

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій
(повне найменування факультету)

Кафедра програмних засобів
(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

магістр

(ступінь вищої освіти)

на тему ДОСЛІДЖЕННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ
АНАЛІТИЧНОГО ЗАСТОСУНКУ-КРОКОМІРА З ВИКОРИСТАННЯМ
АКСЕЛЕРОМЕТРА
RESEARCH AND SOFTWARE IMPLEMENTATION OF AN ANALYTICAL
PEDOMETER APPLICATION USING AN ACCELEROMETER

Виконав: студент(ка) 2 курсу, групи КНТ-123м
Спеціальності 121 Інженерія програмного
забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація) Інженерія
програмного забезпечення

РУДЬ О.К.

(ПРИЗВИЩЕ та ініціали)

Керівник ФЕДОРОНЧАК. Т.В.

(ПРИЗВИЩЕ та ініціали)

Рецензент КОЦУР М.І.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
(повне найменування закладу вищої освіти)

Факультет КНТ
Кафедра програмних засобів
Ступінь вищої освіти магістр
Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
“ ” 2024 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

РУДЬ Орини Костянтинівни

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Дослідження та програмна реалізація аналітичного застосунку-крокоміра з використанням акселерометра. Research and Software Implementation of an Analytical Pedometer Application Using an Accelerometer.

керівник проєкту (роботи) к.т.н., доцент, ФЕДОРОНЧАК Тетяна Василівна,
(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від “07” жовтня 2024 року № 398

2. Строк подання студентом проєкту (роботи) 1 грудня 2024 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз проблеми та постановка завдань дослідження. 2. Матеріали і методи. 3. Проєктування програмного забезпечення. 4. Основні рішення щодо реалізації програмного забезпечення. 5. Експлуатація, тестування та експериментальне дослідження програми.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-5 Основна частина	ФЕДОРОНЧАК Т.В., доцент		
Нормоконтроль	КАЛІНІНА М.В., асистент		

7. Дата видачі завдання « 07 » жовтня 2024 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	2 тиждень	Розділ 1
3	Дослідження проблеми.	3–5 тижні	Розділ 2
4	Розробка архітектури програми.	6 тиждень	Розділ 3
5	Розробка програми.	7–8 тижні	Розділ 4
6	Тестування та експериментальне дослідження програми.	9–10 тижні	Розділ 5
7	Оформлення пояснювальної записки та документів до неї. Нормоконтроль та рецензування.	11 тиждень	Додатки
8	Захист роботи.	12 тиждень	

Студент (ка)

(підпис) Орина РУДЬ
(Імя ПРІЗВИЩЕ)

Керівник проєкту (роботи)

(підпис) Тетяна ФЕДОРОНЧАК
(Імя ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи магістра:
170 с., 7 табл., 31 рис., 3 дод., 25 джерел.

АКСЕЛЕРОМЕТЕР, КРОКОМІР, ВІСТЕЖУВАННЯ АКТИВНОСТІ,
АНАЛІЗ АКТИВНОСТІ, АНАЛІТИЧНИЙ ЗАСТОСУНОК.

Об'єкт дослідження – процес збору та аналізу даних про фізичну активність за допомогою акселерометра мобільних пристроїв.

Предмет дослідження – програмне забезпечення для крокоміра, яке використовує акселерометр для відстеження та аналізу кроків користувача.

Мета роботи – розробка програмного забезпечення для аналізу фізичної активності користувачів через відстеження кількості кроків за допомогою акселерометра, з метою підвищення точності вимірювань та надання користувачам зручної аналітики для моніторингу їхнього здоров'я і активності.

Матеріали, методи та технічні засоби: об'єктно-орієнтоване програмування, середовища розробки Android Studio, мова програмування Kotlin, фреймворки Jetpack Compose для Android та бібліотеки для обробки даних.

Результати. Розроблено програмне забезпечення для крокоміра, яке аналізує дані акселерометра та надає детальну аналітику фізичної активності користувачів, відображаючи кількість кроків, спалені калорії та рівень активності.

Висновки. Розроблене програмне забезпечення для крокоміра надає корисну аналітику користувачам та може бути інтегроване в інші системи моніторингу здоров'я.

Галузь використання – сфера здоров'я та фітнесу.

ABSTRACT

Explanatory note to the diploma qualifying work of the master: 170 pages, 7 tables, 31 figures, 3 appendixes, 25 sources.

ACCELEROMETER, PEDOMETER, ACTIVITY TRACKING, ACTIVITY ANALYSIS, ANALYTICAL APPLICATION.

The object of the research is the process of collecting and analyzing data on physical activity using a mobile device accelerometer.

The subject of the research is pedometer software that uses an accelerometer to track and analyze user steps.

The work aims to develop software for analyzing users' physical activity by tracking the number of steps using an accelerometer. This will improve measurement accuracy and give users convenient analytics for monitoring their health and activity.

Materials, methods, and technical tools: object-oriented programming, the Android Studio development environment, the Kotlin programming language, the Jetpack Compose frameworks for Android, and data processing libraries.

The results. Software for a pedometer has been developed, which analyzes accelerometer data and provides detailed analytics on users' physical activity, displaying the number of steps, calories burned, and activity levels.

Conclusions. The developed pedometer software provides valuable analytics to users and can be integrated into other health monitoring systems.

Field of application – the health and fitness industry.

ЗМІСТ

С.

Перелік скорочень та умовних позначок.....	7
Вступ.....	8
1 Аналіз проблеми та дослідження методів підрахунку кроків.....	9
1.1 Огляд існуючих застосунків для відстежування активності.....	9
1.2 Порівняння аналогів програм для відстеження кроків.....	11
1.3 Висновки за розділом 1.....	15
2 Матеріали і методи.....	17
2.1 Збір даних за допомогою акселерометра.....	17
2.2 Алгоритми обробки даних для підрахунку кроків.....	19
2.3 Зберігання та аналіз активності на основі даних акселерометра.....	22
2.4 Висновки за розділом 2.....	23
3 Проєктування програмного забезпечення.....	25
3.1 Вибір засобів розробки.....	25
3.2 Моделювання програмного забезпечення на основі функціональних вимог.....	31
3.3 Проєктування структури бази даних для зберігання інформації про активність.....	35
3.4 Висновки за розділом 3.....	39
4 Основні рішення щодо реалізації програмного забезпечення.....	40
4.1 Порядок роботи застосунку-крокоміра.....	40
4.2 Технічна реалізація аналітичних функцій.....	42
4.3 Висновки за розділом 4.....	45
5 Експлуатація, тестування та експериментальне дослідження програми.....	46
5.1 Опис роботи застосунку.....	46
5.2 Тестування та перевірка точності підрахунку кроків.....	50
5.3 Висновки за розділом 5.....	53
Висновки.....	54
Перелік джерел посилань.....	56
Додаток А Технічне завдання.....	58
Додаток Б Текст програми.....	62
Додаток В Слайди презентації.....	162

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

ADT – Android Development Tools;
API – Application Programming Interface;
CPU – Central Processing Unit;
DAO – data access object;
DI – Dependency Injection;
DSL – Domain-Specific Language;
IDE – Integrated Development Environment;
JDK – Java Development Kit;
JVM – Java Virtual Machine;
ORM – Object-relational mapping;
SDK – Software development Kit;
UI – User Interface;
БД – база даних;
ОС – операційна система;
ПЗ – програмне забезпечення;
СУБД – система управління базами даних.

ВСТУП

Сектор здоров'я та фітнесу активно впроваджує новітні технології, які допомагають підвищити ефективність моніторингу фізичної активності. Ключовим компонентом цих технологій є крокоміри, які можуть відстежувати активність користувача та надавати детальний аналіз фізичної активності. Мобільні пристрої з акселерометрами відіграють особливу роль у цьому плані, оскільки вони можуть фіксувати рухи користувача і перетворювати їх на кроки.

Сьогодні доступно багато застосунків для відстеження кроків, але не всі вони пропонують достатню точність і простоту використання. Основними проблемами є неточний підрахунок кроків за різних умов, обмежені можливості аналізу та відсутність зворотного зв'язку, який допоміг би користувачам краще зрозуміти свою фізичну активність. Тому існує потреба в розробці більш точних і функціональних аналітичних застосунків для крокомірів.

Метою цієї роботи є створення програмного забезпечення, яке за допомогою акселерометра відстежує кількість кроків користувача та забезпечує розширені можливості аналізу, що дозволяють ефективно моніторити стан здоров'я та рівень фізичної активності користувача.

З огляду на зростаючий інтерес до персоналізованого відстеження здоров'я та активного способу життя, вкрай важливо створювати точні та корисні інструменти для моніторингу фізичної активності. Застосунки для крокомірів корисні не тільки для користувачів, які хочуть залишатися у формі, але й для медичних працівників, які можуть використовувати дані для моніторингу та консультування пацієнтів.

Таким чином, актуальність та практична значущість цієї роботи полягає у створенні сучасного інструменту, який дозволить контролювати фізичну активність за допомогою акселерометрів на мобільних пристроях.

1 АНАЛІЗ ПРОБЛЕМИ ТА ДОСЛІДЖЕННЯ МЕТОДІВ ПІДРАХУНКУ КРОКІВ

1.1 Огляд існуючих застосунків для відстежування активності

У сучасному світі розвиток цифрових технологій призвів до появи широкого спектру програмного забезпечення, покликаного покращити якість життя користувачів. Однією з важливих сфер є крокоміри та інші програми для моніторингу здоров'я та фізичної активності. Ці програми допомагають людям реєструвати свою активність протягом дня, сприяючи поліпшенню загального стану здоров'я і благополуччя. Ці застосунки часто використовуються як невід'ємна частина щоденної рутини, дозволяючи користувачам аналізувати свій прогрес, ставити нові цілі та досягати їх.

Однією з основних технологій, що лежить в основі таких застосунків, є використання акселерометрів - спеціалізованих датчиків, вбудованих у більшість сучасних смартфонів. Акселерометри реагують на рухи користувача і можуть точно вимірювати кількість кроків, зроблених за день. Завдяки цій технології застосунки для крокомірів стають дедалі популярнішими серед звичайних користувачів, професійних спортсменів або тих, хто прагне вести здоровий спосіб життя.

Застосунки-крокоміри слугують кільком цілям. По-перше, вони забезпечують простий і зручний спосіб відстежувати щоденну активність. Вони дозволяють користувачам отримувати детальні звіти про кількість зроблених кроків, пройдену відстань, спалені калорії та тривалість активності. Це дає загальну картину рівня вашої фізичної активності та дозволяє відстежувати ваш прогрес у часі.

По-друге, крокоміри заохочують користувачів бути активними, встановлюючи щоденні або щотижневі цілі. Багато застосунків дозволяють користувачам встановлювати індивідуальні цілі, наприклад, досягти певної кількості кроків на день, що може мотивувати їх до підвищення рівня фізичної активності. Ще одним важливим аспектом є те, що такі застосунки можуть

надсилати нагадування та сповіщення, щоб нагадати користувачам про необхідність рухатися.

Крім того, застосунки для підрахунку кроків часто пропонують додаткові функції, корисні для різних категорій користувачів. Наприклад, деякі застосунки інтегрують можливість відстежувати інші показники здоров'я, такі як частота серцебиття та аналіз якості сну. Це дає комплексне уявлення про стан здоров'я та активність користувача протягом дня.

З технічної точки зору, застосунки для підрахунку кроків повинні бути енергоефективними. Це пов'язано з тим, що безперервний моніторинг активності може суттєво вплинути на час автономної роботи смартфонів. Розробники таких застосунків зосереджуються на оптимізації акселерометра, щоб мінімізувати споживання енергії без шкоди для точності підрахунку кроків.

Користувацький інтерфейс також є важливим фактором при розробці таких застосунків. Простий та інтуїтивно зрозумілий інтерфейс робить такі програми привабливими для широкого кола користувачів. Щоб застосунок був популярним, він повинен бути простим у використанні, забезпечувати чітку візуалізацію даних і бути доступним для користувачів різних вікових груп.

На ринку існує велика кількість застосунків для підрахунку кроків, кожен з яких має свої переваги та унікальні функції. Деякі застосунки пропонують розширені функції для аналізу активності, тоді як інші зосереджені на мінімалістичному підході з базовими функціями відстеження кроків. Крім того, деякі з доступних застосунків пропонують платні та безкоштовні рішення. Таким чином, користувачі можуть вибрати варіант, який найкраще відповідає їхнім потребам та вподобанням.

Таким чином, сучасні застосунки для підрахунку кроків є потужними інструментами для моніторингу фізичної активності. Застосунки можуть допомогти користувачам підтримувати здоровий спосіб життя, мотивуючи їх бути більш активними та надаючи життєво важливу інформацію для аналізу стану здоров'я.

1.2 Порівняння аналогів програм для відстеження кроків

У рамках дослідження було проаналізовано три популярні застосунки для підрахунку кроків, які доступні на платформі Google Play: Pedometer - Step Counter від Leap Fitness Group, Step Counter - Pedometer від Hitchhike Tech та Pedometer від ITO Technologies, Inc. Для оцінки програм були взяті до уваги такі критерії: зручність використання, точність підрахунку кроків, наявність додаткових функцій, оцінка користувачів і кількість завантажень.

1.2.1 Огляд застосунку Pedometer - Step Counter

Застосунок [1] вирізняється простим і зрозумілим інтерфейсом, який підходить для широкого кола користувачів. Головна сторінка застосунку зображена на рис. 1.1. Основною функцією є підрахунок кроків, проте він також надає детальні дані про витрачені калорії, відстань, швидкість руху та час активності. Користувачі високо оцінюють точність підрахунку кроків, а також низьке споживання батареї.

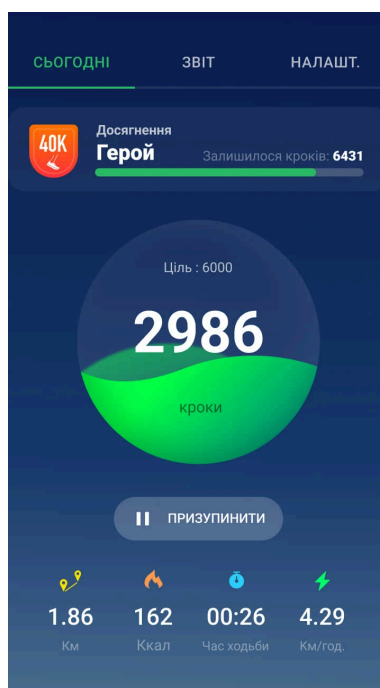


Рисунок 1.1 – Головний екран застосунку Pedometer - Step Counter [1]

Важливою особливістю є можливість використовувати застосунок без доступу до інтернету. Однак деякі користувачі відзначають проблеми з підрахунком кроків при малорухомих активностях (наприклад, підйом по сходах).

Розробник: Leap Fitness Group.

Оцінка користувачів: 4.9/5.

Кількість завантажень: 50 млн+.

1.2.2 Огляд застосунку Step Counter - Pedometer

Цей застосунок [2] пропонує схожий функціонал, зокрема підрахунок кроків, аналіз калорій, пройденої відстані та часу активності. Головна сторінка застосунку зображена на рис. 1.2. Особливістю є доступ до даних за тривалі періоди часу, що дозволяє користувачам бачити свою активність за день, тиждень чи місяць.

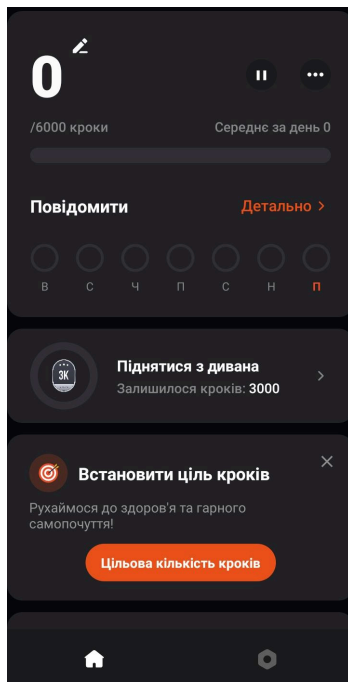


Рисунок 1.2 – Головний екран застосунку Step Counter - Pedometer [2]

Користувачі позитивно відгукуються про інтерфейс, який є мінімалістичним та інтуїтивним. Застосунок також підтримує синхронізацію з іншими

фітнес-платформами. Основною перевагою є адаптивність до різних типів фізичних навантажень та висока точність при різних умовах використання.

Розробник: Hitchhike Tech.

Оцінка користувачів: 4.9/5.

Кількість завантажень: 10 млн+.

1.2.3 Огляд застосунку Pedometer

Застосунок [3] фокусується на простоті та легкості використання. Він не вимагає створення облікових записів або використання додаткових сервісів, що робить його привабливим для користувачів, які не хочуть втрачати час на реєстрацію.

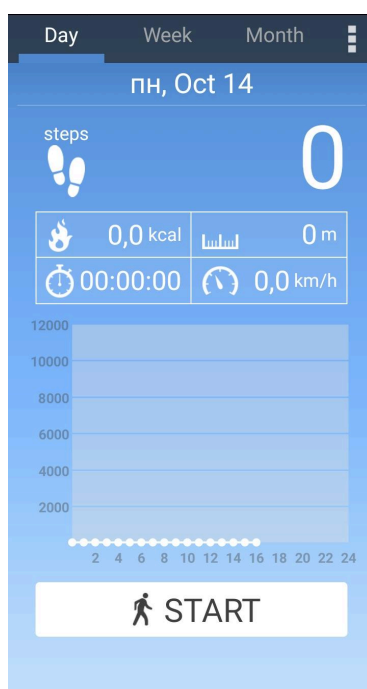


Рисунок 1.3 – Головний екран застосунку Pedometer [3]

Основною перевагою є дуже низьке споживання енергії та стабільна робота в фоновому режимі. Однак застосунок пропонує менше функцій порівняно з конкурентами. Він не має таких розширених можливостей, як аналітика за тривалі

періоди часу або відображення швидкості руху, що може обмежувати його використання для більш досвідчених користувачів.

Розробник: ITO Technologies, Inc.

Оцінка користувачів: 4.8/5.

Кількість завантажень: 50 млн+.

1.2.4 Порівняльний аналіз

У табл. 1.1 представлено порівняння трьох популярних застосунків для підрахунку кроків, які використовують акселерометри мобільних пристроїв. Аналіз включає основні характеристики, такі як оцінка користувачів, кількість завантажень, точність підрахунку кроків та функціонал. Це допоможе визначити переваги та недоліки кожного з застосунків, що, в свою чергу, спростить вибір найбільш підходящого рішення для моніторингу фізичної активності.

Таблиця 1.1 – Порівняння популярних застосунків для підрахунку кроків

Характеристика	Pedometer - Step Counter	Step Counter - Pedometer	Pedometer
1	2	3	4
Оцінка користувачів	4.9/5	4.9/5	4.8/5
Кількість завантажень	50 млн+	10 млн+	50 млн+
Інтерфейс	Простий і інтуїтивно зрозумілий	Мінімалістичний	Простота використання
Точність підрахунку кроків	Хороша, але з похибками	Висока та стабільна	Висока, але з обмеженнями
Функціонал	Підрахунок кроків, калорій, відстані	Підрахунок кроків, аналіз активності	Базовий підрахунок кроків

Продовження таблиці 1.1

1	2	3	4
Способи аналізу активності	Дані за день, тиждень, місяць	Дані за день, тиждень, місяць	Немає розширеної аналітики
Інтеграція з іншими платформами	Немає	Наявна	Немає
Можливості роботи у фоновому режимі	Низьке споживання батареї	Низьке споживання батареї	Низьке споживання батареї
Реклама	Так	Так	Так
Інтернет для роботи	Не потрібен	Потрібен для синхронізації	Не потрібен
Особливі функції	Немає	Синхронізація з фітнес-застосунками	Немає реєстрації

1.3 Висновки за розділом 1

Проаналізувавши наявні застосунки для відстеження активності, можна зробити висновок, що ринок мобільних застосунків у цій сфері пропонує різноманітні рішення, кожне з яких має свої унікальні особливості та переваги. Порівняльний аналіз застосунків показує, що кожен застосунок намагається задовольнити потреби користувачів, використовуючи акселерометри для точної реєстрації кількості зроблених кроків.

На основі цих даних можна виділити такі функціональні вимоги до застосунку:

- застосунок має точно підраховувати кількість зроблених користувачем кроків, використовуючи дані акселерометра;
- надання детальної статистики про щоденну, щотижневу, щомісячну та щорічну активність користувача;

- відображення даних про витрачені калорії, пройдену відстань та тривалість активності;
- інтуїтивно зрозумілий та мінімалістичний інтерфейс для зручності користувача;
- графічне відображення статистики активності;
- можливість встановлення індивідуальних цілей. Таких як, певна кількість кроків на день;
- система досягнень для мотивації користувача;
- застосунок повинен працювати у фоновому режимі з низьким споживанням енергії, забезпечуючи безперервний підрахунок кроків.

Загалом, аналіз показує, що ринок мобільних трекерів активності є динамічним і конкурентним, а також має багато можливостей для вдосконалення. Подальший розвиток у цій сфері може бути зосереджений на інтеграції нових технологій, таких як штучний інтелект для підвищення точності вимірювань та створення зручних і функціональних інтерфейсів для забезпечення кращого користувацького досвіду.

2 МАТЕРІАЛИ І МЕТОДИ

У цьому розділі представлено матеріали та методи, використані для реалізації застосунку аналітичного крокоміра, який використовує акселерометри для вимірювання активності користувача. Процес розробки включав збір даних, обробку сигналів, розробку алгоритму визначення кроків та візуалізацію інформації. Метою було створення застосунку, який міг би відстежувати фізичну активність та надавати користувачеві корисну інформацію про кількість зроблених кроків.

2.1 Збір даних за допомогою акселерометра

Акселерометри в сучасних мобільних пристроях надають інформацію про рух користувача і тому є важливим інструментом для реалізації функціональності, пов'язаної з підрахунком кроків. Збір даних акселерометра стає все більш поширеним при розробці мобільних застосунків, призначених для моніторингу активності користувача.

Збір даних для акселерометрів є складним завданням, що передбачає інтеграцію декількох ключових компонентів. Основною метою цього процесу є розробка системи, здатної отримувати та аналізувати дані з датчиків на мобільних пристроях в режимі реального часу.

Розробка застосунка здійснюватиметься в Android Studio з використанням мови програмування Kotlin. Такий вибір обумовлений кількома важливими факторами. Android Studio є офіційним середовищем розробки для створення Android-застосунків. Воно пропонує потужний набір інструментів для налагодження, тестування та профілювання, а також тісно інтегрується з Android SDK. Використання цього середовища дозволяє забезпечити стабільну розробку, підтримку останніх версій платформи та доступ до всіх необхідних функцій. Kotlin є сучасною, ефективною та офіційно підтримуваною мовою для

Android-розробки, що широко застосовується завдяки своїй лаконічності та багатому функціоналу, який підходить для роботи з даними в реальному часі.

Основним компонентом для збору даних є клас `SensorManager`. Цей клас є частиною Android SDK і забезпечує доступ до сенсорів пристрою, включно з акселерометром. Він відповідає за управління сенсорами та взаємодію з ними. Процес збору даних можна розділити на кілька етапів. Спочатку необхідно ініціалізувати клас `SensorManager`, який відповідає за доступ до сенсорів пристрою. Це забезпечує доступ до акселерометра, який використовується для збору даних про рух.

Наступним кроком є реалізація класу-слухача датчика, який реалізує інтерфейс `SensorEventListener`. Цей клас відіграє важливу роль в обробці даних з акселерометра і гарантує, що вони будуть належним чином оброблені методом `onSensorChanged`. У методі `onSensorChanged` можна обробляти отримані дані, аналізуючи зміни у значеннях акселерометра. Для підрахунку кроків зазвичай використовується алгоритм фільтрації даних, що дозволяє виділити періоди, коли користувач робить кроки, на основі змін у показниках прискорення.

Наступним кроком є забезпечення актуальності даних про рух користувача шляхом реєстрації слухачів у методах життєвого циклу активності. Зокрема, слухачі вмикаються в методі `onResume` та вимикаються в методі `onPause`, щоб уникнути зайвого споживання ресурсів.

Обробка отриманих даних передбачає аналіз значень акселерометра, що дозволяє користувачеві визначити час ходьби. Використання ефективних алгоритмів фільтрації даних забезпечує точність при визначенні активності користувача.

Тому збір даних акселерометра є важливим етапом у розробці аналітичних застосунків крокомірів, які демонструють високий потенціал у сфері моніторингу активності користувача та підрахунку кроків.

2.2 Алгоритми обробки даних для підрахунку кроків

Підрахунок кроків на основі даних, отриманих з акселерометра, є складним завданням, яке вимагає застосування різноманітних алгоритмів обробки даних. З ростом популярності фітнес-застосунків та мобільних технологій, розробники шукають ефективні способи для точної детекції кроків та моніторингу фізичної активності користувачів.

Основою для підрахунку кроків є обробка даних, отриманих з акселерометра, за допомогою алгоритмів, які наведені у табл. 2.1. [4] - [6].

Таблиця 2.1 – Алгоритми для обробки даних з акселерометра

Алгоритм	Опис	Переваги	Недоліки
1	2	3	4
Метод простого порогового значення	Використовує порогове значення для показників акселерометра. При перевищенні порогу система зараховує це як один крок.	Простота реалізації	Може бути неточним у шумних умовах або при нерівномірному ритмі ходьби
Фільтрація сигналів	Використовує фільтри (наприклад, фільтр Калмана, фільтр низьких частот) для усунення шумів та непотрібних коливань, виділяючи значущі рухи	Покращує точність підрахунку	Вимагає додаткових обчислень і налаштувань
Алгоритм детекції піків	Алгоритм детекції піків Аналізує дані акселерометра для виявлення різких змін у векторі прискорення, які вказують на крок. Всі виявлені піки підраховуються як кроки	Висока точність при правильно налаштованій системі	Може пропустити кроки при нерегулярному русі

Продовження таблиці 2.1

1	2	3	4
Комбіновані алгоритми	Комбіновані алгоритми Поеднує кілька методів, наприклад, спочатку фільтрацію для усунення шумів, а потім алгоритм детекції піків для підрахунку реальних кроків	Знижує ймовірність помилок	Складніша реалізація і потребує більше ресурсів
Машинне навчання	Машинне навчання Використовує моделі машинного навчання, навчені на великих наборах даних, для виявлення складних патернів руху та точної детекції кроків	Висока точність, врахування різних умов	Необхідність в великій кількості даних для навчання

У `SensorManager` використовується алгоритм, який працює на основі детекції піків прискорення (алгоритм детекції піків), що відбуваються під час ходьби користувача. Цей алгоритм вбудований у клас `StepDetector`, що входить до складу `API Android`, зокрема, він доступний з `API Level 19 (Android KitKat)`.

`StepDetector` аналізує дані акселерометра та гіроскопа для визначення піків у векторі прискорення, які відповідають кожному кроку. Якщо виявлено пік, він зараховується як крок. Таким чином, цей алгоритм визначає кроки на основі чітких пікових значень змін прискорення, що відповідають характерним рухам при ходьбі.

Використання цього підходу дає змогу досягти високої точності в ситуаціях із регулярним ходінням, однак можуть бути пропуски при нерегулярних рухах або в умовах сильної зміни швидкості.

У сучасних мобільних застосунках, що сприяють здоровому способу життя та фітнесу, важливу роль відіграє правильний збір даних про активність користувача. Один з основних інструментів для цього – `SensorManager`, клас у

Android, який дозволяє отримувати дані з різних сенсорів, включаючи акселерометр.

SensorManager є частиною Android Framework і забезпечує програмний інтерфейс для доступу до сенсорів пристрою. Він управляє різними сенсорами, включаючи акселерометри, гіроскопи, магнітометри та інші. Коли програма запитує дані з акселерометра, SensorManager працює за принципом обробки подій, відстежуючи зміни в сенсорах та передаючи ці дані до слухачів (listeners), які підписуються на ці оновлення [7].

Коли застосунок ініціює запит на отримання даних з акселерометра, SensorManager виконує кілька ключових функцій:

- адаптація до фізичного обладнання: SensorManager має доступ до низькорівневих драйверів сенсорів, які обробляють дані безпосередньо з апаратних компонентів. Це дозволяє йому адаптуватися до різних моделей пристроїв і їхніх характеристик, забезпечуючи єдиний API для розробників;

- планування оновлень: SensorManager реалізує механізм планування, який управляє частотою оновлень даних з сенсорів. Розробники можуть вказати, з якою частотою вони хочуть отримувати дані (наприклад, висока частота для реального часу або нижча для енергозбереження). SensorManager оптимізує отримання даних, щоб зменшити навантаження на процесор і зберегти енергію батареї;

- кешування даних: Для забезпечення ефективності та зменшення затримок SensorManager може кешувати дані, отримані від сенсорів. Це дозволяє застосунку отримувати більш швидкий доступ до останніх вимірювань, зменшуючи час реакції на зміни активності користувача;

- обробка подій: Коли дані з акселерометра оновлюються, SensorManager створює події, які передаються слухачам. Ці події містять інформацію про нові вимірювання, а також їх точність. Це дозволяє застосункам реалізовувати складні алгоритми обробки даних, такі як виявлення кроків, з використанням алгоритмів фільтрації та аналізу руху;

- управління ресурсами: Для уникнення надмірного використання ресурсів SensorManager дозволяє розробникам скасовувати реєстрацію слухачів, коли вони

більше не потрібні, тим самим знижуючи енергоспоживання та навантаження на процесор [7] - [10].

Таким чином, `SensorManager` є невід'ємною частиною архітектури застосунків, які використовують дані акселерометра для підрахунку кроків та моніторингу активності користувача. Його здатність обробляти дані в реальному часі забезпечує точність та ефективність у розробці сучасних фітнес-застосунків.

2.3 Зберігання та аналіз активності на основі даних акселерометра

Аналіз активності користувачів на основі даних акселерометрів є ключовим елементом у розробці сучасних фітнес-застосунків. Акселерометр надає інформацію про рух і прискорення і може виявляти різні види активності, такі як ходьба, біг, стояння або сидіння. Цей аналіз допомагає користувачеві отримати корисну інформацію про фізичну активність і забезпечує точність підрахунку кроків і моніторингу стану здоров'я.

Для реалізації аналізу активності користувачів було розроблено кілька важливих етапів збору, обробки та зберігання даних про трафік.

2.3.1 Методологія аналізу

По-перше, ми використовуємо клас `SensorManager` для отримання даних про прискорення з усіх 3 осей (X, Y, Z) у режимі реального часу. Це дозволяє збирати детальну інформацію про рухи користувача під час фізичної активності. Зібрані дані проходять процес попередньої обробки, який включає фільтрацію для усунення шуму та непотрібних коливань. Такий підхід забезпечує точність визначення тривалості активності.

Важливим кроком є аналіз даних, який може визначати різні типи активності користувачів. Основними показниками, які слід враховувати, є кількість кроків, вид діяльності і сила руху. Це допоможе скласти повну картину фізичної активності користувача і точно визначити, як він буде проводити свій час.

2.3.2 Зберігання даних

Всі отримані дані про діяльність користувача зберігаються в базі даних Room, бібліотеці для роботи з SQLite на Android. Room дозволяє організовувати ваші дані структурованим чином і спрощує доступ до історії подій. Це дозволяє користувачам переглядати результати, аналізувати свій прогрес і отримувати інформацію про фізичну активність протягом певного періоду часу. Зберігаючи дані в Room, ви також можете виконувати різні запити для отримання інформації про подію. Користувачі можуть отримувати звіти про фізичну активність, включаючи як щоденні, так і щотижневі показники. Завдяки цим звітам користувачі можуть краще розуміти свою фізичну активність і при необхідності змінювати свій розпорядок дня.

2.3.3 Візуалізація результатів

Після збору та аналізу даних важливо їх візуалізувати. Це може включати графіки, які надають користувачам зручний спосіб оцінки фізичної активності. Такі візуалізації дозволяють користувачам переглядати щоденні, щотижневі або щомісячні звіти, які допомагають їм відстежувати свій прогрес і досягнення.

2.4 Висновки за розділом 2

У розділі 2 детально описано матеріали та методи, використані при розробці аналітичного крокоміра на основі даних акселерометра. Процес розробки застосунку складається з декількох важливих етапів, включаючи збір, обробку, зберігання та аналіз даних, які забезпечують точний та ефективний моніторинг фізичної активності користувача.

Збір даних акселерометра є основою для реалізації функції підрахунку кроків; клас `SensorManager` дозволяє отримувати та обробляти дані в режимі реального часу, сприяючи точності вимірювань та можливості адаптації до різних

станів активності. Алгоритми обробки даних, такі як фільтрація та виявлення піків, забезпечують виділення важливих рухів, що призводить до надійного підрахунку кроків.

Зберігання даних у базі даних Room дозволяє організувати їх у структурований спосіб і надає користувачеві легкий доступ до історії активності. Користувачі можуть переглядати свої результати, аналізувати свій прогрес та отримувати звіти про свою фізичну активність, що сприяє більш усвідомленому підходу до здоров'я.

Візуалізація даних є важливим кроком, а графіки та діаграми забезпечують зручний спосіб оцінки фізичної активності. Це допомагає користувачам відстежувати свої досягнення та мотивувати їх докладати більше фізичних зусиль.

Таким чином, розробка крокоміра, що аналізує дані акселерометра, передбачає інтеграцію різних методів і прийомів, які сприяють створенню точного та ефективного інструменту для моніторингу фізичної активності користувача. Результати цього дослідження можуть слугувати основою для подальших досліджень у цій галузі та сприяти покращенню здоров'я та фізичної активності користувачів.

3 ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

У цьому розділі розглядається процес проєктування аналітичного застосунку-крокоміра, який використовує дані акселерометра для відстеження фізичної активності. Основну увагу зосереджено на виборі технологій розробки, створенні моделей, що відповідають функціональним вимогам, та проєктуванні бази даних для зберігання інформації про активність користувача.

3.1 Вибір засобів розробки

Для успішної реалізації програмного забезпечення важливо правильно обрати інструменти, які забезпечать ефективну розробку, зручність підтримки та високу продуктивність застосунку. У даному підрозділі обґрунтовується вибір мов програмування, середовищ розробки та фреймворків. Зокрема, для створення мобільного застосунку-крокоміра на платформі Android використовуються Kotlin та Jetpack Compose для сучасного й гнучкого UI. Для обробки даних з акселерометра застосовується SensorManager, який забезпечує доступ до датчиків пристрою. Також обговорюється інтеграція Hilt для впровадження залежностей і Room для роботи з базою даних.

3.1.1 Kotlin та Android Studio

Для розробки мобільного застосунку-крокоміра обрано мову Kotlin з таких причин:

- сучасний синтаксис і зручність написання коду. Kotlin має лаконічний і виразний синтаксис у порівнянні з Java, що дозволяє писати менш громіздкий код. Наприклад, декларація об'єктів, використання лямбда-виразів і робота з колекціями в Kotlin значно простіші;

- підвищена безпека від помилок. Kotlin надає вбудовану підтримку null safety, що знижує ризик помилок, пов'язаних із NullPointerException. Це критично для мобільних застосунків, де важлива стабільність роботи;

- повна сумісність із Java. Kotlin повністю сумісний із Java, що дозволяє використовувати вже існуючі бібліотеки та фреймворки, написані на Java. Це полегшує інтеграцію з іншими компонентами та розширює можливості застосунку;

- підтримка від Google. Починаючи з 2017 року, Google офіційно рекомендує Kotlin як основну мову для розробки Android-застосунків. Це гарантує підтримку нових функцій і технологій;

- функціональні можливості. Kotlin підтримує функціональне програмування, що спрощує обробку потоків даних і підвищує читабельність коду. Це особливо корисно для роботи з асинхронними операціями, які часто застосовуються в мобільних застосунках;

- покращена продуктивність. Компіляція Kotlin-джерел забезпечує продуктивність, аналогічну Java, але з додатковими перевагами за рахунок оптимізації коду [11] - [12].

Android Studio є офіційним середовищем розробки (IDE) для створення Android-застосунків і була обрана з таких причин:

- інтеграція з Android SDK. Android Studio забезпечує глибоку інтеграцію з Android SDK, що дозволяє легко налаштовувати середовище для роботи з останніми версіями платформи Android;

- інструменти для швидкої розробки. Android Studio включає такі корисні інструменти, як Layout Editor, що дозволяє швидко створювати та налаштовувати інтерфейс користувача за допомогою Drag-and-Drop. Також підтримується Jetpack Compose, що спрощує створення сучасних UI;

- потужна система дебагінгу. Інтегровані інструменти для налагодження (debugging) і моніторингу продуктивності допомагають швидко виявляти та виправляти помилки;

- емулятори Android. Android Studio пропонує вбудований емулятор, що дозволяє тестувати застосунок на різних пристроях і версіях Android без фізичного доступу до реальних пристроїв;
- інструменти профілювання. Android Studio містить набір інструментів для аналізу продуктивності застосунку, таких як Memory Profiler, CPU Profiler і Network Profiler. Це важливо для оптимізації ресурсів, враховуючи обмежені можливості мобільних пристроїв;
- підтримка Hilt і Room. Android Studio забезпечує повну інтеграцію з фреймворками Hilt (для Dependency Injection) і Room (для роботи з базами даних), що значно прискорює розробку та підвищує якість коду;
- безперервна інтеграція та тестування. Інструменти, вбудовані в Android Studio, дозволяють легко налаштувати CI/CD процеси для автоматизації тестування та збірки проєктів, що покращує надійність застосунку [13].

Таким чином, вибір Kotlin та Android Studio забезпечує швидку, ефективну та зручну розробку надійного мобільного застосунку з широкими функціональними можливостями.

3.1.2 Jetpack Compose та SensorManager

Для створення мобільного застосунку-крокоміра важливо обрати сучасні інструменти, які забезпечать високу продуктивність розробки, гнучкість дизайну та зручність використання. Надалі детально розглядається вибір фреймворку для створення інтерфейсу користувача — Jetpack Compose, а також використання SensorManager для обробки даних з акселерометра.

Jetpack Compose — це сучасний інструмент для створення інтерфейсів на платформі Android, який був обраний через такі переваги:

- декларативний підхід. На відміну від традиційного XML-проектування, Compose використовує декларативний підхід, який дозволяє описувати UI у вигляді функцій. Це спрощує процес розробки, оскільки розробник зосереджується на тому, що має відображатися, а не як це реалізувати;

- гнучкість і простота створення UI. Завдяки Compose легко створювати складні й динамічні інтерфейси. Він дозволяє швидко налаштовувати вигляд компонентів, використовуючи зручний і лаконічний синтаксис;
- менше коду. Створення UI в Compose вимагає значно менше коду порівняно з традиційним підходом, що зменшує кількість потенційних помилок і прискорює процес розробки;
- повна інтеграція з Jetpack. Compose легко інтегрується з іншими компонентами Jetpack, такими як LiveData, ViewModel, Navigation, що забезпечує зручне керування станом і навігацією;
- адаптивний дизайн. Compose дозволяє легко створювати адаптивні інтерфейси, які коректно відображаються на різних розмірах екранів і пристроях, що є критично важливим для сучасних мобільних застосунків;
- модульність і повторне використання компонентів. У Compose легко створювати власні UI-компоненти, які можна повторно використовувати у різних частинах застосунку, що знижує складність коду і підвищує його підтримуваність [14].

Для відстеження фізичної активності користувача застосунком використовує дані акселерометра, обробка яких забезпечується за допомогою `SensorManager`. Цей вибір обґрунтований наступними перевагами:

- прямий доступ до датчиків пристрою. `SensorManager` надає API для роботи з різними датчиками, включаючи акселерометр, гіроскоп і магнітометр. Це дозволяє зчитувати дані в реальному часі, необхідні для підрахунку кроків;
- ефективне використання енергії. `SensorManager` оптимізований для енергоефективної роботи. Це важливо для збереження заряду батареї, особливо в застосунках, які працюють у фоновому режимі;
- гнучкість у налаштуванні частоти опитування. Розробник може налаштовувати частоту отримання даних з датчика, залежно від потреб застосунку. Це дозволяє досягти балансу між точністю та споживанням ресурсів;

- події та обробка даних у реальному часі. `SensorManager` дозволяє обробляти дані акселерометра в режимі реального часу, що необхідно для коректного підрахунку кроків та виявлення фізичної активності;

- широка підтримка пристроїв. `SensorManager` є стандартним компонентом `Android SDK`, тому він підтримується практично на всіх сучасних `Android`-пристроях, що забезпечує стабільну роботу застосунку на різних платформах [15].

Таким чином, `Jetpack Compose` забезпечує ефективну розробку сучасного інтерфейсу, а `SensorManager` надає потужний інструментарій для роботи з апаратними датчиками, що дозволяє реалізувати точне та енергоефективне відстеження фізичної активності користувача.

3.1.3 Hilt та Room

Для забезпечення високої продуктивності розробки, підтримуваності коду та ефективної роботи з даними в мобільному застосунку, важливо використовувати відповідні інструменти та бібліотеки.

`Hilt` — це офіційний фреймворк для впровадження залежностей в `Android`-застосунках, розроблений на базі `Dagger`. Його вибір обґрунтований такими перевагами:

- спрощення управління залежностями. `Hilt` автоматизує процес впровадження залежностей, що дозволяє зменшити кількість ручного коду, необхідного для ініціалізації об'єктів. Це підвищує продуктивність розробки та знижує ризик помилок;

- модульність і розділення відповідальностей. Використання `Hilt` дозволяє легко створювати незалежні модулі з чітко визначеними залежностями. Це полегшує тестування і покращує підтримку коду;

- гнучка інтеграція з життєвими циклами `Android`-компонентів. `Hilt` надає анотації для автоматичного впровадження залежностей у компоненти з різними

життєвими циклами, такими як Activity, Fragment, ViewModel або Service. Це значно знижує складність роботи з багатокomпонентними застосунками;

- масштабованість і підтримка. Hilt ідеально підходить для масштабних проєктів, де залежності можуть бути складними і численними. Його підтримка на рівні Jetpack гарантує сумісність із сучасними технологіями Android;

- зменшення boilerplate-коду. Завдяки Hilt, розробники можуть уникнути великої кількості шаблонного коду, який зазвичай супроводжує ручне впровадження залежностей, що робить код більш читабельним і підтримуваним [16] - [17].

Room — це офіційна бібліотека для роботи з локальною базою даних у застосунках Android. Вона базується на SQLite, але значно спрощує взаємодію з нею. Вибір Room обґрунтований наступними перевагами:

- спрощена робота з базою даних. Room надає об'єктно-реляційне відображення (ORM), що дозволяє працювати з базою даних за допомогою звичайних об'єктів Java/Kotlin. Це зменшує складність і обсяг коду порівняно з прямою роботою з SQLite;

- синтаксична перевірка запитів. Room виконує перевірку SQL-запитів на етапі компіляції, що дозволяє виявляти помилки ще до запуску застосунку. Це підвищує надійність коду;

- автоматична синхронізація з потоками даних. Підтримка LiveData та Flow дозволяє автоматично отримувати оновлення з бази даних у реальному часі, що спрощує реалізацію реактивного програмування;

- інтеграція з Hilt. Room легко інтегрується з Hilt, що дозволяє автоматично впроваджувати базу даних і DAO у будь-які компоненти застосунку;

- міграції бази даних. Room забезпечує зручні механізми для виконання міграцій бази даних, що дозволяє оновлювати структуру таблиць без втрати даних користувача;

- підвищена продуктивність і оптимізація. Room використовує перевірені практики роботи з базою даних і оптимізує запити для підвищення продуктивності навіть на пристроях з обмеженими ресурсами.

Таким чином, використання Hilt забезпечує ефективне управління залежностями, а Room — просту та надійну роботу з базою даних, що є критично важливим для створення масштабованого і підтримуваного застосунку [18] - [19].

3.2 Моделювання програмного забезпечення на основі функціональних вимог

Функціональна структура програми визначає, які задачі система виконує, як користувачі взаємодіють з нею, і як внутрішні компоненти взаємодіють між собою для виконання цих задач. У данному проєкті основні функції реалізовані за допомогою кількох взаємопов'язаних модулів. Для ілюстрації цієї структури використовуються діаграми прецедентів, класів та послідовностей.

Діаграми прецедентів описують взаємодію між користувачем і системою, зображуючи функціональні можливості програмного забезпечення. Основні прецеденти наведені у табл. 3.1.

Таблиця 3.1 – Основні прецеденти користувача в застосунку для підрахунку кроків.

Прецедент	Опис
Перегляд кількості кроків	Користувач отримує інформацію про пройдені кроки за поточний день
Аналіз активності	Користувач переглядає історичні дані про свою фізичну активність, такі як графіки кроків, середні значення
Отримання досягнення	Користувач отримує нагороди або бачить повідомлення про досягнення певних цілей
Налаштування профілю	Користувач оновлює особисту інформацію (ім'я, вік, стать, мету кроків)

Діаграма прецедентів наведена на рис. 3.1.

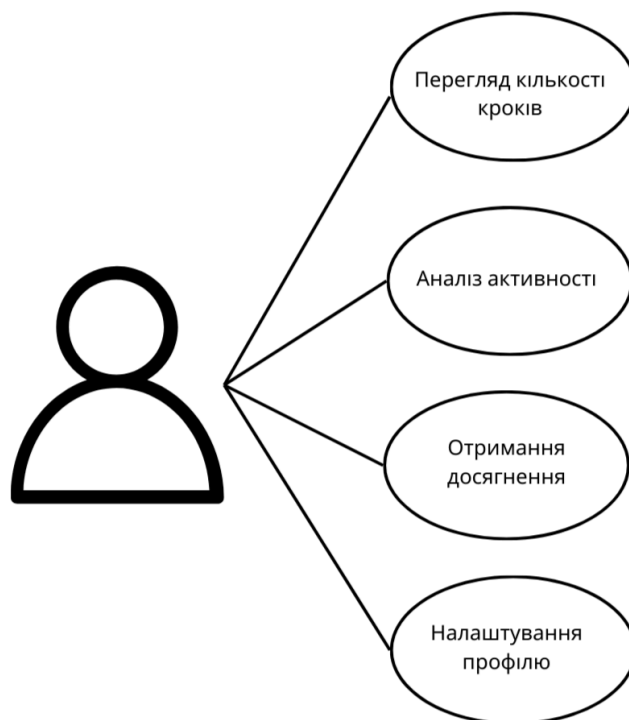


Рисунок 3.1 – Діаграма прецедентів

Діаграма класів відображає структуру програмного забезпечення, зокрема основні класи, їх атрибути, методи та зв'язки. Основні класи наведені у табл. 3.2.

Таблиця 3.2 – Структура класів і їх методів для застосунку підрахунку кроків

Клас	Атрибути	Методи
1	2	3
User	name, age, gender, height, weight	updateProfile(), getGoal()
StepRepository	немає (залежить від бази даних)	getDailySteps(), saveSteps(), getStepsByDateRange()
Achievement	id, title, description, dateEarned	checkAchievement()

Продовження таблиці 3.2

1	2	3
ProfileViewModel	userData: Flow<User>	updateUserProfile(), getUserData()
AnalyticsViewModel	dailySteps: Flow<List<StepData>>	getAnalyticsData(), calculateTrends()
DatabaseModule	немає	надає DAO-об'єкти

Діаграма класів зображена на рис. 3.2.

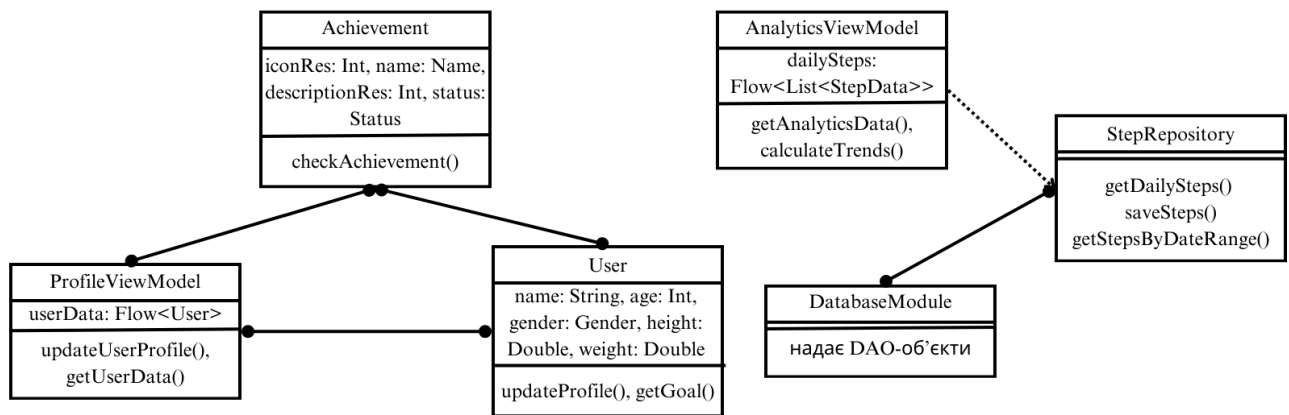


Рисунок 3.2 – Діаграма класів

Діаграма послідовностей демонструє порядок взаємодії між користувачем, інтерфейсом, репозиторіями та базою даних.

Взаємодія для перегляду щоденного прогресу:

- користувач відкриває застосунок;
- HomeScreen викликає метод у HomeViewModel для отримання кроків за день;
- HomeViewModel звертається до StepRepository для отримання даних;
- StepRepository виконує запит до бази даних через DAO і повертає кількість кроків;
- дані передаються через ViewModel назад до HomeScreen;

– HomeScreen оновлює UI, відображаючи кількість кроків.

У будь-якому програмному забезпеченні важливо правильно організувати взаємодію між різними його компонентами для забезпечення ефективності, зручності в обслуговуванні та підтримці. У контексті мобільних застосунків, розроблених з використанням архітектури MVVM, взаємодія між компонентами відбувається в кількох основних етапах. Це дозволяє чітко розподілити відповідальність між компонентами, сприяючи гнучкості і масштабованості програми.

Перший етап взаємодії відбувається між користувачем та інтерфейсом користувача (UI). Користувач взаємодіє з програмою через елементи UI, такі як кнопки, текстові поля, графічні елементи. Вони дозволяють здійснювати основні операції, наприклад, перегляд статистики кроків, оновлення профілю або перегляд досягнень. UI — це те, що бачить користувач, тому він повинен бути інтуїтивно зрозумілим і зручним. Взаємодія з UI здійснюється через події, такі як натискання кнопок або введення тексту, які в свою чергу ініціюють виклики відповідних методів у ViewModel.

Після того, як користувач взаємодіє з UI, інформація передається у ViewModel. ViewModel слугує посередником між UI та бізнес-логікою програми. Він відповідає за обробку даних, отриманих з репозиторію, і підготовку цих даних до відображення на екрані. ViewModel не має безпосереднього доступу до даних або сховища, тому він звертається до репозиторію за інформацією або за командою для її оновлення.

Репозиторій — це спеціальний компонент, який відповідає за взаємодію з джерелами даних. Це можуть бути як локальні бази даних, так і віддалені сервери. Репозиторій абстрагує доступ до даних, знижуючи складність для ViewModel, дозволяючи йому не турбуватися про те, звідки ці дані надходять (локальний або віддалений ресурс).

На наступному рівні знаходиться взаємодія між репозиторієм і базою даних. У програмі, що використовує Room Database, репозиторій здійснює запити до бази даних через об'єкти DAO. DAO є спеціалізованими класами, які визначають

методи для доступу до даних (наприклад, отримання, збереження, оновлення або видалення записів). Репозиторій працює з DAO, отримує необхідні дані та передає їх в ViewModel.

Таким чином, кожен компонент має свою чітко визначену відповідальність, що дозволяє підтримувати чисту і добре організовану архітектуру програми. Взаємодія між ними є основою для забезпечення коректної роботи застосунку і оптимізації процесів обробки даних [20] - [22].

Функціональна структура програми базується на архітектурі MVVM, що забезпечує модульність і спрощує обслуговування. Діаграми прецедентів, класів і послідовностей відображають ключові функціональні можливості програми та їх реалізацію через взаємодію модулів.

3.3 Проєктування структури бази даних для зберігання інформації про активність

У цьому підрозділі розглядається проєктування бази даних для зберігання даних, що стосуються фізичної активності користувачів застосунку. Оскільки застосунок відслідковує кількість кроків, особисті дані користувачів та їхні досягнення на основі активності, потрібно спроектувати ефективну структуру бази даних для зберігання та обробки цих даних.

Розглянемо основні таблиці для зберігання даних, схема бази даних наведена на рис. 3.3.

yearly_steps	year: TEXT month: INTEGER steps: INTEGER
monthly_steps	month: TEXT day: INTEGER steps: INTEGER
daily_steps	id: INTEGER PK date: TEXT hour: INTEGER steps: INTEGER
total_steps	id: INTEGER PK steps: INTEGER

Рисунок 3.3 – Схема бази даних

DailyStepData – зберігає дані про кількість кроків, здійснених користувачем за кожну годину протягом певного дня. Вона дозволяє детально відстежувати активність користувача на годинному рівні, що забезпечує точніший аналіз його фізичної активності протягом дня.

Структура таблиці:

- id: унікальний ідентифікатор запису (тип: INTEGER, PRIMARY KEY);
- date: дата, коли були зафіксовані кроки (тип: STRING);
- hour: година в межах дня, для якої записані кроки (тип: INTEGER);
- steps: кількість кроків, зроблених користувачем у цю годину (тип: LONG).

Така структура дозволяє зберігати кількість кроків для кожної години окремо, що може бути корисним для подальшого аналізу активності протягом дня.

MonthlyStepData – зберігає дані про кількість кроків, здійснених користувачем протягом певного місяця. Кожен запис у таблиці містить інформацію про конкретний день місяця та кількість кроків, що були зроблені в цей день. Використовуючи таку структуру, можна зручно аналізувати активність користувача на рівні місяця, що дозволяє робити висновки про його загальну активність протягом кожного місяця.

Структура таблиці:

- month: місяць, в якому були зафіксовані кроки (тип: TEXT, формат "YYYY-MM");
- day: день місяця, в який були здійснені кроки (тип: INTEGER);
- steps: кількість кроків, здійснених цього дня (тип: INTEGER).

Унікальним ідентифікатором запису є комбінація місяця і дня, що дозволяє зберігати дані на рівні дня для кожного місяця.

YearlyStepData – зберігає інформацію про кількість кроків, здійснених користувачем протягом певного місяця в конкретному році. Кожен запис в таблиці містить рік, місяць і підсумкову кількість кроків за цей місяць.

Структура таблиці:

- year: рік, до якого належать дані (тип: STRING, формат "YYYY");
- month: місяць, до якого належать дані (тип: INTEGER, від 1 до 12);

– steps: загальна кількість кроків за вказаний місяць (тип: LONG).

Таблиця використовується для агрегації даних за місяць і рік, що дозволяє швидко отримати статистику про активність користувача на більших тимчасових інтервалах.

TotalStepData – зберігає загальну кількість кроків, здійснених користувачем за весь період часу використання застосунку. Данні агрегації дозволяють ефективно відстежувати активність на загальних проміжках часу та проводити швидший аналіз.

Структура таблиці:

- id: унікальний ідентифікатор запису (тип: INTEGER, PRIMARY KEY);
- steps: загальна кількість кроків за період часу (тип: INTEGER).

У рамках проєктування бази даних для зберігання інформації про активність користувачів була розроблена спеціальна логіка агрегації даних, що дозволяє зберігати їх у різних часових інтервалах — день, місяць, рік. Це дозволяє значно прискорити аналіз даних, зменшуючи навантаження на систему при запитах і обробці великих обсягів інформації.

Для ефективного зберігання даних і забезпечення швидкого доступу було вирішено агрегувати дані по кроках для кожного користувача в різних часових інтервалах. Такий підхід дозволяє зменшити кількість даних, що зберігаються в таблиці на один день, і дає змогу швидко виконувати запити на аналіз активності користувачів за більші періоди.

Агрегація на рівні дня: У таблиці DailyStepData зберігається кількість кроків, здійснених користувачем за певний день. Кожен запис у цій таблиці має бути пов'язаний з конкретною датою та годиною, що дозволяє відстежувати активність на щоденній основі. Це дозволяє здійснювати точний аналіз активності користувача на рівні дня.

Агрегація на рівні дня: Для кожного дня зберігається час фіксації даних (наприклад, кроки можуть бути записані кожну годину чи через певні інтервали, що дозволяє більш точно аналізувати активність протягом дня).

Агрегація на рівні місяця: Для забезпечення швидкого аналізу даних на більш великих тимчасових інтервалах, додано агрегацію даних на рівні місяця. Кількість кроків, здійснених користувачем протягом місяця, можна обчислювати як суму всіх значень за кожен день цього місяця. Такий підхід дає змогу отримати узагальнену інформацію про активність користувача за певний місяць без необхідності проводити складні запити.

Агрегація на рівні року: Для довгострокового аналізу та статистики був додан ще один рівень агрегації — на рівні року. Це дозволяє швидко отримувати загальну інформацію про активність користувача протягом року. Дані для цієї агрегації розраховуються шляхом підсумовування кроків за кожен місяць року.

Крім того, агрегація виконується автоматично при кожному отриманні даних з акселерометра. Якщо під час отримання нового кроку перша записана дата в таблиці `DailyStepData` відповідає сьогоднішній, оновлюється відповідне поле для поточної години. Якщо ж це не так, перша записана дата вважається попереднім днем. У цьому випадку відбувається підсумовування кроків за попередній день, і ця сума записується в таблицю `MonthlyStepData`. У таблиці `MonthlyStepData` перевіряється, чи відповідає перший запис поточному місяцю. Якщо так, то оновлюється відповідне поле для минулого дня. В іншому випадку підсумовуються всі кроки за минулий місяць і ця сума переноситься в таблицю `YearlyStepData`. Таблиця `TotalStepData` оновлюється при кожному отриманні даних з акселерометра шляхом додавання нових кроків до загальної суми.

Це забезпечує швидший аналіз даних, так як:

- зменшення обсягу даних. Агрегування даних за місяць і рік зменшує кількість записів, що потрібно зберігати на кожен період. Це особливо важливо при роботі з великими обсягами даних. Зберігання лише підсумкових значень для місяця та року дозволяє зменшити кількість записів, що зберігаються в базі даних;
- швидші запити та аналіз. Завдяки агрегації, запити, які обробляють великі обсяги даних (наприклад, запит на отримання середньої кількості кроків за місяць чи рік), можна виконати швидше, оскільки база даних не повинна обробляти

мільйони окремих записів. Замість цього, вона працює з агрегованими даними, що значно зменшує час виконання запитів;

– легкість у прогнозуванні та аналізі трендів. Агреговані дані на рівні місяця та року дозволяють ефективно здійснювати аналіз трендів активності користувача, порівнювати активність між різними періодами, а також робити прогнози щодо досягнень та цілей. Це особливо важливо для функціоналу, який може відображати статистику за різні тимчасові інтервали [23] - [25].

3.4 Висновки за розділом 3

У цьому розділі було детально розглянуто процес проєктування програмного забезпечення для аналітичного крокоміра, який використовує дані акселерометра для відстеження фізичної активності. Вибір технологій розробки, таких як мова Kotlin, середовище Android Studio, фреймворки Hilt і Room, а також використання Jetpack Compose для створення інтерфейсу, забезпечують високу ефективність, зручність підтримки і масштабованість застосунку. Інтеграція SensorManager для обробки даних акселерометра дозволяє досягти точності та енергоефективності, що є важливими для мобільних застосунків.

Застосування архітектури MVVM сприяє чіткому розподілу відповідальності між компонентами, що забезпечує зручність в обслуговуванні та модульність коду. Розробка моделей даних і бази даних за допомогою Room дозволяє ефективно зберігати й обробляти інформацію про активність користувача. Це, в свою чергу, забезпечує стабільну роботу застосунку та можливість його подальшого розширення. У цілому, розробка аналітичного крокоміра на основі зазначених інструментів і підходів дозволяє створити продуктивний і зручний застосунок для користувачів.

4 ОСНОВНІ РІШЕННЯ ЩОДО РЕАЛІЗАЦІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

У цьому розділі розглянуто основні рішення, що були прийняті під час розробки програмного забезпечення для крокоміра. Описано, як були організовані основні функціональні та технічні аспекти, а також обґрунтовані вибрані підходи до реалізації застосунку з урахуванням вимог до точності вимірювань та зручності використання.

4.1 Порядок роботи застосунку-крокоміра

При відкритті застосунку ініціалізуються всі необхідні компоненти, що забезпечують коректну роботу програми. Це включає налаштування бази даних для збереження інформації користувача та кроків, а також сервіси, відповідальні за підрахунок кроків. Крім того, ініціалізується ViewModel, який відповідає за відображення даних на екрані, таких як інформація про користувача та досягнення.

Перше, що виконується при запуску застосунку — це запит дозволу на використання функціональності для відстеження активності користувача, зокрема, дозволу на доступ до даних про активність (ACTIVITY_RECOGNITION). Після цього запускається фоновий сервіс, який відповідає за підрахунок кроків в режимі реального часу. Використовується сервіс, що працює у передньому плані (startForegroundService), для забезпечення стабільної роботи фонових процесів, навіть коли застосунок знаходиться в фоні або вимкнений.

Наступним етапом є перевірка, чи є користувач зареєстрованим. Для цього у ViewModel здійснюється перевірка за допомогою методів, які взаємодіють з репозиторіями для отримання даних користувача та досягнень. Якщо користувач не зареєстрований, його перенаправляють на екран реєстрації, а якщо зареєстрований — на головний екран.

Для підрахунку кількості кроків у фоновому режимі було реалізовано сервіс `StepsService`, який використовує `SensorManager` для отримання даних від акселерометра в реальному часі. Цей сервіс працює як фоновий сервіс, щоб забезпечити неперервний моніторинг кроків, навіть коли застосунок закритий. Кожен новий крок обробляється і зберігається в локальній базі даних за допомогою бібліотеки `Room`.

Першим етапом є ініціалізація сенсора. При створенні сервісу ініціалізується `SensorManager`, і вибирається сенсор для визначення кроків типу `Sensor.TYPE_STEP_DETECTOR`. Після цього реєструється слухач подій для цього сенсора. Надалі йде обробка даних сенсора. Коли сенсор фіксує зміну, метод `onSensorChanged` отримує дані про кількість кроків, що були зроблені користувачем. Ці дані передаються до репозиторію `StepRepository`, де зберігаються у базі даних. Кожен об'єкт містить кількість кроків та мітку часу. Сервіс працює в передньому плані через виклик `startForegroundService`, що дозволяє йому продовжувати працювати навіть при закритті застосунку. Це необхідно для виконання задачі, яка не повинна перериватися. Для цього створюється сповіщення, яке інформує користувача про роботу сервісу. Коли сервіс зупиняється або закривається, виконується відключення слухача сенсора та скасування всіх активних корутин.

У застосунку користувач може відстежувати поточний прогрес по кроках, що дозволяє їм бачити кількість кроків, зроблених протягом дня, а також інші метрики, такі як витрачені калорії та пройденої відстань. На головному екрані (`HomeScreen`) користувач бачить кількість кроків, зроблених за поточний день, а також інші показники, такі як ціль по кроках, калорії та відстань. Ці дані отримуються з репозиторіїв за допомогою `StepRepository` і оновлюються в реальному часі через `StateFlow`. Користувач може також отримати доступ до аналітики і досягнень через відповідні елементи інтерфейсу.

У розділі “Аналітика” користувач має змогу переглядати детальну статистику щодо своєї фізичної активності за різні періоди часу: тиждень, місяць і рік. Даний функціонал забезпечується класом `AnalyticsViewModel`, який витягує

дані з репозиторіїв і здійснює необхідні обчислення для визначення таких метрик, як кількість пройдених кроків, витрачені калорії та подолана відстань. Це дозволяє користувачеві отримати не лише загальну статистику, а й тренди його активності за обраний період часу.

У розділі “Досягнення” користувачі можуть відслідковувати свої нагороди за досягнення певних цілей, наприклад, подолання 10 000 кроків на день. Відповідальність за перевірку нових досягнень та їх збереження в застосунку покладається на клас `AchievementsManager`. Це забезпечує динамічну перевірку стану досягнень користувача, залежно від їх активності, кроків, пройденої відстані та витрачених калорій.

У розділі “Профіль” застосунку користувач може редагувати свої особисті дані, такі як ім'я, вік, вага, зріст та мета кроків. Ці зміни синхронізуються з локальним сховищем даних, що дозволяє миттєво відображати їх у відповідних розділах програми. У разі зміни даних, таких як вага чи зріст, оновлені значення відображаються в основних екранах, що дозволяє користувачу зберігати актуальну інформацію.

4.2 Технічна реалізація аналітичних функцій

Відображення прогресу користувача — ключова функція аналітичної частини застосунка. На основі зібраних даних здійснюється розрахунок наступних показників: кількість кроків, витрачені калорії, пройдена дистанція, а також їх візуалізація у вигляді графіків та статистики за різні періоди часу.

Розрахунок ключових показників:

— кількість кроків за день зберігається в базі даних за допомогою методу `insert()` у класі `StepRepository`. Для отримання щоденних, щотижневих, щомісячних та щорічних показників використовуються методи: `getDailySteps()`, `getStepsByWeek()`, `getStepsByMonth()`, `getStepsByYear()`. Метод `calculateSteps()` у `AnalyticsViewModel` викликає відповідні методи репозиторію для обчислення загальної статистики;

– витрати калорій залежать від кількості кроків та ваги користувача. Розрахунок здійснюється за формулою (4.1), яка враховує середнє значення витрат енергії на крок:

$$Calories = steps \times CALORIES_PER_STEP, \quad (4.1)$$

де *Calories* — кількість спалених калорій;

steps — кількість кроків;

CALORIES_PER_STEP — коефіцієнт витрати калорій на один крок, що дорівнює 0,04.

Більшість наукових досліджень вказують, що середній показник спалювання калорій під час ходьби становить від 0,03 до 0,06 калорій на крок. Це залежить від різних факторів, таких як вага людини, швидкість ходьби та рівень фізичної підготовки. Значення 0,04 було обрано як середнє, що є найбільш універсальним і підходить для широкого кола користувачів. Метод `calculateCalories()` у `AnalyticsViewModel` використовує `ActivityMetricsCalculator` для розрахунків: `calculator.calculateCalories(steps.toLong());`

– відстань розраховується на основі кількості кроків та зросту користувача. Зріст впливає на середню довжину кроку.

Розрахування проводиться за формулою (4.2):

$$Distance = (steps \times (height/100 \times STEP_LENGTH_RATIO)) / 1000, \quad (4.2)$$

де *Distance* — пройдена відстань у кілометрах;

steps — кількість кроків;

height — зріст користувача в сантиметрах;

STEP_LENGTH_RATIO — коефіцієнт, який визначає довжину кроку як частку від зросту, значення 0,413.

Метод `calculateDistance()` викликає функцію `calculateDistanceInKm()`: `calculator.calculateDistanceInKm(userHeight, steps.toLong())`

Для візуалізації прогресу використовуються структури `StateFlow`, які забезпечують реактивне оновлення UI при зміні даних. Відповідні дані для кожного періоду (день, тиждень, місяць, рік) зберігаються в змінних:

- `stepsByWeek`, `caloriesByWeek`, `distanceByWeek` — для тижневих даних;
- `stepsByMonth`, `caloriesByMonth`, `distanceByMonth` — для місячних даних;
- `stepsByYear`, `caloriesByYear`, `distanceByYear` — для річних даних.

UI автоматично підписується на ці потоки і відображає оновлену статистику у вигляді графіків та списків.

Для зберігання даних про кроки використовується `Room`, яка забезпечує зручний доступ до даних через DAO. `StepDao` містить методи для зчитування та збереження даних:

- `getStepsForDay()` — отримання даних за конкретний день;
- `getStepsForMonth()` — отримання даних за місяць;
- `getStepsForYear(year: String)` — отримання даних за рік;
- `insertOrUpdateDailySteps()` — збереження даних для конкретної години;
- `updateTotalSteps()` — оновлення загальної кількості кроків.

Особисті дані користувача та цілі зберігаються у `DataStore`, яка забезпечує асинхронний та надійний спосіб роботи з налаштуваннями. Реалізація `StepUpPreferencesRepository` дозволяє зберігати і отримувати дані користувача (зріст, вага, ціль по кроках), а також зберігати досягнення у форматі JSON.

Ключові методи:

- `setUserData(userData: User)` — збереження даних користувача;
- `getUserData()` — отримання даних користувача;
- `setAchievements(list: List<Achievement>)` — збереження досягнень;
- `getAchievements()` — отримання списку досягнень;

Обробка історичних даних відбувається у `StepRepository`: Дані за день обробляються методом `processDailySteps()`. Якщо дата змінилась, викликається `clearDailySteps()`, що архівує щоденні дані. Дані за місяць і рік обробляються аналогічно методами `processMonthlySteps()` та `updateYearlySteps()`. Агрегація даних для щоденних, тижневих, місячних і річних звітів відбувається за

допомогою реактивних потоків (Flow). Це дозволяє уникнути повторного зчитування великих обсягів даних і забезпечує швидкий доступ до статистики.

Реалізація аналітичних функцій включає в себе збір, обробку та візуалізацію даних про активність користувача. Завдяки поєднанню Room для історичних даних і DataStore для налаштувань, застосунок забезпечує гнучкість і високу продуктивність у зберіганні та обробці інформації.

4.3 Висновки за розділом 4

У четвертому розділі детально розглянуто технічні рішення, прийняті під час розробки крокоміра. Основну увагу приділено організації роботи застосунку, обробці та збереженню даних, а також їхній візуалізації для користувача. Було забезпечено інтуїтивно зрозумілий процес реєстрації, який включає збір персональних даних для персоналізації. Основний інтерфейс застосунку надає користувачу доступ до актуальних показників активності в реальному часі і забезпечує навігацію до розширених розділів: аналітики та досягнень.

Для обробки та збереження даних було використано Room для роботи з історичними даними та DataStore для зберігання налаштувань користувача. Взаємодія з сенсорами фізичної активності реалізована через SensorManager для забезпечення точного підрахунку кроків. Реалізовано функції для збору, обробки та відображення статистики активності за різні періоди (день, тиждень, місяць, рік). Візуалізація даних досягнута за допомогою StateFlow, що забезпечує динамічне оновлення інтерфейсу відповідно до змін даних.

Застосунок працює офлайн, дозволяючи користувачу переглядати історичні дані та досягнення без підключення до інтернету. Застосунок автоматично синхронізує дані при наступних запусках, що сприяє точності довгострокового аналізу. Таким чином, реалізовані функціональні можливості застосунку забезпечують високу точність обчислень, інтуїтивність інтерфейсу та надійність збереження даних. Це дозволяє користувачам ефективно відстежувати прогрес у досягненні своїх фітнес-цілей.

5 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРЕМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

5.1 Опис роботи застосунку

При першому запуску застосунок запитує дозвіл на доступ до даних про фізичну активність, рис. 5.1. Цей крок необхідний для забезпечення коректного підрахунку кроків і отримання інформації з датчиків руху.

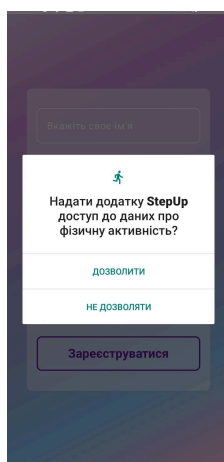
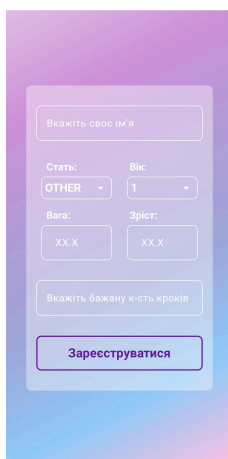


Рисунок 5.1 – Скріншот застосунку з дозволом доступу до даних

Після надання дозволів користувач переходить на екран реєстрації, рис. 5.2. Тут потрібно ввести особисті дані: ім'я, стать, вік, вагу, зріст і бажану кількість кроків на день, рис. 5.3. Ці дані використовуються для персоналізації програми, розрахунку витрачених калорій та інших показників.

Заповнення форми завершується натисканням кнопки «Зареєструватися», після чого дані зберігаються у внутрішній базі застосунку. Після реєстрації користувач автоматично переходить на головний екран.

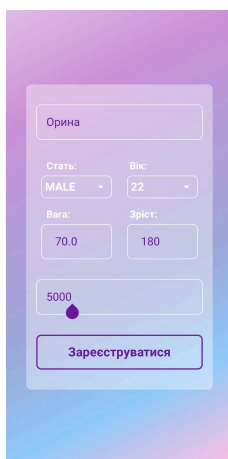
На головному екрані, рис. 5.4, відображається загальна кількість кроків, пройдених за день, витрачені калорії та пройдена дистанція. Для збору цих даних застосунок використовує акселерометр і сенсори фізичної активності пристрою. Дані обробляються в реальному часі, забезпечуючи точний підрахунок кроків.



Registration form with the following fields and options:

- Text input: "Вкажіть своє ім'я"
- Gender: "Стать:" with dropdown menu "OTHER"
- Age: "Вік:" with dropdown menu "1"
- Weight: "Вага:" with input "XX.X"
- Height: "Зріст:" with input "XX.X"
- Text input: "Вкажіть бажану к-сть кроків"
- Button: "Зареєструватися"

Рисунок 5.2 – Скріншот застосунку, екран реєстрації



Registration form with the following filled data:

- Text input: "Орина"
- Gender: "Стать:" with dropdown menu "MALE"
- Age: "Вік:" with dropdown menu "22"
- Weight: "Вага:" with input "70.0"
- Height: "Зріст:" with input "180"
- Text input: "5000" (with a slider indicator)
- Button: "Зареєструватися"

Рисунок 5.3 – Скріншот застосунку з заповненими даними для реєстрації

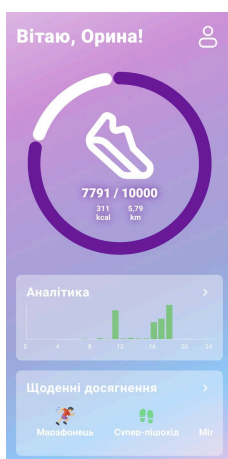


Рисунок 5.4 – Скріншот застосунку головний екран

Окрім цього, є доступ до розширеної аналітики. Наприклад, аналіз кроків по годинах протягом дня дозволяє користувачам бачити, коли вони були найбільш активними. З головного екрану також можна перейти до меню досягнень.

Застосунок має кілька важливих розділів.

Профіль: На цьому екрані користувач може змінювати свої дані (ім'я, стать, вік, вагу, зріст, цільову кількість кроків) у разі зміни параметрів або цілей, рис. 5.5.

Статистика: У цьому розділі представлена аналітика кроків, калорій і дистанції за різні періоди (тиждень, місяць, рік). Вона розбита на три підрозділи:

- кількість кроків, рис 5.6;
- витрачені калорії, рис. 5.7;
- пройдена дистанція, рис. 5.8.

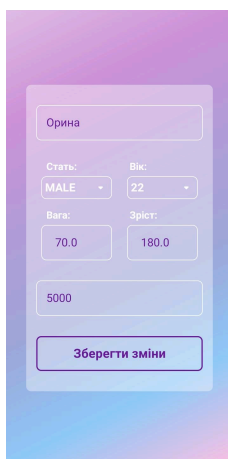


Рисунок 5.5 – Скріншот застосунку, екран профілю для змін даних

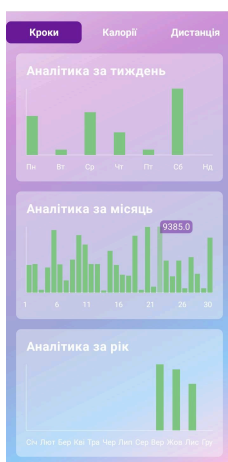


Рисунок 5.6 – Скріншот застосунку, екран аналітики кроків

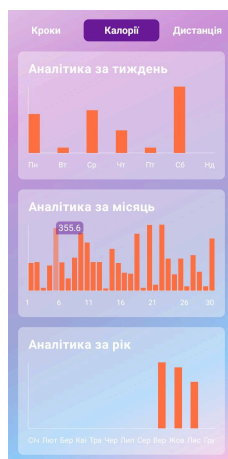


Рисунок 5.7 – Скріншот застосунку, екран аналітики калорій

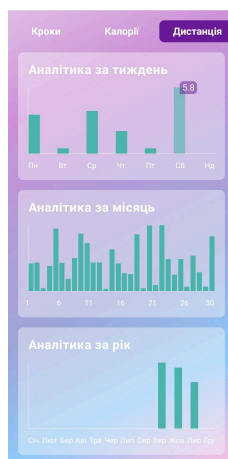


Рисунок 5.8 – Скріншот застосунку, екран аналітики дистанції

Досягнення: Тут відображаються всі досягнення користувача. Досягнення, які вже виконано, мають зелений фон, а невиконані — сірий, рис. 5.9. При виборі досягнення відкривається детальна інформація, рис. 5.10.

Усі введені користувачем дані, а також статистика активності зберігаються у внутрішній базі даних та кеші програми. Це дозволяє застосунку працювати офлайн, а також забезпечує швидкий доступ до історії користувача. Дані синхронізуються при наступних запусках і можуть бути використані для аналізу довгострокових трендів. Завдяки цьому користувачі можуть відслідковувати свій прогрес і досягати нових цілей.

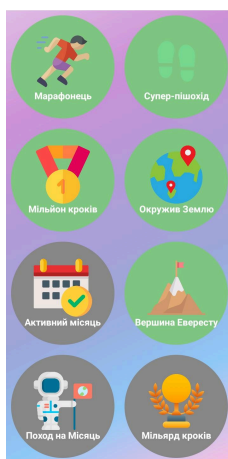


Рисунок 5.9 – Скрішот застосунку, екран досягнень

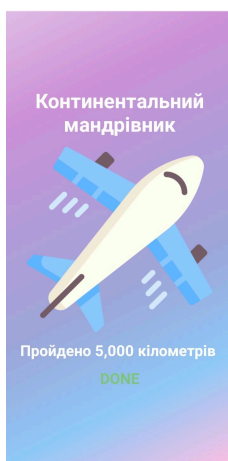


Рисунок 5.10 – Скріншот застосунку, екран детальної інформації про досягнення

5.2 Тестування та перевірка точності підрахунку кроків

Однією з ключових вимог до застосунку є точність підрахунку кроків, адже основною метою є допомога користувачам в моніторингу їх фізичної активності. Для перевірки точності алгоритму підрахунку кроків було проведено серію тестів, які включали використання застосунку в порівнянні з іншими популярними застосунками для вимірювання фізичної активності, які були проаналізовані у пункті 1.1 данної роботи. Це дозволило оцінити відхилення в кількості підрахованих кроків та визначити рівень точності застосунку.

Для проведення тестування було обрано кілька популярних застосунків для підрахунку кроків, серед яких були як вбудовані функції в смартфонах, так і

сторонні програми, які мають хорошу репутацію за точність підрахунку кроків. Двоє користувачів, що брали участь у тестуванні, використовували різні моделі смартфонів, щоб перевірити, чи залежить точність підрахунку від моделі пристрою.

Для порівняння результатів тестування обирали декілька різних сценаріїв:

- прогулянка на 1000 кроків. Учасники тестували підрахунок на невеликій дистанції, щоб оцінити точність за короткий проміжок часу;
- більш тривала активність. Учасники тестували підрахунок на відстані 5000 кроків, що дозволило оцінити точність під час більш інтенсивної фізичної активності;
- складні умови. Тестування проводилось також на нерівних поверхнях, при ходьбі вгору чи вниз, щоб оцінити, чи правильно застосунок фіксує кроки за різних умов.

Результати тестування показали, що розбіжність у кількості підрахованих кроків між нашим застосунком та іншими застосунками була незначною і складала в межах 1-2%. Це підтверджує високий рівень точності підрахунку кроків у застосунку.

Ось деякі приклади результатів тестування.

Тест 1: 1000 кроків, результати наведено у табл. 5.1. У цьому випадку різниця між результатами нашого застосунку та іншими застосунками становила 1,8% у порівнянні з Застосунком А та 1,4% менше у порівнянні з Застосунком В. Це показує, що відхилення в межах 2% є допустимим для такого типу застосунків.

Таблиця 5.1 – Результати тесту на 1000 кроків

Дані вимірювання	StepUp	Застосунок А (сторонній)	Застосунок В (вбудований)
Кількість кроків	1000	1018	986

Тест 2: 5000 кроків, результати наведено у табл. 5.2. Різниця в 39 кроків (0,82%) між нашим застосунком і Застосунком А та 58 кроків (1.16%) між нашим застосунком і Застосунком В також демонструє високий рівень точності.

Таблиця 5.2 – Результати тесту на 5000 кроків

Дані вимірювання	StepUp	Застосунок А (сторонній)	Застосунок В (вбудований)
Кількість кроків	4994	5035	4936

Тест 3: Активність на нерівній поверхні (2000 кроків), результати наведені у табл. 5.3. Навіть за умов ходьби по нерівній поверхні, наш застосунок показав точність, що не відрізняється більше ніж на 1% від інших програм.

Таблиця 5.3 – Результати тесту активності на нерівній поверхні

Дані вимірювання	StepUp	Застосунок А (сторонній)	Застосунок В (вбудований)
Кількість кроків	1997	2011	1988

Відхилення в підрахунку кроків можуть бути спричинені кількома факторами:

- алгоритм обробки даних. Кожен застосунок використовує свій власний алгоритм для фіксації руху, що може впливати на точність підрахунку;
- чутливість сенсора. Різні пристрої можуть мати сенсори різної чутливості, що може впливати на точність підрахунку, особливо при швидкій ходьбі або бігу;
- тип поверхні. Ходьба по різних поверхнях (гладка, нерівна, підйом чи спуск) може впливати на кількість зафіксованих кроків в залежності від точності алгоритму детекції руху.

Попри це, наш застосунок продемонстрував достатню точність, з розбіжностями в межах 1-2%, що є цілком прийнятним для користувачів, які використовують застосунок для загального моніторингу фізичної активності.

5.3 Висновки за розділом 5

Тестування та перевірка точності підрахунку кроків показали, що наш застосунок демонструє високу точність під час вимірювання фізичної активності. Різниця в кількості підрахованих кроків між нашим застосунком та іншими популярними застосунками для вимірювання кроків не перевищує 2%, що є мінімальним відхиленням для подібного роду програм.

Згідно з результатами тестування, найбільші розбіжності спостерігалися при вимірюванні кроків на більшій відстані, однак навіть у таких випадках відхилення не перевищувало 1-2%. Це означає, що застосунок працює надійно як для коротких прогулянок, так і для більш тривалих фізичних активностей. Крім того, навіть за складних умов, таких як ходьба по нерівній поверхні, застосунок зміг зберегти точність, не відхиляючись більше ніж на 1% від інших застосунків.

Ці результати підтверджують, що застосунок є достатньо точним засобом для відстеження фізичної активності, забезпечуючи користувачам правильні дані для досягнення їхніх цілей у фітнесі.

ВИСНОВКИ

Основний висновок до дипломної роботи полягає в тому, що розроблене програмне забезпечення для крокоміра на базі акселерометра мобільного пристрою є потужним інструментом для моніторингу фізичної активності користувачів, який забезпечує високу точність вимірювань та зручний інтерфейс для аналізу даних. Використання сучасних технологій, зокрема мови програмування Kotlin, середовища розробки Android Studio, а також фреймворків Jetpack Compose і SensorManager, дозволяє досягти ефективності, точності та енергоефективності в реалізації функціоналу.

Процес розробки програмного забезпечення включав кілька ключових етапів: збір, обробка та збереження даних акселерометра, що дозволяє точно реєструвати фізичну активність користувача в реальному часі. Алгоритми обробки даних, такі як фільтрація та виявлення піків, дозволяють чітко визначати важливі рухи та підраховувати кроки навіть за різних умов фізичної активності. Зберігання даних у структурованій базі даних Room дозволяє ефективно організувати доступ до історії активності, що дає користувачам змогу моніторити свій прогрес та отримувати корисну аналітику щодо своїх досягнень.

Особливу увагу в роботі було приділено проектуванню зручного інтерфейсу користувача. Завдяки використанню Jetpack Compose та архітектури MVVM, застосунок має інтуїтивно зрозуміле середовище для перегляду поточної активності, аналізу статистики та досягнень, що дозволяє користувачам активно взаємодіяти з застосунком та отримувати мотивацію для досягнення фітнес-цілей. Візуалізація даних через графіки та діаграми надає наочне відображення прогресу в реальному часі, що є важливим для користувачів, які прагнуть досягти кращих результатів у своєму фізичному розвитку.

Тестування точності підрахунку кроків продемонструвало, що розроблене програмне забезпечення має мінімальні відхилення від даних інших популярних застосунків, що свідчить про високу надійність та ефективність під час різних фізичних активностей, таких як короткі прогулянки чи більш тривалі тренування.

Порівняння з іншими програмами показало, що різниця в підрахунку кроків не перевищує 2%, що є дуже точним результатом для мобільних застосунків такого типу. В результаті, розроблене програмне забезпечення для крокометра не тільки демонструє високу точність і ефективність, але й має великий потенціал для подальшого вдосконалення. Існує можливість інтегрувати додаткові функції, такі як штучний інтелект для покращення прогнозування активності та адаптації до індивідуальних особливостей користувача, що дозволить підвищити точність вимірювань і зробити застосунок ще більш персоналізованим.

Таким чином, розроблене програмне забезпечення має практичне значення в контексті сфери здоров'я та фітнесу, оскільки допомагає користувачам стежити за рівнем своєї фізичної активності, аналізувати власні досягнення та покращувати здоров'я. Застосунок може бути інтегрований в більш складні системи моніторингу здоров'я, а також служити основою для майбутніх досліджень і розробок у галузі мобільних технологій для покращення фізичної активності та здоров'я користувачів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Leap Fitness Group. Step Counter - Pedometer [Electronic resource]. – Access mode:
<https://play.google.com/store/apps/details?id=pedometer.steptracker.calorieburner.stepcounter>.
2. Tech H. Step Counter - Pedometer [Electronic resource]. – Access mode:
<https://play.google.com/store/apps/details?id=stepcounter.pedometer.stepstracker&hl=ua>.
3. Inc I. T. Pedometer - Step Counter App [Electronic resource]. – Access mode:
<https://play.google.com/store/apps/details?id=com.tayu.tau.pedometer>.
4. Hanlon M. Real-time gait event detection using wearable sensors [Electronic resource]. – Access mode:
<https://www.sciencedirect.com/science/article/abs/pii/S0966636209003373>.
5. Accelerometer Based Joint Step Detection and Adaptive Step Length Estimation Algorithm Using Handheld Devices [Text] / [Yu Liu, Yanping Chen, Lili Shi et al.] // Journal of Communications. – 2015. – Vol. 10, № 7.
6. Kumar S. Mobile Health: Sensors, Analytic Methods, and Applications [Text] / Santosh Kumar, James M. Rehg, Susan A. Murphy. – Springer, 2017. – 582 p.
7. SensorManager [Electronic resource]. – Access mode:
<https://developer.android.com/reference/android/hardware/SensorManager>.
8. Sensors Overview [Electronic resource]. – Access mode:
https://developer.android.com/develop/sensors-and-location/sensors/sensors_overview.
9. Phillips B. Android Programming: The Big Nerd Ranch Guide [Text] / [B. Phillips, C. Stewart, B. Hardy, K. Marsicano]. – Pearson Education, Limited, 2022.
10. Orosz G. Building Mobile Apps at Scale: 39 Engineering Challenges [Text] / G. Orosz. – Primedia E-launch LLC, 2021. – 236 p.
11. Kotlin Documentation: Kotlin Programming Language [Electronic resource]. – Access mode: <https://kotlinlang.org/docs/home.html>.
12. Екерт Дж. Професійне програмування на Kotlin / Дж. Екерт, М. Чират. —

К.: Бібат, 2020. — 432 с.

13. Android Studio Documentation: The official IDE for Android [Electronic resource]. — Access mode: <https://developer.android.com/studio>.

14. Jetpack Compose Documentation: Build better apps faster with Jetpack Compose [Electronic resource]. — Access mode: <https://developer.android.com/jetpack/compose>.

15. SensorManager API Guide: Sensors overview [Electronic resource]. — Access mode: https://developer.android.com/guide/topics/sensors/sensors_overview.

16. Hilt Documentation: Simplify dependency injection with Hilt [Electronic resource]. — Access mode: <https://developer.android.com/training/dependency-injection/hilt-android>.

17. Dagger Documentation: A fast dependency injector for Android and Java [Electronic resource]. — Access mode: <https://dagger.dev/>.

18. Room Documentation: Save data in a local database using Room [Electronic resource]. — Access mode: <https://developer.android.com/training/data-storage/room>.

19. SQLite Documentation: The SQLite Database Engine [Electronic resource]. — Access mode: <https://sqlite.org/docs.html>.

20. Соловійов, А. А. Основи програмування на Android / А. А. Соловійов. — Київ: Видавництво "Наука", 2020. — 256 с.

21. Шаїн, В. В. Архітектура мобільних додатків: основи проєктування та розробки / В. В. Шаїн. — Харків: Видавництво "Літера", 2019. — 210 с.

22. Пономаренко, А. С. Основи проєктування баз даних / А. С. Пономаренко. — Харків: Видавництво "Технологія", 2017. — 210 с.

23. Петров, А. С. Базы даних: основи проєктування та аналізу / А. С. Петров. — Київ: Либідь, 2016. — 320 с.

24. Ребров, І. В. Основи проєктування інформаційних систем / І. В. Ребров. — Київ: Наукова думка, 2017. — 280 с.

25. Глібов, В. М. Програмування баз даних / В. М. Глібов. — Львів: Видавництво "Світ", 2014. — 300 с.

ДОДАТОК А
Технічне завдання

Вступ

У роботі розглядається галузь розробки програмного забезпечення, що пов'язана з дослідженням та програмною реалізацією методів моніторингу фізичної активності за допомогою мобільного застосунку на основі даних акселерометра. Зокрема, у рамках дипломної роботи здійснюється розробка програмного продукту для підрахунку кроків та аналізу фізичної активності користувачів.

A.1 Підстава для розробки

Підставою для розробки є завдання на дипломний проєкт на тему «Дослідження та програмна реалізація аналітичного застосунку - крокоміра з використанням акселерометра», затверджене наказом по національному університету «Запорізька політехніка», протокол № 398 від 7 жовтня 2024 р.

A.2 Призначення розробки

Програмне забезпечення StepUp призначене для моніторингу фізичної активності користувачів, зокрема підрахунку кількості кроків, аналізу рівня активності та відображення досягнень користувача. Застосунок буде використовувати дані акселерометра мобільного пристрою для точного вимірювання кроків і зберігання інформації про активність у базі даних для подальшого аналізу. Користувачі зможуть переглядати графіки, статистику та досягнення, а також налаштовувати свої цілі щодо фізичної активності.

A.3 Основні вимоги до програмного забезпечення, що розробляється

A.3.1 Вимоги до функціональних характеристик

Програмне забезпечення повинно забезпечувати наступні функції:

- підрахунок кроків за допомогою акселерометра мобільного пристрою;
- збереження даних про фізичну активність користувача в базі даних;
- відображення статистики про активність користувача (щоденні, тижневі, місячні звіти);
- налаштування мети по кількості кроків на день;
- відображення досягнень (наприклад, досягнення певної кількості кроків);
- підтримка української та англійської мов.

A.3.2 Вимоги до надійності

Програма повинна забезпечувати:

- високу точність підрахунку кроків і моніторингу активності;
- надійне збереження даних користувача без втрат при перезавантаженні пристрою або інших непередбачених ситуаціях;
- обробку виключних ситуацій, таких як відключення сенсорів або помилки доступу до бази даних;
- стабільну роботу на різних мобільних пристроях з операційною системою Android.

A.3.3 Умови експлуатації

Програмний продукт буде використовуватись на мобільних пристроях з операційною системою Android 10 та вище. Користувачі повинні мати доступ до інтернету для завантаження та оновлення застосунку.

A.3.4 Вимоги до складу і параметрів технічних засобів

Для нормальної роботи програми на мобільних пристроях необхідно:

- операційна система Android 10 або новіша;
- процесор з тактовою частотою не нижче 1.8 ГГц;
- 2 ГБ оперативної пам'яті або більше;
- наявність акселерометра для вимірювання кроків.

A.4 Порядок контролю та приймання

Строки виконання дипломної кваліфікаційної роботи магістра визначені в завданні на роботу безпосередньо, включаючи строк подання роботи, календарний план виконання роботи з зазначеними етапами, результатами та строками. Захист роботи відбувається за оголошеним завчасно графіком захисту дипломних кваліфікаційних робіт і відповідно графіком роботи екзаменаційної комісії.

ДОДАТОК Б
Текст програми

```

Constants.kt
package com.app.stepup.constants

import com.app.stepup.R
import com.app.stepup.model.data.Achievement
import com.app.stepup.model.data.Gender
import com.app.stepup.model.data.Name
import com.app.stepup.model.data.Status
import com.app.stepup.model.data.User

object Constants {
    const val MAIN = "MAIN"
    const val REGISTRATION = "REGISTRATION"
    const val HOME = "HOME"
    const val PROFILE = "PROFILE"
    const val ANALYTICS = "ANALYTICS"
    const val ACHIEVEMENTS = "ACHIEVEMENTS"
    const val ACHIEVEMENT = "ACHIEVEMENT"

    const val STEPS_COUNTER_CHANNEL_ID = "step_counter_channel"
    const val STEPS_COUNTER_CHANNEL_NAME = "Step Counter"
    const val STEPS_COUNTER_NOTIFICATION_ID = 1

    const val DATABASE_NAME = "step_database"
    const val DATA_STORE_NAME = "step_up_preferences"

    const val KEY_ACHIEVEMENT = "key_achievement"

    const val MARATHON_DISTANCE = 42195
    const val ONE_HUNDRED_THOUSAND = 100000
    const val MILLION = 1000000
    const val CIRCLED_THE_WORLD = 50000000
    const val ACTIVE_MONTH = 300000
    const val EVEREST_SUMMIT = 58070
    const val TRIP_TO_THE_MOON = 478000000
    const val BILLION = 1000000000
    const val BURNED_PIZZA = 300
    const val BURNED_CAKE = 2000
    const val DAILY_NORM = 2500
    const val MONTH_WORTH = 75000
    const val BURNED_BIG_MAC = 550

```

```

const val CIRCLED_THE_WORLD_TWICE = 80000
const val FIVE = 5
const val TEN = 10
const val FIFTY = 50
const val ONE_HUNDRED = 100
const val TWO_HUNDRED = 200
const val ONE_THOUSAND = 1000
const val FIVE_THOUSAND = 5000
const val TEN_THOUSAND = 10000

val emptyUser = User(
    "", 0, Gender.OTHER, 0.0, 0.0
)

val defaultAchievements = listOf(
    Achievement(
        R.drawable.ic_running,
        Name.MARATHONER,
        R.string.marathoner_descr,
        Status.UNDONE
    ),
    Achievement(
        R.drawable.ic_footprints,
        Name.SUPER_WALKER,
        R.string.super_walker_descr,
        Status.UNDONE
    ),
    Achievement(
        R.drawable.ic_medal,
        Name.MILLION_STEPS,
        R.string.million_steps_descr,
        Status.UNDONE
    ),
    Achievement(
        R.drawable.ic_globe,
        Name.CIRCLED_THE_EARTH,
        R.string.circled_the_earth_descr,
        Status.UNDONE
    ),
    Achievement(
        R.drawable.ic_calendar,

```

```

        Name.ACTIVE_MONTH,
        R.string.active_month_descr,
        Status.UNDONE
    ),
    Achievement(
        R.drawable.ic_mountain,
        Name.EVEREST_SUMMIT,
        R.string.everest_summit_descr,
        Status.UNDONE
    ),
    Achievement(
        R.drawable.ic_astronaut,
        Name.TRIP_TO_THE_MOON,
        R.string.trip_to_the_moon_descr,
        Status.UNDONE
    ),
    Achievement(
        R.drawable.ic_trophy,
        Name.BILLION_STEPS,
        R.string.billion_steps_descr,
        Status.UNDONE
    ),
    Achievement(
        R.drawable.ic_pizza,
        Name.BURNED_PIZZA,
        R.string.burned_pizza_descr,
        Status.UNDONE
    ),
    Achievement(
        R.drawable.ic_torch,
        Name.MINI_MARATHON,
        R.string.mini_marathon_of_calories_descr,
        Status.UNDONE
    ),
    Achievement(
        R.drawable.ic_cake,
        Name.BURNED_CAKE,
        R.string.burned_cake_descr,
        Status.UNDONE
    ),
    Achievement(

```

```

        R.drawable.ic_eat,
        Name.BURNED_DAILY_NORM,
        R.string.burned_daily_norm_descr,
        Status.UNDONE
    ),
    Achievement(
        R.drawable.ic_metabolism,
        Name.TEN_K_CALORIES,
        R.string.ten_k_calories_descr,
        Status.UNDONE
    ),
    Achievement(
        R.drawable.ic_calendar,
        Name.MONTH_WORTH_CALORIES,
        R.string.burned_month_worth_calories_descr,
        Status.UNDONE
    ),
    Achievement(
        R.drawable.ic_metabolism,
        Name.ONE_HUNDRED_THOUSAND_CALORIES,
        R.string.one_hundred_thousand_calories_descr,
        Status.UNDONE
    ),
    Achievement(
        R.drawable.ic_burger,
        Name.BURNED_BIG_MAC,
        R.string.burned_big_mac_descr,
        Status.UNDONE
    ),
    Achievement(
        R.drawable.ic_trophy,
        Name.MILLION_CALORIES,
        R.string.million_calories_descr,
        Status.UNDONE
    ),
    Achievement(
        R.drawable.ic_footprints,
        Name.FIVE_KM_ON_FOOT,
        R.string.five_km_on_foot_descr,
        Status.UNDONE
    ),

```

```

Achievement(
    R.drawable.ic_footprints,
    Name.GOLDEN_TEN,
    R.string.golden_ten_descr,
    Status.UNDONE
),
Achievement(
    R.drawable.ic_compass,
    Name.FIFTY_KM_OF_JOURNEY,
    R.string.fifty_km_journey_descr,
    Status.UNDONE
),
Achievement(
    R.drawable.ic_office,
    Name.WALKING_TO_WORK,
    R.string.walking_to_work_descr,
    Status.UNDONE
),
Achievement(
    R.drawable.ic_map,
    Name.TWO_HUNDRED_KM_WEEKEND,
    R.string.two_hundred_km_weekend_descr,
    Status.UNDONE
),
Achievement(
    R.drawable.ic_distance,
    Name.ONE_THOUSAND_KM_IN_YEAR,
    R.string.one_thousand_km_in_year_descr,
    Status.UNDONE
),
Achievement(
    R.drawable.ic_airplane,
    Name.CONTINENTAL_TRAVELER,
    R.string.continental_traveler_descr,
    Status.UNDONE
),
Achievement(
    R.drawable.ic_pedometer,
    Name.ULTRA_HIKE,
    R.string.ultra_hike_descr,
    Status.UNDONE

```

```

    ),
    Achievement(
        R.drawable.ic_global,
        Name.CIRCLED_THE_EARTH_TWICE,
        R.string.circled_the_earth_twice_descr,
        Status.UNDONE
    )
)
}

```

DatabaseModule.kt

```
package com.app.stepup.di
```

```

import android.content.Context
import androidx.room.Room
import com.app.stepup.constants.Constants.DATABASE_NAME
import com.app.stepup.model.StepRepository
import com.app.stepup.model.room.MIGRATION_1_2
import com.app.stepup.model.room.MIGRATION_2_3
import com.app.stepup.model.room.MIGRATION_3_4
import com.app.stepup.model.room.MIGRATION_4_5
import com.app.stepup.model.room.StepDao
import com.app.stepup.model.room.StepDatabase
import com.app.stepup.utils.DateParser
import dagger.Module
import dagger.Provides
import dagger.hilt.InstallIn
import dagger.hilt.android.qualifiers.ApplicationContext
import dagger.hilt.components.SingletonComponent
import javax.inject.Singleton

```

```
@Module
```

```
@InstallIn(SingletonComponent::class)
```

```
object DatabaseModule {
```

```
    @Provides
```

```
    @Singleton
```

```

    fun provideStepDatabase(@ApplicationContext appContext: Context): StepDatabase {
        return Room.databaseBuilder(
            appContext,
            StepDatabase::class.java,
            DATABASE_NAME

```

```

        )

        .addMigrations(MIGRATION_1_2)
        .addMigrations(MIGRATION_2_3)
        .addMigrations(MIGRATION_3_4)
        .addMigrations(MIGRATION_4_5)
        .build()
    }

    @Provides
    @Singleton
    fun provideDao(stepDatabase: StepDatabase): StepDao {
        return stepDatabase.stepDao()
    }

    @Provides
    @Singleton
    fun provideStepRepository(
        @ApplicationContext appContext: Context,
        stepDao: StepDao,
        dateParser: DateParser
    ): StepRepository {
        return StepRepository(appContext, stepDao, dateParser)
    }
}

```

DataStoreModule.kt

```
package com.app.stepup.di
```

```

import android.content.Context
import androidx.datastore.core.DataStore
import androidx.datastore.preferences.core.Preferences
import androidx.datastore.preferences.preferencesDataStore
import com.app.stepup.constants.Constants.DATA_STORE_NAME
import com.app.stepup.model.datastore.StepUpPreferencesRepository
import com.app.stepup.model.datastore.StepUpPreferencesRepositoryImpl
import dagger.Binds
import dagger.Module
import dagger.Provides
import dagger.hilt.InstallIn
import dagger.hilt.android.qualifiers.ApplicationContext
import dagger.hilt.components.SingletonComponent

```

```

import javax.inject.Singleton

private val Context.dataStore by preferencesDataStore(DATA_STORE_NAME)

@Module
@InstallIn(SingletonComponent::class)
abstract class DataStoreModule {

    @Binds
    @Singleton
    abstract fun bindPreferencesRepository(
        stepUpPreferencesRepository: StepUpPreferencesRepositoryImpl
    ): StepUpPreferencesRepository

    companion object {
        @Provides
        @Singleton
        fun provideUserDataStorePreferences(
            @ApplicationContext applicationContext: Context
        ): DataStore<Preferences> {
            return applicationContext.dataStore
        }
    }
}

```

Achievement.kt

```
package com.app.stepup.model.data
```

```

data class Achievement(
    val iconRes: Int,
    val name: Name,
    val descriptionRes: Int,
    val status: Status
)

```

AnalyticsType.kt

```
package com.app.stepup.model.data
```

```

enum class AnalyticsType {
    STEPS,
    CALORIES,
}

```

```

        DISTANCE
    }

Gender.kt
package com.app.stepup.model.data

enum class Gender {
    OTHER,
    MALE,
    FEMALE
}

Name.kt
package com.app.stepup.model.data

import com.app.stepup.R

enum class Name(private val nameRes: Int) {
    MARATHONER(R.string.marathoner),
    SUPER_WALKER(R.string.super_walker),
    MILLION_STEPS(R.string.million_steps),
    CIRCLED_THE_EARTH(R.string.circled_the_earth),
    ACTIVE_MONTH(R.string.active_month),
    EVEREST_SUMMIT(R.string.everest_summit),
    TRIP_TO_THE_MOON(R.string.trip_to_the_moon),
    BILLION_STEPS(R.string.billion_steps),
    BURNED_PIZZA(R.string.burned_pizza),
    MINI_MARATHON(R.string.mini_marathon_of_calories),
    BURNED_CAKE(R.string.burned_cake),
    BURNED_DAILY_NORM(R.string.burned_daily_norm),
    TEN_K_CALORIES(R.string.ten_k_calories),
    MONTH_WORTH_CALORIES(R.string.burned_month_worth_calories),
    ONE_HUNDRED_THOUSAND_CALORIES(R.string.one_hundred_thousand_calories),
    BURNED_BIG_MAC(R.string.burned_big_mac),
    MILLION_CALORIES(R.string.million_calories),
    FIVE_KM_ON_FOOT(R.string.five_km_on_foot),
    GOLDEN_TEN(R.string.golden_ten),
    FIFTY_KM_OF_JOURNEY(R.string.fifty_km_journey),
    WALKING_TO_WORK(R.string.walking_to_work),
    TWO_HUNDRED_KM_WEEKEND(R.string.two_hundred_km_weekend),
    ONE_THOUSAND_KM_IN_YEAR(R.string.one_thousand_km_in_year),

```

```

    CONTINENTAL_TRAVELER(R.string.continental_traveler),
    ULTRA_HIKE(R.string.ultra_hike),
    CIRCLED_THE_EARTH_TWICE(R.string.circled_the_earth_twice);

    fun getNameRes(): Int {
        return nameRes
    }
}

```

Status.kt

```
package com.app.stepup.model.data
```

```
enum class Status {
    DONE,
    UNDONE
}

```

User.kt

```
package com.app.stepup.model.data
```

```
data class User(
    var name: String,
    var age: Int,
    var gender: Gender,
    var height: Double,
    var weight: Double
)

```

StepUpPreferencesRepository.kt

```
package com.app.stepup.model.datastore
```

```
import com.app.stepup.model.data.Achievement
import com.app.stepup.model.data.User
import kotlinx.coroutines.flow.Flow

```

```
interface StepUpPreferencesRepository {
    suspend fun setUserData(userData: User)
    fun getUserData(): Flow<User>
    suspend fun setStepGoal(stepGoal: Int)
    fun getStepGoal(): Flow<Int>
    suspend fun setAchievements(list: List<Achievement>)
}

```

```

        fun getAchievements(): Flow<List<Achievement>>
    }

StepUpPreferencesRepositoryImpl.kt
package com.app.stepup.model.datastore

import androidx.datastore.core.DataStore
import androidx.datastore.preferences.core.Preferences
import androidx.datastore.preferences.core.doublePreferencesKey
import androidx.datastore.preferences.core.edit
import androidx.datastore.preferences.core.intPreferencesKey
import androidx.datastore.preferences.core.stringPreferencesKey
import com.app.stepup.model.data.Achievement
import com.app.stepup.model.data.Gender
import com.app.stepup.model.data.User
import com.google.gson.Gson
import kotlinx.coroutines.flow.map
import javax.inject.Inject

class StepUpPreferencesRepositoryImpl @Inject constructor(
    private val datastore: DataStore<Preferences>
) : StepUpPreferencesRepository {

    private object PreferencesKeys {
        val USER_NAME = stringPreferencesKey("user_name")
        val USER_AGE = intPreferencesKey("user_age")
        val USER_GENDER = stringPreferencesKey("user_gender")
        val USER_HEIGHT = doublePreferencesKey("user_height")
        val USER_WEIGHT = doublePreferencesKey("user_weight")
        val STEP_GOAL = intPreferencesKey("step_goal")
        val ACHIEVEMENTS = stringPreferencesKey("achievements")
    }

    override suspend fun setUserData(userData: User) {
        datastore.edit { prefs ->
            prefs[PreferencesKeys.USER_NAME] = userData.name
            prefs[PreferencesKeys.USER_AGE] = userData.age
            prefs[PreferencesKeys.USER_GENDER] = userData.gender.name
            prefs[PreferencesKeys.USER_HEIGHT] = userData.height
            prefs[PreferencesKeys.USER_WEIGHT] = userData.weight
        }
    }
}

```

```

    }

    override fun getUserData() = datastore.data.map { prefs ->
        User(
            prefs[PreferencesKeys.USER_NAME] ?: "",
            prefs[PreferencesKeys.USER_AGE] ?: 0,
            Gender.valueOf(prefs[PreferencesKeys.USER_GENDER] ?: Gender.OTHER.name),
            prefs[PreferencesKeys.USER_HEIGHT] ?: 0.0,
            prefs[PreferencesKeys.USER_WEIGHT] ?: 0.0
        )
    }

    override suspend fun setStepGoal(stepGoal: Int) {
        datastore.edit { prefs ->
            prefs[PreferencesKeys.STEP_GOAL] = stepGoal
        }
    }

    override fun getStepGoal() = datastore.data.map { prefs ->
        prefs[PreferencesKeys.STEP_GOAL] ?: 0
    }

    override suspend fun setAchievements(list: List<Achievement>) {
        datastore.edit { prefs ->
            prefs[PreferencesKeys.ACHIEVEMENTS] = Gson().toJson(list)
        }
    }

    override fun getAchievements() = datastore.data.map { prefs ->
        val json = prefs[PreferencesKeys.ACHIEVEMENTS]
        if (json != null) {
            Gson().fromJson(json, Array<Achievement>::class.java).toList()
        } else {
            emptyList()
        }
    }
}

```

DailyStepData.kt

```
package com.app.stepup.model.room
```

```
import androidx.room.ColumnInfo
```

```
import androidx.room.Entity
```

```
import androidx.room.PrimaryKey
```

```
@Entity(tableName = "daily_steps")
```

```
data class DailyStepData(
```

```
    @PrimaryKey(autoGenerate = true) val id: Int = 0,
```

```
    @ColumnInfo(name = "date") val date: String,
```

```
    @ColumnInfo(name = "hour") val hour: Int,
```

```
    @ColumnInfo(name = "steps") val steps: Long
```

```
)
```

MonthlyStepData.kt

```
package com.app.stepup.model.room
```

```
import androidx.room.Entity
```

```
@Entity(tableName = "monthly_steps", primaryKeys = ["month", "day"])
```

```
data class MonthlyStepData(
```

```
    val month: String, //"YYYY-MM"
```

```
    val day: Int,
```

```
    val steps: Long
```

```
)
```

StepDao.kt

```
package com.app.stepup.model.room
```

```
import androidx.room.Dao
```

```
import androidx.room.Insert
```

```
import androidx.room.OnConflictStrategy
```

```
import androidx.room.Query
```

```
import kotlinx.coroutines.flow.Flow
```

```
@Dao
```

```
interface StepDao {
```

```
    @Query("SELECT * FROM daily_steps WHERE date = :date AND hour = :hour")
```

```
    suspend fun getStepDataByDateAndHour(date: String, hour: Int): DailyStepData?
```

```

@Query("SELECT * FROM daily_steps ORDER BY hour")
fun getFlowStepsForDay(): Flow<List<DailyStepData>>

@Query("SELECT * FROM daily_steps ORDER BY hour")
suspend fun getStepsForDay(): List<DailyStepData>

@Query("SELECT * FROM daily_steps WHERE date = :date ORDER BY hour")
suspend fun getStepsForDayByDate(date: String): List<DailyStepData>?

@Insert(onConflict = OnConflictStrategy.REPLACE)
suspend fun insertOrUpdateDailySteps(dailyStep: DailyStepData)

@Query("DELETE FROM daily_steps WHERE date = :date")
suspend fun clearStepsForDay(date: String)

@Query("SELECT * FROM monthly_steps WHERE month = :month")
suspend fun getStepsByMonth(month: String): List<MonthlyStepData>?

@Query("SELECT * FROM monthly_steps WHERE month = :month AND day = :day")
suspend fun getStepsByMonthAndDay(month: String, day: Int): MonthlyStepData?

@Query("SELECT * FROM monthly_steps ORDER BY day")
suspend fun getStepsForMonth(): List<MonthlyStepData>

@Insert(onConflict = OnConflictStrategy.REPLACE)
suspend fun insertOrUpdateMonthlySteps(monthlyStep: MonthlyStepData)

@Query("DELETE FROM monthly_steps WHERE month = :month")
suspend fun clearStepsForMonth(month: String)

@Query("SELECT * FROM yearly_steps WHERE month = :month AND year = :year")
suspend fun getStepsFromYearByMonth(year: String, month: Int): YearlyStepData?

@Query("SELECT * FROM yearly_steps WHERE year = :year ORDER BY month")
suspend fun getStepsForYear(year: String): List<YearlyStepData>

@Insert(onConflict = OnConflictStrategy.REPLACE)
suspend fun insertOrUpdateYearlySteps(yearlyStep: YearlyStepData)

@Query("SELECT steps FROM total_steps WHERE id = 1")

```

```

fun getTotalSteps(): Flow<Long>

@Query("UPDATE total_steps SET steps = steps + :newSteps WHERE id = 1")
suspend fun updateTotalSteps(newSteps: Long)

@Insert(onConflict = OnConflictStrategy.REPLACE)
suspend fun insertInitial(totalSteps: TotalStepData)
}

```

StepData.kt

```
package com.app.stepup.model.room
```

```

import androidx.room.ColumnInfo
import androidx.room.Entity
import androidx.room.PrimaryKey

```

```

@Entity(tableName = "steps")
data class StepData(
    @PrimaryKey(autoGenerate = true) val uid: Int = 0,
    @ColumnInfo(name = "steps") val steps: Long,
    @ColumnInfo(name = "created_at") val createdAt: String,
)

```

StepDatabase.kt

```
package com.app.stepup.model.room
```

```

import androidx.room.Database
import androidx.room.RoomDatabase
import androidx.room.migration.Migration
import androidx.sqlite.db.SupportSQLiteDatabase

```

```

@Database(
    entities = [
        StepData::class,
        TotalStepData::class,
        DailyStepData::class,
        MonthlyStepData::class,
        YearlyStepData::class
    ],
    version = 5
)

```

```

abstract class StepDatabase : RoomDatabase() {
    abstract fun stepDao(): StepDao
}

val MIGRATION_1_2 = object : Migration(1, 2) {
    override fun migrate(db: SupportSQLiteDatabase) {
        db.execSQL(
            """
            CREATE TABLE IF NOT EXISTS daily_steps (
                id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
                date TEXT NOT NULL,
                hour INTEGER NOT NULL,
                steps INTEGER NOT NULL
            )
            """
        )

        db.execSQL(
            """
            CREATE TABLE IF NOT EXISTS monthly_steps (
                month TEXT PRIMARY KEY NOT NULL,
                day INTEGER NOT NULL,
                steps INTEGER NOT NULL
            )
            """
        )

        db.execSQL(
            """
            CREATE TABLE IF NOT EXISTS yearly_steps (
                year TEXT PRIMARY KEY NOT NULL,
                month TEXT NOT NULL,
                steps INTEGER NOT NULL
            )
            """
        )

        db.execSQL(
            """
            CREATE TABLE IF NOT EXISTS total_steps (
                id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,

```

```

        steps INTEGER NOT NULL
    )
    ""
)
}
}

val MIGRATION_2_3 = object : Migration(2, 3) {
    override fun migrate(db: SupportSQLiteDatabase) {
        db.execSQL(
            ""
            CREATE TABLE IF NOT EXISTS `yearly_steps_new` (
                `year` TEXT PRIMARY KEY NOT NULL,
                `month` INTEGER NOT NULL,
                `steps` INTEGER NOT NULL
            )
            ""
        )

        db.execSQL(
            ""
            INSERT INTO `yearly_steps_new` (`year`, `month`, `steps`)
            SELECT `year`, CAST(`month` AS INTEGER), `steps` FROM `yearly_steps`
            ""
        )

        db.execSQL("DROP TABLE `yearly_steps`")

        db.execSQL("ALTER TABLE `yearly_steps_new` RENAME TO `yearly_steps`")
    }
}

val MIGRATION_3_4 = object : Migration(3, 4) {
    override fun migrate(db: SupportSQLiteDatabase) {
        db.execSQL(
            ""
            CREATE TABLE IF NOT EXISTS new_monthly_steps (
                month TEXT NOT NULL,
                day INTEGER NOT NULL,
                steps INTEGER NOT NULL,
                PRIMARY KEY(month, day)
            )
            ""
        )
    }
}

```

```

        )
        """
    )

    db.execSQL(
        """
        INSERT INTO new_monthly_steps (month, day, steps)
        SELECT month, day, steps FROM monthly_steps
        """
    )

    db.execSQL("DROP TABLE IF EXISTS monthly_steps")

    db.execSQL("ALTER TABLE new_monthly_steps RENAME TO monthly_steps")
}

}

val MIGRATION_4_5 = object : Migration(4, 5) {
    override fun migrate(db: SupportSQLiteDatabase) {
        db.execSQL(
            """
            CREATE TABLE IF NOT EXISTS yearly_steps_new (
                year TEXT NOT NULL,
                month INTEGER NOT NULL,
                steps INTEGER NOT NULL,
                PRIMARY KEY(year, month)
            )
            """
        )

        db.execSQL(
            """
            INSERT INTO yearly_steps_new (year, month, steps)
            SELECT year, month, steps FROM yearly_steps
            """
        )

        db.execSQL("DROP TABLE yearly_steps")

        db.execSQL("ALTER TABLE yearly_steps_new RENAME TO yearly_steps")
    }
}

```

```
}
```

TotalStepData.kt

```
package com.app.stepup.model.room
```

```
import androidx.room.ColumnInfo
```

```
import androidx.room.Entity
```

```
import androidx.room.PrimaryKey
```

```
@Entity(tableName = "total_steps")
```

```
data class TotalStepData(
```

```
    @PrimaryKey(autoGenerate = true) val id: Int = 1,
```

```
    @ColumnInfo(name = "steps") val steps: Long
```

```
)
```

YearlyStepData.kt

```
package com.app.stepup.model.room
```

```
import androidx.room.Entity
```

```
@Entity(tableName = "yearly_steps", primaryKeys = ["year", "month"])
```

```
data class YearlyStepData(
```

```
    val year: String, // "YYYY"
```

```
    val month: Int,
```

```
    val steps: Long
```

```
)
```

StepRepository.kt

```
package com.app.stepup.model
```

```
import android.content.Context
```

```
import com.app.stepup.R
```

```
import com.app.stepup.model.room.DailyStepData
```

```
import com.app.stepup.model.room.MonthlyStepData
```

```
import com.app.stepup.model.room.StepDao
```

```
import com.app.stepup.model.room.StepData
```

```
import com.app.stepup.model.room.TotalStepData
```

```
import com.app.stepup.model.room.YearlyStepData
```

```
import com.app.stepup.utils.DateParser
```

```
import dagger.hilt.android.qualifiers.ApplicationContext
```

```
import kotlinx.coroutines.flow.Flow
```

```

import kotlinx.coroutines.flow.firstOrNull
import kotlinx.coroutines.flow.map
import java.time.LocalDate
import java.time.LocalDateTime
import javax.inject.Inject

class StepRepository @Inject constructor(
    @ApplicationContext private val context: Context,
    private val stepDao: StepDao,
    private val dateParser: DateParser
) {
    suspend fun insert(steps: StepData) {
        addToTotalDb(steps)
        processDailySteps(steps)
    }

    fun getTotalSteps() = stepDao.getTotalSteps()

    fun getFlowDailySteps(): Flow<List<Long>> {
        val steps = stepDao.getFlowStepsForDay()
        return steps.map { hourlySteps ->
            val stepsList = MutableList(24) { 0L }
            hourlySteps.forEach { stepData ->
                if (stepData.hour in 0..23) {
                    stepsList[stepData.hour] = stepData.steps
                }
            }
            stepsList
        }
    }

    suspend fun getDailySteps(): Long {
        return stepDao.getStepsForDay().sumOf { it.steps }
    }

    suspend fun getMonthlySteps() = stepDao.getStepsForMonth()

    suspend fun getYearlySteps(year: String) = stepDao.getStepsForYear(year)

    suspend fun insertTotalDb() {
        val stepsTotal = stepDao.getTotalSteps().firstOrNull()
    }

```

```

        if (stepsTotal == null)
            stepDao.insertInitial(TotalStepData(steps = 0L))
    }

    private suspend fun addToTotalDb(steps: StepData) {
        stepDao.updateTotalSteps(steps.steps)
    }

    private suspend fun processDailySteps(steps: StepData) {
        val dateInDb = stepDao.getFlowStepsForDay().firstOrNull()?.firstOrNull()?.date
        val currentDate = LocalDateTime.now().toLocalDate().toString()
        updateDailySteps(steps, currentDate)
        if (!dateInDb.isNullOrEmpty() && dateInDb != currentDate)
            clearDailySteps(dateInDb)
    }

    private suspend fun updateDailySteps(steps: StepData, currentDate: String) {
        val currentHour = LocalDateTime.now().hour
        val existingHourData = stepDao.getStepDataByDateAndHour(currentDate, currentHour)
        stepDao.insertOrUpdateDailySteps(
            existingHourData?.copy(
                steps = existingHourData.steps + steps.steps
            ) ?: DailyStepData(date = currentDate, hour = currentHour, steps =
steps.steps)
        )
    }

    private suspend fun clearDailySteps(previousDate: String) {
        val dailySteps = stepDao.getStepsForDayByDate(previousDate)
        val totalStepsForDay = dailySteps?.sumOf { it.steps } ?: 0
        processMonthlySteps(totalStepsForDay, previousDate)
        stepDao.clearStepsForDay(previousDate)
    }

    private suspend fun processMonthlySteps(totalStepsForDay: Long, previousDate: String)
{
        val monthInDb = stepDao.getStepsForMonth().firstOrNull()?.month
        val currentMonth = LocalDate.now().toString().substring(0, 7) // "YYYY-MM"
        updateMonthlySteps(totalStepsForDay, previousDate)
        if (!monthInDb.isNullOrEmpty() && monthInDb != currentMonth) {
            clearMonthlySteps(monthInDb)
        }
    }

```

```

    }
}

private suspend fun updateMonthlySteps(totalStepsForDay: Long, previousDate: String) {
    val previousMonth = previousDate.substring(0, 7) // "YYYY-MM"
    val day = dateParser.getDayFromDate(previousDate)
    val existingMonthData = stepDao.getStepsByMonthAndDay(previousMonth, day)

    stepDao.insertOrUpdateMonthlySteps(
        existingMonthData?.copy(
            steps = existingMonthData.steps + totalStepsForDay
        ) ?: MonthlyStepData(
            previousMonth, day, totalStepsForDay
        )
    )
}

private suspend fun clearMonthlySteps(previousData: String) {
    val monthlySteps = stepDao.getStepsByMonth(previousData)
    val totalStepsForMonth = monthlySteps?.sumOf { it.steps } ?: 0
    updateYearlySteps(totalStepsForMonth, previousData)
    stepDao.clearStepsForMonth(previousData)
}

private suspend fun updateYearlySteps(totalStepsForMonth: Long, previousData: String)
{
    val monthAndYear = dateParser.getMonthAndYearFromDate(previousData)
    val existingYearlyStepData =
        stepDao.getStepsFromYearByMonth(monthAndYear.second, monthAndYear.first)
    stepDao.insertOrUpdateYearlySteps(
        existingYearlyStepData?.copy(
            steps = existingYearlyStepData.steps + totalStepsForMonth
        ) ?: YearlyStepData(
            monthAndYear.second, monthAndYear.first, totalStepsForMonth
        )
    )
}

fun getStepsByWeek(
    dailySteps: Long,
    weeklySteps: List<MonthlyStepData>

```

```

): Map<String, Double> {
    val stepsByDay = weeklySteps.associate { stepData ->

        LocalDate.parse("${stepData.month}-${"%02d".format(stepData.day)}").dayOfWeek.value to
        stepData.steps.toDouble()

    }

    val daysOfWeek = context.resources.getStringArray(R.array.days_of_week).toList()

    return daysOfWeek.associateWith { day ->
        val dayIndex = daysOfWeek.indexOf(day) + 1
        if (dayIndex == LocalDate.now().dayOfWeek.value) {
            dailySteps.toDouble()
        } else {
            stepsByDay[dayIndex] ?: 0.0
        }
    }
}

fun getStepsByMonth(
    monthlySteps: List<MonthlyStepData>,
    currentDaySteps: Long
): Map<Int, Double> {
    val stepsByDay = monthlySteps.associate { it.day to it.steps.toDouble() }

    val daysInMonth = LocalDate.now().lengthOfMonth()

    return (1..daysInMonth).associateWith { day ->
        if (day == LocalDate.now().dayOfMonth) {
            currentDaySteps.toDouble()
        } else {
            stepsByDay[day] ?: 0.0
        }
    }
}

fun getStepsByYear(
    yearlySteps: List<YearlyStepData>,
    currentMonthSteps: Long
): Map<String, Double> {
    val stepsByMonth = yearlySteps.associate { it.month to it.steps.toDouble() }

```

```

        val monthsOfYear =
context.resources.getStringArray(R.array.months_of_year).toList()

        return monthsOfYear.associateWith { month ->
            val monthIndex = monthsOfYear.indexOf(month) + 1
            if (monthIndex == LocalDate.now().monthValue) {
                currentMonthSteps.toDouble()
            } else {
                stepsByMonth[monthIndex] ?: 0.0
            }
        }
    }
}

```

ScreenRoutes.kt

```
package com.app.stepup.navigation
```

```

import com.app.stepup.constants.Constants.ACHIEVEMENT
import com.app.stepup.constants.Constants.ACHIEVEMENTS
import com.app.stepup.constants.Constants.ANALYTICS
import com.app.stepup.constants.Constants.HOME
import com.app.stepup.constants.Constants.MAIN
import com.app.stepup.constants.Constants.PROFILE
import com.app.stepup.constants.Constants.REGISTRATION

```

```

sealed class ScreenRoutes(val route: String) {
    data object MainScreen : ScreenRoutes(MAIN)
    data object RegistrationScreen : ScreenRoutes(REGISTRATION)
    data object HomeScreen : ScreenRoutes(HOME)
    data object ProfileScreen : ScreenRoutes(PROFILE)
    data object AnalyticsScreen : ScreenRoutes(ANALYTICS)
    data object AchievementsScreen : ScreenRoutes(ACHIEVEMENTS)
    data object AchievementScreen : ScreenRoutes(ACHIEVEMENT)
}

```

StepUpNavigation.kt

```
package com.app.stepup.navigation
```

```

import androidx.compose.foundation.layout.PaddingValues
import androidx.compose.runtime.Composable

```

```

import androidx.navigation.NavType
import androidx.navigation.compose.NavHost
import androidx.navigation.compose.composable
import androidx.navigation.compose.rememberNavController
import androidx.navigation.navArgument
import com.app.stepup.constants.Constants.KEY_ACHIEVEMENT
import com.app.stepup.model.data.Achievement
import com.app.stepup.ui.screens.AchievementScreen
import com.app.stepup.ui.screens.AchievementsScreen
import com.app.stepup.ui.screens.AnalyticsScreen
import com.app.stepup.ui.screens.HomeScreen
import com.app.stepup.ui.screens.MainScreen
import com.app.stepup.ui.screens.ProfileScreen
import com.app.stepup.ui.screens.RegistrationScreen
import com.google.gson.Gson

@Composable
fun StepUpNavigation(paddingValues: PaddingValues) {
    val navController = rememberNavController()
    NavHost(navController = navController, startDestination =
ScreenRoutes.MainScreen.route) {
        composable(ScreenRoutes.MainScreen.route) {
            MainScreen(navController)
        }
        composable(ScreenRoutes.RegistrationScreen.route) {
            RegistrationScreen(paddingValues, navController)
        }
        composable(ScreenRoutes.HomeScreen.route) {
            HomeScreen(paddingValues, navController)
        }
        composable(ScreenRoutes.AnalyticsScreen.route) {
            AnalyticsScreen(paddingValues)
        }
        composable(ScreenRoutes.ProfileScreen.route) {
            ProfileScreen(paddingValues, navController)
        }
        composable(ScreenRoutes.AchievementsScreen.route) {
            AchievementsScreen(paddingValues, navController)
        }
        composable(
            route = buildString {

```

```

        append(ScreenRoutes.AchievementScreen.route)
        append("/{ $KEY_ACHIEVEMENT}")
    },
    arguments = listOf(navArgument(KEY_ACHIEVEMENT) { type = NavType.StringType })
) {
    val json = it.arguments?.getString(KEY_ACHIEVEMENT)
    AchievementScreen(
        paddingValues,
        if (json != null) {
            Gson().fromJson(json, Achievement::class.java)
        } else {
            null
        }
    )
}
}
}
}

```

StepsService.kt

```
package com.app.stepup.services
```

```

import android.app.Notification
import android.app.NotificationChannel
import android.app.NotificationManager
import android.app.Service
import android.content.Context
import android.content.Intent
import android.hardware.Sensor
import android.hardware.SensorEvent
import android.hardware.SensorEventListener
import android.hardware.SensorManager
import android.os.IBinder
import androidx.core.app.NotificationCompat
import com.app.stepup.R
import com.app.stepup.constants.Constants.STEPS_COUNTER_CHANNEL_ID
import com.app.stepup.constants.Constants.STEPS_COUNTER_CHANNEL_NAME
import com.app.stepup.constants.Constants.STEPS_COUNTER_NOTIFICATION_ID
import com.app.stepup.model.StepRepository
import com.app.stepup.model.room.StepData
import dagger.hilt.android.AndroidEntryPoint
import kotlinx.coroutines.CoroutineScope

```

```

import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.Job
import kotlinx.coroutines.launch
import java.time.Instant
import javax.inject.Inject

@AndroidEntryPoint
class StepsService : Service(), SensorEventListener {
    @Inject
    lateinit var stepRepository: StepRepository

    private lateinit var sensorManager: SensorManager

    private val serviceJob = Job()
    private val coroutineScope = CoroutineScope(serviceJob + Dispatchers.IO)

    private var stepSensor: Sensor? = null

    override fun onCreate() {
        super.onCreate()
        startForegroundService()
        initialSensor()
    }

    override fun onStartCommand(intent: Intent?, flags: Int, startId: Int): Int {
        return START_STICKY
    }

    override fun onSensorChanged(event: SensorEvent?) {
        if (event?.values != null) {
            coroutineScope.launch(Dispatchers.IO) {
                stepRepository.insert(
                    StepData(
                        steps = event.values[0].toLong(),
                        createdAt = Instant.now().toString()
                    )
                )
            }
        }
    }
}

```

```

override fun onAccuracyChanged(sensor: Sensor?, accuracy: Int) {}

override fun onBind(intent: Intent?): IBinder? = null

override fun onDestroy() {
    super.onDestroy()
    sensorManager.unregisterListener(this)
    serviceJob.cancel()
}

private fun initialSensor() {
    sensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
    stepSensor = sensorManager.getDefaultSensor(Sensor.TYPE_STEP_DETECTOR)
    stepSensor?.let {
        sensorManager.registerListener(this, it, SensorManager.SENSOR_DELAY_NORMAL)
    }
}

private fun startForegroundService() {
    val notificationManager = getSystemService(
        Context.NOTIFICATION_SERVICE
    ) as NotificationManager
    val channel = NotificationChannel(
        STEPS_COUNTER_CHANNEL_ID,
        STEPS_COUNTER_CHANNEL_NAME,
        NotificationManager.IMPORTANCE_LOW
    )
    notificationManager.createNotificationChannel(channel)

    val notification: Notification = NotificationCompat.Builder(this,
STEPS_COUNTER_CHANNEL_ID)
        .setContentTitle(getString(R.string.app_name))
        .setContentText(getString(R.string.service_count_steps))
        .setSmallIcon(R.mipmap.ic_launcher)
        .build()

    startForeground(STEPS_COUNTER_NOTIFICATION_ID, notification)
}
}

```

```

AchievementView.kt
package com.app.stepup.ui.components

import androidx.compose.foundation.Image
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.padding
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.res.stringResource
import androidx.compose.ui.text.TextStyle
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.text.style.TextAlign
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import com.app.stepup.model.data.Achievement

@Composable
fun AchievementView(achieve: Achievement) {
    Column(
        Modifier
            .fillMaxSize()
            .padding(16.dp),
        horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.Center
    ) {
        Image(
            painter = painterResource(id = achieve.iconRes),
            contentDescription = achieve.iconRes.toString(),
            modifier = Modifier
                .fillMaxSize(0.7f)
        )
        Text(
            text = stringResource(id = achieve.name.getNameRes()),
            style = TextStyle(
                fontSize = 15.sp,

```

```

        color = Color.White,
        fontWeight = FontWeight.Bold,
        textAlign = TextAlign.Center
    )
)
}
}

```

AnalyticsColumnChart.kt

```
package com.app.stepup.ui.components
```

```

import androidx.compose.foundation.background
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.fillMaxHeight
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.saveable.rememberSaveable
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.graphics.Color.Companion.White
import androidx.compose.ui.graphics.SolidColor
import androidx.compose.ui.res.stringResource
import androidx.compose.ui.text.TextStyle
import androidx.compose.ui.text.font.FontWeight.Companion.Bold
import androidx.compose.ui.text.font.FontWeight.Companion.SemiBold
import androidx.compose.ui.text.style.TextAlign
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import com.app.stepup.R
import com.app.stepup.ui.screens.checkIfNeedToMakeShortly
import com.app.stepup.ui.theme.TransparentWhite
import ir.ehsannarmani.compose_charts.ColumnChart
import ir.ehsannarmani.compose_charts.models.BarProperties
import ir.ehsannarmani.compose_charts.models.Bars

```

```

import ir.ehsannarmani.compose_charts.models.DividerProperties
import ir.ehsannarmani.compose_charts.models.GridProperties
import ir.ehsannarmani.compose_charts.models.HorizontalIndicatorProperties
import ir.ehsannarmani.compose_charts.models.LabelHelperProperties
import ir.ehsannarmani.compose_charts.models.LabelProperties
import ir.ehsannarmani.compose_charts.models.LineProperties
import ir.ehsannarmani.compose_charts.models.PopupProperties

@Composable
fun AnalyticsColumnChart(
    title: Int,
    data: Map<String, Double>,
    modifier: Modifier,
    barWidth: Int,
    barColor: Color,
    rotation: Float = 0f,
    isMakeShortly: Boolean = false
) {
    Column(
        modifier
            .fillMaxWidth()
            .padding(horizontal = 16.dp)
            .background(TransparentWhite, shape = RoundedCornerShape(8.dp))
    ) {
        if (data.values.isEmpty() or data.values.all { it == 0.0 }) {
            NoData()
        } else {
            Title(string = title)
            ColumnChart(
                modifier = Modifier
                    .fillMaxHeight()
                    .padding(16.dp),
                data = rememberSaveable {
                    data.toList().mapIndexed { index, pair ->
                        Bars(
                            label = checkIfNeedToMakeShortly(
                                isMakeShortly,
                                index,
                                pair.first,
                                data.size
                            ),

```

```

        values = listOf(
            Bars.Data(
                value = pair.second,
                color = SolidColor(barColor)
            )
        ),
    )
}

},
barProperties = BarProperties(thickness = barWidth.dp, spacing = 1.dp),
gridProperties = GridProperties(enabled = false),
labelHelperProperties = LabelHelperProperties(enabled = false),
dividerProperties = DividerProperties(
    xAxisProperties = LineProperties(color = SolidColor(White)),
    yAxisProperties = LineProperties(color = SolidColor(White))
),
popupProperties = PopupProperties(
    textStyle = TextStyle(color = White),
    containerColor = MaterialTheme.colorScheme.primary.copy(0.5f)
),
labelProperties = LabelProperties(
    labels = data.keys.toList(),
    enabled = true,
    textStyle = TextStyle(
        color = White,
        fontSize = 12.sp
    ),
    rotationDegreeOnSizeConflict = rotation
),
indicatorProperties = HorizontalIndicatorProperties(enabled = false)
)
}
}

@Composable
fun NoData() {
    Column(
        Modifier
            .fillMaxSize()
            .padding(16.dp),

```

```

        horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.Center
    ) {
        Text(
            text = stringResource(id = R.string.no_data),
            fontSize = 18.sp,
            color = White,
            fontWeight = SemiBold
        )
        Text(
            text = stringResource(id = R.string.no_data_description),
            color = White,
            fontSize = 14.sp,
            textAlign = TextAlign.Center
        )
    }
}

```

```

@Composable
fun Title(string: Int) {
    Text(
        text = stringResource(id = string),
        color = White,
        fontSize = 20.sp,
        fontWeight = Bold,
        modifier = Modifier
            .padding(horizontal = 16.dp)
            .padding(top = 16.dp)
    )
}

```

Background.kt

```
package com.app.stepup.ui.components
```

```

import androidx.compose.foundation.background
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Brush
import androidx.compose.ui.graphics.TileMode

```

```
import androidx.compose.ui.tooling.preview.Preview
import com.app.stepup.ui.theme.Lavender
import com.app.stepup.ui.theme.LavenderBlue
import com.app.stepup.ui.theme.PastelBlue
import com.app.stepup.ui.theme.PastelPink
import com.app.stepup.ui.theme.PastelPurple
import com.app.stepup.ui.theme.SoftLilac

val pastelGradientColors = listOf(
    Lavender, SoftLilac, PastelPurple, LavenderBlue, PastelBlue, PastelPink
)
```

```
@Composable
fun Background() {
    Box(
        modifier = Modifier
            .background(
                brush = Brush.linearGradient(
                    colors = pastelGradientColors,
                    tileMode = TileMode.Repeated
                )
            )
        .fillMaxSize()
    )
}
```

```
@Preview
@Composable
fun PreviewBackground() {
    Background()
}
```

DefaultButton.kt

```
package com.app.stepup.ui.components
```

```
import androidx.compose.foundation.BorderStroke
import androidx.compose.foundation.layout.PaddingValues
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.material3.Button
```

```

import androidx.compose.material3.ButtonDefaults
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.res.stringResource
import androidx.compose.ui.text.TextStyle
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp

@Composable
fun DefaultButton(text: Int, onClick: () -> Unit) {
    Button(
        onClick = { onClick() },
        colors = ButtonDefaults.buttonColors(
            containerColor = Color.Transparent
        ),
        modifier = Modifier
            .fillMaxWidth()
            .padding(horizontal = 16.dp),
        shape = RoundedCornerShape(8.dp),
        border = BorderStroke(2.dp, MaterialTheme.colorScheme.primary),
        contentPadding = PaddingValues(16.dp)
    ) {
        Text(
            text = stringResource(id = text),
            style = TextStyle(
                color = MaterialTheme.colorScheme.primary,
                fontSize = 20.sp,
                fontWeight = FontWeight.Bold
            )
        )
    }
}

```

DropDownMenu.kt

```
package com.app.stepup.ui.components
```

```
import androidx.compose.foundation.Image
```

```

import androidx.compose.foundation.border
import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.heightIn
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.width
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.material3.DropdownMenu
import androidx.compose.material3.DropdownMenuItem
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.LaunchedEffect
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableIntStateOf
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.saveable.rememberSaveable
import androidx.compose.runtime.setValue
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.platform.LocalConfiguration
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.text.TextStyle
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import com.app.stepup.R

```

```
@Composable
```

```

fun <T> DropDownMenu(items: List<T>, initialSelectedIndex: Int = 0, onClick: (item: T) ->
Unit) {
    var isDropDownGenderExpanded by rememberSaveable { mutableStateOf(false) }
    var itemPosition by rememberSaveable { mutableIntStateOf(initialSelectedIndex) }
    LaunchedEffect(initialSelectedIndex) {
        itemPosition = initialSelectedIndex
    }
    Box(

```

```

        modifier = Modifier
            .border(1.dp, Color.White, RoundedCornerShape(8.dp))
            .padding(6.dp)
            .width(100.dp)
    ) {
        Row(
            modifier = Modifier
                .clickable {
                    isDropDownGenderExpanded = true
                }
                .fillMaxWidth(),
            verticalAlignment = Alignment.CenterVertically,
            horizontalArrangement = Arrangement.SpaceBetween
        ) {
            Text(
                text = items[itemPosition].toString(),
                style = TextStyle(
                    color = Color.White,
                    fontWeight = FontWeight.Bold,
                    fontSize = 18.sp
                )
            )
            Image(
                painter = painterResource(id = R.drawable.ic_drop_down),
                contentDescription = R.drawable.ic_drop_down.toString()
            )
        }
        DropdownMenu(
            modifier = Modifier.heightIn(max =
LocalConfiguration.current.screenHeightDp.dp / 2),
            expanded = isDropDownGenderExpanded,
            onDismissRequest = {
                isDropDownGenderExpanded = false
            }) {
            items.forEachIndexed { index, value ->
                DropdownMenuItem(
                    text = {
                        Text(
                            text = value.toString(),
                            style = TextStyle(
                                color = MaterialTheme.colorScheme.primary,

```



```

import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.res.stringResource
import androidx.compose.ui.text.TextStyle
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp

@Composable
fun PairTitle(firstTitle: Int, secondTitle: Int) {
    Text(
        text = stringResource(id = firstTitle),
        modifier = Modifier.width(100.dp),
        style = TextStyle(
            color = Color.White,
            fontSize = 16.sp,
            fontWeight = FontWeight.Bold
        )
    )
    Text(
        text = stringResource(id = secondTitle),
        modifier = Modifier.width(100.dp),
        style = TextStyle(
            color = Color.White,
            fontSize = 16.sp,
            fontWeight = FontWeight.Bold
        )
    )
}

showToast.kt
package com.app.stepup.ui.components

import android.content.Context
import android.widget.Toast

fun showToast(context: Context, string: Int) {
    Toast.makeText(context, context.getString(string), Toast.LENGTH_SHORT).show()
}

```

```

AchievementScreen.kt
package com.app.stepup.ui.screens

import androidx.compose.foundation.Image
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.PaddingValues
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.aspectRatio
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.padding
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color.Companion.Gray
import androidx.compose.ui.graphics.Color.Companion.White
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.res.stringResource
import androidx.compose.ui.text.TextStyle
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.text.font.FontWeight.Companion.SemiBold
import androidx.compose.ui.text.style.TextAlign
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import com.app.stepup.R
import com.app.stepup.model.data.Achievement
import com.app.stepup.model.data.Status
import com.app.stepup.ui.components.Background
import com.app.stepup.ui.theme.SoftGreen

@Composable
fun AchievementScreen(paddingValues: PaddingValues, achievement: Achievement?) {
    Box(
        modifier = Modifier
            .fillMaxSize()
            .padding(paddingValues)
    ) {

```

```

Background()
if (achievement != null) {
    Column(
        Modifier
            .fillMaxSize()
            .padding(horizontal = 16.dp),
        horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.Center
    ) {
        Text(
            text = stringResource(id = achievement.name.getNameRes()),
            style = TextStyle(
                fontSize = 32.sp,
                color = White,
                fontWeight = FontWeight.Bold,
                textAlign = TextAlign.Center
            )
        )
        Spacer(modifier = Modifier.height(16.dp))
        Image(
            painter = painterResource(id = achievement.iconRes),
            contentDescription = achievement.iconRes.toString(),
            modifier = Modifier
                .fillMaxWidth(0.9f)
                .aspectRatio(1f)
        )
        Spacer(modifier = Modifier.height(16.dp))
        Text(
            text = stringResource(id = achievement.descriptionRes),
            style = TextStyle(
                fontSize = 24.sp,
                color = White,
                fontWeight = FontWeight.Bold,
                textAlign = TextAlign.Center
            )
        )
        Spacer(modifier = Modifier.height(16.dp))
        Text(
            text = achievement.status.name,
            style = TextStyle(
                fontSize = 24.sp,

```

```

        color = when (achievement.status) {
            Status.DONE -> SoftGreen
            Status.UNDONE -> Gray
        },
        fontWeight = FontWeight.Bold,
        textAlign = TextAlign.Center
    )
)
}
} else {
    Column(
        Modifier.fillMaxSize(),
        horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.Center
    ) {
        Text(
            text = stringResource(id = R.string.something_went_wrong),
            fontSize = 24.sp,
            color = White,
            fontWeight = SemiBold
        )
    }
}
}
}

```

AchievementsScreen.kt

```
package com.app.stepup.ui.screens
```

```

import androidx.compose.foundation.background
import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.PaddingValues
import androidx.compose.foundation.layout.aspectRatio
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.lazy.grid.GridCells
import androidx.compose.foundation.lazy.grid.LazyVerticalGrid
import androidx.compose.foundation.lazy.grid.items
import androidx.compose.foundation.shape.CircleShape
import androidx.compose.runtime.Composable

```

```

import androidx.compose.runtime.LaunchedEffect
import androidx.compose.runtime.collectAsState
import androidx.compose.runtime.getValue
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.unit.dp
import androidx.hilt.navigation.compose.hiltViewModel
import androidx.navigation.NavController
import com.app.stepup.model.data.Status
import com.app.stepup.navigation.ScreenRoutes
import com.app.stepup.ui.components.AchievementView
import com.app.stepup.ui.components.Background
import com.app.stepup.ui.components.LoadingIndicator
import com.app.stepup.ui.theme.SoftGreen
import com.app.stepup.view_models.AchievementsViewModel
import com.google.gson.Gson

@Composable
fun AchievementsScreen(
    paddingValues: PaddingValues,
    navController: NavController,
    viewModel: AchievementsViewModel = hiltViewModel()
) {
    val achievements by viewModel.achievements.collectAsState(initial = emptyList())

    LaunchedEffect(achievements) {
        if (achievements.isNotEmpty())
            viewModel.checkAchievements(achievements)
    }

    if (achievements.isEmpty()) {
        LoadingIndicator()
    } else {
        Box(
            modifier = Modifier
                .fillMaxSize()
                .padding(paddingValues)
        ) {
            Background()
            LazyVerticalGrid(

```

```

        columns = GridCells.Fixed(2),
        modifier = Modifier.fillMaxSize()
    ) {
        items(achievements) { achieve ->
            Box(
                modifier = Modifier
                    .fillMaxSize()
                    .aspectRatio(1f)
                    .padding(8.dp)
                    .background(
                        when (achieve.status) {
                            Status.DONE -> SoftGreen
                            Status.UNDONE -> Color.Gray
                        },
                        CircleShape
                    )
                .clickable {
                    navController.navigate(
                        buildString {
                            append(ScreenRoutes.AchievementScreen.route)
                            append("/${Gson().toJson(achieve)}")
                        }
                    )
                },
                contentAlignment = Alignment.Center
            ) {
                AchievementView(achieve)
            }
        }
    }
}
}
}
}

```

AnalyticsScreen.kt

```
package com.app.stepup.ui.screens
```

```

import androidx.compose.foundation.background
import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Box

```

```

import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.PaddingValues
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.collectAsState
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.saveable.rememberSaveable
import androidx.compose.runtime.setValue
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.graphics.Color.Companion.Transparent
import androidx.compose.ui.graphics.Color.Companion.White
import androidx.compose.ui.res.stringResource
import androidx.compose.ui.text.TextStyle
import androidx.compose.ui.text.font.FontWeight.Companion.Bold
import androidx.compose.ui.text.style.TextAlign
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import androidx.hilt.navigation.compose.hiltViewModel
import com.app.stepup.R
import com.app.stepup.model.data.AnalyticsType
import com.app.stepup.ui.components.AnalyticsColumnChart
import com.app.stepup.ui.components.Background
import com.app.stepup.ui.components.LoadingIndicator
import com.app.stepup.ui.theme.SoftGreen
import com.app.stepup.ui.theme.SoftTeal
import com.app.stepup.ui.theme.WarmCoral
import com.app.stepup.view_models.AnalyticsViewModel

@Composable
fun AnalyticsScreen(

```

```

paddingValues: PaddingValues,
viewModel: AnalyticsViewModel = hiltViewModel()
) {
    var showInfo by rememberSaveable { mutableStateOf(AnalyticsType.STEPS) }

    Box(
        modifier = Modifier
            .fillMaxSize()
            .padding(paddingValues)
    ) {
        Background()
        Column(
            Modifier.fillMaxSize(),
            horizontalAlignment = Alignment.CenterHorizontally,
            verticalArrangement = Arrangement.SpaceAround
        ) {
            Spacer(modifier = Modifier.height(16.dp))
            Row(
                Modifier.fillMaxWidth(),
                verticalAlignment = Alignment.CenterVertically,
                horizontalArrangement = Arrangement.SpaceEvenly
            ) {
                Text(
                    text = stringResource(id = R.string.steps),
                    style = TextStyle(
                        color = White,
                        fontSize = 16.sp,
                        fontWeight = Bold,
                        textAlign = TextAlign.Center
                    ),
                    modifier = Modifier
                        .background(
                            if (showInfo == AnalyticsType.STEPS)
MaterialTheme.colorScheme.primary else Transparent,
                            shape = RoundedCornerShape(8.dp)
                        )
                        .padding(8.dp)
                        .weight(1f)
                        .clickable {
                            showInfo = AnalyticsType.STEPS
                        }
                )
            }
        }
    }
}

```

```

    )
    Text(
        text = stringResource(id = R.string.calories),
        style = TextStyle(
            color = White,
            fontSize = 16.sp,
            fontWeight = Bold,
            textAlign = TextAlign.Center
        ),
        modifier = Modifier
            .background(
                if (showInfo == AnalyticsType.CALORIES)
MaterialTheme.colorScheme.primary else Transparent,
                shape = RoundedCornerShape(8.dp)
            )
            .padding(8.dp)
            .weight(1f)
            .clickable {
                showInfo = AnalyticsType.CALORIES
            }
    )
    Text(
        text = stringResource(id = R.string.distance),
        style = TextStyle(
            color = White,
            fontSize = 16.sp,
            fontWeight = Bold,
            textAlign = TextAlign.Center
        ),
        modifier = Modifier
            .background(
                if (showInfo == AnalyticsType.DISTANCE)
MaterialTheme.colorScheme.primary else Transparent,
                shape = RoundedCornerShape(8.dp)
            )
            .padding(8.dp)
            .weight(1f)
            .clickable {
                showInfo = AnalyticsType.DISTANCE
            }
    )

```

```

    }
    when (showInfo) {
        AnalyticsType.STEPS -> {
            val byWeek by viewModel.stepsByWeek.collectAsState(initial =
emptyMap())
            val byMonth by viewModel.stepsByMonth.collectAsState(initial =
emptyMap())
            val byYear by viewModel.stepsByYear.collectAsState(initial =
emptyMap())
            if (byWeek.isEmpty() && byMonth.isEmpty() && byYear.isEmpty())
                LoadingIndicator()
            else
                AnalyticsCharts(
                    byWeek,
                    byMonth,
                    byYear,
                    SoftGreen
                )
        }
    }

    AnalyticsType.CALORIES -> {
        viewModel.calculateCalories()
        val byWeek by viewModel.caloriesByWeek.collectAsState()
        val byMonth by viewModel.caloriesByMonth.collectAsState()
        val byYear by viewModel.caloriesByYear.collectAsState()
        if (byWeek.isEmpty() && byMonth.isEmpty() && byYear.isEmpty())
            LoadingIndicator()
        else
            AnalyticsCharts(
                byWeek,
                byMonth,
                byYear,
                WarmCoral
            )
    }

    AnalyticsType.DISTANCE -> {
        viewModel.calculateDistance()
        val byWeek by viewModel.distanceByWeek.collectAsState()
        val byMonth by viewModel.distanceByMonth.collectAsState()
        val byYear by viewModel.distanceByYear.collectAsState()
    }

```

```

        if (byWeek.isEmpty() && byMonth.isEmpty() && byYear.isEmpty())
            LoadingIndicator()
        else
            AnalyticsCharts(
                byWeek,
                byMonth,
                byYear,
                SoftTeal
            )
    }
}
}
}
}
}

```

@Composable

```

fun AnalyticsCharts(
    dataFromWeek: Map<String, Double>,
    dataFromMonth: Map<Int, Double>,
    dataFromYear: Map<String, Double>,
    barColor: Color
) {
    Column(
        Modifier.fillMaxSize(),
        horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.SpaceAround
    ) {
        Spacer(modifier = Modifier.height(16.dp))
        AnalyticsColumnChart(
            R.string.analytics_week,
            dataFromWeek,
            Modifier.weight(1f),
            barWidth = 18,
            barColor = barColor,
        )
        Spacer(modifier = Modifier.height(16.dp))
        AnalyticsColumnChart(
            R.string.analytics_month,
            dataFromMonth.mapKeys { it.key.toString() },
            Modifier.weight(1f),
            8,

```

```

        isMakeShortly = true,
        barColor = barColor
    )
    Spacer(modifier = Modifier.height(16.dp))
    AnalyticsColumnChart(
        R.string.analytics_year,
        dataFromYear,
        Modifier.weight(1f),
        barColor = barColor,
        barWidth = 12,
        rotation = -45f
    )
    Spacer(modifier = Modifier.height(16.dp))
}
}

```

```

fun checkIfNeedToMakeShortly(isMakeShortly: Boolean, index: Int, name: String, size: Int):
String {
    return if (isMakeShortly) when {
        index == 0 || index == size - 1 || index % 5 == 0 -> name
        else -> " "
    } else name
}

```

HomeScreen.kt

```
package com.app.stepup.ui.screens
```

```

import androidx.compose.foundation.Image
import androidx.compose.foundation.background
import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.PaddingValues
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.aspectRatio
import androidx.compose.foundation.layout.fillMaxHeight
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.lazy.LazyRow

```

```
import androidx.compose.foundation.lazy.items
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.material3.CircularProgressIndicator
import androidx.compose.material3.Icon
import androidx.compose.material3.IconButton
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.LaunchedEffect
import androidx.compose.runtime.collectAsState
import androidx.compose.runtime.getValue
import androidx.compose.runtime.saveable.rememberSaveable
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color.Companion.White
import androidx.compose.ui.graphics.Shadow
import androidx.compose.ui.graphics.SolidColor
import androidx.compose.ui.graphics.StrokeCap
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.res.stringResource
import androidx.compose.ui.text.TextStyle
import androidx.compose.ui.text.font.FontWeight.Companion.Black
import androidx.compose.ui.text.font.FontWeight.Companion.Bold
import androidx.compose.ui.text.font.FontWeight.Companion.SemiBold
import androidx.compose.ui.text.style.TextAlign
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import androidx.hilt.navigation.compose.hiltViewModel
import androidx.navigation.NavController
import com.app.stepup.R
import com.app.stepup.constants.Constants.emptyUser
import com.app.stepup.navigation.ScreenRoutes
import com.app.stepup.ui.components.AchievementView
import com.app.stepup.ui.components.Background
import com.app.stepup.ui.theme.SoftGreen
import com.app.stepup.ui.theme.TransparentWhite
import com.app.stepup.view_models.HomeViewModel
import ir.ehsannarmani.compose_charts.ColumnChart
import ir.ehsannarmani.compose_charts.models.BarProperties
import ir.ehsannarmani.compose_charts.models.Bars
import ir.ehsannarmani.compose_charts.models.DividerProperties
```

```

import ir.ehsannarmani.compose_charts.models.GridProperties
import ir.ehsannarmani.compose_charts.models.HorizontalIndicatorProperties
import ir.ehsannarmani.compose_charts.models.LabelHelperProperties
import ir.ehsannarmani.compose_charts.models.LabelProperties
import ir.ehsannarmani.compose_charts.models.LineProperties
import ir.ehsannarmani.compose_charts.models.PopupProperties
import java.util.Locale

@Composable
fun HomeScreen(
    paddingValues: PaddingValues,
    navController: NavController,
    viewModel: HomeViewModel = hiltViewModel()
) {
    val sumSteps by viewModel.sumSteps.collectAsState(initial = 0)
    val dailySteps by viewModel.dailySteps.collectAsState(initial = emptyList())
    val user by viewModel.user.collectAsState(initial = emptyUser)
    val stepGoal by viewModel.stepGoal.collectAsState(initial = 0)
    val calories by viewModel.calories.collectAsState()
    val distance by viewModel.distance.collectAsState()
    val achievement by viewModel.achievement.collectAsState(initial = emptyList())

    LaunchedEffect(Unit) {
        viewModel.calculateActivityMetrics()
    }

    Background()
    Column(
        horizontalAlignment = Alignment.CenterHorizontally,
        modifier = Modifier
            .fillMaxSize()
            .padding(paddingValues)
    ) {
        Row(
            modifier = Modifier
                .fillMaxWidth()
                .padding(16.dp),
            verticalAlignment = Alignment.CenterVertically,
            horizontalArrangement = Arrangement.SpaceBetween
        ) {
            Title(user.name)

```

```

        IconButton(onClick = {
navController.navigate(ScreenRoutes.ProfileScreen.route) }) {
            Icon(
                painter = painterResource(id = R.drawable.ic_profile),
                contentDescription = R.drawable.ic_profile.toString(),
                tint = White,
                modifier = Modifier
                    .fillMaxHeight(0.9f)
                    .aspectRatio(1f)
            )
        }
    }
    CurrentStepsIndicator(
        modifier = Modifier
            .fillMaxWidth(0.9f)
            .aspectRatio(1f),
        sumSteps,
        stepGoal,
        calories,
        distance
    )
    Column(
        modifier = Modifier
            .fillMaxSize()
            .padding(16.dp)
    ) {
        AnalyticsView(
            dailySteps.map { it.toDouble() },
            Modifier
                .fillMaxWidth()
                .fillMaxHeight(0.5f)
                .padding(bottom = 8.dp)
                .background(TransparentWhite, shape = RoundedCornerShape(8.dp))
                .clickable {
                    navController.navigate(ScreenRoutes.AnalyticsScreen.route)
                }
        )
        Column(
            Modifier
                .fillMaxSize()
                .padding(top = 8.dp)

```

```

        .background(TransparentWhite, shape = RoundedCornerShape(8.dp))
        .clickable {
            navController.navigate(ScreenRoutes.AchievementsScreen.route)
        }
        .padding(16.dp)
    ) {
        Row(
            Modifier.fillMaxWidth(),
            verticalAlignment = Alignment.CenterVertically,
            horizontalArrangement = Arrangement.SpaceBetween
        ) {
            Text(
                text = stringResource(id = R.string.achievements),
                color = White,
                fontSize = 20.sp,
                fontWeight = Bold
            )
            Icon(
                painter = painterResource(id = R.drawable.ic_arrow_right),
                contentDescription = R.drawable.ic_arrow_right.toString(),
                tint = White
            )
        }
        LazyRow(Modifier.fillMaxSize()) {
            items(achievement.take(3)) {
                AchievementView(achieve = it)
            }
        }
    }
}

@Composable
fun Title(name: String) {
    Text(
        text = buildString {
            append(stringResource(id = R.string.hello))
            append(name)
            append(stringResource(id = R.string.exclamation_mark))
        },

```

```

        color = White,
        fontSize = 30.sp,
        fontWeight = Black,
        textAlign = TextAlign.Start
    )
}

```

@Composable

```

fun CurrentStepsIndicator(
    modifier: Modifier,
    stepsCount: Long,
    stepGoal: Int,
    calories: Double,
    distance: Double
) {
    Box(
        modifier = modifier,
        contentAlignment = Alignment.Center
    ) {
        CircularProgressIndicator(
            strokeWidth = 20.dp,
            strokeCap = StrokeCap.Round,
            trackColor = White,
            modifier = modifier.fillMaxSize(),
            progress = {
                (stepsCount.toFloat() / stepGoal)
            }
        )
        CurrentSteps(stepsCount, stepGoal, calories, distance)
    }
}

```

@Composable

```

fun CurrentSteps(stepsCount: Long, stepGoal: Int, calories: Double, distance: Double) {
    Column {
        Image(
            painter = painterResource(id = R.drawable.ic_steps),
            contentDescription = R.drawable.ic_steps.toString(),
            modifier = Modifier
                .fillMaxWidth(0.4f)
                .aspectRatio(1f)
        )
    }
}

```

```

    )
    Text(
        text = buildString {
            append(stepsCount)
            append(" / ")
            append(stepGoal)
        },
        modifier = Modifier.fillMaxWidth(0.4f),
        style = TextStyle(
            color = White,
            fontWeight = Black,
            textAlign = TextAlign.Center,
            fontSize = 20.sp,
            shadow = Shadow(
                color = MaterialTheme.colorScheme.primary, blurRadius = 20f
            )
        ),
    )
)
Row(
    modifier = Modifier
        .fillMaxWidth(0.4f)
        .padding(top = 6.dp),
    verticalAlignment = Alignment.CenterVertically,
    horizontalArrangement = Arrangement.SpaceEvenly
) {
    Text(
        text = buildString {
            append(calories.toInt())
            append("\n")
            append(stringResource(id = R.string.calories_abbreviation))
        },
        style = TextStyle(
            color = White,
            fontSize = 12.sp,
            fontWeight = Bold,
            textAlign = TextAlign.Center,
            shadow = Shadow(
                color = MaterialTheme.colorScheme.primary, blurRadius = 20f
            )
        )
    )
}

```

```

Text(
    text = buildString {
        append(String.format(Locale.getDefault(), "%.2f", distance))
        append("\n")
        append(stringResource(id = R.string.distance_abbreviation))
    },
    style = TextStyle(
        color = White,
        fontSize = 12.sp,
        fontWeight = Bold,
        textAlign = TextAlign.Center,
        shadow = Shadow(
            color = MaterialTheme.colorScheme.primary, blurRadius = 20f
        )
    )
)
}

}
}

```

@Composable

```

fun AnalyticsView(data: List<Double>, modifier: Modifier) {
    Column(modifier.padding(16.dp)) {
        Row(
            Modifier.fillMaxWidth(),
            verticalAlignment = Alignment.CenterVertically,
            horizontalArrangement = Arrangement.SpaceBetween
        ) {
            Text(
                text = stringResource(id = R.string.analytics),
                color = White,
                fontSize = 20.sp,
                fontWeight = Bold
            )
            Icon(
                painter = painterResource(id = R.drawable.ic_arrow_right),
                contentDescription = R.drawable.ic_arrow_right.toString(),
                tint = White
            )
        }
    }
}

```

```

if (data.isEmpty() or data.all { it == 0.0 }) {
    Column(
        Modifier.fillMaxSize(),
        horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.Center
    ) {
        Text(
            text = stringResource(id = R.string.no_data),
            fontSize = 14.sp,
            color = White,
            fontWeight = SemiBold
        )
        Text(
            text = stringResource(id = R.string.no_data_description),
            color = White,
            fontSize = 12.sp,
            textAlign = TextAlign.Center
        )
    }
} else {
    ColumnChart(
        modifier = Modifier
            .fillMaxSize()
            .padding(top = 8.dp),
        data = rememberSaveable {
            data.mapIndexed { index, steps ->
                Bars(
                    label = when {
                        index == 0 || index % 4 == 0 -> index.toString()
                        index == data.size - 1 -> (index + 1).toString()
                        else -> " "
                    },
                    values = listOf(
                        Bars.Data(
                            value = steps,
                            color = SolidColor(SoftGreen)
                        )
                    ),
                )
            }
        },
    ),
}

```

```

        indicatorProperties = HorizontalIndicatorProperties(enabled = false),
        gridProperties = GridProperties(enabled = false),
        labelHelperProperties = LabelHelperProperties(enabled = false),
        dividerProperties = DividerProperties(
            xAxisProperties = LineProperties(color = SolidColor(White)),
            yAxisProperties = LineProperties(color = SolidColor(White))
        ),
        popupProperties = PopupProperties(
            textStyle = TextStyle(color = White),
            containerColor = MaterialTheme.colorScheme.primary.copy(0.5f)
        ),
        labelProperties = LabelProperties(
            enabled = true,
            textStyle = TextStyle(
                color = White,
                fontSize = 10.sp
            ),
            rotationDegreeOnSizeConflict = 0f,
            padding = 6.dp
        ),
        barProperties = BarProperties(thickness = 10.dp),
    )
}
}
}

```

MainScreen.kt

```

package com.app.stepup.ui.screens

import androidx.compose.runtime.Composable
import androidx.compose.runtime.LaunchedEffect
import androidx.compose.runtime.collectAsState
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.setValue
import androidx.hilt.navigation.compose.hiltViewModel
import androidx.navigation.NavController
import com.app.stepup.navigation.ScreenRoutes
import com.app.stepup.view_models.MainViewModel

```

```

@Composable
fun MainScreen(
    navController: NavController,
    viewModel: MainViewModel = hiltViewModel()
) {
    var isSplashVisible by remember { mutableStateOf(true) }
    val isUserRegistered by viewModel.isUserRegistered.collectAsState()
    LaunchedEffect(isUserRegistered) {
        if (isSplashVisible) {
            isSplashVisible = false
        }
    }

    if (isSplashVisible) {
        SplashScreen()
    } else {
        if (isUserRegistered)
            navigateTo(navController, ScreenRoutes.HomeScreen.route)
        else
            navigateTo(navController, ScreenRoutes.RegistrationScreen.route)
    }
}

private fun navigateTo(navController: NavController, route: String) {
    navController.navigate(route) {
        popUpTo(navController.graph.startDestinationId) {
            inclusive = true
        }
    }
}

```

ProfileScreen.kt

```
package com.app.stepup.ui.screens
```

```

import androidx.compose.foundation.background
import androidx.compose.foundation.border
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.PaddingValues
import androidx.compose.foundation.layout.Row

```

```
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.width
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.foundation.text.KeyboardOptions
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Text
import androidx.compose.material3.TextField
import androidx.compose.material3.TextFieldDefaults
import androidx.compose.runtime.Composable
import androidx.compose.runtime.LaunchedEffect
import androidx.compose.runtime.collectAsState
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableIntStateOf
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.setValue
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.res.stringResource
import androidx.compose.ui.text.TextStyle
import androidx.compose.ui.text.input.KeyboardType
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import androidx.hilt.navigation.compose.hiltViewModel
import androidx.navigation.NavController
import com.app.stepup.R
import com.app.stepup.constants.Constants.emptyUser
import com.app.stepup.model.data.Gender
import com.app.stepup.navigation.ScreenRoutes
import com.app.stepup.ui.components.Background
import com.app.stepup.ui.components.DefaultButton
import com.app.stepup.ui.components.DropDownMenu
import com.app.stepup.ui.components.PairTitle
import com.app.stepup.ui.components.showToast
import com.app.stepup.ui.theme.TransparentWhite
```

```

import com.app.stepup.view_models.ProfileViewModel

@Composable
fun ProfileScreen(
    paddingValues: PaddingValues,
    navController: NavController,
    viewModel: ProfileViewModel = hiltViewModel()
) {
    val userVM by viewModel.user.collectAsState(initial = emptyUser)
    var user by remember { mutableStateOf(emptyUser) }
    var userWeight by remember { mutableStateOf("") }
    var userHeight by remember { mutableStateOf("") }

    val stepGoalVM by viewModel.stepGoal.collectAsState(initial = 0)
    var stepGoal by remember { mutableIntStateOf(0) }

    val agesList = (1..100).toList()
    val context = LocalContext.current

    LaunchedEffect(userVM) {
        user = userVM
        userHeight = userVM.height.toString()
        userWeight = userVM.weight.toString()
    }

    LaunchedEffect(stepGoalVM) {
        stepGoal = stepGoalVM
    }

    Box(
        Modifier
            .fillMaxSize()
            .padding(paddingValues),
        contentAlignment = Alignment.Center
    ) {
        Background()
        Column(
            Modifier
                .fillMaxWidth()
                .padding(32.dp)
                .background(TransparentWhite, RoundedCornerShape(8.dp)),

```

```

        verticalArrangement = Arrangement.SpaceAround,
        horizontalAlignment = Alignment.CenterHorizontally
    ) {
        Spacer(modifier = Modifier.height(32.dp))
        TextField(
            value = user.name,
            onChange = { user = user.copy(name = it) },
            modifier = Modifier
                .fillMaxWidth()
                .padding(horizontal = 16.dp)
                .border(1.dp, Color.White, RoundedCornerShape(8.dp)),
            textStyle = TextStyle(color = MaterialTheme.colorScheme.primary, fontSize
= 18.sp),
            placeholder = {
                Text(
                    text = stringResource(id = R.string.enter_name),
                    color = Color.White
                )
            },
            singleLine = true,
            colors = TextFieldDefaults.colors(
                focusedContainerColor = Color.Transparent,
                unfocusedContainerColor = Color.Transparent,
                disabledContainerColor = Color.Transparent,
                errorContainerColor = Color.Transparent,
                unfocusedTextColor = MaterialTheme.colorScheme.primary,
                focusedTextColor = MaterialTheme.colorScheme.primary,
                focusedIndicatorColor = Color.Transparent,
                unfocusedIndicatorColor = Color.Transparent
            )
        )
        Spacer(modifier = Modifier.height(32.dp))
        Column {
            Row(
                Modifier.fillMaxWidth(),
                verticalAlignment = Alignment.CenterVertically,
                horizontalArrangement = Arrangement.SpaceEvenly
            ) {
                PairTitle(firstTitle = R.string.gender, secondTitle = R.string.age)
            }
            Spacer(modifier = Modifier.height(6.dp))

```

```

Row(
    modifier = Modifier.fillMaxWidth(),
    verticalAlignment = Alignment.CenterVertically,
    horizontalArrangement = Arrangement.SpaceEvenly
) {
    DropdownMenu(
        items = Gender.entries,
        initialSelectedIndex = Gender.entries.indexOf(user.gender)
    ) {
        user.gender = Gender.valueOf(it.name)
    }
    DropdownMenu(
        items = agesList,
        initialSelectedIndex = if (user.age != 0) user.age - 1 else 0
    ) {
        user.age = it
    }
}
Spacer(modifier = Modifier.height(16.dp))
Row(
    Modifier.fillMaxWidth(),
    verticalAlignment = Alignment.CenterVertically,
    horizontalArrangement = Arrangement.SpaceEvenly
) {
    PairTitle(firstTitle = R.string.weight, secondTitle = R.string.height)
}
Spacer(modifier = Modifier.height(6.dp))
Row(
    modifier = Modifier.fillMaxWidth(),
    verticalAlignment = Alignment.CenterVertically,
    horizontalArrangement = Arrangement.SpaceEvenly
) {
    TextField(
        value = userWeight,
        onValueChange = { userWeight = it },
        modifier = Modifier
            .border(1.dp, Color.White, RoundedCornerShape(8.dp))
            .padding(horizontal = 6.dp)
            .width(100.dp),
        textStyle = TextStyle(
            color = MaterialTheme.colorScheme.primary,

```

```

        fontSize = 18.sp
    ),
    singleLine = true,
    colors = TextFieldDefaults.colors(
        focusedContainerColor = Color.Transparent,
        unfocusedContainerColor = Color.Transparent,
        disabledContainerColor = Color.Transparent,
        errorContainerColor = Color.Transparent,
        unfocusedTextColor = MaterialTheme.colorScheme.primary,
        focusedTextColor = MaterialTheme.colorScheme.primary,
        focusedIndicatorColor = Color.Transparent,
        unfocusedIndicatorColor = Color.Transparent
    ),
    keyboardOptions = KeyboardOptions.Default.copy(
        keyboardType = KeyboardType.Number
    ),
    placeholder = {
        Text(
            text = stringResource(id = R.string.mask_of_entered),
            color = Color.White
        )
    }
)
TextField(
    value = userHeight,
    onValueChange = { userHeight = it },
    modifier = Modifier
        .border(1.dp, Color.White, RoundedCornerShape(8.dp))
        .padding(horizontal = 6.dp)
        .width(100.dp),
    textStyle = TextStyle(
        color = MaterialTheme.colorScheme.primary,
        fontSize = 18.sp
    ),
    singleline = true,
    colors = TextFieldDefaults.colors(
        focusedContainerColor = Color.Transparent,
        unfocusedContainerColor = Color.Transparent,
        disabledContainerColor = Color.Transparent,
        errorContainerColor = Color.Transparent,
        unfocusedTextColor = MaterialTheme.colorScheme.primary,

```

```

        focusedTextColor = MaterialTheme.colorScheme.primary,
        focusedIndicatorColor = Color.Transparent,
        unfocusedIndicatorColor = Color.Transparent
    ),
    keyboardOptions = KeyboardOptions.Default.copy(
        keyboardType = KeyboardType.Number
    ),
    placeholder = {
        Text(
            text = stringResource(id = R.string.mask_of_entered),
            color = Color.White
        )
    }
)
}
}
Spacer(modifier = Modifier.height(32.dp))
TextField(
    value = if (stepGoal == 0) "" else stepGoal.toString(),
    onValueChange = { stepGoal = if (it == "") 0 else it.toInt() },
    modifier = Modifier
        .fillMaxWidth()
        .padding(horizontal = 16.dp)
        .border(1.dp, Color.White, RoundedCornerShape(8.dp)),
    textStyle = TextStyle(color = MaterialTheme.colorScheme.primary, fontSize
= 18.sp),
    placeholder = {
        Text(
            text = stringResource(id = R.string.enter_step_goal),
            color = Color.White
        )
    },
    singleLine = true,
    colors = TextFieldDefaults.colors(
        focusedContainerColor = Color.Transparent,
        unfocusedContainerColor = Color.Transparent,
        disabledContainerColor = Color.Transparent,
        errorContainerColor = Color.Transparent,
        unfocusedTextColor = MaterialTheme.colorScheme.primary,
        focusedTextColor = MaterialTheme.colorScheme.primary,
        focusedIndicatorColor = Color.Transparent,

```

```

        unfocusedIndicatorColor = Color.Transparent
    ),
    keyboardOptions = KeyboardOptions.Default.copy(
        keyboardType = KeyboardType.Number
    )
)
Spacer(modifier = Modifier.height(32.dp))
DefaultButton(R.string.save_changes) {
    when {
        user.name.isEmpty() -> showToast(context, R.string.pls_enter_name)
        user.age == 0 -> showToast(context, R.string.pls_enter_age)
        userWeight == "" -> showToast(context, R.string.pls_enter_weight)
        userHeight == "" -> showToast(context, R.string.pls_enter_height)
        stepGoal == 0 -> showToast(context, R.string.pls_enter_step_goal)
        else -> {
            viewModel.setUserData(
                user.copy(
                    weight = userWeight.toDoubleOrNull() ?: 0.0,
                    height = userHeight.toDoubleOrNull() ?: 0.0
                ), stepGoal
            )
            navController.navigate(ScreenRoutes.HomeScreen.route)
        }
    }
}
Spacer(modifier = Modifier.height(32.dp))
}
}
}

```

RegistrationScreen.kt

```
package com.app.stepup.ui.screens
```

```
import androidx.compose.foundation.background
import androidx.compose.foundation.border
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.PaddingValues
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.Spacer
```

```

import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.width
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.foundation.text.KeyboardOptions
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Text
import androidx.compose.material3.TextField
import androidx.compose.material3.TextFieldDefaults
import androidx.compose.runtime.Composable
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableIntStateOf
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.saveable.rememberSaveable
import androidx.compose.runtime.setValue
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.res.stringResource
import androidx.compose.ui.text.TextStyle
import androidx.compose.ui.text.input.KeyboardType
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import androidx.hilt.navigation.compose.hiltViewModel
import androidx.navigation.NavController
import com.app.stepup.R
import com.app.stepup.model.data.Gender
import com.app.stepup.model.data.User
import com.app.stepup.navigation.ScreenRoutes
import com.app.stepup.ui.components.Background
import com.app.stepup.ui.components.DefaultButton
import com.app.stepup.ui.components.DropDownMenu
import com.app.stepup.ui.components.PairTitle
import com.app.stepup.ui.components.showToast
import com.app.stepup.ui.theme.TransparentWhite
import com.app.stepup.view_models.RegistrationViewModel

```

```
@Composable
```

```

fun RegistrationScreen(
    paddingValues: PaddingValues,
    navController: NavController,
    viewModel: RegistrationViewModel = hiltViewModel()
) {
    var userName by rememberSaveable { mutableStateOf("") }
    var userAge by rememberSaveable { mutableIntStateOf(0) }
    var userGender by rememberSaveable { mutableStateOf(Gender.OTHER.name) }
    var userHeight by rememberSaveable { mutableStateOf("") }
    var userWeight by rememberSaveable { mutableStateOf("") }
    var stepGoal by rememberSaveable { mutableIntStateOf(0) }

    val agesList = (1..100).toList()
    val context = LocalContext.current

    Box(
        modifier = Modifier
            .fillMaxSize()
            .padding(paddingValues),
        contentAlignment = Alignment.Center
    ) {
        Background()
        Column(
            Modifier
                .fillMaxWidth()
                .padding(32.dp)
                .background(TransparentWhite, RoundedCornerShape(8.dp)),
            verticalArrangement = Arrangement.SpaceAround,
            horizontalAlignment = Alignment.CenterHorizontally
        ) {
            Spacer(modifier = Modifier.height(32.dp))
            TextField(
                value = userName,
                onValueChange = { userName = it },
                modifier = Modifier
                    .fillMaxWidth()
                    .padding(horizontal = 16.dp)
                    .border(1.dp, Color.White, RoundedCornerShape(8.dp)),
                textStyle = TextStyle(color = MaterialTheme.colorScheme.primary, fontSize
= 18.sp),
                placeholder = {

```

```

        Text(
            text = stringResource(id = R.string.enter_name),
            color = Color.White
        )
    },
    singleLine = true,
    colors = TextFieldDefaults.colors(
        focusedContainerColor = Color.Transparent,
        unfocusedContainerColor = Color.Transparent,
        disabledContainerColor = Color.Transparent,
        errorContainerColor = Color.Transparent,
        unfocusedTextColor = MaterialTheme.colorScheme.primary,
        focusedTextColor = MaterialTheme.colorScheme.primary,
        focusedIndicatorColor = Color.Transparent,
        unfocusedIndicatorColor = Color.Transparent
    )
)
Spacer(modifier = Modifier.height(32.dp))
Column {
    Row(
        Modifier.fillMaxWidth(),
        verticalAlignment = Alignment.CenterVertically,
        horizontalArrangement = Arrangement.SpaceEvenly
    ) {
        PairTitle(firstTitle = R.string.gender, secondTitle = R.string.age)
    }
    Spacer(modifier = Modifier.height(6.dp))
    Row(
        modifier = Modifier.fillMaxWidth(),
        verticalAlignment = Alignment.CenterVertically,
        horizontalArrangement = Arrangement.SpaceEvenly
    ) {
        DropdownMenu(items = Gender.entries) {
            userGender = it.name
        }
        DropdownMenu(items = agesList) {
            userAge = it
        }
    }
    Spacer(modifier = Modifier.height(16.dp))
    Row(

```

```

        Modifier.fillMaxWidth(),
        verticalAlignment = Alignment.CenterVertically,
        horizontalArrangement = Arrangement.SpaceEvenly
    ) {
        PairTitle(firstTitle = R.string.weight, secondTitle = R.string.height)
    }
    Spacer(modifier = Modifier.height(6.dp))
    Row(
        modifier = Modifier.fillMaxWidth(),
        verticalAlignment = Alignment.CenterVertically,
        horizontalArrangement = Arrangement.SpaceEvenly
    ) {
        TextField(
            value = userWeight,
            onValueChange = { userWeight = it },
            modifier = Modifier
                .border(1.dp, Color.White, RoundedCornerShape(8.dp))
                .padding(horizontal = 6.dp)
                .width(100.dp),
            textStyle = TextStyle(
                color = MaterialTheme.colorScheme.primary,
                fontSize = 18.sp
            ),
            singleline = true,
            colors = TextFieldDefaults.colors(
                focusedContainerColor = Color.Transparent,
                unfocusedContainerColor = Color.Transparent,
                disabledContainerColor = Color.Transparent,
                errorContainerColor = Color.Transparent,
                unfocusedTextColor = MaterialTheme.colorScheme.primary,
                focusedTextColor = MaterialTheme.colorScheme.primary,
                focusedIndicatorColor = Color.Transparent,
                unfocusedIndicatorColor = Color.Transparent
            ),
            keyboardOptions = KeyboardOptions.Default.copy(
                keyboardType = KeyboardType.Number
            ),
            placeholder = {
                Text(
                    text = stringResource(id = R.string.mask_of_entered),
                    color = Color.White
                )
            }
        )
    }
}

```

```

        )
    }
)
TextField(
    value = userHeight,
    onValueChange = { userHeight = it },
    modifier = Modifier
        .border(1.dp, Color.White, RoundedCornerShape(8.dp))
        .padding(horizontal = 6.dp)
        .width(100.dp),
    textStyle = TextStyle(
        color = MaterialTheme.colorScheme.primary,
        fontSize = 18.sp
    ),
    singleLine = true,
    colors = TextFieldDefaults.colors(
        focusedContainerColor = Color.Transparent,
        unfocusedContainerColor = Color.Transparent,
        disabledContainerColor = Color.Transparent,
        errorContainerColor = Color.Transparent,
        unfocusedTextColor = MaterialTheme.colorScheme.primary,
        focusedTextColor = MaterialTheme.colorScheme.primary,
        focusedIndicatorColor = Color.Transparent,
        unfocusedIndicatorColor = Color.Transparent
    ),
    keyboardOptions = KeyboardOptions.Default.copy(
        keyboardType = KeyboardType.Number
    ),
    placeholder = {
        Text(
            text = stringResource(id = R.string.mask_of_entered),
            color = Color.White
        )
    }
)
}

Spacer(modifier = Modifier.height(32.dp))
TextField(
    value = if (stepGoal == 0) "" else stepGoal.toString(),
    onValueChange = { stepGoal = if (it == "") 0 else it.toInt() },

```

```

modifier = Modifier
    .fillMaxWidth()
    .padding(horizontal = 16.dp)
    .border(1.dp, Color.White, RoundedCornerShape(8.dp)),
textStyle = TextStyle(color = MaterialTheme.colorScheme.primary, fontSize
= 18.sp),

placeholder = {
    Text(
        text = stringResource(id = R.string.enter_step_goal),
        color = Color.White
    )
},
singleLine = true,
colors = TextFieldDefaults.colors(
    focusedContainerColor = Color.Transparent,
    unfocusedContainerColor = Color.Transparent,
    disabledContainerColor = Color.Transparent,
    errorContainerColor = Color.Transparent,
    unfocusedTextColor = MaterialTheme.colorScheme.primary,
    focusedTextColor = MaterialTheme.colorScheme.primary,
    focusedIndicatorColor = Color.Transparent,
    unfocusedIndicatorColor = Color.Transparent
),
keyboardOptions = KeyboardOptions.Default.copy(
    keyboardType = KeyboardType.Number
)
)
Spacer(modifier = Modifier.height(32.dp))
DefaultButton(R.string.registration) {
    when {
        userName.isEmpty() -> showToast(context, R.string.pls_enter_name)
        userAge == 0 -> showToast(context, R.string.pls_enter_age)
        userWeight == "" -> showToast(context, R.string.pls_enter_weight)
        userHeight == "" -> showToast(context, R.string.pls_enter_height)
        stepGoal == 0 -> showToast(context, R.string.pls_enter_step_goal)
        else -> {
            viewModel.setUserData(
                User(
                    userName,
                    userAge,
                    Gender.valueOf(userGender),

```

```
                userHeight.toDouble(),  
                userWeight.toDouble()  
            ),  
            stepGoal  
        )  
        navController.navigate(ScreenRoutes.HomeScreen.route)  
    }  
}  
  
Spacer(modifier = Modifier.height(32.dp))  
}  
}
```

SplashScreen.kt

```
package com.app.stepup.ui.screens
```

```
import androidx.compose.foundation.Image
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.aspectRatio
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.shape.CircleShape
import androidx.compose.runtime.Composable
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.clip
import androidx.compose.ui.res.painterResource
import com.app.stepup.R
import com.app.stepup.ui.components.Background
```

@Composable

```
fun SplashScreen() {  
    Box(  
        modifier = Modifier.fillMaxSize()  
    ) {  
        Background()  
        Column(  
            verticalArrangement = Arrangement.Center,
```

```

        horizontalAlignment = Alignment.CenterHorizontally,
        modifier = Modifier.fillMaxSize()
    ) {
        Image(
            painter = painterResource(id = R.drawable.ic_logo),
            contentDescription = R.drawable.ic_logo.toString(),
            modifier = Modifier
                .fillMaxWidth(0.6f)
                .aspectRatio(1f)
                .clip(CircleShape)
        )
    }
}

```

Color.kt

```
package com.app.stepup.ui.theme
```

```
import androidx.compose.ui.graphics.Color
```

```

val Lavender = Color(0xFFE1BEE7)
val SoftLilac = Color(0xFFCE93D8)
val PastelPurple = Color(0xFFB39DDB)
val LavenderBlue = Color(0xFF9FA8DA)
val PastelBlue = Color(0xFF90CAF9)
val PastelPink = Color(0xFFFFC1E3)
val TransparentWhite = Color(0x66FFFFFF)

```

```

val RoyalPurple = Color(0xFF673AB7)
val DeepIndigo = Color(0xFF3F51B5)
val Plum = Color(0xFF6A1B9A)
val DarkSlateBlue = Color(0xFF3949AB)

```

```

val SoftGreen = Color(0xFF81C784)
val WarmCoral = Color(0xFFFF7043)
val SoftTeal = Color(0xFF4DB6AC)

```

Theme.kt

```
package com.app.stepup.ui.theme
```

```
import android.os.Build
```

```

import androidx.compose.foundation.isSystemInDarkTheme
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.darkColorScheme
import androidx.compose.material3.dynamicDarkColorScheme
import androidx.compose.material3.dynamicLightColorScheme
import androidx.compose.material3.lightColorScheme
import androidx.compose.runtime.Composable
import androidx.compose.ui.platform.LocalContext

private val DarkColorScheme = darkColorScheme(
    primary = DarkSlateBlue,
    secondary = DeepIndigo,
    tertiary = PastelBlue
)

private val LightColorScheme = lightColorScheme(
    primary = Plum,
    secondary = RoyalPurple,
    tertiary = PastelPurple
)

@Composable
fun StepUpTheme(
    darkTheme: Boolean = isSystemInDarkTheme(),
    dynamicColor: Boolean = true,
    content: @Composable () -> Unit
) {
    val colorScheme = when {
        dynamicColor && Build.VERSION.SDK_INT >= Build.VERSION_CODES.S -> {
            val context = LocalContext.current
            if (darkTheme) dynamicDarkColorScheme(context) else
dynamicLightColorScheme(context)
        }

        darkTheme -> DarkColorScheme
        else -> LightColorScheme
    }

    MaterialTheme(
        colorScheme = colorScheme,
        typography = Typography,

```

```

        content = content
    )
}

```

Type.kt

```
package com.app.stepup.ui.theme
```

```

import androidx.compose.material3.Typography
import androidx.compose.ui.text.TextStyle
import androidx.compose.ui.text.font.FontFamily
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.unit.sp

```

```

val Typography = Typography(
    bodyLarge = TextStyle(
        fontFamily = FontFamily.Default,
        fontWeight = FontWeight.Normal,
        fontSize = 16.sp,
        lineHeight = 24.sp,
        letterSpacing = 0.5.sp
    )
)

```

AchievementsManager.kt

```
package com.app.stepup.utils
```

```

import com.app.stepup.constants.Constants.ACTIVE_MONTH
import com.app.stepup.constants.Constants.BILLION
import com.app.stepup.constants.Constants.BURNED_BIG_MAC
import com.app.stepup.constants.Constants.BURNED_CAKE
import com.app.stepup.constants.Constants.BURNED_PIZZA
import com.app.stepup.constants.Constants.CIRCLED_THE_WORLD
import com.app.stepup.constants.Constants.CIRCLED_THE_WORLD_TWICE
import com.app.stepup.constants.Constants.DAILY_NORM
import com.app.stepup.constants.Constants.EVEREST_SUMMIT
import com.app.stepup.constants.Constants.FIFTY
import com.app.stepup.constants.Constants.FIVE
import com.app.stepup.constants.Constants.FIVE_THOUSAND
import com.app.stepup.constants.Constants.MARATHON_DISTANCE
import com.app.stepup.constants.Constants.MILLION
import com.app.stepup.constants.Constants.MONTH_WORTH

```

```

import com.app.stepup.constants.Constants.ONE_HUNDRED
import com.app.stepup.constants.Constants.ONE_HUNDRED_THOUSAND
import com.app.stepup.constants.Constants.ONE_THOUSAND
import com.app.stepup.constants.Constants.TEN
import com.app.stepup.constants.Constants.TEN_THOUSAND
import com.app.stepup.constants.Constants.TRIP_TO_THE_MOON
import com.app.stepup.constants.Constants.TWO_HUNDRED
import com.app.stepup.model.StepRepository
import com.app.stepup.model.data.Achievement
import com.app.stepup.model.data.Name
import com.app.stepup.model.data.Status
import com.app.stepup.model.datastore.StepUpPreferencesRepository
import kotlinx.coroutines.flow.first
import java.time.LocalDate
import javax.inject.Inject
import javax.inject.Singleton

@Singleton
class AchievementsManager @Inject constructor(
    private val repository: StepUpPreferencesRepository,
    private val stepRepository: StepRepository,
    private val calculator: ActivityMetricsCalculator
) {
    private var allSteps: Long = 0
    private var stepsByToday: Long = 0
    private var stepsByWeek: Long = 0
    private var stepsByMonth: Long = 0
    private var stepsByYear: Long = 0
    private var userHeight: Double = 0.0

    suspend fun checkAchievements(achievements: List<Achievement>) {
        allSteps = stepRepository.getTotalSteps().first()
        val monthlyData = stepRepository.getMonthlySteps()
        stepsByToday = stepRepository.getDailySteps()
        stepsByWeek = calculator.getCurrentWeekSteps(monthlyData).sumOf { it.steps } +
stepsByToday
        stepsByMonth = monthlyData.sumOf { it.steps } + stepsByToday
        stepsByYear =
            stepRepository.getYearlySteps(LocalDate.now().year.toString()).sumOf {
it.steps }
        userHeight = repository.getUserData().first().height
    }

```

```

    achievements.map {
        when (it.status) {
            Status.DONE -> {}
            Status.UNDONE -> checkAchievement(it)
        }
    }
}

private suspend fun checkAchievement(achievement: Achievement) {
    when (achievement.name) {
        Name.MARATHONER -> checkMarathoner(achievement)
        Name.SUPER_WALKER -> checkSuperWalker(achievement)
        Name.MILLION_STEPS -> checkMillionSteps(achievement)
        Name.CIRCLED_THE_EARTH -> checkCircledTheWorld(achievement)
        Name.ACTIVE_MONTH -> checkActiveMonth(achievement)
        Name.EVEREST_SUMMIT -> checkEverestSummit(achievement)
        Name.TRIP_TO_THE_MOON -> checkTripToTheMoon(achievement)
        Name.BILLION_STEPS -> checkBillionSteps(achievement)
        Name.BURNED_PIZZA -> checkBurnedPizza(achievement)
        Name.MINI_MARATHON -> checkMiniMarathon(achievement)
        Name.BURNED_CAKE -> checkBurnedCake(achievement)
        Name.BURNED_DAILY_NORM -> checkBurnedDailyNorm(achievement)
        Name.TEN_K_CALORIES -> checkTenKCalories(achievement)
        Name.MONTH_WORTH_CALORIES -> checkMonthWorth(achievement)
        Name.ONE_HUNDRED_THOUSAND_CALORIES ->
checkOneHundredThousandCalories(achievement)
        Name.BURNED_BIG_MAC -> checkBurnedBigMac(achievement)
        Name.MILLION_CALORIES -> checkMillionCalories(achievement)
        Name.FIVE_KM_ON_FOOT -> checkFiveKmOnFoot(achievement)
        Name.GOLDEN_TEN -> checkGoldenTen(achievement)
        Name.FIFTY_KM_OF_JOURNEY -> checkFiftyKmOnJourney(achievement)
        Name.WALKING_TO_WORK -> checkWalkingToWork(achievement)
        Name.TWO_HUNDRED_KM_WEEKEND -> checkTwoHundredKmWeekend(achievement)
        Name.ONE_THOUSAND_KM_IN_YEAR -> checkOneThousandKmInYear(achievement)
        Name.CONTINENTAL_TRAVELER -> checkContinentalTraveler(achievement)
        Name.ULTRA_HIKE -> checkUltraHike(achievement)
        Name.CIRCLED_THE_EARTH_TWICE -> checkCircledTheWorldTwice(achievement)
    }
}

private suspend fun checkMarathoner(achievement: Achievement) {

```

```

        if (allSteps.toInt() >= MARATHON_DISTANCE) updateAchieve(achievement)
    }

    private suspend fun checkSuperWalker(achievement: Achievement) {
        if (allSteps.toInt() >= ONE_HUNDRED_THOUSAND) updateAchieve(achievement)
    }

    private suspend fun checkMillionSteps(achievement: Achievement) {
        if (allSteps.toInt() >= MILLION) updateAchieve(achievement)
    }

    private suspend fun checkCircledTheWorld(achievement: Achievement) {
        if (allSteps.toInt() >= CIRCLED_THE_WORLD) updateAchieve(achievement)
    }

    private suspend fun checkActiveMonth(achievement: Achievement) {
        if (stepsByMonth.toInt() >= ACTIVE_MONTH) updateAchieve(achievement)
    }

    private suspend fun checkEverestSummit(achievement: Achievement) {
        if (allSteps.toInt() >= EVEREST_SUMMIT) updateAchieve(achievement)
    }

    private suspend fun checkTripToTheMoon(achievement: Achievement) {
        if (allSteps.toInt() >= TRIP_TO_THE_MOON) updateAchieve(achievement)
    }

    private suspend fun checkBillionSteps(achievement: Achievement) {
        if (allSteps.toInt() >= BILLION) updateAchieve(achievement)
    }

    private suspend fun checkBurnedPizza(achievement: Achievement) {
        if (getCaloriesByToday() >= BURNED_PIZZA) updateAchieve(achievement)
    }

    private suspend fun checkMiniMarathon(achievement: Achievement) {
        if (getCaloriesByWeek() >= ONE_THOUSAND) updateAchieve(achievement)
    }

    private suspend fun checkBurnedCake(achievement: Achievement) {
        if (getAllCalories() >= BURNED_CAKE) updateAchieve(achievement)
    }

```

```

}

private suspend fun checkBurnedDailyNorm(achievement: Achievement) {
    if (getCaloriesByToday() >= DAILY_NORM) updateAchieve(achievement)
}

private suspend fun checkTenKCalories(achievement: Achievement) {
    if (getAllCalories() >= TEN_THOUSAND) updateAchieve(achievement)
}

private suspend fun checkMonthWorth(achievement: Achievement) {
    if (getCaloriesByMonth() >= MONTH_WORTH) updateAchieve(achievement)
}

private suspend fun checkOneHundredThousandCalories(achievement: Achievement) {
    if (getAllCalories() >= ONE_HUNDRED_THOUSAND) updateAchieve(achievement)
}

private suspend fun checkBurnedBigMac(achievement: Achievement) {
    if (getAllCalories() >= BURNED_BIG_MAC) updateAchieve(achievement)
}

private suspend fun checkMillionCalories(achievement: Achievement) {
    if (getAllCalories() >= MILLION) updateAchieve(achievement)
}

private suspend fun checkFiveKmOnFoot(achievement: Achievement) {
    if (getDistanceByToday() >= FIVE) updateAchieve(achievement)
}

private suspend fun checkGoldenTen(achievement: Achievement) {
    if (getDistanceByToday() >= TEN) updateAchieve(achievement)
}

private suspend fun checkFiftyKmOnJourney(achievement: Achievement) {
    if (getAllDistance() >= FIFTY) updateAchieve(achievement)
}

private suspend fun checkWalkingToWork(achievement: Achievement) {
    if (getAllDistance() >= ONE_HUNDRED) updateAchieve(achievement)
}

```

```

private suspend fun checkTwoHundredKmWeekend(achievement: Achievement) {
    if (getDistanceByMonth() >= TWO_HUNDRED) updateAchieve(achievement)
}

private suspend fun checkOneThousandKmInYear(achievement: Achievement) {
    if (getDistanceByYear() >= ONE_THOUSAND) updateAchieve(achievement)
}

private suspend fun checkContinentalTraveler(achievement: Achievement) {
    if (getAllDistance() >= FIVE_THOUSAND) updateAchieve(achievement)
}

private suspend fun checkUltraHike(achievement: Achievement) {
    if (getAllDistance() >= TEN_THOUSAND) updateAchieve(achievement)
}

private suspend fun checkCircledTheWorldTwice(achievement: Achievement) {
    if (getAllDistance() >= CIRCLED_THE_WORLD_TWICE) updateAchieve(achievement)
}

private fun getCaloriesByToday(): Int =
calculator.calculateCalories(stepsByToday).toInt()

private fun getCaloriesByWeek(): Int =
calculator.calculateCalories(stepsByWeek).toInt()

private fun getCaloriesByMonth(): Int =
calculator.calculateCalories(stepsByMonth).toInt()

private fun getAllCalories(): Int = calculator.calculateCalories(allSteps).toInt()

private fun getAllDistance(): Int =
calculator.calculateDistanceInKm(userHeight, allSteps).toInt()

private fun getDistanceByToday(): Int =
calculator.calculateDistanceInKm(userHeight, stepsByToday).toInt()

private fun getDistanceByMonth(): Int =
calculator.calculateDistanceInKm(userHeight, stepsByMonth).toInt()

```

```

private fun getDistanceByYear(): Int =
    calculator.calculateDistanceInKm(userHeight, stepsByYear).toInt()

private suspend fun updateAchieve(achievement: Achievement) {
    val oldAchievements = repository.getAchievements().first()
    val updatedAchievements = oldAchievements.map {
        if (it.name == achievement.name) {
            it.copy(status = Status.DONE)
        } else {
            it
        }
    }
    repository.setAchievements(updatedAchievements)
}
}

ActivityMetricsCalculator.kt
package com.app.stepup.utils

import com.app.stepup.model.room.MonthlyStepData
import java.time.LocalDate
import java.time.format.DateTimeFormatter
import java.time.temporal.WeekFields
import java.util.Locale
import javax.inject.Inject
import javax.inject.Singleton

@Singleton
class ActivityMetricsCalculator @Inject constructor() {
    fun calculateCalories(steps: Long): Double {
        return steps * CALORIES_PER_STEP
    }

    fun calculateDistanceInKm(height: Double, steps: Long): Double {
        val stepLength = (height / 100.0) * STEP_LENGTH_RATIO
        return (steps * stepLength) / 1000
    }

    fun getCurrentWeekSteps(data: List<MonthlyStepData>): List<MonthlyStepData> {
        val today = LocalDate.now()
        val currentWeekFields = WeekFields.of(Locale.getDefault())

```

```

        val startOfWeek = today.with(currentWeekFields.dayOfWeek(), 1)
        val endOfWeek = today.with(currentWeekFields.dayOfWeek(), 7)

        return data.filter { stepData ->
            val date = LocalDate.parse(
                "${stepData.month}-${stepData.day.toString().padStart(2, '0')}",
                DateTimeFormatter.ofPattern("yyyy-MM-dd")
            )
            date in startOfWeek..endOfWeek
        }
    }

    companion object {
        private const val STEP_LENGTH_RATIO = 0.413
        private const val CALORIES_PER_STEP = 0.04
    }
}

```

DateParser.kt

```
package com.app.stepup.utils
```

```

import java.time.LocalDate
import java.time.YearMonth
import java.time.format.DateTimeFormatter
import javax.inject.Inject
import javax.inject.Singleton

```

```
@Singleton
```

```

class DateParser @Inject constructor() {
    fun getDayFromDate(dateString: String): Int {
        val formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd")
        val date = LocalDate.parse(dateString, formatter)
        return date.dayOfMonth
    }

    fun getMonthAndYearFromDate(dateString: String): Pair<Int, String> {
        val formatter = DateTimeFormatter.ofPattern("yyyy-MM")
        val yearMonth = YearMonth.parse(dateString, formatter)
        return Pair(yearMonth.monthValue, yearMonth.year.toString())
    }
}

```

```
}
```

```
AchievementsViewModel.kt
```

```
package com.app.stepup.view_models
```

```
import androidx.lifecycle.ViewModel
```

```
import androidx.lifecycle.viewModelScope
```

```
import com.app.stepup.model.data.Achievement
```

```
import com.app.stepup.model.datastore.StepUpPreferencesRepository
```

```
import com.app.stepup.utils.AchievementsManager
```

```
import dagger.hilt.android.lifecycle.HiltViewModel
```

```
import kotlinx.coroutines.Dispatchers
```

```
import kotlinx.coroutines.launch
```

```
import javax.inject.Inject
```

```
@HiltViewModel
```

```
class AchievementsViewModel @Inject constructor(
```

```
    private val repository: StepUpPreferencesRepository,
```

```
    private val achievementsManager: AchievementsManager
```

```
) : ViewModel() {
```

```
    val achievements = repository.getAchievements()
```

```
    fun checkAchievements(achievements: List<Achievement>) {
```

```
        viewModelScope.launch(Dispatchers.IO) {
```

```
            achievementsManager.checkAchievements(achievements)
```

```
        }
```

```
    }
```

```
}
```

```
AnalyticsViewModel.kt
```

```
package com.app.stepup.view_models
```

```
import androidx.lifecycle.ViewModel
```

```
import androidx.lifecycle.viewModelScope
```

```
import com.app.stepup.model.StepRepository
```

```
import com.app.stepup.model.datastore.StepUpPreferencesRepository
```

```
import com.app.stepup.model.room.MonthlyStepData
```

```
import com.app.stepup.model.room.YearlyStepData
```

```
import com.app.stepup.utils.ActivityMetricsCalculator
```

```
import dagger.hilt.android.lifecycle.HiltViewModel
```

```
import kotlinx.coroutines.Dispatchers
```

```

import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.flow.first
import kotlinx.coroutines.launch
import java.time.LocalDate
import javax.inject.Inject

@HiltViewModel
class AnalyticsViewModel @Inject constructor(
    private val stepRepository: StepRepository,
    private val calculator: ActivityMetricsCalculator,
    private val repository: StepUpPreferencesRepository
) : ViewModel() {

    init {
        calculateSteps()
    }

    private var _stepsByWeek: MutableStateFlow<Map<String, Double>> =
MutableStateFlow(emptyMap())
    private var _stepsByMonth: MutableStateFlow<Map<Int, Double>> =
MutableStateFlow(emptyMap())
    private var _stepsByYear: MutableStateFlow<Map<String, Double>> =
MutableStateFlow(emptyMap())

    private var _caloriesByWeek: MutableStateFlow<Map<String, Double>> =
        MutableStateFlow(emptyMap())
    private var _caloriesByMonth: MutableStateFlow<Map<Int, Double>> =
MutableStateFlow(emptyMap())
    private var _caloriesByYear: MutableStateFlow<Map<String, Double>> =
        MutableStateFlow(emptyMap())

    private var _distanceByWeek: MutableStateFlow<Map<String, Double>> =
        MutableStateFlow(emptyMap())
    private var _distanceByMonth: MutableStateFlow<Map<Int, Double>> =
MutableStateFlow(emptyMap())
    private var _distanceByYear: MutableStateFlow<Map<String, Double>> =
        MutableStateFlow(emptyMap())

    val stepsByWeek = _stepsByWeek
    val stepsByMonth = _stepsByMonth
    val stepsByYear = _stepsByYear

```

```

val caloriesByWeek = _caloriesByWeek
val caloriesByMonth = _caloriesByMonth
val caloriesByYear = _caloriesByYear

val distanceByWeek = _distanceByWeek
val distanceByMonth = _distanceByMonth
val distanceByYear = _distanceByYear

fun calculateCalories() {
    viewModelScope.launch(Dispatchers.IO) {
        _caloriesByWeek.emit(
            stepsByWeek.first().mapValues { (_, steps) ->
                calculator.calculateCalories(steps.toLong())
            }
        )
        _caloriesByMonth.emit(
            stepsByMonth.first().mapValues { (_, steps) ->
                calculator.calculateCalories(steps.toLong())
            }
        )
        _caloriesByYear.emit(
            stepsByYear.first().mapValues { (_, steps) ->
                calculator.calculateCalories(steps.toLong())
            }
        )
    }
}

fun calculateDistance() {
    viewModelScope.launch(Dispatchers.IO) {
        val userHeight = repository.getUserData().first().height
        _distanceByWeek.emit(
            stepsByWeek.first().mapValues { (_, steps) ->
                calculator.calculateDistanceInKm(userHeight, steps.toLong())
            }
        )
        _distanceByMonth.emit(
            stepsByMonth.first().mapValues { (_, steps) ->
                calculator.calculateDistanceInKm(userHeight, steps.toLong())
            }
        )
    }
}

```

```

        )
        _distanceByYear.emit(
            stepsByYear.first().mapValues { (_, steps) ->
                calculator.calculateDistanceInKm(userHeight, steps.toLong())
            }
        )
    }
}

private fun calculateSteps() {
    viewModelScope.launch(Dispatchers.IO) {
        val dailySteps = stepRepository.getDailySteps()
        val monthlyStepData: List<MonthlyStepData> = stepRepository.getMonthlySteps()
        val weeklySteps: List<MonthlyStepData> =
calculator.getCurrentWeekSteps(monthlyStepData)
        val yearlySteps: List<YearlyStepData> =
            stepRepository.getYearlySteps(LocalDate.now().year.toString())
        _stepsByWeek.emit(stepRepository.getStepsByWeek(dailySteps, weeklySteps))
        _stepsByMonth.emit(stepRepository.getStepsByMonth(monthlyStepData,
dailySteps))
        _stepsByYear.emit(
            stepRepository.getStepsByYear(
                yearlySteps,
                (monthlyStepData.sumOf { it.steps } + dailySteps)
            )
        )
    }
}
}
}

```

HomeViewModel.kt

```
package com.app.stepup.view_models
```

```

import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import com.app.stepup.constants.Constants.defaultAchievements
import com.app.stepup.model.StepRepository
import com.app.stepup.model.data.Achievement
import com.app.stepup.model.datastore.StepUpPreferencesRepository
import com.app.stepup.utils.ActivityMetricsCalculator
import dagger.hilt.android.lifecycle.HiltViewModel

```

```

import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.flow.StateFlow
import kotlinx.coroutines.flow.first
import kotlinx.coroutines.flow.map
import kotlinx.coroutines.launch
import javax.inject.Inject

@HiltViewModel
class HomeViewModel @Inject constructor(
    private val stepRepository: StepRepository,
    private val repository: StepUpPreferencesRepository,
    private val calculator: ActivityMetricsCalculator
) : ViewModel() {

    init {
        saveDefaultAchievements()
    }

    private var _calories = MutableStateFlow(0.0)

    private var _distance = MutableStateFlow(0.0)

    val dailySteps = stepRepository.getFlowDailySteps()
    val sumSteps = dailySteps.map { it.sum() }

    val stepGoal = repository.getStepGoal()
    val user = repository.getUserData()
    val achievement = repository.getAchievements()
    val calories: StateFlow<Double> = _calories
    val distance: StateFlow<Double> = _distance
    fun calculateActivityMetrics() {
        viewModelScope.launch {
            _calories.value = calculator.calculateCalories(sumSteps.first())
            _distance.value = calculator.calculateDistanceInKm(
                user.first().height, sumSteps.first()
            )
        }
    }

    private fun saveDefaultAchievements() {

```

```

        viewModelScope.launch(Dispatchers.IO) {
            if (repository.getAchievements().first() == emptyList<Achievement>())
                repository.setAchievements(defaultAchievements)
        }
    }
}

```

MainViewModel.kt

```
package com.app.stepup.view_models
```

```

import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import com.app.stepup.constants.Constants.emptyUser
import com.app.stepup.model.StepRepository
import com.app.stepup.model.data.User
import com.app.stepup.model.datastore.StepUpPreferencesRepository
import com.app.stepup.utils.AchievementsManager
import dagger.hilt.android.lifecycle.HiltViewModel
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.flow.StateFlow
import kotlinx.coroutines.flow.first
import kotlinx.coroutines.launch
import javax.inject.Inject

```

```
@HiltViewModel
```

```

class MainViewModel @Inject constructor(
    private val repository: StepUpPreferencesRepository,
    private val stepRepository: StepRepository,
    private val achievementsManager: AchievementsManager
) : ViewModel() {

```

```

    init {
        checkUserRegistration()
    }

```

```

    private val _isUserRegistered = MutableStateFlow(false)
    val isUserRegistered: StateFlow<Boolean> = _isUserRegistered

```

```

    private fun checkUserRegistration() {
        viewModelScope.launch(Dispatchers.IO) {

```

```

        stepRepository.insertTotalDb()
        repository.getUserData().collect {
            val isRegistered = isUserRegistered(it)
            _isUserRegistered.value = isRegistered
            if (isRegistered) {

achievementsManager.checkAchievements(repository.getAchievements().first())
                }
            }
        }
    }

    private fun isUserRegistered(user: User): Boolean {
        return user != emptyUser
    }
}

```

ProfileViewModel.kt

```
package com.app.stepup.view_models
```

```

import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import com.app.stepup.model.data.User
import com.app.stepup.model.datastore.StepUpPreferencesRepository
import dagger.hilt.android.lifecycle.HiltViewModel
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.launch
import javax.inject.Inject

```

```
@HiltViewModel
```

```

class ProfileViewModel @Inject constructor(
    private val repository: StepUpPreferencesRepository
) : ViewModel() {

```

```

    val user = repository.getUserData()
    val stepGoal = repository.getStepGoal()

```

```

    fun setUserData(user: User, stepGoal: Int) {
        viewModelScope.launch(Dispatchers.IO) {
            repository.setUserData(user)
            repository.setStepGoal(stepGoal)

```

```

        }
    }
}

```

RegistrationViewModel.kt

```
package com.app.stepup.view_models
```

```

import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import com.app.stepup.model.data.User
import com.app.stepup.model.datastore.StepUpPreferencesRepository
import dagger.hilt.android.lifecycle.HiltViewModel
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.launch
import javax.inject.Inject

```

```
@HiltViewModel
```

```

class RegistrationViewModel @Inject constructor(
    private val repository: StepUpPreferencesRepository
) : ViewModel() {
    fun setUserData(user: User, stepGoal: Int) {
        viewModelScope.launch(Dispatchers.IO) {
            repository.setUserData(user)
            repository.setStepGoal(stepGoal)
        }
    }
}

```

MainActivity.kt

```
package com.app.stepup
```

```

import android.content.Intent
import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.activity.enableEdgeToEdge
import androidx.compose.material3.Scaffold
import androidx.core.app.ActivityCompat
import com.app.stepup.navigation.StepUpNavigation
import com.app.stepup.services.StepsService
import com.app.stepup.ui.theme.StepUpTheme

```

```
import dagger.hilt.android.AndroidEntryPoint

@AndroidEntryPoint
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        ActivityCompat.requestPermissions(
            this,
            arrayOf(android.Manifest.permission.ACTIVITY_RECOGNITION),
            1
        )
        startForegroundService(Intent(this, StepsService::class.java))
        setContentView {
            StepUpTheme {
                Scaffold { paddingValues ->
                    StepUpNavigation(paddingValues)
                }
            }
        }
    }
}
```

```
StepUpApplication.kt
package com.app.stepup
```

```
import android.app.Application
import dagger.hilt.android.HiltAndroidApp
```

```
@HiltAndroidApp
class StepUpApplication : Application()
```

```
strings.xml (EN)
```

```
<resources>
    <string name="app_name">StepUp</string>
    <string name="service_count_steps">Let\'s make this day active!</string>
    <string name="hello">Hello, </string>
    <string name="exclamation_mark">!</string>
    <string name="achievements">Daily achievements</string>
    <string name="analytics">Analytics</string>
    <string name="calories_abbreviation" translatable="false">kcal</string>
```

```

<string name="distance_abbreviation" translatable="false">km</string>
<string name="something_went_wrong">Something went wrong</string>

<string name="analytics_week">Analytics for the week</string>
<string name="analytics_month">Analytics for the month</string>
<string name="analytics_year">Analytics for the year</string>
<string name="steps">Steps</string>
<string name="calories">Calories</string>
<string name="distance">Distance</string>
<string name="no_data">No data</string>
<string name="no_data_description">No data has been entered for the specified period,
or you do not have access to it</string>

<string name="enter_name">Enter your name</string>
<string name="gender">Gender:</string>
<string name="age">Age:</string>
<string name="weight">Weight:</string>
<string name="height">Height:</string>
<string name="enter_step_goal">Enter your step goal</string>
<string name="mask_of_entered" translatable="false">XX.X</string>
<string name="registration">Registration</string>
<string name="save_changes">Save changes</string>
<string name="pls_enter_name">Please enter your name</string>
<string name="pls_enter_age">Please enter your age</string>
<string name="pls_enter_height">Please enter your height</string>
<string name="pls_enter_weight">Please enter your weight</string>
<string name="pls_enter_step_goal">Please enter your step goal</string>

<string name="marathoner">Marathoner</string>
<string name="marathoner_descr">42,195 steps completed (the length of a marathon
distance)</string>
<string name="super_walker">Super Walker</string>
<string name="super_walker_descr">100,000 steps completed</string>
<string name="million_steps">Million Steps</string>
<string name="million_steps_descr">Reached 1,000,000 steps in total</string>
<string name="circled_the_earth">Circled the Earth</string>
<string name="circled_the_earth_descr">50,000,000 steps completed (approximately the
length of the equator)</string>
<string name="active_month">Active Month</string>
<string name="active_month_descr">300,000 steps completed in one month</string>
<string name="everest_summit">Everest Summit</string>

```

```

    <string name="everest_summit_descr">58,070 steps completed (approximately the height
of Mount Everest in steps)</string>
    <string name="trip_to_the_moon">Trip to the Moon</string>
    <string name="trip_to_the_moon_descr">478,000,000 steps completed (the distance to the
Moon in steps)</string>
    <string name="billion_steps">Billion Steps</string>
    <string name="billion_steps_descr">1,000,000,000 steps completed in total</string>

    <string name="burned_pizza">Burned Pizza</string>
    <string name="burned_pizza_descr">Burned 300 calories (approximately the number of
calories in one slice of pizza)</string>
    <string name="mini_marathon_of_calories">Mini-Marathon of Calories</string>
    <string name="mini_marathon_of_calories_descr">Burned 1,000 calories in a
week</string>
    <string name="burned_cake">Burned Cake</string>
    <string name="burned_cake_descr">Burned 2,000 calories (the approximate calorie
content of a whole cake)</string>
    <string name="burned_daily_norm">Burned Daily Norm</string>
    <string name="burned_daily_norm_descr">Burned 2,500 calories in one day (recommended
daily intake for an adult)</string>
    <string name="ten_k_calories">10,000 Calories</string>
    <string name="ten_k_calories_descr">A total of 10,000 calories burned</string>
    <string name="burned_month_worth_calories">Burned a Month\'s Worth of
Calories</string>
    <string name="burned_month_worth_calories_descr">Burned 75,000 calories in a
month</string>
    <string name="one_hundred_thousand_calories">100,000 Calories</string>
    <string name="one_hundred_thousand_calories_descr">A total of 100,000 calories
burned</string>
    <string name="burned_big_mac">Burned "Big Mac"</string>
    <string name="burned_big_mac_descr">Burned 550 calories (approximately the calorie
content of a popular burger)</string>
    <string name="million_calories">Million Calories</string>
    <string name="million_calories_descr">Burned 1,000,000 calories over time</string>

    <string name="five_km_on_foot">5 km on Foot</string>
    <string name="five_km_on_foot_descr">Walked 5 kilometers in one day</string>
    <string name="golden_ten">Golden Ten</string>
    <string name="golden_ten_descr">Walked 10 kilometers in one day</string>
    <string name="fifty_km_journey">50 km Journey</string>
    <string name="fifty_km_journey_descr">Total distance walked: 50 kilometers</string>

```

```

<string name="walking_to_work">Walking to Work</string>
<string name="walking_to_work_descr">Walked 100 kilometers</string>
<string name="two_hundred_km_weekend">200 km Weekend</string>
<string name="two_hundred_km_weekend_descr">Walked 200 kilometers in a month</string>
<string name="one_thousand_km_in_year">1,000 km in a Year</string>
<string name="one_thousand_km_in_year_descr">Walked 1,000 kilometers in a
year</string>
<string name="continental_traveler">Continental Traveler</string>
<string name="continental_traveler_descr">Walked 5,000 kilometers</string>
<string name="ultra_hike">Ultra-Hike</string>
<string name="ultra_hike_descr">Total distance walked: 10,000 kilometers</string>
<string name="circled_the_earth_twice">Circled the Earth Twice</string>
<string name="circled_the_earth_twice_descr">Walked 80,000 kilometers (twice the
circumference of the Earth)</string>

<string-array name="days_of_week">
    <item>Mon</item>
    <item>Tue</item>
    <item>Wed</item>
    <item>Thu</item>
    <item>Fri</item>
    <item>Sat</item>
    <item>Sun</item>
</string-array>

<string-array name="months_of_year">
    <item>Ja</item>
    <item>Fb</item>
    <item>Ma</item>
    <item>Ap</item>
    <item>May</item>
    <item>Jun</item>
    <item>Jul</item>
    <item>Au</item>
    <item>Se</item>
    <item>Oc</item>
    <item>No</item>
    <item>De</item>
</string-array>
</resources>

```

strings.xml (UA)

```
<resources>
    <string name="app_name">StepUp</string>
    <string name="service_count_steps">Давай зробимо цей день активним!</string>
    <string name="hello">"Вітаю, "</string>
    <string name="exclamation_mark">!"</string>
    <string name="achievements">Щоденні досягнення</string>
    <string name="analytics">Аналітика</string>
    <string name="analytics_week">Аналітика за тиждень</string>
    <string name="analytics_month">Аналітика за місяць</string>
    <string name="no_data">Данні відсутні</string>
    <string name="no_data_description">За вказаний період дані не внесені, або ви не маєте
до них доступу</string>
    <string name="analytics_year">Аналітика за рік</string>

    <string-array name="days_of_week">
        <item>Пн</item>
        <item>Вт</item>
        <item>Ср</item>
        <item>Чт</item>
        <item>Пт</item>
        <item>Сб</item>
        <item>Нд</item>
    </string-array>

    <string-array name="months_of_year">
        <item>Січ</item>
        <item>Лют</item>
        <item>Бер</item>
        <item>Кві</item>
        <item>Тра</item>
        <item>Чер</item>
        <item>Лип</item>
        <item>Сер</item>
        <item>Вер</item>
        <item>Жов</item>
        <item>Лис</item>
        <item>Гру</item>
    </string-array>
    <string name="enter_name">Вкажіть своє ім'я</string>
    <string name="gender">Стать:</string>
```

```

<string name="age">Вік:</string>
<string name="weight">Вага:</string>
<string name="height">Зріст:</string>
<string name="registration">Зареєструватися</string>
<string name="save_changes">Зберегти зміни</string>
<string name="pls_enter_name">Будь-ласка вкажіть ваше ім'я</string>
<string name="pls_enter_age">Будь-ласка вкажіть ваш вік</string>
<string name="pls_enter_height">Будь-ласка вкажіть ваш зріст</string>
<string name="pls_enter_weight">Будь-ласка вкажіть вашу вагу</string>
<string name="enter_step_goal">Вкажіть бажану к-сть кроків</string>
<string name="pls_enter_step_goal">Будь-ласка вкажіть вашу бажану кількість
кроків</string>
<string name="steps">Кроки</string>
<string name="calories">Калорії</string>
<string name="distance">Дистанція</string>
<string name="marathoner">Марафонець</string>
<string name="marathoner_descr">Пройдено 42 195 кроків (довжина марафонської
дистанції)</string>
<string name="super_walker">Супер-пішохід</string>
<string name="super_walker_descr">Пройдено 100 000 кроків</string>
<string name="million_steps">Мільйон кроків</string>
<string name="million_steps_descr">Досягнуто 1 000 000 кроків за весь час</string>
<string name="circled_the_earth">Окружив Землю</string>
<string name="circled_the_earth_descr">Пройдено 50 000 000 кроків (приблизна довжина
екватора)</string>
<string name="active_month">Активний місяць</string>
<string name="active_month_descr">Пройдено 300 000 кроків за один місяць</string>
<string name="everest_summit">Вершина Евересту</string>
<string name="everest_summit_descr">Пройдено 58 070 кроків (приблизно висота гори в
кроках)</string>
<string name="trip_to_the_moon">Поход на Місяць</string>
<string name="trip_to_the_moon_descr">Пройдено 478 000 000 кроків (відстань до Місяця в
кроках)</string>
<string name="billion_steps">Мільярд кроків</string>
<string name="billion_steps_descr">Пройдено 1 000 000 000 кроків за весь час</string>
<string name="burned_pizza">Спалена піца</string>
<string name="burned_pizza_descr">Спалено 300 калорій (приблизна кількість калорій в
одному шматочку піци)</string>
<string name="mini_marathon_of_calories">Міні-марафон калорій</string>
<string name="mini_marathon_of_calories_descr">Спалено 1,000 калорій за
тиждень</string>

```

```

    <string name="burned_cake">Спалений торт</string>
    <string name="burned_cake_descr">Спалено 2,000 калорій (приблизна калорійність цілого
торта)</string>
    <string name="burned_daily_norm">Спалена денна норма</string>
    <string name="burned_daily_norm_descr">Спалено 2,500 калорій за один день
(рекомендована денна норма для дорослого)</string>
    <string name="ten_k_calories">10,000 калорій</string>
    <string name="ten_k_calories_descr">Всього спалено 10,000 калорій</string>
    <string name="burned_month_worth_calories">Спалено місячну норму калорій</string>
    <string name="burned_month_worth_calories_descr">Спалено 75,000 калорій за
місяць</string>
    <string name="one_hundred_thousand_calories">100,000 калорій</string>
    <string name="one_hundred_thousand_calories_descr">Всього спалено 100,000
калорій</string>
    <string name="burned_big_mac">Спалений \"Біг Мак\"</string>
    <string name="burned_big_mac_descr">Спалено 550 калорій (приблизна калорійність
популярного бургера)</string>
    <string name="million_calories">Мільйон калорій</string>
    <string name="million_calories_descr">Спалено 1,000,000 калорій за весь час</string>
    <string name="five_km_on_foot">5 км Пішки</string>
    <string name="five_km_on_foot_descr">Пройдено 5 кілометрів за один день</string>
    <string name="golden_ten">Золота десятка</string>
    <string name="golden_ten_descr">Пройдено 10 кілометрів за один день</string>
    <string name="fifty_km_journey">50 км Подорож</string>
    <string name="fifty_km_journey_descr">Загалом пройдено 50 кілометрів</string>
    <string name="walking_to_work">Пішки на роботу</string>
    <string name="walking_to_work_descr">Пройдено 100 кілометрів</string>
    <string name="two_hundred_km_weekend">Вихідні на 200 км</string>
    <string name="two_hundred_km_weekend_descr">Пройдено 200 кілометрів за місяць</string>
    <string name="one_thousand_km_in_year">1,000 км за рік</string>
    <string name="one_thousand_km_in_year_descr">Пройдено 1,000 кілометрів за рік</string>
    <string name="continental_traveler">Континентальний мандрівник</string>
    <string name="continental_traveler_descr">"Пройдено 5,000 кілометрів "</string>
    <string name="ultra_hike">Ультра-Похід</string>
    <string name="ultra_hike_descr">Загальна відстань: 10,000 кілометрів</string>
    <string name="circled_the_earth_twice">Окружив Землю двічі</string>
    <string name="circled_the_earth_twice_descr">Пройдено 80,000 кілометрів (два екватори
Землі)</string>
    <string name="something_went_wrong">Щось пішло не так</string>
</resources>

```

ДОДАТОК В
Слайди презентації

Міністерство освіти і науки України
Національний університет “Запорізька політехніка”

кафедра програмних засобів

Дипломна кваліфікаційна робота магістра
на тему

**ДОСЛІДЖЕННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ АНАЛІТИЧНОГО ЗАСТОСУНКУ-
КРОКОМІРА З ВИКОРИСТАННЯМ АКСЕЛЕРОМЕТРА
RESEARCH AND SOFTWARE IMPLEMENTATION OF AN ANALYTICAL PEDOMETER
APPLICATION USING AN ACCELEROMETER**

Виконала:
студентка групи КНТ - 123м

Орина РУДЬ

Керівник:
к.т.н., доцент

Тетяна ФЕДОРОНЧАК

Рисунок В.1 – Титульний слайд

Мета, об’єкт та предмет дослідження

Мета роботи – розробка програмного забезпечення для аналізу фізичної активності користувачів через відстеження кількості кроків за допомогою акселерометра, з метою підвищення точності вимірювань та надання користувачам зручної аналітики для моніторингу їхнього здоров'я і активності.

Об’єкт дослідження – процес збору та аналізу даних про фізичну активність за допомогою акселерометра мобільних пристроїв.

Предмет дослідження – програмне забезпечення для крокоміра, яке використовує акселерометр для відстеження та аналізу кроків користувача.

Рисунок В.2 – Мета, об’єкт і предмет дослідження

Завдання дослідження

- проаналізувати існуючі додатки для моніторингу фізичної активності, зокрема мобільні додатки для підрахунку кроків та аналізу активності;
- розробити методи моніторингу фізичної активності за допомогою акселерометра мобільних пристроїв;
- виконати проєктування програмного забезпечення для підрахунку кроків і аналізу фізичної активності користувача;
- реалізувати мобільний додаток для моніторингу фізичної активності користувачів;
- виконати експериментальне дослідження точності підрахунку кроків та ефективності використання акселерометра для вимірювання фізичної активності.

3

Рисунок В.3 – Завдання дослідження

Аналіз аналогів

Характеристика	Pedometer - Step Counter	Step Counter - Pedometer	Pedometer
Способи аналізу активності	Дані за день, тиждень, місяць	Дані за день, тиждень, місяць	Немає розширеної аналітики
Інтеграція з іншими платформами	Немає	Наявна	Немає
Можливості роботи у фоновому режимі	Низьке споживання батареї	Низьке споживання батареї	Низьке споживання батареї
Реклама	Так	Так	Так
Інтернет для роботи	Не потрібен	Потрібен для синхронізації	Не потрібен
Особливі функції	Немає	Синхронізація з фітнес-застосунками	Немає реєстрації
Оцінка користувачів	4.9/5	4.9/5	4.8/5
Кількість завантажень	50 млн+	10 млн+	50 млн+
Інтерфейс	Простий і інтуїтивно зрозумілий	Мінімалістичний	Простота використання
Точність підрахунку кроків	Хороша, але з похибками	Висока та стабільна	Висока, але з обмеженнями
Функціонал	Підрахунок кроків, калорій, відстані	Підрахунок кроків, аналіз активності	Базовий підрахунок кроків

4

Рисунок В.4 – Аналіз аналогів

Етапи вирішення поставленої задачі

- використання акселерометра для збору даних про рухи користувача;
- розробка сервісу який працює у передньому плані щоб забезпечити безперервну роботу слухання даних з акселерометра;
- зберігання та аналіз даних. Зібрані дані про активність користувача зберігаються в базі даних та кеші застосунку;
- візуалізація результатів. Після збору та обробки даних створюються графіки, які дозволяють користувачеві переглядати інформацію про фізичну активність;
- створення інтерфейсу, який дозволяє зручно переглядати історію активності, звіти та досягнення користувача, що сприяє мотивації до подальшої фізичної активності;
- тестування та оптимізація. Перевірка працездатності застосунку в різних умовах для забезпечення точності підрахунку кроків, а також оптимізація використання ресурсів, таких як батарея, шляхом ефективного управління обробкою даних.

5

Рисунок В.5 – Етапи вирішення поставленої задачі

Алгоритми для обробки даних з акселерометра

У SensorManager використовується алгоритм, який працює на основі детекції піків прискорення (алгоритм детекції піків), що відбуваються під час ходьби користувача. Цей алгоритм вбудований у клас StepDetector, що входить до складу API Android, зокрема, він доступний з API Level 19 (Android KitKat).

StepDetector аналізує дані акселерометра та гіроскопа для визначення піків у векторі прискорення, які відповідають кожному кроку. Якщо виявлено пік, він зараховується як крок. Таким чином, цей алгоритм визначає кроки на основі чітких пікових значень змін прискорення, що відповідають характерним рухам при ходьбі.

Алгоритм	Опис	Переваги	Недоліки
Алгоритм детекції піків	Алгоритм детекції піків Аналізує дані акселерометра для виявлення різких змін у векторі прискорення, які вказують на крок. Всі виявлені піки підраховуються як кроки	Висока точність при правильно налаштованій системі	Може пропустити кроки при нерегулярному русі

6

Рисунок В.6 – Алгоритми для обробки даних з акселерометра

Агрегація даних

Агрегація виконується автоматично при кожному отриманні даних з акселерометра. Якщо під час отримання нового кроку перша записана дата в таблиці DailyStepData відповідає сьогоднішній, оновлюється відповідне поле для поточної години. Якщо ж це не так, перша записана дата вважається попереднім днем. У цьому випадку відбувається підсумовування кроків за попередній день, і ця сума записується в таблицю MonthlyStepData. У таблиці MonthlyStepData перевіряється, чи відповідає перший запис поточному місяцю. Якщо так, то оновлюється відповідне поле для минулого дня. В іншому випадку підсумовуються всі кроки за минулий місяць і ця сума переноситься в таблицю YearlyStepData. Таблиця TotalStepData оновлюється при кожному отриманні даних з акселерометра шляхом додавання нових кроків до загальної суми.

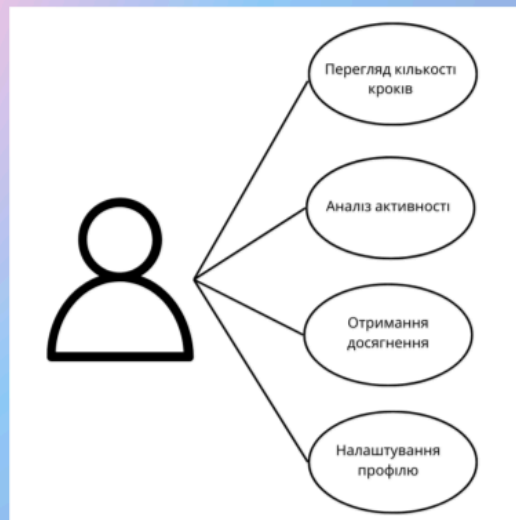
Це забезпечує швидший аналіз даних, так як:

- зменшення обсягу даних;
- швидші запити та аналіз;
- легкість у прогнозуванні та аналізі трендів.

7

Рисунок В.7 – Агрегація даних

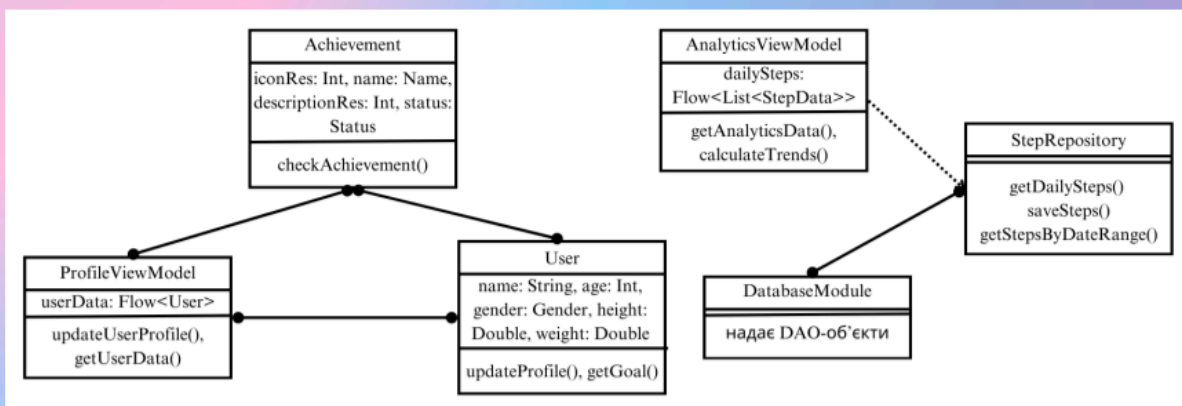
Основні прецеденти користувача в застосунку для підрахунку кроків



8

Рисунок В.8 – Основні прецеденти користувача в застосунку для підрахунку кроків

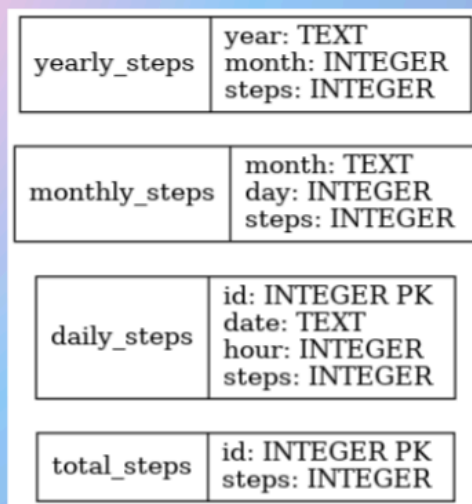
Діаграма класів



9

Рисунок В.9 – Діаграма класів

Схема бази даних



10

Рисунок В.10 – Схема бази даних

Формули для реалізації аналітичних функцій

$$\text{Calories} = \text{steps} \times \text{CALORIES_PER_STEP}$$

де Calories — кількість спалених калорій,
 steps — кількість кроків,
 CALORIES_PER_STEP — коефіцієнт витрати калорій на один крок, що дорівнює 0,04.

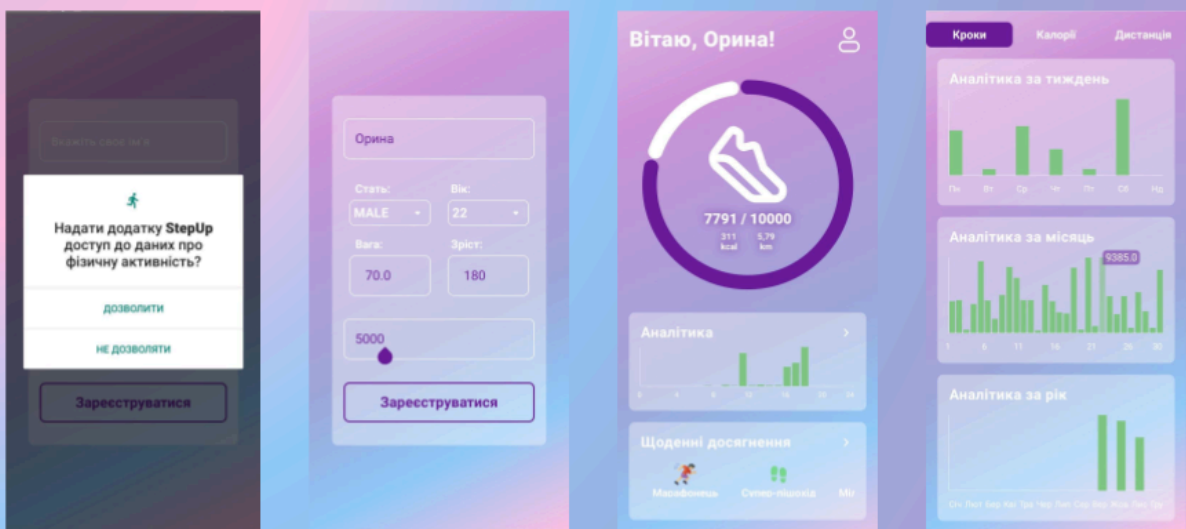
$$\text{Distance} = (\text{steps} \times (\text{height}/100 \times \text{STEP_LENGTH_RATIO})) / 1000$$

де Distance — пройдена відстань у кілометрах,
 steps — кількість кроків,
 height — зріст користувача в сантиметрах,
 STEP_LENGTH_RATIO — коефіцієнт, що визначає довжину кроку як частку від зросту, значення 0,413.

11

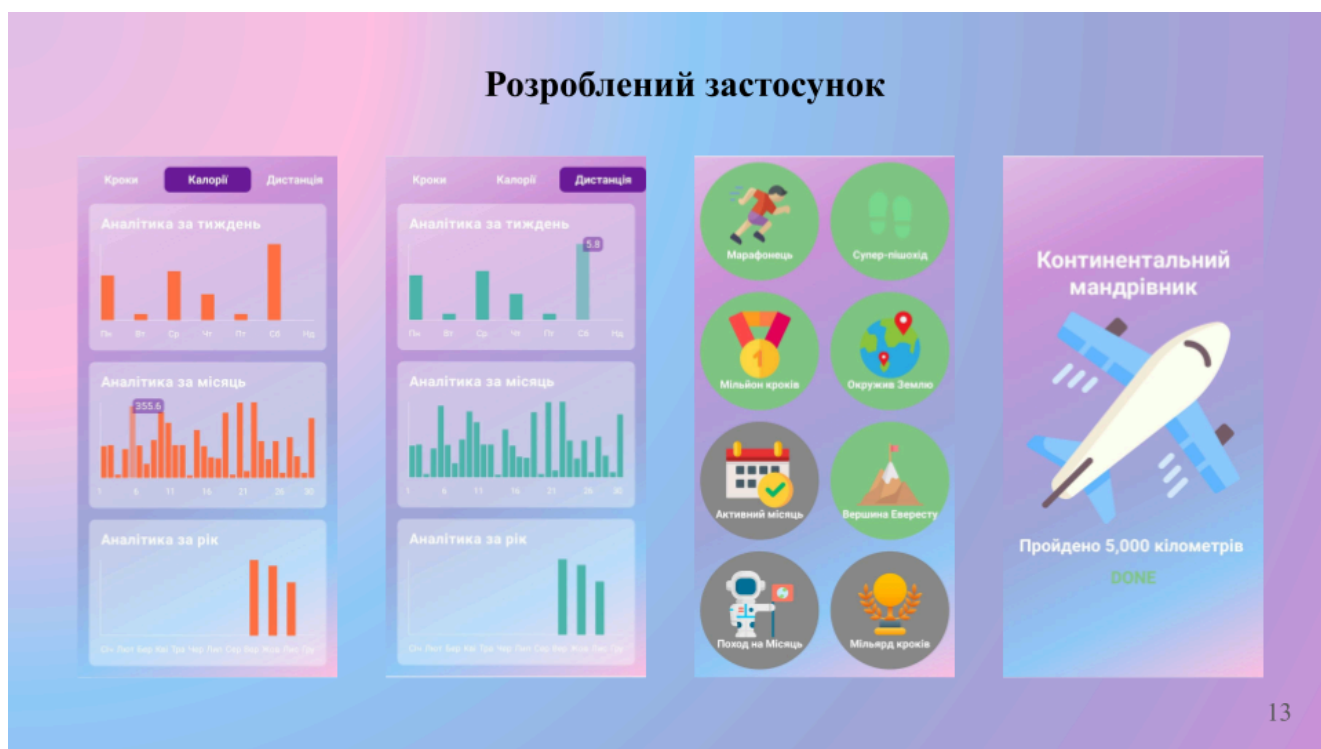
Рисунок В.11 – Формули для реалізації аналітичних функцій

Розроблений застосунок



12

Рисунок В.12 – Розроблений застосунок



13

Рисунок В.13 – Розроблений застосунок

Результати експериментального дослідження

Результати тесту на 1000 кроків

Дані вимірювання	StepUp	Додаток А	Додаток В
Кількість кроків	1000	1018	986

Результати тесту на 5000 кроків

Дані вимірювання	StepUp	Додаток А	Додаток В
Кількість кроків	4994	5035	4936

Результати тесту активності на нерівній поверхні

Дані вимірювання	StepUp	Додаток А	Додаток В
Кількість кроків	1997	2011	1988

14

Рисунок В.14 – Результати експериментального дослідження

Висновки

Процес розробки програмного забезпечення включав кілька ключових етапів: збір, обробка та збереження даних акселерометра, що дозволяє точно реєструвати фізичну активність користувача в реальному часі. Алгоритми обробки даних, такі як фільтрація та виявлення піків, дозволяють чітко визначати важливі рухи та підраховувати кроки навіть за різних умов фізичної активності. Зберігання даних у структурованій базі даних Room дозволяє ефективно організувати доступ до історії активності, що дає користувачам змогу моніторити свій прогрес та отримувати корисну аналітику щодо своїх досягнень.

Особливу увагу в роботі було приділено проектуванню зручного інтерфейсу користувача. Завдяки використанню Jetpack Compose та архітектури MVVM, застосунок має інтуїтивно зрозуміле середовище для перегляду поточної активності, аналізу статистики та досягнень, що дозволяє користувачам активно взаємодіяти з додатком та отримувати мотивацію для досягнення фітнес-цілей. Візуалізація даних через графіки та діаграми надає наочне відображення прогресу в реальному часі, що є важливим для користувачів, які прагнуть досягти кращих результатів у своєму фізичному розвитку.

Тестування точності підрахунку кроків продемонструвало, що розроблене програмне забезпечення має мінімальні відхилення від даних інших популярних додатків, що свідчить про високу надійність та ефективність під час різних фізичних активностей, таких як короткі прогулянки чи більш тривалі тренування. Порівняння з іншими програмами показало, що різниця в підрахунку кроків не перевищує 2%, що є дуже точним результатом для мобільних додатків такого типу. В результаті, розроблене програмне забезпечення для крокометра не тільки демонструє високу точність і ефективність, але й має великий потенціал для подальшого вдосконалення. Існує можливість інтегрувати додаткові функції, такі як штучний інтелект для покращення прогнозування активності та адаптації до індивідуальних особливостей користувача, що дозволить підвищити точність вимірювань і зробити додаток ще більш персоналізованим.

Таким чином, розроблене програмне забезпечення має практичне значення в контексті сфери здоров'я та фітнесу, оскільки допомагає користувачам стежити за рівнем своєї фізичної активності, аналізувати власні досягнення та покращувати здоров'я. Застосунок може бути інтегрований в більш складні системи моніторингу здоров'я, а також служити основою для майбутніх досліджень і розробок у галузі мобільних технологій для покращення фізичної активності та здоров'я користувачів.

Рисунок В.15 – Висновки