Міністерство освіти і науки України
Кам’янець-Подільський національний університет імені Івана Огієнка
Фізико-математичний факультет
Кафедра комп’ютерних наук

Кваліфікаційна робота магістра

з теми: **«Інформаційна система моніторингу та фіксації міських проблем із передачею даних до органів місцевого самоврядування»**

Виконав: здобувач вищої освіти групи KN1-M24

спеціальності 122 Комп’ютерні науки

Середюк Валентин Васильович

(прізвище, ім’я та по батькові здобувача вищої освіти)

Керівник: Федорчук Володимир Анатолійович, професор, доктор технічних наук

(прізвище, ім’я та по батькові, науковий ступінь, вчене звання керівника)

Рецензент: Оптасюк Сергій Васильович, доцент, кандидат фізико-математичних наук

(прізвище, ім’я та по батькові, науковий ступінь, вчене звання рецензента)

м. Кам’янець-Подільський – 2025 р.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ІС – Інформаційна система
ОМС – Органи місцевого самоврядування
ПЗ – Програмне забезпечення
OS (ОС) – Operating System, операційна система
GPS – Global Positioning System, глобальна система позиції
SDK – Software Development Kit, набір для розробки програмного забезпечення
API – Application Program Interface, інтерфейс програмного застосунку
MVC – Model-View-Controller, модель-представлення-контролер
MVP – Model-View-Presenter, модель-представлення-представник
MVVM – Model-View-ViewModel, модель-представлення-модель представлення
HTTPS – Hypertext Transfer Protocol Secure, протокол безпечної передачі гіпертексту
TLS – Transport Layer Security, безпека транспортного рівня
JSON – JavaScript Object Notation, нотація об’єкта мовою javascript
REST – Representational State Transfer, передача репрезентативного стану
URL – Uniform Resource Locator, уніфікований локатор ресурсів
ACID – Atomicity, Consistency, Isolation, Durability
ADT – Android Development Tools, знаряддя для розробки Android-застосунків
JDK – Java Development Kit, комплект для розробки мовою програмування Java
IDE – Integrated Development Environment, інтегроване середовище розробки
JVM – Java Virtual Machine, віртуальна машина Java
БД – База даних

АНОТАЦІЯ

Тема роботи: Інформаційна система моніторингу та фіксації міських проблем із передачею даних до органів місцевого самоврядування.

Виконавець: Валентин Середюк, здобувач вищої освіти другого (магістерського) рівня 2 року навчання, спеціальність F3 Комп'ютерні науки.

Науковий керівник: Володимир Федорчук, професор, доктор технічних наук.

Кваліфікаційна робота магістра присвячена розробці плану інформаційної системи на базі мобільного додатку для фіксації порушень благоустрою та інфраструктурних проблем міста. Функціонал передбачає автоматизовану передачу заявок муніципальним органам та контроль за їх вирішенням. У ході дослідження проведено аналіз вимог, зацікавлених сторін та ризиків проєкту. Ключову увагу приділено вдосконаленню механізмів обробки даних, що дозволяє підвищити ефективність комунального менеджменту. Проєкт успішно розроблено; результати підтверджують ефективність запропонованого підходу щодо пришвидшення обробки звернень громадян. На поточному етапі проєкт є повністю розроблений, а результати вказують на перспективний підхід щодо скорочення часу обробки запитів.

Ключові слова: інформаційна система, мобільний застосунок, моніторинг та фіксація, міські проблеми, організація зворотного зв'язку.

ABSTRACT

Topic: Information system for monitoring and recording urban problems with data transfer to local government bodies.

Author: Valentin Seredyuk, second-year master's student, majoring in Computer Science (F3).

Scientific supervisor: Volodymyr Fedorchuk, professor, Doctor of Technical Sciences.

The master's thesis is devoted to the development of an information system plan based on a mobile application for recording violations of public order and infrastructure problems in the city. The functionality provides for the automated transfer of requests to municipal authorities and monitoring of their resolution. The study analyzed the requirements, stakeholders, and risks of the project. Key attention was paid to improving data processing mechanisms, which allows for increased efficiency in municipal management. The project was successfully developed; the results confirm the effectiveness of the proposed approach to speeding up the processing of citizens' requests. At the current stage, the project is fully developed, and the results indicate a promising approach to reducing the time required to process requests.

Keywords: information system, mobile application, monitoring and recording, urban problems, feedback organization.

ВСТУП

Сучасні міста мають багато інфраструктурних та соціальних проблем, ефективне розв’язання яких потребує швидкого й достовірного отримання даних. Традиційні способи звернення громадян є повільними й незручними, через що частина проблем лишається невчасно поміченою. Наукова проблема полягає у створенні сучасних засобів збору, структурування та передачі інформації, зокрема — розробці системи, що дозволить громадянам оперативно фіксувати недоліки міської інфраструктури, описувати їх і передавати дані відповідальним органам для швидкого реагування. Така система має бути зручною, надійною та сумісною з існуючими сервісами місцевої влади.

Об'єкт дослідження: програмне забезпечення для моніторингу та фіксації міських проблем.

Предмет дослідження: процеси розроблення, тестування та впровадження програмного забезпечення та аналізу взаємозв’язків між органами місцевої влади та жителями міста.

Мета: розробка інформаційної системи з інтегрованим мобільним застосунком для моніторингу та фіксації міських проблем із передачею даних до органів місцевого самоврядування та з можливістю контролю за усуненням проблем.

Завдання:

1. Проаналізувати існуючі інформаційні системи та сервіси, що використовуються для фіксації міських проблем, визначити їх переваги та недоліки.
2. Визначити вимоги до функціоналу інформаційної системи, включно з можливостями фотофіксації, геолокації, опису проблеми та передачею даних.
3. Розробити концептуальну архітектуру системи, визначити її основні компоненти, інформаційні потоки та способи інтеграції з органами місцевого самоврядування.

4. Створити модель бази даних, що забезпечуватиме зручне збереження, пошук та обробку інформації про зафіксовані міські проблеми.
5. Розробити програмний прототип інформаційної системи, реалізувавши основні функціональні можливості для користувачів та адміністраторів.
6. Здійснити тестування прототипу, проаналізувати результати його роботи та визначити можливості для подальшого вдосконалення.

Методи дослідження: у процесі розробки використовуються методи аналізу та систематизації вимог, моделювання бізнес-процесів, а також методи проєктування архітектури програмних систем, об’єктно-орієнтоване програмування, прототипування інтерфейсу користувача, методи тестування (модульне, інтеграційне та користувацьке).

Практичне значення одержаних результатів: розроблена система забезпечує оперативний збір та передачу достовірної інформації про міські проблеми, підвищивши ефективність взаємодії між громадянами та органами місцевого самоврядування. Її впровадження сприяє швидшому реагуванню на інфраструктурні недоліки, прозорості процесу обробки звернень та покращенню якості міського середовища.

Апробація результатів: тези на тему “Розробка інформаційної системи для моніторингу та фіксації міських проблем із передачею даних до органів місцевого самоврядування”

Структура роботи: вступ, 4 розділи, висновки, список використаних джерел (25) та додатків (1).

РОЗДІЛ 1 ПЕРЕДПРОЄКТНЕ ОБСТЕЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз предметної області

Інформаційна система (ІС) — це сукупність взаємопов’язаних програмних, технічних, організаційних та людських компонентів, призначених для збору, зберігання, обробки, передачі та представлення інформації користувачам для підтримки їхньої діяльності та прийняття рішень [1].

ІС можна класифікувати за різними критеріями, залежно від їхньої функції, масштабу чи способу використання.

Основні види ІС:

1. За призначенням і функціональністю:
    - Оперативні (транзакційні) системи — автоматизують рутинні операції, наприклад, системи банківських платежів, обліку продажів.
    - Інформаційно-аналітичні системи — допомагають аналізувати дані та підтримують прийняття рішень, наприклад, BI-системи, системи прогнозування.
    - Системи підтримки прийняття рішень (СППР) — спеціалізовані для прийняття управлінських рішень на основі моделювання і аналізу даних.
    - Експертні системи — використовують базу знань і правила логіки для надання рекомендацій у складних ситуаціях.
2. За рівнем охоплення:
    - Корпоративні — використовуються на рівні організації для інтеграції бізнес-процесів.
    - Локальні — призначені для окремих відділів або користувачів.
3. За галузевою спрямованістю:
    - Фінансові, освітні, медичні, транспортні, муніципальні та інші спеціалізовані ІС.
4. За типом взаємодії з користувачем:

- Мобільні — доступні через смартфони та планшети.
- Веб-застосунки — працюють через браузер і інтернет.

Мобільний застосунок є найкращим рішенням для системи моніторингу та фіксації міських проблем, оскільки смартфон завжди під рукою і дозволяє миттєво зробити фото, автоматично визначити геолокацію та одразу передати дані до органів місцевого самоврядування. Він забезпечує зручність для користувачів, швидку подачу звернень, можливість отримувати push-сповіщення про статус розгляду та навіть працювати офлайн. Завдяки цьому мобільний застосунок гарантує найвищу оперативність, точність і доступність взаємодії між громадянами та міськими службами.

Мобільний застосунок (додаток) — це програмне забезпечення для роботи на портативних пристроях, таких як смартфони та планшети. Він може виконувати різні функції: від спілкування та розваг до навчання, роботи й онлайн-покупок [2].

Вебзастосунок (додаток) — це програмне забезпечення, яке працює через веб браузер і доступне користувачам через інтернет чи локальну мережу. На відміну від звичайних програм, вебзастосунок не потребує встановлення на пристрій — достатньо перейти за веб адресою. Він може виконувати різноманітні функції: від обміну повідомленнями й управління даними до онлайн-магазинів, сервісів бронювання чи систем електронного документообігу [3].

Важливим функціоналом мобільного застосунку для опису міських проблем є можливість додавання фотофіксації, яка наочно продемонструє наявність проблем.

Фотофіксація — це процес створення фотографічних зображень з метою документування певного об’єкта, події, стану або ситуації. Її основна мета — забезпечити точну та надійну візуальну фіксацію фактів, яку можна використовувати як доказ, підтвердження або інформаційний

матеріал у різних сферах: журналістиці, науці, будівництві, правозастосуванні чи моніторингу міського середовища.

На сьогоднішній день, рішення, які існують на ринку не пропонують швидкої комунікації та вирішення міських проблем. Для повідомлення щодо проблеми доводиться або довго телефонувати на гарячі лінії, або йти особисто писати скарги. В результаті такі заявки часто губляться та залишаються без уваги, адже неможливо прослідкувати в якому статусі наразі знаходиться заявка щодо проблеми. Також варто зазначити, що через незручність комунікації жителя та органів міської влади, в більшості випадків люди залишають проблеми без уваги і не витрачають час на повідомлення про це. Простий процес комунікації зможе вирішити це питання і покращити стан міста.

Згідно зі щорічною доповіддю Уповноваженого Верховною Радою України з прав людини, за 2024 рік з Хмельниччини надійшло 4435 звернень до омбудсмена з питань порушення прав чи недотриманням процедур (даний підрахунок включає в себе недотримання стандартів комунікації). Інформація про кількість звернень жителів Хмельницької області до омбудсмена протягом 2022-2024 років наведено на рис.1.1.

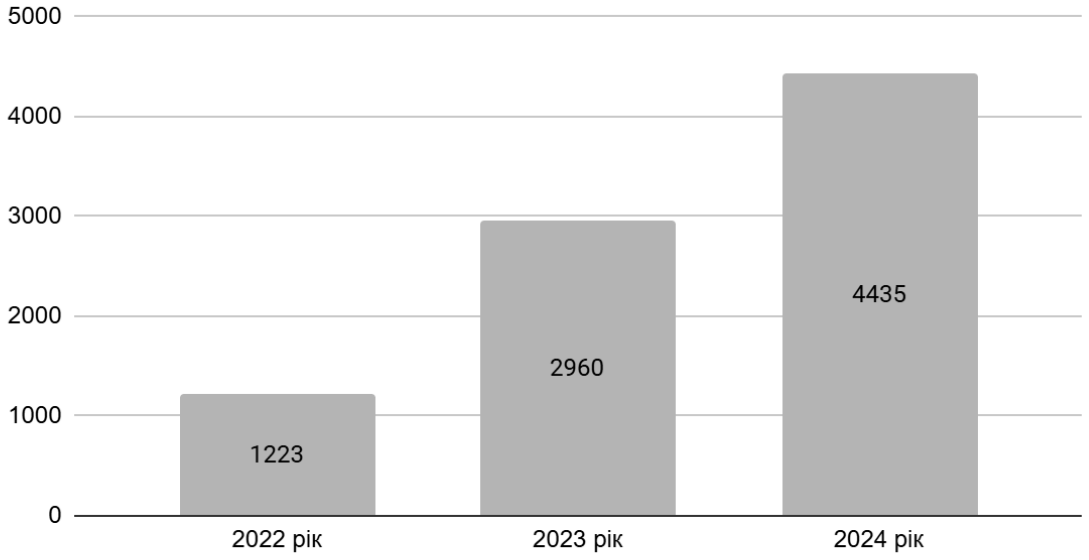

Рисунок 1.1 – Кількість звернень громадян [4]

З рис. 1.1 очевидним є значне щорічне збільшення кількості звернень, так до прикладу в 2024 році кількість звернень в 4 рази більша за

2022 рік. Що підкреслює значущість проблеми комунікації між громадянами та ОМС.

У листопаді 2025 року омбудсмен провів «безвиїзну перевірку» Служба місцевих доріг Хмельниччини — і виявив, що вона не повністю виконує вимоги закону щодо публічності та доступності інформації (наприклад, не публікує форми звернень, немає переліку послуг, не інформує про механізми оскарження) — тобто фактично порушується право людей на звернення / доступ до інформації.

Перевірки на кшталт стосовно Служби доріг показують, що інституції місцевого самоврядування або підпорядковані їм служби не завжди забезпечують право на звернення та доступ до інформації — і це цілком може бути підґрунтям для скарг.

Способом розв'язання цієї проблеми є розробка простору, де громадяни можуть залишити опис проблеми в місті, додавати фотофіксацію та контролювати статус своєї заявки, а ОМС зможуть отримувати швидко інформацію щодо проблем та їх вирішити.

Отже, було вирішено розробити мобільний застосунок для моніторингу та фіксації міських проблем із передачею даних до ОМС.

1.2 Аналіз існуючих рішень

Розглянемо популярні сучасні технічні рішення, що допоможуть у реалізації мобільного застосунку для моніторингу та фіксації міських проблем із передачею даних до ОМС. Надалі розглянемо уже готові технічні та програмні рішення й інструменти розробки.

1.2.1 Аналіз відомих програмних продуктів

Зазначимо, що подібні ПЗ уже існують на ринку. Серед конкурентів можна виділити наступні сервіси для комунікації з ОМС: Київ Цифровий, Misto та SeeClickFix, які забезпечують обмін інформацією. Також є застосунок FixMyStreet, проте він орієнтований на певні країни чи регіони, і для них потрібна адаптація до місцевого контексту.

Київ Цифровий

Київ Цифровий — міський застосунок для мешканців Києва, який дозволяє оплачувати транспорт, паркування, подавати звернення щодо міських проблем, отримувати довідки й користуватися різними міськими послугами в одному місці.

Misto

Misto – український застосунок/платформа для фіксації міських проблем, де користувачі можуть додавати фото, опис проблеми та передавати інформацію до місцевих органів влади для подальшого опрацювання

SeeClickFix

SeeClickFix – міжнародний сервіс і мобільний застосунок, що дозволяє громадянам повідомляти про проблеми у міському середовищі (ямки, пошкоджені дороги, сміття тощо) й відстежувати статус їх вирішення місцевими органами влади.

Кожен із цих застосунків має унікальні особливості, проте наразі не існує додатку, який надає змогу фотофіксації скарги та відображення статусу по заявці. Для наочного порівняння мобільних застосунків для моніторингу та фіксації міських проблем із передачею даних до ОМС скористаємось таблицею 1.1.

Таблиця 1.1 – Порівняння з аналогами

Особливості та функціонал застосунків	Розроблений мобільний застосунок	Київ Цифровий	Misto	SeeClickFix
Основна мета	Система для фіксації та обробки міських проблем	Екосистема цифрових послуг міста	Комунікація громадян з владою, локальні послуги	Фіксація міських проблем + взаємодія з владою
Подача скарг/ звернень	Основний функціонал	Реалізовано через інтеграцію з 1551	Реалізовано не у всіх містах	Основний функціонал
Фотофіксація проблем	Основний функціонал	Частково, не для всіх типів проблем	Часом є, але залежить від міста	Основний функціонал
Автоматична геолокація	Основний функціонал	Не для всіх форм	Частково, залежить від реалізації	Основний функціонал
Відстеження статусу звернення	Повний цикл: створено → прийнято → в роботі → виконано	Реалізовано тільки для звернень 1551	Частково	Реалізовано
Публічна мапа проблем міста	Основний функціонал	Відсутня	Відсутня	Реалізовано
Аналітика для ОМС	Реалізовано розширену аналітику	Реалізовано в загальному вигляді	Залежить від реалізації	Реалізовано
Прив’язка до конкретного міста	Будь-яке місто/громада	Лише Київ	Одне місто/громада	Лише міста, що підключені
Зручність для користувача	Адаптовано під швидке створення звернень	Фокус на сервісах, а не проблемах	Застарілий інтерфейс	Оптимізовано під фіксацію проблем
Орієнтація на українські реалії	Так	Так	Так	Ні

1.2.2 Аналіз відомих алгоритмічних та технічних рішень

Розробка клієнтської частини інформаційної системи для фіксації муніципальних проблем вимагає ретельного підходу до вибору цільової платформи. Оскільки ефективність системи напряму залежить від кількості активних користувачів, які надсилатимуть дані про проблеми, критичним фактором є максимальне охоплення аудиторії.

На сучасному ринку мобільних технологій домінує дуополія двох операційних систем: Android від Google та iOS від Apple. Станом на кінець 2024 року, ці дві платформи сумарно займають понад 99% ринку [5]. Для прийняття рішення щодо пріоритетної платформи розробки доцільно проаналізувати глобальний розподіл часток ринку (рис. 1.2).

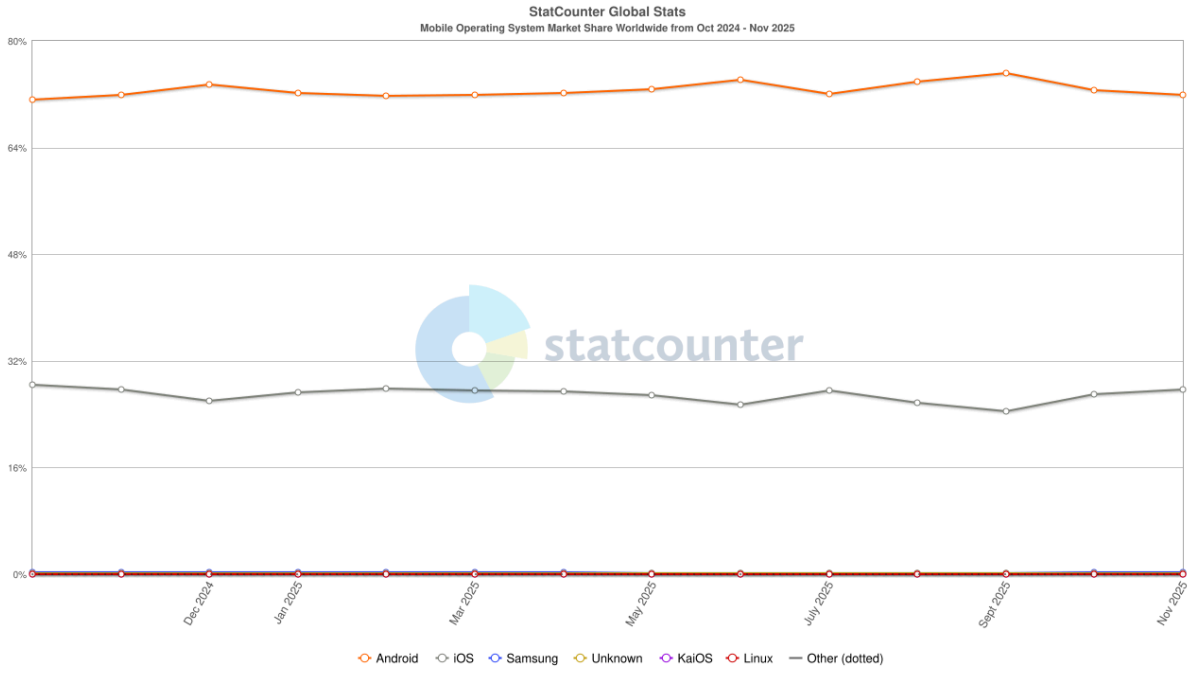

Рисунок 1.2 – Графік розподілу мобільних операційних систем по світовому ринку [5]

Як свідчать дані діаграми, операційна система Android утримує лідерство з часткою ринку близько 71%. Для соціально орієнтованого проекту, яким є система фіксації міських проблем, вибір Android як пріоритетної платформи є стратегічно обґрунтованим з кількох причин:

Соціальна доступність: Android встановлюється на пристроях усіх цінових категорій — від бюджетних смартфонів до флагманів. Це дозволяє

залучити до використання системи всі верстви населення, незалежно від рівня доходу, що є критично важливим для муніципального сервісу.

Апаратна гнучкість: Широкий спектр пристроїв дозволяє тестувати роботу камери та GPS-модулів (ключових компонентів для фотофіксації та геолокації проблем) у різних умовах.

Екосистема розробки: Наявність потужних інструментів розробки та документації дозволяє ефективно реалізувати роботу з картографічними сервісами (Google Maps) та фонову передачу даних на сервер.

Важливим етапом проектування є визначення мінімальної підтримуваної версії (Minimum SDK). Підтримка занадто старих версій ускладнює розробку через необхідність зворотної сумісності та створює ризики безпеки. Водночас, відмова від підтримки старіших версій може відсіяти частину активних громадян.

Аналіз фрагментації версій Android (рис. 1.3) показує, що частка пристроїв з версіями нижче Android 9.0 (Pie) стрімко скорочується і наразі становить менше 5-8% активних пристроїв [6].

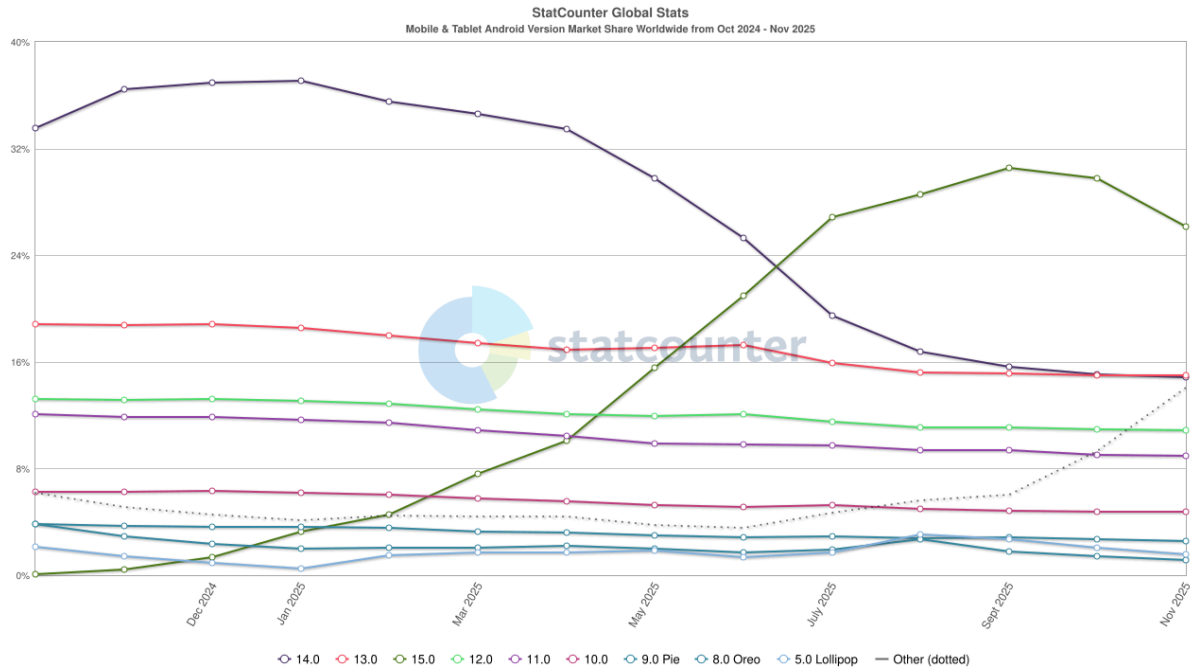

Рисунок 1.3 – Графік розподілу версій Android [6]

Для розроблюваного застосунку було прийнято рішення встановити мінімальну версію Android 10 (API Level 29). Це рішення обумовлено наступним:

1. Безпека та приватність: Починаючи з Android 10, Google суттєво змінив механізм роботи з дозволами на геолокацію та доступ до файлової системи, що є критичним для коректної та безпечної відправки звітів про проблеми.
2. Оптимізація: Сучасні API дозволяють краще керувати енергоспоживанням під час роботи з GPS та камерою.
3. Актуальність: Версії Android 10, 11, 12, 13 та 14 сумарно покривають понад 85% сучасних пристроїв, що забезпечує достатнє охоплення аудиторії без необхідності підтримки застарілих технологій.

Таким чином, вибір платформи Android з підтримкою версій 10+ є оптимальним балансом між охопленням аудиторії, вартістю розробки та технічними можливостями для реалізації функціонала фотофіксації.

Для забезпечення масштабованості та тестованості системи фіксації міських проблем, критично важливим є поділ коду на логічні шари. У розробці під Android стандартом є використання патернів сімейства MV* (Model-View-...), зокрема MVC, MVP та MVVM.

Model-View-Controller (MVC) Найстаріший підхід, де Controller керує введенням даних. У контексті додатку для моніторингу та фіксації, головним недоліком MVC є тісний зв'язок між логікою обробки зображення та відображенням інтерфейсу (рис. 1.4). Це призводить до створення "масивних контролерів" (Massive View Controller), коли код роботи з камерою, GPS та UI змішується в одному класі, ускладнюючи підтримку [7].

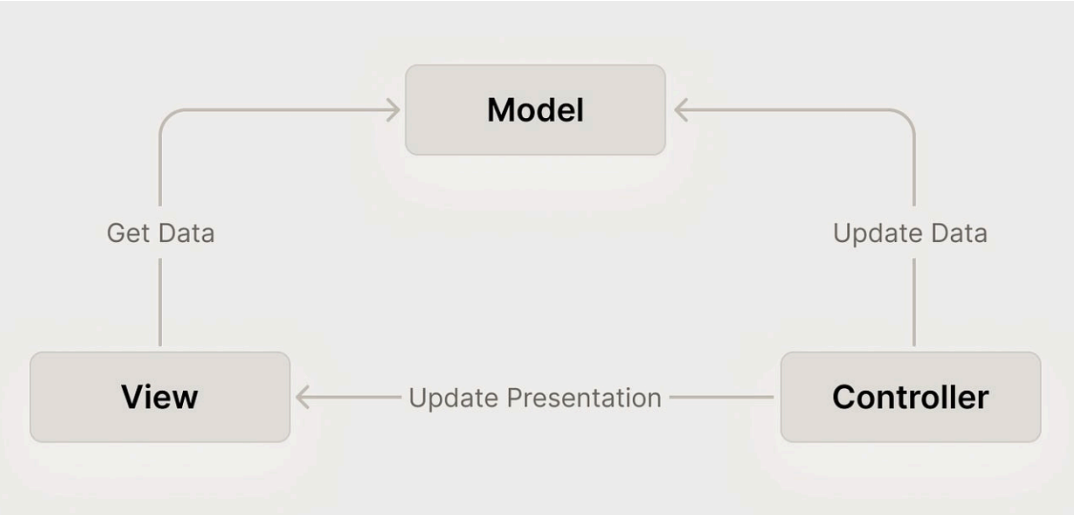

Рисунок 1.4 – Схема архітектурного патерну MVC [7]

Model-View-Presenter (MVP) Еволюція MVC, що розриває прямий зв'язок між View та Model через посередника — Presenter (рис. 1.5).

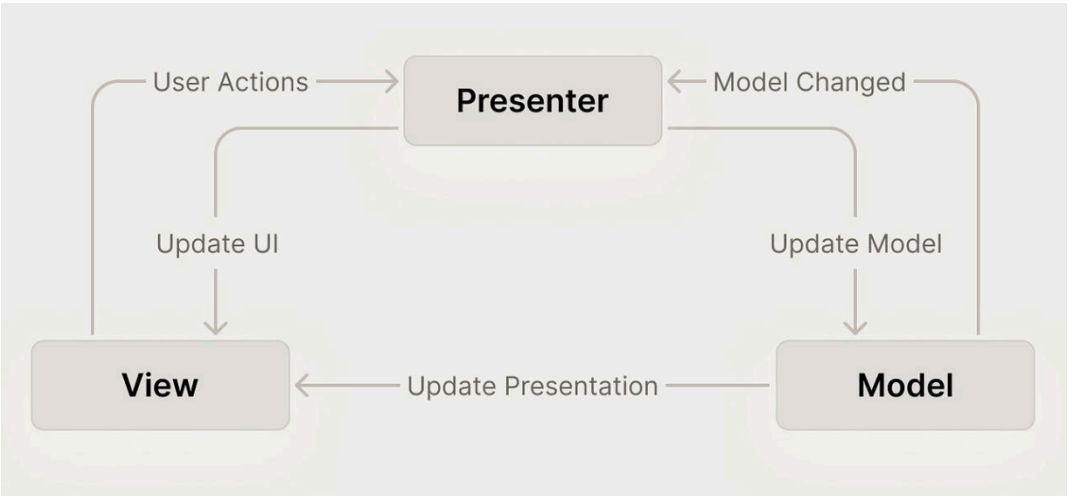

Рисунок 1.5 – Схема архітектурного патерну MVP [7]

Перевага: Дозволяє легко покрити Unit-тестами логіку валідації форми опису проблеми, не запускаючи емулятор.

Недолік: Необхідність писати багато шаблонного коду для ручного оновлення UI (наприклад, показ прогрес-бару при відправці фото на сервер).

Model-View-ViewModel (MVVM) Сучасний стандарт, рекомендований Google (Android Jetpack). Основна відмінність — використання Data Binding та Observable полів (рис. 1.6).

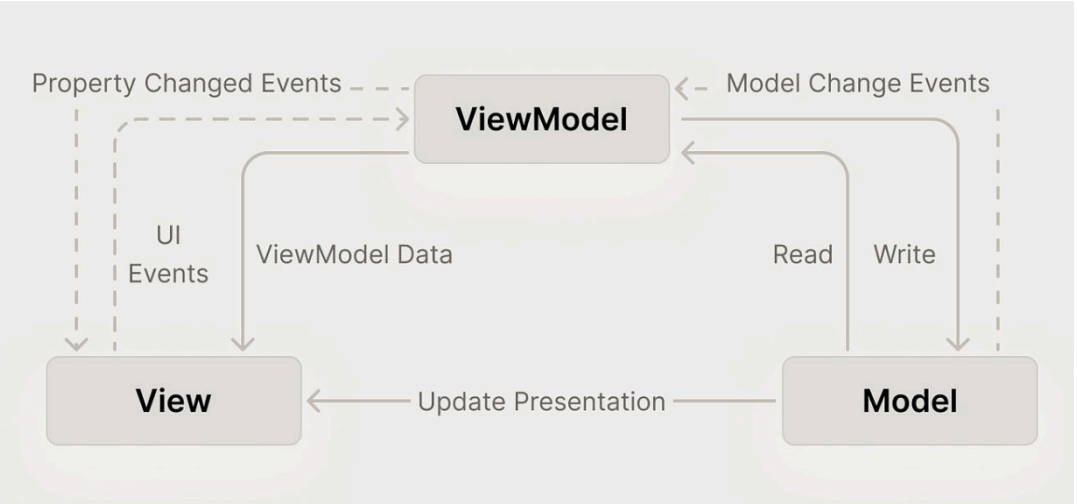

Рисунок 1.6 – Схема архітектурного патерну MVVM [7]

Застосування у проекті: ViewModel автоматично зберігає стан форми (опис проблеми, вибрана категорія) при повороті екрану або вхідному дзвінку. View автоматично реагує на зміну статусу завантаження фотографії ("підписка" на події), що робить інтерфейс більш чутливим.

Для розробки системи було обрано MVVM, оскільки цей патерн найкраще підходить для асинхронної природи задачі (фонове завантаження фото та координат) і забезпечує збереження даних користувача під час життєвого циклу Android-компонентів [8,9].

1.3 Постановка задачі

Метою даної розробки є полегшити процес комунікації громадян та ОМС, а також покращити стан міста.

Результатом виконання даної кваліфікаційної роботи повинна бути інформаційна система для моніторингу та фіксації міських проблем із передачею даних до ОМС, яка передбачає можливість створювати акаунти, авторизуватись в них; жителі як користувачі матимуть змогу додавати фотофіксацію проблеми, створювати скарги, переглядати статус скарги та редагувати її, визначати місце розташування, а також для ОМС мати можливість отримувати скарги та звітність.

Висновки до розділу

У результаті виконання даного розділу кваліфікаційної роботи було проведено аналіз предметної області. Розглянуто сутність інформаційних систем, їх класифікацію. Обґрунтовано переваги використання саме мобільного застосунку для моніторингу та фіксації проблем. Виконано аналіз поточного стану комунікації громадян з ОМС у Хмельницькій області та виявлено тенденцію до зростання кількості скарг, що підтверджує необхідність покращення взаємодії.

Проведено дослідження аналогів, їх можливості та виконано порівняння з існуючими конкурентами, такими як «Київ Цифровий», «Misto» та «SeeClickFix». Порівняння показало, що існуючі рішення не зовсім покривають предметну область через локальну прив’язку до конкретних міст або відсутність повного циклу обробки заявки.

Для актуальності розробки, було проведено дослідження платформи, яка підійде найбільше. Серед фаворитів було обрано ОС Android, оскільки частка ринку становить близько 71%, що забезпечує соціальну доступність проєкту. Також було визначено версії, які будуть підтримуватись. За мінімальний рівень підтримки було взято Android 10 для балансу між безпекою, функціональними можливостями та охопленням аудиторії.

Для кращого розуміння побудови Android-додатків, було проведено аналіз архітектурних патернів, які найчастіше використовуються у розробці. Серед найпопулярніших було виділено MVC, MVP та MVVM. Після порівняння, для майбутнього застосунку було обрано використовувати архітектурний патерн MVVM, який рекомендований Google та найкраще підходить для асинхронної роботи з даними та фотофіксацією.

Як результат роботи, в кінці було визначено мету даної розробки, а також сформовано текст умови із усіма загальними задачами, які повинна вирішувати інформаційна система для покращення благоустрою міста.

РОЗДІЛ 2 РОЗРОБЛЕННЯ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Варіанти використання програмного забезпечення

Основною функцією інформаційної системи для моніторингу та фіксації міських проблем із передачею даних до органів місцевого самоврядування є забезпечення усіх ключових процесів щодо обробки запитів користувачів з наявними проблемами та реагування на них. До таких процесів відносяться авторизація користувачів, створення звернень з можливістю додавання фото та розташування, відслідковування статусу звернень, перегляд наявних звернень від інших користувачів, отримання змін, при реагуванні органів місцевого самоврядування. Детальну інформацію щодо усіх функцій зображено на рисунку 2.1.

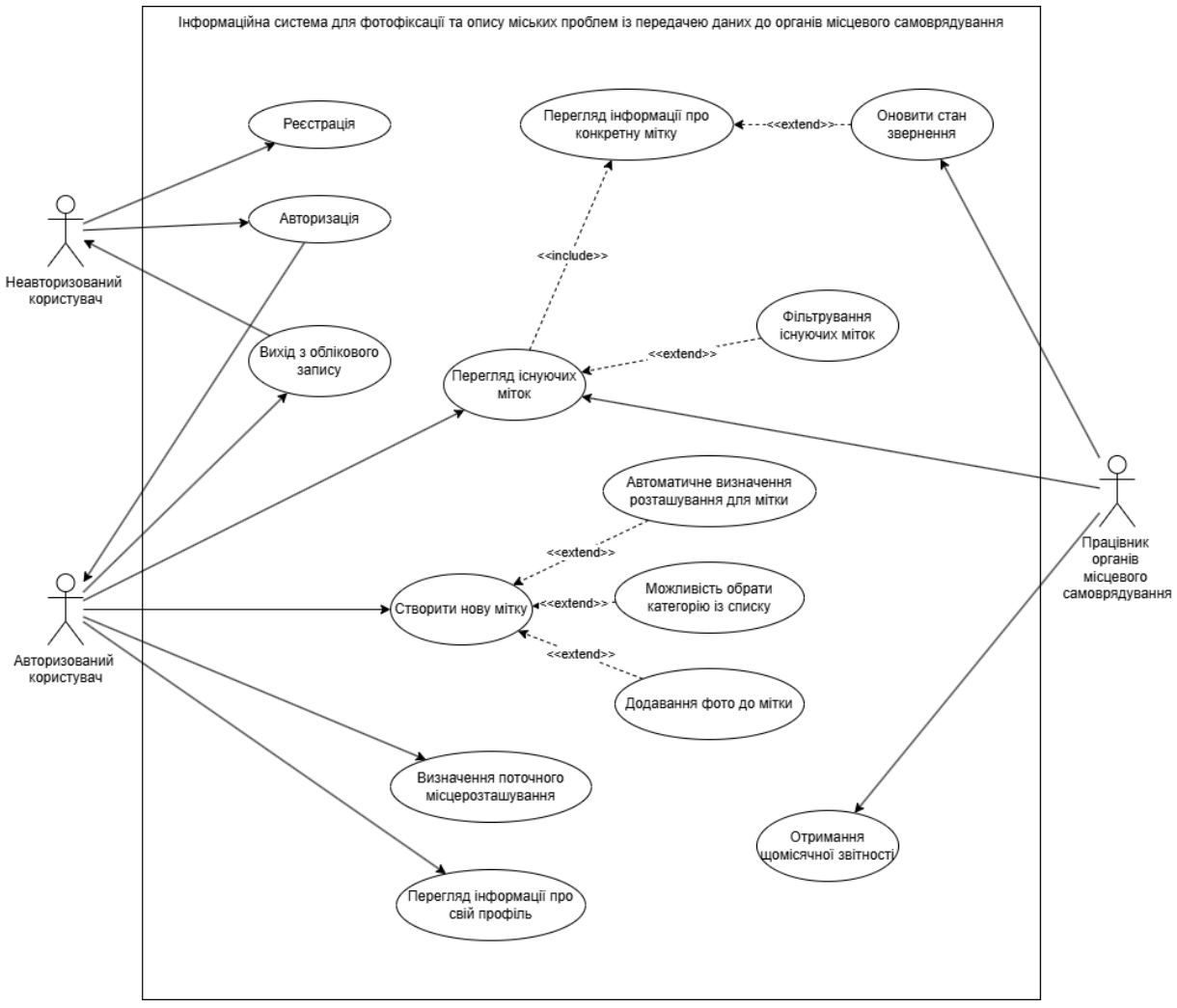

Рисунок 2.1 – Діаграма варіантів використання

У додатку А, а саме у таблицях А.1 – А.10 наведені усі варіанти використання програмного забезпечення.

2.2 Розроблення функціональних вимог

З метою декомпонування задачі та полегшення процесів розробки, інформаційна система була поділена на модулі, які дозволять чітко визначити етапи та їх пріоритетність. У додатку Б, у таблиці Б.1 наведено загальну модель вимог програмного забезпечення, а в таблиці Б.2 наведений детальний опис кожної функціональної вимоги. У результатів було створено матрицю трасування вимог (рис. 2.2).

	FR-1	FR-2	FR-3	FR-4	FR-5	FR-6	FR-7	FR-8	FR-9	FR-10	FR-11	FR-12	FR-13	FR-14	FR-15	FR-16	FR-17	FR-18	FR-19	FR-20
UC-01	+	+																		
UC-02	+		+																	
UC-03	+			+																
UC-04							+	+	+							+				
UC-05							+			+										
UC-06							+				+	+	+		+					
UC-07														+		+				
UC-08					+	+														
UC-09							+	+	+	+							+			
UC-10																	+	+		
UC-11																			+	+

Рисунок 2.2 – Матриця трасування вимог

2.3 Розроблення нефункціональних вимог

Перелік нефункціональних вимог, які будуть висунуті до ІС для моніторингу та фіксації міських проблем наведений у таблиці 2.1.

Таблиця 2.1 – Перелік нефункціональних вимог

Назва вимоги	Опис вимог
Продуктивність	Система повинна обробляти запит (завантаження фото + опис + координати) не довше ніж за 3 секунди при нормальному навантаженні.
Масштабованість	Архітектура має підтримувати горизонтальне масштабування для обробки збільшення кількості користувачів і обсягу даних.
Доступність	Система повинна бути доступною не менше 99.5% часу на місяць.
Надійність	У разі відмови одного сервісу система повинна коректно відновлюватися без втрати даних.
Безпека даних	Дані користувачів та фото мають передаватися через захищений протокол (HTTPS, TLS 1.2+).
Авторизація та контроль доступу	Доступ до адміністративної панелі та службових функцій повинен здійснюватися через рольову модель доступу.
Конфіденційність	Персональні дані (контактні дані заявника, його геолокація) повинні бути захищені згідно з законодавством та не передаватися без потреби.
Цілісність даних	Система повинна гарантувати, що дані заявки не можуть бути змінені сторонніми особами.
Аудит та логування	Усі запити, зміни статусів та дії службових осіб мають логуватися з часовими мітками.
Портативність	Сервіс має розгортатися у контейнерах (Docker) та бути придатним для перенесення між різними хмарними платформами.
Зручність використання	Інтерфейс повинен бути інтуїтивним, з можливістю подачі заявки за ≤ 5 кліків.
Часова узгодженість	Усі дані у заявках мають фіксуватися із використанням єдиної часової зони та синхронізованих серверних годинників.

Висновки до розділу

У цьому розділі кваліфікаційної роботи визначено вимоги до інформаційної системи, призначеної для фіксації міських проблем, додавання фотографій та передачі зібраних даних до органів місцевого самоврядування.

Для забезпечення чіткого розуміння всіх процесів роботи системи була побудована діаграма варіантів використання, на якій відображено дії, доступні різним учасникам системи — від звичайного користувача до представників муніципальних служб. На основі цієї діаграми були сформовані усі сценарії використання (use cases), кожному з яких надано унікальний код і описано умови запуску, послідовність дій та очікувані результати.

Щоб уніфікувати вимоги до системи та спростити подальшу розробку, була створена структурована модель функціональних вимог. Кожна вимога отримала власний ідентифікатор і розгорнутий опис, що дозволяє розробникам чітко розуміти обсяг необхідного функціоналу.

На основі сформованих варіантів використання та переліку функціональних вимог було побудовано матрицю трасування. Вона дозволяє визначити взаємозв’язок між вимогами та конкретними сценаріями роботи системи, забезпечуючи прозорість і повноту покриття всіх потреб.

Для підвищення якості роботи системи та покращення користувацького досвіду були сформовані нефункціональні вимоги. Їх дотримання гарантує стабільність, безпеку та зручність використання сервісу, що напряму впливає на рівень залучення користувачів.

РОЗДІЛ 3 КОНСТРУЮВАННЯ ТА РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Архітектура програмного забезпечення

У якості архітектурної основи інформаційної системи для моніторингу та фіксації міських проблем із передачею даних до ОМС використано клієнт-серверний підхід.

Клієнт-серверна архітектура — це модель мережевої взаємодії, в якій обчислювальне навантаження розподіляється між клієнтами (ініціаторами запитів, які потребують доступу до ресурсів чи послуг) та серверами (вузлами, що зберігають дані, виконують бізнес-логіку та надсилають відповіді) [10]. У цій схемі комунікація завжди починається з боку клієнта, який надсилає запит через мережу, тоді як сервер працює в режимі очікування, приймає цей запит, обробляє його і повертає результат, що дозволяє централізувати управління інформацією та забезпечити одночасний доступ до неї для великої кількості користувачів.

Клієнт-серверна архітектура дозволяє чітко розмежувати відповідальності між мобільним застосунком та серверною частиною [10]. Такий підхід забезпечує масштабованість, гнучкість та можливість подальшого розширення функціональних можливостей системи без суттєвих змін у клієнтському застосунку.

Мобільний застосунок реалізовано відповідно до патерну MVVM, як зазначалось вище, що дозволяє відокремити логіку представлення від бізнес-логіки та спростити підтримку й тестування коду. Серверна частина побудована за принципами шарової архітектури (Layered Architecture), яка структурує програмний код на контролери, сервіси та репозиторії.

Архітектура Android-клієнта складається з трьох основних компонентів:

Модель (Model) — реалізує доступ до даних через HTTP-клієнти, відповідає за виконання запитів до серверного API, обробку відповідей та формування доменних моделей. До моделі також належать сутності, що

використовуються всередині застосунку, та допоміжні компоненти (наприклад, менеджер токенів або модулі для роботи з фотографіями і геолокацією).

Представлення (View) — активності Android, що відповідають за відображення інтерфейсу користувача, отримання вводу, ініціацію бізнес-операцій та відображення результатів. До складу представлення належать екрани завантаження, авторизації, реєстрації, перегляду карти, створення звітів та профілю.

Модель представлення (ViewModel) — проміжна ланка між View та Model. Опрацьовує дії користувача, виконує бізнес-операції через модель та повертає результат у зручній формі. Відповідає за логіку інтерфейсу та містить мінімум залежностей від Android-компонентів.

Архітектуру клієнтської частини доцільно подавати у вигляді діаграми пакетів, де підсистема views містить елементи представлення, viewmodels — відповідні моделі представлення, а models — логіку роботи з API, токенами та доменними сутностями.

Серверна частина використовує шарову архітектуру, розділену на такі компоненти:

Контролери (Controller layer) — реалізують обробку HTTP-запитів, виконують валідацію вхідних даних, формують відповіді API та делегують бізнес-логіку сервісам.

Сервіси (Service layer) — містять бізнес-логіку системи: реєстрацію та авторизацію користувачів, створення звітів, визначення найближчого міста за координатами, оновлення профілю тощо. Сервіси координують роботу репозиторіїв та відповідають за цілісність бізнес-процесів.

Репозиторії (Repository layer) — забезпечують доступ до даних у базі PostgreSQL, включаючи виконання геопросторових запитів завдяки розширенню PostGIS. Репозиторії реалізують формування SQL-запитів та повертають структуровані доменні моделі.

Моделі (Model layer) — описують сутності бази даних, такі як користувач, звіт чи місто.

Структура серверної частини доповнена middleware-компонентами, що забезпечують автентифікацію, логування та контроль CORS-запитів, а також інтеграцією з Redis для зберігання чорного списку JWT-токенів.

У даному випадку, розроблену архітектуру зручно буде подати у вигляді діаграми, яка зображена на рисунку 3.1.

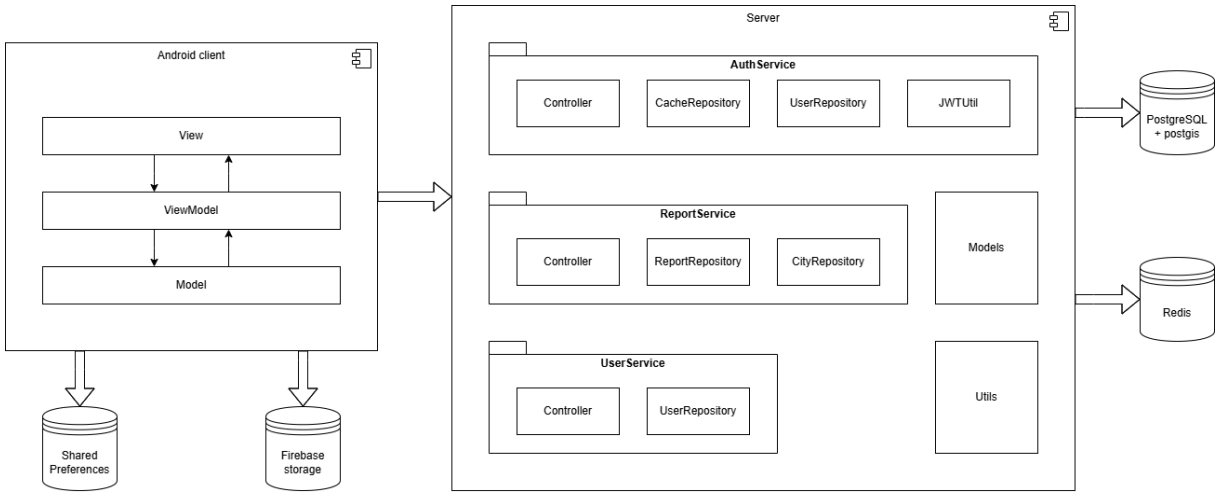

Рисунок 3.1 – Архітектура програмного забезпечення

Для обміну даними між мобільним застосунком та серверною частиною в проєкті використано REST API, що передає інформацію у форматі JSON. Такий підхід забезпечує просту, передбачувану та стандартизовану взаємодію між компонентами системи.

REST API (Representational State Transfer Application Programming Interface) — це архітектурний стиль побудови веб-застосунків, який базується на використанні стандартних HTTP-методів (GET, POST, PUT, DELETE) та чітко визначених ресурсів [11]. Кожен ресурс має власний URL, а запити є статичними та не містять інформації про попередній контекст, що робить REST безстанним і спрощує масштабування.

JSON (JavaScript Object Notation) використовується як основний формат серіалізації даних. Він має компактну структуру, легко читається людиною й ефективно обробляється на мобільних пристроях. У контексті

Android застосунку це дозволяє мінімізувати затрати ресурсів та пришвидшити обмін інформацією.

Використання REST API має низку переваг порівняно з іншими способами комунікації, такими як SOAP, GraphQL або RPC-орієнтовані протоколи:

1. Простота реалізації - REST не потребує складних специфікацій чи додаткових транспортних протоколів. Достатньо підтримки HTTP, що присутня практично у будь-якій платформі.
2. Легкість інтеграції - більшість клієнтських бібліотек, включаючи Android Networking, Retrofit чи OkHttp, мають вбудовану підтримку REST, що спрощує розробку та тестування.
3. Швидкість роботи й низький overhead - JSON є легким форматом обміну даними й передбачає менше накладних витрат, ніж XML у SOAP або складні структури GraphQL.
4. Масштабованість - безстанна модель REST дозволяє серверу обробляти більше паралельних запитів без зберігання сесій, що важливо для навантажених систем.
5. Гнучкість і універсальність - REST не накладає обмежень на типи клієнтів — ним можуть користуватися мобільні застосунки, веб-клієнти, автоматизовані сервіси та сторонні системи [11].

Основою сховища даних є PostgreSQL. PostgreSQL — це потужна об'єктно-реляційна система керування базами даних (ORDBMS) з відкритим вихідним кодом, яка відома своєю винятковою надійністю, суворим дотриманням стандартів SQL та архітектурною розширюваністю [12]. Вона підтримує як класичні реляційні (табличні), так і нереляційні (JSON, XML) типи даних, гарантує повну транзакційну цілісність (ACID) і дозволяє розробникам створювати власні функції, типи даних та індекси, що робить її універсальним стандартом для високонавантажених систем.

ACID — це набір із чотирьох ключових властивостей транзакційної системи, які гарантують надійність та цілісність даних навіть у разі збоїв обладнання чи помилок [13]. Абревіатура розшифровується як Atomicity (Атомарність — транзакція виконується повністю або не виконується зовсім, принцип "все або нічого"), Consistency (Узгодженість — транзакція переводить базу даних з одного валідного стану в інший, не порушуючи встановлених правил та обмежень), Isolation (Ізольованість — паралельні транзакції виконуються так, ніби вони відбуваються послідовно, не впливаючи одна на одну до завершення), та Durability (Довговічність — після підтвердження транзакції зміни зберігаються постійно, навіть якщо система одразу після цього вимкнеться).

Розширення PostGIS для PostgreSQL забезпечує виконання геопросторових операцій: визначення найближчого міста, сортування звітів за відстанню, зберігання координат у форматі Point. Використання GIST-індексів підвищує продуктивність запитів за географічними координатами.

Система не зберігає зображення безпосередньо на сервері — замість цього використовується Firebase Storage.

Firebase Storage — це масштабований сервіс хмарного зберігання об’єктів, побудований на інфраструктурі Google Cloud Storage, який оптимізовано для обслуговування мобільних та веб-застосунків [14]. Він дозволяє розробникам безпечно завантажувати та зберігати користувацький контент (зображення, аудіо, відео) без необхідності підтримувати власні сервери, при цьому автоматично обробляючи проблеми нестабільного мережевого з'єднання (наприклад, відновлюючи завантаження після обриву). Сервіс інтегрується з системою аутентифікації Firebase, що дозволяє налаштовувати гнучкі правила доступу до файлів залежно від статусу користувача, забезпечуючи надійність рівня enterprise у зручній для розробки обгортці.

Клієнт завантажує файл у хмарне сховище, отримує публічний URL і передає його в бекенд разом із даними звіту. Такий підхід зменшує навантаження на сервер та забезпечує високу доступність фото через CDN.

У інформаційній системі використовується двофазна аутентифікація на основі JWT: короткочасний access-токен та довготривалий refresh-токен з механізмом ротації. Хешування паролів здійснюється за допомогою алгоритму bcrypt. На клієнті токени зберігаються у зашифрованому сховищі EncryptedSharedPreferences, що запобігає компрометації облікових даних.

Для забезпечення інтероперабельності системи та її безшовної інтеграції в існуючу IT-інфраструктуру міста, архітектурою передбачено реалізацію зовнішнього прикладного програмного інтерфейсу (RESTful API). Цей модуль виступає єдиною точкою входу для обміну даними між системою моніторингу та інформаційними ресурсами муніципалітету. Відкритість API дозволяє технічним фахівцям органів місцевого самоврядування самостійно розробляти сценарії інтеграції, налаштовувати автоматичний імпорт заявок у внутрішні системи електронного документообігу (наприклад CRM) або створювати власні аналітичні дашборди без необхідності втручання у вихідний код ядра системи. Обмін даними здійснюється у форматі JSON з використанням захищених протоколів передачі, що гарантує конфіденційність та цілісність інформації.

Паралельно, у випадку відсутності ресурсів на розробку у відділа IT-інфраструктури міста, для безпосередньої роботи співробітників комунальних служб та диспетчерів, на стороні сервера реалізовано адміністративний WEB-інтерфейс (за допомогою HTML та CSS). Цей компонент не вимагає встановлення додаткового програмного забезпечення на робочі місця, оскільки доступ до нього здійснюється через стандартний веб-браузер. Функціонал веб-інтерфейсу надає візуалізований доступ до бази даних проблем, дозволяючи уповноваженим особам переглядати нові

заявки з прив’язкою до карти, змінювати статуси виконання робіт («В обробці», «Виконано» і т.д.) та формувати звіти. Такий підхід забезпечує готове рішення для громад, які не мають технічної можливості проводити складну інтеграцію через API.

Для роботи із мапою, було обрано використовувати Maps SDK for Android. Maps SDK for Android — це програмний інструментарій від Google, який дозволяє інтегрувати інтерактивні карти, маркери, геолокаційні сервіси та елементи навігації безпосередньо в Android-застосунки [15]. Він надає доступ до відображення місцевості, роботи з координатами, взаємодії з картою через жести, а також можливість динамічної побудови елементів на карті. У межах проєкту цей SDK використовується для показу поточного місцезнаходження користувача, розміщення маркерів проблемних точок, візуалізації створених звітів на мапі та відображення вибраної локації під час формування нового звернення, що забезпечує зручну та інтуїтивну взаємодію з геопросторовими даними.

Технологічний стек, який склався на даний момент, наведено в таблиці 3.1.

Таблиця 3.1 – Технологічний стек

Тип технології	Назва технології
Операційна система (клієнт)	Android OS
Архітектурний патерн (клієнт)	MVVM
Хмарне сховище медіафайлів	Firebase Storage
База даних (клієнт)	EncryptedSharedPreferences
Формат передачі даних	JSON (REST API)
API протокол	REST over HTTPS
Архітектурний патерн (сервер)	Шаровий модульний моноліт
База даних (сервер)	PostgreSQL + PostGIS
Кеш	Redis
Обробка геолокації	PostGIS (ST_Distance, ST_Point, інше)
Керування токенами	JWT (access + refresh)
WEB-інтерфейс	HTML + CSS
Засоби для роботи із мапою	Maps SDK for Android

3.2 Обґрунтування вибору засобів розробки

Для завершення формування технологічного стеку інформаційної системи необхідно визначити основні інструменти розробки, мови програмування та інтегровані середовища (IDE), що застосовуються під час створення клієнтської та серверної частини застосунку. Вибір інструментів безпосередньо впливає на продуктивність розробки, швидкість розгортання та якість підтримки програмного продукту.

Під час створення Android-застосунку для даного проєкту використано Android Studio, оскільки це офіційне середовище розробки від Google, яке надає повний набір інструментів для створення, налагодження та тестування мобільних додатків [16, 17]. Серед ключових можливостей Android Studio варто виділити:

1. гнучку систему збирання проєктів на базі Gradle;
2. високопродуктивний Android-емулятор з широкими можливостями налаштування;
3. механізм «живого редагування» інтерфейсу, що дозволяє переглядати зміни в реальному часі;
4. інтегровані засоби для модульного, UI- та інструментального тестування;
5. повну підтримку сервісів Google, зокрема Firebase, що значно спрощує підключення автентифікації, хмарного сховища та бази даних [17].

До складу Android Studio входить Android SDK — офіційний набір інструментів для розробки під Android. До SDK входять емулятори, бібліотеки підтримки, системні API, документація, приклади коду та засоби відлагодження. Саме SDK забезпечує доступ до функцій мобільної платформи, таких як камера, GPS, робота з файлами та мережею.

Окрім Android Studio, у середовищі розробки активно використовується IntelliJ IDEA — IDE, на базі якої фактично створено Android Studio. IntelliJ IDEA відома високою продуктивністю,

інтелектуальними підказками, розвинутими інструментами рефакторингу та підтримкою широкого спектра JVM-мов [18].

Історично для Android існувало й середовище розробки Eclipse з ADT (Android Development Tools), однак після офіційного запуску Android Studio підтримка ADT була припинена [19]. Через це Eclipse з ADT не розглядається як актуальний інструмент для сучасної Android-розробки.

Беручи до уваги переваги та функціональність, Android Studio є оптимальним вибором для створення клієнтської частини інформаційної системи для моніторингу та фіксації міських проблем із передачею даних до ОМС.

Для розробки Android-клієнта обрано мову програмування Java, яка є однією з основних мов платформ Android і JVM. Java характеризується стабільністю, надійністю та широкою підтримкою бібліотек, що важливо для розробки мобільного застосунку, який працює з мережею, геолокацією, камерою та хмарними сервісами.

Серед ключових переваг використання Java:

1. об'єктно-орієнтований підхід, що полегшує підтримку й розширення проєкту;
2. платформонезалежність, яка дозволяє частково переносити бізнес-логіку на інші платформи;
3. широка екосистема бібліотек і фреймворків, необхідних для роботи з мережею, JSON, геолокацією тощо;
4. велика спільнота розробників, що забезпечує постійний розвиток інструментів і доступ до матеріалів підтримки [20].

Для роботи з Java використовується JDK (Java Development Kit) — офіційний набір інструментів, який містить компілятор, стандартну бібліотеку, документацію та інші службові утиліти, необхідні для створення, компіляції та запуску програм [20].

Альтернативою Java у сучасній Android-розробці є мова Kotlin. Вона є більш лаконічною, безпечнішою у плані типів і повністю сумісною із

Java, оскільки також працює на JVM [21]. Kotlin дозволяє зменшити кількість коду, уникнути типових помилок (наприклад, NullPointerException) та забезпечує розширену підтримку сучасних парадигм програмування. Попри це, у межах проєкту було прийнято рішення використовувати Java як основну мову клієнтської частини, з огляду на легкість інтеграції та відповідність вимогам до стабільності.

Для серверної частини обрано мову Go (Golang), яка відома високою швидкодією, простотою синтаксису та ефективною роботою з багатопоточністю. Go є особливо придатною для створення високонавантажених серверів, REST API та мікросервісів, що відповідає вимогам побудови серверної частини інформаційної системи [22].

Основні переваги використання Go

1. Висока продуктивність - Go компілюється у машинний код, що дозволяє серверу працювати швидше, ніж інтерпретовані або віртуалізовані мови (наприклад, Python, JavaScript або PHP). Завдяки цьому серверна частина обробляє мережеві запити з мінімальними затримками, що є критичним під час роботи з геолокаційними запитами, фільтрацією даних та авторизацією.
2. Вбудована підтримка багатопоточності - Go має легковагові потоки виконання — goroutines, які дозволяють серверу обробляти тисячі одночасних запитів без суттєвого збільшення споживання пам'яті. Це робить Go оптимальним вибором для серверної частини, яка обслуговує Android-клієнт із потенційно високою кількістю паралельних звернень.
3. Проста та зрозуміла модель конкурентності - Go використовує принципи communicating sequential processes (CSP), де взаємодія потоків відбувається через канали (channels). Такий підхід знижує ймовірність типових проблем конкурентності та суттєво спрощує розробку надійного API.

4. Мінімалістичний, але потужний синтаксис - Go свідомо уникає надлишкової складності мов на кшталт C++ або Java — у ньому немає складних ієрархій спадкування, операторів перевантаження чи надмірного функціоналу. Це робить код чистим, легким для читання та зменшує час навчання нових розробників.
5. Статична типізація і безпечність - Go є статично типізованою мовою, що дозволяє виявляти помилки ще на етапі компіляції. Це особливо важливо для серверної частини, де надійність і передбачуваність поведінки мають критичне значення.
6. Готові інструменти для роботи з мережею та файловою системою - Go постачається з багатою стандартною бібліотекою, яка включає все необхідне для реалізації REST API:
    a. HTTP-сервер і клієнт,
    b. інструменти роботи з JSON,
    c. підтримку TLS,
    d. роботу з файлами та процесами,
    e. засоби логування.

Це дозволяє будувати серверну частину без залежності від великої кількості сторонніх бібліотек. Порівняння Go з популярними альтернативами — такими як Node.js, Java, Python або PHP — можна побачити у таблиці 3.2.

Таблиця 3.2 – Порівняння мови програмування Go з конкурентами

Мова	Переваги Go над аналогом
Node.js	Go швидше завдяки компіляції; краще працює з багатопоточністю; менш чутливий до проблем callback-heavy логіки
Python	Вища продуктивність; меморі-ефективність; легша конкурентність
Java	Простіше налаштування; менше пам’яті; легший деплой без JVM; чистіший код
PHP	Вища швидкодія; сильна типізація; краща підтримка паралельних обчислень

3.3 Конструювання програмного забезпечення

Android-застосунок можна представити у вигляді окремих екранів (View), для кожного з якого було розроблена модель (Model) та модель представлення (ViewModel). Опис основних компонентів мобільного клієнта наведено у таблиці 3.3.

Таблиця 3.3 – Опис основних компонентів мобільного застосунку

№	Ім’я компоненти	Опис компоненти
1	LoginModel	Надає функціонал для входу в обліковий запис.
2	RegistrationModel	Надає функціонал для реєстрації нового облікового запису.
3	LoaderModel	Визначає, чи був попередньо виконаний вхід із збереженням токенів аутентифікації. У випадку валідних токенів, виконується вхід, в протилежному користувач буде змушений заново авторизуватись.
4	MainModel	Забезпечує роботу ключового функціоналу із відображенням скарг на мапі та створенням нових скарг.
5	ProfileModel	Перегляд інформації про обліковий запис користувача, а також вихід з нього.
6	TokenManager	Забезпечує ключовий функціонал по обробці токенів на клієнті.
7	LocationHelper	Забезпечує визначення розташування користувача.
8	PhotoHelper	Забезпечує коректну роботу із фото для скарг.

Другорядні класи та класи реалізації View та ViewModel не були згадані, оскільки вони не відіграють значної ролі в бізнес логіці.

Серверна частина застосунку розбита на модульний моноліт з якого можна виділити основні компоненти, представлені у таблиці 3.4.

Таблиця 3.4 – Опис основних компонентів серверної частини

№	Ім’я компоненти	Опис компоненти
1	AuthService	Сервіс обробки авторизації та аутентифікації користувачів.
2	UserService	Сервіс обробки запитів, які стосуються даних користувачів.
3	ReportService	Сервіс обробки запитів пов’язаних із скаргами користувачів. Забезпечує фільтрацію, обмеження повернутих значень.
4	JWTUtil	Відповідає за роботу із токенами авторизації та аутентифікації (створення, зберігання, читання, оновлення, видалення, порівняння).

Як було раніше підмічено, у якості бази даних для серверної частини застосунку було обрано PostgreSQL із розширенням PostGIS, яке дозволило полегшити роботу із локаціями корисутвачів та скарг. Для кращого розуміння структури бази даних, було спроєктовано ER-діаграму (рис. 3.2), а детальний опис усіх сутностей наведено у додатку В, у таблицях В.1-В.8.

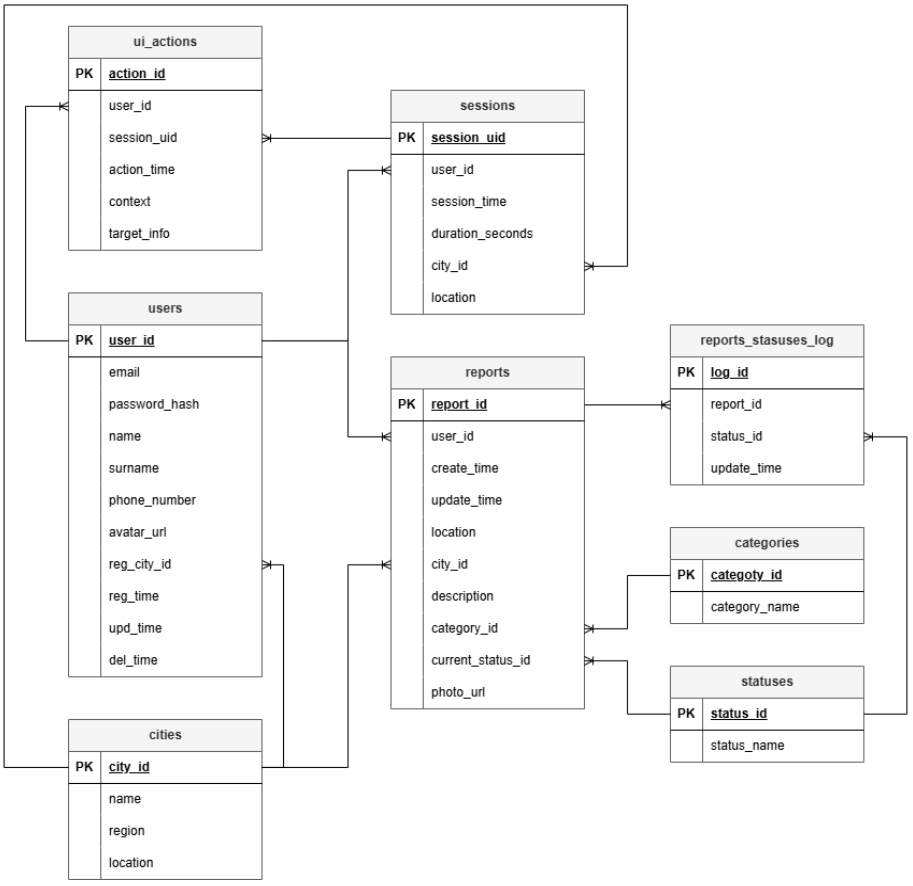

Рисунок 3.2 – ER-діаграма сутностей бази даних

Опис усіх програмних засобів, які були залученні під час створення програмного застосунку, наведено в таблиці 3.5.

Таблиця 3.5 – Опис утиліт

№ п/п	Назва утиліти	Опис застосування
1	Android Studio	Використовується для написання коду клієнтської частини, проектування інтерфейсу, тестування на емуляторах та зневадження (дебаггінгу) програми.
2	GoLand	Спеціалізоване середовище розробки від JetBrains для мови програмування Go. Застосовується для розробки серверної частини (бекенду), забезпечує зручну навігацію по коду, автоматичний рефакторинг та інструменти для роботи з базами даних.
3	Firebase console	Веб-інтерфейс для керування хмарними сервісами платформи Google Firebase. Використовується для налаштування роботи із Firebase Storage.
4	Postman	Інструмент для тестування та документування API. Дозволяє надсилати HTTP-запити до сервера, перевіряти коректність відповідей, налагоджувати взаємодію між клієнтом та сервером, а також створювати колекції тестових запитів.
5	Docker	Платформа для контейнеризації програмного забезпечення. Використовується для пакування застосунку та його залежностей у контейнери, що забезпечує швидке розгортання та однакове функціонування сервісів у різних середовищах.
6	Github	Веб-сервіс для хостингу репозиторіїв та спільної розробки на базі системи контролю версій Git. Забезпечує зберігання вихідного коду, керування версіями проекту та відстеження змін/
7	diagrams.net (draw.io)	Графічний редактор для створення технічної документації. Використовується для побудови UML-діаграм, схем архітектури додатку, блок-схем алгоритмів та ER-діаграм баз даних.

У додатку Д додано посилання на репозиторії із програмним кодом

3.4 Аналіз безпеки даних

Безпека даних є одним із ключових аспектів під час розробки інформаційної системи для моніторингу та фіксації міських проблем,

оскільки застосунок опрацьовує облікові дані користувачів, їхні фотографії, координати місць та іншу чутливу інформацію. Важливо забезпечити захист від несанкціонованого доступу, модифікації або перехоплення даних під час передачі. Для мінімізації ризиків проведено аналіз можливих вразливостей та визначено методи їх усунення. Отримані результати подано у таблицях 3.6 – 3.9.

Таблиця 3.6 – Вразливість аутентифікації та авторизації

Вразливість	Аутентифікація та авторизація через JWT
Загроза	Отримання доступу до облікових записів шляхом компрометації токенів або їх повторного використання
Виконане рішення	Використано пару токенів (access + refresh), ротацію refresh-токенів, шифрування та зберігання токенів у EncryptedSharedPreferences на клієнті; створено чорний список токенів у Redis на сервері
Варіанти покращення	Додати двофакторну аутентифікацію; зменшити час життя access-токена; реалізувати детекцію підозрілих сесій

Таблиця 3.7 – Вразливість безпеки зберігання даних

Вразливість	Зберігання персональних даних та звітів
Загроза	Несанкціонований доступ до бази даних, модифікація інформації або витік фото й координат
Виконане рішення	Усі дані зберігаються у захищеній базі PostgreSQL з обмеженнями доступу; фото зберігаються у Firebase Storage з правилами доступу; сервер обмежує доступ до API через JWT
Варіанти покращення	Додаткове шифрування файлів на стороні клієнта; введення політик резервного копіювання та моніторингу доступів

Таблиця 3.8 – Вразливість передачі даних

Вразливість	Передача даних між клієнтом та сервером
Загроза	Перехоплення персональної інформації під час взаємодії з сервером чи Firebase Storage
Виконане рішення	Усі HTTP-запити виконуються через протокол HTTPS; використано TLS-шифрування, а дані передаються у форматі JSON поверх захищеного каналу
Варіанти покращення	Використання додаткового шифрування payload; впровадження механізму TLS-pinning для захисту від MITM-атак

Таблиця 3.9 – Вразливість геолокаційних даних

Вразливість	Робота з координатами та геопозицією
Загроза	Несанкціонований доступ до точних координат користувачів або маніпуляція геоданими
Виконане рішення	Геодані зберігаються у PostGIS із обмеженими правами доступу; картографічний функціонал працює виключно на клієнті через Maps SDK; сервер перевіряє коректність координат та їх приналежність до міста
Варіанти покращення	Анонімізація координат; перевірка відхилень GPS; обмеження частоти надсилання локації

Так як наразі це лише прототип застосунку, наявних методів захисту більш ніж достатньо.

Висновки до розділу

У результаті виконання даного розділу кваліфікаційної роботи було проведено розробку архітектури та створено програмне забезпечення інформаційної системи для моніторингу та фіксації міських проблем. Детально розглянуто структурний підхід до побудови мобільного застосунку, що ґрунтується на архітектурному патерні MVVM, який забезпечує розділення логіки представлення, бізнес-логіки та управління даними. На основі даного патерну побудовано діаграму пакетів, що

демонструє взаємозв’язки між основними структурними елементами застосунку.

Для серверної частини було обрано шарову архітектуру, яка включає контролери, сервіси та репозиторії. Таке рішення дало можливість забезпечити передбачуваність виконання бізнес-процесів, підвищити гнучкість і масштабованість системи, а також спростити подальшу підтримку бекенд-частини.

У проєкті визначено та обґрунтовано використаний технологічний стек, який включає Android Studio як основне IDE для клієнта, мову програмування Java для мобільного застосунку та Go (Golang) для серверної частини. Також описано використання Firebase Storage для зберігання медіафайлів і PostgreSQL із розширенням PostGIS для роботи з геопросторовими даними. Для передачі даних до представників ОМС буде розроблено зовнішні програмний інтерфейс (REST API), який вони зможуть самостійно інтегрувати у наявні системи, а також WEB-інтерфейс для використання одразу. Окрему увагу приділено роботі Maps SDK for Android, який використано для відображення картографічних даних та взаємодії користувача з геолокаційною інформацією.

У рамках реалізації програмного забезпечення було створено множину модулів та класів, відповідальних за авторизацію, роботу з мережею, обробку запитів, керування геолокацією, роботу зі звітами та доступ до бази даних. Для основних сутностей побудовано ER-діаграму та наведено детальний опис їхніх атрибутів.

З метою забезпечення надійності програмного забезпечення проведено оцінку вразливостей, пов’язаних із автентифікацією, обробкою даних, їх передачею та зберіганням. Для кожної потенційної вразливості визначено загрозу, реалізоване рішення та можливі варіанти покращень. Аналіз показав, що всі критичні ризики були усунені шляхом використання надійних методів аутентифікації, захищених каналів передачі даних, контрольованого доступу до бази даних та сучасних механізмів безпеки.

РОЗДІЛ 4 ТЕСТУВАННЯ, РОЗГОРТАННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Опис процесів тестування

Під час розробки серверної частини проєкту, були також реалізовані модульні тести для окремих методів, а також інтеграційні тести для перевірки коректності роботи HTTP запитів.

Модульне тестування (Unit Testing) — це рівень тестування програмного забезпечення, що передбачає перевірку правильності роботи окремих, найменших атомарних одиниць вихідного коду (модулів, функцій, методів або класів) в ізольованому середовищі [23]. Основною метою цього процесу є верифікація відповідності поведінки окремих компонентів системи закладеним функціональним вимогам та бізнес-логіці до моменту їх інтеграції в загальну систему.

Покриті модулі інформаційної системи коректно працюють, відповідаючи на поставлені перед ними функціональні вимоги. Результати проходження модульних тестів можна побачити на рис. 4.1.

```
make test-unit-short
"Running unit tests (simplified output)..."
ok      github.com/serediukit/civix-backend/internal/services coverage: 64.9% of statements
ok      github.com/serediukit/civix-backend/pkg/jwt           coverage: 82.1% of statements
ok      github.com/serediukit/civix-backend/pkg/hash          coverage: 100.0% of
statements
```

Рисунок 4.1 – Результати виконання модульних тестів.

Інтеграційне тестування (Integration Testing) — це етап контролю якості програмного забезпечення, на якому окремі, попередньо перевірені програмні модулі об’єднуються та тестуються як єдина сукупність [24]. Основною метою цього процесу є верифікація коректності взаємодії між компонентами системи, перевірка дотримання контрактів інтерфейсів, а також виявлення дефектів, що виникають при передачі даних між різними підсистемами (наприклад, між бізнес-логікою та базою даних або

зовнішніми API). Цей рівень тестування логічно слідує за модульним і передує системному тестуванню.

Покриті інтеграційними тестами функціонал повністю відповідає очікуванням. Результати проходження інтеграційних тестів можна побачити на рис. 4.2.

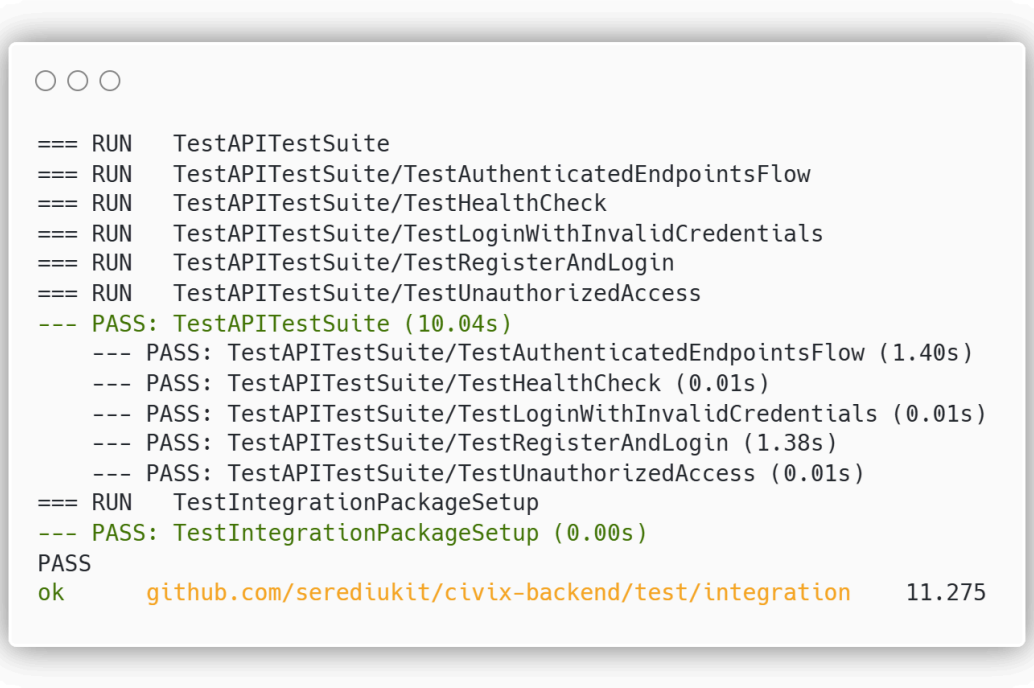

Рисунок 4.2 – Результати виконання інтеграційних тестів.

Для клієнтської частини ІС, а саме для мобільного застосунку, було проведене ручне тестування.

Ручне тестування (Manual Testing) — це метод верифікації та валідації програмного забезпечення, що передбачає безпосереднє виконання тестових сценаріїв фахівцем (QA-інженером) без застосування засобів автоматизації [25]. Ключовою особливістю цього підходу є емуляція поведінки кінцевого користувача, що дозволяє оцінити не лише функціональну коректність системи, але й характеристики зручності використання, візуальну цілісність інтерфейсу та загальний користувацький досвід. Ручне тестування відіграє вирішальну роль при проведенні дослідницького аналізу та перевірці сценаріїв, які є економічно недоцільними або технічно складними для автоматизації.

Клієнтська частини повністю відповідає поставленими перед нею функціональним та нефункціональним вимогам. Опис основної частини тестів наведено у додатку Г, у таблицях Г.1 – Г.10.

4.2 Розгортання програмного забезпечення

Для забезпечення надійності, масштабованості та переносності серверної частини системи було обрано технологію контейнеризації Docker.

Docker — це програмна платформа для автоматизації розгортання, масштабування та управління додатками, що використовує метод віртуалізації на рівні операційної системи (OS-level virtualization) [26]. На відміну від апаратної віртуалізації (де гіпервізор емулює повний стек апаратного забезпечення), Docker використовує спільне ядро хост-системи, забезпечуючи ізоляцію процесів у просторі користувача.

Використання контейнерів дозволяє вирішити класичну проблему сумісності, гарантуючи, що програмне забезпечення буде працювати ідентично в будь-якому середовищі — від локального ноутбука розробника до виробничого сервера.

Переваги використання Docker у проєкті:

1. Ізоляція: Кожен сервіс (додаток, база даних, кеш) працює у власному ізольованому середовищі з чітко визначеними ресурсами та доступами.
2. Керування залежностями: Усі необхідні бібліотеки та версії мов програмування «запаковані» всередину образу, що виключає конфлікти версій у системі.
3. Легковажність: На відміну від віртуальних машин, контейнери використовують спільне ядро ОС, що значно знижує споживання ресурсів [26].

Для опису процесу побудови образу серверного застосунку розроблено Dockerfile, який використовує технологію multi-stage build (багатоетапна збірка):

1. Етап збірки (Builder stage): Використовується образ **`golang:1.23-alpine`**. На цьому етапі відбувається завантаження залежностей (**`go mod download`**) та компіляція вихідного коду у бінарний файл. Важливою деталлю є встановлення прапорів **`CGO_ENABLED=0`** та **`-ldflags="-s -w"`**, що дозволяє створити статично скомпонований, оптимізований файл без зайвої налагоджувальної інформації.
2. Фінальний етап (Final stage): Використовується мінімалістичний образ **`alpine:3.18`**, в який копіюється лише скомпільований бінарний файл. Це дозволяє зменшити розмір кінцевого образу та підвищити безпеку, оскільки у фінальному контейнері відсутні компілятори та вихідний код.

Для оркестрації (управління взаємодією) сервісів використовується інструмент Docker Compose. У файлі конфігурації docker-compose.yml визначено три взаємопов'язані сервіси:

1. **`app`**: Основний серверний застосунок. Він налаштований на роботу через порт 8443 та використовує змінні середовища для підключення до бази даних та Redis. Для забезпечення безпеки передачі даних реалізовано підтримку протоколу TLS/SSL, сертифікати для якого монтуються безпосередньо у контейнер застосунку.
2. **`db`**: Система управління базами даних PostgreSQL. Оскільки проєкт потребує роботи з геоданими (мапи, координати), використовується спеціалізований Dockerfile_postgis, який встановлює розширення PostGIS поверх стандартного образу **`Postgres 16`**.
3. **`redis`**: Сервіс для кешування та обробки сесій, що працює на базі образу **`redis:7-alpine`**.

Усі сервіси об'єднані у внутрішню мережу **network**, що дозволяє їм взаємодіяти за назвами контейнерів, не виставляючи внутрішні порти назовні (крім необхідних). Дані бази даних та Redis зберігаються у персистентних сховищах (**volumes**), що гарантує збереження інформації після перезапуску контейнерів. Також для підвищення відмовостійкості налаштовано механізми перевірки працездатності для бази даних та кешу, які дозволяють автоматично відстежувати стан сервісів та перезапускати їх у разі зависання.

Процес розгортання серверної частини: Для запуску системи на сервері або локальній машині необхідно виконати наступну команду в терміналі (в директорії проєкту):

```
docker-compose up -d
```

Ця команда виконує наступні дії:

1. Збирає образи для застосунку та бази даних згідно з Dockerfile.
2. Створює віртуальну мережу та томи для даних.
3. Запускає контейнери у фоновому режимі (прапорець **-d**), забезпечуючи автоматичний перезапуск у разі збою.
4. Після чого застосунок стає доступним за адресою **localhost:8443**.

Для розповсюдження клієнтської частини системи створюється інсталяційний пакет формату .apk (Android Package Kit). Цей процес включає компіляцію вихідного коду мобільного застосунку, пакування ресурсів (зображень, шрифтів) та підписання пакету ключем розробника.

Алгоритм підготовки клієнта мобільного застосунку:

1. Конфігурація мережі: Оскільки мобільний телефон та сервер розробки знаходяться на різних пристроях, застосунок не може використовувати адресу **localhost**. Для коректного з’єднання необхідно визначити локальну IP-адресу комп’ютера розробника в мережі (наприклад, через команду ipconfig у Windows або ifconfig у Linux/macOS). Отриману адресу необхідно прописати у конфігураційному файлі мобільного застосунку як **BASE_URL**.

Важливо, щоб сервер та мобільний пристрій були підключені до однієї локальної мережі (Wi-Fi).

2. Генерація .apk файлу: За допомогою Android Studio використовується готовий інструмент для збірки .apk файлів (рис. 4.3).

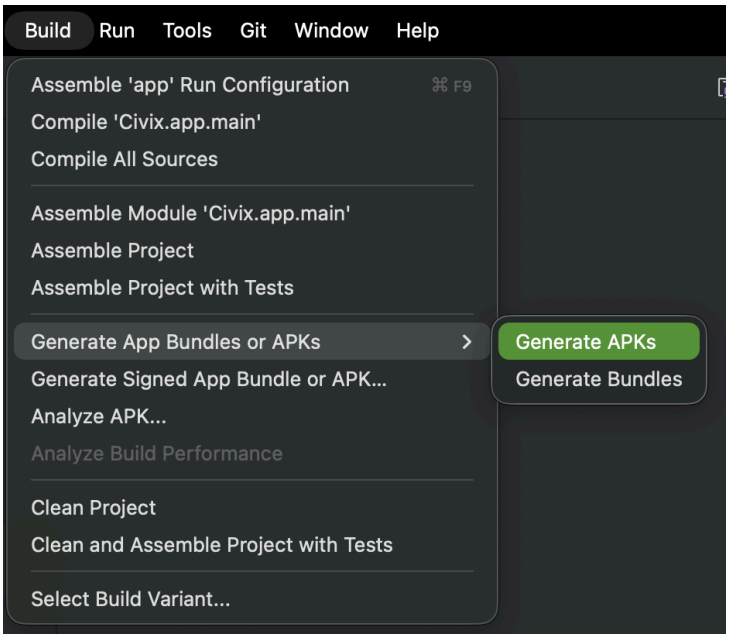

Рисунок 4.3 – Створення .apk файлу засобами Android Studio

Після успішної збірки проєкту, .apk файл можна знайти у директиві **`..\app\build\outputs\apk\debug\`**.

3. Встановлення: Файл передається на смартфон через USB-кабель або хмарне сховище та інсталюється стандартними засобами ОС Android. Якщо ОС Android 11+ версії, можна використати вбудований інструмент для Android Studio для під’єднання мобільного пристрою та запуску застосунку.
4. Використання емулятора: У разі використання Android Emulator від Android Studio, для доступу до локального хоста комп'ютера використовується спеціальна зарезервована адреса **`10.0.2.2`**, яка автоматично перенаправляє запити на **`localhost`** хост-машини.

У разі успішного виконання усіх пунктів для розгортання серверної та клієнтської частини інформаційної системи, мобільний застосунок дозволить отримати доступ до серверу та використовувати його функціонал. Також, для спрощення процесів збору мобільного застосунку,

у репозиторії знаходиться остання версія .apk файлу, у якому усі запити йдуть на адресу **`10.0.2.2`**.

4.3 Супровід програмного забезпечення

Забезпечення актуальності та стабільності роботи інформаційної системи реалізується через використання централізованої системи контролю версій Git. Вихідний код як серверної частини, так і клієнтського мобільного застосунку зберігається у віддалених репозиторіях, посилання на які вказано у додатку Д. Такий підхід гарантує цілісність історії змін, можливість командної розробки та швидке розгортання виправлень.

Процес супроводу та встановлення оновлень розділено на два напрямки відповідно до архітектури системи.

1. Оновлення серверної частини: Оскільки серверне середовище розгорнуто за допомогою технології контейнеризації Docker, процес оновлення є стандартизованим та мінімізує ризик виникнення конфліктів конфігурації. Алгоритм отримання оновлень на сервері виглядає наступним чином:
    a. Синхронізація коду: Виконується актуалізація локальної копії репозиторію шляхом отримання останніх змін з головної гілки за допомогою команди **`git pull`**.
    b. Перезбірка контейнерів: Оскільки зміни можуть стосуватися як логіки програми, так і конфігурації інфраструктури, ініціюється примусова перезбірка образів Docker. Використання прапорця **`--build`** у команді **`docker-compose up`** гарантує, що бінарні файли будуть перекомпільовані з урахуванням нових змін у коді.
    c. Перезапуск сервісів: Оновлені контейнери автоматично замінюють попередні версії без втрати даних, що зберігаються у персистентних сховищах.

2. Оновлення клієнтського застосунку: Для мобільного застосунку процес оновлення передбачає генерацію нової версії інсталяційного пакету.
    a. Отримання змін: Розробник або адміністратор завантажує оновлений вихідний код проєкту в середовище Android Studio (**`git pull`**).
    b. Вирішення залежностей: Система Gradle автоматично перевіряє та завантажує нові версії бібліотек, якщо зміни було внесено у файл build.gradle.
    c. Компіляція релізу: Ініціюється процес збірки нового .apk файлу.
    d. Розгортання на пристрої: Користувачам надається оновлений файл для встановлення. При цьому операційна система Android виконує оновлення існуючого застосунку зі збереженням локальних даних користувача (кеш, налаштування), за умови, що цифровий підпис нового файлу співпадає з попереднім.

Висновки до розділу

У цьому розділі було проведено комплексний аналіз процесів тестування, розгортання та подальшого супроводу розробленого програмного забезпечення. На етапі тестування серверної частини виконано модульні та інтеграційні тести, що підтвердили коректність роботи окремих компонентів і взаємодію між ними. Результати тестування показали стабільність алгоритмів обробки даних, відповідність бізнес-логіці та відсутність критичних помилок під час виконання HTTP-запитів.

Клієнтська частина системи була перевірена за допомогою ручного тестування. Це дозволило оцінити поведінку застосунку з позиції кінцевого користувача, перевірити основні сценарії взаємодії та переконатися у відповідності функціональних і нефункціональних вимог. Усі протестовані

сценарії продемонстрували очікувані результати, що підтверджує коректність роботи мобільного застосунку.

Також детально описано підхід до розгортання серверної частини за допомогою Docker та Docker Compose. Обрані технології дозволили забезпечити ізоляцію сервісів, контроль залежностей, масштабованість та однакові умови виконання у будь-якому середовищі. Використання багатоетапної збірки образу суттєво зменшило розмір кінцевого контейнера й підвищило безпеку. Окремо налаштовано роботу з PostGIS, Redis, TLS-сертифікатами та персистентними сховищами даних, що забезпечує надійність і відмовостійкість системи.

У результаті виконання цього розділу підтверджено працездатність усіх ключових компонентів інформаційної системи, забезпечено можливість стабільного розгортання та визначено підходи до подальшої підтримки. Це формує основу для надійної експлуатації програмного забезпечення в реальних умовах та подальшого масштабування системи.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було створено повноцінну інформаційну систему для моніторингу та фіксації міських проблем із можливістю передачі даних до органів місцевого самоврядування. Система об’єднує мобільний застосунок та серверну частину, забезпечуючи повний цикл роботи зі зверненнями — від створення скарги користувачем до її подальшої обробки та відслідковування статусу.

На етапі передпроєктного аналізу були досліджені існуючі рішення, визначені їхні недоліки та сформульована потреба у системі, що поєднує фотофіксацію, геолокацію, прозорий облік звернень та зручну взаємодію мешканців із муніципальними службами. На основі дослідження було сформовано мету та завдання роботи, що охоплюють аналіз доменної області, розроблення вимог, проєктування архітектури та реалізацію програмного забезпечення.

У ході розроблення вимог було створено діаграму варіантів використання, сформовано перелік функціональних та нефункціональних вимог, а також побудовано матрицю трасування, що забезпечила повне покриття потреб користувачів та логічну цілісність системи.

Під час конструювання програмного забезпечення спроєктовано клієнт-серверну архітектуру, визначено структуру бази даних із підтримкою геопросторових даних (PostGIS), реалізовано мобільний застосунок на базі архітектурного патерну MVVM та розроблено серверну частину на Go з використанням JWT-аутентифікації, Redis та PostgreSQL. Для передачу даних до представників ОМС було розроблено зовнішній програмний інтерфейс для інтеграфій ІС у наявну систему ІТ-інфраструктури міста, а також WEB-інтерфейс для простого використання. Було проведено аналіз безпеки та впроваджено механізми захисту, що усувають критичні ризики, пов’язані з обробкою даних та автентифікацією.

Тестування системи охопило модульні та інтеграційні тести серверної частини, а також ручне тестування мобільного застосунку. Усі сценарії продемонстрували коректну роботу, а стабільність взаємодії між компонентами системи підтвердила правильність архітектурних рішень.

Для забезпечення можливості розгортання та супроводу була реалізована контейнеризація серверної частини за допомогою Docker та Docker Compose, налаштована робота з PostGIS і Redis та визначено повний цикл оновлення як клієнтської, так і серверної частини. Мобільний застосунок може бути встановлений на пристрої за допомогою сформованого .apk файлу, що спрощує процес розповсюдження.

Розроблена інформаційна система забезпечує оперативний збір і передачу достовірних даних про міські проблеми, підвищує ефективність взаємодії між громадянами та органами місцевого самоврядування і створює передумови для більш прозорої та результативної роботи муніципальних служб. Отримані результати мають практичну цінність і можуть бути використані для подальшого масштабування та інтеграції системи в інші міські сервіси.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Юдкова К. Особливості визначення поняття “інформаційна система”. Інформація і право. 2015. Т. 2, № 14. С. 39–44. URL: https://ippi.org.ua/sites/default/files/ykvovpis_14_2_2015.pdf.
2. Мобільні застосунки: їх види та особливості. Highload.tech - медіа для розробників. URL: https://highload.tech/uk/mobilni-zastosunki-yih-vidi-ta-osoblivosti/.
3. Вебзастосунок – Вікіпедія. URL: https://uk.wikipedia.org/wiki/Вебзастосунок.
4. Права людини в Україні: регіональний вимір – Щорічна доповідь Уповноваженого Верховної Ради України з прав людини - 2024 рік. Уповноважений Верховної Ради України з прав людини. URL: https://ombudsman.gov.ua/report-2024/prava-liudyny-v-ukraini-rehionalnyi-vymir#khmelnytska-oblast.
5. Mobile Operating System Market Share Worldwide | Statcounter Global Stats. StatCounter Global Stats. URL: https://gs.statcounter.com/os-market-share/mobile/worldwide.
6. Mobile & Tablet Android Version Market Share Worldwide | Statcounter Global Stats. StatCounter Global Stats. URL: https://gs.statcounter.com/android-version-market-share/mobile-tablet/worldwide.
7. Android Architecture Patterns | Dashwave. Android Cloud Builds & Collaborative Emulators | Dashwave. URL: https://dashwave.io/blog/android-architecture-patterns.
8. Boudjnah E. Clean Architecture for Android: Implement Expert-led Design Patterns to Build Scalable, Maintainable, and Testable Android Apps (English Edition). BPB Publications, 2022. 262 p.
9. Android Architecture Patterns | Dashwave. Android Cloud Builds & Collaborative Emulators | Dashwave. URL: https://dashwave.io/blog/android-architecture-patterns.

10. Клієнт-серверна архітектура та ролі серверів. Medium. URL: https://medium.com/@IvanZmerzlyi/клієнт-серверна-архітектура-та-ролі-серверів-9893d8048229.
11. IBM. What Is a REST API (RESTful API)? | IBM. IBM. URL: https://www.ibm.com/think/topics/rest-apis.
12. PostgreSQL: About. PostgreSQL: The world's most advanced open source database. URL: https://www.postgresql.org/about/.
13. What are ACID Transactions?. Databricks. URL: https://www.databricks.com/glossary/acid-transactions.
14. Firebase | Google's Mobile and Web App Development Platform. Firebase. URL: https://firebase.google.com/.
15. Maps SDK for Android overview | Google for Developers. Google for Developers. URL: https://developers.google.com/maps/documentation/android-sdk/overview
16. Android Developers. URL: https://developer.android.com/tools/releases/platform-tools.
17. Android SDK. Wikipedia. URL: https://en.wikipedia.org/wiki/Android_SDK.
18. JetBrains. IntelliJ IDEA – the Leading Java and Kotlin IDE. JetBrains. URL: https://www.jetbrains.com/idea/.
19. ADT Plugin for Eclipse | Android Developers. Distributed Object Computing (DOC) Group for DRE Systems. URL: https://www.dre.vanderbilt.edu/~schmidt/android/android-4.0/out/target/common/docs/doc-comment-check/sdk/eclipse-adt.html.
20. Java documentation. Oracle. URL: https://docs.oracle.com/en/java/.
21. Kotlin Docs. Kotlin Help. URL: https://kotlinlang.org/docs/home.html.
22. Documentation - The Go Programming Language. The Go Programming Language. URL: https://go.dev/doc/.
23. Powell P., Smalley I. What is Unit Testing? | IBM. IBM. URL: https://www.ibm.com/think/topics/unit-testing.

24. Powell P., Smalley I. What is Integration Testing? | IBM. IBM. URL: https://www.ibm.com/think/topics/integration-testing.
25. Contributors to Wikimedia projects. Manual testing - Wikipedia. Wikipedia, the free encyclopedia. URL: https://en.wikipedia.org/wiki/Manual_testing.
26. Home. Docker Documentation. URL: https://docs.docker.com/.

ДОДАТКИ

Додаток А

Варіанти використання інформаційної системи

Таблиця А.1 – Варіант використання UC-1

Use case name	Реєстрація користувача
Use case ID	UC-01
Goals	Реєстрація нового користувача в системі
Actors	Неавторизований користувач
Trigger	Користувач бажає зареєструватися
Pre-conditions	Неавторизований користувач завантажив та відкрив мобільний застосунок.
Flow of Events	При відкритті застосунку, відображається екран входу в обліковий запис. Користувач обирає пункт “Реєстрація”, що переносить користувача на сторінку реєстрації. У відповідні поля користувач вносить свої дані (електронна пошта, пароль, ім’я та прізвище, номер телефону), після чого натискає кнопку “Зареєструватись”. Користувач потрапляє на сторінку входу в обліковий запис та після введення електронної пошти та паролю, які він попередньо зареєстрував, натискає кнопку “Вхід” та потрапляє на головний екран мобільного застосунку.
Extension	У випадку введення електронної пошти або ж номеру телефону, для яких уже існують облікові записи, користувач буде проінформований, а реєстрація скасується. У випадку залишення обов’язкових полів порожніми - вони підсвітяться для інформування користувача стосовно необхідності їх заповнення.
Post-Condition	Користувач реєструється в системі та перенаправляється на головну сторінку.

Таблиця А.2 – Варіант використання UC-2

Use case name	Авторизація користувача
Use case ID	UC-02

Goals	Авторизація існуючого користувача в систему
Actors	Неавторизований користувач
Trigger	Користувач бажає авторизуватись
Pre-conditions	У користувача вже є створений обліковий запис, а також завантажений застосунок.
Flow of Events	При відкритті застосунку, користувач потрапляє на сторінку авторизації. У відповідні поля необхідно обов’язково ввести електронну пошту та пароль до облікового запису із цією поштою. Після натискання кнопки “Вхід” користувач буде перенесений на головну сторінку.
Extension	У випадку введення не відповідних даних для пари значень електронна пошта - пароль, вхід буде невдалим. У випадку залишення одного з полів порожнім - воно підсвітиться з нагадуванням, що поля є обов’язковими для введення.
Post-Condition	Користувач авторизується в системі та перенаправляється на головну сторінку.

Таблиця А.3 – Варіант використання UC-3

Use case name	Вихід з облікового запису
Use case ID	UC-03
Goals	Вийти з облікового запису користувача
Actors	Авторизований користувач
Trigger	Користувач бажає вийти з свого облікового запису
Pre-conditions	У користувача вже є створений обліковий запис, а також виконано вхід у нього.
Flow of Events	Користувач переходить на сторінку свого профілю, після чого натискає клавішу “Вийти”.
Extension	-
Post-Condition	Користувач покидає сторінку профілю та автоматично переноситься на сторінку авторизації для входу в інший, або ж цей самий, обліковий запис.

Таблиця А.4 – Варіант використання UC-4

Use case name	Перегляд існуючих міток
Use case ID	UC-04
Goals	Переглянути наразі створенні мітки усіма користувачами у певному радіусі.
Actors	Авторизований користувач
Trigger	Користувач переходить на головний екран.
Pre-conditions	Користувач виконав вхід в обліковий запис, надав дозвіл на використання розташування та увімкнув розташування.
Flow of Events	Користувач потрапляє на головну сторінку застосунку, на якій він бачить мапу з своєю міткою та мітками, які залишили інші користувачі у цьому ж місці.
Extension	Відображаються лише актуальні мітки (видалені відсутні), а також для оптимізації додано обмеження на 100 одночасних міток, відсортованих по відстані до користувача. Колір міток залежить від типу скарги, яку залишив інший користувач (дорога, електрика, вода і т.д.) та від статусу скарг (нова, вирішена, скасована і т.д.).
Post-Condition	Користувач переглядає мапу із присутніми на ній мітками щодо наявних скарг.

Таблиця А.5 – Варіант використання UC-5

Use case name	Перегляд інформації про конкретну мітку
Use case ID	UC-05
Goals	Переглянути наявну інформацію про створену мітку
Actors	Авторизований користувач
Trigger	Користувач натискає на мітку на мапі.
Pre-conditions	Користувач успішно провантажив мапу із сусідніми для нього мітками скарг.
Flow of Events	Користувачу відображається діалогове вікно з інформацією про скаргу, яка була залишена на певному місці на мапі.

Extension	За наявності фото прикріпленого до мітки - воно теж буде відображене.
Post-Condition	Користувач переглядає інформацію про обрану ним мітку.

Таблиця А.6 – Варіант використання UC-6

Use case name	Створення нової мітки
Use case ID	UC-06
Goals	Додати нову скаргу на мапу для подальшої передачі інформації органам місцевого самоврядування.
Actors	Авторизований користувач
Trigger	Користувач натискає на кнопку з символом “+” на мапі.
Pre-conditions	Користувач успішно провантажив мапу із сусідніми для нього мітками скарг.
Flow of Events	Користувачу відображається діалогове вікно з необхідною для введення інформацією при створенні нової мітки. Після чого користувач натискає кнопку “Відправити” створення нової скарги.
Extension	При відправці скарги буде автоматично визначене розташування користувача, що дозволить встановити місце розташування для скарги на мапі. За наявності фото - користувач може прикріпити його до скарги. Користувач може обрати одну з категорій для мітки, що спростить процес фільтрації міток у майбутньому.
Post-Condition	Після успішної відправки, користувач побачить нову мітку на мапі поблизу свого маркера.

Таблиця А.7 – Варіант використання UC-7

Use case name	Визначення поточного розташування користувача
Use case ID	UC-07
Goals	Дізнатись де зараз знаходиться користувач на мапі
Actors	Авторизований користувач
Trigger	Користувач натискає на кнопку з символом локації на мапі.

Pre-conditions	Користувач успішно провантажив мапу із сусідніми для нього мітками скарг.
Flow of Events	Мапа плавно переміщується до розташування користувача.
Extension	-
Post-Condition	Користувач бачить де саме він знаходиться та які мітки скарг є поруч із ним.

Таблиця А.8 – Варіант використання UC-8

Use case name	Перегляд інформації про свій профіль
Use case ID	UC-08
Goals	Переглянути інформацію про користувача, у чий обліковий запис було виконано вхід.
Actors	Авторизований користувач
Trigger	Користувач натискає на кнопку з символом людини на мапі.
Pre-conditions	Користувач успішно провантажив мапу із сусідніми для нього мітками скарг.
Flow of Events	Користувачу переносить на сторінку профіля, де він може дізнатись ключову інформацію про свій обліковий запис.
Extension	-
Post-Condition	-

Таблиця А.9 – Варіант використання UC-9

Use case name	Перегляд існуючих міток
Use case ID	UC-09
Goals	Переглянути наразі створенні мітки усіма користувачами у певному місті.
Actors	Працівник органів місцевого самоврядування.
Trigger	Користувач входить у систему.
Pre-conditions	Користувач виконав вхід в обліковий запис із правами для працівників органів місцевого самоврядування.
Flow of Events	Користувач авторизуції у веб-застоснку та потрапляє на головну сторінку, на якій зображені скарги у вигляді

	списку, з можливістю фільтрації їх по категоріях та статусах, а також сортуванні по даті надходження скарги.
Extension	Відображаються лише актуальні мітки (видалені відсутні), для оптимізації додано пагінацію.
Post-Condition	-

Таблиця А.10 – Варіант використання UC-10

Use case name	Оновлення стану звернення
Use case ID	UC-10
Goals	Змінити статус наявної скарги
Actors	Працівник органів місцевого самоврядування.
Trigger	Скаргу було усуното або ж вирішено проблему іншим чином.
Pre-conditions	Користувач виконав вхід в обліковий запис із правами для працівників органів місцевого самоврядування.
Flow of Events	Працівник органів місцевого самоврядування обирає скаргу, для якої необхідно змінити статус, а також обирає новий статус, який буде їх привласнений.
Extension	Відображається лише ті статуси, які можуть бути надані скарзі (наприклад статус “Новий” не може бути наданий будь-якій скарзі).
Post-Condition	Статус скарги змінюється в усіх користувачів.

Додаток Б

Загальна модель вимог та список функціональних вимог

Таблиця Б.1 – Загальна модель вимог

№	Назва	ID вимоги	Пріоритети	Ризики
1	Система авторизації	FR-1	Високий	Високий
1.1	Реєстрація користувача	FR-2	Високий	Високий
1.2	Вхід в обліковий запис	FR-3	Високий	Середній
1.3	Вихід з облікового запису	FR-4	Середній	Низький
2	Система роботи з профілем	FR-5	Середній	Низький
2.1	Перегляд інформації про свій профіль	FR-6	Середній	Низький
3	Система роботи із скаргами	FR-7	Високий	Середній
3.1	Перегляд існуючих міток (на мапі)	FR-8	Середній	Низький
3.1.1	Фільтрування існуючих скарг	FR-9	Середній	Низький
3.1.2	Перегляд інформації про конкретну скаргу	FR-10	Середній	Низький
3.2	Створення нової скарги (мітки)	FR-11	Високий	Середній
3.2.1	Можливість обрати категорію зі списку	FR-12	Середній	Низький
3.2.2	Додавання фото до мітки	FR-13	Середній	Низький
3.3	Визначення геолокації	FR-14	Середній	Середній
3.3.1	Автоматичне визначення розташування для мітки	FR-15	Середній	Середній
3.3.2	Визначення поточного місцерозташування (користувачем)	FR-16	Середній	Середній
4	Система обробки звернень	FR-17	Високий	Середній
4.1	Оновлення стану звернення	FR-18	Високий	Середній
5	Звітність	FR-19	Середній	Низький
5.1	Отримання щомісячної звітності	FR-20	Середній	Низький

Таблиця Б.2 – Перелік функціональних вимог

ID вимоги	Назва та опис
FR-1	Система авторизації Сервіс який надасть повноцінний функціонал для обробки авторизації та аутентифікації користувача.
FR-2	Реєстрація користувача Користувач повинен мати можливість створити новий обліковий запис, за умови наявності унікальної електронної пошти.
FR-3	Вхід в обліковий запис Неавторизований користувач може увійти у свій обліковий запис із початкової сторінки.
FR-4	Вихід з облікового запису Авторизований користувач може вийти із свого облікового запису із сторінки профілю та бути перенесений на початковий екран.
FR-5	Система роботи з профілем Сервіс отримання інформації про профіль користувачів.
FR-6	Перегляд інформації про свій профіль Авторизований користувач може переглянути інформацію про свій обліковий запис на сторінці профілю.
FR-7	Система роботи із скаргами Сервіс обробки звернень авторизованих користувачів. Забезпечує повний життєвий цикл роботи із скаргами.
FR-8	Перегляд існуючих міток (на мапі) На головному екрані застосунку, авторизований користувач може переглянути мапу навколишньої території з відміченими скаргами усіх користувачів.
FR-9	Фільтрування існуючих скарг Користувач не повинен бачити скарги, які були відхилені або виконані більше ніж N днів тому або ж скасовані іншими користувачами. Також потрібно реалізувати обмеження на одночасне відображення скарг для забезпечення оптимізації роботи системи.
FR-10	Перегляд інформації про конкретну скаргу Користувач може натиснути на мітку на карті, після чого отримає інформацію про скаргу, її статус, категорію, опис, фото (за наявності), час створення.
FR-11	Створення нової скарги (мітки) Користувач повинен мати можливість створити нову скаргу, до якої зможе внести необхідну інформацію

Продовження таблиці Б.2

ID вимоги	Назва та опис
FR-12	Можливість обрати категорію зі списку При створенні нової мітки користувач повинен мати можливість обрати одну з наперед визначених категорій для створення нової скарги.
FR-13	Додавання фото до мітки За необхідності - користувач може прикріпити фото до своєї скарги із своєї галереї (або ж іншого джерела медіаоб’єктів)
FR-14	Визначення геолокації Система повинна забезпечити отримання дозволів від користувача на визначення його геолокації.
FR-15	Автоматичне визначення розташування для мітки При створенні нової мітки, користувачу не потрібно вручну вказувати локацію скарги, так як вона буде автоматично визначена з його розташування.
FR-16	Визначення поточного місцерозташування (користувачем) Користувача може дізнатись своє розташування на мапі.
FR-17	Система обробки звернень Сервіс який дозволяє приймати рішення по скаргам користувачів, доступ до якого мають працівники ОМС.
FR-18	Оновлення стану звернення Працівник ОМС повинен мати можливість змінити статус звернення у своєму місті.
FR-19	Звітність Для працівників ОМС має бути можливість отримання даних для звітності.
FR-20	Отримання щомісячної звітності Щомісячна звітність агрегує дані скарг за останній місяць та надається для працівників ОМС.

Додаток В

Перелік та опис сутностей бази даних

Таблиця В.1 – Опис сутності cities

Назва поля	Тип даних	Опис
city_id	uuid PK	Унікальний ідентифікатор міста
name	varchar	Назва міста
region	varchar	Регіон (область) міста
location	geometry(POINT, 4326)	Розташування міста

Таблиця В.2 – Опис сутності users

Назва поля	Тип даних	Опис
user_id	serial PK	Унікальний ідентифікатор користувача
email	varchar	Електронна пошта користувача
password_hash	varchar	Захешированний пароль користувача
name	varchar	Ім’я користувача
surname	varchar	Прізвище користувача
phone_number	varchar	Мобільний номер телефону користувача
avatar_url	varchar	Посилання на аватарне фото користувача
reg_city_id	uuid FK	Ідентифікатор міста реєстрації користувача
reg_time	timestampz	Час реєстрації користувача
upd_time	timestampz	Час оновлення даних користувача
del_time	timestampz	Час видалення користувача

Таблиця В.3 – Опис сутності reports

Назва поля	Тип даних	Опис
report_id	uuid PK	Унікальний ідентифікатор скарги
user_id	int FK	Ідентифікатор користувача, що залишив скаргу
create_time	timestampz	Час створення скарги
update_time	timestampz	Час оновлення скарги
location	geometry(POINT, 4326)	Розташування скарги
city_id	uuid FK	Ідентифікатор міста розташування скарги
description	text	Опис скарги
category_id	int FK	Ідентифікатор категорії скарги
current_status_id	int FK	Ідентифікатор статусу скарги
photo_url	text	Посилання на фото скарги

Таблиця В.4 – Опис сутності reports_statuses_log

Назва поля	Тип даних	Опис
log_id	uuid PK	Унікальний ідентифікатор запису
report_id	uuid FK	Ідентифікатор скарги
status_id	int FK	Ідентифікатор статусу скарги
update_time	timestampz	Час оновлення скарги

Таблиця В.5 – Опис сутності categories

Назва поля	Тип даних	Опис
category_id	int PK	Унікальний ідентифікатор категорії
category_name	varchar	Назва категорії

Таблиця В.6 – Опис сутності statuses

Назва поля	Тип даних	Опис
status_id	int PK	Унікальний ідентифікатор статусу
status_name	varchar	Назва статусу

Таблиця В.7 – Опис сутності sessions

Назва поля	Тип даних	Опис
session_uid	uuid PK	Унікальний ідентифікатор сесії
user_id	int FK	Ідентифікатор користувача, що розпочав сесію
session_time	timestampz	Час початку сесії
duration_seconds	int	Час тривалості сесії
city_id	uuid FK	Ідентифікатор міста старту сесії
location	geometry(POINT, 4326)	Розташування користувача на момент старту сесії

Таблиця В.8 – Опис сутності ui_actions

Назва поля	Тип даних	Опис
action_id	uuid PK	Унікальний ідентифікатор дії
user_id	int FK	Ідентифікатор користувача, що здійснив дію
session_uid	uuid FK	Ідентифікатор сесії
action_dt	timestampz	Час створення дії
context	varchar	Контекст дії
target_info	varchar	Додаткова інформація про ціль дії

Додаток Г

Тестові випадки для мануального тестування мобільного застосунку

Таблиця Г.1 – Тест 1.1 Реєстрація користувача

Початковий стан системи	Користувач знаходиться на екрані входу в мобільний застосунок
Вхідні дані	Електронна пошта, пароль, ім’я, прізвище, номер телефону.
Опис проведення тесту	Користувач натискає кнопку “Зареєструватись” після чого потрапляє на екран реєстрації. Користувач вводить у відповідні поля свою електронну пошту, пароль для облікового запису, ім’я, прізвище та номер телефону. Після чого користувач натискає кнопку “Зареєструватись”.
Очікуваний результат	Користувач успішно реєструється в системі. Відкривається екран входу в обліковий запис.
Фактичний результат	Відповідає очікуваному.

Таблиця Г.2 – Тест 1.2 Авторизація користувача

Початковий стан системи	Користувач знаходиться на екрані входу в мобільний застосунок
Вхідні дані	Електронна пошта, пароль.
Опис проведення тесту	Користувач вводить електронну пошту та відповідний для неї пароль до облікового запису. Після чого натискає на кнопку “Вхід”.
Очікуваний результат	Користувач успішно виконує авторизацію в системі. Відкривається головний екран застосунку із мапою.
Фактичний результат	Відповідає очікуваному.

Таблиця Г.3 – Тест 1.3 Авторизація користувача із іншими даними.

Початковий стан системи	Користувач знаходиться на екрані входу в мобільний застосунок
Вхідні дані	Електронна пошта, пароль.
Опис проведення тесту	Користувач вводить електронну пошту та не відповідний для неї пароль до облікового запису. Після чого натискає на кнопку “Вхід”.
Очікуваний результат	З’являється повідомлення про некоректність даних для входу в обліковий запис.
Фактичний результат	Відповідає очікуваному.

Таблиця Г.4 – Тест 2.1 Відображення міток на мапі

Початковий стан системи	Користувач знаходиться на головному екрані з мапою
Вхідні дані	Відсутні
Опис проведення тесту	При потраплянні на головний екран, у користувача запитують дозвіл на отримання розташування. Користувач натискає “Надати доступ”.
Очікуваний результат	Автоматично визначається розташування, а мапа переноситься на місце розташування користувача, а також відображаються скарги, поблизу користувача.
Фактичний результат	Відповідає очікуваному.

Таблиця Г.5 – Тест 2.2 Перегляд існуючої мітки на мапі

Початковий стан системи	Користувач знаходиться на головному екрані з провантаженою мапою із мітками скарг.
Вхідні дані	Відсутні
Опис проведення тесту	Користувач обирає будь-яку мітку на мапі та натискає на неї.
Очікуваний результат	Відкривається діалогове вікно із детальною інформацією про мітку та фото (за наявності).
Фактичний результат	Відповідає очікуваному.

Таблиця Г.6 – Тест 2.3 Створення нової скарги

Початковий стан системи	Користувач знаходиться на головному екрані з провантаженою мапою із мітками скарг.
Вхідні дані	Категорія скарги, опис скарги, фото для скарги.
Опис проведення тесту	Користувач натискає клавішу із символом “+” у правому нижньому куту екрану, після чого відкривається діалогове вікно створення скарги. Користувач обирає відповідну категорію із випадаючого списку, вводить опис проблеми, а також прикріпляє до неї фото. Після чого натискає кнопку “Відправити”.
Очікуваний результат	Діалогове вікно закривається, користувач інформується про успішне створення нової скарги, на мапі з’являється нова мітка на місці розташування користувача, яка помічає новостворену скаргу.
Фактичний результат	Відповідає очікуваному.

Таблиця Г.7 – Тест 2.4 Створення нової скарги без фото.

Початковий стан системи	Користувач знаходиться на головному екрані з провантаженою мапою із мітками скарг.
Вхідні дані	Категорія скарги, опис скарги.
Опис проведення тесту	Користувач натискає клавішу із символом “+” у правому нижньому куту екрану, після чого відкривається діалогове вікно створення скарги. Користувач обирає відповідну категорію із випадаючого списку та вводить опис проблеми. Після чого натискає кнопку “Відправити”.
Очікуваний результат	Діалогове вікно закривається, користувач інформується про успішне створення нової скарги, на мапі з’являється нова мітка на місці розташування користувача, яка помічає новостворену скаргу.
Фактичний результат	Відповідає очікуваному.

Таблиця Г.8 – Тест 2.5 Перегляд власного розташування на мапі

Початковий стан системи	Користувач знаходиться на головному екрані з мапою.
Вхідні дані	Відсутні.
Опис проведення тесту	Користувач натискає клавішу із символом геолокації у правому верхньому куту екрану.
Очікуваний результат	Мапа переміщається на розташування користувача, яке помічене синім кружечком.
Фактичний результат	Відповідає очікуваному.

Таблиця Г.9 – Тест 3.1 Перегляд власного облікового запису

Початковий стан системи	Користувач знаходиться на головному екрані з мапою.
Вхідні дані	Відсутні.
Опис проведення тесту	Користувач натискає клавішу із символом особи у лівому верхньому куту екрану.
Очікуваний результат	Користувач потрапляє на екран профілю, де він може ознайомитись із інформацією про свій профіль.
Фактичний результат	Відповідає очікуваному.

Таблиця Г.10 – Тест 3.2 Вихід із облікового запису

Початковий стан системи	Користувач знаходиться на екрані профілю.
Вхідні дані	Відсутні.
Опис проведення тесту	Користувач натискає клавішу “Вийти”.
Очікуваний результат	Користувач потрапляє на екран входу в обліковий запис.
Фактичний результат	Відповідає очікуваному.

Додаток Д

Посилання на репозиторії із програмним кодом

Серверна частина: https://github.com/serediukit/civix-backend

Клієнтська частина: https://github.com/serediukit/civix-android-app