

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”**

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:

В.о завідувача кафедри

Михайло НОВОТАРСЬКИЙ

(підпис)

“__” _____ 2025 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою “Інженерія програмного

забезпечення комп’ютерних систем”

спеціальності 121 “Інженерія програмного забезпечення”

на тему: Довідкова система для моніторингу наявності вільних місць у
найближчих сховищах

Виконав : студент 4 курсу, групи ІМ-13
(шифр групи)

Куш Артем Віталійович

(прізвище, ім’я, по батькові)

(підпис)

Керівник доцент, к.т.н. Павлов В.Г.

(посада, науковий ступінь, вчене звання, прізвище та ініціали)

(підпис)

Консультант нормоконтроль асистент, Нікольський С. С.

(назва розділу)

(посада, вчене звання, науковий ступінь, прізвище та ініціали)

(підпис)

Рецензент професор, Корнієнко Б. Я.

(посада, науковий ступінь, вчене звання, прізвище та ініціали)

(підпис)

Засвідчую, що у цьому дипломному
проєкті немає запозичень з праць інших
авторів без відповідних посилань.

Студент _____

(підпис)

Київ – 2025 р

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалавр)

Освітньо-професійна програма

“Інженерія програмного забезпечення комп’ютерних систем”

спеціальності 121 “Інженерія програмного забезпечення”

ЗАТВЕРДЖУЮ

В.о завідувача кафедри

_____ Михайло НОВОТАРСЬКИЙ

(підпис)

“ _ ” _____ 2025 р.

ЗАВДАННЯ

на бакалаврський дипломний проєкт студента

Куща Артем Віталійовича

1. Тема проєкту Довідкова система для моніторингу наявності вільних місць у найближчих сховищах

керівник проєкту Павлов Валерій Георгійович, доцент, к.т.н.,

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від 23 травня 2025 р. №1705-с

2. Термін здачі студентом закінченого проєкту 2 червня 2025 р.

3. Вихідні дані до проєкту технічна документація, теоретичні дані.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які розробляються)

Розділ 1. Аналіз предметної області та постановка задачі

Розділ 2. Проєктування системи та вибір технологій

Розділ 3. Опис розробки застосунку.

Розділ 4. Методика роботи розробленого застосунку

5. Перелік графічного матеріалу (з точним позначенням обов'язкових креслень) діаграма класів (функціональна схема), архітектурна схема (структурна схема), алгоритм взаємодії користувача в системі (принципова схема).

6. Консультанта проєкту, з вказівкою розділів проєкту, які до них вносяться

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормоконтроль	Нікольський С. С.		

7. Дата видачі завдання: «28» січня 2025 р.

Календарний план

№ п/п	Найменування етапів дипломного проєкту	Терміни виконання етапів проєкту	Примітки
1.	<i>Затвердження теми проєкту</i>	<i>01.01.2025-28.01.2025</i>	
2.	<i>Вивчення та аналіз завдання</i>	<i>29.01.2025-01.04.2025</i>	
3.	<i>Розробка архітектури та загальної структури системи</i>	<i>02.04.2025-24.04.2025</i>	
4.	<i>Розробка структур окремих підсистем</i>	<i>25.04.2025-30.04.2025</i>	
5.	<i>Програмна реалізація системи</i>	<i>01.05.2025-19.05.2025</i>	
6.	<i>Оформлення пояснювальної записки</i>	<i>20.05.2025-02.06.2025</i>	
7.	<i>Захист програмного продукту</i>	<i>05.06.2025</i>	
8.	<i>Передзахист</i>	<i>06.06.2025</i>	
9.	<i>Захист</i>	<i>19.06.2025</i>	

Студент-дипломник _____ Артем КУЩ
(підпис)

Керівник проєкту _____ Валерій ПАВЛОВ
(підпис)

АНОТАЦІЯ

У даній дипломній роботі розглянуто проблему оперативного інформування населення про наявність вільних місць у найближчих сховищах під час надзвичайних ситуацій.

Проведено аналіз існуючих підходів та обґрунтовано вибір технологій для розробки мобільного застосунку. За результатами дослідження розроблено програмний застосунок, що забезпечує відображення укриттів на інтерактивній карті, моніторинг їхньої приблизної заповненості за допомогою механізму користувацьких чек-інів/чек-аутів.

Для розробки програмного забезпечення було використано мову програмування Kotlin, платформу Android, UI-фреймворк Jetpack Compose, базу даних Firebase Firestore та картографічний сервіс OpenStreetMap.

Ключові слова: мобільний застосунок, сховища, цивільний захист, моніторинг, Android, Kotlin, Jetpack Compose, Firebase, OpenStreetMap, краудсорсинг.

ANNOTATION

This thesis deals with the problem of promptly informing the population about the availability of free places in the nearest shelters during emergencies.

The existing approaches are analyzed and the choice of technologies for the development of a mobile application is substantiated. Based on the results of the study, a software application has been developed that provides a display of shelters on an interactive map, monitoring their approximate occupancy using the mechanism of user check-ins/checks-outs.

The software was developed using the Kotlin programming language, the Android platform, the Jetpack Compose UI framework, the Firebase Firestore database, and the OpenStreetMap mapping service.

Keywords: mobile application, storage facilities, civil protection, monitoring, Android, Kotlin, Jetpack Compose, Firebase, OpenStreetMap, crowdsourcing.

справки	Формат	Значення			Найменування	Кіл. листів	№ екземпляра	Додаток
					Документація загальна			
					Знову розроблена			
	A4	ІАЛЦ.467200.002 ТЗ			Довідкова система для моніторингу наявності вільних місць у найближчих сховищ	4		
					Технічне завдання			
	A4	ІАЛЦ.467200.003 ПЗ			Довідкова система для моніторингу наявності вільних місць у найближчих сховищ	65		
					Пояснювальна записка			
	A4	ІАЛЦ.467200.004 Д1			Довідкова система для моніторингу наявності вільних місць у найближчих сховищ	1		
					Алгоритм взаємодії користувача в системі (принципова схема)			
	A4	ІАЛЦ.467200.005 Д2			Довідкова система для моніторингу наявності вільних місць у найближчих сховищ	1		
					Діаграма класів (функціональна схема)			
	A4	ІАЛЦ.467200.006 Д3			Довідкова система для моніторингу наявності вільних місць у найближчих сховищ	1		
					Архітектурна схема мобільного застосунку (структурна схема)			
	A4	ІАЛЦ.467200.007 Д4			Довідкова система для моніторингу наявності вільних місць у найближчих сховищ	44		
					Текст програмного коду			
					ІАЛЦ.467200.001 ОА			
Зм	Лист	№ докум.	Під п.	Дата				
Розробив	Куш А. В.				Довідкова система для моніторингу наявності вільних місць у найближчих сховищах Опис альбому			
Перевірів	Павлов В. Г.							
Реценз.	Корнієнко Б. Я.							
Н. Контр.	Нікольський С. С.					КПІ ім. Ігоря Сікорського ФІОТ, ІМ-13		
Затв.	Новотарський М. А.							

**ТЕХНІЧНЕ ЗАВДАННЯ
ДО ДИПЛОМНОГО ПРОЄКТУ**

на тему: «Довідкова система для моніторингу наявності
вільних місць у найближчих сховищах»

Київ – 2025 р.

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ	2
2 ПРИЧИНИ ДЛЯ РОЗРОБКИ	2
3 МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ.....	2
4 ДЖЕРЕЛА РОЗРОБКИ.....	3
5 ТЕХНІЧНІ ВИМОГИ.....	3
5.1. Вимоги до розробленого продукту	3
5.2. Вимоги до програмного забезпечення	3
5.3. Вимоги до апаратної частини	4
6 ЕТАПИ РОЗРОБКИ	4

					ІАЛЦ.467200.002 ТЗ			
Зм.	Арк.	№ докум.	Підпис	Дата				
Розробив	Куц А. В.				Довідкова система для моніторингу наявності вільних місць у найближчих сховищах Технічне завдання	Літ.	Аркуш	Аркушів
Перевірив	Павлов В. Г..						1	4
Реценз.	Корнієнко Б. Я.					КПП ім. Ігоря Сікорського ФІОТ, ІМ-13		
Н. Контр.	Нікольський С. С.							
Затв.	Новотарський М. А.							

1 НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

Дане технічне завдання поширюється на розробку мобільного застосунку довідкової системи для моніторингу наявності вільних місць у найближчих сховищах. Область застосування: надання громадянам України оперативного доступу до інформації про розташування, характеристики та орієнтовну заповненість найближчих укриттів та сховищ з метою підвищення особистої безпеки під час повітряних тривог та інших надзвичайних ситуацій.

2 ПРИЧИНИ ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на виконання бакалаврського проєкту за освітньо-професійною програмою «Інженерія програмного забезпечення» спеціальності 121 «Інженерія програмного забезпечення», затверджене кафедрою інженерії програмного забезпечення Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

3 МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою дипломної роботи є розробка мобільного додатку для платформи Android, який би надавав користувачам інформацію щодо наявності вільних місць у найближчих сховищах. Призначення розробки – підвищити рівень обізнаності та сприяти безпеці громадян шляхом надання зручного доступу до критично важливої інформації під час надзвичайних ситуацій

					ІАЛЦ.467200.002 ТЗ	Арк.
						2
Зм.	Арк.	№ докум.				

4 ДЖЕРЕЛА РОЗРОБКИ

Джерелами розробки є науково-технічна література з питань розробки мобільних застосунків, офіційна документація, а також аналітичні статті, публікації в мережі Інтернет та відкриті дані державних служб.

5 ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до розробленого продукту

- Інтерфейс мобільного застосунку повинен бути простим, інтуїтивно зрозумілим та адаптованим для швидкого використання в стресових ситуаціях.
- Забезпечення можливості перегляду сховищ на інтерактивній карті з відображенням їх точного розташування.
- Надання детальної інформації про кожне сховище: адреса, тип, загальна місткість, орієнтовна поточна заповненість, відповідальна особа (за наявності даних).
- Реалізація механізму анонімного користувацького чек-іну/чек-ауту для збору даних про поточну заповненість сховищ (краудсорсинг).
- Забезпечення швидкого завантаження даних та відгуку інтерфейсу.

5.2. Вимоги до програмного забезпечення

- Операційна система Android 8.0 (Oreo, API рівень 26) та вище.

					ІАЛЦ.467200.002 ТЗ	Арк.
						3
Зм.	Арк.	№ докум.				

5.3. Вимоги до апаратної частини

- Смартфон або планшетний комп'ютер під управлінням операційної системи Android
- Стабільне підключення до мережі Інтернет (Wi-Fi або мобільні дані)
- Вільний простір на внутрішньому накопичувачі пристрою не менше 50 Мбайт

6 ЕТАПИ РОЗРОБКИ

Назва етапів виконання	Термін виконання
Затвердження теми проєкту	01.01.2025-28.01.2025
Вивчення та аналіз завдання	29.01.2025-01.04.2025
Розробка архітектури та загальної структури системи	02.04.2025-24.04.2025
Розробка структур окремих підсистем	25.04.2025-30.04.2025
Програмна реалізація системи	01.05.2025-19.05.2025
Оформлення пояснювальної записки	20.05.2025-02.06.2025

**ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО ДИПЛОМНОГО ПРОЄКТУ**

на тему: «Довідкова система для моніторингу наявності
вільних місць у найближчих сховищах»

Київ – 2025 р.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ.....	3
ВСТУП.....	4
РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ ...	7
1.1 Актуальність теми та необхідність систем інформування про сховище	7
1.2 Аналіз існуючих рішень та їх обмежень	8
1.2.1 Застосунок "Дія"	8
1.2.2 Застосунок "Київ Цифровий"	9
1.2.3 Застосунок "DozoR Укриття"	10
1.2.4 Офіційні ресурси ДСНС та місцевих органів влади	11
ВИСНОВОК ДО РОЗДІЛУ 1	13
РОЗДІЛ 2. ПРОЄКТУВАННЯ СИСТЕМИ ТА ВИБІР ТЕХНОЛОГІЙ.....	14
2.1. Огляд та вибір технологій для розробки клієнтської частини (мобільний додаток Android)	14
2.1.1 Критерії вибору технологій клієнтської частини	14
2.1.2. Мова програмування: Kotlin	16
2.1.3. UI-фреймворк: Jetpack Compose.....	18
2.1.4. Архітектурний патерн: MVVM (Model-View-ViewModel) ..	20
2.1.5. Картографічний сервіс	22
2.1.6. Локальне сховище даних.....	24
2.2. Огляд та вибір технологій для розробки серверної частини	26
2.2.1. Хмарна база даних реального часу: Firebase Firestore	27
2.3. Аналіз та вибір методу визначення заповненості укриттів	29
ВИСНОВОК ДО РОЗДІЛУ 2	34

					ІАЛЦ.467200.002 ПЗ								
Зм.	Арк.	№ докум.	Підпис	Дата									
Розробив		Куш А. В.			Довідкова система для моніторингу наявності вільних місць у найближчих сховищах Пояснювальна записка			Літ.		Аркуш		Аркушів	
Перевірів		Павлов В. Г..								1		65	
Реценз.		Корнієнко Б. Я.						КПІ ім. Ігоря Сікорського ФІОТ, ІМ-13					
Н. Контр.		Нікольський С. С.											
Затв.		Новотарський М. А.											

РОЗДІЛ 3. ОПИС РОЗРОБКИ ЗАСТОСУНКУ	35
3.1 Опис функціонала системи	35
3.2 Інтерфейс та екрани мобільного застосунку	38
3.3 Архітектура застосунку	40
3.3.1 Модель(Model):	41
3.3.2 Вигляд(View):	42
3.3.3 МодельПодання(ViewModel):	43
3.4 Процес автентифікації користувача	45
3.5 Розгортання системи	47
ВИСНОВОК ДО РОЗДІЛУ 3	49
РОЗДІЛ 4. МЕТОДИКА РОБОТИ РОЗРОБЛЕНОГО ЗАСТОСУНКУ	51
4.1 Головний екран та навігація	51
4.2 Загальний вигляд карти з усіма сховищами	52
4.3 Зовнішній вигляд маркера (значка) сховища на карті	53
4.4 Картка з інформацією про сховище (InfoWindow)	54
4.5 Інформація після входу (чек-іну) в сховище	56
4.6 Інформація після виходу (чек-ауту) зі сховища	58
ВИСНОВОК ДО РОЗДІЛУ 4	60
ВИСНОВКИ	61
СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ	63

ПЕРЕЛІК СКОРОЧЕНЬ

API (Application Programming Interface) – Інтерфейс прикладного програмування

UI (User Interface) – Інтерфейс користувача

SDK (Software Development Kit) – Комплект засобів для розробки програмного забезпечення

IDE (Integrated Development Environment) – Інтегроване середовище розробки

JSON (JavaScript Object Notation) – Нотація об'єктів JavaScript

GPS (Global Positioning System) – Глобальна система позиціонування

OS (Operating System) – Операційна система (в контексті Android OS)

HTTP (HyperText Transfer Protocol) – Протокол передачі гіпертексту

MVVM (Model-View-ViewModel) – Модель-Вигляд-МодельПодання (архітектурний патерн)

OSM (OpenStreetMap) – Проект зі створення вільної географічної бази даних, що редагується спільнотою

osmdroid – Бібліотека для інтеграції OpenStreetMap в Android-застосунки

ДСНС – Державна служба України з надзвичайних ситуацій

МГН – Маломобільні групи населення

Wi-Fi (Wireless Fidelity) – Технологія бездротової локальної мережі

FAB(Floating Action Button) - Плаваюча кнопка дії

					ІАЛЦ.467200.002 ТЗ	Арк.
						3
Зм.	Арк.	№ докум.				

ВСТУП

Актуальність теми. В умовах повномасштабної війни, що триває в Україні, забезпечення безпеки цивільного населення під час повітряних тривог набуває першочергового значення. Часті та тривалі повітряні атаки в багатьох регіонах країни створюють постійну загрозу життю та здоров'ю громадян. Практика свідчить, що перебування у власних помешканнях під час ракетних обстрілів чи атак дронів є вкрай ризикованим, і навіть "правило двох стін" не гарантує належного захисту. Єдиним рекомендованим способом убезпечити себе є переміщення до спеціальних захисних споруд цивільного захисту. Важливість наявності достатньої кількості належно облаштованих укриттів визнається на державному рівні. Однак, сама лише наявність сховища поблизу не гарантує безпеки, якщо воно виявиться недоступним або переповненим. Невизначеність щодо наявності вільних місць створює значний психологічний тиск та підвищує ризики, змушуючи людей шукати альтернативу вже під час активної повітряної тривоги.

Аналіз останніх досліджень і публікацій. Існуючі інформаційні системи та мобільні додатки, такі як функція "Незламність" у застосунку "Дія", застосунок "Київ Цифровий", "DozoR Укриття" та офіційні ресурси ДСНС і місцевих органів влади, надають корисну інформацію про розташування та характеристики укриттів. Проте, ключовим недоліком усіх розглянутих систем є відсутність інформації про поточну заповненість укриттів у режимі, близькому до реального часу. Ця інформаційна прогалина є критичною, оскільки знання про наявність вільних місць дозволило б громадянам приймати більш обґрунтовані та безпечні рішення. Існуючі рішення зосереджені на статичних характеристиках, а не на динамічному стані укриттів. Також спостерігається фрагментація даних та відсутність стандартизованого API для отримання оперативної інформації.

					ІАЛЦ.467200.002 ТЗ	Арк.
						4
Зм.	Арк.	№ докум.				

Метою дипломної роботи є розробка мобільного додатку для платформи Android, який би надавав користувачам інформацію щодо наявності вільних місць у найближчих сховищах.

Об'єктом дослідження є процес розробки та функціонування довідкових систем для моніторингу об'єктів інфраструктури в режимі реального часу.

Предметом дослідження є моделі, методи та програмні засоби для створення мобільного додатку, що інформує про наявність вільних місць у найближчих сховищах.

Для досягнення поставленої мети необхідно вирішити такі завдання:

- Проаналізувати предметну область, обґрунтувати актуальність розробки та розглянути існуючі аналогічні рішення.
- Сформулювати вимоги до програмного продукту та обрати необхідні технології для його розробки.
- Розробити архітектуру мобільного додатку та бази даних.
- Реалізувати основний функціонал додатку, включаючи відображення укриттів на карті та механізм інформування про їхню заповненість.
- Провести тестування розробленого прототипу та оцінити його ефективність.

Методи дослідження. У роботі використано методи аналізу науково-технічної літератури з питань цивільного захисту та мобільної розробки, системного аналізу для визначення вимог до системи, об'єктно-орієнтованого проєктування для розробки архітектури додатку, а також методи тестування програмного забезпечення. Використовувалася технічна документація платформи Android, мови Kotlin, UI-фреймворку Jetpack Compose, Google Maps SDK та Firebase Firestore.

Практичне значення одержаних результатів полягає у створенні прототипу мобільного додатку, який потенційно може підвищити рівень поінформованості та безпеки громадян під час повітряних тривог. Надання

оперативної інформації про доступні місця в сховищах дозволить користувачам приймати більш виважені рішення щодо вибору маршруту евакуації.

					ІАЛЦ.467200.002 ТЗ	Арк.
						6
Зм.	Арк.	№ докум.				

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Актуальність теми та необхідність систем інформування про сховище

Потреба в надійних та доступних сховищах під час повітряних тривог є однією з найгостріших проблем безпеки для цивільного населення України в умовах повномасштабної війни [1]. Частота та тривалість повітряних атак у багатьох регіонах країни створюють постійну загрозу життю та здоров'ю громадян. Як свідчить практика, перебування у власних помешканнях під час ракетних обстрілів чи атак дронів є вкрай ризикованим, незалежно від типу будинку. Навіть використання "правила двох стін" не забезпечує належного захисту від потужних вибухів чи прямих влучань і може розглядатися лише як тимчасовий захід при раптовій небезпеці. Тому єдиним рекомендованим способом убезпечити себе є переміщення до спеціальних захисних споруд цивільного захисту. Важливість наявності достатньої кількості укриттів та їх належного стану визнається на державному рівні. Сховище повинно бути не просто підвальним приміщенням, а простором, що забезпечує базові умови для збереження життя: наявність свіжого повітря, зв'язку, місць для сидіння чи лежання, доступу до води. Особлива увага приділяється забезпеченню доступності укриттів для маломобільних груп населення (МГН), що регламентується державними будівельними нормами (ДБН), які передбачають наявність пандусів, ліфтів, відповідної ширини проходів та поручнів [2].

Однак, сама лише наявність сховище поблизу не гарантує безпеки, якщо воно виявиться недоступним або переповненим. Невизначеність щодо наявності вільних місць створює значний психологічний тиск на людей, які

					ІАЛЦ.467200.002 ТЗ	Арк.
						7
Зм.	Арк.	№ докум.				

змушені швидко приймати рішення в стресовій ситуації. Прибуття до переповненого сховища змушує шукати альтернативу під час активної повітряної тривоги, що значно підвищує ризики [3].

Існуючі інформаційні системи та мобільні додатки, хоча й надають корисну інформацію про розташування укриттів, їх тип, а іноді й про наявність певних зручностей чи доступність для МГН, не вирішують проблему інформування про поточну заповненість. Ця інформаційна прогалина є критичною. Знання про те, скільки приблизно вільних місць є в тому чи іншому укритті, дозволило б громадянам приймати більш обґрунтовані рішення щодо вибору найбезпечнішого та найдоступнішого варіанту, потенційно рятуючи життя. Саме тому розробка системи, яка б надавала таку інформацію в режимі, близькому до реального часу, є нагальною потребою.

1.2 Аналіз існуючих рішень та їх обмежень

Для ефективного інформування населення про захисні споруди було створено низку цифрових інструментів. Проте аналіз їх функціональності виявляє певні обмеження, особливо в частині надання динамічної інформації про заповненість.

1.2.1 Застосунок "Дія"

Інтегрована функція "Незламність" дозволяє знаходити найближчі сховища на інтерактивній мапі [4]. Користувачі можуть бачити адресу, тип сховища, його рейтинг, фото (за наявності), інформацію про доступність (наприклад, для людей з інвалідністю) та загальну місткість. Важливою перевагою є можливість завантажити мапу області для офлайн-доступу. Дані

					ІАЛЦ.467200.002 ТЗ	Арк.
						8
Зм.	Арк.	№ докум.				

базуються на інформації від ДСНС та органів місцевого самоврядування, а також оновлюються завдяки відгукам користувачів, які можуть повідомити про стан сховища (наприклад, якщо воно зачинене). Однак, ключовим недоліком є відсутність інформації про поточну кількість вільних місць або фактичну заповненість сховища в реальному часі. Оновлення стану залежить від реакції користувачів та перевірок офіційними органами, що може бути недостатньо оперативно. До того ж, з міркувань безпеки, мапа недоступна в деяких прифронтових областях.

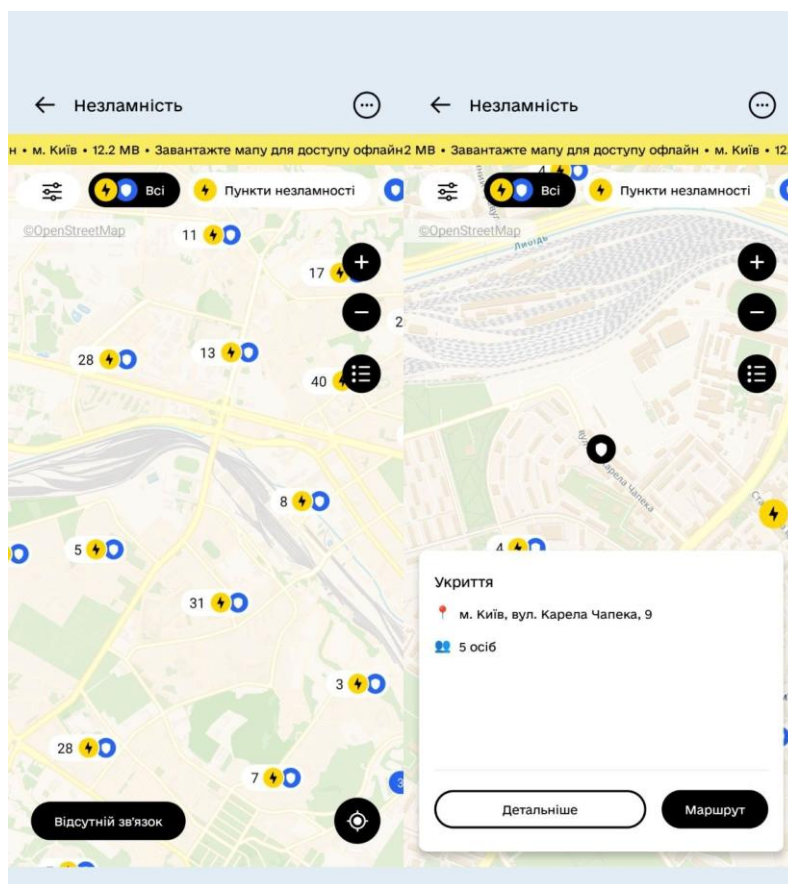


Рисунок 1.1 – Застосунок "Дія"

1.2.2 Застосунок "Київ Цифровий"

Надає аналогічну функціональність, але обмежену територією міста Києва [5]. Мапа укриттів також доступна офлайн після попереднього

					ІАЛЦ.467200.002 ТЗ	Арк.
						9
Зм.	Арк.	№ докум.				

завантаження. Реалізовано механізм зворотного зв'язку, що дозволяє користувачам повідомляти про стан сховища (неналежні умови, зачинено) безпосередньо відповідальному департаменту КМДА. Як і "Дія", застосунок не надає даних про поточну заповненість укриттів.

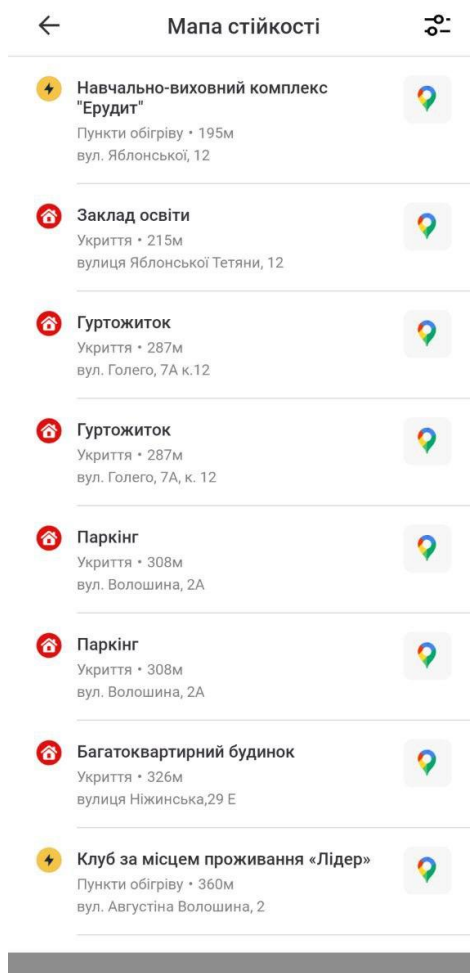


Рисунок 1.2 – Застосунок "Київ Цифровий"

1.2.3 Застосунок "DozoR Укриття"

Цей додаток, згідно з описом в App Store, відображає найпростіші сховища на карті, надсилає push-сповіщення про тривогу та відбій, і, що цікаво, автоматично показує найближче відчинене сховище під час тривоги. Він також показує адресу, тип, балансоутримувача та фото сховища, будує маршрут.

					ІАЛЦ.467200.002 ТЗ	Арк.
						10
Зм.	Арк.	№ докум.				

Проте, залишається незрозумілим, як саме визначається статус "відчинене" – чи це просто означає, що сховище є в офіційному списку і не позначене як закрите, чи є якийсь механізм перевірки фізичної доступності. Найголовніше, додаток не надає інформації про кількість вільних місць чи поточну заповненість [6].

1.2.4 Офіційні ресурси ДСНС та місцевих органів влади

Державна служба України з надзвичайних ситуацій (ДСНС) та місцеві адміністрації підтримують власні реєстри та інтерактивні карти захисних споруд. Наприклад, існує портал shelters.dsns.gov.ua, який містить дані про десятки тисяч укриттів. Також функціонує сервіс "Електронний облік фонду захисних споруд". Місцеві органи влади також можуть публікувати дані на порталах відкритих даних, як, наприклад, у Львові чи Закарпатті. Ці ресурси є важливим джерелом базової інформації, але вони, як правило, надають статичні дані. Інформація про поточну заповненість відсутня [7].

Аналіз показує, що хоча існують інструменти для пошуку укриттів, жоден з них не вирішує ключову проблему – надання інформації про наявність вільних місць у реальному часі. Існуючі рішення зосереджені на *розташуванні та статичних характеристиках* укриттів (адреса, тип, загальна місткість, доступність), але не на їхньому *динамічному стані* (поточна заповненість). Механізми зворотного зв'язку є реактивними і не призначені для моніторингу заповненості.

Незважаючи на те, що вихідна інформація про сховища може надходити з різних платформ (загальнодержавних, регіональних, відомчих), позитивним зрушенням є наявність єдиного стандартизованого, загальнодоступного API, що надає консолідовані дані про всі офіційно зареєстровані сховища на території України. Це значно спрощує завдання розробників зі створення єдиної, повної та актуальної картини розташування та базових характеристик

захисних споруд, усуваючи значну частину проблеми фрагментації даних. Наявність такого API відкриває нові можливості для розробки інноваційних рішень у сфері цивільного захисту. Таким чином, розробка додатку, подібного до запропонованого в цій роботі, може ефективно використовувати цей загальний API як надійне джерело базової інформації про сховища. Основний акцент та інноваційна цінність запропонованого додатку в такому випадку полягатиме у наданні додаткових функцій, які можуть не охоплюватися цим загальним API, зокрема, у реалізації механізму **моніторингу наявності вільних місць у сховищах у режимі, близькому до реального часу, на основі користувацького краудсорсингу**, а також в інтеграції інших корисних для користувача сервісів та унікального інтерфейсу.

Таблиця 1.1 - Порівняння існуючих рішень для інформування про сховища

Характеристика	Київ Цифровий	DozoR Україна	ДСНС Карти/Портали	Пропонований додаток
Платформа	Android, iOS	Android, iOS	Веб	Android
Географія	Київ	Вся Україна	Україна (портали)	Україна
Джерело даних	КМДА	Не визначено	ДСНС, ОМС	API ДСНС
Офлайн карти	Так (авто)	Ні	Ні	Так (кешування)
Реальний час	Ні	Ні	Ні	Так (основна функція)
API для розроб.	Ні	Ні	Немає	Ні

ВИСНОВОК ДО РОЗДІЛУ 1

У першому розділі дипломної роботи обґрунтовано високу актуальність розробки мобільного додатку для інформування про наявність вільних місць в сховищах в Україні. Аналіз поточної ситуації, пов'язаної з регулярними повітряними атаками, та небезпеки перебування поза сховищами під час тривоги підкреслює життєву важливість своєчасного доступу до безпечних місць.

Проведений аналіз існуючих цифрових рішень ("Дія", "Київ Цифровий", "DozoR Укриття", портали ДСНС) виявив їхню значну корисність у наданні інформації про розташування та статичні характеристики укриттів. Однак, ключовим недоліком усіх розглянутих систем є відсутність функції моніторингу та відображення поточної заповненості укриттів у реальному часі. Ця інформаційна прогалина є критичною і створює значні ризики для населення.

Також було виявлено проблеми фрагментації даних про сховища та відсутності єдиного, стандартизованого та загальнодоступного API для їх отримання. Це створює певні виклики для розробки, особливо в частині забезпечення повноти та актуальності базової інформації про захисні споруди.

Розглянуто основні вимоги до захисних споруд, включаючи їх типи, розташування, облаштування та доступність для МГН, що визначає необхідний набір даних для відображення в додатку.

Таким чином, аналіз підтвердив актуальність обраної теми, виявив незадоволену потребу користувачів та обґрунтував необхідність розробки запропонованої системи.

					ІАЛЦ.467200.002 ТЗ	Арк.
						13
Зм.	Арк.	№ докум.				

РОЗДІЛ 2. ПРОЄКТУВАННЯ СИСТЕМИ ТА ВИБІР ТЕХНОЛОГІЙ

Успішна розробка будь-якого програмного продукту значною мірою залежить від правильного вибору технологічного стеку. Для мобільного додатку, який має надавати критично важливу інформацію в режимі, близькому до реального часу, та забезпечувати зручний користувацький досвід, вибір технологій є особливо важливим. У цьому розділі буде проведено огляд потенційних засобів розробки для клієнтської (Android) та серверної частин системи, їх порівняльний аналіз та обґрунтування вибору конкретних технологій.

2.1. Огляд та вибір технологій для розробки клієнтської частини (мобільний додаток Android)

Клієнтська частина є основним інструментом взаємодії користувача з системою. Вона має бути інтуїтивно зрозумілою, швидкою, надійною та ефективно відображати картографічну інформацію та дані про заповненість укриттів.

2.1.1 Критерії вибору технологій клієнтської частини

Загальні критерії вибору технологій клієнтської частини: Вибір технологій для розробки клієнтської частини мобільного додатку ґрунтується на низці ключових загальних критеріїв. До них належать:

					ІАЛЦ.467200.002 ТЗ	Арк.
						14
Зм.	Арк.	№ докум.				

Продуктивність та швидкість роботи додатку: Забезпечення швидкого завантаження, плавної анімації та чутливості інтерфейсу.

Швидкість та ефективність процесу розробки: Можливість швидкого прототипування та ітеративної розробки.

Стабільність та надійність обраних інструментів: Мінімізація помилок та збоїв у роботі додатку.

Доступність якісної документації та активність спільноти розробників: Можливість швидко знаходити рішення проблем та отримувати підтримку.

Відповідність технологій специфічним вимогам проєкту: Ефективна робота з картами, обробка даних в реальному часі, забезпечення інтуїтивно зрозумілого користувацького інтерфейсу, підтримка офлайн-режиму для ключових даних.

Вартість використання (ліцензії, платні сервіси): Перевага надається безкоштовним або доступним рішенням для студентського проєкту.

Клієнтська частина є основним інструментом взаємодії користувача з системою. Вона має бути інтуїтивно зрозумілою, швидкою, надійною та ефективно відображати картографічну інформацію та дані про заповненість сховищ.

Обрані технології наведено на рис. 2.1



Рисунок 2.1 – Обрані технології

2.1.2. Мова програмування: Kotlin



Рисунок 2.2 – Kotlin

Вибір мови програмування є фундаментальним рішенням при розробці мобільного додатку. Для платформи Android основними претендентами є Kotlin та Java.

Java: Традиційна мова для розробки під Android, що існує значно довше. Має величезну спільноту, велику кількість бібліотек та ресурсів. Java є зрілою та стабільною платформою.

Переваги Java: Широка база коду, велика кількість розробників, перевірена часом стабільність.

Недоліки Java для сучасного Android: Багатослівність коду, що призводить до збільшення часу розробки та потенційних помилок; проблема NullPointerException (NPE), яка є однією з найчастіших причин збоїв у Java-додатках; менш ефективна робота з асинхронними операціями порівняно з сучасними підходами.

Kotlin: Сучасна статично типізована мова програмування, розроблена компанією JetBrains. З 2019 року Google офіційно рекомендує Kotlin як пріоритетну мову для розробки Android-додатків [8].

Переваги Kotlin:

Лаконічність та виразність: Kotlin дозволяє писати значно менше коду порівняно з Java для виконання тих самих завдань, що підвищує читабельність та зменшує ймовірність помилок.

Null-безпека (Null Safety): Система типів Kotlin розрізняє типи, що можуть містити null, та типи, що не можуть. Це дозволяє компілятору виявляти потенційні NullPointerException на етапі компіляції, а не під час виконання, що значно підвищує надійність додатку.

Повна інтероперабельність з Java: Kotlin може вільно взаємодіяти з існуючим Java-кодом та бібліотеками. Це дозволяє поступово мігрувати проекти або використовувати перевірені Java-бібліотеки без обмежень.

Корутини (Coroutines): Kotlin надає потужний та елегантний механізм для асинхронного програмування. Корутини спрощують написання неблокуючого коду, роблячи його більш читабельним та легким для розуміння порівняно з традиційними підходами в Java (наприклад, AsyncTask, Callbacks). Це критично важливо для мобільного додатку, який має виконувати мережеві запити та оновлювати UI без "зависань".

Розширені функції (Extension Functions): Дозволяють додавати нову функціональність до існуючих класів без необхідності їх успадкування, що робить код більш гнучким.

Активна підтримка та розвиток Google: Google інвестує значні ресурси в розвиток Kotlin для Android, надаючи нові бібліотеки (Jetpack) та інструменти, оптимізовані саме для цієї мови.

Обґрунтування вибору Kotlin: Для проєкту обрано мову програмування Kotlin. Незважаючи на зрілість Java, переваги Kotlin у лаконічності, null-безпеці та особливо в реалізації асинхронних операцій за допомогою корутин є вирішальними [9]. Можливість писати більш надійний та короткий код безпосередньо впливає на швидкість розробки та якість кінцевого продукту. Оскільки додаток передбачає активну взаємодію з мережею (завантаження даних про сховище, оновлення статусу заповненості), ефективне асинхронне програмування, яке надають корутини Kotlin, є значною перевагою. Офіційна підтримка Kotlin з боку Google також гарантує довгострокову актуальність та доступ до найсучасніших інструментів розробки.

2.1.3. UI-фреймворк: Jetpack Compose

Для побудови користувацького інтерфейсу (UI) в Android-додатках існують два основних підходи: традиційний на основі XML-макетів (Android Views) та сучасний декларативний підхід за допомогою Jetpack Compose.

Android Views (XML-макети): Імперативний підхід, де UI описується в XML-файлах, а логіка управління ним – у Kotlin/Java коді. Це традиційний, добре вивчений спосіб.

Переваги Android Views: Зрілість, велика кількість навчальних матеріалів та готових рішень, візуальний редактор макетів в Android Studio.

Недоліки Android Views: Багатослівність XML та коду для управління станом View; складність управління станом UI, особливо в динамічних інтерфейсах; необхідність часто використовувати findViewById або View

Binding, що може призводити до помилок; менша гнучкість при створенні складних кастомних UI-компонентів.

Jetpack Compose: Сучасний декларативний UI-фреймворк від Google, написаний на Kotlin. Дозволяє описувати UI за допомогою Kotlin-функцій (composable functions).

Переваги Jetpack Compose:

Декларативний підхід: Розробник описує, *яким* має бути UI в певному стані, а Compose сам дбає про його оновлення при зміні стану. Це значно спрощує логіку управління UI.

Менше коду: UI описується безпосередньо в Kotlin, що усуває необхідність в XML-файлах та зменшує загальну кількість коду.

Інтуїтивність та швидкість розробки: Побудова UI за допомогою функцій Kotlin є більш природною та швидкою, особливо для розробників, знайомих з мовою. Функції можна легко комбінувати та перевикористовувати.

Потужне управління станом: Compose має вбудовані механізми для управління станом (наприклад, remember, State), що робить роботу з динамічними даними простішою та надійнішою.

Інтеграція з Kotlin та Jetpack: Compose ідеально інтегрується з Kotlin (використовуючи корутини, лямбди тощо) та іншими бібліотеками Jetpack (Navigation, ViewModel).

Підвищення продуктивності розробника: Швидкі ітерації, живий перегляд змін (Live Preview) та менша кількість шаблонного коду сприяють прискоренню розробки.

Обґрунтування вибору Jetpack Compose: Для розробки UI додатку обрано Jetpack Compose. Декларативний підхід та тісна інтеграція з Kotlin роблять його потужним інструментом для створення сучасних та інтерактивних інтерфейсів. Здатність Compose ефективно оновлювати UI при зміні стану даних є критично важливою для додатку, який має відображати інформацію про заповненість укриттів у реальному часі [10]. Хоча Jetpack Compose є відносно

новішим за Android Views, він активно розвивається, підтримується Google та швидко набирає популярність у спільноті, що забезпечує доступ до необхідних ресурсів та підтримки. Для даного проєкту, де важлива швидкість розробки та можливість створення динамічного UI, переваги Compose є суттєвими.



Рисунок 2.3 – Jetpack Compose

2.1.4. Архітектурний патерн: MVVM (Model-View-ViewModel)

Вибір архітектурного патерну визначає структуру коду, його масштабованість, тестованість та легкість підтримки. Для Android-додатків популярними є декілька патернів: MVC (Model-View-Controller), MVP (Model-View-Presenter) та MVVM (Model-View-ViewModel).

MVC (Model-View-Controller):

Model: Дані та бізнес-логіка.

View: Відображення UI.

Controller: Обробляє введення користувача, взаємодіє з Model та оновлює View. В Android часто Activity/Fragment виконують роль і View, і Controller, що призводить до їх перевантаження.

Недоліки для Android: Тісний зв'язок між View та Controller, складність тестування, перевантажені Activity/Fragment.

MVP (Model-View-Presenter):

Model: Дані та бізнес-логіка.

View: Пасивний інтерфейс, що відображає дані та передає дії користувача Presenter'у. Activity/Fragment є View.

Presenter: Отримує дані від Model, готує їх для відображення та викликає методи View для оновлення UI. View та Presenter взаємодіють через інтерфейс.

Переваги: Кращий розподіл відповідальності порівняно з MVC, вища тестованість Presenter'а.

Недоліки: Може призводити до великої кількості методів в інтерфейсі View, Presenter все ще може мати посилання на Android-специфічні компоненти View.

MVVM (Model-View-ViewModel):

Model: Дані та бізнес-логіка (аналогічно до MVC/MVP).

View: Відображає UI (Activity/Fragment з XML або Composable функції в Jetpack Compose). Підписується на зміни у ViewModel.

ViewModel: Містить логіку UI та керує станом View. Надає дані для View через спостережувані об'єкти (Observable data holders, наприклад, LiveData або StateFlow в Kotlin). ViewModel не має прямого посилання на View, що сприяє кращому відокремленню.

Переваги MVVM:

Розподіл відповідальності: Чітке відокремлення логіки UI (ViewModel) від самого UI (View).

Тестованість: ViewModel не залежить від Android Framework компонентів (якщо правильно реалізовано), що робить її легкою для юніт-тестування.

Стійкість до змін конфігурації: ViewModel може переживати зміни конфігурації Android (наприклад, поворот екрану), зберігаючи свій стан.

Data Binding / Observe: View автоматично оновлюється при зміні даних у ViewModel завдяки механізмам спостереження (наприклад, LiveData або State у Compose).

Рекомендовано Google: MVVM є частиною рекомендованих архітектурних компонентів Android Jetpack.

Обґрунтування вибору MVVM: Для проєкту **обрано архітектурний патерн MVVM**. Цей патерн найкраще відповідає сучасним підходам до розробки Android-додатків, особливо при використанні Jetpack Compose, який природно інтегрується з ViewModel та механізмами спостереження за станом (State) [11]. MVVM забезпечує високий рівень тестованості, стійкість до змін конфігурації та чіткий розподіл відповідальності, що спрощує розробку та підтримку додатку. Використання ViewModel з Jetpack дозволить ефективно керувати станом UI, таким як список укриттів, дані про їх заповненість, та забезпечувати їх автоматичне оновлення на екрані.

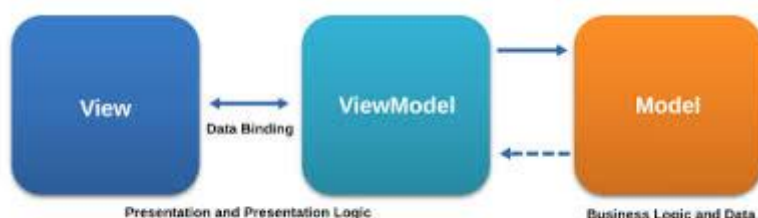


Рисунок 2.4 – Приклад патерну MVVM

2.1.5. Картографічний сервіс

Для відображення укриттів на карті та побудови маршрутів необхідний надійний картографічний сервіс.

Перелік альтернатив:

Google Maps SDK for Android

Mapbox SDK for Android

OpenStreetMap (OSM) з використанням відповідних бібліотек (наприклад, osmdroid, MapLibre GL Native)

Послідовний розгляд альтернатив:

Google Maps SDK for Android: Надає високоякісні карти, зручне API, інтеграцію з екосистемою Google. Має велику спільноту та детальну документацію. Пропонує щедрий безкоштовний рівень використання.

Переваги: Висока якість карт, відмінна продуктивність, розвинене API.

Недоліки: Платний при перевищенні безкоштовних лімітів; певні обмеження щодо повної кастомізації та використання даних поза екосистемою Google; офлайн-можливості можуть бути обмежені для завантаження великих регіонів без спеціальних рішень.

Mapbox SDK for Android: Потужний та гнучкий інструмент з широкими можливостями кастомізації стилів карт та візуалізації даних. Надає власні SDK для мобільних платформ.

Переваги: Високий ступінь кастомізації, якісні інструменти для розробників.

Недоліки: Переважно платне рішення, вартість може бути значною для масштабних проєктів або при великій кількості користувачів.

OpenStreetMap (OSM): Це глобальний проєкт зі створення вільної географічної бази даних, що редагується спільнотою. Дані OSM є відкритими та безкоштовними. Для використання OSM в Android-додатку потрібні сторонні бібліотеки для рендерингу карт [12].

Переваги OSM: Повністю безкоштовне використання даних; відкрита ліцензія даних (ODbL), що дає велику свободу; активна глобальна спільнота, яка постійно оновлює дані; можливість завантажувати дані та використовувати їх повністю офлайн; високий ступінь контролю над даними та їх відображенням.

Недоліки OSM: Якість та повнота даних можуть варіюватися залежно від регіону та активності місцевої спільноти (хоча для України дані OSM є досить детальними); вимагає вибору та інтеграції сторонньої бібліотеки для рендерингу (наприклад, osmdroid, MapLibre GL Native, Mapsforge), що може

додати складності в налаштуванні; стилізація карт може вимагати більше зусиль.

Бібліотеки для роботи з OSM на Android:

osmdroid: Популярна, безкоштовна бібліотека з відкритим кодом, що є прямою альтернативою Google Maps API v2. Підтримує відображення тайлів OSM, кешування, маркери, полілінії тощо.

MapLibre GL Native for Android: Вільний форк Mapbox GL Native, що дозволяє використовувати векторні тайли (включно з OSM data sources, наприклад, через Maptiler або інші сервіси, що надають OSM векторні тайли) та кастомні стилі.

Обґрунтування вибору OpenStreetMap з osmdroid: Для прототипу додатку обрано використання даних OpenStreetMap з інтеграцією через бібліотеку osmdroid. Ключовими перевагами є безкоштовність, відкритість даних, можливість повноцінної роботи в офлайн-режимі (що критично для ситуацій з нестабільним зв'язком) та гнучкість у використанні даних. Незважаючи на потенційні виклики з налаштуванням та стилізацією, ці переваги є вирішальними для проєкту, орієнтованого на доступність та надійність в умовах обмежених ресурсів та можливих проблем зі зв'язком [13].

2.1.6. Локальне сховище даних

Для кешування даних про сховище (адреси, координати, тип, місткість), отриманих з сервера, та забезпечення їх доступності навіть при тимчасових проблемах з інтернет-з'єднанням, необхідне локальне сховище.

Альтернативи:

Room Persistence Library (надбудова над SQLite)

Чистий SQLite з SQLiteOpenHelper

SharedPreferences

Зберігання у файлі (наприклад, JSON, CSV, серіалізовані об'єкти)

Послідовний розгляд альтернатив:

Room Persistence Library / Чистий SQLite:

SQLite – це повноцінна реляційна база даних, вбудована в Android. Room – це бібліотека абстракції над SQLite, яка спрощує роботу з базою даних, зменшує кількість шаблонного коду та забезпечує перевірку SQL-запитів на етапі компіляції.

Переваги SQLite/Room: Потужні можливості для зберігання структурованих даних, підтримка складних запитів, транзакцій, індексів. Room значно спрощує роботу з SQLite [8].

Недоліки SQLite/Room: Може бути надлишковим для дуже простих завдань зберігання даних (наприклад, простого списку об'єктів без складних зв'язків чи запитів). Вимагає визначення схеми даних.

SharedPreferences:

Призначений для зберігання невеликих обсягів даних у форматі ключ-значення (примітивні типи даних або їх рядкові представлення).

Переваги: Дуже простий у використанні для налаштувань додатку або збереження простих прапорців.

Недоліки: Не підходить для зберігання великих обсягів структурованих даних, таких як списки об'єктів сховищ.

Зберігання у файлі (наприклад, JSON, CSV, серіалізовані об'єкти):

Пряме збереження даних у файли у внутрішньому або зовнішньому сховищі пристрою. Дані можуть бути збережені у текстовому форматі (JSON, CSV) або бінарному (серіалізовані об'єкти).

Переваги: Простота реалізації для нескладних структур даних; повний контроль над форматом даних; не потребує додаткових бібліотек (окрім, можливо, парсерів для JSON/CSV); легко реалізувати кешування даних, завантажених з мережі.

Недоліки: Відсутність вбудованих механізмів для складних запитів чи індексації; операції читання/запису та серіалізації/десеріалізації потрібно реалізовувати вручну (або за допомогою бібліотек типу Gson/Moshi для JSON); менш ефективний для частого оновлення окремих частин великого файлу.

Обґрунтування вибору Зберігання у файлі (JSON): Для локального кешування даних про сховища (адреси, координати, тип, місткість тощо), які отримуються з Firebase Firestore, обрано підхід зберігання даних у внутрішньому файлі додатку у форматі JSON. Для прототипу, де основна задача – кешувати список об'єктів для офлайн-доступу та швидкого відображення, цей метод є достатньо простим в реалізації та ефективним. JSON є легкочитним форматом, для роботи з яким існують зручні бібліотеки (наприклад, вбудовані в Kotlinx.serialization або сторонні як Gson/Moshi). Це дозволить уникнути складності, пов'язаної з налаштуванням реляційної бази даних (Room/SQLite), яка може бути надлишковою для поточних потреб проєкту. Оновлення кешу відбуватиметься шляхом перезапису файлу актуальними даними з сервера [14, 15].

2.2. Огляд та вибір технологій для розробки серверної частини

Серверна частина відповідає за зберігання даних про сховище, їх характеристики, а також за обробку та агрегацію інформації про поточну заповненість.

					ІАЛЦ.467200.002 ТЗ	Арк.
						26
Зм.	Арк.	№ докум.				

2.2.1. Хмарна база даних реального часу: Firebase Firestore

Для зберігання даних про сховище та, що найважливіше, для оновлення та отримання інформації про їх заповненість у режимі, близькому до реального часу, потрібне відповідне серверне рішення.

Альтернативи:

Власний Backend + Реляційна БД (наприклад, Node.js/Python + PostgreSQL/MySQL):

Переваги: Повний контроль над логікою та даними, гнучкість у виборі технологій, можливість реалізації складної бізнес-логіки.

Недоліки: Вимагає значних зусиль на розробку, розгортання, масштабування та підтримку серверної інфраструктури; реалізація функціоналу реального часу (наприклад, через WebSockets) є нетривіальною задачею; вищі початкові витрати часу та потенційно коштів.

Власний Backend + NoSQL БД (наприклад, Node.js/Python + MongoDB):

Переваги: Краща масштабованість та гнучкість схеми даних порівняно з реляційними БД для певних типів завдань.

Недоліки: Аналогічні до варіанту з реляційною БД щодо розробки та підтримки бекенду.

Інші BaaS (Backend as a Service) платформи (наприклад, AWS Amplify, Supabase):

Переваги: Спрощують розробку бекенду, надаючи готові компоненти (автентифікація, бази даних, API).

Недоліки: Можуть мати власну криву навчання, модель ціноутворення може стати обтяжливою при зростанні навантаження.

Firebase Firestore (частина платформи Firebase від Google):

Переваги Firebase Firestore:

NoSQL документо-орієнтована база даних: Гнучка схема даних, що добре підходить для зберігання різномірної інформації про сховище.

Оновлення в реальному часі: Ключова перевага для цього проєкту. Firestore дозволяє клієнтам (мобільним додаткам) підписуватися на зміни даних, і ці зміни автоматично надсилаються на клієнт без необхідності постійних запитів (polling). Це ідеально для відображення актуальної заповненості укриттів.

Масштабованість: Автоматично масштабується залежно від навантаження.

Інтеграція з іншими сервісами Firebase: Легко інтегрується з Firebase Authentication (для автентифікації користувачів, якщо це буде потрібно), Firebase Cloud Functions (для серверної логіки), Firebase Hosting тощо.

Потужні можливості запитів та індексації.

Офлайн-підтримка на клієнті: Клієнтські SDK Firestore автоматично кешують дані, що дозволяє додатку працювати з локальними даними при відсутності мережі та синхронізувати зміни при відновленні з'єднання.

Щедрий безкоштовний рівень (Spark Plan): Дозволяє розробляти та тестувати додаток без значних фінансових витрат, що важливо для студентського проєкту.

Простота налаштування та використання: Значно швидше почати роботу порівняно з розробкою власного бекенду.

Обґрунтування вибору Firebase Firestore: Для серверної частини та зберігання даних **обрано Firebase Firestore**. Її здатність до оновлення даних у реальному часі є вирішальною для основного функціоналу додатку – відображення актуальної заповненості укриттів. Гнучкість NoSQL моделі, автоматична масштабованість, тісна інтеграція з іншими сервісами Firebase та наявність щедрого безкоштовного тарифу роблять Firestore оптимальним вибором для швидкої розробки та прототипування мобільного додатку. Це дозволяє зосередитися на розробці клієнтської логіки та користувацького досвіду, мінімізуючи витрати часу на розробку та підтримку традиційного бекенду [16, 17].



Рисунок 2.5 – Firebase

2.3. Аналіз та вибір методу визначення заповненості укриттів

Одним із ключових та найскладніших завдань при розробці системи є вибір та реалізація механізму визначення поточної заповненості укриттів. Розглянемо основні можливі підходи, їхні переваги, недоліки та доцільність застосування в рамках даної дипломної роботи.

Варіант 1: Користувацький краудсорсинг (User Crowdsourcing) – система "Чек-ін/Чек-аут"

Опис: Користувачі додатку самостійно відмічають свій прихід в сховище ("чек-ін") та вихід з нього ("чек-аут"). Система агрегує ці дані для кожного сховища та розраховує приблизну поточну кількість присутніх.

Переваги:

Потенційно висока оперативність оновлення даних (близька до реального часу).

Низькі витрати на інфраструктуру: не потребує встановлення додаткового обладнання в сховищах.

Використовує наявні у користувачів пристрої.

Недоліки:

Низька точність та надійність: Дуже залежить від дисципліни та чесності користувачів. Люди можуть забувати відмічатися при вході/виході, робити це

із запізненням або взагалі ігнорувати цю функцію. Існує високий ризик навмисного спотворення даних (фейкові чекіни/чекауті).

Потреба у валідації: Необхідні складні механізми перевірки достовірності даних: підтвердження геолокації користувача в момент чекіну (з урахуванням обмежень точності GPS та можливості її підробки), можливо, використання геоогорож (geofencing), часових обмежень на перебування, автоматичного "чек-ауту" при виході з зони сховище або після певного часу.

Залежність від мережі: Чек-ін/чек-аут потребує активного інтернет-з'єднання в момент дії.

Конфіденційність: Потребує доступу до геолокації користувача.

Складність реалізації: Потребує розробки механізмів валідації, обробки великої кількості подій, можливо, системи репутації користувачів.

Стратегії валідації/пом'якшення недоліків: Використання геолокації з перевіркою точності та захистом від підробки; встановлення тайм-аутів для автоматичного чек-ауту; використання геоогорож (з урахуванням їх обмежень точності); можливо, перехресна перевірка даних від різних користувачів; застосування алгоритмів виявлення аномалій.

Варіант 2: Ручне оновлення відповідальною особою (Manual Update by Responsible Person)

Опис: У кожному укритті призначається відповідальна особа (комендант, черговий, волонтер), яка періодично (або при суттєвій зміні кількості людей) вручну підраховує присутніх та оновлює дані через спеціальний інтерфейс у додатку або веб-панель.

Переваги:

Потенційно вища точність підрахунку порівняно з неконтрольованим краудсорсингом.

Менша вразливість до випадкових помилок користувачів чи навмисних масових фальсифікацій.

Простіша валідація (основана на довірі до відповідальної особи та можливості верифікації її повноважень).

Недоліки:

Не є реальним часом: Актуальність даних залежить від частоти оновлень відповідальною особою. Можливі значні затримки.

Масштабованість: Потребує наявності та постійної роботи відповідальної особи в *кожному* укритті, що може бути нереалістично, особливо для великої кількості найпростіших укриттів.

Людський фактор: Залежить від сумлінності, уважності та можливостей відповідальної особи (наприклад, під час сильного напливу людей точний підрахунок ускладнюється).

Залежність від мережі: Відповідальній особі потрібне інтернет-з'єднання для надсилання оновлень.

Організаційні складнощі: Потребує створення системи призначення, навчання та контролю відповідальних осіб.

Загальні проблеми ручних систем: Схильність до помилок, затримок, відсутність автоматизації.

Варіант 3: Сенсорні системи (Sensor-Based Systems)

Опис: Встановлення в сховищах автоматизованих систем підрахунку людей на основі різних технологій: інфрачервоні датчики , відеоаналітика , Wi-Fi трекінг , Bluetooth-маячки (beacons).

Переваги:

Найвищий потенціал точності та роботи в реальному часі.

Пасивний збір даних (не потребує дій від користувачів чи відповідальних осіб).

Недоліки:

Висока вартість: Значні витрати на закупівлю, монтаж та обслуговування обладнання.

Інфраструктурні вимоги: Потребує стабільного електроживлення та, як правило, мережевого підключення в кожному укритті.

Технічна складність: Розробка та інтеграція таких систем є складним інженерним завданням.

Конфіденційність: Технології на основі відео чи Wi-Fi трекінгу викликають серйозні питання щодо приватності.

Недоцільність для дипломної роботи: Реалізація такого підходу виходить далеко за межі типового обсягу та ресурсів бакалаврського проєкту.

Вибір методу та обґрунтування:

Враховуючи обмеження бакалаврської дипломної роботи (час, ресурси, відсутність можливості встановлення обладнання), сенсорні системи є недоцільними для реалізації. Вибір стоїть між користувацьким краудсорсингом та ручним оновленням відповідальною особою.

Ручне оновлення видається більш надійним з точки зору точності підрахунку в конкретний момент часу, коли оновлення зроблене. Однак, його головний недолік – відсутність гарантії актуальності даних у проміжках між оновленнями та значні організаційні труднощі масштабування на велику кількість укриттів.

Користувацький краудсорсинг є більш привабливим з точки зору потенційної роботи в реальному часі та масштабованості без потреби в спеціально призначених людях у кожному укритті. Його основний виклик – низька достовірність даних та потреба в механізмах валідації.

Для цілей даної дипломної роботи, яка передбачає розробку *прототипу*, обирається метод користувацького краудсорсингу (чек-ін/чек-аут). Цей вибір дозволяє продемонструвати основну ідею системи – відображення динамічної інформації про заповненість – використовуючи доступні технології (мобільний додаток, геолокація, хмарна БД). При цьому важливо чітко усвідомлювати та задокументувати обмеження цього методу щодо точності та надійності. У рамках прототипу будуть реалізовані базові механізми валідації (наприклад,

перевірка приблизної геолокації під час чекіну та захист від очевидних спроб підробки координат). Питання підвищення надійності даних залишається напрямком для подальших досліджень та вдосконалень.

Вибір методу визначення заповненості має фундаментальний вплив на архітектуру та функціональність системи. Краудсорсинговий підхід вимагає реалізації функцій реєстрації/автентифікації користувачів (хоча для прототипу це може бути спрощено), інтерфейсів для чекіну/чекауту, логіки валідації на клієнті та/або сервері, а також ретельного проєктування правил безпеки Firestore для запобігання несанкціонованому доступу та маніпуляціям. Надійність даних стає ключовим викликом, оскільки користувачі покладатимуться на цю інформацію в критичних ситуаціях, і будь-які суттєві розбіжності між відображуваною та реальною заповненістю можуть підірвати довіру до додатку.

					ІАЛЦ.467200.002 ТЗ	Арк.
						33
Зм.	Арк.	№ докум.				

ВИСНОВОК ДО РОЗДІЛУ 2

У даному розділі було проведено детальний аналіз та обґрунтування вибору технологій для розробки мобільного додатку . Для клієнтської частини (Android) обрано:

Мова програмування: **Kotlin**.

UI-фреймворк: **Jetpack Compose**.

Архітектурний патерн: **MVVM**.

Картографічний сервіс: **OpenStreetMap** (з використанням osmdroid або аналогічної бібліотеки).

Локальне сховище даних: **Зберігання у файлі (формат JSON)**.

Для серверної частини та бази даних обрано **Firebase Firestore** для зберігання даних та забезпечення оновлень у реальному часі. Прийнято рішення **відмовитися від використання спеціалізованої серверної логіки (наприклад, Firebase Cloud Functions)** на користь спрощення архітектури та використання можливостей Firestore Security Rules та клієнтської логіки.

Також було детально проаналізовано та обґрунтовано вибір методу визначення заповненості сховищ на основі **користувацького краудсорсингу (чек-ін/чек-аут)**, з визначенням його переваг, недоліків, базових механізмів валідації для прототипу та окреслено напрямки для подальшого вдосконалення надійності даних.

Обраний технологічний стек є сучасним, переважно безкоштовним для розробки, добре підтримуваним та відповідає вимогам проєкту щодо функціональності, надійності та можливості реалізації в рамках кваліфікаційної роботи бакалавра.

					ІАЛЦ.467200.002 ТЗ	Арк.
						34
Зм.	Арк.	№ докум.				

РОЗДІЛ 3. ОПИС РОЗРОБКИ ЗАСТОСУНКУ

У цьому розділі детально розглянуто процес проектування та розробки мобільного застосунку, призначеного для моніторингу наявності вільних місць у найближчих сховищах. Описано основний функціонал системи, ключові компоненти користувацького інтерфейсу та екрани, обґрунтовано вибір архітектурних рішень, деталізовано процес автентифікації користувачів та розглянуто можливі шляхи розгортання та розповсюдження готового програмного продукту.

3.1 Опис функціонала системи

Розроблений мобільний застосунок є довідково-інформаційною системою, що має на меті надання громадянам України актуальної та легкодоступної інформації про укриття та сховища, а також про їхню орієнтовну поточну заповненість. Основний функціонал системи включає наступні ключові можливості:

Відображення сховищ на інтерактивній карті: Застосунок інтегрує картографічний сервіс OpenStreetMap, використовуючи бібліотеку osmdroid для платформи Android. Це дозволяє візуалізувати геолокацію зареєстрованих укриттів у вигляді маркерів на карті. Користувачі можуть взаємодіяти з картою шляхом масштабування (збільшення/зменшення), панорамування (переміщення видимого фрагменту карти) та центрування на своєму поточному місцезнаходженні (за наявності відповідних дозволів).

Надання детальної інформації про сховища: При виборі конкретного укриття (натисканням на маркер на карті) користувач отримує доступ до розширеної інформації про нього. Це може включати точну адресу, тип споруди (наприклад, протирадіаційне укриття, найпростіше укриття, підвальне

					ІАЛЦ.467200.002 ТЗ	Арк.
						35
Зм.	Арк.	№ докум.				

приміщення), загальну розрахункову місткість, наявність специфічних умов (наприклад, доступність для маломобільних груп населення, наявність питної води, санвузла), а також, за наявності в базі даних, контактні дані відповідальної особи чи балансоутримувача.

Моніторинг орієнтовної заповненості сховищ через краудсорсинг: Центральною функцією є механізм збору даних про поточну заповненість укриттів, що базується на активності самих користувачів. Користувачі мають змогу анонімно "zareєstrуватися" (зробити чек-ін) при вході до обраного сховища та, відповідно, "vipисатися" (зробити чек-аут) при виході. Система агрегує ці дані в реальному часі, розраховуючи та відображаючи приблизну кількість зайнятих та вільних місць. Точність цих даних залежить від активності та сумлінності користувачів.

Синхронізація даних у реальному часі: Інформація про характеристики сховищ та, що особливо важливо, їх поточна заповненість зберігається та синхронізується за допомогою хмарної NoSQL бази даних Firebase Firestore. Це забезпечує миттєве оновлення даних на пристроях усіх користувачів застосунку, щойно відбуваються зміни (наприклад, новий чек-ін).

Початкове завантаження та оновлення даних про укриття: Передбачено механізм отримання початкового переліку укриттів та їх базових характеристик з офіційного API Державної служби України з надзвичайних ситуацій (ДСНС) (за умови його стабільної роботи, доступності та відповідності необхідному формату даних). Це дозволяє наповнити базу даних початковою інформацією. У подальшому можливе періодичне оновлення цих базових даних.

Анонімна автентифікація користувачів: Для участі у системі чек-інів/чек-аутів та забезпечення унікальності дій, користувачі проходять швидку та просту процедуру анонімної автентифікації через сервіс Firebase Authentication. Це дозволяє системі розрізняти користувачів для коректного підрахунку заповненості, не вимагаючи при цьому створення повноцінного облікового запису з введенням персональних даних.

					ІАЛЦ.467200.002 ТЗ	Арк.
						36
Зм.	Арк.	№ докум.				

Загальну логіку взаємодії користувача з основними функціями системи, послідовність дій та переходи між станами ілюструє діаграма, наведена нижче на рис 3.1.

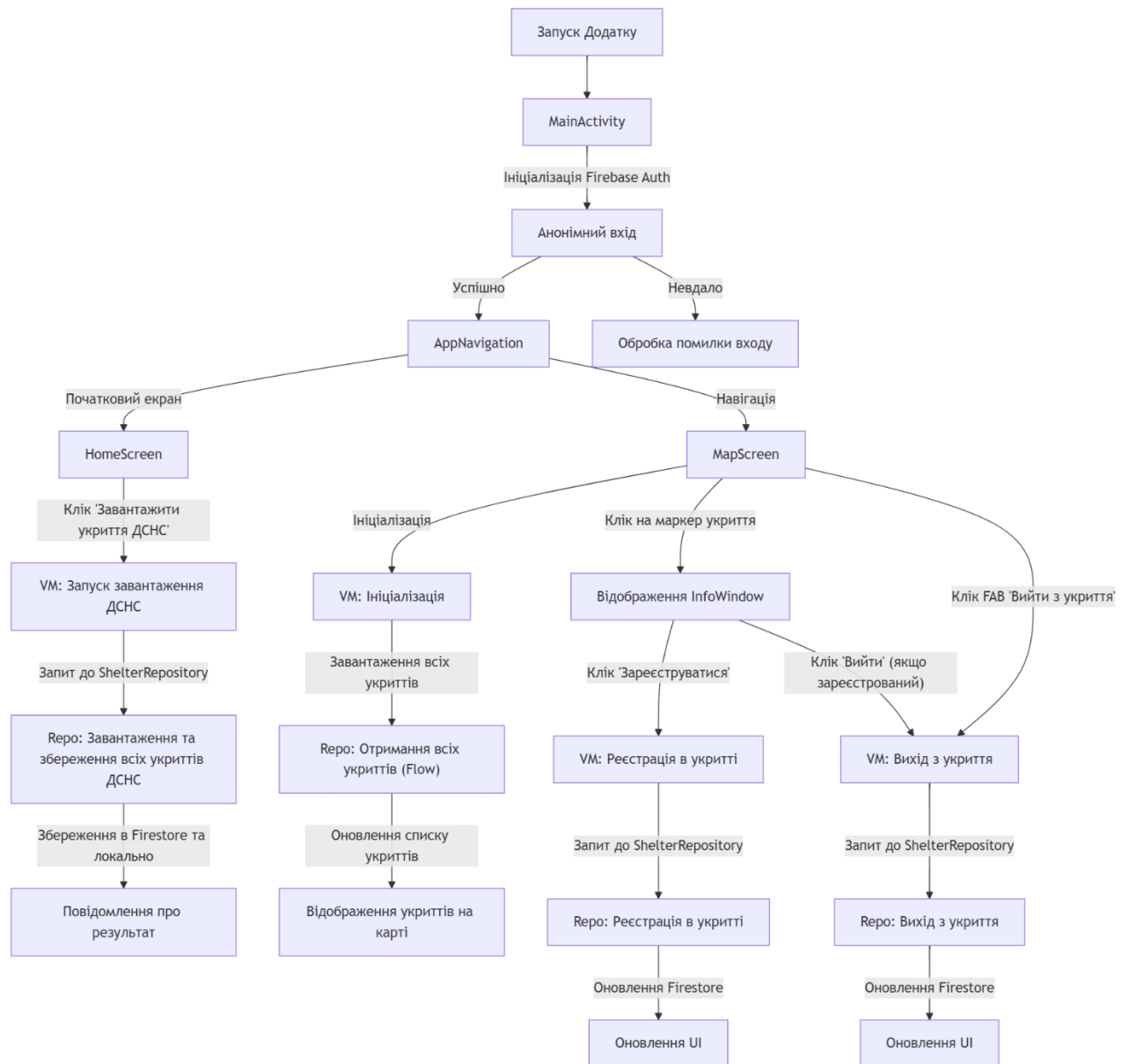


Рисунок 3.1 – Діаграма алгоритму взаємодії користувача в системі

3.2 Інтерфейс та екрани мобільного застосунку

Інтерфейс користувача розробленого мобільного застосунку створено з використанням сучасного декларативного UI-фреймворку Jetpack Compose для платформи Android. При розробці було дотримано принципів Material Design, що забезпечує інтуїтивно зрозумілий, візуально привабливий та узгоджений користувацький досвід, адаптований для швидкого використання, особливо в умовах обмеженого часу та потенційно стресових ситуацій.

Основні екрани застосунку та їх компоненти:

Головний екран (HomeScreen): Цей екран слугує вхідною точкою для користувача після запуску застосунку. Він може містити коротку вітальну інформацію, основні інструкції щодо користування системою, а також кнопку для переходу до основного функціоналу – інтерактивної карти сховищ. Важливою функцією, доступною з цього екрану, є ініціювання процесу завантаження або оновлення даних про укриття з API ДСНС, що дозволяє користувачеві отримати найсвіжішу базову інформацію про доступні сховища.

Екран карти (MapScreen): Це центральний та найбільш функціонально насичений екран застосунку. Основну його частину займає інтерактивна карта, реалізована за допомогою бібліотеки osmdroid та компонента AndroidView у Jetpack Compose. На карті відображаються маркери, що позначають геолокацію сховищ. Користувач може вільно масштабувати карту, переміщуватися по ній для огляду різних районів. При натисканні на маркер сховища на карті з'являється інформаційне вікно (InfoWindow), яке містить ключову інформацію: адресу, тип, місткість та поточну орієнтовну заповненість. В цьому ж вікні розташовані кнопки "Чек-ін" та "Чек-аут", які дозволяють користувачеві повідомити про свій статус відносно даного сховища.

Навігаційна панель (BottomNavBar): Для зручного перемикання між основними розділами застосунку (наприклад, між Головним екраном та

Картою) використовується стандартна нижня навігаційна панель. Її реалізація та логіка переходів визначені у файлі `AppNavigation.kt`.

Структура файлів та класів, що відповідають за реалізацію користувацького інтерфейсу та навігації:

``com.akushch.shelterapp.screens.HomeScreen.kt``: Містить `Composable`-функції, що описують вигляд та поведінку головного екрану.

``com.akushch.shelterapp.screens.MapScreen.kt``: Реалізує екран з картою, логіку відображення маркерів сховищ, обробку взаємодії з картою та інформаційними вікнами.

``com.akushch.shelterapp.navigation.AppNavigation.kt``: Визначає навігаційний граф застосунку, включаючи всі доступні маршрути (екрани) та логіку переходів між ними. Також тут реалізовано нижню навігаційну панель.

``com.akushch.shelterapp.MainActivity.kt``: Головна активність (`Activity`) застосунку, яка є точкою входу. Вона відповідає за початкове налаштування, ініціалізацію `Firebase`, конфігурацію бібліотеки `osmdroid` та запуск кореневої `Composable`-функції, що містить навігаційну систему.

``com.akushch.shelterapp.ui.theme``: Папка містить файли (``Color.kt``, ``Theme.kt``, ``Type.kt``), що визначають візуальну тему застосунку: кольорову палітру, типографіку та загальні стилі відповідно до `Material Design`.

Дизайн застосунку свідомо спрямований на мінімалізм та функціональність. Пріоритетом є швидкий доступ до критично важливої інформації без перевантаження користувача зайвими візуальними елементами чи складною навігацією. Використання `Jetpack Compose` дозволило створити гнучкий та реактивний інтерфейс, що ефективно оновлюється при зміні даних.

3.3 Архітектура застосунку

Для забезпечення належної структури, масштабованості, тестованості та легкості підтримки коду мобільного застосунку було обрано архітектурний патерн MVVM (Model-View-ViewModel). Цей патерн рекомендований Google для розробки Android-застосунків і чудово інтегрується з декларативним підходом Jetpack Compose.

Загальний опис архітектурного патерну MVVM (Model-View-ViewModel):

MVVM – це архітектурний шаблон проєктування програмного забезпечення, який сприяє відокремленню розробки графічного інтерфейсу користувача (View) від розробки бізнес-логіки або логіки представлення даних (Model). ViewModel у цьому патерні виступає як посередник, що готує дані з Моделі для відображення у Вигляді та обробляє команди від Вигляду [18].

Основні компоненти патерну MVVM:

1) Model (Модель): Представляє дані та бізнес-логіку застосунку. Модель не має жодного уявлення про View або ViewModel. Вона відповідає за отримання, зберігання, обробку даних та реалізацію основної функціональності системи. Це можуть бути локальні бази даних, мережеві сервіси, класи, що інкапсулюють правила бізнес-логіки тощо.

2) View (Вигляд): Відповідає за відображення користувацького інтерфейсу (UI). У контексті Android це можуть бути Activity, Fragment або, у випадку з Jetpack Compose, набір Composable-функцій. View спостерігає за змінами у ViewModel (зазвичай через механізми прив'язки даних або спостережувані об'єкти) та оновлює себе відповідно. View також передає дії користувача (наприклад, натискання кнопок, введення тексту) до ViewModel для обробки. Важливо, що View не містить бізнес-логіки чи логіки представлення даних.

3) ViewModel (Модель Подання): Є посередником між Моделлю та Виглядом. ViewModel отримує дані з Моделі, обробляє їх та готує у форматі, зручному для відображення у Вигляді. Вона експонує дані (часто через спостережувані властивості, такі як LiveData або StateFlow), на які може підписатися View. ViewModel також обробляє команди, що надходять від View (наприклад, запити на завантаження даних або виконання певних дій), взаємодіючи при цьому з Моделлю. Ключовою особливістю ViewModel є те, що вона не має прямого посилання на View, що забезпечує слабку зв'язаність та полегшує тестування ViewModel незалежно від UI. ViewModel також часто розробляється таким чином, щоб переживати зміни конфігурації пристрою (наприклад, поворот екрану), зберігаючи свій стан.

Переваги використання MVVM включають покращений розподіл відповідальності, підвищену тестованість (особливо ViewModel), кращу підтримку та масштабованість коду, а також спрощення роботи з життєвим циклом компонентів Android.

3.3.1 Модель(Model):

Шар Моделі інкапсулює дані та бізнес-логіку, пов'язану з ними. Він не залежить від UI і є джерелом правди для застосунку. У розробленому застосунку Модель представлена наступними компонентами:

Класи даних (Data Classes): Ці класи описані у файлах ``FirestoreShelter.kt`` та ``DsnsApiModels.kt`` (в пакеті ``com.akushch.shelterapp.data``). Вони визначають структуру об'єктів сховищ, які отримуються з API ДСНС (наприклад, ``DsnsShelterItem``, ``ApiRegion``) та тих, що зберігаються і використовуються в Firebase Firestore та на клієнті (наприклад, ``FirestoreShelter``). Ці класи містять поля, що відповідають

атрибутам сховищ, визначають типи цих даних та, за потреби, можуть включати методи для перетворення або валідації даних.

Репозиторій (`ShelterRepository`): Цей клас, розташований у файлі ``com.akushch.shelterapp.data.ShelterRepository.kt``, є центральним елементом модельного шару і реалізує патерн "Репозиторій". Він слугує єдиною точкою взаємодії для ViewModel з джерелами даних, абстрагуючи їхню конкретну реалізацію. Основні відповідальності ``ShelterRepository`` включають:

Взаємодію з хмарною базою даних Firebase Firestore: читання даних про сховища (включаючи підписку на оновлення в реальному часі для відображення актуальної інформації) та запис змін (наприклад, оновлення лічильників заповненості при чек-інах/чек-аутах).

Виконання мережевих запитів до API ДСНС: використання HTTP-клієнта для отримання початкового списку укриттів та їхніх базових характеристик. Для обробки JSON-відповідей від API використовується бібліотека Gson [19].

Обробка логіки чек-інів та чек-аутів: оновлення відповідних лічильників у Firestore на основі дій користувачів.

Керування локальним кешуванням даних: збереження отриманих даних (наприклад, у JSON-файлі у внутрішньому сховищі застосунку) для забезпечення базової функціональності застосунку в умовах відсутності або нестабільного інтернет-з'єднання.

Перетворення даних: адаптація даних, отриманих з API ДСНС, у формат, який використовується для зберігання в Firebase Firestore та для відображення на клієнтській стороні.

3.3.2 Вигляд(View):

Шар Вигляду відповідає виключно за візуальне представлення даних користувачеві та за передачу його дій (інтеракцій) до ViewModel. У даному

					ІАЛЦ.467200.002 ТЗ	Арк.
						42
Зм.	Арк.	№ докум.				

проекти, що використовує Jetpack Compose, Вигляд реалізований за допомогою Composable-функцій. Ці функції є декларативними, тобто описують, як має виглядати UI в певному стані, і автоматично реагують на зміни цього стану.

Composable-функції екранів: Основні екрани застосунку, такі як `'HomeScreen'` та `'MapScreen'` (розташовані в пакеті `'com.akushch.shelterapp.screens'`), складаються з набору Composable-функцій. Ці функції отримують дані для відображення від відповідної ViewModel. Вони використовують механізм `'collectAsState()'` для підписки на `'StateFlow'` (потік стану) з ViewModel. При будь-якій зміні даних у `'StateFlow'`, Composable-функції, що на нього підписані, автоматично перекомпонуються (перемальовуються) для відображення актуального стану.

Інтеграція з osmdroid: Для відображення інтерактивної карти використовується бібліотека osmdroid. Компонент `'MapView'` з цієї бібліотеки та пов'язані з ним елементи (такі як маркери сховищ, інформаційні вікна) інтегруються в ієрархію Jetpack Compose за допомогою спеціальної Composable-функції `'AndroidView'`. Логіка оновлення стану карти (наприклад, додавання маркерів, центрування) також керується даними та командами, що надходять від ViewModel.

3.3.3 МодельПодання(ViewModel):

Шар ViewModel виступає як посередник між Моделлю та Виглядом. Його основне завдання – готувати та надавати дані з Моделі у форматі, зручному для відображення у Вигляді, а також обробляти логіку, пов'язану з користувацьким інтерфейсом та діями користувача. ViewModel не має прямого доступу до компонентів View (Activity, Fragment чи конкретних Composable-функцій), що забезпечує їхню слабку зв'язаність та значно полегшує тестування логіки ViewModel.

``MapViewModel``: Цей клас (розташований у файлі ``com.akushch.shelterapp.ui.map.MapViewModel.kt``) є прикладом `ViewModel` у розробленому застосунку. Він успадковується від ``AndroidViewModel``, що надає йому доступ до ``ApplicationContext`` (може бути корисним для ініціалізації деяких сервісів або доступу до ресурсів). ``MapViewModel`` відповідає за всю логіку, пов'язану з екраном карти:

Взаємодія з ``ShelterRepository``: ``MapViewModel`` запитує дані про сховища у ``ShelterRepository`` та отримує їх (часто у вигляді потоку даних, наприклад, ``Flow``).

Зберігання та надання стану для UI: `ViewModel` містить ``StateFlow`` (наприклад, ``_shelters`` як ``MutableStateFlow<List<FirestoreShelter>>``), який зберігає поточний список сховищ для відображення. Цей ``MutableStateFlow`` експонується для `View` як незмінний ``StateFlow``, на який підписуються `Composable`-функції.

Обробка дій користувача: Коли користувач виконує певну дію у `View` (наприклад, натискає кнопку "Чек-ін", пересуває карту, змінюючи видимі межі), `View` повідомляє про це `ViewModel`. `ViewModel` обробляє цю дію, за потреби взаємодіючи з ``ShelterRepository`` (наприклад, викликаючи метод для здійснення чек-іну).

Використання корутин Kotlin: Для виконання довготривалих або асинхронних операцій (таких як мережеві запити або операції з базою даних, що виконуються в ``ShelterRepository``) `ViewModel` використовує корутини Kotlin. Це дозволяє виконувати такі операції без блокування основного потоку, забезпечуючи плавність роботи користувацького інтерфейсу.

Керування станом UI-подій: Для сповіщення `View` про події, що не є частиною основного стану даних (наприклад, необхідність відобразити тимчасове повідомлення `Toast`), `ViewModel` може використовувати окремий ``StateFlow`` або інший механізм передачі одноразових подій.

Нижче наведена схема архітектури MVVM додатку:

					ІАЛЦ.467200.002 ТЗ	Арк.
						44
Зм.	Арк.	№ докум.				

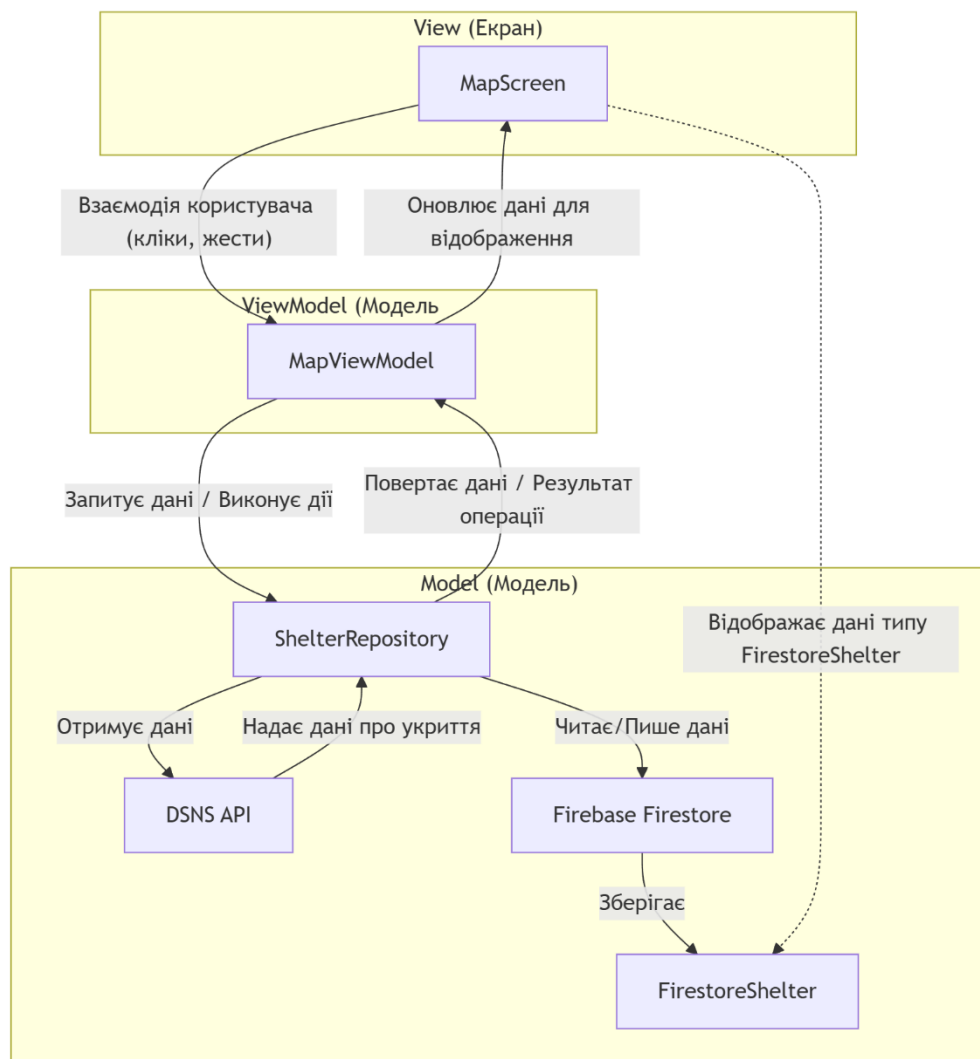


Рисунок 3.2 – Архітектурна схема мобільного застосунку

3.4 Процес автентифікації користувача

Для забезпечення можливості користувачам впливати на дані про заповненість сховищ через механізм чек-ін/чек-аут, а також для унікальної ідентифікації цих дій, у розробленому застосунку реалізовано систему анонімної автентифікації за допомогою сервісу Firebase Authentication [20]. Вибір анонімної автентифікації зумовлений прагненням максимально спростити взаємодію користувача з застосунком у потенційно стресових ситуаціях, уникнути необхідності створення облікового запису з введенням

персональних даних (логін, пароль, email) та зберегти високий рівень приватності користувачів, оскільки основний функціонал не вимагає персоніфікації.

Процес анонімної автентифікації відбувається наступним чином:

1. Ініціалізація та перевірка: При першому запуску мобільного застосунку, або коли користувач намагається виконати дію, що вимагає ідентифікації (наприклад, здійснити чек-ін у сховище), `'MainActivity'` (або відповідний `ViewModel`) звертається до екземпляру `'FirebaseAuth'`. Відбувається перевірка, чи існує вже активний анонімний сеанс для поточного користувача.

2. Анонімний вхід: Якщо активного сеансу немає (це перший запуск застосунку на пристрої, або користувач якимось чином вийшов з попереднього анонімного сеансу), ініціюється процес анонімного входу шляхом виклику методу `'auth.signInAnonymously()'`.

3. Генерація UID: У відповідь на успішний виклик `'signInAnonymously()'`, сервіс `Firebase Authentication` генерує унікальний, випадковий ідентифікатор користувача (UID). Цей UID є стійким для даного екземпляру застосунку на конкретному пристрої, доки користувач не видалить дані застосунку або не відбудеться явний вихід з системи.

4. Збереження та використання UID: Отриманий UID зберігається на клієнті (зазвичай керується `Firebase SDK`) і використовується для всіх подальших операцій, що потребують ідентифікації користувача. Наприклад, при здійсненні чек-іну в сховище, цей UID асоціюється із записом про чек-ін у базі даних `Firebase Firestore`. Це дозволяє системі коректно враховувати дії різних анонімних користувачів та запобігати, наприклад, багаторазовим чек-інам одного користувача в одному сховищі одночасно.

5. Доступ до функціоналу: Після успішної анонімної автентифікації користувач отримує можливість повноцінно взаємодіяти з функціоналом чек-інів та чек-аутів, впливаючи на відображення даних про заповненість сховищ.

					ІАЛЦ.467200.002 ТЗ	Арк.
						46
Зм.	Арк.	№ докум.				

Цей підхід забезпечує необхідний рівень ідентифікації для коректної роботи краудсорсингового механізму, водночас мінімізуючи бар'єри для користувача та зберігаючи його анонімність у контексті відсутності збору персональних даних [21].

3.5 Розгортання системи

Розроблений мобільний застосунок призначений для функціонування на пристроях під управлінням операційної системи Android. Процес розгортання та розповсюдження Android-застосунків передбачає декілька можливих шляхів, кожен з яких має свої особливості:

1. Публікація в офіційному магазині Google Play Store: Це основний, найбільш поширений та рекомендований канал для розповсюдження Android-застосунків серед широкого загалу користувачів. Публікація в Google Play вимагає створення облікового запису розробника Google Play (платна разова реєстрація), ретельної підготовки релізної версії застосунку у форматі Android App Bundle (.aab) або APK (.apk), підписання його цифровим сертифікатом розробника. Також необхідно підготувати маркетингові матеріали (іконка, скріншоти, опис), визначити вікові обмеження, політику конфіденційності та пройти процес модерації з боку Google, який перевіряє відповідність застосунку правилам платформи. Перевагами цього шляху є максимальне охоплення аудиторії, довіра користувачів, механізми автоматичного оновлення, збір статистики та відгуків.

2. Розповсюдження через альтернативні магазини застосунків: Окрім Google Play, існують інші магазини застосунків для Android, такі як Amazon Appstore, Samsung Galaxy Store, Huawei AppGallery та інші. Публікація в них може розширити аудиторію, особливо в регіонах або на пристроях, де ці

магазини є пріоритетними. Процедури публікації та вимоги можуть відрізнятись для кожного магазину.

3. Пряме завантаження APK-файлу (Sideload): Розробник може розмістити інсталяційний APK-файл застосунку на власному веб-сайті, файлообміннику або іншому ресурсі, надаючи користувачам можливість завантажити та встановити його вручну. Цей метод не вимагає проходження модерації магазинів, але має суттєвий недолік: користувачам необхідно в налаштуваннях свого пристрою дозволити встановлення застосунків з невідомих джерел, що може викликати побоювання щодо безпеки. Також відсутні автоматичні оновлення.

4. Внутрішнє тестування та розгортання для обмеженої аудиторії: Перед публікацією для широкого загалу, застосунок зазвичай проходить етапи тестування. Google Play Console надає інструменти для внутрішнього тестування (для команди розробників), закритого альфа-тестування (для обмеженої групи довірених тестувальників) та відкритого бета-тестування (для ширшої аудиторії, яка бажає випробувати нові версії). Платформи, такі як Firebase App Distribution, також дозволяють легко розповсюджувати тестові збірки застосунку серед визначеної групи тестувальників.

На поточному етапі розробки та тестування даного дипломного проєкту, мобільний застосунок встановлювався на фізичні пристрої Android та емулятори безпосередньо з інтегрованого середовища розробки Android Studio. Це дозволяє швидко ітерувати, вносити зміни та перевіряти функціональність. Для подальшого розповсюдження та використання широким колом користувачів, після завершення розробки та ретельного тестування, найбільш доцільним та ефективним шляхом є публікація застосунку в Google Play Store. Це забезпечить максимальну доступність для користувачів, зручність встановлення та отримання оновлень, а також підвищить довіру до розробленого програмного продукту.

ВИСНОВОК ДО РОЗДІЛУ 3

У третьому розділі дипломного проєкту було детально описано процес розробки мобільного застосунку, призначеного для моніторингу наявності вільних місць у найближчих сховищах. Було представлено ключовий функціонал системи, який включає відображення сховищ на інтерактивній карті OpenStreetMap, надання детальної інформації про них, реалізацію механізму краудсорсингового моніторингу заповненості через чек-іни/чек-аути, синхронізацію даних з Firebase Firestore та початкове завантаження інформації з API ДСНС.

Описано основні екрани застосунку ('HomeScreen', 'MapScreen') та їх компоненти, розроблені з використанням Jetpack Compose та принципів Material Design для забезпечення інтуїтивності та зручності користувацького інтерфейсу. Детально розглянуто архітектуру застосунку, побудовану на основі патерну MVVM (Model-View-ViewModel). Було описано компоненти кожного шару: Модель (класи даних, 'ShelterRepository'), Вигляд (Composable-функції екранів) та Модель Подання ('MapViewModel'), а також їхню взаємодію. Було зазначено місця для розміщення відповідних діаграм (загальної архітектурної структури, схеми MVVM та діаграми класів), що візуалізують ці аспекти.

Також було висвітлено процес анонімної автентифікації користувачів за допомогою Firebase Authentication, який дозволяє ідентифікувати дії користувачів для системи чек-інів, не вимагаючи створення повноцінних облікових записів. На завершення розділу розглянуто можливі шляхи розгортання та розповсюдження розробленого Android-застосунку, з акцентом на публікацію в Google Play Store як найбільш доцільний варіант для широкого загалу.

Таким чином, у цьому розділі представлено комплексний опис технічної реалізації мобільного застосунку, що охоплює його функціональні можливості,

					ІАЛЦ.467200.002 ТЗ	Арк.
						49
Зм.	Арк.	№ докум.				

користувацький інтерфейс, архітектурні рішення та підходи до автентифікації й розгортання.

РОЗДІЛ 4. МЕТОДИКА РОБОТИ РОЗРОБЛЕНОГО ЗАСТОСУНКУ

У цьому розділі детально описано послідовність дій користувача при взаємодії з основними функціями розробленого мобільного застосунку для моніторингу наявності вільних місць у сховищах. Кожен крок супроводжується поясненням очікуваної поведінки системи та візуалізується відповідними скріншотами.

4.1 Головний екран та навігація

Після запуску мобільного застосунку користувач потрапляє на Головний екран (HomeScreen). Цей екран є початковою точкою взаємодії.

-Відображення інформації: На Головному екрані відображається назва застосунку (отримана з рядкових ресурсів, наприклад, `context.getString(R.string.app_name)`). Екран має простий дизайн, центруючи увагу користувача на основній інформації та навігації.

Навігація: У нижній частині екрану розташована навігаційна панель (`BottomNavBar`), яка містить вкладки для переходу до основних розділів застосунку. Зазвичай це вкладки "Головна" (поточний екран) та "Карта". При натисканні на вкладку "Карта" відбувається перехід до екрану `MapScreen`.

4.2 Загальний вигляд карти з усіма сховищами

Після переходу на Екран карти ('MapScreen') користувачеві відображається інтерактивна карта (на базі OpenStreetMap), на якій позначено доступні сховища.

Відображення карти: Карта завантажується з початковим масштабом, що охоплює, наприклад, всю територію України (центрована на координатах приблизно 49.0 широти, 32.0 довготи, з масштабом 6.0, як вказано в коді 'MapScreen.kt').

Маркери сховищ: Кожне сховище, дані про яке наявні в системі та мають валідні координати, позначається на карті спеціальним маркером (іконкою). Для оптимізації відображення великої кількості маркерів використовується 'RadiusMarkerClusterer', який групує близькі маркери в кластери при віддаленні карти; при наближенні кластери розпадаються на окремі маркери.

Взаємодія з картою: Користувач може вільно взаємодіяти з картою:

Масштабування: за допомогою жестів "pinch-to-zoom" (зведення/розведення пальців) або кнопок +/-

Панорамування: перетягуванням карти пальцем.

Визначення власного місцезнаходження: на даний момент явна кнопка для центрування на користувачеві не описана в наданому коді 'MapScreen.kt', але такий функціонал є стандартним для картографічних застосунків і може бути доданий.

Завантаження та оновлення даних: При ініціалізації 'MapViewModel' відбувається запит на отримання потоку даних про сховища з 'ShelterRepository'. 'MapScreen' підписується на цей потік і оновлює маркери на карті при надходженні нових даних або їх зміні.

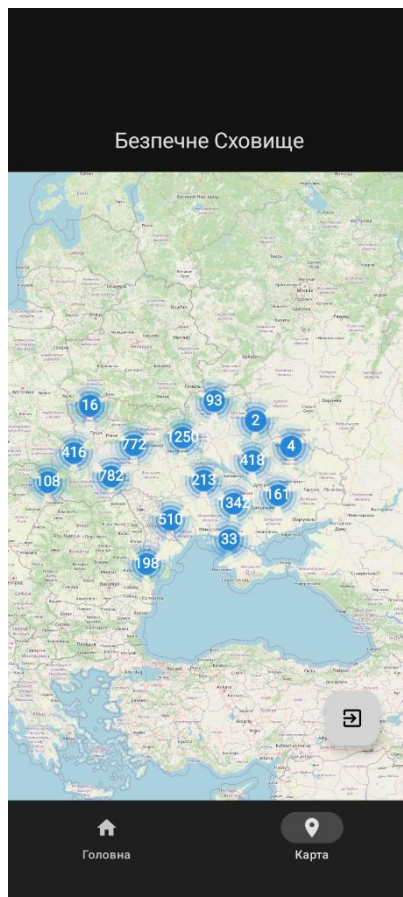


Рисунок 4.1 – Загальний вигляд карти з усіма сховищами

4.3 Зовнішній вигляд маркера (значка) сховища на карті

Маркери, що позначають сховища на карті, мають візуально зрозумілий дизайн для швидкої ідентифікації.

Стандартний вигляд: За замовчуванням, якщо користувач не здійснив чек-ін у сховище, використовується стандартна іконка маркера бібліотеки osmdroid.

Вигляд маркера при чек-іні: Якщо користувач здійснив чек-ін у конкретне сховище (`shelter.documentId == userCheckedInShelterId`), іконка маркера цього сховища змінюється. У коді `MapScreen.kt` робиться спроба завантажити стандартну системну іконку `android.R.drawable.presence_online` (яка зазвичай

виглядає як зелений кружечок, що символізує онлайн-статус або присутність). Якщо ця іконка не знайдена, використовується стандартний маркер.

Кластеризація: Як згадувалося, 'RadiusMarkerClusterer' групує маркери. Вигляд самих кластерів (іконка, що показує кількість згрупованих маркерів) зазвичай визначається бібліотекою 'RadiusMarkerClusterer' і може бути налаштований.

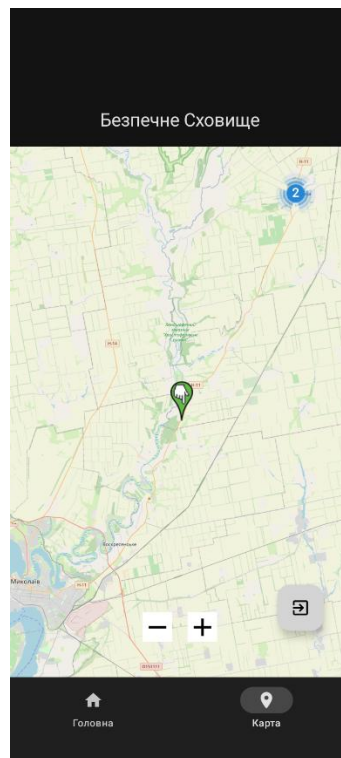


Рисунок 4.2 – Зовнішній вигляд маркера (значка) сховища на карті

4.4 Картка з інформацією про сховище (InfoWindow)

При натисканні користувачем на окремий маркер сховища на карті (або на маркер всередині розкритого кластера) з'являється інформаційне вікно ('InfoWindow'), яке надає детальну інформацію про обране укриття.

Відкриття та закриття InfoWindow: Одночасно на карті може бути відкритим лише одне інформаційне вікно. При натисканні на інший маркер або на вільну ділянку карти попереднє InfoWindow закривається.

					ІАЛЦ.467200.002 ТЗ	Арк.
						54
Зм.	Арк.	№ докум.				

Відображувана інформація: Вміст InfoWindow динамічно формується на основі даних конкретного сховища (``FirestoreShelter``) та поточного статусу чек-іну користувача. Воно включає:

Заголовок (``bubble_title``): Адреса сховища. Якщо користувач зареєстрований у цьому сховищі, до адреси додається індикатор (наприклад, текст " (ви тут)" або відповідний рядок з ресурсів ``R.string.shelter_checked_in_indicator``).

Опис (``bubble_description``): Інформація про загальну місткість та поточну розрахункову заповненість (наприклад, "Місткість: 100, Зараз: 15").

Додатковий опис (``bubble_subdescription``): Детальніша інформація, така як тип будівлі, вид сховища, відповідальна особа та контактний телефон (якщо ці дані наявні). Кожен пункт відображається з відповідним префіксом (наприклад, "Тип: ", "Вид: "). Якщо таких даних немає, цей блок може бути прихованим.

Кнопка дії (``bubble_moreinfo``):

Якщо користувач не зареєстрований у жодному сховищі, кнопка має напис "Зайти" (або відповідний рядок з ``R.string.action_check_in``). Натискання на неї ініціює процес чек-іну в дане сховище через ``mapViewModel.checkIn(currentShelterData.documentId)``.

Якщо користувач вже зареєстрований у цьому сховищі, кнопка має напис "Вийти" (або ``R.string.action_check_out``). Натискання ініціює процес чек-ауту через ``mapViewModel.checkOut()``.

Якщо користувач вже зареєстрований в іншому сховищі, кнопка "Зайти" для поточного сховища стає неактивною (``actionButton.isEnabled = false``), і при спробі натискання може з'являтися повідомлення про те, що користувач вже зареєстрований в іншому місці.



Рисунок 4.3 – Картка з інформацією про сховище (InfoWindow)

4.5 Інформація після входу (чек-іну) в сховище

Після успішного чек-іну користувача в сховище відбуваються наступні зміни в інтерфейсі та системі:

Оновлення InfoWindow (якщо відкрите):

Заголовок InfoWindow для сховища, де відбувся чек-ін, оновлюється, додаючи індикатор реєстрації (наприклад, "(ви тут)").

Текст на кнопці дії змінюється на "Вийти".

Інформація про поточну заповненість сховища оновлюється (лічильник `currentOccupancy` збільшується на 1).

Зміна іконки маркера: Маркер сховища, в якому користувач зареєструвався, змінює свою іконку на `'android.R.drawable.presence_online'` (якщо доступна), візуально позначаючи активний чек-ін.

Оновлення стану в `'MapViewModel'`: `'userCheckedInShelterId'` оновлюється, зберігаючи ID сховища, де користувач здійснив чек-ін. Це впливає на доступність кнопки "Зареєструватися" в InfoWindow інших сховищ (вона стає неактивною).

Відображення кнопки "Загальний вихід": На екрані карти з'являється плаваюча кнопка дії (Floating Action Button - FAB) з іконкою "виходу" (`'Icons.Filled.ExitToApp'`).

Повідомлення користувачу: Застосунок відображає Toast-повідомлення про успішний чек-ін (наприклад, з текстом з `'R.string.toast_checked_in_success'`).

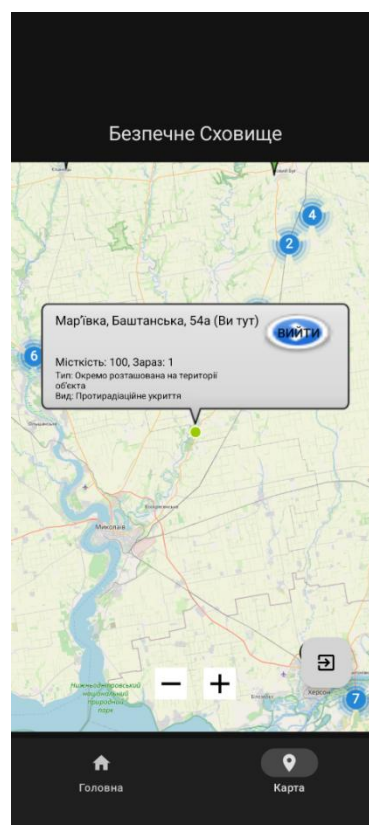


Рисунок 4.4 – Інформація після входу (чек-іну) в сховище

4.6 Інформація після виходу (чек-ауту) зі сховища

Після успішного чек-ауту користувача зі сховища (або через кнопку "Вийти" в InfoWindow, або через загальну кнопку FAB) відбуваються відповідні зміни:

Оновлення InfoWindow (якщо було відкрите для цього сховища):

З заголовка InfoWindow зникає індикатор реєстрації.

Текст на кнопці дії змінюється назад на "Зареєструватися".

Інформація про поточну заповненість сховища оновлюється (лічильник ``currentOccupancy`` зменшується на 1).

Зміна іконки маркера: Маркер сховища, з якого користувач вийшов, повертається до свого стандартного вигляду.

Оновлення стану в ``MapViewModel``: ``userCheckedInShelterId`` встановлюється в ``null``, що робить кнопку "Зареєструватися" знову активною для всіх сховищ.

Приховування кнопки "Загальний вихід": Плаваюча кнопка дії (FAB) для загального виходу зникає з екрану карти.

Повідомлення користувачу: Застосунок відображає Toast-повідомлення про успішний чек-аут (наприклад, з текстом з ``R.string.toast_checked_out_success``).

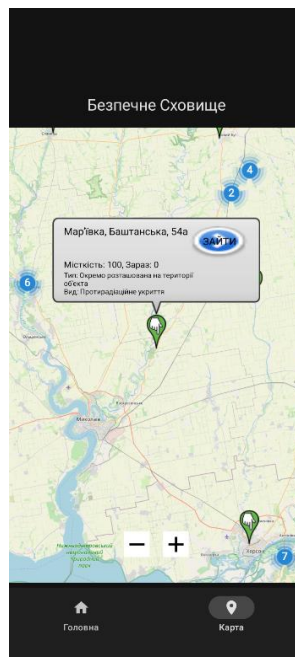


Рисунок 4.5 – Інформація після виходу (чек-ауту) зі сховища

					ІАЛЦ.467200.002 ТЗ	Арк.
						59
Зм.	Арк.	№ докум.				

ВИСНОВОК ДО РОЗДІЛУ 4

У четвертому розділі дипломного проєкту було детально описано методику роботи розробленого мобільного застосунку для моніторингу наявності вільних місць у сховищах. На основі аналізу реалізованого функціоналу та користувацьких сценаріїв було продемонстровано ключові аспекти взаємодії користувача з системою.

Розглянуто початковий екран застосунку ('HomeScreen') та механізми навігації до основного екрану карти ('MapScreen'). Детально описано процес відображення сховищ на інтерактивній карті OpenStreetMap, включаючи візуалізацію маркерів, їх кластеризацію для зручності сприйняття та можливості користувача щодо масштабування та панорамування карти.

Особливу увагу приділено механізму отримання детальної інформації про конкретне сховище через інформаційне вікно ('InfoWindow'), яке з'являється при натисканні на маркер. Описано склад інформації, що надається (адреса, тип, місткість, поточна заповненість, додаткові характеристики), та інтерактивні елементи, зокрема кнопки "Чек-ін" та "Чек-аут".

Продемонстровано зміни в інтерфейсі та стані системи, що відбуваються після успішного чек-іну користувача в сховище: оновлення інформаційного вікна, зміна іконки маркера, поява плаваючої кнопки дії (FAB) для швидкого виходу. Аналогічно описано процес чек-ауту та відповідні зміни у візуальному представленні та даних.

Таким чином, четвертий розділ надав вичерпний опис користувацького досвіду при роботі з основними функціями застосунку, підтверджуючи реалізацію поставлених завдань щодо інформування про сховища та моніторингу їх заповненості на основі краудсорсингу. Опис супроводжувався посиланнями на відповідні скріншоти, що ілюструють кожен етап роботи.

ВИСНОВКИ

У ході виконання даного дипломного проєкту було успішно розроблено мобільний застосунок для платформи Android, призначений для моніторингу наявності вільних місць у найближчих сховищах на території України. Актуальність розробки зумовлена нагальною потребою громадян в оперативній та достовірній інформації про безпечні укриття під час повітряних тривог та інших надзвичайних ситуацій.

На першому етапі було проведено аналіз предметної області, розглянуто існуючі рішення для інформування про сховища та виявлено їхні обмеження, зокрема відсутність динамічного моніторингу заповненості. Це обґрунтувало необхідність створення запропонованої системи.

На другому етапі було здійснено вибір технологічного стеку. Для розробки клієнтської частини було обрано мову програмування Kotlin, UI-фреймворк Jetpack Compose та архітектурний патерн MVVM. Для роботи з картами використано бібліотеку osmdroid (OpenStreetMap), а для серверної частини та синхронізації даних у реальному часі – платформу Firebase (Firestore та Authentication). Було також обґрунтовано вибір методу краудсорсингу (чек-ін/чек-аут) для визначення заповненості сховищ.

Третій розділ присвячено детальному опису процесу розробки застосунку. Розглянуто основний функціонал, структуру користувацького інтерфейсу та ключові екрани. Детально описано реалізацію архітектури MVVM, процес анонімної автентифікації користувачів та можливі шляхи розгортання готового продукту.

У четвертому розділі представлено методику роботи розробленого застосунку з точки зору користувача, описуючи основні сценарії взаємодії: навігацію, перегляд карти та інформації про сховища, а також процеси чек-іну та чек-ауту.

В результаті виконання дипломного проєкту створено функціональний прототип мобільного застосунку, який вирішує поставлені завдання. Застосунок надає користувачам інструмент для пошуку сховищ та отримання інформації про їхню орієнтовну заповненість, що може сприяти підвищенню рівня особистої безпеки. Реалізовано ключові функції, такі як відображення сховищ на карті, детальна інформація про них та механізм чек-інів/чек-аутів.

Розроблений програмний продукт має потенціал для подальшого розвитку, включаючи розширення функціоналу (наприклад, фільтрація сховищ, сповіщення, інтеграція з іншими сервісами), покращення точності даних про заповненість шляхом впровадження більш досконалих алгоритмів валідації, а також розширення географічного покриття та інтеграцію з офіційними державними даними на постійній основі.

Таким чином, завдання дипломного проєкту виконано в повному обсязі, а розроблений мобільний застосунок є актуальним та практично значущим рішенням у сфері цивільного захисту населення.

					ІАЛЦ.467200.002 ТЗ	Арк.
						62
Зм.	Арк.	№ докум.				

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. На Київщині у 2024 році рідше, ніж у 2023-му, оголошували повітряні тривоги, але вони були тривалішими. URL: <https://ps.informator.ua/2025/01/01/na-kyivshhyni-u-2024-roczy-ridshe-nizh-u-2023-mu-ogoloshuvaly-povitryani-tryvogy-ale-vony-buly-tryvalishymy/> (дата звернення: 02.06.2025).
2. Державні будівельні норми. URL: <https://dbn.co.ua/> (дата звернення: 02.06.2025).
3. Сховище, укриття, захисна споруда: які до них вимоги та хто відповідальний. URL: <https://weukraine.tv/top/shovyshhe-ukryttya-zahysna-sporuda-yaki-do-nyh-vymogy-ta-hto-vidpovidalnyj/> (дата звернення: 02.06.2025).
4. Інтерактивна карта укриттів у Дії: як знайти найближче сховище під час повітряної тривоги. URL: <https://visitukraine.today/uk/blog/3412/interactive-map-of-shelters-in-diia-how-to-find-the-nearest-shelter-during-an-air-raid> (дата звернення: 02.06.2025).
5. Мапа укриттів у Києві Цифровий - як переглянути без інтернету. URL: <https://epravda.com.ua/news/2024/02/9/709723/> (дата звернення: 02.06.2025).
6. DozoR укриття. URL: <https://dozor.tech/zastosunok-dozor-ukryttya-dostupno-na-android-ta-ios> (дата звернення: 02.06.2025).
7. ДСНС ресурс. URL: <https://shelters.dsns.gov.ua> (дата звернення: 02.06.2025).
8. Kotlin Documentation. URL: <https://kotlinlang.org/docs/home.html> (дата звернення: 02.06.2025).

9. Kotlin recommended for Android. URL: <https://developer.android.com/kotlin>
(дата звернення: 02.06.2025).
10. Jetpack Compose. URL: <https://developer.android.com/jetpack/compose>
(дата звернення: 02.06.2025). MVVM Example. URL:
<https://learn.microsoft.com/en-us/dotnet/architecture/maui/mvvm> (дата
звернення: 02.06.2025).
11. MVVM architecture in Android. URL:
[https://www.geeksforgeeks.org/mvvm-model-view-viewmodel-architecture-
pattern-in-android/](https://www.geeksforgeeks.org/mvvm-model-view-viewmodel-architecture-pattern-in-android/) (дата звернення: 02.06.2025).
12. Openstreetmap. URL: <https://www.openstreetmap.org/> (дата звернення:
02.06.2025).
13. Osmdroid. URL: <https://github.com/osmdroid/osmdroid> (дата звернення:
02.06.2025).
14. Save data in a local database using Room. URL:
<https://developer.android.com/training/data-storage/room> (дата звернення:
02.06.2025).
15. Json. URL: <https://www.json.org/json-en.html> (дата звернення: 02.06.2025).
16. Get started with Cloud Firestore. URL:
<https://firebase.google.com/docs/firestore/quickstart> (дата звернення:
02.06.2025).
17. Firebase Documentation. URL: <https://firebase.google.com/docs> (дата
звернення 02.06.2025)
18. MVVM vs MVC. URL: <https://shakuro.com/blog/mvc-vs-mvvm> (дата
звернення 02.06.2025)
19. Gson. URL: <https://github.com/google/gson> (дата звернення: 02.06.2025).
20. User authentication using firebase in android. URL:
[https://www.geeksforgeeks.org/user-authentication-using-firebase-in-
android/](https://www.geeksforgeeks.org/user-authentication-using-firebase-in-android/) (дата звернення: 02.06.2025).

21. Anonymous auth in Firebase.

URL:

<https://firebase.google.com/docs/auth/android/anonymous-auth>

(дата

звернення: 02.06.2025).

					ІАЛЦ.467200.002 ТЗ	Арк.
						65
Зм.	Арк.	№ докум.				

ДОДАТОК А

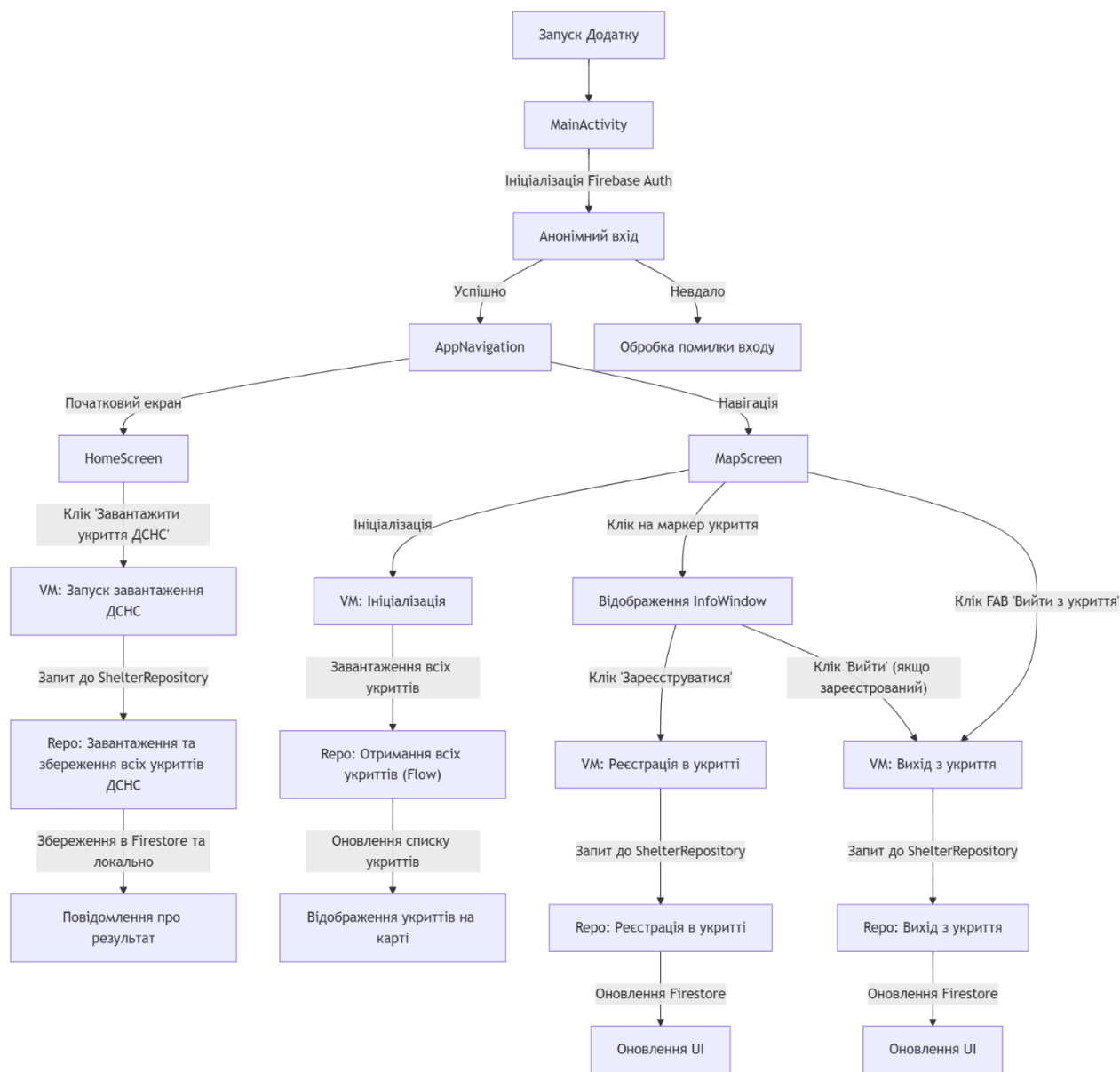
**Довідкова система для моніторингу наявності вільних місць у
найближчих сховищах**

**Алгоритм взаємодії користувача в системі (принципова
схема)**

ІАЛЦ.467200.004 Д1

Аркушів 1

Київ – 2025 р.



					ІАЛЦ.467200.004 Д1										
Зм.	Арк.	№ докум.	Підпис	Дата											
Розробив		Куц А. В.			Довідкова система для моніторингу наявності вільних місць у найближчих сховищах Алгоритм взаємодії користувача в системі (принципова схема)					Літ.		Аркуш	Аркушів		
Перевірив		Павлов В. Г.											1	1	
Реценз.		Корнієнко Б. Я.								КПІ ім. Ігоря Сікорського ФІОТ, ІМ-13					
Н. Контр.		Нікольський С. С.													
Затв.		Новотарський М. А.													

ДОДАТОК Б

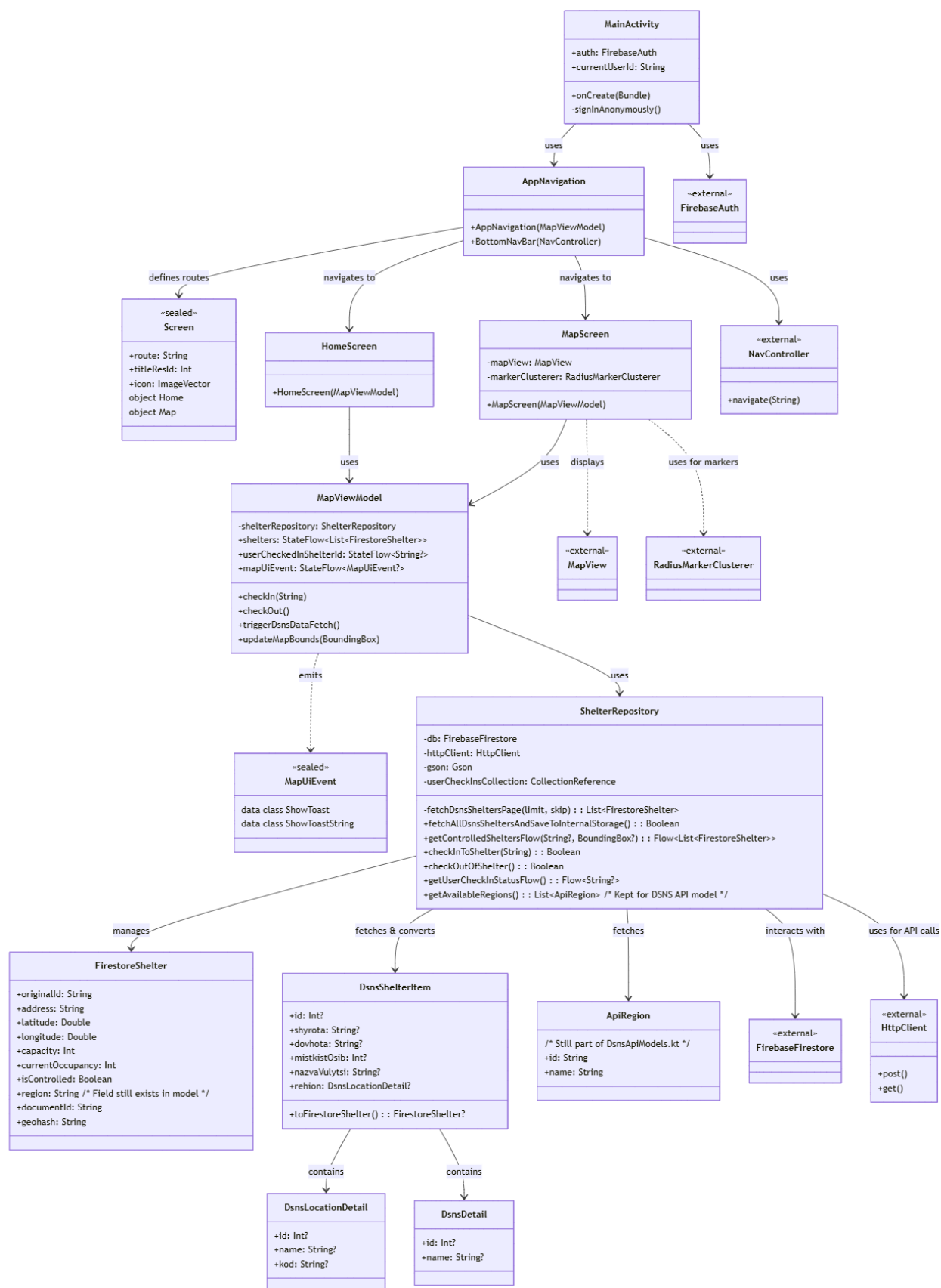
**Довідкова система для моніторингу наявності вільних місць у
найближчих сховищах**

Діаграма класів (функціональна схема)

ІАЛЦ.467200.004 Д2

Аркушів 1

Київ – 2025 р.



ІАЛЦ.467200.004 Д2

					ІАЛЦ.467200.004 Д2									
Зм.	Арк.	№ докум.	Підпис	Дата										
Розробив		Куш А. В.			Довідкова система для моніторингу наявності вільних місць у найближчих сховищах Діаграма класів (функціональна схема)				Літ.		Аркуш		Аркушів	
Перевірив		Павлов В. Г.									1		1	
Реценз.		Корнієнко Б. Я.							КПІ ім. Ігоря Сікорського ФІОТ, ІМ-13					
Н. Контр.		Нікольський С. С.												
Затв.		Новотарський М. А.												

ДОДАТОК В

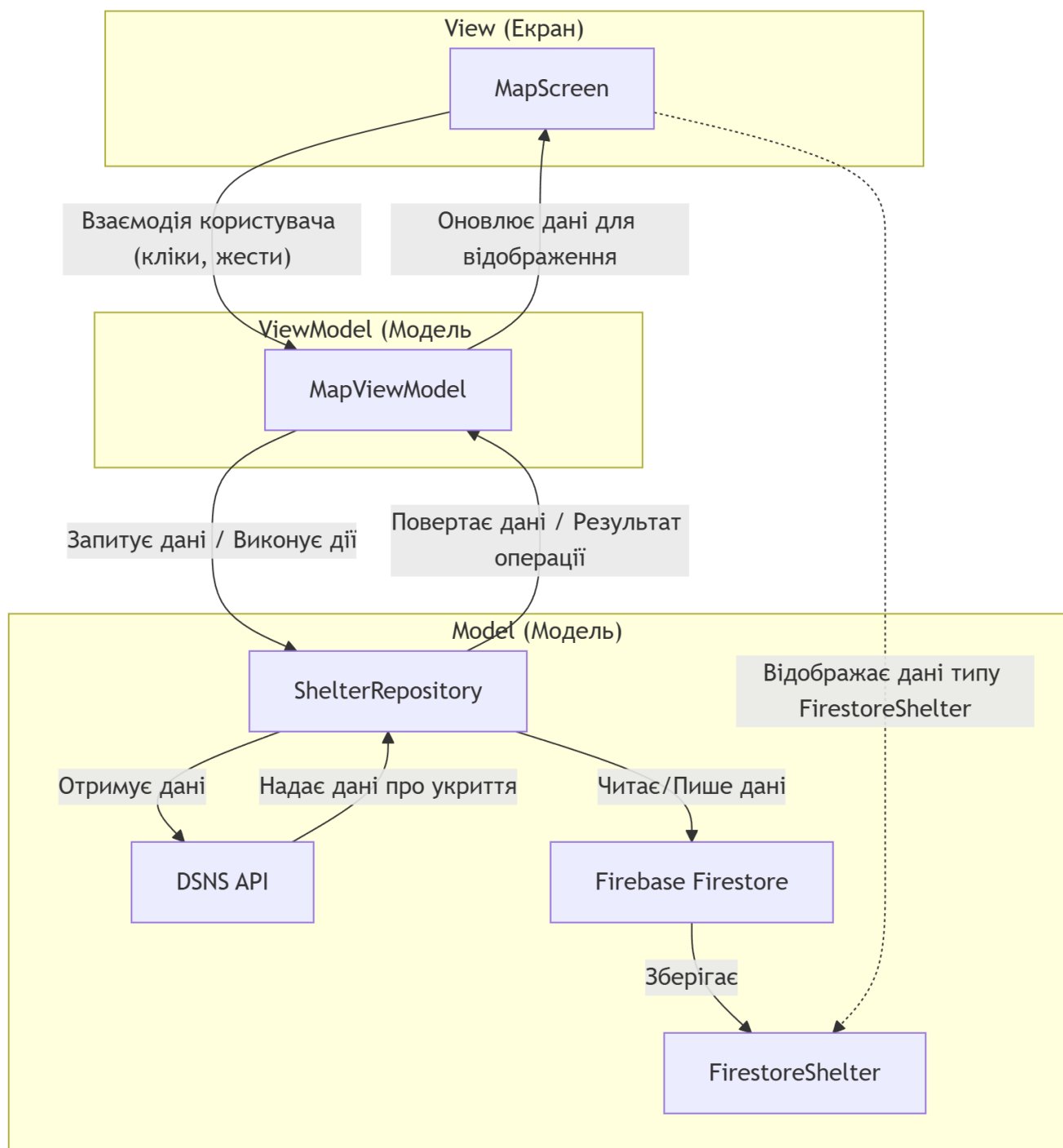
**Довідкова система для моніторингу наявності вільних місць у
найближчих сховищах**

**Архітектурна схема мобільного застосунку (структурна
схема)**

ІАЛЦ.467200.004 ДЗ

Аркушів 1

Київ – 2025 р.



					ІАЛЦ.467200.004 ДЗ									
Зм.	Арк.	№ докум.	Підпис	Дата										
Розробив		Куш А. В.			Довідкова система для моніторингу наявності вільних місць у найближчих сховищах Архітектурна схема мобільного застосунку (структурна схема)				Літ.		Аркуш		Аркушів	
Перевірив		Павлов В. Г.									1		1	
Реценз.		Корнієнко Б. Я.							КПІ ім. Ігоря Сікорського ФІОТ, ІМ-13					
Н. Контр.		Нікольський С. С.												
Затв.		Новотарський М. А.												

ДОДАТОК Г

Довідкова система для моніторингу наявності вільних місць у
найближчих сховищах

Текст програмного коду

ІАЛЦ.467200.007 Д4

Аркушів 43

Київ – 2025 р.

MainActivity.kt

```
package com.akushch.shelterapp
```

```
import android.os.Bundle
```

```
import androidx.activity.ComponentActivity
```

```
import androidx.activity.compose.setContent
```

```
import androidx.activity.enableEdgeToEdge
```

```
import androidx.compose.foundation.layout.fillMaxSize
```

```
import androidx.compose.material3.MaterialTheme
```

```
import androidx.compose.material3.Surface
```

```
import androidx.compose.ui.Modifier
```

```
import com.akushch.shelterapp.navigation.AppNavigation
```

```
import com.akushch.shelterapp.ui.theme.ShelterAppTheme
```

```
import org.osmdroid.config.Configuration
```

```
import com.google.firebase.auth.FirebaseAuth
```

```
import com.google.firebase.auth.ktx.auth
```

```
import com.google.firebase.ktx.Firebase
```

```
import android.util.Log
```

```
class MainActivity : ComponentActivity() {
```

```
    private lateinit var auth: FirebaseAuth
```

```
    companion object {
```

```
        var currentUserId: String? = null
```

```
        private set // Only MainActivity can set it, others can read
```

```
    }
```

```
    override fun onCreate(savedInstanceState: Bundle?) {
```

```
        super.onCreate(savedInstanceState)
```

```
        auth = FirebaseAuth
```

```
        signInAnonymously()
```

					ІАЛЦ.467200.004 Д4			
Зм.	Арк.	№ докум.	Підпис	Дата				
Розробив	Куш А. В.				Довідкова система для моніторингу наявності вільних місць у найближчих сховищах Текст програмного коду	Літ.	Аркуш	Аркушів
Перевірив	Павлов В. Г.						1	44
Реценз.	Корнієнко Б. Я.					КПІ ім. Ігоря Сікорського ФІОТ, ІМ-13		
Н. Контр.	Нікольський С. С.							
Затв.	Новотарський М. А.							

```

// Configure OSMDroid
val ctx = applicationContext
Configuration.getInstance().userAgentValue = packageName
Configuration.getInstance().load(ctx,
androidx.preference.PreferenceManager.getDefaultSharedPreferences(ctx))

enableEdgeToEdge()
setContent {
    ShelterAppTheme {
        Surface(
            modifier = Modifier.fillMaxSize(),
            color = MaterialTheme.colorScheme.background
        ) {
            AppNavigation()
        }
    }
}

private fun signInAnonymously() {
    if (auth.currentUser == null) {
        auth.signInAnonymously()
            .addOnCompleteListener(this) { task ->
                if (task.isSuccessful) {
                    // Sign in success, update UI with the signed-in user's information
                    Log.d("MainActivityAuth", "signInAnonymously:success")
                    val user = auth.currentUser
                    currentUserId = user?.uid
                    Log.d("MainActivityAuth", "User UID: $currentUserId")
                    // You might want to trigger a recomposition or update a ViewModel here
                    // if other parts of your UI depend on the user ID immediately.
                } else {
                    // If sign in fails, display a message to the user.
                    Log.w("MainActivityAuth", "signInAnonymously:failure", task.exception)
                    currentUserId = null
                    // Handle error (e.g., show a toast)
                }
            }
    }
}

```

					ИАЛЦ.467200.002 Д4	Арк.
						2
Зм.	Арк.	№ докум.				

```
    }  
  } else {  
    currentUserId = auth.currentUser?.uid  
    Log.d("MainActivityAuth", "User already signed in. UID: $currentUserId")  
  }  
}  
}
```

					ІАЛЦ.467200.002 Д4	Арк.
						3
Зм.	Арк.	№ докум.				

MapViewModel.kt

```
package com.akushch.shelterapp.ui.map

import android.app.Application
import android.util.Log
import androidx.lifecycle.AndroidViewModel
import androidx.lifecycle.ViewModelScope
import com.akushch.shelterapp.data.FirestoreShelter
import com.akushch.shelterapp.data.ShelterRepository
import com.akushch.shelterapp.MainActivity // To access currentUserId
import com.akushch.shelterapp.R
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.flow.StateFlow
import kotlinx.coroutines.flow.asStateFlow
import kotlinx.coroutines.flow.collectLatest
import kotlinx.coroutines.launch

// Sealed class for UI events/messages to be shown (e.g., Toasts)
sealed class MapUiEvent {
    data class ShowToast(val messageResId: Int, val longDuration: Boolean = false) : MapUiEvent()
    data class ShowToastString(val message: String, val longDuration: Boolean = false) : MapUiEvent() //
    For dynamic messages
}

class MapViewModel(application: Application) : AndroidViewModel(application) {

    private val shelterRepository = ShelterRepository(application.applicationContext)

    private val _shelters = MutableStateFlow<List<FirestoreShelter>>(emptyList())
    val shelters: StateFlow<List<FirestoreShelter>> = _shelters.asStateFlow()

    private val _userCheckedInShelterId = MutableStateFlow<String?>(null)
    val userCheckedInShelterId: StateFlow<String?> = _userCheckedInShelterId.asStateFlow()

    private val _mapUiEvent = MutableStateFlow<MapUiEvent?>(null)
    val mapUiEvent: StateFlow<MapUiEvent?> = _mapUiEvent.asStateFlow()
```

					IAJIЦ.467200.002 Д4	Арк.
						4
Зм.	Арк.	№ докум.				

```

// To hold the current map bounds for querying
private val _currentMapBounds = MutableStateFlow<org.osmdroid.util.BoundingBox?>(null)

init {
    // Fetch all shelters without region filtering
    viewModelScope.launch {
        Log.d("MapViewModel", "Fetching all shelters for Ukraine.")
        shelterRepository.getControlledSheltersFlow(selectedRegion = null, mapBounds = null)
            .collectLatest { shelterList ->
                Log.d("MapViewModel", "Received ${shelterList.size} shelters.")
                _shelters.value = shelterList
            }
    }

    // User check-in status
    viewModelScope.launch {
        shelterRepository.getUserCheckInStatusFlow().collectLatest { checkedInId ->
            Log.d("MapViewModel", "User checked-in status updated: $checkedInId")
            _userCheckedInShelterId.value = checkedInId
        }
    }
}

fun updateMapBounds(bounds: org.osmdroid.util.BoundingBox?) {
    // _currentMapBounds.value = bounds // Temporarily disable map bounds updates to simplify
    // If re-enabled, this should trigger a new collection from the repository
    // with the new bounds, or the repository's Flow should react to bounds changes.
    // For now, we load all shelters once and rely on Firestore updates.
    Log.d("MapViewModel", "updateMapBounds called with: $bounds. Currently not re-fetching based
on bounds.")
}

fun checkIn(shelterId: String) {
    viewModelScope.launch {
        if (MainActivity.currentUserId == null) {
            _mapUiEvent.value = MapUiEvent.ShowToastString(getApplication<Application>().getString(
                R.string.user_not_signed_in))
        }
    }
}

```

					ІАЛЦ.467200.002 Д4	Арк.
						5
Зм.	Арк.	№ докум.				

```

        return@launch
    }
    if (_userCheckedInShelterId.value != null) {
        _mapUiEvent.value =
        MapUiEvent.ShowToast(com.akushch.shelterapp.R.string.toast_already_checked_in_elsewhere)
        return@launch
    }

    val success = shelterRepository.checkInToShelter(shelterId)
    if (success) {
        _mapUiEvent.value =
        MapUiEvent.ShowToast(com.akushch.shelterapp.R.string.toast_checked_in_success)
    } else {
        _mapUiEvent.value =
        MapUiEvent.ShowToast(com.akushch.shelterapp.R.string.toast_checked_in_fail, longDuration = true)
    }
}

fun checkOut() {
    viewModelScope.launch {
        if (MainActivity.currentUserId == null) {
            _mapUiEvent.value =
            MapUiEvent.ShowToastString(getApplication<Application>().getString(R.string.user_not_signed_in))
            return@launch
        }
        val success = shelterRepository.checkOutOfShelter()
        if (success) {
            _mapUiEvent.value =
            MapUiEvent.ShowToast(com.akushch.shelterapp.R.string.toast_checked_out_success)
        } else {
            _mapUiEvent.value =
            MapUiEvent.ShowToast(com.akushch.shelterapp.R.string.toast_checked_out_fail)
        }
    }
}

fun onUiEventHandled() {

```

					ИАЛЦ.467200.002 Д4	Арк.
						6
Зм.	Арк.	№ докум.				

```

        _mapUiEvent.value = null
    }

    fun triggerDsnsDataFetch() {
        viewModelScope.launch {
            _mapUiEvent.value =
            MapUiEvent.ShowToastString(getApplication<Application>().getString(R.string.dsns_fetch_starting))
            val success = shelterRepository.fetchAllDsnsSheltersAndSaveToInternalStorage()
            if (success) {
                _mapUiEvent.value =
                MapUiEvent.ShowToastString(getApplication<Application>().getString(R.string.dsns_fetch_completed))
                // Optionally, refresh data if the map is currently showing old data
                // For now, user might need to navigate away and back or restart to see changes
                // if getControlledSheltersFlow doesn't auto-pick up internal file changes immediately.
                // However, our current getControlledSheltersFlow reads the file on each collection,
                // so forcing a re-collection of _currentMapBounds might refresh it.
                _currentMapBounds.value = _currentMapBounds.value // Force re-emission
            } else {
                _mapUiEvent.value =
                MapUiEvent.ShowToastString(getApplication<Application>().getString(R.string.dsns_fetch_failed))
            }
        }
    }
}

```

MapScreen.kt

```
package com.akushch.shelterapp.screens

import android.widget.Toast
import androidx.compose.foundation.layout.*
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.ExitToApp
import androidx.compose.material3.* // Ensures all Material 3 components are available
import androidx.compose.runtime.*
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.unit.dp
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.platform.LocalLifecycleOwner
import androidx.compose.ui.viewinterop.AndroidView
import androidx.lifecycle.Lifecycle
import androidx.lifecycle.LifecycleEventObserver
import androidx.lifecycle.viewmodel.compose.viewModel
import com.akushch.shelterapp.R
import com.akushch.shelterapp.data.FirestoreShelter
import com.akushch.shelterapp.ui.map.MapUiEvent
import com.akushch.shelterapp.ui.map.MapViewModel
import kotlinx.coroutines.flow.collectLatest
import org.osmdroid.config.Configuration
import org.osmdroid.util.BoundingBox
import org.osmdroid.util.GeoPoint
import org.osmdroid.views.MapView
import org.osmdroid.views.overlay.Marker
import org.osmdroid.views.overlay.infowindow.InfoWindow
import android.widget.Button
import android.widget.TextView
import android.util.Log
import android.view.View
import org.osmdroid.bonuspack.clustering.RadiusMarkerClusterer
import org.osmdroid.events.MapListener
import org.osmdroid.events.ScrollEvent
```

					ИАЛЦ.467200.002 Д4	Арк.
						8
Зм.	Арк.	№ докум.				

```

import org.osmdroid.events.ZoomEvent
import kotlinx.coroutines.Job
import kotlinx.coroutines.delay
import kotlinx.coroutines.launch

@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun MapScreen(mapViewModel: MapViewModel = viewModel()) {
    val context = LocalContext.current
    val lifecycleOwner = LocalLifecycleOwner.current
    val coroutineScope = rememberCoroutineScope()

    val sheltersState by mapViewModel.shelters.collectAsState()
    val userCheckedInShelterId by mapViewModel.userCheckedInShelterId.collectAsState()
    val mapUiEvent by mapViewModel.mapUiEvent.collectAsState()
    // val availableApiRegions by mapViewModel.availableApiRegions.collectAsState() // Now
    List<ApiRegion> - REMOVED
    // val selectedRegionName by mapViewModel.selectedRegion.collectAsState() // Name of the selected
    region - REMOVED

    var initialZoomDone by remember { mutableStateOf(false) }
    var mapMovedJob by remember { mutableStateOf<Job?>(null) }

    LaunchedEffect(mapUiEvent) {
        mapUiEvent?.let { event ->
            when (event) {
                is MapUiEvent.ShowToast -> {
                    Toast.makeText(context, context.getString(event.messageResId), if (event.longDuration)
                    Toast.LENGTH_LONG else Toast.LENGTH_SHORT).show()
                }
                is MapUiEvent.ShowToastString -> {
                    Toast.makeText(context, event.message, if (event.longDuration) Toast.LENGTH_LONG else
                    Toast.LENGTH_SHORT).show()
                }
            }
            mapViewModel.onUiEventHandled()
        }
    }
}

```

					ИАЛЦ.467200.002 Д4	Арк.
						9
Зм.	Арк.	№ докум.				

```

    }

    LaunchedEffect(Unit) {
        Configuration.getInstance().load(context,
        androidx.preference.PreferenceManager.getDefaultSharedPreferences(context))
        Configuration.getInstance().userAgentValue = context.packageName
    }

    val markerClusterer = remember { RadiusMarkerClusterer(context) }
    var openInfoWindowMarkerId by remember { mutableStateOf<String?>(null) } // Track open
    InfoWindow

    // Declare mapView first
    val mapView = remember { MapView(context) }

    // Setup MapEventsOverlay after mapView is initialized
    remember(mapView) { // Re-evaluate if mapView instance changes (though it shouldn't here)
        val mapEventsReceiver = object : org.osmdroid.events.MapEventsReceiver {
            override fun singleTapConfirmedHelper(p: GeoPoint?): Boolean {
                InfoWindow.closeAllInfoWindowsOn(mapView) // Now mapView is accessible
                openInfoWindowMarkerId = null
                return true
            }
            override fun longPressHelper(p: GeoPoint?): Boolean {
                return false
            }
        }
        val mapEventsOverlay = org.osmdroid.views.overlay.MapEventsOverlay(mapEventsReceiver)

        // Configure mapView and add overlays
        mapView.apply {
            setTileSource(org.osmdroid.tileprovider.tilesource.TileSourceFactory.MAPNIK)
            setMultiTouchControls(true)
            controller.setZoom(6.0)
            controller.setCenter(GeoPoint(49.0, 32.0))

            // Ensure overlays are managed correctly, add mapEventsOverlay first
            overlays.clear() // Clear any existing overlays from previous compositions if mapView is reused

```

					ИАЛЦ.467200.002 Д4	Арк.
						10
Зм.	Арк.	№ докум.				

```

overlays.add(0, mapEventsOverlay) // Add at index 0
overlays.add(markerClusterer) // Add markerClusterer after

addMapListener(object : MapListener {
    override fun onScroll(event: ScrollEvent?): Boolean {
        mapMovedJob?.cancel()
        mapMovedJob = coroutineScope.launch {
            delay(500) // Debounce: wait 500ms after scroll stops
            Log.d("MapScreen", "Map scrolled, new bounds: ${boundingBox}")
            mapViewModel.updateMapBounds(boundingBox)
        }
        return true
    }

    override fun onZoom(event: ZoomEvent?): Boolean {
        mapMovedJob?.cancel()
        mapMovedJob = coroutineScope.launch {
            delay(500) // Debounce: wait 500ms after zoom stops
            Log.d("MapScreen", "Map zoomed, new bounds: ${boundingBox}")
            mapViewModel.updateMapBounds(boundingBox)
        }
        return true
    }
})

// Initial bounds update after map is ready
LaunchedEffect(mapView) {
    delay(1000) // Wait a bit for map to initialize fully
    mapViewModel.updateMapBounds(mapView.boundingBox)
}

LaunchedEffect(sheltersState, userCheckedInShelterId, markerClusterer) {
    markerClusterer.items.clear()
    val allMarkersForBounds = mutableListOf<Marker>()

    sheltersState.forEach { shelter ->

```

					ІАЛЦ.467200.002 Д4	Арк.
						11
Зм.	Арк.	№ докум.				

```

if (shelter.latitude != 0.0 && shelter.longitude != 0.0) {
    val shelterPoint = GeoPoint(shelter.latitude, shelter.longitude)
    val marker = Marker(mapView) // Create the marker instance
    marker.position = shelterPoint
    marker.setAnchor(Marker.ANCHOR_CENTER, Marker.ANCHOR_BOTTOM)
    // Default title and snippet, will be overridden in onOpen if InfoWindow is shown
    marker.title = shelter.address.isEmpty { context.getString(R.string.shelter_default_title) }
    marker.snippet = context.getString(R.string.shelter_snippet_format, shelter.capacity,
shelter.currentOccupancy)
    marker.id = shelter.documentId

    // Set icon based on check-in status
    if (shelter.documentId == userCheckedInShelterId) {
        try {
            val checkedInIcon = androidx.core.content.ContextCompat.getDrawable(context,
android.R.drawable.presence_online)
            if (checkedInIcon != null) {
                marker.icon = checkedInIcon
            } else {
                Log.w("MapScreen", "Default checked-in icon (presence_online) not found.")
            }
        } catch (e: Exception) {
            Log.e("MapScreen", "Error setting checked-in icon", e)
        }
    } else {
        // Let osmdroid use its default marker icon if not checked in
    }

    marker.infoWindow = object :
InfoWindow(org.osmdroid.bonuspack.R.layout.bonuspack_bubble, mapView) {
    override fun onOpen(item: Any?) {
        // Close other InfoWindows before opening a new one
        markerClusterer.items.forEach { m ->
            if (m.id != marker.id && m.isInfoWindowShown) {
                m.closeInfoWindow()
            }
        }
    }
}

```

```

openInfoWindowMarkerId = marker.id

val currentShelterData = sheltersState.find { it.documentId == marker.id } ?: return

val titleTextView =
mView.findViewById<TextView>(org.osmdroid.bonuspack.R.id.bubble_title)
    val descriptionTextView =
mView.findViewById<TextView>(org.osmdroid.bonuspack.R.id.bubble_description)
        val subDescriptionTextView =
mView.findViewById<TextView>(org.osmdroid.bonuspack.R.id.bubble_subdescription)
            val actionButton =
mView.findViewById<Button>(org.osmdroid.bonuspack.R.id.bubble_moreinfo)

var titleText = currentShelterData.address.isEmpty
{ context.getString(R.string.shelter_default_title) }
    val isThisShelterCheckedIn = userCheckedInShelterId == currentShelterData.documentId
    if (isThisShelterCheckedIn) {
        titleText += " ${context.getString(R.string.shelter_checked_in_indicator)}"
    }
    titleTextView.text = titleText

descriptionTextView.text = context.getString(R.string.shelter_snippet_format,
currentShelterData.capacity, currentShelterData.currentOccupancy)

val subDescriptionParts = mutableListOf<String>()
if (currentShelterData.typeBuilding.isNotBlank()) {
    subDescriptionParts.add(context.getString(R.string.shelter_info_type,
currentShelterData.typeBuilding))
}
if (currentShelterData.kind.isNotBlank()) {
    subDescriptionParts.add(context.getString(R.string.shelter_info_kind,
currentShelterData.kind))
}
if (currentShelterData.responsiblePerson?.isNotBlank() == true) {
    subDescriptionParts.add(context.getString(R.string.shelter_info_responsible,
currentShelterData.responsiblePerson))
}
if (currentShelterData.phone?.isNotBlank() == true) {

```

					IAJИЦ.467200.002 Д4	Арк.
						13
Зм.	Арк.	№ докум.				

```

        subDescriptionParts.add(context.getString(R.string.shelter_info_phone,
currentShelterData.phone))
    }

    if (subDescriptionParts.isNotEmpty()) {
        subDescriptionTextView.text = subDescriptionParts.joinToString("\n")
        subDescriptionTextView.visibility = View.VISIBLE
    } else {
        subDescriptionTextView.visibility = View.GONE
    }

    actionButton.visibility = View.VISIBLE
    if (isThisShelterCheckedIn) {
        actionButton.text = context.getString(R.string.action_check_out)
        actionButton.setOnClickListener {
            mapViewModel.checkOut()
            // No explicit close() here, rely on state change to re-render or user to tap map
        }
    } else {
        actionButton.text = context.getString(R.string.action_check_in)
        actionButton.isEnabled = userCheckedInShelterId == null
        actionButton.setOnClickListener {
            if (userCheckedInShelterId == null) {
                mapViewModel.checkIn(currentShelterData.documentId)
            } else {
                Toast.makeText(context,
context.getString(R.string.toast_already_checked_in_elsewhere), Toast.LENGTH_SHORT).show()
            }
            // No explicit close()
        }
    }
}

override fun onClose() {
    if (openInfoWindowMarkerId == marker.id) {
        openInfoWindowMarkerId = null
    }
}
}
}

```

```

        markerClusterer.add(marker)
        allMarkersForBounds.add(marker)
    }
}

markerClusterer.invalidate()
mapView.invalidate()

// Simpler InfoWindow refresh: if the open InfoWindow's ID matches the user's checked-in shelter
ID,
// or if the user just checked out from the open InfoWindow, refresh it.
// This is less aggressive but might be more stable.
val currentOpenMarker = openInfoWindowMarkerId?.let { markerId -> markerClusterer.items.find
{ it.id == markerId } }
currentOpenMarker?.let { marker ->
    val shelterDataForOpenMarker = sheltersState.find { s -> s.documentId == marker.id }
    val shouldIndicateCheckedIn = marker.id == userCheckedInShelterId
    val currentlyIndicatesCheckedIn =
marker.title.endsWith(context.getString(R.string.shelter_checked_in_indicator)) ||
                (marker.infoWindow as?
InfoWindow)?.view?.findViewById<TextView>(org.osmdroid.bonuspack.R.id.bubble_title)?.text.toString
().endsWith(context.getString(R.string.shelter_checked_in_indicator)) == true

    val occupancyInSnippet = marker.snippet.split("3apa3: ").getOrNull(1)?.filter
{ it.isDigit() }?.toIntOrNull()
    val currentOccupancyInState = shelterDataForOpenMarker?.currentOccupancy

    if (marker.isInfoWindowShown && (shouldIndicateCheckedIn != currentlyIndicatesCheckedIn ||
occupancyInSnippet != currentOccupancyInState) ) {
        Log.d("MapScreen", "Refreshing InfoWindow for ${marker.id} due to state change.")
        marker.closeInfoWindow()
        marker.showInfoWindow()
        openInfoWindowMarkerId = marker.id // Ensure it's still tracked after reopen
    }
}
}

DisposableEffect(lifecycleOwner) {

```

					IAJIЦ.467200.002 Д4	Арк.
						15
Зм.	Арк.	№ докум.				

```

val observer = LifecycleEventObserver { _, event ->
    when (event) {
        Lifecycle.Event.ON_RESUME -> mapView.onResume()
        Lifecycle.Event.ON_PAUSE -> mapView.onPause()
        else -> {}
    }
}
lifecycleOwner.lifecycle.addObserver(observer)
onDispose {
    lifecycleOwner.lifecycle.removeObserver(observer)
}
}

// var regionDropDownExpanded by remember { mutableStateOf(false) } // REMOVED
// var regionSearchQuery by remember { mutableStateOf("") } // REMOVED

Scaffold(
    topBar = {
        // Region selection dropdown removed
        CenterAlignedTopAppBar(
            title = { Text(context.getString(R.string.app_name)) } // Display App Name or similar
        )
    },
    floatingActionButton = {
        if (userCheckedInShelterId != null) {
            FloatingActionButton(
                onClick = {
                    mapViewModel.checkOut()
                    Toast.makeText(context, context.getString(R.string.toast_attempting_checkout),
Toast.LENGTH_SHORT).show()
                },
                modifier = Modifier.padding(16.dp)
            ) {
                Icon(
                    imageVector = Icons.Filled.ExitToApp,
                    contentDescription = context.getString(R.string.action_force_checkout_desc)
                )
            }
        }
    }
)

```

```

        }
    }
) { innerPadding ->
    Box(
        modifier = Modifier
            .padding(innerPadding)
            .fillMaxSize()
    ) {
        AndroidView(
            factory = { mapView },
            modifier = Modifier.fillMaxSize()
        )
    }
}
}

```

					ІАЛЦ.467200.002 Д4	Арк.
						17
Зм.	Арк.	№ докум.				

HomeScreen.kt

```
package com.akushch.shelterapp.screens

import androidx.compose.foundation.layout.*
import androidx.compose.material3.Button
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.rememberCoroutineScope
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import com.akushch.shelterapp.ui.map.MapViewModel
import kotlinx.coroutines.launch
import android.util.Log
import android.widget.Toast
import com.akushch.shelterapp.R

@Composable
fun HomeScreen(mapViewModel: MapViewModel) {
    val context = LocalContext.current
    // val scope = rememberCoroutineScope() // Already available in MapViewModel

    Box(
        modifier = Modifier.fillMaxSize(),
        contentAlignment = Alignment.Center
    ) {
        Column(horizontalAlignment = Alignment.CenterHorizontally) {
            Text(
                text = context.getString(R.string.app_name), // Example of using string resource
                fontSize = 24.sp,
                modifier = Modifier.padding(16.dp)
            )
            // Spacer(modifier = Modifier.height(20.dp)) // Removed Spacer
            // Button(onClick = { // Removed Button
```

					ИАЛЦ.467200.002 Д4	Арк.
						18
Зм.	Арк.	№ докум.				

```

// // Trigger the DSNS data fetch via the ViewModel
// mapViewModel.triggerDsnsDataFetch()
// }) {
// Text(context.getString(R.string.fetch_all_dsns_shelters_dev_utility))
// }
}
}
}

```

					ІАЛЦ.467200.002 Д4	Арк.
						19
Зм.	Арк.	№ докум.				

AppNavigation.kt

```
package com.akushch.shelterapp.navigation

import androidx.compose.foundation.layout.padding
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.Home
import androidx.compose.material.icons.filled.Place
import androidx.compose.material3.Icon
import androidx.compose.material3.NavigationBar
import androidx.compose.material3.NavigationBarItem
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.getValue
import androidx.compose.ui.graphics.vector.ImageVector
import androidx.compose.ui.res.stringResource
import androidx.annotation.StringRes
import androidx.navigation.NavController
import androidx.navigation.NavGraph.Companion.findStartDestination
import androidx.navigation.compose.NavHost
import androidx.navigation.compose.composable
import androidx.navigation.compose.currentBackStackEntryAsState
import androidx.navigation.compose.rememberNavController
import com.akushch.shelterapp.screens.HomeScreen
import com.akushch.shelterapp.screens.MapScreen
import com.akushch.shelterapp.ui.map.MapViewModel
import androidx.lifecycle.viewmodel.compose.viewModel // For viewModel()

// Define navigation routes
sealed class Screen(val route: String, @StringRes val titleResId: Int, val icon: ImageVector) {
    object Home : Screen("home", com.akushch.shelterapp.R.string.bottom_nav_home, Icons.Filled.Home)
    object Map : Screen("map", com.akushch.shelterapp.R.string.bottom_nav_map, Icons.Filled.Place)
}

// List of all screens for bottom navigation
val items = listOf(
    Screen.Home,
```

					ИАЛЦ.467200.002 Д4	Арк.
						20
Зм.	Арк.	№ докум.				

```

        Screen.Map
    )

@Composable
fun AppNavigation(mapViewModel: MapViewModel = viewModel()) { // Get MapViewModel instance
    val navController = rememberNavController()

    // Scaffold with bottom navigation
    androidx.compose.material3.Scaffold(
        bottomBar = { BottomNavBar(navController) }
    ) { innerPadding ->
        NavHost(
            navController = navController,
            startDestination = Screen.Home.route,
            modifier = androidx.compose.ui.Modifier.padding(innerPadding),
            builder = {
                composable(Screen.Home.route) {
                    HomeScreen(mapViewModel = mapViewModel) // Pass ViewModel
                }
                composable(Screen.Map.route) {
                    MapScreen(mapViewModel = mapViewModel) // Pass ViewModel
                }
            }
        )
    }
}

```

```

@Composable
fun BottomNavBar(navController: NavController) {
    val navBackStackEntry by navController.currentBackStackEntryAsState()
    val currentRoute = navBackStackEntry?.destination?.route

```

```

    NavigationBar {
        items.forEach { screen ->
            NavigationBarItem(
                icon = { Icon(screen.icon, contentDescription = stringResource(id = screen.titleResId)) },
                label = { Text(stringResource(id = screen.titleResId)) },
                selected = currentRoute == screen.route,

```

					ИАЛЦ.467200.002 Д4	Арк.
						21
Зм.	Арк.	№ докум.				

```

onClick = {
  navController.navigate(screen.route) {
    // Pop up to the start destination of the graph to
    // avoid building up a large stack of destinations
    popUpTo(navController.graph.findStartDestination().id) {
      saveState = true
    }
    // Avoid multiple copies of the same destination when
    // reselecting the same item
    launchSingleTop = true
    // Restore state when reselecting a previously selected item
    restoreState = true
  }
}
)
}
}
}

```

					ІАЛЦ.467200.002 Д4	Арк.
						22
Зм.	Арк.	№ докум.				

ShelterRepository.kt

```
package com.akushch.shelterapp.data

import android.content.Context
import android.util.Log
import com.google.firebase.firestore.FirebaseFirestore
import com.google.firebase.firestore.ktx.firestore
import com.google.firebase.ktx.Firebase
import kotlinx.coroutines.tasks.await
import kotlinx.coroutines.flow.Flow
import kotlinx.coroutines.flow.map
import com.google.firebase.firestore.ListenerRegistration
import com.google.firebase.firestore.snapshots
import com.google.gson.Gson
import io.ktor.client.*
import io.ktor.client.engine.cio.*
import io.ktor.client.request.*
import io.ktor.client.statement.*
import io.ktor.http.*
import ch.hsr.geohash.GeoHash // Correct Geohash import
import org.osmdroid.util.BoundingBox // For map bounds parameter
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.withContext
import java.io.File
import java.io.FileOutputStream
import java.io.InputStreamReader

const val ALL_UKRAINE_SHELTERS_FILENAME = "all_ukraine_shelters.json"

class ShelterRepository(private val context: Context) {

    private val db = FirebaseFirestore
    private val httpClient = HttpClient(CIO)
    private val gson = Gson()
    private val userCheckInsCollection = db.collection("user_check_ins")
```

					ІАЛЦ.467200.002 Д4	Арк.
						23
Зм.	Арк.	№ докум.				

```

// Fetches a single page from DSNS API
private suspend fun fetchDsnsSheltersPage(limit: Int, skip: Int): List<FirestoreShelter> {
    val dsnsApiUrl = "https://shelters.dsns.gov.ua/api/v1/public/ukryttya/get"
    val requestBodyJson = ""
    {
        "search": "", "versiyaId": null, "yeVyklyuchenoyu": null, "format": "json", "sortBy": "",
"sortOrder": "",
        "dataProvedennyaOtsinkyStanuHotovnosti": "", "formaVlasnostiId": [], "isMinimumInfo": false,
        "katehoriyaNaselennyaId": [], "limit": $limit, "maxMistkist": null, "maxPloshcha": null,
        "maxRikVvedennyaVEkspluatatsiyu": null, "minMistkist": null, "minPloshcha": null,
        "minRikVvedennyaVEkspluatatsiyu": null, "nayavniZasobyZvyazkuId": [],
        "nayavnistDostupuMalomobilnykhVerstvNaselennya": null, "nayavnistYERODV": null,
        "nazvaBalansoutrymuvacha": "", "nazvaNaselenohoPunktuId": [], "nazvaRayonuMistaId": [],
        "nebezpechniZonyId": [], "oblikovyyNomerMistyt": "", "osnovnyjVydEkonDiyalnostiId": [],
        "rayonId": [], "rehionId": [], "reyestrovyyNomerSporudy": "", "rezhymyFiltroventylyatsiyId":
[],
        "searchOnlyFields": [], "skip": $skip, "stanHotovnostiId": [],
        "statusProvedennyaInventoryzatsiyi": null, "terytorialnaHromadaId": [], "typSporudyId": [],
        "vydSporudyId": [], "yurydychnaAdresaBalansoutrymuvacha": "", "zakhysniVlastyvostiId": []
    }
    """.trimIndent()
    try {
        val response: HttpResponse = httpClient.post(dsnsApiUrl) {
            contentType(ContentType.Application.Json)
            setBody(requestBodyJson)
        }
        if (response.status == HttpStatusCode.OK) {
            val responseBody = response.bodyAsText()
            val dsnsResponse = gson.fromJson(responseBody, DsnsApiResponse::class.java)
            if (dsnsResponse.success && dsnsResponse.data?.items != null) {
                return dsnsResponse.data.items.mapNotNull { it.toFirestoreShelter() }
            } else {
                Log.e("ShelterRepository", "DSNS API page request not successful or data is null. Skip:
$skip, Response: $responseBody")
            }
        } else {
            Log.e("ShelterRepository", "DSNS API page request failed: ${response.status} - Skip: $skip")
        }
    }
}

```

```

    } catch (e: Exception) {
        Log.e("ShelterRepository", "Error fetching DSNS shelters page (skip $skip): ${e.message}", e)
    }
    return emptyList()
}

suspend fun fetchAllDsnsSheltersAndSaveToInternalStorage(): Boolean { // Consider renaming to
fetchAndSyncShelters
    return withContext(Dispatchers.IO) {
        val allShelters = mutableListOf<FirestoreShelter>() // This will store only isControlled=true items
        var skip = 0
        val apiPageLimit = 1000 // API request page size
        var pageCount = 0
        val maxPagesToFetch = 15 // Limit to fetch ~15,000 items from API (15 pages * 1000 items/page)

        Log.d("ShelterRepository", "Starting to fetch DSNS shelters, aiming for up to $maxPagesToFetch
pages (approx. ${maxPagesToFetch * apiPageLimit} raw API items)...")

        try {
            while (pageCount < maxPagesToFetch) { // Loop up to maxPagesToFetch
                pageCount++
                Log.d("ShelterRepository", "Fetching page $pageCount of $maxPagesToFetch (skip $skip,
limit $apiPageLimit)")

                val pageItems = fetchDsnsSheltersPage(apiPageLimit, skip) // pageItems are already
processed by toFirestoreShelter

                if (pageItems.isEmpty()) {
                    if (pageCount == 1) { // If first page is empty, something is wrong with API or request
                        Log.e("ShelterRepository", "First page from DSNS API is empty. Aborting.")
                        return@withContext false
                    }
                    // If a subsequent page is empty, it means API has no more data within our page limit
                    Log.d("ShelterRepository", "No more items from DSNS API after page ${pageCount - 1}.
Fetched all available up to this point.")
                    break
                }

                // Filter for isControlled == true (this was already the logic, ensuring only controlled ones are
added)

```

					ИАЛЦ.467200.002 Д4	Арк.
						25
Зм.	Арк.	№ докум.				

```

        val controlledPageItems = pageItems.filter { it.isControlled } // This line is technically
redundant if fetchDsnsSheltersPage already did this, but safe.

        // The previous change ensured only controlled items are
added to allShelters.

        // Let's assume pageItems from fetchDsnsSheltersPage
are mixed, and we filter here.

        // Actually, the previous change was to filter *within*
this loop. So this is correct.

        allShelters.addAll(controlledPageItems)

        Log.d("ShelterRepository", "API page $pageCount: received ${pageItems.size} items (after
toFirestoreShelter mapping), added ${controlledPageItems.size} controlled items. Total controlled so far:
${allShelters.size}")

        skip += apiPageLimit
        kotlinx.coroutines.delay(200)
    }

    Log.d("ShelterRepository", "Finished fetching from API after $pageCount pages. Total
controlled shelters collected: ${allShelters.size}")

    if (allShelters.isNotEmpty()) {
        val jsonString = gson.toJson(allShelters) // allShelters now only contains isControlled=true
items

        val file = File(context.filesDir, ALL_UKRAINE_SHELTERS_FILENAME)
        FileOutputStream(file).use {
            it.write(jsonString.toByteArray())
        }

        Log.d("ShelterRepository", "All DSNS shelters saved to ${file.absolutePath}")

        // Now, additionally sync these shelters to Firestore
        Log.d("ShelterRepository", "Starting sync of ${allShelters.size} shelters to Firestore...")
        val sheltersCollection = db.collection("shelters")
        var newSheltersAdded = 0
        var existingSheltersSkipped = 0
        for (shelter in allShelters) {
            if (shelter.originalId.isBlank()) {
                Log.w("ShelterRepository", "Skipping shelter with blank originalId: ${shelter.address}")
                continue
            }
        }
    }

```

					IAJIC.467200.002 Д4	Арк.
						26
Зм.	Арк.	№ докум.				

```

try {
    val query = sheltersCollection.whereEqualTo("original_id", shelter.originalId).limit(1)
    val existingDocs = query.get().await()
    if (existingDocs.isEmpty) {
        // Add new shelter to Firestore. Firestore will generate the document ID.
        // The `documentId` field in the `shelter` object itself is not used for the new doc's ID.
        sheltersCollection.add(shelter).await()
        newSheltersAdded++
        Log.d("ShelterRepository", "Added shelter with original_id ${shelter.originalId} to
Firestore.")
    } else {
        existingSheltersSkipped++
        // Optionally, you could update existing documents here if needed
        // Log.d("ShelterRepository", "Shelter with original_id ${shelter.originalId} already in
Firestore. Skipping.")
    }
} catch (e: Exception) {
    Log.e("ShelterRepository", "Error syncing shelter original_id ${shelter.originalId} to
Firestore: ${e.message}", e)
}
}
Log.d("ShelterRepository", "Firestore sync complete. New shelters added:
$newSheltersAdded. Existing skipped/updated: $existingSheltersSkipped.")

true
} else {
    Log.w("ShelterRepository", "No shelters fetched from DSNS to save or sync.")
    false
}
} catch (e: Exception) {
    Log.e("ShelterRepository", "Error in fetchAllDsnsSheltersAndSave (and sync): ${e.message}",
e)
    false
}
}
}

private fun readSheltersFromInternalStorage(): List<FirestoreShelter> {
    return try {

```

```

val file = File(context.filesDir, ALL_UKRAINE_SHELTERS_FILENAME)
if (file.exists()) {
    InputStreamReader(file.inputStream()).use { reader ->
        val type = com.google.gson.reflect.TypeToken.getParameterized(List::class.java,
FirestoreShelter::class.java).type
        gson.fromJson(reader, type) ?: emptyList()
    }
} else {
    Log.w("ShelterRepository", "$ALL_UKRAINE_SHELTERS_FILENAME not found in internal
storage.")
    emptyList()
}
} catch (e: Exception) {
    Log.e("ShelterRepository", "Error reading shelters from internal storage: ${e.message}", e)
    emptyList()
}
}

fun getControlledSheltersFlow(selectedRegion: String? = null, mapBounds: BoundingBox? = null):
Flow<List<FirestoreShelter>> {
    var query: com.google.firebase.firestore.Query = db.collection("shelters")
        .whereEqualTo("is_controlled", true) // Reverted: Filter for controlled shelters

    if (selectedRegion != null) {
        query = query.whereEqualTo("region", selectedRegion)
        Log.d("ShelterRepository", "Filtering shelters for region: $selectedRegion")
    } else {
        Log.d("ShelterRepository", "Fetching shelters for all regions.")
    }

    // Geo-filtering is currently disabled as per previous steps.
    // If re-enabled, it would be applied here based on mapBounds.
    // For now, log that it's not being used for clarity if mapBounds is passed.
    if (mapBounds != null) {
        Log.d("ShelterRepository", "Map bounds provided but geo-filtering is currently disabled. Fetching
based on region filter only (if any).")
    } else {

```

```

        Log.d("ShelterRepository", "No map bounds provided. Fetching based on region filter only (if
any).")
    }

    return query.snapshots()
        .map { querySnapshot ->
            Log.i("ShelterRepository", "Firestore snapshot received. Documents count:
${querySnapshot.size()}. Query: ${query.toString()}")
            if (querySnapshot.metadata.hasPendingWrites()) {
                Log.w("ShelterRepository", "Firestore snapshot has pending writes. Data might be stale
temporarily.")
            }
            querySnapshot.documentChanges.forEach { dc ->
                Log.d("ShelterRepository", "Document change: Type: ${dc.type}, ID: ${dc.document.id}")
            }

            val mappedShelters = querySnapshot.documents.mapNotNull { document ->
                try {
                    val shelter = document.toObject(FirestoreShelter::class.java)?.copy(documentId =
document.id)
                    if (shelter == null) {
                        Log.e("ShelterRepository", "Failed to convert document ${document.id} to
FirestoreShelter. Data: ${document.data}")
                    } else {
                        // Log.v("ShelterRepository", "Successfully mapped document ${document.id}:
${shelter.address}, controlled: ${shelter.isControlled}")
                    }
                    shelter
                } catch (e: Exception) {
                    Log.e("ShelterRepository", "Error converting Firestore document: ${document.id}. Data:
${document.data}", e)
                    null
                }
            }.filter { shelter ->
                val isValidCoordinates = shelter.latitude != 0.0 && shelter.longitude != 0.0
                // if (!isValidCoordinates) Log.w("ShelterRepository", "Filtered out shelter due to invalid
coordinates: ${shelter.documentId} - ${shelter.address}")
            }
        }

```

					ИАЛЦ.467200.002 Д4	Арк.
						29
Зм.	Арк.	№ докум.				

```

        // val isActuallyControlled = shelter.isControlled // This field is part of the query, but good to
        verify if issues persist

        // if (!isActuallyControlled) Log.w("ShelterRepository", "Filtered out shelter because
        isControlled is false after mapping: ${shelter.documentId} - ${shelter.address}")

        isValidCoordinates // && isActuallyControlled // Re-enable isActuallyControlled filter if
        is_controlled query isn't reliable
    }

    Log.i("ShelterRepository", "Mapped shelters count: ${mappedShelters.size}. Emitting to
    ViewModel.")

    mappedShelters
}

// */ // Keep this comment if you want to easily switch back for testing
// --- END OF TARGET FIRESTORE GEO-QUERY LOGIC ---
}

```

```

suspend fun getControlledSheltersFromFirestore(): List<FirestoreShelter> {
    val shelters = mutableListOf<FirestoreShelter>()
    try {
        val result = db.collection("shelters")
            .whereEqualTo("is_controlled", true) // Reverted: Filter for controlled shelters
            .limit(500) // Reverted: Limit for this specific function
            .get()
            .await()
        for (document in result) {
            try {
                shelters.add(document.toObject(FirestoreShelter::class.java).copy(documentId =
                document.id))
            } catch (e: Exception) {Log.e("ShelterRepository", "Error converting document:
                ${document.id}", e)}
        }
    } catch (e: Exception) {Log.e("ShelterRepository", "Error fetching controlled shelters", e)}
    return shelters.filter { it.latitude != 0.0 && it.longitude != 0.0 }
}

```

```

suspend fun checkInToShelter(shelterId: String): Boolean {
    val userId = com.akushch.shelterapp.MainActivity.currentUserId ?: return false
    val shelterRef = db.collection("shelters").document(shelterId)
    val userCheckInRef = userCheckInsCollection.document(userId)
}

```

```

return try {
    db.runTransaction { transaction ->
        val userCheckInDoc = transaction.get(userCheckInRef)
        if (userCheckInDoc.exists()) {
            Log.w("ShelterRepository", "User $userId already checked in. Cannot check in again.")
            return@runTransaction false
        }
        val shelterDoc = transaction.get(shelterRef)
        if (!shelterDoc.exists()){
            Log.e("ShelterRepository", "Shelter document $shelterId does not exist in Firestore for check-
in.")

            return@runTransaction false // Shelter doesn't exist in Firestore to update occupancy
        }
        val currentOccupancy = shelterDoc.getLong("current_occupancy") ?: 0L
        val capacity = shelterDoc.getLong("capacity") ?: 0L
        if (capacity == 0L) {
            Log.w("ShelterRepository", "Shelter $shelterId has 0 capacity.")
            return@runTransaction false
        }
        if (currentOccupancy < capacity) {
            transaction.update(shelterRef, "current_occupancy", currentOccupancy + 1)
            transaction.set(userCheckInRef, mapOf("checked_in_shelter_id" to shelterId, "timestamp" to
com.google.firebase.firestore.FieldValue.serverTimestamp()))
            true
        } else {
            Log.w("ShelterRepository", "Shelter $shelterId is full.")
            false
        }
    }.await()
} catch (e: Exception) {
    Log.e("ShelterRepository", "Error checking in to shelter $shelterId for user $userId", e)
    false
}
}

suspend fun checkOutOfShelter(): Boolean {
    val userId = com.akushch.shelterapp.MainActivity.currentUserId ?: return false
    val userCheckInRef = userCheckInsCollection.document(userId)

```

					IAJЦ.467200.002 Д4	Арк.
						31
Зм.	Арк.	№ докум.				

```

return try {
    db.runTransaction { transaction ->
        val userCheckInDoc = transaction.get(userCheckInRef)
        if (!userCheckInDoc.exists()) {
            Log.w("ShelterRepository", "User $userId is not checked in.")
            return@runTransaction false
        }
        val checkedInShelterId = userCheckInDoc.getString("checked_in_shelter_id")
        if (checkedInShelterId == null) {
            transaction.delete(userCheckInRef) // Clean up malformed doc
            return@runTransaction false
        }
        val shelterRef = db.collection("shelters").document(checkedInShelterId)
        val shelterDoc = transaction.get(shelterRef)
        // It's possible the shelter doc doesn't exist if data is inconsistent, handle gracefully
        if (shelterDoc.exists()) {
            val currentOccupancy = shelterDoc.getLong("current_occupancy") ?: 0L
            if (currentOccupancy > 0) {
                transaction.update(shelterRef, "current_occupancy", currentOccupancy - 1)
            }
        } else {
            Log.w("ShelterRepository", "Shelter $checkedInShelterId not found in Firestore during
checkout for user $userId.")
        }
        transaction.delete(userCheckInRef)
        true
    }.await()
} catch (e: Exception) {
    Log.e("ShelterRepository", "Error checking out user $userId", e)
    false
}
}

fun getUserCheckInStatusFlow(): Flow<String?> {
    val userId = com.akushch.shelterapp.MainActivity.currentUserId
    if (userId == null) return kotlinx.coroutines.flow.flowOf(null)
    return userCheckInsCollection.document(userId)
        .snapshots()

```

					ИАЛЦ.467200.002 Д4	Арк.
						32
Зм.	Арк.	№ докум.				

```

        .map { it.getString("checked_in_shelter_id") }
    }

suspend fun getAvailableRegions(): List<ApiRegion> { // Now returns List<ApiRegion>
    val regionsApiUrl = "https://shelters.dsns.gov.ua/api/v1/public/katootth/regions"
    try {
        val response: HttpResponse = httpClient.get(regionsApiUrl)
        if (response.status == HttpStatusCode.OK) {
            val responseBody = response.bodyAsText()
            // Assuming the API returns a direct JSON array of ApiRegion objects
            val typeToken = com.google.gson.reflect.TypeToken.getParameterized(List::class.java,
ApiRegion::class.java).type
            val apiRegions: List<ApiRegion> = gson.fromJson(responseBody, typeToken) ?: emptyList()

            // Filter out any regions with blank names, sort them, and log
            val validRegions = apiRegions.filter { it.name.isNotBlank() }.sortedBy { it.name }
            Log.d("ShelterRepository", "Fetched ${validRegions.size} valid regions from API:
$validRegions")
            return validRegions
        } else {
            Log.e("ShelterRepository", "Failed to fetch regions from KATOOTTH API:
${response.status}")
        }
    } catch (e: Exception) {
        Log.e("ShelterRepository", "Error fetching regions from KATOOTTH API: ${e.message}", e)
    }
    return emptyList()
}
}

```

ShelterModels.kt

```
package com.akushch.shelterapp.data

import com.google.firebase.firestore.PropertyName
// Renaming to FirestoreShelter to represent a document in Firestore
data class FirestoreShelter(
    @get:PropertyName("original_id") @set:PropertyName("original_id") var originalId: String = "",
    var address: String = "",
    var district: String = "",
    var latitude: Double = 0.0,
    var longitude: Double = 0.0,
    @get:PropertyName("type_building") @set:PropertyName("type_building") var typeBuilding: String =
    "",
    var kind: String = "",
    var capacity: Int = 0,
    @get:PropertyName("current_occupancy") @set:PropertyName("current_occupancy") var
    currentOccupancy: Int = 0,
    @get:PropertyName("is_controlled") @set:PropertyName("is_controlled") var isControlled: Boolean =
    false,
    @get:PropertyName("responsible_person") @set:PropertyName("responsible_person") var
    responsiblePerson: String? = null,
    var phone: String? = null,
    var region: String = "", // Added field for region
    // Firestore document ID, will be populated after fetching
    @get:PropertyName("document_id") @set:PropertyName("document_id") var documentId: String = "",
    var geohash: String = "" // For geo-queries
) {
    // Default constructor is required for Firestore deserialization
    constructor() : this("", "", "", 0.0, 0.0, "", "", 0, 0, false, null, null, "", "", "")
}
```

DsnsApiModels.kt

```
package com.akushch.shelterapp.data

import com.google.gson.annotations.SerializedName
import ch.hsr.geohash.GeoHash // Switched to HSR Geohash library

// Top-level response structure
data class DsnsApiResponse(
    val success: Boolean,
    val statusCode: Int,
    val data: DsnsData?
)

data class DsnsData(
    val items: List<DsnsShelterItem>?
    // Potentially other fields like 'totalCount' if API supports it
)

// Detailed shelter item from DSNS API
data class DsnsShelterItem(
    val id: Int?,
    val oblikovyyNomer: String?,
    val dataVzyattyaSporudyNaOblik: String?, // ISO Date String
    val informatsiyaProVlasnykiv: List<DsnsOwnerInfo>?,
    val ploshcha: Double?, // Area
    val mistkistOsib: Int?, // Capacity
    val nazvaVulytsi: String?,
    val inshiRekvizytyAdresy: String?, // Building number etc.
    val shyrota: String?, // Latitude as String
    val dovhota: String?, // Longitude as String
    val pibVidpovidalnoyiOsoby: String?,
    val telefonVidpovidalnoyiOsoby: String?,
    val statusProvedennyaInventoryzatsiyi: Boolean?,
    val kilkistVkhodivVykhodiv: Int?,
    val kilkistAvariynykhVykhodiv: Int?,
    val systemaOpalennya: String?,
```

					ІАЛЦ.467200.002 Д4	Арк.
						35
Зм.	Арк.	№ докум.				

```

val systemaEletrozhyvlennya: String?,
val systemaVodopostachannya: String?,
val systemaVentilyatsiyi: String?,
val systemaKanalizatsiyi: String?,
val prymitka: String?,
val yeVyklyuchenoyu: Boolean?,

```

```
// Nested objects for detailed types/statuses
```

```
@SerializedName("VydSporudy") val vydSporudy: DsnsDetail?, // Shelter Type (e.g., Anti-radiation)
```

```
@SerializedName("VidomchaPrynalezhnist") val vidomchaPrynalezhnist: DsnsDetail?, // Departmental
affiliation
```

```
@SerializedName("TypSporudy") val typSporudy: DsnsDetail?, // Building Type (e.g., Built-in)
```

```
@SerializedName("ZakhysniVlastyvosti") val zakhysniVlastyvosti: DsnsDetail?, // Protective properties
```

```
@SerializedName("FormaVlasnosti") val formaVlasnosti: DsnsDetail?, // Ownership form
```

```
@SerializedName("StanHotovnosti") val stanHotovnosti: DsnsDetail?, // Readiness state
```

```
@SerializedName("FunktsionalnePryznachennya") val funktsionalnePryznachennya: DsnsDetail?, //
```

```
Functional purpose
```

```
@SerializedName("Rehion") val rehion: DsnsLocationDetail?,
```

```
@SerializedName("Rayon") val rayon: DsnsLocationDetail?,
```

```
@SerializedName("TerytorialnaHromada") val terytorialnaHromada: DsnsLocationDetail?,
```

```
@SerializedName("NazvaNaselenohoPunktu") val nazvaNaselenohoPunktu: DsnsLocationDetail?,
```

```
val naimenuvanniaMistsiaDliaUkryttia: String?, // Name/description of shelter place
```

```
@SerializedName("imya") val imyaNested: String?, // Seems to be a duplicate of VydSporudy.imya or
similar
```

```
val createdAt: String?, // ISO Date String
```

```
val updatedAt: String? // ISO Date String
```

```
)
```

```
data class DsnsOwnerInfo(
```

```
val vlasnyk: Boolean?,
```

```
@SerializedName("kodYEDRPOU") val kodYedrpou: String?,
```

```
val yurydychnaAdresa: String?,
```

```
val naymenuvannyaOsoby: String?, // Name of owner/responsible entity
```

```
val kontaktnyyNomerTelefonu: String?,
```

```
val kilkistOsibUkryttya: Int? // Capacity from owner's perspective
```

					ІАЛЦ.467200.002 Д4	Арк.
						36
Зм.	Арк.	№ докум.				

```

)

// Generic class for nested detail objects like VydSporudy, FormaVlasnosti etc.
data class DsnsDetail(
    val id: Int?,
    @SerializedName("imya") val name: String? // "imya" seems to be the common field for
name/description
)

// Generic class for nested location detail objects
data class DsnsLocationDetail(
    val id: Int?,
    @SerializedName("imya") val name: String?,
    val kod: String?
    // other fields like 'analitika' are ignored for now
)

// Data class for the /katootth/regions API response
data class ApiRegion(
    @SerializedName("id") // Assuming 'id' or similar unique identifier from API
    val id: String,
    @SerializedName("name", alternate = ["nazva"]) // Assuming 'name' or 'nazva' from API
    val name: String
)

// If the API returns a root object containing a list, adjust accordingly:
// data class KatootthRegionsResponse(val data: List<ApiRegion>)

// Helper function to convert DSNS item to our FirestoreShelter model
fun DsnsShelterItem.toFirestoreShelter(): FirestoreShelter? {
    val lat = shyrota?.toDoubleOrNull()
    val lon = dovhota?.toDoubleOrNull()

    // Validate essential data including coordinate ranges
    if (id == null || lat == null || lon == null ||
        lat < -90.0 || lat > 90.0 || // Standard latitude range
        lon < -180.0 || lon > 180.0) { // Standard longitude range

```

```

        android.util.Log.w("DsnsApiModels", "Skipping shelter ID $id due to missing essential data or
invalid coordinates: lat=$lat, lon=$lon")
        return null
    }

    // Determine address
    val addressParts = mutableListOf<String?>()
    nazvaNaselenohoPunktu?.name?.let { addressParts.add(it) }
    nazvaVulytsi?.let { addressParts.add(it) }
    inshiRekvizytyAdresy?.let { addressParts.add(it) }
    val fullAddress = addressParts.filterNotNull().joinToString(", ").ifEmpty { "Адреса не вказана" }

    // Refined logic to determine 'isControlled' status
    var determinedIsControlled = false

    // 1. Explicit Exclusions (Highest Priority)
    val mainDescription = this.imyaNested // Corrected from this.imya
    if (mainDescription != null && mainDescription.equals("Підвальне приміщення", ignoreCase = true))
    {
        determinedIsControlled = false
    } else {
        val functionalPurposeId = funktsionalnePryznachennya?.id
        val excludedFunctionalPurposeIds = setOf(2, 5) // 2:Складські, 5:Гаражі/стоянки
        if (functionalPurposeId != null && functionalPurposeId in excludedFunctionalPurposeIds) {
            determinedIsControlled = false
        } else {
            // 2. Strong Positive Indicators
            val typeSporudyId = typSporudy?.id
            val positiveTypeSporudyIds = setOf(3, 4, 6, 96, 20)
            // 3:освіти, 4:соц.захисту, 6:культури, 96:охорони здоров'я, 20:спортивні

            val positiveFunctionalPurposeIds = setOf(6, 7, 4, 13)
            // 6:навч.занять, 7:культури, 4:охорони здоров'я, 13:адмін/офісні

            val hasValidResponsiblePerson = pibVidpovidalnoyiOsoby != null &&
            pibVidpovidalnoyiOsoby.isNotBlank() && pibVidpovidalnoyiOsoby != "-"

```

					ІАЛЦ.467200.002 Д4	Арк.
						38
Зм.	Арк.	№ докум.				

```

val isGoodShelterType = vydSporudy?.name.equals("Сховище", ignoreCase = true) ||
    vydSporudy?.name.equals("Протирадіаційне укриття", ignoreCase = true)

// Reverted to less strict: OR condition for positive indicators
val hasStrongPositiveIndicator = (typeSporudyId != null && typeSporudyId in
positiveTypeSporudyIds) ||
    (functionalPurposeId != null && functionalPurposeId in
positiveFunctionalPurposeIds) ||
    hasValidResponsiblePerson ||
    isGoodShelterType

// Apply the main logic, including special handling for residential (TypSporudy.id == 1)
if (typeSporudyId == 1 && !hasStrongPositiveIndicator) { // Residential building without other
strong indicators
    determinedIsControlled = false
} else if (hasStrongPositiveIndicator) {
    determinedIsControlled = true
} else {
    // Default to false if no strong positive indicators and not explicitly excluded prior
    determinedIsControlled = false
}
}

// Ensure explicit exclusions override any positive indicators.
// The nested if/else structure above should handle this, but re-checking for clarity/safety.
if (mainDescription != null && mainDescription.equals("Підвальне приміщення", ignoreCase = true))
{
    determinedIsControlled = false
}

// Re-check functional purpose exclusion, as it might have been overridden by a positive indicator above
val functionalPurposeIdAfterPositiveCheck = funktsionalnePryznachennya?.id
val excludedFunctionalPurposeIdsAfterPositiveCheck = setOf(2, 5) // 2:Складські, 5:Гаражі/стоянки
if (functionalPurposeIdAfterPositiveCheck != null && functionalPurposeIdAfterPositiveCheck in
excludedFunctionalPurposeIdsAfterPositiveCheck) {
    determinedIsControlled = false
}

```

```

return FirestoreShelter(
    originalId = id.toString(),
    address = fullAddress,
    district = rayon?.name ?: "",
    region = rehion?.name ?: "",
    latitude = lat,
    longitude = lon,
    typeBuilding = typSporudy?.name ?: vydSporudy?.name ?: mainDescription ?: "",
    kind = vydSporudy?.name ?: typSporudy?.name ?: "",
    capacity = mistkistOsib ?: informatsiyaProVlasnykiv?.firstOrNull()?.kilkistOsibUkryttya ?: 0,
    currentOccupancy = 0,
    isControlled = determinedIsControlled, // Use the refined flag
    responsiblePerson = pibVidpovidalnoyiOsoby,
    phone = telefonVidpovidalnoyiOsoby,
    documentId = "",
    geohash = GeoHash.withCharacterPrecision(lat, lon, 9).toBase32()
)
}

```

					ІАЛЦ.467200.002 Д4	Арк.
						40
Зм.	Арк.	№ докум.				

Strings.xml

<resources>

<string name="app_name">Безпечне Сховище</string>

<string name="shelter_default_title">Укриття</string>

<string name="shelter_snippet_format">Місткість: %1\$d, Запас: %2\$d</string>

<string name="action_check_out">Вийти</string>

<string name="action_check_in">Зайти</string>

<string name="toast_checked_out_success">Ви успішно вийшли!</string>

<string name="toast_checked_out_fail">Не вдалося вийти.</string>

<string name="toast_checked_in_success">Ви успішно зайшли!</string>

<string name="toast_checked_in_fail">Не вдалося зайти (можливо, укриття повне або ви вже в іншому).</string>

<string name="toast_already_checked_in_elsewhere">Ви вже зареєстровані в іншому укритті.</string>

<string name="toast_attempting_checkout">Спроба вийти...</string>

<string name="action_force_checkout_desc">Примусово вийти з укриття</string>

<string name="fetch_all_dsns_shelters_dev_utility">Завантажити всі укриття ДСНС (Для розробника)</string>

<string name="user_not_signed_in">Користувач не авторизований.</string>

<string name="dsns_fetch_starting">Починаю завантаження даних ДСНС... Це може зайняти кілька хвилин. Перевірте Logcat.</string>

<string name="dsns_fetch_completed">Завантаження даних ДСНС завершено та збережено.</string>

<string name="dsns_fetch_failed">Не вдалося завантажити дані ДСНС. Перевірте Logcat.</string>

<string name="shelter_info_type">Тип: %1\$s</string>

<string name="shelter_info_kind">Вид: %1\$s</string>

<string name="shelter_info_responsible">Відповідальний: %1\$s</string>

<string name="shelter_info_phone">Телефон: %1\$s</string>

<string name="shelter_checked_in_indicator">(Ви тут)</string>

<string name="all_regions_filter">Всі регіони</string>

<string name="region_filter_label">Region</string>

<string name="region_search_no_results">No results for \"%1\$s\"</string>

<string name="bottom_nav_home">Головна</string>

<string name="bottom_nav_map">Карта</string>

					ІАЛЦ.467200.002 Д4	Арк.
						41
Зм.	Арк.	№ докум.				

</resources>

					ІАЛЦ.467200.002 Д4	Арк.
						42
Зм.	Арк.	№ докум.				

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">

    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"
android:maxSdkVersion="28" />
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />

    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportRtl="true"
        android:largeHeap="true"
        android:theme="@style/Theme.ShelterApp"
        tools:targetApi="31">
        <activity
            android:name=".MainActivity"
            android:exported="true"
            android:label="@string/app_name"
            android:theme="@style/Theme.ShelterApp">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
```

					ІАЛЦ.467200.002 Д4	Арк.
						43
Зм.	Арк.	№ докум.				

</manifest>

					ІАЛЦ.467200.002 Д4	Арк.
						44
Зм.	Арк.	№ докум.				