Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління
(повна назва)

Кафедра комп'ютерних інтелектуальних технологій та систем
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

Рівень вищої освіти перший (бакалаврський)

Розробка мобільного додатка для фіксації та повідомлення про проблеми інфраструктури міста

(тема)

Виконав:
здобувач IV року навчання,
групи КІУКІ-21-10
Лоліта ГАСАНОВА
(власне ім’я, прізвище)

Спеціальність
123 «Комп’ютерна інженерія»
(код і повна назва спеціальності)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма
Комп’ютерна інженерія
(повна назва освітньої програми)

Керівник: проф. Наталія АКСАК
(посада, власне ім’я, прізвище)

Допускається до захисту

Завідувач кафедри ЕОМ _____ Олег РУДЕНКО
(підпис) (власне ім’я, прізвище)

2025 р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління

Кафедра комп'ютерних інтелектуальних технологій та систем

Рівень вищої освіти перший (бакалаврський)

Спеціальність 123 «Комп’ютерна інженерія»
(код і повна назва)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Комп’ютерна інженерія
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
“____” _____ 20__ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Гасановій Лоліті Михайлівні
(прізвище, ім’я, по батькові)

1. Тема роботи Розробка мобільного додатка для фіксації та повідомлення про проблеми інфраструктури міста

затверджена наказом по університету від “ 21 ” травня 2025 р. № 399 Ст

2. Термін подання здобувачем роботи до екзаменаційної комісії 19 червня 2025 р.

3. Вхідні дані до роботи

1) Фреймворк для розробки мобільних додатків (React Native)
2) Документація для навігації в React-додатках
3) Документація Django для розробки API
4) Системи керування базами даних (PostgreSQL)
5) Документація для інтеграції з сервісами геолокації
6) Документація Docker для контейнеризації

4. Перелік питань, що потрібно опрацювати у роботі

1) аналіз предметної області;
2) аналіз використовуваних технологій;
3) програмна реалізація;
4) інструкція користувача;
5) висновки.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій

Слайд-презентація – 17 слайдів

6. Консультанти розділів роботи (заповнюється за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Ознайомлення з літературними	26.05.25-28.05.25	
	джерелами, аналіз та вибір методу		
	вирішення поставленої задачі		Виконано
2	Проектування алгоритмів рішення	28.05.25-1.06.25	Виконано
3	Розробка програмного забезпечення	1.06.25-7.06.25	Виконано
4	Тестування, виправлення програмних помилок	7.06.25-9.06.25	Виконано
5	Збір пристрою та тестування функціоналу	9.06.25-10.06.25	Виконано
6	Оформлення матеріалів кваліфікаційної роботи	10.06.25-15.06.25	Виконано
7	Подання кваліфікаційної роботи керівникові та	15.06.25-16.06.25	Виконано
	її попередній захист		

Дата видачі завдання " 26 " травня 2025 р.

Здобувач _____
(підпис)

Керівник роботи _____ проф. Наталія АКСАК
(підпис) (посада, власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 83 с., 16 рис., 1 табл., 2 дод., 25 джерел.

МОБІЛЬНИЙ ДОДАТОК, ІНФРАСТРУКТУРА МІСТА, ФІКСАЦІЯ ПРОБЛЕМ, REACT NATIVE, DJANGO, POSTGRESQL, DOCKER, API, ГЕОЛОКАЦІЯ, UI/UX, КРОСПЛАТФОРМЕННА РОЗРОБКА

Метою даної кваліфікаційної роботи є розробка мобільного додатка для фіксації та обробки звернень про інфраструктурні проблеми міста.

Клієнтська частина реалізована на крос-платформовому фреймворку React Native для сумісності з iOS та Android, що дозволяє забезпечити широку доступність додатка для користувачів на різних пристроях. Серверна частина розроблена з використанням Django (Python), що забезпечує ефективну обробку запитів і високий рівень безпеки даних.

Система керування базами даних базується на PostgreSQL, гарантуючи надійність, продуктивність та швидкість роботи з великими обсягами даних. Розгортання автоматизовано через Docker для стандартизації та стабільності процесу, що знижує ризик помилок під час налаштування середовища.

Функціонал додатка включає фотофіксацію інфраструктурних проблем з геолокацією (через API), формування структурованих звернень, що сприяє зручності та швидкому реагуванню. Архітектура системи побудована за принципом клієнт-сервер, з інтеграцією через RESTful API для ефективної взаємодії між фронтендом та бекендом.

Для безпеки даних впроваджено JWT-автентифікацію та авторизацію, що забезпечує високий рівень захисту особистої інформації користувачів. Особливу увагу приділено UI/UX дизайну, що дозволяє користувачам швидко та інтуїтивно працювати з додатком.

ABSTRACT

Bachelor’s thesis: 83 pages, 16 figures, 1 tables, 2 appendices, 25 sources.

MOBILE APPLICATION, CITY INFRASTRUCTURE, ISSUE REPORTING, REACT NATIVE, DJANGO, POSTGRESQL, DOCKER, API, GEOLOCATION, UI/UX, CROSS-PLATFORM DEVELOPMENT

The goal of this qualification work is to develop a mobile application for reporting and processing infrastructure issues in the city. The client-side part is implemented using the cross-platform framework React Native, ensuring compatibility with iOS and Android, which allows for broad accessibility of the app across different devices. The server-side is developed using Django (Python), providing efficient request handling and a high level of data security.

The database management system is based on PostgreSQL, ensuring reliability, performance, and speed when working with large volumes of data. Deployment is automated through Docker to standardize and stabilize the process, reducing the risk of errors during environment setup.

The functionality of the app includes photo documentation of infrastructure issues with geolocation (via API), the generation of structured requests, promoting convenience and quick response. The system architecture is client-server based, integrated through a RESTful API for efficient interaction between the frontend and backend.

For data security, JWT authentication and authorization are implemented, ensuring a high level of protection for users' personal information. Special attention is given to UI/UX design, allowing users to work with the app quickly and intuitively.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

ДТП – дорожньо-трпспорта пригода

API – інтерфейс програмування додатків (англ. Application Programming Interface)

JWT – JSON веб-токен (англ. JSON Web Token)

UI – користувацький інтерфейс (англ. User Interface)

UX – досвід користувача (англ. User Experience)

RESTful – архітектура для побудови веб-сервісів (англ. Representational State Transfer)

SQL – мова структурованих запитів (англ. Structured Query Language)

REST – протокол для обміну даними між клієнтом і сервером через HTTP (англ. Representational State Transfer)

GPS – глобальна система позиціонування, яка дозволяє визначати точне місцезнаходження (англ. Global Positioning System)

HTTPS – захищений протокол передачі гіпертексту, що забезпечує шифрування даних між клієнтом і сервером (англ. HyperText Transfer Protocol Secure)

ВСТУП

У сучасному світі цифрові технології стали важливим елементом розвитку міських інфраструктур та управління ними. Особливо це стосується взаємодії між органами місцевого самоврядування та громадянами, оскільки швидкий і ефективний зворотний зв'язок є запорукою належної роботи комунальних служб. Міська інфраструктура постійно стикається з різними проблемами, що потребують термінового реагування: від пошкоджень дорожнього покриття до несправного освітлення та аварійних ситуацій з комунікаціями. Однак традиційні методи звернення громадян до органів влади часто є недостатньо оперативними або малоефективними. Це спричиняє зниження довіри до місцевої влади і зменшує громадську активність у вирішенні міських проблем.

У зв’язку з цим виникає необхідність у створенні інноваційного цифрового інструменту, що дозволяє швидко фіксувати та подавати інформацію про інфраструктурні проблеми. Мобільні додатки стають одним з найбільш зручних та доступних інструментів для швидкої взаємодії між громадянами та органами місцевого самоврядування. Вони дають змогу не лише надсилати повідомлення про проблеми, але й забезпечують швидку обробку даних, автоматизацію процесів, а також відстеження статусу звернень у реальному часі.

Актуальність цієї роботи обумовлена необхідністю створення ефективного інструменту для взаємодії між громадянами та місцевими органами влади, а також прагненням підвищити ефективність управління міською інфраструктурою. Зручність, доступність та оперативність такого інструменту допоможе не тільки підвищити ефективність вирішення проблем, а й сприяти активному залученню громадян до процесу покращення міського середовища. Створення мобільного додатка для швидкої фіксації проблем міської інфраструктури є надзвичайно важливим для покращення

взаємодії між громадянами та комунальними службами. Оскільки більшість населення активно використовує мобільні пристрої, такий інструмент дозволяє забезпечити доступність і швидкість взаємодії.

Метою цієї кваліфікаційної роботи є розробка мобільного додатка, що дозволяє громадянам фіксувати проблеми міської інфраструктури, зокрема пошкодження доріг, несправність освітлення, аварійні дерева, засмічення територій тощо, за допомогою фотографій, геолокації та опису проблеми. Зібрані дані будуть автоматично надсилатися відповідним службам для їхнього вирішення або зберігатись для подальшого аналізу. Це дозволить оперативно реагувати на проблеми та забезпечити зворотний зв'язок з громадянами.

Завдання цієї кваліфікаційної роботи полягає у проектуванні архітектури мобільного додатка, який підтримує функціонал фіксації та подачі повідомлень про інфраструктурні проблеми, а також у виборі відповідних технологій для розробки додатка [16]. Будуть реалізовані основні функції додатка, включаючи фотофіксацію, геолокацію, формування структурованих звернень та їхнє автоматичне відправлення. Також важливо забезпечити захист персональних даних користувачів, впровадивши сучасні механізми автентифікації та авторизації (JWT), а також створити інтуїтивно зрозумілий інтерфейс користувача (UI/UX), що полегшить взаємодію з додатком.

Практична значущість роботи полягає в розробці мобільного додатка, який сприятиме покращенню взаємодії між громадянами та комунальними службами. Цей додаток дозволить не лише оперативно реагувати на проблеми міської інфраструктури, а й забезпечить надійний зворотний зв'язок, що позитивно вплине на управління міським середовищем і підвищить довіру громадян до органів влади.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Міська інфраструктура

Міська інфраструктура складається з кількох основних складових: дорожнє покриття, освітлення, комунікації та екологічні системи. Кожен із цих елементів має важливе значення для функціонування міста, і проблеми в цих сферах потребують оперативного реагування.

За оцінками Державного агентства автомобільних доріг України, 39% міської вулично-дорожньої мережі перебуває в неналежному стані (вибоїни, тріщини, руйнування покриття) [2; 3]; у Києві, наприклад, щороку фіксується понад 2 000 звернень щодо аварійних ям тільки на проїзній частині [4]. Проблеми з вуличним освітленням не менш критичні: згідно з даними Асоціації міст України, кожна третя опора (≈31%) потребує заміни або ремонту, що безпосередньо впливає на рівень дорожньо-транспортного травматизму та криміногенної обстановки [1].

Комунікаційна інфраструктура також зношена: понад 45% водопровідних та 37% каналізаційних мереж експлуатуються понад нормативний строк у 30 років, що зумовлює регулярні прориви й затоплення вулиць [2]. У сфері екології Держекоінспекція щорічно виявляє понад 1 800 несанкціонованих сміттєзвалищ та фіксує перевищення ГДК твердих частинок $PM_{2.5}$ у великих містах протягом 20–35% днів року [2]. Сукупність цих показників свідчить про гостру потребу в оперативному механізмі зворотного зв’язку між громадянами та комунальними службами. Запропонований у роботі мобільний додаток забезпечує фіксацію проблем через фото- та геолокаційні дані, автоматизує маршрутизацію звернень відповідальним підрозділам і надає прозорий канал для відстеження статусу виправлення, що потенційно скорочує час реагування та підвищує ефективність утримання міської інфраструктури.

1.2 Інфраструктурні проблеми

Інфраструктурні збої у міському середовищі охоплюють пошкодження доріг, аварії на інженерних мережах, засмічення територій і поломки обладнання, кожен із цих типів безпосередньо впливає на комфорт і безпеку мешканців. Майже 39% міської вулично-дорожньої мережі України класифікується як такою, що перебуває в незадовільному стані, а у столиці щорічно реєструється понад 2 000 звернень щодо аварійних ям на проїзній частині, що суттєво підвищує ризик ДТП і логістичні витрати [3]. Інженерні комунікації так само зношені: 45% водопровідних і 37% каналізаційних мереж експлуатуються понад нормативний строк, спричиняючи регулярні прориви, затоплення вулиць і збої в ресурсопостачанні [2]. Не менш актуальною є проблема засмічення довкілля: Держекоінспекція щороку фіксує понад 1 800 стихійних сміттєзвалищ, які погіршують санітарний стан і якість повітря [2]. Дефіцит належного освітлення поглиблює негативний ефект, оскільки приблизно 31% вуличних опор потребує ремонту або заміни, що корелює зі зростанням рівня злочинності й небезпеки для пішоходів у темний час доби [1]. Усі ці статистично підтверджені виклики вказують на потребу в оперативному механізмі зворотного зв’язку; запропонований мобільний додаток із функціями фото- та геофіксації дозволяє мешканцям негайно повідомляти про будь-який із перелічених типів проблем, що, своєю чергою, скорочує час реагування служб і підвищує ефективність управління міською інфраструктурою.

1.3 Технічні аспекти інфраструктури

Інфраструктурні збої безпосередньо впливають на соціально-економічну динаміку міста: зношені дороги, аварії на інженерних мережах, засмічення та поломки об’єктів благоустрою формують сукупний ризик для безпеки й якості життя населення. За даними Державного агентства

автомобільних доріг, у незадовільному стані перебуває 39% міської вулично-дорожньої мережі [3], що корелює з понад 2 000 щорічних звернень щодо аварійних ям лише у столиці [4]; це підвищує імовірність ДТП, утворює затори та підвищує логістичні витрати бізнесу. Дефіцит належного освітлення поглиблює проблему: кожна третя опора вуличного світла потребує заміни, а недостатня освітленість корелює зі зростанням злочинності на 11% у темний час доби [1]. Зношеність інженерних комунікацій теж критична: 45% водопровідних і 37% каналізаційних мереж експлуатуються понад нормативний строк, що зумовлює регулярні пориви та підтоплення [7]. Екологічний вимір демонструє щорічне виявлення понад 1800 стихійних сміттєзвалищ, які погіршують санітарний стан і якість повітря [2]. У сукупності ці фактори обумовлюють необхідність оперативного виявлення та ліквідації порушень; мобільний додаток з функціями фото- та геофіксації може зменшити час реагування міських служб, забезпечити прозорий моніторинг та оптимізувати використання ресурсів, що є принципово важливим для сталого розвитку урбаністичного середовища.

1.4 Проблема зв'язку адміністрації з громадськістю

Рівень ефективності міського управління безпосередньо залежить від якісної двосторонньої комунікації між громадянами та органами місцевого самоврядування; водночас опитування Київського міжнародного інституту соціології засвідчує, що місцевій владі нині довіряє лише близько 50% населення, тоді як 46% висловлюють недовіру [5]. Однією з причин такого скепсису є повільне та непрозоре опрацювання звернень у традиційних каналах (письмові листи, телефонні дзвінки, особисті прийоми), що не забезпечує своєчасного зворотного зв’язку. Одночасно країна демонструє високу цифрову готовність: мобільним застосунком “Дія” вже користуються понад 22,5 млн українців [6], а проникнення сучасних смартфонів охоплює

практично всю аудиторію (частка Android-пристроїв становить 67,2%, iOS – 32,3% [25]). Враховуючи ці тенденції, запровадження мобільної платформи для подання й відстеження інфраструктурних звернень стає логічним кроком для підвищення оперативності, прозорості та підзвітності міських служб: додаток, розроблений у межах цієї роботи, інтегрується з інформаційними системами муніципалітету, забезпечує громадянам миттєве інформування про статус заявки і, завдяки геоміткам і мультимедіа, надає службам повні дані для пріоритезації та швидкого реагування; таким чином цифровий канал не лише скорочує час вирішення проблем, а й відчутно зміцнює довіру населення до місцевої влади, сприяючи більш демократичному та ефективному управлінню міською інфраструктурою.

1.5 Огляд існуючих рішень

Сучасний ринок цифрових рішень для міського управління пропонує декілька платформ, що полегшують взаємодію між громадянами та органами влади. У цьому розділі проведено детальний аналіз ключових існуючих рішень, які вирішують схожі до данного завдання проблеми.

1.5.1 Мобільний додаток FixMyStreet

FixMyStreet – це безкоштовна веб- та мобільна платформа від британської організації mySociety, що з 2007 року дозволяє мешканцям повідомляти про локальні інфраструктурні неполадки трьома простими кроками: вибір місця на карті, короткий опис і фото. Звернення автоматично надходять до відповідальних місцевих рад, а всі повідомлення відображаються на інтерактивній мапі, що створює публічний аудит і підвищує відповідальність влади.

До переваг сервісу належать зрозумілий інтерфейс із низьким порогом входу, прозорість процесу й можливість масштабування на десятки

муніципалітетів. Проте ефективність залежить від технічної інтеграції з радою – за її відсутності звернення лише відправляється електронною поштою та може бути проігнороване. Зворотний зв’язок обмежується статусом “відправлено/закрито”, а мобільний інтерфейс позбавлений push-сповіщень, офлайн-режиму та адаптований лише під британську адресну систему (рисунок 1. 1).

Застосунок підтримує дороги, освітлення, комунікації й екологію; має push-повідомлення та детальне відстеження етапів виконання; інтегрується з міськими BackOffice API; локалізує адресний реєстр; і використовує крос-платформенний інтерфейс на React Native для сучасного UX. Це дає швидкий, зручний і прозорий канал комунікації українських громадян із міськими службами.

Рисунок 1.1 – Структура проекту клієнтської частини

1.5.2 Платформа “Дія”

Платформа “Дія” позиціонується як універсальний електронний сервіс самообслуговування громадян, до функціоналу якого у 2024-2025 рр (рисунок 1.2). додано модулі подання скарг: нині користувач може повідомити про відсутність або погану якість мобільного зв’язку, заповнивши коротку форму з автоматичним визначенням геолокації та

вибором оператора [8], причому уряд затвердив офіційний порядок обробки таких звернень для подальшого моніторингу якості електронних комунікацій [10]; у I кв. 2025 р. планується впровадження AI-асистента, який прийматиме скарги з різних секторів міської інфраструктури та автоматично переадресовуватиме їх до профільних державних органів [9]. Концептуальна перевага “Дії” полягає у високій проникності (понад 22 млн користувачів), централізації сервісів “у одному вікні” та поступовій автоматизації маршрутизації звернень.

До переваг належать: зручність мобільного доступу й мінімальна тривалість заповнення форми; прозорість – користувач отримує push-сповіщення про зміну статусу скарги; інтеграція зі статистичними панелями для органів влади.

Серед недоліків слід відзначити обмеженість тематики нинішніх скарг, залежність від стабільності інтернет-каналу самого користувача та ризики перевантаження служби підтримки за відсутності достатнього штату операторів до повного запуску AI-компонента. Попри ці обмеження, досвід “Дії” демонструє, що мобільна платформа здатна істотно підвищити оперативність і прозорість взаємодії громадян з владою, що кореспондує з цілями розроблюваного у дипломній роботі застосунку.

Рисунок 1.2 – Платформа “Дія”

1.5.3 CitySourced

CitySourced – американський SaaS-застосунок для муніципалітетів, доступний у “white-label” форматі з 2011 року. Мешканець фотографує проблему (наприклад, вибоїну, зламаний світлофор чи графіті), GPS-координати автоматично фіксуються, а сформоване звернення надходить до бек-офісу міста через інтегрований модуль CRM (рисунок 1.3).

Цей сервіс пропонує готовий набір API та аналітичну панель із тепловими картами інцидентів, інструментами пріоритезації заявок і можливістю призначати відповідальних департаментів. Користувачі відстежують статус у режимі реального часу й отримують push-сповіщення, а міста можуть гнучко кастомізувати інтерфейс під власний бренд. Платформа працює в хмарі й підтримує iOS та Android, а також включає модуль електронних опитувань для консультацій із громадою.

Рисунок 1.3 – CitySourced

Водночас щорічна ліцензія від $10 000 робить рішення недешевим для малих громад, а збереження даних на американських серверах AWS створює питання суверенітету інформації для українських міст. Геокодування та

адресний пошук “з коробки” зорієнтовані на Nominatim/USPS і потребують глибокої локалізації під українські реєстри. Крім того, платформа фокусується передусім на базових інфраструктурних інцидентах (дороги, світло, сміття) і не містить вбудованих модулів для складних аварій, таких як водоканал чи газова служба, тому розширення функціоналу вимагатиме окремої розробки.

1.5.4 SeeClickFix

SeeClickFix – американська хмарна платформа “community reporting”, що з 2008 року дозволяє мешканцям фотографувати місцеві проблеми, додавати короткий опис і автоматично зазначати GPS-координати. Після публікації звернення з’являється на інтерактивній карті й завдяки вбудованому маршрутизатору надходить до відповідного департаменту міської адміністрації (рисунок 1.4).

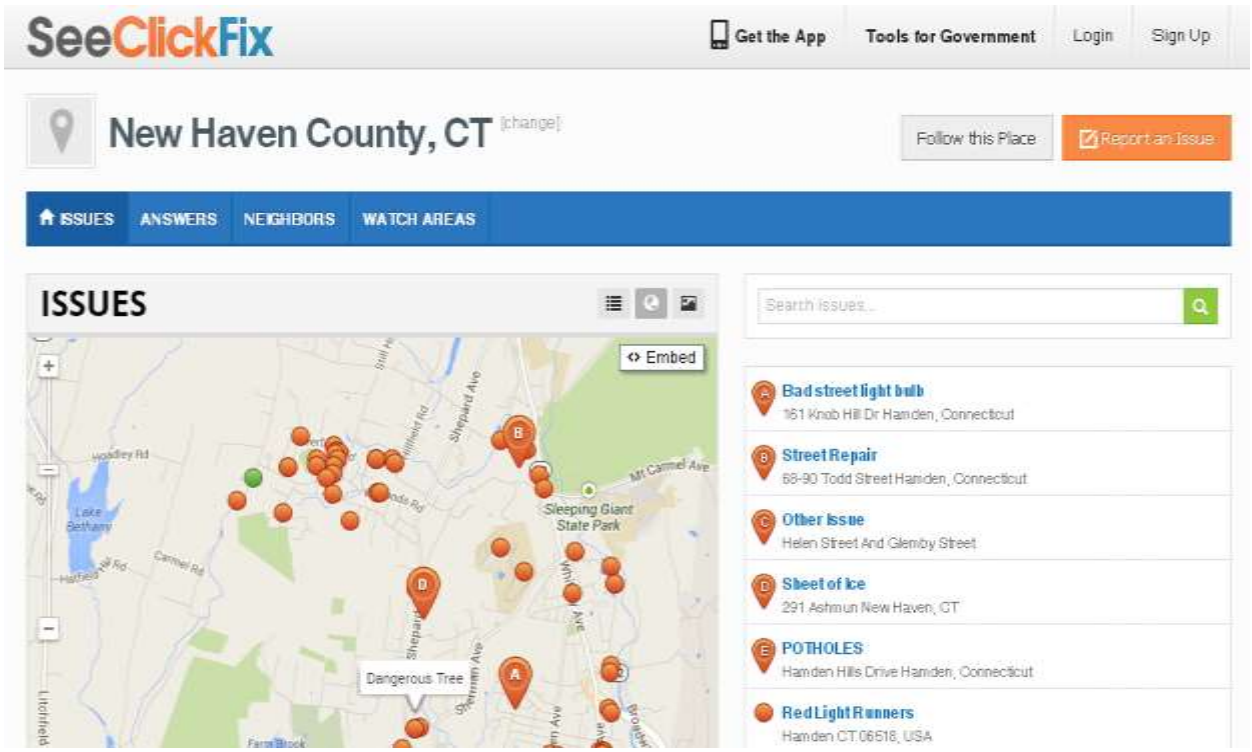


Рисунок 1.4 – CitySourced

Платформа інтегрована з CRM Clariti (раніше Rock Solid) та GIS-рішеннями ESRI, що дає чиновникам готові інструменти для аналітики, моніторингу SLA та автоматичного створення робочих ордерів. SeeClickFix підтримує iOS, Android і веб-портал, надсилає push-сповіщення про зміну статусу заявки і має гейміфікаційний модуль “Civic Points” для заохочення користувачів.

З понад 8 мільйонами звернень у США й Канаді та відкритою API-шиною для підключення сторонніх сервісів, SeeClickFix пропонує зрілу інфраструктуру та white-label-брендування. Водночас платформа передбачає ліцензійні платежі, зберігає дані на американських серверах AWS і використовує довідники, прив’язані до місцевих адресних реєстрів, тому для України потрібна повна локалізація та додаткові модулі для складних аварій.

1.6 Висновки та формування вимог

Аналіз поточного стану міської інфраструктури та існуючих прикладів цифрових сервісів (FixMyStreet, SeeClickFix, CitySourced, “Дія”) показав, що найбільшою проблемою в українських реаліях залишається фрагментований зворотний зв’язок між громадянами та муніципалітетами.

Статистика свідчить: 39% дорожньої мережі міст перебуває у незадовільному стані, понад 45% водопровідних мереж фізично зношені, а щороку реєструється близько 1 800 стихійних сміттєзвалищ [1–5]. Існуючі платформи-аналоги довели ефективність моделі “фото, геомітка, автоматична маршрутизація”, однак жодна з них не поєднує повну локалізацію під українські адресні реєстри, інтеграцію з комунальними аварійними службами та безоплатний доступ для користувача.

Отже, розробка крос-платформового мобільного додатка на React Native з бекендом Django є доцільною й обґрунтованою: рішення покращує оперативність реагування, підвищує прозорість процесів і зміцнює довіру громадян до органів місцевого самоврядування.

Перелік функціональних вимог до системи:

- можливість створення звернення з фото, описом, автоматичною фіксацією GPS-координат і ручним введенням адреси;
- категоризація проблем (дороги, освітлення, комунікації, екологія) з подальшою маршрутизацією до профільних служб;
- публічна інтерактивна карта з фільтрами за категорією та статусом;
- особистий кабінет і персональними даними;

Перелік нефункціональних вимог до системи:

- продуктивність: обробка 95% запитів REST API ≤ 300 мс при навантаженні 1 000 одночасних користувачів;
- масштабованість: контейнеризація Docker та Docker Compose, можливість горизонтального масштабування бекенду;
- безпека: автентифікація JWT, шифрування трафіку HTTPS, зберігання даних у серверному сегменті, що відповідає вимогам КСЗІ;
- доступність: підтримка iOS 13+ та Android 8+, адаптивний UI, локалізація українською/англійською;
- конфіденційність;
- надійність: резервне копіювання бази даних щодня, цільове показник доступності 99,5% на рік.

2 АНАЛІЗ ВИКОРИСТАНИХ ТЕХНОЛОГІЙ

2.1 Використання React Native для розробки мобільного застосунку

У сучасній розробці мобільних додатків важливим аспектом є створення крос-платформенних рішень, що забезпечують високу продуктивність та швидке впровадження функціоналу для різних операційних систем. React Native є одним з найбільш популярних фреймворків для розробки мобільних додатків, який дозволяє створювати додатки одночасно для платформ iOS та Android, використовуючи одну кодову базу. Завдяки використанню JavaScript та можливості інтеграції з нативними компонентами, React Native забезпечує ефективний процес розробки, зберігаючи високу продуктивність та зручний інтерфейс для кінцевих користувачів. Цей підрозділ присвячений детальному аналізу можливостей React Native, а також обґрунтуванню його вибору для розробки мобільного застосунку в рамках цієї дипломної роботи [22].

2.1.1 Обґрунтування вибору React Native

При виборі технології для розробки мобільного застосунку ключовим був крос-платформений підхід, що дозволяє одночасно підтримувати iOS і Android. React Native забезпечує єдину кодову базу для обох платформ, суттєво зменшуючи час та витрати на розробку й подальшу підтримку, що особливо важливо для стартапів і малих команд із обмеженими ресурсами.

Оскільки React Native ґрунтується на JavaScript і React [21], знайомих багатьом веб-розробникам, освоєння технології проходить швидко, а велика спільнота та численні бібліотеки дозволяють оперативно вирішувати типові завдання та дода--вати нові функції без написання “з нуля”. Крім того, React Native підтримує механізм гарячого перезавантаження (hot reload), що

прискорює ітерації розробки й тестування, а сервіси типу CodePush дають змогу випускати оновлення без необхідності проходити повторну модерацію в App Store і Google Play.

Додатковою перевагою є прямий доступ до нативних модулів (GPS, камера, сенсори), який гарантує високу продуктивність та можливість використовувати специфічний для платформи функціонал, водночас забезпечуючи “рідний” інтерфейс і відчуття застосунку. Це поєднання швидкості розробки, можливостей нативної інтеграції та широкої екосистеми робить React Native оптимальним вибором для поточного проєкту.

2.1.2 Архітектура та основні можливості React Native

Архітектура React Native ґрунтується на механізмі Bridge, який синхронізує JavaScript-потік, де виконується логіка та обробляються події, із Native-потоком, що відповідає за рендер і виклики системних API (камера, GPS, файлові операції) [18]. Єдина кодова база для iOS і Android прискорює розробку, а підключення нативних модулів на Objective-C, Java чи Kotlin дає доступ до специфічних функцій пристроїв і складних анімацій [18]. Гаряче перезавантаження пришвидшує цикл редагування, велика екосистема бібліотек спрощує інтеграцію сторонніх сервісів, а CodePush забезпечує “over-the-air” оновлення без повторної модерації у сторах. Сукупно ці можливості гарантують високу продуктивність, гнучкість і скорочення time-to-market крос-платформених мобільних застосунків.

2.1.3 Бібліотеки та інструменти для розробки на React Native

React Native підтримує велику кількість бібліотек і інструментів, що спрощують розробку мобільних додатків, підвищують продуктивність і зручність користувачів, а також дозволяють інтегрувати додаток з різними сервісами та технологіями, що робить React Native дуже потуєним

інструментом та зручним для написання проектів. У цьому підрозділі розглядаються основні бібліотеки та інструменти, які були використані під час розробки клієнтської частини додатку в рамках кваліфікаційної роботи (таблиця 2.1).

Таблиця 2.1 – Бібліотеки та інструменти для розробки на React Native

Бібліотека/ Інструмент	Призначення	Ключові переваги	Використання в проєкті	Примітки
1	2	3	4	5
React Navigation [24]	Керування навігацією в додатку	Легкість інтеграції; Багата документація; Гнучкі анімації переходів.	Реалізація навігації між екранами (стекова, табова)	Підтримує вкладені навігатори, модальні вікна
React Hooks (useEffect)	Керування побічними ефектами у функціональних компонентах React	Спрощує роботу з життєвим циклом компонента; Дозволяє виконувати код після рендерингу; Замінює класові методи.	Отримання даних з API при монтуванні компонент; Підписка на зовнішні події; Очищення ресурсів.	Важливо правильно вказувати залежності (dependencies), щоб уникнути зайвих викликів або "зациклювання"
React Native Image Picker [22]	Робота з медіафайлами	Доступ до галереї/камери; Просте API; Підтримка різних форматів.	Додавання фото до звернень	Потрібні дозволи для доступу до камери/сховища

Продовження таблиці 2.1

1	2	3	4	5
Fetch API [17]	Виконання HTTP-запитів у клієнтських додатках	Вбудований у сучасні браузери; Підтримка промісів для асинхронної обробки; Простий та інтуїтивний синтаксис;	Отримання даних з REST API; Відправка форм та даних на сервер; Взаємодія з бекендом через CRUD-операції.	Вимагає додаткової обробки помилок (не відхиляє проміс при HTTP-помилках 4xx/5xx)
React Native Maps [23]	Інтеграція картографічних сервісів	Підтримка Google Maps/Mapbox; Маркери, маршрути; Гнучкі налаштування.	Візуалізація місць інфраструктурних проблем	Вимагає API-ключі для картографічних сервісів

2.2 Використання Django для серверної частини додатка

Django – високопродуктивний Python-фреймворк, який забезпечує масштабованість, безпеку та ефективність для створення серверної частини веб-додатків. Завдяки вбудованим механізмам роботи з базами даних (ORM і міграції), обробки запитів і захисту від типових уразливостей, він гарантує стабільну роботу під великим навантаженням. Django також полегшує розгортання RESTful API для клієнтських застосунків та пришвидшує

розробку завдяки широкому набору готових інструментів. Його вибір обґрунтовано високою продуктивністю, надійністю та зручністю підтримки сучасних веб-систем [14].

2.2.1 Django та Django REST Framework

У рамках цієї дипломної роботи для реалізації серверної частини було обрано Django разом з його розширенням Django REST Framework (DRF), яке спеціалізується на створенні RESTful API. Django надає інструменти для швидкого розгортання веб-сервісів, що включають вбудовані рішення для обробки запитів, а також підтримку баз даних через ORM (Object-Relational Mapping). Однак саме Django REST Framework дозволяє розробникам ефективно будувати та обслуговувати API-сервіси, що підтримують стандартні методи HTTP (GET, POST, PUT, DELETE), а також обробляти запити у форматах JSON, XML та інших, що робить серверну частину системи, що побудована на Django, гнучкою та адаптованою до вимог клієнтських додатків.

Django REST Framework також має вбудовану підтримку для автентифікації, авторизації та валідації даних, що суттєво спрощує розробку сервісів, які повинні працювати з великою кількістю користувачів. Це розширення дозволяє швидко налаштовувати сервіси, що мають складну бізнес-логіку та потребують інтеграції з іншими компонентами системи, що робить Django ідеальним вибором для реалізації серверної частини даного проєкту [13].

2.2.2 Архітектура RESTful API

Архітектура RESTful API була вибрана для створення серверної частини додатка через її простоту, масштабованість і гнучкість. REST (Representational State Transfer) є архітектурним стилем, який визначає спосіб

організації взаємодії між клієнтом і сервером через HTTP-запити. Оскільки REST не накладає обмежень на типи клієнтських додатків, ці сервіси можна використовувати з різними системами, такими як веб-додатки, мобільні додатки на React Native або інші сервіси, що забезпечують взаємодію через HTTP.

RESTful API має низку переваг: це дозволяє створити гнучку архітектуру, яка може обробляти високі навантаження, а також забезпечує легку інтеграцію з іншими зовнішніми системами і платформами. Завдяки стандарту HTTP і JSON, клієнтські додатки можуть здійснювати запити до сервера для отримання або відправлення даних, що необхідно для обробки інфраструктурних проблем. Ця архітектура дозволяє забезпечити ефективне управління даними і спрощує процес налаштування інтеграцій з іншими веб-сервісами та платформами.

2.2.3 Системи аутентифікації та безпеки

Одним з найважливіших аспектів для серверної частини є забезпечення безпеки та конфіденційності даних. Django надає потужні вбудовані механізми безпеки, такі як захист від SQL-ін’єкцій, CSRF, XSS та інші стандартні уразливості. Для реалізації аутентифікації та авторизації в системі використовуються механізми JWT, які дозволяють здійснювати безпечну аутентифікацію користувачів через токени, замість зберігання сесій на сервері.

JWT дає змогу безпечно передавати дані між сервером і клієнтом через HTTP-запити, що гарантує їхню цілісність та конфіденційність [15]. Кожен раз після успішної автентифікації сервер генерує токен, який відправляється користувачеві, і далі цей токен використовується для доступу до захищених ресурсів сервера.

Вибір Django для серверної частини цього проєкту обґрунтований його здатністю забезпечити швидку розробку і масштабованість при збереженні

високого рівня безпеки. Використання Django REST Framework дозволяє створювати ефективні API, які забезпечують зручну інтеграцію з мобільними додатками та іншими системами. Вбудовані механізми безпеки забезпечують захист даних користувачів на всіх етапах обробки запитів, а також дозволяють уникнути можливих загроз безпеці.

Отже, Django є ідеальним вибором для реалізації серверної частини додатка завдяки своїм багатим можливостям і гнучкості, що дозволяє легко масштабувати систему і інтегрувати нові функції.

2.3 База даних PostgreSQL

PostgreSQL є однією з найбільш популярних реляційних систем управління базами даних (СУБД), відомою своєю стабільністю, розширюваністю та високою продуктивністю. Вона широко використовується для розробки складних веб-додатків, оскільки забезпечує надійність та ефективність роботи з великими обсягами даних та є безкоштовною.

2.3.1 Переваги PostgreSQL для проекту

PostgreSQL є вибором для цього проекту через свої численні переваги, серед яких можна виокремити кілька основних. По-перше, вона є безкоштовною і відкритою, що дозволяє значно знизити витрати на ліцензування та забезпечує повну контрольованість за процесами розробки. Це має велике значення для проектів з обмеженим бюджетом, таких як цей, де важливо зберігати доступ до бази даних без додаткових витрат на ліцензії.

По-друге, PostgreSQL підтримує масштабованість. Це дозволяє адаптувати систему під різні вимоги проекту – від невеликих до великих обсягів даних. Ключовим елементом є підтримка індексів та можливість роботи з партиціями даних, що дозволяє збільшувати швидкість запитів.

По-третє, PostgreSQL підтримує потужні функції для забезпечення безпеки даних, такі як шифрування, контроль доступу на рівні записів, підтримка аутентифікації через SSL. Це особливо важливо для проекту, що працює з конфіденційними даними користувачів, забезпечуючи належний рівень захисту інформації.

Крім того, PostgreSQL підтримує ACID-принципи (атомарність, консистентність, ізоляція та довговічність), що робить її надійним вибором для проектів, де потрібна висока стабільність і коректність транзакцій, особливо при великому навантаженні та одночасному доступі до бази даних.

2.3.2 Геопросторові можливості PostgreSQL (PostGIS)

PostGIS – розширення до PostgreSQL, що додає просторові типи даних (точки, лінії, полігони), індексовані запити за координатами та операції над геометріями, зокрема обчислення відстаней, перевірку вміщення й перетину областей. У застосунку воно зберігає точки інфраструктурних проблем і дає змогу оперативно знаходити найближчі інциденти для карти, а також виконувати буферизацію та об’єднання геометрій без зовнішніх сервісів. Підтримка GeoJSON і WKT спрощує обмін геоданими з клієнтськими бібліотеками на кшталт Google Maps чи Leaflet. Таким чином, комбінація PostgreSQL і PostGIS забезпечує єдину, продуктивну платформу для роботи з просторовою інформацією.

2.4 Використання Docker для контейнеризації

Docker є популярною технологією для контейнеризації додатків, що дозволяє створювати, тестувати та розгортати додатки в ізольованих середовищах – контейнерах. Використання Docker для контейнеризації в рамках цього проєкту забезпечує високий рівень стабільності та масштабованості системи, дозволяючи швидко налаштувати середовище для

розробки та продуктивне середовище для розгортання додатка. Docker забезпечує ізоляцію між різними компонентами додатка, що дозволяє зберігати наявність чітких меж між серверними компонентами, базою даних і іншими службами, а також дає змогу зручно управляти конфігураціями середовища без необхідності змінювати налаштування хост-системи.

2.4.1 Принципи контейнеризації додатка за допомогою Docker

Основний принцип роботи Docker полягає в тому, що кожен додаток запускається в ізольованому середовищі, яке називається контейнером. Контейнер містить не тільки сам додаток, але й усі необхідні для його роботи залежності: бібліотеки, середовище виконання, налаштування та конфігураційні файли. Завдяки цьому контейнер може бути розгорнутий на будь-якому сервері, де встановлений Docker, і працювати однаково незалежно від конфігурації хост-системи.

Основними принципами контейнеризації є:

- ізоляція середовища;
- мобільність;
- легкість у розгортанні;
- продуктивність.

У рамках цього проєкту Docker використовується для створення ізольованих середовищ для кожного компоненту серверної частини.

2.4.2 Docker Compose для оркестрації контейнерів

Для ефективного управління кількома контейнерами одночасно в рамках одного додатка використовується Docker Compose – інструмент для визначення та запуску багатоконтейнерних Docker-додатків. Docker Compose дозволяє описувати конфігурацію всіх контейнерів додатка за допомогою одного YAML файлу, що забезпечує спрощене управління середовищем.

Docker Compose дозволяє задати контейнери для різних служб, таких як веб-сервер, база даних, кешування, повідомлення тощо, та автоматично налаштовувати їх взаємодію. У цьому проекті Docker Compose використовується для визначення конфігурації для контейнерів, що містять серверну частину на Django, базу даних PostgreSQL та інші необхідні служби. Використовуючи Docker Compose, можна легко створювати, зупиняти та оновлювати всі компоненти додатка з одного командного файлу, тому він є дуже зручним інструментом.

Основними перевагами використання Docker Compose є:

- спрощення управління багатоконтейнерними додатками;
- легкість у налаштуванні залежностей між контейнерами;
- автоматизація процесів.

Docker Compose дозволяє значно зменшити кількість рутинних задач, таких як налаштування та запуск контейнерів вручну, і спрощує підтримку складних середовищ розробки та продуктивності.

2.5 Інтеграція технологій та архітектура системи

Архітектура системи поєднує мобільний клієнт на React Native із серверною частиною на Django, що взаємодіють через RESTful API, реалізоване за допомогою Django REST Framework. Уся комунікація відбувається за стандартними HTTP-методами, причому дані передаються у форматі JSON. Запити клієнта надходять на ендпоінти DRF, де спочатку виконуються перевірки прав доступу через JWT-токени, потім дані валідуються серіалізаторами, і, нарешті, обробляються бізнес-логікою у view-функціях чи класах. У відповідь сервер повертає структуровані повідомлення із кодом статусу та корисним навантаженням, що дозволяє клієнту коректно відобразити результат або повідомити користувача про помилку.

Обробка геолокації та мультимедіа відбувається повністю на стороні клієнта, що значно знижує навантаження на сервер. Для отримання

координат застосовано Expo Location із підтримкою GPS і мережевого позиціонування, а для роботи з фотографіями – React Native Image Picker, який дозволяє швидко вибрати або зробити знімок. Отримані координати упаковуються у запит до API та зберігаються в PostgreSQL з розширенням PostGIS, що дає змогу виконувати просторові запити на пошук інцидентів поблизу або визначення відстані між точками. Фото та інші медіафайли передаються через multipart-запит і зберігаються в окремому файловому сховищі, а в базі зберігається лише шлях до ресурсу – це прискорює резервне копіювання та полегшує масштабування.

Безпека та надійність комунікації забезпечуються комплексно: всі запити й відповіді шифруються через HTTPS із TLS-сертифікатами, а клієнт зберігає JWT у зашифрованому AsyncStorage із механізмом автоматичного оновлення токенів при їхньому закінченні. Сторона сервера додатково застосовує обмеження частоти запитів (throttling) для запобігання brute-force-атакам, а вбудовані засоби Django – захист від CSRF, SQL-ін’єкцій та XSS. Для оптимізації продуктивності використовуються кешування результатів просторових запитів у Redis, а також HTTP-заголовки ETag і Last-Modified для умовної перевірки актуальності даних. Такий підхід гарантує швидку, безпечну та масштабовану роботу системи, забезпечуючи користувачів актуальною інформацією в режимі реального часу.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Клієнтська частина додатка

3.1.1 Опис архітектури клієнтської частини

Архітектура клієнтської частини мобільного додатка була спроектована з урахуванням принципів модульності, масштабованості та легкості підтримки. Використання React Native забезпечило крос-платформенність, що дозволило створити додаток, здатний працювати як на iOS, так і на Android, з мінімальними змінами в коді. Основною метою проектування архітектури було створення структури, яка забезпечує гнучкість, ефективність взаємодії компонентів, а також зручність для майбутніх оновлень і підтримки.

Структура проекту організована таким чином, щоб кожен компонент мав свою чітко визначену роль, що спрощує навігацію в проекті та підвищує його підтримуваність (рисунок 3.1).

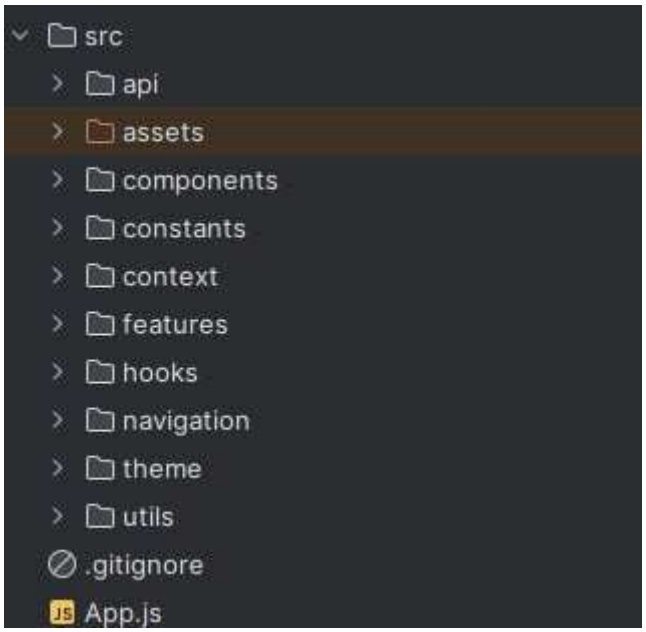


Рисунок 3.1 – Структура проекту клієнтської частини

Архітектура застосунку дотримується принципів MVC: View формує інтерфейс, Model зберігає дані й бізнес-логіку, а роль Controller виконують React-компоненти, що обробляють запити та координують взаємодію з сервером. Чітке розмежування обов’язків кожного компонента знижує складність коду й підвищує його модульність, спрощуючи розробку та підтримку.

Структурно-логічна модель клієнтської реєстрації (рисунок 3.2) окреслює зонування функцій, потік даних і імпортні залежності. По контуру розташовано п’ять екранів майстра: введення телефону, визначення локації, анкетні дані, опис профілю та підтвердження акаунта.

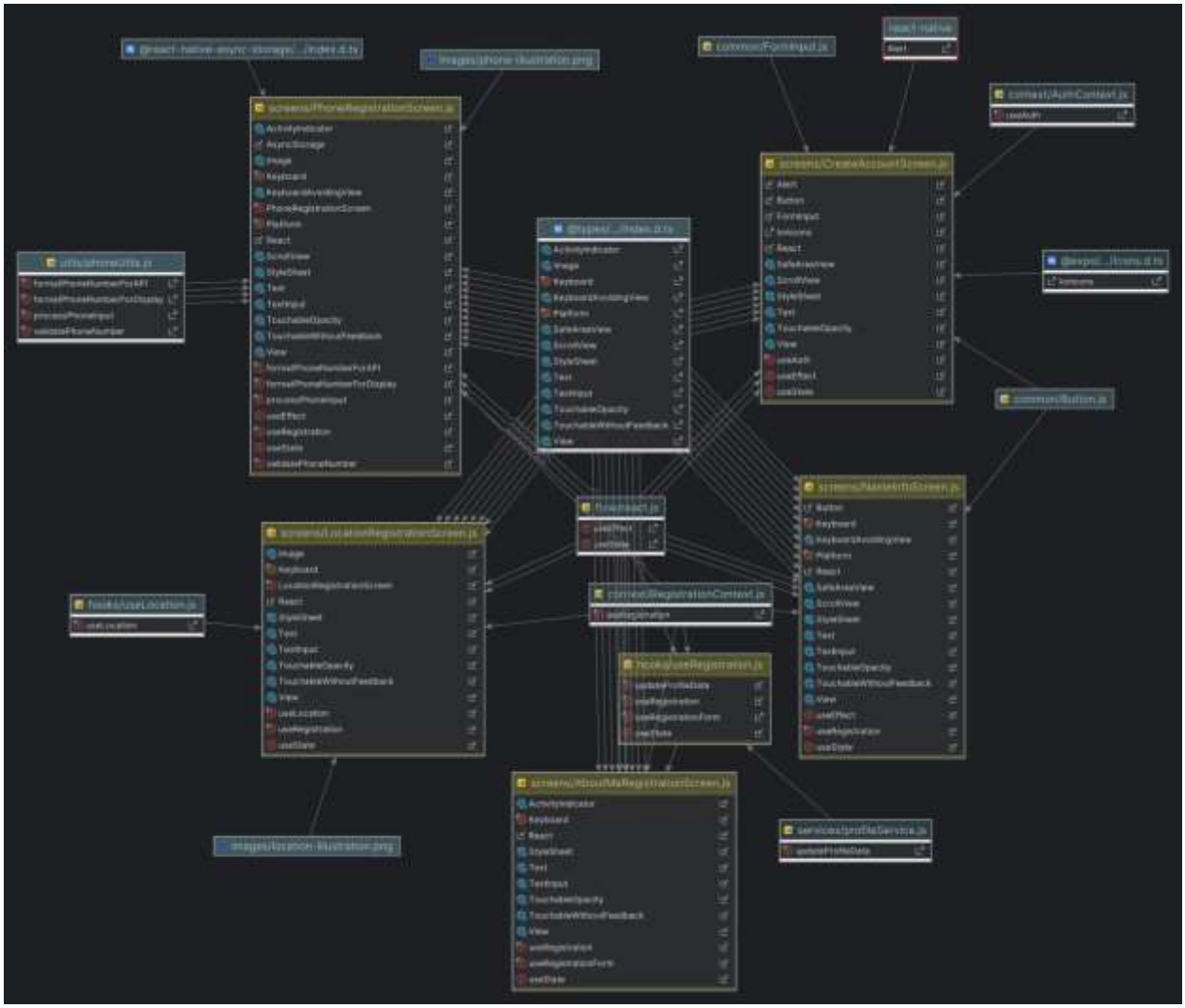


Рисунок 3.2 – Модель клієнтської підсистеми реєстрації користувача

Кожен екран надсилає зібрану інформацію до центрального RegistrationContext – єдиного джерела істини, доступ до якого забезпечує хук useRegistration для модифікації й валідації даних.

Середній рівень утворюють утиліти useLocation (асинхронне отримання координат) і phoneUtils (форматування та перевірка номерів). Їхня незалежність від інших модулів спрощує автономне тестування. Інфраструктурний шар представлений profileService; його метод updateProfileData відправляє агреговані дані на сервер, ілюструючи принцип інверсії залежностей: UI та логіка не залежать від мережевої реалізації.

Спільні елементи FormInput і Button, а також типові декларації й графічні ресурси, підтримують єдиний стиль і повторне використання коду, підвищуючи супроводжуваність системи.

Структурно-логічна модель клієнтської підсистеми інтерактивної карти (рисунок 3.3) відображає взаємозв’язки між модулями, фіксуючи розмежування обов’язків між екраном, гачками обробки геоданих і сервісами віддаленого API.

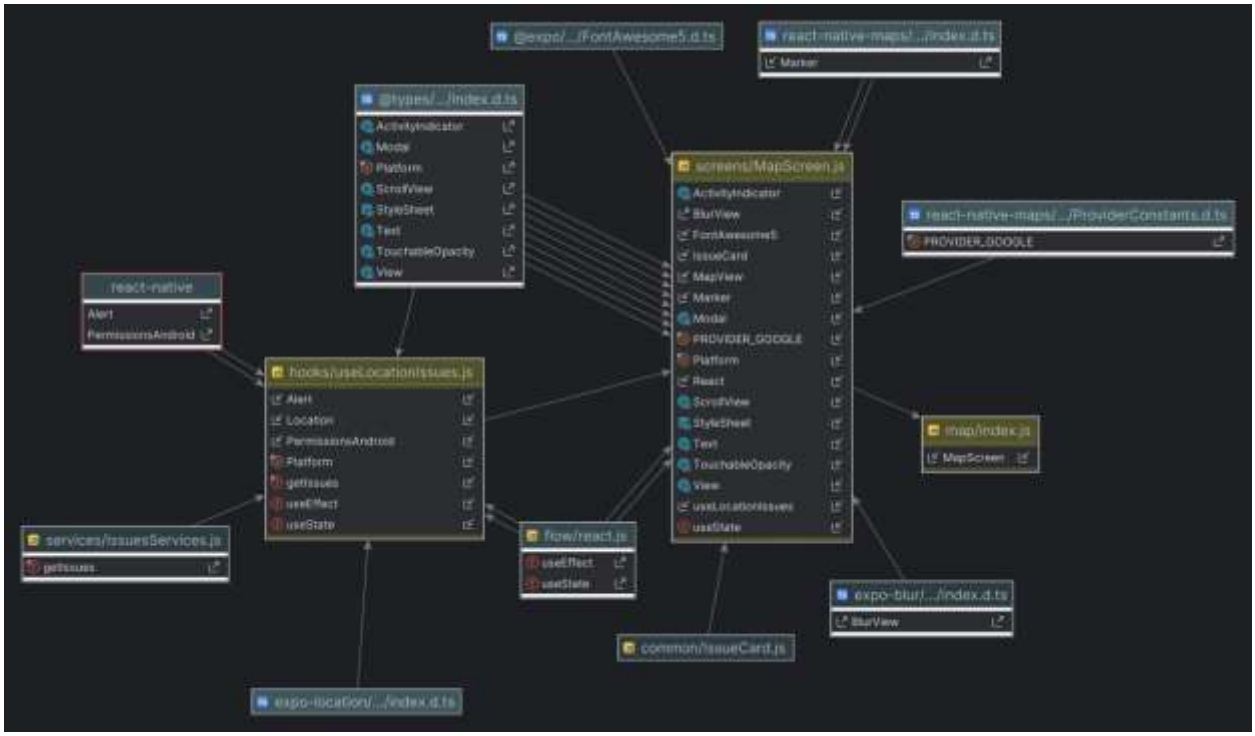


Рисунок 3.3 – Модель клієнтської підсистеми інтерактивної карти

Центральний елемент – MapScreen.js: він рендерить карту з маркерами й картками інцидентів та ініціює подальші запити. Візуальне представлення проблем реалізує повторно використовуваний компонент IssueCard.js. Геодані обробляє гачок useLocationIssues.js, який звертається до expo-location, платформених об’єктів PermissionsAndroid і Alert, а також до сервісу issuesServices.js, що формує запити до бекенда. Статичний контроль типів забезпечують декларації Flow і TypeScript, тоді як API нативних діалогів і системних дозволів імпортуються з пакета react-native. У підсумку схема демонструє трирівневу організацію: зовнішній рівень – екран карти, середній – гачок, що інкапсулює геологіку, внутрішній – сервіс доступу до даних.

3.1.2 Технології та інструменти для розробки

Для клієнтської частини обрано React Native – сучасний крос-платформений фреймворк, який дозволяє використовувати єдину кодову базу для одночасного створення додатків під iOS та Android. Завдяки прямому доступу до нативних компонентів і механізмів рендерингу, React Native забезпечує продуктивність і “рідний” вигляд інтерфейсу.

Інтеграцію з серверною частиною через RESTful API реалізовано за допомогою стандартного fetch або бібліотек типу axios, що дозволяє легко обмінюватися JSON-запитами та підтримувати автентифікацію через JWT-токени. Для роботи з геолокацією використано Expo Location, а для фотографій – React Native Image Picker, завдяки чому клієнт оперативно отримує координати й медіадані без зайвого навантаження на бекенд.

Стан додатка управляється за допомогою React-хуків: useState для локальних змін, useEffect для побічних ефектів (запити до API, обробка геолокації) і useContext для доступу до загальних сервісів (наприклад, темізація або мовні налаштування). Відмова від Redux на користь простіших рішень зменшила обсяг шаблонного коду та пришвидшила ітерації розробки. Крім того, гаряче перезавантаження (Hot Reload) і відлагоджувачі React

Native значно прискорюють цикл “зміна–тест”, що сприяє швидкому впровадженню нових функцій і виправленню багів.

Використання useEffect дозволяє зберегти простоту управління станом, оскільки кожен компонент може самостійно обробляти свої побічні ефекти та взаємодіяти з іншими компонентами через пропси або контекст (лістинг 3.1).

Лістинг 3.1 – Реалізація API запиту для отримання проблем в радіусі

```
useEffect(() => {
    const fetchIssues = async () => {
  try {
    const response = await
fetch(`${base_url}/api/v1/citizens/issues/?latitude=${latitude}&
longitude=${longitude}&radius=${radius}`, {
      method: 'GET',
      headers: {
        'Authorization': `Bearer ${token}`,
        'Content-Type': 'application/json'
      }
    });
```

Для взаємодії з сервером використовується вбудована функція fetch. Вона дозволяє здійснювати HTTP-запити, зокрема методи GET, POST, PUT і DELETE, для передачі даних між клієнтською частиною та сервером. fetch підтримує проміси, що дає змогу обробляти асинхронні запити та отримувати відповіді без блокування головного потоку. Ця функція є стандартним інструментом для роботи з API і дозволяє налаштовувати запити, що є дуже зручним.

Для організації навігації між екранами мобільного додатка була обрана бібліотека React Navigation, яка є стандартом для React Native. React Navigation дозволяє створювати різноманітні типи навігації, включаючи стекову, табличну і бічну навігацію, що відповідає за переміщення між екранами. Крім того, бібліотека підтримує вкладені стеки навігації, що дозволяє створювати складні навігаційні структури з багатьма рівнями переходів (лістинг 3.2).

React Navigation дозволяє налаштовувати анімації при переходах між екранами, що забезпечує плавні і природні анімації, покращуючи користувацький досвід. Вона також інтегрується з іншими інструментами React Native, що робить її ідеальним вибором для управління навігацією.

Лістинг 3.2 – Конфігурація стекової навігації (React Navigation)

```
import React from 'react';
import { NavigationContainer } from '@react-navigation/native';
import { createStackNavigator } from '@react-navigation/stack';
import Index from './screens/Index';
import EditProfile from './screens/EditProfile';

const Stack = createStackNavigator();
const AppNavigator = () => {
  return (
    <NavigationContainer>
      <Stack.Navigator initialRouteName="Home">
        <Stack.Screen name="Index" component={Index} />
        <Stack.Screen name="EditProfile" component={EditProfile}
/>
      </Stack.Navigator>
    </NavigationContainer>
  );
};
export default AppNavigator;
```

Оскільки важливою частиною цього проекту є фіксація інфраструктурних проблем з прив'язкою до геолокації, було використано бібліотеку Expo Location, яка дозволяє отримувати точні координати пристрою. Це забезпечує відправку точних координат проблемних зон на сервер для подальшої обробки (лістинг 3.3).

Лістинг 3.3 – Завантаження проблем з урахуванням поточної геолокації

```
const fetchIssues = async () => {
    try {
      let { status } = await
Location.requestForegroundPermissionsAsync();
      if (status !== "granted") {
        Alert.alert(
```

```
          "Permission Denied",
          "Location permission is required to fetch issues."
        );
        setLoading(false); return;
      }
      let location = await Location.getCurrentPositionAsync({});
      const latitude = location.coords.latitude;
      const longitude = location.coords.longitude;
      setUserLocation({ latitude, longitude });
      const response = await getIssues({latitude,longitude,
        radius: 50,
        limit: 50,
        type: "",});
      if (!response || !response.results ||
!Array.isArray(response.results)) {
        throw new Error("Issues data should be an array.");
      }
      const formattedIssues: Issue[] =
response.results.map((issue: Issue) => ({
        ...issue,
        imageUrls: issue.images?.map((img) => img.photo_url) ||
[], }));
      setIssues(formattedIssues);
    } catch (error) {
      if (error instanceof Error) {
        Alert.alert("Error", error.message || "Failed to load
issues.");
        setIssues([]);
      }
    } finally {
      setLoading(false);
    }
  };
```

Для роботи з мультимедійними даними, такими як фотографії або відео, використовувалась бібліотека React Native Image Picker, яка дозволяє користувачам вибирати медіафайли з галереї або безпосередньо через камеру. Це є ключовою частиною для фіксації інцидентів і передачі даних на сервер (лістинг 3.4).

Лістинг 3.4 – Функція HandlePickImage

```
const handlePickImage = async () => {
    const permissionResult =
      await ImagePicker.requestMediaLibraryPermissionsAsync();
    if (!permissionResult.granted) {
```

```
      Alert.alert(
        "Permission Required",
        "You need to allow access to your gallery to upload a
profile picture."
      );
      return;
    }
    const result = await ImagePicker.launchImageLibraryAsync({
      mediaTypes: ImagePicker.MediaTypeOptions.Images,
      allowsEditing: true,
      aspect: [1, 1],
      quality: 1,
    });
    if (!result.canceled) {
      setUserInfo((prev) => ({
        ...prev,
        profile_picture: result.assets[0].uri,
      }));
      await uploadProfilePicture(result.assets[0].uri);
    }
  };
```

3.1.3 Реалізація інтерфейсу користувача (UI)

Інтерфейс користувача є ключовою складовою мобільного додатка, оскільки визначає зручність і ефективність взаємодії. Його спроєктовано за базовими принципами UI/UX, щоб гарантувати інтуїтивність, адаптивність і швидкий доступ до основних функцій. Дизайн орієнтовано на типові сценарії роботи: подання звернень про інфраструктурні проблеми, перегляд їх деталей і інтерактивну взаємодію з картою; для кожного сценарію створено спеціалізовані компоненти, що забезпечують просту навігацію та комфортне користування.

Форма подачі звернень є важливим компонентом інтерфейсу, який дозволяє користувачам подати звернення щодо інфраструктурних проблем. Форма складається з кількох блоків:

- введення тексту (лістинг 3.5);
- мультимедійні вкладення (лістинг 3.6, 3.7);
- геолокація: автоматичне зчитування геопросторових координат через бібліотеку Expo Location (лістинг 3.8).

Лістинг 3.5 - Компонент для введення заголовку проблеми

```
<TextInput
  placeholder="Issue Title"
  style={styles.input}
  placeholderTextColor="#7D7C7C"
  value={title}
  onChangeText={setTitle}
/>
```

Лістинг 3.6 - Функція вибору зображення

```
const handleImagePick = async () => {
  const result = await ImagePicker.launchImageLibraryAsync({
    mediaTypes: ImagePicker.MediaTypeOptions.Images,
    allowsEditing: true,
    quality: 1,});
  if (!result.canceled && result.assets && result.assets[0]) {
setImages((prevImages) => [...prevImages,
result.assets[0].uri]);}};
```

Лістинг 3.7 - Обробка зображень при відправці

```
if (images.length > 0) {
  const imageUri = images[0];

  const uriParts = imageUri.split("/");
  const fileName = uriParts[uriParts.length - 1];
  let fileType = fileName.includes(".")
    ? fileName.split(".").pop()?.toLowerCase()
    : "jpeg";
  try {
    const response = await fetch(imageUri);
    const blob = await response.blob();
    formData.append("pictures", blob,
`${fileName}.${fileType}`);
  } catch (error) {
    console.error("Error processing image:", error);
    Alert.alert("Error", "Image upload failed.");
    return;
  }
}
```

Інтерактивна карта є ключовим елементом для відображення інфраструктурних проблем на карті. Для цього використовувалась бібліотека

React Native Maps, яка забезпечує інтеграцію з картами Google Maps. Користувачі можуть переглядати розташування поданих звернень за допомогою маркерів, які позначають різні типи проблем. Карта підтримує функціонал масштабування, прокручування та інтерактивної взаємодії, що дозволяє зручно знаходити проблеми на карті(лістинг 3.9, 3.10).

Лістинг 3.8 - Функція вибору зображення

```
const fetchUserLocation = async () => {
  try {setLoadingLocation(true);
    const { status } = await
Location.requestForegroundPermissionsAsync();
    if (status !== "granted") {
      setErrorMsg("Permission to access location was denied.");
      setLoadingLocation(false); return;}
    const location = await Location.getCurrentPositionAsync({});
    const { latitude, longitude } = location.coords;
    console.log(" Got Coordinates:", latitude, longitude);
    const response = await fetch(
`https://maps.googleapis.com/maps/api/geocode/json?latlng=${lati
tude},${longitude}&key=${GOOGLE_PLACES_API_KEY}`
    );
    const data = await response.json();
    if (!data.results || data.results.length === 0) {
      throw new Error("No location data found");
    }
    const addressComponents =
data.results[0].address_components;
    const formattedAddress = data.results[0].formatted_address;
    const city = addressComponents.find((comp) =>
      comp.types.includes("locality"))?.long_name || "Unknown";
    const county = addressComponents.find((comp) =>
  comp.types.includes("administrative_area_level_2"))?.long_name
|| "Unknown";
    const postal_code = addressComponents.find((comp) =>
      comp.types.includes("postal_code"))?.long_name ||
"Unknown";
    setCurrentLocation({
      latitude, longitude, city, county,
      place: formattedAddress,
      postal_code,
    });
  } catch (error) {
    console.error("Error fetching location:", error);
    setErrorMsg("Failed to fetch location.");
  } finally {
    setLoadingLocation(false);}};
```

Лістинг 3.9 – Підключення компонентів карти та служби визначення місцезнаходження

```
import MapView, { Marker, PROVIDER_GOOGLE } from "react-native-
maps"; import * as Location from "expo-location";
```

Лістинг 3.10 – Основний компонент карти

```
{loading ? (
  <View style={styles.loadingContainer}>
    <ActivityIndicator size="large" color="#4C7CFC" />
  </View>
) : (
  <MapView style={styles.map}
    provider={Platform.OS === "android" ? PROVIDER_GOOGLE :
undefined}

    initialRegion={{
      latitude: userLocation?.latitude || 37.7749,
      longitude: userLocation?.longitude || -122.4194,
      latitudeDelta: 0.0922,
      longitudeDelta: 0.0421,
    }}>

    {userLocation && (
      <Marker coordinate={userLocation}
onPress={handleUserPinPress}>
        <FontAwesome5 name="map-marker-alt" size={40}
color="red" />
      </Marker>
    )}
    {issues.map((issue) => (
      <Marker
        key={issue.id}
        coordinate={{ latitude: issue.lat, longitude: issue.lng
}}
        title={issue.title}
        description={issue.description}
        onPress={() => handleMarkerPress(issue)}
      >
        <FontAwesome5 name="map-pin" size={30} color="#4C7CFC"
/>

      </Marker>
    ))}
  </MapView>)}
```

3.1.4 Навігація в додатку

Навігаційна система додатку побудована на основі ієрархічної структури, що включає декілька рівнів навігації для забезпечення логічного групування функціональності та оптимізації користувацького досвіду. Основою навігаційної архітектури є NavigationContainer, який служить кореневим контейнером для всіх навігаційних компонентів та забезпечує глобальний контекст навігації в додатку.

Для реалізації основних переходів між екранами використовується стекова навігація (Stack Navigation), яка забезпечує природну для мобільних пристроїв модель переходів типу "push/pop". Стекова навігація особливо ефективна для лінійних сценаріїв використання, таких як процес реєстрації, створення звернення або перегляд деталей (лістинг 3.11).

Лістинг 3.11 – Стекова навігація

```
import React from 'react';
import { NavigationContainer } from '@react-navigation/native';
import { createStackNavigator } from '@react-navigation/stack';
import Index from './screens/Index';
import EditProfile from './screens/EditProfile';

const Stack = createStackNavigator();


const AppNavigator = () => {
  return (
    <NavigationContainer>
      <Stack.Navigator initialRouteName="Home">
        <Stack.Screen name="Index" component={Index} />
        <Stack.Screen name="EditProfile" component={EditProfile}
/>
      </Stack.Navigator>
    </NavigationContainer>
  );
};


export default AppNavigator;
```

Для програмного керування навігацією в додатку використовується хук useRouter від Expo Router, який надає простий та типобезпечний API для переходів між екранами. Цей підхід забезпечує декларативний стиль навігації та інтеграцію з файловою системою маршрутизації (лістинг 3.12).

Лістинг 3.12 – Реєстрація та використання useRouter

```
import { useRouter } from "expo-router";
const router = useRouter();
const handleRegister = async () => {
  try {
    setError("");
    const data = await register(username, email, password);

    if (data.success) {
      console.log("Registration successful:", data);
      router.push("/get-name");
    } else {
      throw new Error("Registration failed. No token
received.");
    }
  } catch (error) {
    if (error instanceof Error) {
      console.error("Registration error:", error.message);
      setError(error.message || "Registration failed. Please try
again.");
    }
  }
};
```

Для забезпечення плавної роботи навігації в додатку реалізовані наступні оптимізації:

- “Lazy Loading”: Екрани завантажуються тільки при необхідності, що зменшує початковий час завантаження додатку;
- мемоізація компонентів: Використання React.memo() для уникнення непотрібних повторних рендерів навігаційних компонентів;
- кешування стану.

Така архітектура навігації забезпечує масштабованість додатку, легкість підтримки коду та високу продуктивність користувацького інтерфейсу, що є критично важливим для мобільних.

3.1.5 Взаємодія з сервером (API)

Для реалізації взаємодії з сервером на клієнтській стороні було використано fetch для здійснення HTTP-запитів. fetch є вбудованим методом для роботи з асинхронними запитами, що забезпечує зручний і ефективний спосіб взаємодії з сервером без необхідності використання зовнішніх бібліотек, таких як Axios. Основною перевагою fetch є його простота, можливість роботи з промісами та інтеграція з async/await, що дозволяє легко обробляти асинхронні операції.

Взаємодія з сервером передбачає використання основних типів HTTP-запитів, таких як GET, POST, PUT та DELETE. Кожен з цих запитів виконується відповідно до функціональності, яку має виконати сервер та повертає відповідь:

- GET-запити: Використовуються для отримання даних з серверної частини додатка.
- POST-запити: Використовуються для надсилання нових даних на сервер, наприклад, коли користувач подає нове звернення.
- PUT-запити: Використовуються для оновлення існуючих даних на сервері, наприклад, для зміни статусу звернення або редагування інформації про подану проблему.
- DELETE-запити: Використовуються для видалення даних з серверу, наприклад, для видалення звернення або його частини (лістинг 3.13).

Кожен fetch-запит повертає Promise, що дозволяє обробляти результат асинхронно. Після виконання запиту клієнт отримує відповідь від сервера у форматі JSON, яку можна обробити в методі then() або через async/await. У випадку, коли запит не вдався або виникла помилка, обробка помилки здійснюється в методі catch(), що дозволяє ефективно реагувати на непередбачувані ситуації. Обробка асинхронних запитів відбувається за допомогою async/await (лістинг 3.14).

Лістинг 3.13 – DELETE-запит

```
export const removeDownvote = async (issueId) => {
  try {
    const token = await getToken();

    if (!token) throw new Error("Authentication required.");
    const response = await fetch(
      `${BASE_URL}citizens/issues/${String(issueId)}/downvote`,
      {
        method: "DELETE",
        headers: { Authorization: `Bearer ${token}` },
      }
    );
    if (!response.ok) throw new Error("Failed to remove
downvote.");
    return true;
  } catch (error) {
    console.error("Error removing downvote:", error);
    return false;
  }
};
```

Лістинг 3.14 – Обробка асинхронних запитів

```
const fetchProfile = async () => {
      const profile = await getPublicProfile(citizen);
      setUsername(profile.username);
      setAvatarUrl(profile.avatarUrl);
    };
    fetchProfile();
```

Всі запити до серверу виконуються за протоколом HTTPS, що забезпечує захищену передачу даних. Для аутентифікації користувачів та авторизації доступу до захищених ресурсів використовується JWT (JSON Web Tokens), що дозволяє передавати токен автентифікації разом з кожним запитом, що забезпечує безпечний доступ до даних.

Додатково для підвищення безпеки всі дані, що передаються між клієнтом і сервером, шифруються, а також здійснюється перевірка на стороні сервера для запобігання атакам, таким як CSRF або SQL-ін’єкції.

3.2 Серверна частина додатка

3.2.1 Опис архітектури серверної частини

Архітектура серверної частини додатка побудована на основі Django, високопродуктивного веб-фреймворка для мови програмування Python, який надає велику кількість вбудованих інструментів для швидкої розробки веб-додатків. Для цього проекту було використано підхід до розробки, який передбачає використання Django apps для модульної організації коду, що дозволяє легко масштабувати додаток і додавати нові функціональності без значних змін в основній архітектурі.

Проект побудований за принципом "клієнт-сервер" та розділений на декілька основних компонентів, що реалізують різні аспекти функціональності.

Структура проекту базується на Django apps, де кожен додаток (або "app") відповідає за певну частину функціональності. Це дозволяє логічно поділити систему на окремі функціональні блоки, які легко тестувати, підтримувати та масштабувати. У даному випадку проект містить такі основні Django apps:

- citizens: відповідає за функціональність, пов'язану з громадянами, їх реєстрацією, зверненнями та взаємодією з додатком.
- comments: управляє коментарями до звернень та іншими додатковими даними.
- core: надає глобальні утиліти та загальні функції, які використовуються в іншому коді.
- councils: реалізує управління радами або органами, які опрацьовують звернення.
- issues: відповідає за відслідковування та обробку інфраструктурних проблем.
- jwt_auth: надає систему авторизації через JWT (JSON Web Token) для

безпечного доступу до ресурсів.

- reports: обробка і генерація звітів, аналіз даних.

- statistics: реалізація аналітики і статистичних даних для моніторингу роботи системи.

- support: модуль для підтримки користувачів та збору зворотного зв’язку.

- users: управління користувачами, реєстрація, автентифікація та авторизація.

Структура проекту містить важливі компоненти для налаштування середовища та автоматизації процесів:

- docker/: містить файли для налаштування контейнеризації додатка через Docker, що забезпечує ізольоване середовище для розробки та розгортання.

- requirements/: визначає залежності Python для проекту, включаючи базові та виробничі залежності.

- scripts/: містить корисні скрипти для управління сервером та його налаштуванням.

- manage.py: основний файл для запуску команд Django, таких як міграції, запуск серверу тощо.

- middleware.py: налаштування кастомного посередника для обробки запитів та виконання додаткових функцій (наприклад, логування).

Структурно-логічна схема доменної моделі серверної частини (рисунок 3.4) побудована на базовому класі django.db.models.base.Model, який уніфікує персистентність, валідацію та транзакційну цілісність даних. Блок автентифікації охоплює стандартні моделі User, Group і Permission: користувач може належати до кількох груп, а групи агрегують довільні набори дозволів, забезпечуючи гнучке керування ролями.

Паспортні дані ізольовано в моделі Citizen, що має зв’язок “один-до-одного” з User, завдяки чому чутлива інформація відокремлена від базової обліковки і легко розширюється.

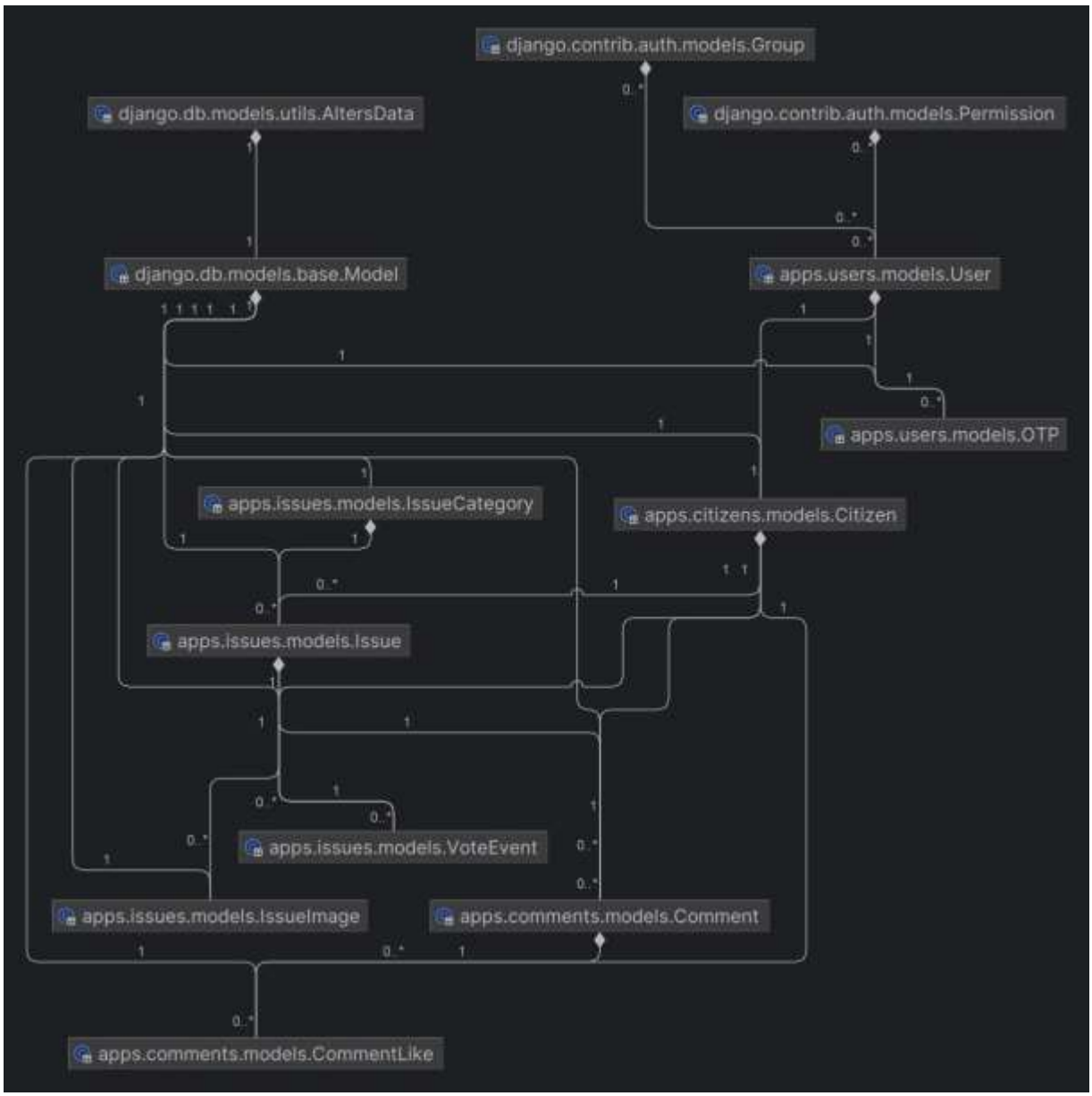


Рисунок 3.4 – Схема доменної моделі серверної частини

Нижній блок діаграми охоплює модулі “issues” та “comments”. Сутність IssueCategory має зв’язок “один-до-багатьох” з Issue, що описує конкретне звернення громадянина. Кожен Issue обов’язково прив’язано до Citizen й може містити необмежену кількість об’єктів IssueImage та Comment. Соціальну взаємодію реалізують VoteEvent та CommentLike: перша фіксує голосування за звернення, друга – реакції на коментарі; обидві моделі мають зв’язки “багато-до-одного” з Citizen і відповідними сутностями, що унеможливлює дублювання голосів і забезпечує коректну аналітику підтримки.

Схема не містить циклічних залежностей, відповідає нормальним формам та спрощує транзакційне керування. Чітке зонування моделі дозволяє безболісно додавати нові події чи атрибути, тоді як успадкування від базового класу Django ORM гарантує дотримання принципів ACID. Діаграма таким чином відображає повну логіку предметної області, показуючи інтеграцію механізмів доступу із системою звернень і соціальною взаємодією.

3.2.2 Обробка запитів та взаємодія з клієнтською частиною

Обробка запитів між серверною та клієнтською частинами додатка є основною складовою архітектури сучасних веб-додатків. Взаємодія здійснюється через API, що дозволяє серверу приймати запити від клієнтської частини, обробляти їх, виконувати необхідні операції з базою даних і надавати відповіді. У рамках цього проекту для обробки запитів на сервері використовуються стандартні методи HTTP: GET, POST, PUT, і DELETE, кожен з яких виконує певні операції, що відповідають за зчитування, створення, оновлення та видалення даних. Цей підрозділ описує, як саме організовано оброблення цих запитів на сервері та взаємодія з клієнтською частиною.

GET-запити використовуються для отримання даних з сервера. Вони є найбільш поширеними запитами, коли клієнт потребує доступу до інформації, наприклад, для перегляду списку звернень або отримання детальної інформації про конкретну проблему або для отримання списку усіх проблем за місцезнаходженням або за отриманням власним даних для відображення у профілі. Сервер обробляє GET-запити, виконуючи відповідні операції на базі даних через Django ORM, що дозволяє ефективно вибирати необхідні записи та передавати їх клієнтській частині у форматі JSON (лістинг 3.15).

Лістинг 3.15 – GET-метод для отримання інформації про конкретну проблему

```
    def get(self, request, issue_id):
        try:
            issue =
Issue.objects.prefetch_related("images").get(id=issue_id)
            serializer = IssueSerializer(issue)
            return Response(serializer.data,
status=status.HTTP_200_OK)
        except Issue.DoesNotExist:
            return Response({"error": "Issue not found."},
status=status.HTTP_404_NOT_FOUND)
```

Запити до серверної частини обробляються через Django REST Framework (DRF), який надає зручні інструменти для створення та обробки RESTful API. Усі дані передаються у форматі JSON, що забезпечує зручність у роботі з ними як на сервері, так і на клієнтській стороні. Сервер забезпечує перевірку вхідних даних через сервісні серіалізатори (наприклад, IssueSerializer), що гарантує, що дані будуть у правильному форматі перед тим, як потраплять до бази даних.

3.2.3 Міграції баз даних

Міграції бази даних є важливим елементом серверної частини додатка, оскільки вони дозволяють автоматизувати процес оновлення схеми бази даних під час змін у моделях даних. У цьому проекті для управління міграціями було використано Django migrations, які є вбудованою частиною Django ORM і надають потужний механізм для створення, зміни та оновлення таблиць бази даних без необхідності вручну змінювати SQL-код.

У Django міграції працюють на основі моделей даних, які визначають структуру таблиць у базі даних. Кожна зміна в моделі (наприклад, додавання нового поля або зміна типу даних) автоматично генерує відповідну міграцію, що забезпечує синхронізацію структури бази даних з моделями.

Переваги використання міграцій:

- автоматизація процесу оновлення;
- сумісність із різними середовищами;
- можливість відкату.

Процес міграції включає кілька етапів:

- кожна зміна в моделях автоматично реєструється через команду python manage.py makemigrations, що генерує файли міграцій, що містять інструкції для створення, зміни або видалення таблиць або їх стовпців;
- після створення міграцій необхідно застосувати їх до бази даних за допомогою команди python manage.py migrate. Ця команда виконує інструкції міграцій, оновлюючи структуру бази даних у відповідності до змін;
- якщо потрібно скасувати зміни, це можна зробити за допомогою команди python manage.py migrate app_name previous_migration_name, що дозволяє відкотити базу даних до попередньої версії.

3.2.4 Логування та моніторинг

Логування та моніторинг є невід'ємними складовими для підтримки стабільності та продуктивності серверної частини додатка. Вони дозволяють відстежувати роботу додатка, оперативно реагувати на помилки та оптимізувати роботу серверу. У даному проекті для логування та моніторингу було реалізовано кілька механізмів, що забезпечують ефективний збір і аналіз даних.

Для збору та аналізу логів у проекті використовується стандартний механізм логування Django, який дозволяє записувати важливу інформацію про запити, помилки та інші події. Логи зберігаються у спеціальних файлах, таких як application.log, що дозволяє відслідковувати роботу додатка в реальному часі.

Основні типи подій, які фіксуються у логах, включають:

- запити до API;

- помилки та виключення;
- інформаційні повідомлення.

Для налаштування рівнів логування у Django можна використовувати конфігурацію в файлі settings.py, де визначаються різні рівні важливості повідомлень (DEBUG, INFO, WARNING, ERROR, CRITICAL). Це дозволяє фільтрувати логи в залежності від рівня важливості, щоб зменшити кількість менш значущих повідомлень у продукційному середовищі.

3.3 База даних

PostgreSQL – це реляційна СУБД з відкритим кодом, яка підтримує стандарт SQL і надає додаткові функції, такі як розширення для роботи з геопросторовими даними (PostGIS) [20], потоки обробки даних, складні запити, транзакції та багатокористувацький доступ. PostgreSQL обирається для проектів, де необхідна надійність, масштабованість і висока продуктивність при роботі з великими обсягами даних [19].

У рамках цього проекту PostgreSQL використовується для зберігання інформації про звернення громадян, інфраструктурні проблеми, статуси звернень, а також додаткові дані, які взаємодіють з клієнтською частиною через API. Для зберігання та маніпулювання цими даними використовуються Django ORM (Object-Relational Mapping), що дозволяє працювати з базою даних через об'єктно-орієнтовану модель, без необхідності писати SQL-код вручну.

Для організації даних у базі даних було створено кілька основних моделей, кожна з яких відповідає за певну частину інформації в додатку. Наприклад, для зберігання звернень було створено модель Issue, яка включає такі поля, як опис проблеми, дата подачі, статус звернення та інші атрибути.

Між таблицями в базі даних встановлені різні зв'язки, такі як:

- один до багатьох;
- багато до багатьох.

Це дозволяє зберігати дані у відносно нормалізованому вигляді та зберігати цілу структуру даних, що легко піддається аналізу та обробці (рисунок 3.5).

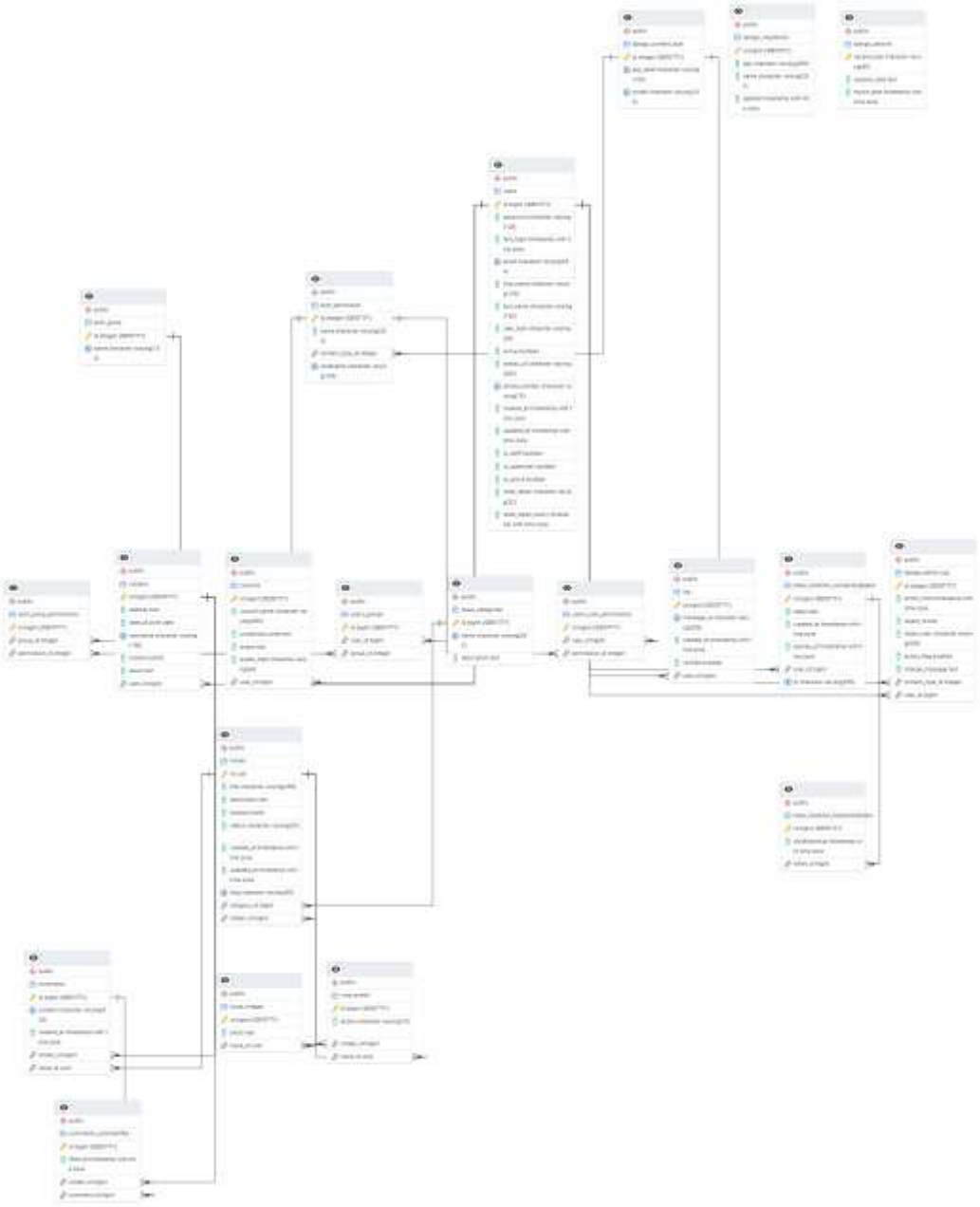


Рисунок 3.5 – Структура проекту клієнтської частини

3.4 Розгортання додатка

Процес розгортання мобільного додатку реалізовано у дві послідовні стадії. На першій здійснюється підготовка середовища: у системі Expo (емулятори й фізичні пристрої) перевіряють коректність залежностей та

працездатність, що підтверджує узгодженість взаємодії клієнта React Native із сервером Django. Друга стадія передбачає контейнеризацію бекенду: сервер-застосунок на базі Django і PostgreSQL з JWT-автентифікацією пакується у Docker-контейнери, завдяки чому досягаються ізоляція залежностей, відтворюваність середовища та усунення версійних конфліктів (лістинг 3.18).

Для керування кількома контейнерами (наприклад, для роботи серверної частини та бази даних PostgreSQL) було використано Docker Compose [11], що дозволяє зручно налаштовувати та запускати ці сервіси одночасно, що значно спрощує процес розгортання (лістинг 3.19).

Лістинг 3.18 – Файл Dockerfile

```
FROM python:3.8
ARG REQUIREMENTS_FILE
WORKDIR /app
EXPOSE 80
ENV PYTHONUNBUFFERED 1
# Install system dependencies
RUN apt-get update && apt-get install -y \
    gdal-bin \
    libgdal-dev \
    libgeos-dev \
    libpq-dev \
    wget \
    && rm -rf /var/lib/apt/lists/*
# Copy the wait-for script
RUN wget -O /wait-for
https://raw.githubusercontent.com/eficode/wait-for/master/wait-
for && \
    chmod +x /wait-for
# Copy requirements and install Python dependencies
COPY ./requirements/ ./requirements
RUN pip install --no-cache-dir -r ./requirements/base.txt
# Copy application code
COPY . ./
# Add SQL script execution during container startup
# TODO devops task.
# COPY fixtures/categories.sql /docker-entrypoint-initdb.d/
CMD ["sh", "/entrypoint-web.sh"]
COPY ./docker/ /
COPY ./requirements/ ./requirements
RUN pip install -r ./requirements/${REQUIREMENTS_FILE}
```

```
COPY . ./
```

Лістинг 3.19 – Файл docker-compose.yml

```
version: '3'
services:
  db:
    image: postgres:14.2
    ports:
      - 5433:5432
    environment:
      POSTGRES_DB: ${DB_NAME}
      POSTGRES_USER: ${DB_USER}
      POSTGRES_PASSWORD: ${DB_PASSWORD}
    volumes:
      - db-data:/data/postgres
  web:
    build:
      context: .
      args:
        REQUIREMENTS_FILE: base.txt
    restart: always
    ports:
      - 8001:8000
    env_file: .env
    command: 'sh -c "./manage.py migrate && ./manage.py
runserver 0.0.0.0:8000"'
    volumes:
      - ./:/app
    depends_on:
      - db
volumes:
  db-data:
```

4 ІНСТРУКЦІЯ КОРИСТУВАЧА

4.1 Реєстрація та автентифікація

Після запуску мобільного додатку користувач потрапляє на екран автентифікації, де він має два варіанти дій:

- Якщо вже має обліковий запис, достатньо ввести електронну адресу та пароль, щоб отримати доступ до основного функціоналу застосунку.
- Якщо облікового запису немає, користувач натискає кнопку “Sign Up”, яка відкриває форму реєстрації.

У формі реєстрації потрібно заповнити такі обов’язкові поля:

- Username – ім’я користувача (мінімум 4 символи);
- Email – має містити символ @, після якого повинна йти хоча б одна літера, крапка та ще одна літера;
- Password – пароль повинен містити мінімум 8 символів, принаймні одну цифру та один спеціальний символ;
- Repeat Password – поле підтвердження пароля, яке має повністю збігатися з попереднім.

До моменту, поки всі поля не будуть заповнені відповідно до валідаційних вимог, кнопка підтвердження реєстрації залишається неактивною. Лише після коректного введення всіх даних вона стає активною, і користувач може завершити реєстрацію (рисунок 4.1).

Також на екрані реєстрації передбачена можливість повернутися до екрана автентифікації, натиснувши кнопку “Sign In”, що дає змогу швидко перемикатися між формами входу та створення акаунта.

Таким чином, процес реєстрації та автентифікації побудований логічно і зручно для користувача. Валідація даних у режимі реального часу, зрозуміла структура форми та чіткі інструкції сприяють швидкому і безпечному доступу до функціоналу додатку.

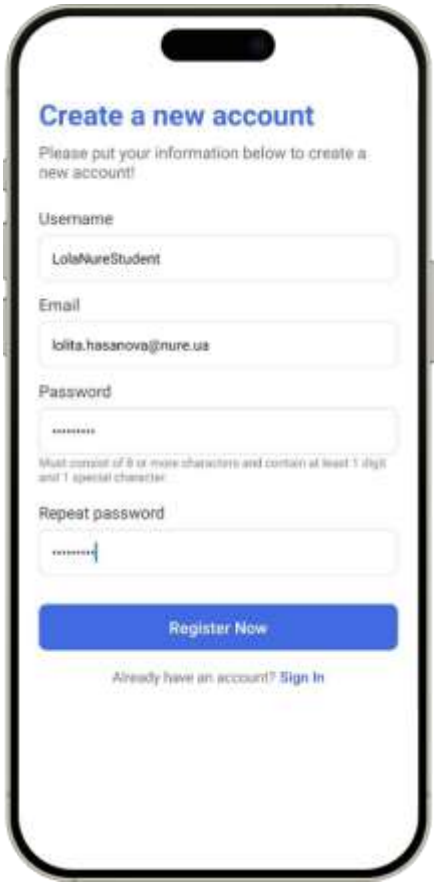
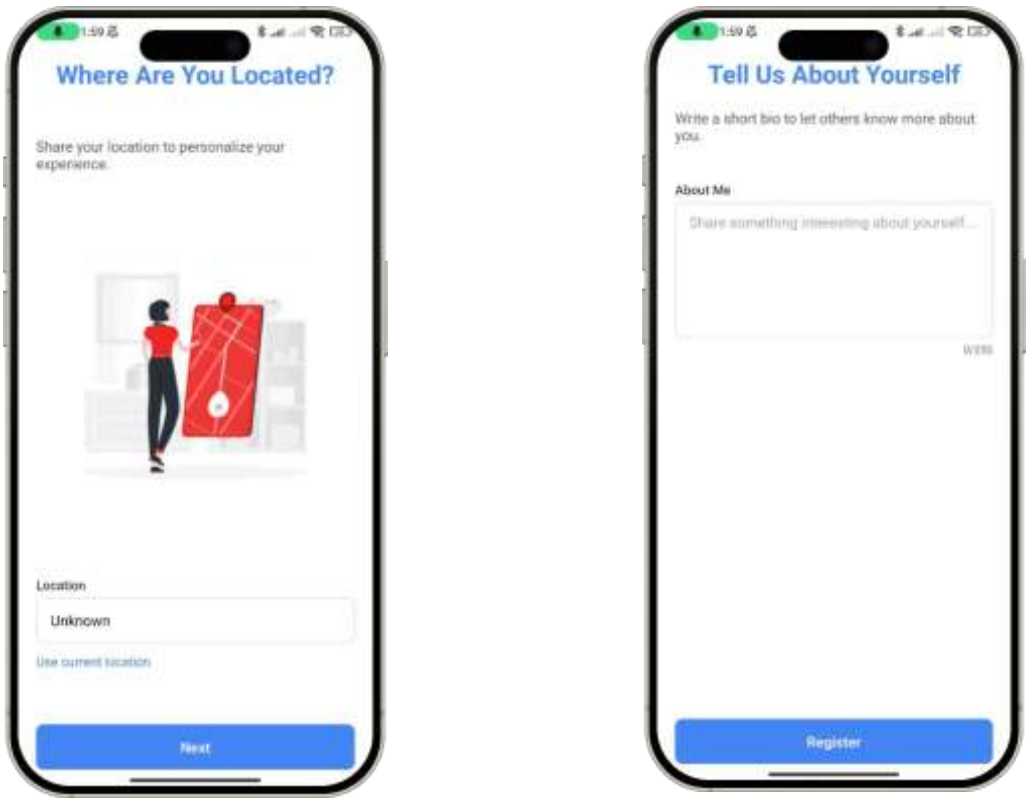


Рисунок 4.1 – Сторінка реєстраії

Після створення облікового запису користувач переходить до наступного етапу – заповнення особистих даних. На першому кроці він вводить ім’я та прізвище. Кожне з цих полів повинно містити щонайменше 2 символи, інакше кнопка “Next” залишається неактивною, що запобігає некоректному введенню.

На наступному кроці відкривається форма введення номеру телефону. Після цього користувач переходить до етапу вибору геолокації – її можна ввести вручну або автоматично визначити за допомогою кнопки “Use current location” (рисунок 4.2 а).

На фінальному етапі користувач заповнює поле “About Me”, у якому надає коротку інформацію про себе (рисунок 4.2 б). Це поле допомагає створити базовий профіль для подальшої ідентифікації користувача в системі.



а) б)

Рисунок 4.2 – Додаткові етапи заповнення профілю:
а) Вибір геолокації; б) Поле “About Me”

Після натискання кнопки “Register” усі введені дані надсилаються на сервер і зберігаються в базі даних. У разі успішної реєстрації користувач автоматично перенаправляється на головну сторінку додатку, де він може одразу почати взаємодію з основним функціоналом системи.

4.2 Головний екран та навігація

Після успішної автентифікації користувач потрапляє на головний екран застосунку (рисунок 4.3), який є стартовою точкою для подальшої взаємодії з функціоналом. Інтерфейс головного екрана організовано відповідно до принципів зручності, доступності та швидкого доступу до ключових функцій.

Основним інструментом перемикання між розділами є панель навігації, розташована внизу екрана. Вона містить чотири основні вкладки:

- “Home” – головна сторінка, де відображаються останні подані звернення, короткі новини або оголошення;

- “Map” – інтерактивна карта, на якій позначені всі зафіксовані інфраструктурні проблеми;

- “Report Issue” – форма для подання нового звернення, що дозволяє прикріпити фото, описати проблему та вказати її розташування;

- “Profile” – особистий кабінет користувача з доступом до власних звернень, налаштувань облікового запису та функції виходу із системи.

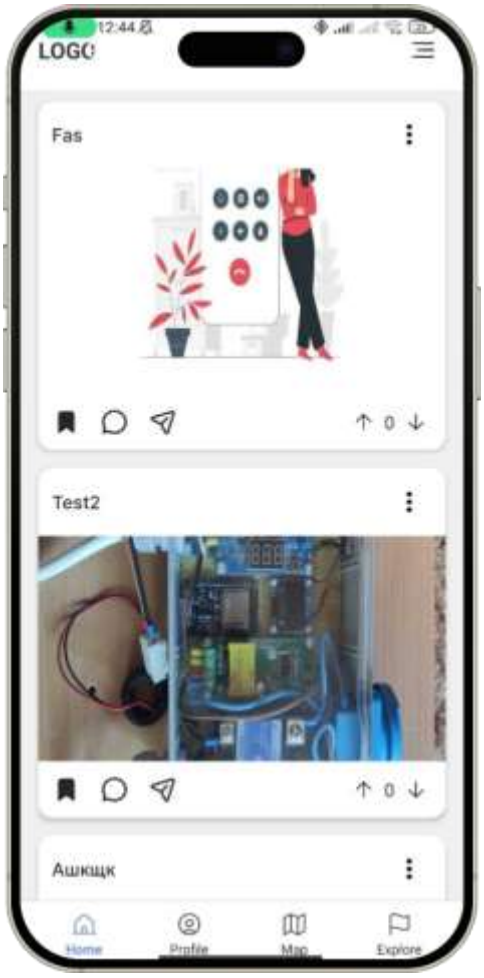


Рисунок 4.3 – Головний екран із навігаційною панеллю

Навігація реалізована за допомогою бібліотеки React Navigation, що забезпечує плавний і стабільний перехід між розділами. Кожна вкладка відповідає окремому стеку екранів, що дозволяє користувачеві повертатися до попередніх дій без втрати контексту. Перехід здійснюється одним

натисканням на відповідну іконку на панелі. Для деяких дій (наприклад, перегляд деталей звернення) передбачено вкладену навігацію зі стрілкою “Назад”. Усі переходи супроводжуються анімаціями згідно з гайдлайнами платформи (Android/iOS), що підвищує зручність сприйняття інтерфейсу.

4.3 Створення поста про проблему

Щоб повідомити про інфраструктурну проблему, користувач має перейти до розділу “Report Issue” через нижню навігаційну панель (іконка з прапорцем). На цьому екрані доступна форма для створення поста з детальним описом проблеми (рисунок 4.4).

Елементи форми:

- назва проблеми. Короткий заголовок, який узагальнює суть проблеми (наприклад: “Провал асфальту”, “Несправне вуличне освітлення”).

- категорія. Випадаючий список з варіантами типових проблем (дороги, каналізація, дерева, сміття тощо). Вибір категорії допомагає відповідальним службам швидше класифікувати звернення.

- опис проблеми. Текстове поле для введення детального опису ситуації. Можна вказати масштаби, ризики для мешканців, тривалість існування проблеми. Максимальна довжина – 500 символів.

- локація. Поле для ручного введення адреси або частини адреси. Воно використовується, якщо геолокацію не вдалося визначити автоматично.

- додати зображення. Кнопка дозволяє прикріпити фотографії, які ілюструють місце, характер і ступінь пошкодження. Після вибору зображення вони відображаються у вигляді мініатюр, які можна видалити натисканням на іконку кошика.

- поділитися проблемою. Після заповнення всіх обов’язкових полів та додавання хоча б одного зображення, кнопка “Share a problem” стає активною. Натискання на неї надсилає звернення до серверної частини, де воно зберігається.

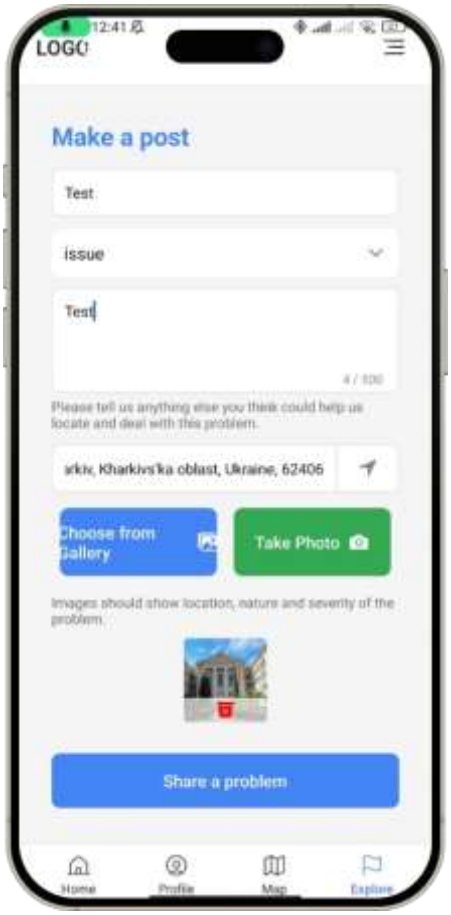


Рисунок 4.4 – Форма створення поста про проблему

4.4 Перегляд детальної інформації про звернення

Для перегляду детальної інформації щодо певного звернення користувач може скористатися одним із двох способів:

- через головний екран;
- через інтерактивну карту.

У обох випадках натискання на картку звернення відкриває нову сторінку, де подано розширену інформацію про проблему (рисунок 4.5).

Основні елементи сторінки:

- автор звернення – ім’я користувача та аватар;
- назва проблеми – короткий заголовок, наприклад “Broken streetlight on main road”;
- категорія – відображається у вигляді кольорового тега, що допомагає

швидко класифікувати проблему (наприклад, road works);

- фото – зображення проблеми, яке ілюструє її характер і масштаб;
- опис – детальний текст про місце, суть і можливі наслідки інциденту;
- інтерактивні елементи – кнопки для вподобання, коментування, сповіщення або поділитися зверненням;
- коментарі користувачів – відображаються нижче, з іменами, аватарами та часом публікації. Користувач може залишати власний коментар або вподобати чужий.

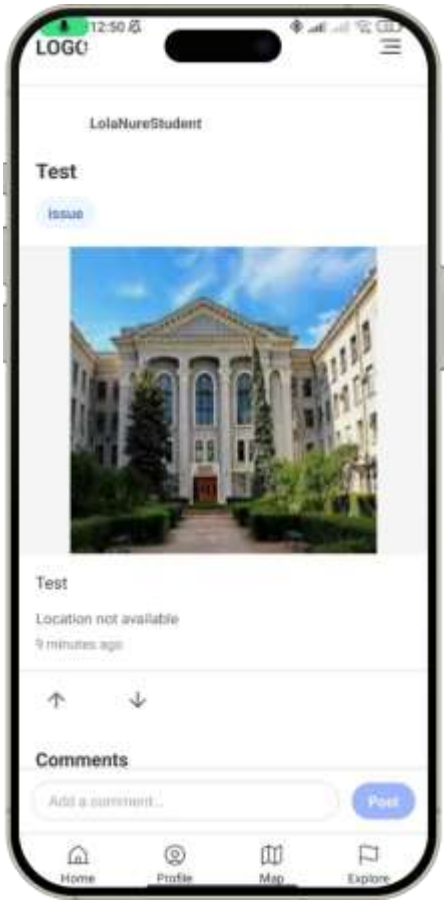


Рисунок 4.5 – Сторінка детального перегляду звернення

Детальний перегляд звернення дозволяє іншим мешканцям уточнити обставини проблеми, долучитися до її обговорення, а міським службам – отримати усю необхідну інформацію для реагування. Така візуальна та текстова деталізація підвищує прозорість і ефективність комунікації в межах міської інфраструктурної взаємодії.

4.5 Інтерактивна карта проблем

Інтерактивна карта є одним з ключових функціональних розділів додатку та доступна через вкладку Map на нижній панелі навігації. Цей розділ дозволяє користувачам переглядати розташування всіх зафіксованих проблем у місті у зручному візуальному форматі, з використанням інтерактивних маркерів (рисунок 4.6).

Основні можливості карти:

- маркування проблем;
- отримання інформації;
- масштабування та навігація;
- автоматичне позиціювання.

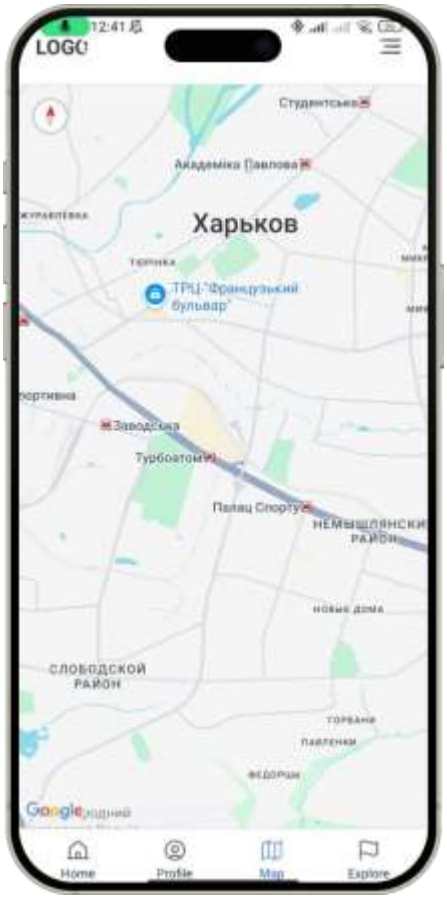


Рисунок 4.6 – Інтерактивна карта

Кожне звернення відображається на карті у вигляді пін-маркера, який позначає географічне розташування проблеми. Колір або форма маркера може відповідати категорії або статусу звернення.

Натискання на пін відкриває коротку інформацію про звернення (назва, категорія, дата подачі). Щоб переглянути повні деталі, потрібно натиснути на картку – це перенаправить користувача на сторінку з розгорнутою інформацією про проблему.

Карту можна збільшувати, зменшувати або переміщувати в межах усієї території міста для зручного перегляду звернень у певному районі.

За допомогою функції “Визначити моє місцезнаходження” користувач може швидко перемістити карту до власної геолокації, щоб перевірити, які проблеми зафіксовані поруч.

Отже, інтерактивна карта дає змогу візуально оцінити масштаби та розподіл інфраструктурних проблем у місті, швидко знайти звернення за місцезнаходження та підвищити прозорість даних про роботу міських служб.

4.6 Налаштування та профіль користувача

Розділ “Profile” дозволяє користувачеві переглядати та змінювати свої персональні дані (рисунок 4.7). Для доступу до профілю необхідно натиснути на відповідну іконку в нижній навігаційній панелі (іконка з аватаром).

На сторінці профілю відображається така інформація:

- фото профілю – користувач може завантажити або змінити зображення, натиснувши на іконку камери;
- “Full Name” – повне ім’я, яке можна змінити;
- “Email” – електронна адреса, яка є унікальним ідентифікатором облікового запису;
- “Location” – місце проживання або будь-яка інша довільна геолокація;
- “Bio” – коротка інформація про себе, до 250 символів.

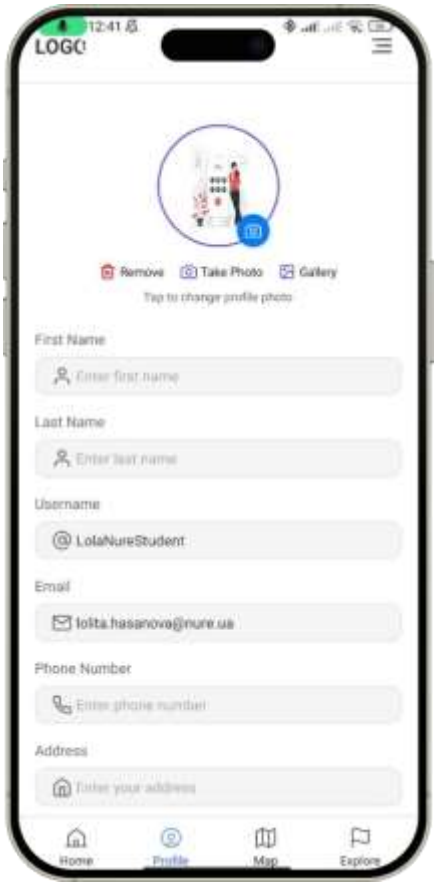


Рисунок 4.7 – Екран редагування профілю

Щоб оновити особисту інформацію, достатньо відредагувати відповідні поля та натиснути кнопку **“**Update Profile**”**. Зміни зберігаються автоматично після успішної перевірки валідності введених даних.

У нижній частині екрана доступна опція **“**Delete account**”**, яка дозволяє повністю видалити обліковий запис користувача зі всіма пов’язаними даними. Перед видаленням система може запропонувати підтвердження для уникнення випадкового видалення.

ВИСНОВКИ

У результаті виконання цієї кваліфікаційної роботи було розроблено ефективний мобільний додаток для фіксації та повідомлення про інфраструктурні проблеми міста, що значно покращує взаємодію громадян з органами місцевого самоврядування. Метою проекту було створення зручного і доступного інструменту для громадян, що дозволяє оперативно повідомляти про різноманітні проблеми міської інфраструктури, такі як пошкоджене дорожнє покриття, несправне освітлення, аварійні дерева або засмічені території.

Для розробки мобільної частини додатка було обрано React Native, що дозволяє створити крос-платформенний додаток для iOS та Android, скорочуючи час і ресурси на розробку, при збереженні високої продуктивності та нативного вигляду додатка. Серверна частина була реалізована з використанням Django, що забезпечує безпеку, масштабованість і ефективність обробки запитів, а також використовує PostgreSQL для зберігання структурованих даних і PostGIS для обробки геопросторової інформації. Це дозволяє точно фіксувати геолокацію інфраструктурних проблем і ефективно їх відображати на карті.

Особливістю проекту є наявність BackOffice вебсайту, що призначений для органів управління та міських служб. Вебсайт дозволяє адміністраторам додатка переглядати подані звернення, оновлювати їх статуси, аналізувати статистику і забезпечувати оперативне реагування на інфраструктурні проблеми. Це важливий інструмент для підвищення ефективності управління міськими ресурсами і для покращення зворотного зв'язку між громадянами та місцевою владою.

У розробці інтерфейсу для мобільного додатка було використано принципи UI/UX дизайну, що забезпечили інтуїтивно зрозуміле управління та доступність для різних груп користувачів. Важливими аспектами були

простота навігації, адаптивність інтерфейсу до різних розмірів екранів, а також увага до доступності для користувачів з обмеженими можливостями зору чи слабким досвідом роботи з мобільними додатками.

Використано Docker для контейнеризації серверної частини, що забезпечило стабільність і гнучкість під час розгортання додатка на різних середовищах. Використання Docker Compose дозволило спростити налаштування серверного середовища та інтеграцію з базою даних, що також знижує ризики помилок під час розгортання.

У ході тестування додатка була проведена перевірка на різних пристроях і платформах, що дозволило виявити та усунути потенційні проблеми з сумісністю та продуктивністю. Під час тестування особливу увагу було приділено правильному відображенню мультимедійних файлів і коректній обробці геолокаційних даних.

Таким чином, цей проект успішно поєднує функціональність для громадян та органи місцевого управління в єдиній системі. Мобільний додаток дозволяє громадянам оперативно повідомляти про проблеми інфраструктури, а веб-платформа BackOffice надає органам місцевої влади зручний інструмент для ефективного управління інфраструктурними ресурсами. Подальші етапи розвитку проекту можуть включати розширення функціоналу, інтеграцію з іншими міськими службами та використання передових аналітичних інструментів для прогнозування інфраструктурних потреб.

Завдяки реалізованим технологіям та архітектурним рішенням, додаток готовий до масштабування та подальшого вдосконалення, що робить його важливим інструментом для покращення управління міськими проблемами та ефективності взаємодії громадян з органами місцевого самоврядування.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Асоціація міст України. Дослідження вуличного освітлення в населених пунктах. URL: https://auc.org.ua/sites/default/files/street_lighting_study_2024.pdf (дата звернення: 01.06.2025)

2. Державна екологічна інспекція України. Щорічна доповідь про стан довкілля та кількість виявлених стихійних сміттєзвалищ. URL: https://dei.gov.ua/reports/environment_state_illegal_dumps_2024.pdf (дата звернення: 06.06.2025)

3. Державне агентство автомобільних доріг України. Звіт про стан експлуатаційного утримання міських вулиць. URL: https://ukravtodor.gov.ua/files/report_city_streets_2024.pdf (дата звернення: 04.06.2025)

4. Київська міська державна адміністрація. Аналітична довідка “Аварійні ямкові ремонти вуличної мережі”. URL: https://kyivcity.gov.ua/data/analytics_pothole_repairs_2023.pdf (дата звернення: 03.06.2025)

5. Київський міжнародний інститут соціології. Довіра населення України до органів влади та інституцій: результати всеукраїнського опитування, травень 2024 р. Київ : КМІС, 2024.

6. Міністерство цифрової трансформації України. Прес-реліз “Кількість користувачів застосунку “Дія” перевищила 22,5 млн”. URL: https://thedigital.gov.ua/news/225m-dia-users-2025 (дата звернення: 01.06.2025)

7. Міністерство розвитку громад, територій та інфраструктури України. Статистичний бюлетень “Комунальні мережі та їх технічний стан”, 2023 р. URL: https://minregion.gov.ua/upload/documents/comms_infra_bulletin_2023.pdf (дата звернення: 01.06.2025)

8. УНН. “Українці зможуть скаржитися на проблеми зі зв’язком через “Дію”“. URL: https://www.unn.com.ua/news/2024/12/17/dia-mobile-problems-complaints (дата звернення: 05.06.2025)

9. У “Дії” з’явиться AI-асистент для обробки скарг громадян. URL: https://financenews.com.ua/3099-u-dii-z-iavytsia-ai-asystent-dlia-obrobky-skarh-hromadian/ (дата звернення: 30.05.2025)

10. Як поскаржитися на поганий чи відсутній мобільний зв’язок через “Дію”. URL: https://hromadske.radio/news/2025/01/31/yak-poskarzhytysia-na-pohanyy-zv-iazok-abo-yoho-vidsutnist-cherez-diiu-instruktsiia (дата звернення: 03.06.2025)

11. Docker. Docker Compose Documentation. URL: https://docs.docker.com/compose (дата звернення: 04.06.2025)

12. Docker. Docker Engine & CLI Documentation. URL: https://docs.docker.com/engine (дата звернення: 05.06.2025)

13. Django REST framework. Official Documentation. URL: https://www.django-rest-framework.org (дата звернення: 03.06.2025)

14. Django Software Foundation. Django 4.x Documentation. URL: https://docs.djangoproject.com/en/4.2/ (дата звернення: 01.06.2025)

15. IETF. RFC 7519: JSON Web Token (JWT). URL: https://datatracker.ietf.org/doc/html/rfc7519 (дата звернення: 02.06.2025).

16. L. Hasanova, N. Axak Development of a mobile application for recording and reporting urban infrastructure issues. Proceedings of the 1th International Scientific and Practical Conference ”Modern Information Technologies and Artificial Intelligence Systems MIT@AIS 2025”, Part 2, May 19-22, 2025 Kharkiv - Yaremche, Ukraine P.65 – 66. URL: https://ist-conf-nure.com.ua/mitais/index.html (дата звернення: 03.06.2025)

17. MDN Web Docs. Fetch API – Using Fetch. URL: https://developer.mozilla.org/docs/Web/API/Fetch_API (дата звернення: 04.06.2025)

18. Meta. React – Context API Documentation. URL:

https://react.dev/reference/react/useContext (дата звернення: 06.06.2025)

19. PostgreSQL Global Development Group. PostgreSQL 15 Documentation. URL: https://www.postgresql.org/docs/15 (дата звернення: 01.06.2025)

20. PostGIS Project. PostGIS 3 Documentation. URL: https://postgis.net/documentation (дата звернення: 03.06.2025)

21. React Native. Official Documentation – Getting Started. URL: https://reactnative.dev/docs/getting-started (дата звернення: 04.06.2025)

22. React Native Image Picker. GitHub Repository. URL: https://github.com/react-native-image-picker/react-native-image-picker (дата звернення: 05.06.2025)

23. React Native Maps. GitHub Repository. URL: https://github.com/react-native-maps/react-native-maps (дата звернення: 02.06.2025)

24. React Navigation. Getting Started Guide. URL: https://reactnavigation.org/docs/getting-started (дата звернення: 01.06.2025)

25. StatCounter Global Stats. Mobile Operating System Market Share in Ukraine, May 2025. URL: https://gs.statcounter.com/os-market-share/mobile/ukraine (дата звернення: 03.06.2025)