Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)
Кафедра Інформатики
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

РОЗРОБЛЕННЯ КРОСПЛАТФОРМЕННОГО ЗАСТОСУНКУ ДЛЯ КРАУДСОРСИНГУ ПРОБЛЕМ ДОСТУПНОСТІ У МІСТІ

(тема)

Виконав:
здобувач 4 року навчання,
групи ІТІНФ-22-3
Зіма В. О.
(прізвище, ініціали)

Спеціальність 122 Комп'ютерні науки
(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

Керівник доц. Кобилін О. А.
(посада, прізвище, ініціали)

Допускається до захисту

Завідувач кафедри інформатики _____ Кобилін О. А.
(підпис) (прізвище, ініціали)

2026 р.

Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту

Кафедра Інформатики

Рівень вищої освіти перший (бакалаврський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Зімі Володимиру Олександровичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розроблення кросплатформенного застосунку для краудсорсингу проблем доступності у місті

затверджена наказом університету від 26 травня 2026 року № 435Ст
2. Термін подання здобувачем роботи до екзаменаційної комісії 19 червня 2026 р.
3. Вихідні дані до роботи науково-методична та науково-технічна література, матеріали наукових конференцій, дані інтернет-мережі, фреймворк Flutter, хмарна платформа Firebase, Google Maps SDK, набір даних Sidewalk Accessibility v3.0.

4. Перелік питань, що потрібно опрацювати в роботі
1. Аналіз стану доступності міського середовища в Україні та огляд існуючих застосунків картування доступності.
2. Проєктування архітектури системи, моделі даних та моделей валідації краудсорсингових даних.
3. Програмна реалізація застосунку, експериментальна перевірка моделі валідації фотоматеріалів та тестування на платформах Android і Web.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) актуальність проблеми доступності, постановка задачі, архітектура системи, UML-діаграми, модель даних Firestore, схема HITL-пайплайну, результати тестування моделі, скріншоти застосунку.

6. Консультанти розділів роботи (п.6 включається до завдання 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

№ з/п	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	13.04.2026	
2	Аналіз завдання, підбір літератури	14.04.26-16.04.26	
3	Аналіз літератури з проблеми, що вирішується	17.04.26-21.04.26	
4	Аналіз існуючих застосунків картування доступності	22.04.26-25.04.26	
5	Проєктування архітектури системи та моделей даних	26.04.26-07.05.26	
6	Програмна реалізація застосунку	08.05.26-25.05.26	
7	Експериментальна перевірка та аналіз результатів	26.05.26-04.06.26	
8	Оформлення пояснювальної записки	05.06.26-10.06.26	
9	Перевірка на нормоконтроль	11.06.26-12.06.26	
10	Перевірка на плагіат	13.06.26-14.06.26	
11	Рецензування	15.06.26-20.06.26	
12	Підготовка презентації та доповіді	21.06.26-27.06.26	
13	Занесення роботи в електронний архів	30.06.26	
14	Попередній захист кваліфікаційної роботи	30.06.26	

Дата видачі завдання 13 квітня 2026 р.

Здобувач _____
(підпис)

Керівник роботи _____ доц. Кобилін О. А.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ/ABSTRACT

Пояснювальна записка до кваліфікаційної роботи: 82 с., 9 табл., 19 рис., 45 джерел.

ДОСТУПНІСТЬ, КРАУДСОРСИНГ, КРОСПЛАТФОРМНИЙ ЗАСТОСУНОК, МАЛОМОБІЛЬНІ ГРУПИ НАСЕЛЕННЯ, GOOGLE MAPS.

Об’єктом роботи є кросплатформний мобільний застосунок для краудсорсингового збору даних про доступність міської інфраструктури.

Метою роботи є розроблення застосунку для додавання геоприв’язаних міток на карту міста з описами об’єктів доступності для маломобільних груп населення.

Під час роботи використано: методи краудсорсингу та VGI, розробку на Flutter/Dart, хмарні сервіси Firebase, інтеграцію Google Maps Flutter SDK.

Практична цінність полягає у створенні інструменту для збору актуальних даних про доступність міського середовища для маломобільних осіб та міських органів.

Розроблено застосунок із картографічним шаром геоприв’язаних міток, механізмом їх додавання з фото та описами й переглядом детальної інформації.

Область застосування: міське планування, соціальна інфраструктура, цивільні GIS-платформи, програми відновлення міст.

ACCESSIBILITY, CROWDSOURCING, CROSS-PLATFORM APPLICATION, PEOPLE WITH REDUCED MOBILITY, GOOGLE MAPS.

The objective of this work is a cross-platform mobile application for crowdsourced collection of urban accessibility data.

The goal of this work is to develop an application for adding geolocated markers to a city map with descriptions of accessibility features for people with reduced mobility.

The following were used: crowdsourcing and VGI methods, Flutter/Dart development, Firebase cloud services, Google Maps Flutter SDK integration.

The practical value lies in creating a tool for collaborative collection of urban accessibility data for use by people with reduced mobility and city authorities.

A cross-platform application has been developed with a map layer of geolocated markers, a mechanism for adding them with photos and descriptions, and detailed information display.

Applications: urban planning, social infrastructure, civic GIS platforms, city recovery programs.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ВГІ – волонтерська географічна інформація

ДБН – державні будівельні норми

ГІС – геоінформаційна система

API – Application Programming Interface (інтерфейс програмного застосунку)

Dart – мова програмування, що використовується у середовищі Flutter

Firebase – хмарна платформа від Google для зберігання даних та автентифікації

Firestore – хмарна база даних у реальному часі від Google Firebase

Flutter – кросплатформний фреймворк розроблення мобільних і вебзастосунків

GIS – Geographic Information System (геоінформаційна система)

GPS – Global Positioning System (глобальна система позиціонування)

SDK – Software Development Kit (комплект засобів розроблення програмного забезпечення)

UI – User Interface (інтерфейс користувача)

UX – User Experience (досвід взаємодії користувача із застосунком)

VGI – Volunteered Geographic Information (волонтерська географічна інформація)

ВСТУП

Забезпечення безбар’єрного міського середовища є однією з ключових умов повноцінної участі людей з інвалідністю у суспільному житті. В Україні ця проблема набула особливої гостроти внаслідок повномасштабного збройного вторгнення Російської Федерації, розпоченого у лютому 2022 року: кількість осіб з набутою інвалідністю та обмеженнями рухливості суттєво зросла серед як ветеранів, так і цивільного населення. За оцінками міжнародних організацій, кількість людей, які потребують допоміжних засобів пересування, може сягати кількох сотень тисяч осіб, що робить питання доступності міської інфраструктури пріоритетним у контексті повоєнної відбудови.

Незважаючи на наявність нормативної бази, зокрема ДБН В.2.2-17:2006 та відповідних законів України, фактичний стан більшості об’єктів інфраструктури не відповідає сучасним вимогам доступності. Більшість будівель і тротуарів, зведених у радянський період, позбавлені пандусів, занижених бордюрів та рівних покриттів, необхідних для безпечного пересування на кріслі колісному. Традиційні методи інвентаризації доступності є ресурсоємними і не дають змоги забезпечити актуальність даних у масштабі цілого міста.

Актуальність роботи визначається потребою у цифровому інструменті, який дозволяв би систематично збирати, верифікувати та відображати інформацію про стан доступності міського середовища із залученням широкого кола користувачів. Підхід краудсорсингу та волонтерської географічної інформації (VGI) забезпечує масштабованість і актуальність даних при низькій вартості збору порівняно із централізованими методами. Водночас наявні застосунки або обмежені оцінкою окремих закладів (Wheelmaps, AXS Map), або не дозволяють фіксувати довільні елементи вуличної інфраструктури (Google Maps Accessible Places), що обумовлює необхідність розроблення нового рішення.

1 АНАЛІЗ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Доступність міського середовища в Україні

Доступність міського середовища є однією з ключових умов повноцінної участі людей з інвалідністю у суспільному житті. Під доступністю розуміють сукупність характеристик будівель, споруд, транспортної інфраструктури та відкритих міських просторів, що забезпечують їх безперешкодне використання маломобільними групами населення, зокрема користувачами крісел колісних, людьми з порушеннями зору та слуху, а також особами похилого віку та батьками з дитячими візочками [1].

Нормативну основу забезпечення доступності в Україні становлять державні будівельні норми ДБН В.2.2-17:2006 «Доступність будинків і споруд для маломобільних груп населення», які регламентують параметри пандусів, тактильних покриттів, ширини проходів та висоти порогів. Крім того, Закон України «Про реабілітацію у сфері охорони здоров’я» та Закон «Про основи соціальної захищеності осіб з інвалідністю в Україні» закріплюють право на безбар’єрне середовище на законодавчому рівні. На виконання цих норм органи місцевого самоврядування зобов’язані забезпечувати відповідність нової та реконструйованої інфраструктури встановленим вимогам.

Попри наявність нормативної бази, фактичний стан міського середовища в Україні залишається незадовільним. За даними Міністерства соціальної політики України, станом на початок 2022 року в країні нараховувалося понад 2,7 млн осіб з інвалідністю, з яких значна частка має обмеження рухливості [2]. При цьому більшість об’єктів інфраструктури, зведених у радянський період, не відповідають сучасним вимогам доступності: відсутні пандуси на вході до будівель, бордюри не мають

знижень для в’їзду крісла колісного, тротуарне покриття є нерівним або пошкодженим.

Повномасштабне вторгнення Російської Федерації в Україну, розпочате у лютому 2022 року, докорінно змінило масштаб проблеми. Унаслідок бойових дій кількість людей, які отримали поранення та набули інвалідність, суттєво зросла.

За оцінками ООН та Міжнародного комітету Червоного Хреста, внаслідок збройного конфлікту в Україні кількість людей з набутою інвалідністю може сягнути кількох сотень тисяч осіб, що робить проблему доступності міського середовища особливо гострою в контексті повоєнної відбудови [3]. Серед ветеранів та цивільних осіб, які отримали ампутації або травми опорно-рухового апарату, найбільш поширеною є потреба у пересуванні з використанням крісла колісного, що висуває підвищені вимоги до стану тротуарів, перехресть, входів до будівель та громадського транспорту.

Відповідно до Національної програми відбудови України, прийнятої у 2023 році, принцип «будуємо краще, ніж було» передбачає обов’язкове дотримання стандартів доступності при реконструкції об’єктів, зруйнованих або пошкоджених унаслідок воєнних дій [4]. Проте ефективна реалізація цього принципу потребує не лише нормативного регулювання, а й інструментів моніторингу фактичного стану інфраструктури в режимі реального часу. Традиційні методи інвентаризації, що передбачають виїзд спеціалістів та складання актів обстеження, є ресурсоємними і не дозволяють забезпечити актуальність даних у масштабі міста.

Міжнародно-правову основу забезпечення доступності становить Конвенція ООН про права осіб з інвалідністю (CRPD), ратифікована Україною у 2009 році. Стаття 9 Конвенції зобов’язує держави-учасниці вживати належних заходів для забезпечення доступу осіб з інвалідністю нарівні з іншими до фізичного оточення, транспорту, інформації та зв’язку, а також до об’єктів і послуг, відкритих або таких, що надаються населенню.

Зокрема, Конвенція вимагає виявлення та усунення перешкод і бар'єрів доступності у будівлях, на дорогах, у транспорті та в інших об'єктах інфраструктури. Ратифікація CRPD фактично підтверджує міжнародні зобов'язання України щодо системного моніторингу та усунення бар'єрів у міському середовищі, що потребує відповідних цифрових інструментів збору та аналізу даних.

Таким чином, існує об'єктивна потреба у розробці цифрових інструментів, які дозволяли б систематично збирати, верифікувати та візуалізувати інформацію про стан доступності міського середовища із залученням широкого кола користувачів. Саме цей підхід, відомий як краудсорсинг географічних даних, розглядається у наступному підрозділі.

1.2 Краудсорсинг та VGI як метод збору даних

Краудсорсинг (від англ. crowdsourcing – залучення «натовпу» до виконання завдань) є підходом до збору даних, обробки інформації або вирішення задач шляхом розподілу роботи між великою кількістю учасників через цифрові платформи. Термін введено у науковий обіг Джеффом Гауі у 2006 році для позначення моделі, за якої функції, традиційно виконувані штатними фахівцями, передаються невизначеному колу добровольців [5]. У контексті збору географічних даних краудсорсинг набув окремого визначення – волонтерська географічна інформація (VGI, Volunteered Geographic Information), запропонованого Майклом Гудчайлдом у 2007 році [6].

VGI описує процес, за якого звичайні користувачі, не обов'язково підготовлені як картографи чи геодезисти, самостійно створюють, редагують та поширюють просторові дані за допомогою мобільних пристроїв та вебплатформ. Найбільш відомим прикладом VGI є проєкт OpenStreetMap, що станом на 2024 рік нараховує понад 10 млн зареєстрованих учасників і

містить детальні картографічні дані для більшості країн світу [7]. Іншими прикладами є навігаційний сервіс Waze, де водії в режимі реального часу повідомляють про затори та дорожні події, та платформа iNaturalist, на якій волонтери фіксують спостереження за дикою природою.

Ключовою перевагою краудсорсингового підходу є здатність охоплювати території та явища, систематичний моніторинг яких є надто витратним для централізованих організацій.

На відміну від традиційних методів картографування, краудсорсинг забезпечує високу актуальність даних, оскільки учасники фіксують зміни у реальному часі безпосередньо з місця події. Для задач моніторингу міської доступності це є принциповою перевагою: стан пандуса, бордюрного зниження або тротуарного покриття може змінитися після ремонту, пошкодження чи сезонного погіршення, і лише систематичне залучення місцевих мешканців дозволяє підтримувати базу даних у актуальному стані.

Разом з тим краудсорсингові дані мають характерні обмеження, що необхідно враховувати при проєктуванні системи. По-перше, нерівномірність просторового покриття: активність учасників зазвичай вища у центральних районах міста та знижується на периферії, що призводить до прогалин у даних [8, 10]. По-друге, варіативність якості: різні користувачі можуть по-різному інтерпретувати критерії доступності, припускатися помилок при геолокації або завантажувати нерелевантні фотоматеріали. По-третє, проблема вандалізму та навмисного спотворення даних, характерна для відкритих платформ.

Для подолання цих обмежень у сучасних краудсорсингових геоінформаційних системах застосовують механізми валідації: перехресна перевірка повідомлень від кількох незалежних користувачів, автоматичний аналіз достовірності на основі історії активності учасника, а також алгоритмічна дедуплікація близько розташованих міток [9, 11]. Розробка подібних механізмів є складовою частиною цієї роботи і детально розглядається у розділі 2.

Окремим аспектом ефективності краудсорсингових систем є мотивація учасників до регулярного внеску даних. Дослідження у сфері VGI виокремлюють кілька основних мотиваційних факторів: альтруїстичне бажання зробити місто зручнішим для інших, особиста потреба у актуальній інформації про доступність маршрутів, а також елементи гейміфікації – рейтинги, досягнення та візуальний зворотній зв’язок про власний внесок на карті. Для цільової аудиторії розроблюваного застосунку, що включає як самих маломобільних користувачів, так і волонтерів та родичів людей з інвалідністю, особиста зацікавленість у якості даних є природним мотиватором, що відрізняє цю платформу від загальних картографічних проєктів.

Для порівняння масштабу проблеми доцільно розглянути альтернативні підходи до збору даних про доступність. Традиційна інвентаризація інфраструктури силами муніципальних служб передбачає виїзд спеціалістів, складання актів обстеження та внесення даних до реєстрів. За оцінками, повне обстеження пішохідної інфраструктури міста з населенням 500 тисяч осіб потребує від кількох місяців до року роботи профільних фахівців та відповідного фінансування. При цьому отримані дані швидко застарівають: середній термін актуальності результатів одноразової інвентаризації не перевищує двох-трьох років через природне зношення покриття, сезонні пошкодження та будівельні роботи. Краудсорсинговий підхід усуває цей недолік, забезпечуючи безперервне оновлення даних без додаткових витрат на організацію польових робіт.

Виходячи з цього, краудсорсинг та VGI є обґрунтованою методологічною основою для створення системи моніторингу міської доступності, що поєднує масштабованість, актуальність та низьку вартість збору даних порівняно з централізованими підходами.

1.3 Огляд існуючих застосунків

На сьогоднішній день існує ряд мобільних та вебзастосунків, що частково вирішують задачу інформування користувачів про доступність міського середовища. Аналіз наявних рішень дає можливість виявити їх спільні обмеження та обґрунтувати необхідність розробки нового інструменту.

Wheelmaps – веб та мобільний застосунок, розроблений німецькою організацією Sozialhelden e.V. [12], що дає змогу користувачам позначати доступність громадських місць для людей на кріслах колісних за триступеневою шкалою: повністю доступно, частково доступно, недоступно. База даних застосунку побудована на основі OpenStreetMap, що забезпечує широке географічне охоплення. Проте Wheelmaps орієнтований виключно на оцінку закладів як цілісних об’єктів і не дозволяє фіксувати конкретні елементи інфраструктури – окремий пандус, знижений бордюр або перешкоду на тротуарі. Крім того, відсутній механізм завантаження фотоматеріалів безпосередньо до мітки, що знижує інформативність повідомлень.

AXS Map [13] – американська платформа краудсорсингового картування доступності закладів, що функціонує за принципом, подібним до Wheelmaps. Користувачі оцінюють доступність ресторанів, магазинів, готелів та інших об’єктів за кількома критеріями, включаючи наявність пандуса при вході, доступний туалет та достатню ширину проходів. Перевагою AXS Map є детальніша структура опитувальника порівняно з Wheelmaps, однак застосунок також обмежений рівнем окремого закладу і не охоплює стан вуличної інфраструктури між об’єктами.

Google Maps у 2017 році запровадив функцію «Accessible places» [14], що дозволяє відображати інформацію про доступність закладів безпосередньо у картці об’єкта. Дані формуються на основі відповідей власників закладів та відгуків користувачів через механізм Google Local

Guides. Перевагою є масштаб охоплення та інтеграція з існуючою навігацією, проте підхід має принципове обмеження: інформація прив’язана виключно до зареєстрованих у Google закладів і не може бути застосована до довільних точок міського простору, таких як перехрестя, тротуарні ділянки або підземні переходи.

Окрему категорію становлять застосунки маршрутизації для маломобільних користувачів, зокрема Accessnow та Route4U. Вони пропонують побудову маршрутів з урахуванням доступності, однак спираються на заздалегідь зібрані або імпортовані набори даних, а не на динамічний краудсорсинговий шар. Це робить їх залежними від актуальності вихідної бази, яка оновлюється нерегулярно.

Порівняльний аналіз розглянутих рішень наведено у таблиці 1.1.

Таблиця 1.1 – Характеристики та параметри детекторів

Застосунок	**Платформа**	**Тип об’єктів**	**Фото до мітки**	**Довільні точки**	**Краудсорсинг**
Wheelmaps	Web, iOS, Android	Заклади	Ні	Ні	Так
AXS Map	Web, iOS, Android	Заклади	Так	Ні	Так
Google Maps	Web, iOS, Android	Заклади	Так	Ні	Частково
Accessnow	iOS, Android	Заклади	Так	Ні	Так
Route4U	iOS, Android	Маршрути	Ні	Ні	Ні
A11y Map	**iOS, Android, Web**	**Довільні точки**	**Так**	**Так**	**Так**

Аналіз наведених рішень дозволяє виявити спільну архітектурну обмеженість: усі розглянуті застосунки використовують модель «заклад як одиниця даних», де об’єктом оцінки є зареєстрована точка інтересу з фіксованою адресою. Така модель принципово не охоплює простір між закладами – тротуари, перехрестя, підземні переходи, зупинки громадського транспорту та інші елементи пішохідної інфраструктури, що є першочерговими бар’єрами для користувача крісла колісного. Людина на кріслі колісному стикається з проблемою доступності не лише на вході до закладу, а передусім на шляху до нього: один непрохідний бордюр або зруйнована ділянка тротуару може унеможливити весь маршрут незалежно від рівня доступності кінцевої точки.

Додатковим обмеженням існуючих рішень є залежність від зовнішніх баз даних закладів. Wheelmaps та AXS Map можуть відображати лише ті об’єкти, які вже присутні в OpenStreetMap або Google Places відповідно. Це означає, що новий пандус, встановлений у рамках програми відбудови, або тимчасова перешкода через будівельні роботи не можуть бути задокументовані до оновлення базової бази даних, що може тривати тижні або місяці. Розроблюваний застосунок усуває цю залежність, дозволяючи розміщувати мітки у довільних координатах незалежно від наявності відповідного об’єкта у зовнішніх реєстрах.

Як видно з таблиці, жоден з розглянутих застосунків не забезпечує можливості розміщення геоміток у довільних точках міського простору з прикріпленням фотоматеріалів та формуванням відкритого краудсорсингового шару поверх базової карти. Саме ця функціональна прогалина визначає актуальність розроблюваного застосунку.

1.4 Обґрунтування необхідності розробки нового рішення

Проведений аналіз існуючих застосунків виявив системну обмеженість наявних рішень: усі вони орієнтовані на оцінку доступності конкретних закладів як цілісних об’єктів і не надають можливості фіксувати стан інфраструктури між ними. Водночас для користувача крісла колісного саме вуличне середовище є першочерговим бар’єром: відсутність бордюрного зниження на перехресті, зруйноване тротуарне покриття або припаркований на пішохідному переході автомобіль можуть зробити маршрут непрохідним незалежно від рівня доступності кінцевого пункту призначення.

Окремою проблемою є статичність існуючих баз даних. Більшість платформ формують дані одноразово або оновлюють їх нерегулярно, що не відповідає динамічному характеру міського середовища. Стан тротуару може погіршитися після зими, пандус може бути демонтовано під час ремонту фасаду, а нова знижена бордюрна плита з’явитися після реконструкції перехрестя. Лише безперервний краудсорсинговий збір даних від мешканців міста здатний забезпечити актуальність такої інформації.

В умовах повоєнної відбудови України зазначені проблеми набувають додаткової гостроти. Масштабне руйнування інфраструктури у постраждалих містах та одночасне зростання кількості людей з набутою інвалідністю формують запит на інструменти, що дозволяють оперативно документувати як існуючі бар’єри, так і нові елементи доступності, що з’являються в процесі відбудови. Централізований моніторинг у такому масштабі є практично нездійсненним без залучення самих мешканців.

Таким чином, розробка кросплатформного мобільного застосунку для краудсорсингового картування проблем доступності є обґрунтованою відповіддю на виявлену функціональну прогалину. Застосунок має забезпечувати розміщення геоміток у довільних точках міського простору, прикріплення фотоматеріалів, категоризацію об’єктів за типом та

формування динамічного шару даних поверх базової карти, доступного для перегляду і поповнення будь-яким користувачем.

1.5 Огляд технологічного стеку

Вибір інструментів розробки є визначальним для досягнення ключових вимог застосунку: кросплатформності, масштабованості та низького порогу входу для кінцевого користувача. У цьому підрозділі розглядаються основні технології, обрані для реалізації застосунку, та обґрунтовується доцільність їх використання.

Flutter – це відкритий фреймворк для розробки кросплатформних застосунків, створений компанією Google та вперше представлений у стабільній версії у 2018 році [15]. Фреймворк використовує мову програмування Dart і дозволяє з єдиної кодової бази генерувати нативні застосунки для Android, iOS, Web, Windows, macOS та Linux. Ключовою архітектурною особливістю Flutter є власний рушій рендерингу Skia, що відмальовує інтерфейс безпосередньо на полотні без використання нативних компонентів платформи. Це забезпечує візуальну однорідність застосунку на різних операційних системах та високу продуктивність інтерфейсу.

Для задач розроблюваного застосунку Flutter є оптимальним вибором з кількох причин. По-перше, необхідність підтримки як мобільних платформ, так і вебверсії без дублювання коду суттєво скорочує обсяг розробки. По-друге, екосистема пакетів pub.dev містить готові інтеграції з Firebase та Google Maps SDK, що дозволяє зосередитись на прикладній логіці застосунку. По-третє, гаряче перезавантаження у процесі розробки прискорює ітераційний цикл при відлагодженні інтерфейсу.

Для обґрунтування вибору Flutter доцільно порівняти його з найближчою альтернативою – React Native, розробленим компанією Meta. Обидва фреймворки вирішують задачу кросплатформної розробки, однак

відрізняються архітектурним підходом. React Native використовує міст між JavaScript-кодом та нативними компонентами платформи, що забезпечує нативний вигляд інтерфейсу, але вносить затримку при передачі даних через міст та залежність від актуальності нативних компонентів на кожній платформі. Flutter, натомість, рендерить інтерфейс власним рушієм, що гарантує однакову поведінку на всіх платформах і вищу продуктивність анімацій. Для розроблюваного застосунку, де центральним елементом є інтерактивна карта з динамічним шаром маркерів, стабільна продуктивність рендерингу є критичною вимогою. Крім того, офіційна підтримка вебплатформи у Flutter є зрілішою порівняно з React Native Web, що є визначальним фактором з огляду на вимогу кросплатформності.

Firebase – хмарна платформа компанії Google, що надає набір бекенд-сервісів для мобільних та вебзастосунків за моделлю «бекенд як сервіс» (BaaS) [16]. У розроблюваному застосунку використовуються два сервіси платформи.

Firestore – документно-орієнтована хмарна база даних з підтримкою реального часу. Дані зберігаються у вигляді колекцій документів з довільною структурою полів, що забезпечує гнучкість схеми без необхідності міграцій. Для зберігання геоміток використовується колекція reports, кожен документ якої містить координати, тип об’єкта, текстовий опис, посилання на фото та часову мітку створення.

Firebase Storage – сервіс хмарного зберігання файлів, оптимізований для завантаження та отримання користувацьких медіаматеріалів. У застосунку використовується для зберігання фотографій, прикріплених до геоміток. Файли організовані у папці reports/ та доступні через публічні URL, що дозволяє відображати зображення безпосередньо у картці мітки без проміжного сервера.

Перевагою Firebase як платформи є безсерверна архітектура: розробник не адмініструє серверну інфраструктуру, масштабування відбувається автоматично залежно від навантаження. Безкоштовний тарифний план Spark

забезпечує 50 000 читань та 20 000 записів до Firestore на добу і 5 ГБ сховища, що є достатнім для прототипу та початкового етапу експлуатації застосунку. Суттєвим аспектом є також географічне розміщення: для розроблюваного застосунку обрано регіон europe-west3 (Франкфурт), що мінімізує затримку для українських користувачів.

Альтернативою Firebase могла б слугувати власна серверна інфраструктура на основі Node.js та PostgreSQL з розширенням PostGIS для зберігання геопросторових даних. Такий підхід забезпечував би повний контроль над схемою даних та більші можливості для складних геопросторових запитів. Однак для прототипу та початкового етапу розгортання застосунку він є надмірно складним: потребує адміністрування сервера, налаштування резервного копіювання, моніторингу та ручного масштабування. Firebase як BaaS-платформа перекладає ці операційні витрати на постачальника, дозволяючи зосередитись на прикладній логіці. З огляду на обмежений обсяг даних на початковому етапі та безкоштовний тарифний план Spark, Firebase є економічно обґрунтованим вибором для поточної стадії розробки з можливістю міграції на власну інфраструктуру у разі масштабування проєкту.

Google Maps Flutter SDK надає віджет інтерактивної карти з підтримкою розміщення кастомних маркерів, обробки подій дотику та керування камерою [17]. Для розроблюваного застосунку карта є центральним елементом інтерфейсу: користувач переглядає мітки як шар маркерів поверх базової мапи та взаємодіє з ними через тап. SDK підтримує роботу як у мобільному середовищі через нативні Android та iOS компоненти, так і у вебсередовищі через Maps JavaScript API, що відповідає вимозі кросплатформності.

Використання Google Maps як картографічної основи обґрунтовано високою якістю базових даних для українських міст, підтримкою кирилиці в назвах об’єктів та знайомістю інтерфейсу для цільової аудиторії. Щомісячний безкоштовний кредит у розмірі 200 доларів США від Google

Cloud Platform є достатнім для обслуговування застосунку на етапі прототипування та початкового розгортання.

Таким чином, обраний технологічний стек – Flutter, Firebase та Google Maps SDK – забезпечує реалізацію всіх ключових вимог застосунку в рамках єдиної екосистеми інструментів Google, що спрощує інтеграцію компонентів та знижує операційні витрати на початковому етапі розвитку продукту.

1.6 Постановка задачі

Забезпечення доступності міського середовища для маломобільних груп населення є актуальною соціальною проблемою, що набула особливої гостроти в умовах збройного конфлікту в Україні та пов’язаного з ним зростання кількості людей з набутою інвалідністю. Існуючі цифрові інструменти картування доступності не забезпечують можливості фіксації довільних елементів вуличної інфраструктури з прикріпленням фотоматеріалів та формуванням динамічного краудсорсингового шару даних. Тому ставиться завдання розробити кросплатформний мобільний застосунок, що дозволить користувачам розміщувати геомітки проблем та об’єктів доступності у довільних точках міського простору із залученням широкого кола учасників.

Об’єктом роботи є розроблення кросплатформного застосунку для краудсорсингу проблем доступності міського середовища.

Метою роботи є розроблення мобільного застосунку на основі Flutter та Firebase, що забезпечує збір, зберігання та візуалізацію геопросторових даних про стан доступності міської інфраструктури для користувачів крісел колісних із використанням Google Maps як картографічної основи.

Для досягнення мети необхідно вирішити такі завдання:

– проаналізувати стан доступності міського середовища в Україні та існуючі підходи до його моніторингу;

– провести огляд наявних застосунків картування доступності та виявити їх функ ціональні обмеження;

– обґрунтувати вибір технологічного стеку та спроєктувати архітектуру застосунку;

– розробити модель даних та UML-діаграми системи;

– реалізувати функціонал розміщення геоміток з фотоматеріалами та їх відображення на інтерактивній карті;

– забезпечити кросплатформну роботу застосунку на Android та у вебсередовищі;

– провести тестування реалізованого функціоналу та визначити перспективи подальшого розвитку.

2 МОДЕЛЮВАННЯ СИСТЕМИ

2.1 Концептуальна модель: архітектура клієнт-хмара-БД

Розроблюваний застосунок побудований за трирівневою архітектурою, що включає клієнтську частину, хмарну інфраструктуру та картографічний сервіс. Взаємодія між рівнями здійснюється через офіційні SDK та REST-подібні інтерфейси відповідних платформ.

Клієнтська частина реалізована у вигляді Flutter-застосунку, що виконується безпосередньо на пристрої користувача. Вона відповідає за відображення інтерактивної карти, обробку користувацьких дій, формування та відправку даних про нові мітки, а також отримання і візуалізацію існуючих записів з бази даних. Завдяки архітектурі Flutter один і той самий код компілюється як у нативний Android-застосунок, так і у вебзастосунок для браузера.

Хмарна інфраструктура побудована на платформі Firebase і включає два сервіси. Firestore виконує функцію основного сховища структурованих даних: координати, тип, опис та метадані кожної мітки зберігаються як окремий документ у колекції reports. Firebase Storage забезпечує зберігання фотоматеріалів, завантажених користувачами, та надає публічні URL для їх відображення у клієнтській частині.

Картографічний сервіс Google Maps виконує роль базового шару візуалізації. Google Maps Flutter SDK інтегрується у клієнтську частину та забезпечує відображення карти, обробку жестів керування та розміщення кастомних маркерів у заданих координатах. Дані про мітки, отримані з Firestore, накладаються на базову карту у вигляді окремого шару, що є ключовою архітектурною ідеєю застосунку.

Загальна схема взаємодії компонентів наведена на рисунку 2.1.

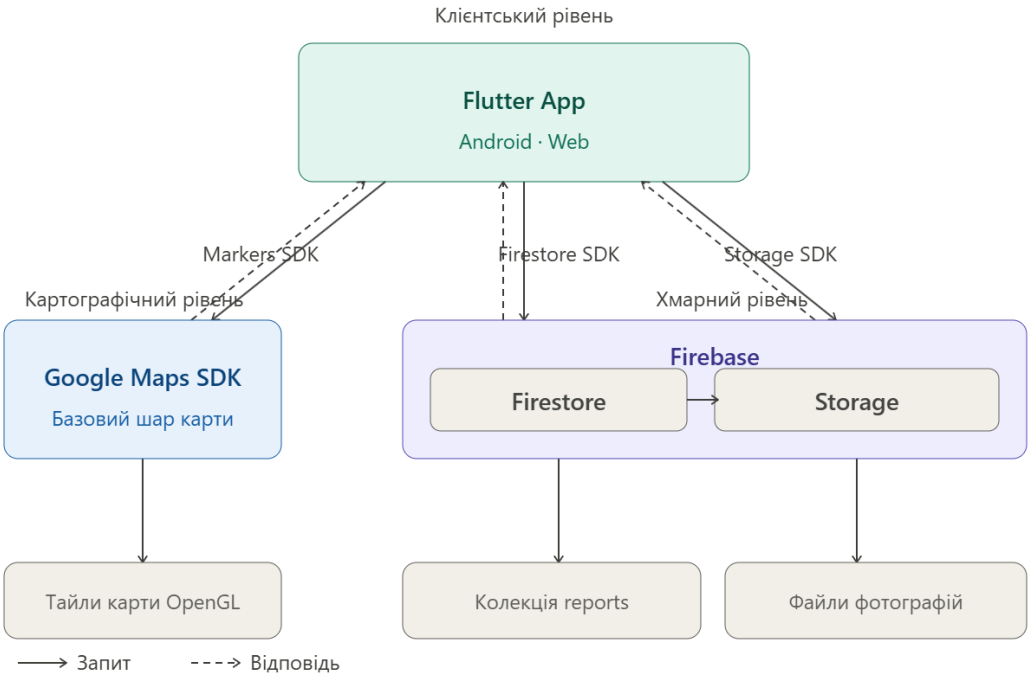

Рисунок 2.1 – Архітектура системи: Flutter App → Firebase (Firestore + Storage) ← → Google Maps SDK

2.2 Діаграми варіантів використання та послідовності

Для формалізованого опису функціональної поведінки системи використовується уніфікована мова моделювання UML. У цьому підрозділі наведено діаграму варіантів використання, що визначає акторів системи та їх взаємодію з функціональними сценаріями, а також діаграму послідовності для ключового сценарію додавання геомітки.

Система має одного основного актора – користувача мобільного або вебзастосунку. На поточному етапі розробки розмежування ролей не реалізовано: усі користувачі мають однаковий рівень доступу до функціоналу перегляду та додавання міток. У перспективі передбачається введення ролі модератора, відповідального за верифікацію повідомлень.

Основні варіанти використання системи:

– перегляд карти з існуючими мітками;
– фільтрація міток за типом об’єкта;

– перегляд детальної картки мітки з фотоматеріалом та описом;

– додавання нової мітки із зазначенням координат, типу, опису та фотографії;

– завантаження фотоматеріалу з галереї пристрою.

Діаграма варіантів використання наведена на рисунку 2.2.

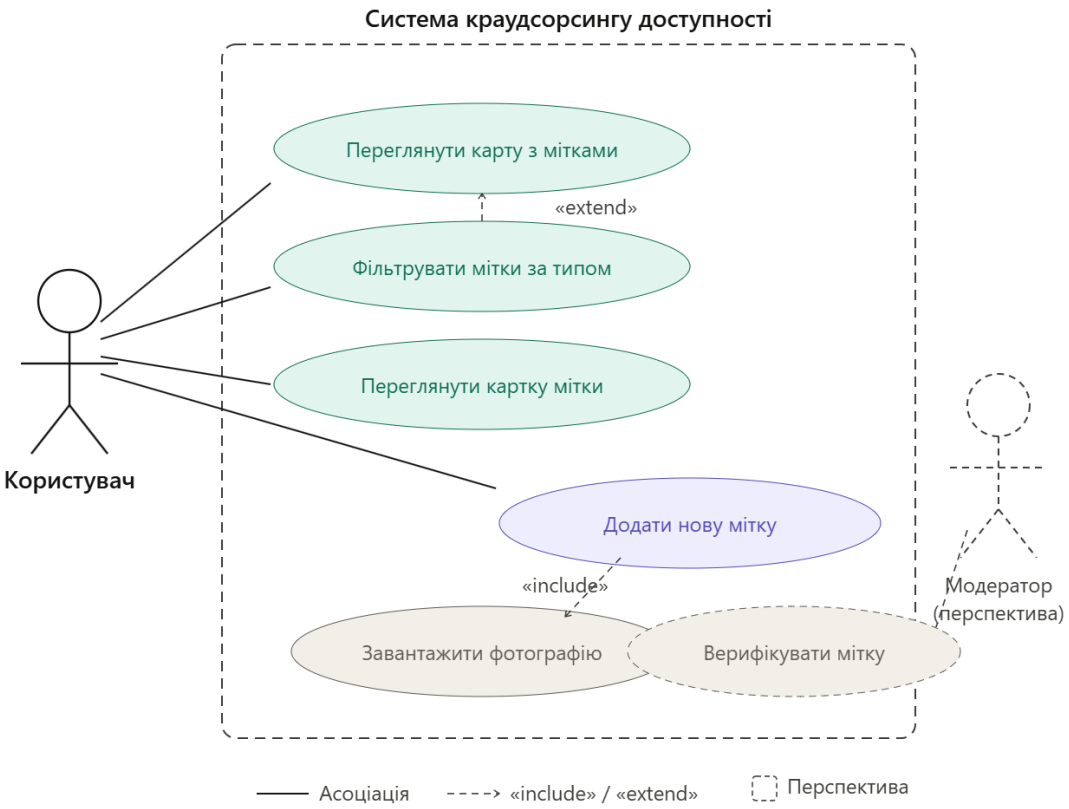

Рисунок 2.2 – Діаграма варіантів використання

Діаграма послідовності описує взаємодію компонентів системи при виконанні сценарію додавання нової геомітки. Сценарій ініціюється користувачем і охоплює чотири учасники: інтерфейс застосунку, Google Maps SDK, Firebase Storage та Firestore.

Сценарій розгортається у такій послідовності. Користувач натискає кнопку «Додати мітку», після чого інтерфейс переходить у режим розміщення – у центрі екрана з’являється пін, а карта стає керованою для точного позиціонування. Після переміщення карти до потрібної точки користувач підтверджує координати, і застосунок відкриває форму введення

даних. Користувач обирає тип об’єкта, вводить текстовий опис та за бажанням прикріплює фотографію з галереї пристрою.

Після підтвердження форми застосунок виконує два послідовні запити до хмарної інфраструктури. Спочатку фотографія завантажується до Firebase Storage, у відповідь повертається публічний URL файлу. Потім до колекції reports у Firestore записується новий документ, що містить координати, тип, опис, отриманий URL фотографії та часову мітку створення. Після успішного запису інтерфейс повертається до режиму перегляду карти, а новий маркер з’являється на ній у вказаній точці.

Діаграма послідовності наведена на рисунку 2.3.

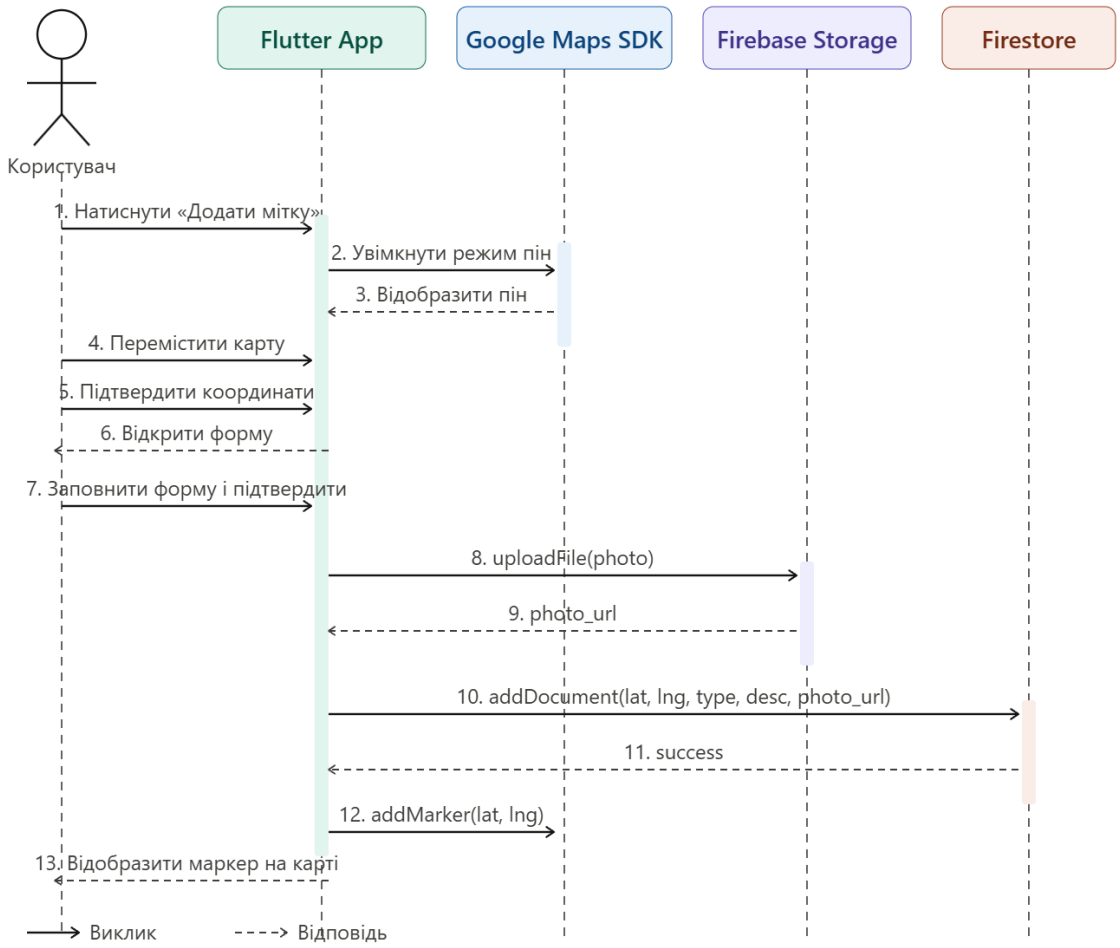

Рисунок 2.3 – Діаграма послідовності додавання геомітки

Описана взаємодія компонентів є базовою для розуміння архітектурних рішень, реалізованих у третьому розділі. Механізми валідації та верифікації

даних, що накладаються на цей сценарій, розглядаються у наступних підрозділах.

Для повноти формального опису поведінки системи розглянемо також діаграму активності для сценарію перегляду існуючої геомітки. На відміну від діаграми послідовності, що фіксує взаємодію між компонентами, діаграма активності описує логіку прийняття рішень та розгалуження потоку керування з точки зору користувача.

Сценарій розпочинається з відображення карти з існуючими маркерами. Діаграма активності наведена на рисунку 2.4.

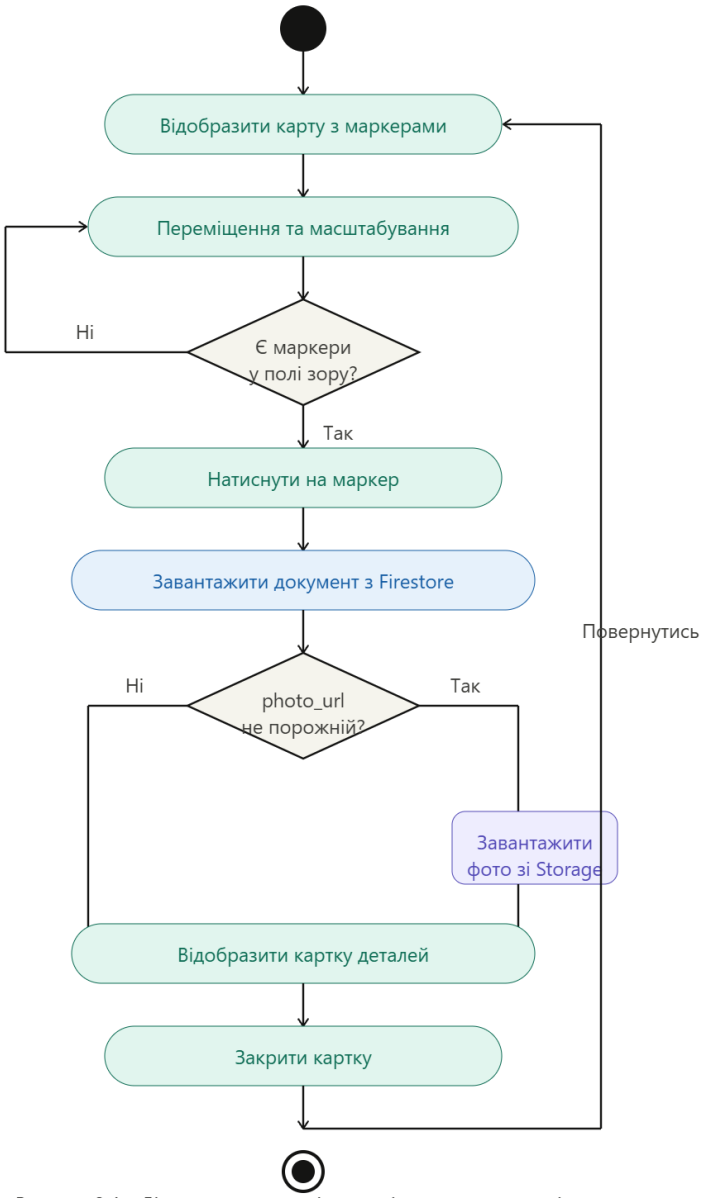

Рисунок 2.4 – Діаграма активності сценарію перегляду геомітки

Користувач переміщує карту та масштабує її до потрібної ділянки. Якщо у полі зору є маркери – користувач обирає один із них натисканням. Система перевіряє наявність фотоматеріалу у документі Firestore: якщо поле photo_url не є порожнім – завантажується зображення з Firebase Storage та відображається у картці деталей разом з описом і типом об’єкта; якщо фото відсутнє – картка відображається лише з текстовими полями. Після перегляду користувач закриває картку та повертається до режиму перегляду карти. Якщо у полі зору маркерів немає – користувач продовжує переміщення карти або переходить до сценарію додавання нової мітки.

2.3 Модель даних

Основою сховища даних застосунку є колекція reports у Firebase Firestore. Кожен документ колекції відповідає одній геомітці, розміщеній користувачем на карті, та містить такі поля:

– lat (double) – географічна широта точки розміщення мітки;

– lng (double) – географічна довгота точки розміщення мітки;

– type (string) – категорія об’єкта: «ramp» (пандус або знижений бордюр) або «obstacle» (перешкода);

– description (string) – текстовий опис, введений користувачем;

– confidence_score (double) – агрегована метрика достовірності звіту в діапазоні від 0,0 до 1,0;

– photo_url (string, nullable) – публічне посилання на фотографію у Firebase Storage;

– created_at (timestamp) – дата та час створення запису.

Для кращого розуміння структури даних наведемо приклад реального документа колекції reports у JSON-нотації, що відповідає мітці про відсутність бордюрного зниження на перехресті.

Лістинг 2.1 Приклад документа колекції reports у JSON-нотації:

{

"lat": 46.4831,

"lng": 30.7214,

"type": "obstacle",

"description": "Відсутнє бордюрне зниження, з'їзд на проїжджу частину неможливий",

"confidence_score": 0.5,

"photo_url": "https://storage.googleapis.com/a11y-map-c90a8.firebasestorage.app/reports/img_20250518_143201.jpg",

"created_at": "2025-05-18T14:32:01Z"

}

Варто зазначити архітектурне рішення щодо зберігання геокоординат. Firestore не має нативного індексованого типу для двовимірних просторових запитів у стилі PostGIS. Для простого завантаження всіх міток і відображення їх на карті клієнтом це не є проблемою: застосунок зчитує всю колекцію та передає координати безпосередньо у Google Maps SDK. Однак для майбутніх задач просторової фільтрації – наприклад, завантаження лише міток у межах поточного вікна карти – використовуватиметься бібліотека GeoFlutterFire, що реалізує геохешування на основі алгоритму Geohash і дозволяє виконувати просторові запити через стандартний Firestore API без зміни схеми даних.

Документно-орієнтована модель Firestore має суттєву перевагу перед реляційною на етапі активного розвитку застосунку: відсутність жорсткої схеми дозволяє додавати нові поля до окремих документів без міграції всієї колекції. Наприклад, після впровадження системи голосування до документів буде додано поле votes з вкладеними лічильниками підтверджень та заперечень, а після введення авторизації – поле user_id для прив'язки мітки до автора. Існуючі документи без цих полів продовжать коректно оброблятись клієнтом завдяки null-safe обробці у Dart.

Поле confidence_score є ключовим для системи валідації та детально розглядається у наступних підрозділах. На поточному етапі розробки його значення встановлюється рівним 0,5 за замовчуванням і буде обчислюватись динамічно після впровадження відповідних механізмів верифікації.

Документно-орієнтована модель Firestore не передбачає жорсткої схеми, що дозволяє додавати нові поля без міграції існуючих даних. Це є перевагою на етапі активного розвитку застосунку, коли структура даних може уточнюватись. Зокрема, у перспективі планується додавання поля verified_by для зберігання ідентифікатора модератора, що підтвердив мітку, та поля votes для зберігання кількості голосів підтвердження від інших користувачів.

Схема структури документа колекції reports наведена на рисунку 2.5.

Поле	Тип	Опис
lat	double	Географічна широта мітки
lng	double	Географічна довгота мітки
type	string	"ramp" або "obstacle"
description	string	Текстовий опис від користувача
confidence_score	double	Достовірність звіту [0.0–1.0]
photo_url	string | null	URL фото у Firebase Storage
created_at	timestamp	Дата та час створення запису

Рисунок 2.5 – Структура документа колекції reports у Firestore

2.4 Модель валідації краудсорсингових звітів

Ефективність краудсорсингової системи визначається не лише кількістю зібраних даних, а й їх якістю. Основними проблемами при зборі геопросторових репортів від користувачів є: позиційна похибка GPS, дублювання повідомлень про один і той самий об’єкт від різних користувачів, семантична неоднорідність описів та недостатня інформативність фотоматеріалів.

Для комплексного оцінювання достовірності кожного звіту пропонується агрегована метрика – індикатор $conf_{score}$, що обчислюється як зважена сума чотирьох складових:

$$conf_{score} = w_1 \cdot GPS_{conf} + w_2 \cdot photo_{conf} + w_3 \cdot crowd_{conf} + w_4 \cdot meta_{conf}, \quad (2.1)$$

де GPS_{conf} – позиційна надійність у діапазоні [0, 1], визначається на основі точності сигналу GPS та відстані до найближчого об’єкта OpenStreetMap після застосування map-matching;

$photo_{conf}$ – якість фотоматеріалу у діапазоні [0, 1]: за наявності моделі комп’ютерного зору – оцінка відповідності фото заявленій категорії; за відсутності – 1,0 за наявності чіткого фото, 0,5 за темне або малоінформативне зображення, 0,0 за відсутність фото;

$crowd_{conf}$ – нормалізована кількість підтверджень від інших користувачів у діапазоні [0, 1];

$meta_{conf}$ – повнота метаданих у діапазоні [0, 1], враховує наявність текстового опису, категорії та фотографії.

Для корекції позиціонування використовується алгоритм map-matching: GPS-координати кожного повідомлення прив’язуються до найближчого об’єкта OpenStreetMap з обмеженням відстані у 20 метрів. Такий підхід зменшує похибку позиціонування в умовах міської забудови, де типова

точність смартфона становить від 3 до 20 метрів через явище «урбан-каньйону» – відбиття сигналів від фасадів будівель.

Для виявлення та усунення дублікатів застосовується двоетапна процедура, схема якої наведена на рисунку 2.6. На першому етапі алгоритм DBSCAN (Density-Based Spatial Clustering of Applications with Noise) [18] виконує просторову кластеризацію звітів: повідомлення, координати яких знаходяться у межах заданого радіуса, об’єднуються в один кластер; адаптивні методи кластеризації потоків даних, придатні для опрацювання неперервно надходячих репортів, описані у [35, 36]. На другому етапі всередині кожного кластера виконується семантична перевірка: порівнюються категорії об’єктів та обчислюється схожість фотографій за допомогою алгоритму перцептивного хешування (pHash) або векторних ембеддингів; структурні методи порівняння та пошуку візуальних об’єктів детально розглянуті у [25, 26]. Якщо обидві умови виконуються, кластер вважається дублікатом і замінюється одним агрегованим записом з підвищеним значенням crowdconf.

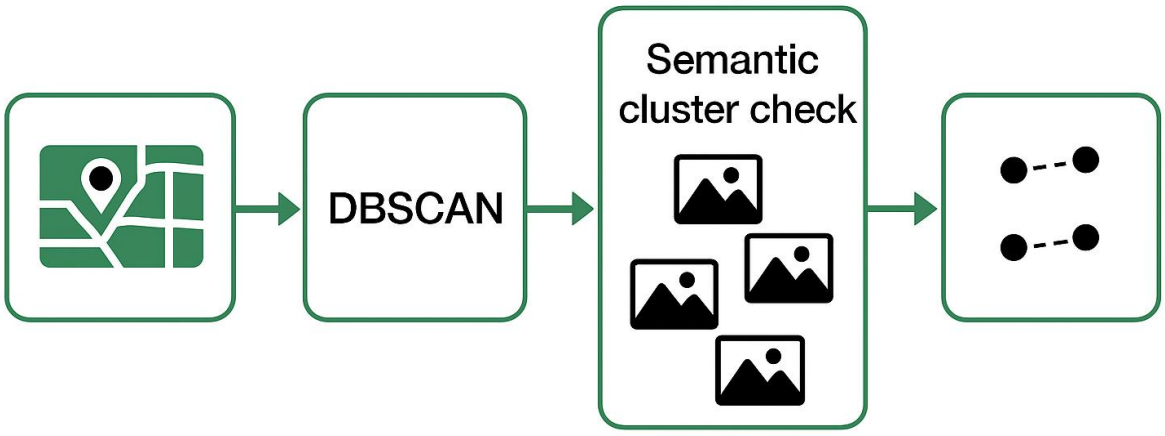

Рисунок 2.6 – Двоетапна схема виявлення дублікатів: просторова кластеризація DBSCAN та семантична перевірка

Для наочності розглянемо числовий приклад обчислення confidence_score для двох звітів про один і той самий об’єкт. Нехай два користувачі незалежно зафіксували відсутній пандус на вході до будівлі.

Перший звіт містить чітке фото, повний текстовий опис та точний GPS-сигнал; другий надіслано без фото, з коротким описом та з помірною точністю геолокації. Значення складових та результуючий показник для кожного звіту наведено у таблиці 2.1.

Таблиця 2.1 – Числовий приклад обчислення confidence_score

Складова	**Звіт 1**	**Звіт 2**	**Ваговий коефіцієнт**
GPSconf	0,90	0,60	w1 = 0,30
photoconf	1,00	0,00	w2 = 0,35
crowdconf	0,50	0,50	w3 = 0,20
metaconf	1,00	0,50	w4 = 0,15
confidence_score	**0,875**	**0,430**	

Звіт 1 з показником 0,875 перевищує поріг автоматичного прийняття і публікується на карті без ручної перевірки. Звіт 2 з показником 0,430 потрапляє у зону невизначеності та позначається для верифікації модератором. Після того як алгоритм DBSCAN об’єднує обидва звіти в один кластер, агрегований запис отримує підвищене значення crowdconf = 1,0, що відображає факт підтвердження об’єкта двома незалежними джерелами. Перерахований confidence_score агрегованого запису становить 0,935, що однозначно переводить його до категорії верифікованих.

Параметри алгоритму DBSCAN для задачі дедуплікації геоміток обираються з урахуванням типової точності GPS-сигналу смартфона в умовах міської забудови. Значення epsilon встановлюється рівним 20 метрам, що відповідає максимальній очікуваній похибці позиціонування в умовах «урбан-каньйону». Мінімальна кількість точок у кластері minPts дорівнює 2, оскільки навіть два незалежних звіти про один об’єкт є достатньою підставою для їх об’єднання. Шумові точки – звіти, що не увійшли до

жодного кластера – залишаються як окремі записи з початковим значенням confidence_score до надходження нових підтверджень.

Порогове значення confidence_score використовується для автоматичного маршрутизування звіту: репорти з високим показником приймаються автоматично, з низьким – позначаються для ручної перевірки модератором; загальні технології прийняття рішень в інформаційних системах розглянуті у [41]. Така багаторівнева система дозволяє автоматизовано фільтрувати звіти низької якості, одночасно зберігаючи сумнівні випадки для верифікації, та підвищує корисність даних для планування доступності інфраструктури.

2.5 Просторова модель покриття та пріоритизація зон

Класичною системною проблемою краудсорсингових платформ є похибка вибірки (sampling bias): більшість повідомлень формується у центральних, добре відвідуваних районах міста, тоді як периферійні зони залишаються недостатньо задокументованими. Це призводить до того, що система генерує хибні висновки про просторові пріоритети та залишає «сліпі зони» на мапі доступності. Проблема є замкненою: користувачі обирають вже відомі маршрути, нові ділянки не пропонуються для дослідження, оскільки їхній показник впевненості маршруту є нижчим за вже задокументовані.

Для вирішення цієї проблеми пропонується модель динамічної пріоритизації зон збору даних. Територія міста розбивається на просторову сітку за допомогою діаграми Вороного або гексагональної сітки H3, де кожна комірка отримує динамічний індекс пріоритету P(z), що обчислюється за формулою:

$$P(Z_i) = \alpha \cdot \frac{1}{D(Z_i)+\epsilon} + \beta \cdot I(Z_i), \tag{2.2}$$

де $D(Z_i)$ – кількість існуючих валідованих репортів у зоні z;

ϵ – мала константа для уникнення ділення на нуль;

$I(Z_i)$ – коефіцієнт інфраструктурної важливості зони (враховує наявність лікарень, зупинок громадського транспорту, соціальних закладів);

α та β – вагові коефіцієнти, що дозволяють системі гнучко балансувати між дослідженням нових територій та деталізацією критично важливих зон.

На основі цього індексу реалізується підсистема просторового мікро-таскінгу (spatial micro-tasking). Коли користувач із увімкненою геолокацією наближається до зони з критично високим показником пріоритету, система ініціює таргетований запит у вигляді push-повідомлення. Після того як користувач виконує завдання і завантажує валідований репорт, значення R(z) для цієї зони зростає, що автоматично знижує її індекс пріоритету. Система динамічно переключає фокус на наступну «сліпу зону», реалізуючи принцип саморегуляції.

Схема динамічної саморегуляції краудсорсингової системи наведена на рисунку 2.7.

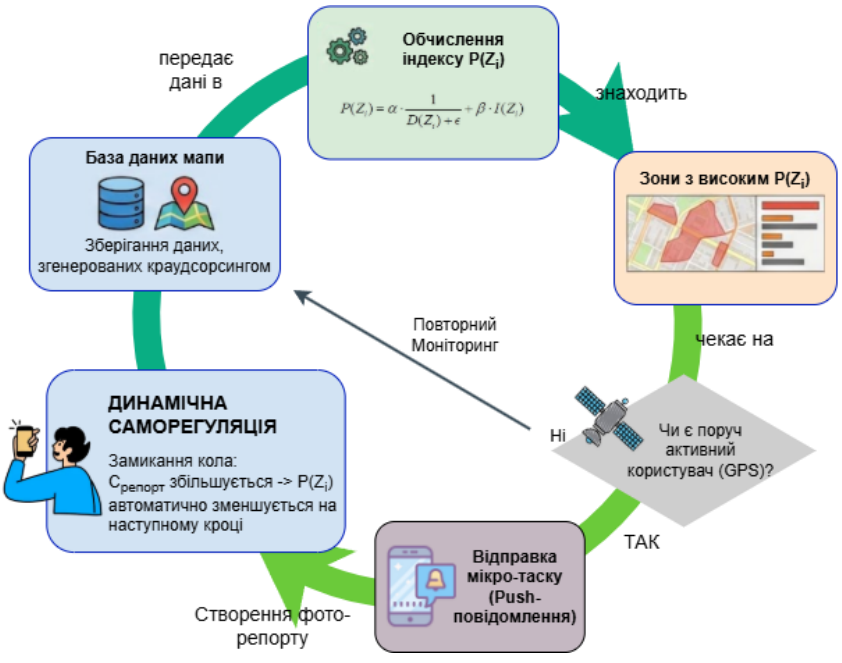

Рисунок 2.7 – Схема динамічної саморегуляції: цикл перерахунку індексу пріоритету

Застосування елементів гейміфікації – рейтингів та віртуальних нагород – додатково стимулює користувачів свідомо відхилятися від звичних маршрутів заради збору соціально важливих просторових даних, розриваючи проблему замкненого кола недостатнього покриття.

Поведінку запропонованого індексу зручно простежити на якісному порівнянні двох зон. У центральному районі з великою кількістю валідованих репортів та помірною інфраструктурною важливістю індекс пріоритету залишається низьким – система не спрямовує туди додаткові запити, оскільки покриття вже достатнє. Натомість периферійний район з малою кількістю репортів отримує високий індекс навіть за нижчого коефіцієнта інфраструктурної важливості: брак даних переважує у підсумковій оцінці. Саме до таких зон система надсилає цільові запити користувачам, які опиняються поруч.

Ключовою властивістю моделі є саморегуляція. Щойно користувач завантажує валідований репорт до пріоритетної зони, кількість репортів у ній зростає, а її індекс пріоритету відповідно знижується. Фокус системи автоматично переключається на наступну недостатньо задокументовану ділянку. У такий спосіб кожен новий репорт зменшує пріоритет своєї зони та підвищує відносну привабливість сусідніх, що поступово вирівнює покриття території міста.

2.6 Модель гібридної валідації фотоматеріалів

Фотографії, прикріплені до геоміток, є ключовим джерелом інформації для верифікації звітів, проте їх автоматична обробка пов’язана з суттєвими труднощами. Методи цифрової обробки та аналізу зображень, що становлять основу автоматичної верифікації, детально розглянуті у [23, 24] Якість знімків, зроблених користувачами в польових умовах, є вкрай неоднорідною: частина фотографій є нечіткою, темною або не відповідає заявленій категорії

об’єкта; методи нормалізації зображень відносно еталона, що можуть передувати класифікації, описані у [39]. Крім того, наявні моделі комп’ютерного зору не забезпечують достатнього рівня впевненості для повністю автоматичної класифікації у всіх випадках.

Для вирішення цієї проблеми пропонується гібридний підхід, що поєднує автоматичну класифікацію за допомогою згорткової нейронної мережі (CNN) з механізмом Human-in-the-Loop (HITL) [22]. Принцип роботи пайплайну є таким: якщо впевненість моделі у класифікації фотографії перевищує встановлений поріг, звіт приймається автоматично; якщо ж впевненість є нижчою за поріг, фотографія скеровується на перевірку до експерта-модератора. Такий підхід дозволяє автоматизувати обробку більшості звітів, зберігаючи людський контроль над складними та неоднозначними випадками.

Схема гібридного пайплайну валідації фотоматеріалів наведена на рисунку 2.8.

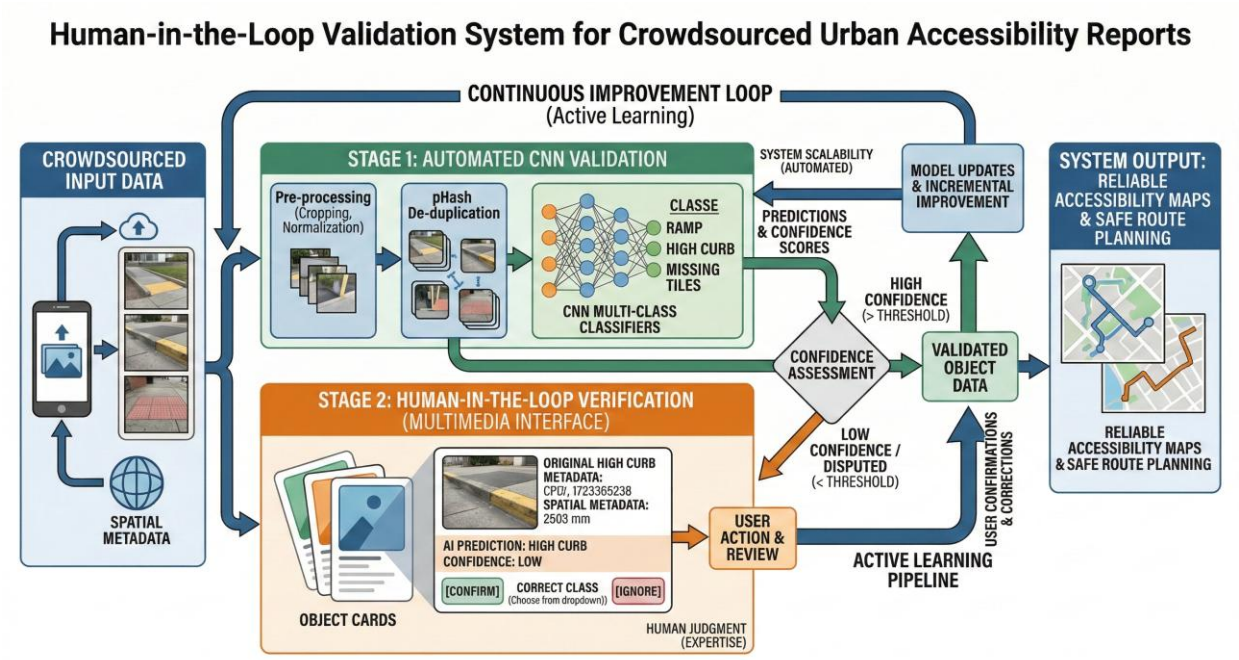

Рисунок 2.8 – Схема HITL-пайплайну

Важливою властивістю запропонованого підходу є можливість безперервного покращення моделі за рахунок механізму активного навчання (active learning). Фотографії, що були скеровані на ручну перевірку та отримали підтверджену експертом категорію, повертаються до навчальної вибірки як нові розмічені зразки. Періодичне донавчання моделі на цих зразках поступово підвищує її точність та зменшує частку звітів, що потребують ручної перевірки. Таким чином механізм HITL виконує подвійну функцію: забезпечує якість поточної валідації та формує джерело даних для вдосконалення моделі.

Вибір конкретної архітектури згорткової мережі та порогового значення впевненості, а також кількісна оцінка ефективності запропонованого підходу, виконані експериментально та наведені у наступних підрозділах.

Окремою концепцією, що розвиває запропонований підхід, є «стежки доступності» (accessibility trails) – реконструкція фактичних маршрутів маломобільних користувачів на основі GPS-треків їхніх переміщень. Накопичені треки дозволяють виявляти ділянки, яких користувачі систематично уникають, та формувати припущення про наявність бар’єрів навіть за відсутності явних звітів.

2.7 Функціональні та нефункціональні вимоги

На основі проведеного аналізу предметної області та спроєктованих моделей формулюються вимоги до розроблюваного застосунку.

Функціональні вимоги визначають поведінку системи з точки зору користувача та наведені у таблиці 2.2.

Таблиця 2.2 – Функціональні вимоги до застосунку

Код	**Вимога**
ФВ-1	Застосунок повинен відображати інтерактивну карту з усіма існуючими геомітками
ФВ-2	Користувач повинен мати можливість розмістити нову мітку у довільній точці карти
ФВ-3	При створенні мітки користувач повинен мати можливість обрати тип об’єкта та ввести опис
ФВ-4	Користувач повинен мати можливість прикріпити фотографію до мітки з галереї пристрою
ФВ-5	При натисканні на маркер застосунок повинен відображати детальну картку мітки з фото
ФВ-6	Застосунок повинен зберігати дані про мітки у хмарній базі даних
ФВ-7	Застосунок повинен коректно функціонувати на платформах Android та Web

Нефункціональні вимоги визначають якісні характеристики системи та наведені у таблиці 2.3.

Таблиця 2.3 – Нефункціональні вимоги до застосунку

Код	**Вимога**
НФВ-1	Час завантаження карти з мітками не повинен перевищувати 3 секунди при стабільному з’єднанні
НФВ-2	Застосунок повинен зберігати функціональність при обсязі бази даних до 10 000 міток
НФВ-3	Інтерфейс повинен бути адаптований для керування однією рукою на мобільному пристрої
НФВ-4	Застосунок повинен використовувати захищене з’єднання HTTPS для всіх запитів до Firebase

Продовження таблиці 2.3

Код	Вимога
НФВ-5	Вихідний код повинен бути організований таким чином, щоб забезпечити можливість розширення функціоналу без переписування базової архітектури

На момент написання звіту реалізація застосунку знаходиться на етапі функціонального прототипу. Для забезпечення прозорості щодо стану розробки доцільно зіставити сформульовані вимоги з фактичним рівнем їх виконання. Відповідна інформація наведена у таблиці 2.4.

Таблиця 2.4 – Стан виконання вимог у поточній реалізації

Код	Вимога	Стан	Примітка
ФВ-1	Відображення карти з мітками	Реалізовано повністю	Завантаження з Firestore при запуску
ФВ-2	Розміщення мітки у довільній точці	Реалізовано повністю	Режим пін з підтвердженням координат
ФВ-3	Вибір типу та введення опису	Реалізовано повністю	Форма з двома полями
ФВ-4	Прикріплення фотографії	Реалізовано повністю	Вибір з галереї через image_picker
ФВ-5	Картка деталей мітки з фото	Реалізовано повністю	Bottom sheet з фото та описом
ФВ-6	Зберігання у хмарній БД	Реалізовано повністю	Firestore + Firebase Storage
ФВ-7	Робота на Android та Web	Реалізовано повністю	Протестовано на Redmi Note 8 Pro та Chrome

Продовження таблиці 2.4

Код	Вимога	Стан	Примітка
НФВ-1	Час завантаження до 3 секунд	Реалізовано частково	Без пагінації – може сповільнитись при >500 міток
НФВ-2	Робота при 10 000 міток	Не реалізовано	Потребує пагінації або просторової фільтрації
НФВ-3	Керування однією рукою	Реалізовано частково	Кнопки розміщено внизу екрана, UX не оптимізовано
НФВ-4	Захищене з’єднання HTTPS	Реалізовано повністю	Firebase SDK використовує HTTPS за замовчуванням
НФВ-5	Розширюваність архітектури	Реалізовано частково	Код у одному файлі, рефакторинг заплановано

Аналіз таблиці 2.4 показує, що весь ключовий функціонал застосунку реалізовано повністю. Основні невиконані вимоги стосуються масштабованості при великих обсягах даних та оптимізації UX, що є природним для стадії прототипу та планується до виконання у наступних ітераціях розробки. Зокрема, впровадження просторової фільтрації через GeoFlutterFire вирішить проблему НФВ-2, а рефакторинг з виділенням окремих віджетів і сервісів покращить відповідність НФВ-5.

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

3.1 Обґрунтування вибору інструментів

Вибір технологічного стеку для реалізації застосунку здійснювався з урахуванням трьох ключових критеріїв: кросплатформності, зрілості екосистеми та мінімізації операційних витрат на початковому етапі розробки. У цьому підрозділі наведено порівняльний аналіз розглянутих альтернатив та обґрунтування прийнятих рішень.

Для реалізації клієнтської частини розглядалися два основних фреймворки кросплатформної розробки: Flutter та React Native. Порівняльний аналіз за ключовими характеристиками наведено у таблиці 3.1.

Таблиця 3.1 – Порівняльний аналіз Flutter та React Native

Критерій	**Flutter**	**React Native**
Мова програмування	Dart	JavaScript / TypeScript
Підхід до рендерингу	Власний рушій (Skia/Impeller)	Нативні компоненти платформи
Підтримка Web	Офіційна, стабільна	Експериментальна (React Native Web)
Підтримка Google Maps	Офіційний пакет google_maps_flutter	Сторонній пакет react-native-maps
Інтеграція з Firebase	Офіційні FlutterFire плагіни	Сторонні бібліотеки
Продуктивність анімацій	Висока	Середня (залежить від мосту JS)
Розмір спільноти	Зростаюча	Велика, зріла

За результатами порівняння Flutter обрано як основний фреймворк з кількох причин. По-перше, офіційна підтримка веб-платформи є критичною вимогою для забезпечення кросплатформності без додаткових бібліотек. По-друге, наявність офіційних пакетів google_maps_flutter та FlutterFire гарантує сумісність з обраною хмарною інфраструктурою та регулярні оновлення від самої Google. По-третє, власний рушій рендерингу Skia забезпечує стабільну продуктивність при відображенні динамічного шару маркерів на карті незалежно від платформи.

Для вибору бекенд-інфраструктури розглядалися два підходи: Firebase як BaaS-платформа та власний сервер на основі Node.js з базою даних PostgreSQL та розширенням PostGIS. Перевагою власного сервера є повний контроль над схемою даних і можливість виконання складних геопросторових запитів засобами SQL. Однак цей підхід потребує адміністрування серверної інфраструктури, налаштування резервного копіювання та ручного масштабування, що є надмірним для поточної стадії розробки. Firebase як BaaS перекладає операційні витрати на постачальника і дає можливість зосередитись на прикладній логіці застосунку. Безкоштовний тарифний план Spark з лімітом 50 000 читань та 20 000 записів на добу є достатнім для прототипу та початкового етапу експлуатації.

Для картографічної основи розглядалися Google Maps SDK, OpenStreetMap через бібліотеку flutter_map та Mapbox. OpenStreetMap є безкоштовним і має відкриті дані, однак якість покриття українських міст є нижчою порівняно з Google Maps, а бібліотека flutter_map не має офіційної підтримки. Mapbox забезпечує високу якість карт та гнучке стилювання, але є платним після перевищення ліміту запитів. Google Maps обрано з огляду на найвищу якість даних для України, знайомість інтерфейсу для цільової аудиторії та щомісячний безкоштовний кредит у 200 доларів США від Google Cloud Platform, якого достатньо для обслуговування застосунку на етапі прототипування.

3.2 Архітектура застосунку та налаштування проєкту

Застосунок реалізовано як єдиний Flutter-проєкт, що компілюється у нативний Android-застосунок та веб-застосунок без змін у вихідному коді. Проєкт створено за допомогою команди flutter create зі стандартною структурою каталогів, до якої внесено мінімальні зміни для організації логіки застосунку.

Файлова структура проєкту представлено на зображенні 3.1

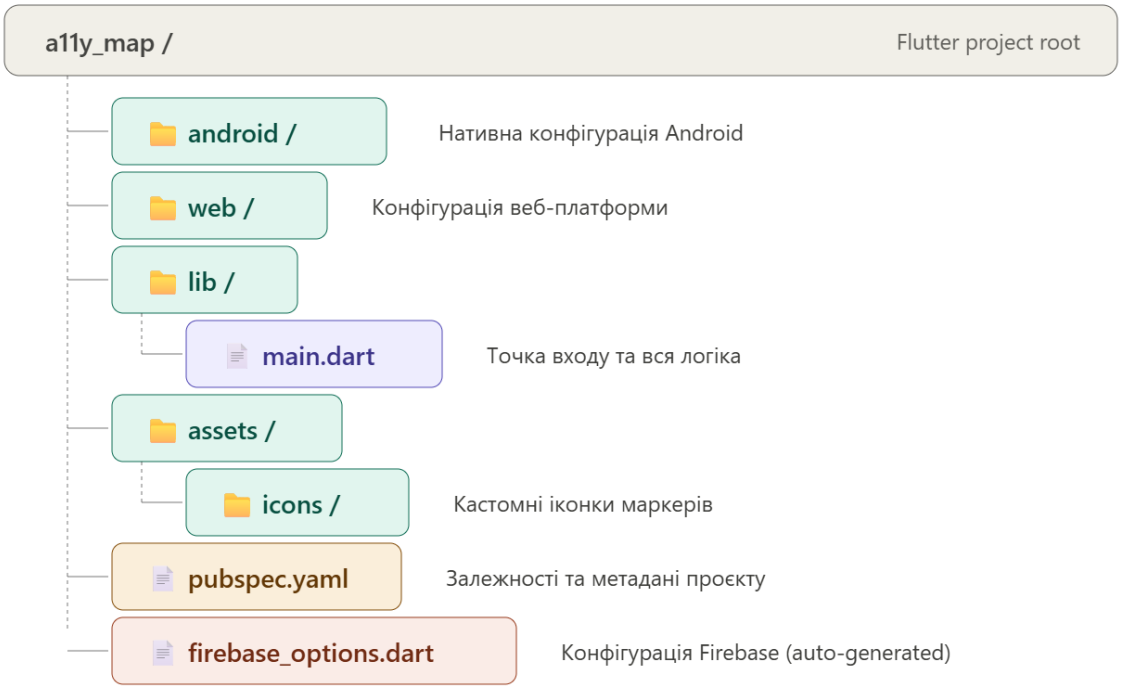

Рисунок 3.1 – Файлова структура проєкту a11y_map

На поточному етапі розробки вся логіка застосунку зосереджена у файлі main.dart. Такий підхід є прийнятним для прототипу і дозволяє швидко ітерувати функціонал. У перспективі планується рефакторинг з виділенням окремих файлів для екранів, сервісів та моделей даних відповідно до архітектурного патерну Feature-first або Layer-first.

Файл pubspec.yaml визначає залежності проєкту. Основні використовувані пакети наведено у таблиці 3.2.

Таблиця 3.2 – Основні залежності проєкту

Пакет	Версія	Призначення
google_maps_flutter	^2.5.0	Відображення інтерактивної карти
cloud_firestore	^4.15.0	Робота з базою даних Firestore
firebase_storage	^11.6.0	Завантаження фотографій
firebase_core	^2.27.0	Ініціалізація Firebase
image_picker	^1.0.7	Вибір фото з галереї пристрою
uuid	^4.3.3	Генерація унікальних імен файлів

Ініціалізація застосунку виконується у функції main() файлу main.dart. Перед запуском Flutter-рушія необхідно послідовно ініціалізувати платформні прив'язки та Firebase.

Лістинг 3.1 Ініціалізація Firebase у функції main():

```
void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp(
    options: DefaultFirebaseOptions.currentPlatform,
  );
  runApp(const MyApp());
}
```

Виклик WidgetsFlutterBinding.ensureInitialized() є обов'язковим при використанні асинхронного коду до запуску рушія. Об'єкт DefaultFirebaseOptions.currentPlatform генерується автоматично утилітою FlutterFire CLI та зберігається у файлі firebase_options.dart. Він містить окремі конфігурації для кожної платформи – Android, iOS та Web – і обирає потрібну під час компіляції.

Для коректної роботи Google Maps SDK необхідне додаткове платформне налаштування. На платформі Android ключ API Google Maps вноситься до файлу android/app/src/main/AndroidManifest.xml у вигляді метаданих застосунку. На веб-платформі відповідний скрипт підключається у

файлі web/index.html до завантаження Flutter-рушія. Обидва ключі отримуються через Google Cloud Console та обмежуються відповідними платформами для запобігання несанкціонованому використанню.

Структура кореневого віджета MyApp є мінімальною: визначає MaterialApp з темою та встановлює MapScreen як головний екран. Уся прикладна логіка зосереджена в MapScreen – StatefulWidget, що керує станом карти, колекцією маркерів та формою додавання мітки. Такий підхід є прийнятним для прототипу, однак при подальшому розвитку застосунку передбачається розподіл відповідальності між окремими віджетами та сервісними класами відповідно до принципу єдиної відповідальності.

3.3 Реалізація інтерактивної карти та відображення маркерів

Центральним елементом інтерфейсу застосунку є інтерактивна карта з динамічним шаром геоміток. Реалізація карти побудована на основі пакету google_maps_flutter та включає три взаємопов’язані задачі: ініціалізацію картографічного віджету, завантаження міток з Firestore та відображення кастомних маркерів у відповідних координатах.

Картографічний віджет GoogleMap розміщується як основний елемент екрана та займає весь доступний простір. При ініціалізації задається початкова позиція камери – координати центру міста та рівень масштабування, що охоплює достатню площу для огляду існуючих міток. (рис. 3.2)

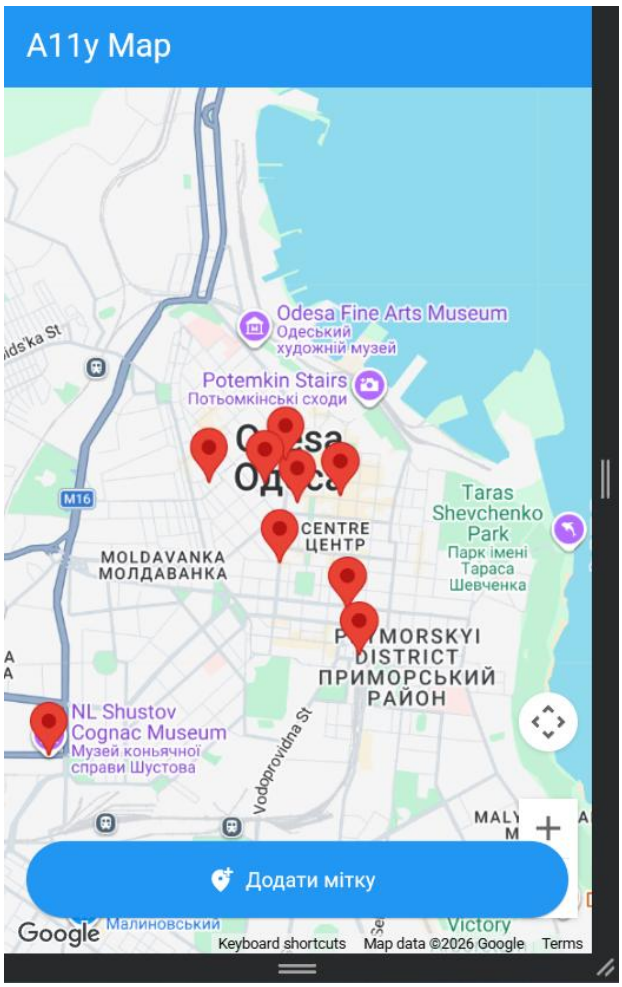

Рисунок 3.2 – Приклад інтерфейсу мапи застосунку та маркерів

Лістинг 3.2 Конфігурація віджета GoogleMap:

```
GoogleMap(
  onMapCreated: _onMapCreated,
  initialCameraPosition: CameraPosition(
    target: LatLng(49.9935, 36.2304),
    zoom: 13.0,
  ),
  markers: _markers,
  onTap: _onMapTap,
  myLocationEnabled: true,
  myLocationButtonEnabled: true,
)
```

Параметр markers отримує множину об’єктів типу Set<Marker>, що зберігається у стані віджету. При зміні множини Flutter автоматично перемальовує шар маркерів без повного перезавантаження карти, що забезпечує плавність інтерфейсу. Параметри myLocationEnabled та myLocationButtonEnabled активують відображення поточного місцезнаходження користувача та кнопку центрування – стандартні елементи, знайомі користувачу з Google Maps.

Завантаження міток з Firestore виконується одноразово при ініціалізації екрана у методі _loadMarkers(). Запит отримує всі документи колекції reports та перетворює кожен з них на об’єкт Marker.

Лістинг 3.3 Завантаження маркерів з колекції Firestore:

```
Future<void> _loadMarkers() async {
 final snapshot = await FirebaseFirestore.instance
    .collection('reports')
    .get();
 final markers = snapshot.docs.map((doc) {
  final data = doc.data();
  return Marker(
    markerId: MarkerId(doc.id),
    position: LatLng(data['lat'], data['lng']),
    icon: data['type'] == 'ramp'
       ? _rampIcon
       : _obstacleIcon,
    onTap: () => _showMarkerDetails(data),
  );
 }).toSet();
 setState(() => _markers = markers);
}
```

Для візуального розрізнення типів об’єктів застосовуються кастомні іконки маркерів. Стандартний червоний маркер Google Maps не несе семантичного навантаження, тому для категорій ramp та obstacle завантажуються окремі зображення з папки assets/icons/. Іконки попередньо завантажуються у методі _loadCustomIcons(), що викликається при ініціалізації екрана.

Лістинг 3.4 Попереднє завантаження кастомних іконок маркерів:

```
Future<void> _loadCustomIcons() async {
  _rampIcon = await BitmapDescriptor.fromAssetImage(
    const ImageConfiguration(size: Size(48, 48)),
    'assets/icons/ramp_marker.png',
  );
  _obstacleIcon = await BitmapDescriptor.fromAssetImage(
    const ImageConfiguration(size: Size(48, 48)),
    'assets/icons/obstacle_marker.png',
  );
}
```

Використання BitmapDescriptor.fromAssetImage замість стандартного BitmapDescriptor.defaultMarker дає змогу застосовувати довільні растрові зображення як маркери. Розмір 48×48 логічних пікселів обраний з урахуванням щільності екранів сучасних смартфонів та забезпечує достатню розбірливість на картах різного масштабу.

При натисканні на маркер викликається метод _showMarkerDetails(), що відображає нижній аркуш (ModalBottomSheet) з детальною інформацією про мітку. Нижній аркуш містить фотографію об’єкта, завантажену за збереженим URL з Firebase Storage, текстовий опис та категорію. Якщо поле photo_url є порожнім, замість зображення відображається заглушка з іконкою камери.

Лістинг 3.5 Відображення картки деталей мітки:

```
void _showMarkerDetails(Map<String, dynamic> data) {
 showModalBottomSheet(
  context: context,
  builder: (_) => Padding(
   padding: const EdgeInsets.all(16),
   child: Column(
    mainAxisSize: MainAxisSize.min,
    crossAxisAlignment: CrossAxisAlignment.start,
    children: [
     if (data['photo_url'] != null)
      Image.network(data['photo_url'],
        height: 180, fit: BoxFit.cover)
     else
      const Icon(Icons.camera_alt, size: 64),
     const SizedBox(height: 12),
     Text(data['type'] == 'ramp'
       ? 'Пандус / знижений бордюр'
       : 'Перешкода'),
     const SizedBox(height: 8),
     Text(data['description'] ?? ''),
    ],
   ),
  ),
 );
}
```

Поточна реалізація завантажує всі мітки колекції одноразово при запуску. Такий підхід є достатнім для бази даних обсягом до кількох сотень

записів, однак при зростанні кількості міток потребуватиме впровадження просторової фільтрації через бібліотеку GeoFlutterFire, що описано у наступних розділах як перспективу розвитку.

3.4 Реалізація форми додавання геомітки

Сценарій додавання нової геомітки є ключовим функціональним сценарієм застосунку і реалізований у два послідовних етапи: визначення координат через режим позиціонування та заповнення форми з даними про об’єкт.

Режим позиціонування активується натисканням плаваючої кнопки у правому нижньому куті екрана. При переході до цього режиму у центрі екрана фіксується піктограма пін, а карта залишається керованою для точного вибору місця. (рис. 3.3) Координати визначаються з центру поточного положення камери через метод _controller.getVisibleRegion() та перетворення у центральну точку.

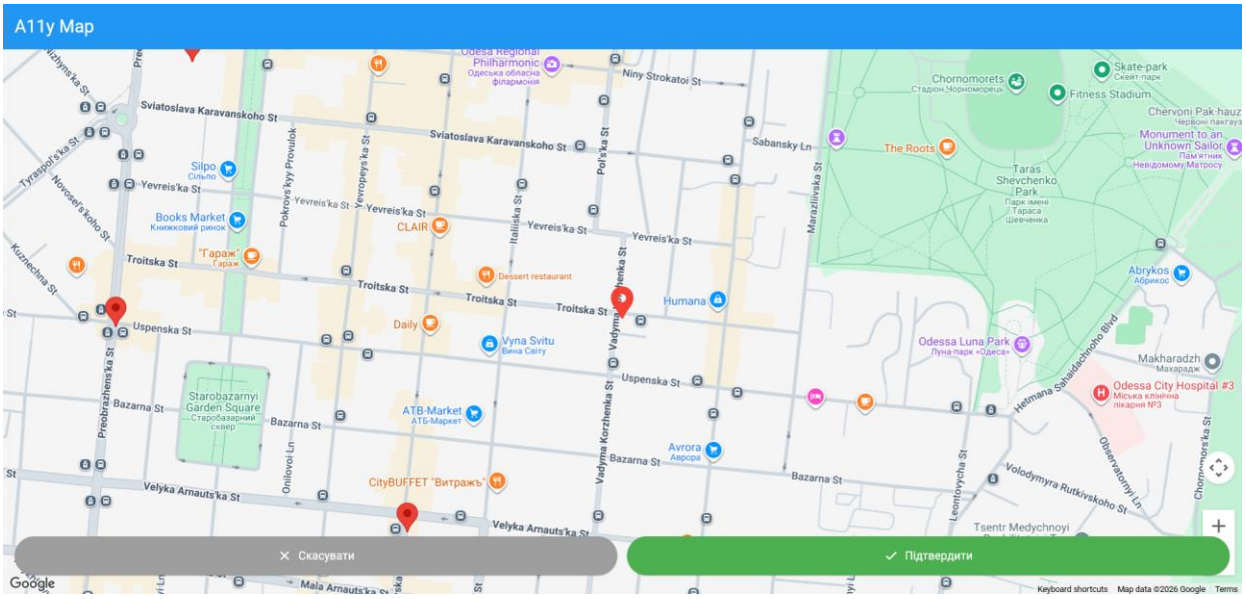

Рисунок 3.3 – Приклад інтерфейсу керування встановлення маркера

Лістинг 3.6 Визначення координат у режимі позиціонування:

```
void _onConfirmLocation() async {
 final LatLngBounds bounds =
    await _controller.getVisibleRegion();
 final center = LatLng(
   (bounds.northeast.latitude +
      bounds.southwest.latitude) / 2,
   (bounds.northeast.longitude +
      bounds.southwest.longitude) / 2,
 );
 setState(() {
   _pendingLocation = center;
   _isPickingLocation = false;
 });
 _showAddMarkerForm();
}
```

Такий підхід є точнішим за використання події onTap на карті, оскільки дозволяє користувачу спочатку точно спозиціонувати карту, а потім підтвердити координати – аналогічно до механізму замовлення таксі у популярних застосунках.

Після підтвердження координат відкривається форма у вигляді модального нижнього аркуша. Форма містить три поля: випадаючий список для вибору типу об’єкта, текстове поле для опису та кнопку прикріплення фотографії. Тип об’єкта обирається з двох варіантів – «Пандус / знижений бордюр» та «Перешкода» – що відповідають значенням ramp та obstacle у моделі даних:

Лістинг 3.7 Поле вибору типу об’єкта у формі додавання:

```
DropdownButtonFormField<String>(
```

```
 value: _selectedType,
 items: const [
  DropdownMenuItem(
     value: 'ramp',
     child: Text('Пандус / знижений бордюр')),
  DropdownMenuItem(
     value: 'obstacle',
     child: Text('Перешкода')),
 ],
 onChanged: (val) =>
    setState(() => _selectedType = val!),
 decoration: const InputDecoration(
    labelText: 'Тип об\'єкта'),
)
```

Завантаження фотографії реалізовано через пакет image_picker, що надає уніфікований інтерфейс для доступу до галереї пристрою як на Android, так і у веб-середовищі. При натисканні кнопки «Додати фото» викликається метод _pickImage().

Лістинг 3.8 Вибір фотографії з галереї пристрою:

```
Future<void> _pickImage() async {
 final picker = ImagePicker();
 final XFile? image = await picker.pickImage(
   source: ImageSource.gallery,
   maxWidth: 1024,
   maxHeight: 1024,
   imageQuality: 85,
 );
 if (image != null) {
```

```
    setState(() => _selectedImage = image);
  }
 }
```

Параметри maxWidth, maxHeight та imageQuality виконують стиснення зображення на стороні клієнта перед завантаженням. Обмеження у 1024 пікселі та якість 85% забезпечують достатню деталізацію для ідентифікації об’єкта при розмірі файлу до 200–300 кілобайт. Це суттєво скорочує час завантаження та обсяг використаного сховища Firebase Storage.

Після заповнення форми користувач підтверджує створення мітки. Метод _submitMarker() виконує послідовно два хмарні запити – спочатку завантаження фото, потім запис документа до Firestore.

Лістинг 3.9 Завантаження фото та запис документа до Firestore:

```
Future<void> _submitMarker() async {
  String? photoUrl;
  if (_selectedImage != null) {
    final ref = FirebaseStorage.instance
        .ref()
        .child('reports/${const Uuid().v4()}.jpg');
    await ref.putFile(File(_selectedImage!.path));
    photoUrl = await ref.getDownloadURL();
  }
  await FirebaseFirestore.instance
      .collection('reports')
      .add({
    'lat': _pendingLocation!.latitude,
    'lng': _pendingLocation!.longitude,
    'type': _selectedType,
    'description': _descriptionController.text,
```

```
    'photo_url': photoUrl,
    'confidence_score': 0.5,
    'created_at': FieldValue.serverTimestamp(),
  });
  _loadMarkers();
  Navigator.pop(context);
}
```

Для іменування файлів у Firebase Storage використовується пакет uuid, що генерує унікальний ідентифікатор UUID v4. Це виключає конфлікти імен при одночасному завантаженні файлів від різних користувачів. Поле created_at заповнюється значенням FieldValue.serverTimestamp() – серверним часом Firestore – що гарантує коректну часову мітку незалежно від налаштувань годинника пристрою користувача. Після успішного запису викликається _loadMarkers() для оновлення шару маркерів на карті та Navigator.pop(context) для закриття форми.

3.5 Реалізація картки деталей мітки

Перегляд детальної інформації про мітку є завершальним елементом основного користувацького сценарію. При натисканні на маркер відкривається модальний нижній аркуш з фотографією, категорією та описом об'єкта. Вибір компонента ModalBottomSheet замість окремого екрана обумовлений UX-міркуваннями: користувач не втрачає контекст карти, аркуш закривається свайпом донизу, а перехід відбувається миттєво без анімації навігації.

Структура картки складається з трьох вертикальних блоків. Верхній блок займає фотографія об'єкта, що завантажується з Firebase Storage через віджет Image.network за збереженим у документі URL. Під час завантаження

зображення відображається індикатор прогресу, а у разі помилки мережі – іконка попередження:

Лістинг 3.10 Обробка завантаження та помилок зображення у картці деталей:

```
Image.network(
  data['photo_url'],
  height: 180,
  width: double.infinity,
  fit: BoxFit.cover,
  loadingBuilder: (context, child, progress) {
    if (progress == null) return child;
    return const SizedBox(
      height: 180,
      child: Center(child: CircularProgressIndicator()),
    );
  },
  errorBuilder: (context, error, stackTrace) =>
      const SizedBox(
    height: 180,
    child: Center(child: Icon(Icons.broken_image, size: 48)),
  ),
)
```

Використання loadingBuilder та errorBuilder є важливим для користувацького досвіду в умовах нестабільного мобільного інтернету: застосунок коректно обробляє повільне завантаження та обриви з'єднання замість відображення порожнього блоку.

Середній блок містить категорію об'єкта, оформлену кольоровим індикатором: зелений колір для пандусів та знижених бордюрів,

помаранчевий – для перешкод. Кольорове кодування дублює семантику іконок маркерів на карті та забезпечує миттєву ідентифікацію типу об’єкта. Нижній блок відображає текстовий опис, введений автором мітки, з обробкою порожнього значення.

Для міток без фотографії картка адаптується: замість блоку зображення відображається компактна заглушка з іконкою камери та підписом «Фото відсутнє». Це зберігає візуальну цілісність інтерфейсу та водночас стимулює користувачів додавати фотоматеріали до власних міток.

Зовнішній вигляд картки деталей для мітки з фотографією та без неї наведено на рисунку 3.4.

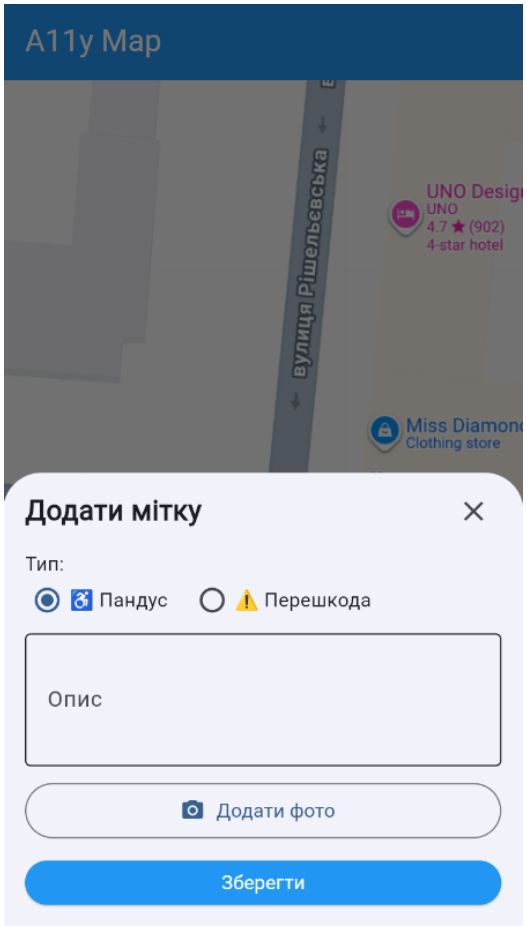

Рисунок 3.4 – Приклад зовнішнього вигляду картки деталей для мітки

3.6 Забезпечення кросплатформної роботи застосунку

Однією з ключових вимог до застосунку є коректна робота як на мобільній платформі Android, так і у веб-браузері з єдиної кодової бази. Flutter забезпечує таку можливість на рівні фреймворку, однак практична реалізація виявила низку платформних відмінностей, що потребували окремої обробки.

Першою відмінністю є механізм роботи з файлами при завантаженні фотографій. На платформі Android метод XFile.path повертає шлях до файлу у файловій системі пристрою, що дозволяє використовувати клас File для читання вмісту. У веб-середовищі файлова система недоступна з міркувань безпеки браузера, і XFile.path повертає тимчасовий blob URL. Для уніфікованої роботи на обох платформах застосовано метод readAsBytes(), що повертає вміст файлу у вигляді масиву байтів незалежно від платформи:

Лістинг 3.11 Кросплатформне завантаження зображення методом readAsBytes:

```
final bytes = await _selectedImage!.readAsBytes();
await ref.putData(
  bytes,
  SettableMetadata(contentType: 'image/jpeg'),
);
```

Явне зазначення contentType у метаданих є необхідним при використанні putData: без нього Firebase Storage зберігає файл з типом application/octet-stream, що порушує коректне відображення зображення у браузері.

Другою відмінністю є попередній перегляд вибраного фото у формі додавання мітки. Віджет Image.file недоступний на веб-платформі, а

Image.network з blob URL не працює на Android. Для розгалуження використано константу kIsWeb з бібліотеки flutter/foundation:

Лістинг 3.12 Розгалуження перегляду фото за допомогою kIsWeb:

```
child: kIsWeb
   ? Image.network(_selectedImage!.path,
      height: 160, fit: BoxFit.cover)
   : Image.file(File(_selectedImage!.path),
      height: 160, fit: BoxFit.cover),
```

Константа kIsWeb обчислюється під час компіляції, тому невикористана гілка коду виключається зі збірки відповідної платформи без впливу на розмір застосунку.

Третьою суттєвою відмінністю є механізм підключення Google Maps SDK. На Android картографічний рушій працює через нативний компонент MapView, а ключ API зчитується з маніфесту застосунку. На веб-платформі карта рендериться через Maps JavaScript API, скрипт якого підключається у файлі web/index.html:

Лістинг 3.13 Підключення Google Maps JavaScript API у web/index.html:

```
<script
src="https://maps.googleapis.com/maps/api/js?key=API_KEY"></script>
```

Це означає, що для двох платформ використовуються два окремі API-ключі з різними обмеженнями: ключ Android прив’язується до цифрового підпису застосунку та імені пакета, ключ Web – до доменного імені сайту. Такий розподіл є рекомендованою практикою безпеки Google Cloud Platform, оскільки компрометація одного ключа не впливає на інший.

Окремої уваги потребує відмінність у поведінці жестів керування картою. На мобільному пристрої масштабування виконується жестом щипка,

тоді як у браузері – коліщатком миші з затиснутою клавішею Ctrl або кнопками інтерфейсу. Пакет google_maps_flutter_web обробляє ці відмінності автоматично, однак при тестуванні було виявлено, що плаваюча кнопка додавання мітки на вузьких вікнах браузера перекривала елементи керування картою. Проблему вирішено адаптивним позиціонуванням кнопки з відступом, що обчислюється від фактичних розмірів вікна.

Тестування кросплатформності проводилось на фізичному пристрої Redmi Note 8 Pro під керуванням Android 11 та у браузері Google Chrome на операційній системі Windows. Обидві збірки створювались з однієї кодової бази командами flutter build apk та flutter build web відповідно без жодних змін у вихідному коді. Усі ключові сценарії – перегляд карти, додавання мітки з фотографією, перегляд картки деталей – функціонують ідентично на обох платформах.

Порівняння зовнішнього вигляду застосунку на Android та у веб-браузері наведено на рисунках 3.5 та 3.6.

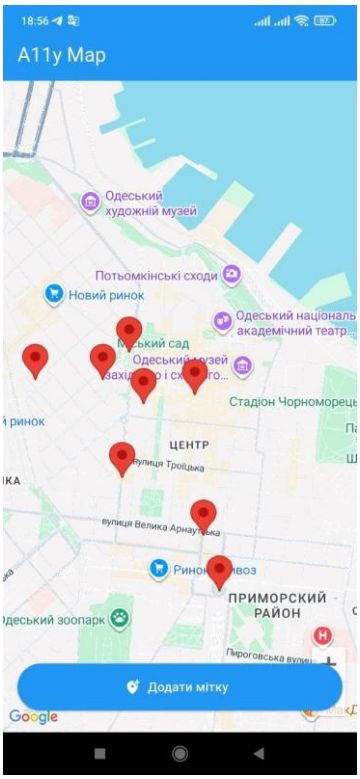

Рисунок 3.5 – Зовнішній вигляд застосунку на Android

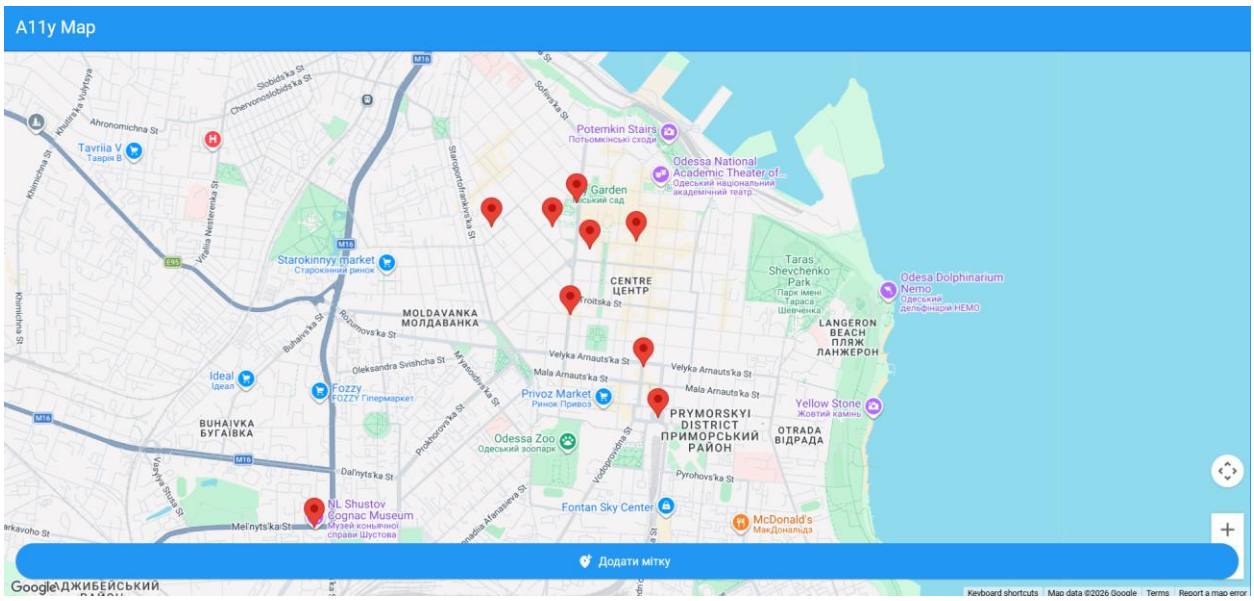

Рисунок 3.6 – Зовнішній вигляд застосунку у веб-браузері

3.7 Тестування функціоналу застосунку

Тестування реалізованого функціоналу проводилось методом ручного функціонального тестування за заздалегідь визначеними сценаріями. Цей підхід є доцільним для прототипу з обмеженим набором функцій, де автоматизація тестування не виправдовує витрат на її впровадження. Кожен сценарій перевірявся на обох цільових платформах – Android та Web.

Тестування охоплювало основні користувацькі сценарії, граничні випадки та обробку помилкових ситуацій. Перелік тест-кейсів та результати їх виконання наведено у таблиці 3.3.

Таблиця 3.3 – Результати функціонального тестування

№	Сценарій	Очікуваний результат	Android	Web
1	Запуск застосунку	Карта завантажується з маркерами існуючих міток	Пройдено	Пройдено

Продовження таблиці 3.3

№	Сценарій	Очікуваний результат	Android	Web
2	Додавання мітки з фото	Мітка з’являється на карті, фото у Storage, документ у Firestore	Пройдено	Пройдено
3	Додавання мітки без фото	Мітка створюється, photo_url = null, картка показує заглушку	Пройдено	Пройдено
4	Перегляд картки мітки	Bottom sheet відображає фото, тип та опис	Пройдено	Пройдено
5	Додавання мітки без опису	Мітка створюється з порожнім описом	Пройдено	Пройдено
6	Робота без інтернету	Застосунок не аварійно завершується, показує індикатор завантаження	Пройдено частково	Пройдено частково
7	Завантаження великого фото (>10 МБ)	Фото стискається до прийнятного розміру на клієнті	Пройдено	Пройдено
8	Швидке повторне натискання кнопки підтвердження	Створюється лише одна мітка	Не пройдено	Не пройдено

За результатами тестування виявлено дві проблеми, що потребують виправлення у наступних ітераціях розробки.

Перша проблема стосується поведінки застосунку за відсутності інтернет-з’єднання. При запуску без мережі застосунок не завершується аварійно, однак замість інформативного повідомлення користувач бачить нескінченний індикатор завантаження. Це пов’язано з тим, що запит до

Firestore не має явно встановленого таймауту, а вбудований механізм офлайн-кешування Firestore на веб-платформі за замовчуванням вимкнений. Запропоноване рішення – обгортання запиту в timeout() з відображенням повідомлення про відсутність з'єднання та кнопкою повторної спроби:

Лістинг 3.14 Обмеження часу очікування запиту до Firestore:

```
final snapshot = await FirebaseFirestore.instance
    .collection('reports')
    .get()
    .timeout(const Duration(seconds: 10),
        onTimeout: () => throw TimeoutException(
            'Немає з\'єднання з сервером'));
```

Друга проблема – створення дублікатів мітки при швидкому повторному натисканні кнопки підтвердження форми. Між натисканням і завершенням запису до Firestore проходить від однієї до трьох секунд залежно від швидкості з'єднання, і протягом цього часу кнопка залишається активною. Користувач, не бачачи миттєвої реакції, натискає повторно, що призводить до створення двох ідентичних документів. Стандартне рішення – блокування кнопки на час виконання запиту через прапорець стану:

Лістинг 3.15 Блокування повторного надсилання форми:

```
bool _isSubmitting = false;

Future<void> _submitMarker() async {
  if (_isSubmitting) return;
  setState(() => _isSubmitting = true);
  try {
    // ... завантаження фото та запис до Firestore
  } finally {
```

```
      setState(() => _isSubmitting = false);
    }
  }
```

Конструкція try-finally гарантує розблокування кнопки навіть у разі помилки запиту, що запобігає блокуванню інтерфейсу. Додатково кнопка під час виконання запиту відображає індикатор прогресу замість тексту, надаючи користувачу візуальний зворотний зв’язок.

Окрім функціонального тестування, виконано перевірку коректності даних на стороні хмарної інфраструктури. Через консоль Firebase підтверджено, що документи колекції reports містять усі обов’язкові поля з коректними типами, фотографії у Storage мають правильний contentType та доступні за згенерованими URL, а часові мітки created_at відповідають серверному часу. Знімок консолі Firebase з тестовими записами наведено на рисунку 3.7.

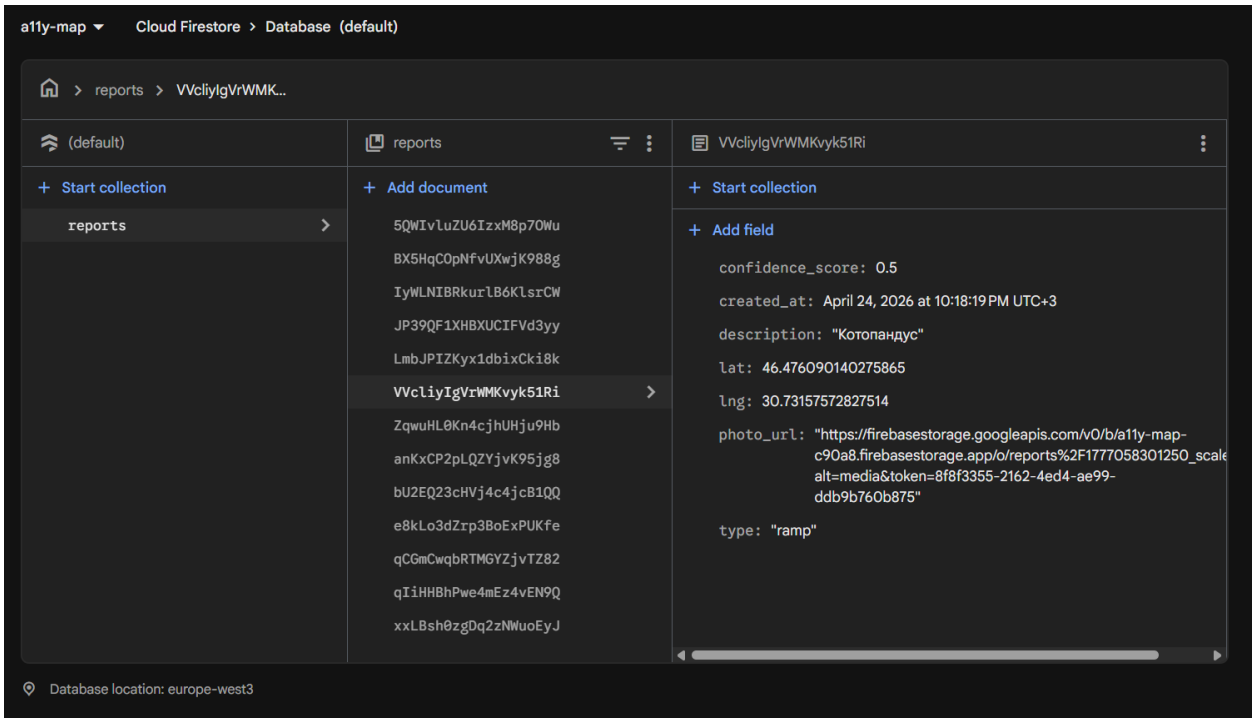

Рисунок 3.7 – Знімок консолі Firebase з тестовими записами

Загальний результат тестування підтверджує працездатність усіх ключових сценаріїв застосунку на обох цільових платформах. Виявлені проблеми мають відомі стандартні рішення та заплановані до виправлення, що відображено у підрозділі про перспективи розвитку.

3.8 Експериментальна перевірка моделі валідації фотоматеріалів

Для підтвердження працездатності гібридної моделі валідації фотоматеріалів, спроєктованої у підрозділі 2.6, було виконано її експериментальну перевірку на реальному наборі зображень. Метою перевірки було визначення архітектури згорткової нейронної мережі з найкращими показниками детектування елементів пішохідної інфраструктури та оцінювання впливу механізму Human-in-the-Loop на якість класифікації.

Перевірку проведено на наборі даних Sidewalk Accessibility v3.0, що містить 2603 розмічених зображення елементів міської пішохідної інфраструктури – пандусів, знижених бордюрів, сходів та перешкод. Набір даних поділено на навчальну, валідаційну та тестову вибірки у співвідношенні 70/20/10. Зображення мають різну роздільність, умови освітлення та ракурси зйомки, що наближає умови перевірки до реальної експлуатації застосунку, де фотографії завантажуються користувачами з різних пристроїв.

Для порівняння обрано три архітектури, що представляють різні підходи до детектування об’єктів [32]: YOLOv8n – полегшена версія для пристроїв з обмеженими ресурсами, YOLOv8m – збалансована версія середнього розміру [19], та ResNet-50 – класична згорткова мережа з залишковими з’єднаннями [20]. Усі моделі навчались на однаковій вибірці протягом однакової кількості епох з ідентичними параметрами аугментації даних для забезпечення коректності порівняння.

Якість детектування оцінювалась за трьома метриками; методологія порівняльного оцінювання нейромережевих моделей на спеціалізованих наборах даних застосована за аналогією з [34]. Precision (точність) відображає частку правильно класифікованих об’єктів серед усіх, які модель віднесла до певної категорії. Recall (повнота) показує частку виявлених об’єктів серед усіх наявних на зображеннях. Метрика mAP@50 (mean Average Precision при пороговому значенні IoU 0,5) є інтегральним показником якості детектування, що враховує як правильність класифікації, так і точність локалізації об’єкта на зображенні.

Результати порівняння трьох архітектур наведено у таблиці 3.4.

Таблиця 3.4 – Порівняння архітектур на наборі Sidewalk Accessibility

Модель	**mAP@50**	**Precision**	**Recall**
YOLOv8n	0,381	0,471	0,398
ResNet-50	0,412	0,503	0,421
YOLOv8m	0,446	0,538	0,452

Як видно з таблиці 3.4, найкращі показники за всіма трьома метриками продемонструвала модель YOLOv8m: mAP@50 = 0,446, Precision = 0,538, Recall = 0,452. Полегшена версія YOLOv8n очікувано показала найнижчі результати через меншу кількість параметрів, а класична архітектура ResNet-50 зайняла проміжне положення. З огляду на це для подальшої роботи обрано YOLOv8m як модель з оптимальним балансом точності та обчислювальної складності. Практика застосування згорткових нейронних мереж для класифікації зображень та відеоданих у задачах різної предметної спрямованості описана у [27, 28, 31]

Для перевірки того, на які саме ділянки зображення спирається мережа при класифікації, застосовано метод візуалізації Grad-CAM, що будує теплову карту значущості пікселів. Аналіз карт показав, що при правильній класифікації пандуса мережа концентрує увагу саме на похилій площині та

зоні переходу між тротуаром і проїжджою частиною, а не на сторонніх елементах фону. Це підтверджує, що модель навчилась релевантним ознакам об’єкта. Приклад теплової карти Grad-CAM наведено на рисунку 3.6.видно з таблиці 3.4, найкращі показники за всіма трьома метриками продемонструвала модель YOLOv8m: mAP@50 = 0,446, Precision = 0,538, Recall = 0,452. Полегшена версія YOLOv8n очікувано показала найнижчі результати через меншу кількість параметрів, а класична архітектура ResNet-50 зайняла проміжне положення. З огляду на це для подальшої роботи обрано YOLOv8m як модель з оптимальним балансом точності та обчислювальної складності. Порівняння показників трьох архітектур наведено на рисунку 3.8.

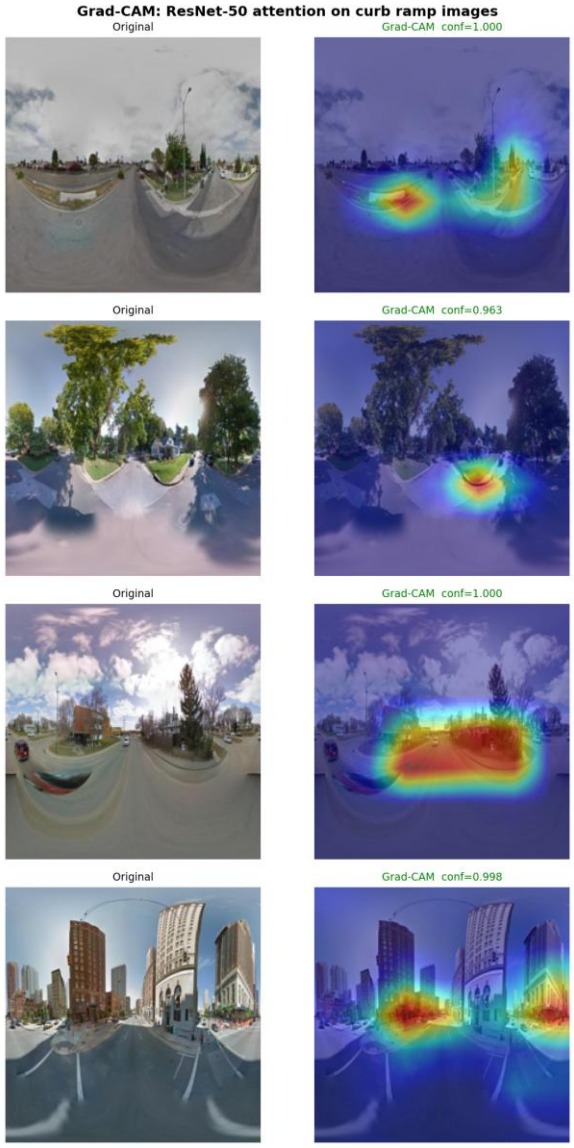

Рисунок 3.8 – Зони уваги ResNet-50 при класифікації зображень пандусів

Після вибору базової моделі перевірено роботу гібридного HITL-пайплайну. Порогове значення впевненості встановлено рівним 0,5: фотографії, для яких модель повертає впевненість вище за цей поріг, обробляються автоматично, решта скеровуються на ручну перевірку. На валідаційній вибірці з 217 зображень отримано такий розподіл: 144 зображення (66,4%) пройшли автоматичну валідацію зі значенням впевненості 0,5 і вище, а 73 зображення (33,6%) було скеровано на перевірку модератору. Середнє значення впевненості становило 0,577, медіана – 0,640. Розподіл значень впевненості наведено на рисунку 3.9.

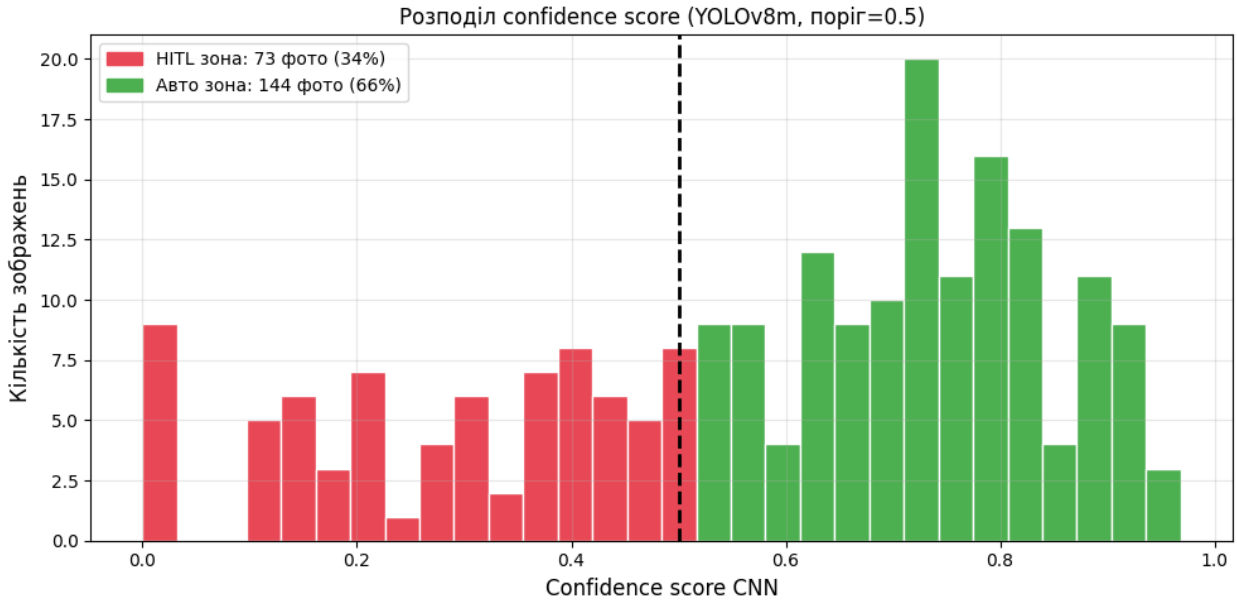

Рисунок 3.9 – Розподіл значень впевненості YOLOv8m при порозі $\tau = 0,5$

Аналіз зображень, що потрапили до зони ручної перевірки, показав, що це переважно панорамні знімки з широким кутом огляду, фотографії за нестандартного освітлення (пряме сонячне світло, глибокі тіні) та зображення з частковим перекриттям об’єкта. Це підтверджує, що модель коректно ідентифікує складні випадки та потребує участі людини саме там, де автоматичне рішення є найменш надійним. Приклади зображень із зони ручної перевірки наведено на рисунку 3.10.

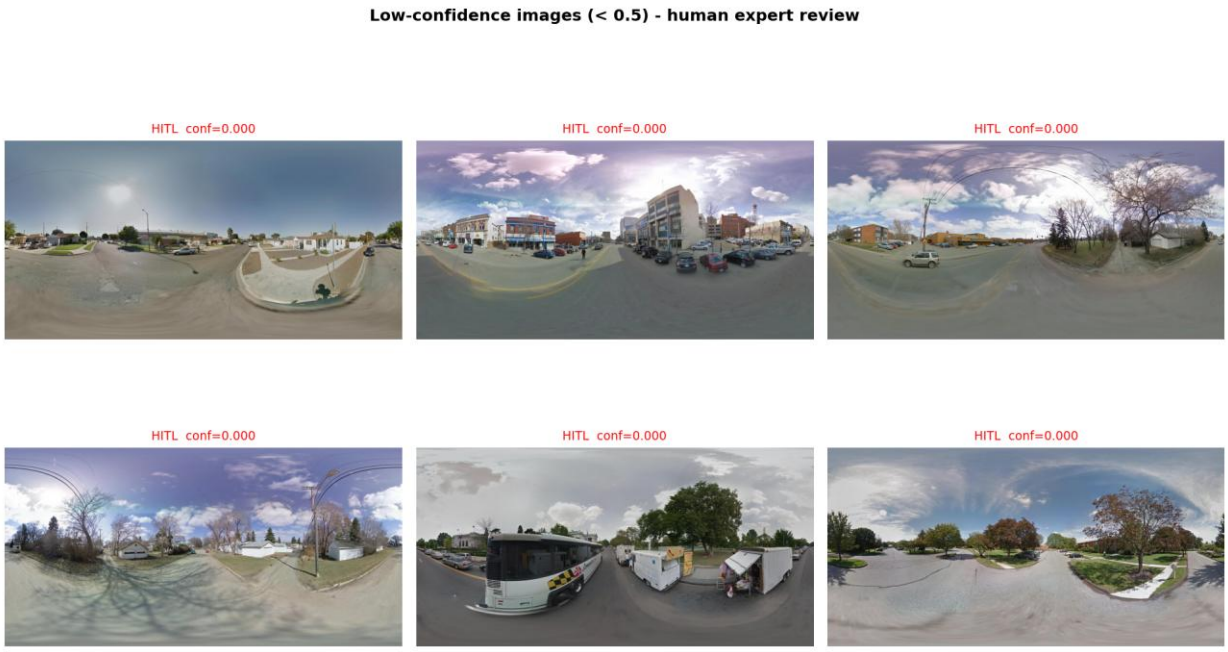

Рисунок 3.10 – Приклади зображень з низькою впевненістю (зона HITL, conf < 0,5)

Отриманий розподіл підтверджує практичну цінність гібридного підходу: дві третини звітів обробляються без участі людини, що суттєво знижує навантаження на модератора порівняно з повністю ручною перевіркою, водночас зберігаючи контроль над складними випадками.

Для перевірки ефективності механізму активного навчання модель YOLOv8m донавчено на 73 зразках, що були верифіковані експертом у зоні ручної перевірки. Результати до та після донавчання наведено у таблиці 3.5 та на рисунку 3.11.

Таблиця 3.5 – Вплив активного навчання на показники моделі YOLOv8m

Метрика	**До донавчання**	**Після донавчання**	**Зміна**
mAP@50	0,446	0,478	+7,2%
Precision	0,538	0,596	+10,8%
Recall	0,452	0,467	+3,3%

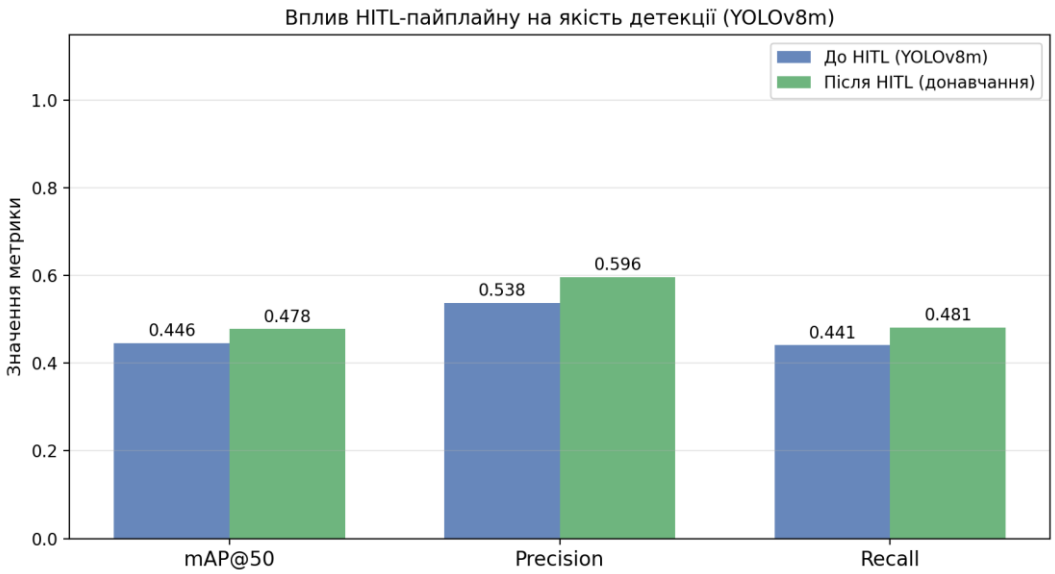

Рисунок 3.11 – Порівняння метрик YOLOv8m до та після HITL-пайплайну

Донавчання на верифікованих зразках підвищило mAP@50 на 7,2%, а Precision – на 10,8%. Зростання Precision є особливо важливим для задачі валідації, оскільки означає зменшення частки хибно прийнятих звітів. Отримані результати підтверджують працездатність концепції активного навчання: звіти, що пройшли ручну перевірку, не лише валідуються, а й слугують джерелом для поступового вдосконалення моделі, замикаючи цикл самонавчання системи. Питання оцінювання значущості ознак для підвищення продуктивності класифікації зображень висвітлено у [40].

Таким чином, перевірка підтвердила обґрунтованість вибору архітектури YOLOv8m та практичну ефективність гібридного підходу до валідації фотоматеріалів. Запропонована модель забезпечує автоматичну обробку більшості звітів за прийнятної точності та має механізм безперервного покращення якості класифікації.

3.9 Безпека та обмеження поточної реалізації

Безпека хмарної інфраструктури застосунку забезпечується механізмом правил безпеки Firebase (Security Rules), що визначають умови доступу до

даних Firestore та файлів Storage. Правила виконуються на стороні серверів Google до обробки кожного запиту, що унеможливлює їх обхід з боку клієнта.

На етапі розробки проєкт було створено у тестовому режимі (test mode), що надає повний доступ на читання та запис будь-якому клієнту протягом тридцяти днів. Такий режим є прийнятним виключно для відлагодження, оскільки відкрита база даних може бути використана зловмисниками для запису довільних даних, видалення існуючих записів або вичерпання квот безкоштовного тарифного плану. Перед публічним розгортанням застосунку правила було замінено на обмежувальні:

Лістинг 3.16 Правила безпеки Firestore для колекції reports:

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    match /reports/{reportId} {
      allow read: if true;
      allow create: if request.resource.data.keys()
          .hasAll(['lat', 'lng', 'type', 'created_at'])
        && request.resource.data.lat is number
        && request.resource.data.lng is number
        && request.resource.data.type in ['ramp', 'obstacle'];
      allow update, delete: if false;
    }
  }
}
```

Наведені правила реалізують такі обмеження. Читання колекції reports дозволено всім – це відповідає публічній природі краудсорсингової карти. Створення нового документа дозволено лише за умови наявності

обов’язкових полів та коректності їх типів: координати мають бути числами, а тип – одним з двох допустимих значень. Оновлення та видалення заборонено повністю – це захищає існуючі дані від вандалізму до впровадження системи авторизації та модерації.

Аналогічні правила застосовано до Firebase Storage: завантаження файлів обмежено типом вмісту image/* та максимальним розміром у 5 мегабайт, що запобігає використанню сховища для зберігання сторонніх даних.

Окремим аспектом безпеки є захист API-ключів Google Maps. Ключі неминуче присутні у клієнтському коді – у маніфесті Android-застосунку та у HTML-файлі веб-версії – тому їх неможливо приховати повністю. Захист реалізовано через обмеження на стороні Google Cloud Console: ключ Android прив’язано до відбитка SHA-1 підпису застосунку та імені пакета, ключ Web – до конкретного доменного імені. Запити з інших застосунків або доменів відхиляються незалежно від наявності ключа. Додатково налаштовано бюджетне сповіщення, що повідомляє про наближення витрат до межі безкоштовного кредиту у 200 доларів США на місяць.

Поточна реалізація має низку обмежень, усвідомлення яких є важливим для планування подальшого розвитку.

Перше обмеження – відсутність авторизації користувачів. Усі мітки створюються анонімно, що унеможливлює модерацію на рівні автора, обмеження частоти створення міток одним користувачем та побудову репутаційної системи. Це також означає, що поле crowdconf моделі валідації, описаної у підрозділі 2.4, не може бути обчислене коректно, оскільки неможливо відрізнити підтвердження від різних користувачів.

Друге обмеження – відсутність серверної валідації вмісту. Правила безпеки Firestore перевіряють структуру та типи даних, але не їх семантику: координати можуть вказувати на середину океану, опис може містити нерелевантний текст, а фотографія – сторонній об’єкт. Реалізація моделей валідації з розділу 2 – confidence score, дедуплікація DBSCAN та CNN-

класифікація фото – потребує серверного компонента, наприклад Cloud Functions, що виходить за межі поточного етапу розробки.

Третє обмеження – ліміти безкоштовного тарифного плану Spark: 50 000 читань та 20 000 записів Firestore на добу, 5 гігабайт сховища та 1 гігабайт вихідного трафіку Storage. Поточна реалізація завантажує всю колекцію при кожному запуску застосунку, тому кожен запуск споживає кількість читань, що дорівнює кількості міток у базі. При базі у 500 міток ліміт вичерпується після ста запусків на добу. Це обмеження безпосередньо мотивує впровадження просторової фільтрації, описаної у наступному підрозділі.

3.10 Перспективи розвитку застосунку

Реалізований прототип забезпечує базовий функціонал краудсорсингового збору даних про доступність, однак для перетворення його на повноцінний продукт необхідна реалізація низки додаткових компонентів. Перспективи розвитку структуровано за пріоритетом впровадження.

Першочерговим напрямом є впровадження авторизації користувачів через Firebase Authentication. Сервіс підтримує автентифікацію через обліковий запис Google, електронну пошту та анонімні сесії з можливістю подальшого зв’язування. Авторизація розблоковує ланцюг залежних можливостей: прив’язку міток до автора через поле user_id, обмеження частоти створення міток, репутаційну систему користувачів та коректне обчислення складової crowdconf моделі валідації. Технічна складність впровадження є низькою – пакет firebase_auth інтегрується з існуючою інфраструктурою без змін у моделі даних, потребуючи лише додавання екрана входу та оновлення правил безпеки.

Другим напрямом є просторова фільтрація запитів через бібліотеку GeoFlutterFire. Поточне завантаження всієї колекції при запуску не масштабується: і за споживанням квот Firestore, і за обсягом трафіку, і за часом відгуку. GeoFlutterFire реалізує геохешування – кодування координат у рядковий префікс [37], що дозволяє запитувати лише документи у радіусі видимої області карти. Впровадження потребує додавання поля geohash до документів колекції та переписування методу _loadMarkers() на запит за межами поточного вікна камери з повторним запитом при суттєвому переміщенні карти.

Третім напрямом є реалізація серверних компонентів валідації через Cloud Functions – безсерверні функції, що виконуються у відповідь на події Firestore. Тригер onCreate колекції reports дає змогу автоматично обчислювати confidence score нового звіту, виконувати пошук просторових дублікатів у радіусі 20 метрів та, у перспективі, надсилати фотографію на класифікацію до розгорнутої моделі YOLOv8m. У перспективі текстові описи міток можуть опрацьовуватись великими мовними моделями для автоматичної категоризації та виявлення суперечностей; підходи до комбінованого використання таких моделей розглянуті у [38]. Така архітектура переносить обчислювальне навантаження з клієнта на хмару та реалізує моделі, спроєктовані у розділі 2, без зміни клієнтського коду. Підходи до побудови гібридних інтелектуальних систем, що поєднують декілька моделей в одному конвеєрі обробки, розглянуті у [29].

Четвертим напрямом є реалтайм-оновлення карти через механізм snapshot listeners Firestore. Поточна реалізація завантажує мітки одноразово при запуску, тому мітки, створені іншими користувачами під час сесії, не з’являються на карті до перезапуску застосунку. Заміна одноразового запиту get() на підписку snapshots() забезпечує автоматичне оновлення шару маркерів при кожній зміні колекції. У поєднанні з просторовою фільтрацією це створює ефект живої карти, де нові мітки з’являються у всіх користувачів

протягом секунд після створення. Питання моніторингу та діагностики стану систем у режимі реального часу розглянуті у [33]

П’ятим напрямом є маршрутизація з урахуванням даних про доступність – функція, що становить кінцеву мету розвитку застосунку. Ідея полягає у модифікації вагової функції графа пішохідних маршрутів: ділянки з підтвердженими перешкодами отримують підвищену вагу або виключаються з графа, а ділянки з верифікованими пандусами – знижену. Реалізація можлива на основі Google Directions API з постобробкою маршруту або через власний роутинг на даних OpenStreetMap з бібліотекою GraphHopper. Це найскладніший з напрямів, що потребує як значного обсягу накопичених верифікованих даних, так і серверної інфраструктури маршрутизації.

Шостим напрямом є концепція «стежок доступності» (accessibility trails), описана у підрозділі 2.6 – пасивний збір анонімізованих GPS-треків переміщень користувачів за їх згодою. Агрегація треків дозволяє виявляти ділянки, які користувачі систематично оминають, та формувати гіпотези про наявність незадокументованих бар’єрів. Цей напрям потребує особливої уваги до приватності: треки мають збиратися виключно за явною згодою, анонімізуватися на пристрої та агрегуватися без можливості ідентифікації окремого користувача.

Послідовність впровадження зазначених напрямів сформована за принципом нарощування цінності: авторизація та просторова фільтрація усувають поточні технічні обмеження, серверна валідація та реалтайм підвищують якість і актуальність даних, а маршрутизація та стежки доступності перетворюють накопичені дані на безпосередню практичну цінність для кінцевого користувача.

ВИСНОВКИ

У рамках кваліфікаційної роботи був розроблений і реалізований кросплатформний застосунок для краудсорсингового збору даних про проблеми доступності міського середовища на основі фреймворку Flutter, хмарної платформи Firebase та картографічного сервісу Google Maps. Актуальність задачі обумовлена зростанням кількості людей з порушеннями мобільності в Україні внаслідок повномасштабного вторгнення та потребою у цифрових інструментах моніторингу безбар'єрності в умовах повоєнної відбудови міст.

У ході виконання роботи отримано такі результати.

Проведено аналіз стану доступності міського середовища в Україні. Розглянуто краудсорсинг та волонтерську географічну інформацію як методологічну основу збору даних, визначено їх переваги для задач моніторингу міського середовища та характерні обмеження, що потребують компенсації на рівні проєктування системи.

Виконано порівняльний аналіз існуючих застосунків картування доступності – Wheelmaps, AXS Map, Google Maps Accessible Places, Accessnow та Route4U. Виявлено спільну функціональну прогалину: всі розглянуті рішення оцінюють доступність зареєстрованих закладів і не дозволяють документувати довільні елементи вуличної інфраструктури – тротуари, перехрестя, бордюрні зниження, що є першочерговими бар'єрами для користувачів крісел колісних.

Спроєктовано трирівневу архітектуру системи, побудовано UML-діаграми варіантів використання, послідовності та активності, розроблено модель даних на основі документної колекції Firestore.

Розроблено три взаємодоповнюючі моделі забезпечення якості краудсорсингових даних: модель валідації звітів на основі агрегованого показника confidence score з дедуплікацією методом DBSCAN; просторову модель пріоритизації зон збору даних на гексагональній сітці H3 з

динамічним індексом пріоритету; гібридну модель валідації фотоматеріалів, що поєднує класифікацію згортковою нейронною мережею YOLOv8m з механізмом Human-in-the-Loop. Експериментальна перевірка останньої на наборі данних Sidewalk Accessibility v3.0 продемонструвала підвищення показника mAP@50 на 7,2% та Precision на 10,8% після донавчання моделі на верифікованих експертом зразках.

Реалізовано функціональний прототип застосунку, що забезпечує відображення інтерактивної карти з кастомними маркерами, розміщення геоміток у довільних точках простору з категоризацією та текстовим описом, прикріплення фотографій із завантаженням до Firebase Storage та перегляд детальної картки мітки. Забезпечено кросплатформну роботу застосунку з єдиної кодової бази, що підтверджено тестуванням на пристрої Redmi Note 8 Pro під керуванням Android та у браузері Google Chrome. Функціональне тестування за вісьмома сценаріями підтвердило працездатність усіх ключових функцій; для двох виявлених проблем запропоновано рішення.

Визначено перспективи розвитку застосунку: впровадження авторизації користувачів, серверної валідації через Cloud Functions, оновлення карти у реальному часі, маршрутизації з урахуванням даних про доступність та концепції стежок доступності на основі агрегованих GPS-треків.

Результати роботи апробовано у вигляді 3 тез доповідей під час II Міжнародної науково-практичної студентської конференції «ІТ-простір сьогодення: тенденції, інновації та перспективи розвитку» (м. Харків, ХНУ імені В.Н. Каразіна, 15 жовтня 2025 р.) [42], XVI Всеукраїнської конференції молодих вчених «Молоді вчені 2026 – від теорії до практики» (м. Дніпро, УДУНТ, 18 березня 2026 р.) [43], XXX Міжнародного молодіжного форуму «Радіоелектроніка та молодь у XXI столітті» (м. Харків, ХНУРЕ, 2026 р.) [44] та статті у міжнародному науковому журналі «Grail of Science» (№ 68, 2026) [45].

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Державні будівельні норми ДБН В.2.2-17:2006. Доступність будинків і споруд для маломобільних груп населення. Київ: Мінбуд України, 2007. 21 с.

2. Міністерство соціальної політики України. Статистична інформація щодо осіб з інвалідністю. URL: https://www.msp.gov.ua (дата звернення 14.04.2026).

3. United Nations. (2023). Report on the human rights situation in Ukraine. UN Human Rights Monitoring Mission in Ukraine. URL: https://www.ohchr.org/en/countries/ukraine (дата звернення 14.04.2026).

4. Кабінет Міністрів України. (2023). Національна програма відбудови України. URL: https://www.kmu.gov.ua (дата звернення 14.04.2026).

5. Конвенція про права осіб з інвалідністю: Конвенція ООН від 13.12.2006, ратифікована Законом України № 1767-VI від 16.12.2009. URL: https://zakon.rada.gov.ua/laws/show/995_g71 (дата звернення 14.04.2026).

6. Goodchild, M. F. (2007). Citizens as sensors: the world of volunteered geography. GeoJournal, 69(4), pp. 211–221.

7. OpenStreetMap Foundation. OpenStreetMap statistics. URL: https://www.openstreetmap.org/stats/data_stats.html (дата звернення 14.04.2026).

8. Haklay, M. (2010). How good is volunteered geographical information? A comparative study of OpenStreetMap and Ordnance Survey datasets. Environment and Planning B: Planning and Design, 37(4), pp. 682–703.

9. Heipke, C. (2010). Crowdsourcing geospatial data. ISPRS Journal of Photogrammetry and Remote Sensing, 65(6), pp. 550–557.

10. Zhang, G., & Zhu, A. X. (2018). The representativeness and spatial bias of volunteered geographic information: a review. Annals of GIS, 24(3), pp. 151–162.

11. Fonte, C. C., Antoniou, V., Bastin, L., Estima, J., Arsanjani, J. J., Bayas, J.-C. L., See, L., & Vatseva, R. (2017). Assessing VGI Data Quality. In Mapping and the Citizen Sensor. London: Ubiquity Press, pp. 137–163.

12. Wheelmap.org. URL: https://wheelmap.org/ (дата звернення 14.04.2026).

13. AXS Map. URL: https://www.axsmap.com/ (дата звернення 14.04.2026).

14. Google. Accessible places in Google Maps. URL: https://support.google.com/maps (дата звернення 14.04.2026).

15. Flutter documentation. URL: https://docs.flutter.dev (дата звернення 14.04.2026).

16. Firebase documentation. URL: https://firebase.google.com/docs (дата звернення 14.04.2026).

17. Google Maps Platform documentation. URL: https://developers.google.com/maps/documentation (дата звернення 14.04.2026).

18. Ester, M., Kriegel, H. P., Sander, J., & Xu, X. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96), pp. 226–231.

19. Jocher, G., Chaurasia, A., & Qiu, J. (2023). Ultralytics YOLOv8. URL: https://github.com/ultralytics/ultralytics (дата звернення 14.04.2026).

20. He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 770–778.

21. Selvaraju, R. R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., & Batra, D. (2017). Grad-CAM: visual explanations from deep networks via gradient-based localization. Proceedings of the IEEE International Conference on Computer Vision (ICCV), pp. 618–626.

22. Mosqueira-Rey, E., Hernández-Pereira, E., Alonso-Ríos, D., Bobes-Bascarán, J., & Fernández-Leal, Á. (2023). Human-in-the-loop machine learning: a state of the art. Artificial Intelligence Review, 56(4), pp. 3005–3054.

23. Кобилін, О.А., & Творошенко, І.С. (2021). Методи цифрової обробки зображень: навч. посібник. Харків: ХНУРЕ.

24. Путятін, Є.П., Гороховатський, В.О., & Матат, О.О. (2006). Методи та алгоритми комп'ютерного зору: навч. посібник. Харків: ХНУРЕ.

25. Kobylin O., Gorokhovatskyi V., Tvoroshenko I., and Peredrii O. (2020). The application of non-parametric statistics methods in image classifiers based on structural description components. Telecommunications and Radio Engineering, 79(10), pp. 855–863.

26. Gorokhovatskyi, V., Tvoroshenko, I., Kobylin, O., & Vlasenko, N. (2023). Search for visual objects by request in the form of a cluster representation for the structural image description. Advances in Electrical and Electronic Engineering, 21(1), pp. 19–27.

27. Tvoroshenko I., Gorokhovatskyi V., Kobylin O., and Tvoroshenko A. (2023). Application of deep learning methods for recognizing and classifying culinary dishes in images. International Journal of Academic and Applied Research, 7(9), pp. 57–70.

28. Tvoroshenko I., Pomazan V., Gorokhovatskyi V., and Kobylin O. (2023). Application of video data classification models using convolutional neural networks. International Journal of Academic and Applied Research, 7(11), pp. 134–145.

29. Bohdan N., Tvoroshenko I., Gorokhovatskyi V., and Kobylin O. (2025). Development of a hybrid method to enhance context memory for a chatbot application based on large language models. International Journal of Academic Information Systems Research, 9(10), pp. 7–18.

30. Gorokhovatskyi V., Tvoroshenko I., and Yakovleva O. (2024). Transforming image descriptions as a set of descriptors to construct classification

features. Indonesian Journal of Electrical Engineering and Computer Science, 33(1), pp. 113–125.

31. Pomazan, V., Tvoroshenko, I., & Gorokhovatskyi, V. (2023). Development of an application for recognizing emotions using convolutional neural networks. International Journal of Academic Information Systems Research, 7(7), pp. 25–36.

32. Кобилін, І. О., & Харченко, А. І. (2024). Classification techniques for computer vision. Системи обробки інформації, 3(178), pp. 33–41.

33. Кобилін, І. О., & Ніколайчук, А. І. (2024). Monitoring and diagnosing faults in online mode using time series data. Системи обробки інформації, 3(178), pp. 27–32.

34. Боденчук-Пастухов, Є. В., & Кобилін, І. О. (2026). Оцінка моделей трансформерів машинного перекладу на низько-ресурсних, азійсько-українських мовних парах. Системи обробки інформації, 1(184), pp. 116–123.

35. Bodyanskiy, Y., Vynokurova, O., Szymański, Z., Kobylin, I., & Kobylin, O. (2016). Adaptive robust models for identification of nonstationary systems in data stream mining tasks. 2016 IEEE First International Conference on Data Stream Mining & Processing (DSMP), pp. 263–268.

36. Bodyanskiy, Y., Vynokurova, O., Kobylin, I., & Kobylin, O. (2016). Adaptive fuzzy clustering of short time series with unevenly distributed observations in data stream mining tasks. Information Technology and Management Science, 19(1), pp. 23–28.

37. Gorokhovatskyi V., Tvoroshenko I., Yakovleva O., and Hudáková M. (2026). Feature space quantization and application of metrics in structural methods of image recognition. IEEE Access, vol. 14, pp. 17812–17824.

38. Suprun A., Tvoroshenko I., Gorokhovatskyi V., and Yakovleva O. (2025). Development and research of a method for the combined use of large language models for text generation. International Journal of Academic and Applied Research, 9(10), pp. 249–263.

39. Larin I., Tvoroshenko I., Gorokhovatskyi V., and Liubchenko V. (2025). Research and comparison of facial color normalization methods relative to an etalon image. International Journal of Academic and Applied Research, 9(11), pp. 36–47.

40. Гороховатський В.О., Творошенко І.С. (2025). Оцінювання значущості ознак для підвищення продуктивності структурних методів класифікації зображень. Проблеми інформатики та моделювання (ПІМ-2025): тези 25-ї міжнародної науково-технічної конференції (25–28 вересня 2025 р.). Харків: НТУ «ХПІ», С. 38–43.

41. Творошенко, І.С. (2021). Технології прийняття рішень в інформаційних системах: навч. посібник. Харків: ХНУРЕ.

42. Зіма, В. О. (2025). Оцінка якості і валідація краудсорс-репортів про проблеми доступності міської інфраструктури. ІТ-простір сьогодення: тенденції, інновації та перспективи розвитку: збірник тез доповідей ІІ Міжнародної науково-практичної студентської конференції (15 жовтня 2025 р., м. Харків). Харків: ХНУ імені В. Н. Каразіна, С. 208–211.

43. Зіма, В. О., & Кобилін, І. О. (2026). Системний аналіз просторового покриття та оптимізація краудсорсингового збору даних міської доступності. Молоді вчені 2026 – від теорії до практики: матеріали XVI Всеукраїнської конференції молодих вчених (18 березня 2026 р., м. Дніпро). Дніпро: Журфонд, С. 382–384.

44. Зіма, В. О. (2026). Мультимедійна система валідації об’єктів міської інфраструктури: порівняльний аналіз нейромережевих моделей та експертних оцінок. Радіоелектроніка та молодь у XXI столітті: матеріали XXX Міжнародного молодіжного форуму (м. Харків, 2026 р.). Харків: ХНУРЕ, Т. 7, С. 62–64.

45. Zima, V., Kobylin, I., & Putiatina, O. (2026). A hybrid system for validating crowd-sourced reports about ramps based on CNN and HITL. Grail of Science, (68), pp. 1038–1045.