НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«До захисту допущено»
Завідувач кафедри
_____ Євгенія СУЛЕМА
«___» _____ 2023 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інженерія програмного забезпечення мультимедійних та інформаційно-пошукових систем»

спеціальності 121 Інженерія програмного забезпечення

на тему: «Мобільний застосунок для знаходження найближчого укриття»

Виконала:
студентка IV курсу, групи КП-92
Вергун Марія Олександрівна _____

Керівник:
Ст. викладач кафедри ПЗКС, к.т.н.,
Сулема Ольга Костянтинівна _____

Консультант з нормоконтролю:
Доцент кафедри ПЗКС, к.т.н., доцент,
Онай Микола Володимирович _____

Рецензент:
Асистент кафедри СПіСКС,
Радченко Костянтин Олександрович _____

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

Київ – 2023 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет прикладної математики
Кафедра програмного забезпечення комп'ютерних систем

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення мультимедійних та інформаційно-пошукових систем»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Євгенія СУЛЕМА
«___»_____2022 р.

ЗАВДАННЯ

на дипломний проєкт студенту

Вергун Марії Олександрівні

1. Тема проєкту «Мобільний застосунок для знаходження найближчого укриття», керівник проекту Сулема Ольга Костянтинівна, к.т.н., ст. викладач, затверджені наказом по університету від «31» травня 2023 р. № 2107-с

2. Термін подання студентом проєкту «16» червня 2023 р.

3. Вихідні дані до проєкту: див. Технічне завдання.

4. Зміст пояснювальної записки:
   - аналіз мов програмування та технологій розроблення мобільних застосунків;
   - розроблення мобільного застосунку для знаходження найближчого укриття;
   - опис розроблених алгоритмів та підпрограм;
   - аналіз розробленого застосунку.

5. Перелік обов'язкового графічного матеріалу:
   - алгоритм пошуку (креслення);
   - алгоритм фільтрації (креслення);
   - структура бази даних (креслення);

- архітектура застосунку (плакат);
- діаграма прецедентів (плакат).

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Онай М.В., доцент		

7. Дата видачі завдання «31» жовтня 2022 р.

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	Вивчення літератури за тематикою проєкту	13.11.2022	
2.	Розроблення та узгодження технічного завдання	25.11.2022	
3.	Розроблення структури мобільного застосунку	15.12.2022	
4.	Підготовка матеріалів першого розділу дипломного проєкту	30.12.2022	
5.	Розроблення дизайну сторінок та графічних елементів	03.02.2023	
6.	Підготовка матеріалів другого розділу дипломного проєкту	19.02.2023	
7.	Програмна реалізація мобільного застосунку	10.03.2023	
8.	Тестування мобільного застосунку	17.03.2023	
9.	Підготовка матеріалів третього розділу дипломного проєкту	30.03.2023	
10.	Підготовка матеріалів четвертого розділу дипломного проєкту	12.04.2023	
11.	Підготовка графічної частини дипломного проєкту	21.04.2023	
12.	Оформлення документації дипломного проєкту	26.05.2023	

Студент Марія ВЕРГУН

Керівник проєкту Ольга СУЛЕМА

АНОТАЦІЯ

Даний дипломний проєкт присвячений створенню мобільного застосунку для автоматичного знаходження найближчого укриття цивільного населення м. Києва відносно місця розташування користувача застосунку.

У даній роботі було виконано аналіз наявних рішень – безпекових застосунків з аналогічним призначенням. Завдяки цьому було виділено основні функціональні та нефункціональні вимоги системи. Також було зроблено глибокий аналіз методів та алгоритмів, які дозволяють задовольнити поставлені вимоги до системи. В результаті було обрано ефективні та швидкі алгоритми для знаходження найближчих географічних координат до наявних та побудови маршрутів. В результаті проведених аналізів було розроблено систему, що представляє користувачеві легкий мобільний застосунок, який покриває основні потреби в знаходженні укриття, маршруту до нього. Розробка проводилася із використанням кросплатформенних інструментів, тому дозволяє використання широкому спектру користувачів.

В даному проекті розроблено та досліджено: архітектуру серверної та клієнтської частини додатку, алгоритм автоматичного пошуку найближчого укриття, алгоритми фільтрації інформації, а також графічні елементи та дизайн вікон.

ABSTRACT

This diploma project is devoted to the creation of a mobile application for automatically finding the nearest shelter of the civilian population of Kyiv relative to the location of the application user.

In this work, an analysis of existing solutions – security applications with a similar purpose – was performed. Thanks to this, the main functional and non-functional requirements of the system were highlighted. An in-depth analysis of the methods and algorithms that allow meeting the set requirements for the system was also made. As a result, effective and fast algorithms were chosen for finding the closest geographic coordinates to the existing ones and building routes. As a result of the conducted analyses, a system that presents the user with an easy mobile application that covers the basic needs for finding a shelter and a route to it was developed. The development was carried out with the use of cross-platform tools, therefore it allows the use of a wide range of users.

This project developed and researched: the architecture of the server and client part of the application, the algorithm for automatic search of the nearest shelter, information filtering algorithms, as well as graphic elements and window design.

ДП.045440-01-90 Мобільний застосунок для знаходження найближчого укриття. Відомість проєкту

Позначення	Найменування	Кіл-ть	Примітка
	Документація проєкту		
ДП.045440-02-91	Мобільний застосунок	5	
	для знаходження		
	найближчого укриття.		
	Технічне завдання		
ДП.045440-03-81	Мобільний застосунок	85	
	для знаходження		
	найближчого укриття.		
	Пояснювальна записка		
ДП.045440-04-51	Мобільний застосунок	4	
	для знаходження		
	найближчого укриття.		
	Програма та методика		
	тестування		
ДП.045440-05-34	Мобільний застосунок	17	
	для знаходження		
	найближчого укриття.		
	Керівництво користувача		
ДП.045440-06-99	Мобільний застосунок	1	
	для знаходження		
	найближчого укриття.		
	Алгоритм пошуку		
	найближчого укриття.		
	Блок-схема алгоритму		

Позначення	Найменування	Кіл-ть	Примітка
ДП.045440-07-99	Мобільний застосунок	1	
	для знаходження		
	найближчого укриття.		
	Структура бази даних.		
	ER-діаграма		
ДП.045440-08-99	Мобільний застосунок	1	
	для знаходження		
	найближчого укриття.		
	Алгоритм фільтрації.		
	Блок-схема алгоритму		
ДП.045440-09-98	Мобільний застосунок	1	
	для знаходження		
	найближчого укриття.		
	Компакт-диск		

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

„Київський політехнічний інститут імені Ігоря Сікорського”

Факультет прикладної математики

“ЗАТВЕРДЖЕНО”

Завідувач кафедри

_____ Євгенія СУЛЕМА

“___” _____ 2022 р.

МОБІЛЬНИЙ ЗАСТОСУНОК ДЛЯ ЗНАХОДЖЕННЯ НАЙБЛИЖЧОГО УКРИТТЯ

Технічне завдання

ДП.045480-02-91

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Ольга СУЛЕМА

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____Марія ВЕРГУН

2022

ЗМІСТ

1. Найменування та галузь застосування 3
2. Підстава для розроблення 3
3. Призначення розробки 3
4. Вимоги до програмного продукту 3
5. Вимоги до проєктної документації 4
6. Етапи проєктування 5
7. Порядок тестування розробки 5

1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: Мобільний застосунок для побудови маршруту до найближчого укриття відповідно до місцезнаходження користувача.

Галузь застосування: інформаційні технології.

2. ПІДСТАВА ДЛЯ РОЗРОБЛЕННЯ

Підставою для розроблення є завдання на дипломне проєктування, затверджене кафедрою програмного забезпечення комп'ютерних систем Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського» (КПІ ім. Ігоря Сікорського).

3. ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для пошуку місць найближчих укриттів цивільного населення з автоматичним пошуком найближчого за розташуванням, а також з можливістю фільтрації за типом укриттів та їхнім призначенням.

4. ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ

Мобільний застосунок має забезпечувати таку функціональність:

1) Пошук найближчого укриття із зазначенням відстані до нього.
2) Розширений пошук за відстанню з можливістю обрати альтернативні варіанти.
3) Фільтрація за типом укриттів та їхнім призначенням.
4) Побудова маршруту до обраного укриття на мапі.
5) Перегляд інформаційної довідки та визначення умовних позначень.

Додаткові функції мають включати:

1) Автентифікація та авторизація.
2) Пошук укриття за адресою.
3) Додавання обраних укриттів.
4) Збереження історії місцезнаходжень.
5) Можливість додати власне укриття, не помічене на мапі (через модерацію адміністратора).
6) Можливість подати інформацію про недостовірність знаходження укриття (через модерацію адміністратора).

Серверну частину системи виконати мовою програмування Python, мобільний застосунок – за допомогою бібліотеки React Native.

5. ВИМОГИ ДО ПРОЄКТНОЇ ДОКУМЕНТАЦІЇ

У процесі виконання проєкту повинна бути розроблена наступна документація:

1. Пояснювальна записка.
2. Програма та методика тестування.
3. Керівництво користувача.
4. Креслення:
   4.1. «Структура бази даних. ER-діаграма».
   4.2. «Алгоритм пошуку найближчого укриття. Блок-схема алгоритму».
   4.3. «Алгоритм фільтрації. Блок-схема алгоритму».

6. ЕТАПИ ПРОЄКТУВАННЯ

Вивчення літератури за тематикою роботи…….. 30.10.2022
Розроблення та узгодження технічного завдання…… 08.11.2022
Розроблення структури застосунку……... 10.01.2023
Розроблення дизайну вікон та графічних елементів……05.03.2023
Програмна реалізація серверної частини…….. 19.03.2023
Програмна реалізація клієнтської частини……... 02.04.2023
Тестування системи…… 16.04.2023
Підготовка матеріалів текстової частини проєкту…….. 30.04.2023
Підготовка матеріалів графічної частини проєкту…….. 14.05.2023
Оформлення технічної документації проекту……. 28.05.2023

7. ПОРЯДОК ТЕСТУВАННЯ РОЗРОБКИ

Тестування розробленого програмного продукту виконується відповідно до “Програми та методики тестування”.

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

"ЗАТВЕРДЖЕНО"

Завідувач кафедри

_____Євгенія СУЛЕМА

"___" _____ 2023 р.

МОБІЛЬНИЙ ЗАСТОСУНОК ДЛЯ ЗНАХОДЖЕННЯ НАЙБЛИЖЧОГО УКРИТТЯ

Пояснювальна записка

ДП.045440-03-81

"ПОГОДЖЕНО"

Керівник проєкту:

_____ Ольга СУЛЕМА

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Марія ВЕРГУН

2023

ЗМІСТ

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ 3
ВСТУП 5
1. АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ. ПОСТАНОВКА ЗАВДАННЯ 7
1.1. Проблематика та актуальність 7
1.2. Огляд існуючих рішень 7
1.3. Порівняння наявних рішень 12
1.4. Постановка задачі 14
1.5. Висновки до розділу 18
2. ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 19
2.1. Алгоритмічне підґрунтя 19
2.2. Огляд програмних засобів розробки 27
2.3. Висновки до розділу 35
3. ОПИС РЕАЛІЗАЦІЇ ПРОГРАМНОГО ЗАСОБУ 37
3.1. Архітектура програмного забезпечення 37
3.2. Структура бази даних 39
3.3. Опис модулів програмного забезпечення 42
3.4. Організація взаємодії модулів програмного забезпечення 50
3.5. Висновки до розділу 52
4. АНАЛІЗ РОЗРОБЛЕНИХ ПРОГРАМНИХ ЗАСОБІВ 53
4.1. Опис інтерфейсу застосунку 53
4.2. Тестування програмного забезпечення 66
4.3. Рекомендації щодо подальшого вдосконалення 79
4.4. Висновки до розділу 80
ВИСНОВКИ 81
СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ 83
ДОДАТКИ 85

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

ПЗ – програмне забезпечення.

БД – база даних.

СУБД – система управління базою даних.

SQL – Structured Query Language. Декларативна мова для взаємодії з реляційними базами даних.

UML – Unified Modeling Language. Уніфікована мова для моделювання, що використовується для графічного зображення елементів розробки ПЗ.

MoSCoW – метод оцінки вимог системи. Назва є абревіатурою елементів шкали: Must, Should, Could, Won’t have.

API – Application Programming Interface. Інтерфейс для взаємодії з системою.

REST – Representational State Transfer. Архітектурний стиль, підхід до моделювання API з певними обмеженнями.

HTTP – Hypertext Transfer Protocol. Мережевий протокол передачі даних.

DOM – Document Object Model. Об’єктна модель документа, що дозволяє роботу зі структурованими документами, наприклад JSON, XML, тощо.

JSON – JavaScript Object Notation. Структурована модель для передачі інформації.

MVC – Model, View, Controller. Парадигма розробки ПЗ.

CI/CD – Continuous Integration / Continuous Deployment. Безперервна інтеграція та безперервна розробка. Сучасний підхід до розробки програмного забезпечення.

ВСТУП

Життя людини є різноманітним та вражаючим. Існує велика кількість можливостей для власного розвитку та розвитку суспільства, наприклад досі є відкритою і не дослідженою космічна галузь, також є простір для нових технологій та медичних відкриттів. Сучасним людям під силу робити неможливі і шокуючі речі. І головним фактором існування цих можливостей є життя та безпека.

В умовах існування небезпеки людський мозок не спроможний працювати над складними задачами і фокусує свою увагу на єдиному – самозбереженні себе та своїх рідних. Саме тому, вважаючи на перманентну небезпеку, що існує передусім в українському суспільстві, необхідне ефективне використання сучасних технологій для допомоги людям у складній стресовій ситуації. В такій ситуації, в якій зволікання на декілька хвилин може бути ключовим у можливості забезпечення безпеки людини.

Одним з найпростіших прикладів місць, які можуть вберегти людину від небезпеки, вже кілька десятиліть є укриття цивільного населення – в більшості підземні споруди, які можуть забезпечити відносно спокійне перебування людей у момент прямої загрози. Такі укриття мають бути розташовані відповідно до кількості людей, що проживає в тому чи іншому районі, та працювати цілодобово. Тоді у кожного громадянина є можливість скористатися укриттям та самостійно прокласти маршрут до нього. Проте не кожна людина, а особливо під дією стресового фактору, здатна виважено обрати найближчий варіант укриття. До того ж враховуючи те, що здебільшого інформації про розташування укриттів досі недостатньо, то таке завдання може бути зовсім непосильним.

Задля подолання цих проблем пропонується створення інформаційно-безпекової системи. Така система має використовувати ефективні алгоритми пошуку та побудови найкращого шляху відповідно до місця розташування користувача, а також надавати інформацію про всі наявні

укриття з можливістю зручної фільтрації. Автоматизація пошуку сприятиме швидкому вибору укриття та значно скорочуватиме час на прокладання шляху до нього.

Даний дипломний проєкт присвячено розробці безпекової системи у вигляді мобільного застосунку, який може ефективно вирішувати поставлені задачі знаходження найкращого варіанту укриття користувачу.

1. АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ. ПОСТАНОВКА ЗАВДАННЯ

1.1. Проблематика та актуальність

Зважаючи на реалії, в яких жителі України живуть останній рік, різко виникла гостра потреба в наявності укриттів цивільного населення. Доступ до них має бути якомога швидшим і кожен, незалежно від свого місця розташування, має мати змогу знайти сховище за мінімально можливий час. В ідеалі цей час не перевищує 10 хвилин.

Проте, ці вимоги часто є недосяжними через ряд причин:

- мала кількість укриттів чи їх відсутність;
- неможливість знайти інформацію про розташування укриттів;
- стан приміщення, що не відповідає вимогам укриття;
- неможливість доступу до укриття через фізичні обмеження.

Більшість з цих проблем можна було б вирішити належним інформуванням суспільства про технічний стан та розташування укриття. Наприклад, створенням мапи укриттів того чи іншого міста чи містечка. Беручи до уваги м. Київ, така мапа наявна на офіційному порталі міста [1].

Проте навіть наявність такої мапи не гарантує того, що людина зможе в критичній ситуації зорієнтуватися та прокласти маршрут до найближчого укриття. Через це доцільно створити такий додаток, який би міг автоматично підібрати найоптимальніший варіант сховища в залежності від поточного місця розташування користувача. Це значно б спростило задачу пошуку та зменшило б час на переміщення у безпечне місце. Тим самим цей додаток став би в нагоді великій кількості користувачів, що забезпечує їм іноді життєво необхідні хвилини на порятунок.

1.2. Огляд існуючих рішень

1.2.1. *Застосунок «ДеУкриття»*

Застосунок розроблений для жителів і гостей міста Тернополя. Інтерфейс застосунку являє собою мапу з нанесеними на неї мітками –

розташуваннями укриттів по місту. Головний екран застосунку зображений на рис. 1.1.

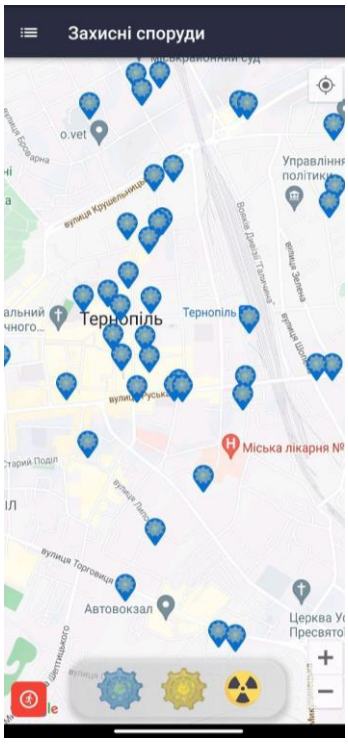


Рис. 1.1. Головний екран застосунку “ДеУкриття”

У застосунку наявна фільтрація за типом укриття:

- найпростіше укриття;
- сховище;
- укриття на зупинці;
- протирадіаційне укриття.

Застосунок дозволяє перегляд інформації про кожне з укриттів: його адресу, наявність Wi-Fi, примітки, тощо. При оголошенні повітряної тривоги застосунок надсилає сповіщення користувачу про це. Також він виконує основну функцію побудови маршруту до найближчого з укриттів через перенаправлення користувача на сторонній сервіс від Google Maps. Для побудови маршруту необхідно натиснути на червону кнопку у лівому нижньому куті екрану, після чого користувач автоматично

перенаправляється на сервіс Google Maps з побудованим маршрутом до найближчого укриття.

Переваги рішення:

- наявність фільтрації за типом;
- лаконічність дизайну;
- інтеграція з місцевими органами, що оголошують тривогу;
- наявність автоматичного вибору найближчого укриття.

Недоліки рішення:

- мала інформативність – обмежена і неповна інформація про кожне укриття;
- незручний інтерфейс вибору найближчого укриття – кнопка є майже непомітною.

1.2.2. *Застосунок «Прилуки незламні»*

Застосунок надає інформацію про місцезнаходження захисних споруд міста Прилуки та Прилуцького району Чернігівської області. Інтерфейс застосунку є доволі лаконічним.

На головній сторінці доступні три опції:

- Карта захисних споруд.
- Пункти Незламності.
- Резервні номери телефонів.

При виборі користувачем карти захисних споруд йому стає доступним інтерактивна мапа з нанесеними на неї укриттями району.

Про кожне укриття доступний перегляд його адреси та опису. Місцезнаходження користувача відображається міткою, відмінною від мітки укриття – мітка синього кольору. Мапа укриттів застосунку зображена на рис. 1.2.

При натисканні користувача на маркер укриття відкривається розширена інформація про його адресу та назву. Більш детального опису укриттів не передбачено.

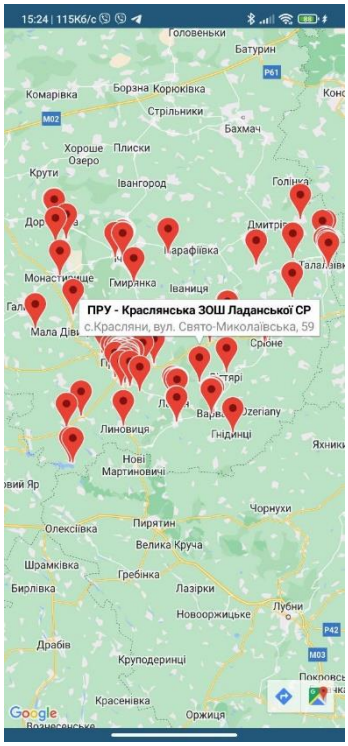


Рис. 1.2. Головний екран застосунку «Прилуки незламні»

Застосунок надає функціонал побудови маршруту через сторонній сервіс Google Maps,. Вибору найближчого укриття відповідно до розташування користувача не реалізовано – користувач обирає укриття самостійно шляхом натискання на маркер укриття.

Крім основної функціональності, застосунок надає інформацію про наявні Пункти Незламності та резервні номери телефонів екстрених служб.

Переваги рішення:

- наявність розширених функцій, а саме номерів телефонів екстрених служб;
- лаконічність дизайну.

Недоліки рішення:

- відсутня детальна інформація про укриття;
- відсутність автоматичного підбору найближчого укриття.

1.2.3. *Додаток «Київ Цифровий»*

«Київ Цифровий» – це мультифункціональний додаток для гостей і жителів міста Києва. Мапа укриттів не є його основним функціоналом, проте ця опція міститься у розділі «Безпека». При виборі даної опції на пристрій користувача завантажується мапа з укриттями, яка в подальшому доступна для користувача у офлайн режимі. Користувачеві доступна фільтрація сховищ по районам міста. При виборі того чи іншого укриття відображається повна інформація про його тип, власника, форму власності, наявність пристосування для доступу осіб з інвалідністю та інших маломобільних груп (пандусу) та примітки. Також надається номер телефону власника і нещодавно була додана форма відгуку для повідомлення порушень в облаштуванні укриття. Мапа укриттів додатку зображена на рис. 1.3.

Рис. 1.3. Мапа укриттів додатку «Київ Цифровий»

Для побудови маршруту до укриття застосунок пропонує використати той самий сервіс Google Maps – користувач самостійно обирає укриття та натискає кнопку “Покроково” для автоматичної побудови маршруту.

Крім безпекових, “Київ Цифровий” пропонує ряд адміністративних та інформаційних функцій, зокрема:

- оплата проїзду та паркування;
- перегляд інформації про штрафи для водіїв;
- запис до Центру Надання Адміністративних Послуг;
- мапа евакуаційних пунктів;
- сповіщення про зміни графіку, маршруту руху громадського транспорту та ін.

Переваги рішення:

- наявність детальної інформації про кожне укриття;
- наявність форми зворотнього зв’язку.

Недоліки рішення:

- відсутня можливість фільтрації по типу укриття чи призначенню. Фільтрація по районам є малоінформативною;
- відсутність автоматичного підбору найближчого укриття.

1.3. Порівняння наявних рішень

Виконаємо порівняння наявних рішень за такими критеріями:

- Наявність автоматичного підбору найближчого укриття.
- Наявність можливості перегляду детальної інформації про кожне укриття.
- Реалізована можливість фільтрації укриттів за типом/призначенням/наявністю пандусу для маломобільних груп населення.

Ці критерії необхідні для ефективного вирішення задач, які поставлені перед безпековими застосунками.

Складаємо порівняльну таблицю (табл. 1.1). Якщо певний критерій виконується в застосунку, то відповідна клітинка таблиці буде позначена знаком «+», а в протилежному випадку – знак «–».

Таблиця 1.1

Порівняльна характеристика наявних рішень

	Наявність автоматичного підбору найближчого укриття	Наявність можливості перегляду детальної інформації про кожне укриття	Реалізована можливість фільтрації за типом, призначенням, наявністю пандусу
«ДеУкриття»	+	+	+
«Прилуки Незламні»	–	–	–
«Київ Цифровий»	–	+	–

Отже, виходячи з порівняльного аналізу найбільшу функціональність реалізує застосунок «ДеУкриття». В ньому наявні всі функції, які забезпечують швидке знаходження укриття і побудову маршруту до нього. Якщо брати до уваги застосунок «Київ Цифровий», оскільки саме він є прямим конкурентом розроблюваної системи, то хоча і основний функціонал в ньому наявний, проте він все ще недостатній для швидкої побудови маршруту і знаходження укриття користувачем, а також не імплементує необхідну фільтрацію за типом, призначенням та наявністю пандусу.

1.4. Постановка задачі

1.4.1. *Визначення мети розробки*

Зважаючи на результати аналізу предметної галузі та порівняльного аналізу наявних рішень, метою даної дипломної роботи є розробка мобільного застосунку, який би забезпечував такі основні функції, як:

- Автоматичний пошук найближчого укриття.
- Можливість переглянути детальну інформацію по укриттю.
- Можливість фільтрації за типом, призначенням та наявністю пандусу в укритті.

Розроблюваний мобільний застосунок має мати зручний інтерфейс з мінімальною кількістю кроків для отримання результатів пошуку. Це забезпечить швидку реакцію користувача на необхідність пройти в укриття цивільного захисту.

Крім того можлива наявність розширеного функціоналу. Наприклад збереження місць розташування для подальшого пошуку відносно кожного з них, чи збереження обраних укриттів для швидкого пошуку останніх. Також варто додати можливість форми зворотнього відгуку з можливістю додати нове укриття чи ввести зміни до наявного при невідповідності інформації. Такий розширений функціонал, очевидно потребує авторизації користувача у системі, яка має бути максимально простою швидкою.

1.4.2. *Функціональні вимоги*

В результаті проведеного аналізу було виділено функціональні вимоги до розроблюваної системи (табл. 1.2).

Пріоритетність кожної вимоги була оцінена за методом MoSCoW [2]. Шкала пріоритетності наступна:

1. *Must* – вимога обов’язкова до реалізації. Представляє основний функціонал застосунку.
2. *Should* – вимога рекомендована до реалізації. Забезпечує розширення до основного функціоналу, проте не є критичною.

3. *Could* – вимога може бути реалізована. Такі вимоги представляють розширений функціонал затосунку і надають більше функцій користувачеві, проте їх відсутність не змінить основної важливості застосунку.
4. *Won't have* – вимога буде реалізована в майбутньому. Цей тип вимог є покращенням функціональності, яке буде здійснено в подальшому.

Таблиця 1.2

Функціональні вимоги системи

Код вимоги	Опис вимоги	Пріоритет вимоги
Ф1	Додаток надає можливість автоматичного пошуку найближчого до місця положення користувача укриття із зазначенням відстані до нього	Must
Ф2	Додаток дозволяє розширений пошук за відстанню з можливістю обрати інші варіанти укриттів	Should
Ф3	Додаток передбачає можливість фільтрації укриттів за типом (підвал, станція метро, підземний перехід), призначенням (найпростіше укриття, споруди подвійного призначення), адресою та наявністю пандусу.	Should
Ф4	Додаток надає можливість автоматичної побудови маршруту до найближчого або обраного користувачем укриття на мапі.	Must
Ф5	Користувач має можливість переглянути інформаційну довідку мапи з відповідними умовними позначеннями.	Must

Продовження табл. 1.2

Ф6	Додаток передбачає можливість авторизації. Є три ролі користувачів: • Неавторизований користувач – користувач, що не має облікових даних. • Авторизований користувач – користувач що має облікові дані. • Адміністратор.	Could
Ф7	Пошук найближчого укриття, фільтрація та побудова маршруту має бути доступним без авторизації користувача в системі.	Must
Ф8	Додаток дозволяє Авторизованому користувачу такі розширені функції: • додавання обраних укриттів та пошукових запитів за адресою; • збереження історії місцезнаходжень користувача (за вимогою користувача); • можливість додавання власного укриття, не поміченого на мапі (за модерацією Адміністратора) • можливість подати інформацію про недостовірність місцезнаходження укриття (через модерацію Адміністратора)	Could
Ф9	Система дозволяє Адміністратору виконувати такі функції: • перегляд усіх укриттів; • перегляд запитів користувачів на додавання укриття; • перегляд запитів користувачів на зміну інформації про укриття; • підтвердження запиту користувача і додавання нового укриття, створеного запитом; • відхилення запиту; • додавання укриття; • видалення укриття.	Won’t have

На основі визначених функціональних вимог створено діаграму прецедентів у нотації UML, яка зображена на рис. 1.4.

1.4.3. *Нефункціональні вимоги*

Крім функціональних вимог, до системи також було визначено ряд нефункціональних вимог. Нефункціональні вимоги включають в себе ті вимоги, що вказують на якість та ефективність програмного забезпечення. Нефункціональні вимоги системи наведені у табл. 1.3.

Таблиця 1.3

Нефункціональні вимоги системи

Код вимоги	Опис вимоги	Пріоритет вимоги
НФ1	Додаток має бути доступним у будь-який час доби.	Must
НФ2	Інтерфейс пошуку укриття має бути наочним та вимагати мінімальної кількості кроків для отримання маршруту.	Should
НФ3	Швидкість побудови маршруту має бути максимально можливою.	Must
НФ4	Основні функції додатку мають бути доступними будь-якому користувачеві незалежно від наявності в нього облікових даних.	Should

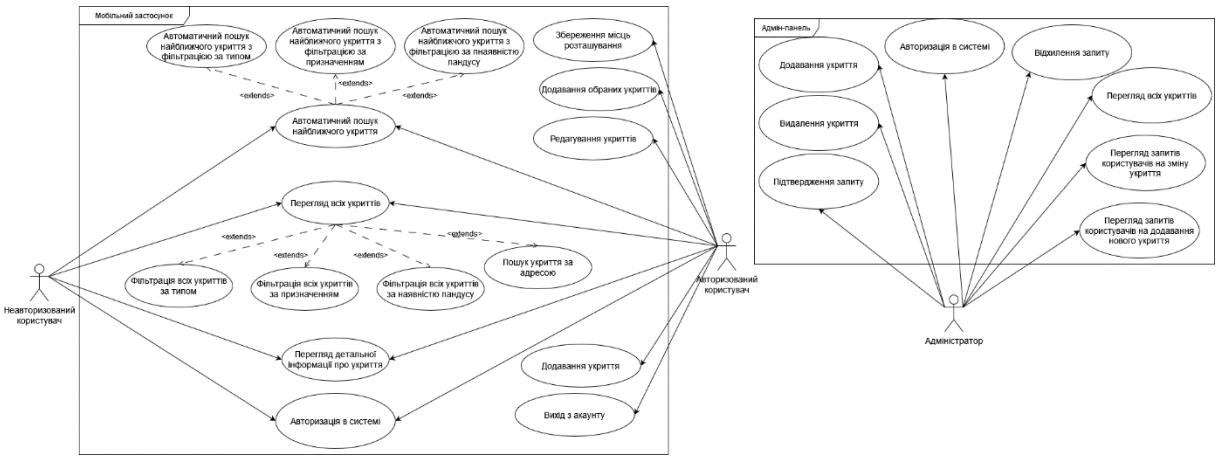


Рис. 1.4. Діаграма прецедентів у нотації UML

1.5. Висновки до розділу

В даному розділі дипломного проєкту було проаналізовано предметну галузь, визначено основні проблеми та ступінь їх актуальності. Проблема наявності безпекових застосунків з інформацією про укриття цивільного населення є досить актуальною в наш непростий час, тому вирішення існуючих проблем є необхідним завданням.

Крім цього було досліджено та проаналізовано наявні рішення, що вирішують існуючу проблему. Було виконано порівняльний аналіз рішень за трьома основними критеріями. Аналіз допоміг визначити ступінь вирішення проблеми існуючими конкурентами та сформувати на їх базі власні підходи для її розв’язання.

До розроблюваної системи були визначені функціональні та нефункціональні вимоги, оцінено їх пріоритетність та створено діаграму дій (прецедентів) у нотації UML.

2. ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1. Алгоритмічне підґрунтя

2.1.1. *Алгоритм пошуку найближчого укриття*

Для автоматичного і швидкого пошуку найближчого укриття потрібен певний алгоритм, який би забезпечував швидкість та точність обчислення і вибору мінімального за відстанню укриття. Такі алгоритми відносять до класу алгоритмів пошуку шляху. В дійсності, задача пошуку шляху, як вказано у [3], тісно пов'язана з задачею про пошук найкоротшого шляху з теорії графів, що має на меті пошук найоптимальнішого шляху між двома нодами зваженого графу. Тому часто для даного пошуку використовують такі структури даних, як граф чи дерево і відповідні алгоритми на них. Наприклад, найвідоміша компанія Google та їх спеціалізована система з відображення мап Google Maps, для пошуку та побудови найоптимальніших маршрутів використовує алгоритм Дейкстри та A – алгоритм [4, 5].

Алгоритм Дейкстри – один з найпопулярніших алгоритмів для пошуку найкоротшого шляху у зваженому графі. Він був розроблений та запропонований у 1956 році нідерландським науковцем Едсгером Вібе Дейкстрою [6].

Для реалізації алгоритму необхідно мати зважений граф та обрати в ньому початкову ноду та кінцеву. Граф – це структура, що являє собою сукупність об’єктів із зв’язками, що відображає певну систему. Об’єкти в графі називають *нодами* або *вузлами, вершинами*, а зв’язки – *ребрами*. Зваженим граф вважається тоді, коли всі його ребра мають певну “вагу” – додаткову інформацію про це ребро. Для випадку з алгоритмом Дейкстри в якості ваг використовують числові значення, наприклад відстані від одного вузла до іншого на карті.

Після вибору початкового вузла графу, кожен інший вузол позначається як невідвіданий (*non-visited*), а відстань від початкового вузла до всіх інших вузлів графу позначається як *нескінченність*. Відстань від

початкової ноди до неї ж самої – 0. Всі невідвідані вершини формують множину, де кожен елемент визначається відстанню (*орієнтовною відстанню*) до цього елемента від початкового вузла. Далі початкова нода стає *поточною* і для неї розглядаються всі її невідвідані сусіди та визначається відстань до них. Відстань вираховується таким чином: сума відстані орієнтовного шляху попереднього вузла та відстань від попереднього до цього вузла. Якщо отримане значення менше за *орієнтовну відстань* цього вузла, то воно приймається як нова орієнтовна відстань, в іншому випадку значення залишається незмінним. Після того, як всі сусіди цього вузла оцінені, поточний вузол відмічається як *відвіданий* (*visited*) і його вилучають з множини невідвіданих, а наступним поточним вузлом стає той, орієнтовна відстань до якого від поточного є найменшою. Алгоритм закінчується тоді, як кінцевий вузол позначається відвіданим. Якщо кінцева нода не була відвідана або найменша орієнтовна відстань на якомусь етапі алгоритму залишилась нескінченною, то це означає, що шляху між двома вузлами не існує. Блок-схема алгоритму Дейкстри зображена на рис. 2.1.

Складність цього алгоритму залежить від того, яку структуру даних обрано для реалізації множини невідвіданих вузлів. В більшості випадків це черга з пріоритетом. При реалізації цього алгоритму за допомогою бінарної черги з пріоритетом складність алгоритму складає $O((|E|+|V|)\log|V|)$, де E – кількість вершин, а V – кількість ребер. Якщо для черги з пріоритетом буде використано купу Фібоначчі, то складність алгоритму можна покращити до [7]: $O(|E|+|V|\log|V|)$.

В розроблюваній системі для пошуку найближчого укриття використано інший алгоритм, який був досліджений у [8] як покращення алгоритму Дейкстри, а саме алгоритм пошуку на основі *kd-дерева*.

Дерево є схожою структурою за морфологією до графа, адже воно є підвидом графа, у якому відсутні цикли і наявний шлях між кожною з його вершин. Тобто, дерево – зв'язний ациклічний граф.

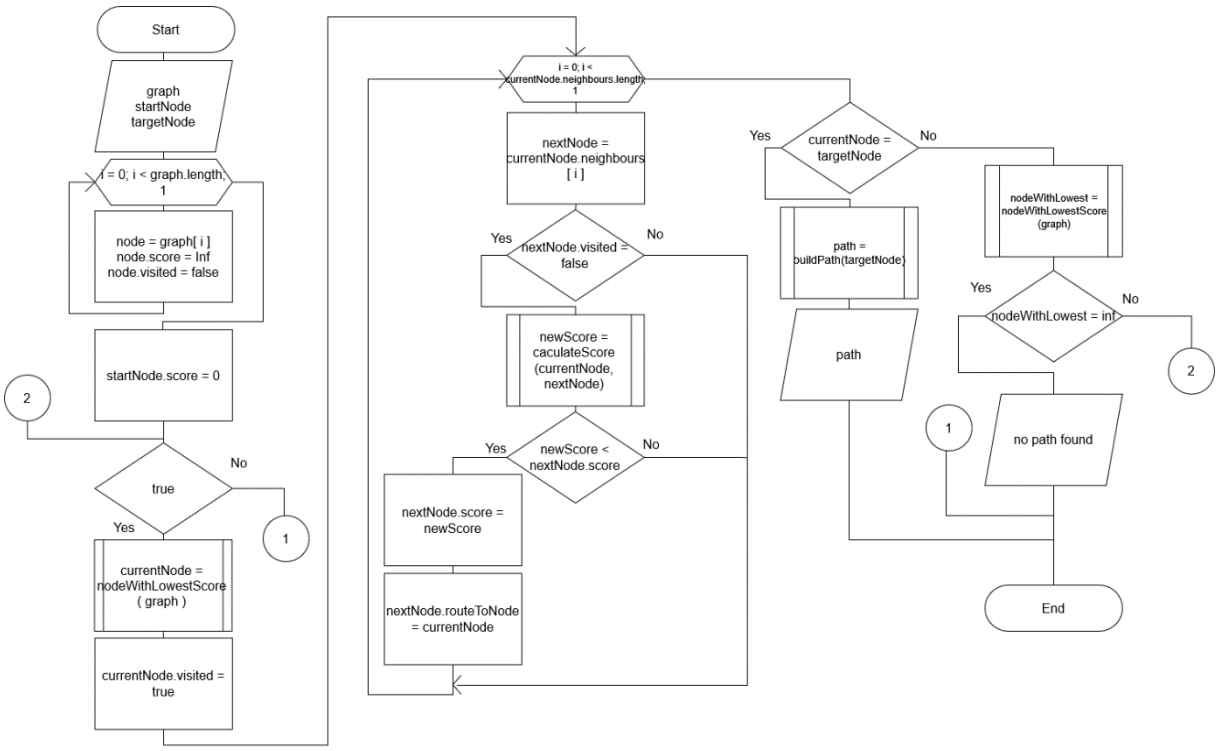


Рис. 2.1. Блок-схема алгоритму Дейкстри

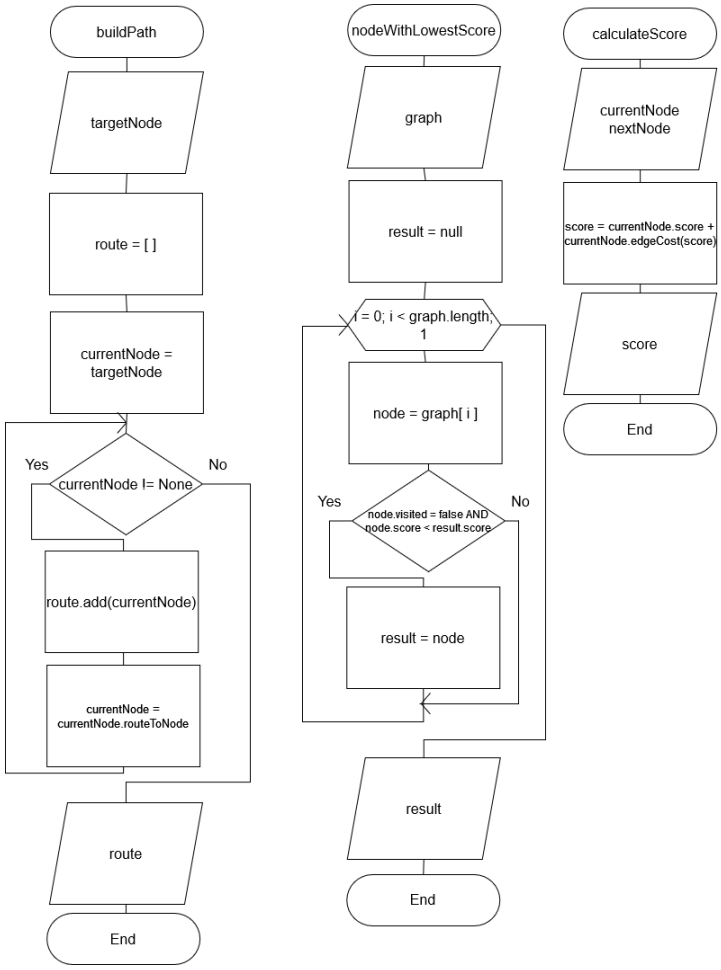


Рис. 2.2. Допоміжні підпрограми алгоритму Дейкстри

На відміну від звичайного графа, дерево має:

- корінь – відокремлений вузол, основа дерева, має множину піддерев;
- вузли – внутрішні вузли дерева, які також мають піддерева;
- листя – вузли дерева, що не мають піддерев, крайні вузли, вільні ноди.

kd-дерево, або *k-вимірне* дерево – це структура даних з поділом простору для впорядкування точок у k-вимірному просторі [9]. Така структура є ідеальною при використанні для обробки просторових даних і у випадку географічних координат на площині Землі вона зводиться до 2d дерева.

Побудова дерева за канонічним методом відбувається наступним чином:

1. Якщо точка у множині одна, то вона стає листовим вузлом.
2. Якщо точок декілька, робиться поділ координат на дві області, прямою, перпендикулярною до однієї з осей. Утворюється дві підмножини точок, що стають піддеревами, а коренем дерева стає координата прямої поділу по осі.
3. У кожному піддереві відбувається поділ на дві області прямою, перпендикулярної до іншої координати, тобто якщо на першому кроці був поділ по координаті *x*, на цьому кроці робиться поділ по *y*. На кожному наступному кроці – чергування осей.
4. Такий поділ відбувається рекурсивно до моменту, поки у області не перебуває лише одна точка.

Поділ точок перпендикулярними лініями можна робити у декілька стратегій. У канонічному методі координати прямих поділу обираються як медіани підмножини відповідних координат точок простору. Іншим популярним варіантом поділу є поділ посередині розмаху координат. Побудоване дерево за канонічним методом буде збалансованим та,

відповідно до дослідження [8], буде приводити до найкращих результатів складності алгоритму пошуку.

Власне пошук найкоротшого шляху з використанням структури k-вимірного дерева базується на тому, що площина всіх координат точок буде представлена у вигляді дерева. При пошуку шляху від точки до найближчого сусіда, для даної точки буде здійснено її введення у дерево як листового вузла і подальше обрахування відстані від неї до листових вузлів сусідніх піддерев. Обрахування відстані здійснюватиметься за формулою Евклідової відстані (2.2).

$$p = \left(p_1, p_2, ..., p_n\right), q = \left(q_1, q_2, ..., q_n\right), \tag{2.1}$$

$$d = \sqrt{\sum_{i=1}^{n} \left(p_i - q_i\right)^2}. \tag{2.2}$$

У формулі (2.1) p, q – дві точки n-вимірного простору.

Таким чином в результаті пошуку будуть знайдені n найближчих сусідів, значення відстаней яких буде відсортовано за зростанням і обрано найближчого.

Алгоритм пошуку наступний:

1. Якщо абсциса кореня менша за абсцису точки, переходимо у ліве піддерево, якщо більша переходимо у праве піддерево.
2. Якщо ордината точки менша за ординату кореня піддерева, переходимо у ліве піддерево для даного, якщо ж більша – у праве.
3. Чергуючи координати і далі, проходимо по дереву до того моменту, як досягаємо листового вузла. Він вважається першим з найближчих.
4. Повертаємось на рівень вище – до кореня останнього піддерева. Він також вважається одним за найближчих.
5. Перевіряємо, чи є ще вузли у сусідньому піддереві, що можуть бути найближчими. Якщо так – додаємо їх у список.
6. Продовжуємо підніматися по рівням, поки не отримаємо всі листові точки сусідніх піддерев.

7. Для кожного отриманого сусіда обчислюємо відстань від початкової точки і обираємо найближчого.

Блок-схема алгоритму пошуку наведена на рис. 2.3.

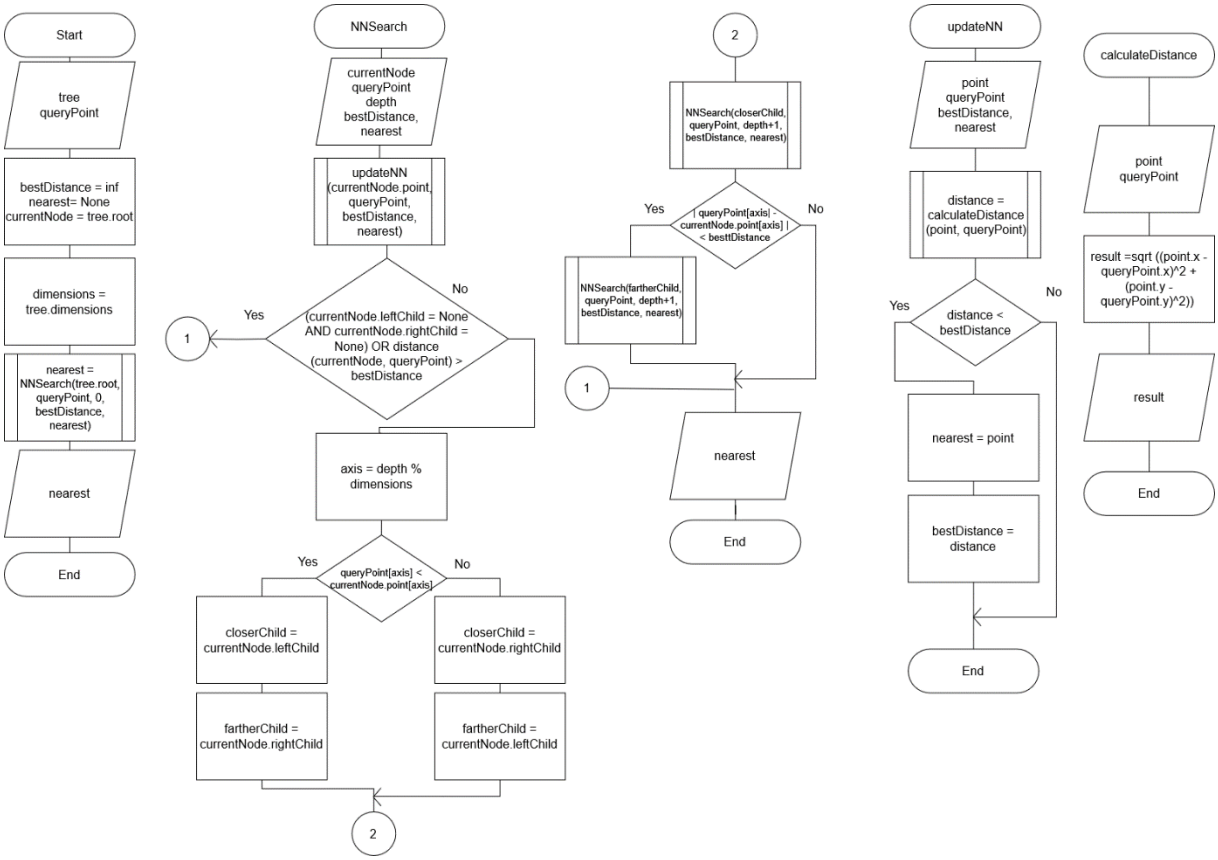


Рис. 2.3. Блок-схема алгоритму пошуку найближчого сусіда з використанням kd-дерева

Складність алгоритму пошуку на основі kd-дерева відповідно до [10] складає $O(n \log n)$.

У порівнянні з пошуком Дейкстри, даний алгоритм показує більшу ефективність, тому був обраний для реалізації у системі. Алгоритм Дейкстри, як вже було сказано, використаний при побудові маршруту до знайденого найближчого укриття на платформі Google Maps.

2.1.2. *Алгоритми фільтрації*

Відповідно до функціональних вимог розроблюваного програмного забезпечення, передбачена філтрація укриттів за типом, призначенням та наявністю пандусу.

Фільтрація буде відбуватися на стороні сервера бази даних. При фільтрації буде використані SQL запити із оператором WHERE. WHERE – стандартний оператор для накладання фільтру до даних у БД. Він оголошує умову фільтрації, а результатом запиту з таким фільтром є перелік записів, які задовольняють дану умову. Для прикладу, при фільтрації по типу укриття, запит буде мати вигляд, наведений у лістингу 2.1.

Лістинг 2.1. Приклад запиту для фільтрації укриття за типом

```
SELECT <укриття> FROM <ім'я_таблиці> WHERE <тип>='станція метро';
```

При виконанні запиту з лістингу 2.1, за замовчуванням використовується порядкове сканування всієї підфільтерної таблиці і порівняння значення поля типу для кожного рядка, тобто виконується лінійний пошук. Складність лінійного пошуку *O(n).*

Такий алгоритм є затратним як по ресурсам пам'яті, так і по часу, адже маючи значну кількість рядків СУБД необхідно буде просканувати всі з них.

Для прискорення виконання запитів у СУБД використовують індекси – структури даних, які покращують виконання запитів шляхом використання алгоритмів пошуку відповідно цим структурам. Найчастіше в якості структур індексу використовують дерева чи геш-таблиці.

Існує декілька видів табличних індексів. Їхні основні характеристики на прикладі індексів PostgreSQL [11]:

- *B-Tree*: індекс на основі збалансованого бінарного дерева. Складність пошуку становить *O(log n)*. Підходить для виконання запитів порівняння та діапазону на відсортованих даних.

- *Hash*: індекс на основі геш-таблиці, що зберігає 32-бітний геш-код, взятий зі значення індексної колонки таблиці. Ідеально підходить для операцій порівняння. Складність пошуку в середньому становить *O(1).*
- *GiST*: індекс на основі B+ дерева. В більшості підходить для геометричних даних і не підтримує операції порівняння.
- *SP-GiST*: індекс на основі різних видів дерев (quadtree, kd-tree, radix tree). Так само як і GiST підходить для геометричних даних та запитів і не підтримує операції порівняння.
- *GIN*: індекс на основі бінарного дерева. Найкраще працює з багатокомпонентними даними, такими як масиви.
- *BRIN*: індекс, що заснований на логіці поділу таблиці на блоки. Найкраще підходить для даних, які вже певним чином відсортовані та корельовані з їх фізичним розташуванням.

Для фільтрації в системі буде використано геш-індекс, оскільки він відповідає за операції порівняння і є швидшим за пошук у бінарному дереві, що пропонується по замовчуванню планувальником запитів.

Як вже було зазначено, геш-індекс базується на геш-таблиці – структурі даних, що являє собою асоціативний масив типу ключ-значення. В ролі ключа в індексі виступає геш-код поля, а в ролі значення – записи таблиці з даним полем.

При фільтрації таблиці, в якій на полі, за яким здійснюється фільтрація, створено геш-індекс, алгоритм наступний:

1. Обчислення геш-функції умови фільтрації $i=hash(value)$.
2. Пошук у геш таблиці відповідної комірки, що відповідає значенню гешу $H[i]$.
3. Повернення результатів – записів комірки $H[i]$.

Блок-схема алгоритму зображена на рис. 2.4.

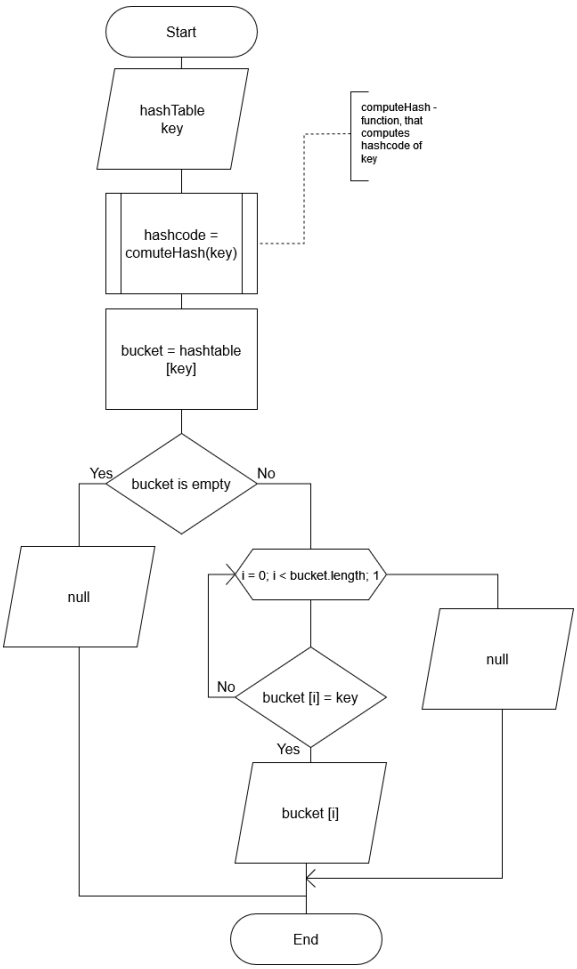


Рис. 2.4. Блок схема алгоритму фільтрації на основі геш-таблиці

У PostgreSQL функція гешування є такою, що ставить у відповідність гешованому значенню 32-бітне ціле число.

2.2. Огляд програмних засобів розробки

2.2.1. *Мова розробки Python*

Python – потужна високорівнева мова програмування, що широко використовується розробниками веб-додатків, десктопних додатків, а особливо складних обчислювальних систем, як систем машинного навчання (machine learning) і дата-аналітиків. Популярність цієї мови залишається високою впродовж декількох років і, відповідно до індексу TIOBE [12], наразі є найпопулярнішою мовою розробки.

Особливості включають в себе [13, 14]:

- Підтримка графічного інтерфейсу розробки.
- Об’єктно-орієнтований підхід.
- Високо портативна мова (дозволяє розробку на різних платформах без необхідності зміни конфігурації).
- Широкий список бібліотек.
- Підтримка інтеграції з іншими мовами (будучи заснованою на C, Python підтримує виконання коду, написаного на Java, C, C#).

Python підтримує динамічну типізацію, що дозволяє визначати тип змінної безпосередньо під час виконання програми.

Найбільшою відмінністю даної мови від інших є те, що вона є інтерпретованою мовою – виконання програми йде порядково та немає необхідності створювати виконувані файли. Вихідний код одразу інтерпретується в машинний код (*bytecode*). Це дозволяє підтримувати більш прозорий процес розробки та простіше відслідковування помилок у вихідному коді.

Однією з важливих особливостей мови Python є її об’єктно-орієнтованість. Усі змінні, функції та класи, створювані у коді цією мовою вважаються об’єктами.

Особливості та можливості реалізації ООП в Python [13]:

- Клас є об’єктом.
- Підтримка інкапсуляції, поліморфізму та наслідування.
- Приховані (приватні) об’єкти класу відрізняються від публічних лише особливим іменем.
- Наявність керування життєвим циклом об’єкта – конструктори, деструктори, розподільники пам’яті.
- Доступність метапрограмування (тригери на створення класів, управління створенням класів, тощо).
- Підтримка вкладеності функцій, класів у функції та інші класи.

Завдяки властивості об’єктності функції, у мові Python також реалізована парадигма функціонального програмування.

Python має вбудований механізм обробки винятків. Для цього у мові існують оператори *try, except, else, finally* та *raise.* Завдяки тому, що для запуску коду немає потреби у її попередній компіляції, обробка помилок не погіршує час виконання коду та не вимагає значних витрат ресурсу пам’яті, що є дуже доцільним при вимозі швидкого виконання обчислень не невеликих машинах.

Також однією з переваг Python є те, що вона є широко портовною і працює на більшості платформ. Серед них:

- Microsoft Windows;
- FreeBSD;
- GNU/Linux;
- Mac OS/Mac OS X;
- iPhone OS 2.0 і вище;
- Symbian;
- Android.

У UNIX системах Python вбудована в ядро і основні її функції доступні прямо “з коробки” – відразу після встановлення даної ОС на пристрій.

Значною перевагою, яка приваблює більшість розробників на Python це наявність великого реєстру пакунків (бібліотек) для розробки *PyPI.* Даний реєстр розміщений у відкритому доступі і кожен може долучитися до створення власного пакету з перед встановленими функціями. Станом на травень 2023 року реєстр налічує 451 764 проєкти та 697 138 зареєстрованих користувачів, а кількість незареєстрованих користувачів варто рахувати мільйонами. Наявність пакетів та бібліотек значно спрощує розробку програмного забезпечення, оскільки велика кількість складного функціоналу може бути реалізована за допомогою написання кількох рядків коду.

Найбільш широко використовуваною бібліотекою Python є NumPy. Дана бібліотека надає широкий спектр математичних методів та типів для роботи з даними. Вона дозволяє обробку та перетворення даних для їх дослідження у статистичному аналізі чи математичній обробці даних для подальшого їх використання.

Ще однією з використовуваних бібліотек Python є Scipy – потужний інструмент алгоритмічної обробки даних. Вона реалізує більшість алгоритмів для оптимізації, інтеграції, обробки сигналів, багатовимірної обробки зображень та статистичного аналізу даних. Scipy написана як обгортка над більш низькорівневий мовами, як Fortran, C та C++, що дозволяє зберегти їхню високу швидкість та продуктивність і створити легкий інструмент для використання у розробці аналітичних систем.

Крім бібліотек, Python має великий перелік фреймворків. Фреймворк – це колекція пакетів чи модулів що дозволяє розробку різних застосунків. Оскільки Python часто використовується як інструмент веброзробки, то вона налічує більше 10 вебфреймворків. Серед них варто відмітити Flask – мікрофреймворк для вебдодатків на основі Werkzeug та рушія шаблонів Jinja2. До основних властивостей цього фреймворку відносять:

- Сервер розробки та вбудований відлагоджувач.
- Вбудована підтримка unit-тестування.
- Управління запитами RESTful.
- Підтримка безпечних файлів кукі.

Ці та інші особливості роблять Flask зручним у використанні для написання серверної частини вебзастосунків. А у поєднанні з математичним інструментарієм Python дозволяє створення потужних обчислювальних серверів для розробки мобільних застосунків і, тим самим, перенесення складної логіки програмного забезпечення на сторону серверу.

2.2.2. *Фреймворк React Native*

React Native – фреймворк інтерфейсу користувача для розробки мобільних застосунків, розроблений компанією Meta (Facebook) та базується на бібліотеці React мови JavaScript. React Native дозволяє розробку додатків для мобільного пристрою за принципом, сходим до написання вебдодатків. Його реалізація схожа на реалізацію React, крім того, що React Native працює безпосередньо на пристрої без використання об'єктної моделі документа (DOM), за допомогою інтерпретації коду у JavaScript та спілкування його з нативними бібліотеки пристрою за допомогою серіалізованих даних через асинхронний і пакетний міст [15].

Концепція React Native дуже добре відображена у їхньому девізі: "Вивчи раз, пиши всюди" ("*Learn once, write anywhere.*") [16]. Через те, що фреймворк взаємодіє напряму з нативними бібліотеками пристроїв (Java для Android та Swift для iOS), він дозволяє легку кросплатформну розробку застосунків.

Як вже було зазначено, React Native базується на бібліотеці React і тому використовує більшість можливостей цієї бібліотеки. До таких належать:

- Декларативність – опис інтерфейсу застосунку виконується декларативними компонентами, що дозволяє легку розробку та більшу прозорість коду.
- Використання компонент – JSX декларовані компоненти створюють перелік елементів, які можуть бути використані для розробки інтерфейсу.
- Швидке відлагодження за допомогою режиму *live reload*.
- Використання атрибутів елементів.
- Використання виразів JavaScript та ін.

Як і у React, основною одиницею розробки інтерфейсу є *компонента* – певний елемент інтерфейсу, з яким може взаємодіяти користувач. Приклад базової компоненти – компонента *View*. Вона являє собою контейнер

інтерфейсу з різними способами розміщення елементів всередині, що є базою для створення графічного інтерфейсу. Аналог з DOM – контейнер *div.*

Для будь-якої компоненти React Native можна налаштувати параметри її відображення, а також прив'язати певні динамічні функції у відповідь на взаємодію користувача з цією компонентою. Як приклад, для компоненти *Button* найпоширенішою властивістю є *onPress*, що визначає поведінку додатку після натискання на цю кнопку користувачем. Така можливість, як реактивність на інтеракцію користувача зі застосунком дозволяє розробляти елегантні та зручні інтерфейси.

Популярність React Native як інструменту кросплатформної розробки зростає впродовж останніх років. Серед його переваг над нативними інструменами розробки є:

- швидкість та розробки;
- простота використання;
- висока можливість перевикористання коду;
- наперед розроблені бібліотеки компонент;
- більш сучасний інтерфейс розробленого додатку;
- легкість застосунку, менше використання пам'яті на клієнтському пристрої;
- велике ком'юніті розробників.

Останній пункт не варто недооцінювати, оскільки саме завдяки великій кількості розробників швидкість вирішення проблем та підтримка у розробці значно спрощують саму розробку і роблять цей процес ефективнішим. Крім того, React Native має документацію з великою кількістю прикладів, що допомагає малодосвідченим розробникам починати створення додатків набагато швидше.

Як і React, React Native має велику кількість бібліотек та пакетів, що дозволяють використання наперед визначених функцій у декілька рядків і значно спрощують логіку застосунку. Наразі однією з таких бібліотек виступає Expo Go. *Expo* – це набір інструментів і сервісів, побудованих на

базі React Native, і, хоча він має багато функцій, найбільш важливою функцією є те, що він може допомогти написати додаток за лічені хвилини [17]. Цей інструмент дозволяє відлагодження застосунку на різних платформах шляхом встановлення їх застосунку Expo Go на iOS чи Android. Після цього, за умови підключення пристроїв до однієї локальної мережі, ви можете тестувати застосунок одразу на цільових платформах. Ця можливість дозволяє спростити відлагодження та швидкість тестування, а також спрощує розміщення застосунку у магазинах застосунків.

Отже, React Native є ефективним інструментом розробки кросплатформенних мобільних застосунків з широким спектром функцій. Він дозволяє швидку розробку та оновлення, а також підтримує взаємодію з нативними компонентами, що робить його універсальним.

2.2.3. *СУБД PostgreSQL*

PostgreSQL – популярна система керування (управління) базами даних, заснована на прототипі POSTGRES версії 4.2, що був розроблений в Університеті Берклі. Як зазначають самі розробники, POSTGRES розробив багато концепцій, які стали доступними в деяких комерційних СУБД набагато пізніше [18].

Основною перевагою PostgreSQL над іншими СУБД є те, що вона має відкритий код та не контролюється однією компанією. Її програмне забезпечення – це робота групи людей, які постійно вдосконалюють систему та створюють нові версії ПЗ. Сервер PostgreSQL написано мовою програмування C. Для інсталяції відповідні файли мають бути скомпільовані.

PostgreSQL підтримує частину стандарту SQL, хоча деякі можливості з нього підтримуються зміненим інтерфейсом і функціями. Проте це не заважає PostgreSQL мати велику кількість можливостей. Варто зазначити, що з кожним релізом нової версії розробники намагаються задовольнити

повну відповідність PostgreSQL стандарту, хоча наразі жодна з існуючих СУБД повністю не відповідає йому.

Основні можливості PostgreSQL [11]:

- складні запити;
- зовнішні ключі;
- тригери;
- розрізи даних (database view);
- цілісність транзакцій;
- керування багатоверсійним паралельним виконанням (MVCC);
- індексування;
- підтримка всіх рівнів ізоляції транзакцій;
- планувальник запитів для збирання статистики запитів та ін.

Дана СУБД має багато вбудованих функцій та процедур. Крім вбудованих, користувач має змогу створити власні на одній з мов, що підтримуються сервером PostgreSQL, а саме:

1. C, C++ та Java (за допомогою PL/Java).
2. Мови розробки сценаріїв PL/Perl, PL/Python, PL/Ruby, тощо.
3. Спеціальною вбудованою мовою PL/pgSQL.

Також PostgreSQL підтримує індексування таких типів, як B-дерево, геш-індекс, GiST, GIN, SP-GiST, BRIN.

PostgreSQL підтримує зберігання різних типів даних [18]:

- Примітивних:
  - цілочисельний тип Integer;
  - числовий тип Numeric;
  - рядки String;
  - логічний Boolean.
- Структурних:
  - час та дата Date/Time;
  - масив даних Array;
  - діапазони значень Range/Multirange;

- унікальний ідентифікатор UUID.

- Документних:
  - JSON;
  - XML;
  - ключ-значення (Hstore).
- Геометричних:
  - точка Point;
  - пряма Line;
  - коло Circle;
  - багатокутник Polygon.
- Користувацьких:
  - композитні типи;
  - користувацькі типи.

Завдяки тому, що PostgreSQL являє собою сервер бази даних, то її функціонал включає в себе можливість реплікації. Це забезпечує надійність серверу і надає високу доступність даних у будь-який момент часу. Реплікація може відбуватися синхронно, асинхронно чи логічно. Також у сервер PostgreSQL вбудовано певну кількість користувацьких рядкових утиліт адміністрування, як от pg_dump чи pg_basebackup – утиліти, що дозволяють робити резервне копіювання та бекап кластеру БД. Таким чином, розробниками може бути налаштована високо стабільна та безпечна система, що передбачає резервне копіювання, реплікацію та попередження несанкціонованого доступу до даних.

2.3. Висновки до розділу

Даний розділ дипломного проєкту призначений для аналізу та вибору методів і засобів розробки програмного забезпечення. В рамках цього розділу було проведено дослідження математичного підґрунтя для пошуку найближчого укриття і обрано ефективні алгоритми для реалізації пошуку. Для пошуку укриття буде використано алгоритм пошуку найближчого

сусіда на основі kd-дерева, а для побудови маршруту – використано алгоритм Дейкстри.

Крім алгоритмів пошуку були також описані розповсюджені алгоритми фільтрації та виконано їх аналіз. В результаті аналізу для фільтрації було обрано алгоритм пошуку в геш-таблиці.

Визначені алгоритми мають забезпечувати ефективну математичну обробку та виконувати основну задачу застосунку.

Також в даному розділі було обґрунтовано вибір засобів розробки застосунку. Основними критеріями вибору були кросплатформність інструменту, доступність та його потужність. Таким чином для розробки клієнтського застосунку було обрано фреймворк React Native, для розробки серверної частини – мову програмування Python та її фреймворк для розроби вебзастосунків. В якості бази даних було обрано PostgreSQL. Ці засоби дозволяють ефективну розробку системи та є перевіреними часом.

3. ОПИС РЕАЛІЗАЦІЇ ПРОГРАМНОГО ЗАСОБУ

3.1. Архітектура програмного забезпечення

Розроблене програмне забезпечення має *трирівневу архітектуру*. Такий тип архітектури системи є підвидом загальної багаторівневої архітектури *Multi-tier architecture*. В основі ідеї багаторівневості лежить розділення монолітної системи на декілька компонент за їхньою функціональністю. В більшості багаторівневих систем використовується наступне виділення компонентів:

- Презентаційний компонент – включає в себе користувацький інтерфейс та комунікаційний шар системи. Його основне призначення полягає в наданні графічного інтерфейсу користувача для роботи з системою: надання даних користувачеві та отримання даних від нього для подальшої обробки та збереження. Зазвичай являє собою вебсторінку або мобільний застосунок.
- Компонент бізнес-логіки – надає основну логіку системи. Тут зосереджена основна логічна обробка даних: відбувається отримання даних від користувацького інтерфейсу, їх обробка відповідно до задач системи та повернення оброблених даних назад до користувача або збереження їх у базі даних.
- Компонент даних – компонент, що відповідає за збереження оброблених даних. В сенсі компоненти даних виступає база даних, що зберігає дані системи та на запити від компоненту бізнес-логіки здатна повертати дані, які задовольняють запит.

Комунікація між компонентами відбувається порівнево, тобто презентаційний компонент може комунікувати лише з бізнес-логікою, як і компонент даних. Напряму доступ з презентаційного компоненту до компоненту бази даних неможливий.

Така структура системи має ряд переваг на монолітною. По-перше, вона дозволяє системі бути більш гнучкою та витривалою до змін. По-друге,

компоненти системи слабо зв’язані один з одним і добре параметризовані, що дозволяє їх перевикористання в розробці інших систем. По-третє, дана архітектура збільшує безпеку системи, оскільки при потенційному зломі однієї з компонент, інші при правильно налаштованих параметрах аутентифікації та взаємодії, залишаються закритими для зловмисника.

Архітектура системи зображена на рис. 3.1.

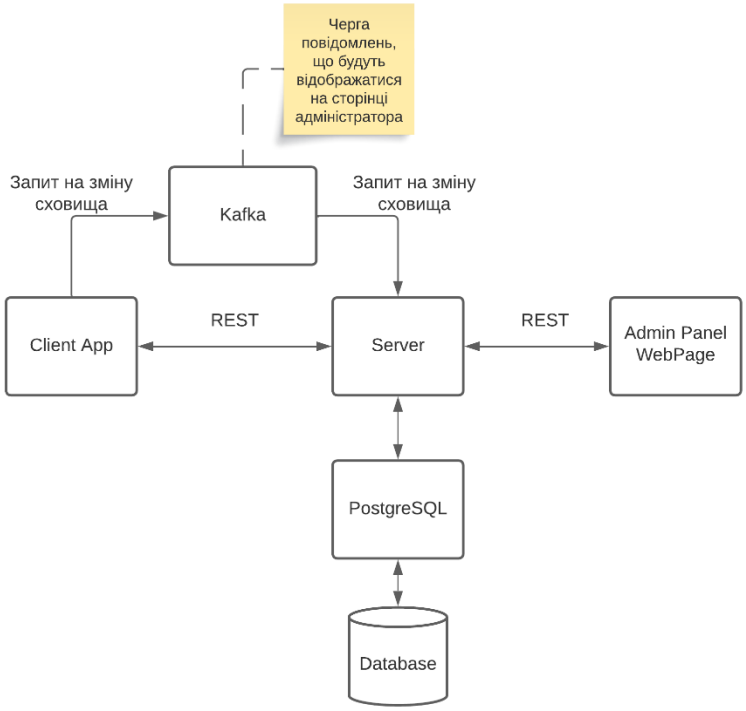


Рис. 3.1. Архітектура системи

Архітектура складається з таких компонент:

- *Client App* – компонент клієнтського застосунку – застосунок, який користувач встановлює на мобільний пристрій. Надає інтерфейс для роботи з системою для Неавторизованого користувача та Авторизованого користувача.
- *Server* – компонент бізнес-логіки – веб-сервер алгоритмічної обробки. Надає основний функціонал системи, а саме визначення найближчого укриття за допомогою алгоритмів, описаних у Розділі 2, та відповідає за оброблення даних користувача.

- *Database* – компонент даних – база даних системи. Зберігає дані про укриття, а також дані користувача такі, як його обрані укриття та збереженні місця.
- *PostgreSQL* – СУБД для бази даних. Надає API інтерфейс для комунікації з базою даних.
- *Admin Panel Web Page* – панель управління для Адміністратора. Вебсторінка для управління базою даних, що дозволяє перегляд активних запитів на створення чи зміну сховища у базі даних (в розробці).
- *Kafka* – черга запитів – сервер брокера повідомлень. Дозволяє зберігати запити користувачів на зміну чи створення нового сховища. Дані додаються клієнтом через клієнтський застосунок. Дані, що зберігаються в цій черзі, надходять на панель Адміністратора.

3.2. Структура бази даних

Для даної системи у якості бази даних використано СУБД PostgreSQL. Її структура наведена на рис. 3.2.

База данх складається з декількох таблиць. Приведемо їх детальний опис.

Таблиця Users – таблиця користувачів. Зберігає у собі дані користувача та присвоює кожному користувачеві унікальний ідентифікатор. Містить такі поля:

- id – унікальний цілочисловий ідентифікатор користувача; первинний ключ таблиці;
- displayName – псевдонім користувача, його ім’я, яке буде відображатися у додатку; рядкове значення;
- email – електронна пошта користувача. Виступає в ролі логіна користувача; унікальне рядкове значення;

- password – пароль користувача у зашифрованому вигляді. Шифрування відбувається на стороні клієнтського застосунку за алгоритмом md5; md5 рядок.

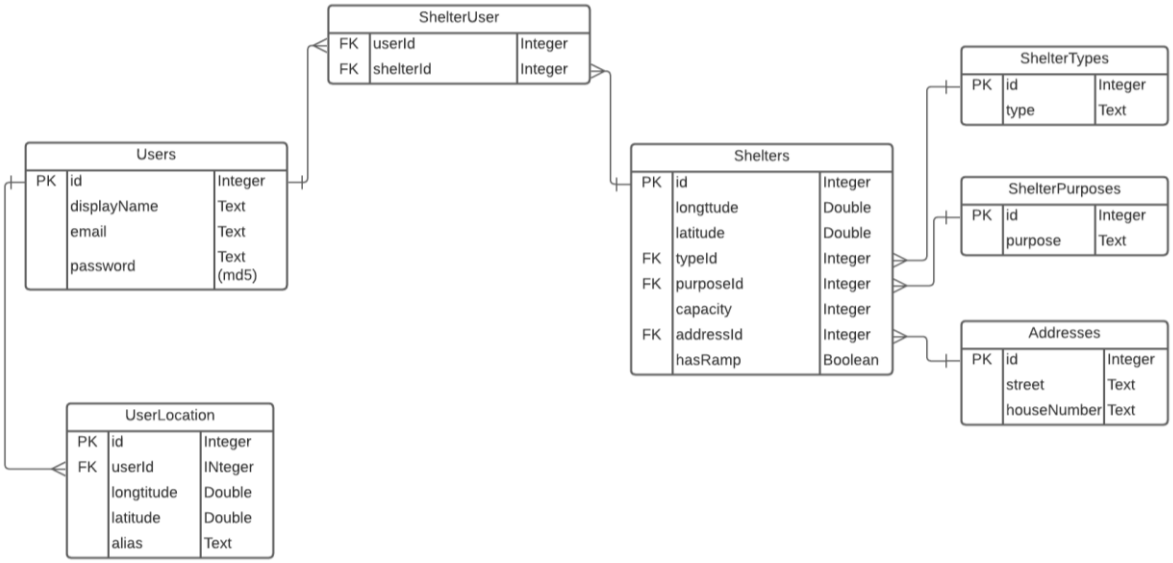


Рис. 3.2. Структура бази даних

Таблиця UserLocation зберігає збережені місця користувача. Містить такі поля:

- id – унікальний цілочисловий ідентифікатор запису; первинний ключ таблиці;
- userId – унікальний ідентифікатор користувача, якому належить даний запис місця; зовнішній ключ таблиці, цілочислове значення;
- longitude – координата довготи місця; дробове значення подвійної точності;
- longitude – координата широти місця; дробове значення подвійної точності;
- alias – користувацька назва місця; текстове значення.

Таблиця User та UserLocation пов’язані одна з одною зв’язком типу Один-до-Багатьох.

Таблиця Shelters – основна таблиця збереження даних про укриття. Містить такі поля:

- id – унікальний цілочисловий ідентифікатор укриття; первинний ключ таблиці;
- longitude – координата довготи укриття; дробове значення подвійної точності;
- longitude – координата широти укриття; дробове значення подвійної точності;
- typeId – ідентифікатор типу укриття; цілочислове значення; зовнішній ключ таблиці;
- purposeId – ідентифікатор призначення укриття; цілочислове значення; зовнішній ключ таблиці;
- capacity – місткість укриття, чоловік; цілочислове значення;
- addressId – ідентифікатор адреси укриття; цілочислове значення; зовнішній ключ таблиці;
- hasRamp – поле, що визначає наявність в укриття засобів для маломобільних груп населення (пандусу); приймає значення true або false.

Таблиця ShelterTypes зберігає ідентифікатори типів укриття. Містить такі поля:

- id – унікальний цілочисловий ідентифікатор типу; первинний ключ таблиці;
- type – назва типу; рядкове значення, що приймає такі типи:
  - підвал;
  - станція метро;
  - підземний перехід.

Таблиця ShelterPurposes зберігає ідентифікатори призначень укриттів. Містить такі поля:

- id – унікальний цілочисловий ідентифікатор призначення; первинний ключ таблиці;

- purpose – назва типу; рядкове значення, що приймає такі призначення:
  - найпростіше укриття;
  - споруди подвійного призначення.

Таблиця Addresses зберігає записи адрес укриттів. Містить такі поля:

- id – унікальний цілочисловий ідентифікатор адреси; первинний ключ таблиці;
- street – назва вулиці, на якій розташоване укриття; рядкове значення;
- houseNuber – номер будинку, в якому розташоване укриття; рядкове значення.

Таблиця ShelterUser – проміжна таблиця для створення зв'язку Багато-до-Багатьох між таблицями Users та Shelters. Містить такі поля:

- userId – ідентифікатор користувача; цілочисельне значення; зовнішній ключ таблиці;
- shelterId – ідентифікатор укриття; цілочисельне значення; зовнішній ключ таблиці.

Структура БД була приведена до третьої нормальної форми. Третя нормальна форма характеризується відсутністю транзитивних залежностей полів таблиці. Всі таблиці були перевірені на транзитивність і показали їх відсутність. Отже, база даних може вважатися нормалізованою.

3.3. Опис модулів програмного забезпечення

3.3.1. *Модуль клієнтського застосунку*

Фізичним представленням модуля клієнтського застосунку є власне розроблений мобільний додаток. Відповідно до технічного завдання, модуль розроблений за допомогою React Native та його допоміжного фреймворку Expo.

Розробка проводилася з використанням функціональних компонент React Native. Основний модуль міститься у файлі *App.js*. Тут розміщується

головний екран застосунку та його вкладки (*Знайти, Укриття, Профіль*). У початковому стані користувачеві відображається вкладка *Знайти.* Кожна зі вкладок є окремою функціональною компонентою та розміщена у директорії *src/screens.* Приклад опису компоненти наведений у лістингу 3.1

Лістинг 3.1. Опис компоненти профілю користувача у застосунку

```
import { StyleSheet, View, Text, Button } from "react-native";

import { useSelector, useDispatch } from "react-redux";
import { logout } from "../redux/actions/loginAction";

// Import components
import { LoginForm } from "../components/LoginForm";
import { UserComponent } from "../components/UserComponent";

export function UserScreen() {
 const dispatch = useDispatch();

 const isLogged = useSelector((store) => store.isLogged.isLogged);

 const handleLogout = () => {
   dispatch(logout());
 };

 if (isLogged) {
   return (
     <UserComponent />
   );
 } else {
   return <LoginForm />;
 }
}

const styles = StyleSheet.create({
 container: {
   flex: 1,
   backgroundColor: "#fff",
   alignItems: "center",
   justifyContent: "center",
 },
});
```

Стан функціональних компонент зберігається за допомогою React Hooks – спеціальних змінних, що є аналогами властивостям класу. Вони дозволяють зберігати дані, необхідні для даної функціональної компоненти, а також автоматично змінювати ці дані за допомогою спеціальної функції-сетера або такої, що встановлює нове значення стану. Приклад хуку основного екрану можна побачити у лістингу 3.2. На ньому змінна *location* –

стан, що визначає, чи відображається модальне вікно, а *setLocation* – метод, який дозволяє зміну цього значення.

Лістинг 3.2. Фрагмент коду компоненти головного екрану застосунку

```
export function HomeScreen() {
const [errorMsg, setErrorMsg] = useState(null);
const [location, setLocation] = useState(null);
const isLogged = useSelector((store) => store.isLogged.isLogged);

      …
```

Допоміжні компоненти динамічного інтерфейсу знаходяться у директорії *src/components*. Ці компоненти використовуються для створення певних елементів сторінки чи підсторінок затосунку, в той час як компоненти-екрани виступають в ролі “контейнерів”.

Для збереження стану застосунку використовується Redux сховище – локальне сховище глобальних станів мобільного застосунку. Його конфігурація розташована у директорії *src/redux*. У сховищі зберігається інформація про те, чи увійшов користувач у систему під своїм обліковим записом. Ключ сховища, що відповідає за стан входу у систему – *isLogged*. Також зберігається токен користувача, якщо він авторизований у системі. Ключ сховища для цього – *token*. При початковому відкритті застосунку значення ключа isLogged є false, що означає, що користувач не авторизований, тому йому буде доступна функціональність Неавторизованого Користувача. Після успішної авторизації цей ключ приймає значення true, встановлюється ключ токену і таким чином буде доступна більша функціональність (функціональність Авторизованого користувача), а отже будуть відображатися відповідні компоненти Авторизованого користувача. Для їх завантаження використовується умовні конструкції. Приклад умовної конструкції зображений на лістингу 3.3. На ньому наведений фрагмент екрану перегляду детальної інформації про укриття. Авторизований користувач має можливість редагувати та додавати чи видаляти укриття з обраних.

Лістинг 3.3. Фрагмент екрану перегляду інформації про укриття

```
...
{isLogged && ( // УМОВНА КОНСТРУКЦІЯ ДЛЯ РОЗШИРЕННЯ ФУНКЦІОНАЛУ
  <View
    style={{
    flexDirection: "row",
    justifyContent: "space-between",
    alignItems: "center",
    width: "20%",
   }}
  >
    <Pressable
      onPress={() =>
      navigation.navigate("Editor", { shelter: shelter })
     }
    >
      <Icon name={"edit"} size={25} color={"#ee6c4d"} />
    </Pressable>
    <Pressable onPress={handleAddFavourite}>
      <Icon
        name={isFavourite ? "star" : "star-o"}
        size={25}
        color={"#ee6c4d"}
      />
    </Pressable>
  </View>
)}
      ...
```

Модуль клієнтського застосунку не зберігає даних користувача, а використовує ті дані, які зберігаються у віддаленій базі даних. Це робить його легким та динамічним і надає можливість мати завжди актуальні дані.

Для отримання даних від БД застосунок у відповідь на дії користувача робить відповідні запити до вебсервера – модуля алгоритмічної обробки, та при успішній відповіді повертає користувачеві отримані дані, інтегровані в інтерфейс. Ці запити виконуються клієнтом за допомогою *Fetch API* запитів. Це асинхронний інтерфейс, що дозволяє виконувати HTTP запити та отримувати відповіді, оброблювати їх у потрібний формат одразу після отримання. В клієнтському застосунку відповіді вебсервера обробляються у JSON формат. Це дозволяє їх використання для наповнення інтерфейсу даними. Приклад запиту на отримання користувача наведений у лістингу 3.4.

Лістинг 3.4. Фрагмент коду сторінки авторизації

```
...
const handleLogin = async () => {
  var iserr = false;
  if (useremail === "" || pass === "") {
    iserr = true;
    setIsErrorness(() => {
      return true;
    });
  } else {
      creds = {
         email: useremail,
         password: pass,
      };
  var response = await fetch("http://10.0.2.2:4567/user/login", { //
ЗАПИТ АВТОРИЗАЦІЇ КОРИСТУВАЧА
    method: "POST",
    headers: {
      "Content-Type": "application/json",
    },
    body: JSON.stringify(creds),
  });


  data = await response.json();
  if (data.token == undefined) {
    iserr = true;
    setIsErrorness(() => {
      return true;
    });
    setErrorMsg(() => {
      return data.message;
    });
  }
}


  if (!iserr) dispatch(login(data.token));
};
...
```

Для визначення координат користувача застосунок використовує методи *requestForegroundPermissionsAsync* та *getCurrentPositionAsync* з бібліотеки expo-location. Ці методи дозволяють отримати дозвіл на отримання місцезнаходження та власне отримати координати місцезнаходження користувача.

Крім отримання даних клієнтський застосунок також відправляє дані користувача стосовно нового укриття чи зміни інформації про наявне укриття на сервер. Ці дані перш ніж зберегтися у базі даних мають пройти

модерацію Адміністратора, тому для їхнього оброблення пропонується створити сервер *Kafka* – брокера повідомлень. Клієнтський застосунок виступає в ролі сервісу, що створює повідомлення у черзі повідомлень Kafka, а вебсервер у свою чергу зчитує повідомлення з черги для їх обробки. Використовуючи такий підхід, при авторизації на панелі Адміністратора він матиме можливість переглянути запити, що надійшли від користувачів про неактуальність укриття чи розміщення нового. Це забезпечить цілісність та актуальність даних у застосунку.

3.3.2. *Модуль алгоритмічної обробки*

Вебсервер застосунку розроблений відповідно до технічного завдання мовою Python з використанням бібліотеки Flask. Задля комунікації з базою даних застосунку було використано бібліотеку SQLAlchemy. Конфігурація вебсервера задовольняє паттерн розробки MVC (Model, View, Controller). В якості View виступає клієнтський застосунок і представляє собою динамічний інтерфейс користувача, а інші компоненти паттерну цілком і повністю покриває модуль алгоритмічної обробки.

У файлі *main.py* міститься основна конфігурація вебсервера, а саме підключення модулів моделей його роботи та шляхи, за якими може здійснюватися запит. Цей файл визначає, які контролери відповідальні за той чи інший запит і переправляє обробку запиту до цих контролерів. Приклад конфігурації серверу наведений у лістингу 3.5.

Контролери – методи, які виконують основну бізнес-логіку вебсерверу, обробляють вхідні дані, надсилають відповідні запити до бази даних, обробляють результат запитів та повертають їх до користувацького інтерфейсу. Таким чином контролери змінюють компонент View – користувацький інтерфейс.

Лістинг 3.5. Конфігурація вебсервера (main.py)

```
from flask import Flask
from flask_sqlalchemy import SQLAlchemy
```

```
from routes.misc import blueprint as misc_blueprint
from routes.shelters import blueprint as shelters_blueprint
from routes.users import blueprint as users_blueprint
from models import base


def create_app():
 db = base.db
 app = Flask(__name__)
 app.config.from_object('config')
 db.init_app(app)
 return app


app = create_app()

# Register blueprints
app.register_blueprint(misc_blueprint, url_prefix='/misc')
app.register_blueprint(shelters_blueprint, url_prefix='/shelters')
app.register_blueprint(users_blueprint, url_prefix='/user')

@app.route("/")
def hello():

 return "Hello world"


if __name__ == "__main__":
 app.run(host='0.0.0.0', port=4567, debug=True)
```

Модуль *routes* відповідає за шляхи, наявні для надсилання запитів до вебсервера. Для кожного запиту на вебсервері існує окремий шлях, що вказує який саме контролер має обробити цей запит. До головного файлу вебсерверу ці шляхи підключаються у вигляді *blueprint*, які «розкруслюють» яким чином буде переходити запит до контролера. Приклад опису конфігурацій шляхів наведено у лістингу 3.6.

Модуль *controllers* відповідає за компонент Controller паттерну. Тут зосереджені контролери вебзапитів. Типова задача контролера – відправити запит до бази даних, обробити його і отриманий результат відправити назад у відповідь.

Лістинг 3.6. Налаштування шляхів для запитів на отримання укриття (routes/shelters.py)

```
from flask import Blueprint
from controllers.shelterController import *

blueprint = Blueprint('shelters', __name__)
```

```
blueprint.route('/', methods=['GET'])(index)
blueprint.route('/all', methods=['GET'])(shelters)
blueprint.route('/all/filters', methods=['GET'])(sheltersWithFilters)
blueprint.route('/<int:id>', methods=['GET'])(getShelterById)
blueprint.route('/add/<int:id>', methods=['POST'])(addShelter)
blueprint.route('/add', methods=['POST'])(addShelter)
blueprint.route('/del/<int:id>', methods=['POST'])(delShelter)
```

Контролери оперують даними, отриманими у відфільтрованому вигляді з сервера бази даних. Для їх коректної роботи було описано третій компонент паттерну – Model. Моделі, описані у файлі *models.py*, представляють собою опис відповідних таблиць бази даних. Вони описують таблиці як окремі типізовані об’єкти даних з відповідними полями, обмеженнями та зв’язками. Після цього дані моделі використовуються для роботи вебсервера з базою даних. Лістинг 3.7 містить опис моделі користувача.

Для роботи з БД вебсервер використовує бібліотеку SQLAlchemy. Дана бібліотека є ORM (Object Relational Mapping) бібліотекою, а отже комунікує з базою даних за допомогою описаних у її нотації моделей. Запит з використанням ORM є більш зрозумілий розробнику і оперує об’єктами, а не чистим SQL для доступу до даних.

Лістинг 3.7. Опис моделі користувача

```
from sqlalchemy import *
from sqlalchemy.orm import relationship

from models.base import Base
from models.shelter import shelter_user_association
from models.location import Location
# from models.favourites import Favourites


class User(Base):
 '''users'''
 __tablename__='users'
 id = Column(Integer,
 Sequence('users_id_seq'),
 primary_key=True,
 server_default=Sequence('users_id_seq').next_value(),
 autoincrement=True)
 email = Column(Text, nullable=False)
 password = Column(Text, nullable=False)
 displayName = Column(Text, nullable=False)
 locations = relationship("Location", backref='users',
```

```
passive_deletes=True)
 shelters = relationship('Shelter',
secondary=shelter_user_association)

 def __init__(self, email, password, display_name):
 self.email = email
 self.password = password
 self.displayName = display_name

 def as_dict(self):
 return {c.name: getattr(self, c.name) for c in
self.__table__.columns}
```

Крім контролерів, логіка також зосереджується у модулі *services.* Тут міститься додаткова логіка, а саме логіка обчислення найближчого укриття та логіка перевірки користувацького токену. Далі ці сервіси імпортуються до контролерів для їхнього використання при обробці запиту.

Пошук найближчого укриття здійснюється з використанням бібліотеки *scipy*. Вона містить модуль *spatial*, що спеціально призначений для обробки географічних даних. Модуль містить методи для побудови k-вимірного дерева та методи з пошуку у ньому, зокрема і метод пошуку найближчого сусіда.

Як було зазначено у підрозділі 3.1, вебсервер виступає в ролі читача повідомлень з черги повідомлень Kafka для обробки даних про створення нового укриття чи зміну інформації про вже існуюче укриття. Для отримання повідомлень використовується бібліотека kafka-python. При кожному вході Адміністратора на вебпанель сервер збирає всі повідомлення з черги та виводить, форматує у JSON формат та надає їх Адміністратору на його вебпанель для подальшого збереження чи відхилення запитів.

3.4. Організація взаємодії модулів програмного забезпечення

Для візуального представлення конфігурації системи програмного забезпечення було розроблено діаграму розгортання у нотації UML. Діаграма розгортання наведена на рис. 3.3.

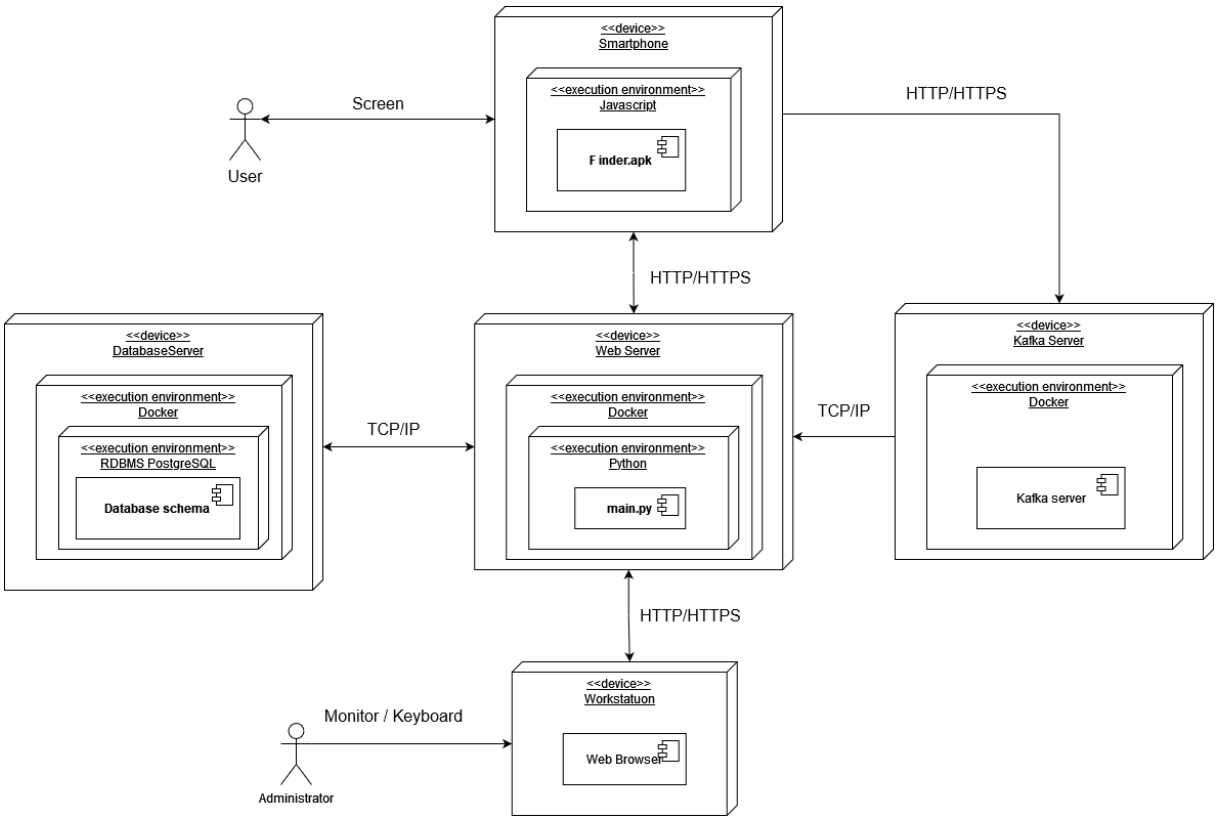


Рис. 3.3. Діаграма розгортання у нотації UML

Діаграма розгортання – це окремий вид діаграм, що супроводжують розробку програмного забезпечення. Вони дозволяють зобразити фізичну конфігурацію системи, а також протоколи взаємодії її модулів.

В розробленій системі існують такі компоненти:

- Smartphone – пристрій користувача, на якому виконується скомпільований код мобільного застосунку. Користувач взаємодіє з пристроєм за допомогою стандартних засобів вводу/виводу. Модуль клієнтського застосунку розміщується на даному пристрої.
- WebServer – пристрій, на якому у захищеному середовищі Docker розміщується модуль алгоритмічної обробки (main.py). Комунікація з модулем з клієнтського застосунку відбувається за протоколом HTTP.
- DatabaseServer – пристрій із середовищем Docker, в якому розміщується сервер бази даних застосунку. Комунікація з модулем відбувається лише з модуля вебсерверу за протоколом TCP/IP.

- KafkaServer – пристрій з середовищем Docker, в якому виконується вихідний код Apache Kafka. Сервер виступає брокером повідомлень, тому комунікація з ним відбувається як зі сторони клієнтського застосунку за протоколом HTTP, так і зі сторони вебсерверу за протоколом TCP/IP.
- Workstation – пристрій для роботи Адміністратора в системі. Комунікація з вебсервером відбувається за протоколом HTTP, а взаємодія з пристроєм – через стандартні засоби вводу/виводу.

3.5. Висновки до розділу

Даний розділ пояснювальної записки був присвячений опису реалізації програмного засобу. Розділ містить архітектурний та логічний опис системи. Було обґрунтовано та описано архітектуру розроблюваної системи. Також розділ містить деталізований опис структури бази даних – були згадані всі таблиці і поля, вказано їх тип та призначення. Структура бази даних відповідає третій нормальній формі, а отже сама база даних вважається нормалізованою.

Крім деталізації структури в даному розділі було детально описано основні модулі програмного забезпечення, а саме модуль клієнтського застосунку та модуль алгоритмічної обробки даних (вебсервер). Для кожного модуля було вказано його основні принципи конфігурації, типи файлів, логіку зв’язків та методи роботи. До кожного модуля було наведено ряд лістингів, які наочно демонструють згадані в описі конфігурації.

На основі структури розробленої системи було розроблено діаграму розгортання. Ця діаграма демонструє конфігурацію системи та способи взаємодії модулів між собою.

4. АНАЛІЗ РОЗРОБЛЕНИХ ПРОГРАМНИХ ЗАСОБІВ

4.1. Опис інтерфейсу застосунку

Розроблений користувацький інтерфейс застосунку має в основі вкладковий дизайн, тобто кожен функціонально визначений компонент представлений у вигляді вкладки на головному екрані. Наявні вкладки зображені на рис. 4.1.

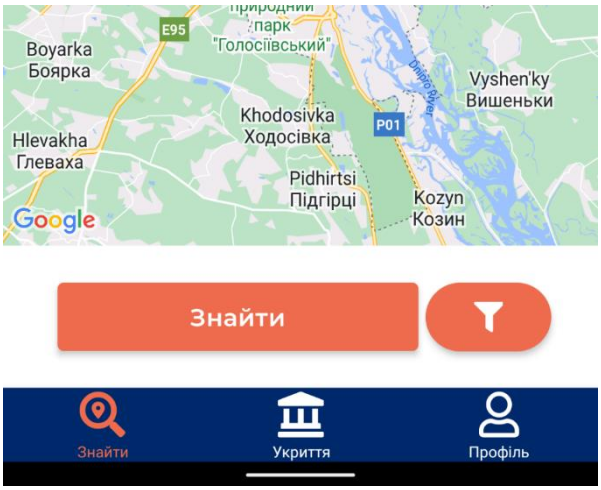


Рис. 4.1. Вкладкова навігація застосунку

Застосунок містить такі вкладки:

- *Знайти* – головне вікно застосунку. Основний функціонал з автоматичного пошуку розміщений на цій вкладці.
- *Укриття* – вкладка зі списком усіх доступних укриттів.
- *Профіль* – вкладка для авторизації та перегляду розширених функцій, доступних Авторизованому користувачеві.

Розглянемо детально інтерфейс кожної вкладки.

Вкладка *«Знайти»* відкривається за замовчуванням при кожному вході у застосунок. При першому вході користувача йому приходить запит на надання дозволу застосунку мати інформацію про поточне місцезнаходження користувача (рис. 4.2).

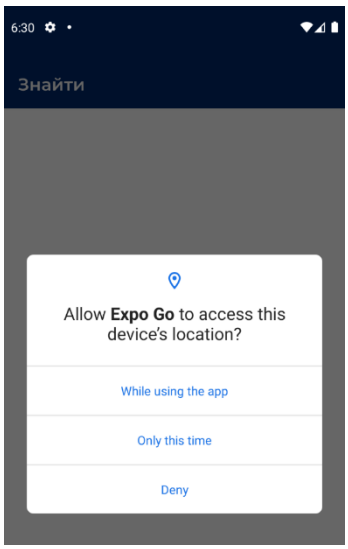


Рис. 4.2. Запит на доступ до служб локації

Після успішного підтвердження доступу до локації, застосунку потрібен деякий час для отримання даних. В цей час користувач бачить вікно завантаження, що свідчить про те, що місцезнаходження отримується.

Після успішного отримання геопозиції, користувачеві відображається інтерактивна карта Києва з його відміченим положенням на ній (рис. 4.3). Маркер користувача відображається синім кольором. Якщо користувач випадково перемістить карту так, що не бачитиме свій маркер, то в правому верхньому куті мапи розташовано кнопку, що повертає камеру карти таким чином, що положення користувача знаходиться посередині вікна. Якщо ж користувач бажає змінити масштаб мапи, то в правому нижньому куті знаходяться дві кнопки регулювання масштабу.

Для знаходження найближчого укриття та кількох наступних за близькістю укриттів, користувачеві необхідно натиснути на кнопку «*Знайти*». Після натискання на дану кнопку, на карті з'являються маркери результатів пошуку.

Найближче укриття позначається зеленим кольором, інші – червоним. При натисканні на кожне з укриттів над ним відкривається вікно примітки, в якій вказується адреса укриття, а також розрахована відстань від поточного місця розташування до нього (рис. 4.3).

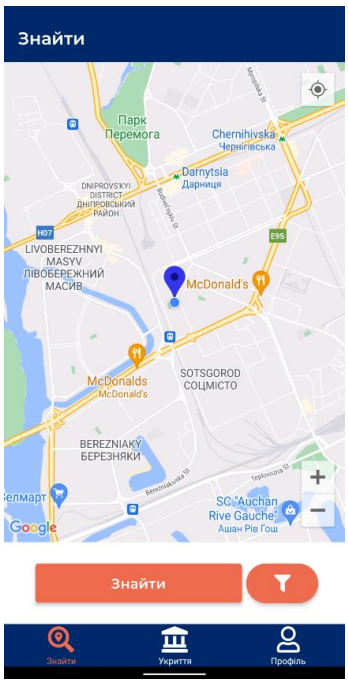


Рис. 4.3. Визначене місцезнаходження користувача

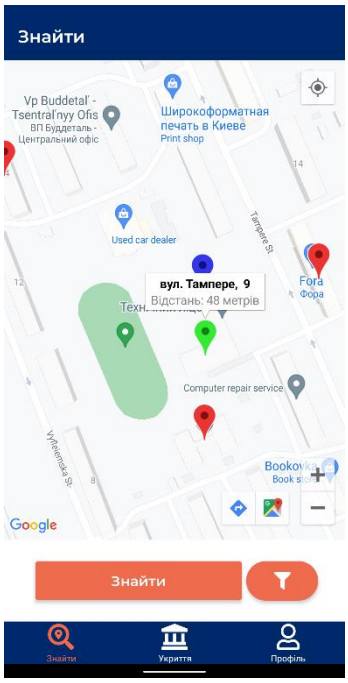


Рис. 4.4. Знайдені найближчі укриття

Для побудови маршруту користувач має обрати укриття з представлених, натиснути на його маркер на карті та натиснути на значок побудови маршруту у правому нижньому куті карти (рис. 4.5).

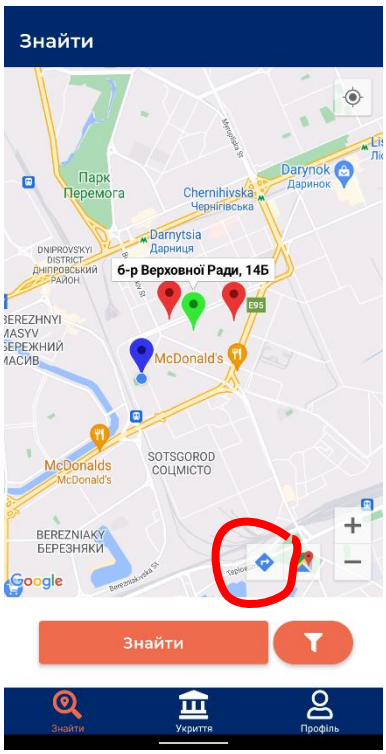


Рис. 4.5. Кнопка для побудови маршруту

Після натискання на даний значок, користувач перенаправляється на сервіс Google Maps, де знаходить побудований маршрут на карті (рис. 4.6).

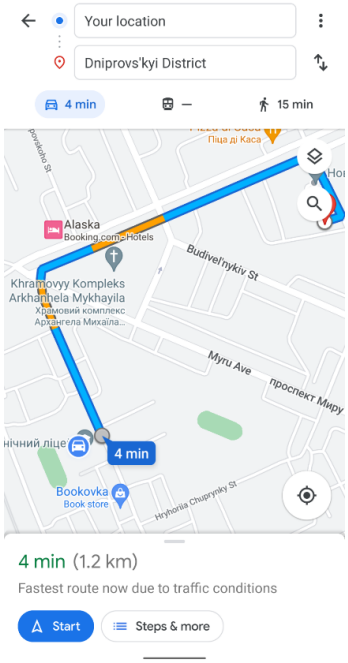


Рис. 4.6. Побудований маршрут

Також користувач може застосувати фільтри укриттів перед знаходженням найближчого. Для цього біля кнопки «Знайти» розташовується кнопка зі знаком фільтру, при натисканні на яку відкривається модальне вікно фільтрів з двома кнопками – «Відміна» та «Зберегти», а також перелік фільтрів з перемикачами, що дозволяють їх застосування (рис. 4.7).

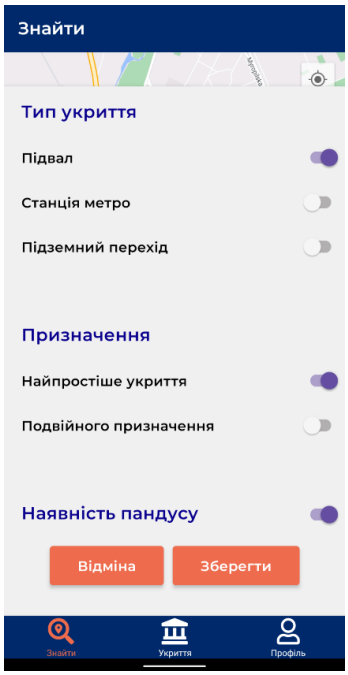


Рис. 4.7. Фільтри пошуку

При натисканні на кнопку «Зберегти» фільтри зберігаються та для їх застосування необхідно знову виконати пошук з основного екрану. При натисканні на кнопку «Відміна» усі фільтри скидаються.

Для Авторизованого користувача у правому верхньому куті екрана доступна кнопка «Зберегти поточне місце» (рис. 4.8). Після натискання на ць кнопку відкривається вікно збереження позиції користувача, яке потім можна буде знайти на вкладці «Профіль» – «Збережені місця».

Вкладка *“Укриття”* містить повний перелік доступних укриттів у вигляді списку (рис. 4.9).

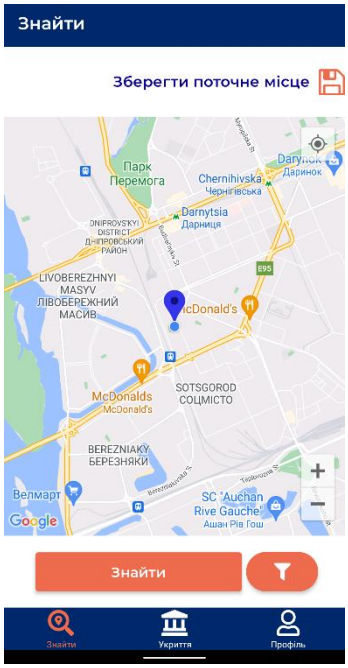


Рис. 4.8. Головний екран Авторизованого користувача

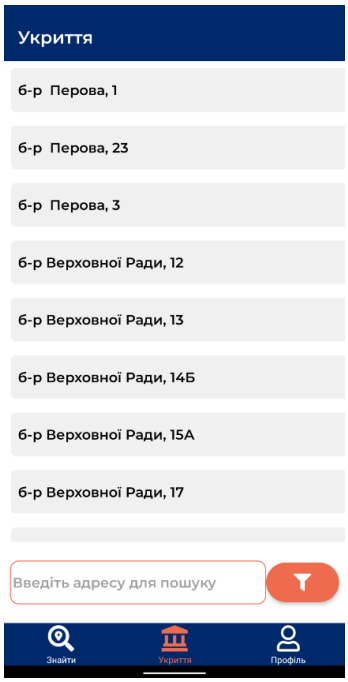


Рис. 4.9. Список всіх укриттів

Користувач може здійснити пошук за адресою, ввівши адресу чи частину адреси у поле для текстового пошуку і натиснувши клавішу Enter на екранній клавіатурі. Після цього результати пошуку будуть відображені

на цій же сторінці, а пошуковий запит зображений вгорі сторінки з можливістю відміни фільтрації (рис. 4.10).

Рис. 4.10. Фільтрація за адресою

Крім фільтрації за адресою, тут, так само як і на головному екрані, доступне модальне вікно фільтрів, які дозволяють пошук укриттів, що задовольняють ті фільтри. Пошук відбувається серед усіх наявних укриттів.

При натисканні на елемент списку, користувач переходить на сторінку з детальною інформацією про це укриття. Інформація представлена у вигляді таблиці з такими рядками:

- Адреса укриття.
- Тип.
- Призначення.
- Місткість.
- Наявність пандусу.

Крім текстової інформації користувач може побачити розташування укриття на карті.

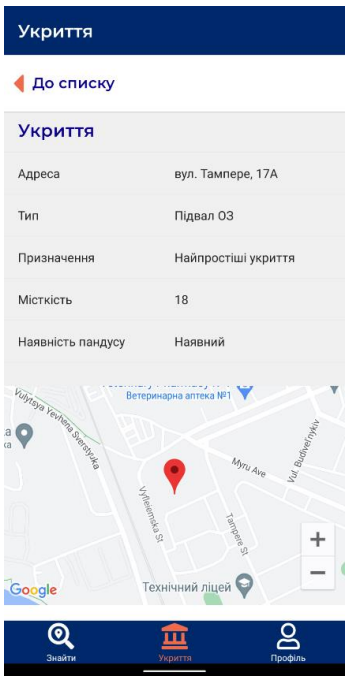


Рис. 4.11. Сторінка укриття

Якщо користувач авторизований, то з цієї сторінки він має можливість додати дане укриття у вибрані або зробити його редагування (рис. 4.12).

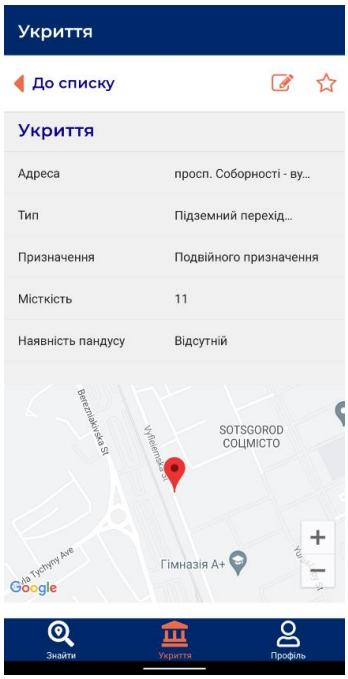


Рис. 4.12. Сторінка укриття для Авторизованого користувача

При натисканні на значок редагування Авторизованому користувачу відкривається вікно редагування (рис. 4.13). На цьому вікні є можливість

змінити текстові дані про укриття, а також змінити його розташування, натиснувши на місце на інтерактивній мапі. Після редагування для збереження змін необхідно натиснути на знак збереження у правому верхньому куті екрану. Після цього, користувач знову перейде на сторінку укриття і після модерації Адміністратора через певний час побачить внесені зміни.

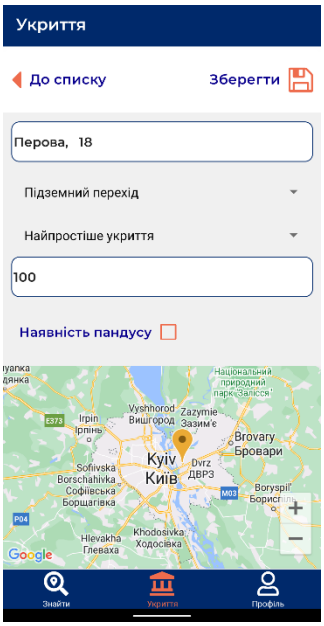


Рис. 4.13. Сторінка редагування

Вкладка *«Профіль»* – вкладка розширеного функціоналу Авторизованого користувача. Для Неавторизованого користувача тут міститься форма авторизації (рис. 4.14).

При відсутності облікових даних, користувачеві доступна проста реєстрація. Для її проведення, необхідно натиснути на кнопку «Реєстрація», після чого відкриється вікно реєстрації користувача (рис. 4.15). Після успішної реєстрації, користувач переправляється на форму авторизації і має ввести свої облікові дані для входу в систему.

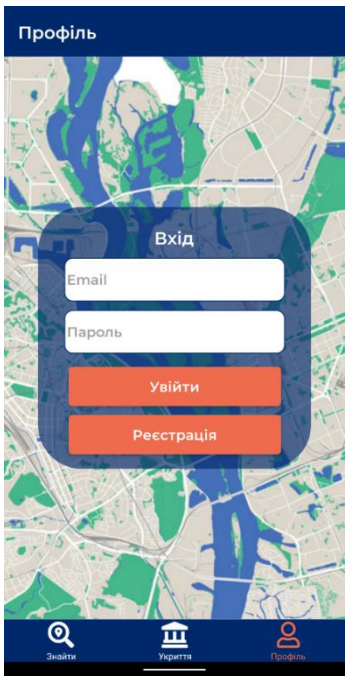


Рис. 4.14. Вкладка профілю з формою авторизації

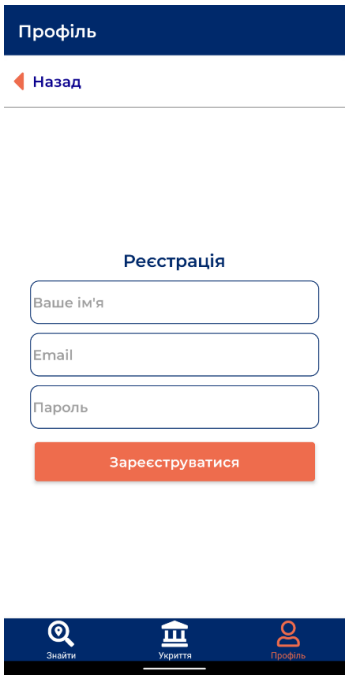


Рис. 4.15. Форма реєстрації

Після входу у систему, на місці форми авторизації у вкладці «Профіль» користувач бачить вікно свого профілю. Тут розміщується ім’я користувача, а також його розширені можливості у вигляді списку.

До опцій, доступних Авторизованому користувачеві належать:

- Обрані укриття.
- Збережені місця.
- Додати укриття.

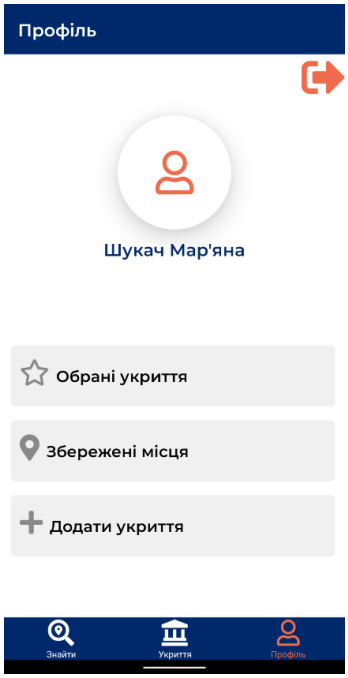


Рис. 4.16. Профіль користувача

Вкладка *«Обрані укриття»* відображає список обраних користувачем укриттів, аналогічний до списку всіх укриттів, лише без пошуку та фільтрації (рис. 4.17). При натисканні на елемент списку користувач так само бачить вікно з детальною інформацією про укриття та можливістю його редагування і вилучення з обраних.

Вкладка *«Збережені місця»* надає список збережених користувачем місць у вигляді списку (рис. 4.18).

При натисканні на кожне зі збережених місць, користувач переходить на сторінку місця, де розташована мапа та відмічене розташування збереженого місця (рис. 4.19).

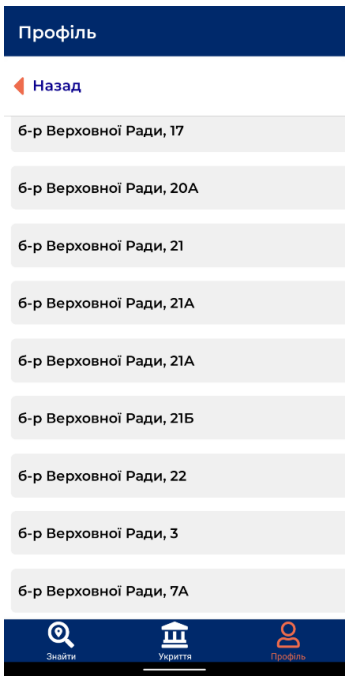


Рис. 4.17. Обрані укриття користувача

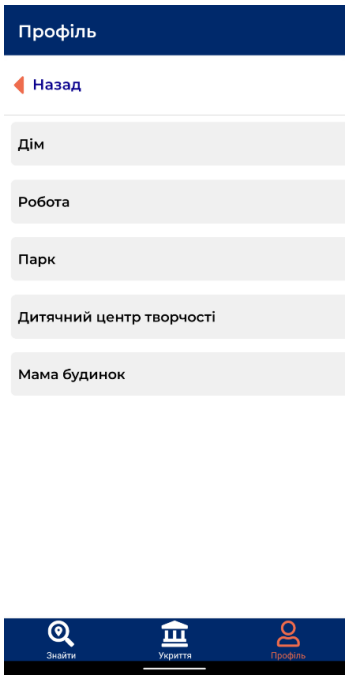


Рис. 4.18. Збережені місця користувача

Також користувачеві доступна кнопка «Знайти найближче», що знаходить найближче укриття до обраного місця (рис. 4.20). Аналогічно до основного пошуку, найближче укриття зображене зеленим маркером, всі інші – червоним. Для побудови маршруту треба вибрати укриття і натиснути на знак побудови в правому нижньому кутку мапи.

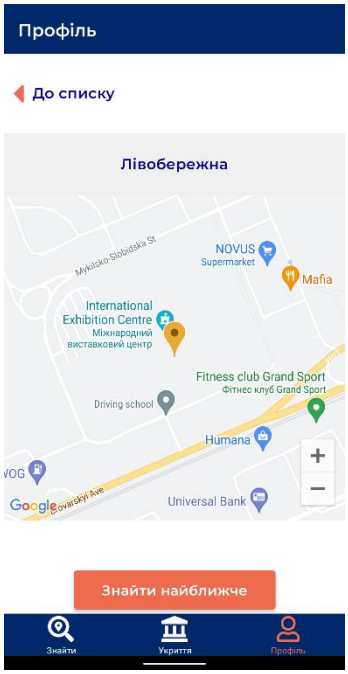


Рис. 4.19. Сторінка збереженого місця

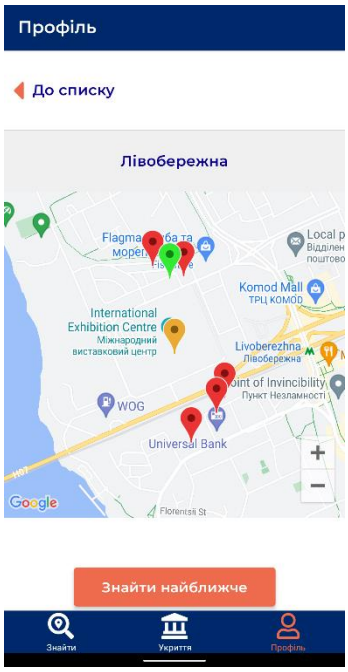


Рис. 4.20. Сторінка збереженого місця зі знайденими укриттями

Вкладка *«Додати укриття»* надає пусту форму для додавання укриття з такими полями:

- Адреса укриття.
- Тип.

- Призначення.
- Місткість.
- Наявність пандусу.

Розташування обирається натисканням на місце на мапі, де за інформацією користувача, знаходиться додане укриття. Після натискання на місце, на мапі з’являється маркер, який змінює положення зі зміною місця натискання (рис. 4.21).

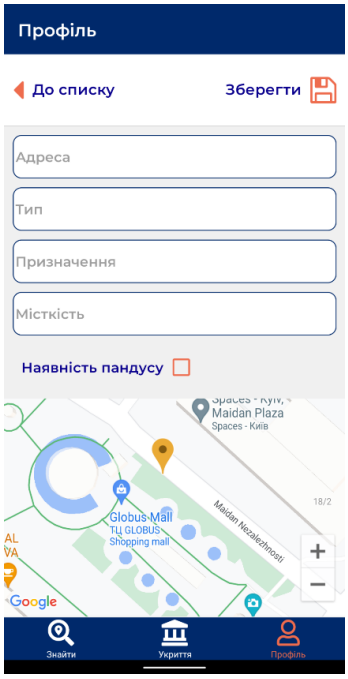


Рис. 4.21. Сторінка додавання укриття

Для збереження змін на надсилання запиту на модерацію Адміністратора, у правому верхньому куті розміщена кнопка збереження.

Таким чином, основний функціонал залишається доступним будь-якому користувачеві, а додаткові функції розміщені інтуїтивно зручно та вони є стандартизованими впродовж всього додатку.

4.2. Тестування програмного забезпечення

Тестування є невід’ємним етапом розробки програмного забезпечення, оскільки воно дозволяє перевірити систему на коректність роботи та виявити помилки на стадії розробки. Таким чином, тестування

дозволяє випускати якісний програмний продукт з меншою кількістю помилок.

Сучасна розробка програмного забезпечення обов'язково включає в себе етап тестування. Виділяють два основні види тестування:

- Мануальне – "ручне" тестування, здійснюється безпосередньо тестувальником;
- Автоматизоване – програмоване тестування, здійснюється за допомогою інструментів автоматизації.

Автоматизоване тестування, хоча і здається зручнішим, оскільки потребує меншого втручання тестувальника, не завжди здатне покрити усі тестові випадки, тому б більшості систем використовують комбінацію цих двох способів тестування.

Над розробленою системою проводилося мануальне димове тестування (Smoke Testing) [19] за методологією "чорного ящика" [20]. Тестові сценарії наведено у табл. 4.1 - 4.24.

Таблиця 4.1

Тестування встановлення застосунку

Номер сценарію	A11
Опис сценарію	Встановлення застосунку
Передумови	1. Наявність мобільного пристрою з ОС Android чи iOS. 2. Доступ у мережу WiFi чи мобільного оператора.
Послідовність дій сценарію	1. Завантажити .apk файл застосунку. 2. Відкрити .apk файл. 3. Виконати вказівки встановника.
Очікуваний результат	Віконка додатку розташована на головному екрані пристрою.

Таблиця 4.2

Тестування запуску застосунку

Номер сценарію	А12
Опис сценарію	Запуск застосунку
Передумови	1. Виконано сценарій А11. 2. Доступ у мережу WiFi чи мобільного оператора.
Послідовність дій сценарію	1. Натиснути на іконку застосунку.
Очікуваний результат	Застосунок відкривається на вкладці головного екрану, відкривається вікно запиту на отримання геопозиції пристрою додатком.

Таблиця 4.3

Тестування надання доступу до геопозиції

Номер сценарію	Б11
Опис сценарію	Надання доступу до геопозиції
Передумови	1. Виконано сценарій А12. 2. Доступ у мережу WiFi чи мобільного оператора.
Послідовність дій сценарію	1. Надати дозвіл на отримання геопозиції;
Очікуваний результат	Вікно запиту закривається, відкрита вкладка “Знайти” зі завантаженою мапою і позначкою місця розташування пристрою у вигляді синього маркеру.

Таблиця 4.4

Тестування відмови надання доступу до геопозиції

Номер сценарію	Б12
Опис сценарію	Відмова у доступі до геопозиції
Передумови	1. Виконано сценарій А12. 2. Доступ у мережу WiFi чи мобільного оператора.
Послідовність дій сценарію	1. Відмовити у дозволі на отримання геопозиції;
Очікуваний результат	Вікно запиту закривається, відкрита вкладка “Знайти” з повідомленням про необхідність надати дозвіл.

Таблиця 4.5

Тестування знаходження найближчих укриттів

Номер сценарію	В11
Опис сценарію	Знайти найближчі укриття
Передумови	1. Виконано сценарій Б11 2. Доступ у мережу WiFi чи мобільного оператора.
Послідовність дій сценарію	1. Натиснути на кнопку “Знайти”.
Очікуваний результат	На мапі з’являються 5 найближчих укриттів, найближче з яких позначено зеленим маркером, інші – червоним.

Таблиця 4.6

Тестування побудови маршруту

Номер сценарію	В12
Опис сценарію	Побудувати маршрут до найближчого укриття
Передумови	1. Виконано сценарій В11. 2. Доступ у мережу WiFi чи мобільного оператора.
Послідовність дій сценарію	1. Натиснути на зелений маркер. 2. У правому нижньому куті мапи натиснути на знак маршруту.
Очікуваний результат	Відкривається застосунок Google Maps з побудованим маршрутом.

Таблиця 4.7

Тестування відкриття вікна фільтрів

Номер сценарію	В13
Опис сценарію	Відкрити вікно фільтрів
Передумови	1. Виконано сценарій Б11. 2. Доступ у мережу WiFi чи мобільного оператора.
Послідовність дій сценарію	1. Натиснути на знак фільтру справа від кнопки “Знайти”.
Очікуваний результат	Відкривається модальне вікно з фільтрами.

Таблиця 4.8

Тестування фільтрації за одним параметром

Номер сценарію	B14
Опис сценарію	Здійснити фільтрацію за типом
Передумови	1. Виконано сценарій B13. 2. Доступ у мережу WiFi чи мобільного оператора. 3. Тип для фільтрації: станція метро.
Послідовність дій сценарію	1. Увімкнути фільтр “Станція метро” натиснувши на перемикач біля нього. 2. Натиснути кнопку “Зберегти”. 3. На відкритому головному вікні натиснути кнопку “Знайти”.
Очікуваний результат	На мапі з’являються 5 найближчих укриттів, найближче з яких позначено зеленим маркером, інші – червоним.

Таблиця 4.9

Тестування фільтрації за декількома параметрами

Номер сценарію	B15
Опис сценарію	Здійснити фільтрацію за типом та призначенням
Передумови	1. Виконано сценарій B14. 2. Доступ у мережу WiFi чи мобільного оператора. 3. Призначення для фільтрації: Подвійного призначення.
Послідовність дій сценарію	1. Увімкнути фільтр “Подвійного призначення” натиснувши на перемикач біля нього. 2. Натиснути кнопку “Зберегти”. 3. На відкритому головному вікні натиснути кнопку “Знайти”.

Продовження табл. 4.9

Очікуваний результат	На мапі з’являються 5 найближчих укриттів, найближче з яких позначено зеленим маркером, інші – червоним.

Таблиця 4.10

Тестування відміни фільтрації

Номер сценарію	В16
Опис сценарію	Відмінити всі фільтри
Передумови	1. Виконано сценарій В14 або В15. 2. Доступ у мережу WiFi чи мобільного оператора.
Послідовність дій сценарію	1. Натиснути кнопку “Відміна”. 2. На відкритому головному вікні натиснути кнопку фільтра.
Очікуваний результат	На відкритому вікні фільтрів всі перемикачі фільтрів вимкнуті.

Таблиця 4.11

Тестування перегляду всіх укриттів

Номер сценарію	Г11
Опис сценарію	Переглянути список всіх укриттів
Передумови	1. Виконано сценарій Б11. 2. Доступ у мережу WiFi чи мобільного оператора.
Послідовність дій сценарію	1. Натиснути на вкладку “Укриття”.
Очікуваний результат	Відкрита вкладка зі списком укриттів.

Таблиця 4.12

Тестування перегляду інформації про укриття

Номер сценарію	Г12
Опис сценарію	Переглянути інформацію про довільне укриття зі списку
Передумови	1. Виконано сценарій Г11. 2. Доступ у мережу WiFi чи мобільного оператора.
Послідовність дій сценарію	1. Натиснути на довільний елемент списку.
Очікуваний результат	Відкрита вкладка з детальною інформацією про укриття.

Таблиця 4.13

Тестування пошуку укриття за адресою

Номер сценарію	Г13
Опис сценарію	Здійснити пошук укриття за адресою
Передумови	1. Виконано сценарій Г11. 2. Доступ у мережу WiFi чи мобільного оператора.
Послідовність дій сценарію	1. Натиснути на поле для введення пошукового запиту. 2. У відкритій екранній клавіатурі ввести адресу. 3. Натиснути клавішу ввід/збереження на екранній клавіатурі.
Очікуваний результат	Над списком відображається пошуковий запит, список заповнено елементами, що задовольняють пошуковий запит.

Таблиця 4.14

Тестування відміни пошуку укриття за адресою

Номер сценарію	Г14
Опис сценарію	Відмінити фільтрацію за адресою
Передумови	1. Виконано сценарій Г13. 2. Доступ у мережу WiFi чи мобільного оператора.
Послідовність дій сценарію	1. Натиснути на знак закриття пошукового запиту.
Очікуваний результат	Пошуковий запит зникає, список укриттів відображається без фільтрації.

Таблиця 4.15

Тестування відкриття форми реєстрації

Номер сценарію	Д11
Опис сценарію	Відкрити форму реєстрації
Передумови	1. Виконано сценарій Б11. 2. Доступ у мережу WiFi чи мобільного оператора.
Послідовність дій сценарію	1. Відкрити вкладку “Профіль”. 2. Натиснути кнопку “Реєстрація”.
Очікуваний результат	Форма реєстрації відкрита.

Таблиця 4.16

Тестування реєстрації

Номер сценарію	Д12
Опис сценарію	Зареєструватися у системі
Передумови	1. Виконано сценарій Д11. 2. Доступ у мережу WiFi чи мобільного оператора.
Послідовність дій сценарію	1. Ввести нові облікові дані у поля вводу. 2. Натиснути кнопку “Зареєструватися”.
Очікуваний результат	Відкривається сторінка входу в систему.

Таблиця 4.17

Тестування входу в систему

Номер сценарію	Д12
Опис сценарію	Увійти в систему
Передумови	1. Виконано сценарій Б11. 2. Доступ у мережу WiFi чи мобільного оператора. 3. Користувач не авторизований у системі.
Послідовність дій сценарію	1. Перейти на вкладку “Профіль”. 2. Ввести облікові дані у поля вводу. 3. Натиснути кнопку “Вхід”.
Очікуваний результат	Відкривається сторінка профілю з введеним іменем на кроці 2 сценарію.

Таблиця 4.18

Тестування входу в систему

Номер сценарію	Д13
Опис сценарію	Переглянути обрані укриття
Передумови	1. Виконано сценарій Д12. 2. Доступ у мережу WiFi чи мобільного оператора.
Послідовність дій сценарію	1. Перейти на вкладку “Профіль”. 2. Натиснути опцію “Обрані укриття”.
Очікуваний результат	Відкривається сторінка зі списком обраних укриттів.

Таблиця 4.19

Тестування перегляду збережених місць

Номер сценарію	Д14
Опис сценарію	Переглянути збережені місця
Передумови	1. Виконано сценарій Д12. 2. Доступ у мережу WiFi чи мобільного оператора.
Послідовність дій сценарію	1. Перейти на вкладку “Профіль”. 2. Натиснути опцію “Збережені місця”.
Очікуваний результат	Відкривається сторінка зі списком Збережених місць.

Таблиця 4.20

Тестування додавання укриття

Номер сценарію	Д15
Опис сценарію	Додати укриття
Передумови	1. Виконано сценарій Д12. 2. Доступ у мережу WiFi чи мобільного оператора.
Послідовність дій сценарію	1. Перейти на вкладку “Профіль”. 2. Натиснути опцію “Додати укриття”. 3. У відкритому вікні додавання ввести інформацію про нове укриття. 4. На мапі вікна додавання обрати місце розташування укриття, натиснувши на нього. 5. У правому верхньому куті натиснути кнопку “Зберегти”.
Очікуваний результат	Поля заповнюються, маркер розташування ставиться на мапі. Після натискання на “Зберегти” відкривається вкладка “Профіль”. Після модерації Адміністратора інформація буде додана.

Таблиця 4.21

Тестування додавання укриття в обрані

Номер сценарію	Д16
Опис сценарію	Додати укриття в обрані
Передумови	1. Виконано сценарії Д11 чи Д12 та Г11. 2. Доступ у мережу WiFi чи мобільного оператора.
Послідовність дій сценарію	1. Натиснути на знак “зірочка” у правому верхньому куті; 2. Виконати сценарій Д13.
Очікуваний результат	Обране укриття з’явилося у списку обраних.

Таблиця 4.22

Тестування редагування укриття

Номер сценарію	Д17
Опис сценарію	Редагувати укриття
Передумови	1. Виконано сценарії Д11 чи Д12 та Г11. 2. Доступ у мережу WiFi чи мобільного оператора.
Послідовність дій сценарію	1. Натиснути на знак “редагування” у правому верхньому куті. 2. Ввести оновлені дані укриття. 3. Натиснути на кнопку “Зберегти”.
Очікуваний результат	Відкривається вікно з інформацією про укриття. Після модерації Адміністратора інформація буде змінена.

Таблиця 4.23

Тестування збереження поточного місця розташування

Номер сценарію	Д18
Опис сценарію	Зберегти поточне місце
Передумови	1. Виконано сценарії Д11 чи Д12. 2. Доступ у мережу WiFi чи мобільного оператора.
Послідовність дій сценарію	1. Відкрити вкладку “Знайти”. 2. У правому верхньому куті натиснути на знак “Зберегти місце”. 3. У модальному вікні ввести назву місця. 4. Виконати сценарій Д14.
Очікуваний результат	Ново збережене місце відображене у списку збережених місць.

Таблиця 4.24

Тестування виходу з системи

Номер сценарію	Д19
Опис сценарію	Вийти з системи
Передумови	1. Виконано сценарії Д11 чи Д12. 2. Доступ у мережу WiFi чи мобільного оператора.
Послідовність дій сценарію	1. Натиснути на знак “Вихід” у правому верхньому куті.
Очікуваний результат	Відкривається вікно з формою входу у систему. Розширені функції недоступні.

Проведене тестування повністю покриває функції застосунку. Більшість тестових сценарії пройшло успішно, помилки, виявлені при тестуванні були виправлені.

4.3. Рекомендації щодо подальшого вдосконалення

Система є доволі функціональною і надає широкий спектр опцій, які можна застосовувати не лише у безпековій сфері, а й у інших галузях. Наприклад, схожу систему можна імплементувати для пошуку найближчих громадських вбиралень, кафетеріїв, пунктів видачі води, тощо. Основними у даній системі є координати, тому предметна галузь може легко змінюватися. Задля цього треба модифікувати систему на більшу формалізованість, з великою кількістю шаблонів. Це допоможе легко пристосовувати функціональність до нових галузей.

Подальшим вдосконаленням є імплементація стабільного функціоналу Адміністратора, адже ця функція дозволить розширити можливості застосунку та імплементує відмінну від інших конкурентів необхідну функцію зворотнього зв’язку. Це також допоможе розширити

коло користувачів, адже вони матимуть можливість вносити власні дані у систему.

Одним з покращень було би можливість авторизації за допомогою стороннії провайдерів, наприклад Google чи Facebook. Це допомогло би скоротити час для авторизації у системі.

З безпекової точки зору є необхідність імплементації комунікації модулів по захищеному протоколу HTTPS. Для цього варто налаштувати SSL термінацію на вебсервері, та сервері БД. Це може убезпечити дані від викрадення їх зловмисником.

Для покращення подальшої розробки також варто налаштувати автоматизоване тестування, збирання програмного коду та його безперервну доставку (CI/CD). Це допоможе додавати нові функції швидше та стабільніше.

4.4. Висновки до розділу

Даний розділ присвячений аналізу розробленої системи. В рамках цього розділу було розглянуто та описано інтерфейс користувача застосунку. В результаті опису було зображено приклади роботи системи, її шляхи виконання та типові сценарії.

Також в даному розділі розміщено детальний опис проведеного тестування системи. Тестування проходило мануально. Всі тестові сценарії були описані та опрацьовані в процесі тестування. Це дозволило визначити слабкі місця системи та виправити їх.

В даному розділі також було наведено перелік можливих покращень та планів подальшого розвитку проєкту. Адже проєкт є доволі актуальним і при розширенні його функціоналу та збільшенні стабільності роботи, його застосування у повсякденному житті кожного громадянина є незамінним.

ВИСНОВКИ

Мета даного дипломного проєкту полягала у розробці мобільного застосунку для вибору та побудови маршруту до найближчого укриття відповідно до місця розташування користувача.

В процесі розробки було проведено порівняльний аналіз наявних рішень. Результати порівняльного аналізу показали, що серед існуючих рішень загальними проблемами є недостатність автоматизації пошуку укриття, мала інформативність та обмеженість функціоналу, зокрема фільтрації. На основі висновків аналізу було сформульовано основні функціональні та нефункціональні вимоги до системи.

Крім аналізу рішень також було проведено аналіз засобів розробки, що дозволило визначити ефективні інструменти, які забезпечують стабільність системи застосунку та її швидкодію. Окрім цього було виконано дослідження математичного підґрунтя пошуку найближчого укриття. Для виконання пошуку було обрано модель пошуку найближчого сусіда та її реалізацію шляхом використання алгоритмів на графах, зокрема алгоритму на основі kd-дерева та алгоритму Дейкстри. Для виконання фільтрації було обрано алгоритм пошуку у геш-таблиці. Ці алгоритми показали себе ефективними та швидкими для виконання основних завдань системи.

Розроблений застосунок забезпечує легкий та ефективний пошук найближчого укриття. Крім звичайного пошуку система надає користувачеві ряд можливостей та переваг, а саме:

- можливість пошуку укриття за адресою;
- можливість пошуку найближчого укриття з фільтрацією;
- дозволяє збереження обраних укриттів чи місць користувача;
- має форму для додавання чи редагування укриття;
- забезпечує зручний інтерфейс з наочно зрозумілими елементами.

Під час розробки особлива увага була приділена інтерфейсу користувача та його зручності. Було розроблено ефективний інтерфейс з мінімальною кількістю кроків для отримання результату пошуку. Також було проведено ряд тестувань системи відповідно до затвердженої програми та методики тестування. Всі наявні недоліки та помилки були відредаговані задля забезпечення якості та стабільності роботи системи.

Розроблена система задовольняє основні вимоги та виконана у повному обсязі.

Застосунок має широку аудиторію користувачів. Він підійде для користування будь-якому мешканцю чи гостю столиці. Застосунок може бути встановлений на найбільш поширені операційні системи, оскільки розроблений за допомогою інструментів кросплатформної розробки. Він не має обмежень у віці, легкий та зрозумілий.

Мобільний застосунок має широкий потенціал для розширення своєї функціональності. Також розроблена система може бути використана для розробки інших застосунків з побудови маршрутів та пошуків найближчих географічних місць.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Мапа укриттів Києва [Електронний ресурс] – Режим доступу до ресурсу: https://kyivcity.gov.ua/news/karta_ukrittiv_kiyeva/.
2. MoSCoW method [Електронний ресурс] // Wikipedia – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/MoSCoW_method.
3. Pathfinding [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Pathfinding.
4. Google Maps–it's just one big graph [Електронний ресурс] // Cornell University. – 2011. – Режим доступу до ресурсу: https://blogs.cornell.edu/info2040/2011/09/14/google-maps-its-just-one-big-graph/.
5. Crovari P. Google Maps and graph theory [Електронний ресурс] / Pietro Crovari. – 2019. – Режим доступу до ресурсу: https://magazine.impactscool.com/en/speciali/google-maps-e-la-teoria-dei-grafi/.
6. Dijkstra's algorithm [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm.
7. Introduction to Algorithms / T. H.Cormen, C. E. Leiserson, R. L. Rivest, C. Stein. – London, England: The MIT Press, 2022. – 1290 с. – (Massachusetts Institute of Technology).
8. Schütz B. Partition-Based Speed-Up of Dijkstra's Algorithm [Електронний ресурс] / Birk Schütz. – 2004. – Режим доступу до ресурсу: https://i11www.iti.kit.edu/_media/teaching/theses/files/studienarbeit-schuetz-05.pdf.
9. k-d Tree [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/K-d_tree.
10. Friedman J. H. An algorithm for finding best matches in logarithmic expected time [Електронний ресурс] / J. H. Friedman, J. L. Bentley, R. A.

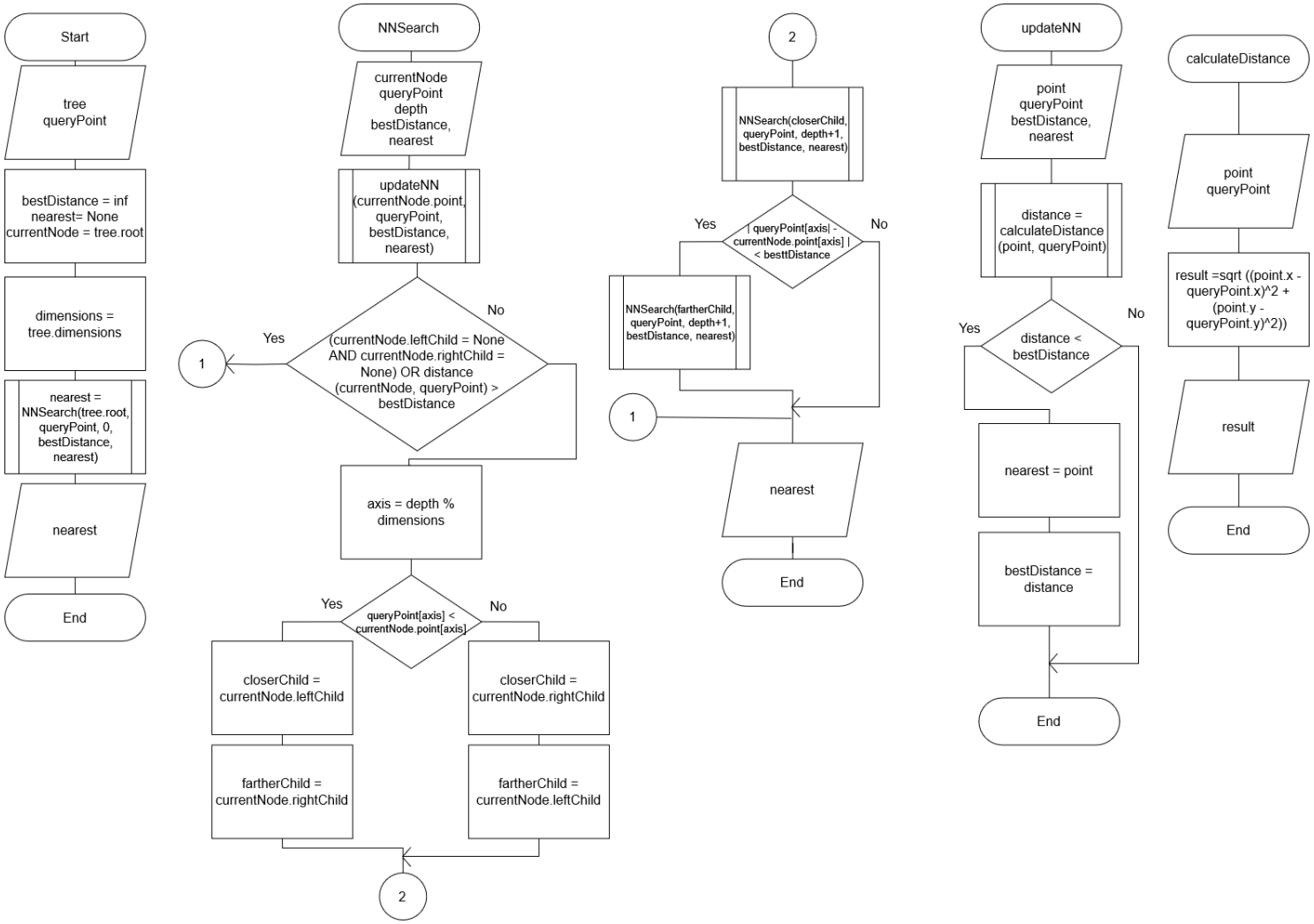
Finkel // Stanford University. – 1976. – Режим доступу до ресурсу: https://inspirehep.net/files/3c9ac14423f008f75b237d8b95b68777.

11. Документація PostgreSQL [Електронний ресурс] – Режим доступу до ресурсу: https://www.postgresql.org/docs/.
12. TIOBE index of May 2023 [Електронний ресурс] – Режим доступу до ресурсу: https://www.tiobe.com/tiobe-index/.
13. Python [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Python.
14. Top 10 Features of Python You Need to Know [Електронний ресурс] // Edureka. – 2023. – Режим доступу до ресурсу: https://www.edureka.co/blog/python-features/.
15. React Native [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/React_Native.
16. React Native official page [Електронний ресурс] – Режим доступу до ресурсу: https://reactnative.dev/.
17. Документація React Native: Setting up the development environment [Електронний ресурс] – Режим доступу до ресурсу: https://reactnative.dev/docs/environment-setup.
18. PostgreSQL About [Електронний ресурс] – Режим доступу до ресурсу: https://www.postgresql.org/about/.
19. Smoke testing (software) [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Smoke_testing_(software).
20. Black-Box testing [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Black-box_testing.

ДОДАТКИ

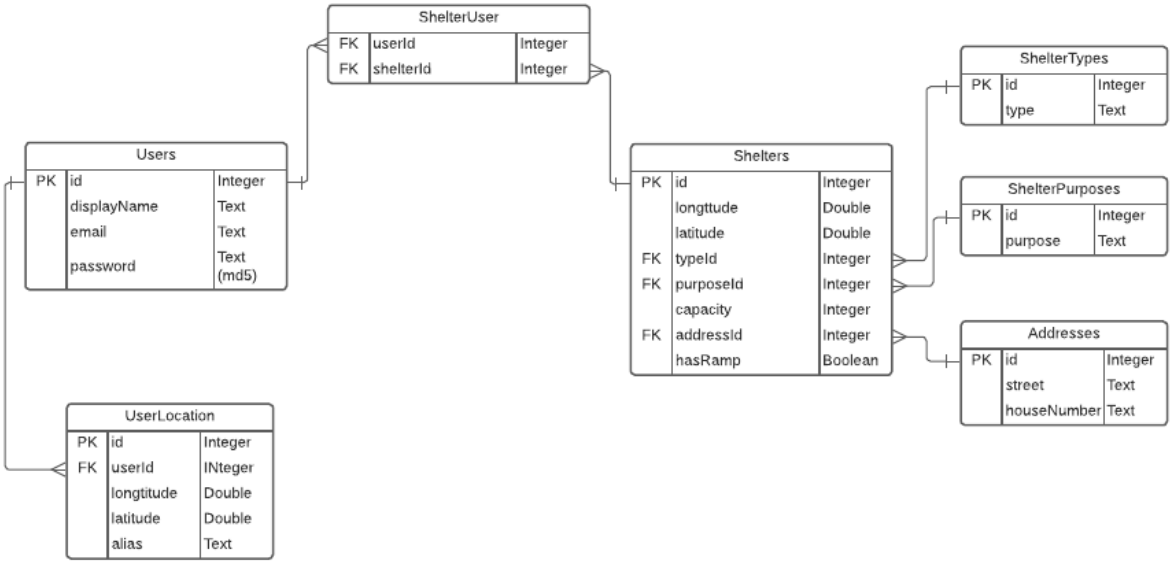
Додаток 1

Копії графічних матеріалів

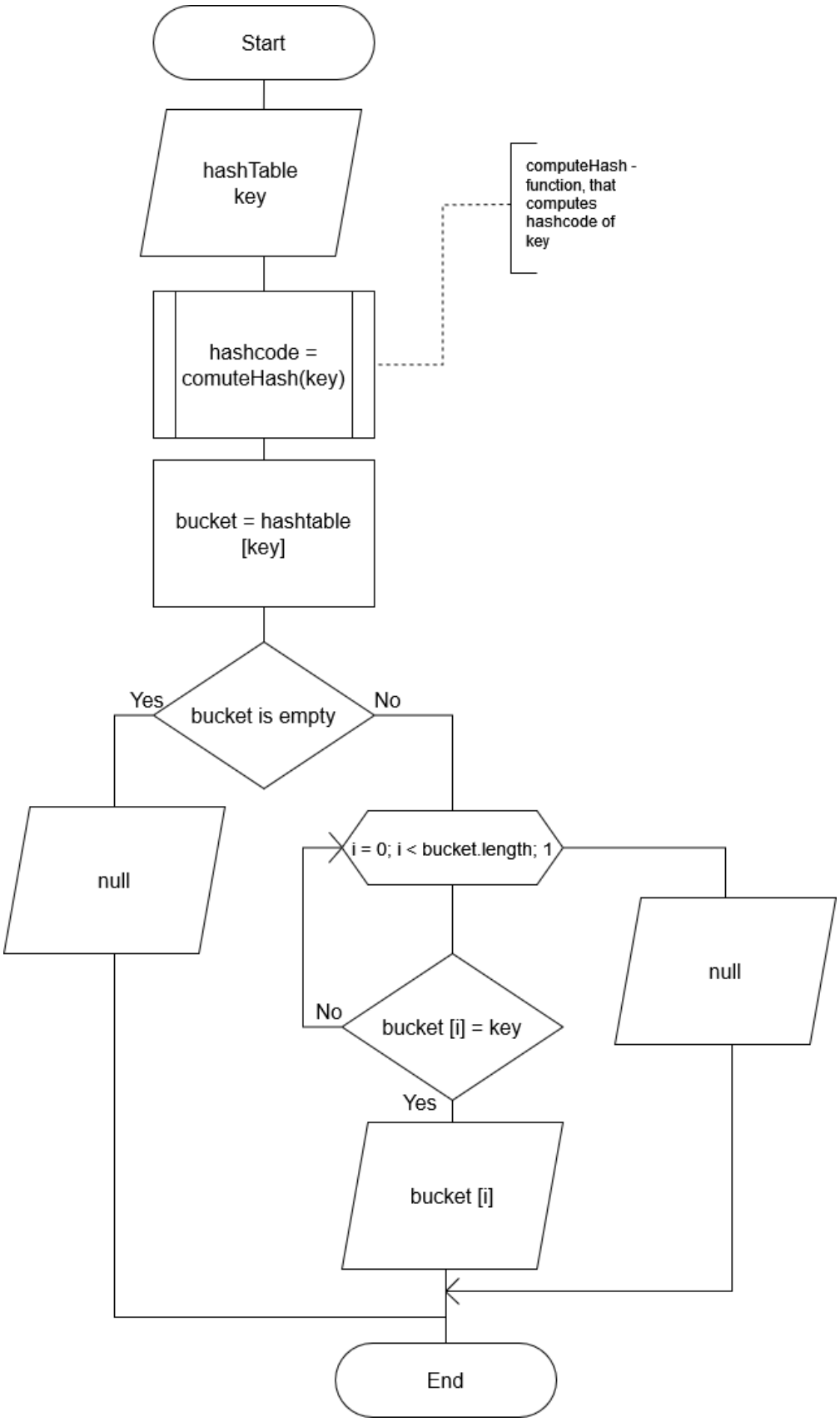


ДП.045490-06-99

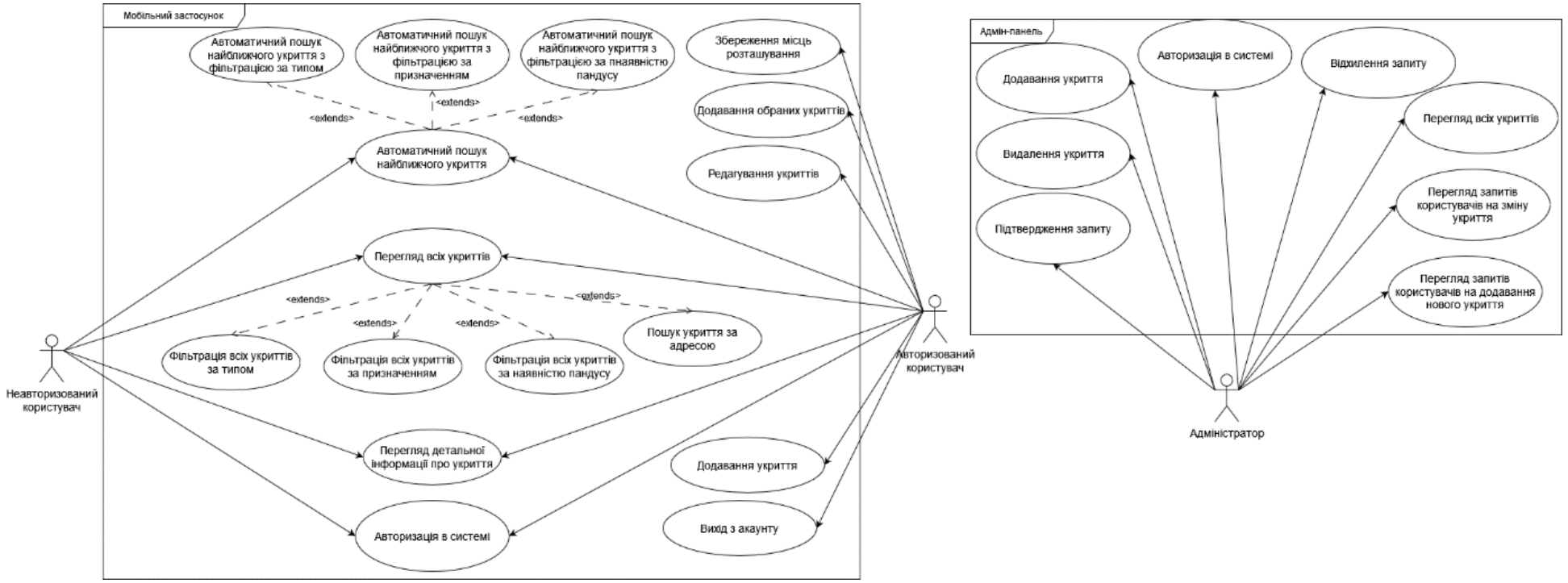
Мобільний застосунок для знаходження найближчого укриття. Алгоритм пошуку найближчого укриття. Блок-схема алгоритму



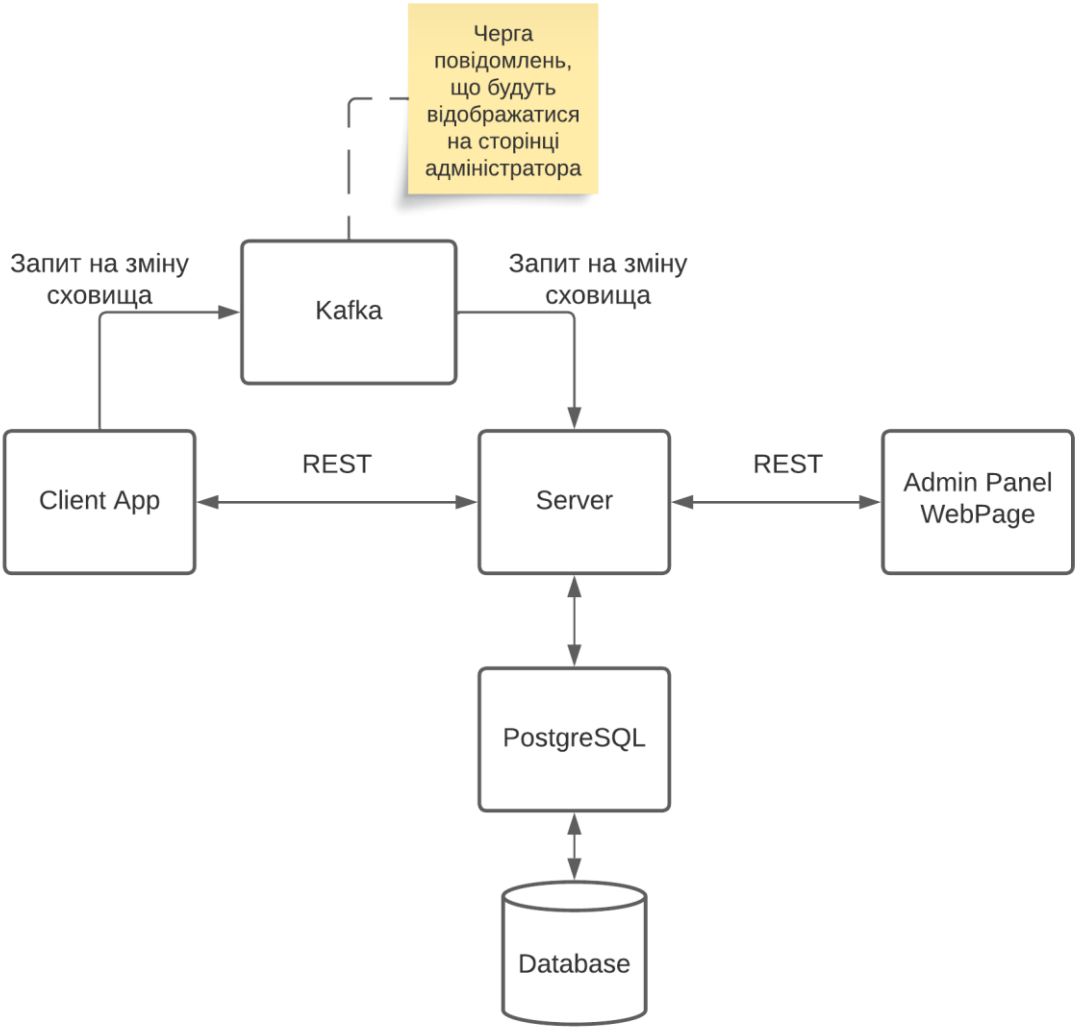
ДП.045490-07-99
Мобільний застосунок для знаходження найближчого укриття. Структура бази даних. ER-діаграма



ДП.045490-08-99
Мобільний застосунок для знаходження найближчого укриття. Алгоритм фільтрації. Блок-схема алгоритму



Діаграма прецедентів у нотації UML
Вергун М.О., група КП-92



Архітектура системи
Вергун М.О., група КП-92

Додаток 2

Лістинг програми

Фрагменти коду головної сторінки мобільного застосунку

```
import {
  StyleSheet,
  View,
  Platform,
  ActivityIndicator,
  Pressable,
  Text,
} from "react-native";
import { createNativeStackNavigator } from "@react-navigation/native-
stack";
import * as Location from "expo-location";
import MapView from "react-native-maps";
import { Marker } from "react-native-maps";
import { useSelector } from "react-redux";
import Icon from "react-native-vector-icons/FontAwesome";

// Import components
import { Button } from "../components/MainButton";
import { FilterButton } from "../components/FilterButton";
import { Filters } from "../components/FilterComponent";
import { useEffect, useState } from "react";
import { LocAdder } from "../components/AddLocation";

export function HomeScreen() {
  const [errorMsg, setErrorMsg] = useState(null);
  const [location, setLocation] = useState(null);
  const isLogged = useSelector((store) => store.isLogged.isLogged);

  useEffect(() => {
    (async () => {
      var { status } = await Location.requestForegroundPermissionsAsync();
      if (status !== "granted") {
        setErrorMsg("Permission to access location was denied");
        return;
      }
      var location = await Location.getCurrentPositionAsync({
        accuracy: Location.Accuracy.Highest,
        maximumAge: 100,
      });
      setLocation(() => {
        return {
          latitude: location.coords.latitude,
          longitude: location.coords.longitude,
        };
      });
    })();
  }, []);

  function MainScreen({ navigation, route }) {
    const [shelters, setShelters] = useState([]);
    const [isFound, setIsFound] = useState(false);
    const [typesFilters, setTypesFilters] = useState(route.params.types);
    const [purposesFilters, setPurposesFilters] = useState(
      route.params.purposes
    );
    const [hasRampFilter, setHasRampFilter] =
useState(route.params.hasRamp);

    function SheltersList({ navigation }) {
      if (location) {
        var markers = shelters || [];
        return (
```

```
          <View style={styles.mapContainer}>
            <MapView
              style={styles.map}
              initialRegion={{
                latitude: location.latitude,
                longitude: location.longitude,
                latitudeDelta: 0.05,
                longitudeDelta: 0.05,
              }}
              loadingEnabled={true}
              loadingIndicatorColor="#666666"
              loadingBackgroundColor="#eeeeee"
              moveOnMarkerPress={true}
              showsUserLocation={true}
              showsCompass={true}
              showsPointsOfInterest={false}
              provider="google"
              zoomControlEnabled={true}
            >
              <Marker coordinate={location} pinColor={"blue"} title="Ви
тут" />

              {markers &&
                markers.map((shelter) => (
                  <Marker
                    key={shelter.id}
                    coordinate={{
                      latitude: parseFloat(shelter.latitude),
                      longitude: parseFloat(shelter.longitude),
                    }}
                    title={shelter.address}
                    description={
                      "Відстань: " + Math.round(shelter.distance) + "
метрів"
                    }
                    pinColor={shelters.indexOf(shelter) == 0 ? "green" :
"red"}
                  />
                ))}
            </MapView>
          </View>
        );
      } else if (errorMsg) {
        return (
          <View>
            <Text>Для пошуку необхідно надати дозвіл до геопозиції</Text>
          </View>
        );
      } else {
        return (
          <View>
            <ActivityIndicator size="large" color="#ee6c4d" />
          </View>
        );
      }
    }

    function handleFind() {
      if (
        (hasRampFilter === undefined ||
          hasRampFilter === false ||
          hasRampFilter === "") &&
        typesFilters.length <= 0 &&
        purposesFilters.length <= 0
```

```
      ) {
        fetch(
          "http://10.0.2.2:4567/misc/nearest?lat=" +
            location.latitude +
            "&lng=" +
            location.longitude
        )
          .then((response) => response.json())
          .then((json) => setShelters(json))
          .catch((err) => console.error(err))
          .finally(() => setIsFound(true));
      } else {
        var url =
          "http://10.0.2.2:4567/misc/nearest/filters?lat=" +
          location.latitude +
          "&lng=" +
          location.longitude;
        if (typesFilters.length > 0) {
          url = url.concat("&type=", typesFilters.join());
        }
        if (purposesFilters.length > 0) {
          url = url.concat("&purpose=", purposesFilters.join());
        }
        if (hasRampFilter === true) {
          url = url.concat("&hasRamp=", hasRampFilter);
        }
        fetch(url)
          .then((response) => response.json())
          .then((json) => setShelters(json))
          .catch((err) => console.error(err));
      }
    }

    handleSave = () => {
      console.log("save position");
      navigation.navigate("Saver", {
        alias: "",
        position: {
          longitude: location.longitude,
          latitude: location.latitude,
        },
      });
    };

    return (
      <View style={styles.container}>
        {isLogged && location && (
          <View style={styles.save}>
            <Pressable
              style={({ pressed }) =>
                pressed ? styles.pressablePressed : styles.pressable
              }
              onPress={handleSave}
            >
              <Text style={styles.headeringText}>Зберегти поточне
місце</Text>
              <Icon name="save" size={35} color={"#ee6c4d"} />
            </Pressable>
          </View>
        )}
        <View style={styles.listContainer}>
          <SheltersList />
        </View>
        {!errorMsg && (
```

```
          <View
            style={
              Platform.OS == "android"
                ? styles.buttonContainerAndroid
                : styles.buttonContainerIOS
            }
          >
            <View flex={3}>
              <Button title="Знайти" onPress={handleFind} />
            </View>
            <View flex={1}>
              <FilterButton
                onPress={() =>
                  navigation.navigate("Filters", { parentWin: "Results" })
                }
              ></FilterButton>
            </View>
          </View>
        )}
      </View>
    );
  }

  const RootStack = createNativeStackNavigator();

  return (
    <RootStack.Navigator screenOptions={{ headerShown: false }}>
      <RootStack.Group>
        <RootStack.Screen
          name="Results"
          component={MainScreen}
          initialParams={{
            types: [],
            purposes: [],
            hasRamp: "",
          }}
        />
      </RootStack.Group>
      <RootStack.Group
        screenOptions={{
          presentation: "transparentModal",
          animation: "fade_from_bottom",
        }}
      >
        <RootStack.Screen
          name="Filters"
          component={Filters}
          initialParams={{ parentWin: "Results" }}
        />
        <RootStack.Screen
          name="Saver"
          component={LocAdder}
          initialParams={{
            alias: "",
            position: {
              longitude: "",
              latitude: "",
            },
          }}
        />
      </RootStack.Group>
    </RootStack.Navigator>
  );
}
```

```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: "#ffffff",
  },
  listContainer: {
    flex: 6,
    flexDirection: "column",
    alignItems: "center",
    justifyContent: "center",
  },
  buttonContainerAndroid: {
    flex: 1,
    flexDirection: "row",
    alignItems: "center",
    justifyContent: "space-between",
    borderWidth: 1,
    borderColor: "#ffffff",
    paddingHorizontal: 30,
  },
  buttonContainerIOS: {
    flex: 1,
    flexDirection: "row",
    alignItems: "center",
    justifyContent: "space-between",
    borderWidth: 1,
    borderColor: "#ffffff",
    shadowRadius: 10,
    shadowColor: "#000",
    paddingHorizontal: 30,
  },
  mapContainer: {
    width: "100%",
    height: "100%",
  },
  map: {
    width: "100%",
    height: "100%",
  },
  headeringText: {
    padding: 10,
    fontSize: 17,
    lineHeight: 21,
    fontFamily: "Monserrat-SemiBold",
    letterSpacing: 0.25,
    color: "#09008e",
    // color: "#ee6c4d"
  },
  save: {
    flexDirection: "row",
    flex: 1,
    width: "100%",
    justifyContent: "flex-end",
  },
  pressable: {
    flexDirection: "row",
    justifyContent: "flex-start",
    alignItems: "center",
  },
  pressablePressed: {
    opacity: 0.5,
    flexDirection: "row",
    justifyContent: "flex-start",
```

```
    alignItems: "center",
  },
});
```

Фрагменти коду сторінки всіх укриттів мобільного застосунку

```
import { useCallback, useEffect, useMemo, useState } from "react";
import {
  StyleSheet,
  View,
  Text,
  RefreshControl,
  FlatList,
  TextInput,
  Pressable,
  ActivityIndicator,
} from "react-native";
import { createNativeStackNavigator } from "@react-navigation/native-
stack";
import Icon from "react-native-vector-icons/FontAwesome";

import { ShelterItem } from "./../components/ShelterItem";
import { Filters } from "../components/FilterComponent";
import { FilterButton } from "../components/FilterButton";
import { useSelector, useDispatch } from "react-redux";
import { Shelter } from "../components/ShelterComponent";
import { ShelterEdit } from "../components/ShelterEdit";
import { Adder } from "../components/AddShelter";

export function SheltersScreen({ navigation }) {
  const [shelters, setShelters] = useState([]);
  const [fileteredShelters, setFilteredShelters] = useState([]);
  const [search, setSearch] = useState("");

  const [typesFilters, setTypesFilters] = useState([]);
  const [purposesFilters, setPurposesFilters] = useState([]);
  const [hasRampFilter, setHasRampFilter] = useState("");

  const isLogged = useSelector((store) => store.isLogged.isLogged);
  const [refreshing, setRefreshing] = useState(false);

  async function searchFilter(text) {
    if (text) {
      const newShelters = shelters;
      const filtered = newShelters.filter((shelter) => {
        const shelterData = shelter.address
          ? shelter.address.toLowerCase()
          : "";
        const textData = text.toLowerCase();
        return shelterData.indexOf(textData) > -1;
      });
      setFilteredShelters(() => {
        return filtered;
      });
      setSearch(() => {
        return text;
      });
    } else {
      setFilteredShelters(shelters);
      setSearch(text);
    }
  }

  const RootStack = createNativeStackNavigator();

  const SearchAndFilters = () => {
    const [temp, setTemp] = useState("");
```

```
    return (
      <View style={styles.filterContainer}>
        <TextInput
          flex={5}
          style={styles.textInput}
          placeholder="Введіть адресу для пошуку"
          value={search.text}
          onChangeText={(text) => setTemp(text)}
          onSubmitEditing={() => searchFilter(temp)}
        />
        <FilterButton flex={1} onPress={() =>
navigation.navigate("Filters")} />
      </View>
    );
  };
  useEffect(() => {
    if (
      (hasRampFilter === undefined ||
        hasRampFilter === false ||
        hasRampFilter === "") &&
      typesFilters.length <= 0 &&
      purposesFilters.length <= 0
    ) {
      fetch("http://10.0.2.2:4567/shelters/all")
        .then((response) => response.json())
        .then((json) => {
          setRefreshing(() => {
            return false;
          });
          setFilteredShelters(json);
          setShelters(json);
        })
        .catch((err) => console.error(err));
    } else {
      var url = "http://10.0.2.2:4567/shelters/all/filters?";
      if (typesFilters.length > 0) {
        url = url.concat("&type=", typesFilters.join());
      }
      if (purposesFilters.length > 0) {
        url = url.concat("&purpose=", purposesFilters.join());
      }
      if (hasRampFilter === true) {
        url = url.concat("&hasRamp=", hasRampFilter);
      }
      fetch(url)
        .then((response) => response.json())
        .then((json) => {
          setRefreshing(() => {
            return false;
          });
          setFilteredShelters(json);
        })
        .catch((err) => console.error(err));
    }
  }, [typesFilters, purposesFilters, hasRampFilter]);

  refresh = useCallback( async () => {
    if (
      (hasRampFilter === undefined ||
        hasRampFilter === false ||
        hasRampFilter === "") &&
      typesFilters.length <= 0 &&
      purposesFilters.length <= 0
    ) {
```

```
    fetch("http://10.0.2.2:4567/shelters/all")
      .then((response) => response.json())
      .then((json) => {
        setRefreshing(false);
        setFilteredShelters(json);
        setShelters(json);
      })
      .catch((err) => console.error(err));
  } else {
    var url = "http://10.0.2.2:4567/shelters/all/filters?";
    if (typesFilters.length > 0) {
      url = url.concat("&type=", typesFilters.join());
    }
    if (purposesFilters.length > 0) {
      url = url.concat("&purpose=", purposesFilters.join());
    }
    if (hasRampFilter === true) {
      url = url.concat("&hasRamp=", hasRampFilter);
    }
    fetch(url)
      .then((response) => response.json())
      .then((json) => {
        setRefreshing(false);
        setFilteredShelters(json);
      })
      .catch((err) => console.error(err));
  }
}, [refreshing]);

const ShelterFlatList = () => {
  return (
    <View style={styles.container}>
      <FlatList
        data={fileteredShelters}
        renderItem={(itemData) => {
          return (
            <ShelterItem
              text={itemData.item.address}
              onPress={() =>
                navigation.navigate("Shelter", {
                  id: itemData.item.id,
                })
              }
            />
          );
        }}
        refreshControl={
          <RefreshControl refreshing={refreshing} onRefresh={refresh} />
        }
      />
    </View>
  );
};

function SheltersList({ navigation, route }) {
  const [types, setTypes] = useState(route.params.types);
  const [purposes, setPurposes] = useState(route.params.purposes);
  const [hasRamp, setHasRamp] = useState(route.params.hasRamp);

  useEffect(() => {
    setTypesFilters(types);
    setPurposesFilters(purposes);
    setHasRampFilter(hasRamp);
  }, [route.params.types, route.params.purposes, route.params.hasRamp]);
```

```
    return (
      <View style={styles.container}>
        {search && (
          <View style={styles.searchLabel}>
            <Text style={styles.searchText}>{search}</Text>
            <Pressable
              onPress={() => {
                setSearch("");
                setFilteredShelters(() => {
                  return shelters;
                });
              }}
            >
              <Icon name="times" size={20} color={"white"} />
            </Pressable>
          </View>
        )}
        {fileteredShelters ? <ShelterFlatList /> : <ActivityIndicator />}
        <SearchAndFilters />
      </View>
    );
  }

  return (
    <RootStack.Navigator
      screenOptions={{ headerShown: false, headerTransparent: true }}
      initialRouteName="SheltersList"
    >
      <RootStack.Group>
        <RootStack.Screen
          name="SheltersList"
          component={SheltersList}
          initialParams={{
            types: [],
            purposes: [],
            hasRamp: "",
          }}
        />
        <RootStack.Screen
          name="Shelter"
          component={Shelter}
          initialParams={{ parentWin: "SheltersList" }}
        />
        <RootStack.Screen name="Editor" component={Adder} />
      </RootStack.Group>
      <RootStack.Group
        screenOptions={{
          presentation: "transparentModal",
          animation: "fade_from_bottom",
        }}
      >
        <RootStack.Screen
          name="Filters"
          component={Filters}
          initialParams={{ parentWin: "SheltersList" }}
        />
      </RootStack.Group>
    </RootStack.Navigator>
  );
}

const styles = StyleSheet.create({
  container: {
```

```
    flex: 7,
    width: "100%",
    backgroundColor: "#fff",
    alignItems: "center",
    justifyContent: "center",
  },
  filterContainer: {
    flex: 1,
    width: "100%",
    height: "10%",
    flexDirection: "row",
    borderColor: "#fff",
    alignItems: "center",
    justifyContent: "space-between",
    padding: 7,
  },
  textInput: {
    fontFamily: "Monserrat-SemiBold",
    borderColor: "#ee6c4d",
    borderWidth: 1,
    fontSize: 16,
    padding: 3,
    height: 50,
    borderRadius: 10,
  },
  searchLabel: {
    flexDirection: "row",
    justifyContent: "space-between",
    backgroundColor: "#ee6c4d",
    elevation: 20,
    borderWidth: 1,
    borderColor: "#ee6c4d",
    borderRadius: 15,
    padding: 10,
    margin: 10,
  },
  searchText: {
    fontFamily: "Monserrat-SemiBold",
    fontSize: 15,
    paddingHorizontal: 5,
    color: "#fff",
  },
});
```

Фрагменти коду вебсервера

```
from flask import Flask
from flask_sqlalchemy import SQLAlchemy
from routes.misc import blueprint as misc_blueprint
from routes.shelters import blueprint as shelters_blueprint
from routes.users import blueprint as users_blueprint
from models import base


def create_app():
    db = base.db
    app = Flask(__name__)
    app.config.from_object('config')
    db.init_app(app)
    return app


app = create_app()

# Register blueprints
app.register_blueprint(misc_blueprint, url_prefix='/misc')
app.register_blueprint(shelters_blueprint, url_prefix='/shelters')
app.register_blueprint(users_blueprint, url_prefix='/user')

@app.route("/")
def hello():

    return "Hello world"


if __name__ == "__main__":
    app.run(host='0.0.0.0', port=4567, debug=True)
```

Фрагменти коду роутера укриттів вебсервера

```
from flask import Blueprint
from controllers.shelterController import *

blueprint = Blueprint('shelters', __name__)

blueprint.route('/', methods=['GET'])(index)
blueprint.route('/all', methods=['GET'])(shelters)
blueprint.route('/all/filters', methods=['GET'])(sheltersWithFilters)
blueprint.route('/<int:id>', methods=['GET'])(getShelterById)
blueprint.route('/add/<int:id>', methods=['POST'])(addShelter)
blueprint.route('/add', methods=['POST'])(addShelter)
blueprint.route('/del/<int:id>', methods=['POST'])(delShelter)
```

Фрагменти коду моделі укриття вебсервера

```
from sqlalchemy import *
from sqlalchemy.orm import relationship, mapped_column
from models.base import Base
# from models.favourites import Favourites


shelter_user_association = Table(
    'shelter_user', Base.metadata,
    Column('userId', Integer, ForeignKey('users.id')),
    Column('shelterId', Integer, ForeignKey('shelters.id'))
)


class Shelter(Base):
    '''shelters'''
    __tablename__='shelters'
    id = Column(Integer,
                Sequence('users_id_seq'),
                primary_key=True,
                server_default=Sequence('users_id_seq').next_value(),
                autoincrement=True)
    longitude = Column(DOUBLE_PRECISION, nullable=False)
    latitude = Column(DOUBLE_PRECISION, nullable=False)
    capacity = Column(Integer, nullable=False)
    hasRamp = Column(Boolean, nullable=False)
    typeId = Column(Integer, ForeignKey('shelter_types.id',
onupdate='CASCADE', ondelete='SET NULL'), nullable=False)
    type = relationship("Type", back_populates='shelters')
    purposeId = Column(Integer, ForeignKey('shelter_purposes.id',
onupdate='CASCADE', ondelete='SET NULL'), nullable=False)
    addressId = Column(Integer, ForeignKey('addresses.id',
onupdate='CASCADE', ondelete='SET NULL'), nullable=False)
    users = relationship('User', secondary=shelter_user_association,
back_populates='shelters')

    def __init__(self, longitude, latitude, capacity, hasRamp, typeId,
purposeId, addressId):
        self.longitude = longitude
        self.latitude = latitude
        self.capacity = capacity
        self.hasRamp = hasRamp
        self.typeId = typeId
        self.purposeId = purposeId
        self.addressId = addressId

    def as_dict(self):
        return {c.name: getattr(self, c.name) for c in
self.__table__.columns}
```

Фрагменти коду контролера укриттів вебсервера

```
from flask import jsonify, make_response, request
from sqlalchemy.exc import SQLAlchemyError
from sqlalchemy.sql import text

from models import shelter, base, address, type, purpose


db = base.db


def index():
    return 'Shelters index'


def shelters():
    shelters = db.session.query(shelter.Shelter,
address.Address).join(address.Address).all()
    res = []
    for shelter_res, address_res in shelters:
        dict_shelter = shelter_res.as_dict()
        dict_address = address_res.as_dict()
        if not dict_address['houseNumber'] is None:
            address_str = str(dict_address['street']) + ', ' +
str(dict_address['houseNumber'])
        else:
            address_str = str(dict_address['street'])
        dict = {
            'id': dict_shelter['id'],
            'address': address_str
        }
        res.append(dict)
    return make_response(jsonify(res), 200)
    # return "Shelters"


def sheltersWithFilters():
    types = request.args.getlist('type', None)
    purposes = request.args.getlist('purpose', None)
    hasRamp = request.args.get('hasRamp', None)


    query = ''
    if len(types) > 0:
        types_str = ','.join(types)
        if len(purposes) > 0:
            purposes_str = ','.join(purposes)
            if not hasRamp is None:
                shelters = db.session.query(shelter.Shelter,
address.Address).join(address.Address)\

.filter(shelter.Shelter.typeId.in_(types[0].split(',')),

shelter.Shelter.purposeId.in_(purposes[0].split(',')),
                                shelter.Shelter.hasRamp == hasRamp).all()
            else:
                shelters = db.session.query(shelter.Shelter,
address.Address).join(address.Address) \

.filter(shelter.Shelter.typeId.in_(types[0].split(',')),

shelter.Shelter.purposeId.in_(purposes[0].split(','))).all()
        elif not hasRamp is None:
```

```
            shelters = db.session.query(shelter.Shelter,
address.Address).join(address.Address) \
                .filter(shelter.Shelter.typeId.in_(types[0].split(',')),
shelter.Shelter.hasRamp == hasRamp).all()
        else:
            shelters = db.session.query(shelter.Shelter,
address.Address).join(address.Address) \

.filter(shelter.Shelter.typeId.in_(types[0].split(','))).all()
    elif len(purposes) > 0:
        purposes_str = ','.join(purposes)
        if not hasRamp is None:
            shelters = db.session.query(shelter.Shelter,
address.Address).join(address.Address) \

.filter(shelter.Shelter.purposeId.in_(purposes[0].split(',')),
shelter.Shelter.hasRamp == hasRamp).all()
        else:
            shelters = db.session.query(shelter.Shelter,
address.Address).join(address.Address) \

.filter(shelter.Shelter.purposeId.in_(purposes[0].split(','))).all()
    else:
        shelters = db.session.query(shelter.Shelter,
address.Address).join(address.Address) \
                    .filter(shelter.Shelter.hasRamp == hasRamp).all()


    res = []
    for shelter_res, address_res in shelters:
        dict_shelter = shelter_res.as_dict()
        dict_address = address_res.as_dict()
        if not dict_address['houseNumber'] is None:
            address_str = str(dict_address['street']) + ', ' +
str(dict_address['houseNumber'])
        else:
            address_str = str(dict_address['street'])
        dict = {
            'id': dict_shelter['id'],
            'address': address_str
        }
        res.append(dict)
    return make_response(jsonify(res), 200)


def getShelterById(id):
    query = text(f"""SELECT  sh.id, latitude, longitude,
concat(addresses."street", ', ', addresses."houseNumber") AS address,
                    shelter_types.type AS type,
                    shelter_purposes.purpose AS purpose, capacity,
sh."hasRamp"
                    FROM shelters sh
                    INNER JOIN addresses ON "addressId" = addresses.id
                    INNER JOIN shelter_types ON "typeId" =
shelter_types.id
                    INNER JOIN shelter_purposes ON "purposeId" =
shelter_purposes.id
                    WHERE sh.id = {id}""")
    shelter_res = db.session.execute(query).first()
    res = {}
    if not shelter_res is None:
        res = {
            'id': shelter_res[0],
            'latitude': shelter_res[1],
```

```
            'longitude': shelter_res[2],
            'address': shelter_res[3],
            'type': shelter_res[4],
            'purpose': shelter_res[5],
            'capacity': shelter_res[6],
            'hasRamp': shelter_res[7]
        }
    else:
        return make_response(jsonify(res), 404)
    return make_response(jsonify(res), 200)


def addShelter(id=-1):
    shelter_raw = request.json
    print(shelter_raw)

    address_raw = shelter_raw['address']
    address_arr = address_raw.split(',')
    old_address =
db.session.query(address.Address).filter(address.Address.street ==
address_arr[0],

address.Address.houseNumber == address_arr[1]).first()
    if old_address is None:
        address_model = address.Address(street=address_arr[0],
                                        house_number=address_arr[1])
        try:
            db.session.add(address_model)
            db.session.commit()
        except Exception as err:
            print(err)
            return make_response(jsonify({'message': 'Error syncing with
db', 'error': f'{err}'}), 503)
    else:
        address_model = old_address

    if id is not -1:
        old_shelter =
db.session.query(shelter.Shelter).filter(shelter.Shelter.id == id).first()
        shelter_type = db.session.query(type.Type).filter(type.Type.id ==
shelter_raw['type']).first()
        shelter_purpose =
db.session.query(purpose.Purpose).filter(purpose.Purpose.id ==
shelter_raw['purpose']).first()
        old_shelter.latitude = shelter_raw['latitude']
        old_shelter.longitude = shelter_raw['longitude']
        old_shelter.capacity = shelter_raw['capacity']
        old_shelter.hasRamp = shelter_raw['hasRamp']
        shelter_type.shelters.append(old_shelter)
        shelter_purpose.shelters.append(old_shelter)
        address_model.shelters.append(old_shelter)
        try:
            db.session.commit()
            return make_response(jsonify({'message': 'Successfully
updated'}), 201)
        except SQLAlchemyError as err:
            print(err)
            return make_response(jsonify({'message': 'Error syncing with
db', 'error': f'{err}'}), 503)

    shelter_new = shelter.Shelter(latitude=shelter_raw['latitude'],
                                  longitude=shelter_raw['longitude'],
                                  capacity=shelter_raw['capacity'],
                                  hasRamp=shelter_raw['hasRamp'],
```

```
                                    typeId=shelter_raw['type'],
                                    purposeId=shelter_raw['purpose'],
                                    addressId=address_model.id)
    try:
        db.session.add(shelter_new)
        db.session.commit()
        return make_response(jsonify({'message': 'Successfully added'}),
201)
    except SQLAlchemyError as err:
        print(err)
        return make_response(jsonify({'message': 'Error syncing with db',
'error': f'{err}'}), 503)


def delShelter(id):
    try:
        shelter_res =
db.session.query(shelter.Shelter).filter_by(id=int(id)).delete()
        db.session.commit()
        if shelter_res == 0:
            return make_response(jsonify(shelter_res), 404)
        return make_response(jsonify(shelter_res), 201)
    except SQLAlchemyError as e:
        error = str(e.__dict__['orig'])
        return make_response(jsonify(error), e.code)
```

Додаток З
Копія презентації

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

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

КАФЕДРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ КОМП’ЮТЕРНИХ СИСТЕМ

МОБІЛЬНИЙ ЗАСТОСУНОК ДЛЯ ЗНАХОДЖЕННЯ НАЙБЛИЖЧОГО УКРИТТЯ

Виконала: Вергун Марія Олександрівна

Керівник: ст. викладач, к.т.н. Сулема Ольга Костянтинівна

Київ – 2023

ПОСТАНОВКА ЗАДАЧІ

Мета проєкту: забезпечення швидкого пошуку укриття, найближчого до місця перебування користувача, шляхом розроблення відповідного мобільного застосунку.

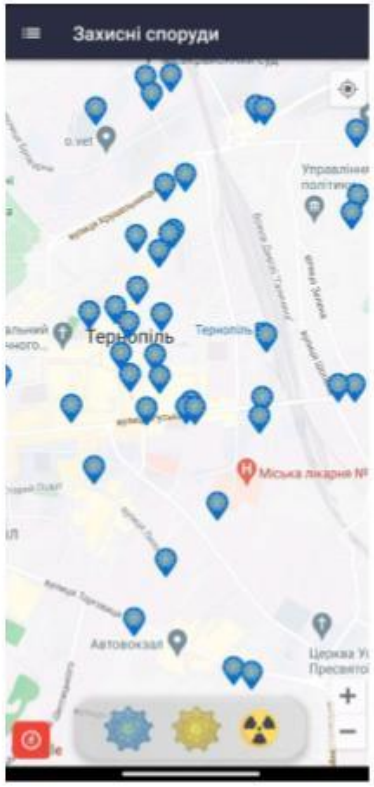
Завдання:

1. Проаналізувати наявні рішення у галузі безпекових застосунків.
2. Проаналізувати наявні методи, алгоритми та засоби розробки.
3. Розробити мобільний застосунок.
4. Протестувати роботу системи.

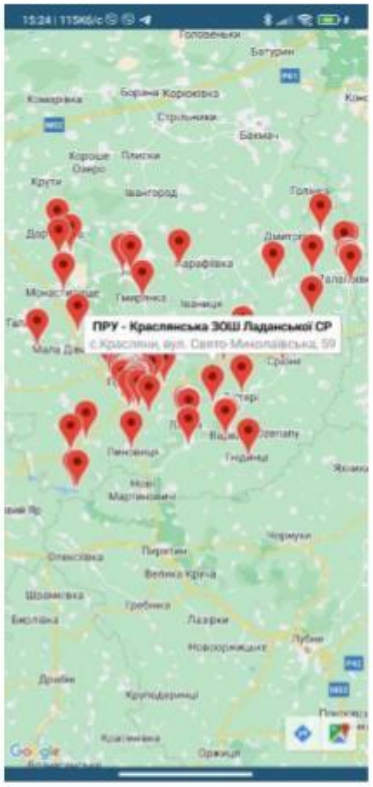
АКТУАЛЬНІСТЬ

- Мало інформації про розміщення укриттів.
- Неактуальність наявної інформації.
- Автоматизоване рішення скорочує час на пошуки.
- Безпека людини – передусім.

ІСНУЮЧІ РІШЕННЯ



«ДеУкриття»

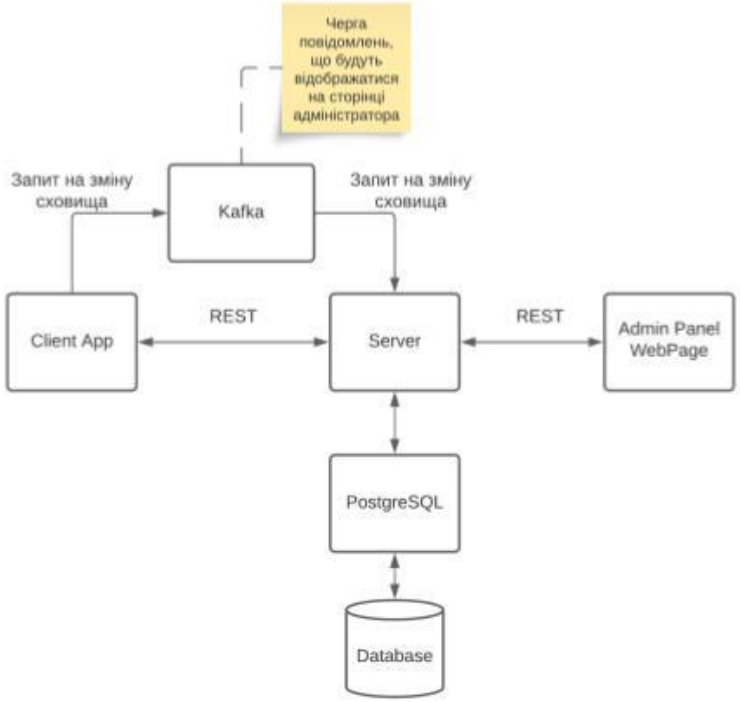


«Прилуки Незламні»

«Київ Цифровий»

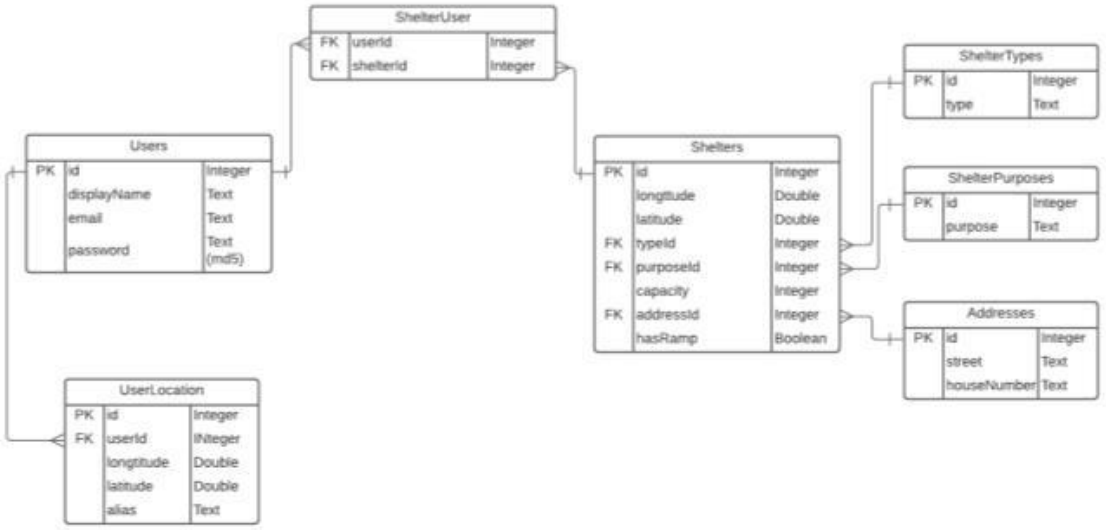
АРХІТЕКТУРА СИСТЕМИ

Структура розроблюваної системи:



СТРУКТУРА БАЗИ ДАНИХ

UML діаграма бази даних:



ЗАСОБИ РОЗРОБКИ

РОЗРОБЛЕНІ АЛГОРИТМИ

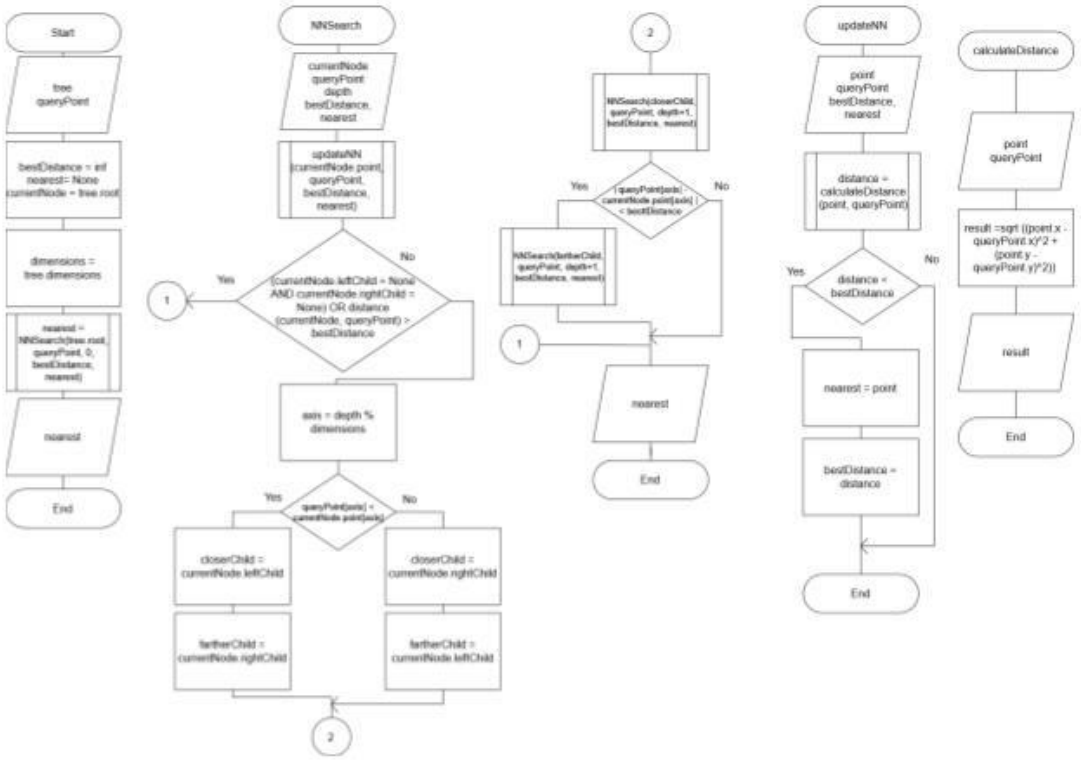
- Алгоритм пошуку найближчого укриття
- Алгоритм фільтрації за типом, призначенням та наявністю пандусу

АЛГОРИТМ ПОШУКУ НАЙБЛИЖЧОГО УКРИТТЯ (1/2)

1. Отримання геопозиції користувача.
2. Отримання укриттів з бази даних в межах 1 км від розташування.
3. Побудова kd-дерева з геопозицій укриттів.
4. Знаходження найближчого сусіда до геопозиції користувача в дереві
5. Розміщення знайденого результату на початку масиву геопозицій
6. Повернення списку геопозицій
7. Розміщення міток на карті

АЛГОРИТМ ПОШУКУ НАЙБЛИЖЧОГО УКРИТТЯ (2/2)

Блок-схема алгоритму пошуку найближчого сусіда у kd-дереві:



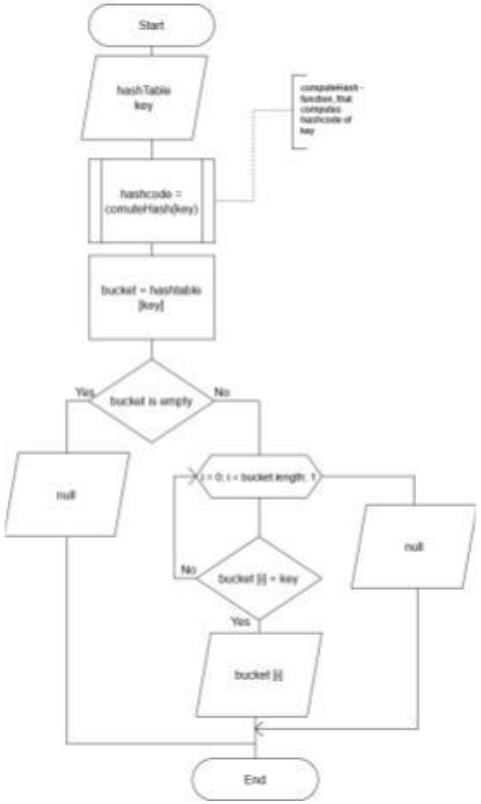
АЛГОРИТМ ФІЛЬТРАЦІЇ ЗА ТИПОМ (1/2)

Передумова: створення геш-індексу таблиці укриттів бази даних за полем типу

1. Отримання ідентифікатора типу від клієнтського застосунку
2. Обчислення геш-функції ідентифікатора
3. Пошук у геш-таблиці комірки з отриманим геш-кодом
4. Повернення результатів – елементів комірки
5. Оновлення списку укриттів на клієнтському застосунку

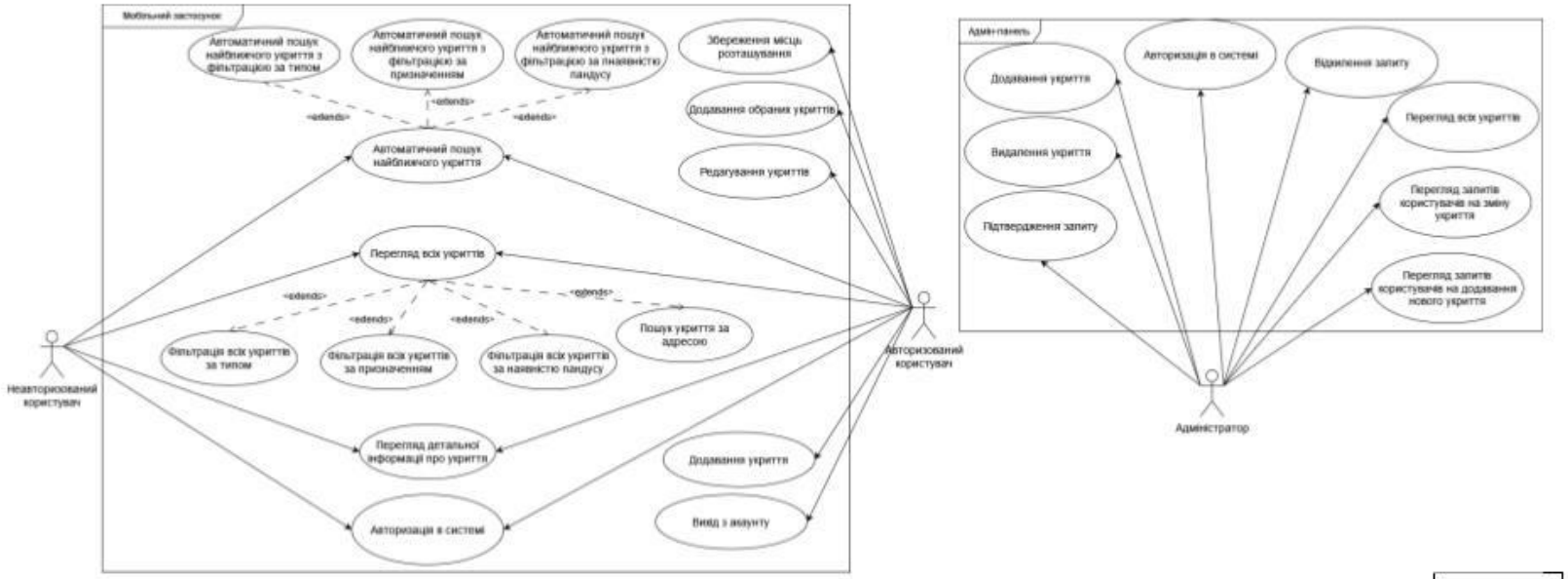
АЛГОРИТМ ФІЛЬТРАЦІЇ ЗА ТИПОМ (2/2)

Блок-схема алгоритму пошуку у геш-таблиці:

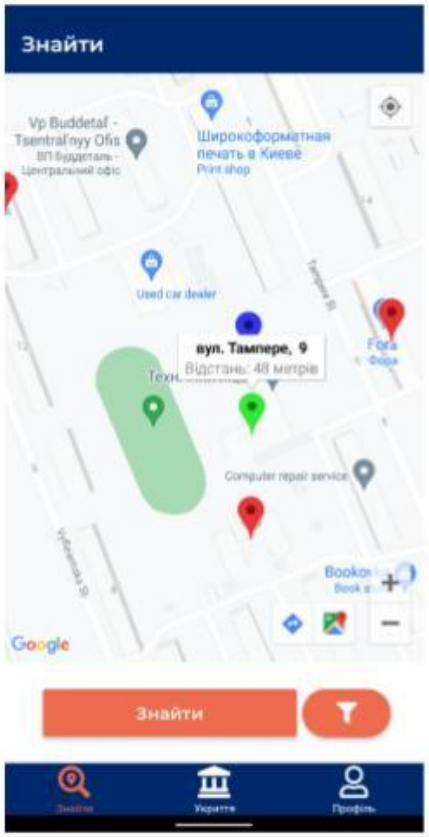


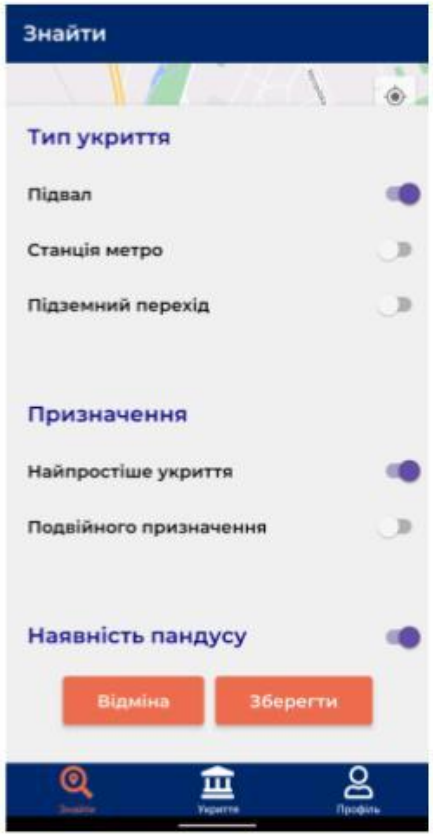
СЦЕНАРІЇ ВИКОРИСТАННЯ

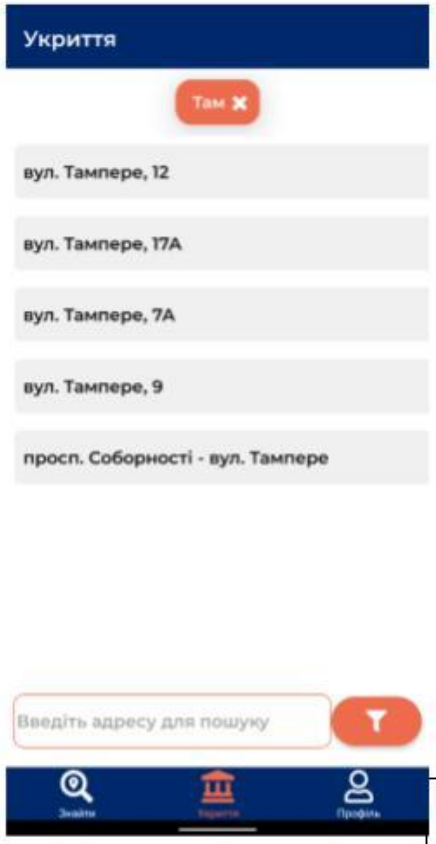
UML діаграма прецедентів:



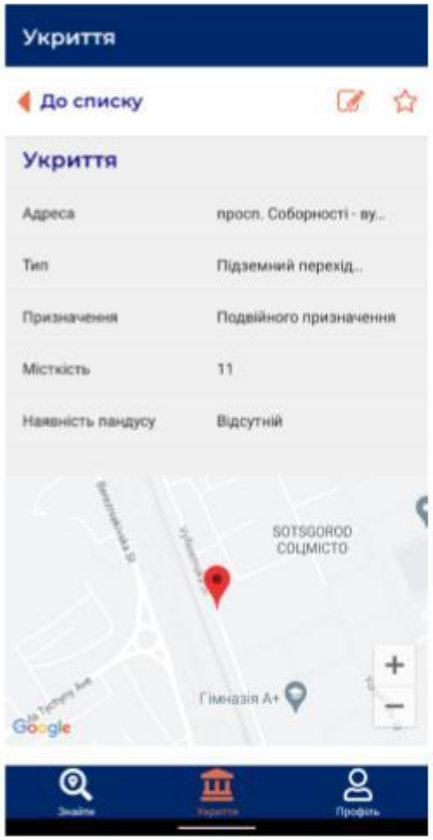
ПРИКЛАДИ РОБОТИ ПРОГРАМИ (1/2)

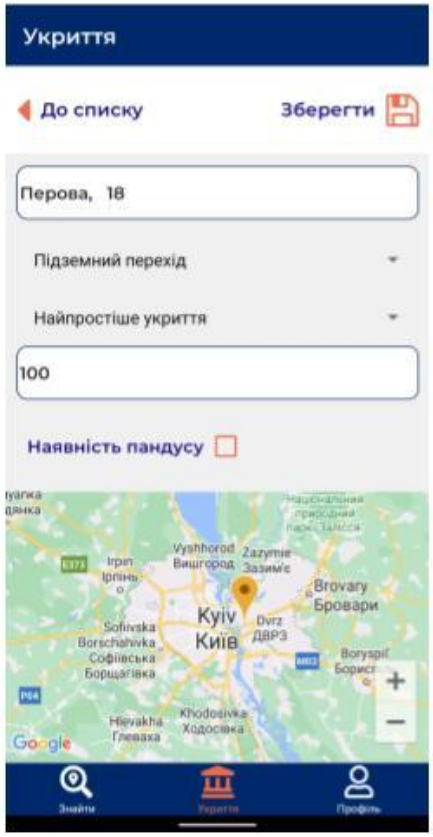


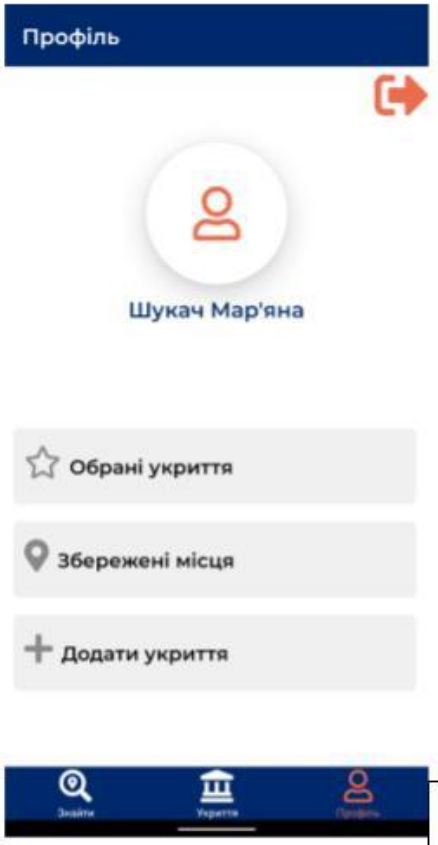




ПРИКЛАДИ РОБОТИ ПРОГРАМИ (2/2)







ОЦІНКА РОЗРОБЛЕНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Складність алгоритмів:

- Алгоритм пошуку в kd – дереві: $O(n \log n)$ [1]
- Алгоритм фільтрації на основі геш-таблиці: $O(1)$

1. Friedman J. H. An algorithm for finding best matches in logarithmic expected time [Електронний ресурс] / J. H. Friedman, J. L. Bentley, R. A. Finkel // Stanford University. – 1976. – Режим доступу до ресурсу: https://inspirehep.net/files/3c9ac14423f008f75b237d8b95b68777.

ПОРІВНЯЛЬНА ОЦІНКА РІШЕНЬ

	Наявність автоматичного підбору найближчого укриття	Наявність можливості перегляду детальної інформації про кожне укриття	Реалізована можливість фільтрації за типом, призначенням, наявністю пандусу
«ДеУкриття»	+	+	+
«Прилуки Незламні»	–	–	–
«Київ Цифровий»	–	+	–
Розроблена система	+	+	+

ВИСНОВКИ (1/2)

1. Проаналізовано програмні рішення в сфері безпекових застосунків
2. Проаналізовано алгоритми, методи та засоби розробки
3. Запропоновано клієнт-серверну архітектуру застосунку
4. Розроблено алгоритми пошуку найближчого укриття та алгоритми фільтрації
5. Розроблено нормалізовану структуру бази даних
6. Розроблено клієнтський застосунок
7. Розроблено серверну частину ПЗ
8. Протестовано та порівняно ПЗ з аналогами за:
   - Наявністю автоматичного пошуку найближчого укриття.
   - Доступністю детальної інформації по кожному укриттю.
   - Можливістю пошуку та фільтрації укриттів.

ВИСНОВКИ (2/2)

- Розроблена система покриває необхідні вимоги.
- Основний функціонал застосунку протестовано.
- Система надає автоматичний пошук найближчого укриття.
- Система перевершує функціонал існуючих рішень за порівняльною оцінкою

УНІКАЛЬНІСТЬ

UNICHECK
by Turnitin

Ім'я користувача:
Сулема Ольга Костянтинівна

ID перевірки:
1015620545

Дата перевірки:
16.06.2023 01:15:22 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
16.06.2023 01:19:13 EEST

ID користувача:
100012488

Назва документа: **Вергун_антиплагіат**

Кількість сторінок: **48** Кількість слів: **8749** Кількість символів: **67028** Розмір файлу: **76.15 KB** ID файлу: **1015267453**

0.63%
Схожість

Найбільша схожість: **0.11%** з Інтернет-джерелом (https://uk.wikipedia.org/wiki?curid=1964831)

0.22% Джерела з Інтернету	3	Сторінка 50
0.51% Джерела з Бібліотеки	23	Сторінка 50

0.67% Цитат

Цитати	2	Сторінка 51

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Дякую за увагу!

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

"ЗАТВЕРДЖЕНО"

Завідувач кафедри

_____ Євгенія СУЛЕМА

"___" _____ 2022 р.

МОБІЛЬНИЙ ЗАСТОСУНОК ДЛЯ ЗНАХОДЖЕННЯ НАЙБЛИЖЧОГО УКРИТТЯ

Програма та методика тестування

ДП.045440-04-51

"ПОГОДЖЕНО"

Керівник проєкту:

_____ Ольга СУЛЕМА

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Марія ВЕРГУН

2022

ЗМІСТ

1. Об’єкт випробувань……... 3
2. Мета тестування…… 3
3. Методи тестування……...…. 3
4. Засоби та порядок тестування…….. 4

1. ОБ’ЄКТ ВИПРОБУВАНЬ

Мобільний застосунок для знаходження найближчого укриття, що створений за використання фреймворку React Native та Expo.

2. МЕТА ТЕСТУВАННЯ

У процесі тестування має бути перевірено наступне:

1) функціональна працездатність елементів сторінок мобільного застосунку;
2) наявність доступу до Google Maps API;
3) наявність доступу до вебсерверу системи;
4) наявність доступу до бази даних системи;
5) відповідність форматів та протоколів передачі даних між вебсервером системи та мобільним застосунком;
6) забезпечення належного рівня безпеки даних;
7) зручність роботи з мобільним застосунком;
8) відповідність дизайну вимогам Технічного завдання.

3. МЕТОДИ ТЕСТУВАННЯ

Тестування виконується методом Gray Box Testing. Перевіряється як код, так і безпосередньо програмний продукт на відповідність функціональним вимогам.

Використовуються наступні методи:

1) функціональне тестування, зокрема на рівні Critical path test (базове тестування);
2) тестування продуктивності програмного забезпечення, зокрема Stability testing (тестування стабільності) та Load testing (навантажувальне тестування);
3) тестування інтерфейсу.

4. ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Тестування вебсерверу виконується засобами інструментарію Postman.

Працездатність мобільного застосунку перевіряється шляхом:

1) динамічного ручного тестування – введенням граничних та недопустимих значень в поля, які можна редагувати;
2) динамічного ручного тестування на відповідність функціональним вимогам;
3) статичного тестування коду;
4) тестування мобільного застосунку на різних платформах;
5) тестування при максимальному навантаженні;
6) тестування стабільності роботи при різних умовах;
7) тестування зручності використання;
8) тестування інтерфейсу.

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

"ЗАТВЕРДЖЕНО"

Завідувач кафедри

_____ Євгенія СУЛЕМА

"___" _____ 2023 р.

МОБІЛЬНИЙ ЗАСТОСУНОК ДЛЯ ЗНАХОДЖЕННЯ НАЙБЛИЖЧОГО УКРИТТЯ

Керівництво користувача

ДП.045440-05-34

"ПОГОДЖЕНО"

Керівник проєкту:

_____ Ольга СУЛЕМА

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Марія ВЕРГУН

2023

ЗМІСТ

1. Опис структури мобільного застосунку……..3
2. Опис вмісту сторінок мобільного застосунку…… 3
3. Процедура пошуку найближчого укриття…… 11
4. Використання фільтрів для пошуку…….. 14
5. Процедура авторизації користувача…….. 15

1. Опис структури мобільного застосунку

Мобільний застосунок представляє собою динамічний інтерфейс з 3 основних функціональних вкладок:

- Знайти – вкладка, з якої відбувається пошук найближчого укриття.
- Укриття – вкладка, що містить перелік усіх укриттів та поля для їх пошуку та фільтрації.
- Профіль – вкладка з функціоналом для авторизованого користувача. Надає можливість авторизації.

Кожна вкладка містить сторінку, з якої доступна навігація всередині вкладки. Наприклад, на вкладці користувача доступний перехід до сторінки реєстрації, а затим допускає перехід по сторінкам обраних укриттів, інформації про обране укриття та редагування укриття.

2. Опис вмісту сторінок мобільного застосунку

Сторінка «Знайти» (рис. 1) містить інтерактивну мапу, на якій розміщуються маркери розташування користувача та результатів пошуку укриттів. Також на сторінці можна побачити кнопку «Знайти» та кнопку зі знаком фільтрів, що відкриває вікно фільтрів пошуку (рис. 2). Авторизований користувач також бачить на сторінці кнопку «Зберегти поточне місце» для збереження свого місця розташування. Натискання на цю кнопку відкриває вікно збереження поточного місця (рис. 3).

Сторінка «Укриття» (рис. 4) містить список всіх укриттів, доступних у базі даних застосунку. Кожен елемент списку є інтерактивним і дозволяє натискання на нього. Також доступне аналогічне до сторінки «Пошук» вікно фільтрів та поле пошуку укриттів за адресою. Для оновлення списку необхідно потягнути його зверху. При натисканні на обраний елемент списку, відкривається його сторінка з детальною інформацією (рис. 5).

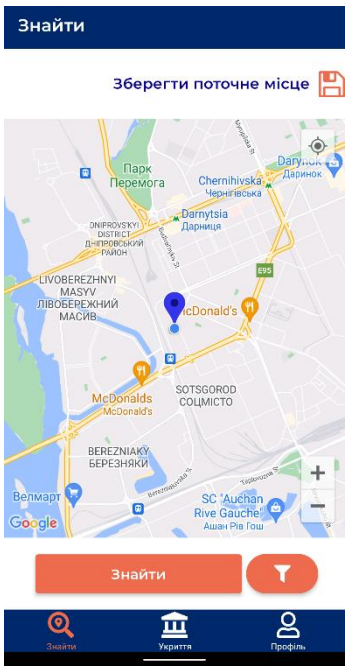


Рис. 1. Сторінка «Знайти»

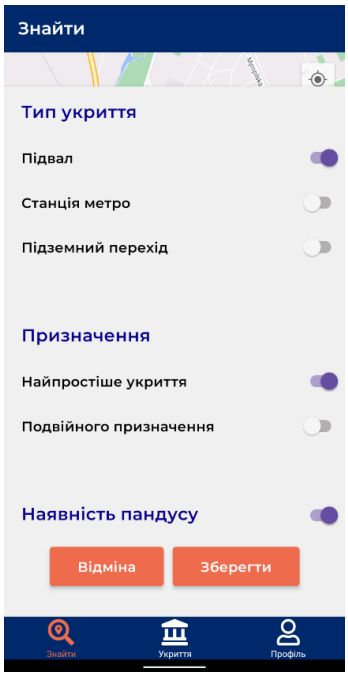


Рис. 2. Вікно фільтрів пошуку

Сторінка з детальною інформацією про укриття містить таблицю з такими полями:

- Адреса – адреса розташування укриття.
- Тип – тип укриття.
- Призначення – призначення укриття.

- Місткість – кількість людей, на яку розраховане укриття.
- Наявність пандусу.

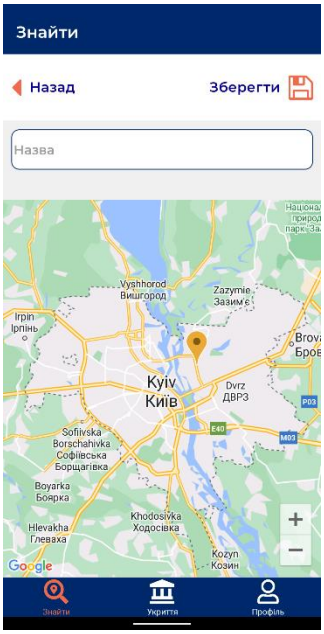


Рис. 3. Вікно збереження поточного місця

Рис. 4. Сторінка «Укриття»

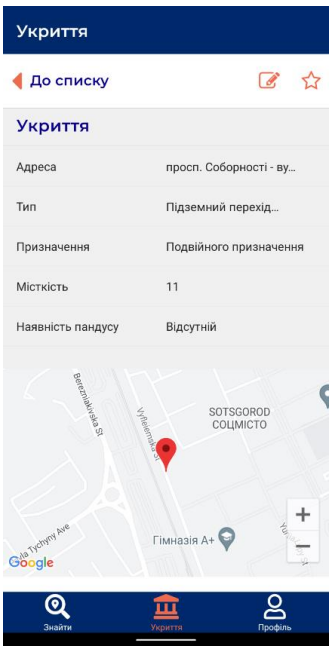


Рис. 5. Сторінка з детальною інформацією про укриття (Авторизований користувач)

Авторизованому користувачеві доступний також функціонал редагування через натискання на символ редагування у правому верхньому куті екрану. Він відкриває сторінку редагування (рис. 6).

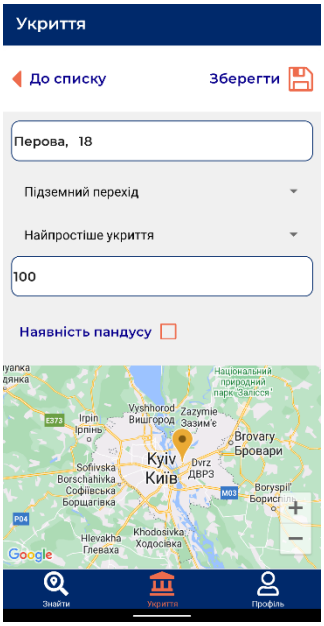


Рис. 6. Сторінка редагування

На даній сторінці розміщуються:

- тектове поле з адресою;
- випадаючі списки для типу укриття та його призначення;
- числове поле для введення місткості;
- чек пойнт для позначення наявності пандусу;
- інтерактивна мапа для відмітки розташування укриття.

Поряд з кнопкою редагування розміщується також кнопка додання в обрані у вигляді зірки. Якщо укриття додано користувачем в обрані, то зірка буде залита кольором, якщо ж ні, то міститиме лише її контур.

Сторінка «Профіль» (рис. 7) без авторизації містить форму авторизації та кнопку, що відкриває сторінку реєстрації.

Форма авторизації складається з поля для введення електронної пошти та паролю користувача.

Форма реєстрації (рис. 8) надає поле введення імені користувача, його адреси електронної пошти та паролю.

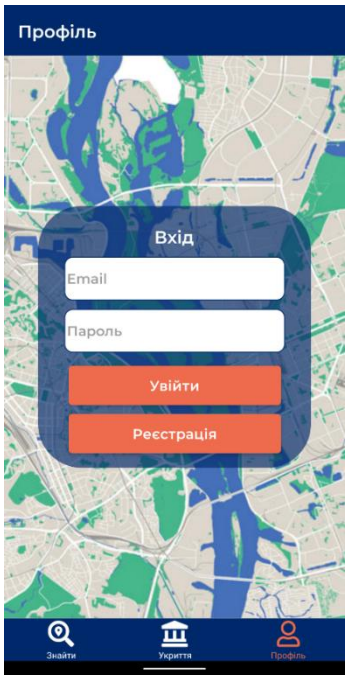


Рис. 7. Сторінка «Профіль» з формою авторизації (якщо користувач неавторизований)

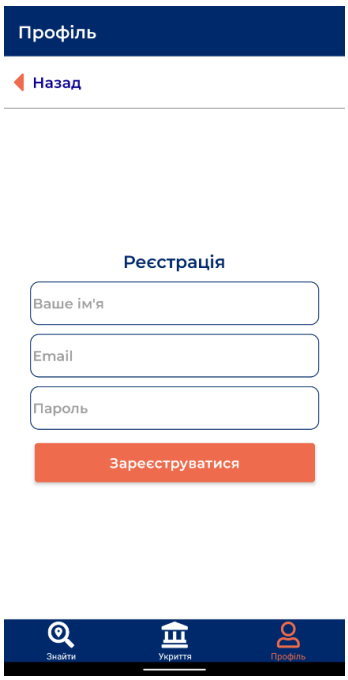


Рис. 8. Сторінка реєстрації

Після авторизації вкладка «Профіль» містить сторінку профілю користувача (рис. 9).

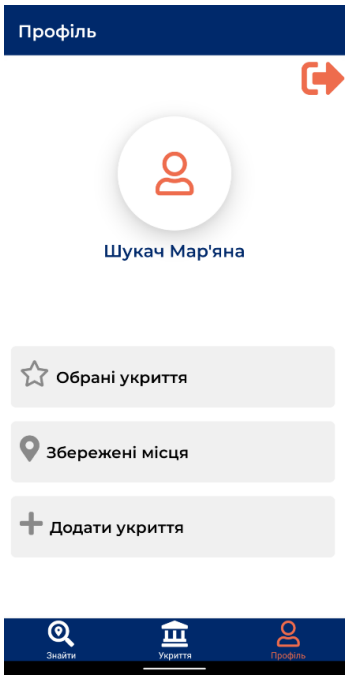


Рис. 9. Сторінка профілю користувача

На сторінці профілю наявні три опції, які відкривають три відповідні сторінки:

- Обрані укриття – відкриває сторінку обраних укриттів.
- Збережені місця – відкриває сторінку збережених користувачем місць.
- Додати укриття – відкриває сторінку додавання укриття.

У правому верхньому куті розміщується кнопка виходу з облікового запису. При натисканні на неї, користувач стає неавторизованим і сторінка замінюється сторінкою входу в систему.

На сторінці «Обрані укриття» (рис. 10) міститься інтерактивний список укриттів, обраних користувачем. При натисканні на елемент списку відкривається сторінка детальної інформації про укриття, аналогічна такій же сторінці на вкладці «Укриття».

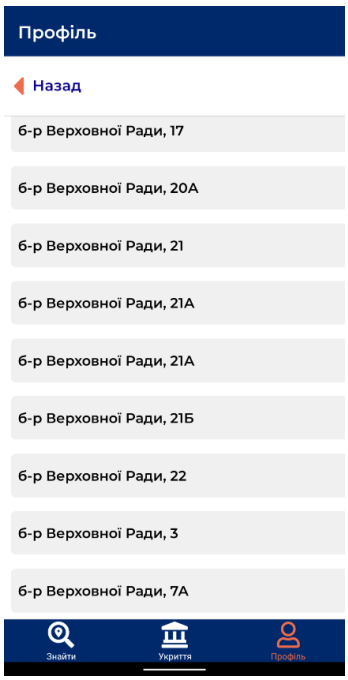


Рис. 10. Сторінка обраних укриттів користувача

На сторінці «Збережені місця» (рис. 11) міститься інтерактивний список збережених місць користувача. При натисканні на елемент списку відкривається сторінка перегляду збереженого місця (рис. 12), на якій

розміщено інтерактивну мапу з відміченим маркером місця. Також на цій сторінці розміщується кнопка пошуку найближчого укриття.

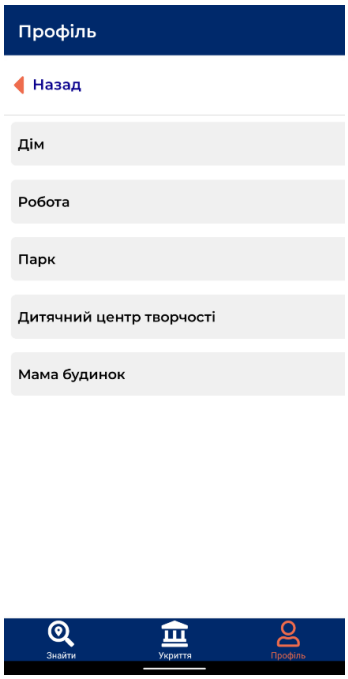


Рис. 11. Сторінка збережених місць

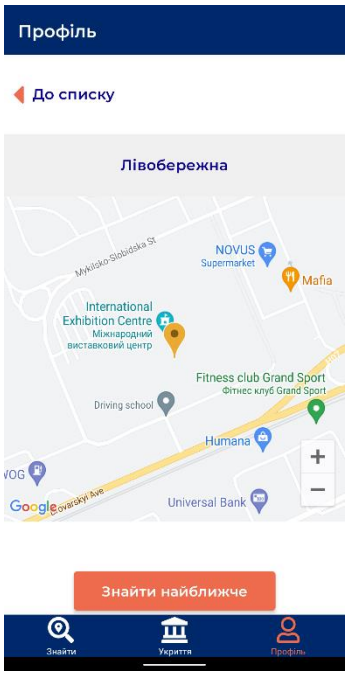


Рис. 12. Сторінка збереженого місця

Сторінка «Додати укриття» (рис. 13) являє собою форму для створення нового укриття. Форма містить такі елементи вводу:

- Текстове поле адреси.
- Випадаючий список типу.
- Випадаючий список призначення.
- Числове поле введення місткості.
- Чек пойнт для відмітки наявності пандусу.

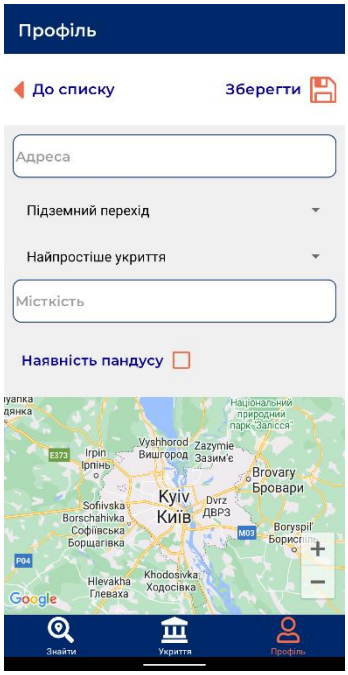


Рис. 13. Сторінка додавання укриття

Для збереження введеної інформації необхідно натиснути на кнопку «Зберегти» у правому верхньому куті.

3. Процедура пошуку найближчого укриття

Для пошуку найближчого укриття користувачеві необхідно дати дозвіл на отримання геопозиції пристрою у діалоговому вікні, що відкривається при першому запуску застосунку (рис. 14).

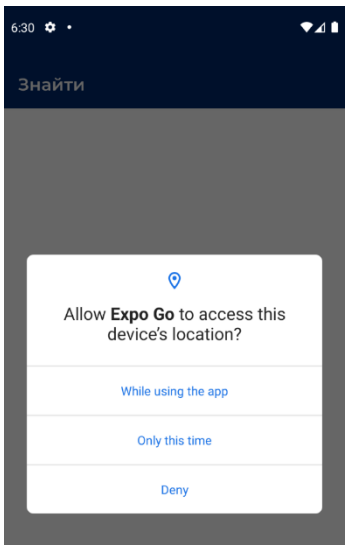


Рис. 14. Діалогове вікно запиту геопозиції

Після того користувачеві відкривається головне вікно – сторінка «Знайти». На інтерактивній мапі синім кольором відмічено поточне розташування користувача. Для знаходження укриття необхідно натиснути кнопку «Знайти найближче». Після натискання та успішного пошуку на мапі з'являються маркери – місця розташування укриттів (рис. 15).

Зеленим кольором на мапі відмічено найближче зі знайдених укриттів. При натисканні на укриття відкривається мітка цього маркера з адресою укриття та зазначенням відстані до нього від користувача.

Щоб побудувати маршрут, необхідно при обраному укритті натиснути на значок побудови маршруту на мапі у правому нижньому куті (рис. 16). Після цього користувач переправляється на сторонній сервіс Google Maps, де йому доступний побудований маршрут до укриття (рис. 17).

Користувач також може обрати альтернативні варіанти укриття – будь-який червоний маркер на карті – для побудови маршруту до нього.

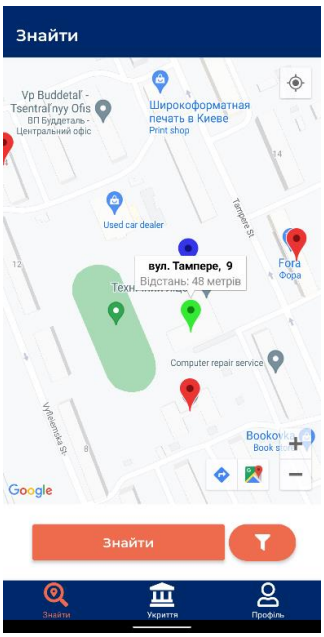


Рис. 15. Знайдені найближчі укриття

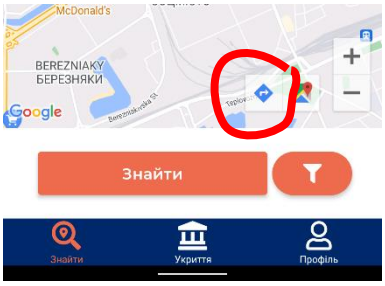


Рис. 16. Кнопка побудови маршруту

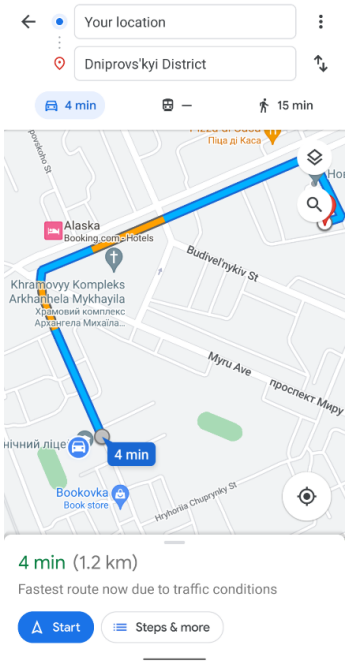


Рис. 17. Побудований маршрут

4. Використання фільтрів для пошуку

Для пошуку найближчого укриття передбачені фільтри пошуку. Вікно фільтрів (рис. 18) можна відкрити, натиснувши на відповідну кнопку поряд з кнопкою «Знайти найближче».

На відкритому вікні користувач обирає фільтри, які йому потрібні. Після того, як всі необхідні фільтри обрані, необхідно натиснути кнопку «Зберегти». При натисканні на цю кнопку, вікно фільтрів закривається і на попередньому екрані знову доступна кнопка «Знайти». При натисканні на неї, користувачу будуть видані результати пошуку з фільтрами.

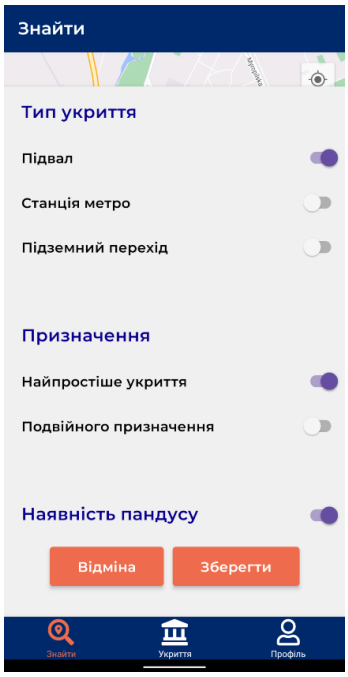


Рис. 18. Вікно фільтрів

Для відміни фільтрів, необхідно відкрити вікно фільтрів та натиснути кнопку «Відміна». Ця кнопка знімає всі збережені фільтри.

Для пошуку усіх укриттів з фільтрами, на сторінці «Укриття» аналогічно необхідно натиснути на значок фільтрів поруч з полем пошуку за адресою. Відкриється вікно фільтрів, після обрання необхідних та натискання кнопки «Зберегти», список усіх укриттів буде відфільтровано. Відміна фільтрів пошуку аналогічна.

5. Процедура авторизації користувача

Для авторизації користувача у системі йому треба мати облікові дані, зареєстровані у системі. До облікових даних належать електронна пошта та пароль.

Якщо користувач має облікові дані, то для авторизації йому потрібно пройти процедуру автентифікації. Для цього необхідно зайти на сторінку «Профіль» (рис. 19) та у формі авторизації ввести свої облікові дані. У поле *email* – електронну пошту, у поле *password* – пароль та натиснути кнопку «Увійти».

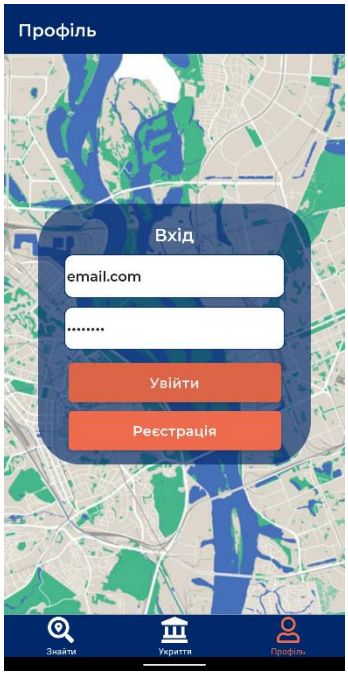


Рис. 19. Сторінка авторизації

При виникненні помилки облікових даних користувачеві буде повідомлено про це шляхом виділення полів введення червоним кольором та виводом помилки у формі авторизації (рис. 20).

Якщо облікових даних у користувача немає, то він може зареєструватися у системі. Для цього необхідно відкрити вікно реєстрації. Вікно реєстрації знаходиться на сторінці «Профіль» – «Зареєструватися».

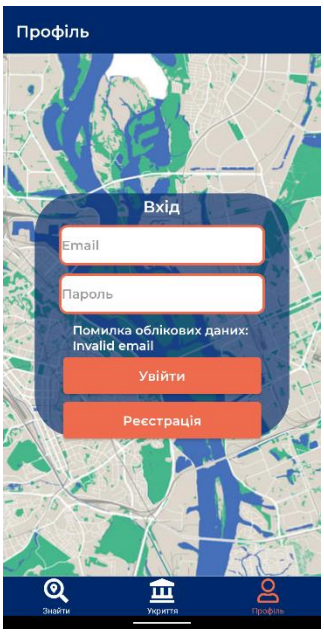


Рис. 20. Помилкові облікові дані

Для реєстрації користувач має ввести своє ім’я у поле «Ваше ім’я», ввести адресу електронної пошти у поле «Email» та придумати пароль і ввести його у поле «Password» (рис. 21). Якщо користувач з такою адресою електронної пошти вже зреєстрований, то користувачу буде про це повідмлено спливаючим сповіщенням (рис. 22).

Після реєстрації користувач перенаправляється на форму входу і може здійснити вхід за своїми обліковими даними.

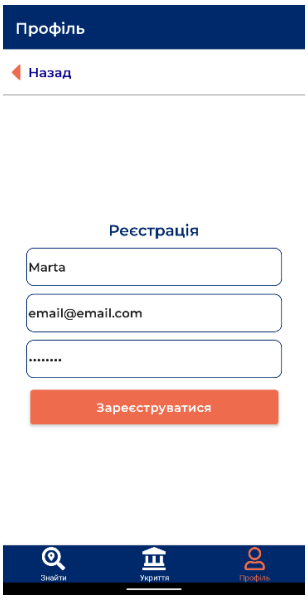


Рис. 21. Форма реєстрації

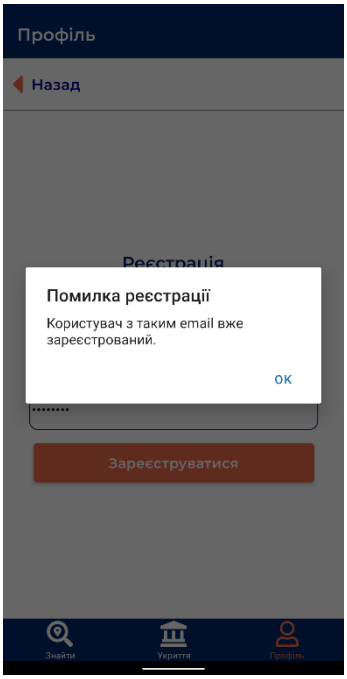


Рис. 22. Повідомлення про помилку реєстрації