МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА

РЕЗНІК РОМАН ЮРІЙОВИЧ

Допускається до захисту:
завідувач кафедри
інформаційних технологій,
к. т. н., доцент
_____ Т. В. Нескородєва
«_____» _____ 2021р.

РОЗРОБКА ФІТНЕС ПОМІЧНИКА ДЛЯ ПЛАТФОРМИ ANDROID

Спеціальність 122 «Комп’ютерні науки»

Кваліфікаційна (бакалаврська) робота

Науковий керівник:
Антонов Ю.С., доцент кафедри
інформаційних технологій,
к. фіз.-мат. н., доцент

Оцінка: _____/_____/_____
(бали за шкалою ЄКТС/за національною шкалою)
Голова ЕК: _____
(підпис)

Вінниця – 2021

АНОТАЦІЯ

Резнік Р.Ю. Розробка фітнес помічника для платформи Android. Спеціальність 122 «Комп'ютерні науки». Донецький національний університет імені Василя Стуса, Вінниця, 2021.

Основною метою виконання кваліфікаційної роботи є створення фітнес помічника.

В ході виконання роботи досліджено предметну галузь та існуючі варіанти фітнес помічників. Показано як вони працюють та які функції виконують. Проаналізовані закономірності в фізичних та біологічних процесах та системах. Встановлено як залежать витрати енергії в залежності від фізичних активностей.

В результаті, використовуючи мову kotlin, було розроблено фітнес помічник, під платформу android, який дозволяє слідкувати та покращувати стан власного здоров’я.

Ключові слова: фітнес помічник, Android, Kotlin, мобільний додаток, розробка додатків, трекінг.

Табл. 6. Рис. 16. Бібліограф.:52 найм.

ABSTRACT

Reznik R.Y. Development of a fitness assistant for the Android platform. Specialty 122 “Computer Science”. Vasyl’ Stus Donetsk National University, Vinnitsa, 2021.

The main purpose of qualifying work is the creation of fitness assistant.

In the course of the work the subject branch and the existing variants of fitness assistants were investigated. Demonstrated how they work and what functions they perform. Regularities in physical and biological processes and systems are analyzed. It is established how energy consumption depends on physical activities.

As a result, using the kotlin language, a fitness assistant was developed, for the android platform, which allows you to monitor and improve your health.

Keywords: fitness assistant, Android, Kotlin, mobile application, application development, tracking.

ЗМІСТ

ВСТУП 4
РОЗДІЛ 1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ 7
1.1 Інформаційні технології на захисті здоров'я 7
1.2 Фітнес помічники 8
1.3 Огляд існуючих рішень 9
1.4 Огляд платформ для розробки 13
1.5 Огляд мов програмування для розробки 18
Висновок до розділу 1 23
РОЗДІЛ 2 ЗАКОНОМІРНОСТІ В ФІЗИЧНИХ ТА БІОЛОГІЧНИХ ПРОЦЕСАХ І СИСТЕМАХ 24
2.1 Особливості витрат калорій під час різних активностей 24
2.2 Добова норма енергії 29
2.3 Постановка задачі 30
2.4 Які дані потрібно зберігати в базі даних 32
2.5 UML діаграми 33
2.5.1. Діаграма станів 33
2.5.2. Діаграма розгортування 34
2.5.3. Діаграма класів 35
Висновок до розділу 2 35
РОЗДІЛ 3 ПРОГРАМНА РЕАЛІЗАЦІЯ ДОДАТКУ 36
3.1 Структура БД 36
3.2 Обґрунтування обраних технологій 38
3.2.1 Обґрунтування вибору платформи розробки 38
3.2.2 Обґрунтування вибору мови програмування для розробки 41
3.3 Перелік необхідних бібліотек для реалізації додатку 44
3.4 Програмна реалізація та аналіз роботи 49
Висновок до розділу 3 58
ВИСНОВКИ 59
ПЕРЕЛІК ВИКОРСТАНИХ ДЖЕРЕЛ 60
ДОДАТКИ 65

ВСТУП

Актуальність роботи

На сьогоднішній день спостерігається позитивна тенденція, все більше і більше людей починають слідкувати за станом свого здоров'я. Це – здорове харчування, дотримання режиму дня, відвідування спортивних центрів, щоденне виконання фізичних вправ тощо.

Також помітно те, що в сучасному ритмі життя, більшість людей не можуть приділити велику кількість часу для відстежування та покращення стану свого здоров'я. Переважна кількість людей вирішує свої справи на ходу, дуже часто через смартфон (оплата комунальних послуг, орієнтування на місцевості, замовлення їжі). Помітивши дану тенденцію, певна частина розробників, почала також працювати в цьому напрямку, в розробці систем, які тим чи іншим чином, допомагають відстежувати та покращувати стан власного здоров'я.

Для того, щоб не відставати від сучасних тенденцій, дуже багато галузей вкладають великі кошти в продукцію, яка сприяє здоровому способу життя. Заклади харчування та служби з доставки їжі, розробляють спеціальні страви та меню, які є корисними для споживачів. Постійно відкриваються фітнес центри та спортивні зали тощо. Не виключенням є і сфера IT. На даний момент реалізовані системи обліку пацієнтів та лікарів, які дозволяють систематизувати роботу лікарні. Розроблені варіанти кишенькових дієтологів, за допомогою яких можливо слідкувати за своїм раціоном, за кількістю спожитих кілокалорій, отримувати рекомендації щодо харчування та формування дієти індивідуально під кожного користувача. Реалізовані трекери, які надають інформацію, щодо кількісті пройденої відстані, кількості спалених кілокалорій під час подолання дистанції, швидкості особи під час руху, часу в якому користувач перебуває в русі, з можливістю перегляду маршрутів на мапі. Існують додатки в яких реалізована функція тренера, наприклад: користувач проходить опитування, після якого отримує рекомендації та комплекс фізичних вправ які допомагають вирішувати проблеми здоров'я користувача. Є варіанти додатків провізорів, які

нагадують про прийом ліків, вказують які препарати потрібно приймати та їх кількість.

Не можливо і не потрібно зупиняти інтеграцію інформаційних технологій в сфери людської діяльності. ІТ в сфері охорони здоров'я – це великий крок до великої мети – зниження рівня захворювань до мінімуму.

Мета і завдання дослідження

Метою даної роботи є розробка фітнес помічника, який поєднає у собі найкращі функції та риси схожих додатків і виконуватиме функцію трекінгу кишенькового дієтолога, та «тренера». Для досягнення поставленої мети слід розв'язати такі задачі:

- проаналізувати процеси, що відбуваються під час повсякденного життя (ходьба, біг, фізичні вправи, прийом їжі тощо) та їх особливості;
- розглянути вже існуючі варіанти фітнес помічників, кишенькових дієтологів та трекерів;
- розглянути платформи та інструменти, які будуть використані при розробці додатку;
- на основі попередніх досліджень визначити технічні вимоги до додатку та реалізувати цей додаток;

Об'єкт дослідження

Закономірності в фізичних та біологічних процесах та системах.

Предмет дослідження дипломної роботи

Процес обміну енергію під час виконання фізичних активностей вправ та харчування.

Методи дослідження

Вивчення та порівняння існуючих варіантів в реалізації подібних додатків, виділення кращих рис, узагальнення зібраної інформації, власна розробка.

Теоретичне та практичне значення одержаних результатів

Полягає у створенні Android додатку – фітнес помічника, який дозволить відслідковувати та покращувати стан власного здоров'я. Додаток розрахований

для масового використання, тобто, будь-яка особа зможе встановити та використовувати додаток для власних потреб.

Апробація результатів дослідження

Доповідь із публікацією тез на тему «Огляд фітнес помічника для платформи Android» у II Всеукраїнській науково-практичній конференції студентів, аспірантів та молодих вчених «Прикладні інформаційні технології»(ПІТ-2021).

Зв'язок роботи з науковими програмами, планами, темами

Наведені у бакалаврській роботі дослідження пов'язані з фундаментальною науково-дослідною роботою «Дослідження та комп'ютерно-математичне моделювання складних систем та процесів у науці, освіті та інформаційно-комунікаційній діяльності підприємств» (№ держреєстрації 0116U002394, 2018-2022 рр.).

Структура роботи

Кваліфікаційна робота складається зі вступу, трьох розділів та висновків до них, списку використаних джерел та додатка.

У першому розділі бакалаврської роботи проведені дослідження предметної галузі, розглянуті існуючі додатки схожі за тематикою (фітнес помічників, кишенькових дієтологів тощо), розглянуті платформ для розробки та мови програмування для відповідних платформ.

У другому розділі розглянуто закономірності в фізичних та біологічних процесах і системах. Визначені функції додатку та те які дані потрібно зберігати в БД, для повноцінної роботи додатку.

У третьому розділі розглянуто програмну реалізацію та проаналізовано роботу android додатку.

Кваліфікаційна робота включає в себе 60 сторінок, 16 рисунків і список літератури з 52 джерел.

РОЗДІЛ 1
ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

Даний розділ присвячений дослідженню предметної області, а саме: як інформаційні технології «зливаються» з сферою охорони здоров'я. Формуванню чітких рамок, що таке фітнес помічник, які функції та можливості повинен виконувати. Огляду готових рішень фітнес помічників, з виділянням їх найкращих якостей. Огляду платформ для розробки та мови програмування для відповідних платформ, виявленню переваг та недоліків кожних з них.

1.1 Інформаційні технології на захисті здоров'я

Сучасний період розвитку людства можна охарактеризувати сильним впливом на нього інформаційних технологій, які плавно вбудовуються в усі сфери людської діяльності, утворюючи всесвітній інформаційний простір. Інформаційні технологій швидко перетворилися на життєво важливий стимул розвитку не тільки світової економіки, а й інших сфер людської діяльності[1]. Прикладом якої є сфера охорони здоров'я. Декілька десятиліть тому було важко уявити те, що бездумна «машина» зможе давати рекомендації щодо лікування, аналізувати стан здоров'я, контролювати фізіологічні показники тощо. Але все змінюється, за останні роки рівень застосування ІТ в медицині – значно зріс.

Збільшується частота застосувань інформаційних технологій при визначені захворювань. Під час використання комп'ютера в лабораторних медичних дослідженнях в програму закладають певний алгоритм діагностики. Створюється база захворювань, де кожному захворюванню відповідають певні симптоми чи синдроми. У процесі тестування, використовуючи алгоритм, людині задаються питання. На підставі її відповідей підбираються симптоми (синдроми), максимально відповідної групи захворювань. Наприкінці тесту видається ця група захворювань з позначенням у відсотках – наскільки це захворювання ймовірно у даного тестування. Чим вище відсотки, тим вища ймовірність цього захворювання. Робляться також спроби створити таку систему (алгоритм), яка б видавала не кілька, а один діагноз. Але все це поки що на стадії

розробки і тестування. Взагалі, на сьогоднішній день у світі створено понад 200 комп'ютерних експертних систем.

Використання нових інформаційних технологій в сучасних медичних центрах дозволить легко вести повний облік всіх наданих послуг, зданих аналізів, виписаних рецептів. Також при автоматизації медичної установи заповнюються електронні амбулаторні карти і історії хвороби, складаються звіти і ведеться повна медична статистика, до того ж, значно зменшується шанс втрати корисної інформації[2].

Ще одна корисна функція – це запис на прийом до лікаря через форму на сайті, яка значно економить час клієнта та полегшує роботу персоналу з систематизації записів клініки. Форма запису на прийом може мати додаткові опції, наприклад, відвідувач може записатися до конкретного лікаря зі списку, дізнатися його графік прийомів, вільні для запису дні або години тощо[3].

Для контролю за власними фізіологічними показниками часто застосовують спеціалізоване програмне забезпечення. Для моніторингу фізичних активностей, щоденного раціону доцільно застосовувати фітнес помічники.

1.2 Фітнес помічники

Кожному хто прагне мати здорове і доглянуте тіло, крім занять спортом, відпочинку та здорового харчування, доводиться також стежити за безліччю взаємопов'язаних даних про свій фізичний стан. Що не так просто в наш, і без того перенавантажений різноплановою інформацією, час. За всім просто не встежиш. Проте, з цим завданням чудово справляється, спеціально розроблений для цих цілей фітнес помічник.

Фітнес помічник – пристрій який працює 24 години на добу і 7 днів на тиждень, аналітик, який невпинно збирає та обробляє інформацію про кількість подоланої відстані, енергетичний обмін, фізичні активності тощо.

Отримані на дисплей смартфона дані дозволяє Вам не тільки в реальному часі відстежувати важливу інформацію про стан свого здоров'я, а й ефективніше планувати свої майбутні тренування.

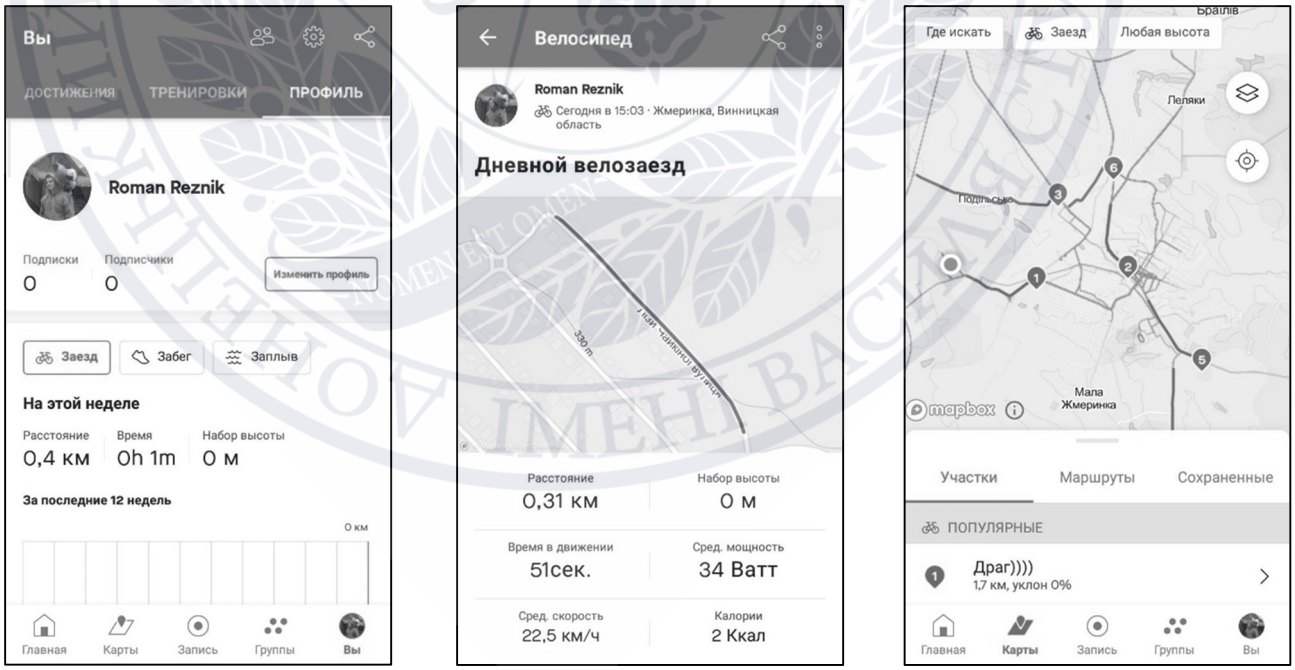
Ще для чого потрібен фітнес помічник, так це для підрахунку витрачених та спожитих калорій. Що також є незамінною допомогою в ретельному плануванні свого раціону.

Всім, хто цікавиться станом свого здоров'я і бажає покращити ефективність своїх поточних тренувань, варто розглянути для себе варіант подібного додатку і переконається в його користі самостійно[4].

1.3 Огляд існуючих рішень

В вільному доступі вже є готові варіанти подібних додатків, але в кожного з них є свої переваги та недоліки. Метою огляду існуючих рішень, є виділення найкращих рис та функцій кожного з них, і на їх основі створення власного, ідеального фітнес помічника.

Яскравим представником додатку схожого плану є додаток Strava (див.рис.1.1) який посідає 3 місце в Play Market серед додатків з розділу «здоров'я та фітнес». Цей додаток є трекером, з його допомогою можна фіксувати свої переміщення, середню швидкість, кількість витрачених калорій та час в дорозі.



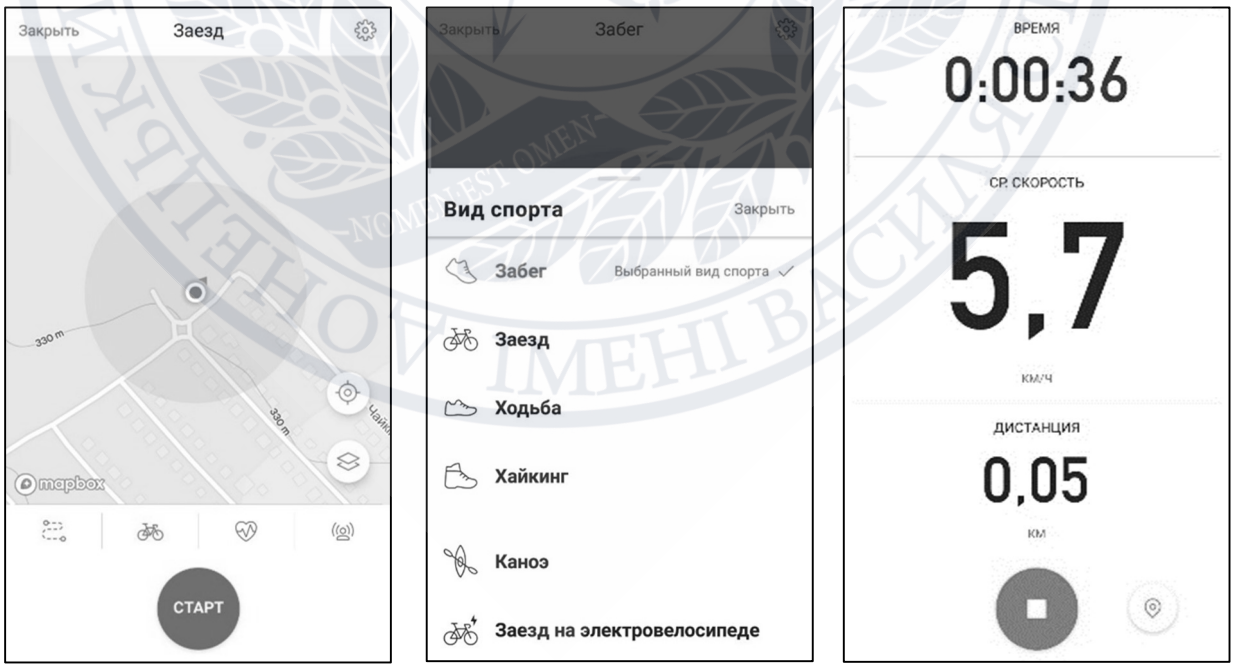
а) б) в)

Рисунок 1.1 – Додаток Strava[5]

Структура додатку дуже зручна. Знизу є меню – BottomNavigation (див рис. 1.1.а та 1.1.в), з асоціативними зображеннями та написами до них. З допомогою яких можна вільно переміщуватись між вікнами додатку. Все досить інтуїтивно. В додатку реалізована соціальна модель. Можливо долучатись до груп та спільнот, спілкуватися в них, домовлятися про спільні тренування й активності.

На (рис.1.1.а) зображено вікно власного профілю додатка Strava, на якому можна переглядати інформацію щодо власного профілю, де за необхідності можна змінити її. Також тут можна переглянути інформацію, про тренування, щодо поточного тижня, за різних видів активностей. Це пройдена відстань, час в дорозі, набір висоти.

На (рис.1.1.б) ілюструється можливість перегляду детальної інформації окремо по кожні активності . В цьому розділі подана інформація про вид активності (біг, ходьба, велозаїзд), час, дату та місця виконання активності. З можливістю перегляду маршруту на мапі, з демонстрацією подоланої відстані, набором висоти, часу в дорозі, середньої потужності в дорозі, середньої швидкості та кількості спалених кілокалорій. За потреби можна додати якусь помітку до даного маршруту, зберегти його як постійний чи видалити.



а) б) в)

Рисунок 1.2 – Функціонал виконання активностей в додатку Strava[5]

За допомогою функціоналу на (рис.1.1.в), можна переглядати та шукати популярні ділянки дороги і маршрути, читати відгуки про ці маршрути, проаналізувавши які, можна обрати найбільш зручний для себе. Також можна переглянути маршрути збережені самим користувачем та опублікувати їх для інших користувачів.

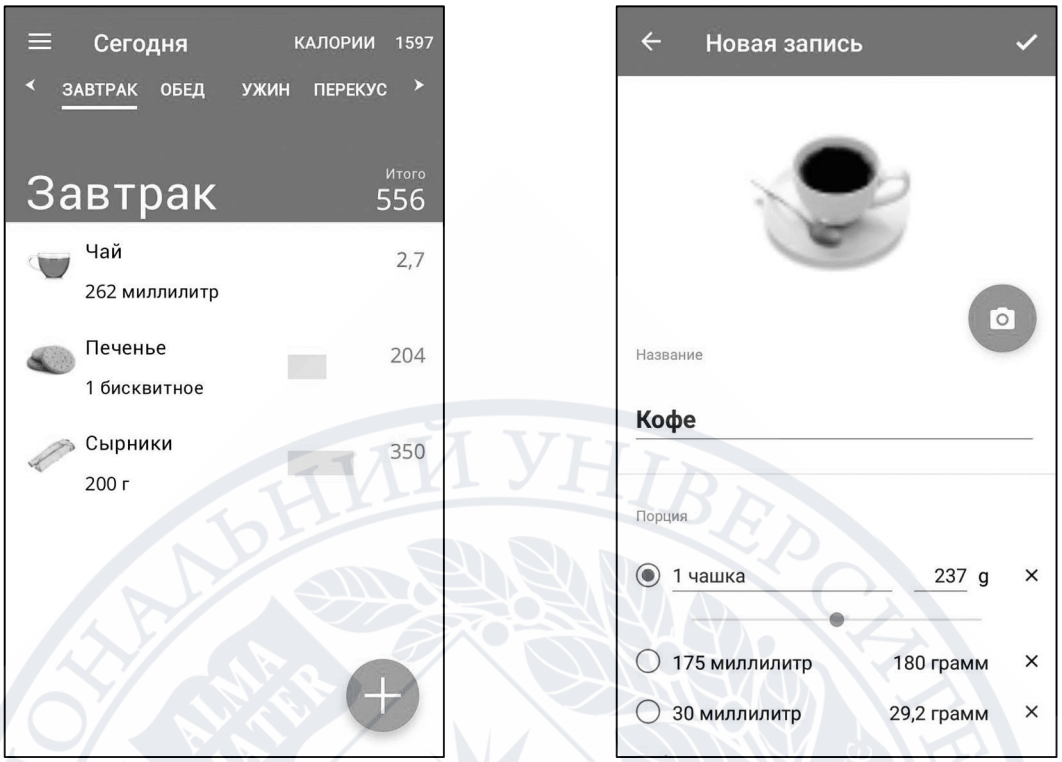
Запис активностей в додатку Strava реалізовано за допомогою функціоналу на (рис 1.2.а). Перед початком слід обрати тип пересування (див. рис. 1.2.б) та за необхідності підключити додаткові пристрої, прикладом яких є пульсометр, трекер, смарт взуття тощо, і натиснути кнопку старт. Після чого додаток набуде вигляду як на (рис.1.2.в). На дисплеї можна переглянути час активності, середню швидкість та подолану дистанцію. Після натискання на кнопку стоп, активність завершиться і всі дані збережуться.

В цілому, додаток Strava орієнтований на трекінг, тренування та спілкування з однодумцями. Але в ньому відсутня функція кишенькового дієтолога, не реалізована функція детального перегляду витрачених калорій та неможливо переглянути статистику за кожен день окремо[5].

Оскільки в попередньому додатку була відсутня функція кишенькового дієтолога, наступним, доречно розглядати додаток спрямований саме в цю сторону. Одним з таких є Щоденник харчування, який має високий рейтинг і входить в топ серед додатків з категорії «здоров’я та фітнес». За допомогою даного додатку можна занотовувати спожиті продукти з вказанням їхньої ваги і отримувати результат в калоріях. Те ж саме можна робити з фізичними вправами.

На (рис 1.3.а) зображено головне вікно додатку. В якому можна перемикатись між прийомами їжі, та занотовувати в кожен з них спожиті продукти. Можна переглянути кількість спожитих калорій окремо по-кожному продуктові, за один прийом їжі, чи загалом за добу.

На (рис.1.3.б) демонструється додавання нового виду продукту. Спершу обирається вид продукту та його кількість, з можливістю вибору з різних одиниць виміру: чашки, скибки, порції, мл., мг., штуки.



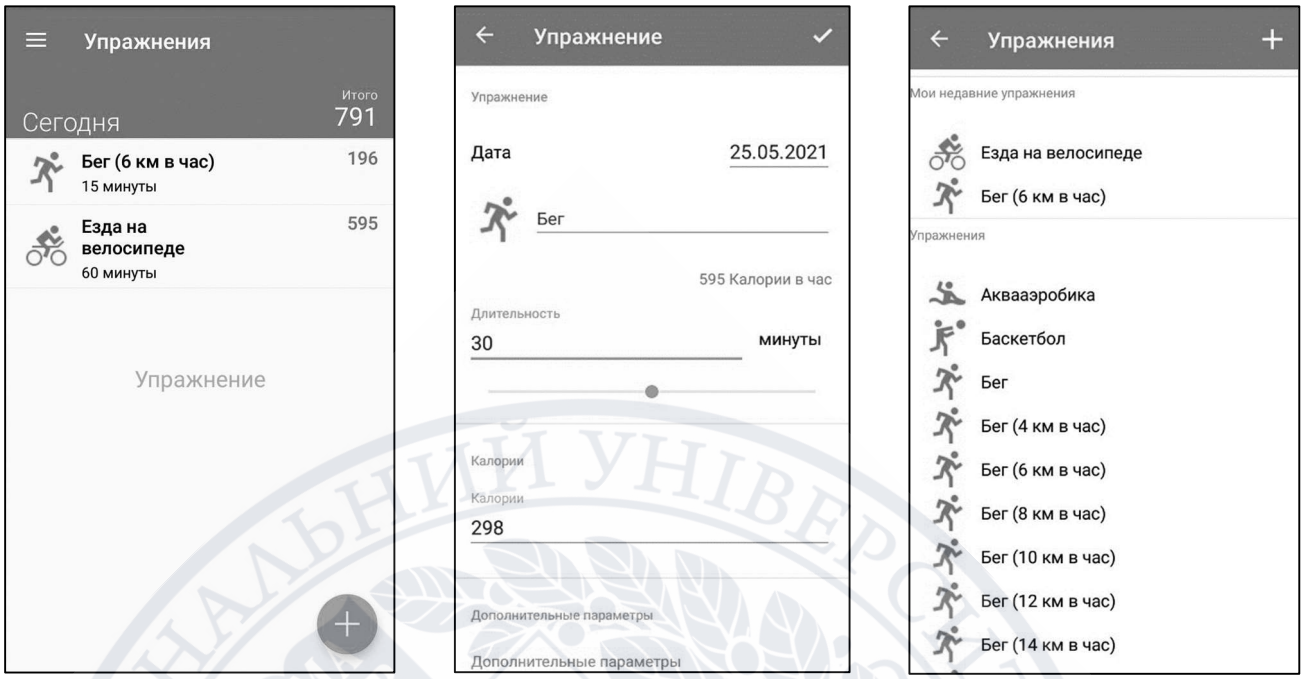
а) б)

Рисунок 1.3 – Функціонал додавання спожитих продуктів в додатку Щоденник харчування[6]

На (рис.1.4.а) демонструються виконані вправи, з додатковою інформацією про кількість витрачених калорій по кожній вправі окремо та за добу. Також тут реалізовано додавання нової вправи.

На (рис 1.4.б) зображено вікно додавання фізичних активностей. В якому користувач може обрати вид активності (див. рис. 1.4.в), яку він виконав з вказанням часових рамок, і додаток сам розрахує затрати калорій. Для зручності, виводиться список нещодавно виконаних вправ, для того щоб при додаванні нової активності не шукати з довгого списку всіх вправ.

В даному додатку зібраний потрібний мінімум, для повноцінної роботи кишенькового дієтолога. Не вистачає лише функцій трекінга та якогось обліку подоланої відстані, для того, щоб казати, що це повноцінний фітнес помічник[6].



а) б) в)

Рисунок 1.4 – Функціонал додавання виконаних активностей в додатку Щоденник харчування[6]

1.4 Огляд платформ для розробки

Перше з чим доводиться стикатися розробнику, це з вибором платформи розробки. Під час реалізації проекту, розробник робить вибір на користь тієї чи іншої платформи, і це дуже важливий крок, наслідки якого можуть бути незворотні. Якщо, звичайно, незабаром він не захоче змінити її. Відтепер розробник стає в прямому сенсі заручником операційної системи, на якій він веде розробку. Тепер йому доступні тільки ті функції та можливості, які написані спеціально під його платформу.

І якщо розробник віддає перевагу якомусь девайсу протягом деякого часу, він стає справжнім адептом тієї чи іншої платформи. Він звикає до інтерфейсу, і потім насилу погоджується змінити операційну систему, адже це принесе йому багато зайвого клопоту. Інтерфейс програми під iOS відрізняється від інтерфейсу під Android чи Windows, і тому дизайнери додатків слідують спеціальним гайдлайнам, які підтримують поведінкові звички користувачів. З цими звичками користувачі розлучаються неохоче[7].

Можна не гадати, а відразу створити додаток під всі платформи світу. Але реалізація програми під конкретну платформу – це окрема розробка, що вимагає праці різних розробників і індивідуального опрацювання дизайну інтерфейсу кожної програми. Тобто, ви замовляєте розробку не одного додатка під декілька платформ, а декілька додатків з різним кодом і дизайном, дуже сильно схожих один на одного. Тому, перш ніж переходити до розробки додатку варто зважити всі «за» і «проти»[8].

На даний момент найпопулярнішими платформами для розробки є Windows, Android та iOS. Кожна з них має свої переваги та недоліки. Для кращої роботи додатку потрібно щоб апаратний засіб, постійно був з користувачем. Тому, як платформа для персональних комп'ютерів – Windows відразу відпадає, адже вона не дасть змогу в повному обсязі використовувати функціонал додатку. З цих причин, подальший розгляд цієї платформи завершено.

Сьогодні вже важко знайти таку людину, яка б не любила оточувати себе «розумною» технікою. Фраза «без телефону – як без рук» звучить все частіше і частіше, а уявити собі життя без мобільних пристроїв стає неможливим. Тому переходимо до розгляду мобільних платформ для розробки, яскравими представниками яких є Android та iOS.

Android – це платформа для мобільних технологій, що забезпечує мобільні телефони, планшети та інші кишенькові пристрої (навіть нетбуки) потужністю і мобільністю операційної системи Linux. Надійністю і мобільністю мови високого рівня і інтерфейсу прикладного програмування, а також великим набором корисних додатків. Додатки для платформи Android написані в основному на мові Java (з використанням таких інструментів, як Eclipse і Android Studio), скомпільовані за допомогою Android API і переведені в байт-код для віртуальної машини Android[9].

Таким чином, платформа Android пов'язана з сімейством операційних систем, призначених для інших проектів сотових телефонів на базі Linux. Мова програмування пов'язує платформу Android з платформою Java ME для старих телефонів від BlackBerry, а також з більш широкою сферою докладання мови

Java і платформи Java Enterprise. Не кажучи вже про те, що всі поточні пристрої BlackBerry можуть запускати додатки для Android, і, по суті, до того, як компанія продала залишки бізнесу, пов'язаного зі смартфонами, останні пристрої BlackBerry могли виконувати лише додатки на платформі Android.

Відповідно, андроїд – це віртуальний робот, який сидить всередині мобільного пристрою, який відповідає за виконання всіх процесів, що відбуваються всередині цього пристрою.

ОС Android за декілька років існування швидко виросла з альтернативної ОС в найпопулярнішу платформу, яка дала змогу суттєво розширити потенціал мобільних пристроїв. В даний час вважається, що платформа Android займає майже три чверті світового ринку смартфонів, хоча вона не витіснила iPad Apple на ринку планшетів. Обсяги продажів змінюються весь час, але ясно, що Android надовго залишиться одним з домінуючих гравців в мобільному просторі.

Система Android також доступна у вигляді декількох спеціалізованих платформ. Платформа Android Wear реалізує модель програмування Android в інтелектуальних наручних годинниках та інших аксесуарах для таких додатків, як браслети для спортивних занять. Платформа Android Auto призначена для управління розважальними пристроями в автомобілях. Платформа Android TV працює в інтелектуальних телевізорах і контролерах менш складних телевізорів. Нарешті, платформа Android Things спроектована для ринку вбудованих систем, який тепер відомий як Інтернет речей (Internet of Things – IoT). Кожна з цих платформ вельми цікава, але, в основі, всі орієнтуються на звичайну платформу Android, призначену для смартфонів і планшетних додатків.

Android – не просто платформа, це серйозний помічник в бізнесі. Безцінним еволюційним етапом стала розробка мобільних додатків під Android для бізнесу, які виводять спілкування з клієнтами на новий рівень. Потужний інструмент, корисний в будь-якій області: від служби таксі до юридичної консультації. Привертає увагу до компанії, не спираючись на традиційну рекламу.

Додатки поширюються вірусним способом, і це вкрай ефективно: не потрібно витрачатись на рекламну кампанію, тільки на створення програмного

продукту для Android. Грамотно розроблений додаток зручно, розважає, наприклад ігровим компонентом, і безвідмовно вирішує завдання, тобто дає швидкий доступ до потрібної інформації[10].

Перевага даної платформи полягає в тому, що андроїд має зручний і дуже зрозумілий інтерфейс, а також гнучку і мультифункціональну систему, яка дозволяє запускати кілька додатків одночасно і експериментувати з налаштуваннями.

Крім усього іншого, численні програми, створені спеціально для платформи андроїд, роблять безмежно щасливими користувачів цієї системи, надаючи безмежні можливості. Адже за допомогою цих додатків можна розплачуватися за покупки, робити фотографії, дивитися фільми або читати книги.

Можна зробити висновок про те, що ця платформа створена для творчих людей, адже два однакових мобільних пристрої можуть виглядати зовсім по-різному.

Андроїд дозволяє завжди мати при собі все – особистого фітнес-тренера, доктора, іграшку або телевізор, роблячи людське життя максимально комфортним.

Операційна система від Apple – прямий конкурент Android. Однак на відміну від Android, iOS заганяє своїх користувачів в дуже жорсткі рамки, вихід за які прирівнюється до смертної кари. Словом, ця операційна система сповідує зовсім інший підхід до роботи з користувачем. Добре це чи погано, з’ясуємо на прикладах?

Основна особливість даної платформи полягає в тому, що вона може бути встановлена тільки на телефонах і планшетах, які випускає Apple. ОС iOS створена на основі концепції прямого маніпулювання за допомогою натискання кнопок прямо на екрані мобільного пристрою.

Те, що iOS, як система, отримала широке поширення не викликає ніякого подиву. З огляду на той факт, що продукція корпорації Apple відрізняється високою якістю своїх розробок, власна платформа дозволяє реалізовувати ряд специфічних завдань, з якими не в змозі впоратися інші системи.

Перед тим, як приступити до огляду платформи iOS, слід детально розібратися, що це за «звір». Як уже згадувалося вище, операційна система iOS – унікальна платформа для зручної взаємодії користувача з мобільним пристроєм компанії Apple. Дана ОС має приголомшливий інтерфейс і дозволяє за допомогою мобільного телефону вирішувати найскладніші завдання. Відмінними рисами платформи iOS є:

Конфіденційність даних – ніяка програма без вашої згоди не отримає доступ до особистих даних. Тільки з вашої згоди сторонні додатки отримають доступ до телефонної книги, вашого місцезнаходження, фото та відео файлів.

Високий рівень безпеки – розробник ОС намагається максимально убезпечити систему від можливого зараження шкідливим програмним забезпеченням.

Величезна кількість вбудованих функцій – в системі iOS включено величезна кількість корисних для користувача функцій, які порадують будь-якого щасливого власника мобільним пристроєм від Apple[11].

Оперативна система iOS – універсальний помічник у навчанні, роботі та повсякденному житті. Завдяки вбудованим функціям мобільний пристрій здатний допомогти розібратися з найскладнішими завданнями. Серед вбудованих функцій слід виділити наступні:

- Touch ID – завдяки даній технології користувач не зможе отримати доступ до чужого мобільного пристрою. Доступ до даних телефону або планшета буде тільки у власника, якого пристрій розпізнає по відбитку пальця.
- VoiceOver – ця функція дозволяє користуватися розробкою Apple людям з поганим зором або взагалі сліпим. В основі цієї технології лежить супровід озвученням усіх дій, які виконує користувач.
- Made for iPhone – за допомогою даної функцією можна поліпшити звук в Bluetooth, який можна використовувати як для розмови, так і для прослуховування музики.

- Гід-доступ – це функція дає можливість відключити ряд програм крім обраних. Вона може бути корисною для батьків, які бажають обмежити доступ дітей до тих чи інших програм пристрою.
- Поліглот – унікальна функція, яка дозволяє користуватися телефоном або планшетом людям, які не знають англійську мову. За допомогою Поліглота можна переключити розкладку клавіатури більш ніж на 50 мов. Крім того, додаток може розпізнавати більше 20 мов «на слух».

Слід додати, що операційна система IOC здатна підтримувати не тільки вбудовані функції, але і безліч сторонніх додатків, які можна завантажити з App Store.

Операційна система Apple iOS має низку особливостей:

Висока швидкість роботи – платформа iOS має високу швидкість роботи. Динаміка використання інтерфейсу здатна здивувати того, хто вперше взяв гаджет від Apple в свої руки.

Інтуїтивно зрозумілий інтерфейс – навіть самий не досвідчений користувач зможе швидко і легко розібратися з усіма особливостями платформи. Простота, зручність і багатофункціональність інтерфейсу перетворюють iOS в одну з найбільш надійних і популярних платформ.

Зручна файлова система – для того, щоб знайти будь-який необхідний Вам файл достатньо здійснити кілька натискань пальцем на екран. Файлова система максимально просто і зрозуміла.

Наявність величезної кількості додатків для ОС – з моменту виходу першої версії платформи і до сьогоднішнього дня було створено безліч спеціальних програм для вирішення різних завдань.

Постійне підвищення функціональності – завдяки регулярному оновленню, функціональність мобільного пристрою постійно підвищується[12].

1.5 Огляд мов програмування для розробки

Від того, яку платформу обрав розробник, залежить те, яку мову він будете використовувати при розробці. Тому, далі розглянуто найпопулярніші мови

програмування для iOS і Android, з виділенням їх переваги та недоліків. Для того, щоб оцінити кожну з них, і вибрати найбільш підходящий варіант.

Але можна схитрувати і створити нативний додаток. Кросплатформені мови програмування, такі як C++, C#, Xamarin, дозволяють один раз створити нативний додаток, надати код в спільне використання та запускати його в iOS, Android і Windows. Як недолік можна виділити те, що програми написані на подібних мовах будуть значно менш швидко працювати і знизять продуктивність роботи з системою, на відміну від програм написаних під певну операційну систему. Також при реалізації нативних додатків проявляються наступні недоліки:

- недостатня гнучкість;
- кожна платформа має унікальний набір функцій і можливостей, які роблять нативні додатки більш скупими на функціонал;
- збільшення часу на розробку;
- збільшення витрат під час розробки.

Тому краще визначитись з однією платформою та працювати з нею, а згодом з отриманням нових ресурсів реалізовувати цей проект на інші платформі, за необхідності.

В першу чергу відповімо на питання про те, які мови програмування для Android додатків підходять найбільше, це:

- Java
- Kotlin
- C#

Java

Часто іменована як офіційна мова операційної системи Android, Java-код на сьогоднішній день вважається одним з найбільш затребуваних серед Android розробників. Майже всі мови мобільного програмування будуються на Java, так як вона володіє найбільш широким функціоналом в мобільному програмуванні.

Як би не хотілося, але якщо прийнято рішення писати код для Android, познайомитися з цією мовою доведеться в будь-якому випадку.

В випадку мобільного програмування дана мова, підтримується фреймворком Android Studio. Який є типовим прикладом середовища розробки. Ознайомитися з роботою якого доведеться в будь-якому випадку, якщо було прийнято рішення програмувати під Android. У цьому вам допоможе набір інструментів Android SDK. Створення програм на цій мові програмування не складе особливих труднощів.

Через те, що Java основа всіх мов мобільного програмування, подальше вивчення нових мов буде простішим, але вона сама, певною мірою складна в освоєнні. Та програми написані на Java дуже громіздкі[13].

C#

Вважається свого роду покращеною версією Java з виправленими недоліками і більш широким і зручним функціоналом.

Дуже добре поєднується з моделюванням в Unity 3D, роблячи ваші можливості в мобільній розробці практично безмежними дозволяючи створювати не тільки програми, але і гри.

Завдяки функції збору сміття (garbage collection) з’являється можливість видалення об’єктів, не використовуваних в додатку.

Для роботи з ним необхідні обов’язкові знання Java[14].

Kotlin

Відмінна мова програмування, яка так само як і C# в поєднанні з Java розкриває свої можливості ще краще. Основною перевагою Kotlin є повна відсутність недоліків що дозволяє створювати програми під Android практично безперешкодно.

Компіляція в байт-код JVM, як і у Java, дає Вам можливість перенести написаний раніше код на Java в Kotlin і прибрати шаблонний код «boilerplate code» що зробить синтаксис більш читабельним та простим.

Офіційно Kotlin був представлений компанією Google в 2017 році, позиціонуючи себе як наступну офіційну мову написання програми під Android після Java. Так як і Java повністю Сумісний з Android Studio.

Навчання Kotlin-у забере не так багато часу як вивчення Java, і буде набагато простішою, але для його повного розуміння знання Java знову таки є обов'язковою[15].

На Android і iOS код пишеться по-різному і відповідно на різних мовах, але різниця насправді не настільки виражається. Давайте розберемо список основних мов програмування на яких пишуть програми для iOS, це:

- Swift
- C #
- Objective-C
- C ++

Кількість розробників, які знають мови програмування iOS додатків значно менше ніж тих хто вміє писати під Android, що робить їх більш затребуваними на ринку праці.

Swift

Swift – це перш за все офіційна мова написання коду для iOS, представлена Apple ще в 2014 році. Запозичивши велику частину з Objective-C і C++, стала неймовірно функціональною та простою для вивчення.

Допомагає уникати помилок в коді ще на етапі розробки додатків завдяки строгі типізації.

Надає дуже зручний функціонал, який робить розробку додатків для iOS простим і захоплюючим процесом, серед якого можна відзначити множинне повернення значень, замикання і дженерики. Залишається найбільш популярною мовою розробки додатків для iOS навіть в 2019 році тому більшість iOS розробників, віддають перевагу саме їй.

Яскравими перевагами Swift є: спрощена навігація, легко читабельний синтаксис, взаємодія з Objective-C, підтримка динамічних бібліотек.

Але без мінусів також не обходиться: часті оновлення вимагають регулярного вивчення нового матеріалу; так званий «Міст», який дозволяє взаємодіяти з Objective-C уповільнює процес складання програми[16].

C#

З його допомогою можна писати не лише для Android, але і для iOS, що дозволяє створювати кросплатформені додатки.

Але для його вивчення необхідно обов'язково знати Java. У зв'язку з тим, що C# є кросплатформеним, додатки на ньому будуть не настільки оптимізованими, як додатки написані на офіційних мовах Apple[14].

Objective-C

Перша мова програмування, яка створена спеціально для пристроїв на базі iOS, таких як iphone, ipod, ipad. Попередник Swift, який до цих пір часто використовується розробниками.

Код додатків написаних на ньому був надбудовою над мовою C, що зобов'язало програмістів, які пишуть на ньому додатки, знати і мову C.

Так само, код на ньому пишеться не тільки для мобільних пристроїв, але і для macOS.

Недоліків дуже багато: застарілий; більш складний в освоєнні ніж Swift, в зв'язку з чим в найближчі кілька років перестане використовуватися; складний синтаксис; низька продуктивність в порівнянні з Swift[17].

C++

C++ також дозволяє програмувати під iOS. Обробка великих обсягів даних, вирішення складних логічних завдань, реалізація та виконання складних алгоритмів. Підтримує об'єктивно-орієнтоване програмування, процедурне програмування і узагальнене.

Однак написати повноцінний додаток на C++ не вдасться, через що потрібно додатково використовувати Objective-C. Також він непростий у вивченні.

З наведених прикладів, стає зрозуміло, що для розробки додатків можна використовувати практично будь-яку мову програмування, кожна з яких має свої

переваги і недоліки. Час витрачений на розробку мобільних додатків на кожній з мов також може коливатися в досить широкому діапазоні[8].

Висновок до розділу 1

В межах даного розділу було детально розглянуто предметну область, а саме. Були розглянуті в загальних рисах фітнес помічники їх властивості та функції. Розглянуто уже існуючі рішення подібних додатків, з виділенням позитивних та негативних рис. Розглянуті платформи для розробки та мови програмування для відповідних платформ.

РОЗДІЛ 2
ЗАКОНОМІРНОСТІ В ФІЗИЧНИХ ТА БІОЛОГІЧНИХ ПРОЦЕСАХ І СИСТЕМАХ

Даний розділ присвячений дослідженню закономірностей в фізичних та біологічних процесах і системах. Огляду особливостей витрат калорій під час різних видів фізичних активностей. Формуванню чіткої постановки задачі, та визначенню функцій додатку. Визначенню тих даних, які необхідно зберігати в базі даних.

2.1 Особливості витрат калорій під час різних активностей

Один і той самий маршрут, кожна людина може долати по-різному. Вона може пройти його в звичайному темпі, пробігти маршрут в середньому темпі, чи взагалі використати велосипед. Зважаючи на ці обставини швидкість руху та час, який затратили на подолання цієї відстані, буде різним. З цього випливає, що і кількість спалених кілокалорій також буде різна. Тому потрібно визначити, як залежить кількість спалених кілокалорій від того яким чином буде подолано маршрут.

Один автор наводить приклад міркувань про те, скільки калорій ми витрачаємо під час фізичних навантажень. Він був впевнений, що при подолані однакової відстані, і ходьба і біг сприяють спалюванню однакової кількості калорій. По одній вагомі причині: оскільки, при подолані маршруту різними способами, вам потрібно перемістити одну і ту ж саму вагу тіла на однакову дистанцію, виходить, що об’єм виконаної роботи однаковий. Тому ходьба повинна сприяти спалюванню приблизно тієї ж кількості калорій, що і при бігу. Але автор продовжує свої міркування і приходить до іншого висновку: коли ти виконуєш циклічну вправу, твій організм спалює 5 калорій за кожен літр спожитого кисню. А при бігу, ти споживаєш на багато більше кисню, ніж при ходьбі. Це вже більше схоже на правду, що при бігу ми спалюємо більшу кількість калорій ніж при ходьбі[18].

Продовжуючи збирати інформацію відносно кількості спалених калорій, було знайдено відповідну статтю. Стаття називалася «Energy Expenditure of

Walking and Running: Comparison with Prediction Equations», опублікована, в журналі «Medicine & Science in Sports & Exercise».

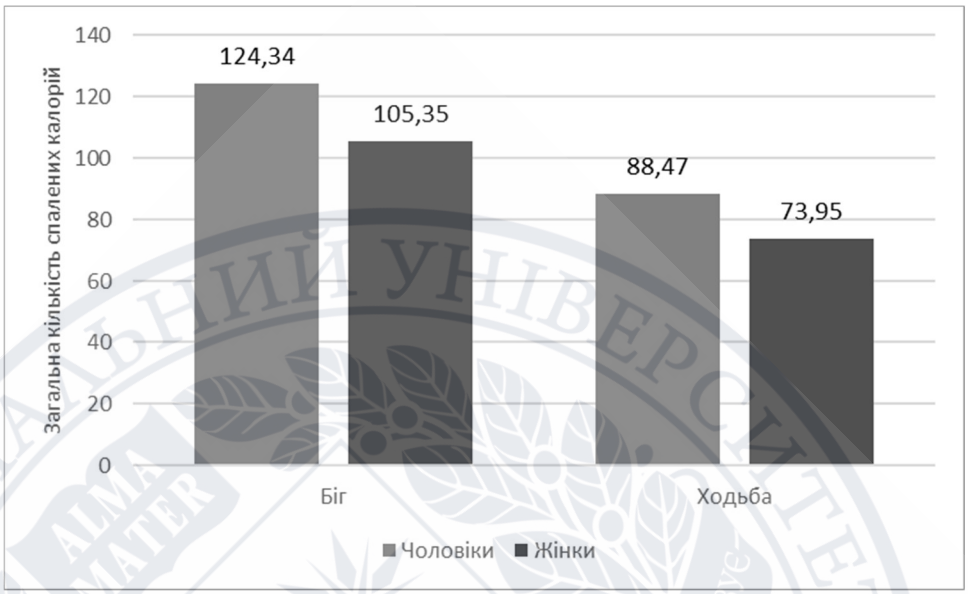


Рисунок 2.1 – Загальна кількість спалених калорій чоловіків та жінок, при ходьбі і бігові на бігові доріжці [19]

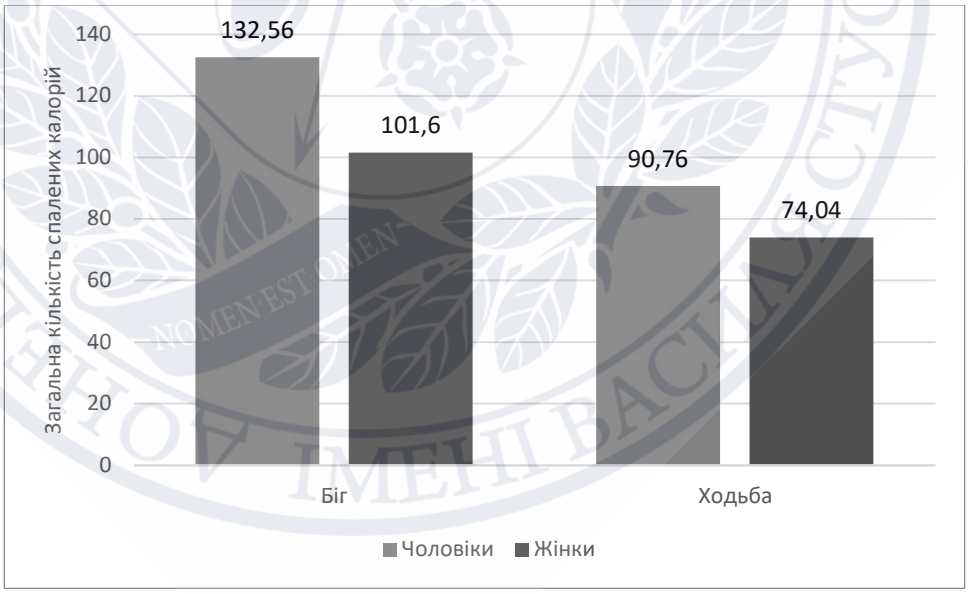


Рисунок 2.2 – Загальна кількість спалених калорій чоловіків та жінок, при ходьбі і бігові на трекові [19]

У статті описується як група університетських вчених досліджували і вимірювали витрату калорій у 12 чоловіків і 12 жінок під час бігу і ходьби.

Досліджувані бігли і йшли 1600 метрів (округлено 1 милю) на біговій доріжці та на трекові. В результаті досліджень з'ясувалось:

- В середньому, на однакові дистанції, чоловіки витрачали 128 калорії при бігу і всього 89 під час ходьби (див.рис.2.1);
- В середньому, на однакові дистанції, жінки витрачали 104 калорії при бігу і всього 74 під час ходьби (див.рис.2.2).

Дослідники пояснили чому так відбувається. Біг і ходьба не зовсім схожі. Коли людина йде – вона в більшості випадків тримає ноги рівними і центр інерції досить плавно рухається вперед по верхній частині ніг. А коли людина біжить – ми стрибаємо з однієї ноги на іншу. Це і є біг (переміщення з однієї ноги на іншу через фазу польоту). Кожен стрибок піднімає центр інерції, коли тіло йде вгору, а далі центр інерції йде вниз, коли ми приземляємося на іншу ногу. Ми згинаємо коліно для поглинання удару об землю. Повторюваний підйом і зниження центру інерції вимагає величезної кількісті ньютонівської сили (закон гравітації), і при відштовхуванні і при приземленні.

Але, ми відводимо себе від істини, коли замість чистого обсягу спалених калорій під час тренування, ми рахуємо загальну кількість спалених калорії. Щоб визначити чисте споживання калорій при будь-якої активності, ми повинні відняти витрачені калорії під час стану спокою або відпочинку. Після віднімання калорій які спалюються в стані спокою виходить, що для того, щоб пробігти одну милю чоловікові необхідно 105 калорій, а пройти – 52 калорії. Жінці 91 калорія, щоб пробігти і 43 калорії, щоб пройти. Отримані дані дозволяють стверджувати, що на однаковому маршруті, кількість спалених калорій під час бігу більша за кількість спалених калорій під час ходьби, біг спалює приблизно на 50% калорій більше, ніж ходьба. Але це справедливо лише для звичайного темпу ходьби[19].

В одній з робіт зазначається, що спортивна ходьба спалює більше калорій, ніж біг при тій же швидкості. Для доведення цієї гіпотези, використали датчик ЧСС, який був одягнутий на тестувальників. Спершу вони бігли по 2 хвилини на швидкостях зазначених у таблиці(1.1).Після 10 хвилин відпочинку, ці ж тестувальників відпочивали, дававши змогу своїй ЧСС повернутися в норму, і

повторювали те ж саме, але з ходьбою. Під час дослідження фіксувалася ЧСС в кінці кожного двохвилинного інтервалу, і середній результат записано таблиці(1.1).

Таблиця 1.1 – Показники ЧСС під час бігу та ходьби на різних швидкостях

Швидкість, км/год	Кількість серцевих скорочень	
	Під час бігу	Під час ходьби
4,8	99	81
5,6	104	85
6,5	109	94
7,2	114	107
8	120	126
8,8	122	145

З таблиці(1.1) виходить, що біг енергозатратніший, ніж ходьба при темпі нижче, ніж 7,6 км/год. На більш високому темпі – ходьба складніше, ніж біг. Пояснення цього процесу таке, що ходьба на високих швидкостях призводить до неприродніх рухів тіла, для яких воно не було створено. Саме це і створює незвичні рухи і неекономічність, які збільшують ЧСС, споживання кисню та власне спалювання калорій[18].

Тому при підрахунку калорій, які витрачаються різними способами, доцільно застосовувати різні формули.

Для підрахунку кількості витраченої енергії під час ходьби застосовується наступна формула:

$$K = (0,035m + (v^2 / h) \times 0,029m) \times t, \qquad (2.1)$$

де K – кількість витраченої енергії, ккал;

m – маса людини, кг;

v – середня швидкість під час руху, м/с;

h – зріст людини, м;

t – тривалість забігу, хв[20].

Для підрахунку кількості витраченої енергії під час бігу застосовується наступна формула:

$$K = m \times S, \tag{2.2}$$

де K – кількість витраченої енергії, ккал;

m – маса людини, кг;

S – подоланий шлях, км[21].

При їзді на велосипеді важко точно визначити кількість спалених калорій. Адже чинників які впливають на цей показник дуже багато. Це вага тіла, швидкість руху, час тренування, властивості рельєфу, тип місцевості, погодні умови (спекотно, холодно, вітряно тощо), тип та вага велосипеда і це лише основні.

Для підрахунку кількості витраченої енергії під час їзди на велосипеді застосовується наступна формула:

$$K = \frac{m \times S \times intensity}{50}, \tag{2.3}$$

де K – кількість витраченої енергії, ккал;

m – маса людини, кг;

S – подоланий шлях, км;

$$intensity(v) = \begin{cases} 15, v \le 15; \\ 19, v \in (15;20]; \\ 20, v \in (20;25]; \\ 22, v > 25. \end{cases}, \tag{2.4}$$

де v – середня швидкість руху, км/год[22].

2.2 Добова норма енергії

Калорійність або енергетична цінність їжі - це одна з найбільш важливих характеристик продуктів харчування. Вона визначається кількістю енергії, яку отримує організм при повному засвоєнні компонентів, що входять в споживану їжу.

Організму людини для нормальної життєдіяльності потрібно енергія. Приблизно 20% енергії використовується для метаболізму в головному мозку. Також багато енергії потрібно для інших обмінних процесів, наприклад, кровообігу і дихання. При знаходженні в теплій навколишньому середовищу організму потрібно менше енергії для підтримки температури тіла, ніж в холоді. Для пересування і активності м'язів витрачається також багато енергії. Простіше кажучи, калорії потрібні людині для того, щоб залишатися в живих.

Кількість енергії, яку організм витрачає протягом доби на підтримку своєї працездатності, і є добова норма калорій. Добова норма калорій людини залежить від віку, ваги, зросту, статі, способу життя, вона визначається енергетичними затратами організму на різні види діяльності. Мінімальна кількість енергії, необхідне для забезпечення нормальної життєдіяльності організму у жінок менше, ніж у чоловіків - приблизно на 15-20 відсотків[23].

Існують формули, що дозволяють порахувати денну норму калорій в залежності від статі, віку, росту і ваги.

У 1919 році з'явилася формула Харріса-Бенедикта, для сучасної людини вона вже не є достатньо точною, має похибку приблизно 5%.

Добова норма калорій для жінок по ній розраховується за формулою (2.5), розрахувати норму калорій для чоловіків можна за формулою (2.6):

$$K = 655{,}1 + 9{,}563 \times m + 1{,}85 \times h - 4{,}676 \times age, \quad (2.5)$$

$$K = 66{,}5 + 13{,}75 \times m + 5{,}003 \times h - 6{,}775 \times age, \quad (2.6)$$

де K – кількість витраченої енергії, ккал;

m – маса людини, кг;

h – зріст людини, см;

age – вік людини[24].

Оскільки формула Харріса-Бенедикта з'явилась давно, вона має певну похибку. Найбільш точною на сьогоднішній день визнана формула Міффліна-Сан Жеора. Для жінок (2.7), а для чоловіків (2.8):

$$K = 10 \times m + 6,25 \times h - 5 \times age - 161, \quad (2.7)$$

$$K = 10 \times m + 6,25 \times h - 5 \times age + 5, \quad (2.8)$$

де K – кількість витраченої енергії, ккал;

m – маса людини, кг;

h – зріст людини, см;

age – вік людини[25].

2.3 Постановка задачі

Перед розробкою власного фітнес помічника потрібно визначити які функції повинен виконувати додаток.

1. Реалізувати реєстрацію та авторизацію облікового запису, за допомогою особистої електронної пошти та персонального паролю.
2. Під час реєстрації розробити функціонал, який дозволить отримати дані для повноцінної роботи додатку, а саме: ім'я, зріст, вагу, стать та вік.
3. Відображати та зберігати маршрут на мапі – для перегляду своїх маршрутів та їх аналізу.
4. Фіксувати тривалість маршруту, адже ця інформація знадобиться в подальшому для визначення середньої швидкості маршруту та визначення кількості витраченої енергії.

5. Фіксувати відстань маршруту, адже ця інформація знадобиться в подальшому для визначення середньої швидкості маршруту та визначення кількості витраченої енергії.

6. Обраховувати середню швидкість під час подолання маршруту, базуючись на тривалості та відстані маршруту. Адже ця інформація знадобиться для визначення кількості витраченої енергії.

7. Фіксувати дату виконання забігу, для того щоб можливо було вести щоденну статистику.

8. Для маршрутів розробити функцію сортування за такими параметрами як: дата, тривалість маршруту, дистанція маршруту, середня швидкість на маршруті, витрачені калорії.

9. Реалізувати функцію видалення маршрутів, за необхідності.

10. Розробити функціонал який дозволить перед початком руху визнати тип пересування (ходьба, біг, заїзд на велосипеді), адже від цього буть залежати обчислення, пов’язані з кількістю витраченої енергії.

11. Реалізувати підрахунок витраченої енергії, в залежності від: часу в дорозі, подоланої відстані, типу пересування, та власних показників (зріст, вага, стать, вік).

12. Реалізувати підрахунок витраченої енергії, під час занять різними видами спорту, в залежності від витраченого часу.

13. Зберігати виконані вправи у вигляді списку, з функцією автоматичного очищення списку після завершення доби.

14. Реалізувати підрахунок спожитих калорій. Шляхом вибору певного виду продукції з вказанням її ваги, після чого кількість отриманих калорій розраховується автоматично.

15. Зберігати спожиті продукти у вигляді списку, з функцією автоматичного очищення списку після завершення доби.

16. Вести детальну статистику за поточну добу. В статистиці відображати:

- кількість витрачених калорій під час занять спортом;

- кількість витрачених калорій під час виконання активностей;
- кількість спожитих калорій;
- кількість витрачених калорій, які потрібно організму, для виконання життєво необхідних функцій;
- загальний обмін енергією, за поточну добу. Тобто, витрачені калорії під час занять спортом, плюс, витрачені калорії під час активностей, плюс, калорії, які потрібні організму, для виконання життєво необхідних функцій і мінус, калорії отримані під час споживання їжі.

17. Вести загальну статистику активностей, за весь час. В статистиці відображати:
    - час в дорозі
    - сумарну подолану дистанцію
    - суму витрачених калорій
    - загальна середня швидкість
18. Відображати стовпцеву діаграму забігів, з відображенням дати, тривалості, середньої швидкості, подоланої дистанції маршруту та кількість витрачених калорій.
19. Додати пункт з налаштуваннями, де можна змінити ім’я, вагу, зріст, вік та стать.

2.4 Які дані потрібно зберігати в базі даних

Перед розробкою потрібно визначити яку інформацію необхідно зберігати в базі даних, для їх подальшого використання та взаємодії з додатком.

1. Маршрут на мапі – для того, щоб в подальшому переглядати свої маршрути на мапі.
2. Дату виконання активності – для того, щоб при формуванні статистики за поточну добу, робити запити і відбирати лише ті активності, дата яких збігається з поточною добою.
3. Відстань, в метрах – для того, щоб визначати середню швидкість під час подолання маршруту та визначити кількість витраченої енергії.

4. Час в дорозі – для того, щоб визначати середню швидкість під час подолання маршруту та визначити кількість витраченої енергії.

5. Середню швидкість – для визначення кількості витраченої енергії.

6. Тип активності – для кращого визначення кількості витраченої енергії.

7. Кількість витрачених калорій – для ведення щоденної та зальної статистики.

8. Кількість кілокалорій для кожного виду продукції, в розрахунку на 100 гр. – для підрахунку спожитої кількості калорій.

9. Кількість кілокалорій для кожного виду спорту, на 1 годину – для підрахунку витраченої кількості калорій.

2.5 UML діаграми

Для кращого розуміння роботи додатку та для простішого його проектування, доцільно спроектувати діаграми, завдяки уніфіковані мові моделювання UML(англ. Unified Modeling Language) — уніфікована мова моделювання, використовується у парадигмі об'єктно-орієнтованого програмування. Є невід'ємною частиною уніфікованого процесу розробки програмного забезпечення. UML є мовою широкого профілю, це відкритий стандарт, що використовує графічні позначення для створення абстрактної моделі системи, яка називається UML-моделлю. UML був створений для визначення, візуалізації, проєктування й документування в основному програмних систем. UML не є мовою програмування, але в засобах виконання UML-моделей як інтерпретованого коду можлива кодогенерація[26].

2.5.1. Діаграма станів

Діаграма станів — діаграма, що визначає зміну станів об'єкту у часі, одна з діаграм моделювання поведінки в UML. Подає об'єкт як автомат з теорії автоматів зі стандартизованими умовними позначеннями.

Елементами діаграми є:

- Коло, що позначає початковий стан.
- Коло з маленьким колом усередині, що позначає кінцевий стан (якщо є).
- Округлений прямокутник, що позначає окремий стан. Верхівка прямокутника містить назву стану, в середині може бути горизонтальна лінія, під якою записуються активності, що відбуваються в даному стані.
- Стрілка, що позначає перехід. Назва події (якщо є), що викликає перехід, відзначається над/під стрілкою. Вартовий вираз може бути доданий перед «/» і укладений у квадратні дужки (назва_події), він означає, що перехід відбувається лише за умови істинності виразу. Якщо при переході відбувається якась активність, то воно додається після «/» (назва події).
- Товста горизонтальна лінія, яка є точкою об'єднання або розгалуження переходів[27].

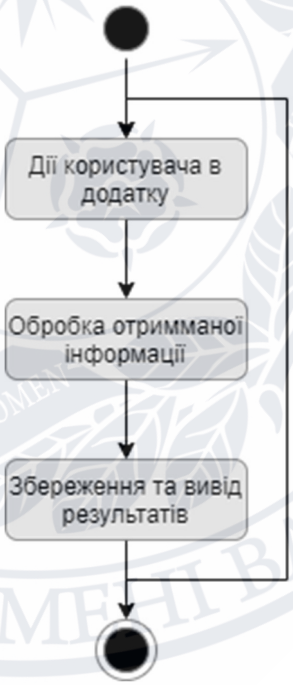


Рисунок 2.3 – Діаграма станів

2.5.2. Діаграма розгортування

Діаграма розгортання — діаграма в UML, на якій відображаються обчислювальні вузли під час роботи програми, компоненти, та об'єкти, що виконуються на цих вузлах. Компоненти відповідають представленню робочих

екземплярів одиниць коду. Компоненти, що не мають представлення під час роботи програми на таких діаграмах не відображаються; натомість, їх можна відобразити на діаграмах компонент. Діаграма розгортання відображає робочі екземпляри компонент, а діаграма компонент, натомість, відображає зв'язки між типами компонент[28].

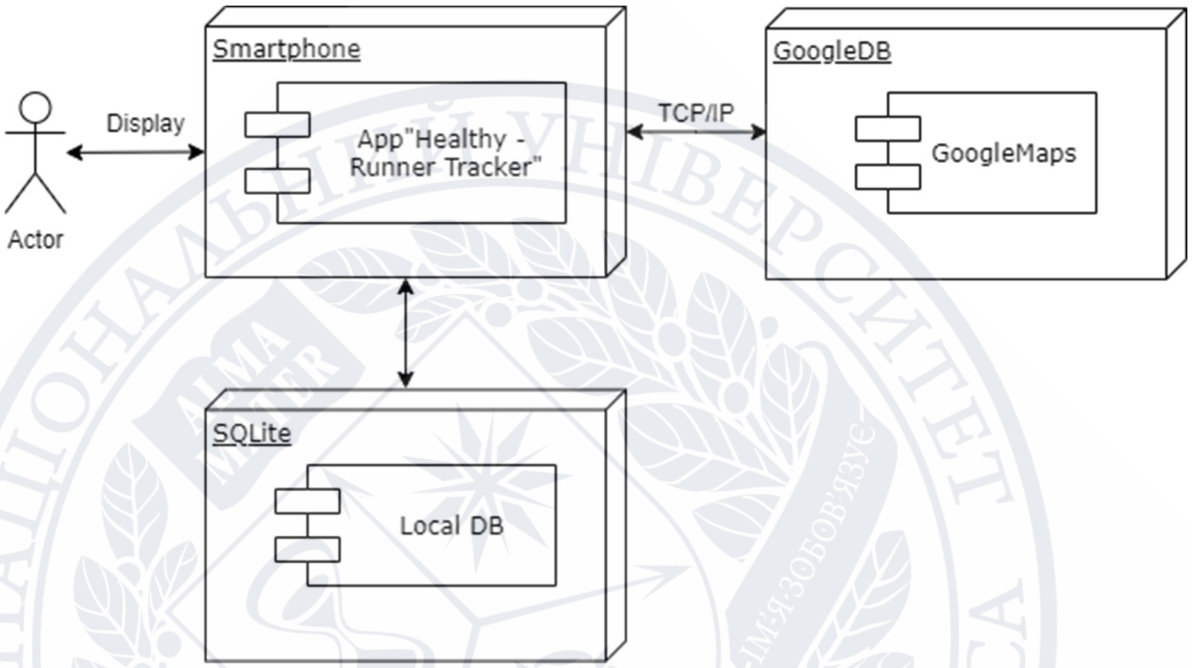


Рисунок 2.4 – Діаграма розгортання

2.5.3. Діаграма класів

Діаграма класів — статичне представлення структури моделі. Відображає статичні (декларативні) елементи, такі як: класи, типи даних, їх зміст та відношення. Діаграма класів може містити позначення для пакетів та може містити позначення для вкладених пакетів. Також, діаграма класів може містити позначення деяких елементів поведінки, однак їх динаміка розкривається в інших типах діаграм. Діаграма класів служить для представлення статичної структури моделі системи в термінології класів об'єктноорієнтованого програмування. На цій діаграмі показують класи, інтерфейси, об'єкти й кооперації, а також їхні відносини[29].

Діаграма класів представлена в (дод. А).

Висновок до розділу 2

В межах даного розділу були розглянуті закономірності в фізичних та біологічних процесах і системах, досліджено як впливає кожен з видів активностей на кількість витраченої енергії. Визначено які функції повинен виконувати додаток та які дані слід зберігати в базі даних, для коректної роботи додатку.

РОЗДІЛ 3
ПРОГРАМНА РЕАЛІЗАЦІЯ ДОДАТКУ

Даний розділ присвячений огляду структури БД, з описом таблиць представлених в ній. Обґрунтуванню обраних технологій. Опису використаних бібліотек та їх функціоналу. Програмні реалізації та аналізу роботи додатку.

3.1 Структура БД

Дана база даних спроектована на основі реляційної моделі. Виконано аналіз сутностей, в ході реалізації функцій фітнес помічника. Проведена нормалізація таблиць даних та визначені зовнішні і первинні ключі. На (рис. 3.1) відображена схема бази даних фітнес помічника для платформи Android.

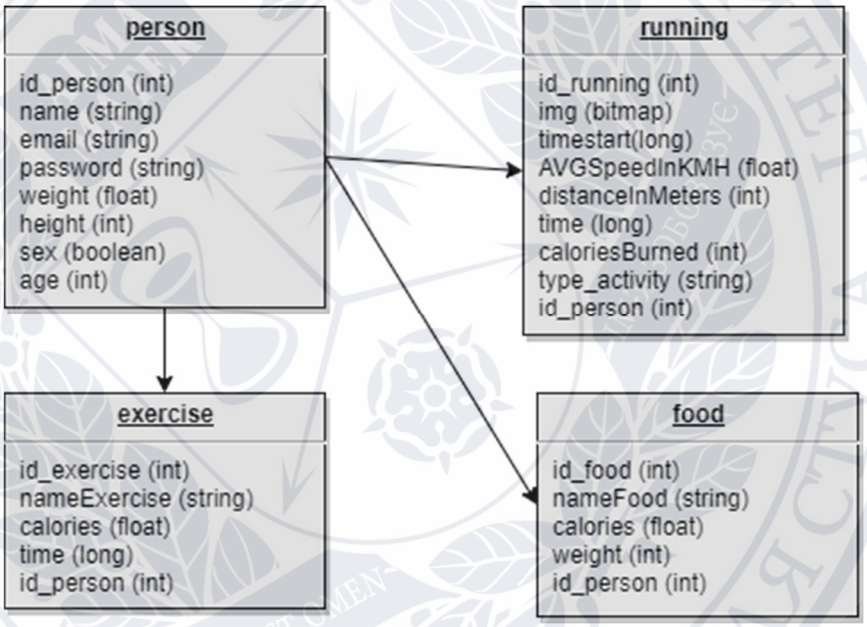


Рисунок 3.1 – Схема бази даних

Опис таблиць бази даних, які представлених на схемі(див. рис. 3.1):

Таблиця 3.1 – person

Назва	Тип	Опис
id_person	int	Ідентифікатор
name	string	Ім'я користувача
email	string	Пошта для авторизації та реєстрації
password	string	Пароль для авторизації та реєстрації

Продовження таблиці 3.1

weight	float	Вага користувача, потрібна для обчислень пов'язаних з енергообміном
height	int	Зріст користувача, потрібен для обчислень пов'язаних з енергообміном
sex	boolean	Стать користувача, потрібна для обчислень пов'язаних з енергообміном
ade	int	Вік користувача, потрібен для обчислень пов'язаних з енергообміном

Таблиця 3.2 -running

Назва	Тип	Опис
id_running	int	Ідентифікатор
img	bitmap	Зображення маршруту на мапі
timestart	long	Час початку забігу
AVGSpeedKMH	float	Середня швидкість в км/год
distanceInMeters	int	Відстань маршруту
time	long	Затрати часу
caloriesBurned	int	Кількість спалених ккалорій
type_activity	string	Вид активності
id_person	int	Ідентифікатор користувача

Таблиця 3.3 – exercise

Назва	Тип	Опис
id_exercise	int	Ідентифікатор
nameExercise	string	Назва вправи
calories	float	Скільки калорій витрачається, ккал/год
time	long	Час виконання вправи
ip_person	int	Ідентифікатор користувача

Таблиця 3.4 – food

Назва	Тип	Опис
id_food	int	Ідентифікатор
nameFood	string	Назва продукту
calories	float	Скільки калорій набирається при споживані продуктів, ккал/100гр
weight	int	Вага продукту
ip_person	int	Ідентифікатор користувача

3.2 Обґрунтування обраних технологій

3.2.1 Обґрунтування вибору платформи розробки

Операційна система Android – ця назва сьогодні відомо практично кожному жителю планети. Під управлінням даної ОС працюють смартфони і планшети. Вона встановлюється на нетбуки і широко просувається на ринку розумних телевізорів і HD плеєрів. Андроїд – це найпоширеніша операційна система для мобільних і мережевих пристроїв. На сьогодні девайси, що використовують Android, займають приблизно 80% мобільного ринку. Це означає, що переважна більшість всіх людей на планеті, які мають смартфони, є «адептами зеленої платформи». Географічно Android переважає країнах СНД, Східної і деяких країнах Західної Європи, а також в Китаї, і інших країнах сходу і Африки. А на iOS припадає приблизно 15% частки мобільного ринку. Більшість з цих 15% живуть в США, Канаді, Австралії, Великобританії, Франції та скандинавських країнах[7].

Андроїд відрізняється від інших операційних систем підходом до універсалізації, використання єдиних схем управління компонентами апаратної платформи і взаємодії з кінцевим користувачем. Операційна система має такі переваги:

Реалізована схема синхронізації. Маючи кілька пристроїв на Android, можна мати на них ідентичний набір програм, контактів, записів календаря. При цьому користувачеві не потрібно докладати ніяких зусиль для цього.

Це відкрита платформа. Можна встановлювати як програми з магазину, так і написані сторонніми розробниками. Сьогодні програму для Android можна написати самостійно.

Немає єдиного магазину додатків. Цим не можуть похвалитися інші операційні системи. Крім Google Play, користувачі можуть користуватися депозитарієм UpToDown і іншими сторонніми сервісами.

На Android сьогодні можна вибрати телефон з дійсно гігантського списку пропозицій. Дана ОС є відкритою і безкоштовною, тому конкуренція на ринку виробників мобільних пристроїв йде, що називається, до смерті. На Android доступні як смартфони бюджетного рівня, так і продукти елітного класу.

А операційна система iOS виключно для iPhone, iPad і носимих пристроїв фірми Apple[30].

Так само, як і для створення картинок, відеороликів або будь-яких інших продуктів, для створення додатків необхідна власне інтегроване середовище розробки (IDE). Близько 10 років тому найпопулярнішим середовищем розробки був Eclipse, але зараз IDE Android Studio повністю витіснила Eclipse з ринку мобільної розробки[31].

Android Studio — інтегроване середовище розробки (IDE) для платформи Android, написане на мові програмування Java. Середовище розробки адаптоване для виконання типових завдань, що вирішуються в процесі розробки застосунків для платформи Android[32].

Android Studio надає ряд важливих переваг, в порівняні з конкурентами, до яких можна віднести:

- Під час створення додатків і утиліт для операційної системи Android, користувач програмного забезпечення може спостерігати за змінами в проекті, в режимі реального часу.
- Дозволяє оптимізувати роботу майбутніх додатків для роботи не тільки на смартфонах, але і на планшетах, портативних ПК, які працюють на основі даної операційної системи.

- У Android Studio вбудований емулятор, що дозволяє перевірити коректну роботу програми на пристроях з різними екранами, з різними співвідношеннями сторін.
- Позитивна відмінність емулятора - перегляд приблизних показників продуктивності при запуску програми на найпопулярніших пристроях.
- Середовище розробки для додатків Android Studio є зручним навіть для початківців розробників. У програмі реалізовані всі сучасні засоби для упаковки коду, його маркування. Наявна функція Drag-n-Drop, що полегшує перенесення компонентів в середовище розробки.
- Локалізація додатків стає значно простішою з функцією SDK, яка також входить до переліку переваг Android Studio.
- Середовище розробки підтримує роботу з декількома мовами програмування, до яких відносяться найпопулярніші - C / C ++, Java, Kotlin.
- Редактор коду, з яким зручно працювати.
- Тестування коректності роботи нових ігор, утиліт, їх продуктивності на тій чи іншій системі, відбувається безпосередньо в емуляторі.
- Рефакторинг вже готового коду.
- Досить велика кількість бібліотек з готовими шаблонами і компонентами для розробки ПЗ.
- Розробка програми для Android будь-яких версій операційної системи.
- Попередня перевірка вже створеного додатка на предмет помилок в ньому.
- Великий набір засобів інструментів для тестування кожного елемента програми, гри.
- Для недосвідчених / початківців розробників спеціально створено посібник з використання Android Studio, розміщений на офіційному сайті утиліти[33].

Android Studio дійсно створює приємне перше враження. Цю IDE оцінить як досвідчений розробник, так і початківець, який тільки освоює ази. Простота, в поєднанні з широкими можливостями роблять свою справу. Багатий набір

інструментів, гнучкість в розробці, можливості тестування, підтримка декількох мов програмування і вбудований емулятор роблять утиліту однією з кращих в своїй ніші[34].

3.2.2 Обґрунтування вибору мови програмування для розробки

Оскільки платформа під яку ведеться розробка Android, а IDE Android Studio, то мову програмування потрібно обирати відповідну. Обирати доводиться з Java та Kotlin.

І Java і Kotlin – статично типізована мова програмування. Вони підтримує як об'єктно-орієнтоване, так і процедурне програмування. Основний код Kotlin та Java програм пишеться в функції main, якій передається масив аргументів командного рядка[35].

Kotlin і Java - повністю інтероперабельні мови, тому одна і та ж програма може бути частково написана на Kotlin, частково на Java. Програми на Kotlin можуть використовувати всі наявні Java-бібліотеки, і навпаки[36].

Загалом Kotlin і Java дуже схожі, але Kotlin надає ряд важливих переваг:

Гнучкість і простота синтаксису. Прості функції і структури можна оголосити одним рядком. Гетери і сетери задаються за межами для інтероперабельності з Java-кодом. Додавання data-анотації до класу активує автоматичну генерацію різних шаблонів[37].

Null-безпека

Як вже говорилося раніше, Kotlin не допускає виникнення NullPointerException, видаючи помилку компіляції.

Класи даних (Data Classes)

У Kotlin з'явилися спеціальні класи, призначені спеціально для зберігання даних. Вони генерують різні шаблони: equals (), hashCode (), toString (), гетери і сетери і т.д. Для порівняння код на Java:

```
1  class Book {
2  private String title;
3  private Author author;
4  public String getTitle() {
```

```
5   return title;
6   }
7   public void setTitle(String title) {
8   this.title = title;
9   }
10  public Author getAuthor() {
11  return author;
12  }
13  public void setAuthor(Author author) {
14  this.author = author;
15  }
16  }
```

І на Kotlin:

```
1   data    class    Book(var    title:String,var
author:Author)
```

Також легко створювати копії класів даних за допомогою методу copy ():

```
1   val book = Book("Kotlin", "JetBrains")
2   val copy = book.copy()
```

[38]

Функції-розширення

Kotlin дозволяє розширювати функціональність існуючих класів, не вдаючись до наслідування. Це робиться за допомогою функцій-розширень. Для оголошення такої функції до її імені потрібно приписати префікс у вигляді розширюваного типу. Ось так можна додати функцію swap в MutableList:

```
1   fun MutableList<Int>.swap(index1:Int,index2:Int){
2   val tmp=this[index1]
3   this[index1]=this[index2]
4   this[index2]=tmp
5   }
```

Ключове слово this всередині функції-розширення відноситься до об'єкту-одержувачу, який передається перед крапкою. Тепер можна застосувати функцію swap до будь-якого змінюваному списку:

```
1   val abc = mutableListOf (1, 2, 3)
2   abc.swap (0, 2)
```
[39]

Розумні приведення типів

Компілятор Kotlin дуже розумний, коли мова заходить про приведення типів. У більшості випадків не потрібно явно вказувати оператори приведення, оскільки в мові є оператор is, який робить за вас всю роботу:

```
1   fun demo (x: Any) {
2   if (x is String) {
i.print (x.length) // x автоматично приводиться до типу
String
3   }
4   }
```

Типізація

У Kotlin необов'язково явно вказувати тип змінної:

```
1   / * Неявне визначення * /
2   fun main (args: Array <String>) {
3   val text = 10
4   println (text)
5   }
6   / * Явне визначення * /
7   fun main (args: Array <String>) {
8   val text: Int = 10
9   println (text)
10  }
```
[40]

Функціональне програмування

Kotlin заточений під функціональне програмування. Він надає велику кількість корисних можливостей, наприклад, функції вищого порядку, лямбда-

вирази, перевантаження операторів і ледачі обчислення логічних виразів. Ось приклад роботи з колекціями:

```
1   fun main (args: Array <String>) {
2   val numbers = arrayListOf (15, -5, 11, -39)
3   val nonNegativeNumbers = numbers.filter {it> = 0}
4   println (nonNegativeNumbers)
5   }
6   // Результат: 15, 11
```

Функції вищого порядку - це функції, які приймають інші функції в якості аргументів і повертають функції. Приклад:

```
1   fun alphaNum (func: () -> Unit) {}
```

У ньому func - це ім'я аргументу, а () -> Unit - це тип функції. Розуміється, що func буде функцією, що не приймає аргументів і нічого не повертає.

Лямбда-вирази, або анонімні функції - це функції, які не оголошуються, а передаються у вигляді виразів. Приклад:

```
1   val sum: (Int, Int) -> Int = {x, y -> x + y}
```

Оголошується змінна sum, яка бере два числа, складає їх і приймає значення суми, приведене до цілого. Для виклику досить простого sum (2,2)[41].

Порівняння швидкості Java і Kotlin

Перша збірка Kotlin-коду займає приблизно на 15-20% більше часу, ніж аналогічний процес на Java. Однак інкрементна збірка Kotlin навіть трохи швидше, ніж у Java. Таким чином, мови приблизно рівні за швидкістю компіляції.

Kotlin - це наступний етап розвитку Java, з якої він повністю сумісний. Це робить його відмінним інструментом для мобільних додатків[42].

3.3 Перелік необхідних бібліотек для реалізації додатку

Сучасні програми занадто складні, щоб створювати їх з нуля, тому розробники використовують бібліотеки – так код пишеться набагато швидше.

Не потрібно винаходити велосипед кожен раз, коли потрібно їхати в магазин, набагато простіше взяти готовий і їхати на ньому. Те ж саме і в

програмуванні. Програміст повинен думати про реалізацію логіки, а не про те, як працює обладнання або за яким протоколом передаються дані.

Кожна бібліотека надає можливості для вирішення якихось конкретних завдань:

- виконання математичних операцій;
- роботи з графікою;
- роботи з файлами;
- роботи з мережею;
- шифрування і так далі.

Ці бібліотеки можуть бути вбудовані в мову або додаватися окремо.

В випадку розробки фітнес помічника, потрібно застосовувати бібліотеки для роботи з графікою, для роботи з базами даних, для роботи з користувацьким інтерфейсом, для роботи з мапами та декілька сторонніх бібліотек. А саме:

Android-lifecycle

Використовується для контролю життєвих циклів Android додатку (див. рис. 3.2). Ці цикли жорстко контролюється системою і залежить від потреб користувача, доступних ресурсів тощо. Наприклад, ми запускаємо фітнес помічник. Рішення про запуск програми приймає система. Хоча останнє слово і залишається за системою, вона підпорядковується певним заданим і логічним правилам, що дозволяє визначити, чи можна завантажити, призупинити додаток або припинити його роботу. Якщо в даний момент користувач працює з певним вікном, система дає пріоритет відповідного додатку. І навпаки, якщо вікно невидимо і система вирішує, що роботу програми необхідно зупинити, щоб мобілізувати додаткові ресурси, буде припинено роботу програми, що має більш низький пріоритет. В Android ресурси більш обмежені, тому Android більш жорстко контролює роботу додатків.

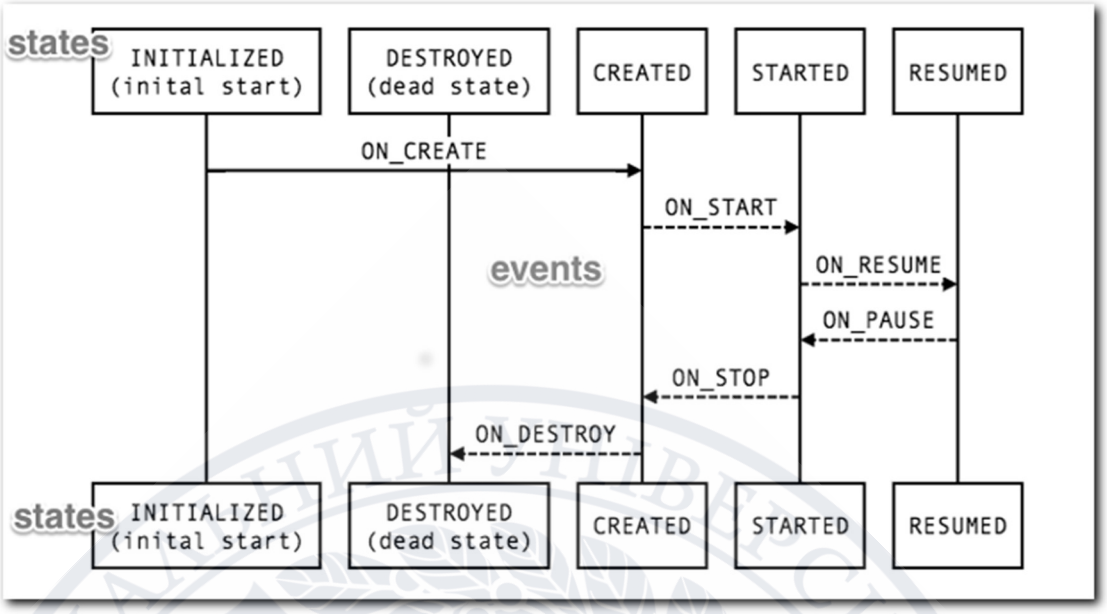


Рисунок 3.2 – Життєвий цикл додатка на Android[43]

Основні методи життєвого циклу додатків:

- onCreate() – цей метод викликається при створенні або перезапуску додатку
- onStart() – викликається безпосередньо перед тим, як додаток стає видимим для користувача та супроводжується викликом методу onResume()
- onRestart() – викликається в випадку, якщо вікно повертається в пріоритетний режим після виклику onStop(). Тобто викликається після того, як активність була зупинена і знову була запущена користувачем.
- onResume() – викликається після onStart (), навіть коли вікно працює в пріоритетному режимі і користувач може його спостерігати. У цей момент користувач взаємодіє зі створеним вікном.
- onPause() – коли користувач вирішує перейти до роботи з новим додатком, система викличе метод onPause () для переривання вікна. По суті відбувається згортання активності, зі збереженням незафіксованих даних. Також звільняються ресурси та зупиняється відтворення відео, аудіо та анімацій.

- onStop() – викликається, коли вікно стає невидимим для користувача. Це може статися при знищенні додатку, або якщо була запущений інший додаток, що перекрив вікно поточної додатку.
- onDestroy() – метод викликається після закінчення роботи додатку, при виклику методу finish() або в разі, коли система сама знищує цей екземпляр активності для звільнення ресурсів[44].

Material-components

Бібліотека для роботи з стилями та темами. Для роботи з текстовими полями формами кнопками, та їх дизайном. За допомогою цієї бібліотеки були розроблені 2 стилі: звичайний та dark mold – для темної пори доби. Було використано компоненти:

- Bottom navigation – для реалізації навігації в додатку;
- Buttons – для створення кнопок;
- Rabioboxes – радіокнопки для вибору одного варіанту;
- Dialogs – для створення спливаючих діалогових вікон;
- Tabs – для упорядкування вмісту на різних екранах, наборів даних та інших взаємодіях;
- Text fields – для створення текстових полів[45].

Glidle

Бібліотека для завантаження зображень, відео та анімаційних GIF. З Glide можна завантажувати і відображати медіа з різних джерел, наприклад віддалених серверів або з локальної файлової системи.

Використовувалась для завантаження та відображення мапи, після процесу трекінгу[46].

Navigation components

Для організації навігації в додатку, переходів між вікнами. Для опрацювання натискань кнопок, візуалізації всіх можливих шляхів та переходів які доступні користувачеві[47].

EasyPermissions

При створенні або розробці програми для Android виникають ситуації, коли необхідно використовувати функції деяких інших системних додатків, присутніх на кожному мобільному пристрої. Наприклад, для створення додатку «Галерея зображень», крім функцій самого додатку, також буде потрібно використовувати функції системних додатків, таких як камера і сховище файлів. Один із способів досягнення цієї мети – створити власний додаток для камери та сховища файлів. Це можна зробити, але це нерозумно – написати так багато рядків коду, якщо можна досягти того ж, просто написавши 4-5 рядків коду.

В додатку фітнес помічник, під час виконання функції трекінгу, потрібен доступ до місцезнаходження пристрою. Оскільки це системна функція, доцільно застосувати бібліотеку EasyPermissions, для отримання доступу до системних функцій, а саме місцезнаходження[48].

MPAndroidChart

Застосовувався для відображення стовпчикової діаграми, а саме для детальної статистики по кожному забігові[49].

Room, Room-ktx

Бібліотека room забезпечує рівень абстракції над SQLite, що забезпечує більш надійний доступ до бази даних, і дає змогу використовувати всі можливості SQLite.

Бібліотека room-ktx додає підтримку функцій для транзакцій баз даних, а саме: select, insert, update, delete[50].

Coroutines

Застосовується тоді коли потрібно завантажити щось з мережі, витягти дані з бази даних або просто виконати довгі обчислення і при цьому не заблокувати інтерфейс користувача. Власне під час роботи з БД в фітнес помічникові, для того щоб не блокувати інтерфейс, було застосовано корунтини[51].

Google-maps

Для функції трекінгу були додані мапи на основі даних сервісу «Google-maps», а також налаштований їх зовнішній вигляд і реакція на жести. Крім того, додані маркери на мапі, лінія маршруту, з заданням частоти оновлення[52].

3.4 Програмна реалізація та аналіз роботи

Для зручності використання додатку, прийнято рішення розділити функціонал по вікнам, так щоб кожні складні функції відповідало 1 вікно.

Реєстрація та авторизація

Для реєстрації та авторизації в базі даних створено відповідну таблицю person (див. рис. 3.1 та табл. 3.2). Створено окремі вікна в додатку (див. рис. 3.3).



а) б) в)

Рисунок 3.3 – Реєстрація та авторизація в додатку «Healthy – Runner Tracker»

На (рис. 3.3.а) зображено процес авторизації. В цьому вікні присутні:

- привітальне повідомлення – Welcome Back.
- Два текстових поля з підказками до них, Email – поле для введення пошти та Password – поле для введення паролю.
- Присутня кнопка Login – яка виконує функцію авторизації. А саме порівнює введену пошту з списком поштових скриньок, які зберігаються а базі даних, якщо така пошта є, то далі порівнюється введений в текстове поле пароль з паролем якому відповідає введена поштова скриня. І якщо паролі також співпадають користувач авторизований.

- Присутня кнопка-напис «Have an account?», яка перенаправляє на сторінку створення облікового запису (див. рис. 3.3.б).

На (рис 3.3.б) зображено процес реєстрації, в цьому вікні присутні:

- привітальне повідомлення – Welcome.
- Три текстових поля з підказками до них, Name – поле для введення свого імені, Email – поле для введення пошти та Password – поле для введення паролю.
- Присутня кнопка Register – яка виконує функцію реєстрації. Спершу порівнюється пошта, яку ввів, з списком поштових скриньок, які зберігаються а базі даних, якщо така пошта є, то користувачеві видається повідомлення, що: «обліковий запис з такою поштою існує, використайте іншу пошту». В випадку якщо пошта унікальна, то в базі даних створюється новий запис, і користувач зареєстрований. Далі іде перенаправлення на вікно з внесенням персональних даних (див. рис. 3.3.в).

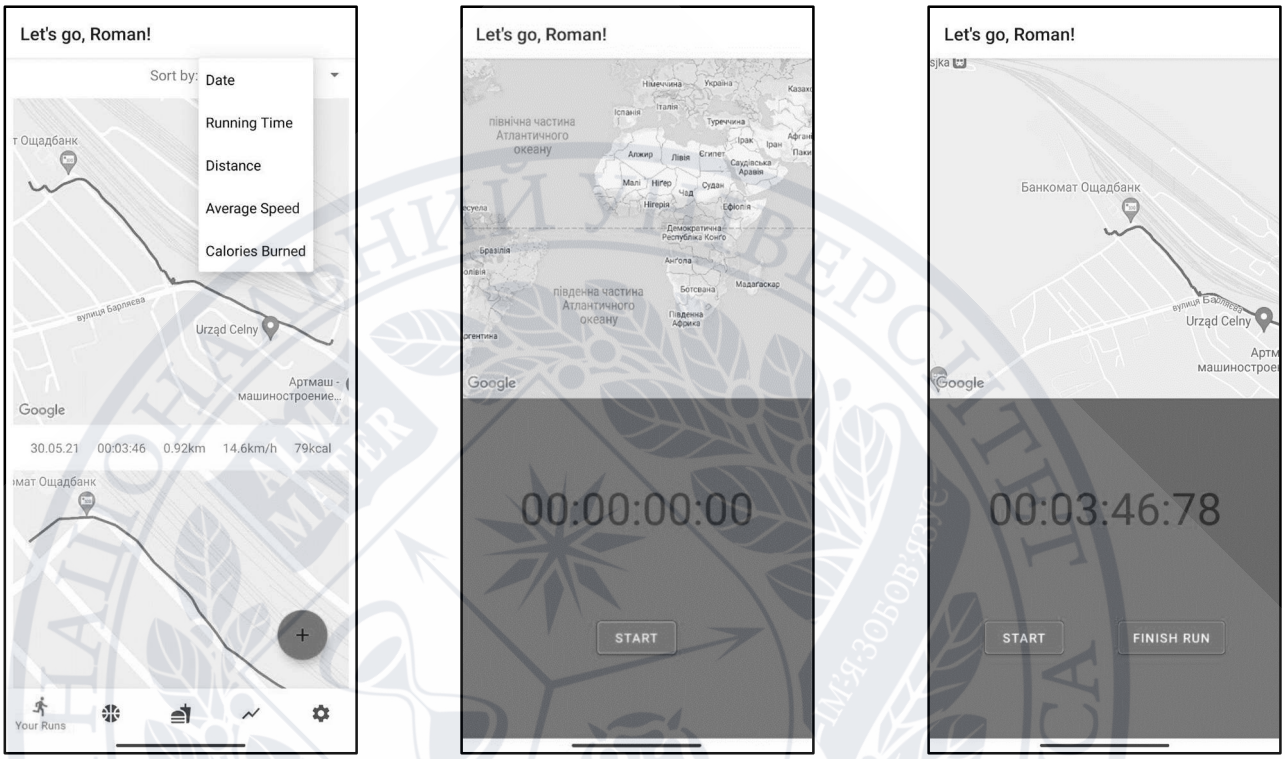
На (рис 3.3.в) зображено процес реєстрації, в цьому вікні присутні:

- Текстове поле, з підказкою Name – поле призначене для імені
- Текстове поле, з підказкою Your Weight – поле призначене для введення своєї ваги.
- Текстове поле, з підказкою Your Height – поле призначене для введення свого зросту.
- Текстове поле, з підказкою Your Age – поле призначене для введення свого віку.
- Радіокнопки Male, Female – використовуються для вибору своєї статті.

Вся інформація буде зберігатись до БД і в подальшому використовуватися для обчислень пов’язаних з витраченими калоріями. Основний код для реєстрації та авторизації (див. додаток Б класи LoginFragment, RegisterFragment, SettingsFragment).

Функція трекінгу

Для коректної роботи функції трекінгу, в базі даних створено відповідну таблицю running (див. рис. 3.1 та табл. 3.2 та додаток Б клас Run). Створено окремі вікна в додатку (див. рис. 3.4).



а) б) в)

Рисунок 3.4 – Функція трекінгу в додатку «Healthy – Runner Tracker»

На (рис. 3.4.а) зображено вигляд вікна з забігами. Тут можна детально переглядати статистику по кожному забігові окремо, переглянути маршрут на мапі, дату забігу, тривалість та відстань маршруту, середню швидкість на маршруті та кількість витрачених калорій.

За потреби, можна виконати сортування забігів (див. рис. 3.4.а), можна відсортувати забіги від новіших до старіших. Відсортувати забіги за тривалістю, в порядку зменшення. За відстанню, в порядку зменшення. Відсортувати в порядку зменшення за середньою швидкістю на маршруті. Також можна відсортувати в порядку зменшення, за кількість витрачених калорій, під час подолання маршруту. Типи сортування (див. додаток Б, клас SortType.kt та TrackingRepository).

Для створення нового забігу потрібно натиснути на зелену кнопку з плюсом, тоді додаток прийме вигляд як на (рис. 3.4.б), це початковий вигляд забігу. Видно, що таймер в початковому положенні, і місцезнаходження поки не визначається. Для початку забігу потрібно натиснути на кнопку Start, таймер запуститься, а маршрут буде фіксуватися на мапі.

Також реалізована функція паузи трекінгу, якщо користувач стомився чи відбулась якась надзвичайна ситуація, він може натиснути на кнопку паузи. Тоді додаток прийме вигляд як на (рис. 3.4.в). Таймер призупиняється і переміщення на мапі не фіксуються. Далі користувач може продовжити або завершити маршрут. Основний код для трекінгу (див. додаток Б клас TrackingService та TrackingRepository, TrackingFragment).

Після завершення маршруту отримані дані будуть занесені в локальну БД, а кількість витрачених калорій розрахується за однією з формул: (2.1) – для ходьби, (2.2) – для бігу або (2.3) – для велозаїзду.

Калькулятор калорій фізичних активностей та їжі



а) б)

Рисунок 3.5 – Калькулятор калорій фізичних активностей та їжі в додатку «Healthy – Runner Tracker»

Для підрахунку калорій, під час фізичних активностей та споживанні їжі, в базі даних створенні відповідні таблиці exercise та food (див. рис. 3.1, табл. 3.3 та табл.3.4). Створено окремі вікна в додатку (див. рис. 3.3).

На (рис. 3.5.б) зображено вигляд вікна калькулятора калорій фізичних активностей. Функціонал в цьому вікні дозволяє визначити скільки енергії витрачає користувач, під час занять різними видами спорту. Все відбувається таким чином: користувач нажимає на кнопку, з зображенням плюса (див. рис. 3.5.б), йому видається весь список вправ, збережених в БД. Він обирає потрібну вправу і вказує скільки часу він займався даною спортивною активністю. Кількість витраченої енергії під час фізичних активностей розраховується за формулою (3.1)

$$K = \frac{t \times kkal_{sport}}{60}, \tag{3.1}$$

де K – кількість витраченої енергії, ккал;

t – час виконання фізичної активності, хв;

$kkal_{sport}$ – кількість ккал які витрачаються при зайнятті певною фізичною активністю, ккал/год.

Всі виконані спортивні активності зберігаються і доступні користувачеві в розділі Today’s exercises, але вони завжди видаляються о 00.00 (отримую значення часу з смартфона), адже це щоденний облік фізичних активностей.

На (рис. 3.5.а) зображено вигляд вікна калькулятора калорій їжі. Функціонал в цьому вікні дозволяє визначити скільки енергії отримує користувач, під час споживання певного виду їжі. Все відбувається таким чином: користувач вводить в текстове поле Food ту їжу яку він спожив, а в поле Weight приблизну масу спожитого продукту. Далі автоматично розрахується кількість спожитих калорій, і всі внесені дані запишуться в БД. А для користувача ця інформація знаходиться в розділі Today’s food (див. рис. 3.5.а), але ці дані завжди видаляються о 00.00 (отримую значення часу з смартфона), адже це щоденний

облік спожитої їжі. Кількість отриманої енергії під споживання їжі розраховується за формулою (3.2)

$$K = \frac{m \times kkal_{food}}{100}, \tag{3.2}$$

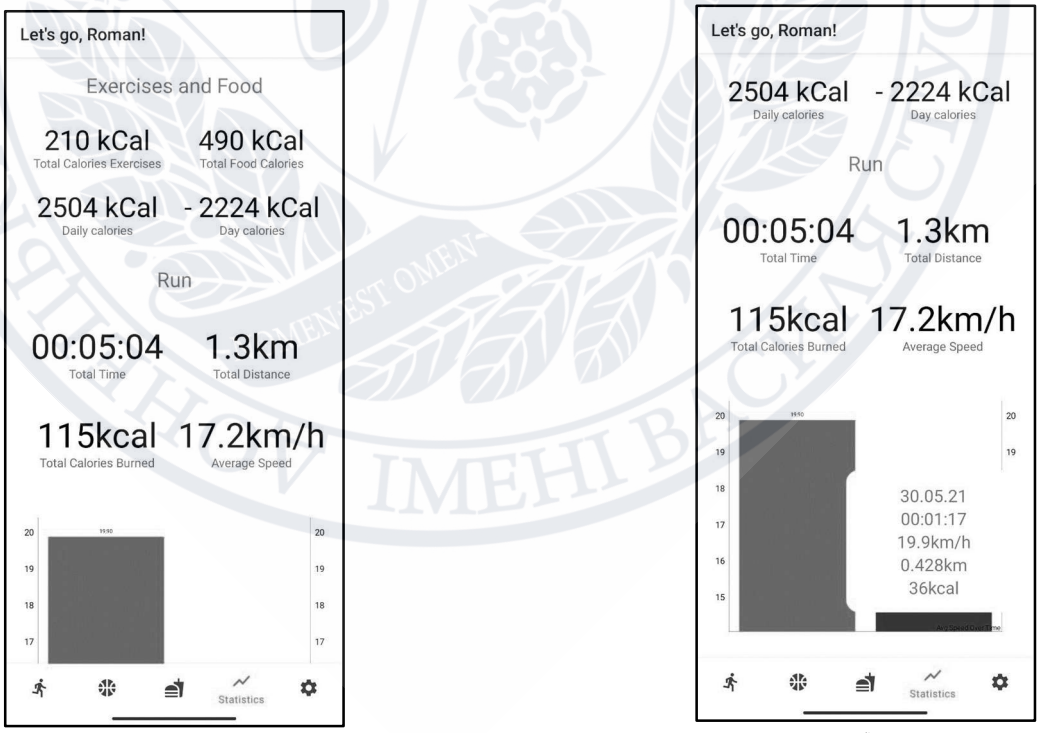
де K – кількість отриманої енергії, ккал;

m – маса спожитої їжі, гр;

$kkal_{food}$ – кількість ккал отримуються під час вживання певного виду їжі, ккал/100гр. Основний код для даної функції (див. додаток Б класи ExercisesFragment FoodFragment).

Детальна статистика

Для систематизації даних та виділення основної статистики, створено окреме вікно в додатку (див. рис. 3.6).



а) б)

Рисунок 3.6 – Калькулятор калорій фізичних активностей та їжі в додатку «Healthy – Runner Tracker»

В розділі Daily statistics, можна переглянути статистику обміну енергією за поточну добу.

Total Calories Exercises – зміна в якій зберігається сума витраченої енергії під час занять спортом, ккал.

Total Food Calories – зміна в якій зберігається сума отриманої енергії шляхом споживання їжі, ккал.

Daily Calories – зміна в якій зберігається кількість енергії необхідної організму, для виконання життєво необхідних функцій, ккал.

Day Calories – зміна в якій обраховується добова кількість енергії за формулою (3.3), ккал.

$$K_{day} = K_{food} - (K_{run} + K_{sport} + K_{body}), \tag{3.3}$$

де K_{day} - добова кількість енергії, ккал;

K_{food} - отримується з значення Total Food Calories, ккал;

K_{run} - робиться запит до БД, відбираються і смутується енергія витрачена під час забігів, за сьогоднішню добу, ккал;

K_{sport} - отримується з значення Total Calories Exercises, ккал;

K_{body} - отримується з значення Daily Calories, ккал.

В розділі Run можна переглянути загальну статистику забігів.

Total Time – змінна в якій зберігається сума часу всіх забігів, тобто скільки часу користувач проводить активно час.

Total Distance – змінна в якій зберігається сума дистанцій всіх забігів, тобто яку відстань користувач подолав за весь час, км.

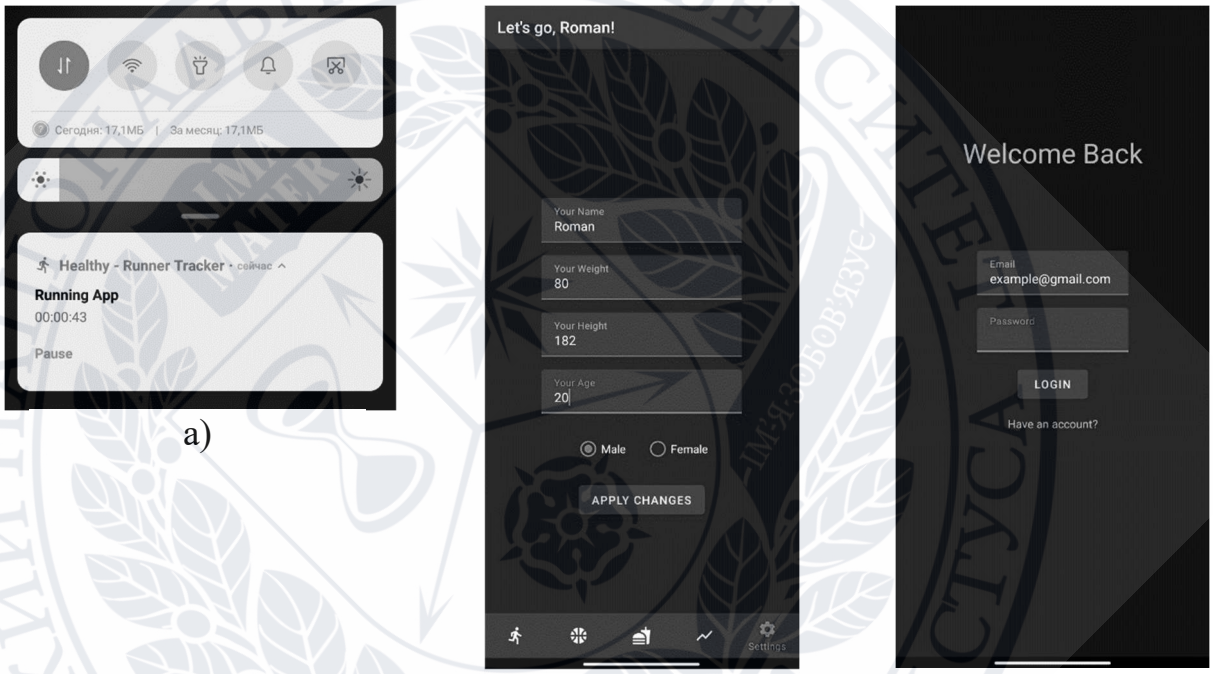
Totals Calories Burned – змінна в якій зберігається сума витраченої енергії всіх забігів, тобто скільки енергії витратив користувач під час забігів, ккал.

Average Speed - змінна в якій зберігається середня швидкість пересування користувача за весь час.

Також в цьому розділі розміщена стовпчикова діаграма окремо по кожному забігові. Тут можна переглядати свій прогрес середньої швидкості, за весь час. Основний код для даної функції (див. додаток Б клас StatisticsFragment).

Інші функції

Для коректної роботи функції трекінгу, додаток спроектований таким чином, що при ввімкнені функції трекінгу, додаток працює як служба. Тобто, додаток не припинає роботу коли він працює у фоновому режимі і тоді коли він заблокований (див. рис. 3.7.а).



а) б) в)

Рисунок 3.7 – Інші функції в додатку «Healthy – Runner Tracker»

Також для зручності у використанні, в темну пору доби, розроблено темну тему (див. рис. 3.7.б та рис. 3.7.в), завдяки бібліотеці Material-components.

Тестування функціоналу трекінгу

Для тестування функції трекінгу було проведено 16 тестів. 8 тестів було проведено на дистанції 300м., для ходьби та бігу по 4 тести. Та 8 тестів на дистанції 1250м., для ходьби та бігу по 4 тести. Для зручності отримані дані систематизовано та зведено до табличного виду (див. табл. 3.5).

Таблиця 3.5 – Тестові забіги для перевірки функції трекінгу

Дистанція, м	Час, хв	Середня швидкість, км/год	Кількість витраченої енергії, ккал	Тип активності
304	3,04	6,0	20,53	Ходьба
304	3,32	5,5	20,42	Ходьба
306	3,53	5,2	20,56	Ходьба
305	3,33	5,5	20,48	Ходьба
305	2,65	6,9	26,23	Біг
302	2,52	7,2	25,97	Біг
301	2,58	7,0	25,89	Біг
303	2,67	6,8	26,06	Біг
1245	14,94	5,0	83,82	Ходьба
1240	13,53	5,5	83,28	Ходьба
1242	14,06	5,3	83,41	Ходьба
1249	14,41	5,2	83,91	Ходьба
1246	10,68	7,0	107,16	Біг
1244	10,51	7,1	106,98	Біг
1243	11,13	6,7	106,90	Біг
1245	10,83	6,9	107,07	Біг

Для наглядності, з отриманих табличних значень було створено стовпцеву діаграму (див. рис. 3.8).

Як видно з (табл. 3.5 та рис. 3.8), кількість витраченої енергії під час бігу дійсно більша за кількість витраченої енергії під час ходьби. Також видно те, що чим більша інтенсивність (середня швидкість) на однаковому маршруті, тим більша кількість витраченої енергії.

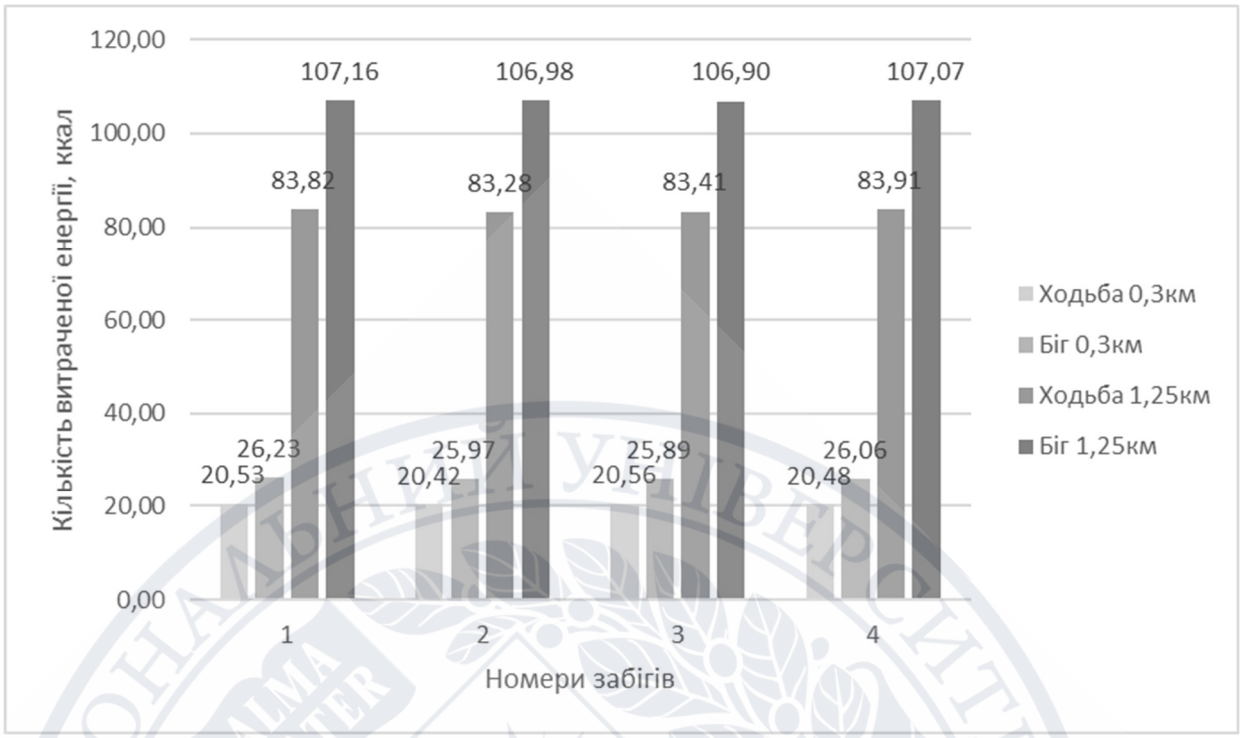


Рисунок 3.8 – Кількість витраченої енергії при ходьбі та бігові на різних дистанціях

З наведеного вище, видно, що додаток коректно працює і виконує всі функції які зазначені в постановці завдання.

Висновок до розділу 3

В межах даного розділу була розглянута структура бази даних, з описом таблиць представлених в дані БД. Наведені аргументи в користь обраних технологій розробки. Наведено перелік застосованих бібліотек, та те для чого вони застосовувались. Розглянуто програмну реалізацію та проаналізовані результати роботи додатку.

ВИСНОВКИ

В дані бакалаврські роботі був спроектований і розроблений додаток – фітнес помічник, під платформу Android, з використанням мови програмування Kotlin.

Були розглянуті в загальних рисах фітнес помічники їх властивості та функції. Розглянуто уже існуючі рішення подібних додатків, з виділенням позитивних та негативних рис. Розглянуті платформи для розробки та мови програмування для відповідних платформ. Наведені аргументи в користь обраних технологій розробки.

В результаті сконструйований додаток має якості, відібрані в результаті аналізу програм конкурентів. Проведена робота дозволила вивчити існуючи методи та підходи в розробці подібних додатків.

Розроблений додаток має наступні можливості:

- Зручний користувацький інтерфейс, а саме розроблені 2 теми, звичайна та dark mode. Реалізований зручний перехід між вікнами додатку завдяки компоненту bottom navigation.
- Можливість створення особистого акаунту з фіксуванням всіх активностей та змін персональних даних.
- Функція трекінгу, з отриманням зображення маршруту на мапі, інформації про дату забігу, тривалість та відстань маршруту, середню швидкість на маршруті і кількість витрачених калорій.
- Калькулятор калорій спортивних активностей.
- Калькулятор калорій спожитої їжі.
- Детальна статистика фізичних активностей за поточну добу та за весь період.

Створений додаток може бути використаний як повноцінний фітнес помічник з базовими функціями для моніторингу процесів пов’язаних з обміном енергії.

Додаток розрахований для масового використання, тобто, будь-яка особа зможе встановити та використовувати додаток для власних потреб.

ПЕРЕЛІК ВИКОРСТАНИХ ДЖЕРЕЛ

1. Інформаційні технології в медицині. URL: https://uk.wikipedia.org/wiki/%D0%86%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D1%96%D0%B9%D0%BD%D1%96_%D1%82%D0%B5%D1%85%D0%BD%D0%BE%D0%BB%D0%BE%D0%B3%D1%96%D1%97_%D0%B2_%D0%BC%D0%B5%D0%B4%D0%B8%D1%86%D0%B8%D0%BD%D1%96 (дата звернення: 01.03.2021).
2. ІТ ТЕХНОЛОГІЇ В МЕДИЦИНІ, Фрідріхсон Наталія Володимирівна, ВП НУБіП України «Ірпінський економічний коледж».
3. Інформаційні технології в медицині. URL: https://med.oits.pro/2017/09/12/informatsijni-tehnologiyi-v-medytsyni/ (дата звернення: 05.03.2021).
4. Фитнес трекер – современный помощник для занятий спортом. URL: https://rdxgym.com.ua/fitnes-treker-sovremennyj-pomoshhnik-dlya-zanyatij-sportom/ (дата звернення: 05.03.2021).
5. Strava: Track Running, Cycling & Swimming. URL: https://play.google.com/store/apps/details?id=com.strava&hl=uk (дата звернення: 16.03.2021).
6. Щоденник харчування. URL: https://play.google.com/store/apps/details?id=com.dailybits.foodjournal&hl=uk (дата звернення: 16.03.2021).
7. iOS или Android: выбираем платформу для мобильного приложения. URL: https://punicapp.com/blog/pages/544/ios-ili-android-vybiraem-platformu-dlya-mobilnogo-prilozheniya/ (дата звернення: 01.04.2021).
8. Выбор платформы для разработки мобильных приложений. URL: https://docs.microsoft.com/ru-ru/azure/developer/mobile-apps/choose-mobile-framework (дата звернення: 01.04.2021).
9. Что такое платформа Android и для чего она нужна. URL: https://mob-mobile.ru/statya/4188-chto-takoe-platforma-androida-i-dlya-chego-ona-nuzhna.html (дата звернення: 05.04.2021).

10. Платформа Android – история создания и современность. URL: https://woxapp.com/ru/our-blog/android-platform-history-of-creation-and-modernity/ (дата звернення: 12.04.2021).
11. Плюсы и минусы ОС IOS и Андроид. URL: https://hs-store.ru/articles/interesnoe/plyusy-i-minusy-os-ios-i-android/ (дата звернення: 01.03.2021).
12. Android или iOS — Что выбрать в 2021 году. URL: https://droidblog.org/help/android-ili-ios (дата звернення: 12.04.2021).
13. "Head First Java, Изучаем Java", Кэти Сьерра, Берт Бэйтс, 2003.
14. C# in Depth, John Skit, 2008. 544с.
15. Learning Kotlin, Good Library, 2021. 181с.
16. SwiftUI Apprentice (1st Edition), Razeware LLC, Audrey Tam, Caroline Begbie, 2021. 706с.
17. More iPhone Development with Objective-C, Apress, Horovitz, Alex;Kim, Kevin;LaMarche, Jeff;Mark, David;Varma, Jayant, 2015. 512с.
18. Сколько калорий мы тратим при беге и ходьбе? URL: https://livelong.pro/skolko-kalorij-my-tratim-pri-bege-i-hodbe/ (дата звернення: 21.04.2021).
19. Energy Expenditure of Walking and Running: Comparison with Prediction Equations. Cameron Hall, Arturo Figueroa, Bo Fernhall, Jill A. Kanaley, 2004.
20. Сколько калорий сжигается при ходьбе (калькулятор). URL: http://frs24.ru/st/kalkulator-rashoda-kalorij-pri-hodbe/ (дата звернення: 25.04.2021).
21. Сколько калорий сжигается при беге (калькулятор). URL: http://frs24.ru/st/kalkulator-rashoda-kalorij-pri-bege/ (дата звернення: 27.04.2021).
22. Расход калорий при езде на велосипеде (калькулятор). URL: http://frs24.ru/st/kalkulator-rashoda-kalorij-pri-ezde-na-velosipede/ (дата звернення: 27.04.2021).

23. Рахуємо калорії та худнемо: як визначити свою добову норму). URL: https://gobeauty.blog/ua/rahuyemo-kaloriyi-ta-hudnemo-yak-vyznachyty-svoyu-dobovu-normu/ (дата звернення: 01.05.2021).
24. Дефіцит калорій: що це і як правильно розрахувати? URL: https://stylus.ua/uk/articles/814.html (дата звернення: 01.05.2021).
25. A new predictive equation for resting energy expenditure in healthy individuals, M.D. Mifflin, S.T. St. Jeor, L.A. Hill, B.J. Scott, S.A. Daugherty, Y.O. Koh, 1990. ст. 241–247.
26. The Unified Modeling Language User Guide, Grady Booch, Addison-Wesley, 2005. 475с.
27. UML 2003 -- The Unified Modeling Language, Modeling Languages and Applications: 6th International Conference San Francisco, California, 2003. 418с.
28. The Unified Software Development Proces, Jacobson, 1999. 496с.
29. Aspect-oriented Software Development with Use Cases, Ivar Jacobson, Pan-Wei Ng, 2005. 307с.
30. Андроид: универсальная и удобная операционная система. URL: https://tehnika.expert/cifrovaya/smartfon/s-os-android.html (дата звернення: 03.05.2021).
31. ANDROID STUDIO VS ECLIPSE: ПРЕИМУЩЕСТВО КАЖДОЙ ИЗ СРЕД. URL: https://kitapp.pro/android-studio-vs-eclipse-preimushhestvo-kazhdoj-iz-sred/ (дата звернення: 10.05.2021).
32. Df Android Studio 4.2 Development Essentials - Kotlin Edition: Developing Android Apps Using Android Studio 4.2, Kotlin and Android Jetpack, Payload Media, Neil Smyth, 2021. 544с.
33. Android for Absolute Beginners: Getting Started with Mobile Apps Development Using the Android Java SDK, Allen, Grant, 2021.
34. Android Studio Game Development: Concepts and Design, Jerome DiMarzio, 2021. 95с.
35. On Java 8, Bruce Eckel 2021. 136с.

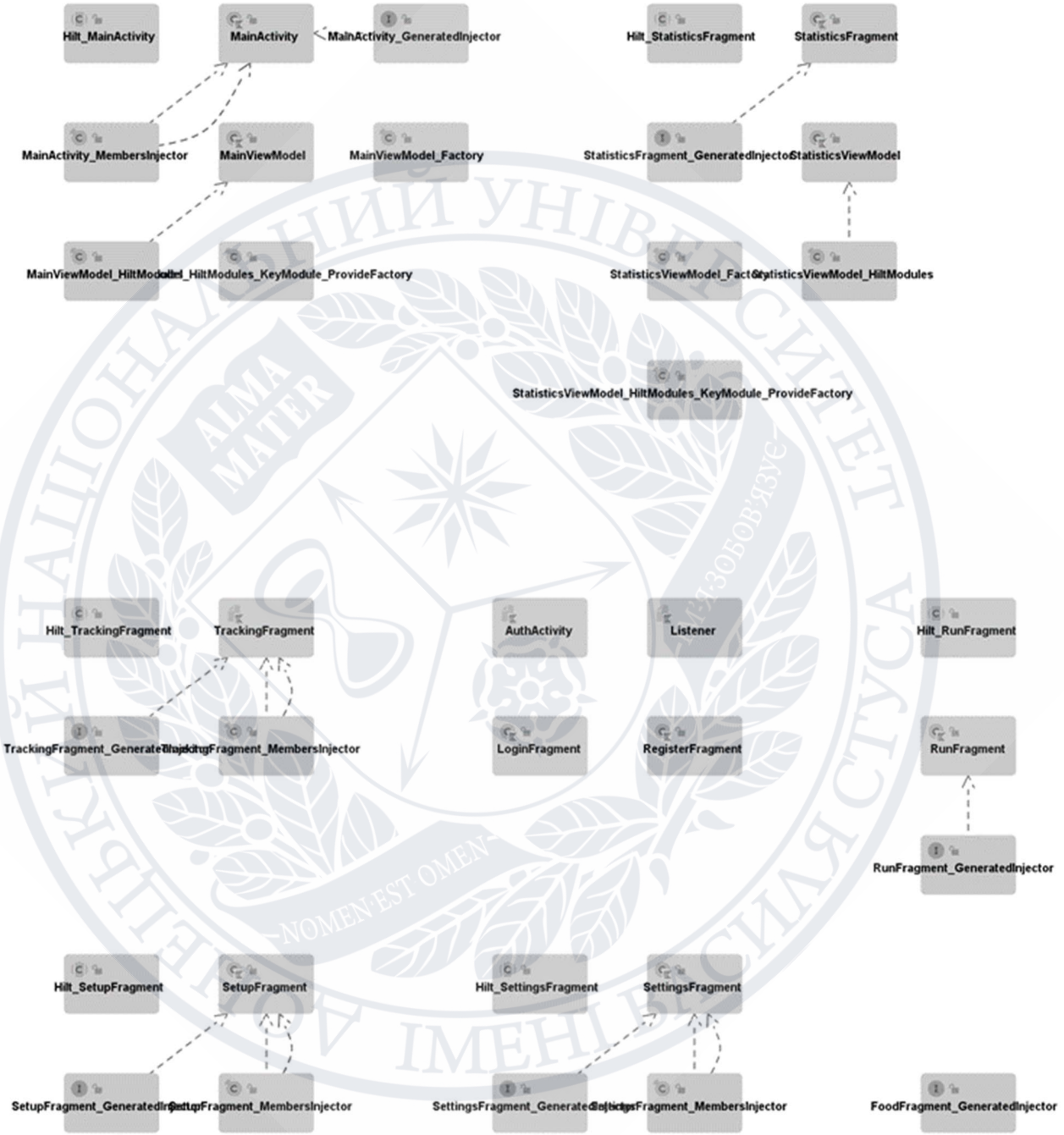
36. Java 9: Swing, Documents And Printing, Poul Klausen, 2017, 254p.
37. Java to Kotlin (Early Release), O'Reilly Media, Duncan McGregor, Nat Pryce, 2021. 92c.
38. Effective Kotlin: Best practices, Marcin Moskala, 2020.
39. Elements Of Android Room 0.5, Leanpub, Mark L. Murphy, 2021. 439c.
40. Android Apprentice (4th Edition), Razeware LLC, Darryl Bayliss, Tom Blankenship, Fuad Kamal, 2021. 724c.
41. Elements Of Kotlin 1.0, CommonsWare, Mark L. Murphy, 2021. 418c.
42. Волшебство Kotlin, ДМК Пресс, Сомон П.И., 2020. 536с.
43. Lifecycle. URL: https://startandroid.ru/ru/courses/architecture-components/27-course/architecture-components/524-urok-1.html (дата звернення: 15.05.2021).
44. Programming Android with Kotlin: Achieving Structured Concurrency with Coroutines (Early Release), O'Reilly Media, Pierre-Olivier Laurence, Amanda Hinchman-Dominguez, Mike Dunn, 2021. 543c.
45. Kotlin: программирование на примерах, СПб: БХВ-Петербург, Аделекан И., 2020. 432 с.
46. Programming DSLs in Kotlin: Design Expressive and Robust Special Purpose Code, Pragmatic Programmers, LLC, Venkat Subramaniam, 2020. 50c.
47. Kotlin Apprentice (3rd Edition), Razeware LLC, Irina Galata, Joe Howard, Ellen Shapiro. 202c.
48. Kotlin Programming for Beginners: An Introduction to Learn the Kotlin Programming Language with Tutorials and Hands-On Examples, Lightbulb Publishing, Nathan Metzler, 2021. 162c.
49. Head First Android Development, 3rd Edition, O'Reilly Media, Inc, Dawn Griffiths, David Griffiths, 2020. 127c.
50. R2DBC Revealed: Reactive Relational Database Connectivity for Java and JVM Programmers, Apress, Robert Hedgpeth, 2021. 235c.
51. Elements Of Kotlin Coroutines 0.3, CommonsWare, Mark L. Murphy, 2021. 244c.

52.Programming Kotlin Applications: Building Mobile and Server-Side Applications with Kotlin, Wiley, Brett McLaughli, 2021. 380c.

ДОДАТКИ

ДОДАТОК А

Діаграма класів



ДОДАТОК Б

Код

MainActvity

```
package com.healthy.tracker.ui.pages.base
import android.content.Intent
import android.os.Bundle
import android.view.View
import androidx.appcompat.app.AppCompatActivity
import androidx.navigation.fragment.NavHostFragment
import androidx.navigation.fragment.findNavController
import androidx.navigation.ui.setupWithNavController
import com.healthy.tracker.R
import com.healthy.tracker.other.Constants.Companion.ACTION_SHOW_TRACKING_FRAGMENT
import dagger.hilt.android.AndroidEntryPoint
import kotlinx.android.synthetic.main.activity_main.*
import javax.inject.Inject

@AndroidEntryPoint
class MainActivity : AppCompatActivity() {

    @Inject
    lateinit var name: String

    private val navHostFragment by lazy {
        supportFragmentManager.findFragmentById(R.id.navHostFragment) as NavHostFragment
    }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        //
        // setSupportActionBar(toolbar)
        bottomNavigationView.setupWithNavController(navHostFragment.findNavController())
        bottomNavigationView.setOnNavigationItemReselectedListener { /* NO-OP */ }

        navigateToTrackingFragmentIfNeeded(intent)

        if (name.isNotEmpty()) {
            val toolbarTitle = "Let's go, $name!"
            toolbar.title = toolbarTitle
        }

        navHostFragment.findNavController()
            .addOnDestinationChangedListener { _, destination, _ ->
                when (destination.id) {
                    R.id.setupFragment2, R.id.trackingFragment ->
bottomNavigationView.visibility =
                        View.GONE
                    else -> bottomNavigationView.visibility = View.VISIBLE
                }
            }
    }

    override fun onNewIntent(intent: Intent?) {
        super.onNewIntent(intent)
        navigateToTrackingFragmentIfNeeded(intent)
    }

    private fun navigateToTrackingFragmentIfNeeded(intent: Intent?) {
        if(intent?.action == ACTION_SHOW_TRACKING_FRAGMENT) {

navHostFragment.findNavController().navigate(R.id.action_global_trackingFragment)
        }
    }
}
```

MainViewModel

```
package com.healthy.tracker.ui.pages.base
import androidx.lifecycle.MediatorLiveData
import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import com.healthy.tracker.db.entities.Run
import com.healthy.tracker.other.SortType
import com.healthy.tracker.repositories.TrackingRepository
import com.healthy.tracker.utils.log
import dagger.hilt.android.lifecycle.HiltViewModel
import kotlinx.coroutines.launch
import javax.inject.Inject

@HiltViewModel
class MainViewModel @Inject constructor(
    private val trackingRepository: TrackingRepository
) : ViewModel() {

    private val runsSortedByDate = trackingRepository.getAllRunsSortedByDate()
    private val runsSortedByDistance = trackingRepository.getAllRunsSortedByDistance()
    private val runsSortedByTimeInMillis =
trackingRepository.getAllRunsSortedByTimeInMillis()
    private val runsSortedByAvgSpeed = trackingRepository.getAllRunsSortedByAvgSpeed()
    private val runsSortedByCaloriesBurned =
trackingRepository.getAllRunsSortedByCaloriesBurned()

    val runs = MediatorLiveData<List<Run>>()

    var sortType = SortType.DATE

    /**
     * Posts the correct run list in the LiveData
     */
    init {
        runs.addSource(runsSortedByDate) { result ->
            log("RUNS SORTED BY DATE")
            if (sortType == SortType.DATE) {
                result?.let { runs.value = it }
            }
        }
        runs.addSource(runsSortedByDistance) { result ->
            if (sortType == SortType.DISTANCE) {
                result?.let { runs.value = it }
            }
        }
        runs.addSource(runsSortedByTimeInMillis) { result ->
            if (sortType == SortType.RUNNING_TIME) {
                result?.let { runs.value = it }
            }
        }
        runs.addSource(runsSortedByAvgSpeed) { result ->
            if (sortType == SortType.AVG_SPEED) {
                result?.let { runs.value = it }
            }
        }
        runs.addSource(runsSortedByCaloriesBurned) { result ->
            if (sortType == SortType.CALORIES_BURNED) {
                result?.let { runs.value = it }
            }
        }
    }

    fun sortRuns(sortType: SortType) = when (sortType) {
        SortType.DATE -> runsSortedByDate.value?.let { runs.value = it }
        SortType.DISTANCE -> runsSortedByDistance.value?.let { runs.value = it }
        SortType.RUNNING_TIME -> runsSortedByTimeInMillis.value?.let { runs.value = it }
        SortType.AVG_SPEED -> runsSortedByAvgSpeed.value?.let { runs.value = it }
        SortType.CALORIES_BURNED -> runsSortedByCaloriesBurned.value?.let { runs.value =
it }
    }.also {
```

```
        this.sortType = sortType
    }

    fun insertRun(run: Run) = viewModelScope.launch {
        trackingRepository.insertRun(run)
    }

    fun deleteRun(run: Run) = viewModelScope.launch {
        trackingRepository.deleteRun(run)
    }
}
```

AppModule

```
package com.healthy.tracker.di.modules
import android.app.Application
import android.content.Context.MODE_PRIVATE
import android.content.SharedPreferences
import androidx.room.Room
import com.healthy.tracker.db.dao.RunDao
import com.healthy.tracker.db.database.RunningDatabase
import com.healthy.tracker.other.Constants.Companion.DATABASE_NAME
import com.healthy.tracker.other.Constants.Companion.KEY_FIRST_TIME_TOGGLE
import com.healthy.tracker.other.Constants.Companion.KEY_NAME
import com.healthy.tracker.other.Constants.Companion.KEY_WEIGHT
import com.healthy.tracker.other.Constants.Companion.SHARED_PREFERENCES_NAME
import dagger.Module
import dagger.Provides
import dagger.hilt.InstallIn
import dagger.hilt.components.SingletonComponent
import javax.inject.Singleton

/**
 * AppModule, provides application wide singletons
 */
@Module
@InstallIn(SingletonComponent::class)
object AppModule {

    @Singleton
    @Provides
    fun provideAppDb(app: Application): RunningDatabase {
        return Room.databaseBuilder(app, RunningDatabase::class.java, DATABASE_NAME)
            .fallbackToDestructiveMigration()
            .build()
    }

    @Singleton
    @Provides
    fun provideRunDao(db: RunningDatabase): RunDao {
        return db.getRunDao()
    }

    @Singleton
    @Provides
    fun provideSharedPreferences(app: Application): SharedPreferences {
        return app.getSharedPreferences(SHARED_PREFERENCES_NAME, MODE_PRIVATE)
    }

    @Singleton
    @Provides
    fun provideName(sharedPreferences: SharedPreferences): String {
        return sharedPreferences.getString(KEY_NAME, "") ?: ""
    }

    @Singleton
    @Provides
    fun provideWeight(sharedPreferences: SharedPreferences): Float {
        return sharedPreferences.getFloat(KEY_WEIGHT, 80f)
    }
```

```
    @Singleton
    @Provides
    fun provideFirstTimeToggle(sharedPreferences: SharedPreferences): Boolean {
        return sharedPreferences.getBoolean(
            KEY_FIRST_TIME_TOGGLE, true
        )
    }
}
```

ServiceModule

```
package com.healthy.tracker.di.modules
import android.app.PendingIntent
import android.content.Context
import android.content.Intent
import androidx.core.app.NotificationCompat
import com.healthy.tracker.R
import com.healthy.tracker.other.Constants
import com.healthy.tracker.ui.pages.base.MainActivity
import com.google.android.gms.location.FusedLocationProviderClient
import dagger.Module
import dagger.Provides
import dagger.hilt.InstallIn
import dagger.hilt.android.components.ServiceComponent
import dagger.hilt.android.qualifiers.ApplicationContext
import dagger.hilt.android.scopes.ServiceScoped

/**
 * ServiceModule, provides dependencies for the TrackingService
 */
@Module
@InstallIn(ServiceComponent::class)
object ServiceModule {

    @ServiceScoped
    @Provides
    fun providesFusedLocationProviderClient(
        @ApplicationContext context: Context
    ) = FusedLocationProviderClient(context)

    @ServiceScoped
    @Provides
    fun provideBaseNotificationBuilder(
        @ApplicationContext context: Context,
        pendingIntent: PendingIntent
    ) = NotificationCompat.Builder(context, Constants.NOTIFICATION_CHANNEL_ID)
        .setAutoCancel(false)
        .setOngoing(true)
        .setSmallIcon(R.drawable.ic_directions_run_black_24dp)
        .setContentTitle("Running App")
        .setContentText("00:00:00")
        .setContentIntent(pendingIntent)

    @ServiceScoped
    @Provides
    fun provideActivityPendingIntent(
        @ApplicationContext context: Context
    ) =
        PendingIntent.getActivity(
            context,
            0,
            Intent(context, MainActivity::class.java).apply {
                action = Constants.ACTION_SHOW_TRACKING_FRAGMENT
            },
            PendingIntent.FLAG_UPDATE_CURRENT
        )
}
```

RunningDatabase

```
package com.healthy.tracker.db.database
import androidx.room.Database
```

```
import androidx.room.RoomDatabase
import androidx.room.TypeConverters
import com.healthy.tracker.db.converters.Converters
import com.healthy.tracker.db.dao.RunDao
import com.healthy.tracker.db.entities.Run

@Database(
    entities = [Run::class],
    version = 1
)
@TypeConverters(Converters::class)
abstract class RunningDatabase : RoomDatabase() {

    abstract fun getRunDao(): RunDao
}
```

Run

```
package com.healthy.tracker.db.entities
import android.graphics.Bitmap
import androidx.room.Entity
import androidx.room.PrimaryKey
import java.util.*

@Entity(tableName = "running_table")
data class Run(
    var img: Bitmap? = null,
    var timestamp: Long = 0L,
    var avgSpeedInKMH: Float = 0f,
    var distanceInMeters: Int = 0,
    var timeInMillis: Long = 0,
    var caloriesBurned: Int = 0
) {
    @PrimaryKey(autoGenerate = true)
    var id: Int? = null
}
```

SortType

```
package com.healthy.tracker.other
/**
 * Describes the sort type of the run list
 */
enum class SortType {
    DATE,
    RUNNING_TIME,
    DISTANCE,
    AVG_SPEED,
    CALORIES_BURNED
}
```

TrackingRepository

```
package com.healthy.tracker.repositories
import com.healthy.tracker.db.entities.Run
import com.healthy.tracker.db.dao.RunDao
import javax.inject.Inject

class TrackingRepository @Inject constructor(private val runDao: RunDao) {

    suspend fun insertRun(run: Run) = runDao.insertRun(run)

    suspend fun deleteRun(run: Run) = runDao.deleteRun(run)

    fun getAllRunsSortedByDate() = runDao.getAllRunsSortedByDate()

    fun getAllRunsSortedByTimeInMillis() = runDao.getAllRunsSortedByTimeInMillis()

    fun getAllRunsSortedByDistance() = runDao.getAllRunsSortedByDistance()

    fun getAllRunsSortedByCaloriesBurned() = runDao.getAllRunsSortedByCaloriesBurned()
```

```
    fun getAllRunsSortedByAvgSpeed() = runDao.getAllRunsSortedByAvgSpeed()

    fun getTotalDistance() = runDao.getTotalDistance()

    fun getTotalTimeInMillis() = runDao.getTotalTimeInMillis()

    fun getTotalAvgSpeed() = runDao.getTotalAvgSpeed()

    fun getTotalCaloriesBurned() = runDao.getTotalCaloriesBurned()
}
```

TrackingService

```
package com.healthy.tracker.services
import android.annotation.SuppressLint
import android.app.NotificationChannel
import android.app.NotificationManager
import android.app.NotificationManager.IMPORTANCE_LOW
import android.app.PendingIntent
import android.app.PendingIntent.FLAG_UPDATE_CURRENT
import android.content.Context
import android.content.Intent
import android.location.Location
import android.os.Build
import android.os.Looper
import androidx.annotation.RequiresApi
import androidx.core.app.NotificationCompat
import androidx.lifecycle.LifecycleService
import androidx.lifecycle.MutableLiveData
import androidx.lifecycle.observe
import com.google.android.gms.location.FusedLocationProviderClient
import com.google.android.gms.location.LocationCallback
import com.google.android.gms.location.LocationRequest
import com.google.android.gms.location.LocationRequest.PRIORITY_HIGH_ACCURACY
import com.google.android.gms.location.LocationResult
import com.google.android.gms.maps.model.LatLng
import com.healthy.tracker.R
import com.healthy.tracker.other.Constants
import com.healthy.tracker.other.Constants.Companion.ACTION_PAUSE_SERVICE
import com.healthy.tracker.other.Constants.Companion.ACTION_START_OR_RESUME_SERVICE
import com.healthy.tracker.other.Constants.Companion.ACTION_STOP_SERVICE
import com.healthy.tracker.other.Constants.Companion.FASTEST_LOCATION_UPDATE_INTERVAL
import com.healthy.tracker.other.Constants.Companion.LOCATION_UPDATE_INTERVAL
import com.healthy.tracker.other.Constants.Companion.NOTIFICATION_CHANNEL_ID
import com.healthy.tracker.other.Constants.Companion.NOTIFICATION_CHANNEL_NAME
import com.healthy.tracker.other.Constants.Companion.NOTIFICATION_ID
import com.healthy.tracker.other.TrackingUtility
import com.healthy.tracker.utils.log
import dagger.hilt.android.AndroidEntryPoint
import kotlinx.coroutines.CoroutineScope
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.delay
import kotlinx.coroutines.launch
import javax.inject.Inject

typealias Polyline = MutableList<LatLng>
typealias Polylines = MutableList<Polyline>

@AndroidEntryPoint
class TrackingService : LifecycleService() {

    private val timeRunInSeconds = MutableLiveData<Long>()

    private var isFirstRun = true
    private var serviceKilled = false

    companion object {
        val timeRunInMillis = MutableLiveData<Long>()
        val isTracking = MutableLiveData<Boolean>()
        val pathPoints = MutableLiveData<Polylines>()
```

```
}

/**
 * Base notification builder that contains the settings every notification will have
 */
@Inject
lateinit var baseNotificationBuilder: NotificationCompat.Builder

/**
 * Builder of the current notification
 */
private lateinit var curNotification: NotificationCompat.Builder

@Inject
lateinit var fusedLocationProviderClient: FusedLocationProviderClient

override fun onCreate() {
    super.onCreate()
    curNotification = baseNotificationBuilder
    postInitialValues()

    isTracking.observe(this) {
        updateNotificationTrackingState(it)
        updateLocationChecking(it)
    }
}

private fun postInitialValues() {
    timeRunInMillis.postValue(0L)
    isTracking.postValue(false)
    pathPoints.postValue(mutableListOf())
    timeRunInSeconds.postValue(0L)
}

override fun onStartCommand(intent: Intent?, flags: Int, startId: Int): Int {
    intent?.let {
        when (it.action) {
            ACTION_START_OR_RESUME_SERVICE -> {
                if(isFirstRun) {
                    startForegroundService()
                    isFirstRun = false
                    serviceKilled = false
                } else {
                    startTimer()
                }
            }
            ACTION_PAUSE_SERVICE -> {
                log("Paused Service")
                pauseService()
            }
            ACTION_STOP_SERVICE -> {
                log("Stopped service.")
                killService()
            }
        }
    }
    return super.onStartCommand(intent, flags, startId)
}

/**
 * Stops the service properly.
 */
private fun killService() {
    serviceKilled = true
    isFirstRun = true
    pauseService()
    postInitialValues()
    stopForeground(true)
    stopSelf()
}
```

```
    /**
     * Enables or disables location tracking according to the tracking state.
     */
    @SuppressLint("MissingPermission")
    private fun updateLocationChecking(isTracking: Boolean) {
        if (isTracking) {
            if (TrackingUtility.hasLocationPermissions(this)) {
                val request = LocationRequest().apply {
                    interval = LOCATION_UPDATE_INTERVAL
                    fastestInterval = FASTEST_LOCATION_UPDATE_INTERVAL
                    priority = PRIORITY_HIGH_ACCURACY
                }
                fusedLocationProviderClient.requestLocationUpdates(request,
locationCallback, Looper.getMainLooper())
            }
        } else {
            fusedLocationProviderClient.removeLocationUpdates(locationCallback)
        }
    }

    /**
     * Location Callback that receives location updates and adds them to pathPoints.
     */
    private val locationCallback = object : LocationCallback() {
        override fun onLocationResult(result: LocationResult?) {
            super.onLocationResult(result)
            if(isTracking.value!!) {
                result?.locations?.let { locations ->
                    for(location in locations) {
                        addPathPoint(location)
                    }
                }
            }
        }
    }

    private var isTimerEnabled = false
    private var lapTime = 0L // time since we started the timer
    private var timeRun = 0L // total time of the timer
    private var timeStarted = 0L // the time when we started the timer
    private var lastSecondTimestamp = 0L

    /**
     * Starts the timer for the tracking.
     */
    private fun startTimer() {
        addEmptyPolyline()
        isTracking.postValue(true)
        timeStarted = System.currentTimeMillis()
        isTimerEnabled = true
        CoroutineScope(Dispatchers.Main).launch {
            while (isTracking.value!!) {
                // time difference between now and time started
                lapTime = System.currentTimeMillis() - timeStarted
                // post the new laptime
                timeRunInMillis.postValue(timeRun + lapTime)
                // if a new second was reached, we want to update timeRunInSeconds, too
                if (timeRunInMillis.value!! >= lastSecondTimestamp + 1000L) {
                    timeRunInSeconds.postValue(timeRunInSeconds.value!! + 1)
                    lastSecondTimestamp += 1000L
                }
                delay(Constants.TIMER_UPDATE_INTERVAL)
            }
            timeRun += lapTime
        }
    }

    /**
     * Disables the timer and tracking.
```

```
     */
    private fun pauseService() {
        isTimerEnabled = false
        isTracking.postValue(false)
    }

    /**
     * This adds the location to the last list of pathPoints.
     */
    private fun addPathPoint(location: Location?) {
        location?.let {
            val pos = LatLng(location.latitude, location.longitude)
            pathPoints.value?.apply {
                last().add(pos)
                pathPoints.postValue(this)
            }
        }
    }

    /**
     * Will add an empty polyline in the pathPoints list or initialize it if empty.
     */
    private fun addEmptyPolyline() = pathPoints.value?.apply {
        add(mutableListOf())
        pathPoints.postValue(this)
    } ?: pathPoints.postValue(mutableListOf(mutableListOf()))

    /**
     * Starts this service as a foreground service and creates the necessary notification
     */
    private fun startForegroundService() {
        log("TrackingService started.")

        val notificationManager =
            getSystemService(Context.NOTIFICATION_SERVICE) as NotificationManager

        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
            createNotificationChannel(notificationManager)
        }

        startForeground(NOTIFICATION_ID, curNotification.build())
        //curNotification = curNotification.setContentIntent(getActivityPendingIntent())
        startTimer()
        isTracking.postValue(true)

        // updating notification
        timeRunInSeconds.observe(this) {
            if(!serviceKilled) {
                val notification = curNotification
                    .setContentText(TrackingUtility.getFormattedStopWatchTime(it *
1000L))
                notificationManager.notify(NOTIFICATION_ID, notification.build())
            }
        }
    }

    /**
     * Updates the action buttons of the notification
     */
    private fun updateNotificationTrackingState(isTracking: Boolean) {
        val notificationActionText = if (isTracking) "Pause" else "Resume"
        val pendingIntent = if (isTracking) {
            val pauseIntent = Intent(this, TrackingService::class.java).apply {
                action = ACTION_PAUSE_SERVICE
            }
            PendingIntent.getService(this, 1, pauseIntent, FLAG_UPDATE_CURRENT)
        } else {
            val resumeIntent = Intent(this, TrackingService::class.java).apply {
                action = ACTION_START_OR_RESUME_SERVICE
            }
```

```
            PendingIntent.getService(this, 2, resumeIntent, FLAG_UPDATE_CURRENT)
        }
        val notificationManager =
            getSystemService(Context.NOTIFICATION_SERVICE) as NotificationManager

        curNotification.javaClass.getDeclaredField("mActions").apply {
            isAccessible = true
            set(curNotification, ArrayList<NotificationCompat.Action>())
        }

        if(!serviceKilled) {
            curNotification = baseNotificationBuilder
                .addAction(R.drawable.ic_pause_black_24dp, notificationActionText,
pendingIntent)
            notificationManager.notify(NOTIFICATION_ID, curNotification.build())
        }
    }

    @RequiresApi(Build.VERSION_CODES.O)
    private fun createNotificationChannel(notificationManager: NotificationManager) {
        val channel = NotificationChannel(NOTIFICATION_CHANNEL_ID,
NOTIFICATION_CHANNEL_NAME, IMPORTANCE_LOW)
        notificationManager.createNotificationChannel(channel)
    }
}
```

LoginFragment

```
package com.healthy.tracker.ui.pages.auth

import android.os.Bundle
import android.view.View
import androidx.fragment.app.Fragment
import com.healthy.tracker.R
import kotlinx.android.synthetic.main.fragment_login.*

class LoginFragment : Fragment(R.layout.fragment_login) {

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        login.setOnClickListener {
            (activity as Listener).onClick()
        }
    }
}
```

RegisterFragment

```
package com.healthy.tracker.ui.pages.auth

import android.content.Intent
import android.os.Bundle
import android.view.View
import androidx.fragment.app.Fragment
import com.healthy.tracker.R
import com.healthy.tracker.ui.pages.base.MainActivity
import kotlinx.android.synthetic.main.fragment_register.*

class RegisterFragment : Fragment(R.layout.fragment_register) {

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        register.setOnClickListener {
            startActivity(Intent(requireContext(), MainActivity::class.java))
        }
    }
}
```

ExercisesFragment

```
package com.healthy.tracker.ui.pages.exercises

import android.os.Bundle
```

```
import android.view.View
import android.widget.TextView
import androidx.annotation.IdRes
import androidx.fragment.app.Fragment
import com.healthy.tracker.R
import dagger.hilt.android.AndroidEntryPoint

@AndroidEntryPoint
class ExercisesFragment : Fragment(R.layout.fragment_exercises) {

   override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
      super.onViewCreated(view, savedInstanceState)

      setupExercise(R.id.exercise_2, "Football", "- 150 kCal")
      setupExercise(R.id.exercise_3, "Volleyball", "- 100 kCal")

      setupTodayExercise(R.id.todayExercises2, "Volleyball", "- 100 cKal")
   }

   private fun setupTodayExercise(@IdRes id: Int, name: String, kCal: String) {
      view?.findViewById<View>(id)?.let {
         it.findViewById<TextView>(R.id.exercise).text = name
         it.findViewById<TextView>(R.id.force).text = kCal
      }
   }

   private fun setupExercise(@IdRes id: Int, name: String, kCal: String) {
      view?.findViewById<View>(id)?.let {
         it.findViewById<TextView>(R.id.exercise).text = name
         it.findViewById<TextView>(R.id.force).text = kCal
      }
   }
}
```

FoodFragment

```
package com.healthy.tracker.ui.pages.food

import android.os.Bundle
import android.view.View
import android.widget.TextView
import androidx.annotation.IdRes
import androidx.fragment.app.Fragment
import com.healthy.tracker.R
import dagger.hilt.android.AndroidEntryPoint

@AndroidEntryPoint
class FoodFragment : Fragment(R.layout.fragment_food) {

   override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
      super.onViewCreated(view, savedInstanceState)

      setupFood(R.id.food_2, "Boiled Potato", "150 g", "+ 120 kCal")
   }

   private fun setupFood(@IdRes id: Int, name: String, weight: String, kCal: String) {
      view?.findViewById<View>(id)?.let {
         it.findViewById<TextView>(R.id.exercise).text = name
         it.findViewById<TextView>(R.id.weight).text = weight
         it.findViewById<TextView>(R.id.force).text = kCal
      }
   }
}
```

RunFragment

```
package com.healthy.tracker.ui.pages.run

import android.Manifest
import android.os.Build
import android.os.Bundle
```

```
import android.view.View
import android.widget.AdapterView
import androidx.fragment.app.Fragment
import androidx.fragment.app.viewModels
import androidx.lifecycle.Observer
import androidx.navigation.fragment.findNavController
import androidx.recyclerview.widget.ItemTouchHelper
import androidx.recyclerview.widget.LinearLayoutManager
import androidx.recyclerview.widget.RecyclerView
import com.healthy.tracker.R
import com.healthy.tracker.ui.adapters.RunAdapter
import com.healthy.tracker.other.Constants.Companion.REQUEST_CODE_LOCATION_PERMISSION
import com.healthy.tracker.other.SortType
import com.healthy.tracker.other.TrackingUtility
import com.healthy.tracker.ui.pages.base.MainViewModel
import com.google.android.material.snackbar.Snackbar
import dagger.hilt.android.AndroidEntryPoint
import kotlinx.android.synthetic.main.fragment_run.*
import pub.devrel.easypermissions.AppSettingsDialog
import pub.devrel.easypermissions.EasyPermissions

@AndroidEntryPoint
class RunFragment : Fragment(R.layout.fragment_run), EasyPermissions.PermissionCallbacks
{

    lateinit var runAdapter: RunAdapter

    private val viewModel: MainViewModel by viewModels()

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        //viewModel = (activity as MainActivity).mainViewModel
        runAdapter = RunAdapter()

        setupRecyclerView()
        requestPermissions()
        fab.setOnClickListener {
            findNavController().navigate(R.id.action_runFragment2_to_trackingFragment)
        }

        when (viewModel.sortType) {
            SortType.DATE -> spFilter.setSelection(0)
            SortType.RUNNING_TIME -> spFilter.setSelection(1)
            SortType.DISTANCE -> spFilter.setSelection(2)
            SortType.AVG_SPEED -> spFilter.setSelection(3)
            SortType.CALORIES_BURNED -> spFilter.setSelection(4)
        }
        viewModel.runs.observe(viewLifecycleOwner, Observer { runs ->
            runAdapter.submitList(runs)
        })

        spFilter.onItemSelectedListener = object : AdapterView.OnItemSelectedListener {
            override fun onNothingSelected(adapterView: AdapterView<*>?) {}

            override fun onItemSelected(
                adapterView: AdapterView<*>?,
                view: View?,
                pos: Int,
                id: Long
            ) {
                when (pos) {
                    0 -> viewModel.sortRuns(SortType.DATE)
                    1 -> viewModel.sortRuns(SortType.RUNNING_TIME)
                    2 -> viewModel.sortRuns(SortType.DISTANCE)
                    3 -> viewModel.sortRuns(SortType.AVG_SPEED)
                    4 -> viewModel.sortRuns(SortType.CALORIES_BURNED)
                }
            }
        }
    }
```

```
    /**
     * Handles swipe-to-delete
     */
    private val itemTouchHelperCallback = object : ItemTouchHelper.SimpleCallback(
        ItemTouchHelper.UP or ItemTouchHelper.DOWN, ItemTouchHelper.LEFT or
ItemTouchHelper.RIGHT
    ) {
        override fun onMove(
            recyclerView: RecyclerView,
            viewHolder: RecyclerView.ViewHolder,
            target: RecyclerView.ViewHolder
        ): Boolean {
            return true
        }

        override fun onSwiped(viewHolder: RecyclerView.ViewHolder, direction: Int) {
            val position = viewHolder.layoutPosition
            val run = runAdapter.differ.currentList[position]
            viewModel.deleteRun(run)
            Snackbar.make(requireView(), "Successfully deleted run",
Snackbar.LENGTH_LONG).apply {
                setAction("Undo") {
                    viewModel.insertRun(run)
                }
                show()
            }
        }
    }

    private fun setupRecyclerView() = rvRuns.apply {
        adapter = runAdapter
        layoutManager = LinearLayoutManager(activity)
        ItemTouchHelper(itemTouchHelperCallback).attachToRecyclerView(this)
    }

    private fun requestPermissions() {
        if (TrackingUtility.hasLocationPermissions(requireContext())) {
            return
        }
        if (Build.VERSION.SDK_INT < Build.VERSION_CODES.Q) {
            EasyPermissions.requestPermissions(
                this,
                "You need to accept location permission to use this app",
                REQUEST_CODE_LOCATION_PERMISSION,
                Manifest.permission.ACCESS_FINE_LOCATION,
                Manifest.permission.ACCESS_COARSE_LOCATION
            )
        } else {
            EasyPermissions.requestPermissions(
                this,
                "You need to accept location permissions to use this app",
                REQUEST_CODE_LOCATION_PERMISSION,
                Manifest.permission.ACCESS_FINE_LOCATION,
                Manifest.permission.ACCESS_COARSE_LOCATION,
                Manifest.permission.ACCESS_BACKGROUND_LOCATION
            )
        }
    }

    override fun onPermissionsDenied(requestCode: Int, perms: MutableList<String>) {
        if (EasyPermissions.somePermissionPermanentlyDenied(this, perms)) {

AppSettingsDialog.Builder(this).setThemeResId(R.style.AlertDialogTheme).build().show()
        } else {
            requestPermissions()
        }
    }

    override fun onPermissionsGranted(requestCode: Int, perms: MutableList<String>) {}
```

```
    override fun onRequestPermissionsResult(
        requestCode: Int,
        permissions: Array<out String>,
        grantResults: IntArray
    ) {
        super.onRequestPermissionsResult(requestCode, permissions, grantResults)
        EasyPermissions.onRequestPermissionsResult(requestCode, permissions,
grantResults, this)
    }
}
```

SettingsFragment

```
package com.healthy.tracker.ui.pages.settings

import android.content.SharedPreferences
import android.os.Bundle
import android.view.View
import androidx.fragment.app.Fragment
import com.healthy.tracker.R
import com.healthy.tracker.other.Constants.Companion.KEY_NAME
import com.healthy.tracker.other.Constants.Companion.KEY_WEIGHT
import com.google.android.material.snackbar.Snackbar
import dagger.hilt.android.AndroidEntryPoint
import kotlinx.android.synthetic.main.activity_main.*
import kotlinx.android.synthetic.main.fragment_settings.*
import javax.inject.Inject

@AndroidEntryPoint
class SettingsFragment : Fragment(R.layout.fragment_settings) {

    @Inject
    lateinit var sharedPref: SharedPreferences

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        loadFieldsFromSharedPref()

        btnApplyChanges.setOnClickListener {
            val success = applyChangesToSharedPref()
            if(success) {
                Snackbar.make(requireView(), "Saved changes",
Snackbar.LENGTH_SHORT).show()
            } else {
                Snackbar.make(requireView(), "Please fill out all the fields",
Snackbar.LENGTH_SHORT).show()
            }
        }
    }

    private fun loadFieldsFromSharedPref() {
        val name = sharedPref.getString(KEY_NAME, "")
        val weight = sharedPref.getFloat(KEY_WEIGHT, 80f)
        etName.setText(name)
        etWeight.setText(weight.toString())
    }

    private fun applyChangesToSharedPref(): Boolean {
        val nameText = etName.text.toString()
        val weightText = etWeight.text.toString()
        if(nameText.isEmpty() || weightText.isEmpty()) {
            return false
        }
        sharedPref.edit()
            .putString(KEY_NAME, nameText)
            .putFloat(KEY_WEIGHT, weightText.toFloat())
            .apply()
        val toolbarText = "Let's go, $nameText!"
        requireActivity().toolbar.title = toolbarText
        return true
```

```
    }
}
```

SetupFragment

```
package com.healthy.tracker.ui.pages.setup

import android.content.SharedPreferences
import android.os.Bundle
import android.view.View
import androidx.fragment.app.Fragment
import androidx.navigation.NavOptions
import androidx.navigation.fragment.findNavController
import com.healthy.tracker.R
import com.healthy.tracker.other.Constants.Companion.KEY_FIRST_TIME_TOGGLE
import com.healthy.tracker.other.Constants.Companion.KEY_NAME
import com.healthy.tracker.other.Constants.Companion.KEY_WEIGHT
import com.google.android.material.snackbar.Snackbar
import dagger.hilt.android.AndroidEntryPoint
import kotlinx.android.synthetic.main.activity_main.*
import kotlinx.android.synthetic.main.fragment_setup.*
import javax.inject.Inject

@AndroidEntryPoint
class SetupFragment : Fragment(R.layout.fragment_setup) {

    @Inject
    lateinit var sharedPref: SharedPreferences

    @set:Inject
    var firstTimeAppOpen: Boolean = true

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        if (!firstTimeAppOpen) {
            val navOptions = NavOptions.Builder()
                .setPopUpTo(R.id.setupFragment2, true)
                .build()
            findNavController().navigate(
                R.id.action_setupFragment2_to_runFragment2,
                savedInstanceState,
                navOptions
            )
        }

        tvContinue.setOnClickListener {
            val success = writePersonalDataToSharedPref()
            if (success) {
                findNavController().navigate(R.id.action_setupFragment2_to_runFragment2)
            } else {
                Snackbar.make(requireView(), "Please enter all the fields.",
Snackbar.LENGTH_SHORT)
                    .show()
            }
        }
    }

    /**
     * Saves the name and the weight in shared preferences
     */
    private fun writePersonalDataToSharedPref(): Boolean {
        val name = etName.text.toString()
        val weightText = etWeight.text.toString()
        if (name.isEmpty() || weightText.isEmpty()) {
            return false
        }
        sharedPref.edit()
            .putString(KEY_NAME, name)
            .putFloat(KEY_WEIGHT, weightText.toFloat())
            .putBoolean(KEY_FIRST_TIME_TOGGLE, false)
```

```
            .apply()
        val toolbarText = "Let's go, $name!"
        requireActivity().toolbar.title = toolbarText
        return true
    }

}
```

StatisticsFragment

```
package com.healthy.tracker.ui.pages.statistics

import android.content.res.Configuration
import android.graphics.Color
import android.os.Bundle
import android.view.View
import androidx.core.content.ContextCompat
import androidx.fragment.app.Fragment
import androidx.fragment.app.viewModels
import androidx.lifecycle.Observer
import com.github.mikephil.charting.components.XAxis
import com.github.mikephil.charting.data.*
import com.healthy.tracker.R
import com.healthy.tracker.other.TrackingUtility
import com.healthy.tracker.ui.view.CustomMarkerView
import dagger.hilt.android.AndroidEntryPoint
import kotlinx.android.synthetic.main.fragment_statistics.*
import kotlin.math.round

@AndroidEntryPoint
class StatisticsFragment : Fragment(R.layout.fragment_statistics) {

    private val viewModel: StatisticsViewModel by viewModels()

   private val chartColor: Int
      get() = when (resources.configuration.uiMode and Configuration.UI_MODE_NIGHT_MASK)
{
          Configuration.UI_MODE_NIGHT_YES -> Color.WHITE
          else -> Color.BLACK
      }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        setupLineChart()
        subscribeToObservers()
    }

    private fun setupLineChart() {
      barChart.xAxis.apply {
            position = XAxis.XAxisPosition.BOTTOM
            setDrawLabels(false)
            axisLineColor = chartColor
            textColor = chartColor
            setDrawGridLines(false)
        }
        barChart.axisLeft.apply {
            axisLineColor = chartColor
            textColor = chartColor
            setDrawGridLines(false)
        }
        barChart.axisRight.apply {
            axisLineColor = chartColor
            textColor = chartColor
            setDrawGridLines(false)
        }
        barChart.apply {
            description.text = "Avg Speed Over Time"
            legend.isEnabled = false
        }
    }
```

```
    private fun subscribeToObservers() {
        viewModel.totalDistance.observe(viewLifecycleOwner, Observer {
         // in case DB is empty it will be null
         it?.let {
            val km = it / 1000f
            val totalDistance = round(km * 10) / 10f
            val totalDistanceString = "${totalDistance}km"
            tvTotalDistance.text = totalDistanceString
         }
      })

        viewModel.totalTimeInMillis.observe(viewLifecycleOwner, Observer {
         it?.let {
            val totalTimeInMillis = TrackingUtility.getFormattedStopWatchTime(it)
            tvTotalTime.text = totalTimeInMillis
         }
      })

        viewModel.totalAvgSpeed.observe(viewLifecycleOwner, Observer {
         it?.let {
            val roundedAvgSpeed = round(it * 10f) / 10f
            val totalAvgSpeed = "${roundedAvgSpeed}km/h"
            tvAverageSpeed.text = totalAvgSpeed
         }
      })

        viewModel.totalCaloriesBurned.observe(viewLifecycleOwner, Observer {
         it?.let {
            val totalCaloriesBurned = "${it}kcal"
            tvTotalCalories.text = totalCaloriesBurned
         }
      })

        viewModel.runsSortedByDate.observe(viewLifecycleOwner, Observer {
         it?.let {
            val allAvgSpeeds = it.indices.map { i -> BarEntry(i.toFloat(),
it[i].avgSpeedInKMH) }

            val bardataSet = BarDataSet(allAvgSpeeds, "Avg Speed over Time")
            bardataSet.apply {
               valueTextColor = chartColor
               color = ContextCompat.getColor(requireContext(), R.color.colorAccent)
            }
            val lineData = BarData(bardataSet)
            barChart.data = lineData
            val marker = CustomMarkerView(
                  it.reversed(),
                  requireContext(),
                  R.layout.marker_view
            )
            barChart.marker = marker
            barChart.invalidate()
         }
      })
    }
}
```

TrackingFragment

```
package com.healthy.tracker.ui.pages.tracking

import android.annotation.SuppressLint
import android.content.Intent
import android.os.Bundle
import android.view.*
import androidx.fragment.app.Fragment
import androidx.fragment.app.viewModels
import androidx.lifecycle.Observer
import androidx.navigation.fragment.findNavController
import com.healthy.tracker.R
import com.healthy.tracker.db.entities.Run
```

```
import com.healthy.tracker.other.Constants.Companion.ACTION_PAUSE_SERVICE
import com.healthy.tracker.other.Constants.Companion.ACTION_START_OR_RESUME_SERVICE
import com.healthy.tracker.other.Constants.Companion.ACTION_STOP_SERVICE
import com.healthy.tracker.other.Constants.Companion.MAP_VIEW_BUNDLE_KEY
import com.healthy.tracker.other.Constants.Companion.MAP_ZOOM
import com.healthy.tracker.other.Constants.Companion.POLYLINE_COLOR
import com.healthy.tracker.other.Constants.Companion.POLYLINE_WIDTH
import com.healthy.tracker.other.TrackingUtility
import com.healthy.tracker.services.TrackingService
import com.healthy.tracker.ui.pages.base.MainViewModel
import com.google.android.gms.maps.CameraUpdateFactory
import com.google.android.gms.maps.GoogleMap
import com.google.android.gms.maps.model.LatLng
import com.google.android.gms.maps.model.LatLngBounds
import com.google.android.gms.maps.model.PolylineOptions
import com.google.android.material.snackbar.Snackbar
import com.healthy.tracker.ui.dialog.CancelRunDialog
import com.healthy.tracker.utils.log
import dagger.hilt.android.AndroidEntryPoint
import kotlinx.android.synthetic.main.fragment_tracking.*
import java.util.*
import javax.inject.Inject
import kotlin.math.round

const val CANCEL_DIALOG_TAG = "CancelDialog"

@AndroidEntryPoint
class TrackingFragment : Fragment(R.layout.fragment_tracking) {

    @set:Inject
    var weight: Float = 80f

    private var map: GoogleMap? = null

    private var isTracking = false
    private var curTimeInMillis = 0L
    private var pathPoints = mutableListOf<MutableList<LatLng>>()

    private val viewModel: MainViewModel by viewModels()

    private var menu: Menu? = null

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        setHasOptionsMenu(true)
        return super.onCreateView(inflater, container, savedInstanceState)
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        val mapViewBundle = savedInstanceState?.getBundle(MAP_VIEW_BUNDLE_KEY)
        mapView.onCreate(mapViewBundle)

        // restore dialog instance
        if(savedInstanceState != null) {
            val cancelRunDialog =
parentFragmentManager.findFragmentByTag(CANCEL_DIALOG_TAG) as CancelRunDialog?
            cancelRunDialog?.setYesListener {
                stopRun()
            }
        }

        btnToggleRun.setOnClickListener {
            toggleRun()
        }

        btnFinishRun.setOnClickListener {
```

```
            zoomToWholeTrack()
            endRunAndSaveToDB()
        }

        mapView.getMapAsync {
            map = it
            addAllPolylines()
        }
        subscribeToObservers()
    }

    /**
     * Subscribes to changes of LiveData objects
     */
    private fun subscribeToObservers() {
        TrackingService.isTracking.observe(viewLifecycleOwner, Observer {
            updateTracking(it)
        })

        TrackingService.pathPoints.observe(viewLifecycleOwner, Observer {
            pathPoints = it
            addLatestPolyline()
            moveCameraToUser()
        })

        TrackingService.timeRunInMillis.observe(viewLifecycleOwner, Observer {
            curTimeInMillis = it
            val formattedTime = TrackingUtility.getFormattedStopWatchTime(it, true)
            tvTimer.text = formattedTime
        })
    }

    /**
     * Will move the camera to the user's location.
     */
    private fun moveCameraToUser() {
        if (pathPoints.isNotEmpty() && pathPoints.last().isNotEmpty()) {
            map?.animateCamera(
                CameraUpdateFactory.newLatLngZoom(
                    pathPoints.last().last(),
                    MAP_ZOOM
                )
            )
        }
    }

    /**
     * Adds all polylines to the pathPoints list to display them after screen rotations
     */
    private fun addAllPolylines() {
        for (polyline in pathPoints) {
            val polylineOptions = PolylineOptions()
                .color(POLYLINE_COLOR)
                .width(POLYLINE_WIDTH)
                .addAll(polyline)
            map?.addPolyline(polylineOptions)
        }
    }

    /**
     * Draws a polyline between the two latest points.
     */
    private fun addLatestPolyline() {
        // only add polyline if we have at least two elements in the last polyline
        if (pathPoints.isNotEmpty() && pathPoints.last().size > 1) {
            val preLastLatLng = pathPoints.last()[pathPoints.last().size - 2]
            val lastLatLng = pathPoints.last().last()
            val polylineOptions = PolylineOptions()
                .color(POLYLINE_COLOR)
                .width(POLYLINE_WIDTH)
```

```
            .add(preLastLatLng)
            .add(lastLatLng)

        map?.addPolyline(polylineOptions)
    }
}

/**
 * Updates the tracking variable and the UI accordingly
 */
private fun updateTracking(isTracking: Boolean) {
    this.isTracking = isTracking
    if (!isTracking && curTimeInMillis > 0L) {
        btnToggleRun.text = getString(R.string.start_text)
        btnFinishRun.visibility = View.VISIBLE
    } else if (isTracking) {
        btnToggleRun.text = getString(R.string.stop_text)
        menu?.getItem(0)?.isVisible = true
        btnFinishRun.visibility = View.GONE
    }
}

/**
 * Toggles the tracking state
 */
@SuppressLint("MissingPermission")
private fun toggleRun() {
    if (isTracking) {
        menu?.getItem(0)?.isVisible = true
        pauseTrackingService()
    } else {
        startOrResumeTrackingService()
        log("Started service")
    }
}

/**
 * Starts the tracking service or resumes it if it is currently paused.
 */
private fun startOrResumeTrackingService() =
    Intent(requireContext(), TrackingService::class.java).also {
        it.action = ACTION_START_OR_RESUME_SERVICE
        requireContext().startService(it)
    }

/**
 * Pauses the tracking service
 */
private fun pauseTrackingService() =
    Intent(requireContext(), TrackingService::class.java).also {
        it.action = ACTION_PAUSE_SERVICE
        requireContext().startService(it)
    }

/**
 * Stops the tracking service.
 */
private fun stopTrackingService() =
    Intent(requireContext(), TrackingService::class.java).also {
        it.action = ACTION_STOP_SERVICE
        requireContext().startService(it)
    }

override fun onSaveInstanceState(outState: Bundle) {
    val mapViewBundle = outState.getBundle(MAP_VIEW_BUNDLE_KEY)
    mapView?.onSaveInstanceState(mapViewBundle)
}

/**
 * Zooms out until the whole track is visible. Used to make a screenshot of the
```

```
     * MapView to save it in the database
     */
    private fun zoomToWholeTrack() {
        val bounds = LatLngBounds.Builder()
        for (polyline in pathPoints) {
            for (point in polyline) {
                bounds.include(point)
            }
        }
        val width = mapView.width
        val height = mapView.height
        map?.moveCamera(
            CameraUpdateFactory.newLatLngBounds(
                bounds.build(),
                width,
                height,
                (height * 0.05f).toInt()
            )
        )
    }

    /**
     * Saves the recent run in the Room database and ends it
     */
    private fun endRunAndSaveToDB() {
        map?.snapshot { bmp ->
            var distanceInMeters = 0
            for (polyline in pathPoints) {
                distanceInMeters +=
TrackingUtility.calculatePolylineLength(polyline).toInt()
            }
            val avgSpeed =
                round((distanceInMeters / 1000f) / (curTimeInMillis / 1000f / 60 / 60) *
10) / 10f
            val timestamp = Calendar.getInstance().timeInMillis
            val caloriesBurned = ((distanceInMeters / 1000f) * weight).toInt()
            val run =
                Run(bmp, timestamp, avgSpeed, distanceInMeters, curTimeInMillis,
caloriesBurned)
            viewModel.insertRun(run)
            Snackbar.make(
                requireActivity().findViewById(R.id.rootView),
                "Run saved successfully.",
                Snackbar.LENGTH_LONG
            ).show()
            stopRun()
        }
    }

    /**
     * Finishes the tracking.
     */
    private fun stopRun() {
        log("STOPPING RUN")
        tvTimer.text = "00:00:00:00"
        stopTrackingService()
        findNavController().navigate(R.id.action_trackingFragment_to_runFragment2)
    }

    override fun onOptionsItemSelected(item: MenuItem): Boolean {
        when (item.itemId) {
            R.id.miCancelTracking -> {
                showCancelTrackingDialog()
            }
        }
        return super.onOptionsItemSelected(item)
    }

    override fun onCreateOptionsMenu(menu: Menu, inflater: MenuInflater) {
        super.onCreateOptionsMenu(menu, inflater)
```

```
        inflater.inflate(R.menu.toolbar_menu_tracking, menu)
        this.menu = menu
    }

    override fun onPrepareOptionsMenu(menu: Menu) {
        super.onPrepareOptionsMenu(menu)
        // just checking for isTracking doesnt trigger this when rotating the device
        // in paused mode
        if (curTimeInMillis > 0L) {
            this.menu?.getItem(0)?.isVisible = true
        }
    }

    /**
     * Shows a dialog to cancel the current run.
     */
    private fun showCancelTrackingDialog() {
        CancelRunDialog().apply {
            setYesListener {
                stopRun()
            }
        }.show(parentFragmentManager, CANCEL_DIALOG_TAG)
    }

    override fun onResume() {
        mapView.onResume()
        super.onResume()
    }

    override fun onStart() {
        super.onStart()
        mapView.onStart()
    }

    override fun onPause() {
        super.onPause()
        mapView.onPause()
    }

    override fun onStop() {
        super.onStop()
        mapView.onStop()
    }

    override fun onLowMemory() {
        super.onLowMemory()
        mapView.onLowMemory()
    }

}
```