ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка Android-застосунку для допомоги ведення здорового способу життя мовою Kotlin»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Дмитро ХАРЕНКО
(підпис)

Виконав: здобувач вищої освіти групи ПД-42

Дмитро ХАРЕНКО

Керівник: Світлана ШЕВЧЕНКО
к.п.н., доцент

Рецензент: _____

Київ 2024

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
___ Ірина ЗАМРІЙ
« ____ » _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Харенку Дмитру Олександровичу

1. Тема кваліфікаційної роботи: «Розробка Android-застосунку для допомоги ведення здорового способу життя мовою Kotlin»
керівник кваліфікаційної роботи к.п.н., доц., доцент кафедри ІПЗ Світлана Шевченко,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «27» лютого 2024 р. № 36.

2. Строк подання кваліфікаційної роботи «28» травня 2024 р.

3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості про проблему ведення здорового способу життя; вбудовані засоби операційної системи View та Activity, інтегроване середовище Android Studio, мова програмування Kotlin.
4. Зміст розрахунково-пояснювальної записки:
1. Провести аналіз літературних джерел з проблеми ведення здорового способу життя.
2. Провести аналіз засобів для розробки програмного забезпечення Android-застосунку для допомоги ведення здорового способу життя.
3. Сформувати вимоги до розробки Android-застосунку для допомоги ведення здорового способу життя.

4. Спроєктувати архітектуру Android-застосунку для допомоги ведення здорового способу життя.
5. Розробити Android-застосунку для допомоги ведення здорового способу життя.
6. Провести тестування Android-застосунку для допомоги ведення здорового способу життя.

5. Перелік графічного матеріалу (презентація):
1. Аналіз аналогів
2. Задачі дипломної роботи
3. Вимоги до додатку
4. Програмні засоби реалізації
5. Діаграма варіантів використання
6. Діаграма класів
7. Екранні форми
8. Апробація результатів дослідження

6. Дата видачі завдання «28» лютого 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	28.02-06.03.2024	
2	Аналіз та дослідження програм аналогів	07.03-13.03.2024	
3	Аналіз засобів реалізації	14.03-20.03.2024	
4	Проєктування архітектури системи	21.03-27.03.2024	
5	Розробка програмного забезпечення	28.03-16.04.2024	
6	Тестування системи	17.04-28.04.2024	
7	Оформлення роботи: вступ, висновки, реферат	29.04-05.05.2024	
8	Розробка демонстраційних матеріалів	06.05-12.05.2024	
9	Попередній захист роботи	17.05.2024	

Здобувач вищої освіти _____ Дмитро ХАРЕНКО
(підпис)

Керівник
кваліфікаційної роботи _____ Світлана ШЕВЧЕНКО
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 49 стор., 6 табл., 24 рис., 50 джерел.

Мета роботи – підтримка процесів допомоги ведення здорового способу життя за рахунок використання Android-застосунку мовою Kotlin.

Об'єкт дослідження – ведення здорового способу життя.

Предмет дослідження – програмно-апаратні засоби для допомоги ведення здорового способу життя.

Короткий зміст роботи. Робота присвячена створенню Android-застосунку для допомоги у веденні здорового способу життя під назвою «HealthTone». У роботі висвітлюється актуальність теми, наголошуючи на необхідності контролю за здоров'ям в умовах сучасного життя, яке супроводжується високими темпами розвитку технологій та інформаційним перенасиченням. Підкреслюється, що програми-органайзери допомагають користувачам структурувати інформацію та нагадують про важливі аспекти здорового способу життя.

Також робота описує процес створення та налагодження програми, а також детально розглядає функціонал та алгоритми роботи програми. Визначено, що застосунок реалізовано за допомогою мови програмування Kotlin, яка підтримує об'єктно-орієнтоване програмування. Використання сучасних методів розробки та системи контролю версій Git дозволило підвищити ефективність роботи над проєктом.

Цільовою аудиторією застосунку є люди будь-якого віку, яких турбує стан свого зоров'я, можуть усвідомлено поєднувати фізичну активність, правильне харчування та регулярний відпочинок.

КЛЮЧОВІ СЛОВА: ЗДОРОВИЙ СПОСІБ ЖИТТЯ, ANDROID-ЗАСТОСУНОК, МОВА KOTLIN

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

1. Android — операційна система для мобільних пристроїв від компанії Google.
2. XML — англ. Extensible Markup Language, розширювана мова розмітки від консорціума World Wide Web Consortium.
3. Git — розподілена система керування версіями файлів і вихідного коду, розроблена Лінусом Торвальдсом.
4. XP — анлг. Extreme Programming, методологія розробки програмного забезпечення.
5. Грн — грошова одиниця гривня.
6. ЕОМ — електронна обчислювальна машина.
7. ДЕСТ — державний стандарт.
8. Репозиторій — проект, що працює на основі системи керування версій.
9. ПЕОМ — персональний комп’ютер.
10. UML — англ. Unified Modeling Language, уніфікована мова моделювання.
11. AVD — англ. Android Virtual Device, віртуальний пристрій Android.
12. ADB — англ. Android Debug Bridge, інтерфейс відладки в ОС Android.
13. QEMU — програмне забезпечення для емуляції апаратного забезпечення різних платформ.
14. X86 — архітектура процесора з однойменною системою команд.

ВСТУП

Фізичне здоров’я - головний аспект існування людини, тому що це є базовим показником, що напряму визначає тривалість та якість життя. Проте основною причиною погіршення стану здоров’я є не процес старіння, а частіше всього невірний чи недостатній догляд за собою. Сучасній людині потрібно постійно слідкувати за фізичним навантаженням, якістю харчування, щоденним споживанням води та іншим.

Але сучасне життя - далеко не тільки про фізичну складову. З швидким підвищенням рівню технологічного розвитку, комп’ютери почали посідати все більш важливі позиції в нашому побуті, а наразі вони взагалі лягли в основу майже усіх сфер діяльності людини. Проте, через діджиталізацію різко збільшився об’єм інформації, з якою людина стикається або працює кожного дня. Це, зважаючи на природні розумові обмеження нашого виду, піднімає низку проблем, пов’язаних з психічними розладами. Основною проблемою є інформаційне перенасичення населення, через що людині стає все важче концентруватися на окремих речах. Особливо гостро ця проблема постала в умовах постійного карантину, коли люди змушені ще тісніше працювати з інформаційними технологіями.

В таких умовах, людям часто допомагають програми-органайзери, що дозволяють не запам’ятовувати інформацію, яка може досить швидко забутися через величезний інформаційний потік впродовж дня, а надають можливість швидкого та зручного перегляду записаної інформації та функції автоматичного нагадування користувачу про щось.

Окремою категорією програм-органайзерів є додатки, що допомагають слідкувати за веденням здорового образу життя. На даний момент існує багато таких програм, але їх проблема в тому, що вони або не збирають інформацію по всім основним показникам, опираючись лише на один, або повністю прив’язані до апаратного пристрою-компаньйона, тому наразі тема автономного додатку до смартфону з комплексним моніторингом є значущою.

Цільовою платформою було обрано операційну систему Android, яка працює на смартфонах і є на них найбільш популярною у мобільному сегменті.

Актуальність теми полягає у створенні додатку, що допоможе користувачу звертати увагу відразу на всі основні показники, що напряму впливають на якість життя, а також стимулювати його до покращення цієї статистики.

Прикладами таких програм можуть бути: Water Drink Remind - контроль за питтям води; FatSecret KCal Calculator - калькулятор спожитих калорій; Simple Design StepCounter - крокомір. Дослідження основних методів розробки під операційну систему Android, вивчення найбільш доцільного варінанта та його використання для розробки застосунку для допомоги ведення здорового спосособу життя підтверджує актуальність та важливість даної роботи.

Об'єкт дослідження – ведення здорового способу життя.

Предмет дослідження – програмно-апаратні засоби для допомоги ведення здорового способу життя.

Мета роботи – підтримка процесів допомоги ведення здорового способу життя за рахунок використання Android-застосунку мовою Kotlin.

Для досягнення цієї мети в роботі необхідно вирішити такі *завдання*:

1. Провести аналіз літературних джерел з проблеми ведення здорового способу життя.
2. Здійснити огляд та аналіз існуючих програмних продуктів для допомоги ведення здорового способу життя, визначити їх переваги та недоліки.
3. Провести огляд засобів для розробки програмного забезпечення Android-застосунку.
4. Сформувати вимоги до розробки Android-застосунку.
5. Спроєктувати архітектуру Android-застосунку.
6. Розробити Android-застосунок для допомоги ведення здорового способу життя мовою Kotlin.
7. Провести тестування Android-застосунка для допомоги ведення здорового способу життя мовою Kotlin.
8. Провести оцінку вартості програмного продукту.

Методи дослідження. Для вирішення вищезгаданих завдань у роботі використано наступні методи: системно-структурні методи, порівняльний аналіз. Для розробки програмного забезпечення використані такі технології, інструменти та мови програмування: засоби операційної системи View та Activity, інтегроване середовище Android Studio, мова програмування Kotlin.

Наукова новизна одержаних результатів. Наукова новизна розробленого Android-застосунку полягає в комплексному моніторингу основних показників здоров'я користувача, таких як якість харчування, водний баланс, якість сну та фізичні навантаження.

Практична значущість результатів полягає у створенні програмного продукту у вигляді Android-застосунку для допомоги ведення здорового способу життя, який дозволить моніторити основні показники здоров'я користувача, таких як якість харчування, водний баланс, якість сну та фізичні навантаження.

Створений програмний продукт може бути корисним людям будь-якого віку, яких турбує стан свого зоров'я, можуть усвідомлено поєднувати фізичну активність, правильне харчування та регулярний відпочинок.

.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Постановка задачі

Android-застосування «Health tone» являє собою інструмент з постійного спостереження за життєвими показниками. Програми такого функціоналу і направленості називаються трекерами, проте більшість таких програм є «компаньйонами» - тобто вони прив’язані до фізичного пристрою, наприклад, фітнес-браслета. Але для більш точного аналізу вхідних даних, наприклад, з допомогою штучного інтелекту, потрібні обчислювальні ресурси, які мініатюрний пристрій просто фізично не може надати. Також тільки невеликий відсоток людей використовує ці пристрої, тому автономний мобільний застосунок є найкращим варіантом.

Програма має надавати зручний графічний інтерфейс, на головному меню якого повинна відображатися коротка статистика по усім показникам і швидкий аналіз. Звідси має бути можливість переходу по категоріям для кожного з показників, а також до меню налаштувань додатку. В категорії кожного показника має бути детальна статистика з графіком її зміни за останні дні, а також більш детальний її аналіз.

Повний перелік функціоналу такий:

– відображення короткої статистики за усіма показниками;

– відображення детальної статистики за кожним з показників та її аналіз;

– автоматичне заповнення статистики;

– заповнення статистики вручну.

1.2 Дослідження і аналіз об’єкту програмування

Даний проект є Android-застосуванням для моніторингу здорового способу життя. Складність створення такого рішення для комплексного моніторингу полягає у розробці методів автоматичного збору та аналізу інформації.

Для збору інформації може використовуватися:

– статистика використання пристрою, надана операційною системою;

– інформація з датчиків пристрою;

– геолокація, швидкість руху і пройдений шлях через GPS;

– рухи пристрою через акселерометр, що сигналізує про фізичну активність.

Було визначено такий перелік показників, за якими потрібно слідити людині:

1. Якість харчування - правильне вживання їжі забезпечує нормальний ріст, розвиток людини, сприяє профілактиці її здоров'я. При дотриманні правил та норм здорового харчування, знижується ризик виникнення хронічних захворювань, ожиріння, діабету, підвищеного тиску та інших проблем, що можуть виникнути через або стимулюватися неякісним харчуванням, але будуть впливати на загальне самопочуття.

Такий сильний вплив на здоров'я обумовлений тим, що більшість мінералів, елементів та вітамінів, потрібних для роботи різних механізмів тіла, людина споживає саме через продукти харчування. Також, харчування напряму впливає на емоційний стан людини - саме при цьому процесі виробляється дофамін, що є головним гормоном щастя. Кількість спожитої їжі вимірюється в калоріях, для чого встановлені такі норми на день:

– для здорових дітей - близько 2,000 калорій;

– для здорових дорослих - близько 3,000 калорій;

– для здорових дорослих літнього віку - близько 2,000 калорій;

– водний баланс: вода є основою нашого тіла(60% від загальної маси) і вона бере участь у всіх наших життєвих процесах - наприклад, регуляції температури через потовиділення і обміну речовин, у якому рідина - переносник елементів. Тому порушення водного балансу, тобто балансу споживання та втрати рідини, може привести до значних наслідків, таких як: зневоднення, проблеми з травленням, інтоксикація, проблеми з суглобами, проблеми з шкірою, головний біль, хронічна втома, проблеми з нирками. Встановлені таки норми споживання води на день:

– для здорових дітей і дорослих - 1.5 літрів;

– для дорослих з зайвою вагою - 2 літри;

– для дорослих з хворобами серця чи нирок - 1 літр;

2. Якість сну: сон - дуже важливий процес, що сильно впливає на мозок. Він ділиться на 2 основні фази, що чередуються: швидкий та повільний. Під час швидкого сну відбувається засвоєння нової інформації мозком, а під час повільного клітини мозку відновлюють запаси енергії, що не встигли поповнитися під час бадьорості.

Невірний графік сну може викликати порушення обміну речовин, підвищення ризику серцево-судинних захворювань чи погіршення психологічного стану людини. Критеріями якісного сну є його тривалість та безперервність. Безперервність важлива тому, що під час сну постійно змінюються його фази, причому це відбувається з певною циклічністю, а порушення графіку сну може викликати збій цієї системи, і, наприклад, переходів в повільну фазу сну буде менше, ніж потрібно. Недавно почалися дослідження роботи поліфазного сну, коли людина спить декілька разів на день.

Популярними прикладами таких систем сну можуть бути:

– Biphasic - 7 годин на добу за 2 заходи, з яких один вночі тривалістю 7 годин і один протягом дня тривалістю 20 хвилин. На рисунку 1.1 зображено графічне представлення цієї системи;

– Everyman - 4 години на добу за 4 заходи, з яких один тривалістю в 1 годину, а інші - по 20 хвилин. На рисунку 1.2 зображено графічне представлення цієї системи;

– Dymaxion - 2 години на добу за 4 заходи, кожний з яких по 30 хвилин. На рисунку 1.3 зображено графічне представлення цієї системи;

– Uberman - 2 години на добу за 6 заходів, кожний з яких по 20 хвилин. На рисунку 1.4 зображено графічне представлення цієї системи;

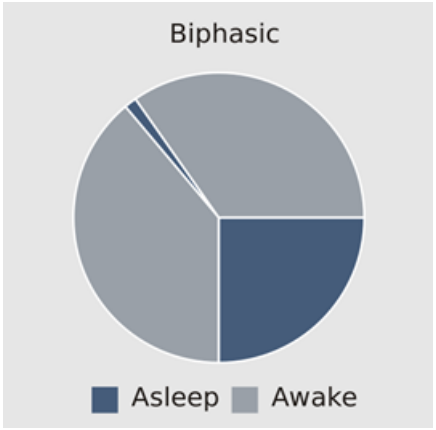


Рис. 1.1 Система сну Biphasic

Рис. 1.2 Система сну Everyman

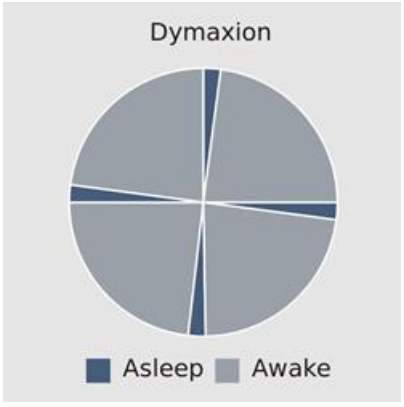


Рис. 1.3 Система сну Dymaxion

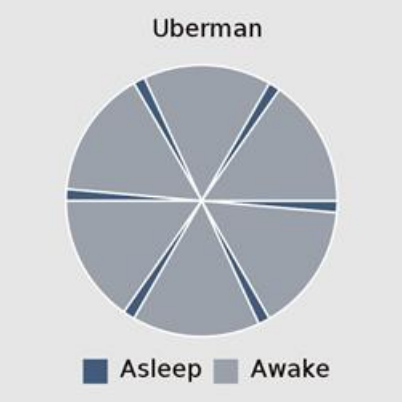


Рис. 1.4 Система сну Uberman

Проте наразі такий спосіб сну не досліджений, і його корисність чи шкода не доведені, тому застосунок буде орієнтуватися лише на стандартний спосіб сну, для якого норми тривалості на добу такі:

- для дітей - 10 годин;
- для підлітків - 9 годин;
- для дорослих - 8 годин;
- для дорослих літнього віку - 7 годин;

3. Фізичні навантаження - організм людини адаптується під умови існування, включаючи не тільки довгострокові видові зміни(еволюційно до навколишнього середовища), але й короткострокові, які можуть явно вплинути безпосередньо на людину. Найяскравішим прикладом цього є ситуація, коли тіло людини набирає м’язову масу лише при фізичних навантаженнях. При їх відсутності організм починає атрофувати м’язи, і, наприклад, для підтримки нормальної форми через відсутність ходьби, космонавти постійно тренуються на спеціальних тренажерах.

Проте схожа проблема виникає і на Землі - пасивний образ життя призводить до стану малорухомості, який викликає сповільнення обміну речовин, послаблення м’язів та кісток, а також підвищує ризики серцево-судинних захворювань.

Тому, хоч невеликі, але регулярні фізичні навантаження просто необхідні сучасним людям, коли більшість робіт - саме сидячого типу. Одним з таких регулярних навантажень є звичайна прогулянка з денною нормою в 10,000 кроків.

Переглянувши список популярних програм, пов’язаних зі здоров’ям, було виявлено, що зараз на ринку ПЗ, орієнтованого на слідкування з способом життя, існує дуже багато конкурентів, але їх спільна проблема полягає у тому, що вони спрямовані на стеження лише за одним певним показником. Прикладами можуть бути:

– Water Reminder від recorder & smart apps - має досить зручний інтерфейс і функцію нагадувань, але здійснює моніторинг лише за водним балансом і має рекламу. На рисунку 1.5 зображено інтерфейс програми;

– Step Tracker від Leap Fitness Group - має не дуже зручний і красивий інтерфейс, слідкує за пройденим шляхом і кількістю лише спалених калорій, наявна реклама, але є зручний віджет, в якому наглядно показується статистика. На рисунку 1.6 зображено інтерфейс програми;

– Huawei Health від Huawei - збирає статистику за фізичним навантаженням та якістю сну, але потребує підключення до фітнес-браслету і входу до облікового запису. Має зручний інтерфейс і не містить реклами, є функція нагадувань.

На таблиці 1.1 зображено порівняння програм-аналогів.

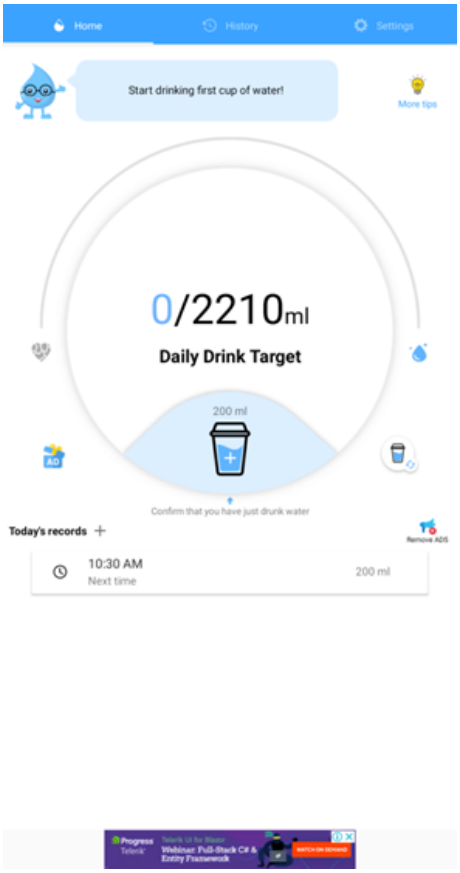


Рис. 1.5 Інтерфейс програми Water Reminder

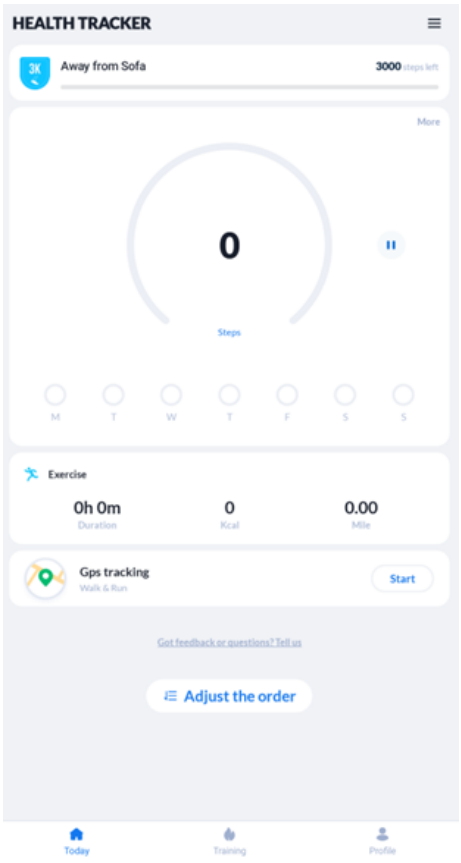


Рис. 1.6 Інтерфейс програми Step Tracker

Таблиця 1.1

Порівняння програм-аналогів

	Water Reminder	Step Tracker	Huawei Health	*Health Tone*
Моніторинг сну	-	-	+	+
Моніторинг якості харчування	-	-	-	+
Моніторинг водного балансу	+	-	-	+
Моніторинг фізичних навантажень	-	+	+	+
Спрощений вхід в додаток без облікового запису	-	-	+	+

Отже, проаналізувавши програми-аналоги, можна виділити такі вимоги до проекту: зручний інтерфейс, моніторинг сну, якості харчування, водного балансу, фізичних навантажень і відсутність вимоги до входу в обліковий запис, тому що для роботи даних функцій це не потрібно і лише погіршить досвід використання програми у користувача.

1.3 Використані програмні засоби

Програмне забезпечення для операційної системи Android представляється у вигляді APK файлу. APK(Android Package) - формат файлів, у якому упаковано скомпільований для Android код програми та усі додаткові ресурси, що потрібні для його роботи(зазвичай, мультимедійні файли).

Основним способом встановлення такого програмного забезпечення є платформа Google Play, функціями якої є:

– встановлення додатку - буде автоматично завантажено APK файл з серверу Google Play, перевірено його цифровий підпис, та встановлено;

– придбання додатку - якщо розробник зробив додаток платним, користувач має змогу купити цей додаток, після чого він з’явиться в його біблітеці додатків. Вона прив’язана до облікового запису Google, тому навіть після зміни пристрою буде можливість встановити придбані програми;

– відгуки - користувачі, що встановили або купили додаток, можуть оцінити його, встановивши оцінку від 1 до 5 та написавши коментар;

– оновлення додатків - при публікації нової версії додатку, що встановлений на телефоні, вона буде завантажена та встановлена автоматично.

Для публікації власних додатків, потрібно зареєструвати обліковий запис розробника у Google Play Console (https://play.google.com/console/developers).

Проєкт було реалізовано за допомогою мови Kotlin, що підтримує об’єктно-орієнтоване програмування. Саме вона рекомендується компанією-розробником операційної системи.

Об’єктно-орієнтоване програмування - стиль написання програмного забезпечення, при якому програмісти групують дані та функції по об’єктам, що значно полегшує взаємодію з вихідним кодом програми. Часто, при використанні цього підходу технології, програмісти притримуються паттернів програмування - способів проектування взаємодії між об’єктами.

Порівняно з Java, ця мова програмування не дуже відрізняється - вони працюють на однаковій віртуальній машині, виконують однакові функції, проте

Kotlin має зовсім інший, більш новітній синтаксис, що значно зменшує об’єм вихідного коду та підвищує ефективність роботи програмістів.

Віртуальна машина - спосіб роботи програмного забезпечення, який базується на виконанні спрощеного(компільованого) коду застосунку за допомогою програми, що є посередником між операційною системою і додатком. Такий підхід значно покращує можливість крос-платформенного запуску ПЗ, тому що окремо для кожної платформи потрібно реалізовувати лише базові функції віртуальної машини, а код самої програми при цьому залишається незмінним.

Програма має графічний інтерфейс - тобто вона візуалізує елементи керування та відображення даних. Для його побудови використовувалися вбудовані в операційну систему інструменти, такі як View та Activity.

View - це будь-який візуальний компонент, проте це можуть бути не тільки елементи керування, а також й так званий ViewGroup - групуючий контейнер, що сам по собі нічого не представляє, але вміщує інші візуальні компоненти.

Activity - це контейнери, що напряму пов’язані з екраном пристрою користувача, проте вони також здійснюють групуючу функцію.

Реалізовувалася програма на 64-розрядній UNIX-подібній операційній системі Manjaro Linux KDE. Основні відмінності цієї системи від популярної Windows:

– модульність - вся система являє собою набір пакетів. Пакетом, наприклад, може бути програма, бібліотека, додаткові шрифти;

– вартість - операційна система безкоштовна;

– відкритий код - операційна система, її ядро і майже все програмне забезпечення має відкритий код, який можна вільно переглядати та змінювати;

– безпека - так як операційна система має відкритий код, то його часто переглядають ентузіасти, які знаходять там помилки та виправляють їх до того, як їх використають хакери. На відміну від принципу Security By Obscurity(безпека через неясність - тобто хакери не можуть знайти помилки в коді, бо його не відкривають, проте якщо ж помилка буде знайдена, не розголошуючи, її можна використовувати достатньо довго, поки компанія-розробник нічого не знає), що

використовується у Windows, підхід відкритого коду забезпечує вищий рівень безпеки;

– розвиток - головні компоненти операційної системи розроблюються не тільки ентузіастами, але й великими компаніями, що надає набагато більші ресурси, ніж якби операційна система створювалася однією компанією. Таким чином, операційні системи на базі Linux розвиваються дуже швидко;

– продуктивність - завдяки тому, що операційні системи на базі Linux розвиваються швидко, в них використовуються найновітніші технології, через це в більшості задач такі операційні системи швидші за конкурентів.

Інтегрованим середовищем розробки було обрано Android Studio, тому що вона рекомендується Google, має офіційну підтримку від неї і спеціалізована на створенні додатків саме під цю платформу. На рисунку 1.8 зображено інтерфейс Android Studio.

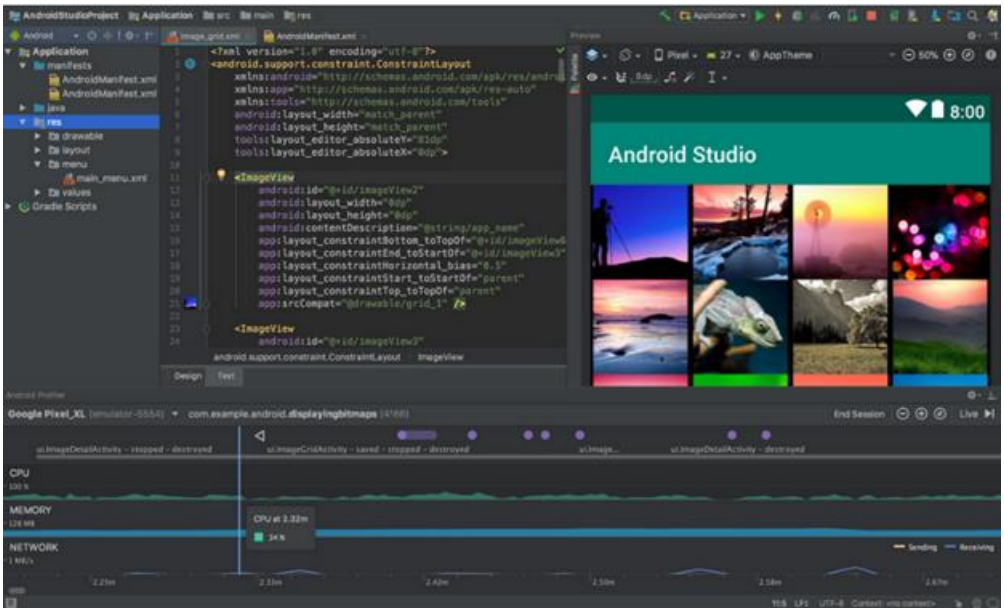


Рис. 1.8 Інтерфейс IDE Android Studio

Основою для неї є IDE IntelliJ IDEA від JetBrains, що має безкоштовну версію, підтримує розширення функціоналу через плагіни та працює на усіх основних платформах.

1.4 Вимоги до апаратного та програмного забезпечення

Мінімальні вимоги до смартфону:

1. операційна система: Android версії, не нижче 9;
2. процесор: чотирьох ядерний;
3. оперативна пам’ять: 1 гігабайт;
4. вільна пам’ять на пристрої: 100 мегабайт.

2 ПРОЄКТУВАННЯ АРХІТЕКТУРИ ТА ОПИС РЕАЛІЗАЦІЇ СИСТЕМИ

2.1 Створення та налагодження програми

Робота була створена за допомогою безкоштовного інтегрованого середовища розробки Android Studio версії 4.2.1 від Google. Воно є єдиною IDE, яка повністю підтримує весь функціонал платформи Android і оброблює специфічні для неї ситуації. Працює дане середовище на OpenJDK Java Virtual Machine 11.0.8, яке вже включено в пакет програмного забезпечення.

Спочатку було створено новий проект. Це було здійснено за допомогою навігаційного меню програми «File > New > New project...». Шаблоном проекта було обрано «Empty Activity», у якому наявна реалізація, необхідна для роботи застосування, але немає надбудов по конкретним випадкам. Базою версією API було обрано «28: Android 9.0 (Pie)», тому що вона підтримує усі нові пристрої, але при цьому дозволяє не використовувати інструменти сумісності з застарілими версіями ОС Android, що спростило подальшу розробку. На рисунку 2.1 зображено меню створення нового проекта.

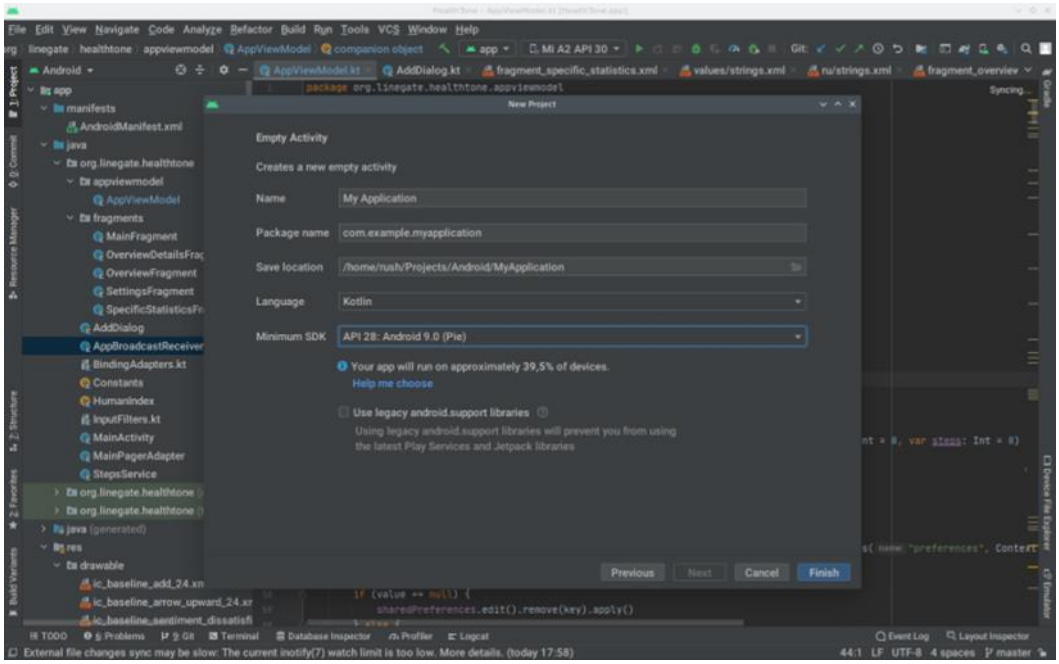


Рис. 2.1 Меню створення нового пректа

Після створення, Android Studio почне ініціалізовувати проект за допомогою системи компілювання Gradle, яка є стандартом для мов Java і Kotlin. Під час цього будуть завантажуватися усі необхідні бібліотеки і підключатимуться модулі програмного коду. На рисунку 2.2 зображено новостворений проект.

Весь вихідний код пишеться у пакетах Java, метою яких є групування реалізацій. Код самого додатку міститься у пакеті, назву якого було обрано при створенні проекту, причому явно файли вихідного коду підключати не потрібно, як це зазвичай є в мові C++ - система компіляції і IDE автоматично реагують на нові файли. Також, при необхідності можна створювати користувацькі пакети.

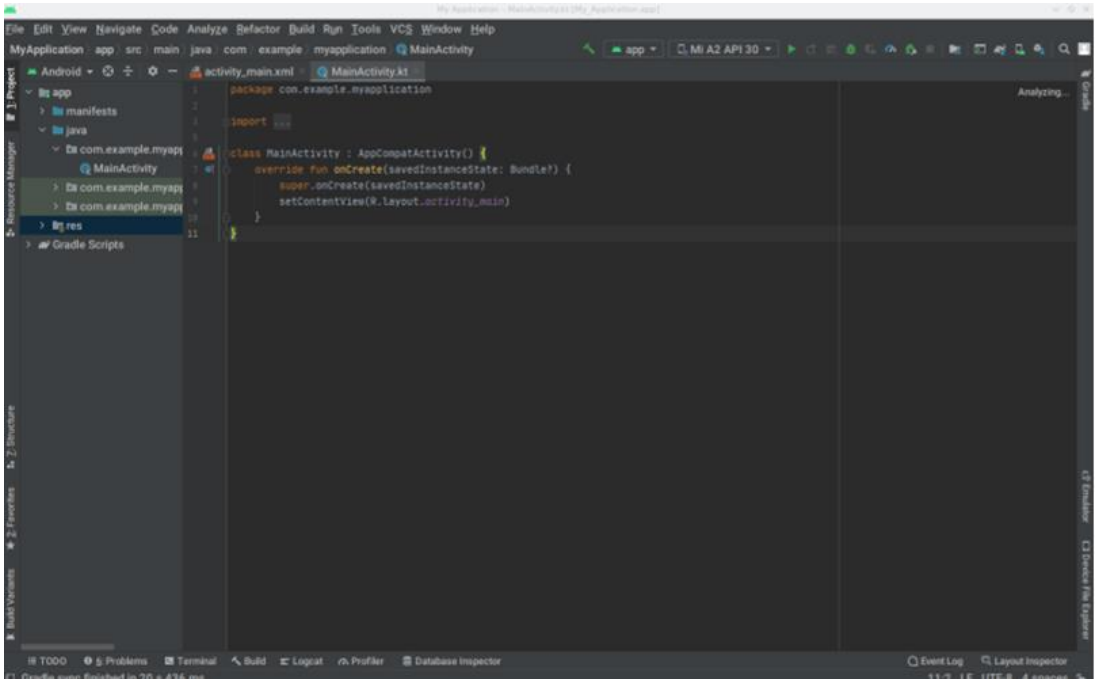


Рис. 2.2 Створений проект в Android Studio

Запуск додатку відбувається через AVD - Android Virtual Device, віртуальний пристрій Android. Фактично це віртуальна машина (на основі системи віртуалізації QEMU) з версією операційної системи, адаптованої під архітектуру X86 для більш швидкої роботи на звичайних комп’ютерах. Для використання цієї системи потрібно завантажити образ пристрію, що можна зробити в Android Studio.

Запускаючи проєкт, Gradle перевіряє усі залежності і завантажує їх у разі необхідності, після чого компілює проект і встановлює цільовий APK-файл на віртуальну машину. Також, після запуску, Android Studio може під’єднатися до

пристрою через Debug Interface, що дозволить переглядати журнал подій і явно спостерігати за роботою додатку, включаючи можливість відладки. Це саме можна виконати і зі звичайним телефоном, але потрібно буде встановити драйвер ADB. На рисунку 2.4 зображено запущений проєкт.

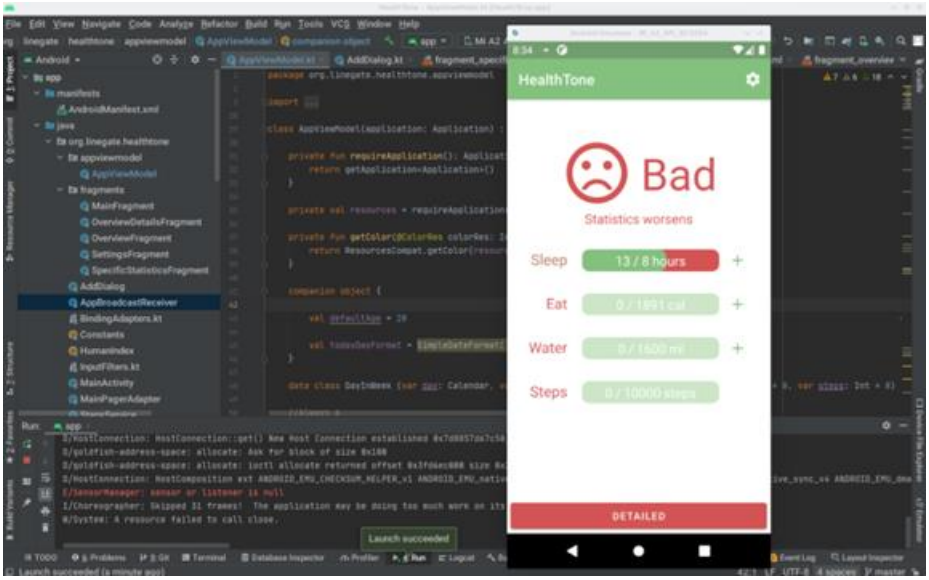


Рис. 2.4 Запущений проєкт

Робота з проєктом здійснювалася через систему контролю версій Git. Суть її полягає в тому, що проєкт в центральному сховищі репозиторія перебуває в незмінному стані і змінює його лише при завантаженні до нього коммітів - одиниць зміни вихідного коду, між якими можна вільно перемикатися. Це дозволяє зручно стежити за усіма змінами коду і вільно звертатись до минулих версій програмного забезпечення. Проте це - лише система, до якої існує багато інтерфейсів, як текстових, так і графічних. На рисунку 2.5 зображено інтерфейс популярної роботи з Git-репозиторіями GitExtensions.

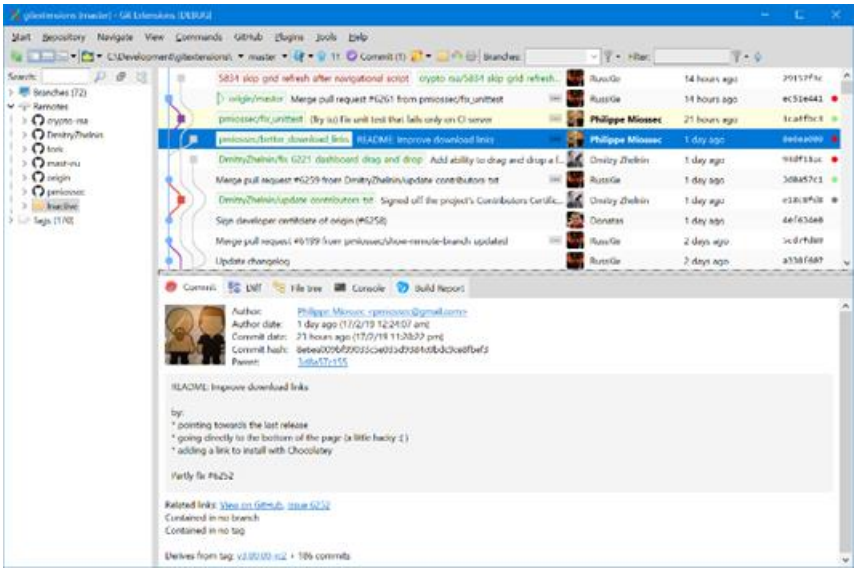


Рис. 2.5 Інтерфейс програми GitExtensions

2.2 Стратегія розробки програмного продукту

Моделлю розробки даного проєкту було обрано екстремальне програмування, як найбільш розповсюдженне на ринку розробки мобільних додатків і гнучке з точки зору внесення подальших змін, рішення.

Екстремальне програмування(XP) - це спрощений та гнучкий метод розробки програмного забезпечення, що передбачає дуже низький рівень ризику. XP характеризується такими ознаками:

3. в XP використовуються надзвичайно короткі цикли розробки ПЗ, завдяки чому забезпечується ефективний і швидкий зворотній зв’язок;

4. в XP використовується планування по наростаючій, в результаті чого загальний план виникає досить швидко, але з часом він еволюціонує;

5. XP базується на автоматичних тестах - завдяки цьому можна стежити за процесом розробки та відразу виявляти існуючі в системі дефекти;

6. XP заснована на тісному спілкуванні людей між собою.

Мобільний ринок конкурентний і швидко розвивається, тому метод екстремального програмування ідеально підійшов завдяки таким перевагам:

7. швидкість розробки - версії програмного забезпечення створюються і розгортаються якомога швидше;

8. менше приділяється уваги документації, завдяки чому розробники краще концетруються на створенні ПЗ;

9. гнучкість - модель передбачає активні зміни замовником своїх вимог;

10. ефективне використання бюджету завдяки оперативному відкиданню непотрібних елементів.

На рисунку 2.6 зображено схему розробки за моделлю XP.

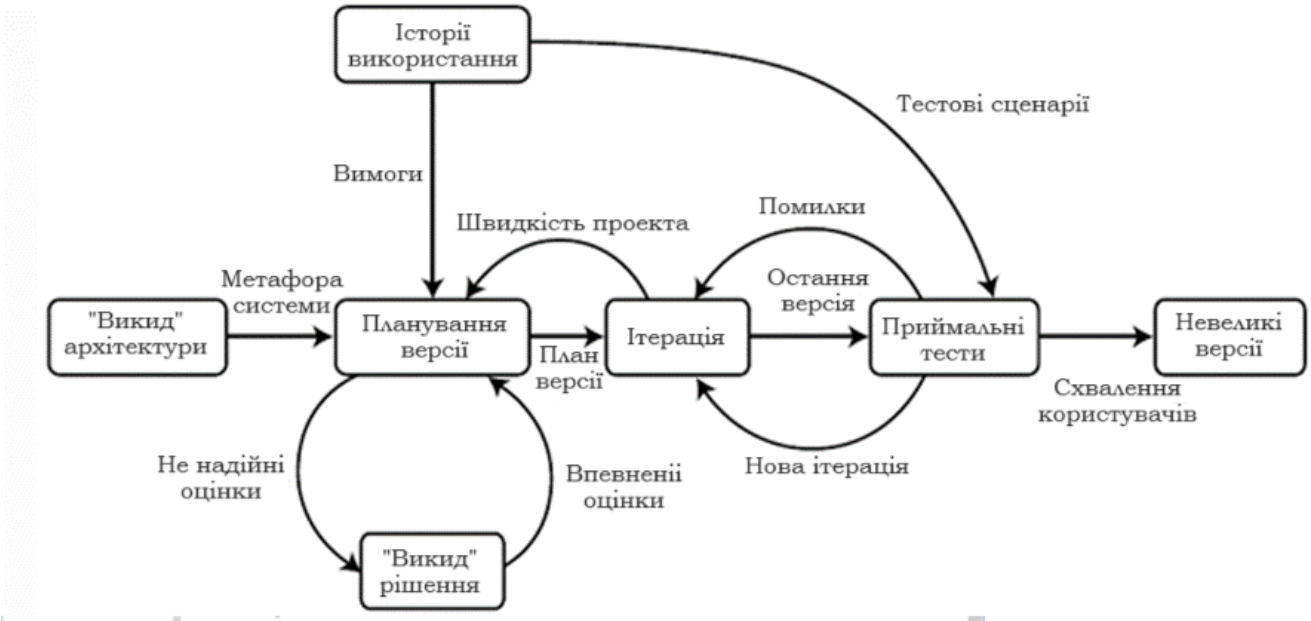


Рис. 2.6 Схема процесу розробки за моделлю XP

2.2 Опис програми та її алгоритмів

2.2.1 Опис функціоналу програми

На рисунку 2.7 зображено діаграму прецедентів програмного продукту.

Рис. 2.7 Діаграма прецедентів програмного продукту

На діаграмі перелічено такі прецеденти для користувача:

– перегляд загальної статистики по усім показникам: відкривши додаток, користувачу відображається загальна статистика по усім показникам;

– перегляд статистики по показнику "сон": користувач може подивитися статистику по показнику сну;

– перегляд статистики по показнику "їжа": користувач може подивитися статистику по показнику споживання їжі;

– перегляд статистики по показнику "вода": користувач може подивитися статистику по показнику споживання води;

– перегляд статистики по показнику "кроки": користувач може подивитися статистику по показнику кількості кроків;

– перегляд детальної статистики і історії її зміни: користувач може відкрити меню детального перегляду загальної статистики по усім показникам або по конкретному показнику, де буде відображатися історія зміни цього показника за останні дні;

– заповнення статистики: користувач може заповнити статистику по

показникам «сон», «їжа» та «вода»;

– отримання оцінки програми: користувач може переглянути оцінку програми щодо статусу ведення здорового образу життя та його зміни за останні дні;

– зміна налаштувань програми: користувач може змінити налаштування програма, вказавши свої дані;

– встановлення віку: користувач може вказати свій вік в налаштуваннях програми для більш точних розрахунків;

– встановлення зросту: користувач може вказати свій зріст в налаштуваннях програми для більш точних розрахунків;

– встановлення ваги: користувач може вказати свою вагу в налаштуваннях програми для більш точних розрахунків;

– отримання рекомендацій по заповненню параметрів: під час введення даних, користувачу надаються рекомендації по вазі та зросту на базі параметрів, що він увів.

o Окремим актором в діаграмі прецедентів є фоновий сервіс програми, який має таку функцію:

– заповнення статистики по показнику "кроки": сервіс отримує інформацію від операційної системи та записує дані у сховище даних програми.

o Також актором виступає сама операційна система Android, у якої такий перелік функцій, пов’язаних з розробленим програмним продуктом:

– читання інформації з датчиків: ОС періодично опитує датчики пристрою та опрацьовує інформацію з них;

– передача інформації фоновому сервісу програми: після опрацювання даних з датчиків, ОС надає готові результати сервісам, що потребують цю інформацію.

2.3 Опис класів

На рисунку 2.8 зображено загальну UML-діаграму класів проекту.

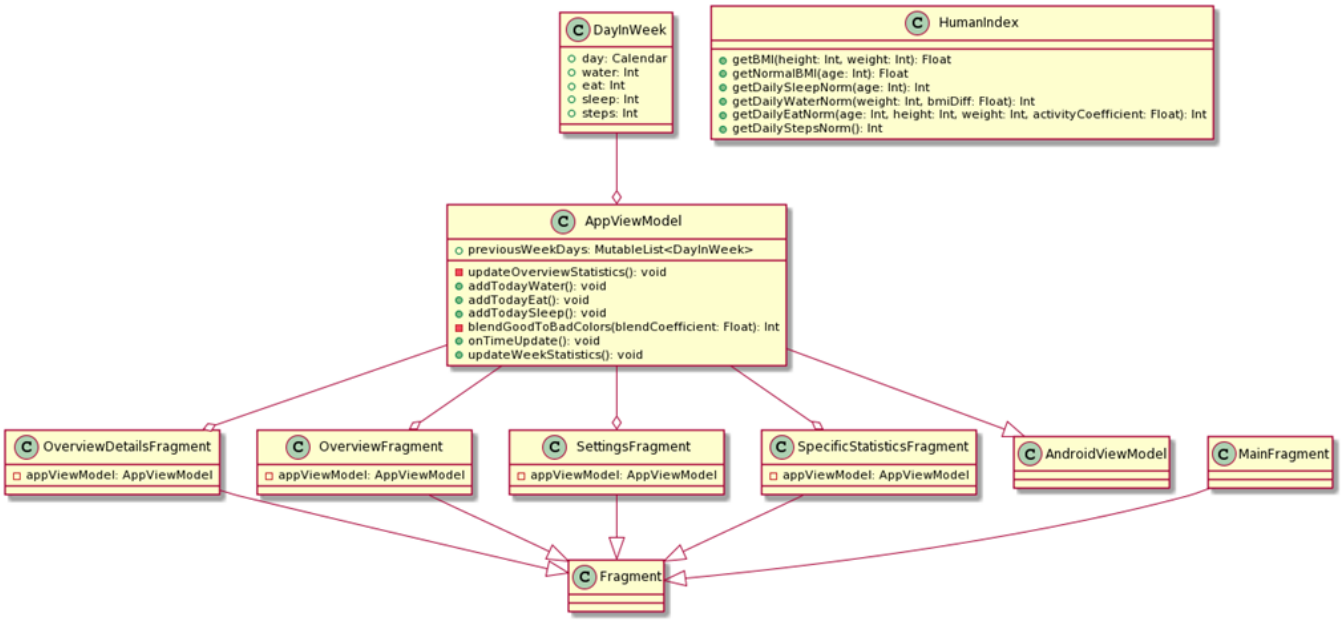


Рис. 2.8 Загальна UML діаграма класів проекту

Наведено такий перелік основних класів та їх членів:

– клас HumanIndex - інкапсулює в собі функції для роботи з індексом маси тіла, загальною статистикою по людям і розрахунок денних норм для показників сну, кількості кроків, їжі та води. Визначено такі члени:

o getBMI (height: Int, weight: Int): Floa - приймає вагу та зріст людини, після чого розраховує індекс маси тіла;

o getNormalBMI (age: Int): Float - приймає вік людини і повертає стандартне значення індексу маси тіла на основі статистики;

o getDailySleepNorm (age: Int): Int - приймає вік людини і на його основі розраховує денну норму сну у годинах;

o getDailyWaterNorm (weight: Int, bmiDiff: Float) - Int: приймає вагу людини та різницю індексу маси тіла цієї людини від норми, після чого розраховує денну норму споживання води у мілілітрах;

o getDailyEatNorm (age: Int, height: Int, weight: Int, activityCoefficient: Float) - Int: приймає вік людини, її зріст та вагу та розраховує денну норму споживання їжі в калоріях;

o getDailySteepNorm (): Int - повертає денну норму кількості кроків для людини;

– клас DayInWeek - містить у собі статистику по показникам за минулі дні. Містить такі члени:

- day: Calendar - день, за який вказано статистику;
- water: Int - об’єм випитої води у мілілітрах;
- eat: Int - спожита їжа в калоріях;
- sleep: Int - тривалість сну в годинах;
- steps: Int - кількість кроків;

– клас AndroidViewModel - шаблон, наданий операційною системою Android для того, щоб зберігати дані, пов’язані з графічними елементами;

– клас AppViewModel - зберігає інформацію, пов’язану з усіма графічними елементами в застосуванні, а також відповідає за збереження налаштувань і інших даних.

Містить такі члени:

- previousWeekDays: MutableList<DayInWeek> - сховище статистики за останні дні;
- updateOverviewStatistics () - функція оновлення даних на графічних елементах, пов’язаних з загальною статистикою;
- addTodayWater (value: Int) - функція додавання об’єму випитої води до статистики;
- addTodayEat (value: Int) - функція додавання кількості спожитої їжі до статистики;
- addTodaySleep (value: Int) - функція додавання тривалості сну до статистики;
- blendGoodToBadColors (blendCoefficient: Float) Int - функція перетворення позитивного(зеленого) кольору в негативний(червоний);
- onTimeUpdate () - фунція оновлення статистики в залежності від часу;
- updateWeekStatistics () - функція оновлення даних на графічних елементах, пов’язаних зі статистикою за минулі дні.

– клас Fragment - клас, наданий операційною системою Android для створення графічних меню;

– клас OverviewDetailsFragment - меню детальної статистики. Містить:

o appViewModel: AppViewModel - об’єкт, спільний для усього додатку, що містить налаштування графічних елементів;

– клас OverviewFragment - меню загальної статистики. Містить:

o appViewModel: AppViewModel - об’єкт, спільний для усього додатку, що містить налаштування графічних елементів;

– клас SettingsFragment - меню налаштувань. Містить:

o appViewModel: AppViewModel - об’єкт, спільний для усього додатку, що містить налаштування графічних елементів;

– клас SpecificStatisticsFragment: меню статистики по певному показнику. Містить:

o appViewModel: AppViewModel - об’єкт, спільний для усього додатку, що містить налаштування графічних елементів.

2.4 Інструкція оператора

Для того, щоб запустити програму, потрібно її встановити. Це можна зробити шляхом завантаження вихідного APK-файлу програми та його запуску на цільовому пристрої. Проте для успішного встановлення, в налаштуваннях Android потрібно встановити дозвіл на «встановлення програм з невідомих джерел». Під час першого запуску програми, вона запитає дозвіл на читання датчиків, який користувач має підтвердити, після чого має перезавантажити пристрій. На рисунку 2.19 зображено запит дозволу на читання датчиків.

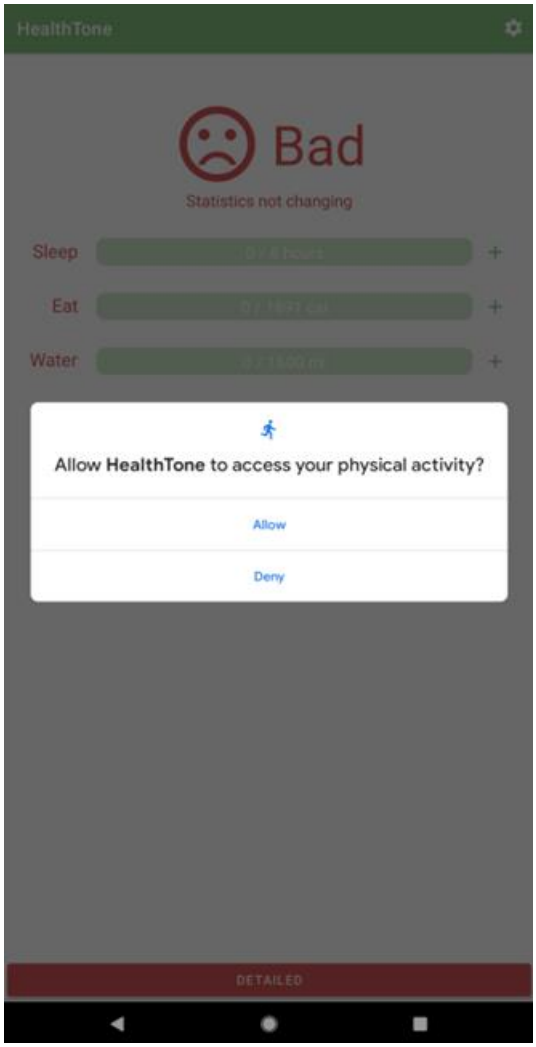


Рис. 2.19 Запит дозволу на читання датчиків

Після надання доступу, користувачу стане доступне головне меню програми, яке буде попереджувати про дуже поганий стан ведення здорового образу життя. Покращуючи статистику за останні 7 днів, користувач може побачити позитивну оцінку своїх дій. На рисунку 2.20 зображено позитивну оцінку програми.

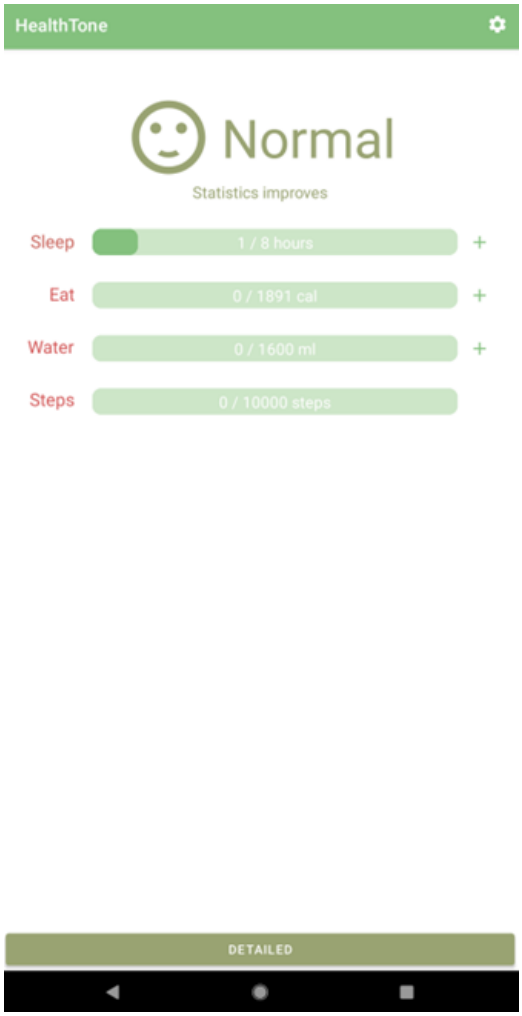


Рис. 2.20 Позитивна оцінка програми

Натиснувши на кнопку «Detailed» або «Детальніше», користувач потрапить у меню, в якому відображається загальна статистика за минулі дні. Повернутись назад можна через натискання кнопки повернення зліва зверху, або за допомогою кнопки «назад».

На головному екрані програми користувач може натиснути справа зверху на кнопку налаштувань, після чого відкриється відповідне меню, де користувач може увести свої дані. Після вводу потрібно зберегти інформацію за допомогою натискання однойменної кнопки знизу екрана. На рисунку 2.21 зображено меню налаштувань програми.

Також, на головному меню програми користувач може ввести інформацію по показникам. Для цього потрібно натиснути на кнопку «+» біля лінії показнику. Відкриється меню, в якому можна буде ввести інформацію і підтвердити дію.

Рис. 2.21 Меню налаштувань програми

Також користувач має змогу перегорнути нижче, де буде відображатися більш детальна інформація за кожним з показників. На рисунку 2.22 зображено вигляд даного меню.

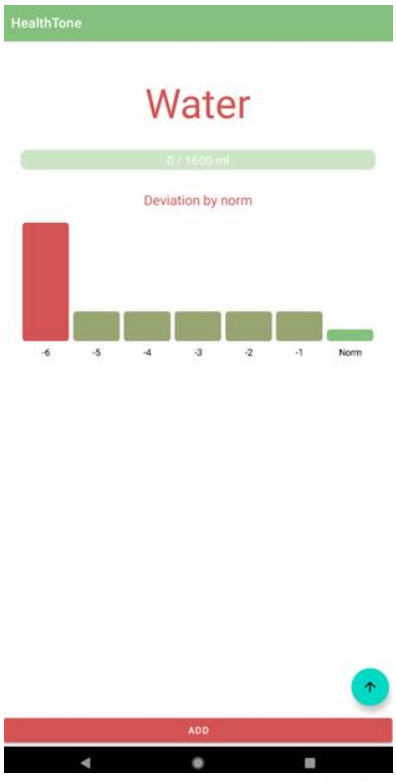


Рис. 2.22 Меню детальної інформації

3 ПРИКЛАДИ ВИКОРИСТАННЯ ТА ТЕСТУВАННЯ

3.1 Опис роботи мобільного додатку

Мобільний додаток HealthTone призначений для моніторингу основних показників здоров'я користувача, таких як сон, харчування, споживання води та фізична активність. Додаток надає можливість відстежувати ці показники щоденно, аналізувати дані та отримувати рекомендації щодо покращення загального стану здоров'я. Зараз ми детально розглянемо функціональні можливості додатку, його інтерфейс, структуру та основні компоненти. Крім того, будуть представлені скріншоти ключових екранів додатку для кращого розуміння його функціональності.

Основний екран додатку HealthTone (рис 3.1) надає користувачеві швидкий доступ до основних показників здоров'я. На цьому екрані відображається загальна статистика щодо сну, харчування, споживання води та кількості кроків, зроблених за день. У випадку, якщо якийсь з показників відхиляється від норми, додаток відображає це червоним кольором та сумним смайликом, вказуючи на погіршення статистики.

Рис. 3.1 Головний екран

Перша функція, яку користувач може змінити в додатку HealthTone є моніторинг сну (рис 3.2). Цей розділ дозволяє користувачеві вести облік кількості годин, проведених у сні. Користувач може вводити дані вручну або синхронізувати додаток з фітнес-браслетом для автоматичного зчитування інформації. У разі відхилення від норми, додаток показує графік, що відображає ці відхилення, що допомагає користувачеві зрозуміти, наскільки якісно він спить, та вчасно вносити необхідні корективи.

Рис. 3.2 Розділ моніторингу сну

Друга функція, яку користувач може змінити в додатку HealthTone є моніторинг споживання калорій протягом дня (рис. 3.3). Користувач може вводити інформацію про кожен прийом їжі, зазначаючи кількість калорій. Додаток автоматично підсумовує ці дані та відображає їх у вигляді графіка, що показує відхилення від рекомендованої норми. Це допомагає користувачеві контролювати своє харчування та підтримувати здоровий баланс калорій.

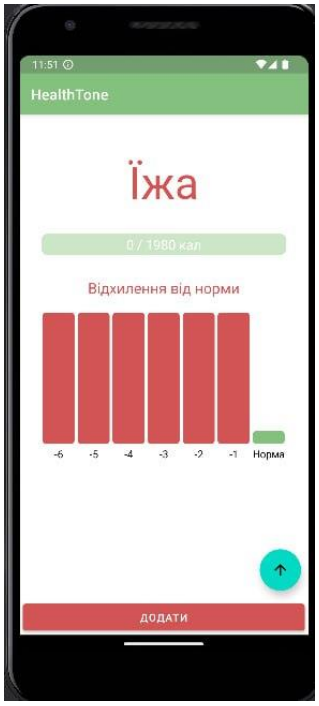


Рис. 3.3 Розділ моніторингу споживання калорій

Третя функція нашого додатку HealthTone дозволяє користувачеві вести облік випитої води, вводячи дані вручну або використовуючи смарт-пляшки, що синхронізуються з додатком (рис. 3.4). Графік у цьому розділі відображає кількість випитої води та відхилення від рекомендованої норми, що допомагає користувачеві стежити за своїм водним балансом.

Рис. 3.4 Розділ обліку випитої води

Додаток також відстежує кількість кроків, зроблених користувачем протягом дня (рис. 3.5). Ця функція може синхронізуватися з фітнес-браслетами або смартфонами, які підтримують функцію підрахунку кроків. Графік у цьому розділі показує кількість зроблених кроків та порівнює їх з рекомендованою нормою. Це стимулює користувача до активного способу життя та досягнення щоденних цілей.

Рис. 3.5 Розділ фізичної активності

Розділ налаштувань дозволяє користувачеві вводити свої персональні дані, такі як вік, зріст та вага (рис. 3.6). Ці дані використовуються додатком для розрахунку індивідуальних норм сну, споживання калорій та води, а також фізичної активності. Крім того, користувач може налаштувати сповіщення та інші параметри додатку відповідно до своїх потреб.

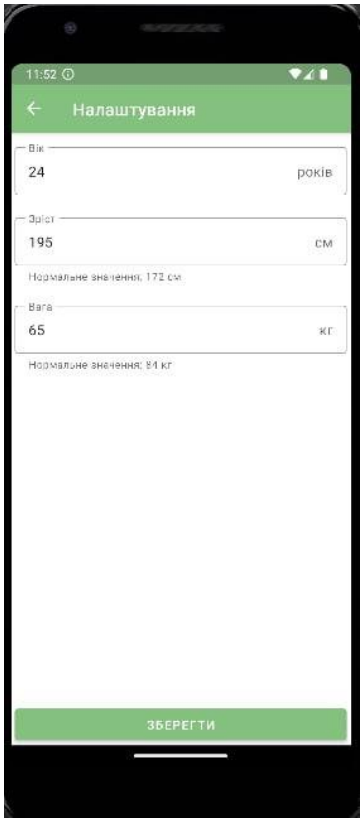


Рис. 3.6 Розділ налаштувань користувача

Також, як можна побачити зі скрін-шотів, інтерфейс користувача мобільного додатку HealthTone розроблений з урахуванням принципів зручності та інтуїтивності. Дизайн інтерфейсу виконаний у світлих тонах з використанням зрозумілих іконок та графіків. Всі основні функції додатку доступні з головного екрану, що дозволяє користувачеві швидко отримати доступ до необхідної інформації та ввести нові дані. Кожен розділ додатку має чітко структуровану навігацію, що робить процес використання додатку максимально зручним та ефективним.

3.2 Тестування мобільного додатку

Тестування нашого додатку проводилось з використанням кількох підходів, кожен з яких має свої переваги та особливості.

Основними підходами до тестування були:

1. Функціональне тестування - перевірка того, чи всі функції додатку працюють згідно з вимогами. Включає перевірку введення даних, обчислень, навігації та інших функціональних можливостей додатку.

2. Тестування зручності користування (Usability Testing) - оцінка зручності та інтуїтивності інтерфейсу додатку. Тестувальники перевіряли, наскільки легко користувачі можуть виконувати основні завдання, такі як введення даних, перегляд статистики та налаштування додатку.

3. Тестування продуктивності - наскільки ефективно працює додаток під різними навантаженнями. Включає тестування часу завантаження, швидкості обробки даних та реакції додатку на користувацькі дії.

4. Тестування на різних пристроях (Compatibility Testing) - перевірка роботи додатку на різних моделях смартфонів та версіях операційної системи Android. Це допомагає забезпечити сумісність додатку з широким спектром пристроїв.

5. Тестування споживання батареї - визначення впливу додатку на споживання батареї смартфона. Перевіряється, як робота додатку впливає на тривалість роботи пристрою від батареї та чи є додаток енергоефективним.

План тестування мобільного додатку HealthTone включав наступні етапи:

1. Підготовка тестового середовища;
2. Розробка тестових сценаріїв;
3. Виконання тестів;
4. Аналіз результатів;
5. Виправлення дефектів.

Результати тестування мобільного додатку HealthTone показали, що додаток відповідає основним функціональним вимогам та працює стабільно на більшості тестованих пристроїв. Але розглянемо більш детальніше в таблиці 3.2.

Таблиця. 3.2

Результати тестування

Підхід	Результати
Функціональне тестування	Всі основні функції додатку (введення даних про сон, харчування, воду, кроки) працюють коректно.
	Графіки та статистика відображаються правильно.
	Навігація між екранами працює без помилок.
Тестування зручності користування	Інтерфейс додатку нами визнано зручним та інтуїтивним для користувачів.
Тестування продуктивності	Додаток показав гарну продуктивність, швидкість завантаження екранів була задовільною.
	Обробка введених даних здійснюється швидко, без затримок.
Тестування на різних пристроях	Додаток працює коректно на більшості популярних моделей смартфонів з операційною системою Android.
	Виявлені незначні проблеми сумісності на деяких старих моделях пристроїв, які були виправлені у наступних версіях.
Тестування споживання батареї	Тестування показало, що додаток споживає більше батареї, ніж очікувалось, особливо під час використання функцій синхронізації з фітнес-браслетами та іншими смарт-пристроями.
	Було визначено, що це проблема, яка залишилась невирішеною на момент тестування, але її можна виправити шляхом оптимізації коду та зменшення використання ресурсів під час синхронізації та обробки даних.

4 ОРГАНІЗАЦІЙНО-ЕКОНОМІЧНА ЧАСТИНА

4.1 Оцінка вартості програмного продукту

Розробка мобільного застосунку «HealthTone» базується на детальному аналізі ринку та конкурентів. У сучасних умовах цифровізації здоров'я, ринок переповнений різноманітними додатками, що допомагають користувачам слідкувати за своїм здоров'ям. Вони пропонують функціонал для моніторингу фізичної активності, контролю за харчуванням, гідратацією та сном. Серед основних конкурентів можна виділити такі популярні програми, як Water Drink Reminder, FatSecret, StepCounter, Huawei Health та інші. Кожна з них має свої переваги та недоліки. Наприклад, Water Drink Reminder має зручний інтерфейс та функцію нагадувань, проте моніторить лише водний баланс. StepCounter від Leap Fitness Group, хоча і забезпечує відстеження фізичної активності, має обмежений функціонал та не дуже зручний інтерфейс. Huawei Health збирає дані про фізичну активність та сон, але потребує фітнес-браслета для повноцінної роботи. Проведений аналіз показав, що на ринку відсутній комплексний автономний додаток, який би одночасно стежив за всіма основними показниками здоров'я без прив'язки до додаткових пристроїв, що обумовлює актуальність розробки HealthTone.

Розробка програмного забезпечення потребує значних інтелектуальних і трудових ресурсів, а також обов'язкового використання комп'ютерної техніки, що визначає специфіку розрахунку собівартості програмного продукту. Оцінка вартості програмування включає аналіз витрат коштів і часу, необхідних для реалізації етапів проекту. Вартість програмного продукту складається з наступних компонентів:

- собівартість програмного забезпечення;
- запланований прибуток.

4.1.1 Розрахунок часу

Загальний час на створення програми складається з різних компонентів.

Структура загального часу на створення програмного продукту представлена в таблиці 4.1.

Таблиця 4.1

Структура загального часу на створення проограмного продукту

№ етапу	Позначення часу даного етапу	Зміст етапу
1	$T_{ПО}$	Підготовка опису завдання
2	$T_{О}$	Опис завдання
3	$T_{А}$	Розробка алгоритмів
4	$T_{БС}$	Розробка блок-схеми алгоритмів
5	$T_{ПІ}$	Проектування інтерфейсу
6	$T_{Н}$	Написання програми (програмування)
7	$T_{ВТ}$	Відладка і тестування програми
8	$T_{Д}$	Оформлення документації, інструкції користувачеві, записки пояснення

Час розраховується в людино-годинах, причому $T_{ПО}$ та $T_{Д}$ береться по фактично відпрацьованому часу, а час останніх етапів визначається розрахунковий по умовному числу команд Q.

Умовне число команд Q визначається за формулою

$$Q = q \square з, \qquad (4.1)$$

де q - коефіцієнт, що враховує умовне число команд залежно від типу завдання. Значення коефіцієнта q вибирається з таблиці 4.2. Значення коефіцієнта з вибирається з таблиці 4.3.

Таблиця 4.2

Значення коефіцієнту q

Тип завдання	Значення коефіцієнту q
Завдання обліку	1400...1500
Завдання оперативного управління	1500...1700
Завдання планування	3000...3500
Багатоваріантні завдання	4500...5000
Комплексні завдання	5000...5500

Для даного завдання приймається коефіцієнт q = 1450.

Програмні продукти за мірою новизни можуть бути віднесені до однієї з чотирьох груп:

– група А - розробка принципово нових завдань;

– група Б - розробка оригінальних програм;

– група В - розробка програм з використанням типових рішень;

– група Г - разове типове завдання.

Для даного завдання міра новізни: Б.

За складністю програмні продукти можуть бути віднесені до однієї з трьох груп:

– 1 - алгоритми оптимізації і моделювання систем;

– 2 - завдання обліку, звітності і статистики;

– 3 - стандартні алгоритми.

Дане завдання може бути віднесено до другої групи складності низького рівня.

Коефіцієнт (з) визначається з таблиці 4.3 в залежності від складності і міри новизни.

Таблиця 4.3

Значення коефіцієнту з

Мова програмування	Група складності	Міра новизни			
		А	Б	В	Г
Високого рівня	1	1,38	1,26	1,15	0,69
	2	1,30	1,19	1,08	0,65
	3	1,20	1,10	1,00	0,60
Низького рівня	1	1,58	1,45	1,32	0,79
	2	1,49	1,37	1,24	0,74
	3	1,38	1,26	1,15	0,69

Для даного завдання коефіцієнт з = 1,19.

За формулою 3.1 визначається умовне число команд Q.

$$Q = 1450 \text{ x } 1,19 = 1725,5 \text{ чол / годин.}$$

Визначається час, витрачений на кожен етап створення програмного продукту:

– $T_{ПО}$ (час на підготовку опису завдання), береться по факту і для даного завдання

$$ТПО = 40 \text{ чол / годин.}$$

– $T_О$ (час на опис завдання) визначається за формулою

$$ТО = Q \text{ x } В / (50 \text{ x } К) \qquad (4.2)$$

де В - коефіцієнт обліку змін завдання. Коефіцієнт В залежно від складності завдання і числа змін вибирається в інтервалі від 1,2 до 1,5. Для даного завдання В = 1,4.

К - коефіцієнт, що враховує кваліфікацію програміста. Значення коефіцієнта вибирається з таблиці 4.4.

Таблиця 4.4

Значення коефіцієнту К

Стаж програміста	Значення коефіцієнту К
до 2-х років	0,8
від 2 до 3 років	1,0
від 3 до 5 років	1,1...1,2
від 5 до 10 років	1,2...1,3
понад 10 років	1,3...1,5

В даному випадку коефіцієнт К = 1,0.

За формулою 3.2 підраховується час на опис завдання.

ТО = 1725,5 х 1,4 / (50 х 1,0) = 48,3 чол / годин.

– $Т_А$ (час на розробку алгоритмів) розраховується за формулою

ТА = Q / (50 х К) (4.3)

За формулою 3.3 підраховується час на розробку алгоритмів.

ТА = 1450 / (50 х 1,0) = 29 чол / годин.

– $Т_{БС}$ (час на розробку блок-схем) визначається аналогічно $Т_А$ за формулою 4.3 і складає

ТБС = 1450 / (50 х 1,0) = 29 чол / годин.

– $Т_{ПІ}$ (час на проектування інтерфейсу) визначається за формулою

ТПІ = Q /50 (4.4)

За формулою 4.4 підраховується час на проектування інтерфейсу.

ТПІ = 1450 / 50 = 29 чол / годин.

– $Т_Н$ (час на написання програми на мові програмування) визначається за формулою

ТН = Q х1,5 / (50 х К) (4.5)

За формулою 4.5 підраховується час на написання програми на мові програмування.

ТН = 1450 х 1,5 / (50 х 1,0) = 43,5 чол / годин.

– Твт (час відладки і тестування програми) визначається за формулою

ТВТ = Q х 4,2 / (50 х К) (4.6)

За формулою 4.6 підраховується час на відладку і тестування програми.

ТВТ = 1450 х 4,2 / (50 х 1,0) = 121,8 чол / годин.

– $Т_Д$ (час на оформлення документації), береться по факту і для даного завдання

ТД = 40 чол / годин.

Підраховується загальний час на створення програмного продукту за формулою

Т = ТПО + ТО + ТА + ТБС + ТПІ + ТН + ТВТ + ТД (4.7)

$$Т = 40 + 48,3 + 29 + 29 + 29+ 43,5 + 121,8 + 40 = 380,6 \text{ чол / годин}$$

або спрощений варіант:

$$Т = ТПІ + ТВТ + ТД \qquad (4.8)$$

$$Т = 29+ 121,8 + 40 = 190,8 \text{ чол / годин.}$$

4.1.2 Розрахунок заробітної плати виконавця робіт зі створення програмного продукту

Основна ЗП визначається за формулою

$$ЗПосн. = (ЗП\ 1р \times Кт \times Т) / (Чр \times tр.д.) \times (1 + П / 100) \qquad (4.8)$$

де: ЗП 1р - місячна зарплата 1-го розряду, грн.; Кт - тарифний коефіцієнт, відповідний розряду тарифної сітки, за яким працює виконавець; Т - загальний час на створення програмного продукту, чол. / годин; Чр - кількість робочих днів в місяць; tр.д. - тривалість робочого дня в годинах; П - відсоток премії.

Для даного завдання:

- ЗП 1р = 450,00 грн.;
- виконавець має 14 розряд;
- для 14 розряду тарифний коефіцієнт Кт = 3,36;
- загальний час створення програмного продукту Т = 380,6 чол. / годин;
- кількість робочих днів Чр = 21;
- тривалість робочого дня tр.д. = 8 годин;
- відсоток премії П=15%.

Таким чином, визначаємо основну заробітну плату виконавця робіт зі створення програмного продукту

$$ЗПосн. = (450,00 \times 3,36 \times 380,6) / (21 \times 8) \times (1 + 15 / 100) = 3929,21 \text{ грн.}$$

Додаткова заробітна плата береться у розмірі 15 % від основної і розраховується за формулою

$$ЗПдод. = ЗПосн \times П / 100 \quad (4.9)$$

$$ЗПдод. = 3929 \times 15 / 100 = 589,35 \text{ грн.}$$

Загальна заробітна плата розраховується за формулою

$$ЗПзагальна = ЗПосн. + ЗПдод. \quad (4.10)$$

$$ЗПзагальна = 3929,21 + 589,35 = 4518,56 \text{ грн.}$$

4.1.3 Розрахунок нарахувань на заробітну плату (єдиного соціального внеску)

Єдиний соціальний внесок нараховується на заробітну плату за формулою 3.11 і складає 22 % від загальної заробітної плати

$$ЄСВ = ЗПзагальна \times 22 / 100 \quad (4.11)$$

$$ЄСВ = 4518,56 \times 22 / 100 = 994,08 \text{ грн.}$$

4.1.4 Розрахунок витрат на збереження та експлуатацію ПЕОМ, що відносяться до даного програмного продукту

Витрати на збереження та експлуатацію ПЕОМ визначаються у розмірі 130% від основної заробітної плати працівників, що забезпечують функціонування ПЕОМ.

До таких витрат відносяться витрати на:

– основну заробітну плату працівників, що забезпечують функціонування ПЕОМ (інженер-електронник; системний програміст; оператор);

– основна ЗП адміністративного і допоміжного персоналу;

– амортизаційні відрахування;

– витрати на електричну енергію;

– витрати на матеріали (до їх числа входять диски, картриджі і папір для принтерів та інше);

– витрати на профілактику ПЕОМ з периферією;

– витрати на опалювання виробничих площ;

– витрати на обслуговування виробничих площ;

– інші виробничі витрати.

Розрахунок витрат на збереження та експлуатацію ПЕОМ, що відносяться до даного програмного продукту за формулою

$$\text{В зб. експл.} = \text{ЗП осн.} \square 130 / 100 \quad (4.12)$$

$$\text{В зб. експл.} = 3929{,}21 \square 130 / 100 = 5107{,}97 \text{ грн.}$$

4.2 Розрахунок собівартості програмного продукту

У собівартість програмного продукту входять наступні елементи:

– основна заробітна плата виконавця робіт із створення програмного продукту;

– додаткова заробітна плата виконавця робіт із створення програмного продукту;

– нарахування на заробітну плату (єдиний соціальний внесок);

– витрати на збереження та експлуатацію ПЕОМ, що відносяться до даного програмного продукту;

– інші витрати.

Інші витрати складають 10% від суми перших чотирьох елементів і розраховуються за формулою

$$\text{І.вит.}=(\text{ЗП осн.} + \text{ЗП дод.} + \text{ЄСВ} + \text{В зб. експл.}) \square 10 / 100 \quad (4.13)$$

І.вит.= (3929,21 + 589,35 + 994,08 + 5107,97) 10 / 100 = 1062,06 грн.

Визначається собівартість програмного продукту за формулою

Сп.п. = ЗП осн. + ЗП дод. + ЄСВ + В зб.експл. + І.вит. (4.14)

Сп.п. = 3929,21 + 589,35 + 994,08 + 5107,97 + 1062,06 = 11682,67 грн.

Структура собівартості програмного продукту відображається в таблиці 4.5.

Таблиця 4.5

Структура собівартості програмного продукту

Елементи структури	Сума грн	Питома вага %
1 Основна заробітна плата виконавця	3929,21	33,6
2 Додаткова заробітна плата виконавця	589,35	5
3 Нарахування на заробітну плату (ЄСВ)	994,08	8,5
4 Витрати збереження та експлуатацію	5107,97	43,9
5 Інші витрати	1062,06	9
Собівартість програмного продукту	11682,67	100

4.3 Розрахунок ціни програмного продукту

Ціна програмного продукту складається з декількох компонентів і розраховуються за формулою

Ц = Сп.п. + П + ПВД (4.15)

де Сп.п.- собівартість програмного продукту, грн.;

П - плановий прибуток, грн.;

ПДВ - податок на додану вартість, грн.;

П - прибуток складає 40% від повної собівартості програмного продукту і розраховується за формулою

$$П = Сп.п. \ 40 / 100 \quad (4.16)$$

$$П = 11682,67 \ 40 / 100 = 4673,07 \text{ грн.}$$

ПДВ - податок на додану вартість, який береться у розмірі 20% від суми собівартості і прибутку розраховуються за формулою

$$ПДВ = (Сп.п. + П) \ 20 / 100 \quad (4.17)$$

$$ПДВ = (11682,67 + 4673,07) \ 20 / 100 = 3271,15 \text{ грн.}$$

За формулою 3.15 визначається ціна програмного продукту.

$$Ц = 11682,67 + 4673,07 + 3271,15 = 19626,89 \text{ грн}$$

ВИСНОВКИ

У ході виконання дипломної роботи було створено Android-застосування для допомоги ведення здорового образу життя «HealthTone». Було досліджено поставлену задачу, виявлено і проаналізовано програми-конкуренти, визначено перелік необхідних вимог і перелік переваг та недоліків, за якими даний програмний продукт займає міцне місце.

Було досліджено прикладну область згідно обраній темі, визначено основний принцип роботи та детально дослідженно кожний з показників ведення здорового образу життя і встановлено стандарти, які потім використовувалися при розробці проекту.

Перед початком реалізації програмного продукту було обрано інтегроване середовище розробки та ознайомлено з ним. З причини написання застосування під нові версії операційної системи Android, у ході реалізації було використано сучасні методи розробки застосувань згідно інструкціям, курсам та документацією від компанії-розробника Google, а також було розроблено алгоритми, орієнтовані на ведення статистики. Для підвищення зручності роботи з вихідним кодом було ознайомлено з системою керування версій Git та застосовано її до даного проекту.

Графічний інтерфейс програми було спроектовано на основі досвіду використання подібних програм, за допомогою чого було впроваджено зручний та мінімалістичний для користувача інтерфейс.

Після розробки програмний продукт було протестовано та відлагоджено, що дозволило позбутися низки проблем у реалізації і покращити досвід використання.

Після реалізації програмного продукту, було досліджено основні етапи публікації власного програмного продукту на офіційній для Android платформі Play Market.

Після закінчення створення програмного продукту було описано економічно-організаційну частину і розраховано усі базові показники, включаючи собівартість і ціну, а також детально досліджено тему безпеки життєдіяльності при роботі з ЕОМ.

Також, в кінці було проведене тестування мобільного додатку, яке показало, що додаток здатний ефективно виконувати свої основні функції, включаючи моніторинг сну, харчування, споживання води та фізичної активності. Інтерфейс додатку був визнаний зручним та інтуїтивним, що полегшує його використання.

Тестування продуктивності виявило задовільну швидкість завантаження та обробки даних, а сумісність з більшістю популярних моделей смартфонів була підтверджена. Виявлені незначні проблеми сумісності на старих моделях були успішно виправлені.

Проте було зафіксовано підвищене споживання батареї, особливо під час синхронізації з фітнес-браслетами та іншими смарт-пристроями. Ця проблема може бути вирішена шляхом оптимізації коду та зменшення використання ресурсів.

Загалом, HealthTone є надійним та зручним інструментом для моніторингу здоров'я користувачів, а незначні недоліки можуть бути виправлені в наступних оновленнях додатку.

ПЕРЕЛІК ПОСИЛАНЬ

1. «UML 2 and the Unified Process Practical Object-Oriented Analysis and Design Second Edition» - Jim Arlow, Ila Neustadt.

2.«Designing Concurrent, Distributed, and Real-Time Applications with UML» Hassan Gomaa.

3.«The Unified Modeling Language User Guide» Grady Booch, James Rumbaugh, Ivar Jacobson.

4.«UNIX and Linux System Administration Handbook (5th Edition)» Evi Nemeth, Garth Snyder, Trent R. Hein, Trent R. Hein, Dan Mackin's.

5.Закон України Про оплату праці.

6.Методичні вказівки щодо виконання дипломної роботи для студентів спеціальності 121 «Інженерія програмного забезпечення».

7.Методичні рекомендації щодо виконання економічної частини дипломної роботи для студентів спеціальності 121 «Інженерія програмного забезпечення».

8.Методичні вказівки щодо виконання розділу «Охорона праці та навколишнього середовища» дипломної роботи для студентів спеціальності 121 «Інженерія програмного забезпечення».

9.Закон України Про підприємництво.

10. Закон України Про оподаткування прибутку підприємств.

11. Закон України Про охорону праці.

12. «How to Build Android Apps with Kotlin» Alex Forrester, Eran Boudjnah, Alexandru Dumbravan, Jomar Tigcal.

13. «Android Programming for Beginners - Third Edition» John Horton.

14. «Learn Kotlin Programming - Second Edition» Stephen Samuel, Stefan Bocutiu.

15. «Android System Programming» Roger Ye.

16. «Android Programming with Kotlin for Beginners» John Horton.

17. «Android 9 Development Cookbook - Third Edition» Rick Boyer.

18. «Hands-On Android UI Development» Jason Morris.

19. «Building Android UIs with Custom Views» Raimon Ràfols Montané.

20. «Kotlin Programming Cookbook» Aanand Shekhar Roy, Rashi Karanpuria.

21. «Asynchronous Android Programming - Second Edition» Helder Vasconcelos.

22. «Android Design Patterns and Best Practice» Kyle Mew.

23. «Reactive Programming in Kotlin» Rivu Chakraborty.

24. «Mastering Android Development with Kotlin» Miloš Vasić.

25. «Learning Material Design» Kyle Mew.

26. «Android Studio Cookbook» Mike van Drongelen.

27. «Android Development with Kotlin» Marcin Moskala, Igor Wojda.

28. «Mastering Android Application Development» Antonio Pachón Ruiz.

29. «Creating Dynamic UIs with Android Fragments - Second Edition» Jim Wilson.

30. «Android Sensor Programming By Example» Varun Nagpal.

31. «Learning Android Application Development» Raimon Ràfols Montané, Laurence Dawson.

32. «Android Application Development Cookbook - Second Edition» Rick Boyer, Kyle Mew.

33. «The Complete Edition - Software Engineering for Real-Time Systems» Jim Cooling.

34. «UML 2.0 in Action: A project-based tutorial» Henriette Baumann, Patrick Grassle, Philippe Baumann.

35. «GitLab Quick Start Guide» Adam O'Grady.

36. «Mastering GitLab 12» Joost Evertse.

37. «Git Version Control Cookbook - Second Edition» Kenneth Geisshirt, Emanuele Zattin, Aske Olsson, Rasmus Voss.

38. «Mastering Git» Jakub Narębski.

39. «GitHub Essentials - Second Edition» Achilleas Pipinellis.

40. «GitLab Cookbook» Jeroen van Baarsen.

41. «Practical Remote Pair Programming» Adrian Bolboacă.

42. «Version Control with Git and GitHub» Alex Magana, Joseph Muli, Sahil Malik.

43. «Git Best Practices Guide» Eric Pidoux.

44. «Git: Version Control Made Easy» Achilleas Pipinellis, Aske Olsson, Ferdinando Santacroce.

45. «Mastering Linux Security and Hardening - Second Edition» Donald A. Tevault.

46. «Mastering Linux Shell Scripting - Second Edition» Mokhtar Ebrahim, Andrew Mallett.

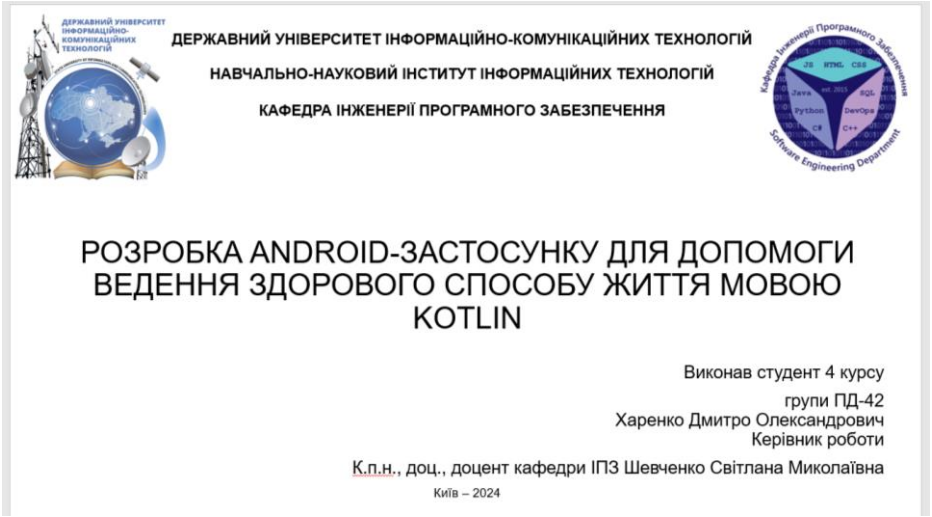
47. «Linux Shell Scripting Cookbook - Third Edition» Clif Flynt, Sarath Lakshman, Shantanu Tushar.

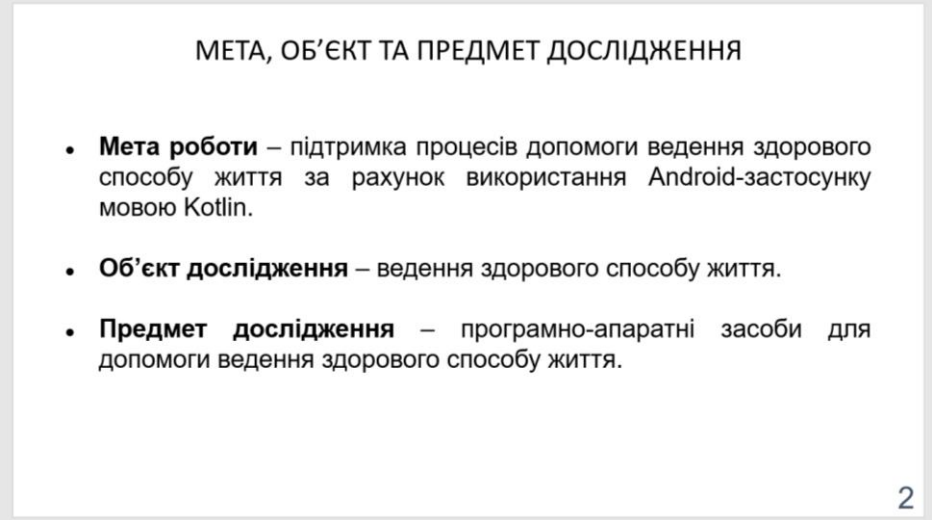
48. «SELinux System Administration - Third Edition» Sven Vermeulen.

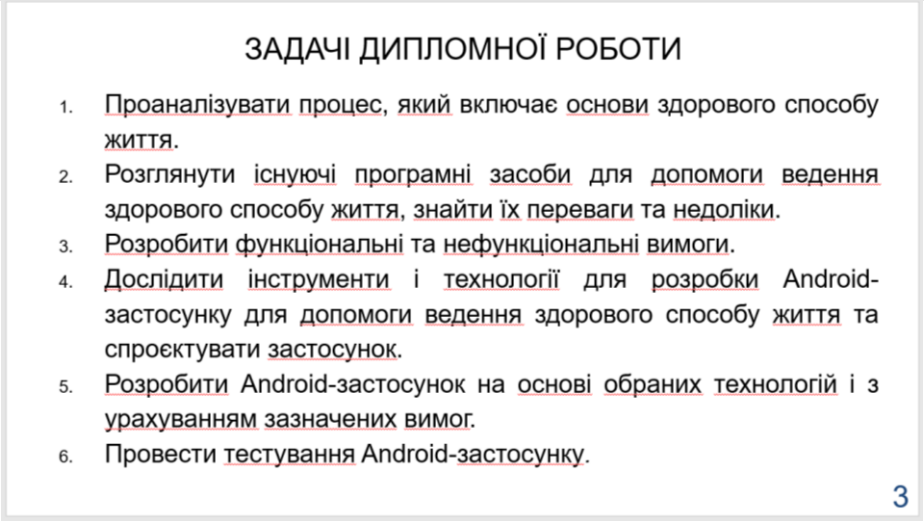
49. «Linux Administration Cookbook» Adam K. Dean.

50. «Android Studio 4.1 Development Essentials – Kotlin Edition» Neil Smyth.

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ (ПРЕЗЕНТАЦІЯ)







АНАЛІЗ АНАЛОГІВ

Назва	Water Reminder	Step Tracker	Huawei Health	Health Tone
Моніторинг сну	-	-	+	+
Моніторинг якості харчування	-	-	-	+
Моніторинг водного балансу	+	-	-	+
Моніторинг фізичних навантажень	-	+	+	+
Спрощений вхід в додаток без облікового запису	-	-	+	+

4

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги:

1. Можливість відображення короткої статистики за усім показниками.
2. Можливість відображення детальної статистики за кожним з показників та її аналіз.
3. Можливість автоматичного заповнення статистики.
4. Можливість заповнення статистики вручну.

Нефункціональні вимоги:

1. Збір інформації про кроки має бути через датчик кроків.
2. Статистика кроків у додатку працює у фоні та запам'ятовує у фоні.

5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

Kotlin

android studio

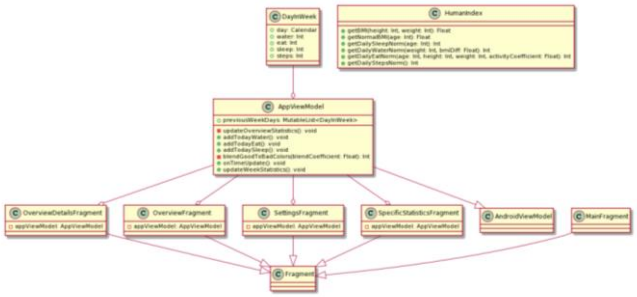
Java

6

ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ

7

ДІАГРАМА КЛАСІВ



8

ЕКРАННІ ФОРМИ

ГОЛОВНИЙ ЕКРАН

ЗАГАЛЬНА СТАТИСТИКА

НАЛАШТУВАННЯ

ВІДЕО ПРЕДСТАВЛЕННЯ

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Харенко Д.О. Розробка Android-застосунку для підтримки здорового способу життя. *Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях»*. 24 квітня 2024р., Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2024. С. 79

11

ВИСНОВКИ

1. Проведено дослідження прикладної області відповідно до обраної теми. Визначено основний принцип роботи та детально проаналізовано кожен із показників здорового способу життя.
2. Розглянуто існуючі програмні засоби для допомоги ведення здорового способу життя: Water Reminder, Health Tracker, Huawei Health, визначено їх переваги та недоліки.
3. Розроблено функціональні та нефункціональні вимоги до застосунку.
4. Досліджено інструменти і технології для розробки Android-застосунку та спроєктовано сам застосунок. Розглянуто наступні технології: мова Kotlin для написання логіки та розмітки додатку; Android Studio як середовище для розробки; Java як віртуальна машина на поверхні JVM.
5. Розроблено Android-застосунок для допомоги ведення здорового способу життя мовою Kotlin на основі обраних технологій і з урахуванням зазначених вимог.
6. Проведено тестування Android-застосунку. Застосунок відповідає зазначеним вимогам.

12

ДОДАТОК Б. ЛІСТИНГИ ОСНОВНИХ МОДУЛІВ

HumanIndex.kt

```
package org.linegate.healthtone

import kotlin.math.roundToInt
import kotlin.math.sqrt

object HumanIndex {

    val normalBMIDeviation = 3f

    /**
     * @param[personBMI] Person's BMI.
     * @param[normalBMI] Normal BMI for person.
     * @return BMI difference.
     */
    fun getBMIDiff(personBMI: Float, normalBMI: Float): Float {
        return personBMI - normalBMI
    }

    /**
     * @param[height] Person's height in cm.
     * @param[weight] Person's weight in cm.
     * @return Calculated person's BMI.
     */
    fun getBMI(height: Int, weight: Int): Float {
        return weight / Math.pow(height / 100.0, 2.0).toFloat()
    }

    /**
     * @param[age] Person's age.
     * @return Standart body mass index for specified age.
     */
    fun getNormalBMI(age: Int) : Float {
        if (age <= 8) {
            return 16f
        } else if (age >= 20) {
            return 22f
        } else {
            return 16f + (age - 8f) * 0.5f
        }
    }

    /**
     * @param[age] Person's age.
     * @param[weight] Person's weight for more accuracy calculation with BMI in kg. If not specified, standart height is used.
     * @return Calculated person's height in cm.
     */
    fun getNormalHeight(age: Int, weight: Int? = null) : Int {
        weight?.let {
            return (sqrt(weight / getNormalBMI(age)) * 100).roundToInt()
        }
        if (age >= 20) {
            return 170
        } else {
            return (80 + (age - 1) * ((170 - 80) / 19f)).roundToInt()
        }
    }

    /**
     * @param[age] Person's age.
     * @param[height] Person's height for more accuracy calculation with BMI in cm. If not specified, standart weight is used.
     * @return Calculated person's weight in kg.
     */
    fun getNormalWeight(age: Int, height: Int? = null) : Int {
        height?.let {
            return (getNormalBMI(age) * Math.pow(height / 100.0, 2.0)).roundToInt()
        }
        return (getNormalBMI(age) * Math.pow(getNormalHeight(age) / 100.0, 2.0)).roundToInt()
    }

    /**
     * @return Daily steps count norm for person.
     */
    fun getDailyStepsNorm(): Int {
        return 10_000
    }

    /**
     * @param[stepsCount] Person's steps count.
     * @return Person's activity coefficient based on steps count.
     */
    fun getActivityCoefficientBySteps(stepsCount: Int): Float {
```

```
            return 1.45f + (stepsCount - getDailyStepsNorm()) * 0.000025f
        }

        /**
         * @param[age] Person's age.
         * @return Calculated daily sleep norm for person in hours.
         */
        fun getDailySleepNorm(age: Int): Int {
            if (age <= 12) {
                return 10
            } else if (age <= 18) {
                return 9
            } else if (age <= 40) {
                return 8
            } else {
                return 7
            }
        }

        /**
         * @param[weight] Person's weight for calculation in kg.
         * @param[bmiDiff] Difference between person's BMI and standart BMI for his age.
         * @return Calculated daily water norm for person in ml.
         */
        fun getDailyWaterNorm(weight: Int, bmiDiff: Float): Int {
            if (bmiDiff > normalBMIDeviation) {
                return weight * 30
            }
            return weight * 25
        }

        /**
         * @param[age] Person's age.
         * @param[height] Person's height in cm.
         * @param[weight] Person's weight in kg.
         * @param[activityCoefficient] Person's activity between 1.2(no activity) and 1.9(professional sport). Standart - 1.45.
         * @return Calculated person's daily eat norm in cal.
         */
        fun getDailyEatNorm(age: Int, height: Int, weight: Int, activityCoefficient: Float): Int {
            return ((360.785 + 11.675 * weight + 3.425 * height - 5.715 * age) * activityCoefficient).roundToInt()
        }
    }
```

BindingAdapters.kt

```
package org.linegate.healthtone

import android.util.Log
import android.widget.Button
import android.widget.ImageView
import androidx.core.view.doOnDetach
import androidx.core.view.doOnPreDraw
import androidx.databinding.BindingAdapter
import com.db.williamchart.data.Scale
import com.db.williamchart.view.BarChartView
import com.db.williamchart.view.HorizontalBarChartView

@BindingAdapter("imageFromResource")
fun bindImageFromResource(imageView: ImageView, imageResource: Int?) {
    imageResource?.let {
        imageView.setImageResource(imageResource)
    }
}

@BindingAdapter("color")
fun bindColor(imageView: ImageView, color: Int?) {
    color?.let {
        imageView.setColorFilter(color)
    }
}

@BindingAdapter("color")
fun bindColor(button: Button, color: Int?) {
    color?.let {
        button.setBackgroundColor(color)
    }
}

@BindingAdapter("color")
fun bindColor(horizontalBarChartView: HorizontalBarChartView, color: Int?) {
    color?.let {
        horizontalBarChartView.resetPivot()
        horizontalBarChartView.barsColor = color
    }
}

@BindingAdapter("backgroundColor")
fun bindBackgroundColor(horizontalBarChartView: HorizontalBarChartView, color: Int?) {
    color?.let {
        horizontalBarChartView.resetPivot()
        horizontalBarChartView.barsColor = color
        horizontalBarChartView.scale = Scale(0f, 1f)

        horizontalBarChartView.show(listOf("" to 0f, "" to 1f))
    }
}

@BindingAdapter("bar")
fun bindBar(horizontalBarChartView: HorizontalBarChartView, bar: Pair<Float, Float>?) {
    bar?.let {
```

```
                horizontalBarChartView.scale = Scale(0f, bar.second)

horizontalBarChartView.show(listOf("" to 0f, "" to bar.first))
            }
        }

        @BindingAdapter("bars")
        fun bindBars(chartView: BarChartView, bars: List<Pair<String, Float>>?) {
            bars?.let {
                //chartView.scale = Scale(0f, bars.last().second)
                chartView.animate(bars)
            }
        }

        @BindingAdapter("colors")
        fun bindColors(chartView: BarChartView, colors: List<Int>?) {
            colors?.let {
                chartView.resetPivot()
                chartView.barsColorsList = colors
            }
        }

        @BindingAdapter("barsBackgroundColor")
        fun bindBarsBackgroundColor(chartView: BarChartView, color: Int?) {
            color?.let {
                chartView.resetPivot()
                chartView.barsBackgroundColor = color
            }
        }
```

StepsService.kt

```
        package org.linegate.healthtone

        import android.app.*
        import android.content.Context
        import android.content.Intent
        import android.hardware.Sensor
        import android.hardware.SensorEvent
        import android.hardware.SensorEventListener2
        import android.hardware.SensorManager
        import android.os.Binder
        import android.os.IBinder
        import android.util.Log
        import android.widget.Toast
        import androidx.core.app.NotificationCompat
        import androidx.lifecycle.ViewModelProvider
        import org.linegate.healthtone.appviewmodel.AppViewModel
        import kotlin.math.roundToInt

        class StepsService : Service(), SensorEventListener2 {

            companion object {
                var isRunning = false
            }

            inner class StepsServiceBinder : Binder() {
                fun getService(): StepsService = this@StepsService
            }

            private lateinit var sensorManager: SensorManager

            private var previousSteps = 0

            private var firstStepCount = true

            val binder = StepsServiceBinder()

            var appViewModel: AppViewModel? = null

            private fun requireViewModel(): AppViewModel {
                return appViewModel!!
            }

            fun createViewModel() {
                appViewModel = ViewModelProvider.AndroidViewModelFactory.getInstance(application).create(AppViewModel::class.java)
            }

            override fun onCreate() {
                super.onCreate()

                createViewModel()

                sensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
                val stepsSensor = sensorManager.getDefaultSensor(Sensor.TYPE_STEP_COUNTER)
                sensorManager.registerListener(this, stepsSensor, SensorManager.SENSOR_DELAY_FASTEST)

                isRunning = true
            }

            override fun onBind(intent: Intent?): IBinder? {
                return binder
            }

            override fun onStartCommand(intent: Intent?, flags: Int, startId: Int): Int {
                when (intent?.action) {
```

```
Constants.actionStartForegroundService -> {
                val channel = NotificationChannel(Constants.foregroundNotificationChannelID, Constants.foregroundNotificationChannelName, NotificationManager.IMPORTANCE_LOW)

(getSystemService(Context.NOTIFICATION_SERVICE) as NotificationManager).createNotificationChannel(channel)

                val notificationIntent = Intent(this, MainActivity::class.java)
                notificationIntent.action = MainActivity::class.java.name

notificationIntent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK or Intent.FLAG_ACTIVITY_CLEAR_TASK)
                val notificationPendingIntent = PendingIntent.getActivity(this, 0, notificationIntent, 0)
                startForeground(1, NotificationCompat.Builder(this, Constants.foregroundNotificationChannelID)
                    .setContentTitle(getString(R.string.s_in_background, getString(R.string.app_name)))
                    .setSmallIcon(R.drawable.ic_launcher_foreground)
                    .setContentIntent(notificationPendingIntent)
                    .build())
            }
        }
        return START_STICKY
    }

    override fun onSensorChanged(event: SensorEvent?) {
        val newSteps = event!!.values[0].roundToInt()
        if (!firstStepCount) {

requireViewModel().onTimeUpdate()

requireViewModel().addTodaySteps(newSteps - previousSteps)
        } else {
            firstStepCount = false
        }
        previousSteps = newSteps
    }

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

    }

    override fun onFlushCompleted(sensor: Sensor?) {
    }

    override fun onDestroy() {
        isRunning = false
        super.onDestroy()
    }
}
```

MainActivity.kt

```
package org.linegate.healthtone

import android.Manifest
import android.content.*
import android.content.pm.PackageManager
import android.os.Bundle
import android.os.IBinder
import android.util.Log
import android.widget.Toast
import androidx.activity.viewModels
import androidx.appcompat.app.AppCompatActivity
import androidx.core.app.ActivityCompat
import androidx.core.content.ContextCompat
import androidx.navigation.NavController
import androidx.navigation.fragment.NavHostFragment
import androidx.navigation.ui.AppBarConfiguration
import androidx.navigation.ui.setupActionBarWithNavController
import org.linegate.healthtone.appviewmodel.AppViewModel
import org.linegate.healthtone.databinding.ActivityMainBinding


class MainActivity : AppCompatActivity() {

    private val appViewModel: AppViewModel by viewModels()

    private lateinit var stepsService: StepsService

    private lateinit var binding: ActivityMainBinding

    private lateinit var navController: NavController

    private val dayChangedBroadcastReceiver = object : BroadcastReceiver() {
        val intentFiler = IntentFilter().apply {

addAction(Intent.ACTION_TIME_TICK)
```

```
addAction(Intent.ACTION_TIMEZONE_CHANGED)

addAction(Intent.ACTION_TIME_CHANGED)

addAction(Intent.ACTION_DATE_CHANGED)
                }

                override fun onReceive(context: Context?, intent: Intent?) {
                    if (intent?.action == Intent.ACTION_TIME_TICK || intent?.action == Intent.ACTION_TIMEZONE_CHANGED || intent?.action == Intent.ACTION_TIME_CHANGED) {
                        appViewModel.onTimeUpdate()
                    }
                }
            }

            override fun onCreate(savedInstanceState: Bundle?) {
                super.onCreate(savedInstanceState)
                binding = ActivityMainBinding.inflate(layoutInflater)
                setContentView(binding.root)

                val navHostFragment = supportFragmentManager.findFragmentById(R.id.nav_host_fragment) as NavHostFragment
                navController = navHostFragment.navController
                val appBarConfiguration = AppBarConfiguration.Builder(R.id.mainFragment).build()

setupActionBarWithNavController(navController, appBarConfiguration)

                if (ContextCompat.checkSelfPermission(this, Manifest.permission.ACTIVITY_RECOGNITION) != PackageManager.PERMISSION_GRANTED) {

ActivityCompat.requestPermissions(this, arrayOf(Manifest.permission.ACTIVITY_RECOGNITION), 0)
                }

                val stepsServiceIntent = Intent(applicationContext, StepsService::class.java)

                //startService(stepsServiceIntent)
                if (!StepsService.isRunning) {
                    stepsServiceIntent.action = Constants.actionStartForegroundService

startForegroundService(stepsServiceIntent)
                }

                bindService(stepsServiceIntent, object: ServiceConnection {
                    override fun onServiceConnected(className: ComponentName, binder: IBinder) {
                        stepsService = (binder as StepsService.StepsServiceBinder).getService()
                        stepsService.appViewModel = appViewModel
                    }

                    override fun onServiceDisconnected(name: ComponentName?) {

                    }
                }, BIND_AUTO_CREATE)


registerReceiver(dayChangedBroadcastReceiver, dayChangedBroadcastReceiver.intentFiler)
            }

            override fun onDestroy() {

unregisterReceiver(dayChangedBroadcastReceiver)
                stepsService.createViewModel()
                super.onDestroy()
            }

            override fun onSupportNavigateUp(): Boolean {
                return navController.navigateUp() || super.onSupportNavigateUp()
            }
        }
```

MainPagerAdapter.kt

```
package org.linegate.healthtone

import androidx.fragment.app.Fragment
import androidx.viewpager2.adapter.FragmentStateAdapter

class MainPagerAdapter(fragment: Fragment) : FragmentStateAdapter(fragment) {

    private var fragments = mutableListOf<Fragment>()

    fun addFragment(fragment: Fragment) {
        fragments.add(fragment)
    }

    override fun getItemCount(): Int {
        return fragments.size
    }

    override fun createFragment(position: Int): Fragment {
        return fragments[position]
    }

}
```