МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА

НАГОЛЮК ДМИТРО ОЛЕКСАНДРОВИЧ

Допускається до захисту:
в.о. завідувача кафедри інформаційних технологій
канд. техн. наук, доцент
_____ О. В. Зелінська
«____»_____
20__ р.

МОБІЛЬНИЙ ДОДАТОК ДЛЯ ПРОКЛАДАННЯ ОПТИМАЛЬНОГО МАРШРУТУ В МІСТІ В РЕЖИМІ РЕАЛЬНОГО ЧАСУ

Спеціальність 122 Комп’ютерні науки

Кваліфікаційна (магістерська) робота

Науковий керівник:
Ніколюк П.К.,
професор кафедри інформаційних технологій, д.ф-м.н, професор
_____

Оцінка: _____ / _____ / _____
(бали/за шкалою ЄКТС/за національною шкалою)

Голова ЕК: _____

Вінниця — 2024

АНОТАЦІЯ

Наголюк Д.О. Мобільний додаток для прокладання оптимального маршруту в місті в режимі реального часу. Спеціальність 122 «Комп’ютерні науки», освітня програма «Комп’ютерні технології обробки даних». Донецький національний університет імені Василя Стуса, Вінниця, 2024.

Кваліфікаційна (магістерська) робота присвячена розробці мобільного додатку для прокладання оптимального маршруту в місті в режимі реального часу.

Ключові слова: додаток, маршрут, оптимальний 65 сторінок, 49 рис., 15 джерел, 6 додатків.

ABSTRACT

Naholiuk D.O. A mobile application for planning the optimal route in the city in real time. Specialty 122 "Computer science", educational program "Computer data processing technologies". Vasyl Stus Donetsk National University, Vinnytsia, 2024.

The qualification (master's) work is devoted to the development of a mobile application for planning the optimal route in the city in real time.

Keywords: appendix, route, optimal 65 pages, 49 figures, 15 sources, 6 applications.

ЗМІСТ

ВСТУП

Навігаційна система – це сукупність приладів, алгоритмів і програмного забезпечення, що забезпечують орієнтування об'єкта в просторі [1].

Навігаційні системи стали невід’ємною частиною нашого повсякденного життя і відіграють надзвичайно важливу роль у різних областях. Вони є незамінними помічниками, які сприяють знаходженню шляху до різних місць та допомагають в різних сферах:

- Автомобільна навігація: однією з найпоширеніших форм використання навігаційних систем є їх використання в автомобілях. GPS–пристрої в автомобілях допомагають водіям знаходити оптимальний маршрут до пункту призначення, запобігаючи загубленню та заощаджуючи час. Вони також надають інформацію про дорожні умови, пробки та об'їзні маршрути, допомагаючи уникнути непотрібних затримок та підвищити безпеку на дорогах.
- Авіація та мореплавство: У сферах авіації та мореплавства навігаційні системи відіграють важливу роль. Вони дозволяють пілотам та морякам точно визначати своє місцезнаходження, контролювати маршрут та безпечно досягати пункту призначення. Без навігаційних систем авіаційні та морські подорожі були б значно складнішими та небезпечними.
- Допомога у надзвичайних ситуаціях: Навігаційні системи відіграють важливу роль у наданні допомоги під час надзвичайних ситуацій, таких як природні катастрофи, аварії або пошуки загублених осіб. Наприклад, системи навігації допомагають рятувальним службам точно визначити місцезнаходження потерпілих або потенційно небезпечних об'єктів, що дозволяє швидко реагувати та зберігати життя.

Окрім цього, розвиток мобільних додатків став ще одним стимулом для розвитку навігаційних систем, оскільки тепер їх можна використовувати на особистих мобільних пристроях.

Актуальність додатків для навігації систем в сучасному світі важко переоцінити. Ось декілька аспектів, які підкреслюють їх значення:

- **Зручність та комфорт:** вони роблять наше життя більш зручним та комфортним. Вони дозволяють нам швидко та легко знаходити шлях до будь–якого місця, будь то нова кав'ярня в місті або віддалений готель у незнайомому краї. Вже не потрібно розгублюватись в незнайомих місцях чи залежати від запам'ятовування складних маршрутів.
- **Безпека на дорозі:** додатки для навігації є важливим інструментом для підвищення безпеки на дорозі. Вони надають інформацію про дорожні умови, пробки, ризиковані ділянки та інші фактори, що можуть вплинути на безпеку. Водії можуть обирати оптимальні маршрути, уникати небезпек та зменшувати ризик аварій.
- **Ефективність та економія ресурсів:** навігаційні додатки допомагають зберігати ресурси, такі як паливо та час. Вони допомагають підібрати найкоротший та найшвидший маршрут, оптимізувати рух транспорту та уникати зайвих затрат. Це особливо важливо у великих містах, де пробки та затори можуть спричиняти втрати часу та підвищення екологічного впливу.

Мета магістерської роботи: розробити мобільний додаток для прокладання маршруту в режимі реального часу. Вивчити методику розробки сучасних програм для мобільних пристроїв.

Задачами дослідження є:

1. Огляд існуючих програм для навігації.
2. Вивчення необхідного списку технологій для розробки додатка.
3. Розробка мобільного додатку для навігації в місті в режимі реал

Об’єкт дослідження: мобільний додаток для прокладання оптимального

маршруту в місті в режимі реально часу.

Предмет дослідження: процес розробки мобільного додатку для прокладання оптимального маршруту в місті в режимі реального часу.

Структура роботи: кваліфікаційна робота складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків. Робота містить 65 сторінок, 49 рисунків та список літератури з 15 джерел, 6 додатків.

РОЗДІЛ 1
ПОСТАНОВКА ЗАДАЧІ І ОГЛЯД АНАЛОГІВ ІСНУЮЧИХ ДОДАТКІВ ЗА ДАНОЮ ТЕМАТИКОЮ

В даному розділі наведена постановка задачі дослідження. Проведена порівняльна характеристика з існуючими аналогами.

1.1 Постановка задачі

Використовуючи технологію «React Native» розробити мобільний додаток для прокладання оптимального маршруту в місті в режимі реального часу. Додаток має бути адаптивним: однаково добре відображатися на пристроях з різною діагоналлю екрану – планшетах та смартфонах. Програма має складатися з 4 основних екранів.

Перший екран – карта. На ній потенційний користувач повинен бачити своє місцезнаходження на мапі. Він має мати змогу масштабувати карту, переміщатися по ній. Також повинен відображатися компас, при натисненні на який, карта має бути направлена на Північ. Також має відображатися кнопка повернення до поточного місцезнаходження на карті. У верхній частині екрану повинна відображатися стрічка пошуку адрес: вона має знаходити їх по ключових словах, використовуючи для цього сервіси геокодингу. Геокодинг – процес перетворення слів в координати і навпаки [2]. Знайдені адреси мають бути представлені списком, при натисненні на елемент списку користувача повинно перенаправляти на карті в вибрану точку і показуватися модальне вікно.

У графічному інтерфейсі користувача модальним називається вікно, що блокує роботу користувача з батьківським застосунком доти, доки користувач це вікно не зачинить. У вигляді модальних переважно реалізуються діалогові вікна. Також модальні вікна часто використовуються для привернення уваги користувача до важливої події чи критичної ситуації[3].

Модальне вікно повинно бути з наступними опціями:

1. Додавання адреси у вибране.
2. Закриття модального вікна.
3. Навігація до вибраної точки.

Якщо користувач, натисне на кнопку навігації, то на карті повинен відобразитися спеціальний режим – «режим навігації». В цьому режимі на карті повинен малюватися шлях до вибраної точки. В кутку екрану повинна відображатися кнопка виходжу з режиму.

Другий екран – список улюблених місць, адрес. У ньому повинні відображатися вибрані адреси. Повинна бути реалізована можливість переходу на карті до вибраної адреси та видалення зі списку.

Третій екран – інформаційний. В ньому має міститися інформація та способи зв’язку з розробником.

Четвертий екран – налаштування. На ньому повинні конфігуруватися джерело такі параметри:

1. Тема – вибір теми світлий або темний
2. Автоматична тема – автоматичний вибір світлої або темної теми в залежності від обраної на телефоні.
3. Затори – вибір чи показувати затори на мапі.
4. Зміна мови – вибір мови між українською та англійською.

В нижній частині екрану повинно відображатися меню, за допомогою якого можна буде переходити між різними екранами.

Окремо варто згадати про функціонал, який буде робити додаток унікальним серед інших. В додаток буде надана можливість обирати постачальника карт. Кожен постачальник карт має свої переваги та недоліки, а давши користувачу обирати, ми беремо все найкраще з них.

1.2 Орієнтування на карті

Тут потрібно торкнутися основи – це те як відбувається орієнтування карті і познайомитися з термінами довготи та широти.

Географічна широта – це відстань у градусах від екватора до точки, координати якої потрібно визначити. Географічна широта буває північною, якщо об’єкт розміщений північніше екватора, та південною, коли об’єкт розміщений південніше екватора. Екватор – це початок відліку географічної широти, тому його широта становить 0°. Кінцеві точки, до яких можна визначити широту, – це Північний та Південний полюси. Їхня географічна широта становить 90° відповідно північної і південної широти. Отже, географічна широта на Землі може змінюватися в інтервалі від 0° до 90°. Усі точки, розміщені на одній паралелі, мають однакову географічну широту [3].

Географічна довгота – це відстань у градусах від початкового меридіана до точки, координати якої треба визначити. Вона може бути східною, якщо об’єкт розміщений на схід від початкового меридіана, і західною, якщо об’єкт розміщений на захід від початкового меридіана. Початковий меридіан – це початок відліку географічної довготи і має довготу 0°. Оскільки вся земна куля має 360°, то в кожній з півкуль – Східній чи Західній – довгота може бути в інтервалі від 0° до 180°. Усі точки, розміщені на одному меридіані, мають однакову географічну довготу [4].

1.3 Огляд аналогів

Карти «Google» – безкоштовний картографічний веб-сервіс від компанії «Google». Станом на 2020 рік, сервісом користувалися 1 мільярд користувачів щомісяця[5].

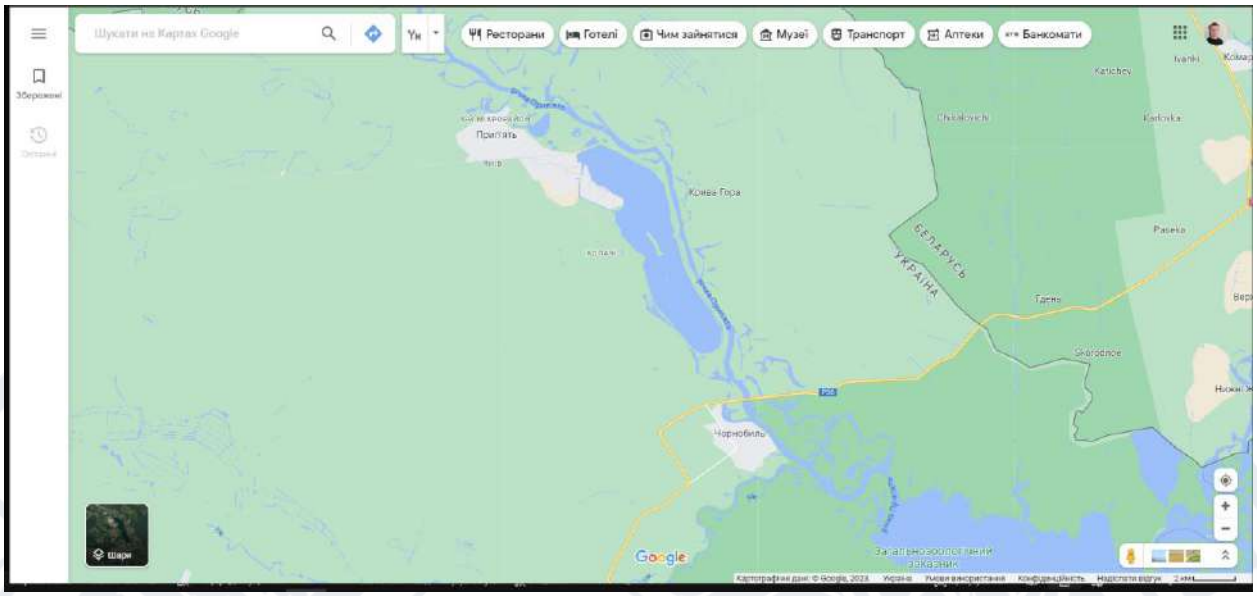

Рисунок 1.1 – Інтерфейс карт «Google»

Карти «Here»

«Here» – картографічний сервіс використовується в численних пакетах програмного забезпечення, включаючи системи навігації. Доступний для браузерів і телефонів на різних платформах. Карти включають такі функції, як пошук по мапі, супутникові мапи, побудова маршрутів, відображення пробок в реальному часі[6].

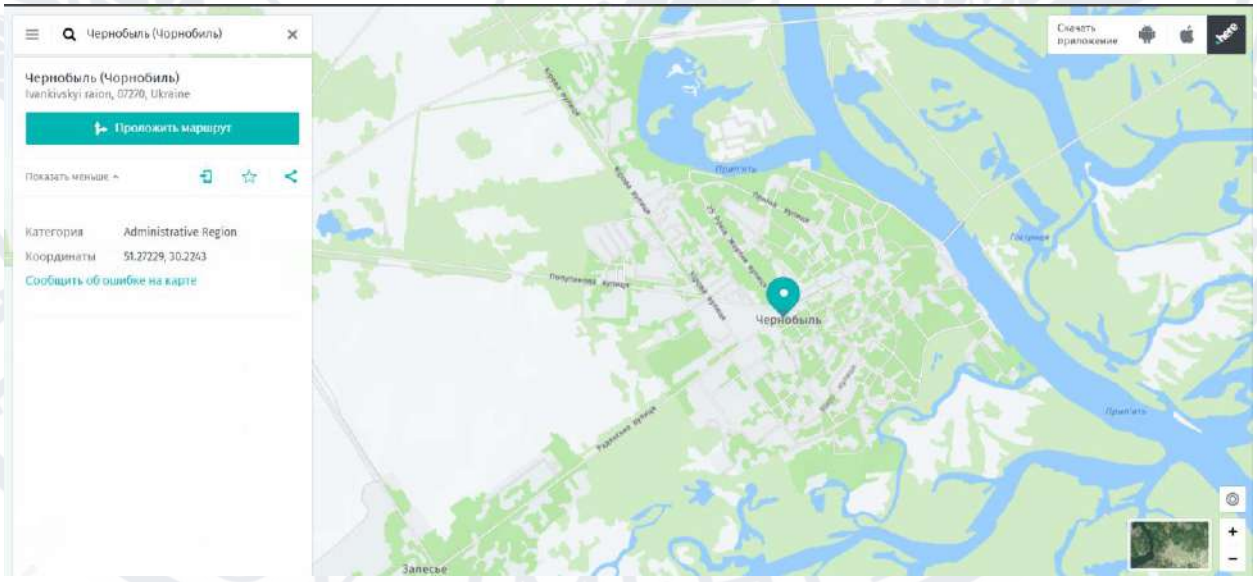

Рисунок 1.2 – Інтерфейс карт «Here»

Карти «Waze»

«Waze» – безкоштовний навігатор для мобільних пристроїв та комп’ютерів , що дає змогу відстежувати ситуацію на дорогах у режимі реального часу, прокладати оптимальні маршрути. Його особливість у тому, що завдяки йому можна дізнаватися про розташування радарів швидкості, камер фіксації. А найголовніший його функціонал в тому, що самі користувачі можуть попереджувати інших користувачів про ситуацію на дорогах.

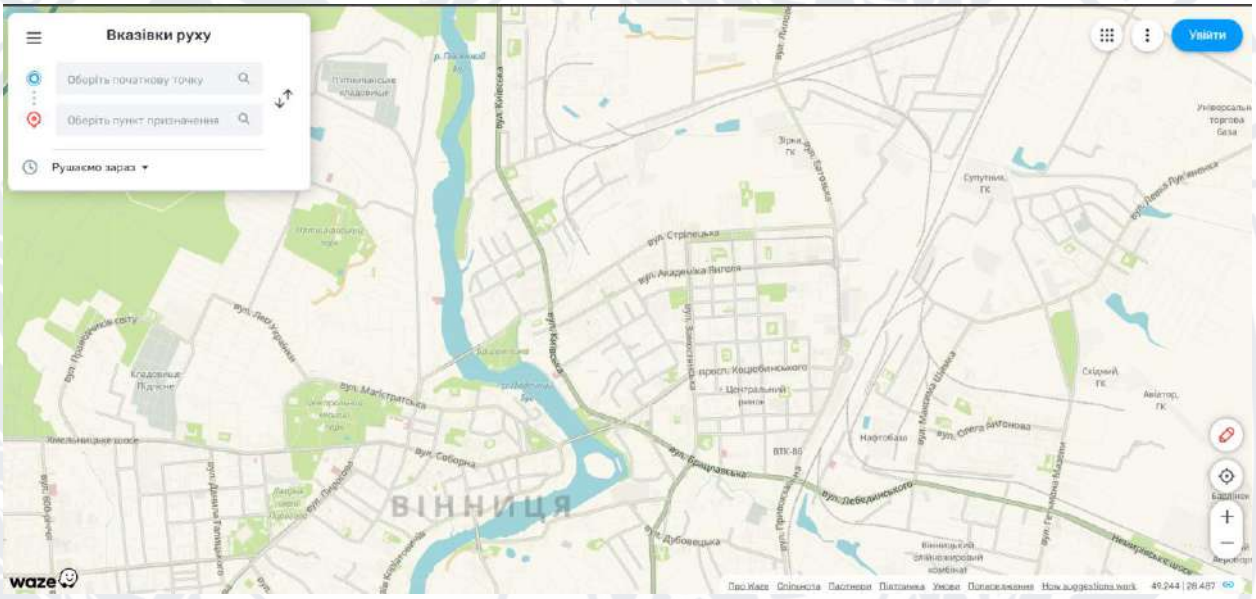

Рисунок 1.3 – Інтерфейс карт «Waze»

Висновок до розділу 1

В розділі була сформульована постановка задачі з розробки мобільного додатку для прокладання оптимального маршруту в місті в режимі реального часу, а також було проведено огляд сучасних систем навігації. Описано їх особливості та функціонал. Наступний розділ буде присвячений огляду інструментів для розробки та технологій.

РОЗДІЛ 2
ОГЛЯД ІНТРУМЕНТІВ ТА ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ

2.1 Редактор коду «Visual Studio Code»

Перше з чого починається розробка чого-небудь – це вибір «IDE». «IDE» – розшифровується як «Integrated Development Environment» або «інтегроване середовище розробки». Це набір інструментів, які дозволяють створювати програми. Іншими словами, якщо говорити дуже просто, то «IDE» – це програма, де створюються інші програми[7].

На мою суб’єктивну думку, «Visual Studio Code» – найкраще рішення для розробки. Ця розробка корпорації «Microsoft» має суттєві переваги перед конкурентами:

- Вона безкоштовна.
- Має зручний термінал для вводу команд з можливістю робити вкладки з різними терміналами та присвоювати їм імена.
- Вона не з’їдає багато ресурсів комп’ютера, а цей аспект важливий, тому що розробка буде вестись за допомогою ноутбуку.

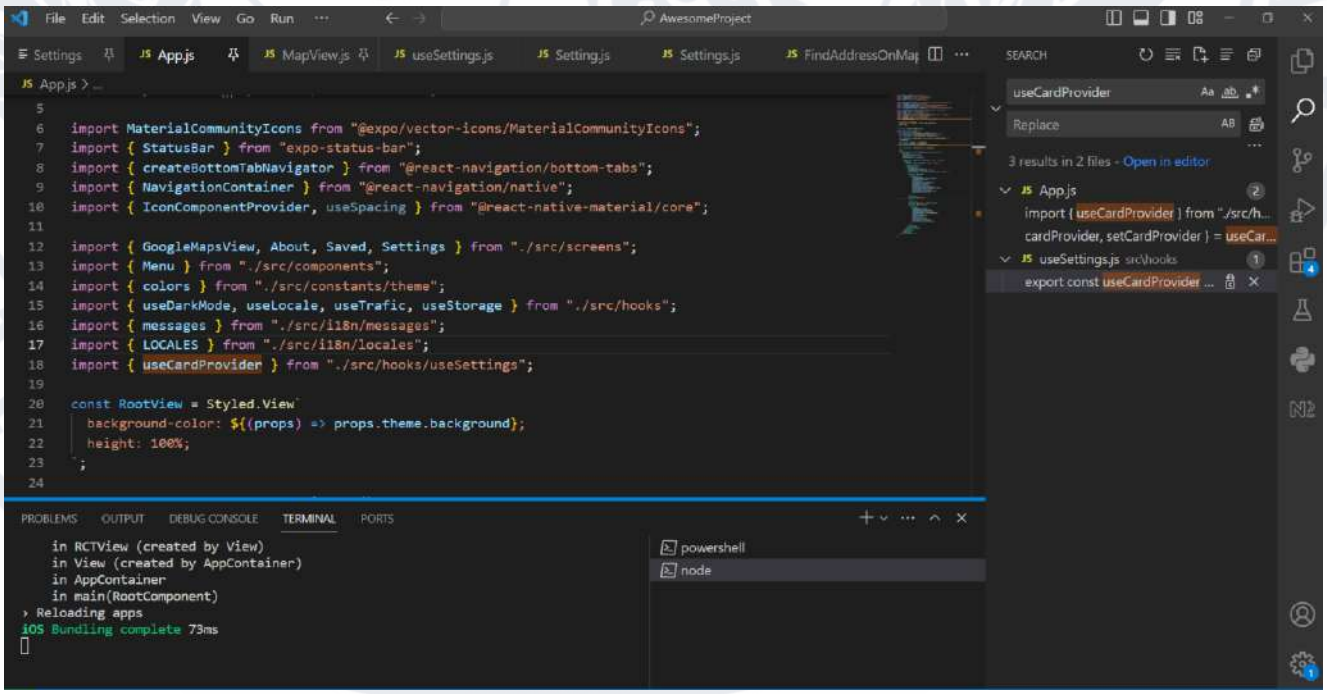

Рисунок 2.1 – інтерфейс «Visual Studio Code»

На рисунку 2.1 зображено запущений редактор. Інтерфейс складається з правого меню – де відображаються всі файли проекту, основний екран програми представлений полем для відображення вмісту файлів з можливістю перемикатися між файлами як між вкладками. Знизу знаходиться термінал для вводу команд, що дуже часто використовується при розробці. Є можливість розширення базового функціоналу за допомогою плагінів.

Якщо говорити науковою мовою, плагін – це компільований модуль, націлений на розширення можливостей будь-якої програми. Попросту він є доповненням, надбудовою, завдяки якій вже встановлена на комп'ютер програма стає краще для користувача [8].

Ще одним плюсом цього редактора є можливість створення хмарного акаунта та синхронізація налаштувань і плагінів між різними пристроями.

2.2 Бібліотека «React Native»

«React Native» – це кросплатформений фреймворк з відкритим вихідним кодом, написаний на «JavaScript». «React Native» використовується для створення додатків, як для мобільних систем «Android» та «iOS», так і для настільних операційних систем – «Windows» та «macOS». В останні роки бібліотека набула величезної популярності тому що: прискорює швидкість розробки, скорочує час виходу продукту на ринок, дає можливості покривати декілька платформ без додаткових вкладень. Дає можливість використовувати спільні рішення для мобільних та веб-додатків. Використовує JavaScript, що є найпопулярнішою мовою програмування у світі [9].

Компоненти «React» обгортають існуючий нативний код. «React» – це бібліотека, яка використовується для розробки інтерфейсів користувача в браузері [10]. Нативний код – це код цільової платформи, для якої буде вестися розробка. Це дає змогу створювати додатки для всіх платформ, використовуючи один синтаксис написання коду, що пришвидшує розробку.

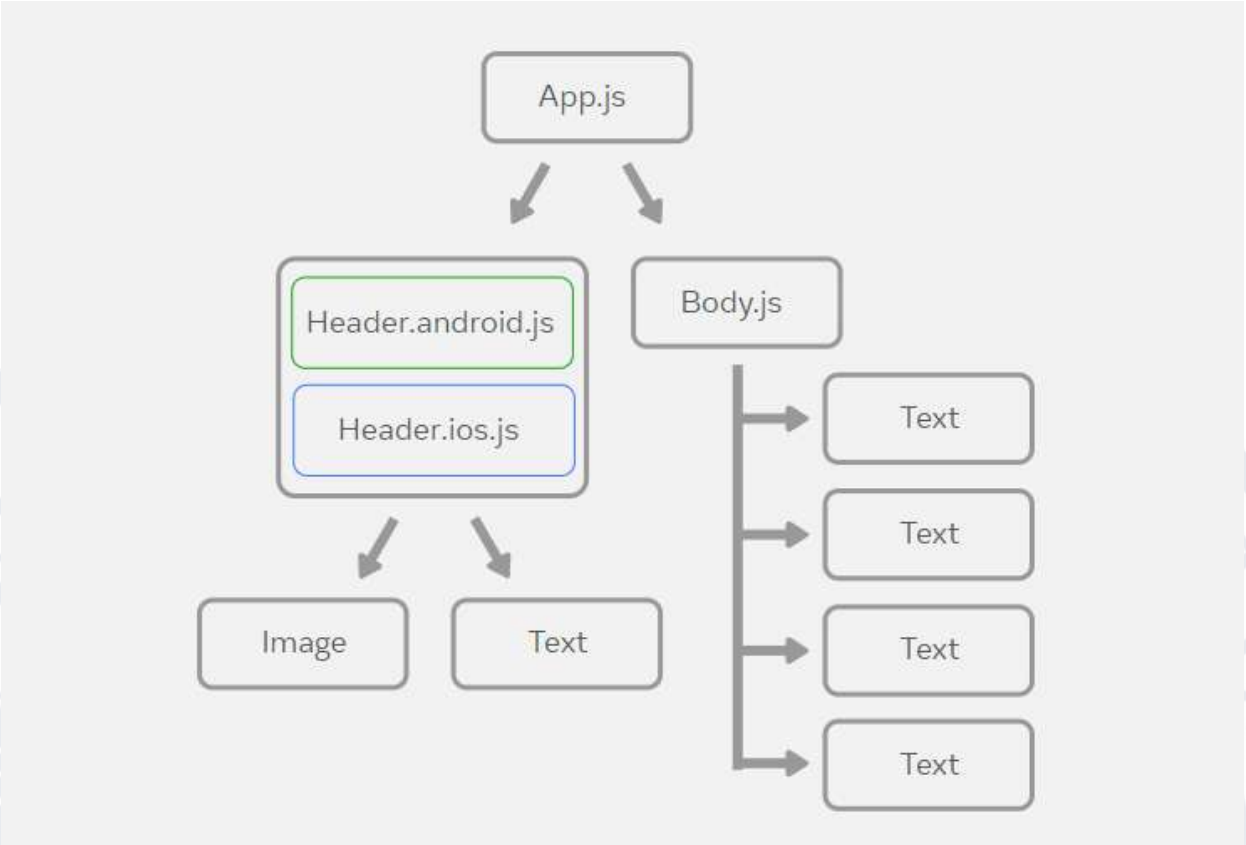

Рисунок 2.2 – Як працює «React Native»

2.3 Середовище «Expo»

«Expo» – це відкрите та безкоштовне фреймворк для розробки мобільних додатків на платформах «iOS» та «Android». Він надає розробникам інструменти та середовище для швидкої і зручної розробки додатків, використовуючи веб-технології, такі як «JavaScript», «React» та «React Native».

«Expo» надає набір готових компонентів і рішень, які дозволяють розробникам створювати функціональні додатки з мінімальними зусиллями. Він також має можливості для швидкого перегляду та тестування додатків на реальних пристроях, що спрощує процес розробки та пошуку помилок в програмі.

Особливістю «Expo» є його здатність до одночасної розробки для «iOS» та «Android», уникнення необхідності налаштовувати середовища розробки для кожної платформи окремо. Крім того, «Expo» надає зручні інструменти для розгортання та публікації додатків, що спрощує процес

релізу та оновлення.

Для початку роботи з «Expo» потрібно встановити на телефон додаток «Expo Go». Запустити програму через термінал і сканувати код через камеру телефону, на якому буде вестися розробка. Приклад такого коду зображено на рисунку 2.3.

Рисунок 2.3 – код для підключення додатку.

Потім на телефоні має з’явитися підключений розроблюваний додаток. Можна додавати безліч проектів, що дуже зручно та буває корисно у випадках, коли потрібно розробляти складні системи, які взаємопов’язані.

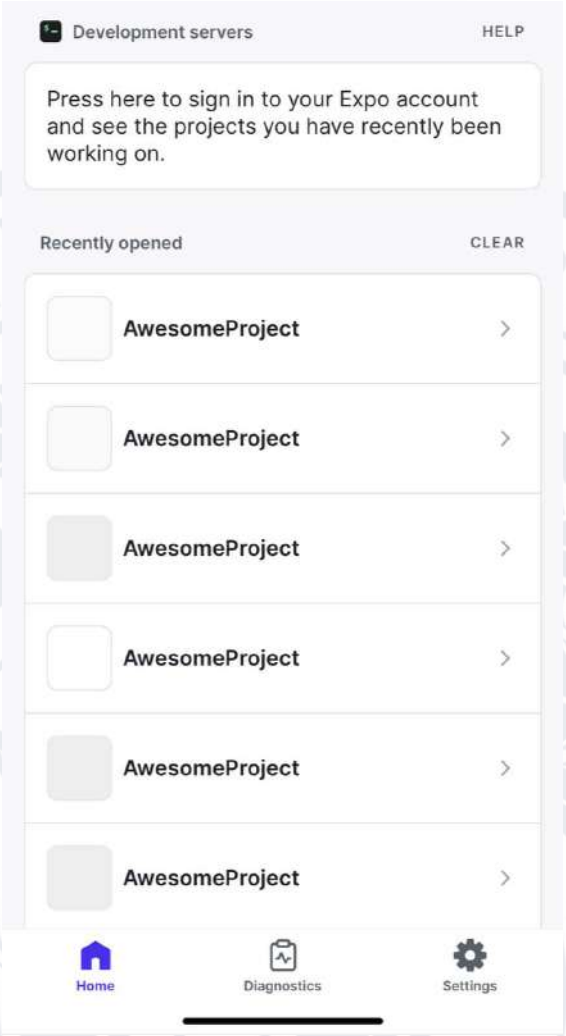

Рисунок 2.4 – Інтерфейс «Expo Go»

І можна вибрати і запустити додаток. Як бачимо, ця програма дозволяє в режимі реального часу запускати розроблювані додатки, що спрощує процес розробки та придає йому в швидкості.

2.4 Бібліотека «Styled components»

«Styled components» – це бібліотека для реактивного стилізування компонентів у «React»–додатках. Вона дозволяє використовувати «CSS» в декларативному стилі, вбудовуючи стилі безпосередньо в компоненти[11]. «Styled components» дозволяє створювати стилі прямо в JavaScript коді. Вона використовує спеціальний синтаксис, який дозволяє описувати стилі компонента за допомогою шаблонних рядків. Це дозволяє зберігати стилі біля компонентів, зробити їх легкими для управління та підтримки.

«Styled components» також підтримує використання динамічних стилів, змінюючи їх залежно від властивостей або стану компонента. Вона автоматично генерує унікальні класи для кожного компонента, що дозволяє уникнути конфліктів імен класів.

Перевагами її використання є більша чистота та організованість коду, більша переносимість стилів між компонентами, а також можливість використання всіх можливостей «JavaScript» для стилізації, таких як умовні оператори та цикли.

Узагальнюючи, «Styled components» – це бібліотека, яка дозволяє стилізувати компоненти у додатках за допомогою «CSS», вбудованого безпосередньо в «JavaScript» код. Вона забезпечує більшу організованість та переносимість стилів, а також використовує всі можливості «JavaScript» для динамічної стилізації.

2.5 Бібліотека «React Native Maps»

«React Native Maps» – це бібліотека для розробки мобільних додатків на платформі «React Native», яка надає можливості використання інтерактивних карт [12].

Ця бібліотека базується на популярних мапових сервісах, таких як «Google Maps» та «Apple Maps», і надає простий спосіб відображення та взаємодії з картами у «React Native» додатку.

«React Native Maps» дозволяє вставляти карту в компоненти вашого додатку, керувати розміщенням та масштабом, позначати мітки на карті, створювати полігони та шляхи, інтегрувати функції геолокації, відстежувати рух користувача та багато іншого.

Ця бібліотека надає простий «API» – інтерфейс для взаємодії з картами та мітками, а також можливості налаштування вигляду та стилізації карт. Вона також підтримує інтерактивні події, такі як клацання на мітку чи зміна розташування на карті.

2.6 Бібліотека «Expo Geolocation»

«Expo Geolocation» – це частина «Expo», яка надає доступ до геолокаційних можливостей пристроїв. Вона дозволяє додаткам отримувати інформацію про поточне місцезнаходження користувачів, використовуючи «GPS», мережу «Wi-Fi» або інші доступні джерела геоданих. «Expo Geolocation» дуже корисна для створення мобільних додатків, які потребують геолокаційного функціоналу, такого як мапи, служби доставки, місцевий пошук та інше [13].

2.7 useEffect та useState

«useEffect» та «useState» – це два дуже важливих хуки в бібліотеці «React», які використовуються для управління станом та ефектами в компонентах «React».

Хук – це певна функція, яка керує життєвим циклом компонента.

«useState» – це хук, який дозволяє компонентам «React» зберігати та оновлювати свій власний локальний стан. Використовується для створення змінних стану та їх оновлення в компонентах [14].

«useEffect» – це хук, який дозволяє виконувати певний код ефектів (наприклад, підписка на дані, маніпуляції з деревом відображення) після кожного рендеру компонента. Використовується для обробки побічних ефектів в компонентах [15].

Висновок до розділу 2

Отже, розробка програмного продукту, такого як додаток для прокладання оптимального маршруту в місті в режимі реального, потребує знання багатьох технологій та інструментів. Грамотне їх поєднання – запорука успішної розробки. Було оглянуто бібліотеки, які будуть використовуватися. Наведено їх переваги. Оглянуто середовища розробки. В наступному розділі буде розглянуто розробка додатку для прокладання та огляд інтерфейсу.

РОЗДІЛ 3
СТВОРЕННЯ ТА ОГЛЯД ПРОГРАМНОГО ПРОДУКТУ

3.1 Структура проєкту

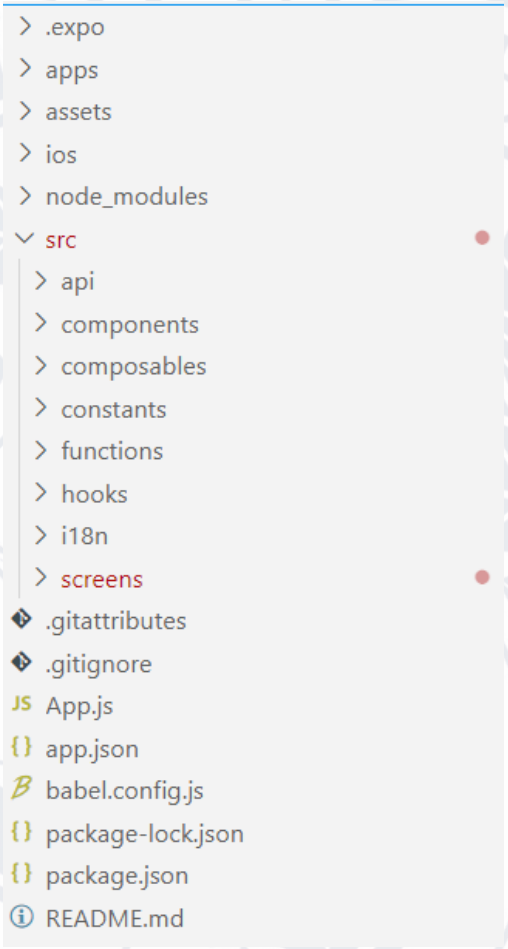

Рисунок 3.1 – структура проєкту

Структура проєкту представлена деревом папок і файлів. В папці «assets» зберігаються всі картинки, або ще якісь файли, наприклад шрифти. В папці «node_modules» знаходяться всі підключені бібліотеки. Файл «App.js» є точкою входу в нашу програму. «package.json» – файл конфігурації підключених бібліотек. В папці «src» зберігається вихідний код програми. В свою чергу там для зручності були створені наступні папки: «screens» – тут зберігаються всі сторінки програми. В папці «components» – окремі великі функціональні частини коду, які не є сторінками, наприклад, меню. В папці «hooks» зберігаються файли з корисними функціями.

3.2 Файл «App.js»

```
<ThemeProvider theme={colors[isDarkMode ? "dark" : "light"]}>
  <RootView>
    <StatusBar style={isDarkMode ? "light" : "dark"} />

    <NavigationContainer>
      <Tab.Navigator
        initialRouteName="Map"
        screenOptions={{ headerShown: false }}
        tabBar={(props) => !isNavigationMode && <Menu {...props} />}
      >
        <Tab.Screen name="Map">
          {() => (···
          )}
        </Tab.Screen>
        <Tab.Screen name="Saved">
          {() => (···
          )}
        </Tab.Screen>
        <Tab.Screen name="About" component={About} />
        <Tab.Screen name="Settings">
          {() => (···
          )}
        </Tab.Screen>
      </Tab.Navigator>
    </NavigationContainer>
  </RootView>
</ThemeProvider>
```

Рисунок 3.2 – «App.js»

На малюнку зображено код кореню додатка. Повний код файлу відображено в додатку А. В ньому за допомогою функції «App», яка повертає розмітку і відбувається побудова програми. За допомогою компонента «NavigationContainer», «Tab.Navigator», «Tab.Screen» було реалізовано механізм перемикання між сторінками програми. Сторінки навігатора представлені такими пунктами:

- «Map» – сторінка мапи.
- «Saved» – сторінка зі збереженими місцями.
- «About» – сторінка інформації про навігатор і автора.
- «Settings» – сторінка налаштувань

3.3 Головний екран програми

За замовчуванням відкривається сторінка з мапою. На ній видно назви вулиць, показано найближчі заклади, відображається поточне місцезнаходження.

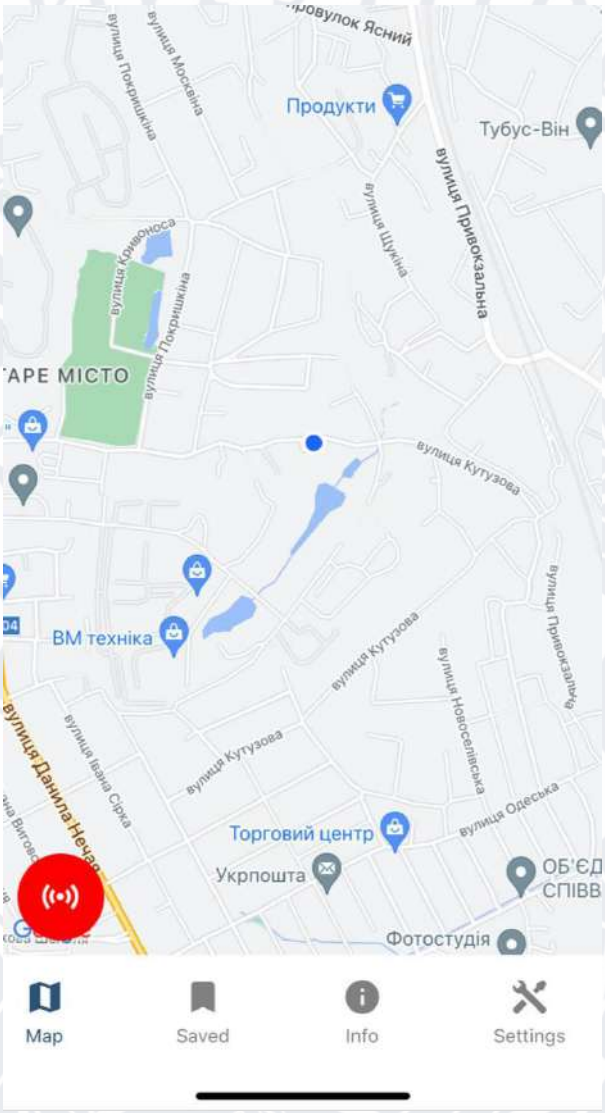

Рисунок 3.3 – Головний екран програми.

Внизу знаходиться меню для переходу між сторінками. Завдяки бібліотеці «React Native Maps», у нас є карта, з якою можна проводити різні маніпуляції – такі як приближення, віддалення та інше.

```
<MapView
  initialRegion={location.coords}
  maxZoomLevel={15}
  ref={mapRef}
  showsCompass={true}
  style={{ flex: 1, width: "100%" }}
  provider={PROVIDER_GOOGLE}
  customMapStyle={isDarkMode && darkMapStyles}
  showsTraffic={trafic}
  onRegionChange={(region) => {
    region.latitudeDelta &&
      setLocationRadius(region.latitudeDelta * 1000);
  }}
>
```

Рисунок 3.4 – компонента мапи в файлі «App.js»

Було задано стилі для мапи, що вона була на весь екран, вказано, що за основу хочемо використовувати карти від компанії «Google». Також було додано стилі для темної теми, задано параметр відображення компасу при повороті карти, передано дані початкового регіону задано максимальне приближення на карті. Також була проведена підписка на подію, коли користувач прокручує карту, це необхідно для масштабування компоненти поточної геопозиції. Повний код компоненти мапи викладено у додатку Б.

3.4 Реалізація темної теми

Уявити сучасний додаток без темної теми неможливо. Вона має безліч переваг – дозволяє вберегти зір користувача, заощадити заряд батареї пристрою з "AMOLED" екраном. "AMOLED" – розшифровується як Активна Матриця на Органічних Світлодіодах. Основна її перевага – вона може подавати електричний заряд на кожен піксель окремо. Це забезпечує глибокий чорний колір на екрані та економію заряду пристрою, тому що на пікселі, на які припадає чорний колір, електричний заряд не подається. Саме з вищевказаних причин реалізація темної теми не забаганка, а необхідність, особливо в такому енергозатратному застосунку для прокладання маршруту в режимі реального часу.

Розглянемо деталі реалізації навігатора в застосунку. Для початку

створимо об'єкт, в якому буде зберігатися значення чи активний темний режим і чи активна динамічна тема (тобто чи буде відображатися темна тема в залежності від обраної теми на телефоні).

```
5
6   const store = {
7     isDarkMode: true,
8     isAutoDarkMode: false,
9   };
```

Рисунок 3.5 – компонента мапи.

Потім нам необхідно якимось чином зберегти ці значення в пам’яті телефону і зчитувати. Для цього скористаємося раніше написаними функціями для зберігання у пам’яті. Наш створений вище об’єкт помістимо в поле «@themeSetting». За допомогою хука «useEffect» з параметром «[]» відслідковується коли програма тільки-но запустилася і всередині відбувається наступна логіка – в пам’яті телефону запитується поле налаштувань теми, якщо дане поле не існує, ми записуємо в нього об’єкт створений вище, якщо існує – то ми зберігаємо дані з телефону в локальну пам’ять компоненти, щоб було простіше оперувати з цими даними.

```
useEffect(() => {
  getStorageData("@themeSetting")
    .then((data) => {
      if (data === null) {
        setStorageData("@themeSetting", store);
      } else {
        setIsAutoDarkMode(data.isAutoDarkMode);
        setIsDarkMode(data.isDarkMode);
      }
    })
    .catch((data) => {
      console.log("failed", data);
    });
}, []);
```

Рисунок 3.6 – Оперування з пам’яттю телефону.

Далі було реалізовано відслідковування зміни локального сховища компоненти і оновлення в пам’яті телефону. Це потрібно для того, щоб потім додати можливість зміни цих параметрів. Для цього був створений окремий файл, повний код якого викладено у додатку В.

```
useEffect(() => {
  store.isDarkMode = isDarkMode;

  setStorageData("@themeSetting", store);
}, [isDarkMode]);
```

Рисунок 3.7 – Відслідковування зміни теми

Наступний крок – це створення перемикача налаштувань, за допомогою якого користувач би міг. Імпортуємо компоненту перемикач. І будемо змінювати параметр теми в залежності від положення вимикача. У додатку Г відображено весь вихідний код налаштувань.

```
<SettingItem>
  <LabelTextStyle>
    <FormattedMessage id="darkTheme" />
  </LabelTextStyle>
  <Switch
    trackColor={{ false: "#767577", true: "#4DD863" }}
    thumbColor={isDarkMode ? "#fff" : "#f4f3f4"}
    ios_backgroundColor="#3e3e3e"
    disabled={isAutoDarkMode}
    onValueChange={() => {
      setIsDarkMode((state) => !state);
    }}
    value={isDarkMode}
  />
</SettingItem>
```

Рисунок 3.8 – Код перемикача

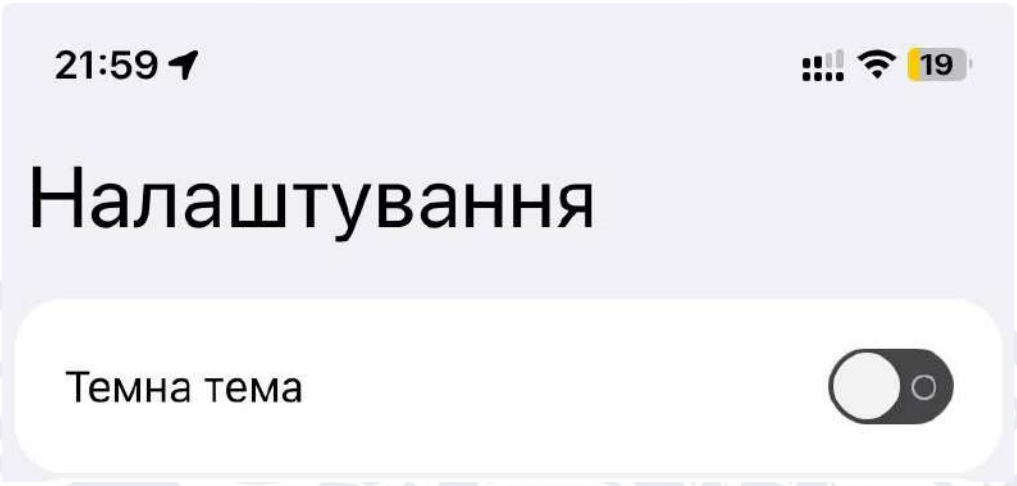

Рисунок 3.9 – Інтерфейс перемикача.

Заключним етапом є створення стилів для світлої та темної теми відповідно. Для цього створимо об’єкт, в якому будуть знаходитися основні кольори, якими ми будемо оперувати в темах.

```
export const colors = {
  dark: {
    primary: "#2B5278",
    background: "#212121",
    text: "#fff",
    background2: "#000",
    menuText: "gray",
    inputOutlined: "#000",
  },
  light: {
    primary: "#2B5278",
    background: "#fff",
    text: "#000",
    background2: "#F2F1F6",
    menuText: "gray",
    inputOutlined: "#eee",
  },
};
```

Рисунок 3.10 – об’єкт кольорів тем

Тобто всього в темі буде 6 кольорів, які будуть зчитуватися з 6

відповідних змінних, змінюючи значення змінних, буде мінятися і тема відповідно. Але для цього необхідно передати ці змінні в місце, де застосовуються стилі. Бібліотека «Styled components» підтримує такий функціонал.

Для початку необхідно обернути весь наш код у спеціальну компоненту, яка буде передавати значення змінної «theme» усім своїм дочірнім елементам.

```
>
  <ThemeProvider theme={colors[isDarkMode ? "dark" : "light"]}>···
  </ThemeProvider>
</IntlProvider>
```

Рисунок 3.11 – Передавання стилів теми

Далі ми орієнтуємося, чи активна темна тема, якщо так – передаємо стилі від темної, якщо ні – від світлої. Потім, коли ми будемо прописувати стилі, ми зможемо зчитати значення з «theme» і використовувати передані кольори і таким чином додаток стає більш гнучким, що дозволяє в майбутньому додати ще тем.

```
export const BodyText = Styled.Text`
  color: ${(props) => props.theme.text};
  font-size: 16px;
  line-height: 25px;
`;
```

Рисунок 3.12 – Використання змінної з кольором тексту в компоненті для стилізування звичайного тексту.

3.5 Автоматична темна тема

Для зручності користувача було реалізовано механізм автоматичної темної теми. Суть в тому, що в сучаних телефонах є темний режим і програма відстежує стан цього режиму і вмикає та вимикає темну тему в залежності від налаштувань телефону користувача. Для цього «React Native»

має функцію, яка дозволяє отримати поточну кольорову схему пристрою.

```
const colorShemeOfDevice = useColorScheme();
```

Рисунок 3.13 – Отримання кольорової схеми пристрою

Далі за допомогою вже відомого «useEffect» будемо відслідковувати зміну отриманих параметрів присвоювати в пам’ять телефону відповідні значення.

```
useEffect(() => {
  store.isAutoDarkMode = isAutoDarkMode;

  if (isAutoDarkMode) {
    store.isDarkMode = colorShemeOfDevice === "dark";
    setIsDarkMode(colorShemeOfDevice === "dark");
  }

  setStorageData("@themeSetting", store);
}, [isAutoDarkMode, colorShemeOfDevice]);
```

Рисунок 3.14 – Слідкування за зміною теми на телефоні

3.6 Локалізація додатку

Локалізація додатку – одна із найнеобхідніших у нас час. Це значно збільшую вибірку потенційних користувачів додатку. Тому було вирішено локалізувати додаток для наступних мов – англійська та українська. Локалізувати додаток будемо за допомогою зручної бібліотеки під назвою «React Intl».

Починається все з написання кодів культур. В нашому випадку української та англійської (США).

```
export const LOCALES = {
  ENGLISH: "en-US",
  UKRAINIAN: "uk-UA",
};
```

Рисунок 3.15 – Коди культур

Далі ми створюємо об’єкт, в якому будуть зберігатися варіанти перекладу для двох мов.

```
export const messages = {
  [LOCALES.ENGLISH]: {
    map: "Map",
    saved: "Saved",
    info: "Info",
    settings: "Settings",
    darkTheme: "Dark theme",
    autoDarkTheme: "Auto dark mode",
    language: "Language",
    aboutApplicationLabel: "About applicatior
    aboutApplicationText:
      "This application was developed by a st
    connectWithMeLabel: "Contacts:",
    trafic: "Trafic on road",
  },
  [LOCALES.UKRAINIAN]: {
    map: "Карта",
    saved: "Збережене",
    info: "Інфо",
    settings: "Налаштування",
    darkTheme: "Темна тема",
    autoDarkTheme: "Автоматична темна тема",
    language: "Мова",
```

Рисунок 3.16 – Варіанти перекладу для різних мов

Принцип тут схожий з темою, коли міняється культура в налаштуваннях, підтягуються переклади для відповідної культури, а в коді ми замість статичного тексту використовуємо спеціальну компоненту з

ключем.

```
<PageTitle>
  <FormattedMessage id="settings" />
</PageTitle>
```

Рисунок 3.17 – Приклад компоненти для перекладу.

В залежності від обраної мови ця компонента буде повертати текст з поля «settings» прописаний в ключах до кожної культури.

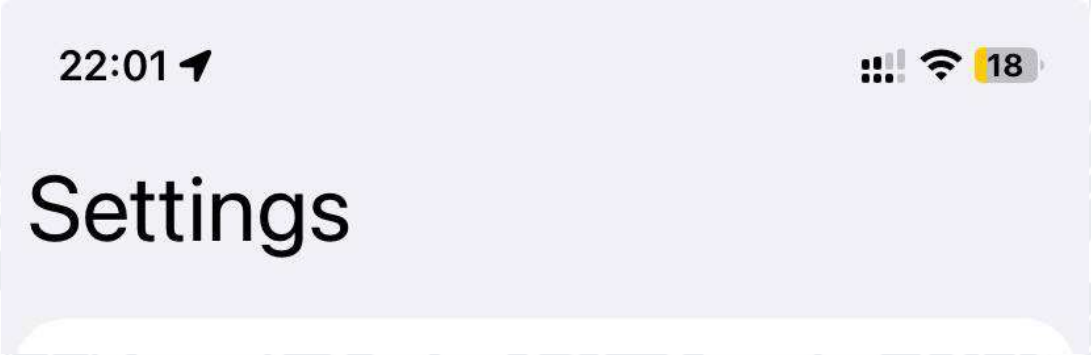

Рисунок 3.18 – Англійський переклад.

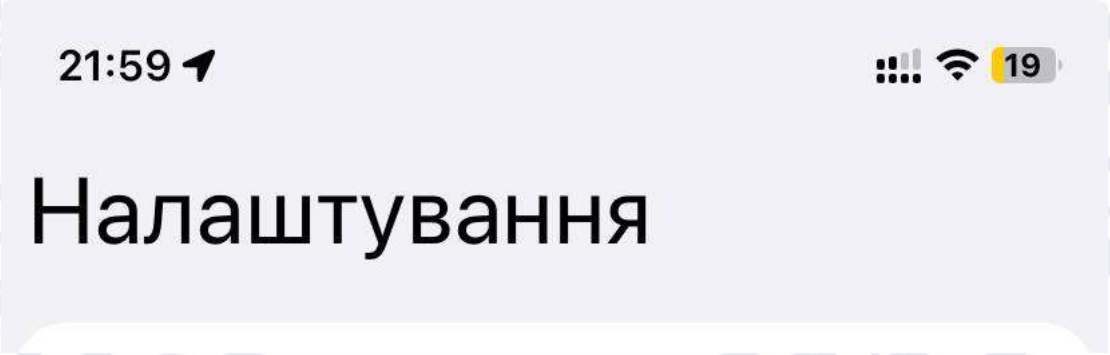

Рисунок 3.19 – Український переклад.

Реалізовано перемикання мови за допомогою компоненти, яку вже сама операційна система пропонує, це значно полегшує процес. При натисканні на поле вибору мови в налаштуваннях, з’являється спеціальне

меню вибору, де список мов, які підтримуються.

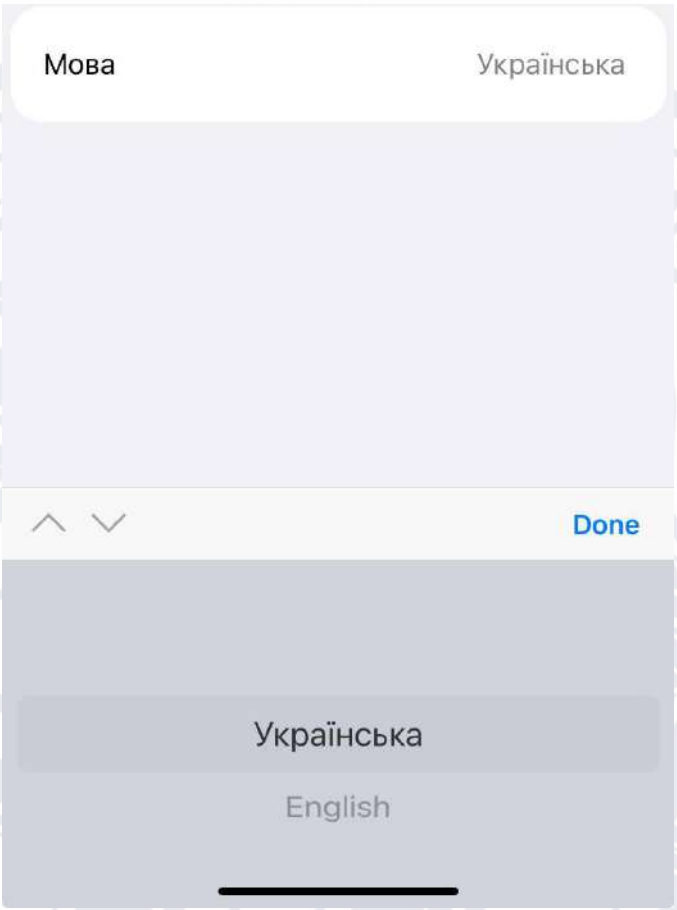

Рисунок 3.20 – Інтерфейс вибору мов.

Логіка зберігання поточної мови схожа з логікою зберігання поточної теми, тому повторюватися немає сенсу. Код наведений у додатку Г.

3.7 Компонента пошуку адреси

Певно одна з найважливіших частин будь-якого навігатора – це механізм пошуку адреси. Орієнтування на карті відбувається за допомогою довготи та широти, описаної в першому розділі. Суть проблеми в тому – щоби перетворити текст в координати довготи та широти. Для цього необхідна просто гігантська база даних, яку заповнити не так то просто. Тоді на допомогу приходять служби геокодингу. Геокодинг – процес перетворення адреси в координати і навпаки.

Для реалізації компоненти пошуку адреси була використана служба

«opencagedata». За допомогою бібліотеки «axios» ми посилаємо текст, який користувач введе в поле для пошуку адреси. Запит до цієї служби поверне нам список знайдених адреси. Якщо запит був успішний, то відбувається зберігання знайдених адрес в локальну пам'ять. Це відбувається в функції «then». З сервісу приходять координати («coordinates») та опис («formatted») знайдених місць у вигляді масиву. Його ми обертаємо в компоненти і показуємо.

```
export const MapAPI = {
  searchByAddress(address, setFoundAddresses, setIsPending) {
    setIsPending(true);
    const formattedAddres = encodeURI(address);

    instance
      .get(
        `https://api.opencagedata.com/geocode/v1/json?q=${addres
      )
      .then((res) => {
        setFoundAddresses(
          [...res.data.results].map((item) => ({
            formatted: item.formatted,
            coordinates: item.geometry,
          }))
        );

        setIsPending(null);
      })
      .catch((res) => {
        setIsPending(null);
      });
  },
};
```

Рисунок 3.21 – Посилання запиту на геокодинг

Знайдені адреси будуть відображатися нижче поля для вводу. За допомогою функції «map» відбувається те саме перетворення масиву в компоненти, які можна відобразити. Повний код всього що стосується компоненти пошуку адреси буде наведено у додатку Г.

```
{!isPending && (
  <ScrollView>
    {foundAddresses.map((address, index) => {
      return (
        <ResultItem
          {...address}
          key={index}
          setActivePoint={setActivePoint}
          setFoundAddresses={setFoundAddresses}
          setAddress={setAddress}
          animateToRegion={animateToRegion}
        />
      );
    })}
  </ScrollView>
)}
```

Рисунок 3.22 – Відображення знайдених адрес

Рисунок 3.23 – Інтерфейс пошуку адрес

При натисканні на знайдену адресу, користувача перекидає на місце на карті, де знаходиться адреса. І з’являється інформація про адресу, можливість закрити діалогове вікно, додати адресу до обраного або ж розпочати навігацію до цього місця.

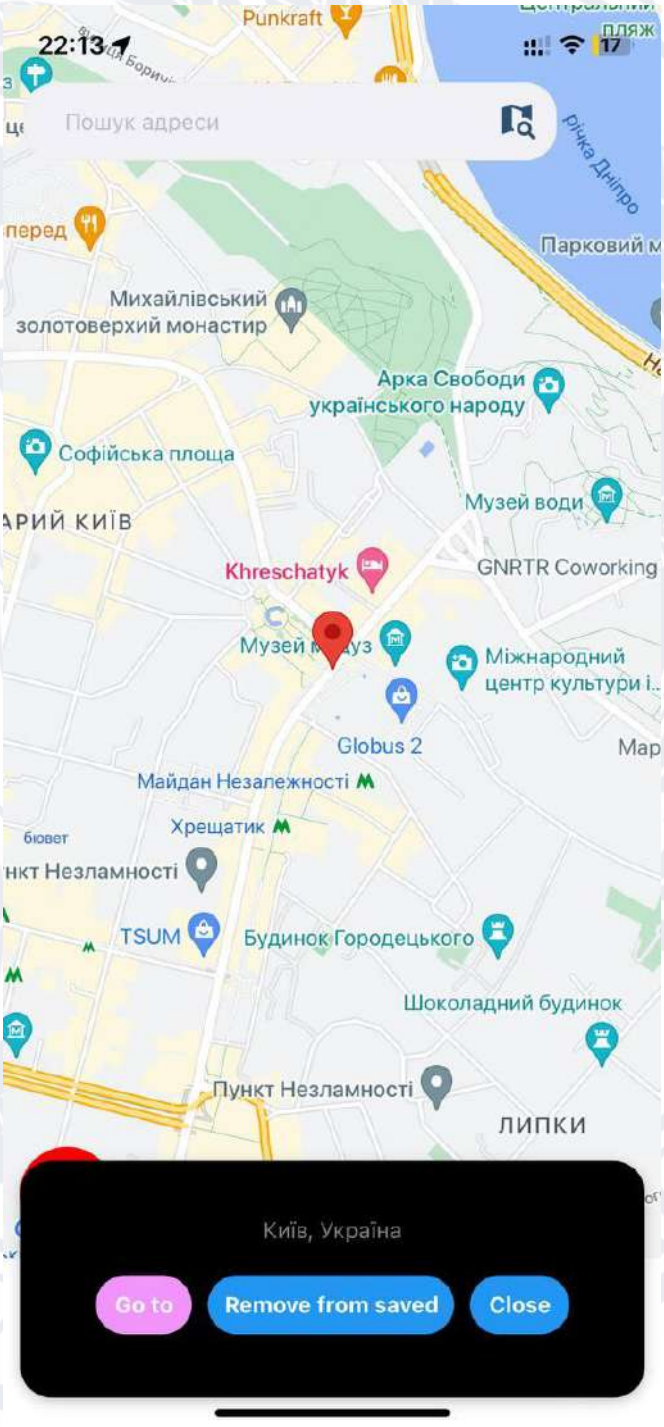

Рисунок 3.24 – Знайдена адреса

3.8 Компонента відображення місцезнаходження на карті

Важливою частиною кожного навігатора є процес відображення поточного місцезнаходження. Це реалізовано за допомогою описаної вище бібліотеки «expo location».

Перед першим запуском відбувається запит на отримання доступу до модуля розташування. Якщо доступ отримано успішно відбувається зберігання в локальну пам’ять, щоб потім на основі цього поля намалювати компонент на карті. Якщо ж користувач не наддасть доступ до геолокації, то висвітиться помилка.

```
useEffect(() => {
  (async () => {
    try {
      let { status } = await Location.requestForegroundPermissionsAsync();

      if (status !== "granted") {
        setErrorMsg("Permission to access location was denied");
        return;
      }

      let locationSubscription = await Location.watchPositionAsync(
        {
          accuracy: Location.Accuracy.BestForNavigation,
          timeInterval: 5000,
          distanceInterval: 10,
        },
        (newLocation) => {
          setLocation(newLocation);
        }
      );

      return () => {
        if (locationSubscription) {
          locationSubscription.remove();
        }
      };
    } catch (error) {
      setPending(false);
```

Рисунок 3.25 – Запит на доступ до місцезнаходження

Потім повертається місцезнаходження (довгота і широта) і за допомогою компоненти «Circle», в яку ми ці дані передаємо, відбувається відображення на карті поточного місцезнаходження. Воно відбувається у вигляді синього кола.

```
<Circle
  key={"fd"}
  center={{
    latitude: location?.coords?.latitude,
    longitude: location?.coords?.longitude,
  }}
  radius={locationRadius}
  strokeWidth={3}
  strokeColor={"#fff"}
  fillColor={"#1a66ff"}
  style={{ zIndex: 9999 }}
/>
```

Рисунок 3.26 – відображення поточного місцезнаходження

В інтерфейсі користувача це виглядає наступним чином.

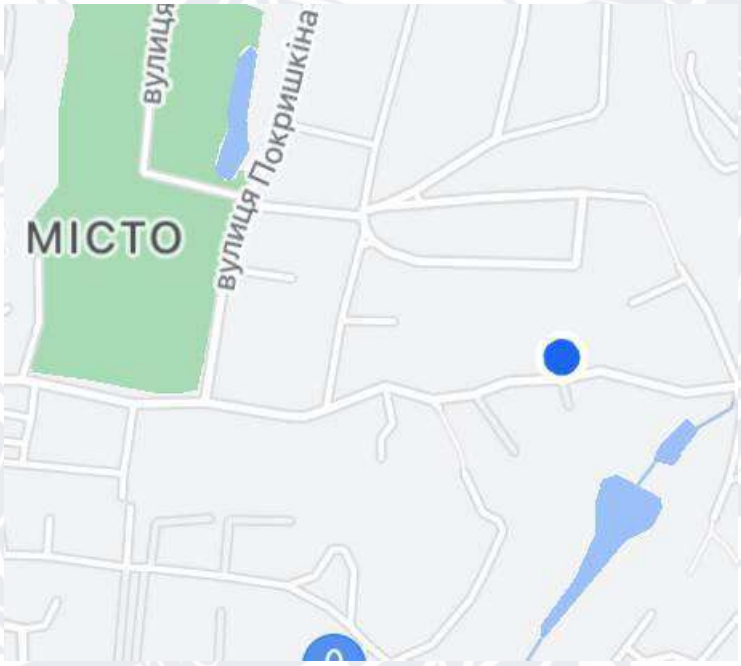

Рисунок 3.27 – Відображення місцезнаходження на карті.

3.9 Компонента повернення до поточного місцезнаходження

Зручно мати кнопку, яка б повертала на поточне місцезнаходження, тому було вирішено реалізувати і таку можливість. Спочатку створимо функцію, яка буде робити анімований перехід на передані координати на карті.

```
export const animateToRegion = (mapRef, { latitude, longitude }) => {
  mapRef.current.animateToRegion({ latitude, longitude }, 1000);
};
```

Рисунок 3.28 – Функція анімованого переходу по координатам

Потім створимо кнопку, при натисканні на яку буде відбуватися виклик функції з координатами поточного місцезнаходження.

```
<TouchableHighlight
  onPress={() => {
    animateToRegion(mapRef, intialRegion.coords);
  }}
>
  <GoToCurrentLocationButton>
    <Icon name="access-point" color="#fff" size={30} />
  </GoToCurrentLocationButton>
</TouchableHighlight>
```

Рисунок 3.29 – Код кнопки

В інтерфейсі це буде виглядати наступним чином.

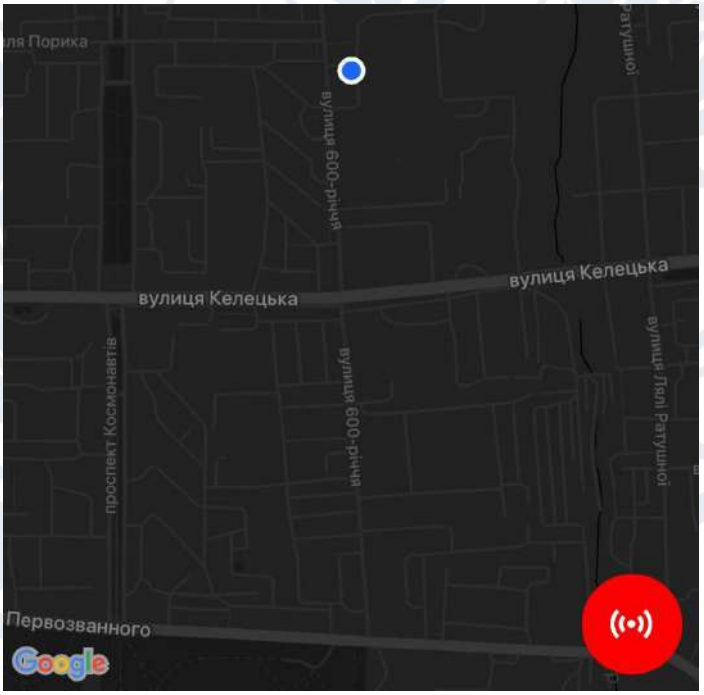

Рисунок 3.30 – Інтерфейс кнопки

3.10 Збережене

Для зручності використання програми, було зроблено механізм збереження адрес в список, щоб зберігати улюблені адреси і не витрачати час на їх пошук. Так само відбувається процес зберігання в пам’ять телефону та відслідковування зміни.

```
export const useStorage = () => {
  const [savedPlaces, setSavedPlaces] = useState([]);

  useEffect(() => {
    getStorageData("@saved").then((saved) => {
      if (saved) {
        setSavedPlaces(saved);
      } else {
        setStorageData("@saved", []);
      }
    });
  }, []);

  useEffect(() => {
    setStorageData("@saved", savedPlaces);
  }, [savedPlaces]);

  return { savedPlaces, setSavedPlaces };
};
```

Рисунок 3.31 – Зберігання адреси в пам’яті телефону

Потім на окремій сторінці було створено відображення збережених адрес з можливістю видалення або переходу по ним.

```
<SavedItemStyled>
  <TouchableOpacity
    onPress={() => {
      setActivePoint({ formatted, coordinates });
      navigation.navigate("Map");
    }}
  >
    <WrapperIcon>
      <Icon name="content-save-outline" color="gray" size={30} />
    </WrapperIcon>

    <BodyText>{formatted}</BodyText>
  </TouchableOpacity>
  <TouchableOpacity
    onPress={() =>
      createTwoButtonAlert("Do really wants to delete it ?", () => {
        const temp = [...savedPlaces].filter((item) => item.id !== id);

        setSavedPlaces(temp);
      })
    }
  >
    <Icon name="delete" color="red" size={30} />
  </TouchableOpacity>
</SavedItemStyled>
```

Рисунок 3.32 – Код збережених адрес.

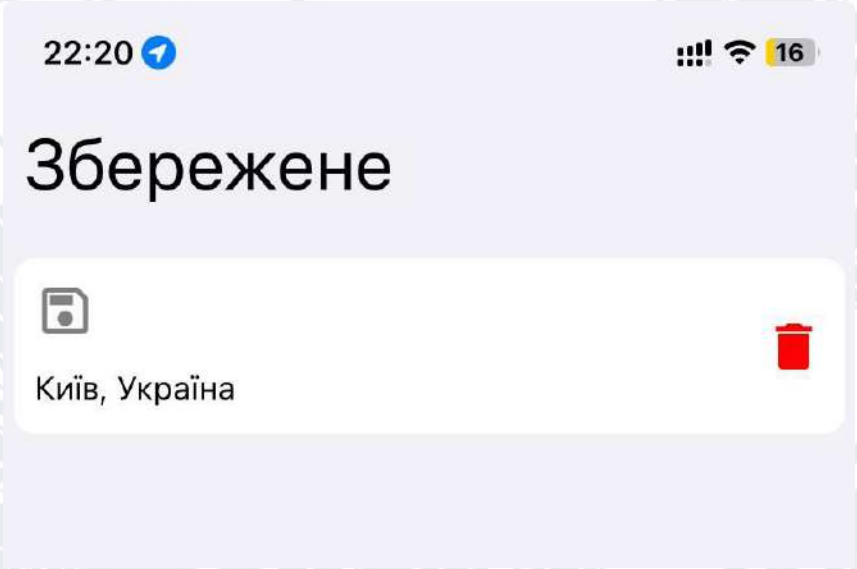

Рисунок 3.33 – Список збереженого.

Інтерфейс зображений на рисунку 3.33. При натисканні кнопки видалити з’являється модальне вікно, яке вимагає підтвердити дію.

3.11 Інформація

Було реалізовано сторінку з інформацією, де є короткі відомості про додаток, а також кнопки для зв’язку з розробником. Все було зроблено за допомогою простих текстових компонентів та посилань, на яких не має сенсу зупинятися детально. Повний код буде наведений у додатку Д.

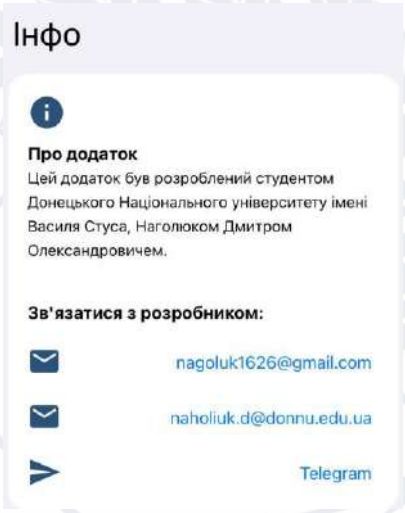

Рисунок 3.35 – Інформація про додаток

3.12 Інтерфейс налаштувань

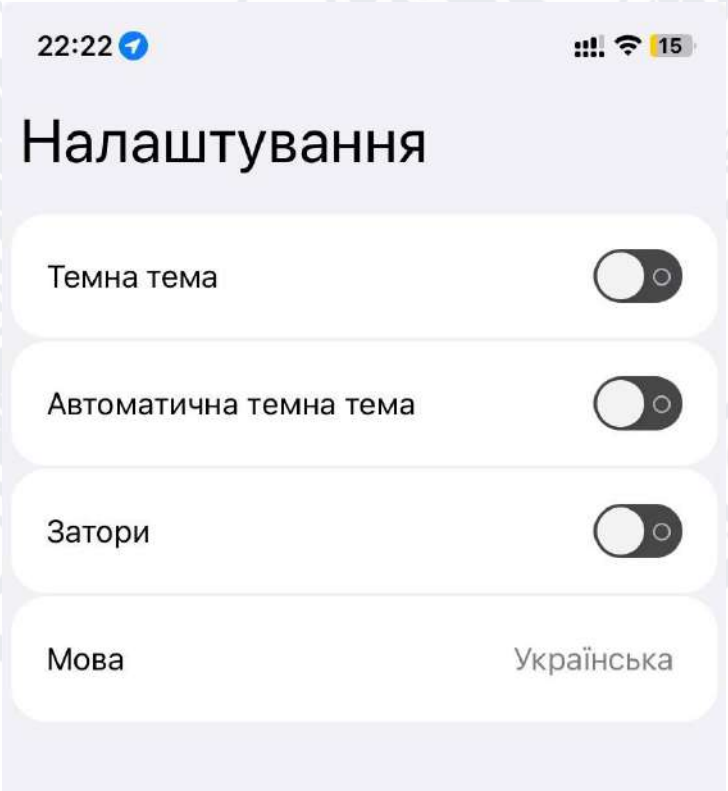

Рисунок 3.36 – Інтерфейс налаштувань

Варто звернути увагу на те, що коли вмикається автоматична темна тема, то поле вибору теми блокується. На тестовому телефоні було увімкнута біла тема, тож коли було активовано режим автоматичної темної теми, поле темна тема стало неактивним і в додатку активувався світлий режим.

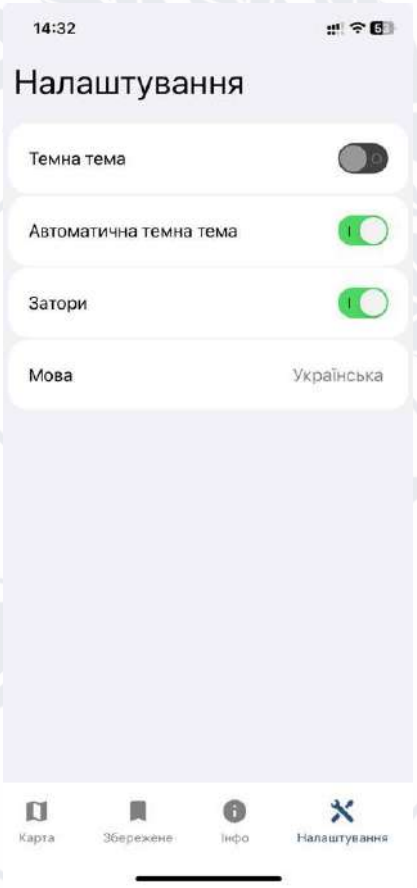

Рисунок 3.37 – Світлий режим

Бібліотека карт пропонує функцію відображення заторів на дорогах, на жаль, на території України зараз це не працює, але на території інших країн – так. Було вирішено зробити цей перемикач, щоб можна було вмикати та вимикати функцію заторів на дорогах. Код який за це відповідає наведений у додатку Б і є не дуже великим – «showsTraffic={trafic}». Одне поле, яке потрібно задати в конфігурацію мапи, приймає значення «true» або «false».

Рисунок 3.38 – Функція заторів на дорогах

3.13 Режим навігації

Було створено спеціальний режим навігації до вибраної точки. Після того як користувач введе адресу і натисне кнопку їхати до вибраної точки, йому будується маршрут за допомогою сервісу «Google direction api» [16]. Маршрут динамічно оновлюється, якщо користувач випадково зверне не туди.

В коді це виглядає таким чином. Ми передаємо в спеціальну компоненту пункт призначення та нашу поточну геопозицію. Ця компонента будує маршрут від нашої точки. Це все буде відображатися при умові, що режим навігації увімкнений та заданий пункт призначення. Під час переміщення користувача також переміщується і карта, щоб компонента місцезнаходження була в фокусі. Змінна «isNavigationMode» керує тим чи показувати певні компоненти чи ні. За допомогою компоненти «MapViewDirections» відбувається прокладання маршруту. Все що потрібно

для прокладання маршруту це передати пункт призначення, відправлення, ключ для доступу до сервісу прокладання оптимального маршруту. У додатку Б викладено повний код.

```
{finalLocation && isNavigationMode && (
  <MapViewDirections
    origin={location.coords}
    destination={finalLocation}
    apikey={GOOGLE_MAPS_APIKEY}
    strokeColor="hotpink"
    strokeWidth={5}
  />
)}
```

Рисунок 3.39 – Прокладання маршруту код

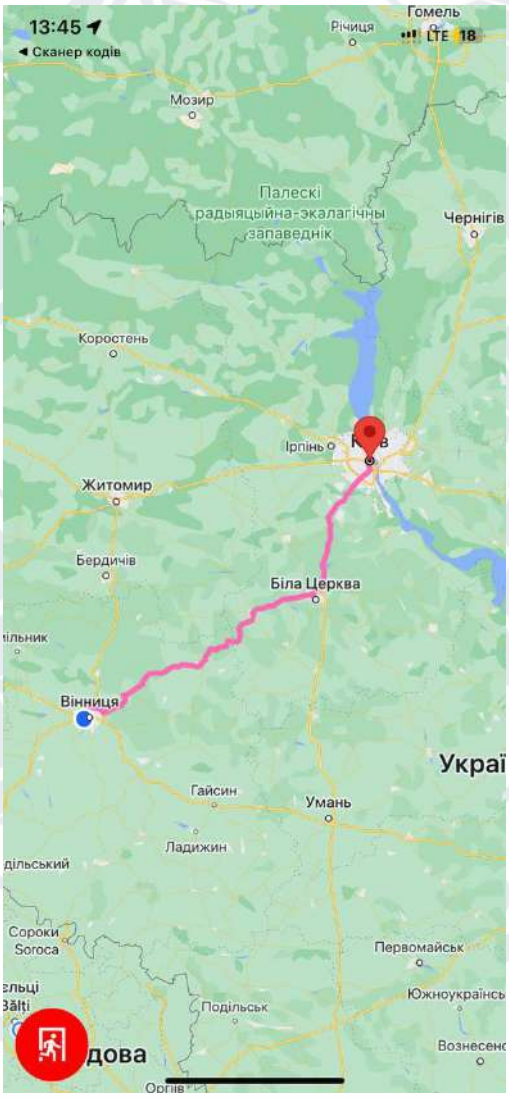

Рисунок 3.40 – Прокладання маршруту до міста Київ

Під час режиму навігації багато чого приховується – компонента пошуку адреси, меню. Замість кнопки повернення до поточної геопозиції відображається кнопка виходу з режиму навігації. Після натиснення на яку відбувається повернення компонентів пошуку адреси та меню.

3.14 Вибір постачальника карт

Унікальним цей додаток буде робити те, що користувач зможе обирати постачальника карт. Це було реалізовано, додавши в налаштування вибір постачальника карт. Механізм збереження вибраного значення у пам’яті телефону схожий з механізмом збереження мови, викладено у додатку В.

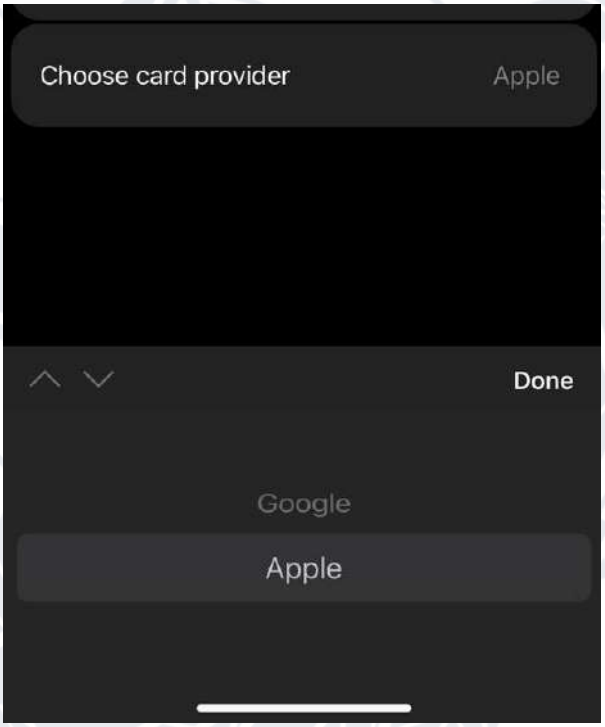

Рисунок 3.41 – Налаштування постачальника карт

Обравши іншого постачальника карт і повернувшись на початкову сторінку користувач буде бачити нові карти. Це дозволяє бути користувачу більш гнучким використовувати переваги того чи іншого постачальника карт.

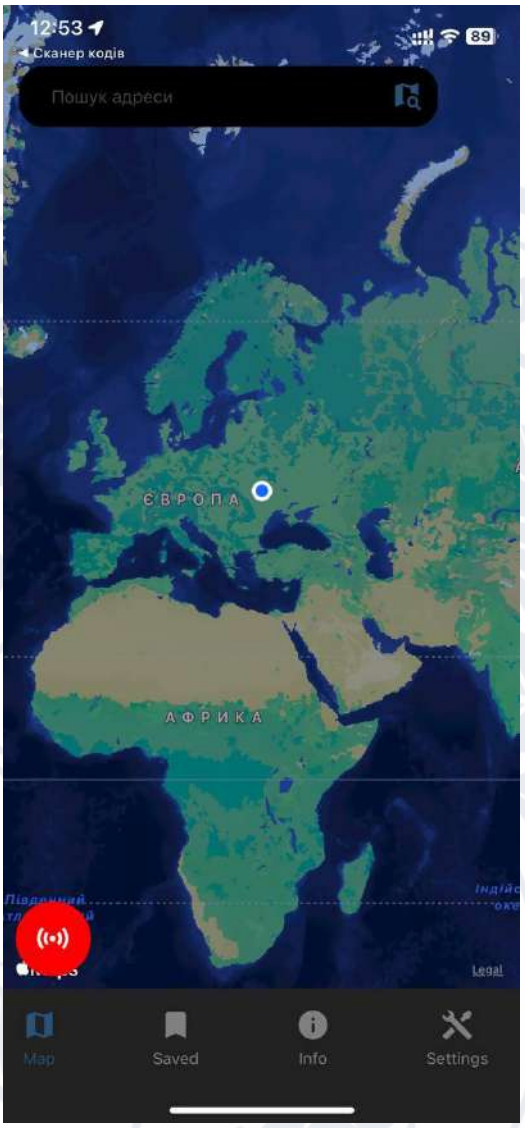

Рисунок 3.42 – Додаток з постачальником карт від «Apple»

Висновок до розділу 3

В даному розділі було висвітлено основні процеси розробки додатку для прокладання оптимального маршруту в місті в режимі реального часу. Було проведено огляд важливих частин коду з поясненнями. Розглянуто результат програмування та зроблено огляд основних можливостей додатку.

ВИСНОВКИ

В ході виконання магістерської роботи було розроблено додаток для прокладання маршруту в місті в режимі реального часу. Вивчено методику розробки сучасних програм для мобільних пристроїв.

Спроектована система складається з чотирьох ключових екранів, кожен з яких виконує важливі завдання та доповнює загальну ідею додатка.

Перший екран, представлений інтерактивною мапою, яка надає користувачеві зручні інструменти для визначення свого місцезнаходження, а також має ряд функцій, що полегшують навігацію та пошук адрес. Модальне вікно, що виводиться при виборі адреси з пошуку, дозволяє користувачеві здійснити різноманітні дії - додавання адреси у вибране та навігацію до вибраної точки.

Другий екран присвячений списку улюблених місць, де користувач може переглядати та управляти вибраними адресами – видаляти, відкривати їх на карті Це дозволяє ефективно організовувати та відстежувати улюблені місця.

Третій екран містить інформацію про розробника та способи зв'язку, що створює прямий механізм обміну інформацією між користувачем та розробником.

Четвертий екран - налаштувань дозволяє користувачеві персоналізувати додаток згідно зі своїми уподобаннями, вибираючи тему, автоматичний вибір теми, показ заторів та мову інтерфейсу.

Загальний функціонал доповнюється можливістю вибору постачальника карт, що робить додаток унікальним та готовим задовольнити різноманітні потреби користувачів у сфері навігації. Цей функціонал робить його адаптованим до різних вимог та передових тенденцій в сфері навігації, виходячи за межі звичайних засобів. Користувачу надається можливість обрати серед низки постачальників карт того, який найкраще відповідає його потребам та перевагам,

дозволяючи створити індивідуальний досвід навігації.

Отже, можливість вибору постачальника карт відзначає додаток як унікальний та готовий задовольнити різноманітні потреби та вподобання користувачів у сфері навігації.

Усі ці аспекти додатку формують інтегровану систему, яка сприяє ефективній та зручній навігації в місті, визначаючи його конкурентоспроможність та привабливість на ринку мобільних додатків.

Було продемонстровано інтерфейс програми, описано на яких технологіях вона була побудована. Розроблена в ході даної роботи програма суттєво розширила мій кругозір як розробника.

Всі ці функції та можливості роблять розроблений додаток інноваційним та конкурентоспроможним на ринку мобільних додатків для навігації.

.

СПИСОК ЛІТЕРАТУРИ

1. Навігаційна система [Електронний ресурс] – Режим доступу до ресурсу: https://www.wikidata.uk-ua.nina.az/%D0%9D%D0%B0%D0%B2%D1%96%D0%B3%D0%B0%D1%86%D1%96%D0%B9%D0%BD%D0%B0_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0.html
2. What is geocoding [Електронний ресурс] – Режим доступу до ресурсу: https://desktop.arcgis.com/en/arcmap/latest/manage-data/geocoding/what-is-geocoding.htm
3. Модальне вікно [Електронний ресурс] – Режим доступу до ресурсу: https://www.wikiwand.com/uk/%D0%9C%D0%BE%D0%B4%D0%B0%D0%BB%D1%8C%D0%BD%D0%B5_%D0%B2%D1%96%D0%BA%D0%BD%D0%BE
4. Географічна координати [Електронний ресурс]] – Режим доступу до ресурсу: https://vshkole.in.ua/geografichni-koordinati/
5. Карти «Google» [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/%D0%9A%D0%B0%D1%80%D1%82%D0%B8_Google
6. Карти «Here» [Електронний ресурс] – Режим доступу до ресурса: https://uk.wikipedia.org/wiki/Here
7. Що таке IDE та SDK? "Страшні" терміни простими словами [Електронний ресурс] – Режим доступу до ресурсу: https://apix-drive.com/ua/blog/useful/ide-i-sdk-strashnye-terminy-prostymi-slovami
8. Що таке плагіни і де вони застосовуються ? [Електронний ресурс] – Режим доступу до ресурсу: https://moyaosvita.com.ua/internet/shho-take-plagini-i-de-voni-zastosovuyutsya/
9. React Native ретрейнінг-програма [Електронний ресурс] – Режим доступу до ресурсу: https://careers.epam.ua/events/react-native-retraining-01022022
10. What is React. Автор оригіналу: Alesia Sirotka. URL: https://flatlogic.com/blog/what-is-react/

11. Styled components [Електронний ресурс] – Режим доступу до ресурсу: https://styled-components.com/
12. Map View. [Електронний ресурс] – Режим доступу до ресурсу: https://docs.expo.dev/versions/latest/sdk/map-view/
13. Expo Geolocation Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.expo.dev/versions/latest/sdk/location/
14. useState [Електронний ресурс] – Режим доступу до ресурсу: Using the State Hook – React (reactjs.org)
15. useeffect [Електронний ресурс] – Режим доступу до ресурсу: Using the Effect Hook – React (reactjs.org)
16. Google directions api [Електронний ресурс] – Режим доступу до ресурсу: https://developers.google.com/maps/documentation/directions/overview?hl=ua

ДОДАТОК А

Код файлу «App.js»

```
import React, { useState } from "react";
import { ThemeProvider } from "styled-components";
import { IntlProvider } from "react-intl";
import Styled from "styled-components/native";

import MaterialCommunityIcons from "@expo/vector-icons/MaterialCommunityIcons";
import { StatusBar } from "expo-status-bar";
import { createBottomTabNavigator } from "@react-navigation/bottom-tabs";
import { NavigationContainer } from "@react-navigation/native";
import { IconComponentProvider, useSpacing } from "@react-native-material/core";

import { GoogleMapsView, About, Saved, Settings } from "./src/screens";
import { Menu } from "./src/components";
import { colors } from "./src/constants/theme";
import { useDarkMode, useLocale, useTrafic, useStorage } from "./src/hooks";
import { messages } from "./src/i18n/messages";
import { LOCALES } from "./src/i18n/locales";
import { useCardProvider } from "./src/hooks/useSettings";

const RootView = Styled.View`
  background-color: ${(props) => props.theme.background};
  height: 100%;
`;

const Tab = createBottomTabNavigator();

export default function App() {
  const [activePoint, setActivePoint] = useState(null);
  const [isNavigationMode, setIsNavigationMode] = useState(false);
  const [finalLocation, setFinalLocation] = useState(null);

  const { isDarkMode, setIsDarkMode, isAutoDarkMode, setIsAutoDarkMode } =
    useDarkMode();

  const { locale, setLocale } = useLocale();
  const { trafic, setTrafic } = useTrafic();
  const { cardProvider, setCardProvider } = useCardProvider();

  const { savedPlaces, setSavedPlaces } = useStorage();

  return (
    <IconComponentProvider IconComponent={MaterialCommunityIcons}>
      <IntlProvider
        messages={messages[locale]}
        locale={locale}
        defaultLocale={LOCALES.ENGLISH}
      >
        <ThemeProvider theme={colors[isDarkMode ? "dark" : "light"]}>
          <RootView>
            <StatusBar style={isDarkMode ? "light" : "dark"} />

            <NavigationContainer>
              <Tab.Navigator
                initialRouteName="Map"
                screenOptions={{ headerShown: false }}
```

```
                tabBar={(props) => !isNavigationMode && <Menu {...props} />}
              >
                <Tab.Screen name="Map">
                  {() => (
                    <GoogleMapsView
                      isNavigationMode={isNavigationMode}
                      isDarkMode={isDarkMode}
                      trafic={trafic}
                      setSavedPlaces={setSavedPlaces}
                      savedPlaces={savedPlaces}
                      activePoint={activePoint}
                      setActivePoint={setActivePoint}
                      setIsNavigationMode={setIsNavigationMode}
                      finalLocation={finalLocation}
                      setFinalLocation={setFinalLocation}
                      cardProvider={cardProvider}
                    />
                  )}
                </Tab.Screen>
                <Tab.Screen name="Saved">
                  {() => (
                    <Saved
                      savedPlaces={savedPlaces}
                      setSavedPlaces={setSavedPlaces}
                      activePoint={activePoint}
                      setActivePoint={setActivePoint}
                    />
                  )}
                </Tab.Screen>
                <Tab.Screen name="About" component={About} />
                <Tab.Screen name="Settings">
                  {() => (
                    <Settings
                      isDarkMode={isDarkMode}
                      setIsDarkMode={setIsDarkMode}
                      isAutoDarkMode={isAutoDarkMode}
                      setIsAutoDarkMode={setIsAutoDarkMode}
                      locale={locale}
                      setLocale={setLocale}
                      trafic={trafic}
                      setTrafic={setTrafic}
                      cardProvider={cardProvider}
                      setCardProvider={setCardProvider}
                    />
                  )}
                </Tab.Screen>
              </Tab.Navigator>
            </NavigationContainer>
          </RootView>
        </ThemeProvider>
      </IntlProvider>
    </IconComponentProvider>
  );
}
```

ДОДАТОК Б

Код екрану мапи

```
import { Text } from "react-native";
import { useEffect, useRef, useState } from "react";
import MapView, { PROVIDER_GOOGLE, Circle, Marker } from "react-native-maps";
import MapViewDirections from "react-native-maps-directions";

import { animateToRegion } from "../functions/antimateToRegion";
import { updateCurrentGeolocation } from "../hooks";
import { darkMapStyles } from "../constants";
import { ScreenWapper, RelativeBlock, ButtonsOnMap } from "../components";
import { FindAddressOnMap, AddressModal } from "../composables";

export const GoogleMapsView = ({
  isDarkMode,
  trafic,
  setSavedPlaces,
  savedPlaces,
  activePoint,
  setActivePoint,
  isNavigationMode,
  setIsNavigationMode,
  finalLocation,
  setFinalLocation,
  cardProvider,
}) => {
  const [isInitialRegionAnimated, setIsInitialRegionAnimated] = useState(false);
  const mapRef = useRef(null);
  const [locationRadius, setLocationRadius] = useState(100);
  const location = updateCurrentGeolocation();

  useEffect(() => {
    if (location) {
      isNavigationMode && animateToRegion(mapRef, location.coords);
    }
  }, [location?.coords?.latitude, location?.coords?.longitude]);

  useEffect(() => {
    if (activePoint) {
      animateToRegion(mapRef, {
        latitude: activePoint.coordinates.lat,
        longitude: activePoint.coordinates.lng,
      });
    }
  }, [activePoint]);

  useEffect(() => {
    if (!isInitialRegionAnimated && location) {
      animateToRegion(mapRef, location.coords);
      setIsInitialRegionAnimated(true);
    }
  }, [location]);

  if (location === null) return <Text>Loading</Text>;

  return (
    <RelativeBlock>
```

```
<ScreenWapper>
  {!isNavigationMode && (
    <FindAddressOnMap
      setActivePoint={setActivePoint}
      animateToRegion={(coordinates) => {
        animateToRegion(mapRef, {
          latitude: coordinates.lat,
          longitude: coordinates.lng,
        });
      }}
    />
  )}
  <MapView
    initialRegion={location.coords}
    maxZoomLevel={15}
    ref={mapRef}
    showsCompass={true}
    style={{ flex: 1, width: "100%" }}
    provider={cardProvider === "Google" ? PROVIDER_GOOGLE : undefined}
    customMapStyle={isDarkMode && darkMapStyles}
    showsTraffic={trafic}
    onRegionChange={(region) => {
      region.latitudeDelta &&
        setLocationRadius(region.latitudeDelta * 1000);
    }}
  >
    {activePoint && (
      <Marker
        flat
        coordinate={{
          latitude: activePoint.coordinates.lat,
          longitude: activePoint.coordinates.lng,
        }}
      />
    )}

    {finalLocation && isNavigationMode && (
      <Marker flat coordinate={finalLocation} />
    )}

    <Circle
      key={"fd"}
      center={{
        latitude: location?.coords?.latitude,
        longitude: location?.coords?.longitude,
      }}
      radius={locationRadius}
      strokeWidth={3}
      strokeColor={"#fff"}
      fillColor={"#1a66ff"}
      style={{ zIndex: 9999 }}
    />

    {finalLocation && isNavigationMode && (
      <MapViewDirections
        origin={location.coords}
        destination={finalLocation}
        apikey={GOOGLE_MAPS_APIKEY}
        strokeColor="hotpink"
```

```
                strokeWidth={5}
              />
            )}
          </MapView>

          <AddressModal
            activePoint={activePoint}
            animateToLocationCallback={() => {
              if (location) animateToRegion(mapRef, location.coords);
            }}
            setActivePoint={setActivePoint}
            setIsNavigationMode={setIsNavigationMode}
            setSavedPlaces={setSavedPlaces}
            savedPlaces={savedPlaces}
            setFinalLocation={setFinalLocation}
          />

          <ButtonsOnMap
            isNavigationMode={isNavigationMode}
            setIsNavigationMode={setIsNavigationMode}
            animateToLocationCallback={() => {
              if (location) animateToRegion(mapRef, location.coords);
            }}
          />
        </ScreenWapper>
      </RelativeBlock>
    );
};
```

ДОДАТОК В

Логіка зберігання даних у пам’яті телефону

```
import { useEffect, useState } from "react";
import { useColorScheme, NativeModules } from "react-native";
import { getStorageData, setStorageData } from "../functions";
import { LOCALES } from "../i18n/locales";

const store = {
  isDarkMode: true,
  isAutoDarkMode: false,
};

export const useDarkMode = () => {
  const [isDarkMode, setIsDarkMode] = useState(true);
  const [isAutoDarkMode, setIsAutoDarkMode] = useState(false);
  const colorShemeOfDevice = useColorScheme();

  useEffect(() => {
    getStorageData("@themeSetting")
      .then((data) => {
        if (data === null) {
          setStorageData("@themeSetting", store);
        } else {
          setIsAutoDarkMode(data.isAutoDarkMode);
          setIsDarkMode(data.isDarkMode);
        }
      })
      .catch((data) => {
        console.log("failed", data);
      });
  }, []);

  useEffect(() => {
    store.isDarkMode = isDarkMode;

    setStorageData("@themeSetting", store);
  }, [isDarkMode]);

  useEffect(() => {
    store.isAutoDarkMode = isAutoDarkMode;

    if (isAutoDarkMode) {
      store.isDarkMode = colorShemeOfDevice === "dark";
      setIsDarkMode(colorShemeOfDevice === "dark");
    }

    setStorageData("@themeSetting", store);
  }, [isAutoDarkMode, colorShemeOfDevice]);

  return { isDarkMode, setIsDarkMode, isAutoDarkMode, setIsAutoDarkMode };
};

export const useLocale = () => {
  const [locale, setLocale] = useState(LOCALES.ENGLISH);

  useEffect(() => {
    getStorageData("@locale")
```

```
      .then((data) => {
        if (data === null) {
          setStorageData("@locale", { locale: LOCALES.ENGLISH });
          setLocale(LOCALES.ENGLISH);
        } else {
          setLocale(data.locale);
        }
      })
      .catch((data) => {
        console.log("failed", data);
      });
  }, []);

  useEffect(() => {
    setStorageData("@locale", { locale });
    setLocale(locale);
  }, [locale]);

  return { locale, setLocale };
};

export const useCardProvider = () => {
  const [cardProvider, setCardProvider] = useState("Google");

  useEffect(() => {
    getStorageData("@cardProvider")
      .then((data) => {
        if (data === null) {
          setStorageData("@cardProvider", { cardProvider: "Google" });
          setCardProvider("Google");
        } else {
          setCardProvider(data.cardProvider);
        }
      })
      .catch((data) => {
        console.log("failed", data);
      });
  }, []);

  useEffect(() => {
    setStorageData("@cardProvider", { cardProvider });
    setCardProvider(cardProvider);
  }, [cardProvider]);

  return { cardProvider, setCardProvider };
};

export const useTrafic = () => {
  const [trafic, setTrafic] = useState(false);

  useEffect(() => {
    getStorageData("@trafic")
      .then((data) => {
        if (data === null) {
          setStorageData("@trafic", false);
          setTrafic(false);
        } else {
          setTrafic(data);
        }
```

```
        })
        .catch((data) => {
          console.log("failed", data);
        });
    }, []);

    useEffect(() => {
      setStorageData("@trafic", trafic);
    }, [trafic]);

    return { trafic, setTrafic };
};
```

ДОДАТОК Г

Файл налаштувань

```
import { SafeAreaView, Switch, View } from "react-native";
import RNPickerSelect from "react-native-picker-select";
import { FormattedMessage, useIntl } from "react-intl";

import {
  ScreenWapper,
  PageTitle,
  LabelTextStyle,
  SettingItem,
  pickerSelectStyles,
} from "../components";
import { LOCALES } from "../i18n/locales";

export const Settings = ({
  isDarkMode,
  setIsDarkMode,
  isAutoDarkMode,
  setIsAutoDarkMode,
  locale,
  setLocale,
  trafic,
  setTrafic,
  cardProvider,
  setCardProvider,
}) => {
  return (
    <ScreenWapper>
      <SafeAreaView>
        <PageTitle>
          <FormattedMessage id="settings" />
        </PageTitle>

        <SettingItem>
          <LabelTextStyle>
            <FormattedMessage id="darkTheme" />
          </LabelTextStyle>
          <Switch
            trackColor={{ false: "#767577", true: "#4DD863" }}
            thumbColor={isDarkMode ? "#fff" : "#f4f3f4"}
            ios_backgroundColor="#3e3e3e"
            disabled={isAutoDarkMode}
            onValueChange={() => {
              setIsDarkMode((state) => !state);
            }}
            value={isDarkMode}
          />
        </SettingItem>

        <SettingItem>
          <LabelTextStyle>
            <FormattedMessage id="autoDarkTheme" />
          </LabelTextStyle>
          <Switch
            trackColor={{ false: "#767577", true: "#4DD863" }}
            thumbColor={isDarkMode ? "#fff" : "#f4f3f4"}
```

```
      ios_backgroundColor="#3e3e3e"
      onValueChange={() => {
        setIsAutoDarkMode((state) => !state);
      }}
      value={isAutoDarkMode}
    />
  </SettingItem>

  <SettingItem>
    <LabelTextStyle>
      <FormattedMessage id="trafic" />
    </LabelTextStyle>
    <Switch
      trackColor={{ false: "#767577", true: "#4DD863" }}
      thumbColor={isDarkMode ? "#fff" : "#f4f3f4"}
      ios_backgroundColor="#3e3e3e"
      onValueChange={() => {
        setTrafic((state) => !state);
      }}
      value={trafic}
    />
  </SettingItem>

  <SettingItem>
    <LabelTextStyle>
      <FormattedMessage id="language" />
    </LabelTextStyle>

    <RNPickerSelect
      placeholder={{}}
      onValueChange={(value) => setLocale(value)}
      darkTheme={isDarkMode}
      style={pickerSelectStyles}
      value={locale}
      items={[
        {
          label: "Українська",
          value: LOCALES.UKRAINIAN,
        },
        { label: "English", value: LOCALES.ENGLISH },
      ]}
    />
  </SettingItem>

  <SettingItem>
    <LabelTextStyle>Choose card provider</LabelTextStyle>

    <RNPickerSelect
      placeholder={{}}
      onValueChange={(value) => setCardProvider(value)}
      darkTheme={isDarkMode}
      style={pickerSelectStyles}
      value={cardProvider}
      items={[
        {
          label: "Google",
          value: "Google",
        },
        { label: "Apple", value: "Apple" },
```

```
            ]}
          />
        </SettingItem>
      </SafeAreaView>
    </ScreenWapper>
  );
};
```

ДОДАТОК Ґ

Код пошуку адреси

```
import React, { useState } from "react";
import Styled from "styled-components/native";
import { ScrollView, View, TouchableOpacity, Keyboard } from "react-native";
import { ActivityIndicator } from "@react-native-material/core";
import { Icon } from "@react-native-material/core";

import { TextInputStyled, BodyText, MarginBlock } from "../components";
import { MapAPI } from "../api/MapAPI";
import { colors } from "../constants";

const Wrapper = Styled.View`
    position: absolute;
    z-index: 9999;
    top: 40px;
    width: 80%;
    margin-top: 2%
    margin-left: 3%;
 `;

const SearchListWrapper = Styled.View`
  background: ${(props) => props.theme.background}
  height: 100%;
  margin-top: 100px;
  padding-bottom: 100px;
`;

const StyledResultItem = Styled.View`
  padding: 15px;
  flex-direction: row;
`;
const SearchTextWrapper = Styled.View`
  flex-direction: row;
  flex-wrap: wrap;
  padding: 0px 10px 5px 10px;
`;

export const ResultItem = ({
  formatted,
  coordinates,
  setActivePoint,
  setFoundAddresses,
  setAddress,
  animateToRegion,
}) => {
  return (
    <StyledResultItem>
      <View>
        <Icon name="map-search" color="gray" size={30} />
      </View>
      <TouchableOpacity
        onPress={() => {
          setActivePoint({ coordinates, formatted });
          animateToRegion(coordinates);
          setFoundAddresses([]);
          setAddress("");
```

```
            Keyboard.dismiss();
          }}
        >
          <SearchTextWrapper>
            {formatted && <BodyText>{formatted} </BodyText>}
          </SearchTextWrapper>
        </TouchableOpacity>
      </StyledResultItem>
    );
};

export const FindAddressOnMap = ({ setActivePoint, animateToRegion }) => {
  const [address, setAddress] = useState("");
  const [foundAddresses, setFoundAddresses] = useState([]);
  const [isPending, setIsPending] = useState(null);

  return (
    <>
      <Wrapper>
        <TextInputStyled
          placeholder="Пошук адреси "
          icon="map-search"
          value={address}
          onChangeText={(text) => {
            setAddress(text);
          }}
          resetCallback={() => setFoundAddresses([])}
          onSubmitEditing={({ nativeEvent: { text } }) => {
            if (text === "") setFoundAddresses([]);
            text &&
              MapAPI.searchByAddress(text, setFoundAddresses, setIsPending);
          }}
        />
      </Wrapper>
      {address && (
        <SearchListWrapper>
          {foundAddresses.length === 0 && (
            <MarginBlock marginTop={"10px"}>
              <BodyText> Empty list</BodyText>
            </MarginBlock>
          )}
          {!isPending && (
            <ScrollView>
              {foundAddresses.map((address, index) => {
                return (
                  <ResultItem
                    {...address}
                    key={index}
                    setActivePoint={setActivePoint}
                    setFoundAddresses={setFoundAddresses}
                    setAddress={setAddress}
                    animateToRegion={animateToRegion}
                  />
                );
              })}
            </ScrollView>
          )}
          {isPending === true && (
            <MarginBlock marginTop={"10px"}>
```

```
            <ActivityIndicator color={colors.dark.primary} size={"large"} />
          </MarginBlock>
        )}
      </SearchListWrapper>
    )}
  </>
 );
};
```

Логіка запиту на сервер за даними нижче.

```
import axios from "axios";

const instance = axios.create({
  timeout: 1000,
});

export const MapAPI = {
  searchByAddress(address, setFoundAddresses, setIsPending) {
    setIsPending(true);
    const formattedAddres = encodeURI(address);

    instance
      .get(

`https://api.opencagedata.com/geocode/v1/json?q=${address}&key=902190453c7d49b69
eaf56109c52ec`
      )
      .then((res) => {
        setFoundAddresses(
          [...res.data.results].map((item) => ({
            formatted: item.formatted,
            coordinates: item.geometry,
          }))
        );

        setIsPending(null);
      })
      .catch((res) => {
        setIsPending(null);
      });
  },
};
```

ДОДАТОК Д

Код сторінки інформації

```
import { SafeAreaView, Linking, Button } from "react-native";
import { FormattedMessage } from "react-intl";
import { Icon } from "@react-native-material/core";
import {
  ScreenWapper,
  PageTitle,
  BodyText,
  Block,
  BoldText,
  MarginBlock,
  SpaceBetween,
} from "../components";

export const About = () => {
  return (
    <ScreenWapper>
      <SafeAreaView>
        <PageTitle>
          <FormattedMessage id="info" />
        </PageTitle>

        <Block>
          <MarginBlock marginBottom="40px">
            <MarginBlock>
              <Icon name="information" color="#2B5278" size={40} />
            </MarginBlock>
            <BoldText>
              <FormattedMessage id="aboutApplicationLabel" />
            </BoldText>
            <BodyText>
              <FormattedMessage id="aboutApplicationText" />
            </BodyText>
          </MarginBlock>

          <MarginBlock marginBottom="20px">
            <BoldText>
              <FormattedMessage id="connectWithMeLabel" />
            </BoldText>
          </MarginBlock>

          <MarginBlock marginBottom="20px">
            <SpaceBetween>
              <Icon name="email" color="#2B5278" size={35} />
              <Button
                onPress={() => Linking.openURL("mailto:nagoluk1626@gmail.com")}
                title="nagoluk1626@gmail.com"
              />
            </SpaceBetween>
          </MarginBlock>
          <MarginBlock marginBottom="20px">
            <SpaceBetween>
              <Icon name="email" color="#2B5278" size={35} />
              <Button
                onPress={() =>
```

```
                    Linking.openURL("mailto:naholiuk.d@donnu.edu.ua")
                  }
                  title="naholiuk.d@donnu.edu.ua"
                />
              </SpaceBetween>
            </MarginBlock>

            <SpaceBetween>
              <Icon name="send" color="#2B5278" size={35} />
              <Button
                onPress={() => Linking.openURL("https://t.me/dmytro_naholiuk")}
                title="Telegram"
              />
            </SpaceBetween>
          </Block>
        </SafeAreaView>
      </ScreenWapper>
    );
};
```

Додаток 2 до наказу
від «31» березня 2023 року
№119/05

ДЕКЛАРАЦІЯ

про дотримання академічної доброчесності

Я, _____

_____

Повністю вказується ПІБ та статус (посада для працівників, освітня (освітньо-наукова) програма – для здобувачів вищої освіти)

що нижче підписалась/підписався, розуміючи та підтримуючи загальновизнані засади справедливості, доброчесності та законності,

ЗОБОВ'ЯЗУЮСЬ:

дотримуватися принципів та правил академічної доброчесності, що визначені законодавством України, локальними нормативними актами Донецького національного університету імені Василя Стуса, положеннями, правилами, умовами, визначеними іншими суб'єктами, та не допускати їх порушення.

ПІДТВЕРДЖУЮ:

що мені відомі положення статті 42 Закону України «Про освіту»;

що у даній роботі не представляла/представляв чиїсь роботи повністю або частково як свої власні. Там, де я скористалася/скористався працею інших, я зробила/зробив відповідні посилання на джерела інформації;

що дана робота не передавалась іншим особам і подається вперше, не порушує авторських та суміжних прав закріплених статтями 21-25 Закону України «Про авторське право та суміжні права», а дані та інформація не отримувались в недозволений спосіб.

УСВІДОМЛЮЮ:

що ця робота може бути перевірена університетом на плагіат або інші порушення академічної доброчесності, в тому числі з використанням спеціалізованих сервісів;

що у разі порушення академічної доброчесності, до мене можуть бути застосовані процедури, передбачені законодавством України та Кодексом академічної доброчесності та корпоративної етики Донецького національного університету імені Василя Стуса, іншими локальними нормативними актами університету, та я можу бути притягнута/притягнутий до академічної відповідальності.

_____ _____

(дата) (підпис)