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

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:
Завідувач кафедри
_____ Сергій СТІРЕНКО
(підпис)
“__” _____2024 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою “Комп'ютерні системи та мережі”

спеціальності 123 “Комп’ютера інженерія”

на тему: Застосунок для спортивних велосипедних подорожей

Виконав : студент __4__ курсу, групи _ІО-03
(шифр групи)

_____Малій Микита Максимович_____
(прізвище, ім’я, по батькові) (підпис)

Керівник _____доцент, к.т.н. Волокита А. М._____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант (нормоконтроль) Іваніщев Б. В_____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2024 р.

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

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалавр)

Освітньо-професійна програма

“ Комп'ютерні системи та мережі”

спеціальності 123 “ Комп’ютера інженерія ”

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Сергій СТІРЕНКО
(підпис)
“__” _____2024 р.

ЗАВДАННЯ
на бакалаврський дипломний проєкт студента

Малія Микити Максимовича

1. Тема проєкту *Застосунок для спортивних велосипедних подорожей*
керівник проєкту Волокита Артем Миколайович, доцент, к.т.н.,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від

2. Термін здачі студентом закінченого проєкту *07 червня 2024 р.*

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

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

Розділ 1. Велоіндустрія: огляд і порівняння застосунків для велоспорту.

Розділ 2. Огляд технологій та підходів для розробки ПЗ.

Розділ 3. Деталі реалізації системи

Розділ 4. Огляд та тестування розробленного застосунку

5. Перелік графічного матеріалу (з точним позначенням обов'язкових креслень) структурна схема системи, функціональна схема (діаграма класів), алгоритм дій програмного забезпечення.

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

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормоконтроль	Іваніщев Б. В.		

7. Дата видачі завдання «12» жовтня 2023 р.

Календарний план

№ п/п	Найменування етапів дипломного проєкту	Терміни виконання етапів проєкту	Примітки
1.	*Затвердження теми проєкту*	28.11.2023-01.12.2023	
2.	*Вивчення та аналіз завдання*	01.12.2023-25.03.2024	
3.	*Розробка архітектури та загальної структури системи*	25.03.2024-01.04.2024	
4.	*Розробка структур окремих підсистем*	25.03.2024-05.04.2024	
5.	*Програмна реалізація системи*	05.04.2024-19.05.2024	
6.	*Оформлення пояснювальної записки*	19.05.2024-25.05.2024	
7.	*Захист програмного продукту*	25.05.2024-05.06.2024	
8.	*Передзахист*	06.06.2024	
9.	*Захист*		

Студент-дипломник _____ Малій МИКИТА
(підпис)

Керівник проєкту _____ Артем ВОЛОКИТА

АНОТАЦІЯ

У даній дипломній роботі було виконано всебічне дослідження, розробку та тестування мобільного застосунку для велосипедних подорожей на платформі Android, з метою створення зручного інструменту для велосипедистів. Було проаналізовано, виклики велосипедистів, способи покращення велоподорожей, а також проведено огляд існуючих рішень на ринку. Реалізовано базові функції застосунку, включаючи авторизацію, побудову маршрутів, відслідковування велоподорожей та експорт даних. Інтегровано базу даних Firestore, проведено детальний огляд інтерфейсу та процес створення релізної версії. Робота над проектом включала аналіз ринку, вибір найкращих технологій, реалізацію ключових функцій та тестування кінцевого продукту, що забезпечило конкурентоспроможність та привабливість застосунку для користувачів.

Ключові слова: Android, Kotlin, мобільний застосунок, mapbox, будування маршрутів, аналіз тренувань, GPX, TCX, Strava.

ANNOTATION

This thesis involved comprehensive research, development, and testing of a mobile application for cycling trips on the Android platform, aimed at creating a convenient tool for cyclists. The challenges faced by cyclists, ways to improve cycling trips, and an overview of existing solutions on the market were analyzed. The basic functions of the application were implemented, including user authentication, route building, trip tracking, and data export. The Firestore database was integrated, and a detailed review of the interface and the process of creating the release version was conducted. The project work included market analysis, selection of the best technologies, implementation of key functions, and testing of the final product, ensuring the application's competitiveness and attractiveness to users.

Keywords: Android, Kotlin, mobile application, Mapbox, route building, workout analysis, GPX, TCX, Strava.

справки	Формат	*Значення*	*Найменування*	Кіл. листів	№ екземпля	*Додаток*
			Завдання на дипломний проект	2		
	А4	***ІАЛЦ.467200.002 ТЗ***	Технічне завдання	6		
	А4	***ІАЛЦ.467200.003 ПЗ***	Пояснювальна записка	94		
	А4	***ІАЛЦ.467200.004 Д1***	Структурна схема системи	1		
	А4	***ІАЛЦ.4672008.005 Д2***	Функціональна схема	1		
	А4	***ІАЛЦ.467200.006 Д3***	Алгоритм дій програмного забезпечення	1		
	А4	***ІАЛЦ.467200.007 Д4***	Текст програмного коду	50		

Зм	*Лист*	*№ докум.*	*Підп*	*Дата*	***ІАЛЦ.467200.001 ОА***		
Розроб		Малій М.М.			*Застосунок для спортивних велосипедних подорожей* Опис альбому	Літ.	Аркуш 1 / Аркушів 1
Перев		Волокита А.М.					
						КПІ ім. Ігоря Сікорського ФІОТ ІО-03	

ТЕХНІЧНЕ ЗАВДАННЯ
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «*Застосунок для спортивних велосипедних подорожей*»

Київ 2024

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Дане технічне завдання поширюється на розробку програмного забезпечення для мобільних пристроїв на базі операційної системи Android, призначеного для спортивних велосипедних подорожей. Передбачається подальша підтримка та вдосконалення розроблених функцій.

Програмне забезпечення призначене для використання як велосипедистами-любителями, так і професійними спортсменами з метою покращення тренувань. Додаток надаватиме користувачам можливість планувати маршрути, відстежувати прогрес, аналізувати результати тренувань і ділитися досягненнями.

2. ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки мобільного застосунку на базі операційної системи Android для спортивних велосипедних подорожей є завдання для виконання роботи кваліфікаційно-освітнього рівня «бакалавр інженерії програмного забезпечення», який був затверджений факультетом “Інформатики та обчислювальної техніки” кафедрою обчислювальної техніки Національного технічного Університету України «Київський Політехнічний інститут ім. Ігоря Сікорського».

3. ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою та призначенням роботи є створення застосунку, для спортивних велосипедних подорожей, який дозволить користувачам планувати маршрути, відстежувати тренування та аналізувати результати. Забезпечити велосипедистів-любителів та професійних спортсменів зручним інструментом

для покращення ефективності тренувань, планування поїздок та моніторингу фізичної активності.

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1. Вимоги до функціональних характеристик

Програмне забезпечення повинно виконувати наступні функції

- Реєстрація користувача по пошті та паролі
- Реєстрація користувача по обліковому запису Google
- Авторизація користувача
- Створення маршрутів, вибираючи точки на мапі, та перегляд їх метрик.
- Збереження створених маршрутів у базі даних.
- Експорт маршрутів у форматі GPX для подальшого використання.
- Експорт маршрутів у форматі TCX для подальшого використання.
- Видалення маршруту
- Запуск та зупинка сесій тренування по збережених маршрутах.
- Відстеження основних метрик під час тренування, таких як дистанція, час, швидкість, висота, калорії та інші.
- Збереження даних про тренування у базі даних для подальшого аналізу.
- Відображення детальної інформації про тренування у вигляді графіків та таблиць.
- Порівняння результатів різних тренувань.
- Експорт даних про тренування у застосунок Strava.

4.2. Вимоги до безпеки

- Забезпечення безпеки персональних даних користувачів.
- Шифрування збережених даних та захист від несанкціонованого доступу.

4.3. Вимоги до технічних засобів

4.3.1. Апаратні вимоги

- Процесор: Чотирьох ядерний процесор з тактовою частотою не менше 1.8 ГГц.
- Оперативна пам’ять: Мінімум 2 ГБ оперативної пам’яті для стабільної роботи додатку.
- Внутрішнє сховище: Мінімум 100 МБ вільного місця для встановлення додатку, з можливістю зберігання даних (карт, маршрутів, тренувальних даних тощо) на додатковій карті пам'яті або в хмарному сховищі.
- Графічний процесор (GPU): Підтримка OpenGL ES 3.0 для належної роботи графічних компонентів додатку.

4.3.2. Програмні вимоги

- Операційна система: Android версії 7.0 або новіше.
- Додаткове програмне забезпечення: Підключення до інтернету для використання онлайн-карт та синхронізації даних з сервісами, такими як Strava.

4.3.3. Комунікаційні вимоги

- GPS-модуль: Пристрій повинен мати вбудований GPS-модуль для відстеження маршрутів та визначення місцезнаходження користувача.

- Мобільні мережі та Wi-Fi: Підтримка 3G/4G/LTE для передачі даних та завантаження карт в реальному часі, а також модуль Wi-Fi для доступу до інтернету в зонах покриття.

4.3.4. Сенсори

- Акселерометр і гіроскоп: Для точного відстеження руху та нахилу пристрою під час тренувань.
- Барометр: Для визначення висоти над рівнем моря (якщо підтримується пристроєм).

4.3.5. Дисплей

- Розмір екрану: Мінімальний розмір екрану 4.5 дюйма для зручного відображення карт та іншої інформації.
- Роздільна здатність: Роздільна здатність екрану не менше 720x1280 пікселів для чіткого відображення даних.

4.3.6. Акумулятор

- Ємність батареї: Мінімум 3000 мАг для забезпечення тривалої роботи додатку під час тривалих поїздок без необхідності частого заряджання.

4 ДЖЕРЕЛА РОЗРОБКИ

Для написання даного дипломного проекту використовувались такі джерела інформації: наукові публікації, офіційні документи, статті з онлайн-ресурсів та науково-технічна література, що стосуються даної тематики.

5 ЕТАПИ ТА СТАДІЇ РОЗРОБКИ

Назва етапів виконання	Термін виконання
Затвердження теми роботи	28.11.2023-01.12.2023
Вивчення та аналіз завдання	01.12.2023-25.03.2024
Розробка архітектури та загальної структури системи	25.03.2024-01.04.2024
Розробка структур окремих частин системи	25.03.2024-05.04.2024
Програмна реалізація системи	05.04.2024-19.05.2024
Виправлення помилок	19.05.2024-25.05.2024
Оформлення пояснювальної записки	06.06.2024

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «*Застосунок для спортивних велосипедних подорожей*»

Київ 2024

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

XML	eXtensible Markup Language
GPS	Global Positioning System
GPX	GPS Exchange Format
TCX	Training Center XML
FIT	Flexible and Interoperable Data Transfer
ОС	Операційна система
IOS	Iphone Operating System
API	Application Programming Interface
IDE	Integrated Development Environment
SDK	Software Development Kit
ПЗ	Програмне забезпечення
БД	База даних
APK	Android Package
AAB	Android App Bundle

ВСТУП

Під час війни люди переживають щоденне психологічне навантаження – це і біль, і злість, невизначеність майбутнього, страх за здоров'я та життя своїх рідних та близьких тощо. Все це послаблює фізичне та насамперед психологічне здоров'я людей. Оскільки зараз наша країна як ніколи потребує сильних та здорових людей. особливо важливо підтримувати фізичну активність, яка дасть змогу організму функціонувати у режимі регулярного стресу.

Серед великої кількості видів спорту, все більшої популярності в Україні набирає велоспорт, що має багато різновидів. Я сам та мої друзі вже понад 5 років захоплюємось цим видом спорту, що дозволяє підтримувати фізичну форму, відволікатися від напруженості сьогодення та просто отримувати насолоду від швидкості, що в свою чергу надає змогу відчути себе щасливим та більш впевненим у себе та у своїх силах.

Створення програм і додатків для відстеження спортивної активності, допомагають тренуватися та дозволяють всім, хто обожнює велопрогулянки або займається велоспортом, зберігати результати своєї активності, фіксувати її, аналізувати, порівнювати та ділитися результатом з друзями.

У цьому контексті, розробка застосунку для спортивних велосипедних подорожей має велике значення. Метою даного дипломного проекту є розробка та реалізація застосунку для мобільних пристроїв на платформі Android, який дозволить користувачам побудувати оптимальні маршрути, використовуючи інтерактивну мапу, та відстежувати свої тренування, а також ефективно аналізувати свої досягнення. Буде розглянуто процес розробки застосунку, описано його функціонал та можливості, а також проведено аналіз його ефективності та можливостей подальшого розвитку.

РОЗДІЛ 1. ВЕЛОІНДУСТРІЯ: ОГЛЯД І ПОРІВНЯННЯ ЗАСТОСУНКІВ ДЛЯ ВЕЛОСПОРТУ

1.1 Велосипеди та їх соціокультурний вплив: статистика та факти

1.1.1 Популярність велосипедів та “велосипедний бум” 2020 року

Чи знаєте ви, що велосипеди є одним із найпоширеніших видів транспорту у світі? Так, і це не порожні слова, давайте подивимося на статистичні дані. За даними Європейського парламенту [1], сьогодні у світі існує приблизно 1 мільярд велосипедів – стільки ж, скільки і легкових автомобілів.

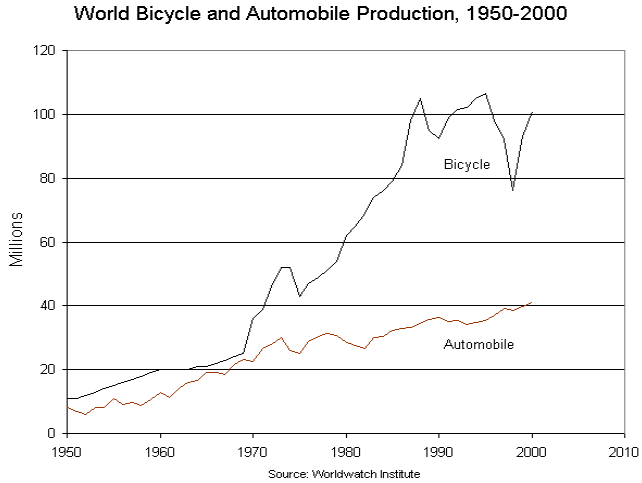


Рисунок. 1.1 – Графік виробництва велосипедів та автомобілів (1950-2000 роки) [2]

Графік на рисунку 1.1 відображає динаміку виробництва велосипедів та автомобілів у період з 1950 по 2000 роки (дані представлені у мільйонах одиниць). На перший погляд можна помітити, що виробництво автомобілів показує стабільний та поступовий зростання протягом усього розглянутого періоду. Однак, цікавим є те, що велосипеди періодично виходять в лідери за

виробництвом, переважно у періоди криз та економічних нестабільностей. Це пов'язано з кращою доступністю та більш низькою вартістю велосипедів порівняно з автомобілями, що робить їх привабливими альтернативними засобами транспорту у не легких економічних умовах. Таким чином, хоча виробництво автомобілів продовжує зростати, велосипеди залишаються важливим елементом транспортної інфраструктури, особливо у періоди економічних труднощів.

У 2020 році сталося значне зростання популярності велосипедів, яке було викликане пандемією COVID-19. В статті, популярного британского журналу BBC, “The great bicycle boom of 2020” [3], розповідається, що в умовах локдаунів і обмежень, багато людей звернулися до велосипедів як альтернативного засобу транспорту та способу підтримки фізичної активності. Таким чином люди уникали громадського транспорту через страх заразитися. Закрилося багато спортивних залів, і звичайно були введені обмеження на різні види спорту, це змусило людей шукати нові способи залишатися активними.

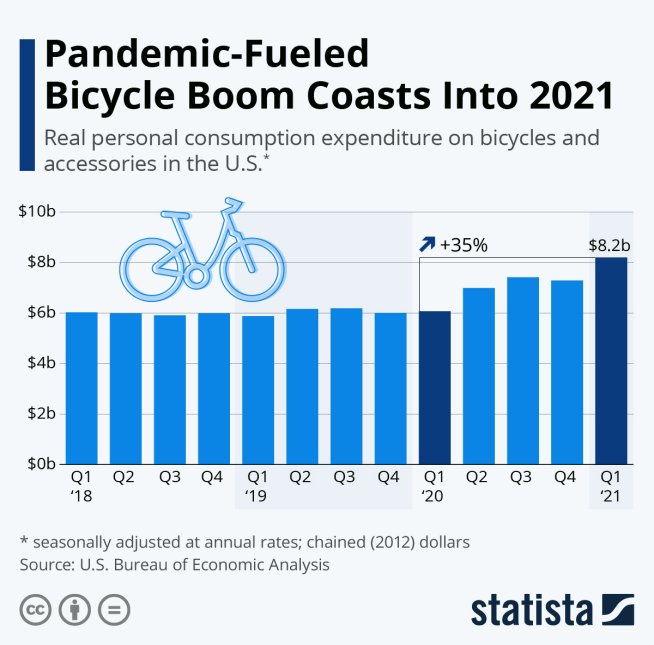


Рисунок. 1.2 – Графік, який демонструє велосипедний бум під час пандемії [4]

На рисунку 1.2, відображено реальні витрати на аксесуари для велосипедів у США за період з першого кварталу 2018 року до першого кварталу 2021 року. Можна бачити різкий ріст витрат, майже на 35%, починаючи з другого кварталу 2020 року.

Стаття також зазначає, що цей тренд може тривати й після пандемії, якщо міста продовжать інвестувати в велосипедну інфраструктуру та підтримувати безпечні умови для велосипедистів.

1.1.2 Важливість велосипеду у сучасному світі

Стаття з Environmental Health Perspectives [5] досліджує різні переваги для здоров'я та ризики, пов'язані з їздою на велосипеді.

Дослідження підкреслює, що користь для здоров'я від їзди на велосипеді значно переважає ризики. Регулярні поїздки на велосипеді можуть значно покращити стан серцево-судинної системи, зменшити ризик хронічних захворювань, наприклад, діабет 2 типу та деякі види раку, а також сприяти психічному благополуччю. Велосипедна активність допомагає контролювати вагу та знижує ризик ожиріння.

Варто зазначити, що велосипед є екологічно чистим видом транспорту, який зменшує викиди парникових газів і забруднення порівняно з автотранспортом. Це сприяє покращенню якості повітря та зменшенню проблем зі здоров'ям, пов'язаних із небезпечними викидами.

Хоча і існують такі ризики, як дорожньо-транспортні пригоди, дослідження підкреслює, що загальна користь для здоров'я перекриває ці ризики. Загалом у статті зроблено висновок, що просування велосипедного транспорту за допомогою кращої інфраструктури, заходів безпеки та може призвести до значних переваг для здоров'я нації.

1.1.3 Виклики та потреби велосипедистів

Велосипедисти стикаються з різноманітними викликами та проблемами під час своїх подороже. Серед них

- Забезпечення безпеки.
- Ефективне планування маршрутів
- Регулярне відстеження прогресу велосипедистів
- Комунікація з іншими велосипедистами по всьому світу

У сучасному світі велосипедисти потребують зручних та ефективних рішень для цих проблем.

На щастя, з появою цифрових технологій заняття велоспортом стало ще простішим. Сучасні GPS-навігатори, застосунки для трекінгу тренувань та соціальні мережі для велосипедистів допомагають ефективніше планувати маршрути, відстежувати прогрес та підтримувати зв’язок з іншими вело ентузіастами по всьому світу.

1.2 Способи покращення велоподорожей

1.2.1 Планування маршрутів

Однією з основних потреб велосипедистів є ефективне планування маршрутів. Це включає визначення найбільш оптимальних шляхів, врахування висот, стану доріг та інших факторів, які суттєво впливають на якість велоподорожей. Сучасні застосунки для велосипедистів пропонують розширені можливості для планування маршрутів, використовуючи GPS-навігацію та інтеграцію з картографічними сервісами.

Існує декілька форматів файлів для зберігання маршрутів: GPX, TCX та FIT [6].

GPX (GPS Exchange Format):

GPX є одним з найпоширеніших форматів файлів для обміну географічних даних, таких як точки маршруту та шляхи. Він може зберігати інформацію про координати, висоту, час, швидкість та інші параметри, які можуть бути відстеженні та збережені за допомогою GPS-пристроїв. GPX файли можуть бути використані в різних програмах та сервісах для побудови маршрутів, відстеження тренувань, обміну маршрутами між користувачами тощо.

Шаблон gpx файлу:

```
<gpx version="1.1" creator="Автор маршруту">
 <trk>
  <name>Мій маршрут</name>
  <trkseg>
   <trkpt lat="Довгота" lon="Широта">
    <ele>Висота</ele>
    <time>Час</time>
   </trkpt>
   <!-- Інші точки маршруту -->
  </trkseg>
 </trk>
</gpx>
```

TCX (Training Center XML):

TCX є розробкою компанії Garmin, і використовується для зберігання даних про тренування та змагання, отриманих з фітнес-пристроїв Garmin та інших сумісних пристроїв. Цей формат може містити інформацію про тривалість тренування, відстань, швидкість, пульс, каденс, висоту, показники ефективності тощо. TCX файли добре підходять для аналізу тренувань, інтеграції з фітнес-додатками та спортивними платформами, такими як Strava, Garmin Connect тощо.

Шаблон tcx файлу:

```
<TrainingCenterDatabase  xmlns ="
http://www.garmin.com/xmlschemas/TrainingCenterDatabase/v2">
  <Activities>
        <Activity Sport="Тип активності">
        <Lap StartTime="Час початку кола ">
        <TotalTimeSeconds>Загальний час</TotalTimeSeconds>
        <DistanceMeters>Загальна відстань</DistanceMeters>
        <!-- Інші параметри тренування -->
                </Lap>
                <Track>
                <Trackpoint>
                <Time>Час</Time>
                <Position>
                        <LatitudeDegrees>Довгота</LatitudeDegrees>
                        <LongitudeDegrees>Широта</LongitudeDegrees>
                </Position>
                <AltitudeMeters>Висота у метрах </AltitudeMeters>
                <DistanceMeters>Відстань від початку </DistanceMeters>
        <!-- Інші параметри точки маршруту -->
        </Trackpoint>
        <!-- Інші точки маршруту -->
        </Track>
        </Activity>
  </Activities>
</TrainingCenterDatabase>
```

FIT (Flexible and Interoperable Data Transfer):

FIT є також форматом файлів, розробленим Garmin, і використовується для зберігання даних про фізичну активність, отриманих з пристроїв Garmin. Він є більш компактним за розміром порівняно з іншими форматами, що робить його ідеальним для зберігання великої кількості даних про довготривалі тренування. FIT файли також підтримуються різними програмами та сервісами для аналізу та візуалізації даних про тренування.

Зазвичай для експорту і імпорту маршрутів використовується саме формат GPX, оскільки він підтримується більшістю GPS-пристроїв та застосунків. Формати TCX та FIT використовуються набагато рідше, головним чином у професійних спортивних середовищах та для більш глибокого аналізу тренувальних даних.

1.2.2 Відслідковування тренувань

Відслідковування тренувань є важливим елементом для велосипедистів, що дозволяє їм аналізувати свої досягнення, встановлювати нові цілі та покращувати свою фізичну форму. Сучасні додатки для велосипедистів пропонують широкий спектр функцій для відслідковування тренувань у реальному часі, зберігання та аналізу даних.

Основні функції відслідковування тренувань

- Відстеження маршруту в реальному часі. Додатки, використовуючи GPS, можуть зберігати точний маршрут поїздки, а також відстань, швидкість та висоту точок на маршруті. Однією з найголовніших функцій є позиціонування користувача на карті в реальному часі, що допомагає орієнтуватися під час поїздки.
- Навігація по маршруту. Користувачі мають можливість обирати підготовлені маршрути для тренувань, що значно спрощує процес планування поїздок. Підготовлені маршрути можуть бути створені самим користувачем або отримані від інших спортсменів через застосунок.
- Реєстрація даних про тренування. В ці дані входять час тренування, середня та максимальна швидкість, пройдена дистанція, дані про висоту, включаючи набір та скидання висоти, частота серцевих скорочень (якщо підключений пульсометр), каденс (оберти педалей

в хвилину, при наявності обладнання) та потужність у ваттах (при наявності паверметра).

- Підрахунок витрачених калорій. Ще одна важлива функція додатків, підрахунок витрачених калорій на основі даних про швидкість, відстань, вагу та зріст користувача.

1.2.3 Аналіз тренувань

Аналіз тренувань є ключовим аспектом для велосипедистів, які прагнуть покращити свою фізичну форму, результати, відстежувати прогрес і досягати нових цілей. Завдяки сучасним технологіям, додатки для велосипедистів пропонують широкий спектр інструментів для детального аналізу тренувальних даних.

Після завершення тренування користувачі можуть переглянути звіти по різноманітним метриками, у вигляді графіків та діаграм. Аналіз таких даних допомагає зрозуміти свій прогрес, та покращити результати у майбутньому.

Основні метрики для аналізу:

- Дистанція - загальна відстань, пройдена під час тренування.
- Час - загальний час тренування та час у русі.
- Середня швидкість - середня швидкість протягом усього тренування.
- Максимальна швидкість - найвища швидкість, досягнута під час тренування.
- Висота - набір висоти та зниження, профіль висоти маршруту.
- Серцевий ритм - дані про серцевий ритм, якщо використовується відповідний датчик.

- Спалені калорії - оцінка кількості спалених калорій під час тренування.

1.2.4 Експорт тренувань

Експорт тренувань є ще одним важливим аспектом для велосипедистів, які бажають аналізувати свої дані за допомогою різних платформ, наприклад, ділитися результатами з тренерами або іншими спортсменами, а також зберігати свої маршрути та тренування в зручному форматі. Сучасні додатки для велосипедистів підтримують експорт тренувань у різноманітних форматах файлів (такі формати як GPX, TCX та FIT були згадані вище), це дозволяє переносити дані між різними програмами.

Підтримка експорту, також дозволяє користувачам інтегрувати свої дані з такими платформами, як Strava, Garmin Connect та інші. Це надає додаткові можливості для аналізу і спільного використання та планування тренувань, роблячи тренування більш ефективним, корисним та мотивуючим.

1.3 Огляд та аналіз існуючих рішень

Під час аналізу існуючих рішень, було проаналізовано найпопулярніші застосунки у таких сервісах як Play Market (магазин застосунків під ОС Андроїд) та App Store (магазин застосунків під ОС IOS).

1.3.1 Strava

Strava [7]– це одна з найпопулярніших платформ для відстеження та аналізу спортивної активності, включаючи велосипедні тренування. Вона дозволяє користувачам записувати маршрути, аналізувати свої тренування та ділитися досягненнями з іншими спортсменами. Застосунок підтримує різні види спорту, але особливо популярна серед велосипедистів та бігунів. В якості інтерактивних карт сервіс використовує Mapbox та OpenStreetMap.

Основний функціонал застосунку є відстеження та запис активності, це можливо зробити за допомогою смартфону або імпортувати дані з інших GPS-пристроїв, таких як смарт браслети, смарт годинники, або велокомп'ютери. Ці дані включають відстань, час, швидкість, висоту, каденс, серцевий ритм тощо, в залежності від пристрою для трекінгу даних.

Застосунок дозволяє ділитися своїми тренуваннями, маршрутами та досягненнями з друзями та підписниками. Платформа підтримує функції лайків, коментарів та обговорень, що сприяє соціальній взаємодії між спортсменами по всьому світу.

Популярною функцією Strava, є сегменти, це такі відрізки на карті, де користувачі можуть змагатися, порівнюючи свої результати з результатами інших спортсменів. Також, платформа надає детальні статистичні дані та аналіз тренувань, включаючи графіки, карти та порівняння з попередніми тренуваннями, є можливість ставити цілі та відстежувати свій прогрес.

Що ж до маршрутів, ця функція є платною, користувачі можуть створювати власні маршрути або використовувати готові, які створили інші. Функція навігації допомагає слідувати заданому маршруту під час тренування.

Strava добре інтегрована з іншими фітнес-додатками, такими як TrainingPeaks, Garmin Connect, Zwift тощо. Експорт даних можливий у форматах GPX, TCX та FIT, що дозволяє легко переносити дані між різними системами.

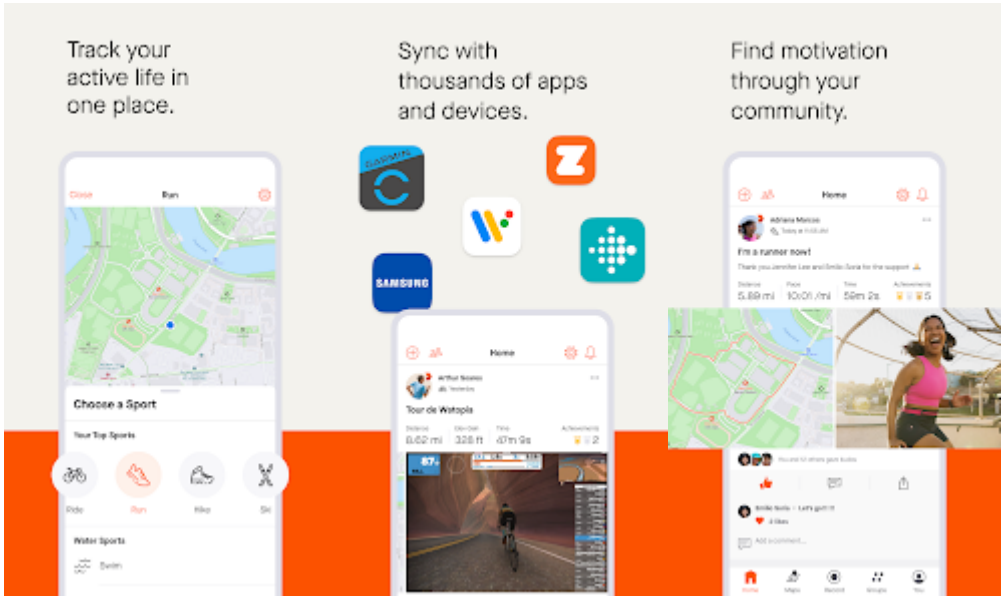


Рисунок. 1.3 – Інтерфейс застосунку Strava

Отже переваги Strava наступні:

- Широкий функціонал, що забезпечує великий набір інструментів для відстеження та аналізу тренувань, це чудово підходить для спортсменів різного рівня.
- Можливість ділитися досягненнями та змагатися з іншими спортсменами.
- Підтримка різноманітних фітнес-пристроїв дозволяє користувачам легко записувати тренування.
- Перегляд найпопулярніших маршрутів на картах, де спортсмени катаються найбільше, а де менше.

Але також є деякі недоліки:

- Деякі функції, такі як детальна аналітика та створення маршрутів, доступні лише для платних підписників Strava.
- Збільшена соціальна взаємодія може створювати ризики для конфіденційності даних.

1.3.2 Komoot

Komoot [8] – це система для створення та аналізу велосипедних і піших маршрутів. Вона дозволяє користувачам створювати докладні маршрути, з інформацією про тип та якість покриття, складність маршруту та інші важливі фактори, відстежувати свою активність та відкривати нові цікаві місця завдяки рекомендаціям спільноти. Застосунок особливо корисний для велосипедистів, туристів та бігунів, які хочуть досліджувати нові маршрути та місця. Застосунок використовує мапи OpenStreetMap та власні карти KomootMap

Komoot дозволяє переглядати маршрути, рекомендовані іншими членами спільноти, а також ділитися власними. Функція рекомендацій допомагає знайти найкращі маршрути для різних видів активностей. Платформа також пропонує функцію голосової навігації, яка допомагає користувачам слідувати заданому маршруту без необхідності постійно дивитися на екран.

Варто зазначити, що система підтримує інтеграцію з різними фітнес-додатками та платформами, такими як Strava, Garmin Connect та інші. Експорт даних можливий у форматах GPX та FIT, що дозволяє легко переносити дані між різними системами.

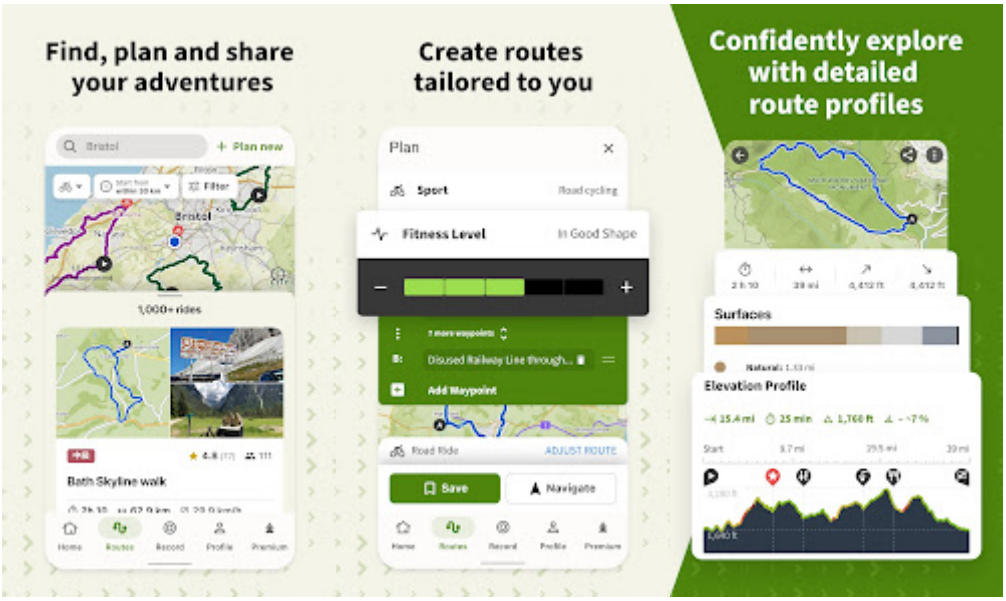


Рисунок. 1.4 – Інтерфейс застосунку Komoot

Переваги Komoot включають:

- Широкий функціонал планування, який пропонує інструменти для створення докладних маршрутів з урахуванням різних параметрів, що робить його ідеальним для планування складних поїздок.
- Функція голосової навігації забезпечує зручність та безпеку під час поїздок.
- Можливість отримувати рекомендації від інших користувачів та ділитися власними маршрутами сприяє відкриттю нових цікавих місць.

Однак, є і недоліки.

- Деякі функції, такі як завантаження офлайн карт та розширені функції планування, доступні лише для платних підписників Komoot.
- Порівняно з іншими платформами, Komoot має меншу кількість інтеграцій з іншими сервісами та пристроями.

1.3.3 TrainingPeaks

TrainingPeaks [9]— це популярна платформа, яка орієнтовано більше для професійних та полупрофесійних спортсменів, які використовують її для планування і аналізу тренувань. Вона підтримує імпорт та експорт файлів у різних форматах, таких як TCX, GPX та FIT, для інтеграції з різними пристроями та додатками.

Застосунок дозволяє користувачам створювати індивідуальні тренувальні плани з детальним описом кожної активності. Календарний інтерфейс полегшує планування та перегляд тренувань. Користувачі можуть імпортувати дані тренувань з різних пристроїв (велокомп'ютери, смарт-годинники, тощо) для відстеження своїх результаті, так як платформа підтримує імпорт файлів у різноманітних форматах.

Для детального аналізу даних активностей доступні різні інструменти, включаючи графіки та звіти. Користувачі можуть аналізувати метрики, такі як серцевий ритм, дистанція, потужність спортсмена, швидкість, каденс, тощо. Тренери можуть отримувати доступ до даних своїх атлетів та надавати зворотний зв'язок, а атлети можуть ділитися своїми тренувальними даними з тренерами для отримання індивідуальних рекомендацій. TrainingPeaks інтегрується з багатьма іншими популярними фітнес-платформами та пристроями, такими як Garmin, Zwift, Strava та інші.

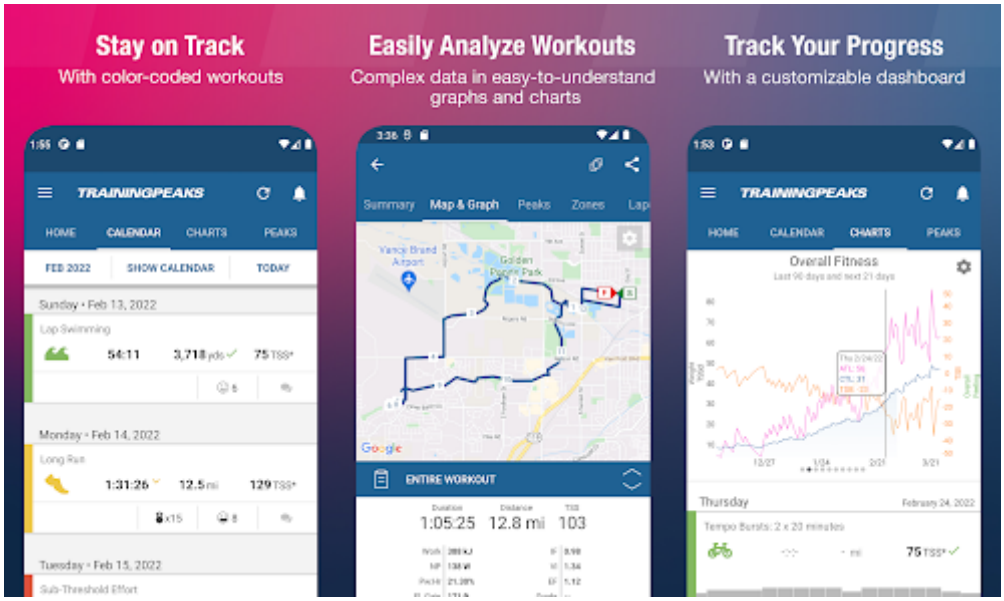


Рисунок. 1.5 – Інтерфейс застосунку TrainingPeaks

Основні переваги застосунку:

- Можливість більш глибокого аналізу даних допомагає користувачам покращувати свої результати.
- Експорт тренувань у Strava дозволяє користувачам ділитися своїми досягненнями з іншими спортсменами.

Недоліки:

- Відсутні інтерактивні мапи.
- Відсутня можливість створення маршруту та трекінгу тренування.

- Не зрозумілий інтерфейс для нових користувачів.

1.3.4 Suunto App

Suunto App [10] — це мобільний додаток, розроблений компанією Suunto, що надає користувачам можливість відстежувати та аналізувати свої спортивні активності. Додаток підтримує інтеграцію з різними пристроями Suunto та пропонує широкий набір функцій для спортсменів та фітнес-ентузіастів.

Відстеження активностей, є основною функцією Suunto App, вона дозволяє користувачам відстежувати різні види спортивних активностей, такі як біг, плавання, велоспорт, піші прогулянки тощо. Дані з пристроїв Suunto синхронізуються з додатком для подальшого аналізу.

Застосунок пропонує інструменти для детального аналізу тренувань, включаючи графіки та звіти. Користувачі можуть аналізувати такі метрики, як дистанція, тривалість, швидкість, серцевий ритм, калорії та інші показники. Користувачі можуть створювати маршрути на інтерактивній карті, а також імпортувати та експортувати маршрути у форматах GPX та FIT.

Інтеграція з іншими сервісами: Додаток інтегрується з популярними платформами, такими як Strava, TrainingPeaks, Apple Health та інші. Це дозволяє користувачам автоматично синхронізувати свої тренування та отримувати додатковий аналіз.

Suunto App розроблений в першу чергу для пристроїв Suunto, включаючи смарт-годинники та велокомп’ютери. Синхронізація відбувається через Bluetooth, що забезпечує зручне перенесення даних.

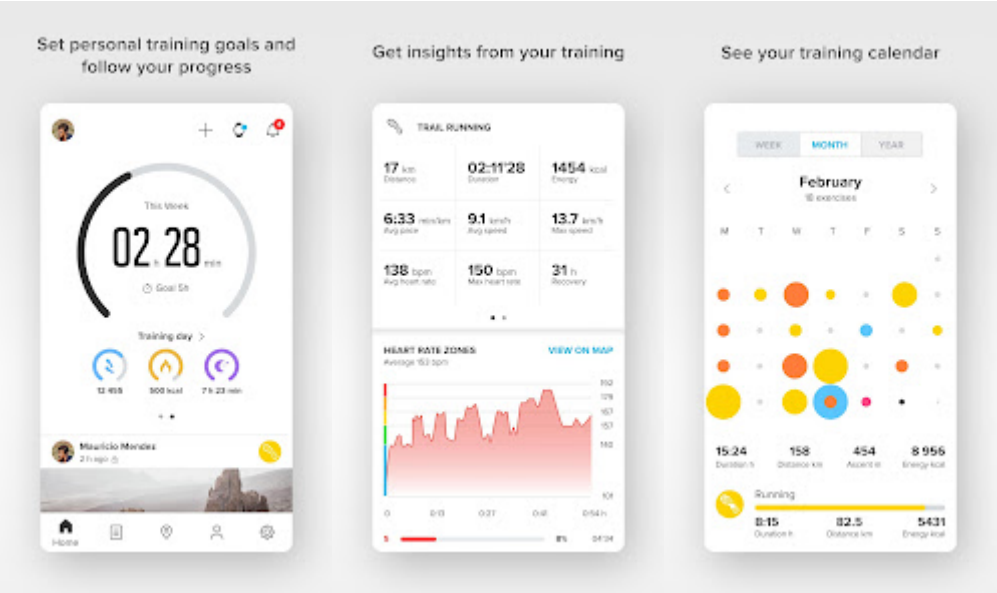


Рисунок. 1.6 – Інтерфейс застосунку Suunto App

Переваги:

- Suunto App надає широкий спектр функцій для відстеження та аналізу тренувань, що робить його корисним для спортсменів різного рівня підготовки.
- Інтерактивні карти, а саме можливість створення та збереження маршрутів на інтерактивних картах додає зручності для користувачів, які займаються активностями на відкритому повітрі.
- Повна підтримка фітнес-девайсів компанії Suunto.

Недоліки:

- Suunto App орієнтований лише на пристрої Suunto.
- Інтерфейс користувача хоча і є інтуїтивно зрозумілим, проте деякі користувачі можуть знайти його занадто складним через велику кількість функцій і налаштувань.

1.3.5 Wahoo Fitness App

Wahoo Fitness App [11]— це мобільний додаток, розроблений компанією Wahoo, як і попередній дозволяє відстежувати та аналізувати спортивні

активності. В першу чергу, він підтримує інтеграцію з пристроями Wahoo, що робить його незамінним інструментом для спортсменів, які використовують аксесуари даної фірми, такі як, смарт-годинники, велотренажери, паверметри, пристрої для вимірювання каденсу та інше.

Основні функції Wahoo Fitness App особливо не відрізняються від інших застосунків, це і відстеження активностей, яке дозволяє користувачам відстежувати різні види активностей, включаючи біг, велоспорт, плавання та інші види спорту. Дані з пристроїв Wahoo синхронізуються з додатком для подальшого аналізу. Застосунок також інтегрований з іншими популярними сервісами.

Додаток має цікаву функцію, яка називається "Live Track", вона дозволяє користувачам ділитися своїм місцезнаходженням у реальному часі з друзями або тренерами.

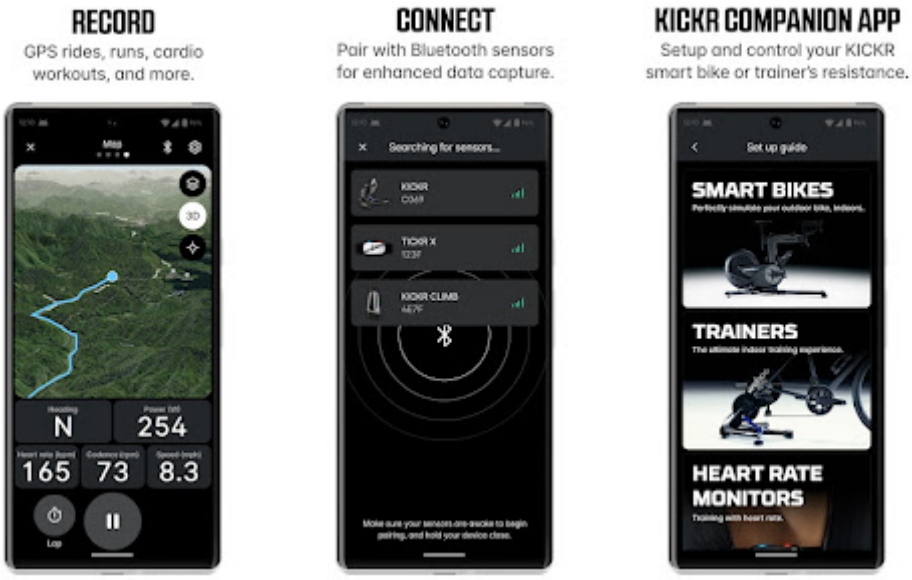


Рисунок. 1.7 – Інтерфейс застосунку Wahoo Fitness App

Переваги:

- Повна підтримка пристроїв Wahoo.

- Функція "Live Track" дозволяє користувачам ділитися своїм місцезнаходженням у реальному часі.
- Інтуїтивно зрозумілий інтерфейс, який полегшує використання додатку навіть новачкам.

Недоліки:

- Неможливість створення маршрутів
- Невеликий функціонал порівняно з іншими застосунками

1.3.6 Maplocs

Maplocs [12]— це спеціалізована програма для планування велосипедних маршрутів, розроблена для того, щоб зробити прокладання маршрутів простим і ефективним для велосипедистів. Додаток, розроблений компанією Pedal Rhythm, пропонує високо оптимізовані та точні маршрути для різних видів їзди на велосипеді, зокрема гірських, шосейних і звичайних велосипедів. Він використовує OpenRouteService для точності маршруту та пропонує детальні профілі висоти, що дозволяє користувачам бачити загальний набір і втрату висоти для запланованих маршрутів.

Користувачі можуть вибирати з різних типів карт, таких як Google Satellite, Hybrid або Terrain maps, які також відображають поточну інформацію про дорожній рух. Програма підтримує формат GPX, що дозволяє користувачам експортувати та ділитися своїми маршрутами. Він також дозволяє пряму інтеграцію з пристроями Garmin і Wahoo. Однією з особливостей є профілі висоти, які включають докладні дані про висоту та допомагають велосипедистам передбачити складність своїх маршрутів. У додатку відсутня реклама, але є платна підписка.

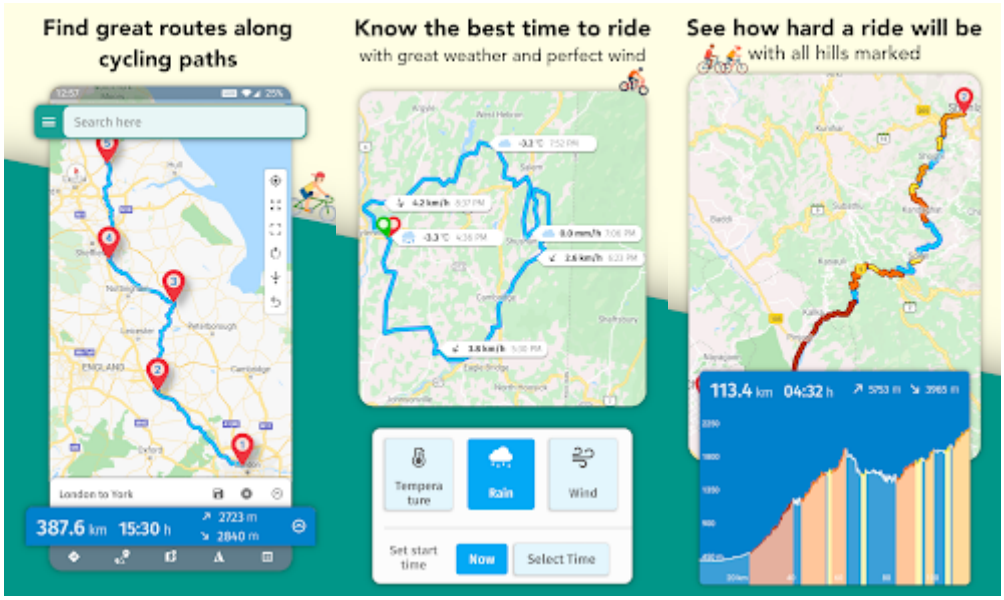


Рисунок. 1.8 – Інтерфейс застосунку Maplocs

Переваги:

- Побудова маршруту з докладними метриками
- Імпорт та експорт маршрутів
- Взаємодія з іншими фітнес-сервісами
- Відстеження погоди

Недоліки

- Щоб використовувати повний спектр функцій, треба купувати платну підписку
- Відсутня можливість відслідковування активності
- Доступний тільки під ОС Android.

1.3.7 WorkOutDoors

WorkOutDoors [13] — це мобільний додаток для відстеження тренувань, який зосереджується на детальних картах і навігації. Застосунок доступний тільки під операційну систему iOS, особливо добре інтегрований з Apple Watch, що робить його популярним серед користувачів цієї платформи.

WorkOutDoors пропонує детальні векторні карти, які можна завантажувати для використання офлайн, що є незамінним для тренувань у віддалених місцях без доступу до інтернету. Застосунок підтримує навігацію, що допомагає користувачам слідувати заздалегідь запланованим маршрутам або імпортованим трекам, та попереджає користувача, коли той сходить, або повертається на маршрут.

Застосунок повністю інтегрований з Apple Watch, надаючи користувачам можливість використовувати функціональність додатку прямо з зап'ястя. Це включає доступ до карт, метрик тренувань, таких як, відстань, час, швидкість, висота, серцевий ритм.

WorkOutDoors підтримує імпорт і експорт маршрутів у форматах GPX та TCX, що дозволяє користувачам обмінюватися маршрутами та використовувати їх на інших пристроях чи в інших додатках. Також доступний безпосередній імпорт тренування у застосунок Strava.

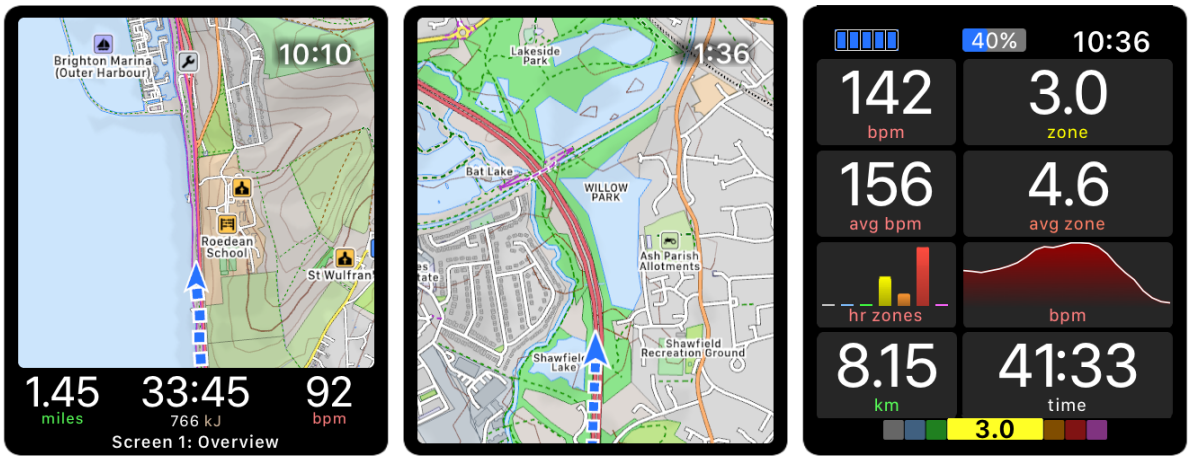


Рисунок. 1.9 – Інтерфейс застосунку WorkOutDoors на годинниках Apple Watch

Переваги:

- Детальні векторні карти, які можна використовувати офлайн.
- Повна інтеграція з Apple Watch.

- Гнучкість та налаштування інтерфейсу програми .
- Імпорт та експорт маршрутів.

Недоліки:

- Доступний тільки на iOS.
- Додаток є платним.
- Заплутаний інтерфейс, що може бути перепоною для новачків.

1.3.8 Порівняльний аналіз існуючих застосунків

У таблиці , описані усі переваги та недоліки існуючих застосунків, для велоподорожей, та порівняння з застосунком, який представлено в даній роботі.

Таблиця 1.1 – Переваги та недоліки існуючих застосунків

Критерій	Strava	Komoot	TrainingPeaks	Suunto App	Wahoo Fitness App	Maplocs	WorkOutDoors	Дана робота
Доступний під Android	✓	✓	✓	✓	✓	✓	✗	✓
Доступний під iOS	✓	✓	✓	✓	✓	✗	✓	✗
Картографічні сервіси	✓	✓	✗	✓	✓	✓	✓	✓
Побудова маршруту	✓	✓	✗	✗	✗	✓	✗	✓
Збереження та аналіз маршруту	✓	✓	✓	✓	✓	✗	✓	✓
Експорт маршруту у форматі GPX	✓	✓	✓	✓	✓	✓	✓	✓
Експорт маршруту у форматі TCX	✗	✗	✓	✗	✓	✗	✓	✓
Трекінг спортивної активності	✓	✓	✗	✓	✓	✗	✓	✓

Продовження таблиці 1.1

Критерій	Strava	Komoot	TrainingPeaks	Suunto App	Wahoo Fitness App	Maplocs	WorkOutDoors	Дана робота
Навігація по заданому маршруту	✓	✓	✗	✗	✓	✗	✓	✓
Збереження та експорт активності	✓	✓	✓	✓	✓	✓	✓	✓
Доступне API сервісу	✓	✓	✓	✓	✓	✗	✗	✗
Збереження даних у хмарі	✓	✓	✓	✓	✓	✓	✗	✓
Аналіз тренування	✓	✓	✓	✓	✓	✗	✓	✓

Як видно з таблиці 1.1 існуючи системи мають наступні спільні риси, а саме доступність на платформі Android, підтримку картографічних сервісів, збереження та аналіз маршрутів, а також можливість експорту маршрутів у форматі GPX.

Недоліки ж варіюються від відсутності підтримки певних платформ, до обмежень у функціональності, таких як побудова маршрутів, експорт у форматі TCX, відслідковування активності, навігації по маршруту та збереження даних у хмарі. Таким чином, актуальною є задача розробки нового застосунку для спортивних велосипедних подорожей.

ВИСНОВОК ДО РОЗДІЛУ 1

У розділі "Велоіндустрія: огляд і порівняння застосунків для велоспорту" було розглянуто важливі аспекти розвитку велосипедної промисловості, зокрема його бурхливий розвиток у 2020 році після пандемії COVID-19. Цей період відзначився збільшеним попитом на велосипеди та велосипедне обладнання, внаслідок чого індустрія пережила свого роду "бум". Було досліджено позитивний вплив велосипедного спорту на здоров'я людей та навколишнє середовище, зокрема його роль у зменшенні викидів шкідливих речовин та сприянні активному способу життя.

Було відмічено, що велосипедисти стикаються з різними викликами, включаючи безпеку на дорогах, нестабільність погодних умов, а також потребу у підтримці та мотивації під час тренувань. Ці виклики ставлять перед ними завдання, які можуть бути вирішені за допомогою сучасних технологій та застосунків для велоспорту.

Було розглянуто різноманітні формати, в яких можна зберігати дані про тренування, зокрема GPX, TCX та FIT. Ці формати дозволяють велосипедистам зберігати свої дані та аналізувати їх у відповідності до власних потреб та цілей. Описані їх переваги та недоліки, а також приведено у приклад шаблон цих файлів.

Під час аналізу застосунків для велоспорту було виявлено, що популярні додатки, такі як Strava, Komoot та інші, надають користувачам широкий спектр можливостей для планування маршрутів, відстеження тренувань та спілкування з іншими велосипедистами. Кожен з цих додатків має свої унікальні особливості та функції, що робить їх привабливими для різних категорій користувачів.

Отже, важко переоцінити важливість велосипедного спорту як засобу покращення здоров'я та збереження навколишнього середовища. Було проаналізовано переваги і недоліки різних вело застосунків, що дозволить у майбутньому, спираючись на цей досвід, розробити застосунок, який зможе перекреслити недоліки та покращити вже готові рішення на ринку.

РОЗДІЛ 2. ОГЛЯД ПІДХОДІВ ТА ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ ПЗ

2.1 Обґрунтування вибору операційної системи для розробки мобільного застосунку

Вибір операційної системи для розробки мобільного застосунку є важливим етапом, що впливає на його ефективність та подальшу підтримку. Правильний вибір ОС визначає цільову аудиторію, технічні можливості, витрати на розробку та підтримку, а також можливості для масштабування та оновлення застосунку.

За даними Statista [14], 87% компаній прагнуть запустити програму в Google Play Store, тоді як, тільки 60% респондентів розробляють програми для Apple App Store.

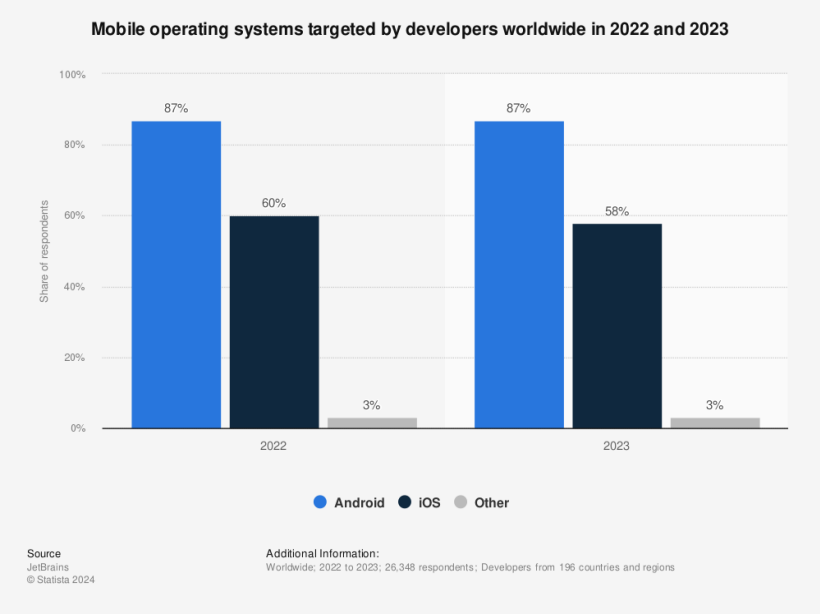


Рисунок 2.1 – Мобільні ОС, націлені на розробників у всьому світі у 2022 та 2023 роках

Обидві платформи мають свої переваги та недоліки.

Android:

- Android має найбільшу частку ринку мобільних пристроїв у світі, зокрема в країнах, що розвиваються.
- Широкий спектр пристроїв від різних виробників, включаючи годинники, планшети, телевізори та інше. Проте ця різноманітність, може ускладнити розробку та тестування, оскільки потрібно враховувати різні розміри екранів, продуктивність пристроїв та версії операційної системи.
- Можливість кастомізації та використання різних версій Android.
- Інтеграція з численними сервісами Google, такими як Google Play, Google Maps, Firebase тощо.

В свою чергу iOS:

- Більша частка ринку у розвинених країнах, через більш високу ціну на пристрої.
- Менше різноманіття пристроїв порівняно з Android, що спрощує розробку та тестування.
- Тісна інтеграція з продуктами та сервісами Apple.

Оцінивши усі переваги і недоліки операційних систем, вибір системи Android для розробки мобільного застосунку обґрунтований її високою ринковою часткою, широкими можливостями для кастомізації, потужними інструментами для розробки та інтеграції з екосистемою Google. Ці фактори роблять Android ідеальною платформою для реалізації проекту та забезпечують успішний запуск та використання застосунку на ринку.

2.2 Вибір інструмента для розробки ПЗ

Для створення застосунків для Android існує кілька популярних підходів і інструментів. Розберемо основні варіанти, включаючи їхні переваги та недоліки.

2.2.1 Java/Kotlin (Android Studio)

Android Studio [15] — це інтегроване середовище розробки для створення додатків на платформі Android, розроблене компанією Google. Воно забезпечує все необхідне для розробки, тестування та налагодження додатків під Android.

Для комфортної розробки доступний потужний редактор коду з підтримкою розумного автозаповнення, навігації по коду, рефакторингу, доповнення коду штучним інтелектом та багато іншого. Найчастіше використовується Java або Kotlin, які є найпопулярнішими мовами програмування для розробки Android-додатків. Java - традиційна, об'єктно-орієнтована мова з великою спільнотою та багатою екосистемою бібліотек. Kotlin - сучасна мова з лаконічним синтаксисом, безпекою від нульових значень та функціональними можливостями, що дозволяє прискорити розробку та зменшити кількість помилок. Обидві мови підтримуються платформою Android та можуть використовуватися разом у проектах, що забезпечує гнучкість у виборі та розвитку додатків.

Інструмент Android Emulator, для емуляції різних Android пристроїв та тестування додатків напряму через IDE, без необхідності використовувати фізичні пристрої.

Система збірки Gradle, яка автоматизує завдання збірки, тестування, упаковки та розгортання додатків. Також доступні інструменти для профілювання додатку, включаючи аналіз продуктивності, споживання пам’яті, використання процесора та мережевої активності.

Основною перевагою є те, що Android Studio — офіційне IDE від Google, що забезпечує найкращу інтеграцію з інструментами та сервісами Android, тому це є найкращий вибір для розробки нативних Android-додатків.

2.2.2 Flutter

Flutter [16] — це фреймворк для створення кросплатформенних мобільних додатків, розроблений Google. Він дозволяє розробляти додатки для Android, iOS, веб-застосунків та настільних платформ з використанням єдиного коду. Flutter використовує мову програмування Dart, також розроблену компанією Google.

Основною перевагою фреймворку є один код для додатків на Android, iOS, веб-застосунків та настільних платформ, що зменшує час і витрати на розробку. Доступні вбудовані віджети для створення інтерфейсів для різних платформ. Flutter використовує власний графічний рушій Skia для швидкого рендерингу інтерфейсу користувача, що дає можливість миттєвого перегляду змін у коді без повної перезавантаження додатку.

Здавалося б Flutter є чудовим рішенням для розробки під Android, проте є ряд недоліків, які можуть впливати на вибір цієї технології, а саме, розмір застосунку та обмежена підтримка нативних функцій. Через те що розробка на Flutter не нативна, розмір релізного застосунку буде більшим за порівняно з нативними застосунками, а також, іноді потрібно писати код для доступу до специфічних функцій для окремої платформи.

2.2.3 Kotlin Multiplatform Mobile (KMM)

Kotlin Multiplatform Mobile (KMM) [17] — це інструмент від JetBrains, який дозволяє розробляти кросплатформенні додатки з використанням однієї кодової бази на мові програмування Kotlin. KMM дозволяє спільно використовувати код між платформами Android та iOS, що допомагає скоротити час і витрати на розробку.

Основною особливостю KMM є спільний код, що дозволяє використовувати бізнес-логіку, моделі даних, обробку мережі, та інші частини коду між Android та iOS додатками. При зборці проекту код компілюється у нативний код, що забезпечує високу продуктивність на обох платформах, порівняно з іншими фреймворками. Варто зауважити, KMM також підтримує JVM, JavaScript та інші платформи, що робить його універсальним рішенням для кросплатформної розробки.

2.2.4 React Native

React Native [18] — це фреймворк для створення кросплатформенних мобільних застосунків, розроблений компанією Facebook. Основою React Native є бібліотека React, яка використовується для побудови користувацьких інтерфейсів.

Фреймворк, надає можливість розробникам створювати мобільні додатки для iOS та Android за допомогою одного коду. Це робить його надзвичайно вигідним рішенням, адже економить час і кошти на розробку. Завдяки функції "hot reloading" розробники можуть миттєво бачити результати внесених змін, що значно прискорює процес створення та тестування додатків.

2.2.5 Фінальне порівняння та підсумок, щодо вибору інструменту

Підіб’ємо підсумки, у таблиці 2.1 порівняємо розглянуті вище інструменти для розробки та оцінивши усі переваги і недоліки, оберемо найкраще рішення для даної роботи.

Таблиця 2.1 – Порівняння технологій для розробки ПЗ

	Java/Kotlin	Flutter	Kotlin Multiplatform Mobile (KMM)	React Native
Мова програмування	Java, Kotlin	Dart	Kotlin	JavaScript
Кросплатформеність	Тільки Android	Android, iOS, веб, настільні платформи	Android, iOS	Android, iOS
Підтримка нативного коду	Повна	Обмежена	Повна	Можлива, через нативні модулі
Продуктивність	Висока, нативна	Висока	Висока, нативна	Середня, може поступатися нативним додаткам
Інструменти розробки	Android Studio	Android Studio, Visual Studio Code	Android Studio, IntelliJ IDEA	Visual Studio Code, IntelliJ IDEA
Спільнота та підтримка	Велика, багаторічна	Швидко зростає, активна	Швидко зростає, активна	Велика, активна
Документація та ресурси	Багато ресурсів та прикладів	Добра документація, багато прикладів	Розвивається, зростає кількість ресурсів	Велика кількість ресурсів та прикладів
Типізація	Статична, строгий контроль типів	Статична	Статична	Динамічна
Можливості нативного доступу	Повний доступ до всіх нативних API	Потребує написання платформо-залежного коду	Повний доступ через спільний і нативний код	Можливий через нативні модулі

Отже, для створення мобільного додатка для велосипедних подорожей було розглянуто кілька варіантів інструментів та технологій, зокрема Flutter, Kotlin Multiplatform Mobile (KMM) та React Native. Після детального аналізу переваг і недоліків кожного з цих варіантів, було прийнято рішення використовувати Java/Kotlin в середовищі розробки Android Studio.

Вибір Java/Kotlin з Android Studio для розробки мобільного застосунку був обґрунтований прагненням забезпечити високу продуктивність, повний доступ до нативних можливостей платформи та використання потужних інструментів розробки. Комбінація цих факторів дозволяє створити надійний, швидкий та функціональний додаток, який відповідає всім вимогам та очікуванням користувачів.

2.3 Архітектура застосунку

Clean Architecture - це підхід до архітектури програмного забезпечення, який надає структуровану та масштабовану основу для розробки застосунків. Його мета - забезпечити легкість тестування, підтримки та розширення функціональності у майбутньому. Clean Architecture розділяє додаток на окремі шари, що дозволяє чітко розділити обов'язки кожного з компонентів. Основними шарами в Clean Architecture є Data, Domain та Presentation [19].

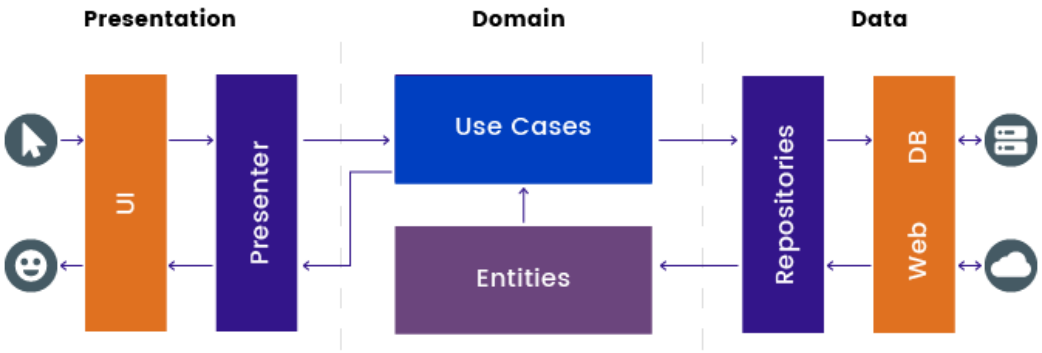


Рисунок 2.2 – Схема базової архітектури застосунку [20]

2.3.1 Data Layer

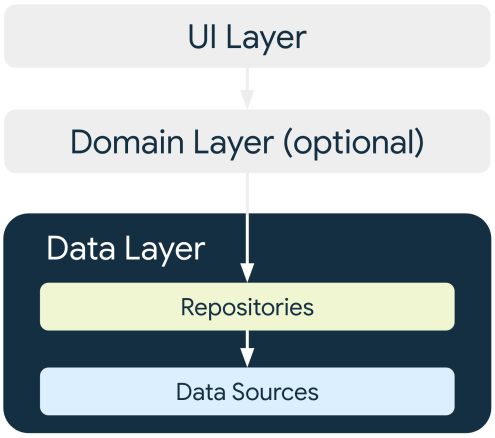


Рисунок 2.3 – Роль data layer в архітектурі програми.

Data Layer відповідає за доступ до даних із зовнішніх джерел, таких як бази даних, API, або файлові системи. Він є найнижчим шаром в архітектурі та повинен бути повністю ізольованим від деталей реалізації інших шарів. Основні компоненти Data Layer включають:

- Repositories - репозиторії служать посередниками між Data Layer і Domain Layer. Вони відповідають за отримання та збереження даних з різних джерел. Репозиторії інкапсулюють деталі реалізації доступу до даних, забезпечуючи чистий інтерфейс для взаємодії з ними.
- Data Sources - джерела даних можуть бути локальними, наприклад, внутрішні сховища, кеш і тд, або віддаленими REST API, хмарні БД. Вони забезпечують фактичний доступ до даних, використовуючи відповідні технології.
- Models - моделі даних (DTOs) представляють структуру даних, які зберігаються і передаються між різними джерелами даних.

2.3.2 Domain Layer

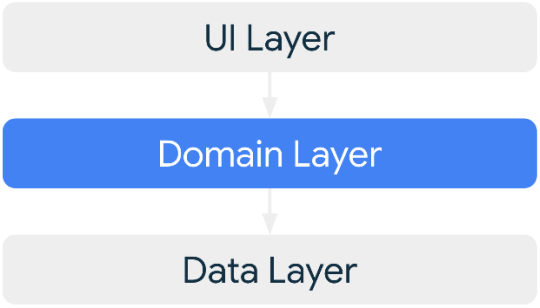


Рисунок 2.4 – Роль domain layer в архітектурі програми.

Domain Layer містить бізнес-логіку та правила, які визначають, як дані можуть бути створені, збереженні та змінені. Domain Layer повинен бути повністю ізольованим від інших шарів і не повинен залежати від будь-яких деталей реалізації. Основні компоненти Domain Layer включають:

- **Entities** - сутності представляють основні об'єкти бізнес-логіки. Вони містять атрибути та поведінку, пов'язані з цими об'єктами.
- **Use Cases (Interactors)** - юзкейси реалізують конкретні бізнес-вимоги та правила. Вони визначають взаємодію між сутностями та координують виконання бізнес-логіки.
- **Repositories Interfaces** - нтерфейси репозиторіїв визначають контракт, який Data Layer повинен реалізувати для надання даних. Це дозволяє Domain Layer бути незалежним від конкретних реалізацій джерел даних.

2.3.3 Presentation Layer

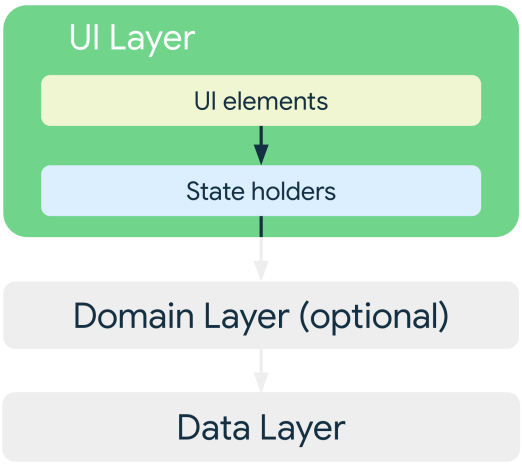


Рисунок 2.4 – Роль presentation layer в архітектурі програми.

Presentation Layer відповідає за взаємодію з кінцевим користувачем і відображення даних. Він містить логіку відображення та обробки подій користувача. Як правило в presentation шарі використовують паттерн MVVM

MVVM (Model-View-ViewModel) — це такий шаблон архітектури ПО, який розділяє бізнес-логіку та інтерфейс користувача. MVVM сприяє тестуванню, модульності та масштабованості коду.

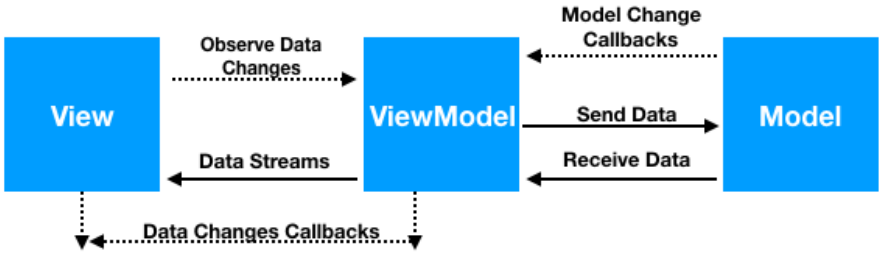


Рисунок 2.5 – Схема MVVM паттерну [21]

Компоненти MVVM:

Model (Модель) - Відповідає за управління даними додатка. Включає бізнес-логіку, отримання даних з мережі або бази даних. Не містить жодних залежностей від UI і може використовуватися в різних частинах додатка (data layer у нашому випадку).

View (Представлення) - Відповідає за відображення даних та взаємодію з користувачем. Включає Activity чи Fragment. Важливо зазначити, що View, не містить бізнес-логіки, а вся логіка обробки подій передається у ViewModel.

ViewModel (Модель представлення) - Зв’язує Model і View. Зберігає стан View та обробляє логіку інтерфейсу. Відповідає за підготовку даних для відображення у View і реагує на події з View. Використовує реактивні підходи для зв’язку з View.

Тоді взаємодія між компонентами виходить наступна - Model взаємодіє з джерелами даних і надає їх ViewModel. В свою чергу, ViewModel, отримуючи дані від Model, обробляє їх та надає View, для выдобрежнення даних користувачу. Також, реагує на дії користувача, що надходять від View (наприклад, натискання на кнопку збереження). В свою чергу, View спостерігає за змінами у ViewModel через LiveData або інші реактивні підходи та оновлює UI.

Переваги Clean Architecture

1. Чітке розділення шарів дозволяє легко додавати нові функції та компоненти без впливу на інші частини системи.
2. Ізольовані компоненти легко тестувати незалежно один від одного. Можливість використання моків для заміни залежностей полегшує написання юніт-тестів.
3. Завдяки чіткій структурі і розділенню відповідальностей, підтримка та модифікація коду стає значно простішою.

4. Можливість змінювати або оновлювати окремі компоненти без порушення роботи всієї системи робить код гнучким та адаптивним до змін.

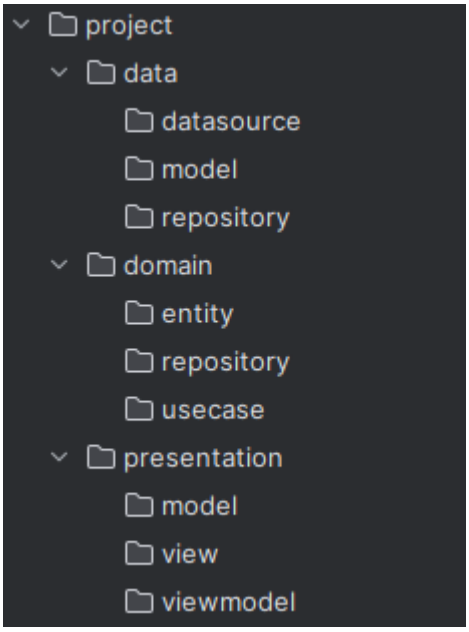


Рисунок 2.6 - Приклад структури проекту з Clean architecture

Clean Architecture надає чітку та структуровану основу для розробки мобільних додатків, забезпечуючи їхню масштабованість, тестованість та підтримуваність.

2.4 Вибір мови програмування

Проаналізуємо основні мови програмування для розробки мобільних додатків під платформу Android [22].

Почнемо з Java, ця мова програмування має багаторічну історію та стабільність, що робить її дуже популярною серед розробників. Її синтаксис довший і більш формальний, і хоча Java відома проблемами з NullPointerException, вона пропонує широку підтримку та велику кількість бібліотек. Java повністю сумісна з існуючими Android-бібліотеками та є

традиційною об'єктно-орієнтованою мовою з високою продуктивністю. Проте для досягнення тієї ж функціональності, як у Kotlin, потрібно більше коду.

Kotlin, який є відносно новою мовою та тільки набирає популярність. Ця мова забезпечує вбудовану null-безпеку, зменшуючи кількість помилок, і підтримується Google з повною інтеграцією в Android Studio. Також, Kotlin повністю сумісна з Java-бібліотеками і підтримує як об'єктно-орієнтоване, так і функціональне програмування. Завдяки лямбда-виразам та іншим можливостям, дозволяє писати менше коду та отримувати той самий результат. Варто зазначити, що Kotlin має вбудовану підтримку корутин для асинхронного програмування та розширювальні функції, які дозволяють додавати нову функціональність до існуючих класів. Як і Java, вона пропонує автоматичне керування пам'яттю та високу продуктивність.

Після детального аналізу переваг і недоліків Java та Kotlin, вибір був зроблений на користь Kotlin. Ця мова програмування має більш лаконічний і виразний характер, зменшуючи кількість необхідного коду та підвищуючи його читабельність, що сприяє зменшенню кількості помилок.

Null-безпека в Kotlin є вбудованою, що запобігає виникненню NullPointerException на етапі компіляції, підвищуючи надійність та стабільність додатка. Також великим плюсом Kotlin є підтримка корутин, що спрощує написання асинхронного коду та дозволяє ефективно керувати потоками виконання.

Швидке зростання спільноти розробників Kotlin забезпечує доступ до різноманітних ресурсів, бібліотек та прикладів коду, підтримуваних компаніями JetBrains та Google.

2.5 API та бібліотеки

Наступний розділ містить опис бібліотек та API, які будуть використовуватися у проекті.

2.5.1 Jetpack compose

У сучасній розробці Android, Jetpack Compose виступає як прогресивний фреймворк для побудови інтерфейсів користувача [23].

Jetpack Compose – це бібліотека Kotlin для створення декларативних інтерфейсів користувача в Android-додатках. На відміну від традиційного імперативного підходу, де UI будується поетапно за допомогою коду, Compose використовує декларативний стиль. Це означає, що розробник описуєте бажаний стан інтерфейсу, а Compose сам опікується його візуалізацією та оновленнями.

Ключовим компонентом Jetpack Compose є стейт, він відіграє фундаментальну роль у створенні динамічних інтерфейсів користувача. Ця концепція лежить в основі декларативного підходу Compose, дозволяючи описувати те, як інтерфейс має виглядати та реагувати на зміни даних, не заглиблюючись у складну логіку.

Стейт – це будь-яка інформація, яка може змінюватися з часом і впливати на візуальне представлення користувачу. Це можуть бути значення змінних, дані з мережі, або будь-які інші динамічно оновлювані дані. Jetpack Compose використовує паттерн спостерігач (Observer) для автоматичного оновлення інтерфейсу користувача при зміні стану даних. Коли стан змінюється, Compose автоматично перерисовує відповідний компонент інтерфейсу з новими даними. Це дозволяє програмістам описувати UI як функцію стану, яка автоматично оновлюється при зміні стану даних, не потребуючи додаткових кроків для оновлення інтерфейсу.

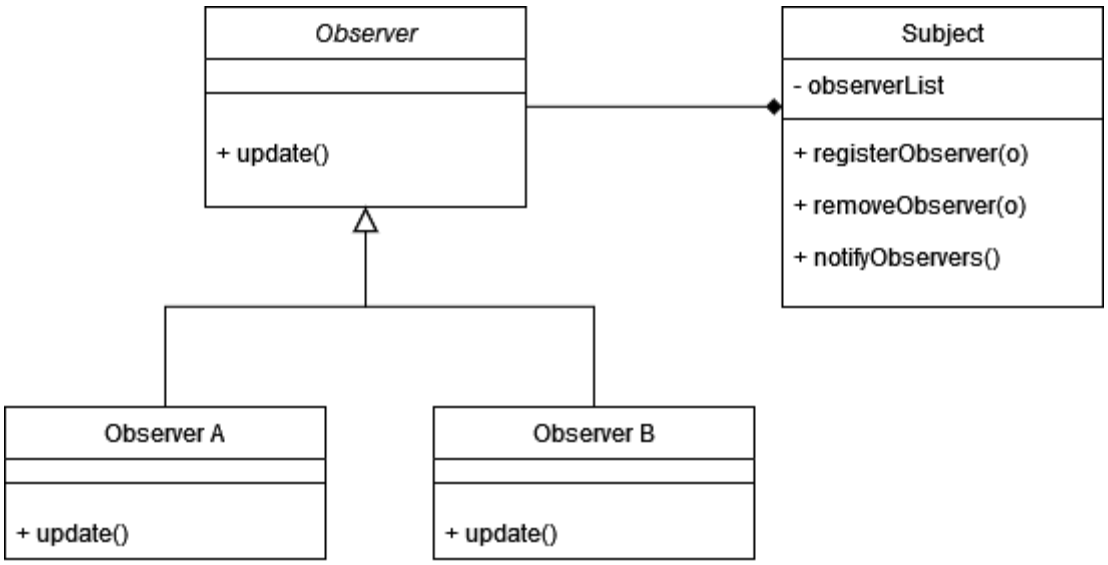


Рисунок 2.7 – Діаграма патерну спостерігач

Переваги використання Jetpack Compose:

- Дозволяє писати менше коду для досягнення того ж результату порівняно з імперативним підходом. Це робить код більш читабельним та легшим для підтримки.
- Пропонує інструменти для попереднього перегляду UI прямо в Android Studio, що дозволяє швидше перевіряти зміни та пришвидшує розробку.
- Тісно інтегрується з іншими бібліотеками Jetpack, такими як ViewModel та Navigation, що спрощує розробку складних додатків.
- Повністю підтримує Material Design, що дозволяє легко створювати красиві та сучасні інтерфейси користувача.

2.5.2 Впровадження залежностей та Dagger-Hilt

Впровадження залежностей – це фундаментальний принцип програмного забезпечення, який полягає у наданні об'єктам їхніх залежностей замість того, щоб змушувати їх створювати ці залежності самостійно. Це робить код більш модульним, тестованим та гнучким.

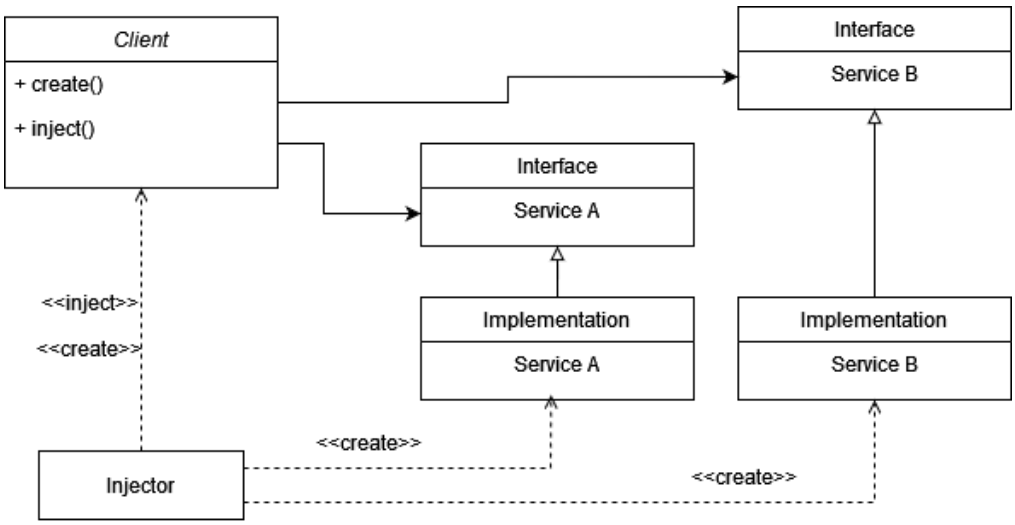


Рисунок 2.7 – Діаграма патерну впровадження залежностей

Dagger-Hilt – це бібліотека, побудована на основі Dagger 2, яка спрощує впровадження залежностей в Android-додатках. Вона тісно інтегрується з Jetpack Compose та іншими бібліотеками Jetpack, роблячи процес впровадження залежностей ще більш зручним.

Переваги використання впровадження залежностей з Dagger-Hilt:

- Код стає більш чітко структурованим та поділеним на незалежні модулі, що полегшує його розуміння та обслуговування.
- Залежності можна легко замінити для написання тестових сценаріїв, роблячи тестування коду більш ефективним.
- Зміни в залежностях можна легко вносити без необхідності модифікувати код, який їх використовує.
- Dagger-Hilt автоматично генерує код, необхідний для ін'єкції залежностей, що економить час та зусилля для розробників.

Як правило в проекті створюється окрема директорія, яка називається di, де і зберігаються усі модулі для залежностей.

2.5.3 Firebase: Authentication, Firestore, Analytics та Storage

Firebase - це платформа для розробки мобільних та веб-додатків [24], яка надає набір інструментів для прискорення розробки додатків. Ключові сервіси Firebase, які використовуються у проекті, це - Authentication, Firestore, Analytics та Storage. Почнемо з Firebase Authentication.

Firebase Authentication - це сервіс для аутентифікації користувачів, що підтримує різні методи входу в додаток. Він спрощує процес додавання автентифікації до додатку, забезпечуючи готові рішення для входу у систему. У проекті доступно два види аутентифікації. Перший, класичний метод, де користувачі реєструються за допомогою своєї електронної пошти та пароля. Другий, вхід через обліковий запис Google.

Важливо відзначити, що даний сервіс є цілком безпечним, по ряду причин. Всі дані, що передаються між клієнтом та сервером Firebase, шифруються за допомогою протоколу SSL/TLS. Це забезпечує захист від перехоплення та модифікації даних під час їх передачі. Також, впроваджений захист від атак типу "людина посередині" (MITM), для цього використовуються SSL/TLS сертифікати, для запобігання атакам, де зловмисник може спробувати перехопити комунікацію між клієнтом та сервером. Для автентифікації через соціальні мережі використовуються безпечні OAuth токени, що знижує ризик компрометації облікових даних користувачів.

Firebase Authentication використовує безпечні методи зберігання облікових даних користувачів. Паролі зберігаються у вигляді хешів, що забезпечує їх захист у випадку витоку бази даних.

Firestore - це гнучка, NoSql хмарна база даних, яка дозволяє зберігати і синхронізувати дані між користувачами та пристроями в режимі реального часу. Firestore надає такі можливості як:

- Синхронізація даних у режимі реального часу між клієнтами.
- Дані зберігаються у вигляді колекцій та документів, що дозволяє створювати структуровані та ієрархічні схеми даних.
- Підтримка складних запитів з фільтрацією, сортуванням та обмеженням результатів.
- Забезпечення офлайн-підтримки, завдяки якій, додатки можуть працювати з кешованими даними та синхронізувати їх, коли з'єднання з інтернетом відновлюється.

І нарешті, сервіс Firebase Storage, він відрізняється від Firestore, розглянутого раніше, тим, що надає можливість зберігати великі користувацькі файли, наприклад зображення, відео, аудіо та інші документи.

Firebase Storage полегшує зберігання та керування файлами, дозволяючи швидко інтегрувати функціональність завантаження та скачування файлів у додаток.

Загалом, використання Firebase забезпечує потужний набір інструментів для аутентифікації користувачів, зберігання та синхронізації даних, аналізу поведінки користувачів та управління файлами. Кожен з цих сервісів інтегрується з іншими компонентами Firebase, що спрощує розробку, забезпечує високу якість додатку та покращує взаємодію користувачів з ним.

2.5.4 Retrofit

Retrofit – це популярна бібліотека для Android, яка спрощує виконання мережевих запитів HTTP та обробку JSON-відповідей. Її зручний інтерфейс та потужні можливості роблять її цінним інструментом, для роботи з API.

Retrofit підтримує різні типи HTTP-запитів, такі як GET, POST, PUT, DELETE, HEAD та PATCH. Також, бібліотека надає зручний функціонал для обробки помилок під час запитів. Може автоматично перетворює JSON-

відповіді в об'єкти, що економить час та зусилля розробника. Ще однією особливістю є кешування відповіді, щоб покращити продуктивність та зменшити трафік даних.

2.5.5 Mapbox

Mapbox - це платформа для роботи з картам та геоданими, яка надає інструменти для створення інтерактивних карт та вбудованих навігаційних рішень у додатки. Вона включає в себе SDK (набір інструментів для розробки програмного забезпечення) для різних платформ, включаючи Android, а також можливість роботи з веб-картами.

Основною причиною вибору Mapbox, стала їх політика для розробників, яка дозволяє безкоштовно використовувати їхні сервіси, для 25000 активних користувачів, що є актуальним для стартапів.

Turf – це потужна JavaScript бібліотека для роботи з геопросторовими даними, яка пропонує широкий спектр функцій для аналізу та обробки геометрій. Її мобільний порт, Turf for Android, можливість інтегрувати деякі можливості у застосунок. Великою перевагою Turf є повна сумісність з Mapbox SDK, тому ця бібліотека стала незамінною у проекті. Такі функції, як обчислення геометрій точок, ліній та полігонів, або аналіз відстаней між точками активно використовується у коді програми.

Directions API від Mapbox - це сервіс, який надає можливість розрахунку оптимального маршруту між двома або більше точками на мапі. Він враховує різні фактори, такі як дорожні умови, трафік, типи доріг тощо, для надання найкращого маршруту. В даній роботі нам потрібно будувати маршрути для велосипедистів, тому даний сервіс нам підходить найкраще, тому що, можна налаштувати маршрут спеціально для велосипеда, тип самим уникаючи небезпечних місць, наприклад, завантажених доріг або трас.

ВИСНОВОК ДО РОЗДІЛУ 2

У цьому розділі було розглянуто підходи та технології, що застосовуються при розробці програмного забезпечення для мобільних пристроїв. Основними етапами дослідження стали вибір операційної системи, інструментів розробки, архітектурного підходу та використання сторонніх API та бібліотек.

Обґрунтування вибору операційної системи зосереджувалося на визначенні платформи, яка найкраще відповідає вимогам цільової аудиторії та забезпечує необхідний функціонал для реалізації проекту. В результаті було прийнято рішення зосередитися на платформі Android через її велику ринкову частку та гнучкі можливості розробки.

При виборі інструментів для розробки ПЗ було розглянуто декілька популярних варіантів, включаючи Java/Kotlin (Android Studio), Flutter, Kotlin Multiplatform Mobile (KMM) та React Native. Кожен з інструментів мав свої переваги та недоліки, але фінальний вибір було зроблено на користь Java/Kotlin у середовищі Android Studio, завдяки їхній високій продуктивності, надійності та підтримці найновіших функцій Android.

Розробка архітектури застосунку базувалася на використанні архітектурного паттерну MVVM (Model-View-ViewModel), що забезпечує чітке розділення відповідальностей і полегшує тестування та підтримку коду.

Вибір мови програмування також був важливим аспектом, де порівнювалися Java та Kotlin. Зрештою, було обрано Kotlin через його сучасні можливості, зручний синтаксис і повну сумісність з екосистемою Java.

Огляд API та бібліотек включав Jetpack Compose для побудови інтерфейсу користувача, Dagger-Hilt для впровадження залежностей, Firebase

для автентифікації, зберігання даних і аналітики, а також Retrofit для роботи з мережею та Mapbox для інтеграції картографічних сервісів.

Таким чином, проведений аналіз та обґрунтування вибору інструментів, технологій та підходів забезпечили надійну основу для подальшої розробки мобільного застосунку, що відповідає сучасним стандартам якості та функціональності.

РОЗДІЛ 3. ДЕТАЛІ РЕАЛІЗАЦІЇ СИСТЕМИ

Згідно з дослідженими матеріалами та обраного способу реалізації системи, можемо перейти до фази реалізації програмного продукту. Щоб створити зручний і багатофункціональний застосунок для спортивних велосипедних подорожей на платформі Android, необхідно описати основні компоненти розробки, функціональні блоки програми та схему їх взаємодії. Це включатиме інтеграцію мап, побудову маршрутів, відстеження тренувань і їхній аналіз, а також можливість експорту даних у різні формати та синхронізацію з сервісами, такими як Strava.

3.1 Реалізація базових функцій застосунку

У цьому підрозділі буде докладно описано реалізацію базових функцій застосунку, а також розглянуто класи, моделі та бізнес-логіку. Кожна з цих складових є важливою для забезпечення функціональності та ефективності роботи програми. Особлива увага приділяється інтеграції мап, побудові маршрутів, відстеженню тренувань, аналізу даних та їхньому експорту. Наведені приклади коду та пояснення допоможуть зрозуміти, як реалізовані ті чи інші функції, які технології та бібліотеки було використано.

Важливо зазначити, що далі буде використовуватися термін юзкейс. Юзкейс (Use case) - це специфічні сценарії, які описують взаємодію користувача із системою для досягнення певної мети. У нашому випадку, ми розглянемо ключові юзкейси для застосунку для велоподорожей на платформі Android.

3.1.1 Авторизація користувача

Авторизація користувача є критично важливою частиною будь-якого сучасного додатка. Вона забезпечує безпеку, зручність та персоналізацію для користувачів, а також дозволяє додатку зберігати та захищати дані.

Використовуючи Firebase Auth, можна легко інтегрувати різні методи авторизації, такі як авторизація через Google аккаунт, електронну пошту та пароль, а також керування скиданням пароля.

Авторизація гарантує, що тільки авторизовані користувачі мають доступ до ресурсів та функцій додатка. Це захищає особисті дані користувачів від несанкціонованого доступу. В нашому випадку це геодані користувача, та інша персональна інформація, така як зріст, вага, вік та інше.

Авторизовані користувачі можуть отримувати доступ до власного профілю, історії маршрутів і тренувань, налаштувань та інших персональних даних. Використання авторизації через Google дозволяє спростити процес входу для користувачів, зменшуючи кількість необхідних кроків для реєстрації та входу.

У таблиці 3.1, описані основні юзкейси, які використовуються для реєстрації та авторизації користувача у системі.

Таблиця 3.1 Опис юзкейсів для авторизації

Юзкейс	Опис	Параметри	Результат
AuthWithGoogle	Авторизація через обліковий запис Google	googleToken: String	Результат авторизації (успішно чи помилка)
GetCurrentAuthState	Отримання поточного стану авторизації користувача	-	Стан авторизації
GetUserId	Отримання ідентифікатора поточного користувача	-	Ідентифікатор користувача (userId) або null

Продовження таблиці 3.1

Юзкейс	Опис	Параметри	Результат
SignInWith EmailAndPassword	Авторизація за допомогою електронної пошти та пароля	email: String, password: String	Результат авторизації (успішно чи помилка)
SignUp WithEmailAndPassword	Реєстрація нового користувача за допомогою електронної пошти та пароля	email: String, password: String	Результат реєстрації (успішно чи помилка)
SignOut	Вихід користувача із системи	-	Результат виходу (успішно чи ні)

На рисунку 3.1 показано алгоритм авторизації користувача у системі. Алгоритм починається з точки входу в програму. Спершу перевіряється, чи користувач вже авторизований. Якщо так, то відбувається навігація на головний екран. Якщо ні, користувача перекидають на екран авторизації. Тут він може обрати авторизацію через пошту або через Google акаунт. При авторизації через пошту користувач вводить свої дані. Якщо дані коректні, відбувається запит на сервер. Якщо дані некоректні або користувач забув пароль, він отримує повідомлення про помилку або переходить на екран відновлення пароля. При авторизації через Google акаунт користувач також вводить дані про свій аккаунт, і у випадку коректності інформації отримує доступ до програми. Якщо користувач не зареєстрований, він може перейти на екран реєстрації, ввести свої дані, і якщо дані коректні, відбувається запит на сервер для створення нового облікового запису. Успішно створений користувач також отримує доступ до головного екрану. У всіх випадках некоректності даних користувач отримує повідомлення про помилку.

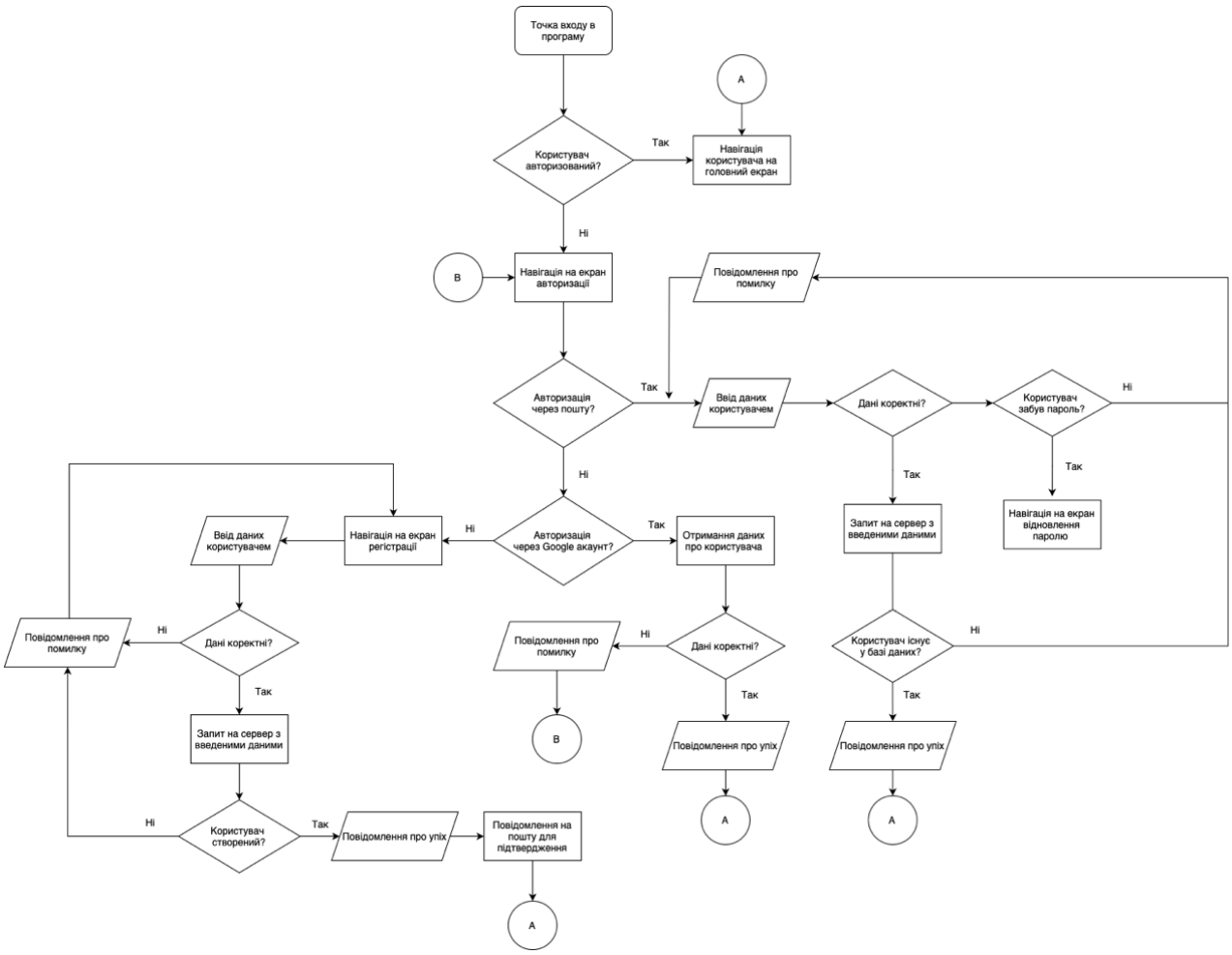


Рисунок 3.1 Алгоритм авторизації користувача у системі

3.1.2 Побудова маршрутів

Для побудови маршрутів у застосунку використано Mapbox SDK для інтеграції карт, а також Direction API для визначення оптимальних маршрутів між заданими точками. У цьому підрозділі розглянемо процес створення карт та побудови маршрутів за допомогою Mapbox SDK, а також наведемо приклади коду.

Mapbox SDK дозволяє легко інтегрувати карти в мобільні додатки [25] та надає інструменти для роботи з геолокаційними даними. Як було згадано у попередньому розділі, у проєкті використовується паттерн MVVM, це означає, що основна логіка побудови маршрутів, буде реалізована у View Model, а View буде лише спостерігати за стейтом, та реагувати на його зміни, наприклад,

відображати точки на карті, або малювати сам маршрут. На рисунку 3.2 показано архітектуру та взаємодію між класами у системі побудови маршрутів для велоподорожей.

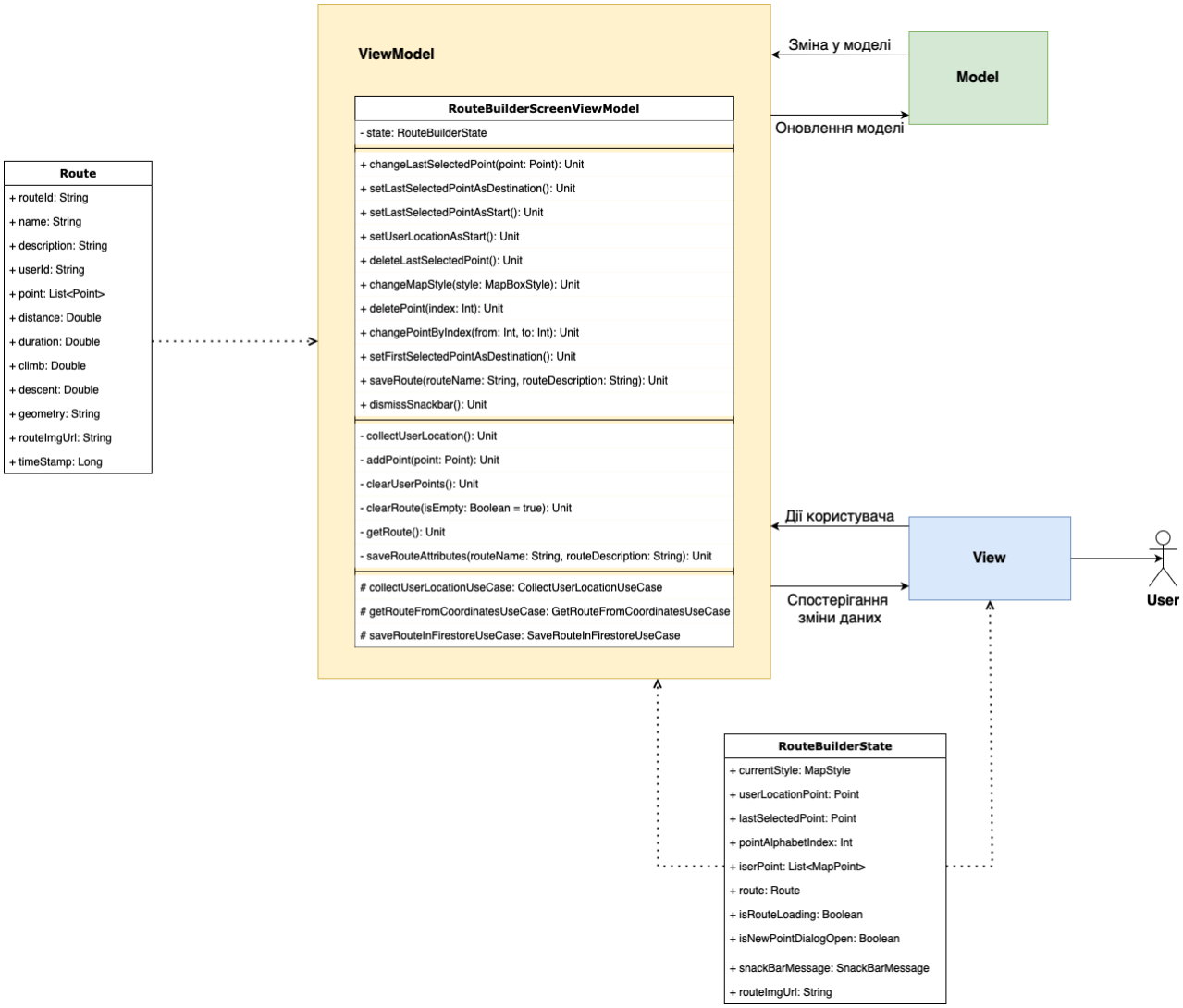


Рисунок 3.2 Діаграма взаємодії між класами, які відповідають за побудову маршруту

RouteBuilderScreenViewModel основний клас, який реалізовує інтерфейс ViewModel, він містить стан екрану побудови маршруту (RouteBuilderState) і забезпечує логіку взаємодії з іншими компонентами.

Таблиця 3.2 Основні публічні методи класу RouteBuilderScreenViewModel

Метод	Опис
changeLastSelectedPoint(point: Point): Unit	Змінює останню обрану точку та відкриває діалогове вікно для нової точки.
setLastSelectedPointAsDestination(): Unit	Встановлює останню обрану точку як кінцеву точку маршруту.
setLastSelectedPointAsStart(): Unit	Встановлює останню обрану точку як початкову точку маршруту, очищаючи існуючий маршрут та точки.
setUserLocationAsStart(): Unit	Встановлює місцезнаходження користувача як початкову точку.
deleteLastSelectedPoint(): Unit	Видаляє останню обрану.
changeMapStyle(style: MapBoxStyle): Unit	Змінює поточний стиль карти.
deletePoint(index: Int): Unit	Видаляє точку з маршруту за індексом.
changePointByIndex(from: Int, to: Int): Unit	Змінює позицію точки в списку користувацьких точок.
setFirstSelectedPointAsDestination(): Unit	Встановлює першу обрану користувачем точку як кінцеву.
saveRoute(routeName: String, routeDescription: String): Unit	Зберігає поточний маршрут з заданими іменем та описом, оновлюючи Firestore.
dismissSnackbar(): Unit	Приховує будь-яке поточне повідомлення snackbar.

Публічні методи використовуються для виклику із View. Можна побачити, що методи не повертають ніяких значень, оскільки ці методи тільки змінюють стан (стейт), на який вже реагує View для зміни інтерфейсу застосунку. Це означає, що всі зміни в інтерфейсі застосунку відбуваються автоматично у відповідь на зміну стану, без необхідності безпосередньо викликати методи для оновлення інтерфейсу. Такий підхід дозволяє зменшити

кількість коду, який відповідає за взаємодію між бізнес-логікою та інтерфейсом, і зробити код більш чистим і зрозумілим.

Таблиця 3.3 Основні приватні методи класу RouteBuilderScreenViewModel

collectUserLocation(): Unit	Збирає дані про місцезнаходження користувача. (Викликається при створені класу)
addPoint(point: Point): Unit	Додає точку до списку користувацьких точок та оновлює маршрут.
clearUserPoints(): Unit	Очищає всі користувацькі точки.
clearRoute(isEmpty: Boolean = true): Unit	Очищає поточний маршрут, за необхідності скидаючи індекс точки.
getRoute(): Unit	Отримує маршрут на основі поточних користувацьких точок.
saveRouteAttributes(routeName: String, routeDescription: String): Unit	Зберігає додаткові атрибути до поточного маршруту перед збереженням.

View, або іншими словами екран, який демонстує користувачу інтерфейс. Він взаємодіє з RouteBuilderScreenViewModel для відображення поточного стану екрану та змін у model.

Model – представляє собою Data шар, який передає дані до ViewModel з віддаленого або локального джерела, а також оновлює їх, якщо про це сповіщає ViewModel.

Для побудови маршрутів використовується 3 юзкейси, давайте опишемо їх.

Юзкейс 1: Збір поточної локації користувача (CollectUserLocationUseCase)

Ціль: Отримати поточну локацію користувача для початку побудови маршруту.

Передумови:

- Користувач надав дозвіл на використання локації.
- Локаційні сервіси пристрою ввімкнені.

Опис:

- Користувач відкриває застосунок.
- Система автоматично запитує поточну локацію користувача.
- Якщо локація успішно отримана, вона відображається на карті.

Альтернативні ситуації:

- Якщо локаційні сервіси вимкнені, система відображає повідомлення з проханням увімкнути їх.
- Якщо дозвіл на використання локації не надано, система відображає повідомлення з проханням надати дозвіл.

Результат: Поточна локація користувача відображається на карті.

Юзкейс 2: Отримання маршруту по координатам (GetRouteFromCoordinatesUseCase)

Ціль: Отримати маршрут у вигляді закодованих точок, для подальшої побудови маршруту на карті.

Передумови: Користувач додав принаймні дві точки на карті.

Опис:

- Користувач натискає кнопку для побудови маршруту.
- Система відправляє запит до Direction API з координатами точок.
- Система отримує маршрут від API і відображає його на карті.

Альтернативні ситуації:

- Якщо запит до API не вдається, система відображає повідомлення про помилку.

Результат: Маршрут між точками відображається на карті.

Далі представлена реалізація отримання маршруту по координатам

```
override suspend fun getRoute(coordinates: List<Point>): Response<Route?> {
    return suspendCoroutine { continuation ->
        // Get retrofit client with custom parameters
        val client = MapboxDirections.builder() MapboxDirections.Builder!
            // Mapbox access token
            .accessToken(publicToken) MapboxDirections.Builder
            .routeOptions(
                RouteOptions.builder() RouteOptions.Builder!
                    // Route response format
                    .geometries(DirectionsCriteria.GEOMETRY_POLYLINE) RouteOptions.Builder
                    // User coordinates
                    .coordinates(coordinates.toDirectionsString())
                    // Set cycling profile
                    .profile(DirectionsCriteria.PROFILE_CYCLING)
                    // Get response in full overview
                    .overview(DirectionsCriteria.OVERVIEW_FULL)
                    .build()
            ).build()
        // Execute request
        client?.enqueueCall(object : retrofit2.Callback<DirectionsResponse> {
            override fun onResponse(
                call: Call<DirectionsResponse>,
                response: retrofit2.Response<DirectionsResponse>,
            ) {
                // Check response
                if (response.body() == null || ((response.body()?.routes()?.size ?: 0) < 1)) {
                    continuation.resume(Failure("Response is empty"))
                    return
                }
                // Get route
                if (response.isSuccessful) {...}

            }
            // on failure
            override fun onFailure(call: Call<DirectionsResponse>, t: Throwable) {...}
        })
    }
}
```

Рисунок 3.3 Реалізація отримання маршруту по координатам

Цей код визначає функцію *getRoute,* яка викликає асинхронний запит до API Mapbox Directions для отримання маршруту між заданими координатами. Функція використовує корутини і повертає об'єкт *Response<Route?>*. Спочатку створюється клієнт *MapboxDirections* з використанням *publicToken* та налаштувань маршруту *RouteOptions*. Налаштування маршруту включають геометрію полігону, координати, профіль (в нашому випадку, велосипедний) і огляд маршруту. Клієнт виконує запит до API, використовуючи метод *enqueueCall*, який виконує асинхронний запит. Якщо відповідь порожня або не містить жодного маршруту, повертається *Failure* з повідомленням *"Response is empty"*. Якщо запит успішний (*isSuccessful*), повертається перший маршрут з відповіді. Створюється об'єкт Route з отриманими даними (відстань, тривалість, координати, геометрія і тд) та повертається у вигляді *Success*. Якщо запит не вдається, повертається *Failure* з повідомленням про помилку. Функція використовує корутину (*suspendCoroutine*) для відновлення потоку виконання після завершення асинхронного запиту, що забезпечує зручний спосіб роботи з асинхронними запитами у Kotlin.

Юзкейс 3: Збереження маршруту

Ціль: Зберегти побудований маршрут у БД.

Передумови:

- Користувач побудував маршрут.

Опис:

- Користувач натискає кнопку для збереження маршруту.
- Система відкриває діалогове вікно для введення назви та опису маршруту.
- Користувач вводить назву та опис маршруту і підтверджує збереження.

- Система зберігає маршрут у БД Firestore.

Альтернативні ситуації:

- Користувач скасовує збереження маршруту.
- Через відсутність інтернету, або помилку при зберіганні, виводити користувачу повідомлення про невдачу

Результат: Маршрут збережено у Firestore і доступний для подальшого використання.

Важливо зазначити, що DirectionApi повертає маршрут без висоти точки над рівнем моря, а у даному випадку, це є важливо, щоб користувач міг аналізувати свій маршрут та корегувати його в залежності від значення набору висоти. Для того, щоб отримати ці дані, було прийнято рішення використовувати стороннє API, а саме https://api.open-meteo.com/v1/, яке дозволяє отримати висоту як однієї точки, так і списку з точок. Так як маршрут може мати багато точок, звернення до сервісу, виконується по декілька запитів з максимальною кількістю 100 точок (100 точок, це найбільша кількість яку може прийняти сервіс за один запит).

Для оптимізації звернень до сервісу, проходить перевірка, якщо точка вже має висоту, тоді знову не потрібно отримувати її.

3.1.3 Відслідковування велоподорожі

У застосунку для відслідковування активності користувача під час велоподорожі використовується спеціальний сервіс. Цей сервіс відповідає за збір даних про місцезнаходження користувача та інші параметри тренування в режимі реального часу. Його перевагою над звичайною ViewModel яку ми використовували до цього, є те, що сервіси це компонент Android, який виконує тривалі операції у фоновому режимі без взаємодії з користувачем. Сервіс корисний для вирішення завдань, які потребують тривалого виконання,

таких як відтворення музики, завантаження файлів, або, як у нашому випадку, відслідковування активності користувача під час велоподорожі. Сервіси можуть працювати навіть коли користувач переключається між додатками, що дозволяє забезпечити безперервність процесу.

На рисунку 3.4 зображено UML діаграму класу TrackingUserWorkoutService. Цей клас відповідає за відстеження тренувань користувача, збір і збереження даних про маршрути, швидкість, висоту і інші параметри тренування. Він виконує ключову роль у забезпеченні функціональності відстеження та аналізу спортивної активності в застосунку. Діаграма UML показує структуру класу, його атрибути і методи, а також взаємодію з іншими класами та компонентами системи, що допомагає краще зрозуміти його внутрішню логіку і взаємозв'язки.

Рисунок 3.4 UML діаграма сервісу відслідковування тренування

Давайте розберемо основні методи цього класу у таблиці 3.4

Таблиця 3.4 Методи класу TrackingUserWorkoutService

Метод	Опис
saveWorkout(name: String, onResult: (Throwable?) -> Unit)	Зберігає тренування у сховищі Firestore
startTracking()	Розпочинає відстеження тренування, починаючи відсідковування основних метрик
pauseTracking()	Призупиняє відстеження тренування.
resumeTracking()	Відновлює відстеження тренування.
stopTracking()	Зупиняє відстеження тренування та зберігає дані.
calculateBurnedCalories(durationInMinutes: Double, age: Int, weight: Int, isMale: Boolean,): Double	На основі даних про користувача підраховує калорії, які були витрачені під час тренування

Давайте більш детально розглянемо метод *calculateBurnedCalories*

Для того, аби підрахувати спалені калорії, використовується формула Міффліна-Сент-Джеора [26], для велосипедної активності. Формула розраховує кількість спалених калорій на основі введених параметрів, використовуючи різні формули для чоловіків (формула 3.1) та для жінок (3.2).

Для чоловіків:

$$C = (A * 0.2017 - W * 0.09036 + H * 0.6309 - 55.0969) * \frac{D}{4.184} \quad (3.1)$$

Для жінок:

$$C = (A * 0.074 - W * 0.05741 + H * 0.4472 - 20.4022) * \frac{D}{4.184} \quad (3.2)$$

де,

C — Кількість спалених калорій

D — тривалість активності в хвилинах.

A — Вік людини.

H — Середній пульс спортсмена під час тренування

W — Вага людини в кілограмах.

На рисунку 3.показана реалізація підрахунку калорій

```
// https://tourdevines.com.au/blog/how-many-calories-does-cycling-burn/
Nikita *
private fun calculateBurnedCalories(
    durationInMinutes: Double,
    age: Int,
    weight: Int,
    isMale: Boolean,
): Double {
    return if (isMale) {
        // Calculate calories for men
        abs( x: ((age * 0.2017) + (weight * 0.09036) - (avgHeartRate * 0.6309) - 55.0969) * ((durationInMinutes + 1) / 4.184))
    } else {
        // Calculate calories for women
        abs( x: ((age * 0.074) + (weight * 0.05741) - (avgHeartRate * 0.4472) - 20.4022) * ((durationInMinutes + 1) / 4.184))
    }
}
```

Рисунок 3.4 Реалізація методу для підрахунку спалених калорій

Для розрахунку дистанції між двома точками на карті використовується формула Гаверсинуса [27]. Вона враховує кривизну Землі та дає більш точні результати, ніж просто вимірювання відстані по прямій лінії.

$$d = 2R \arcsin\left(\sqrt{\sin^2\left(\frac{lat2-lat1}{2}\right) + \cos(lat1) * \cos(lat2) * \sin^2\left(\frac{lon2-lon1}{2}\right)}\right) \quad (3.3)$$

де

d - відстань між двома точками (в кілометрах)

R - радіус Землі (приблизно 6371 кілометрів)

lat1 - широта першої точки (в радіантах)

lat2 - широта другої точки (в радіантах)

lon1 - довгота першої точки (в радіантах)

lon2 - довгота другої точки (в радіантах)

У сервісі для трекінгу активності, використовується декілька юзкейсів:

Юзкейс 1: Збір поточної локації користувача (CollectUserLocationUseCase)

Цей юзкейс використовується і у попередньому випадку, для побудови маршрута, але при відслідковуванні активності для нас має значення ще декілька параметрів, а саме speed, altitude та timestamp:

- speed: Швидкість пересування користувача у метрах за секунду. Цей параметр буде використаний для оцінки інтенсивності фізичної активності та розрахунку спалених калорій.
- altitude: Висота над рівнем моря у метрах. Цей параметр буде корисним для врахування змін висоти під час активності, що також може вплинути на розрахунок спалених калорій.
- timestamp: Часова мітка у форматі UNIX time, яка вказує на точний час отримання даних локації. Це важливо для відстеження змін локації з часом та обчислення дистанції, пройденої за певний період.

Юзкейс 2: Збереження тренування

Ціль: Зберегти тренування у БД.

Передумови:

- Користувач почав тренування та пройшов принаймні 2 точки.

Опис:

- Користувач зупиняє тренування.
- Система відкриває діалогове вікно для підтвердження завершення тренування.
- Користувач вводить назву і підтверджує збереження.
- Система упиняє сервіс, та повідомляє про це користувачу.
- Система зберігає маршрут у БД Firestore.

Альтернативні ситуації:

- Користувач скасовує збереження тренування, і воно продовжується.
- Через відсутність інтернету, або помилку при зберіганні, виводити користувачу повідомлення про невдачу

Результат: Тренування збережено у Firestore і доступне для подальшого використання.

3.1.4 Експорт маршрутів та тренувань.

У цьому розділі розглянемо, як реалізується експорт маршрутів та тренувань у форматах GPX та TCX. Для цього використовується власний парсер, який генерує файли відповідних форматів, а також збереження даних у внутрішньому сховищі пристрою.

На рисунку 3.5, показана реалізація створення маршрутів та тренувань у форматі GPX. Варто зазначити, що файл тренування відрізняється від файлу маршрута, тільки додаванням часу, коли була пройдена кожна точка маршруту.

```
override suspend fun saveWorkoutInInternalStorage(workout: Workout): Boolean {
    // create gpx file in external storage
    val gpxFile = File(context.filesDir,  child: "${workout.name}.gpx")
    // create header
    val header =
        "<?xml version=\"1.0\" encoding=\"UTF-8\" standalone=\"no\" ?>" +
                "<gpx xmlns=\"http://www.topografix.com/GPX/1/1\" " +
                "creator=\"MapSource 6.15.5\" version=\"1.1\" " +
                "xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\"  " +
                "xsi:schemaLocation=\"http://www.topografix.com/GPX/1/1 " +
                "http://www.topografix.com/GPX/1/1/gpx.xsd\"><trk>\n"
    // create name
    val name = "<name>${workout.name}</name><trkseg>\n"
    // create segments
    val segments = StringBuilder()
    // add all points
    workout.points?.forEach { point ->
        segments.append(
            // add point latitude and longitude
            "<trkpt lat=\"" + point.point.latitude() +
                    "\" lon=\"" + point.point.longitude() + "\">" +
                    // add altitude
                    "\n<ele>" + point.point.altitude() + "</ele>\n" +
                    // add time in yyyy-MM-dd'T'HH:mm:ss.SSS'Z' format
                    "<time>" + point.timeStamp.convertTimestampToTime() + "</time>" +
                    "</trkpt>\n"
        )
    }
    // create footer
    val footer = "</trkseg></trk></gpx>"
    // return gpx file
    return gpxFile.createFile(header, name, segments.toString(), footer)
}
```

Рисунок 3.5 Реалізація методу для створення GPX файлу

Цей код визначає функцію saveWorkoutInInternalStorage, яка зберігає тренування у внутрішньому сховищі у форматі GPX. Спочатку створюється файл GPX у внутрішньому сховищі з ім'ям, що відповідає назві тренування. Потім створюється заголовок GPX-файлу, який містить метадані про файл. Додається назва тренування і відкривається трек-сегмент. Далі створюється список сегментів з точками тренування. Кожна точка додається до сегментів з зазначенням широти, довготи, висоти та часу у форматі yyyy-MM-dd'T'HH:mm:ss.SSS'Z'. Після цього створюється кінцівка GPX-файлу, яка закінчує файл. Далі усі дані записуються у файл, і функція повертає true, якщо файл успішно створений.

```
override suspend fun saveWorkoutInInternalStorage(workout: Workout): Boolean {
    // create tcx file in external storage
    val tcxFile = File(context.filesDir,  child: "${workout.name}.tcx")
    // create header
    val name = "<Activities>\n" +
            "<Activity Sport=\"Biking\">\n" +
            "<Id>${workout.timeStamp.convertTimestampToTime()}</Id>\n" +
            "<Lap StartTime=\"${workout.timeStamp.convertTimestampToTime()}\">\n" +
            "<TotalTimeSeconds>${workout.durationInSeconds}</TotalTimeSeconds>\n" +
            "<DistanceMeters>${workout.distance * 1000}</DistanceMeters>\n" +
            "<MaximumSpeed><Speed>${workout.maxSpeed}</Speed></MaximumSpeed>\n" +
            "<Calories>${workout.calories}</Calories>" +
            "<Track>\n"
    val segments = StringBuilder()
    // create segments
    workout.points?.forEach { point ->
        segments.append(
            "<Trackpoint>\n",
            "<Time>${point.timeStamp.convertTimestampToTime()}</Time>\n",
            "<Position>\n",
            "<LatitudeDegrees>${point.point.latitude()}</LatitudeDegrees>\n",
            "<LongitudeDegrees>${point.point.longitude()}</LongitudeDegrees>\n",
            "</Position>\n",
            "<AltitudeMeters>${point.point.altitude()}</AltitudeMeters>\n",
            "<DistanceMeters>${point.distanceFromStart}</DistanceMeters>\n",
            "<Extensions>\n",
            "<TPX xmlns=\"http://www.garmin.com/xmlschemas/ActivityExtension/v2\">\n",
            "<Speed>${point.speed}</Speed></TPX>\n",
            "</Extensions>\n",
            "</Trackpoint>"
        )
    }
    // create footer
    val footer = "</Track>\n" +
            "</Lap>\n" +
            "</Activity>\n" +
            "</Activities>\n" +
            "</TrainingCenterDatabase>"
    // write to file
    return tcxFile.createFile(header, name, segments.toString(), footer)
}
```

Рисунок 3.6 Реалізація методу для створення TCX файлу

На рисунку 3.6, показана реалізація подібно до попереднього методу, зберігає дані тренування у внутрішньому сховищі, але використовує формат TCX замість GPX. В обох методах створюється файл з ім'ям, що відповідає назві тренування, та додається заголовок з метаданими. Основна відмінність полягає

у форматі та структурі файлу. Метод для TCX включає детальнішу інформацію про тренування, таку як загальний час, дистанцію в метрах, максимальну швидкість та кількість спалених калорій. Кожна точка тренування включає час, координати, висоту, дистанцію від початку, швидкість та інші розширення, які записуються у вигляді трекпоінтів. Отже, метод для TCX надає більше інформації про тренування, роблячи його більш придатним для аналізу та відстеження прогресу у порівнянні з простішим форматом GPX.

Тепер використовуючи дані методи, ми можемо експортувати тренування до популярної платформи Strava. Щоб зробити це, треба авторизувати користувача у нашому застосунку, за допомогою протокола авторизації OAuth 2.0, який дозволяє додаткам отримувати обмежений доступ до користувацьких облікових записів на інших сервісах, без необхідності розкриття паролів. У процесі OAuth 2.0 користувачі авторизуються на сторонньому сервісі (в нашому випадку, Strava) і дають дозвіл додатку на використання деяких даних.

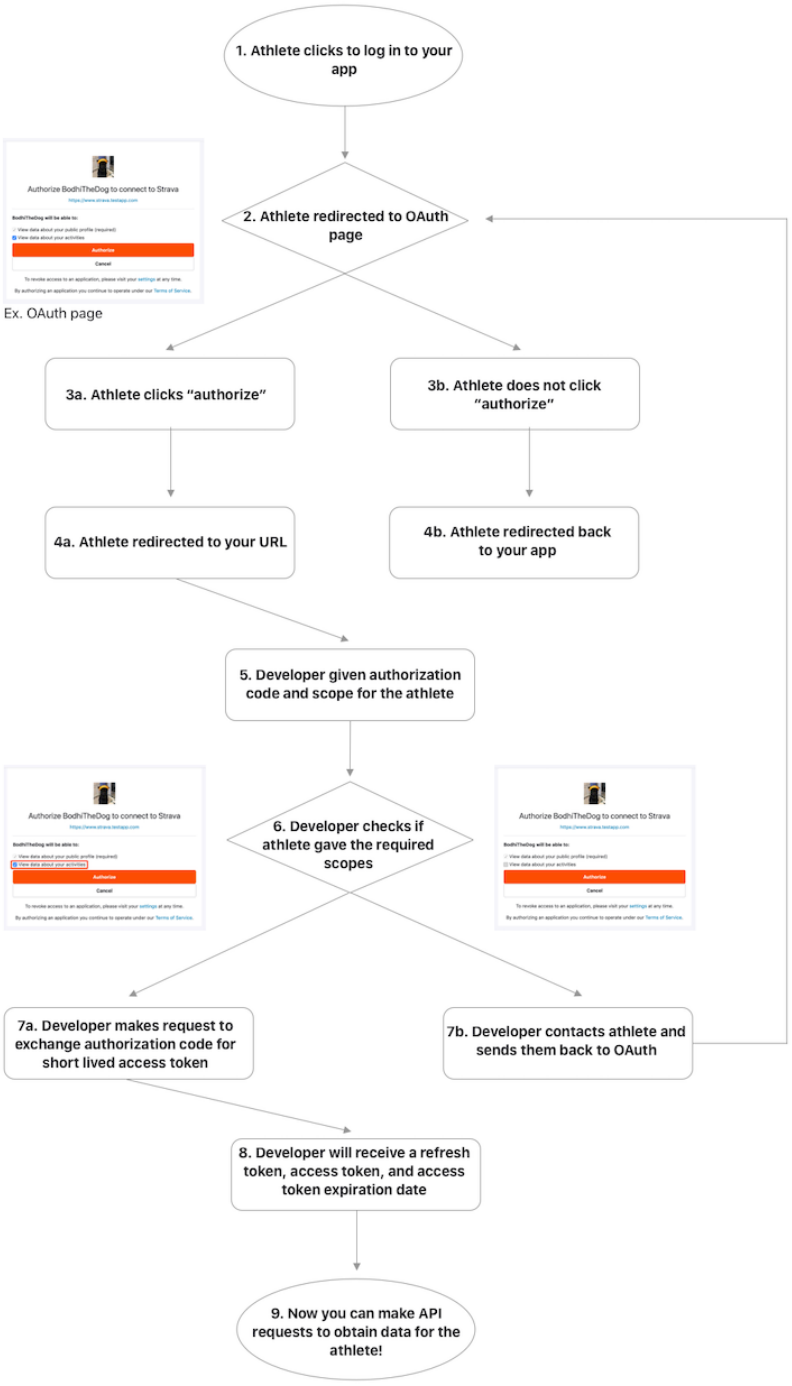


Рисунок 3.7 Авторизації через OAuth 2.0 у сервісі Strava [28]

На рисунку 3.7 показано процес авторизації через OAuth 2.0 для експорту тренувань до Strava. Спочатку користувач натискає кнопку для входу в додаток, після чого його перенаправляють на сторінку OAuth. Якщо користувач натискає "авторизувати", його перенаправляють назад до додатку з кодом

авторизації, якщо ж користувач відмовився, або сталась якась інша помилка, авторизація скасовується і відбувається навігація у додаток.

Далі програма отримує код авторизації та перевіряє, чи надано необхідні дозволи. Якщо так, відбувається запит для обміну коду авторизації на короткостроковий токен доступу, який потрібен для запитів до API сервісу Strava. Цей процес дозволяє безпечно експортувати тренування з додатку до Strava, забезпечуючи при цьому контроль за доступом до особистих даних користувачів.

3.2 Інтеграція бази даних Firestore

Важливим компонентом програми є зберігання маршрутів, тренувань та інших даних користувача у хмарній базі даних. Переваги БД Firestore, були розглянуті у попередньому розділі, тому у цьому, пропоную розібрати деталі роботи з нею.

3.2.1 Колекції та моделі документів.

Firestore є гнучкою, масштабованою базою даних для зберігання, синхронізації та запитів даних для мобільних застосунків. Вона використовує структуру зберігання на основі колекцій та документів. Кожен документ є набором пар "ключ-значення" і може містити вкладені піддокументи або колекції. Колекції та документи забезпечують потужний та зручний спосіб структурування даних. Документи в межах колекції можуть мати різні структури, хоча зазвичай вони мають схожі поля для спрощення запитів і обробки даних.

У цьому розділі буде розглянуто моделі документів для зберігання маршрутів, тренувань та користувачів у Firestore. Це включає детальне пояснення моделей даних, їхніх полів і використання для серіалізації та десеріалізації даних між додатком та Firestore. Моделі даних маршруту і тренування дозволяють ефективно зберігати інформацією про фізичну

активність користувачів, забезпечуючи швидкий доступ та зручні засоби для пошуку, сортування та фільтрації даних.

Почнемо з маршрутів, вони представленні в БД такою моделлю, яка описана у таблиці 3.5.

Таблиця 3.5 Опис моделі маршрута у БД Firestore

Назва	Тип	Опис
RouteFirestorePojo		
routeId	String	Унікальний ідентифікатор маршрута
userId	String	Унікальний ідентифікатор користувача, який створив маршрут
name	String	Назва маршруту
description	String	Опис маршруту
points	List<RoutePoint>	Список з точками маршруту
distance	Double	Відстань у метрах
duration	Double	Тривалість маршруту у секундах
climb	Double	Загальний підйом маршруту у метрах
descent	Double	Загальний спуск маршруту у метрах
geometry	String	Закодовані точки маршрута
routeImgUrl	String	Посилання на зображення маршруту
timeStamp	Long	Дата, коли був створенний маршрут
RoutePoint		
lng	Double	Довгота точки
lat	Double	Широта точки
altitude	Double	Висота над рівнем моря точки

Тепер у таблиці 3.6, опишемо модель тренування

Таблиця 3.6 Опис моделі тренування у БД Firestore

Назва	Тип	Опис
WorkoutFirestorePojo		
workoutId	String	Унікальний ідентифікатор тренування
userId	String	Унікальний ідентифікатор користувача, який створив тренування
name	String	Назва тренування
points	List Workout	Список з точками тренування
distance	Double	Відстань у метрах
duration	Double	Тривалість маршруту у секундах
climb	Double	Загальний підйом маршруту у метрах
descent	Double	Загальний спуск маршруту у метрах
avgSpeed	Double	Середня швидкість в кілометрах на годину
maxSpeed	Double	Максимальна швидкість під час тренування в кілометрах на годину
calories	Double	Спалені калорії під час тренування в кілокалоріях
routeImgUrl	String	Посилання на зображення тренування
timeStamp	Long	Час створення тренування у вигляді мітки часу
WorkoutFirestoreRoutePoint		
lng	Double	Довгота точки
lat	Double	Широта точки
altitude	Double	Висота над рівнем моря точки
dFromStart	Double	Дистанція від початку тренування у метрах
speed	Double	Швидкість користувача в кілометрах на годину
timeStamp	Long	Час у даній точці тренування

3.2.2 Фільтрація та сортування запитів.

Для забезпечення швидкого доступу та отримання потрібної інформації з колекцій документів у Firestore, використовуються індекси, фільтри та сортування.

Індекси автоматично створюються для окремих полів, але для складніших запитів, що включають кілька фільтрів або сортування за кількома полями, знадобитися створення складених індексів [29]. Наприклад, у застосунку можливе фільтрування по дистанції від найдовшого маршруту до найкоротшого та навпаки, тому для цієї ситуації треба створювати окремий індекс. Це дозволяє швидко знаходити та отримувати необхідні дані без сканування всієї колекції та як наслідок, до менших витрат на обслуговування БД.

Фільтри допомагають обмежити результати запиту [30], тільки тими документами, які відповідають певним умовам. Це зменшує обсяг переданих даних і підвищує ефективність запитів. У нашому випадку, можливо знаходити маршрут або тренування за ключовим словом, для цього використовується фільтри whereGreaterThan та whereLessThan.

Сортування ж дозволяє впорядковувати дані у певному порядку, що полегшує їх використання. У Firestore дані можна сортувати за будь-яким полем за допомогою методу orderBy.

Далі ми розглянемо, як використовувати індекси, фільтри та сортування для ефективної роботи з даними про тренування і маршрути. У нашому додатку існують різні типи фільтрів та параметри сортування, які ми застосовуємо під час запитів до Firestore. Розглянемо ці параметри та їх застосування на прикладі коду, рисунок 3.8, що демонструє, як отримати тренування з Firestore з використанням фільтрів та сортування.

```
override fun getWorkouts(
    searchWorkoutParams: SearchWorkoutParams,
    userId: String,
): Flow<Response<List<Workout>>> = callbackFlow { this: ProducerScope<Response<List<Workout>>>
    // create query for workouts
    var query = firestore
        // choose workout collection
        .collection(WORKOUT_COLLECTION) CollectionReference
        // get only user's workouts
        .whereEqualTo( field: "userId", userId) Query
        // sort by user filters
        .orderBy(searchWorkoutParams.sortWorkoutType.dbValue, if (searchWorkoutParams.sortDirection == SortDirection.ASC) {
            Direction.ASCENDING
        } else Direction.DESCENDING)
    // search by text
    if (searchWorkoutParams.searchText.isNotEmpty()) {
        query = query
            .whereGreaterThanOrEqualTo( field: "name", searchWorkoutParams.searchText)
            .whereLessThanOrEqualTo( field: "name", value: searchWorkoutParams.searchText + '\uf8ff')
    }
    // add snapshot listener
    val snapshotListener = query.addSnapshotListener { snapshot, e ->
        val response = if (snapshot != null) {
            // map firestore pojos to domain pojos
            val workouts = snapshot.toObjects(WorkoutFirestorePojo::class.java).map { it: WorkoutFirestorePojo!
                mapper.mapFirestorePojoToEntity(it)
            }
            Response.Success(workouts)
        } else {
            Response.Failure(e?.message ?: e.toString())
        }
        trySend(response).isSuccess
    }
    // close listener
    awaitClose {
        snapshotListener.remove()
    }
}
```

Рисунок 3.8 Отримання тренувань з використанням фільтрів та сортування

ВИСНОВОК ДО РОЗДІЛУ 3

У розділі було детально розглянуто реалізацію основних функцій застосунку та інтеграцію бази даних Firestore. Спочатку було описано механізми авторизації користувачів, що забезпечують безпечний і надійний доступ до системи. Далі були розглянуті методи побудови маршрутів і відслідковування велоподорожей, що є основою функціональністю застосунку. Особливу увагу було приділено експорту маршрутів та тренувань у різних форматах, таких як GPX та TCX, що дозволяє користувачам зберігати свої дані та використовувати їх у інших сервісах. Впроваджено безпечну та легку інтеграцію з сервісом Strava, що дозволяє на пряму експортувати тренування у сервіс.

Інтеграція з базою даних Firestore була докладно висвітлена через опис колекцій та моделей документів, що дозволяє ефективно зберігати і обробляти дані користувачів. Було також розглянуто важливість індексів, фільтрів та сортування для оптимізації запитів до бази даних. Це забезпечує швидкий доступ до необхідної інформації та підвищує продуктивність застосунку.

Таким чином, розділ охоплює всі ключові аспекти реалізації системи, починаючи від базових функцій і закінчуючи складними запитами до бази даних. Це гарна основа для подальшого розвитку і вдосконалення застосунку, забезпечуючи його ефективність і зручність для користувачів.

РОЗДІЛ 4. ОГЛЯД ТА ТЕСТУВАННЯ РОЗРОБЛЕННОГО ЗАСТОСУНКУ

4.1 Огляд інтерфейсу авторизації

Почнемо з огляду екрану входу та реєстрації, які представленні на рисунку 4.1. Якщо користувач тільки встановив застосунок, йому буде запропоновано зайти у свій обліковий запис, якщо ж облікового запису немає, тоді користувач може створити його.

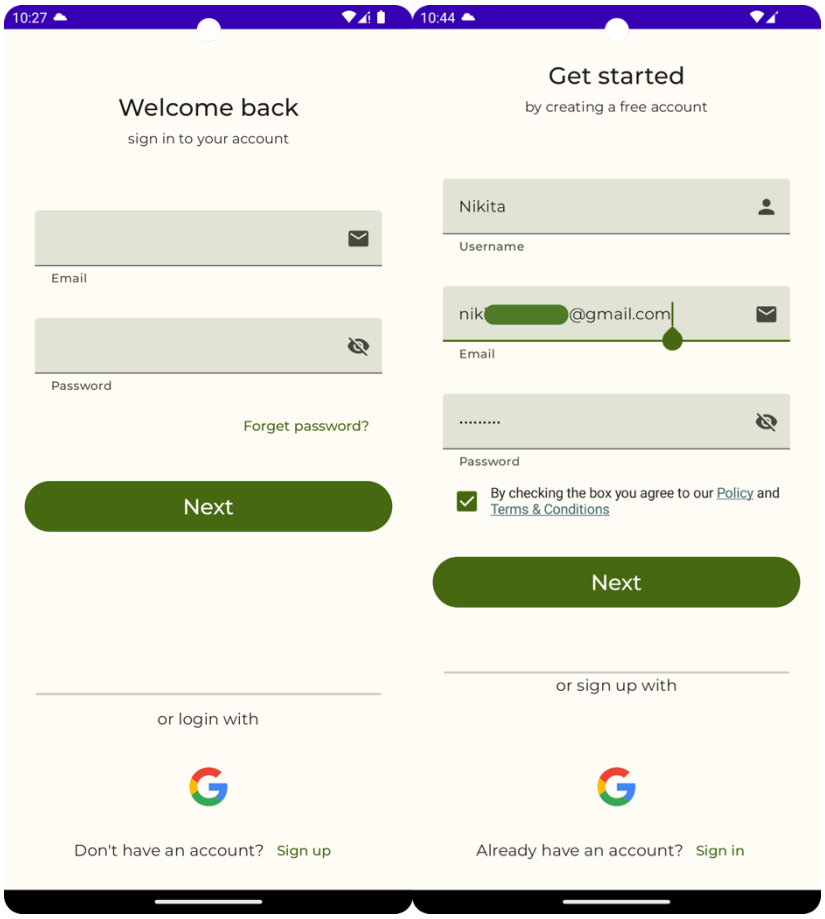


Рисунок 4.1 Екрани Входу та реєстрації у застосунку

Екран авторизації дозволяє користувачам входити до свого акаунту за допомогою електронної адреси та пароля. Поле для введення електронної адреси, дозволяє ввести свою зареєстровану електронну адресу, щоб розпочати

процес авторизації. Для чіткої ідентифікації поле має підпис "Email" (Електронна адреса). Користувачі можуть ввести свій пароль для підтвердження особистості та доступу до свого акаунту. Для забезпечення безпеки, пароль прихований за крапками, що можна змінити натиснувши на іконку ока справа від поля вводу.

Після введення електронної адреси та пароля, користувачі можуть натиснути цю кнопку, щоб продовжити процес авторизації. Якщо дані введено коректно, користувач одразу потрапить на домашній екран застосунку, якщо ж трапиться помилка, буде виведено повідомлення зі змістом помилки, та поля для вводу стануть червоного кольору.

Екран реєстрації схожий до екрана входу, але тут користувач має ввести своє ім’я да погодитися з правилами користування застосунком. Коли ж дані будуть введені та перевірені на коректність, користувач буде перенаправлений на екран з повідомленням про те, що він отримає лист на пошту (рисунок 4.2), в якому має підтвердити її, аби успішно зареєструватися у застосунку.

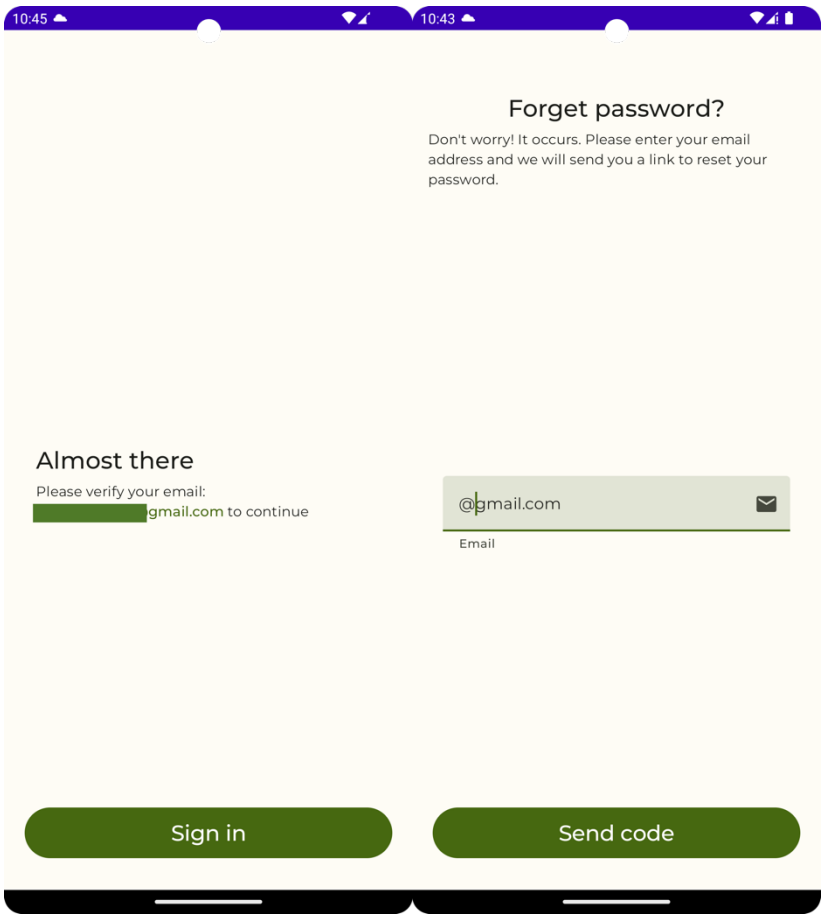


Рисунок 4.2 Допоміжні екран при реєстрації

Якщо ж користувач забув свій пароль, він може перейти за посиланням "Forget password?" (Забули пароль?), щоб відновити його. Посилання розташоване під полем для введення пароля. Екран відновлення пароля зображено на рисунку 4.2, після вводу своєї електронної пошти, на неї буде відправлено лист з подальшими інструкціями.

4.2 Огляд основного інтерфейсу застосунку

Після успішної авторизації, користувач потрапляє на домашній екран застосунку, який має чотири вкладки у нижній частині екрану, а саме *Workouts*, *Routes*, *Create* та *Profile.*

4.2.1 Екран тренувань

На цьому екрані відображаються усі тренування користувача, які були записані програмою. На рисунку 4.3 можна бачити список з тестових тренувань,

а також їх статистику. Тренування можна сортувати за назвою, Наприклад, ввести початок назви, тоді будуть виведені усі тренування з такою назвою. Також можна сортувати по різним параметрам, наприклад дистанція, середня швидкість, спалені калорії та інше. Сортування можна налаштувати як по зростанню, так і по спаданню, наприклад, від найдовшого тренування до найкоротшого. За замовчуванням, тренування сортуються по даті створення, найновіші відображаються зверху списку.

Натиснувши, на так звану, плаваючу кнопку в нижньому правому куті, можна почати нове тренування.

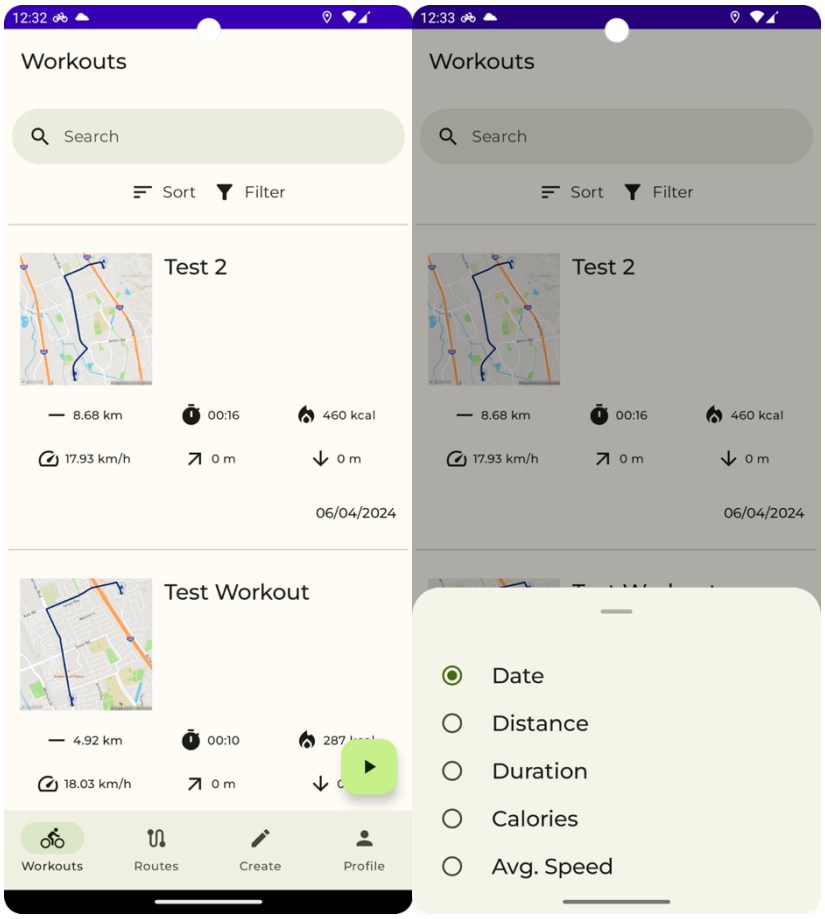


Рисунок 4.3 Екран зі списком тренувань

По натисканню на саме тренування, відкриється екран з більш детальним його описом (рисунок 4.4) . Можна побачити усі основні метрики активності, такі як дистанція, час, середня швидкість, спалені калорії, підйом та спуск, а

також, використовуючи інтерактивний графік можна порівняти швидкість та висоту на різних ділянках маршруту.

Натиснувши на іконку у правому верхньому куті, можна експортувати тренування у форматі GPX або TCX. Кнопка *Export to Strava* експортує тренування напряму до сервісу Strava, при умові, якщо користувач попередньо авторизувався у ньому, через вкладку *Profile.*

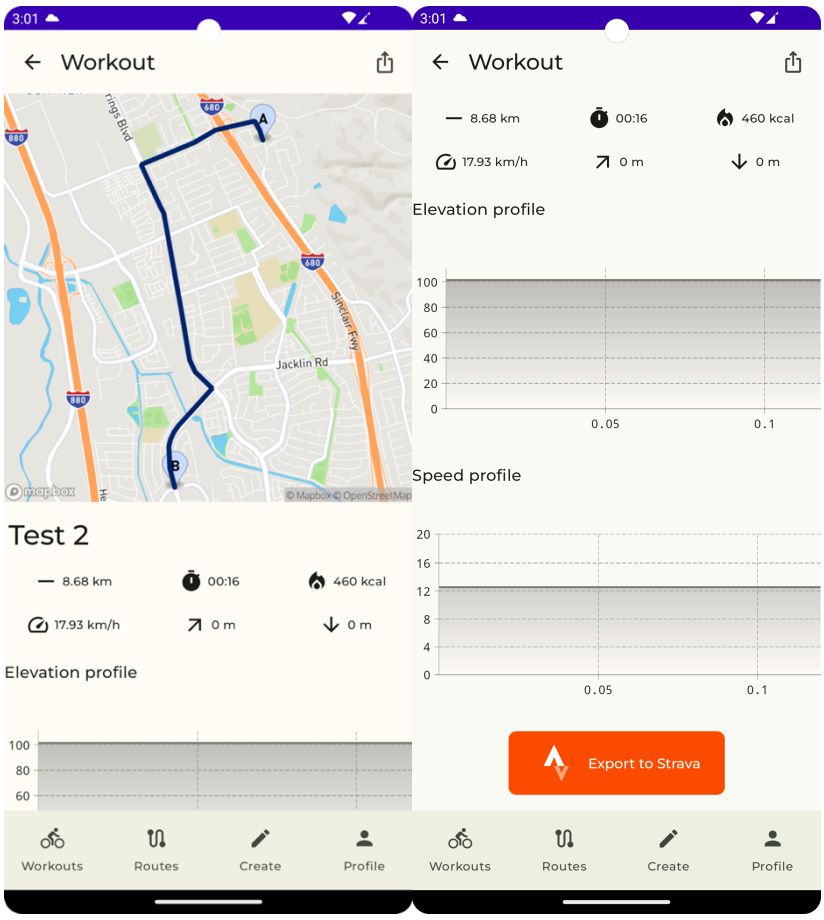


Рисунок 4.4 Екран детального огляду тренування

4.2.2 Екран збережених маршрутів

Всі створені маршрути, відображаються на цьому екрані (рисунок 4.5). Він схожий за інтерфейсом та функціональністю, на попередній екран, тут так само можна почати тренування, відфільтрувати та відсортувати маршрути, а також доступний пошук по назві, але доступно менше фільтрів для сортування,

через те що відсутні показники спалених калорій, середньої та максимальної швидкостей.

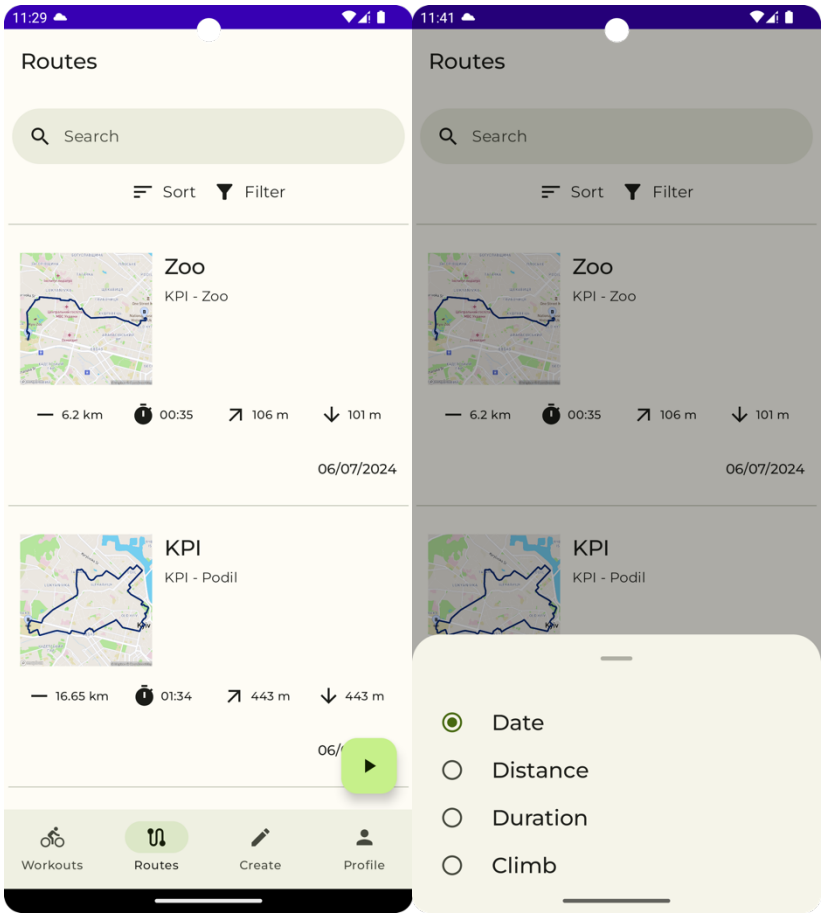


Рисунок 4.5 Екран зі списком маршрутів

Натиснувши на маршрут, користувач потрапляє на екран з детальним описом (Рисунок 4.6). Так само як і тренування, можна експортувати маршрут у форматах GPX та TCX, або ж почати нове тренування, тоді на карті, буде іншим кольором відображатися цей маршрут і користувач з легкістю може слідувати за ним. Доступний перегляд висоти маршруту по всій його довжині.

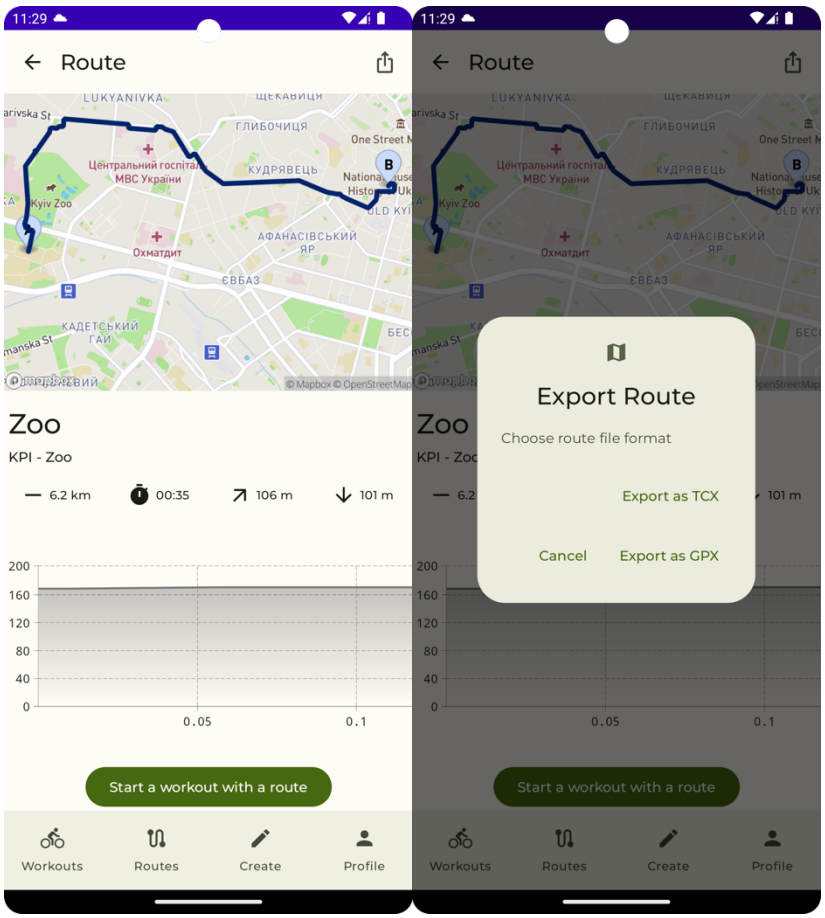


Рисунок 4.6 Екран детального опису маршруту

4.2.3 Екран відслідковування спортивної активності

На рисунку 4.7 показано приклад запису тренування. Синім кольором (колір та товщину лінії можна змінити за бажанням у налаштуваннях) відображається обраний маршрут за яким може слідувати спортсмен. Оранжевим кольором показано маршрут, який вже подолав користувач. Доступно відслідковування поточної та середньої швидкості за час тренування, пройденої дистанції та загальний час тренування. Натиснувши на паузу, відслідковування та секундомір будуть зупинені. Якщо завершити тренування, з`явиться діалогове вікно, з підтвердженням завершення та полем для вводу назви. У стопці повідомлень, відображається повідомлення про час спортивної активності, якщо ж користувач, вийде із застосунку, то зможе повернутися до тренування натиснувши на це повідомлення.

Як і на стандартних застосунках з картами, можна змінити стиль карти, наприклад на супутниковий, що є необхідним під час подорожей по гірський або лісовій місцевості. Також можливо закріпити камеру для слідкуванням за локацією спортсмена, натиснувши відповідну кнопку.

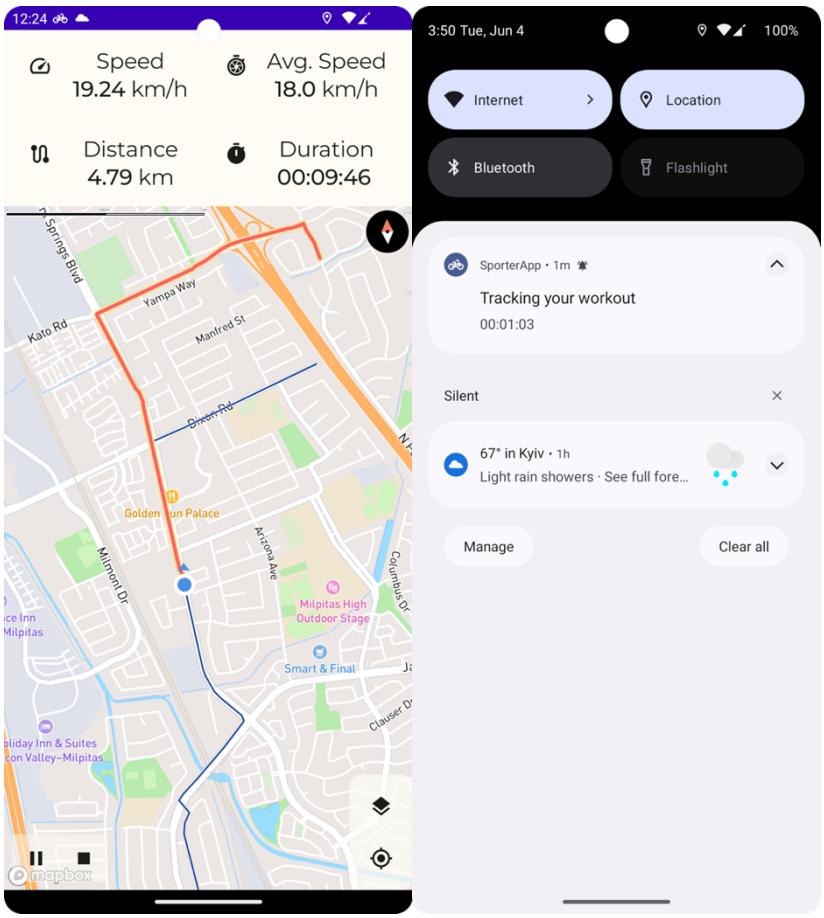


Рисунок 4.7 Екран відслідковування велоподорожі

4.2.4 Екран побудови маршрута

Найголовнішою функцією застосунку, є побудова безкоштовна маршрута, для подальшого використання під час тренувань. Користувач може обирати точки на карті, а система буде будувати найоптимальніший маршрут між ними. Точки відображаються у списку у верхній частині екрану, їх можна міняти місцями, видаляти та додавати нові. У нижній частині екрану відображаються основні дані про побудований маршрут, такі як, дистанція, час,

висота підйому та спуску. Також можливо змінити стиль карти, тоді при збережені, маршрут буде відображатися саме у такому стилі.

У правому верхньому куті, якщо натиснути на три крапки, з’явиться меню, у якому можна відкрити список усіх точок, сфокусувати камеру на маршруті, поставити початкову точку, як кінцеву точку, а також зберегти створений маршрут.

При збережені, відкривається діалогове вікно, я якому потрібно ввести назву та опис маршруту, для подальшого збереження.

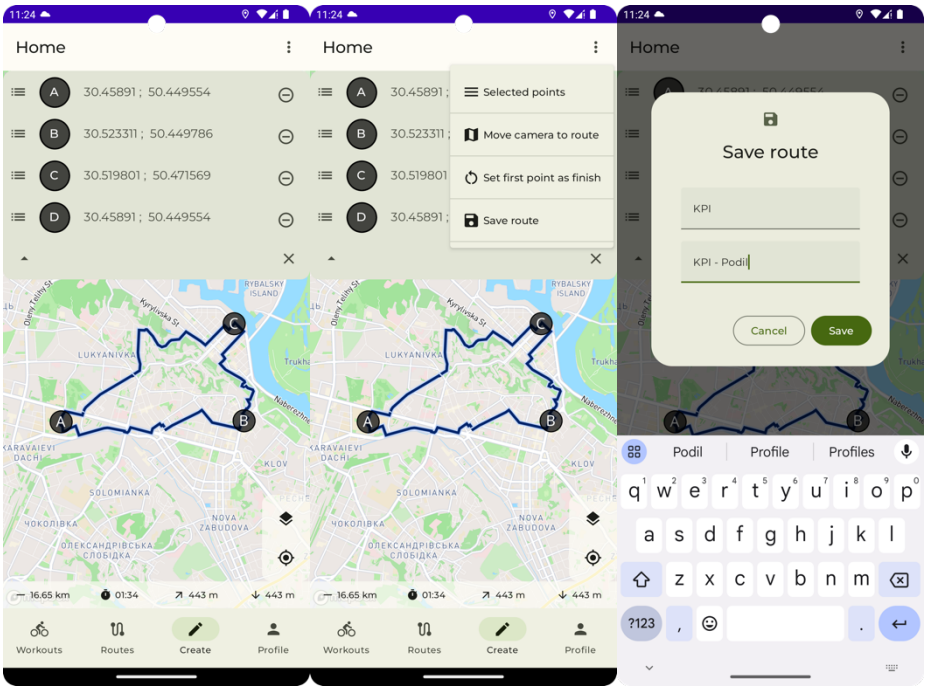


Рисунок 4.8 Екран створення маршруту

4.2.5 Екран налаштувань та профілю користувача

Останнім є профіль спортсмена. На цьому екрані відображається тижнева статистика у вигляді стовпчастої діаграми, яка показує кількість тренувань, проведених користувачем протягом тижня. Кожен стовпчик діаграми відповідає певному дню тижня. Висота стовпчика відповідає кількості тренувань, проведених у цей день.

У розділі *Personal data* можна змінити вік, вагу та зріст спортсмена для найбільш точного розрахунку спалених калорій під час тренування. Далі йдуть налаштування *General*, де можна встановити товщину ліній на карті, змінити відстань камери під час слідкуванням за спортсменом, а. також обрати колір для лінії маршруту та лінії, яка відображає пройдений шлях під час тренування.

Кнопка *Login with Strava*, розпочне процес авторизації у цьому сервісі, для подальшого експорту тренувань, напряму у аккаунт спортсмена. Після натискання, відкриється зовнішній браузер в якому користувач може безпечно ввести дані та завершити процес авторизації у сервісі.

У правому верхньому куті розташована кнопка для виходу із облікового запису, після підтвердження виходу, користувач буде перенаправлений на екран авторизації.

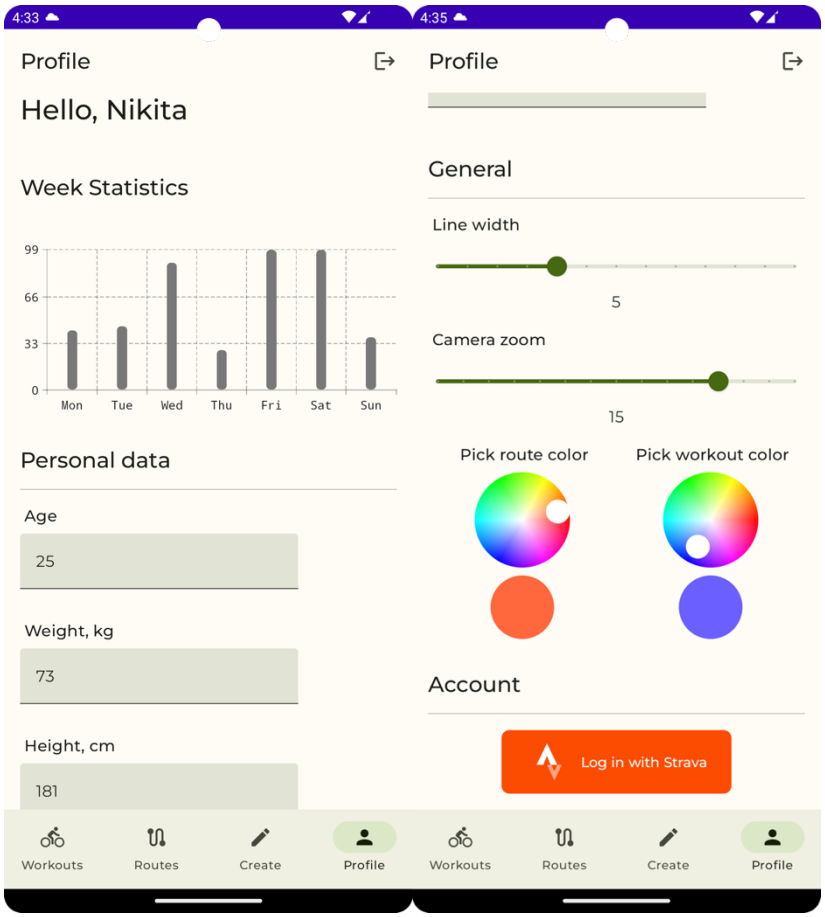


Рисунок 4.9 Профіль та налаштування користувача

4.3 Створення релізної версії системи

Для того, щоб користувачі могли встановити та використовувати застосунок на своїх пристроях, необхідно створити релізну версію застосунку. Це включає збірку APK та AAB файлу, що є готовим пакетом для розповсюдження через Google Play Store або інші магазини застосунків для системи Android.

APK (Android Package) - це формат файлу для встановлення застосунків на ОС Android. APK файл містить усі головні компоненти застосунку, такі як код, ресурси, активності та маніфест. Створення APK файлу є стандартним методом розповсюдження застосунків.

AAB (Android App Bundle) - це більш новий формат, запропонований Google, що дозволяє ефективніше розповсюджувати застосунки через Google Play Store. AAB файл, також містить усі основні ресурси та компоненти застосунку, але на відміну від APK, дозволяє Google Play створювати оптимізовані APK файли для різних конфігурацій пристроїв. Це сприяє зменшенню розміру файлу та покращує продуктивність на різних пристроях користувачів.

Перед створенням релізного файлу необхідно налаштувати підпис застосунку. Він забезпечує автентичність та цілісність файлу. Підпис здійснюється за допомогою ключового файлу keystore, який містить приватний ключ розробника.

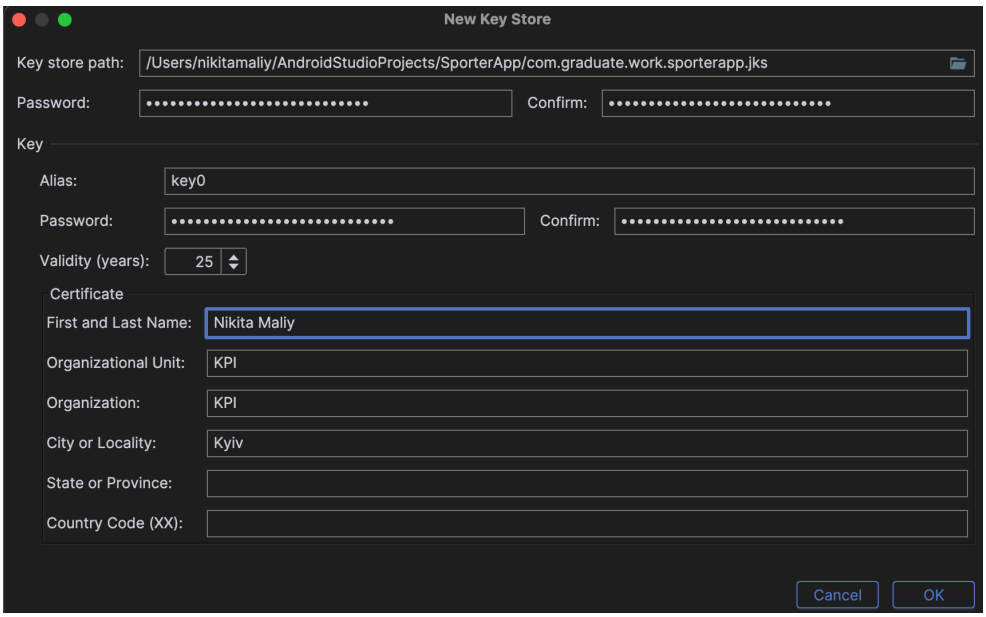


Рисунок 4.10 Створення файлу jks для підпису застосунку

Після створення релізного файлу, його можна завантажити на Google Play Console для розповсюдження серед користувачів. Вибір формату AAB дозволить Google Play автоматично створювати оптимізовані APK файли для різних пристроїв.

На жаль, не вдалося швидко завантажити застосунок у Google Play Store з декількох причин [31]. По-перше Google має суворі політики щодо безпеки та відповідності контенту. Весь новий контент проходить ретельну перевірку на відповідність цим політикам, що займає деякий час. По-друге вимагається додаткове тестування застосунку, особливо якщо він використовує нові функції або інтегрується з іншими сервісами, такими як Strava. Це може включати автоматизоване та ручне тестування для забезпечення безпеки та функціональності.

4.4 Ідеї для покращення застосунку

У процесі розробки було реалізовано багато функцій, проте завжди є простір для подальшого вдосконалення. Далі наведені деякі ідеї для можливого покращення системи у майбутньому:

- Додати можливість відстежування інших видів спортивної активності, таких як біг, плавання чи походи, розширивши таким чином аудиторію користувачів.
- Розширити інтеграцію з популярними спортивними платформами, такими як Garmin, Polar та Suunto, для автоматичної синхронізації тренувальних даних.
- Впровадити більш детальну аналітику тренувань з графіками прогресу, персональними рекомендаціями та передбаченнями на основі історичних даних користувача.
- Додати можливість ділитися тренуваннями та маршрутами з друзями, створювати спільні маршрути та організовувати групові поїздки.
- Впровадити можливість створення та налаштування індивідуальних планів тренувань на основі цілей користувача та його фізичної підготовки.
- Інтеграція з новими типами спортивних пристроїв, такими як смарт-годинники, фітнес-браслети та датчики пульсу, для більш точного збору та аналізу даних.

У майбутньому, реалізація цих покращень допоможе зробити застосунок більш функціональним, зручним та привабливим для широкої аудиторії користувачів.

ВИСНОВОК ДО РОЗДІЛУ 4

У даному розділі було детально розглянуто та протестовано основні функції розробленого застосунку для велосипедних подорожей на платформі Android. Зокрема, провели огляд інтерфейсу авторизації, основного інтерфейсу застосунку та його ключових екранів: тренувань, збережених маршрутів, відслідковування спортивної активності, побудови маршрута, а також екран налаштувань та профілю користувача.

Описаний процес створення релізної версії застосунку, включаючи формат APK та AAB, а також труднощі, які виникли під час завантаження застосунку до Google Play Store. Це дозволило виявити та вирішити потенційні проблеми, що можуть виникнути під час розповсюдження застосунку серед користувачів.

Окрім цього, було представлено ідеї для покращення системи, які можуть бути реалізовані у майбутніх версіях застосунку. Ці ідеї спрямовані на покращення користувацького досвіду, підвищення функціональності та інтеграції з іншими сервісами, що, в свою чергу, допоможе зробити застосунок ще більш корисним та зручним для користувачів.

ВИСНОВКИ

У даній дипломній роботі було виконано всебічне дослідження, розробку та тестування мобільного застосунку для велосипедних подорожей на платформі Android. Основна мета полягала в створенні зручного та функціонального інструменту для велосипедистів, який дозволяє планувати маршрути, відслідковувати тренування, аналізувати дані та експортувати результати до інших сервісів.

Було проведено глибокий аналіз популярності велосипедів, їх соціокультурного впливу та викликів, з якими стикаються велосипедисти. Також розглянуто способи покращення велоподорожей, включаючи планування маршрутів, відслідковування та аналіз тренувань, а також експорт даних. Виконано огляд та порівняння існуючих рішень на ринку, таких як Strava, Komoot, TrainingPeaks та інші.

Обґрунтовано вибір операційної системи Android для розробки мобільного застосунку. Проведено порівняльний аналіз інструментів для розробки ПЗ, таких як Java/Kotlin (Android Studio), Flutter, Kotlin Multiplatform Mobile (KMM) та React Native. Визначено архітектуру застосунку, що включає Data Layer, Domain Layer та Presentation Layer. Описано вибір мови програмування та необхідних API та бібліотек, таких як Jetpack Compose, Dagger-Hilt, Firebase та Mapbox.

Реалізовано базові функції застосунку, включаючи авторизацію користувачів, побудову маршрутів, відслідковування велоподорожей та експорт даних. Інтегровано базу даних Firestore, що дозволяє зберігати, фільтрувати та сортувати дані користувачів.

Проведено детальний огляд інтерфейсу авторизації та основного інтерфейсу застосунку, включаючи екрани тренувань, збережених маршрутів, відслідковування активності, побудови маршрутів та налаштувань профілю

користувача. Описано процес створення релізної версії системи, включаючи формати APK та AAB, а також труднощі, що виникли під час завантаження застосунку до Google Play. Представлено ідеї для подальшого покращення системи, що можуть бути реалізовані у майбутніх версіях застосунку.

Розроблений застосунок для велосипедних подорожей, відповідає сучасним вимогам велосипедистів та надає широкий спектр функцій для покращення їхнього досвіду. Робота над проектом включала аналіз ринку, вибір найкращих технологій та підходів, реалізацію ключових функцій та тестування кінцевого продукту. У процесі роботи було виявлено можливості для подальшого вдосконалення застосунку, що забезпечить його конкурентоспроможність та привабливість для користувачів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. World Bicycle Day [Електронний ресурс] – Режим доступу до ресурсу: https://www.europarl.europa.eu/RegData/etudes/ATAG/2022/729463/EPRS_ATA(2022)729463_EN.pdf.
2. Bicycle Statistics: Usage, Production, Sales, Import, Export [Електронний ресурс] – Режим доступу до ресурсу: https://www.ibike.org/library/statistics-data.htm.
3. The great bicycle boom of 2020 [Електронний ресурс] – Режим доступу до ресурсу: https://www.bbc.com/future/bespoke/made-on-earth/the-great-bicycle-boom-of-2020.html.
4. Keeping pace: American spending on cycling continues amidst pandemic [Електронний ресурс] – Режим доступу до ресурсу: https://www.weforum.org/agenda/2021/06/pandemic-bicycle-boom-covid19-united-states/.
5. https://ehp.niehs.nih.gov/doi/full/10.1289/ehp.0901747 De Hartog, J. J., Boogaard, H., Nijland, H., & Hoek, G. (2010). Do the health benefits of cycling outweigh the risks?. Environmental health perspectives, 118(8), 1109-1116.
6. GPX, TCX, FIT: How to choose the best file extension for sport activity transfer ? [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/decathlondigital/gpx-tcx-fit-how-to-choose-the-best-file-extension-for-sport-activity-transfer-403487337c04.
7. Strava about [Електронний ресурс] – Режим доступу до ресурсу: https://www.strava.com/about.
8. Koomot about [Електронний ресурс] – Режим доступу до ресурсу: https://www.komoot.com/about.
9. TrainingPeaks about us [Електронний ресурс] – Режим доступу до ресурсу: https://www.trainingpeaks.com/about-us.html#about.

10. Suunto app about us [Електронний ресурс] – Режим доступу до ресурсу: https://www.suunto.com/suunto-app/suunto-app-2022/.
11. WE ARE WAHOO ABOUT US [Електронний ресурс] – Режим доступу до ресурсу: https://eu.wahoofitness.com/about-us.
12. The best cycling route planner you\'ll find [Електронний ресурс] – Режим доступу до ресурсу: https://maplocs.app/.
13. Workoutdoors overview [Електронний ресурс] – Режим доступу до ресурсу: http://www.workoutdoors.net.
14. Mobile operating systems targeted by developers worldwide in 2022 and 2023 [Електронний ресурс] – Режим доступу до ресурсу: https://www.statista.com/statistics/1078678/software-development-operating-system-mobile/.
15. Moskala M., Wojda I. Android Development with Kotlin. – Packt Publishing Ltd, 2017.
16. The Good and the Bad of Flutter App Development [Електронний ресурс] – Режим доступу до ресурсу: https://www.altexsoft.com/blog/pros-and-cons-of-flutter-app-development/.
17. Introduction to Multiplatform Programming in Kotlin [Електронний ресурс] – Режим доступу до ресурсу: https://www.baeldung.com/kotlin/multiplatform-programming.
18. React Native – one framework to rule them all [Електронний ресурс] – Режим доступу до ресурсу: https://www.netguru.com/glossary/react-native.
19. Guide to app architecture [Електронний ресурс] – Режим доступу до ресурсу: https://developer.android.com/topic/architecture.
20. Clean Architecture of Android Apps with Practical Examples [Електронний ресурс] – Режим доступу до ресурсу: https://rubygarage.org/blog/clean-android-architecture.

21. Model-View-ViewModel — Android Architecture Patterns [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/@kailashbarochiya/model-view-viewmodel-android-architecture-patterns-5a7122984ee1.
22. Kotlin VS Java – What\'s the Difference? [Електронний ресурс] – Режим доступу до ресурсу: https://www.freecodecamp.org/news/kotlin-vs-java-whats-the-difference/.
23. Використання Jetpack Compose у сучасній Android-розробці [Електронний ресурс] – Режим доступу до ресурсу: https://careers.epam.ua/blog/jetpack-compose-in-android-development.
24. Kesavan R. et al. Firestore: The nosql serverless database for the application developer //2023 IEEE 39th International Conference on Data Engineering (ICDE). – IEEE, 2023. – С. 3376-3388.
25. Map Box Explained [Електронний ресурс] – Режим доступу до ресурсу: https://blakebhowe.medium.com/introduction-2b278b438adb.
26. How many calories does cycling burn? [Електронний ресурс] – Режим доступу до ресурсу: https://tourdevines.com.au/blog/how-many-calories-does-cycling-burn/.
27. Winarno E., Hadikurniawati W., Rosso R. N. Location based service for presence system using haversine method //2017 international conference on innovative and creative information technology (ICITech). – IEEE, 2017. – С. 1-4.
28. Getting Started with the Strava API [Електронний ресурс] – Режим доступу до ресурсу: https://developers.strava.com/docs/getting-started/#oauth.
29. Manage indexes in Cloud Firestore [Електронний ресурс] – Режим доступу до ресурсу: https://firebase.google.com/docs/firestore/query-data/indexing.

30. Query with range and inequality filters on multiple fields overview [Електронний ресурс] – Режим доступу до ресурсу: https://firebase.google.com/docs/firestore/query-data/multiple-range-fields.
31. Правила програми для розробників [Електронний ресурс] – Режим доступу до ресурсу: https://support.google.com/googleplay/android-developer/answer/14906471?hl=uk&visit_id=638531730025693985-1785814437&rd=1.

ДОДАТОК 1

Застосунок для спортивних велосипедних подорожей

Структурна схема системи

ІАЛЦ.467200.004 Д1

Аркушів 1

Київ 2024 р

Зм.	Арк.	№ докум.	Підпис	Дата	ІАЛЦ.467200.004 Д1		
Розробив		Малій М.М			**Застосунок для спортивних велосипедних подорожей**	Літ.	Аркуш 1 / Аркушів 1
Перевірив		Волокита А. М.					
Н. Контр.		Іваніщев Б. В.			Структурна схема системи	КПІ ім. Ігоря Сікорського, ФІОТ, ІО-03	
Затвердив							

ДОДАТОК 2

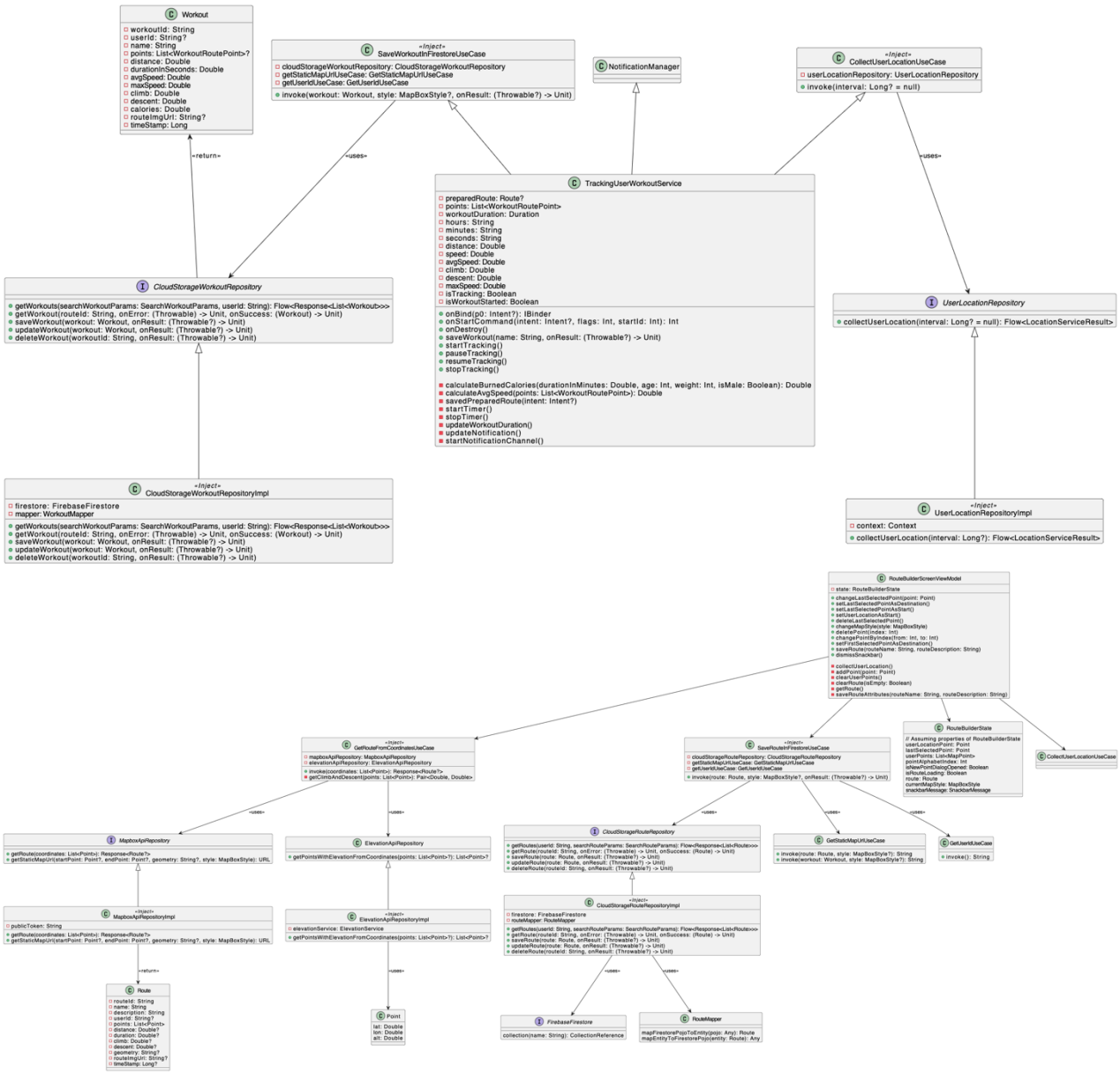
Застосунок для спортивних велосипедних подорожей

Функціональна схема

ІАЛЦ.467200.005 Д2

Аркушів 1

Київ 2024 р



					ІАЛЦ.467200.005 Д2		
Зм.	Арк.	№ докум.	Підпис	Дата			
Розробив		Малій М.М			**Застосунок для спортивних велосипедних подорожей** Функціональна схема	Літ.	Аркуш 1 / Аркушів 1
Перевірив		Волокита А. М.					
						КПІ ім. Ігоря Сікорського, ФІОТ, ІО-03	
Н. Контр.		Іваніщев Б. В.					
Затвердив							

ДОДАТОК 3

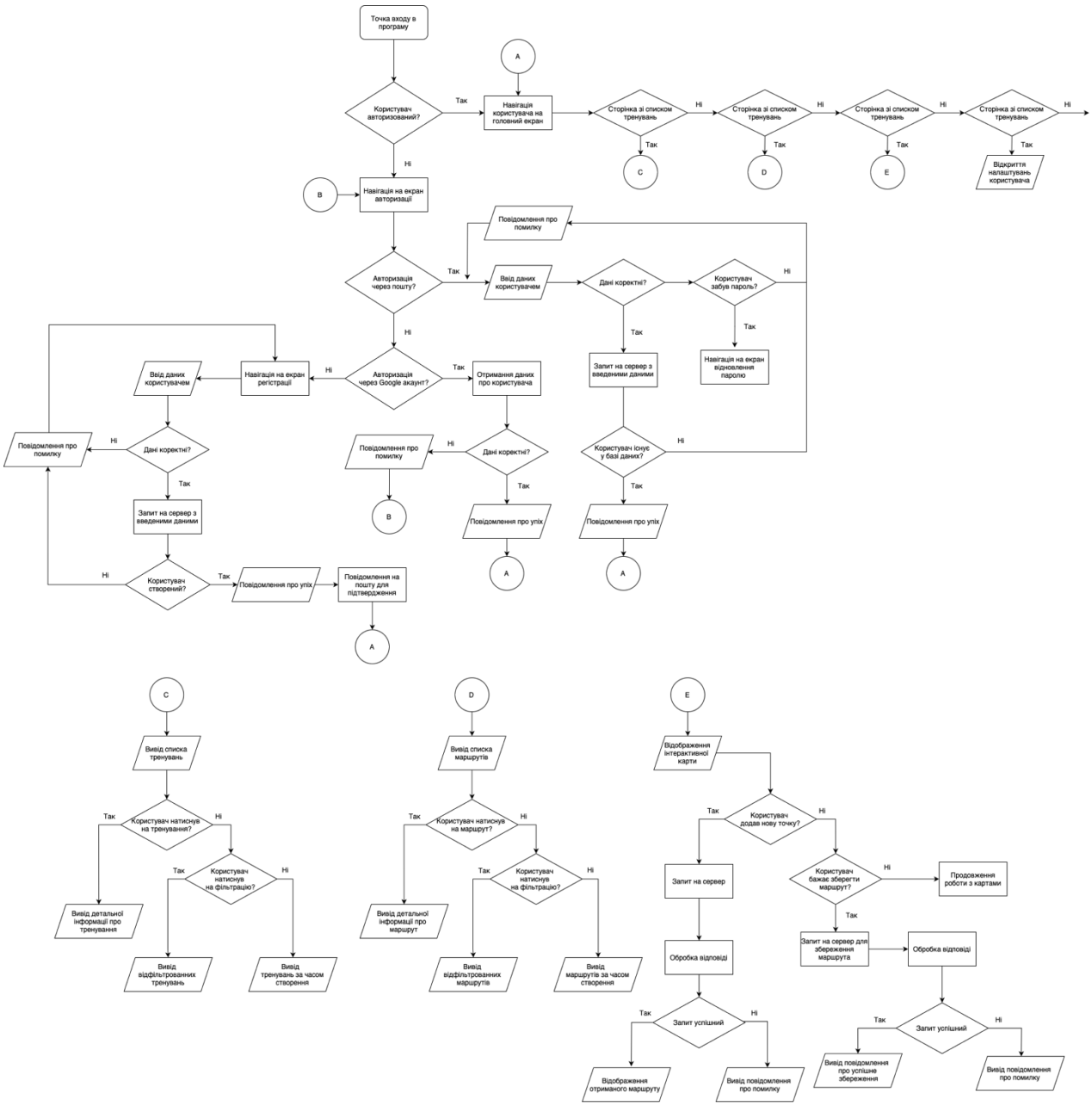
Застосунок для спортивних велосипедних подорожей

Алгоритм дій програмного забезпечення

ІАЛЦ.467200.006 Д3

Аркушів 1

Київ 2024 р



					ІАЛЦ.467200.006.Д3		
Зм.	Арк.	№ докум.	Підпис	Дата			
Розробив		Малій М.М			**Застосунок для спортивних велосипедних подорожей**	Літ.	Аркуш 1 / Аркушів 1
Перевірив		Волокита А. М.			Алгоритм дій програмного забезпечення	КПІ ім. Ігоря Сікорського, ФІОТ, ІО-03	
Н. Контр.		Іваніщев Б. В.					
Затвердив							

ДОДАТОК 4

Застосунок для спортивних велосипедних подорожей

Текст програмного коду

ІАЛЦ.467200.007 Д4

Аркушів 50

Київ 2024 р

MainActivity.kt

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

    private val viewModel by viewModels<MainViewModel>()

    private val launcher = registerForActivityResult(
        ActivityResultContracts.RequestPermission()
    ) {}

    private var isTrackingServiceBound by mutableStateOf(false)
    private lateinit var trackingService: TrackingUserWorkoutService

    // Tracking service binding
    private val connection = object : ServiceConnection {
        override fun onServiceConnected(p0: ComponentName?, p1: IBinder?) {
            val binder = p1 as TrackingUserWorkoutService.LocalBinder
            trackingService = binder.getService()
            isTrackingServiceBound = true
        }

        override fun onServiceDisconnected(p0: ComponentName?) {
            isTrackingServiceBound = false
        }
    }

    override fun onStart() {
        super.onStart()
        Intent(this, TrackingUserWorkoutService::class.java).also { intent ->
            bindService(intent, connection, BIND_AUTO_CREATE)
        }
    }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        installSplashScreen().apply {
            setKeepOnScreenCondition { viewModel.isSplashLoading.value }
        }
        requestNotificationPermission()
        setContent {
            if (isTrackingServiceBound) {
                val authState = viewModel.currentAuthState.collectAsState().value
                AppTheme {
                    Surface(
                        modifier = Modifier.fillMaxSize(),
                        color = MaterialTheme.colorScheme.background
                    ) {
                        val navController = rememberNavController()
                        NavHost(
                            navController = navController,
                            startDestination = if (authState?.isEmailVerified == true) {
                                AppNavigation.Home.HOME_FEATURE_SCREEN_ROUTE
                            } else {
                                if (trackingService.isWorkoutStarted) {
                                    AppNavigation.Workout.TRACK_FEATURE_SCREEN_ROUTE
                                } else {
                                    AppNavigation.Auth.AUTH_FEATURE_SCREEN_ROUTE
                                }
                            }
                        ) {
                            navigation(
                                startDestination = AppNavigation.Auth.SignInScreen.route,
                                route = AppNavigation.Auth.AUTH_FEATURE_SCREEN_ROUTE,
                            ) {
                                composable(AppNavigation.Auth.SignInScreen.route) {
                                    SignInCompleteScreen(
```

```
                                    email = authState?.email,
                                    navController = navController
                                )
                            }
                            composable(AppNavigation.Auth.SignUpScreen.route) {
                                SignUpCompleteScreen(navController = navController)
                            }
                            composable(AppNavigation.Auth.ForgetPasswordScreen.route) {
                                ForgetPasswordCompleteScreen(navController = navController)
                            }
                            composable(AppNavigation.Auth.EmailVerificationScreen.route) {
                                EmailVerificationCompleteScreen(
                                    email = authState?.email ?: "",
                                    navController = navController,
                                )
                            }
                        }
                        navigation(
                            route = AppNavigation.Workout.TRACK_FEATURE_SCREEN_ROUTE,
                            startDestination = AppNavigation.Workout.TrackScreen.route
                        ) {
                            composable(
                                AppNavigation.Workout.TrackScreen.route,
                                deepLinks = listOf(
                                    navDeepLink {
                                        uriPattern = pendingUri
                                    }
                                ),
                                arguments = listOf(

navArgument(AppNavigation.Workout.TrackScreen.ROUTE_ID_ARG) {
                                        type = NavType.StringType
                                    },

                                )
                            ) { backStackEntry ->
                                val routeId = backStackEntry.arguments?.getString(
                                    AppNavigation.Workout.TrackScreen.ROUTE_ID_ARG
                                )
                                TrackCompleteScreen(routeId, trackingService) {
                                    navController.navigate(
                                        AppNavigation.Home.HOME_FEATURE_SCREEN_ROUTE,
                                        navOptions = navOptions {

popUpTo(AppNavigation.Home.HOME_FEATURE_SCREEN_ROUTE) {
                                                inclusive = true
                                            }
                                        })
                                    }
                                }
                            }
                            composable(
                                AppNavigation.Home.HOME_FEATURE_SCREEN_ROUTE,
                            ) {
                                HomeCompleteScreen(
                                    navigateToTrackScreen = {
                                        Log.d("AAAAAA", "navigateToTrackScreen: $it");
                                        navController.navigate(

AppNavigation.Workout.TrackScreen.createTrackScreen(it),
                                            navOptions = navOptions {

popUpTo(AppNavigation.Home.HOME_FEATURE_SCREEN_ROUTE) {
                                                inclusive = true
                                            }
                                        })
                                    }
                                )
                            }
                        }
```

```
                    }
                }
            }
        }
    }

    private fun requestNotificationPermission() {
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.TIRAMISU) {
            val permission = Manifest.permission.POST_NOTIFICATIONS
            if (checkSelfPermission(permission) != PackageManager.PERMISSION_GRANTED) {
                launcher.launch(permission)
            }
        }
    }

    override fun onStop() {
        super.onStop()
        unbindService(connection)
        isTrackingServiceBound = false
    }
}
```

MainViewModel.kt

```
@HiltViewModel
class MainViewModel @Inject constructor(
    private val getCurrentAuthStateUseCase: GetCurrentAuthStateUseCase,
): ViewModel() {

    val currentAuthState = getAuthState()

    var isSplashLoading = mutableStateOf(true)
        private set

    init {
        getAuthState()
        viewModelScope.launch {
            currentAuthState.collect {
                isSplashLoading.value = false
            }
        }
    }

    private fun getAuthState() = getCurrentAuthStateUseCase(viewModelScope)
}
```

AppNavigation.kt

```
enum class AuthScreens(val screenName: String) {
    SIGN_UP("signUp"),
    SIGN_IN("signIn"),
    FORGET_PASSWORD("forgetPassword"),
    EMAIL_VERIFICATION("emailVerification")
}

sealed class AppNavigation {

    sealed class Auth(val route: String) : AppNavigation() {
        data object SignInScreen : Auth(AuthScreens.SIGN_IN.screenName)
```

```
        data object SignUpScreen : Auth(AuthScreens.SIGN_UP.screenName)

        data object ForgetPasswordScreen : Auth(AuthScreens.FORGET_PASSWORD.screenName)

        data object EmailVerificationScreen : Auth(AuthScreens.EMAIL_VERIFICATION.screenName)
        companion object {
            const val AUTH_FEATURE_SCREEN_ROUTE = "auth"
        }
    }

    sealed class Home(val route: String) : AppNavigation() {
        data object SavedRoutesScreen : Home("savedRoute")
        data object WorkoutsScreen : Home("workout")
        data object CreateRouteScreen : Home("createRoute")
        data object ProfileScreen : Home("profile")
        data object RedirectStravaScreen : Home("redirectStrava")
        data object RoutePageScreen : Home("routePage/{routeId}") {
            const val ROUTE_ID_ARG = "routeId"
            fun createRoutePageScreen(routeId: String) = "routePage/$routeId"
        }
        data object WorkoutPageScreen : Home("workoutPage/{workoutId}") {
            const val WORKOUT_ID_ARG = "workoutId"
            fun createWorkoutPageScreen(workoutId: String) = "workoutPage/$workoutId"
        }

        companion object {
            const val HOME_FEATURE_SCREEN_ROUTE = "home"
        }
    }

    sealed class Workout(val route: String) : AppNavigation() {
        data object TrackScreen : Workout("track/{routeId}") {
            const val ROUTE_ID_ARG = "routeId"
            fun createTrackScreen(routeId: String?) = "track/$routeId"
        }

        companion object {
            const val TRACK_FEATURE_SCREEN_ROUTE = "track"
        }
    }
}
```

ElevationService.kt

```
@ApiUrl(url = "https://api.open-meteo.com/v1/")
interface ElevationService {
    @GET("elevation")
    fun getElevation(
        @Query("latitude") latitudes: List<Double>,
        @Query("longitude") longitudes: List<Double>,
    ): Call<ElevationResponsePojo>
}
```

ElevationApiRepositoryImpl.kt

```
class ElevationApiRepositoryImpl @Inject constructor(
    private val elevationService: ElevationService,
) : ElevationApiRepository {

    override suspend fun getPointsWithElevationFromCoordinates(
```

```
        points: List<Point>?,
    ): List<Point>? = suspendCoroutine { continuation ->
        if (points.isNullOrEmpty()) {
            continuation.resume(null)
            return@suspendCoroutine
        }

        val pointsMutableList = points.toMutableList()
        val batchSize = 100
        val batches = pointsMutableList.chunked(batchSize)

        val allPointsWithAltitude = mutableListOf<Point>()

        // process first batch
        fun processBatch(batch: List<Point>) {
            // get points without altitude
            val pointsWithoutAltitude: MutableMap<Int, Point> = mutableMapOf()
            batch.forEachIndexed { index, point ->
                // filter points without altitude
                if (!point.hasAltitude()) {
                    pointsWithoutAltitude[index] = point
                }
            }
            // get points with altitude
            if (pointsWithoutAltitude.isEmpty()) {
                allPointsWithAltitude.addAll(batch)
                // check if all points are processed
                if (allPointsWithAltitude.size == pointsMutableList.size) {
                    continuation.resume(allPointsWithAltitude)
                } else {
                    val remainingBatches = batches.drop(allPointsWithAltitude.size / batchSize)
                    remainingBatches.firstOrNull()?.let { processBatch(it) }
                }
                return
            }
            // get points latitudes for api request
            val latitudes = pointsWithoutAltitude.values.map { it.latitude() }
            // get points longitudes for api request
            val longitudes = pointsWithoutAltitude.values.map { it.longitude() }
            // get points altitudes
            elevationService.getElevation(
                latitudes = latitudes,
                longitudes = longitudes
            ).enqueue(object : Callback<ElevationResponsePojo> {
                // on success response
                override fun onResponse(
                    call: Call<ElevationResponsePojo>,
                    response: Response<ElevationResponsePojo>,
                ) {
                    val altitudes = response.body()?.elevation
                    var altitudeIndex = 0
                    if (altitudes != null) {
                        pointsWithoutAltitude.forEach { (key, point) ->
                            pointsWithoutAltitude[key] = Point.fromLngLat(
                                point.longitude(),
                                point.latitude(),
                                altitudes[altitudeIndex]
                            )
                            altitudeIndex++
                        }
                    }
                    allPointsWithAltitude.addAll(pointsWithoutAltitude.values)
                    if (allPointsWithAltitude.size == pointsMutableList.size) {
                        continuation.resume(allPointsWithAltitude)
                    } else {
                        val remainingBatches = batches.drop(allPointsWithAltitude.size /
batchSize)
                        remainingBatches.firstOrNull()?.let { processBatch(it) }
                    }
                }
```

```
                // on failure
                override fun onFailure(call: Call<ElevationResponsePojo>, t: Throwable) {
                    allPointsWithAltitude.addAll(pointsWithoutAltitude.values)
                    if (allPointsWithAltitude.size == pointsMutableList.size) {
                        continuation.resume(allPointsWithAltitude)
                    } else {
                        val remainingBatches = batches.drop(allPointsWithAltitude.size /
batchSize)
                        remainingBatches.firstOrNull()?.let { processBatch(it) }
                    }
                }
            })
        }
        // process first batch
        processBatch(batches.first())
    }
}
class RetrofitApiFactory {
    private val defaultApiUrl = "https://random-word-api.herokuapp.com/"

    fun <T> createInstance(clazz: Class<T>): T {
        val apiUrlAnnotation = clazz.annotations.find { it is ApiUrl } as ApiUrl?
        val url = apiUrlAnnotation?.url ?: defaultApiUrl
        return retrofit(url).create(clazz)
    }

    private fun retrofit(apiUrl: String) = Retrofit.Builder()
        .baseUrl(apiUrl)
        .client(okHttpClient())
        .addConverterFactory(GsonConverterFactory.create())
        .build()

    private fun okHttpClient(): OkHttpClient = OkHttpClient.Builder().build()
}
```

StravaApi.kt

```
interface StravaApiRepository {

    suspend fun getToken(
        clientId: String,
        clientSecret: String,
        code: String
    ): String?

    suspend fun uploadGpxFile(name: String, token: String): Response<Unit>
}


data class StravaTokenResponse(
    @SerializedName("access_token") val accessToken: String,
    @SerializedName("athlete") val athlete: Athlete
)

data class Athlete(
    @SerializedName("id") val id: Long,
    @SerializedName("username") val username: String
)


data class UploadResponse(
    @SerializedName("id") val id: Long,
    @SerializedName("id_str") val idStr: String,
    @SerializedName("external_id") val externalId: String,
    @SerializedName("error") val error: String?,
    @SerializedName("status") val status: String,
```

```
    @SerializedName("activity_id") val activityId: Long?
)


@ApiUrl(url = "https://www.strava.com/")
interface StravaService {
    @FormUrlEncoded
    @POST("oauth/token")
    fun getToken(
        @Field("client_id") clientId: String,
        @Field("client_secret") clientSecret: String,
        @Field("code") code: String,
        @Field("grant_type") grantType: String = "authorization_code"
    ): Call<StravaTokenResponse>

    @Multipart
    @POST("api/v3/uploads")
    fun uploadActivity(
        @Part file: MultipartBody.Part,
        @Part("name") name: String,
        @Part("description") description: String,
        @Query("data_type") data_type: String,
        @Header("Authorization") authorization: String
    ): Call<UploadResponse>
}


class StravaApiRepositoryImpl @Inject constructor(
    private val stravaService: StravaService,
    @ApplicationContext private val context: android.content.Context,
) : StravaApiRepository {
    override suspend fun getToken(
        clientId: String,
        clientSecret: String,
        code: String,
    ): String? {
        return suspendCoroutine { continuation ->
            val call = stravaService.getToken(clientId, clientSecret, code)
            call.enqueue(object : retrofit2.Callback<StravaTokenResponse> {
                override fun onResponse(
                    call: Call<StravaTokenResponse>,
                    response: retrofit2.Response<StravaTokenResponse>,
                ) {
                    if (response.isSuccessful) {
                        val tokenResponse = response.body()
                        val accessToken = tokenResponse?.accessToken
                        continuation.resume(accessToken)
                    } else {
                        continuation.resume(null)
                    }
                }

                override fun onFailure(call: Call<StravaTokenResponse>, t: Throwable) {
                    continuation.resume(null)
                }
            })
        }
    }

    override suspend fun uploadGpxFile(
        name: String,
        token: String,
    ): com.graduate.work.sporterapp.core.Response<Unit> {
        val filePart = prepareFilePart(filePath = name)
        val call = stravaService.uploadActivity(
            file = filePart,
            data_type = "gpx",
            name = name,
            description = "Uploaded via SporterApp",
            authorization = "Bearer $token"
```

```
        )
        return suspendCoroutine { continuation ->
            call.enqueue(object : Callback<UploadResponse> {
                override fun onResponse(
                    call: Call<UploadResponse>,
                    response: retrofit2.Response<UploadResponse>,
                ) {
                    if (response.isSuccessful) {
                        continuation.resume(Success(Unit))
                    } else {
                        continuation.resume(
                            com.graduate.work.sporterapp.core.Response.Failure(
                                response.errorBody()?.string() ?: "Unknown error"
                            )
                        )
                    }
                }

                override fun onFailure(call: Call<UploadResponse>, t: Throwable) {
                    continuation.resume(
                        com.graduate.work.sporterapp.core.Response.Failure(
                            t.localizedMessage ?: "Unknown error"
                        )
                    )
                }
            })
        }
    }

    fun prepareFilePart(filePath: String): MultipartBody.Part {
        val file = File(context.filesDir, filePath)
        val requestFile = file.asRequestBody("application/gpx+xml".toMediaTypeOrNull())
        return MultipartBody.Part.createFormData("file", file.name, requestFile)
    }
}
```

FirebaseAuthRepositoryImpl.kt

```
interface FirebaseAuthRepository {
    fun getAuthState(scope: CoroutineScope): StateFlow<FirebaseUser?>

    suspend fun authWithGoogle(credential: GetCredentialResponse): Response<AuthResult>

    suspend fun signUpWithMailAndPassword(email: String, password: String): Response<Unit>

    suspend fun signInWithEmailAndPassword(
        email: String,
        password: String,
    ): Response<AuthResult>

    suspend fun sendPasswordResetEmail(email: String): Response<Unit>

    fun getUserId(): String?
}


class FirebaseAuthRepositoryImpl @Inject constructor(
    private val auth: FirebaseAuth,
) : FirebaseAuthRepository {

    override fun getAuthState(scope: CoroutineScope): StateFlow<FirebaseUser?> = callbackFlow {
        val authStateListener = FirebaseAuth.AuthStateListener { auth ->
            trySend(auth.currentUser)
        }
        auth.addAuthStateListener(authStateListener)
        awaitClose {
            auth.removeAuthStateListener(authStateListener)
```

```
        }
    }.stateIn(scope, SharingStarted.WhileSubscribed(), auth.currentUser)

    override suspend fun authWithGoogle(credential: GetCredentialResponse): Response<AuthResult> {
        try {
            val googleIdTokenCredential =
                GoogleIdTokenCredential.createFrom(credential.credential.data)
            val googleIdToken = googleIdTokenCredential.idToken
            val firebaseCredential =
                GoogleAuthProvider.getCredential(googleIdToken, null)
            return suspendCoroutine { continuation ->
                auth.signInWithCredential(firebaseCredential)
                    .addOnCompleteListener { task ->
                        if (task.isSuccessful) {
                            continuation.resume(Response.Success(task.result))
                        } else {

continuation.resume(Response.Failure(task.exception?.localizedMessage.toString()))
                        }
                    }
            }
        } catch (e: Exception) {
            // TODO move to string resource
            return Response.Failure("Unknown error")
        }
    }

    override suspend fun signUpWithMailAndPassword(
        email: String,
        password: String,
    ): Response<Unit> {
        return suspendCoroutine { continuation ->
            auth.createUserWithEmailAndPassword(email, password)
                .addOnCompleteListener { task ->
                    if (task.isSuccessful) {
                        auth.currentUser?.sendEmailVerification()
                        continuation.resume(Response.Success(Unit))
                    } else {

continuation.resume(Response.Failure(task.exception?.localizedMessage.toString()))
                    }
                }
        }
    }

    override suspend fun signInWithEmailAndPassword(
        email: String,
        password: String,
    ): Response<AuthResult> {
        return suspendCoroutine { continuation ->
            auth.signInWithEmailAndPassword(email, password)
                .addOnCompleteListener { task ->
                    if (task.isSuccessful) {
                        if (task.result.user?.isEmailVerified == false) {
                            task.result.user?.sendEmailVerification()
                        }
                        continuation.resume(Response.Success(task.result))
                    } else {

continuation.resume(Response.Failure(task.exception?.localizedMessage.toString()))
                    }
                }
        }
    }

    override suspend fun sendPasswordResetEmail(email: String): Response<Unit> {
        return suspendCoroutine { continuation ->
            auth.sendPasswordResetEmail(email)
                .addOnCompleteListener {
                    if (it.isSuccessful) {
```

```
                        continuation.resume(Response.Success(Unit))
                    } else {

continuation.resume(Response.Failure(it.exception?.localizedMessage.toString()))
                    }
                }
            }
        }

        override fun getUserId(): String? {
            return auth.currentUser?.uid
        }
}
```

CloudStorageRouteRepositoryImpl.kt

```
interface CloudStorageRouteRepository {
    fun getRoutes(
        userId: String,
        searchRouteParams: SearchRouteParams,
    ): Flow<Response<List<Route>>>

    fun getRoute(routeId: String, onError: (Throwable) -> Unit, onSuccess: (Route) -> Unit)
    fun saveRoute(route: Route, onResult: (Throwable?) -> Unit)
    fun updateRoute(route: Route, onResult: (Throwable?) -> Unit)
    fun deleteRoute(routeId: String, onResult: (Throwable?) -> Unit)
}

class CloudStorageRouteRepositoryImpl @Inject constructor(
    private val firestore: FirebaseFirestore,
) : CloudStorageRouteRepository {

    private val routeMapper = RouteMapper()

    override fun getRoutes(
        userId: String,
        searchRouteParams: SearchRouteParams
    ): Flow<Response<List<Route>>> = callbackFlow {
        var query = firestore
            .collection(ROUTES_COLLECTION)
            .whereEqualTo("userId", userId)
            .orderBy(searchRouteParams.sortType.dbValue, if (searchRouteParams.sortDirection ==
SortDirection.ASC) Query.Direction.ASCENDING else Query.Direction.DESCENDING)
        if (searchRouteParams.searchText.isNotEmpty()) {
            query = query
                .whereGreaterThanOrEqualTo("name", searchRouteParams.searchText)
                .whereLessThanOrEqualTo("name", searchRouteParams.searchText + '\uf8ff')
        }
        val snapshotListener = query.addSnapshotListener { snapshot, e ->
            val response = if (snapshot != null) {
                val workouts = snapshot.toObjects(FirestoreRoutePojo::class.java).map {
                    routeMapper.mapFirestorePojoToEntity(it)
                }
                Response.Success(workouts)
            } else {
                Response.Failure(e?.message ?: e.toString())
            }
            trySend(response).isSuccess
        }
        awaitClose {
            snapshotListener.remove()
        }
    }

    override fun getRoute(
        routeId: String,
        onError: (Throwable) -> Unit,
```

```
            onSuccess: (Route) -> Unit,
        ) {
            firestore.collection(ROUTES_COLLECTION).document(routeId).get()
                .addOnSuccessListener {
                    onSuccess(
                        routeMapper.mapFirestorePojoToEntity(
                            it.toObject() ?: FirestoreRoutePojo()
                        )
                    )
                }
                .addOnFailureListener {
                    onError(it)
                }
        }

        override fun saveRoute(route: Route, onResult: (Throwable?) -> Unit) {
            firestore.collection(ROUTES_COLLECTION).add(routeMapper.mapEntityToFirestorePojo(route))
                .addOnCompleteListener {
                    onResult(it.exception)
                }
        }

        override fun updateRoute(route: Route, onResult: (Throwable?) -> Unit) {
            firestore.collection(ROUTES_COLLECTION).document(route.routeId).set(route)
                .addOnCompleteListener {
                    onResult(it.exception)
                }
        }

        override fun deleteRoute(routeId: String, onResult: (Throwable?) -> Unit) {
            firestore.collection(ROUTES_COLLECTION).document(routeId).delete()
                .addOnCompleteListener {
                    onResult(it.exception)
                }
        }

        companion object {
            private const val ROUTES_COLLECTION = "routes"
        }
    }


    class RouteMapper {


        fun mapFirestorePojoToEntity(entity: FirestoreRoutePojo) = Route(
            routeId = entity.routeId,
            name = entity.name,
            description = entity.description,
            userId = entity.userId,
            points = entity.points?.map { Point.fromLngLat(it.lng, it.lat, it.altitude) }
                ?: emptyList(),
            distance = entity.distance,
            duration = entity.duration,
            climb = entity.climb,
            descent = entity.descent,
            geometry = entity.geometry,
            routeImgUrl = entity.routeImgUrl,
            timeStamp = entity.timeStamp
        )

        fun mapEntityToFirestorePojo(route: Route) = FirestoreRoutePojo(
            routeId = route.routeId,
            name = route.name,
            description = route.description,
            userId = route.userId,
            points = route.points.map {
                RoutePoint(
                    lat = it.latitude(),
                    lng = it.longitude(),
```

```
                altitude = it.altitude()
            )
        },
        distance = route.distance,
        duration = route.duration,
        climb = route.climb,
        descent = route.descent,
        geometry = route.geometry,
        routeImgUrl = route.routeImgUrl,
        timeStamp = route.timeStamp
    )
}


data class RoutePoint(
    val lng: Double = 0.0,
    val lat: Double = 0.0,
    val altitude: Double = 0.0,
)

data class FirestoreRoutePojo(
    @DocumentId val routeId: String = "",
    val userId: String? = null,
    val name: String = "",
    val description: String = "",
    val points: List<RoutePoint>? = null,
    val distance: Double? = null,
    val duration: Double? = null,
    val climb: Double? = null,
    val descent: Double? = null,
    val geometry: String? = null,
    val routeImgUrl: String? = null,
    val timeStamp: Long? = null,
)
```

CloudStorageWorkoutRepositoryImpl.kt

```
interface CloudStorageWorkoutRepository {
    fun getWorkouts(
        searchWorkoutParams: SearchWorkoutParams,
        userId: String,
    ): Flow<Response<List<Workout>>>
    fun getWorkout(routeId: String, onError: (Throwable) -> Unit, onSuccess: (Workout) -> Unit)
    fun saveWorkout(workout: Workout, onResult: (Throwable?) -> Unit)
    fun updateWorkout(workout: Workout, onResult: (Throwable?) -> Unit)
    fun deleteWorkout(workoutId: String, onResult: (Throwable?) -> Unit)
}


class CloudStorageWorkoutRepositoryImpl @Inject constructor(
    private val firestore: FirebaseFirestore,
) : CloudStorageWorkoutRepository {

    private val mapper = WorkoutMapper()
    override fun getWorkouts(
        searchWorkoutParams: SearchWorkoutParams,
        userId: String,
    ): Flow<Response<List<Workout>>> = callbackFlow {
        // create query for workouts
        var query = firestore
            // choose workout collection
            .collection(WORKOUT_COLLECTION)
            // get only user's workouts
            .whereEqualTo("userId", userId)
            // sort by user filters
```

```
                .orderBy(searchWorkoutParams.sortWorkoutType.dbValue, if
(searchWorkoutParams.sortDirection == SortDirection.ASC) {
                    Direction.ASCENDING
                } else Direction.DESCENDING)
            // search by text
            if (searchWorkoutParams.searchText.isNotEmpty()) {
                query = query
                    .whereGreaterThanOrEqualTo("name", searchWorkoutParams.searchText)
                    .whereLessThanOrEqualTo("name", searchWorkoutParams.searchText + '\uf8ff')
            }
            // add snapshot listener
            val snapshotListener = query.addSnapshotListener { snapshot, e ->
                val response = if (snapshot != null) {
                    // map firestore pojos to domain pojos
                    val workouts = snapshot.toObjects(WorkoutFirestorePojo::class.java).map {
                        mapper.mapFirestorePojoToEntity(it)
                    }
                    Response.Success(workouts)
                } else {
                    Response.Failure(e?.message ?: e.toString())
                }
                trySend(response).isSuccess
            }
            // close listener
            awaitClose {
                snapshotListener.remove()
            }
        }

        override fun getWorkout(
            routeId: String,
            onError: (Throwable) -> Unit,
            onSuccess: (Workout) -> Unit,
        ) {
            firestore.collection(WORKOUT_COLLECTION).document(routeId).get()
                .addOnSuccessListener {
                    onSuccess(
                        mapper.mapFirestorePojoToEntity(
                            it.toObject() ?: WorkoutFirestorePojo()
                        )
                    )
                }
                .addOnFailureListener {
                    onError(it)
                }
        }

        override fun saveWorkout(workout: Workout, onResult: (Throwable?) -> Unit) {
            firestore.collection(WORKOUT_COLLECTION).add(mapper.mapEntityToFirestorePojo(workout))
                .addOnCompleteListener {
                    onResult(it.exception)
                }
        }

        override fun updateWorkout(workout: Workout, onResult: (Throwable?) -> Unit) {
            firestore.collection(WORKOUT_COLLECTION).document(workout.workoutId).set(workout)
                .addOnCompleteListener {
                    onResult(it.exception)
                }
        }

        override fun deleteWorkout(workoutId: String, onResult: (Throwable?) -> Unit) {
            firestore.collection(WORKOUT_COLLECTION).document(workoutId).delete()
                .addOnCompleteListener {
                    onResult(it.exception)
                }
        }

        companion object {
```

```
        private const val WORKOUT_COLLECTION = "workouts"
    }

data class WorkoutFirestoreRoutePoint(
    val lng: Double = 0.0,
    val lat: Double = 0.0,
    val altitude: Double = 0.0,
    val distanceFromStart: Double = 0.0,
    val speed: Double = 0.0,
    val timeStamp: Long = 0,
)

data class WorkoutFirestorePojo(
    @DocumentId val workoutId: String = "",
    val userId: String? = null,
    val name: String = "",
    val points: List<WorkoutFirestoreRoutePoint>? = null,
    val distance: Double = 0.0,
    val duration: Double = 0.0,
    val avgSpeed: Double = 0.0,
    val maxSpeed: Double = 0.0,
    val climb: Double = 0.0,
    val descent: Double = 0.0,
    val calories: Double = 0.0,
    val routeImgUrl: String? = null,
    val timeStamp: Long = 0,
)


class WorkoutMapper {

    fun mapFirestorePojoToEntity(pojo: WorkoutFirestorePojo): Workout {
        return Workout(
            workoutId = pojo.workoutId,
            name = pojo.name,
            userId = pojo.userId,
            points = pojo.points?.map {
                WorkoutRoutePoint(
                    point = Point.fromLngLat(it.lng, it.lat, it.altitude),
                    distanceFromStart = it.distanceFromStart,
                    speed = it.speed,
                    timeStamp = it.timeStamp
                )
            },
            distance = pojo.distance,
            durationInSeconds = pojo.duration,
            climb = pojo.climb,
            descent = pojo.descent,
            routeImgUrl = pojo.routeImgUrl,
            avgSpeed = pojo.avgSpeed,
            maxSpeed = pojo.maxSpeed,
            calories = pojo.calories,
            timeStamp = pojo.timeStamp
        )
    }

    fun mapEntityToFirestorePojo(workout: Workout) = WorkoutFirestorePojo(
        workoutId = workout.workoutId,
        name = workout.name,
        userId = workout.userId,
        points = workout.points?.map {
            WorkoutFirestoreRoutePoint(
                lat = it.point.latitude(),
                lng = it.point.longitude(),
                altitude = it.point.altitude(),
                distanceFromStart = it.distanceFromStart,
                speed = it.speed,
                timeStamp = it.timeStamp
            )
        },
```

```
            distance = workout.distance,
            duration = workout.durationInSeconds,
            climb = workout.climb,
            descent = workout.descent,
            routeImgUrl = workout.routeImgUrl,
            avgSpeed = workout.avgSpeed,
            maxSpeed = workout.maxSpeed,
            calories = workout.calories,
            timeStamp = workout.timeStamp
        )
    }
```

ExportFileService.kt

```
abstract class ExportFileService {

    abstract suspend fun importRoutes(routesPath: String): Route

    abstract suspend fun saveRouteInInternalStorage(route: Route): Boolean

    abstract suspend fun saveWorkoutInInternalStorage(workout: Workout): Boolean
}
```

GpxFileService.kt

```
class GpxFileService @Inject constructor(
    @ApplicationContext private val context: android.content.Context,
) : ExportFileService() {

    override suspend fun importRoutes(routesPath: String): Route {
        TODO("Not yet implemented")
    }

    override suspend fun saveRouteInInternalStorage(route: Route): Boolean {
        val gpxFile = File(context.filesDir, "${route.name}.gpx")
        val header =
            "<?xml version=\"1.0\" encoding=\"UTF-8\" standalone=\"no\" ?>" +
                    "<gpx xmlns=\"http://www.topografix.com/GPX/1/1\" " +
                    "creator=\"MapSource 6.15.5\" version=\"1.1\" " +
                    "xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\"  " +
                    "xsi:schemaLocation=\"http://www.topografix.com/GPX/1/1 " +
                    "http://www.topografix.com/GPX/1/1/gpx.xsd\"><trk>\n"
        val name = "<name>${route.name}</name><trkseg>\n"
        val segments = StringBuilder()
        route.points.forEach { point ->
            segments.append("<trkpt lat=\"" + point.latitude() + "\" lon=\"" + point.longitude() +
"\">" + "\n<ele>" + point.altitude() + "</ele>\n" + "</trkpt>\n")
        }
        val footer = "</trkseg></trk></gpx>"
        return gpxFile.createFile(header, name, segments.toString(), footer)
    }

    override suspend fun saveWorkoutInInternalStorage(workout: Workout): Boolean {
        // create gpx file in external storage
        val gpxFile = File(context.filesDir, "${workout.name}.gpx")
        // create header
        val header =
            "<?xml version=\"1.0\" encoding=\"UTF-8\" standalone=\"no\" ?>" +
                    "<gpx xmlns=\"http://www.topografix.com/GPX/1/1\" " +
                    "creator=\"MapSource 6.15.5\" version=\"1.1\" " +
```

```
                    "xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\"  " +
                    "xsi:schemaLocation=\"http://www.topografix.com/GPX/1/1 " +
                    "http://www.topografix.com/GPX/1/1/gpx.xsd\"><trk>\n"
        // create name
        val name = "<name>${workout.name}</name><trkseg>\n"
        // create segments
        val segments = StringBuilder()
        // add all points
        workout.points?.forEach { point ->
            segments.append(
                // add point latitude and longitude
                "<trkpt lat=\"" + point.point.latitude() +
                        "\" lon=\"" + point.point.longitude() + "\">" +
                        // add altitude
                        "\n<ele>" + point.point.altitude() + "</ele>\n" +
                        // add time in yyyy-MM-dd'T'HH:mm:ss.SSS'Z' format
                        "<time>" + point.timeStamp.convertTimestampToTime() + "</time>" +
                        "</trkpt>\n"
            )
        }
        // create footer
        val footer = "</trkseg></trk></gpx>"
        // return gpx file
        return gpxFile.createFile(header, name, segments.toString(), footer)
    }
}
```

TcxFileService.kt

```
class TcxFileService @Inject constructor(
    @ApplicationContext private val context: Context,
) : ExportFileService() {
    val header = "<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n" +
            "<TrainingCenterDatabase
xsi:schemaLocation=\"http://www.garmin.com/xmlschemas/TrainingCenterDatabase/v2
http://www.garmin.com/xmlschemas/TrainingCenterDatabasev2.xsd\"
xmlns:ns5=\"http://www.garmin.com/xmlschemas/ActivityGoals/v1\"
xmlns:ns4=\"http://www.garmin.com/xmlschemas/ProfileExtension/v1\"
xmlns:ns3=\"http://www.garmin.com/xmlschemas/ActivityExtension/v2\"
xmlns:ns2=\"http://www.garmin.com/xmlschemas/UserProfile/v2\"
xmlns=\"http://www.garmin.com/xmlschemas/TrainingCenterDatabase/v2\"
xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\">"

    override suspend fun importRoutes(routesPath: String): Route {
        TODO("Not yet implemented")
    }

    override suspend fun saveRouteInInternalStorage(route: Route): Boolean {
        val tcxFile = File(context.filesDir, "${route.name}.tcx")
        val name = "<Course>\n<Name>${route.name}</Name>\n<Track>\n"
        val segments = StringBuilder()
        route.points.forEach { point ->
            segments.append(

"<Trackpoint>\n<Position>\n<LatitudeDegrees>${point.latitude()}</LatitudeDegrees>\n" +
                        "<LongitudeDegrees>${point.longitude()}</LongitudeDegrees>\n" +
                        "</Position>\n" +
                        "<AltitudeMeters>${point.altitude()}</AltitudeMeters>\n" +
                        "</Trackpoint>"
            )
        }
        val footer = "</Track>\n</Course></TrainingCenterDatabase>"
        return tcxFile.createFile(header, name, segments.toString(), footer)
    }
    override suspend fun saveWorkoutInInternalStorage(workout: Workout): Boolean {
        // create tcx file in external storage
```

```
        val tcxFile = File(context.filesDir, "${workout.name}.tcx")
        // create header
        val name = "<Activities>\n" +
                "<Activity Sport=\"Biking\">\n" +
                "<Id>${workout.timeStamp.convertTimestampToTime()}</Id>\n" +
                "<Lap StartTime=\"${workout.timeStamp.convertTimestampToTime()}\">\n" +
                "<TotalTimeSeconds>${workout.durationInSeconds}</TotalTimeSeconds>\n" +
                "<DistanceMeters>${workout.distance * 1000}</DistanceMeters>\n" +
                "<MaximumSpeed><Speed>${workout.maxSpeed}</Speed></MaximumSpeed>\n" +
                "<Calories>${workout.calories}</Calories>" +
                "<Track>\n"
        val segments = StringBuilder()
        // create segments
        workout.points?.forEach { point ->
            segments.append(
                "<Trackpoint>\n",
                "<Time>${point.timeStamp.convertTimestampToTime()}</Time>\n",
                "<Position>\n",
                "<LatitudeDegrees>${point.point.latitude()}</LatitudeDegrees>\n",
                "<LongitudeDegrees>${point.point.longitude()}</LongitudeDegrees>\n",
                "</Position>\n",
                "<AltitudeMeters>${point.point.altitude()}</AltitudeMeters>\n",
                "<DistanceMeters>${point.distanceFromStart}</DistanceMeters>\n",
                "<Extensions>\n",
                "<TPX xmlns=\"http://www.garmin.com/xmlschemas/ActivityExtension/v2\">\n",
                "<Speed>${point.speed}</Speed></TPX>\n",
                "</Extensions>\n",
                "</Trackpoint>"
            )
        }
        // create footer
        val footer = "</Track>\n" +
                "</Lap>\n" +
                "</Activity>\n" +
                "</Activities>\n" +
                "</TrainingCenterDatabase>"
        // write to file
        return tcxFile.createFile(header, name, segments.toString(), footer)
    }
}
```

TrackingUserWorkoutService.kt

```
@AndroidEntryPoint
class TrackingUserWorkoutService : Service() {

    private val scope = CoroutineScope(SupervisorJob() + Dispatchers.IO)

    private val binder = LocalBinder()

    @Inject
    lateinit var saveWorkoutInFirestoreUseCase: SaveWorkoutInFirestoreUseCase

    @Inject
    lateinit var collectUserLocationUseCase: CollectUserLocationUseCase

    @Inject
    lateinit var notificationManager: NotificationManager

    @Inject
    lateinit var notificationBuilder: NotificationCompat.Builder


    private lateinit var timer: Timer

    var preparedRoute by mutableStateOf<Route?>(null)
        private set
```

```
var points: List<WorkoutRoutePoint> = mutableStateListOf()
    private set

private var workoutDuration: Duration = Duration.ZERO

var hours by mutableStateOf("00")
    private set

var minutes by mutableStateOf("00")
    private set

var seconds by mutableStateOf("00")
    private set

var distance by mutableDoubleStateOf(0.00)
    private set

var speed by mutableDoubleStateOf(0.00)
    private set

var avgSpeed by mutableDoubleStateOf(0.00)
    private set

var climb by mutableDoubleStateOf(0.00)
    private set

var descent by mutableDoubleStateOf(0.00)
    private set

var maxSpeed by mutableDoubleStateOf(0.00)
    private set

var isTracking by mutableStateOf(false)
    private set

var isWorkoutStarted by mutableStateOf(false)
    private set

override fun onBind(p0: Intent?): IBinder = binder

override fun onStartCommand(intent: Intent?, flags: Int, startId: Int): Int {
    when (intent?.action) {
        TRACKING_ACTION_START -> startTracking()
        TRACKING_ACTION_STOP -> stopTracking()
        TRACKING_ACTION_PAUSE -> pauseTracking()
        TRACKING_ACTION_RESUME -> resumeTracking()
        TRACKING_ACTION_CREATE_PREPARED_ROUTE -> savedPreparedRoute(intent)
    }
    return super.onStartCommand(intent, flags, startId)
}

override fun onDestroy() {
    super.onDestroy()
    scope.cancel()
}

fun saveWorkout(name: String, onResult: (Throwable?) -> Unit) {
    val calories = calculateBurnedCalories(
        durationInMinutes = workoutDuration.inWholeMinutes.toDouble(),
        age = 20,
        weight = 70,
        isMale = true
    )
    val c1 = calculateBurnedCalories(
        durationInMinutes = workoutDuration.inWholeMinutes.toDouble(),
        age = 23,
        weight = 0,
        isMale = false
    )
```

```
            val workout = Workout(
                name = name,
                points = points,
                distance = distance,
                durationInSeconds = workoutDuration.inWholeSeconds.toDouble(),
                avgSpeed = avgSpeed,
                maxSpeed = speed,
                climb = climb,
                descent = descent,
                calories = calories,
                timeStamp = Date().time
            )
            saveWorkoutInFirestoreUseCase(workout, MapBoxStyle.STREET, onResult)
        }
        val avgHeartRate = 120
        // https://tourdevines.com.au/blog/how-many-calories-does-cycling-burn/
        private fun calculateBurnedCalories(
            durationInMinutes: Double,
            age: Int,
            weight: Int,
            isMale: Boolean,
        ): Double {
            return if (isMale) {
                // Calculate calories for men
                abs(((age * 0.2017) + (weight * 0.09036) - (avgHeartRate * 0.6309) - 55.0969) *
((durationInMinutes + 1) / 4.184))
            } else {
                // Calculate calories for women
                abs(((age * 0.074) + (weight * 0.05741) - (avgHeartRate * 0.4472) - 20.4022) *
((durationInMinutes + 1) / 4.184))
            }
        }

        private fun calculateAvgSpeed(points: List<WorkoutRoutePoint>): Double {
            val totalDistance = points.sumOf { it.speed }
            return totalDistance / points.size
        }

        private fun savedPreparedRoute(intent: Intent?) {
            val preparedRoute = intent?.parcelable<Route>(PREPARED_ROUTE_KEY)
            if (preparedRoute != null) {
                this.preparedRoute = preparedRoute
            }
        }

        private fun startTracking() {
            isTracking = true
            isWorkoutStarted = true
            collectUserLocationUseCase(TRACKING_INTERVAL).onEach { locationResult ->
                when (locationResult) {
                    is LocationServiceResult.Failure -> {
                        // TODO: handle error
                    }

                    is LocationServiceResult.Success -> {
                        if (isTracking) {
                            val location = locationResult.location
                            val lastPoint = points.lastOrNull()
                            if (lastPoint != null) {
                                val distanceFromLastPoint =
                                    TurfMeasurement.distance(location.toPoint(), lastPoint.point)
                                distance += distanceFromLastPoint.roundTo2()
                                val diff = location.altitude - lastPoint.point.altitude()
                                if (diff < 0) {
                                    descent += -diff
                                } else {
                                    climb += diff
                                }
                            }
                            val parsedSpeed =
```

```
                            location.speed.convertMetersPerSecondToKilometersPerHour()
                                .toDouble().roundTo2()
                        speed = parsedSpeed
                        if (speed > maxSpeed) {
                            maxSpeed = speed
                        }
                        avgSpeed = calculateAvgSpeed(points).roundTo2()
                        points = points + WorkoutRoutePoint(
                            point = location.toPoint(),
                            distanceFromStart = distance,
                            speed = parsedSpeed,
                            timeStamp = locationResult.location.time
                        )
                    }
                }
            }
        }.launchIn(scope)

        startTimer()
        startNotificationChannel()
        startForeground(NOTIFICATION_ID, notificationBuilder.build())
    }

    private fun pauseTracking() {
        isTracking = false
        speed = 0.0
    }

    private fun resumeTracking() {
        isTracking = true
    }

    private fun stopTracking() {
        isWorkoutStarted = false
        isTracking = false
        stopTimer()
        notificationManager.cancel(NOTIFICATION_ID)
        stopForeground(STOP_FOREGROUND_DETACH)
        stopSelf()
    }

    private fun startTimer() {
        timer = fixedRateTimer(initialDelay = TRACKING_INTERVAL, period = TRACKING_INTERVAL) {
            if (isTracking) {
                workoutDuration = workoutDuration.plus(1.seconds)
                updateWorkoutDuration()
                updateNotification()
            }
        }
    }

    private fun stopTimer() {
        if (this::timer.isInitialized) timer.cancel()
    }

    private fun updateWorkoutDuration() {
        workoutDuration.toComponents { hours, minutes, seconds, _ ->
            this.hours = hours.padTimerValue()
            this.minutes = minutes.padTimerValue()
            this.seconds = seconds.padTimerValue()
        }
    }

    private fun updateNotification() {
        notificationManager.notify(
            NOTIFICATION_ID,
            notificationBuilder.setContentText("$hours:$minutes:$seconds").build()
        )
    }
```

```
    private fun startNotificationChannel() {
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
            val channel = NotificationChannel(
                CHANNEL_ID,
                NOTIFICATION_CHANNEL_NAME,
                NotificationManager.IMPORTANCE_DEFAULT
            )
            notificationManager.createNotificationChannel(channel)
        }
    }

    companion object {
        private const val TRACKING_ACTION_START = "start"
        private const val TRACKING_ACTION_STOP = "stop"
        private const val TRACKING_ACTION_PAUSE = "pause"
        private const val TRACKING_ACTION_RESUME = "resume"
        private const val TRACKING_ACTION_CREATE_PREPARED_ROUTE = "create_prepared_route"
        private const val PREPARED_ROUTE_KEY = "prepared_route"

        const val CHANNEL_ID = "tracking_channel"

        private const val NOTIFICATION_ID = 1
        private const val NOTIFICATION_CHANNEL_NAME = "tracking_channel"
        private const val TRACKING_INTERVAL = 1000L

        fun createStartIntent(context: Context): Intent {
            val intent = Intent(context, TrackingUserWorkoutService::class.java)
            intent.action = TRACKING_ACTION_START
            return intent
        }

        fun createToggleIntent(isTracking: Boolean, context: Context): Intent {
            val intent = Intent(context, TrackingUserWorkoutService::class.java)
            val action = if (isTracking) TRACKING_ACTION_PAUSE else TRACKING_ACTION_RESUME
            intent.action = action
            return intent
        }

        fun createRouteIntent(context: Context, route: Route): Intent {
            val intent = Intent(context, TrackingUserWorkoutService::class.java)
            intent.action = TRACKING_ACTION_CREATE_PREPARED_ROUTE
            intent.putExtra(PREPARED_ROUTE_KEY, route)
            return intent
        }

        fun createStopIntent(context: Context): Intent {
            val intent = Intent(context, TrackingUserWorkoutService::class.java)
            intent.action = TRACKING_ACTION_STOP
            return intent
        }
    }

    inner class LocalBinder : Binder() {
        fun getService(): TrackingUserWorkoutService = this@TrackingUserWorkoutService
    }
}
```

UserLocationRepositoryImpl.kt

```
class UserLocationRepositoryImpl @Inject constructor(
    @ApplicationContext private val context: Context,
) : UserLocationRepository {
    @SuppressLint("MissingPermission")
    override fun collectUserLocation(interval: Long?): Flow<LocationServiceResult> = callbackFlow
{
        if (!context.isLocationPermissionGranted()) {

trySend(LocationServiceResult.Failure(LocationServiceResult.FailureReason.PERMISSION_IS_DENIED))
            close()
```

```
            return@callbackFlow
        }
        val locationManager =
            context.getSystemService(Context.LOCATION_SERVICE) as android.location.LocationManager
        val isGpsEnabled =
            locationManager.isProviderEnabled(android.location.LocationManager.GPS_PROVIDER)
        val isNetworkEnabled =
            locationManager.isProviderEnabled(android.location.LocationManager.NETWORK_PROVIDER)
        if (!isGpsEnabled && !isNetworkEnabled) {

trySend(LocationServiceResult.Failure(LocationServiceResult.FailureReason.LOCATION_IS_DISABLED))
        }
        val locationService: LocationService = LocationServiceFactory.getOrCreate()
        var locationProvider: DeviceLocationProvider? = null
        val request = LocationProviderRequest.Builder()
            .interval(
                IntervalSettings.Builder()
                    .interval(interval ?: DEFAULT_INTERVAL)
                    .minimumInterval(MINIMUM_INTERVAL)
                    .maximumInterval(MAX_INTERVAL)
                    .build()
            )
            .displacement(DISPLACEMENT)
            .accuracy(AccuracyLevel.HIGH)
            .build()

        val result = locationService.getDeviceLocationProvider(request)
        if (result.isValue) {
            locationProvider = result.value
        } else {

trySend(LocationServiceResult.Failure(LocationServiceResult.FailureReason.LOCATION_IS_DISABLED))
        }
        val locationObserver =
            LocationObserver { locations ->
                launch { trySend(LocationServiceResult.Success(locations[0].toAndroidLocation())) }
            }
        locationProvider?.addLocationObserver(locationObserver)

        awaitClose {
            locationProvider?.removeLocationObserver(locationObserver)
        }
    }

    companion object {
        private const val DEFAULT_INTERVAL: Long = 2000
        private const val MINIMUM_INTERVAL: Long = 1000
        private const val MAX_INTERVAL: Long = 5000
        private const val DISPLACEMENT: Float = 5F
    }
}
```

MapboxApiRepositoryImpl.kt

```
interface MapboxApiRepository {

    suspend fun getRoute(coordinates: List<Point>): Response<Route?>

    fun getStaticMapUrl(
        startPoint: Point?,
        endPoint: Point?,
        geometry: String?,
        style: MapBoxStyle,
    ): URL

}
```

```
class MapboxApiRepositoryImpl @Inject constructor(
    @MapboxPublicToken private val publicToken: String,
) : MapboxApiRepository {

    override suspend fun getRoute(coordinates: List<Point>): Response<Route?> {
        return suspendCoroutine { continuation ->
            // Get retrofit client with custom parameters
            val client = MapboxDirections.builder()
                // Mapbox access token
                .accessToken(publicToken)
                .routeOptions(
                    RouteOptions.builder()
                        // Route response format
                        .geometries(DirectionsCriteria.GEOMETRY_POLYLINE)
                        // User coordinates
                        .coordinates(coordinates.toDirectionsString())
                        // Set cycling profile
                        .profile(DirectionsCriteria.PROFILE_CYCLING)
                        // Get response in full overview
                        .overview(DirectionsCriteria.OVERVIEW_FULL)
                        .build()
                ).build()
            // Execute request
            client?.enqueueCall(object : retrofit2.Callback<DirectionsResponse> {
                override fun onResponse(
                    call: Call<DirectionsResponse>,
                    response: retrofit2.Response<DirectionsResponse>,
                ) {
                    // Check response
                    if (response.body() == null || ((response.body()?.routes()?.size ?: 0) < 1)) {
                        continuation.resume(Failure("Response is empty"))
                        return
                    }
                    // Get route
                    if (response.isSuccessful) {
                        val route = response.body()?.routes()?.get(0)
                        val distance = route?.distance() ?: 0.0
                        val duration = route?.duration() ?: 0.0
                        val polylineWithoutElevation =
                            route?.geometry()?.getCoordinates() ?: emptyList()
                        val geometry = route?.geometry()
                        val mapRoute = Route(
                            distance = distance,
                            duration = duration,
                            points = polylineWithoutElevation,
                            geometry = geometry
                        )
                        route?.let {
                            continuation.resume(Success(mapRoute))
                        } ?: kotlin.run {
                            continuation.resume(Failure("Response is not successful"))
                        }
                    }

                }
                // on failure
                override fun onFailure(call: Call<DirectionsResponse>, t: Throwable) {
                    continuation.resume(Failure(t.localizedMessage ?: "Unknown error"))
                }
            })
        }
    }

    override fun getStaticMapUrl(
        startPoint: Point?,
        endPoint: Point?,
        geometry: String?,
        style: MapBoxStyle,
```

```
    ): URL {
        val styleId = when (style) {
            MapBoxStyle.STREET -> StaticMapCriteria.STREET_STYLE
            MapBoxStyle.SATELLITE -> StaticMapCriteria.SATELLITE_STYLE
            MapBoxStyle.LIGHT -> StaticMapCriteria.LIGHT_STYLE
            MapBoxStyle.DARK -> StaticMapCriteria.DARK_STYLE
        }
        val staticImage = MapboxStaticMap.builder()
            .accessToken(publicToken)
            .styleId(styleId)
            .staticMarkerAnnotations(
                listOf(
                    StaticMarkerAnnotation
                        .builder()
                        .label(FIRST_ROUTE_POINT_MARKER_LABEL)
                        .color(ROUTE_LINE_COLOR)
                        .lnglat(startPoint)
                        .build(),
                    StaticMarkerAnnotation
                        .builder()
                        .label(LAST_ROUTE_POINT_MARKER_LABEL)
                        .color(ROUTE_LINE_COLOR)
                        .lnglat(endPoint)
                        .build(),
                )
            )
            .staticPolylineAnnotations(
                listOf(
                    geometry?.let {
                        StaticPolylineAnnotation.builder().fillColor("00246b").strokeWidth(5.0)
                            .polyline(geometry)
                            .build()
                    }
                )
            )
            .cameraAuto(true)
            .width(STATIC_MAP_SIZE)
            .height(STATIC_MAP_SIZE)
            .retina(true)
            .build()
        return staticImage.url().toUrl()
    }

    companion object {
        const val FIRST_ROUTE_POINT_MARKER_LABEL = "a"
        const val LAST_ROUTE_POINT_MARKER_LABEL = "b"
        const val ROUTE_LINE_COLOR = "cadcfc"
        const val STATIC_MAP_SIZE = 420
    }
}
```

RouteBulderScreen.kt

```
@OptIn(MapboxExperimental::class, ExperimentalMaterial3Api::class, ExperimentalLayoutApi::class)
@Composable
fun RouteBuilderScreen(
    snackbarHostState: SnackbarHostState,
    uiState: RouteBuilderState,
    onEvent: (RouteBuilderScreenEvent) -> Unit,
) {
    val context = LocalContext.current
    var shouldShowLocationDisableDialog by remember { mutableStateOf(false) }
    var showDropdownStyleMenu by remember { mutableStateOf(false) }
    var userPointsListOpenedState by remember { mutableStateOf(UserPointsListState.NOT_OPENED_YET)
}
    val newPointSheetState = rememberModalBottomSheetState()
    var showDropdownActionsMenu by remember { mutableStateOf(false) }
    var showSaveRouteDialog by remember { mutableStateOf(false) }
```

```
val drawable = remember {
    ResourcesCompat.getDrawable(
        context.resources,
        R.drawable.ic_checkpoint,
        null
    )
}
val bitmap = remember {
    drawable?.toBitmap(
        drawable.intrinsicWidth,
        drawable.intrinsicHeight,
        Bitmap.Config.ARGB_8888
    )
}

val mapViewportState = rememberMapViewportState {
    setCameraOptions {
        center(Point.fromLngLat(0.0, 0.0))
        zoom(0.0)
        pitch(0.0)
    }
    MapAnimationOptions.mapAnimationOptions {
        duration(3000)
    }
}

val permissionRequest = rememberLauncherForActivityResult(
    contract = ActivityResultContracts.RequestMultiplePermissions(),
    onResult = { permissions ->
        if (!permissions.values.all { it }) {
            shouldShowLocationDisableDialog = true
        } else {
            mapViewportState.flyToUserPosition()
        }
    }
)

val mapBoxUiSettings: GesturesSettings by remember {
    mutableStateOf(GesturesSettings {
        rotateEnabled = true
        pinchToZoomEnabled = true
        pitchEnabled = true
    })
}

val compassSettings: CompassSettings by remember {
    mutableStateOf(CompassSettings {
        enabled = true
    })
}

val attributionSettings: AttributionSettings by remember {
    mutableStateOf(AttributionSettings {
        enabled = false
    })
}

val scaleBarSetting: ScaleBarSettings by remember {
    mutableStateOf(ScaleBarSettings {
        enabled = true
    })
}

LaunchedEffect(Unit) {
    permissionRequest.launch(MapUtils.locationPermissions)
}

LaunchedEffect(uiState.lastSelectedPoint) {
    uiState.lastSelectedPoint?.let { point ->
```

```
                mapViewportState.flyTo(
                    cameraOptions {
                        center(point)
                    },
                )
            }
        }

        LaunchedEffect(uiState.userPoints.size) {
            if (userPointsListOpenedState == UserPointsListState.NOT_OPENED_YET &&
uiState.userPoints.size > 1) {
                userPointsListOpenedState = UserPointsListState.OPENED
            }
        }
        Scaffold(
            snackbarHost = { SnackbarHost(hostState = snackbarHostState) },
            topBar = {
                TopAppBar(
                    title = {
                        Text(text = stringResource(id = R.string.home))
                    },
                    actions = {
                        Box {
                            IconButton(onClick = {
                                showDropdownActionsMenu = !showDropdownActionsMenu
                            }) {
                                Icon(
                                    imageVector = Icons.Default.MoreVert,
                                    contentDescription = stringResource(
                                        R.string.more
                                    )
                                )
                            }
                            RouteBuilderDropDownMenu(
                                showDropdownActionsMenu = showDropdownActionsMenu,
                                onShowDropdownActionsMenuChange = { showDropdownActionsMenu = it },
                            ) {
                                when (it) {
                                    RouteBuilderDropDownMenuState.MoveCameraToCurrentGeometry -> {
                                        uiState.route?.points?.let { points ->
                                            mapViewportState.transitionToGeometry(points)
                                        }
                                    }

                                    RouteBuilderDropDownMenuState.OpenUserPointsList -> {
                                        userPointsListOpenedState = UserPointsListState.OPENED
                                    }

                                    RouteBuilderDropDownMenuState.SaveRoute -> {
                                        uiState.route?.let {
                                            showSaveRouteDialog = true
                                        } ?: kotlin.run {
                                            onEvent(RouteBuilderScreenEvent.ShowSnackbar("Route is
empty"))

                                        }
                                    }

                                    RouteBuilderDropDownMenuState.SetFirstPointAsFinish -> {
                                        onEvent(RouteBuilderScreenEvent.SetFirstPointAsDestination)
                                    }
                                }
                            }
                        }
                    }
                )
            },
            contentWindowInsets =
ScaffoldDefaults.contentWindowInsets.exclude(NavigationBarDefaults.windowInsets)
        ) { innerPadding ->
            ConstraintLayout(
```

```
            modifier = Modifier
                .fillMaxSize()
                .padding(innerPadding)
        ) {
            SnackbarMessageHandler(
                snackbarMessage = uiState.snackbarMessage,
                onDismissSnackbar = { onEvent(RouteBuilderScreenEvent.DismissSnackbar) },
            )
            val (pointsListRef, mapRef, routeInfoRef) = createRefs()
            MapboxMap(
                modifier = Modifier.constrainAs(mapRef) {
                    width = Dimension.matchParent
                    height = Dimension.matchParent
                },
                mapInitOptionsFactory = { context ->
                    MapInitOptions(
                        context = context,
                        styleUri = MapBoxStyle.STREET.style,
                    )
                },
                compassSettings = compassSettings,
                mapViewportState = mapViewportState,
                scaleBarSettings = scaleBarSetting,
                gesturesSettings = mapBoxUiSettings,
                attributionSettings = attributionSettings,
                onMapLongClickListener = { point ->
                    onEvent(RouteBuilderScreenEvent.ChangeLastSelectedPoint(point))
                    true
                }
            ) {
                MapEffect(Unit) { mapView ->
                    with(mapView) {
                        location.locationPuck = createDefault2DPuck(withBearing = true)
                        location.enabled = true
                    }
                }
                MapEffect(uiState.currentMapStyle) { mapView ->
                    with(mapView) {
                        if (mapboxMap.isValid()) {
                            mapboxMap.loadStyle(uiState.currentMapStyle.style)
                        }
                    }
                }
                bitmap?.let { it1 ->
                    PointAnnotationGroup(
                        annotations = uiState.userPoints.map {
                            PointAnnotationOptions()
                                .withPoint(it.point)
                                .withIconImage(it1)
                                .withIconSize(0.75)
                                .withTextSize(20.0)
                                .withTextColor(Color.WHITE)
                                .withTextField(it.name)
                        }
                    )
                }
                bitmap?.let { it1 ->
                    PointAnnotationGroup(
                        annotations = uiState.userPoints.map {
                            PointAnnotationOptions()
                                .withPoint(it.point)
                                .withIconImage(it1)
                                .withIconSize(0.75)
                                .withTextSize(20.0)
                                .withTextColor(Color.WHITE)
                                .withTextField(it.name)
                        },
                        annotationConfig = AnnotationConfig(
                            layerId = "point-layer",
                        )
```

```
                        )
                        uiState.lastSelectedPoint?.let {
                            PointAnnotationGroup(
                                annotations = listOf(
                                    PointAnnotationOptions()
                                        .withPoint(it)
                                        .withIconImage(it1)
                                        .withIconSize(0.75)
                                        .withTextSize(20.0)
                                        .withTextColor(Color.WHITE)

.withTextField(uiState.pointAlphabetIndex.getAlphabetLetterByIndex())
                                )
                            )
                        }

                    }
                    uiState.route?.points?.let {
                        PolylineAnnotationGroup(
                            annotations = listOf(
                                PolylineAnnotationOptions()
                                    .withPoints(points = it)
                                    .withLineBorderWidth(2.0)
                                    .withLineBorderColor(context.getColor(R.color.light_blue))
                                    .withLineWidth(7.0)
                                    .withLineColor(context.getColor(R.color.dark_blue))
                            ),
                            annotationConfig = AnnotationConfig(
                                belowLayerId = "point-layer",
                                layerId = "route-layer",
                            ),
                            lineCap = LineCap.ROUND
                        )
                    }
                }
                AnimatedVisibility(
                    visible = userPointsListOpenedState.isOpened(),
                    modifier = Modifier.constrainAs(pointsListRef) {
                        top.linkTo(parent.top)
                        width = Dimension.matchParent
                    }) {
                    RouteInfoPanel(
                        points = uiState.userPoints,
                        onMove = { from, to ->
                            onEvent(RouteBuilderScreenEvent.OnPointIndexChanged(from, to))
                        },
                        onPointClick = {
                            uiState.userPoints[it].let {
                                mapViewportState.flyTo(
                                    cameraOptions {
                                        center(it.point)
                                    }
                                )
                            }
                        },
                        onPointDeleteClick = {
                            onEvent(RouteBuilderScreenEvent.OnPointDeleteClick(it))
                        },
                        onListClose = {
                            userPointsListOpenedState = UserPointsListState.CLOSED
                        }
                    )
                }
                AnimatedVisibility(
                    visible = uiState.isRouteLoading,
                    modifier = Modifier
                        .padding(8.dp)
                        .constrainAs(createRef()) {
                            centerHorizontallyTo(parent)
                            top.linkTo(parent.top)
```

```
                width = Dimension.fillToConstraints
            },
        enter = slideInVertically(),
        exit = slideOutVertically()
    ) {
        LinearProgressIndicator()
    }
    Row(
        modifier = Modifier
            .constrainAs(routeInfoRef) {
                bottom.linkTo(parent.bottom)
                width = Dimension.matchParent
                height = Dimension.preferredWrapContent.atMost(36.dp)
            }
            .background(
                color = MaterialTheme.colorScheme.surface.copy(0.7f)
            ),
        verticalAlignment = Alignment.CenterVertically,
        horizontalArrangement = Arrangement.Absolute.Center
    ) {
        RouteMetrics(uiState.route)
    }
    Column(
        modifier = Modifier
            .background(
                color = MaterialTheme.colorScheme.surface.copy(0.7f),
                shape = MaterialTheme.shapes.medium
            )
            .padding(8.dp)
            .constrainAs(createRef()) {
                bottom.linkTo(routeInfoRef.top)
                end.linkTo(parent.end)
            },
        verticalArrangement = Arrangement.spacedBy(4.dp)
    ) {
        Box {
            IconButton(onClick = { showDropdownStyleMenu = true }) {
                Icon(
                    Icons.Default.Layers,
                    contentDescription = stringResource(R.string.map_layers)
                )
            }
            DropdownMenu(
                expanded = showDropdownStyleMenu,
                onDismissRequest = { showDropdownStyleMenu = false }) {
                MapBoxStyle.entries.forEach {
                    DropdownMenuItem(
                        modifier = Modifier.padding(4.dp),
                        text = {
                            Row(verticalAlignment = Alignment.CenterVertically) {
                                Image(
                                    painter = painterResource(id = it.resId),
                                    contentDescription = stringResource(id = it.textId)
                                )
                                Spacer(modifier = Modifier.width(4.dp))
                                Text(text = stringResource(id = it.textId))
                            }
                        }, onClick = {
                            onEvent(RouteBuilderScreenEvent.ChangeRouteBuilderStyle(it))
                            showDropdownStyleMenu = false
                        })
                    HorizontalDivider()
                }
            }
        }
        IconButton(onClick = {
            mapViewportState.flyToUserPosition()
        }) {
            Icon(
                Icons.Default.MyLocation,
```

```
                            contentDescription = stringResource(R.string.my_location)
                        )
                    }
                }
                if (uiState.isNewPointDialogOpened) {
                    ModalBottomSheet(
                        sheetState = newPointSheetState,
                        onDismissRequest = {
                            onEvent(RouteBuilderScreenEvent.DeleteLastSelectedPoint)
                        },
                    ) {
                        NewPointView(
                            latitude = uiState.lastSelectedPoint?.latitude(),
                            longitude = uiState.lastSelectedPoint?.longitude(),
                            isSetAsDestinationBtnVisible = uiState.userPoints.isEmpty() &&
uiState.userLocationPoint != null,
                            setUserLocationAsStart = {
onEvent(RouteBuilderScreenEvent.SetUserLocationAsStart) },
                            setLastPointAsDestination = {
onEvent(RouteBuilderScreenEvent.SetLastPointAsDestination) },
                            setLastSelectedPointAsStart = {
onEvent(RouteBuilderScreenEvent.SetLastSelectedPointAsStart) }
                        )
                        Spacer(
                            Modifier.windowInsetsBottomHeight(
                                WindowInsets.navigationBarsIgnoringVisibility
                            )
                        )
                    }
                }
            }

            if (shouldShowLocationDisableDialog) {
                LocationDisableAlertDialog(
                    onDialogDismiss = { shouldShowLocationDisableDialog = false },
                    onRequestLocation = {
                        shouldShowLocationDisableDialog = false
                        permissionRequest.launch(MapUtils.locationPermissions)
                    }
                )
            }
            if (showSaveRouteDialog) {
                SaveUserRouteAlertDialog(
                    onSave = { routeName, routeDescription ->
                        onEvent(
                            RouteBuilderScreenEvent.SaveRoute(
                                routeName,
                                routeDescription
                            )
                        )
                        showSaveRouteDialog = false
                    },
                    onCancel = { showSaveRouteDialog = false }
                )
            }
        }
    }

    sealed class RouteBuilderScreenEvent {
        data object SetLastSelectedPointAsStart : RouteBuilderScreenEvent()
        data object SetUserLocationAsStart : RouteBuilderScreenEvent()
        data object SetLastPointAsDestination : RouteBuilderScreenEvent()
        data object SetFirstPointAsDestination : RouteBuilderScreenEvent()
        data class ChangeLastSelectedPoint(val point: Point) : RouteBuilderScreenEvent()
        data object DeleteLastSelectedPoint : RouteBuilderScreenEvent()
        data class ChangeRouteBuilderStyle(val style: MapBoxStyle) : RouteBuilderScreenEvent()
        data class OnPointIndexChanged(val from: Int, val to: Int) : RouteBuilderScreenEvent()
        data class OnPointDeleteClick(val index: Int) : RouteBuilderScreenEvent()
        data class SaveRoute(val routeName: String, val routeDescription: String) :
            RouteBuilderScreenEvent()
```

```
    data object DismissSnackbar : RouteBuilderScreenEvent()
    data class ShowSnackbar(val message: String) : RouteBuilderScreenEvent()
}


@HiltViewModel
class RouteBuilderScreenViewModel @Inject constructor(
    private val collectUserLocationUseCase: CollectUserLocationUseCase,
    private val getRouteFromCoordinatesUseCase: GetRouteFromCoordinatesUseCase,
    private val saveRouteInFirestoreUseCase: SaveRouteInFirestoreUseCase,
) : ViewModel() {

    var state by mutableStateOf(RouteBuilderState())
        private set

    init {
        collectUserLocation()
    }

    private fun collectUserLocation() {
        viewModelScope.launch {
            collectUserLocationUseCase().collect { userLocation ->
                state = when (userLocation) {
                    is LocationServiceResult.Failure -> {
                        val message = when (userLocation.reason) {
                            LocationServiceResult.FailureReason.PERMISSION_IS_DENIED -> {
                                R.string.location_permission_denied
                            }

                            LocationServiceResult.FailureReason.LOCATION_IS_DISABLED -> {
                                R.string.location_disabled
                            }
                        }
                        state.copy(snackbarMessage =
SnackbarMessage.from(UserMessage.from(message)))
                    }

                    is LocationServiceResult.Success -> {
                        state.copy(userLocationPoint = userLocation.location.toPoint())
                    }
                }
            }
        }
    }

    fun changeLastSelectedPoint(point: Point) {
        state = state.copy(lastSelectedPoint = point, isNewPointDialogOpened = true)
    }

    fun setLastSelectedPointAsDestination() {
        state.lastSelectedPoint?.let { addPoint(it) }
    }

    fun setLastSelectedPointAsStart() {
        clearRoute()
        clearUserPoints()
        state.lastSelectedPoint?.let { addPoint(it) }
    }

    fun setUserLocationAsStart() {
        state.lastSelectedPoint?.let {
            addPoint(state.userLocationPoint ?: it)
            addPoint(it)
        }
    }

    private fun addPoint(point: Point) {
        state = state.copy(
            userPoints = state.userPoints + MapPoint(
                state.pointAlphabetIndex.getAlphabetLetterByIndex(),
```

```
                    point
                ),
                pointAlphabetIndex = state.pointAlphabetIndex + 1
            )
            deleteLastSelectedPoint()
            getRoute()
        }

        private fun clearUserPoints() {
            state = state.copy(userPoints = emptyList())
        }

        fun deleteLastSelectedPoint() {
            viewModelScope.launch {
                state = state.copy(isNewPointDialogOpened = false)
                // delay for animation
                delay(NEW_POINT_DIALOG_ANIMATION_DURATION)
                state = state.copy(lastSelectedPoint = null)
            }
        }

        private fun clearRoute(isEmpty: Boolean = true) {
            state = state.copy(
                route = null,
                userPoints = emptyList(),
                pointAlphabetIndex = if (isEmpty) 0 else state.pointAlphabetIndex
            )
        }

        private fun getRoute() {
            if (state.userPoints.size < 2) {
                return
            }
            state = state.copy(isRouteLoading = true)
            viewModelScope.launch {
                val points = state.userPoints.map { it.point }
                getRouteFromCoordinatesUseCase(points).let { response ->
                    state = state.copy(isRouteLoading = false)
                    state = when (response) {
                        is Response.Failure -> {
                            state.copy(snackbarMessage =
    SnackbarMessage.from(UserMessage.from(response.message)))
                        }

                        is Response.Success -> {
                            state.copy(route = response.data)
                        }

                        Response.Loading -> {
                            state
                        }
                    }
                }
            }
        }

        fun changeMapStyle(style: MapBoxStyle) {
            state = state.copy(currentMapStyle = style)
        }

        fun deletePoint(index: Int) {
            state = state.copy(userPoints = state.userPoints - state.userPoints[index])
            if (state.userPoints.size < 2) {
                clearRoute(state.userPoints.isEmpty())
            } else {
                getRoute()
            }
        }

        fun changePointByIndex(from: Int, to: Int) {
```

```
            state = state.copy(
                userPoints = state.userPoints.toMutableList().move(from, to)
            )
            getRoute()
        }

        fun setFirstSelectedPointAsDestination() {
            state.userPoints.getOrNull(0)?.let { addPoint(it.point) }
        }

        fun saveRoute(routeName: String, routeDescription: String) {
            viewModelScope.launch {
                state = state.copy(isRouteLoading = true)
                saveRouteAttributes(routeName, routeDescription)
                state.route?.let {
                    saveRouteInFirestoreUseCase(it, state.currentMapStyle) { error ->
                        state = state.copy(isRouteLoading = false)
                        if (error == null) {
                            state = state.copy(
                                snackbarMessage = SnackbarMessage.from(
                                    UserMessage.from(R.string.route_saved)
                                )
                            )
                            clearRoute()
                        } else {
                            state = state.copy(
                                snackbarMessage = SnackbarMessage.from(
                                    UserMessage.from(R.string.route_saving_error),
                                )
                            )
                        }

                    }
                }
            }
        }

        private fun saveRouteAttributes(routeName: String, routeDescription: String) {
            state =
                state.copy(
                    route = state.route?.copy(
                        name = routeName,
                        description = routeDescription,
                        timeStamp = Date().time
                    ),
                )
        }

        fun dismissSnackbar() {
            state = state.copy(snackbarMessage = null)
        }

        companion object {
            private const val NEW_POINT_DIALOG_ANIMATION_DURATION = 300L
        }
    }


    data class RouteBuilderState(
        val currentMapStyle: MapBoxStyle = MapBoxStyle.STREET,
        val userLocationPoint: Point? = null,
        val lastSelectedPoint: Point? = null,
        val pointAlphabetIndex: Int = 0,
        val userPoints: List<MapPoint> = emptyList(),
        val route: Route? = null,
        val isRouteLoading: Boolean = false,
        val isNewPointDialogOpened: Boolean = false,
        val snackbarMessage: SnackbarMessage? = null,
    )
```

```
@Composable
fun RouteBuilderCompleteScreen(
    snackbarHostState: SnackbarHostState,
    snackbarController: SnackbarController = LocalSnackbarController.current,
) {
    val viewModel: RouteBuilderScreenViewModel = hiltViewModel<RouteBuilderScreenViewModel>()
    RouteBuilderScreen(snackbarHostState, viewModel.state) { event ->
        when (event) {
            RouteBuilderScreenEvent.SetLastPointAsDestination -> {
                viewModel.setLastSelectedPointAsDestination()
            }

            RouteBuilderScreenEvent.SetLastSelectedPointAsStart -> {
                viewModel.setLastSelectedPointAsStart()
            }

            is RouteBuilderScreenEvent.ChangeLastSelectedPoint -> {
                viewModel.changeLastSelectedPoint(event.point)
            }

            RouteBuilderScreenEvent.DeleteLastSelectedPoint -> {
                viewModel.deleteLastSelectedPoint()
            }

            RouteBuilderScreenEvent.SetUserLocationAsStart -> {
                viewModel.setUserLocationAsStart()
            }

            is RouteBuilderScreenEvent.ChangeRouteBuilderStyle -> {
                viewModel.changeMapStyle(event.style)
            }

            is RouteBuilderScreenEvent.OnPointDeleteClick -> {
                viewModel.deletePoint(event.index)
            }

            is RouteBuilderScreenEvent.OnPointIndexChanged -> {
                viewModel.changePointByIndex(event.from, event.to)
            }

            RouteBuilderScreenEvent.SetFirstPointAsDestination -> {
                viewModel.setFirstSelectedPointAsDestination()
            }

            is RouteBuilderScreenEvent.SaveRoute -> {
                viewModel.saveRoute(event.routeName, event.routeDescription)
            }

            RouteBuilderScreenEvent.DismissSnackbar -> {
                viewModel.dismissSnackbar()
            }

            is RouteBuilderScreenEvent.ShowSnackbar -> {
                snackbarController.showMessage(event.message)
            }
        }
    }
}
```

WorkoutScreen.kt

```
@OptIn(ExperimentalMaterial3Api::class, ExperimentalLayoutApi::class)
@Composable
fun WorkoutsScreen(
    snackbarHostState: SnackbarHostState,
    snackbarController: SnackbarController = LocalSnackbarController.current,
    navToWorkoutPage: (String) -> Unit,
```

```
    navToWorkout: () -> Unit
) {
    val viewModel = hiltViewModel<WorkoutsScreenViewModel>()
    var search by rememberSaveable {
        mutableStateOf("")
    }
    var isActiveSearch by rememberSaveable { mutableStateOf(false) }
    var searchWorkoutParams by remember { mutableStateOf(SearchWorkoutParams()) }
    val searchDirectionSheetState = rememberModalBottomSheetState()
    val searchTypeSheetState = rememberModalBottomSheetState()
    var isSearchDirectionWindowOpen by remember { mutableStateOf(false) }
    var isSearchTypeWindowOpen by remember { mutableStateOf(false) }

    Scaffold(
        snackbarHost = { SnackbarHost(hostState = snackbarHostState) },
        topBar = {
            TopAppBar(
                title = {
                    Text(text = stringResource(R.string.workouts))
                },
                actions = {

                }
            )
        },
        floatingActionButton = {
            FloatingActionButton(onClick = { navToWorkout() }) {
                Icon(Icons.Default.PlayArrow, contentDescription = "Start workout")
            }
        },
        contentWindowInsets =
ScaffoldDefaults.contentWindowInsets.exclude(NavigationBarDefaults.windowInsets)
    ) { innerPadding ->
        LaunchedEffect(searchWorkoutParams) {
            viewModel.getWorkoutList(searchWorkoutParams)
        }
        ConstraintLayout(
            modifier = Modifier
                .fillMaxSize()
                .padding(innerPadding)
        ) {
            val (searchBarRef, searchSettingsRef) = createRefs()
            SearchBar(
                modifier = Modifier.padding(8.dp).constrainAs(searchBarRef) {
                    top.linkTo(parent.top)
                    centerHorizontallyTo(parent)
                    width = Dimension.fillToConstraints
                },
                query = search,
                onQueryChange = { search = it },
                onSearch = {
                    isActiveSearch = false
                    searchWorkoutParams = searchWorkoutParams.copy(searchText = search)
                },
                active = isActiveSearch,
                onActiveChange = {
                    isActiveSearch = it
                },
                placeholder = {
                    Text(text = "Search")
                },
                leadingIcon = {
                    Icon(
                        imageVector = Icons.Default.Search,
                        contentDescription = null
                    )
```

```
        },
        trailingIcon = {
            if (isActiveSearch) {
                Icon(
                    imageVector = Icons.Default.Close,
                    contentDescription = null,
                    modifier = Modifier.clickable {
                        isActiveSearch = false; search = "";
                        searchWorkoutParams = searchWorkoutParams.copy(
                            searchText = ""
                        )
                    }
                )
            }
        },
        colors = androidx.compose.material3.SearchBarDefaults.colors()
    ) {

    }
    Row(modifier = Modifier
        .constrainAs(searchSettingsRef) {
            top.linkTo(searchBarRef.bottom, margin = 8.dp)
            centerHorizontallyTo(parent)
        }
    ) {
        Row(Modifier.clickable { isSearchDirectionWindowOpen = true }) {
            Icon(Icons.AutoMirrored.Filled.Sort, contentDescription = null)
            Spacer(modifier = Modifier.width(8.dp))
            Text(text = "Sort")
        }
        Spacer(modifier = Modifier.width(16.dp))
        Row(Modifier.clickable { isSearchTypeWindowOpen = true }) {
            Icon(Icons.Default.FilterAlt, contentDescription = null)
            Spacer(modifier = Modifier.width(8.dp))
            Text(text = "Filter")
        }
    }
    when (val response = viewModel.workoutsResponse) {
        is Response.Failure -> {

        }
        Response.Loading -> {
            LinearProgressIndicator(
                modifier = Modifier
                    .fillMaxWidth()
                    .padding(8.dp)
                    .constrainAs(createRef()) {
                        top.linkTo(searchBarRef.bottom)
                        centerHorizontallyTo(parent)
                    })
        }

        is Response.Success -> {
            response.data?.let {
                LazyColumn(Modifier.constrainAs(createRef()) {
                    top.linkTo(searchSettingsRef.bottom, 8.dp)
                    height = Dimension.fillToConstraints
                    bottom.linkTo(parent.bottom)
                }) {
                    items(items = it, key = { it.workoutId }) { workout ->
                        HorizontalDivider(
                            modifier = Modifier.padding(
                                start = 4.dp,
                                end = 4.dp,
                                top = 12.dp,
                                bottom = 12.dp
```

```
                                    )
                                )
                                WorkoutLayout(workout = workout) {
                                    navToWorkoutPage(workout.workoutId)
                                }
                            }
                        }
                    }
                }
            }
        }
        if (isSearchDirectionWindowOpen) {
            ModalBottomSheet(
                sheetState = searchDirectionSheetState,
                onDismissRequest = {
                    isSearchDirectionWindowOpen = false
                },
            ) {
                Column {
                    SortDirection.entries.forEach { direction ->
                        Row(
                            Modifier
                                .fillMaxWidth()
                                .selectable(
                                    selected = (searchWorkoutParams.sortDirection == direction),
                                    onClick = {
                                        searchWorkoutParams =
searchWorkoutParams.copy(sortDirection = direction)
                                    }
                                )
                                .padding(horizontal = 16.dp),
                            verticalAlignment = Alignment.CenterVertically
                        ) {
                            RadioButton(
                                selected = (searchWorkoutParams.sortDirection == direction),
                                onClick = {
                                    searchWorkoutParams = searchWorkoutParams.copy(sortDirection
= direction)
                                }
                            )
                            Text(
                                text = direction.value,
                                style = MaterialTheme.typography.titleLarge,
                                modifier = Modifier.padding(start = 16.dp)
                            )
                        }
                    }
                }
                Spacer(
                    Modifier.windowInsetsBottomHeight(
                        WindowInsets.navigationBarsIgnoringVisibility
                    )
                )
            }
        }
        if (isSearchTypeWindowOpen) {
            ModalBottomSheet(
                sheetState = searchTypeSheetState,
                onDismissRequest = {
                    isSearchTypeWindowOpen = false
                },
            ) {
                Column {
                    Spacer(modifier = Modifier.height(16.dp))
                    SortWorkoutType.entries.forEach { type ->
```

```
                    Row(
                        Modifier
                            .fillMaxWidth()
                            .selectable(
                                selected = (type == searchWorkoutParams.sortWorkoutType),
                                onClick = {
                                    searchWorkoutParams =
searchWorkoutParams.copy(sortWorkoutType = type)
                                }
                            )
                            .padding(horizontal = 16.dp),
                        verticalAlignment = Alignment.CenterVertically
                    ) {
                        RadioButton(
                            selected = (type == searchWorkoutParams.sortWorkoutType),
                            onClick = { searchWorkoutParams =
searchWorkoutParams.copy(sortWorkoutType = type) }
                        )
                        Text(
                            text = type.userValue,
                            style = MaterialTheme.typography.titleLarge,
                            modifier = Modifier.padding(start = 16.dp)
                        )
                    }
                }
            }
            Spacer(
                Modifier.windowInsetsBottomHeight(
                    WindowInsets.navigationBarsIgnoringVisibility
                )
            )
        }
    }
}

@Composable
fun WorkoutLayout(modifier: Modifier = Modifier, workout: Workout, onClick: () -> Unit) {
    ConstraintLayout(
        modifier = modifier
            .fillMaxWidth()
            .clickable {
                onClick()
            }
            .padding(16.dp),
    ) {
        val (staticMapRef, nameRef, routeMetricsRef) = createRefs()
        SubcomposeAsyncImage(
            model = ImageRequest.Builder(LocalContext.current)
                .data(workout.routeImgUrl)
                .crossfade(true)
                .build(),
            contentDescription = "Route image",
            contentScale = ContentScale.Crop,
            loading = {
                CircularProgressIndicator()
            },
            alignment = Alignment.TopCenter,
            modifier = Modifier.constrainAs(staticMapRef) {
                top.linkTo(parent.top)
                start.linkTo(parent.start)
                width = Dimension.percent(0.35f)
                height = Dimension.ratio("1:1")
            }
        )
        Text(
            text = workout.name,
            style = MaterialTheme.typography.titleLarge,
            maxLines = 3,
```

```
                modifier = Modifier
                    .constrainAs(nameRef) {
                        top.linkTo(parent.top)
                        start.linkTo(staticMapRef.end)
                        end.linkTo(parent.end)
                        width = Dimension.fillToConstraints
                    }
                    .padding(start = 12.dp)
            )
            Column(modifier = Modifier.constrainAs(routeMetricsRef) {
                top.linkTo(staticMapRef.bottom, margin = 8.dp)
                width = Dimension.fillToConstraints
                centerHorizontallyTo(parent)
            }) {
                Column(
                    horizontalAlignment = Alignment.CenterHorizontally,
                    modifier = Modifier
                        .fillMaxWidth()
                ) {
                    WorkoutMetrics(workout)
                }
                Spacer(modifier = Modifier.height(24.dp))
                Text(
                    text = workout.timeStamp.getDateTime(),
                    style = MaterialTheme.typography.titleSmall,
                    modifier = Modifier.align(Alignment.End)
                )
            }
        }
    }

    @Composable
    fun ColumnScope.WorkoutMetrics(workout: Workout?) {
        workout?.let {
            Row {
                Row(
                    Modifier
                        .weight(1f)
                        .padding(top = 8.dp, bottom = 8.dp),
                    verticalAlignment = Alignment.CenterVertically,
                    horizontalArrangement = Arrangement.Absolute.Center
                ) {
                    Icon(
                        Icons.Default.HorizontalRule,
                        contentDescription = "Distance",
                        modifier = Modifier.padding(2.dp)
                    )
                    Spacer(modifier = Modifier.width(2.dp))
                    Text(
                        it.distance.roundTo2().toString() + " km",
                        textAlign = TextAlign.Center, style = MaterialTheme.typography.bodySmall
                    )
                }
                Row(
                    Modifier
                        .weight(1f)
                        .padding(top = 8.dp, bottom = 8.dp),
                    verticalAlignment = Alignment.CenterVertically,
                    horizontalArrangement = Arrangement.Absolute.Center
                ) {
                    Icon(
                        Icons.Default.Timer,
                        contentDescription = "Time",
                        modifier = Modifier.padding(2.dp)
                    )
                    Spacer(modifier = Modifier.width(2.dp))
                    Text(
                        it.durationInSeconds.parseSeconds(),
                        textAlign = TextAlign.Center,
                        style = MaterialTheme.typography.bodySmall
```

```
                )
            }
            Row(
                Modifier
                    .weight(1f)
                    .padding(top = 8.dp, bottom = 8.dp),
                verticalAlignment = Alignment.CenterVertically,
                horizontalArrangement = Arrangement.Absolute.Center
            ) {
                Icon(
                    Icons.Default.LocalFireDepartment,
                    contentDescription = "Calories",
                    modifier = Modifier.padding(2.dp)
                )
                Spacer(modifier = Modifier.width(2.dp))
                Text(
                    (it.calories.toInt()).toString() + " kcal",
                    textAlign = TextAlign.Center,
                    style = MaterialTheme.typography.bodySmall
                )
            }
        }
        Row {
            Row(
                Modifier
                    .weight(1f)
                    .padding(top = 8.dp, bottom = 8.dp),
                verticalAlignment = Alignment.CenterVertically,
                horizontalArrangement = Arrangement.Absolute.Center
            ) {
                Icon(
                    Icons.Default.Speed,
                    contentDescription = "Avg. Speed",
                    modifier = Modifier.padding(2.dp)
                )
                Spacer(modifier = Modifier.width(2.dp))
                Text(
                    it.avgSpeed.roundTo2().toString() + " km/h",
                    textAlign = TextAlign.Center,
                    style = MaterialTheme.typography.bodySmall
                )
            }
            Row(
                Modifier
                    .weight(1f)
                    .padding(top = 8.dp, bottom = 8.dp),
                verticalAlignment = Alignment.CenterVertically,
                horizontalArrangement = Arrangement.Absolute.Center
            ) {
                Icon(
                    Icons.Default.ArrowOutward,
                    contentDescription = "Climb",
                    modifier = Modifier.padding(2.dp)
                )
                Spacer(modifier = Modifier.width(2.dp))
                Text(
                    (it.climb.toInt()).toString() + " m",
                    textAlign = TextAlign.Center,
                    style = MaterialTheme.typography.bodySmall
                )
            }
            Row(
                Modifier
                    .weight(1f)
                    .padding(top = 8.dp, bottom = 8.dp),
                verticalAlignment = Alignment.CenterVertically,
                horizontalArrangement = Arrangement.Absolute.Center
            ) {
                Icon(
                    Icons.Default.ArrowDownward,
```

```
                    contentDescription = "Descent",
                    modifier = Modifier.padding(2.dp)
                )
                Spacer(modifier = Modifier.width(2.dp))
                Text(
                    (it.descent.toInt()).toString() + " m",
                    textAlign = TextAlign.Center,
                    style = MaterialTheme.typography.bodySmall
                )
            }
        }
    }
}

@HiltViewModel
class WorkoutsScreenViewModel @Inject constructor(
    private val getUserIdUseCase: GetUserIdUseCase,
    private val cloudStorageWorkoutRepository: CloudStorageWorkoutRepository,
) : ViewModel() {

    var workoutsResponse by mutableStateOf<Response<List<Workout>>>(Response.Loading)
        private set

    fun getWorkoutList(searchWorkoutParams: SearchWorkoutParams = SearchWorkoutParams()) =
viewModelScope.launch {
        val userId = getUserIdUseCase()
        cloudStorageWorkoutRepository.getWorkouts(
            searchWorkoutParams,
            userId.toString()
        ).collect { response ->
            workoutsResponse = response
        }
    }
}
```

HomeCompleteScreen.kt

```
data class BottomNavItem(
    @StringRes val screenNameId: Int,
    val selectedIcon: ImageVector,
    val route: String,
)

@Composable
fun HomeCompleteScreen(navigateToTrackScreen: (routeId: String?) -> Unit) {
    val vm = hiltViewModel<HomeCompleteViewModel>()
    val bottomNavController = rememberNavController()
    val snackbarHostState = remember { SnackbarHostState() }
    val coroutineScope = rememberCoroutineScope()
    val listOfScreens = listOf(
        BottomNavItem(
            screenNameId = R.string.workouts,
            selectedIcon = Icons.AutoMirrored.Filled.DirectionsBike,
            route = AppNavigation.Home.WorkoutsScreen.route
        ),
        BottomNavItem(
            screenNameId = R.string.saved_routes,
            selectedIcon = Icons.Filled.Route,
            route = AppNavigation.Home.SavedRoutesScreen.route
        ),
        BottomNavItem(
            screenNameId = R.string.create_route,
            selectedIcon = Icons.Filled.Create,
            route = AppNavigation.Home.CreateRouteScreen.route
        ),
        BottomNavItem(
            screenNameId = R.string.profile,
            selectedIcon = Icons.Filled.Person,
```

```
                route = AppNavigation.Home.ProfileScreen.route
            ),
        )
        ProvideSnackbarController(
            snackbarHostState = snackbarHostState,
            coroutineScope = coroutineScope
        ) {
            var snackbarMessage: SnackbarMessage? by remember {
                mutableStateOf(null)
            }
            SnackbarMessageHandler(
                snackbarMessage = snackbarMessage,
                onDismissSnackbar = {  },
            )
            Column(
                verticalArrangement = Arrangement.Bottom,
                modifier = Modifier.fillMaxSize()
            ) {
                NavHost(
                    navController = bottomNavController,
                    startDestination = AppNavigation.Home.WorkoutsScreen.route,
                    modifier = Modifier.weight(1f)
                ) {
                    composable(
                        AppNavigation.Home.RedirectStravaScreen.route, deepLinks = listOf(
                            navDeepLink {
                                uriPattern = "strava://redirect?code={code}"
                            })
                    ) { navBackStackEntry ->
                        val code = navBackStackEntry.arguments?.getString("code")
                        if (!code.isNullOrEmpty()) {
                            vm.getAndSaveStravaToken(code = code, onError = {
                                snackbarMessage = SnackbarMessage.from(UserMessage.from("Strava auth
error"))
                            }, onSuccess = {
                                snackbarMessage = SnackbarMessage.from(UserMessage.from("Strava auth
success"))
                            })
                            bottomNavController.navigate(AppNavigation.Home.WorkoutsScreen.route,
navOptions = navOptions {
                                popUpTo(AppNavigation.Home.WorkoutsScreen.route) { inclusive = true }
                            })
                        }
                    }
                    composable(AppNavigation.Home.CreateRouteScreen.route) {
                        RouteBuilderCompleteScreen(snackbarHostState)
                    }
                    composable(AppNavigation.Home.WorkoutsScreen.route) {
                        WorkoutsScreen(snackbarHostState = snackbarHostState, navToWorkoutPage = {
                            bottomNavController.navigate(
                                AppNavigation.Home.WorkoutPageScreen.createWorkoutPageScreen(it)
                            )
                        }, navToWorkout = {
                            navigateToTrackScreen("routeId")
                        })
                    }
                    composable(AppNavigation.Home.WorkoutPageScreen.route, arguments = listOf(
                        navArgument(
                            name = AppNavigation.Home.WorkoutPageScreen.WORKOUT_ID_ARG,
                        ) {
                            type = NavType.StringType
                        }
                    )) {
                        val workoutId =

it.arguments?.getString(AppNavigation.Home.WorkoutPageScreen.WORKOUT_ID_ARG)
                        workoutId?.let {
                            WorkoutPageCompleteScreen(
                                snackbarHostState = snackbarHostState,
                                workoutId = workoutId,
```

```
                        ) {
                            bottomNavController.popBackStack()
                        }
                    }
                }
                composable(
                    AppNavigation.Home.SavedRoutesScreen.route
                ) {
                    SavedRoutesScreen(snackbarHostState, navToRoutePage =  { routeId ->
                        bottomNavController.navigate(
                            AppNavigation.Home.RoutePageScreen.createRoutePageScreen(
                                routeId
                            )
                        )
                    }, navToWorkout = {
                        navigateToTrackScreen("routeId")
                    })
                }
                composable(
                    AppNavigation.Home.ProfileScreen.route
                ) {
                    ProfileScreen(snackbarHostState)
                }
                composable(AppNavigation.Home.RoutePageScreen.route, arguments = listOf(
                    navArgument(
                        name = AppNavigation.Home.RoutePageScreen.ROUTE_ID_ARG,
                    ) {
                        type = NavType.StringType
                    }
                )) { backStackEntry ->
                    val routeId =

backStackEntry.arguments?.getString(AppNavigation.Home.RoutePageScreen.ROUTE_ID_ARG)
                    routeId?.let {
                        RoutePageScreenCompleteScreen(
                            snackbarHostState = snackbarHostState,
                            routeId = routeId,
                            startWorkout = {
                                navigateToTrackScreen(it)
                            }, onBack = {
                                bottomNavController.popBackStack()
                            })
                    }
                }
            }
            NavigationBar {
                val navBackStackEntry by bottomNavController.currentBackStackEntryAsState()
                val currentDestination = navBackStackEntry?.destination
                listOfScreens.forEach { screen ->
                    NavigationBarItem(
                        selected = currentDestination?.hierarchy?.any { it.route == screen.route }
== true,
                        onClick = {
                            bottomNavController.navigate(screen.route) {
                                popUpTo(bottomNavController.graph.findStartDestination().id) {
                                    saveState = true
                                }
                                launchSingleTop = true
                                restoreState = true
                            }
                        },
                        icon = {
                            Icon(
                                imageVector = screen.selectedIcon,
                                contentDescription = stringResource(id = screen.screenNameId)
                            )
                        },
                        label = {
                            Text(text = stringResource(id = screen.screenNameId))
                        })
```

```
                }
            }
        }
    }
}
```

TrackScreen.kt

```
@Composable
fun WorkoutInfoPanel(
    modifier: Modifier = Modifier,
    speed: Double,
    distance: Double,
    avgSpeed: Double,
    hours: String,
    minutes: String,
    seconds: String,
) {
    Column(modifier = modifier) {
        Row(
            modifier = Modifier
                .padding(16.dp)
                .fillMaxWidth()
        )
        {
            WorkoutPanelDetail(
                modifier = Modifier.weight(1f),
                title = "Speed",
                value = speed.toString(),
                units = "km/h",
                icon = Icons.Default.Speed
            )
            Spacer(modifier = Modifier.width(16.dp))
            WorkoutPanelDetail(
                modifier = Modifier.weight(1f),
                title = "Avg. Speed",
                value = avgSpeed.toString(),
                units = "km/h",
                icon = Icons.Default.ShutterSpeed
            )
        }
        Row(
            modifier = Modifier
                .fillMaxWidth()
                .padding(16.dp)
        )
        {
            WorkoutPanelDetail(
                modifier = Modifier.weight(1f),
                title = "Distance",
                value = distance.toString(),
                units = "km",
                icon = Icons.Default.Route
            )
            Spacer(modifier = Modifier.width(16.dp))
            WorkoutPanelDetail(
                modifier = Modifier.weight(1f),
                title = "Duration",
                value = "$hours:$minutes:$seconds",
                units = "",
                icon = Icons.Default.Timer
            )
        }
    }
}

@Composable
fun WorkoutPanelDetail(
    modifier: Modifier,
```

```
    title: String,
    value: String,
    units: String,
    icon: ImageVector,
    description: String? = null,
) {
    Row(modifier) {
        Icon(
            icon, contentDescription = description, modifier = Modifier.padding(8.dp)
        )
        Text(
            text = buildAnnotatedString {
                withStyle(style = SpanStyle(fontWeight = FontWeight.Thin)) {
                    append(title)
                }
                append("\n")
                withStyle(style = SpanStyle(fontWeight = FontWeight.Bold)) {
                    append(value)
                }
                withStyle(style = SpanStyle(fontWeight = FontWeight.Thin)) {
                    append(" $units")
                }
            },
            modifier = Modifier
                .weight(1f)
                .align(Alignment.CenterVertically),
            textAlign = TextAlign.Center,
            style = MaterialTheme.typography.titleLarge
        )
    }
}

object TrackHelper {

    val pendingUri =

"sporterapp://${AppNavigation.Workout.TRACK_FEATURE_SCREEN_ROUTE}/${AppNavigation.Workout.TrackScr
een.route}"

    const val NOTIFICATION_CLICK_REQUEST_CODE = 100
    fun createPendingIntent(context: Context): PendingIntent {
        val intent =
            Intent(Intent.ACTION_VIEW, pendingUri.toUri(), context, MainActivity::class.java)
        return TaskStackBuilder.create(context).run {
            addNextIntentWithParentStack(intent)
            getPendingIntent(NOTIFICATION_CLICK_REQUEST_CODE, PendingIntent.FLAG_IMMUTABLE)
        }
    }
}


sealed class TrackScreenEvent {
    data object OnBack : TrackScreenEvent()
    data class ChangeMapStyle(val style: MapBoxStyle) : TrackScreenEvent()
    data class StopWorkout(val name: String) : TrackScreenEvent()
}

@OptIn(MapboxExperimental::class)
@Composable
fun TrackScreen(
    trackingService: TrackingUserWorkoutService,
    uiState: TrackScreenUiState,
    onEvent: (TrackScreenEvent) -> Unit,
) {
    val context = LocalContext.current
    var showDropdownStyleMenu by remember { mutableStateOf(false) }
    var isStopWorkoutDialogOpen by remember { mutableStateOf(false) }
    var workoutName by remember { mutableStateOf("") }
    val mapViewportState = rememberMapViewportState {
        setCameraOptions {
```

```
            center(Point.fromLngLat(0.0, 0.0))
            zoom(0.0)
            pitch(0.0)
        }
        MapAnimationOptions.mapAnimationOptions {
            duration(3000)
        }
    }

    val permissionRequest = rememberLauncherForActivityResult(
        contract = ActivityResultContracts.RequestMultiplePermissions(),
        onResult = { permissions ->
            if (!permissions.values.all { it }) {
                onEvent(TrackScreenEvent.OnBack)
            } else {
                mapViewportState.flyToUserPosition()
            }
        }
    )

    val mapBoxUiSettings: GesturesSettings by remember {
        mutableStateOf(GesturesSettings {
            rotateEnabled = true
            pinchToZoomEnabled = true
            pitchEnabled = true
        })
    }

    val compassSettings: CompassSettings by remember {
        mutableStateOf(CompassSettings {
            enabled = true
        })
    }

    val attributionSettings: AttributionSettings by remember {
        mutableStateOf(AttributionSettings {
            enabled = false
        })
    }

    val scaleBarSetting: ScaleBarSettings by remember {
        mutableStateOf(ScaleBarSettings {
            enabled = true
        })
    }
    LaunchedEffect(Unit) {
        permissionRequest.launch(MapUtils.locationPermissions)
    }
    ConstraintLayout(modifier = Modifier.fillMaxSize()) {
        val (mapRef, workoutInfoRef) = createRefs()
        WorkoutInfoPanel(
            modifier = Modifier.constrainAs(workoutInfoRef) {
                top.linkTo(parent.top)
                width = Dimension.matchParent
                height = Dimension.fillToConstraints
            },
            speed = trackingService.speed,
            distance = trackingService.distance.roundTo2(),
            avgSpeed = trackingService.avgSpeed,
            hours = trackingService.hours,
            minutes = trackingService.minutes,
            seconds = trackingService.seconds,
        )
        MapboxMap(
            modifier = Modifier.constrainAs(mapRef) {
                width = Dimension.matchParent
                height = Dimension.fillToConstraints
                top.linkTo(workoutInfoRef.bottom)
                bottom.linkTo(parent.bottom)
            },
```

```
            mapInitOptionsFactory = { context ->
                MapInitOptions(
                    context = context,
                    styleUri = MapBoxStyle.STREET.style,
                )
            },
            compassSettings = compassSettings,
            mapViewportState = mapViewportState,
            scaleBarSettings = scaleBarSetting,
            gesturesSettings = mapBoxUiSettings,
            attributionSettings = attributionSettings,
        ) {
            MapEffect(Unit) { mapView ->
                with(mapView) {
                    location.locationPuck = createDefault2DPuck(withBearing = true)
                    location.pulsingEnabled = false
                    location.enabled = true
                }
            }
            MapEffect(uiState.currentMapStyle) { mapView ->
                with(mapView) {
                    if (mapboxMap.isValid()) {
                        mapboxMap.loadStyle(uiState.currentMapStyle.style)
                    }
                }
            }
            trackingService.preparedRoute?.points?.let {
                PolylineAnnotationGroup(
                    annotations = listOf(
                        PolylineAnnotationOptions()
                            .withPoints(points = it)
                            .withLineBorderWidth(1.0)
                            .withLineBorderColor(context.getColor(R.color.light_blue))
                            .withLineWidth(3.0)
                            .withLineColor(context.getColor(R.color.dark_blue))
                    ),
                    annotationConfig = AnnotationConfig(
                        belowLayerId = "track-layer",
                        layerId = "prepared_track-layer",
                    ),
                    lineCap = LineCap.ROUND,
                )
            }
            PolylineAnnotationGroup(
                annotations = listOf(
                    PolylineAnnotationOptions()
                        .withPoints(points = trackingService.points.map { it.point })
                        .withLineBorderWidth(2.0)
                        .withLineBorderColor(context.getColor(R.color.yellow))
                        .withLineWidth(7.0)
                        .withLineColor(context.getColor(R.color.light_red))
                ),
                annotationConfig = AnnotationConfig(
                    layerId = "track-layer",
                ),
                lineCap = LineCap.ROUND,
            )
        }
        Row(
            Modifier
                .constrainAs(createRef()) {
                    start.linkTo(parent.start, margin = 8.dp)
                    bottom.linkTo(parent.bottom, margin = 8.dp)
                    width = Dimension.fillToConstraints
                }
                .background(MaterialTheme.colorScheme.surface.copy(0.5f))) {
            IconButton(onClick = {
                with(context) {
                    val intent = if (trackingService.isWorkoutStarted) {
                        TrackingUserWorkoutService.createToggleIntent(
```

```
                            trackingService.isTracking,
                            this
                        )
                    } else {
                        TrackingUserWorkoutService.createStartIntent(this)
                    }
                    startService(intent)
                }
            }) {
                Icon(
                    if (trackingService.isTracking) Icons.Default.Pause else
Icons.Default.PlayArrow,
                    contentDescription = "Toggle tracking"
                )
            }
            IconButton(onClick = {
                if (!trackingService.isWorkoutStarted) {
                    onEvent(TrackScreenEvent.OnBack)
                } else {
                    isStopWorkoutDialogOpen = true
                }
            }) {
                Icon(
                    Icons.Default.Stop,
                    contentDescription = "Stop tracking"
                )
            }
        }
        Column(
            modifier = Modifier
                .background(
                    color = MaterialTheme.colorScheme.surface.copy(0.7f),
                    shape = MaterialTheme.shapes.medium
                )
                .padding(8.dp)
                .constrainAs(createRef()) {
                    bottom.linkTo(parent.bottom)
                    end.linkTo(parent.end)
                },
            verticalArrangement = Arrangement.spacedBy(4.dp)
        ) {
            Box {
                IconButton(onClick = { showDropdownStyleMenu = true }) {
                    Icon(
                        Icons.Default.Layers,
                        contentDescription = stringResource(R.string.map_layers)
                    )
                }
                DropdownMenu(
                    expanded = showDropdownStyleMenu,
                    onDismissRequest = { showDropdownStyleMenu = false }) {
                    MapBoxStyle.entries.forEach {
                        DropdownMenuItem(
                            modifier = Modifier.padding(4.dp),
                            text = {
                                Row(verticalAlignment = Alignment.CenterVertically) {
                                    Image(
                                        painter = painterResource(id = it.resId),
                                        contentDescription = stringResource(id = it.textId)
                                    )
                                    Spacer(modifier = Modifier.width(4.dp))
                                    Text(text = stringResource(id = it.textId))
                                }
                            }, onClick = {
                                onEvent(TrackScreenEvent.ChangeMapStyle(it))
                                showDropdownStyleMenu = false
                            })
                        HorizontalDivider()
                    }
                }
```

```
                }
                IconButton(onClick = {
                    mapViewportState.flyToUserPosition()
                }) {
                    Icon(
                        Icons.Default.MyLocation,
                        contentDescription = stringResource(R.string.my_location)
                    )
                }
            }
        }
        if (isStopWorkoutDialogOpen) {
            AlertDialog(
                icon = {
                    Icon(Icons.Default.Warning, contentDescription = "Warning Icon")
                },
                title = {
                    Text(text = "Stop workout?")
                },
                text = {
                    TextField(
                        modifier = Modifier
                            .padding(16.dp),
                        value = workoutName,
                        onValueChange = {
                            workoutName = it
                        },
                        singleLine = true,
                        minLines = 1,
                        isError = workoutName.isBlank()
                    )
                },
                onDismissRequest = {
                    isStopWorkoutDialogOpen = false
                },
                confirmButton = {
                    TextButton(
                        enabled = workoutName.isNotBlank(),
                        onClick = {
                            onEvent(TrackScreenEvent.StopWorkout(workoutName))
                            with(context) {
                                val intent = TrackingUserWorkoutService.createStopIntent(this)
                                startService(intent)
                            }
                            isStopWorkoutDialogOpen = false
                        }
                    ) {
                        Text("Save workout")
                    }
                },
                dismissButton = {
                    TextButton(
                        onClick = {
                            isStopWorkoutDialogOpen = false
                        }
                    ) {
                        Text("Cancel")
                    }
                }
            )
        }
    }

    data class TrackScreenUiState(
        val preparedRoute: Route? = null,
        val currentMapStyle: MapBoxStyle = MapBoxStyle.STREET,
        val snackbarMessage: SnackbarMessage? = null,
    )

    @HiltViewModel(assistedFactory = TrackScreenViewModel.TrackScreenViewModelFactory::class)
```

```
class TrackScreenViewModel @AssistedInject constructor(
    @Assisted val routeId: String?,
    private val getRouteByIdUseCase: GetRouteByIdUseCase,
) : ViewModel() {

    var state by mutableStateOf(TrackScreenUiState())
        private set

    fun getRoute(onRouteLoaded: (route: Route?) -> Unit) {
        viewModelScope.launch {
            routeId?.let {
                getRouteByIdUseCase(it,
                    onError = {
                        onGetRouteError()
                        onRouteLoaded(null)
                    },
                    onSuccess = { route: Route ->
                        onGetRouteSuccess(route)
                        onRouteLoaded(route)
                    })
            }
        }
    }

    private fun onGetRouteError() {
        state = state.copy(
            snackbarMessage = SnackbarMessage.from(UserMessage.from(R.string.route_loading_error))
        )
    }

    private fun onGetRouteSuccess(route: Route) {
        state = state.copy(preparedRoute = route)
    }

    fun changeMapStyle(style: MapBoxStyle) {
        state = state.copy(currentMapStyle = style)
    }

    fun workoutSavedError(message: String) {
        state = state.copy(snackbarMessage = SnackbarMessage.from(UserMessage.from(message)))
    }

    @AssistedFactory
    interface TrackScreenViewModelFactory {
        fun create(id: String?): TrackScreenViewModel
    }
}

@Composable
fun TrackCompleteScreen(
    routeId: String?,
    trackingService: TrackingUserWorkoutService,
    onBack: () -> Unit,
) {
    val context = LocalContext.current
    val vm =
        hiltViewModel<TrackScreenViewModel, TrackScreenViewModel.TrackScreenViewModelFactory> {
factory ->
            factory.create(routeId)
        }
    LaunchedEffect(Unit) {
        with(context) {
            vm.getRoute { route ->
                if (route == null || route.routeId.isEmpty()) return@getRoute
                val intent = TrackingUserWorkoutService.createRouteIntent(this, route)
                startService(intent)
            }
        }
    }
```

```
        TrackScreen(trackingService, uiState = vm.state) {
            when (it) {
                is TrackScreenEvent.ChangeMapStyle -> {
                    vm.changeMapStyle(it.style)
                }

                TrackScreenEvent.OnBack -> {
                    onBack()
                }

                is TrackScreenEvent.StopWorkout -> {
                    if (trackingService.points.isEmpty() || trackingService.points.size < 2) {
                        onBack()
                        return@TrackScreen
                    }
                    trackingService.saveWorkout(it.name) {
                        onBack()
                    }
                }
            }
        }
    }
```