МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ІВАНА ФРАНКА

Факультет прикладної математики та інформатики
(повне найменування назва факультету)

кафедра інформаційних систем
(повна назва кафедри)

ДИПЛОМНА РОБОТА

Проєктування мобільного додатку “Карта укриттів”

Студента 4 курсу, групи ПМі-44,
спеціальності 122 – комп’ютерні науки

Маліш Б.Б.
(підпис) (прізвище та ініціали)

Керівник _____
(підпис) (прізвище та ініціали)

Рецензент _____
(підпис) (прізвище та ініціали)

Львів 2023

ЗМІСТ

ВСТУП

Мобільні додатки стали невід'ємною частиною повсякденного життя. Завдяки їм можна отримати швидкий доступ до різноманітної інформації та виконувати різні завдання та спілкуватися з користувачами в будь-який час і з будь-якого місця. Мобільні додатки дозволяють використовувати смартфони та планшети для розваг, освіти, роботи, здоров'я, фітнесу, подорожей та багатьох інших сфер життя. Вони пропонують широкий вибір функціональності, включаючи сповіщення, геолокацію, камеру, датчики, інтеграцію з соціальними мережами та багато іншого. З ростом популярності мобільних додатків зростає і їхня роль у різних галузях. Вони стають потужним інструментом для бізнесу, дозволяючи компаніям просувати свої продукти та послуги, залучати нових клієнтів і покращувати комунікацію зі своєю аудиторією. Крім того, мобільні додатки можуть сприяти ефективній організації та управлінню різними процесами, забезпечувати доступ до особистої інформації та сприяти розвитку освіти та культури. Розробка мобільних додатків вимагає не тільки технічної експертизи, але й уважного аналізу потреб користувачів та створення зручного та привабливого інтерфейсу. Постійне зростання ринку мобільних додатків викликає конкуренцію серед розробників та підтримку нових інновацій і технологій. Таким чином, мобільні додатки стали невід'ємною частиною нашого цифрового життя, пропонуючи нам широкий спектр можливостей та забезпечуючи зручність та доступність у використанні.

На сьогоднішній день спостерігається значний тренд у напрямку переходу багатьох компаній до підходу "**APP-FIRST**" (спрямованість на мобільні додатки). Це означає, що компанії активно інвестують у розробку та підтримку мобільних додатків як основного каналу взаємодії зі своїми клієнтами та користувачами. Існує кілька причин, чому багато компаній переходять до стратегії "APP-FIRST":

1. Зростання використання мобільних пристроїв: Мобільні пристрої, такі як смартфони та планшети, стали невід'ємною частиною повсякденного життя. Вони забезпечують зручний доступ до інформації та послуг в будь-який час і з будь-якого місця. Це робить мобільні додатки ефективним і ефективним способом залучення та задоволення потреб користувачів.

2. Покращений користувацький досвід: Мобільні додатки надають більш персоналізований та зручний користувацький досвід порівняно з веб-сайтами. Вони можуть використовувати функції пристрою, такі як камера, GPS та сповіщення, для створення більш інтерактивного та привабливого досвіду. Крім того, мобільні додатки можуть працювати в автономному режимі, що дозволяє користувачам використовувати їх без доступу до Інтернету.

3. Більш ефективна комунікація та взаємодія: Мобільні додатки надають компаніям можливість безпосередньо спілкуватися зі своїми клієнтами через сповіщення, особисті повідомлення та інші комунікаційні канали. Це дозволяє забезпечити швидку відповідь на запити та підтримку користувачів, покращуючи задоволення клієнтів.

4. Збільшення залученості та лояльності: Мобільні додатки можуть бути використані для створення програм лояльності, бонусних систем та персоналізованих пропозицій, що спонукають користувачів до повернення та використання додатку на постійній основі. Це допомагає підвищити залученість та збільшити лояльність клієнтів.

5. Розширення ринкових можливостей: Розробка мобільних додатків відкриває нові можливості для компаній у розширенні своєї аудиторії та досягненні нових ринків. Мобільні додатки можуть бути доступними для завантаження в магазинах додатків, що забезпечує їм більшу видимість та доступність для потенційних користувачів.

6. Можливість персоналізації: Мобільні додатки дозволяють компаніям збирати та аналізувати дані про поведінку користувачів. Це дозволяє створювати персоналізовані пропозиції, рекомендації та контент для кожного користувача на основі його індивідуальних потреб та вподобань. Це сприяє поліпшенню задоволення клієнтів та збільшенню ефективності маркетингових зусиль.

7. Використання мобільних функцій: Мобільні додатки мають доступ до різних функцій пристрою, таких як камера, геолокація, мікрофон та інші. Це дозволяє розробникам створювати унікальні та інноваційні функції, які використовують потужні можливості мобільних пристроїв. Наприклад, додаток "Карти укриттів під час повітряних тривог" може використовувати геолокацію для визначення місцеположення користувача та надання актуальної інформації щодо найближчих укриттів.

8. Забезпечення більш глибокої взаємодії: Мобільні додатки дозволяють створювати більш глибоку та безпосередню взаємодію з користувачами через використання різних функцій, таких як оповіщення, сповіщення, інтерактивність жестами та багато іншого. Це дозволяє створити більш іммерсивний та захоплюючий досвід для користувачів, що сприяє залученню та задоволенню клієнтів.

9. Підтримка офлайн-режиму: Однією з переваг мобільних додатків є їх здатність працювати в офлайн-режимі. Це означає, що користувачі можуть використовувати додаток без доступу до Інтернету. Наприклад, у разі повітряної тривоги, коли з'єднання може бути обмеженим, мобільний додаток "Карти укриттів" може надавати користувачам доступну інформацію про найближчі укриття та інструкції, навіть у відсутності мережі.

Всі ці переваги сприяють зростанню популярності мобільних додатків та переходу багатьох компаній до стратегії "APP-FIRST". Розробка мобільного додатку на React Native дозволяє швидко та ефективно використовувати ці переваги, надаючи клієнтам зручну, інноваційну та персоналізовану взаємодію.

1. ПОСТАНОВКА ЗАДАЧІ ТА ВИБІР ТЕХНОЛОГІЙ ДЛЯ СТВОРЕННЯ ДОДАТКУ

1.1 Постановка задачі

Мета проєкту:

Розробити мобільний додаток "Карти укриттів", який надасть користувачам інформацію про найближчі укриття під час повітряних тривог, а також допоможе їм знайти найбезпечніші місця для притулку.

Опис проєкту:

Мобільний додаток "Карти укриттів" буде призначений для забезпечення безпеки користувачів в разі повітряних тривог, спричинених різними чинниками , такими як російська агресія проти українців чи

Інші природні небезпеки , які можуть виникнути в інших країнах. Додаток надасть користувачам доступ до актуальних карт, які показують місця, де можна знайти укриття або безпечні зони під час таких подій.

Архітектура проєкту:

Для розробки додатку було використано архітектурний підхід, що базується на клієнт-серверній моделі.

Для розробки додатку було використано наступні технології:

1. *FrontEnd (APP-SIDE):*
    a. *Typescript*
    b. *React Native*
    c. *Expo-CLI*
    d. *Google Maps(integration for React Native)*
2. *BackEnd(Server):*
    a. *Typescript*
    b. *Node.js*
    c. *Express*
3. *Database*
    a. *MongoDB*

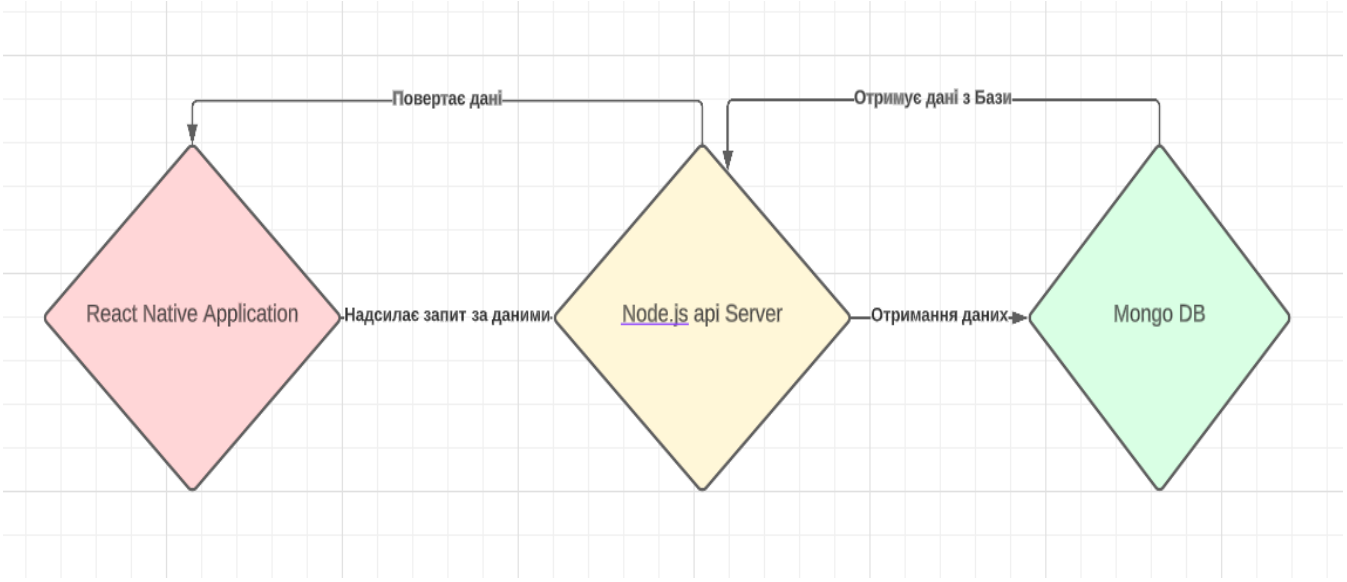

Рис 1.1 Архітектурна Структура Проєкту

Процес взаємодії мобільного додатку з сервером Node.js і базою даних MongoDB включає наступні етапи:

1. Клієнтська сторона: Мобільний додаток взаємодіє з інтерфейсом, генерує запити до сервера.
2. Відправка запиту: Запит відправляється з мобільного додатку на сервер за допомогою HTTP або HTTPS протоколу.
3. Прийом запиту на сервері: Сервер Node.js (наприклад, Express.js) отримує запит, обробляє його.
4. Взаємодія з MongoDB: Сервер Node.js взаємодіє з базою даних MongoDB через Mongoose або MongoDB драйвер, виконує потрібні операції.
5. Відправка відповіді: Сервер формує відповідь на запит, відправляє її назад до мобільного додатку.
6. Обробка відповіді: Мобільний додаток отримує відповідь, виконує дії залежно від статусу запиту та отриманих даних.

Цей процес дозволяє мобільному додатку взаємодіяти з сервером Node.js і базою даних MongoDB для отримання та збереження даних.

1.2 Аналіз ринку для додатку "Карти укриттів":

Для успішної розробки та впровадження мобільного додатку "Карти укриттів" необхідно провести аналіз ринку. Забезпечення безпеки користувачів під час повітряних тривог та пошук безпечних місць для притулку стають все більш актуальними питаннями.

У мобільному додатку "Карти укриттів" є прагненя задовольнити ринкову потребу у зручному та доступному способі отримання інформації про укриття та безпечні зони. Зростаюча популярність мобільних додатків, спрямованих на безпеку та надзвичайні ситуації, свідчить про високу зацікавленість користувачів у таких рішеннях.

Попередній аналіз конкурентного середовища вказує на наявність схожих додатків на ринку, проте у даного додатку є конкурентні переваги у вигляді унікального функціоналу та простоти використання. Оцінка потенційних конкурентів допоможе удосконалити наш продукт та розробити ефективну маркетингову стратегію для залучення користувачів.

Також важливим елементом аналізу ринку є вивчення цільової аудиторії та її потреб. Зусилля будуть направлені на привернення уваги та задоволення потреб потенційних користувачів, надаючи їм цінну інформацію та забезпечуючи безпеку під час надзвичайних ситуацій.

Загалом, аналіз ринку підтверджує актуальність та потенціал успіху додатку "Карти укриттів".

1.3 Опис використаних технологій та підходів

Для розробки проєкту було використано різний спектр технологій та програмних підходів. Основні з них такі:

a) Typescript - це розширення мови JavaScript, яке надає статичну типізацію та додаткові можливості для поліпшення розробки програмного забезпечення.
b) React Native - це фреймворк для розробки мобільних додатків, який дозволяє використовувати JavaScript та React для побудови нативних інтерфейсів користувача.
c) Expo-CLI - це набір інструментів, який спрощує розробку мобільних додатків з використанням React Native. Він надає зручність у налаштуванні та розгортанні додатків на різних платформах.
d) Node.js: Node.js - це середовище виконання JavaScript, яке дозволяє виконувати JavaScript на сервері
e) Express: Express - це легкий та гнучкий веб-фреймворк для Node.js, який дозволяє швидко створювати веб-додатки та API.
f) MongoDB - це система управління базами даних NoSQL, яка зберігає дані у документоорієнтованому форматі. Вона надає гнучкість у схемі даних та підтримує розширені можливості для роботи з даними в сучасних веб-додатках.

Застосування цих технологій у розробці додатку відкриває широкі можливості для створення ефективних та потужних мобільних додатків та серверних рішень.

1.3.1 TypeScript

TypeScript[1] - це розширення мови програмування JavaScript, яке надає статичну типізацію та додаткові можливості для поліпшення розробки програмного забезпечення. Він дозволяє розробникам визначати типи даних для змінних, параметрів функцій, повернених значень та інших елементів коду, що допомагає виявляти і усувати помилки на етапі компіляції.

TypeScript дозволяє створювати більш надійний та підтримуваний код. Він забезпечує покращену розуміння програмного забезпечення за допомогою розширеної інформації про типи, автодоповнення та перевірки на етапі розробки. Крім того, TypeScript підтримує сучасні функції ECMAScript та надає можливості для використання нових функціональностей JavaScript.

Одна з головних переваг TypeScript полягає в його сумісності з вже існуючим JavaScript-кодом. Розробники можуть поступово впроваджувати TypeScript у свої проекти та поступово розширювати його функціональність.

Завдяки інструментам, таким як компілятор TypeScript і редактори коду, TypeScript надає більшу надійність, покращує продуктивність розробників і забезпечує більшу безпеку в розробці програмного забезпечення.

1.3.2 React Native

React Native[2] є популярним фреймворком для розробки мобільних додатків, який базується на JavaScript та використовується для побудови нативних додатків для платформ Android та iOS. Його основною перевагою є можливість розробки додатків за допомогою одного коду, який може бути використаний на обох платформах.

Завдяки React Native розробники можуть використовувати знайомий синтаксис JavaScript та бібліотеку React для побудови високопродуктивних та нативно виглядаючих мобільних додатків. Він пропонує широкий спектр готових компонентів, які можна використовувати для швидкої розробки інтерфейсу користувача, а також дозволяє створювати власні компоненти для вирішення конкретних потреб проєкту.

Завдяки використанню нативних компонентів, додатки, розроблені на React Native, мають високий рівень продуктивності та швидкість роботи. Крім того, React Native підтримує гарячу перезавантаження, що дозволяє швидко оновлювати код додатку та бачити результати змін в реальному часі без повного перезапуску додатку.

React Native також має активну спільноту розробників, яка надає підтримку, документацію та набір сторонніх модулів і бібліотек для розширення можливостей фреймворку.

В цілому, React Native є потужним інструментом для розробки мобільних додатків, який поєднує продуктивність, ефективність та переносимість коду, дозволяючи розробникам створювати високоякісні додатки для різних платформ з використанням знайомих інструментів і технологій.

1.3.3 Expo-CLI

Expo-CLI[3] є набором інструментів для розробки мобільних додатків з використанням React Native. Він надає простий та зручний спосіб розробки, тестування та розгортання додатків для платформ Android, iOS та веб-браузерів.

Однією з основних переваг Expo-CLI є його можливість здійснювати швидку настройку та налагодження проєкту без необхідності встановлення та налаштування інших компонентів. Він має інтегроване середовище розробки, що дозволяє запускати додатки безпосередньо на фізичних пристроях або емуляторах.

Expo-CLI також надає доступ до різноманітних готових компонентів та бібліотек, що полегшують створення інтерфейсу користувача, роботу з камерою, геолокацією, віджетами та багато іншого. Крім того, він має вбудовану систему управління версіями та можливість автоматичного оновлення додатків без необхідності повторного розгортання.

Expo-CLI забезпечує простоту та швидкість розробки, спрощуючи процес створення мобільних додатків з використанням React Native. Його інтуїтивний інтерфейс та розширені можливості роблять його популярним вибором серед розробників, що працюють з React Native.

1.3.4 Node.Js

Node.js[4] є серверною платформою, побудованою на двигуні V8 JavaScript компанії Google, яка дозволяє виконувати JavaScript на стороні сервера. Його головна перевага полягає в тому, що він пропонує подієву та не-блокуючу модель програмування, що дозволяє обробляти багато запитів одночасно з використанням одного потока.

Ця модель програмування робить Node.js особливо ефективним для створення швидких та масштабованих додатків. Він володіє високою продуктивністю завдяки використанню асинхронного вводу/виводу та подієвого циклу, що дозволяє додаткам обробляти багато запитів одночасно без блокування інших операцій.

Node.js має велику кількість модулів, доступних через систему управління пакетами npm. Ці модулі допомагають в розробці різноманітних додатків, включаючи веб-сервери, API-сервери, додатки реального часу, мікросервіси та багато іншого. Також, Node.js забезпечує зручний спосіб створення власних модулів та пакетів, що сприяє повторному використанню коду та розширенню функціональності додатків.

Node.js використовується великою кількістю компаній та розробників для розробки широкого спектру додатків, від невеликих проєктів до великих систем. Він дозволяє розробникам ефективно працювати з JavaScript на сервері, забезпечуючи швидку реакцію на запити, високу масштабованість та розширені можливості розробки.

1.3.5 Express

Express[5] є популярним та розширюваним веб-фреймворком для розробки серверних додатків з використанням Node.js. Він надає простоту, гнучкість та ефективність у побудові веб-серверів та обробці запитів.

Одна з головних переваг Express полягає в його мінімалістичному підході, що дозволяє розробникам вільно визначати архітектуру та функціональність своїх додатків. Він не насильно накладає жорсткі правила або обмеження, що дає вам велику свободу вибору та контролю над процесом розробки.

Express також має багатий екосистему розширень та модулів, які дозволяють додатково розширювати його функціональність та працювати з іншими популярними інструментами. Наприклад, ви можете легко інтегрувати Express з базами даних, створювати API, робити аутентифікацію та авторизацію, робити роутинг, обробляти помилки та багато іншого.

Завдяки простоті та широкому спектру можливостей, Express використовується великою кількістю розробників та компаній для розробки різноманітних веб-додатків. Він дозволяє розробникам швидко створювати надійні та масштабовані сервери з великою продуктивністю та ефективністю.

1.3.6 MongoDB

MongoDB[6] є потужною та гнучкою нереляційною базою даних, яка використовує модель документів замість традиційних таблиць і рядків, що забезпечує більш просту та гнучку організацію даних. Вона стала популярною серед розробників завдяки своїй простоті використання та масштабованості.

Одна з основних переваг MongoDB - це гнучкість схеми даних. Ви можете зберігати документи різної структури в одній колекції, що дозволяє легко змінювати схему та додавати нові поля без необхідності змінювати всі існуючі документи. Це особливо корисно в розробці додатків, які вимагають гнучкості та еволюції структури даних з часом.

MongoDB також має підтримку розподіленого зберігання даних та горизонтального масштабування. Ви можете легко розподілити дані на кілька серверів та налаштувати реплікацію для забезпечення високої доступності та надійності даних. Благодаря цьому MongoDB підходить для обробки великого обсягу даних та високого навантаження.

MongoDB також пропонує мову запитів, відому як MongoDB Query Language (MQL), яка дозволяє виконувати потужні та складні запити до бази даних. Ви можете швидко здійснювати пошук, фільтрацію, сортування та агрегацію даних, використовуючи різноманітні операції та оператори.

Завдяки своїм перевагам та можливостям, MongoDB стала популярним вибором для розробників, які шукають нереляційну базу даних з гнучкими можливостями та хорошою масштабованістю.

2. АНАЛІЗ КОДОВОЇ РЕАЛІЗАЦІЇ ТА ОПИС

У React Native додатку “Карта укриттів” було використано компонентний підхід для створення інтерфейсу користувача. Цей підхід дозволяє розбити веб-додаток на невеликі і самодостатні компоненти, які можна повторно використовувати та легко управляти.

Кожен компонент відповідає за відображення певної частини інтерфейсу або виконання певної функціональності. Наприклад, можна мати компонент для відображення заголовка сторінки, компонент для відображення списку елементів або компонент для взаємодії з сервером.

Компоненти можуть бути вкладені один в одного, що дозволяє створювати складні ієрархії інтерфейсу. Наприклад, можливо мати компонент сторінки, який містить компоненти заголовка, списку елементів і форми вводу.

У даному проекті використано модульний підхід для розробки Backend застосунку. Це означає, що код розбито на невеликі, самодостатні модулі, кожен з яких відповідає за свою конкретну функціональність.

Кожен модуль виконує певну роботу і має чітко визначені обов'язки, що спрощує розробку та підтримку коду. Можна розробляти та тестувати кожен модуль незалежно, що дозволяє прискорити розвиток проекту. Крім того, модульна структура дозволяє перевикористовувати код, наприклад, можна використовувати модулі в інших проектах або в різних частинах нашого застосунку.

Завдяки модульному підходу отримано більш організований та масштабований код, що полегшує його розуміння та роботу з ним.

2.1 Кодова реалізація Frontend

2.1.1 Root component

```
import React from 'react';
import {createStackNavigator} from '@react-navigation/stack';
import {useColorScheme} from 'react-native';
import {useFonts} from 'expo-font';
import FontAwesome from '@expo/vector-icons/FontAwesome';
import Layout from "./pages";

const Stack :TypedNavigator<ParamListBase, ... = createStackNavigator();

no usages  b.malish *
export default function RootLayout() :JSX.Element | null {
    const [loaded :boolean , error :Error | null ] = useFonts( map: {
        SpaceMono: require('../assets/fonts/SpaceMono-Regular.ttf'),
        ...FontAwesome.font,
    });

    const colorScheme :ColorSchemeName = useColorScheme();

    if (error) {
        console.error('Error loading fonts', error);
        return null;
    }

    if (!loaded) {
        return null
    }

    return (
        <Layout/>
    );
}
```

Рис. 1 Root Component

Root Component , зображений на рис. 1 відповідає за налаштування початкового макету та відображення основного вмісту додатку. Він включає компонент Layout, який містить основний вміст додатку, такий як екрани, навігацію та стилі. Цей рут-компонент є важливою частиною архітектури додатку, визначаючи його початкову точку входу та структуру візуального представлення.

2.1.2 Роутизація та створення маршрутів для різних екранів

Код на рисунку 2 використовує бібліотеку react-navigation для створення навігаційного стеку в React Native додатку.

Функція Layout експортується як основний компонент і повертає навігаційний стек Stack.Navigator. У налаштуваннях screenOptions встановлена властивість headerShown на значення false, що означає, що заголовок екрану не буде відображатись на сторінках.

В компоненті Stack.Screen використовуються різні екрани, представлені компонентами WelcomeScreen, CitySelection, SheltersList, Map та ShelterDetails. Кожен з цих компонентів буде показаний на відповідному екрані в навігаційному стеці.

```
const Stack : TypedNavigator<ParamListBase, ...  = createStackNavigator();

2 usages
export default function Layout() : JSX.Element {
    return (
        <Stack.Navigator screenOptions={{headerShown: false}}>
            <Stack.Screen name="pages/Welcome" component={WelcomeScreen}/>
            <Stack.Screen name="pages/City" component={CitySelection}/>
            <Stack.Screen name="pages/SheltersList" component={SheltersList}/>
            <Stack.Screen name="pages/Map" component={Map}/>
            <Stack.Screen name="pages/ShelterDetails" component={ShelterDetails}/>
        </Stack.Navigator>
    );
}
```

Рис. 2 Роутизація та створення маршрутів

2.1.3 Welcome компонент

Рисунок 3 представляє компонент WelcomeScreen, який використовується для відображення екрану привітання. Він містить зображення логотипу, заголовок, підзаголовок та кнопку "Почати". При натисканні на кнопку виконується перехід на інший екран. Компонент також включає нижній колонтитул з посиланнями на політику конфіденційності та умови використання.

```
type WelcomeScreenNavigationProp = StackNavigationProp<RootStackParamList, 'Welcome'>;
2 usages  new *
export default function WelcomeScreen({navigation}: { navigation: WelcomeScreenNavigationProp }) {
    return (
        <View style={styles.container}>
            <View style={styles.content}>
                <Image source={require('../../assets/images/logo.png')} style={styles.logo}/>
                <Text style={styles.title}>Ласкаво просимо до Карти укриттів!</Text>
                <Text style={styles.subtitle}>Знайдіть укриття поблизу вас у разі надзвичайної ситуації.</Text>
                <TouchableOpacity style={styles.startButton} onPress={() => navigation.navigate('pages/City')}>
                    <Text style={styles.startButtonText}>Почати</Text>
                </TouchableOpacity>
            </View>
            <View style={styles.footer}>
                <Text style={styles.footerText}>Політика конфіденційності</Text>
                <Text style={styles.footerText}>Умови використання</Text>
            </View>
        </View>
    );
}
```

Рис. 3 компонент «Welcome Screen»

На рисунку 4 зображено об'єкт styles для визначення стилів компонента WelcomeScreen. Основні особливості деяких стилів для цього компонента наступні:

- container: Встановлює вигляд контейнера, який займає весь доступний простір на екрані. Він використовує гнучкість (flex) для розміщення елементів по центру і задає горизонтальні та вертикальні відступи.

- logo: Визначає стилі для зображення логотипу. Задає його розміри, режим зміни розмірів (resizeMode) та відступи.
- title та subtitle: Встановлюють стилі для заголовка та підзаголовка, такі як розмір шрифту, жирність тексту та вирівнювання.
- startButton: Визначає стилі для кнопки "Почати". Встановлює фоновий колір, горизонтальні та вертикальні відступи, радіус закруглення та колір тексту.

```
const styles :  = StyleSheet.create({
    container: {alignItems: 'center'...},
    logo: {resizeMode: 'contain'...},
    title: {fontWeight: 'bold'...},
    subtitle: {
        fontSize: 20,
        textAlign: 'center',
        marginTop: 20,
        marginBottom: 20,
    },
    startButton: {backgroundColor: '#3a83f1'...},
    startButtonText: {
        color: '#fff',
        fontSize: 18,
        fontWeight: 'bold',
    },
    footer: {flexDirection: 'row'...},
    footerText: {
        fontSize: 14,
        textDecorationLine: 'underline',
    },
    content: {
        alignItems:'center',
        flexGrow: 1,
        paddingBottom: 50, // Висота футера
    },
});
```

Рис. 4 Стилі для компоненту «Welcome Screen»

2.1.4 Компонент «Shelter List»

Код на рис. 5 та 6 представляє компонент SheltersList, який відповідає за відображення списку укриттів на екрані. Основні особливості цього коду наступні:

- Компонент отримує параметри navigation та route, які використовуються для навігації між екранами і передачі даних.
- За допомогою стану використовуються змінні citySelected, shelters та search, які відповідають відповідно за обраний місто, список укриттів та текст пошуку.
- Функція onSearch викликається при зміні значення тексту пошуку.
- За допомогою фільтрації filteredShelters застосовується пошук за адресою у списку укриттів.
- Функція handleShelterSelection викликається при виборі певного укриття, що спрацьовує навігацію до екрану ShelterDetails.
- За допомогою useEffect виконується завантаження списку укриттів за певним містом.
- На екрані відображаються заголовок, поле введення для пошуку, список укриттів, кнопка для перегляду на карті та навігаційна панель.

Цей компонент відповідає за відображення списку укриттів у певному місті, фільтрацію за адресою, можливість вибору конкретного укриття для перегляду деталей, перехід до екрану мапи для відображення на карті та навігацію до екрану вибору міста.

```
const SheltersList: React.FC<SheltersListScreenProps> = ({navigation : SheltersListScreenNavigationPr... , route : {params: {city: string}} }) => {
    const {city : string } = route.params;
    const [citySelected : tCity | null , setCitySelected : React.Dispatch<React.SetStateA... ] = useState<tCity | null>( initialState: null);
    const [shelters : tShelter[] , setShelters : React.Dispatch<React.SetStateA... ] = useState<Array<tShelter>>( initialState: []);
    const [search : string , setSearch : React.Dispatch<React.SetStateA... ] = useState( initialState: '');

    1 usage
    const onSearch = (text: string) : void  => {...};

    const filteredShelters : tShelter[]  = shelters?.filter((shelter : tShelter ) =>
        shelter.address.toLowerCase().includes(search.toLowerCase())
    );

    1 usage
    const handleShelterSelection = (shelterId: string) : void  => {...};

    useEffect( effect: () : void  => {...}, deps: [])
```

Рис. 5 Компонент «ShleterList»

```
    return (
        <View style={styles.container}>
            <Text style={styles.title}>Укриття в місті {citySelected?.name}</Text>
            <TextInput
                style={styles.searchInput}
                onChangeText={onSearch}
                value={search}
                placeholder="Пошук за адресою..."
            />
            <View style={styles.flatListContainer}>
                <FlatList
                    data={filteredShelters}
                    style={styles.listContainer}
                    renderItem={({item : tShelter }) => ...}
                    keyExtractor={(item : tShelter ) => item.id}
                />
            </View>
            <TouchableOpacity
                style={styles.mapButton}
                onPress={() => navigation.navigate('pages/Map', {cityId: citySelected?.id || ''})}
            ...>
            <View style={styles.navBar}...>
        </View>
    );
}
```

Рис. 6 Компонент «ShleterList»

2.2 Кодова реалізація Backend

2.2.1 Ініціалізація сервера

На рисунку 7 показана реалізація з використанням Express.js. Основні особливості цього коду наступні:

- Створюється екземпляр Express додатку за допомогою `const app = express();`.
- Використовується middleware `cors`, який дозволяє крос-доменні запити шляхом додавання заголовку `Access-Control-Allow-Origin: *` до відповідей сервера.
- Створюються роути для обробки запитів. Роут `/cities` обробляє GET запити та повертає список міст за допомогою `citiesHandler()`. Роут `/cities/:id/shelters/` повертає укриття для певного міста за допомогою `shelterListHandler(id)`.
- Якщо параметри запиту відсутні або не коректні, сервер відправляє відповідні статуси помилок і повідомлення.

```
import express from "express";
const cors = require('cors');
const app :Express  = express();
app.use(cors({
  origin: '*'
}));
const port :3000  = 3000;
import { citiesHandler, shelterListHandler, shelterInfoHandler } from "./db";
// Отримання списку укриттів для певного міста
app.get("/cities", (req: any, res: any) :void  => {
  const cities :tCity[]  = citiesHandler();
  res.json(cities);
});

app.get("/cities/:id/shelters/", (req: any, res: any) :void  => {
  const id = req.params.id;
  if (!id) {
    res.status(400).send("You forgot to pass id");
  }
  const shelterList :{city: tCity, shelterList: tSh...  = shelterListHandler(id);
  res.json(shelterList);
});

app.get("/sheltersInfo/:shelterId/", (req: any, res: any) :void  => {
  const { shelterId } = req.params;
  if (!shelterId) {
    res.status(400).send("You forgot to pass  shelterId");
  }
  const shelter :tShelterInfo  = shelterInfoHandler(shelterId);
  if(!shelter){
    res.status(400).send("Shelter not found");
    return;
  }
  res.json(shelter);
});

// Запуск сервера
app.listen(port, callback: () :void  => {
  console.log(`Shelter API is listening at http://localhost:${port}`);
});
```

Рис. 7 Ініціалізація Api Server

2.2.2 Використання handler

Handler - це функція або модуль, який виконує обробку певної операції або запиту. В даному випадку shelterListHandler є handler, який обробляє запит на отримання списку укриттів для певного міста. Він приймає ідентифікатор міста як вхідний параметр, знаходить відповідне місто та фільтрує список укриттів, щоб повернути результат знайденого міста та відповідних укриттів.

За допомогою методу find фільтрується список міст з використанням data.cities, щоб знайти місто з відповідним ідентифікатором.За допомогою методу filter фільтрується список укриттів з використанням data.shelters, щоб знайти усі укриття, що належать до вибраного міста (shelter.cityId === id).

Повертається об'єкт, що містить знайдене місто та список укриттів, як об'єкт типу { city: tCity; shelterList: Array<tShelter> }.

```
export const shelterListHandler = (
  id: string
): { city: tCity; shelterList: Array<tShelter> } => {
  const city = data.cities.find((elem: tCity) :boolean  => elem.id === id);
  const shelterList = data.shelters.filter(
    (shelter: tShelter) :boolean  => shelter.cityId === id
  );
  return { city, shelterList };
};
```

Рис. 8 Приклад функції-хендлера

3. РЕЗУЛЬТАТ РОБОТИ ТА ОГЛЯД ІНТЕРФЕЙСУ

3.1 Початковий екран

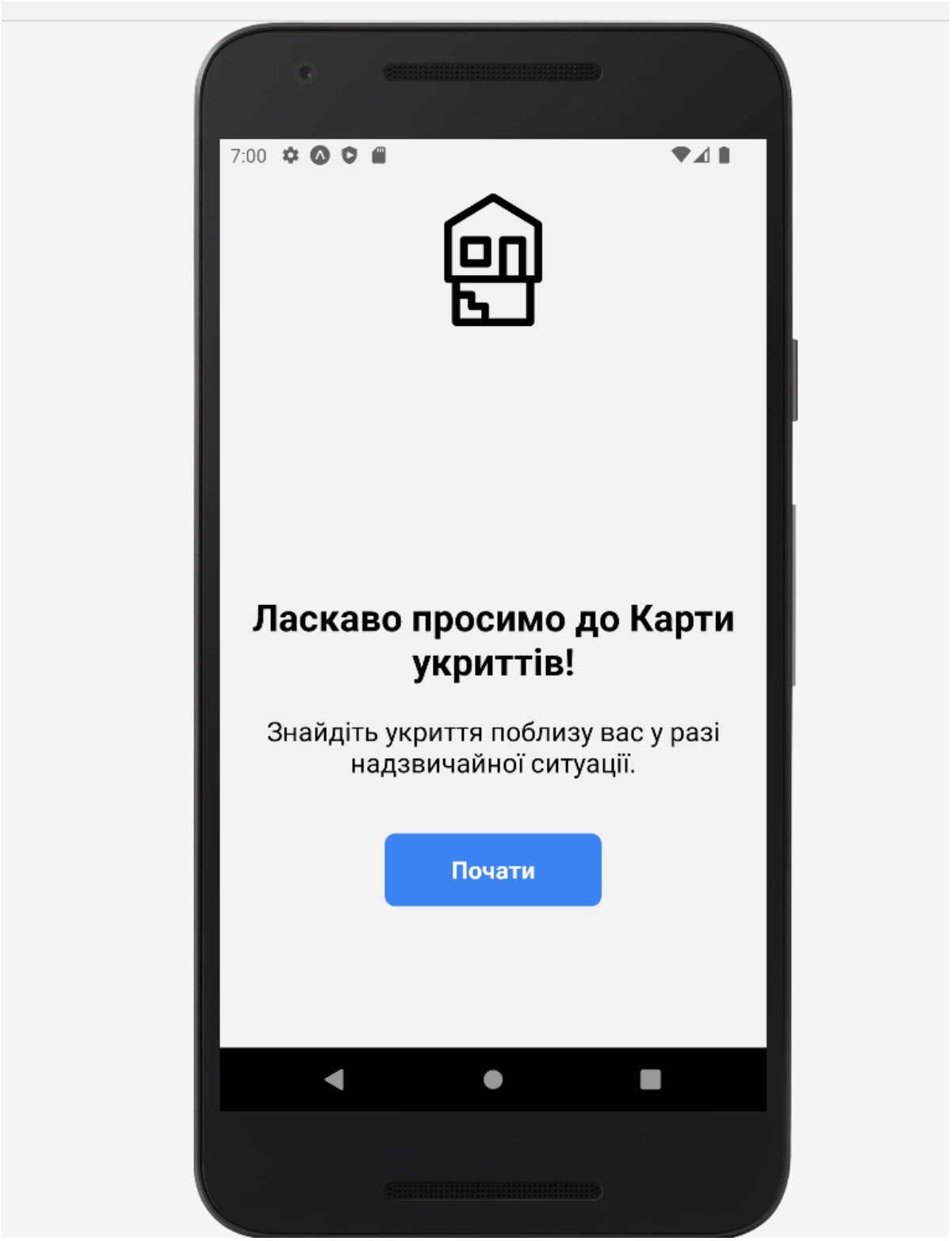

Рис. 9 Початковий екран

3.2 Екран вибору Міста

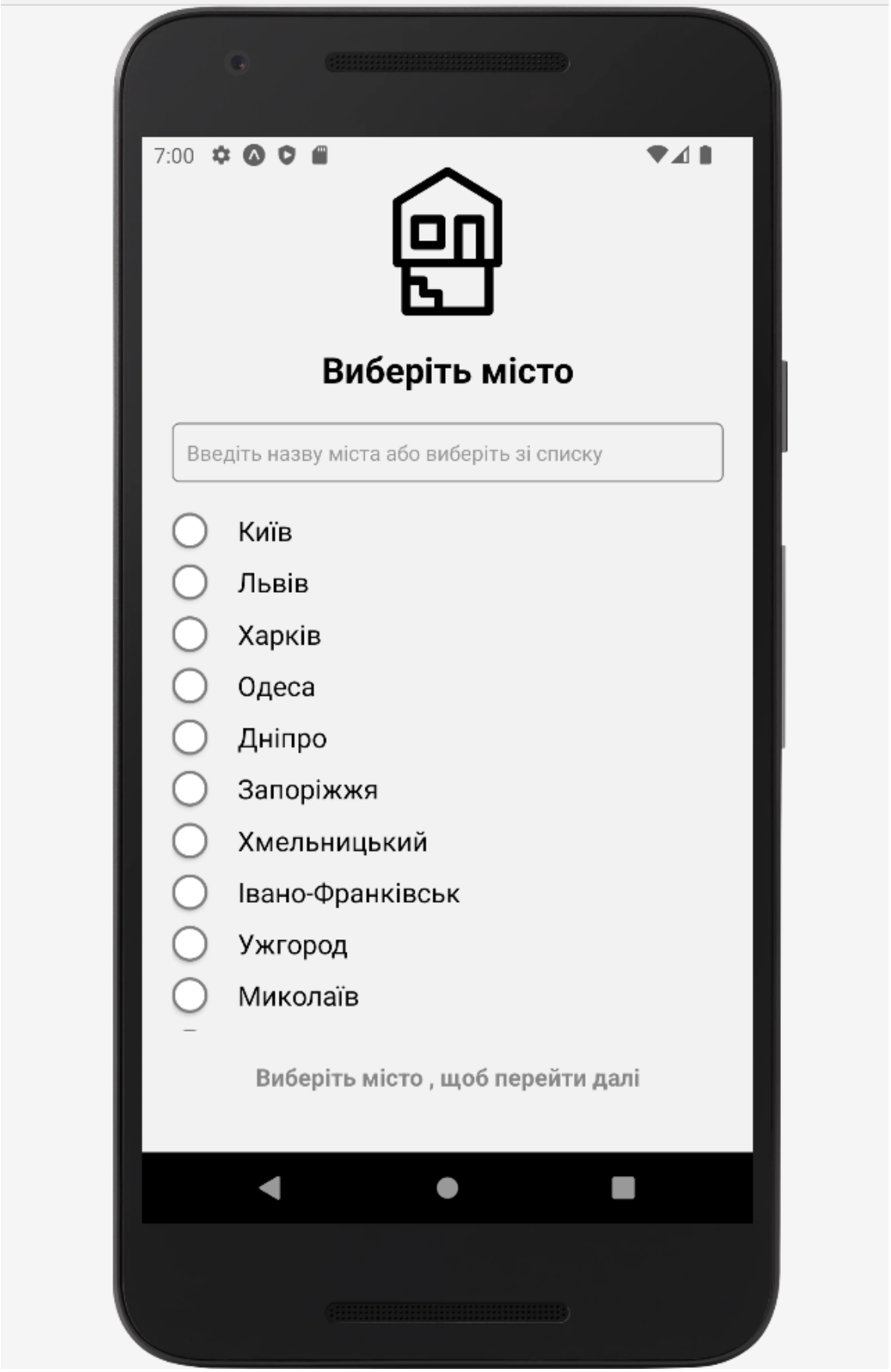

Рис. 10 Початковий екран вибору міста

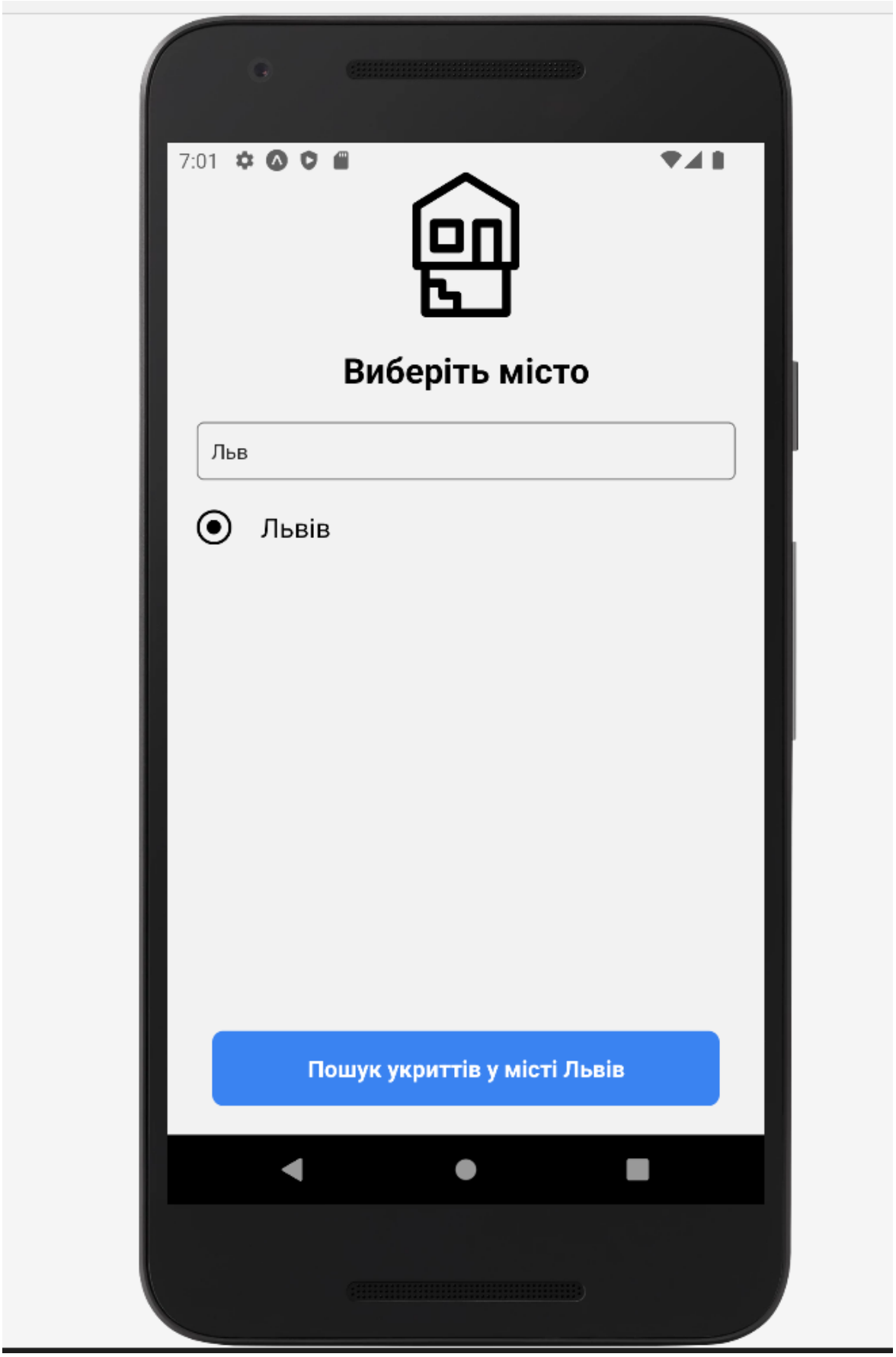

Рис. 11 Відфільтровані міста

3.3 Екран списку укриттів

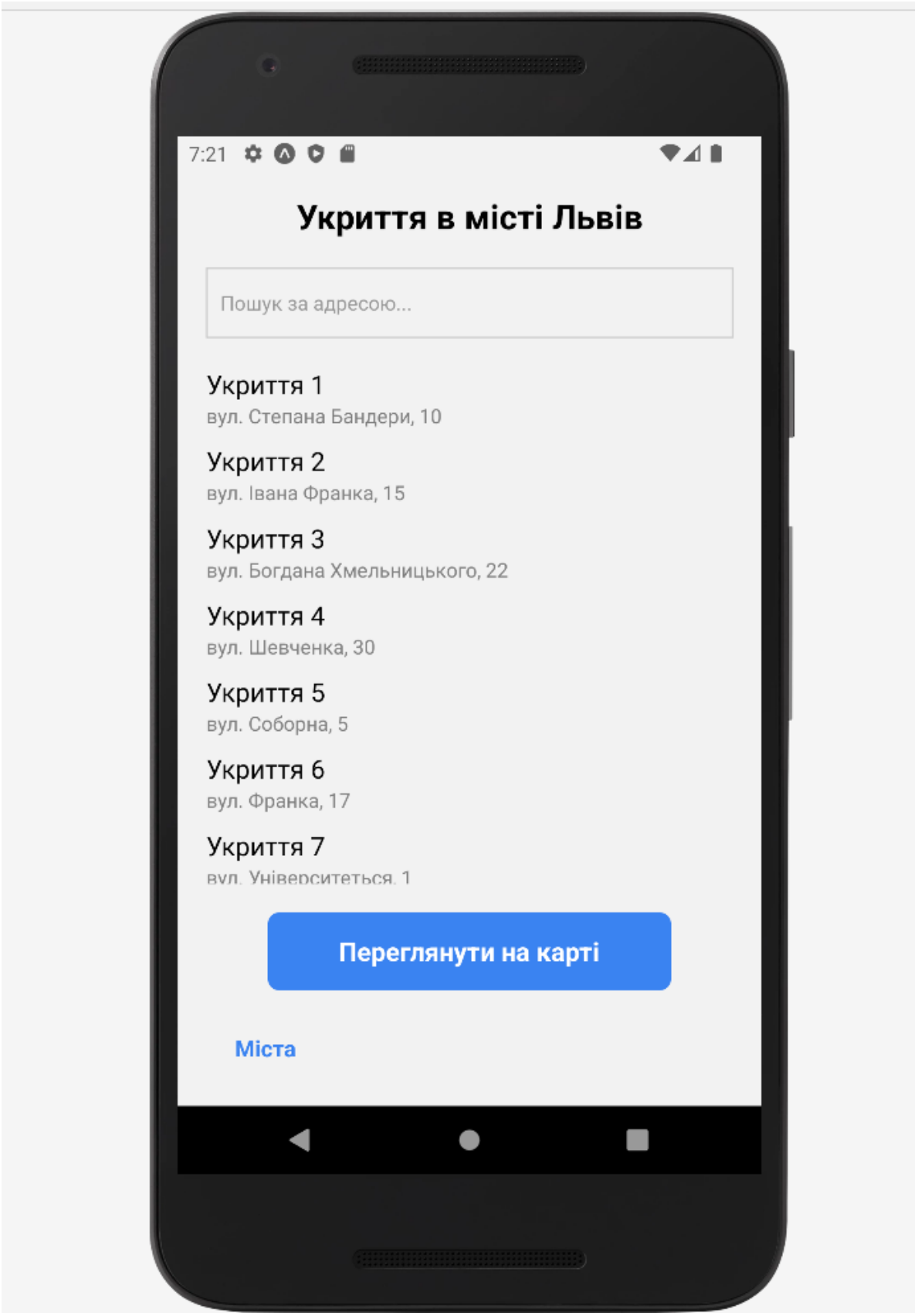

Рис. 12 Початковий вигляд списку укриттів

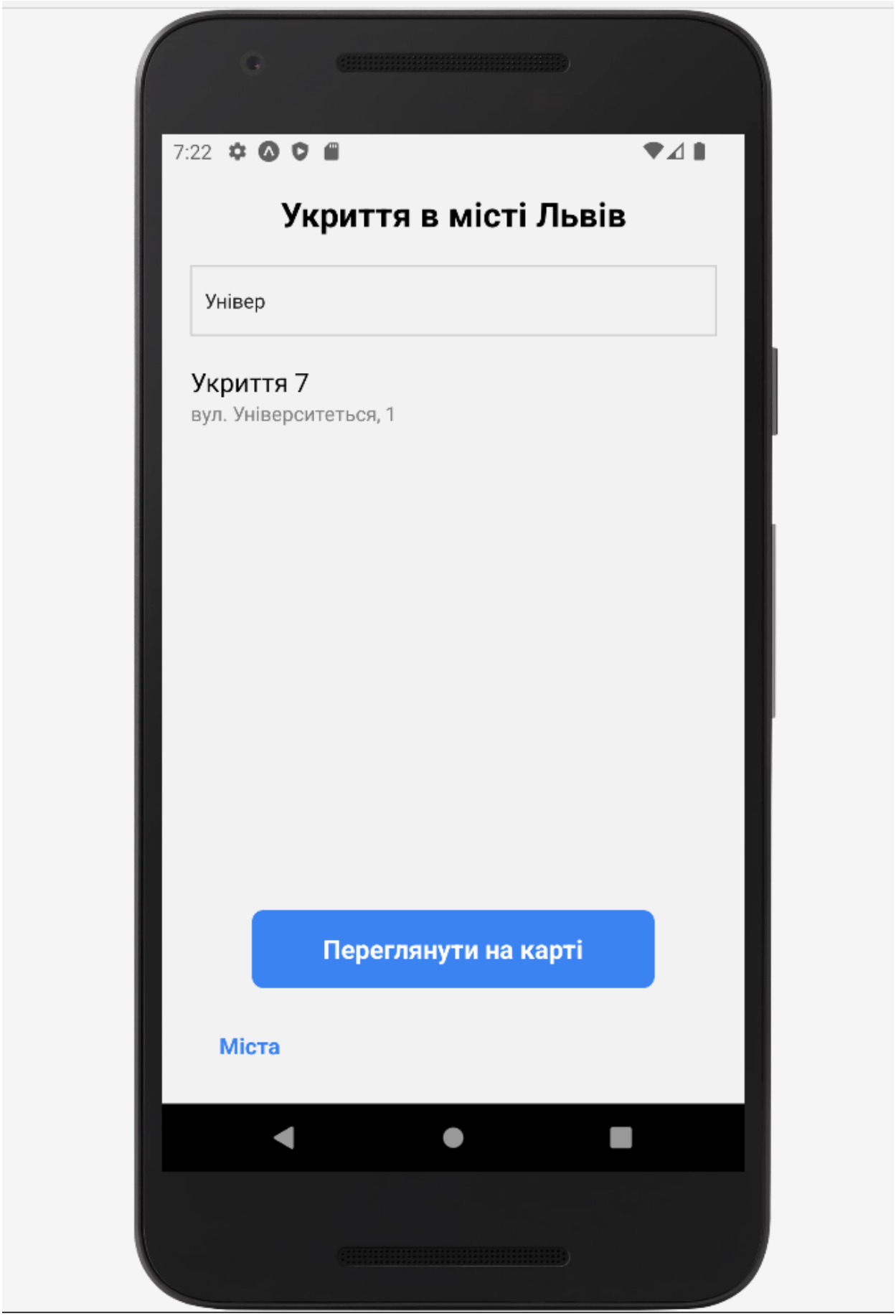

Рис. 13 Відфільтрований вигляд списку укриттів

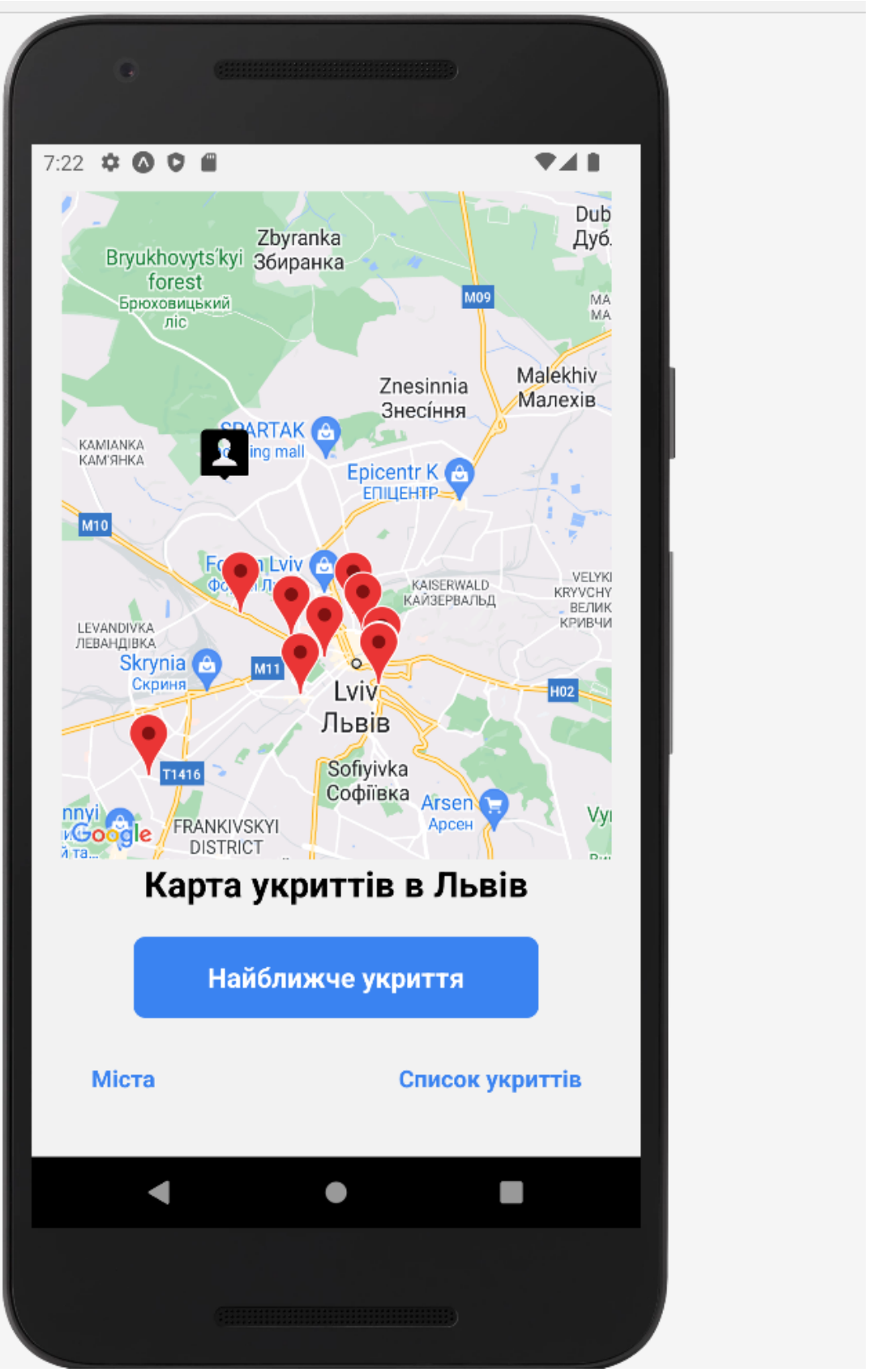

Рис. 14 Вигляд списку укриттів у форматі мапи

3.4 Екран детальної інформації про укриття

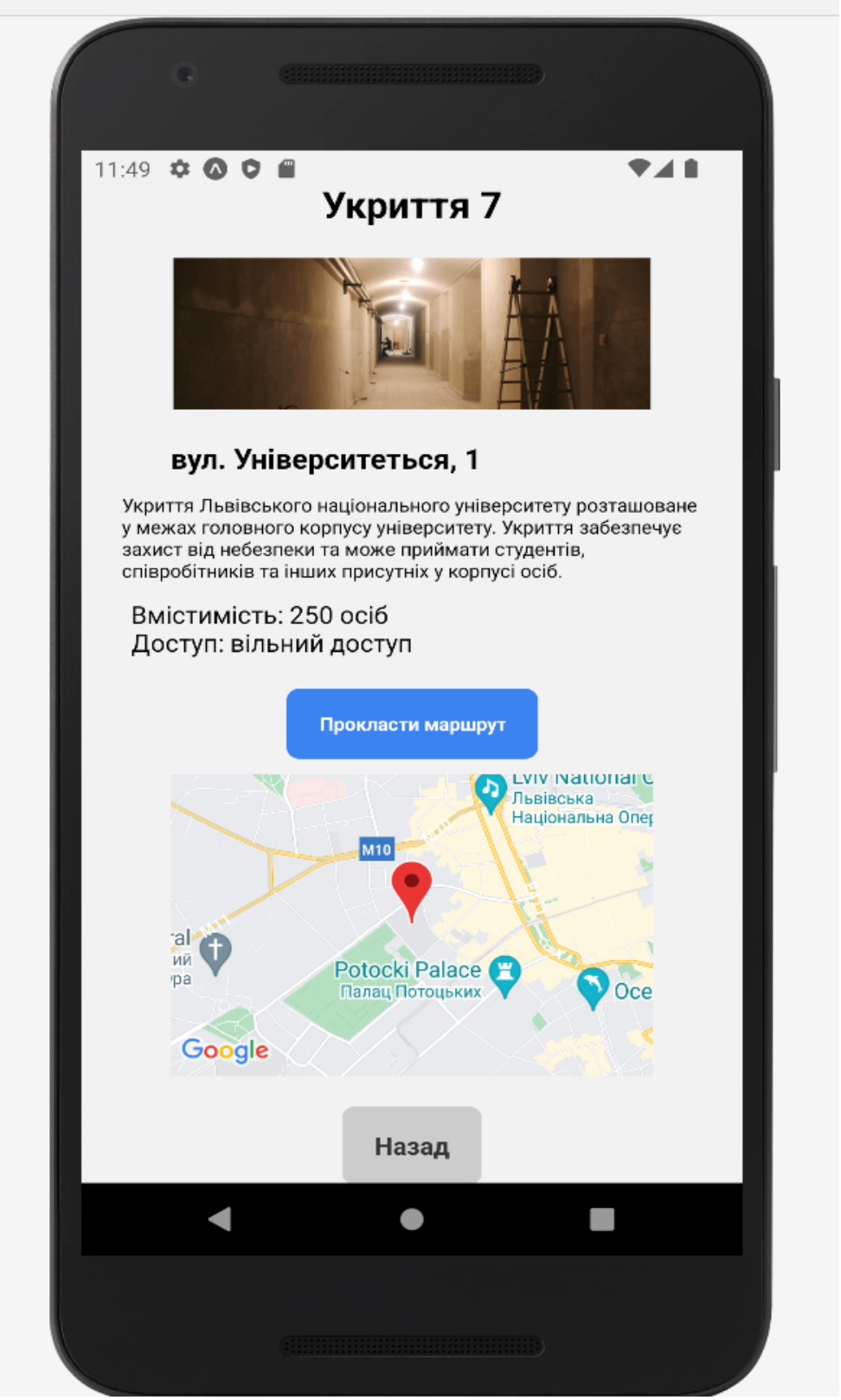

Рис. 15 Вигляд сторінки деталей укриття

ВИСНОВКИ

В рамках даного проєкту було розроблено мобільний додаток "Карти укриттів", який має на меті надати користувачам інформацію про найближчі укриття та допомогти їм знайти безпечні місця для притулку під час повітряних тривог. Додаток використовує архітектурний підхід, базуючись на клієнт-серверній моделі.

FrontEnd частина додатку реалізована з використанням Typescript та фреймворку React Native, що дозволяє писати мобільні додатки для платформ Android та iOS з використанням однієї кодової бази. Також було використано Expo-CLI для спрощення процесу розробки та інтеграцію з Google Maps API для відображення карт.

BackEnd частина додатку побудована на базі Node.js, використовуючи мову програмування Typescript. Для створення серверної частини використано фреймворк Express, який надає зручні інструменти для створення веб-додатків. Для збереження даних використовується MongoDB, що забезпечує ефективне та масштабоване збереження інформації.

Використання таких технологій, як Typescript, React Native, Expo-CLI та Google Maps API, надає потужні інструменти для розробки функціонального та зручного мобільного додатка. Цей стак дозволяє створити нормальний додаток зі зручним інтерфейсом, мобільними можливостями та інтеграцією з картами.

Загалом, проєкт "Карти укриттів" є реалізацією мобільного додатку, який сприяє забезпеченню безпеки користувачів та надає зручний доступ до необхідної інформації. Його використання може бути цінним в різних ситуаціях, пов'язаних з повітряними тривогами та пошуком безпечних місць для притулку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Офіційна документація TypeScript: https://www.typescriptlang.org/docs/
2. Офіційна документація React Native: https://reactnative.dev/docs/
3. Офіційна документація Expo-CLI: https://docs.expo.io/
4. Офіційна документація Google Maps API для React Native: https://developers.google.com/maps/documentation/react-native/overview
5. Офіційна документація Node.js: https://nodejs.org/en/docs/
6. Офіційна документація Express: https://expressjs.com/
7. Офіційна документація MongoDB: https://docs.mongodb.com/
8. Додаткові джерела інформації, такі як статті, блоги та форуми, пов'язані з використанням цих технологій у розробці мобільних додатків.