МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Львівський національний університет імені Івана Франка
Факультет електроніки та комп'ютерних технологій
Кафедра системного проектування

Допустити до захисту

Завідувач кафедри
_____ доцент Шувар Р. Я.
«____»_____ 2023 р.

Курсова робота

Магістр
(освітній ступінь)

Додаток на ОС Android для відстеження шляху користувача

Виконав:
Студент І курсу групи ФеІм-12
спеціальності 122 –
Комп'ютерні науки

_____ Колибєлкін О.І.

Науковий керівник:
_____ Юзевич В.М.
«____»_____ 2023 р.

Львів — 2023

АНОТАЦІЯ

Під час виконання даної роботи була проведена оцінка актуальності додатку, а також аналіз конкурентів на ринку та сформульовано чітке бачення того, яким має бути додаток для відстеження шляху користувача. Цей додаток розроблено з метою відстежування активності користувача, зокрема відстеження параметрів його руху та фізичної активності. Він надає можливість користувачам відстежувати, зокрема, пройдену відстань, час забігу та кількість кроків, які вони зробили.

Завдяки успішному виконанню, у програмі додатку продемонстровано надійність та стабільність у роботі. Всі основні функціональні підпрограми забезпечують можливості праці додатка без збоїв та відповідають очікуванням користувачів. Коректна робота програми підтверджує її готовність для використання у реальних умовах.

ABSTRACT

During the execution of this work, an assessment of the relevance of the application was carried out, as well as an analysis of competitors on the market, and a clear vision of what the application for tracking the user's path should be was formulated. This application is designed to track the user's activity, in particular to track parameters of his movement and physical activity. It allows users to track, in particular, the distance traveled, the time of the run and the number of steps they have taken.

Thanks to its successful execution, the application program has demonstrated reliability and stability in operation. All main functional subroutines ensure that the application works without failures and meet the expectations of users. Correct operation of the program confirms its readiness for use in real conditions.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ **4**
1.1 Актуальність задачі 4
1.2 Аналіз наявних додатків та конкурентів 5
1.3.Постановка задачі 8
2. Інструменти **9**
2.1 Android Studio 9
2.2 Google Maps API 12
2.3 Kotlin 14
3. Створення додатку **17**
3.1 Додавання до проекту свого API 17
3.2 Розташування 22
3.3 Відображення руху користувача 28
3.4 Представлення даних про місцезнаходження 31
3.5 Розпізнавання активності користувача 36
3.6 Запит на інші дозволи 39
3.7 Користувацький інтерфейс 41
Висновки **49**
ПЕРЕЛІК ПОСИЛАНЬ **51**

Вступ

Мета цієї курсової роботи полягає в розробці додатку відстеження бігу для платформи Android, який надасть користувачам можливість відстежувати свою активність, відображаючи їхню позицію на мапі разом із маршрутом. Даний додаток буде корисним для людей , які займаються бігом, або іншими видами активного відпочинку.

У ході розробки додатку я буду використовувати API Google Maps, яке надає широкі можливості для відображення картографічних даних та взаємодії з ними. Я також буду використовувати функціонал Android для отримання поточного місцезнаходження користувача та отримання оновлень про його рухи.

Цей проект надасть нам можливість практично використати отримані знання з програмування на платформі Android та роботи з API Google Maps. Будучи унікальним та корисним додатком, він може стати відмінним зразком для подальшого удосконалення його функціоналу.

У даній курсовій роботі я покроково розгляну всі необхідні етапи розробки додатку відстеження бігу, починаючи з налаштування проекту, роботи з API Google Maps та виконання основних функцій. Після завершення роботи ми отримаємо повнофункціональний додаток, який зможе задовольнити потреби користувачів, які бажають відстежувати свою активність та спортивні досягнення.

Під час роботи над цим проектом я набув важливих навичок програмування для Android, використання сторонніх сервісів, роботу з місцезнаходженням та відображенням даних на мапі. Та отримав практичний досвід розробки додатків, що відповідають сучасним вимогам та потребам користувачів.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Актуальність задачі

В сучасному світі здоровий спосіб життя та фізична активність набувають все більшої популярності. З ростом свідомості про важливість збереження здоров'я та піклування про фізичну форму, з'являється попит на додатки, які допомагають користувачам відстежувати свої тренування та досягнення. У цьому контексті додаток для відстеження руху користувача є дуже актуальним і має значний потенціал.

Одним із головних факторів, що робить такий додаток актуальним, є розповсюдження смартфонів та їх загальна доступність. Багато людей володіють смартфонами, які мають вбудовані сенсори та можливості вимірювання, такі як GPS та акселерометр. Це дає змогу використовувати ці функції для відстеження руху та збереження даних про тренування без потреби в додаткових пристроях або носимих пристроях.

Додаток для відстеження руху також актуальний у контексті здоров'я та фітнесу. Багато людей мають бажання тренуватись, покращувати свою фізичну форму або досягати специфічних цілей, таких як зниження ваги чи підготовка до змагань. Відстеження руху, дистанції, швидкості та інших метрик може стати цінним інструментом для цих людей, допомагаючи їм відслідковувати свій прогрес, встановлювати нові цілі та залишатись мотивованими.

Крім того, такий додаток може мати практичне застосування для спортивних тренувань. Багато спортсменів, які займаються бігом, велоспортом або іншими активними видами спорту, цікавляться відстеженням своєї траси, аналізом свого руху та вдосконаленням своїх результатів. Додаток може надати їм зручний інструмент для цих цілей, допомагаючи відстежувати маршрути, аналізувати дані тренувань та вдосконалювати свої навички.

Також варто зазначити, що зростання популярності соціальних мереж і спільнот для спортсменів створює попит на додатки, які дозволяють користувачам поділитися своїми досягненнями та тренуваннями з іншими. Додаток для відстеження руху може поєднати ці дві потреби, надаючи можливість публікувати результати тренувань у соціальних мережах та ділитися ними зі спортивною спільнотою.

Загалом, додаток для відстеження руху користувача є актуальним і корисним інструментом для людей, які цікавляться фізичною активністю, здоров'ям та спортом. Він відповідає потребам сучасного суспільства, що все більше звертає увагу на здоровий спосіб життя та саморозвиток. Завдяки своїм функціям та можливостям, такий додаток може стати незамінним помічником у досягненні тренувальних цілей та підтримки мотивації користувачів.

1.2 Аналіз наявних додатків та конкурентів

Додаток для відстеження руху користувача можна порівняти з такими відомими додатками, як "Strava", "Nike Run Club" і "Runkeeper". Ось короткий огляд цих додатків:

1. Strava: «Strava є одним з найпопулярніших додатків для відстеження тренувань і спортивної активності. Він пропонує відстеження руху, включаючи біг, їзду на велосипеді, плавання та інші види активності. Strava має широкі можливості для аналізу тренувань, спільноти користувачів, підписки на виклики та багато іншого.» [1]

Рис. 1.2.1 - Лого додатку “Strava”

2. Nike Run Club: «Цей додаток, розроблений компанією Nike, спеціалізується на бігу. Він пропонує функції відстеження руху, аудіосповіщення про пройдену відстань та швидкість, налаштовані тренувальні плани та багато інших функцій, спрямованих на покращення бігових тренувань.»[2]

Рис. 1.2.2 - Вигляд додатку “Nike Run Club”

3. Runkeeper:« Runkeeper також є добре відомим додатком для відстеження руху та бігу. Він пропонує функції GPS-відстеження, статистику тренувань, метрики швидкості, відстані та часу, спільноту користувачів та інші функції, які допомагають зміцнити мотивацію та досягнення цілей.»[3]

Рис. 1.2.2 - Вигляд додатку “Runkeeper”

Ці додатки вже встановили собі сильну позицію на ринку та мають широкий спектр функціональності для відстеження руху. Проте, наш додаток може відрізнятися в ряді аспектів, наприклад, унікальним дизайном, інтуїтивним інтерфейсом, спеціалізованими функціями або більш гнучкими налаштуваннями. Важливо зосередитися на унікальних особливостях нашого додатку та перевагах, які він надає користувачам у порівнянні з конкурентами, щоб викликати інтерес у потенційних користувачів і забезпечити його успішність на ринку.

1.3.Постановка задачі

Постановка задачі для додатку "Відстеження руху користувача":

Наша мета - розробити додаток для відстеження руху користувача на платформі Android, який надає зручний і простий спосіб відстежування пройденої відстані, часу тренування та інших параметрів, що допомагають користувачам покращити свої фітнес-цілі та мотивувати себе.

Основні вимоги до додатку:

1. Відстеження руху: Додаток повинен відстежувати маршрут користувача за допомогою GPS-технологій, зберігати дані про відстань, швидкість та час тренування.

2. Візуалізація на мапі: Додаток повинен відображати пройдений маршрут користувача на мапі, щоб користувачі могли візуально спостерігати за своїми переміщеннями.

3. Статистика тренування: Додаток повинен розраховувати загальну відстань, середню швидкість, час тренування та інші параметри, що дозволяють користувачам аналізувати свої досягнення.

4. Інтерфейс користувача: Додаток повинен мати зрозумілий та зручний інтерфейс, що дозволяєлегко навігуватися по функціях додатку і забезпечує позитивний користувацький досвід.

Наша постановка задачі полягає в розробці додатку, що задовольняє ці вимоги та надає користувачам зручні інструменти для відстеження їхнього руху та досягнення фітнес-цілей. Ми плануємо реалізувати цей додаток, використовуючи Android Studio, Google Maps API та мову програмування Kotlin, та протестувати його на пристрої з ОС Android 11.

2. Інструменти

2.1 Android Studio

«Android — це повний набір програмного забезпечення для мобільних пристроїв, таких як планшетні комп’ютери, ноутбуки, смартфони, пристрої для читання електронних книг, приставки тощо.»[4]

Android Studio — це інтегроване середовище розробки (IDE), розроблене компанією Google, яке надає розробникам засоби для створення додатків для платформи Android OS. Android Studio можна встановити на операційні системи Windows, Mac і Linux. Для отримання облікового запису розробника додатків в Google Play App Store потрібно заплатити 25 доларів. Android Studio базується на IntelliJ IDEA.

Це безкоштовне середовище розробки, яке надає макети для створення користувацького інтерфейсу (UI), що зазвичай є початковим кроком у розробці додатків. В Android Studio є інструменти для розробки рішень для смартфонів, планшетів, а також для платформ Android TV, Android Wear, Android Auto, Glass і додаткових контекстних модулів.

Середовище Android Studio призначене як для невеликих команд розробників мобільних додатків (навіть з однією людиною), так і для великих міжнародних організацій з використанням систем управління версіями, таких як GIT. Досвідчені розробники можуть вибрати інструменти, які найкраще підходять для масштабних проектів. Для розробки Android-додатків використовуються мови програмування Java або C++. Основою робочого процесу Android Studio є концепція неперервної інтеграції, що дозволяє виявляти проблеми негайно. Постійна перевірка коду забезпечує ефективний зворотний зв'язок з розробниками і дозволяє швидко опублікувати версію мобільного додатка в Google Play App Store. Для цього також є підтримка інструментів LINT, ProGuard і App Signing.

За допомогою інструментів оцінки продуктивності можна визначити стан файлу програмного пакету. Візуалізація графіки дозволяє перевірити, чи відповідає додаток орієнтиру Google у 16 мілісекунд. Інструмент для

візуалізації пам'яті допомагає визначити, коли додаток використовує надто багато оперативної пам'яті і коли відбувається "збір сміття". Інструменти для аналізу батареї показують навантаження на пристрій.

Android Studio сумісна з платформою Google App Engine для швидкої інтеграції нових API та функцій в хмарі. У середовищі розробки доступні різні API, такі як Google Play, Android Pay і Health. Підтримка охоплює всі версії Android, починаючи з Android 1.6. Є варіанти Android, які суттєво відрізняються від версії Google Android, найпопулярнішим з яких є Amazon Fire OS. В Android Studio можна створювати APK-файли для цієї ОС. Підтримка Android Studio надається через онлайн-форуми.

Рис. 2.1.1 - Лого додатку “Android Studio”

Основні функції Android Studio:

- Розширений редактор макетів з WYSIWYG-інтерфейсом, здатність працювати з компонентами користувацького інтерфейсу за допомогою перетягування, попередній перегляд макету на різних конфігураціях екрану.

- Збірка додатків на основі Gradle.

- Різні типи збірок та генерація кількох файлів .apk.

- Рефакторинг коду.

- Статичний аналізатор коду (Lint), який допомагає виявляти проблеми продуктивності, несумісність версій та інші проблеми.

- Вбудований ProGuard та утиліта для підпису додатків.

- Шаблони основних макетів та компонентів Android.

- Підтримка розробки додатків для Android Wear і Android TV.

- Вбудована підтримка Google Cloud Platform, яка включає інтеграцію з сервісами Google Cloud Messaging та App Engine.

- Android Studio 2.1 підтримує Android Preview SDK, що дозволяє розробникам почати роботу над додатком для нової програмної платформи.

- В новій версії Android Studio 2.1 покращена підтримка компілятора Jack, а також покращена підтримка Java 8 та функція Instant Run.

- Починаючи з Platform-tools 23.1.0, для Linux доступна лише 64-бітна версія.

- В Android Studio 3.0 будуть стандартно включені інструменти мови Kotlin, які базуються на IDE від JetBrains.

2.2 Google Maps API

Google Maps API - це набір інтерфейсів програмування додатків, що дозволяє клієнту взаємодіяти з інтегрованими сервісами. Це дає змогу створювати прості додатки для більш складних програмних рішень на основі розташування для інтернету iOS и Android.

Спочатку API було створено для розміщення карти на сайті компанії, де бізнесмен може позначити своє місцезнаходження, щоб клієнти могли швидко знайти офіс. Сьогодні функціонал додатка значно розширився.

Рис. 2.2.1 - Лого додатку “Google Maps”

Основні можливості Google Maps API:

Більшість інструментів працює на основі JavaScript. До них належать:

- Відображення Google Maps у браузері, на пристроях iOS або Android, а також можливість вказати конкретні географічні координати (широту та довготу).

- Показ InfoWindow, яке є спливаючим вікном для відображення інформації про місцезнаходження над маркером при натисканні.

- Геокодування, або процес перетворення поштової адреси на географічні координати (широту та довготу), а також зворотне геокодування.

- Вибір найближчого відповідного місця на основі пошукового запиту користувача (вказується місцезнаходження, його тип і радіус пошуку).

- Автозаповнення відображає пропоновані назви вулиць у розкривному списку під полем вводу по мірі того, як користувач вводить дані, що дуже зручно, коли сервіс визначення місцезнаходження не підтримується браузером або пристроєм.

- Матриця відстаней розраховує відстань і час у дорозі між двома або більше адресами на основі способів переміщення, таких як водіння, ходьба, їзда на велосипеді та громадський транспорт.

2.3 Kotlin

«Kotlin - це статично типізована, об'єктно-орієнтована мова програмування, яка взаємодіє з віртуальною машиною Java (JVM), бібліотеками класів Java і Android. »[5]

Основна мета мови Kotlin - запропонувати більш компактну, продуктивну і безпечну альтернативу мові Java, придатну для використання всюди, де сьогодні використовується Java. Java - надзвичайно популярна мова, яка використовується в різних середовищах, починаючи від смарт-карт (технологія Java Card) до найбільших обчислювальних центрів таких компаній, як Google, Twitter і LinkedIn. У більшості таких середовищ застосування Kotlin може

допомогти розробникам досягати своїх цілей за меншим обсягом коду і уникати багатьох неприємностей.

Рис. 2.1.1 - Лого мови “Kotlin”

Для чого використовується Kotlin?

Kotlin є загальнопризначеною мовою розробки, яка використовується головним чином для розробки мобільних додатків Android. Крім додатків для Android, Kotlin також корисний для таких завдань:

- Розробка на стороні сервера. Традиційно для розробки веб-додатків на стороні сервера використовується Java. Kotlin може використовуватись разом з Java для розробки на сервері. Kotlin підтримує бібліотеки класів Java.
- Розробка повного стеку веб-додатків. Розробники використовують Kotlin для JavaScript, щоб перетворювати рядки коду Kotlin на JavaScript для розробки клієнтської частини веб-додатків. Цей підхід дозволяє використовувати один і той же код як на фронтенді, так і на бекенді.

- Розробка мобільних додатків для різних платформ. Розробники використовують Kotlin для розробки додатків для Android та інших мобільних платформ, включаючи Apple iOS, Apple watchOS і Linux.
- Наука про дані. Kotlin часто використовується для роботи з науковими завданнями з обробки даних, такими як побудова потоків даних та впровадження моделей машинного навчання. Jupyter та Zeppelin, два популярні інструменти для науки про дані, які використовуються з Java, також підтримують Kotlin.

Організації, які створюють додатки для Android за допомогою Kotlin, також можуть використовувати ці навички для керування ресурсами у хмарі.

Опис мов розробки для мобільних додатків

Kotlin - одна з кількох мов розробки для мобільних додатків.

Переваги Kotlin:

- Сумісність. Kotlin взаємодіє з Java, оскільки обидві мови компілюються в один і той же байт-код. Kotlin може бути скомпільований в JavaScript або код LLVM-кодера, що дозволяє програмістам виконувати компіляцію у режимі реального часу для забезпечення безперервної роботи коду, вбудованого в іншу програму. Крім того, Kotlin має спільний набір інструментів з Java, що полегшує міграцію Java-додатків на Kotlin.
- Безпека. Kotlin розроблений з метою уникнення поширених помилок в коді, які можуть призводити до його розриву або створення вразливостей. Мова підтримує безпеку щодо значень null і усуває помилки вказівника null.
- Ясність. Kotlin усуває деяку зайвість у базовому синтаксисі популярних мов, таких як Java. Kotlin є засобом економії часу для

розробників, оскільки надає більш компактний код. Розробники можуть писати програми з меншою кількістю заповнювачів, що збільшує їх продуктивність.

- Підтримка інструментарію. Kotlin має підтримку інструментарію від Android, зокрема інструментів, оптимізованих для розробки Android, таких як Android Studio, Android KTX і Android SDK.
- Підтримка спільноти. Незважаючи на те, що Kotlin є відносно новою мовою порівняно з Java, вона має спільноту розробників, які працюють над вдосконаленням мови і надають документацію.

3. Створення додатку

3.1 Додавання до проекту свого API

В проекті ми будемо:

- Використовувати API Карт Google для відображення мапи та роботи з геолокацією.
- Отримувати останню записану позицію користувача.
- Отримувати оновлення про рухи користувача.
- Розпізнавати тип руху користувача.
- Наносити маршрут користувача на карту.

Завантажуємо початковий проект, натиснувши кнопку «Завантажити матеріали» у верхній або нижній частині посібника. Коли проект завантажився у свою IDE, перейдимо до Google Cloud Console і входимо у свій обліковий запис Google. На сторінці облікових даних вибираємо проект або створюємо новий.

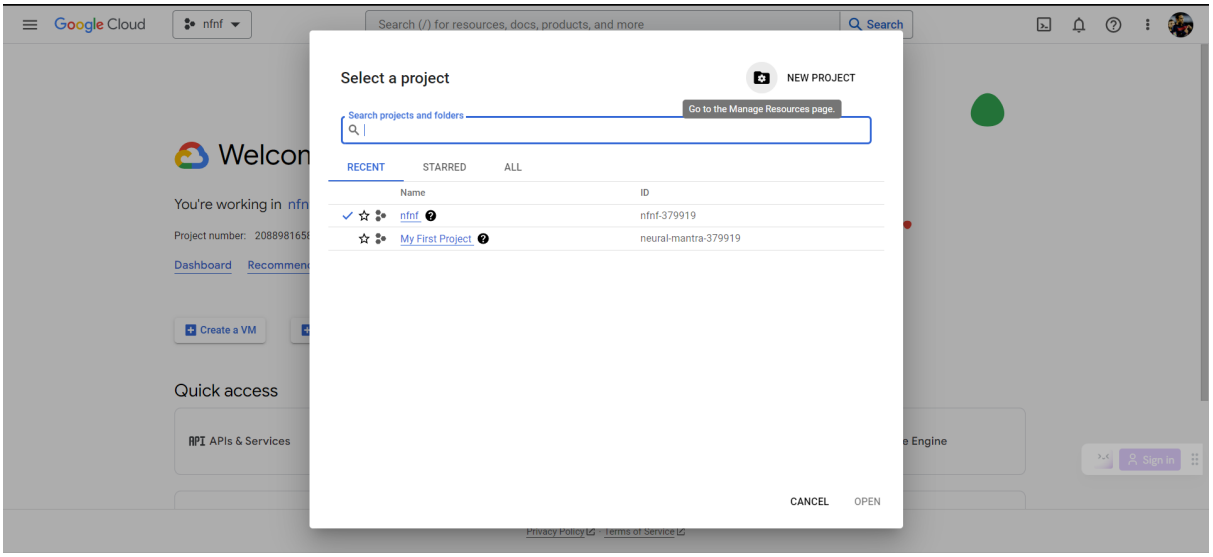

Рис. 3.1.1 - Google Cloud Console

У верхній частині сторінки натискаємо «Create Credentials» та вибираємо «API Key» . Копіюємо свій ключ API, оскільки він нам знадобиться в подальшому.

Також вмикаємо Maps SDK для Android у бібліотеці API

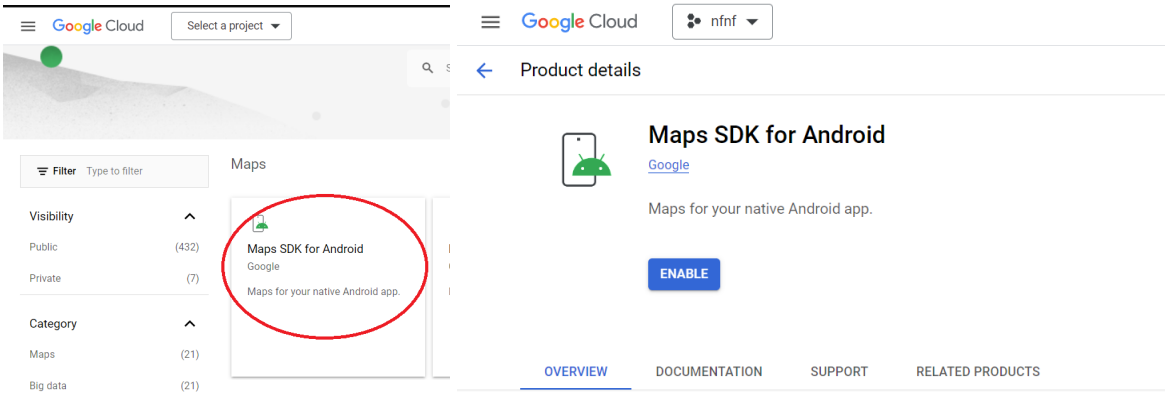

Рис. 3.1.2 - Maps SDK для Android

Тепер в IDE вибираємо два файли google_maps_api.

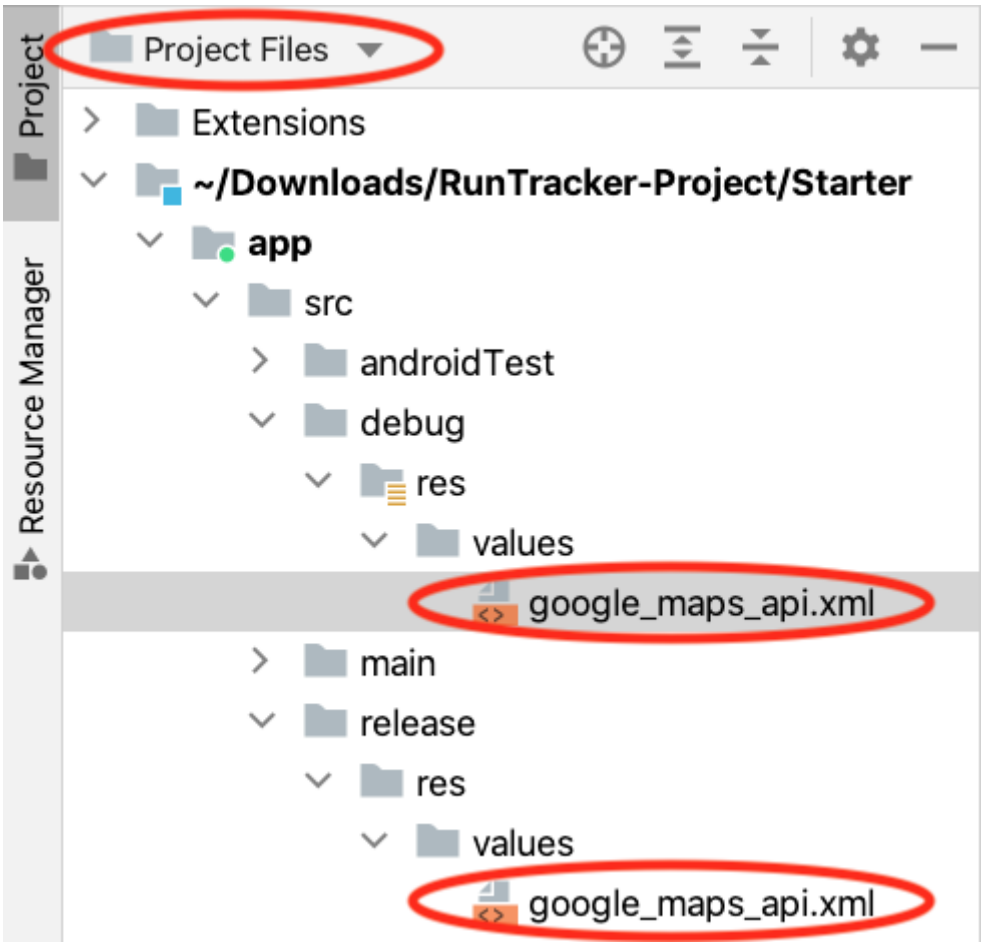

Рис. 3.1.3 - Загальна структура проекту

Перший — для варіанта налагодження, а інший — для випуску. Тут ми вставляємо свій API ключ для подальшої праці:

```
< resources >
  < string name = "google_maps_key"
templateMergeStrategy = "preserve" translatable =
"false" > KEY_HERE </ string >
</ resources >
```

Натиснувши кнопку запуску зможемо побачити такий інтерфейс:

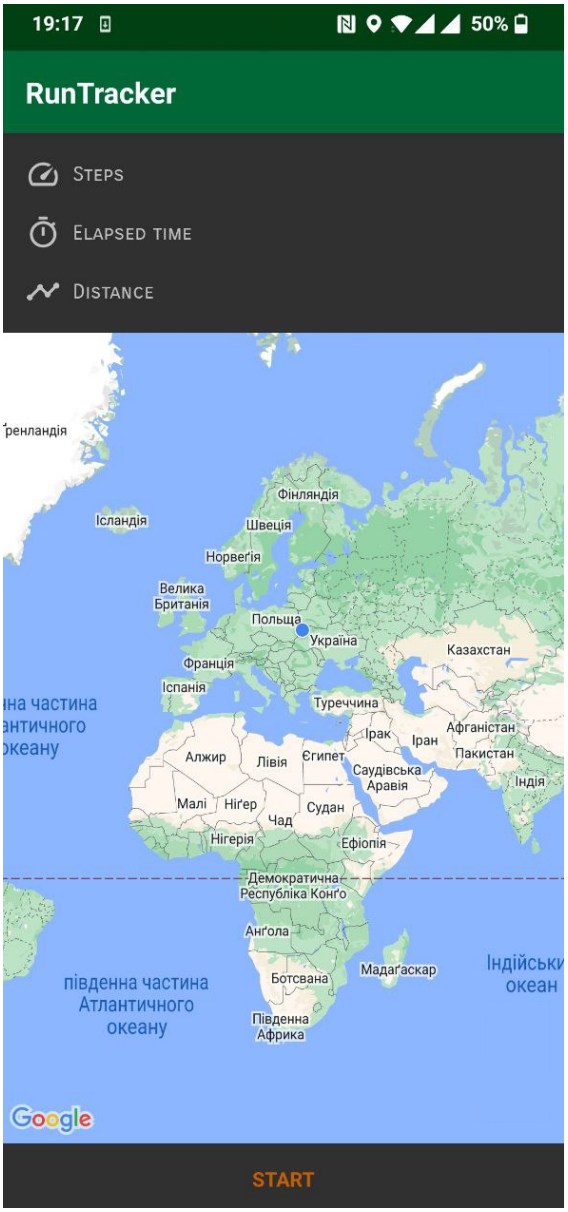

Рис. 3.1.4 - Загальна структура інтерфейсу

У верхній частині екрана розташовані три поля даних: кількість кроків, час, що минув, і відстань, яку користувач подолав за сеанс. Ці значення автоматично оновлюються під час руху користувача.

Нижче розташована карта, яка відображає розташування кожного користувача з використанням API ключа. Таким чином, ми можемо бачити своє власне місцезнаходження на карті.

І останнє, але не менш важливе, це кнопки «Start» та «Stop», за допомогою яких ми можемо розпочати та завершити відстеження користувача.Вони міняються в залежності від того, чи увімкнена програми чи вимкнена.

3.2 Розташування

Перш ніж виконайте будь-який запит, пов’язаний із розташуванням, вам потрібно правильно розмістити ваш зовнішній вигляд.

У папці програми відкриваємо build.gradle для модуля програми. В залежності лямбда пишемо такий код :

```
    implementation
'com.google.android.gms:play-services-maps:17.0.1'

    implementation
    'com.google.android.gms:play-services-location:18.0.
0'

    implementation
    'com.google.maps.android:android-maps-utils:2.2.3'

    implementation
'com.google.maps.android:maps-utils-ktx:3.2.0'
```

Нам потрібно отримати місцезнаходження користувача. Цей крок складається з двох дій, які відображають позицію користувача на карті.

Спочатку ми отримаємо координати користувача, а потім ми будуємо логіку запиту дозволу на місцезнаходження. Щойно користувач надасть цей дозвіл, програма відобразить його позицію на карті.

Отримання місцезнаходження користувача

У основному пакеті створюємо новий клас під назвою LocationProvider та вставляємо цей код у його файл:

```
    class LocationProvider ( private val activity:
AppCompatActivity) { //1 private val client
         by lazy {
LocationServices.getFusedLocationProviderClient(activity
) } //2 private val locations = mutableListOf<LatLng>()
//3 val liveLocation = MutableLiveData<LatLng>( ) //4
fun getUserLocation () {
        client.lastLocation.addOnSuccessListener {
location -> val latLng = LatLng(location.latitude,
location.longitude)
          locations.add(latLng)
          liveLocation.value = latLng
        }
```

```
    }
}
```

[6]

Ось розбивка коду:

1. По-перше, ми отримуємо об’єкт, який нам потрібен для визначення місцезнаходження користувача. Нам потрібна Activity, щоб створити його, але оскільки ми викличемо його лише після того, як переконаємося, що наша програма має дозвіл, збережимо його lazy, щоб LocationProvider створював його лише за потреби.
2. Цей рядок містить список місць, які нам знадобляться пізніше. На даний момент він містить лише початкову позицію користувача, але незабаром це зміниться.

3. Тут міститься інформація LiveData про місцезнаходження пристрою. Ми прослухаємо ці дані за кілька хвилин!

4. Нарешті, ось наш запит щодо позиції користувача. Ми беремо клієнта, який ми створили, і додаємо до нього слухача. Щоразу, коли він отримує нове місцезнаходження, він перетворює його на LatLng, який містить широту та довготу місцезнаходження, додає його до списку місцезнаходження та передає через LiveData.

[6]

Запит на дозвіл отримати місцезнаходження нашого користувача

Створюємо інший клас і називаємо його PermissionManager. Всередину вставте такий код:

```
    class PermissionsManager (
        activity: AppCompatActivity, private val
locationProvider: LocationProvider) { //1 private val
locationPermissionProvider =
activity.registerForActivityResult(
          ActivityResultContracts.RequestPermission()) {
granted -> if (granted) {
          locationProvider.getUserLocation()
        }
      } //2 fun requestUserLocation () {

locationPermissionProvider.launch(Manifest.permission.AC
CESS_FINE_LOCATION)
      }
    }
```

Ось розбивка нашого коду:

1. По-перше, ми реєструємо зворотний виклик Activity, коли користувач надає дозвіл. Цей зворотний виклик запускається та надає відповідь, щойно користувач натискає кнопку в діалоговому вікні дозволу. Якщо відповідь позитивна, ми запитуємо розташування пристрою з тим же класом, який створили в попередньому кроці.
2. Тоді ми запитуємо дозвіл. Щоразу, коли наша програма готова відобразити місцезнаходження, ми викликаємо цю функцію, система

запитує дозвіл, і щойно користувач торкається чогось, запускається зворотній виклик.

Перше тестування

В MapsActivity.kt і додаємо такі рядки вгорі.Вони потрібні лише для перевірки коду, який ми щойно написали:

```
private val locationProvider = LocationProvider(
this )
 private val permissionManager = PermissionsManager(
this , locationProvider)
[7]
```

Тепер прокрутимо вниз до onMapReady методу та видалимо увесь його вміст і замінимо на:

```
override fun onMapReady(googleMap: GoogleMap) {
  map = googleMap
  //1
  locationProvider.liveLocation.observe(this) {
latLng ->
map.animateCamera(CameraUpdateFactory.newLatLngZoom(latL
ng, 14f))
  }
  //2
  permissionManager.requestUserLocation()
  map.uiSettings.isZoomControlsEnabled = true
}
```

[8]

Ось розбивка коду:

1. По-перше, ми спостерігаємо за LiveData, створеним нами. Коли об’єкт LocationProvider знаходить нове місце, він випромінює його, і ми переміщуємо камеру відповідно.
2. Далі ми використовуємо PermissionManager, щоб запитати необхідний дозвіл.

Після запуску додатку карта вже буде готова та функціонуватиме належним чином, що дозволить нам отримувати інформацію про місцезнаходження користувача. Таким чином, ми зможемо з легкістю відстежувати своє положення на власному пристрої та бачити актуальну карту.

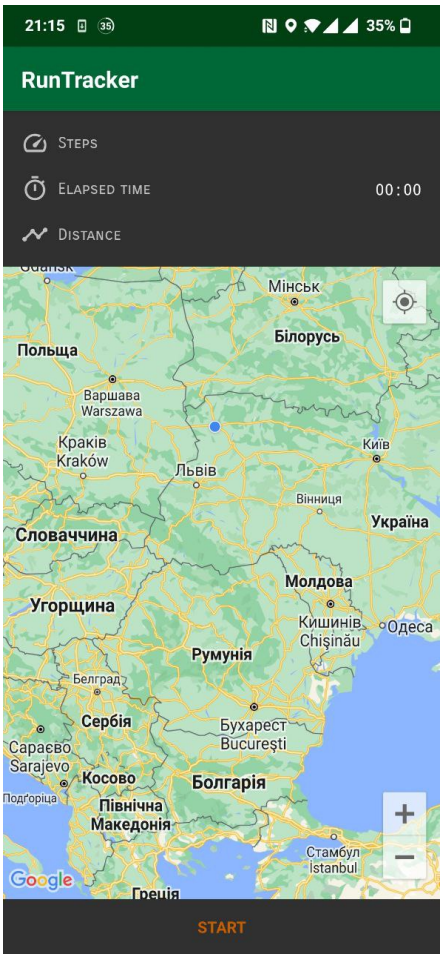

Рис. 3.2.1 - інтерфейс після додання коду

3.3 Відображення руху користувача

Тепер ми можемо відстежувати рух нашого користувача та позначати його місцезнаходження на карті, також намалюємо шлях, по якому він рухався. Для досягнення цієї мети нам потрібно внести зміни в клас LocationProvider.

Відкриваємо LocationsProvider.kt і прямо під locations списком додаємо distance:

```
private var distance = 0
```

Тепер наш код оновлює відстань кожного разу, коли ми фіксуємо нове положення користувача. Проте, нам також потрібно видавати відстань через LiveData, щоб мати можливість відслідковувати її зміни.

Під liveLocation додаємо наступний код:

```
val liveLocations = MutableLiveData<List<LatLng>>()
val liveDistance = MutableLiveData<Int>()
```

Оскільки для роботи API оновлення позиції потрібен зворотній виклик, нам необхідно створити цей зворотний виклик перед тим, як запитувати місцезнаходження. Ми можемо створити його як вкладений клас у файлі LocationProvider.kt. Ось як це можна зробити:

```
    private val locationCallback = object :
LocationCallback() {
override fun onLocationResult (result: LocationResult )
{ //1                                                val
currentLocation = result.lastLocation          val
latLng = LatLng(currentLocation.latitude,
currentLocation.longitude) //2 val lastLocation =
```

```
locations.lastOrNull () //3 if (lastLocation != null ) {
distance +=

SphericalUtil.computeDistanceBetween(lastLocation,
latLng).roundToInt()
            liveDistance.value = distance
          } //4
          locations.add(latLng)
          liveLocations.value = locations
      }
    }
```

Ось розбивка коду:

1. Ми отримуємо поточне записане місцезнаходження та перетворюємо його на LatLng, що дозволить нам легко нанести його на карту.
2. Далі ми перевіряємо, чи є інші місця. Якщо нам потрібно обчислити відстань між останньою точкою користувача та поточною точкою, нам знадобиться останнє розташування перед поточним.
3. Якщо поточне розташування не є першим, ми використовуємо функції SphericalUtil для обчислення відстані між двома точками та додавання її до відстані, яку ми випромінюємо через LiveData.
4. У випадку, якщо це перше місце, ми просто додаємо поточне місцезнаходження до списку записаних позицій.

Тепер, коли зворотний виклик готовий, нам потрібно зареєструвати його. Вставимо функцію trackUser в LocationProvider:

```
    fun trackUser() {
//1                                                    val
locationRequest = LocationRequest.create()
```

```
locationRequest.priority =
LocationRequest.PRIORITY_HIGH_ACCURACY
locationRequest.interval = 5000
      //2
client.requestLocationUpdates(locationRequest,
locationCallback,
Looper.getMainLooper())
    }
```

У першій частині фрагмента ми створюємо запит з високою точністю, оскільки нам потрібно відстежувати рух користувача, який рухається з відносно низькою швидкістю. Також ми встановлюємо інтервал запису в 5 секунд, що є більш ніж достатнім для такого типу руху.

Це дозволяє нам отримувати достатньо точні та звичні дані про місцезнаходження, забезпечуючи при цьому ефективне використання ресурсів пристрою. Таке налаштування є оптимальним для наших потреб і дозволяє нам ефективно відстежувати рух користувача.

Видаляємо цей зворотній виклик, коли користувач припиняє відстеження:

```
    fun stopTracking () {
      client.removeLocationUpdates(locationCallback)
      locations.clear()
      distance = 0
}
```

Цей метод виконує очищення всіх раніше зібраних даних, що дозволяє нам підготуватись до нового відстеження. Після його виконання клас стає готовим для збору нових даних і початку нового циклу відстеження. Таким чином, ми можемо збирати та аналізувати дані з початку, готові до нових вимірювань та відстеження користувача.

3.4 Представлення даних про місцезнаходження

Розділення трьох LiveDatas в інтерфейсі користувача, а також одного для підрахунку кроків, може бути незручним. Замість цього створимо щось, що може представляти стан екрану.

У пакеті основної програми створимо файл з назвою MVP.kt.

Для цього додатку використовується спрощена архітектура, яка краще підходить і полегшує роботу з цією темою.

У свій новий файл запишемо цей код:

```
data class Ui(
    val formattedPace: String,
    val formattedDistance: String,
    val currentLocation: LatLng?,
    val userPath: List<LatLng>
) {
 companion object {
 val EMPTY = Ui(
        formattedPace = "",
        formattedDistance = "",
        currentLocation = null,
        userPath = emptyList()
    )
```

Цей простий клас містить всю необхідну інформацію для належного відображення нашого інтерфейсу: відстань, яку ми пройшли, наше поточне місцезнаходження та список записаних місць.

Тепер нам потрібен механізм, який отримує дані з сенсорів, поєднує їх і повідомляє інтерфейсу про наявність нових даних. В тому ж файлі створюємо новий клас наступним чином:

```
class MapPresenter(private val activity:
AppCompatActivity) {
}
```

Оголосимо поле в презентаторі, яке видаватиме нові дані кожного разу, коли буде оновлення:

```
val ui = MutableLiveData(Ui.EMPTY)
```

Далі створимо необхідні залежності, такі як LocationProvider та PermissionManager:

```
private val locationProvider =
LocationProvider(activity)
private val permissionsManager =
PermissionsManager(activity, locationProvider)
```

Тепер ми повинні створити функцію, яка об'єднає всі дані в класі даних Ui. Додамо цю функцію до MapPresenter:

```
fun onViewCreated() {

}
```

Далі нам потрібно отримати три LiveData з LocationProvider, додавши спостерігача до кожного з них. Для цього нам знадобиться Activity, яке ми маємо у конструкторі, і використати функцію copy класу даних.

У методі onViewCreated пишемо такий код наступний код:

```
    locationProvider.liveLocations.observe(activity) {
locations ->
      val current = ui.value
      ui.value = current?.copy(userPath = locations)
    }

    locationProvider.liveLocation.observe(activity) {
currentLocation ->
      val current = ui.value
      ui.value = current?.copy(currentLocation =
currentLocation)
    }

    locationProvider.liveDistance.observe(activity) {
distance ->
      val current = ui.value
      val formattedDistance =
activity.getString(R.string.distance_value, distance)
      ui.value = current?.copy(formattedDistance =
formattedDistance)
    }
```

Для кожного LiveData, від яких ми отримуємо значення, форматуємо їх дані у більш зручний для читання вигляд та оновлюємо ui LiveData для видання нових даних.

Для того,щоб запобігти помилці “R.string.distance_value isn’t available” відкриваємо файл strings.xml і вставтетакий код:

```
<string name="distance_value">%d meters</string>
```

Ця мітка прийматиме число як параметр і помістить його там, де є %d, так що якщо ви передаєте 5, ви отримуєте рядок, який каже "5 метрів".

На наступному етапі ми додамо логіку для запиту дозволу від презентера, додавши onMapLoaded:

```
fun onMapLoaded() {
  permissionsManager.requestUserLocation()
}
```

Цей метод буде запускатись, як тільки контейнер Google Maps буде готовий.

Тепер додамо два методи, які будуть обробляти натискання користувачем кнопок "Старт" і "Стоп" у користувацькому інтерфейсі. Наразі вони будуть лише викликати відповідні методи у LocationProvider:

```
fun startTracking() {
  locationProvider.trackUser()
}
fun stopTracking() {
  locationProvider.stopTracking()
}
```

Тепер ми взаємодіємо лише з одним об'єктом, що є набагато зручніше і більш чистим, ніж працювати з багатьма об'єктами одночасно.

Запустимо наш додаток та натиснемо кнопку "Старт". Якщо ми будемо йти то програма буде відстежувати наш рух та відображати наше оновлене положення.

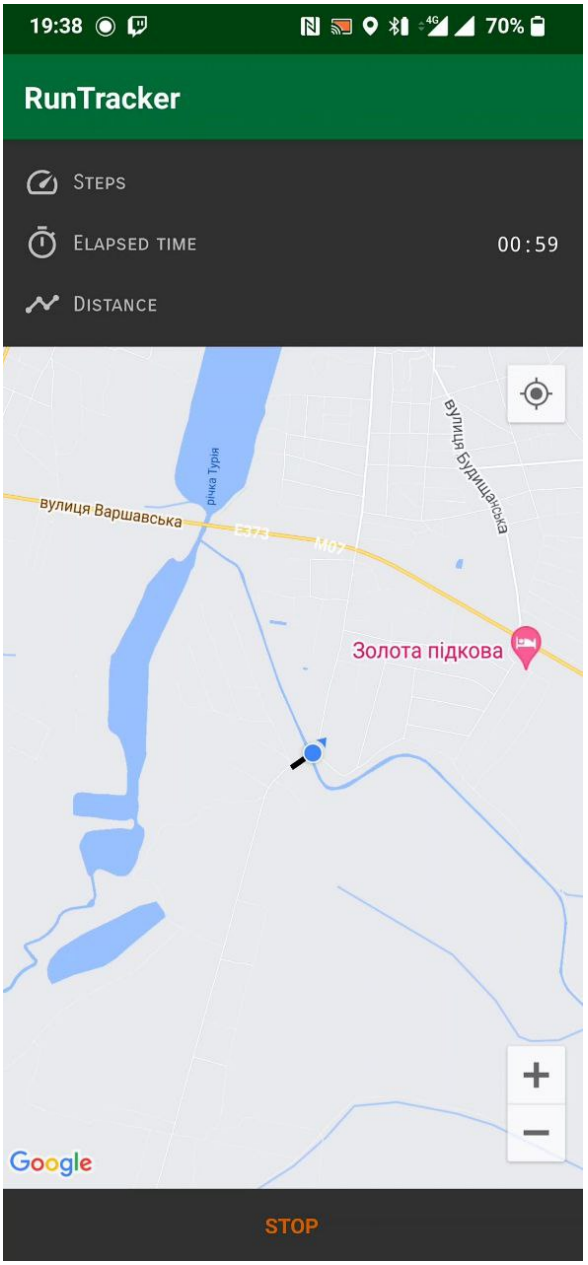

Рис. 3.4.1 - Відображення користувача на карті

3.5 Розпізнавання активності користувача

До цього моменту нам ще не вистачає такої функціональності, як темп або кількість кроків, які ми робимо, перед тим як оновити наш інтерфейс повністю ми маємо дізнатись чи користувач іде.

Для цього створимо ще один файл Kotlin і назвемо його StepCounter.kt. У цьому файлі ми будемо перевіряти діяльність користувача та кількість кроків, які ми робимо.

Спочатку створимо клас, який розширює SensorEventListener:

```
class StepCounter(private val activity:
AppCompatActivity) : SensorEventListener {

}
```

[9]

У верхній частині цього класу нам потрібно оголосити LiveData, яка містить кількість кроків, екземпляр SensorManager, який надає доступ до конкретних датчиків, та екземпляр as в якості полів. Крім того, ми оголосимо ще одну змінну, яка містить початкову кількість кроків, оскільки кількість кроків скидається при кожному перезавантаженні пристрою. Таким чином, у будь-який момент ми отримаємо кількість кроків, яку користувач пройшов з останнього перезавантаження телефону.

За допомогою цієї змінної, ми можемо отримати набагато точніше вимірювання кількості кроків, які користувач зробив під час активності.

Створимо наступний код в верхній частині файлу StepCounter:

```
val liveSteps = MutableLiveData<Int>()
private val sensorManager by lazy {
  activity.getSystemService(SENSOR_SERVICE) as
SensorManager
}

private val stepCounterSensor: Sensor? by lazy {
sensorManager.getDefaultSensor(Sensor.TYPE_STEP_COUN
TER)
}
private var initialSteps = -1
```

Оскільки ми реалізували інтерфейс SensorEventListener, нам також потрібно реалізувати кілька методів. Головним методом є onSensorChanged, де ми отримуємо потрібні оновлення.

В даному випадку нам потрібний лише другий метод onAccuracyChanged, якщо ми збираємось виконувати логіку на основі точності даних. Однак, в даному випадку ми можете просто повернути Unit:

```
override fun onSensorChanged(event: SensorEvent) {
  event.values.firstOrNull()?.toInt()?.let {
newSteps ->
    //1
    if (initialSteps == -1) {
```

```
            initialSteps = newSteps
        }

        //2
        val currentSteps = newSteps - initialSteps

        //3
        liveSteps.value = currentSteps
    }
}

override fun onAccuracyChanged(sensor: Sensor,
accuracy: Int) = Unit
```

Проаналізуємо перший метод. Як відомо, кількість кроків, яку ми отримуємо, є кількістю кроків, розрахованих від часу включення пристрою. Тому, щоб уникнути плутанини, ми відстежуємо кроки за допомогою змінної initialSteps.

1. Якщо ця змінна все ще має від'ємне значення, це означає, що ми ще не записали початок, тому ми встановлюємо її значення рівним даним з датчика.
2. Потім ми розраховуємо різницю між новими даними з датчика та initialSteps. Перший раз, коли ми отримуємо дані, ця різниця буде нульовою, як і очікувалося.
3. У останньому рядку ми встановлюємо LiveData рівним поточним крокам, які ми розрахували, тому всі оператори будуть реагувати на зміни.

Тепер нам потрібно ще два методи: один додає цей самий клас оператора до SensorManager, а інший видаляє оператора:

```
fun setupStepCounter() {
  if (stepCounterSensor != null) {
    sensorManager.registerListener(this,
stepCounterSensor, SENSOR_DELAY_FASTEST)
  }
}
fun unloadStepCounter() {
  if (stepCounterSensor != null) {
    sensorManager.unregisterListener(this)
  }
}
```

3.6 Запит на інші дозволи

Починаючи з Android Q, для визначення активності користувача потрібен новий дозвіл. Тому нам потрібно додати додаткову логіку всередині PermissionManager.

Спочатку змінимо конструктор PermissionManager так, щоб він приймав екземпляр StepCounter:

```
class PermissionsManager(activity:
AppCompatActivity,
    private val locationProvider: LocationProvider,
    private val stepCounter: StepCounter)
```

Потім оголосимо зворотний виклик, який виконується, коли користувач надає дозвіл на Android Q і новіші версії. Це налаштує лічильник кроків:

```
private val activityRecognitionPermissionProvider =
  activity.registerForActivityResult(
```

```
            ActivityResultContracts.RequestPermission()
        ) { granted ->
            if (granted) {
                stepCounter.setupStepCounter()
            }
    }
```

Наостанок, створимо метод, який запитує дозвіл на Android Q та новіші версії або просто налаштовує лічильник кроків на старіших версіях операційної системи:

```
    fun requestActivityRecognition() {
        if (Build.VERSION.SDK_INT >=
Build.VERSION_CODES.Q) {

activityRecognitionPermissionProvider.launch(Manifest.pe
rmission.ACTIVITY_RECOGNITION)
        } else {
            stepCounter.setupStepCounter()
        }
    }
```

Тепер повернемося до файлу MapPresenter.kt і додамо новий екземпляр StepCounter біля оголошення LocationProvider:

```
    private val stepCounter = StepCounter(activity)
```

Нам також потрібно передати цей новий `stepCounter` до оновленого `PermissionsManager`:

```
private val permissionsManager =
PermissionsManager(activity, locationProvider,
stepCounter)
```

В `onViewCreated` додаємо оператор до `LiveData`, щоб дані з `StepCounter` також передавалися разом з іншими, які ми раніше опрацьовували:

```
stepCounter.liveSteps.observe(activity) { steps ->
  val current = ui.value
  ui.value = current?.copy(formattedPace = "$steps")
}
```

Далі, в `startTracking` додаємо запит дозволу для кількості кроків. Оскільки користувач може в будь-який момент зняти дозвіл, кожного разу, коли він відстежує свою активність, нам потрібно буде знову запитати дозвіл. Якщо у нас вже є дозвіл, зворотний виклик виконається автоматично.

```
permissionsManager.requestActivityRecognition()
```

Під кінець вивантажуємо, StepCounter коли користувач припиняє записувати свою активність, у onStopTracking:

```
stepCounter.unloadStepCounter()
```

3.7 Користувацький інтерфейс

Відкриємо файл MapsActivity.kt. тавидалимо посилання на PermissionManager та LocationProvider. Натомість, додамо посилання на MapPresenter:

```
private val presenter = MapPresenter(this)
```

Після виконання цього кроку, у методі onCreate нашої активності використовуємо прив'язку, щоб отримати посилання на кнопку "Старт" і додамо таку логіку:

```
    binding.btnStartStop.setOnClickListener {
      if (binding.btnStartStop.text ==
getString(R.string.start_label)) {
        //1
        startTracking()

binding.btnStartStop.setText(R.string.stop_label)
      } else {
        //2
        stopTracking()

binding.btnStartStop.setText(R.string.start_label)
      }
    }
```

У цьому фрагменті перевіряється, чи мітка кнопки "START". Якщо так, ми починаємо відстеження і змінюємо мітку на "STOP". В іншому випадку ми припускаємо, що мітка вже є "STOP", зупиняємо відстеження і знову змінюємо мітку на "START".

Далі додамо зворотний виклик оператора безпосередньо під прив’язками для кнопки:

```
    presenter.onViewCreated()
```

Цей метод повідомляє презентеру, що ваш користувацький інтерфейс готовий і потрібно прикріпити всі оператори до LiveData.

Далі ми змінимо користувацький інтерфейс, щоб він краще відповідав коду, який ми збираємось написати.

Відкриємо файл activity_maps.xml і замінимо тег include на:

```
<include
        android:id="@+id/container"
        layout="@layout/layout_indicators" />
```

Тут ми додаєте новий ідентифікатор контейнера, який будемо використовувати пізніше для доступу до полів у макеті.

```
<Chronometer
        android:id="@+id/txtTime"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:fontFamily="monospace"
        tools:text="45 minutes"

app:layout_constraintBottom_toBottomOf="@+id/txtTimeLabe
l"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="1.0"
app:layout_constraintStart_toEndOf="@+id/txtTimeLabel"
app:layout_constraintTop_toTopOf="@+id/txtTimeLabel" />
```

Тут ми змінюєте TextView, щоб він відстежував пройдений час набагато простіше і відображав його у належному TextView.

Для того щоб зняти попередження з IDE додаємо кілька рядків коду. Та вставимо цей код в клас MapsActivity як метод класу:

```
private fun startTracking() {
```

```
        //1
        binding.container.txtPace.text = ""
        binding.container.txtDistance.text = ""
        //2
        binding.container.txtTime.base =
SystemClock.elapsedRealtime()
        binding.container.txtTime.start()
        //3
        map.clear()
       //4 presenter.startTracking()
      }
      private fun stopTracking() {
        presenter.stopTracking()
        binding.container.txtTime.stop()
      }
```

Тут відбуваються кілька корисних дій:

1. По-перше, ми скидаємо значення полів pace та distance, оскільки ми хочемо записати нову активність.
2. По-друге, ми встановлюємо мітку часу на поточний час та перезапускаємо її. Ми робимо це, оскільки лічильник часу починає відлік з останнього записаного моменту, тому якщо ми його не скинемо, у нас будуть дивні дані.
3. По-третє, ми очищаємо мапу. Нам не потрібні старі маркери, оскільки ми цікавимося лише новими, які приходять кожні 5 секунд
4. І наостанок, але не менш важливо, ми просимо презентера розпочати відстеження, і всі дані будуть заново відображатись

Коли ми припиняємо відстеження, ми зупиняємо всі функції презентера і також зупиняємо лічильник часу. Наразі нам не потрібно скидати жодні значення. Таким чином, у нас також є красивий резюмуючий екран для наших користувачів.

Далі для коректної роботи додатка дописуємо цей код:

```
@SuppressLint("MissingPermission")
private fun updateUi(ui: Ui) {
  //1
  if (ui.currentLocation != null &&
ui.currentLocation != map.cameraPosition.target) {
    map.isMyLocationEnabled = true

map.animateCamera(CameraUpdateFactory.newLatLngZoom(ui.c
urrentLocation, 14f))
  }
  //2
  binding.container.txtDistance.text =
ui.formattedDistance
  binding.container.txtPace.text = ui.formattedPace
  //3
  drawRoute(ui.userPath)
}
```

Ось що ми зробили:

1. Ми перевіряємо, що нове положення не є нульовим і відрізняється від положення на карті. Якщо ми пропустимо цей крок, можуть виникнути помилки через те, що положення може бути нульовим або спричиняти дивні миготіння, коли ми переміщуємо камеру в ту саму

позицію. Якщо перевірка пройшла успішно, ми переміщуємо камеру до нового місцезнаходження, як ми робили раніше.

2. Далі, ми прив'язуємо форматовані дані до прив'язок.
3. Після цього м малюємо шлях по якому ми рухаємось.

Нижче в MapsActivity додаємо таку функцію :

```
private fun drawRoute(locations: List<LatLng>) {
  val polylineOptions = PolylineOptions()
  map.clear()
  val points = polylineOptions.points
  points.addAll(locations)
  map.addPolyline(polylineOptions)
}
```

Ця функція містить все необхідне для малювання лінії з декількома точками на екрані.

1. Спочатку ми створюємо об'єкт PolylineOptions, який містить точки для малювання на мапі.
2. Потім ми очищаємо все на мапі.
3. Ми додаємо всі нові локації та малюємо лінію на мапі.

Змінюємо метод onMapReady таким чином:

```
override fun onMapReady(googleMap: GoogleMap) {
  map = googleMap

  //1
  presenter.ui.observe(this) { ui ->
    updateUi(ui)
  }
```

```
//2
presenter.onMapLoaded()
map.uiSettings.isZoomControlsEnabled = true}
```

1. У методі `onMapReady` ми налаштовуємо реагування на зміни даних від презентера, щоб оновлювати користувацький інтерфейс відповідно.
2. Також ми повідомляємо презентер, що карта завантажена, щоб він міг виконати необхідні дії, наприклад, запитати дозвіл на місцезнаходження у разі його відсутності.

Кінцевий вигляд програми

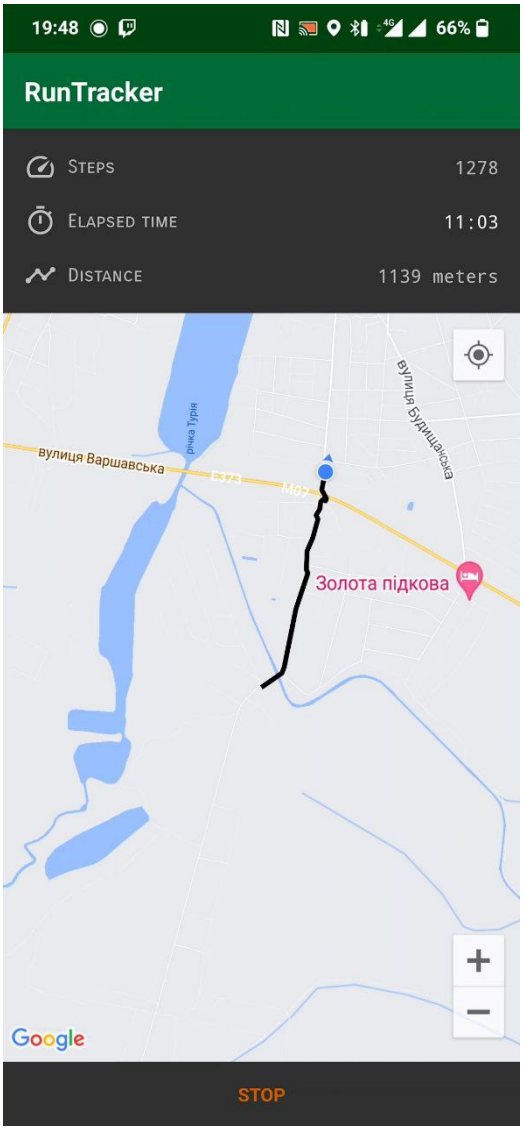

Рис. 3.4.1 - Кінцевий вигляд програми

Висновки

Під час виконання курсової роботи розроблено додаток для відстеження руху користувача. Цей додаток допомагає користувачам залишатись мотивованими та відстежувати прогрес своїх результатів з часом.

Для розробки додатку використано відповідні інструменти, такі як Android Studio, Google Maps API та мову програмування Kotlin. Для перевірки працездатності додатку використано Android-смартфон з операційною системою Android 11. Додаток продемонстрував стабільну роботу без відхилень.

У самому додатку реалізовано функціонал, який включає підрахунок кроків, відлік часу та пройдену відстань. Одним з ключових елементів додатку є можливість побудови шляху користувача на мапі, що дозволяє візуально спостерігати за своїм переміщенням.

Загалом, цей додаток може бути цінним інструментом для людей, які бажають відстежувати своє переміщення та залишатись мотивованими. Він простий у використанні та може бути налаштований з урахуванням потреб кожного окремого користувача.

У майбутньому можна розглянути ряд покращень для цього додатку:

1. Додати можливість поділитися результатами пробігу в соціальних мережах, щоб користувачі могли ділитися своїми досягненнями з друзями та родиною.

2. Додати таблицю лідерів, де користувачі зможуть змагатися між собою за кращі результати у контексті часу та відстані.

3. Включити відображення середньої швидкості користувача під час сесії пробігу.

4. Додати загальний звіт, який відображатиме характеристики всіх сесій пробігу користувача.

Ці покращення розширять функціональність додатку та зроблять його ще більш цікавим та корисним для користувачів.

Це лише кілька ідей для поліпшення додатку. З невеликими творчими зусиллями додаток може бути перетворено в потужний інструмент для бігунів будь-якого рівня.

ПЕРЕЛІК ПОСИЛАНЬ

1. Strava [Електронний ресурс]. — Режим доступу: https://blog.strava.com/
2. Nike Run Club [Електронний ресурс]. — Режим доступу: https://www.nike.com/nrc-app
3. Runkeeper [Електронний ресурс]. — Режим доступу: https://runkeeper.com/cms/
4. Android Studio Tutorial [Електронний ресурс]. — Режим доступу: https://www.javatpoint.com/android-tutorial
5. Жемеров Д., Ісакова С. *Kotlin в дії*. Manning, 2017. –23с.
6. Відкладена ініціалізація в Kotlin [Електронний ресурс]. — Режим доступу: https://www.baeldung.com/kotlin/lazy-initialization
7. Kotlin Android Google Map Current Location [Електронний ресурс]. — Режим доступу: https://www.javatpoint.com/kotlin-android-google-map-current-location
8. Introduction to Google Maps API for Android with Kotlin [Електронний ресурс]. — Режим доступу: https://www.kodeco.com/230-introduction-to-google-maps-api-for-android-with-kotlin
9. How to Build a Step Counting Application in Android Studio? [Електронний ресурс]. — Режим доступу: https://www.geeksforgeeks.org/how-to-build-a-step-counting-application-in-android-studio/

Додаток

LocatuonProvider.kt

```
package com.raywenderlich.android.runtracker

import android.annotation.SuppressLint
import android.os.Looper
import androidx.appcompat.app.AppCompatActivity
import androidx.lifecycle.MutableLiveData
import com.google.android.gms.location.LocationCallback
import com.google.android.gms.location.LocationRequest
import com.google.android.gms.location.LocationResult
import com.google.android.gms.location.LocationServices
import com.google.android.gms.maps.model.LatLng
import com.google.maps.android.SphericalUtil
import kotlin.math.roundToInt

@SuppressLint("MissingPermission")
class LocationProvider(private val activity:
AppCompatActivity) {

 private val client by lazy {
LocationServices.getFusedLocationProviderClient(activity
) }

 private val locations = mutableListOf<LatLng>()
 private var distance = 0

 val liveLocations = MutableLiveData<List<LatLng>>()
```

```
 val liveDistance = MutableLiveData<Int>()
 val liveLocation = MutableLiveData<LatLng>()

 private val locationCallback = object :
LocationCallback() {
   override fun onLocationResult(result: LocationResult)
{
     val currentLocation = result.lastLocation
     val latLng = LatLng(currentLocation.latitude,
currentLocation.longitude)

     val lastLocation = locations.lastOrNull()

     if (lastLocation != null) {
       distance +=
SphericalUtil.computeDistanceBetween(lastLocation,
latLng).roundToInt()
       liveDistance.value = distance
     }

     locations.add(latLng)
     liveLocations.value = locations
   }
 }

 fun getUserLocation() {
   client.lastLocation.addOnSuccessListener { location
->
```

```
      val latLng = LatLng(location.latitude,
location.longitude)
      locations.add(latLng)
      liveLocation.value = latLng
    }
  }

  fun trackUser() {
    val locationRequest = LocationRequest.create()
    locationRequest.priority =
LocationRequest.PRIORITY_HIGH_ACCURACY
    locationRequest.interval = 5000
    client.requestLocationUpdates(locationRequest,
locationCallback, Looper.getMainLooper())
  }

  fun stopTracking() {
    client.removeLocationUpdates(locationCallback)
    locations.clear()
    distance = 0
  }
}
```

MapsActivity.kt

```
package com.raywenderlich.android.runtracker

import android.annotation.SuppressLint
import android.os.Bundle
```

```
import android.os.SystemClock
import androidx.appcompat.app.AppCompatActivity
import com.google.android.gms.maps.CameraUpdateFactory
import com.google.android.gms.maps.GoogleMap
import com.google.android.gms.maps.OnMapReadyCallback
import com.google.android.gms.maps.SupportMapFragment
import com.google.android.gms.maps.model.LatLng
import com.google.android.gms.maps.model.PolylineOptions
import
com.raywenderlich.android.runtracker.databinding.Activit
yMapsBinding

class MapsActivity : AppCompatActivity(),
OnMapReadyCallback {

 private lateinit var map: GoogleMap
 private lateinit var binding: ActivityMapsBinding

 private val presenter = MapPresenter(this)

 override fun onCreate(savedInstanceState: Bundle?) {
   setTheme(R.style.AppTheme)

   super.onCreate(savedInstanceState)

   binding = ActivityMapsBinding.inflate(layoutInflater)
   setContentView(binding.root)
```

```
   // Obtain the SupportMapFragment and get notified
when the map is ready to be used.
   val mapFragment = supportFragmentManager
       .findFragmentById(R.id.map) as SupportMapFragment
   mapFragment.getMapAsync(this)

   binding.btnStartStop.setOnClickListener {
     if (binding.btnStartStop.text ==
getString(R.string.start_label)) {
       startTracking()
       binding.btnStartStop.setText(R.string.stop_label)
     } else {
       stopTracking()

binding.btnStartStop.setText(R.string.start_label)
     }
   }

   presenter.onViewCreated()
 }

 override fun onMapReady(googleMap: GoogleMap) {
   map = googleMap

   presenter.ui.observe(this) { ui ->
     updateUi(ui)
   }

   presenter.onMapLoaded()
```

```
    map.uiSettings.isZoomControlsEnabled = true
  }

  private fun startTracking() {
    binding.container.txtPace.text = ""
    binding.container.txtDistance.text = ""
    binding.container.txtTime.base =
SystemClock.elapsedRealtime()
    binding.container.txtTime.start()
    map.clear()

    presenter.startTracking()
  }

  private fun stopTracking() {
    presenter.stopTracking()
    binding.container.txtTime.stop()
  }

  @SuppressLint("MissingPermission")
  private fun updateUi(ui: Ui) {
    if (ui.currentLocation != null && ui.currentLocation
!= map.cameraPosition.target) {
      map.isMyLocationEnabled = true

map.animateCamera(CameraUpdateFactory.newLatLngZoom(ui.c
urrentLocation, 14f))
    }
```

```
    binding.container.txtDistance.text =
ui.formattedDistance
    binding.container.txtPace.text = ui.formattedPace
    drawRoute(ui.userPath)
  }

  private fun drawRoute(locations: List<LatLng>) {
    val polylineOptions = PolylineOptions()

    map.clear()

    val points = polylineOptions.points
    points.addAll(locations)

    map.addPolyline(polylineOptions)
  }
```

MVP.kt

```
package com.raywenderlich.android.runtracker

import androidx.appcompat.app.AppCompatActivity
import androidx.lifecycle.MutableLiveData
import com.google.android.gms.maps.model.LatLng

class MapPresenter(private val activity:
AppCompatActivity) {

  val ui = MutableLiveData(Ui.EMPTY)
```

```
 private val locationProvider =
LocationProvider(activity)

 private val stepCounter = StepCounter(activity)

 private val permissionsManager =
PermissionsManager(activity, locationProvider,
stepCounter)

 fun onViewCreated() {

   locationProvider.liveLocations.observe(activity) {
locations ->
     val current = ui.value
     ui.value = current?.copy(userPath = locations)
   }

   locationProvider.liveLocation.observe(activity) {
currentLocation ->
     val current = ui.value
     ui.value = current?.copy(currentLocation =
currentLocation)
   }

   locationProvider.liveDistance.observe(activity) {
distance ->
     val current = ui.value
     val formattedDistance =
activity.getString(R.string.distance_value, distance)
```

```
      ui.value = current?.copy(formattedDistance =
formattedDistance)
    }

    stepCounter.liveSteps.observe(activity) { steps ->
      val current = ui.value
      ui.value = current?.copy(formattedPace = "$steps")
    }
  }

  fun onMapLoaded() {
    permissionsManager.requestUserLocation()
  }

  fun startTracking() {
    permissionsManager.requestActivityRecognition()
    locationProvider.trackUser()

    val currentUi = ui.value
    ui.value = currentUi?.copy(
        formattedPace = Ui.EMPTY.formattedPace,
        formattedDistance = Ui.EMPTY.formattedDistance
    )
  }

  fun stopTracking() {
    locationProvider.stopTracking()
    stepCounter.unloadStepCounter()
  }
```

```
}

data class Ui(
   val formattedPace: String,
   val formattedDistance: String,
   val currentLocation: LatLng?,
   val userPath: List<LatLng>
) {

 companion object {

   val EMPTY = Ui(
       formattedPace = "",
       formattedDistance = "",
       currentLocation = null,
       userPath = emptyList()
   )
 }
}
```

PremissionManager.kt

```
package com.raywenderlich.android.runtracker

import android.Manifest
import android.os.Build
import
androidx.activity.result.contract.ActivityResultContract
s
import androidx.appcompat.app.AppCompatActivity
```

```
class PermissionsManager(activity: AppCompatActivity,
   private val locationProvider: LocationProvider,
   private val stepCounter: StepCounter) {

 private val locationPermissionProvider =
activity.registerForActivityResult(
     ActivityResultContracts.RequestPermission()) {
granted ->
   if (granted) {
     locationProvider.getUserLocation()
   }
 }

 private val activityRecognitionPermissionProvider =
activity.registerForActivityResult(
     ActivityResultContracts.RequestPermission()) {
granted ->
   if (granted) {
     stepCounter.setupStepCounter()
   }
 }

 fun requestUserLocation() {

locationPermissionProvider.launch(Manifest.permission.AC
CESS_FINE_LOCATION)
 }
```

```
 fun requestActivityRecognition() {
   if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.Q) {

activityRecognitionPermissionProvider.launch(Manifest.pe
rmission.ACTIVITY_RECOGNITION)
   } else {
     stepCounter.setupStepCounter()
   }
 }
}
```

StepCounter.kt

```
package com.raywenderlich.android.runtracker

import android.content.Context.SENSOR_SERVICE
import android.hardware.Sensor
import android.hardware.SensorEvent
import android.hardware.SensorEventListener
import android.hardware.SensorManager
import android.hardware.SensorManager.*
import androidx.appcompat.app.AppCompatActivity
import androidx.lifecycle.MutableLiveData

class StepCounter(private val activity:
AppCompatActivity) : SensorEventListener {

 interface StepUpdater {

   fun onStepUpdated(steps: Int)
 }
```

```
 val liveSteps = MutableLiveData<Int>()

 private val sensorManager by lazy {
   activity.getSystemService(SENSOR_SERVICE) as
SensorManager
 }

 private val stepCounterSensor: Sensor? by lazy {

sensorManager.getDefaultSensor(Sensor.TYPE_STEP_COUNTER)
 }

 private var initialSteps = -1

 fun setupStepCounter() {
   if (stepCounterSensor != null) {
     sensorManager.registerListener(this,
stepCounterSensor, SENSOR_DELAY_FASTEST)
   }
 }

 override fun onSensorChanged(event: SensorEvent) {
   event.values.firstOrNull()?.toInt()?.let { newSteps
->
     if (initialSteps == -1) {
       initialSteps = newSteps
     }
```

```
        val currentSteps = newSteps - initialSteps

        liveSteps.value = currentSteps
      }
    }

    fun unloadStepCounter() {
      if (stepCounterSensor != null) {
        sensorManager.unregisterListener(this)
      }
    }

    override fun onAccuracyChanged(sensor: Sensor,
accuracy: Int) = Unit
```