

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

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

Кафедра програмних засобів
(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

бакалавр

(ступінь вищої освіти)

на тему РОЗРОБКА МОБІЛЬНОГО ЗАСТОСУНКУ ШАГОМІРУ
DEVELOPMENT OF A PEDOMETER MOBILE APPLICATION

Виконав(ла): студент(ка) 4 курсу, групи КНТ-139
Спеціальності 121 Інженерія програмного
забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

РУДЬ О.К.

(ПРИЗВИЩЕ та ініціали)

Керівник ФЕДОРОНЧАК Т.В.

(ПРИЗВИЩЕ та ініціали)

Рецензент ШИТКОВА О.В.

(ПРИЗВИЩЕ та ініціали)

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

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

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
“ ” 2023 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

РУДЬ Орини Костянтинівни

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Розробка мобільного застосунку шагоміру
Development of a Pedometer Mobile Application

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

затверджені наказом закладом вищої освіти від “04” квітня 2023 року № 87

2. Строк подання студентом проєкту (роботи) 23 травня 2023

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

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз проблеми та постановка завдань дослідження.
2. Матеріали і методи. 3. Опис програми. 4. Експлуатація, тестування та експериментальне дослідження програми.

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

Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	ФЕДОРОНЧАК Т. В., доцент		
Нормоконтроль	ДЕЙНЕГА Л. Ю., ст. викладач		

7. Дата видачі завдання « 13 » лютого 2023 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	1 тиждень	Розділ 1
3	Вибір мови програмування та інших технологій розробки.	2 тиждень	Розділ 2
4	Розробка архітектури програми.	2 тиждень	Розділ 3
5	Розробка програми.	3-4 тижні	Розділ 3, 4
6	Тестування та експериментальне дослідження програмного забезпечення.	5 тиждень	Розділ 4
7	Оформлення пояснювальної записки та документів до неї.	6 тиждень	Додатки
8	Нормоконтроль та рецензування.	7 тиждень	
9	Захист роботи.	8 тиждень	

Студент(ка)

(підпис) Орина РУДЬ
(Ім'я ПРІЗВИЩЕ)

Керівник проєкту (роботи)

(підпис) Тетяна ФЕДОРОНЧАК
(Ім'я ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 143с., 33 рис., 13 табл., 3 дод., 20 джерел.

МОБІЛЬНИЙ ЗАСТОСУНОК, ІНТЕРАКТИВНА ГРА, АКТИВНІСТЬ, ПІДВИЩЕННЯ СЕРЕДНЬОДОБОВОЇ АКТИВНОСТІ, HEALTH CONNECT.

Об'єкт дослідження – мобільні застосунки для відслідковування активності користувача.

Предмет дослідження - мобільні застосунки для відслідковування кількості кроків.

Мета роботи – створення мобільного застосунку для підвищення середньодобової активності та заохочення ходити більше.

Матеріали, методи та технічні засоби: об'єктно-орієнтоване програмування, мова програмування Kotlin, мобільний пристрій з операційною системою Android.

Результати. Створено мобільний застосунок, який у формі гри заохочує користувача підвищити середньодобову активність та підвищує його інтерес до цього.

Висновки. Розроблено мобільний застосунок для підвищення середньодобової активності користувача.

Галузь використання – особисте використання для відслідковування персональної добової активності.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 143 pages, 33 figures, 13 tables, 3 appendixes, 20 sources.

MOBILE APP, INTERACTIVE GAME, ACTIVITY, INCREASE AVERAGE DAILY ACTIVITY, HEALTH CONNECT.

Object of study is mobile applications for tracking user activity.

Subject of study is mobile applications for tracking the number of steps.

The purpose of the work is to create a mobile application to increase average daily activity and encourage walking more.

Materials, methods and tools: object-oriented programming, Kotlin programming language, mobile device with the Android operating system.

Results. A mobile application has been created, which in the form of a game encourages the user to increase the average daily activity and increases his interest to this.

Conclusions. A mobile application has been developed to increase the average daily activity of the user.

The field of use is personal use for tracking personal daily activity.

ЗМІСТ

	С.
Перелік скорочень та умовних позначок	8
Вступ	9
1 Аналіз предметної області	10
1.1 Актуальність теми розробки	10
1.2 Загальні поняття	11
1.2.1 Android SDK.....	11
1.2.2 Health Connect API.....	12
1.2.3 Розробка мобільних застосунків для Android з використанням Kotlin	13
1.3 Дослідження програмних засобів для відстеження кроків	14
1.3.1 Google Fit.....	14
1.3.2 StepsApp	15
1.4 Постановка завдання роботи.....	16
2 Опис програми.....	18
2.1 Порівняльний аналіз мов програмування	18
2.1.1 Kotlin	18
2.1.2 Java	19
2.1.3 Порівняння мов програмування Kotlin та Java	20
2.2 Порівняльний аналіз архітектури програми	24
2.2.1 MVP	24
2.2.2 MVVM.....	26
2.2.3 Порівняння архітектур	28
2.3 Порівняльний аналіз методів підрахування кроків	30

2.4 Порівняльний аналіз ін'єкції залежностей	33
2.4.1 Hilt	33
2.4.2 Koin.....	34
2.4.3 Порівняння ін'єкції залежностей	34
2.5 Висновок з розділу 2	37
3 Розробка програми	39
3.1 Структурна схема програми	39
3.2 Файли, методи, та функції	40
3.3 Використання HealthConnect API.....	47
4 Експлуатація, тестування та експериментальне дослідження програми.....	50
4.1 Призначення й умови застосування програми.....	50
4.2 Опис програми.....	51
4.2.1 Загальні відомості.....	51
4.2.2 Функціональне призначення	51
4.2.3 Опис логічної структури	52
4.2.4 Використані технічні засоби	52
4.2.5 Виклик та завантаження.....	52
4.3 Інтерфейс	53
Висновки.....	63
Перелік джерел посилання	64
Додаток А Технічне завдання.....	66
Додаток Б Текст програми.....	70
Додаток В Слайди презентації	137

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API	– Application Programming Interface;
IDE	– Integrated Development Environment;
ADT	– Android Development Tools;
JDK	– Java Development Kit;
SDK	– Software development Kit;
CPU	– Central Processing Unit;
JVM	– Java Virtual Machine;
DI	– Dependency Injection;
DSL	– Domain-Specific Language;
ПЗ	– Програмне забезпечення;
ОС	– Операційна система.

ВСТУП

В сучасному світі здоровий спосіб життя та фізична активність набувають все більшої популярності. Ходьба - одна з найбільш доступних та простих форм фізичної активності. У зв'язку з цим, розробка мобільного застосунку шагоміру може стати відмінним рішенням для зручного відстеження кількості кроків та ефективного заохочення до цього користувачів.

Метою даної дипломної роботи є розробка мобільного застосунку шагоміру, який дозволить користувачам відстежувати кількість кроків, пройдених впродовж дня та піклуватися про свого віртуального друга. Застосунок буде розроблено для платформи Android з використанням мови програмування Kotlin, а також застосуванням фреймворка Android Studio.

У рамках дипломної роботи будуть виконані такі завдання: розроблення інтерфейсу користувача, розробка архітектури проєкту, підключення Health Connect API для відображення актуальної кількості кроків пройденої за день, зберігання інформації про кроки користувача, тестування застосунку та аналіз його ефективності.

Основна мета розробки мобільного застосунку шагоміра – заохотити людей вести здоровий спосіб життя, а також надати користувачам зручний та ефективний інструмент для відстеження свого фізичного стану та прогресу.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Актуальність теми розробки

У наш час турбота про здоров'я стає дедалі популярнішою. Люди більше замислюються над тим, як проводять свій день і скільки фізичної активності вони отримують. Для цього існують спеціальні трекери активності, які дозволяють користувачеві відстежувати свої показники та контролювати свою фізичну активність.

Розробка мобільного застосунку шагоміра з тамагочі є актуальною темою через високу популярність застосунків для відстеження активності. Така програма може значно покращити досвід використання застосунку, додавши елемент ігрової механіки для підвищення мотивації користувача.

Основна мета розробки мобільного застосунку шагоміра - допомогти користувачеві контролювати свою фізичну активність та підвищити рівень свого здоров'я. Застосунок буде відстежувати кількість кроків, пройдених користувачем протягом дня, а також виводити повідомлення, нагадуючи користувачу про необхідність проводити більше часу на свіжому повітрі та збільшувати свою фізичну активність.

Також у застосунку буде присутній елемент тамагочі, що додає ігрову механіку використання трекера активності. Це дозволить підвищити мотивацію користувача та зробити процес відстеження фізичної активності більш цікавим та захоплюючим.

Розробка мобільного застосунку шагоміра з тамагочі мовою програмування Kotlin є важливим завданням, оскільки дозволяє створити інноваційний продукт, який поєднує в собі функціональність трекера активності та елементи ігрової механіки. Такий продукт може стати популярним серед людей, які дбають про своє здоров'я та бажають зробити процес відстеження фізичної активності більш цікавим та захоплюючим.

1.2 Загальні поняття

1.2.1 Android SDK

Android SDK — це набір інструментів розробки, які використовуються для розробки програм для платформи Android, яка стала найбільшим конкурентом Apple у сфері смартфонів [1].

Кожного разу, коли Google випускає нову версію Android, також випускається відповідний SDK. Щоб мати можливість писати програми з найновішими функціями, розробники повинні завантажити та встановити SDK кожної версії для конкретного телефону. SDK по суті представляє набір інструментів Android для певної версії та технології її операційних систем [1].

Платформи розробки, сумісні з SDK, включають такі операційні системи, як Windows (XP або новіша версія), Linux (будь-який останній дистрибутив Linux) і Mac OS X (10.4.9 або новіша версія) [1]. Компоненти Android SDK можна завантажити окремо. Застосунки сторонніх розробників також доступні для завантаження.

Хоча SDK можна використовувати для написання програм Android у командному рядку, найпоширенішим методом є використання IDE. Рекомендовано IDE Eclipse із плагіном ADT. Однак інші IDE, такі як NetBeans або IntelliJ, також працюватимуть. Більшість із цих IDE надають графічний інтерфейс, що дозволяє розробникам швидше виконувати завдання розробки. Оскільки програми для Android написані на кодї Java, користувач повинен мати встановлений JDK [1].

Крім того, експерти іноді говорять про оптимізацію використання пакетів SDK для Android, наприклад, використання цільового управління версіями, щоб усунути потребу в додаткових загорнутих викликах API, або можуть посилатися на націлювання на SDK як на «додану вартість до маніфесту Android», що може допомогти з реалізаціями ОС, багато хто рекомендує встановити цільову версію та мінімальну версію в маніфесті [1].

1.2.2 Health Connect API

Health Connect — це платформа Android, яка дозволяє застосункам для здоров'я та фітнесу використовувати ті самі дані на пристрої в єдиній екосистемі [2].

Для розробників Android він надає API для читання та запису даних про здоров'я та фізичну форму користувача на пристрої. Для користувачів Android це місце, де можна контролювати, які програми мають доступ до даних про здоров'я та фізичну форму для читання та/або запису [2].

Екосистема Android включає широкий спектр пристроїв і платформ, включно зі смартфонами, носіями та численними застосунками для здоров'я, фітнесу та оздоровлення. Щоб адаптувати ці технології, API Android мають широкий вибір типів даних. Відповідно, користувачі Android хочуть здійснювати суворий і зручний контроль даних про своє здоров'я та фізичну форму з єдиного сховища даних [2].

Health Connect задовольняє потреби користувачів різними способами:

- покращує загальні знання про їхнє здоров'я та фітнес-діяльність;
- дозволяє користувачам отримати загальне уявлення про те, як дані про їхнє здоров'я та фізичну форму зберігаються та обмінюються ними;
- забезпечує швидкий і зручний доступ до різних елементів керування конфіденційністю [2].

Health Connect містить широкий спектр типів даних, які дозволяють користувачам відстежувати всі їхні дії щодо здоров'я та здоров'я за такими ключовими категоріями даних:

- активність фіксує будь-яку діяльність, яку виконує користувач, наприклад біг, плавання та їзда на велосипеді;
- вимірювання тіла фіксує загальні дані, пов'язані з тілом користувача, наприклад вагу або базальний рівень метаболізму;

- відстеження циклу фіксує менструальні цикли та пов'язані з ними точки даних, наприклад двійковий результат тесту на овуляцію;
- харчування фіксує типи даних про гідратацію та харчування, включаючи додаткові поля, такі як калорії, споживання цукру та вміст магнію;
- сон фіксує дані про інтервали, пов'язані з тривалістю та типом сну користувача;
- життєві показники фіксують важливу інформацію про загальний стан здоров'я користувача. Він включає все: від рівня глюкози в крові до температури тіла та насичення крові киснем [2].

1.2.3 Розробка мобільних застосунків для Android з використанням Kotlin

Kotlin – це мова програмування, яка може використовуватися для розробки мобільних програм під Android. Він став офіційною мовою розробки для Android у 2017 році і поступово стає все більш популярним.

Під час розробки мобільного застосунку на Kotlin для Android необхідно враховувати такі аспекти:

- середовище розробки - для розробки мобільних програм на Kotlin рекомендується використовувати Android Studio - офіційну IDE для розробки під Android, це безкоштовний та зручний інструментальний засіб, який має інтегровану підтримку мови Kotlin;
- архітектура програми - при розробці мобільного застосунку рекомендується використовувати архітектурний патерн MVP, MVVM або Clean Architecture, ці патерни допомагають створювати програми з високою продуктивністю та у майбутньому легше підтримувати та розширювати код;

- бібліотеки та інструменти - для розробки мобільних програм існує безліч бібліотек та інструментів, які полегшують роботу та прискорюють процес розробки;
- тестування - при розробці мобільного застосунку необхідно приділяти увагу тестуванню, як функціональну частину програми, так і інтерфейс користувача;
- оптимізація та управління ресурсами - при розробці мобільного застосунку необхідно враховувати обмеження ресурсів пристроїв, необхідно переконатися, що застосунок споживає мінімальну кількість ресурсів, таких як пам'ять та процесорний час, та оптимізувати його продуктивність.

Отже, необхідно приділяти достатньо часу на вивчення та практичне застосування всіх необхідних технологій та інструментів, щоб створити якісну програму. Також важливо продумати дизайн програми, його функціональність та зручність використання для користувачів. Рекомендується також звернути увагу на вимоги та рекомендації Google для розробки програм під Android, щоб переконатися, що програма буде відповідати всім необхідним стандартам та вимогам.

1.3 Дослідження програмних засобів для відстеження кроків

1.3.1 Google Fit

Google Fit — статистична платформа для відстеження фітнесу та змін здоров'я, розроблена Google для Android. Це набір API-інтерфейсів, що збирає відомості з різних застосунків і пристроїв. Застосунок використовує датчики руху в девайсі або мобільному пристрої, щоб отримати дані про фізичну діяльність (наприклад, про біг або їзду на велосипеді). Користувачі можуть вибирати, хто побачить дані про їхню активність, а також видалити їх у будь-який час [3]. Головний екран застосунку наведено на рис. 1.1.

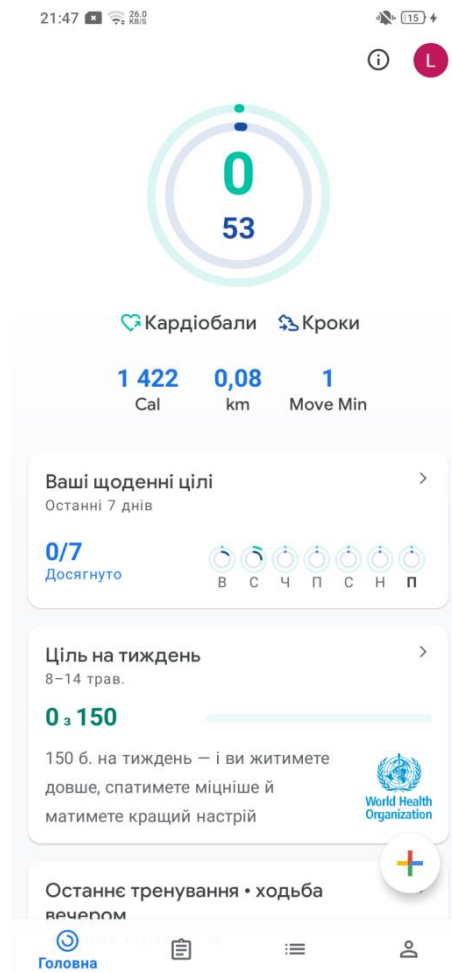


Рисунок 1.1 — Зображення мобільного застосунку Google Fit

1.3.2 StepsApp

StepsApp - мобільний застосунок для відстеження фізичної активності, що допомагає користувачам контролювати кількість кроків, які вони проходять протягом дня, а також інші показники, такі як дистанція, час активності, калорії, сон і т.д. Головний екран застосунку можна побачити на рис. 1.2.

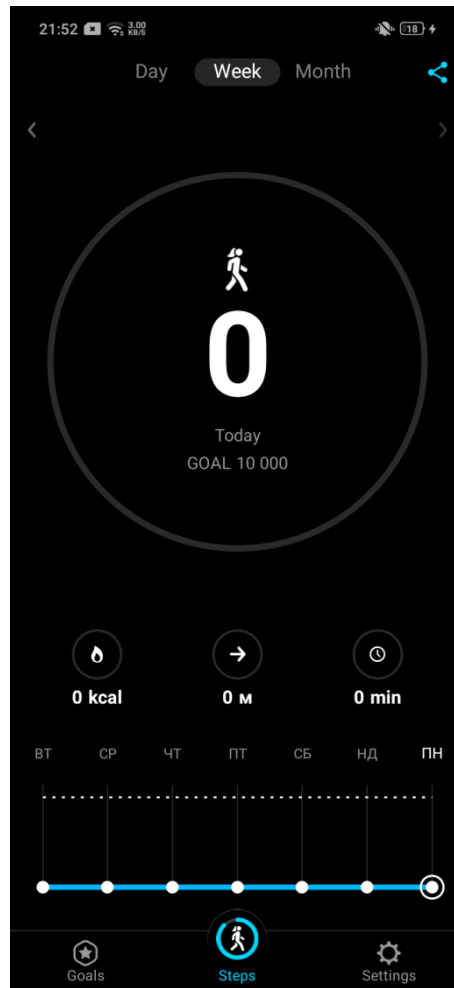


Рисунок 1.2 — Зображення мобільного застосунку StepsApp

1.4 Постановка завдання роботи

Для відстеження кількості кроків необхідно розробити мобільний застосунок з ігровими елементами для підвищення зацікавленості користувача.

Найменування застосунку – «TamiGo».

Застосунок повинен містити:

- відображення кількості пройдених кроків;
- привабливий та інтуїтивно зрозумілий інтерфейс користувача;
- високу продуктивність і швидкодію;
- підтримку роботи на різних пристроях;
- зчитування даних про кроки з Health Connect API;

- позитивне проходження тестування;
- підтримку зацікавленості користувача.

2 ОПИС ПРОГРАМИ

2.1 Порівняльний аналіз мов програмування

2.1.1 Kotlin

Ранні мови зосереджувалися на апаратних обмеженнях. Оскільки комп'ютери стають потужнішими, нові мови переходять до більш складного програмування з акцентом на надійності [4]-[6].

Ранні програмісти працювали безпосередньо з числами, що становлять машинні інструкції процесора. Цей метод приводив до численних помилок, тому було створено мову асемблера, щоб замінити числа мнемонічними операційними кодами, тобто словами та іншими корисними інструментами, які програмістам легко запам'ятовувати і читати. Однак між інструкціями мови асемблера та інструкціями машинної мови існувало дублювання один до одного, і програмісту доводилося писати кожен рядок коду асемблера. Крім того, кожен комп'ютерний процесор використовував власну мову асемблера.

Розробка програм на мові асемблера дуже витратна. Мови високого рівня допомагають вирішити цю проблему, створюючи інший рівень абстракції, ніж низькорівневі мови асемблера.

Kotlin збирається, а не інтерпретується. Інструкції інтерпретованої мови виконуються безпосередньо окремою програмою, яка називається інтерпретатором [4]-[6].

Навпаки, вихідний код скомпільованої мови перетворюється на інше представлення, яке виконується як власна програма або безпосередньо на апаратному процесорі, або на віртуальній машині, яка емулює процесор: такі мови, як C, C++, Go та Rust, компілюються у машинний код, який виконується безпосередньо на базовому апаратному CPU [4]-[6].

Такі мови, як Java і Kotlin, компілюються в байт-код, який є форматом середнього рівня, який працює не безпосередньо на апаратному процесорі, а

замість цього на віртуальній машині, яка є програмою, яка виконує інструкції байт-коду [4]-[6]. JVM-версія Kotlin працює на JVM [4]-[6].

Портативність – важлива перевага віртуальних машин. Один і той самий байт-код може виконуватися на будь-якому комп'ютері з віртуальною машиною. Віртуальні машини можна оптимізувати для конкретного обладнання, щоб вирішити проблеми зі швидкістю. JVM має такі оптимізації вже багато років і реалізована на багатьох платформах.

Під час компіляції код перевіряється компілятором на наявність помилок компіляції. Якщо помилок під час компіляції не виявлено, вихідний код компілюється у байт-код. Помилки виконання не можуть бути виявлені під час компіляції і виникають лише під час виконання програми. Як правило, такі помилки важко виявити і дорого виправити. Статично типізовані мови, такі як Kotlin, виявляють якомога більше помилок під час компіляції, тоді як динамічні мови виконують перевірку безпеки під час виконання.

2.1.2 Java

Java — це мова програмування та обчислювальна платформа, вперше випущена компанією Sun Microsystems у 1995 році [12]. Вона еволюціонувала від скромних початків до значної частини сучасного цифрового світу, забезпечуючи надійну платформу, на якій побудовано багато служб і програм [12]. Нові інноваційні продукти та цифрові послуги, розроблені для майбутнього, також продовжують покладатися на Java [12].

Вона багатоплатформна, об'єктно-орієнтована і сетецентрична, яка сама по собі може використовуватися як платформа [13]. Це швидка, безпечна та надійна мова програмування для всього: від мобільних застосунків та корпоративного ПЗ до застосунків для роботи з великими даними та серверних технологій [13].

Оскільки Java є безкоштовною та універсальною мовою, на ній створюються локалізовані та розповсюджені програми. Типові приклади використання Java:

- мобільні, комп'ютерні та відеоігри;
- децентралізовані хмарні застосунки;
- механізми обробки даних, які можуть працювати зі складними наборами та великими обсягами даних у режимі реального часу;
- застосунки штучного інтелекту, таких як обробка природної мови та глибоке навчання;
- програмування датчиків та апаратного забезпечення периферійних пристроїв, які можуть незалежно підключатися до Інтернету [13].

Секрет популярності Java полягає у простоті його використання. Деякі причини, через які розробники віддають перевагу Java іншим мовам програмування:

- детальна документація, вичерпні друковані матеріали та курси;
- багата екосистема вбудованих функцій та бібліотек для розробки низки застосунків;
- активна підтримка спільноти;
- високоякісні інструменти розробки;
- незалежність від платформи;
- користувачі можуть завантажувати ненадійний код Java через мережу і запускати його в безпечному середовищі, в якому він не може завдати жодної шкоди [13].

2.1.3 Порівняння мов програмування Kotlin та Java

Мови програмування Java та Kotlin є двома з найпопулярніших мов для розробки мобільних застосунків та веб-застосунків. Обидві мови мають

свої переваги та недоліки, які можуть вплинути на вибір програміста в залежності від конкретного завдання. У табл. 2.1 розглянуто основні характеристики Java та Kotlin та проаналізовано їх відмінності та подібності.

Таблиця 2.1 – Порівняння мов програмування Kotlin та Java

Характеристика	Kotlin	Java
Синтаксис	має більш чистий, простий та лаконічний синтаксис, що спрощує читання та написання коду	має більш багатослівний і більш громіздкий синтаксис, що ускладнює читання та написання коду
Безпека типів	забезпечує безпеку типів під час компіляції, що дозволяє запобігати помилкам часу виконання та покращує якість коду	не забезпечує безпеку типів під час компіляції, що може призводити до помилок часу виконання
Null-безпека	надає концепцію null-безпеки, що запобігає помилкам через невизначені значення null	не має концепції null-безпеки, що може призводити до NullPointerException
Функціональне програмування	підтримує функціональне програмування, що дозволяє писати простіший і зрозуміліший код	підтримує функціональне програмування, але не на тому ж рівні, що і Kotlin

Продовження таблиці 2.1

Характеристика	Kotlin	Java
Інтероперабельність	повністю сумісний з Java, що дозволяє використовувати існуючий код Java і бібліотеки в проектах Kotlin	не повністю сумісний з Kotlin, що обмежує використання Kotlin у проектах Java
Розширення функцій	дозволяє додавати нові функції до існуючих класів, навіть без успадкування цих класів, що полегшує написання коду і підвищує його читабельність	не дозволяє додавати нові функції до існуючих класів
Мультиплатформенність	дозволяє писати код для декількох платформ, включаючи Android, iOS, веб та сервер, що спрощує розробку програм на різних платформах	не має такого ж ступеня мультиплатформенності
Швидкість розробки	має більш простий та лаконічний синтаксис, що спрощує та прискорює процес розробки	має більш багатослівний синтаксис, що ускладнює та уповільнює процес розробки

Продовження таблиці 2.1

Характеристика	Kotlin	Java
Робота з функціями	підтримка функціонального програмування та лямбда-виразів	підтримка функціонального програмування, але без лямбда-виразів до версії 8
Використання змінних	визначення типу змінної може бути неявним (inferred)	тип змінної має бути вказано явно
Обробка винятків	Винятки є перевіреними (checked)	Винятки можуть бути як перевіреними, так і неперевіреними (unchecked)
Використання в Android-розробці	Офіційно підтримується Google і є рекомендованою мовою для Android-розробки	Довгий час був основною мовою для Android-розробки, але тепер рекомендується переходити на Kotlin
Популярність	Швидко зростає популярність, особливо у сфері Android-розробки	Є однією з найпопулярніших мов програмування у світі

Обидві мови мають свої переваги та недоліки. Котлін є сучаснішою та лаконічною мовою з високою безпекою типів та підтримкою функціонального програмування. В той час, як Java є більш поширеною мовою з ширшою базою знань та інструментів. Обидві мови підтримують

автоматичне складання сміття та мають можливість роботи з винятками. Однак Kotlin має більш високу null-безпеку, що може допомогти уникнути помилок у коді. В цілому, Kotlin і Java є хорошими мовами програмування, і вибір між ними залежить від конкретних потреб та переваг розробника.

2.2 Порівняльний аналіз архітектури програми

Архітектура програмного забезпечення відіграє важливу роль у створенні мобільних програм. Вона визначає структуру застосування та взаємодію його компонентів. Одними з найпоширеніших архітектур є MVP (Model-View-Presenter) та MVVM (Model-View-ViewModel). У цьому розділі розглянемо основні принципи кожної з цих архітектур і порівняємо їх переваги та недоліки.

2.2.1 MVP

MVP є скороченням від Model-View-Presenter і є похідним від шаблону проектування MVC (Model View Controller) [7].

Шаблон MVP — це спосіб відокремити фонові завдання від дій/фрагментів/переглядів, щоб зробити програму простішою, код коротшим і зручнішою для обслуговування. MVP робить все це, розділяючи логіку на три рівні:

- Модель - представляє об'єкти в застосунку. Це має логіку того, звідки потрібно отримати дані;
- Перегляд - відображає інформацію для користувачів і містить файл .xml компонента інтерфейсу користувача, дії, фрагменти та діалогове вікно під шаром перегляду;
- Презентатор - виконує завдання контролера і діє як посередник між видом і моделлю [7]-[8].

Проблема Android полягає в тому, що його дії тісно пов'язані як з інтерфейсом користувача, так і з механізмом доступу до даних. Необхідно визначити достатньо відокремлені шари, щоб полегшити розширення та підтримку додатку [9].

MVP робить подання незалежним від джерела даних. Ми можемо розділити додаток щонайменше на три різні шари і тестувати кожен з них незалежно. MVP видаляє більшу частину логіки з діяльності, щоб її можна було протестувати без використання інструментального тестування [9].

MVP має деякі ризики, і найважливіше, про що ми забуваємо, це те, що доповідач назавжди прив'язаний до перегляду. А перегляд — це діяльність, яка означає, що:

- ми можемо витікати з довготривалих завдань;
- ми можемо спробувати оновити дії, які вже загинули [9].

По-перше, якщо ви можете гарантувати, що ваші фонові завдання завершаються за розумний проміжок часу, я б не дуже хвилювався. Витік активності на 5-10 секунд не зробить ваш застосунок набагато гіршим, і рішення для цього зазвичай складні [9].

Другий момент викликає більше занепокоєння. Якщо ви відправляєте запит на сервер, який триває 10 секунд, але користувач закриває дію через 5 секунд. Коли викликається зворотній виклик і оновлюється інтерфейс користувача, дія завершується, і система падає. Щоб вирішити цю проблему, необхідно викликати метод `onDestroy()` для очищення представлення. Таким чином, уникається виклик дій у неузгодженому стані.

Схема роботи архітектури MVP наведена у рис. 2.1.

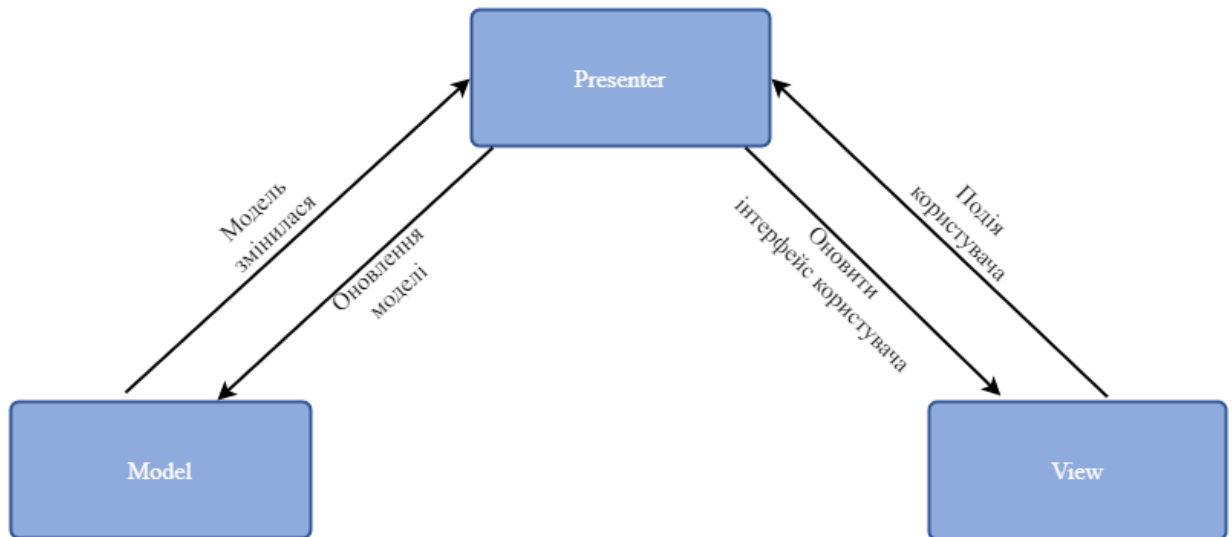


Рисунок 2.1 — Схема роботи архітектури MVP

2.2.2 MVVM

Патерн MVVM допомагає чітко відокремити бізнес-логіку та представлення додатку від користувацького інтерфейсу. Дотримання чіткого поділу між логікою програми та користувацьким інтерфейсом допомагає вирішити низку проблем розробки та спрощує тестування, підтримку та розробку додатків. Це також значно покращує повторне використання коду і дозволяє розробникам і дизайнерам користувацького інтерфейсу легше співпрацювати при проектуванні пов'язаних частин програми [10].

У шаблоні MVVM є три основні компоненти: модель, представлення та модель представлення. На рис. 2.2 показані зв'язки між трьома компонентами.

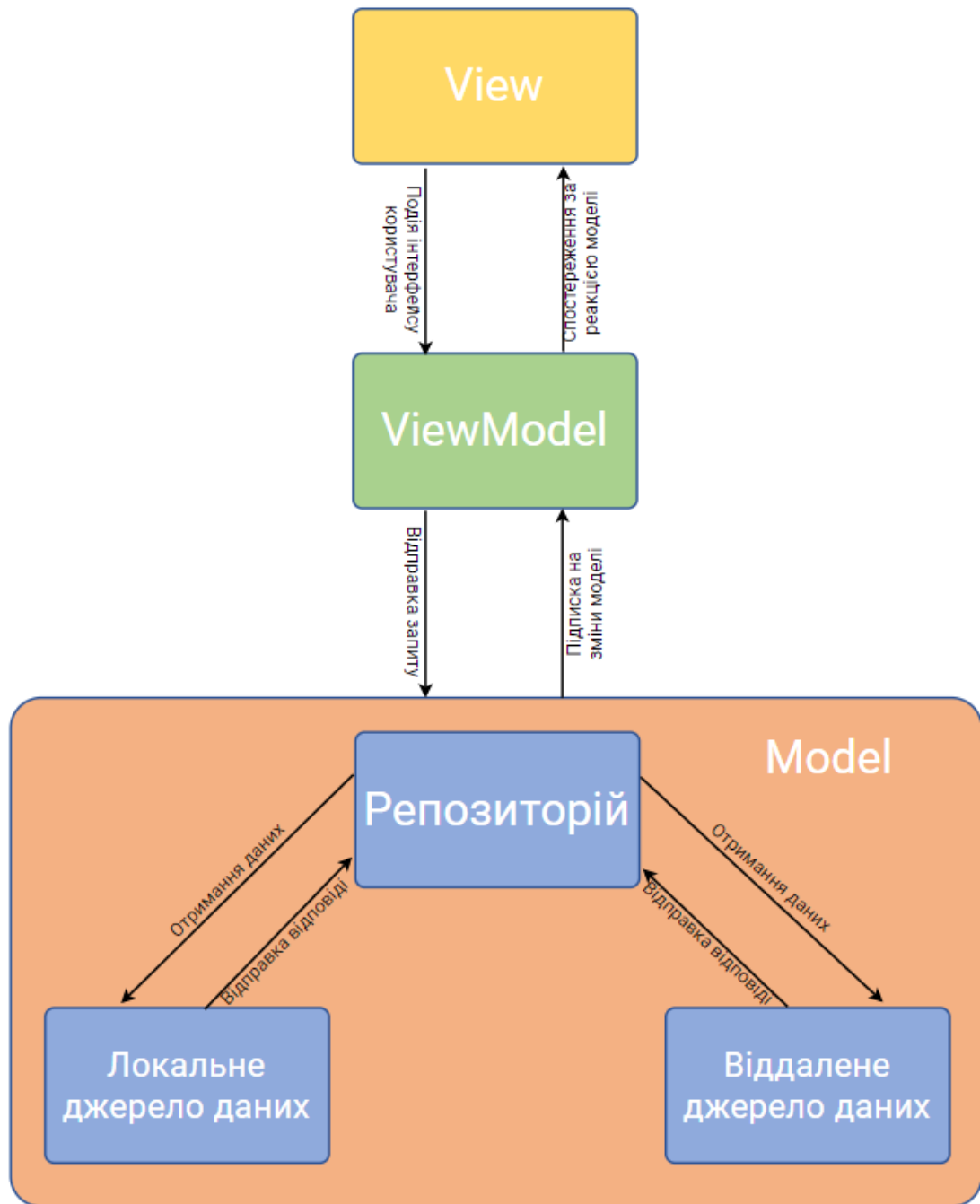


Рисунок 2.2 — Схема роботи архітектури MVVM

Окрім розуміння обов'язків кожного компонента, також важливо розуміти, як вони взаємодіють. На високому рівні представлення «знає про» модель представлення, а модель представлення «знає про» модель, але модель не знає про модель представлення, а модель представлення не знає про представлення. Таким чином, модель представлення ізолює

представлення від моделі та дозволяє моделі розвиватися незалежно від представлення [10].

Переваги використання шаблону MVVM такі:

- якщо існуюча реалізація моделі інкапсулює існуючу бізнес-логіку, змінити її може бути важко або ризиковано;
- розробники можуть створювати модульні тести для моделі перегляду та моделі без використання представлення;
- інтерфейс програми можна змінити, не торкаючись моделі подання та коду моделі;
- дизайнери та розробники можуть працювати незалежно та одночасно над своїми компонентами під час розробки [10].

Ключ до ефективного використання MVVM полягає в розумінні того, як розділити код програми на правильні класи та як класи взаємодіють.

2.2.3 Порівняння архітектур

Шаблони проектування MVP та MVVM – це два поширені підходи до проектування архітектури програмного забезпечення. Обидва шаблони використовуються для створення програм, які легко тестуються, супроводжуються та масштабуються. У табл. 2.2 порівнюються основні характеристики цих двох шаблонів проектування.

Таблиця 2.2 — Порівняння архітектур MVP та MVVM

Характеристики	MVP	MVVM
Поділ на шари	View, Model та Presenter	View, Model та ViewModel
Зв'язок між компонентами	компоненти пов'язані безпосередньо між собою	компоненти пов'язані через централізовану ViewModel

Продовження таблиці 2.2

Характеристики	MVP	MVVM
Тестування	легко тестується завдяки поділу на шари та відсутності прямого зв'язку з моделями	легко тестується завдяки відсутності прямого зв'язку з уявленнями та моделями
Відстеження змін	відстеження змін уявлення та моделі відбувається у презентері	відстеження змін відбувається у ViewModel
Управління станом	стан управляється презентером	стан управляється ViewModel
Складність реалізації	може вимагати великої кількості коду через прямий зв'язок між компонентами	може вимагати великої кількості коду через централізоване управління ViewModel
Популярність	шаблон проектування широко використовується у світі Android-розробки	шаблон проектування стає все більш популярним у світі Android-розробки
Область застосування	прості програми з мінімальною логікою	великі проекти з багатьма залежностями
Зрозумілість	простий для розуміння	може бути складним для початківців
Продуктивність	швидше, ніж MVVM	низька

2.3 Порівняльний аналіз методів підрахування кроків

При виборі відповідного варіанта для підрахування кроків варто враховувати безліч факторів. Спочатку я планувала реалізувати програму з використанням акселерометра, проте в процесі дослідження була виявлена можливість використання HealthConnect API, що викликало питання – що краще вибрати? У цьому пункті буде проведено аналіз та порівняння двох підходів, з урахуванням їх переваг та недоліків, щоб вибрати найбільш оптимальне рішення для розробки застосунку.

Для прийняття рішення про те, що краще використовувати – Health Connect API або самостійно реалізований крокомір через акселерометр, необхідно розглянути їхні переваги та недоліки.

HealthConnect API надає розробникам доступ до великого обсягу інформації про здоров'я користувачів, включаючи дані про кроки, витрачені калорії, пульс і т. д. Це може бути корисно, якщо потрібний моніторинг здоров'я користувачів на високому рівні та інтеграція з іншими програмами та сервісами в області здоров'я.

Самостійно реалізований крокомір через акселерометр є більш простим та доступним рішенням. У цьому випадку не потрібно зв'язуватися із зовнішніми сервісами та використовувати сторонні API, що може зменшити витрати на розробку. Однак, реалізація такого крокоміра може бути менш точною та надійною, особливо якщо пристрій користувача не оснащений якісним акселерометром. Також при розробці самостійно реалізованого крокоміра необхідно враховувати безліч нюансів, пов'язаних з фізичними характеристиками пристрою, а також з можливими помилками при обробці даних.

Щоб більш точно проаналізувати обидва варіанти була створена табл. 2.3 для остаточного рішення.

Таблиця 2.3 — Порівняння методів підрахунку кроків.

	Самостійно реалізований крокомір через акселерометр	HealthConnect API
Плюси використання	застосунок може бути безкоштовним для скачування та використання, оскільки не потребує використання додаткових сторонніх сервісів	використовує безліч сенсорів для визначення кількості кроків, таких як акселерометри, гіроскопи та магнітометри - це дозволяє отримати більш точну інформацію, ніж простий вимір через акселерометр
	реалізація крокоміра через акселерометр дозволяє користувачеві використовувати програму в будь-яких місцях, навіть без підключення до Інтернету	можна швидко та легко отримати доступ до даних про кроки, не витрачаючи час на розробку та тестування власного крокоміра
	користувач може бути впевнений, що дані не передаються стороннім компаніям, що підвищує рівень конфіденційності	може надавати додаткові функції, такі як відстеження сну та серцевого ритму, які можуть бути корисними для різних цілей
	акселерометр у сучасних мобільних пристроях має високу точність вимірювання, що дозволяє отримувати більш точні результати	простота використання та інтеграції
		підтримка стандарту HL7 для обміну медичними даними

Продовження таблиці 2.3

	Самостійно реалізований крокомір через акселерометр	HealthConnect API
Мінуси використання	реалізація крокоміра через акселерометр може бути обмежена лише функціями підрахунку кроків та розрахунку дистанції, що може не задовольняти користувачів, які бажають ширшого функціоналу	доступний лише для певних пристроїв та операційних систем - це може обмежити кількість користувачів, які можуть використовувати програму
	реалізація крокоміра через акселерометр споживає додатковий заряд акумулятора, що може бути небажаним для користувачів, які не хочуть витрачати більше часу на зарядку свого пристрою	вимагає доступу до Інтернету для роботи, що може бути небажаним для деяких користувачів
	для досягнення максимальної точності при використанні акселерометра необхідно дотримуватися певних умов, таких як правильне розташування пристрою на тілі користувача і відсутність інших об'єктів, що рухаються поруч	при використанні можуть виникнути побоювання щодо безпеки персональних даних користувачів, оскільки API може мати доступ до чутливої інформації

Продовження таблиці 2.3

	Самостійно реалізований крокомір через акселерометр	HealthConnect API
Мінуси використання	зберігання даних у пристрої користувача може становити ризик для конфіденційності, особливо якщо не використовуються надійні механізми захисту даних	залежність від стороннього сервісу
	можливість помилок при обробці даних	обмежений контроль над обробкою даних
	необхідність у розробці та обслуговуванні	

2.4 Порівняльний аналіз ін'єкції залежностей

2.4.1 Hilt

Hilt — це бібліотека для ін'єкцій залежностей для Android, яка скорочує шаблонне виконання ручного ін'єкції залежностей у проєкті. Виконання ін'єкції залежностей вручну вимагає створювати кожен клас і його залежності вручну, а також використовувати контейнери для повторного використання та керування залежностями [14].

Hilt пропонує стандартний спосіб використання DI у програмі, надаючи контейнери для кожного класу Android у проєкті та автоматично керуючи їхніми життєвими циклами. Hilt створено на основі популярної бібліотеки DI Dagger, щоб отримати переваги від коректності під час компіляції, продуктивності під час виконання, масштабованості та підтримки Android Studio, яку надає Dagger [14].

Цілями Hilt є:

- спростити пов'язану з Dagger інфраструктуру для програм Android;
- створення стандартного набору компонентів і областей для полегшення налаштування, читабельності/розуміння та спільного використання коду між програмами;
- забезпечити простий спосіб надання різних прив'язок до різних типів складання (наприклад, тестування, налагодження чи випуск) [15].

2.4.2 Koin

Koin - це легкий фреймворк для впровадження залежностей (DI) у застосунки, написані мовою програмування Kotlin. Він надає простий та декларативний підхід до управління залежностями, роблячи процес ін'єкції залежностей більш зрозумілим та гнучким.

Головна мета Koin - забезпечити зручну та інтуїтивно зрозумілу роботу із залежностями, мінімізуючи необхідність у налаштуваннях та складності використання. Він прагне зробити процес DI легким та зрозумілим для розробників Kotlin.

Він простий у використанні, не впливає на час компіляції та не вимагає додаткового налаштування плагіна. Завдяки мові Kotlin Koin розширює платформу Android і надає нові функції як частину оригінальної платформи [16].

2.4.3 Порівняння ін'єкції залежностей

У табл. 2.4 наведено порівняння фреймворків для ін'єкції залежностей, таких як Hilt та Koin.

Таблиця 2.4 — Порівняння ін'єкції залежностей Hilt та Koin

Характеристика	Hilt	Koin
Використання	розроблений як офіційне рішення для залежностей у Android	легковаговий фреймворк для впровадження залежностей у Kotlin
Підтримка	розробляється та підтримується Google	відкритий і активно розвивається на GitHub
Компіляційна перевірка	перевірка залежностей під час компіляції	працює під час виконання, відсутня компіляційна перевірка
Інтеграція з Android	повна інтеграція з Android-фреймворком	інтеграція з Android-фреймворком, але менш прямолінійна
Відстеження залежностей	використовує анотації для вказівки залежностей	описує залежності у вигляді модулів та функцій
Область видимості залежностей	підтримує різні рівні області видимості залежностей	відсутня підтримка різних рівнів області видимості
Конфігурування залежностей	відносно складне та розширюване конфігурування	просте та гнучке конфігурування
Впровадження у конструкторі	підтримується	Підтримується
Впровадження у поля	підтримується	Підтримується
Впровадження до методів	підтримується	Підтримується

Продовження таблиці 2.4

Характеристика	Hilt	Koin
Декларативність	менш декларативний, вимагає більше налаштувань та конфігурації	більш декларативний, використовує DSL для опису залежностей
Роздільна здатність циклічних залежностей	підтримується	Підтримується
Сумісність з іншими фреймворками	сумісний з іншими фреймворками в екосистемі Android	сумісний з іншими фреймворками в екосистемі Kotlin
Розмір бібліотеки	більше, через інтеграцію з Android-фреймворком	менше, легкий і компактний
Навчання та документація	обширна документація та посібники з боку Google	обширна документація та посібники на офіційному сайті
Спільнота	велика та активна спільнота розробників	активна спільнота, але менша, ніж у Hilt
Підтримка анотацій	підтримує анотації для позначення залежностей	використовує DSL для опису залежностей
Масштабованість	добре масштабується та підходить для великих проєктів	добре масштабується, але може бути складна у налаштуванні
Продуктивність	краща продуктивність завдяки компіляційній перевірці	хороша продуктивність, але менша через відсутність компіляційної перевірки
Використання в Kotlin	використовується в Java та Kotlin проєктах	використовується тільки в Kotlin проєктах

2.5 Висновок з розділу 2

З проведеного аналізу були розглянуті мови програмування для розробки програми на Android, зокрема Java та Kotlin. Після порівняння та аналізу основних характеристик та переваг обох мов, було прийнято рішення використовувати Kotlin для розробки програми. Kotlin є більш сучасною мовою, яка пропонує ряд переваг перед Java, таких як безпечніший код, більш проста і компактна синтаксична структура, зручний null-безпечний виклик методів, функції, що розширюються, і багато іншого. Крім того, Kotlin дозволяє легше створювати асинхронний код, що дуже важливо для мобільних програм, де продуктивність відіграє ключову роль. У результаті, використання Kotlin для розробки програми дозволить прискорити процес розробки та створити безпечніший та ефективніший застосунок для користувачів.

Також під час проведеного аналізу архітектур MVP і MVVM можна зробити висновок, що обидві архітектури мають свої переваги та недоліки. MVP добре підходить для маленьких застосунків, які не вимагають складної логіки, а MVVM рекомендується використовувати у великих проектах, де є складна бізнес-логіка та велика кількість даних. В даному випадку було прийнято рішення використовувати MVVM, оскільки застосунок має великий потенціал для розширення та розвитку, і потребує складної бізнес-логіки. MVVM забезпечує більш гнучку та масштабовану архітектуру, що у свою чергу спрощує розробку, тестування та супровід програми надалі.

Також було вирішено використовувати HealthConnect API тому що було встановлено, що це дозволить суттєво знизити навантаження на батарею пристрою, покращити точність вимірювань та отримати доступ до додаткової функціональності, такої як місцезнаходження та аналіз сну. Крім того, використання готового API дозволить значно прискорити процес розробки та скоротити витрати на розробку та тестування власного алгоритму обробки даних із акселерометра.

Виходячи з порівняльної таблиці, було прийнято рішення використовувати фреймворк Koin для застосування залежностей у застосунку. Незважаючи на те, що Hilt також пропонує ряд переваг і добре інтегрований з Android-платформою, Koin виявився більш підходящим вибором для цього проекту. Koin забезпечує легке та гнучке впровадження залежностей, що дозволить ефективно керувати залежностями та забезпечити плавну роботу програми.

3 РОЗРОБКА ПРОГРАМИ

3.1 Структурна схема програми

У цьому розділі буде розглянута структурна схема програми, де буде відображення основних компонентів та їх взаємозв'язків у програмі. Це допоможе краще зрозуміти архітектуру та організацію програмного коду, а також полегше супровід та модифікацію програми.

Фізична модель мобільного застосунку шагоміру відображає впорядкований набір файлів різного типу (класи, ресурси, маніфест).

Фізична структура мобільного застосунку шагоміру представлена на рис. 3.1.

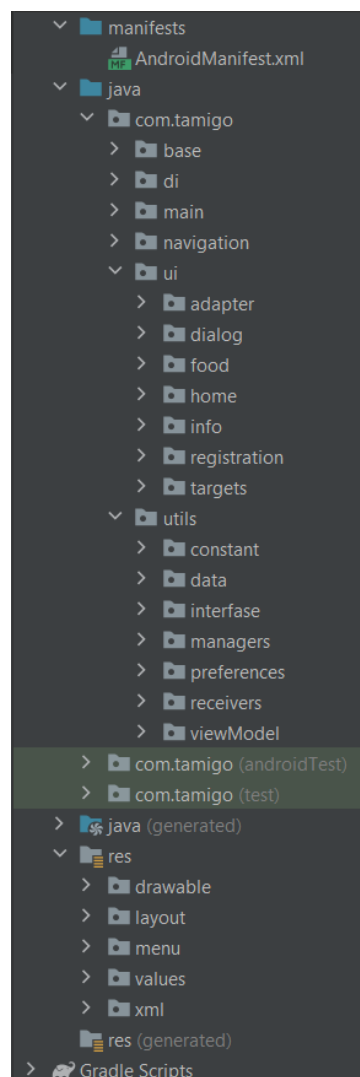


Рисунок 3.1 — Фізична структура застосунку

На рис. 3.2 наведено загальну структуру мобільного застосунку шагоміру.

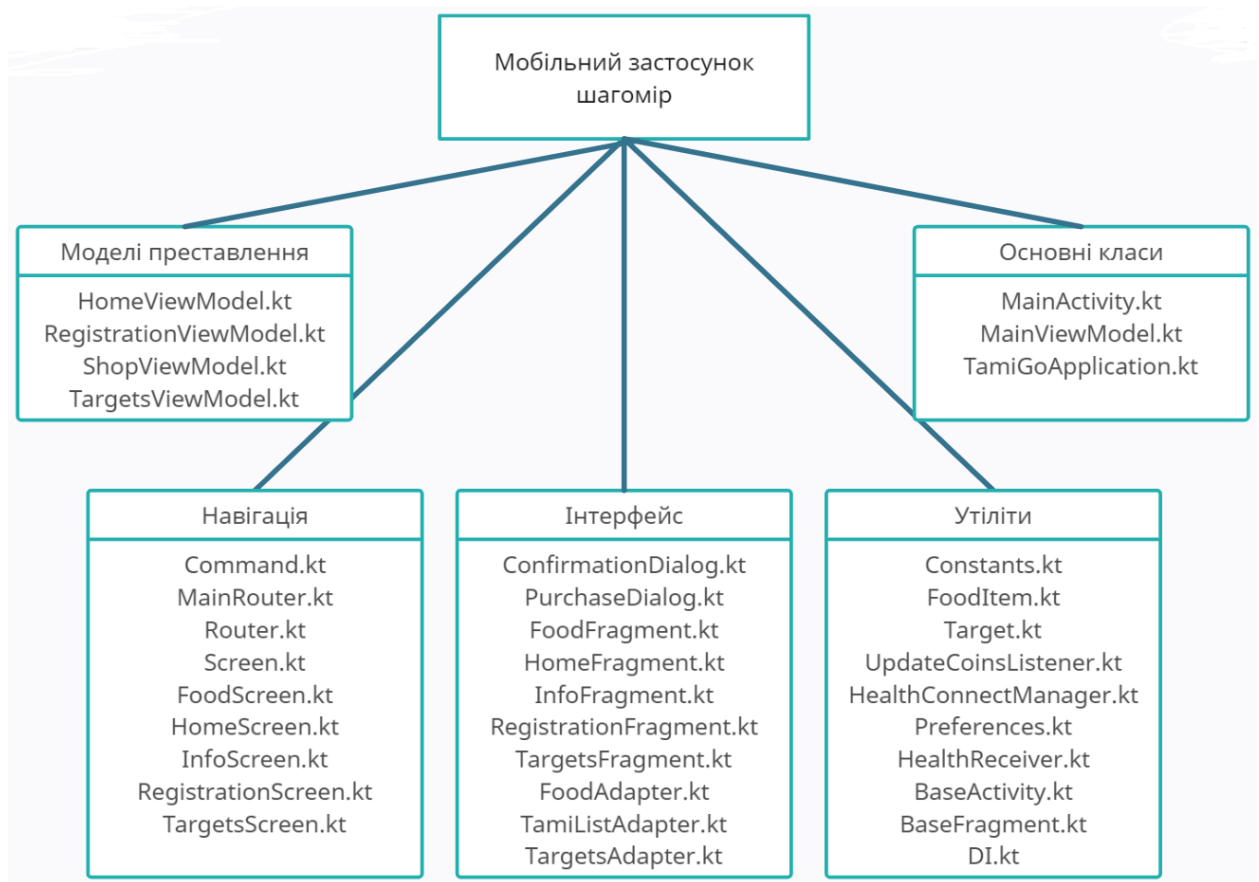


Рисунок 3.2 — Загальна структура застосунку

Отже, загальна структура програми відображає організацію та логічну структуру програмного коду, в той час як фізична структура програми відображає спосіб, яким цей код організований та зберігається на фізичному носії.

3.2 Файли, методи, та функції

Файл `TamiGoApplication.kt` є головним класом застосунку, який являє собою точку входу та керує життєвим циклом програми. У ньому було реалізовано підключення до ін'єкції залежностей `Koin`.

Файл MainActivity.kt є головною активністю застосунку. У ній реалізована логіка відображення екрану завантаження та бокового меню. В табл. 3.1 наведено опис призначення методів та змінних програмного коду.

Таблиця 3.1 — Опис призначення методів та змінних файлу TamiGoApplication.kt

Назва	Призначення
router	запуск та закріплення навігаційного роутера до активності
mainViewModel	об'єкт вьюмоделі активності
healthConnectManager	запуск та закріплення менеджера по роботі з HealthConnect API до активності
onUpdateCoinsBalance()	оновлення балансу яке сигналізує з одного фрагменту до іншого
openConfirmationDialog()	відкриття діалогового вікна підтвердження

Файл MainViewModel.kt реалізує бізнес логіку застосунку. Він містить клас який має ідентичну назву, але в кінці додано Impl. Це було виконано тому що використання імплементації класів дозволяє розділити загальну абстракцію від конкретної реалізації, забезпечуючи кращу структуру коду, гнучкість та перевикористання. Аналогічно були реалізовані класи HomeViewModel.kt, RegistrationViewModel.kt, ShopViewModel.kt, TargetsViewModel.kt. У табл. 3.2, 3.3, 3.4, 3.5, 3.6 наведено опис признаення методів та змінних програмного коду.

Таблиця 3.2 — Опис призначення методів та змінних файлу MainViewModel.kt

Назва	Призначення
navigate()	навігація між фрагментами
checkHealthConnectStatusAndPermissions()	перевірка статусу та дозволу на використання HealthConnect API
navigateInfo()	навігація до інформаційного фрагменту
onClickRestart()	сповіщає про те що було обрано перезапуск застосунку та реалізує очищення даних

Таблиця 3.3 — Опис призначення методів та змінних файлу HomeViewModel.kt

Назва	Призначення
health	відображає актуальне здоров'я віртуального вихованця
getTamiName()	отримання ім'я віртуального вихованця
getTamiSkin()	отримання зображення віртуального вихованця
navigateToRegistrationScreen()	навігація до екрану реєстрації
openTargetsFragment()	відкриття фрагменту з цілями
openShopFragment()	відкриття фрагмента з магазином
getCoins()	отримання монет
isNeedStartAlarm()	перевірка чи потрібно запускати таймер для реалізації механіки здоров'я
registerPrefsListener()	реєстрація слухача зміни даних програми
unregisterPrefsListener()	відміна реєстрації слухача зміни даних
getHealthFromPref()	отримання здоров'я з даних програми

Таблиця 3.4 — Опис призначення методів та змінних файлу RegistrationViewModel.kt

Назва	Призначення
navigateToHomeFragment()	навігація на домашній екран
saveInfoAboutTami()	збереження даних про віртуального вихованця

Таблиця 3.5 — Опис призначення методів та змінних файлу ShopViewModel.kt

Назва	Призначення
setProductToInventory()	збереження купленого продукту до інвентаря
getProductsFromInventory()	отримання продуктів з інвентаря
removeProductFromInventory()	видалення продукту з інвентаря
isBuyProduct()	перевірка чи можливо купити продукт

Таблиця 3.6 — Опис призначення методів та змінних файлу TargetsViewModel.kt

Назва	Призначення
currentSteps	відображає кроки які здійснив користувач
getSteps()	отримання кроків які здійснив користувач
setTarget()	встановлення цілі користувача
getTarget()	отримання цілі користувача
setCoinsToBalance()	додавання монет до балансу

Файл FoodFragment.kt реалізує інтерфейс користувача для відображення продуктів у інвентарі та магазину де їх можна придбати.

Файл HomeFragment.kt реалізує інтерфейс користувача для відображення віртуального вихованця, його ім'я та здоров'я, балансу монет. Також має у собі контейнер для відображення інших фрагментів.

Файл InfoFragment.kt реалізує інтерфейс користувача для відображення інформації про застосунок та його розробника.

Файл RegistrationFragment.kt реалізує інтерфейс користувача для реєстрації користувача у застосунку, відображає введення ім'я та вибір зображення віртуального вихованця.

Файл TargetsFragment.kt реалізує інтерфейс користувача для відображення обраної цілі, її прогрес виконання та вибору інших можливих цілей.

Файли FoodAdapter.kt, TamiListAdapter.kt, TargetsAdapter.kt створені для зв'язку даних з елементами інтерфейсу користувача, що відображаються у вигляді списку або сітки.

Файли ConfirmationDialog.kt та PurchaseDialog.kt використовуються для взаємодії з користувачем у застосунку. Являють собою спливаючі вікна.

Файли Command.kt, MainRouter.kt, Router.kt, Screen.kt, FoodScreen.kt, HomeScreen.kt, InfoScreen.kt, RegistrationScreen.kt, TargetsScreen.kt створені для навігації між фрагментами які відображають інтерфейс користувача.

Файл Constants.kt зберігає в собі списки з можливими зображеннями віртуального вихованця, можливі цілі, данні про продукти та константи для намірів.

Файли FoodItem.kt та Target.kt це класи даних які являють собою моделі для зберігання даних.

Файл UpdateCoinsListener.kt це інтерфейс який використовується як слухач зміни балансу монет.

Файл HealthConnectManager.kt це клас який працює з HealthConnect API. У табл. 3.7 наведено опис признаення методів та змінних програмного коду.

Таблиця 3.7 — Опис призначення методів та змінних файлу HealthConnectManager.kt

Назва	Призначення
activity	об'єкт активності
permissionRequest	об'єкт класу SuspendActivityResultLauncher
attach()	призначена для підключення менеджера до активності
checkHealthConnect()	перевірка чи присутній на пристрої Health Connect
getSteps()	отримання кількості кроків з API
checkPermissions()	перевірка чи надані дозволи застосунку
SuspendActivityResultLauncher	внутрішній клас для асинхронної перевірки дозволів

Файл Preferences.kt створений для єдиного доступу до даних застосунку. У табл. 3.8 наведено опис призначення методів та змінних програмного коду.

Таблиця 3.8 — Опис призначення методів та змінних файлу Preferences.kt

Назва	Призначення
registerListener()	реєстрація слухача зміни даних
unregisterListener()	відміна реєстрації слухача зміни даних
setTamiName()	встановлення ім'я віртуального вихованця
getTamiName()	отримання ім'я віртуального вихованця
setTamiSkin()	встановлення зображення віртуального вихованця

Продовження таблиці 3.8

Назва	Призначення
getTamiSkin()	отримання зображення віртуального вихованця
setTarget()	встановлення цілі
getTarget()	отримання цілі
setCoinsToBalance()	встановлення монет до балансу
removeCoinsFromBalance()	видалення монет з балансу
getCoinsBalance()	отримання монет з балансу
setProducts()	встановлення продуктів до інвентаря
getProducts()	отримання продуктів з інвентаря
setFirstLaunch()	встановлення ключа який відображає чи потрібно запускати логіку підрахунку здоров'я
isFirstLaunch()	отримання ключа який відображає чи потрібно запускати логіку підрахунку здоров'я
setStartTargetTime()	встановлення часу коли була прийнята ціль
getStartTargetTime()	отримання часу коли була прийнята ціль
setHealth()	встановлення здоров'я
getHealth()	отримання здоров'я

Файл HealthReceiver.kt створений для реалізації логіки втрати здоров'я кожну годину та відображення цього у повідомленнях. У табл. 3.9 наведено опис признаення методів та змінних програмного коду.

Таблиця 3.9 — Опис призначення методів та змінних файлу HealthReceiver.kt

Назва	Призначення
minusHealth	віднімає здоров'я кожен раз як приходить намір
plusHealth	додає здоров'я кожен раз як приходить намір
clearGame	очищує дані гри якщо віртуальний вихованець втратив весь баланс здоров'я
sendNotification	відправка повідомлення
setAlarm	встановлює таймер на одну годину для реалізації логіки стану здоров'я

Файл DI.kt використовується для керування залежностями між компонентами програмної системи. Основна мета полягає в тому, щоб відокремити створення об'єктів від їх використання та забезпечити передачу залежностей зовнішнім чином, замість того, щоб клас самостійно створював їх.

Файли BaseActivity.kt та BaseFragment.kt створені для спрощення та стандартизації розробки, а також для повторного використання коду. Допомагають скоротити дублювання коду, підвищити ефективність та забезпечити однаковість усередині застосунка.

3.3 Використання HealthConnect API

Для реалізації функціональності відстеження кроків та активності користувача у мобільному застосунку шагоміру було використано HealthConnect API. HealthConnect API надає доступ до даних про кроки та інші показники користувача.

У класі `HealthConnectManager`, який є менеджером для роботи з `HealthConnect API`, було реалізовано такі методи:

а) метод `checkHealthConnect()` перевіряє доступність `HealthConnect API` та реагує відповідним чином залежно від статусу доступності:

1) якщо `API` недоступне, метод завершується без подальших дій;

2) якщо потрібне оновлення `HealthConnect`, відкривається відповідний додаток для оновлення;

3) в іншому випадку виконується перевірка дозволів;

б) метод `getSteps()` отримує кількість кроків користувача протягом певного періоду часу, всередині методу формується запит на читання записів типу `StepsRecord` у заданому часовому діапазоні, а потім, використовуючи `HealthConnect API`, виконується читання записів і повертається сумарна кількість кроків;

в) приватний метод `checkPermissions()` перевіряє дозволи для роботи з `HealthConnect API`, всередині методу відбувається асинхронна перевірка вже наданих дозволів, якщо не всі необхідні дозволи надано, використовується `SuspendActivityResultLauncher` для запуску процесу запиту дозволів у користувача;

г) клас `SuspendActivityResultLauncher` є допоміжним класом для очікування на результат запиту дозволів, він використовує `registerForActivityResult` для реєстрації контракту запиту дозволів та повертає отримані дозволи у вигляді `Set<String>`;

Для роботи з `HealthConnect API` у маніфесті програми було додано дозвіл `android.permission.health.READ_STEPS`, який потрібний для читання даних про кроки користувача.

Також у маніфесті були додані такі елементи:

```

    <action android:name="androidx.health.ACTION_SHOW_PERMISSIONS_RATIONALE" />

```

- для відображення пояснення необхідності надання дозволів користувачеві.

- `<meta-data>` - містить ім'я бібліотеки та значення, яке необхідно визначити.
- `<queries>` - визначення пакета `com.google.android.apps.healthdata`, необхідного для взаємодії з HealthConnect API.

Використання HealthConnect API в мобільному застосунку шагоміру, дозволяє отримувати і відстежувати дані про кроки користувача, що є важливою частиною функціональності програми. API надає надійний та зручний спосіб отримання цих даних, що допомагає підвищити точність та надійність роботи шагоміра.

4 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРЕМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

4.1 Призначення й умови застосування програми

Мобільний застосунок шагомір «TamGo» – це інтерактивна мобільна програма, розроблена для пристроїв на платформі Android. Він пропонує користувачеві захоплюючу ігрову механіку, засновану на годівлі та догляді за віртуальним вихованцем шляхом рахунку кроків. Застосунок стимулює користувачів підвищити свою середньодобову активність та створює зацікавленість у турботі про віртуального вихованця.

Основною метою застосунку є мотивація користувачів до активного способу життя та покращення фізичної активності. Часто в сучасному суспільстві люди ведуть сидячий спосіб життя і відчують нестачу руху. Це може негативно позначатися на їхньому здоров'ї та загальному самопочутті. Застосунок створено для стимуляції користувачів до піклування про віртуального вихованця, який потребує регулярного годування та фізичної активності.

Умови застосування програми:

- встановлення застосунку «TamGo» на пристрій із операційною системою Android;
- проходження процесу реєстрації, вказавши ім'я вихованця та обравши його забарвлення;
- необхідність стежити за станом вихованця та забезпечувати його потреби;
- необхідно активувати функцію підрахунку кроків на пристрої, які здійснюються протягом дня;
- зароблені монети можуть бути витрачені на купівлю продуктів, якими можна годувати віртуального вихованця;

Застосунок «TamGo» пропонує користувачеві унікальну комбінацію ігрової механіки, фізичної активності та турботи про віртуального вихованця. Вона ставить перед користувачами мету покращити свій спосіб життя і зробити його активнішим. Застосунок дозволяє користувачам отримати винагороду за свою фізичну активність у вигляді монет, які можна витратити на покупку продуктів для годування. В результаті, користувачі можуть досягти кращої фізичної форми, підвищити свою середньодобову активність та отримати задоволення від турботи про свого віртуального вихованця.

4.2 Опис програми

4.2.1 Загальні відомості

Застосунок є ігровою програмою, розробленою для платформи Android. Воно надає користувачам можливість піклуватися про віртуального вихованця, підтримуючи його життя та здоров'я шляхом відстеження кроків. Користувачі отримують винагороду у вигляді монет за кожну виконану ціль, які можуть бути використані для придбання продуктів харчування для вихованця.

4.2.2 Функціональне призначення

Основна мета застоснку - підвищення середньодобової активності користувачів та зацікавленості у догляді за віртуальним вихованцем. Користувачі повинні активно ходити та робити кроки, щоб заробити монети. Зароблені монети можуть бути витрачені на купівлю різних продуктів харчування, якими можна годувати віртуального вихованця.

4.2.3 Опис логічної структури

Застосунок складається з кількох основних компонентів. В його основі знаходиться головний екран, на якому користувач може бачити свого віртуального вихованця, його стан та баланс монет. Також включає фрагменти, пов'язані зі списком доступних продуктів та магазином, а також екраном де відображається обрана ціль та список можливих цілей. Вся ця структура забезпечує плавну навігацію користувача і дозволяє йому взаємодіяти з різними функціями програми.

4.2.4 Використані технічні засоби

Мобільний застосунок шагомір розроблено за допомогою мови програмування Kotlin, для підрахунку кроків використано HealthConnect API, архітектура MVVM, ін'єкція залежностей виконана за допомогою бібліотеки Koin. Для візуального представлення інтерфейсу програми застосовуються розмітки XML. Додатково, програма використовує такі бібліотеки та інструменти:

- Splash screen;
- Glide;
- Gson.

4.2.5 Виклик та завантаження

Для запуску програми користувач повинен мати файл формату .apk та встановити його на свій пристрій, сумісний із платформою Android. Після встановлення застосунок може бути запущеним з головного екрана пристрою або зі списку програм. При запуску застосунок завантажує всі необхідні ресурси, включаючи дані про вихованця та продукти харчування.

4.3 Інтерфейс

Перший запуск застосунку відкриває нам екран реєстрації. Зображення екрану реєстрації наведено на рис. 4.1. На ньому ми можемо ввести ім'я віртуального вихованця та обрати його зображення.

Після цього відкривається домашній екран на якому відображено ім'я, зображення та шкала здоров'я віртуального вихованця. Також там знаходяться баланс монет, дві кнопки для переходу між фрагментами та контейнер для фрагментів. Зображення домашнього екрану наведено на рис. 4.2.

На рис. 4.3 зображено домашній екран з відкритим фрагментом який відображає саму ціль, прогрес її проходження та список можливих цілей.

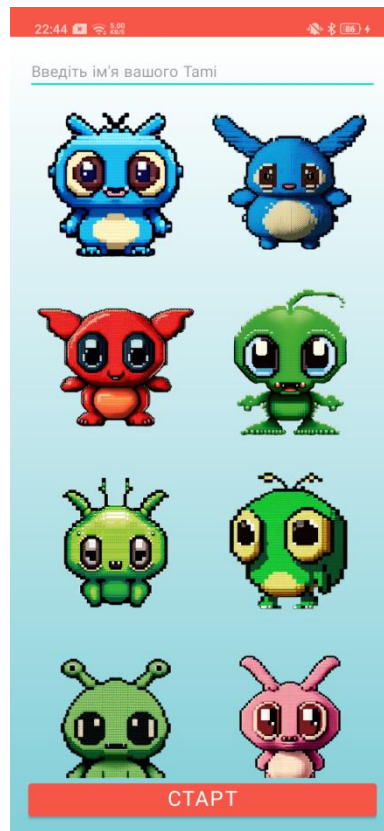


Рисунок 4.1 — Екран реєстрації



Рисунок 4.2 — Домашній екран



Рисунок 4.3 — Домашній екран з фрагментом цілей

Після виконання заданої цілі активується кнопка для отримання винагороди, це відображено на рис. 4.4.

Після отримання винагороди активна ціль скидається, а винагорода додається до балансу монет на домашньому екрані, можна це побачити на рис. 4.5.

Якщо на домашньому екрані натиснути на кнопку «Магазин», екран з цілями зміниться на екран з продуктами, які ми можемо купити, або які знаходяться в інвентарі. Екран з продуктами відображен на рис. 4.6.



Рисунок 4.4 — Екран з виконаною ціллю та активною кнопкою винагороди



Рисунок 4.5 — Екран після отримання винагороди



Рисунок 4.6 — Домашній екран який відображає фрагмент з продуктами

Коли необхідно купити продукт, необхідно натиснути на обраний та після цього з'явиться діалогове вікно, яке відображене на рис. 4.7. Можна відмовитися від покупки та вікно зачиниться, або погодитися після чого можна побачити обраний продукт у інвентарі, який зображено на рис. 4.8. Якщо коштів буде не достатньо, на екрані з'явиться впливаюче повідомлення, яке зображено на рис. 4.9.

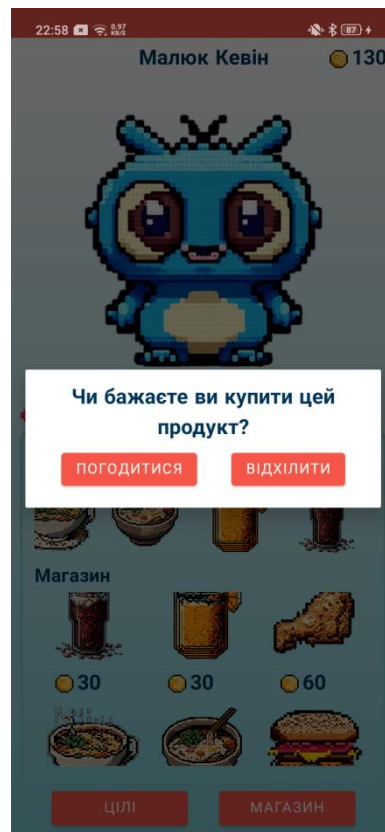


Рисунок 4.7 — Діалогове вікно для покупки продукта



Рисунок 4.8 — Домашній екран з інвентарем та магазином



Рисунок 4.9 — Вспливає повідомлення «Не достатньо коштів»

Коли прийде час годувати віртуального вихованця, необхідно використати продукт з інвентаря. Для цього необхідно натиснути на обраний продукт, після чого з'явиться діалогове вікно, яке відображено на рис. 4.10. Можна його відхилити та воно зачинеться, або погодитися, після чого у віртуального вихованця підніметься шкала здоров'я, що відображено на рис. 4.11, а продукт буде видалено з інвентаря.

Декілька разів на день, коли шкала здоров'я вихованця змінюється, застосунок буде надсилати повідомлення, для того щоб нагадувати про активність, та необхідність погодувати віртуального вихованця. Приклад повідомлення наведено на рис. 4.12.

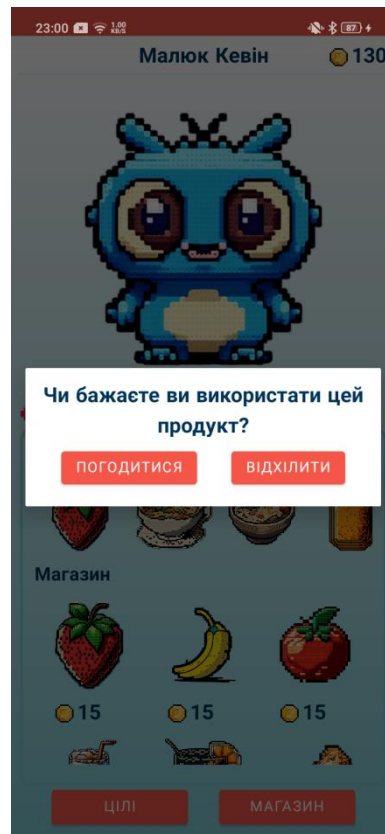


Рисунок 4.10 — Діалогове вікно для використання продукту з інвентаря



Рисунок 4.11 — Піднята шкала здоров'я

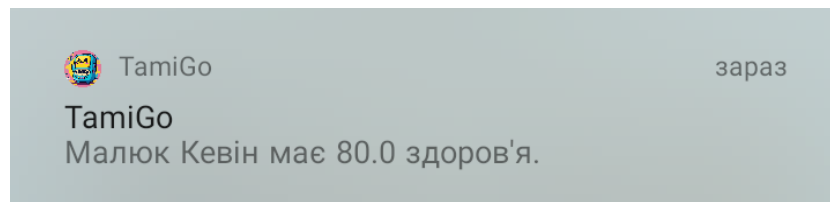


Рисунок 4.12 — Приклад повідомлення про стан здоров'я віртуального вихованця

Також у застосунку наявне бокове меню, яке відчиняється свайпом зліва направо. Бокове меню застосунку наведено на рис. 4.13. Воно створене для відкриття екрану інформації, що відображен на рис. 4.14, перезапуску гри, або повернення на головну сторінку. Якщо натиснути на кнопку «Головна» користувач буде повернен на домашній екран.

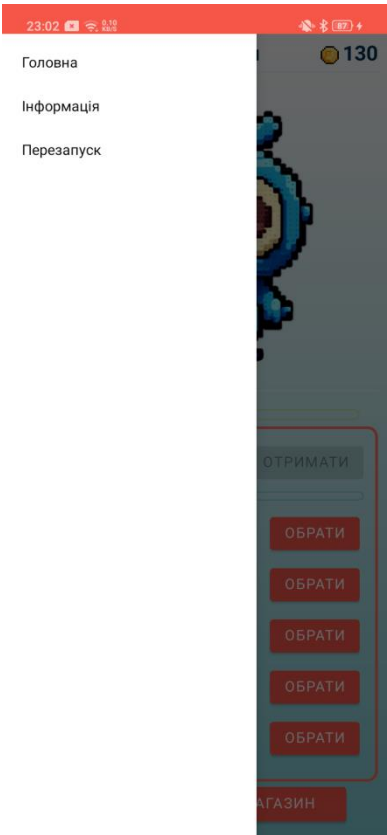


Рисунок 4.13 — Бокове меню застосунку



Розробник: Рудь Орина Костянтинівна
Дизайнер: Рудь Орина Костянтинівна
Версія додатку: 1

Рисунок 4.14 — Екран інформації

Якщо натиснути на «Перезапуск», ми побачимо діалогове вікно, який наведено на рис. 4.15, його можна відхилити, або погодитися, після чого дані гри будуть стерті і відкриється екран реєстрації.

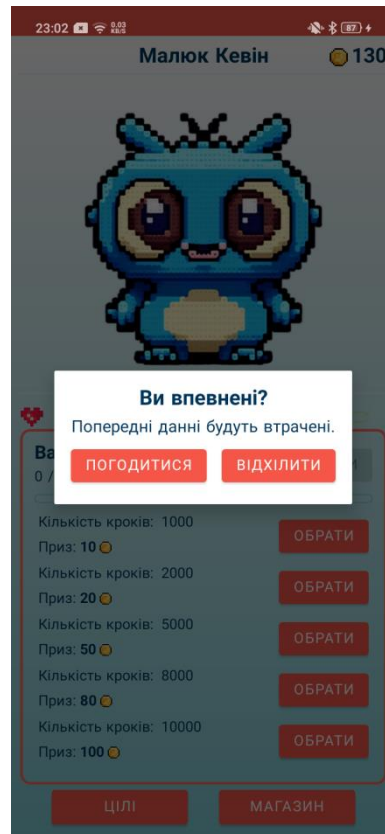


Рисунок 4.15 — Діалогове вікно для перезапуску гри

ВИСНОВКИ

У ході розробки мобільного застосунку шагоміра було отримано цінний досвід та результативний продукт. В результаті роботи над дипломною роботою було розроблено мобільний застосунок, призначений для відстеження активності користувача у вигляді шагоміра. Ігрова механіка тамагочі додала елементи розваги та мотивації, дозволяючи користувачеві отримувати монети за виконання цілей. Ці монети можуть бути використані для покупки продуктів, якими можна годувати віртуального вихованця.

У процесі розробки програми була використана мова програмування Kotlin, яка має переваги в плані безпеки, виразності та простоти використання. MVVM був обраний для забезпечення поділу обов'язків та покращення підтримуваності коду. Для впровадження залежностей у застосунок був обраний фреймворк Koin, який надає простий та гнучкий підхід у розробці. Це спростило процес налаштування та використання залежностей у проекті. Data Binding був використаний для зв'язування даних і уявлень у застосунку, що сприяє поліпшенню продуктивності та спрощенню роботи з інтерфейсом користувача. Застосунок розроблено з використанням сучасних технологій та інструментів, що забезпечує його ефективність, продуктивність та зручність використання.

Подальший розвиток програми може включати розширення функціональності, додавання нових ігрових елементів, покращення дизайну та оптимізацію продуктивності. Також можна розглянути можливість інтеграції з іншими сторонніми сервісами та платформами для подальшого поліпшення досвіду користувача.

В цілому, розробка мобільного застосунку шагоміра представляла цікавий та корисний досвід. Результати роботи дозволяють зробити висновок про те, що створення інноваційних та мотивуючих застосунків, заснованих на ігрових механіках, є актуальним та перспективним напрямом розробки мобільних застосунків.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Rouse M. Android SDK [Electronic resource] / M. Rouse. — Access mode: <https://www.techopedia.com/definition/4220/android-sdk>.
2. Health Connect [Electronic resource]. — Access mode: <https://developer.android.com/guide/health-and-fitness/health-connect#:~:text=Health%20Connect%20is%20an%20Android,device%20health%20and%20fitness%20data>.
3. Google Fit [Електрон. ресурс]. — Access mode: https://uk.wikipedia.org/wiki/Google_Fit.
4. Eckel B. Atomic Kotlin [Text] / B. Eckel, S. Isakova. — Mindview LLC, 2021. — 636 p.
5. Griffiths D. Head First Kotlin [Text] / D. Griffiths, D. Griffiths. — O'Reilly Media, 2019. — 480 p.
6. Jemerov D. Kotlin in Action [Text] / D. Jemerov, S. Isakova. — Manning, 2017. — 360 p.
7. MVP (deprecated) [Electronic resource]. — Access mode: <https://infinum.com/handbook/android/project-architecture/mvp-deprecated>.
8. Mainkar P. MVP (Model View Presenter) [Electronic resource] / P. Mainkar. — Access mode: <https://www.oreilly.com/library/view/expert-android-programming/9781786468956/49638756-a11b-4b7f-bd37-7b863209dc81.xhtml>.
9. Leiva A. MVP for Android: how to organize the presentation layer [Electronic resource] / A. Leiva. — Access mode: <http://antonioleiva.com/mvp-android/>.
10. Model-View-ViewModel [Electronic resource]. — Access mode: <https://learn.microsoft.com/en-us/dotnet/architecture/maui/mvvm>.
11. Mishra R. Difference Between MVP and MVVM Architecture Pattern in Android [Electronic resource] / R. Mishra. — Access mode:

<https://www.geeksforgeeks.org/difference-between-mvp-and-mvvm-architecture-pattern-in-android/>.

12. What is Java technology and why do I need it? [Electronic resource]. — Access mode: https://www.java.com/en/download/help/whatis_java.html.

13. What is Java? [Electronic resource]. — Access mode: https://aws.amazon.com/what-is/java/?nc1=h_ls.

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

15. Hilt [Electronic resource]. — Access mode: <https://dagger.dev/hilt/>.

16. The pragmatic Kotlin & Kotlin Multiplatform Dependency Injection framework [Electronic resource]. — Access mode: <https://insert-koin.io/>.

17. Giuliani A. Now In Android with Koin—part 1 [Electronic resource] / A. Giuliani. — Access mode: <https://proandroiddev.com/now-in-android-with-koin-part-1-2b871d8549f1>.

18. Ampire E. Dependency Injection with Koin [Electronic resource] / E. Ampire. — Access mode: <https://medium.com/swlh/dependency-injection-with-koin-6b6364dc8dba>.

19. Rykov M. Koin: Lightweight Dependency Injection [Electronic resource] / M. Rykov. — Access mode: <https://www.baeldung.com/kotlin/koin-di>.

20. Tanvir F. Dependency Injection with HILT in Android [Electronic resource] / F. Tanvir. — Access mode: <https://levelup.gitconnected.com/dependency-injection-with-hilt-in-android-73921b76c661>.

ДОДАТОК А
Технічне завдання

Вступ

Мобільний застосунок шагомір може використовуватись для персонального відстеження добової активності з підтримкою зацікавленості користувача до цього за допомогою віртуального вихованця.

A.1 Підстава до розробки

Підставою для розробки є завдання на дипломну кваліфікаційну роботу на тему «Розробка мобільного за стосунку шагоміру», затверджене наказом Національного університету «Запорізька політехніка» № 87 від 04 квітня 2023 р.

A.2 Призначення розробки

Мобільний застосунок призначений для персонального використання для підвищення зацікавленості користувачів в підвищенні середньодобової активності.

A.3 Основні вимоги до програми, що розробляється

A.3.1 Вимоги до функціональних характеристик

Мобільний застосунок шагомір має включати такі функції:

- отримання та відображення кількості кроків з Health Connect API;
- підтримання життя віртуального вихованця;
- механіка взаємодії з віртуальним вихованцем;
- додавання та відстеження виконання добових цілей;
- сповіщення користувача про стан віртуального вихованця та прогрес виконання добової цілі.

A.3.2 Вимоги до інтерфейсу програми

Інтерфейс мобільного застосунку шагоміру повинен бути зручним та зрозумілим для користувачів. Взаємодія з застосунком можлива за допомогою смартфона на базі Android.

A.3.3 Вимоги до надійності

Мобільний застосунок шагомір повинен забезпечити обробку винятків, проблемних ситуацій та забезпечувати стабільну роботу на різних мобільних платформах.

A.3.4 Умови експлуатації

Для використання мобільного застосунку шагоміра необхідно мати мобільний пристрій на операційній системі Android. Застосунок повинен мати можливість доступу до Health Connect API та має бути застосунок який відслідковує активність користувача та надсилає ці данні до Health Connect API, наприклад Google Fit, для обліку кількості кроків у застосунку.

A.3.5 Вимоги до складу та параметрів технічних засобів

Для експлуатації мобільного застосунку шагоміру необхідні такі засоби:

- мобільний пристрій з операційною системою Android;
- мінімальна версія Android 8.0;
- максимальна версія Android 13.0;
- щонайменше 200MB вільної пам'яті на пристрої.

A.4 Вхідні данні до роботи

Вхідними даними є данні введені користувачем, кількість кроків отриманих з Health Connect API.

Вихідними даними є зображення віртуального вихованця, допомога у постанові цілей та їх досягнення.

ДОДАТОК Б
Текст програми

BaseActivity.kt

```
package com.tamigo.base
```

```
import android.os.Bundle
```

```
import android.view.LayoutInflater
```

```
import androidx.appcompat.app.AppCompatActivity
```

```
import androidx.viewbinding.ViewBinding
```

```
typealias BindingInflation<T> = (LayoutInflater) -> T
```

```
abstract class BaseActivity<VB : ViewBinding> : AppCompatActivity() {
    private var _binding: VB? = null
    protected val binding: VB
        get() = _binding ?: throw IllegalStateException(EXCEPTION_MESSAGE)
    abstract val bindingInflation: BindingInflation<VB>
```

```
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        _binding = bindingInflation.invoke(layoutInflater)
        setContentView(binding.root)
    }
```

```
    override fun onDestroy() {
        super.onDestroy()
        _binding = null
    }
```

```
    companion object {
        const val EXCEPTION_MESSAGE =
            "Binding must not be null and can be accessed in range from onCreate to onDestroy"
    }
}
```

BaseFragment.kt

```
package com.tamigo.base
```

```
import android.os.Bundle
```

```
import android.view.LayoutInflater
```

```
import android.view.View
```

```

import android.view.ViewGroup
import androidx.fragment.app.Fragment
import androidx.viewbinding.ViewBinding

abstract class BaseFragment<VB : ViewBinding> : Fragment() {
    private var _binding: VB? = null
    protected val binding: VB
        get() = _binding
        ?: throw IllegalStateException(EXCEPTION_MESSAGE)
    abstract val bindingInflation: BindingInflation<VB>

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        _binding = bindingInflation.invoke(layoutInflater)
    }

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        return binding.root
    }

    override fun onDestroyView() {
        super.onDestroyView()
        _binding = null
    }

    companion object {
        const val EXCEPTION_MESSAGE =
            "Binding must not be null and can be accessed in range from onCreate to onDestroy"
    }
}

DL.kt

package com.tamigo.di

import android.content.Context
import android.content.SharedPreferences

```

```

import com.tamigo.main.MainViewModel
import com.tamigo.main.MainViewModelImpl
import com.tamigo.utils.managers.HealthConnectManager
import com.tamigo.navigation.MainRouter
import com.tamigo.navigation.Router
import com.tamigo.utils.preferences.Preferences
import com.tamigo.utils.preferences.PreferencesImpl
import com.tamigo.utils.viewModel.*
import org.koin.android.ext.koin.androidApplication
import org.koin.androidx.viewmodel.dsl.viewModel
import org.koin.dsl.bind
import org.koin.dsl.module

val navigationModule = module {
    single {
        MainRouter()
    } bind (Router::class)
}

val prefsRepositoryModule = module {
    single {
        getSharedPreferences(androidApplication())
    }
    single {
        PreferencesImpl(get())
    }.bind(Preferences::class)
}

val viewModelModule = module {
    viewModel<HomeViewModel> { HomeViewModelImpl(get(), get()) }
    viewModel<RegistrationViewModel> { RegistrationViewModelImpl(get(), get()) }
    viewModel<MainViewModel> { MainViewModelImpl(get(), get(), get()) }
    viewModel<TargetsViewModel> { TargetsViewModelImpl(get(), get()) }
    viewModel<ShopViewModel> { ShopViewModelImpl(get()) }
}

val managersModule = module {
    single {
        HealthConnectManager(androidApplication(), get())
    }
}

```

```
private fun getSharedPreferences(context: Context): SharedPreferences {
    return context.getSharedPreferences(PreferencesImpl.KEY, Context.MODE_PRIVATE)
}
```

```
val appModules = arrayListOf(
    navigationModule,
    prefsRepositoryModule,
    viewModelModule,
    managersModule
)
```

MainActivity.kt

```
package com.tamigo.main
```

```
import android.os.Build
import android.os.Bundle
import androidx.core.splashscreen.SplashScreen.Companion.installSplashScreen
import androidx.core.view.GravityCompat
import com.tamigo.R
import com.tamigo.base.BaseActivity
import com.tamigo.base.BindingInflation
import com.tamigo.databinding.ActivityMainBinding
import com.tamigo.utils.interfase.UpdateCoinsListener
import com.tamigo.utils.managers.HealthConnectManager
import com.tamigo.navigation.Router
import com.tamigo.ui.dialog.ConfirmationDialog
import com.tamigo.ui.home.HomeFragment
import com.tamigo.ui.home.HomeScreen
import org.koin.android.ext.android.inject
import org.koin.androidx.viewmodel.ext.android.viewModel
```

```
class MainActivity : BaseActivity<ActivityMainBinding>(), UpdateCoinsListener {
    override val bindingInflation: BindingInflation<ActivityMainBinding> =
        ActivityMainBinding::inflate
    private val router by inject<Router>()
    private val mainViewModel: MainViewModel by viewModel()
    private val healthConnectManager by inject<HealthConnectManager>()
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
```

```

setContentView(binding.root)
router.attachTo(this)

healthConnectManager.attach(this)
mainViewModel.checkHealthConnectStatusAndPermissions()

binding.navigation.setNavigationItemSelectedListener {
    when (it.itemId) {
        R.id.home -> {
            mainViewModel.navigate()
        }
        R.id.info -> {
            mainViewModel.navigateInfo()
        }
        R.id.restart -> {
            openConfirmationDialog()
        }
    }
    binding.drawerLayout.closeDrawer(GravityCompat.START)
    return@setNavigationItemSelectedListener true
}

if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.R)
    installSplashScreen().setKeepOnScreenCondition {
        false
    }

mainViewModel.navigate()
}

override fun onUpdateCoinsBalance() {
    val homeFragment = supportFragmentManager
        .findFragmentByTag(HomeScreen().tag) as HomeFragment
    homeFragment.updateCoinsBalance()
}

private fun openConfirmationDialog() {
    ConfirmationDialog {
        mainViewModel.onClickRestart()
    }.show(supportFragmentManager, ConfirmationDialog.TAG)
}

```

```
}
```

MainViewModel.kt

```
package com.tamigo.main
```

```
import androidx.lifecycle.ViewModel
import com.tamigo.utils.data.Target
import com.tamigo.utils.managers.HealthConnectManager
import com.tamigo.navigation.MainRouter
import com.tamigo.utils.preferences.Preferences
import com.tamigo.ui.home.HomeScreen
import com.tamigo.ui.info.InfoScreen
import com.tamigo.ui.registration.RegistrationScreen
```

```
abstract class MainViewModel : ViewModel() {
    abstract fun navigate()
    abstract fun checkHealthConnectStatusAndPermissions()
    abstract fun navigateInfo()
    abstract fun onClickRestart()
}
```

```
class MainViewModelImpl(
    private val router: MainRouter,
    private val preferences: Preferences,
    private val healthConnectManager: HealthConnectManager
) : MainViewModel() {
    override fun navigate() {
        if (!preferences.getTamiName().isEmpty())
            router.navigate(HomeScreen())
        else
            router.navigate(RegistrationScreen())
    }

    override fun checkHealthConnectStatusAndPermissions() {
        healthConnectManager.checkHealthConnect()
    }

    override fun navigateInfo() {
        router.navigate(InfoScreen())
    }
}
```

```

override fun onClickRestart() {
    preferences.setTamiName("")
    preferences.setTamiSkin(0)
    preferences.setProducts("")
    preferences.removeCoinsFromBalance(preferences.getCoinsBalance())
    preferences.setTarget(Target(0, 0))
    preferences.setFirstLaunch(true)
    preferences.setHealth(100f)
    router.navigate(RegistrationScreen())
}
}

```

TamiGoApplication.kt

```
package com.tamigo.main
```

```

import android.app.Application
import com.tamigo.di.appModules
import org.koin.android.ext.koin.androidContext
import org.koin.core.context.startKoin

```

```

class TamiGoApplication : Application() {

    override fun onCreate() {
        super.onCreate()
        startKoin {
            androidContext(this@TamiGoApplication)
            modules(appModules)
        }
    }
}

```

Command.kt

```
package com.tamigo.navigation
```

```

sealed class Command {
    object Back : Command()
    data class Navigate(val screen: Screen) : Command()
}

```

MainRouter.kt

```
package com.tamigo.navigation
```

```
import androidx.appcompat.app.AppCompatActivity
```

```
import androidx.fragment.app.FragmentActivity
```

```
import androidx.fragment.app.FragmentManager
```

```
import androidx.lifecycle.Lifecycle
```

```
import androidx.lifecycle.LifecycleEventObserver
```

```
import androidx.lifecycle.LifecycleOwner
```

```
class MainRouter : Router {
```

```
    private var fragmentManager: FragmentManager? = null
```

```
    private val autoDetacher: ActivityLifecycleDetacher? = null
```

```
    override fun attachTo(host: Any) {
```

```
        if (fragmentManager != null) {
```

```
            throw IllegalStateException("Router is already attached")
```

```
        }
```

```
        if (host is AppCompatActivity) {
```

```
            fragmentManager = host.supportFragmentManager
```

```
            autoDetachOnDestroy(host)
```

```
        } else {
```

```
            throw IllegalArgumentException(
```

```
                "Main router cannot be attached to instance of ${host.javaClass}. " +
```

```
                "It has to be ${FragmentManager::javaClass} subclass"
```

```
            )
```

```
        }
```

```
    }
```

```
    private fun autoDetachOnDestroy(activity: FragmentActivity) {
```

```
        val observer = ActivityLifecycleDetacher(activity = activity)
```

```
        activity.lifecycle.addObserver(observer)
```

```
    }
```

```
    override fun navigate(screen: Screen) {
```

```
        execute(Command.Navigate(screen))
```

```
    }
```

```
    override fun back() {
```

```
        execute(Command.Back)
```

```

    }

    private fun execute(command: Command) {
        val fragmentManager = fragmentManager ?: throw IllegalStateException(
            "Router cannot be used when fragment manager is not attached"
        )
        when (command) {
            Command.Back -> fragmentManager.popBackStackImmediate()
            is Command.Navigate -> command.screen.execute(fragmentManager)
        }
    }

    override fun detach() {
        autoDetacher?.release()
        fragmentManager = null
    }

    inner class ActivityLifecycleDetacher(
        private val activity: FragmentActivity
    ) : LifecycleEventObserver {

        override fun onStateChanged(source: LifecycleOwner, event: Lifecycle.Event) {
            if (event == Lifecycle.Event.ON_DESTROY) {
                detach()
                source.lifecycle.removeObserver(this)
            }
        }

        fun release() {
            activity.lifecycle.removeObserver(this)
        }
    }
}

```

Router.kt

```
package com.tamigo.navigation
```

```

interface Router {
    fun attachTo(host: Any)
    fun navigate(screen: Screen)
}

```

```

        fun back()
        fun detach()
    }

```

Screen.kt

```
package com.tamigo.navigation
```

```

import androidx.annotation.IdRes
import androidx.fragment.app.Fragment
import androidx.fragment.app.FragmentManager
import com.tamigo.R

```

```

abstract class Screen {
    abstract val tag: String

    abstract fun execute(fragmentManager: FragmentManager)

    protected fun FragmentManager.replace(
        fragment: Fragment,
        @IdRes container: Int = R.id.fragment_container,
        addToBackStack: Boolean = true
    ) {
        beginTransaction()
            .replace(container, fragment, tag)
            .apply {
                if (addToBackStack) {
                    addToBackStack(tag)
                }
            }
            .commit()
    }
}

```

FoodAdapter.kt

```
package com.tamigo.ui.adapter
```

```

import android.annotation.SuppressLint
import android.view.LayoutInflater
import android.view.ViewGroup

```

```

import androidx.recyclerview.widget.RecyclerView
import androidx.recyclerview.widget.RecyclerView.ViewHolder
import com.tamigo.utils.data.FoodItem
import com.tamigo.databinding.ItemFoodBinding
import com.tamigo.databinding.ItemShopBinding

class FoodAdapter(
    private val listFood: MutableMap<Int, List<FoodItem>?>,
    private val onItemClick: (FoodItem) -> Unit
) : RecyclerView.Adapter<ViewHolder>() {

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ViewHolder {
        return when (viewType) {
            SHOP_VIEW_HOLDER -> ShopViewHolder(
                ItemShopBinding.inflate(
                    LayoutInflater.from(parent.context),
                    parent,
                    false
                )
            )
            INVENTORY_VIEW_HOLDER -> InventoryViewHolder(
                ItemFoodBinding.inflate(
                    LayoutInflater.from(parent.context),
                    parent,
                    false
                )
            )
            else -> InventoryViewHolder(
                ItemFoodBinding.inflate(
                    LayoutInflater.from(parent.context),
                    parent,
                    false
                )
            )
        }
    }

    override fun onBindViewHolder(holder: ViewHolder, position: Int) {
        val item = listFood.values.first()
        if (item != null)
            when (holder) {

```

```

        is ShopViewHolder -> holder.bind(item[position]) {
            onItemClick(it)
        }
        is InventoryViewHolder -> holder.bind(item[position]) {
            onItemClick(it)
        }
    }
}

override fun getItemCount(): Int {
    val items = listFood.values.first()
    return if (items?.size != null)
        items.size
    else
        0
}

override fun getItemViewType(position: Int): Int {
    return listFood.keys.first()
}

@SuppressLint("NotifyDataSetChanged")
fun updateAdapter(items: List<FoodItem>?) {
    listFood[INVENTORY_VIEW_HOLDER] = items
    notifyDataSetChanged()
}

class ShopViewHolder(private val binding: ItemShopBinding) :
    ViewHolder(binding.root) {
    fun bind(item: FoodItem, onItemClick: (FoodItem) -> Unit) {
        binding.iconFood.setImageResource(item.foodSrcId)
        binding.cost.text = item.cost.toString()
        binding.root.setOnClickListener {
            onItemClick(item)
        }
    }
}

class InventoryViewHolder(private val binding: ItemFoodBinding) :
    ViewHolder(binding.root) {
    fun bind(item: FoodItem, onItemClick: (FoodItem) -> Unit) {

```

```

        binding.iconFood.setImageResource(item.foodSrcId)
        binding.root.setOnClickListener {
            onItemClick(item)
        }
    }
}

companion object {
    const val SHOP_VIEW_HOLDER = 1
    const val INVENTORY_VIEW_HOLDER = 2
}
}

TamiListAdapter.kt

package com.tamigo.ui.adapter

import android.annotation.SuppressLint
import android.graphics.Color
import android.view.LayoutInflater
import android.view.ViewGroup
import androidx.lifecycle.MutableLiveData
import androidx.recyclerview.widget.RecyclerView
import com.tamigo.R
import com.tamigo.databinding.ItemTamiListBinding

class TamiListAdapter(
    private val listTami: List<Int>,
    private val onItemClick: (Int) -> Unit
) : RecyclerView.Adapter<TamiListAdapter.TamiListViewHolder>() {

    private var selectedItem = MutableLiveData<Int>()

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): TamiListViewHolder {
        val binding = ItemTamiListBinding.inflate(
            LayoutInflater.from(parent.context),
            parent,
            false
        )
        return TamiListViewHolder(binding)
    }
}

```

```

@SuppressLint("NotifyDataSetChanged")
override fun onBindViewHolder(holder: TamiListViewHolder, position: Int) {
    holder.bind(listTami[position])
    holder.itemView.setOnClickListener {
        onItemClick(listTami[position])
        selectedItem.value = holder.adapterPosition
        notifyDataSetChanged()
    }
    if (selectedItem.value == position) {
        holder.itemView.setBackgroundResource(R.drawable.ui_background_border)
    } else
        holder.itemView.setBackgroundColor(Color.TRANSPARENT)
}

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

class TamiListViewHolder(private val binding: ItemTamiListBinding) :
    RecyclerView.ViewHolder(binding.root) {
    fun bind(item: Int) {
        binding.image.setImageResource(item)
    }
}
}

```

TargetsAdapter.kt

```
package com.tamigo.ui.adapter
```

```

import android.view.LayoutInflater
import android.view.ViewGroup
import androidx.recyclerview.widget.RecyclerView
import com.tamigo.utils.data.Target
import com.tamigo.databinding.ItemTargetsBinding

```

```

class TargetsAdapter(
    private val listTargets: List<Target>,
    private val onItemClick: (Target) -> Unit
) : RecyclerView.Adapter<TargetsAdapter.TamiListViewHolder>() {

```

```

override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): TamiListViewHolder {
    val binding = ItemTargetsBinding.inflate(
        LayoutInflater.from(parent.context),
        parent,
        false
    )
    return TamiListViewHolder(binding)
}

```

```

override fun onBindViewHolder(holder: TamiListViewHolder, position: Int) {
    holder.bind(listTargets[position]) {
        onItemClick(it)
    }
}

```

```

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

```

```

class TamiListViewHolder(private val binding: ItemTargetsBinding) :
    RecyclerView.ViewHolder(binding.root) {
    fun bind(item: Target, onClickChooseButton: (Target) -> Unit) {
        binding.prizeCoins.text = item.coins.toString()
        binding.numberSteps.text = item.steps.toString()
        binding.btnChoose.setOnClickListener {
            onClickChooseButton(item)
        }
    }
}

```

ConfirmationDialog.kt

```
package com.tamigo.ui.dialog
```

```

import android.os.Bundle
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import androidx.fragment.app.DialogFragment
import com.tamigo.databinding.DialogConfirmationBinding

```

```

class ConfirmationDialog(
    private val onCLickAgree: () -> Unit
) : DialogFragment() {

    private lateinit var binding: DialogConfirmationBinding

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        binding = DialogConfirmationBinding.inflate(inflater, container, false)
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        binding.btnAgree.setOnClickListener {
            dialog?.dismiss()
            onCLickAgree()
        }
        binding.btnCancel.setOnClickListener {
            dialog?.dismiss()
        }
    }

    companion object {
        val TAG: String = ConfirmationDialog::class.java.simpleName
    }
}

```

PurchaseDialog.kt

```
package com.tamigo.ui.dialog
```

```

import android.os.Bundle
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import androidx.fragment.app.DialogFragment

```

```

import com.tamigo.databinding.DialogPurchaseBinding

class PurchaseDialog(
    private val title: String,
    private val onCLickAgree: () -> Unit
) : DialogFragment() {

    private lateinit var binding: DialogPurchaseBinding

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        binding = DialogPurchaseBinding.inflate(inflater, container, false)
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        binding.title.text = title
        binding.btnAgree.setOnClickListener {
            dialog?.dismiss()
            onCLickAgree()
        }
        binding.btnCancel.setOnClickListener {
            dialog?.dismiss()
        }
    }

    companion object {
        val TAG: String = PurchaseDialog::class.java.simpleName
    }
}

```

FoodFragment.kt

```

package com.tamigo.ui.food

```

```

import android.annotation.SuppressLint
import android.content.Context

```

```

import android.content.Intent
import android.os.Bundle
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.Toast
import androidx.lifecycle.MutableLiveData
import androidx.lifecycle.LifecycleScope
import androidx.recyclerview.widget.GridLayoutManager
import com.tamigo.R
import com.tamigo.base.BaseFragment
import com.tamigo.base.BindingInflation
import com.tamigo.databinding.FragmentShopBinding
import com.tamigo.ui.adapter.FoodAdapter
import com.tamigo.ui.adapter.FoodAdapter.Companion.INVENTORY_VIEW_HOLDER
import com.tamigo.ui.adapter.FoodAdapter.Companion.SHOP_VIEW_HOLDER
import com.tamigo.ui.dialog.PurchaseDialog
import com.tamigo.utils.constant.Constants.PLUS_HEALTH_INTENT
import com.tamigo.utils.constant.Constants.shopList
import com.tamigo.utils.data.FoodItem
import com.tamigo.utils.interfase.UpdateCoinsListener
import com.tamigo.utils.receivers.HealthReceiver
import com.tamigo.utils.receivers.HealthReceiver.Companion.PLUS_HEALTH
import com.tamigo.utils.viewModel.ShopViewModel
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.launch
import org.koin.androidx.viewmodel.ext.android.viewModel

class FoodFragment : BaseFragment<FragmentShopBinding>() {
    override val bindingInflation: BindingInflation<FragmentShopBinding> =
        FragmentShopBinding::inflate

    private val viewModel: ShopViewModel by viewModel()
    private var updateCoinsListener: UpdateCoinsListener? = null

    private val inventory = MutableLiveData<List<FoodItem>?>()

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ) {

```

```

): View {
    return binding.root
}

@SuppressLint("NotifyDataSetChanged")
override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
    super.onViewCreated(view, savedInstanceState)
    val inventoryAdapter = FoodAdapter(
        mutableMapOf(Pair(INVENTORY_VIEW_HOLDER, inventory.value)),
        onItemClick = {
            openConfirmationDialog(AGREE_USE, it)
        }
    )
    val shopAdapter = FoodAdapter(
        mutableMapOf(Pair(SHOP_VIEW_HOLDER, shopList)),
        onItemClick = {
            openConfirmationDialog(AGREE_BUY, it)
        }
    )
    inventory.observe(viewLifecycleOwner) {
        inventoryAdapter.updateAdapter(it)
    }
    with(binding) {
        shop.layoutManager = GridLayoutManager(requireContext(), 3)
        inventory.adapter = inventoryAdapter
        shop.adapter = shopAdapter
    }
}

override fun onAttach(context: Context) {
    super.onAttach(context)
    updateCoinsListener = context as? UpdateCoinsListener
}

override fun onResume() {
    super.onResume()
    inventory.value = viewModel.getProductsFromInventory()
}

override fun onDetach() {
    super.onDetach()
}

```

```

        updateCoinsListener = null
    }

    private fun openConfirmationDialog(title: String, foodItem: FoodItem) {
        PurchaseDialog(
            title
        ) {
            when (title) {
                AGREE_BUY -> {
                    if (viewModel.isBuyProduct(foodItem.cost)) {
                        updateCoinsListener?.onUpdateCoinsBalance()
                        viewModel.setProductToInventory(listOf(foodItem))
                        inventory.postValue(viewModel.getProductsFromInventory())
                    } else {
                        Toast.makeText(
                            requireContext(),
                            R.string.not_enough_money,
                            Toast.LENGTH_SHORT
                        ).show()
                    }
                }
                AGREE_USE -> {
                    lifecycleScope.launch(Dispatchers.IO) {
                        viewModel.removeProductFromInventory(foodItem)
                        inventory.postValue(viewModel.getProductsFromInventory())
                        val intent = Intent(requireContext(), HealthReceiver::class.java)
                        intent.action = PLUS_HEALTH_INTENT
                        intent.putExtra(PLUS_HEALTH, foodItem.healthRecovery)
                        requireContext().sendBroadcast(intent)
                    }
                }
            }
        }.show(parentFragmentManager, PurchaseDialog.TAG)
    }

    companion object {
        private const val AGREE_BUY = "Чи бажаєте ви купити цей продукт?"
        private const val AGREE_USE = "Чи бажаєте ви використати цей продукт?"
    }
}

```

FoodScreen.kt

```
package com.tamigo.ui.food
```

```
import androidx.fragment.app.FragmentManager
```

```
import com.tamigo.R
```

```
import com.tamigo.navigation.Screen
```

```
class FoodScreen : Screen() {
```

```
    override val tag: String = FoodScreen::class.java.simpleName
```

```
    override fun execute(fragmentManager: FragmentManager) {
```

```
        fragmentManager.replace(FoodFragment(), container = R.id.home_fragment_container)
```

```
    }
```

```
}
```

HomeFragment.kt

```
package com.tamigo.ui.home
```

```
import android.os.Bundle
```

```
import android.view.LayoutInflater
```

```
import android.view.View
```

```
import android.view.ViewGroup
```

```
import com.tamigo.base.BaseFragment
```

```
import com.tamigo.base.BindingInflation
```

```
import com.tamigo.databinding.FragmentHomeBinding
```

```
import com.tamigo.utils.receivers.HealthReceiver
```

```
import com.tamigo.utils.viewModel.HomeViewModel
```

```
import org.koin.androidx.viewmodel.ext.android.viewModel
```

```
class HomeFragment : BaseFragment<FragmentHomeBinding>() {
```

```
    override val bindingInflation: BindingInflation<FragmentHomeBinding> =
        FragmentHomeBinding::inflate
```

```
    private val homeViewModel: HomeViewModel by viewModel()
```

```
    override fun onCreateView(
```

```
        inflater: LayoutInflater, container: ViewGroup?,
```

```
        savedInstanceState: Bundle?
```

```

): View {
    return binding.root
}

override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
    super.onViewCreated(view, savedInstanceState)
    homeViewModel.registerPrefsListener()
    homeViewModel.openTargetsFragment()
    with(binding) {
        txtName.text = homeViewModel.getTamiName()
        skin.setImageResource(homeViewModel.getTamiSkin())
        btnTarget.setOnClickListener {
            homeViewModel.openTargetsFragment()
        }
        btnShop.setOnClickListener {
            homeViewModel.openShopFragment()
        }
        coinBalance.text = homeViewModel.getCoins()
        progressHealth.progress = homeViewModel.getHealthFromPref().toInt()
        homeViewModel.health.observe(viewLifecycleOwner) {
            progressHealth.progress = it.toInt()
        }
    }
}

override fun onResume() {
    super.onResume()
    if (homeViewModel.isNeedStartAlarm()) {
        HealthReceiver.setAlarm(requireContext())
    }
}

override fun onDestroy() {
    super.onDestroy()
    homeViewModel.unregisterPrefsListener()
}

fun updateCoinsBalance() {
    binding.coinBalance.text = homeViewModel.getCoins()
}
}

```

HomeScreen.kt

```
package com.tamigo.ui.home
```

```
import androidx.fragment.app.FragmentManager
```

```
import com.tamigo.navigation.Screen
```

```
class HomeScreen : Screen() {
```

```
    override val tag: String = HomeScreen::class.java.simpleName
```

```
    override fun execute(fragmentManager: FragmentManager) {
```

```
        fragmentManager.replace(HomeFragment(), addToBackStack = false)
```

```
    }
```

```
}
```

InfoFragment.kt

```
package com.tamigo.ui.info
```

```
import android.os.Bundle
```

```
import android.view.LayoutInflater
```

```
import android.view.View
```

```
import android.view.ViewGroup
```

```
import com.tamigo.base.BaseFragment
```

```
import com.tamigo.base.BindingInflation
```

```
import com.tamigo.databinding.FragmentInfoBinding
```

```
class InfoFragment : BaseFragment<FragmentInfoBinding>() {
```

```
    override val bindingInflation: BindingInflation<FragmentInfoBinding> =
```

```
        FragmentInfoBinding::inflate
```

```
    override fun onCreateView(
```

```
        inflater: LayoutInflater,
```

```
        container: ViewGroup?,
```

```
        savedInstanceState: Bundle?
```

```
): View {
```

```
    return binding.root
```

```
}
```

```
}
```

InfoScreen.kt

```
package com.tamigo.ui.info

import androidx.fragment.app.FragmentManager
import com.tamigo.navigation.Screen

class InfoScreen : Screen() {
    override val tag: String = InfoScreen::class.java.simpleName

    override fun execute(fragmentManager: FragmentManager) {
        fragmentManager.replace(InfoFragment())
    }
}
```

RegistrationFragment.kt

```
package com.tamigo.ui.registration

import android.os.Bundle
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import androidx.recyclerview.widget.GridLayoutManager
import com.tamigo.R
import com.tamigo.base.BaseFragment
import com.tamigo.base.BindingInflation
import com.tamigo.databinding.FragmentRegistrationBinding
import com.tamigo.ui.adapter.TamiListAdapter
import com.tamigo.utils.constant.Constants.tamiList
import com.tamigo.utils.viewModel.RegistrationViewModel
import org.koin.androidx.viewmodel.ext.android.viewModel

class RegistrationFragment : BaseFragment<FragmentRegistrationBinding>() {

    override val bindingInflation: BindingInflation<FragmentRegistrationBinding> =
        FragmentRegistrationBinding::inflate

    private val registrationViewModel: RegistrationViewModel by viewModel()

    override fun onCreateView(
```

```

        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        with(binding) {
            var selectedSkin: Int = R.drawable.tami_blue
            recyclerTamiList.layoutManager = GridLayoutManager(requireContext(), 2)
            recyclerTamiList.adapter = TamiListAdapter(listTami = tamiList) {
                selectedSkin = it
            }
            button.setOnClickListener {
                registrationViewModel.saveInfoAboutTami(etTamiName.text.toString(), selectedSkin)
                registrationViewModel.navigateToHomeFragment()
            }
        }
    }
}

```

RegistrationScreen.kt

```
package com.tamigo.ui.registration
```

```
import androidx.fragment.app.FragmentManager
```

```
import com.tamigo.navigation.Screen
```

```

class RegistrationScreen : Screen() {
    override val tag: String = RegistrationScreen::class.java.simpleName

    override fun execute(fragmentManager: FragmentManager) {
        fragmentManager.replace(RegistrationFragment(), addToBackStack = false)
    }
}

```

TargetsFragment.kt

```
package com.tamigo.ui.targets

import android.content.Context
import android.os.Bundle
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import androidx.lifecycle.MutableLiveData
import com.tamigo.base.BaseFragment
import com.tamigo.base.BindingInflation
import com.tamigo.databinding.FragmentTargetsBinding
import com.tamigo.ui.adapter.TargetsAdapter
import com.tamigo.ui.dialog.ConfirmationDialog
import com.tamigo.utils.constant.Constants
import com.tamigo.utils.data.Target
import com.tamigo.utils.interfase.UpdateCoinsListener
import com.tamigo.utils.viewModel.TargetsViewModel
import org.koin.androidx.viewmodel.ext.android.viewModel

class TargetsFragment : BaseFragment<FragmentTargetsBinding>() {
    override val bindingInflation: BindingInflation<FragmentTargetsBinding> =
        FragmentTargetsBinding::inflate
    private val viewModel: TargetsViewModel by viewModel()

    private var target = MutableLiveData<Target?>()
    private var updateCoinsListener: UpdateCoinsListener? = null

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        target.value = viewModel.getTarget()
        with(binding) {

```

```

viewModel.currentSteps.observe(viewLifecycleOwner) {
    txtCurrentSteps.text = it.toString()
    progressTarget.progress = getProgress(it, target.value?.steps).toInt()
    if (it == 0L)
        btnGetPrize.isEnabled = false
    else
        btnGetPrize.isEnabled = it.toString() >= target.value?.steps.toString()
}
target.observe(viewLifecycleOwner) {
    txtTargetSteps.text = (it?.steps ?: "0").toString()
}

targetsList.adapter = TargetsAdapter(
    listTargets = Constants.targetsList,
    onItemClick = {
        openConfirmationDialog {
            viewModel.setTarget(it)
            target.value = it
        }
    }
)
btnGetPrize.setOnClickListener {
    viewModel.setCoinsToBalance(target.value?.coins)
    updateCoinsListener?.onUpdateCoinsBalance()
    removeTarget()
}
}

override fun onAttach(context: Context) {
    super.onAttach(context)
    updateCoinsListener = context as? UpdateCoinsListener
}

override fun onResume() {
    super.onResume()
    viewModel.getSteps()
}

override fun onDetach() {
    super.onDetach()

```

```

        updateCoinsListener = null
    }

    private fun openConfirmationDialog(onClickAgree: () -> Unit) {
        ConfirmationDialog {
            onClickAgree()
        }.show(parentFragmentManager, ConfirmationDialog.TAG)
    }

    private fun getProgress(currentSteps: Long, targetSteps: Int?): Double {
        return when {
            currentSteps == 0L || targetSteps == null -> {
                0.0
            }
            else -> {
                (currentSteps.toDouble() / targetSteps.toDouble()) * 100
            }
        }
    }

    private fun removeTarget() {
        binding.btnGetPrize.isEnabled = false
        binding.progressTarget.progress = 0
        binding.txtTargetSteps.text = ""
        val emptyTarget = Target(0, 0)
        target.value = emptyTarget
        viewModel.setTarget(emptyTarget)
    }
}

```

TargetsScreen.kt

```

package com.tamigo.ui.targets

import androidx.fragment.app.FragmentManager
import com.tamigo.R
import com.tamigo.navigation.Screen

class TargetsScreen : Screen() {
    override val tag: String = TargetsScreen::class.java.simpleName
}

```

```

    override fun execute(fragmentManager: FragmentManager) {
        fragmentManager.replace(TargetsFragment(), container = R.id.home_fragment_container)
    }
}

```

Constants.kt

```
package com.tamigo.utils.constant
```

```

import com.tamigo.R
import com.tamigo.utils.data.FoodItem
import com.tamigo.utils.data.Target

```

```

object Constants {
    val tamiList = listOf(
        R.drawable.tami_blue,
        R.drawable.tami_dark_blue,
        R.drawable.tami_garnet,
        R.drawable.tami_green,
        R.drawable.tami_jade,
        R.drawable.tami_lime,
        R.drawable.tami_mint,
        R.drawable.tami_pink,
        R.drawable.tami_purple,
        R.drawable.tami_red,
        R.drawable.tami_violet,
        R.drawable.tami_yellow
    )

```

```

    val targetsList = listOf(
        Target(1000, 10),
        Target(2000, 20),
        Target(5000, 50),
        Target(8000, 80),
        Target(10000, 100),
        Target(12000, 120),
        Target(15000, 150),
        Target(18000, 180),
        Target(20000, 200),
        Target(25000, 250),
        Target(50000, 500),
    )

```

```

    Target(100000, 1000)
)

```

```

val shopList = listOf(
    FoodItem(R.drawable.food_strawberry, 15, 5),
    FoodItem(R.drawable.food_banana, 15, 5),
    FoodItem(R.drawable.food_tomato, 15, 5),
    FoodItem(R.drawable.food_coke, 30, 10),
    FoodItem(R.drawable.food_orange_juice, 30, 10),
    FoodItem(R.drawable.food_chicken, 60, 25),
    FoodItem(R.drawable.food_soup, 60, 25),
    FoodItem(R.drawable.food_noodles, 60, 25),
    FoodItem(R.drawable.food_burger, 100, 50),
    FoodItem(R.drawable.food_pizza, 100, 50),
    FoodItem(R.drawable.food_cake, 200, 100),
    FoodItem(R.drawable.food_cookie, 200, 100)
)

```

```

const val MINUS_HEALTH_INTENT = "MINUS_HEALTH_INTENT"
const val PLUS_HEALTH_INTENT = "PLUS_HEALTH_INTENT"
}

```

FoodItem.kt

```

package com.tamigo.utils.data

```

```

data class FoodItem(
    val foodSrcId: Int,
    val cost: Int,
    val healthRecovery: Int
)

```

Target.kt

```

package com.tamigo.utils.data

```

```

data class Target(
    val steps: Int,
    val coins: Int
)

```

UpdateCoinsListener.kt

```
package com.tamigo.utils.interfase
```

```
interface UpdateCoinsListener {
    fun onUpdateCoinsBalance()
}
```

HealthConnectManager.kt

```
package com.tamigo.utils.managers
```

```
import android.content.Context
import android.content.Intent
import android.net.Uri
import androidx.activity.ComponentActivity
import androidx.health.connect.client.HealthConnectClient
import androidx.health.connect.client.PermissionController
import androidx.health.connect.client.permission.HealthPermission
import androidx.health.connect.client.records.StepsRecord
import androidx.health.connect.client.request.ReadRecordsRequest
import androidx.health.connect.client.time.TimeRangeFilter
import com.tamigo.utils.preferences.Preferences
import kotlinx.coroutines.CompletableDeferred
import kotlinx.coroutines.CoroutineScope
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.launch
import java.time.ZonedDateTime
import java.time.temporal.ChronoUnit
```

```
class HealthConnectManager(
    private val context: Context,
    private val preferences: Preferences
) {
    private var activity: ComponentActivity? = null
    private var permissionRequest: SuspendActivityResultLauncher? = null

    fun attach(activity: ComponentActivity) {
        this.activity = activity
        permissionRequest = SuspendActivityResultLauncher(
            activity = activity
        )
    }
}
```

```

    )
}

fun checkHealthConnect() {
    val availabilityStatus = HealthConnectClient.sdkStatus(context)
    if (availabilityStatus == HealthConnectClient.SDK_UNAVAILABLE) {
        return
    }
    if (availabilityStatus == HealthConnectClient.SDK_UNAVAILABLE_PROVIDER_UPDATE_REQUIRED) {
        context.startActivity(
            Intent(Intent.ACTION_VIEW).apply {
                setPackage(MARKET_PACKAGE)
                data = Uri.parse(HEALTH_CONNECT_URI)
                putExtra(OVERLAY, true)
                putExtra(CALLER_ID, context.packageName)
            }
        )
        return
    }
    val healthConnectClient = HealthConnectClient.getOrCreate(context)
    checkPermissions(healthConnectClient)
}

suspend fun getSteps(): Long {
    val now = ZonedDateTime.now()
    val startTargetTime: ZonedDateTime = try {
        ZonedDateTime.parse(preferences.getStartTargetTime())
    } catch (e: java.lang.Exception) {
        now.truncatedTo(ChronoUnit.DAYS)
    }
    val timeRangeFilter = TimeRangeFilter.between(
        startTargetTime.toLocalDateTime(),
        now.toLocalDateTime()
    )

    val stepsRecordRequest = ReadRecordsRequest(StepsRecord::class, timeRangeFilter)
    val healthConnectClient = HealthConnectClient.getOrCreate(context)
    return healthConnectClient.readRecords(stepsRecordRequest)
        .records.sumOf { it.count }
}

```

```
private fun checkPermissions(healthConnectClient: HealthConnectClient?) {
    CoroutineScope(Dispatchers.IO).launch {
        val granted = healthConnectClient?.permissionController?.getGrantedPermissions()
        if (granted?.containsAll(permissions) == false) {
            permissionRequest?.await(permissions)
        }
    }
}
```

```
private class SuspendActivityResultLauncher(
    activity: ComponentActivity
) {
    private var result: CompletableDeferred<Set<String>>? = null
```

```
    private val launcher = activity.registerForActivityResult(
        PermissionController
            .createRequestPermissionResultContract()
    ) {
        result?.complete(it)
    }
```

```
suspend fun await(input: Set<String>): Set<String> {
    result = CompletableDeferred()
    launcher.launch(input)
    return result?.await() ?: throw IllegalStateException(EXCEPTION_MESSAGE)
}
}
```

```
companion object {
    private const val HEALTH_CONNECT_URI =
        "market://details?url=healthconnect%3A%2F%2Fonboarding"
    private const val MARKET_PACKAGE = "com.android.vending"

    private const val OVERLAY = "overlay"
    private const val CALLER_ID = "callerId"

    private const val EXCEPTION_MESSAGE = "Inconsistent result state"

    private val permissions = setOf(
        HealthPermission.getReadPermission(StepsRecord::class),
        HealthPermission.getWritePermission(StepsRecord::class)
```

```

    )
}

}

```

Preferences.kt

```
package com.tamigo.utils.preferences
```

```
import android.content.SharedPreferences
import androidx.core.content.edit
import com.google.gson.Gson
import com.tamigo.R
import com.tamigo.utils.data.Target
import java.time.ZonedDateTime
```

```
interface Preferences {
    fun registerListener(listener: SharedPreferences.OnSharedPreferenceChangeListener)
    fun unregisterListener(listener: SharedPreferences.OnSharedPreferenceChangeListener)
    fun setTamiName(name: String)
    fun getTamiName(): String?
    fun setTamiSkin(skinResource: Int)
    fun getTamiSkin(): Int
    fun setTarget(target: Target)
    fun getTarget(): Target?
    fun setCoinsToBalance(coins: Int)
    fun removeCoinsFromBalance(coins: Int)
    fun getCoinsBalance(): Int
    fun setProducts(products: String)
    fun getProducts(): String?
    fun setFirstLaunch(isFirst: Boolean)
    fun isFirstLaunch(): Boolean
    fun setStartTargetTime(now: ZonedDateTime)
    fun getStartTargetTime(): String?
    fun setHealth(value: Float)
    fun getHealth(): Float
}
```

```
class PreferencesImpl(
    private val prefs: SharedPreferences
) : Preferences {
```

```

override fun registerListener(listener: SharedPreferences.OnSharedPreferenceChangeListener) {
    prefs.registerOnSharedPreferenceChangeListener(listener)
}

override fun unregisterListener(listener: SharedPreferences.OnSharedPreferenceChangeListener) {
    prefs.unregisterOnSharedPreferenceChangeListener(listener)
}

override fun setTamiName(name: String) {
    prefs.edit {
        putString(TAMI_NAME, name)
    }
}

override fun getTamiName(): String? {
    return prefs.getString(TAMI_NAME, "")
}

override fun setTamiSkin(skinResource: Int) {
    prefs.edit {
        putInt(TAMI_SKIN, skinResource)
    }
}

override fun getTamiSkin(): Int {
    return prefs.getInt(TAMI_SKIN, R.drawable.tami_blue)
}

override fun setTarget(target: Target) {
    prefs.edit {
        putString(TARGET, Gson().toJson(target))
    }
}

override fun getTarget(): Target? {
    return Gson().fromJson(prefs.getString(TARGET, ""), Target::class.java)
}

override fun setCoinsToBalance(coins: Int) {
    prefs.edit {
        putInt(COINS_BALANCE, prefs.getInt(COINS_BALANCE, 0) + coins)
    }
}

```

```

    }
}

override fun removeCoinsFromBalance(coins: Int) {
    prefs.edit {
        putInt(COINS_BALANCE, prefs.getInt(COINS_BALANCE, 0) - coins)
    }
}

override fun getCoinsBalance(): Int {
    return prefs.getInt(COINS_BALANCE, 0)
}

override fun setProducts(products: String) {
    prefs.edit {
        putString(PRODUCTS, products)
    }
}

override fun getProducts(): String? {
    return prefs.getString(PRODUCTS, "[{}]")
}

override fun setFirstLaunch(isFirst: Boolean) {
    prefs.edit {
        putBoolean(FIRST_LAUNCH, isFirst)
    }
}

override fun isFirstLaunch(): Boolean {
    return prefs.getBoolean(FIRST_LAUNCH, true)
}

override fun setStartTargetTime(now: ZonedDateTime) {
    prefs.edit {
        putString(START_TARGET, now.toString())
    }
}

override fun getStartTargetTime(): String? {
    return prefs.getString(START_TARGET, "")
}

```

```

    }

    override fun setHealth(value: Float) {
        prefs.edit {
            putFloat(HEALTH, value)
        }
    }

    override fun getHealth(): Float {
        return prefs.getFloat(HEALTH, 100f)
    }

    companion object {
        const val KEY = "tami_prefs"

        const val TAMI_NAME = "tami_name"
        const val TAMI_SKIN = "tami_skin"
        const val TARGET = "target"
        const val COINS_BALANCE = "coins_balance"
        const val PRODUCTS = "products_inventory"
        const val FIRST_LAUNCH = "is_first_launch"
        const val START_TARGET = "time_when_start_target"
        const val HEALTH = "tami_health"
    }
}

```

HealthReceiver.kt

```

package com.tamigo.utils.receivers

import android.annotation.SuppressLint
import android.app.AlarmManager
import android.app.NotificationChannel
import android.app.NotificationManager
import android.app.PendingIntent
import android.content.BroadcastReceiver
import android.content.Context
import android.content.Intent
import android.content.SharedPreferences
import androidx.core.app.NotificationCompat
import androidx.core.content.edit

```

```

import com.tamigo.R
import com.tamigo.ui.registration.RegistrationFragment
import com.tamigo.utils.constant.Constants.MINUS_HEALTH_INTENT
import com.tamigo.utils.constant.Constants.PLUS_HEALTH_INTENT
import com.tamigo.utils.preferences.PreferencesImpl
import com.tamigo.utils.preferences.PreferencesImpl.Companion.COINS_BALANCE
import com.tamigo.utils.preferences.PreferencesImpl.Companion.FIRST_LAUNCH
import com.tamigo.utils.preferences.PreferencesImpl.Companion.HEALTH
import com.tamigo.utils.preferences.PreferencesImpl.Companion.PRODUCTS
import com.tamigo.utils.preferences.PreferencesImpl.Companion.TAMI_NAME
import com.tamigo.utils.preferences.PreferencesImpl.Companion.TAMI_SKIN
import com.tamigo.utils.preferences.PreferencesImpl.Companion.TARGET

class HealthReceiver : BroadcastReceiver() {
    override fun onReceive(context: Context?, intent: Intent?) {
        val pref = context?.getSharedPreferences(PreferencesImpl.KEY, Context.MODE_PRIVATE)
        if (pref != null) {
            when (intent?.action) {
                MINUS_HEALTH_INTENT -> {
                    minusHealth(context, pref)

                }
                PLUS_HEALTH_INTENT -> {
                    val value = intent.extras?.getInt(PLUS_HEALTH)
                    if (value != null) {
                        plusHealth(value, pref)
                    }
                }
            }
        }
        when (pref.getFloat(HEALTH, 100f)) {
            in 90f..91f, in 80f..81f, in 70f..71f, in 60f..61f, in 50f..51f,
            in 40f..41f, in 30f..31f, in 20f..21f, in 10f..11f -> {
                sendNotification(
                    context,
                    "${pref.getString(TAMI_NAME, APP_NAME)} має ${
                        pref.getFloat(
                            HEALTH,
                            100f
                        )
                    } здоров'я."
                )
            }
        }
    }
}

```

```

        }
    }
}

private fun minusHealth(context: Context?, pref: SharedPreferences) {
    var health = pref.getFloat(HEALTH, MAX_HEALTH).toDouble()
    health -= PROPORTION
    if (health > MIN_HEALTH) {
        pref.edit { putFloat(HEALTH, health.toFloat()) }
        setAlarm(context)
    } else {
        clearGame(context, pref)
        if (context != null) {
            sendNotification(context, "Ти програв. Почни спочатку.")
        }
    }
}

private fun plusHealth(value: Int, pref: SharedPreferences) {
    var health = pref.getFloat(HEALTH, MAX_HEALTH)
    health += value
    if (health > MAX_HEALTH) {
        health = MAX_HEALTH
    }
    pref.edit { putFloat(HEALTH, health) }
}

private fun clearGame(context: Context?, pref: SharedPreferences) {
    pref.edit {
        putString(TAMI_NAME, "")
        putInt(TAMI_SKIN, 0)
        putString(PRODUCTS, "")
        putInt(COINS_BALANCE, 0)
        putString(TARGET, "")
        putBoolean(FIRST_LAUNCH, true)
        putFloat(HEALTH, 100f)
    }
    context?.startActivity(Intent(context, RegistrationFragment::class.java))
}

```

```

private fun sendNotification(context: Context, text: String) {
    val notificationManager = context.getSystemService(
        Context.NOTIFICATION_SERVICE
    ) as NotificationManager

    val importance = NotificationManager.IMPORTANCE_DEFAULT
    val channel = NotificationChannel(CHANNEL_ID, CHANNEL_NAME, importance)
    notificationManager.createNotificationChannel(channel)

    val notificationBuilder = NotificationCompat.Builder(context, CHANNEL_ID)
        .setSmallIcon(R.drawable.app_logo)
        .setContentTitle(APP_NAME)
        .setContentText(text)
        .setPriority(NotificationCompat.PRIORITY_DEFAULT)
        .setAutoCancel(true)

    notificationManager.notify(NOTIFICATION_ID, notificationBuilder.build())
}

companion object {
    private const val MAX_HEALTH = 100f
    private const val MIN_HEALTH = 0f
    private const val FINAL_TIME = 72f
    private const val PROPORTION = MAX_HEALTH / FINAL_TIME
    private const val CHANNEL_ID = "tami_notification_channel_id"
    private const val CHANNEL_NAME = "TamiGo Channel"
    private const val APP_NAME = "TamiGo"
    private const val NOTIFICATION_ID = 1

    const val PLUS_HEALTH = "plus_health"

    @SuppressWarnings("UnspecifiedImmutableFlag")
    fun setAlarm(context: Context?) {
        val alarmManager = context?.getSystemService(Context.ALARM_SERVICE) as? AlarmManager
        val alarmIntent = Intent(context, HealthReceiver::class.java)
        alarmIntent.action = MINUS_HEALTH_INTENT
        val pendingIntent =
            PendingIntent.getBroadcast(
                context,
                0,
                alarmIntent,

```

```

        PendingIntent.FLAG_UPDATE_CURRENT
    )
    val triggerTime = System.currentTimeMillis() + (60 * 60 * 1000)
    alarmManager?.setExact(AlarmManager.RTC_WAKEUP, triggerTime, pendingIntent)
}
}
}

```

HomeViewModel.kt

```

package com.tamigo.utils.viewModel

import android.content.SharedPreferences
import androidx.lifecycle.MutableLiveData
import androidx.lifecycle.ViewModel
import com.tamigo.navigation.MainRouter
import com.tamigo.ui.food.FoodScreen
import com.tamigo.ui.registration.RegistrationScreen
import com.tamigo.ui.targets.TargetsScreen
import com.tamigo.utils.preferences.Preferences
import com.tamigo.utils.preferences.PreferencesImpl

abstract class HomeViewModel : ViewModel() {
    abstract val health: MutableLiveData<Float>
    abstract fun getTamiName(): String
    abstract fun getTamiSkin(): Int
    abstract fun navigateToRegistrationScreen()
    abstract fun openTargetsFragment()
    abstract fun openShopFragment()
    abstract fun getCoins(): String
    abstract fun isNeedStartAlarm(): Boolean
    abstract fun registerPrefsListener()
    abstract fun unregisterPrefsListener()
    abstract fun getHealthFromPref(): Float
}

class HomeViewModelImpl(
    private val preferences: Preferences,
    private val router: MainRouter,
) : HomeViewModel() {
    override val health = MutableLiveData<Float>()
}

```

```

private val listener = SharedPreferences.OnSharedPreferenceChangeListener { _, key ->
    if (key == PreferencesImpl.HEALTH) {
        health.value = preferences.getHealth()
    }
}

override fun getTamiName(): String {
    return preferences.getTamiName() ?: "Tami"
}

override fun getTamiSkin(): Int {
    return preferences.getTamiSkin()
}

override fun navigateToRegistrationScreen() {
    router.navigate(RegistrationScreen())
}

override fun openTargetsFragment() {
    router.navigate(TargetsScreen())
}

override fun openShopFragment() {
    router.navigate(FoodScreen())
}

override fun getCoins(): String {
    return preferences.getCoinsBalance().toString()
}

override fun isNeedStartAlarm(): Boolean {
    val isNeed = preferences.isFirstLaunch()
    if (isNeed) {
        preferences.setFirstLaunch(false)
        preferences.setHealth(100f)
    }
    return isNeed
}

override fun registerPrefsListener() {

```

```

        preferences.registerListener(listener)
    }

    override fun unregisterPrefsListener() {
        preferences.unregisterListener(listener)
    }

    override fun getHealthFromPref(): Float {
        return preferences.getHealth()
    }
}

RegistrationViewModel.kt

package com.tamigo.utils.viewModel

import androidx.lifecycle.ViewModel
import com.tamigo.navigation.MainRouter
import com.tamigo.ui.home.HomeScreen
import com.tamigo.utils.preferences.Preferences

abstract class RegistrationViewModel : ViewModel() {
    abstract fun navigateToHomeFragment()
    abstract fun saveInfoAboutTami(name: String, skin: Int)
}

class RegistrationViewModelImpl(
    private val preferences: Preferences,
    private val router: MainRouter
) : RegistrationViewModel() {

    override fun navigateToHomeFragment() {
        router.navigate(HomeScreen())
    }

    override fun saveInfoAboutTami(name: String, skin: Int) {
        preferences.setTamiName(name)
        preferences.setTamiSkin(skin)
    }
}

```

ShopViewModel.kt

```
package com.tamigo.utils.viewModel
```

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

```
import com.google.gson.Gson
```

```
import com.google.gson.reflect.TypeToken
```

```
import com.tamigo.utils.data.FoodItem
```

```
import com.tamigo.utils.preferences.Preferences
```

```
abstract class ShopViewModel : ViewModel() {
    abstract fun setProductToInventory(product: List<FoodItem>)
    abstract fun getProductsFromInventory(): List<FoodItem>?
    abstract fun removeProductFromInventory(product: FoodItem)
    abstract fun isBuyProduct(cost: Int): Boolean
}
```

```
class ShopViewModelImpl(
```

```
    private val preferences: Preferences
```

```
): ShopViewModel() {
```

```
    private val gson = Gson()
```

```
    private val listType = object : TypeToken<List<FoodItem>>() {}.type
```

```
    override fun setProductToInventory(product: List<FoodItem>) {
```

```
        val currentInventory = preferences.getProducts()
```

```
        if (currentInventory.isNullOrEmpty())
```

```
            preferences.setProducts(gson.toJson(product))
```

```
        else {
```

```
            val inventoryData: MutableList<FoodItem> = gson.fromJson(currentInventory, listType)
```

```
            inventoryData.add(product.first())
```

```
            preferences.setProducts(gson.toJson(inventoryData))
```

```
        }
```

```
    }
```

```
    override fun getProductsFromInventory(): List<FoodItem>? {
```

```
        return gson.fromJson(preferences.getProducts(), listType)
```

```
    }
```

```
    override fun removeProductFromInventory(product: FoodItem) {
```

```
        val currentInventory = preferences.getProducts()
```

```
        if (!currentInventory.isNullOrEmpty()) {
```

```

        val inventoryData: MutableList<FoodItem> = gson.fromJson(currentInventory, listType)
        inventoryData.remove(product)
        preferences.setProducts(gson.toJson(inventoryData))
    }
}

override fun isBuyProduct(cost: Int): Boolean {
    return if (preferences.getCoinsBalance() >= cost) {
        preferences.removeCoinsFromBalance(cost)
        true
    } else
        false
    }
}

```

TargetsViewModel.kt

```

package com.tamigo.utils.viewModel

import androidx.lifecycle.LiveData
import androidx.lifecycle.MutableLiveData
import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import com.tamigo.utils.data.Target
import com.tamigo.utils.managers.HealthConnectManager
import com.tamigo.utils.preferences.Preferences
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.launch
import java.time.ZonedDateTime

abstract class TargetsViewModel : ViewModel() {
    abstract val currentSteps: LiveData<Long>
    abstract fun getSteps()
    abstract fun setTarget(target: Target)
    abstract fun getTarget(): Target?
    abstract fun setCoinsToBalance(coins: Int?)
}

class TargetsViewModelImpl(
    private val healthConnectManager: HealthConnectManager,
    private val preferences: Preferences

```

```

): TargetsViewModel() {
    override val currentSteps = MutableLiveData<Long>()

    override fun getSteps() {
        viewModelScope.launch(Dispatchers.IO) {
            currentSteps.postValue(healthConnectManager.getSteps())
        }
    }

    override fun setTarget(target: Target) {
        preferences.setTarget(target)
        preferences.setStartTargetTime(ZonedDateTime.now())
        getSteps()
    }

    override fun getTarget(): Target? {
        return preferences.getTarget()
    }

    override fun setCoinsToBalance(coins: Int?) {
        if (coins != null)
            preferences.setCoinsToBalance(coins)
    }
}

```

activity_main.xml

```

<?xml version="1.0" encoding="utf-8"?>
<androidx.drawerlayout.widget.DrawerLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:id="@+id/drawer_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <FrameLayout
        android:id="@+id/fragment_container"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

    <com.google.android.material.navigation.NavigationView
        android:id="@+id/navigation"

```

```

        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:layout_gravity="start"
        app:menu="@menu/drawer_menu" />

```

```
</androidx.drawerlayout.widget.DrawerLayout>
```

dialog_confirmation.xml

```

<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:padding="10dp">

```

```

<TextView
    android:id="@+id/title"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_margin="8dp"
    android:text="@string/are_you_agree"
    android:textColor="@color/oxford_blue"
    android:textSize="20sp"
    android:textStyle="bold"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

```

```

<TextView
    android:id="@+id/description"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_margin="8dp"
    android:gravity="center"
    android:text="@string/warning"
    android:textColor="@color/oxford_blue"
    android:textSize="16sp"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/title" />

```

```

<Button
    android:id="@+id/btn_agree"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_margin="8dp"
    android:text="@string/agree"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toStartOf="@+id/btn_cancel"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/description" />

```

```

<Button
    android:id="@+id/btn_cancel"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_margin="8dp"
    android:text="@string/cancel"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toEndOf="@id/btn_agree"
    app:layout_constraintTop_toBottomOf="@+id/description" />
</androidx.constraintlayout.widget.ConstraintLayout>

```

dialog_purchase.xml

```

<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:padding="10dp">

    <TextView
        android:id="@+id/title"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_margin="8dp"
        android:gravity="center"
        android:textColor="@color/oxford_blue"

```

```

        android:textSize="20sp"
        android:textStyle="bold"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        tools:text="@string/are_you_agree" />

```

```
<Button
```

```

        android:id="@+id/btn_agree"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_margin="8dp"
        android:text="@string/agree"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toStartOf="@+id/btn_cancel"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/title" />

```

```
<Button
```

```

        android:id="@+id/btn_cancel"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_margin="8dp"
        android:text="@string/cancel"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toEndOf="@id/btn_agree"
        app:layout_constraintTop_toBottomOf="@+id/title" />

```

```
</androidx.constraintlayout.widget.ConstraintLayout>
```

fragment_home.xml

```
<?xml version="1.0" encoding="utf-8"?>
```

```

<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@drawable/ui_background"
    android:padding="10dp">

```

```
<TextView
```

```
    android:id="@+id/txt_name"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textColor="@color/oxford_blue"
    android:textSize="20sp"
    android:textStyle="bold"
    app:layout_constraintBottom_toTopOf="@id/view"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    tools:text="TamiGo" />
```

```
<ImageView
```

```
    android:id="@+id/image_coin"
    android:layout_width="30dp"
    android:layout_height="30dp"
    android:src="@drawable/ui_coin"
    app:layout_constraintBottom_toBottomOf="@+id/coin_balance"
    app:layout_constraintEnd_toStartOf="@+id/coin_balance"
    app:layout_constraintTop_toTopOf="@+id/coin_balance"
    tools:ignore="ContentDescription" />
```

```
<TextView
```

```
    android:id="@+id/coin_balance"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textColor="@color/oxford_blue"
    android:textSize="20sp"
    android:textStyle="bold"
    app:layout_constraintBottom_toBottomOf="@+id/txt_name"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    tools:text="9999" />
```

```
<View
```

```
    android:id="@+id/view"
    android:layout_width="0dp"
    android:layout_height="1dp"
    android:layout_marginVertical="6dp"
```

```

    android:background="@color/light_blue"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/txt_name" />

```

```

<ImageView
    android:id="@+id/skin"
    android:layout_width="350dp"
    android:layout_height="350dp"
    android:scaleType="centerCrop"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/view"
    tools:ignore="ContentDescription"
    tools:src="@drawable/tami_blue" />

```

```

<View
    android:id="@+id/view2"
    android:layout_width="0dp"
    android:layout_height="1dp"
    android:background="@color/light_blue"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/skin" />

```

```

<ImageView
    android:id="@+id/ic_health"
    android:layout_width="26dp"
    android:layout_height="26dp"
    android:src="@drawable/ui_heart"
    app:layout_constraintBottom_toBottomOf="@+id/progress_health"
    app:layout_constraintEnd_toStartOf="@+id/progress_health"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="@+id/progress_health"
    tools:ignore="ContentDescription,ImageContrastCheck" />

```

```

<ProgressBar
    android:id="@+id/progress_health"
    style="@style/Widget.AppCompat.ProgressBar.Horizontal"
    android:layout_width="0dp"
    android:layout_height="12dp"

```

```

    android:layout_margin="20dp"
    android:max="100"
    android:min="0"
    android:progressDrawable="@drawable/progress_health"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toEndOf="@id/ic_health"
    app:layout_constraintTop_toBottomOf="@+id/view2" />

```

```
<FrameLayout
```

```

    android:id="@+id/home_fragment_container"
    android:layout_width="0dp"
    android:layout_height="0dp"
    app:layout_constraintBottom_toTopOf="@+id/btn_shop"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.476"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/ic_health"
    app:layout_constraintVertical_bias="0.0" />

```

```
<Button
```

```

    android:id="@+id/btn_target"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:minWidth="150dp"
    android:minHeight="50dp"
    android:text="@string/targets"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toStartOf="@id/btn_shop"
    app:layout_constraintStart_toStartOf="parent"
    tools:ignore="TextContrastCheck" />

```

```
<Button
```

```

    android:id="@+id/btn_shop"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:minWidth="150dp"
    android:minHeight="50dp"
    android:text="@string/shop"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toEndOf="@+id/btn_target"

```

```
tools:ignore="TextContrastCheck" />
```

```
</androidx.constraintlayout.widget.ConstraintLayout>
```

fragment_info.xml

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
```

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/developer"
    android:textColor="@color/oxford_blue"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    android:gravity="center"
    android:textStyle="bold"
    android:textSize="20sp"/>
```

```
</androidx.constraintlayout.widget.ConstraintLayout>
```

fragment_registration.xml

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="20dp"
    android:background="@drawable/ui_background">
```

```
<EditText
    android:id="@+id/et_tami_name"
    android:layout_width="0dp"
```

```

        android:layout_height="wrap_content"
        android:textSize="15sp"
        android:hint="@string/enter_name"
        android:importantForAutofill="no"
        android:inputType="text"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

```

```

<androidx.recyclerview.widget.RecyclerView
    android:id="@+id/recycler_tami_list"
    android:layout_width="0dp"
    android:layout_height="0dp"
    app:layoutManager="androidx.recyclerview.widget.GridLayoutManager"
    app:layout_constraintBottom_toTopOf="@+id/button"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/et_tami_name" />

```

```

<Button
    android:id="@+id/button"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="@string/start"
    android:textSize="20sp"
    app:layout_constraintBottom_toBottomOf="parent"
    tools:layout_editor_absoluteX="20dp" />

```

```

</androidx.constraintlayout.widget.ConstraintLayout>

```

fragment_shop.xml

```

<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@drawable/ui_background_border"
    android:padding="16dp">

```

```

<TextView

```

```

    android:id="@+id/txt_inventory"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/inventory"
    android:textColor="@color/oxford_blue"
    android:textSize="18sp"
    android:textStyle="bold"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

```

```

<androidx.recyclerview.widget.RecyclerView
    android:id="@+id/inventory"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_marginVertical="6dp"
    android:orientation="horizontal"
    app:layoutManager="androidx.recyclerview.widget.LinearLayoutManager"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/txt_inventory" />

```

```

<TextView
    android:id="@+id/txt_shop"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginVertical="6dp"
    android:text="@string/shop"
    android:textColor="@color/oxford_blue"
    android:textSize="18sp"
    android:textStyle="bold"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/inventory" />

```

```

<androidx.recyclerview.widget.RecyclerView
    android:id="@+id/shop"
    android:layout_width="0dp"
    android:layout_height="0dp"
    android:layout_marginVertical="6dp"
    android:orientation="horizontal"
    app:layoutManager="androidx.recyclerview.widget.GridLayoutManager"
    app:layout_constraintBottom_toBottomOf="parent"

```

```

app:layout_constraintEnd_toEndOf="parent"
app:layout_constraintStart_toStartOf="parent"
app:layout_constraintTop_toBottomOf="@+id/txt_shop" />

```

```
</androidx.constraintlayout.widget.ConstraintLayout>
```

fragment_targets.xml

```

<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@drawable/ui_background_border_pink"
    android:padding="16dp">

```

```

<TextView
    android:id="@+id/txt_description"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/your_current_target"
    android:textColor="@color/oxford_blue"
    android:textSize="18sp"
    android:textStyle="bold"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

```

```

<TextView
    android:id="@+id/txt_current_steps"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginVertical="6dp"
    android:textColor="@color/oxford_blue"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@id/txt_description"
    tools:text="5000" />

```

```

<TextView
    android:id="@+id/txt_slash"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"

```

```

android:layout_marginHorizontal="6dp"
android:text="@string/slash"
android:textColor="@color/oxford_blue"
app:layout_constraintBottom_toBottomOf="@+id/txt_current_steps"
app:layout_constraintStart_toEndOf="@id/txt_current_steps"
app:layout_constraintTop_toTopOf="@id/txt_current_steps" />

```

<TextView

```

android:id="@+id/txt_target_steps"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_marginHorizontal="6dp"
android:textColor="@color/oxford_blue"
app:layout_constraintBottom_toBottomOf="@+id/txt_slash"
app:layout_constraintStart_toEndOf="@id/txt_slash"
app:layout_constraintTop_toTopOf="@id/txt_slash"
tools:text="10000" />

```

<Button

```

android:id="@+id/btn_getPrize"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="@string/get"
app:layout_constraintEnd_toEndOf="parent"
app:layout_constraintTop_toTopOf="parent" />

```

<ProgressBar

```

android:id="@+id/progress_target"
style="@style/Widget.AppCompat.ProgressBar.Horizontal"
android:layout_width="0dp"
android:layout_height="10dp"
android:layout_marginVertical="10dp"
android:max="100"
android:min="0"
android:progressDrawable="@drawable/progress_target"
app:layout_constraintEnd_toEndOf="parent"
app:layout_constraintStart_toStartOf="parent"
app:layout_constraintTop_toBottomOf="@+id/txt_current_steps" />

```

<androidx.recyclerview.widget.RecyclerView

```

android:id="@+id/targets_list"

```

```

    android:layout_width="0dp"
    android:layout_height="0dp"
    android:layout_marginTop="8dp"
    app:layoutManager="androidx.recyclerview.widget.LinearLayoutManager"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@id/progress_target"
    app:layout_constraintVertical_bias="0.0"
    tools:itemCount="8"
    tools:listitem="@layout/item_targets" />

```

```
</androidx.constraintlayout.widget.ConstraintLayout>
```

item_food.xml

```

<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">

    <ImageView
        android:id="@+id/icon_food"
        android:layout_width="100dp"
        android:layout_height="100dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.0"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.0"
        tools:ignore="ContentDescription"
        tools:src="@drawable/food_burger" />

</androidx.constraintlayout.widget.ConstraintLayout>

```

item_shop.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">

    <ImageView
        android:id="@+id/icon_food"
        android:layout_width="120dp"
        android:layout_height="120dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.0"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.0"
        tools:ignore="ContentDescription"
        tools:src="@drawable/food_burger" />

    <ImageView
        android:id="@+id/ic_coin"
        android:layout_width="30dp"
        android:layout_height="30dp"
        android:src="@drawable/ui_coin"
        app:layout_constraintEnd_toStartOf="@+id/cost"
        app:layout_constraintTop_toBottomOf="@id/icon_food"
        tools:ignore="ContentDescription" />

    <TextView
        android:id="@+id/cost"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textColor="@color/oxford_blue"
        android:textSize="20sp"
        android:textStyle="bold"
        app:layout_constraintBottom_toBottomOf="@+id/ic_coin"
        app:layout_constraintEnd_toEndOf="@+id/icon_food"
        app:layout_constraintStart_toStartOf="@+id/icon_food"
```

```

    app:layout_constraintTop_toTopOf="@+id/ic_coin"
    tools:text="200" />

```

```

</androidx.constraintlayout.widget.ConstraintLayout>

```

item_tami_list.xml

```

<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">

    <ImageView
        android:id="@+id/image"
        android:layout_width="200dp"
        android:layout_height="200dp"
        android:scaleType="centerInside"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        tools:ignore="ContentDescription"
        tools:src="@drawable/tami_lime" />

</androidx.constraintlayout.widget.ConstraintLayout>

```

item_targets.xml

```

<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:padding="4dp">

    <TextView
        android:id="@+id/txt_number_steps"
        android:layout_width="wrap_content"

```

```

android:layout_height="wrap_content"
android:text="@string/number_of_steps"
android:textColor="@color/oxford_blue"
android:textSize="14sp"
app:layout_constraintStart_toStartOf="parent"
app:layout_constraintTop_toTopOf="parent" />

```

```
<TextView
```

```

    android:id="@+id/txt_prize"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginVertical="10dp"
    android:text="@string/prize"
    android:textColor="@color/oxford_blue"
    android:textSize="14sp"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/txt_number_steps" />

```

```
<TextView
```

```

    android:id="@+id/number_steps"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginHorizontal="8dp"
    android:textColor="@color/oxford_blue"
    android:textSize="14sp"
    app:layout_constraintStart_toEndOf="@+id/txt_number_steps"
    app:layout_constraintTop_toTopOf="parent"
    tools:text="10000" />

```

```
<TextView
```

```

    android:id="@+id/prize_coins"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginHorizontal="4dp"
    android:textColor="@color/oxford_blue"
    android:textSize="14sp"
    android:textStyle="bold"
    app:layout_constraintBottom_toBottomOf="@+id/txt_prize"
    app:layout_constraintStart_toEndOf="@+id/txt_prize"
    app:layout_constraintTop_toTopOf="@+id/txt_prize"
    tools:text="10 000" />

```

```

<ImageView
    android:id="@+id/ic_coin"
    android:layout_width="20dp"
    android:layout_height="20dp"
    android:src="@drawable/ui_coin"
    app:layout_constraintBottom_toBottomOf="@+id/txt_prize"
    app:layout_constraintStart_toEndOf="@+id/prize_coins"
    app:layout_constraintTop_toTopOf="@+id/txt_prize"
    tools:ignore="ContentDescription" />

```

```

<Button
    android:id="@+id/btn_choose"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/choose"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    tools:ignore="TextContrastCheck" />

```

```

</androidx.constraintlayout.widget.ConstraintLayout>

```

AndroidManifest.xml

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">

    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.POST_NOTIFICATIONS" />
    <uses-permission android:name="android.permission.SCHEDULE_EXACT_ALARM" />
    <uses-permission android:name="android.permission.health.READ_STEPS" />
    <uses-permission android:name="com.android.alarm.permission.SET_ALARM" />

    <queries>
        <package android:name="com.google.android.apps.healthdata" />
    </queries>

    <application
        android:name=".main.TamiGoApplication"

```

```

        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@drawable/app_logo"
        android:label="@string/app_name"
        android:roundIcon="@drawable/app_round_logo"
        android:supportsRtl="true"
        android:theme="@style/Theme.TamiGo"
        tools:targetApi="31">
        <activity
            android:name=".main.MainActivity"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <action android:name="androidx.health.ACTION_SHOW_PERMISSIONS_RATIONALE" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>

            <meta-data
                android:name="android.app.lib_name"
                android:value="" />
            </activity>
            <receiver
                android:name=".utils.receivers.HealthReceiver"
                android:exported="false">
                <intent-filter>
                    <action android:name="MINUS_HEALTH_INTENT" />
                    <action android:name="PLUS_HEALTH_INTENT" />
                </intent-filter>
            </receiver>
        </application>

</manifest>

```

build.gradle (Project: TamiGo)

```

plugins {
    id 'com.android.application' version '7.3.1' apply false
    id 'com.android.library' version '7.3.1' apply false
    id 'org.jetbrains.kotlin.android' version '1.7.20' apply false

```

```

}

build.gradle (Module: TamiGo.app)

plugins {
    id 'com.android.application'
    id 'org.jetbrains.kotlin.android'
}

android {
    namespace 'com.tamigo'
    compileSdk 33

    defaultConfig {
        applicationId "com.tamigo"
        minSdk 26
        targetSdk 33
        versionCode 1
        versionName "1.0"

        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
    }

    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'
        }
    }
    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
    }
    kotlinOptions {
        jvmTarget = '1.8'
    }
    buildFeatures {
        viewBinding = true
    }
    apply plugin: 'kotlin-kapt'
}

```

```
dependencies {
```

```
    implementation 'androidx.core:core-ktx:1.7.0'
    implementation 'androidx.appcompat:appcompat:1.6.1'
    implementation 'com.google.android.material:material:1.9.0'
    implementation 'androidx.constraintlayout:constraintlayout:2.1.4'
    testImplementation 'junit:junit:4.13.2'
    androidTestImplementation 'androidx.test.ext:junit:1.1.5'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.5.1'
```

```
    implementation "androidx.core:core-splashscreen:1.0.1"
```

```
    implementation "io.insert-koin:koin-android:3.3.3"
    implementation "io.insert-koin:koin-androidx-workmanager:3.3.3"
```

```
    implementation 'com.github.bumptech.glide:glide:4.12.0'
    kapt 'com.github.bumptech.glide:compiler:4.12.0'
```

```
    implementation "androidx.health.connect:connect-client:1.0.0-alpha11"
    implementation(platform("org.jetbrains.kotlin:kotlin-bom:1.8.0"))
```

```
    implementation 'com.google.code.gson:gson:2.10.1'
```

```
}
```

```
drawer_menu.xml
```

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">
    <group android:checkableBehavior="single">
        <item
            android:id="@+id/home"
            android:title="@string/home"
            app:showAsAction="withText|ifRoom" />

        <item
            android:id="@+id/info"
            android:title="@string/info"
            app:showAsAction="withText|ifRoom" />
```

```
<item
    android:id="@+id/restart"
    android:title="@string/restart"
    app:showAsAction="withText|ifRoom" />
</group>
</menu>
```

ДОДАТОК В
Слайди презентації

Національний університет «Запорізька політехніка»

Кафедра програмних засобів

Дипломна кваліфікаційна робота

Розробка мобільного застосунку шагоміру

Виконала
ст. гр. КНТ-139

Орина РУДЬ

Керівник
доцент

Тетяна ФЕДОРОНЧАК

Рисунок В.1 — Слайд 1

Огляд роботи

Об'єкт дослідження – мобільні додатки для відслідковування активності.

Предмет дослідження - мобільні додатки для відслідковування кількості кроків.

Мета роботи – створення мобільного додатку для підвищення середньодобової активності та заохочення ходити більше.

Рисунок В.2 — Слайд 2

Задачі роботи

- виконати аналіз предметної області відносно процесу відстеження активності та заохочення користувачів у її підвищенні;
- виконати план проектування застосунку та визначитися з мовою, шаблонами проектування та бібліотеками;
- розробити мобільний застосунок шагомір з ігровою механікою для підвищення зацікавленості користувача;
- виконати тестування розробленого мобільного застосунку шагоміру.

Рисунок В.3 — Слайд 3

Вибір мови програмування

Характеристика	Kotlin	Java
Синтаксис	біль чистий та лаконічний	багатослівний та громіздкий
Розширення функцій	дозволяє додавати нові функції до існуючих класів	не дозволяє додавати нові функції до існуючих класів
Швидкість розробки	швидка	низька
Обробка винятків	винятки є перевіреними	винятки є перевіреними та не перевіреними
Використання для Android розробки	є рекомендованою та офіційно підтримується Google	довгий час був основною мовою
Популярність	швидко зростає	висока

Рисунок В.4 — Слайд 4

Вибір архітектури

Характеристика	MVP	MVVM
Зв'язок між компонентами	пов'язані між собою	пов'язані через централізовану ViewModel
Популярність	широко використовується	стає все більш популярним
Область застосування	прості програми з мінімальною логікою	великі проекти з багатьма залежностями
Зрозумілість	простий для розуміння	може бути складним для новачків

Рисунок В.5 — Слайд 5

Вибір методу підрахунку кроків

Самостійно реалізований крокомір через акселерометр

Позитивні сторони:

- використання додатку без підключення до мережі Інтернет;
- вбудований акселерометр у сучасних мобільних пристроях має високу точність вимірювання руху, що дозволяє отримувати точні результати.

Негативні сторони:

- реалізація обмежена лише функціями підрахунку кроків та розрахунку дистанції;
- споживає додатковий заряд акумулятора, що може бути небажаним для користувачів;
- для досягнення максимальної точності при використанні акселерометра необхідно дотримуватися певних умов, таких як правильне розташування пристрою на тілі користувача і відсутність інших об'єктів, що рухаються поруч.

Рисунок В.6 — Слайд 6

Вибір методу підрахунку кроків

HealthConnect API

Позитивні сторони:

- використовує безліч сенсорів для визначення кількості кроків, таких як акселерометри, гіроскопи та магнітометри;
- можна швидко та легко отримати доступ до даних про кроки, не витрачаючи час на розробку та тестування власного крокоміра;
- може надавати додаткові функції, такі як відстеження сну та серцевого ритму, які можуть бути корисними для різних цілей;
- простота використання та інтеграції;
- підтримка стандарту HL7 для обміну медичними даними.

Негативні сторони:

- доступний лише для певних пристроїв та операційних систем;
- вимагає доступу до Інтернету для роботи;
- залежність від стороннього сервісу;
- обмежений контроль над обробкою даних.

Рисунок В.7 — Слайд 7

Вибір ін'єкції залежностей

Характеристика	Hilt	Koin
Використання	розроблений як офіційне рішення для залежностей у Android	легковаговий фреймворк для впровадження залежностей у Kotlin
Підтримка	розробляється та підтримується Google	відкритий і активно розвивається на GitHub
Компіляційна перевірка	перевірка залежностей під час компіляції	працює під час виконання, відсутня компіляційна перевірка
Відстеження залежностей	використовує анотації для вказівки залежностей	описує залежності у вигляді модулів та функцій
Конфігурування залежностей	відносно складне та розширюване конфігурування	просте та гнучке конфігурування
Декларативність	менш декларативний, вимагає більше налаштувань та конфігурації	більш декларативний, використовує DSL для опису залежностей

Рисунок В.8 — Слайд 8

Загальна структура застосунку

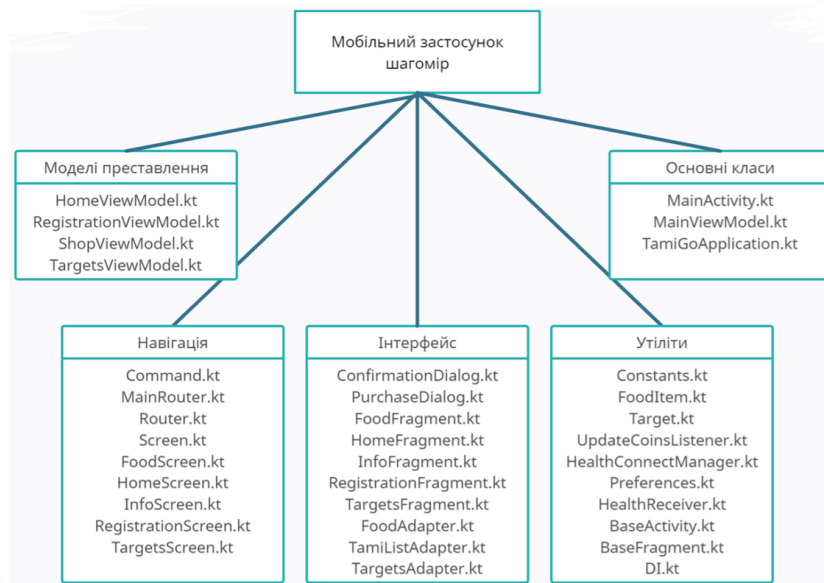


Рисунок В.9 — Слайд 9

Інтерфейс програми

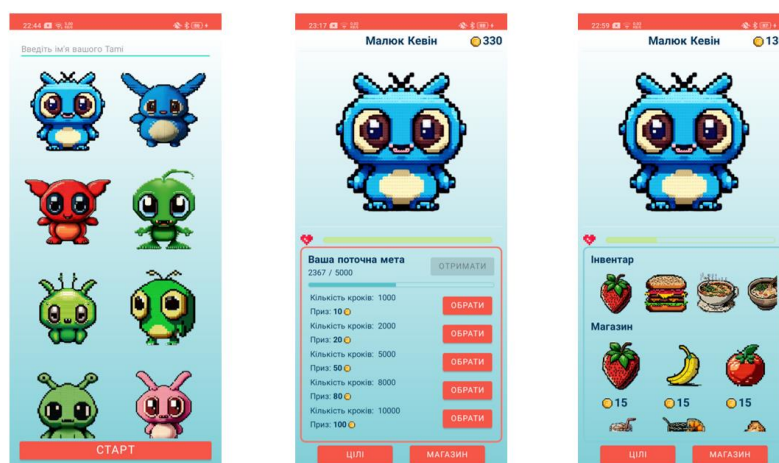


Рисунок В.10 — Слайд 10

Інтерфейс програми

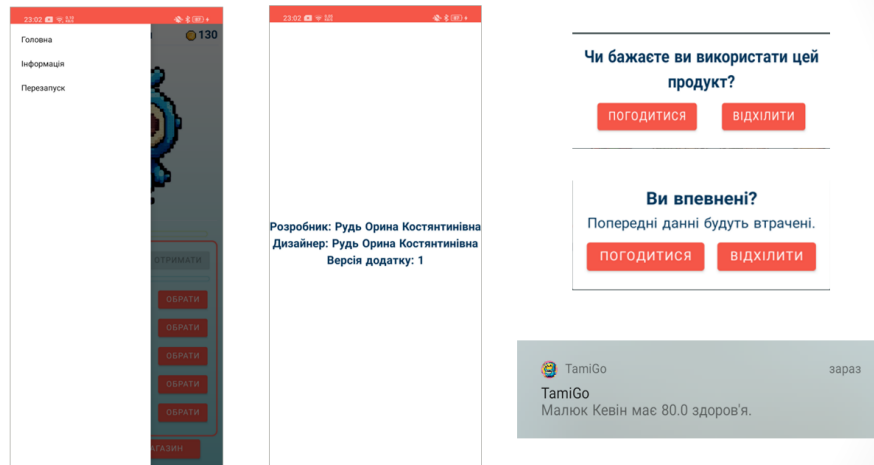


Рисунок В.11 — Слайд 11

Висновок

У процесі розробки програми була використана мова програмування Kotlin, яка має переваги в плані безпеки, виразності та простоти використання. MVVM був обраний як архітектурний шаблон для забезпечення поділу обов'язків та покращення підтримуваності коду. Для управління залежностями та впровадження залежностей у додаток був обраний фреймворк Koin, який надає простий та гнучкий підхід до ін'єкції залежностей. Це спростило процес налаштування та використання залежностей у проекті. Data Binding був використаний для зв'язування даних і уявлень у додатку, що сприяє поліпшенню продуктивності та спрощенню роботи з інтерфейсом користувача. Додаток розроблено з використанням сучасних технологій та інструментів, що забезпечує його ефективність, продуктивність та зручність використання.

Подальший розвиток програми може включати розширення функціональності, додавання нових ігрових елементів, покращення дизайну та оптимізацію продуктивності. Також можна розглянути можливість інтеграції з іншими сторонніми сервісами та платформами для подальшого поліпшення досвіду користувача.

Рисунок В.12 — Слайд 12