КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
імені ТАРАСА ШЕВЧЕНКА
Факультет інформаційних технологій

Кафедра прикладних інформаційних систем

122 «Комп'ютерні науки»

(шифр і назва спеціальності)

«Прикладне програмування»

(назва освітньої програми)

Кваліфікаційна робота бакалавра

на тему: «Мобільний застосунок з пошуку домашніх тварин»

Виконав _____
(Підпис)
Злобін Ілля Володимирович
(прізвище, ім'я, по батькові)

Керівник _Ващіліна Олена Валеріївна
(прізвище, ім'я, по батькові)

_____
(Резолюція «До захисту»)

Попередній захист:

_____
(Висновок: "До захисту в екзаменаційній комісії ")
Завідувач кафедри _____ Плескач В.Л.
(Підпис) (Прізвище, ініціали)
(Дата)

Київ – 2021

Київський національний університет
імені Тараса Шевченка
Факультет інформаційних технологій
Кафедра прикладних інформаційних систем

Назва теми: «Мобільний застосунок з пошуку домашніх тварин»

Освітня програма: Прикладне програмування

Спеціальність: Комп'ютерні науки

ПІБ	Підпис
Злобін Ілля Володимирович	

Назва роботи українською та англійською мовами

Мобільний застосунок з пошуку домашніх тварин A mobile application for pets searching

Мета бакалаврської кваліфікаційної роботи, завдання

Мета роботи: Створення зручного сервісу для організації пошуку домашніх тварин на основі Android застосунку, який забезпечить можливість подання та перегляду об'яв про загублених домашніх тварин, а також можливість пошуку господарів для тварин з притулків чи розплідників.

План роботи:

1. Методи та підходи в розробці мобільних застосунків на основі Android
2. Аналіз функціональних та архітектурних принципів для мобільних застосунків
3. Розробка та впровадження мобільного застосунку для пошуку домашніх тварин з використанням мов програмування TypeScript

ПІБ, ступінь, звання наукового керівника роботи: Ващіліна О.В., к.ф.-м.н., доцент

КАЛЕНДАРНИЙ ПЛАН ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ БАКАЛАВРА

	Назва етапів кваліфікаційноїроботи бакалавра	Термін виконання етапів кваліфікаційної роботи бакалавра	Відмітка про виконання
1.	Вибір теми та наукового керівника кваліфікаційноїроботи бакалавра	26.10.2020	виконано
2.	Видача завданнякваліфікаційної роботи бакалавра	23.11.2020	виконано
3.	Настановча групова співбесіда зпитань кваліфікаційної роботибакалавра	01.12.2020	виконано
4.	Затвердження кваліфікаційноїбакалавра плану роботи	18.02.2021	виконано
5.	Підбір та вивчення літературнихта інших джерел з теми дослідження	25.02.2021	виконано
6.	Підготовка і подання науковому керівнику першого варіанту І розділу роботи	05.03.2021	виконано
7.	Підготовка і подання науковому керівнику першого варіанту Ірозділу роботи	09.04.2021	виконано
8.	Підготовка і подання науковому керівнику першого варіанту ІІІ розділу роботи	07.05.2021	виконано
9.	Подання роботи у першому варіанті	11.05.2021	виконано
10.	Оформлення пояснювальноїзаписки кваліфікаційної роботи бакалавра	12.05.2021	виконано
11.	Подання кваліфікаційної роботи бакалавра на попередній захист	**24.05.2021**	виконано
12.	Врахування зауваженькерівника і подання роботи в остаточному варіанті (з відповідним висновком про допуск) на кафедру	28.05.2021	виконано
	Затвердження роботи в цілому (підготовка письмового відгуку керівника, письмова рецензія на бакалаврської роботу)	11.06.2021	
14.	Захист кваліфікаційної роботибакалавра	22.06.2021	

ВІДОМІСТЬ ДИПЛОМНОЇ РОБОТИ

Складові частини дипломної роботи	Обсяг, арк.
Титульний аркуш	1
Завдання до дипломної роботи	1
Календарний план дипломної роботи	1
Відомість дипломної роботи	1
Анотація	1
Анотація (іноземною мовою-англійською)	1
Зміст	1
Перелік скорочень, умовних позначень, термінів	2
Вступ	
1	
2	
3	
Висновки	
Перелік використаних джерел	
Додатки	

				ДП XXXX 00.000.00		
	ПІБ	Підп.	Дата			
Розробн.	Злобін І.В			Відомість дипломної роботи	Лист	Листів
Керівн.	Ващіліна О.В					
Н/контр.						
Зав.каф.	Плескач В.Л.					

АНОТАЦІЯ (РЕФЕРАТ)

Дипломна робота:

Ця дипломна робота присвячена проектуванню та розробленню мобільного застосунку для пошуку домашніх тварин.

Метою дипломної роботи є створення онлайн-сервісу для організації пошуку домашніх тварин на основі Android-застосунку.

Для досягнення поставленої мети треба вирішити такі **завдання**:

1. Проаналізувати методи та підходи в розробці мобільних застосунків на основі Android.
2. Здійснити аналіз функціональних та архітектурних принципів для мобільних застосунків.
3. Спроектувати та розробити та впровадити мобільний застосунок для пошуку домашніх тварин з використанням мови програмування TypeScript.

Об’єкт дослідження. Процеси розшуку загублених домашніх тварин, а також пошуку господарів для тварин з притулків чи розплідників з використанням інформаційних технологій.

Предмет дослідження. Програмо-технічні підходи та принципи побудови мобільного застосунку для подачі та перегляду об’яв про домашніх тварин, який забезпечить можливість їх ефективного пошуку.

Методи дослідження. Системний аналіз, теорію розробки та впровадження клієнт-серверної архітектури на прикладі мобільного застосунку, аналіз ефективності використання сучасний застосунків для пошуку тварин. Результати дипломної роботи можна розміщувати в магазині застосунків, що дасть змогу завантажувати застосунок іншим користувачам для подальшого використання. Застосунок був розроблений за допомогою мови TypeScript, платформи Ionic та фреймворка Nest на серверній частині, для зберігання даних була використана база даних MySql.

Ключові слова: мобільний застосунок, пошук тварин, платформа Ionic, операційна система Android

ANOTATION (ABSTRACT)

Thesis:

This thesis is dedicated to the design and development of a mobile application for finding pets.

The purpose if the thesis is the creation of an online service for organizing the search for pets based on the Android application.

To archive the goal of the work you need to solve the following **tasks**:

1. Analyze methods and approaches in the development of mobile applications based on Android.
2. Analyze functional and architectural principles for mobile applications.
3. Design and develop and implement a mobile application for finding pets using the TypeScript programming language.

Object of study. Processes of searching for lost pets, as well as finding owners for animals from shelters or nurseries using information technology.

Subject of study. Software and technical approaches and principles for building a mobile application for submitting and viewing ads about pets, which will provide the ability to effectively search for them.

Research methods. System analysis, theory of development and implementation of client-server architecture on the example of a mobile application, analysis of the effectiveness of modern applications for animal retrieval. The results of the thesis can be placed in the application store, which will allow you to download the application to other users for future use. The application was developed using the TypeScript language, the Ionic platform and the Nest framework on the server side, and the MySql database was used to store the data.

Keywords: Mobile application, pet searching, Ionic platform, Android system.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ANDROID – операційна система, яка використовується на більшості мобільних пристроїв та планшетах.

AJAX (Asynchronous JavaScript and XML) – підхід в розробці інтерактивних веб-застосунків, в основі якого лежить обмін даними між браузером та сервером.

BLUETOOTH – технологія бездротової передачі даних

CLI (Command-line interface) – інтерфейс командного рядка, який обробляє команду до певної комп'ютерної програми у вигляді тексу.

CSS (Cascading Style Sheets) – спеціалізована мова для стилізації елементів веб-сторінок.

DART – оптимізована мова програмування для розробки застосунків, основна мета яких – висока продуктивність.

FLUTTER – портативний набір інтерфейсних інструментів для створення мобільних застосунків.

FRAMEWORK – платформа для розробки програмних застосунків, яка забезпечує основу, на якій розробники ПЗ можуть створювати програми для певних платформ.

GPS (Global Positioning System) – навігаційна система, що використовує супутники, приймач та алгоритми для синхронізації даних про місцезнаходження, швидкість та час для повітряних, морських та наземних подорожей.

HTML (HyperText Markup Language) – мова тегів, якою пишуться гіпертекстові документи для мережі Інтернет.

HTTP (Hyper Text Transfer Protocol) – протокол передачі даних, що використовується в мережі інтернет.

JS (JavaScript) – не типізована мова програмування, яка використовується для розробки веб-застосунків.

SDK (Software Development Kit) – певний набір інструментів, які можна використовувати для розробки програмного забезпечення.

SPA (Single Page Application) – сайт або веб-застосунок, який взаємодіє з користувачем шляхом динамічного завантаження нових даних, замість того, щоб завантажувати нову сторінку.

SQL (Structured query language) – мова структурованих запитів.

TS (TypeScript) – мова програмування з відкритим вихідним кодом, яка базується на мові JavaScript та підтримує систему типів.

URI (Uniform Resource Identifier) – унікальна послідовність символів, яка ідентифікує логічний або фізичний ресурс у мережі інтернет.

URL (Uniform Resource Locator) – посилання на веб-ресурс, який визначає його місцезнаходження в комп’ютерній мережі та механізм його отримання.

REST (Representational State Transfer) – архітектурний стиль для взаємодії компонентів розподіленого застосунку в мережі.

БД – база даних.

ПЗ – програмне забезпечення.

МП – мова програмування.

СУБД – система управління базами даних.

ВСТУП

Актуальність дипломної роботи витікає із проблем які пов’язані з тваринами та поганим ставленням до них з боку людей. Проблема безпритульних тварин в нас час є досить важливою.

В сучасному світі, мобільні телефони та застосунки до них відіграють важливу роль в житті людей. Сучасна людина може отримувати безліч інформації кожної хвилини, лише тримаючи в руках телефон. Мобільні застосунки дозволять об’єднувати людей з схожими інтересами, що в свою чергу допомагає вирішувати проблеми великих масштабів.

Практична частина дипломної роботи представлятиме собою проектування та розроблення мобільного застосунку для пошуку домашніх тварин, який матиме сторінки для реєстрації та авторизації, додавання власних улюбленців, створення оголошень та їх перегляду.

Мета бакалаврської роботи: створення онлайн-сервісу для організації пошуку домашніх тварин на основі Android-застосунку.

Завдання дипломної роботи:

4. Аналіз Методів та підходів в розробці мобільних застосунків на основі Android.
5. Аналіз функціональних та архітектурних принципів для мобільних застосунків.
6. Розробка та впровадження мобільного застосунку для пошуку домашніх тварин з використанням мов програмування JavaScript та TypeScript

Об’єкт дослідження: Процеси розшуку загублених домашніх тварин, а також пошуку господарів для тварин з притулків чи розплідників з використанням інформаційних технологій.

Предмет дослідження: Програмо-технічні підходи та принципи побудови мобільного застосунку для подачі та перегляду об’яв про домашніх тварин, який забезпечить можливість їх ефективного пошуку.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

У сучасну епоху інформаційних люди звикли користуватися комп'ютером, але ринок мобільних додатків – це швидко зростаючий сектор. Багато людей використовує мобільні застосунки для зв’язку з друзями, перегляду сайтів, управління власними справами, розваги тощо. Люди можуть робити багато речей з свого повсякденного та ділового життя не ховаючи телефон до кишені. Мобільні застосунки мають вплив не тільки на кінцевого користувача, так і на бізнес в цілому. Багато компанії отримують дохід, використовуючи мобільні застосунки.

1.1 Поняття мобільного застосунку

Мобільний застосунок – це програмне забезпечення, яке працює на мобільному пристрої та виконує певні завдання для користувача. Мобільні застосунки – це новий і швидко розвиваючий сегмент глобальних інформаційних та комунікаційні технології. Такі програмні продукти не займаються багато пам’яті, зручні, недорогі, здатні завантажуватись та запускатись на великій кількості мобільних телефонів, включаючи недорогі телефони та телефони початкового рівня. Мобільний застосунок має широкий спектр використання, наприклад: дзвінки, обмін повідомленнями, перегляд відео, чат, спілкування в соціальних мережах, прослуховування аудіо, ігри тощо. Більшість застосунків встановлена в телефони за замовчуванням, як правило їх вистачає для буденних справ, але інші застосунки ви можете завантажити з інтернету.

Залежно від області застосування існують різні категорії мобільних застосунків:

1. Ігрові – це найпопулярніша категорія мобільних застосунків. Підприємства інвестують все більше часу та ресурсів у створення ігор та мобільних версій відомих стаціонарних ігор, оскільки це прибутковий ринок. Згідно з недавнім дослідженням, на мобільні ігри припадає 33% усіх завантажень додатків, 74% споживчих витрат і 10% всього часу, витраченого на використання

програм. Найуспішніші є такі мобільні ігри: Candy Crush Saga, Angry Birds, Clash Of Clans, Rise of Kingdoms та інші.

2. Програми для бізнесу або продуктивності – ці програми займають сьогодні велику частину ринку, оскільки люди все частіше схильні використовувати свої смартфони та планшети для виконання багатьох складних завдань на ходу. Наприклад, програми можуть допомогти їм забронювати квитки, надсилати електронні листи або відстежувати їхній хід роботи. Бізнес-програми спрямовані на підвищення продуктивності та мінімізацію витрат, оскільки дозволяють користувачам виконувати широкий спектр завдань, починаючи від придбання нових картриджів для офісних принтерів і закінчуючи набором нового офісного менеджера.

3. Освітні програми – до цієї категорії належать мобільні застосунки, які допомагають користувачам здобувати нові навички та знання. Наприклад, програми для вивчення мови, такі як Duolingo або Lingualeo, стали неймовірно популярними, оскільки надають користувачам гнучкість, яку вони шукають у навчанні. Навчальні ігрові програми – це чудовий інструмент для дітей. Багато освітніх програм виявляються популярними і серед викладачів, які використовують їх для кращої організації навчального процесу або подальшого навчання.

4. Лайфстайл програми – ця широка категорія додатків охоплює магазини, моду, віртуальні примірки, тренування, побачення та дієтичні програми. Ці програми в основному фокусуються на різних аспектах особистого життя людей.

5. Комерційні програми – найпопулярніші торгові програми, такі як Amazon або eBay, пропонують мобільним користувачам досвід своїх версій для настільних ПК. Програми мобільної комерції надають клієнтам зручний доступ до продуктів та безперебійні способи оплати для оптимального досвіду покупок

6. Розважальні програми – це програми, які дозволяють користувачам транслювати відео вміст, шукати події, спілкуватися в чаті або переглядати веб сайти в Інтернеті. Програми для соціальних мереж, такі як Facebook або

Instagram, є чудовими прикладами. Більше того, програми для потокового відео, такі як Netflix або Amazon Prime Video, стали неймовірно популярними серед користувачів у всьому світі. Ці програми зазвичай сприяють залученню користувачів, повідомляючи учасників про оновлення та нещодавно додані продукти.

7. Службові програми – це програми які призначені для повсякденного користування. Насправді у службових програм зазвичай є найкоротший час сеансу для користувача – люди використовують їх, щоб щось робити, а потім рухатись далі. Найпопулярніші типи службових програм – це сканери штрих-коду, трекери або програми для охорони здоров’я.

8. Програми для подорожей, основна ідея цієї категорії – це допомогти користувачам легко подорожувати. Програми для подорожей перетворюють смартфон або планшет на щоденник подорожей та довідник, який допомагає користувачам дізнатися все, що їм потрібно знати про сайт, який вони відвідують. Більшість туристів - це цифрові кмітливі мандрівники, які знають, як використовувати програми на свою користь. Важко уявити, як би виглядали подорожі без Google Maps, Airbnb або Uber.

1.2 Розвиток та розробка мобільних застосунків

На початку 21 століття розробники мобільних застосунків почали обговорення стосовно інтернет застосунків для мобільних пристроїв. Для того щоб люди мали змогу вільно під’єднуватись до мережі інтернет для вирішення повсякденних справ. В цей час, європейські компанії «Sony Erixon» і «Nokia» заснували свої штаб квартири в яких народились мобільні інновації. Головна проблема в тому, що дані компанії робили мобільні телефони для операторів мобільного зв’язку. Але в деяких місцях мобільна індустрія відстає, тому що в таких місцях мобільний зв’язок не є розвиненим.

В Америці, 2007 року, компанія Apple оголосила, що випускає мобільний телефон, який назавжди змінить ринок мобільних телефонів. Компанії було байдуже на запити мобільних операторів, вони хотіли створити телефон та операційну систему, які будуть зручні для користувача, і нарешті вони це зробили. Після випуску iPhone, він став найбільш популярним телефоном у світі, а інші компанії розробники телефонів до сьогодення намагаються зробити щось схоже. Головними особливостями iPhone були: зручність використання, відсутність кнопок та операційна система iOS. Дана операційна система дала змогу запускати найбільш динамічні програми та ділитись ними в мережі, це призвело до революції в сфері мобільних застосунків. Система iOS має майданчик з застосунками, де можна знайти будь-яку програму під власні потреби, крім того це чудове місця для розробників.

Розробка мобільних застосунків – це процес, який багато що черпає з традиційної розробки програмного забезпечення. Однак вона зосереджена на створенні програмного забезпечення, яке використовує унікальні особливості апаратного забезпечення мобільних пристроїв.

Найбільш простим сценарієм створення мобільного додатка є використання настільного додатка та імпорт його на мобільний пристрій. Однак, оскільки додаток стає більш надійним, цей прийом може стати проблематичним.

Кращий підхід передбачає розробку спеціально для мобільного середовища. Це техніка, яка використовує всі переваги мобільних пристроїв. Процес враховує їх обмеження та допомагає власникам підприємств збалансувати витрати та функціональність.

Наприклад, додатки, які використовують такі функції, як карти, завжди створюються з нуля з урахуванням мобільних пристроїв. Послуги, що базуються на розташуванні, що надаються в настільному додатку, мають менший сенс, оскільки користувачі настільних ПК не рухаються.

Сучасні смартфони та планшети оснащені такими функціями, як Bluetooth, зв’язок ближнього поля (NFC), GPS, гіроскопічні датчики, камери та багато іншого. Розробники можуть використовувати ці функції для створення застосунків з такими технологіями, як віртуальна або доповнена реальність, сканування штрих-коду, послуги на основі місцезнаходження та багато іншого. Найуспішніші та найпопулярніші мобільні застосунки використовують функції смартфона найкращим чином.

Питання апаратного забезпечення мобільних пристроїв створює ще одну складність. Хоча розробники, які створюють додатки для iOS, можуть розраховувати, що програми працюватимуть лише на двох типах пристроїв (iPhone та iPad), розробники Android не можуть сказати те саме. Насправді, для них кожен смартфон і планшет може працювати на різному обладнанні та різних версіях операційної системи.

1.3 Архітектура мобільних застосунків

Мобільний застосунок складається з різних компонентів. Важливо розуміти різні компоненти, щоб спроектувати і розробити ефективний мобільний застосунок. Знання архітектури для різних видів мобільних пристроїв і їх застосування допоможуть обрати найбільш підходящі методи розробки мобільного застосунку. Архітектура мобільного застосунку також дає уявлення про дизайн користувальницького інтерфейсу, аспекти інтеграції та інше. Загалом, мобільні застосунки можна розділити на три категорії:

1. Мобільні веб-застосунки на основі браузера. Це так звані веб-застосунки, які розроблені з використанням загальноприйнятих методів розробки веб-дизайну та можуть обслуговувати різноманітні пристрої.

2. Нативні мобільні застосунки. Ці застосунки розроблені для конкретних мобільних платформ (наприклад iOS або Android), які в свою чергу можуть повністю використовувати можливості пристрою. Зазвичай, нативні застосунки збираються за допомогою SDK, які надає платформа.

3. Гібридні застосунки. Дані застосунки можуть бути розроблені за допомогою веб-технологій, наприклад мов програмування JavaScript або TypeScript, також можуть використовувати можливості пристрою за допомогою різноманітних інструментів розробки.

Таблиця 1.1 – Основні відмінності між трьома типами мобільних застосунків

	Веб-застосунки на основі браузера	Нативні застосунки	Гібридні застосунки
Технологія розробки	Зазвичай, використовують популярні технології розробки веб-дизайну за допомогою медіа запитів CSS3	Розробляються за допомогою мобільних платформ iOS та Android	Розробляються з використанням веб-фреймворків (React native, Ionic). Можуть використовуватись на любій нативній платформі
Особливості	Використовуються тільки в оточенні інтернет браузера. Мають більш гнучкий дизайн та широкі можливості для його модифікації	Можуть повністю використовувати всі можливості пристрою. Забезпечує кращий користувацький досвід	Можуть розроблятися за допомогою різноманітних інструментів.
Сумісність	Працють з мобільними пристроями на будь яких платформах	Працює на специфічній мобільній платформі	Працює на різноманітних мобільних платформах
Переваги	Малі затрати на розробку	Висока продуктивність	Малі затрати на розробку
Продуктивність	Низька	Оптимальна	Середня
Розробка	Швидка та невисока вартість	Висока вартість	Відносно низька вартість

1.4 Аналіз існуючих рішень

Перед початком роботи над застосунком, було проаналізовано ряд існуючих застосунків для пошуку тварин. Основна задача при аналізі – виділити основні функції застосунків та їхні переваги та недоліки.

Застосунок «Pet adoption». Основна ідея застосунку – зберігання анкет домашніх тварин. Головну сторінка застосунку з списком найближчих до мене тварин зображена на рисунку 1.1. Реєстрація проходить за допомогою номеру телефону користувача, на який приходить код з підтвердженням. Варто зазначити, що відсутня можливість входу та реєстрації за допомогою соціальних мереж. Також відсутня можливість змінювати данні користувача, скинути або відновити пароль, відсутність даних можливостей може негативно впливати на безпеку самого застосунку. Серед переваг можна виділити можливість перегляду розташування улюбленців на карті та швидко зв’язатись з власником за допомогою електронної пошти. Серед мінусів є наявність великої кількості реклами, відсутня можливість пошуку за критеріями або місцем розташування, не зручний користувацький інтерфейс

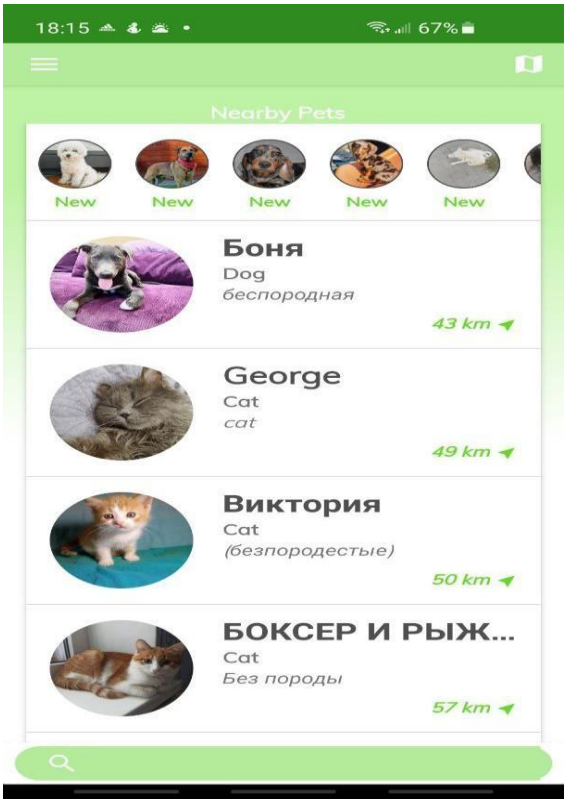

Рисунок 1.1 – Головний екран застосунку «Pet adopted»

Застосунок «Missing pets». Даний застосунок дає змогу створювати оголошення про загублених та знайдених тварин, головна сторінка зображена на рисунку 1.2. Реєстрація в системі працює через пошту користувача, наявність реєстрації в системі за допомогою соціальних мереж – відсутня. Серед переваг слід виділити зручність користування, а саме створення оголошення про зникнення домашньої тварини, або навпаки, її знайдення. До кожного оголошення користувач повинен обрати місце на карті де він загубив або знайшов тварину. Тому для кожного з двох типів оголошень існують координати на карті. Застосунок має можливість редагування особистих даних користувача. Серед недоліків слід зазначити відсутність реєстрації та авторизації за допомогою соціальних мереж, можливість пошуку оголошення за назвою та іншими критеріями (ім’я тварини, тип тварини, місце розташування і тд), відсутність пошуку через карту.

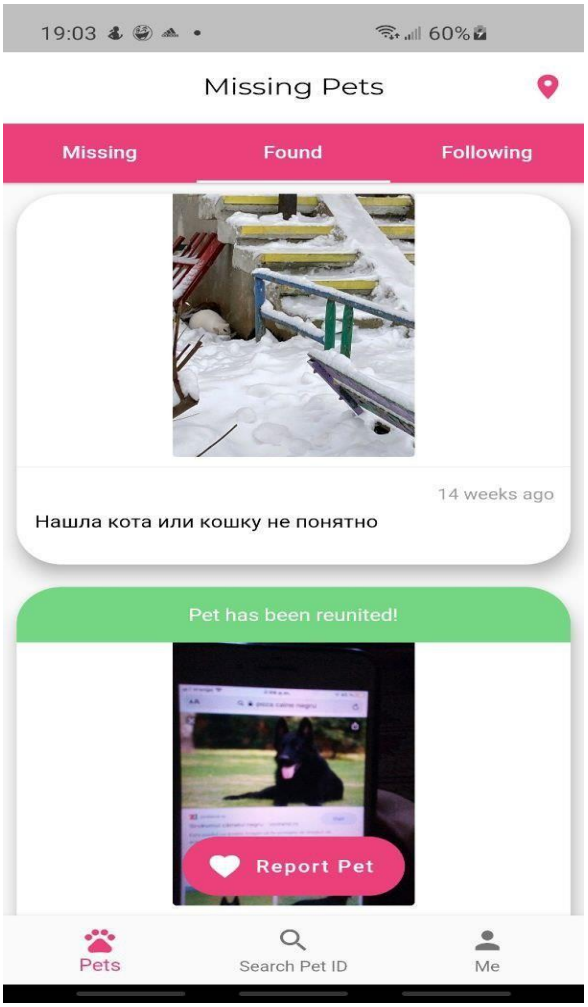

Рисунок 1.2 – Головна екран застосунку «Missing pets»

1.5 Висновки до розділу

В даному розділі було проаналізовано поняття «мобільного застосунку», їх різноманітні категорії та історію розвитку.

Визначено основні відмінності між трьома типами мобільних застосунків, а саме: веб-застосунки на основі браузера, нативні та гібридні застосунки. Спираючись на аналіз, можна зробити висновок, що гібридні мобільні застосунки є найбільш ефективними для розробки та впровадження, оскільки мають можливість використовувати сучасні веб-фреймворки для розробки та використовують інструменти для компіляції вихідного коду в програмний код певної операційної системи.

Також у цьому розділі було проведено аналіз існуючих застосунків для пошуку домашніх тварин. Проаналізовано основний функціонал, взято до уваги користувацький інтерфейс кожного застосунку та зручність користування ним.

РОЗДІЛ 2. ТЕХНІЧНІ ТА ПРОЕКТНІ РІШЕННЯ. АРХІТЕКТУРА ЗАСТОСУНКУ

2.2 Гібридні мобільні застосунки

Розробка додатків зазнала багато змін після виходу перших смартфонів. Цікавою річчю, про яку думає чи навіть пам’ятає не так багато людей, є те, що перший iPhone, який був анонсований 9 січня 2007 року, навіть не містив App Store. Тоді застосунків ще не було, єдиним, що могло б бути схоже на них, був веб-сайт із закладками. Лише через чотирнадцять місяців, у березні 2008 року, Apple представила свій SDK для програм та iTunes App Store, який мав величезний успіх. На мою думку, Apple проводило паралель між веб-застосунками та власними програмами. Apple вважала, що нативні застосунки забезпечать найкращий досвід для кінцевих користувачів. З часом були представлені різноманітні платформи для смартфонів, наприклад: Android, Blackberry, Windows. На даний момент нативна розробка стала складною та дорогою для розробки власних програм. Раніше компанії повинні були мати одну команду для iOS, одну для Android та одну команду для кожної іншої платформи, для якої вони хотіли б публікувати програми. У порівнянні з веб-розробкою це була величезна різниця. У веб-розробці не потрібно, наприклад, одна команда для Internet Explorer, одна для Chrome, одна для Firefox, просто є один код, яким можна керувати з будь-якого браузера. На ранніх етапах, коли мобільні веб-застосунки оживали і ставали дедалі поширенішими, це були в основному веб-сайти, призначені для роботи на смартфонах. Дизайн та функціональність були розроблені для роботи на смартфонах, але не створених спеціально для них. Це був новий та дешевий спосіб розробки застосунків для смартфонів. Недоліком цих програм було те, що продуктивність не була особливо гарною. Ще одним недоліком такого виду мобільних застосунків було те, що вони потребували мережевого з’єднання.

У 2009 році на заході iPhone DevCamp у Сан-Франциско була створена компанія Apache Cordova, яка виграла премію People’s Choice Award на конференції O’Reilly Media 2009 Web 2.0. Спочатку Apache Cordova була

створена компанією Nitobi, але в 2011 році була придбана компанією Adobe Systems і ребрендирована під PhoneGap. Сьогодні PhoneGap є однією з найпопулярніших гібридних платформ. Гібридна платформа – це, в основному, платформа, яка дозволяє писати один код, який може працювати на всіх платформах. Мова відрізняється залежно від того, яку платформу обрати, але більшість гібридних платформ використовують JavaScript, HTML5 та CSS3. Код пишеться як код веб-сайту, потім він відправляється у гібридну платформу, і як результат, після компіляції через гібридну платформу з'являється застосунок під певну операційну систему (iOS, Android, Windows тощо). Однією з великих відмінностей, завдяки якій гібридні застосунки стали настільки популярними порівняно з мобільними веб-застосунка, була можливість роботи в автономному режимі, оскільки вони вже не були «веб-сайтом», а тепер стали фактичним застосунком.

Гібридна мобільна розробка може передбачати або розробку оригінальної програми на нативній платформі (якою може бути iOS, Android, Windows Mobile, BlackBerry тощо), або розробку оригінальної програми в окремому середовищі, що надалі дозволить завантажувати програму на різні платформи. Гібридні євангелісти твердо підтримують свій метод, оскільки він надає можливість моделювати програми у абстрагованому вигляді та забезпечує кращий досвід роботи на багатьох пристроях. Існують різні технології та два підходи, засновані на розробці в одному середовищі та розгортанні на багатьох платформах (крос-платформа). Розробка з використанням цих підходів може дати гарний результат у часі та мінімізувати витрат, оскільки дозволяє розробникам писати лише однією з мов та використовувати єдиний фреймворк, який можна було б інтерпретувати на багато платформ.

Веб підхід базується на веб-браузерах для мобільних пристроїв. Додатки, засновані на веб підході, реалізовані з використанням HTML, CSS та JavaScript і покладаються на браузер, як середовище роботи. В рамках цього підходу застосунок реалізовано як єдиний оптимізований веб-сайт для мобільних пристроїв. Ця оптимізація повинна враховувати різні розміри екранів пристроїв

та принцип їх використання. Перевага веб-мобільних застосунків полягає в тому, що вони однаково існують у мобільних веб-браузерах на всіх платформах. Таким чином, оновлення мобільних додатків не потрібно. Недоліком веб-підходу є той факт, що доступ до власних функціональних можливостей пристрою (таких як система сповіщень, GPS, список контактів тощо) обмежений. Другий недолік полягає в тому, що час, необхідний для відтворення веб-сторінок шляхом завантаження їх із мережі, довший, ніж у власного мобільного інтерфейсу користувача. Більше того, веб-застосунки доступні лише за URL адресою і не можуть бути легко доступними в магазинах мобільних додатків.

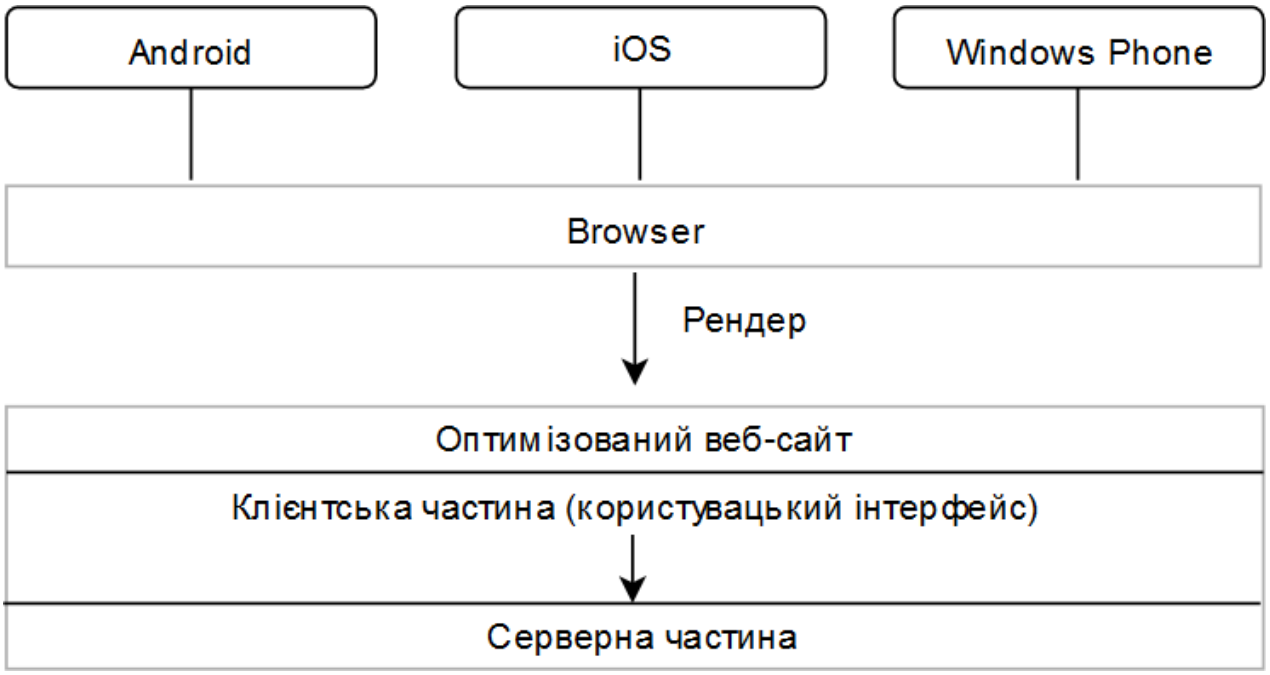

Рисунок 2.1 – Принцип роботи веб підходу

Гібридний підхід. Цей підхід використовує механізм браузера на пристрої та вбудовує вміст HTML у власний веб-контейнер (WebView в Android, UI WebView в iOS). Власні функціональні можливості доступні за допомогою фреймворків які працюють з нативними платформами. На відміну від веб-застосунків, гібридні розповсюджуються через сховища програм. Сучасна розробка гібридних застосунків перевищує використання WebView або UI WebView, останні платформи, такі як Flutter, використовують мову dart, мають

власний компілятор, який компілює програмний код у двійковий еквівалент нативної системи.

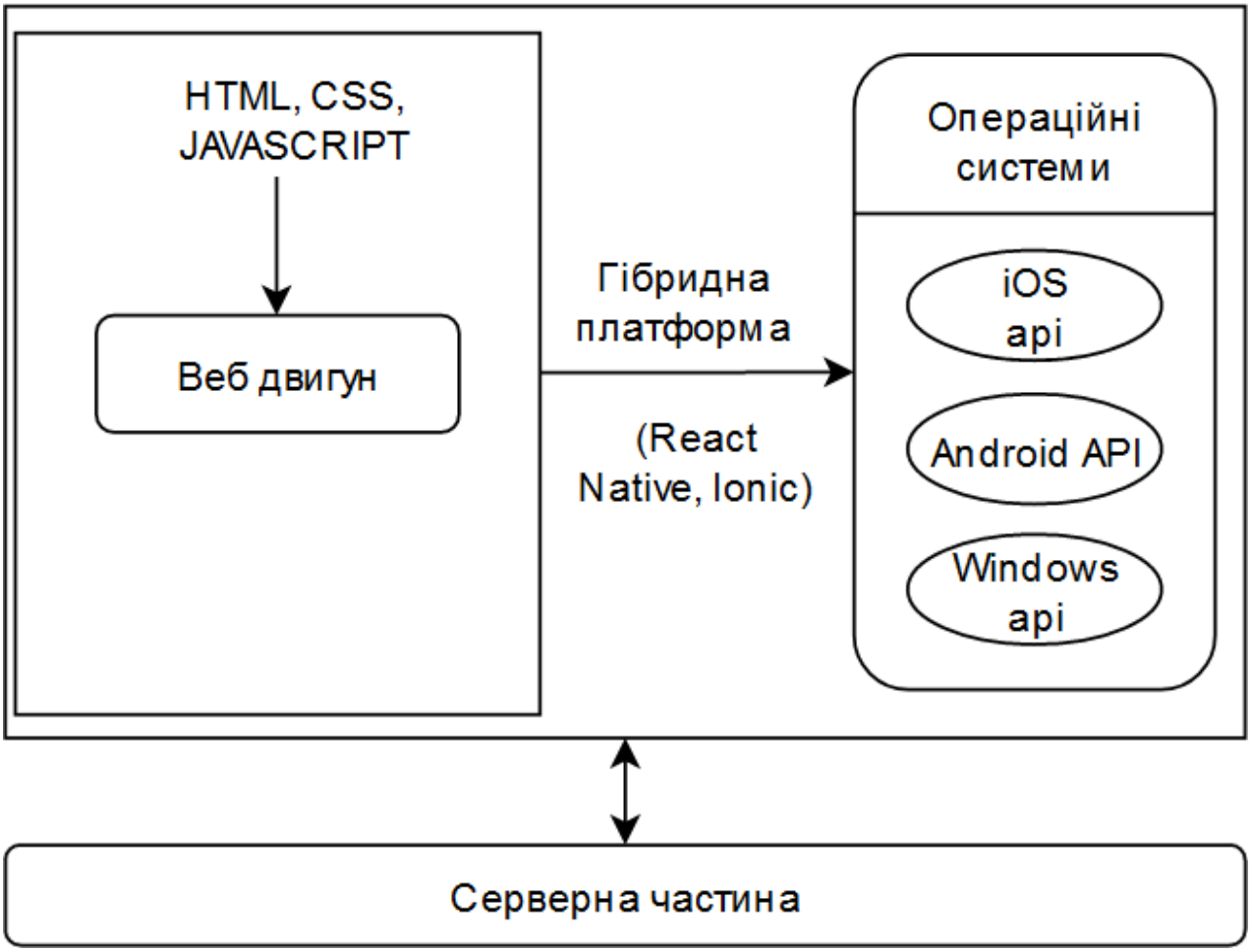

Рисунок 2.2 – Принцип гібридного підходу

Обмеження у розробці гібридних додатків. Хоча економія коштів може бути однією з переваг крос-платформ, типовий процес розробки мобільних застосунків має накладні витрати, пов'язані з визначенням вимог, аналізом та дизайном. Окрім інших подібних проблем, він також має специфічні для платформи міркування щодо дизайну, такі як форм-фактор, можливості операційних систем та обладнання.

Що стосується єдиної кодової бази, якщо виявлено та виправлено певну проблему або додано нову функцію на одній платформі, слід повністю перевірити всі інші платформи на принцип коректної роботи. Те саме застосовується, навіть якщо зміни потрібні лише для однієї платформи. Той факт,

що код використовується для всіх платформ, вводить обов’язкові накладні витрати – тестування на кожній платформі, кожного разу, коли зміна готова до подання. Будь-які зміни для певної платформи можуть мати непередбачені наслідки для не інших платформ.

Обмежений користувальницький інтерфейс. Гібридні програми мають дизайн, який не може використовувати всі можливості платформи. Таким чином, інтерфейс користувача не такий бездоганний. Можливості (наприклад, 3D) також обмежені через те, що використовується веб перегляд і це не дозволяє використовувати весь потенціал пристроїв.

Багато крос-платформних рішень дозволяють веб розробникам створювати застосунки за допомогою власних навичок, але проблеми, що виникають, вимагають спеціальних знань базової платформи або мови програмування.

Ймовірно, буде відбуватися подальша фрагментація мобільних пристроїв та технологій, що відіграватиме величезну роль у нарощуванні витрат та часових рамок. У той же час вони додадуть складності процесу розвитку. Буде більше питань, пов’язаних із безпекою, інтеграцією та модернізацією. Також можуть бути нові канали розповсюдження, де розробники можуть продавати свої програми безпосередньо споживачам, замість того, щоб проходити через магазини додатків. Соціальні медіа та їх потужність будуть і надалі зростати, і їх ефект буде відчутним у майбутньому мобільному просторі. Тому організації повинні застосовувати гнучкий підхід, для якого підтримка, масштабованість та інтеграція стають чинниками, які слід враховувати. Коли це станеться, рішення про правильну платформу чи підхід стануть на свої місця.

2.2 Платформа Ionic

Ionic не є новою технологією на ринку розробки мобільних застосунків. Фреймворк був створений в 2013 році як Sdk з відкритим кодом для гібридних мобільних застосунків. На даний час, в світі існує понад 5 мільйонів застосунків, які зроблені за допомогою платформи Ionic. Найбільшу популярність фреймворку надає можливість використовувати власні елементи

користувацького інтерфейсу для iOS, Android та підтримка різноманітних інструментів веб розробки.

Перші версії Ionic були засновані на Angular, популярному фреймворку для створення динамічних веб-сторінок та прогресивних веб-додатків, скорочено PWA. Ionic може використовувати Angular CLI (інтерфейс командної строки) та компоненти для створення повнофункціональних мобільних застосунків.

Друга версія Ionic, що відповідає за доступ до вбудованого функціоналу, заснована на плагінах Apache Cordova. Cordova – це інструмент для створення мобільних додатків з використанням веб-технологій, заснований на власних інтерфейсах API, а не на інтерфейсі API для конкретних платформ. Поки Ionic використовує WebView, за умовою у нього немає доступу до апаратного пристрою API. Cordova пропонує ці API, упаковані у вигляді плагінів, для доступу до таких функцій, наприклад, камера смартфону.

З моменту свого створення Ionic мав залежність від компонентів фреймворку Angular. Зміни доторкнулись четвертої та останньої версій Ionic, які мають багато нових функцій. З четвертою версією інструмент став незалежним від фреймворку, що означає незалежність від Angular. В майбутньому планується додати підтримку веб-фреймворків React і Vue. Це зміна стала можливою завдяки використанню веб-компонентів та просування інтерфейсу командної строки Ionic. Інтерфейс командної строки був перероблений та адаптований для роботи з Angular CLI. Таким чином, існує можливість працювати з Angular, якщо це не потрібно, а також використовувати інші підтримувані фреймворки, щоб розширити стек технології, який можна використовувати з Ionic. Ще одна велика зміна – це перехід на веб-компоненти. Веб-компоненти – це набори функцій, які використовують стандартні інтерфейси API, що підтримуються практично у всіх браузерах. Таким чином, їх можна розгорнути на будь-якій мобільній платформі або використовувати для створення настільних застосунків за допомогою платформ Electron або PWA. Використання веб-компонентів також дозволяє використовувати будь-які фреймворки з Ionic. Веб компоненти в основному представляють собою інкапсульовані елементи HTML, які взаємодіють один з

одним. Кожен з елементів, за замовчуванням містить підтримку дизайну iOS та Android, яку можна налаштовувати. Це спростовує завдання дизайну у багатьох додатках. За даними AppBrain, Ionic пропонує ряд зручностей для розробки мобільних додатків, що охоплюють більше 3,2% від усього ринку мобільних додатків.

Єдина кодова база для різних платформ**.** Ionic був побудований на основі Angular та Apache Cordova, а також із використанням HTML 5, CSS та JavaScript у якості основних технологій для розробки застосунків. Хоча остання версія Ionic вперше пропонує незалежне використання власних компонентів, все рівно існує можливість використовувати Angular з усіма його плюсами та мінусами. Тим більше, кожен, хто знає веб-технології та Angular, може застосовувати свої навички, використовуючи веб інструменти для створення повнофункціональних застосунків. Формування єдиної кодової бази коду для всіх платформ гарантує:

1. Зниження витрат на розробку застосунків та витрат на підтримку кодової бази.
2. Більш швидкий вхід застосунків в експлуатацію для обох платформ.
3. Простота обслуговування за допомогою вбудованих інструментів браузера та інструментів налагодження.

Економічна цінність використання Ionic очевидна, якщо зосередитись на швидкому впроваджені застосунків у обох магазинах мобільних застосунків. Розробка на пристроях Android, iOS набагато дешевше в рамках єдиної кодової бази за порівнянням із нативною розробкою.

Популярні технології та простота навчання. Ні для кого не секрет, що веб-технології отримали велику популярність, причому найпопулярнішою мовою програмуванням є JavaScript. Відповідно до опитування сайту StackOverflow 2020, веб-розробники є третьою за величиною групою серед усіх типових розробників. Використання Ionic у якості основного інструменту розробки мобільних застосунків гарантує, що не буде проблеми з пошуком розробників.

Ionic враховує простоту в освоєнні інструменту: розробники зовнішнього інтерфейсу можуть швидко освоїти основи або вибрати між різними веб-

фреймворками, які підтримують Ionic. Звісно, наявність досвіду у власній розробці стане лише плюсом, оскільки Ionic не компілює весь додаток на рідній мові програмування. Замість цього він компілює елементи користувацького інтерфейсу, використовуючи плагіни Cordova або Capacitor. За допомогою всього лише стека веб-технології, можна легко створювати та підтримувати застосунок.

Широкий спектр можливих інтеграцій та плагінів. Якщо ви вважаєте, що у вашого застосунку не достатньо функціональних можливостей, ви завжди можете інтегрувати його за допомогою багатьох інструментів. Офіційний список технологій для інтеграції можна знайти на веб-сайті Ionic, що забезпечує легкий доступ до аналітичних інструментів, платних систем, засобів безпеки та тестування. Він також містить ряд плагінів, які допомагають інтегруватися з обладнанням пристроїв. Але відомо, що деякі плагіни доступні як частина корпоративної версії Ionic, яка вимагає оплати за використання плагінів та інструментів Premier. Для отримання додаткових плагінів також можна переглянути список плагінів Cordova.

Обширний вибір елементів користувацького інтерфейсу та швидке прототипування. Більш старі версії Ionic вже довели свою ефективність у імітації зовнішнього вигляду власними застосунками завдяки своїй бібліотеці компонентів користувацького інтерфейсу. Ці компоненти можна використовувати як готові елементи для створення графічного інтерфейсу користувача (GUI) або використовувати ці елементи для налаштування. У співпраці з веб-компонентами, Ionic може прискорити процес розробки логіки користувацького інтерфейсу та збереження естетичного виду без додаткових затрат.

Компоненти користувацького інтерфейсу Ionic складаються з двох частин, які можна розбити на фактично графічні елементи графічного інтерфейсу та його функціонал. Існує можливість отримавши доступ до коду компонента користувацького інтерфейсу, змінювати логіку роботи елементів. Наприклад,

можна додати анімацію до кнопки, змінити тип прокрутки, відновити порядок елементів і так далі.

Ще один аспект, підвищення швидкості розробки Ionic – це можливість створення прототипів. Використання готових елементів користувацького інтерфейсу допомагає створювати прототипи майбутніх додатків у порівняльно коротких строках. Для цього можна використовувати інструмент для створення прототипів під назвою Ionic Creator. Він підтримується командою Ionic, пропонуючи інтерфейс перетаскування для створення інтерактивних прототипів.

Зручність тестування. Поки додатки Ionic працюють тільки через веб-перегляд, браузер пристрою можна використовувати для тестування програми. Це набагато зручніше, тому що не потрібно використовувати тестовий пристрій, щоб переконатися, що все працює без певних проблем. Та ж концепція може бути застосована до безлічі мобільних пристроїв в сучасному світі. Браузери пропонують вбудовані інструменти для тестування і налагодження, які роблять весь процес тестування досить таки зручним. Для тестування компонентів Angular, використовуваних в старіших версіях, можна використовувати Angular CLI, в той час як Ionic CLI підходить для тестування веб-компонентів. Таким чином, тестовий пристрій або емулятор може знадобитися тільки для тестування деяких вбудованих функцій.

2.3 Фреймворк Angular

Angular – це платформа з відкритим вихідним кодом на основі мови програмування Typescript, для створення клієнтських веб-застосунків. Для більш детального розуміння принципів роботи з Angular, слід знати основні поняття веб-розробки та мови JavaScript. Дана мова програмування працює на стороні клієнта в Інтернеті, який можна використовувати для розробки або програмування поведінки веб-сторінок при виникненні певних подій. Зазвичай Js використовується для взаємодії з інтерфейсами, та надають певної інтерактивності веб-сторінкам. JavaScript швидко розвивається і використовується для програмування на стороні сервера, наприклад в Node.js, розробці ігор та інше. Має справу з динамічним контентом, що є важливим аспектом веб-розробки. Під динамічним контентом розуміється постійно мінливий контент, який адаптується до конкретних користувачів. Наприклад, за допомогою Js можна визначити, чи потрібно відображати мобільну версію сайту чи ні, перевіривши до цього пристрій, з якого здійснюється запит доступу до сайту. Це спонукало веб-розробників почати створювати свої власні користувацькі бібліотеки для зменшення кількості рядків коду і можливості просто реалізовувати складні функції.

jQuery – це швидка, невелика і багатофункціональна бібліотека Js, яка значно спрощує такі речі, як маніпуляція з елементами HTML документами, обробка подій, анімація та http запити за допомогою простого у використанні API. jQuery став найпопулярнішим, тому що він був простим у використанні і надзвичайно потужним. Оскільки jQuery не має реальної структури, розробник має повну свободу створювати проекти на свій розсуд. Однак відсутність структури також означає, що легше потрапити в пастку «спагетті-коду», що може призвести до плутанини в більших проектах без чіткого напрямку дизайну або підтримки коду. У таких ситуаціях фреймворк на кшталт Angular може виявитися великою допомогою. В таблиці 2.1 описано порівняння бібліотек JQuery та Angular.

Таблиця 2.1 – Порівняння бібліотек JQuery та Angular

	JQuery	Angular
Маніпуляція з DOM	Присутня	Присутня
RESTful API	Відсутня	Присутня
Підтримка анімації	Присутня	Присутня
Валідація форм	Відсутня	Присутня
Двостороння прив’язка даних	Відсутня	Присутня
AJAX запити	Присутня	Присутня

Фреймворк Angular був створений спеціально, щоб допомогти розробникам створювати SPA (односторінкові застосунки) відповідно до кращих практик веб-розробки. За рахунок створення структурованого середовища для створення SPA значно знижується ризик створення «спагетті-коду». Односторінкові додатки (або SPA) – це додатки, які доступні через веб-браузер, як і інші сайти, але пропонують більш динамічну взаємодію, що нагадує мобільні та настільні додатки. Найпомітніша різниця між звичайним веб-сайтом і SPA – це менша кількість оновлень сторінок. У SPA частіше використовується AJAX – спосіб зв'язку з внутрішніми серверами без повного оновлення сторінки для завантаження даних в застосунок. В результаті процес рендеринга сторінок відбувається в основному на стороні клієнта. Діаграма роботи односторінкових застосунків зображено на рисунку 2.3.

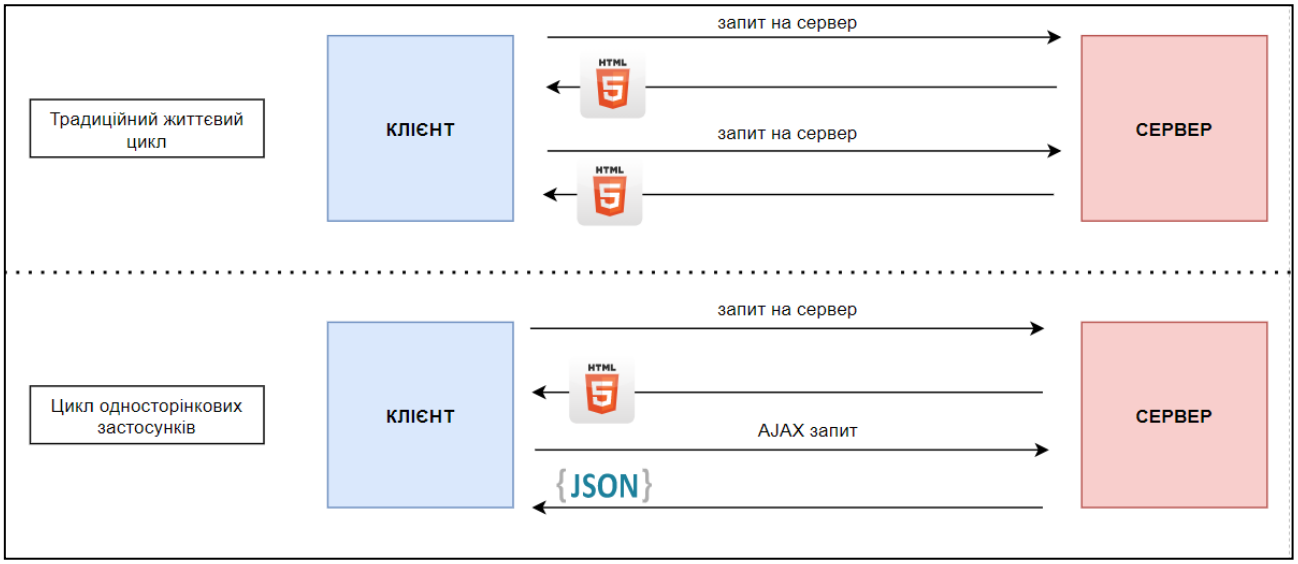

Рисунок 2.3 – Діаграма роботи SPA

Слід розрізняти поняття Angular та AngularJS. Angular – це заснована на мові TypeScript платформа інтерфейсних веб-застосунків з відкритим вихідним кодом, очолювана командою Angular в Google і співтовариством приватних осіб і корпорацій. Це повністю переписаний продукт тієї ж команди, яка створювала AngularJS. Але Angular суттєво відрізняється від AngularJS, а саме можна виділити такі ключові відмінності:

1. Архітектура застосунків Angular відрізняється від AngularJS. Основними будівельними блоками Angular є модулі, компоненти, метадані, прив'язка даних, директиви, служби та впровадження залежностей.

2. Angular не має концепції «області дії» або контролерів, замість цього він використовує ієрархію компонентів в якості своєї основної архітектурної концепції.

3. Angular має більш простий синтаксис виразів з акцентом на «[]» для прив'язки властивостей і «()» для прив'язки подій.

4. Мобільна розробка. Настільна розробка набагато простіше, якщо в першу чергу вирішувати проблеми з продуктивністю мобільних пристроїв. Таким чином, Angular в першу чергу займається мобільного розробкою.

5. Підтримка реактивного програмування з використанням бібліотеки RxJS.

Фреймворк Angular складається з таких структурних блоків: модулі, складові частини, шаблони, метадані, зв’язування даних, директиви, сервіси, впровадження залежностей.

Модулі. Застосунки Angular є модульними, кожний застосунок Angular містить як мінімум один модуль, тобто кореневої модуль. Зазвичай він називається AppModule. Кореневий модуль може бути єдиним модулем в невеликому застосунку. Хоча в більшості застосунків є кілька модулів. Можна сказати, що модуль – це згуртований блок коду зі зв'язаним набором можливостей. Будь який модуль – це клас з декоратором @NgModule. Декоратори – це функції, які змінюють класи JavaScript. Декоратори в основному використовуються для приєднання метаданих до класів, щоб розробник знав конфігурацію цих класів і те, як вони повинні працювати. NgModule – це функція-декоратор, яка приймає об'єкти метаданих, властивості яких описують модуль.

Складові частини. Компонент керує одною або більше секцією на екрані. Наприклад, якщо створювати застосунок зі списком курсів, то можуть бути створені такі компоненти, як компонент файлу завантаження, компонент курсу, компонент відомостей про курс і так далі. Усередині компонента визначається його логіка. Кожний застосунок має основний компонент, який завантажується всередині основного модуля, тобто AppComponent.

Метадані кажуть Angular, як обробляти клас. Щоб повідомити Angular, що компонент є саме компонентом, до класу прикріплюються метадані. У TypeScript метадані приєднуються за допомогою декораторів.

Зв'язування даних. Якщо не використовувати фреймворк, то потрібно помістити значення даних в елементи управління HTML і перетворити відповіді користувачів до деяких дії і оновити значення. Написання такої логіки push / pull може створити багато труднощів, значною кількістю помилок і жахом для читання. Angular підтримує прив'язку даних, механізм для узгодження частин

шаблону з частинами компонента. Потрібно додати розмітку прив'язки в HTML-код шаблону, щоб повідомити Angular, як з'єднати обидві сторони. Двостороння прив'язка даних є важливою частиною, оскільки вона об'єднує прив'язку властивостей і подій в єдиній нотації за допомогою директиви ngModel. При двосторонньої прив'язці значення властивості даних перетікає в поле вводу від компонента, як і при прив'язці властивостей. Зміни користувача також передаються назад в компонент, скидаючи властивість до останнього значення, як у випадку прив'язки події. Angular обробляє всі прив'язки даних один раз за цикл подій. Зв'язування даних відіграє важливу роль в зв'язку між шаблоном і його компонентом. Прив'язка даних також важлива для зв'язку між батьківськими і дочірніми компонентами.

Сервіси – це широка категорія, що охоплює будь – які значення, функції або можливості, які необхідні застосунку. Сервіс зазвичай являє собою клас з чітко визначеною метою. У Angular немає конкретного визначення сервісу. Проте, сервіси є фундаментальними для будь – якої програми Angular, можна сказати, що компоненти є споживачами сервісів. Сервіси є скрізь. Класи компонентів не отримують дані з сервера, не перевіряють введення даних користувачем. Такі завдання делегуються на сервіси. Завдання компонента – забезпечити зручність роботи користувача і не більше того. Він є посередником між шаблоном і логікою програми. Хороший компонент представляє властивості і методи для прив'язки даних. Angular дійсно допомагає слідувати цим принципам, спрощуючи поділ логіки застосунку на служби і роблячи ці служби доступними для компонентів за допомогою впровадження залежностей.

Впровадження залежностей – це спосіб надати новому примірнику класу повністю сформовані залежності, які йому потрібні. Більшість залежностей – це сервіси. Angular використовує впровадження залежностей, щоб надати новим компонентам необхідні їм сервіси. Angular може визначити, які сервіси потрібні компоненту, подивившись на типи параметрів його конструктора. Коли Angular створює компонент, він спочатку запитує у інжектор, які сервіси потрібні компоненту. Інжектор підтримує раніше створений контейнер примірників

сервісів. Якщо запитаний екземпляр сервісу відсутній в контейнері, інжектор створює його і додає в контейнер перед поверненням сервісу. Коли всі запити сервісів дозволені і повернуті, Angular може викликати конструктор компонента з цими службами в якості аргументів. Це ін'єкція залежності. Приклад введення залежностей зображено на рисунку 2.4.

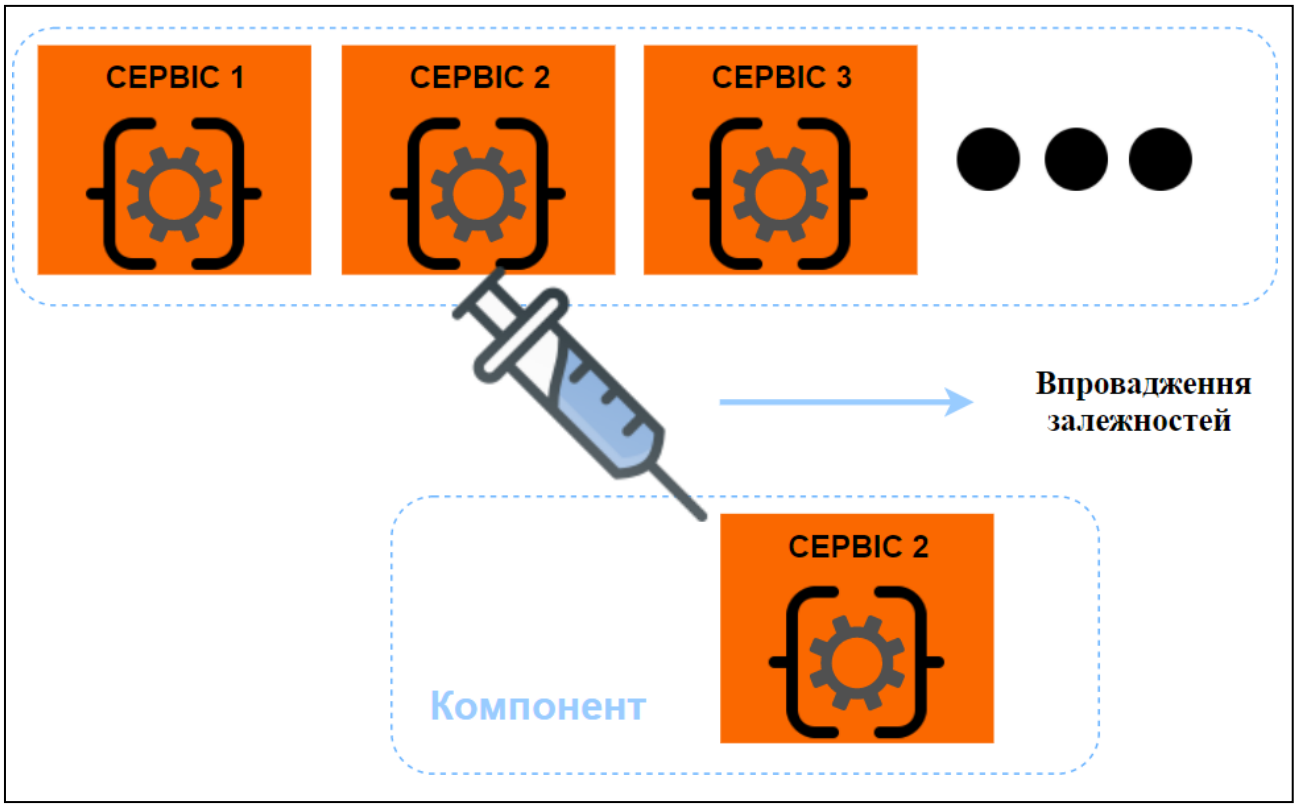

Рисунок 2.4 – Приклад впровадження залежностей

2.4 Серверна частина для мобільних застосунків. Фреймворк Nest

З 1999-го року, коли вперше було введено поняття "веб-застосунок", їх використання зросло до величезних масштабів, і багато користувачів Інтернету використовували їх раніше навіть не усвідомлюючи цього. Сьогодні мільйони людей щодня використовують різноманітні сервіси через Інтернет для виконання електронних банківських операцій, подання декларації про доходи в режимі онлайн, обміну фотографіями або відео в соціальних мережах, спілкування з іншими людьми в соціальних мережах, спілкуватися з іншими людьми в месенджерах і так далі. Наприклад, Google Docs – це веб-застосунок для роботи та управління текстом, який зберігає дані онлайн і дозволяє користувачам завантажувати файли на персональний комп'ютер. Що стосується розробки, то в сучасну епоху Node.js є популярним вибором для створення серверної частини. Node.js – це середовище виконання JavaScript, яке формує основу багатьох популярних сьогодні фреймворків. Вона відома своєю подіє–орієнтованої моделлю, яка надає високошвидкісну синхронізовану обробку даних. В результаті, фреймворки Node стали вибором багатьох компаній для створення серверів, що вимагають гнучку і високомасштабовану обробку даних. Однак, як показано на рисунку 2.5, екосистема NodeJS отримала великий розвиток і існує широкий спектр унікальних фреймворків. Кожен фреймворк пропонує різні технічні сценарії і вибір одного з них повинен бути ретельно вирішене на основі проектних вимог і цілей.

Рисунок 2.5 – Популярні фреймворки Node

При проектуванні і розробці як веб так і мобільних застосунків, найбільш фундаментальними етапами є розробка клієнтської та серверної частини. Розробка клієнтської частини, в основному, зосереджена на кінцевих користувачів. На відміну від цього, основний фокус розробки серверу – це те, як працює застосунок всередині. Серверна частина взаємодіє з клієнтською, доставляючи і отримуючи дані для подання їх на веб-сторінках або екранах телефонів. При будь якій активності користувача на сторінці, клієнт передає запит на сервер і отримує від нього відповідь. Отже, розробка серверу часто складається з таких складових частин:

1. Програмування бізнес-логіки за допомогою мов програмування Java, C#, JavaScript, TypeScript, Php, Python і так далі.
2. Архітектура серверу.
3. Адміністрування бази даних.
4. Масштабування.
5. Безпека.
6. Тестування.
7. Резервне копіювання.

Фреймворк Nest.js був розроблений Камілем Мишлівцем з метою створення ефективних, масштабованих серверних-застосунків Node. Фреймворк підтримує JavaScript і TypeScript, більш того, в ньому змішані компоненти функціонального, об’єктно орієнтованого та функціонально реактивного програмування.

Структура фреймворку. Наведені нижче концепції є основними для Nest.js:

1. NestCLI – це інструмент командного рядка, який не тільки пришвидшує розробку застосунків, але також автоматизує ініціалізацію серверу.
2. Контролери – керує запитами і повертає дані клієнта. Основним завданням контролера є отримувати конкретні запити для сервера з сторони клієнту. Механізм маршрутизації визначає, який контролер отримує ті чи інші запити. Іноді кожен контролер має більше одного маршруту, і кілька маршрутів можуть виконувати різні дії.

2.1 Для того, щоб створити базовий контролер, Nest використовує класи та декоратори. Декоратори зв’язують з необхідними метаданими і дозволяє фреймворку створити карту маршрутизації.

3. Провайдери – це фундаментальна концепція в Nest. Основна ідея провайдера полягає в тому, що він може впроваджувати залежності. Це означає, що об'єкти можуть будувати різні відносини один з одним. Якщо коротко, провайдер – це просто клас, анотований декоратор @Injectable().

4. Сервіси. Вони відповідають за роботу бізнес логіки і призначені для керування контролерами, тому, він також може бути визначений як провайдерю

5. Модулі. Так само як і в фрейморку Angular, Nest використовує модулі для групування певних частин застосунку.

6. Ін’єкція залежностей – модель програмування, в якій замість створення залежності, клас вимагає залежностей з зовнішніх джерел. При такому підході, клас отримує необхідний сервіс або провайдер в конструкторі класу.

В загальному вигляді архітектура Nest має чотири основні переваги серед конкурентів.

По перше, NestJS – це фреймворк, який викликає безліч думок, в якому існують вже прокладені принципи проектування та вирішення проблем і розробникам необхідно слідувати наданими рекомендаціями по структурі, не роблячи ніяких припущень.

По друге, застосування NestJS написані на мові TypeScript. За допомогою не типізованих мов програмування, таких як JavaScript або PHP, код може бути написаний без особливих зусиль, тому ризик появи помилок дуже великий без використання суворої типізації. З іншого боку, Nest, ймовірно, є єдиною доступною та добре структурованою системою, яка повністю написана на TypeScript – це типізована версія JavaScript, яка надає підтримку типів під час компіляції і під час виконання. Для великих додатків ця надійність особливо корисна. TypeScript також пропонує користувачам доступ до нових можливостей JavaScript, які ще не були представлені.

По третє, NestJS має міцну Angular подібну архітектуру. В даному інструменті структура папок в значній мірі заснована на Angular. Отже, при наявності знань Angular, це забезпечує мінімальний час для створення застосунку. Кожен компонент отримує свою папку, модулі та основні файли, розташовані в корені (плюс деякі додаткові файли конфігурації). Цей фреймворк так само простий, як і звучить, і змушує розробників приділяти більше уваги дизайну кінцевих точок замість загальної структури програми.

Підсумовуючи, можна виділити наступні переваги та недоліки використання фреймворку Nest:

1. Мова програмування TypeScript.
2. Наявність Angular подібної архітектури.
3. Інтерфейс командного рядка, який дозволяє полегшити ініціалізацію нових компонентів.
4. Висока продуктивність.
5. Масштабованість.

Недоліки:

1. Високий рівень входу для нових розробників.

2.1 Висновки до розділу

В даному розділі були детально проаналізована архітектура гібридних мобільних застосунків та популярні інструменти розробки серверної частини.

Підсумовуючи, можна сказати, що для розробки гібридних мобільних застосунків можна використовувати більш звичні веб-інструменти (html, css, javascript), які в свою чергу проходять етап компіляції та перетворюються на компоненти певної мобільної операційної системи. Данна можливість дає велику перевагу таким інструментам як Ionic в розробці мобільних застосунків, оскільки рівень входу є нижчим, в порівнянні з розробкою нативних застосунків.

Основним фреймворком на платформі Ionic був обраний веб-фреймворк Angular, який має модульну структуру, підтримку мови програмування TypeScript та має власну реалізацію контейнера залежностей.

Основною платформою для розробки серверної частини, було обрана популярна платформа для JavaScript розробників – Node.js. На даній платформі існує фреймворк для серверних частин, який швидко розвивається – Nest. За основу він взяв ядро популярного фреймворку Express але додав підтримку мови TypeScript та багатьох нових функцій. Це дало підставу обрати Nest для розробки серверу.

РОЗДІЛ 3. ПРОЕКТУВАННЯ ТА РОЗРОБЛЕННЯ МОБІЛЬНОГО ЗАСТОСУНКУ

3.1 Розробка мобільного застосунку та серверної частини

При проектуванні користувацького інтерфейсу, було взято за основу популярні практики взаємодії сторінок мобільного застосунку. На рисунку 3.1 зображено взаємодію компонентів мобільного застосунку та потоків даних.

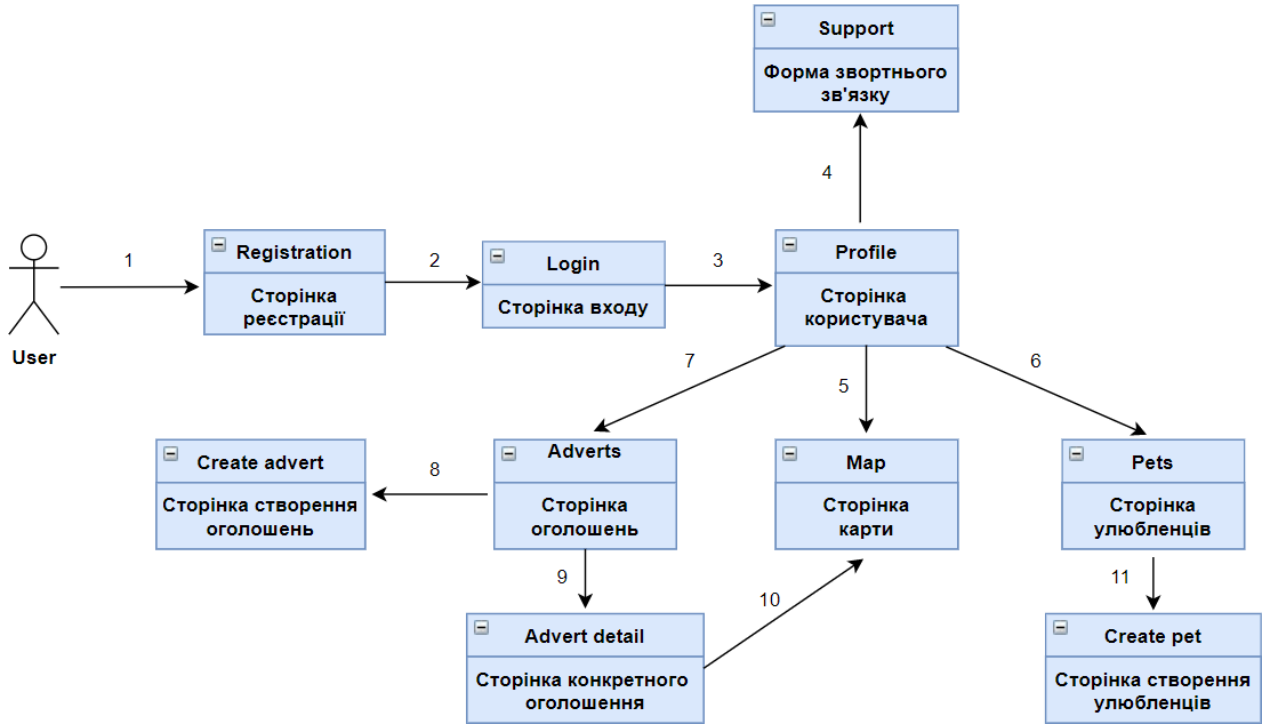

Рисунок 3.1 – Діаграма взаємодії компонентів застосунку

Потоки даних на рисунку 3.1 зображені цифрами від 1 до 11, які мають такі визначення:

1. Ввід даних, які потрібні для реєстрації користувача в системі.
2. Перехід на сторінку входу в систему, після успішної реєстрації.
3. Ввід особистих даних користувача, а саме пошти та паролю для авторизації в системі.
4. В разі виникнення проблем, користувач може зв’язатись з підтримкою за допомогою зворотньої форми.

5. Перехід на сторінку з картами, на якій відображаються оголошення про зникнення тварин.

6. Перехід на сторінку з власними улюбленцями, яких користувач може власноруч додати.

7. Перехід на сторінку оголошень, на якій відображається список з загубленими тваринами.

8. Створення власного оголошення про зникнення.

9. Перехід на сторінку конкретного оголошення.

10. Відображення місця розташування оголошення на карті

11. Додавання в систему власну тварину

Для зберігання даних мобільного застосунку, на стороні сервера використовується база даних MySQL. Оскільки мобільний застосунок має архітектуру «Клієнт – Сервер», запити обробляються контролерами, які в свою чергу викликають певні сервіси з бізнес-логікою. Слідуючи принципам чистої архітектури, рівень бізнес логіки має чітку межу з рівнем контролерів. Таким чином, всі запити до бази відбуваються, безпосередньо в сервісах. Для більш зручної роботи з базою та автоматизацією запитів до неї, використовується об’єктно реляційна модель для сутностей бази даних. За допомогою неї, можна суттєво полегшити роботу з таблицями та маніпуляції з ними. Зручний інтерфейс та можливість підтримки механізму введення залежностей, дозволяє легко використовувати дану об’єктну модель на сервері веб-застосунку. Детальна схема бази даних зображена на рисунку 3.2, опис конкретних таблиць показаний в таблиці 3.1.

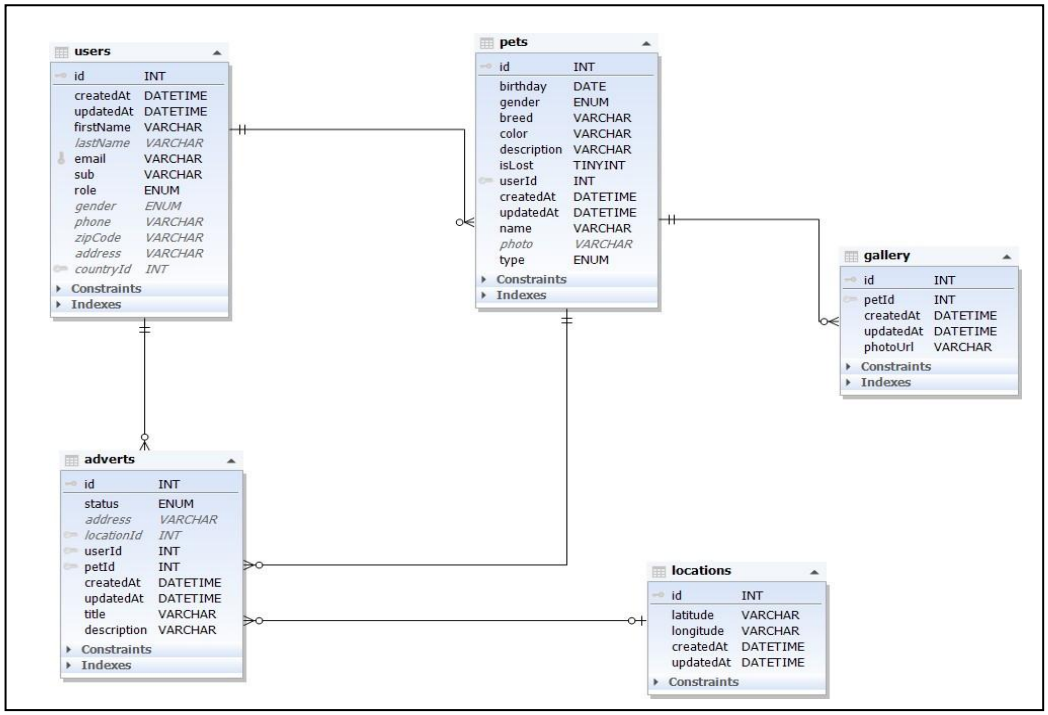

Рисунок 3.2 – Схема бази даних

Таблиця 3.1 – Опис сутностей бази даних

Назва таблиці	**Призначення**
Users (Користувачі)	Слугує для зберігання основних даних користувача застосунку.
Pets (Улюбленці)	Таблиця з даними улюбленців, яка в свою чергу має зв’язок з таблицею «користувачі», оскільки кожен користувач може мати багато улюбленців.
Gallery	Слугує для зберігання посилань на фотографії домашніх тварин. В свою чергу має зв’язок один до багатьох з таблицею «улюбленці».
Adverts (Оголошення)	Таблиця слугує для зберігання даних оголошень про домашніх тварин, які зникли
Locations (Дані місцезнаходження)	Таблиця, яка зберігає значення широти та довготи.

3.2 Інструкція користувача

Для можливості використання повного функціоналу застосунку, користувачу потрібно пройти реєстрацію, яка зображена на рисунку 3.3.

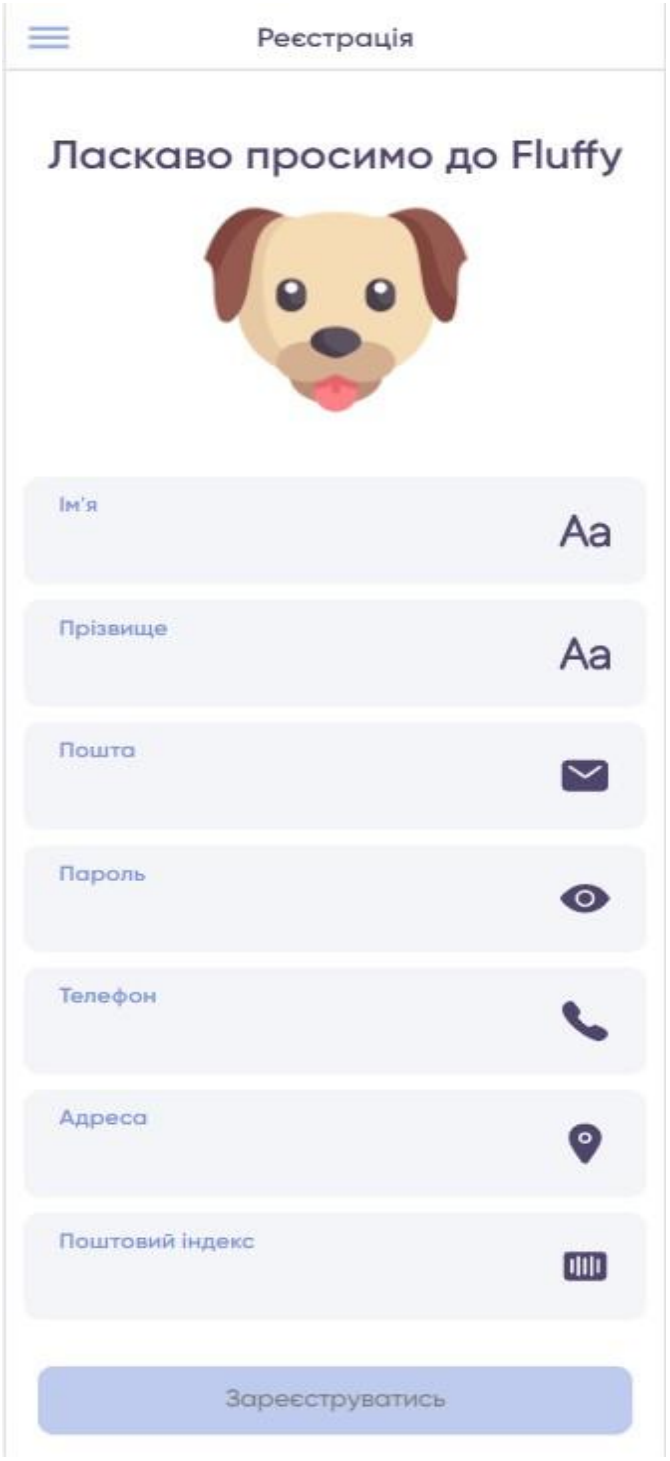

Рисунок 3.3 – Сторінка реєстрації

Для успішної реєстрації, користувач повинен записати всі необхідні дані, після чого він має можливість увійти в застосунок за допомогою пошти та паролю (рисунок 3.4).

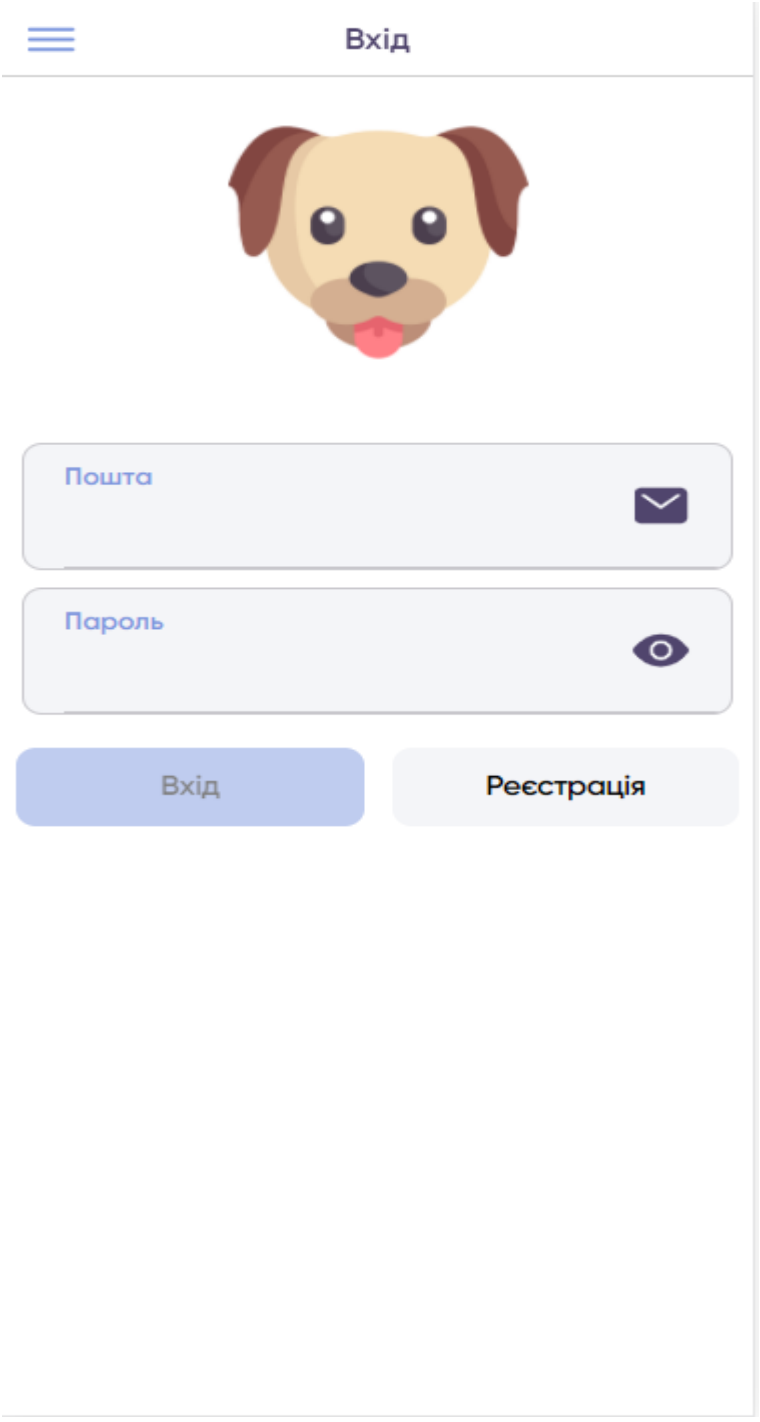

Рисунок 3.4 – Сторінка входу

Застосунок має нижню панель навігації з кнопками: оголошення, карта, улюбленці та профіль. Для перегляду вже існуючих оголошень, користувач може

натиснути по відповідній кнопці навігації та перейти на сторінку з списком оголошень (рисунок 3.5)

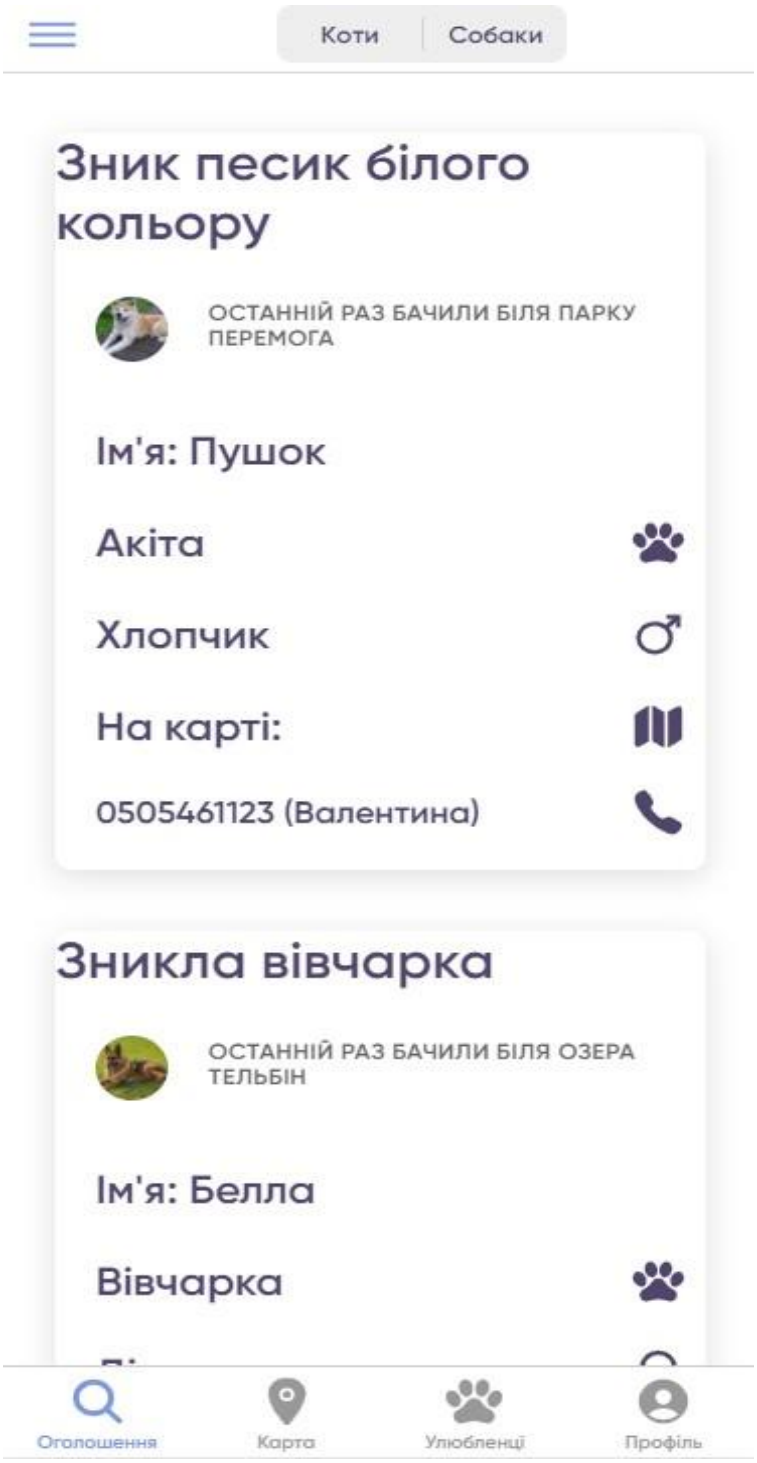

Рисунок 3.5 – Сторінка з оголошеннями

Користувацький інтерфейс дає змогу продивитись кожне оголошення на карті, для цього користувачу потрібно натиснути на зображення мапи на певному

оголошені, після чого користувача буде переведено на сторінку «Карта» в якому буде місце де зникла та чи інша тварина (рисунок 3.6)

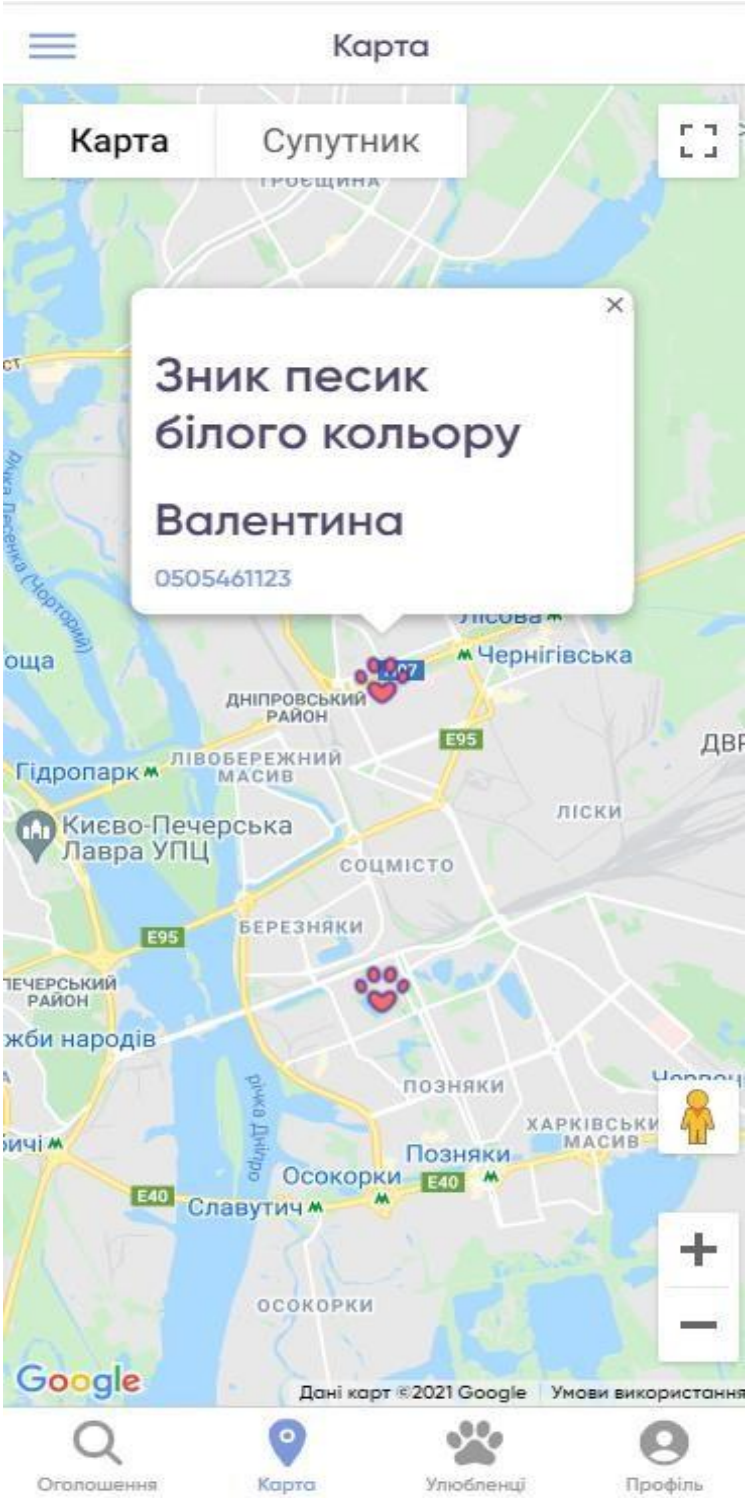

Рисунок 3.6 – Сторінка карти

ВИСНОВОК

Результатом виконання дипломної роботи є створення мобільного застосунку для пошуку домашніх тварин. Для цього було обрано мову програмування TypeScript та платформу для розробки Ionic. Дана платформа має переваги у вигляді: простоти розробки, великий вибір компонентів, можливість програмувати на різних фреймворках. Для розробки серверної частини, був обраний фреймворк Nest, який має обширний функціонал розробки серверу на основі платформи Node. Підібраний стек технологій дозволяє розробити повноцінний мобільний застосунок з урахуванням сучасних підходів та практик.

Під час виконання дипломної роботи:

1. Дослідив теоретичні основи побудови мобільного застосунку для пошуку домашніх тварин.
2. Провів аналіз існуючих рішень.
3. Проаналізував сучасні методи та рішення в розробці мобільних застосунків на базі операційної системи Android.
4. Спроектував мобільний застосунок для пошуку домашніх тварин.

Під час написання дипломної роботи, було проаналізовано основні методи та підходи в розробці мобільних застосунків, загальні концепції архітектури застосунків. Був про проведений аналіз об’єкту програмування згідно до технічного завдання. Також була надана інструкція користувача щодо користування мобільним застосунком для пошуку домашніх тварин.

Розроблений застосунок має переваги в зручності використання, простоті графічного інтерфейсу користувача та підтримки адаптивних розмірів екрану.

Оскільки архітектура застосунку та серверної частини відповідає принципам модульності та інверсії залежностей, мобільний застосунок має властивість до масштабування.

Розроблений мобільний застосунок відповідає основним вимогам технічного завдання, та може бути впроваджений у повсякдення використання користувачами Android застосунків.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Marjin Haverbeke. Eloquent JavaScript: A Modern Introduction to Programming – No Starch Press, 2019. С. 544.

2. David Flanagan. JavaScript: The Definitive Guide – O'Reilly, 2017. С. 722.

3. Carlos Taborda. The Complete Book on Angular 2 – Fullstack.io, 2016. С. 348.

4. Опис продукту Visual Studio Code [Електронний ресурс]. 2020. Режим доступу до ресурсу: https://code.visualstudio.com/docs/supporting/faq.

5. Девід Макфарланд. Нова велика книга CSS – Питер Пресс, 2020. С. 518.

6. Роберт Мартін. Чиста Архітектура – Фабула, 2020. С. 45-130.

7. Роберт Мартін. Чистий Код – Фабула, 2020. С. 50-75.

8. Ерік Фрімен. Патерни проектування – Фабула, 2020. С. 73-218

9. Ari Lerner. The Complete Book on Angular 2 (Volume 2) 1st Edition – FullStack.io, 2019. С. 152-456.

10. Criss Griffith. Mobile App Development with Ionic – O'Reilly, 2018. С. 295.

11. Jeremy Wilken. Angular in Action – Manning Publications. 2019. С. 318.

12. Amit Gharat. Reactive State for Angular with NgRx – BPB Publications. 2018. С. 210-311.

13. Ethan Brown. Web Development with Node and Express: Leveraging the JavaScript Stack – O'Reilly. 2019. С 112-265.

14. Офіційна документація Angular [Електронний ресурс]. 2020. Режим доступу до ресурсу: https://angular.io/docs.

15. Офіційна документація Ionic [Електронний ресурс]. 2020. Режим доступу до ресурсу: https://ionicframework.com/docs.

16. Nest official documentation [Електронний ресурс]. 2020. Режим доступу до ресурсу: https://docs.nestjs.com.

17. Запити до таблиць в MySQL [Електронний ресурс]. 2020. Режим доступу до ресурсу: https://metanit.com/sql/mysql.

18. Офіційна документація по Google Maps API для мови програмування JavaScript [Електронний ресурс]. 2020. Режим доступу до ресурсу: https://developers.google.com/maps/documentation/javascript/overview.

19. Офіційна документація по бібліотеці реактивного програмування RxJs [Електронний ресурс]. 2020. Режим доступу до ресурсу: https://rxjs.dev/guide/overview.

20. Офіційна документація по бібліотеці зберігання станів застосунку NgRx [Електронний ресурс]. 2020. Режим доступу до ресурсу: https://ngrx.io/docs.

21. Загальноприйняті принципи написання коду на JavaScript [Електронний ресурс]. 2020. Режим доступу до ресурсу: https://github.com/goldbergyoni/nodebestpractices.

22. Правила та принципи розробки REST API [Електронний ресурс]. 2020. Режим доступу до ресурсу: https://stackoverflow.blog/2020/03/02/best-practices-for-rest-api-design.

ДОДАТКИ

Додаток А

Приклад лістингу коду серверної частини застосунку

```
import {
  Body,
  Controller,
  Get,
  HttpStatus,
  Post,
  UseGuards,
} from '@nestjs/common';
import { LoginRequest } from './dto/login/login.request';
import { AuthService } from './services/auth.service';
import { ApiResponse, ApiTags } from '@nestjs/swagger';
import { RegisterRequest } from './dto/register/register.request';
import { LoginResponse } from './dto/login/login.response';
import { RegisterResponse } from './dto/register/register.response';
import { RefreshSessionRequest } from './dto/refresh/refreshSession.request';
import { RefreshSessionResponse } from './dto/refresh/refreshSession.response';
import {
  AuthUser,
  FriendlyHttpException,
  JwtAuthGuard,
  UserEntity,
} from '../../core';

@Controller('auth')
@ApiTags('auth')
export class AuthController {
  constructor(private readonly authService: AuthService) {}

  @Post('/login')
  @ApiResponse({
  status: HttpStatus.OK,
```

```
  description: 'Return tokens pair',
  type: LoginResponse,
})
async login(@Body() model: LoginRequest) {
  try {
    const { cognitoUserSession, user } = await this.authService.login(model);
    const accessToken = cognitoUserSession.getAccessToken().getJwtToken();
    const refreshToken = cognitoUserSession.getRefreshToken().getToken();
    const idToken = cognitoUserSession.getIdToken().getJwtToken();

    const response = new LoginResponse(
      user,
      accessToken,
      refreshToken,
      idToken,
    );

    return response;
  } catch (err) {
    const { message, friendlyStatus } = err;
    throw new FriendlyHttpException(
      friendlyStatus ? friendlyStatus : HttpStatus.BAD_REQUEST,
      message,
    );
  }
}

@Post('/register')
@ApiResponse({
status: HttpStatus.OK,
  description: 'Return new user',
  type: RegisterResponse,
})
async register(@Body() model: RegisterRequest) {
  const user = await this.authService.register(model);
```

```
    const response = new RegisterResponse(user);
    return response;
  }

  @Post('/refresh')
  @UseGuards(JwtAuthGuard)
  @ApiResponse({
    status: HttpStatus.OK,
    description: 'Return updated tokens',
    type: RefreshSessionResponse,
  })
  async refreshSession(@Body() model: RefreshSessionRequest) {
    const { idToken, refreshToken } = model;

    const result = await this.authService.refreshSession(idToken, refreshToken);

    const response = new RefreshSessionResponse(
      result.getAccessToken().getJwtToken(),
      result.getRefreshToken().getToken(),
      result.getIdToken().getJwtToken(),
    );

    return response;
  }

  @Get('/callback')
  @UseGuards(JwtAuthGuard)
  async callback(@AuthUser() user: UserEntity) {
    return user;
  }
}

import { HttpStatus, Injectable, UnauthorizedException } from '@nestjs/common';
import { InjectRepository } from '@nestjs/typeorm';
import {
```

```
  CountryEntity,
  IsEntityExist,
  Roles,
  UserEntity,
} from '../../../core/typeorm';
import { Repository } from 'typeorm';
import { LoginRequest } from '../dto/login/login.request';
import { FriendlyHttpException, ValidationException } from 'src/core';
import { RegisterRequest } from '../dto/register/register.request';
import { UserService } from 'src/modules/user/services/user.service';
import {
  AuthenticationDetails,
  CognitoIdToken,
  CognitoRefreshToken,
  CognitoUser,
  CognitoUserAttribute,
  CognitoUserPool,
  CognitoUserSession,
} from 'amazon-cognito-identity-js';

@Injectable()
export class AuthService {
  constructor(
  @InjectRepository(UserEntity)
    private readonly userRepo: Repository<UserEntity>,
    @InjectRepository(CountryEntity)
    private readonly countryRepo: Repository<CountryEntity>,
    private readonly userService: UserService,
    private cognitoPool: CognitoUserPool,
  ) {}

  async login(
    model: LoginRequest,
  ): Promise<{ user: UserEntity; cognitoUserSession: CognitoUserSession }> {
    const { email, password } = model;
```

```
  const user = await IsEntityExist<UserEntity>(this.userRepo, { email });
  if (!user) {
    throw new FriendlyHttpException(
      HttpStatus.NOT_FOUND,
      'User is not found',
      ['email'],
    );
  }
  const authDetails = new AuthenticationDetails({
    Username: email,
    Password: password,
  });

  const cognitoUser = new CognitoUser({
    Pool: this.cognitoPool,
    Username: email,
  });

  const result = new Promise<CognitoUserSession>((res, rej) => {
    cognitoUser.authenticateUser(authDetails, {
      onSuccess: result => {
        res(result);
      },
      onFailure: err => {
        rej(err);
      },
    });
  });

  const cognitoUserSession = await result;

  return { user, cognitoUserSession };
}
```

```
async register(model: RegisterRequest) {
  const { email, firstName, password, phone } = model;

  const country = await IsEntityExist<CountryEntity>(this.countryRepo, {
    id: model.countryId,
  });

  if (!country) {
    throw new ValidationException('Country is not exist', ['countryId']);
  }

  const existedUser = await IsEntityExist<UserEntity>(this.userRepo, {
    email: model.email,
  });

  if (existedUser) {
    throw new FriendlyHttpException(HttpStatus.OK, 'User is already exist');
  } else {
    const attributeList = [];
    attributeList.push(
      new CognitoUserAttribute({ Name: 'name', Value: firstName }),
    );
    attributeList.push(
      new CognitoUserAttribute({ Name: 'phone_number', Value: phone }),
    );
    attributeList.push(
      new CognitoUserAttribute({ Name: 'custom:role', Value: Roles.USER }),
    );

    const cognitoUserSub = new Promise((resolve, reject) => {
      this.cognitoPool.signUp(
        email,
        password,
        attributeList,
        null,
```

```
      (err, result) => {
        if (err) {
        reject(err);
        } else {
          resolve(result.userSub);
        }
      },
    );
  });

  const sub = (await cognitoUserSub) as string;

  const user = await this.userService.create({
    email,
    firstName,
    phone,
    country,
    sub,
  });

  return user;
 }
}

async refreshSession(idToken: string, refreshToken: string) {
 const IdToken = new CognitoIdToken({ IdToken: idToken });
 const RefreshToken = new CognitoRefreshToken({
 RefreshToken: refreshToken,
 });

 if (!IdToken.payload.email) {
  throw new UnauthorizedException('IdToken is not valid');
 }

 const user = await IsEntityExist<UserEntity>(this.userRepo, {
```

```
      email: IdToken.payload.email,
    });

    if (!user) {
      throw new FriendlyHttpException(
        HttpStatus.NOT_FOUND,
        'User is not found',
        ['email'],
      );
    }

    const cognitoUser = new CognitoUser({
      Pool: this.cognitoPool,
      Username: IdToken.payload.email,
    });

    return new Promise<CognitoUserSession>((res, rej) => {
      cognitoUser.refreshSession(RefreshToken, (err, session) => {
      if (err) {
          rej(err);
        }
        res(session);
      });
    });
  }
}
```