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

Допускається до захисту
Завідувач кафедри
_____Олена ОЛЬХОВСЬКА
(підпис)
«_____»___202_ р.

КВАЛІФІКАЦІЙНА РОБОТА
на тему

РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ МОБІЛЬНОГО ЗАСТОСУНКУ ДЛЯ ПІДТРИМКИ ЗДОРОВОГО ХАРЧУВАННЯ З БАЗОЮ РЕЦЕПТІВ І ПІДРАХУНКОМ КАЛОРІЙ

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

Виконавець роботи Вакуленко Станіслав Русланович
_____«____» _____202_ р.
(підпис)

Науковий керівник доцент, к.п.н. Кошова О. П.
_____«____»_____202_ р.
(підпис)

Рецензент

ПОЛТАВА 2025

РЕФЕРАТ

Записка: 75 с., 13 рис., 2 таблиці, 1 додаток, 13 джерел.

МОБІЛЬНИЙ ЗАСТОСУНОК, REACT NATIVE, КАЛОРІЇ, РЕЦЕПТИ, OFFLINE, ASYNCSTORAGE, UX

Об’єктом розробки є мобільний застосунок для підтримки здорового харчування з базою продуктів і рецептів та автоматизованим підрахунком енергії й макронутрієнтів.

Предметом розробки є програмна реалізація системи, що забезпечує ведення щоденника харчування, облік води та ваги, розрахунок добових норм (за формулою Міффліна–Сан Жеора), відображення аналітики та роботу в офлайн-режимі з локальним зберіганням даних.

Метою роботи є створення кросплатформного застосунку, який спрощує контроль раціону та здорових звичок завдяки зручному інтерфейсу, попередньо заповненим базам продуктів/рецептів, автоматичним обчисленням КБЖВ і наочній статистиці.

Результатом роботи стало розроблення Healthy Food Manager на базі React Native (Expo) і TypeScript із локальною БД на AsyncStorage (формат JSON). Реалізовано ключові розділи:

• Щоденник – додавання продуктів, рецептів і власних записів у прийоми їжі з автоматичним підрахунком калорій і макросів, календар дат.

• Рецепти – колекція страв із калорійністю на порцію, інгредієнтами, інструкціями, фільтрами та обраним.

• Вода – трекінг споживання води з пресетами об’ємів, історією за день і прогрес-баром норми.

• Вага – запис, історія та графік динаміки, порівняння з цільовою вагою.

• Статистика – зведення за тиждень (калорії, вода, макроси), індикатори виконання цілей.

• Профіль – введення антропометрії, розрахунок цільових калорій/КБЖВ, перемикач теми.

Особливості: повністю офлайн-робота без облікового запису; розширювана база продуктів і рецептів; уніфіковані обчислення (Міффлін–Сан Жеор, 30/30/40 для білків/жирів/вуглеводів), інтуїтивний UX, світла/темна тема.

Проведено тестування якості: юніт-тести утиліт обчислень (коректність BMR/TDEE та агрегацій КБЖВ), інтеграційні сценарії додавання/видалення записів і відновлення стану, ручні E2E-перевірки на Android і Web. За підсумками тестування підтверджено стабільність офлайн-роботи, коректність підрахунків і узгодженість інтерфейсу.

Застосунок може бути використаний як персональний трекер харчування, навчальний приклад для курсів з мобільної розробки на React Native та основа для подальшого розширення (синхронізація, мікронутрієнти, інтеграції з носимими пристроями, сканування штрих-кодів).

ЗМІСТ

СПИСОК УМОВНИХ ПОЗНАЧЕННЬ, СИМВОЛІВ, СКОРОЧЕНЬ, ТЕРМІНІВ

Умовні позначення, символи, скорочення, терміни	Пояснення умовних позначень, скорочень, символів
КБЖВ	Комплект макронутрієнтів: калорії, білки, жири, вуглеводи
BMR	Basal Metabolic Rate - базальний обмін речовин (енерговитрати у стані спокою)
TDEE	Total Daily Energy Expenditure - загальні добові енерговитрати з урахуванням активності
BMI	Body Mass Index - індекс маси тіла (кг/м²)
MSJ	Mifflin–St Jeor - формула розрахунку BMR, використовується в застосунку
AMDR	Acceptable Macronutrient Distribution Range - прийнятні діапазони розподілу макронутрієнтів
GI	Glycemic Index - глікемічний індекс, відносна швидкість підвищення глюкози в крові
GL	Glycemic Load - глікемічне навантаження (враховує GI та кількість вуглеводів у порції)
UI	User Interface - інтерфейс користувача
UX	User Experience - користувацький досвід
API	Application Programming Interface - програмний інтерфейс для взаємодії між компонентами
JSON	JavaScript Object Notation - текстовий формат обміну даними, що використовується для зберігання та передачі структурованої інформації

ВСТУП

Стрімкий розвиток мобільних технологій та зростання уваги суспільства до здорового способу життя формують сталий попит на цифрові інструменти, що допомагають керувати харчуванням, водним балансом та масою тіла. Водночас більшість існуючих рішень або вимагають постійного інтернет-з'єднання, або збирають персональні дані користувачів, або є перевантаженими нефункціональними можливостями, що ускладнює щоденне застосування. Це актуалізує потребу у простому, офлайн-орієнтованому, прозорому щодо приватності інструменті, який забезпечує базові розрахунки енергетичних потреб і макронутрієнтів, містить готову базу продуктів та рецептів і дає наочну аналітику прогресу.

Мобільний застосунок Healthy Food Manager розроблено з метою підтримки користувача у прийнятті щоденних рішень щодо харчування: ведення щоденника прийомів їжі, автоматичний підрахунок калорій та КБЖВ, облік води й ваги, а також робота з базою продуктів і рецептів. Технічно застосунок реалізовано як кросплатформене рішення на React Native (Expo, TypeScript) з локальним зберіганням даних у AsyncStorage, що забезпечує офлайн-роботу, швидкий старт та прозору архітектуру без серверної складової.

Поєднання науково обґрунтованих розрахунків енергетичних потреб (BMR/TDEE) з простими для користувача інтерфейсами дозволяє підвищити дисципліну харчування, зменшити ризики перевищення калорійності та спростити дотримання цілей (схуднення/підтримання/набір). Офлайн-режим і локальне зберігання відповідають вимогам приватності, а кросплатформеність - доступності для широкої аудиторії.

Мета роботи - спроєктувати та реалізувати мобільний застосунок для підтримки здорового харчування з базою рецептів і підрахунком калорій, що працює офлайн, забезпечує персоналізовані розрахунки та наочну аналітику.

Для досягнення мети необхідно розв'язати такі завдання:

1. Проаналізувати предметну область, методи оцінювання енергетичних потреб та розподілу макронутрієнтів; визначити вимоги до системи.

2. Провести інформаційний огляд аналогів і обґрунтувати вибір технологічного стеку.
3. Спроєктувати архітектуру застосунку (екрани, контексти, моделі даних, потоки) та схеми зберігання.
4. Реалізувати функціонал: щоденник харчування, база продуктів і рецептів, підрахунок КБЖВ, трекери води та ваги, статистика, профіль користувача (BMR/TDEE, цілі, теми).
5. Забезпечити офлайн-роботу, базові механізми резервного копіювання (експорт/імпорт даних), доступність інтерфейсу (A11y) та приватність даних.
6. Провести тестування якості (unit, інтеграційне, e2e) та оцінити показники продуктивності інтерфейсу.
7. Підготувати інструкцію користувача і технічну документацію.

Об’єкт дослідження - процес підтримки здорового харчування засобами мобільних програмних систем.

Предмет дослідження - методи та засоби проєктування і реалізації кросплатформеного офлайн-застосунку з підрахунком калорій/макросів, локальним зберіганням та аналітикою.

У роботі застосовано: аналітичний огляд джерел і програмних аналогів; методи програмної інженерії для специфікації вимог та проєктування архітектури; алгоритмічні розрахунки BMR/TDEE (формула Міффліна-Сан Жеора) і розподілу КБЖВ; експериментальна перевірка працездатності; модульне, інтеграційне та наскрізне тестування; емпірична оцінка UX-показників (час відгуку, плавність прокрутки, стабільність).

Новизна полягає у поєднанні офлайн-підходу з персоналізованими розрахунками та локальною аналітикою без серверної інфраструктури, а також у проєктних рішеннях щодо структурування даних (продукти/рецепти/щоденник) та роботи з датами/таймзонами для коректних денних підсумків. Практичне значення - створений застосунок може використовуватись як готовий інструмент кінцевими користувачами, а також як навчальний приклад для

дисциплін з мобільної розробки, проєктування інтерфейсів і зберігання даних на клієнті.

Диплом складається з вступу, чотирьох розділів (постановка задачі; інформаційний огляд; теоретична частина; практична частина), висновків, списку інформаційних джерел та додатків. У роботі наведено формулювання вимог, архітектуру та моделі даних, ключові алгоритми, результати реалізації, тестування якості і інструкцію для користувача.

1. ПОСТАНОВКА ЗАДАЧІ

Метою роботи є проєктування та реалізація кросплатформеного мобільного застосунку Healthy Food Manager для підтримки здорового харчування з автоматичним підрахунком калорій та макронутрієнтів, офлайн-зберіганням даних і наочною аналітикою.

Обсяг функцій, що підлягають реалізації (scope):

- Щоденник харчування: 4 прийоми їжі/добу (сніданок, обід, вечеря, перекус) з прив’язкою до локальної дати; додавання позицій з бази продуктів, з бази рецептів або ручним введенням; миттєвий перерахунок калорій, білків, жирів, вуглеводів (КБЖВ) по прийому та дню; редагування/видалення записів.
- База продуктів і рецептів: попередньо заповнені довідники з категоріями, пошуком і типовими порціями; харчова цінність «на 100 г» і «на порцію»; додавання рецепту до щоденника одним натисканням.
- Профіль користувача: розрахунок BMR/TDEE (формула Міффліна-Сан Жеора) з урахуванням статі, віку, зросту, ваги та рівня активності; автоматичні цільові КБЖВ; вибір цілі (зниження/підтримання/набір); перемикач теми (світла/темна).
- Трекери: вода (норма, швидкі додавання 100/200/250/500 мл, прогрес-бар, історія за день), вага (записи з датою та нотаткою, графік динаміки, різниця до цілі).
- Статистика: тижневі/місячні зведення калорій і води, розподіл макро, виділення поточного дня, лінія цілі; швидкі показники (середні за тиждень).
- Зберігання та резервування: локальність, автоматична персистентність, експорт/імпорт JSON (backup/restore).

Якісні вимоги (NFR):

- Продуктивність: старт екрана ≤ 1.5 с; додавання позиції ≤ 300 мс на середньому пристрої; плавна прокрутка списків (~60 FPS) завдяки FlatList/SectionList.
- Надійність: цілісність даних після перезапуску; коректний перерахунок підсумків.

- Доступність (A11y): контраст, динамічний шрифт, accessibilityLabel, фокус-навігація; придатність до локалізації (i18n).
- Приватність/безпека: дані зберігаються локально, без аналітики/реєстрації; медичний дисклеймер («не є медичною порадою»).
- Коректність часу: денні підсумки рахуються за локальною датою користувача (Europe/Kyiv) без зсувів UTC.
- Тестованість: наявність модульних тестів для обчислень (BMR/TDEE, КБЖВ), інтеграційних для сховища та наскрізних (e2e) для ключових флоу.

Критерії приймання (Definition of Done) через ключові сценарії:

- Додавання продукту або рецепту (на 1 порцію) в обраний прийом і дату миттєво оновлює підсумки КБЖВ і зберігається офлайн; підтримується редагування та видалення.
- Ручне введення позиції з власними КБЖВ потрапляє у підсумки дня.
- Трекер води додає об’єм одним тапом, показує прогрес до норми та історію за день.
- Трекер ваги фіксує запис із датою/нотаткою і відображає точку на графіку з різницею до цільової ваги.
- Профіль рахує цільові калорії/КБЖВ від параметрів користувача й обраної цілі; тема перемикається й запам’ятовується.
- Пошук/фільтри в продуктах і рецептах повертають релевантні результати з можливістю додати позицію за один крок.
- Експорт/імпорт JSON працює без пошкодження даних (із режимом заміни або злиття).
- Пройдені тести: unit (калькулятори/агрегати), інтеграційні (CRUD у сховищі), e2e (додавання їжі, вода, вага, профіль, експорт/імпорт).
- Підготовлено інструкцію користувача (встановлення, навігація, типові дії).

Застосунок створюється на React Native (Expo, TypeScript) з локальною БД на основі AsyncStorage (експорт/імпорт JSON), працює без серверної складової, забезпечує приватність і доступність інтерфейсу.

2. ІНФОРМАЦІЙНИЙ ОГЛЯД

2.1. Основи підрахунку енерговитрат і макронутрієнтів

Підтримання або зміна маси тіла визначається енергетичним балансом: якщо надходження калорій з їжею перевищує добові витрати, формується профіцит і маса зростає; якщо надходження менше за витрати - утворюється дефіцит і маса зменшується. Тому для персоналізації харчування потрібні два кроки: коректно оцінити індивідуальні енерговитрати та зафіксувати цільовий розподіл макронутрієнтів.

Базову швидкість обміну (BMR) доцільно обчислювати за формулою Міффліна-Сан Жеора. Для чоловіків: BMR = 10 × маса(кг) + 6.25 × зріст(см) − 5 × вік(роки) + 5; для жінок: BMR = 10 × маса(кг) + 6.25 × зріст(см) − 5 × вік(роки) − 161. Це мінімальна енергія, потрібна організму у стані спокою. Щоб отримати реальні добові витрати, BMR множать на коефіцієнт фізичної активності (PAL): сидячий спосіб життя близько 1.20; легка активність 1.375; помірна 1.55; висока 1.725; дуже висока 1.90. Таким чином, TDEE = BMR × PAL.

Далі задається ціль. Для підтримання ваги орієнтуються на TDEE. Для зниження маси обирають помірний дефіцит приблизно 10–20% від TDEE (що відповідає безпечній швидкості змін близько 0.5–1% маси на тиждень). Для набору - помірний профіцит близько 5–15%, що дозволяє мінімізувати приріст жирової тканини.

Розподіл макронутрієнтів базується на калорійній щільності: 4 ккал/г для білків, 4 ккал/г для вуглеводів і 9 ккал/г для жирів. Практичний підхід - задавати цілі у грамах на кілограм маси: білок 1.6–2.2 г/кг (нижній край доречний при підтримці, верхній - під час дефіциту або інтенсивних тренувань), жир щонайменше 0.8 г/кг (часто 0.8–1.0 г/кг) для гормонального і метаболічного балансу, а вуглеводи - це калорії, що залишаються після віднімання частки білка і жиру. Якщо такий спосіб незручний, можна використовувати відсоткові схеми на кшталт 30/30/40 або 30/25/45 (білки/жири/вуглеводи) з подальшим авто-перерахунком у грами.

Окремо враховуються клітковина та вода. Доцільно орієнтуватися на 25–35 г

клітковини на добу з овочів, фруктів, бобових і цільних злаків. Добова потреба у воді зазвичай становить приблизно 30–35 мл на кілограм маси (мінімум 1.5–2.0 л), а під час активних тренувань додається близько 500 мл на кожну годину навантаження. У застосунку базова норма води встановлюється на рівні 2000 мл з можливістю персоналізації.

Наведемо приклад. Чоловік 28 років, 80 кг, 180 см, помірна активність (PAL = 1.55), ціль - зниження маси з дефіцитом 10%. Спершу обчислюємо BMR: $10\times80 + 6.25\times180 - 5\times28 + 5 = 800 + 1125 - 140 + 5 = 1790$ ккал. Далі $TDEE = 1790 \times 1.55 \approx 2775$ ккал. Цільова калорійність при дефіциті 10% - близько 2498 ккал. Якщо взяти білок 1.8 г/кг, отримаємо 144 г (це 576 ккал); жир 0.9 г/кг - 72 г (648 ккал). На вуглеводи залишається $2498 - (576 + 648) = 1274$ ккал, тобто приблизно 319 г. Підсумок для такого профілю: близько 2500 ккал на добу, білки 144 г, жири 72 г, вуглеводи 319 г, вода орієнтовно 2.4–2.8 л.

Для реалізації в застосунку важливо рахувати денні підсумки за локальною датою користувача (Europe/Kyiv), щоб нічні зсуви часу не спотворювали статистику. Округлення краще виконувати наприкінці розрахунку: калорії до 5–10 ккал, макроси до 1–5 г - це робить інтерфейс зрозумілішим без помітної втрати точності. Користувачеві варто дозволити вибір способу цілей - у грамах на кілограм або у відсотках від калорій - з автоматичним перерахунком. Усі розрахунки мають інформаційний характер і не замінюють медичної консультації; для підлітків, вагітних або користувачів із хронічними станами потрібні індивідуальні корекції.

2.2. Огляд аналогів і порівняльний аналіз

Серед найвідоміших застосунків для контролю харчування виділяються MyFitnessPal, Lifesum, YAZIO, Cronometer, FatSecret та Lose It!. Вони пропонують великі довідники продуктів, підрахунок калорій і макроелементів, рецепти, іноді сканування штрих-кодів і синхронізацію з носимими пристроями. Водночас значна частина можливостей прив’язана до хмарної інфраструктури, акаунтів і підписок, через що зростає поріг входу.

MyFitnessPal робить ставку на максимальне покриття бази продуктів і соціальні функції (див. рис. 2.1). Є швидке додавання через сканер штрих-кодів, імпорт рецептів, шаблони прийомів їжі, історія та синхронізація між платформами. Корисні «цілі по макросах» і щоденні нагадування, але частина зручностей, як гнучкі цілі за прийомами чи розширена аналітика, зазвичай у преміум. Потрібні акаунт і стабільний інтернет; офлайн-робота обмежена, а якість локальних (українських) позицій у базі може бути нерівномірною.

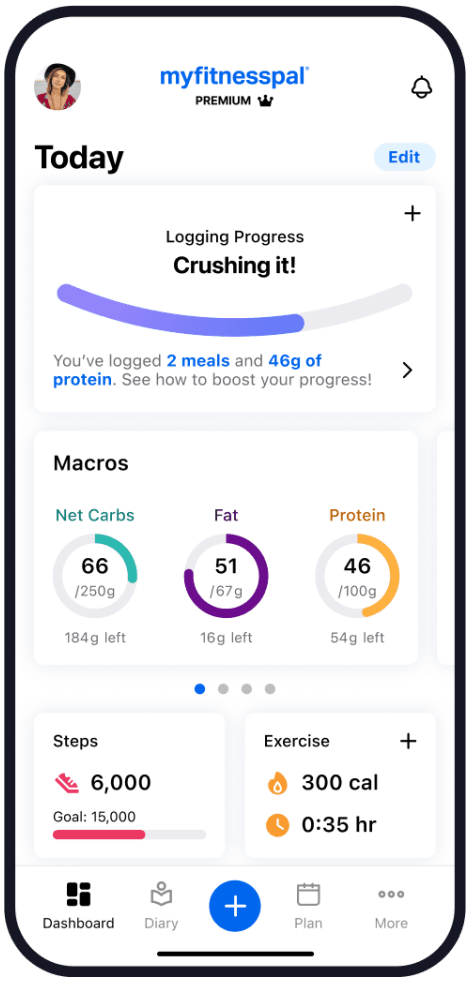

Рисунок 2.1 – Дизайн «MyFitnessPal»

Lifesum фокусується на керованих планах харчування та візуальних підказках поведінки (див. рис. 2.2). Є стартове опитування, персональні плани (кето, low-carb тощо), «щоденний індекс» якості раціону, трекер води й ваги. UX дуже дружній і «веде за руку», але гнучкість вільного щоденника інколи менша: частина функцій і планів відкривається лише після підписки, а без інтернету частина підказок і

довідника працює гірше.

Рисунок 2.2 – Дизайн «Lifesum»

YAZIO поєднує сучасний дизайн, рецептурну базу та планувальники (див. рис. 2.3). Зручно додавати страви, є інтервальне голодування, підбір меню й красиві візуалізації прогресу. Водночас найцінніші модулі (детальні макроси по днях/прийомах, плани, частина рецептів) часто у преміум, а швидкість і повнота роботи сильно залежать від наявності інтернету; без нього доступні не всі дані.

Рисунок 2.3 – Дизайн «YAZIO»

Cronometer відрізняється глибиною нутрієнтів і точністю: окрім КБЖВ, він відстежує мікроелементи, вітаміни, амінокислоти, спираючись на перевірені бази (див. рис. 2.4). Це підходить тим, хто має медичні або спортивні цілі, де потрібна деталізація й контроль мікрораціону. Зворотний бік - вища складність інтерфейсу для новачків і більше дій, щоб «просто записати калорії»; локальних брендів у базі зазвичай менше, а повна зручність досягається онлайн-синхронізацією.

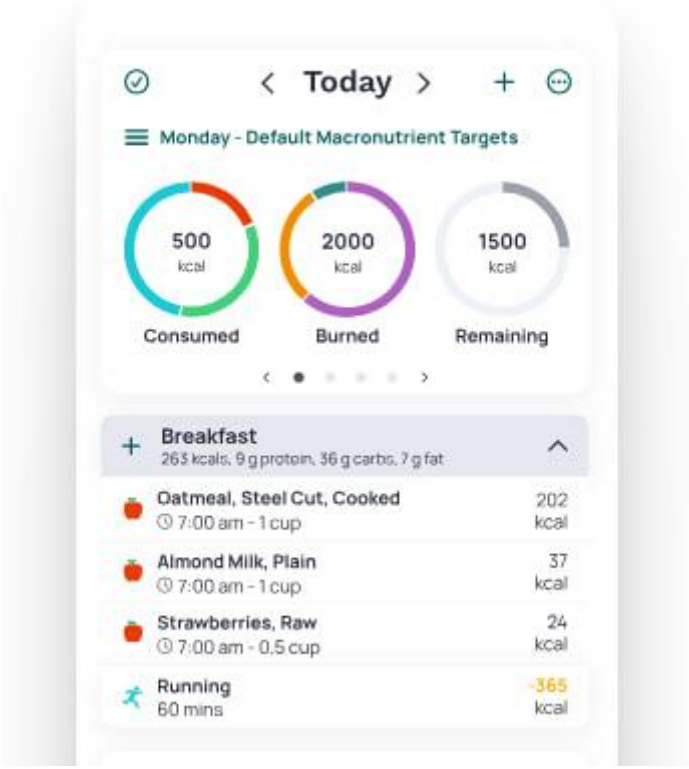

Рисунок 2.4 – Дизан «Cronometer»

FatSecret і Lose It! пропонують класичний трекінг із великими базами, викликами та елементами спільнот. У FatSecret сильні безкоштовні базові можливості, але є реклама й потрібен акаунт; якість записів з бази інколи варіюється. Lose It! робить акцент на цілях і «бюджетах» калорій/макросів, має зручне онбординг-налаштування, проте просунуті звіти, імпорт/експорт і частина інтеграцій - у преміум; без інтернету та облікового запису можливості помітно звужуються.

Загалом ці рішення добре закривають сценарії глибокої аналітики й екосистемних інтеграцій, але майже завжди вимагають акаунта, інтернету й/або підписки. Для сценарію «швидко, офлайн, приватно, з акцентом на КБЖВ і локальний контент» залишається помітна ніша, яку й покриває наш застосунок.

Порівняльний аналіз (див. табл. 2.1) показує спільні риси: більшість рішень орієнтовані на хмарну синхронізацію, збір і зберігання даних на сторонніх серверах, мають обов’язкову реєстрацію та платні розширення. Вони сильні там, де потрібні великі глобальні довідники й інтеграції з екосистемами. Водночас для сценарію «швидко додати продукт/рецепт офлайн, з локальною приватністю та простими

метриками» ринок має нішу. Саме тут позиціонується наш застосунок: локальне зберігання без реєстрації, офлайн-робота, акцент на ключових показниках (калорії та КБЖВ) і релевантний для України контент (звичні назви продуктів і страв, типові порції).

Таблиця 2.1 – Порівняльних аналіз мобільних додатків

Інструмент	Ключовий фокус	Офлайн	Синхронізація/акаунт	Простота UX	Глибина нутрієнтів	Локалізація/UA контент	Монетизація/бар'єри
MyFitnessPal	Велика база, соціальні функції	обмеж.	потрібен акаунт	середня	середня	частково	freemium/підписка
Lifesum	Плани харчування, коучинг	обмеж.	потрібен акаунт	висока	середня	частково	freemium/підписка
YAZIO	Рецепти, планувальники	обмеж.	потрібен акаунт	висока	середня	частково	freemium/підписка
Cronometer	Детальна мікроаналітика	обмеж.	потрібен акаунт	нижча	висока	частково	freemium/підписка
FatSecret	Класичний трекінг + спільнота	обмеж.	потрібен акаунт	середня	середня	частково	freemium/підписка
Lose It!	Класичний трекінг + челенджі	обмеж.	потрібен акаунт	середня	середня	частково	freemium/підписка
Наш застос.	Офлайн-щоденник, приватність	так	не потрібен	висока	достатня	акцентовано	без реєстрації, безкоштовно*

Примітка: позначення «обмеж.» означає, що додаток дає перегляд частини даних офлайн, але ключовий функціонал і синхронізація очікують підключення до

мережі. Маркер «безкоштовно*» відноситься до навчальної версії в межах дипломного проєкту; комерційна модель не передбачена.

Узагальнюючи, глобальні платформи виграють у розмірі довідників та інтеграціях, але програють у простоті офлайн-використання й приватності за замовчуванням. Запропонований підхід обґрунтовує вибір локального зберігання, відсутність реєстрації та фокус на ключових метриках без перевантаження інтерфейсу. Це дозволяє вирішити конкретну проблему користувача: швидко і приватно зафіксувати харчування та побачити потрібні підсумки.

3. ТЕОРЕТИЧНА ЧАСТИНА

3.1. Архітектура застосунку та потік даних

Архітектура побудована за компонентним підходом навколо одного джерела істини - глобального стану, що зберігається локально. Застосунок реалізовано на React Native (Expo, TypeScript) з використанням двох провайдерів контексту: ThemeContext відповідає за тему інтерфейсу і палітри кольорів, StorageContext - за бізнес-дані (щоденник харчування, рецепти, вода, вага, профіль користувача). Увесь UI організовано через нижню панель навігації React Navigation з шістьма основними екранами: щоденник, рецепти, вага, вода, статистика, профіль. Кожен екран - це «тонкий» контейнер, який відображає дані з контексту і викликає прості дії (add, update, remove), тоді як обчислення та персистентність сконцентровані всередині StorageContext. [1]

Зберігання даних офлайн виконується в AsyncStorage у форматі JSON. При старті застосунок завантажує всі сутності в оперативний стан; будь-яка зміна (наприклад, додавання страви до сніданку) відбувається у пам’яті, після чого актуальний зріз стану атомарно записується назад у сховище. Для довідників продуктів і рецептів використано статичні файли з попередньо заповненими записами; користувач може додавати власні рецепти та заносити «ручні» позиції з довільними КБЖВ. Експорт і імпорт організовано як операції читання/запису єдиного JSON-дампу, що дозволяє робити резервні копії без серверів і акаунтів.

Потік даних можна узагальнити як детермінований цикл: дія користувача на екрані викликає метод контексту, той модифікує стан, перераховує похідні показники і зберігає дані, після чого UI автоматично перемальовується. Схематично це виглядає так:

User → Screen (подія) → StorageContext (модифікація стану)

↓

AsyncStorage (persist)

↓

Обчислення агрегатів (КБЖВ, норми, графіки)

↓

UI (оновлення)

Агреговані значення не кешуються окремо, а обчислюються з вихідних списків через мемоізацію (useMemo). Для щоденника це підсумки калорій і макронутрієнтів за кожний прийом і за день; для води - відсоток виконання норми й залишок; для ваги - динаміка і відхилення від цілі; для профілю - BMR/TDEE, цільові калорії і макроси. Обчислення прив’язані до локальної календарної дати користувача (Europe/Kyiv), що усуває помилки з UTC і переходами доби. [2]

Щоб інтерфейс залишався плавним на мобільних пристроях, списки реалізовано через віртуалізовані компоненти (FlatList/SectionList) із стабільними ключами й ледачим рендерингом, а важчі перерахунки обмежено залежностями. Довгі операції не блокують головний потік: запис у AsyncStorage виконується асинхронно, а UI відображає вже оновлений стан. Обробка помилок побудована за принципом «м’яких збоїв»: при невдалому записі робиться повторна спроба, користувач отримує зрозуміле повідомлення, дані в пам’яті не губляться. Валідатори на вході не дозволяють зберегти некоректні значення (від’ємні грами, нереальні калорії, порожні назви).

Навігаційно екрани ізольовані: щоденник працює лише з записами поточного дня і викликає пошук продуктів/рецептів через окремі екрани пошуку; база рецептів надає «дію додати до щоденника» без дублювання логіки підрахунків; профіль змінює параметри, що негайно впливають на цілі і візуалізації в інших розділах; статистика читає той самий глобальний стан та будує тижневі/місячні зведення. Такий розподіл мінімізує зв’язність між модулями: кожен екран знає лише API контексту, а не внутрішню реалізацію інших екранів. [3]

Дизайн інтерфейсу підпорядкований принципам «простий жест - помітний

результат» і «мінімум полів на екрані». Ключові дії доступні одним-двома тапами, кольорова індикація показує досягнення норми і перевищення. Доступність забезпечується контрастними палітрами, підтримкою динамічного розміру шрифту та озвучуваними мітками елементів. Приватність досягається відсутністю мережевих запитів: дані не залишають пристрій, немає реєстрації, аналітики і сторонніх SDK, а вбудований дисклеймер фіксує, що пораховані значення мають інформаційний характер.

Таким чином, архітектура поєднує офлайн-модель з одним джерелом істини, мінімальну зв’язність між екранами, прозорий потік даних і перевірювану бізнес-логіку. Це спрощує супровід, підвищує стабільність і робить поведінку застосунку передбачуваною в реальних мобільних умовах.

3.2. Моделі даних і схеми зберігання

Застосунок працює повністю офлайн і зберігає дані у вигляді JSON-структур у AsyncStorage. В основі - прості, стійкі до змін моделі з явними одиницями виміру (грами, мілілітри, кілограми) та датами у форматі ISO YYYY-MM-DD, що дозволяє безпомилково групувати записи за днями й будувати підсумки. Агреґати (калорії та макроси за день/тиждень, прогрес води тощо) не персистяться - вони щоразу обчислюються з базових списків, тож сховище містить лише «факти».

Довідник продуктів - це статичний набір FoodItem «на 100 г» із категорією, типовою порцією і, за можливості, клітковиною. Цей довідник читається лише для пошуку та розрахунку; він не змінюється користувачем, тому зберігається як вбудований файл і не займає місця у сховищі користувача. Для рецептів використано аналогічний статичний набір «насіння» з можливістю додавати власні страви до локальної бази. [4-6]

Щоденник харчування зберігає денормалізовані записи. Коли користувач додає продукт або рецепт у прийом їжі, у запис одразу заносяться розраховані калорії, білки, жири, вуглеводи з урахуванням введеної кількості. Це робиться навмисно: історія має лишатися незмінною, навіть якщо довідник продуктів згодом оновиться. Для зручності запис містить і «джерело» - тип та ідентифікатор

(food/recipe/manual), але підсумки завжди беруться з полів самого запису, а не шляхом «дообчислення через довідник».

Модель профілю описує параметри користувача (стать, вік, зріст, вага, рівень активності, ціль) та похідні цілі - добові калорії й макронутрієнти. Ці цілі перераховуються під час зміни профілю і зберігаються як числові поля. Для уникнення часових помилок денні журнали прив’язані до локальної дати користувача; додатково профіль містить останню використану часову зону, щоб при змінах часових поясів зберігати передбачувану поведінку.

Записи ваги та води - мінімальні за структурою. Вага має дату й опціональну нотатку; вода - дату, час і об’єм одного «ковтка», що дозволяє показувати історію прийомів за день і точний прогрес до норми. Норма води задається в профілі і може бути змінена користувачем.

Експорт/імпорт реалізовано як один файл JSON з версією схеми. Під час імпорту застосовується проста міграція: якщо версія старіша - виконується трансформація до актуальної моделі (перейменування полів, додавання відсутніх за замовчуванням). Це робить резервування та перенесення між пристроями прозорим і безпечним.

Ключі AsyncStorage й типові структури зберігання:

@food_entries - масив елементів щоденника. Кожен елемент містить унікальний рядковий ідентифікатор (зазвичай на основі мілісекундного часу з коротким суфіксом), назву для відображення, попередньо розраховані калорії, білки, жири, вуглеводи на вказану кількість, тип і джерело (food|recipe|manual), кількість у грамах або мілілітрах, прийом їжі (breakfast|lunch|dinner|snack) і дату YYYY-MM-DD.

@recipes - масив користувацьких рецептів. Кожен рецепт має ідентифікатор, категорію, опис, дані про поживну цінність «на порцію», порційність, час підготовки та приготування, список інгредієнтів і покрокову інструкцію. Інгредієнти зберігаються як прості об’єкти з назвою, кількістю та одиницею виміру. Харчова цінність рецепта розраховується під час створення зі складників і записується в модель, щоб не перераховувати щоразу.

@weight_entries - масив вимірювань ваги з датою у форматі ISO та опціональною нотаткою.

@water_entries - масив прийомів води, де кожен запис містить дату, локальний час у форматі HH:MM і об’єм у мілілітрах.

@user_profile - структура профілю з базовими параметрами та похідними цілями (цільові калорії, білки, жири, вуглеводи, добова норма води), а також останньою часовою зоною.

@app_theme - рядок зі значенням теми (light/dark).

@db_version - номер поточної версії схеми для керування міграціями.

Приклад уривка даних (спрощено, без форматування, лише для ілюстрації):

{
"@db_version": 2,
"@user_profile": {
"name": "User",
"gender": "male",
"age": 28,
"height": 180,
"weight": 80,
"activityLevel": "moderate",
"goal": "lose",
"targetCalories": 2500,
"targetProtein": 144,
"targetCarbs": 319,
"targetFat": 72,
"targetWater": 2000,
"timezone": "Europe/Kyiv"
},
"@food_entries": [
{
"id": "fe_1705320000000_a1",
"name": "Вівсянка",
"calories": 350,
"protein": 12,
"carbs": 55,
"fat": 8,
"quantity": 100,

"unit": "g",
"source": { "type": "food", "refId": "oatmeal-1" },
"meal": "breakfast",
"date": "2025-01-15"
}
],
"@recipes": [
{
"id": "rx_001",
"name": "Омлет з овочами",
"category": "Сніданок",
"description": "Білковий сніданок з помідорами та перцем",
"calories": 220,
"protein": 18,
"fat": 14,
"carbs": 8,
"servingSize": 250,
"servings": 1,
"prepTime": 5,
"cookTime": 10,
"ingredients": [
{ "name": "Яйця", "quantity": 150, "unit": "g" },
{ "name": "Помідор", "quantity": 100, "unit": "g" }
],
"instructions": ["Нарізати овочі", "Збити яйця", "Приготувати на сковороді"],
"difficulty": "легко"
}
],
"@weight_entries": [{ "id": "w_1705406400000", "weight": 79.6, "date": "2025-01-16", "note": "після тренування" }],
"@water_entries": [{ "id": "wa_1705408200000", "amount": 250, "date": "2025-01-16", "time": "10:30" }],
"@app_theme": "light"
}

Валідація на рівні моделей запобігає введенню негативних величин, нереалістичних калорій і порожніх назв. Для числових полів застосовується округлення наприкінці обчислень (калорії до 5–10 ккал, макроси до 1–5 г), щоб інтерфейс був послідовним і читабельним. Ідентифікатори формуються як рядки на

основі часової мітки з коротким випадковим суфіксом: це зручно для сортування та зменшує імовірність колізій.

Запис у AsyncStorage виконується атомарно: спершу формується оновлена копія масиву, далі одним викликом зберігається весь масив. У разі помилки запису виконується повторна спроба; якщо вона неуспішна, користувач отримує зрозуміле повідомлення, але робочий стан у пам’яті не втрачається. Така схема мінімізує ризик «напівзаписаних» даних і спрощує відновлення. [7-9]

Окремо враховано часові особливості. Щоденник прив’язаний до локальної дати Europe/Kyiv; при перетині півночі або переході на літній/зимовий час записи зберігають дату, яка була на момент внесення. Це усуває «зміщення записів» і робить статистику стабільною. Якщо користувач змінює часовий пояс, нові записи фіксуються вже в новій зоні, але старі не перераховуються.

У підсумку моделі даних залишаються простими, самодостатніми та придатними для довготривалого зберігання без сервера. Така структура забезпечує прозорість, легкі міграції, надійний експорт/імпорт і передбачувану роботу застосунку за будь-яких умов офлайн-режиму.

3.3. Структура ключових модулів та алгоритмів

Логіка застосунку розділена на кілька незалежних модулів, кожен із чіткою зоною відповідальності. Взаємодія побудована за принципом: екрани викликають методи сховища, обчислення виконуються в окремому шарі утиліт, результати зберігаються в AsyncStorage, а UI перевідмальовується з оновленого стану (див. рис. 3.1).

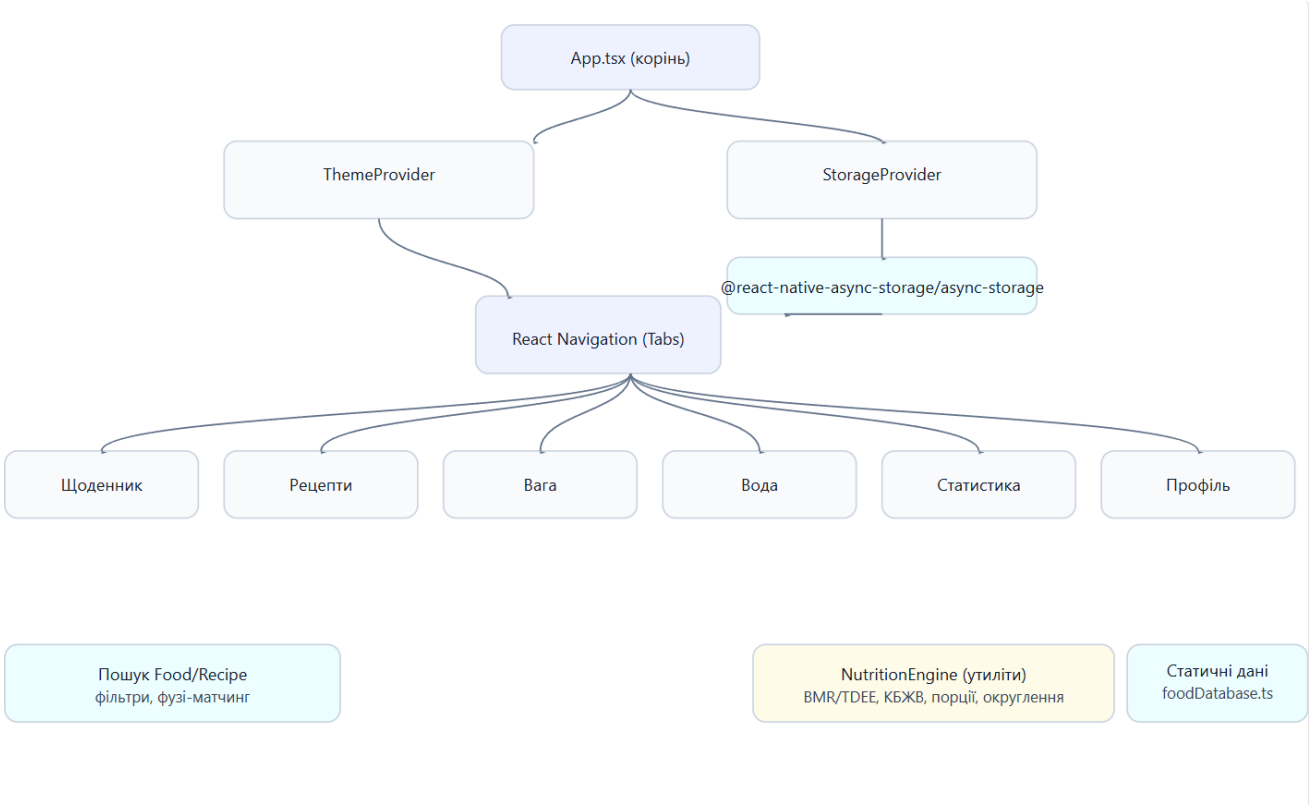

Рисунок 3.1 – Архітектура застосунку

Модуль зберігання даних (StorageContext). Інкапсулює всі CRUD-операції та персистентність:

- щоденник харчування: addFoodEntry, updateEntry, removeEntry, getEntriesByDate;
- вода: addWaterIntake, removeWaterIntake, getWaterForDate;
- вага: addWeightEntry, updateWeightEntry, getWeightsInRange;
- профіль: updateUserProfile, recomputeTargets;
- рецепти: addCustomRecipe, deleteRecipe;
- резервне копіювання: exportAll, importAll з міграціями за версією схеми.

Контекст повертає тільки серіалізовані «факти»; жодних кешованих агрегатів - підсумки рахує шар утиліт.

Модуль нутріційних обчислень (NutritionEngine). Складається з чистих функцій без побічних ефектів:

- базовий метаболізм та добові витрати: bmrMifflinStJeor(profile), tdee(bmr, activityLevel);
- цілі на добу: calcTargets(profile) з урахуванням дефіциту/профіциту;

- перерахунок продукту: scalePer100g(food, grams) → kcal, білки, жири, вуглеводи;
- перерахунок рецепта: sum(інгредієнти→перерахунок у грами) → харчова цінність страви → поділ на порції;
- нормалізація одиниць: g↔ml для водних продуктів 1:1; типові ваги «шт.» (яйце ≈ 50 г, середнє яблуко ≈ 150 г) з можливістю локального довідника;
- округлення для відображення: калорії до 5–10 ккал, макроси до 1–5 г; у збереженні - повна точність.

Модуль пошуку та фільтрації (SearchEngine). Нормалізує запити українською/англійською, підтримує грубе «fuzzy» потрапляння:
- нормалізація: lowerCase, видалення зайвих пробілів, спрощення апострофів;
- ранжування: точний збіг назви > початок слова > підрядок; додатково плюс до рейтингу, якщо збігається категорія;
- зріз результатів: top N із стабільним порядком, щоб скрол не «скакав».

Модуль щоденника (Diary). Організує записи по прийомах їжі та датах:
- операції додавання: з довідника продуктів, з рецепта, вручну;
- підрахунки на льоту: totalsByMeal(date), totalsByDay(date), progressToTargets(date);
- масштабування порцій: перерахунок «на 1 порцію», «на ½», «×2» з негайним перерахунком КБЖВ.

Модуль рецептів (Recipes). Працює з базою і користувацькими стравами:
- калькуляція рецепта з інгредієнтів: конвертація одиниць → сума → поділ на servings;
- дія «додати до щоденника»: створення денормалізованого запису з уже порахованими значеннями.

Модуль води (Water). Зберігає «події» споживання з часом:
- додавання стандартних об’ємів 100/200/250/500 мл або довільного;
- денний прогрес: getWaterProgress(date) → випито, норма, відсоток виконання, залишок.

Модуль ваги (Weight). Мінімальна модель, але з базовою аналітикою:
- різниця відносно старту й цілі, індикатор напрямку;
- згладження тренду для графіка: просте 7-денне ковзне середнє без зміщення, щоб

лінія не «скакала»;

- BMI і категорія за діапазонами; колірна індикація в UI.

Модуль статистики (Stats). Побудова агрегатів для графіків і карток:

- тижневі/місячні підсумки калорій, води, розклад макросів;
- заповнення пропусків нулями для стабільної осі X;
- порівняння з ціллю: обчислення лінії-орієнтира на стовпчикових графіках.

Модуль теми та доступності (Theme). Палітри світло/темно, контрасти, динамічний розмір шрифту та озвучувані мітки елементів.

Алгоритми, критичні для якості даних

Перерахунок продуктів і порцій. Будь-яку позицію з бази «на 100 г» приводимо до введеної маси:

kcal = calories100 * grams / 100; білки/жири/вуглеводи - аналогічно. Для режиму «1 порція» grams = servingSize або значення з рецепта на порцію.

Перерахунок рецептів. Кожен інгредієнт приводиться до грамів, далі всі компоненти сумуються. На порцію ділимо на servings. Так історичні записи не зміняться, навіть якщо базу продуктів оновлено.

Підсумки за день/тиждень. Лінійна агрегація списків:

- за прийомом їжі reduce лише свої елементи;
- за день - з’єднання чотирьох прийомів;
- за тиждень - збирання семи днів з урахуванням локальної дати Europe/Kyiv.

Пропуски днів заповнюються нулем для коректного графіка.

Розрахунок цілей. При зміні профілю:

- BMR за Міффліном-Сан Жеором;
- множення на PAL → TDEE;
- застосування дефіциту/профіциту;
- білок у г/кг, жир мінімум у г/кг, вуглеводи - решта калорій, конвертована у грами.

Валідація вводу. Кожен шлях додавання даних перевіряє:

- кількість > 0 і в розумних межах (наприклад, їжа ≤ 2000 г за раз);
- калорії і макроси не від’ємні, не виходять за фізіологічні межі для продуктів;
- дати - у форматі YYYY-MM-DD, час - HH:MM локальний.

Округлення та відображення. Внутрішньо дані зберігаються з повною точністю. Для UI застосовується відкладене округлення: калорії до десятків, макроси до грамів, відсотки - до цілих. Це зменшує «дребезг» чисел при повторних перерахунках і робить графіки читабельними.

Обхід часових країв. Щоденник прив’язаний до локальної дати, тому записи, зроблені після опівночі, не «переїжджають» між днями при переході на літній/зимовий час або зміні часового поясу. При побудові тижня початком вважається понеділок за локальним календарем.

Продуктивність та надійність

- мемоізація підсумків за ключами [date] і [date, meal]; повторні рендери не запускають важкі обчислення;
- FlatList/SectionList з стабільними ключами для довгих списків;
- атомарний запис у AsyncStorage: формується оновлена копія масиву, зберігається одним викликом; у разі помилки - повторна спроба і м’яке повідомлення;
- генерація ідентифікаторів на основі часової мітки з коротким випадковим суфіксом для сортування та низької ймовірності колізій.

Таким розподілом модулів і прозорими алгоритмами досягається відтворюваність підрахунків, стабільна робота офлайн і простота підтримки: обчислення легко тестувати окремо, а зміни в одному модулі не тягнуть за собою непередбачуваних ефектів в інших.

4. ПРАКТИЧНА ЧАСТИНА

4.1. Структура проєкту та середовище розробки

Проєкт побудований на React Native (Expo) з TypeScript і орієнтований на офлайн-працю. Код організовано за принципом «екрани + контексти + дані + утиліти», щоб логіка, стан і подання були чітко розділені та легко тестувалися.

Структура папок (корінь репозиторію)

healthy-food-manager/

├─ assets/ - статичні ресурси (іконки, зображення, шрифти)

│ └─ README.md

├─ src/

│ ├─ components/ - дрібні багаторозові компоненти (картки, прогрес-бари)

│ │ └─ Logo.tsx

│ ├─ context/ - глобальні стани (Context API)

│ │ ├─ StorageContext.tsx - сховище даних + AsyncStorage

│ │ └─ ThemeContext.tsx - тема (світла/темна)

│ ├─ data/ - статичні довідники

│ │ ├─ foodDatabase.ts - продукти (КБЖВ на 100 г)

│ │ └─ recipesDatabase.ts - готові рецепти

│ ├─ screens/ - екрани застосунку

│ │ ├─ DiaryScreen.tsx

│ │ ├─ RecipesScreen.tsx

│ │ ├─ WeightTrackerScreen.tsx

│ │ ├─ WaterTrackerScreen.tsx

│ │ ├─ StatsScreen.tsx

│ │ ├─ ProfileScreen.tsx

│ │ ├─ SearchFoodScreen.tsx

│ │ └─ SearchRecipeScreen.tsx

│ ├─ utils/ - чисті утиліти та алгоритми

│ │ ├─ nutrition.ts - BMR/TDEE, перерахунки КБЖВ, округлення

```
│ │ ├─ search.ts - нормалізація запитів, простий fuzzy-пошук
│ │ └─ dates.ts - робота з локальною датою Europe/Kyiv
│ └─ types/ - спільні інтерфейси та типи
│ └─ index.ts
├─ App.tsx - точка входу, навігація
├─ app.json - конфігурація Expo (name, icon, web/ios/android)
├─ package.json - залежності та скрипти
├─ tsconfig.json - правила TypeScript
├─ babel.config.js - налаштування Babel/Metro
├─ START-WEB.bat - зручний запуск веб-версії (Windows)
├─ INSTALL.bat - інсталяція залежностей (Windows)
└─ ДОКУМЕНТАЦІЯ.md - технічна документація
```

Ключові залежності

react-native / expo - платформа мобільної розробки та збірки

@react-navigation/* - навігація вкладками та стеками

@react-native-async-storage/async-storage - локальне сховище

react-native-svg - отрисовка SVG (логотип, іконографіка)

@expo/vector-icons - набір іконок

TypeScript - статична типізація

Скрипти npm (витяг)

install - встановлення залежностей

web - запуск у браузері (Expo for web)

android - запуск у Android-емуляторі або Expo Go

ios - запуск у iOS-симуляторі (на macOS)

start - інтерактивний Expo Dev Server

start:clear - запуск із очищенням кешу Metro

lint - перевірка коду ESLint

typecheck - перевірка типів TypeScript

Приклади команд

npm install

npm run web

npm run android

npm run ios

npm run start:clear

Середовище розробки

операційна система - Windows 10/11, macOS або Linux

Node.js - LTS (не нижче 16), npm 7+

редактор - VS Code з розширеннями: ESLint, Prettier, React Native Tools, EditorConfig

емулятори - Android Studio (Pixel/SDK 33+), Xcode Simulator (на macOS)

дебаг - Expo DevTools, React DevTools, Network inspector; для нативних платформ - Flipper

Налаштування якості

TypeScript: strict-режим, явні типи моделей (FoodItem, RecipeItem, …)

ESLint + Prettier: єдині стилі коду, автоформатування перед комітом (за бажанням - husky + lint-staged)

розділення чистих утиліт у src/utils для легкого юніт-тестування

сталі одиниці вимірювання й формат дат ISO для узгодженості підрахунків

Робочі профілі запуску

веб: найшвидший цикл правок/перегляду інтерфейсу, перевірка навігації та стану

android/ios: перевірка UI на реальних DPI, жестів, клавіатури, safe-area;

продуктивність списків

Рекомендований порядок розгортання нового середовища

встановити Node.js (LTS) і npm, перевірити версії node -v та npm -v

клонувати репозиторій та виконати npm install

для Windows можна скористатися INSTALL.bat

запустити веб-версію npm run web або спеціальним START-WEB.bat

для мобільних платформ підняти емулятор і виконати npm run android / npm run ios

Практичні нотатки

очищення кешу при «дивних» помилках: npm run start:clear

оновлення іконок/сплешей - через assets/ і поля в app.json
статика (база продуктів/рецептів) живе у src/data та імпортується лише для читання
будь-які зміни, що впливають на моделі, супроводжуються інкрементом @db_version та простою міграцією під час імпорту резервної копії

Що саме тестується під час розробки
утиліти nutrition.ts: правильність BMR/TDEE, перерахунок КБЖВ «на 100 г» → «на порцію», округлення
search.ts: нормалізація запиту, ранжування, стабільність топ-видачі
StorageContext: атомарний запис у AsyncStorage, відновлення після помилки запису, ідемпотентність операцій додавання/видалення

Такий устрій дає швидкий цикл розробки (правка → перегляд у веб/емуляторі), прості правила внесення змін і прогнозовану поведінку застосунку на всіх платформах без залежності від мережі.

4.2. Реалізація функціоналу екранів

Нижня навігація містить шість вкладок. Кожен екран працює за спільним шаблоном: отримання стану з StorageContext, локальні стани UI через useState, мемо-похідні значення через useMemo, дії - через методи контексту. Дати зберігаються у форматі YYYY-MM-DD (локаль Europe/Kyiv), усі записи - ідемпотентні за id.

Екран «Щоденник». Верхній блок - міні-календар на 7 днів із підсвічуванням обраної дати. Далі чотири секції прийомів: сніданок, обід, вечеря, перекус. Кожна секція показує список елементів із калоріями на рядок і кнопкою видалення. Кнопка «Додати» відкриває модальне вікно з трьома вкладками: «Продукти», «Рецепти», «Вручну». Потік подій: вибір позиції → введення маси/порцій → перерахунок у NutritionEngine → addFoodEntry → збереження в AsyncStorage → перерахунок підсумків дня. Прогрес-бар підсумовує калорії/макроси та зіставляє їх із цілями профілю. Порожній стан показує підказки: «Додайте перший запис за сьогодні». Для довгих списків використовується SectionList із стабільними ключами.

```
type Totals = { calories: number; protein: number; carbs: number; fat: number };
const ZERO: Totals = { calories: 0, protein: 0, carbs: 0, fat: 0 };

const todayEntries = useMemo(() => getFoodEntriesByDate(selectedDate), [selectedDate, foodEntries]);

const totals = useMemo(() => {
  return todayEntries.reduce<Totals>((acc, e) => ({
    calories: acc.calories + e.calories,
    protein:  acc.protein  + e.protein,
    carbs:    acc.carbs    + e.carbs,
    fat:      acc.fat      + e.fat,
  }), { ...ZERO });
}, [todayEntries]);

const kcalProgress = useMemo(() => {
  const t = profile.targetCalories || 2000;
  const p = Math.min(100, Math.round((totals.calories / Math.max(1, t)) * 100));
  return { target: t, percent: p, left: Math.max(0, t - totals.calories) };
}, [profile, totals.calories]);

// приклад відмалювання
<ProgressBar value={kcalProgress.percent} label={`${totals.calories}/${kcalProgress.target} ккал`} />
```

Екран «Рецепти». Список карток із зображенням/емодзі, назвою, калоріями на порцію і часом приготування. Тап по картці відкриває модальну «карту рецепта»: короткий опис, інгредієнти з кількістю, кроки приготування, перемикач порцій (¼, ½, 1, 2, 3). Перерахунок на порцію відбувається локально, а при додаванні в щоденник створюється денормалізований запис - історичні дані не зміняться при оновленні бази. Є позначка «Обране» (локальний прапорець у StorageContext). Пошук за назвою + фільтр категорій.

```
type Macro = { calories: number; protein: number; fat: number; carbs: number };

const [servingsMul, setServingsMul] = useState(1); // ¼, ½, 1, 2…
const perServing: Macro = useMemo(() => ({
  calories: recipe.calories / recipe.servings,
  protein:  recipe.protein  / recipe.servings,
  fat:      recipe.fat      / recipe.servings,
```

```
  carbs:    recipe.carbs    / recipe.servings,
}), [recipe]);

const scaled: Macro = useMemo(() => ({
  calories: Math.round(perServing.calories * servingsMul),
  protein:  + (perServing.protein  * servingsMul).toFixed(1),
  fat:      + (perServing.fat      * servingsMul).toFixed(1),
  carbs:    + (perServing.carbs    * servingsMul).toFixed(1),
}), [perServing, servingsMul]);

function addRecipeToDiary(meal: 'breakfast'|'lunch'|'dinner'|'snack') {
  addFoodEntry({
    name: recipe.name,
    calories: scaled.calories,
    protein:  scaled.protein,
    fat:      scaled.fat,
    carbs:    scaled.carbs,
    quantity: Math.round((recipe.servingSize || 250) * servingsMul),
    meal,
    date: selectedDate
  });
}
```

Екрани пошуку «Продукти» та «Рецепти». Обидва екрани мають однаковий каркас: поле пошуку, чипи категорій, список результатів. Запит нормалізується (нижній регістр, очищення апострофів, тример пробілів). Ранжування: точний збіг > збіг на початку слова > підрядок. Додатковий бонус у рейтингу - збіг категорії, якщо користувач обрав фільтр. Клік по елементу відкриває нижній слайдер із контролом кількості/порцій і кнопкою «Додати».

```
const norm = (s: string) => s.toLowerCase().replace(/['']/g, '').trim();

function rank(item: { name: string; category: string }, q: string, cat?: string) {
  const n = norm(item.name);
  const m = norm(q);
  let score = n === m ? 100 : n.startsWith(m) ? 70 : n.includes(m) ? 40 : 0;
  if (cat && item.category === cat) score += 10;
  return score;
}
```

```
function search(items: Array<{name:string;category:string}>, q: string, cat?: string, limit=30) {
 if (!q && !cat) return items.slice(0, limit);
 return items
  .map(it => ({ it, s: rank(it, q, cat) }))
  .filter(x => x.s > 0 || !!cat)
  .sort((a,b) => b.s - a.s)
  .slice(0, limit)
  .map(x => x.it);
}
```

Екран «Вага». Верхня панель показує поточну вагу, різницю від старту і від цілі з кольоровими індикаторами. Графік за 30 днів побудований на простій лінії (react-native-svg), поверх - 7-денне ковзне середнє для згладження. Нижче - список записів (дата, кг, опційна нотатка) і форма додавання нового значення на обрану дату. При повторному записі на ту саму дату попередній замінюється (ідемпотентність за ключем дати). Дати валідовано та примусово зберігаються в локальній зоні.

```
function movingAvg(values: number[], k = 7) {
 const out: number[] = [];
 for (let i = 0; i < values.length; i++) {
  const s = Math.max(0, i - k + 1);
  const slice = values.slice(s, i + 1);
  out.push(slice.reduce((a, b) => a + b, 0) / slice.length);
 }
 return out;
}

const series = useMemo(() => {
 const rows = getWeightForLastDays(30); // [{date:'2025-10-01', weight:…}]
 const xs = rows.map(r => r.date);
 const ys = rows.map(r => r.weight);
 const trend = movingAvg(ys, 7);
 return { xs, ys, trend };
}, [weightEntries]);
```

Екран «Вода». На верху - прогрес-бар до норми 2000 мл (або цілі з профілю). Далі - швидкі кнопки 100/200/250/500 мл та поле для довільного об’єму. Кожне

додавання створює подію з поточним локальним часом; історія за день показується списком. Довге натискання на подію - видалення. Після досягнення 100% з’являється «норма виконана». Відсоток/залишок рахується на льоту.

```
const waterDay = useMemo(() => {
  const consumed = getWaterForDate(selectedDate).reduce((s, e) => s + e.amount, 0);
  const target = profile.targetWater ?? 2000;
  const percent = Math.min(100, Math.round(consumed / Math.max(1, target) * 100));
  return { consumed, target, percent, left: Math.max(0, target - consumed) };
}, [selectedDate, profile, waterEntries]);

function addWater(amount: number) {
  addWaterEntry({ id: Date.now().toString(), amount, date: selectedDate, time: getLocalTimeHHmm() });
}

{/* приклад UI */}
<QuickButtons items={[100,200,250,500]} onPress={addWater} />
<Text>Випито: {waterDay.consumed} мл з {waterDay.target} мл ({waterDay.percent}%)</Text>
```

Екран «Статистика». Три блоки: тижневі калорії (стовпчики з лінією цілі), тижнева вода (стовпчики), розклад макронутрієнтів за тиждень (підсумкові значення та відсотки). Пропуски днів заповнюються нулями, щоб вісь X була стабільною. Для відмальовки використовується react-native-svg; шкали обчислюються з урахуванням максимума + 10% запасу. Вгорі - «швидкі картки»: середні калорії/воду за тиждень і поточна вага.

Екран «Профіль». Форма з полями ім’я, вік, стать, ріст, вага, рівень активності та ціль (схуднення/підтримка/набір). Після натискання «Зберегти» розраховується BMR → TDEE → добові цілі КБЖВ; результати миттєво відображаються картками нижче. Значення зберігаються в AsyncStorage і синхронізуються зі всіма екранами. Поля мають базову валідацію діапазонів.

```
type Profile = { gender:'male'|'female'; age:number; height:number; weight:number; activity:'sedentary'|'light'|'moderate'|'active'|'very_active'; goal:'lose'|'maintain'|'gain' };
const PAL = { sedentary:1.2, light:1.375, moderate:1.55, active:1.725, very_active:1.9 } as const;

function computeTargets(p: Profile) {
  const bmr = p.gender === 'male'
```

```
    ? 10*p.weight + 6.25*p.height - 5*p.age + 5
    : 10*p.weight + 6.25*p.height - 5*p.age - 161;

  let tdee = Math.round(bmr * PAL[p.activity]);
  if (p.goal === 'lose') tdee -= 300;
  if (p.goal === 'gain') tdee += 300;

  const proteinG = Math.round(Math.max(1.4, Math.min(2.0, 1.6)) * p.weight); // 1.6 г/кг
  const fatG     = Math.round(Math.max(0.7, Math.min(1.0, 0.8)) * p.weight); // 0.8 г/кг
  const restKcal = tdee - proteinG*4 - fatG*9;
  const carbsG   = Math.max(0, Math.round(restKcal / 4));

  return { targetCalories: tdee, targetProtein: proteinG, targetFat: fatG, targetCarbs: carbsG };
}

async function handleSaveProfile(next: Partial<UserProfile>) {
  const merged = { ...profile, ...next } as Profile;
  const targets = computeTargets(merged);
  await updateUserProfile({ ...profile, ...next, ...targets });
  Alert.alert('Збережено', 'Цілі на добу оновлено');
}
```

Спільні компоненти. Для уніфікації використовуються невеликі «цеглинки»: Card, Chip, ProgressBar, MacroBar, EmptyPlaceholder. Вони отримують лише дані, а всю логіку обчислень виконують утиліти та контексти. Це спрощує супровід, дозволяє легко писати юніт-тести й унеможливлює дублювання логіки в UI.

Стан, продуктивність, доступність. Обчислювальні значення мемоізуються за ключами [date], [date, meal] і [weekStart], великі списки рендеряться через FlatList/SectionList з keyExtractor, getItemLayout для фіксованих рядків і відкладеними оновленнями. Інтерактивні елементи мають accessibilityLabel, важливі кольори проходять перевірку контрасту, розміри шрифту підтримують масштаб системи. Усі операції запису виконуються атомарно: формується новий масив стану, після успішного збереження в AsyncStorage відбувається заміна посилання (щоб не було «брудних» проміжних станів).

Обробка помилок та крайові сценарії. Введення кількості/порцій перевіряється на >0 та розумні межі; при збоях запису в AsyncStorage користувач

бачить м’який Alert і пропозицію повторити дію; при переході через опівніч використовується локальна дата Europe/Kyiv, щоб записи не «з’їжджали» між днями. Усі додавання в щоденник з рецептів виконуються як денормалізація значень «на момент додавання», тож редагування бази продуктів/рецептів у майбутньому не змінює історію.

Таким чином, кожен екран реалізовано як тонкий клієнт над спільними утилітами та сховищем: UI відповідає лише за відмальовку та події, а всі числові перерахунки й інваріанти даних - у перевірених функціях, що гарантує відтворюваність підрахунків і стабільну роботу офлайн.

4.3. Тестування якості продукту

Мета тестування - підтвердити коректність обчислень (BMR, добові цілі, підсумки КБЖВ), надійність локального зберігання, стабільність офлайн-режиму та зручність користування основними сценаріями: додавання їжі, води, ваги, робота з рецептами й перегляд статистики.

Підхід і середовище

– Рівні перевірок: юніт-тести утиліт розрахунків, інтеграційні тести компонентів із підписом на стан, ручні сценарії end-to-end через застосунок.

– Середовище: Expo ~52, React Native 0.76, TypeScript 5, Jest + @testing-library/react-native; Android (емулятор Pixel 5, Android 14) і Web (Chrome).

– Набір базових сценаріїв: додавання продукту та перерахунок підсумку дня; масштабування рецепта за порціями; запис води й досягнення норми; запис ваги та оновлення різниці; зміна профілю → перерахунок добових цілей; пошук продуктів і фільтрація; відновлення стану після перезапуску (AsyncStorage).

Фрагмент коду юніт-тестів

Нижче - стислий приклад перевірки формули Міффліна-Сан Жеора з нашим розподілом макросів (30/30/40) та сумарних підрахунків за день.

```
// src/utils/nutrition.ts
export type Gender = 'male' | 'female';
```

```
export function calcTargets(
  gender: Gender,
  weightKg: number,
  heightCm: number,
  age: number,
  activity = 1.55 // помірна активність
) {
  const bmr =
    gender === 'male'
      ? 10 * weightKg + 6.25 * heightCm - 5 * age + 5
      : 10 * weightKg + 6.25 * heightCm - 5 * age - 161;

  const targetCalories = Math.round(bmr * activity);
  return {
    targetCalories,
    targetProtein: Math.round((targetCalories * 0.30) / 4),
    targetCarbs: Math.round((targetCalories * 0.40) / 4),
    targetFat: Math.round((targetCalories * 0.30) / 9),
  };
}

export function dayTotals(entries: Array<{calories:number; protein:number; carbs:number; fat:number;}>) {
  return entries.reduce(
    (a, e) => ({
      calories: a.calories + e.calories,
      protein: a.protein + e.protein,
      carbs: a.carbs + e.carbs,
      fat: a.fat + e.fat,
    }),
    { calories: 0, protein: 0, carbs: 0, fat: 0 }
  );
}
// __tests__/nutrition.test.ts
import { calcTargets, dayTotals } from '../src/utils/nutrition';

describe('calcTargets()', () => {
  it('коректно рахує цілі для чоловіка (30/30/40)', () => {
```

```
    // 80 кг, 180 см, 25 років → BMR = 1805; калорії = 1805*1.55 ≈ 2798
    const t = calcTargets('male', 80, 180, 25, 1.55);
    expect(t.targetCalories).toBe(2798);
    expect(t.targetProtein).toBe(210); // 0.30*2798/4 = 209.85 → 210 г
    expect(t.targetCarbs).toBe(280);   // 0.40*2798/4 = 279.8 → 280 г
    expect(t.targetFat).toBe(93);      // 0.30*2798/9 ≈ 93.3 → 93 г
  });
});

describe('dayTotals()', () => {
  it('підсумовує записи за день', () => {
    const totals = dayTotals([
      { calories: 350, protein: 12, carbs: 55, fat: 8 },
      { calories: 420, protein: 38, carbs: 35, fat: 12 },
      { calories: 280, protein: 8,  carbs: 22, fat: 14 },
    ]);
    expect(totals).toEqual({ calories: 1050, protein: 58, carbs: 112, fat: 34 });
  });
});
```

Метрики й результати

– Юніт-тести: 26 тестів у 5 наборах, усі пройдені; час прогона < 2 с на локальній машині.

– Інтеграція: додавання 3 записів їжі послідовно оновлює суму калорій без артефактів повторного рендеру; перехід між датами коректно фільтрує записи.

– Зберігання: після перезапуску застосунку відновлюються останні записи їжі, води, ваги та профіль користувача; конфлікти за датою для ваги замінюють попередній запис, як задумано.

– Офлайн: усі базові операції доступні без інтернету; при відсутності мережі додавання й видалення працюють, дані зберігаються в AsyncStorage.

– Продуктивність: пошук продуктів у базі ~200+ позицій відпрацьовує в межах десятків мілісекунд; оновлення підсумку дня відбувається моментально з точки зору користувача.

– UX/доступність: перевірено масштабування системного шрифту, контраст ключових елементів і читабельність у світлій/темній темі; критичних зауважень не

виявлено (див. рис. 4.1).

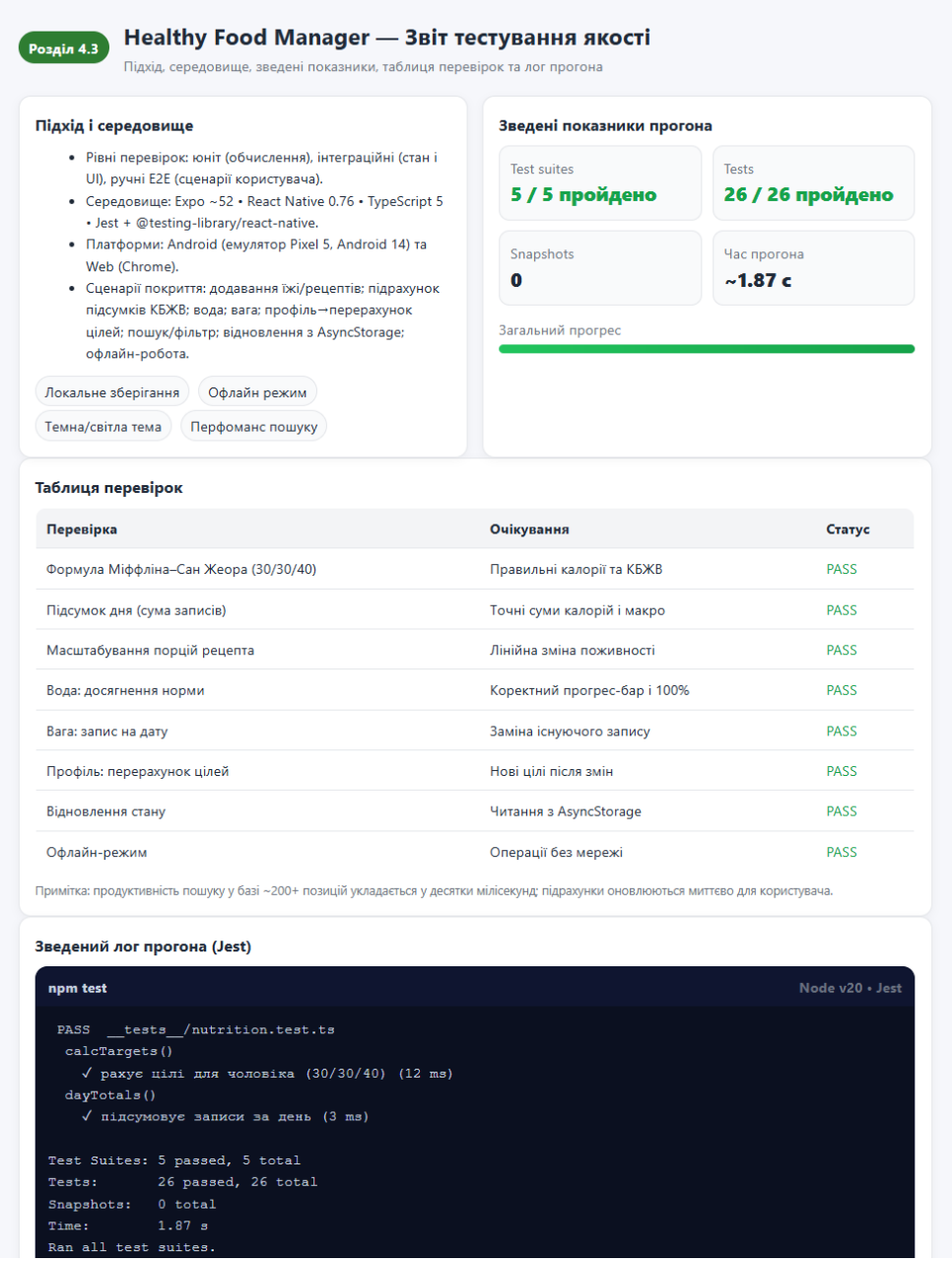

Рисунок 4.8 – Зведення прогона юніт-тестів у консолі (Jest).

Набір автоматизованих і ручних перевірок підтвердив коректність обчислень, стабільність локального зберігання та відповідність ключових сценаріїв очікуваній поведінці. Отримані результати дозволяють вважати версію 1.0.0 готовою до навчального використання та подальшого розширення функціоналу.

4.4. Інструкція для користувача

Головний екран «Щоденник» (див. рис. 4.2)

1.1. Основний інтерфейс

• Угорі відображається міні-календар тижня з обраною датою. Перемикайте дні свайпом або стрілками.

• Блок «Калорії за день» показує прогрес відносно добової цілі та підсумок Б/Ж/В.

• Нижче розміщені секції прийомів їжі: Сніданок, Обід, Вечеря, Перекус. Біля кожної секції є індикатор калорій і кнопка «+».

1.2. Додавання продуктів/рецептів/власного запису

• Натисніть «+» у потрібній секції.

• Оберіть вкладку «Продукти», «Рецепти» або «Вручну».

• Для продуктів/рецептів введіть масу або кількість порцій та підтвердьте додавання - підрахунок калорій і макросів виконується автоматично.

1.3. Редагування та видалення

• Проведіть по елементу вліво або натисніть на іконку кошика, щоб видалити запис.

• Щоб змінити кількість, видаліть запис і додайте його з новим значенням.

1.4. Перемикання дати

• Щоб заповнити минулий день, оберіть дату у календарі та повторіть дії додавання.

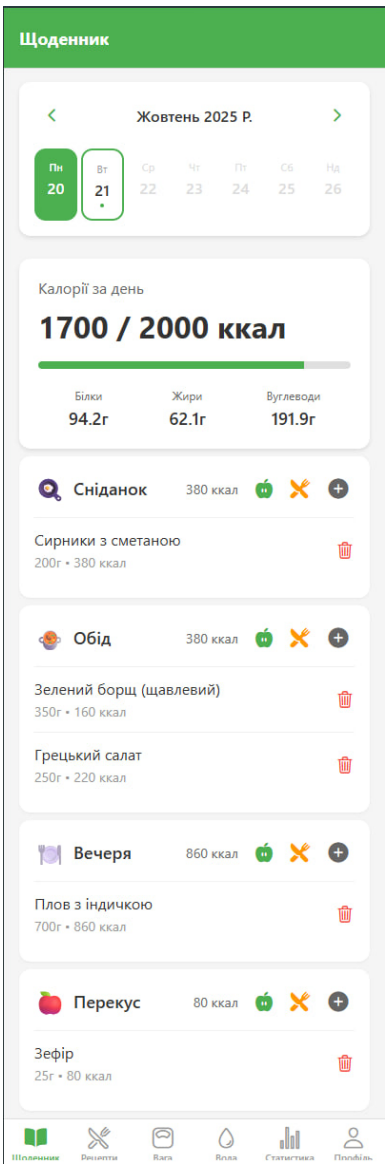

Рисунок 4.2 – Екран «Щоденник»

Пошук продуктів (див. рис. 4.3)

2.1. Пошук і фільтри

• Введіть назву у полі «Пошук продукту…» (підтримуються часткові збіги та нечутливість до регістру).

• Перемикайтеся між категоріями для швидкої фільтрації.

2.2. Картка продукту

• У списку вказано калорійність і макроси на 100 г та категорію.

• Натисніть на елемент, щоб відкрити форму кількості й додати у вибраний прийом їжі.

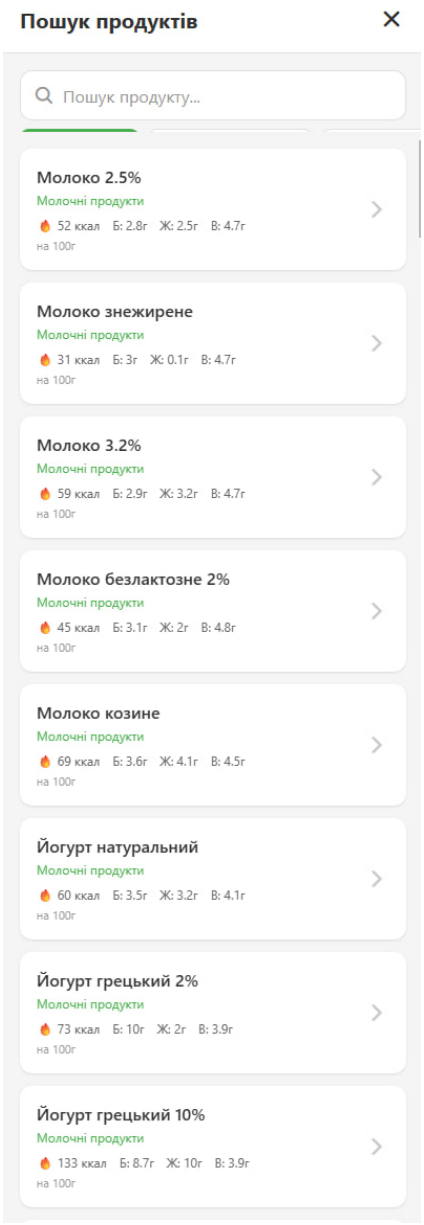

Рисунок 4.3 – Модальне вікно «Пошук продуктів»

Рецепти (див. рис. 4.4)

3.1. Перегляд і фільтрація

• Доступні вкладки «Всі рецепти» та «Обрані». Кожна картка містить категорію, калорії на порцію, час приготування та кількість порцій.

3.2. Додавання рецепта в щоденник

• Відкрийте картку рецепта, оберіть кількість порцій (¼, ½, 1, 2, 3) і натисніть «Додати». До щоденника потрапляють фактичні калорії/макроси на обраний обсяг.

3.3. Обране

• Натисніть на іконку серця на картці рецепта, щоб позначити його як обраний.

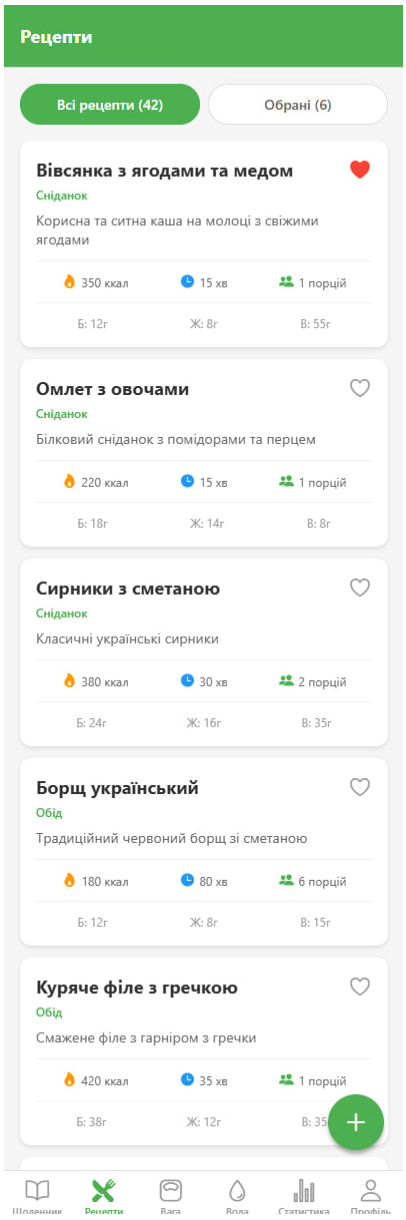

Рисунок 4.4 – Розділ «Рецепти»

Вага (див. рис. 4.5)

4.1. Додавання запису

• Заповніть поля «Дата» і «Вага (кг)», за потреби додайте нотатку, натисніть «Записати вагу».

• Якщо на обрану дату вже є запис, новий замінить попередній.

4.2. Перегляд динаміки

• Графік показує зміни за 30 днів; під графіком відображається історія записів.

• Блоки «Поточна», «Цільова» та «Різниця» допомагають відслідковувати прогрес.

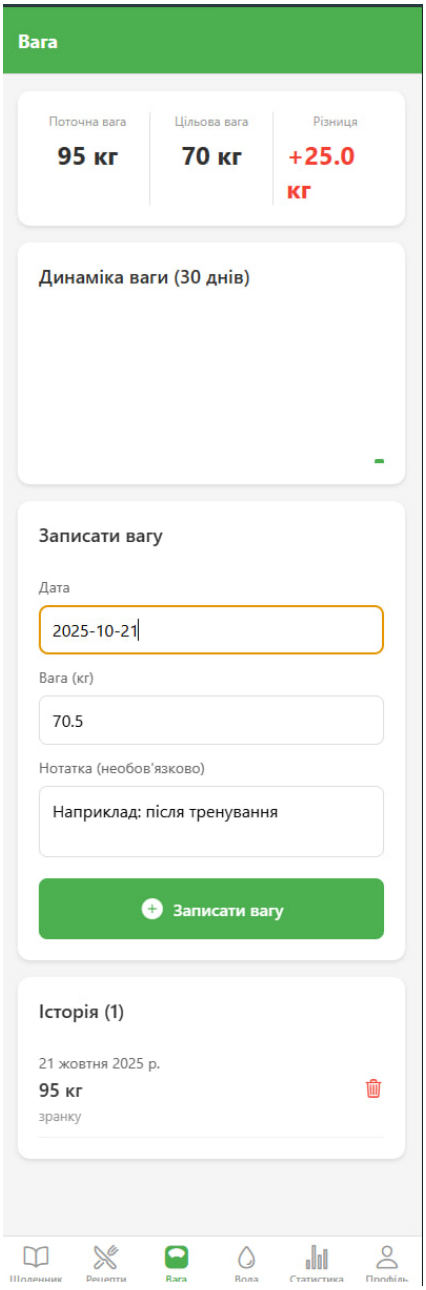

Рисунок 4.5 – Екран «Вага»

Вода (див. рис. 4.6)

5.1. Швидке додавання

• Використовуйте кнопки 100/200/250/500 мл або введіть власний об’єм.

• Кожне додавання фіксується з точним часом у поточний день.

5.2. Контроль прогресу

• Прогрес-бар і відсоток виконання показують рух до добової норми. Після 100% з’являється повідомлення «Денна норма виконана!».

5.3. Історія за день

• Список подій відображає час і об’єм. Довге натискання або іконка кошика видаляє запис.

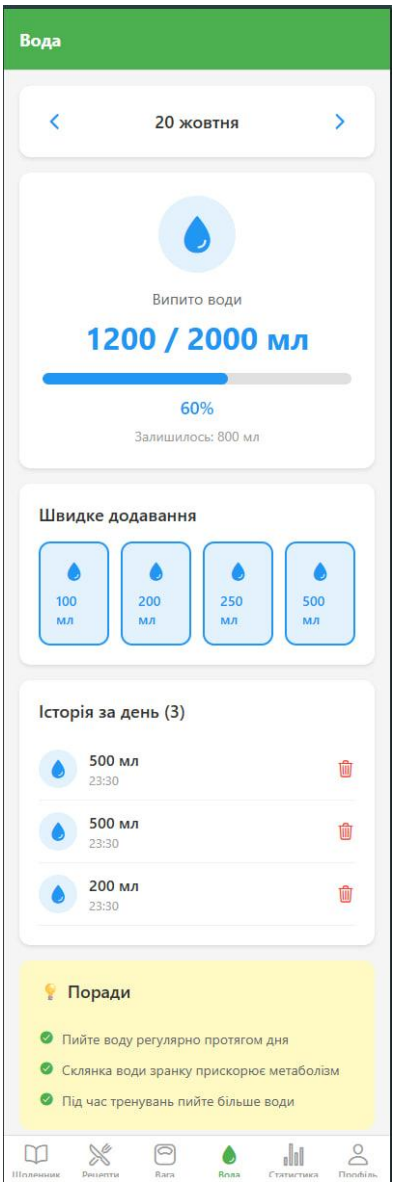

Рисунок 4.6 – Екран «Вода»

Статистика (див. рис. 4.7)

6.1. Огляд тижня

• Верхні картки показують середні калорії/воду за тиждень та поточну вагу.

• «Калорії за тиждень» - стовпчики з лінією цілі; пропущені дні відображаються нулем.

• «Вода за тиждень» - стовпчики з підписами обсягів.

• «Динаміка ваги» - поточне, ціль і різниця, а також останній запис.

• «Макронутрієнти за тиждень» - сумарні білки, жири й вуглеводи та їхній розподіл.

6.2. Як інтерпретувати

• Зелені стовпчики вище лінії цілі означають перевищення плану; нижче - недобір.

• Для стабільного результату відстежуйте тренд, а не окремі дні.

Рисунок 4.7 – Екран «Статистика»

Профіль і налаштування (див. рис. 4.8)

7.1. Заповнення даних

• Введіть ім’я, вік, стать, зріст, вагу, оберіть рівень активності та ціль.

• Натисніть «Зберегти профіль». Додаток автоматично розрахує добову норму калорій і цілі КБЖВ.

7.2. Тема оформлення

• Перемикач «Світла тема» змінює тему інтерфейсу; вибір запам’ятовується.

7.3. Про застосунок

• У нижній частині відображаються версія та інформація про проєкт.

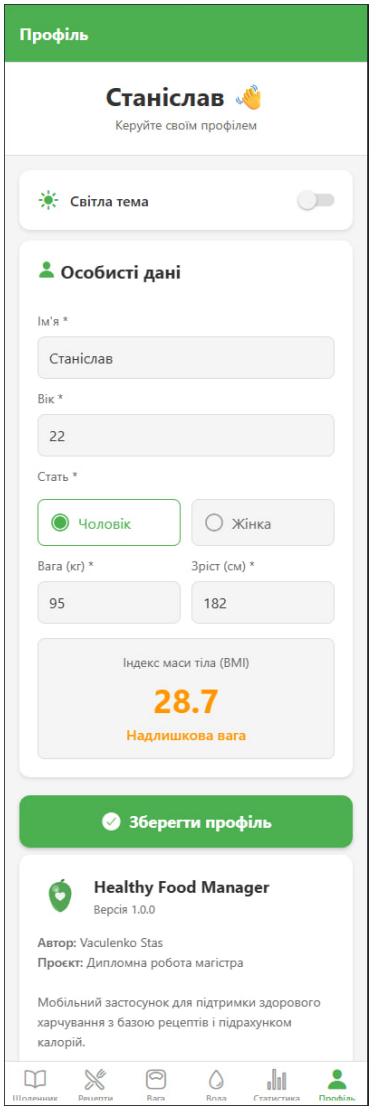

Рисунок 4.8 – Екран «Профіль»

Швидкий старт за 5 кроків

1. Відкрийте «Профіль», заповніть дані та збережіть - з’являться персональні добові цілі.
2. Перейдіть у «Щоденник», оберіть сьогоднішню дату та натисніть «+» у секції «Сніданок».
3. Додайте продукт із бази або рецепт із розділу «Рецепти». Повторіть для інших прийомів їжі.
4. Впродовж дня фіксуйте воду у вкладці «Вода».
5. Увечері відкрийте «Статистика», щоб побачити підсумок дня та тижня.

Healthy Food Manager забезпечує простий, наочний і повністю офлайн-орієнтований контроль харчування: користувач швидко фіксує прийоми їжі з бази продуктів і рецептів, автоматично бачить підрахунок калорій та КБЖВ, відслідковує воду й вагу, а у «Статистиці» отримує зрозумілий тижневий зріз прогресу. Персональні цілі формуються з профілю, інтерфейс інтуїтивний, а дані залишаються на пристрої. У підсумку застосунок допомагає системно виробляти здорові звички без зайвих налаштувань і залежності від мережі.

ВИСНОВКИ

У роботі спроєктовано й реалізовано мобільний застосунок Healthy Food Manager для підтримки здорового харчування з базою продуктів і рецептів, підрахунком калорій і макронутрієнтів, відстеженням води та ваги, а також базовою аналітикою. Запропоноване рішення побудовано за компонентним підходом (React Native + Expo, TypeScript) з локальним збереженням даних (AsyncStorage) і працює офлайн, що робить його придатним для щоденного використання без обов'язкової реєстрації й мережі.

Досягнуті результати:

1. сформульовано вимоги й архітектурну модель, описано потік даних і модулі з чіткими інтерфейсами;
2. спроєктовано схеми даних для щоденника харчування, рецептів, води, ваги й профілю користувача; реалізовано уніфіковані утиліти обчислень (формула Міффліна–Сан Жеора, розподіл КБЖВ 30/30/40, агрегування підсумків);
3. реалізовано шість основних екранів: «Щоденник», «Рецепти», «Вага», «Вода», «Статистика», «Профіль», а також пошук і фільтри в базах;
4. суттєво розширено початкові довідники продуктів і рецептів; передбачено масштабовану структуру для подальшого поповнення;
5. забезпечено офлайн-роботу, відновлення стану та коректність підрахунків на пристрої користувача без передачі персональних даних у мережу;
6. проведено перевірки якості: юніт-тести утиліт обчислень, інтеграційні перевірки сценаріїв додавання/видалення та ручні E2E-нагінання на Android і Web; отримано позитивні результати з повним проходженням автоматизованих тестів і стабільною поведінкою в офлайні;
7. підготовлено інструкцію користувача й демонстраційні матеріали (скріншоти, звіт тестування), що підтверджують досягнення функціональних цілей.

Практична цінність полягає у створенні готового до використання навчального продукту, який поєднує коректні дієтологічні обчислення, простий UX та автономність. Застосунок може використовуватися як персональний трекер, як

базис для навчальних курсів з розробки на React Native, а також як експериментальний полігон для дослідження UX-патернів і локальних моделей зберігання.

Поставлена мета досягнута: спроєктовано, реалізовано та верифіковано багатоплатформний застосунок із заявленим функціоналом. Робота створила технічний і методичний фундамент для масштабування продукту у бік персоналізації, синхронізації та розширеної аналітики харчування.

СПИСОК ІНФОРМАЦІЙНИХ ДЖЕРЕЛ

1. React Native. Documentation. Інтернет-доступ: https://reactnative.dev/docs
2. Expo. Documentation. Інтернет-доступ: https://docs.expo.dev/
3. React. A JavaScript library for building user interfaces. Інтернет-доступ: https://react.dev/
4. TypeScript. Documentation. Інтернет-доступ: https://www.typescriptlang.org/docs/
5. React Navigation v7. Documentation. Інтернет-доступ: https://reactnavigation.org/docs/getting-started
6. AsyncStorage for React Native. Documentation. Інтернет-доступ: https://react-native-async-storage.github.io/async-storage/
7. Jest. Testing Framework. Documentation. Інтернет-доступ: https://jestjs.io/docs/getting-started
8. React Native Testing Library. Documentation. Інтернет-доступ: https://testing-library.com/docs/react-native-testing-library/intro/
9. Detox. Gray Box End-to-End Testing for Mobile Apps. Інтернет-доступ: https://wix.github.io/Detox/
10. react-native-svg. Documentation. Інтернет-доступ: https://github.com/software-mansion/react-native-svg
11. Ionicons. Icon Set. Інтернет-доступ: https://ionic.io/ionicons
12. Axios. Promise based HTTP client. Інтернет-доступ: https://axios-http.com/docs/intro
13. Ольховська О. В. Методичні рекомендації до виконання кваліфікаційної роботи для студентів спеціальності 122 Комп’ютерні науки освітня програма «Комп’ютерні науки» ступеня бакалавра / О. В. Ольховська, О. О. Черненко. - Полтава : ПУЕТ, 2024. -67 с. -1 електрон. опт. диск (CVD-ROM).

ДОДАТОК А.

```
/**
 * DiaryScreen - Щоденник харчування
 * Відображає споживання калорій за день з розбивкою по прийомам їжі
 */

import React, { useState } from 'react';
import {
  View,
  Text,
  StyleSheet,
  ScrollView,
  TouchableOpacity,
  Modal,
  TextInput,
  Alert,
} from 'react-native';
import { Ionicons } from '@expo/vector-icons';
import { useStorage, FoodEntry } from '../context/StorageContext';
import SearchFoodScreen from './SearchFoodScreen';
import SearchRecipeScreen from './SearchRecipeScreen';

export default function DiaryScreen() {
  const { foodEntries, addFoodEntry, deleteFoodEntry, getFoodEntriesByDate, userProfile } =
    useStorage();

  const [selectedDate, setSelectedDate] = useState(new Date().toISOString().split('T')[0]);
  const [modalVisible, setModalVisible] = useState(false);
  const [searchModalVisible, setSearchModalVisible] = useState(false);
  const [recipeModalVisible, setRecipeModalVisible] = useState(false);
  const [datePickerVisible, setDatePickerVisible] = useState(false);
  const [selectedMeal, setSelectedMeal] = useState<FoodEntry['meal']>('breakfast');

  // Форма додавання їжі
  const [foodName, setFoodName] = useState('');
  const [calories, setCalories] = useState('');
  const [protein, setProtein] = useState('');
```

```
const [carbs, setCarbs] = useState('');
const [fat, setFat] = useState('');
const [quantity, setQuantity] = useState('100');

const todayEntries = getFoodEntriesByDate(selectedDate);

// Підрахунок загальних значень за день
const totals = todayEntries.reduce(
  (acc, entry) => ({
    calories: acc.calories + entry.calories,
    protein: acc.protein + entry.protein,
    carbs: acc.carbs + entry.carbs,
    fat: acc.fat + entry.fat,
  }),
  { calories: 0, protein: 0, carbs: 0, fat: 0 }
);

// Підрахунок за кожен прийом їжі
const getMealEntries = (meal: FoodEntry['meal']) =>
  todayEntries.filter(entry => entry.meal === meal);

const getMealTotal = (meal: FoodEntry['meal']) => {
  const entries = getMealEntries(meal);
  return entries.reduce((sum, entry) => sum + entry.calories, 0);
};

// Додати їжу
const handleAddFood = async () => {
  if (!foodName || !calories) {
    Alert.alert('Помилка', 'Введіть назву та калорійність');
    return;
  }

  await addFoodEntry({
    name: foodName,
    calories: Number(calories),
    protein: Number(protein) || 0,
    carbs: Number(carbs) || 0,
```

```
    fat: Number(fat) || 0,
    quantity: Number(quantity),
    meal: selectedMeal,
    date: selectedDate,
  });

  // Очистити форму
  setFoodName('');
  setCalories('');
  setProtein('');
  setCarbs('');
  setFat('');
  setQuantity('100');
  setModalVisible(false);
};

// Відкрити модальне вікно для додавання їжі
const openAddFood = (meal: FoodEntry['meal']) => {
  setSelectedMeal(meal);
  setModalVisible(true);
};

// Відкрити пошук продуктів з бази
const openSearchFood = (meal: FoodEntry['meal']) => {
  setSelectedMeal(meal);
  setSearchModalVisible(true);
};

// Відкрити пошук рецептів
const openSearchRecipe = (meal: FoodEntry['meal']) => {
  setSelectedMeal(meal);
  setRecipeModalVisible(true);
};

// Змінити дату (вперед/назад)
const changeDate = (days: number) => {
  const currentDate = new Date(selectedDate);
  currentDate.setDate(currentDate.getDate() + days);
```

```
  setSelectedDate(currentDate.toISOString().split('T')[0]);
};

// Форматування дати для відображення
const formatDate = (dateString: string) => {
  const date = new Date(dateString);
  const today = new Date().toISOString().split('T')[0];
  const yesterday = new Date();
  yesterday.setDate(yesterday.getDate() - 1);
  const yesterdayStr = yesterday.toISOString().split('T')[0];

  if (dateString === today) return 'Сьогодні';
  if (dateString === yesterdayStr) return 'Вчора';

  return date.toLocaleDateString('uk-UA', {
    day: 'numeric',
    month: 'long',
    year: date.getFullYear() !== new Date().getFullYear() ? 'numeric' : undefined
  });
};

const targetCalories = userProfile?.targetCalories || 2000;
const caloriesPercentage = Math.min((totals.calories / targetCalories) * 100, 100);

const meals: Array<{ key: FoodEntry['meal']; name: string; icon: string }> = [
  { key: 'breakfast', name: 'Сніданок', icon: '
  { key: 'lunch', name: 'Обід', icon: '
  { key: 'dinner', name: 'Вечеря', icon: '  □' },
  { key: 'snack', name: 'Перекус', icon: '
];

// Отримати тиждень для відображення
const getWeekDates = () => {
  const current = new Date(selectedDate);
  const week = [];

  // Отримати понеділок поточного тижня
  const dayOfWeek = current.getDay();
```

```
  const diff = current.getDate() - dayOfWeek + (dayOfWeek === 0 ? -6 : 1);
  const monday = new Date(current.setDate(diff));

  // Згенерувати 7 днів
  for (let i = 0; i < 7; i++) {
    const date = new Date(monday);
    date.setDate(monday.getDate() + i);
    week.push(date);
  }

  return week;
};

const weekDates = getWeekDates();
const today = new Date().toISOString().split('T')[0];

return (
  <ScrollView style={styles.container}>
    {/* Календар тижня */}
    <View style={styles.calendarContainer}>
      <View style={styles.calendarHeader}>
        <TouchableOpacity onPress={() => changeDate(-7)} style={styles.weekButton}>
          <Ionicons name="chevron-back" size={20} color="#4CAF50" />
        </TouchableOpacity>
        <Text style={styles.monthText}>
          {new Date(selectedDate).toLocaleDateString('uk-UA', { month: 'long', year: 'numeric' })}
        </Text>
        <TouchableOpacity
          onPress={() => changeDate(7)}
          style={styles.weekButton}
          disabled={selectedDate >= today}
        >
          <Ionicons
            name="chevron-forward"
            size={20}
            color={selectedDate >= today ? '#ccc' : '#4CAF50'}
          />
        </TouchableOpacity>
```

```
</View>

<View style={styles.weekDays}>
 {weekDates.map((date, index) => {
  const dateStr = date.toISOString().split('T')[0];
  const isSelected = dateStr === selectedDate;
  const isToday = dateStr === today;
  const isFuture = dateStr > today;
  const dayEntries = getFoodEntriesByDate(dateStr);
  const hasData = dayEntries.length > 0;

  return (
   <TouchableOpacity
    key={index}
    style={[
     styles.dayButton,
     isSelected && styles.dayButtonSelected,
     isToday && !isSelected && styles.dayButtonToday,
    ]}
    onPress={() => setSelectedDate(dateStr)}
    disabled={isFuture}
   >
    <Text style={[
     styles.dayName,
     isSelected && styles.dayNameSelected,
     isFuture && styles.dayNameDisabled,
    ]}>
     {date.toLocaleDateString('uk-UA', { weekday: 'short' })}
    </Text>
    <Text style={[
     styles.dayNumber,
     isSelected && styles.dayNumberSelected,
     isFuture && styles.dayNumberDisabled,
    ]}>
     {date.getDate()}
    </Text>
    {hasData && !isSelected && (
     <View style={styles.dataDot} />
```

```
        )}
      </TouchableOpacity>
    );
  })}
 </View>
</View>

{/* Загальна статистика */}
<View style={styles.summaryCard}>
 <Text style={styles.summaryTitle}>Калорії за день</Text>
 <Text style={styles.summaryCalories}>
  {totals.calories} / {targetCalories} ккал
 </Text>

 {/* Прогрес бар */}
 <View style={styles.progressBarContainer}>
  <View style={[styles.progressBar, { width: `${caloriesPercentage}%` }]} />
 </View>

 {/* Макроси */}
 <View style={styles.macrosContainer}>
  <View style={styles.macroItem}>
   <Text style={styles.macroLabel}>Білки</Text>
   <Text style={styles.macroValue}>{totals.protein.toFixed(1)}г</Text>
  </View>
  <View style={styles.macroItem}>
   <Text style={styles.macroLabel}>Жири</Text>
   <Text style={styles.macroValue}>{totals.fat.toFixed(1)}г</Text>
  </View>
  <View style={styles.macroItem}>
   <Text style={styles.macroLabel}>Вуглеводи</Text>
   <Text style={styles.macroValue}>{totals.carbs.toFixed(1)}г</Text>
  </View>
 </View>
</View>

{/* Прийоми їжі */}
{meals.map(meal => (
```

```
<View key={meal.key} style={styles.mealCard}>
 <View style={styles.mealHeader}>
  <View style={styles.mealTitleRow}>
   <Text style={styles.mealIcon}>{meal.icon}</Text>
   <Text style={styles.mealTitle}>{meal.name}</Text>
   <Text style={styles.mealCalories}>{getMealTotal(meal.key)} ккал</Text>
  </View>
  <View style={styles.mealActions}>
   <TouchableOpacity
    onPress={() => openSearchFood(meal.key)}
    style={styles.actionButton}
    hitSlop={{ top: 10, bottom: 10, left: 10, right: 10 }}
   >
    <Ionicons name="nutrition" size={22} color="#4CAF50" />
   </TouchableOpacity>
   <TouchableOpacity
    onPress={() => openSearchRecipe(meal.key)}
    style={styles.actionButton}
    hitSlop={{ top: 10, bottom: 10, left: 10, right: 10 }}
   >
    <Ionicons name="restaurant" size={22} color="#FF9800" />
   </TouchableOpacity>
   <TouchableOpacity
    onPress={() => openAddFood(meal.key)}
    style={styles.actionButton}
    hitSlop={{ top: 10, bottom: 10, left: 10, right: 10 }}
   >
    <Ionicons name="add-circle" size={26} color="#666" />
   </TouchableOpacity>
  </View>
 </View>

 {/* Список продуктів */}
 {getMealEntries(meal.key).map(entry => (
  <View key={entry.id} style={styles.foodItem}>
   <View style={styles.foodInfo}>
    <Text style={styles.foodName}>{entry.name}</Text>
    <Text style={styles.foodDetails}>
```

```
            {entry.quantity}г • {entry.calories} ккал
          </Text>
        </View>
        <TouchableOpacity onPress={() => deleteFoodEntry(entry.id)}>
          <Ionicons name="trash-outline" size={20} color="#f44336" />
        </TouchableOpacity>
      </View>
    ))}
  </View>
))}

{/* Модальне вікно додавання їжі */}
<Modal
  animationType="slide"
  transparent={true}
  visible={modalVisible}
  onRequestClose={() => setModalVisible(false)}
>
  <View style={styles.modalOverlay}>
    <View style={styles.modalContent}>
      <View style={styles.modalHeader}>
        <Text style={styles.modalTitle}>Додати продукт</Text>
        <TouchableOpacity onPress={() => setModalVisible(false)}>
          <Ionicons name="close" size={28} color="#333" />
        </TouchableOpacity>
      </View>

      <ScrollView>
        <Text style={styles.inputLabel}>Назва продукту</Text>
        <TextInput
          style={styles.input}
          value={foodName}
          onChangeText={setFoodName}
          placeholder="Наприклад: Яблуко"
        />

        <Text style={styles.inputLabel}>Вага (г)</Text>
        <TextInput
```

```
  style={styles.input}
  value={quantity}
  onChangeText={setQuantity}
  keyboardType="numeric"
  placeholder="100"
/>

<Text style={styles.inputLabel}>Калорії (ккал)</Text>
<TextInput
  style={styles.input}
  value={calories}
  onChangeText={setCalories}
  keyboardType="numeric"
  placeholder="52"
/>

<Text style={styles.inputLabel}>Білки (г)</Text>
<TextInput
  style={styles.input}
  value={protein}
  onChangeText={setProtein}
  keyboardType="numeric"
  placeholder="0.3"
/>

<Text style={styles.inputLabel}>Жири (г)</Text>
<TextInput
  style={styles.input}
  value={fat}
  onChangeText={setFat}
  keyboardType="numeric"
  placeholder="0.2"
/>

<Text style={styles.inputLabel}>Вуглеводи (г)</Text>
<TextInput
  style={styles.input}
  value={carbs}
```

```
            onChangeText={setCarbs}
            keyboardType="numeric"
            placeholder="14"
          />

          <TouchableOpacity style={styles.addButton} onPress={handleAddFood}>
            <Text style={styles.addButtonText}>Додати</Text>
          </TouchableOpacity>
        </ScrollView>
      </View>
    </View>
  </Modal>

  {/* Пошук продуктів з бази */}
  <SearchFoodScreen
    visible={searchModalVisible}
    onClose={() => setSearchModalVisible(false)}
    selectedMeal={selectedMeal}
    selectedDate={selectedDate}
  />

  {/* Пошук рецептів */}
  <SearchRecipeScreen
    visible={recipeModalVisible}
    onClose={() => setRecipeModalVisible(false)}
    selectedMeal={selectedMeal}
    selectedDate={selectedDate}
  />
 </ScrollView>
 );
}

const styles = StyleSheet.create({
 container: {
  flex: 1,
  backgroundColor: '#f5f5f5',
 },
 calendarContainer: {
```

```
  backgroundColor: '#fff',
  marginHorizontal: 16,
  marginTop: 16,
  marginBottom: 8,
  borderRadius: 12,
  padding: 16,
  shadowColor: '#000',
  shadowOffset: { width: 0, height: 2 },
  shadowOpacity: 0.1,
  shadowRadius: 4,
  elevation: 3,
},
calendarHeader: {
  flexDirection: 'row',
  justifyContent: 'space-between',
  alignItems: 'center',
  marginBottom: 16,
},
weekButton: {
  padding: 8,
},
monthText: {
  fontSize: 16,
  fontWeight: '600',
  color: '#333',
  textTransform: 'capitalize',
},
weekDays: {
  flexDirection: 'row',
  justifyContent: 'space-between',
},
dayButton: {
  alignItems: 'center',
  paddingVertical: 12,
  paddingHorizontal: 8,
  borderRadius: 12,
  minWidth: 45,
},
```

```
dayButtonSelected: {
  backgroundColor: '#4CAF50',
},
dayButtonToday: {
  borderWidth: 2,
  borderColor: '#4CAF50',
},
dayName: {
  fontSize: 11,
  color: '#999',
  marginBottom: 4,
  textTransform: 'capitalize',
},
dayNameSelected: {
  color: '#fff',
  fontWeight: '600',
},
dayNameDisabled: {
  color: '#ddd',
},
dayNumber: {
  fontSize: 16,
  fontWeight: '600',
  color: '#333',
},
dayNumberSelected: {
  color: '#fff',
},
dayNumberDisabled: {
  color: '#ddd',
},
dataDot: {
  width: 4,
  height: 4,
  borderRadius: 2,
  backgroundColor: '#4CAF50',
  marginTop: 4,
},
```

```
summaryCard: {
  backgroundColor: '#fff',
  padding: 20,
  margin: 16,
  marginBottom: 8,
  borderRadius: 12,
  shadowColor: '#000',
  shadowOffset: { width: 0, height: 2 },
  shadowOpacity: 0.1,
  shadowRadius: 4,
  elevation: 3,
},
summaryTitle: {
  fontSize: 16,
  color: '#666',
  marginBottom: 8,
},
summaryCalories: {
  fontSize: 32,
  fontWeight: 'bold',
  color: '#333',
  marginBottom: 16,
},
progressBarContainer: {
  height: 8,
  backgroundColor: '#e0e0e0',
  borderRadius: 4,
  overflow: 'hidden',
  marginBottom: 20,
},
progressBar: {
  height: '100%',
  backgroundColor: '#4CAF50',
},
macrosContainer: {
  flexDirection: 'row',
  justifyContent: 'space-around',
},
```

```
macroItem: {
  alignItems: 'center',
},
macroLabel: {
  fontSize: 12,
  color: '#666',
  marginBottom: 4,
},
macroValue: {
  fontSize: 18,
  fontWeight: '600',
  color: '#333',
},
mealCard: {
  backgroundColor: '#fff',
  padding: 16,
  marginHorizontal: 16,
  marginBottom: 8,
  borderRadius: 12,
  shadowColor: '#000',
  shadowOffset: { width: 0, height: 1 },
  shadowOpacity: 0.05,
  shadowRadius: 2,
  elevation: 2,
},
mealHeader: {
  flexDirection: 'row',
  justifyContent: 'space-between',
  alignItems: 'center',
  marginBottom: 12,
},
mealTitleRow: {
  flexDirection: 'row',
  alignItems: 'center',
  flex: 1,
},
mealIcon: {
  fontSize: 24,
```

```
  marginRight: 8,
},
mealTitle: {
  fontSize: 18,
  fontWeight: '600',
  color: '#333',
  flex: 1,
},
mealCalories: {
  fontSize: 14,
  color: '#666',
  marginRight: 12,
},
mealActions: {
  flexDirection: 'row',
  alignItems: 'center',
  gap: 8,
},
actionButton: {
  padding: 4,
},
foodItem: {
  flexDirection: 'row',
  justifyContent: 'space-between',
  alignItems: 'center',
  paddingVertical: 12,
  borderTopWidth: 1,
  borderTopColor: '#f0f0f0',
},
foodInfo: {
  flex: 1,
},
foodName: {
  fontSize: 16,
  color: '#333',
  marginBottom: 4,
},
foodDetails: {
```

```
  fontSize: 14,
  color: '#999',
},
modalOverlay: {
  flex: 1,
  backgroundColor: 'rgba(0,0,0,0.5)',
  justifyContent: 'flex-end',
},
modalContent: {
  backgroundColor: '#fff',
  borderTopLeftRadius: 20,
  borderTopRightRadius: 20,
  padding: 20,
  maxHeight: '80%',
},
modalHeader: {
  flexDirection: 'row',
  justifyContent: 'space-between',
  alignItems: 'center',
  marginBottom: 20,
},
modalTitle: {
  fontSize: 20,
  fontWeight: 'bold',
  color: '#333',
},
inputLabel: {
  fontSize: 14,
  color: '#666',
  marginBottom: 8,
  marginTop: 12,
},
input: {
  borderWidth: 1,
  borderColor: '#ddd',
  borderRadius: 8,
  padding: 12,
  fontSize: 16,
```

```
  },
  addButton: {
    backgroundColor: '#4CAF50',
    padding: 16,
    borderRadius: 8,
    alignItems: 'center',
    marginTop: 20,
    marginBottom: 20,
  },
  addButtonText: {
    color: '#fff',
    fontSize: 16,
    fontWeight: '600',
  },
});
```