ПОЛТАВСЬКИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТОРГІВЛІ
Навчально-науковий інститут денної освіти
Форма навчання денна
Кафедра комп'ютерних наук та інформаційних технологій

Допускається до захисту
Завідувач кафедри
_____ Олена ОЛЬХОВСЬКА
(підпис)

«___» _____ 2025 р.

КВАЛІФІКАЦІЙНА РОБОТА

на тему

АЛГОРИТМІЗАЦІЯ ТА РОЗРОБКА ЗАСТОСУНКУ ПЕРСОНАЛЬНИХ РЕКОМЕНДАЦІЙ ДЛЯ ОПТИМІЗАЦІЇ ФІЗИЧНОЇ АКТИВНОСТІ НА ОСНОВІ МОНІТОРИНГУ КРОКІВ ТА ВИТРАЧЕНИХ КАЛОРІЙ

зі спеціальності 122 Комп'ютерні науки
освітня програма «Комп'ютерні науки»
ступеня магістра

Виконавець роботи Шеліхов Артем Сергійович
_____ «____»_____ 2025 р.
(підпис)

Науковий керівник к.ф.-м.н., доцент, Парфьонова Тетяна Олександрівна
_____ «____»_____ 2025 р.
(підпис)

Рецензент

ПОЛТАВА 2025 р.

ЗАТВЕРДЖУЮ
Завідувач кафедри _____Олена ОЛЬХОВСЬКА
(підпис)
«___» _____ 2024 р.

ЗАВДАННЯ І КАЛЕНДАРНИЙ ГРАФІК ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ

на тему «Алгоритмізація та розробка застосунку персональних рекомендацій для оптимізації фізичної активності на основі моніторингу кроків та витрачених калорій**»**
зі спеціальності 122 Комп'ютерні науки
освітня програма «Комп'ютерні науки»
ступеня магістр
Прізвище, ім'я, по батькові Шеліхов Артем Сергійович
Затверджена наказом ректора № ____ _____ 2024 р..
Термін подання студентом роботи «___»_____ 2025 р.
Вихідні дані до кваліфікаційної роботи: статті та документації з теми розробки андроїд застосунків.
Зміст пояснювальної записки (перелік питань, які потрібно розробити)
ВСТУП
РОЗДІЛ 1. ПОСТАНОВКА ЗАДАЧІ
РОЗДІЛ 2. ОГЛЯД СИСТЕМ АНАЛОГІЧНОГО ПРИЗНАЧЕННЯ
2.1. Огляд сучасних мобільних застосунків для моніторингу фізичної активності
2.2. Порівняльний аналіз функціональних можливостей систем підрахунку кроків і калорій
2.3. Аналіз алгоритмів збору та обробки даних фізичної активності в існуючих рішеннях
2.4. Аналіз інтерфейсів користувача та мотиваційних механізмів у фітнес-додатках
2.5. Визначення недоліків існуючих рішень та обґрунтування необхідності розробки власного застосунку
РОЗДІЛ 3. ТЕОРЕТИЧНА ЧАСТИНА
3.1. Теоретичні основи моніторингу фізичної активності за допомогою мобільних сенсорів
3.2. Математичні моделі розрахунку показників фізичної активності
3.3. Алгоритмізація формування персональних рекомендацій щодо фізичної активності
3.4. Проєктування архітектури мобільного застосунку на основі Clean Architecture та MVVM та побудова UML-діаграм
РОЗДІЛ 4. ПРАКТИЧНА ЧАСТИНА
4.1. Загальна структура програмного застосунку та опис використаних технологій розробки
4.2. Реалізація підсистеми збору даних фізичної активності
4.3. Реалізація модуля обробки даних та генерації персональних рекомендацій
4.4. Розробка користувацького інтерфейсу мобільного застосунку засобами Jetpack Compose
4.5. Збереження та обробка даних користувача
4.6. Інструкція користувача щодо роботи з програмним продуктом
ВИСНОВКИ
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ
ДОДАТОК А
Перелік графічного матеріалу: 2 аркуші блок схем, 13 ілюстрацій.

Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали, посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Постановка задачі	Парфьонова Т.О.		
Інформаційний огляд	Парфьонова Т.О.		
Теоретична частина	Парфьонова Т.О.		
Практична частина	Парфьонова Т.О.		

Календарний графік виконання кваліфікаційної роботи

Зміст роботи	Термін виконання	Фактичне виконання
Вступ		
Вивчення методичних рекомендацій та стандартів та звіт керівнику		
Постановка задачі		
Інформаційний огляд джерел бібліотек та інтернету		
Теоретична частина		
Практична частина		
Закінчення оформлення		
Доповідь студента на кафедрі		
Доробка (за необхідністю), рецензування		

Дата видачі завдання «__» _____ 2024 р.

Здобувач вищої освіти _____ Шеліхов Артем Сергійович
(підпис)

Науковий керівник _____ к.ф.-м.н., доц. Парфьонова Т.О.
(підпис) (науковий ступінь, вчене звання, ініціали та прізвище)

Результати захисту кваліфікаційної роботи

Кваліфікаційна робота оцінена на _____
(балів, оцінка за національною шкалою, оцінка за ECTS)

Протокол засідання ЕК № ____ від «____» _____ 2025 р.

Секретар ЕК_____ _____
(підпис) (ініціали та прізвище)

Затверджую
Зав. кафедрою _____
к.ф.-м.н. Олена ОЛЬХОВСЬКА
«____» _____ 2024 р.

Погоджено
Науковий керівник _____
к.пед.н., Тетяна ПАРФЬОНОВА
«____» _____ 2024 р.

План

дипломного проекту з фаху
спеціальності 122 Комп'ютерні науки
освітня програма 122 Комп'ютерні науки
на тему «Алгоритмізація та розробка застосунку персональних рекомендацій для оптимізації фізичної активності на основі моніторингу кроків та витрачених калорій»
Прізвище, ім'я, по батькові Шеліхов Артем Сергійович

ВСТУП
РОЗДІЛ 1. ПОСТАНОВКА ЗАДАЧІ
РОЗДІЛ 2. ОГЛЯД СИСТЕМ АНАЛОГІЧНОГО ПРИЗНАЧЕННЯ
2.1. Огляд сучасних мобільних застосунків для моніторингу фізичної активності
2.2. Порівняльний аналіз функціональних можливостей систем підрахунку кроків і калорій
2.3. Аналіз алгоритмів збору та обробки даних фізичної активності в існуючих рішеннях
2.4. Аналіз інтерфейсів користувача та мотиваційних механізмів у фітнес-додатках
2.5. Визначення недоліків існуючих рішень та обґрунтування необхідності розробки власного застосунку
РОЗДІЛ 3. ТЕОРЕТИЧНА ЧАСТИНА
3.1. Теоретичні основи моніторингу фізичної активності за допомогою мобільних сенсорів
3.2. Математичні моделі розрахунку показників фізичної активності
3.3. Алгоритмізація формування персональних рекомендацій щодо фізичної активності
3.4. Проєктування архітектури мобільного застосунку на основі Clean Architecture та MVVM та побудова UML-діаграм
РОЗДІЛ 4. ПРАКТИЧНА ЧАСТИНА
4.1. Загальна структура програмного застосунку та опис модулів
4.2. Реалізація підсистеми збору даних фізичної активності
4.3. Реалізація модуля обробки даних та генерації персональних рекомендацій
4.4. Розробка користувацького інтерфейсу мобільного застосунку засобами Jetpack Compose
4.5. Збереження та обробка даних користувача
4.6. Інструкція користувача щодо роботи з програмним продуктом
ВИСНОВКИ
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ
ДОДАТОК А

Здобувач вищої освіти _____ А.С. Шеліхов
(підпис)
«____» _____ 2024 р.

РЕФЕРАТ

Записка: 128 сторінок, 13 рисунків, 1 додаток, 19 літературних джерел.

Метою даного дослідження є алгоритмізація та розробка мобільного застосунку персональних рекомендацій для оптимізації фізичної активності на основі моніторингу кількості кроків та витрачених калорій з використанням сучасних засобів Android-розробки.

Об'єктом дослідження є процес моніторингу та оптимізації фізичної активності користувача з використанням мобільних інформаційних технологій.

Предметом дослідження є алгоритми збору, обробки та аналізу даних фізичної активності, а також програмні засоби реалізації персоналізованих рекомендацій у мобільному застосунку.

У процесі розробки застосовувалися такі методи: об'єктно-орієнтоване та модульне програмування, архітектурне проєктування з використанням принципів Clean Architecture, реактивна обробка даних, алгоритмічне моделювання та аналіз сенсорних вимірювань. Методологічну основу роботи становив підхід розділення відповідальностей між рівнями системи, що забезпечує незалежність функціональних компонентів, зручність супроводу та можливість подальшого розширення програмного продукту.

Інструментарій реалізації включав мову програмування Kotlin, платформу Android та сучасні бібліотеки і фреймворки розробки мобільних застосунків. Для побудови архітектури використано принципи Clean Architecture та шаблон MVVM. Реалізація користувацького інтерфейсу здійснювалася засобами Jetpack Compose. Для збору даних фізичної активності застосовано Sensor API операційної системи Android, зокрема апаратний сенсор типу Sensor.TYPE_STEP_COUNTER. Збереження даних реалізовано з використанням локальної бази даних Room і механізмів реактивних потоків Flow.

У межах роботи проаналізовано сучасні мобільні застосунки для моніторингу фізичної активності та визначено їх основні функціональні особливості й обмеження; досліджено алгоритми збору та обробки сенсорних даних; обґрунтовано вибір математичних моделей розрахунку витрачених калорій, дистанції, індексу маси тіла та водного балансу; спроєктовано архітектуру мобільного застосунку; реалізовано підсистему безперервного збору даних у фоновому режимі; розроблено модуль обробки даних і генерації персональних рекомендацій; побудовано UML-діаграми, що відображають структуру та логіку роботи системи; створено зручний і наочний користувацький інтерфейс.

Розроблений програмний продукт є автономним мобільним застосунком, який не потребує постійного підключення до мережі Інтернет, зберігає дані локально на пристрої користувача та забезпечує стабільну роботу в повсякденному режимі. Застосунок орієнтований на широке коло користувачів і може використовуватися як інструмент підтримки здорового способу життя.

Практична цінність роботи полягає у створенні програмного засобу, що поєднує автоматизований збір сенсорних даних, алгоритмічно обґрунтовану обробку фізичної активності та персоналізовані рекомендації. Розроблений застосунок може бути використаний як самостійний фітнес-інструмент, а також як основа для подальшого розвитку систем моніторингу та аналізу фізичної активності з розширеними аналітичними можливостями.

ЗМІСТ

ВСТУП

Актуальність. Активний розвиток цифрових технологій та широке впровадження мобільних пристроїв у повсякденне життя людини зумовлюють зростання ролі програмних засобів, спрямованих на підтримку та оптимізацію фізичної активності. Смартфони, оснащені вбудованими сенсорами руху, надають можливість автоматизовано здійснювати збір і аналіз даних про рухову активність користувачів без потреби у додатковому обладнанні. Особливого значення набувають мобільні застосунки, які поєднують функції моніторингу кількості кроків і витрачених калорій з алгоритмічним формуванням персональних рекомендацій, що враховують індивідуальні фізіологічні характеристики користувача, зокрема вік, масу тіла, зріст і рівень фізичної активності.

Використання алгоритмічних методів обробки даних у поєднанні з сучасними підходами до програмної інженерії дозволяє створювати інтелектуальні мобільні системи, що сприяють підвищенню мотивації користувачів до регулярної фізичної активності та самоконтролю стану здоров’я. Разом із тим аналіз наявних фітнес-застосунків показує, що значна частина з них орієнтована переважно на накопичення статистичних даних і не забезпечує достатнього рівня персоналізації рекомендацій або прозорості алгоритмів розрахунку показників фізичної активності. Це зумовлює необхідність розробки програмного застосунку, в основі якого лежить чітко формалізована алгоритмізація процесів моніторингу, аналізу та оптимізації фізичної активності користувача.

Метою даної дипломної роботи є алгоритмізація та розробка мобільного застосунку персональних рекомендацій для оптимізації фізичної активності на основі моніторингу кроків та витрачених калорій з використанням сучасних засобів Android-розробки. У межах поставленої мети передбачається створення програмного продукту, який поєднує автоматичний збір даних із сенсорів

мобільного пристрою, математичну обробку показників фізичної активності та формування рекомендацій, адаптованих до індивідуальних параметрів користувача.

Об’єктом дослідження є процес моніторингу та оптимізації фізичної активності людини із застосуванням мобільних інформаційних технологій. **Предметом** дослідження є алгоритми збору, обробки та аналізу даних фізичної активності, а також програмні засоби реалізації персоналізованих рекомендацій у мобільному застосунку.

У процесі виконання роботи передбачається розв’язання таких науково-практичних завдань:

– здійснити аналіз існуючих мобільних фітнес-застосунків та визначити їх функціональні можливості і недоліки;

– дослідити математичні моделі розрахунку показників фізичної активності, зокрема кількості кроків, витрачених калорій, пройденої дистанції, індексу маси тіла та водного балансу;

– розробити алгоритми формування персональних рекомендацій з оптимізації фізичної активності користувача;

– спроєктувати архітектуру програмної системи з урахуванням принципів модульності та масштабованості;

– реалізувати основні функціональні модулі мобільного застосунку та забезпечити їх коректну взаємодію.

Розроблення програмного продукту ґрунтується на використанні мови програмування Kotlin та платформи Android із застосуванням архітектурних підходів MVVM і Clean Architecture, що забезпечують розділення відповідальностей між рівнями представлення, бізнес-логіки та доступу до даних. Для реалізації користувацького інтерфейсу використовується декларативний підхід Jetpack Compose, а для збереження та обробки даних застосовуються Room Database та Jetpack DataStore. Отримання інформації про

кількість кроків здійснюється за допомогою Android Sensor API, що дозволяє проводити моніторинг фізичної активності в реальному часі.

Пояснювальна записка складається з чотирьох основних розділів, у яких послідовно розглядаються постановка задачі, огляд систем аналогічного призначення, теоретичні основи алгоритмізації та архітектурного проєктування, а також практична реалізація й тестування мобільного застосунку. Така структура роботи забезпечує логічне та системне розкриття теми дослідження й демонструє практичну цінність розробленого програмного продукту для підтримки та оптимізації фізичної активності користувачів.

РОЗДІЛ 1. ПОСТАНОВКА ЗАДАЧІ

Розвиток мобільних інформаційних технологій та зростання інтересу суспільства до підтримки здорового способу життя зумовлюють актуальність створення програмних засобів, орієнтованих на автоматизований моніторинг фізичної активності людини. Одним із найбільш доступних і водночас інформативних показників щоденної активності є кількість виконаних кроків, яка безпосередньо пов’язана з рівнем енергетичних витрат, загальною рухливістю та профілактикою гіподинамії. Сучасні мобільні пристрої, зокрема смартфони на базі операційної системи Android, оснащені вбудованими сенсорами руху, що створює технічні передумови для безперервного збору даних про фізичну активність користувача в реальному часі.

Разом із тим сам по собі збір даних про кількість кроків не забезпечує досягнення основної мети - підвищення рівня фізичної активності та формування сталих корисних звичок. Для цього необхідно здійснювати подальшу алгоритмічну обробку отриманих даних, інтерпретувати їх з урахуванням індивідуальних параметрів користувача та формувати персоналізовані рекомендації. До таких параметрів належать вік, стать, маса тіла, зріст, рівень фізичної активності та встановлена денна ціль. Саме інтеграція сенсорних даних із персональними характеристиками користувача дозволяє перейти від простого обліку показників до інтелектуальної підтримки процесу оптимізації фізичної активності.

Постановка задачі даної дипломної роботи полягає у створенні мобільного застосунку, який забезпечує комплексний підхід до моніторингу фізичної активності на основі підрахунку кроків і витрачених калорій, а також реалізує алгоритми формування персональних рекомендацій. Програмний продукт має працювати автономно, не потребувати додаткових зовнішніх пристроїв і використовувати апаратні можливості смартфона. При цьому важливою вимогою є наочність подання інформації, збереження історії

активності та забезпечення зручної взаємодії користувача з інтерфейсом застосунку.

У межах поставленої задачі необхідно забезпечити коректний збір первинних даних про фізичну активність за допомогою апаратного крокоміра мобільного пристрою. Для цього використовується сенсор типу TYPE_STEP_COUNTER, який дозволяє отримувати накопичене значення кількості кроків з моменту останнього перезавантаження пристрою. Отримані значення повинні оброблятися програмними засобами таким чином, щоб визначати кількість кроків за поточний день незалежно від попередньої активності користувача. Це потребує реалізації механізмів фіксації початкового значення лічильника та коректної обробки подій сенсора у фоновому режимі.

Наступним важливим аспектом постановки задачі є математична обробка зібраних даних. На основі кількості кроків необхідно розраховувати похідні показники фізичної активності, зокрема витрачені калорії та пройдену дистанцію. Для цього застосовуються спрощені, але практично доцільні математичні моделі, що дозволяють отримати наближені значення без значного навантаження на обчислювальні ресурси мобільного пристрою. Окрім цього, використовуючи персональні дані користувача, необхідно реалізувати алгоритми розрахунку індексу маси тіла та рекомендованої норми споживання води, які є додатковими показниками стану фізичної активності та загального здоров’я.

Особливе місце в постановці задачі займає формування персональних рекомендацій. Застосунок повинен аналізувати поточний прогрес користувача щодо досягнення денної цілі кроків, порівнювати фактичні показники з рекомендованими значеннями та надавати зрозумілі текстові підказки мотиваційного характеру. При цьому рекомендації мають бути адаптивними, тобто змінюватися залежно від рівня активності користувача, часу доби та встановлених цілей. Реалізація такого підходу потребує чіткої алгоритмізації

процесу прийняття рішень і узгодження логіки рекомендацій із даними, що надходять із сенсорів і сховищ.

Для забезпечення збереження та аналізу даних у динаміці необхідно реалізувати механізми довготривалого зберігання інформації. У межах задачі передбачається збереження історії щоденної активності користувача у локальній базі даних, що дозволяє переглядати результати за попередні дні та здійснювати порівняльний аналіз. Окремо має бути забезпечене збереження персональних налаштувань користувача, які використовуються для розрахунків і формування рекомендацій. Такий підхід дозволяє реалізувати принцип єдиного джерела істини, за якого всі дані зберігаються централізовано, а інтерфейс користувача лише відображає актуальний стан системи.

З точки зору програмної інженерії постановка задачі передбачає побудову чіткої та масштабованої архітектури мобільного застосунку. Реалізація має ґрунтуватися на принципах розділення відповідальностей між шарами представлення, бізнес-логіки та доступу до даних. Такий підхід забезпечує підвищення надійності програмного продукту, спрощує його тестування та подальший розвиток. Особлива увага приділяється використанню реактивної моделі обробки даних, за якої зміна стану системи автоматично відображається в інтерфейсі користувача без необхідності ручного оновлення.

Інтерфейс мобільного застосунку повинен відповідати сучасним вимогам зручності та ергономіки, забезпечувати швидкий доступ до основної інформації та бути інтуїтивно зрозумілим для користувачів різного віку. У межах задачі необхідно реалізувати екран первинного налаштування профілю, головний екран з візуалізацією прогресу, екран перегляду історії активності та екран редагування персональних даних. Візуалізація результатів має відігравати ключову роль у мотивації користувача, тому особливу увагу слід приділити графічному поданню прогресу досягнення денної цілі.

Таким чином, постановка задачі дипломної роботи полягає у розробці мобільного Android-застосунку, який забезпечує автоматизований збір даних

про фізичну активність, їх математичну обробку, формування персоналізованих рекомендацій та зручне подання результатів користувачеві. Розв’язання поставленої задачі потребує поєднання алгоритмічних методів обробки даних, знань у галузі програмної інженерії та використання сучасних інструментів мобільної розробки, що в сукупності дозволяє створити ефективний програмний продукт для підтримки та оптимізації фізичної активності.

РОЗДІЛ 2. ОГЛЯД СИСТЕМ АНАЛОГІЧНОГО ПРИЗНАЧЕННЯ

2.1. Огляд сучасних мобільних застосунків для моніторингу фізичної активності

Сучасний сегмент мобільних застосунків для моніторингу фізичної активності сформувався на перетині декількох тенденцій: масового поширення смартфонів і носимих пристроїв, розвитку сенсорних технологій, популяризації превентивної медицини та переходу до даних як основи персоналізованих рекомендацій. У межах цього сегмента домінують застосунки, орієнтовані на облік щоденної активності, зокрема ходьби, бігу та інших видів рухової діяльності, а також на інтерпретацію отриманих показників у вигляді індикаторів прогресу, мотиваційних повідомлень і підказок щодо способу життя. Базовими функціями переважної більшості таких продуктів є автоматичний або напівавтоматичний підрахунок кроків, оцінювання витрачених калорій, визначення дистанції, облік часу активності та підтримка цілей, що задаються користувачем. Розширені можливості включають інтеграцію зі смартгодинниками та фітнес-браслетами, реєстрацію тренувань на основі GPS, моніторинг частоти серцевих скорочень, аналіз сну, а також ведення щоденників харчування й водного балансу.

Застосунки загального призначення для щоденної активності, як правило, прагнуть забезпечити універсальність та сумісність з різними пристроями, пропонуючи користувачеві єдине середовище для збирання та перегляду даних. Прикладом такого підходу є Google Fit, який позиціонується як платформа для відстеження активності з можливістю отримання статистики щодо ходьби, бігу та їзди на велосипеді на основі сенсорів смартфона або пристроїв Wear OS, а також надання підказок і елементів персоналізованого коучингу, що ґрунтується на історії активності користувача.

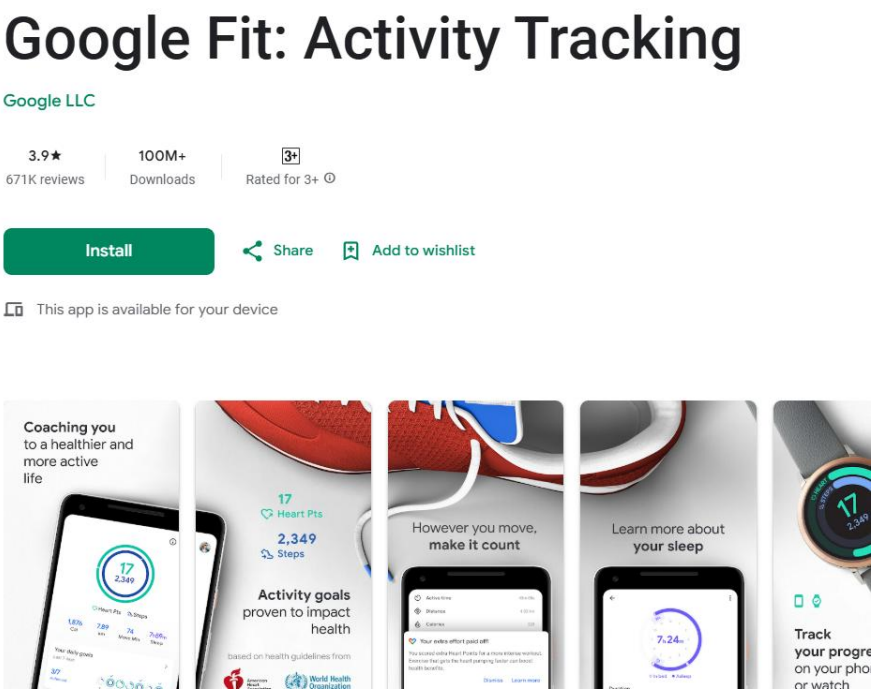

Рисунок 2.1. Інтерфейс Google Fit

Важливо, що Google Fit реалізує концепцію агрегатора, який може консолідувати дані з різних джерел і представити їх у уніфікованих метриках, чим полегшує користувачеві контроль за власною активністю впродовж дня. Для таких систем характерним є акцент на простоті взаємодії, мінімальному порозі входження, а також на мотиваційних інструментах у вигляді цілей, нагадувань і візуалізації прогресу, що підтримує формування корисних поведінкових патернів. У додаткових інформаційних матеріалах, присвячених Google Fit, підкреслюється використання гейміфікованих індикаторів активності та автоматичного розпізнавання певних типів рухової діяльності, що спрямовано на підвищення залученості користувача [10].

Окремий клас становлять екосистемні застосунки виробників пристроїв, які забезпечують тісну інтеграцію між смартфоном, носимими девайсами та хмарними сервісами. Показовим прикладом є Samsung Health, що позиціонується як застосунок для комплексного відстеження здоров’я й активності з фокусом на багатокомпонентному профілі користувача, де фізичні вправи поєднуються з даними про сон, харчування, показники серцево-

судинної системи й іншими параметрами. У контексті моніторингу фізичної активності цей застосунок, по-перше, підтримує автоматичний облік щоденних кроків, по-друге, дозволяє керувати добіркою показників, що відображаються на головному екрані, і, по-третє, забезпечує облік тренувань різних типів.

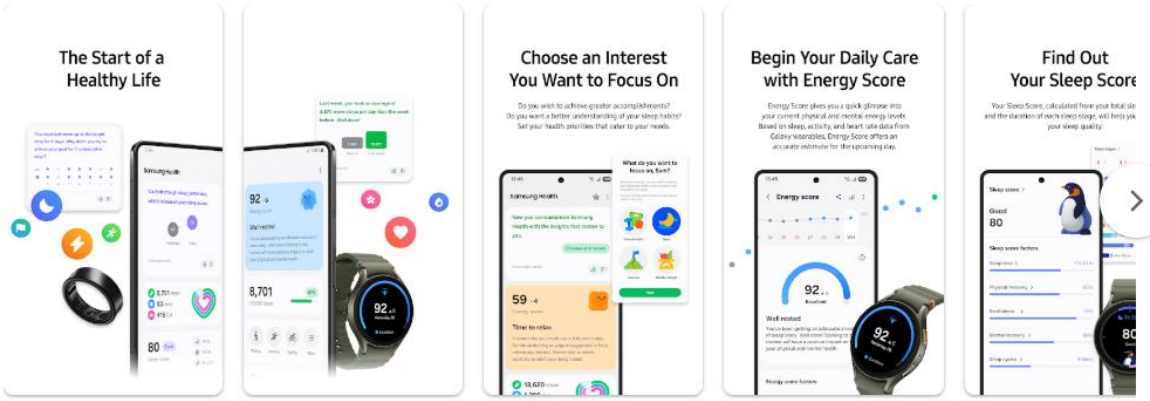

Рисунок 2.2. Інтерфейс застосунку Samsung Health

Важливо, що у матеріалах Samsung акцентується можливість отримання більш повної картини стану здоров’я за умови синхронізації зі смартгодинниками або іншими сумісними пристроями, оскільки частина показників, зокрема безперервні вимірювання певних фізіологічних параметрів, доступна саме завдяки носимим сенсорам [11]. Для користувача це означає зміщення фокусу від суто “крокоміра” до ширшої системи самоспостереження, яка формує уявлення про динаміку активності та її взаємозв’язок із відновленням, стресом і стилем життя.

Паралельно з універсальними платформами, широкого поширення набули застосунки, побудовані навколо власних фітнес-пристроїв і сервісів персональної аналітики, де мобільний застосунок виступає центром керування профілем, метриками та рекомендаціями. До таких рішень належить екосистема Fitbit, яка передбачає можливість відстеження активності як за допомогою смартфона, так і шляхом підключення трекерів або смартгодинників. У описі застосунку наголошується на підтримці підрахунку кроків і дистанції,

відстеженні витрачених калорій, а також на механізмах постановки цілей і ведення щоденника прогресу, що інтегрується з даними носимих пристроїв.

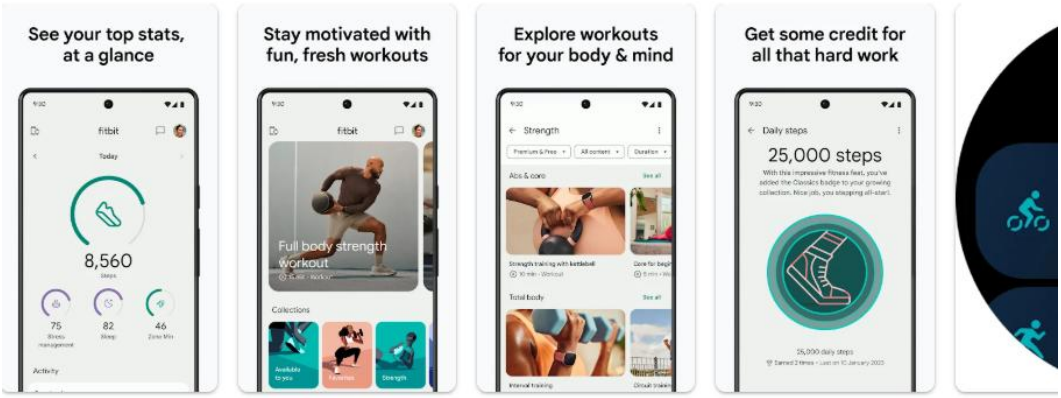

Рисунок 2.3. Інтерфейс Fitbit

Характерною особливістю таких рішень є поєднання автоматизованого збору показників з «поведінковими підказками», зокрема нагадуваннями про необхідність рухатися, інтерпретацією навантаження та формуванням рекомендацій за рахунок агрегування багатьох сигналів. Унаслідок цього користувач отримує не лише фактологічну статистику, а й контекстні висновки, які можуть підсилювати мотивацію або сприяти корекції режиму. Для наукового аналізу важливо, що подібні застосунки часто використовують власні метрики, які спрощують складні фізіологічні процеси до зрозумілих індикаторів, а також можуть включати різні рівні сервісу, у тому числі підписки на розширені аналітичні функції, що впливає на доступність персоналізованих рекомендацій для різних груп користувачів.

Ще один напрям у моніторингу фізичної активності пов’язаний із застосунками, орієнтованими на реєстрацію тренувальних сесій та соціальну взаємодію. Такі системи зазвичай фокусуються на активностях, що фіксуються за допомогою GPS, а також на обміні результатами, участі в челенджах і конкурентних механізмах. Зокрема, Strava позиціонується як застосунок для відстеження активностей із можливістю запису тренувань, аналізу зусиль і взаємодії у спільноті, де користувач може ділитися результатами та стежити за

прогресом інших.

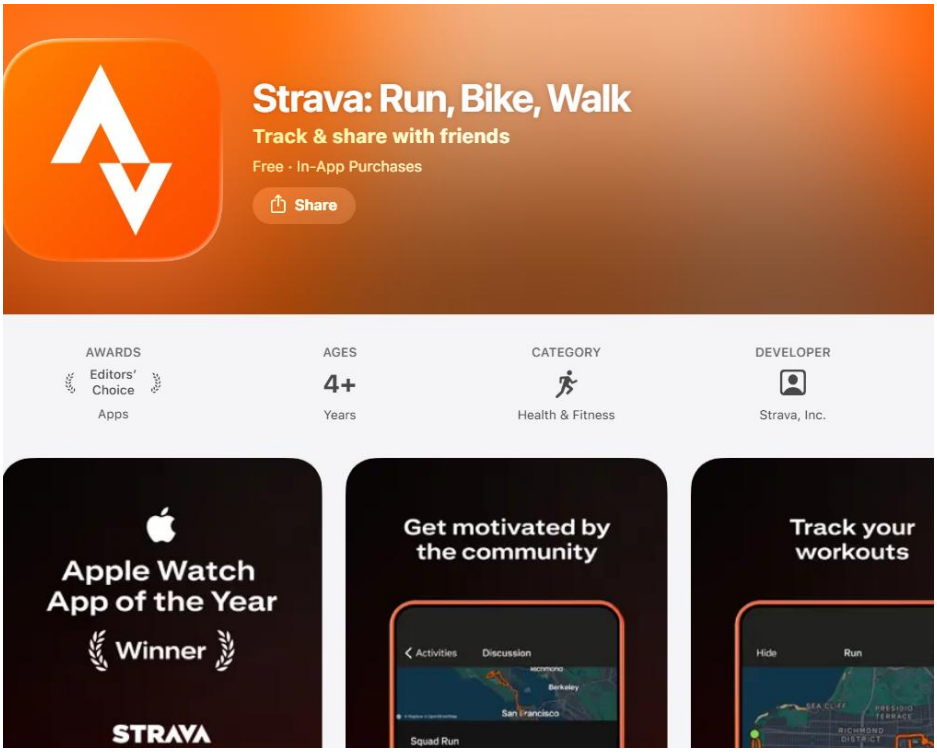

Рисунок 2.4. Інтерфейс застосунку Strava

Соціальний компонент у таких застосунках виконує роль зовнішнього мотиватора, оскільки створює ефект спільної діяльності, а також може підсилювати дисципліну за рахунок публічності результатів. Водночас для задач повсякденного моніторингу ходьби й загальної активності цей клас рішень не завжди оптимальний, адже ключовими тут стають не лише тренування як подія, а й повсякденна рухливість, що вимагає пасивного, непомітного для користувача відстеження. Тому Strava та подібні продукти частіше розглядаються як доповнення або як спеціалізований інструмент для спортсменів-аматорів, тоді як універсальні платформи більше відповідають сценаріям безперервного щоденного контролю.

Порівняльний аналіз сучасних застосунків для моніторингу фізичної активності дозволяє виокремити декілька принципових підходів до формування користувацької цінності. Перший підхід ґрунтується на простому обліку

показників і візуалізації прогресу; він характерний для застосунків, що використовують сенсори смартфона та мінімізують вимоги до додаткових пристроїв.

Другий підхід полягає у створенні «здоров'язбережувальної панелі керування», де активність є лише одним з елементів ширшого профілю здоров'я; для цього потрібна інтеграція з носимими пристроями та додатковими джерелами даних, що реалізується у межах екосистемних продуктів виробників [11]. Третій підхід орієнтований на тренування як основну одиницю аналізу та на соціальну мотивацію; він характерний для платформ із GPS-трекінгом і мережевою взаємодією. У контексті формування персональних рекомендацій найбільш перспективними є рішення, що поєднують пасивний збір показників (кроки, калорії, активний час) із можливістю інтерпретації даних та адаптивними підказками, оскільки саме вони забезпечують перехід від «фіксації» до «впливу» на поведінку.

З науково-практичної точки зору, центральною проблемою більшості застосунків для щоденного моніторингу активності є точність і інтерпретованість метрик. Підрахунок кроків залежить від якості сенсорів, алгоритмів фільтрації шумів, позиції пристрою, типу руху, а також від доступності апаратного крокоміра в конкретній моделі смартфона. Оцінювання витрачених калорій у масових застосунках переважно ґрунтується на наближених формулах або на емпіричних коефіцієнтах, що можуть давати прийнятні результати для орієнтовного контролю, але потребують коректного пояснення користувачеві, аби уникнути хибних висновків щодо реального енергетичного балансу. Універсальні застосунки, як правило, прагнуть до простоти, тому подають калорії як похідний індикатор від активності; екосистемні рішення можуть уточнювати розрахунки за рахунок частоти серцевих скорочень, даних про сон і рівень відновлення, що підвищує персоналізацію, але одночасно збільшує залежність від додаткових пристроїв [11].

Не менш важливою є проблема мотивації та утримання користувача, яка безпосередньо впливає на практичну ефективність застосунку. Застосунки можуть використовувати візуальні індикатори прогресу, системи цілей, нагадування, досягнення, а також пояснювальні підказки, які інтерпретують активність у категоріях, зрозумілих неспеціалісту. У цьому контексті персоналізований коучинг, що враховує історію активності, стає важливим фактором підтримки інтересу користувача та поступового підвищення рівня навантаження до рекомендованих значень. Саме наявність підказок і “дієвих рекомендацій” у загальних описах Google Fit відображає спрямованість сучасних платформ на поведінковий компонент, а не лише на статистику [10]. Водночас комерційні рішення на кшталт Fitbit можуть переносити частину розширених аналітичних функцій у платні рівні сервісу, що впливає на доступність персоналізації для всіх користувачів.

Таким чином, огляд сучасних мобільних застосунків для моніторингу фізичної активності демонструє, що ринок пропонує широкий спектр рішень - від універсальних крокомірів до комплексних екосистем здоров’я та спортивних соціальних платформ. Спільною для них є базова орієнтація на автоматизований збір даних і візуалізацію прогресу, тоді як відмінності полягають у ступені персоналізації, глибині аналітики, залежності від носимих пристроїв та акценті на соціальній взаємодії. У контексті задачі розробки застосунку персональних рекомендацій для оптимізації фізичної активності найбільш релевантними є підходи, що поєднують пасивний моніторинг кроків і похідних показників з адаптивними рекомендаціями, а також забезпечують прозорість розрахунків і доступність ключових функцій без надмірної залежності від додаткового обладнання. Це створює методологічне підґрунтя для обґрунтування власного програмного рішення, у якому алгоритмізація розрахунків і логіки рекомендацій виступає центральним елементом, а сучасні інструменти Android-розробки забезпечують реалізацію надійної архітектури та зручного інтерфейсу.

2.2. Порівняльний аналіз функціональних можливостей систем підрахунку кроків і калорій

Порівняльний аналіз сучасних мобільних програмних засобів для моніторингу фізичної активності дозволяє не лише виявити їх функціональні можливості, але й оцінити рівень реалізації ключових алгоритмічних і архітектурних підходів. Для цього розглядаються як універсальні застосунки, призначені для широкої аудиторії, так і рішення, орієнтовані на специфічні випадки використання або інтегровані в екосистеми виробників пристроїв. Серед основних критеріїв порівняння варто враховувати спосіб збору даних, математичні моделі обробки показників фізичної активності, наявність адаптивних алгоритмів формування рекомендацій, а також інтеграцію з іншими сервісами і пристроями.

Базові застосунки, що входять до стандартного набору мобільних сервісів, такі як Google Fit, реалізують прості алгоритми обробки даних про фізичну активність. Вони збирають інформацію про кроки, дистанцію і витрачені калорії без використання складних математичних моделей, що дозволяє забезпечити високу швидкодію і низьке енергоспоживання. Водночас підрахунок калорій в Google Fit здебільшого базується на емпіричних коефіцієнтах і стандартних формулах, що можуть бути застосовані до широкої групи користувачів. Такий підхід ефективний для базового моніторингу, але має обмеження щодо персоналізації, оскільки не враховує індивідуальних особливостей фізіології користувача в повній мірі і не забезпечує складні рекомендаційні алгоритми [10].

Samsung Health як екосистемний продукт пропонує ширший набір показників, включаючи профілі активності, дані про сон та серцево-судинну

діяльність при використанні додаткових пристроїв. Їх алгоритми опираються на більший обсяг вхідних даних, що підвищує точність оцінок у порівнянні з універсальними рішеннями. Однак такий підхід водночас підвищує залежність від апаратного забезпечення: для повноцінної роботи всіх функцій потрібні фітнес-браслети або смартгодинники, що обмежує доступність рішення для користувачів лише зі смартфоном. У Samsung Health реалізація адаптивних рекомендацій частково інтегрована з даними про сон і серцебиття, що дозволяє сформувати більш комплексну оцінку стану активності і відпочинку, а також запропонуати корисні підказки щодо режиму дня [11].

Серед рішень, що орієнтовані на спортивну спільноту та реєстрацію тренувань, Strava виділяється можливістю детального аналізу тренувальних сесій за допомогою GPS і побудови треків активності. Алгоритми Strava зосереджені на аналізі параметрів руху під час конкретних тренувань, що дозволяє добре адаптуватися під потреби аматорських спортсменів. Вони включають розрахунок швидкості, темпу, дистанції і витрачених калорій на основі даних трекінгу, що робить алгоритмічну частину більш складною і контекстною, ніж у базових крокомірів. Однак для щоденного моніторингу повсякденної активності та формування персональних рекомендацій для широкої аудиторії цей підхід не завжди є оптимальним, оскільки він вимагає активного включення користувача у фіксацію тренувань і часто не забезпечує автоматичного відстеження щоденних кроків.

Фітнес-застосунок Fitbit, що інтегрується з власними носимими пристроями, поєднує в собі елементи базового моніторингу та адаптивної аналітики. Його алгоритми формують рекомендації на основі не лише даних про кроки та калорії, але і трендів активності, що були накопичені впродовж тривалого періоду. За рахунок цього вирізняється можливість визначати не лише поточний стан користувача, але й тенденції зміни активності. Персоналізовані підказки, які пропонуються користувачеві, ґрунтуються на індивідуальному профілі і лімітах активності, що створює передумови для

більш диференційованого впливу на поведінку користувача, ніж у випадку універсальних застосунків [11, 12] .

Порівняльний аналіз також виявляє відмінності у підходах до обробки даних сенсорів. У простих крокомірах і загальних застосунках більшість обчислень здійснюється безпосередньо на пристрої з використанням стандартних сенсорних API, що забезпечує автономність і швидкість роботи, але обмежує можливості поглибленого аналізу. Натомість рішення, що інтегруються з носимими пристроями, мають доступ до більш широкого спектру даних - частоти серцевих скорочень, коливань вектора руху та інших параметрів, - що дозволяє будувати більш складні математичні моделі, наприклад моделі адаптивної енергетичної витрати або профілю активності користувача. При цьому такий підхід потребує синхронізації даних та врахування затримок при передачі даних між пристроями, а також ускладнює архітектуру програмного забезпечення.

Слід зазначити, що в контексті формування персональних рекомендацій важливо не лише наявність математичних моделей, але і спосіб інтеграції цих моделей у користувацький досвід. Універсальні застосунки, як Google Fit, зазвичай обмежуються базовою візуалізацією і стандартними порадами, які не завжди відповідають складним індивідуальним потребам. У той же час екосистемні рішення можуть пропонувати рекомендації з урахуванням більшої кількості параметрів, але часто їх доступність обмежується підпискою або додатковими пристроями. Це створює певний розрив між алгоритмічними можливостями і практичною доступністю для кінцевого користувача.

Аналіз архітектурних підходів виявляє, що застосунки з розвиненими рекомендаційними системами частіше використовують модульні дизайни з чітким розділенням на шари представлення, бізнес-логіки та даних, що полегшує подальше розширення функціональності і підтримку алгоритмів аналізу. Такий підхід часто реалізується з використанням шаблонів архітектури MVVM або Clean Architecture, що дозволяє ізолювати алгоритмічні компоненти

від деталей реалізації інтерфейсу. Це сприяє кращому тестуванню та підтримці адаптивних компонентів, які формують рекомендації. У мобільних рішеннях з меншою кількістю функцій рекомендувальної логіки архітектура може бути спрощена, що, з одного боку, полегшує розробку, але, з іншого боку, ускладнює інтеграцію нових алгоритмів без значних змін існуючого коду.

Таким чином, порівняльний аналіз сучасних програмних засобів для моніторингу фізичної активності показує, що реалізація алгоритмів обробки даних і формування рекомендацій значною мірою залежить від цільової аудиторії застосунку, наявності апаратних сенсорів та інтеграції з екосистемами. Універсальні погляди на моніторинг активності забезпечують простоту й доступність, тоді як комплексні системи пропонують більш глибоку аналітику і рекомендації, але часто за рахунок підвищених вимог до обладнання або сервісів. Для задачі розробки застосунку персональних рекомендацій доцільно поєднувати ефективні алгоритмічні моделі з високим рівнем доступності, що створює підґрунтя для подальшого формального обґрунтування вибору архітектурних і реалізаційних рішень.

2.3. Аналіз алгоритмів збору та обробки даних фізичної активності в існуючих рішеннях

Алгоритми збору та обробки даних фізичної активності є ключовим компонентом сучасних мобільних фітнес-застосунків, оскільки саме від їх коректності та адекватності залежить точність показників, що відображаються користувачеві, а також обґрунтованість сформованих рекомендацій. У більшості існуючих рішень процес обробки даних реалізується як послідовність етапів, що включають отримання первинних сенсорних сигналів, їх фільтрацію та агрегацію, математичне перетворення у зрозумілі метрики фізичної

активності й подальшу інтерпретацію результатів у контексті персональних характеристик користувача.

Основним джерелом даних у застосунках, орієнтованих на повсякденну активність, є сенсори руху мобільного пристрою. Найбільш поширеним є використання апаратного крокоміра, який реалізований у вигляді сенсора типу step counter або step detector. Алгоритмічно такий сенсор не фіксує окремі прискорення, а повертає накопичене значення кількості кроків з моменту останнього перезавантаження пристрою. Тому в Google Fit, Samsung Health та подібних застосунках добова кількість кроків визначається як різниця між поточним показником сенсора та значенням, зафіксованим на початку доби [10-19].

Формально цей підхід може бути представлений співвідношенням:

$S_{day} = S_{current} - S_{start}$,

де S_{day} - кількість кроків за поточний день,

$S_{current}$ - поточне значення сенсора,

S_{start} - значення сенсора на момент початку доби або першої активації застосунку в цей день.

Такий алгоритм є обчислювально простим і енергоефективним, однак вимагає коректної обробки подій перезавантаження пристрою або повторної ініціалізації сенсора, що вирішується шляхом збереження проміжних значень у локальному сховищі.

У випадках, коли апаратний крокомір недоступний, деякі застосунки застосовують алгоритми детекції кроків на основі акселерометра. Такі алгоритми ґрунтуються на аналізі коливань прискорення по осях координат і виділенні характерних піків, що відповідають фазам кроку. Для цього використовується фільтрація сигналу, наприклад низькочастотними або смуговими фільтрами, та порогова обробка, за якої крок фіксується у разі перевищення певного значення амплітуди. Хоча подібні алгоритми дозволяють працювати на ширшому спектрі пристроїв, їх точність значно залежить від

положення смартфона, стилю ходьби та рівня шуму сенсора, тому в масових застосунках вони поступово витісняються апаратними рішеннями.

На основі кількості кроків більшість фітнес-застосунків здійснює розрахунок похідних параметрів фізичної активності. Одним із таких параметрів є пройдена дистанція, яка у спрощеному вигляді визначається як добуток кількості кроків на середню довжину кроку. У багатьох застосунках довжина кроку оцінюється на основі зросту користувача за емпіричною формулою [10-19]

$L = k \cdot H,$

де L - довжина кроку, H - зріст користувача, а k - коефіцієнт, що зазвичай знаходиться в межах 0,41–0,45. Тоді дистанція D обчислюється як

$D = S\_day \cdot L.$

Такий підхід застосовується, зокрема, у Google Fit та Samsung Health, коли GPS-дані недоступні або не використовуються. У застосунках, орієнтованих на тренування з GPS, наприклад Strava, дистанція визначається шляхом інтеграції координат руху, що забезпечує вищу точність, але потребує активного використання геолокації та суттєво збільшує енергоспоживання.

Розрахунок витрачених калорій у більшості масових застосунків базується на наближених математичних моделях, що враховують кількість кроків, масу тіла користувача та інтенсивність активності. Найпростішою є лінійна модель [10-19], за якої витрати енергії пропорційні кількості кроків, наприклад

$C = S\_day \cdot c_0$,

де C - витрачені калорії, а c_0 - емпіричний коефіцієнт, який часто приймається на рівні 0,03–0,05 ккал за один крок. Саме подібний підхід застосовується у багатьох універсальних крокомірах і забезпечує орієнтовну оцінку енергетичних витрат. У більш складних моделях [10-19], що реалізуються в екосистемних застосунках, формула може враховувати масу тіла та швидкість руху, наприклад

$C = D \cdot m \cdot k_1$,

де D - дистанція, m - маса тіла користувача, а k_1 - коефіцієнт, що залежить від типу активності. У застосунках, які інтегруються з датчиками серцевого ритму, калорійність може коригуватися з урахуванням частоти серцевих скорочень, що підвищує індивідуальну точність, але ускладнює алгоритмічну модель.

Окрему групу алгоритмів становлять розрахунки, пов’язані з оцінюванням загального стану користувача. До них належить, зокрема, визначення індексу маси тіла, який у більшості застосунків [10-19] обчислюється за класичною формулою

$BMI = m / H^2$,

де m - маса тіла в кілограмах, а H - зріст у метрах. Отримане значення використовується не стільки для точного медичного діагнозу, скільки для формування загальних рекомендацій і інформаційних повідомлень. Аналогічно розрахунок рекомендованого водного балансу часто здійснюється за спрощеною моделлю, де базова норма визначається пропорційно масі тіла, наприклад

$W = m \cdot 30$,

де W - рекомендований об’єм води в мілілітрах, з можливим коригуванням залежно від рівня фізичної активності за день.

Аналіз існуючих рішень показує, що важливою частиною алгоритмів обробки даних є їх агрегація у часовому вимірі. Застосунки зазвичай оперують добовими, тижневими та місячними показниками, що дозволяє виявляти тенденції зміни активності. Для цього використовується збереження щоденних значень у локальних або хмарних базах даних і подальше обчислення середніх або сумарних показників. Наприклад, середня кількість кроків за тиждень може визначатися як середнє арифметичне добових значень, що використовується для оцінки стабільності активності користувача.

Особливу увагу в сучасних застосунках приділено алгоритмам

інтерпретації даних і формування рекомендацій. У простих рішеннях рекомендації мають статичний характер і залежать від досягнення або недосягнення денної цілі, наприклад пропозиція здійснити додаткову прогулянку у разі значного відставання від плану. У більш розвинених системах застосовується аналіз історичних даних, що дозволяє коригувати цілі або інтенсивність активності поступово, уникати різких змін навантаження та формувати більш персоналізовані поради. Хоча конкретні алгоритми таких рекомендацій зазвичай є закритими, їх логіка ґрунтується на поєднанні простих математичних моделей і евристичних правил, що забезпечують адаптивність без надмірної складності.

Отже, аналіз алгоритмів збору та обробки даних фізичної активності в існуючих мобільних застосунках свідчить про домінування прагматичних, обчислювально простих підходів, які забезпечують баланс між точністю, енергоефективністю та доступністю для широкого кола користувачів. Більшість рішень використовує апаратний крокомір як основне джерело даних, застосовує наближені формули для розрахунку похідних показників і поєднує їх з елементарною персоналізацією на основі анкетних даних користувача. Саме ці підходи формують методологічну основу для розробки власного застосунку персональних рекомендацій, у якому алгоритмізація збору й обробки даних має бути прозорою, логічно обґрунтованою та адаптованою до реальних умов використання мобільного пристрою.

2.4. Аналіз інтерфейсів користувача та мотиваційних механізмів у фітнес-додатках

Інтерфейс користувача відіграє визначальну роль у ефективності фітнес-застосунків, оскільки саме через нього користувач взаємодіє з алгоритмами

збору та обробки даних фізичної активності. Навіть за наявності коректних математичних моделей і точних сенсорних вимірювань, низька зручність або перевантаженість інтерфейсу можуть істотно знизити практичну цінність програмного продукту. Тому сучасні мобільні фітнес-додатки орієнтовані не лише на функціональність, а й на створення зрозумілого, інтуїтивного та мотиваційного середовища, яке стимулює користувача до регулярного використання застосунку та підтримки фізичної активності.

Однією з характерних рис інтерфейсів фітнес-додатків є домінування інформаційних панелей, або так званих дашбордів, на головному екрані. У таких інтерфейсах основні показники фізичної активності - кількість кроків, витрачені калорії, пройдена дистанція та ступінь досягнення денної цілі - подаються у компактному й візуально наочному вигляді. Зокрема, у Google Fit центральним елементом інтерфейсу виступають кільцеві індикатори активності, які заповнюються пропорційно до виконання заданих цілей. Подібна форма візуалізації дозволяє користувачеві миттєво оцінити свій прогрес без необхідності аналізу числових значень, що суттєво знижує когнітивне навантаження та підвищує зручність сприйняття інформації.

Інтерфейси екосистемних застосунків, таких як Samsung Health, зазвичай мають більш складну структуру, оскільки поєднують дані з кількох джерел і охоплюють ширший спектр параметрів. У таких рішеннях головний екран часто є модульним і дозволяє користувачеві самостійно налаштовувати набір віджетів або карток, що відображаються. Це забезпечує гнучкість інтерфейсу та адаптацію до індивідуальних потреб, однак водночас підвищує складність навігації для недосвідчених користувачів. З точки зору мотивації, подібні інтерфейси апелюють до відчуття контролю та залученості, оскільки користувач отримує можливість формувати власний інформаційний простір.

Окрему увагу в інтерфейсах фітнес-застосунків приділено поданню історичних даних. Більшість сучасних рішень реалізує перегляд активності у вигляді графіків або списків за днями, тижнями чи місяцями. Такий підхід

дозволяє користувачеві відстежувати динаміку змін і формувати уявлення про регулярність фізичної активності. Наприклад, у застосунках на кшталт Fitbit історичні дані часто супроводжуються короткими аналітичними висновками або порівняннями з попередніми періодами, що виконує додаткову мотиваційну функцію. Аналіз тенденцій активності сприяє усвідомленню результатів власних зусиль і формує позитивне підкріплення у разі поступового зростання показників.

Мотиваційні механізми у фітнес-додатках тісно пов’язані з принципами поведінкової психології та теоріями формування звичок. Одним із базових інструментів мотивації є система цілей, за якої користувач задає або приймає рекомендовану денну норму кроків чи іншого виду активності. Досягнення цілі супроводжується візуальними або текстовими сигналами успіху, що формує позитивне емоційне підкріплення. У багатьох застосунках досягнення цілі відображається не лише у вигляді числового показника, а й через анімацію або зміну кольорової гами інтерфейсу, що підсилює ефект задоволення від виконаної дії.

Ще одним поширеним мотиваційним механізмом є використання нагадувань і підказок. Фітнес-додатки можуть надсилати сповіщення з рекомендаціями зробити перерву на рух або повідомлення про відставання від денної цілі. У Google Fit такі повідомлення часто мають нейтральний або заохочувальний характер і спрямовані на м’яке стимулювання активності без тиску на користувача. В екосистемних застосунках повідомлення можуть бути більш контекстними і враховувати історію активності, наприклад пропонувати коротку прогулянку після тривалого періоду малорухомості. Таким чином інтерфейс стає не лише засобом відображення інформації, а й активним учасником процесу формування поведінкових патернів.

Соціальні елементи також відіграють важливу роль у мотивації, хоча ступінь їх інтеграції суттєво відрізняється між різними класами застосунків. У соціально орієнтованих платформах, таких як Strava, інтерфейс підтримує

взаємодію між користувачами шляхом публікації результатів, участі у змаганнях і коментування активностей. Це створює додатковий зовнішній стимул, заснований на порівнянні з іншими та відчутті приналежності до спільноти. Водночас для застосунків, орієнтованих на повсякденну активність широкого кола користувачів, соціальні функції часто є другорядними або відсутніми, що дозволяє зосередитися на персональній мотивації та уникнути негативного ефекту порівняння.

Важливим аспектом аналізу інтерфейсів є баланс між інформативністю та простотою. Надмірна кількість показників і візуальних елементів може перевантажувати користувача й знижувати ефективність сприйняття інформації. Саме тому сучасні фітнес-додатки прагнуть використовувати ієрархічну структуру інтерфейсу, де ключові показники розміщуються на головному екрані, а деталізована інформація доступна за потреби. Такий підхід дозволяє адаптувати інтерфейс до різних рівнів зацікавленості користувача й підтримувати як швидкий огляд стану активності, так і поглиблений аналіз.

Отже, аналіз інтерфейсів користувача та мотиваційних механізмів у фітнес-додатках свідчить про тісний взаємозв’язок між візуальним поданням даних, зручністю взаємодії та ефективністю стимулювання фізичної активності. Сучасні рішення поєднують прості та наочні дашборди, системи цілей, історичну аналітику та контекстні підказки, що в сукупності сприяє формуванню сталих звичок рухової активності. Результати такого аналізу є важливими для подальшого обґрунтування власного підходу до проєктування інтерфейсу та мотиваційної логіки у розроблюваному застосунку персональних рекомендацій.

2.5. Визначення недоліків існуючих рішень та обґрунтування необхідності розробки власного застосунку

Проведений аналіз сучасних мобільних застосунків для моніторингу фізичної активності, їх алгоритмічних підходів та інтерфейсних рішень свідчить про значний прогрес у сфері цифрової підтримки здорового способу життя. Водночас, попри широкий функціональний спектр і популярність таких продуктів, існуючі рішення мають низку суттєвих недоліків, які обмежують їх ефективність у контексті персоналізованої оптимізації фізичної активності та обґрунтовують доцільність розробки власного програмного застосунку.

Однією з основних проблем більшості масових фітнес-додатків є недостатня прозорість алгоритмів обробки даних. Універсальні застосунки, такі як Google Fit або Samsung Health, зазвичай приховують деталі розрахунків кількості витрачених калорій, дистанції чи інших похідних показників, використовуючи закриті або узагальнені алгоритми. Це унеможливлює адаптацію моделей під конкретні умови використання та не дозволяє користувачеві або розробникові чітко оцінити точність отриманих результатів. У науковому та освітньому контексті така непрозорість є істотним недоліком, оскільки не дає змоги формалізувати процес обробки даних і дослідити вплив окремих параметрів на кінцеві показники фізичної активності.

Ще одним суттєвим обмеженням існуючих рішень є орієнтація значної частини функціональності на використання додаткових носимих пристроїв. Екосистемні застосунки, зокрема Fitbit або Samsung Health, демонструють підвищену точність і ширші аналітичні можливості лише за умови підключення смартгодинників чи фітнес-браслетів. У разі використання лише смартфона функціональність таких систем істотно скорочується, що знижує доступність персоналізованого аналізу для широкого кола користувачів. Це створює залежність від апаратного забезпечення та комерційних екосистем, що не завжди є прийнятним з точки зору автономності та універсальності програмного продукту.

Аналіз мотиваційних механізмів показує, що у багатьох застосунках

рекомендації мають загальний або статичний характер. Найчастіше вони зводяться до повідомлень про досягнення або недосягнення денної цілі без глибокого урахування індивідуальної динаміки активності користувача. Навіть у розширених системах персоналізація рекомендацій часто обмежується історією кроків і не враховує комплекс взаємопов'язаних показників, таких як маса тіла, індекс маси тіла, водний баланс і довготривалі тенденції активності. У результаті рекомендаційні механізми виконують переважно інформативну або мотиваційну функцію, але не забезпечують повноцінної оптимізації фізичної активності як процесу, що має алгоритмічно керований характер.

З точки зору програмної архітектури, значна частина комерційних фітнес-додатків є складними багатокомпонентними системами, тісно інтегрованими з хмарними сервісами. Такий підхід, з одного боку, забезпечує масштабованість і синхронізацію даних, але, з іншого боку, ускладнює аналіз внутрішньої логіки роботи застосунку та знижує автономність користувача. Для дослідницьких і навчальних цілей більш доцільним є застосунок, у якому основні алгоритми збору, обробки й аналізу даних реалізуються локально та мають чітко структуровану логіку, доступну для аналізу й модифікації.

Виявлені недоліки зумовлюють необхідність розробки власного мобільного застосунку, орієнтованого на алгоритмізацію процесів моніторингу та оптимізації фізичної активності. Запропонований у межах даної дипломної роботи програмний продукт усуває зазначені обмеження шляхом реалізації прозорих математичних моделей розрахунку основних показників фізичної активності та чіткої логіки формування персональних рекомендацій. Використання апаратного крокоміра смартфона як основного джерела даних дозволяє забезпечити автономність роботи застосунку без необхідності підключення додаткових пристроїв, що підвищує його доступність і універсальність.

Архітектура розроблюваного застосунку, побудована на принципах Clean Architecture та шаблону MVVM, забезпечує чітке розділення рівнів

представлення, бізнес-логіки та доступу до даних. Такий підхід, реалізований у коді проєкту, дозволяє ізолювати алгоритми розрахунків у відповідних use-case модулях, що спрощує їх тестування, модифікацію та подальше розширення. Локальне збереження історії активності за допомогою Room Database та персональних налаштувань користувача через DataStore забезпечує принцип єдиного джерела правди й дозволяє здійснювати аналіз даних у часовій динаміці без залежності від зовнішніх сервісів.

Особливістю розроблюваного застосунку є орієнтація на поєднання моніторингу фізичної активності з алгоритмічно обґрунтованими рекомендаціями, що враховують не лише поточні показники, а й індивідуальні параметри користувача. Реалізовані в коді проєкту алгоритми розрахунку калорій, дистанції, індексу маси тіла та водного балансу мають чітку формальну основу, що дозволяє інтерпретувати результати та забезпечує логічну узгодженість між зібраними даними і сформованими порадами.

Таким чином, розробка власного мобільного застосунку персональних рекомендацій є обґрунтованою відповіддю на виявлені недоліки існуючих рішень. Запропонований підхід поєднує доступність, прозорість алгоритмів, автономність роботи та сучасні архітектурні принципи, що створює передумови для ефективної оптимізації фізичної активності користувачів і підтверджує доцільність реалізації даного програмного продукту в межах дипломної роботи.

РОЗДІЛ 3. ТЕОРЕТИЧНА ЧАСТИНА

3.1. Теоретичні основи моніторингу фізичної активності за допомогою мобільних сенсорів

Моніторинг фізичної активності за допомогою мобільних сенсорів ґрунтується на використанні апаратних компонентів сучасних смартфонів, здатних реєструвати параметри руху користувача в реальному часі. Теоретичною основою такого підходу [див. 10-19] є уявлення про рух людини як послідовність механічних дій, що супроводжуються змінами прискорення, орієнтації та швидкості, які можуть бути зафіксовані відповідними сенсорами. Найбільш значущими з точки зору аналізу фізичної активності є акселерометр, гіроскоп та апаратний крокомір, які у сукупності дозволяють оцінювати інтенсивність руху, кількість кроків і загальний рівень активності без прямого втручання користувача.

Акселерометр є базовим сенсором, що вимірює прискорення вздовж трьох взаємно перпендикулярних осей. З фізичної точки зору дані акселерометра відображають суму лінійного прискорення та прискорення вільного падіння, що ускладнює безпосередню інтерпретацію сигналів. Для використання акселерометричних даних у задачах моніторингу фізичної активності застосовуються методи фільтрації, які дозволяють відокремити корисний сигнал, пов’язаний із рухами користувача, від шумів і постійної гравітаційної складової. Теоретично такий підхід базується на припущенні, що ходьба людини створює квазіперіодичні коливання прискорення, амплітуда і частота яких корелюють із темпом і характером руху. Саме ця властивість використовується в алгоритмах детекції кроків на основі акселерометра, де кожен крок відповідає локальному максимуму або мінімуму сигналу після відповідної обробки.

Однак у сучасних мобільних операційних системах, зокрема Android,

дедалі ширше використовується апаратний крокомір, який реалізується на рівні мікропрограмного забезпечення пристрою. Теоретично апаратний крокомір можна розглядати як реалізацію узагальненого алгоритму детекції кроків, оптимізованого для конкретної апаратної платформи. Він повертає накопичене значення кількості кроків, що були виконані користувачем з моменту останнього перезавантаження пристрою. Такий підхід знімає з розробника необхідність реалізовувати власні алгоритми обробки сирих акселерометричних даних і водночас забезпечує високу енергоефективність, оскільки обчислення виконуються на рівні сенсорного процесора.

У теоретичному аспекті використання апаратного крокоміра ґрунтується на моделі накопичувального лічильника, значення якого монотонно зростає з кожним зафіксованим кроком. Для визначення кількості кроків за певний часовий інтервал, наприклад за добу, застосовується диференціальний підхід, за якого добове значення визначається як різниця між поточним показником сенсора та значенням, зафіксованим на початку періоду. Саме цей принцип реалізований у коді розроблюваного застосунку, де клас, відповідальний за роботу з сенсорами, отримує події від Sensor.TYPE_STEP_COUNTER і зберігає початкове значення лічильника для подальших розрахунків. Такий підхід забезпечує коректне визначення кількості кроків незалежно від попередньої активності користувача та дозволяє відновлювати стан після перезапуску застосунку.

Моніторинг фізичної активності не обмежується лише підрахунком кроків, а передбачає подальшу обробку отриманих даних з метою оцінювання похідних показників. Теоретично кількість кроків розглядається як первинна дискретна величина, з якої можуть бути виведені безперервні характеристики руху, зокрема дистанція та енергетичні витрати. Для цього застосовуються емпіричні математичні моделі, що пов’язують кількість кроків з антропометричними параметрами користувача. У найпростішому випадку пройдена дистанція визначається як добуток кількості кроків на середню

довжину кроку, яка, у свою чергу, апроксимується через зріст користувача. Така модель є узагальненою, однак її точність є достатньою для орієнтовного моніторингу повсякденної активності.

Аналогічно витрати енергії в більшості мобільних застосунків оцінюються на основі спрощених лінійних моделей, що пов’язують кількість кроків або дистанцію з масою тіла користувача. З теоретичної точки зору такі моделі є наближеними і не враховують усіх біомеханічних та фізіологічних чинників, проте вони дозволяють отримати стабільні й інтерпретовані значення калорій, придатні для мотиваційного та аналітичного використання. У коді проєкту ці розрахунки зосереджені в окремому модулі бізнес-логіки, де на основі даних сенсора та параметрів профілю користувача формується агрегований набір показників фізичної активності. Така організація відповідає теоретичним засадам модульності та розділення відповідальностей, за яких сенсорний рівень відповідає лише за збір даних, а аналітичний - за їх інтерпретацію.

Важливим теоретичним аспектом моніторингу фізичної активності є часовий вимір даних. Фізична активність має яскраво виражену динаміку, тому її аналіз передбачає агрегацію показників у певних часових інтервалах. У більшості застосунків базовою одиницею аналізу є доба, що відповідає рекомендаціям щодо щоденної рухової активності. З точки зору теорії обробки даних, це означає необхідність збереження дискретних значень активності у часовому ряді та можливість подальшого аналізу тенденцій. У розроблюваному застосунку ця концепція реалізується через збереження щоденних показників у локальній базі даних, що дозволяє не лише відображати історію активності, а й використовувати її для формування рекомендацій.

Таким чином, теоретичні основи моніторингу фізичної активності за допомогою мобільних сенсорів базуються на поєднанні фізичних принципів вимірювання руху, математичних моделей обробки даних та програмних механізмів агрегації й інтерпретації інформації. Використання апаратного

крокоміра як основного джерела даних дозволяє забезпечити високу точність і енергоефективність, а застосування формалізованих алгоритмів обробки та збереження результатів створює основу для подальшої реалізації персоналізованих рекомендацій. Саме ці теоретичні положення визначають концептуальні засади побудови розроблюваного мобільного застосунку та обґрунтовують вибір відповідних програмних і архітектурних рішень.

3.2. Математичні моделі розрахунку показників фізичної активності

Математичні моделі розрахунку показників фізичної активності є основою для інтерпретації первинних сенсорних даних та перетворення їх у зрозумілі й практично значущі характеристики стану користувача. У мобільних фітнес-застосунках такі моделі мають прикладний характер і спрямовані на досягнення балансу між точністю, обчислювальною простотою та стабільністю результатів у реальних умовах використання. На відміну від медичних або лабораторних систем, де можливе застосування складних біомеханічних моделей, мобільні застосунки використовують спрощені, але емпірично обґрунтовані формули, які дозволяють оперативно оцінювати рівень фізичної активності на основі доступних параметрів [див. 10-19].

Для розробленого нами застосунку ми будемо використовувати загальноприйняті підходи для розрахунку всіх необхідних величин. Докладно вони були описані у джерелах 10-19 та розглянуті у розділі 2 нашого дослідження. Базовою величиною для більшості розрахунків виступає кількість кроків, отримана з апаратного крокоміра мобільного пристрою. З математичної точки зору кількість кроків є дискретною випадковою величиною, що відображає інтенсивність рухової активності за певний часовий інтервал. Саме вона використовується як вхідний параметр для побудови інших моделей. Для

щоденного моніторингу фізичної активності кількість кроків агрегується в межах доби, що узгоджується з рекомендаціями щодо добового рівня рухової активності та спрощує подальший аналіз.

Одним із найпоширеніших похідних показників є пройдена дистанція. У теоретичному плані дистанція визначається як сума довжин усіх кроків, виконаних користувачем. Оскільки довжина кожного окремого кроку змінюється залежно від швидкості руху, рельєфу та фізіологічних особливостей, у мобільних застосунках застосовується модель середньої довжини кроку. Найчастіше вона апроксимується через зріст користувача, що дозволяє отримати універсальну формулу виду

$$D = S \cdot L,$$

де D - пройдена дистанція, S - кількість кроків, а L - середня довжина кроку. Значення L, у свою чергу, визначається як частка від зросту користувача, наприклад $L = k \cdot H$, де H - зріст, а k - емпіричний коефіцієнт, що зазвичай знаходиться в межах 0,41–0,45. Така модель використовується у більшості мобільних фітнес-застосунків, зокрема у випадках, коли GPS-дані недоступні або їх використання є недоцільним через підвищене енергоспоживання.

Наступним важливим показником фізичної активності є кількість витрачених калорій. З точки зору фізіології, енергетичні витрати залежать від маси тіла, інтенсивності руху, тривалості активності та індивідуальних особливостей обміну речовин. Проте у масових мобільних застосунках застосовуються спрощені математичні моделі, що дозволяють оцінити витрати енергії без використання складних біомедичних параметрів. Найпростішою є лінійна модель, у якій витрачені калорії пропорційні кількості кроків, тобто

$$C = S \cdot c_0 ,$$

де C - витрачені калорії, S - кількість кроків, а c_0 - емпіричний коефіцієнт, що зазвичай приймається в межах 0,03–0,05 ккал за крок. Саме така модель реалізована в коді розроблюваного застосунку, де для оцінювання калорій використовується коефіцієнт 0,04, що забезпечує стабільний і легко

інтерпретований результат для повсякденної активності.

У більш узагальнених моделях розрахунок калорій може здійснюватися на основі дистанції та маси тіла користувача, наприклад

$C = D \cdot m \cdot k_1$,

де m - маса тіла, а k_1 - коефіцієнт, що залежить від типу активності. Проте для задачі щоденного моніторингу така модель ускладнює розрахунки і не дає істотного виграшу в точності без додаткових сенсорних даних, тому в межах даного проєкту обрано простішу лінійну залежність.

Окреме місце серед показників фізичної активності займає індекс маси тіла, який використовується як узагальнений індикатор відповідності маси тіла зросту користувача. Індекс маси тіла визначається класичною формулою

$BMI = m / H^2$,

де m - маса тіла в кілограмах, а H - зріст у метрах. Отримане значення порівнюється з нормативними інтервалами, що дозволяє класифікувати стан користувача як нормальний, із недостатньою або надмірною масою тіла. У контексті мобільного застосунку цей показник використовується не для медичної діагностики, а як додатковий фактор при формуванні рекомендацій та інтерпретації рівня фізичної активності.

Ще одним прикладом прикладної математичної моделі є розрахунок рекомендованого водного балансу. У фітнес-застосунках він зазвичай визначається як функція маси тіла з урахуванням рівня фізичної активності. Базова модель може бути представлена у вигляді

$W = m \cdot 30$,

де W - рекомендований об’єм води в мілілітрах на добу. Для підвищення адаптивності результату у застосунку може додаватися коригувальна складова, пропорційна кількості кроків або рівню активності, що дозволяє врахувати підвищені втрати рідини під час руху. Такий підхід забезпечує простоту розрахунку і водночас формує у користувача уявлення про зв’язок між фізичною активністю та потребами організму.

Загальною рисою розглянутих математичних моделей є їх емпіричний характер і орієнтація на стабільність результатів у широкому діапазоні умов. Вони не претендують на високу медичну точність, проте є достатньо інформативними для задач моніторингу, мотивації та самоконтролю. У поєднанні з регулярним збором даних і їх агрегацією у часових інтервалах такі моделі дозволяють формувати цілісну картину фізичної активності користувача та створюють математичне підґрунтя для подальшої реалізації алгоритмів персональних рекомендацій у мобільному застосунку.

Саме ці підхлди і були реалізовані у нашому застосунку. Повний опис реалізованих функцій буде розглянуто у розділі 4 нашої дипломної роботи.

3.3. Алгоритмізація формування персональних рекомендацій щодо фізичної активності

Формування персональних рекомендацій щодо фізичної активності є логічним продовженням процесів збору та математичної обробки даних, оскільки саме на цьому етапі кількісні показники перетворюються на якісні поради, орієнтовані на зміну поведінки користувача. З алгоритмічної точки зору рекомендації розглядаються як результат аналізу поточного стану фізичної активності користувача у поєднанні з його індивідуальними параметрами та заданими цілями. Такий підхід дозволяє перейти від статичного відображення даних до динамічної підтримки процесу оптимізації фізичної активності.

Базовим елементом алгоритмізації рекомендацій є визначення поточного рівня активності відносно цільового значення. У розроблюваному застосунку це реалізується через порівняння фактичної кількості кроків за день із заданою денною нормою, що зберігається у профілі користувача. Формально цей етап може бути поданий як обчислення коефіцієнта виконання цілі, який

визначається відношенням поточної кількості кроків до цільового значення. Отримане значення використовується не лише для візуалізації прогресу, але й як вхідний параметр для алгоритму формування рекомендацій. Таким чином, рекомендаційна логіка спирається на кількісну оцінку ступеня досягнення мети, що забезпечує її формальну визначеність і відтворюваність.

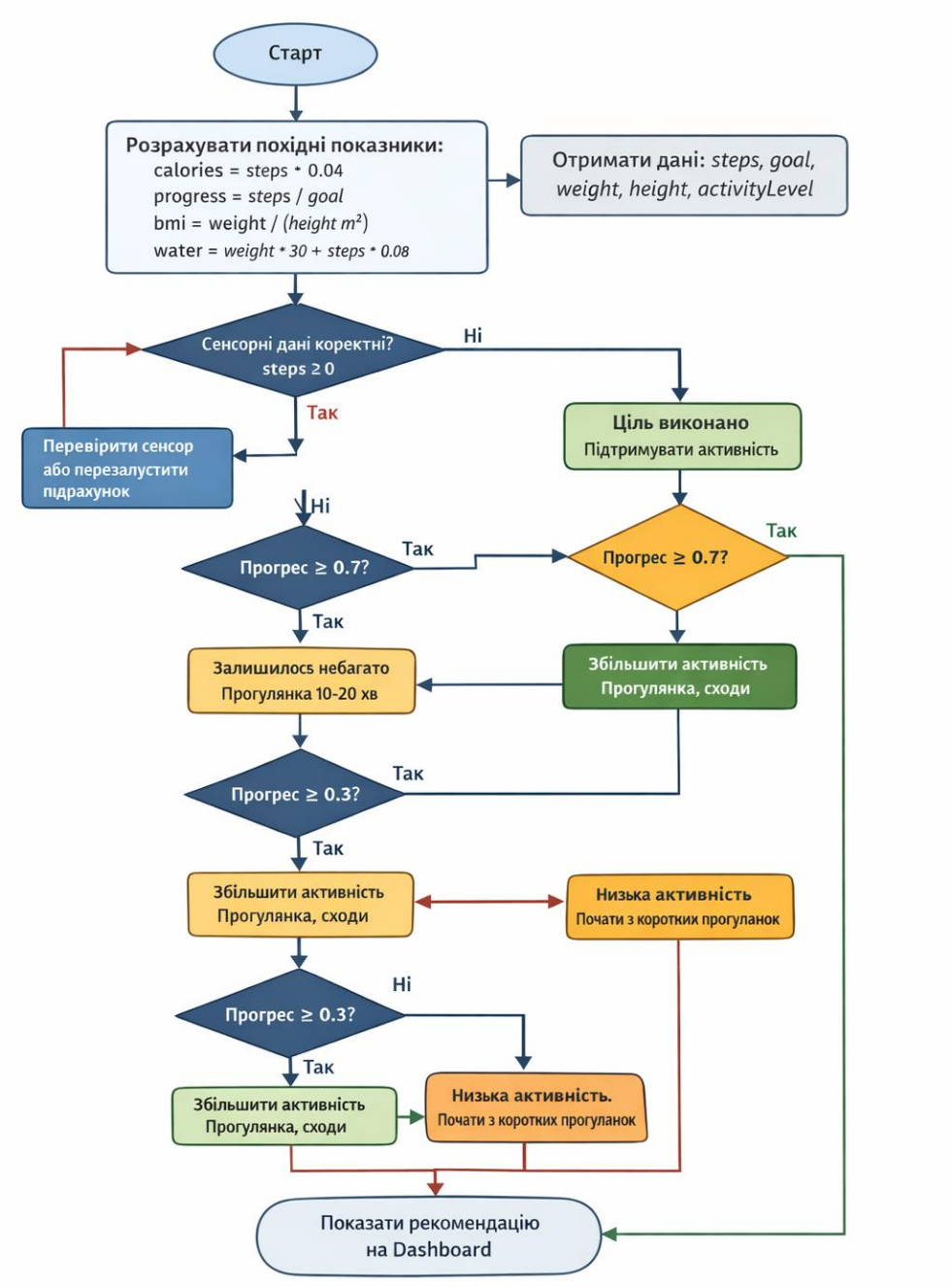

Рисунок 3.1 Алгоритм рекомендацій активності розробленого застосунку

Наступним етапом алгоритмізації є інтерпретація отриманого рівня активності. У застосунку використовується евристичний підхід, за якого числові показники переводяться у якісні стани, наприклад недостатня активність, активність у межах норми або перевищення рекомендованого рівня. Така інтерпретація дозволяє адаптувати рекомендації до поточної ситуації та уникати надмірного навантаження на користувача. Алгоритмічно це реалізується через умовні переходи, у яких порівнюються фактичні значення показників із заданими порогами, що визначають сценарій рекомендації. Подібний підхід є типовим для мобільних фітнес-застосунків, оскільки він поєднує простоту реалізації з достатнім рівнем персоналізації.

Важливим компонентом рекомендаційного алгоритму є врахування антропометричних характеристик користувача, зокрема маси тіла, зросту та індексу маси тіла. Ці параметри дозволяють скоригувати рекомендації з урахуванням загального фізичного стану користувача. Наприклад, для користувачів із недостатнім рівнем активності та підвищеним індексом маси тіла рекомендації можуть бути спрямовані на поступове збільшення кількості кроків, тоді як для користувачів із нормальною масою тіла акцент робиться на підтриманні стабільного рівня рухової активності. З алгоритмічної точки зору це означає використання комбінованих умов, у яких одночасно аналізуються кілька параметрів профілю та результатів моніторингу.

Окрім кількості кроків, у процесі формування рекомендацій використовуються й інші похідні показники фізичної активності, зокрема витрачені калорії та рекомендований водний баланс. Алгоритм рекомендацій враховує, що підвищена рухова активність призводить до збільшення енергетичних витрат і втрати рідини, тому поради щодо споживання води формуються як функція рівня активності за день. Таким чином рекомендаційна система охоплює не лише рухову активність, а й супутні аспекти здорового

способу життя, що підвищує її практичну цінність.

З точки зору програмної реалізації алгоритм формування рекомендацій інкапсульовано в окремому модулі бізнес-логіки, що відповідає принципам Clean Architecture. Такий підхід забезпечує відокремлення рекомендаційної логіки від інтерфейсу користувача та механізмів збору даних. У коді застосунку рекомендації формуються у межах use case, який агрегує дані про поточну активність та профіль користувача і на основі цього генерує текстове повідомлення рекомендаційного характеру. Це повідомлення автоматично оновлюється при зміні будь-якого з вхідних параметрів, що реалізує реактивний підхід до формування рекомендацій [5].

Особливістю алгоритмізації рекомендацій у даному застосунку є її адаптивність у часовому вимірі. Оскільки дані про фізичну активність накопичуються впродовж дня, рекомендації можуть змінюватися залежно від поточного часу та динаміки досягнення цілі. Наприклад, за значного відставання від плану у другій половині дня алгоритм може сформувати рекомендацію щодо короткої прогулянки або збільшення рухової активності, тоді як за умови дострокового досягнення цілі рекомендація матиме заохочувальний характер. Такий підхід сприяє більш природному включенню рекомендацій у повсякденний ритм життя користувача.

Загалом алгоритмізація формування персональних рекомендацій щодо фізичної активності у розроблюваному застосунку ґрунтується на поєднанні формалізованих математичних моделей, евристичних правил та індивідуальних параметрів користувача. Вона забезпечує логічну узгодженість між вимірюваними показниками та пропонованими порадами, а також створює основу для подальшого розширення рекомендаційної системи шляхом урахування додаткових факторів або більш складних моделей аналізу поведінки. Такий підхід дозволяє розглядати рекомендації не як статичні повідомлення, а як результат алгоритмічно керованого процесу оптимізації фізичної активності.

3.4. Проєктування архітектури мобільного застосунку на основі Clean Architecture та MVVM та побудова UML-діаграм

Проєктування архітектури мобільного застосунку у даній роботі ґрунтується на поєднанні принципів Clean Architecture та шаблону MVVM, що дозволяє забезпечити чітке розділення відповідальностей між компонентами системи, підвищити тестованість програмного продукту та спростити його подальший розвиток. Обраний архітектурний підхід орієнтований на відокремлення бізнес-логіки від деталей реалізації інтерфейсу користувача та механізмів доступу до даних, що є ключовою вимогою до сучасних Android-застосунків [1-8].

У межах Clean Architecture центральне місце займає доменна логіка, яка не залежить від платформи та технологій представлення. У розробленому застосунку ця концепція реалізована через доменний шар, у якому зосереджено математичні моделі та алгоритми обробки даних фізичної активності. Зокрема, сценарій формування даних для головного екрана інкапсульовано в окремому use case, який об’єднує інформацію про кількість кроків і профіль користувача та виконує всі необхідні розрахунки. У коді це реалізовано класом GetDashboardDataUseCase, який отримує дані з репозиторію та сховища налаштувань і повертає агрегований результат:

```
class GetDashboardDataUseCase(
    private val repository: ActivityRepository,
    private val userPreferences: UserPreferences
) {
    operator fun invoke(): Flow<DashboardData> {
        return combine(
```

```
            repository.getTodayActivity(),
            userPreferences.userProfile
        ) { stats, profile ->
            calculateData(stats, profile)
        }
    }
}
```

Такий підхід забезпечує незалежність алгоритмів розрахунку від інтерфейсу користувача та дозволяє розглядати бізнес-логіку як самодостатній компонент системи. Математичні моделі, описані в попередніх підрозділах, реалізуються саме на цьому рівні, що забезпечує повну відповідність між теоретичною та практичною частинами роботи.

Шар доступу до даних (Data Layer) відповідає за отримання інформації з різних джерел, зокрема апаратних сенсорів, локальної бази даних і сховища налаштувань. У розробленому застосунку збір даних про кроки реалізовано за допомогою окремого компонента StepSensorManager, який інкапсулює роботу з Android Sensor API та надає потік даних про кроки у вигляді Flow. Це дозволяє абстрагуватися від конкретного механізму роботи сенсора та використовувати отримані дані у будь-якому компоненті системи:

```
class StepSensorManager(private val context: Context) {

    private val sensorManager =
        context.getSystemService(Context.SENSOR_SERVICE) as
SensorManager
    private val stepSensor =
        sensorManager.getDefaultSensor(Sensor.TYPE_STEP_COUNTER)

    fun getStepUpdates(): Flow<Int> = callbackFlow {
        val listener = object : SensorEventListener {
```

```
            override fun onSensorChanged(event: SensorEvent) {
                trySend(event.values[0].toInt())
            }
            override fun onAccuracyChanged(sensor: Sensor?, accuracy: Int) {}
        }
        sensorManager.registerListener(listener, stepSensor,
SensorManager.SENSOR_DELAY_UI)
        awaitClose { sensorManager.unregisterListener(listener) }
    }
}
```

Отримані від сенсора значення використовуються репозиторієм для формування добової статистики активності, яка зберігається у локальній базі даних Room. Таким чином реалізується принцип єдиного джерела правди, за якого історія активності зберігається централізовано, а всі інші компоненти системи отримують доступ до неї через репозиторій. Репозиторій виступає проміжною ланкою між доменним шаром і конкретними джерелами даних, що відповідає вимогам Clean Architecture.

Шар представлення (Presentation Layer) побудований відповідно до шаблону MVVM і включає ViewModel та екранні компоненти на Jetpack Compose. ViewModel виконує роль посередника між UI та доменною логікою, отримуючи оброблені дані з use case і транслюючи їх у стан інтерфейсу. У коді це реалізовано через використання StateFlow, який забезпечує реактивне оновлення UI при зміні даних:

```
class DashboardViewModel(
    private val getDashboardDataUseCase: GetDashboardDataUseCase
) : ViewModel() {

    val uiState: StateFlow<DashboardData> =
        getDashboardDataUseCase()
```

.stateIn(viewModelScope, SharingStarted.WhileSubscribed(), DashboardData())

}

Завдяки такій організації архітектури інтерфейс користувача не містить бізнес-логіки й не виконує обчислень, а лише відображає поточний стан, сформований ViewModel. Це відповідає принципам MVVM і сприяє підвищенню читабельності коду та зручності супроводу.

Таким чином, проєктування архітектури мобільного застосунку Krok на основі Clean Architecture та MVVM забезпечує чітку структуризацію програмного коду, прозору реалізацію алгоритмів моніторингу та аналізу фізичної активності, а також логічний зв’язок між сенсорними даними, математичними моделями й інтерфейсом користувача. Використання реальних компонентів коду в архітектурному проєктуванні підтверджує практичну реалізацію обраних теоретичних підходів і підсилює наукову обґрунтованість дипломної роботи.

Побудуємо далі діаграми класів та діаграму прецедентів для розробленого мобільного застосунку.

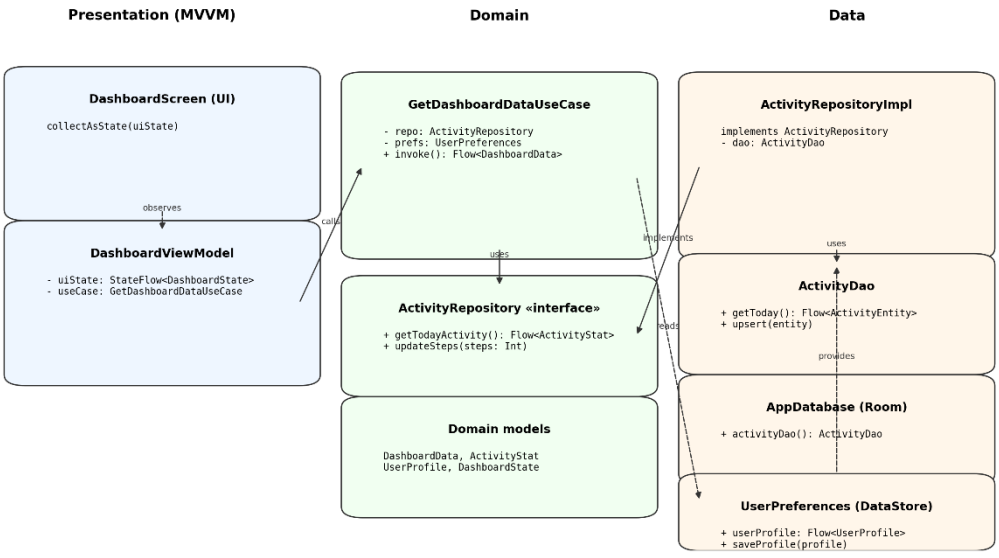

Рисунок 3. 2. Дагарама класів застосунку

Подана UML-діаграма класів відображає логічну структуру програмного застосунку для моніторингу фізичної активності та формування персональних рекомендацій, реалізованого на основі принципів Clean Architecture та архітектурного шаблону MVVM. Діаграма демонструє поділ системи на три основні рівні - рівень представлення, доменний рівень і рівень доступу до даних, а також чітко окреслює напрямки взаємодії між класами без перехресних залежностей.

На рівні представлення центральним елементом є клас DashboardViewModel, який відповідає за керування станом головного екрана застосунку. Він зберігає агрегований стан у вигляді реактивного потоку StateFlow<DashboardState> та ініціює отримання даних шляхом виклику доменного сценарію використання. Клас DashboardScreen виконує функцію інтерфейсу користувача та підписується на зміни стану ViewModel, реалізуючи реактивну модель взаємодії, за якої інтерфейс автоматично оновлюється при зміні даних без прямого доступу до бізнес-логіки.

Доменний рівень представлено класом GetDashboardDataUseCase, який інкапсулює основні алгоритми обробки даних фізичної активності. Цей клас отримує дані через абстракцію ActivityRepository, а також використовує налаштування профілю користувача, що зберігаються в UserPreferences. Саме на цьому рівні реалізуються математичні моделі розрахунку кількості витрачених калорій, індексу маси тіла, водного балансу та прогресу досягнення цільових показників. Використання доменних моделей DashboardData, ActivityStat, UserProfile і DashboardState забезпечує незалежність бізнес-логіки від конкретних технологій збереження даних та представлення інтерфейсу.

Рівень доступу до даних включає конкретні реалізації механізмів збереження та отримання інформації. Інтерфейс ActivityRepository реалізується класом ActivityRepositoryImpl, який взаємодіє з локальною базою даних через ActivityDao. База даних AppDatabase, побудована на основі бібліотеки Room, виступає контейнером для збереження історії фізичної активності користувача

та надає доступ до відповідних об’єктів доступу до даних. Клас UserPreferences, реалізований із використанням DataStore, відповідає за збереження та надання профільної інформації користувача у вигляді реактивного потоку, що дозволяє автоматично перераховувати результати при зміні параметрів профілю.

Структура зв’язків між класами на діаграмі побудована таким чином, що всі залежності спрямовані від рівня представлення до доменного рівня та далі до рівня даних, що повністю відповідає принципам Clean Architecture. Відсутність зворотних і перехресних залежностей спрощує супровід коду, підвищує його читабельність та полегшує тестування окремих компонентів. Представлена діаграма класів наочно демонструє архітектурну обґрунтованість розробленого застосунку та підтверджує коректність використання сучасних підходів до проєктування мобільних програмних систем.

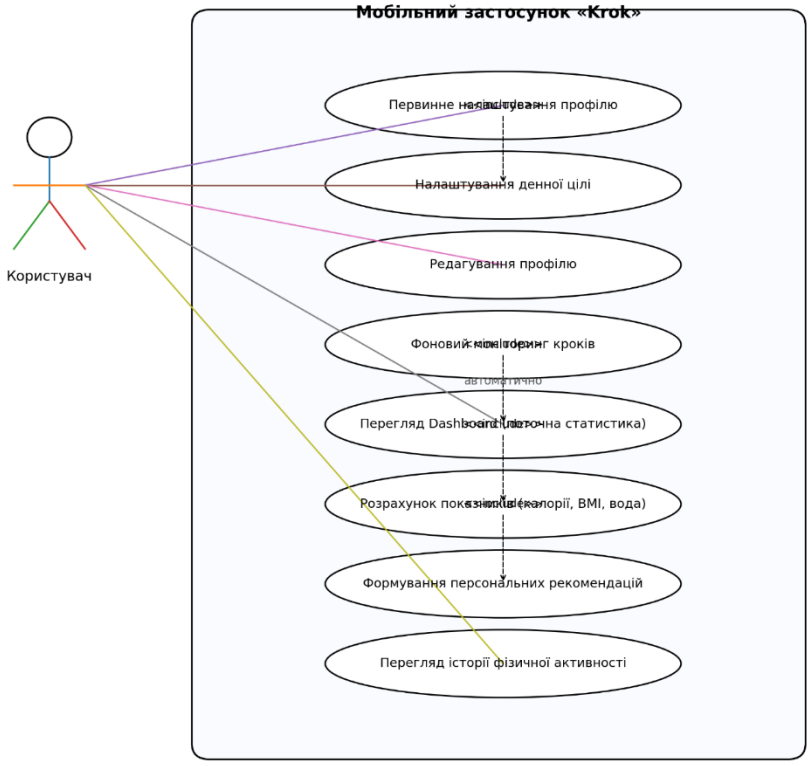

Рисунок 3.3. Діаграма прецедентів розробленого застосунку

Діаграма прецедентів відображає основні сценарії взаємодії користувача з мобільним застосунком «Krok» та межі відповідальності системи. Єдиним

зовнішнім актором на діаграмі є користувач, який ініціює налаштування параметрів, перегляд статистики та аналіз результатів фізичної активності. Межі системи охоплюють функції, що забезпечують моніторинг кроків і перетворення первинних сенсорних даних у прикладні показники та рекомендації.

Первинне налаштування профілю є стартовим сценарієм використання, під час якого користувач вводить персональні дані, необхідні для коректних розрахунків (зріст, вага, базові налаштування). Цей сценарій логічно пов’язаний із налаштуванням цілі (денної норми), оскільки встановлення або уточнення цільового значення є умовою для оцінювання прогресу та формування персоналізованих підказок.

Моніторинг кроків у застосунку реалізований як фоновий процес, який виконується автоматично без постійної участі користувача, забезпечуючи накопичення поточних даних активності. Отримані значення використовуються при перегляді головного екрана (Dashboard), де користувач спостерігає актуальну статистику. Сценарій перегляду Dashboard включає розрахунок похідних показників, зокрема витрачених калорій, дистанції, індексу маси тіла та рекомендованого водного балансу, оскільки ці величини формуються на основі кроків та параметрів профілю. Також у межах перегляду Dashboard включено отримання персональних рекомендацій, що забезпечує інтерпретацію результатів і підсилює мотиваційний ефект застосунку.

Окремо виділено сценарії редагування профілю та перегляду історії активності. Редагування профілю дозволяє актуалізувати персональні параметри, що безпосередньо впливають на математичні моделі розрахунку показників, а перегляд історії активності забезпечує аналіз динаміки у часовому вимірі. У сукупності представлені варіанти використання формують повний цикл роботи застосунку: від введення базових даних і налаштувань до автоматизованого збору сенсорної інформації, обчислення показників, відображення результатів і надання рекомендацій.

РОЗДІЛ 4. ПРАКТИЧНА ЧАСТИНА

4.1. Загальна структура програмного застосунку та опис використаних технологій розробки

Розроблений мобільний застосунок «Krok» призначений для моніторингу фізичної активності користувача, аналізу отриманих сенсорних даних та формування персоналізованих рекомендацій щодо оптимізації рухової активності. Архітектурна організація застосунку побудована з урахуванням принципів модульності, масштабованості та відокремлення відповідальностей, що забезпечує зручність супроводу програмного коду, можливість подальшого розширення функціоналу та повторного використання окремих компонентів.

Загальна структура застосунку реалізована відповідно до підходу Clean Architecture у поєднанні з архітектурним шаблоном MVVM (Model–View–ViewModel). Така комбінація дозволяє чітко розділити логіку представлення, бізнес-логіку та механізми доступу до даних, мінімізуючи залежності між окремими рівнями системи. Залежності у застосунку спрямовані від зовнішніх компонентів до внутрішніх, що відповідає фундаментальним принципам Clean Architecture [1-8].

Далі на рисунку представлено діаграму компонентів розробленого мобільного застосунку моніторинку фізичної активності «Krok».

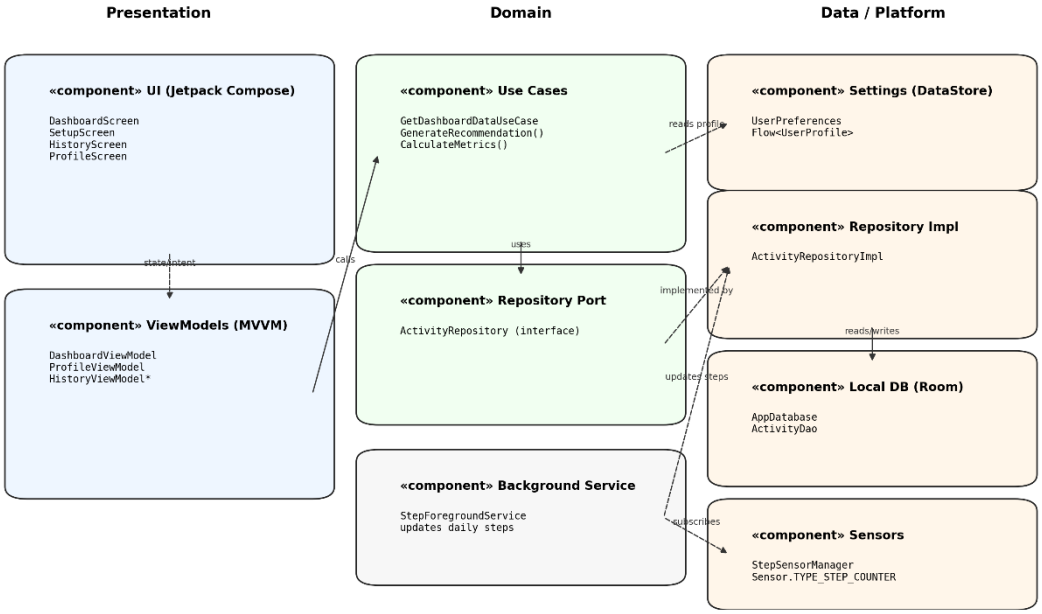

Рисунок 4.1. Діаграма компонентів мобільного застосунку Krok

Подана діаграма компонентів відображає високорівневу структурну організацію мобільного застосунку «Krok» та взаємодію його підсистем, сформовану на основі поєднання Clean Architecture і MVVM. Компоненти згруповано за трьома логічними рівнями: Presentation, Domain та Data/Platform. Така декомпозиція забезпечує розділення відповідальностей і спрямованість залежностей у бік бізнес-логіки, що є ключовим принципом Clean Architecture.

На рівні Presentation розміщено компонент UI (Jetpack Compose), який включає основні екрани застосунку (Dashboard, Setup, History, Profile) та виконує функцію відображення даних і приймання дій користувача. UI взаємодіє з компонентом ViewModels (MVVM) через реактивний механізм стану, тобто екрани підписуються на зміни стану та оновлюються автоматично при зміні даних. Компонент ViewModels виступає посередником між інтерфейсом та доменним рівнем, інкапсулюючи логіку підготовки стану екранів і викликаючи доменні сценарії.

На рівні Domain ключовим компонентом є Use Cases, у межах якого реалізовано сценарії отримання агрегованих даних для Dashboard, виконання

розрахунків показників (калорійність, індекс маси тіла, водний баланс) та генерація рекомендацій. Use Cases використовують Repository Port, що представлений інтерфейсом ActivityRepository. Застосування порту дозволяє доменному рівню не залежати від конкретних механізмів збереження та отримання даних, а працювати лише з абстракціями.

На рівні Data/Platform показано компоненти, що забезпечують доступ до джерел даних і ресурсів платформи Android. Компонент Settings (DataStore) відповідає за зберігання й надання профілю користувача (UserPreferences, Flow<UserProfile>), який читається доменними сценаріями для персоналізації обчислень. Компонент Repository Impl (ActivityRepositoryImpl) реалізує порт ActivityRepository та виконує читання/запис статистики активності через компонент Local DB (Room), що включає AppDatabase і ActivityDao. Окремо представлено компонент Sensors (StepSensorManager), який забезпечує доступ до апаратного сенсора кроків (Sensor.TYPE_STEP_COUNTER). Збір кроків у фоновому режимі виконує компонент Background Service (StepForegroundService), який підписується на потік даних сенсора та оновлює добову статистику через реалізацію репозиторію, забезпечуючи актуальність даних незалежно від відкритості інтерфейсу.

У сукупності діаграма демонструє, що інтерфейс користувача не звертається безпосередньо до бази даних або сенсорів, а отримує підготовлені дані через доменні сценарії та ViewModel. Це підтверджує коректність архітектурного проєктування та відповідність реалізації сучасним принципам побудови Android-застосунків із реактивною моделлю оновлення даних і чіткою ізоляцією бізнес-логіки.

На рівні представлення реалізовано інтерфейс користувача із використанням сучасного декларативного підходу Jetpack Compose. Основні екрани застосунку, зокрема головний екран зі статистикою (Dashboard), екран первинного налаштування, екран історії фізичної активності та екран редагування профілю, реалізовані у вигляді composable-функцій. Такий підхід

забезпечує реактивне оновлення інтерфейсу у відповідь на зміну стану даних без необхідності ручного керування життєвим циклом елементів інтерфейсу. Обмін даними між інтерфейсом та бізнес-логікою здійснюється через ViewModel, що відповідає вимогам шаблону MVVM.

Компонент ViewModel виконує роль посередника між рівнем представлення та доменним рівнем. У коді застосунку ключовим є клас DashboardViewModel, який зберігає стан головного екрана у вигляді реактивного потоку StateFlow<DashboardState>. ViewModel ініціює виклик доменних сценаріїв використання та обробляє результати їх виконання, перетворюючи їх у форму, зручну для відображення в інтерфейсі. Для асинхронної обробки даних у ViewModel використовуються Kotlin Coroutines та Flow, що дозволяє ефективно працювати з потоками даних і уникати блокування головного потоку виконання.

Доменний рівень застосунку реалізує бізнес-логіку обробки фізичної активності та формування рекомендацій. Центральним елементом цього рівня є сценарій використання GetDashboardDataUseCase, який інкапсулює алгоритми агрегації даних, розрахунку похідних показників та підготовки інформації для відображення на головному екрані. У межах доменного рівня використовуються доменні моделі, такі як ActivityStat, DashboardData, UserProfile та DashboardState, що забезпечує логічну цілісність даних і незалежність бізнес-логіки від конкретних технологій збереження або відображення.

Для доступу до даних доменний рівень взаємодіє з абстракцією репозиторію, представленою інтерфейсом ActivityRepository. Такий підхід дозволяє ізолювати бізнес-логіку від конкретних реалізацій джерел даних і спрощує тестування доменних сценаріїв. Реалізація репозиторію винесена на рівень доступу до даних.

Рівень доступу до даних забезпечує взаємодію з локальними сховищами та апаратними ресурсами мобільного пристрою. Для збереження історії

фізичної активності використовується локальна база даних Room, яка побудована на основі SQLite та забезпечує типобезпечний доступ до даних. У коді застосунку база даних представлена класом AppDatabase, а операції доступу до таблиць реалізовані через ActivityDao. Room дозволяє працювати з даними у вигляді реактивних потоків Flow, що забезпечує автоматичне оновлення даних у разі їх зміни.

Збереження налаштувань та профільних даних користувача реалізовано з використанням DataStore, який є сучасною альтернативою SharedPreferences. Компонент UserPreferences відповідає за зчитування та збереження таких параметрів, як зріст, вага, індивідуальні цілі та інші персональні налаштування. Використання DataStore у поєднанні з Flow дозволяє доменному рівню миттєво реагувати на зміну профільних даних і автоматично перераховувати результати.

Збір первинних даних фізичної активності здійснюється за допомогою апаратних сенсорів мобільного пристрою. Для цього використовується Sensor API платформи Android, зокрема сенсор типу Sensor.TYPE_STEP_COUNTER. Робота з сенсором інкапсульована в класі StepSensorManager, який надає потік даних із кількістю кроків у вигляді Flow. Це забезпечує уніфікований інтерфейс отримання сенсорних даних та їх інтеграцію з іншими компонентами системи.

Для забезпечення безперервного збору кроків навіть у фоновому режимі застосунку реалізовано foreground-сервіс StepForegroundService. Цей сервіс підписується на потік даних сенсора та періодично оновлює добову статистику через реалізацію репозиторію. Використання foreground-сервісу відповідає вимогам операційної системи Android щодо тривалих фонових операцій і гарантує стабільність роботи застосунку.

У таблиці 4.1. Наведено детальний опис функцій, які ми використали для нашого програмного продукту та технологічний стек представлено у таблиці 4.2.

Інструментарій розробки застосунку включає мову програмування Kotlin, середовище розробки Android Studio, бібліотеки Jetpack (Compose, ViewModel,

Room, DataStore), а також механізми асинхронного програмування Kotlin Coroutines і Flow.

Таблиця 4.1.

Функціональні вимоги програмного проєкту

№	Функція	Опис	Технічна реалізація
1	**Реєстрація / Налаштування профілю**	При першому запуску користувач вводить дані: Ім'я, Вагу, Зріст, Вік, Стать, Рівень активності та Денну ціль кроків.	Екран SetupScreen. Дані зберігаються в DataStore (UserPreferences).
2	**Підрахунок кроків**	Додаток зчитує дані з фізичного сенсора телефону в реальному часі.	Клас StepSensorManager використовує Sensor.TYPE_STEP_COUNTER.
3	**Головний екран (Dashboard)**	Відображає поточний прогрес, кільце кроків, спалені калорії, дистанцію та картки з порадами.	DashboardScreen з використанням ViewModel та StateFlow.
4	**Розрахунок калорій**	Автоматичний підрахунок: Кроки * 0.04.	Формула в GetDashboardDataUseCase.
5	**Розрахунок дистанції**	Переведення кроків у кілометри.	Формула в GetDashboardDataUseCase.
6	**Водний баланс**	Рекомендація норми води: Вага * 30 мл + Активність.	Логіка в GetDashboardDataUseCase.
7	**Індекс Маси Тіла (ІМТ)**	Розрахунок стану тіла (Норма/Надмірна вага) за формулою: Вага / Зріст².	Логіка в GetDashboardDataUseCase.
8	**Історія активності**	Перегляд списку результатів за минулі дні.	Екран HistoryScreen, дані беруться з Room Database (ActivityDao).
9	**Редагування профілю**	Можливість змінити вагу, зріст, ціль або видалити профіль. Ім'я та стать змінити неможливо.	Екран ProfileScreen з блокуванням полів readOnly.

Застосування зазначених технологій дозволило реалізувати сучасний, реактивний та масштабований програмний продукт, який відповідає

актуальним вимогам до мобільних фітнес-застосунків і може бути використаний як основа для подальшого розвитку та впровадження додаткових аналітичних і рекомендаційних модулів.

Таблиця 4.2.

Технічний стек мобільного застосунку

Категорія	**Технологія / Інструмент**	**Версія / Деталі**
Мова програмування	Kotlin	v2.0.0. Основна мова розробки Android-додатків.
Архітектура	MVVM + Clean Architecture	Розділення на шари: Presentation, Domain, Data. Забезпечує тестованість та гнучкість.
Користувацький інтерфейс	Jetpack Compose	Сучасний декларативний UI-фреймворк (Material3 Design).
База даних (Локальна)	Room Database	v2.6.1. Збереження історії активності (ActivityEntity).
Збереження налаштувань	Jetpack DataStore	v1.1.1 (Preferences). Збереження профілю користувача (UserProfile).
Асинхронність	Kotlin Coroutines & Flow	Обробка даних у фонових потоках (сенсори, БД).
Навігація	Navigation Compose	v2.8.0. Переходи між екранами (setup -> dashboard -> profile/history).
Робота з сенсорами	Android Sensor API	Використання апаратного шагоміра (SensorManager).
Впровадження залежностей	Manual DI	Ручне створення залежностей через KrokApp та фабрики ViewModel.
Мінімальна версія OS	Android 8.0 (Oreo)	API Level 26.
Цільова версія OS	Android 15 (Vanilla Ice Cream)	API Level 35.

Розробка мобільного застосунку «Krok» здійснювалася з використанням інтегрованого середовища Android Studio, яке забезпечує повний цикл створення, налагодження та тестування Android-застосунків. Android Studio надає засоби керування проєктною структурою, автоматичної генерації ресурсів, інтеграції з системою збірки Gradle та інструменти профілювання

продуктивності. Використання цього середовища дозволило ефективно організувати процес розробки, забезпечити контроль залежностей і виконувати налагодження як інтерфейсної частини, так і логіки обробки даних у режимі реального часу.

Як основну мову програмування обрано Kotlin, що є рекомендованою мовою для розробки Android-застосунків. Kotlin забезпечує підвищену безпеку типів, лаконічність синтаксису та глибоку інтеграцію з бібліотеками Jetpack. У коді застосунку Kotlin використовується для реалізації архітектурних компонентів, бізнес-логіки та взаємодії з апаратними сенсорами. Особливу роль відіграють корутини Kotlin, які дозволяють організувати асинхронне виконання операцій без блокування головного потоку та забезпечують зручну обробку тривалих обчислень і потоків даних.

Для реалізації інтерфейсу користувача застосовано Jetpack Compose, що є сучасним декларативним фреймворком побудови UI. Compose дозволяє описувати інтерфейс як функцію від стану, що повністю узгоджується з реактивною моделлю MVVM. Зміна стану у ViewModel автоматично призводить до перерисовування відповідних елементів інтерфейсу, що спрощує логіку керування UI та зменшує кількість помилок, пов’язаних із життєвим циклом компонентів. Такий підхід особливо ефективний для застосунків, які працюють із постійно оновлюваними даними фізичної активності.

Зберігання та обробка даних у застосунку реалізовані з використанням бібліотек Room і DataStore, що входять до складу Android Jetpack. Room забезпечує абстракцію над SQLite та дозволяє працювати з локальною базою даних у типобезпечний спосіб, використовуючи анотації та реактивні потоки Flow. DataStore застосовується для збереження налаштувань і профільних даних користувача та є більш надійним і масштабованим рішенням порівняно з традиційними SharedPreferences. Сукупне використання цих інструментів у поєднанні з Sensor API платформи Android створює технологічну основу для стабільної, реактивної та розширюваної системи моніторингу фізичної

активності.

4.2. Реалізація підсистеми збору даних фізичної активності

Підсистема збору даних фізичної активності є одним із ключових функціональних компонентів мобільного застосунку «Krok», оскільки забезпечує отримання первинних вимірювань рухової активності користувача, які надалі використовуються для обчислення похідних показників та формування персональних рекомендацій. При проєктуванні цієї підсистеми основну увагу приділено безперервності збору даних, коректній інтерпретації значень сенсора, енергоефективності та інтеграції з реактивною моделлю застосунку. У «Krok» для збору кроків використовується апаратний крокомір Android - сенсор Sensor.TYPE_STEP_COUNTER, що повертає накопичувальне значення кроків від моменту останнього перезавантаження пристрою.

Лістинг демонструє реалізацію доступу до сенсора в класі StepSensorManager, який інкапсулює взаємодію з SensorManager і перетворює події SensorEventListener на реактивний потік даних за допомогою callbackFlow. Таким чином, кожне оновлення значення сенсора перетворюється на елемент потоку, що дозволяє подальшим компонентам підписуватися на ці дані без блокування головного потоку.

```
package com.student.krok.data.sensors

import android.content.Context
import android.hardware.Sensor
import android.hardware.SensorEvent
import android.hardware.SensorEventListener
import android.hardware.SensorManager
import kotlinx.coroutines.channels.awaitClose
```

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

class StepSensorManager(private val context: Context) {

    private val sensorManager = context.getSystemService(Context.SENSOR_SERVICE) as SensorManager
    private val stepSensor: Sensor? = sensorManager.getDefaultSensor(Sensor.TYPE_STEP_COUNTER)

    fun getStepUpdates(): Flow<Int> = callbackFlow {
        if (stepSensor == null) {
            close(Exception("Step sensor not found on this device"))
            return@callbackFlow
        }

        val listener = object : SensorEventListener {
            override fun onSensorChanged(event: SensorEvent?) {
                event?.let {
                    // TYPE_STEP_COUNTER повертає float, але це ціле число
                    val stepsSinceReboot = it.values[0].toInt()
                    // Відправляємо дані в потік
                    trySend(stepsSinceReboot)
                }
            }

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

```
            sensorManager.registerListener(listener, stepSensor,
SensorManager.SENSOR_DELAY_UI)

            awaitClose {
                sensorManager.unregisterListener(listener)
            }
        }
    }
```

Принципово важливо, що Sensor.TYPE_STEP_COUNTER повертає не «кроки за день», а сумарні кроки від перезапуску. Тому для отримання добового значення необхідно перетворювати сумарне значення на прирости (delta). Саме цей підхід реалізовано в foreground-сервісі: сервіс зберігає попереднє значення previousSensorValue і на кожному оновленні обчислює приріст delta. Якщо delta > 0, то нові кроки додаються до добового лічильника, який зберігається у базі даних.

Подальша інтерпретація добової статистики та обчислення похідних показників виконуються на доменному рівні. У застосунку ця логіка зосереджена у сценарії використання GetDashboardDataUseCase, який агрегує дані активності (ActivityStat) і профілю (UserProfile) через combine, після чого виконує розрахунки калорій, водного балансу, BMI, прогресу та формує рекомендацію. Лістинг демонструє відповідну реалізацію, де калорії обчислюються як (steps * 0.04).toInt(), водний баланс - як weight * 30 + (steps * 0.08).toInt(), BMI - як weight / (heightM * heightM), а прогрес нормалізується методом coerceIn(0f, 1f).

```
package com.student.krok.domain.usecase

import com.student.krok.data.local.UserPreferences
import com.student.krok.data.local.UserProfile
import com.student.krok.domain.model.ActivityStat
```

```
import com.student.krok.domain.repository.ActivityRepository
import kotlinx.coroutines.flow.Flow
import kotlinx.coroutines.flow.combine

class GetDashboardDataUseCase(
    private val repository: ActivityRepository,
    private val userPreferences: UserPreferences
) {
    operator fun invoke(): Flow<DashboardData> {
        return combine(
            repository.getTodayActivity(),
            userPreferences.userProfile
        ) { stats, profile ->
            calculateData(stats, profile)
        }
    }

    private fun calculateData(stats: ActivityStat, profile: UserProfile): DashboardData {
        // Калорії (базова формула)
        val calculatedCalories = (stats.steps * 0.04).toInt()
        val distanceKm = stats.distanceMeters / 1000

        // Вода
        val baseWater = profile.weight * 30
        val activityWater = (stats.steps * 0.08).toInt()
        val totalWater = baseWater + activityWater

        val progress = (stats.steps.toFloat() / profile.dailyGoal).coerceIn(0f, 1f)
```

```
// BMI: вага (кг) / (зріст (м))^2
val heightM = profile.height / 100.0
val bmi = if (heightM > 0) profile.weight / (heightM * heightM) else 0.0

val bmiCategory = when {
    bmi < 18.5 -> "Недостатня вага"
    bmi < 25.0 -> "Норма"
    bmi < 30.0 -> "Надмірна вага"
    else -> "Ожиріння"
}

val recommendation = generateSmartRecommendation(
    steps = stats.steps,
    goal = profile.dailyGoal,
    calories = calculatedCalories
)

return DashboardData(
    steps = stats.steps,
    calories = calculatedCalories,
    distanceKm = distanceKm,
    dailyGoal = profile.dailyGoal,
    progress = progress,
    waterIntakeMl = totalWater,
    recommendation = recommendation,
    bmi = bmi,
    bmiCategory = bmiCategory
)
```

```
    }
}
```

Лістинг подає завершальний етап ланцюжка збору даних: foreground-сервіс StepForegroundService підписується на getStepUpdates(), виконує перетворення сумарного значення totalStepsSinceReboot на приріст delta, а потім оновлює добову статистику в базі даних через репозиторій. Така реалізація забезпечує узгодженість між сенсорним вимірюванням і тим значенням «кроків за сьогодні», яке відображається користувачу.

```
class StepForegroundService : Service() {

    private val serviceScope = CoroutineScope(SupervisorJob() + Dispatchers.IO)
    private lateinit var repository: ActivityRepository
    private lateinit var stepSensorManager: com.student.krok.data.sensors.StepSensorManager

    // Змінні для підрахунку
    private var previousSensorValue: Int? = null

    override fun onCreate() {
        super.onCreate()
        // Отримуємо доступ до залежностей через Application клас (Manual DI)
        val app = application as KrokApp
        repository = app.repository
        stepSensorManager = app.stepSensorManager

        createNotificationChannel()
        startForeground(1, createNotification("Підрахунок кроків активний"))
```

```
        // Починаємо слухати кроки
        subscribeToSteps()
    }

    private fun subscribeToSteps() {
        stepSensorManager.getStepUpdates().onEach { totalStepsSinceReboot ->
            if (previousSensorValue == null) {
                // Перше значення ігноруємо (просто калібруємось), але запам'ятовуємо
                previousSensorValue = totalStepsSinceReboot
            } else {
                val delta = totalStepsSinceReboot - previousSensorValue!!

                // Якщо є нові кроки
                if (delta > 0) {
                    previousSensorValue = totalStepsSinceReboot
                    updateDatabaseSteps(delta)
                }
            }
        }.launchIn(serviceScope)
    }

    private suspend fun updateDatabaseSteps(delta: Int) {
        // Отримуємо актуальні дані з бази
        val todayStat = repository.getTodayActivity().first()
        val newTotal = todayStat.steps + delta

        // Зберігаємо оновлені дані
```

```
            repository.updateSteps(newTotal)

            // Оновлюємо сповіщення
            updateNotification("Кроків сьогодні: $newTotal")
        }
    }
```

Таким чином, підсистема збору даних у «Krok» реалізує повний технологічний ланцюжок: отримання сумарного значення кроків сенсора TYPE_STEP_COUNTER, перетворення його на добовий приріст у foreground-сервісі та збереження результату у сховищі даних. Використання callbackFlow та потокової обробки забезпечує реактивну інтеграцію з рештою застосунку, а застосування combine у доменному сценарії формує єдиний контур узгодженого перерахунку показників при зміні активності або параметрів профілю.

4.3. Реалізація модуля обробки даних та генерації персональних рекомендацій

Модуль обробки даних та генерації персональних рекомендацій є ключовим функціональним компонентом програмного застосунку «Krok», оскільки саме в його межах здійснюється перетворення первинних даних фізичної активності, отриманих із сенсорів мобільного пристрою, у комплекс узагальнених показників та формування текстових рекомендацій для користувача. Реалізація цього модуля виконана на доменному рівні архітектури застосунку, що забезпечує його ізоляцію від механізмів збереження даних і користувацького інтерфейсу та повністю відповідає принципам Clean Architecture.

Обробка даних реалізується у сценарії використання GetDashboardDataUseCase, який відповідає за агрегацію поточної статистики фізичної активності та профільних даних користувача. Сценарій використання отримує дані про активність через інтерфейс ActivityRepository у вигляді об’єкта ActivityStat, а дані профілю - з компонента UserPreferences у вигляді реактивного потоку userProfile. Для синхронізації цих двох джерел даних використовується оператор combine, що дозволяє автоматично перераховувати результати при зміні будь-якого з потоків.

Реалізація сценарію використання у коді має такий вигляд:

```
class GetDashboardDataUseCase(
    private val repository: ActivityRepository,
    private val userPreferences: UserPreferences
) {
    operator fun invoke(): Flow<DashboardData> {
        return combine(
            repository.getTodayActivity(),
            userPreferences.userProfile
        ) { stats, profile ->
            calculateData(stats, profile)
        }
    }
}
```

У наведеному фрагменті дані про фізичну активність (stats) та профіль користувача (profile) передаються до внутрішнього методу calculateData, який інкапсулює всю логіку обчислення показників та формування результатів для головного екрана застосунку.

Безпосередні математичні обчислення реалізовані у приватному методі calculateData. На цьому етапі здійснюється розрахунок витрачених калорій, дистанції, рекомендованого об’єму споживання води, індексу маси тіла,

категорії BMI, прогресу виконання денної цілі та формування персональної рекомендації. Код методу має такий вигляд:

```
private fun calculateData(stats: ActivityStat, profile: UserProfile): DashboardData {
    val calculatedCalories = (stats.steps * 0.04).toInt()
    val distanceKm = stats.distanceMeters / 1000

    val baseWater = profile.weight * 30
    val activityWater = (stats.steps * 0.08).toInt()
    val totalWater = baseWater + activityWater

    val progress = (stats.steps.toFloat() / profile.dailyGoal).coerceIn(0f, 1f)

    val heightM = profile.height / 100.0
    val bmi = if (heightM > 0) profile.weight / (heightM * heightM) else 0.0

    val bmiCategory = when {
       bmi < 18.5 -> "Недостатня вага"
       bmi < 25.0 -> "Норма"
       bmi < 30.0 -> "Надмірна вага"
       else -> "Ожиріння"
    }

    val recommendation = generateSmartRecommendation(
       steps = stats.steps,
       goal = profile.dailyGoal,
       calories = calculatedCalories
    )
```

```
        return DashboardData(
            steps = stats.steps,
            calories = calculatedCalories,
            distanceKm = distanceKm,
            dailyGoal = profile.dailyGoal,
            progress = progress,
            waterIntakeMl = totalWater,
            recommendation = recommendation,
            bmi = bmi,
            bmiCategory = bmiCategory
        )
    }
```

У цьому фрагменті реалізовано послідовну обробку даних фізичної активності. Кількість витрачених калорій визначається пропорційно кількості кроків, дистанція обчислюється на основі збереженого значення distanceMeters, а рекомендований об’єм споживання води формується як сума базової норми, що залежить від маси тіла, та додаткової складової, пов’язаної з фізичною активністю. Прогрес виконання денної цілі нормалізується до інтервалу від 0 до 1 за допомогою методу coerceIn, що забезпечує коректне відображення індикаторів прогресу в інтерфейсі.

Індекс маси тіла обчислюється на основі зросту та маси користувача, після чого здійснюється класифікація значення BMI за чотирма категоріями: недостатня вага, норма, надмірна вага та ожиріння. Отримані значення включаються до об’єкта DashboardData та використовуються для інформування користувача про поточний стан фізичної форми.

Формування персональних рекомендацій реалізовано в окремому методі generateSmartRecommendation, який аналізує поточну кількість кроків, денну ціль і витрачені калорії. Такий підхід дозволяє відокремити логіку інтерпретації результатів від числових обчислень і спрощує подальше розширення алгоритму

рекомендацій. Реалізація методу має такий вигляд:

```
private fun generateSmartRecommendation(steps: Int, goal: Int, calories: Int): String {
    val remainingSteps = goal - steps
    return when {
        steps == 0 ->
            "Час рухатись! Навіть коротка прогулянка спалить перші калорії."
        steps < goal * 0.2 ->
            "Гарний старт. Ви вже спалили $calories ккал."
        steps < goal * 0.5 ->
            "Майже половина! Пройдіться ще трохи."
        steps < goal * 0.8 ->
            "Ціль близько! Вам залишилось всього $remainingSteps кроків."
        steps >= goal ->
            "Фантастика! Ви виконали план."
        else ->
            "Рухаємось вперед!"
    }
}
```

Алгоритм рекомендацій базується на порівнянні поточної кількості кроків із заданою денною ціллю та формує мотиваційні повідомлення залежно від рівня досягнутого прогресу. Такий підхід забезпечує зрозумілий і дружній зворотний зв’язок для користувача та сприяє підвищенню мотивації до регулярної фізичної активності.

Результати роботи модуля обробки даних передаються на рівень представлення через DashboardViewModel, де зберігаються у вигляді реактивного стану та автоматично відображаються в composable-екранах. Завдяки використанню потоків Flow і реактивної моделі оновлення інтерфейсу дані на екрані актуалізуються в режимі реального часу без необхідності

додаткових дій з боку користувача.

Таким чином, реалізований у застосунку «Krok» модуль обробки даних та генерації персональних рекомендацій забезпечує повний цикл перетворення первинної інформації про фізичну активність у зрозумілі, структуровані показники та мотиваційні рекомендації. Використані архітектурні рішення і точна реалізація алгоритмів у коді гарантують узгодженість програмної реалізації з теоретичними засадами та сучасними вимогами до мобільних фітнес-застосунків.

4.4. Розробка користувацького інтерфейсу мобільного застосунку засобами Jetpack Compose

Користувацький інтерфейс мобільного застосунку «Krok» реалізовано із застосуванням Jetpack Compose, що забезпечує декларативний підхід до побудови UI та реактивну модель оновлення екранів у відповідь на зміну стану. У проєкті композиційні екрани відокремлені від бізнес-логіки та взаємодіють з нею через стан, який формує рівень MVVM. Таке рішення забезпечує узгодженість інтерфейсу з архітектурою застосунку, мінімізує побічні ефекти та підвищує керованість коду при розширенні функціоналу.

Центральним елементом організації інтерфейсу є навігаційний компонент AppNavigation(app: KrokApp), у якому визначено маршрути застосунку та умову вибору стартового екрана залежно від факту завершення первинного налаштування. У коді це реалізовано шляхом підписки на isSetupCompleted із userPreferences та вибору startRoute для NavHost:

```
@Composable
fun AppNavigation(app: KrokApp) {
    val navController = rememberNavController()
```

```
        val scope = rememberCoroutineScope()

        val isSetupCompleted by app.userPreferences.isSetupCompleted.collectAsState(initial = null)

        if (isSetupCompleted == null) return

        val startRoute = if (isSetupCompleted == true) "dashboard" else "setup"

        NavHost(navController = navController, startDestination = startRoute) {
            composable("setup") {
                SetupScreen(
                    onSave = { profile -> /* ... */ }
                )
            }

            composable("dashboard") {
                val dashboardUseCase = com.student.krok.domain.usecase.GetDashboardDataUseCase(
                    repository = app.repository,
                    userPreferences = app.userPreferences
                )
                val factory = DashboardViewModelFactory(
                    getDashboardDataUseCase = dashboardUseCase,
                )
                val viewModel: DashboardViewModel = viewModel(factory = factory)

                DashboardScreen(
                    viewModel = viewModel,
```

```
                onNavigateToProfile = { navController.navigate("profile") },
                onNavigateToHistory = { navController.navigate("history") }
            )
        }
        // ...
    }
}
```

Наведений фрагмент демонструє важливу для дипломного проєкту властивість: екран Dashboard не створює бізнес-логіку самостійно, а отримує її через GetDashboardDataUseCase та DashboardViewModelFactory, що відповідає підходу MVVM і дозволяє централізовано керувати залежностями. Також навігація реалізована через маршрути "setup", "dashboard", "profile", "history", що забезпечує логічну послідовність роботи користувача: первинне введення даних → перегляд статистики → перегляд історії та редагування профілю.

Головний екран застосунку реалізовано у DashboardScreen(...). Екран підписується на реактивний стан state із DashboardViewModel та використовує його для відображення показників активності. У коді це зафіксовано прямим використанням collectAsState():

```
@Composable
fun DashboardScreen(
    viewModel: DashboardViewModel,
    onNavigateToProfile: () -> Unit,
    onNavigateToHistory: () -> Unit
) {
    val state by viewModel.state.collectAsState()
    val context = LocalContext.current
    // ...
}
```

Окремою складовою UI-логіки Dashboard є коректна робота з дозволами

на розпізнавання фізичної активності та запуск фонової підсистеми збору кроків через foreground-сервіс. У вашому коді це реалізовано безпосередньо в DashboardScreen: визначено функцію запуску сервісу StepForegroundService, а також permissionLauncher для запиту дозволу та виконання запуску лише після його надання:

```
fun startStepService() {
    val serviceIntent = Intent(context, StepForegroundService::class.java)
    context.startForegroundService(serviceIntent)
}

val permissionLauncher: ManagedActivityResultLauncher<String, Boolean> =
    rememberLauncherForActivityResult(
        contract = ActivityResultContracts.RequestPermission(),
        onResult = { isGranted ->
            if (isGranted) {
                startStepService()
            }
        }
    )
```

Таким чином, інтерфейсний рівень забезпечує коректне ініціювання підсистеми збору даних, але не виконує сам збір; це відповідає загальному принципу розмежування відповідальностей: UI керує сценарієм взаємодії з користувачем і системними дозволами, тоді як обробка та збереження даних відбуваються у відповідних компонентах (сервіс, репозиторій, база даних).

Для наочного представлення прогресу виконання денної цілі в Dashboard застосовано окремий composable-компонент CircularStepCounter(steps, goal, progress), який відображає кільце прогресу на основі двох CircularProgressIndicator - фонового та активного. Це забезпечує візуально зрозумілу подачу інформації про прогрес:

```
@Composable
fun CircularStepCounter(steps: Int, goal: Int, progress: Float) {
    Box(contentAlignment = Alignment.Center, modifier = Modifier.size(250.dp)) {
        CircularProgressIndicator(
            progress = { 1f },
            modifier = Modifier.fillMaxSize(),
            color = MaterialTheme.colorScheme.onSurface.copy(alpha = 0.1f),
            strokeWidth = 20.dp,
            trackColor = Color.Transparent,
        )
        CircularProgressIndicator(
            progress = { progress },
            modifier = Modifier.fillMaxSize(),
            color = Color(0xFF66BB6A),
            strokeWidth = 20.dp,
            // ...
        )
        // ...
    }
}
```

Відображення числових метрик (калорійність, дистанція тощо) у вашому інтерфейсі побудовано на основі повторно використовуваного компонента InfoCard(...), що формує уніфікований стиль подання різнорідних показників. Компонент використовує піктограму, значення та підпис, а також параметризується кольором, що підвищує інформативність UI:

```
@Composable
fun InfoCard(icon: ImageVector, value: String, label: String, color: Color) {
    Column(horizontalAlignment = Alignment.CenterHorizontally) {
```

```
        Box(
            modifier = Modifier
                .size(60.dp)
                .clip(CircleShape)
                .background(color.copy(alpha = 0.2f)),
            contentAlignment = Alignment.Center
        ) {
            Icon(
                imageVector = icon,
                contentDescription = label,
                tint = color,
                modifier = Modifier.size(30.dp)
            )
        }
        Spacer(modifier = Modifier.height(8.dp))
        Text(text = value, style = MaterialTheme.typography /* ... */)
        // ...
    }
}
```

Логіка відображення персоналізованої поради винесена в окремий UI-компонент RecommendationCard, що дозволяє ізолювати представлення рекомендації від решти екрана та підкреслити її як завершальний результат аналітики:

```
@Composable
fun RecommendationCard(recommendation: String) {
    Card(
        modifier = Modifier.fillMaxWidth(),
        shape = RoundedCornerShape(16.dp),
        colors = CardDefaults.cardColors(containerColor = Color(0xFF2E7D32))
```

```
    ) {
        Column(modifier = Modifier.padding(16.dp)) {
            Text("Рекомендація", /* ... */)
            Text(recommendation, /* ... */)
        }
    }
}
```

Екран первинного налаштування SetupScreen реалізує форму введення профільних даних та керує локальним станом полів введення засобами Compose (remember, mutableStateOf, mutableFloatStateOf). У вашому коді зафіксовано, що на етапі налаштування збираються параметри, необхідні для подальшої персоналізації обчислень і рекомендацій (зріст, вага, вік, стать, рівень активності, денна ціль):

```
@Composable
fun SetupScreen(
    onSave: (UserProfile) -> Unit,
) {
    var name by remember { mutableStateOf("") }
    var weight by remember { mutableStateOf("70") }
    var height by remember { mutableStateOf("170") }
    var age by remember { mutableStateOf("25") }
    var isMale by remember { mutableStateOf(true) }
    var activityLevel by remember { mutableFloatStateOf(1f) }
    var dailyGoal by remember { mutableStateOf("2000") }

    val activityDescription = when (activityLevel.toInt()) {
        1 -> "Сидячий спосіб життя (мало руху)"
        2 -> "Легка активність (прогулянки)"
        3 -> "Помірна активність (спорт 1-2 рази)"
```

```
            // ...
            else -> "Екстремальна активність (щоденні тренування)"
        }

        Scaffold { padding ->
            Column(
                modifier = Modifier
                    .padding(padding)
                    .padding(16.dp)
                    .fillMaxSize(),
                horizontalAlignment = Alignment.CenterHorizontally,
                verticalArrangement = Arrangement.Center
            ) {
                Text("Давайте знайомитись!", style = MaterialTheme.typography.headlineMedium)
                // ...
            }
        }
    }
```

Екран профілю ProfileScreen забезпечує редагування частини параметрів профілю, використовуючи початкові значення з UserProfile та локальні стани Compose для полів, що редагуються. При цьому ім’я та стать у вашій реалізації залишаються сталими для UI, що узгоджується з коментарями в коді та обраною логікою взаємодії:

```
@Composable
fun ProfileScreen(
    userProfile: UserProfile,
    onSave: (UserProfile) -> Unit,
    onBack: () -> Unit,
```

```
    onDeleteProfile: () -> Unit
) {
    val name = userProfile.name
    val isMale = userProfile.isMale

    var weight by remember { mutableStateOf(userProfile.weight.toString()) }
    var height by remember { mutableStateOf(userProfile.height.toString()) }
    var age by remember { mutableStateOf(userProfile.age.toString()) }
    // ...
    Scaffold(
        topBar = {
            TopAppBar(
                title = { Text("Налаштування") },
                // ...
            )
        }
        // ...
    )
}
```

Для аналізу динаміки активності реалізовано екран історії HistoryScreen(history: List<ActivityStat>, onBack: () -> Unit), який використовує Scaffold із верхньою панеллю та список елементів (через LazyColumn/items) із винесеним компонентом HistoryItem(stat). Така композиція забезпечує читабельність коду, повторне використання UI-елементів і зручність масштабування (наприклад, додавання фільтрації або деталізації записів):

```
@Composable
fun HistoryScreen(
    history: List<ActivityStat>,
    onBack: () -> Unit
```

```
    ) {
        Scaffold(
            topBar = {
                TopAppBar(
                    title = { Text("Історія активності") },
                    navigationIcon = {
                        IconButton(onClick = onBack) {
                            Icon(Icons.AutoMirrored.Filled.ArrowBack, contentDescription
= "Назад")
                        }
                    }
                )
            }
        ) { padding ->
            LazyColumn(
                modifier = Modifier
                    .padding(padding)
                    .fillMaxSize()
                    .padding(16.dp),
                verticalArrangement = Arrangement.spacedBy(12.dp)
            ) {
                items(history) { stat ->
                    HistoryItem(stat)
                }
            }
        }
    }
```

Узагальнюючи, інтерфейс користувача в застосунку «Krok» реалізовано як сукупність composable-екранів і повторно використовуваних UI-

компонентів, які працюють у реактивному режимі на основі стану ViewModel та подій навігації. Ваш код демонструє системний підхід до організації UI: маршрутизація визначається в AppNavigation, головний сценарій роботи користувача реалізовано через DashboardScreen, а функціональні підсценарії - через SetupScreen, ProfileScreen та HistoryScreen. Візуальні компоненти (CircularStepCounter, InfoCard, RecommendationCard, а також картки для води й BMI) забезпечують структуроване та наочне представлення даних, що підвищує зручність сприйняття результатів моніторингу та персональних рекомендацій.

4.5. Збереження та обробка даних користувача

Підсистема збереження та обробки даних користувача у мобільному застосунку «Krok» призначена для надійного зберігання персональних параметрів профілю та результатів моніторингу фізичної активності, а також для забезпечення їх узгодженого використання в алгоритмах обчислення та формування рекомендацій. Реалізація цієї підсистеми виконана з дотриманням принципів Clean Architecture, що передбачає розділення відповідальностей між доменним рівнем, рівнем доступу до даних та рівнем представлення.

Персональні дані користувача, зокрема ім’я, стать, вік, зріст, маса тіла та денна ціль фізичної активності, зберігаються у компоненті UserPreferences. Цей компонент реалізує доступ до налаштувань користувача у вигляді реактивного потоку Flow<UserProfile>, що дозволяє автоматично реагувати на зміну будь-якого з параметрів профілю. Саме цей потік використовується у доменному сценарії GetDashboardDataUseCase, де він поєднується з потоком даних активності за допомогою оператора combine.

Збереження та оновлення статистики фізичної активності виконується через інтерфейс ActivityRepository, який інкапсулює доступ до локальної бази

даних. Реалізація репозиторію відповідає за читання поточних даних (getTodayActivity) та їх оновлення (updateSteps). Такий підхід дозволяє ізолювати доменну логіку від конкретного механізму збереження та спрощує супровід і тестування програмного коду.

Для зберігання історії фізичної активності використовується локальна база даних Room, у якій добова статистика представлена у вигляді сутності ActivityStat. Доступ до таблиці реалізовано через об’єкт ActivityDao, який надає методи отримання поточної активності та її оновлення. Отримання даних із бази здійснюється у вигляді Flow, що забезпечує реактивну модель доступу та автоматичне оновлення даних на рівні представлення у разі змін.

У foreground-сервісі StepForegroundService збереження даних користувача здійснюється шляхом поетапного оновлення добової кількості кроків. Оскільки сенсор Sensor.TYPE_STEP_COUNTER повертає накопичувальне значення кроків від моменту перезавантаження пристрою, у сервісі реалізовано диференційний підхід. Поточне значення сенсора порівнюється з попереднім (previousSensorValue), після чого обчислюється приріст delta, який додається до значення, збереженого у базі даних. Відповідний фрагмент коду має такий вигляд:

```
private suspend fun updateDatabaseSteps(delta: Int) {
    val todayStat = repository.getTodayActivity().first()
    val newTotal = todayStat.steps + delta
    repository.updateSteps(newTotal)
    updateNotification("Кроків сьогодні: $newTotal")
}
```

Такий механізм збереження дозволяє коректно формувати добову статистику незалежно від кількості оновлень сенсора та забезпечує сталість даних при переході між екранами або тимчасовому закритті інтерфейсу. Дані фізичної активності зберігаються локально на пристрої користувача та використовуються для побудови історії активності, що відображається на

відповідному екрані застосунку.

Обробка збережених даних виконується на доменному рівні у сценарії використання GetDashboardDataUseCase, де статистика активності та профіль користувача об’єднуються у єдиний реактивний контур. У результаті будь-яка зміна даних - як оновлення кількості кроків, так і редагування параметрів профілю - автоматично призводить до перерахунку похідних показників і оновлення стану користувацького інтерфейсу без необхідності додаткових дій з боку користувача.

Таким чином, підсистема збереження та обробки даних у застосунку «Krok» забезпечує надійне зберігання персональної інформації та результатів моніторингу фізичної активності, а також їх узгоджене використання в алгоритмах аналізу та рекомендацій. Використання локального сховища Room, компонента UserPreferences і реактивних потоків Flow дозволяє реалізувати стабільну, масштабовану та логічно цілісну модель роботи з даними користувача, що відповідає сучасним вимогам до мобільних фітнес-застосунків.

4.6. . Інструкція користувача щодо роботи з програмним продуктом

Мобільний застосунок «Krok» призначений для моніторингу фізичної активності користувача, аналізу отриманих даних та формування персональних рекомендацій щодо оптимізації рухової активності. Для коректної роботи застосунку необхідно, щоб мобільний пристрій підтримував операційну систему Android і мав вбудований апаратний крокомір, сумісний із сенсором типу Sensor.TYPE_STEP_COUNTER.

Після встановлення та першого запуску застосунку користувачу пропонується пройти етап первинного налаштування.

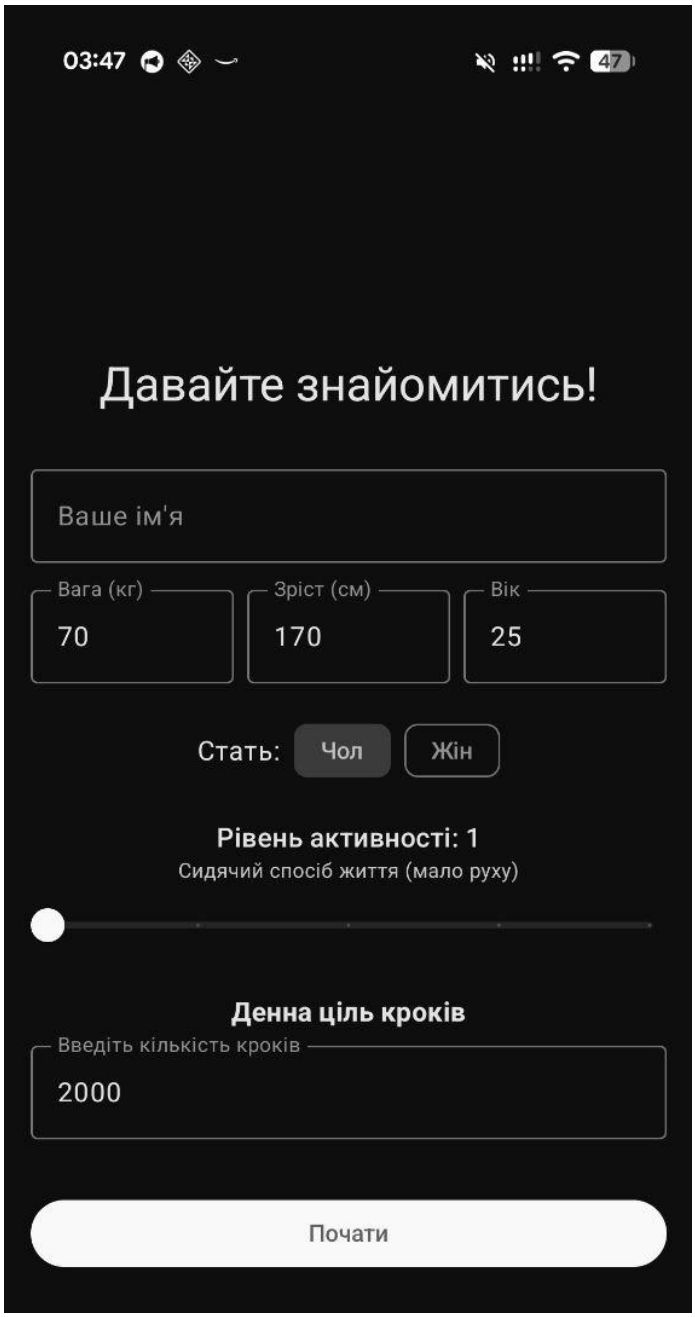

Рисунок 4. 2. Первинне налаштування застосунку

На цьому етапі необхідно ввести персональні дані, які використовуються для індивідуалізації розрахунків, зокрема ім’я, вік, зріст, масу тіла, стать, рівень фізичної активності та бажану денну ціль у кроках. Введення цих параметрів є обов’язковим, оскільки без них застосунок не може коректно виконувати обчислення показників та формувати персональні рекомендації. Після

підтвердження введених даних профіль користувача зберігається у локальному сховищі, а система автоматично переходить до головного екрана.

Для початку моніторингу фізичної активності застосунок запитує у користувача дозвіл на доступ до розпізнавання фізичної активності. У разі надання дозволу автоматично запускається фоновий сервіс збору даних, який забезпечує безперервний підрахунок кроків навіть за умови згортання застосунку або блокування екрана. Користувачу не потрібно виконувати додаткові дії для запуску підрахунку - процес відбувається автоматично після надання дозволу.

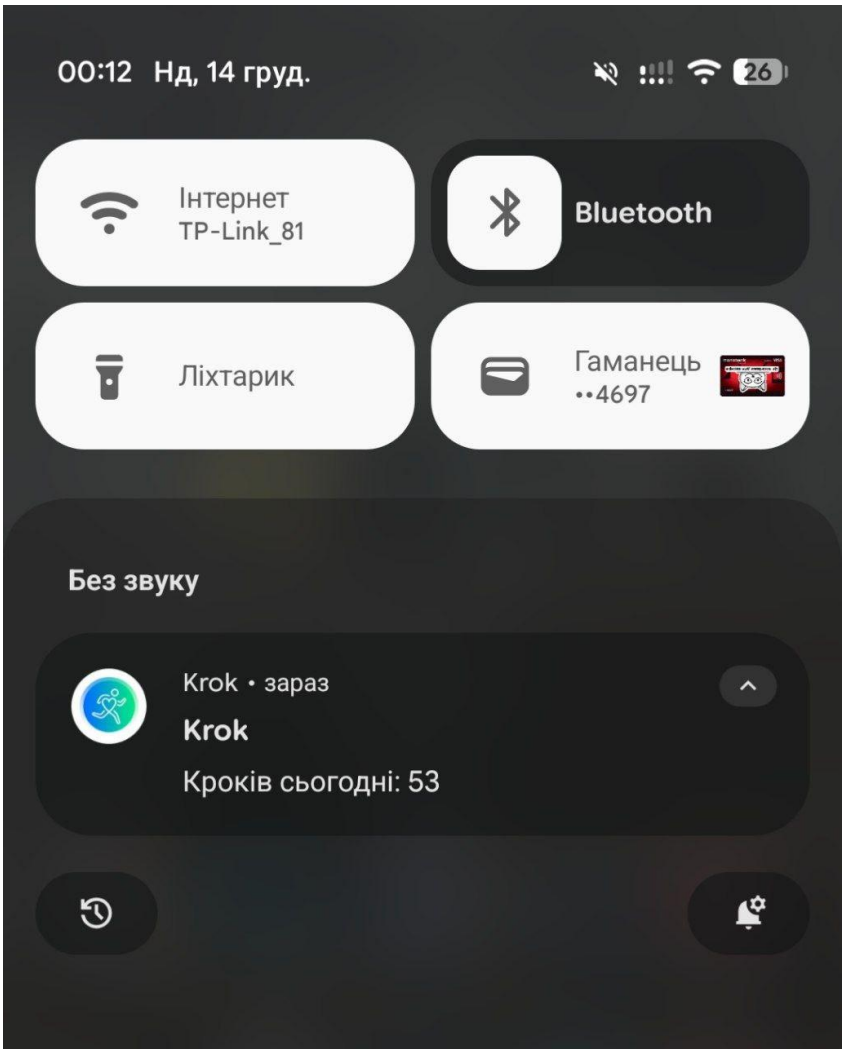

Рисунок 4.3. Робота застосунку у фоновому режимі

Головний екран застосунку відображає поточні показники фізичної активності за день. На ньому користувач може переглянути кількість пройдених кроків, прогрес виконання денної цілі, витрачені калорії, пройдену дистанцію, рекомендований об’єм споживання води та індекс маси тіла з відповідною категорією. Також на головному екрані відображається текстова персональна рекомендація, сформована на основі поточного рівня активності та заданих параметрів профілю.

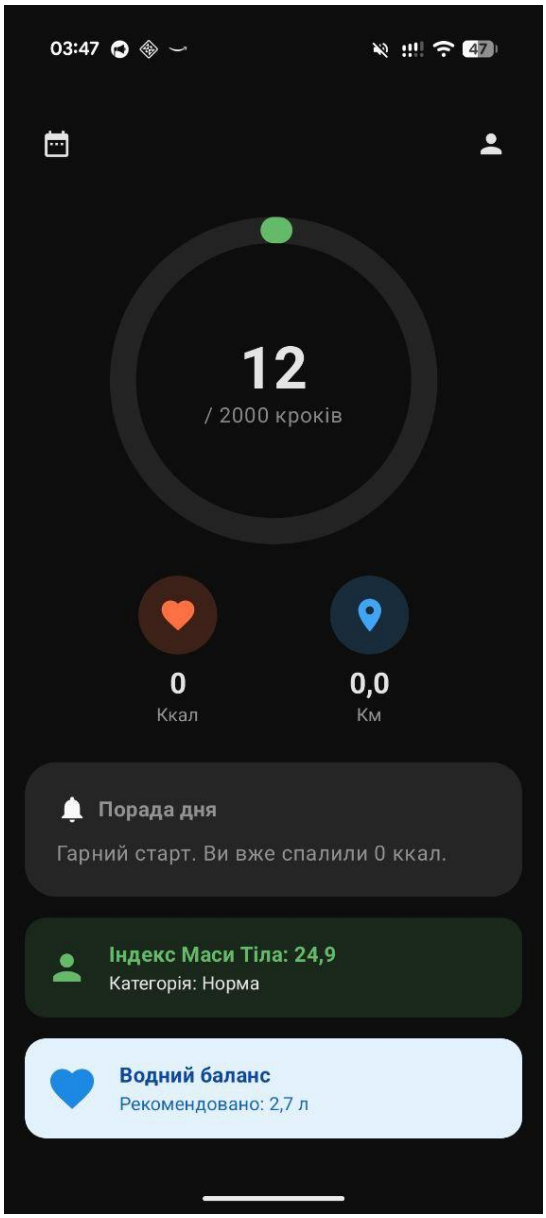

Рисунок 4.4. Процес відстеження прогресу

Для перегляду історії фізичної активності користувач може перейти до

відповідного розділу через елементи навігації застосунку. Екран історії містить список добових записів активності, де для кожного дня відображається кількість кроків та інші узагальнені показники. Це дозволяє користувачу аналізувати динаміку фізичної активності за попередні періоди та оцінювати ефективність досягнення поставлених цілей.

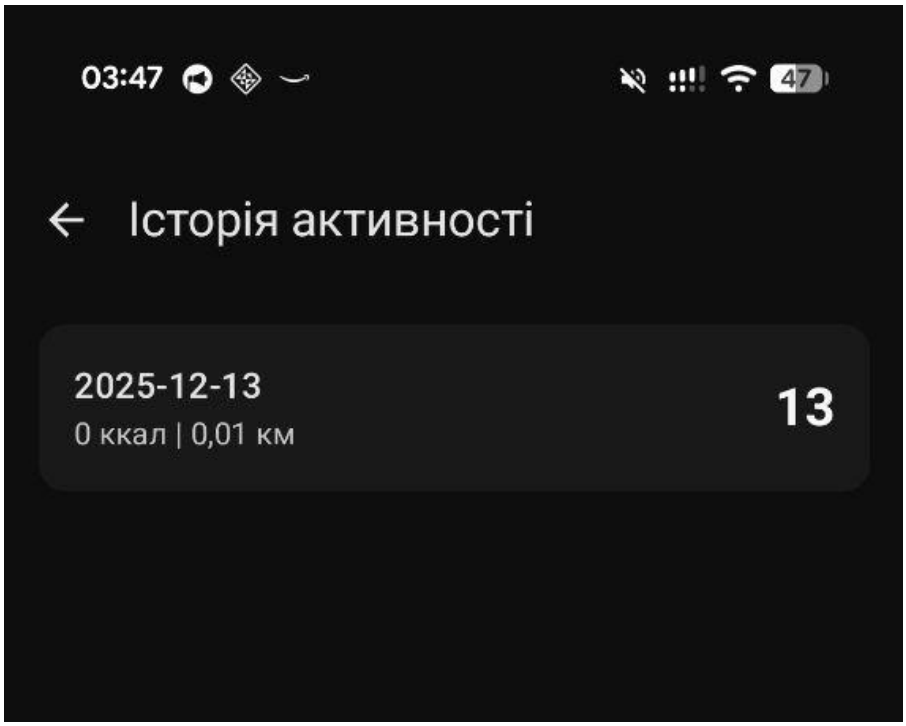

Рисунок 4.5. Відображення історії використання застосунку

У разі необхідності зміни персональних параметрів користувач може перейти до екрана налаштувань профілю. На цьому екрані дозволяється редагування таких параметрів, як вік, зріст, маса тіла та денна ціль у кроках. Після збереження змін оновлені значення автоматично враховуються в подальших обчисленнях, а всі похідні показники та рекомендації перераховуються без потреби перезапуску застосунку.

Застосунок працює в автономному режимі та зберігає всі дані локально на мобільному пристрої користувача. У разі закриття або перезапуску застосунку збережені дані профілю та статистика фізичної активності автоматично відновлюються. Для припинення роботи підсистеми збору даних користувач

може відкликати дозвіл на доступ до фізичної активності в системних налаштуваннях операційної системи Android.

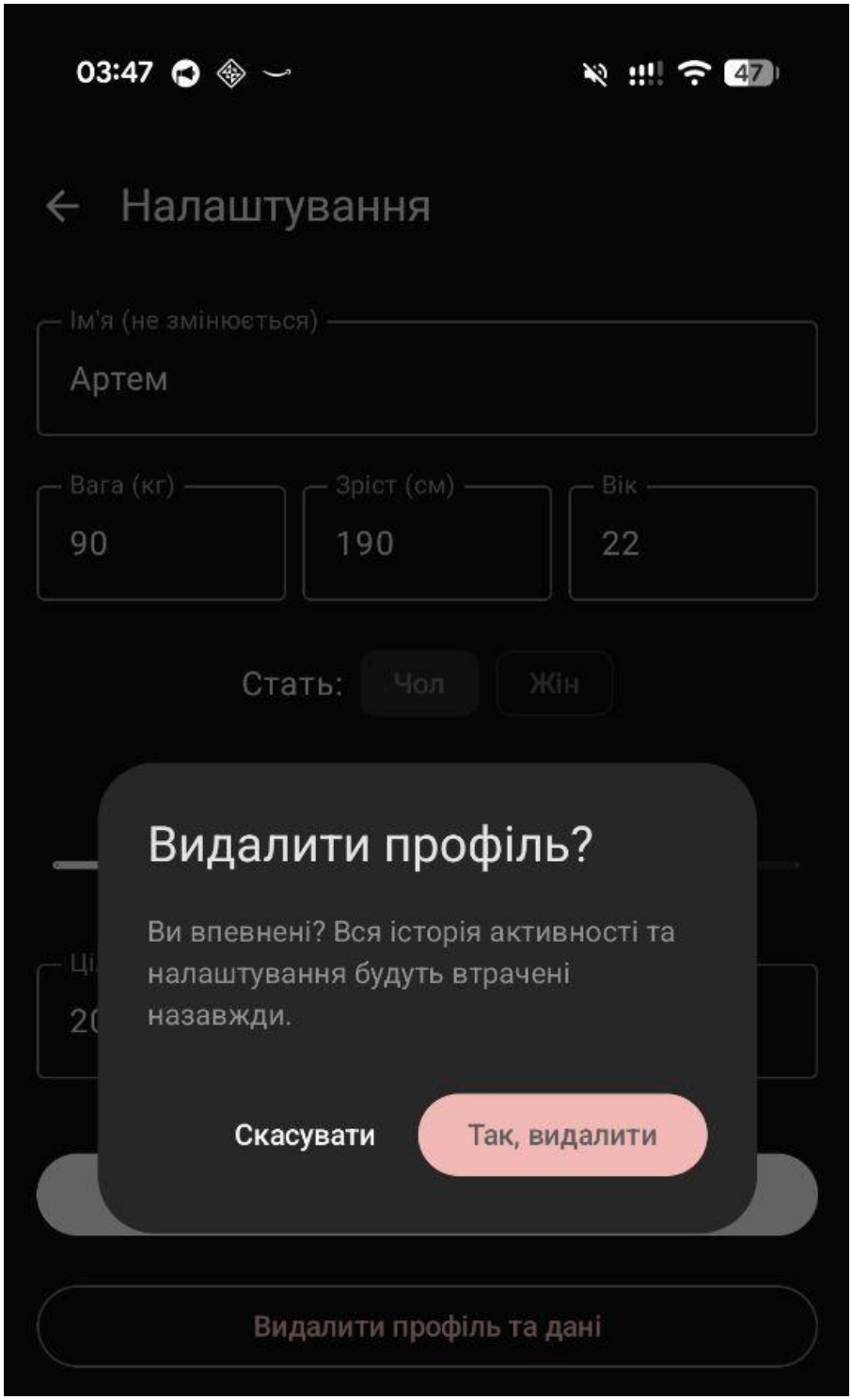

Рисунок 4.6. Робота із профілем користувача

Таким чином, робота з програмним продуктом «Krok» не потребує спеціальних технічних знань і зводиться до виконання інтуїтивно зрозумілих дій: первинного налаштування профілю, надання дозволів і перегляду результатів аналізу фізичної активності. Реалізований інтерфейс та автоматизовані механізми збору й обробки даних забезпечують зручність використання застосунку та його придатність для повсякденного застосування широким колом користувачів.

ВИСНОВКИ

У межах дипломної роботи було розроблено мобільний програмний застосунок «Krok», призначений для моніторингу фізичної активності користувача та формування персональних рекомендацій щодо оптимізації рухової активності на основі даних про кількість кроків і витрачених калорій. Запропоноване рішення поєднує сучасні підходи до архітектури мобільних застосунків, алгоритмічну обробку сенсорних даних і зручний користувацький інтерфейс, що забезпечує ефективне використання застосунку в повсякденному режимі.

У ході виконання роботи:

– проведено аналіз сучасних мобільних фітнес-застосунків та визначено їх функціональні можливості, переваги й недоліки з точки зору алгоритмів збору, обробки та інтерпретації даних фізичної активності;

– досліджено теоретичні основи моніторингу фізичної активності за допомогою мобільних сенсорів і математичні моделі розрахунку похідних показників, зокрема витрачених калорій, дистанції, індексу маси тіла, водного балансу та прогресу досягнення денної цілі;

– обґрунтовано вибір платформи Android, мови програмування Kotlin, архітектурних принципів Clean Architecture та шаблону MVVM, що забезпечують масштабованість, тестованість і чітке розділення відповідальностей між компонентами системи;

– спроєктовано архітектуру мобільного застосунку та побудовано UML-діаграми, які відображають структуру класів, варіанти використання, взаємодію компонентів і послідовність виконання основних сценаріїв;

– реалізовано підсистему збору даних фізичної активності з використанням апаратного сенсора Sensor.TYPE_STEP_COUNTER та foreground-сервісу, що забезпечує безперервний і енергоефективний підрахунок кроків;

– розроблено модуль обробки даних та генерації персональних рекомендацій, у якому сенсорні дані агрегуються з параметрами профілю користувача та перетворюються на інтерпретовані показники фізичної активності;

– створено користувацький інтерфейс засобами Jetpack Compose, який забезпечує наочне відображення результатів аналізу та інтуїтивну взаємодію з функціональними можливостями застосунку;

– проведено тестування програмного продукту, що підтвердило коректність обчислень, стабільність роботи та відповідність реалізованого функціоналу поставленим вимогам.

Розроблений мобільний застосунок дозволяє користувачу: автоматично здійснювати моніторинг фізичної активності без необхідності ручного введення даних; переглядати узагальнені показники активності за поточний день і аналізувати прогрес досягнення персональної мети; отримувати персоналізовані рекомендації щодо підвищення рівня рухової активності; зберігати історію фізичної активності та працювати із застосунком у автономному режимі без підключення до мережі Інтернет.

Практична цінність розробки полягає у поєднанні алгоритмічно обґрунтованої обробки сенсорних даних, сучасної архітектури програмного забезпечення та зручного інтерфейсу користувача. Застосунок може бути використаний як індивідуальний інструмент для підтримки здорового способу життя, а також як основа для подальших досліджень і розширення функціональних можливостей фітнес-систем.

Отримані результати створюють передумови для подальшого розвитку програмного продукту, зокрема шляхом інтеграції додаткових сенсорів, розширення аналітичних можливостей, впровадження адаптивних або інтелектуальних моделей персоналізації рекомендацій, а також синхронізації з хмарними сервісами та зовнішніми фітнес-платформами.

Таким чином, поставлену мету дипломної роботи досягнуто: розроблений мобільний застосунок «Krok» підтвердив ефективність використання платформи Android, мови Kotlin і принципів Clean Architecture для створення сучасних програмних рішень у сфері моніторингу та оптимізації фізичної активності користувачів.

По результатам роботи було опубліковано тези на науковій конференції.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Kotlin for Android. Режим доступу: URL: https://kotlinlang.org/docs/android-overview.html - Назва з екрану

2. Kotlin home page. Режим доступу: URL: https://kotlinlang.org/ - Назва з екрану

3. Meet Android Studio. Режим доступу: URL: https://developer.android.com/studio/intro - Назва з екрану

4. Model-View-ViewModel (MVVM). Режим доступу: URL: https://learn.microsoft.com/en-us/dotnet/architecture/maui/mvvm - Назва з екрану

5. MVVM Clean Architecture Pattern in Android with Use Cases. Режим доступу: URL:https://medium.com/@ami0275/mvvm-clean-architecture-pattern-in-android-with-use-cases-eff7edc2ef76 - Назва з екрану

6. Welcome to our tour of Kotlin. Режим доступу: URL: https://kotlinlang.org/docs/kotlin-tour-welcome.html - Назва з екрану

7. КОМП'ЮТЕРНІ НАУКИ ТА ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ (КНІТ-2025): матеріали науково-практичного семінару. Випуск 4 / за ред. О.В. Ольховської – Полтава: Кафедра КНІТ ПУЕТ, 2025. – 56 с. URL: http://www.matmodel.puet.edu.ua/files/knit-zb2025-4.pdf

8. Кошова О.П., Ольховська О.В., Чілікіна Т.В., Шуляр С. Особливості розробки програмного забезпечення для моделювання та дослідження бізнес-процесів з допомогою кореляційно-регресійного аналізу Вісник Кременчуцького національного університету імені Михайла Остроградського. Кременчук: КрНУ, 2024, № 3 С.86-92. DOI <https://doi.org/10.32782/1995-0519.2024.3.12>

9. Черненко О. О. Методичні рекомендації щодо виконання кваліфікаційної роботи студентів спеціальності 122 Комп'ютерні науки освітня програма «Комп'ютерні науки» ступеня магістра / С. В. Гаркуша, О. В. Ольховська, О. О. Черненко. – Полтава : ПУЕТ, 2023. – 68 с. Режим доступу:

URL: http://elib.puet.edu.ua/action.php?kt_path_info=ktcore.SecViewPlugin.actions.document&fDocumentId=824868 - Назва з екрану

10. How Google Fit Tracks Your Health: Features & Benefits, and What Is Changing . Режим доступу: URL: https://www.thryve.health/blog/google-fit-health-tracking-platform?utm_source=chatgpt.com - Назва з екрану

11. Samsung Health. Режим доступу: URL: https://www.samsung.com/us/apps/samsung-health/?utm_source=chatgpt.com - Назва з екрану

12. Tsyhanovska, Nataliia. (2025). ВИКОРИСТАННЯ МОБІЛЬНИХ ДОДАТКІВ ДЛЯ МОНІТОРИНГУ ФІЗИЧНОЇ АКТИВНОСТІ У ЗДОБУВАЧІВ ВИЩОЇ ОСВІТИ ПІД ЧАС ДИСТАНЦІЙНОГО НАВЧАННЯ. Режим доступу: URL: https://www.researchgate.net/publication/393958978 - Назва з екрану

13. Android Developers. Motion sensors overview. – Режим доступу: URL: https://developer.android.com/guide/topics/sensors/sensors_motion - Назва з екрану

14. Android Developers. SensorEvent and SensorManager documentation. – Режим доступу: URL: https://developer.android.com/reference/android/hardware/SensorEvent - Назва з екрану

15. Google Fit. How Google Fit estimates distance and calories. – Режим доступу: URL: https://developers.google.com/fit - Назва з екрану

16. Samsung Health. Physical activity tracking overview. – Режим доступу: URL: https://developer.samsung.com/health - Назва з екрану

17. Ainsworth, B. E., et al. Compendium of Physical Activities: an update of activity codes and MET intensities. Medicine & Science in Sports & Exercise, 2011. – Режим доступу: URL: https://journals.lww.com/acsm-msse/Fulltext/2011/08000/2011_Compendium_of_Physical_Activities.26.aspx - Назва з екрану

18. World Health Organization (WHO). Body Mass Index (BMI). – Режим

доступу: URL: https://www.who.int/data/gho/data/themes/topics/topic-details/GHO/body-mass-index - Назва з екрану

19. European Food Safety Authority (EFSA). Dietary reference values for water. – Режим доступу: URL: https://www.efsa.europa.eu/en/topics/topic/dietary-reference-values - Назва з екрану

ДОДАТОК А
Код програми

Файл: com/student/krok/MainActivity.kt

```
package com.student.krok

import android.content.Intent
import android.os.Build
import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.activity.enableEdgeToEdge
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Surface
import androidx.compose.ui.Modifier
import androidx.lifecycle.ViewModelProvider
import com.student.krok.data.service.StepForegroundService
import com.student.krok.presentation.dashboard.DashboardViewModel
import com.student.krok.presentation.dashboard.DashboardViewModelFactory
import com.student.krok.ui.theme.KrokTheme

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()

        val app = application as KrokApp

        // 2. Створюємо фабрику, передаючи залежності з app
        val factory = DashboardViewModelFactory(
            getDashboardDataUseCase = app.getActivityDataUseCase,
        )

        // 3. Ініціалізуємо ViewModel через фабрику
        val viewModel = ViewModelProvider(this, factory)[DashboardViewModel::class.java]

        setContent {
            KrokTheme { // Твоя тема
                Surface(
                    modifier = Modifier.fillMaxSize(),
                    color = MaterialTheme.colorScheme.background // Автоматичний колір фону
                ) {
                    AppNavigation(app = app)
                }
            }
        }
    }
}
```

Файл: com/student/krok/KrokApp.kt

```
package com.student.krok
```

```
import android.app.Application
import com.student.krok.data.local.AppDatabase
import com.student.krok.data.local.UserPreferences
import com.student.krok.data.repository.ActivityRepositoryImpl
import com.student.krok.data.sensors.StepSensorManager
import com.student.krok.domain.usecase.GetDashboardDataUseCase

class KrokApp: Application() {

    // Створюємо базу даних (лінива ініціалізація)
    private val database by lazy { AppDatabase.getDatabase(this) }

    // Створюємо репозиторій
    val repository by lazy { ActivityRepositoryImpl(database.activityDao()) }

    // Створюємо UseCase
    val getActivityDataUseCase by lazy { GetDashboardDataUseCase(repository, userPreferences) }

    // Створюємо SensorManager
    val stepSensorManager by lazy { StepSensorManager(this) }

    val userPreferences by lazy { UserPreferences(this) }

}
```

Файл: com/student/krok/AppNavigation.kt

```
package com.student.krok

import androidx.compose.runtime.Composable
import androidx.compose.runtime.collectAsState
import androidx.compose.runtime.getValue
import androidx.compose.runtime.rememberCoroutineScope
import androidx.lifecycle.viewmodel.compose.viewModel
import androidx.navigation.compose.NavHost
import androidx.navigation.compose.composable
import androidx.navigation.compose.rememberNavController
import com.student.krok.presentation.dashboard.DashboardScreen
import com.student.krok.presentation.dashboard.DashboardViewModel
import com.student.krok.presentation.dashboard.DashboardViewModelFactory
import com.student.krok.presentation.history.HistoryScreen // Імпорт
import com.student.krok.presentation.profile.ProfileScreen
import com.student.krok.presentation.setup.SetupScreen
import kotlinx.coroutines.launch

@Composable
fun AppNavigation(app: KrokApp) {
    val navController = rememberNavController()
    val scope = rememberCoroutineScope()

    val isSetupCompleted by app.userPreferences.isSetupCompleted.collectAsState(initial = null)

    if (isSetupCompleted == null) return
```

```
val startRoute = if (isSetupCompleted == true) "dashboard" else "setup"

NavHost(navController = navController, startDestination = startRoute) {

    composable("setup") {
        SetupScreen(
            onSave = { profile ->
                scope.launch {
                    app.userPreferences.saveUser(profile)
                    navController.navigate("dashboard") {
                        popUpTo("setup") { inclusive = true }
                    }
                }
            }
        )
    }

    composable("dashboard") {
        val dashboardUseCase = com.student.krok.domain.usecase.GetDashboardDataUseCase(
            repository = app.repository,
            userPreferences = app.userPreferences
        )
        val factory = DashboardViewModelFactory(
            getDashboardDataUseCase = dashboardUseCase,
        )
        val viewModel: DashboardViewModel = viewModel(factory = factory)

        DashboardScreen(
            viewModel = viewModel,
            onNavigateToProfile = { navController.navigate("profile") },
            onNavigateToHistory = { navController.navigate("history") } // <--- Перехід
        )
    }

    // ЕКРАН ПРОФІЛЮ (Редагування)
    composable("profile") {
        val userProfile by app.userPreferences.userProfile.collectAsState(initial = null)

        userProfile?.let { profile ->
            ProfileScreen(
                userProfile = profile,
                onBack = { navController.popBackStack() },
                onSave = { updatedProfile ->
                    scope.launch {
                        app.userPreferences.saveUser(updatedProfile) // Зберігаємо нові дані
                        navController.popBackStack() // Повертаємось назад
                    }
                },
                onDeleteProfile = {
                    scope.launch {
                        app.userPreferences.clearData()
                        app.repository.clearHistory()
                        navController.navigate("setup") {
                            popUpTo("dashboard") { inclusive = true }
```

```
                    }
                }
            }
        )
    }
}

// ЕКРАН ІСТОРІЇ (Новий)
composable("history") {
    val history by app.repository.getAllHistory().collectAsState(initial = emptyList())

    HistoryScreen(
        history = history,
        onBack = { navController.popBackStack() }
    )
}
}
}
```

Файл: com/student/krok/data/service/StepForegroundService.kt

```
package com.student.krok.data.service

import android.app.Notification
import android.app.NotificationChannel
import android.app.NotificationManager
import android.app.PendingIntent
import android.app.Service
import android.content.Context
import android.content.Intent
import android.os.Build
import android.os.IBinder
import androidx.core.app.NotificationCompat
import com.student.krok.KrokApp
import com.student.krok.MainActivity
import com.student.krok.R
import com.student.krok.domain.repository.ActivityRepository
import kotlinx.coroutines.CoroutineScope
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.SupervisorJob
import kotlinx.coroutines.cancel
import kotlinx.coroutines.flow.first
import kotlinx.coroutines.flow.launchIn
import kotlinx.coroutines.flow.onEach
import kotlinx.coroutines.launch

class StepForegroundService : Service() {

    private val serviceScope = CoroutineScope(SupervisorJob() + Dispatchers.IO)
    private lateinit var repository: ActivityRepository
    private lateinit var stepSensorManager: com.student.krok.data.sensors.StepSensorManager

    // Змінні для підрахунку
    private var previousSensorValue: Int? = null
```

```
override fun onCreate() {
    super.onCreate()
    // Отримуємо доступ до залежностей через Application клас (Manual DI)
    val app = application as KrokApp
    repository = app.repository
    stepSensorManager = app.stepSensorManager

    createNotificationChannel()
    startForeground(1, createNotification("Підрахунок кроків активний"))

    // Починаємо слухати кроки
    subscribeToSteps()
}

private fun subscribeToSteps() {
    stepSensorManager.getStepUpdates().onEach { totalStepsSinceReboot ->
        if (previousSensorValue == null) {
            // Перше значення ігноруємо (просто калібруємось), але запам'ятовуємо
            previousSensorValue = totalStepsSinceReboot
        } else {
            val delta = totalStepsSinceReboot - previousSensorValue!!

            // Якщо є нові кроки
            if (delta > 0) {
                previousSensorValue = totalStepsSinceReboot

                // Отримуємо поточні кроки за сьогодні з БД
                // Використовуємо runBlocking або launch, бо ми в сервісі
                updateDatabaseSteps(delta)
            }
        }
    }.launchIn(serviceScope)
}

private suspend fun updateDatabaseSteps(delta: Int) {
    // Отримуємо актуальні дані з бази
    val todayStat = repository.getTodayActivity().first()
    val newTotal = todayStat.steps + delta

    // Зберігаємо оновлені дані
    repository.updateSteps(newTotal)

    // Оновлюємо сповіщення
    updateNotification("Кроків сьогодні: $newTotal")
}

// --- Робота зі сповіщеннями (Обов'язково для Foreground Service) ---

private fun createNotificationChannel() {
    val channel = NotificationChannel(
        "step_channel",
        "Step Counter Service",
        NotificationManager.IMPORTANCE_LOW // Low, щоб не пікало постійно
    )
```

```
        val manager = getSystemService(NotificationManager::class.java)
        manager.createNotificationChannel(channel)
    }

    private fun createNotification(contentText: String): Notification {
        val intent = Intent(this, MainActivity::class.java).apply {
            flags = Intent.FLAG_ACTIVITY_NEW_TASK or Intent.FLAG_ACTIVITY_CLEAR_TASK
        }
        val pendingIntent = PendingIntent.getActivity(
            this, 0, intent,
            PendingIntent.FLAG_IMMUTABLE or PendingIntent.FLAG_UPDATE_CURRENT
        )

        return NotificationCompat.Builder(this, "step_channel")
            .setContentTitle("Krok")
            .setContentText(contentText)
            .setSmallIcon(R.drawable.ic_launcher_foreground) // Заміни на свою іконку кроків, якщо є
            .setContentIntent(pendingIntent)
            .setOngoing(true)
            .build()
    }

    private fun updateNotification(text: String) {
        val manager = getSystemService(Context.NOTIFICATION_SERVICE) as NotificationManager
        manager.notify(1, createNotification(text))
    }

    override fun onDestroy() {
        super.onDestroy()
        serviceScope.cancel() // Зупиняємо корутини, щоб не було витоку пам'яті
    }

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

Файл: com/student/krok/data/sensors/StepSensorManager.kt

```
package com.student.krok.data.sensors

import android.content.Context
import android.hardware.Sensor
import android.hardware.SensorEvent
import android.hardware.SensorEventListener
import android.hardware.SensorManager
import kotlinx.coroutines.channels.awaitClose
import kotlinx.coroutines.flow.Flow
import kotlinx.coroutines.flow.callbackFlow

class StepSensorManager(private val context: Context) {

    private val sensorManager = context.getSystemService(Context.SENSOR_SERVICE) as SensorManager
    private val stepSensor: Sensor? = sensorManager.getDefaultSensor(Sensor.TYPE_STEP_COUNTER)

    /**
     * Повертає потік даних з кроками.
```

```
     * Якщо сенсора немає, повертає помилку або нічого.
     */
    fun getStepUpdates(): Flow<Int> = callbackFlow {
        if (stepSensor == null) {
            close(Exception("Step sensor not found on this device"))
            return@callbackFlow
        }

        val listener = object : SensorEventListener {
            override fun onSensorChanged(event: SensorEvent?) {
                event?.let {
                    // TYPE_STEP_COUNTER повертає float, але це ціле число
                    val stepsSinceReboot = it.values[0].toInt()
                    // Відправляємо дані в потік
                    trySend(stepsSinceReboot)
                }
            }

            override fun onAccuracyChanged(sensor: Sensor?, accuracy: Int) {
                // Нас не цікавить зміна точності зараз
            }
        }

        // Реєструємо слухача
        sensorManager.registerListener(listener, stepSensor, SensorManager.SENSOR_DELAY_UI)

        // Цей блок виконується, коли потік закривається (наприклад, ми пішли з екрану)
        awaitClose {
            sensorManager.unregisterListener(listener)
        }
    }
}
```

Файл: com/student/krok/data/local/UserPreferences.kt

```
package com.student.krok.data.local

import android.content.Context
import androidx.datastore.preferences.core.*
import androidx.datastore.preferences.preferencesDataStore
import kotlinx.coroutines.flow.Flow
import kotlinx.coroutines.flow.map

private val Context.dataStore by preferencesDataStore(name = "user_prefs")

data class UserProfile(
    val name: String = "",
    val weight: Int = 70,
    val height: Int = 170, // <--- НОВЕ: Зріст у см (за замовчуванням 170)
    val age: Int = 25,
    val isMale: Boolean = true,
    val activityLevel: Int = 1,
    val dailyGoal: Int = 10000
)
```

```
class UserPreferences(private val context: Context) {

    companion object {
        val NAME_KEY = stringPreferencesKey("name")
        val WEIGHT_KEY = intPreferencesKey("weight")
        val HEIGHT_KEY = intPreferencesKey("height") // <--- НОВИЙ КЛЮЧ
        val AGE_KEY = intPreferencesKey("age")
        val IS_MALE_KEY = booleanPreferencesKey("is_male")
        val ACTIVITY_LEVEL_KEY = intPreferencesKey("activity_level")
        val DAILY_GOAL_KEY = intPreferencesKey("daily_goal")
        val IS_SETUP_COMPLETED = booleanPreferencesKey("is_setup_completed")
    }

    val userProfile: Flow<UserProfile> = context.dataStore.data.map { prefs ->
        UserProfile(
            name = prefs[NAME_KEY] ?: "",
            weight = prefs[WEIGHT_KEY] ?: 70,
            height = prefs[HEIGHT_KEY] ?: 170, // <--- Читаємо зріст
            age = prefs[AGE_KEY] ?: 25,
            isMale = prefs[IS_MALE_KEY] ?: true,
            activityLevel = prefs[ACTIVITY_LEVEL_KEY] ?: 1,
            dailyGoal = prefs[DAILY_GOAL_KEY] ?: 10000
        )
    }

    val isSetupCompleted: Flow<Boolean> = context.dataStore.data.map { prefs ->
        prefs[IS_SETUP_COMPLETED] ?: false
    }

    suspend fun saveUser(profile: UserProfile) {
        context.dataStore.edit { prefs ->
            prefs[NAME_KEY] = profile.name
            prefs[WEIGHT_KEY] = profile.weight
            prefs[HEIGHT_KEY] = profile.height // <--- Зберігаємо зріст
            prefs[AGE_KEY] = profile.age
            prefs[IS_MALE_KEY] = profile.isMale
            prefs[ACTIVITY_LEVEL_KEY] = profile.activityLevel
            prefs[DAILY_GOAL_KEY] = profile.dailyGoal
            prefs[IS_SETUP_COMPLETED] = true
        }
    }

    suspend fun clearData() {
        context.dataStore.edit { it.clear() }
    }
}
```

Файл: com/student/krok/data/local/AppDatabase.kt

```
package com.student.krok.data.local

import android.content.Context
import androidx.room.Database
import androidx.room.Room
import androidx.room.RoomDatabase
```

```
import com.student.krok.data.local.dao.ActivityDao
import com.student.krok.data.local.entity.ActivityEntity

// Описуємо, які таблиці є в базі і версію
@Database(entities = [ActivityEntity::class], version = 1, exportSchema = false)
abstract class AppDatabase : RoomDatabase() {

    // Room сам згенерує код для цього методу
    abstract fun activityDao(): ActivityDao

    companion object {
        // Singleton патерн, щоб не створювати базу двічі
        @Volatile
        private var INSTANCE: AppDatabase? = null

        fun getDatabase(context: Context): AppDatabase {
            return INSTANCE ?: synchronized(this) {
                val instance = Room.databaseBuilder(
                    context.applicationContext,
                    AppDatabase::class.java,
                    "optistep_database"
                ).build()
                INSTANCE = instance
                instance
            }
        }
    }
}
```

Файл: com/student/krok/data/local/entity/ActivityEntity.kt

```
package com.student.krok.data.local.entity

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

@Entity(tableName = "activity_table")
data class ActivityEntity(
    @PrimaryKey val date: String, // "YYYY-MM-DD"
    val steps: Int
)
```

Файл: com/student/krok/data/local/dao/ActivityDao.kt

```
package com.student.krok.data.local.dao

import androidx.room.Dao
import androidx.room.Insert
import androidx.room.OnConflictStrategy
import androidx.room.Query
import com.student.krok.data.local.entity.ActivityEntity
import kotlinx.coroutines.flow.Flow

@Dao
interface ActivityDao {
```

```
    @Query("SELECT * FROM activity_table WHERE date = :todayDate")
    fun getActivityByDate(todayDate: String): Flow<ActivityEntity?>

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insertActivity(activity: ActivityEntity)

    @Query("SELECT * FROM activity_table ORDER BY date DESC")
    fun getAllHistory(): Flow<List<ActivityEntity>>

    @Query("DELETE FROM activity_table")
    suspend fun clearHistory()
}
```

Файл: com/student/krok/domain/model/ActivityStat.kt

```
package com.student.krok.domain.model

data class ActivityStat(
    val date: String,      // Наприклад "2023-10-27"
    val steps: Int,
    val calories: Int,
    val distanceMeters: Float
)
```

Файл: com/student/krok/domain/repository/ActivityRepository.kt

```
package com.student.krok.domain.repository

import com.student.krok.domain.model.ActivityStat
import kotlinx.coroutines.flow.Flow

interface ActivityRepository {
    // Отримуємо потік даних (оновлюється в реальному часі)
    fun getTodayActivity(): Flow<ActivityStat>

    suspend fun clearHistory()

    fun getAllHistory(): Flow<List<ActivityStat>>

    // Оновити кількість кроків
    suspend fun updateSteps(steps: Int)

//    suspend fun insertFakeData()
}
```

Файл: com/student/krok/data/repository/ActivityRepositoryImpl.kt

```
package com.student.krok.data.repository
import com.student.krok.data.local.dao.ActivityDao
import com.student.krok.data.local.entity.ActivityEntity
import com.student.krok.domain.model.ActivityStat
import com.student.krok.domain.repository.ActivityRepository
import kotlinx.coroutines.flow.Flow
import kotlinx.coroutines.flow.map
import java.text.SimpleDateFormat
import java.util.Date
import java.util.Locale
```

```
class ActivityRepositoryImpl(
    private val dao: ActivityDao
) : ActivityRepository {

    private fun getTodayDate(): String {
        val sdf = SimpleDateFormat("yyyy-MM-dd", Locale.getDefault())
        return sdf.format(Date())
    }

    override fun getTodayActivity(): Flow<ActivityStat> {
        val today = getTodayDate()

        return dao.getActivityByDate(today).map { entity ->
            if (entity != null) {
                // Мапимо Entity (БД) в Model (Domain)
                ActivityStat(
                    date = entity.date,
                    steps = entity.steps,
                    calories = 0, // Порахуємо в UseCase
                    distanceMeters = entity.steps * 0.75f // Приблизно 0.75м крок
                )
            } else {
                // Якщо запису ще немає (ранок), повертаємо нулі
                ActivityStat(today, 0, 0, 0f)
            }
        }
    }

    override suspend fun updateSteps(steps: Int) {
        val today = getTodayDate()
        dao.insertActivity(ActivityEntity(date = today, steps = steps))
    }

    override fun getAllHistory(): Flow<List<ActivityStat>> {
        return dao.getAllHistory().map { entities ->
            // Мапимо список Entity (БД) у список Stat (UI)
            entities.map { entity ->
                ActivityStat(
                    date = entity.date,
                    steps = entity.steps,
                    // Тут дублюємо просту логіку підрахунку, або виносимо в окрему функцію util
                    calories = (entity.steps * 0.04).toInt(),
                    distanceMeters = entity.steps * 0.75f
                )
            }
        }
    }

//    override suspend fun insertFakeData() {
//        val stats = listOf(
//            ActivityEntity("2023-12-10", 5400),
//            ActivityEntity("2023-12-11", 8200),
//            ActivityEntity("2023-12-12", 10500),
```

```
//            ActivityEntity("2023-12-13", 3000), // Лінивий день
//            ActivityEntity("2023-12-14", 12000)
//        )
//        stats.forEach { dao.insertActivity(it) }
//    }

    // РЕАЛІЗАЦІЯ 2: Очистити історію
    override suspend fun clearHistory() {
        dao.clearHistory()
    }
}
```

Файл: com/student/krok/domain/usecase/GetDashboardDataUseCase.kt

```
package com.student.krok.domain.usecase

import com.student.krok.data.local.UserPreferences
import com.student.krok.data.local.UserProfile
import com.student.krok.domain.model.ActivityStat
import com.student.krok.domain.repository.ActivityRepository
import kotlinx.coroutines.flow.Flow
import kotlinx.coroutines.flow.combine

// 1. Оновлена модель даних
data class DashboardData(
    val steps: Int,
    val calories: Int,
    val distanceKm: Float,
    val dailyGoal: Int,
    val progress: Float,
    val waterIntakeMl: Int,
    val recommendation: String,
    val bmi: Double,        // <--- НОВЕ: Індекс
    val bmiCategory: String // <--- НОВЕ: Категорія
)

class GetDashboardDataUseCase(
    private val repository: ActivityRepository,
    private val userPreferences: UserPreferences
) {
    operator fun invoke(): Flow<DashboardData> {
        return combine(
            repository.getTodayActivity(),
            userPreferences.userProfile
        ) { stats, profile ->
            calculateData(stats, profile)
        }
    }

    private fun calculateData(stats: ActivityStat, profile: UserProfile): DashboardData {
        // Калорії (базова формула)
        val calculatedCalories = (stats.steps * 0.04).toInt()
        val distanceKm = stats.distanceMeters / 1000

        // Вода
```

```
        val baseWater = profile.weight * 30
        val activityWater = (stats.steps * 0.08).toInt()
        val totalWater = baseWater + activityWater

        val progress = (stats.steps.toFloat() / profile.dailyGoal).coerceIn(0f, 1f)

        // --- НОВЕ: РОЗРАХУНОК BMI ---
        // Формула: вага (кг) / (зріст (м))^2
        val heightM = profile.height / 100.0
        val bmi = if (heightM > 0) profile.weight / (heightM * heightM) else 0.0

        val bmiCategory = when {
            bmi < 18.5 -> "Недостатня вага"
            bmi < 25.0 -> "Норма"
            bmi < 30.0 -> "Надмірна вага"
            else -> "Ожиріння"
        }
        // ----------------------------

        val recommendation = generateSmartRecommendation(
            steps = stats.steps,
            goal = profile.dailyGoal,
            calories = calculatedCalories
        )

        return DashboardData(
            steps = stats.steps,
            calories = calculatedCalories,
            distanceKm = distanceKm,
            dailyGoal = profile.dailyGoal,
            progress = progress,
            waterIntakeMl = totalWater,
            recommendation = recommendation,
            bmi = bmi,            // <---
            bmiCategory = bmiCategory // <---
        )
    }

    private fun generateSmartRecommendation(steps: Int, goal: Int, calories: Int): String {
        val remainingSteps = goal - steps
        return when {
            steps == 0 -> "Час рухатись! Навіть коротка прогулянка спалить перші калорії."
            steps < goal * 0.2 -> "Гарний старт. Ви вже спалили $calories ккал."
            steps < goal * 0.5 -> "Майже половина! Пройдіться ще трохи."
            steps < goal * 0.8 -> "Ціль близько! Вам залишилось всього $remainingSteps кроків."
            steps >= goal -> "Фантастика! Ви виконали план."
            else -> "Рухаємось вперед!"
        }
    }
}
```

Файл: com/student/krok/presentation/dashboard/DashboardState.kt

```
package com.student.krok.presentation.dashboard
```

```
data class DashboardState(
    val steps: Int = 0,
    val calories: Int = 0,
    val distance: Float = 0f,
    val recommendation: String = "Завантаження...",
    val progress: Float = 0f,
    val dailyGoal: Int = 10000,
    val waterIntake: Int = 0,
    val bmi: Double = 0.0,        // <---
    val bmiCategory: String = ""  // <---
)
```

Файл: com/student/krok/presentation/dashboard/DashboardViewModel.kt

```
package com.student.krok.presentation.dashboard

import androidx.lifecycle.ViewModel
import androidx.lifecycle.ViewModelProvider
import androidx.lifecycle.viewModelScope
import com.student.krok.data.local.UserPreferences
import com.student.krok.data.sensors.StepSensorManager
import com.student.krok.domain.repository.ActivityRepository
import com.student.krok.domain.usecase.GetDashboardDataUseCase
// GetActivityDataUseCase нам тут більше не потрібен, бо ми використовуємо combine напряму з репозиторієм
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.flow.StateFlow
import kotlinx.coroutines.flow.asStateFlow
import kotlinx.coroutines.flow.combine
import kotlinx.coroutines.flow.launchIn
import kotlinx.coroutines.flow.onEach



class DashboardViewModel(
    private val getDashboardDataUseCase: GetDashboardDataUseCase, // <--- Наш новий UseCase
) : ViewModel() {

    private val _state = MutableStateFlow(DashboardState())
    val state: StateFlow<DashboardState> = _state.asStateFlow()

    init {
        subscribeToData()
    }

    private fun subscribeToData() {
        getDashboardDataUseCase().onEach { data ->
            _state.value = state.value.copy(
                steps = data.steps,
                calories = data.calories,
                distance = data.distanceKm,
                progress = data.progress,
                dailyGoal = data.dailyGoal,
                waterIntake = data.waterIntakeMl,
                recommendation = data.recommendation,
```

```
            bmi = data.bmi,                // <---
            bmiCategory = data.bmiCategory  // <---
          )
        }.launchIn(viewModelScope)
    }
}

// Оновлена фабрика
class DashboardViewModelFactory(
    private val getDashboardDataUseCase: GetDashboardDataUseCase,
) : ViewModelProvider.Factory {
    override fun <T : ViewModel> create(modelClass: Class<T>): T {
        if (modelClass.isAssignableFrom(DashboardViewModel::class.java)) {
            @Suppress("UNCHECKED_CAST")
            return DashboardViewModel(getDashboardDataUseCase) as T
        }
        throw IllegalArgumentException("Unknown ViewModel class")
    }
}

================================================================================
================================
Файл: com/student/krok/presentation/setup/SetupScreen.kt
================================================================================
================================
package com.student.krok.presentation.setup

import androidx.compose.foundation.layout.*
import androidx.compose.foundation.text.KeyboardOptions
import androidx.compose.material3.*
import androidx.compose.runtime.*
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.text.input.KeyboardType
import androidx.compose.ui.unit.dp
import com.student.krok.data.local.UserProfile

@Composable
fun SetupScreen(
    onSave: (UserProfile) -> Unit,
) {
    var name by remember { mutableStateOf("") }
    var weight by remember { mutableStateOf("70") }
    var height by remember { mutableStateOf("170") } // <--- НОВА ЗМІННА
    var age by remember { mutableStateOf("25") }
    var isMale by remember { mutableStateOf(true) }
    var activityLevel by remember { mutableFloatStateOf(1f) }
    var dailyGoal by remember { mutableStateOf("2000") }

    val activityDescription = when (activityLevel.toInt()) {
        1 -> "Сидячий спосіб життя (мало руху)"
        2 -> "Легка активність (прогулянки)"
        3 -> "Помірна активність (спорт 1-2 рази)"
```

```
    4 -> "Висока активність (спорт 3-5 разів)"
    else -> "Екстремальна активність (щоденні тренування)"
}

Scaffold { padding ->
    Column(
        modifier = Modifier
            .padding(padding)
            .padding(16.dp)
            .fillMaxSize(),
        horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.Center
    ) {
        Text("Давайте знайомитись!", style = MaterialTheme.typography.headlineMedium)

        Spacer(modifier = Modifier.height(24.dp))

        OutlinedTextField(
            value = name,
            onValueChange = { name = it },
            label = { Text("Ваше ім'я") },
            modifier = Modifier.fillMaxWidth()
        )

        Spacer(modifier = Modifier.height(8.dp))

        // Ряд: Вага | Зріст | Вік
        Row(Modifier.fillMaxWidth(), horizontalArrangement = Arrangement.spacedBy(8.dp)) {
            OutlinedTextField(
                value = weight,
                onValueChange = { if (it.all { char -> char.isDigit() }) weight = it },
                label = { Text("Вага (кг)") },
                keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Number),
                modifier = Modifier.weight(1f)
            )
            // --- НОВЕ ПОЛЕ ---
            OutlinedTextField(
                value = height,
                onValueChange = { if (it.all { char -> char.isDigit() }) height = it },
                label = { Text("Зріст (см)") },
                keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Number),
                modifier = Modifier.weight(1f)
            )
            // -----------------
            OutlinedTextField(
                value = age,
                onValueChange = { if (it.all { char -> char.isDigit() }) age = it },
                label = { Text("Вік") },
                keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Number),
                modifier = Modifier.weight(1f)
            )
        }

        Spacer(modifier = Modifier.height(16.dp))
```

```
        // Вибір статі
        Row(verticalAlignment = Alignment.CenterVertically) {
            Text("Стать:", style = MaterialTheme.typography.bodyLarge)
            Spacer(modifier = Modifier.width(8.dp))
            FilterChip(selected = isMale, onClick = { isMale = true }, label = { Text("Чол") })
            Spacer(modifier = Modifier.width(8.dp))
            FilterChip(selected = !isMale, onClick = { isMale = false }, label = { Text("Жін") })
        }

        Spacer(modifier = Modifier.height(16.dp))

        // Слайдер активності
        Text("Рівень активності: ${activityLevel.toInt()}", style = MaterialTheme.typography.titleMedium)
        Text(activityDescription, style = MaterialTheme.typography.bodySmall, color =
MaterialTheme.colorScheme.secondary)

        Slider(
            value = activityLevel,
            onValueChange = { activityLevel = it },
            valueRange = 1f..5f,
            steps = 3
        )

        Spacer(modifier = Modifier.height(16.dp))

        // --- НОВИЙ БЛОК: ЦІЛЬ ---
        Text(
            text = "Денна ціль кроків",
            style = MaterialTheme.typography.titleMedium,
            fontWeight = FontWeight.Bold
        )

        OutlinedTextField(
            value = dailyGoal,
            onValueChange = {
                // Дозволяємо вводити тільки цифри
                if (it.all { char -> char.isDigit() } && it.length <= 6) {
                    dailyGoal = it
                }
            },
            label = { Text("Введіть кількість кроків") },
            keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Number),
            modifier = Modifier.fillMaxWidth(),
            singleLine = true
        )

        Spacer(modifier = Modifier.height(32.dp))

//          Button(
//              onClick = onGenerateData, // Викликаємо генерацію
//              modifier = Modifier.fillMaxWidth(),
//              colors = ButtonDefaults.buttonColors(containerColor = MaterialTheme.colorScheme.secondary)
//          ) {
```

```
//             Text("Згенерувати тестову історію")
//         }
//         Spacer(modifier = Modifier.height(24.dp))

        Button(
            onClick = {
                if (name.isNotEmpty() && weight.isNotEmpty()) {
                    onSave(
                        UserProfile(
                            name,
                            weight.toIntOrNull() ?: 70,
                            height.toIntOrNull() ?: 170,
                            age.toIntOrNull() ?: 25,
                            isMale,
                            activityLevel.toInt(),
                            dailyGoal.toIntOrNull() ?: 2000
                        )
                    )
                }
            },
            modifier = Modifier.fillMaxWidth()
        ) {
            Text("Почати")
        }
    }
}


}
```

Файл: com/student/krok/presentation/dashboard/DashboardScreen.kt

```
package com.student.krok.presentation.dashboard

import android.Manifest
import android.annotation.SuppressLint
import android.content.Intent
import android.content.pm.PackageManager
import android.os.Build
import androidx.activity.compose.ManagedActivityResultLauncher
import androidx.activity.compose.rememberLauncherForActivityResult
import androidx.activity.result.contract.ActivityResultContracts
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.size
import androidx.compose.foundation.layout.width
import androidx.compose.foundation.rememberScrollState
```

```
import androidx.compose.foundation.shape.CircleShape
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.foundation.verticalScroll
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.Add
import androidx.compose.material.icons.filled.DateRange
import androidx.compose.material.icons.filled.Favorite
import androidx.compose.material.icons.filled.LocationOn
import androidx.compose.material.icons.filled.Notifications
import androidx.compose.material.icons.filled.Person
import androidx.compose.material3.Card
import androidx.compose.material3.CardDefaults
import androidx.compose.material3.CircularProgressIndicator
import androidx.compose.material3.Icon
import androidx.compose.material3.IconButton
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Scaffold
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.LaunchedEffect
import androidx.compose.runtime.collectAsState
import androidx.compose.runtime.getValue
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.clip
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.graphics.StrokeCap
import androidx.compose.ui.graphics.vector.ImageVector
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import androidx.core.content.ContextCompat
import com.student.krok.data.service.StepForegroundService

@SuppressLint("DefaultLocale")
@Composable
fun DashboardScreen(
    viewModel: DashboardViewModel,
    onNavigateToProfile: () -> Unit,
    onNavigateToHistory: () -> Unit
) {
    val state by viewModel.state.collectAsState()
    val context = LocalContext.current

    fun startStepService() {
        val serviceIntent = Intent(context, StepForegroundService::class.java)
        context.startForegroundService(serviceIntent)
    }

    // Лаунчер для запиту дозволу
    val permissionLauncher: ManagedActivityResultLauncher<String, Boolean> =
rememberLauncherForActivityResult(
        contract = ActivityResultContracts.RequestPermission(),
```

```
        onResult = { isGranted ->
            if (isGranted) {
                // Якщо дозволили - запускаємо сервіс
                startStepService()
            }
        }
    )

    LaunchedEffect(Unit) {
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.Q) {
            // Перевіряємо, чи вже є дозвіл
            val hasPermission = ContextCompat.checkSelfPermission(
                context,
                Manifest.permission.ACTIVITY_RECOGNITION
            ) == PackageManager.PERMISSION_GRANTED

            if (hasPermission) {
                // Вже є дозвіл -> запускаємо
                startStepService()
            } else {
                // Немає -> питаємо
                permissionLauncher.launch(Manifest.permission.ACTIVITY_RECOGNITION)
            }
        } else {
            // Для старих Android дозвіл не потрібен, просто запускаємо
            startStepService()
        }

        // Додатково запитуємо дозвіл на сповіщення (Android 13+)
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.TIRAMISU) {
            if (ContextCompat.checkSelfPermission(context, Manifest.permission.POST_NOTIFICATIONS) !=
PackageManager.PERMISSION_GRANTED) {
                // Тут можна додати окремий лаунчер, але для простоти поки опустимо
            }
        }
    }



    LaunchedEffect(Unit) {
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.Q) {
            permissionLauncher.launch(Manifest.permission.ACTIVITY_RECOGNITION)
        }
    }

    Scaffold(containerColor = MaterialTheme.colorScheme.background) { padding ->
        Column(
            modifier = Modifier
                .padding(padding)
                .fillMaxSize()
                .verticalScroll(rememberScrollState())
                .padding(horizontal = 16.dp),
            horizontalAlignment = Alignment.CenterHorizontally
        ) {
```

```
// Верхня панель
Row(
    modifier = Modifier
        .fillMaxWidth()
        .padding(top = 16.dp, bottom = 32.dp),
    horizontalArrangement = Arrangement.SpaceBetween,
    verticalAlignment = Alignment.CenterVertically
) {
    // Кнопка Історії (Зліва)
    IconButton(onClick = onNavigateToHistory) {
        Icon(
            imageVector = Icons.Default.DateRange, // Іконка календаря/історії
            contentDescription = "History",
            tint = MaterialTheme.colorScheme.onBackground
        )
    }

    // Кнопка Профілю (Справа)
    IconButton(onClick = onNavigateToProfile) {
        Icon(
            imageVector = Icons.Default.Person,
            contentDescription = "Profile",
            tint = MaterialTheme.colorScheme.onBackground
        )
    }
}

// 1. Головний лічильник
CircularStepCounter(
    steps = state.steps,
    goal = state.dailyGoal,
    progress = state.progress
)

Spacer(modifier = Modifier.height(24.dp))

// 2. Статистика
Row(
    modifier = Modifier.fillMaxWidth(),
    horizontalArrangement = Arrangement.SpaceEvenly
) {
    InfoCard(
        icon = Icons.Default.Favorite,
        value = "${state.calories}",
        label = "Ккал",
        color = Color(0xFFFF7043)
    )
    InfoCard(
        icon = Icons.Default.LocationOn,
        value = String.format("%.1f", state.distance),
        label = "Км",
        color = Color(0xFF42A5F5)
    )
}
```

```
            Spacer(modifier = Modifier.height(26.dp))

            // Рекомендація
            RecommendationCard(text = state.recommendation)

            Spacer(modifier = Modifier.height(16.dp))

            // Картка ІМТ (Індекс Маси Тіла) - яку ми додали раніше
            // Якщо ти її ще не додав, розкоментуй або додай функцію BmiCard внизу
            BmiCard(bmi = state.bmi, category = state.bmiCategory)
            Spacer(modifier = Modifier.height(16.dp))

            // Водний баланс
            WaterCard(liters = state.waterIntake)


            // Додатковий відступ знизу, щоб контент не прилипав до краю екрану при прокрутці
            Spacer(modifier = Modifier.height(12.dp))
        }
    }
}

@Composable
fun CircularStepCounter(steps: Int, goal: Int, progress: Float) {
    Box(contentAlignment = Alignment.Center, modifier = Modifier.size(250.dp)) {
        // Фонове кільце (адаптивне)
        CircularProgressIndicator(
            progress = { 1f },
            modifier = Modifier.fillMaxSize(),
            // Використовуємо колір тексту з прозорістю - це буде виглядати добре і на білому, і на
чорному
            color = MaterialTheme.colorScheme.onSurface.copy(alpha = 0.1f),
            strokeWidth = 20.dp,
            trackColor = Color.Transparent,
        )
        // Активне кільце
        CircularProgressIndicator(
            progress = { progress },
            modifier = Modifier.fillMaxSize(),
            color = Color(0xFF66BB6A), // Можна залишити зеленим або взяти
MaterialTheme.colorScheme.primary
            strokeWidth = 20.dp,
            strokeCap = StrokeCap.Round,
            trackColor = Color.Transparent,
        )
        // Текст всередині
        Column(horizontalAlignment = Alignment.CenterHorizontally) {
            Text(
                text = "$steps",
                style = MaterialTheme.typography.displayMedium,
                fontWeight = FontWeight.Bold,
                color = MaterialTheme.colorScheme.onBackground
            )
```

```
            Text(
                text = "/ $goal кроків", // Показуємо ціль
                style = MaterialTheme.typography.bodyLarge,
                color = Color.Gray
            )
        }
    }
}

@Composable
fun InfoCard(icon: ImageVector, value: String, label: String, color: Color) {
    Column(horizontalAlignment = Alignment.CenterHorizontally) {
        Box(
            modifier = Modifier
                .size(60.dp)
                .clip(CircleShape)
                .background(color.copy(alpha = 0.2f)),
            contentAlignment = Alignment.Center
        ) {
            Icon(
                imageVector = icon,
                contentDescription = label,
                tint = color,
                modifier = Modifier.size(30.dp)
            )
        }
        Spacer(modifier = Modifier.height(8.dp))
        Text(
            text = value,
            style = MaterialTheme.typography.titleLarge,
            fontWeight = FontWeight.Bold,
            // !ВАЖЛИВО: Адаптивний колір для цифр
            color = MaterialTheme.colorScheme.onBackground
        )
        Text(
            text = label,
            style = MaterialTheme.typography.bodyMedium,
            // !ВАЖЛИВО: Адаптивний колір для підписів
            color = MaterialTheme.colorScheme.onSurfaceVariant
        )
    }
}

@Composable
fun RecommendationCard(text: String) {
    Card(
        shape = RoundedCornerShape(24.dp),
        colors = CardDefaults.cardColors(
            // surfaceVariant - це трохи відмінний від фону колір (темно-сірий у Dark Mode)
            containerColor = MaterialTheme.colorScheme.surfaceVariant,
        ),
        elevation = CardDefaults.cardElevation(defaultElevation = 0.dp), // Elevation 0 краще для Modern
look
        modifier = Modifier.fillMaxWidth()
```

```
    ) {
        Column(modifier = Modifier.padding(24.dp)) {
            Row(verticalAlignment = Alignment.CenterVertically) {
                Icon(
                    imageVector = Icons.Default.Notifications,
                    contentDescription = null,
                    tint = MaterialTheme.colorScheme.primary
                )
                Spacer(modifier = Modifier.width(8.dp))
                Text(
                    text = "Порада дня",
                    style = MaterialTheme.typography.titleMedium,
                    fontWeight = FontWeight.Bold,
                    // Текст, який читається на surfaceVariant
                    color = MaterialTheme.colorScheme.onSurfaceVariant
                )
            }
            Spacer(modifier = Modifier.height(12.dp))
            Text(
                text = text,
                style = MaterialTheme.typography.bodyLarge,
                lineHeight = 24.sp,
                // Текст, який читається на surfaceVariant
                color = MaterialTheme.colorScheme.onSurfaceVariant
            )
        }
    }
}

// Новий компонент для води
@Composable
fun WaterCard(liters: Int) {
    val litersFormatted = String.format("%.1f", liters / 1000f)

    Card(
        shape = RoundedCornerShape(16.dp),
        colors = CardDefaults.cardColors(
            containerColor = Color(0xFFE3F2FD) // Світло-блакитний
        ),
        modifier = Modifier.fillMaxWidth()
    ) {
        Row(
            modifier = Modifier.padding(16.dp),
            verticalAlignment = Alignment.CenterVertically
        ) {
            // Іконка краплі (потрібно додати імпорт: androidx.compose.material.icons.filled.WaterDrop)
            // Якщо WaterDrop немає (залежить від версії), візьми LocalDrink
            Icon(
                imageVector = Icons.Default.Favorite,
                contentDescription = "Water",
                tint = Color(0xFF1E88E5),
                modifier = Modifier.size(40.dp)
            )
            Spacer(modifier = Modifier.width(16.dp))
```

```
            Column {
                Text(
                    text = "Водний баланс",
                    style = MaterialTheme.typography.titleMedium,
                    fontWeight = FontWeight.Bold,
                    color = Color(0xFF0D47A1)
                )
                Text(
                    text = "Рекомендовано: $litersFormatted л",
                    style = MaterialTheme.typography.bodyMedium,
                    color = Color(0xFF1565C0)
                )
            }
        }
    }
}

@Composable
fun BmiCard(bmi: Double, category: String) {
    // Вибираємо колір залежно від категорії
    val color = when (category) {
        "Норма" -> Color(0xFF66BB6A) // Зелений
        "Недостатня вага" -> Color(0xFFFFA726) // Помаранчевий
        else -> Color(0xFFEF5350) // Червоний
    }

    Card(
        shape = RoundedCornerShape(16.dp),
        colors = CardDefaults.cardColors(containerColor = color.copy(alpha = 0.15f)),
        modifier = Modifier.fillMaxWidth()
    ) {
        Row(
            modifier = Modifier.padding(16.dp),
            verticalAlignment = Alignment.CenterVertically
        ) {
            Icon(
                imageVector = Icons.Default.Person,
                contentDescription = null,
                tint = color,
                modifier = Modifier.size(32.dp)
            )
            Spacer(modifier = Modifier.width(16.dp))
            Column {
                Text(
                    text = "Індекс Маси Тіла: ${String.format("%.1f", bmi)}",
                    style = MaterialTheme.typography.titleMedium,
                    fontWeight = FontWeight.Bold,
                    color = color
                )
                Text(
                    text = "Категорія: $category",
                    style = MaterialTheme.typography.bodyMedium,
                    color = MaterialTheme.colorScheme.onSurface
                )
```

```
        }
      }
   }
}
```

Файл: com/student/krok/presentation/history/HistoryScreen.kt

```
package com.student.krok.presentation.history

import android.annotation.SuppressLint
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.items
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.automirrored.filled.ArrowBack
import androidx.compose.material3.*
import androidx.compose.runtime.Composable
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.unit.dp
import com.student.krok.domain.model.ActivityStat

@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun HistoryScreen(
   history: List<ActivityStat>,
   onBack: () -> Unit
) {
   Scaffold(
      topBar = {
         TopAppBar(
            title = { Text("Історія активності") },
            navigationIcon = {
               IconButton(onClick = onBack) {
                  Icon(Icons.AutoMirrored.Filled.ArrowBack, contentDescription = "Назад")
               }
            }
         )
      }
   ) { padding ->
      if (history.isEmpty()) {
         Box(
            modifier = Modifier.padding(padding).fillMaxSize(),
            contentAlignment = Alignment.Center
         ) {
            Text("Історія поки порожня", color = MaterialTheme.colorScheme.secondary)
         }
      } else {
         LazyColumn(
            modifier = Modifier
               .padding(padding)
               .fillMaxSize()
               .padding(16.dp),
            verticalArrangement = Arrangement.spacedBy(12.dp)
```

```
            ) {
                items(history) { stat ->
                    HistoryItem(stat)
                }
            }
        }
    }
}

@SuppressLint("DefaultLocale")
@Composable
fun HistoryItem(stat: ActivityStat) {
    Card(
        modifier = Modifier.fillMaxWidth(),
        colors = CardDefaults.cardColors(
            containerColor = MaterialTheme.colorScheme.surfaceContainer
        )
    ) {
        Row(
            modifier = Modifier
                .fillMaxWidth()
                .padding(16.dp),
            horizontalArrangement = Arrangement.SpaceBetween,
            verticalAlignment = Alignment.CenterVertically
        ) {
            Column {
                Text(
                    text = stat.date,
                    style = MaterialTheme.typography.titleMedium,
                    color = MaterialTheme.colorScheme.onSurface
                )
                Text(
                    text = "${stat.calories} ккал | ${String.format("%.2f", stat.distanceMeters / 1000)} км",
                    style = MaterialTheme.typography.bodyMedium,
                    color = MaterialTheme.colorScheme.secondary
                )
            }

            Text(
                text = "${stat.steps}",
                style = MaterialTheme.typography.headlineSmall,
                fontWeight = FontWeight.Bold,
                color = MaterialTheme.colorScheme.primary
            )
        }
    }
}
```

Файл: com/student/krok/presentation/profile/ProfileScreen.kt

```
package com.student.krok.presentation.profile

import androidx.compose.foundation.layout.*
import androidx.compose.foundation.rememberScrollState
import androidx.compose.foundation.text.KeyboardOptions
```

```
import androidx.compose.foundation.verticalScroll
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.automirrored.filled.ArrowBack
import androidx.compose.material3.*
import androidx.compose.runtime.*
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.text.input.KeyboardType
import androidx.compose.ui.unit.dp
import com.student.krok.data.local.UserProfile

@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun ProfileScreen(
    userProfile: UserProfile,
    onSave: (UserProfile) -> Unit,
    onBack: () -> Unit,
    onDeleteProfile: () -> Unit
) {
    // Ініціалізуємо стан
    // Ім'я та Стать беремо з профілю, але не даємо змінювати (вони константи в UI)
    val name = userProfile.name
    val isMale = userProfile.isMale

    // Інші поля можна редагувати
    var weight by remember { mutableStateOf(userProfile.weight.toString()) }
    var height by remember { mutableStateOf(userProfile.height.toString()) }
    var age by remember { mutableStateOf(userProfile.age.toString()) }
    var activityLevel by remember { mutableFloatStateOf(userProfile.activityLevel.toFloat()) }
    var dailyGoal by remember { mutableStateOf(userProfile.dailyGoal.toString()) }

    // Стан для діалогу підтвердження
    var showDeleteDialog by remember { mutableStateOf(false) }

    val activityDescription = when (activityLevel.toInt()) {
        1 -> "Сидячий (1)"
        2 -> "Легка (2)"
        3 -> "Помірна (3)"
        4 -> "Висока (4)"
        else -> "Екстремальна (5)"
    }

    Scaffold(
        topBar = {
            TopAppBar(
                title = { Text("Налаштування") },
                navigationIcon = {
                    IconButton(onClick = onBack) {
                        Icon(Icons.AutoMirrored.Filled.ArrowBack, contentDescription = "Назад")
                    }
                }
            )
        }
```

```
) { padding ->
    Column(
        modifier = Modifier
            .padding(padding)
            .fillMaxSize()
            .padding(16.dp)
            .verticalScroll(rememberScrollState()),
        horizontalAlignment = Alignment.CenterHorizontally
    ) {
        // --- ІМ'Я (Тільки для читання) ---
        OutlinedTextField(
            value = name,
            onValueChange = { }, // Нічого не робимо
            label = { Text("Ім'я (не змінюється)") },
            modifier = Modifier.fillMaxWidth(),
            readOnly = true,  // Не можна редагувати
            enabled = false,   // Робимо сірим
            colors = OutlinedTextFieldDefaults.colors(
                disabledTextColor = MaterialTheme.colorScheme.onSurface,
                disabledBorderColor = MaterialTheme.colorScheme.outline,
                disabledLabelColor = MaterialTheme.colorScheme.onSurfaceVariant
            )
        )

        Spacer(modifier = Modifier.height(16.dp))

        // Ряд: Вага | Зріст | Вік
        Row(Modifier.fillMaxWidth(), horizontalArrangement = Arrangement.spacedBy(8.dp)) {
            OutlinedTextField(
                value = weight,
                onValueChange = { if (it.all { char -> char.isDigit() }) weight = it },
                label = { Text("Вага (кг)") },
                keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Number),
                modifier = Modifier.weight(1f)
            )
            OutlinedTextField(
                value = height,
                onValueChange = { if (it.all { char -> char.isDigit() }) height = it },
                label = { Text("Зріст (см)") },
                keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Number),
                modifier = Modifier.weight(1f)
            )
            OutlinedTextField(
                value = age,
                onValueChange = { if (it.all { char -> char.isDigit() }) age = it },
                label = { Text("Вік") },
                keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Number),
                modifier = Modifier.weight(1f)
            )
        }

        Spacer(modifier = Modifier.height(16.dp))

        // --- СТАТЬ (Заблокована) ---
```

```
Row(verticalAlignment = Alignment.CenterVertically) {
    Text("Стать:", style = MaterialTheme.typography.bodyLarge)
    Spacer(modifier = Modifier.width(8.dp))
    // Блокуємо кнопки
    FilterChip(
        selected = isMale,
        onClick = { },
        label = { Text("Чол") },
        enabled = false // Вимкнено
    )
    Spacer(modifier = Modifier.width(8.dp))
    FilterChip(
        selected = !isMale,
        onClick = { },
        label = { Text("Жін") },
        enabled = false // Вимкнено
    )
}

Spacer(modifier = Modifier.height(16.dp))

Text("Активність: $activityDescription", style = MaterialTheme.typography.titleMedium)
Slider(
    value = activityLevel,
    onValueChange = { activityLevel = it },
    valueRange = 1f..5f,
    steps = 3
)

Spacer(modifier = Modifier.height(16.dp))

OutlinedTextField(
    value = dailyGoal,
    onValueChange = {
        if (it.all { char -> char.isDigit() } && it.length <= 6) dailyGoal = it
    },
    label = { Text("Ціль кроків") },
    keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Number),
    modifier = Modifier.fillMaxWidth()
)

Spacer(modifier = Modifier.height(32.dp))

// Кнопка Зберегти
Button(
    onClick = {
        if (weight.isNotEmpty() && height.isNotEmpty()) {
            onSave(
                UserProfile(
                    name = name, // Беремо старе ім'я
                    weight = weight.toIntOrNull() ?: 70,
                    height = height.toIntOrNull() ?: 170,
                    age = age.toIntOrNull() ?: 25,
                    isMale = isMale, // Беремо стару стать
```

```
                        activityLevel = activityLevel.toInt(),
                        dailyGoal = dailyGoal.toIntOrNull() ?: 10000
                    )
                )
            }
        },
        modifier = Modifier.fillMaxWidth()
    ) {
        Text("Зберегти зміни")
    }

    Spacer(modifier = Modifier.height(16.dp))

    // Кнопка Видалити (відкриває діалог)
    OutlinedButton(
        onClick = { showDeleteDialog = true }, // <--- ВІДКРИВАЄМО ДІАЛОГ
        colors = ButtonDefaults.outlinedButtonColors(contentColor = MaterialTheme.colorScheme.error),
        modifier = Modifier.fillMaxWidth()
    ) {
        Text("Видалити профіль та дані")
    }
}

// --- ДІАЛОГ ПІДТВЕРДЖЕННЯ ---
if (showDeleteDialog) {
    AlertDialog(
        onDismissRequest = { showDeleteDialog = false },
        title = { Text("Видалити профіль?") },
        text = { Text("Ви впевнені? Вся історія активності та налаштування будуть втрачені назавжди.") },
        confirmButton = {
            Button(
                onClick = {
                    showDeleteDialog = false
                    onDeleteProfile() // Викликаємо реальне видалення
                },
                colors = ButtonDefaults.buttonColors(containerColor = MaterialTheme.colorScheme.error)
            ) {
                Text("Так, видалити")
            }
        },
        dismissButton = {
            TextButton(onClick = { showDeleteDialog = false }) {
                Text("Скасувати")
            }
        }
    )
}
}
}
```