МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра комп'ютерних наук

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ МОБІЛЬНОГО ЗАСТОСУНКУ ДЛЯ МОНІТОРИНГУ ЗДОРОВ'Я ТА ФІЗИЧНОЇ АКТИВНОСТІ»**

Виконав: студент 4 курсу, групи 6.1222
спеціальності 122 Комп'ютерні науки
(шифр і назва спеціальності)
освітньої програми Комп'ютерні науки
(назва освітньої програми)
М.А. Бистрик
(ініціали та прізвище)
Керівник доцент кафедри комп'ютерних наук, доцент, к.т.н. Миронова Н.О.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент доцент кафедри програмної інженерії, доцент, к.т.н. Лимаренко Ю.О.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя 2026

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра комп'ютерних наук

Рівень вищої освіти бакалавр

Спеціальність 122 Комп'ютерні науки
(шифр і назва)

Освітня програма Комп'ютерні науки

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри комп'ютерних наук, к.пед.н., доцент

_____ Пшенична О.С.
(підпис)

" 08 " грудня 2025 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Бистрику Мирославу Андрійовичу
(прізвище, ім'я та по-батькові)

1. Тема роботи Проєктування та реалізація мобільного застосунку для моніторингу здоров'я та фізичної активності

керівник роботи Миронова Наталя Олексіївна , к.т.н., доцент
(прізвище, ім'я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 02 » грудня 2025 року № 1741-с

2. Строк подання студентом роботи 04.06.2026

3. Вихідні дані до роботи 1. Постановка задачі: розробка застосунку моніторингу
2. Перелік літератури, технічна документація
3. Додаткові відомості: Kotlin, Jetpack Compose

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі: аналіз предметної області.
2. Основні теоретичні відомості: проєктування архітектури системи.
3. Розділ у відповідності з темою: реалізація андроїд застосунку.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
презентація, UML-діаграма, скриншоти

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 05.12.2026

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	15.12.2025	
2.	Збір вихідних даних.	21.01.2026	
3.	Обробка методичних та теоретичних	16.02.2026	
	джерел.		
4.	Розробка першого та другого розділу.	16.03.2026	
5.	Розробка третього розділу.	11.05.2026	
6.	Оформлення та нормоконтроль	15.05.2026	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	18.06.2026	

Студент _____ М.А. Бистрик
(підпис) (ініціали та прізвище)

Керівник роботи _____ Н.О. Миронова
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ О.Г.Спиця
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Проєктування та реалізація мобільного застосунку для моніторингу здоров’я та фізичної активності»: 64 с., 5 рис., 3 табл., 20 джерел, 1 додаток.

БІОПСИХОСОЦІАЛЬНА МОДЕЛЬ, ГОМЕОСТАЗ, ДЕКЛАРАТИВНИЙ ІНТЕРФЕЙС, КОТЛІН, МОБІЛЬНИЙ ЗАСТОСУНОК, МОНІТОРИНГ ЗДОРОВ’Я.

Об’єкт дослідження – процес аналізу та моніторингу біопсихосоціальних показників здоров’я людини за допомогою мобільних технологій.

Предмет дослідження – математичні моделі гомеостазу, алгоритми кінематичної фільтрації даних та методи проєктування інтерфейсів користувача для застосунків моніторингу здоров’я.

Мета роботи: розробити та реалізувати мобільний застосунок для усвідомленого моніторингу здоров’я, який використовує алгоритми нелінійного оцінювання для формування простих рекомендацій.

Методи дослідження – аналітичний та порівняльний (для дослідження ринку існуючих mHealth-систем та обґрунтування власної концепції), описовий (для формалізації вимог та опису архітектури застосунку);

Основною метою проєкту стало розроблення мобільного застосунку для операційної системи Android з метою усвідомленого відстеження ментального здоров’я, самопочуття та фізичного здоров’я. На противагу існуючих рішень, застосунок розроблений з фокусом на приватність, усвідомленість та мінімалізм у відслідковуванні показників. Для розрахунку стану користувача впроваджений розрахунок фізичної та ментальної оцінки використовуючи систему штрафів. Програмну реалізацію виконано за допомогою Kotlin, Jetpak Compose та архітектури MVVM.

SUMMARY

Bachelor's Qualifying Thesis «Design and Implementation of a Mobile Application for Monitoring Health and Physical Activity»: 64 pages, 5 figures, 3 tables, 20 sources, 1 supplement.

BIOPSYCHOSOCIAL MODEL, HOMEOSTASIS, DECLARATIVE INTERFACE, KOTLIN, MOBILE APPLICATION, HEALTH MONITORING.

Object of the study – process of analyzing and monitoring human biopsychosocial health indicators using mobile technologies.

Subject of the study – mathematical models of homeostasis, kinematic data filtering algorithms, and user interface design methods for health monitoring application.

Aim of the study: to design and implement a mobile application for mindful health monitoring that utilizes non-linear estimation algorithms to generate simple recommendations.

Methods of research – analytical and comparative (for researching the market of existing mHealth systems and justifying the proposed concept), descriptive (for formalizing requirements and describing the application architecture).

The primary objective of the project was to develop a mobile application for the Android operating system designed for mindful tracking of mental health, well-being, and physical fitness. In contrast to existing solutions, the application is designed with a core focus on privacy, mindfulness, and minimalism in monitoring health metrics. To evaluate user's condition, a system for calculating integrated physical and mental scores using penalty functions was introduced. The software implementation was carried out using the Kotling programming language, the Jetpack Compose framework and the MVVM architectural pattern.

ЗМІСТ

Завдання на кваліфікаційну роботу……..2
Реферат 4
Summary 5
Зміст 6
Вступ 7
1 Аналіз предметної області та обґрунтування технологій 9
1.1 Дослідження проблем приватності та безпеки даних у сучасних mHealth-системах 9
1.2 Концепція інтерфейсного мінімалізму та «Mindful Tracking» 12
1.3 Обґрунтування вибору інструментів розробки 17
2 Архітектурне проєктування та математичне модулювання застосунку 20
2.1 Проєктування архітектури системи на базі шаблону MVVM 20
2.2 Математична модель біологічного гомеостазу та оцінювання стану користувача 24
2.2.1 Розрахунок інтегральної фізичної оцінки 25
2.2.2 Розрахунок інтегральної ментальної оцінки 26
2.3 Алгоритм кінематичної фільтрації вхідних даних 28
3 Програмна реалізація та валідація результатів 30
3.1 Розробка динамічного користувацького інтерфейсу 30
3.2 Реалізація бізнес-логіки 33
3.3 Експериментальне тестування алгоритмів та аналіз працездатності 36
Висновки 40
Перелік посилань 42
Додаток А Програмний код 45

ВСТУП

Еволюція засобів контролю персональних показників пройшла шлях від паперових блокнотів, таблиць та підручних засобів, що супроводжувалося високим рівнем суб'єктивізму та ризиком втрати даних, до сучасних мобільних систем. З появою та стрімким поширення мобільних платформ, вектор змістився у бік цифровізації. Проте перші рішення були простим переносом паперових рішень у цифровий вимір: електроні таблиці, документи.

Аналіз сучасного ринку mHealth-систем виявляє гостру проблему безпеки персональної інформації. Більшість комерційних застосунків, такі як Google Fit, Mi Fit, Samsung Health, функціонують за архітектурою з обов'язковою цифровою синхронізацією. Це створює серйозні ризики, оскільки у хмарі компанії вільні робити що завгодно з персональною інформацією користувачів, головне достатньо обережно, щоб їх не спіймали правоохоронні органи. Та навіть у випадку, коли викривають їхнє зловживання персональною інформацією, вони платять невеликий штраф, який є в тисячі разів менший за прибуток отриманий зі зловживання ними особистою інформацією: Тренування ШІ моделей, продаж даних рекламодавцям, і тому подібне. Крім того, оскільки дані на хмарі розміщені разом з даними мільйонів інших користувачів це створює реальний вектор атаки для хакерів, що регулярно і відбувається. У зв'язку з цим виникає необхідність розробки повністю локальних рішень, де всі дані зберігаються безпосередньо на пристрої користувача без передачі на сторонні сервери.

Окрім безпекових факторів, вагомим недоліком існуючих трекерів є явище інформаційного перевантаження (Feature Bloat). Сучасні продукти перенасичені функціоналом, більшість якого не використовувані більшістю користувачів, рекламою, складними графіками та підписками, які роздувають вагу застосунку, ціну та його залежність від хмари. Крім того, вони часто залежать від того чи хоче компанія далі підтримувати сервіс. Відомі безліч випадків, коли компанія

вирішує припинити підтримку сервісу за однією чи іншою причиною без попереднього попередження та без повернення грошей користувачам, це все викликає когнітивну втому замість допомоги. Парадигма інтерфейсного мінімалізму та концепція «Mindful Tracking», що покладені в основу розробки, дозволяють вирішити цю проблему шляхом фокусування на базових функціях необхідних для більшості користувачів та надання простого для розуміння відгуку в формі балів для ментального та фізичного фактору, а також простих та зрозумілих рекомендацій. Крім цього, фокусуючись не на сенсорах, а на ручному вводі, застосунок змушує користувача звертати увагу на всі фактори здоров’я: активність, відпочинок, споживання їжі та суб’єктивна оцінка настрою та інше.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ОБҐРУНТУВАННЯ ТЕХНОЛОГІЙ

1.1 Дослідження проблем приватності та безпеки даних у сучасних mHealth-системах

Після широкого впровадження споживчих портативних пристроїв (wearables) та мобільних застосунків для моніторингу здоров’я (mHealth), корпорації отримали доступ до безпрецедентних масштабів збору чутливої інформації[1][5]. Сучасний ринок пропонує користувачам надзвичайно складні апаратно-програмні комплекси, які використовують технології комп'ютерного зору, штучного інтелекту (ШІ) та безперервного пасивного трекінгу для аналізу фізичної активності, якості сну, серцевого ритму та десятків інших фізіологічних показників. Найбільш критичними з них є непрозорість алгоритмів обчислення показників здоров'я, що отримала назву проблеми "чорної скриньки"[4], системні загрози для конфіденційності персональних даних користувачів в умовах хмарних інфраструктур, а також поступове відчуження людини від усвідомленого, когнітивного контролю над власним тілом і психологічним станом.

Концепція систематичної кількісної оцінки життєвих показників набула формальних обрисів на початку 2010-х років у вигляді глобального руху "Quantified Self" (Кількісне вимірювання себе), філософія якого була сформульована редакторами журналу Wired як "самопізнання через цифри"[2]. На ранніх етапах цього напрямку домінував етос «зроби сам». Спільнота ентузіастів мала антропологічний характер, а користувачі брали активну, безпосередню участь у зборі власних даних. Цей процес охоплював ведення паперових та електронних таблиць, використання ранніх носних сенсорів та ручну фіксацію метрик, що варіювалися від аналізу дієтичного харчування до суб’єктивної оцінки настрою та якості навколишнього середовища. Головною

метою цієї діяльності була «авто-аналітика» або «біохакінг» - спроба виявити неочевидні кореляції між повсякденною поведінкою та фізіологічними станами для оптимізації продуктивності. Цей етап характеризувався надзвичайно високим рівнем усвідомленості: акт ручної фіксації інформації вимагав вольового зусилля, що стимулювало глибокий аналіз власних звичок та поведінкових патернів.

Проте у період між 2012 та 2015 роками індустрія здійснила масштабний розворот. Їхньою ціллю було прибрати проблему «тертя» (friction), неминуче пов’язану з ручним введенням даних. Зі стрімким розвитком мікроелектромеханічних систем (MEMS) та їх масовим впровадженням у споживчу електроніку, фокус розробників різко змістився на технології «пасивного трекінгу»[7]. Смартфони та ранні смарт-годинники перетворилися на своєрідні «репозиторії особистості», використовуючи вбудовані цифрові акселерометри, гіроскопи, пульсоксиметри, барометри та оптичні датчики серцевого ритму для автоматичного збору інформації без втручання, а, зачасто і свідомого відома користувача.

Наслідком даного технологічного зсуву стала ситуація, в якій до 2025 року «тягар» моніторингу практично повністю зник, на його місце прийшли «невидимі» пасивні сенсори, інтегровані у смарт-каблучки, годинники та навіть матраци. Разом з цим, «тягар» інтерпретації зібраних даних зник, замість користувачів, він був покладений на генеративний штучний інтелект та хмарні аналітичні платформи. Споживчі пристрої зазнали процесу «медикалізації», перетворившись з простих аксесуарів для стилю життя на інструменти, що здатні виявляти фібриляцію передсердь чи апное уві сні. Хоча такий пасивний трекінг значно знизив бар'єр входу для масового споживача, він призвів до серйозних і часто непередбачуваних психологічних та етичних наслідків, які потребують детального розгляду

Домінування систем пасивного збору даних породило феномен, який дослідники називають "втомою від портативних пристроїв" (wearable fatigue)[9]. Вже до кінця 2010-х років як спільнота прихильників "Quantified Self", так і

широка громадськість почали ставити під сумнів реальну корисність сирих метрик. Популярна мета у "10 000 кроків", хоча і виявилася геніальною маркетинговою евристикою, була визнана досить грубим і науково не обґрунтованим показником реального стану здоров'я. Користувачі почали повідомляти про те, що постійна алгоритмічна вимога системи досягати автоматично згенерованих цілей (наприклад сумнозвісна вимога «закривати кільця активності» в екосистемі Apple) перетворилася з інструменту мотивації на потужне джерело хронічного стресу. Цей період яскраво підсвітив критичну прогалину у філософії пасивного моніторингу: дані без особистого, емоційного контексту є лише інформаційним шумом. Відбулася небезпечна заміна: Замість того, щоб дослухатися власного тіла та свідомо оцінювати свій рівень втоми чи енергії, люди почали повністю покладатися на цифрові показники, згенеровані хмарою, що викликало ефект плацебо[13]. Якщо пристрій повідомляє про «поганий сон», користувач може відчути себе розбитим, навіть якщо об’єктивно він прокинувся бадьорим. Це призводить до поступової атрофії навичок саморефлексії та повної залежності від технологічного посередника в питаннях оцінки власного самопочуття.

Іншою, пов’язаною проблемою сучасних екосистем є проблема «чорної скриньки»[4]. Корпорації використовують закриті, пропрієтарні алгоритми для інтерпретації зібраних даних. Провідні корпорації маркетизують свої пристрої як інтелектуальні системи, здатні всебічно оцінювати рівень енергії, стресу та загальної фізіологічної готовності організму (наприклад, Readiness Score від Oura або Body Battery від Garmin). Ці показники вираховуються на основі безперервного потоку даних з датчиків. Однак алгоритми, що виконують ці обчислення, є класичною алгоритмічною «чорною скринькою» (black box). Парадокс алгоритмічної непрозорості полягає у тому, що внутрішня логіка цих моделей залишається абсолютно невідомою не лише для кінцевого користувача, але й для незалежного академічного та медичного товариства. Не існує відкритої інформації, які саме статистичні ваги присвоюються різним параметрам, як система обробляє аномалії у вхідних даних, або на яких саме демографічних

вибірках тренувалися ці моделі машинного навчання. Це призводить до ситуацій, коли алгоритм може систематично генерувати некоректні висновки щодо стану здоров’я індивіда, повністю ігноруючи його унікальні фізіологічні особливості або, що важливіше, поточний суб’єктивний настрій та моральний стан[14]. Для багатьох ентузіастів фітнесу такі автоматизовані метрики діють як індикатор "готовності", де вищий пульс у стані спокою однозначно трактується системою як ознака стресу, перетренованості або хвороби. Однак неможливість алгоритму врахувати емоційний контекст робить такі оцінки однобокими. Крім того, застосування складного ШІ створює небезпечну інформаційну асиметрію, де пристрій набуває статусу медичного авторитету. Відмова від складних та непрозорих рішень на користь простих («дурних») прозорих та детермінованих алгоритмів, які базуються виключно на введених користувачем даних є не технічним регресом, а усвідомленим архітектурним вибором. Цей підхід повертає користувачеві суб’єктивність: Якщо людина розуміє математичну формулу, за якою обчислюється її рейтинг здоров’я, вона відновлює повний контроль над інтерпретацією власних біометричних даних.

1.2 Концепція інтерфейсного мінімалізму та «Mindful Tracking»

Проєктування мобільного застосунку для моніторингу здоров’я, орієнтованого на абсолютну приватність, вимагає усвідомлення структурних недоліків існуючих комерційних платформ. Сучасні фітнес-трекери та mHealth системи формують глобальні ризики у сфері інформаційної безпеки, оскільки безперервно акумулюють величезні масиви найбільш інтимної інформації про людину[3][10].

Сучасні фітнес-трекери та пов’язані мобільні застосунки безперервно збирають величезну кількість чутливих деталей про тіло користувача, його звички та рутини. Більшість користувачів навіть не підозрюють про реальний обсяг та глибину цієї інформації. Звичайна екосистема розумного годинника

збирає точну історію локацій та переміщень під час прогулянок та пробіжок, тенденції серцевих скорочень у стану спокою в розрізі тижнів та місяців, детальні метрики якості сну (в тому числі й фази глибокого та швидкого сну), статистику споживання калорій, інтенсивність тренувань, час для відновлення організму та точні коливання ваги при синхронізації з розумними вагами.

У традиційній архітектурі (Cloud-First) усі ці дані безперервно синхронізуються з централізованими хмарними серверами компаній-виробників. Компаній агрегують персональні дані з інформацією мільйонів інших користувачів, створюючи масивні бази даних. Наслідки такої архітектури виходять далеко за межі простої статистика та створюють прямі загрози інтересам користувача[12].

Багато компаній мають прямий фінансовий стимул збирати та монетизувати якомога більші обсяги персональних даних. Доступ до агрегованих та «анонімізованих» (хоча сучасні методи деанонімізації часто дозволяють відновити особу) датасетів продається брокерам даних, рекламодавцям та третім особам, про існування яких користувач не має жодного уявлення[3]. Рекламодавці використовують виявлені патерни для мікротаргету товарів та послуг у сфері здоров’я. Крім того, страхові компанії (особливо у країнах з децентралізованою медичною системою, таких як США) виявляють дедалі більший інтерес до таких баз даних для побудови профілів ризику окремих осів, що може призвести до підвищення страхових премій, або навіть відмови у наданні медичного страхування на основі алгоритмічної оцінки способу життя.

Справжні масштаби обміну даними часто приховуються за складними, перенавантаженими юридичним жаргоном політиками конфіденційності, які користувачі рідко читають. Деякі політики містять вкрай розмиті формулювання, резервуючи за компанією право ділитися даними «для бізнес-цілей» з афілійованими партнерами, або ж навіть передавати всі дані у разі злиття або придбання компанії. Інші компанії виправдовують збір даних необхідністю

«покращення продукту та персоналізації досвіду», що де-факто надає їм безмежну свободу в управлінні чутливою інформацією.

Централізовані хмарні сховища біометричних даних є надзвичайно привабливими мішенями для кіберзлочинців. Медичні та біометричні дані є високоцінним товаром на тіньових ринках. Ці загрози не є теоретичними: за останні роки зафіксовано низку подібних катастрофічних інцидентів. В одному з випадків масштабний витік даних з фітнес-застосунку скомпрометував понад 61 мільйон записів користувачів. В іншому прецедентів стався витік точних GPS-координат домашніх адрес користувачів, звідки вони щоранку розпочинали свої пробіжки. У 2018 році розгорівся міжнародний скандал, коли теплові карти популярного фітнес-додатка випадково розкрили точне місцезнаходження та патрульні маршрути секретних військових баз США, оскільки солдати носили трекери під час виконання бойових завдань[10].

З огляду на вищезазначені етичні та юридичні проблеми, дослідники в галузі медичної інформатики та кібербезпеки наголошують (і наголошували) на необхідності прийняття мислення «Privacy by Design»[11]. Відповідно до цього підходу, конфіденційність має розглядатися як фундаментальна, первинна вимога при проєктуванні системи, а не як функція, що додається згодом, як роздум. У контексті мобільного застосунку, відмова від хмарної інфраструктури та складного ШІ дозволяє реалізувати найбільш радикальну, однак найбільш дієву форму Privacy By Design. Оскільки для роботи простих евристичних алгоритмів не потрібен збір великих даних для навчання нейронних мереж, система повністю відповідає принципу мінімізації даних. Застосунок взагалі не передбачає наявності віддалених серверів, реєстрації акаунтів, збору електронних адрес або телеметрії. Вся логіка обробки введених користувачем даних виконується локально на процесорі мобільного пристрою. Це не тільки повністю усуває ризики витоку інформації з хмари або несанкціонованої передачі її брокерам даних, але й радикально знижує витрати на розробку та підтримку серверної інфраструктури, баз даних та складних механізмів автентифікації.

У контексті домінуючих сьогодні систем автоматичного, пасивного відстеження здоров’я, ідея створення «дурного» застосунку, який вимагає від користувача щоденних ручних зусиль для введення даних (тривалості сну, калорій, води, оцінки настрою) може здатися на перший погляд анахронізмом, кроком у минуле та деградацією у зручності використання[7]. Сучасний користувач звик, що смарт-годинники Apple Watch, Google Pixel Watch, або фітнес трекери Garmin та інших компаній самостійно виконують цю роботу, а необхідність ручного вводу часто сприймається як «тертя» (friction) та тягар. Проте з точки зору психології здоров’я, когнітивно-поведінкової терапії (КПТ) та глибокого особистісного розвитку, саме ручний трекінг володіє унікальними терапевтичними та психологічними перевагами, які принципово неможливо відтворити за допомогою повної автоматизації[8].

Психологічна наука розглядає ручне відстеження параметрів власного здоров'я та поведінки не просто як акт збору статистики, а як фундаментальну техніку підвищення усвідомленості (mindfulness) та когнітивного залучення. Для людей, які мають високий рівень мотивації для самовдосконалення, ручний трекінг створює глибинну саморефлексію. Згідно з дослідженнями, пасивне відстеження має тенденцію призводити до повної втрати зв’язку між індивідом та його фізіологічними потребами. Таким чином, дослідження показують, що надмірне покладання на автоматичні вхідні дані може призвести або до виникнення прогалин у розумінні власного тіла, або до повної відмови від усвідомленого контролю над своїми звичками. Коли датчик рахує кроки чи пульс, мозок людини не бере участі в цьому процесі. Людина стає пасивним спостерігачем звітів про власне життя, а не його активним учасником.

На противагу цьому, введення даних вручну вимагає вольового зусилля та фокусування уваги, хоча б на кілька секунд. Регулярний ручний запис інформації про емоційний стан або якість дозвілля безпосередньо залучає префронтальну кору головного мозку, яка відповідає за аналіз, планування та контроль поведінки. Коли користувач застосунку повинен самостійно вказати рівень свого стресу або оцінити настрій, він змушений прислухатися до власних внутрішніх

відчуттів, просканувати своє тіло та свідомість[13]. Такий підхід є не недоліком інтерфейсу, а потужним терапевтичним механізмом, що сприяє розвитку емоційного інтелекту.

У рамках етапної моделі персональної інформатики, розробленої Лі (Li), рефлексія визначається як найважливіша стадія, на якій люди завершують інтеграцію зібраних даних і переходять до планування або здійснення відповідних коригуючих дій для покращення свого здоров'я[8]. Згідно з працями Дональда Шона (Schön) та інших дослідників, когнітивний процес рефлексії в контексті моніторингу поділяється на два взаємодоповнюючі модуси, кожен з яких ефективно стимулюється саме ручним трекінгом:

- рефлексія в дії (Reflection-in-Action);
- рефлексія після дії (Reflection-on-Action).

Рефлексія в дії – це процес обдумування події, емоції чи стану безпосередньо в момент її (його) виникнення. У контексті мобільного застосунку це відбувається безпосередньо під час введення даних. Наприклад, коли користувач відкриває застосунок, щоб зафіксувати, скільки води він випив, або обирає цифру від 1 до 5 для оцінки власного настрою, він миттєво оцінює свій поточний стан. Ця мікро-зупинка стимулює людину задуматися над причиною свого настрою і може призвести до швидких, адаптивних рішень («я відчуваю втому, може варто відпочити»).

Рефлексія після дії – це модус, що стосується процесу навчання на основі власних думок та досвіду вже після того, як подія чи часовий проміжок завершився. Цей вид рефлексії підтримується шляхом надання користувачеві зворотного зв’язку – наприклад, підсумкового балу за день. Переглядаючи свої результати, користувач може ретроспективно проаналізувати свої вибори та скоригувати плани на наступний день.

Дослідження підтверджують, що додатки, які покладаються виключно на автоматизований збір даних без ручного вводу, забезпечують значно гіршу підтримку процесу саморефлексії[7]. Сучасні наукові пошуки вказують на те, що для досягнення найкращих результатів та формування стійких здорових звичок

необхідно комбінувати ефективність автоматизації (де це доречно) із ручною рефлексією, забезпечуючи оптимальний баланс між зручністю та усвідомленістю (mindfulness). Отже, парадигма "дурного", ручного застосунку парадоксальним чином виявляється більш "розумною" у довгостроковій перспективі впливу на психологічний добробут користувача.

Додатково варто зазначити, що інтерфейс таких систем має бути спроєктований таким чином, щоб знизити бар'єри (capture burden), не втрачаючи при цьому переваг рефлексії. Наприклад, дослідження показують, що використання віджетів для смартфонів дозволяє спростити процес ручного введення та покращити доступ до інформації, збільшуючи загальну прихильність (compliance rate) користувачів до ведення щоденників здоров'я[8].

1.3 Обґрунтування вибору інструментів розробки

Для успішної реалізації концепції локального мінімалістичного застосунку (згідно з парадигмами Privacy by Design та Mindful Tracking [7][11]), особливу увагу було приділено вибору сучасного та оптимізованого технологічного стека. Відмова від готових хмарних рішень (BaaS – Backend as a Service) на кшталт Firebase вимагала проєктування надійної автономної архітектури безпосередньо на стороні клієнта. Основними критеріями вибору інструментів стали: безпека управління пам'яттю, швидкість рендерингу інтерфейсу та здатність забезпечити повністю ізольоване збереження даних.

Основною мовою розробки логіки застосунку обрано Kotlin[15]. З 2019 року компанія Google офіційно оголосила Kotlin пріоритетною мовою для розробки під ОС Android. Вибір цієї статично типізованої мови зумовлений її ключовими архітектурними перевагами порівняно з успадкованою Java, такими як: Null-безпека – вбудована на рівні компілятора система перевірки на порожні посилання унеможливлює виникнення критичних помилок типу NullPointerException; Лаконічність та функціональний підхід – використання

функцій вищого порядку, лямбда виразів та колекцій-розширень (наприклад sumOf, filter, groupBy) дозволило елегантно реалізувати алгоритми оцінювання гомеостазу та кінематичної фільтрації даних без надлишкового шаблонного коду.

Для розв'язання проблеми інтерфейсного перенавантаження я відмовитися від застарілої імперативної системи верстки на базі XML-макетів на користь сучасного фреймворку Jetpack Compose[16]. Compose використовує декларативний підхід до побудови UI. Замість ручного оновлення кожного елемента екрану (через findViewById), розробник лише описує як має виглядати інтерфейс при повному стані (State) даних. Коли алгоритм перераховує Фізичну чи Ментальну оцінку, фреймворк здійснює розумну рекомпозицію (Recomposition), оновлюючи лише ті компоненти, які змінилися. Це ідеально лягає в архітектурний шаблон MVVM (Model-View-ViewModel)[17][19] та забезпечує миттєвий зворотній зв'язок при ручному введенні даних.

Оскільки головною вимогою системи є повна конфіденційність та відсутність зовнішніх серверів, постало питання вибору локального сховища. Використання важких реляційних баз даних (як Sqlite, Postgres чи Room) для збереження простих журнальних засобів створило б надмірне архітектурне навантаження. Тому було обрано швидкий та легкий механізм інкапсуляції даних - зв'язку SharedPreferences та бібліотеки Gson[18].

SharedPreferences забезпечує ізольоване збереження пар «ключ-значення» у приватній директорії застосунку (sandbox), до якої не мають доступу інші програми, або операційна система (без прав root). Бібліотека Gson від Google виконує двосторонню серіалізацію об'єктів (перетворення Kotlin класів у формат JSON для збереження на диск, та де-серіалізацію назад при завантаженні історії). Цей підхід гарантує миттєвий доступ до архіву без використання складних SQL-запитів, що економить заряд акумулятор мобільного пристрою.

Всі етапи програмування, написання коду, профілювання, налагодження та тестування проводилися в офіційному середовищі Android Studio. Використання вбудованих інструментів профілювання (Memory Profiler та CPU profiler)

дозволило оптимізувати споживання оперативної пам’яті під час прокручування великого масиву даних в історії застосунку, а інтеграція з системою збирання Gradle забезпечила модульність та зручність управління залежностями.

Отже, обраний технологічний стек повністю задовольняє функціональні та нефункціональні вимоги до системи, забезпечуючи високу швидкість розробки, надійність роботи алгоритмів та безкомпромісний захист персональних даних користувача.

2 АРХІТЕКТУРНЕ ПРОЄКТУВАННЯ ТА МАТЕМАТИЧНЕ МОДУЛЮВАННЯ ЗАСТОСУНКУ

2.1 Проєктування архітектури системи на базі шаблону MVVM

Проєктування архітектури мобільного застосунку базується на використанні сучасного шаблону проєктування MVVM (Model-View-ViewModel), який рекомендований компанією Google, як стандарт для розробки під операційну систему Android [17][18]. Основною метою вибору даного патерну є забезпечення чіткого розділення відповідальності між компонентами (Separation of Concerns), підвищення гнучкості системи, спрощення тестування та супроводу програмного коду.

Архітектура розділена на три взаємопов’язані рівні: даних (Model), представлення (View) та моделі представлення (ViewModel)(рис. 2.1)

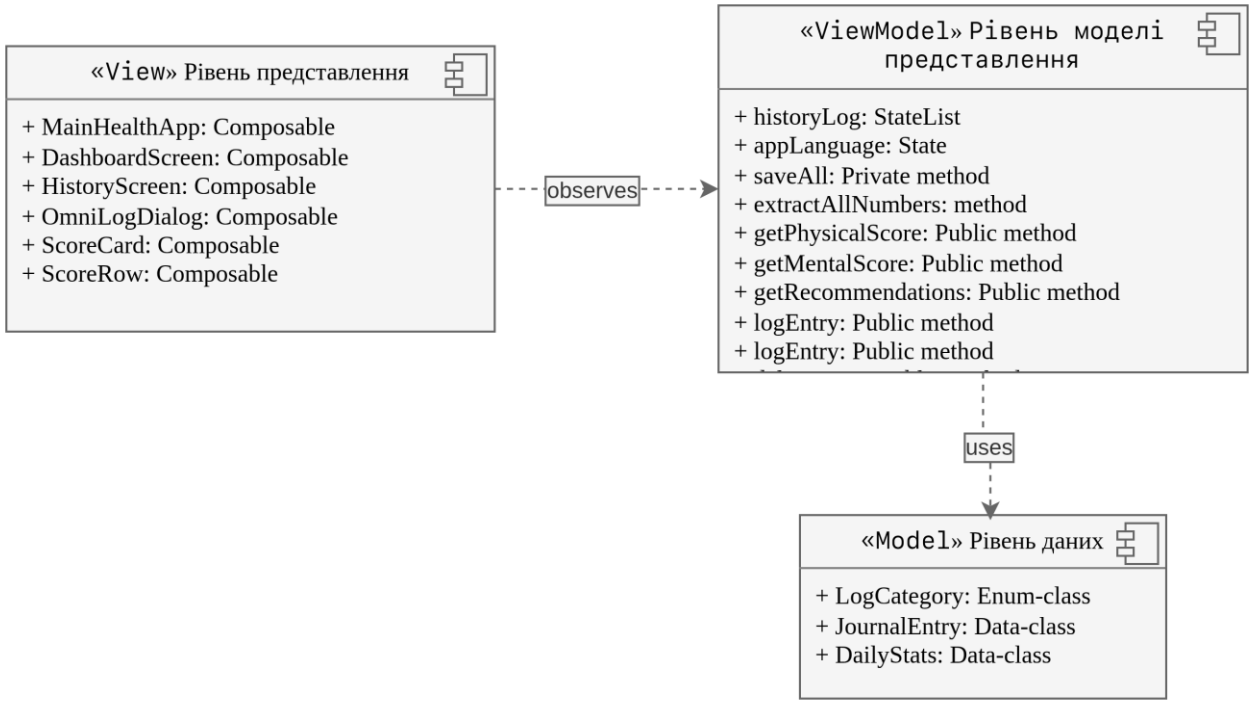


Рисунок 2.1 – UML-діаграма компонент

Рівень даних представлений структурами JounarEntry та DailyStats, а також переліком категорій LogCategory (таблиця 2.1).

Таблиця 2.1 – Специфікація компонентів рівня даних

Назва компоненту	**Тип**	**Опис та призначення**
LogCategory	Enum class	Перелічення всіх доступних категорій для трекінгу (ACTIVITY, WATER, SLEEP, FOOD, MOOD, CAFFEINE, READING, REST, DIARY). Визначає контекст введених даних.
JournalEntry	Data class	Атомарна сутність запису в журналі. Зберігає інформацію про обрану категорію (category), деталі запису у вигляді рядка (detail), а також дату (date) та час (time) створення.
DailyStats	Data class	Структура для агрегації сирих показників користувача за певну добу (активні хвилини, мілілітри води, години сну, спожиті кілокалорії, доза кофеїну). Використовується для генерації рекомендацій.

Клас JounalEntry виступає як атомарна одиниця збереження інформації, що містить тип активності, текстові або числові деталі та часові мітки (дата/час створення запису). Рівень моделі є абсолютно ізольованим від графічного інтерфейсу та містить чисті структури для представлення об’єктів.

Рівень представлення реалізований за допомогою декларативних компонентів Jetpack Compose: MainHealthApp, DashboardScreen, HistoryScreen, OmniLogDialog (таблиця 2.2) Даний рівень відповідає виключно за візуалізацію даних та перехоплення дій користувача (введення показників, перемикання мови, перемикання екранів). Компоненти представлення не містять логіки обробки чи обчислення даних – вони підписані на реактивні стани, які надає архітектурний міст у вигляді ViewModel.

Таблиця 2.2 – Специфікація компонентів рівня представлення

Назва компоненту	**Тип**	**Опис та призначення**
MainHealthApp	Composable	Головний контейнер застосунку (Scaffold). Керує верхньою панеллю, кнопкою виклику діалогу (FAB) та нижньою навігацією між вкладками.
OmniLogDialog	Composable	Атомарна сутність запису в журналі. Зберігає інформацію про обрану категорію (category), деталі запису у вигляді рядка (detail), а також дату (date) та час (time) створення. Універсальне модальне вікно введення даних.
DashboardSreen	Composable	Екран аналітики. Відображає інтегральні фізичні та ментальні оцінки (порівняно зі вчорашнім днем), деталізовану щоденну статистику та блок інтелектуальних рекомендацій.
HistoryScreen	Composable	Екран архіву. Реалізує патерн Progressive Disclosure (угрупування записів за днями з можливістю згортання) та інтегрує жест Swipe-to-Reveal для видалення записів.
ScoreCard / StatRow	Composable	Допоміжні атомарні компоненти для стандартизованого візуального форматування карток з балами та рядків зі статистикою відповідно до принципів Material Design 3.

Рівень моделі представлення реалізований у класі HealthViewModel, що успадковує AndroidViewModel для безпечної взаємодії з контекстом додатка. ViewModel виступає єдиним джерелом істини для інтерфейсу (таблиця 2.3). Вона інкапсулює механізми завантаження та збереження даних через SharedPreferences, виконує серіалізацію за допомогою об’єкта Gson, а також керує реактивними станами historyLog (mutableStateListOf) та isUkrainian (mutableStateOf).

Таблиця 2.3 – Специфікація класу HealthViewModel

Метод / Змінна	**Тип**	**Опис та призначення**
historyLog	Змінна (StateList)	Реактивна колекція (масив) усіх записів журналу. Зміни в ній автоматично ініціюють оновлення інтерфейсу користувача.
appLanguage	Змінна (State)	Реактивна змінна, що зберігає поточну мову системи (en, uk, ja) для динамічної локалізації без перезавантаження.
Init { ... }	Блок ініціалізації	Виконується при старті. Зчитує JSON-архів із SharedPreferences, десеріалізує його через Gson, відфільтровує пошкоджені записи та завантажує їх в оперативну пам'ять.
saveAll()	Приватний метод	Серіалізує поточний стан historyLog у формат JSON та синхронізує його з фізичним сховищем пристрою.
extractAllNumbers()	Допоміжний метод	Використовує регулярні вирази (Regex) для автоматичного парсингу та вилучення числових значень із текстових записів журналу.
getPhysicalScore()	Публічний метод	Алгоритм розрахунку фізичного гомеостазу. Обчислює базові бали за сон, воду і активність.
getMentalScore()	Публічний метод	Алгоритм розрахунку психологічного стану на основі суб'єктивних оцінок настрою та якості відпочинку з дисконтуванням за надлишок кофеїну чи депривацію сну.
getRecommendations()	Публічний метод	Аналітичний модуль. Аналізує щоденну статистику (DailyStats) і через систему логічних умов генерує масив персоналізованих текстових порад для користувача.
logEntry()	Публічний метод	Приймає дані з UI, фіксує поточний системний час та дату, створює об'єкт JournalEntry і додає його на початок історії.
deleteEntry()	Публічний метод	Видаляє конкретний запис із реактивної колекції та ініціює збереження змін.

У системі реалізовано патерн однонаправленого потоку даних. Коли користувач здійснює дію в інтерфейсі (наприклад, зберігає новий запис про спожиту воду), рівень View генерує подію та передає її у ViewModel через метод logEntry().

ViewModel оновлює внутрішній стан колекції та дублює зміни на фізичний диск пристрою за допомогою приватного методу SaveAll().

Оскільки колекція є реактивною, Jetpack Compose автоматично реєструє зміну стану та ініціює точкову рекомпозицію лише тих компонентів екрана, які відображають оновлену інформацію. Це унеможливлює розсинхронізацію даних між екранами історії та головної статистики.

2.2 Математична модель біологічного гомеостазу та оцінювання стану користувача

Для подолання деструктивного впливу лінійних систем оцінювання, властивих деяких комерційних аналогів, у рамках роботи було спроєктовано нелінійну математичну модель оцінювання стану користувача. Модель базується на концепції фізіологічного гомеостазу та використовує інвертовані U-подібні криві, де як дефіцит, так і надлишок певного біологічного чи поведінкового фактора призводять до деградації інтегрального показника.

Обчислення стану розділено на два незалежні домени: Фізичний рейтинг (S_{phys}) та Ментальний рейтинг (S_{ment}). Обидва показники розраховуються динамічно для обраної дати та масштабуються у діапазоні від 0 до 100 балів за формулою:

$$S = clip(B^*(1.0\text{-}P_{total}), 0{,}100),$$

де B – базовий інтегральний показник домену, а P_{total} – кумулятивний відсотковий коефіцієнт штрафів за порушення біологічного балансу.

2.2.1 Розрахунок інтегральної фізичної оцінки

Базовий показник фізичного стану (B_{phys}) формується як активна сума трьох метрик: активності (A_{score}), гідратації (W_{score}) та відновлення сну (R_{score}):

$$B_{phys} = A_{score} + W_{score} + R_{score}.$$

Оцінка активності (A_{score}) обчислюється на основі накопичених балів за тренування, нормованих відносно референтного добового значення, та обмеження верхньою межею у 50 балів:

$$A_{score} = min\left(\frac{P_{act}}{100.0} * 50, 50.0\right).$$

Оцінка гідратації (W_{score}) обчислюється лінійно відносно фізіологічної норми споживання води у 2500 мл, з верхньою межею у 30 балів:

$$W_{score} = min\left(\frac{V_{water}}{2500.0} * 30, 30.0\right).$$

Оцінка відновлення сну (R_{score}) обчислюється за допомогою кусково-постійної функції, що моделює U-криву сну з верхньою межею у 20 балів:

$$M_{sleep} = \begin{cases} 20.0, \text{якщо } T_{sleep} \in [7,9] \text{год}, \\ 10.0, \text{якщо } T_{sleep} \in [5,6] \text{год}, \\ 15.0, \text{якщо } T_{sleep} > 9 \text{год}, \\ 0.0 \text{ в інших випадках.} \end{cases}$$

Кумулятивний штраф фізичного стану (P_{phys_total}) враховує кросдоменні негативні чинники та обмежується максимальним дисконтом у 60% ($min(0.6, \sum P_i)$):

- штраф за профіцит калорій: якщо спожито понад 2500 ккал, нараховується прогресивний штраф $P_1 = min\left(0.4, \frac{E_{kkal}\text{-}2500}{5000.0}\right)$;
- штраф за критичний дефіцит енергії: якщо калорійність раціону становить від 1 до 1000 ккал, нараховується статичний штраф $P_1 = 0.20$;
- штраф за перетренування (виснаження ЦНС): якщо бали активності перевищують 150 одиниць $P_1 = min\left(0.3, \frac{P_{act}\text{-}150}{300}\right)$;
- штраф за зневоднення під час навантажень: якщо активність тривала ($P_{act} > 60$), а рівень споживання води критично низький ($V_{water} < 1500$мл), фіксується штраф $P_4 = 0.15$;
- штраф за гіперсомнію: тривалий сон понад 9 годин створює ефект млявості та дисконтується за формулою $P_5 = min\left(0.2, \left(T_{sleep}\text{-}9\right)*0.05\right)$.

2.2.2 Розрахунок інтегральної ментальної оцінки

Базовий показник ментального стану (B_{ment}) формується на основі якості сну (M_{sleep}), емоційного фону (M_{mood}) та практик усвідомленого відпочинку (M_{mind}):

$$B_{ment} = M_{sleep} + M_{mood} + M_{mind}.$$

Компонент сну (M_{sleep}) має критичне значення для емоційної стабільності, тому його вага становить 40 балів:

$$M_{sleep} = \begin{cases} 40.0, \text{якщо } T_{sleep} \in [7,9]\text{год}, \\ 20.0, \text{якщо } T_{sleep} \in [5,6]\text{год}, \\ 30.0, \text{якщо } T_{sleep} > 9\text{год}, \\ 0.0 \text{ в інших випадках.} \end{cases}$$

Компонент настрою (M_{mood}) базується на середньому арифметичному значенні суб'єктивних оцінок настрою користувача за добу ($R_{mood} \in [1,5]$) з вагою 30 балів:

$$M_{mood} = \frac{\bar{R}_{mood}}{5.0} * 30.0.$$

Компонент рефлексії та відпочинку (M_{mind}) нараховує бали за час, приділений відпочинку (наприклад, читанню, медитації, іграм), його вага становить 30 балів:

$$M_{mind} = min\left(\frac{T_{mind}}{60.0} * 30, 30.0\right).$$

Система штрафів для ментального рейтингу (P_{ment_total}) відображає вплив стимуляторів та дефіциту відпочинку на нервову систему обмежуючись загальним дисконтом у 65% ($min(0.56, \sum P_1)$):

- надмірне споживання кофеїну: доза понад 400 мг може викликати тривожність та веде до штрафу $P_1 = 0.25$; доза від 250 до 400 мг – $P_2 = 0.10$;
- гострий дефіцит сну: тривалість сну від 1 до 4 один руйнує ментальну стабільність і карається жорстким статичним дисконтом $P_3 = 0.35$;
- гіперсомнія: сон понад 9 годин призводить до зниження когнітивної продуктивності та карається статичним дисконтом $P_4 = 0.15$;
- інформаційне або фізичне перевантаження: тривалість тренувань понад 180 хвилин призводить до виснаження ($P_5 = 0.20$), а надмірна тривалість фокусування/медитацій може свідчити про апатію ($P_6 = 0.15$).

2.3 Алгоритм кінематичної фільтрації вхідних даних

Однієї з ключових вразливостей систем ручного або напівавтоматичного введення даних про фізичну активність є проблема навмисного або випадкового викривлення інформації користувачем. Для забезпечення математичної достовірності моделей оцінювання, у рамках роботи розроблено алгоритм кінематичної фільтрації вхідних даних тренувань.

Алгоритм інтегрований у функцію аналізу активності й аналізує два зв'язані кінематичні параметри, що вводяться користувачем: тривалість тренування (t, хв) та подолану відстань (d, м). На основі цих даних обчислюється середня швидкість переміщення за формулою:

$$v = \frac{d/1000}{t/60} = \frac{0.06*d}{t} \text{ (км/год)}.$$

Обчислене значення швидкості виступає критерієм для нелінійної класифікації та фільтрації навантаження за трьома рівнями:

Рівень пасивного або механічного переміщення ($v > 25.0$ км/год). Швидкість понад 25 км/год фізіологічно неможлива для тривалого бігу чи класичного тренування людини без використання транспортних засобів (автомобіль, електросамокат, громадський транспорт). У разі виявленні такого значення алгоритм активує фільтр захисту: показник подоланої відстані повністю ігнорується, а бали активності нараховуються за мінімальною лінійною шкалою, що враховує лише час: $P_{act} = t*0.2$.

Рівень інтенсивного кардіонавантаження ($15 < v \leq 25$ км/год) відповідає професійному бігу або швидкісній їзді на велосипеді. Алгоритм заохочує високу швидкість та розраховує бали з прогресивним ваговим коефіцієнтом для дистанції: $P_{act} = t + \frac{d}{250}$.

Рівень стандартної фізичної активності ($v \leq 15$ км/год) відповідає класичному бігу, підтюпцю, або активній ходьбі. Розрахунок проводиться за базовою формулою: $P_{act} = t + \frac{d}{100}$

Якщо користувач не вказує один з параметрів (наприклад, виконується статичне тренування у тренажерному залі, де відстань дорівнює нулю), алгоритм уникає ділення на нуль завдяки логічному розгалуженню та коректно приймає за фінальний показник чистий час тренування: $P_{act} = t$.

Розроблений кінематичний фільтр дозволяє автоматично відсіювати некоректні дані, забезпечуючи точність та об’єктивність формування результуючих балів гомеостазу.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ВАЛІДАЦІЯ РЕЗУЛЬТАТІВ

Програмна реалізація застосунку [20] базується на декларативному підході Jetpack Compose, що забезпечує модульність інтерфейсу та автоматичний перерендеринг при зміні станів. Архітектура чітко сегментована на візуальний шар (UI) та шар бізнес-логіки з персистентністю (ViewModel), що мінімізує зв'язність компонентів та спрощує подальшу валідацію результатів.

3.1 Розробка динамічного користувацького інтерфейсу

Інтерфейс користувача (UI) застосунку реалізовано за допомогою сучасного декларативного фреймворку Jetpack Compose, що дозволило відмовитися від громіздких XML-макетів та забезпечити високу швидкість рендерингу. Основою графічного інтерфейсу є головна функція MainHealthApp, яка використовує базовий компонент Scaffold для організації простору. Вона містить верхню панель навігації (TopAppBar) з кнопкою динамічної зміни мови інтерфейсу, нижню панель (NavigationBar) для перемикання між екранами Дашборду та Історії, а також плаваючу кнопку (FloatingActionButton), яка викликає діалог логування записів.

Локалізація інтерфейсу та термінології реалізована через допоміжну функцію tr(), яка приймає три рядкові аргументи (переклад тексту англійською, українською та японською мовами) та поточний код мови. Логіка вибору мови інкапсульована у вираз when, що повертає відповідну локалізовану версію залежно від значення змінної стану.

Перелік типів життєвих подій-параметрів для розрахування балів інтегровано за допомогою enum-класу LogCategory. Для нього також оголошено розширення localizedName(), яке делегує виклик функції tr() для динамічного

отримання назв категорій згідно з поточною мовою. Такий підхід гарантує консистентність термінології на всіх екранах без дублювання логіки.

Головною точкою входу візуального шару є композитована (Composable) функція MainHealthApp, яка викликається класом MainActivity.

Функція MainHealthApp приймає модель viewModel() як аргумент, пов'язуючи інтерфейс з бізнес-логікою. Всередині функції оголошено три змінні стану: lang, що отримає поточну мову з viewModel; selectedTab, у якій зберігається цілочисельний індикатор поточної вкладки навігації; та булевий прапорець showOmniDialog, що керує видимістю діалогу введення даних.

Конструкція інтерфейсу у функції MainHealthApp побудована навколо компонента Scaffold, який структурно розділяє екран на зони. У зоні topBar розміщено заголовок та кнопку перемикання мови, яка оновлює стан застосунку через виклик методу viewModel.toggleLanguage().

Нижня панель bottomBar реалізована як NavigationBar з двома елементами (Dashboard та History), що керують значенням змінної selectedTab.

Плаваюча кнопка дій floatinActionButton змінює прапорець showOmniDialog на true, ініціюючи відображення діалогу.

У тілі Scaffold за допомогою умовної логіки рендеру динамічно перемикаються екрани та відображається модальне вікно залежно від станів, що забезпечує централізоване управління навігацією.

Діалог введення даних OmniLogDialog інкапсулює адаптивну форму, оптимізовану під різні типи категорій. Він приймає як аргументи функцію onDismiss, що виконується при закритті діалогу («showOmniDialog = false»), модель viewModel та рядок lang для відображення коректної мови.

Всередині оголошено набір змінних стану для кожного поля: actType, waterMl, sleepHrs, moodScore, foodKcal, caffeineMg, focusMins, diaryText тощо, а також змінна selectedCategory що відстежує поточну категорію запису.

При натисканні на чіп у стрічці категорій оновлюється стан selectedCategory, текст усіх полів вводу та лічильники скидаються до порожнього

рядка («») або значення за замовчуванням, викликаючи функцію resetFields(), що запобігає накопиченню застарілих даних при перемиканні категорій.

Далі використовується компонент AnimatedContent, який анімовано змінює вміст залежно від обраної категорії (selectedCategory). Усередині динамічного блоку за допомогою виразу when створюються відповідні компоненти OutlinedTextField, або слайдер для настрою. Текстові фільтри обмежують введення лише цифрами через перевірку isDigit. Кнопка підтвердження формує структурований рядок деталізації та викликає метод viewModel.logEntry, передаючи категорію та сформовані дані. Після успішної операції діалог закривається через виклик onDismiss. Кнопка скасування просто викликає onDismiss, закриваючи діалог.

Екран панелі керування DashboardScreen відповідає за агрегацію та візуалізацію ключових метрик дня. Він отримує обчислені значення фізичного та ментального станів на сьогоднішній день та вчора через методи viewModel.getPhysicalScore() та viewModel.getMentalScore(), а також отримуючи детальну статистику через viewModel.getDailyStats() та список рекомендацій через viewModel.getRecommendations().

При ініціалізації функція DashboardScreen приймає аргументами модель viewModel та рядок lang з поточною мовою для локалізації. Змінними заданими у функції є physToday, physYest, mentToday, mentYesterday, dailyStats та recommendations.

Візуально інформація структурується за допомогою компонента ScoreCard, який приймає назву категорії, поточний бал, різницю з попереднім днем та колір. Компонент динамічно відображає порівняльний індикатор (зелений «▲ +» або червоний «▼ ») залежно від знака різниці, що забезпечує наочність трендів. Детальна розбивка метрик виводиться через StatRow, який горизонтально вирівнює мітку показника та його значення, форматоване з одиницями виміру (хвилини, літри, калорії, міліграми, тощо). Список порад рендериться у вигляді вертикальної колекції карток з іконкою-індикатором , генеруючись динамічно та основі аналізу даних.

Переходячи до другого екрану – історії, екран HistoryScreen реалізує концепцію прогресивного розкриття інформації. Спочатку колекція записів з ViewModel групується методом groupBy, формуючи словник, де ключем є дата, а значенням – список подій, збережений у змінній groupedHistory. Для керування станом розгортання днів використовується expandedStates (типу mutableStateMapOf<String, Boolean>()). Аргументами ж функція приймає, як і попередній екран – модель viewModel та рядок lang, що передає поточну мову.

Візуалізація даних побудована на основі LazyColumn, що забезпечує ефективне відображення довгих списків. За допомогою директиви stickyHeader кожен день фіксується у верхній частині екрана під час прокрутки. Хедер є інтерактивним та оновлює відповідний ключ у карті станів, перемикаючи логіку відображення вмісту. Якщо стан запису розгорнуто, всередині items() рендеряться окремі картоки з назвою категорії, деталями та часом створення, що дозволяє користувачеві ефективно навігуватися архівом без перенавантаження UI. Крім того, кожна картока є інтерактивною, при проведені пальцем (swiping) вліво, відображається кнопка для видалення запису.

3.2 Реалізація бізнес-логіки

У візуальний шар на кожному етапі передається viewModel, що контролює усю логіку. Вся обчислювальна функціональність інкапсульована у клас HealthViewModel, який успадковує AndroidViewModel для отримання доступу до контексту застосунку.

Для локального збереження даних використовується клас SharedPreferences, ініціалізований з режимом MODE_PRIVATE, що гарантує доступність даних лише в межах застосунку. Процес серіалізації та десеріалізації складних об’єктів довірено бібліотеці Gson, що дозволяє зберігати цілісні списки моделей у форматі JSON без ручного парсингу полів. Реактивний стан ViewModel представлено двома ключовими властивостями: змінна appLanguage

(mutableStateOf), що зберігає поточну мову інтерфейсу та забезпечує автоматичне оновлення UI для виклику методів зміни локалі, а колекція записів historyLog (mutableStateListOf<JournalEntry>), що зберігає дані у форматі списку об’єктів JournalEntry.

Оголошення змінних використовуючи «mutableState» забезпечує, що в них зберігається стан, що робить її спостережувальною для Compose: будь-яка зміна елементів миттєво інформує візуальний шар про необхідність рекомпозиції.

Ініціалізація об’єкта виконується у блоці init, де відбувається завантаження збереженого журналу з SharedPreferecnes. Використання TypeToken<List<JornalEntry>>() {}.type дозволяє Gson коректно відновити тип списку під час десеріалізації. Блок try-catch забезпечує стійкість до пошкоджених JSON-структур: у разі помилки парсингу, файл очищується шляхом перезапису порожнім масивом, а валідні записи фільтруються за наявністю полів date та time. Метод saveAll() виконує синхронізацію пам’яті з диском, серіалізуючи поточний стан historyLog у JSON-рядок та оновлює обидва ключі (app_lang та history) в спільний налаштуваннях, що гарантує цілісність даних після кожного оновлення.

В свою ж чергу метод deleteEntry гарантує видалення запису, прибираючи заданий запис з historyLog та синхронізуючи його викликом saveAll.

Для уніфікації парсингу числових даних з текстових рядків даних використовується приватний метод extractAllNumbers. Він ініціює регулярний вираз Regex(«\\d+»), який шукає всі послідовності цифр. Метод findAll повертає колекцію збігів, які трансформуються у список цілих чисел через map. Цей підхід дозволяє гнучко обробляти дані незалежно від їхнього текстового форматування в інтерфейсі.

Логіка агрегації даних реалізована у методі geDailyStats, який фільтрує історію за переданою датою та агрегує сумарні значення активності, гідратації, сну, їжі та кофеїну через комбінацію filter та sumOf.

Обчислення фізичного (getPhysicalScore) та ментального (getMentalScore) балів базується на алгоритмі, математичну модель якого я розглянув у попередньому розділі. Програмно, метод спершу розраховує базові показники:

для фізичного стану це інтегральний бал за активність (з урахуванням швидкості руху та відстані), гідратація (нормалізована до 2500 мл) та відновлення через сон; для ментального – якість сну, середній настрій (moodScore) та час усвідомленого відпочинку. Далі застосовується система мультиплікативних штрафів (penaltyPercent), що накопичуються за переїдання/недоїдання (>2500 ккал або <1000 ккал), перетренування (>60-150 хв активності при низькій гідратації), надмірне споживання кофеїну (>250/400 мг) та недосип. Кожен компонент обмежується функцією min() для запобігання експоненційному зростанню штрафу. Формула розрахунку має вигляд baseScore * (1.0 - penaltyPercent), після чого результат нормалізується методом coerceIn(0, 100) та перетворюється на ціле число.

Метод getRecommendations реалізує адаптивну систему підказок. Він аналізує сирі дані статистики та порівнює їх з біологічно обґрунтованими порогами за день (наприклад, сон > 9год, активність > 180хв, кофеїн > 250мг, тривала бездіяльність). Для кожного порушення гомеостазу формується відповідний рядок з трьома мовами перекладу. Якщо порушень не виявлено, але записи присутні, повертається підтверджуюча порада про баланс.

Метод logEntry відповідає за створення нового журнального запису, він автоматично фіксує поточну дату та час у форматі HH:mm через DateTimeFormatter, додає об’єкт типу JournalEntry на початок колекції для зручності перегляду, після чого викликає saveAll для збереження. Функція toggleLanguage реалізує циклічне перемикання локалей за алгоритмом «en → uk → ja → en». Оновлення значення appLanguage запускає реактивний ланцюжок, змушуючи весь UI перерендиритися з новою локалізацією, що підтверджується викликом saveAll.

3.3 Експериментальне тестування алгоритмів та аналіз працездатності

Метою проведення експериментального тестування є верифікація розроблених математичних моделей, перевірка стійкості програмної архітектури до аномальних вхідних даних та оцінка загальної продуктивності мобільного застосунку в умовах, наближених до реальної експлуатації.

Тестування проводилося методом "чорного ящика" (Black-box testing) з фокусом на перевірку граничних станів (Edge cases) нелінійної моделі гомеостазу та кінематичних фільтрів.

Сценарій 1 Тестування математичної моделі гомеостазу з аномальними даними.

Для перевірки коректності роботи системи дисконтування було змодельовано ситуацію екстремального порушення біологічного балансу.

Вхідні дані. Користувач ввів 12 годин сну (гіперсомнія), 4500 спожитих кілокалорій (профіцит) та 450 мг кофеїну (рис. 3.1).

Очікуваний результат. Система повинна застосувати понижуючи коефіцієнти (penaltyPercent) та видати відповідні рекомендації.

Фактичний результат (рис. 3.2). Алгоритми getPhysicalScore та getMentalScore успішно ідентифікували відхилення. Фізична та ментальна оцінки були автоматично знижені. Модуль генерації рекомендацій (Actionable Insights Engine) успішно вивів попередження про перевантаження калоріями, надмірний сон та ризик падіння ментального стану через надлишок кофеїну.

Сценарій 2 Верифікація кінематичного античит-фільтру

Метою цього тесту була перевірка захисту системи від введення нереалістичних фізичних показників.

Вхідні дані (рис. 3.3). Введено тренування з параметрами: дистанція 20 000 метрів (20 км) за 15 хвилин.

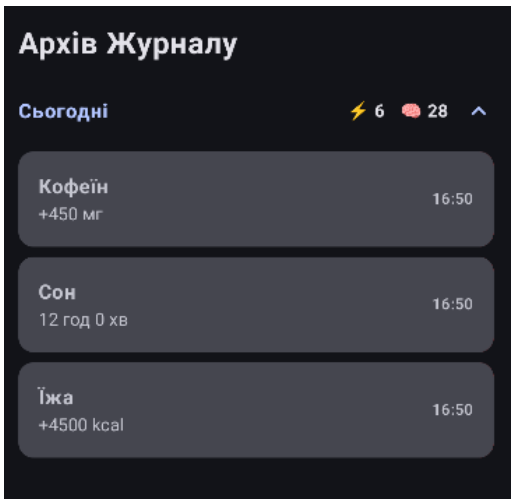
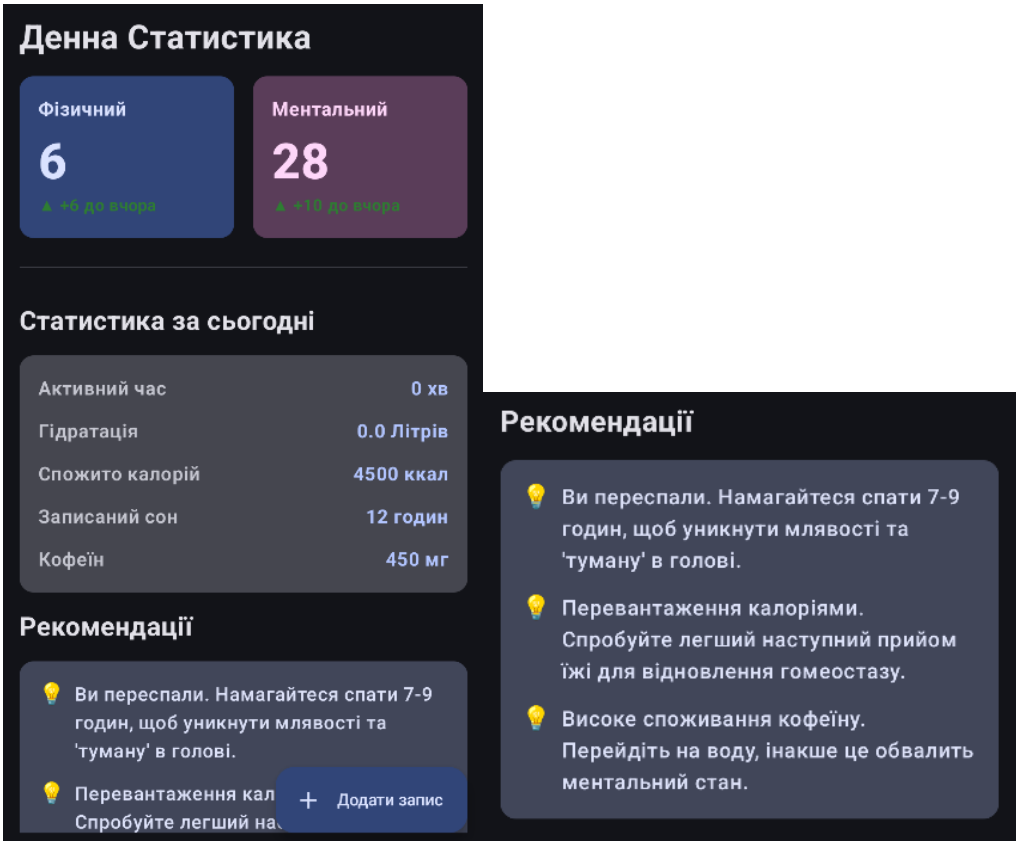


Рисунок 3.1 – Введені дані для тесту



а) денна статистика б) рекомендації

Рисунок 3.2 – Результат тесту

Фактичний результат: (рис. 3.4) Алгоритм розрахував середню швидкість (80 км/год) і класифікував рух як пасивне переміщення на транспортному засобі ($v > 25$ км/год). Замість нарахування аномально великої кількості балів за

дистанцію, система застосувала захисний сценарій: бали були нараховані виключно за базовий час активності.

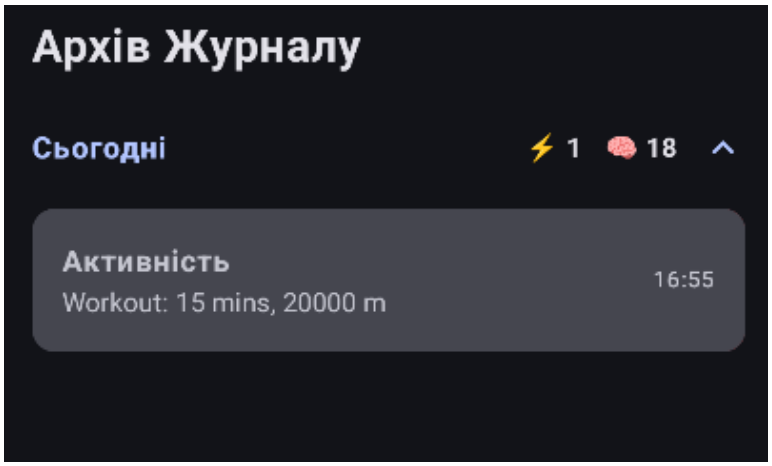


Рисунок 3.3 – Вхідні дані тесту сценарію 2

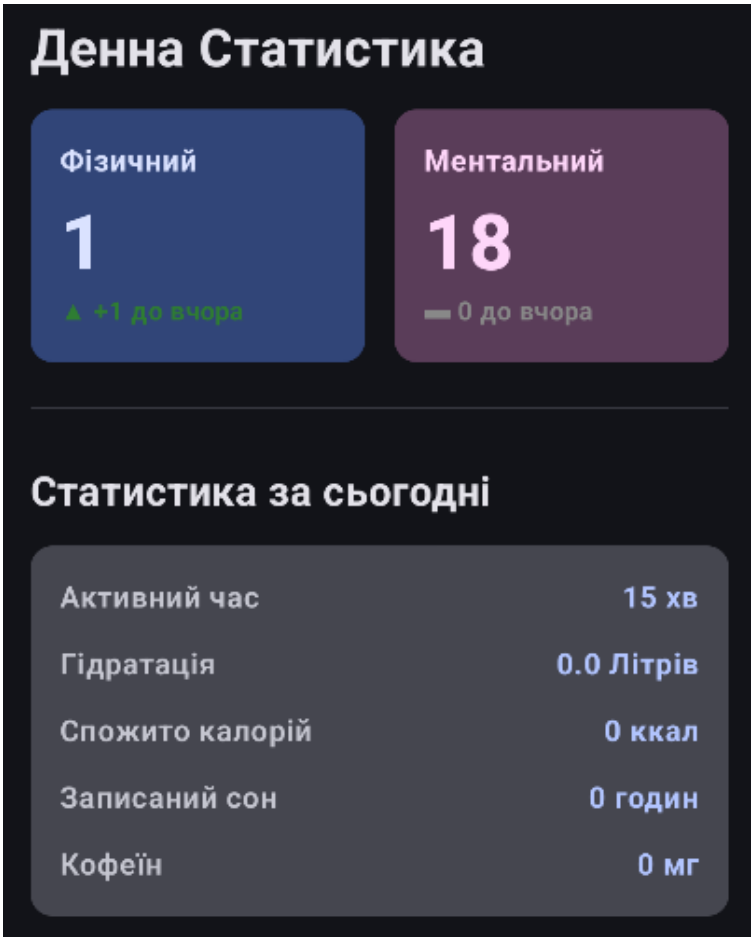


Рисунок 3.4 – Результати тесту сценарію 2

Окрім алгоритмічного тестування, було проведено аналіз стабільності користувацького інтерфейсу та механізмів пам'яті.

Для екрану HistoryScreen було згенеровано понад 100 тестових записів. Використання компонента LazyColumn у Jetpack Compose довело свою ефективність: рендеринг відбувався виключно для видимих на екрані елементів, що забезпечило стабільну продуктивність при швидкому скролінгу. Складна взаємодія з інтерфейсом, зокрема кастомний жест зсуву запису (Swipe-to-Reveal) з використанням Animatable, відпрацьовує плавно, без затримок та просідання кадрів.

Примусове завершення роботи застосунку та його повторний запуск продемонстрували 100% збереження стану. Блок try-catch під час ініціалізації масиву historyLog успішно запобігає виникненню помилки NullPointerException навіть при імітації пошкодженого JSON-файлу, скидаючи архів до безпечного порожнього стану, що гарантує безперебійну роботу програми

ВИСНОВКИ

У кваліфікаційній роботі бакалавра вирішено актуальне науково-прикладне завдання щодо проєктування та програмної реалізації мобільного застосунку для моніторингу біопсихосоціальних показників здоров’я людини з фокусом на абсолютну конфіденційність та усвідомлене відстеження (Mindful Tracking).

За результатами проведеного дослідження та розробки можна зробити наступні висновки.

Дослідження сучасного ринку mHealth-систем виявило критичні вразливості домінуючих комерційних рішень, зокрема ризики несанкціонованої комерціалізації конфіденційних даних через хмарну синхронізацію та явище інтерфейсного перевантаження (Feature Bloat). Доведено, що концепція пасивного моніторингу призводить до когнітивного відчуження користувача. На противагу цьому обґрунтовано необхідність створення мінімалістичного локального застосунку, побудованого за парадигмами «Privacy by Design» та «Mindful Tracking», що повертає суб'єктивний контроль над здоров'ям шляхом усвідомленого ручного введення даних.

Для об'єктивізації оцінювання стану користувача розроблено математичну модель фізіологічного гомеостазу. На відміну від стандартних лінійних систем заохочення, створена модель використовує інвертовані U-подібні криві та систему відсоткового дисконтування (штрафів), яка ефективно реагує як на дефіцит, так і на надлишок біологічних показників (наприклад, гіперсомнію або профіцит калорій). Додатково розроблено та інтегровано кінематичний алгоритм фільтрації вхідних даних активності, який надійно захищає систему від некоректного вводу шляхом аналізу середньої швидкості переміщення.

Створено повноцінний мобільний застосунок для ОС Android з використанням мови Kotlin. Проєктування архітектури виконано за шаблоном MVVM (Model-View-ViewModel) із застосуванням однонаправленого потоку

даних (UDF), що забезпечило високу модульність коду. Графічний інтерфейс реалізовано за допомогою декларативного фреймворку Jetpack Compose, що дозволило впровадити складні UX-патерни, такі як Progressive Disclosure для історії записів та Swipe-to-Reveal для керування даними, забезпечивши при цьому плавний рендеринг без перевантаження системи.

Успішно реалізовано підсистему локальної персистентності на базі механізмів SharedPreferences та двосторонньої серіалізації об'єктів за допомогою бібліотеки Gson. Це дозволило повністю виключити використання зовнішніх мережевих запитів та хмарних серверів, гарантуючи збереження чутливих біометричних даних виключно на фізичному пристрої користувача. Впроваджено модуль прескриптивної аналітики, який генерує персоналізовані текстові рекомендації на основі локально збережених метрик.

Результати тестування розроблених алгоритмів методом моделювання граничних та аномальних станів підтвердили математичну коректність моделей гомеостазу та безпомилкове спрацювання кінематичних фільтрів. Аналіз продуктивності продемонстрував стабільне споживання оперативної пам'яті та безперебійну роботу інтерфейсу при обробці великих масивів історичних даних.

Отже, поставлена мета кваліфікаційної роботи досягнута повністю, а всі визначені завдання виконані. Розроблений мобільний застосунок є готовим до використання програмним продуктом, який пропонує безпечну, конфіденційну та психологічно обґрунтовану альтернативу комерційним фітнес-трекерам.

ПЕРЕЛІК ПОСИЛАНЬ

1. Privacy in consumer wearable technologies : a living systematic review. PubMed Central (PMC). URL : https://pmc.ncbi.nlm.nih.gov/articles/PMC12167361/ (дата звернення : 29.04.2026).

2. The Quantified Self Movement : From Niche Subculture to the Infrastructure of Precision Medicine. Healthcare.Digital. URL : https://www.healthcare.digital/single-post/the-quantified-self-movement-from-niche-subculture-to-the-infrastructure-of-precision-medicine (дата звернення : 06.05.2026).

3. Why Your Fitness Tracker Could Be Exposing Your Health Data to Third Parties. Stayprepared.sg. URL : https://stayprepared.sg/why-your-fitness-tracker-could-be-exposing-your-health-data-to-third-parties/ (дата звернення : 29.04.2026).

4. Decoding U. S. Tort Liability in Healthcare's Black-Box AI Era : Lessons from the European Union. Stanford Law School. URL : https://law.stanford.edu/wp-content/uploads/2024/02/Publish_27-STLR-1-2024_Decoding-U.S.-Tort-Liability-in-Healthcares-Black-Box-AI-Era.pdf (дата звернення : 29.04.2026).

5. Ethical considerations for the use of consumer wearables in health research. ResearchGate. URL : https://www.researchgate.net/publication/368388161_Ethical_considerations_for_the_use_of_consumer_wearables_in_health_research (дата звернення : 28.04.2026).

6. Local-first software : You own your data, in spite of the cloud. Ink & Switch. URL : https://www.inkandswitch.com/essay/local-first/ (дата звернення : 05.05.2026).

7. Types of Self Tracking : Passive vs Manual Tracking. Mark Koester. URL : https://www.markwk.com/types-of-self-tracking.html (дата звернення: 27.04.2026).

8. SleepTight : Low-burden, Self-monitoring Technology for Capturing and Reflecting on Sleep Behaviors. Microsoft Research. URL : https://www.microsoft.com/en-us/research/publication/sleeptight-low-burden-self-

monitoring-technology-capturing-reflecting-sleep-behaviors/ (дата звернення : 05.05.2026).

9. Wearable activity trackers–advanced technology or advanced marketing?. PubMed Central (PMC). URL : https://pmc.ncbi.nlm.nih.gov/articles/PMC9022022/ (дата звернення : 28.04.2026).

10. Privacy Data Ethics of Wearable Digital Health Technology. Brown University. URL : https://cdh.brown.edu/news/2023-05-04/ethics-wearables (дата звернення : 06.05.2026).

11. mHealth Systems Need a Privacy-by-Design Approach : Commentary on “Federated Machine Learning, Privacy-Enhancing Technologies, and Data Protection Laws in Medical Research : Scoping Review”. JMIR. URL : https://www.jmir.org/2023/1/e46700/ (дата звернення : 05.05.2026).

12. Mobile Health. European Data Protection Supervisor. URL : https://www.edps.europa.eu/sites/default/files/publication/15-05-21_mhealth_en_0.pdf (дата звернення : 29.04.2026).

13. The Perceived Utility of Smartphone and Wearable Sensor Data in Digital Self-tracking Technologies for Mental Health. PubMed Central (PMC). URL : https://pmc.ncbi.nlm.nih.gov/articles/PMC11174977/ (дата звернення : 21.04.2026).

14. The Impact of a Health Coaching App on the Subjective Well-Being of Individuals With Multimorbidity : Mixed Methods Study. Journal of Medical Internet Research. URL : https://www.jmir.org/2026/1/e78738 (дата звернення : 16.05.2026).

15. Kotlin Programming Language. Official Documentation. URL : https://kotlinlang.org/docs/home.html (дата звернення : 05.05.2026).

16. Jetpack Compose Basics. Android Developers. URL : https://developer.android.com/jetpack/compose (дата звернення : 05.05.2026).

17. Guide to app architecture. Android Developers. URL : https://developer.android.com/topic/architecture (дата звернення : 05.05.2026).

18. Gson User Guide. Google. URL : https://github.com/google/gson/blob/master/UserGuide.md (дата звернення : 05.05.2026).

19. ViewModel overview. Android Developers. URL : https://developer.android.com/topic/libraries/architecture/viewmodel (дата звернення : 05.05.2026).

20. Вихідний код мобільного застосунку. GitHub. URL : https://github.com/MagnarIUK/mindful-health (дата звернення : 02.06.2026).

ДОДАТОК А

Код програмного забезпечення

```
package com.magnariuk.dyploma

import android.app.Application
import android.content.Context
import androidx.compose.runtime.mutableStateListOf
import androidx.compose.runtime.mutableStateOf
import androidx.lifecycle.AndroidViewModel
import com.google.gson.Gson
import com.google.gson.reflect.TypeToken
import java.time.LocalDate
import java.time.LocalTime
import java.time.format.DateTimeFormatter
import kotlin.math.min
import androidx.core.content.edit

data class DailyStats(val activeMins: Int, val waterMl: Int, val sleepHrs: Int, val foodKcal: Int, val caffeineMg: Int)

data class JournalEntry(val category: LogCategory, val detail: String, val date: String, val time: String)

class HealthViewModel(application: Application) : AndroidViewModel(application) {
    private val prefs = application.getSharedPreferences("HealthPrefs", Context.MODE_PRIVATE)
    private val gson = Gson()

    var appLanguage = mutableStateOf(prefs.getString("app_lang", "en") ?: "en")
    val historyLog = mutableStateListOf<JournalEntry>()

    init {
        val historyJson = prefs.getString("history", "[]")
        val type = object : TypeToken<List<JournalEntry>>() {}.type
        try {
            val savedHistory: List<JournalEntry> = this.gson.fromJson(historyJson, type) ?: emptyList()
            val validHistory = savedHistory.filter { it.date != null && it.time != null }
```

```
            historyLog.addAll(validHistory)
        } catch (e: Exception) {
            prefs.edit { putString("history", "[]") }
        }
    }

    fun deleteEntry(entry: JournalEntry) {
        historyLog.remove(entry)
        saveAll()
    }
    private fun saveAll() {
        prefs.edit {
            putString("app_lang", appLanguage.value)
            putString("history", gson.toJson(historyLog))
        }
    }


    private fun extractAllNumbers(detail: String): List<Int> {
        return Regex("\\d+").findAll(detail).map { it.value.toInt() }.toList()
    }


    fun getDailyStats(targetDate: String = LocalDate.now().toString()): DailyStats {
        val daysLogs = historyLog.filter { it.date == targetDate }

        val aMins = daysLogs.filter { it.category == LogCategory.ACTIVITY }.sumOf {
extractAllNumbers(it.detail).getOrNull(0) ?: 0 }
        val wMl = daysLogs.filter { it.category == LogCategory.WATER }.sumOf {
extractAllNumbers(it.detail).firstOrNull() ?: 0 }
        val sHrs = daysLogs.filter { it.category == LogCategory.SLEEP }.sumOf {
extractAllNumbers(it.detail).firstOrNull() ?: 0 }
        val fKcal = daysLogs.filter { it.category == LogCategory.FOOD }.sumOf {
extractAllNumbers(it.detail).firstOrNull() ?: 0 }
        val cMg = daysLogs.filter { it.category == LogCategory.CAFFEINE }.sumOf {
extractAllNumbers(it.detail).firstOrNull() ?: 0 }

        return DailyStats(aMins, wMl, sHrs, fKcal, cMg)
    }
    fun getPhysicalScore(targetDate: String = LocalDate.now().toString()): Int {
```

```
        val daysLogs = historyLog.filter { it.date == targetDate }

        val activities = daysLogs.filter { it.category == LogCategory.ACTIVITY }
        var activityPoints = 0.0

        for (act in activities) {
            val nums = extractAllNumbers(act.detail)
            val time = nums.getOrNull(0) ?: 0
            val distance = nums.getOrNull(1) ?: 0

            if (time > 0 && distance > 0) {
                val speedKmH = (distance / 1000.0) / (time / 60.0)
                if (speedKmH > 25.0) activityPoints += (time * 0.2)
                else if (speedKmH > 15.0) activityPoints += time + (distance / 250.0)
                else activityPoints += time + (distance / 100.0)
            } else {
                activityPoints += time
            }
        }
        val activityScore = min((activityPoints / 100.0) * 50, 50.0)

        val waterLogs = daysLogs.filter { it.category == LogCategory.WATER }
        val totalWaterMl = waterLogs.sumOf { extractAllNumbers(it.detail).firstOrNull() ?: 0 }
        val waterScore = min((totalWaterMl / 2500.0) * 30, 30.0)

        val sleepHrs = daysLogs.filter { it.category == LogCategory.SLEEP }.sumOf {
extractAllNumbers(it.detail).firstOrNull() ?: 0 }
        val recoveryScore = when {
            sleepHrs in 7..9 -> 20.0
            sleepHrs in 5..6 -> 10.0
            sleepHrs > 9 -> 15.0
            else -> 0.0
        }

        val basePhysicalScore = activityScore + waterScore + recoveryScore

        var penaltyPercent = 0.0

        val foodLogs = daysLogs.filter { it.category == LogCategory.FOOD }
        val totalFoodKcal = foodLogs.sumOf { extractAllNumbers(it.detail).firstOrNull() ?: 0 }
```

```
        if (totalFoodKcal > 2500) penaltyPercent += min(0.4, (totalFoodKcal - 2500) / 5000.0)
        if (totalFoodKcal in 1..1000) penaltyPercent += 0.20
        if (activityPoints > 150) penaltyPercent += min(0.3, (activityPoints - 150) / 300.0)
        if (activityPoints > 60 && totalWaterMl < 1500) penaltyPercent += 0.15
        if (sleepHrs > 9) penaltyPercent += min(0.2, (sleepHrs - 9) * 0.05)

        penaltyPercent = min(0.6, penaltyPercent)

        val totalScore = basePhysicalScore * (1.0 - penaltyPercent)
        return totalScore.toInt().coerceIn(0, 100)
    }

    fun getRecommendations(targetDate: String, lang: String): List<String> {
        val stats = getDailyStats(targetDate)
        val daysLogs = historyLog.filter { it.date == targetDate }
        val activeMins = daysLogs.filter { it.category == LogCategory.ACTIVITY }.sumOf {
extractAllNumbers(it.detail).getOrNull(0) ?: 0 }
        val mindfulMins = daysLogs.filter { it.category == LogCategory.REST || it.category ==
LogCategory.READING }.sumOf { extractAllNumbers(it.detail).firstOrNull() ?: 0 }

        val recs = mutableListOf<String>()

        if (stats.sleepHrs > 9) recs.add(tr(
            "You overslept. Aim for 7-9 hours to avoid brain fog and lethargy.",
            "Ви переспали. Намагайтеся спати 7-9 годин, щоб уникнути млявості та 'туману' в
голові.",
            "寝すぎです。頭ののぼせや倦怠感を防ぐため、7〜9時間を目標にしましょう。",
lang
        ))
        if (stats.sleepHrs in 1..4) recs.add(tr(
            "Severe sleep deprivation! Prioritize rest today to recover your mental score.",
            "Сильний недосип! Пріоритет на відпочинок сьогодні для відновлення ментального
стану.",
            "深刻な睡眠不足です！メンタルスコア回復のため、今日は休息を最優先してくだ
さい。", lang
        ))
        if (stats.foodKcal > 2500) recs.add(tr(
            "Caloric overload. Consider a lighter next meal to restore physical homeostasis.",
```

```
        "Перевантаження калоріями. Спробуйте легший наступний прийом їжі для відновлення гомеостазу.",
        "カロリー過多です。身体のホメオスタシス回復のため、次の食事は軽めにしましょう。", lang
    ))
    if (stats.foodKcal in 1..1000) recs.add(tr(
        "Caloric deficit detected. Fuel your body with nutritious food for energy.",
        "Виявлено дефіцит калорій. Зарядіть тіло поживною їжею для отримання енергії.",
        "カロリー不足が検出されました。エネルギー補給のため、栄養のある食事をとってください。", lang
    ))
    if (activeMins > 150) recs.add(tr(
        "CNS Fatigue warning. If you feel tired, consider making tomorrow a rest day.",
        "Попередження про втому ЦНС. Якщо ви втомлюєтеся, зробіть завтра день відпочинку.",
        "中枢神経疲労の警告。疲労を感じる場合は、明日を休息日にすることを検討してください。。", lang
    ))
    if (activeMins > 60 && stats.waterMl < 1500) recs.add(tr(
        "High activity but low hydration. Drink water immediately to avoid cramping.",
        "Висока активність, але низька гідратація. Негайно випийте води, щоб уникнути спазмів.",
        "運動量に対して水分補給が不足しています。筋肉のけいれんを防ぐため、すぐに水を飲んでください。", lang
    ))
    if (stats.waterMl > 4500) recs.add(tr(
        "Excessive hydration. Reduce water intake to avoid overworking your kidneys.",
        "Надмірна гідратація. Зменште споживання води, щоб не перевантажувати нирки.",
        "水分の過剰摂取です。腎臓への負担を減らすため、飲む量を控えてください。", lang
    ))
    if (stats.caffeineMg > 250) recs.add(tr(
        "High caffeine intake. Switch to water or it will crash your mental score later.",
        "Високе споживання кофеїну. Перейдіть на воду, інакше це обвалить ментальний стан.",
        "カフェインの摂りすぎです。水に変えないと, 後でメンタルスコアが急降下します。", lang
    ))
    if (mindfulMins > 180) recs.add(tr(
```

```
            "Prolonged inactivity detected. Try a short walk to stimulate your brain.",
            "Виявлено тривалу бездіяльність. Спробуйте коротку прогулянку для стимуляції мозку.",
            "長時間の不活動が検出されました。脳を刺激するために軽い散歩をしてみましょう。", lang
        ))

        if (recs.isEmpty() && daysLogs.isNotEmpty()) {
            recs.add(tr(
                "Perfect biological balance! Keep up the great work.",
                "Ідеальний біологічний баланс! Продовжуйте в тому ж дусі.",
                "完璧なバイオリズムです！その調子で頑張りましょう。", lang
            ))
        }

        return recs
    }

    fun getMentalScore(targetDate: String = LocalDate.now().toString()): Int {
        val daysLogs = historyLog.filter { it.date == targetDate }

        val sleepHrs = daysLogs.filter { it.category == LogCategory.SLEEP }.sumOf { extractAllNumbers(it.detail).firstOrNull() ?: 0 }
        val sleepScore = when {
            sleepHrs in 7..9 -> 40.0
            sleepHrs in 5..6 -> 20.0
            sleepHrs > 9 -> 30.0
            else -> 0.0
        }

        val moodLogs = daysLogs.filter { it.category == LogCategory.MOOD }
        val avgMood = if (moodLogs.isNotEmpty()) moodLogs.map { extractAllNumbers(it.detail).firstOrNull() ?: 3 }.average() else 3.0
        val moodScore = (avgMood / 5.0) * 30.0

        val mindfulMins = daysLogs.filter { it.category == LogCategory.REST || it.category == LogCategory.READING }.sumOf { extractAllNumbers(it.detail).firstOrNull() ?: 0 }
        val mindfulScore = min((mindfulMins / 60.0) * 30, 30.0)

        val baseMentalScore = sleepScore + moodScore + mindfulScore
```

```
        var penaltyPercent = 0.0

        val caffeineMg = daysLogs.filter { it.category == LogCategory.CAFFEINE }.sumOf {
extractAllNumbers(it.detail).firstOrNull() ?: 0 }
        if (caffeineMg > 400) penaltyPercent += 0.25
        else if (caffeineMg > 250) penaltyPercent += 0.10

        if (sleepHrs in 1..4) penaltyPercent += 0.35
        if (sleepHrs > 9) penaltyPercent += 0.15

        val activities = daysLogs.filter { it.category == LogCategory.ACTIVITY }
        val activeMins = activities.sumOf { extractAllNumbers(it.detail).getOrNull(0) ?: 0 }
        if (activeMins > 180) penaltyPercent += 0.20
        if (mindfulMins > 180) penaltyPercent += 0.15

        penaltyPercent = min(0.65, penaltyPercent)

        val totalScore = baseMentalScore * (1.0 - penaltyPercent)
        return totalScore.toInt().coerceIn(0, 100)
    }


    fun logEntry(category: LogCategory, detail: String) {
        val date = LocalDate.now().toString()
        val time = LocalTime.now().format(DateTimeFormatter.ofPattern("HH:mm"))
        historyLog.add(0, JournalEntry(category, detail, date, time))
        saveAll()
    }

    fun toggleLanguage() {
        appLanguage.value = when (appLanguage.value) {
            "en" -> "uk"
            "uk" -> "ja"
            else -> "en"
        }
        saveAll()
    }}

package com.magnariuk.dyploma
```

```
import androidx.compose.animation.AnimatedContent
import androidx.compose.animation.animateColorAsState
import androidx.compose.animation.core.Animatable
import androidx.compose.foundation.ExperimentalFoundationApi
import androidx.compose.foundation.background
import androidx.compose.foundation.clickable
import androidx.compose.foundation.gestures.detectHorizontalDragGestures
import androidx.compose.foundation.horizontalScroll
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.items
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.filled.*
import androidx.compose.material3.*
import androidx.compose.material3.HorizontalDivider
import androidx.compose.material3.rememberSwipeToDismissBoxState
import androidx.compose.runtime.*
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.input.pointer.pointerInput
import androidx.compose.ui.platform.LocalDensity
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.text.input.KeyboardType
import androidx.compose.ui.unit.IntOffset
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import androidx.lifecycle.viewmodel.compose.viewModel
import kotlinx.coroutines.launch
import java.time.LocalDate

fun tr(en: String, uk: String, ja: String, lang: String): String = when (lang) {
    "uk" -> uk
    "ja" -> ja
    else -> en
}
```

```
enum class LogCategory { ACTIVITY, FOOD, WATER, SLEEP, MOOD, CAFFEINE,
READING, REST, DIARY }

@Composable
fun LogCategory.localizedName(lang: String): String = when (this) {
    LogCategory.ACTIVITY -> tr("Activity", "Активність", "アクティビティ", lang)
    LogCategory.FOOD -> tr("Food", "Їжа", "食事", lang)
    LogCategory.WATER -> tr("Water", "Вода", "水分", lang)
    LogCategory.SLEEP -> tr("Sleep", "Сон", "睡眠", lang)
    LogCategory.MOOD -> tr("Mood", "Настрій", "気分", lang)
    LogCategory.CAFFEINE -> tr("Caffeine", "Кофеїн", "カフェイン", lang)
    LogCategory.READING -> tr("Reading", "Читання", "読書", lang)
    LogCategory.REST -> tr("Rest/Med.", "Відпочинок", "休憩/瞑想", lang)
    LogCategory.DIARY -> tr("Diary", "Щоденник", "日記", lang)
}

@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun MainHealthApp(viewModel: HealthViewModel = viewModel()) {
    val lang = viewModel.appLanguage.value
    var selectedTab by remember { mutableStateOf(0) }
    var showOmniDialog by remember { mutableStateOf(false) }

    Scaffold(
        topBar = {
            TopAppBar(
                title = { Text(tr("Mindful Health", "Mindful Health", "マインドフル・ヘルス", lang),
fontWeight = FontWeight.Bold) },
                colors = TopAppBarDefaults.topAppBarColors(containerColor =
MaterialTheme.colorScheme.primaryContainer),
                actions = {
                    TextButton(onClick = { viewModel.toggleLanguage() }) {
                        Text(when(lang) {
                            "uk" -> "UA UK"
                            "ja" -> "JP JA"
                            else -> "GB EN"
                        }, fontSize = 18.sp)
                    }
                }
```

```
            )
        },
        floatingActionButton = {
            ExtendedFloatingActionButton(
                onClick = { showOmniDialog = true },
                icon = { Icon(Icons.Default.Add, "Log") },
                text = { Text(tr("Log Entry", "Додати запис", "記録を追加", lang)) }
            )
        },
        bottomBar = {
            NavigationBar {
                NavigationBarItem(selected = selectedTab == 0, onClick = { selectedTab = 0 }, icon = {
Icon(Icons.Default.Dashboard, "Dashboard") }, label = { Text(tr("Dashboard", "Головна", "ダッシュボード", lang)) })
                NavigationBarItem(selected = selectedTab == 1, onClick = { selectedTab = 1 }, icon = {
Icon(Icons.Default.History, "History") }, label = { Text(tr("History", "Історія", "履歴", lang)) })
}
        }
    ) { padding ->
        if (showOmniDialog) { OmniLogDialog(onDismiss = { showOmniDialog = false },
viewModel = viewModel, lang = lang) }
        Box(modifier = Modifier.padding(padding)) {
            if (selectedTab == 0) DashboardScreen(viewModel, lang) else HistoryScreen(viewModel,
lang)
        }
    }
}


@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun OmniLogDialog(onDismiss: () -> Unit, viewModel: HealthViewModel, lang: String) {
    var selectedCategory by remember { mutableStateOf(LogCategory.ACTIVITY) }

    var actType by remember { mutableStateOf("") }
    var actTime by remember { mutableStateOf("") }
    var actDist by remember { mutableStateOf("") }
    var waterMl by remember { mutableStateOf("") }
    var sleepHrs by remember { mutableStateOf("") }
    var sleepMins by remember { mutableStateOf("") }
```

```
    var foodKcal by remember { mutableStateOf("") }
    var moodScore by remember { mutableFloatStateOf(3f) }
    var caffeineMg by remember { mutableStateOf("") }
    var focusMins by remember { mutableStateOf("") }
    var diaryText by remember { mutableStateOf("") }

    fun resetFields(){
        actType = ""; actTime = ""; actDist = ""; waterMl = ""; sleepHrs = ""; sleepMins = "";
foodKcal = ""; caffeineMg = ""; focusMins = ""; diaryText = ""
        moodScore = 3f
    }

    AlertDialog(
        onDismissRequest = onDismiss,
        title = { Text(tr("Log to Journal", "Додати до журналу", "日誌に記録", lang)) },
        text = {
            Column(verticalArrangement = Arrangement.spacedBy(16.dp), modifier =
Modifier.fillMaxWidth()) {
                Row(
                    modifier = Modifier.fillMaxWidth().horizontalScroll(rememberScrollState()),
                    horizontalArrangement = Arrangement.spacedBy(8.dp),
                    verticalAlignment = Alignment.CenterVertically
                ) {
                    LogCategory.entries.forEach { category ->
                        FilterChip(
                            selected = selectedCategory == category,
                            onClick = {
                                selectedCategory = category
                                resetFields()
                            },
                            label = { Text(text = category.localizedName(lang), fontWeight = if
(selectedCategory == category) FontWeight.Bold else FontWeight.Normal) },
                            colors = FilterChipDefaults.filterChipColors(selectedContainerColor =
MaterialTheme.colorScheme.primaryContainer, selectedLabelColor =
MaterialTheme.colorScheme.onPrimaryContainer),
                            modifier = Modifier.defaultMinSize(minHeight = 40.dp)
                        )
                    }
                }
```

```
            AnimatedContent(targetState = selectedCategory, label = "dynamic_inputs") { category ->
                Column(verticalArrangement = Arrangement.spacedBy(12.dp)) {
                    when (category) {
                        LogCategory.ACTIVITY -> {
                            OutlinedTextField(value = actType, onValueChange = { actType = it }, label = { Text(tr("Workout Type", "Тип тренування", "運動の種類", lang)) }, modifier = Modifier.fillMaxWidth())
                            Row(horizontalArrangement = Arrangement.spacedBy(8.dp)) {
                                OutlinedTextField(value = actTime, onValueChange = { if(it.all{ c->c.isDigit()}) actTime = it }, label = { Text(tr("Time (mins)", "Час (хв)", "時間 (分)", lang)) }, keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Number), modifier = Modifier.weight(1f))
                                OutlinedTextField(value = actDist, onValueChange = { if(it.all{ c->c.isDigit()}) actDist = it }, label = { Text(tr("Distance (m)", "Відстань (м)", "距離 (m)", lang)) }, keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Number), modifier = Modifier.weight(1f))
                            }
                        }
                        LogCategory.WATER -> {
                            OutlinedTextField(value = waterMl, onValueChange = { if(it.all{ c->c.isDigit()}) waterMl = it }, label = { Text(tr("Volume (Ml)", "Об'єм (мл)", "量 (ml)", lang)) }, keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Number), modifier = Modifier.fillMaxWidth())
                        }
                        LogCategory.SLEEP -> {
                            Row(horizontalArrangement = Arrangement.spacedBy(8.dp)) {
                                OutlinedTextField(value = sleepHrs, onValueChange = { if(it.all{ c->c.isDigit()}) sleepHrs = it }, label = { Text(tr("Hours", "Години", "時間", lang)) }, keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Number), modifier = Modifier.weight(1f))
                                OutlinedTextField(value = sleepMins, onValueChange = { if(it.all{ c->c.isDigit()}) sleepMins = it }, label = { Text(tr("Minutes", "Хвилини", "分", lang)) }, keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Number), modifier = Modifier.weight(1f))
                            }
                        }
                        LogCategory.FOOD -> {
```

```
                    OutlinedTextField(value = foodKcal, onValueChange = { if(it.all{ c->c.isDigit()}) foodKcal = it }, label = { Text(tr("Calories (kcal)", "Калорії (ккал)", "カロリー (kcal)", lang)) }, keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Number), modifier = Modifier.fillMaxWidth())
                }
                LogCategory.MOOD -> {
                    Column {
                        Text(tr("Rate your mood: ${moodScore.toInt()}/5", "Оцініть настрій: ${moodScore.toInt()}/5", "気分を評価: ${moodScore.toInt()}/5", lang))
                        Slider(value = moodScore, onValueChange = { moodScore = it }, valueRange = 1f..5f, steps = 3)
                    }
                }
                LogCategory.CAFFEINE -> {
                    OutlinedTextField(value = caffeineMg, onValueChange = { if(it.all{ c->c.isDigit()}) caffeineMg = it }, label = { Text(tr("Amount (mg)", "Кількість (мг)", "量 (mg)", lang)) }, keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Number), modifier = Modifier.fillMaxWidth())
                }
                LogCategory.READING, LogCategory.REST -> {
                    OutlinedTextField(value = focusMins, onValueChange = { if(it.all{ c->c.isDigit()}) focusMins = it }, label = { Text(tr("Duration (mins)", "Тривалість (хв)", "時間 (分)", lang)) }, keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Number), modifier = Modifier.fillMaxWidth())
                }
                LogCategory.DIARY -> {
                    OutlinedTextField(value = diaryText, onValueChange = { diaryText = it }, label = { Text(tr("What's on your mind?", "Що на думці?", "いまどうしてる？", lang)) }, modifier = Modifier.fillMaxWidth().height(120.dp), maxLines = 5)
                }
            }
        }
    }
},
confirmButton = {
    Button(
        onClick = {
            when (selectedCategory) {
                LogCategory.ACTIVITY -> {
```

```
                    val t = actTime.toIntOrNull() ?: 0
                    val d = actDist.toIntOrNull() ?: 0
                    if (t > 0 || d > 0) viewModel.logEntry(selectedCategory, "${if(actType.isBlank())
tr("Workout", "Тренування", "運動", lang) else actType}: $t ${tr("mins", "хв", "分", lang)}, $d
${tr("m", "м", "m", lang)}")
                }
                LogCategory.WATER -> { val w = waterMl.toIntOrNull() ?: 0; if (w > 0)
viewModel.logEntry(selectedCategory, "+$w ${tr("ml", "мл", "ml", lang)}") }
                LogCategory.SLEEP -> {
                    val h = sleepHrs.toIntOrNull() ?: 0
                    val m = sleepMins.toIntOrNull() ?: 0
                    if (h > 0 || m > 0) viewModel.logEntry(selectedCategory, "$h ${tr("h", "год", "時
間", lang)} $m ${tr("m", "хв", "分", lang)}")
                }
                LogCategory.FOOD -> { val f = foodKcal.toIntOrNull() ?: 0; if (f > 0)
viewModel.logEntry(selectedCategory, "+$f ${tr("kcal", "ккал", "kcal", lang)}") }
                LogCategory.MOOD -> { viewModel.logEntry(selectedCategory, "${tr("Score",
"Оцінка", "評価", lang)}: ${moodScore.toInt()}/5") }
                LogCategory.CAFFEINE -> { val c = caffeineMg.toIntOrNull() ?: 0; if (c > 0)
viewModel.logEntry(selectedCategory, "+$c ${tr("mg", "мг", "mg", lang)}") }
                LogCategory.READING, LogCategory.REST -> { val fm = focusMins.toIntOrNull()
?: 0; if (fm > 0) viewModel.logEntry(selectedCategory, "$fm ${tr("mins", "хв", "分", lang)}") }
                LogCategory.DIARY -> { if (diaryText.isNotBlank())
viewModel.logEntry(selectedCategory, diaryText) }
            }
            onDismiss()
        }
      ) { Text(tr("Save", "Зберегти", "保存", lang)) }
    },
    dismissButton = { TextButton(onClick = onDismiss) { Text(tr("Cancel", "Скасувати", "キャ
ンセル", lang)) } }
  )
}

@Composable
fun DashboardScreen(viewModel: HealthViewModel, lang: String) {
  val today = LocalDate.now().toString()
  val yesterday = LocalDate.now().minusDays(1).toString()
```

```
    val physToday = viewModel.getPhysicalScore(today)
    val physYest = viewModel.getPhysicalScore(yesterday)
    val mentToday = viewModel.getMentalScore(today)
    val mentYest = viewModel.getMentalScore(yesterday)

    val dailyStats = viewModel.getDailyStats(today)
    val recommendations = viewModel.getRecommendations(today, lang)

    Column(
        modifier = Modifier.fillMaxSize().padding(16.dp).verticalScroll(rememberScrollState()),
        verticalArrangement = Arrangement.spacedBy(16.dp)
    ) {
        Text(tr("Daily Stats", "Денна Статистика", "今日の統計", lang), fontSize = 28.sp,
fontWeight = FontWeight.Bold)

        Row(horizontalArrangement = Arrangement.spacedBy(16.dp), modifier =
Modifier.fillMaxWidth()) {
            Box(modifier = Modifier.weight(1f)) {
                ScoreCard(title = tr("Physical", "Фізичний", "身体的", lang), score = physToday, diff =
physToday - physYest, color = MaterialTheme.colorScheme.primaryContainer, lang = lang)
            }
            Box(modifier = Modifier.weight(1f)) {
                ScoreCard(title = tr("Mental", "Ментальний", "精神的", lang), score = mentToday, diff
= mentToday - mentYest, color = MaterialTheme.colorScheme.tertiaryContainer, lang = lang)
            }
        }

        HorizontalDivider(modifier = Modifier.padding(vertical = 8.dp))

        Text(tr("Today's Breakdown", "Статистика за сьогодні", "今日の詳細", lang), fontSize =
22.sp, fontWeight = FontWeight.Bold, modifier = Modifier.padding(top = 8.dp))

        Card(modifier = Modifier.fillMaxWidth(), colors = CardDefaults.cardColors(containerColor =
MaterialTheme.colorScheme.surfaceVariant)) {
            Column(modifier = Modifier.padding(16.dp), verticalArrangement =
Arrangement.spacedBy(12.dp)) {
                StatRow(label = tr("Active Time", "Активний час", "アクティブ時間", lang), value =
"${dailyStats.activeMins} ${tr("mins", "хв", "分", lang)}")
```

```
            StatRow(label = tr("Hydration", "Гідратація", "水分摂取量", lang), value =
"${dailyStats.waterMl / 1000f} ${tr("Liters", "Літрів", "リットル", lang)}")
            StatRow(label = tr("Calories Consumed", "Спожито калорій", "摂取カロリー", lang),
value = "${dailyStats.foodKcal} ${tr("kcal", "ккал", "kcal", lang)}")
            StatRow(label = tr("Sleep Logged", "Записаний сон", "睡眠時間", lang), value =
"${dailyStats.sleepHrs} ${tr("hours", "годин", "時間", lang)}")
            StatRow(label = tr("Caffeine Intake", "Кофеїн", "カフェイン量", lang), value =
"${dailyStats.caffeineMg} ${tr("mg", "мг", "mg", lang)}")
        }
    }

    if (recommendations.isNotEmpty()) {
        Text(tr("Insights", "Рекомендації", "アドバイス", lang), fontSize = 22.sp, fontWeight =
FontWeight.Bold)
        Card(modifier = Modifier.fillMaxWidth(), colors = CardDefaults.cardColors(containerColor
= MaterialTheme.colorScheme.secondaryContainer)) {
            Column(modifier = Modifier.padding(16.dp), verticalArrangement =
Arrangement.spacedBy(12.dp)) {
                recommendations.forEach { insight ->
                    Row(verticalAlignment = Alignment.Top) {
                        Text("💡", fontSize = 18.sp, modifier = Modifier.padding(end = 8.dp))
                        Text(insight, fontWeight = FontWeight.Medium)
                    }
                }
            }
        }
    }
    Spacer(modifier = Modifier.height(80.dp))
  }
}

@Composable
fun ScoreCard(title: String, score: Int, diff: Int, color: Color, lang: String) {
  Card(modifier = Modifier.fillMaxWidth(), colors = CardDefaults.cardColors(containerColor =
color)) {
    Column(modifier = Modifier.padding(16.dp)) {
        Text(title, fontWeight = FontWeight.Bold, fontSize = 16.sp)
        Text("$score", fontSize = 42.sp, fontWeight = FontWeight.ExtraBold, modifier =
Modifier.padding(top = 8.dp))
```

```
            if (diff != 0) {
                val sign = if (diff > 0) "▲ +" else "▼ "
                val diffColor = if (diff > 0) Color(0xFF2E7D32) else Color(0xFFC62828)
                Text("$sign$diff ${tr("vs yst.", "до вчора", "前日比", lang)}", color = diffColor,
fontWeight = FontWeight.Bold, fontSize = 14.sp)
            } else {
                Text("▬ 0 ${tr("vs yst.", "до вчора", "前日比", lang)}", color = Color.Gray, fontWeight
= FontWeight.Bold, fontSize = 14.sp)
            }
        }
    }
}
@Composable
fun StatRow(label: String, value: String) {
    Row(modifier = Modifier.fillMaxWidth(), horizontalArrangement =
Arrangement.SpaceBetween, verticalAlignment = Alignment.CenterVertically) {
        Row(verticalAlignment = Alignment.CenterVertically) {
            Text(label, fontWeight = FontWeight.SemiBold)
        }
        Text(value, fontWeight = FontWeight.Bold, color = MaterialTheme.colorScheme.primary)
    }
}

@OptIn(ExperimentalFoundationApi::class)
@Composable
fun HistoryScreen(viewModel: HealthViewModel, lang: String) {
    val groupedHistory = viewModel.historyLog.groupBy { it.date }
    val expandedStates = remember { mutableStateMapOf<String, Boolean>() }

    Column(modifier = Modifier.fillMaxSize().padding(16.dp)) {
        Text(tr("Journal Archive", "Архів Журналу", "日誌アーカイブ", lang), fontSize = 24.sp,
fontWeight = FontWeight.Bold, modifier = Modifier.padding(bottom = 16.dp))

        if (groupedHistory.isEmpty()) {
            Text(tr("No entries yet.", "Ще немає записів.", "まだ記録がありません。", lang))
        } else {
            LazyColumn(verticalArrangement = Arrangement.spacedBy(8.dp)) {
                groupedHistory.forEach { (date, entries) ->
                    val isExpanded = expandedStates[date] ?: true
```

```
                stickyHeader {
                    Surface(
                        modifier = Modifier
                            .fillMaxWidth()
                            .padding(vertical = 4.dp)
                            .clickable { expandedStates[date] = !isExpanded },
                        color = MaterialTheme.colorScheme.background
                    ) {
                        Row(
                            modifier = Modifier.fillMaxWidth().padding(vertical = 8.dp),
                            horizontalArrangement = Arrangement.SpaceBetween,
                            verticalAlignment = Alignment.CenterVertically
                        ) {
                            Text(
                                text = if (date == LocalDate.now().toString()) tr("Today", "Сьогодні", "今日
", lang) else date,
                                fontWeight = FontWeight.ExtraBold, color =
MaterialTheme.colorScheme.primary, fontSize = 16.sp
                            )
                            Row(verticalAlignment = Alignment.CenterVertically, horizontalArrangement
= Arrangement.spacedBy(12.dp)) {
                                Text("⚡ ${viewModel.getPhysicalScore(date)}", fontWeight =
FontWeight.Bold, fontSize = 14.sp)
                                Text("🧠 ${viewModel.getMentalScore(date)}", fontWeight =
FontWeight.Bold, fontSize = 14.sp)
                                Icon(imageVector = if (isExpanded) Icons.Default.KeyboardArrowUp else
Icons.Default.KeyboardArrowDown, contentDescription = null, tint =
MaterialTheme.colorScheme.primary)
                            }
                        }
                    }
                }

                if (isExpanded) {
                    items(items = entries, key = { it.hashCode() }) { entry ->

                        val density = LocalDensity.current
                        val buttonWidthDp = 80.dp
                        val buttonWidthPx = with(density) { buttonWidthDp.toPx() }
```

```
                val offsetX = remember { Animatable(0f) }
                val scope = rememberCoroutineScope()

                Box(
                    modifier = Modifier
                        .fillMaxWidth()
                        .pointerInput(Unit) {
                            detectHorizontalDragGestures(
                                onDragEnd = {
                                    scope.launch {
                                        if (offsetX.value < -buttonWidthPx / 2) {
                                            offsetX.animateTo(-buttonWidthPx)
                                        } else {
                                            offsetX.animateTo(0f)
                                        }
                                    }
                                },
                                onHorizontalDrag = { change, dragAmount ->
                                    change.consume()
                                    scope.launch {
                                        val newOffset = (offsetX.value + dragAmount).coerceIn(-
buttonWidthPx, 0f)
                                        offsetX.snapTo(newOffset)
                                    }
                                }
                            )
                        }
                ) {
                    Box(
                        modifier = Modifier.matchParentSize(),
                        contentAlignment = Alignment.CenterEnd
                    ) {
                        Box(
                            modifier = Modifier
                                .width(buttonWidthDp)
                                .fillMaxHeight()
                                .background(Color(0xFFC62828), shape = CardDefaults.shape)
                                .clickable {
                                    viewModel.deleteEntry(entry)
                                },
```

```
                        contentAlignment = Alignment.Center
                    ) {
                        Icon(
                            imageVector = Icons.Default.Delete,
                            contentDescription = "Delete Entry",
                            tint = Color.White
                        )
                    }
                }

                Card(
                    modifier = Modifier
                        .offset { IntOffset(offsetX.value.toInt(), 0) }
                        .fillMaxWidth(),
                    colors = CardDefaults.cardColors(containerColor =
MaterialTheme.colorScheme.surfaceVariant)
                ) {
                    Row(modifier = Modifier.fillMaxWidth().padding(16.dp),
horizontalArrangement = Arrangement.SpaceBetween, verticalAlignment =
Alignment.CenterVertically) {
                        Column {
                            Text(entry.category.localizedName(lang), fontWeight =
FontWeight.Bold)
                            Text(entry.detail, style = MaterialTheme.typography.bodyMedium)
                        }
                        Text(entry.time, style = MaterialTheme.typography.labelMedium)
                    }
                }
            }
        }
    }
  }
}
```