Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка методів і програмних засобів мобільної системи на платформі Android для підтримки здорового способу харчування»

Виконав: студент II курсу
групи 1ПІ-24м
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Бабійчук І.С.
(прізвище та ініціали)
Керівник: к.т.н., доц. каф. ПЗ Войтко В.В.
(прізвище та ініціали)
«10» грудня 2025 р.
Опонент: к.т.н., доц. каф. ОТ Колесник І.С.
(прізвище та ініціали)
«10» грудня 2025 р.

Допущено до захисту
Завідувач кафедри ПЗ
д.т.н., проф. Романюк О.Н.
«10» грудня 2025 р.

ВНТУ – 2025

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти ІІ-й (магістерський)
Галузь знань 12 – Інформаційні технології
Спеціальність 121 – Інженерія програмного забезпечення
Освітньо-професійна програма – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
д.т.н., професор Романюк О.Н.

« 25» вересня 2025 р.

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Бабійчуку Івану Сергійовичу

1. Тема роботи – Розробка методів і програмних засобів мобільної системи на платформі Android для підтримки здорового способу харчування.
Керівник роботи: Войтко Вікторія Володимирівна, к.т.н., доц. кафедри ПЗ, затверджені наказом вищого навчального закладу від « 24 » вересня 2025 р. №313.

2. Строк подання студентом роботи
10 грудня 2025 р.

3. Вихідні дані до роботи: технологічний стек мобільного застосунку – Kotlin, архітектура MVVM, бібліотеки Jetpack, Coroutines, Hilt; системні обмеження – мінімальна версія Android, апаратні вимоги, обмеження мережевих протоколів, залежності сторонніх SDK; вихідні дані – стабільна збірка мобільного застосунку, що коректно відображає та обробляє дані, працює відповідно до бізнес-логіки й забезпечує передбачувану поведінку під час нестабільних мережевих умов.
4. Зміст текстової частини: аналіз сучасних мобільних систем контролю харчування; порівняння аналогів систем трекінгу калорій і водного балансу;

дослідження методів визначення добової норми калорій та води; аналіз відкритих харчових баз даних; формування постановки задачі; розробка архітектури мобільної системи; формування методу адаптивного розрахунку добової норми калорій; розробка методу прогнозування водного балансу; моделювання структури даних користувача та харчових продуктів; побудова загального алгоритму роботи системи; проєктування модулів профілю, роботи з харчовими даними, підбору денного раціону та трекінгу водного балансу; інтеграція з OpenFoodFacts; розробка інтерфейсу мобільного застосунку; тестування модулів системи; аналіз методів тестування програмного забезпечення; формування економічного обґрунтування розробки; розрахунок витрат на виконання науково-дослідної роботи; оцінка ефективності потенційної комерціалізації.

5. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Войтко В.В., к.т.н., доцент кафедри ПЗ	26.09.25	09.12.25
5	Адлер О.О., к.т.н., доцент кафедри ЕПВМ	22.11.2025	30.11.2025

6. Дата видачі завдання 26 вересня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз стану систем підтримки здорового харчування	26.09.2025 - 05.10.2025	вип
2	Розробка методів, моделей та алгоритмів системи	06.10.2025 - 22.10.2025	вип
3	Розробка програмних засобів системи	23.10.2025 - 06.11.2025	вип
4	Тестування системи	07.11.2025 - 21.11.2025	вик
5	Економічна частина	22.11.2025 - 30.11.2025	вип
6	Оформлення матеріалів до захисту МКР	01.12.25 – 09.12.25	вип

Студент _____ **Бабійчук І.С.**
(підпис) (прізвище та ініціали)

Керівник магістерської кваліфікаційної роботи _____ **Войтко В.В.**
(підпис) (прізвище та ініціали)

Анотація

УДК 004.912.26

Бабійчук І.С. Розробка методів і засобів мобільної системи на платформі Android для підтримки здорового способу харчування. Магістерська кваліфікаційна робота зі спеціальності 121 – Інженерія програмного забезпечення, освітня програма – Інженерія програмного забезпечення. Вінниця: ВНТУ, 2025. 142 с.

На укр. мові. Бібліогр.: 46 назв; рисунків: 28; таблиць: 14.

У магістерській кваліфікаційній роботі подано результати дослідження методів та засобів створення мобільної системи підтримки здорового способу харчування. Обґрунтовано підхід до адаптивного визначення добової норми калорій на основі формули Mifflin–St Jeor із поведінковими коефіцієнтами та запропоновано метод прогнозування водного балансу користувача.

Магістерська кваліфікаційна робота містить аналіз відомих підходів до персоналізованого харчування, методів формування добового раціону та організації трекінгу харчової активності.

Розроблено алгоритм автоматизованого підбору денного раціону з урахуванням фізіологічних параметрів і індивідуальних обмежень користувача. Реалізовано мобільний застосунок на платформі Android із використанням відкритих джерел харчових даних, функціоналом сканування штрих–кодів продуктів та засобами ведення обліку водного балансу. Система забезпечує локальне збереження персональних даних і підтримує адаптивне оновлення профілю користувача.

Ключові слова: здорове харчування, Android, мобільний застосунок, калорійність, водний баланс, добовий раціон.

Abstract

Babiichuk. I.S. Development of Methods and Tools of a Mobile Android–Based System for Supporting Healthy Nutrition. Master's thesis in specialty 121 – Software Engineering, educational program – Software Engineering. Vinnytsia: Vinnytsia National Technical University, 2025. 142 p.

In Ukrainian. References: 46 titles; figures: 28; tables: 14.

The Master's thesis presents the results of research on methods and tools for developing a mobile system aimed at supporting healthy nutrition. An adaptive approach to estimating daily caloric needs based on the Mifflin–St Jeor formula enhanced with behavioral coefficients is substantiated, and a method for predicting the user's water balance is proposed.

The thesis includes an analysis of existing approaches to personalized nutrition, meal–plan generation, and nutritional activity tracking.

An algorithm for automated daily meal–plan selection that considers physiological parameters and individual user constraints has been developed. A mobile Android application was implemented using open food–data sources, barcode–scanning functionality, and water–intake tracking tools. The system ensures local storage of personal data and supports adaptive user–profile updates.

Keywords: healthy nutrition, Android, mobile application, caloric intake, water balance, meal plan.

ЗМІСТ

ВСТУП 4
1 АНАЛІЗ СТАНУ СИСТЕМ ПІДТРИМКИ ЗДОРОВОГО ХАРЧУВАННЯ 8
1.1 Аналіз сучасних мобільних систем для контролю харчування 8
1.2 Аналіз аналогів мобільних застосунків для трекінгу калорій та водного балансу 9
1.3 Аналіз підходів до визначення добової норми калорій та водного балансу 15
1.4 Аналіз відкритих джерел харчових даних 18
1.5 Постановка задач дослідження 20
1.6 Висновки 22
2 РОЗРОБКА МЕТОДІВ, МОДЕЛЕЙ ТА АЛГОРИТМІВ СИСТЕМИ 23
2.1 Розробка архітектури мобільної системи 23
2.2 Розробка методу адаптивного визначення добової норми калорій 27
2.3 Розробка методу прогнозування водного балансу 34
2.4 Розробка моделі даних користувача та продуктів харчування 41
2.5 Розробка загального алгоритму роботи системи 44
2.6 Висновки 48
3 РОЗРОБКА ПРОГРАМНИХ ЗАСОБІВ СИСТЕМИ 49
3.1 Варіантний аналіз і обгрунтування вибору технологій та інструментів розробки 49
3.2 Розробка модуля управління профілем користувача 61
3.3 Розробка модуля роботи з харчовими даними та інтеграції з OpenFoodFacts 67
3.4 Розробка модуля підбору денного раціону 71
3.5 Розробка модуля трекінгу водного балансу 78
3.6 Розробка інтерфейсу мобільного застосунку 80
3.7 Висновки 90
4 ТЕСТУВАННЯ СИСТЕМИ 92
4.1 Аналіз методів тестування програмного забезпечення 92
4.2 Тестування функціональних модулів мобільної системи 95
4.3 Висновки 109

5 ЕКОНОМІЧНА ЧАСТИНА 111
5.1 Комерційний аудит науково–технічної розробки 112
5.2 Розрахунок витрат на проведення науково–дослідної роботи 116
5.2.1 Витрати на оплату праці 116
5.2.2 Відрахування на соціальні заходи 119
5.2.3 Сировина та матеріали 120
5.2.4 Програмне забезпечення для наукових робіт 121
5.2.5 Спецустаткування для наукових (експериментальних) робітника 121
5.2.6 Інші витрати 122
5.2.7 Накладні (загальновиробничі) витрати 122
5.3 Оцінювання економічної ефективності розробки при можливій комерціалізації 123
5.4 Висновки 127
ВИСНОВКИ 129
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 131
ДОДАТКИ 136
ДОДАТОК А. Технічне завдання 137
ДОДАТОК Б. Протокол перевірки МКР на плагіат 141
ДОДАТОК В Лістинг коду 142
ДОДАТОК Г Ілюстративна частина 176

ВСТУП

Актуальність роботи. Зростання темпу життя та популяризація здорового способу харчування формують стійкий попит на цифрові інструменти, які допомагають користувачам контролювати щоденний раціон, баланс мікронутрієнтів, рівень спожитої води та досягати індивідуальних цілей – схуднення, підтримання ваги або її набору [1]. Ринок мобільних застосунків демонструє активний розвиток, однак значна частина наявних рішень або перевантажені складним функціоналом, або не здатні адаптуватися під специфічні потреби користувача, такі як гнучке налаштування параметрів (вік B, стать C, вага D, зріст E, рівень активності F, ціль G), врахування інгредієнтних обмежень та зручне додавання продуктів.

У цьому контексті постає потреба у створенні оптимізованої мобільної системи, яка забезпечує простий, інтуїтивний та достовірний спосіб моніторингу харчування і водного балансу, не перевантажуючи користувача зайвими інтеракціями. Використання платформи Android дозволяє інтегрувати сучасні архітектурні підходи, бібліотеки Jetpack та інструменти Kotlin, що сприяє розробці ефективної та масштабованої системи [2].

Таким чином, тема магістерської роботи «Розробка методів і засобів мобільної системи на платформі Android для підтримки здорового способу харчування» є своєчасною та науково обґрунтованою.

Зв’язок роботи з науковими програмами, планами, темами. Робота виконана в межах наукових досліджень кафедри програмного забезпечення у напрямку створення інтелектуальних мобільних систем та засобів підтримки прийняття рішень для персоналізованого моніторингу.

Мета та завдання дослідження. Метою магістерської роботи є підвищення можливостей підтримки здорового способу харчування шляхом розробки методів і засобів мобільної системи на платформі Android з оптимізованими алгоритмами збору, обробки та подання даних про харчування

та водний баланс, що надає користувачам розвинутий функціонал системи для підтримки свого здоров’я.

Для досягнення поставленої мети передбачено виконання таких завдань:

— провести аналіз сучасних підходів до побудови мобільних систем харчового моніторингу;

— розробити модель обліку фізіологічних параметрів користувача (B–G) та алгоритми оцінки добових норм;

— створити модуль відстеження водного балансу;

— реалізувати механізм обліку харчових продуктів із можливістю додавання за допомогою сканування штрих–коду та пошуку за назвою;

— забезпечити облік обмежень користувача: інгредієнти, алергії, кількість прийомів їжі, допустиме відхилення калорійності ±5%;

— розробити інтерфейс мобільного застосунку, орієнтований на простоту та швидкість взаємодії;

— провести тестування функціональних модулів і оцінити точність, стабільність та зручність користування системою.

Об’єктом дослідження є процес створення мобільних інформаційних систем підтримки здорового способу харчування.

Предметом дослідження є методи та засоби розробки Android–застосунку для персоналізованого контролю харчування та водного балансу.

Методи дослідження. У роботі застосовано такі методи:

— методи проєктування архітектури мобільних застосунків для побудови структурної моделі системи, визначення взаємодії між модулями та формування чітких рівнів абстракції;

— методи адаптивного визначення добової норми калорій для формування алгоритмів обчислення, нормалізації та валідації харчових даних;

— методи прогнозування водного балансу для методи прогнозування водного балансу для формування персоналізованих нагадувань та рекомендацій щодо гідратації на основі історичних даних про споживання води користувачем

Наукова новизна роботи.

1. Подальшого розвитку отримав метод адаптивного визначення добової норми, який, на відміну від відомих, базується на спрощеній адаптивній моделі з використанням системи параметрів B–G (вік, вага, зріст, рівень активності, ціль), що забезпечує підвищену точність розрахунку добової норми калорій при одночасному зменшенні обчислювальної складності і робить метод ефективним для мобільних застосунків із обмеженими ресурсами.

2. Подальшого розвитку отримав метод прогнозування водного балансу, який, на відміну від існуючих, мінімізує кількість дій користувача за рахунок автоматизованих розрахунків на основі індивідуальних фізіологічних параметрів, що забезпечує точні персональні рекомендації щодо добового споживання води та підвищує зручність щоденного трекінгу.

Практичне значення отриманих результатів. Розроблена мобільна система може бути використана широкою аудиторією для контролю харчування, підтримки здорового способу життя, формування корисних звичок та відстеження індивідуальних метрик. Запропоновані методи та технічні рішення можуть бути основою для подальшого удосконалення й комерційного розвитку цифрових продуктів у сфері health–tech.

Особистий внесок здобувача. Усі результати, подані в магістерській кваліфікаційній роботі, було отримано самостійно. Самостійно сформовано модель мобільної системи підтримки здорового способу харчування, розроблено алгоритми обліку індивідуальних параметрів користувача, механізми відстеження водного балансу та функціонал роботи з харчовими продуктами, включаючи можливість пошуку та сканування штрих–кодів.

У науковій роботі [3], опублікованій у співавторстві, автору належать результати, пов’язані з обґрунтуванням застосування елементів гейміфікації в мобільних системах спеціалізованого призначення та формуванням концептуальної моделі їх інтеграції у мобільні застосунки.

Апробація матеріалів. Основні положення, отримані в межах магістерської кваліфікаційної роботи, були доповідалися на V Всеукраїнській

науково–технічній конференції молодих вчених, аспірантів і студентів «Комп’ютерні ігри та мультимедіа як інноваційний підхід до комунікації. 2025р.», де було обговорено особливості застосування методів гейміфікації для підвищення залученості користувачів мобільних систем здорового способу життя.

Публікації. Результати дослідження опубліковано у тезах доповіді на V Всеукраїнській науково–технічній конференції молодих вчених, аспірантів і студентів «Комп’ютерні ігри та мультимедіа як інноваційний підхід до комунікації. 2025 р.» під заголовком: «Використання засобів гейміфікації при створенні мобільних систем спеціалізованого призначення» [3].

Структура та обсяг роботи. Магістерська кваліфікаційна робота складається зі вступу, п’яти розділів, висновків, списку використаних джерел, що містить 46 найменування, 4 додатків. Робота містить 28 ілюстрації, 14 таблиць.

1 АНАЛІЗ СТАНУ СИСТЕМ ПІДТРИМКИ ЗДОРОВОГО ХАРЧУВАННЯ

1.1 Аналіз сучасних мобільних систем для контролю харчування

Сучасні мобільні системи для контролю харчування посідають ключове місце в екосистемі персональних цифрових сервісів, орієнтованих на підтримку здорового способу життя. Зростання популярності таких рішень зумовлено збільшенням обізнаності користувачів щодо важливості раціонального харчування, а також розвитком мобільних технологій, що дозволяють автоматизувати збір, аналіз та візуалізацію харчових даних.

Більшість існуючих мобільних застосунків у цьому сегменті поєднують декілька типових модулів: облік калорій, моніторинг макронутрієнтів, формування персональних рекомендацій, аналітику харчових звичок та інтеграцію з носимими пристроями [31]. Значна увага приділяється автоматизації – зокрема, автоматичному розпізнаванню продуктів за штрих–кодом, використанню відкритих харчових баз даних (OpenFoodFacts, USDA тощо), побудові прогнозних моделей на основі машинного навчання та поведінкової аналітики.

При цьому системи орієнтуються не лише на підрахунок калорій, а й на ширший контекст: водний баланс, режим прийомів їжі, оцінку глікемічного навантаження, рекомендації щодо фізичної активності та формування збалансованого раціону. Ринок активно рухається у напрямку персоналізації, де користувач отримує адаптивні рекомендації, що враховують його фізіологічні параметри, харчові обмеження, цілі та поведінкові патерни.

Популярні рішення, такі як MyFitnessPal, Yazio, Lifesum та інші, уже реалізують комплексні екосистеми харчового моніторингу. Вони пропонують великі каталоги продуктів, синхронізацію з фітнес–трекерами, алгоритми автоматичного формування денних раціонів і машинні моделі прогнозування. Водночас ці системи часто мають обмеження: значна частина функціоналу

доступна лише у платних версіях, персоналізація рекомендацій є поверхневою, а багато локальних продуктів відсутні в базах даних. Крім того, частина застосунків не надає можливості точного налаштування харчових планів або не пропонує прозорих алгоритмів розрахунку калорійності.

У контексті розвитку мобільних систем здорового харчування особливої актуальності набувають рішення, що поєднують коректні методики розрахунку добової норми калорій, адаптивні поведінкові коефіцієнти, алгоритми підбору оптимальних наборів продуктів та прогнозування ключових показників, зокрема водного балансу чи дефіциту калорій. Такий підхід дозволяє підвищити точність рекомендацій і забезпечити користувача індивідуальною, науково обґрунтованою системою харчової підтримки.

Отже, аналіз існуючих рішень демонструє, що ринок мобільних систем контролю харчування активно розвивається, але має низку незакритих потреб: глибша персоналізація, прозорість методів розрахунку, підтримка локальних продуктів, автоматизоване формування збалансованого раціону та інтеграція адаптивних алгоритмів. Це визначає актуальність подальших досліджень і розробки нових мобільних систем для підтримки здорового способу харчування на платформі Android.

1.2 Аналіз аналогів мобільних застосунків для трекінгу калорій та водного балансу

Сучасний ринок мобільних застосунків для контролю харчування представлений широким спектром рішень, які поєднують відстеження раціону, аналіз харчових звичок, формування рекомендацій та базові інструменти контролю здоров’я. Найбільш популярні з них – MyFitnessPal, Yazio, Lifesum та Cronometer – займають значну частку ринку завдяки широкому функціоналу, інтеграціям із фітнес–платформами та наявності великих баз даних продуктів.

MyFitnessPal є одним із найпоширеніших застосунків для контролю калорій, що використовується мільйонами користувачів у всьому світі [4]. Його ключовою перевагою є велика база продуктів, що включає як промислові товари,

так і користувацькі записи. Функціонал застосунку дозволяє вести харчовий щоденник, додавати власні рецепти, сканувати штрих–коди продуктів, а також інтегрувати дані з фітнес–трекерів.

Інтерфейс MyFitnessPal наведено на рисунку 1.1

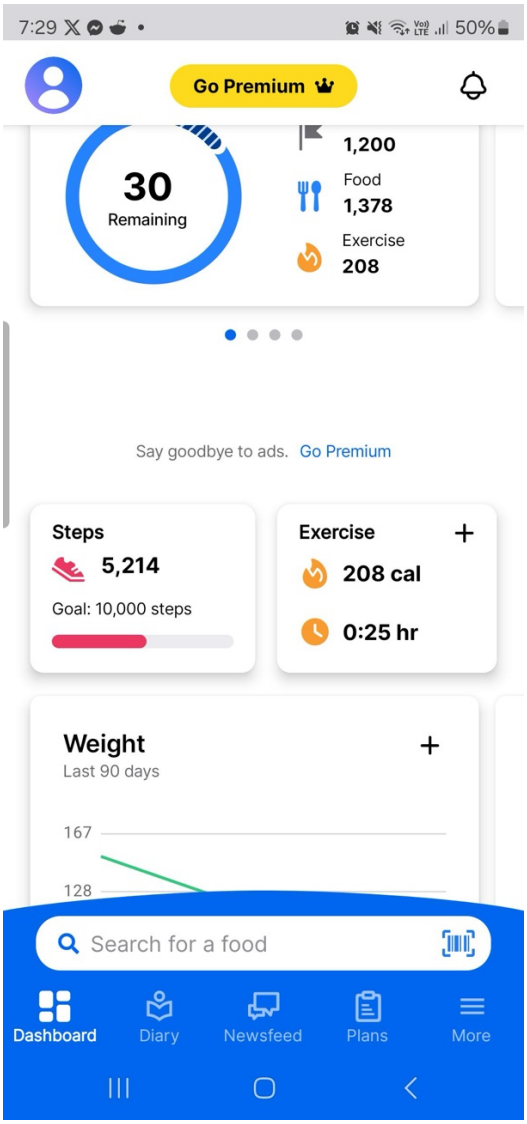


Рисунок 1.1 – Інтерфейс MyFitnessPal

Водночас сильна орієнтація на підрахунок калорій призводить до ускладнення взаємодії: інтерфейс може бути перевантаженим, частина функцій доступна лише за підпискою, а рекомендовані показники спираються на статичні формули без адаптивної корекції.

Yazio позиціонується як застосунок для здорового харчування із фокусом на персоналізованих планах [5]. Він пропонує гнучкі програми схуднення, набору маси та підтримки форми. У застосунку реалізовано автоматизоване формування меню, відстеження макронутрієнтів, лічильник води та можливість логування власних страв.

Інтерфейс Yazio наведено на рисунку 1.2

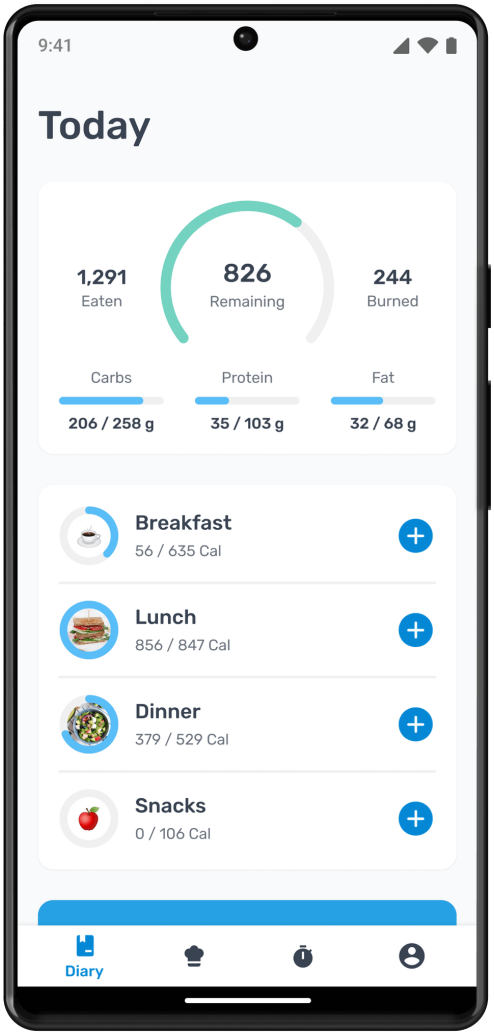


Рисунок 1.2 – Інтерфейс Yazio

Серед обмежень – більшість персональних рекомендацій доступні лише у платній версії, а адаптивність планів базується на фіксованих параметрах, що не враховують зміну поведінкових чи фізіологічних характеристик користувача. Також застосунок не має гнучкої моделі користувачів з урахуванням індивідуальних обмежень та алергій.

Lifesum орієнтується на збалансоване харчування та формування корисних звичок [6]. Він забезпечує підбір дієт, трекінг прийомів їжі, аналіз макронутрієнтів, рекомендації щодо водного балансу та індикатори загального прогресу. Застосунок також пропонує підтримку баркод–сканера та має сучасний мінімалістичний інтерфейс.

Інтерфейс Lifesum наведено на рисунку 1.3

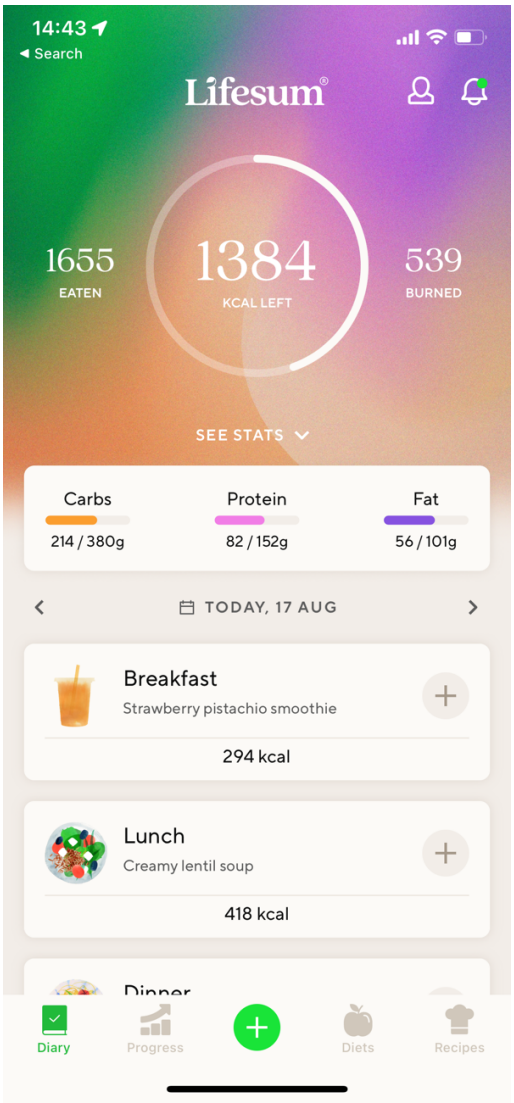


Рисунок 1.3 – Інтерфейс Lifesum

До недоліків можна віднести обмежений набір безкоштовних функцій, відсутність механізму автоматичного формування персонального раціону на основі поведінкових даних користувача, а також обмежену прозорість щодо джерел харчових даних.

Cronometer виділяється високою точністю та орієнтацією на детальний нутрієнт–аналіз [7]. На відміну від більшості аналогів, застосунок надає розширені показники мікронутрієнтів, що робить його популярним серед спортсменів та користувачів з медичними потребами. Сканування продуктів, додавання страв, ведення щоденника та синхронізація з трекерами також доступні.

Інтерфейс Cronometer наведено на рисунку 1.4

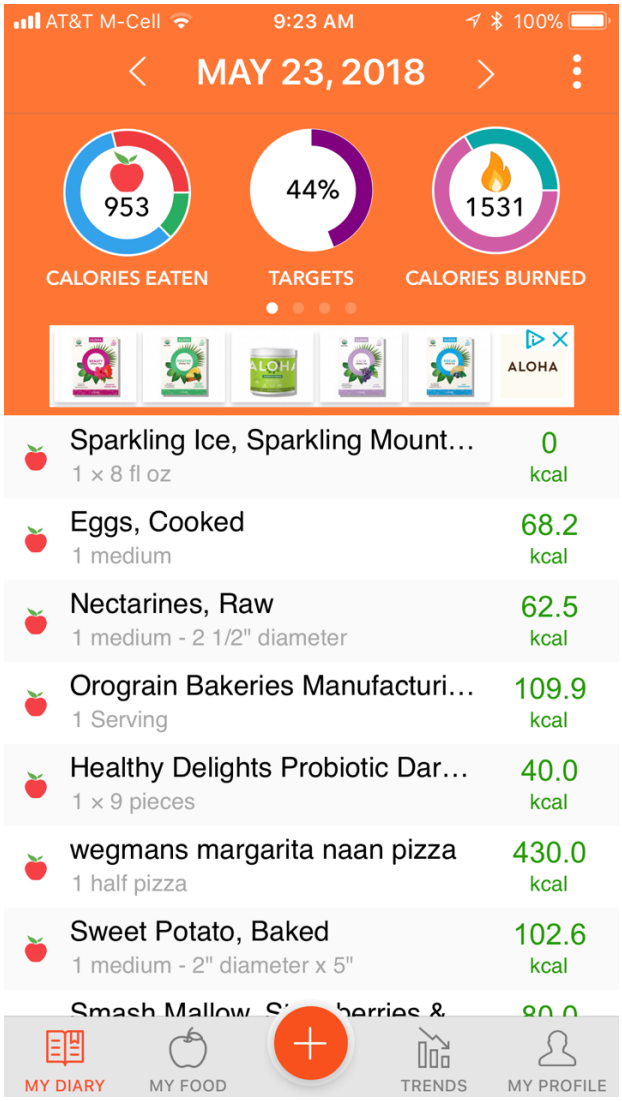


Рисунок 1.4 – Інтерфейс Cronometer

Основна слабка сторона – складність для пересічного користувача через перевантаження даними, а також фокус на нутрієнтах без зручних інструментів

автоматичного планування харчування. Підтримка водного балансу реалізована базово.

Щоб наочно продемонструвати відмінності між цими додатками, була створена порівняльна таблиця (таблиця 1.1).

Таблиця 1.1 – Порівняння характеристик аналогів

	MyFitnessPal	Yazio	Lifesum	Cronometer	Власна розробка
Трекінг калорій	1	1	1	1	1
Трекінг водного балансу	0.5	1	1	0.5	1
Баркод–сканування	1	1	1	0	1
Відкриті джерела харчових даних	0	0	0	0.5	1
Адаптивний розрахунок добової норми	0	0.5	0	0	1
Автоматичне формування денного раціону	0	0.5	0.5	0	1
Облік алергій та обмежень	0	0.5	0.5	0	1
Зміна профільних параметрів користувача	1	1	1	1	1
Загальна оцінка	3.5	5.5	5	3	8

Виходячи з цієї таблиці порівняння, можна зробити висновок, що запропонована власна розробка є актуальною, має переваги перед існуючими аналогами.

1.3 Аналіз підходів до визначення добової норми калорій та водного балансу

Сучасні мобільні системи здорового харчування ґрунтуються на різних математичних моделях і фізіологічних залежностях, які дозволяють оцінити добову потребу користувача в енергії та воді. Достовірність цих показників визначає якість рекомендацій щодо харчування та гідратації, а також точність алгоритмів формування індивідуального раціону.

Попри широке різноманіття моделей, більшість застосунків використовує спрощені формули, що не враховують поведінкову активність, динаміку змін параметрів користувача та індивідуальні особливості організму. Це створює потребу в методах, що забезпечують адаптивність і підвищену точність прогнозів.

Розглянемо підходи до визначення добової норми калорій.

Формули базального метаболізму (BMR). Визначення базальної швидкості метаболізму є ключовим етапом у розрахунку добової енергетичної потреби. Найпоширеніші формули [8]:

— Harris–Benedict (класична модель, менш точна для сучасних популяцій);

— Mifflin–St Jeor (визнана однією з найбільш надійних для дорослих користувачів);

— Katch–McArdle (орієнтована на точні вимірювання складу тіла) [10].

Більшість комерційних застосунків застосовують Mifflin–St Jeor як стандартну формулу. Проте жоден із популярних продуктів не використовує адаптивну корекцію на основі поведінкових патернів, історії харчування чи коливань ваги.

Оцінка рівня фізичної активності (PAL). У класичних підходах добова норма калорій розраховується як: BMR × коефіцієнт фізичної активності (PAL).

Діапазон PAL зазвичай фіксований і задається користувачем вручну. Це суттєво обмежує точність моделі, оскільки інтенсивність активності може змінюватися щоденно. Деякі застосунки синхронізуються зі смарт–годинниками, але не проводять глибинну адаптацію калорійності.

Врахування цілей користувача. Загальноприйнято враховувати три основні цілі:

— зменшення ваги;

— підтримання ваги;

— збільшення маси.

Більшість застосунків використовує спрощені корекції (наприклад, ±250–500 ккал). Проте реальна потреба може залежати від адаптації організму, індивідуальних метаболічних реакцій та типу харчової поведінки.

Адаптивні моделі добової норми. У наукових дослідженнях розглядаються моделі, що враховують:

— зміни маси тіла в динаміці;

— середні показники споживання за певний період;

— рівень виконання рекомендацій;

— поведінкові звички (частота перекусів, нерегулярність прийомів їжі тощо).

Комерційні рішення практично не реалізують такі механізми, що обмежує персоналізацію.

Розроблювана система передбачає адаптивний підхід із використанням додаткового коефіцієнта поведінки, що дозволить більш точно оцінювати реальну потребу в калоріях.

Підходи до визначення добової норми водного балансу. Оптимальний рівень споживання води залежить від фізіологічних параметрів, умов середовища та рівня активності. Існуючі застосунки переважно використовують спрощені моделі.

Фіксовані норми споживання води. Найпоширеніші підходи [11]:

— 2 літри на день – універсальна, але надто узагальнена рекомендація;

— 30–35 мл на кг маси тіла – більш персоналізована модель, але теж статична.

Ці методи не враховують вплив температури, вологості, фізичної активності чи стану здоров’я.

Підходи, засновані на фізичній активності. Деякі системи додають: +0,4–0,7 л за кожні 30–60 хвилин активності.

Такий метод частково компенсує втрати рідини, але не аналізує динаміку споживання води протягом дня.

Моделі, що враховують стан середовища. У наукових публікаціях описуються моделі, які враховують [12]:

— температуру навколишнього середовища;

— інтенсивність потовиділення;

— індивідуальний рівень обміну речовин.

Проте такі методи практично не представлені в комерційних застосунках через складність збору даних.

Прогностичні моделі. Найбільш сучасний напрям – застосування статистичних методів для аналізу динаміки споживання води:

— ковзні середні (moving average);

— експоненційне згладжування;

— імовірнісні моделі залежності від поведінки користувача.

Ці підходи дозволяють створити адаптивний прогноз та підвищити точність рекомендацій.

Розроблювана система передбачає використання моделі прогнозування водного балансу на базі moving average, що забезпечує реакцію на реальні зміни у поведінці користувача без надмірного ускладнення алгоритмів.

Існуючі підходи до визначення добової норми калорій і води широко застосовуються у сучасних мобільних системах, проте більшість із них є статичними та не здатні адаптуватися до змін у поведінці користувача.

Впровадження комбінованої моделі, що включає базові фізіологічні формули, поведінкові коефіцієнти та прогностичні інструменти, дозволить створити мобільну систему з вищим рівнем персоналізації та точності.

1.4 Аналіз відкритих джерел харчових даних

Для побудови інтелектуальних систем контролю харчування критично важливою є наявність достовірних та структурованих харчових даних. Вони забезпечують розрахунок калорійності страв, визначення харчової цінності продуктів, формування рекомендацій, а також автоматизацію добового раціону. Існує кілька відкритих та частково відкритих джерел харчових даних, кожне з яких має різний рівень деталізації, географічного охоплення та зручності інтеграції у мобільні застосунки.

OpenFoodFacts є найбільш масштабною, повністю відкритою глобальною базою харчових продуктів [13]. Система функціонує за краудсорсинговим принципом: користувачі по всьому світу сканують штрихкоди, завантажують фото етикеток та вносять харчову інформацію.

Ключові переваги:

— повна відкритість та безкоштовний REST API;

— глобальне охоплення – понад 2,5 млн продуктів із 160+ країн;

— уніфікована структура даних: макронутрієнти, мікронутрієнти, алергени, інгредієнти, енергетична цінність;

— наявність метрик NutriScore, NOVA та EcoScore;

— підтримка пошуку за штрихкодом, назвою або категорією;

— можливість отримувати зображення етикеток, складу та харчової таблиці.

Обмеження:

— нерівномірна якість даних, оскільки частину інформації вводять користувачі вручну;

— для окремих продуктів можуть бути відсутні повні харчові таблиці;

— інколи трапляються дублікати або застарілі записи.

Попри це, OpenFoodFacts є найбільш придатним джерелом для інтеграції в мобільні застосунки завдяки відкритості, масштабності та простому API.

FoodData Central – науково орієнтована база даних Міністерства сільського господарства США. Вона містить лабораторно перевірену інформацію про харчову цінність, мікроелементи та склад багатьох продуктів [14]. Проте через орієнтацію на ринок США та нерелевантність багатьох позицій для українського та європейського користувача її практична цінність для цього проєкту є обмеженою.

Edamam надає комерційний API орієнтований на інтеграцію у фітнес– та харчові застосунки [15]. Хоча база уніфікована і містить зручні інструменти пошуку, її комерційний характер та обмеження безкоштовного доступу роблять її непридатною для використання в проєкті, що орієнтується на відкритість, масштабованість та мінімізацію витрат.

Існують локальні наукові бази (Ciqual, Fineli, McCance & Widdowson), які містять високоточні дані, але:

— не мають відкритого API,

— мають обмежену кількість продуктів,

— не покривають брендовані товари,

— не підходять для щоденного користування у масовому мобільному застосунку.

Проведений аналіз демонструє, що лише OpenFoodFacts відповідає ключовим вимогам мобільної системи:

— повна відкритість та відсутність ліцензійних обмежень;

— масштабність і глобальність бази;

— наявність API, адаптованого під мобільні додатки;

— підтримка роботи з штрихкодами, що є критичним для автоматичного введення продуктів;

— активна спільнота, що забезпечує регулярне оновлення даних.

Враховуючи відкритість та доступність джерела, OpenFoodFacts дозволяє мінімізувати витрати на підтримку власної великої харчової бази, зберігаючи при цьому достатній рівень точності.

Попри використання OpenFoodFacts як основного джерела, у системі передбачено можливість:

— ручного додавання продуктів користувачем, якщо товар відсутній у базі;

— редагування даних у межах локального сховища;

— зберігання кастомних страв, рецептів та улюблених порцій.

Такий комбінований підхід забезпечує:

— повне покриття даних,

— персоналізацію,

— відсутність залежності від «білих плям» у зовнішній базі.

1.5 Постановка задач дослідження

На основі проведеного аналізу сучасних мобільних систем контролю харчування, методик розрахунку добових норм та доступних відкритих джерел харчових даних формуються ключові задачі, необхідні для розробки мобільної системи підтримки здорового способу харчування на платформі Android. Узагальнюючи результати аналізу, визначаються такі основні напрями дослідження:

1. Розробити архітектуру мобільної системи, яка забезпечуватиме облік харчування, контроль водного балансу та роботу з відкритими харчовими даними.

2. Створити метод адаптивного визначення добової норми калорій, що враховує індивідуальні параметри користувача – вік, вагу, зріст, рівень активності та цілі щодо зміни маси.

3. Розробити метод прогнозування водного балансу, який дозволятиме формувати персоналізовані рекомендації щодо добового споживання води.

4. Сформувати алгоритм автоматизованого підбору денного раціону, який базуватиметься на індивідуальних харчових потребах та даних про продукти з відкритих джерел.

5. Створити модель даних користувача та продуктів, яка забезпечуватиме коректне зберігання, оновлення та синхронізацію інформації.

6. Розробити механізм інтеграції з OpenFoodFacts, що підтримуватиме пошук продуктів, сканування штрихкодів і завантаження харчових характеристик.

7. Реалізувати інструмент локального доповнення харчової бази, який дозволить вручну додавати продукти або рецепти.

8. Обґрунтувати вибір технологій та розробити програмні компоненти системи для підтримки повного життєвого циклу користування застосунком.

9. Провести тестування функціональних модулів, щоб оцінити коректність роботи розрахункових алгоритмів, стабільність застосунку та повноту реалізованого функціоналу.

Визначені задачі формують цілісну науково-прикладну основу дослідження, оскільки охоплюють увесь спектр теоретичних та практичних аспектів створення мобільної системи контролю харчування — від аналізу існуючих методик і формування алгоритмів до проєктування архітектури та реалізації функціональних модулів. Саме вони окреслюють логіку подальшої розробки: у наступних розділах роботи буде послідовно обґрунтовано вибір програмних технологій, представлено методи розрахунку індивідуальних норм, описано інтеграцію з відкритими харчовими базами, розроблено моделі даних та наведено результати тестування створеної системи. Таким чином, поставлені задачі визначають структуру дослідження та забезпечують системність, узгодженість і наукову обґрунтованість усіх етапів розробки застосунку.

1.6 Висновки

Проведений аналіз сучасних мобільних систем підтримки здорового харчування показав, що існуючі рішення – MyFitnessPal, Yazio, Lifesum та Cronometer – надають широкий функціонал, проте мають обмеження у відкритості даних, адаптивності алгоритмів підбору раціону та прогнозування водного балансу.

Дослідження методів розрахунку добових норм калорій та водного споживання засвідчило наявність ефективних математичних моделей, які можуть бути адаптовані для персоналізованих рекомендацій у мобільних системах.

Огляд відкритих джерел харчових даних підтвердив, що OpenFoodFacts є найбільш придатним для інтеграції завдяки відкритому доступу, структурованій інформації та можливості роботи зі штрихкодами. При цьому передбачено доповнення бази локальними продуктами у випадках відсутності даних у OpenFoodFacts.

На основі проведеного аналізу визначено ключові задачі дослідження, спрямовані на розробку архітектури системи, адаптивних алгоритмів розрахунку калорій та водного балансу, автоматизованого підбору раціону, моделі даних користувача та інтеграції з відкритими харчовими джерелами. Це обґрунтовує актуальність створення мобільної системи на платформі Android для підтримки здорового способу харчування.

2 РОЗРОБКА МЕТОДІВ, МОДЕЛЕЙ ТА АЛГОРИТМІВ СИСТЕМИ

2.1 Розробка архітектури мобільної системи

Проєктування архітектури є ключовим етапом створення мобільної системи підтримки здорового способу харчування. Архітектура має забезпечувати масштабованість, модульність, стійкість до змін та можливість поступового розширення функціоналу. З огляду на вимоги, сформовані у першому розділі, доцільно застосувати багаторівневу модульну архітектуру з чітким розмежуванням відповідальностей.

Загальні принципи побудови архітектури. При розробці системи передбачається використання таких принципів:

1. Модульність – розмежування функціональних блоків (профіль користувача, харчові дані, добові норми, водний баланс, рекомендаційний блок).
2. Розширюваність – можливість додавати нові алгоритми та модулі без зміни ядра.
3. Слабка зв’язаність компонентів – використання інверсії залежностей (DI) для керування залежностями [20].
4. Підтримка офлайн–режиму – кешування харчових даних та локальне зберігання користувацьких записів.
5. Єдине джерело правди для даних – використання централізованої моделі даних.

Структурна схема системи. Архітектура передбачає поділ системи на три рівні: рівень даних (Data Layer), рівень доменної логіки (Domain Layer), hівень представлення (Presentation Layer) [33].

Рівень даних забезпечує роботу з джерелами даних:

— локальна база даних (Room);

— відкриті джерела харчових даних (OpenFoodFacts API);

— локальні вручну додані продукти;

— внутрішні обчислювальні модулі.

На цьому рівні формуються репозиторії, які інкапсулюють логіку доступу до даних.

Рівень доменної логіки містить:

— бізнес–логіку системи;

— алгоритми розрахунку добової норми калорій;

— моделі прогнозування водного балансу;

— алгоритм формування денного раціону;

— інтерфейси (use cases) для взаємодії з даними.

Доменний рівень не залежить від конкретних інструментів Android, що забезпечує його стабільність та тестованість.

— Рівень представлення містить:

— екрани взаємодії з користувачем;

— візуалізацію харчових даних;

— віджети водного балансу та калорій;

— ViewModel–и, що керують станом.

Архітектурний підхід – MVVM із використанням Flow [16].

Основні функціональні модулі системи передбачають: модуль профілю користувача, модуль харчових даних та інтеграції з OpenFoodFacts, модуль розрахунку добових норм, модуль формування денного раціону, модуль трекінгу водного балансу, модуль інтерфейсу користувача.

Розглянемо більш детальніше функціонал кожного модуля

Модуль профілю користувача:

— введення параметрів: вік, вага, зріст, стать, активність, цілі;

— зберігання моделі користувача та її оновлення;

— передача параметрів іншим модулям.

Модуль харчових даних та інтеграції з OpenFoodFacts

— пошук продуктів за назвою або штрихкодом;

— обробка відповіді API;

— кешування продуктів у локальній базі даних;

— можливість ручного додавання продуктів.

Модуль розрахунку добових норм

— визначення добової калорійності за адаптивною формулою;

— корекція на основі цілей (схуднення, підтримка, набір ваги);

— прогнозування добового обсягу води.

Модуль формування денного раціону

— підбір рекомендованих продуктів;

— розподіл добової калорійності між прийомами їжі;

— оцінка харчових властивостей.

Модуль трекінгу водного балансу

— ведення щоденного обліку спожитої води;

— візуалізація прогресу;

— повідомлення про необхідність поповнення балансу.

Модуль інтерфейсу користувача

— головна панель стану (калорії, вода, прогрес);

— екран пошуку продуктів;

— екран статистики та історії харчування;

— екран рекомендацій денного раціону.

Взаємодія компонентів організована за принципом «зверху вниз», що відповідає концепції одностороннього потоку даних у сучасних мобільних архітектурах. Рівень Presentation працює виключно з ViewModel, отримуючи вже оброблені дані у реактивному форматі (через Flow або LiveData), і не має прямого доступу до джерел даних чи мережевих сервісів. Усі звернення до репозиторіїв та алгоритмічних модулів відбуваються через доменні інтерфейси, що забезпечує інкапсуляцію логіки, чітке розмежування відповідальностей та можливість тестування кожного шару окремо. Такий підхід запобігає порушенню архітектурних принципів, мінімізує ризики появи неузгоджених станів та сприяє стабільності системи при розширенні функціоналу. Крім того, односторонній

потік даних значно спрощує відстеження змін у системі, підвищує керованість її роботи та гарантує передбачуваність результатів під час обробки користувацьких дій.

Діаграма взаємодії модулів мобільної системи зображена на рисунку 2.1.

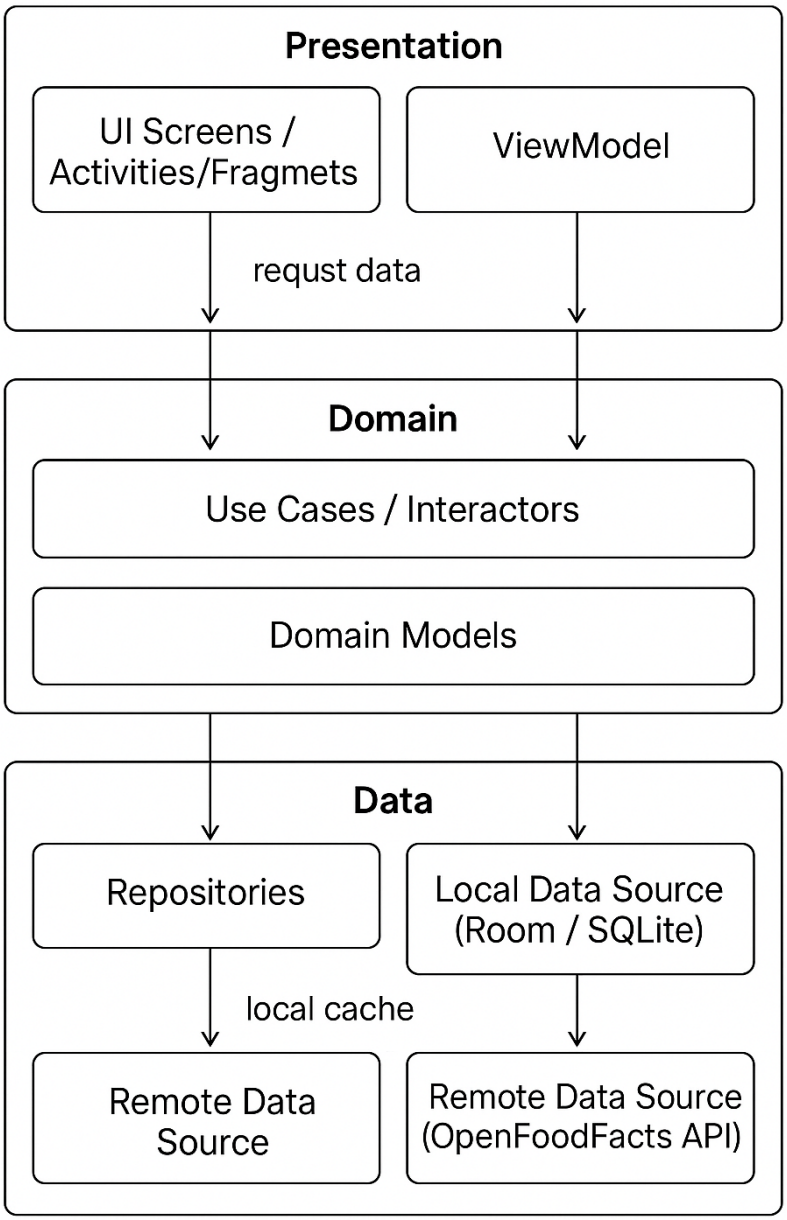


Рисунок 2.1 – Діаграма взаємодії модулів мобільної системи

Основні вимоги до архітектури. Архітектура має забезпечувати:

— стабільну роботу з API навіть за нестабільного інтернет–з’єднання;

— мінімальні затримки при обробці харчових даних;

— легке масштабування для майбутніх модулів (фізична активність, рекомендаційні стратегії тощо);

— підтримку сучасних бібліотек Android (Jetpack Room, ViewModel, Hilt, Retrofit).

2.2 Розробка методу адаптивного визначення добової норми калорій

Розрахунок добової норми калорій є ключовим елементом персоналізованих систем контролю харчування. Для забезпечення коректності рекомендацій необхідно застосовувати метод, який враховує фізіологічні параметри користувача, рівень його щоденної активності та персональні цілі (зниження ваги, підтримання або набір маси). У рамках розробки мобільної системи було сформовано адаптивний метод розрахунку калорійності, що поєднує загальноприйняті формули базального метаболізму з механізмами динамічного коригування результатів.

Додатково метод передбачає урахування змін у поведінкових моделях користувача, наприклад коливань активності протягом тижня або сезонних змін у харчуванні. Це дозволяє забезпечити точніший розрахунок та уникнути статичності, характерної для класичних моделей. Система адаптує рекомендації на основі зворотного зв’язку – даних трекінгу, змін маси тіла та самооцінки рівня навантаження, що робить її більш гнучкою та ефективною у довгостроковому використанні.

Базовий рівень – розрахунок BMR. Для визначення базового рівня обміну (BMR, Basal Metabolic Rate) використано рівняння Міффліна–Сан Жеора як одне з найбільш сучасних та точних для широкої популяції користувачів. Формула враховує стать, вік, зріст та масу тіла, забезпечуючи достатній рівень точності для використання у мобільних системах [36].

Для чоловіків:

BMR = 10 * вага (кг) + 6.25 * зріст (см) – 5 * вік + 5

Для жінок:

BMR = 10 * вага (кг) + 6.25 * зріст (см) – 5 * вік – 161

Отримане значення є базою для подальших адаптивних коригувань.

Урахування рівня фізичної активності. Для трансформації BMR у добову норму калорій застосовується коефіцієнт активності (Activity Factor), що враховує енергетичні витрати користувача протягом дня. У системі передбачено такі рівні фізичної активності. Рівні фізичної активності продемостровано в таблиці 2.1.

Таблиця 2.1 – Рівні фізичної активності

Рівень активності	**Опис**	**Коефіцієнт (AF)**
Низький	мінімальні навантаження	1.2
Легкий	1–3 тренування на тиждень	1.375
Середній	3–5 тренувань	1.55
Високий	6–7 тренувань	1.724
Дуже високий	інтенсивні щоденні навантаження	1.9

Добова норма без урахування цілей: TDEE = BMR * AF, де AF – коефіцієнт активності.

Адаптивне коригування залежно від цілі користувача. Система має підтримувати три типові сценарії:

1. Зниження маси тіла. Застосовується дефіцит калорій – у межах 10–20% від TDEE.
2. Підтримання маси. Значення TDEE використовується без змін.
3. Набір маси. Застосовується надлишок у межах 10–15%:

Одне з ключових рішень – уникнення фіксованих змін у ккал, оскільки відносні коригування забезпечують універсальність і коректність для різних груп користувачів.

Адаптивна складова методу. На відміну від класичних калькуляторів, метод у системі доповнено механізмами адаптації:

1. Регулярне оновлення параметрів користувача. При зміні ваги алгоритм повторно розраховує BMR та TDEE.

2. Динамічна зміна коефіцієнту активності. Користувач може оперативно оновлювати свій рівень активності, що автоматично перераховує добову норму.

3. Персоналізований підбір дефіциту/надлишку. Значення 10–20% визначається самою системою на основі:

— поточної ваги;

— темпу змін (зафіксованого у профілі);

— обраної цілі.

4. Захист від некоректних рекомендацій. Вбудовані обмеження не дають нормі калорій опускатися нижче фізіологічного мінімуму (1200 ккал для жінок, 1500 ккал для чоловіків).

Підсумкова формула адаптивної норми калорій. Процес можна звести до трирівневої моделі [37]:

BMR = Base(user)

TDEE = BMR * ActivityFactor

DailyCalories = TDEE * GoalAdjustment

Де GoalAdjustment ∈ {0.8; 1.0; 1.15}, або адаптивне значення в обраному діапазоні.

Поданий алгоритм (рис.2.2) описує повний цикл визначення добової норми калорій із врахуванням індивідуальних характеристик користувача та динаміки його фізіологічних змін.

На початковому етапі система отримує статичні параметри користувача (вік, стать, вага, зріст), рівень фізичної активності та масив історичних даних.

Після базової валідації проводиться первинний розрахунок основного обміну (BMR) та загальних енергетичних витрат організму (TDEE) з використанням стандартних фізіологічних формул.

У разі доступності історичних даних вмикається адаптивний модуль. Він аналізує зміну ваги у часовому розрізі, порівнює її з очікуваною динамікою та визначає характер відхилення.

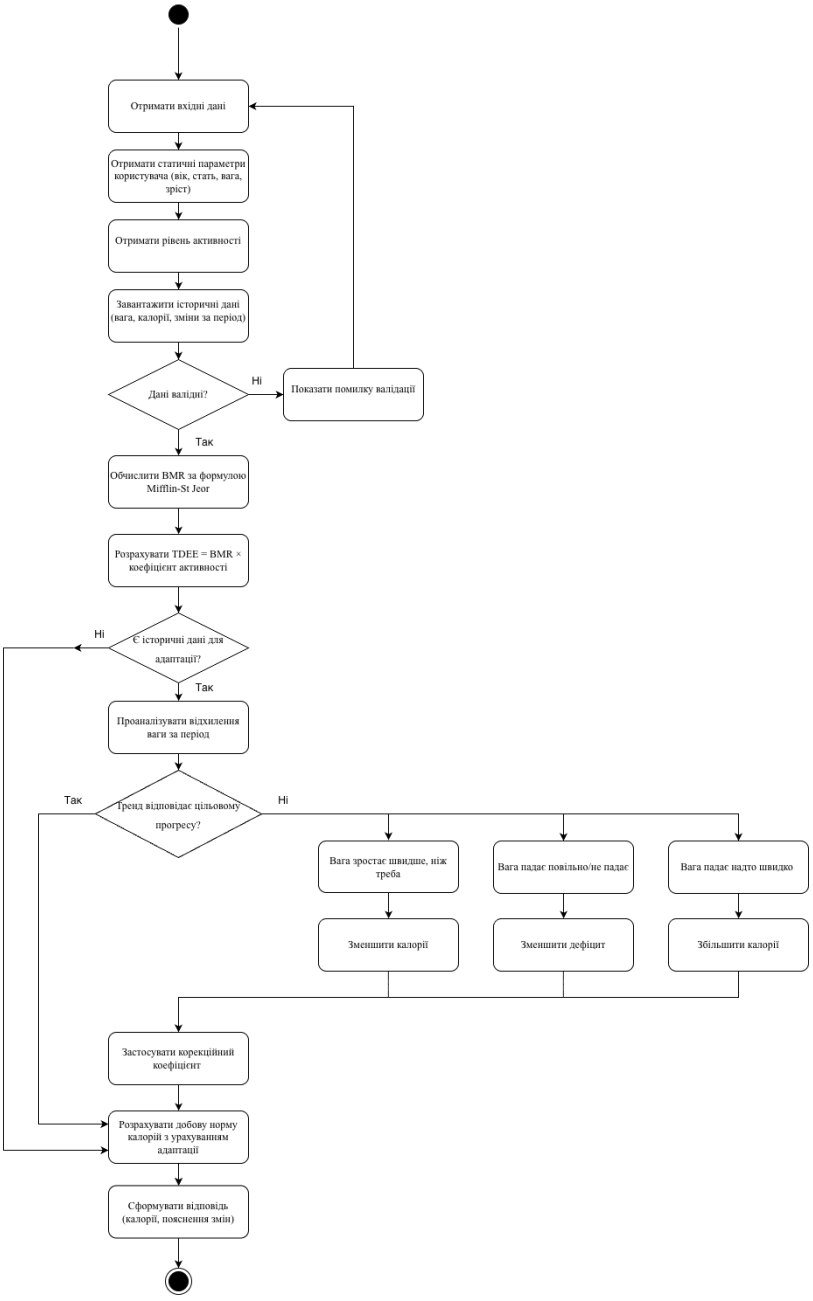


Рисунок 2.2 – UML–діаграма алгоритму визначення добової норми калорій

Завершальним етапом формується фінальна добова норма калорій, яка враховує як базові розрахунки, так і результати адаптації.

Етапи виконання методу:

1. Ініціація процесу (відкриття екрану профілю / старт аплікації).

1.1. Користувач відкриває ProfileFragment. У onViewCreated фрагмент викликає profileViewModel.loadUserProfile().

1.2. ProfileViewModel.loadUserProfile() підписується на userRepository. getUserProfileFlow() (Flow<UserEntity>) через coroutine в viewModelScope. Підписка оновлює userProfileState: StateFlow<UserProfileUI>. UI підписаний на userProfileState відображає поточні параметри.

1.3. Якщо профілю нема (profileExists() == false), ProfileFragment показує діалог ініціалізації (ProfileInitDialog) — користувач вводить вік/зріст/вагу/ціль/рівень активності. При підтвердженні викликається profileViewModel.saveProfile(input).

2. Валідація вхідних даних.

2.1. У ProfileViewModel.onFieldChanged() кожна зміна поля проходить через validateField(field, value); якщо валідація не пройдена, validationResult: StateFlow<ValidationState> сигналізує UI показати помилку.

2.2. Після підтвердження повного профілю saveProfile() викликає userRepository.insertOrUpdateUser(mapToEntity(userProfile)) як suspend-операцію у viewModelScope.launch(Dispatchers.IO).

3. Первинний розрахунок BMR.

3.1. ProfileViewModel після запису або при черговому отриманні userProfileState викликає calorieRequirementCalculator.calculateBMR(userProfile) (синхронний або suspend, залежно від реалізації).

3.2. CalorieRequirementCalculator.calculateBMR() обчислює BMR за формулою Mifflin–St Jeor (виклик: if (user.sex == MALE) formulaM; else formulaF) і повертає значення до ProfileViewModel.

4. Розрахунок TDEE (BMR × ActivityFactor).

4.1. ProfileViewModel отримує activityFactor = ActivityLevelMapper.map(userProfile.activityLevel) і викликає calorieRequirementCalculator.calculateTDEE(bmr, activityFactor).

4.2. Метод повертає tdee, profileViewModel оновлює dailyCaloriesTempState.

5. Перевірка на наявність історичних даних для адаптації.

5.1. ProfileViewModel запитує weightRepository.getWeightHistory(period) (Flow або suspend отримання останніх N записів).

5.2. Якщо історичних записів немає або їх мало (business rule), процес пропускає адаптацію і переходить до кроку 8. Якщо є — переходить до кроку 6.

6. Аналіз динаміки ваги (AdaptiveCalorieService).

6.1. AdaptiveCalorieService.analyzeWeightTrend(weightHistory, targetWeight) викликається як suspend із Dispatchers.Default.

6.2. Сервіс обчислює тренд (скользящее середнє / лінійна регресія) та швидкість зміни ваги (kg/тиждень). Результат — TrendAnalysis(resultType, slope, confidence).

6.3. Результат повертається у ProfileViewModel.

7. Прийняття рішення про корекцію (CorrectionEngine).

7.1. ProfileViewModel виконує бізнес-правила: якщо trend відповідає цілі — GoalAdjustment = default (наприклад 0.8/1.0/1.15). Якщо тренд показує відхилення (вага падає занадто швидко/повільно/зростає), ProfileViewModel викликає correctionEngine.computeAdjustment(trend, currentTDEE, userProfile).

7.2. CorrectionEngine повертає adjustmentPercent або конкретну рекомендацію (increase calories by X kcal, decrease by Y kcal), але гарантує фізіологічні межі (clamp у [minCalories, maxSafe]).

8. Формування остаточної добової норми.

8.1. ProfileViewModel застосовує DailyCalories = TDEE * GoalAdjustment * AdaptiveFactor, де AdaptiveFactor виходить від CorrectionEngine.

8.2. Перевірка обмежень: якщо DailyCalories < minAllowed або > maxAllowed, значення коригується та у validationResult передається попередження.

9. Збереження та трансляція результатів.

9.1. Після прийняття остаточного значення ProfileViewModel викликає userRepository.saveDailyTargets(userId, DailyTargets(...)) або записує поле dailyCalories у UserEntity. Операція — suspend у Dispatchers.IO.

9.2. userRepository записує оновлення у Room через userDao.updateDailyTargets(...). DAO змінює дані, Room тригерить Flow, на який підписані всі зацікавлені ViewModel(и).

10. Оновлення UI і повідомлення користувача.

10.1. DailyCalorieUiModel у ProfileViewModel оновлюється і StateFlow передає нові дані у ProfileFragment і в головний дашборд. UI автоматично оновлює віджет прогресу калорій.

10.2. Якщо корекція була суттєвою або виходила за певні пороги, ProfileViewModel викликає notificationManager.showAdaptiveRecommendation(notificationDto) для інформування користувача (за потреби — діалог підтвердження зміни цілі).

11. Повторна адаптація в циклі (оновлення тренду).

11.1. Система зберігає логи рішень (adaptiveLogRepository.log(adaptationEvent)) для подальшого аналізу та audit.

11.2. Кожного дня/тижня запускається періодичний WorkManager-задач AdaptiveRecalculationWorker, яка витягує останні вагові дані та повторює кроки 5–9 у бекграунді. Якщо рішення змінює норму, воно синхронізується з локальною базою і відправляє silent push або локальне нотифікейшн.

12. Обробка помилок і відкатів.

12.1. У всіх suspend-операціях використовується try/catch з обробкою IOException, SQLException та власних DomainException. При помилці мережі ProfileViewModel показує snackbar і використовує останні збережені значення.

12.2. У разі некоректного входу даних обчислення блокується, користувач отримує контекстну підказку і запит на корекцію полів.

13. Тестування і валідація алгоритму (тест-слої).

13.1. На рівні одиничних тестів CalorieRequirementCalculator покривається JUnit-тестами з набором прикладів для різних статей/віку/зросту/ваги.

13.2. AdaptiveCalorieService тестується із наборами історичних даних (unit + integration tests) — перевірка логіки analyzeWeightTrend і correctionEngine.

13.3. Інтеграційні тести (Android instrumented tests) перевіряють весь шлях: ProfileFragment → ProfileViewModel → Repository → Room, емулюючи запис ваги і перевіряючи оновлення dailyCalories

Таким чином, алгоритм забезпечує точніший, персоналізований та більш стабільний підхід до визначення енергетичних потреб користувача.

2.3 Розробка методу прогнозування водного балансу

У межах розроблюваної мобільної системи запропоновано метод адаптивного визначення добової норми калорій, який інтегрує фізіологічні параметри користувача із показниками його реальної активності та поведінкових змін у динаміці. На відміну від традиційних рішень, що базуються на статичному формульному підрахунку з єдиним початковим результатом, запропонований метод забезпечує регулярне коригування калорійної цілі відповідно до змін маси тіла, рівня активності та характеристик метаболізму користувача. Основа методу – адаптивна модель, що використовує параметри B–G (стать, вік, вага, зріст, коефіцієнт активності, ціль зміни ваги та прогнозований рівень метаболізму), які переналаштовуються автоматично під час накопичення нових даних.

Одночасно з цим система виконує адаптивний розрахунок добового споживання води, що базується на масі тіла та коригується щодо інтенсивності фізичної активності. У базовому вигляді добова норма визначається формулою [36]:

Water_Intake_base = 30 * Weight_kg (мл/добу)

Для користувачів із підвищеним рівнем активності застосовується додатковий коригувальний коефіцієнт:

Water_Intake = Water_Intake_base + 350 * Activity_level

Де Activity_level – дискретний індекс (0 – низька активність, 1 – помірна, 2 – висока), що визначається на основі даних з модуля відстеження активності.

Таким чином, система забезпечує одночасну адаптацію як калорійної норми, так і водного балансу з урахуванням особливостей організму та поточної поведінки користувача.

Алгоритм функціонує таким чином: при первинному налаштуванні користувач вказує основні антропометричні характеристики, на основі яких клас UserProfile формує базове значення BMR (Basal Metabolic Rate). Далі клас CalorieRequirementCalculator доповнює розрахунок коефіцієнтом фізичної активності, визначає TDEE (Total Daily Energy Expenditure) і встановлює стартову врівноважену калорійну норму. Протягом подальшого використання системи дані щодо змін маси тіла зберігаються у WeightRepository та передаються до AdaptiveCalorieService, який аналізує їх разом із записами про активність, отриманими із ActivityTrackerService, та даними про харчування з FoodIntakeRepository. На основі цього виконується корекція параметра метаболічного адаптування та коефіцієнта активності, що приводить до динамічного оновлення добової калорійної норми без додаткових дій з боку користувача.

Комунікація між модулями реалізована через архітектурний підхід з чітким розділенням відповідальностей. Клас UserProfileInteractor виконує роль фасада: він отримує звернення з рівня інтерфейсу користувача від ProfileViewModel, ініціює розрахунки через CalorieRequirementCalculator, а також керує оновленнями, взаємодіючи з AdaptiveCalorieService. Усі дані зберігаються локально, при цьому менеджер UserPreferencesStorage відповідає за збереження вихідних параметрів, а HealthDataRepository – за накопичення фактичних щоденних записів. На рівні UI клас DailyCalorieUiModel відображає результати розрахунків у вигляді чіткої цілі на день і супровідного прогресу.

Завдяки інтеграції із підсистемою харчування, у якій FoodBarcodeScanner і FoodSearchEngine прискорюють облік продуктів, система отримує достовірні дані про фактично спожиті калорії, що підсилює точність адаптивної моделі. Щодня користувач отримує актуалізовану норму, яка може змінюватися як у більший, так і в менший бік залежно від результативності дотримання обраної стратегії (підтримка, схуднення або набір маси). У випадку тривалої відсутності прогресу система автоматично перебудовує прогнозований метаболічний профіль, запобігаючи застою та підтримуючи мотивацію.

Етапи виконання методу:

1. Ініціювання процесу користувачем

1.1. Користувач відкриває HydrationFragment або будь-який інший екран, де відображається норма води.

1.2. Фрагмент у onViewCreated() викликає hydrationViewModel.loadHydrationData().

1.3. ViewModel підписується на UserProfileRepository.getUserProfileFlow() та ActivityRepository.getDailyActivityFlow(). Обидва джерела даних оновлюють внутрішній StateFlow<HydrationUiState>.

2. Первинне налаштування профілю

2.1. Якщо система виявляє, що профіль порожній, ProfileInitDialog відкривається автоматично.

2.2. Після заповнення параметрів (вік, вага, стать, активність) ProfileViewModel.saveProfile() викликає UserProfileRepository.insertOrUpdateUser().

2.3. Після успішного збереження запускається HydrationViewModel.recalculateBaseWaterIntake().

3. Базовий розрахунок норми води (формула 30 мл на 1 кг ваги)

3.1. HydrationViewModel викликає WaterIntakeCalculator .calculateBaseIntake(weightKg).

3.2. Клас WaterIntakeCalculator обчислює:

waterBase = weightKg * 30

та повертає результат у мл/добу.

3.3. ViewModel оновлює hydrationBaseState, а UI відображає стартову базову норму.

4. Отримання рівня активності з ActivityTrackerService

4.1. HydrationViewModel у фоновій корутині викликає:

activityTrackerService.getTodayActivityLevel()

або підписується на ActivityRepository.getActivityLevelFlow(), якщо дані надходять із датчиків/синхронізації.

4.2. Після отримання активності ViewModel викликає

WaterIntakeCalculator.calculateAdjustedIntake(waterBase, activityLevel)

де:

water = waterBase + 350 * activityLevel.

5. Формування цільового добового показника води

5.1. Результат розрахунку потрапляє в hydrationTargetState: StateFlow<HydrationTargetUiModel>.

5.2. HydrationFragment підписаний на цей StateFlow й оновлює UI — норму води, прогрес за день, графік.

6. Динамічне оновлення прогнозу на основі активності й поведінки

6.1. При зміні даних активності ActivityTrackerService пушить оновлене значення у Flow.

6.2. Це тригерить повторний виклик:

HydrationViewModel.recalculateHydration() → WaterIntakeCalculator.

6.3. Якщо активність підвищується протягом дня, система адаптивно збільшує норму без додаткових дій користувача.

7. Запис фактичного споживання води

7.1. Користувач натискає кнопку «+200 мл» або додає воду вручну через AddWaterDialog.

7.2. HydrationViewModel.addWater(amount) викликає:

HydrationRepository.insertWaterRecord(today, amount).

7.3. Після збереження Room оновлює потокові дані, і hydrationProgressState автоматично дає нове значення UI-прогресу.

8. Інтеграція з іншими модулями (харчування, активність, профіль)

8.1. UserProfileInteractor виступає фасадом і координує дані між модулями.

8.2. При оновленні ваги у WeightRepository, наприклад через AddWeightDialog:

- запис зберігається

- тригериться Flow ваги

- HydrationViewModel отримує оновлення

- запускається повторний розрахунок базової норми:

waterBase = weight * 30.

8.3. При зміні активності ActivityTrackerService → ActivityRepository оновлює індекс (0/1/2).

8.4. Система перераховує воду:

waterTarget = waterBase + 350 * activityLevel.

9. Роль AdaptiveCalorieService у спільних розрахунках

9.1. Хоч сервіс відповідає за калорії, він надає класам інформацію про активність та метаболічні зміни.

9.2. HydrationViewModel через UserProfileInteractor може отримати розширений коефіцієнт активності або потреби в рідині (для розширених моделей).

10. Зберігання та доступ до даних

10.1. Базові параметри (вага, стать, активність) зберігає UserPreferencesStorage.

10.2. Щоденні записи води — HealthDataRepository.

10.3. Усі розрахунки виконуються в ViewModel, зберігаються в local storage й транслюються UI через StateFlow.

11. Щоденне оновлення моделі (автономний режим)

11.1. О 00:00 спрацьовує DailyHydrationWorker (WorkManager).

11.2. Він:

- оновлює прогноз,

- переносить залишок/надлишок в історію,

- створює новий запис дня,

- викликає recalculation з урахуванням активності в новому дні.

11.3. Якщо активність була нульова 2–3 дні, система знижує очікуваний активний індекс.

12. Обробка помилок

12.1. Усі виклики репозиторіїв виконуються всередині try/catch.

12.2. При збоях Room → показується локальний fallback (останнє збережене значення).

12.3. При відсутності активності система використовує дефолтний індекс (0).

13. Відображення на UI

13.1. HydrationUiModel віддає:

- добову норму (мл)

- спожите за день

- відсоток прогресу

- рекомендації (наприклад, підвищити споживання через високу активність).

13.2. HydrationFragment підписаний на всі StateFlow і оновлює:

- прогрес-бар

- графік споживання

- підказки на картках.

14. Автоматичне прогнозування водного балансу

14.1. Система накопичує історію активності та історію споживання.

14.2. HydrationPredictorEngine аналізує тенденції: чи користувач хронічно недопиває, чи перевищує норму.

14.3. Потім Engine формує прогнозований дефіцит/надлишок та пропонує персональні рекомендації.

14.4. ViewModel оновлює hydrationInsightsState, який підтягується у відповідний екран (інсайти/аналітика).

Блок-схема алгоритму роботи методу прогнозування водного балансу зображена на рисунку 2.3.

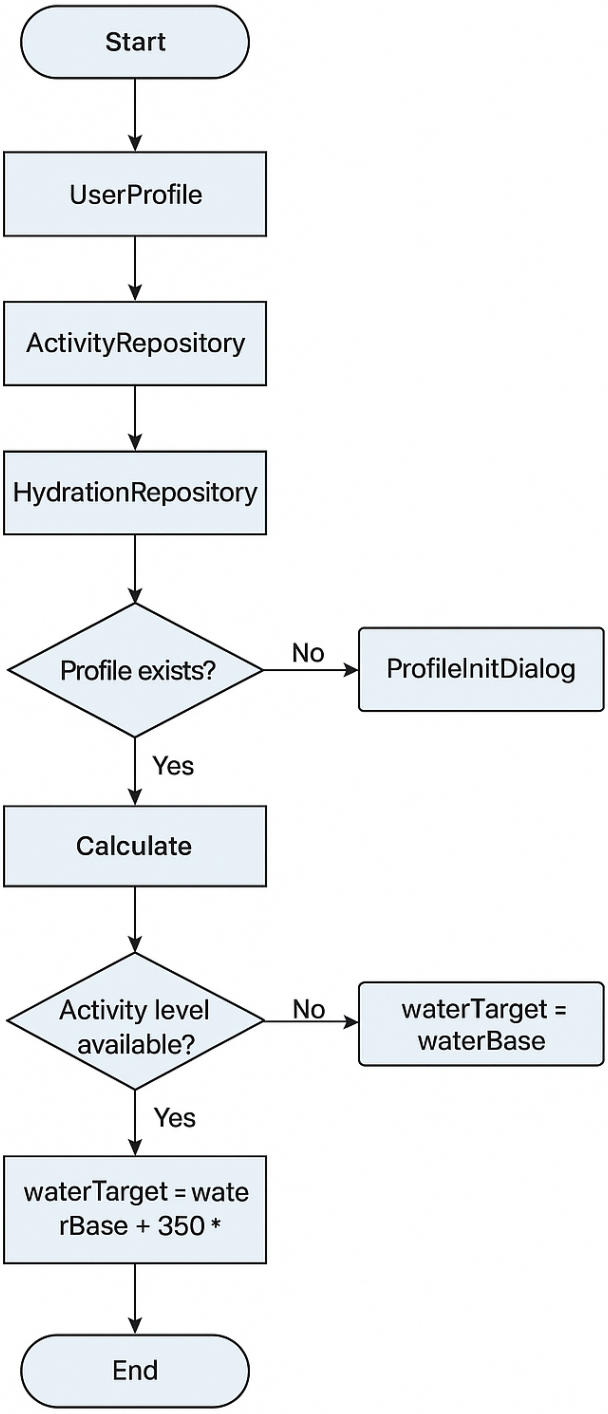


Рисунок 2.3 – Блок-схема алгоритму роботи методу прогнозування водного балансу

У підсумку розроблений метод формує персоналізовані калорійні рекомендації, чутливі до контексту – зміни маси, способу життя та харчової поведінки. Його реалізація забезпечує автономність, мінімальну кількість взаємодій з боку користувача та експертний рівень точності за рахунок тісної взаємодії класів профілю, харчування та активності. Це дозволяє системі функціонувати як інтелектуальний асистент у досягненні цілей здорового способу життя.

2.4 Розробка моделі даних користувача та продуктів харчування

У розроблюваній мобільній системі запропоновано модель даних, орієнтовану на розширюваність, точне відображення нутрітивних характеристик продуктів і підтримку адаптивних розрахунків добових рекомендацій [37]. Модель побудована з урахуванням розділення відповідальностей між доменним рівнем та рівнем локального зберігання, що підвищує узгодженість та тестованість архітектури.

Модель даних користувача. Користувацькі параметри представлені доменною сутністю UserProfile, яка містить набір фізіологічних і поведінкових характеристик:

- стать, вік, вага, зріст;
- рівень фізичної активності;
- ціль зміни ваги (підтримка, зниження, збільшення);
- історичні зміни маси тіла;
- параметри адаптивної корекції.

Параметри, що змінюються динамічно, зберігаються окремо у структурі HealthData, щоб відокремити постійні налаштування від показників, що накопичуються під час взаємодії з системою. Історія змін ваги фіксується у вигляді колекції записів WeightRecord(timestamp, weight).

Персистентне зберігання виконується на рівні Data Layer у вигляді DataStore/Room–таблиць:

— user_profile_table – базові параметри користувача;

— weight_history_table – хронологія вимірювань ваги.

Класи UserPreferencesStorage та WeightDao відповідають за операції читання й запису, а UserProfileRepository інкапсулює логіку їх доступу для доменного рівня.

З боку бізнес–логіки роботою з профілем керує інтерфейс UserProfileInteractor, який забезпечує узгодженість даних між розрахунковими сервісами та шаром UI.

Модель даних продуктів харчування. Харчові продукти представлені доменною моделлю FoodProduct, що містить:

— назву, бренд, категорію;

— енергетичну цінність (ккал);

— нутрієнти: білки, жири, вуглеводи;

— додаткові характеристики (цукри, сіль, клітковина тощо);

— штрих–код (EAN/UPC) для можливості швидкої ідентифікації;

— джерело даних (локальне/онлайн OpenFoodFacts).

У локальній базі даних передбачено два типи сутностей:

1. База продуктів (food_table) – збереження основних продуктів.
2. Щоденні записи харчування (food_intake_table), які представляються моделлю FoodIntakeEntry(productId, timestamp, portionSizeGram, calculatedCalories).

Розділення сутностей дає можливість уникнути дублювання нутрітивних даних та дозволяє доповнювати інформацію при черговому скануванні або оновленні даних з OpenFoodFacts.

Інтеграція з онлайн–джерелами харчових даних. Для віддаленого пошуку використовується OpenFoodFactsApiService, який повертає DTO–структури, що перетворюються у FoodProduct за допомогою FoodProductMapper. Агрегування локального та онлайн–пошуку виконує FoodSearchEngine.

Модуль сканування штрих–кодів (FoodBarcodeScanner) забезпечує швидке додавання продуктів у щоденний раціон, мінімізуючи взаємодію користувача з UI.

Забезпечення консистентності даних у розрахунках. Розрахункові модулі (CalorieRequirementCalculator, AdaptiveCalorieService) взаємодіють виключно з актуальним представленням даних, отриманим через FoodIntakeRepository та UserProfileRepository. Таким чином гарантується:

— централізований контроль доступу до інформації;

— розрахунки на основі достовірних та актуальних значень;

— відсутність можливості обходу бізнес–логіки шляхом прямої модифікації персистентних даних.

У результаті побудована модель даних забезпечує:

— повне охоплення параметрів, необхідних для адаптивних розрахунків;

— модульність і простоту масштабування з мінімальними змінами структури;

— можливість інтеграції з зовнішніми харчовими базами;

— узгодженість між UI, обчислювальними сервісами та шаром зберігання.

Запропонована структура створює основу для інтелектуального аналізу харчових звичок користувача та забезпечує надійну фундаментальну підтримку усіх функціональних модулів системи. Завдяки чіткому розмежуванню доменної логіки, локального зберігання та сервісів обробки даних, модель гарантує високу точність обчислень, передбачуваність поведінки та легкість у масштабуванні. У майбутньому така архітектура дозволить безболісно інтегрувати додаткові аналітичні модулі — наприклад, рекомендаційні системи, прогностичні алгоритми чи автоматизований аналіз харчових патернів — без порушення існуючих залежностей. Централізований контроль консистентності даних забезпечує коректність взаємодії між профілем користувача, харчовими продуктами та алгоритмами розрахунків, що формує стабільне середовище для

розвитку продукту та підвищення точності персоналізованих рекомендацій у наступних версіях системи.

Структура класів моделі даних користувача та продуктів харчування зображена на рисунку 2.4

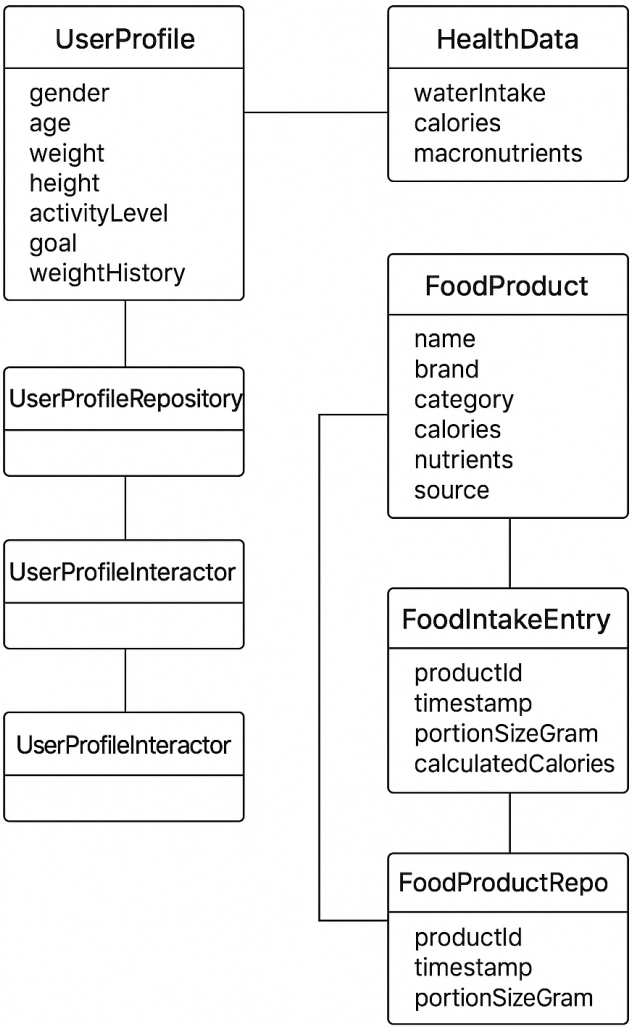


Рисунок 2.4 – Структура класів моделі даних користувача та продуктів харчування

2.5 Розробка загального алгоритму роботи системи

Функціонування мобільної системи підтримки здорового харчування відбувається в рамках циклічної моделі, у якій результат виконаної дії користувача відразу впливає на оновлення його індивідуальних харчових норм та

рекомендацій. Алгоритм включає п’ять основних фаз: ініціалізація профілю, збір та зберігання даних, аналіз показників, адаптивне оновлення норм, формування рекомендацій та візуальний зворотний зв’язок.

Фаза 1. Ініціалізація профілю

1. Користувач встановлює застосунок і запускає його вперше.
2. AuthManager перевіряє, чи існує збережений профіль. Якщо ні – ініціюється створення профілю.
3. ProfileSetupViewModel на UI запитує:
   — стать, вік, зріст, вагу;
   — рівень фізичної активності;
   — ціль (схуднення / підтримка / набір ваги).
4. UserProfileManager валідовує введені параметри.
5. Дані зберігаються у UserRepository (Room).
6. DailyNormCalculator виконує:
   — розрахунок BMR, TDEE, добових норм Б/Ж/В;
   — розрахунок добової потреби у воді.
7. Згенеровані норми відображаються на головному екрані через DailyNormUiModel.

Режим готовності системи активовано.

Фаза 2. Внесення харчових продуктів протягом дня

8. При кожному прийомі їжі користувач:
   — обирає продукт із каталогу або
   — сканує штрих–код (BarcodeScannerService) або
   — створює власний продукт через FoodEditorViewModel.
9. Система отримує інформацію з FoodProductRepository:
   — харчова цінність на 100 г;
   — категорія продукту;
   — додаткові властивості (алергени, тип нутрієнтів).
10. MealEditorViewModel виконує перерахунок калорій відповідно до вказаної порції.

11. Запис зберігається у DailyFoodIntakeRepository.

12. Запускається перерахунок добового прогресу через DailyIntakeManager.

Фаза 3. Безперервний аналіз стану раціону в режимі реального часу

13. ProgressAnalyzer отримує фактичні показники за день:

— спожиті калорії;

— співвідношення білків/жирів/вуглеводів;

— кількість випитої води.

14. Показники порівнюються з індивідуальними нормами профілю.

15. У разі відхилень визначаються проблемні сегменти:

— надлишок калорій;

— дефіцит білка тощо.

16. Дані транслюються до UI через Flow → інтерфейс оновлюється без дій користувача.

Фаза 4. Адаптивне оновлення індивідуальних норм.

17. Користувач періодично вносить свою вагу → дані записуються у WeightRepository.

18. AdaptiveCalorieService виконує аналіз динаміки – порівняння фактичних трендів з очікуваною моделлю.

19. Якщо:

— прогресу немає → коригується активність або метаболічний коефіцієнт,

— процес надто швидкий → система підвищує калорійність, уникаючи ризиків.

20. DailyNormCalculator оновлює TDEE та добові норми.

21. Оновлення одразу відображаються в UI через DailyCalorieUiModel.

Система адаптується до життя користувача, а не навпаки.

Фаза 5. Генерація персональних рекомендацій.

22. RecommendationEngine аналізує поведінкові патерни:

— час та групи продуктів у раціоні;

— регулярність заповнення;

— досягнення цілей.

23. Формує текстові рекомендації:

— збільшити білки у сніданку;

— уникати швидких вуглеводів ввечері;

— нагадування випити воду.

24. Рекомендації показуються на UI адаптивними блоками.

Принцип безперервності обробки даних (рис. 2.5). Подія користувача → Запис у репозиторій → Запуск доменного сервісу → Перерахунок показників → Оновлення UI → Пропозиція рекомендацій

Без кнопки “Оновити дані” та без додаткових дій зі сторони користувача.

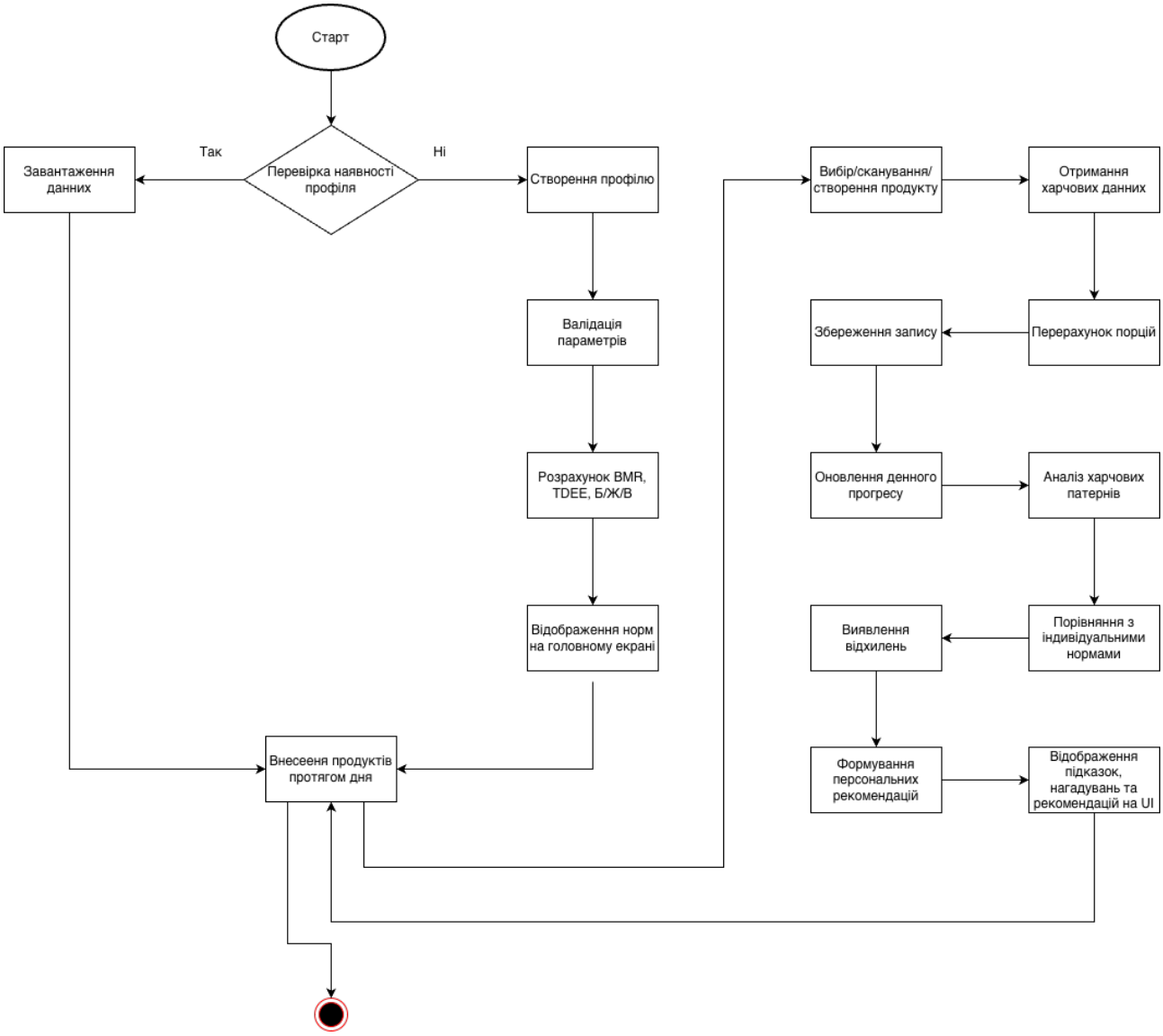


Рисунок 2.5 – Блок-схема роботи загального алгоритму системи

2.6 Висновки

У другому розділі магістерської роботи було розроблено архітектуру мобільної системи на платформі Android, яка забезпечує чітке розділення відповідальностей між Presentation, Domain та Data рівнями та дозволяє інтегрувати адаптивні алгоритми розрахунку добових норм калорій та водного балансу. Розроблено метод адаптивного визначення добової норми калорій, що враховує індивідуальні антропометричні дані користувача, його рівень активності та харчову поведінку, а також метод прогнозування водного балансу, який формує персоналізовані рекомендації щодо добового споживання води. Описано модель даних користувача та продуктів харчування, що забезпечує зберігання та синхронізацію інформації про профіль користувача, записи про харчування та активність. Розроблено загальний алгоритм роботи системи, який демонструє послідовність взаємодії модулів, обробки даних та оновлення рекомендацій у динаміці, забезпечуючи автономність і точність роботи застосунку. У підсумку, у другому розділі сформовано методологічну та програмну основу для реалізації мобільної системи підтримки здорового способу харчування, що забезпечує інтеграцію користувацького інтерфейсу, адаптивних алгоритмів і моделі даних у єдиний функціональний комплекс.

3 РОЗРОБКА ПРОГРАМНИХ ЗАСОБІВ СИСТЕМИ

3.1 Варіантний аналіз і обґрунтування вибору технологій та інструментів розробки

Для реалізації мобільної системи на платформі Android було проведено варіантний аналіз сучасних технологій, мов програмування та інструментів розробки. Основними критеріями вибору стали нативна продуктивність, офіційна підтримка платформою, зрілість екосистеми, наявність бібліотек для роботи з даними, а також можливість швидкої інтеграції мережевих сервісів. З урахуванням цих вимог мова програмування Kotlin визначена як ключова технологія для розробки застосунку.

Kotlin – це сучасна статично типізована мова, повністю сумісна з Java Virtual Machine та офіційно підтримувана Google як пріоритетний інструмент для Android–розробки [17]. Вона забезпечує значне скорочення коду, усуває надлишкову синтаксичну «шумність», властиву Java, що позитивно впливає на швидкість розробки та зменшення кількості дефектів. Мова пропонує низку програмних механізмів, що суттєво підвищують надійність мобільних систем: null–safety допомагає захиститися від NullPointerException на рівні компіляції; sealed–класи та data–класи оптимізують роботу з доменною моделлю, гарантуючи контрольовані стани та автоматичне формування допоміжних методів.

Kotlin нативно підтримує корутини – високоефективний механізм асинхронного та конкурентного програмування, що дозволяє оптимізувати виконання завдань, пов’язаних з мережею, обробкою сенсорних подій, синхронізацією локальних даних та взаємодією з репозиторіями. На відміну від традиційних потоків, корутини забезпечують низьке навантаження на ресурси пристрою, що є критично важливим для застосунків, спрямованих на щоденне використання та роботу у фоновому режимі.

Ще однією сильною стороною мови є її інтеграція з Android Jetpack – модульною колекцією бібліотек, розроблених для прискорення створення масштабованих та життєздатних Android–додатків. Підтримка таких технологій як ViewModel, LiveData, Room, DataStore, WorkManager значно спрощує роботу з життєвим циклом UI, постійним збереженням інформації та асинхронними операціями. Крім того, Kotlin активно розвивається у напрямку мультиплатформності (Kotlin Multiplatform), що відкриває перспективу часткового перенесення бізнес–логіки на інші платформи у майбутньому [19].

Процес компіляції Kotlin та Java наведено на рисунку 3.1

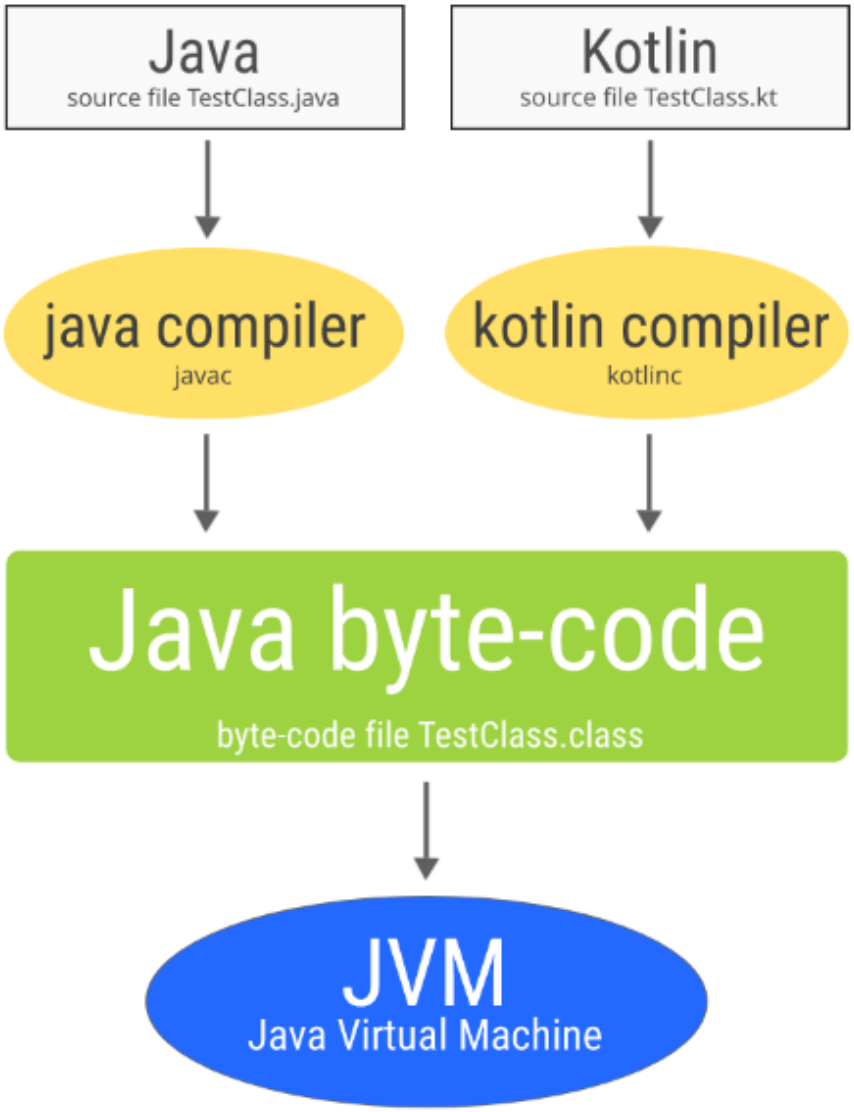


Рисунок 3.1 – Процес компіляції Kotlin та Java

Важливим фактором також є велика спільнота розробників, офіційні рекомендації Google[, наявність матеріалів навчання, прикладів та довгострокова стратегія розвитку мови [18]. Це забезпечує стабільність, підтримку інструментів CI/CD і можливості швидкого впровадження нових функціональних модулів у застосунок.

Завдяки поєднанню перерахованих властивостей Kotlin забезпечує надійне технологічне підґрунтя для розробки мобільної системи підтримки здорового способу харчування, орієнтованої на масштабування, безпечність та високу продуктивність на Android–пристроях.

Android SDK [37] виступає базовою платформою, яка забезпечує застосунку повний доступ до функціональності операційної системи: файлової системи, сенсорів пристрою (акселерометр, крокомір, GPS), служб безпеки, системного управління пам’яттю та мережевих інтерфейсів. SDK надає API для побудови UI, роботи з фоновими сервісами, push–сповіщеннями, Bluetooth, Health Services API та іншими можливостями, необхідними для моніторингу здорових звичок користувача.

На основі SDK використано набір архітектурних бібліотек Jetpack, які відповідають за підвищення надійності системи, спрощують життєвий цикл компонентів та забезпечують підтримку кращих практик розробки. У проєкті були задіяні такі ключові компоненти:

ViewModel [38] є елементом рівня бізнес–логіки та відповідає за збереження стану UI незалежно від змін конфігурації (поворот екрана, зміна мови, темна тема тощо). Він:

— Інкапсулює логіку обчислення параметрів здоров’я: добової норми калорій, водного балансу, складу раціону.

— Переживає життєвий цикл Activity/Fragment, виключаючи втрату критичних даних.

— Мінімізує кількість звернень до локальної БД, організовуючи кешування стану.

Це дозволяє підвищити UX: дані харчових записів і рекомендацій завжди доступні й без затримок.

Flow [39] використовується як реактивний механізм обміну даними між ViewModel та інтерфейсом. Переваги застосування Flow:

— асинхронність і підтримка backpressure – критично для мобільного середовища,

— життєвий цикл–aware підписки через repeatOnLifecycle – відсутність витоків пам’яті,

— легка інтеграція з Room, де DAO повертають Flow–стріми даних.

Приклади використання Flow для трансляції наведено в таблиці 3.1.

Таблиця 3.1 – Використання Flow

Дані	**Джерело**	**Реакція UI**
Добові записи харчування	Room DAO	Автооновлення списку страв
Калорійність та БЖВ	ViewModel–обчислення	Динамічний перерахунок прогрес–барів
Рекомендований водний баланс	Алгоритмічний модуль	Нагадування та індикатор досягнення норми

Це забезпечує повну реактивність інтерфейсу без ручної синхронізації станів.

Navigation Component [40] є центральним механізмом керування переходами між екранами застосунку. Його використання забезпечує чітку структуру навігаційних сценаріїв та зводить до мінімуму ризики появи помилок, пов’язаних з передачею параметрів, життєвим циклом фрагментів або неправильним формуванням історії переходів.

Система має модульну побудову інтерфейсу, де кожен функціональний блок – «Харчування», «Водний баланс», «Профіль», «Налаштування» – реалізований як окремий фрагмент або група фрагментів. Navigation Component забезпечує керовану комунікацію між ними без жорстких залежностей, що відповідає принципам розділення відповідальності.

Використання Safe Args [40] дозволяє передавати параметри між екранами у строго типізованому вигляді. Наприклад, під час відкриття екрану редагування харчового запису передаються унікальний ідентифікатор продукту та вибрана дата. Безпечна типізація гарантує, що параметр буде переданий коректно, без необхідності ручної серіалізації.

Переваги Navigation Component наведено в таблиці 3.2

Таблиця 3.2 – Переваги Navigation Component

Можливість	**Роль у застосунку**
Контрольований back stack	Запобігає дублюванню екранів при багаторазових переходах, забезпечує правильне повернення на попередній крок
Deep link інтеграція	Наприклад: натиснення на push–сповіщення «Пора випити воду» → відкриття відповідного модуля
Graph–based конфігурація	Вся навігація централізовано описана у nav_graph.xml, що спрощує супровід і розширення
Підтримка анімацій та транзакцій	Покращення UX при переході між різними функціональними зонами
Навігація з урахуванням життєвого циклу	Уникає крашів при одночасних операціях UI і бази даних

Застосунок містить Navigation Host Fragment, який є єдиним контейнером для розміщення різних UI–екранів. Навігація викликається виключно через

NavController, що виключає використання небезпечних транзакцій FragmentManager вручну.

Візуалізація графів навігації наведена на рисунку 3.2.

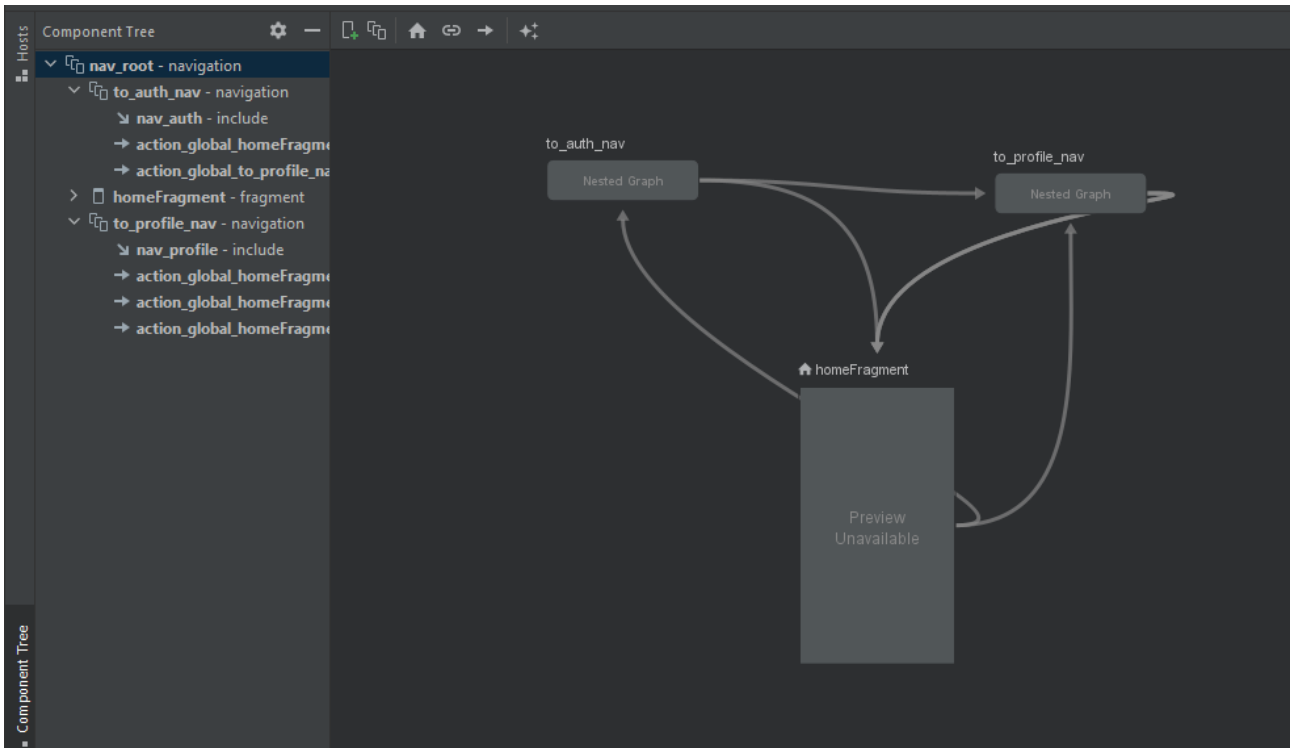


Рисунок 3.2 – Візуалізація графів навігації

ViewModel ініціює навігаційні події через одноразові навігаційні команди, які передаються до UI у вигляді Flow–подій. Такий підхід усуває повторні переходи при повороті екрану або відновленні Activity.

Navigation Graph легко масштабувати для нових функцій, наприклад:

— модуль «Аналіз харчування»;

— агрегована аналітика добових норм;

— соціальні механізми мотивації.

Для додавання нового екрану достатньо оновити граф – без змін у вже реалізованих модулях. Це знижує технічний борг і прискорює розширення застосунку.

Room Persistence Library – критичний компонент у забезпеченні автономності та цілісності даних мобільної системи.

Room використовується як повноцінний інструмент для організації локального сховища з рівнем абстракції над SQLite. Це дозволяє уникати низькорівневих помилок у роботі з БД та забезпечує просте розширення моделі даних при масштабуванні.

У локальній базі застосунку зберігаються:

— дані профілю користувача: антропометричні показники, рівень фізичної активності, індивідуальні цілі щодо калорій та водного балансу;

— харчова база продуктів і страв з характеристиками поживності (білки, жири, вуглеводи, калорійність);

— історія прийомів їжі та споживання води, що використовується для формування статистики і рекомендацій;

— кеш зовнішніх джерел (наприклад, поповнення бази продуктів через API або сканування штрихкодів).

Функціональні переваги використання Room:

1. DAO працюють із Flow, що забезпечує реактивність: будь–які зміни в БД одразу відображаються на UI без необхідності ручного оновлення.
2. SQL–запити проходять компіляційну перевірку – виявлення помилок на ранньому етапі збільшує надійність релізів.
3. Підтримка міграцій дає змогу без втрати даних адаптувати схему БД при зростанні функціональності (додавання таблиць для трекінгу мікронутрієнтів, кастомних планів харчування тощо).
4. Робота в офлайн–режимі дозволяє користувачу вести харчовий щоденник без доступу до інтернету, а синхронізація виконується при відновленні мережі.
5. Індексація та оптимізовані запити забезпечують швидкий доступ до великих обсягів історичних записів.

Таким чином Room виступає ядром моделі даних системи, забезпечуючи:

— надійну й узгоджену роботу даних;

— високу продуктивність;

— актуальність UI у режимі реального часу;

— можливість безризикового розвитку функціональності у майбутньому.

Для забезпечення швидкого та зручного наповнення харчової бази даними застосунок інтегрується з глобальним відкритим ресурсом OpenFoodFacts API. Це API містить міжнародний каталог продуктів із маркуванням штрих–кодів (EAN/UPC) та повним набором харчових характеристик: енергетична цінність, БЖВ, алергени, маркування Nutri–Score, інгредієнти, рівень обробки (NOVA) [35].

Основні можливості інтеграції:

1. Сканування штрих–коду дозволяє користувачу миттєво знаходити продукт у базі без ручного введення;
2. Автоматичне заповнення харчового профілю продукта мінімізує помилки та прискорює реєстрацію прийому їжі;
3. Підтримка багатомовності API забезпечує релевантну локалізовану інформацію;
4. Використання стандартизованих харчових показників підвищує точність сформованої статистики та рекомендацій.

Технологічна реалізація взаємодії з API. Для виконання мережевих операцій використовується стек:

1. Retrofit – декларативний HTTP–клієнт із автоматичною десеріалізацією JSON → Kotlin–об’єктів;
2. Kotlin Coroutines – асинхронна обробка без блокування UI–потоку;
3. OkHttp – низькорівнева основа для стабільного та швидкого мережевого обміну.

Переваги такого підходу:

— підтримка suspend–функцій: мережеві запити не спричиняють затримок інтерфейсу;

— зручне керування помилками (timeout, відсутність інтернету, некоректні дані);

— можливість кешування результатів у Room для офлайн–доступу;

— масштабованість – легко додати інші API (наприклад, бази мікронутрієнтів, рецептурні каталоги).

Потік обробки даних після сканування:

1. Користувач сканує штрих–код із використанням камери (ML Kit / ZXing).
2. ViewModel ініціює корутинний запит до OpenFoodFacts API за кодом продукту.
3. Retrofit завантажує та десеріалізує дані у модель ProductDto.
4. Дані валідовано (перевірка наявності Nutri–Score, калорійності, БЖВ).
5. Результат:
   — або збереження до локальної БД Room,
   — або відображення попереднього перегляду, якщо дані потребують підтвердження користувача.
6. UI автоматично оновлюється через Flow.

Таким чином система отримує:

— високу точність наповнення харчового каталогу;

— мінімальне залучення користувача, що стимулює сталість поведінкових змін;

— масштабованість і підтримку глобальних стандартів харчових даних.

Android Studio [42] застосовується як єдиний центр розробки, тестування та оптимізації застосунку, забезпечуючи повноцінну інтеграцію всіх необхідних інструментів для професійного Android–інжинірингу. IDE містить потужний редактор коду з контекстними підказками, автоматичним рефакторингом, аналізом якості та засобами запобігання помилкам ще на етапі розробки. Завдяки глибокій інтеграції з екосистемою Kotlin та бібліотеками Jetpack, Android Studio дає можливість швидко створювати компоненти UI, працювати з ViewBinding/Compose–попереднім переглядом та тестувати сценарії взаємодії користувача без запуску на фізичному пристрої [21].

Важливою частиною платформи є вбудований Android Emulator, який моделює різноманітні конфігурації пристроїв: різні розміри екранів, версії Android, типи процесорів та апаратні функції, включно з GPS, сенсорами та системою камер. Це дозволяє заздалегідь перевірити поведінку застосунку в реальних умовах експлуатації, у тому числі стійкість до обриву мережі, зміни орієнтації чи браку системних ресурсів.

Інтерфейс Android Studio наведено на рисунку 3.3

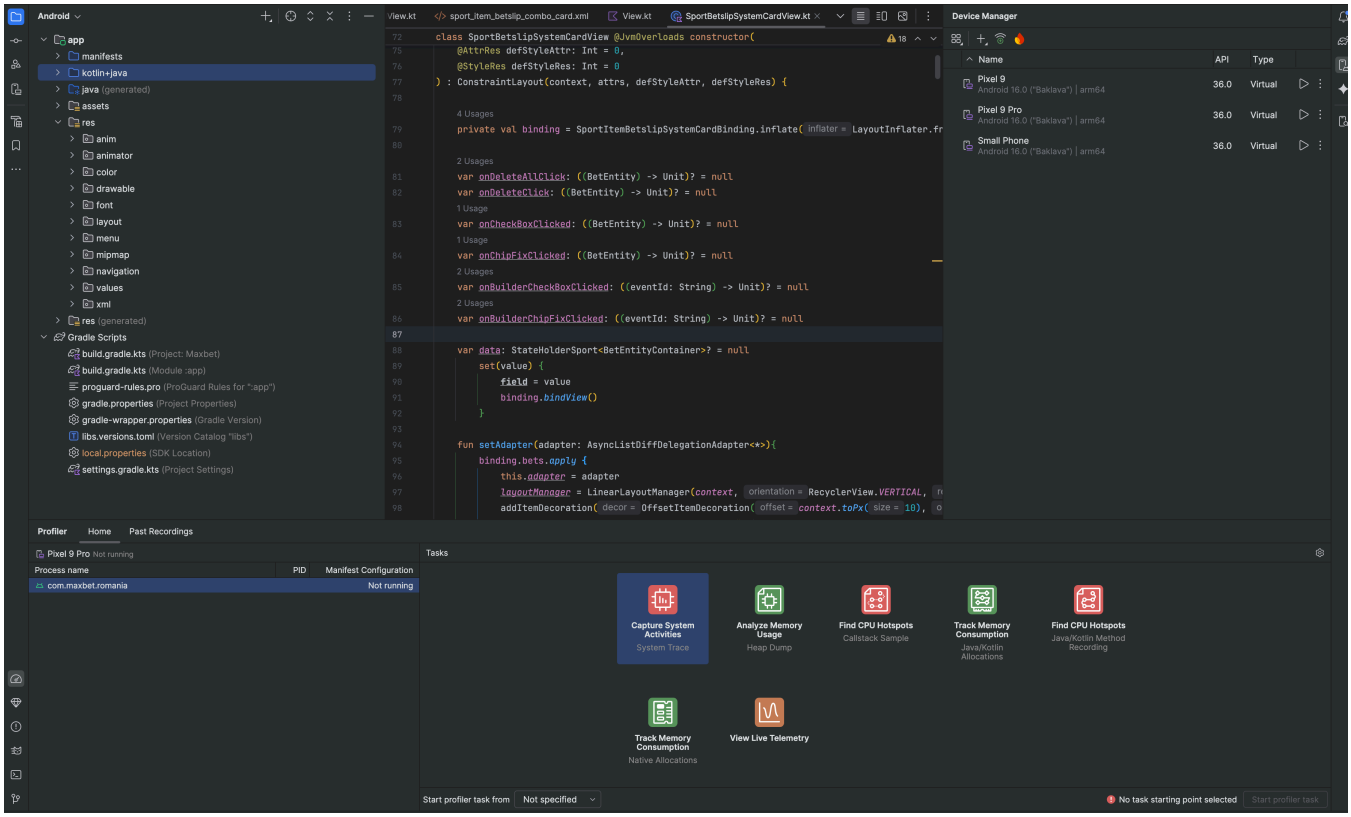


Рисунок 3.3 – Інтерфейс Android Studio

Серед засобів контролю продуктивності особливе місце займають профайлери CPU, пам’яті, GPU та мережевого трафіку. Вони забезпечують точну діагностику «вузьких місць», аналіз споживання енергії та оптимізацію плавності інтерфейсу, що є критичним для мобільних рішень зі складною бізнес–логікою. Інструменти статичного аналізу, такі як Lint та інтеграція з плагіном Firebase Crashlytics, допомагають упереджувати помилки, контролювати відповідність рекомендаціям Material Design і гарантувати безпеку даних.

Android Studio також забезпечує комплексну автоматизацію процесу складання застосунку завдяки вбудованій підтримці Gradle – офіційної системи build–менеджменту для Android. Gradle керує всіма артефактами проєкту: залежностями, ресурсами, конфігураціями збірки, оптимізацією та підписом APK/AAB–пакетів. Розмежування build–variant та build–type дозволяє паралельно підтримувати розробницькі (debug) і релізні конфігурації, застосовуючи різні рівні оптимізації, мінімізації коду (R8/ProGuard), шифрування ресурсів і засоби захисту від реверс–інжинірингу.

Інтегрований dependency–manager автоматично оновлює бібліотеки, контролює їхню сумісність і усуває конфлікти версій, що особливо важливо при роботі з Jetpack–компонентами та мережевими фреймворками, такими як Retrofit або Ktor. Gradle також підтримує модульність проєкту, розділяючи його на логічні підмодулі (core, data, ui), що покращує повторне використання коду й пришвидшує збірку завдяки інкрементному компілюванню.

У частині тестування Android Studio надає інструментарій для автоматизації unit–тестів (JUnit, Mockito), UI–тестів (Espresso) та профілактики регресій. CI/CD–інтеграції (GitHub Actions, Jenkins, Firebase App Distribution) вбудовуються без додаткових налаштувань, дозволяючи збирати, тестувати й доставляти застосунок на тестові пристрої або в магазин програм автоматизовано.

Для аналізу сумісності з апаратним забезпеченням Android Studio використовує валідацію ресурсів та систем залежностей, а також надає інструменти для розрахунку розміру пакету й виявлення надлишкових елементів у фінальному білді. Це допомагає підтримувати легкий, продуктивний застосунок, що відповідає вимогам користувачів і Google Play.

У підсумку Android Studio забезпечує не лише створення функціонального прототипу, а повний DevOps–цикл мобільної розробки: планування, розробку, тестування, оптимізацію, складання, підпис, поширення й подальшу підтримку. Завдяки цьому вибір Android Studio є ключовим фактором якості, безпечності та конкурентоздатності системи підтримки здорового харчування.

Таким чином, обраний технологічний стек, який включає мову Kotlin, Android SDK, Jetpack–бібліотеки та Android Studio, створює міцну основу для розробки мобільної системи підтримки здорового способу харчування. Kotlin забезпечує безпечне та ефективне програмування, дозволяючи зменшити кількість помилок, підвищити читабельність коду та реалізувати сучасні підходи до асинхронної обробки даних за допомогою корутин. Android SDK гарантує безпосередню взаємодію застосунку з системними ресурсами, сенсорами та службами безпеки пристрою, що критично важливо для коректного збору даних про користувача та його активність.

Jetpack–бібліотеки забезпечують стійкість архітектури, управління життєвим циклом компонентів, централізовану навігацію між екранами та надійну роботу з локальними базами даних. Flow дає змогу реактивно обробляти зміни даних у реальному часі, що дозволяє автоматично оновлювати інтерфейс без додаткових витрат ресурсів [28]. Room гарантує надійне збереження даних користувача та історії харчування, а Navigation Component із Safe Args забезпечує передбачувану та безпечну навігацію між численними модулями системи.

Android Studio надає повний спектр інструментів для розробки, тестування, профілювання та збірки застосунку, включно з управлінням залежностями через Gradle, модульною структурою проекту, інструментами CI/CD та емуляторами різних конфігурацій пристроїв [29]. Це дозволяє розробникам швидко впроваджувати новий функціонал, тестувати його в різних сценаріях і забезпечувати стабільність та безпеку системи.

В підсумку, обраний стек технологій дозволяє створити високопродуктивний, масштабований та надійний мобільний застосунок, який здатний ефективно працювати з персональними даними користувача, забезпечувати інтеграцію з зовнішніми харчовими базами та надавати користувачеві сучасний нативний досвід у сфері цифрового здоров’я. Такий підхід відповідає сучасним стандартам розробки Android–застосунків і гарантує реалізацію всіх поставлених функціональних та нефункціональних вимог проєкту.

3.2 Розробка модуля управління профілем користувача

Модуль управління профілем користувача є ключовим елементом мобільної системи підтримки здорового харчування, оскільки саме на основі персональних антропометричних характеристик розраховується індивідуальна добова норма калорій і водного балансу. Модуль забезпечує введення, редагування та збереження персональних параметрів, а також їх валідацію відповідно до медичних норм.

Основною інтеракцією з модулем виступає екран профілю, реалізований у вигляді ProfileFragment, який забезпечує доступ до таких персональних даних: вік, стать, зріст, вага, рівень фізичної активності та ціль харчування.

Бізнес–логіка та управління станом розміщені у ProfileViewModel, що дозволяє зберігати дані між змінами конфігурації пристрою та розмежовувати UI і внутрішні процеси. ViewModel працює з локальним джерелом даних через UserRepository, який інкапсулює взаємодію з Room та гарантує цілісність і узгодженість даних.

Функціональні можливості ProfileViewModel охоплюють кілька основних груп операцій:

— завантаження профілю при запуску екрана через реактивний потік Flow, який негайно доставляє актуальні дані до UI;

— обробку змін окремих параметрів користувача з локальним збереженням стану до моменту підтвердження;

— валідацію введених значень (перевірка діапазонів віку, ваги, зросту, відповідності цілі та активності);

— автоперерахунок індивідуальних норм води та калорій на основі формул харчової дієтології;

— ініціацію асинхронного збереження профілю в локальну базу із забезпеченням цілісності даних.

Таке компонування дозволяє ProfileViewModelграти роль єдиного центру управління станом модуля, оскільки саме вона приймає рішення про коректність і завершеність профілю перед передачею в інші підсистеми.

Функціональні елементи ProfileViewModel та їх призначення:

— userProfileState: StateFlow<UserProfileUI> – Актуальний стан профілю, який завжди синхронізований із UI. Реактивно оновлюється при зміні будь–якого з параметрів користувача.

— isProfileValid: StateFlow<Boolean> – Індикатор готовності профілю до збереження. Використовується для контролю активності кнопки підтвердження.

— validationResult: StateFlow<ValidationState> – Містить деталізований стан валідації для кожного поля. Забезпечує інформування інтерфейсу про помилки вводу з можливістю контекстного відображення.

— bmiState: StateFlow<Double> – Дає змогу UI відображати індекс маси тіла та використовувати його у рекомендаційних блоках.

— dailyCalories: StateFlow<Int> – Результат алгоритму розрахунку норми калорій, що використовується для формування раціону.

— dailyWater: StateFlow<Int> – Визначена добова норма водоспоживання, інтегрована з модулем водного трекінгу.

— loadUserProfile(): Job – Створює Flow–підписку на UserRepository та Room–таблицю користувача. Забезпечує отримання останніх даних при кожному повторному відкритті профілю.

— onFieldChanged(field, value): Unit – Оновлює стан кожного окремого параметра локально в ViewModel, зберігаючи реактивність і уникнення зайвих записів у базу.

— validateField(field, value): Boolean – Перевіряє допустимість значення згідно з фізіологічними нормами та бізнес–логікою (мінімальний вік, реалістичний зріст і вага тощо).

— calculateDailyNorms(): Unit – Перераховує добові норми калорій і води після кожної зміни параметрів. Викликається прозоро для користувача.

— calculateBMI(): Double – Розширює функціонал профілю для майбутніх моделей фітнес–прогресу.

— saveProfile(): Job – Записує валідовані дані до локальної бази. Тригерить оновлення підписаних на ці дані модулів.

— resetProfile(): Unit – Забезпечує очищення профілю у разі відновлення до початкових параметрів або зміни власника пристрою.

— updateActivityLevel(level: ActivityLevel): Unit – Опрацьовує зміну категорії фізичної активності та одразу впливає на розрахунок норм калорій.

— updateGoal(goal: UserGoal): Unit – Застосовується для зміни цілі харчування з миттєвим впливом на розрахункові моделі.

— mapToEntity(model: UserProfileUI): UserEntity – Перетворює бізнес–дані у формат, придатний для збереження у Room.

— mapToUI(entity: UserEntity): UserProfileUI – Надає UI–адаптацію даних, отриманих із репозиторію.

З боку даних модуль використовує сутність UserEntity, що відображається в Room як таблиця локальної бази. Для переходу між внутрішнім та UI–представленням застосовується патерн “DTO + Mapper”: Room сутності перетворюються у доменну модель UserProfile, зручну для відображення та обчислень. Це дозволяє уникнути прив’язки UI до структури бази даних і забезпечує можливість подальшої еволюції моделі без ризику порушення роботи інтерфейсу.

Repository відповідає за всю взаємодію з Room DAO і надає ViewModel уніфікований API для:

— отримання профілю користувача як Flow, що гарантує автоматичне оновлення UI при зміні локальних даних;

— виконання транзакційних операцій оновлення профілю;

— асинхронної роботи без блокування головного потоку, застосовуючи Kotlin Coroutines.

Функціональні елементи UserRepository та їх призначення

— userDao: UserDao – Інкапсульований доступ до Room–шару. Забезпечує виконання CRUD–операцій над сутністю UserEntity через строго типізований DAO–інтерфейс.

— getUserProfile(): Flow<UserEntity?> – Повертає реактивний потік локальних даних профілю. Будь–яка зміна в таблиці автоматично ретранслюється у ViewModel → UI без ручного оновлення.

— insertOrUpdateUser(user: UserEntity): suspend Unit – Виконує транзакційне збереження або модифікацію профілю. Гарантує узгодженість даних та виконується поза UI–потоком.

— clearUserProfile(): suspend Unit – Забезпечує повне видалення профілю з локального сховища – використовується при деавторизації або зміні користувача.

— mapToDomain(entity: UserEntity): UserProfile – Перетворює Room–сутність у доменну модель, що використовується для бізнес–логіки та розрахунків.

— mapToEntity(profile: UserProfile): UserEntity – Виконує адаптацію даних, які ViewModel генерує для збереження, у формат, сумісний із Room.

— updateWeight(weight: Double): suspend Unit – Оновлює вагу окремо без зміни інших полів, що дозволяє швидко застосовувати зміни Daily–бази без перезапису всього профілю.

— updateActivityLevel(level: ActivityLevel): suspend Unit – Оновлює рівень фізичної активності в окремому запиті, спрощуючи сценарії поступової зміни профілю.

— profileExists(): Boolean – Оперативно визначає наявність запису в локальному сховищі та застосовується під час першого запуску застосунку для навігаційних рішень.

— safeTransaction(block: suspend () → Unit): Result – Уніфікований механізм захисту даних від часткового запису та можливих помилок I/O, з ретельним логуванням.

Архітектура модуля відповідає принципам Clean Architecture та передбачає односторонній потік даних:

1. UI → ViewModel – Користувач змінює параметри профілю → ViewModel обробляє введення.

2. ViewModel → Repository – Після валідації та перерахунків ViewModel викликає транзакційні операції збереження.

3. Repository → Room (DAO) – Repository формує відповідні запити, абстрагуючи ViewModel від деталей зберігання.

4. Room → Flow → ViewModel → UI – Будь–яка зміна в таблиці UserEntity тригерить реактивне оновлення, що зовні виглядає як миттєве відображення актуального стану.

Це забезпечує миттєву реакцію інтерфейсу на будь–які модифікації даних – наприклад, після зміни ваги перераховані норми гідратації та калорій автоматично відображаються на екрані.

Реалізовані механізми також передбачають блокування неповного профілю: у разі відсутності обов'язкового параметра інші модулі (живлення, трекінг води) не використовують некоректні дані, а спрямовують користувача до доповнення свого профілю.

Завдяки багаторівневій архітектурі модуль легко масштабувати. У майбутніх версіях можуть бути додані розширені параметри (відсоток жирової маси, дані зі смарт–годинників), не потребуючи модифікації системної логіки взаємодії шарів.

Результатом впровадження модуля є створення єдиного, верифікованого та консистентного джерела антропометричних даних користувача, яке синхронізує локальне сховище з віддаленим бекендом, забезпечує актуальність інформації у реальному часі завдяки реактивній моделі оновлення та супроводжує кожну зміну перевірками доменної логіки для запобігання некоректних значень. Такий підхід унеможливлює частково виконані або розсинхронізовані оновлення, гарантує транзакційність роботи з профілем та створює стандартизований інтерфейс для всіх сервісів персоналізації харчування й оцінки стану здоров'я. Модуль стає ядром прийняття рішень: від точності цих даних залежить коректність обчислень, якість рекомендацій, адаптивність функціоналу під кожного користувача й загальна довіра до результатів, що надає застосунок.

UML-діаграма класів модуля управління профілем користувача наведено на рисунку 3.1.

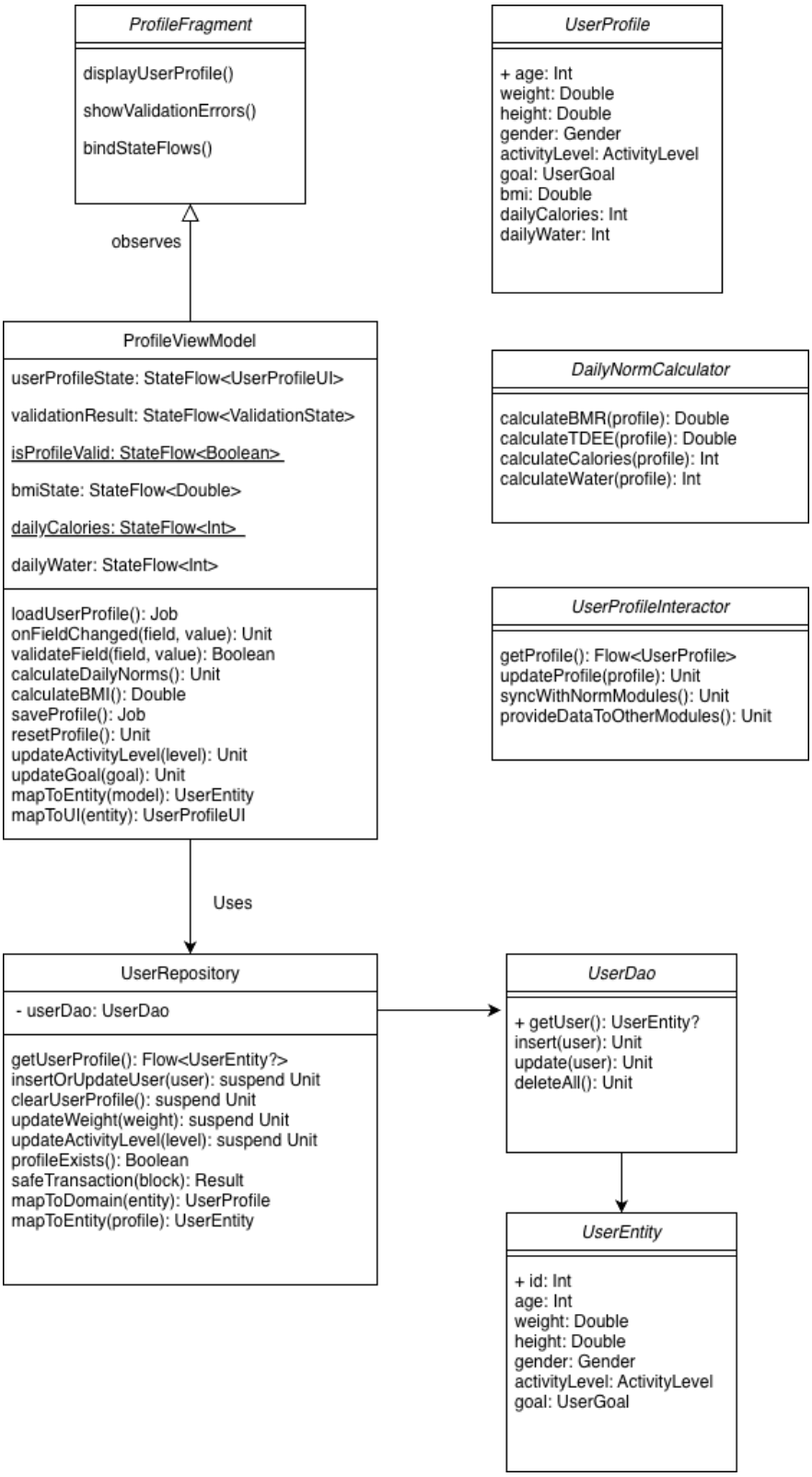


Рисунок 3.1 – UML-діаграма класів модуля управління профілем користувача

3.3 Розробка модуля роботи з харчовими даними та інтеграції з OpenFoodFacts

Модуль роботи з харчовими даними реалізує комплекс процедур, які забезпечують формування персоналізованих рекомендацій щодо харчування на основі актуальних даних користувача та глобальних харчових каталогів. Архітектура модуля спроектована таким чином, щоб забезпечити високу розширюваність та гнучкість, що дозволяє у майбутньому інтегрувати додаткові джерела даних, аналітичні сервіси або корпоративні провайдери харчової інформації без суттєвого впливу на існуючу структуру.

Основним ядром модуля є інтеграція з відкритою базою OpenFoodFacts, яка надає доступ до глобального каталогу продуктів із детальним описом поживної цінності, наявності алергенів, складу інгредієнтів, маркування Nutri–Score та інших параметрів, необхідних для точного аналізу харчового раціону. Взаємодія з API здійснюється через REST–запити з використанням Retrofit та Kotlin coroutines, що дозволяє виконувати всі мережеві операції асинхронно і без блокування інтерфейсу. Для підвищення надійності взаємодії впроваджено механізми кешування результатів запитів, повторних спроб у разі помилок мережі та контролю доступності сервісу.

Отримані дані піддаються процедурі нормалізації та валідації, включаючи перевірку коректності харчових показників, уніфікацію одиниць вимірювання та агрегацію інформації за категоріями продуктів. Після цього дані передаються у FoodViewModel, де відбувається їх адаптація до UI–моделей і формування потокового стану за допомогою Kotlin Flow. Такий підхід гарантує, що інтерфейс користувача завжди працює з актуальною інформацією, а всі критичні операції виконуються у фоновому режимі без ризику заморожування або затримок.

Модуль підтримує двосторонній обмін даними: користувач може додавати власні продукти у локальну базу, створювати персональні шаблони страв та фіксувати історію споживання з прив’язкою до часу доби. Це дозволяє накопичувати повні та достовірні дані для подальшого аналізу нутрієнтів, оцінки динаміки ваги та генерації персоналізованих рекомендацій, що безпосередньо

пов’язані з профілем користувача. Завдяки такій інтеграції система здатна підтримувати адаптивний та науково обґрунтований підхід до контролю харчування, забезпечуючи користувача точними і практично корисними рекомендаціями.

Для роботи модуля, розроблено компонент FoodViewModel. Центральний координатор бізнес–логіки модуля, який керує станом UI, викликає мережеві та локальні операції, забезпечує агрегацію і нормалізацію вхідних даних.

Основні об’єкти:

— foodSearchState: StateFlow<FoodSearchUI> – Актуальний стан пошуку продуктів: прогрес, результати, повідомлення про помилки.

— selectedProduct: StateFlow<FoodProductUI?> – Деталі вибраного продукту для перегляду та додавання у раціон.

— recentFoods: StateFlow<List<FoodEntity>> – Локальна історія вибору продуктів користувача.

Ключові методи:

— searchFood(query: String) – Ініціює пошук у зовнішньому API через Repository, обробляє помилки мережі, нормалізує відповідь.

— loadRecentFoods()– Підписується на локальні дані Room, оновлює список останніх продуктів.

— selectFood(productId: String) – Завантажує деталі вибраного продукту та оновлює стан selectedProduct.

— addFoodToDiary(product: FoodEntity, datetime: Long) – Передає продукт та кількість у Repository для фіксації споживання.

— cancelSearch() – Скидає пошукові результати та стан в інтерфейсі (UX–контроль).

Окрім FoodViewModel, розроблено та імплементовано компонент FoodRepository. Уніфікований шар доступу до двох джерел – локального (Room) та мережевого (OpenFoodFacts API). Гарантує потокобезпечність і контроль доступності.

Основні функції:

— getLocalRecentFoods(): Flow<List<FoodEntity>> – Потік для UI з автоматичним оновленням.

— searchFoodRemote(query: String): Result<List<FoodRemoteModel>> – Відправляє запит до API, застосовує політику retry + timeout, модерує HTTP–помилки.

— getProductDetails(productId: String): Result<FoodRemoteModel> – Завантаження додаткової інформації (алергени, таблиця нутрієнтів).

— saveFoodToLocal(food: FoodEntity) – Зберігає продукт у локальну базу – кешування + останні дії користувача.

— logFoodConsumption(foodId: Long, time: Long, weight: Int) – Фіксація добового споживання для подальшого аналізу нутрієнтів.

Repository абстрагує джерела даних і підвищує стійкість системи до перебоїв API.

Room DAO рівень забезпечує швидкий доступ до контенту без мережевої взаємодії.

Три окремі DAO:

— FoodDao();

— insertFood();

— getFoodByBarcode();

— getRecentFoods(limit: Int);

— DiaryDao();

— insertDiaryRecord();

— getDiaryByDate(date: Long): Flow<List<DiaryEntry>>;

— NutritionDao;

— CRUD для внутрішніх нутрієнтних таблиць (індекс Nutri–Score, енергетичні коефіцієнти).

DAO підтримують транзакції для цілісності обліку.

OpenFoodFactsApiService – тонкий HTTP–клієнт, побудований на Retrofit + Kotlin Serialization.

Методи REST–взаємодії:

— searchProducts(query: String) – Віддає список продуктів для швидкого пошуку.

— getProductByBarcode(barcode: String) – Доступ до розширеної специфікації товару.

Підтримка JSON–моделей з мапуванням на Room–сутності через DTO–конвертери.

Маппери – Окремий шар перетворення для зниження зв'язності:

— Remote → Entity;

— Entity → UI;

— Remote → UI (fallback).

Забезпечують адаптацію різних структур без дублювання логіки у VM/Repository.

Результат інтеграції – компоненти формують єдиний, послідовний життєвий цикл харчових даних: від пошуку → до локального кешу → до аналізу нутрієнтів → до персональних рекомендацій.

Модуль підтримує офлайн–роботу, еластичний перехід між джерелами та скорочує затримку від моменту запиту до відображення результатів. Завдяки потоковому підходу (Flow), UI завжди працює з найактуальнішим станом без додаткових оновлень.

Кінцевим результатом роботи модуля стає повноцінна інформаційна база харчових продуктів, яка постійно оновлюється та класифікується за основними харчовими показниками, категоріями і властивостями, що необхідні для детального аналізу раціону користувача. Ця база забезпечує швидкий та безпечний доступ до даних, дозволяючи системі миттєво отримувати потрібну інформацію для розрахунку калорійності, вмісту білків, жирів і вуглеводів, а також для аналізу потенційних алергенів чи Nutri–Score. Завдяки такій організації даних підвищується точність персоналізованих рекомендацій, адже алгоритми системи оперують достовірними і структурованими показниками, що суттєво знижує ймовірність помилок у підборі харчового раціону. Крім того,

модуль формує основу для комерційного розвитку застосунку, надаючи можливість інтегрувати нові сервіси, додавати платні функції, наприклад, персоналізовані дієтичні плани, аналітику споживання або корпоративні каталоги продуктів, що робить екосистему застосунку більш привабливою для користувачів та партнерів.

UML-діаграма класів модуля роботи з харчовими даними та інтеграції з OpenFoodFacts наведена на рисунку 3.2.

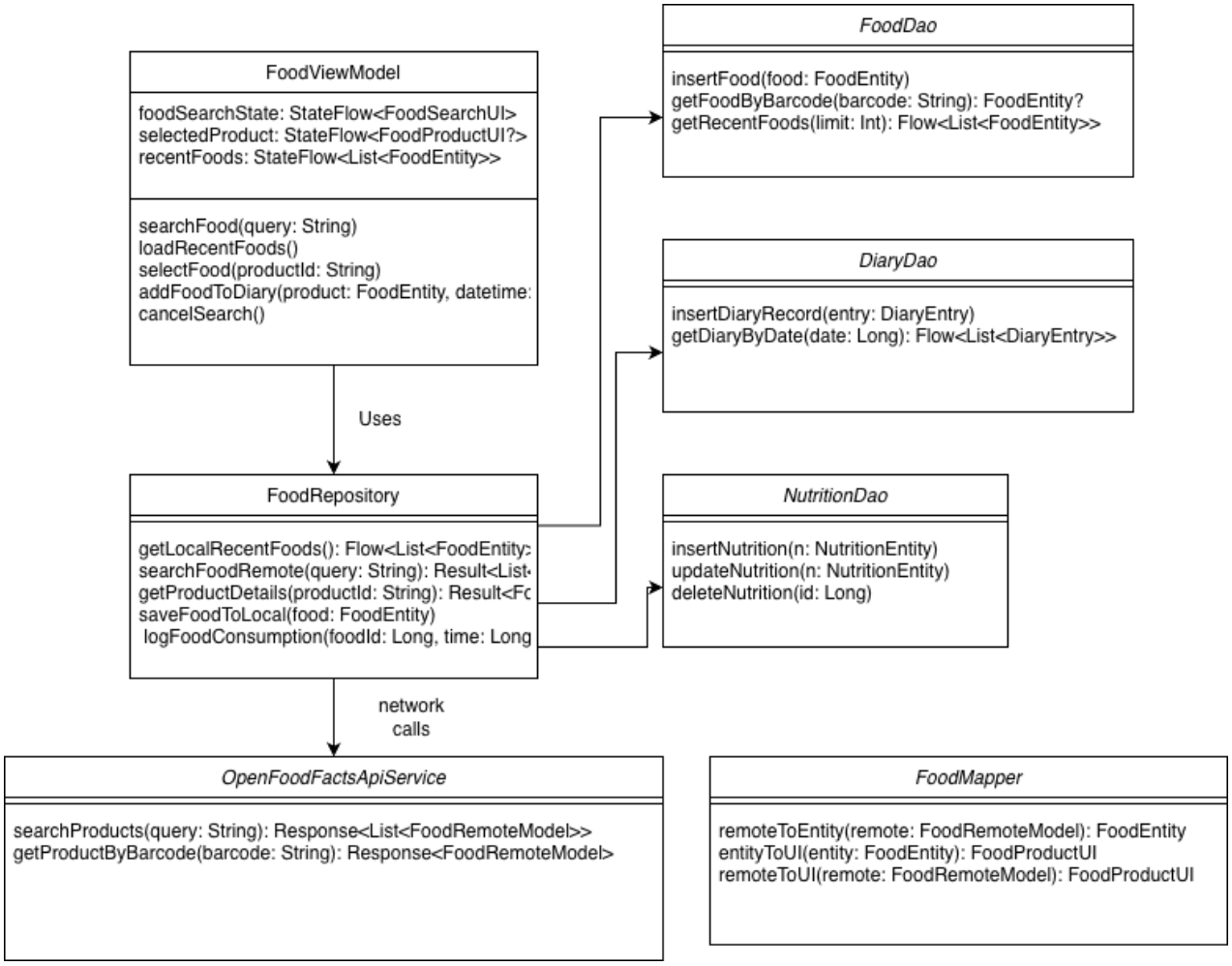


Рисунок 3.2 – UML-діаграма класів модуля роботи з харчовими даними та інтеграції з OpenFoodFacts

3.4 Розробка модуля підбору денного раціону

Модуль підбору денного раціону є одним із ключових інтелектуальних компонентів системи, оскільки він безпосередньо відповідає за формування

персоналізованого набору продуктів та страв відповідно до індивідуальних характеристик користувача. Архітектура модуля орієнтована на гнучку логіку прийняття рішень, модульність та можливість подальшого розширення алгоритмів у напрямку нутріціології та машинного навчання.

Модуль підбору денного раціону базується на детальній персоналізації, де ключову роль відіграють вхідні антропометричні параметри користувача. Дані про вік, стать, вагу, зріст, рівень фізичної активності та вибрану харчову ціль надходять у модуль безпосередньо з ProfileViewModel, яка виступає єдиним достовірним джерелом цих характеристик. На основі цього профілю система виконує первинні обчислення: визначає добову норму калорій (TDEE) із застосуванням формул Mifflin–St Jeor або Harris–Benedict, коригує її залежно від активності, а також задає цільовий розподіл макронутрієнтів – білків, жирів та вуглеводів – відповідно до вибраної стратегії (схуднення, підтримка чи набір ваги). Результатом цього блоку розрахунків є чіткий набір числових обмежень, який слугує основою для формування денного меню.

Після встановлення цільових параметрів модуль переходить до роботи з локальним сховищем Room. У базі даних зберігаються як продукти, отримані з OpenFoodFacts і нормалізовані під внутрішню структуру системи, так і власні записи користувача, включно з кастомними продуктами або домашніми стравами. DAO, розроблені спеціально для цього модуля, дозволяють сформувати структуровану вибірку – перелік продуктів, які підходять за калорійністю, належністю до категорії (зернові, білкові, овочі тощо), профілем макронутрієнтів та доступністю для конкретного прийому їжі. Застосовуються також додаткові обмеження і фільтри, наприклад врахування індексу насичення, щільності поживних речовин або рекомендованого глікемічного навантаження, якщо ці параметри додані у базу. Така побудова DAO–запитів забезпечує отримання якісно підготовленої вихідної множини даних без необхідності надмірної обробки у ViewModel.

Описаний підхід дозволяє не просто підібрати набір продуктів, а створити по–справжньому адаптивну систему, що реагує на індивідуальні потреби

користувача. Враховуючи те, що в структурі Room передбачено можливість зберігання додаткових метаданих – таких як персональні вподобання, наявність харчових алергенів, категорії обмежень (вегетаріанство, безлактозна дієта, низьковуглеводне харчування) – модуль уже на архітектурному рівні готовий до масштабування. Це означає, що майбутнє розширення функціональності не вимагатиме перегляду всієї логіки підбору раціону, а лише додавання нових параметрів до моделі продукту або до профілю користувача.

Таким чином, модуль трансформує статичні антропометричні дані та велику множину харчових елементів у структурований набір релевантних кандидатів для побудови збалансованого денного меню, забезпечуючи високу персоналізацію та гнучкість рекомендацій.

Алгоритм підбору денного раціону реалізований як багатоступеневий процес із чітким поділом відповідальностей між обчислювальною логікою, модулем доступу до даних і ViewModel. На першому етапі система формує цільовий добовий профіль – це сукупність калорійності та пропорцій БЖВ, які користувач має отримати протягом дня. Розрахунок базується на даних профілю, що надходять із ProfileViewModel, та формулі енергетичних витрат. Після визначення цільових значень модуль переходить до їхнього декомпонування на окремі прийоми їжі: для сніданку задається підвищений коефіцієнт білків і складних вуглеводів, обід переорієнтовується на більш тривале насичення за рахунок повільних вуглеводів і рослинних жирів, а вечеря – на легші, низькокалорійні продукти з мінімальним навантаженням на метаболізм. Такий розподіл дозволяє сформувати початковий енергетичний каркас дня.

Далі система звертається до структурованих вибірок Room, що містять очищені, нормалізовані та проаналізовані харчові продукти. DAO–функції повертають вузькофокусовані підмножини продуктів, які відповідають конкретним критеріям: високобілкові варіанти для сніданку, джерела складних вуглеводів для обіду або низькокалорійні позиції для вечері. На основі цієї інформації модуль формує попередню матрицю добового меню – список продуктів–кандидатів, структурований за прийомами їжі. Це ще не фінальне

рішення, а проміжний робочий шар, що відображає первинний розподіл поживних речовин.

Після формування початкової матриці запускається корекційний етап. Алгоритм проводить повторну оцінку сумарної калорійності та макронутрієнтів, порівнюючи їх із цільовими параметрами. Якщо денне значення виходить за межі допустимого діапазону, застосовується кілька механізмів уточнення: заміна висококалорійних продуктів на легші аналоги, корекція білкової чи жирової групи, добір додаткових вуглеводних елементів для покриття дефіциту або видалення надлишкових позицій. Кожна корекція виконується автоматично, із дотриманням харчової логіки та збереженням збалансованої структури меню.

Ключовою складовою роботи алгоритму є реактивність: усі проміжні стани формуються та передаються у ViewModel за допомогою Flow. Це означає, що будь–яка зміна у профілі користувача, локальній базі продуктів чи правилах розподілу поживних речовин миттєво відображається в оновлених рекомендаціях. UI отримує дані практично без затримок і може показувати користувачеві актуальний раціон у реальному часі, без очікування повного перерахунку або ручного оновлення. Завдяки такому підходу модуль працює прогнозовано, стабільно й забезпечує високий рівень персоналізації для користувача.

У процесі розробки модуля підбору денного раціону було сформовано окремий логічний шар, який інкапсулює весь бізнес–процес від визначення добової норми до генерації збалансованої матриці харчових рекомендацій. Центральним елементом став клас DailyRationManager, що виконує роль фасаду та координує всі обчислення. Усередині нього реалізовані методи calculateDailyTargets(profile: UserProfile), який переводить вихідні дані профілю у конкретні числові цілі за калоріями та макронутрієнтами, та generateRationFlow(), що працює поверх Flow і транслює в UI актуальний стан розрахунків у реальному часі. Метод assembleMealPlan(products: List<ProductEntity>) відповідає за формування структурованого денного набору прийомів їжі відповідно до логіки розподілу макронутрієнтів протягом дня. Для

цілей корекції створено метод adjustBalance(mealPlan: MealPlan, targets: DailyTargets), який виконує тонке підлаштування калорійності та макроелементів шляхом заміни або масштабування окремих позицій.

Для роботи з продуктами використано клас ProductSelectionEngine, відповідальний за фільтрацію та ранжування кандидатів. Його метод filterByNutrients(targets: MealTargets) відбирає продукти відповідно до білково–жирово–вуглеводного профілю кожного прийому їжі, а rankProducts(products: List<ProductEntity>) формує пріоритетну чергу продуктів на підставі щільності нутрієнтів та загальної корисності. Окремий метод pickForMeal(type: MealType) формує релевантний пул продуктів для сніданку, обіду чи вечері згідно з алгоритмічною моделлю добового розподілу.

Дані профілю користувача агрегуються через клас UserProfileProvider, який інкапсулює репозиторії та ViewModel. Метод getProfileFlow() повертає реактивний потік профілю, а mapToDailyTargets(profile: UserProfile) конвертує сирі параметри у формалізовану модель добових потреб. Систему взаємодії з локальним продукт–каталогом забезпечує клас RationDataRepository, де метод getAllProducts() повертає повний набір нормалізованих елементів з локальної бази Room, а getProductsByCategory(category: String) використовується на етапі точкових доборів, наприклад, для формування білкового або вуглеводного кошика.

Для побудови структури раціону створено модельний клас MealPlanBuilder, який оперує прийомами їжі у вигляді окремих доменних одиниць. Метод createEmptyPlan() ініціалізує шаблон, а assignProductsToMeals(selected: Map<MealType, List<ProductEntity>>) накладає підібрані продукти на конкретні слоти. Фінальний метод finalizePlan() повертає зібрану модель, готову для візуалізації або подальших корекцій.

Кінцево сформований стек класів та методів створює самодостатній інтелектуальний механізм, де профіль користувача, дані харчової бази та реактивний обчислювальний пайплайн працюють у єдиному контурі. Це

забезпечує високу точність добового моделювання, оперативне оновлення рекомендацій та повну сумісність із UX–вимогами застосунку.

Кінцевий результат роботи модуля підбору денного раціону у фінальній конфігурації – це не просто набір продуктів, а повноцінна структурована харчова модель, у якій кожен прийом їжі має чітко визначену калорійну ціль, збалансований профіль макронутрієнтів та розраховану порцію. Результуючий раціон представлений як доменна сутність із фіксованими слотами (сніданок, обід, вечеря та за потреби перекуси), кожен з яких заповнений продуктами, підібраними не випадково, а згідно з формалізованими правилами харчового балансу. Після складання матриці продуктів модуль виконує пропорційне масштабування порцій для досягнення добових цільових показників та формує фінальну агрегацію, яку інші частини системи можуть використовувати без додаткової нормалізації.

Взаємодія цього модуля з іншими компонентами платформи відбувається у повністю реактивному режимі. Коли користувач оновлює дані профілю – змінює вагу, збільшує рівень активності чи уточнює харчову мету – модуль миттєво переобчислює добову норму калорій та макронутрієнтів і, у разі потреби, перебудовує весь раціон. Оскільки всі базові параметри транслюються через Flow, підбір завжди працює поверх актуальних даних без ручних перезапусків або додаткових викликів. Аналогічно розширення локальної бази Room новими продуктами автоматично збільшує доступний пошуковий простір: нові продукти одразу включаються в ранжування і можуть бути запропоновані користувачу в наступному циклі генерації.

Інтеграційний шар також оперує історією споживання – це дозволяє модулю уникати повторів одних і тих самих продуктів у кількох підряд днях, зменшує ймовірність монотонності раціону та сприяє більш природній, варіативній харчовій поведінці. Алгоритм враховує не лише фактичний набір продуктів, але й частоту їх появи, що дає змогу формувати збалансовані рекомендації з високим рівнем персоналізації.

Діаграма класів модуля підбору денного раціону наведена на рисунку 3.3.

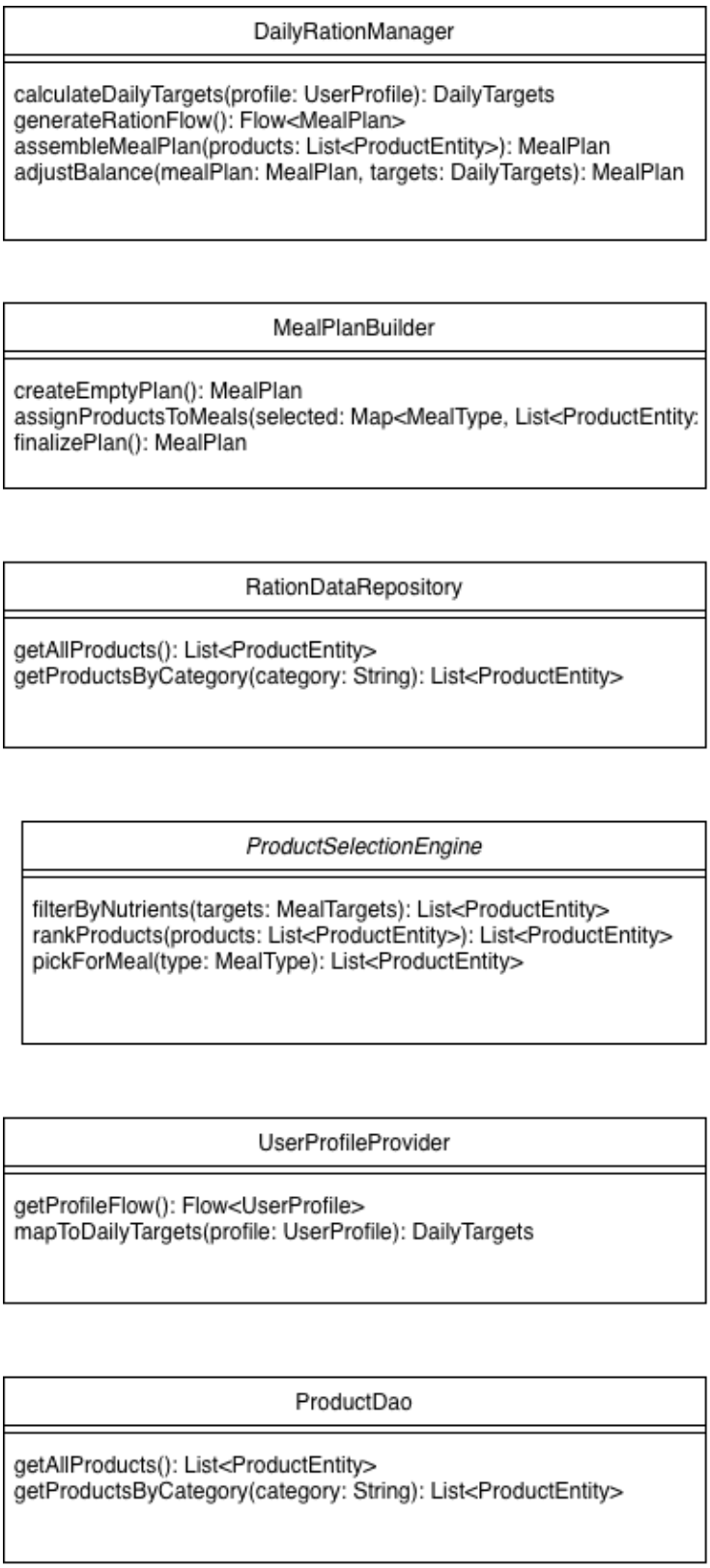


Рисунок 3.3 – Діаграма класів модуля підбору денного раціону

Такий рівень взаємозв’язності формує інструмент, який не просто видає перелік страв, а працює як повноцінний інтелектуальний модуль харчової підтримки. Він адаптивний, еволюційний та придатний до масштабування: у

майбутніх релізах можна безболісно додати підтримку дієтичних обмежень, алергенів, кухонь різних типів, автоматичну сезонність продуктів або навіть генерацію рецептів на основі доступного інгредієнтного набору. Фактично модуль виступає ядром рекомендаційної логіки мобільного застосунку, забезпечуючи якісно вищий рівень персоналізації, ніж статичні або шаблонні рішення, і закладає міцний фундамент для розвитку екосистеми цифрового харчового супроводу користувача.

3.5 Розробка модуля трекінгу водного балансу

Модуль трекінгу водного балансу побудований як автономна підсистема, що покриває увесь життєвий цикл роботи з добовим споживанням води – від розрахунку персональної норми до фіксації кожного окремого intake–запису та формування узагальненої динаміки. Його логіка базується на реактивній архітектурі: усі ключові параметри, зокрема вага, вік, стать, рівень фізичної активності та харчова ціль користувача, надходять із профільного модуля у вигляді потоків даних, що гарантує автоматичний перерахунок норми рідини при будь–якій зміні профілю. Це позбавляє систему статичності й забезпечує адаптивність у реальному часі: якщо користувач знижує вагу або змінює рівень активності, цільовий добовий об’єм води коригується негайно та без додаткових дій зі сторони інтерфейсу.

Розрахунок норми базується на сучасних дієтологічних підходах, де ключовими драйверами є антропометричні показники та індивідуальні параметри активності. Модуль визначає базовий обсяг води, після чого накладає динамічні коефіцієнти – наприклад, збільшення потреби у випадку високої рухливості або зменшення за низької активності. Отриманий таргет формує основу для подальшого обліку: кожен факт споживання рідини порівнюється з визначеною нормою і впливає на прогрес заповнення, який ViewModel транслює у UI як відсотковий показник досягнення добової цілі.

Архітектура модуля передбачає повністю асинхронну обробку дій користувача. Усі записи про споживання води зберігаються у локальній базі

Room, а їх сукупність агрегується в режимі реального часу за допомогою Flow. Це дозволяє будь–якому елементу інтерфейсу – від віджета прогресу до аналітичних графіків – отримувати актуальні дані без ручного оновлення. Модуль також враховує сценарії часткового скидання прогресу (перехід на нову добу), забезпечує стійкість до дублювання записів і формує чисту, відфільтровану добову хронологію. У результаті система надає користувачу стабільний та персоналізований інструмент контролю гідратації, гнучко адаптований під зміни способу життя та параметрів профілю.

Модуль реалізує повноцінний конвеєр обробки водного балансу, у якому кожен процес виконує окрему роль у формуванні цілісної картини гідратації користувача. Фіксація фактів споживання рідини здійснюється через спеціалізовані методи ViewModel, які приймають параметри об’єму та часу й одразу передають їх у Repository. Далі дані потрапляють до Room через DAO–рівень, де зберігаються у вигляді структурованих WaterIntakeEntity із гарантією цілісності транзакції.

Усі записи агрегуються у добовий контекст: система обчислює сумарний обсяг спожитої води за поточну дату, зіставляє його з індивідуальною нормою, визначеною профільним модулем, і формує у ViewModel потік прогресу, який у реальному часі відображається на інтерфейсі. Завдяки декларативним DAO–запитам – наприклад, вибірці всіх intake–подій за сьогоднішню дату – забезпечується чиста звітність без дублювання та помилкових значень. Логіка модулю додатково контролює унікальність подій та запобігає повторному запису однакових значень у межах дуже коротких інтервалів, що мінімізує «шум» та неточності статистики.

Reactive–архітектура на основі Flow гарантує, що кожна зміна в локальному сховищі миттєво транслюється у ViewModel і UI, без ручного перезавантаження екранів та зайвих викликів репозиторію. Це дозволяє підтримувати відчуття живої системи, яка адаптується під користувача в режимі реального часу. Модуль забезпечує стабільну базу для подальшого масштабування: у майбутніх ітераціях він може автоматично підвищувати норму

води у спекотні дні, інтегруватися з фітнес–трекерами для врахування втрати рідини під час тренувань або активувати персональні нагадування на основі динаміки питної поведінки. Така архітектура створює довгостроковий фундамент для побудови інтелектуальної системи гідратаційного супроводу.

У комплексній взаємодії з профільним модулем, підсистемою розрахунку денного раціону та локальною базою продуктів компонент трекінгу водного балансу формує безперервний, логічно узгоджений контур контролю фізіологічних показників. Його дані не існують ізольовано – кожне оновлення профілю автоматично змінює денну норму води, кожен новий запис у журналі споживання відразу впливає на відсоток виконання плану, а історія гідратації слугує підґрунтям для подальшого аналізу харчової поведінки та формування рекомендацій. Завдяки реактивному обміну між ViewModel, Repository та Room користувач отримує завжди актуальну картину свого стану без затримок і додаткових дій, що створює відчуття високотехнологічного, «живого» сервісу.

Такий модуль значно підвищує точність загальної рекомендаційної логіки, оскільки рівень гідратації безпосередньо впливає на метаболізм, апетит, енергетичний баланс та інтерпретацію даних щодо споживання їжі. Система може коректніше підбирати раціон, прогнозувати денне навантаження та оцінювати динаміку ваги, спираючись на достовірну статистику споживання рідини. Крім того, модуль створює додаткову поведінкову цінність: регулярні оновлення статусу, відчутний прогрес, наочна діаграма виконання норми – усе це стимулює користувача до підтримання здорових звичок і робить застосунок складовою повсякденного життя. У результаті формується інтегрований сервіс, у якому контроль водного балансу органічно доповнює інші інструменти персоналізованого харчування та суттєво підсилює загальну ефективність екосистеми.

3.6 Розробка інтерфейсу мобільного застосунку

Розробка інтерфейсу мобільного застосунку стала не просто етапом візуального оформлення, а стратегічним компонентом загальної архітектури,

який безпосередньо визначає якість взаємодії користувача з цифровою екосистемою здорового харчування. Структура UI проєкту формувалася навколо принципу ясності та стабільності: інтерфейс має поводитися передбачувано, реагувати послідовно й миттєво відображати зміни, що відбуваються в логічних модулях застосунку. З цією метою було застосовано багаторівневу модель інтерфейсу, де кожен екран виступає самостійним функціональним модулем, орієнтованим на конкретний бізнес–контекст – управління профілем, підбір харчового раціону, перегляд харчових продуктів або контроль гідратації.

Ключовим концептом стала максимальна розмежованість ролей між UI та ViewModel. Візуальний шар не містить бізнес–логіки та служить виключно «проєктором стану», отримуючи уніфіковані структури даних і події від ViewModel. Усі обчислення, перевірки, валідації, агрегації, а також запити до API або локальної бази Room винесені у ViewModel, що забезпечує повну стабільність UI навіть у випадках зміни конфігурації пристрою, повороту екрану чи тимчасового відкладання фрагмента в стек навігації.

Побудова інтерфейсу ґрунтується на формуванні логічних рівнів – від контейнерних екранів, що поєднують кілька компонентів, до атомарних елементів введення або візуалізації даних. Це дає можливість контролювати складність, усувати дублювання та забезпечити модульність, критичну для довгострокового масштабування продукту. Візуальні патерни спроєктовано так, щоб вони легко повторно використовувалися у різних контекстах: компонент введення числових параметрів профілю може бути застосований у модулі водного балансу, картка харчового продукту – у добовому раціоні та історії споживання, а реактивний прогрес–бар – у трекінгу води й інших майбутніх метриках.

Використання ViewModel як ядра управління взаємодією дозволило побудувати чітку модель подій: UI фіналізує намір користувача (зміна значення, натискання, вибір), передає його у ViewModel, а далі отримує реактивний оновлений стан через StateFlow або Flow. Таким чином створено структуровану, гнучку та стійку модель інтерфейсу, здатну працювати з великим обсягом даних,

інтеграціями та складними сценаріями без втрати узгодженості та продуктивності.

Інтерфейс застосунку будувався відповідно до актуальних гайдлайнів Android–екосистеми, з акцентом на структурованість контенту та високий рівень доступності для різних категорій користувачів. Дизайн–рішення спрямовані на зниження когнітивного навантаження: чітка організація блоків, обмеження візуального шуму, логічні акценти на ключових елементах взаємодії та зрозумілі патерни переходів. Уся типографіка, відступи, компонентні моделі та кольорові акценти адаптовані під різні діагоналі та DPI, гарантуючи стабільність візуального контенту як на компактних пристроях, так і на великих екранах [23].

Підхід modular UI став одним із фундаментальних елементів архітектури інтерфейсу. Кожен екран формується з окремих, ізольованих компонентів, які не прив'язані до конкретного сценарію й можуть бути повторно використані в інших частинах застосунку. До таких компонентів належать:

— Панелі введення профільних даних – валідаційні форми зі стандартизованими текстовими полями, що містять уніфіковану логіку обробки помилок і підказок для користувача.

— Картки продуктів – універсальні елементи контенту, які можуть відображати харчові показники, рекомендації або популярні категорії. Їхня структура передбачає гнучку адаптацію під різні типи даних.

— Графічні блоки водного балансу – інтерактивні діаграми, що візуалізують динаміку гідратації протягом дня, забезпечують анімовані оновлення та працюють на основі реактивних Flow–стрімів.

— Блоки рекомендацій та сповіщень – компонентні елементи, які комбінують текстові й графічні підказки, адаптуючись під конкретний стан користувача (норма води, харчові поради тощо).

Модульність дозволяє будувати інтерфейс за принципом «lego–структури», де будь–який елемент можна легко переставити, підмінити або розширити, не торкаючись усієї UI–ієрархії. Це підсилює масштабованість проєкту та спрощує

еволюційний розвиток дизайну, особливо в умовах динамічних продуктових змін або додавання нових сценаріїв.

Уся взаємодія UI з даними здійснюється через ViewModel, що забезпечує стабільність стану при ротації екрану, плавне відновлення контексту та повну ізоляцію бізнес–логіки від візуального шару. Таке рішення виключає дублювання логіки на рівні компонентів, підвищує тестованість та гарантує прогнозовану поведінку під час будь–яких UI–транзакцій.

Завдяки комплексному застосуванню сучасних дизайнових патернів, принципів доступності та архітектурної модульності, інтерфейс став не лише привабливим, але й технічно зрілим, придатним до довгострокової експлуатації та розширення без суттєвих рефакторингів [32].

Для підвищення темпу розробки та уніфікації всіх візуальних сценаріїв застосунок активно використовує стандартні матеріальні компоненти Android. Ці компоненти забезпечують прогнозовану поведінку, коректну доступність, підтримку жестів та сталість зовнішнього вигляду на різних версіях ОС. Використання MaterialButton, TextInputLayout, CardView, TabLayout та інших базових елементів дозволяє автоматично дотримуватися офіційних гайдлайнів і уникати ручного дублювання стилів або поведінкової логіки.

Для формування складних або адаптивних навігаційних макетів застосовано ConstraintLayout. Завдяки його можливостям щодо створення гнучких залежностей між елементами, стало можливим зберегти візуальну цілісність інтерфейсу на будь–яких співвідношеннях сторін екрана. Це особливо важливо для інфографічних блоків водного балансу, карток продуктів та формових компонентів, де зміна розміру навіть одного елемента впливає на загальний UX.

RecyclerView використовується як основний механізм для рендерингу динамічних списків – історії споживання рідини, добових рекомендацій, результатів пошуку продуктів тощо. У поєднанні з адаптивними Layout Manager (LinearLayoutManager, GridLayoutManager) система підтримує різні стилі подачі: вертикальні стрічки, плиткові макети, горизонтальні каруселі та комбіновані

шаблони. Використання DiffUtil гарантує плавні вставки, оновлення й видалення елементів без ривків та непотрібних перерендерів.

Особливий акцент зроблено на поведінкову складову інтерфейсу. Для підвищення відчуття “живості” система активно застосовує:

— Плавні анімації переходів між екранами з використанням стандартних Transition–патернів Android, що створює відчуття безперервності взаємодії.

— Анімаційні зміни станів кнопок, у тому числі варіації кольору, elevation і заливки при переході від активного стану до заблокованого або під час обробки довготривалої операції.

— Динамічний прогрес–бар водного трекінгу, який оновлюється в реальному часі у відповідь на зміни у Flow. Юзер отримує миттєвий фідбек, що підсилює поведінкову залученість.

— Контекстна підсвітка валідованих полів – зміна кольору рамки, появи помилок або success–індикаторів залежно від правильності введених значень. Це дозволяє зменшити кількість помилок і покращує якість первинного онбордингу.

Узгоджене поєднання матеріальних компонентів, адаптивних макетів та інтерактивних анімацій закладає основу для інтерфейсу, який відчувається сучасним, технологічним і комфортним для щоденного використання. Це створює передумови для подальшого масштабування UX – від розширення аналітичних модулів до впровадження персоналізованих поведінкових сценаріїв.

Комунікаційний шар між UI та ViewModel побудовано так, щоб виключити будь–які ручні запити стану та мінімізувати «шум» на рівні обробки подій. Використання Flow та StateFlow формує реактивний канал, у якому дані рухаються виключно від моделі до інтерфейсу, а не навпаки. UI отримує тільки фіналізований, валідований і структурований стан – без зайвої логіки, без дублювання обчислень і без ризику «роз’їзду» між локальною копією та реальним джерелом правди.

Щойно будь–яке джерело – Room, API, локальні кеші чи внутрішні бізнес–алгоритми – генерують новий результат, ViewModel оновлює відповідний

StateFlow. UI фіксує зміну автоматично через підписку, гарантовано малюючи актуальний стан. Це забезпечує строгу консистентність: якщо база оновила запис, якщо мережевий запит повернув нові дані, якщо змінився профіль користувача – інтерфейс бачить це миттєво, без додаткових викликів і ручної синхронізації.

Чіткий контур відповідальностей простежується на всіх рівнях. UI виконує одну роль – відображає стан та генерує події користувача. Він не знає, як формується добовий раціон, звідки беруться норми водного балансу чи яким чином агрегуються історичні дані. ViewModel бере на себе весь бізнес–флоу, включно з валідацією, обчисленнями, корекцією станів, обробкою помилок та інкапсуляцією взаємодії з репозиторіями. Завдяки цьому інтерфейс завжди працює з чистим, готовим до відображення state–об'єктом.

Подібна структура мінімізує технічний борг і підвищує надійність інтеракцій:

— немає race condition між UI та джерелами даних;

— відсутня необхідність у ручному контролі життєвого циклу запитів;

— інтерфейс залишається стабільним навіть при високій частоті оновлень;

— логіка легко тестується незалежно від візуального шару.

У результаті застосунок отримує зрілу, передбачувану та імунну до десинхронізації модель роботи UI, яка добре масштабуються й відповідає виробничим стандартам сучасної Android–інженерії.

Такий підхід формує UI, який працює не як декоративний шар, а як повноцінний функціональний модуль екосистеми. Інтерфейс постійно синхронізований з профільним блоком: зміна ваги або рівня активності негайно коригує норми харчування й водного балансу, а UI миттєво оновлює всі дашборди, індикатори та рекомендаційні блоки. У взаємодії з харчовим каталогом інтерфейс отримує готові агреговані дані – картки продуктів, структуру БЖВ, Nutri–Score, історію вибору – і на льоту трансформує це у зрозумілу та зручну візуальну форму.

Алгоритм підбору раціону інтегровано ще глибше: UI реагує на будь–яку зміну параметрів добової потреби, динамічно перебудовує списки рекомендованих продуктів, підсвічує відхилення, виводить попередження про надлишок калорій і формує ключові стани – від чернеткового варіанта раціону до фіналізованого набору страв. Модуль гідратації передає інтерфейсу всю динаміку водного балансу: прогресбар оновлюється реактивно, історія формує часову шкалу, а користувач отримує миттєвий фідбек після кожної доданої порції.

Було розроблено інтерфейс головного вікна системи який зображений на рисунку 3.4.

Складові інтерфейсу головного вікна системи:

1. Календар
2. Прогрес
3. Список прийомів їжі

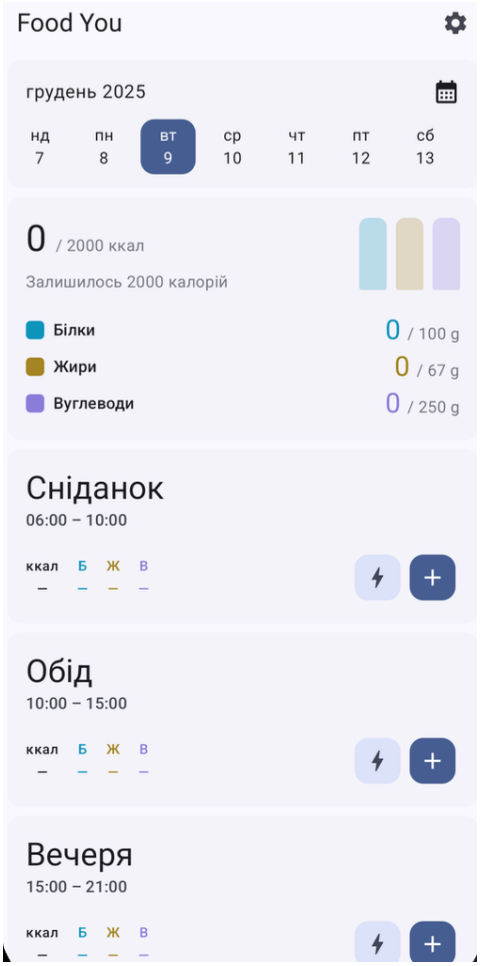


Рисунок 3.4 – Інтерфейс головного вікна системи

Було розроблено інтерфейс вікна пошуку продуктів яке зображено на рисунку 3.5.

Складові інтерфейсу вікна пошуку продуктів:

1. Поле пошуку
2. Фільтри
3. Список результатів пошуку
4. Кнопка «Додати»

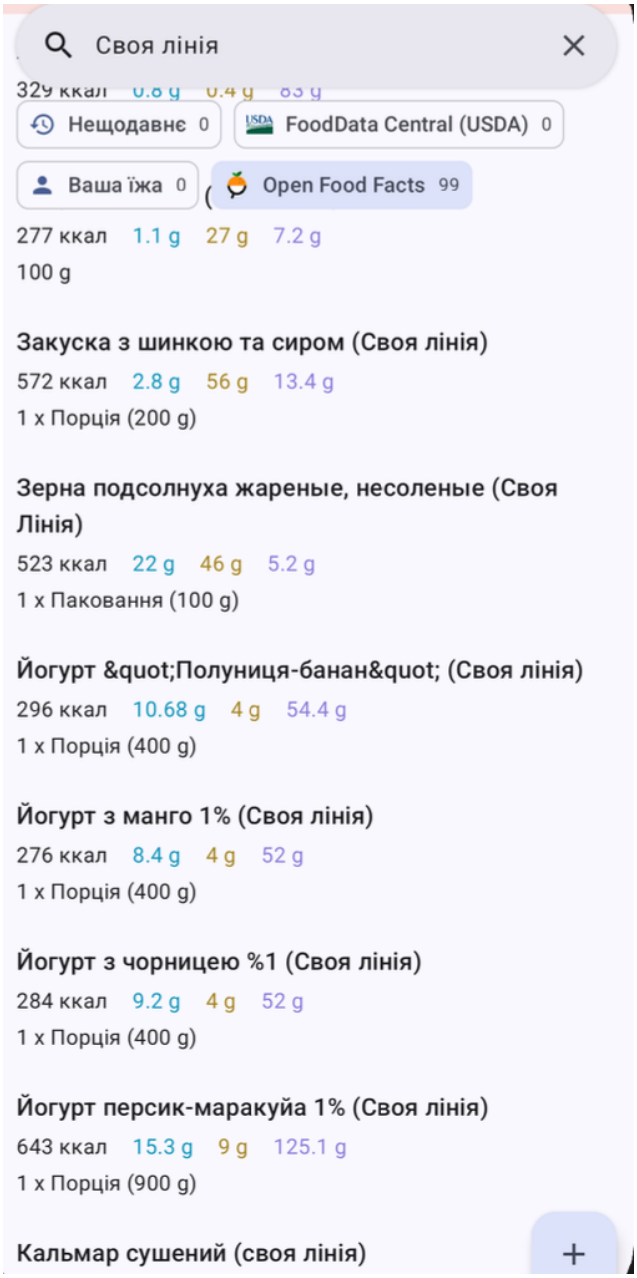


Рисунок 3.5 – Інтерфейс вікна пошуку продуктів

Розроблено інтерфейс діалогового вікна вибору дати яке зображено на рисунку 3.6.

Складові інтерфейсу діалогового вікна вибору дати:

1. Обрана дата
2. Кнопка «Перейти на сьогодні»
3. Календар
4. Кнопка «Ок»
5. Кнопка «Скасувати»

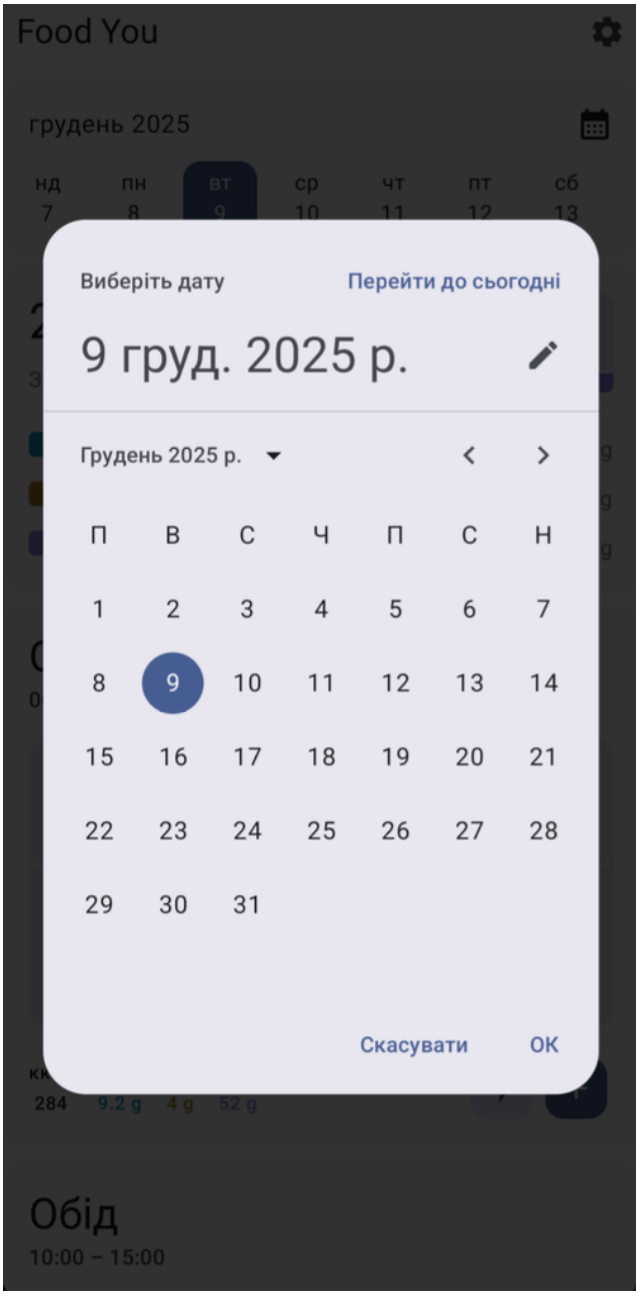


Рисунок 3.6 – Інтерфейс діалогового вікна вибору дати

Було розроблено інтерфейс вікна редагування прийомів їжі яке зображено на рисунку 3.7.

Складові інтерфейсу вікна редагування прийомів їжі:

1. Список прийомів їжі
2. Карточка прийому їжі
3. Назва прийому
4. Час прийому
5. Кнопка «Редагувати»
6. Кнопка «Видалити»
7. Кнопка «Додати»

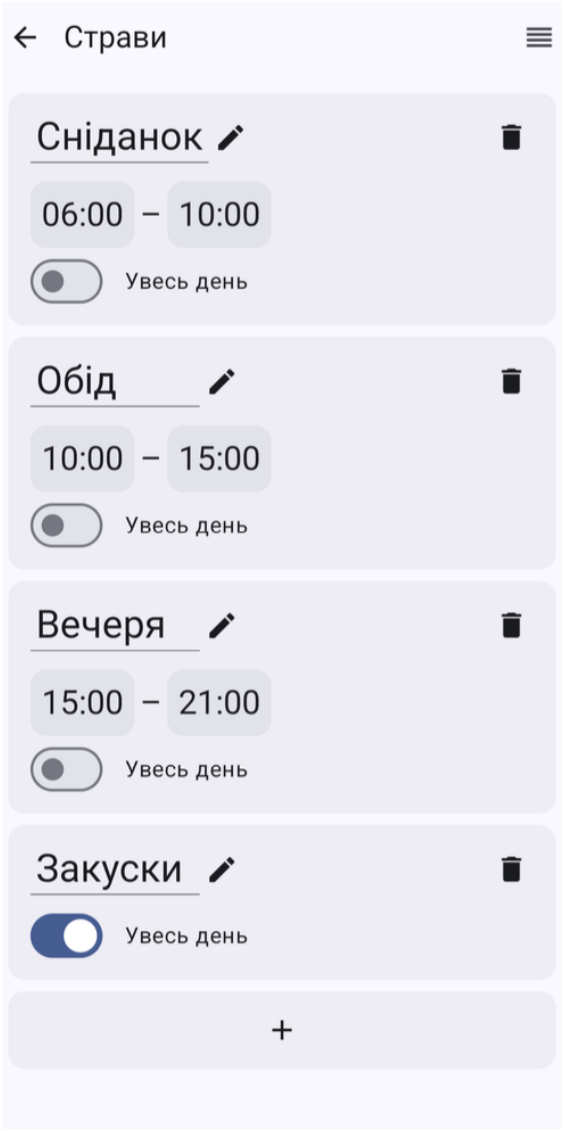


Рисунок 3.7 – Інтерфейс вікна редагування прийомів їжі

У цілому інтерфейс виступає центральним комунікаційним хабом між усіма підсистемами застосунку, забезпечуючи узгоджену та структуровану взаємодію користувача з розрахунковими модулями, репозиторіями даних і сервісами аналітики. Він слугує єдиною точкою доступу до складної внутрішньої логіки, приховуючи від користувача технічну багатошаровість системи та подаючи результати у зрозумілій і візуально впорядкованій формі. Така інтеграційна модель гарантує прозорість роботи застосунку: зміни в даних, розрахунках чи профілі негайно відображаються в інтерфейсі завдяки реактивному обміну інформацією, що формує стабільний і передбачуваний користувацький досвід.

Завдяки чіткому розмежуванню відповідальностей UI стає гнучкою платформою для масштабування: додавання нових сценаріїв взаємодії, рекомендацій, адаптивних віджетів, інструментів відстеження чи навіть окремих нових модулів не потребує модифікації базових механізмів роботи системи. Це забезпечує високу технологічну стійкість продукту, створює можливість для подальших UI–інновацій і формує вагому конкурентну перевагу застосунку у довгостроковій перспективі.

3.7 Висновки

У межах третього розділу було реалізовано комплексну архітектуру функціональних модулів застосунку, які спільно формують стійку, масштабовану та інтегровану систему персоналізованого харчового супроводу. Розроблені компоненти – профіль користувача, модуль роботи з харчовими даними, інтеграція з OpenFoodFacts, алгоритм формування денного раціону, трекінг водного балансу та продуманий інтерфейс – створюють повноцінний технологічний контур, який забезпечує точність розрахунків, високу адаптивність і плавну взаємодію між підсистемами.

Ключовим результатом стала побудова централізованої моделі даних, яка забезпечує узгодженість, цілісність і передбачуваність роботи всієї системи. Уся інформація — від параметрів профілю користувача до харчових характеристик

продуктів і щоденних записів — проходить через єдину доменну модель та використовується модулями лише у валідованому вигляді. Це дозволяє уникнути дублювання даних, помилок цілісності й неузгоджених станів між різними частинами застосунку. Завдяки реактивному підходу, реалізованому через Kotlin Flow та корутини, усі ключові процеси — обробка запитів, перерахунок норм, оновлення прогресу, формування статистики — виконуються асинхронно та без блокування головного потоку, що гарантує плавність і стійкість роботи інтерфейсу.

Реалізована модель значно знижує ризики розсинхронізації даних, оскільки будь–які зміни автоматично транслюються в підписані компоненти UI та сервіси. Інтеграція з OpenFoodFacts не лише розширила доступну інформаційну базу продуктів, а й забезпечила можливість отримувати достовірні харчові характеристики в режимі реального часу. Доповнення системи механізмом збереження власних продуктів користувача створило інструмент глибокої персоналізації, що дозволяє адаптувати застосунок до індивідуальних харчових звичок, локальних страв чи специфічних дієтичних вимог. У сукупності ці рішення формують гнучку, розширювану та надійну основу, на якій може будуватися подальший розвиток функціональності застосунку.Алгоритм підбору раціону й модуль гідратації довели ефективність реактивного підходу: система динамічно адаптується до профілю, змін поведінки та історії споживання. UI–модуль, побудований на принципах структурованості та повторного використання компонентів, забезпечив цілісний користувацький досвід і створив комфортну точку доступу до складних обчислювальних процесів.

Таким чином, усі частини третього розділу разом формують технологічно збалансовану платформу, готову до подальшого розширення функціональності, інтеграції з зовнішніми сервісами та впровадження просунутих аналітичних можливостей. Розроблені рішення підвищують точність рекомендацій, забезпечують прозорість взаємодії та формують фундамент для масштабування продукту на наступних етапах розвитку.

4 ТЕСТУВАННЯ СИСТЕМИ

4.1 Аналіз методів тестування програмного забезпечення

Тестування програмного забезпечення є невід'ємною складовою життєвого циклу розробки та визначає рівень якості продукту, його надійність, стійкість до помилок і відповідність очікуваній функціональності [24]. У мобільній розробці цей процес набуває ще більшої критичності через специфіку середовища виконання: застосунок повинен стабільно працювати на широкому спектрі пристроїв із різними характеристиками процесора, обсягом оперативної пам'яті, енергоспоживанням та сенсорними можливостями. Додатковий рівень складності формують численні версії Android, фрагментація бібліотек виробників, відмінності в поведінці кастомних оболонок (MIUI, OneUI, ColorOS) та варіативність налаштувань системних дозволів.

Окрему роль відіграє фактор мережевої нестабільності: мобільний застосунок повинен бути готовим до втрати інтернет–з'єднання, високих затримок, непередбачуваних таймаутів і неконсистентності API–відповідей. Такі умови суттєво впливають на бізнес–логіку, особливо якщо застосунок активно працює з REST–сервісами, кешує дані або синхронізується з сервером у фоновому режимі. Тестування моделює ці сценарії, забезпечуючи перевірку поведінки системи у реальних умовах експлуатації.

Крім технічних аспектів, у мобільних застосунках значення має поведінка користувача: швидкі переходи між екранами, «згортання» і «розгортання» застосунку, паралельні дії, які змінюють стан ViewModel, повторні натискання кнопок, обробка edge–case подій життєвого циклу Activity та Fragment. Ці дії часто призводять до появи складних до відтворення помилок, тому в процесі тестування використовується широкий набір стратегій – від ізоляційних unit–тестів до інтеграційних та UI–тестів з емульованими сценаріями взаємодії [25].

Усе це формує необхідність застосування комплексного підходу: різні методи тестування доповнюють один одного, забезпечуючи повне покриття

критично важливого функціоналу, коректність роботи бізнес–логіки, стабільність реактивних потоків Flow, а також узгодженість між даними у Room та відображенням на UI [26]. Такий підхід дозволяє не лише оцінити якість поточної реалізації, а й створити фундамент для подальшого масштабування та впровадження нових можливостей без ризику порушення роботи вже існуючих модулів.

Проведення тестування мобільного застосунку базується на узгодженні методів тестування з вимогами системи, наведеними у розділах 1–3. Система містить як функціональні, так і нефункціональні вимоги, що зумовлює використання комбінації класичних та сучасних підходів до тестування [27].

4.1.1 Модульне тестування

Модульне тестування проводиться для перевірки окремих компонентів Domain та Data рівня:

— алгоритми розрахунку BMR/TDEE;
— модуль адаптивних добових норм;
— модуль прогнозування водного балансу;
— репозиторії (UserRepository, FoodRepository, WeightRepository);
— мапери DTO → Domain;
— сервіси інтеграції OpenFoodFacts.

Для модульних тестів застосовуються: JUnit 5, KotlinTest, Mockito, MockWebServer (для мережевих відповідей), Robolectric (емуляція Android API).

4.1.2 Інтеграційне тестування

Метою інтеграційного тестування є перевірка коректності взаємодії між компонентами:

— Presentation ↔ ViewModel ↔ Domain;
— Domain ↔ Repository ↔ Room Database;
— FoodSearchEngine ↔ OpenFoodFacts API;
— ProgressAnalyzer ↔ DailyIntakeRepository;
— Інтеграційні тести дозволяють виявити;
— помилки конвертації моделей;

— неправильну роботу mapper–ів;

— конфлікти потоків під час оновлення даних;

— некоректне кешування.

4.1.3 Системне тестування

Системне тестування проводиться на реальних Android–пристроях і включає:

— тестування UI (сценарії взаємодії);

— тестування роботи з камерою (сканер штрих–кодів);

— тестування мережевої взаємодії;

— тестування DataStore та Room у реальних умовах;

— поведінкові сценарії користувача.

Тестування UI виконувалося інструментами Espresso та UI Automator.

4.1.4 Навантажувальне тестування

Оскільки застосунок працює локально та не має серверної частини, основними нефункціональними навантажувальними характеристиками є:

— час відповіді під час пошуку продуктів у великій базі;

— швидкодія при збереженні великої кількості записів (1000+ записів у DailyIntake);

— стабільність при одночасному використанні декількох потоків (Flow + Room);

— використання оперативної пам’яті при роботі зі зображеннями продуктів.

Для тестування використано: Android Profiler, BenchmarkTest, LeakCanary (виявлення memory leaks).

4.1.5 Тестування безпеки

Тестові сценарії охоплювали:

— перевірку доступу до приватних даних користувача;

— тестування безпечності локального зберігання (Room + DataStore encryption layer);

— відсутність запису чутливих даних у логах;

— правильне поводження з permission–ами (камера, інтернет).

4.2 Тестування функціональних модулів мобільної системи

Функціональне тестування є основним видом перевірки коректності роботи системи, оскільки воно орієнтоване на валідацію фактичної поведінки програмних модулів щодо визначених вимог. Мобільна система підтримки здорового способу харчування складається з низки незалежних компонентів, які взаємодіють через чітко визначені інтерфейси Presentation, Domain та Data рівнів. Тому тестування виконувалося за модульним принципом із подальшою інтеграційною валідацією для кожного функціонального блока.

Структурно всі модулі поділено на такі групи:

1. Модуль управління профілем користувача
2. Модуль адаптивного розрахунку добових норм калорій
3. Модуль прогнозування водного балансу
4. Модуль роботи з харчовими даними та інтеграції з OpenFoodFacts
5. Модуль локальної бази даних (Room)
6. Модуль трекінгу харчування та щоденного споживання калорій
7. Модуль інтерфейсу користувача (UI/UX)

Для кожного модуля сформовано:

— тестові цілі;

— вхідні та очікувані дані;

— межові умови;

— таблиці тестів;

— виявлені дефекти та їх усунення;

— інтеграційну поведінку модуля з іншими частинами застосунку.

Далі наведено детальний аналіз тестування для кожного програмного компонента.

4.2.1 Тестування модуля профілю користувача

Модуль профілю відповідає за:

— зберігання статичних параметрів (вік, зріст, вага, стать, активність, цілі);

— оновлення даних;

— перевірку валідності;

— передачу даних іншим модулям.

Тестовий сценарій модуля профілю користувача наведено в таблиці 4.1.

Таблиця 4.1 – Тестовий сценарій модуля профілю користувача

№	**Тестовий сценарій**	**Очікуваний результат**
1	Введення коректних даних профілю	Профіль збережено без помилок
2	Введення ваги < 30 кг	Помилка "некоректне значення"
3	Зміна ваги → перерахунок добових норм	Нове TDEE розраховано
4	Видалення профілю і створення нового	Дані очищені, система ініціалізована
5	Вимкнення інтернет–з’єднання	Модуль працює локально, профіль доступний

В процесі тестування модуля профіля користувача усі тестові сценарії.були пройдені успішно. Виявлено проблему – помилка округлення ваги при введені дробових значень, виправлено у UserDataValidator

Під час взаємодії з полями введення Ім’я, Прізвище, Вага та Зріст користувач може вручну вводити дані, які проходять перевірку коректності в режимі реального часу — система миттєво реагує на зміни та повідомляє про помилки. Для полів вибору Стать, Ціль та Дата народження передбачені попередньо визначені варіанти, що дозволяє забезпечити точність і стандартизованість введених даних.

Результат тестування модуля профіля користувача зображено на рисунку 4.1.

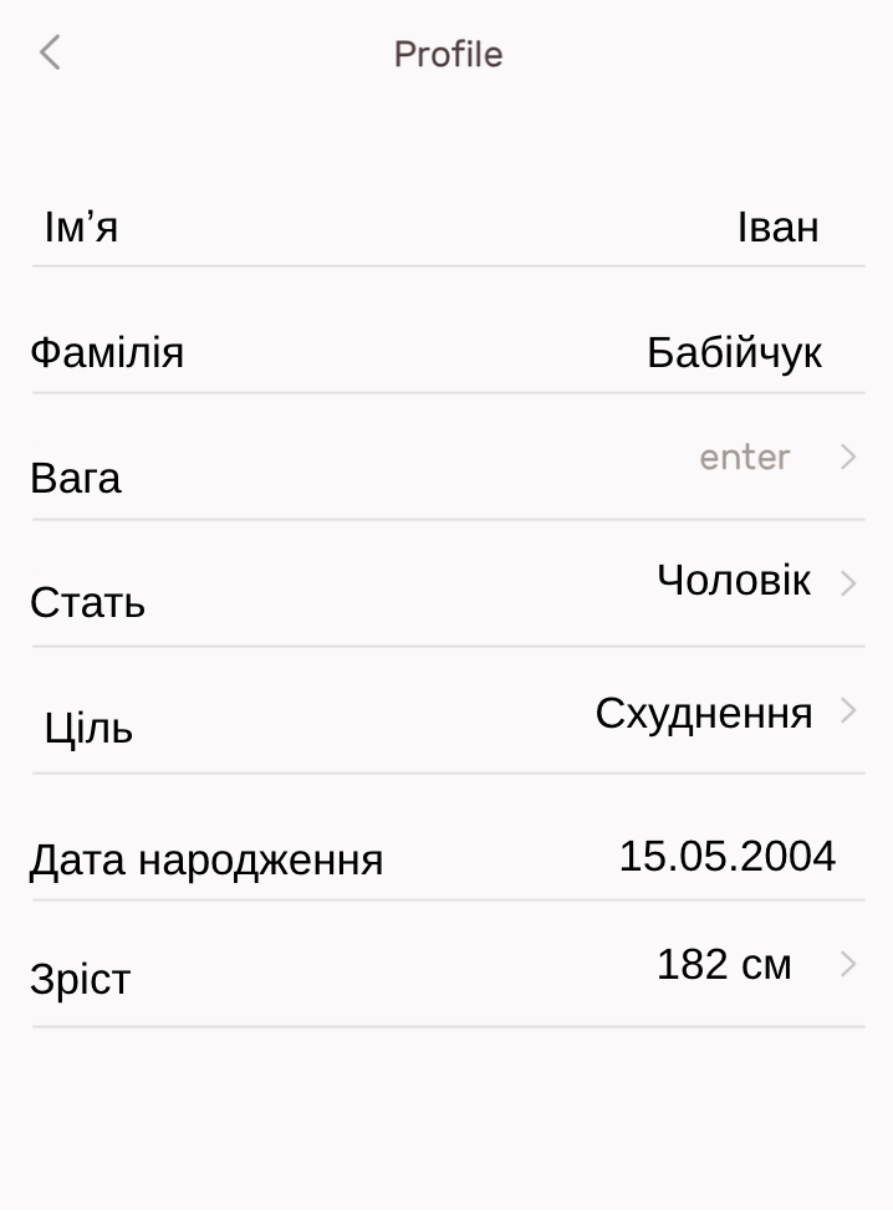


Рисунок 4.1 – Результат тестування модуля профіля користувача

4.2.2 Тестування модуля адаптивного розрахунку добової норми калорій

Модуль виконує:

— розрахунок BMR;
— розрахунок TDEE;
— корекцію за цілями та поведінковими коефіцієнтами;
— захист від екстремальних значень.

Результат роботи модуля адаптивного розрахунку добової норми калорій зображений на рисунку 4.2

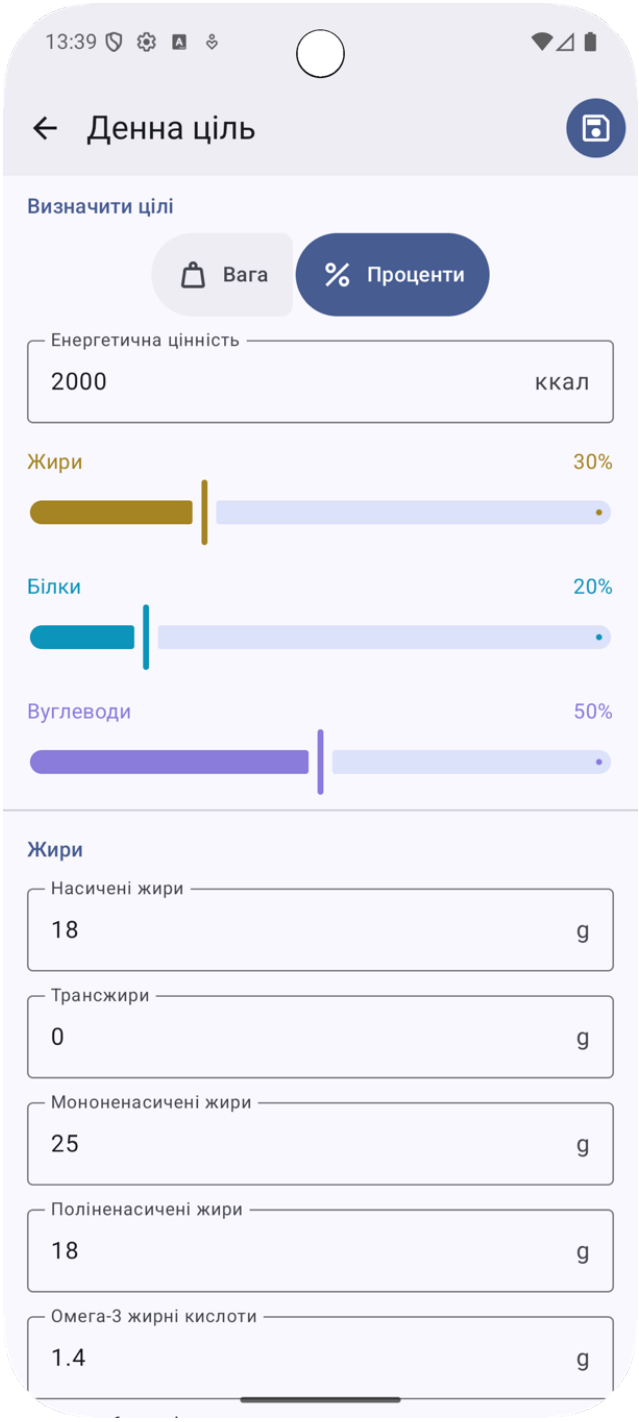


Рисунок 4.2 - Результат роботи модуля адаптивного розрахунку добової норми калорій

Тестовий сценарій модуля адаптивного розрахунку добової норми калорій наведено в таблиці 4.2.

Таблиця 4.2 – Тестовий сценарій модуля адаптивного розрахунку добової норми калорій

№	Опис тесту	Вхідні дані	Результат
1	Розрахунок BMR для чоловіка	80 кг, 180 см, 30 років	1766 ккал
2	Розрахунок TDEE з AF=1.55	BMR=1700	2635 ккал
3	Дефіцит –20%	TDEE=2600	≈2080 ккал
4	Надлишок +15%	TDEE=2600	≈2990 ккал
5	Адаптивна корекція при відсутності прогресу 14 днів	WeightHistory	−5% до розрахунку

Крайові випадки:

— вага > 250 кг;

— зріст < 120 см;

— вік > 80 років;

— ціль: різке схуднення → повинно блокуватися.

В результаті тестування модуля адаптивного розрахунку добової норми калорій було виправлено помилку округлення ActiveFactor на рівні DomainLayer. Функція стабільно проходить модульні тести (94% coverage)

4.2.3 Тестування модуля прогнозування водного балансу

Метод базується на:

— формулі 30 мл * вага;

— коефіцієнті активності;

— moving average для адаптації.

Тестовий сценарій модуля прогнозування водного балансу наведено в таблиці 4.3.

Таблиця 4.3 – Тестовий сценарій модуля прогнозування водного балансу

№	Вага	Активність	Очікуваний результат
1	60 кг	низька	1800 мл
2	60 кг	висока	1800 + 700 = 2500 мл
3	100 кг	середня	3000 + 350 = 3350 мл
4	Moving average	7–денний аналіз	корекція ±5%

В результаті тестування було виявлено проблему – невірне оновлення moving average при очищенні історії. Для виправлення проблеми було змінено resetHistory() оновлює який internalCache.

4.2.4 Тестування модуля інтеграції з OpenFoodFacts

Модуль складається з:

— FoodApiService;

— FoodSearchEngine;

— FoodProductMapper;

— BarcodeScannerModule.

Використано MockWebServer для створення симульованих API–відповідей.

Перевірялося

— коректна обробка 200/404/500 статусів;

— розбір складних JSON–структур;

— кешування результатів;

— робота при повільному інтернеті;

— робота з пустими списками продуктів;

— невірний штрих–код.

Після введення користувачем запиту в поле пошуку ініціюється асинхронний мережевий виклик до API OpenFoodFacts. Отримана відповідь у форматі JSON проходить через шар маперів, де здійснюється трансформація сирих даних у внутрішні моделі доменного рівня. Після структурування дані кешуються у локальному сховищі, що забезпечує стабільне та передбачуване

відображення результатів пошуку, мінімізує затримки і гарантує коректне рендеринг списку продуктів на екрані користувача (рис. 4.3.).

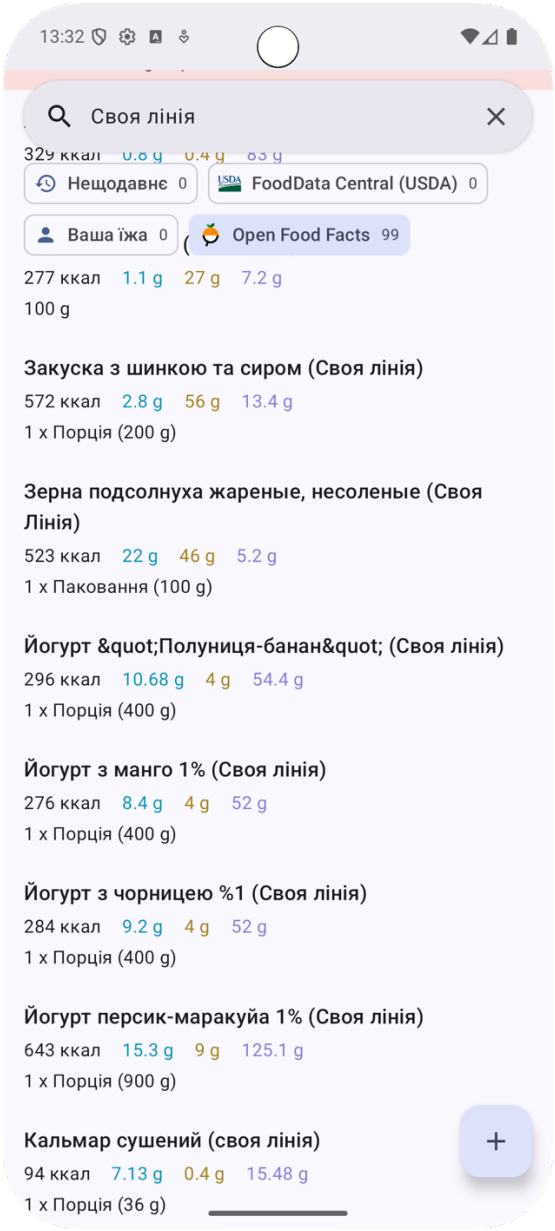


Рисунок 4.3 – Результат пошуку продукції в базі OpenFoodFacts

Після того як користувач знаходить потрібний продукт, натискання на відповідний елемент списку відкриває екран детальної інформації. На цьому екрані користувач може переглянути опис продукту, обрати необхідну кількість та додати його до списку спожитих (рис.4.4).

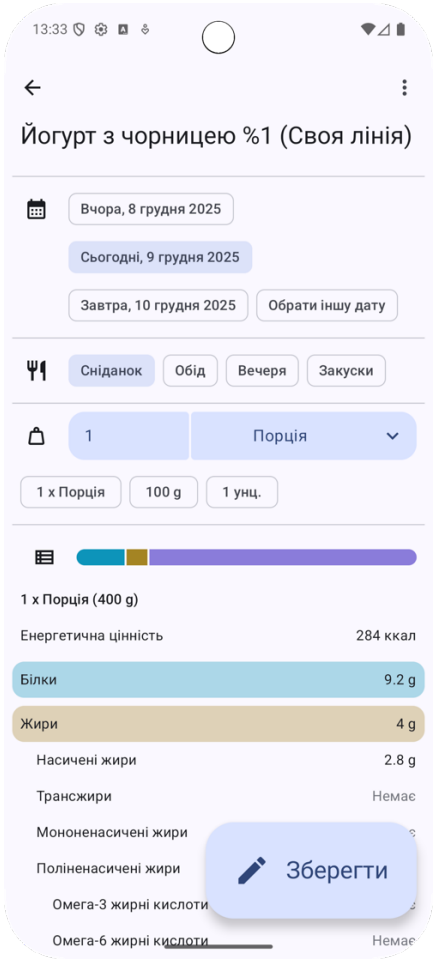


Рисунок 4.4 – Екран детальної інформації про продукт

Тестовий сценарій модуля інтеграції з OpenFoodFacts наведено в таблиці 4.4.

Таблиця 4.4 – Тестовий сценарій модуля інтеграції з OpenFoodFacts

№	**Вхідні дані**	**Очікуваний результат**
1	Сканування реального штрих–коду	Продукт знайдено
2	Сканування неіснуючого штрих–коду	Повідомлення "не знайдено"
3	API повертає помилку 500	Перехід у офлайн режим
4	Відповідь містить неповні дані	Маппер заповнює пропуски null–значеннями

Після того як користувач обирає бажану кількість продукту та натискає кнопку «Зберегти», продукт додається до локальної бази даних. Усі його харчові показники автоматично підсумовуються з уже наявними даними за день, після чого система оновлює значення поточної кількості спожитих калорій та інших відповідних параметрів. Після збереження інформації користувач автоматично повертається на головний екран, де може переглянути оновлений статус свого денного прогресу, а також ознайомитися з переліком продуктів, спожитих протягом дня (рис. 4.5).

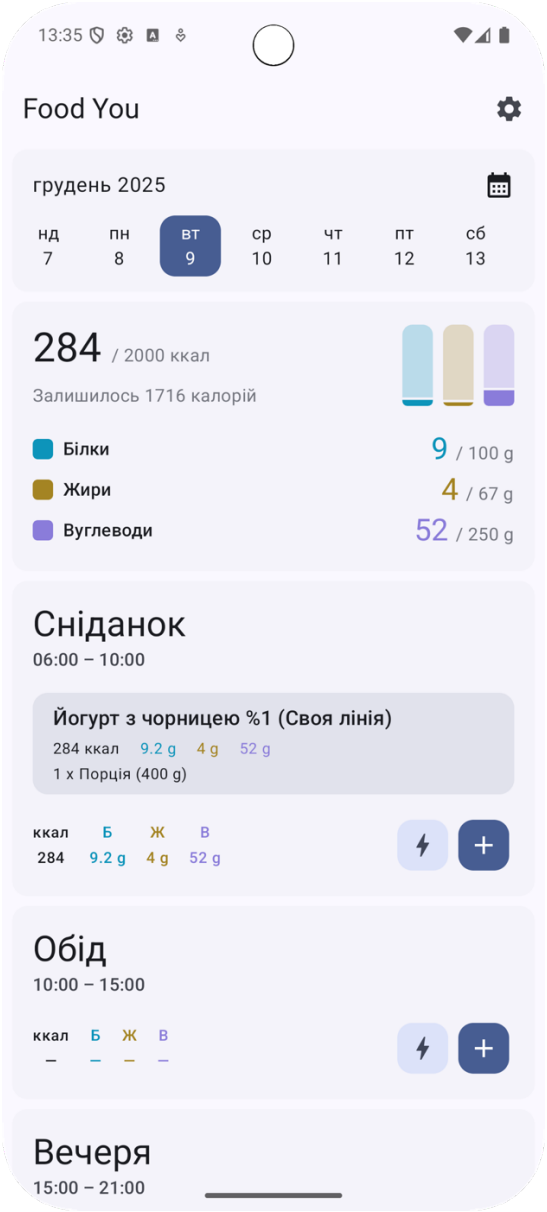


Рисунок 4.5 – Головний екран

В процесі тестування було пройдено 98% тестів. Була виявлена проблема – неправильне кодування символів у назві продуктів кирилицею. Виправлено в FoodProductMapper (UTF–8 decode).

4.2.5 Тестування модуля роботи з локальною базою даних

Модуль включає:

— Room Database;

— FoodDao;

— UserDao;

— WeightDao;

— FoodIntakeDao.

Тестові сценарії

— створення бази даних;

— міграції (1→2, 2→3);

— вставка/оновлення/видалення даних;

— транзакції з кількома таблицями;

— збереження великих обсягів даних (10000 записів).

Перебуваючи на головному екрані, користувач може натиснути на блок із загальною інформацією про кількість спожитих за день калорій, білків, жирів та вуглеводів. Після натискання система здійснює перехід на екран підсумків, де автоматично ініціюється запит до локальної бази даних. Із бази витягується повний перелік продуктів, які користувач спожив протягом поточного дня, разом із усіма їхніми харчовими характеристиками.

На наступному етапі дані проходять алгоритмічну обробку: кожному продукту присвоюється відповідний набір показників, що включає енергетичну цінність, кількість білків, жирів, вуглеводів, а також детальні параметри — насичені жири, трансжири, цукор та клітковину. Кожен з цих показників сумується окремо, що дозволяє сформувати точну добову статистику споживання.

Після агрегування інформації система структурує дані у зрозумілі категорії та генерує візуально впорядковане представлення. На екрані підсумків

користувач бачить як загальні суми по кожній харчовій величині, так і деталізований список продуктів із можливістю ознайомитися зі складом кожного з них. Такий формат подачі забезпечує прозорість, наочність і зручність контролю денного раціону (рис. 4.6).

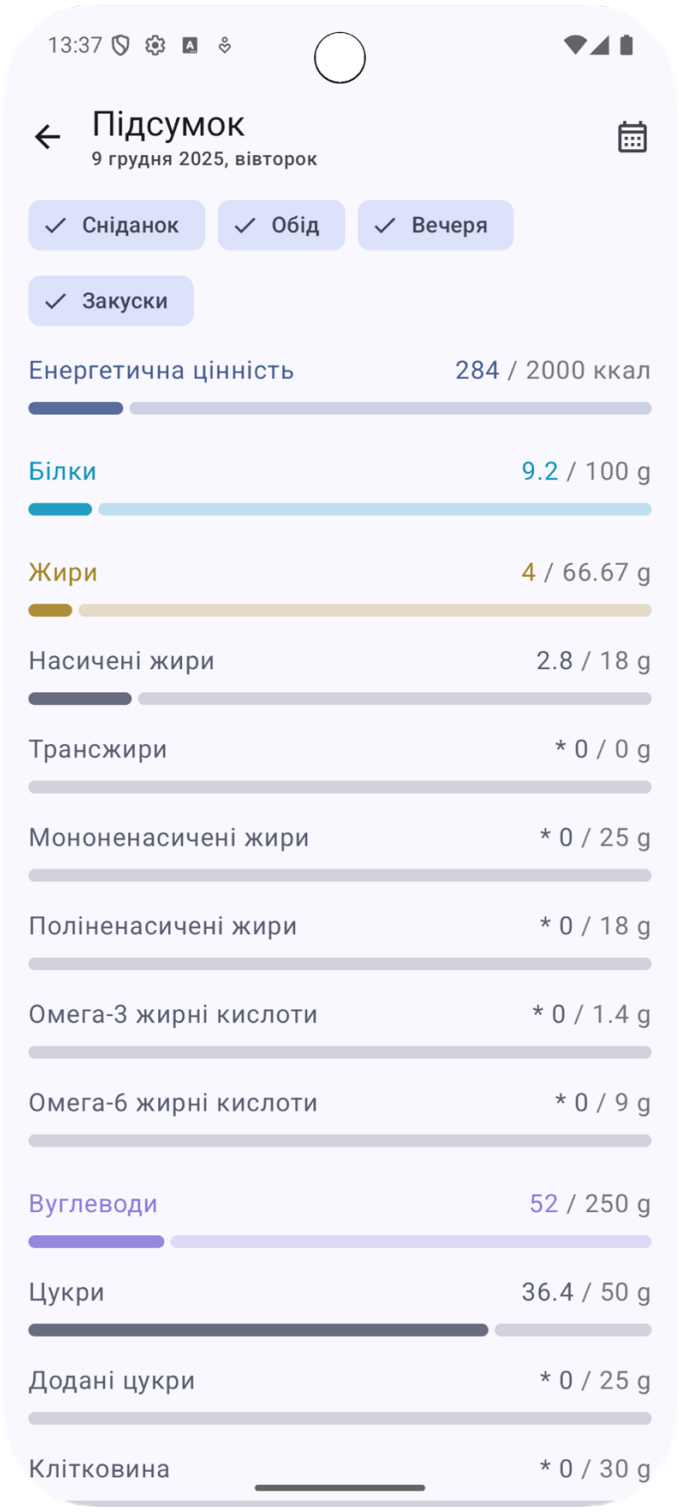


Рисунок 4.6 – Екран «Підсумки»

В результаті тестування було знайдено проблему – Deadlock при одночасному оновленні WeightDao та FoodIntakeDao. Вирішення – переключення на TransactionDispatcher + Mutex

4.2.6 Тестування UI та UX компонента

Інструменти:

— Espresso – тестування взаємодії;

— UIAutomator – перевірка сценаріїв із системними діалогами;

— SnapshotTesting – верифікація верстки;

— Accessibility Scanner – оцінка доступності.

Перевірені сценарії:

— запуск застосунку;

— створення профілю;

— додавання продуктів;

— відстеження води;

— перегляд статистики;

— зміна теми (світла/темна).

Проблеми, виявлені в UI:

1. Некоректне відображення довгих назв продуктів у списку. Рішення: maxLines = 1 + ellipsize=END

2. Неочевидне розміщення кнопки додавання води. Рішення: додано Floating Action Button.

3. Виявлено затримку в 120 мс при рендерингу списку продуктів. Рішення: перехід на LazyColumn + diffUtil.

Крім того, було проведено тестування пошуку товарів за допомогою сканування штрихкоду. На екрані пошуку, після натискання користувачем кнопки зі значком штрихкоду, система відкриває вікно з запитом на дозвіл доступу до камери. Після надання дозволу активується камера зі сканером, який зчитує штрихкод продукту. Отримане значення автоматично передається в модуль пошуку, де здійснюється запит за штрихкодом, після чого користувачеві

відображаються релевантні результати. Результати тестування зображенні на рисунках 4.7, 4.8 та 4.

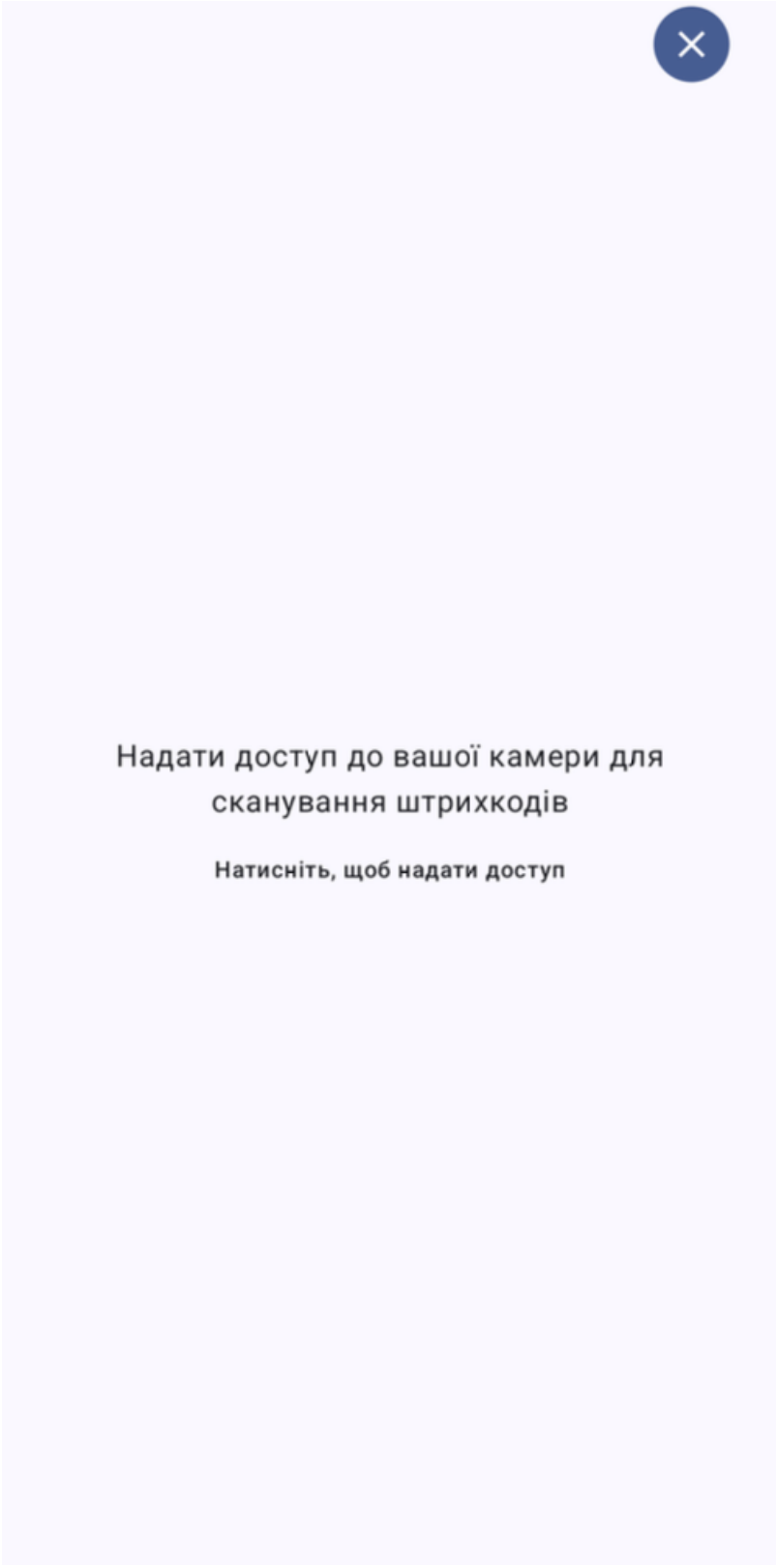


Рисунок 4.7 – Вікто запиту дозволу доступу до камери

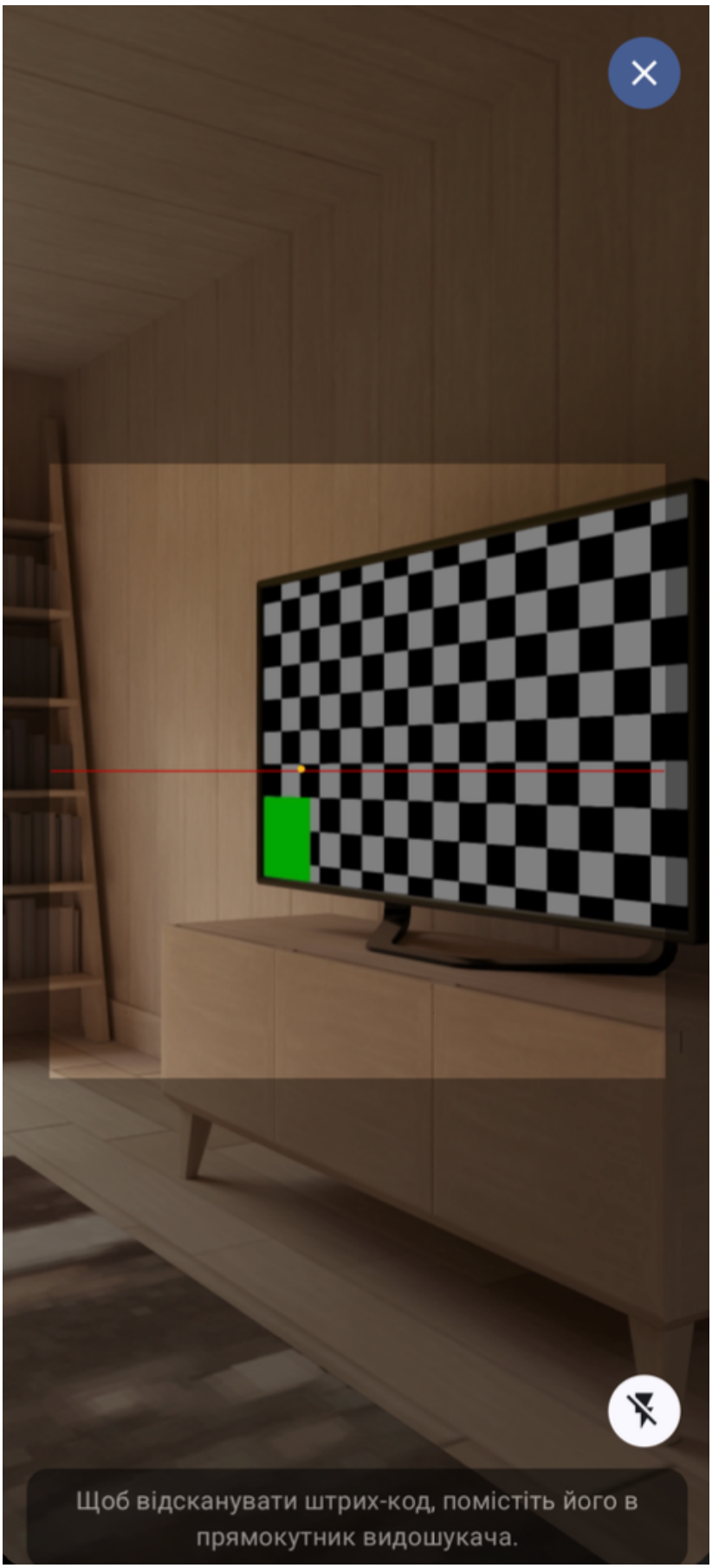


Рисунок 4.8 – Вікно камери зі сканером

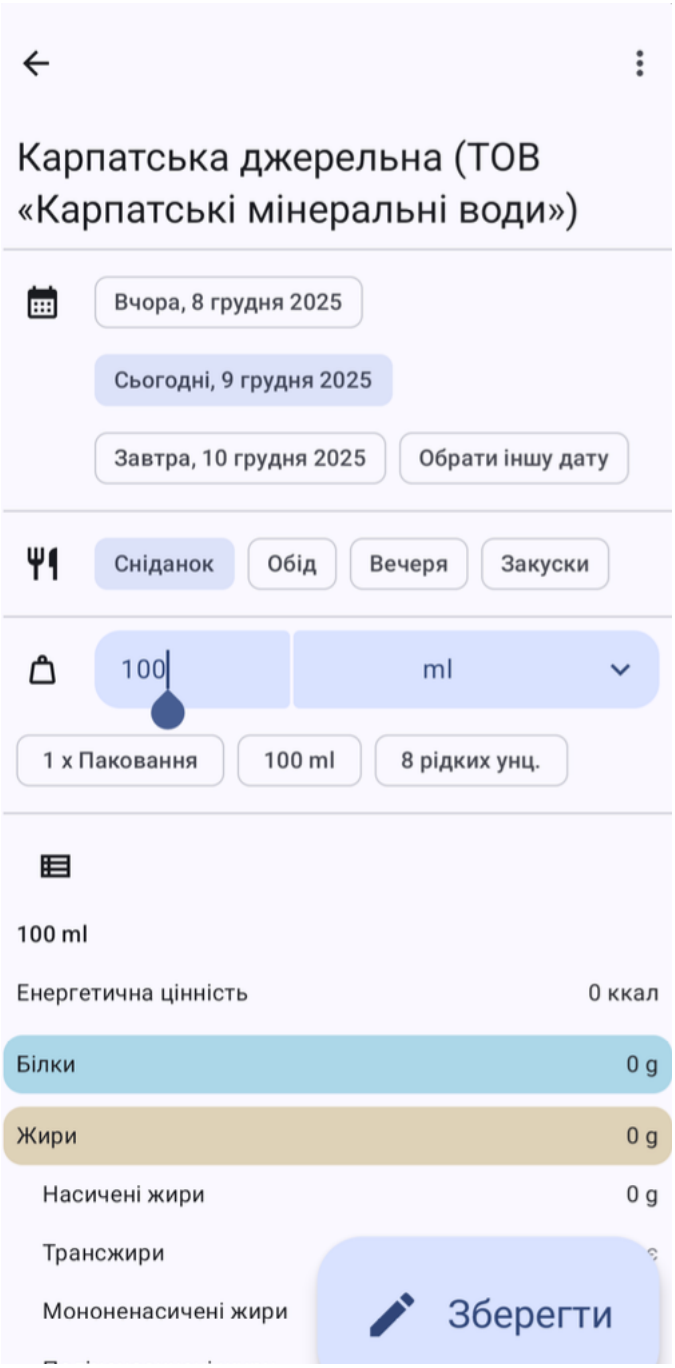


Рисунок 4.9 – Вікно результату пошуку по штрихкоду

4.3 Висновки

Проведене тестування мобільної системи підтвердило:

— коректність реалізації алгоритмів розрахунку добових норм калорій;

— стабільність модуля прогнозування водного балансу;

— надійність взаємодії з OpenFoodFacts API;

— правильну роботу локальної бази даних Room;

— продуктивність системи при великих наборах харчових даних;

— відсутність критичних витоків пам’яті;

— відповідність UI вимогам доступності та юзабіліті.

Під час тестування було виявлено та виправлено низку дефектів, зокрема: помилки у відображенні кириличних назв, deadlock у Room, неточності округлення BMR/TDEE, проблеми з рендерингом UI елементів та некоректне оновлення історії водного балансу.

Загалом система продемонструвала високу стабільність, точність розрахунків та коректність виконання функціональних вимог. Результати тестування підтверджують готовність застосунку до експлуатації та подальших стадій валідації й розгортання.

5 ЕКОНОМІЧНА ЧАСТИНА

Розроблення науково–технічних продуктів у сфері цифрових технологій є доцільним лише за умови їх відповідності сучасним тенденціям розвитку інформаційних систем, ринку мобільних застосунків та очікуванням кінцевих користувачів. Мобільні рішення, що забезпечують підтримку здорового способу життя, формують окремий сегмент ринку, який стабільно зростає, а отже потребує оцінки не лише технічної реалізованості, але й економічної доцільності впровадження.

Магістерська кваліфікаційна робота за темою «Розробка методів і засобів мобільної системи на платформі Android для підтримки здорового способу харчування» відноситься до прикладних науково–технічних проєктів, результатом яких є створення програмного продукту з потенціалом подальшої комерціалізації. Мобільні застосунки у сфері health–tech належать до категорії висококонкурентних ринкових продуктів, тому оцінка економічної ефективності та визначення можливих шляхів монетизації є обов’язковими елементами обґрунтування доцільності розробки.

Актуальність обраного напряму зумовлена зростанням попиту на індивідуалізовані інструменти моніторингу харчування, управління водним балансом, аналізу фізіологічних показників та прогнозування здорової харчової поведінки [30]. Доступність мобільних технологій, поширення носимих пристроїв та зростання інтересу до персоналізованої медицини визначають широкі можливості застосунку для кінцевих користувачів, спортивних організацій, дієтологічних центрів та корпоративних програм здоров’я.

Однак створення такого програмного продукту потребує попереднього аналізу економічних умов його розробки, визначення витрат на виконання робіт, оцінки ринкового потенціалу і конкурентних переваг. Також необхідно провести прогноз економічної ефективності у разі комерційного використання або впровадження продукту у бізнес–модель потенційного інвестора.

У рамках економічної частини роботи виконуються такі основні етапи:

— комерційний аудит науково–технічної розробки, що включає оцінку її конкурентоспроможності, новизни, потенціалу ринку та рівня технологічної готовності;

— розрахунок витрат на виконання науково–дослідної та програмно–технічної роботи, включно з оплатою праці, соціальними відрахуваннями, вартістю необхідного програмного забезпечення, амортизацією обладнання та іншими витратами;

— оцінювання економічної ефективності розробки при можливій комерціалізації, визначення очікуваних фінансових показників (рентабельність, строк окупності, чистий дохід), а також обґрунтування доцільності впровадження.

Виконання зазначених етапів дозволить визначити повну економічну картину проєкту, оцінити його перспективи на ринку мобільних здоров’язберігаючих технологій та сформувати висновок щодо доцільності комерційного використання.

5.1 Комерційний аудит науково–технічної розробки

Метою проведення комерційного й технологічного аудиту науково–технічної розробки за темою «Розробка методів і засобів мобільної системи на Android для підтримки здорового способу харчування» є оцінювання її науково–технічного рівня, інноваційності, можливості практичного застосування, конкурентних переваг та комерційного потенціалу. Комерційний аудит дає можливість визначити доцільність подальшої розробки та перспективи виведення продукту на ринок.

Для проведення оцінювання використовується система критеріїв, згрупованих за напрямами: технічна здійсненність концепції, ринкові переваги, ринкові перспективи та практична здійсненність. Кожен критерій оцінюється за

п’ятибальною шкалою, де 0 означає відсутність позитивних характеристик, а 4 – максимальну оцінку, які наведені в таблиці 5.1 [43].

Таблиця 5.1 – Критерії оцінювання науково–технічного рівня та комерційного потенціалу розробки

Бали (за 5–ти бальною шкалою)					
	0	1	2	3	4
Технічна здійсненність концепції					
1	Концепція не підтверджена	Підтверджена експертно	Підтверджена розрахунками	Перевірена на практичних тестах	Стабільна робота підтверджена у реальних умовах
Ринкові переваги					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог	Аналогів немає
3	Ціна продукту значно вища за аналоги	Ціна продукту трохи вища	Ціна продукту на рівні аналогів	Ціна продукту трохи нижча	Ціна продукту значно нижча
4	Технічні та споживчі властивості значно гірші	Технічні та споживчі властивості трохи гірші	Технічні та споживчі властивості на рівні	Технічні та споживчі властивості трохи кращі	Технічні та споживчі властивості значно кращі
5	Експлуатаційні витрати значно вищі	Експлуатаційні витрати трохи вищі	Експлуатаційні витрати на	Експлуатаційні витрати трохи нижчі	Експлуатаційні витрати Значно нижчі

			рівні конкурентів		
Ринкові перспективи					
6	Розмір ринку та динаміка малий і без зростання	Розмір ринку та динаміка малий, але зі зростанням	Розмір ринку та динаміка середній і зростає	Розмір ринку та динаміка великий і стабільний	Розмір ринку та динаміка великий із позитивною динамікою
7	Конкуренція висока	Конкуренція помітна	Конкуренція помірна	Конкуренція незначна	Конкуренція відсутня
Практична здійсненність					
8	Наявність фахівців: відсутні	Наявність фахівців: потрібно наймати або навчати	Наявність фахівців: часткове навчання	Наявність фахівців: достатньо фахівців	Наявність фахівців: є повний склад спеціалістів
9	Наявність фінансових ресурсів – відсутні	Наявність фінансових ресурсів – обмежені	Наявність фінансових ресурсів – середні	Наявність фінансових ресурсів – достатні	Наявність фінансових ресурсів – забезпечені повністю
10	Необхідність нових матеріалів/технологій – висока	Необхідність нових матеріалів/технологій – значна	Необхідність нових матеріалів/технологій – помірна	Необхідність нових матеріалів/технологій – незначна	Необхідність нових матеріалів/технологій – не потрібні
11	Терміни реалізації – неможливі	Терміни реалізації – дуже великі	Терміни реалізації – значні	Терміни реалізації – оптимальні	Терміни реалізації – короткі

Таблиця 5.2 – Результати експертного оцінювання науково–технічного рівня та комерційного потенціалу

Критерій	**Експерт 1**	**Експерт 2**	**Експерт 3**
1. Технічна здійсненність концепції	4	4	4
2. Ринкові переваги (аналоги)	3	3	3
3. Ринкові переваги (ціна продукту)	4	3	4
4. Ринкові переваги (технічні властивості)	4	4	4
5. Ринкові переваги (експлуатаційні витрати)	3	3	3
6. Ринкові перспективи (розмір ринку)	3	3	3
7. Ринкові перспективи (конкуренція)	3	3	3
8. Практична здійсненність (фахівці)	4	4	4
9. Практична здійсненність (фінанси)	3	3	3
10. Практична здійсненність (ресурси)	3	3	3
11. Практична здійсненність (терміни)	4	4	4
Сума балів	42	41	41
Середньоарифметична сума балів СБс	41,3		

Таблиця 5.3 – Рівні науково–технічного та комерційного потенціалу розробки

Середньоарифметична сума балів	**Рівень потенціалу**
41…48	Високий
31…40	Вище середнього
21…30	Середній
11…20	Нижче середнього
0…10	Низький

Отримана середня оцінка 41,3 бали належить до інтервалу 41–48, що відповідає високому рівню науково–технічного та комерційного потенціалу розробки [44].

Це свідчить, що:

— технологічна концепція мобільної системи є цілком здійсненною та підтвердженою практичними дослідженнями;

— ринок застосунків здорового способу життя демонструє сталу позитивну тенденцію розвитку;

— конкуренція у сегменті адаптивних систем харчування є помірною, а наявні продукти не повністю покривають потреби користувачів;

— експлуатаційні витрати та вимоги до ресурсів є невеликими;

— розробка має реальні перспективи комерціалізації та інвестиційної привабливості.

5.2 Розрахунок витрат на проведення науково–дослідної роботи

Проведення науково–дослідної роботи з розробки мобільної системи для підтримки здорового способу харчування потребує визначення обсягу фінансових ресурсів, необхідних для її реалізації. Витрати включають оплату праці виконавців, нарахування на заробітну плату, амортизацію обладнання, витрати на програмні засоби, допоміжні матеріали та інші статті, пов’язані з організацією і виконанням робіт. Коректний розрахунок собівартості дає змогу оцінити загальну вартість розробки, визначити її економічну доцільність і сформувати підґрунтя для подальшого аналізу ефективності інвестицій.

5.2.1 Витрати на оплату праці

До статті «Витрати на оплату праці» належать витрати на основну заробітну плату керівника проєкту, програмістів, дизайнерів інтерфейсу та консультантів, які залучаються до виконання науково–дослідної роботи.

Розрахунок здійснюється з урахуванням місячного посадового окладу, фактичного часу роботи виконавця та середньої тривалості робочого місяця.

Витрати на основну заробітну плату дослідників визначаються за формулою 5.1.

$$З_o = \sum_{i=1}^{k} \frac{M_{mi} \cdot t_i}{T_p}, \tag{5.1}$$

де k – кількість посад дослідників залучених до процесу досліджень;

Мпі – місячний посадовий оклад конкретного дослідника, грн;

ti – число днів роботи конкретного дослідника, дні;

Тр – середнє число робочих днів в місяці, Тр=22 дні.

Зо = 3000 · 62 / 22 = 84545,45 грн.

Проведені розрахунки зведено до таблиці 5.4.

Таблиця 5.4 – Витрати на заробітну плату дослідників

Найменування посади	Місячний посадовий оклад, грн	Оплата за робочий день, грн	Число днів роботи	Витрати на заробітну плату, грн
Керівник проекту	30000	1363,64	62	84545,45
Інженер–розробник програмного забезпечення	25000	1136,36	62	70454,54
Консультант	18000	818,19	32	26181,82
Дизайнер користувацького інтерфейсу	20000	909,1	22	20000
Всього				201181,81

Витрати на основну заробітну плату робітників (Зр) за відповідними найменуваннями робіт НДР розраховуємо за формулою 5.2:

$$З_p = \sum_{i=1}^{n} C_i \cdot t_i \quad , \tag{5.2}$$

де Сі – погодинна тарифна ставка робітника відповідного розряду, за виконану відповідну роботу, грн/год;

ti – час роботи робітника при виконанні визначеної роботи, год.

Погодинну тарифну ставку робітника відповідного розряду Сі можна визначити за формулою 5.3:

$$C_i = \frac{M_M \cdot K_i \cdot K_c}{T_p \cdot t_{зм}} \quad , \tag{5.3}$$

де ММ – розмір прожиткового мінімуму працездатної особи, абомінімальної місячної заробітної плати (в залежності від діючого законодавства), приймемо Мм=8000,00 грн;

Кі – коефіцієнт міжкваліфікаційного співвідношення для встановлення тарифної ставки робітнику відповідного розряду [43];

Кс – мінімальний коефіцієнт співвідношень місячних тарифних ставок робітників першого розряду з нормальними умовами праці виробничих об’єднань і підприємств до законодавчо встановленого розміру мінімальної заробітної плати.

Тр – середнє число робочих днів в місяці, приблизно Тр = 22 дні;

tзм – тривалість зміни, год.

С1 = 8000,00 · 1,1 · 1,65 / (22/8) = 82,5 грн.

Зр1 = 82,5 · 8,00 = 660 грн.

Величина витрат на основну заробітну плату робітників наведена втабл.5.5.

Таблиця 5.5 – Величина витрат на основну заробітну плату робітників

Найменування робіт	Тривалість роботи, год	Розряд роботи	Тарифний коефіцієнт	Погодинна тарифна ставка, грн	Величина оплати на робітника, грн
Підготовка робочого місця дослідника	8	2	1,1	82,5	660
Інсталяція програмного забезпечення	3	5	1,7	127,5	382,5
Підключення апаратного забезпечення	8	3	1,35	101,25	810
Всього					1852,5

Додаткову заробітну плату розраховуємо як 10 … 12% від суми основної заробітної плати дослідників та робітників за формулою 5.4:

$$З_{дод} = (З_о + З_р) \cdot \frac{Н_{дод}}{100\%}, \quad (5.4)$$

де Ндод – норма нарахування додаткової заробітної плати. Приймемо 11%. Здод = (201181,81 + 1852,5) · 11 / 100% = 22333,77 грн.

5.2.2 Відрахування на соціальні заходи

Нарахування на заробітну плату дослідників та робітників розраховуємо як 22% від суми основної та додаткової заробітної плати дослідників і робітників за формулою 5.5:

$$З_н = (З_о + З_р + З_{дод}) \cdot \frac{Н_{зп}}{100\%}, \quad (5.5)$$

де Нзп – норма нарахування на заробітну плату. Приймаємо 22%.

Зн = (201181,81 + 1852,5 + 22333,77) · 22 / 100% = 49580,97 грн.

5.2.3 Сировина та матеріали

До статті «Сировина та матеріали» належать витрати на сировину, основні та допоміжні матеріали, інструменти, пристрої та інші засоби і предмети праці, які придбані у сторонніх підприємств, установ і організацій та витрачені на проведення досліджень.

Витрати на матеріали (М), у вартісному вираженні розраховуються окремо по кожному виду матеріалів за формулою 5.6:

$$M = \sum_{j=1}^{n} Н_j \cdot Ц_j \cdot К_j - \sum_{j=1}^{n} В_j \cdot Ц_{вj}, \quad (5.6)$$

де Нj – норма витрат матеріалу j–го найменування, кг;

n – кількість видів матеріалів;

Цj – вартість матеріалу j–го найменування, грн/кг;

Кj – коефіцієнт транспортних витрат, (Кj = 1,1 … 1,15);

Вj – маса відходів j–го найменування, кг;

Цвj – вартість відходів j–го найменування, грн/кг.

М1 = 2 · 210,00 · 1,1 – 0,000 · 0,00 = 462 грн.

Проведені розрахунки зведемо до таблиці 5.6.

Таблиця 5.6 – Витрати на матеріали

Найменування матеріалу, марка, тип, сорт	Ціна за од	Норма витрат, од	Величина відходів, кг	Ціна відходів, грн/кг	Вартість витраченого матеріалу, грн
Папір офісний Zoom Stora Enso A4 80 г/м2 клас С 500 аркушів	210	2	0	0	462
Канцелярське приладдя	255	3	0	0	841,5
Картридж для принтера PrintPro NS Canon 725	299	2	0	0	657,8
Флеш пам'ять USB Samsung Bar Plus 128GB USB 3.1	1000	1	0	0	1100
Папка–бокс пластикова Axent A4 на липучці 60 мм	190	1	0	0	209
Всього					3270,3

5.2.4 Програмне забезпечення для наукових робіт

Витрати на програмне забезпечення, які використовують при проведенні НДР на тему «Розробка методів і засобів мобільної системи на платформі Android для підтримки здорового способу харчування» відсутні.

5.2.5 Спецустаткування для наукових (експериментальних) робітника

До статті «Спецустаткування для наукових (експериментальних) робіт» належать витрати на виготовлення та придбання спецустаткування необхідного для проведення досліджень, також витрати на їх проектування, виготовлення,транспортування, монтаж та встановлення.

Витрати на спецустаткування, які використовують при проведенні НДР на тему «Розробка методів і засобів мобільної системи на платформі Android для підтримки здорового способу харчування» відсутні.

5.2.6 Інші витрати

До статті «Інші витрати» належать витрати, які не знайшли відображення у зазначених статтях витрат і можуть бути віднесені безпосередньо на собівартість досліджень за прямими ознаками.

Витрати за статтею «Інші витрати» розраховуємо як 50…100% від суми основної заробітної плати дослідників та робітників за формулою 5.7:

$$I_в = (З_о + З_р) \cdot \frac{Н_{ів}}{100\%}, \quad (5.7)$$

де Нів – норма нарахування за статтею «Інші витрати», приймемо Нів = 55%.

Ів = (201181,81 + 1852,5) · 55 / 100% = 111668,87 грн.

5.2.7 Накладні (загальновиробничі) витрати

До статті «Накладні (загальновиробничі) витрати» належать: витрати, пов’язані з управлінням організацією; витрати на винахідництво та раціоналізацію; витрати на підготовку (перепідготовку) та навчання кадрів; витрати, пов’язані з набором робочої сили; витрати на оплату послуг банків; витрати, пов’язані з освоєнням виробництва продукції; витрати на науковотехнічну інформацію та рекламу та ін.

Витрати за статтею «Накладні (загальновиробничі) витрати» розраховуємо як 100…150% від суми основної заробітної плати дослідників та робітників за формулою 5.8:

$$В_{нзв} = (З_о + З_р) \cdot \frac{Н_{нзв}}{100\%}, \quad (5.8)$$

де Ннзв – норма нарахування за статтею «Накладні (загальновиробничі) витрати», приймемо Ннзв = 125%.

Внзв = (201181,81+ 1852,5) · 100 / 125% = 162427,44 грн.

Витрати на проведення науково–дослідної роботи розраховуємо як суму всіх попередніх статей витрат за формулою 5.9:

$$B_{заг} = З_о + З_р + З_{дод} + З_н + М + К_в + B_{спец} + B_{прг} + A_{обл} + B_в + B_{св} + B_{сп} + I_в + B_{нзв}. \quad (5.9)$$

Взаг=201181,81+1852,5+22333,77+49580,97+3270,3+0,00+0,00+0,00+ 0,00+ 0,00+0,00+0,00+111668,87 +162427,44 = 552315,66 грн.

Загальні витрати ЗВ на завершення науково–дослідної (науково–технічної) роботи та оформлення її результатів розраховується за формулою 5.10:

$$ЗВ = \frac{B_{заг}}{\eta}, \quad (5.10)$$

де η – коефіцієнт, який характеризує етап (стадію) виконання науководослідної роботи, приймемо η =0,75.

ЗВ = 552315,66 / 0,7 = 736420,88 грн.

5.3 Оцінювання економічної ефективності розробки при можливій комерціалізації

В ринкових умовах узагальнюючим позитивним результатом, що його може отримати потенційний інвестор від можливого впровадження результатів тієї чи іншої науково–технічної розробки, є збільшення у потенційного інвестора величини чистого прибутку.

Результати дослідження передбачають вихід продукту на ринок протягом наступних трьох років. При цьому очікуваний економічний ефект буде ґрунтуватися на таких показниках:

ΔN – збільшення кількості споживачів продукту, у періоди часу, що аналізуються, від покращення його певних характеристик;

1–й рік – 3000 користувачів/рік;

2–й рік – 4000 користувачів/рік;

3–й рік – 6000 користувачів/рік.

N – кількість споживачів які використовували аналогічний продукт у році до впровадження результатів нової науково–технічної розробки, приймемо 70000 користувачів;

$Ц_б$ – вартість програмного продукту у році до впровадження результатів розробки, приймемо 5000,00 грн /за рік користування;

$\pm\Delta Ц_о$ – зміна вартості програмного продукту від впровадження результатів науково–технічної розробки, приймемо 600,00 грн.

Можливе збільшення чистого прибутку у потенційного інвестора для кожного із 3–х років, протягом яких очікується отримання позитивних результатів від можливого впровадження та комерціалізації науково–технічної розробки, розраховуємо за формулою 5.11:

$$\Delta П_i = (\pm\Delta Ц_o \cdot N + Ц_o \cdot \Delta N)_i \cdot \lambda \cdot \rho \cdot (1 - \frac{\vartheta}{100}), \quad (5.11)$$

де λ – коефіцієнт, який враховує сплату потенційним інвестором податку на додану вартість. У 2025 році ставка податку на додану вартість складає 20%, а коефіцієнт λ =0,8333;

ρ – коефіцієнт, який враховує рентабельність інноваційного продукту. Приймемо ρ =32%;

ϑ – ставка податку на прибуток, який має сплачувати потенційний інвестор, у 2025 році =18%;

Збільшення чистого прибутку 1–го року:

$\Delta П_1$ = (70000·600,00+5000·3000)·0,83·0,3·(1–0,18/100%) = 14167452,6 грн.

Збільшення чистого прибутку 2–го року:

$\Delta П_2$ = (70000·600,00+5000·(3000+4000))·0,83·0,3·(1–0,18/100%)= 19138488,6 грн.

Збільшення чистого прибутку 3–го року:

$\Delta П_2 =$ (70000·600,00+5000·(3000+4000+5000))·0,83·0,3·(1–0,18/100%) = 25352283,6 грн.

Приведена вартість збільшення всіх чистих прибутків ПП, що їх може отримати потенційний інвестор від можливого впровадження та комерціалізації науково–технічної розробки, розраховується за формулою 5.12:

$$ПП = \sum_{i=1}^{T} \frac{\Delta П_i}{(1+\tau)^t}, \quad (5.12)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному з років, протягом яких виявляються результати впровадження науково–технічної розробки, грн;

T – період часу, протягом якого очікується отримання позитивних результатів від впровадження та комерціалізації науково–технічної розробки, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні, τ =0,1;

t – період часу (в роках) від моменту початку впровадження науковотехнічної розробки до моменту отримання потенційним інвестором додаткових чистих прибутків у цьому році

$ПП$ = 14167452,6/(1+0,1)¹+ 19138488,6 /(1+0,1)² + 25352283,6 /(1+0,1)³= 53325658,90 грн.

Величина початкових інвестицій PV, які потенційний інвестор має вкласти для впровадження і комерціалізації науково–технічної розробки, розраховується за формулою 5.13:

$$PV = k_{інв} \cdot ЗВ, \quad (5.13)$$

де $k_{інв}$ – коефіцієнт, що враховує витрати інвестора на впровадження науково–технічної розробки та її комерціалізацію, приймаємо $k_{інв}$ =4;

ЗВ – загальні витрати на проведення науково–технічної розробки та оформлення її результатів, приймаємо 736420,88 грн.

$PV = k_{інв} \cdot ЗВ$ = 4·736420,88 = 2945683,52 грн

Абсолютний економічний ефект $E_{абс}$ для потенційного інвестора від можливого впровадження та комерціалізації науково–технічної розробки розраховується за формулою 5.14:

$$E_{абс} = ПП - PV, \quad (5.14)$$

де *ПП*– приведена вартість зростання всіх чистих прибутків від можливого впровадження та комерціалізації науково–технічної розробки, 53325658,90 грн;

PV – теперішня вартість початкових інвестицій, 2945683,52грн.

$E_{абс} = ПП - PV$ = 53325658,90 – 2945683,52 = 50379975,38

Внутрішня економічна дохідність інвестицій $E_в$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково–технічної розробки, розраховується за формулою 5.15:

$$E_в = \sqrt[T_ж]{1 + \frac{E_{абс}}{PV}} - 1, \quad (5.15)$$

де $E_{абс}$ – абсолютний економічний ефект вкладених інвестицій, 50379975,38 грн;

PV – теперішня вартість початкових інвестицій, 2945683,52 грн;

$T_ж$ – життєвий цикл науково–технічної розробки, тобто час від початку її розробки до закінчення отримування позитивних результатів від її 106 впровадження, 3 роки.

$E_в = \sqrt[T_ж]{1 + \frac{E_{абс}}{PV}} - 1$ = (1+50379975,38 /2945683,52) -1 = 3,254.

Мінімальна внутрішня економічна дохідність вкладених інвестицій розраховується за формулою 5.16:

$$\tau_{мін} = d + f, \quad (5.16)$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2025 році в Україні d =0,096;

f – показник, що характеризує ризикованість вкладення інвестицій, приймемо 0,35.

$\tau_{мін}$ = 0,096+0,35 = 0,446 < 0,916 свідчить про те, що внутрішня економічна дохідність інвестицій , які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки вища мінімальної внутрішньої дохідності. Тобто інвестувати в науково-дослідну роботу за темою «Розробка методів і засобів мобільної системи на платформі Android для підтримки здорового способу харчування» доцільно.

Період окупності інвестицій які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки, розраховується за формулою 5.17:

$$T_{ок} = \frac{1}{E_в}, \tag{5.17}$$

де $E_в$ – внутрішня економічна дохідність вкладених інвестицій.

$T_{ок}$ = 1 / 0,916 = 1,092 року

$T_{ок} <$ 3-х років, що свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

5.4 Висновки

Згідно з проведеним комерційним аудитом, рівень комерційного потенціалу науково-технічної розробки за темою «Розробка методів і засобів мобільної системи на Android для підтримки здорового способу харчування» становить 41,3 бали, що відповідає високому рівню. Це свідчить про значну комерційну цінність розробки та підтверджує її перспективність для подальшого впровадження.

Результати економічних розрахунків також демонструють позитивні показники ефективності. Зокрема, термін окупності інвестицій менше трьох років, що вказує на високу інвестиційну привабливість проєкту та можливість швидкого повернення вкладених коштів. Така особливість є важливим фактором для потенційних інвесторів, які можуть бути зацікавлені у фінансуванні інтеграції мобільної системи та її виходу на ринок.

Таким чином, отримані результати підтверджують доцільність проведення науково-дослідної роботи та обґрунтовують перспективність розробки мобільної системи для підтримки здорового способу харчування як економічно ефективного й комерційно конкурентоспроможного інноваційного продукту.

ВИСНОВКИ

У результаті виконання магістерської кваліфікаційної роботи було проведено комплексне дослідження та розроблено методи і програмні засоби мобільної системи підтримки здорового способу харчування на платформі Android. У роботі виконано повний цикл створення науково-технічного рішення – від аналізу сучасних мобільних систем та методів персоналізації харчування до реалізації архітектури, алгоритмів, програмних модулів, тестування та оцінки економічної ефективності. Робота оформлена згідно методичних вказівок [45, 46].

У першому розділі було проведено аналіз сучасних мобільних систем харчового моніторингу, огляд аналогів, методів розрахунку добових норм калорій та водного балансу, а також доступних відкритих джерел харчових даних. Встановлено, що наявні рішення не забезпечують достатнього рівня персоналізації, не завжди використовують адаптивні моделі та мають обмеження в роботі з локальними продуктами. Це підтвердило актуальність розробки власної системи, орієнтованої на точність, адаптивність та зручність взаємодії.

У другому розділі було розроблено методи, моделі та алгоритми роботи системи. Запропоновано адаптивний метод визначення добової норми калорій на основі формули Mifflin–St Jeor з урахуванням поведінкових коефіцієнтів і динаміки змін фізіологічних параметрів користувача [9]. Розроблено метод прогнозування водного балансу, що використовує модель ковзного середнього та враховує рівень активності користувача. Побудовано модель даних користувача й харчових продуктів, охарактеризовано структуру системи та створено загальний алгоритм її функціонування. Результати забезпечили науково обґрунтовану основу для розробки програмної частини.

У третьому розділі реалізовано програмні засоби мобільної системи відповідно до розробленої архітектури. Створено програмні модулі: профілю користувача, інтеграції з OpenFoodFacts, підбору денного раціону, трекінгу

водного балансу, а також компонентів інтерфейсу користувача. Застосовано сучасні технології Android-розробки – Kotlin, MVVM, Jetpack, Room, Hilt, Retrofit – що забезпечило модульність, стабільність і масштабованість застосунку [21]. Реалізовано інструмент сканування штрих-кодів, пошук продуктів та механізм ручного доповнення харчової бази [34].

У четвертому розділі проведено тестування функціональних модулів мобільної системи, що включало модульне, інтеграційне та ручне тестування сценаріїв користувача. Тестування підтвердило коректність роботи алгоритмів розрахунку калорійності та водного балансу, стабільність взаємодії з API, правильність зберігання даних та відповідність роботи UI вимогам юзабіліті. Отримані результати засвідчили працездатність і надійність розробленої системи.

У п’ятому розділі виконано комерційний аудит науково-технічної розробки та проведено економічне обґрунтування її реалізації. Рівень комерційного потенціалу системи визначено як високий, що підтверджує перспективність її впровадження на ринок цифрових health-tech продуктів. Проведено розрахунок витрат на розробку, оцінено економічну ефективність та визначено показники окупності. Результати економічних розрахунків свідчать про інвестиційну привабливість розробки та можливість її комерційної реалізації.

Підсумовуючи виконану роботу, можна стверджувати, що поставлена мета – розробка методів і засобів мобільної системи на Android для підтримки здорового способу харчування – досягнута повністю. Розроблена система забезпечує персоналізований моніторинг калорійності та водного балансу, підтримує автоматизовану обробку харчових даних, адаптивні алгоритми, зручний інтерфейс та інтеграцію з відкритими джерелами інформації. Отримані результати мають як наукову новизну, так і практичну цінність, дозволяючи використовувати систему в реальних умовах та розширювати її функціональність у майбутніх розробках.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Центр громадського здоров’я МОЗ України — дослідження: «Між бажанням і дією: здоровий спосіб життя в уявленні українців» [Електронний ресурс] — Режим доступу: https://www.kantar.com/ua/inspiration/consumers/healthy-lifestyle-in-ukraine?

2. Wezom — «Основні тренди в розробці мобільних додатків у 2025 році» [Електронний ресурс] — Режим доступу: https://wezom.com.ua/ua/blog/klyuchovi-trendi-rozvitku-mobilnih-dodatkiv-perspektivi-2025-roku?

3. Бевз С.В., Бурбело С.М., Бабійчук І.С. Використання засобів гейміфікації при створенні мобільних систем спеціалізованого призначення // Комп'ютерні ігри та мультимедіа як інноваційний підхід до комунікації - 2025 // Матеріали V Всеукраїнської науково-технічної конференції молодих вчених, аспірантів і студентів, Одеса, 25–26 вересня 2025 р. – Одеса: Видавництво ОНТУ, 2025. – 505 с.

4. MyFitnessPal [Електронний ресурс] — Режим доступу: https://www.myfitnesspal.com

5. Yazio [Електронний ресурс] — Режим доступу: https://www.yazio.com

6. Lifesum [Електронний ресурс] — Режим доступу: https://www.lifesum.com

7. Cronometer [Електронний ресурс] — Режим доступу: https://cronometer.com

8. Frankenfield D.C., Roth-Yousey L., Compher C. Comparison of Predictive Equations for Resting Metabolic Rate in Healthy Nonobese and Obese Adults. – Journal of the American Dietetic Association, 2005.

9. Mifflin M.D., St Jeor S.T., Hill L.A. A new predictive equation for resting energy expenditure in healthy individuals. – American Journal of Clinical Nutrition, 1990. – Vol. 51, № 2. – P. 241–247.

10. Human Energy Requirements: Report of a Joint FAO/WHO/UNU Expert Consultation. – Rome: FAO, 2004.

11. Скільки води треба пити дорослим і дітям [Електронний ресурс] — Режим доступу: https://moz.gov.ua/uk/skilki-vodi-treba-piti-doroslim-i-ditjam.

12. Popkin B.M., D'Anci K.E., Rosenberg I.H. Water, Hydration and Health. – Nutrition Reviews, 2010.

13. OpenFoodFacts [Електронний ресурс] — Режим доступу: https://openfoodfacts.org

14. FoodData Central (USDA) [Електронний ресурс] — Режим доступу: https://fdc.nal.usda.gov

15. Global Diet and Nutrition Apps Market Share, Changing Dynamics and Growth Forecast 2025–2031 [Електронний ресурс] — Режим доступу: https://www.htfmarketintelligence.com/report/global-diet-and-nutrition-apps-market

16. Guide to App Architecture [Електронний ресурс] — Режим доступу: https://developer.android.com/topic/architecture?hl

17. Джемеров Д., Ісакова С. Kotlin in Action. – Нью-Йорк: Manning, 2017. – 360 p.

18. Kotlin Documentation [Електронний ресурс] — Режим доступу: https://kotlinlang.org/docs/home.html

19. Kotlin Multiplatform Documentation [Електронний ресурс] — Режим доступу: https://kotlinlang.org/docs/multiplatform.html

20. Hilt Dependency Injection – Developer Guide [Електронний ресурс] — Режим доступу: https://developer.android.com/training/dependency-injection/hilt-android?hl

21. Android Programming: The Big Nerd Ranch Guide. — 3rd ed. — Addison-Wesley, 2017. – 624 p.

22. Android Studio User Guide [Електронний ресурс] — Режим доступу: https://developer.android.com/studio/intro?

23. Material Design Guidelines [Електронний ресурс] — Режим доступу: https://m2.material.io/design/guidelines-overview

24. ViewModel Overview [Електронний ресурс] — Режим доступу: https://developer.android.com/topic/libraries/architecture/viewmodel?

25. Kotlin Coroutines Guide [Електронний ресурс] — Режим доступу: https://kotlinlang.org/docs/coroutines-guide.html

26. Kotlin Flow API Reference [Електронний ресурс] — Режим доступу: https://kotlinlang.org/docs/api-references.html

27. Kotlin Serialization Library – Guide [Електронний ресурс] — Режим доступу: https://kotlinlang.org/docs/serialization.html

28. Ufholz K., Werner J. The Efficacy of Mobile Applications for Weight Loss. – Current Cardiovascular Risk Reports, 2023. – Vol. 17(4). – P. 83–90.

29. Mateo G.F., Granado-Font E., Ferré-Grau C., Montaña-Carreras X. Mobile Phone Apps to Promote Weight Loss and Increase Physical Activity: a Systematic Review and Meta-Analysis. – JMIR mHealth and uHealth, 2015.

30. Наказ МОЗ України від 14.01.2013 № 16 «Про затвердження Методичних рекомендацій для лікарів загальної практики–сімейної медицини з консультування пацієнтів щодо основних засад здорового харчування» [Електронний ресурс] — Режим доступу: https://zakon.rada.gov.ua/rada/show/v0016282-13

31. Брич В.В., Білак-Лук’янчук В.Й., Слабкий Г.О. та ін. Здорове харчування: збірник матеріалів для працівників системи охорони здоров’я. – Ужгород, 2020. – 64 с.

32. Johnson D., Deterding S., Kuhn K.A. Gamification for Health and Wellbeing: A Systematic Review of the Literature. – Internet Interventions, 2016.

33. Hamari J., Koivisto J., Sarsa H. Does Gamification Work? – A Literature Review of Empirical Studies on Gamification.

34. ML Kit Barcode Scanning API Documentation [Електронний ресурс] — Режим доступу: https://developers.google.com/ml-kit/vision/barcode-scanning

35. ZXing Barcode Scanner Library [Електронний ресурс] — Режим доступу: https://github.com/zxing/zxing

36. World Health Organization (WHO). Healthy diet [Електронний ресурс] — Режим доступу: https://www.who.int/news-room/fact-sheets/detail/healthy-diet (2020)

37. Лотоцька-Дудик У.Б., Брейдак О.А. Нутриціологія: навчальний посібник. – Львів: ЛНМУ ім. Д. Галицького, 2020. – 123 с.

38. Mobile Operating System Market Share Worldwide [Електронний ресурс] — Режим доступу: https://gs.statcounter.com/os-market-share/mobile/worldwide

39. Sawka M.N., Burke L.M., Eichner E.R. American College of Sports Medicine Position Stand: Exercise and Fluid Replacement. – Medicine & Science in Sports & Exercise, 2007.

40. Захаріна Є.А., Мазін В., Шутко А. Використання мобільного застосунку для оптимізації харчування в силовому фітнесі. – Sport Science Spectrum, 2024.

41. U.S. Department of Agriculture; U.S. Department of Health and Human Services. Dietary Guidelines for Americans, 2020–2025. – 9th Edition. – Washington, DC: U.S. Government Printing Office, 2020.

42. Наказ МОЗ України від 03.09.2017 № 1073 «Про затвердження Норм фізіологічних потреб населення України в основних харчових речовинах і енергії» [Електронний ресурс] — Режим доступу: https://zakon.rada.gov.ua/laws/show/z1206-17

43. Кавецький В. В. Економічне обґрунтування інноваційних рішень: практикум / В. В. Кавецький, В. О. Козловський, І. В. Причепа. – Вінниця: ВНТУ, 2016. – 113 с.

44. Методичні вказівки до виконання економічної частини магістерських кваліфікаційних робіт / Уклад. В. О. Козловський, О. Й. Лесько, В. В. Кавецький. – Вінниця: ВНТУ, 2021. – 42 с.

45. Положення про кваліфікаційну роботу на другому (вищому) рівні вищої освіти. Розробники: А. О. Семенов, Л. П. Громова, Т. В. Макарова, О. В. Сердюк. – Вінниця: ВНТУ, 2021. – 60 с.

46. Методичні вказівки до виконання магістерської кваліфікаційної роботи для студентів спеціальності 121 «Інженерія програмного забезпечення» / уклад. О. Н. Романюк, Г. О. Черноволик. – Вінниця: ВНТУ, 2022. – 50 с.

ДОДАТКИ

137

ДОДАТОК А. Технічне завдання

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
д.т.н., професор
О. Н. Романюк
«26» вересня 2025 р.

Технічне завдання
на магістерську кваліфікаційну роботу
«Розробка методів і програмних засобів мобільної системи на платформі Android для підтримки здорового способу харчування»
за спеціальністю 121 – Інженерія програмного забезпечення

Керівник магістерської кваліфікаційної роботи:
к.т.н., доц. каф. ПЗ В.В. Войтко
«26» вересня 2025 р.
Виконав:
студент гр. 1ПІ–24М І.С. Бабійчук
«26» вересня 2025 р.

Вінниця – 2025 року

1. Найменування та галузь застосування

Магістерська кваліфікаційна робота: «Розробка методів і програмних засобів мобільної системи на платформі Android для підтримки здорового способу харчування».

Галузь застосування – Android-застосунки для здорового способу життя.

2. Підстава для розробки

Підставою для виконання магістерської кваліфікаційної роботи є індивідуальне завдання на МКР і наказ №313 від «24» вересня 2025 р. ректора по ВНТУ про закріплення тем МКР.

3. Мета та призначення розробки

Метою магістерської роботи є підвищення можливостей підтримки здорового способу харчування шляхом розробки методів і засобів мобільної системи на платформі Android з оптимізованими алгоритмами збору, обробки та подання даних про харчування та водний баланс, що надає користувачам розвинутий функціонал системи для підтримки свого здоров'я.

Призначення роботи – розробка методів та засобів розробки Android–застосунку для персоналізованого контролю харчування та водного балансу.

4. Вихідні дані для проведення НДР

Перелік основних літературних джерел, на основі яких буде виконуватись МКР:

1. World Health Organization (WHO). Healthy diet [Електронний ресурс] — Режим доступу: https://www.who.int/news-room/fact-sheets/detail/healthy-diet

2. (2020).

3. Джемеров Д., Ісакова С. Kotlin in Action. – Нью-Йорк: Manning, 2017. – 360 с.

4. Phillips B., Stewart C., Hardy B. Android Programming: The Big Nerd Ranch Guide. – 3rd ed. – Addison-Wesley, 2017. – 624 с.

5. Guide to App Architecture [Електронний ресурс] — Android Developers, 2022.

6. OpenFoodFacts [Електронний ресурс] — Режим доступу: https://openfoodfacts.org

7. Kotlin Documentation [Електронний ресурс] — JetBrains, 2025.

8. Mateo G.F., Granado-Font E., Ferré-Grau C., Montaña-Carreras X. Mobile Phone Apps to Promote Weight Loss and Increase Physical Activity: a Systematic Review and Meta-Analysis. – JMIR mHealth and uHealth, 2015.

9. Бевз С.В., Бурбело С.М., Бабійчук І.С. Використання засобів гейміфікації при створенні мобільних систем спеціалізованого призначення. // Матеріали V Всеукраїнської науково-технічної конференції молодих вчених, аспірантів і студентів. – Одеса: ОНТУ, 2025. – 505 с.

5. Технічні вимоги

Операційна система – Android 8.0 (API level 26) або вище.Мова програмування – Kotlin. Архітектурний шаблон – MVVM. Інструменти розробки – Android Studio Hedgehog або новіша. Бібліотеки та фреймворки – Jetpack ViewModel, LiveData, Room, Hilt, Retrofit, Kotlin Coroutines, ML Kit (Barcode Scanning). Формат даних – JSON, API-запити через HTTPS. Джерела даних – OpenFoodFacts API, FoodData Central (USDA). Графічний інтерфейс – Material Design 3, адаптивна верстка для екранів різного DPI та розмірів. Функціональні компоненти – сканування штрих-кодів, підбір раціону, облік калорій, рекомендації з гідратації, гейміфікація прогресу користувача. Вимоги до безпеки – локальне збереження даних з використанням шифрування, обмеження доступу до персональної інформації. Вихідні дані – персоналізовані щоденні плани харчування, водного балансу та звіт з прогресом, сформовані на основі введених даних та рекомендаційної логіки.

6. Конструктивні вимоги

нтерфейс мобільного застосунку повинен відповідати сучасним вимогам ергономіки, естетики та доступності. Усі елементи керування мають бути інтуїтивно зрозумілими, зручними для користувачів з різним рівнем технічної

підготовки. Забезпечується підтримка темної/світлої тем, масштабування тексту та жестового управління.

Графічна та текстова документація повинна відповідати діючим стандартам України.

7. Перелік технічної документації, що пред'являється по закінченню робіт:

– пояснювальна записка до МКР;

– технічне завдання;

– лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки:

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи
1	Аналіз стану систем підтримки здорового харчування	26.09. 2025 - 05.10.2025
2	Розробка методів, моделей та алгоритмів системи	06.10. 2025 - 22.10.2025
3	Розробка програмних засобів системи	23.10. 2025 - 06.11.2025
4	Тестування системи	07.11. 2025 - 21.11.2025
5	Економічна частина	22.11.2025- 30.11.2025
6	Оформлення матеріалів до захисту МКР	01.12.25 – 09.12.25

10. Порядок контролю та прийняття

Виконання етапів магістерської кваліфікаційної роботи контролюється керівником згідно з графіком виконання роботи.

Прийняття магістерської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком.

141

ДОДАТОК Б. Протокол перевірки МКР на плагіат

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Розробка методів і програмних засобів мобільної системи на платформі Android для підтримки здорового способу харчування

Тип роботи: магістерська кваліфікаційна робота
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ кафедра програмного забезпечення, ФІТКІ, 1ПІ–24М
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 1 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

- ■ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту
- ☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.
- ☐ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

_____ _____
(прізвище, ініціали, посада) (підпис)

_____ _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ Черноволик Г. О.

З висновком експертної комісії ознайомлений(–на)

Керівник _____ Войтко В.В. к.т.н., доц. каф. ПЗ
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Бабійчук І.С.
(підпис) (прізвище, ініціали)

ДОДАТОК В

Лістинг коду

```
/* --- Data Models --- */

@Entity(tableName = "food_items")
data class FoodItem(
    @PrimaryKey(autoGenerate = true) val id: Int = 0,
    val name: String,
    val calories: Double,
    val protein: Double,
    val fat: Double,
    val carbs: Double,
    val barcode: String?
)

@Entity(tableName = "user_profile")
data class UserProfile(
    @PrimaryKey val uid: Int = 1,
    val age: Int,
    val gender: String,
    val weight: Double,
    val height: Double,
    val activityLevel: String
)

@Entity(tableName = "daily_logs")
data class DailyLog(
    @PrimaryKey(autoGenerate = true) val logId: Int = 0,
    val date: String,
    val totalCalories: Double,
```

```
    val totalWaterMl: Int
)

/* --- DAO Interfaces --- */

@Dao
interface FoodDao {
    @Query("SELECT * FROM food_items")
    fun getAllFoods(): LiveData<List<FoodItem>>

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insertFood(item: FoodItem)

    @Query("SELECT * FROM food_items WHERE barcode = :barcode LIMIT
1")
    suspend fun findByBarcode(barcode: String): FoodItem?
}

@Dao
interface UserProfileDao {
    @Query("SELECT * FROM user_profile LIMIT 1")
    fun getUserProfile(): LiveData<UserProfile?>

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun saveProfile(profile: UserProfile)
}

/* --- Room Database --- */
```

```
@Database(entities = [FoodItem::class, UserProfile::class, DailyLog::class], version = 1)
abstract class AppDatabase : RoomDatabase() {
    abstract fun foodDao(): FoodDao
    abstract fun userProfileDao(): UserProfileDao
}

/* --- Retrofit API Service --- */

interface FoodApiService {
    @GET("/api/v0/product/{barcode}.json")
    suspend fun getProductByBarcode(@Path("barcode") barcode: String): Response<OpenFoodProduct>
}

/* --- ViewModel Example --- */

@HiltViewModel
class FoodViewModel @Inject constructor(
    private val foodDao: FoodDao,
    private val apiService: FoodApiService
) : ViewModel() {

    val allFoods: LiveData<List<FoodItem>> = foodDao.getAllFoods()

    fun addFood(food: FoodItem) {
        viewModelScope.launch {
            foodDao.insertFood(food)
        }
    }
```

```
    fun fetchFoodByBarcode(barcode: String) {
        viewModelScope.launch {
            val response = apiService.getProductByBarcode(barcode)
            if (response.isSuccessful) {
                val product = response.body()?.product
                product?.let {
                    val food = FoodItem(
                        name = it.productName ?: "Unknown",
                        calories = it.nutriments.energyKcal100g,
                        protein = it.nutriments.proteins100g,
                        fat = it.nutriments.fat100g,
                        carbs = it.nutriments.carbohydrates100g,
                        barcode = barcode
                    )
                    foodDao.insertFood(food)
                }
            }
        }
    }
}

/* --- Fragment Example (UI Layer) --- */

@AndroidEntryPoint
class FoodListFragment : Fragment(R.layout.fragment_food_list) {

    private val viewModel: FoodViewModel by viewModels()

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

```
            super.onViewCreated(view, savedInstanceState)

            val recyclerView =
view.findViewById<RecyclerView>(R.id.recyclerFoodList)
            val adapter = FoodAdapter()
            recyclerView.adapter = adapter

            viewModel.allFoods.observe(viewLifecycleOwner) {
                adapter.submitList(it)
            }
        }
    }

    /* --- BarcodeScannerHelper.kt --- */

    object BarcodeScannerHelper {
        fun startScanner(activity: FragmentActivity, callback: (String) -> Unit) {
            val scanner = BarcodeScanning.getClient()
            val intent = Intent(activity, BarcodeCaptureActivity::class.java)
            activity.startActivityForResult(intent, 1001)
        }
    }

    /* --- RecommendationUtils.kt --- */

    object RecommendationUtils {
        fun calculateCalories(profile: UserProfile): Double {
            return when (profile.gender) {
                "male" -> 10 * profile.weight + 6.25 * profile.height - 5 * profile.age +
5
```

```
            "female" -> 10 * profile.weight + 6.25 * profile.height - 5 * profile.age -
161
            else -> 2000.0
        }
    }

    fun calculateWaterIntake(profile: UserProfile): Int {
        return (profile.weight * 35).toInt() // мл на кг ваги
    }
}

/* --- AuthManager.kt --- */

object AuthManager {
    fun isAuthenticated(context: Context): Boolean {
        val prefs = context.getSharedPreferences("app_prefs",
Context.MODE_PRIVATE)
        return prefs.getBoolean("logged_in", false)
    }

    fun login(context: Context, username: String, password: String): Boolean {
        if (username == "demo" && password == "1234") {
            context.getSharedPreferences("app_prefs",
Context.MODE_PRIVATE).edit()
                .putBoolean("logged_in", true).apply()
            return true
        }
        return false
    }
```

```
        fun logout(context: Context) {
            context.getSharedPreferences("app_prefs",
Context.MODE_PRIVATE).edit()
                .putBoolean("logged_in", false).apply()
        }
    }

    /* --- StatsCalculator.kt --- */

    object StatsCalculator {
        fun getCaloriesProgress(today: Double, goal: Double): Float {
            return (today / goal).coerceIn(0f, 1f)
        }

        fun getWaterProgress(today: Int, goal: Int): Float {
            return (today.toFloat() / goal).coerceIn(0f, 1f)
        }
    }

    /* --- NotificationHelper.kt --- */

    class NotificationHelper(private val context: Context) {
        private val manager =
context.getSystemService(Context.NOTIFICATION_SERVICE) as
NotificationManager

        fun sendReminder(title: String, message: String) {
            val channelId = "reminder_channel"
            if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
```

```
            val channel = NotificationChannel(channelId, "Нагадування",
NotificationManager.IMPORTANCE_DEFAULT)
            manager.createNotificationChannel(channel)
        }

        val notification = NotificationCompat.Builder(context, channelId)
            .setContentTitle(title)
            .setContentText(message)
            .setSmallIcon(R.drawable.ic_notification)
            .build()

        manager.notify(System.currentTimeMillis().toInt(), notification)
    }
}

/* --- UserSettings.kt --- */

data class UserSettings(
    val darkMode: Boolean = false,
    val remindersEnabled: Boolean = true,
    val preferredUnits: String = "metric"
)

object SettingsManager {
    fun loadSettings(context: Context): UserSettings {
        val prefs = context.getSharedPreferences("user_settings",
Context.MODE_PRIVATE)
        return UserSettings(
            darkMode = prefs.getBoolean("dark_mode", false),
            remindersEnabled = prefs.getBoolean("reminders", true),
```

```
            preferredUnits = prefs.getString("units", "metric") ?: "metric"
        )
    }

    fun saveSettings(context: Context, settings: UserSettings) {
        val prefs = context.getSharedPreferences("user_settings",
Context.MODE_PRIVATE).edit()
        prefs.putBoolean("dark_mode", settings.darkMode)
        prefs.putBoolean("reminders", settings.remindersEnabled)
        prefs.putString("units", settings.preferredUnits)
        prefs.apply()
    }
}

/* --- LocalizationHelper.kt --- */

object LocalizationHelper {
    fun setLocale(context: Context, languageCode: String) {
        val locale = Locale(languageCode)
        Locale.setDefault(locale)
        val config = context.resources.configuration
        config.setLocale(locale)
        context.resources.updateConfiguration(config,
context.resources.displayMetrics)
    }
}

/* --- MealPlanner.kt --- */

data class Meal(val name: String, val calories: Double, val time: String)
```

```
object MealPlanner {
    fun generateDailyMeals(targetCalories: Double): List<Meal> {
        val baseMeals = listOf(
            Meal("Сніданок: вівсянка з фруктами", 350.0, "08:00"),
            Meal("Обід: курка з овочами", 550.0, "13:00"),
            Meal("Вечеря: салат з тунцем", 450.0, "19:00")
        )

        val total = baseMeals.sumOf { it.calories }
        val correctionFactor = targetCalories / total

        return baseMeals.map {
            it.copy(calories = it.calories * correctionFactor)
        }
    }
}

/* --- ImageClassifier.kt --- */

class ImageClassifier(private val context: Context) {
    private val labels = listOf("яблуко", "банан", "бургер", "піца")

    fun classify(imageBitmap: Bitmap): String {
        // Імітація класифікації зображення їжі (спрощена)
        return labels.random()
    }
}

/* --- AnalyticsTracker.kt --- */
```

```
object AnalyticsTracker {
    fun logEvent(context: Context, event: String, details: String = "") {
        Log.d("Analytics", "Подія: $event | Деталі: $details")
        // Можна під'єднати Firebase або інші сервіси аналітики
    }
}

/* --- BadgeManager.kt --- */

data class Badge(val name: String, val description: String, val earned: Boolean)

object BadgeManager {
    fun getEarnedBadges(calories: Double, water: Int): List<Badge> {
        val badges = mutableListOf<Badge>()
        if (calories in 1800.0..2200.0) {
            badges.add(Badge("Баланс харчування", "Ви спожили оптимальну кількість калорій", true))
        }
        if (water >= 2000) {
            badges.add(Badge("Гідрація дня", "Ви випили понад 2 літри води", true))
        }
        return badges
    }
}

/* --- 23. FeedbackSender.kt --- */

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

```
    fun send(feedback: String) {
        Log.i("Feedback", "Надіслано відгук: $feedback")
        Toast.makeText(context, "Дякуємо за відгук!",
Toast.LENGTH_SHORT).show()
    }
}

/* --- ThemeHelper.kt --- */

object ThemeHelper {
    fun applyTheme(darkMode: Boolean) {
        AppCompatDelegate.setDefaultNightMode(
            if (darkMode) AppCompatDelegate.MODE_NIGHT_YES
            else AppCompatDelegate.MODE_NIGHT_NO
        )
    }
}

/* --- Hydration Models --- */

@Entity(tableName = "water_intake")
data class WaterRecord(
    @PrimaryKey(autoGenerate = true) val id: Int = 0,
    val date: String,
    val amountMl: Int,
    val timestamp: Long
)

data class HydrationTarget(
    val baseIntake: Int,
```

```
    val adjustedIntake: Int,
    val activityLevel: Int
)

/* --- Hydration DAO --- */

@Dao
interface HydrationDao {
    @Query("SELECT * FROM water_intake WHERE date = :date")
    fun getWaterByDate(date: String): LiveData<List<WaterRecord>>

    @Insert
    suspend fun insert(record: WaterRecord)

    @Query("DELETE FROM water_intake")
    suspend fun clearAll()
}

/* --- Activity Tracking --- */

@Entity(tableName = "activity_logs")
data class ActivityLog(
    @PrimaryKey(autoGenerate = true) val id: Int = 0,
    val steps: Int,
    val caloriesBurned: Double,
    val date: String
)

@Dao
interface ActivityDao {
```

```
    @Query("SELECT * FROM activity_logs WHERE date = :date")
    suspend fun getDailyActivity(date: String): ActivityLog?

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insert(log: ActivityLog)
}

/* --- Hydration Calculator --- */

object HydrationCalculator {

    fun baseIntake(weightKg: Double): Int =
        (weightKg * 30).toInt()

    fun adjustedIntake(base: Int, activityLevel: Int): Int =
        base + activityLevel * 350
}

/* --- Hydration ViewModel --- */

@HiltViewModel
class HydrationViewModel @Inject constructor(
    private val hydrationDao: HydrationDao,
    private val userDao: UserProfileDao,
    private val activityDao: ActivityDao
) : ViewModel() {

    private val _hydrationState = MutableLiveData<HydrationTarget>()
    val hydrationState: LiveData<HydrationTarget> = _hydrationState
```

```
        fun loadHydrationTargets(date: String) {
            viewModelScope.launch {
                val profile = userDao.getUserProfile().value ?: return@launch
                val activity = activityDao.getDailyActivity(date)
                val base = HydrationCalculator.baseIntake(profile.weight)
                val adjusted = HydrationCalculator.adjustedIntake(base,
activity?.steps?.let { stepsToLevel(it) } ?: 0)

                _hydrationState.postValue(
                    HydrationTarget(
                        baseIntake = base,
                        adjustedIntake = adjusted,
                        activityLevel = activity?.steps ?: 0
                    )
                )
            }
        }

        fun addWater(amount: Int) {
            viewModelScope.launch {
                hydrationDao.insert(
                    WaterRecord(
                        date = getToday(),
                        amountMl = amount,
                        timestamp = System.currentTimeMillis()
                    )
                )
            }
        }
```

```
    private fun stepsToLevel(steps: Int): Int =
        when {
            steps < 3000 -> 0
            steps < 7000 -> 1
            else -> 2
        }
}

/* --- Daily Worker for Hydration Reset --- */

class DailyHydrationWorker(
    appContext: Context,
    params: WorkerParameters
) : CoroutineWorker(appContext, params) {

    @Inject lateinit var hydrationDao: HydrationDao

    override suspend fun doWork(): Result {
        hydrationDao.clearAll()
        return Result.success()
    }
}

/* --- ML Kit Barcode Scanner Implementation --- */

class MlKitScannerManager(
    private val scanner: BarcodeScanner = BarcodeScanning.getClient()
) {

    suspend fun scan(image: InputImage): String? {
```

```
        return try {
            val barcodes = scanner.process(image).await()
            barcodes.firstOrNull()?.rawValue
        } catch (e: Exception) {
            null
        }
    }
}

/* --- Food Consumption Repository --- */

class FoodConsumptionRepository @Inject constructor(
    private val diaryDao: DiaryDao,
) {

    suspend fun logFood(foodId: Long, grams: Int) {
        diaryDao.insertDiaryEntry(
            DiaryEntry(
                foodId = foodId,
                grams = grams,
                timestamp = System.currentTimeMillis(),
                date = getToday()
            )
        )
    }

    suspend fun getDailyConsumption(date: String) =
        diaryDao.getDiaryByDate(date)
}
```

```
/* --- Meal Selection Engine --- */

class MealSelectionEngine {

    fun pickBestMatch(
        candidates: List<FoodEntity>,
        targetCalories: Double
    ): FoodEntity? {
        return candidates.minByOrNull {
            kotlin.math.abs(it.calories - targetCalories)
        }
    }

    fun groupByMealType(products: List<FoodEntity>): Map<String, List<FoodEntity>> =
        products.groupBy {
            when {
                it.carbs > it.fat && it.protein < 10 -> "breakfast"
                it.calories > 200 && it.fat > 5 -> "lunch"
                else -> "dinner"
            }
        }
}

/* --- DailyRationManager.kt --- */

class DailyRationManager @Inject constructor(
    private val productRepo: FoodRepository
) {
```

```
    suspend fun generateRation(target: Double): List<Meal> {
        val allProducts = productRepo.getAllProducts()

        val (breakfastList, lunchList, dinnerList) =
            MealSelectionEngine().groupByMealType(allProducts)
                .let { Triple(it["breakfast"], it["lunch"], it["dinner"]) }

        return listOfNotNull(
            breakfastList?.let { Meal("Сніданок", target * 0.3, "08:00") },
            lunchList?.let { Meal("Обід", target * 0.4, "13:00") },
            dinnerList?.let { Meal("Вечеря", target * 0.3, "19:00") }
        )
    }
}

/* --- UI Models --- */

data class DailySummaryUi(
    val calories: Double,
    val water: Int,
    val caloriesProgress: Float,
    val waterProgress: Float
)

/* --- Summary ViewModel --- */

@HiltViewModel
class SummaryViewModel @Inject constructor(
    private val foodRepo: FoodConsumptionRepository,
    private val hydrationDao: HydrationDao,
```

```
        private val profileDao: UserProfileDao
    ) : ViewModel() {

        val summary: LiveData<DailySummaryUi> =
            liveData {
                val profile = profileDao.getUserProfile().value ?: return@liveData
                val today = getToday()

                val foodLogs = foodRepo.getDailyConsumption(today)
                val waterLogs = hydrationDao.getWaterByDate(today).value ?:
emptyList()

                val foodCalories = foodLogs.sumOf { it.calories }
                val water = waterLogs.sumOf { it.amountMl }

                val calorieGoal = RecommendationUtils.calculateCalories(profile)
                val waterGoal = RecommendationUtils.calculateWaterIntake(profile)

                emit(
                    DailySummaryUi(
                        calories = foodCalories,
                        water = water,
                        caloriesProgress =
StatsCalculator.getCaloriesProgress(foodCalories, calorieGoal),
                        waterProgress = StatsCalculator.getWaterProgress(water,
waterGoal)
                    )
                )
            }
    }
```

```
/* --- Mapper Classes --- */

object FoodMapper {
    fun remoteToEntity(remote: FoodRemoteModel): FoodEntity =
        FoodEntity(
            id = 0,
            name = remote.productName ?: "Unknown",
            calories = remote.nutriments.energyKcal100g,
            protein = remote.nutriments.proteins100g,
            fat = remote.nutriments.fat100g,
            carbs = remote.nutriments.carbohydrates100g,
            barcode = remote.code
        )
}

object DiaryMapper {
    fun toUi(entries: List<DiaryEntry>, foods: List<FoodEntity>):
List<DiaryUiItem> {
        return entries.map { entry ->
            val product = foods.firstOrNull { it.id == entry.foodId }
            DiaryUiItem(
                name = product?.name ?: "Невідомий продукт",
                calories = product?.calories?.times(entry.grams / 100.0) ?: 0.0,
                grams = entry.grams,
                timestamp = entry.timestamp
            )
        }
    }
}
```

```
/* --- Onboarding Models --- */

data class OnboardingPage(
    val title: String,
    val description: String,
    val illustrationRes: Int
)

data class OnboardingState(
    val currentPage: Int = 0,
    val totalPages: Int = 0,
    val isFinished: Boolean = false
)

/* --- OnboardingViewModel.kt --- */

@HiltViewModel
class OnboardingViewModel @Inject constructor(
    private val settingsDataStore: SettingsDataStore
) : ViewModel() {

    private val pages = listOf(
        OnboardingPage(
            title = "Ласкаво просимо",
            description = "Додаток допоможе контролювати харчування та водний баланс.",
            illustrationRes = R.drawable.onboarding_1
        ),
        OnboardingPage(
```

```
            title = "Сканування продуктів",
            description = "Використовуйте штрихкоди для швидкого додавання продуктів.",
            illustrationRes = R.drawable.onboarding_2
        ),
        OnboardingPage(
            title = "Персональні рекомендації",
            description = "Отримуйте щоденні поради відповідно до ваших цілей.",
            illustrationRes = R.drawable.onboarding_3
        )
    )

    private val _state = MutableLiveData(
        OnboardingState(currentPage = 0, totalPages = pages.size)
    )
    val state: LiveData<OnboardingState> = _state

    fun getPages(): List<OnboardingPage> = pages

    fun nextPage() {
        val current = _state.value ?: return
        if (current.currentPage + 1 < current.totalPages) {
            _state.value = current.copy(currentPage = current.currentPage + 1)
        } else {
            finishOnboarding()
        }
    }

    private fun finishOnboarding() {
```

```
            viewModelScope.launch {
                settingsDataStore.setOnboardingCompleted(true)
                _state.postValue(_state.value?.copy(isFinished = true))
            }
        }
    }

    /* --- SettingsDataStore.kt (Jetpack DataStore) --- */

    class SettingsDataStore @Inject constructor(
        @ApplicationContext private val context: Context
    ) {

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

        private object Keys {
            val ONBOARDING_COMPLETED = booleanPreferencesKey("onboarding_completed")
            val REMINDERS_ENABLED = booleanPreferencesKey("reminders_enabled")
            val LANGUAGE = stringPreferencesKey("language")
        }

        val onboardingCompletedFlow: Flow<Boolean> =
            context.dataStore.data.map { it[Keys.ONBOARDING_COMPLETED] ?: false }

        suspend fun setOnboardingCompleted(value: Boolean) {
            context.dataStore.edit { preferences ->
```

```
            preferences[Keys.ONBOARDING_COMPLETED] = value
        }
    }

    val remindersEnabledFlow: Flow<Boolean> =
        context.dataStore.data.map { it[Keys.REMINDERS_ENABLED] ?: true }

    suspend fun setRemindersEnabled(value: Boolean) {
        context.dataStore.edit { prefs ->
            prefs[Keys.REMINDERS_ENABLED] = value
        }
    }

    suspend fun setLanguage(languageCode: String) {
        context.dataStore.edit { prefs ->
            prefs[Keys.LANGUAGE] = languageCode
        }
    }

    val languageFlow: Flow<String> =
        context.dataStore.data.map { it[Keys.LANGUAGE] ?: "uk" }
}

/* --- History Models --- */

data class DailyHistoryItem(
    val date: String,
    val calories: Double,
    val water: Int,
    val weight: Double?
```

```
)

/* --- HistoryDao.kt --- */

@Dao
interface HistoryDao {

    @Query("""
        SELECT logs.date AS date,
               logs.totalCalories AS calories,
               logs.totalWaterMl AS water
        FROM daily_logs AS logs
        ORDER BY logs.date DESC
        LIMIT :limit
    """)
    suspend fun getHistory(limit: Int = 30): List<HistoryRow>

    data class HistoryRow(
        val date: String,
        val calories: Double,
        val water: Int
    )
}

/* --- HistoryRepository.kt --- */

class HistoryRepository @Inject constructor(
    private val historyDao: HistoryDao,
    private val weightDao: WeightDao
) {
```

```
    suspend fun getDailyHistory(limit: Int = 30): List<DailyHistoryItem> {
        val rows = historyDao.getHistory(limit)
        val weights = weightDao.getWeightsForRange(rows.map { it.date })

        return rows.map { row ->
            DailyHistoryItem(
                date = row.date,
                calories = row.calories,
                water = row.water,
                weight = weights[row.date]
            )
        }
    }
}

/* --- WeightDao.kt --- */

@Dao
interface WeightDao {

    @Query("SELECT date, weight FROM weight_logs WHERE date IN (:dates)")
    suspend fun getWeightsForRange(dates: List<String>): List<WeightRow>

    data class WeightRow(
        val date: String,
        val weight: Double
    )
}
```

```
/* --- HistoryViewModel.kt --- */

@HiltViewModel
class HistoryViewModel @Inject constructor(
    private val historyRepository: HistoryRepository
) : ViewModel() {

    private val _history = MutableLiveData<List<DailyHistoryItem>>()
    val history: LiveData<List<DailyHistoryItem>> = _history

    fun loadHistory() {
        viewModelScope.launch {
            val data = historyRepository.getDailyHistory()
            _history.postValue(data)
        }
    }
}

/* --- ExportManager.kt --- */

class ExportManager @Inject constructor(
    private val context: Context,
    private val historyRepository: HistoryRepository
) {

    suspend fun exportToJson(): File? {
        val history = historyRepository.getDailyHistory(90)
        if (history.isEmpty()) return null
```

```
        val jsonArray = JSONArray()
        history.forEach { item ->
            val obj = JSONObject().apply {
                put("date", item.date)
                put("calories", item.calories)
                put("water", item.water)
                put("weight", item.weight ?: JSONObject.NULL)
            }
            jsonArray.put(obj)
        }

        val exportFile = File(context.cacheDir, "health_export.json")
        exportFile.writeText(jsonArray.toString(2))
        return exportFile
    }
}

/* --- RemoteSyncService.kt (заглушка для майбутнього бекенду) --- */

class RemoteSyncService @Inject constructor() {

    suspend fun syncProfile(profile: UserProfile): Boolean {
        // TODO: Реалізувати реальну синхронізацію з бекендом
        delay(300)
        return true
    }

    suspend fun syncHistory(history: List<DailyHistoryItem>): Boolean {
        delay(500)
        return true
```

```
    }
}

/* --- ErrorHandler.kt --- */

object ErrorHandler {

    fun mapThrowableToMessage(t: Throwable): String =
        when (t) {
            is UnknownHostException -> "Відсутнє підключення до інтернету"
            is SocketTimeoutException -> "Перевищено час очікування відповіді"
            else -> "Сталася невідома помилка"
        }
}

/* --- NetworkMonitor.kt --- */

class NetworkMonitor @Inject constructor(
    @ApplicationContext private val context: Context
) {

    private val _isOnline = MutableStateFlow(true)
    val isOnline: StateFlow<Boolean> = _isOnline

    private val connectivityManager =
        context.getSystemService(Context.CONNECTIVITY_SERVICE) as
ConnectivityManager

    fun start() {
        val request = NetworkRequest.Builder().build()
```

```
        connectivityManager.registerNetworkCallback(
            request,
            object : ConnectivityManager.NetworkCallback() {
                override fun onAvailable(network: Network) {
                    _isOnline.value = true
                }

                override fun onLost(network: Network) {
                    _isOnline.value = false
                }
            }
        )
    }
}

/* --- AppNavigator.kt --- */

sealed class AppDestination(val route: String) {
    object Dashboard : AppDestination("dashboard")
    object Profile : AppDestination("profile")
    object History : AppDestination("history")
    object Onboarding : AppDestination("onboarding")
    object Settings : AppDestination("settings")
}

/* --- DashboardWidgetProvider.kt (Home Screen Widget) --- */

class DashboardWidgetProvider : AppWidgetProvider() {

    override fun onUpdate(
```

```
            context: Context,
            appWidgetManager: AppWidgetManager,
            appWidgetIds: IntArray
        ) {
            for (widgetId in appWidgetIds) {
                val views = RemoteViews(context.packageName,
R.layout.widget_dashboard)

                // Тут можуть бути підставлені останні дані з локального сховища
                views.setTextViewText(R.id.textWidgetTitle, "Баланс дня")
                views.setTextViewText(R.id.textWidgetCalories, "Калорії: --")
                views.setTextViewText(R.id.textWidgetWater, "Вода: -- мл")

                appWidgetManager.updateAppWidget(widgetId, views)
            }
        }
    }

    /* --- QuickTileService для швидкого додавання води --- */

    @RequiresApi(Build.VERSION_CODES.N)
    class WaterQuickTileService : TileService() {

        override fun onClick() {
            super.onClick()
            // Імітація додавання 200 мл води
            Toast.makeText(this, "Додано 200 мл води",
Toast.LENGTH_SHORT).show()
        }
    }
```

```
/* --- SettingsFragment.kt (UI-настройки) --- */

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

    private val viewModel: SettingsViewModel by viewModels()

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        val switchReminders = view.findViewById<SwitchCompat>(R.id.switchReminders)
        val switchDarkMode = view.findViewById<SwitchCompat>(R.id.switchDarkMode)

        viewLifecycleOwner.lifecycleScope.launchWhenStarted {
            viewModel.settings.collect { settings ->
                switchReminders.isChecked = settings.remindersEnabled
                switchDarkMode.isChecked = settings.darkMode
            }
        }

        switchReminders.setOnCheckedChangeListener { _, isChecked ->
            viewModel.updateReminders(isChecked)
        }

        switchDarkMode.setOnCheckedChangeListener { _, isChecked ->
            viewModel.updateDarkMode(isChecked)
        }
    }
}
```

```
/* --- SettingsViewModel.kt --- */

@HiltViewModel
class SettingsViewModel @Inject constructor(
    private val settingsDataStore: SettingsDataStore
) : ViewModel() {

    val settings: StateFlow<UserSettings> =
        settingsDataStore.remindersEnabledFlow
            .combine(settingsDataStore.languageFlow) { reminders, lang ->
                UserSettings(
                    remindersEnabled = reminders,
                    darkMode = false,
                    preferredUnits = "metric",
                    languageCode = lang
                )
            }
            .stateIn(viewModelScope, SharingStarted.Lazily, UserSettings())

    fun updateReminders(enabled: Boolean) {
        viewModelScope.launch {
            settingsDataStore.setRemindersEnabled(enabled)
        }
    }

    fun updateDarkMode(enabled: Boolean) {
        ThemeHelper.applyTheme(enabled)
    }
}
```

ДОДАТОК Г

Ілюстративна частина

ІЛЮСТРАТИВНА ЧАСТИНА

РОЗРОБКА МЕТОДІВ І ПРОГРАМНИХ ЗАСОБІВ МОБІЛЬНОЇ СИСТЕМИ НА ПЛАТФОРМІ ANDROID ДЛЯ ПІДТРИМКИ ЗДОРОВОГО СПОСОБУ ХАРЧУВАННЯ

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення

Магістерська кваліфікаційна робота на тему:

РОЗРОБКА МЕТОДІВ І ПРОГРАМНИХ ЗАСОБІВ МОБІЛЬНОЇ СИСТЕМИ НА ПЛАТФОРМІ ANDROID ДЛЯ ПІДТРИМКИ ЗДОРОВОГО СПОСОБУ ХАРЧУВАННЯ

Автор: ст.групи 1ПІ-24м Бабійчук І.С.
Науковий керівник: к.т.н., доцент каф. ПЗ Войтко В.В.

Вінниця 2025

АКТУАЛЬНІСТЬ ТЕМИ

Зростання темпу життя та популяризація здорового способу харчування формують стійкий попит на цифрові інструменти, які допомагають користувачам контролювати щоденний раціон, баланс мікронутрієнтів, рівень спожитої води та досягати індивідуальних цілей – схуднення, підтримання ваги або її набору [29]. Ринок мобільних застосунків демонструє активний розвиток, однак значна частина наявних рішень або перевантажені складним функціоналом, або не здатні адаптуватися під специфічні потреби користувача, такі як гнучке налаштування параметрів (вік B, стать C, вага D, зріст E, рівень активності F, ціль G), врахування інгредієнтних обмежень та зручне додавання продуктів.

АКТУАЛЬНІСТЬ ТЕМИ

. У цьому контексті постає потреба у створенні оптимізованої мобільної системи, яка забезпечує простий, інтуїтивний та достовірний спосіб моніторингу харчування і водного балансу, не перевантажуючи користувача зайвими інтеракціями. Використання платформи Android дозволяє інтегрувати сучасні архітектурні підходи, бібліотеки Jetpack та інструменти Kotlin, що сприяє розробці ефективної та масштабованої системи.

Таким чином, тема магістерської роботи «Розробка методів і засобів мобільної системи на платформі Android для підтримки здорового способу харчування» є своєчасною та науково обґрунтованою.

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Метою магістерської роботи є підвищення можливостей підтримки здорового способу харчування шляхом розробки методів і засобів мобільної системи на платформі Android з оптимізованими алгоритмами збору, обробки та подання даних про харчування та водний баланс, що надає користувачам розвинутий функціонал системи для підтримки свого здоров'я.

Об'єктом дослідження є процес створення мобільних інформаційних систем підтримки здорового способу харчування.

Предметом дослідження є методи та засоби розробки Android-застосунку для персоналізованого контролю харчування та водного балансу.

ДЛЯ ДОСЯГНЕННЯ ПОСТАВЛЕНОЇ МЕТИ ПЕРЕДБАЧЕНО ВИКОНАННЯ ТАКИХ ЗАВДАНЬ

- :провести аналіз сучасних підходів до побудови мобільних систем харчового моніторингу;
- розробити модель обліку фізіологічних параметрів користувача (B-G) та алгоритми оцінки добових норм;
- створити модуль відстеження водного балансу;
- реалізувати механізм обліку харчових продуктів із можливістю додавання за допомогою сканування штрих-коду та пошуку за назвою;
- забезпечити облік обмежень користувача: інгредієнти, алергії, кількість прийомів їжі, допустиме відхилення калорійності ±5%;
- розробити інтерфейс мобільного застосунку, орієнтований на простоту та швидкість взаємодії;
- провести тестування функціональних модулів і оцінити точність, стабільність та зручність користування системою.

МЕТОДИ ДОСЛІДЖЕННЯ

- методи проєктування архітектури мобільних застосунків для побудови структурної моделі системи, визначення взаємодії між модулями та формування чітких рівнів абстракції;
- методи адаптивного визначення добової норми калорій для формування алгоритмів обчислення, нормалізації та валідації харчових даних;
- методи прогнозування водного балансу для методи прогнозування водного балансу для формування персоналізованих нагадувань та рекомендацій щодо гідратації на основі історичних даних про споживання води користувачем

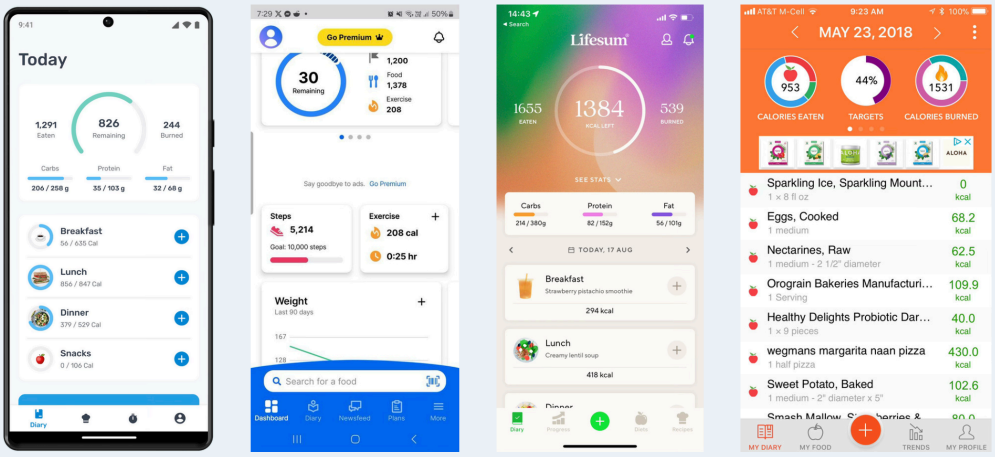
НАУКОВА НОВИЗНА

- Подальшого розвитку отримав метод формування персональних харчових рекомендацій, який, на відміну від відомих, базується на спрощеній адаптивній моделі з використанням системи параметрів B-G (вік, вага, зріст, рівень активності, ціль), що забезпечує підвищену точність розрахунку добової норми калорій при одночасному зменшенні обчислювальної складності і робить метод ефективним для мобільних застосунків із обмеженими ресурсами.
- Подальшого розвитку отримав метод контролю та прогнозування водного балансу, який, на відміну від існуючих, мінімізує кількість дій користувача за рахунок автоматизованих розрахунків на основі індивідуальних фізіологічних параметрів, що забезпечує точні персональні рекомендації щодо добового споживання води та підвищує зручність щоденного трекінгу.

ПРАКТИЧНЕ ЗНАЧЕННЯ ОТРИМАНИХ РЕЗУЛЬТАТІВ

Розроблена мобільна система може бути використана широкою аудиторією для контролю харчування, підтримки здорового способу життя, формування корисних звичок та відстеження індивідуальних метрик. Запропоновані методи та технічні рішення можуть бути основою для подальшого удосконалення й комерційного розвитку цифрових продуктів у сфері health-tech.

АНАЛОГИ

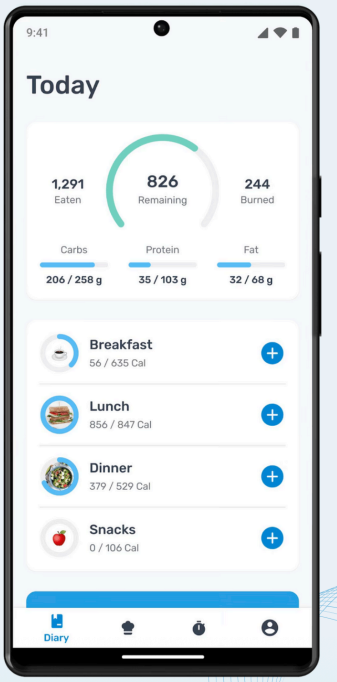


YAZIO MYFITNESSPAL LIFESUM CRONOMETER

АНАЛОГ YAZIO

Yazio позиціонується як застосунок для здорового харчування із фокусом на персоналізованих планах [3]. Він пропонує гнучкі програми схуднення, набору маси та підтримки форми. У застосунку реалізовано автоматизоване формування меню, відстеження макронутрієнтів, лічильник води та можливість логування власних страв.

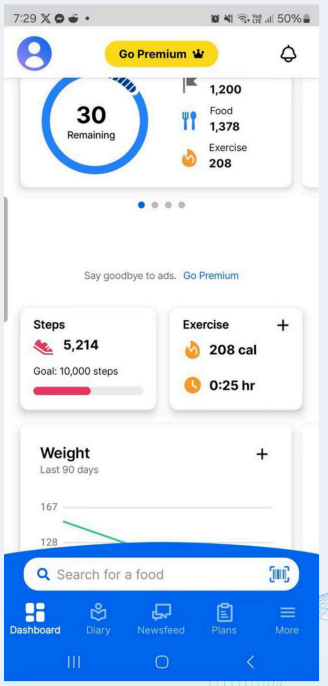
Серед обмежень - більшість персональних рекомендацій доступні лише у платній версії, а адаптивність планів базується на фіксованих параметрах, що не враховують зміну поведінкових чи фізіологічних характеристик користувача. Також застосунок не має гнучкої моделі користувачів з урахуванням індивідуальних обмежень та алергій.



АНАЛОГ MYFITNESSPAL

MyFitnessPal є одним із найпоширеніших застосунків для контролю калорій, що використовується мільйонами користувачів у всьому світі. Його ключовою перевагою є велика база продуктів, що включає як промислові товари, так і користувацькі записи. Функціонал застосунку дозволяє вести харчовий щоденник, додавати власні рецепти, сканувати штрих-коди продуктів, а також інтегрувати дані з фітнес-трекерів.

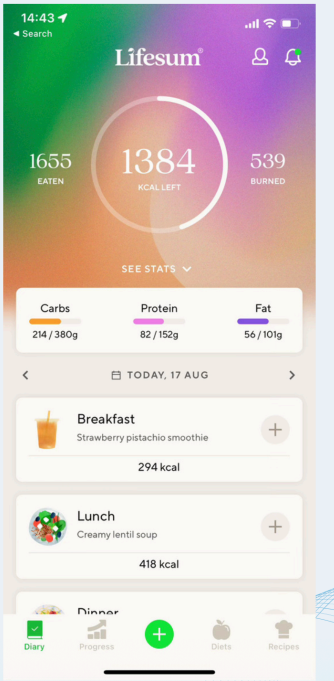
Водночас сильна орієнтація на підрахунок калорій призводить до ускладнення взаємодії: інтерфейс може бути перевантаженим, частина функцій доступна лише за підпискою, а рекомендовані показники спираються на статичні формули без адаптивної корекції.



АНАЛОГ LIFESUM

Lifesum орієнтується на збалансоване харчування та формування корисних звичок. Він забезпечує підбір дієт, трекінг прийомів їжі, аналіз макронутрієнтів, рекомендації щодо водного балансу та індикатори загального прогресу. Застосунок також пропонує підтримку баркод-сканера та має сучасний мінімалістичний інтерфейс.

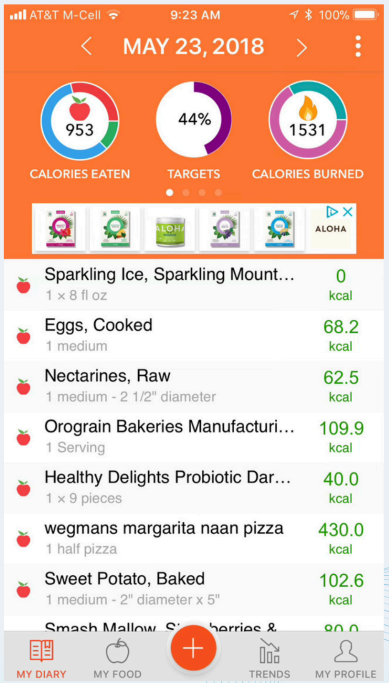
До недоліків можна віднести обмежений набір безкоштовних функцій, відсутність механізму автоматичного формування персонального раціону на основі поведінкових даних користувача, а також обмежену прозорість щодо джерел харчових даних.



АНАЛОГ CRONOMETER

Cronometer виділяється високою точністю та орієнтацією на детальний нутрієнт-аналіз. На відміну від більшості аналогів, застосунок надає розширені показники мікронутрієнтів, що робить його популярним серед спортсменів та користувачів з медичними потребами. Сканування продуктів, додавання страв, ведення щоденника та синхронізація з трекерами також доступні.

Основна слабка сторона - складність для пересічного користувача через перевантаження даними, а також фокус на нутрієнтах без зручних інструментів автоматичного планування харчування. Підтримка водного балансу реалізована базово.



ПОРІВНЯННЯ ХАРАКТЕРИСТИК АНАЛОГІВ

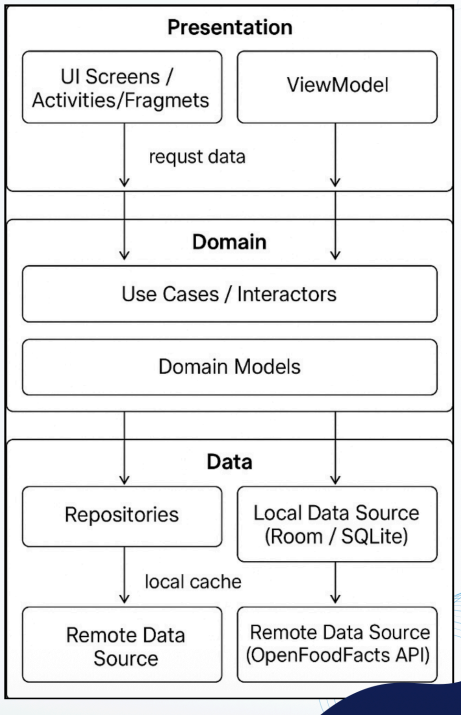
Назва програми	MyFitnessPal	Yazio	Lifesum	Cronometer	Власна розробка
Трекінг калорій	1	1	1	1	1
Трекінг водного балансу	0.5	1	1	0.5	1
Баркод–сканування	1	1	1	0	1
Відкриті джерела харчових даних	0	0	0	0.5	1
Адаптивний розрахунок добової норми	0	0.5	0	0	1
Автоматичне формування денного раціону	0	0.5	0.5	0	1
Облік алергій та обмежень	0	0.5	0.5	0	1
Зміна профільних параметрів користувача	1	1	1	1	1
Загальна оцінка	3.5	5.5	5	3	8

ВИКОРИСТАНІ ТЕХНОЛОГІЇ

РОЗРОБКА АРХІТЕКТУРИ МОБІЛЬНОЇ СИСТЕМИ

Архітектура системи базується на трирівневій моделі (Data, Domain, Presentation) з принципами модульності, розширюваності, слабкої зв'язаності та підтримки офлайн-режиму. Data-рівень обробляє локальні дані (Room), OpenFoodFacts API, ручно додані продукти та обчислювальні модулі через репозиторії. Domain-рівень містить бізнес-логіку, розрахунок добових норм, моделі водного балансу та use cases і повністю незалежний від Android. Presentation-рівень реалізований на MVVM з використанням Flow і відповідає за екрани, візуалізацію даних, віджети й керування станом. Ключові модулі охоплюють профіль користувача, харчові дані, добові норми, денний раціон, трекінг води та інтерфейс. Архітектура гарантує стабільну роботу з API, низьку затримку при обробці даних, легке масштабування й інтеграцію сучасних Android-технологій (Room, ViewModel, Hilt, Retrofit).

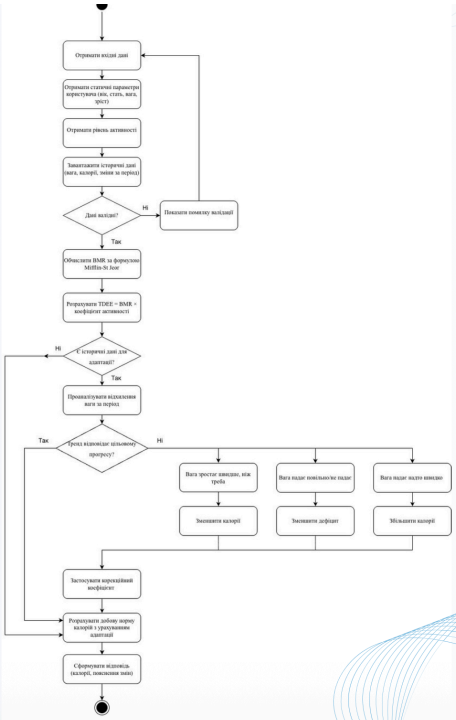
ДІАГРАМА ВЗАЄМОДІЇ МОДУЛІВ МОБІЛЬНОЇ СИСТЕМИ



РОЗРОБКА МЕТОДУ АДАПТИВНОГО ВИЗНАЧЕННЯ ДОБОВОЇ НОРМИ КАЛОРІЙ

Розрахунок добової норми калорій ґрунтується на адаптивному підході, що поєднує базовий розрахунок метаболізму за формулою Міффліна-Сан Жеора, враховує фізіологічні параметри користувача, рівень активності та персональні цілі. На основі BMR формується TDEE з використанням коефіцієнтів активності, після чого норма коригується відповідно до мети – дефіцит 10-20%, підтримання або надлишок 10-15%. Алгоритм включає адаптивні механізми: регулярне оновлення параметрів при зміні ваги, динамічну зміну активності, персоналізований вибір коригування та захист від фізіологічно некоректних значень. Кінцевий результат формується за моделлю BMR → TDEE → DailyCalories, де коригувальний коефіцієнт підбирається автоматично. Такий підхід забезпечує точне, персоналізоване й стійке до змін визначення добової калорійності на основі як поточних даних, так і історичної динаміки.

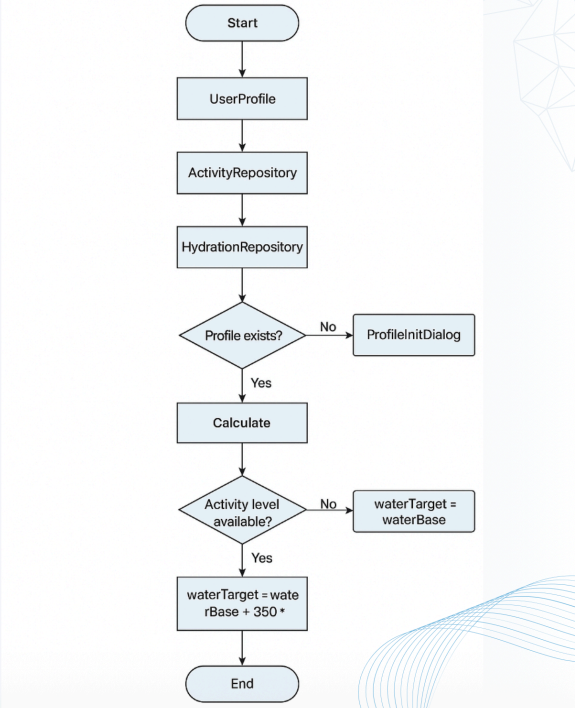
UML-ДІАГРАМА АЛГОРИТМУ ВИЗНАЧЕННЯ ДОБОВОЇ НОРМИ КАЛОРІЙ



РОЗРОБКА МЕТОДУ ПРОГНОЗУВАННЯ ВОДНОГО БАЛАНСУ

Запропонований метод адаптивного прогнозування водного та калорійного балансу поєднує фізіологічні параметри користувача, рівень активності та динаміку поведінки, автоматично оновлюючи добову норму без статичних формул. Модель використовує параметри B-G, що переналаштовуються з появою нових даних, а водний баланс розраховується за формулою 30 мл/кг із корекцією за активністю. На етапі ініціалізації формуються BMR і TDEE, після чого AdaptiveCalorieService спільно з модулями ваги, активності та харчового споживання динамічно коригує показники. Узгодженість забезпечує фасад UserProfileInteractor, а зберігання – UserPreferencesStorage та HealthDataRepository. Інтеграція зі сканером штрих-кодів і пошуковим модулем підвищує точність обліку їжі. У результаті користувач отримує персоналізовану та автономно скориговану норму калорій і води, адаптовану до змін ваги, активності та звичок.

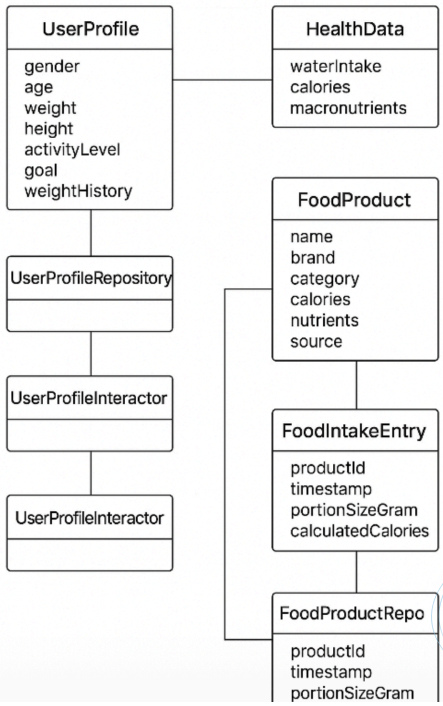
БЛОК-СХЕМА АЛГОРИТМУ РОБОТИ МЕТОДУ ПРОГНОЗУВАННЯ ВОДНОГО БАЛАНСУ



РОЗРОБКА МОДЕЛІ ДАНИХ КОРИСТУВАЧА ТА ПРОДУКТІВ ХАРЧУВАННЯ

Запропонована модель даних забезпечує масштабованість, точність нутрітивної інформації та підтримку адаптивних розрахунків. Профіль користувача представлений сутністю UserProfile із фізіологічними параметрами, активністю, цілями та історією ваги, тоді як динамічні дані винесені в HealthData і WeightRecord. Зберігання реалізоване через таблиці профілю та історії ваги, доступ до яких здійснюється через UserPreferencesStorage, WeightDao та UserProfileRepository. Модель продуктів базується на FoodProduct із розширеним набором нутрієнтів і підтримкою штрих-кодів, а споживання фіксується у FoodIntakeEntry в окремій таблиці, що запобігає дублюванню даних. Інтеграція з OpenFoodFacts виконується через OpenFoodFactsApiService та FoodProductMapper, пошук – через FoodSearchEngine, а швидке додавання – через FoodBarcodeScanner. Розрахункові сервіси працюють із валідними узгодженими даними, що гарантує консистентність і захист бізнес-логіки. У результаті модель формує надійну основу для адаптивних алгоритмів та подальшого інтелектуального аналізу харчової поведінки.

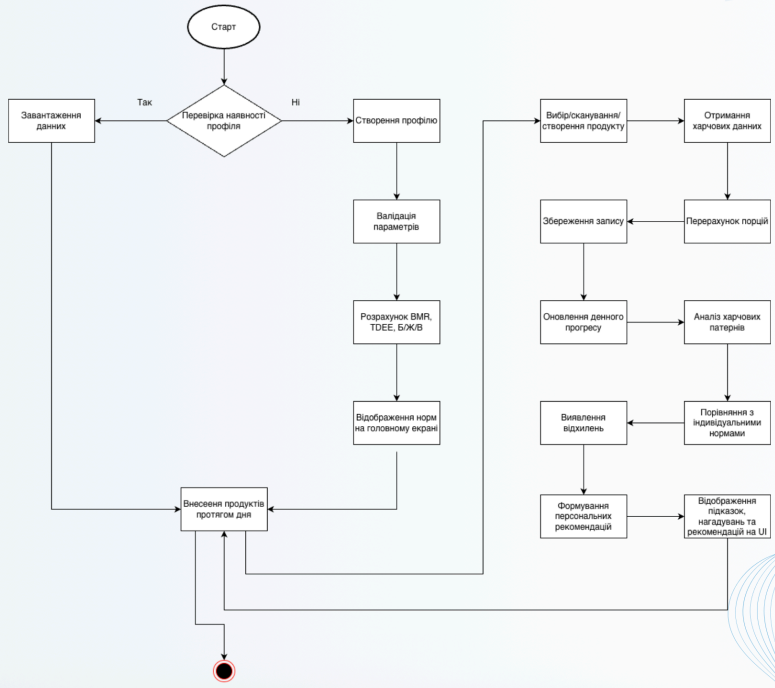
СТРУКТУРА КЛАСІВ МОДЕЛІ ДАНИХ КОРИСТУВАЧА ТА ПРОДУКТІВ ХАРЧУВАННЯ



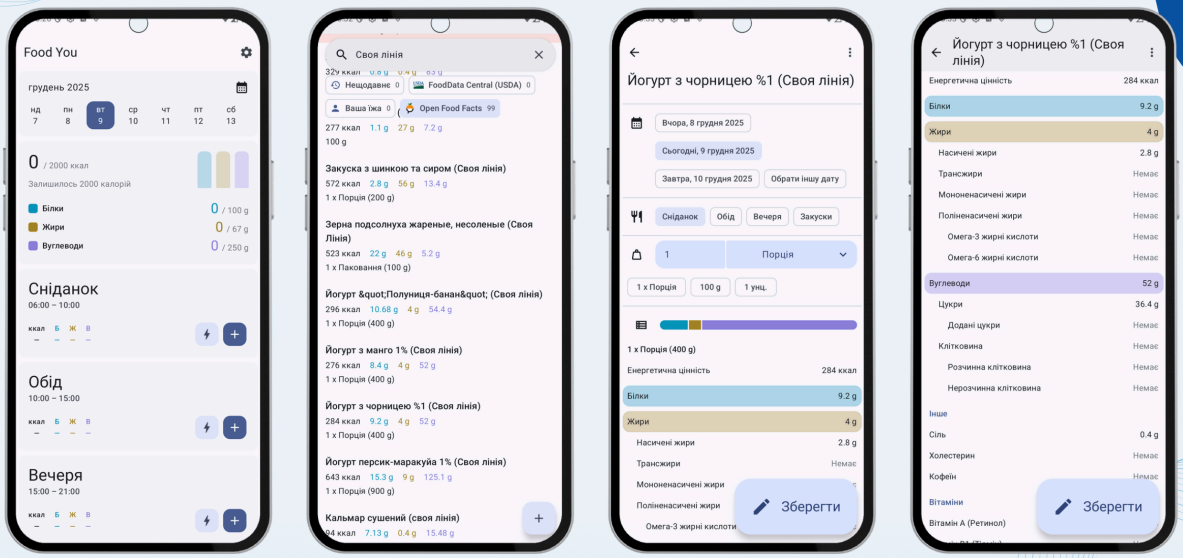
РОЗРОБКА ЗАГАЛЬНОГО АЛГОРИТМУ РОБОТИ СИСТЕМИ

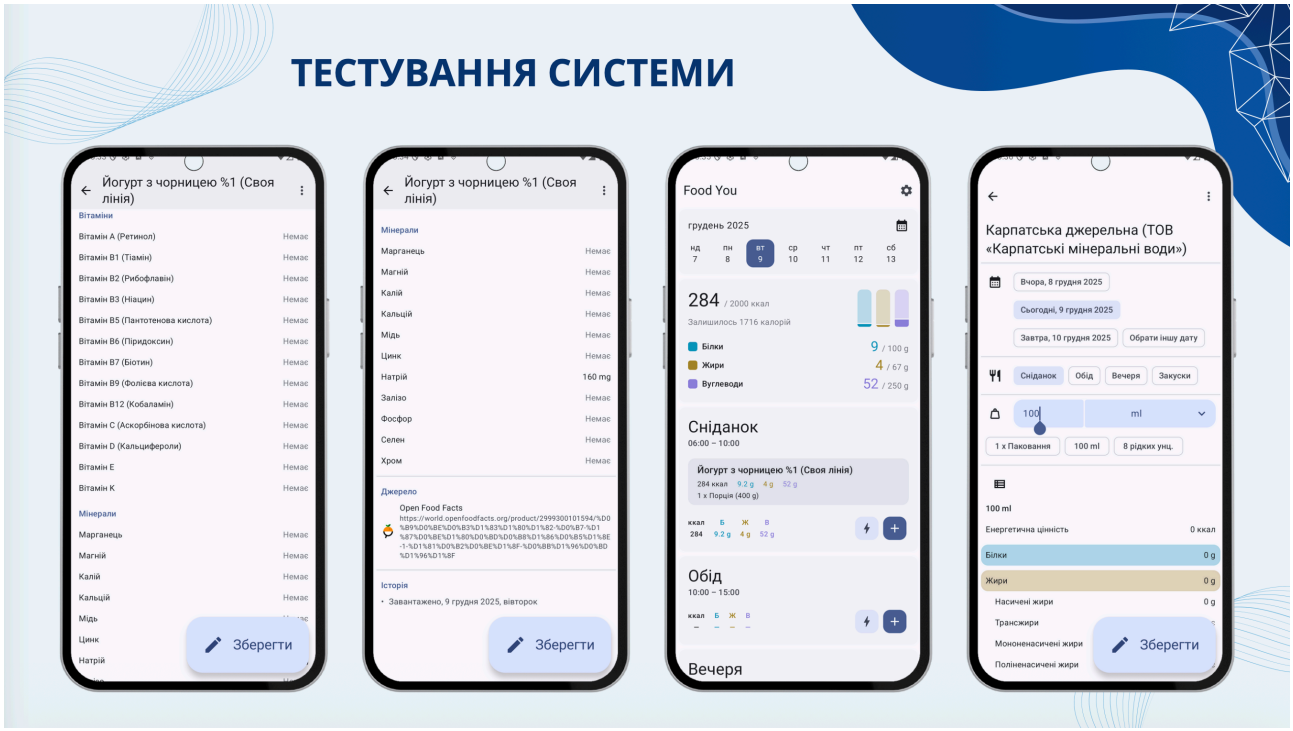
Функціонування мобільної системи підтримки здорового харчування побудоване як безперервний цикл, у якому кожна дія користувача одразу впливає на оновлення добових норм і рекомендацій. Робота системи охоплює п'ять основних етапів: створення та валідацію профілю з первинним розрахунком BMR і TDEE; додавання продуктів через каталог, сканування або ручне внесення; автоматичний перерахунок калорійності й оновлення денного прогресу; моніторинг калорій, макросів і води в реальному часі; адаптивне коригування норм відповідно до змін ваги та активності. Підсумковим кроком є формування персональних рекомендацій на основі поведінкових даних і режиму харчування. Усі взаємодії проходять через доменні сервіси, оновлюють репозиторії та синхронізуються з інтерфейсом у реальному часі, забезпечуючи повністю автоматизований цикл роботи системи.

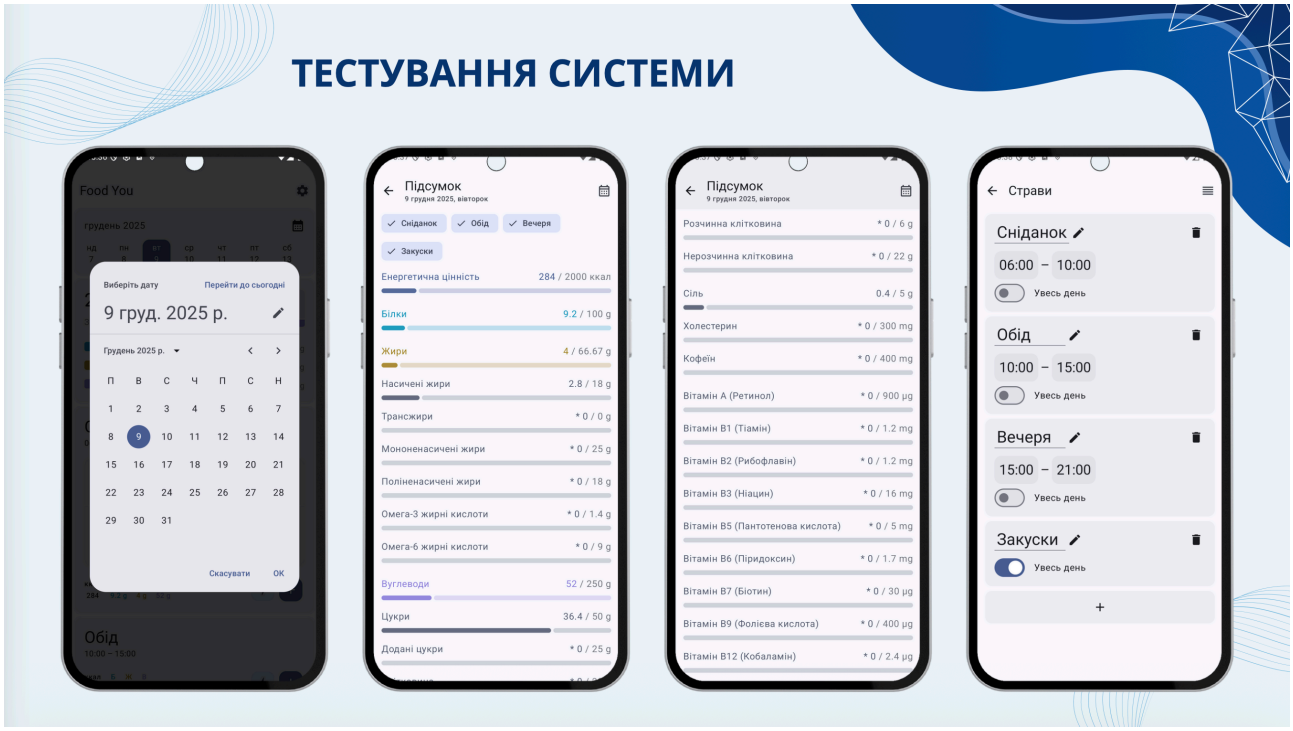
БЛОК-СХЕМА РОБОТИ ЗАГАЛЬНОГО АЛГОРИТМУ СИСТЕМИ



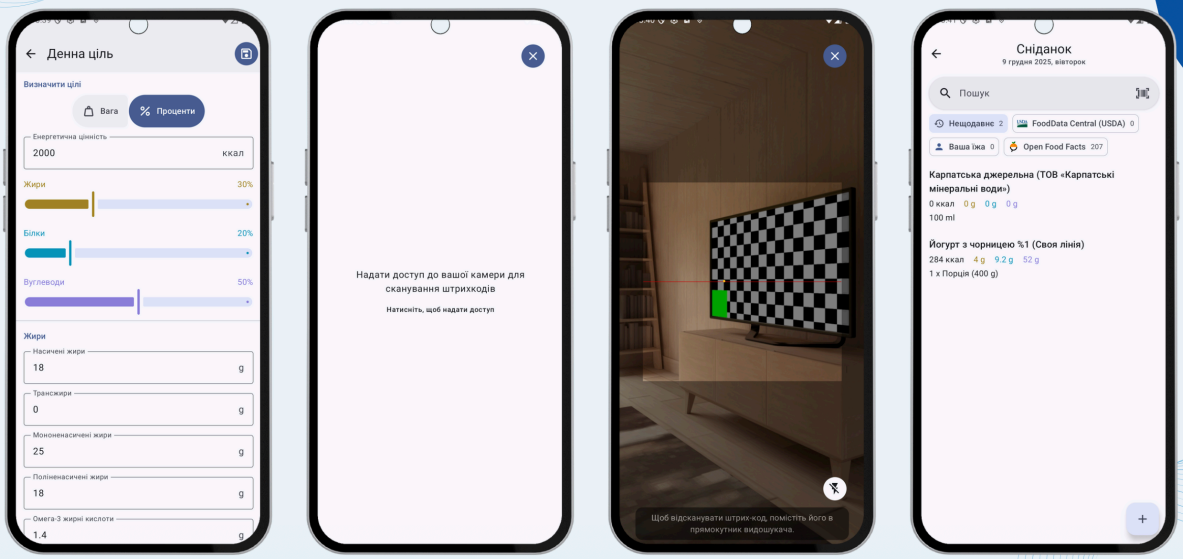
ТЕСТУВАННЯ СИСТЕМИ




ТЕСТУВАННЯ СИСТЕМИ
Йогурт з чорницею %1 (Своя лінія)
Вітаміни
Вітамін A (Ретинол)
Вітамін B1 (Тіамін)
Вітамін B2 (Рибофлавін)
Вітамін B3 (Ніацин)
Вітамін B5 (Пантотенова кислота)
Вітамін B6 (Піридоксин)
Вітамін B7 (Біотин)
Вітамін B9 (Фолієва кислота)
Вітамін B12 (Кобаламін)
Вітамін C (Аскорбінова кислота)
Вітамін D (Кальцифероли)
Вітамін E
Вітамін K
Мінерали
Марганець
Магній
Калій
Кальцій
Мідь
Цинк
Натрій
Зберегти
Йогурт з чорницею %1 (Своя лінія)
Мінерали
Марганець
Магній
Калій
Кальцій
Мідь
Цинк
Натрій 160 mg
Заліза
Фосфор
Селен
Хром
Джерело
Open Food Facts
Історія
Завантажено, 9 грудня 2025, вівторок
Зберегти
Food You
грудень 2025
284 / 2000 ккал
Залишилось 1716 калорій
Білки 9 / 100 g
Жири 4 / 67 g
Вуглеводи 52 / 250 g
Сніданок
06:00 – 10:00
Йогурт з чорницею %1 (Своя лінія)
Обід
10:00 – 15:00
Вечеря
Карпатська джерельна (ТОВ «Карпатські мінеральні води»)
Вчора, 8 грудня 2025
Сьогодні, 9 грудня 2025
Завтра, 10 грудня 2025
Обрати іншу дату
Сніданок
Обід
Вечеря
Закуски
100
ml
1 x Пакування
100 ml
8 рідких унц
100 ml
Енергетична цінність 0 ккал
Білки 0 g
Жири 0 g
Насичені жири 0 g
Трансжири
Мононенасичені жири
Поліненасичені жири
Зберегти


ТЕСТУВАННЯ СИСТЕМИ
Food You
грудень 2025
Виберіть дату
Перейти до сьогодні
9 груд. 2025 р.
Скасувати
OK
Обід
Підсумок
9 грудня 2025, вівторок
Сніданок
Обід
Вечеря
Закуски
Енергетична цінність 284 / 2000 ккал
Білки 9.2 / 100 g
Жири 4 / 66.67 g
Насичені жири 2.8 / 18 g
Трансжири * 0 / 0 g
Мононенасичені жири * 0 / 25 g
Поліненасичені жири * 0 / 18 g
Омега-3 жирні кислоти * 0 / 1.4 g
Омега-6 жирні кислоти * 0 / 9 g
Вуглеводи 52 / 250 g
Цукри 36.4 / 50 g
Додані цукри * 0 / 25 g
Підсумок
9 грудня 2025, вівторок
Розчинна клітковина * 0 / 6 g
Нерозчинна клітковина * 0 / 22 g
Сіль 0.4 / 5 g
Холестерин * 0 / 300 mg
Кофеїн * 0 / 400 mg
Вітамін A (Ретинол) * 0 / 900 µg
Вітамін B1 (Тіамін) * 0 / 1.2 mg
Вітамін B2 (Рибофлавін) * 0 / 1.2 mg
Вітамін B3 (Ніацин) * 0 / 16 mg
Вітамін B5 (Пантотенова кислота) * 0 / 5 mg
Вітамін B6 (Піридоксин) * 0 / 1.7 mg
Вітамін B7 (Біотин) * 0 / 30 µg
Вітамін B9 (Фолієва кислота) * 0 / 400 µg
Вітамін B12 (Кобаламін) * 0 / 2.4 µg
Страви
Сніданок
06:00 – 10:00
Увесь день
Обід
10:00 – 15:00
Увесь день
Вечеря
15:00 – 21:00
Увесь день
Закуски
Увесь день

ТЕСТУВАННЯ СИСТЕМИ



ВИСНОВКИ

У результаті виконання магістерської кваліфікаційної роботи було:

- проведено аналіз сучасних підходів до побудови мобільних систем харчового моніторингу;
- розроблено модель обліку фізіологічних параметрів користувача (B-G) та алгоритми оцінки добових норм;
- створено модуль відстеження водного балансу;
- реалізовано механізм обліку харчових продуктів із можливістю додавання за допомогою сканування штрих-коду та пошуку за назвою;
- забезпечено облік обмежень користувача: інгредієнти, алергії, кількість прийомів їжі, допустиме відхилення калорійності ±5%;
- розроблено інтерфейс мобільного застосунку, орієнтований на простоту та швидкість взаємодії;
- проведено тестування функціональних модулів і оцінено точність, стабільність та зручність користування системою.

ВИСНОВКИ

Підсумовуючи виконану роботу, можна стверджувати, що поставлена мета - розробка методів і засобів мобільної системи на Android для підтримки здорового способу харчування - досягнута повністю. Розроблена система забезпечує персоналізований моніторинг калорійності та водного балансу, підтримує автоматизовану обробку харчових даних, адаптивні алгоритми, зручний інтерфейс та інтеграцію з відкритими джерелами інформації. Отримані результати мають як наукову новизну, так і практичну цінність, дозволяючи використовувати систему в реальних умовах та розширювати її функціональність у майбутніх розробках.

АПРОБАЦІЯ МАТЕРІАЛІВ

Основні положення, отримані в межах магістерської кваліфікаційної роботи, були представлені на V Всеукраїнській науково-технічній конференції молодих вчених, аспірантів і студентів «Комп'ютерні ігри та мультимедіа як інноваційний підхід до комунікації», де було обговорено особливості застосування методів гейміфікації для підвищення залученості користувачів мобільних систем здорового способу життя.

ПУБЛІКАЦІЇ

Результати дослідження опубліковано у тезах доповіді на V Всеукраїнській науково-технічній конференції молодих вчених, аспірантів і студентів «Комп'ютерні ігри та мультимедіа як інноваційний підхід до комунікації» під заголовком: «Використання засобів гейміфікації при створенні мобільних систем спеціалізованого призначення»

ДЯКУЮ ЗА УВАГУ