виборі було враховано досвід роботи з даними IDE.

3.3 Конструювання програмного забезпечення

3.3.1 Опис проєкту

Загалом взаємодія між клієнтом та сервером відбувається завдяки HTTP-протоколу. Суть в тому, що при такому з’єднанні для отримання оновлень слід зробити запит. У випадку взаємодії в режимі реального часу – потрібно використовувати інший тип з’єднання. В такому випадку є WebSocket. Це протокол, що дозволяє встановити двосторонній зв’язок між клієнтом та сервером та обмінюватись інформацією в режимі реального часу. Для роботи з WebSocket в ASP.NET є спеціальна бібліотека – SignalR [23]. Загалом SignalR підтримує кілька різних типів з’єднань, що відрізняються тим, як підтримується постійний зв’язок між клієнтом та сервером. У рамках даної роботи SignalR використовує WebSocket.

Реалізується спеціальний клас, що наслідує SignalR Hub, він використовується для надсилання сповіщень користувачам. Методи цього хабу викликаються в контролерах коли потрібно розіслати сповіщення. Клієнтська частина застосунку встановлює з’єднання з хабом та певним чином оброблює та показує користувачу нотифікації, що приходять з хабу.

Однією з ключових частин застосунку є робота з місцеположенням користувача, адже потрібно враховувати відстань між користувачами при підборі виконавців. Для початку слід загально визначитись яким чином має відбуватись робота з локаціями.

В застосунку потрібно мати 2 представлення локації користувача: у вигляді тексту, тобто так, як буде зрозуміло для інших користувачів та у вигляді двох координат, довготи та широти, щоб можна було за формулою обчислювати відстань між двома точками. Для реалізації такої роботи з локацією є готові API від Google. Places API [27] дозволяє зробити автоматичне доповнення локації коли користувач вводить своє місцеположення вручну. Також використовується Geocoding API [28] задля того, щоб за назвою місця дізнатись його координати.

Задля того, щоб з координат дізнатись місцезнаходження у вигляді тексту, слід скористатись Reverse Geocoding. Загалом використання API від Google є досить не дешевим, тому було прийнято рішення зберігати в базі даних не лише координати локації користувача, а і його місцеположення у текстовому вигляді. Таким чином не потрібно щоразу використовувати Reverse Geocoding для того, щоб текстово відобразити місцеположення користувача в його профілі.

Для обчислення відстані між двома точками, що задані координатами довготи та широти, використовується загальновідома формула.

В якості системи керування базами даних використовується MS SQL. База даних серверу призначена для зберігання даних користувачів, а також даних про їх оголошення, заявки, заброньовані дати, токени, що необхідні для аутентифікації.

Оскільки в застосунку присутня логіка авторизації та аутентифікації, використовується ASP.NET Identity, що дозволило спростити частину цієї роботи. Через це в базі даних також присутні допоміжні таблиці, що були автоматично згенеровані, їх не описано в рамках даного розділу, адже не всі з них використовуються. Опис таблиць бази даних та їх полів, що було створено, наведено у таблицях 3.1 - 3.7. Модель бази даних наведена у графічному матеріалі, креслення 3.

Таблиця 3.1 – Опис таблиці AspNetUsers

Таблиця	Назва поля	Тип даних	Опис
AspNetUsers	Id	nvarchar(450)	ідентифікатор
	surname	nvarchar(max)	прізвище
	name	nvarchar(max)	ім’я
	sex	int	стать

Продовження таблиці 3.1

Таблиця	Назва поля	Тип даних	Опис
AspNetUsers	UserName	nvarchar(256)	унікальне ім’я користувача
	Email	nvarchar(256)	електронна скринька
	PasswordHash	nvarchar(max)	пароль в захешованому вигляді
	PhoneNumber	nvarchar(max)	номер телефону
	locationLat	float	координата місцеположення широти
	locationLng	float	координата місцеположення довготи
	location	nvarchar(max)	місцеположення в текстовому вигляді
	photoFilePath	nvarchar(max)	шлях до фото

Таблиця 3.2 – Опис таблиці adverts

Таблиця	Назва поля	Тип даних	Опис
adverts	Id	int	ідентифікатор
	name	nvarchar(max)	назва
	cost	int	сума виплати
	description	nvarchar(max)	опис
	status	int	статус
	startTime	dateTime2(7)	час початку виконання

Продовження таблиці 3.2

Таблиця	Назва поля	Тип даних	Опис
adverts	endTime	dateTime2(7)	час закінчення виконання
	ownerId	nvarchar(450)	посилання на профіль власника
	performerId	nvarchar(450)	посилання на профіль виконавця
	locationLat	float	координата місцеположення широти
	locationLng	float	координата місцеположення довготи
	location	nvarchar(max)	місцеположення в текстовому вигляді
	photoFilePath	nvarchar(max)	шлях до фото

Таблиця 3.3 – Опис таблиці refreshTokens

Таблиця	Назва поля	Тип даних	Опис
refreshTokens	id	int	ідентифікатор
	token	nvarchar(max)	JWT refresh токен
	created	dateTime2(7)	час створення токену
	expires	dateTime2(7)	час коли токен стане не актуальним
	ownerId	nvarchar(450)	посилання на власника

Продовження таблиці 3.3

Таблиця	Назва поля	Тип даних	Опис
refreshTokens	isNotActual	bit	чи актуальний токен(не актуальний якщо юзер використав вже цей токен, щоб отримати новий)

Таблиця 3.4 – Опис таблиці timeExceptions

Таблиця	Назва поля	Тип даних	Опис
timeExceptions	id	int	ідентифікатор
	userId	nvarchar(450)	посилання на користувача, якому належить ця заброньована дата
	date	dateTime2(7)	недоступна дата

Таблиця 3.5 – Опис таблиці requests

Таблиця	Назва поля	Тип даних	Опис
requests	id	int	ідентифікатор
	userId	nvarchar(450)	посилання на профіль користувача, який подав цю заявку
	status	int	статус заявки

Таблиця 3.6 – Опис таблиці AspNetRoles

Таблиця	Назва поля	Тип даних	Опис
AspNetRoles	Id	nvarchar(450)	ідентифікатор ролі
	Name	nvarchar(256)	назва ролі
	NormalizedName	nvarchar(256)	статус заявки
	ConcurrencyStamp	nvarchar(max)	штамп паралельності

Таблиця 3.7 – Опис таблиці AspNetUserRoles

Таблиця	Назва поля	Тип даних	Опис
AspNetUserRoles	UserId	nvarchar(450)	Ідентифікатор користувача
	RoleId	nvarchar(450)	Ідентифікатор ролі

3.3.2 Опис утиліт, бібліотек та іншого стороннього програмного забезпечення

Опис утиліт, бібліотек та іншого стороннього програмного забезпечення, що використовується у розробці наведено в таблиці 3.8.

Таблиця 3.8 – Опис утиліт

№ п/п	Назва утиліти	Опис застосування
1	Microsoft Visual Studio	Головне середовище розробки програмного забезпечення серверної частини застосунку.

Продовження таблиці 3.8

№ п/п	Назва утиліти	Опис застосування
2	Postman	Програмне забезпечення необхідне для тестування rest запитів. Використовувалось для тестування API інтерфейсів, та клієнтських запитів.
3	SQL Server Express	Зручна система керування базами даних
4	Visual Studio Code	Текстовий редактор, що використовувався для написання коду переважно клієнтської частини застосунку
5	Docker	Програмне забезпечення, що дозволяє автоматизувати розгортання додатку
6	SignalR	Бібліотека, що дозволяє надсилати асинхронні повідомлення
7	Google Places API	API, що дозволяє автоматично доповнювати назву локації

Продовження таблиці 3.8

№ п/п	Назва утиліти	Опис застосування
8	Google Geocoding API	API, що дозволяє з назви визначити координати місцеположення
9	Entity Framework	Фреймворк об’єктно-реляційного відображення, що дозволяє працювати з даними як з об’єктами класів
10	React-native-calendars	Компонент, що дозволяє працювати з різними видами календарів в клієнтській частині застосунку

Тексти програмного коду наведені в окремому документі «Текст програми».

3.4 Аналіз безпеки даних

Загалом до системи не висувається серйозних вимог з приводу безпеки даних, та все ж потрібно дотримуватись принципів, що є загальними практично для будь-якого ПЗ, яким мають користуватись аутентифіковані користувачі.

У цьому підрозділі більш детально пояснено як влаштовані процеси авторизації та аутентифікації. У застосунку ці процеси реалізовано за допомогою JWT-токенів: access та refresh. Стисло слід пояснити що таке JWT [29]. Це певний токен доступу, що складається з таких частин, як заголовок, навантаження та підпис. В цьому токені можна зберігати дані про користувача, його роль. У рамках даної реалізації в токені зберігається унікальний ідентифікатор користувача, щоб розуміти хто саме звертається з таким токеном, роль користувача. Дані в токені закодовані та підписані певним

підписом. Таким чином ніхто інший, крім застосунку, що знає цей підпис, не зможе видавати користувачам ці токени.

Якщо видати користувачу access токен на довгий період часу то зловмисник рано чи пізно якимось чином зможе заволодіти ним та, прикріпивши до свого запиту, зможе отримати доступ до даних користувача. Тому слід використовувати ще один – refresh токен [30]. Таким чином можна реалізувати короткий час життя для access токена, до прикладу, 15 хв, а завдяки refresh токену користувач зможе отримувати нову пару токенів. Налаштовано, що refresh токен валідний протягом 7 днів. Завдяки цьому користувачу не потрібно буде вводити логін та пароль якщо він буде не в мережі менше, ніж 7 днів.

Часто access токени зберігають в локальному сховищі користувача, що є не досить безпечно проти XSS-атак, тобто атаки за допомогою користувацького зловмисного скрипта, що певним чином включається в сторінку [31]. Задля того, щоб зменшити ризок такої та деяких інших атак в нашому токени зберігаються у вигляді HTTP-only cookie. Їх суть полягає в тому, що клієнтська частина застосунку не має жодного доступу до них. Тобто скриптом не можливо просто дістати ці cookie. Їх можна встановлювати та обробляти лише у серверній частині застосунку. Завдяки цьому зловмиснику складніше підставити їх в запит або стягнути їх скриптом у клієнтській частині застосунку.

Також слід приділити увагу протоколу передачі. Потрібно розібратись з різницею між HTTP та HTTPS. HTTP протокол передачі передає інформацію у звичайному вигляді, а HTTPS у зашифрованому. Тобто HTTPS має свої певні надбудови задля безпеки, завдяки чому ним можна передавати таку інформацію як паролі та інші особисті дані.

Також слід правильно налаштувати політику CORS у серверній частині, щоб можна було виконувати запити з клієнтської частини, що знаходиться на іншому домені. Політика CORS це певний механізм безпеки, що дозволяє WEB-застосункам використовувати дані, що знаходяться на інших доменах.

Тобто це певний захист для сторінок, які не відповідають політиці одного походження [32].

Висновки до розділу

Було деталізовано архітектуру програмного забезпечення. Представлено діаграму компонентів, обгрунтовано архітектурні рішення. Описано складові та їх функціонал, патерни, що використовуються.

Розглянуто можливі засоби розробки, за допомогою яких можна реалізувати ПЗ в рамках даного дипломного проекту. Аргументовано вибір допоміжних програмних засобів розробки після проведеного порівняльного аналізу.

Описано проект в цілому. Побудовано модель представлення бази даних, описано основні таблиці та поля.

Виділено та описано бібліотеки, утиліти та інше стороннє програмне забезпечення, що використовуються для реалізації функціоналу.

Проаналізовано рівень безпеки даних застосунку, можливі вразливості та описано підходи та методи, завдяки яким досягається прийнятний рівень безпеки.

4 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Аналіз якості ПЗ

Загалом потрібно проаналізувати 2 частини коду: клієнтську та серверну. Оскільки серверна частина коду є більш структурованою, має більш складну архітектуру та реалізує певні патерни, то ця частина коду є легшою до оцінювання її якості. Існує багато метрик для оцінки коду. На їх вибір впливає тип застосунку та мета, з якою виконується оцінка якості.

Метриками для оцінки якості back-end частини ПЗ обрано наступні:

– Maintainability (підтримуваність);
– Cyclomatic Complexity (цикломатична складність);
– Depth of Inheritance (глибина наслідування);
– Class Coupling (зв’язність класів).

Для аналізу серверної частини застосунку використано вбудований у Visual Studio статичний аналізатор коду. На рисунку 4.1 наведено результати роботи аналізатора. Детальний аналіз якості ПЗ наведено в пунктах 4.1.1 – 4.1.5.

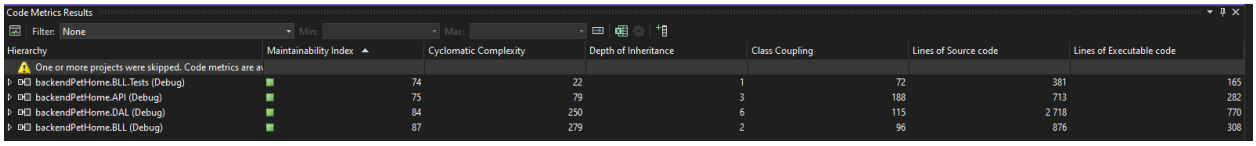


Рисунок 4.1 – Результати вбудованого аналізатора

4.1.1 Підтримуваність

Основний комплексний показник це Maintainability [33]. Він вираховується, базуючись на таких факторах, як складність коду, його структура, наявність проблем у коді, виявлених аналізатором. Набуває значень від 0 до 100 Вище значення цього показника є кращим. Як бачимо, для всіх проектів рішення цей показник є більшим за 70, що означає, що код є досить хорошим в плані його підтримуваності.

Значення метрики Maintainability для проектів рішення наведені в таблиці 4.1.

Таблиця 4.1 – значення метрики “Maintainability ” для проектів

Проект	Значення метрики
backendPetHome.BLL.Tests	74
backendPetHome.BLL.API	75
backendPetHome.BLL.DAL	84
backendPetHome.BLL.BLL	87

4.1.2 Цикломатична складність

Наступним є показник Cyclomatic Complexity. Він вираховується, базуючись на потоці виконання коду та вирахуванні кількості різних шляхів, в які може зайти виконання коду [33]. Нижче значення цього індекса є кращим. Зазвичай його вираховують не на проект в цілому та не на окремі класи, а конкретні методи або функції. Задля цього можна, до прикладу, переглянути значення цього показника для методів одного з сервісів (рисунок 4.2).

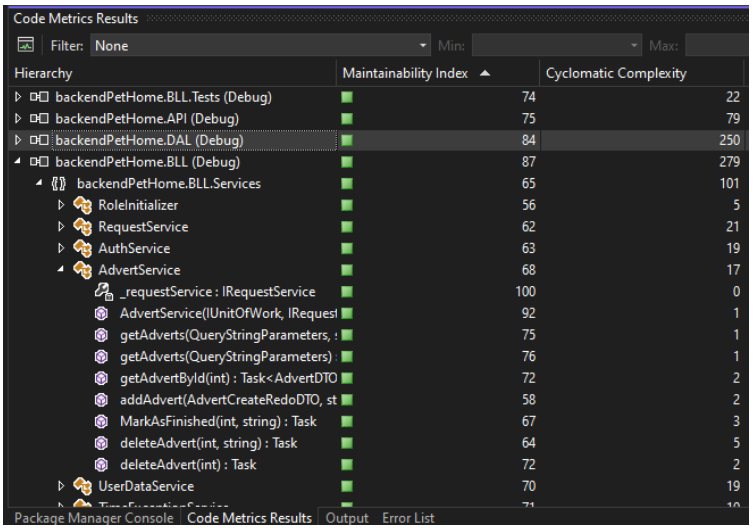


Рисунок 4.2 – Cyclomatic Complexity для більш дрібних складових

Хороші значення цього показника залежать від складності та розміру проекту. У якості орієнтиру можна взяти значення, що менші за 10. Це є прийнятним результатом для невеликих проектів. Можна спостерігати, що за показником цикломатичної складності розроблене програмне забезпечення показує задовільний результат.

4.1.3 Глибина наслідування

Depth of inheritance вимірює глибину наслідування класів [33]. Вищі значення кажуть про складність класової ієрархії, що робить код більш складним до розуміння та підтримки. Нижчі значення є кращими. Оптимальні значення залежать, знову ж таки, від розміру проекту. Для проекту відносно невеликого розміру прийнятним буде значення нижче 7-10. Можна спостерігати, що аналізатор вказує, що ієрархія класів не є сильно складною в ПЗ. Значення метрики для кожного проєкту наведене в таблиці 4.2.

Таблиця 4.2 – значення метрики “Depth of inheritance” для проектів

Проект	Значення метрики
backendPetHome.BLL.Tests	1
backendPetHome.BLL.API	3
backendPetHome.BLL.DAL	6
backendPetHome.BLL.BLL	2

4.1.4 Зв’язність коду

Щодо Class Coupling. Як і у випадку Cyclomatc Complexity, немає конкретного показника, що можна було б використовувати для оцінки всіх проектів. Нижчі значення є кращими [33]. Слід розглядати це значення для конкретних класів. Для прикладу можна розглянути значення для одного з основних методів застосунку в репозиторії, що відповідає за користувачів (рисунок 4.3).

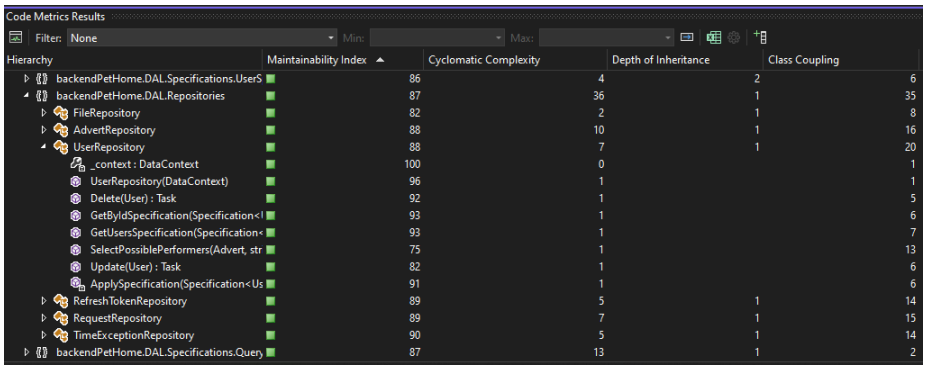


Рисунок 4.3 – Class Coupling для більш дрібних складових

Для класу значення 20, в методі SelectPossiblePerformers значення 13. Значення є доволі високим, адже цей метод посилається та працює з об’єктами багатьох класів, реалізовуючи функціонал підбору користувачів на оголошення, що є практично основним функціоналом ПЗ. Якщо розглянути інші класи, можна дійти до висновку, що значення є прийнятними.

Підсумовуючи, в рамках даної роботи було розроблено доволі якісний код, що в майбутньому може модифікуватись та розвиватись. Досягається це завдяки архітектурним патернам, що було застосовано, та дотримуванню певним загальним підходам до розробки ПЗ.

4.1.5 Клієнтська частина

Щодо клієнтської частини, це RN застосунок. Як було зазначено раніше, код клієнтської частини складніше так детально проаналізувати, адже він є комбінацією Javascript, розмітки сторінок, їх стилів. У випадку RN присутній ще й JSX.

В рамках клієнтської чатсини даної роботи використовувались лінтери для перевірки форматування коду. Це дозволяє зробити код легшим для сприйняття та є особливо необхідним, якщо над застосунком працює команда людей. Має бути певний стандарт форматування. Лінтери допомагають досягти цього.

У рамках клієнтської частини даного застосунку було використано лінтер eslint, що містить в собі правила для форматування [34]. Використовувались набори правил “eslint:recommended” та “plugin:react/recommended”. Це допомогло зручно форматувати код, що робило його легшим для візуального сприйняття та розуміння. Налаштування наведено на рисунку 4.4.

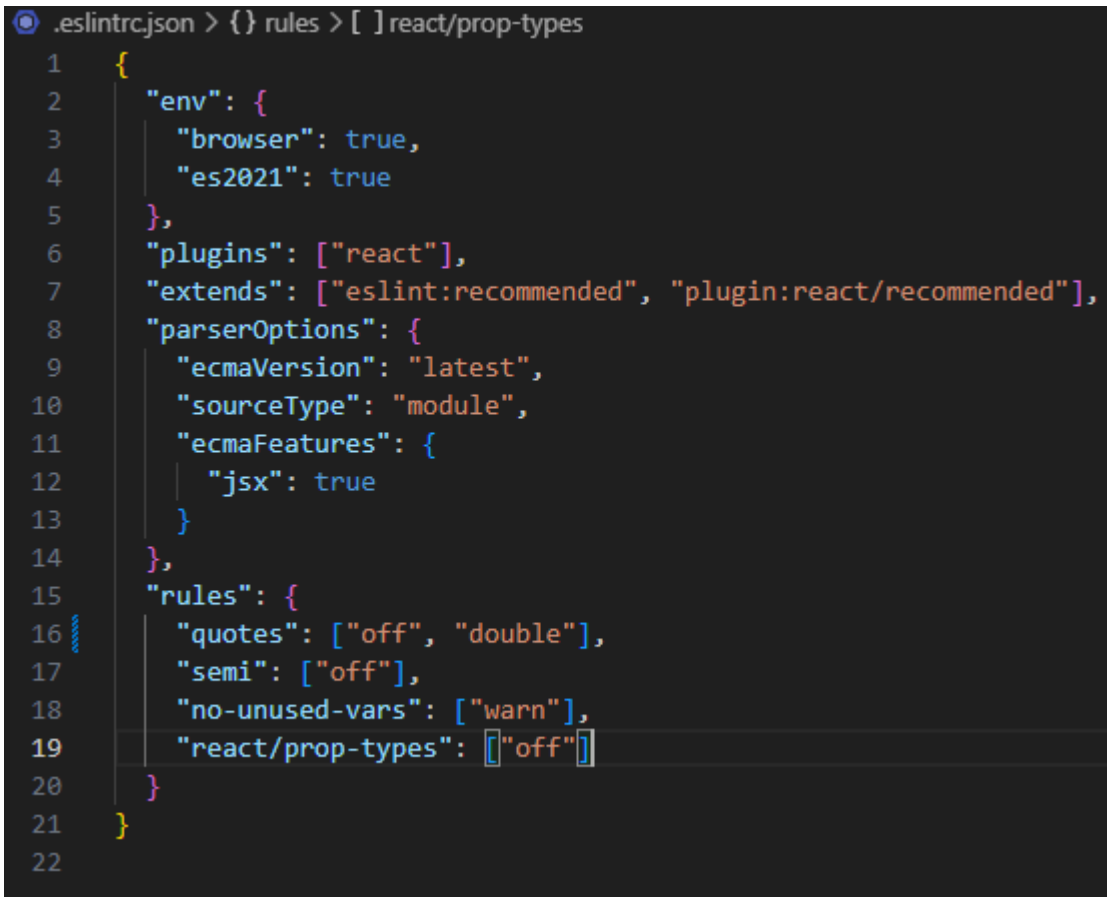


Рисунок 4.4 – Вміст файлу з конфігурацією .eslintrc.json

4.2 Опис процесів тестування

Тестування виконується згідно документу «Програма та методика тестування».

Задля тестування основного функціоналу ПЗ було проведено мануальне тестування.

Також проведено модульне тестування частини сервісів. Опис тестів, що було проведено мануально, наведено у таблицях 4.3 – 4.16.

Таблиця 4.3 – Тест 1. Реєстрація користувача

Початковий стан системи	Користувач знаходиться на сторінці реєстрації
Вхідні дані	Прізвище, ім’я, стать, місцеположення, електронна пошта, номер телефону, логін, пароль, підтвердження паролю, фото

Продовження таблиці 4.3

Опис проведення тесту	У відповідні поля вводяться: прізвище та ім’я з мінімальними довжинами у два символи, адреса електронної пошти, що відповідає загальному шаблону, український номер телефону, унікальний логін, пароль довжиною як мінімум 8 символів, який містить хоча б одну малу та хоча б одну велику літери, як мінімум одне число і один спеціальний символ, підтвердження паролю, яке співпадає з раніше введеним паролем. Також потрібно ввести власноруч та обрати з випадаючого списку, або надати дозвіл та, тим самим, автоматично заповнити місцеположення, додати фото формату .jpg/png/jpeg Після цього слід натиснути на кнопку реєстрації.
Очікуваний результат	Реєстрація проходить успішно, новий профіль користувача додається у систему. Користувач перенаправляється на сторінку авторизації.
Фактичний результат	Реєстрація проходить успішно, новий профіль користувача додається у систему. Користувач перенаправляється на сторінку авторизації.

Таблиця 4.4 – Тест 2. Вхід у профіль

Початковий стан системи	Користувач знаходиться на сторінці авторизації та є зареєстрованим в системі.
Вхідні дані	Логін, пароль
Опис проведення тесту	У відповідні поля коректно вводяться логін існуючого користувача та його пароль. Користувач натискає кнопку входу в систему.

Продовження таблиці 4.4

Очікуваний результат	Авторизація відбувається успішно. Користувач заходить в систему одразу на сторінку оголошень. Тепер він має мати змогу виконувати всі дії, що доступні авторизованим користувачам.
Фактичний результат	Авторизація відбувається успішно. Користувач заходить в систему одразу на сторінку оголошень. Тепер він має мати змогу виконувати всі дії, що доступні авторизованим користувачам.

Таблиця 4.5 – Тест 3. Вхід у профіль за некоректними даними

Початковий стан системи	Користувач знаходиться на сторінці авторизації
Вхідні дані	Логін, пароль
Опис проведення тесту	У відповідні поля вводяться логін не існуючого користувача та його пароль. Користувач натискає кнопку входу в систему.
Очікуваний результат	Авторизація не відбувається. Користувач отримує повідомлення про те, що дані для входу не є коректними.
Фактичний результат	Авторизація не відбувається. Користувач отримує повідомлення про те, що дані для входу не є коректними.

Таблиця 4.6 – Тест 4. Фільтрація оголошень

Початковий стан системи	Користувач знаходиться на сторінці з оголошеннями.
Вхідні дані	Крайні межі виплат (грн), кількість оголошень на сторінці, значення фільтру «У ваші вільні дати»

Продовження таблиці 4.6

Опис проведення тесту	У блоці з фільтрами користувач вказує мінімальну та максимальну межі виплат за виконання оголошення у гривнях. Також він обирає кількість оголошень на сторінці «По 4 шт.». Обирає фільтр «У ваші вільні дати». Натискає кнопку застосування фільтрів.
Очікуваний результат	На сторінці залишаються лише оголошення, що задовольняють обмеженням. У дати всіх оголошень користувач не завантажений. Якщо їх більше за 4 шт, з’являється меню пагінації, в якому можна обирати номер сторінки з оголошеннями.
Фактичний результат	Фільтри застосовано. Всього на сторінці максимум може бути 4 оголошень. З’являється меню пагінації, якщо їх більше за 4.

Таблиця 4.7 – Тест 5. Зміна графіку завантаженості

Початковий стан системи	Користувач знаходиться на сторінці свого профілю
Вхідні дані	Дати, що необхідно додати та видалити
Опис проведення тесту	Користувач натискає кнопку зміни графіку завантаженості. У календарі, користувач обирає, що необхідно додати до недоступних дат та обирає дати, які потрібно видалити. Натискає кнопку збереження.
Очікуваний результат	Автоматично додаються та видаляються дати та відображаються в профілі користувача.
Фактичний результат	Графік завантаженості успішно оновлено та відображається в профілі.

Таблиця 4.8 – Тест 6. Створення оголошення

Початковий стан системи	Користувач знаходиться на сторінці створення оголошення
Вхідні дані	Назва, опис, фото, вартість, місцеположення, дата початку та дата завершення оголошення
Опис проведення тесту	Користувач вводить назву, як мінімум з 5 символів, опис, як мінімум з 10 символів, як максимум з 500 символів. Також вводить вартість у чисельному форматі, як максимум 100000. Також потрібно ввести власноруч та обрати з випадаючого списку, або надати дозвіл та, тим самим, автоматично заповнити місцеположення оголошення, додати фото тварини формату .jpg/png/jpeg. Натиснути кнопку вибору дат та обрати період протягом якого виконавцю слід сидіти з твариною. Далі потрібно натиснути кнопку створення оголошення.
Очікуваний результат	Оголошення має бути створено. Система має автоматично підібрати можливих виконавців та надіслати їм сповіщення.
Фактичний результат	Оголошення створено, можливих виконавців підібрано та їм надіслано сповіщення.

Таблиця 4.9 – Тест 7. Підтвердження згенерованої заявки

Початковий стан системи	Користувач знаходиться на сторінці з його заявками
Вхідні дані	-
Опис проведення тесту	Серед заявок користувача є заявки, які згенерувала система. Користувач натискає кнопку підтвердження однієї з таких заявок.

Продовження таблиці 4.9

Очікуваний результат	Заявка має бути підтверджена. Має відображатись її оновлений статус. Власник оголошення має побачити сповіщення про подачу заявки на його оголошення.
Фактичний результат	Зявку підтверджено, її статус оновлено. Власник отримав сповіщення.

Таблиця 4.10 – Тест 8. Відхилення згенерованої заявки

Початковий стан системи	Користувач знаходиться на сторінці з його заявками
Вхідні дані	Заявки, що згенеровані системою
Опис проведення тесту	Серед заявок користувача є заявки, які згенерувала система. Користувач натискає кнопку відхилення однієї з таких заявок.
Очікуваний результат	Заявка має бути відхилена. Вона не має більше відображатись в заявках користувача.
Фактичний результат	Заявку відхилено, вона більше не відображається на сторінці заявок користувача.

Таблиця 4.11 – Тест 9. Підтвердження заявки, поданої на оголошення користувача

Початковий стан системи	Користувач знаходиться на сторінці його оголошення, на яке було подано заявки.
Вхідні дані	-
Опис проведення тесту	Користувач обирає виконавця серед тих користувачів, що подали заявку та натискає кнопку прийому заявки.

Продовження таблиці 4.11

Очікуваний результат	В оголошення з’являється виконавець. Обраний виконавець має отримати сповіщення про те, що його було обрано. Усі інші мають отримати сповіщення про те, що їх заявки було відхилено.
Фактичний результат	Зявку прийнято, в оголошення з’явився виконавець. Усі користувачі, що подавали заявки, отримати відповідні сповіщення.

Таблиця 4.12 – Тест 10. Відхилення заявки, поданої на оголошення користувача

Початковий стан системи	Користувач знаходиться на сторінці його оголошення, на яке було подано заявки.
Вхідні дані	-
Опис проведення тесту	Користувач обирає яку заявку відхилити. Натискає кнопку відхилення цієї заявки.
Очікуваний результат	Заявка більше не відображається на сторінці оголошення. Статус заявки було оновлено. Виконавець, що подав заявку, отримав сповіщення про те, що її було відхилено.
Фактичний результат	Заявка не відображається на сторінці оголошення. Користувач, що її подав, отримав сповіщення про її відхилення власником. На сторінці з заявками цього користувача ця заявка тепер має оновлений статус.

Таблиця 4.13 – Тест 11. Підтвердження заявки користувача, що має недоступні дати в період, зазначений в оголошенні

Початковий стан системи	Користувач знаходиться на сторінці його оголошення, на яке було подано заявки. Користувач, заявку якого він обере, вже має недоступні дати, що пересікаються з датами виконання оголошення.
Вхідні дані	-
Опис проведення тесту	Користувач обирає виконавця серед тих користувачів, що подали заявку та натискає кнопку прийому заявки.
Очікуваний результат	Користувач має отримати повідомлення про те, що користувач, що подав заявку, вже має недоступні дати в період виконання оголошення та не зможе його виконувати.
Фактичний результат	Заявку не прийнято, отримано повідомлення про неможливість користувачем, що подав заявку, виконувати оголошення

Таблиця 4.14 – Тест 12. Подача заявки на виконання оголошення

Початковий стан системи	Користувач знаходиться на сторінці його оголошення, на яке хоче подати заявку.
Вхідні дані	-
Опис проведення тесту	Користувач натискає кнопку подання заявки на виконання оголошення.
Очікуваний результат	Користувач отримує повідомлення про подачу заявки. Додалась нова заявка, яку можна переглянути на сторінці заявок користувача, власник оголошення отримав сповіщення про подачу.

Продовження таблиці 4.14

Фактичний результат	Зявку подано, користувач отримав повідомлення про подачу. Власник отримав сповіщення.

Таблиця 4.15 – Тест 13. Подача заявки на оголошення, період виконання якого накладається на завантажені дати користувача

Початковий стан системи	Користувач знаходиться на сторінці його оголошення, на яке хоче подати заявку. Користувач має недоступні дати, що пересікаються з датами виконання оголошення.
Вхідні дані	-
Опис проведення тесту	Користувач натискає кнопку подання заявки на виконання оголошення.
Очікуваний результат	Користувач отримує повідомлення про те, що він не може подати заявку, адже має недоступні дати, що пересікаються з датами виконання оголошення.
Фактичний результат	Зявку не подано. Користувач отримав відповідне повідомлення.

Таблиця 4.16 – Тест 14. Вихід з системи

Початковий стан системи	Користувач є авторизованим та знаходиться на будь-якій сторінці застосунку.
Вхідні дані	-
Опис проведення тесту	Користувач натискає кнопку виходу з системи.
Очікуваний результат	Користувач виходить з системи та перенаправляється на сторінку авторизації. Користувач більше не має доступу до даних, що доступні лише авторизованим користувачам.

Продовження таблиці 4.16

Фактичний результат	Користувач переадресовується на сторінку логіну та не має доступу до даних, що доступні лише авторизованим користувачам.

Щодо модульного тестування. Як було зазначено раніше, було виконано модульне тестування кількох сервісів шару з бізнес логікою. Саме тестування було виконано за допомогою інструменту XUnit [35].

Під час цього процесу було використано фреймворк Moq. Завдяки цьому фреймворку можна імітувати об’єкти та вручну налаштовувати те, що мають повертати їх методи. Завдяки використанню інтерфейсів класів, що використовуються в сервісах, можна зручно підставляти необхідні об’єкти, що були штучно налаштовані.

Сама ідея модульного тестування в тому, щоб протестувати роботу окремих складових ПЗ, незалежно від інших його частин.

Також під час процесу модульного тестування було використано бібліотеку Fluent Assertions. Завдяки цьому можна зручно виконувати перевірки та підвищується читабельність коду.

Як приклад можна розглянути тест методу, що відповідає за створення оголошення (рисунок 4.5). Цей метод є доволі складним та використовує в собі об’єкти інших класів.

```
[Fact]
0 references
public async Task addAdvert_CorrectData_AdvertAdded()
{
    //arrange
    Advert advert = _advertToTest;
    var bytes = Encoding.UTF8.GetBytes("This is a dummy file");
    IFormFile file = new FormFile(new MemoryStream(bytes), 0, bytes.Length, "Data", "hairy.jpeg");
    IEnumerable<string> possiblePerformersIds = new List<string>() { "userId1", "userId2"};
    int countPossiblePeformers = 2;

    _unitOfWorkMock.Setup(x =>x.AdvertRepository.Add(advert));
    _unitOfWorkMock.Setup(x =>x.FileRepository.Add(file));
    _unitOfWorkMock.Setup(x =>x.UserRepository.SelectPossiblePerformers(It.IsAny<Advert>(), advert.ownerId)).ReturnsAsync(possiblePerformersIds);
    _requestServiceMock.Setup(x => x.addRequest(It.IsAny<string>(), advert.Id, DAL.Enums.RequestStatusEnum.generated));

    //act
    AdvertCreateRedoDTO advertCreateDTO = _mapper.Map<AdvertCreateRedoDTO>(advert);
    var possiblePerformersIdsAndAdvert  = await _sut.addAdvert(advertCreateDTO, advert.ownerId, file);
    AdvertDTO advertDTO = _mapper.Map<AdvertDTO>(advert);

    //assert
    _unitOfWorkMock.Verify(x => x.AdvertRepository.Add(It.IsAny<Advert>()), Times.Exactly(1));
    _unitOfWorkMock.Verify(x => x.FileRepository.Add(file), Times.Exactly(1));
    _requestServiceMock.Verify(x => x.addRequest(It.IsAny<string>(), advert.Id, DAL.Enums.RequestStatusEnum.generated), Times.Exactly(countPossiblePeformers));
    possiblePerformersIdsAndAdvert.possiblePerformersIds.Should().BeEquivalentTo(possiblePerformersIds);
    possiblePerformersIdsAndAdvert.advertDTO.Should().BeEquivalentTo(advertDTO);
}
```

Рисунок 4.5 – Модульний тест методу додавання оголошення

Під час тестування було дотримано принципу AAA, тобто arrange, act, assert [36]. Він передбачає 3 етапи виконання тесту. Налаштування, дія, перевірка.

На рисунку 4.5 в тесті методу створення оголошення, а саме в секції arrange, налаштовуються Id можливих виконавців, штучний файл, репозиторії.

У розділі act спочатку використовується AutoMapper задля того, щоб отримати необхідний об'єкт оголошення. Далі викликається тестований метод сервісу. Також завдяки AutoMapper отримується DTO (Data Transfer Object) з початкового оголошення [37].

У розділі assert застосовуються Fluent Assertions. Перевіряється кількість викликів необхідних методів репозиторіїв та сервісу, а також перевіряються повернуті Id користувачів та DTO оголошення на рівність тим, що очікуються.

Загалом реалізовано 16 тестів. Усі написані тести виконується успішно, що свідчить про коректну роботу функціоналу, що було покрито модульними тестами (рисунок 4.6).

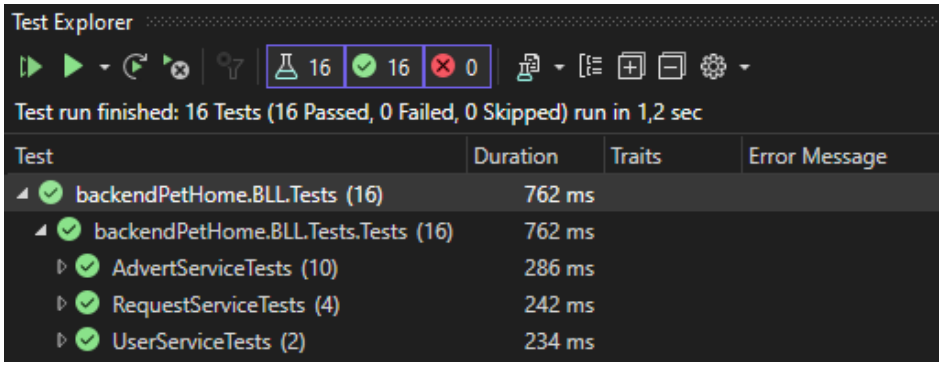


Рисунок 4.6 – Результат виконання тестів

4.3 Опис контрольного прикладу

Оскільки в рамках даної роботи основним функціоналом є підбір виконавців для догляду за тваринами, було описано та продемонстровано саме процес створення оголошення, автоматичного підбору виконавців та сповіщення різних користувачів, їх взаємодії між собою.

Попередньо в застосунку є 2 зареєстрованих користувачі. Слід зайти в профілі користувачів з різних пристроїв (рисунок 4.6).

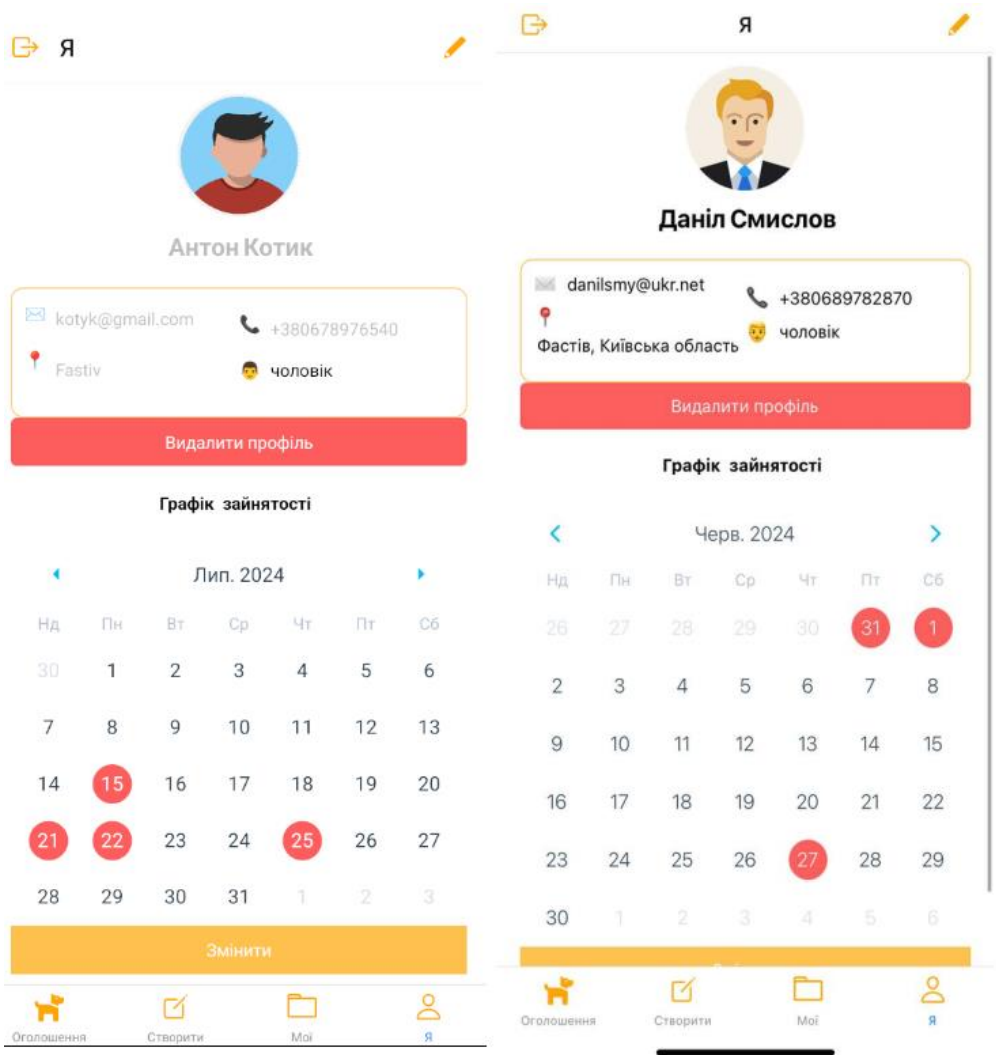


Рисунок 4.6 – Зареєстровані користувачі

Користувач Антон створює нове оголошення (рисунок 4.7). Користувач вводить усі необхідні дані, обирає період, в який виконавець має доглянути за твариною. Обирає фото тварини. Оскільки користувач не надав доступу до геолокації, вводить місцеположення та обирає пропозиції у випадаючому списку. Обирає одну з них.

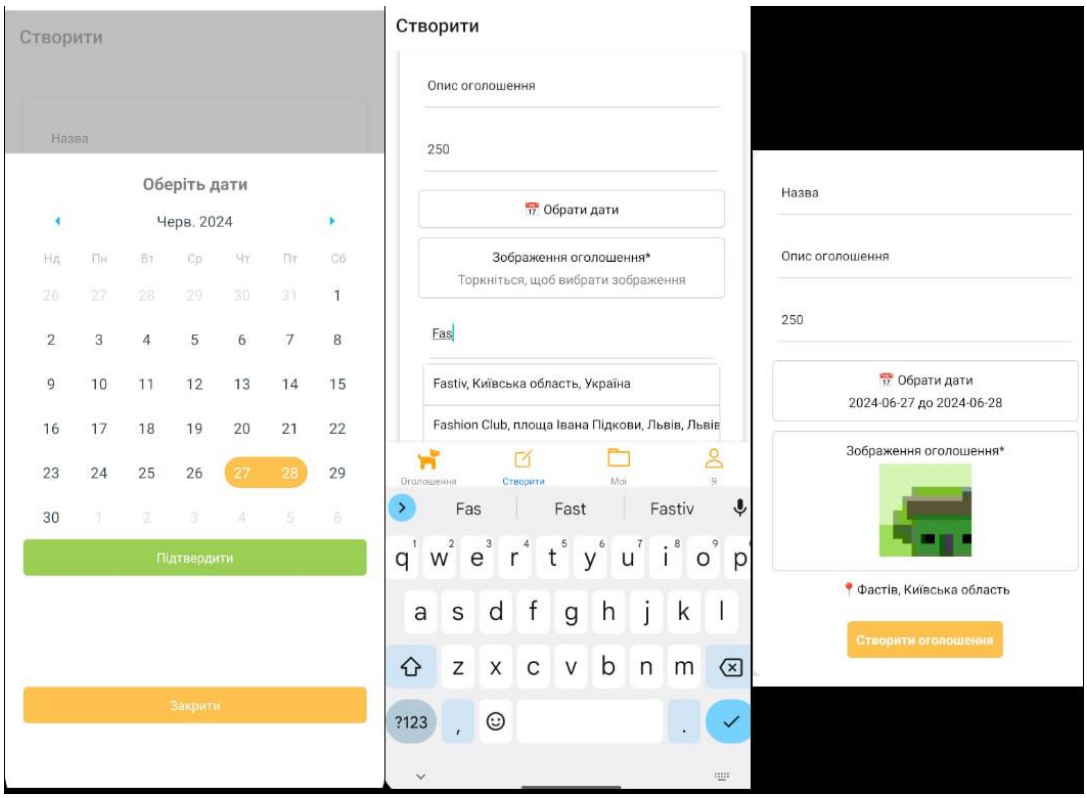


Рисунок 4.7 – Створення оголошення

Користувач Даніл також знаходиться у місті Фастів та 27 та 28 червня не завантажений. Тому користувач отримує сповіщення про підбір системою оголошення (рисунок 4.8).

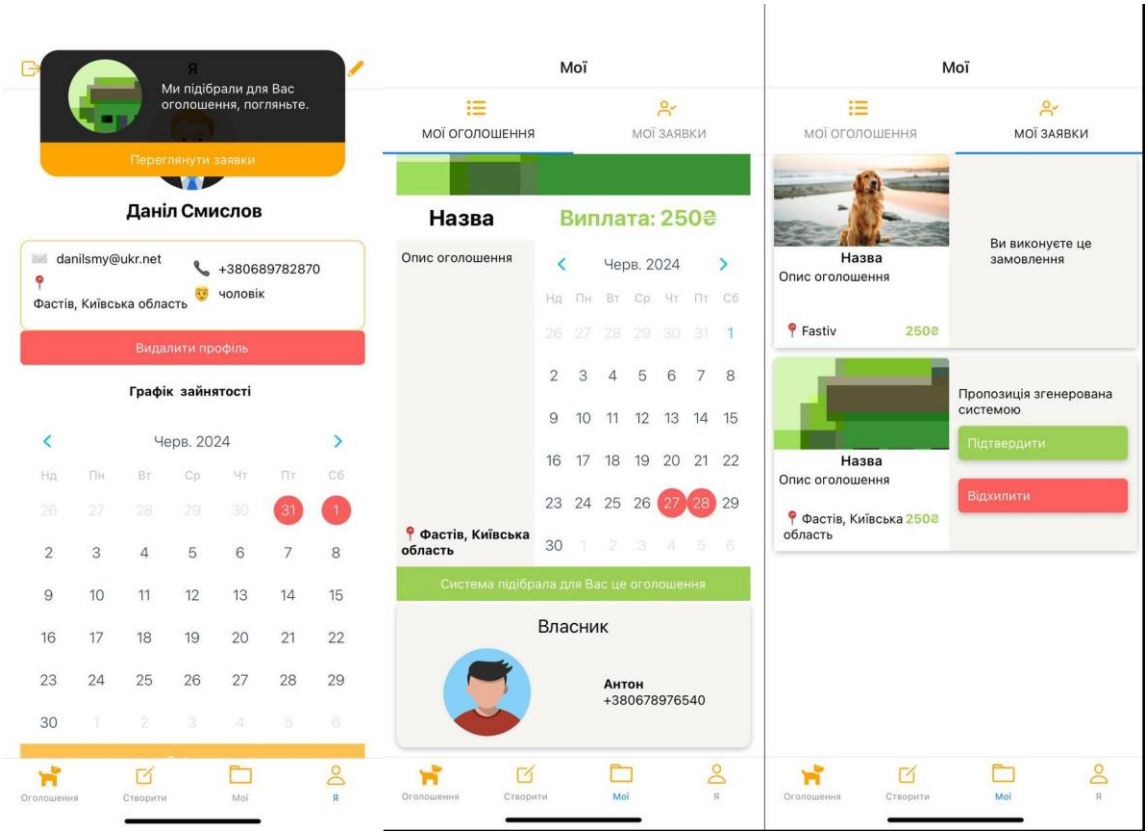


Рисунок 4.8 – Користувач Даніл отримав сповіщення про генерацію заявки

Користувач Даніл підтверджує заявку, згенеровану системою. Користувач Антон отримує відповідне сповіщення та переходить на сторінку свого оголошення (рисунок 4.9).

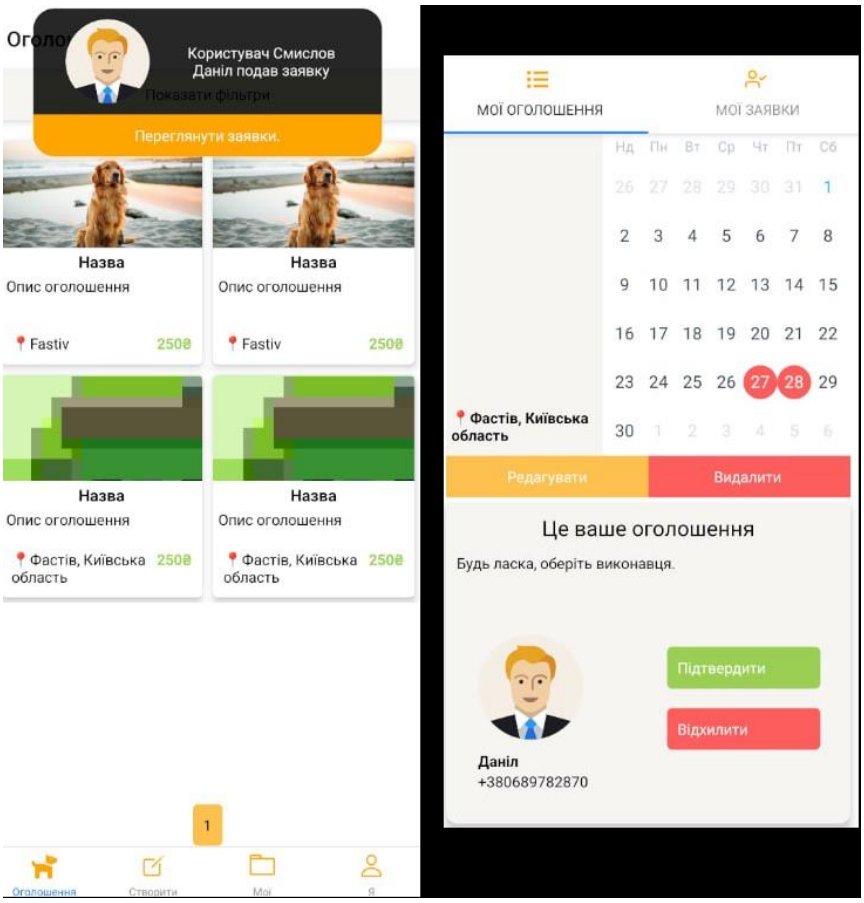


Рисунок 4.9 – Користувач Антон отримав сповіщення про подачу заявки

Користувач Антон може підтвердити або відхилити виконання оголошення Данілом. Антон підтверджує заявку. Користувач Даніл отримує сповіщення про це (рисунок 4.10).

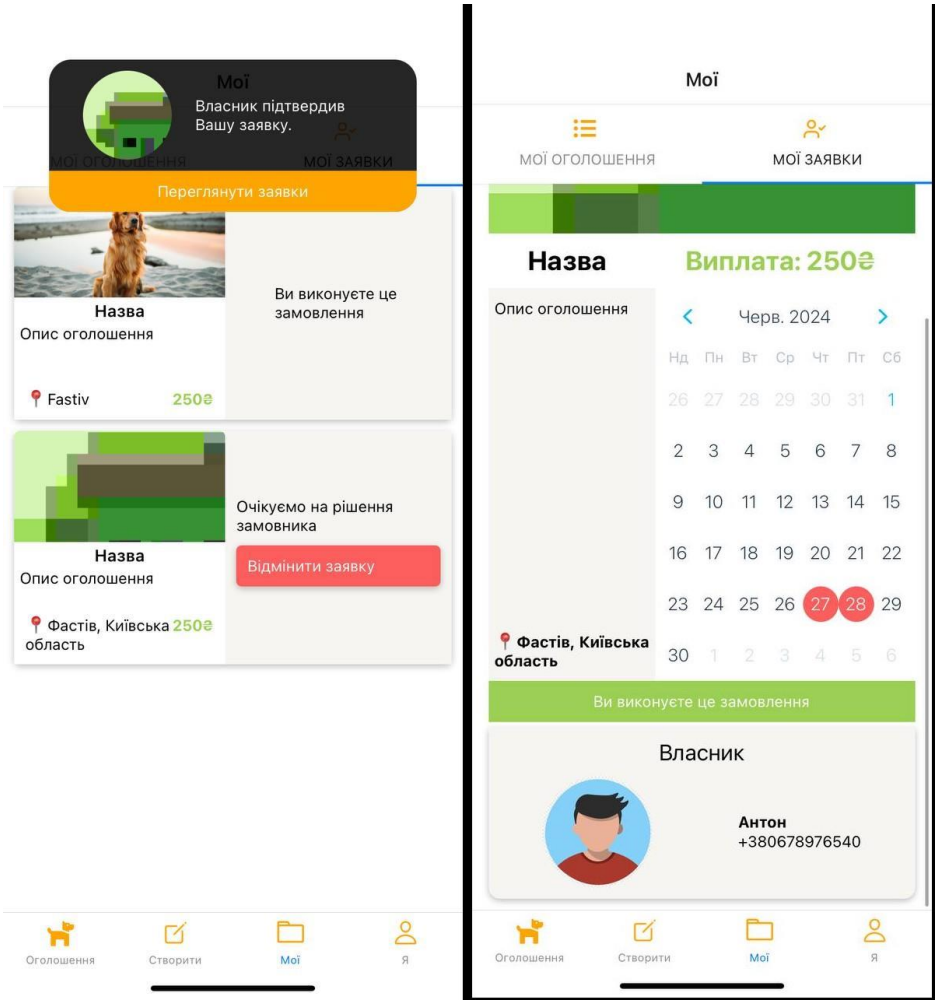


Рисунок 4.10 – Користувач Даніл отримав сповіщення про підтвердження його заявки

Даніла обрано до виконання оголошення. Усі інші заявки, що було подано на це оголошення, відхиляються та користувачі отримують сповіщення про це. Згодом користувач Антон може позначити оголошення як виконане та видалити його.

Висновки до розділу

У рамках даного розділу було проаналізовано якість кодової бази back-end частини застосунку. Було обрано та описано метрики та методи, що використовувались для цього.

Отримані показники з кожної з метрик є хорошим, що свідчить про добре розроблений код, який можна модифікувати та розширювати в

майбутньому. Досягається це завдяки реалізованим патернам та підходам при розробці.

Описано процес мануального тестування програмного забезпечення з деталізацією тестів. У загальному оглянуто процес модульного тестування основних реалізованих сервісів та підходи щодо написання тестів. Проаналізовано результати.

Наведено контрольний приклад, у якому детально описано процес основного функціоналу ПЗ, а саме створення оголошення та підбору виконавців. Наведено ілюстрації, що доводять коректність роботи.

5 РОЗГОРТАННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

5.1 Розгортання програмного забезпечення

При розгортанні програмного забезпечення слід окремо розглядати клієнтську та серверну частини застосунку. Серверна частина дипломного проекту це ASP.NET Web API застосунок, що має розміщатись на хмарних або фізичних потужностях.

Задля реалізації зручного розгортання, поширення, абстрагування від операційної системи, можливо загорнути back-end частину в Linux-контейнер [38]. Завдяки цьому відбувається віртуалізація на рівні операційної системи.

Одним з найбільш популярних інструментів. Що дозволяють працювати з Linux-контейнерами, є Docker. Також необхідно розгорнути базу даних. Задля зручного одночасного запуску було створено docker-compose файл, що описує інструкції щодо взаємодії між контейнерами. І було написано окремий Dockerfile для опису інструкцій щодо побудови контейнера .NET застосунку. Вміст Dockerfile представлено на рисунку 5.1.

```
FROM mcr.microsoft.com/dotnet/aspnet:6.0 AS base
WORKDIR /app
EXPOSE 80
EXPOSE 443

FROM mcr.microsoft.com/dotnet/sdk:6.0 AS build
WORKDIR /src
COPY ["backendPetHome/backendPetHome.API.csproj", "backendPetHome/"]
COPY ["BAL/backendPetHome.BLL.csproj", "BAL/"]
COPY ["DAL/backendPetHome.DAL.csproj", "DAL/"]
RUN dotnet restore "backendPetHome/backendPetHome.API.csproj"
COPY . .
WORKDIR "/src/backendPetHome"
RUN dotnet build "backendPetHome.API.csproj" -c Release -o /app/build

FROM build AS publish
RUN dotnet publish "backendPetHome.API.csproj" -c Release -o /app/publish

FROM base AS final
WORKDIR /app
COPY ["DAL/Data/FunctionsUp/", "DAL/Data/FunctionsUp/"]
COPY ["DAL/Data/FunctionsDown/", "DAL/Data/FunctionsDown/"]
COPY --from=publish /app/publish .
ENTRYPOINT ["dotnet", "backendPetHome.API.dll"]

RUN apt-get clean && apt-get update && apt-get install -y locales
RUN locale-gen uk_UA.UTF-8
```

Рисунок 5.1 – Вміст Dockerfile

Спочатку копіюються локальні файли проекту. Відбувається їх збірка. Копіюються SQL файли з директорій “DAL/Data/FunctionsUp” та “DAL/Data/FunctionsDown”, що містять SQL функції, що використовуються для більш швидкого підбору можливих виконавців та фільтрації оголошень. Також встановлюються налаштування локалізації.

Вміст файлу docker-compose представлено на рисунку 5.2.

```yaml
version: "3.4"

services:
  backendpethomeapi:
    image: backendpethomeapi
    restart: on-failure
    build:
      context: .
      dockerfile: backendPetHome/Dockerfile
    networks:
      - back
    ports:
      - "8080:80"
    depends_on:
      - petHomeDB
    environment:
      ConnectionStrings__DefaultConnection: "server=petHomeDB;database=testdb;User=sa;password=Password123!"
      LANG: uk_UA.UTF-8
      LANGUAGE: uk_UA
      LC_ALL: uk_UA.UTF-8

  petHomeDB:
    image: "mcr.microsoft.com/mssql/server:2022-latest"
    ports:
      - "1433:1433"
    networks:
      - back
    environment:
      SA_PASSWORD: "Password123!"
      ACCEPT_EULA: "Y"

networks:
  back:
    driver: bridge
```

Рисунок 5.2 – Вміст файлу docker-compose.yml

В ньому налаштовуються сервіси backendpethomeapi та petHomeDB, порти, що використовуються ними, мережа між ними.

Після виконання команди docker compose up, запускаються обидва сервіси, об’єднані в одну мережу (рисунок 5.3).

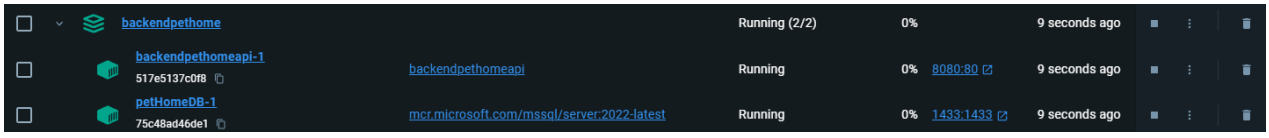


Рисунок 5.3 – Робота сервісів після виконання команди docker compose up

За потреби імеджі Docker можна розміщати в Docker Hub. У такому випадку інші користувачі зможуть завантажувати їх та розгортати локально. У випадку завантаження застосунку в App Store, або Play Market, потрібно розміщувати back-end частину застосунку на сервері, щоб клієнтські RN застосунки мали доступ до неї.

Щодо клієнтської частини. RN застосунок розміщується на платформах App Store, у випадку iOS, та Play Market, у випадку Android. Для цього потрібно зібрати необхідні файли.

Оскільки застосунок в основному розроблявся для операційної системи iOS, його тестування відбувалось на iPhone. Пристрій, на якому розроблено застосунок, має операційну систему Windows. Тому для роботи iPhone було використано фреймворк Expo [39]. Він також надає можливість формування файлів, що необхідні для розміщення у крамницях застосунків.

Сформувати їх можна за допомогою виконання команд «eas build --platform android» та «eas build --platform ios» в терміналі. У випадку Android, формується файл розширення .ABB, що розміщується в Play Market (рисунок 5.4).

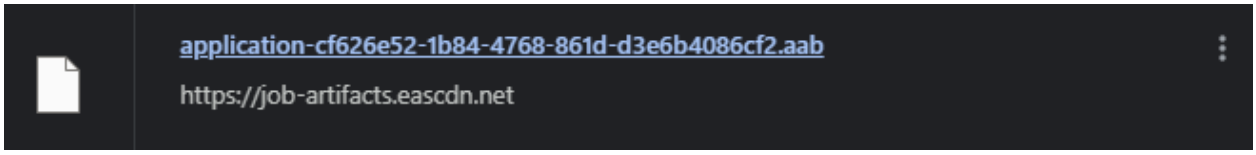


Рисунок 5.4 – Файл розширення .aab, необхідний для розміщення застосунку у Play Market

Результатом виконання команди є сформована збірка(рисунок 5.5)

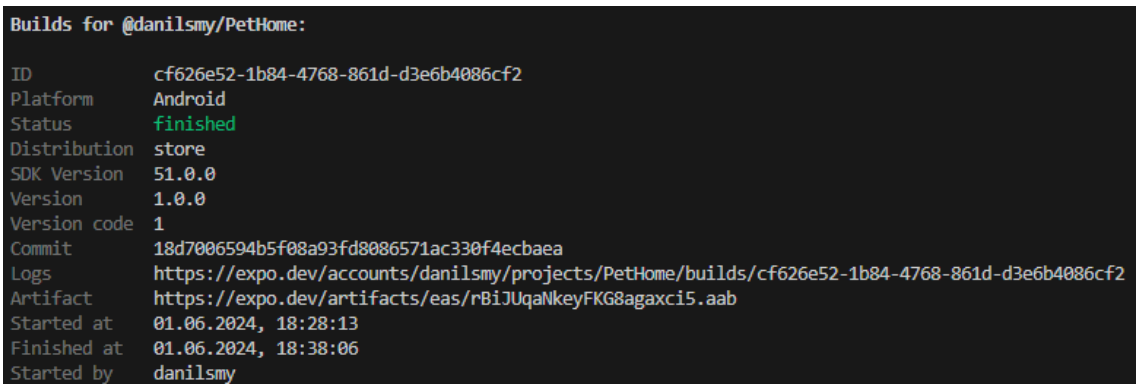


Рисунок 5.5 – Результат виконання команди «eas build --platform android»

У випадку iOS, для формування таких файлів необхідно мати акаунт розробника. Тому, на жаль, не вдалось, виконати збірку (рисунок 5.6).

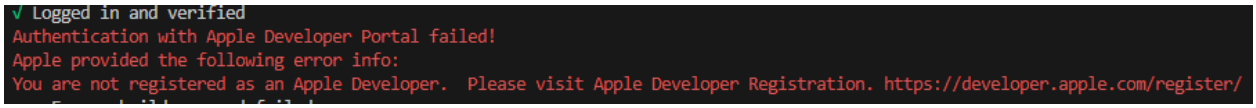


Рисунок 5.6 – Відмова у зв’язку з відсутністю статусу розробника Apple

Загалом побудовані файли можна розмістити в крамницях застосунків і користувачі зможуть завантажувати PetHome на мобільні пристрої.

5.2 Супровід програмного забезпечення

Інструкція користувача наведена в окремому документі.

Користувачі застосунку мають отримувати оновлення.

Задля оновлення back-end частини застосунку було реалізовано github workflow, що дозволяє автоматично при завантаженні оновлень на гілку master збирати імедж та оновлювати його у Docker Hub. Зміст файлу docker.yml наведено на рисунку 5.7.

docker.yml

```yaml
name: Docker images
on:
  push:
    branches: [master]
jobs:
  build-docker-images:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3

      - name: Set up Docker Buildx
        uses: docker/setup-buildx-action@v2

      - name: Login to DockerHub
        uses: docker/login-action@v2
        with:
          username: ${{ secrets.DOCKERHUB_USERNAME }}
          password: ${{ secrets.DOCKERHUB_TOKEN }}

      - name: Checkout
        uses: actions/checkout@v2

      - name: Build and push backend docker image
        uses: docker/build-push-action@v3
        with:
          context: ./backendPetHome
          file: ./backendPetHome/backendPetHome/Dockerfile
          tags: ${{ secrets.DOCKERHUB_USERNAME }}/backend-pet-home:latest
          push: true
          build-args: |
            PROJECT_PORT=80

```

Рисунок 5.7 – Зміст файлу docker.yml

Для авторизації в Docker Hub використовуються дані, що збережені як secrets у github репозиторії. Відбувається збірка імеджу та оновлення його у Docker Hub (рисунок 5.8).

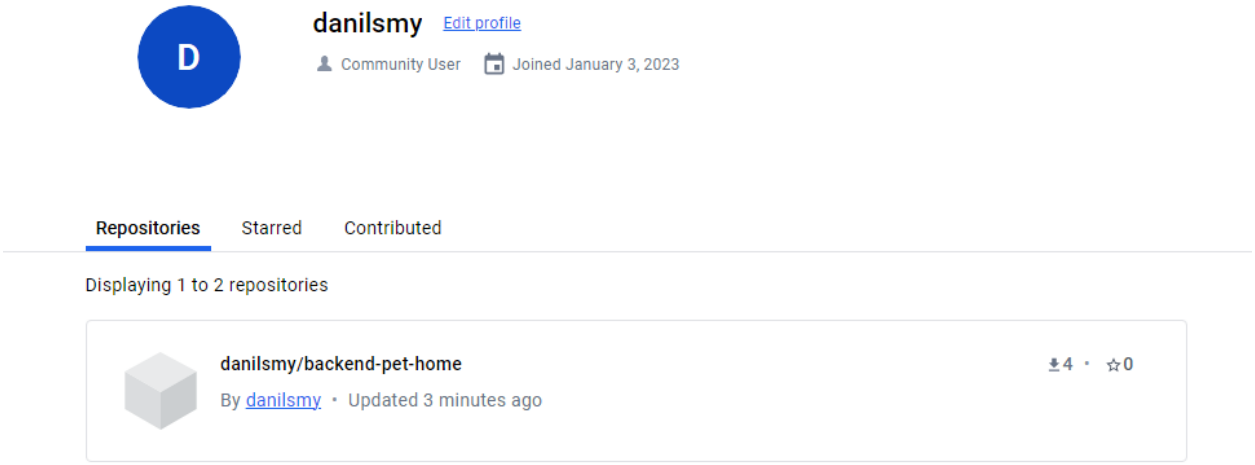


Рисунок 5.8 – Оновлений репозиторій в Docker Hub

Щодо клієнтської частини застосунку, за допомогою expo можна створювати нові збірки та публікувати їх у крамниці застосунків. Користувачам необхідно завантажити нове оновлення на мобільний пристрій.

Висновки до розділу

У рамках даного розділу було детально описано процес розгортання серверної та клієнтської частин розробленого програмного забезпечення. Розглянуто процеси збірки та аргументовано вибір інструментів для впровадження. Завдяки обраним підходам є можливість зручного розгортання ПЗ на сервері та публікації його в крамниці застосунків.

Розглянуто методи, що використовуються для супроводу програмного забезпечення, а саме для зручного оновлення клієнтської та серверної частин. Завдяки обраним підходам користувач може легко отримувати актуальну версію застосунку.

ВИСНОВКИ

Метою даного дипломного проекту є полегшення процесу пошуку виконавців та замовників з формуванням конкретних релевантних пропозицій, враховуючи задані критерії, та часткова автоматизація процесу підбору можливих виконавців.

Для досягнення мети було повною мірою реалізовано сформовані функціональні задачі.

Завдяки частковій автоматизації процесу пошуку виконавців, досягається вища швидкість пошуку виконавців та замовників послуг. При створенні оголошень система автоматично генерує пропозиції можливим виконавцям, враховуючи їх місцеположення та графік завантаженості.

Функціонал фільтрації оголошень, що при підборі може враховувати завантажені дати виконавця, допомагає користувачам знаходити релевантні пропозиції, у яких вони можуть бути зацікавлені.

Завдяки сповіщенням в режимі реального часу, користувачі можуть швидко реагувати на подані заявки, на нові оголошення, на зміну статусу заявки.

Функціонал адміністрування дозволяє модерувати користувачів та оголошення, щоб в застосунку не було забороненого контенту.

Завдяки функціоналу роботи з графіком навантаженості, користувачі можуть гнучко налаштовувати завантажені дні, що враховуються при підборі можливих виконавців системою.

У майбутньому є можливість розширити функціонал додаванням автоматичної роботи з графіком завантаженості користувача у випадку, якщо виконавець може перетримувати одночасно кілька тварин у один день, додаванням чату між виконавцем та замовником, додаванням системи відгуків та оцінок користувачів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1) Pet sitting market report - forecast analysis 2023-2032 by the brainy insights. URL: https://www.thebrainyinsights.com/report/pet-sitting-market-13676 (дата звернення 07.04.2024).

2) Мобільні застосунки: їх види та особливості. URL: https://highload.today/uk/mobilni-zastosunki-yih-vidi-ta-osoblivosti/ (дата звернення 07.04.2024).

3) Annual number of mobile app downloads worldwide 2023. URL: https://www.statista.com/statistics/271644/worldwide-free-and-paid-mobile-app-store-downloads/ (дата звернення 08.04.2024).

4) Hi Paw. Walking pets. *App Store*. URL: https://apps.apple.com/pl/app/hi-paw-walking-pets/id6450741010 (дата звернення 08.04.2024).

5) Перетримка тварин у місті – Київ. *PetLive*. . URL: https://petlive.com.ua/ (дата звернення 08.04.2024).

6) Bonaventure O. Computer Networking: Principles, Protocols and Practice. Louvain-la-Neuve: Université catholique de Louvain, 2011. C. 27-32.

7) Mobile App Architecture: A Comprehensive Guide 2023. *Intellectsoft Blog*. URL: https://www.intellectsoft.net/blog/mobile-app-architecture/(дата звернення 12.04.2024).

8) Що таке PWA додатки. *avada-media*. URL: https://avada-media.ua/ua/services/pwa/. (дата зверення 14.04.2024).

9) Native vs. hybrid vs. cross-platform: which best. *InspiringApps*. URL: https://www.inspiringapps.com/blog/120/app-development-native-vs-hybrid-vs-cross-platform/ (дата звернення 16.04.2024).

10) Mobile App Development: Native, Hybrid, and React Native. *Lithios Apps*. URL: https://www.lithiosapps.com/blog/mobile-app-development-native-hybrid-and-react-native (дата звернення 20.04.2024).

11) Monoliths vs Microservices vs Serverless. *Split*. URL: https://www.split.io/blog/monoliths-vs-microservices-vs-serverless/ (дата звернення 26.04.2024).

12) Chinta M. Code in Clean vs. Traditional Layered architecture. URL: https://medium.com/codenx/code-in-clean-vs-traditional-layered-architecture-net-31c4cad8f815 (дата звернення 30.04.2024).

13) Main differences between Clean Architecture and Hexagonal Architecture. URL: https://www.gregorypacheco.com.br/posts/differences-between-clean-architecture-and-hexagonal-architecture-cshrp-dot-net.html (дата звернення 01.05.2024).

14) 3 Layered Architecture. *Canarys*. URL: https://ecanarys.com/3-layered-architecture/ (дата звернення 01.05.2024).

15) Repository GitHub facebook/react-native: A framework for building native applications using React. URL: https://github.com/facebook/react-native#-requirements (дата звернення 02.05.2024).

16) The new React Native architecture explained. URL: https://commerce.nearform.com/blog/2019/lean-core-part-4/ (дата звернення 03.05.2024).

17) How to Implement 3 Layered Architecture Concepts in ASP.Net. URL: https://www.c-sharpcorner.com/UploadFile/7be47b/how-to-implement-3-tier-architecture-concepts-in-Asp-Net/ (дата звернення 04.05.2024).

18) Implementing the Repository and Unit of Work Patterns in an ASP.NET MVC Application. *Microsoft Learn:* URL: https://learn.microsoft.com/en-us/aspnet/mvc/overview/older-versions/getting-started-with-ef-5-using-mvc-4/implementing-the-repository-and-unit-of-work-patterns-in-an-asp-net-mvc-application (дата звернення 04.05.2024).

19) Mastering the Specification Pattern in .NET Core. *Medium*. URL: https://blog.stackademic.com/mastering-the-specification-pattern-in-net-core-ad847339fbf5 (дата звернення 04.05.2024).

20) Flutter vs React Native: Which One is the Best Framework? *Radixweb*. URL: https://radixweb.com/blog/flutter-vs-react-native (дата звернення 04.05.2024).

21) TypeScript vs JavaScript: Which One Is Better to Choose?. *Radixweb*. URL: https://radixweb.com/blog/typescript-vs-javascript (дата звернення 04.05.2024).

22) ASP.NET overview . *Microsoft Learn:* URL: https://learn.microsoft.com/uk-ua/aspnet/overview (дата звернення 05.05.2024).

23) Introduction to SignalR. *Microsoft Learn:* URL: https://learn.microsoft.com/uk-ua/aspnet/signalr/overview/getting-started/introduction-to-signalr (дата звернення 06.05.2024).

24) Pushing Messages from Server to Client Using SignalR and URL: https://thundaxsoftware.blogspot.com/2017/05/pushing-messages-from-server-to-client.html (дата звернення 06.05.2024).

25) Getting Started - EF Core. *Microsoft Learn:* URL: https://learn.microsoft.com/uk-ua/ef/core/get-started/overview/first-app?tabs=netcore-cli (дата звернення 06.05.2024).

26) What is ORM? Why is it used? What are its pros and cons?. *Medium*. URL: https://medium.com/@kadergenc/what-is-orm-why-is-it-used-what-are-its-pros-and-cons-3ed77c0e6ed2 (дата звернення 08.05.2024).

27) Overview of Places API. *Google for Developers*. URL: https://developers.google.com/maps/documentation/places/web-service/overview (дата звернення 08.05.2024).

28) Geocoding API overview. *Google for Developers*. URL: https://developers.google.com/maps/documentation/geocoding/overview (дата звернення 10.05.2024).

29) JSON Web Tokens Introduction. Documentation. *jwt.io*. URL: https://jwt.io/introduction (дата звернення 10.05.2024).

30) What Are Access and Refresh Tokens? URL: https://www.baeldung.com/cs/access-refresh-tokens (дата звернення 11.05.2024).

31) Що таке XSS атака як вона працює? URL: https://ukeywaf.com/baza/shho-take-xss-ataka-yak-vona-praczyuye1/ (дата звернення 11.05.2024).

32) Cross-Origin Resource Sharing (CORS). *MDN Web Docs*. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS (дата звернення 11.05.2024).

33) How code metrics help identify risks - Visual Studio. *Microsoft Learn*. URL: https://learn.microsoft.com/en-us/visualstudio/code-quality/code-metrics-values?view=vs-2022 (дата звернення 14.05.2024).

34) Find and fix problems in JavaScript code. *ESLint*. URL: https://eslint.org/ (дата звернення 16.05.2024).

35) Unit testing using xUnit. URL: https://xunit.net/ (дата звернення 18.05.2024).

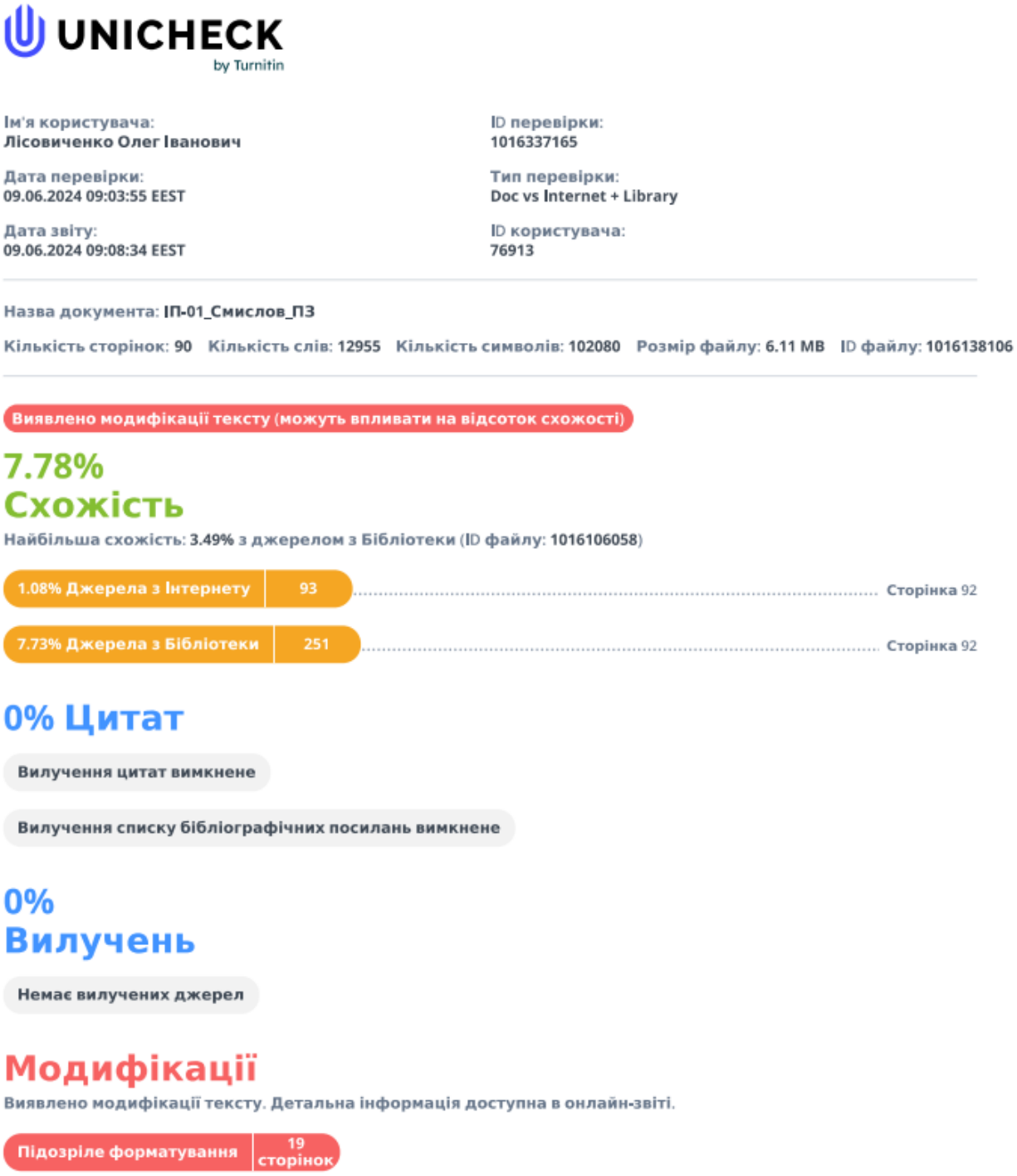
36) The AAA Structure of Tests. *Thinkster*. URL: https://thinkster.io/tutorials/blogs/the-aaa-structure-of-tests (дата звернення 18.05.2024).

37) AutoMapper & Data Transfer Objects (DTOs) in .NET 6. URL: https://dev.to/moe23/net-6-automapper-data-transfer-objects-dtos-49e (дата звернення: 22.05.2024).

38) Linux Containers overview. URL: https://linuxcontainers.org/ (дата звернення 24.05.2024).

39) Expo – framework for building native apps. Documentation. URL: https://expo.dev/ (дата звернення 28.05.2024).

ДОДАТОК А – ПЕРЕВІРКА НА СПІВПАДІННЯ



UNICHECK
by Turnitin

Ім'я користувача:
Лісовиченко Олег Іванович

ID перевірки:
1016337165

Дата перевірки:
09.06.2024 09:03:55 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
09.06.2024 09:08:34 EEST

ID користувача:
76913

Назва документа: ІП-01_Смислов_ПЗ

Кількість сторінок: 90 Кількість слів: 12955 Кількість символів: 102080 Розмір файлу: 6.11 MB ID файлу: 1016138106

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

7.78%
Схожість

Найбільша схожість: 3.49% з джерелом з Бібліотеки (ID файлу: 1016106058)

1.08% Джерела з Інтернету 93 ..... Сторінка 92

7.73% Джерела з Бібліотеки 251 ..... Сторінка 92

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Підозріле форматування 19 сторінок

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2024 р.

Мобільний застосунок для пошуку виконавців та замовників послуг на прикладі догляду за тваринами

Текст програми

КПІ.ІП-0127.045490.03.12

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Олександр ПАВЛОВ

Нормоконтроль:
_____ Максим ГОЛОВЧЕНКО

Виконавець:
_____ Даніл СМИСЛОВ

Київ – 2024

Посилання на репозиторій з повним текстом програмного коду
https://github.com/ErnestoFolting/PetHomeMobile

Файл AdvertsController.cs
Реалізація функціональної вимоги "часткова автоматизація процесу пошуку виконавців, шляхом генерації пропозицій можливим виконавцям, що задовольняють обмеженням"

```
using backendPetHome.API.Attributes;
using backendPetHome.API.Controllers.Abstract;
using backendPetHome.API.Hubs;
using backendPetHome.BLL.DTOs.AdvertDTOs;
using backendPetHome.BLL.Services.Interfaces;
using backendPetHome.DAL.Specifications.QueryParameters;
using Microsoft.AspNet.SignalR;
using Microsoft.AspNetCore.Mvc;
using Newtonsoft.Json;

namespace backendPetHome.API.Controllers
{
    [Route("api/[controller]")]
    public class AdvertsController : BaseController
    {
        private readonly IAdvertService _advertService;
        private readonly PerformerSelectionHub _hub;
        public AdvertsController(IAdvertService
advertService, PerformerSelectionHub hub)
        {
            _advertService = advertService;
            _hub = hub;
        }

        [HttpGet]
        public async
Task<ActionResult<IEnumerable<AdvertDTO>>> GetAdverts([FromQuery]
QueryStringParameters parameters)
        {
            var advertsAndCount = await
_advertService.getAdverts(parameters, UserId);
            Response.Headers.Add("X-Pagination-Total-Count",
JsonConvert.SerializeObject(advertsAndCount.totalCount));
            return Ok(advertsAndCount.fitAdvertsDTO);
        }
        [Authorize(Roles = "Administrator")]
```

```
            [HttpGet("byadmin")]
            public                                          async
Task<ActionResult<IEnumerable<AdvertDTO>>>
GetAdvertsByAdmin([FromQuery] QueryStringParameters parameters)
            {
                var        advertsAndCount        =        await
_advertService.getAdverts(parameters);
                Response.Headers.Add("X-Pagination-Total-Count",
JsonConvert.SerializeObject(advertsAndCount.totalCount));
                return Ok(advertsAndCount.fitAdvertsDTO);
            }

            [HttpGet("{id}")]
            public  async  Task<ActionResult<AdvertDTO>>  Get(int
id)
            {
                return                                    Ok(await
_advertService.getAdvertById(id));
            }

            [HttpPost]
            public   async   Task<ActionResult>   Post([FromForm]
AdvertCreateRedoDTO advertToAdd, IFormFile petPhoto)
            {
                var       possiblePerformers       =       await
_advertService.addAdvert(advertToAdd, UserId, petPhoto);
                if  (possiblePerformers.possiblePerformersIds  !=
null)  await  _hub.Send(possiblePerformers.possiblePerformersIds,
possiblePerformers.advertDTO);
                return Ok();
            }

            [HttpPut("finish/{advertId}")]
            public  async  Task<ActionResult>  MarkAsFinished(int
advertId)
            {
                await    _advertService.MarkAsFinished(advertId,
UserId);
                return Ok();
            }

            [HttpDelete("{advertId}")]
            public  async  Task<ActionResult>  deleteAdvert(int
advertId)
            {
                await      _advertService.deleteAdvert(advertId,
UserId);
```

```
                return Ok();
            }

            [Authorize(Roles ="Administrator")] //can be more
complex moderating logic
            [HttpDelete("adminDelete/{advertId}")]
            public async Task<ActionResult>
deleteAdvertByAdmin(int advertId)
            {
                await _advertService.deleteAdvert(advertId);
                return Ok();
            }
        }
    }
```

Файл IPerformerSelectionHub.cs

Реалізація функціональної вимоги "часткова автоматизація процесу пошуку виконавців, шляхом генерації пропозицій можливим виконавцям, що задовольняють обмеженням"

```
    using backendPetHome.BLL.DTOs.AdvertDTOs;
    using backendPetHome.BLL.DTOs.RequestDTOs;

    namespace backendPetHome.API.Hubs
    {
        public interface IPerformerSelectionHub
        {
            Task Send(IEnumerable<string> possiblePerformerIds,
AdvertDTO advertToSend);
            Task ApplyRequest(RequestDTO requestDTO);
            Task DeleteRequest(RequestDTO requestDTO);
            Task ConfirmRequest(List<RequestDTO>
requestsToReject, RequestDTO requestToConfirm);
            Task RejectRequest(RequestDTO requestToReject);
        }
    }
```

Файл PerformerSelectionHub.cs

Реалізація функціональної вимоги "часткова автоматизація процесу пошуку виконавців, шляхом генерації пропозицій можливим виконавцям, що задовольняють обмеженням"

```
    using backendPetHome.BLL.DTOs.AdvertDTOs;
    using backendPetHome.BLL.DTOs.RequestDTOs;
    using Microsoft.AspNetCore.Authorization;
    using Microsoft.AspNetCore.SignalR;
```

```
namespace backendPetHome.API.Hubs
{
    [Authorize]
    public class PerformerSelectionHub : Hub,
IPerformerSelectionHub
    {
        public async Task Send(IEnumerable<string>
possiblePerformerIds, AdvertDTO advertToSend)
        {
            await
Clients.Users(possiblePerformerIds).SendAsync("Send",
advertToSend);
        }

        public async Task ApplyRequest(RequestDTO requestDTO)
        {
            string? ownerId = requestDTO.advert.ownerId;
            if (ownerId != null)
            {
                await
Clients.User(ownerId).SendAsync("Apply", requestDTO);
            }
        }

        public async Task DeleteRequest(RequestDTO
requestDTO)
        {
            string? ownerId = requestDTO.advert.ownerId;
            if (ownerId != null &&
                requestDTO.advert.status !=
DAL.Enums.AdvertStatusEnum.finished &&
                requestDTO.status !=
DAL.Enums.RequestStatusEnum.rejected
                )
            {
                await
Clients.User(ownerId).SendAsync("Delete", requestDTO);
            }
        }

        public async Task ConfirmRequest(List<RequestDTO>
requestsToReject, RequestDTO requestToConfirm)
        {
            List<string> applierIdsToRejectNotify = new();
            requestsToReject.ForEach(el =>
applierIdsToRejectNotify.Add(el.userId));
```

```
                await
Clients.Users(applierIdsToRejectNotify).SendAsync("Reject",
requestToConfirm);
                await
Clients.User(requestToConfirm.userId).SendAsync("Confirm",
requestToConfirm);
            }

            public    async    Task    RejectRequest(RequestDTO
requestToReject)
            {
                string applierId = requestToReject.userId;
                await
Clients.User(applierId).SendAsync("Reject", requestToReject);
            }
        }
    }
```

Файл distanceBetweenPoints.sql

Реалізація функціональної вимоги "часткова автоматизація процесу пошуку виконавців, шляхом генерації пропозицій можливим виконавцям, що задовольняють обмеженням"

```
use testdb;
go
CREATE FUNCTION DistanceBetweenPlaces (
    @lon1 float,
    @lat1 float,
    @lon2 float,
    @lat2 float
)
RETURNS float
AS
BEGIN
    DECLARE @dlon float = RADIANS(@lon2 - @lon1);
    DECLARE @dlat float = RADIANS(@lat2 - @lat1);

    DECLARE @a float = SIN(@dlat / 2) * SIN(@dlat / 2) +
COS(RADIANS(@lat1)) * COS(RADIANS(@lat2)) * (SIN(@dlon / 2) *
SIN(@dlon / 2));
    DECLARE @angle float = 2 * ATN2(SQRT(@a), SQRT(1 - @a));
    RETURN @angle * 6378.16;
END;
```

Файл getPossiblePerformers.sql

Реалізація функціональної вимоги "часткова автоматизація процесу пошуку виконавців, шляхом генерації пропозицій можливим виконавцям, що задовольняють обмеженням"

```
use testdb;

go
CREATE FUNCTION selectPossiblePerformers (
    @advertStartTime dateTime,
    @advertEndTime dateTime,
@advertLocationLng float,
@advertLocationLat float,
@ownerId nvarchar(450)
)
RETURNS @UserIds table(id nvarchar (450))
AS
BEGIN
INSERT INTO @UserIds
SELECT Id
FROM AspNetUsers
WHERE NOT EXISTS (
    SELECT *
    FROM timeExceptions
    WHERE userId = AspNetUsers.Id
    AND date >= @advertStartTime
    AND date <= @advertEndTime
)
AND Id <> @ownerId
AND     [dbo].DistanceBetweenPlaces(AspNetUsers.locationLng,
AspNetUsers.locationLat, @advertLocationLng, @advertLocationLat)
< 30
RETURN ;
END
go
```

Файл AdvertService.cs

Реалізація функціональної вимоги "часткова автоматизація процесу пошуку виконавців, шляхом генерації пропозицій можливим виконавцям, що задовольняють обмеженням"

```
using AutoMapper;
using backendPetHome.BLL.DTOs.AdvertDTOs;
using backendPetHome.BLL.Services.Abstract;
using backendPetHome.BLL.Services.Interfaces;
using backendPetHome.DAL.Entities;
```

```
using backendPetHome.DAL.Enums;
using backendPetHome.DAL.Interfaces;
using
backendPetHome.DAL.Specifications.AdvertSpecifications;
using backendPetHome.DAL.Specifications.QueryParameters;
using Microsoft.AspNetCore.Http;

namespace backendPetHome.BLL.Services
{
    public class AdvertService : BaseService, IAdvertService
    {
        private readonly IRequestService _requestService;

        public     AdvertService(IUnitOfWork     unitOfWork,
IRequestService requestService, IMapper mapper)
            :base(unitOfWork,mapper)
        {
            _requestService = requestService;
        }
        public async Task<(List<AdvertDTO> fitAdvertsDTO, int
totalCount)>  getAdverts(QueryStringParameters  parameters,string
userId)
        {
            var          fitAdverts          =          await
_unitOfWork.AdvertRepository.GetBySpecificationAndPaging(new
AdvertWithParamsAndPaginationSpecification(userId, parameters));
            List<AdvertDTO>           advertsDTO           =
_mapper.Map<List<AdvertDTO>>(fitAdverts.fitAdverts);
            return (advertsDTO, fitAdverts.totalCount);
        }

        public async Task<(List<AdvertDTO> fitAdvertsDTO, int
totalCount)> getAdverts(QueryStringParameters parameters)
        {
            var          fitAdverts          =          await
_unitOfWork.AdvertRepository.GetBySpecificationAndPaging(new
AllAdvertsWithPaginationSpecification(parameters));
            List<AdvertDTO>           advertsDTO           =
_mapper.Map<List<AdvertDTO>>(fitAdverts.fitAdverts);
            return (advertsDTO, fitAdverts.totalCount);
        }

        public   async   Task<AdvertDTO>   getAdvertById(int
advertId)
        {
```

```
                Advert? advert = await
_unitOfWork.AdvertRepository.GetByIdSpecification(new
AdvertByIdWithOwnerSpecification(advertId));
                if (advert == null) throw new
KeyNotFoundException("Advert not found.");

                AdvertDTO advertDTO =
_mapper.Map<AdvertDTO>(advert);
                return advertDTO;
            }

            public async Task<(IEnumerable<string>
possiblePerformersIds,AdvertDTO advertDTO)>
addAdvert(AdvertCreateRedoDTO advertToAdd, string userId,
IFormFile advertFile)
            {
                Advert newAdvert =
_mapper.Map<Advert>(advertToAdd);
                newAdvert.photoFilePath = "/images/" +
advertFile.FileName;
                newAdvert.ownerId = userId;
                await
_unitOfWork.AdvertRepository.Add(newAdvert);
                await
_unitOfWork.FileRepository.Add(advertFile);
                await _unitOfWork.SaveChangesAsync();

                IEnumerable<string> possiblePerformersIds =
await
_unitOfWork.UserRepository.SelectPossiblePerformers(newAdvert,
userId);
                foreach(string possiblePerformerId in
possiblePerformersIds)
                {
                    await
_requestService.addRequest(possiblePerformerId, newAdvert.Id,
RequestStatusEnum.generated);
                }
                await _unitOfWork.SaveChangesAsync();

                AdvertDTO advertDTO =
_mapper.Map<AdvertDTO>(newAdvert);
                return (possiblePerformersIds, advertDTO);
            }
            public async Task MarkAsFinished(int advertId, string
userId) {
```

```
                var advertInDb = await
_unitOfWork.AdvertRepository.GetByIdSpecification(new
AdvertByIdSpecification(advertId));
                if (advertInDb == null) throw new
KeyNotFoundException("Advert not found.");
                if (advertInDb.ownerId!= userId) throw new
ArgumentException("You do not have the access.");

                advertInDb.status = AdvertStatusEnum.finished;
                await
_unitOfWork.AdvertRepository.Update(advertInDb);
                await _unitOfWork.SaveChangesAsync();
            }
            public async Task deleteAdvert(int advertId, string
userId)
            {
                var advertInDb = await
_unitOfWork.AdvertRepository.GetByIdSpecification(new
AdvertByIdIncludesRequestAndUserSpecification(advertId));
                if (advertInDb == null) throw new
KeyNotFoundException("Advert not found.");
                if (advertInDb.ownerId != userId) throw new
ArgumentException("You do not have the access.");
                if (advertInDb.requests.Any(el => el.status !=
RequestStatusEnum.rejected && el.status
!=RequestStatusEnum.generated) && advertInDb.status !=
AdvertStatusEnum.finished) throw new ArgumentException("There are
requests on advert. Delete them or finish advert before
deleting.");

                await
_unitOfWork.AdvertRepository.Delete(advertInDb);
                await _unitOfWork.SaveChangesAsync();
            }
            public async Task deleteAdvert(int advertId)
            {
                var advertInDb = await
_unitOfWork.AdvertRepository.GetByIdSpecification(new
AdvertByIdSpecification(advertId));
                if (advertInDb == null) throw new
KeyNotFoundException("Advert not found.");

                await
_unitOfWork.AdvertRepository.Delete(advertInDb);
                await _unitOfWork.SaveChangesAsync();
            }
        }
```

```
    }
```

Файл TimeExceptionService.cs

Реалізація функціональної вимоги "часткова автоматизація процесу пошуку виконавців, шляхом генерації пропозицій можливим виконавцям, що задовольняють обмеженням"

```
    using AutoMapper;
    using backendPetHome.BLL.Services.Abstract;
    using backendPetHome.BLL.Services.Interfaces;
    using backendPetHome.DAL.Entities;
    using backendPetHome.DAL.Interfaces;
    using
backendPetHome.DAL.Specifications.TimeExceptionSpecifications;

    namespace backendPetHome.BLL.Services
    {
        public class TimeExceptionService : BaseService,
ITimeExceptionService
        {
            public TimeExceptionService(IUnitOfWork unitOfWork,
IMapper mapper) : base(unitOfWork, mapper)
            {
            }
            public async Task addTimeExceptions(string userId,
IEnumerable<DateTime> dates)
            {
                var userExceptions = await _unitOfWork
                    .TimeExceptionRepository
                    .GetBySpecification(new
TimeExceptionCurrentUserSpecification(userId));

                foreach (var date in dates)
                {
                    TimeException exception = new();
                    exception.userId = userId;
                    exception.date = date;
                    if (userExceptions.FindIndex(el => el.date ==
exception.date) == -1)
                    {
                        await
_unitOfWork.TimeExceptionRepository.Add(exception);
                    }
                }

                await _unitOfWork.SaveChangesAsync();
```

```
            }

            public async Task deleteTimeExceptions(string userId,
IEnumerable<DateTime> datesToRemove)
            {
                var userExceptions = await _unitOfWork
                    .TimeExceptionRepository
                    .GetBySpecification(new
TimeExceptionCurrentUserSpecification(userId));

                foreach (var userException in userExceptions)
                {
                    if (datesToRemove.Any(dateToRemove =>
                    dateToRemove.Year == userException.date.Year
&&
                    dateToRemove.Month                          ==
userException.date.Month &&
                    dateToRemove.Day == userException.date.Day))
                    {
                        await
_unitOfWork.TimeExceptionRepository.Delete(userException);
                    }
                }

                await _unitOfWork.SaveChangesAsync();
            }

            public  async  Task<bool>  checkPerformerDates(string
userId, DateTime advertStart, DateTime advertEnd)
            {
                List<TimeException> fitExceptions = await
                    _unitOfWork
                    .TimeExceptionRepository
                    .GetBySpecification(new
TimeExceptionCurrentUserFitAdvertTimeSpecification(userId,
advertStart, advertEnd));
                return !fitExceptions.Any();
            }
        }
    }
```

Файл Advert.cs

Реалізація функціональної вимоги "часткова автоматизація процесу пошуку виконавців, шляхом генерації пропозицій можливим виконавцям, що задовольняють обмеженням"

```
    using backendPetHome.DAL.Entities.Abstract;
```

```
using backendPetHome.DAL.Enums;
using System.Text.Json.Serialization;

namespace backendPetHome.DAL.Entities
{
    public class Advert : BaseEntity
    {
        public int Id { get; set; }
        public string name { get; set; } = string.Empty;
        public int cost { get; set; }
        public string location { get; set; } = string.Empty;
        public double locationLat { get; set; } = 0;
        public double locationLng { get; set; } = 0;
        public string description { get; set; } = string.Empty;
        public string photoFilePath { get; set; } = string.Empty;
        [JsonConverter(typeof(JsonStringEnumConverter))]
        public AdvertStatusEnum status { get; set; }
        public DateTime startTime { get; set; }
        public DateTime endTime { get; set; }
        public string ownerId { get; set; } = string.Empty;
        public User owner { get; set; }
        public string? performerId { get; set; }
        public User? performer { get; set; }
        public List<Request>? requests { get; set; }

    }
}
```

Файл AdvertRepository.cs

Реалізація функціональної вимоги "часткова автоматизація процесу пошуку виконавців, шляхом генерації пропозицій можливим виконавцям, що задовольняють обмеженням"

```
using backendPetHome.DAL.Data;
using backendPetHome.DAL.Entities;
using backendPetHome.DAL.Interfaces.RepositoryInterfaces;
using backendPetHome.DAL.Specifications;
using Microsoft.EntityFrameworkCore;

namespace backendPetHome.DAL.Repositories
{
    public class AdvertRepository :IAdvertRepository
    {
        private readonly DataContext _context;
        public AdvertRepository(DataContext context)
```

```
            {
                _context = context;
            }

            public async Task Add(Advert advertToAdd)
            {
                await
_context.Set<Advert>().AddAsync(advertToAdd);
            }

            public                                    Task<Advert?>
GetByIdSpecification(Specification<Advert> spec)
            {
                return
ApplySpecification(spec).SingleOrDefaultAsync();
            }

            public                                Task<List<Advert>>
GetBySpecification(Specification<Advert> spec)
            {
                return ApplySpecification(spec).ToListAsync();
            }

            public  async  Task<(List<Advert>  fitAdverts,  int
totalCount)>    GetBySpecificationAndPaging(Specification<Advert>
spec)
            {
                var      fitAdvertsWithPaging      =      await
ApplySpecification(spec).ToListAsync();
                spec.RemovePagination();
                var          totalCount          =          await
ApplySpecification(spec).CountAsync();
                return (fitAdvertsWithPaging,totalCount);
            }
            public async Task Update(Advert advertToUpdate)
            {
                _context.Set<Advert>().Attach(advertToUpdate);
                _context.Entry(advertToUpdate).State           =
EntityState.Modified;
            }
            public async Task Delete(Advert entity)
            {
                _context.Set<Advert>().Remove(entity);
            }

            private                                IQueryable<Advert>
ApplySpecification(Specification<Advert> specification)
```

```
            {
                IQueryable<Advert> set;

                if (specification?.userId !=null)
                {
                     set                                          =
_context.getTimeExceptionFitAdverts(specification.userId);
                }
                else
                {
                    set = _context.Set<Advert>();
                }

                return      SpecificationEvaluator.getQuery(set,
specification);
            }
        }
    }
```

Файл AdvertByIdIncludesRequestAndUserSpecification.cs

Реалізація функціональної вимоги "часткова автоматизація процесу пошуку виконавців, шляхом генерації пропозицій можливим виконавцям, що задовольняють обмеженням"

```
    using backendPetHome.DAL.Entities;

    namespace
backendPetHome.DAL.Specifications.AdvertSpecifications
    {
        public                                              class
AdvertByIdIncludesRequestAndUserSpecification                   :
Specification<Advert>
        {
            public
AdvertByIdIncludesRequestAndUserSpecification(int id)
                : base(a => a.Id == id)
            {
                AddInclude(a => a.requests);

AddInclude($"{nameof(Advert.requests)}.{nameof(Request.user)}");
            }
        }
    }
```

Файл Specification.cs

Реалізація функціональної вимоги "часткова автоматизація процесу пошуку виконавців, шляхом генерації пропозицій можливим виконавцям, що задовольняють обмеженням"

```
using backendPetHome.DAL.Entities;
using backendPetHome.DAL.Specifications.QueryParameters;
using System.Linq.Expressions;

namespace backendPetHome.DAL.Specifications
{
    public abstract class Specification<TEntity> where TEntity : class
    {
        protected Specification(Expression<Func<TEntity, bool>> criteria)
        {
            Criteria = criteria;
        }
        public QueryStringParameters? QueryParameters;
        public string userId{ get; private set; }
        public Expression<Func<TEntity, bool>>? Criteria { get; }
        public List<Expression<Func<TEntity, object>>> IncludeExpressions { get; } = new();
        protected void AddInclude(Expression<Func<TEntity, object>> includeExpression)
        {
            IncludeExpressions.Add(includeExpression);
        }
        public List<string> IncludeStrings { get; } = new List<string>();
        protected virtual void AddInclude(string includeString)
        {
            IncludeStrings.Add(includeString);
        }
        protected void AddPagination(QueryStringParameters parameters)
        {
            QueryParameters = parameters;
        }
        public void RemovePagination()
        {
            QueryParameters = null;
        }
```

```
            protected virtual void AddFitDatesCriteria(string
userId)
            {
                this.userId = userId;
            }
        }
    }
```

Файл SpecificationEvaluator.cs

Реалізація функціональної вимоги "часткова автоматизація процесу пошуку виконавців, шляхом генерації пропозицій можливим виконавцям, що задовольняють обмеженням"

```
    using Microsoft.EntityFrameworkCore;

    namespace backendPetHome.DAL.Specifications
    {
        public static class SpecificationEvaluator
        {
            public static IQueryable<TEntity> getQuery<TEntity>
(IQueryable<TEntity>   inputQueryable,   Specification<TEntity>
specification) where TEntity : class
            {
                IQueryable<TEntity> quaryable = inputQueryable;

                if (specification.Criteria != null)
                {
                    quaryable                                    =
quaryable.Where(specification.Criteria);
                }

                quaryable                                        =
specification.IncludeExpressions.Aggregate(quaryable,   (current,
includeExpression) =>
                    current.Include(includeExpression));

                quaryable                                        =
specification.IncludeStrings.Aggregate(quaryable,
                                    (current,    include)    =>
current.Include(include));

                if (specification.QueryParameters != null)
                {
                    quaryable                                    =
quaryable.Skip((specification.QueryParameters.PageNumber  -  1)  *
specification.QueryParameters.PageSize)
```

```
.Take(specification.QueryParameters.PageSize);
                }
                return quaryable;
            }
        }
    }
```

Файл QueryStringParams.cs

Реалізація функціональної вимоги "фільтрація з можливістю вибору оголошень, що враховують графік завантаженості виконавця"

```
    using backendPetHome.DAL.Enums;

    namespace backendPetHome.DAL.Specifications.QueryParameters
    {
        public class QueryStringParameters
        {
            const int maxPageSize = 36;
            public int PageNumber { get; set; } = 1;
            private int _pageSize = 9;
            public int priceFrom { get; set; } = 0;
            public int priceTo { get; set; } = int.MaxValue;
            public int PageSize
            {
                get
                {
                    return _pageSize;
                }
                set
                {
                    _pageSize = value > maxPageSize ? maxPageSize
: value;
                }
            }
            public AdvertStatusEnum? advertsStatus { get; set; }
= AdvertStatusEnum.search;
            public bool isDatesFit { get; set; } = false;
        }
    }
```

Файл AdvertWithParamsAndPaginationSpecification.cs

Реалізація функціональної вимоги "фільтрація з можливістю вибору оголошень, що враховують графік завантаженості виконавця"

```
    using backendPetHome.DAL.Enums;
    using backendPetHome.DAL.Entities;
    using backendPetHome.DAL.Specifications.QueryParameters;
```

```
     namespace
backendPetHome.DAL.Specifications.AdvertSpecifications
     {
         public class AdvertWithParamsAndPaginationSpecification
: Specification<Advert>
         {
             public
AdvertWithParamsAndPaginationSpecification(string        userId,
QueryStringParameters parameters)
                 : base(el => el.cost >= parameters.priceFrom &&
el.cost <= parameters.priceTo
                 && el.status == parameters.advertsStatus )
             {
                 AddPagination(parameters);
                 if (parameters.isDatesFit)
                 {
                     AddFitDatesCriteria(userId);

                 }
             }
         }
     }
```

Файл Filters.jsx

Реалізація функціональної вимоги "фільтрація з можливістю вибору оголошень, що враховують графік завантаженості виконавця"

```
import React, { useState } from 'react';
import { View, Text, TouchableOpacity, TextInput, Switch } from
'react-native';
import FiltersStyles from './FiltersStyles';
import shallowEqual from '../../../Pages/Me/helper';
import DropDownPicker from 'react-native-dropdown-picker';
import useStore from '../../../Hooks/useAuth';

export default function Filters({ isUserAdverts, queryParams,
setQueryParams }) {
    const store = useStore()
    const [isVisible, setIsVisible] = useState(false)
    const [open, setOpen] = useState(false);
    const [openStatusPicker, setOpenStatusPicker] =
useState(false);

    const [items, setItems] = useState([
        { label: 'По 4', value: 4 },
        { label: 'По 8', value: 8 },
        { label: 'По 12', value: 12 }
```

```
    ]);

    const [statusItems, setStatusItems] = useState([
        { label: 'Пошук', value: "search" },
        { label: 'Виконуються', value: "process" },
        { label: 'Завершені', value: "finished" }
    ]);

    const [queryParamsCopy, setQueryParamsCopy] = useState({
        advertsLimit: queryParams?.advertsLimit.toString(),
        costFrom: queryParams?.costFrom.toString(),
        costTo: queryParams?.costTo.toString(),
        isDatesFit: queryParams?.isDatesFit,
        advertsStatus: queryParams?.advertsStatus,
    })

    const handleApply = () => {
        if (queryParamsCopy?.advertsLimit.trim() === "" ||
isNaN(queryParamsCopy?.advertsLimit) ||
parseInt(queryParamsCopy?.advertsLimit) <= 0) {
            alert("Оберіть коректну кількість оголошень на
сторінку");
            return;
        }
        if (queryParamsCopy?.costFrom.trim() === "" ||
isNaN(queryParamsCopy?.costFrom) ||
parseInt(queryParamsCopy?.costFrom) <= 0) {
            alert("Введіть коректну мінімальну суму");
            return;
        }
        if (queryParamsCopy?.costTo.trim() === "" ||
isNaN(queryParamsCopy?.costTo) ||
parseInt(queryParamsCopy?.costTo) <= 0) {
            alert("Введіть коректну максимальну суму");
            return;
        }

        if (parseInt(queryParamsCopy?.costTo) > 100000) {
            alert("Сума не може бути більше за 100000");
            return;
        }

        if (parseInt(queryParamsCopy?.costTo) <
parseInt(queryParamsCopy?.costFrom)) {
            alert("Максимальна сума не може бути меншою за
мінімальну");
            return;
```

```
        }
        const temp = { ...queryParams, ...queryParamsCopy }

        if (!shallowEqual(queryParams, temp)) {
            setQueryParams({ ...queryParams, ...queryParamsCopy,
currentPage: 1 });
        } else {
            setIsVisible(!isVisible)
        }
    }

    return (
        <View style={[FiltersStyles.container,
FiltersStyles.shadow]}>
            {isVisible ?
                <View style={{ width: '100%', alignItems:
'center' }}>
                    <View style={{ zIndex: 6 }}>
                        <Text
style={FiltersStyles?.label}>Оголошень на сторінку</Text>
                        <DropDownPicker
                            open={open}

value={parseInt(queryParamsCopy?.advertsLimit)}
                            items={items}
                            setOpen={setOpen}
                            onSelectItem={(item) =>
setQueryParamsCopy({ ...queryParamsCopy, advertsLimit:
String(item.value), currentPage: 1 })}
                            setItems={setItems}
                            containerStyle={{ width: '50%' }}
                        />
                    </View>
                    {!store?.role?.includes("Administrator") &&
<View style={FiltersStyles.costSelectionContainer}>
                        <Text style={FiltersStyles?.label}>Ціна
від:</Text>
                        <TextInput
                            style={FiltersStyles.input}
                            value={queryParamsCopy?.costFrom}
                            onChangeText={text =>
setQueryParamsCopy({ ...queryParamsCopy, costFrom: text })}
                            keyboardType="numeric"
                        />
                        <Text style={FiltersStyles?.label}>Ціна
до:</Text>
                        <TextInput
```

```
                            style={FiltersStyles.input}
                            value={queryParamsCopy?.costTo}
                            onChangeText={text =>
setQueryParamsCopy({ ...queryParamsCopy, costTo: text })}
                            keyboardType="numeric"
                        />
                    </View>
                    }
                    {
                        !isUserAdverts &&
!store?.role?.includes("Administrator") ? <View style={{
flexDirection: 'row', alignItems: 'center' }}>
                            <Text style={FiltersStyles?.label}>У
Ваші вільні дати</Text>
                            <Switch

value={queryParamsCopy?.isDatesFit}
                                onValueChange={(e) =>
setQueryParamsCopy({ ...queryParamsCopy, isDatesFit: e })}
                            />
                        </View> :
                            <View style={{ zIndex: 5 }}>
                                <Text
style={FiltersStyles?.label}>Статус</Text>
                                <DropDownPicker
                                    open={openStatusPicker}

value={(queryParamsCopy?.advertsStatus)}
                                    items={statusItems}

setOpen={setOpenStatusPicker}
                                    onSelectItem={(item) =>
setQueryParamsCopy({ ...queryParamsCopy, advertsStatus:
String(item.value), currentPage: 1 })}
                                    setItems={setStatusItems}
                                    containerStyle={{ width:
'50%' }}
                                />
                            </View>
                    }

                    <TouchableOpacity
style={FiltersStyles.apply} onPress={handleApply}>
                        <Text>Застосувати</Text>
                    </TouchableOpacity>
                </View>
```

```
                : <View style={{ width: '100%'
}}><TouchableOpacity style={FiltersStyles.showFiltersButton}
onPress={() => setIsVisible(!isVisible)}><Text>Показати
фільтри</Text></TouchableOpacity></View>
            }
        </View>
    );
}
```

Файл getTimeExceptionFitAdverts.sql

Реалізація функціональної вимоги "фільтрація з можливістю вибору оголошень, що враховують графік завантаженості виконавця"

```
use testdb;


go
CREATE FUNCTION dbo.getTimeExceptionFitAdverts (
    @userId NVARCHAR(100)
)
RETURNS TABLE
AS
RETURN (
    SELECT *
    FROM adverts a
    WHERE NOT EXISTS (
        SELECT 1
        FROM timeExceptions te
        WHERE te.userId = @userId
        AND CAST(te.date AS DATE) >= CAST(a.startTime AS
DATE)
        AND CAST(te.date AS DATE) <= CAST(a.endTime AS DATE)
    )
);
go
```

Файл TimeException.cs

Реалізація функціональної вимоги "фільтрація з можливістю вибору оголошень, що враховують графік завантаженості виконавця"

```
using backendPetHome.DAL.Entities.Abstract;
using System.ComponentModel.DataAnnotations;

namespace backendPetHome.DAL.Entities
{
    public class TimeException : BaseEntity
    {
        public int id { get; set; }
```

```
            public string userId { get; set; }
            public User user { get; set; }
            [Required]
            public DateTime date { get; set; }
        }
    }
```

Файл AdvertNotification.jsx

Реалізація функціональної вимоги "сповіщення в режимі реального часу"

```
    import { View, Text, Image } from 'react-native'
    import React from 'react'
    import MyButton from '../../Common/MyButton'
    import             AdvertNotificationStyles             from
'./AdvertNotificationStyles'
    import Colors from '../../../Constants/Colors'

    export default function AdvertNotification({ advert, status,
navigationRef, hide, ...props }) {

                        const        advertPhotoFilePath        =
`${process.env.EXPO_PUBLIC_API_URL}${advert?.photoFilePath}`;

        function toRequests() {
           navigationRef.current?.navigate('Деталі мого запиту',
{ adId: advert?.id });
            hide()
        }

        function renderSwitch(status) {
            let text = ''
            switch (status) {
                case 'generated':
                      text = 'Ми підібрали для Вас оголошення,
погляньте.'
                    break
                case 'confirm':
                    text = 'Власник підтвердив Вашу заявку.'
                    break
                case 'reject':
                    text = 'Власник відхилив Вашу заявку.'
                    break
                default:
                    text = 'Очікуємо'
            }
            return <Text style={{ color: 'white' }}>{text}</Text>
        }
```

```
        return (
                                                                <View
style={AdvertNotificationStyles.notificationBlock}>
                                                                <View
style={AdvertNotificationStyles.contentSection}>
                                                                <View
className={AdvertNotificationStyles.imgSection}>
                            <Image
                               source={{ uri: advertPhotoFilePath }}
                                alt='advertPhoto'
                                style={AdvertNotificationStyles.img}
                            />
                        </View>
                                                                <View
style={AdvertNotificationStyles.infoSection}>
                            {status && renderSwitch(status)}
                        </View>
                    </View>
                    <MyButton style={{ backgroundColor: Colors.main,
borderBottomLeftRadius:  20,  borderBottomRightRadius:  20  }}
onPress={toRequests}>Переглянути заявки</MyButton>
                </View>
        )
    }
```

Файл UserNotification.jsx

Реалізація функціональної вимоги "сповіщення в режимі реального часу"

```
    import { View, Text, Image } from 'react-native'
    import React from 'react'
    import              UserNotificationStyles              from
'./UserNotificationStyles';
    import MyButton from '../Common/MyButton';
    import Colors from '../../Constants/Colors';

    export default function UserNotification({ request, status,
navigationRef, hide, ...props }) {
                            const         userPhotoFilePath         =
`${process.env.EXPO_PUBLIC_API_URL}${request?.user?.photoFilePat
h}`;

        function toRequests() {
                    navigationRef.current?.navigate('Деталі  мого
оголошення', { adId: request?.advertId });
                hide()
        }

        function renderSwitch(status) {
```

```
            switch (status) {
                case 'delete':
                       return request?.status === 'generated' ?
<Text>відмовився від згенерованої заявки</Text> : <Text>відмінив
заявку</Text>;
                case 'apply':
                    return <Text>подав заявку</Text>;
                default:
                    return <Text>Очікуємо</Text>;
            }
        }

        return (
                                                                <View
style={UserNotificationStyles.notificationBlock}>
                                                                <View
style={UserNotificationStyles.contentSection}>
                                                                <View
style={UserNotificationStyles.imgSection}>
                        <Image
                            source={{ uri: userPhotoFilePath }}
                            alt="user image"
                           style={UserNotificationStyles.img} />
                    </View>
                                                                <View
style={UserNotificationStyles.infoSection}>
                               <Text style={{ color: Colors.white,
textAlign: 'center' }}>
                                Користувач {request?.user?.surname}
{request?.user?.name}{' '}
                                {status && renderSwitch(status)}
                            </Text>
                        </View>
                </View>

                <MyButton style={{ backgroundColor: Colors.main,
borderBottomLeftRadius:  20,  borderBottomRightRadius:  20  }}
onPress={toRequests}>Переглянути заявки.</MyButton>
            </View>
        );
    }
```

Файл Navigator.jsx

Реалізація функціональної вимоги "сповіщення в режимі реального часу"

```
import { TouchableOpacity, Vibration } from "react-native"
import React, { useEffect, useState } from "react"
import useStore from "../Hooks/useAuth";
```

```
    import { FontAwesome5, Ionicons, AntDesign, Entypo, Feather
} from "@expo/vector-icons";
    import   {   createBottomTabNavigator   }   from   "@react-
navigation/bottom-tabs";
    import      {      NavigationContainer,      DefaultTheme,
createNavigationContainerRef } from "@react-navigation/native";
    import    {    createStackNavigator    }    from    "@react-
navigation/stack";
    import { observer } from "mobx-react-lite"
    import My from "../Pages/My/My";
    import Adverts from "../Pages/Adverts/Adverts";
    import Colors from "../Constants/Colors";
    import Me from "../Pages/Me/Me";
    import Login from "../Pages/Login/Login";
    import                  Registration                  from
"../Pages/Registration/Registration";
    import Loader from "./Loader/Loader";
    import                  AdvertDetail                  from
"../Pages/AdvertDetail/AdvertDetail";
    import UserProfile from "../Pages/UserProfile/UserProfile";
    import                  CreateAdvert                  from
"../Pages/CreateAdvert/CreateAdvert";
    import AdminPanel from "../Pages/Administrator/AdminPanel";
    import Toast from 'react-native-toast-message'
    import                UserNotification                from
"./Notifications/UserNotification";
    import               AdvertNotification               from
"./Notifications/AdvertNotification/AdvertNotification";

    const Tab = createBottomTabNavigator();
    const Stack = createStackNavigator();

    const MyTheme = {
        ...DefaultTheme,
        colors: {
            ...DefaultTheme.colors,
            background: 'white',
        },
    };

    export default observer(function Navigator() {
        const [isEditing, setIsEditing] = useState(false)
        const store = useStore();

        useEffect(() => {
            async function checkAuth() {
                await store.checkAuth()
```

```
                store.setLoading(false)
            }
            checkAuth()
        }, []);

        useEffect(() => {
            if (store) {
                setIsEditing(store.isEditing)
            }
        }, [store.isEditing]);

        const toastConfig = {
            notificationUserToast: ({ props }) => (
                < UserNotification {...props} />
            ),
            notificationAdvertToast: ({ props }) => (
                < AdvertNotification {...props} />
            )
        };

        useEffect(() => {
            async function createHubConnection() {
                await store.createHubConnection()
            }
            if (store.isAuth) {
                createHubConnection()
            }
        }, [store.isAuth]);

        const hideToast = () => Toast.hide()

        useEffect(() => {
            if (store?.myHubConnection) {
                store?.myHubConnection?.on("Apply", (request) =>
{
                    Toast.show({
                        type: 'notificationUserToast',
                        props: {
                            request: request,
                            status: "apply",
                            navigationRef: navigationRef,
                            hide: hideToast
                        }
                    });
                })
                            store?.myHubConnection?.on("Delete",
(deletedRequest) => {
```

```
                    Toast.show({
                        type: 'notificationUserToast',
                        props: {
                            request: deletedRequest,
                            status: "delete",
                            navigationRef: navigationRef,
                            hide: hideToast
                        }
                    });
                })
                store?.myHubConnection?.on("Send", (advert) => {
                    Toast.show({
                        type: 'notificationAdvertToast',
                        props: {
                            advert: advert,
                            status: "generated",
                            navigationRef: navigationRef,
                            hide: hideToast
                        }
                    });
                }
                )
                        store?.myHubConnection?.on("Confirm",
(confirmedRequest) => {
                    Toast.show({
                        type: 'notificationAdvertToast',
                        props: {
                            advert: confirmedRequest?.advert,
                            status: "confirm",
                            navigationRef: navigationRef,
                            hide: hideToast
                        }
                    });
                })
                        store?.myHubConnection?.on("Reject",
(rejectedRequest) => {
                    Toast.show({
                        type: 'notificationAdvertToast',
                        props: {
                            advert: rejectedRequest?.advert,
                            status: "reject",
                            navigationRef: navigationRef,
                            hide: hideToast
                        }
                    });
                })
            }
```

```
        }, [store?.myHubConnection]);

        if (store.isLoading) {
            return <Loader />
        }

        const AdvertsStack = () => {
            return (
                <Stack.Navigator>
                        <Stack.Screen name="Перелік оголошень"
component={Adverts} options={{
                            headerShown: false
                        }}
                            initialParams={{
                                isUserAdverts: false
                            }} />
                                    <Stack.Screen name="Деталі"
component={AdvertDetail} options={{
                            headerShown: false
                        }} />
                                <Stack.Screen name="Профіль"
component={UserProfile} options={{
                            headerShown: false
                        }} />
                </Stack.Navigator>
            );
        };

        const navigationRef = React.createRef();

                            const authContent =
store?.role?.includes("Administrator") ?
             <AdminPanel theme={MyTheme} Tab={Tab} Stack={Stack}
advertsStack={AdvertsStack} />
             :
                            <NavigationContainer theme={MyTheme}
ref={navigationRef}>
                <Tab.Navigator screenOptions={{
                    lazy: false
                }}>
                                    <Tab.Screen name="Оголошення"
component={AdvertsStack}
                            options={{
                                    tabBarIcon: () => <FontAwesome5
name="dog" color={Colors.main} size={24} />
                            }}
                        />
```

```
                                    <Tab.Screen   name="Створити"
component={CreateAdvert}
                            options={{
                                        tabBarIcon:  ()  =>  <Ionicons
name="create-outline" size={24} color={Colors.main} />,
                                lazy: true
                            }} />
                        <Tab.Screen name="Мої" component={My}
                            options={{
                                      tabBarIcon:  ()  =>  <AntDesign
name="folder1" size={24} color={Colors.main} />
                            }}
                        />
                        <Tab.Screen name="Я" component={Me}
                            options={{
                                        tabBarIcon:  ()  =>  <Ionicons
name="person-outline" size={24} color={Colors.main} />,
                                headerLeft: () => (
                                                    <TouchableOpacity
onPress={store.logout}>
                                        <Ionicons name="exit-outline"
size={24} color={Colors.main} style={{ marginLeft: 20 }} />
                                    </TouchableOpacity>
                                ),
                                headerRight: () => isEditing
                                    ? (
                                        <TouchableOpacity onPress={()
=> store.setIsEditing(!store.isEditing)}>
                                                    <Feather name="check"
size={24} color={Colors.main} style={{ marginRight: 20 }} />
                                        </TouchableOpacity>
                                    )
                                    : (
                                        <TouchableOpacity onPress={()
=> store.setIsEditing(!store.isEditing)}>
                                                    <Entypo name="edit"
size={20} color={Colors.main} style={{ marginRight: 20 }} />
                                        </TouchableOpacity>
                                    ),
                            }} />
                    </Tab.Navigator>
                       <Toast  config={toastConfig}  autoHide={false}
visibilityTime={3500}   onShow={()   =>   Vibration.vibrate(50)}
topOffset={50} />
                </NavigationContainer>

            return (
```

```
            store.isAuth ? (
                authContent
            ) : (
                <NavigationContainer>
                    <Stack.Navigator>
                                        <Stack.Screen  name="Логін"
component={Login} />
                                    <Stack.Screen  name="Реєстрація"
component={Registration} />
                    </Stack.Navigator>
                </NavigationContainer>
            )
        );
    });
```

Файл AdminUsers.jsx

Реалізація функціональної вимоги "функціонал адміністрування для модерації профілів та оголошень"

```
    import { ScrollView, Text, View, TextInput } from 'react-
native'
    import React, { useState, useEffect } from 'react'
    import useFetching from '../../../Hooks/useFetching'
    import AdminService from '../../../HTTP/API/AdminService'
    import                    ProfileItem                    from
'../../../Components/Profile/ProfileItem'
    import Loader from '../../../Components/Loader/Loader'
    import MyModal from '../../../Components/MyModal/MyModal'
    import useStore from '../../../Hooks/useAuth'
    import { observer } from 'mobx-react-lite'
    import MyButton from '../../../Components/Common/MyButton'
    import Colors from '../../../Constants/Colors'
    import AdminUsersStyles from './AdminUsersStyles'

    export default observer(function AdminUsers({ navigation })
{
        const store = useStore()
        const [users, setUsers] = useState([])
             const    [isModalVisible,    setIsModalVisible]    =
useState(false)
         const  [isAdminModalVisible,  setIsAdminModalVisible]  =
useState(false)
          const  [addAdminData,  setAddAdminData]  =  useState({
username: "check", password: 'Password123!' })
        const [fetchUsers, loading, error] = useFetching(async ()
=> {
            const userResponse = await AdminService.getAllUsers()
             setUsers(userResponse)
```

```
        })
        const [addAdmin, loading2, error2] = useFetching(async ()
=> {
            await AdminService.addAdmin(addAdminData)
        })

        useEffect(() => {
            async function fetchData() {
                try {
                    await fetchUsers()
                } catch (e) {
                    setIsModalVisible(true)
                }
            }
            fetchData()
        }, [store?.usersNeedUpdate])

        const addAdminHandler = async () => {
            try {
                setIsAdminModalVisible(false)
                await addAdmin()
                alert('Створено')
            } catch (e) {
                console.log(e);
                setIsModalVisible(true)
            }
        }

        const addAdminForm = <View style={{ marginBottom: 220 }}>
              <Text  style={{  textAlign:  'center',  margin:  30,
fontWeight: 'bold' }}>Уведіть дані</Text>
            <TextInput
                placeholder="Логін*"
                value={addAdminData.username}
                       onChangeText={text  =>  setAddAdminData({
...addAdminData, username: text })}
                style={AdminUsersStyles.input}
            />

            <TextInput
                placeholder="Пароль*"
                value={addAdminData.password}
                       onChangeText={text  =>  setAddAdminData({
...addAdminData, password: text })}
                style={AdminUsersStyles.input}
            />
```

```
            <MyButton style={{ backgroundColor: Colors.green }}
onPress={addAdminHandler}>Додати</MyButton></View>


        if (loading || loading2) return <Loader />
        return (
            <ScrollView>
                        <MyModal  isModalVisible={isModalVisible}
setIsModalVisible={setIsModalVisible}
content={<Text>{error}{error2}</Text>}></MyModal>

                    <MyModal  isModalVisible={isAdminModalVisible}
setIsModalVisible={setIsAdminModalVisible}
content={addAdminForm}></MyModal>
                <MyButton style={{ backgroundColor: Colors.light
}}    onPress={()    =>    setIsAdminModalVisible(true)}>Додати
адміністратора</MyButton>
                {users.map((el) =>
                    <ProfileItem
                        key={el?.id}
                        requestData={el}
                        navigation={navigation}
                        profileData={el}
                        isBig
                        borderRadiusTop
                        id={el?.id}
                    />
                )}
            </ScrollView>
        )
    })
```

Файл AdminPanel.jsx

Реалізація функціональної вимоги "функціонал адміністрування для модерації профілів та оголошень"

```
    import React from 'react'
    import    {    NavigationContainer    }    from    "@react-
navigation/native";
    import AdminUsers from './AdminUsers/AdminUsers';
    import { FontAwesome5, Ionicons } from "@expo/vector-icons";
    import Colors from '../../Constants/Colors';
    import UserProfile from '../UserProfile/UserProfile';
    import { TouchableOpacity, View, Text } from 'react-native';
    import useStore from '../../Hooks/useAuth';
```

```
    export default function AdminPanel({ theme, Tab, Stack,
advertsStack }) {
        const store = useStore()
        const AdminUsersStack = () => {
            return (<Stack.Navigator>
                     <Stack.Screen name="Перелік користувачів"
component={AdminUsers} options={{
                    headerShown: false
                }} />
                                    <Stack.Screen   name="Профіль"
component={UserProfile} options={{
                    headerShown: false
                }} />
            </Stack.Navigator>)
        }

                const    logoutBtn    =    <TouchableOpacity
onPress={store.logout}>
                     <Ionicons   name="exit-outline"   size={24}
color={Colors.main} style={{ marginLeft: 20 }} />
        </TouchableOpacity>

          const adminTitle = <View style={{ backgroundColor:
Colors.red, marginRight: 20, padding: 3, borderRadius: 5, }}><Text
style={{ color: 'white' }}>Адміністратор</Text></View>

        return (
            <NavigationContainer theme={theme} >
                <Tab.Navigator>
                            <Tab.Screen name="Всі оголошення"
component={advertsStack}
                        options={{
                                tabBarIcon: () => <FontAwesome5
name="dog" color={Colors.main} size={24} />,
                            headerLeft: () => (logoutBtn),
                            headerRight: () => (adminTitle)
                        }}

                    />
                            <Tab.Screen name="Всі користувачі"
component={AdminUsersStack}
                        options={{
                                  tabBarIcon: () => <Ionicons
name="person-outline" size={24} color={Colors.main} />,
                            headerLeft: () => (logoutBtn),
                            headerRight: () => (adminTitle)
                        }}
```

```
                    />

                </Tab.Navigator>
            </NavigationContainer>
        )
    }
```

Файл UsersController.cs

Реалізація функціональної вимоги "функціонал адміністрування для модерації профілів та оголошень"

```
using Microsoft.AspNetCore.Mvc;
using backendPetHome.BLL.Services;
using backendPetHome.BLL.DTOs.UserDTOs;
using Microsoft.AspNetCore.Authorization;
using backendPetHome.API.Controllers.Abstract;
using backendPetHome.BLL.DTOs.AdminDTOs;

namespace backendPetHome.API.Controllers
{
    [Route("api/[controller]")]
    public class UsersController : BaseController
    {
        private readonly UserService _userService;

        public UsersController(UserService userService)
        {
            _userService = userService;
        }

        [HttpGet("{id}")]
        public async Task<ActionResult<UserDTO>> Get(string
id)
        {
            var user = await _userService.getCertainUser(id);
            return Ok(user);
        }

        [Authorize(Roles = "Administrator")]
        [HttpPost("adminAdd")]
        public async Task<IActionResult> AddAdmin([FromBody]
AdminAddDTO creds)
        {
            await _userService.addAdmin(creds);
            return Ok();
        }

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

```
            [HttpGet]
            public    async    Task<ActionResult<List<UserDTO>>>
GetUsers()
            {
                var users = await _userService.getUsers(UserId);
                return Ok(users);

            }

            [Authorize(Roles = "Administrator")]
            [HttpDelete("{userToDeleteId}")]
            public    async    Task<ActionResult<List<UserDTO>>>
DeleteUser(string userToDeleteId)
            {
                if(UserId == userToDeleteId) return BadRequest();
                await
_userService.deleteUserProfile(userToDeleteId);
                return Ok();
            }
        }
    }
```

Файл TimeExceptionsController.cs

Реалізація функціональної вимоги "робота з графіком завантаженості"

```
    using Microsoft.AspNetCore.Mvc;
    using backendPetHome.API.Controllers.Abstract;
    using backendPetHome.BLL.Services.Interfaces;

    namespace backendPetHome.API.Controllers
    {
        [Route("api/[controller]")]
        public class TimeExceptionsController : BaseController
        {
            private        readonly        ITimeExceptionService
_timeExceptionServise;
            public
TimeExceptionsController(ITimeExceptionService
timeExceptionService)
            {
                _timeExceptionServise = timeExceptionService;
            }
            [HttpPost]
            public   async   Task<ActionResult>   Post([FromBody]
IEnumerable<DateTime> dates)
            {
                await
_timeExceptionServise.addTimeExceptions(UserId, dates);
```

```
                return Ok();
            }
            [HttpDelete]
            public async Task<ActionResult> Delete([FromBody]
IEnumerable<DateTime> dates)
            {
                await
_timeExceptionServise.deleteTimeExceptions(UserId, dates);
                return Ok();
            }
        }
    }
```

Файл TimeExceptionsService.cs

Реалізація функціональної вимоги "робота з графіком завантаженості"

```
    using AutoMapper;
    using backendPetHome.BLL.Services.Abstract;
    using backendPetHome.BLL.Services.Interfaces;
    using backendPetHome.DAL.Entities;
    using backendPetHome.DAL.Interfaces;
    using
backendPetHome.DAL.Specifications.TimeExceptionSpecifications;

    namespace backendPetHome.BLL.Services
    {
        public class TimeExceptionService : BaseService,
ITimeExceptionService
        {
            public TimeExceptionService(IUnitOfWork unitOfWork,
IMapper mapper) : base(unitOfWork, mapper)
            {
            }
            public async Task addTimeExceptions(string userId,
IEnumerable<DateTime> dates)
            {
                var userExceptions = await _unitOfWork
                    .TimeExceptionRepository
                    .GetBySpecification(new
TimeExceptionCurrentUserSpecification(userId));

                foreach (var date in dates)
                {
                    TimeException exception = new();
                    exception.userId = userId;
                    exception.date = date;
                    if (userExceptions.FindIndex(el => el.date ==
exception.date) == -1)
```

```
                    {
                      await
_unitOfWork.TimeExceptionRepository.Add(exception);
                    }
                }

                await _unitOfWork.SaveChangesAsync();
            }

            public async Task deleteTimeExceptions(string userId,
IEnumerable<DateTime> datesToRemove)
            {
                var userExceptions = await _unitOfWork
                    .TimeExceptionRepository
                    .GetBySpecification(new
TimeExceptionCurrentUserSpecification(userId));

                foreach (var userException in userExceptions)
                {
                    if (datesToRemove.Any(dateToRemove =>
                    dateToRemove.Year == userException.date.Year
&&
                    dateToRemove.Month                          ==
userException.date.Month &&
                    dateToRemove.Day == userException.date.Day))
                    {
                        await
_unitOfWork.TimeExceptionRepository.Delete(userException);
                    }
                }

                await _unitOfWork.SaveChangesAsync();
            }

            public  async  Task<bool>  checkPerformerDates(string
userId, DateTime advertStart, DateTime advertEnd)
            {
                List<TimeException> fitExceptions = await
                    _unitOfWork
                    .TimeExceptionRepository
                    .GetBySpecification(new
TimeExceptionCurrentUserFitAdvertTimeSpecification(userId,
advertStart, advertEnd));
                return !fitExceptions.Any();
            }
        }
    }
```

Файл TimeExceptionRepository.cs
Реалізація функціональної вимоги "робота з графіком завантаженості"

```
using backendPetHome.DAL.Data;
using backendPetHome.DAL.Entities;
using backendPetHome.DAL.Interfaces.RepositoryInterfaces;
using backendPetHome.DAL.Specifications;
using Microsoft.EntityFrameworkCore;

namespace backendPetHome.DAL.Repositories
{
    public class TimeExceptionRepository:
ITimeExceptionRepository
    {
        private readonly DataContext _context;
        public TimeExceptionRepository(DataContext context)
        {
            _context = context;
        }

        public async Task Add(TimeException
timeExceptionToAdd)
        {
            await
_context.Set<TimeException>().AddAsync(timeExceptionToAdd);
        }

        public async Task Delete(TimeException
timeExceptionToDelete)
        {

_context.Set<TimeException>().Remove(timeExceptionToDelete);
        }

        public Task<List<TimeException>>
GetBySpecification(Specification<TimeException> spec)
        {
            return ApplySpecification(spec).ToListAsync();
        }

        private IQueryable<TimeException>
ApplySpecification(Specification<TimeException> specification)
        {
            return
SpecificationEvaluator.getQuery(_context.Set<TimeException>(),
specification);
        }
```

```
        }
    }
```

Файл MyCalendar.jsx

Реалізація функціональної вимоги "робота з графіком завантаженості"

```
    import { View, Button } from 'react-native'
    import React, { useEffect, useState } from 'react'
    import MyCalendarStyles from './MyCalendarStyles';
    import Colors from '../../Constants/Colors';
    import { processDates } from './CalendarHelper';
    import useFetching from '../../Hooks/useFetching';
    import                TimeExceptionService                from
'../../HTTP/API/TimeExceptionService';
    import Loader from '../Loader/Loader';
    import UkrCalendar from './UkrCalendar';
    import MyButton from '../Common/MyButton';

    export default function MyCalendar() {

        const [datesChanging, setDatesChanging] = useState(false)
        const [needUpdate, setNeedUpdate] = useState(false)
        const [timeExceptions, setTimeExceptions] = useState([])
         const  [datesToUpdate,  setDatesToUpdate]  =  useState({
datesToAdd: [], datesToDelete: [] })
        const [dates, setDates] = useState({})
        const [prevDates, setPrevDates] = useState({})

         const [updateDates, loading, error] = useFetching(async
() => {
            if (datesToUpdate.datesToAdd.length) {
                                                              await
TimeExceptionService.addUserTimeExceptions(datesToUpdate.datesTo
Add)
            }
            if (datesToUpdate.datesToDelete.length) {
                                                              await
TimeExceptionService.deleteUserTimeExceptions(datesToUpdate.date
sToDelete)
            }

        })

        const [fetchDates, loading2, error2] = useFetching(async
() => {
                                const     response     =     await
TimeExceptionService.getUserTimeExceptions()
            setTimeExceptions(response)
        })

        useEffect(() => {
            async function fetchData() {
                try {
                    await updateDates()
```

```
                    setNeedUpdate(!needUpdate)
                } catch (e) {
                    console.log(e);
                }
            }
                     if    (datesToUpdate.datesToAdd.length    ||
datesToUpdate.datesToDelete.length) {
                fetchData()
            }
        }, [datesToUpdate])

        useEffect(() => {

            async function fetch() {
                try {
                    await fetchDates()
                } catch (e) {
                    console.log(e);
                }
            }
            fetch()
        }, [])

        const selectHandler = (date) => {
            if (datesChanging) {
                const updatedData = { ...dates }
                if (updatedData[date]) {
                    delete updatedData[date];
                } else {
                          updatedData[date]  =  {  selected:  true,
selectedColor: Colors.red };
                }
                setDates(updatedData)
            }
        }

        const changeHandler = () => {
            setDatesChanging(!datesChanging)
            setPrevDates(dates)
        }

        const saveHandler = () => {
            setDatesChanging(!datesChanging)

                     const   {   deleteDates,   addDates   }   =
processDates(prevDates, dates)

                    setDatesToUpdate({   datesToAdd:   addDates,
datesToDelete: deleteDates })
        }

        useEffect(() => {
            if (timeExceptions) {
                const markedDates = {};
```

```
                timeExceptions.forEach((exception) => {
                     markedDates[exception.date.split('T')[0]] =
{ selected: true, selectedColor: Colors.red };
                });
                setDates(markedDates)
            }

        }, [timeExceptions])

        if (loading || loading2) return <Loader />
        return (
            <View>
                <UkrCalendar
                    markedDates={dates}
                    style={MyCalendarStyles.calendar}
                                        onDayPress={(day)    =>
selectHandler(day.dateString)}
                />
                {datesChanging
                            ? <MyButton style={{ backgroundColor:
Colors.green }} onPress={saveHandler} isRound>Зберегти</MyButton>
                            : <MyButton style={{ backgroundColor:
Colors.light              }}              onPress={changeHandler}
isRound>Змінити</MyButton>
                }
            </View>
        )
    }
```

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2024 р.

Мобільний застосунок для пошуку виконавців та замовників послуг на прикладі догляду за тваринами
Програма та методика тестування
КПІ.ІП-0127.045490.04.51

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Олександр ПАВЛОВ

Нормоконтроль:
_____ Максим ГОЛОВЧЕНКО

Виконавець:
_____ Даніл СМИСЛОВ

Київ – 2024

ЗМІСТ

1 ОБ’ЄКТ ВИПРОБУВАНЬ

Об’єктом випробування є мобільний застосунок для пошуку виконавців та замовників послуг на прикладі догляду за тваринами.

Основним функціоналом є частково автоматизований підбір виконавців послуг, фільтрація оголошень, що при підборі може враховувати графік завантаженості виконавця, сповіщення в режимі реального часу, функціонал адміністрування, робота з графіком завантаженості.

Застосунок має коректно працювати та відображатись на усіх сучасних підтримуваних мобільних пристроях з операційною системою iOS, починаючи з версії 13.4.

2 МЕТА ТЕСТУВАННЯ

Метою тестування є наступне:

– перевірка правильності роботи програмного забезпечення відповідно до функціональних вимог;

– перевірка збереження даних;

– перевірка коректності роботи застосунку на сучасних підтримуваних мобільних пристроях з операційною системою iOS, починаючи з версії 13.4;

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

– перевірка зручності графічного інтерфейсу;

– перевірка швидкодії програмного забезпечення.

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;

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

– модульне тестування - автоматизоване тестування окремих модулів програми в ізоляції від інших складових системи. Протестовано частину основного функціоналу програмного забезпечення, що відповідає за роботу з оголошеннями, користувачами та, частково, з заявками, поданими на оголошення.

4 ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Тестування виконується мануально та написанням unit-тестів, 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 роздільною здатністю екрану;

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

– тестування працездатності програми у випадку відсутності з’єднання до мережі;

– тестування інтерфейсу користувача;

– тестування зручності використання.

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2024 р.

Мобільний застосунок для пошуку виконавців та замовників послуг на прикладі догляду за тваринами

Керівництво користувача

КПІ.ІП-0127.045490.05.34

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Олександр ПАВЛОВ

Нормоконтроль:
_____Максим ГОЛОВЧЕНКО

Виконавець:
_____ Даніл СМИСЛОВ

Київ – 2024

ЗМІСТ

1 ПРИЗНАЧЕННЯ ПРОГРАМИ

Система пошуку виконавців та замовників послуг не прикладі догляду за тваринами [Pethome] – це мобільний застосунок для пошуку виконавців, що могли б доглянути за домашнім улюбленцем замовників. При цьому підборі враховується місцеположення користувачів та їх вільні дати.

Дана розробка призначена для широкого кола користувачів.

У відкритому доступі опубліковано зібраний імедж back-end частини застосунку. Згенеровано файли, що можуть використовуватись для розміщення в Play Market. Зібрати файли, що необхідні для розміщення в App Store, не вдалось через відсутність акаунта розробника Apple.

2 ПІДГОТОВКА ДО РОБОТИ З ПРОГРАМНИМ ЗАБЕЗПЕЧЕННЯМ

2.1 Системні вимоги для коректної роботи

Програмне забезпечення повинно функціонувати на сучасних пристроях з операційною системою iOS починаючи з версії 13.4.

Мінімальна рекомендована конфігурація технічних засобів:

– версія операційної системи iOS: 13.4;

– підключення до мережі Інтернет зі швидкістю від 5 мегабіт;

– об’єм вільної пам’яті: 50 Мб;

– об’єм оперативної пам’яті: можливість пристрою підтримки iOS версії 13.4;

– характеристики процесора: можливість пристрою підтримки iOS версії 13.4;

– роздільна здатність екрану: 750x1334 пікселів.

2.2 Завантаження застосунку

Необхідний для розгортання імедж серверної частини застосунку можна завантажити з Docker Hub.

Через вимогу наявності статусу розробника Apple для збірки інсталяційних файлів та розміщення додатку в крамниці застосунків, застосунок не розміщено в App Store. Проте за допомогою інструменту Expo можна використовувати його, відсканувавши QR-код.

2.3 Перевірка коректної роботи

У випадку розгортання – контейнери мають працювати, мають успішно відправлятись запити з клієнтської на серверну частину застосунку.

Після сканування QR-коду в застосунку Expo Go користувач має перейти на сторінку логіну для входу у застосунок. Якщо користувач не має акаунту – зможе зареєструвати його.

3 ВИКОНАННЯ ПРОГРАМИ

3.1 Функціонал користувача

Для початку користувачу потрібно зареєструвати новий профіль (рисунок 3.1). Потрібно ввести ім’я, прізвище, обрати стать. Прикріпити фото, ввести коректну адресу поштової скриньки, український номер телефону, унікальний логін, вказати пароль та підтвердити його. Якщо користувач надав доступ до місцеположення – буде доступно автоматично використати локацію. У протилежному випадку – ввести вручну та обрати з випадаючого списку.

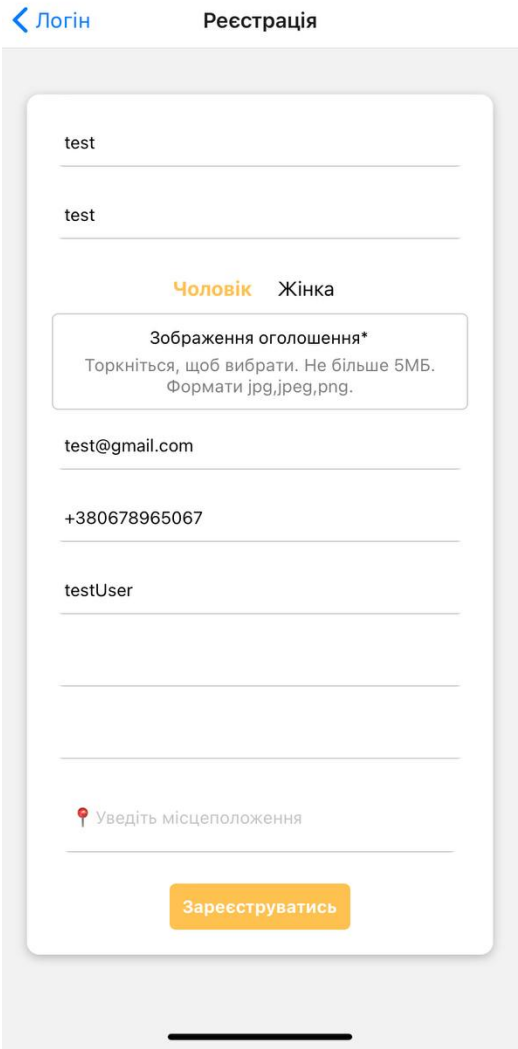


Рисунок 3.1 – Сторінка реєстрації

Далі потрібно увійти в акаунт, ввівши коректний логін та пароль (рисунок 3.2).

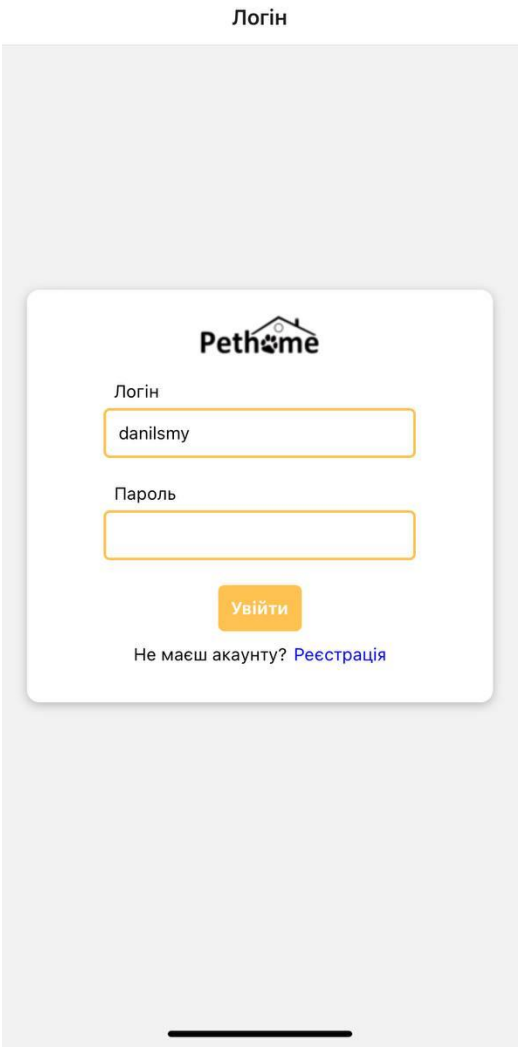


Рисунок 3.2 – Сторінка авторизації

Користувач спостерігає зареєстрований профіль (рисунок 3.3.).

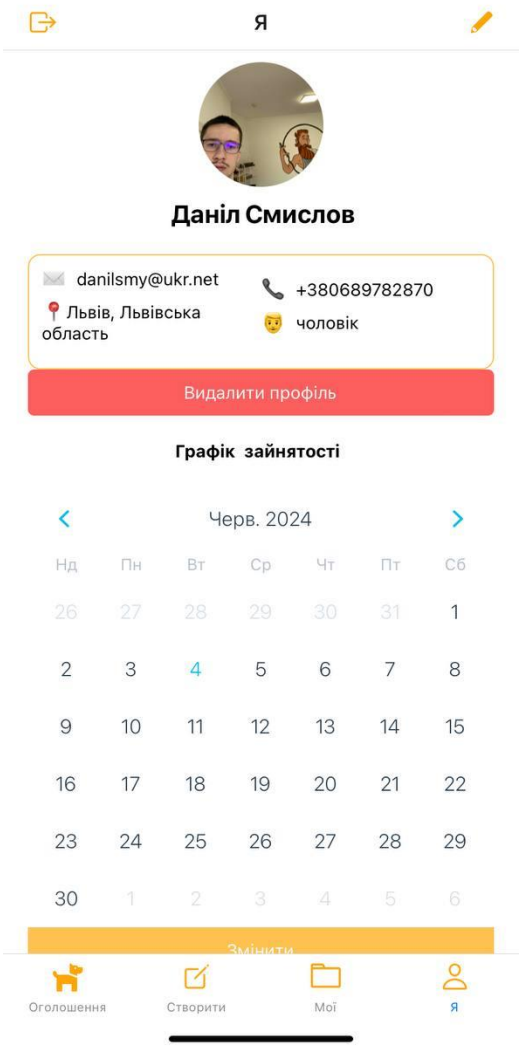


Рисунок 3.3 – Сторінка профілю користувача

Натиснувши кнопку зміни графіку завантаженості, користувач може додавати або видаляти заброньовані дати. Після цього потрібно натиснути кнопку збереження (рисунок 3.4).

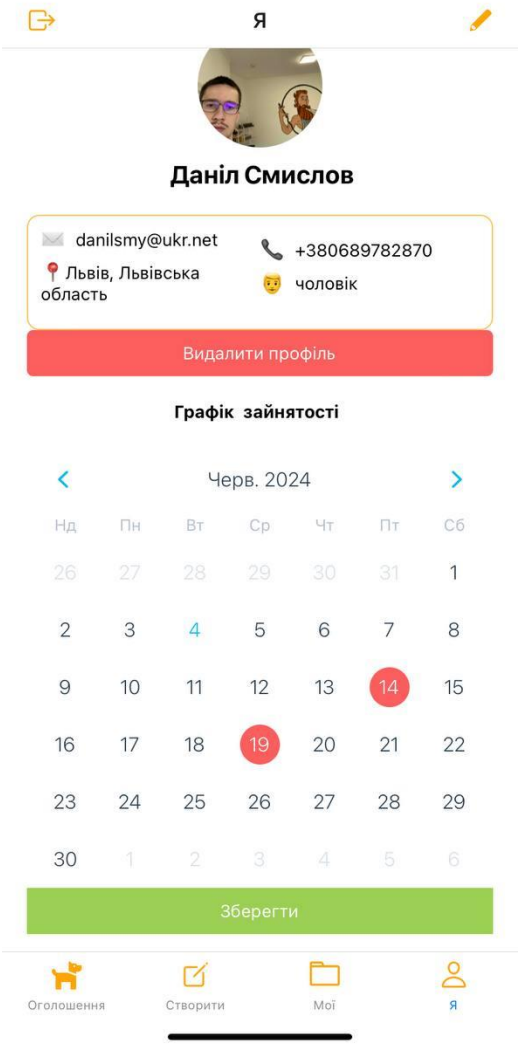


Рисунок 3.4 – Робота з графіком завантаженості

Оновлені дати збережено(рисунок 3.5).

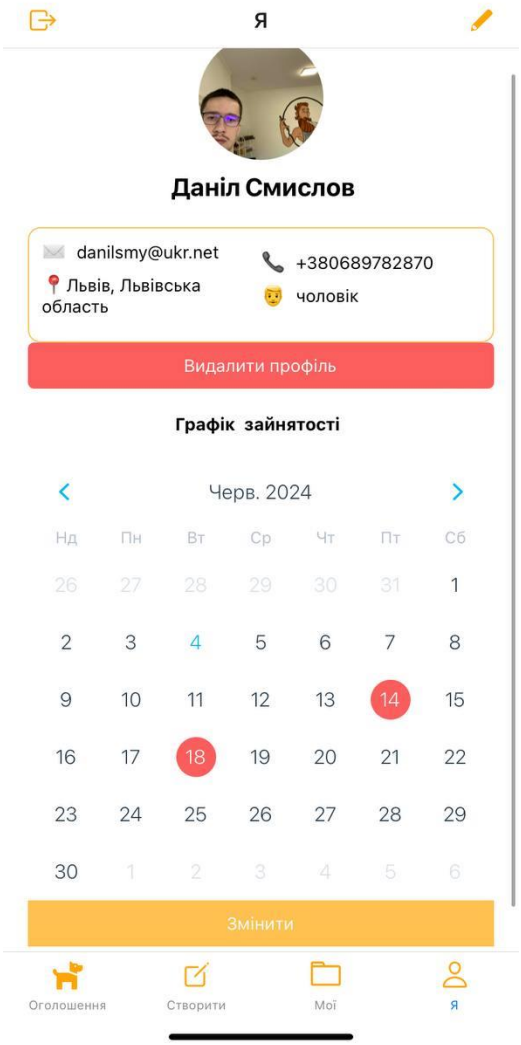


Рисунок 3.5 – Дати збережено

Користувач може змінювати персональну інформацію (рисунок 3.6), натиснувши в правому верхньому куту екрану на кнопку редагування.

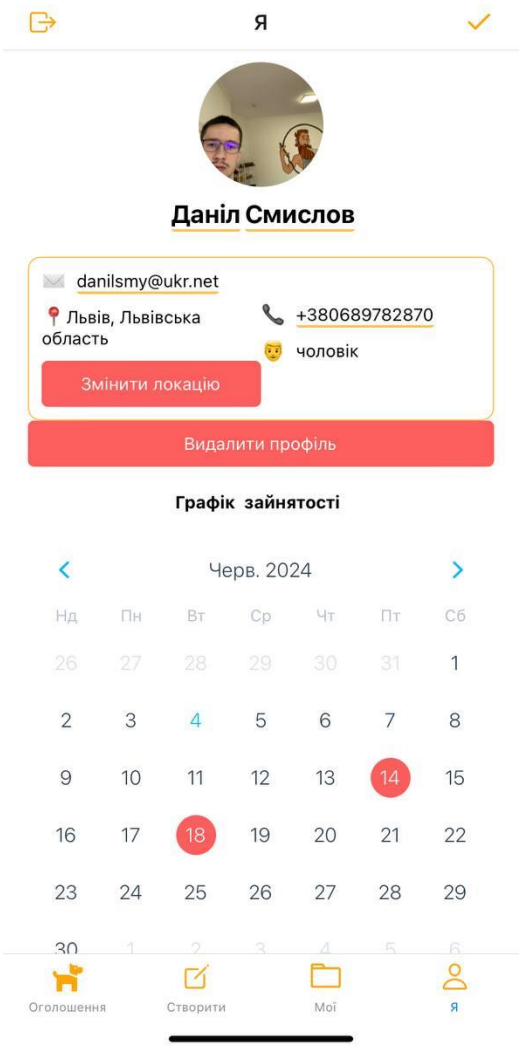


Рисунок 3.6 – Редагування профіля

Зокрема користувач може змінити місцеположення (рисунок 3.7).

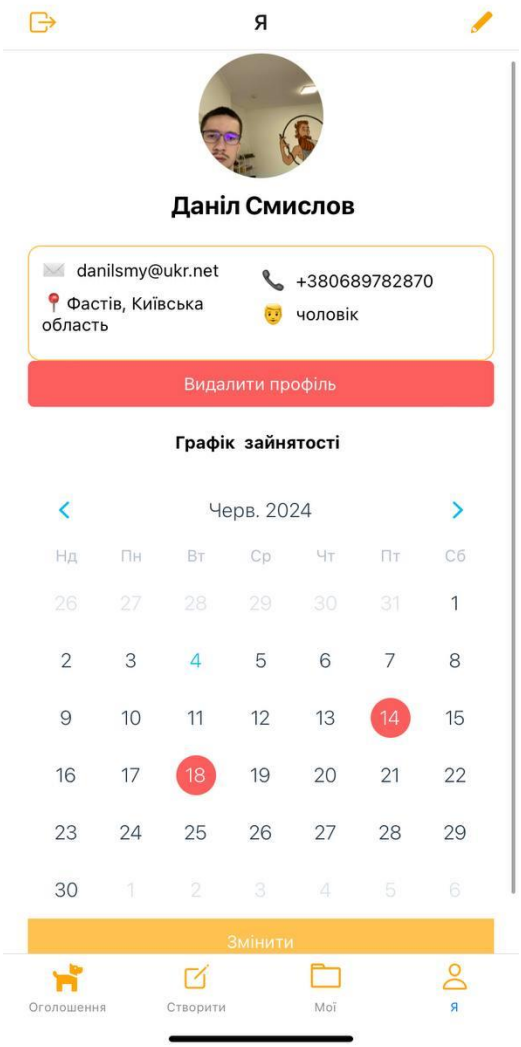


Рисунок 3.7 – Локацію змінено

При створенні оголошення користувач має ввести назву, опис, суму винагороди, обрати зображення, місцеположення (рисунок 3.8).

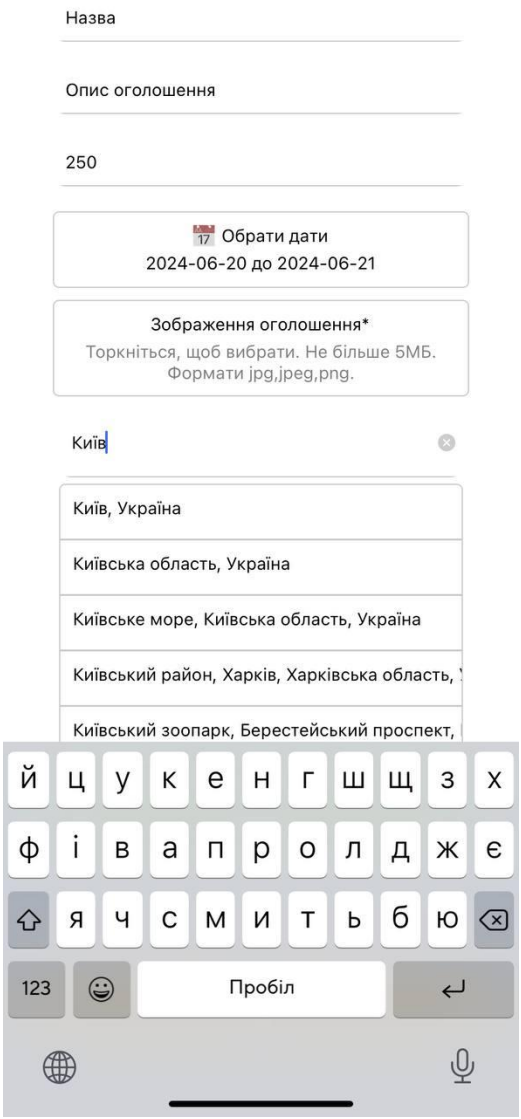


Рисунок 3.8 – Створення оголошення

Також користувачу при створенні оголошення потрібно обрати період виконання (рисунок 3.9).

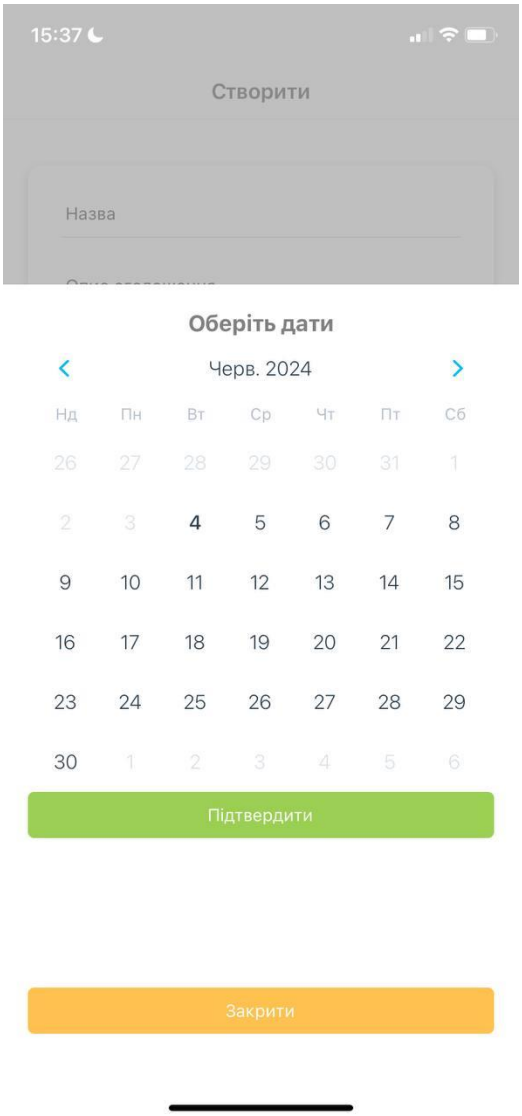


Рисунок 3.9 – Вибір періоду виконання оголошення

Після розміщення, оголошення відображається в списку оголошень, створених користувачем (рисунок 3.10).

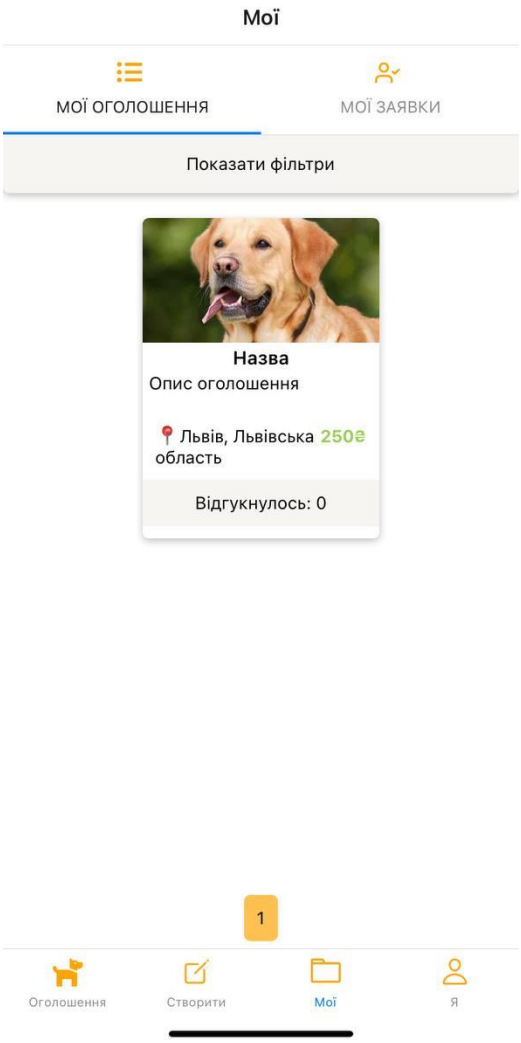


Рисунок 3.10 – Оголошення, створені користувачем

На рисунку 3.11 сторінка оголошення, створеного користувачем.

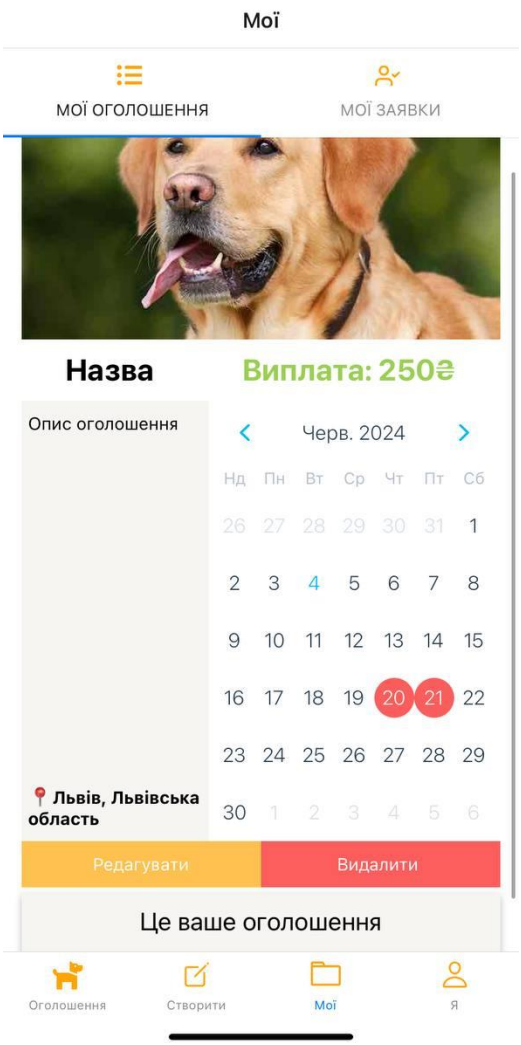


Рисунок 3.11 – Сторінка оголошення, створеного користувачем

Користувач може редагувати оголошення (рисунок 3.12). Зокрема змінювати період виконання, зображення, локацію.

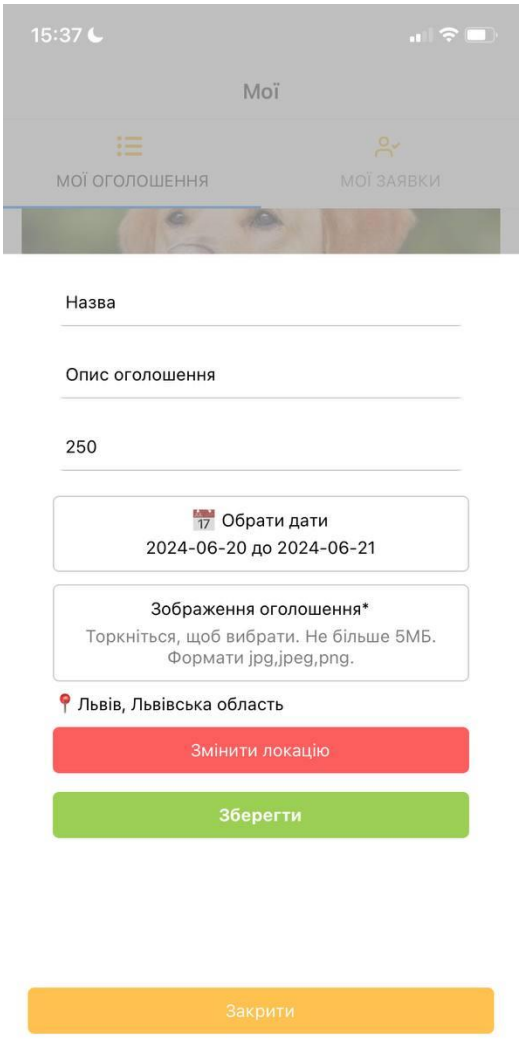


Рисунок 3.12 – Редагування оголошення

На головній сторінці з переліком оголошень користувач може налаштовувати фільтри. Зокрема «У Ваші вільні дати» (рисунок 3.13).

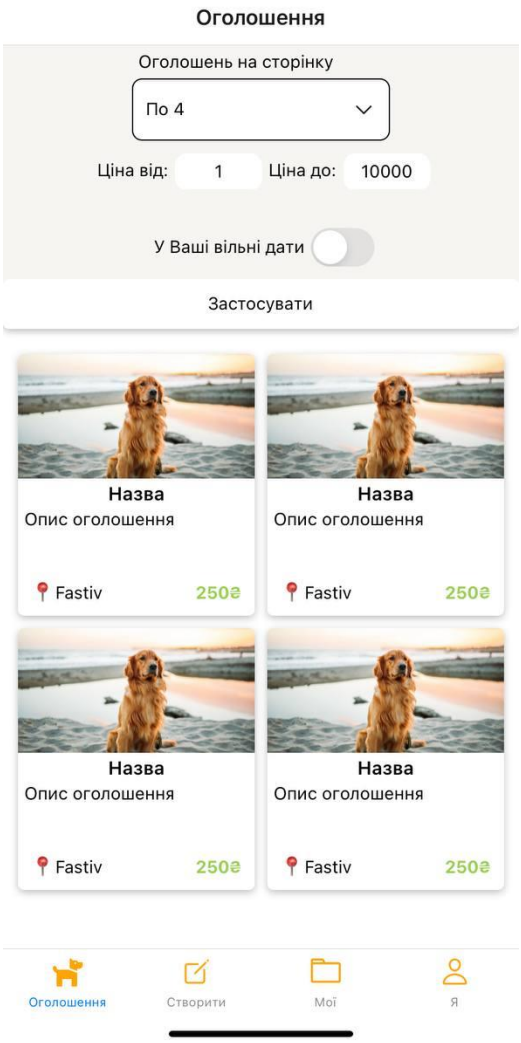


Рисунок 3.13 – Головна сторінка з переліком оголошень

Також користувач може обирати кількість оголошень, що представлено на сторінці (рисунок 3.14).

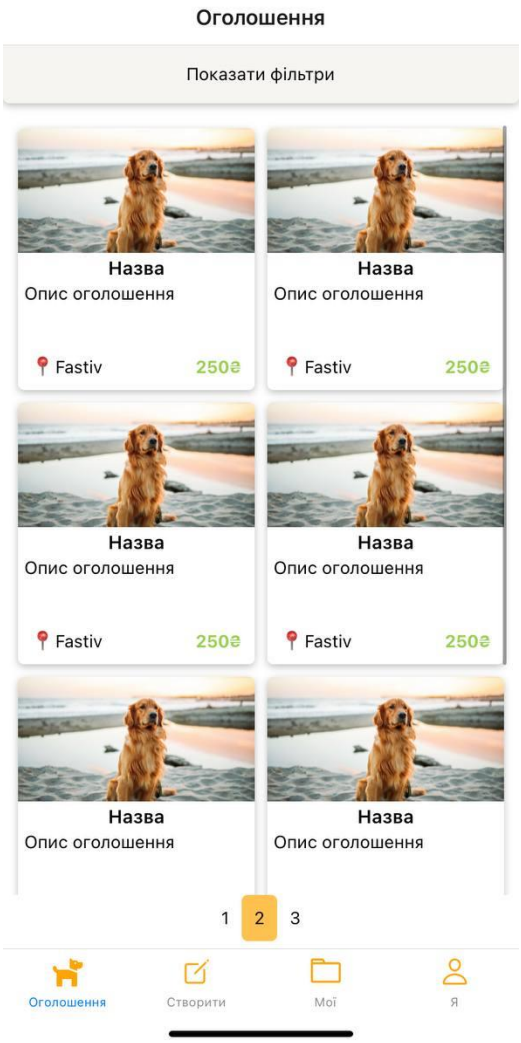


Рисунок 3.14 – Обрано кількість оголошень на сторінці «По 8»

Користувач може переглянути деталі оголошення (рисунок 3.15).

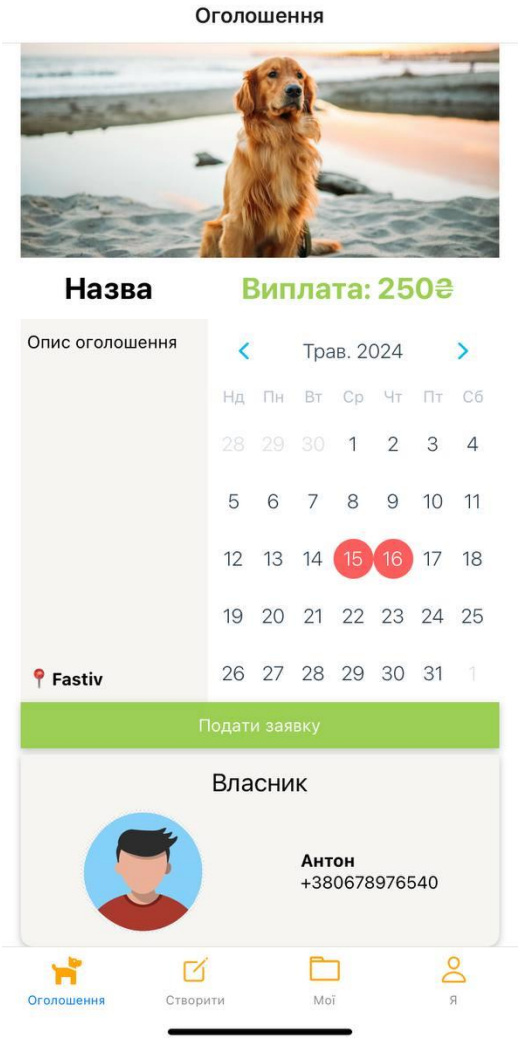


Рисунок 3.15 – Сторінка з деталями оголошення

Також може переглянути інформацію про власника (рисунок 3.16).

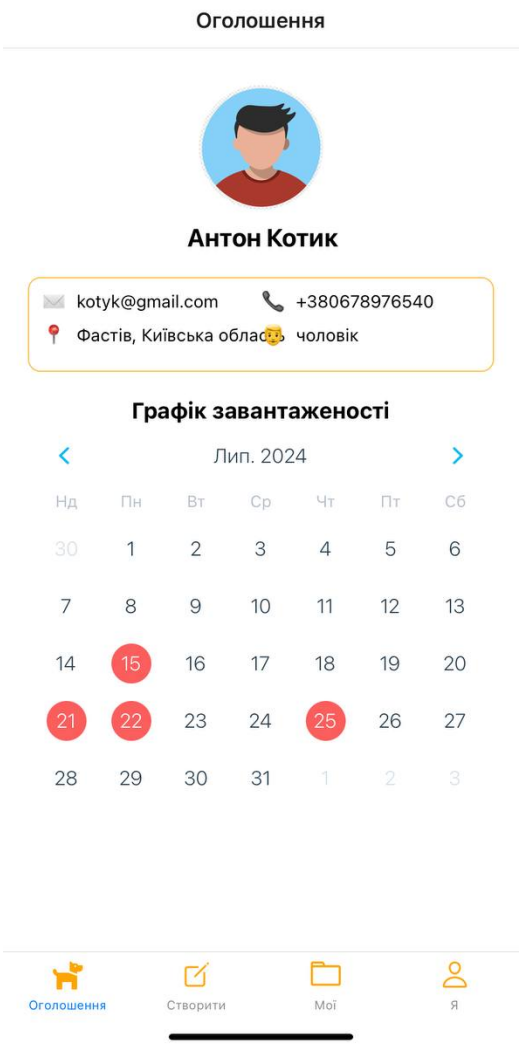


Рисунок 3.16 – Сторінка авторизації

Є можливість подати заявку, натиснувши на відповідну кнопку на сторінці оголошення. Тоді заявка буде відображатись на сторінці з заявками користувача (рисунок 3.17). Власнику оголошення прийде відповідне сповіщення.

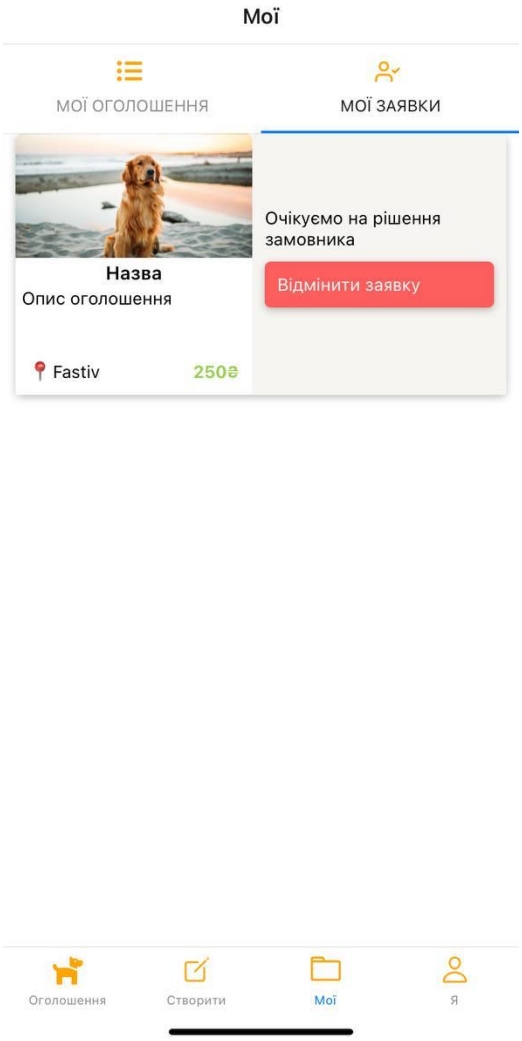


Рисунок 3.17 – Сторінка заявок користувача

У випадку якщо система підібрала оголошення для користувача, він отримає відповідне сповіщення та може перейти до переліку заявок (рисунок 3.18).

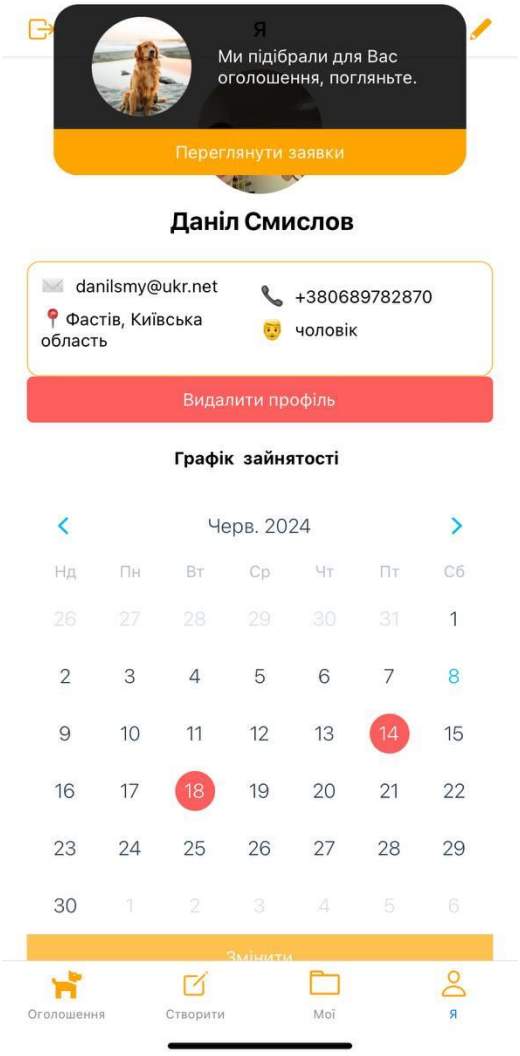


Рисунок 3.18 – Сповіщення про підбір оголошення

На сторінці з заявками користувача відображається пропозиції, згенеровані системою. Він може підтвердити, або відхилити їх (рисунок 3.19).

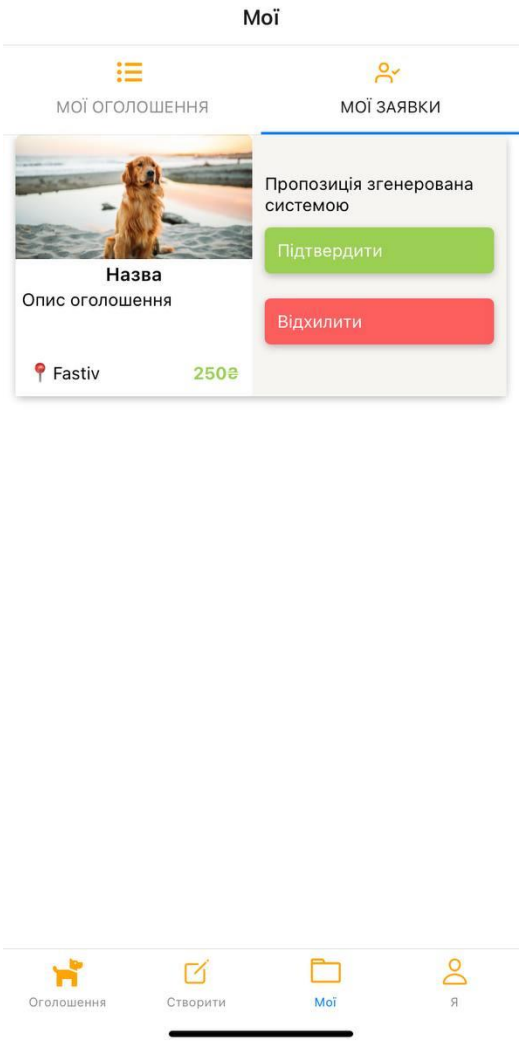


Рисунок 3.19 – Сторінка з заявками користувача

Власник оголошення бачить перелік можливих виконавців, що подали заявку на виконання оголошення (рисунок 3.20).

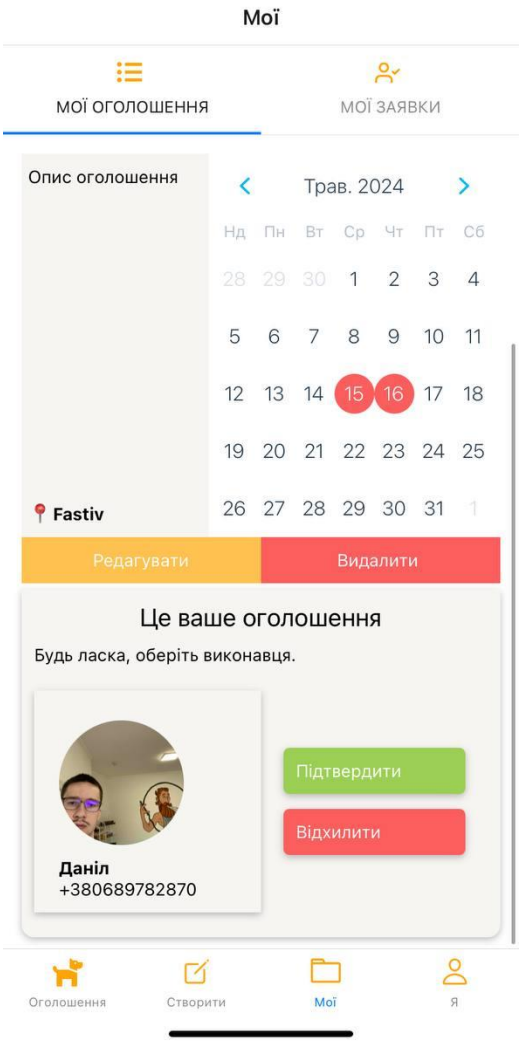


Рисунок 3.20 – Сторінка авторизації

Після вибору виконавця (рисунок 3.21), виконавець отримає сповіщення про це. Усі інші користувачі, що подавали заявки, отримують сповіщення про відхилення їх заявок. Після виконання оголошення власник може позначити оголошення як «виконане». Після цього зможе видалити оголошення.

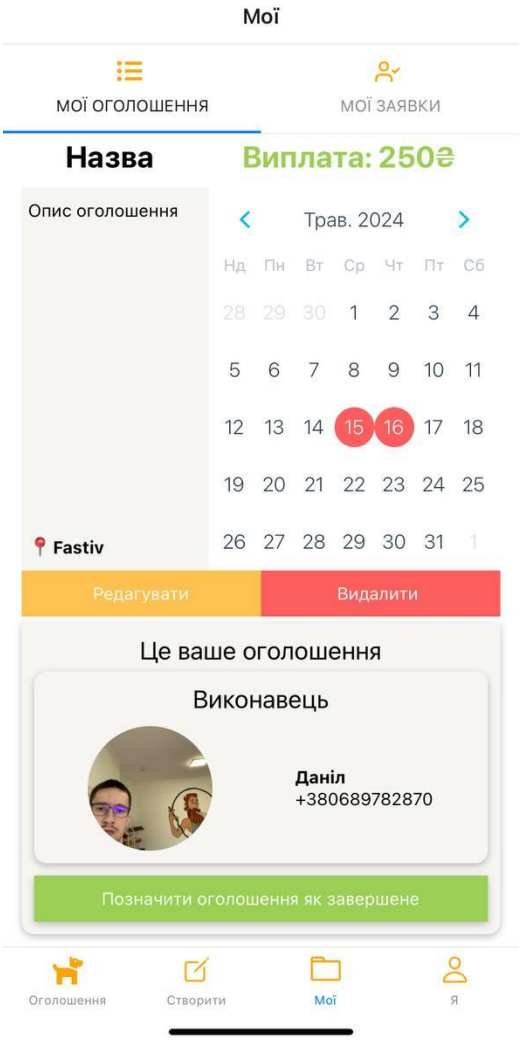


Рисунок 3.21 – Виконавця обрано

3.2 Функціонал адміністратора

Адміністратор застосунку може переглядати список оголошень користувачів (рисунок 3.22). Може видаляти оголошення.

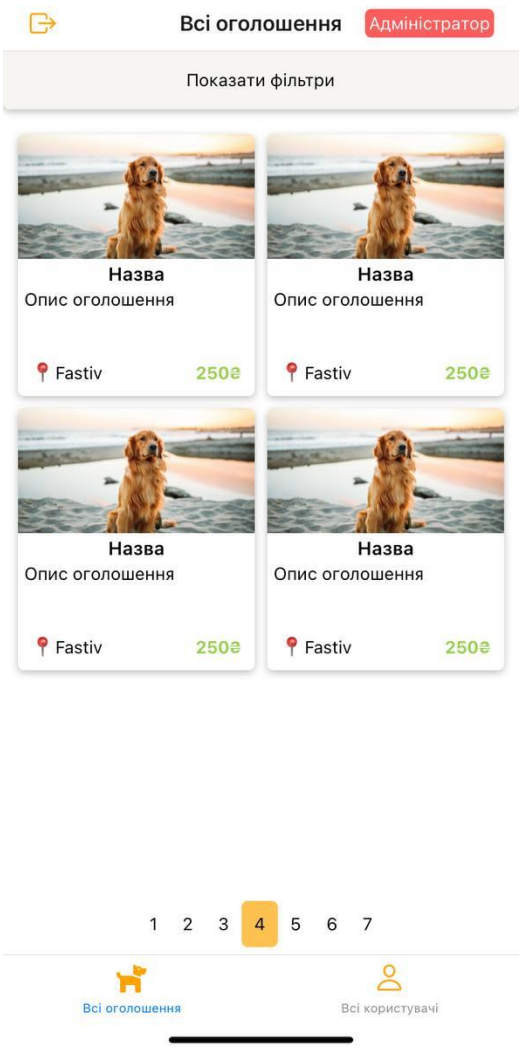


Рисунок 3.22 – Перегляд оголошень адміністратором.

Також може переглядати список користувачів застосунку (рисунок 3.23).

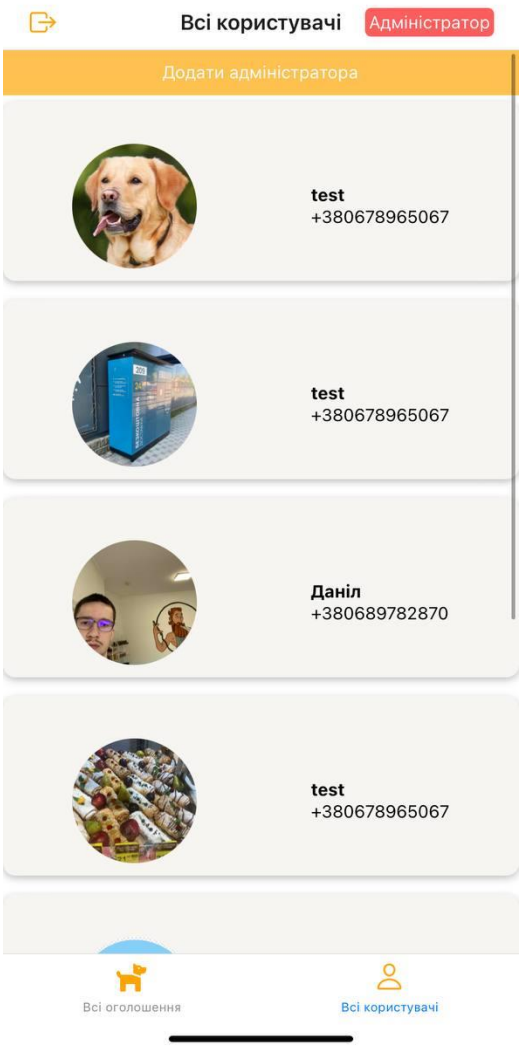


Рисунок 3.23 – Список користувачів застосунку

Також переглядати профіль конкретного користувача та видаляти його (рисунок 3.24).

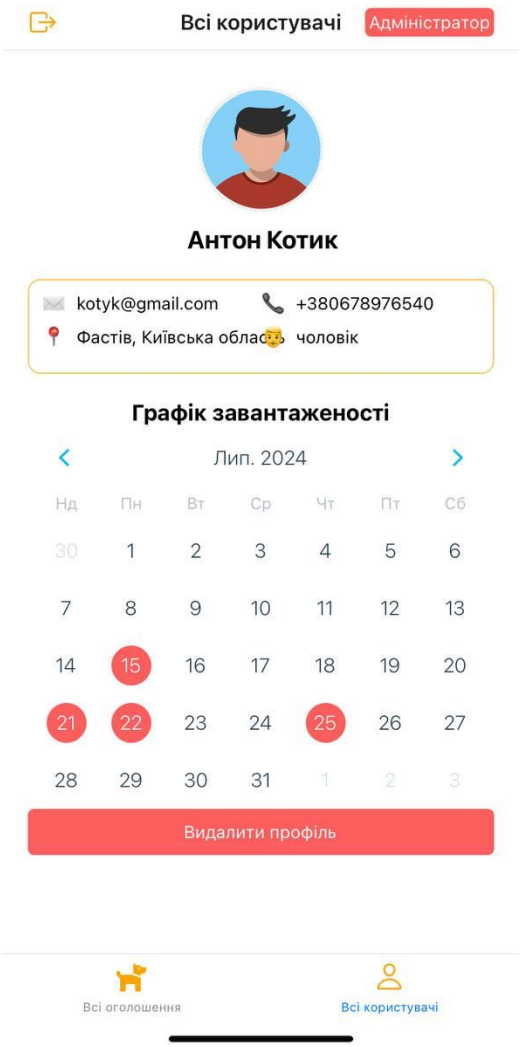


Рисунок 3.24 – Перегляд профілю користувача адміністратором

Також може додавати нових адміністраторів (рисунок 3.25).

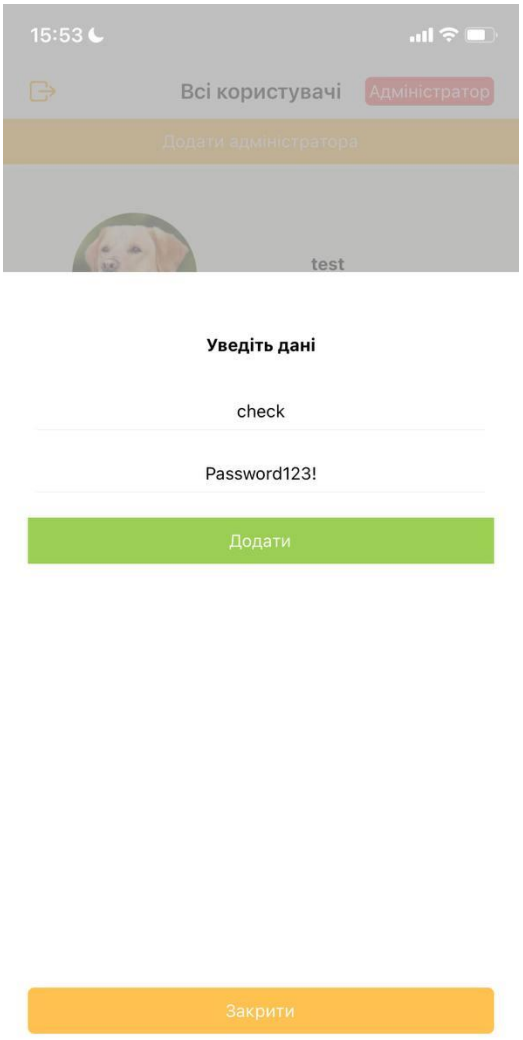


Рисунок 3.25 – Додавання нового адміністратора

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2024 р.

Мобільний застосунок для пошуку виконавців та замовників послуг на прикладі догляду за тваринами
Графічний матеріал
КПІ.ІП-0127.045490.06.99

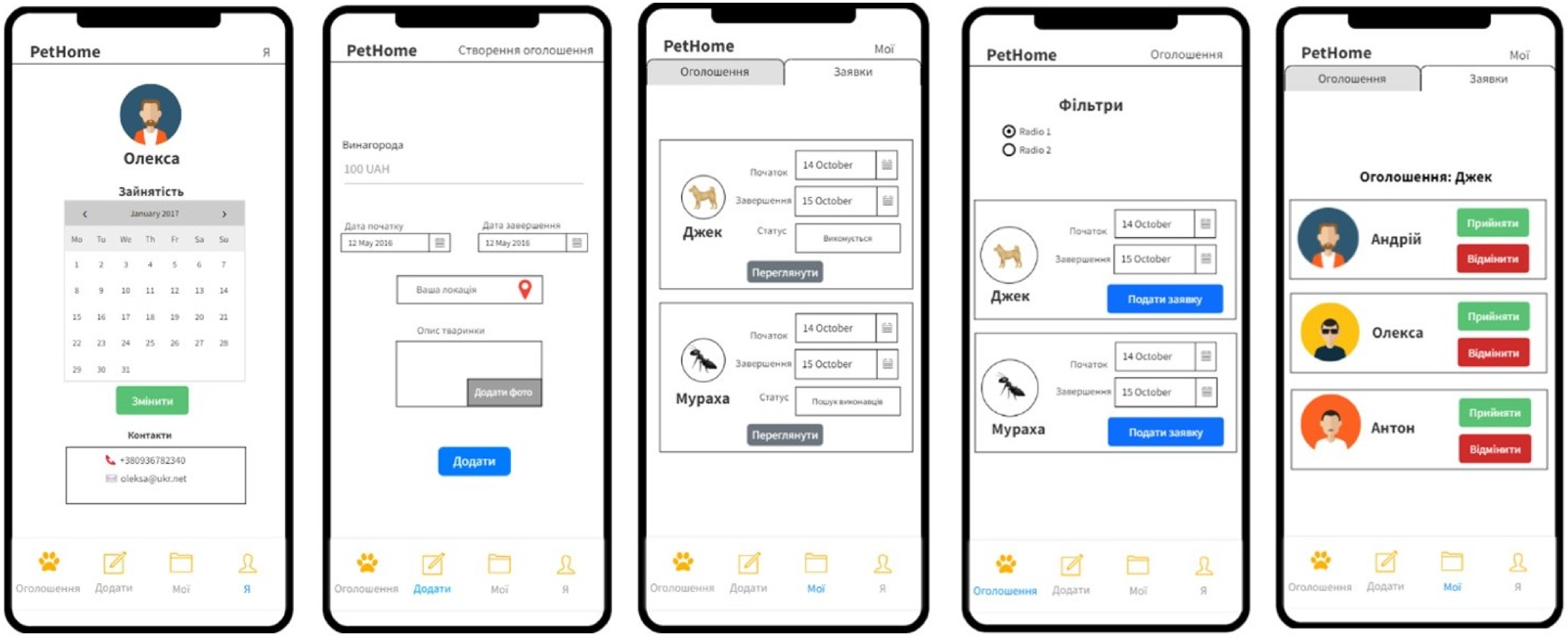
“ПОГОДЖЕНО”
Керівник проєкту:
_____ Олександр ПАВЛОВ

Нормоконтроль:
_____ Максим ГОЛОВЧЕНКО

Виконавець:
_____ Даніл СМИСЛОВ

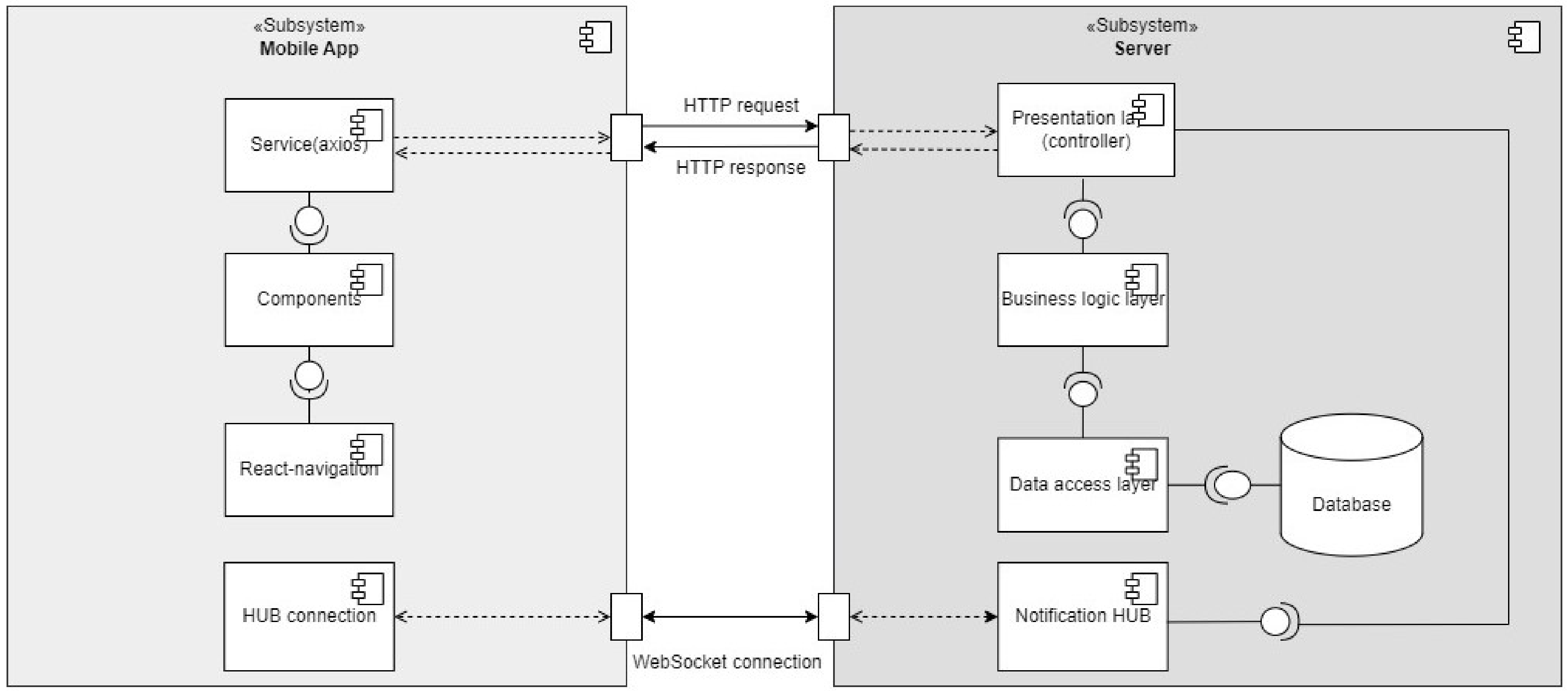
Київ – 2024

КПІ.ІП-0127.045490.06.99.КЕ

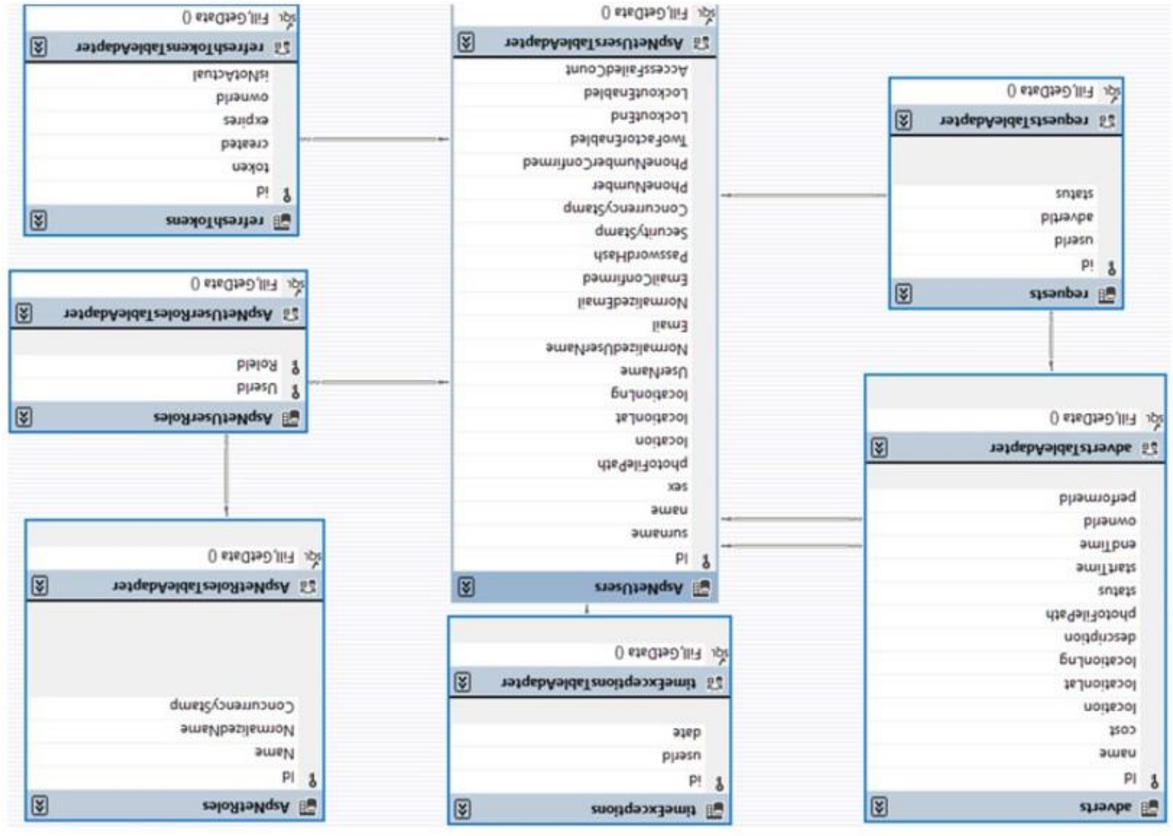


					КПІ.ІП-0127.045490.06.99.КЕ			
Зм.	Арк.	№ документа	Підпис	Дата		Літера	Маса	Масштаб
Розробив		Смислов Д.Ю.			Креслення вигляду екранних форм			
Перевірив		Павлов О.А.						
Т. контр.						Аркуш 1		Аркушів 1
Н. контр.		Головченко М.М.			Мобільний застосунок для пошуку виконавців та замовників послуг на прикладі догляду за тваринами	КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІП-01		
Затвердив		Жаріков Е.В.						

КПІ.ІП-0127.045490.06.99.ССМ



					КПІ.ІП-0127.045490.06.99.ССМ		
Зм.	Арк.	№ документа	Підпис	Дата	Схема структурна компонентів програмного забезпечення	Літера	Маса	Масштаб
Розробив		Смислов Д.Ю.						
Перевірив		Павлов О.А.				Аркуш 1	Аркушів 1	
Т. контр.								
					Мобільний застосунок для пошуку виконавців та замовників послуг на прикладі догляду за тваринами	КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІП-01		
Н. контр.		Головченко М.М.						
Затвердив		Жаріков Е.В.						



									КПІ.ІП-0127.045490.06.99.СБД	
Зм.	Арк.	№ документа	Підпис	Дата						
Розробив		Смислов Д.Ю.								
Перевірив		Ларнов О.А.			Схема бази даних					
Т. контр.										
Н. контр.		Голованюк М.М.								
Затвердив		Жарікова Е.В.								
					Мобільний застосунок для пошуку виконавців та замовників послуг на прикладі догляду за тваринами				КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІП-01	
						Літера	Маса	Масштаб		
						Аркуш 1		Аркушів 1		

КПІ.ІП-0127.045490.06.99.СБД