Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

Бакалаврська кваліфікаційна робота

на тему: «Розробка мобільного застосунку для моніторингу збалансованого харчування»

Виконав: студент 4 курсу
групи 2ПІ-21б спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Трач Вячеслав Олександрович
(прізвище та ініціали)
Керівник: к.т.н., доц. каф. ПЗ Ткаченко О. М.
(прізвище та ініціали)
Рецензент: к.т.н., доц. каф. ОТ Кадук О. В.
(прізвище та ініціали)

Допущено до захисту
Завідувач кафедри ПЗ
д.т.н., проф. Романюк О.Н.
«09» 06 2025 р.

ВНТУ – 2025

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

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О.Н._____
« 21» березня 2025 р.

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

Трачу Вячеславу Олександровичу

1. Тема роботи – «Розробка мобільного застосунку для моніторингу збалансованого харчування".»

Керівник роботи: Ткаченко Олександр Миколайович, к.т.н., доц. кафедри ПЗ, затверджені наказом вищого навчального закладу від 20 березня 2025 р. № 97.

2. Строк подання студентом роботи 2 червня 2025 р.

3. Вихідні дані до роботи: мова програмування Kotlin, середовище розробки Android Studio, мова розмітки XML.

4. Зміст розрахунково-пояснювальної записки: вступ; аналіз стану розвитку застосунків для моніторингу збалансованого харчування; розробка моделі та алгоритмів застосунку; розробка застосунку; тестування застосунку; висновки; список використаних джерел; додатки.

5. Перелік графічного матеріалу: титульний слайд; актуальність теми; мета, об'єкт та предмет дослідження; задачі дослідження, наукова новизна, практична цінність одержаних результатів; порівняльний аналіз аналогів.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада Консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Ткаченко О.М., к.т.н., доц. кафедри ПЗ	24.03.2025	

7. Дата видачі завдання 24 березня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи	Примітки
1	Аналіз стану розвитку застосунків для моніторингу збалансованого харчування	25.03.25 – 31.03.25	Вик.
2	Розробка моделі та алгоритмів застосунку	01.04.25 - 18.04.25	вик.
3	Варіантний аналіз та вибір мови програмування	19.04.25 - 06.05.25	вик.
4	Розробка застосунку	07.05.25 - 24.05.25	вик.
5	Тестування застосунку	25.05.25 - 30.05.25	вик.
6	Оформлення матеріалів до захисту БКР	25.03.25 – 31.03.25	вик.

Студент (підпис) Трач В.О. (прізвище та ініціали)

Керівник бакалаврської кваліфікаційної роботи (підпис) Ткаченко О.М. (прізвище та ініціали)

АНОТАЦІЯ

УДК 004.912.032.26

Трач В. О. Розробка мобільного застосунку для моніторингу збалансованого харчування: бакалаврська кваліфікаційна робота зі спеціальності 121 – інженерія програмного забезпечення, освітня програма – інженерія програмного забезпечення. – Вінниця: ВНТУ, 2025. – 87 с.

На укр. мові. Бібліогр.: 25 назв; рис.: 22; табл.: 3.

У бакалаврській кваліфікаційній роботі розроблено мобільний застосунок для моніторингу харчування та формування персоналізованих рекомендацій щодо збалансованого раціону. Реалізовано програмні модулі для ведення обліку КБЖВ (калорій, білків, жирів, вуглеводів), візуалізації добового балансу нутрієнтів, аналізу харчової збалансованості раціону та генерації персоналізованих рекомендацій.

Програмний продукт орієнтований на широке коло користувачів, які прагнуть підтримувати здоровий спосіб життя. Він може застосовуватись у сфері персонального фітнесу, дієтології, а також для самоконтролю харчування.

Застосунок реалізовано мовою програмування Kotlin у середовищі Android Studio з використанням XML розмітки. Інтерфейс адаптований до потреб користувача, підтримує офлайн-режим роботи.

ANNOTATION

UDC 004.912.032.26

Trach V.O. Development of a Mobile Application for Balanced Nutrition Monitoring: Bachelor's Qualification Thesis in Specialty 121 – Software Engineering, Educational Program – Software Engineering. – Vinnytsia: VNTU, 2025. – 87 p.

In Ukrainian. References: 25 sources; figures: 22; tables: 3.

This bachelor’s qualification thesis presents the development of a mobile application for nutrition monitoring and the generation of personalized recommendations for a balanced diet. The implemented software modules enable tracking of macronutrients (calories, proteins, fats, carbohydrates), visualization of daily nutrient balance, analysis of dietary adequacy, and generation of individualized suggestions.

The software product is intended for a wide range of users aiming to maintain a healthy lifestyle. It can be applied in personal fitness, diet planning, and nutritional self-control.

The application is developed using the Kotlin programming language in the Android Studio environment with XML markup. The user interface is tailored to user needs and supports offline operation.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження.

Розвиток цифрових технологій та широке розповсюдження мобільних пристроїв кардинально змінюють підходи до підтримання здорового способу життя. У сучасному суспільстві зростає усвідомлення важливості збалансованого харчування для профілактики захворювань, підтримання фізичної форми та загального добробуту. У зв’язку з цим мобільні застосунки для моніторингу харчування й оцінки раціону набувають все більшої популярності, оскільки дозволяють користувачам отримувати швидкий, доступний і персоналізований інструмент для контролю споживання калорій, білків, жирів і вуглеводів (КБЖВ), формування раціону та досягнення дієтичних цілей.

Попри суттєвий прогрес у цій галузі, критичний аналіз існуючих мобільних застосунків виявляє низку значних недоліків. Більшість доступних на ринку рішень надають лише базові можливості, такі як підрахунок калорій або відстеження ваги. При цьому не всі застосунки здатні враховувати індивідуальні особливості користувача – вік, стать, рівень фізичної активності, харчові вподобання, алергії, а також динамічні зміни у способі життя чи стані здоров’я. Через це користувачі змушені використовувати декілька програм одночасно, що ускладнює процес моніторингу та знижує мотивацію до регулярного ведення харчового щоденника.

Ще однією проблемою є відсутність глибокого аналізу харчових звичок і відсутність ефективного модуля персоналізованих рекомендацій. Багато програм не здатні формувати корисні поради щодо поліпшення харчування на основі зібраних даних, або ж ці поради мають загальний характер і не враховують поточний стан КБЖВ-балансу користувача. Крім того, деякі програми мають перевантажений або незручний інтерфейс, що не сприяє щоденному використанню, особливо серед початківців або людей з обмеженим досвідом користування цифровими технологіями.

Також актуальним залишається питання конфіденційності даних про

харчування, медичні параметри та спосіб життя. Не всі розробники впроваджують сучасні засоби захисту персональної інформації, що може створити ризики витоку конфіденційних відомостей.

Отже, актуальною є розробка зручного, функціонального та персоналізованого мобільного застосунку для моніторингу харчування та формування рекомендацій щодо збалансованого раціону.

Зв'язок роботи з науковими програмами, планами, темами.

Робота виконувалася згідно з планом виконання наукових досліджень на кафедрі програмного забезпечення.

Мета та завдання дослідження.

Метою роботи є підвищення якості процесу контролю та оптимізації особистого раціону, що дозволяє відстежувати споживання КБЖВ і формувати персоналізовані рекомендації щодо харчування за допомогою розробки програмного застосунку для моніторингу збалансованого харчування.

Основними задачами дослідження є:

−розробити модель застосунку для моніторингу збалансованого харчування;

−розробити метод оцінки харчової збалансованості раціону;

−розробити алгоритми роботи застосунку;

−розробити модулі застосунку для моніторингу збалансованого харчування;

−провести тестування розробленого застосунку.

Об'єкт дослідження – процес розробки засобів мобільної системи для моніторингу харчування.

Предмет дослідження – методи та засоби розробки програмного забезпечення для моніторингу харчування й формування раціональних рекомендацій.

Методи дослідження.

У процесі дослідження використовувались: методи аналізу літературних джерел та сучасних підходів до формування збалансованого раціону; методи

моделювання та проєктування програмних систем для розробки програмного забезпечення; методи тестування та верифікації розроблених програмних модулів мобільного застосунку для перевірки його працездатності; статистичні методи для аналізу результатів експериментального використання функцій моніторингу КБЖВ та рекомендацій.

Наукова новизна отриманих результатів.

Вперше запропоновано архітектуру застосунку, особливістю якої є вбудована підтримка генерації персоналізованих рекомендацій на основі щоденного аналізу, що дозволило забезпечити оперативне формування рекомендацій щодо збалансованого харчування.

Отримав подальшого розвитку метод оцінки харчової збалансованості раціону, який відрізняється від існуючих використанням індивідуалізованих норм і градацією відхилень із формуванням персональних порад, що дозволило підвищити точність оцінки відповідності раціону добовим нормам КБЖВ.

Практична цінність отриманих результатів.

Розроблений застосунок може бути використаний широким колом користувачів для щоденного моніторингу харчування, контролю споживання поживних речовин і досягнення дієтичних або спортивних цілей.

Особистий внесок здобувача.

Усі наукові результати, викладені у бакалаврській кваліфікаційній роботі, отримані автором особисто. В роботі [1], опублікованій у співавторстві, автору належить розробка архітектури мобільного застосунку.

Апробація матеріалів бакалаврської кваліфікаційної роботи. Результати роботи було представлено на Міжнародній науково-практичній інтернет-конференції «Молодь в науці: дослідження, проблеми, перспективи (МН-2025)».

Публікації. Результати роботи опубліковано в матеріалах Міжнародної науково-практичної інтернет-конференції «Молодь в науці: дослідження, проблеми, перспективи (МН-2025)» [1].

1 АНАЛІЗ СТАНУ РОЗВИТКУ ЗАСТОСУНКІВ ДЛЯ МОНІТОРИНГУ ЗБАЛАНСОВАНОГО ХАРЧУВАННЯ

1.1 Аналіз стану питання актуальності розробки

Питання здорового харчування стає все більш актуальним, адже від якості раціону напряму залежить фізичний стан, самопочуття, рівень енергії та профілактика поширених хронічних захворювань. У сучасному світі, де значна частина населення веде малорухомий спосіб життя та має нерегулярний режим харчування, зростає потреба в інструментах, які дозволяють контролювати споживання їжі та аналізувати її вплив на організм. Мобільні застосунки стали одним із найзручніших способів такого моніторингу завдяки доступності, простоті використання та широким функціональним можливостям[2, 3].

Наразі існує чимало популярних застосунків для ведення харчового щоденника – зокрема, MyFitnessPal, YAZIO, Lifesum, FatSecret [4, 5, 6]. Вони дозволяють фіксувати спожиті продукти, підраховувати калорії та нутрієнти, переглядати статистику та слідкувати за прогресом. Проте більшість таких рішень мають суттєві недоліки: обмежений безкоштовний функціонал, складні або перевантажені інтерфейси, відсутність україномовної локалізації, а також недостатнє урахування місцевих кулінарних традицій та звичок. Зокрема, бази даних продуктів не включають типові українські страви, а рекомендації часто є шаблонними та не персоналізованими.

Також варто зазначити, що не всі аналоги передбачають гнучку систему адаптації під конкретного користувача – з урахуванням віку, статі, рівня активності чи цілей. Це суттєво знижує ефективність таких застосунків, адже користувачам доводиться або ігнорувати ці функції, або шукати сторонні джерела для налаштування. Водночас у багатьох з них відсутні прості й зрозумілі візуалізації, а підтримка щоденного введення даних ускладнюється через надмірну кількість опцій чи незрозумілий інтерфейс.

У зв’язку з цим виникає потреба у створенні нового мобільного застосунку, який би враховував вказані недоліки. Запропонована в межах кваліфікаційної

роботи розробка вирізняється адаптованістю до українських реалій, зручним інтерфейсом, підтримкою національних страв, а також персоналізованим підходом до оцінки харчування. Це дозволяє зробити моніторинг раціону доступним широкому колу користувачів та забезпечити інструмент, який справді сприяє формуванню здорових звичок.

1.2 Порівняльний аналіз аналогів

На ринку мобільних застосунків для контролю харчування існує низка готових рішень, які активно використовуються у повсякденному житті користувачів. Найпопулярнішими серед них є MyFitnessPal, YAZIO та Lifesum. Ці застосунки дозволяють вести облік спожитих калорій і нутрієнтів, фіксувати фізичну активність, відстежувати прогрес у досягненні цілей щодо маси тіла. Користувачі мають змогу формувати харчові звички, будувати плани харчування, переглядати статистику за день, тиждень чи місяць. Також передбачено зручні механізми сканування штрихкодів, синхронізацію з фітнес-трекерами, створення власних рецептів або копіювання прийомів їжі.

Окрім базових функцій, деякі програми пропонують додаткові можливості: автоматичне визначення рекомендованих добових норм, індивідуальні підказки щодо дієти, нагадування про прийоми їжі або пиття води. У розширених версіях можуть бути доступні персоналізовані плани схуднення або набору ваги, розроблені з урахуванням статі, віку та стилю життя. Таким чином, дані застосунки охоплюють широкий спектр потреб користувача – від простого підрахунку калорій до комплексного управління способом життя.

Проте, попри функціональність і багатий досвід команд розробників, більшість популярних рішень є закордонними, що тягне за собою обмеження для україномовної аудиторії. В інтерфейсах відсутній переклад українською, а у базах продуктів – характерні локальні страви. Також часто функції персоналізації обмежені або доступні лише за підпискою, що не є зручним для всіх категорій користувачів.

MyFitnessPal є одним з найстаріших та наймасовіших сервісів. Він містить велику базу продуктів, інтеграцію з фітнес-трекерами та аналітику. Недоліками є перевантажений інтерфейс і платний доступ до розширених функцій. Таким чином користувачу може бути складно ефективно та швидко користуватись додатком. Інтерфейс застосунку наведено на рисунку 1.1.

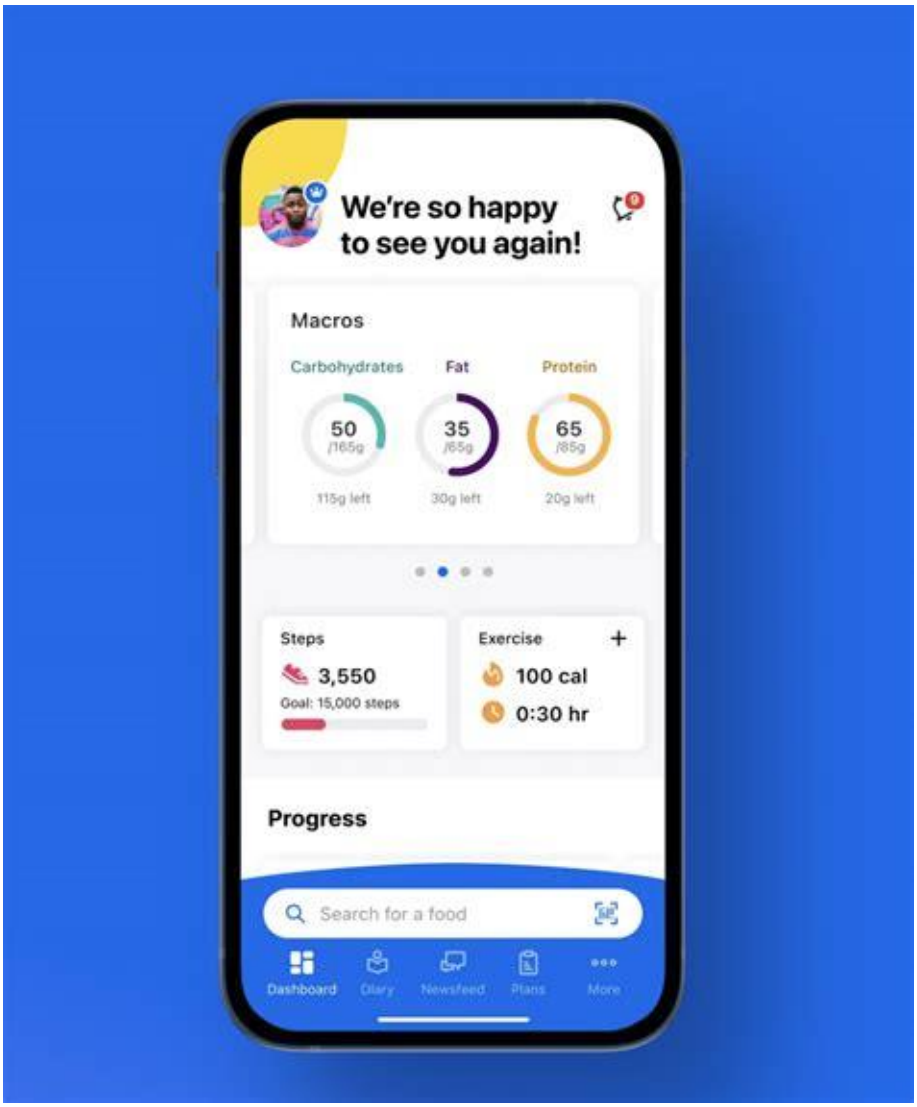


Рисунок 1.1 – Інтерфейс MyFitnessPal

YAZIO позиціонує себе як зручний додаток для контролю харчування та планування дієт. Має зручний інтерфейс і сучасний дизайн, однак вільна версія обмежена лише базовими функціями. Інтерфейс YAZIO зображено на рисунку 1.2.

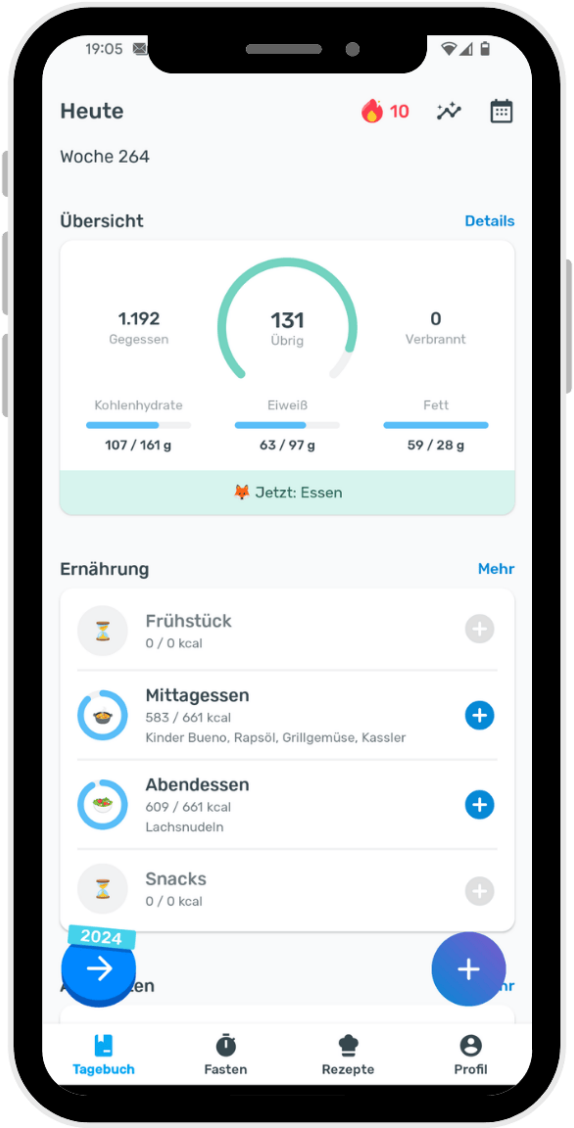


Рисунок 1.2 – Інтерфейс YAZIO з планами харчування та рецептами.

Lifesum дозволяє обирати дієтичні плани, оцінює якість прийомів їжі за допомогою шкали “харчового балансу”, проте значна частина його функцій доступна лише у преміум-версії, що і є великим мінусом цього застосунку. Інтерфейс додатку зображено на рисунку 1.3.

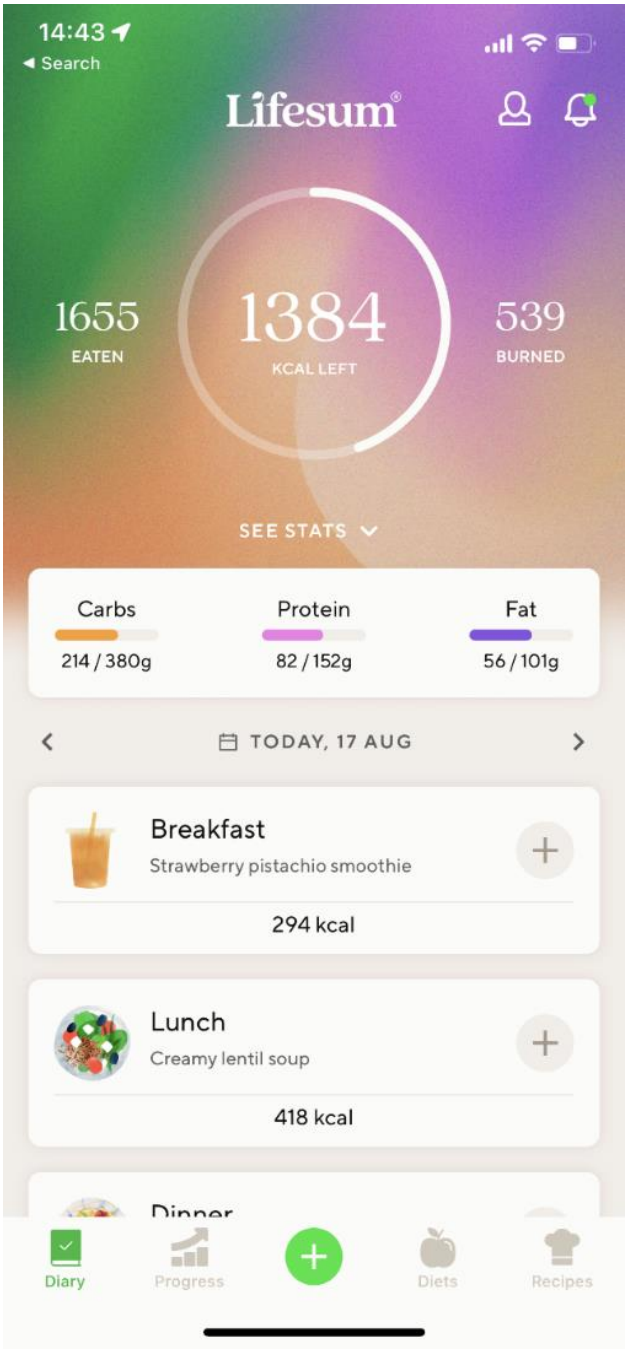


Рисунок 1.3 – Головна панель Lifesum з відображенням спожитих калорій та макроелементів.

З метою обґрунтування актуальності розробки нового мобільного застосунку, здійснено порівняння основного функціоналу існуючих рішень із функціональністю, яка передбачена у власній розробці.

Порівняння описано у таблиці 1.1.

Таблиця 1.1 – Порівняльний аналіз аналогів

Функція / Застосунок	MyFitness Pal	YAZIO	Lifesum	Власна розробка
Підрахунок КБЖВ	+	+	+	+
База локальних продуктів	–	–	–	+
Персональні рекомендації	–	–	–	+
Інтеграція з фітнес-трекером	+	+	+	–
Візуалізація статистики	+	+	+	+
Безкоштовна повна версія	–	–	–	+
Сумарний бал (з 6)	3	3	3	5

Таким чином, порівняльний аналіз показав, що жоден із наявних застосунків не задовольняє повною мірою потреби користувача у доступності повного функціоналу та гнучкої системи персональних рекомендацій. Саме тому розробка власного мобільного застосунку є доцільною.

1.3 Аналіз методів розв’язання задачі

Процес контролю харчування за допомогою мобільного застосунку включає кілька ключових етапів: введення користувачем даних про спожиті продукти, обчислення їхньої харчової цінності, порівняння з індивідуальними нормами, аналіз тенденцій та формування рекомендацій. Для кожного з цих етапів використовуються відповідні методи та алгоритми, ефективність яких визначає загальну якість застосунку.

Найпоширенішим способом введення є ручне заповнення щоденника харчування, де користувач обирає продукт із бази даних та вказує його кількість. Для зручності часто застосовуються поля автозаповнення, а також можливість збереження “улюблених” страв. Більш просунуті рішення реалізують сканування штрих-коду або розпізнавання продуктів за фото, однак такі функції значно ускладнюють розробку та збільшують обсяг ресурсів. У межах цієї роботи буде

використано комбінований підхід: база продуктів + ручне введення.

Дані, отримані від користувача, обробляються алгоритмами підрахунку КБЖВ – загальноприйнята методика, згідно з якою кожен грам білків і вуглеводів дорівнює 4 ккал, а грам жирів – 9 ккал. Це дозволяє точно розраховувати калорійність кожного прийому їжі та загальний добовий баланс.

Щоб визначити, наскільки раціон є збалансованим, використовуються нормовані дієтологічні рекомендації (наприклад, дані ВООЗ або локальні стандарти). Порівняння фактичного споживання з рекомендованими значеннями відбувається за допомогою відносного відхилення показаного на формулі 1.1 [7].

$$\text{Відхилення} = \frac{\text{Факт} - \text{Норма}}{\text{Норма}} \times 100\% \tag{1.1}$$

де

Факт – фактична кількість спожитого нутрієнта,

Норма – рекомендована добова норма нутрієнта,

Відхилення – відсоткове відхилення фактичного споживання від норми.

Значні відхилення підсвічуються або зазначаються у звіті. Також можуть використовуватися умовні “оцінки балансу” – наприклад, 0–25% відхилення вважається нормою, 25–50% – помірним відхиленням, понад 50% – критичним.

Найбільш актуальним напрямом є персоналізовані рекомендації – не загальні поради, а дії, які враховують тип харчування, цілі користувача (схуднення, набір ваги, підтримка), його фізичну активність, стать, вік. У простій реалізації система формує повідомлення на кшталт:

– “Ви перевищили норму жирів на 35%. Спробуйте замінити смажені страви на тушковані або відварені.”

– “Вчора спожито на 30% менше білка, ніж потрібно. Додайте більше білкових продуктів сьогодні.”

На додатковому рівні реалізується аналіз харчових звичок – наприклад,

виявлення системних відхилень у певні дні тижня або вночі, що дозволяє надавати рекомендації для формування стабільного режиму харчування.

Для відображення інформації користувачеві застосовуються графіки (лінійні або кругові), індикатори, таблиці з денними/тижневими підсумками, кольорові сигнали (зелений – в нормі, червоний – перевищено тощо). Ці засоби допомагають швидко оцінити ситуацію без потреби вчитуватись у цифри.

Таким чином, сукупність методів, обраних для реалізації у застосунку, дозволяє забезпечити наочний, персоналізований і доступний підхід до контролю харчування користувача у мобільному середовищі.

1.4 Постановка задач

Згідно з індивідуальним завданням, основною метою бакалаврської кваліфікаційної роботи є створення мобільного застосунку, який дозволяє здійснювати моніторинг харчування користувача, вести облік КБЖВ (калорій, білків, жирів, вуглеводів) та формувати персоналізовані рекомендації щодо збалансованого раціону. Розроблений програмний продукт повинен бути зручним у використанні, інтуїтивно зрозумілим для широкого кола користувачів та орієнтованим на покращення щоденних харчових звичок.

Додаток має не лише збирати інформацію про спожиті продукти, але й аналізувати відхилення від рекомендованих норм, враховуючи фізичну активність, цілі користувача (схуднення, набір маси, підтримка форми) та інші індивідуальні особливості. Крім того, система повинна забезпечити збереження історії харчування, візуалізацію змін у раціоні у вигляді графіків, і своєчасне формування порад, що сприяють поліпшенню якості харчування. Застосунок має функціонувати стабільно на Android-пристроях, мати локалізацію українською мовою та підтримувати офлайн-доступ до бази продуктів.

Для досягнення поставленої мети необхідно вирішити такі задачі:

- розробити модель застосунку для моніторингу харчування;
- розробити метод оцінки харчової збалансованості раціону;
- розробити алгоритми обліку КБЖВ та генерації рекомендацій;

- створити модулі застосунку для введення харчових даних та перегляду статистики;
- провести тестування розробленого застосунку.
- Технічне завдання на розробку наведено в додатку А.

1.5 Висновки

У розділі було проаналізовано сучасний стан розробки мобільних застосунків для моніторингу харчування. Встановлено, що більшість з них мають обмеження у безкоштовному доступі, відсутність персоналізованих рекомендацій, складний інтерфейс та слабку локалізацію, що зумовлює потребу у новому рішенні.

На основі аналізу аналогів визначено ключовий функціонал, який необхідно реалізувати: облік КБЖВ, оцінка збалансованості раціону, генерація рекомендацій та візуалізація даних. Обрано методи обробки інформації, побудови графіків і підрахунків, що відповідають прикладному спрямуванню роботи.

Сформульовано чіткі задачі дослідження, реалізація яких дозволить створити ефективний мобільний застосунок для щоденного контролю харчування з персоналізованим підходом.

2 РОЗРОБКА МОДЕЛІ ТА АЛГОРИТМІВ РОБОТИ

2.1 Аналіз інформаційного забезпечення

У цьому підрозділі розглядається інформаційне забезпечення мобільного застосунку для моніторингу збалансованого харчування. Розглянута система орієнтована на користувача, що прагне слідкувати за добовим споживанням основних нутрієнтів (калорій, білків, жирів і вуглеводів), формувати корисні харчові звички та досягати певних цілей (схуднення, набір маси, підтримка ваги).

Інформаційне забезпечення є сукупністю структурованих даних, які використовуються застосунком для зберігання, обробки та аналізу інформації про харчування. До основних інформаційних об'єктів системи належать:

User – клас користувача, який зберігає ім’я, електронну пошту, пароль, зріст, вагу, стать, вік і ціль (наприклад, зниження чи підтримка ваги). Ці дані використовуються для розрахунку індивідуальної потреби в калоріях та персоналізації рекомендацій.

Food – модель продукту харчування. Містить поля: назву продукту, калорійність, а також кількість білків, жирів і вуглеводів на 100 грамів. Користувач додає ці продукти вручну через інтерфейс.

TodayEatingHistory – структура, що відображає усі записи про спожиті продукти за поточний день. Вона використовується для обчислення сумарної кількості спожитих калорій за день.

EatingHistory – база записів за попередні дні, яка зберігає хронологію прийомів їжі. Дані групуються за датами та використовуються для формування звітів та графіків змін раціону з часом.

CalorieDiaries – агрегований клас, який узагальнює інформацію про дату, калорії, кількість спожитих продуктів та вміст нутрієнтів. Служить для побудови загальної історії харчування користувача.

Зберігання даних реалізовано у вигляді SQLite-бази за допомогою класу DBHelper.kt. База містить таблиці user, food, calorie_diary, eating_history. Створення, оновлення та зчитування даних здійснюється методами класу

DBHelper, які оперують SQL-запитами на низькому рівні.

Користувач вводить дані вручну через зручний інтерфейс: додає нові продукти, фіксує прийоми їжі, переглядає статистику. На основі даних про поточну та минулу активність система формує добовий баланс КБЖВ та відображає результат у графічному вигляді. Валідація введення забезпечується перевірками у формі: наприклад, кількість калорій повинна бути додатною, а поля не повинні залишатись порожніми.

Таким чином, інформаційне забезпечення застосунку охоплює зберігання профілю користувача, продуктів, історії прийомів їжі, а також статистики добового споживання. Його реалізація через класичну SQLite-модель є ефективною для офлайн-режиму, а структура застосунку дозволяє легко масштабувати функціонал у майбутньому (додавання рекомендацій, API, локалізованих продуктів тощо

2.2 Розробка архітектури мобільного застосунку

Модель застосунку визначає загальну структуру взаємодії між основними функціональними компонентами, що забезпечують моніторинг харчування, облік нутрієнтів, зберігання даних та виведення персоналізованих результатів для користувача. Архітектура реалізована у вигляді багаторівневої структури, де розмежовано обробку даних, логіку додатку та взаємодію з інтерфейсом.

Модель застосунку представлена через наступні функціональні модулі:

MainActivity – основна активність, яка ініціалізує застосунок, визначає стартовий екран (в залежності від наявності акаунту), координує перемикання між екранами та виконує базову ініціалізацію.

User Data Layer – представлена класом User, що зберігає персональні параметри користувача. Дані вводяться у процесі реєстрації та редагуються через EditProfileActivity. Ці параметри впливають на розрахунок рекомендованої калорійності та мети користувача.

Food Management – реалізований через Food, FoodSource та AddFoodActivity. Користувач може додати продукт, вказати його харчову

цінність. Дані зберігаються у локальну базу SQLite та використовуються для подальшого формування щоденного раціону.

Tracking Module – відповідає за облік та збереження спожитих продуктів. Для поточного дня використовується TodayEatingHistory, а для збереження історії – EatingHistory. Дані передаються між екранами, агрегуються та аналізуються у CalorieDiaries.kt.

DBHelper.kt – центральний клас для взаємодії з SQLite-базою. Реалізує методи додавання, оновлення, видалення та зчитування даних з таблиць користувача, продуктів, щоденників і історії прийомів їжі. База створюється при першому запуску або оновленні версії застосунку.

Візуалізація та аналітика – здійснюється у MainActivity, де відображається зведена інформація: добова норма калорій, спожиті калорії, статус досягнення мети. Виведення даних супроводжується прогрес-барами та кольоровими індикаторами.

Для зберігання та обміну даними між компонентами застосовується передача об’єктів через Intent, а також використання глобальних списків для денного споживання. Навігація між активностями реалізована стандартними засобами Android SDK.

На рисунку 2.1 наведено загальну структурну модель застосунку.

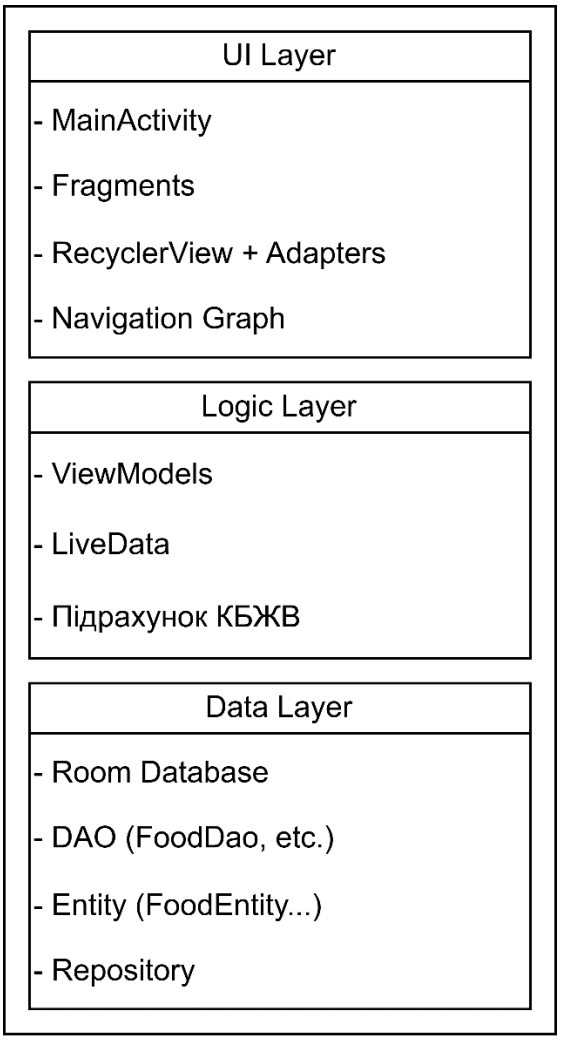


Рисунок 2.1 – Структурна модель застосунку

Таким чином, розроблена модель застосунку охоплює всі ключові аспекти роботи мобільного трекера харчування: введення та облік їжі, профіль користувача, агрегацію історії та відображення прогресу. Застосунок побудований із урахуванням розширюваності, тому в майбутньому можливе доповнення його новими функціями – рекомендаціями, автоматичним розпізнаванням продуктів, синхронізацією з хмарним сховищем тощо.

2.3 Розробка методу оцінки харчової збалансованості раціону

Оцінка харчової збалансованості є ключовим етапом у процесі моніторингу раціону користувача, оскільки дозволяє не лише фіксувати споживання їжі, а й аналізувати, наскільки фактичне харчування відповідає рекомендованим нормам.

Для реалізації цієї функції у застосунку було розроблено метод, який порівнює кількість спожитих калорій, білків, жирів і вуглеводів з індивідуальними добовими нормами користувача, визначеними на основі його фізіологічних параметрів та обраної мети.

Розрахунок норми калорій здійснюється за формулою 2.1, яка є формулою Міфліна-Сан Жеора:

$$N_{\text{кал}} = (10 \times \text{вага (кг)} + 6.25 \times \text{зріст (см)} - 5 \times \text{вік (років)} + S) \times \text{КФ} \quad (2.1)$$

де

S = +5 для чоловіків і –161 для жінок,

КФ – коефіцієнт фізичної активності, обраний зі шкали (1.2 – малорухомий, 1.55 – помірний, 1.9 – активний).

На основі загальної калорійності розраховуються добові норми основних нутрієнтів:

- білки: 15–20% загальної калорійності (1 г = 4 ккал);
- жири: 25–35% (1 г = 9 ккал);
- вуглеводи: 45–60% (1 г = 4 ккал).

Після фіксації кожного прийому їжі дані зберігаються у структурі TodayEatingHistory, яка дозволяє накопичити точну інформацію про сумарне споживання нутрієнтів за день. Кожна записана страва або продукт містить інформацію про вагу (у грамах), що використовується для розрахунку калорійності та вмісту КБЖВ з урахуванням базових значень у 100 г.

Для оцінки збалансованості використовується відносне відхилення фактичного значення від норми (формула 2.2):

$$\text{Відхилення (\%)} = \frac{(\text{Факт} - \text{Норма})}{\text{Норма}} \times 100 \quad (2.2)$$

де

Факт – фактичний показник,

Норма – нормальний показник.

У залежності від результатів відхилення інтерпретується наступним чином:

- до 10% – в межах норми;
- 10–25% – помірне відхилення (жовтий індикатор);
- понад 25% – критичне відхилення (червоний індикатор).

На основі результатів оцінки система виводить повідомлення з рекомендаціями, наприклад:

- "Споживання білків на 30% менше від норми. Додайте білкові продукти до вечері."
- "Жири перевищують норму на 40%. Спробуйте уникати смажених страв."

У цьому застосунку рекомендації формуються у модулі CalorieDiaries.kt, а відображаються у MainActivity, де реалізовано механізм графічної візуалізації результатів. Система дозволяє користувачеві швидко оцінити загальний стан харчування та вчасно відкоригувати раціон.

Таким чином, реалізований метод оцінки харчової збалансованості є простим і наочним способом надати користувачеві зворотний зв’язок щодо якості раціону. Поєднання математичних розрахунків і кольорових індикаторів забезпечує ефективне сприйняття інформації навіть без глибоких знань у сфері дієтології.

2.4 Розробка алгоритмів роботи застосунку

Для реалізації функціональних можливостей застосунку було розроблено низку алгоритмів, які забезпечують облік, обробку, аналіз та візуалізацію харчових даних користувача. Кожен із модулів, описаних у попередньому підрозділі, має власну логіку, яка відповідає за відповідні операції – від реєстрації до формування рекомендацій. У цьому підрозділі розглянемо базові алгоритми,

реалізовані в рамках бакалаврської кваліфікаційної роботи.

Спочатку користувач реєструється, вводить свої особисті дані, а саме зріст, вагу, вік, стать і активність і застосунок зберігає цю інформацію в базу даних. Блок-схему алгоритму реєстрації наведено на рисунку 2.2.

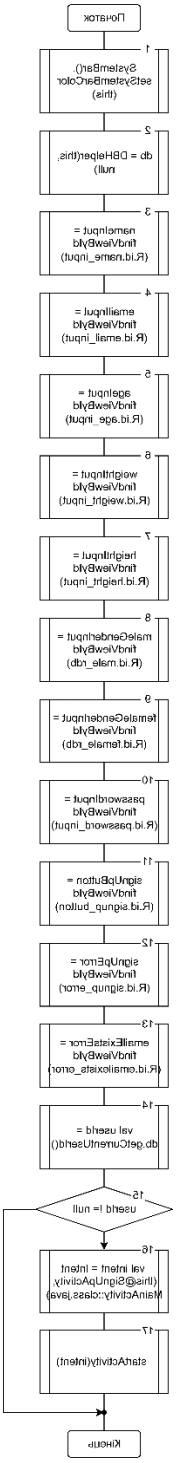


Рисунок 2.2 – Блок-схема алгоритму реєстрації

Після введення користувачем базових особистих даних – таких як зріст, поточна маса тіла, вік, стать та рівень фізичної активності – система виконує автоматичний розрахунок індивідуальної добової потреби в калоріях. Для цього використовується науково обґрунтована формула Міфліна-Сан Жеора, яка дозволяє з достатньою точністю оцінити базовий рівень метаболізму. Виходячи з отриманого значення добової калорійності, програма додатково формує персоналізовані норми споживання білків, жирів і вуглеводів, необхідні для забезпечення збалансованого харчування. Блок-схема алгоритму розрахунку добових норм КБЖВ наведено на рисунку 2.3.

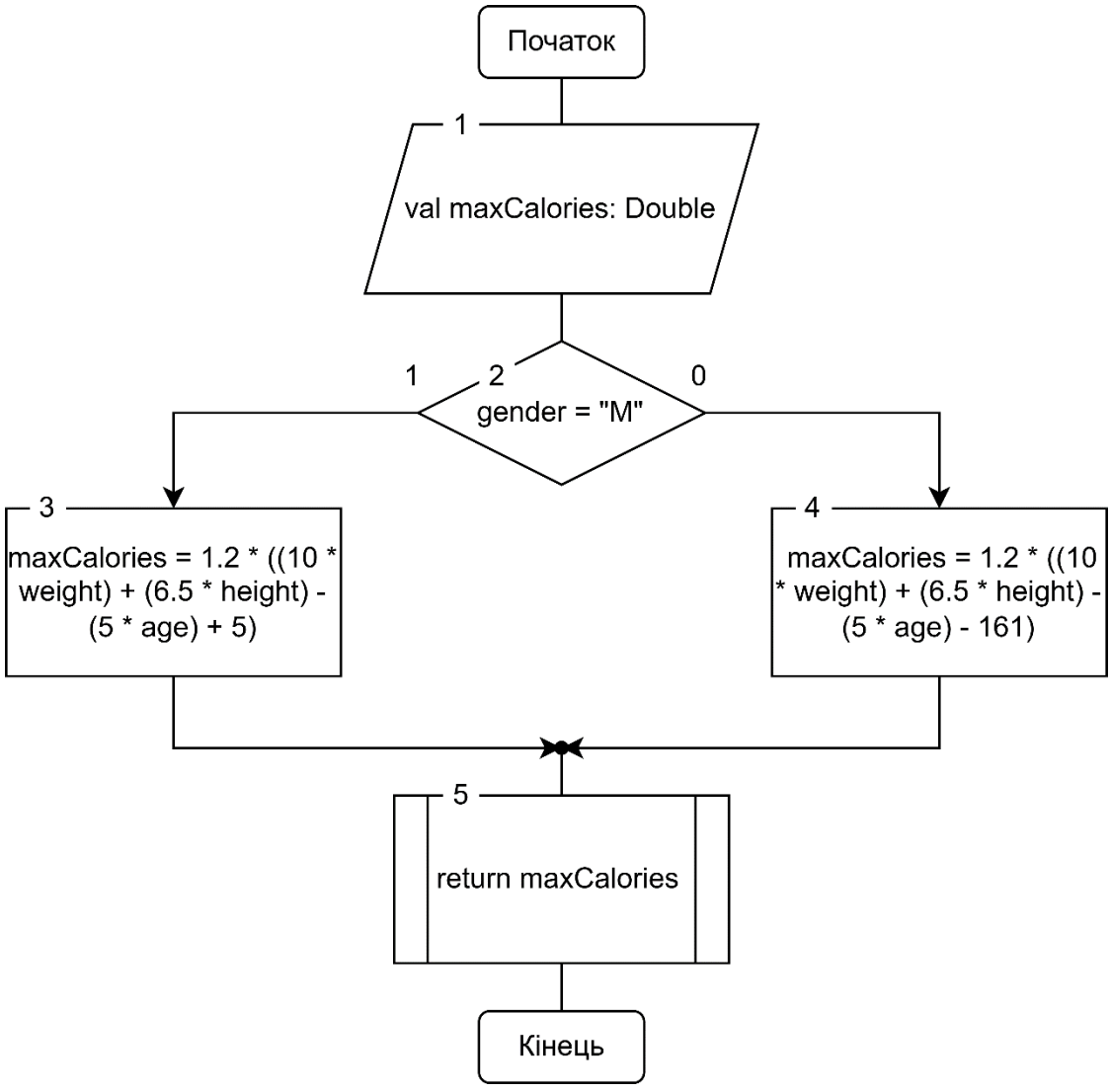


Рисунок 2.3 – Блок-схема розрахунку добових норм КБЖВ

Після того як користувач обирає продукт зі списку або додає новий, йому пропонується ввести масу порції, яку він планує спожити. Ця введена величина автоматично використовується для точного перерахунку основних поживних показників – калорійності, вмісту білків, жирів та вуглеводів. Після натискання кнопки підтвердження всі введені дані зберігаються у структурі TodayEatingHistory, яка відповідає за облік прийомів їжі, та одночасно оновлюють статистичні показники для поточного дня.

Алгоритм додавання нового продукту в систему зображено у вигляді блок-схеми на рисунку 2.4.

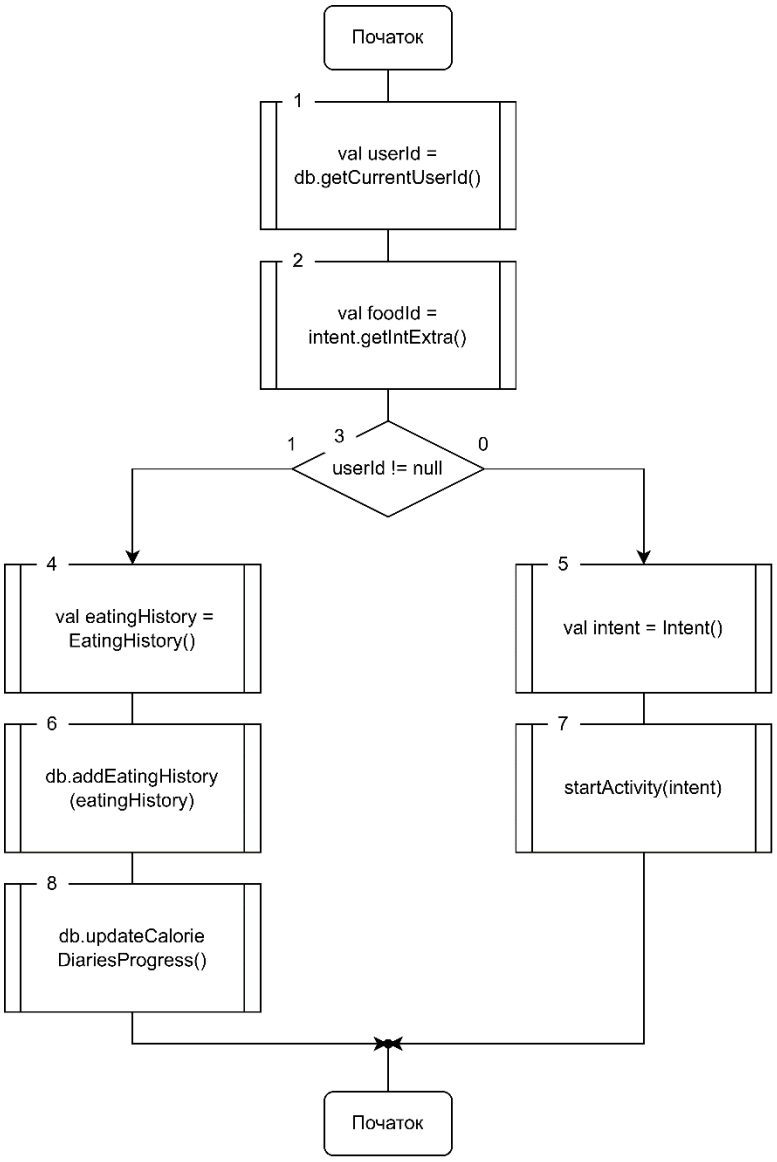


Рисунок 2.4 – Блок-схема алгоритму додавання продукту

Для кожного з основних макронутрієнтів – білків, жирів та вуглеводів – проводиться визначення рівня відхилення від встановленої добової норми споживання. У випадках, коли таке відхилення перевищує заздалегідь визначені порогові значення, система автоматично генерує візуальне сповіщення у вигляді кольорового індикатора, а також супроводжує його текстовою порадою щодо можливих змін у раціоні. Усі результати розрахунку надсилаються у відповідний програмний модуль CalorieDiaries.kt для подальшої обробки. Алгоритм оцінки збалансованості харчового раціону, реалізований у програмі, представлений у вигляді блок-схеми на рисунку 2.5.

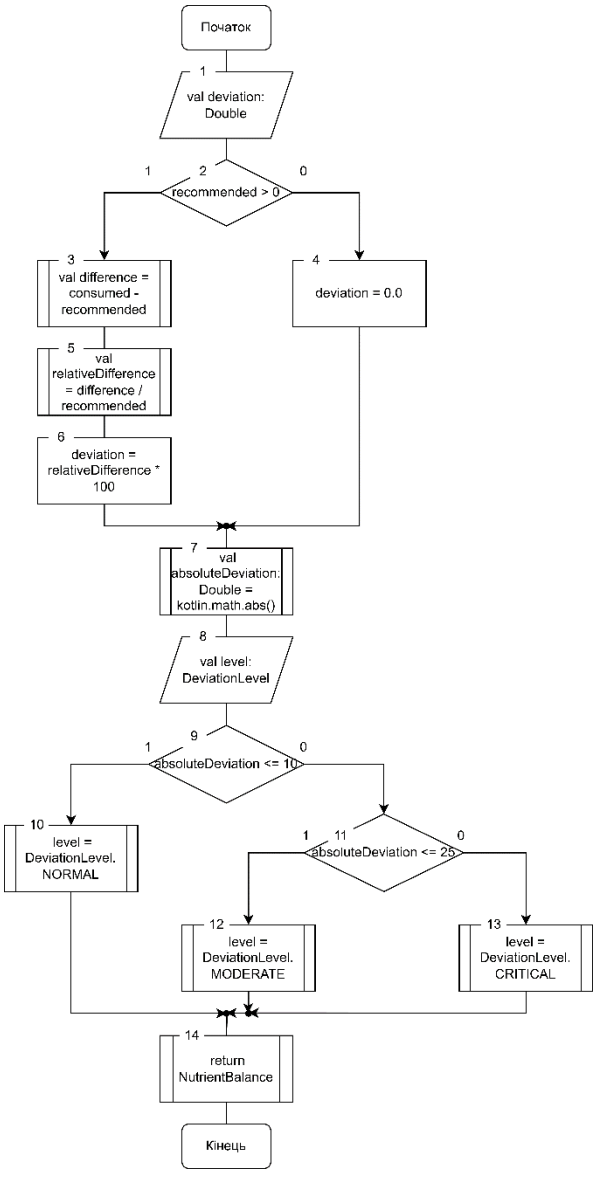


Рисунок 2.5 – Блок-схема алгоритму оцінки збалансованості харчового раціону

На основі результатів розрахунків, збережених у CalorieDiaries.kt, здійснюється аналіз відхилень і виведення текстового повідомлення. Це повідомлення містить поради щодо покращення раціону, наприклад: зменшити жирну їжу, збільшити білкові продукти тощо. Блок-схема алгоритму формування рекомендацій наведена на рисунку 2.6.

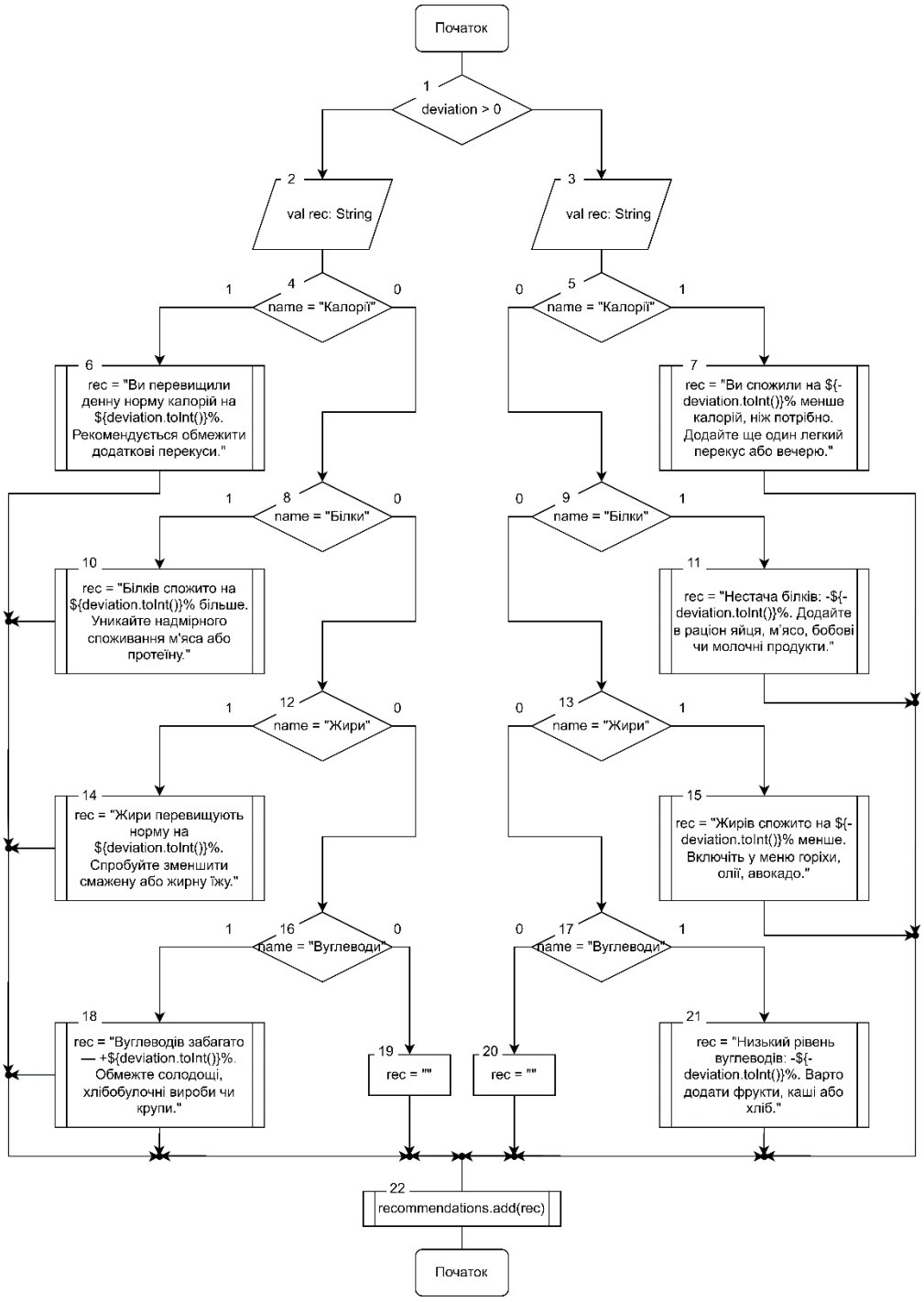


Рисунок 2.6 – Алгоритм формування текстових рекомендацій користувачу

Усі описані алгоритми реалізовано мовою Kotlin із використанням бази SQLite та активностей Android. Вони працюють в режимі офлайн, забезпечуючи повну автономність і захист даних.

Таким чином, кожна ключова функція застосунку базується на чітко визначеному алгоритмі, що сприяє підвищенню надійності, стабільності та зрозумілості використання мобільного інструменту моніторингу харчування.

2.5 Висновки

У результаті виконання другого розділу було проведено детальний аналіз інформаційного забезпечення майбутнього застосунку, сформовано логічну структуру системи з урахуванням її функціональних модулів, а також розроблено модель оцінки збалансованості раціону. Крім того, реалізовано ключові алгоритми роботи – додавання прийому їжі, аналіз щоденного харчування та формування рекомендацій. Створені блок-схеми алгоритмів ілюструють логіку реалізації програмної частини та слугують основою для наступного етапу – безпосередньої розробки програмних модулів.

3 РОЗРОБКА ЗАСТОСУНКУ

3.1 Варіантний аналіз та вибір мови програмування

На етапі проєктування мобільного застосунку для моніторингу харчування одним із найважливіших технічних рішень є обґрунтований вибір мови програмування, яка відповідатиме всім ключовим вимогам до продуктивності, масштабованості, підтримуваності коду та гнучкості інтеграції з екосистемою Android. Особливої уваги потребує також можливість ефективної роботи з сучасними бібліотеками, архітектурними компонентами, інструментами UI-розробки та оптимізації асинхронних задач, що є критично важливим для мобільних застосунків у сфері здоров'я та харчування [8].

У цьому контексті було проаналізовано три найбільш релевантні технологічні варіанти – Kotlin, Java та Flutter (Dart) – кожен із яких має свої сильні сторони та обмеження, що впливають на вибір залежно від поставлених цілей і специфіки проєкту.

Kotlin – це сучасна мова програмування, створена компанією JetBrains та офіційно рекомендована Google як основна мова для Android-розробки [9]. Вона вирізняється лаконічним, інтуїтивно зрозумілим синтаксисом, високим рівнем безпеки типів (null safety) і широкими можливостями для функціонального програмування. Завдяки цим характеристикам, код на Kotlin є менш громіздким і водночас більш надійним та читабельним. Однією з ключових переваг є підтримка корутин – потужного інструменту для асинхронного програмування, що дозволяє ефективно обробляти фонові процеси, такі як збереження харчових даних, мережеві запити до API та оновлення інтерфейсу без затримок або зависань.

Ще однією перевагою Kotlin є повна сумісність із Java, що відкриває можливість використовувати тисячі вже існуючих бібліотек, створених для Android. Крім того, Kotlin тісно інтегрується з архітектурними компонентами Android Jetpack – такими як ViewModel, LiveData, Room, Navigation, DataStore, StateFlow тощо [10, 11]. Це дозволяє реалізовувати сучасні шаблони архітектури

(наприклад, MVVM, Clean Architecture) та значно пришвидшує розробку стабільного, тестованого та масштабованого застосунку. У рамках цієї кваліфікаційної роботи Kotlin обрано як основну мову реалізації, оскільки він найкраще поєднує швидкість розробки, сучасні практики, безпеку й підтримку спільнотою.

Java – це класична, перевірена часом мова програмування, яка довгий час була стандартом для Android-розробки [12]. Вона характеризується високим рівнем стабільності, широким охопленням документації, великою базою розробників та сумісністю з більшістю інструментів Android SDK. Java все ще залишається актуальною у багатьох корпоративних рішеннях. Проте, порівняно з Kotlin, Java має громіздкіший синтаксис і вимагає більше коду для реалізації тих самих функціональних можливостей. Також вона менш ефективна у впровадженні сучасних підходів до UI та реактивного програмування, що знижує продуктивність команди розробки та ускладнює підтримку проєкту в довгостроковій перспективі.

Flutter (Dart) – це сучасний фреймворк для кросплатформної розробки, який дозволяє створювати застосунки одразу для Android та iOS, використовуючи єдину кодову базу [13]. Його головними перевагами є висока швидкість розробки, багатий набір готових UI-компонентів, «живий» перегляд змін (hot reload) та привабливий зовнішній вигляд інтерфейсу. Flutter активно використовується у стартапах і прототипуванні. Водночас, у випадках, коли потрібна глибока інтеграція з нативними Android-компонентами – наприклад, із Room, LiveData, StateFlow, Jetpack Compose або бібліотеками на Java/Kotlin – Flutter вимагає складнішої міжплатформної взаємодії через платформи-канали. Це ускладнює розробку та нівелює перевагу єдиної кодової бази, особливо у застосунках, орієнтованих на Android.

На таблиці 3.1 наведено детальний порівняльний аналіз мов програмування, які стояли перед вибором.

Таблиця 3.1 – Порівняльний аналіз мов програмування

Характеристика	Kotlin	Java	Dart (Flutter)
Підтримка Android	+	+	+
Лаконічність коду	+	–	+
Підтримка сучасної архітектури (MVVM)	+	–	+
Інтеграція з Jetpack	+	+	–
Кросплатформність	–	–	+
Графічна продуктивність UI	+	–	+
Сумарний бал (з 6)	5	2	5

Зважаючи на потребу в глибокій інтеграції з Android SDK, необхідність стабільної роботи з базами даних та графіками, а також бажання зменшити кількість коду без втрати функціональності, для реалізації застосунку обрано мову Kotlin [14, 15]. Вона дозволяє використовувати всі переваги сучасної Android-екосистеми та забезпечує високу швидкість розробки при збереженні читаємості й підтримуваності коду.

3.2 Розробка модуля оцінки харчової збалансованості раціону

Метою модуля оцінки харчової збалансованості є виявлення відхилень у споживанні основних макронутрієнтів – калорій, білків, жирів та вуглеводів – відносно встановлених добових норм, що дозволяє надавати користувачеві обґрунтовані рекомендації щодо корекції раціону.

Для реалізації цієї функціональності у застосунку створено окремий об'єкт BalanceEvaluator, який включає визначення ступеня відхилення для кожного з нутрієнтів і класифікацію цього відхилення за рівнями критичності. Клас CalorieDiaries, що зберігає поточні значення фактичного та рекомендованого споживання КБЖВ, передається як вхідний параметр до методу evaluateBalance().

Метод виконує оцінку збалансованості за чотирма показниками: калорійністю (Calories), білками (Proteins), жирами (Fats) та вуглеводами (Carbs). Для кожного нутрієнта викликається допоміжна функція evaluate(), яка обчислює відсоткове відхилення між фактичною та рекомендованою кількістю.

Результат інтерпретується за трирівневою шкалою:

- NORMAL – відхилення до 10% (в межах норми);
- MODERATE – від 10% до 25% (помірне відхилення, жовтий індикатор);
- CRITICAL – понад 25% (критичне відхилення, червоний індикатор).

Після класифікації для кожного нутрієнта створюється об'єкт NutrientBalance, який містить:

- назву нутрієнта;
- фактичне споживання;
- рекомендовану норму;
- відсоткове відхилення;
- рівень відхилення (DeviationLevel).

Ці об'єкти повертаються у вигляді списку, що дозволяє одночасно відобразити стан усіх основних нутрієнтів у загальній аналітичній панелі користувача. Візуалізація результатів здійснюється в основному інтерфейсі (MainActivity) у вигляді кола балансу, яке має червоний колір, якщо рівень критичний, жовтий, якщо помірний, зелений – якщо нормальний.

Таким чином, модуль BalanceEvaluator є ключовим елементом логіки прийняття рішень у застосунку, оскільки забезпечує оперативний аналіз стану КБЖВ-балансу та дозволяє формувати адаптивні підказки. Його використання забезпечує простий, інформативний та наочний спосіб інтерпретації даних про харчування навіть для користувачів без спеціальних знань у сфері дієтології.

3.3 Розробка модуля обліку КБЖВ та генерації рекомендацій

Однією з ключових функцій мобільного застосунку є не лише облік фактичного споживання основних макронутрієнтів (калорій, білків, жирів і вуглеводів), але й формування інформативних, персоналізованих рекомендацій на основі відхилень від добових норм. Для реалізації цього функціоналу було створено окремий модуль, який складається з двох основних компонентів: RecommendationGenerator – логіки формування рекомендацій на основі оцінки балансу, та RecommendationActivity – інтерфейсу їх відображення у застосунку.

Основна функція модуля RecommendationGenerator – це метод generateRecommendations(balanceList: List<NutrientBalance>), який отримує список результатів оцінки нутрієнтів, створений у BalanceEvaluator, і формує набір текстових рекомендацій залежно від типу та ступеня відхилення.

Кожен нутрієнт (калорії, білки, жири, вуглеводи) аналізується окремо. Якщо відхилення потрапляє до рівнів MODERATE або CRITICAL, система генерує відповідне повідомлення. При цьому враховується як перевищення, так і нестача споживання нутрієнта. Наприклад:

1. Якщо білків спожито на 30% менше від норми – користувач отримає пораду "Protein deficit: 30%. Add eggs, meat, legumes, or dairy."
2. Якщо жирів перевищено на 40% – рекомендація буде "Fat intake exceeds the norm by 40%. Try reducing fried or fatty foods."

Таким чином, рекомендації формуються динамічно, із врахуванням напрямку та величини відхилення, що дозволяє зробити поради персоналізованими та адаптивними.

Другим важливим елементом є активність RecommendationActivity, яка відповідає за графічне представлення сформованих порад користувачеві. Після запуску активність виконує такі кроки:

1. Отримує ідентифікатор поточного користувача та дату.
2. Завантажує дані про КБЖВ із бази даних через клас DBHelper.
3. Викликає метод evaluateBalance() з модуля BalanceEvaluator.
4. Передає список NutrientBalance до RecommendationGenerator.
5. Отримує список рядків-рекомендацій і динамічно додає їх у вигляді елементів інтерфейсу TextView.

Якщо баланс нутрієнтів знаходиться в межах норми (усі відхилення $\leqslant$ 10%), то замість рекомендацій виводиться повідомлення "Your diet is balanced today!". Такий підхід сприяє підвищенню мотивації користувача до дотримання рекомендацій та спрощує сприйняття інформації.

Модуль обліку КБЖВ та генерації рекомендацій є центральним компонентом застосунку з точки зору зворотного зв’язку. Він об’єднує результати

аналізу харчування з візуалізацією та поясненням для користувача. Його використання дозволяє не лише накопичувати статистику про харчування, а й активно впливати на зміну харчової поведінки завдяки зрозумілим і практичним порадам.

Завдяки використанню Kotlin та архітектурних патернів Android (розмежування логіки і UI), реалізація цього модуля є розширюваною: у майбутньому можливо додати генерацію порад на основі історії харчування, динамічне врахування цілей (схуднення/набір ваги), а також локалізацію мовами користувачів.

3.4 Розробка інтерфейсу користувача

Інтерфейс мобільного застосунку для моніторингу збалансованого харчування створено з урахуванням основних принципів сучасного дизайну – зокрема, простоти використання, візуальної наочності та зосередженості на ключових потребах користувача. Основна увага приділялася забезпеченню зручного доступу до основних функцій, серед яких – моніторинг щоденного харчування, швидке додавання нових продуктів, перегляд персоналізованих рекомендацій та ефективне керування особистим профілем. Щоб зробити користування застосунком інтуїтивно зрозумілим, для кожного з ключових сценаріїв взаємодії було розроблено окремий інтерфейсний екран із чіткою логікою навігації. Структура й логіка цих екранів детально представлена на відповідних схемах, що ілюструють послідовність дій користувача та розміщення основних елементів інтерфейсу (рис. 3.4 – 3.10).

При запуску застосунку програма пропонує увійти або зареєструватись. На цьому екрані є 2 поля для введення логіну та паролю та кнопка «Log in». Структурну схему екрану авторизації зображено на рисунку 3.4.

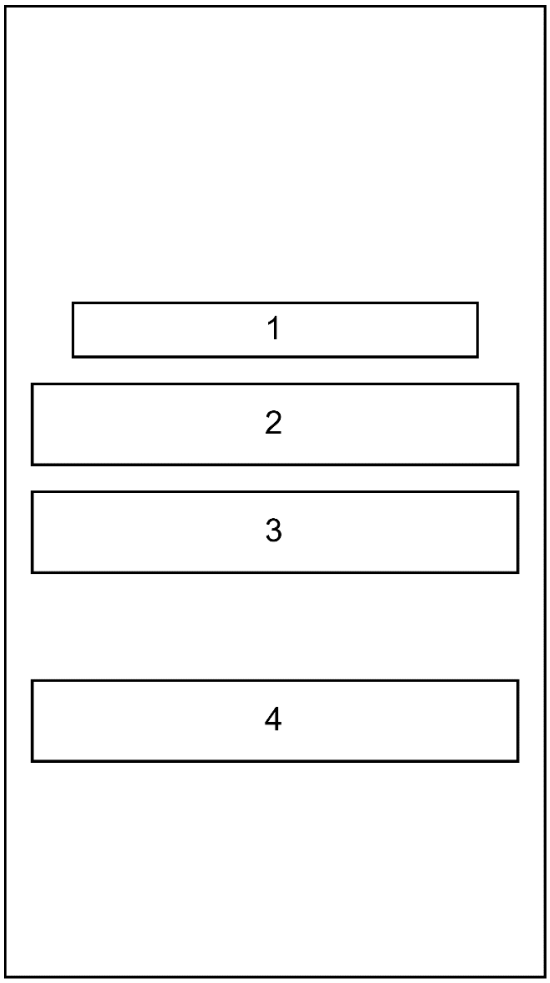


Рисунок 3.4 – Структурна схема екрану авторизації

Складові елементи екрану авторизації:

1. Заголовок.
2. Поле для вводу електронної пошти.
3. Поле для вводу паролю.
4. Кнопка «Log in» .

Після авторизації користувач потрапляє на головний екран, де представлено загальну картину добового харчування. У верхній частині знаходиться заголовок. Нижче – блок статистики за день: тут показано, скільки калорій, білків, жирів і вуглеводів уже спожито відносно добової норми. Також реалізовано кнопку переходу до екрану профілю. Нижче – поле для додавання прийомів їжі за день та кнопка для відкриття екрану персоналізованих рекомендацій.

Структурну схему головного екрану зображено на рисунку 3.5.

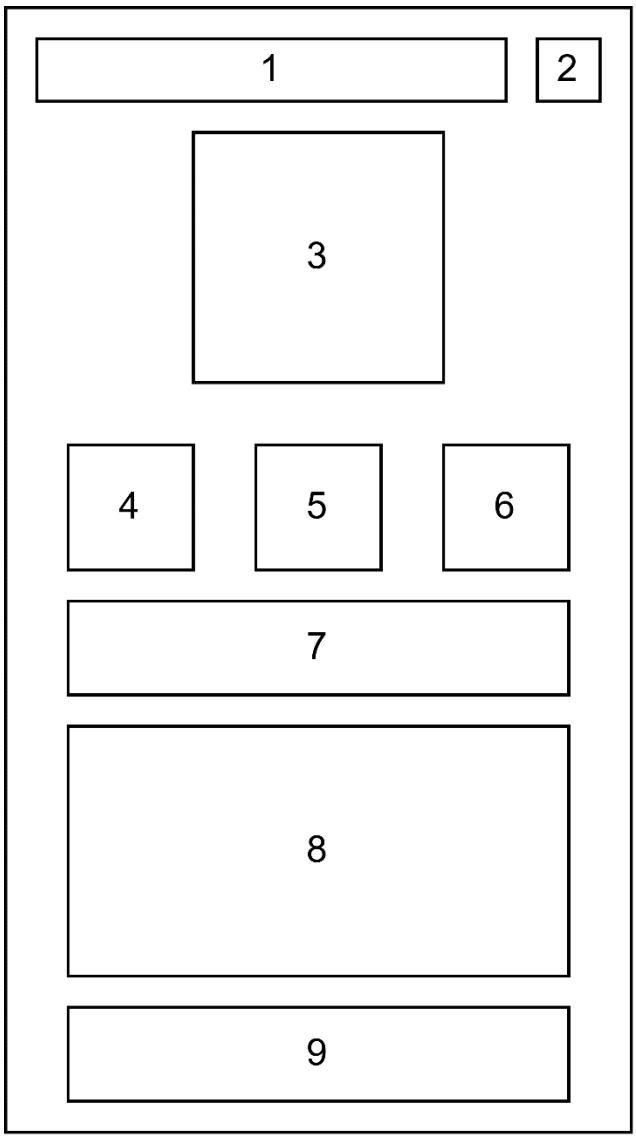


Рисунок 3.5 – Структурна схема головного екрану

Складові елементи головного екрану застосунку:

1. Заголовок.
2. Кнопка переходу до екрану профілю.
3. Коло прогресу калорійності та індикатор збалансованості раціону.
4. Коло прогресу вуглеводів.
5. Коло прогресу білків.
6. Коло прогресу жирів.

7. Кнопка «Today».
8. Прийоми їжі.
9. Персоналізовані рекомендації.

На екрані пошуку їжі користувач може додати конкретну страву або продукт. Основну увагу зосереджено на полях для вибору продукту та введення назви продукту. Після натискання на обраний продукт відбувається перехід до екрану додавання обаної страви до списку спожитого за день.

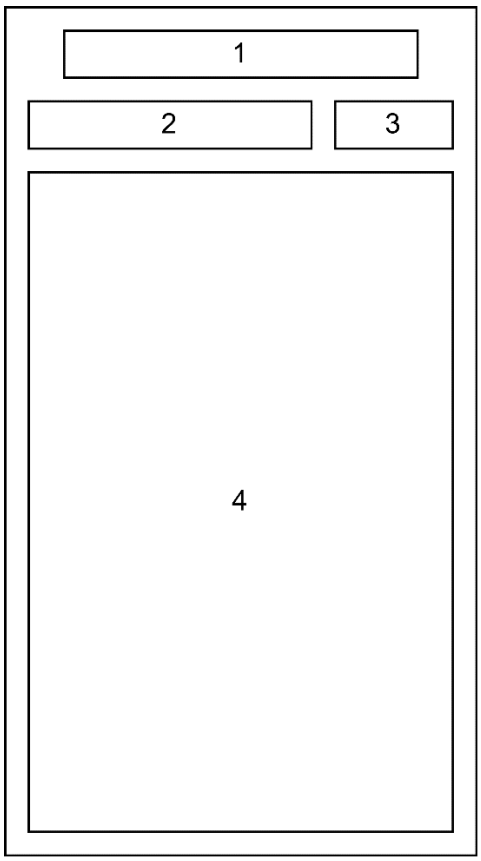


Рисунок 3.6 – Структурна схема екрану пошуку їжі

Складові елементи екрану пошуку страви:

1. Співвідношення спожитих калорій та норми за день.
2. Поле для введення тексту.
3. Кнопка пошуку.
4. Поле для вибору страви.

Після натискання на обрану страву відкривається екран додавання їжі. На

цьому екрані вказана назва трави або продукту, поле для введення маси спожитого продукту та розрахунок КБЖВ відносно до маси спожитої страви чи продукту. Також нижче є кнопка для додавання обраної кількості страви до денного раціону.

Структурну схему екрану додавання їжі зображено на рисунку 3.7.

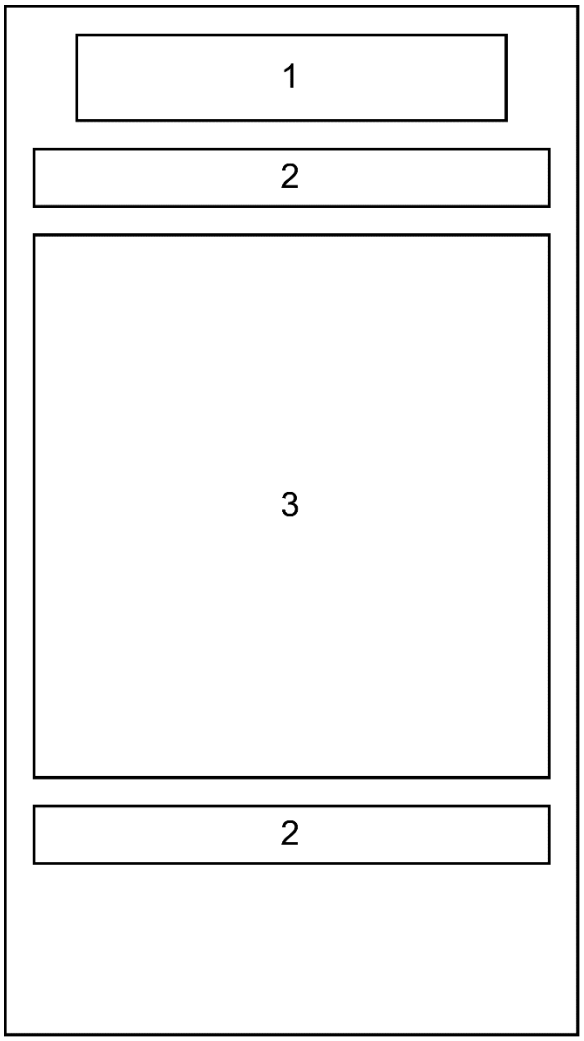


Рисунок 3.7 – Структурна схема екрану додавання їжі

Складові елементи екрану додавання їжі:

1. Назва страви.
2. Текстове поле для введення кількості грамів.
3. КБЖВ страви відносно введеної маси.
4. Кнопка «Add».

Усі екрани мобільного застосунку логічно охоплюють основні дії користувача на всіх етапах взаємодії з системою – від авторизації до щоденного моніторингу харчування та перегляду персоналізованих порад. Інтерфейс побудовано так, щоби зробити процес максимально зручним, інтуїтивним і зрозумілим навіть для нових користувачів.

Форма авторизації дозволяє швидко увійти до облікового запису, після чого відкривається головний екран, де представлено зведену інформацію про стан споживання КБЖВ за день. Завдяки візуальним індикаторам прогресу та зручній навігації користувач може одразу оцінити свій поточний стан харчування.

Екран пошуку продуктів і страв забезпечує швидке додавання спожитих позицій до денного раціону. У випадку складних страв (з кількох інгредієнтів) реалізовано функцію введення ваги, після чого система автоматично розраховує поживну цінність.

Модуль персоналізованих рекомендацій формує текстові підказки відповідно до фактичного відхилення від добових норм. Якщо раціон є збалансованим, виводиться мотивувальне повідомлення. Інакше користувач отримує короткі практичні поради щодо змін у харчуванні (наприклад, «Додайте білкові продукти» або «Зменште споживання жирів»).

Екран профілю дозволяє вносити зміни до параметрів, що впливають на розрахунок добових потреб – вік, стать, вага, зріст тощо. Змінити можна також ім’я та email. Окремі елементи інтерфейсу виділено для збереження внесених змін або повернення назад без збереження.

Кожен екран зосереджений на виконанні конкретного завдання – авторизації, перегляду статистики, додавання їжі, перегляду порад або редагування профілю. Така чітка структуризація дозволяє забезпечити послідовний користувацький досвід і підвищити ефективність використання застосунку як інструменту для щоденного контролю харчування.

3.5 Висновки

У третьому розділі було здійснено аналіз мов програмування та обґрунтовано вибір Kotlin як основної мови реалізації мобільного застосунку завдяки її сумісності з Android, підтримці сучасної архітектури та зручному синтаксису.

Розроблено модуль оцінки харчової збалансованості BalanceEvaluator, який аналізує відхилення у споживанні основних нутрієнтів та класифікує їх за трирівневою шкалою. Створено також модуль RecommendationGenerator, що формує персоналізовані поради на основі цих відхилень.

Інтерфейс застосунку охоплює всі ключові сценарії: авторизацію, додавання їжі, перегляд статистики, рекомендацій та редагування профілю. Завдяки поєднанню аналітики та зручного UI реалізовано ефективний інструмент для щоденного контролю харчування.

4 ТЕСТУВАННЯ ЗАСТОСУНКУ

4.1 Аналіз методів тестування

Тестування застосунків для моніторингу харчування є критично важливою частиною життєвого циклу розробки, оскільки ці програми безпосередньо впливають на рішення користувача щодо здоров’я, дієти та способу життя. Якісне тестування забезпечує коректність розрахунків споживаних нутрієнтів, стабільність роботи застосунку та правильність формування персоналізованих рекомендацій. Такі застосунки мають специфічні вимоги до функціональності, включаючи облік калорій та КБЖВ, порівняння із нормами, відображення статистики та генерацію порад.

Функціональне тестування є основним методом перевірки відповідності реалізованих функцій заявленим вимогам. У контексті дієтичного застосунку воно охоплює такі аспекти: точність обчислення добової калорійності, правильність розрахунку відхилень по білках, жирах і вуглеводах, логіка формування рекомендацій, облік денного споживання тощо. Завдяки функціональному тестуванню можна виявити помилки в обчисленнях, які безпосередньо впливають на якість рекомендацій.

Тестування інтерфейсу користувача (UI-тестування) зосереджене на перевірці зручності використання застосунку, відповідності елементів інтерфейсу очікуваному вигляду та адекватності реакції на дії користувача. Для додатків з відстеження раціону особливо важливо, щоб процес додавання продукту був простим, а візуальні індикатори (кольори, графіки, прогрес-бари) чітко передавали стан КБЖВ-балансу. UI-тестування допомагає виявити труднощі у навігації, неправильне відображення даних або некоректне реагування на дії користувача.

Ручне тестування є ефективним способом виявлення помилок у реальних сценаріях використання. Тестувальник має змогу вручну дослідити роботу додатку, змінюючи параметри профілю, додаючи страви, перевіряючи рекомендації та спостерігаючи за реакцією системи на нестандартні або граничні

значення (наприклад, надто велика порція або повна відсутність прийомів їжі). Такий підхід дозволяє гнучко реагувати на виявлені помилки та перевіряти сценарії, які важко автоматизувати.

На основі порівняння методів тестування було обрано саме ручне функціональне тестування як найбільш адаптивне до цілей цієї розробки. Воно дозволяє змоделювати поведінку реального користувача, виявити помилки в логіці формування порад, перевірити стабільність збереження даних та якість взаємодії між модулями. З огляду на важливість точності обчислень і зручності щоденного використання, саме цей метод дав змогу виявити й усунути критичні недоліки на ранніх етапах.

4.2 Тестування застосунку для моніторингу харчування

Тестування застосунку починається з авторизації. Потрібно ввести пароль на логін для того, щоб зайти в профіль та отримувати актуальну оцінку та рекомендації щодо раціону. Екран авторизації зображено на рисунку 4.1.

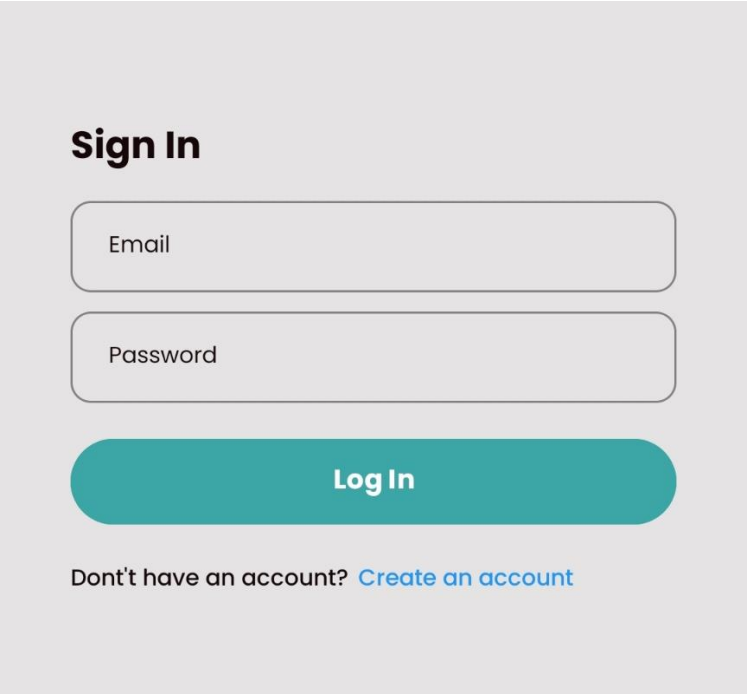


Рисунок 4.1 – Екран авторизації застосунку

Для того щоб увійти в застосунок потрібно спочатку зареєструватись у ньому. Для цього натиснемо кнопку «Create an account». Після цього буде виведено екран реєстрації, у якому потрібно ввести свої дані такі як ім’я, пошту, вік, вагу, зріст, стать та пароль. Екран реєстрації наведено на рисунку 4.2.

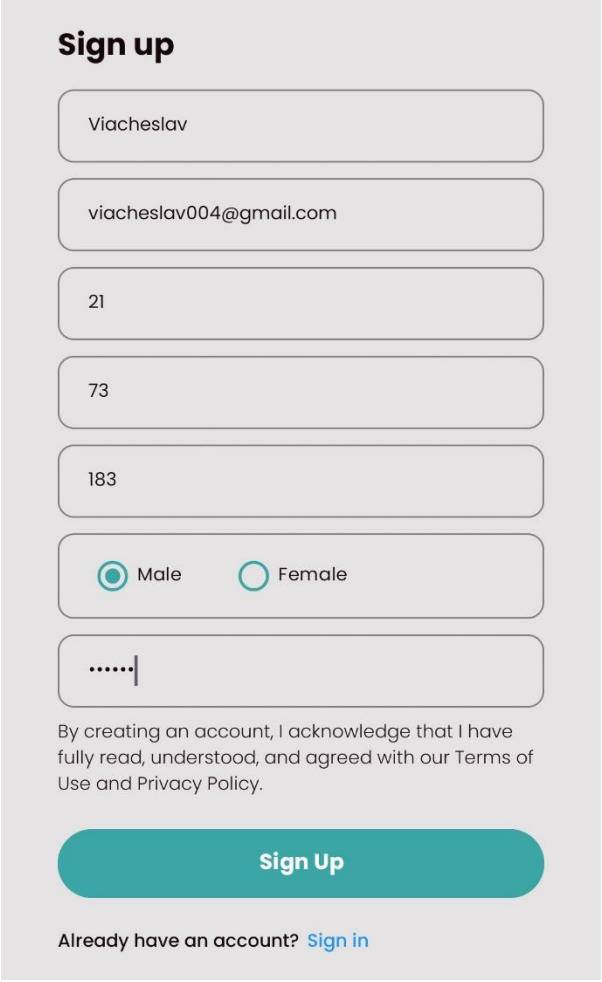


Рисунок 4.2 – Реєстрація у застосунку

Після того як буде натиснута кнопка «Sign Up», буде виведено головний екран застосунку, який містить всі відомості про спожиті за день калорії, білки, жири та вуглеводи. Виведено коло індикацію харчової збалансованості раціону та додано всю навігацію застосунку від змін даних про користувача до генерації персоналізованих рекомендацій.

Головний екран застосунку зображено на рисунку 4.3.

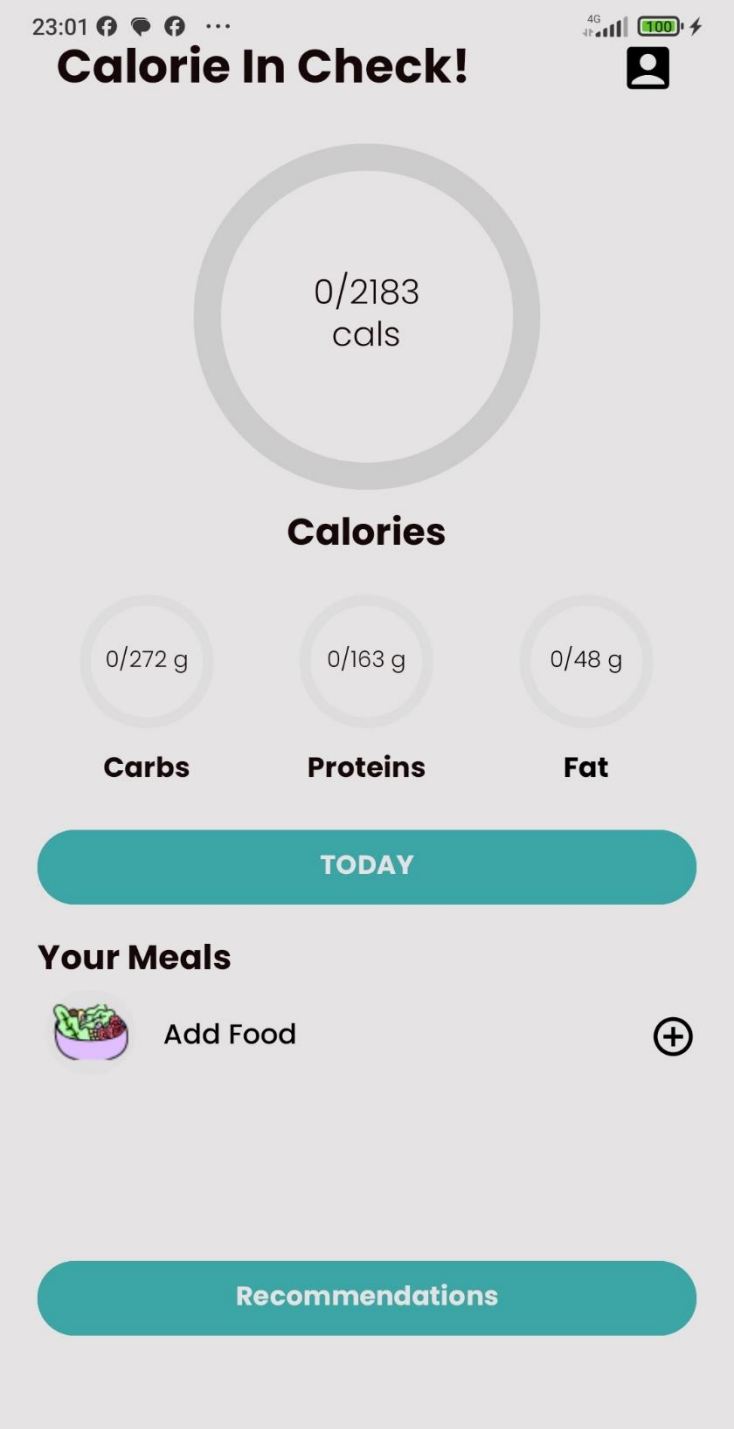


Рисунок 4.3 – Головний екран застосунку

Наступним кроком буде додавання в добовий раціон визначеної страви або продукту серед списку запропонованих локальною базою. Для цього потрібно натиснути кнопку «Add Food». Після чого відкриється меню пошуку потрібної страви або продукту.

Екран пошуку їжі наведено на рисунку 4.4.

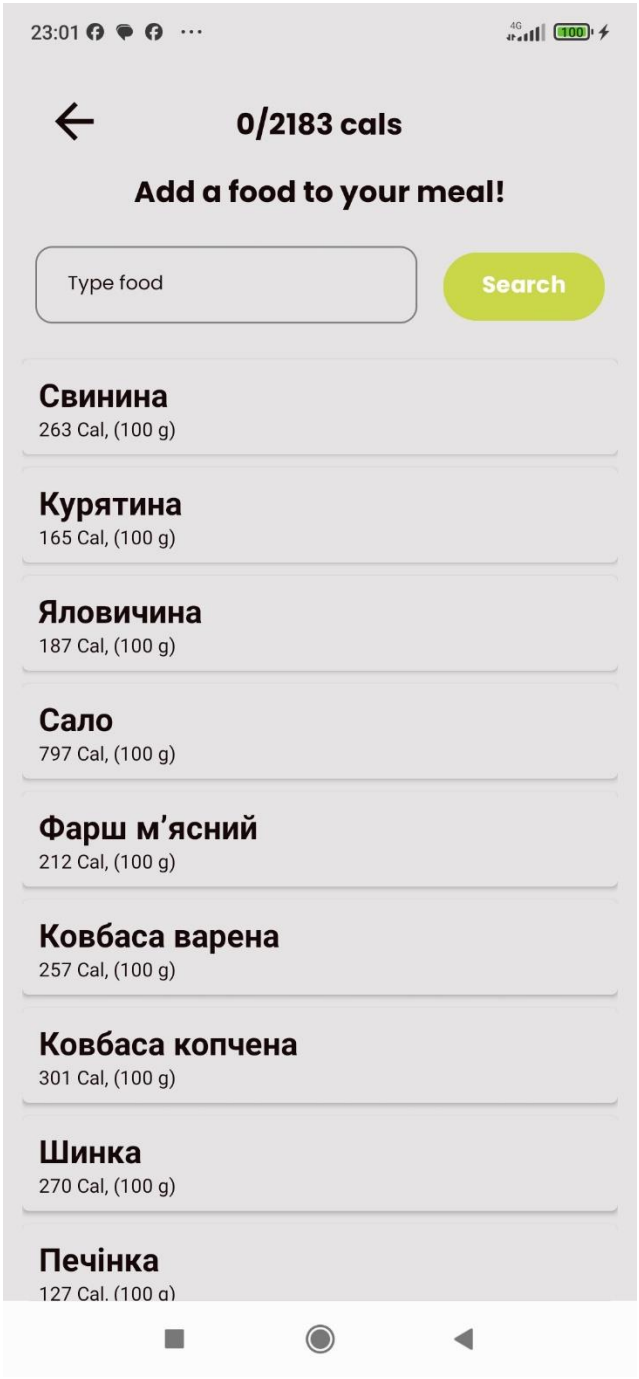


Рисунок 4.4 – Пошук їжі у застосунку

Серед списку запропонованих продуктів можна обрати той, який ви спожили, або збираєтесь спожити, також тут є наявна функція пошуку продукту за назвою. Після того як буде обрано продукт потрібно натиснути на нього, після чого відкриється екран додавання їжі у добовий раціон.

Екран додавання їжі у добовий раціон зображено на рисунку 4.5.

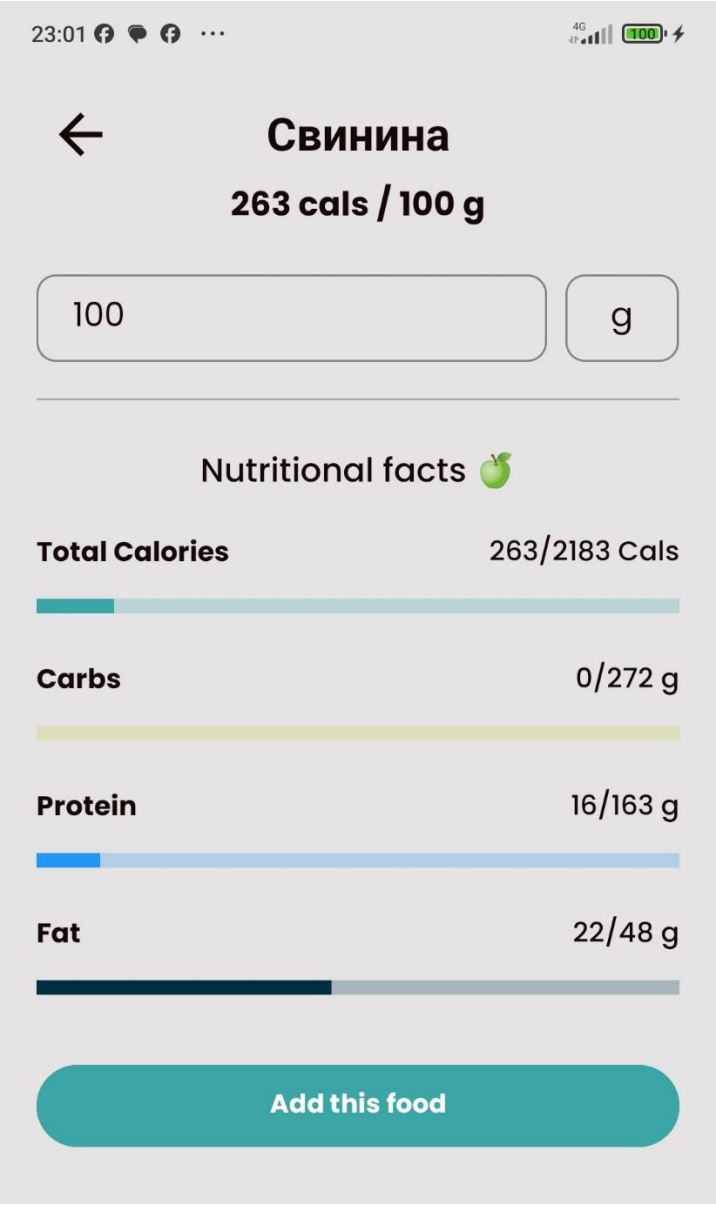


Рисунок 4.5 – Екран додавання їжі

Після виведення екрану додавання їжі буде надано всі відомості про калорійність та нутрієнти. А саме на шкалі зображено співвідношення калорійності продукту до потрібної калорійності на добу. Також це працює і з вуглеводами, білками та жирами. Також наявною є можливість ввести кількість спожитого продукту у грамах. Екран додавання їжі зі зміненими показниками наведено на рисунку 4.6.

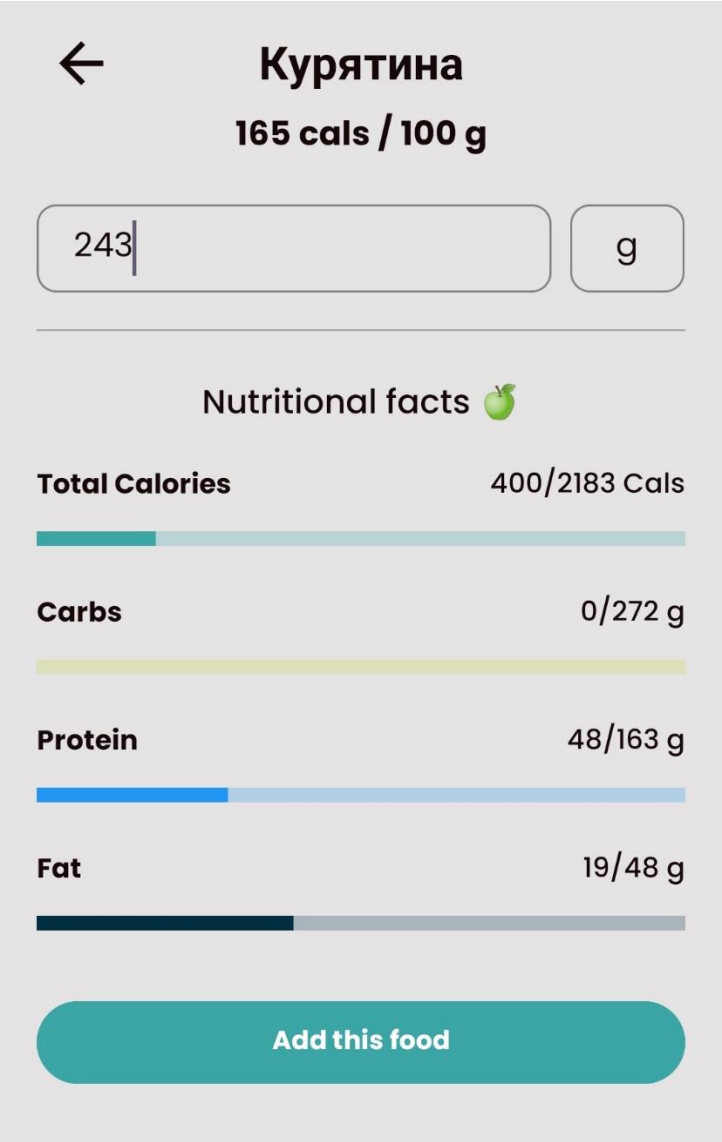


Рисунок 4.6 – Екран додавання їжі зі зміненими показниками

Після того як буде введено потрібну кількість спожитого продукту потрібно натиснути на кнопку «Add this food». Після чого їжу буде додано в добовий раціон.

Для демонстрації функціоналу застосунку було додано ще декілька продуктів у добовий раціон, щоб показати роботу методу оцінки харчової збалансованості раціону. Після додавання ще декількох продуктів, коло прогресу калорійності стало червоним, що є індикатором того, що баланс раціону є критичним і його негайно потрібно коригувати. Робота індикатору оцінки харчової збалансованості раціону зображено на рисунку 4.7.

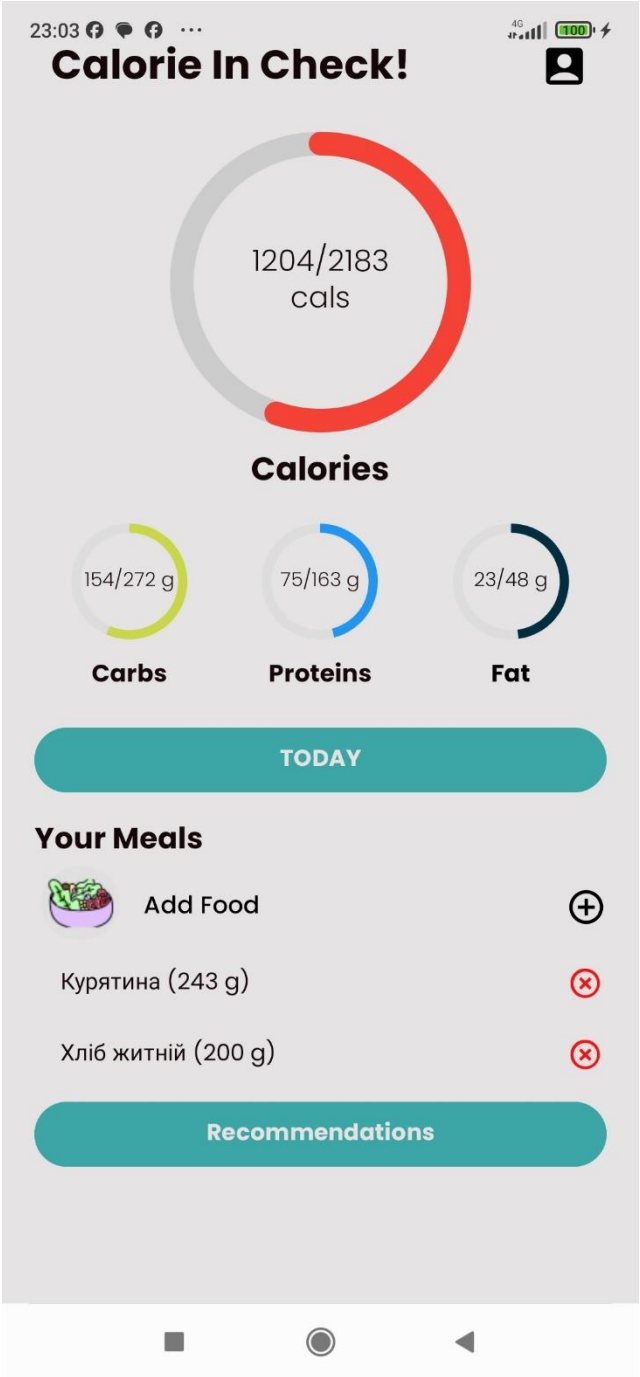


Рисунок 4.7 – Робота індикатора оцінки харчової збалансованості роціону

Після додавання продуктів до добового раціону та моніторингу всіх нутрієнтів, калорійності та збалансованості раціону, можна натиснути на кнопку «Recomendations», після чого буде виведено екран з персоналізованими рекомендаціями.

Екран персоналізованих рекомендацій зображено на рисунку 4.8.

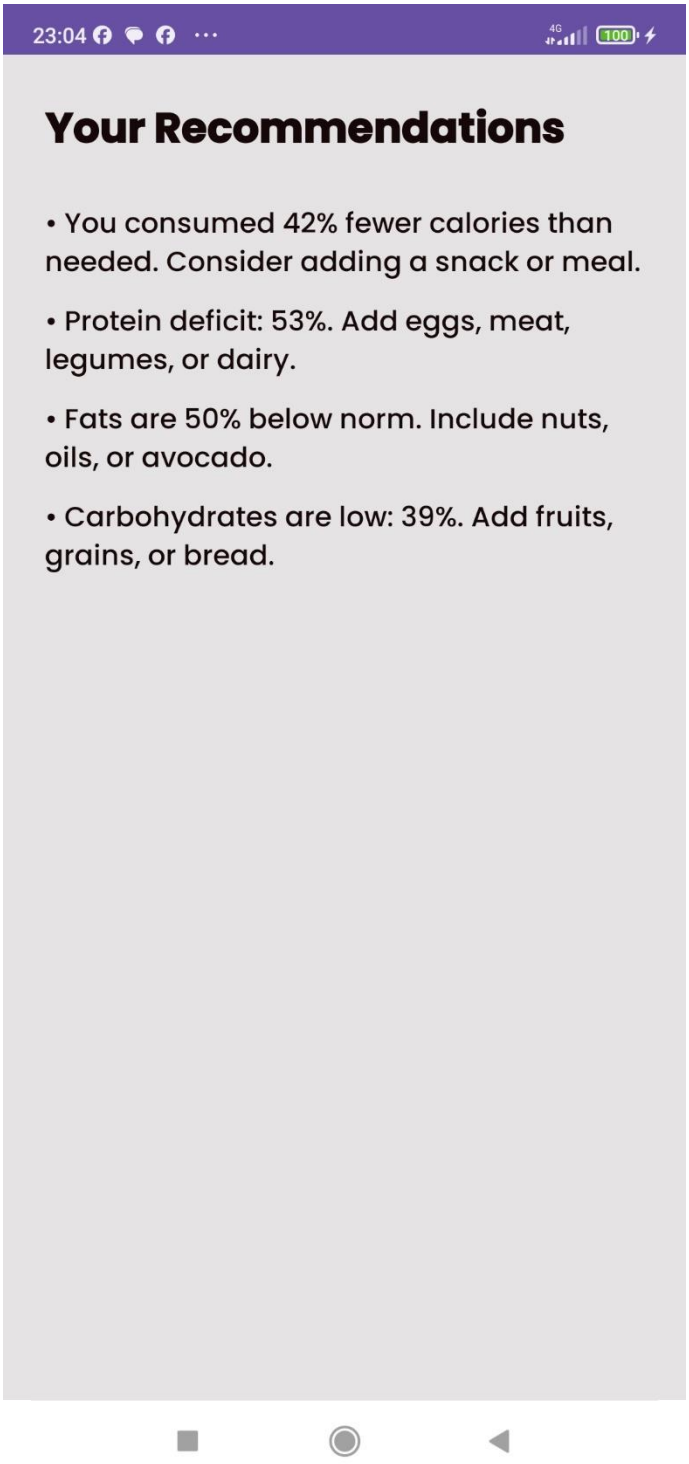


Рисунок 4.8 – Екран персоналізованих рекомендацій

Після виведення екрану персоналізованих рекомендацій можна побачити на ньому аналіз виконання норми калорійності, білків, жирів та вуглеводів на добу. А також рекомендації щодо того, що потрібно додати до раціону щоб доповнити його.

Додамо декілька продуктів до добового раціону для того щоб продемонструвати зміни індикатора збалансованості та персоналізованих рекомендацій.

Головний екран зі зміненими даними наведено на рисунку 4.9.

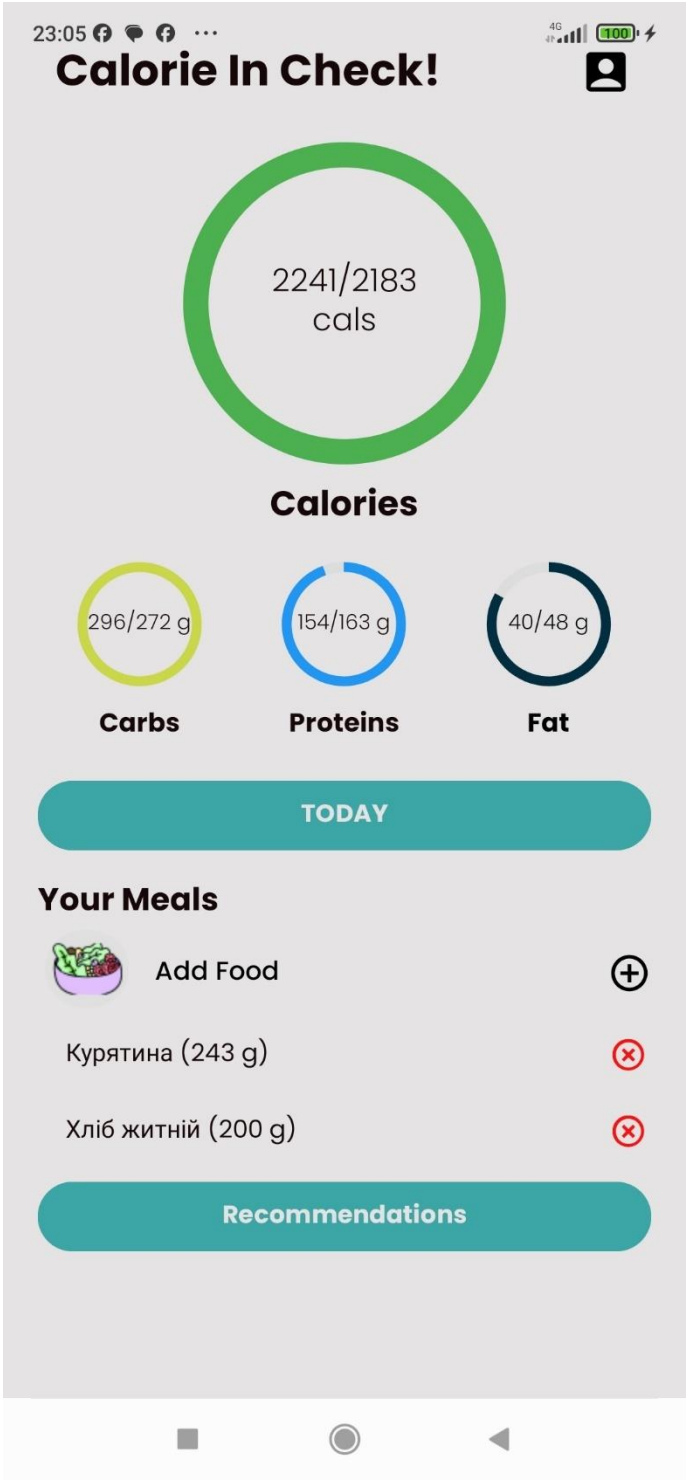


Рисунок 4.9 – Головний екран зі зміненими даними

Можна побачити, що після додавання необхідної кількості їжі до добового раціону змінився індикатор його збалансованості на зелений колір, тобто це показує, що раціон збалансований і є в нормі. Також після цих дій було змінено дані на екрані персоналізованих рекомендацій. Екран персоналізованих рекомендацій наведено на рисунку 4.10.

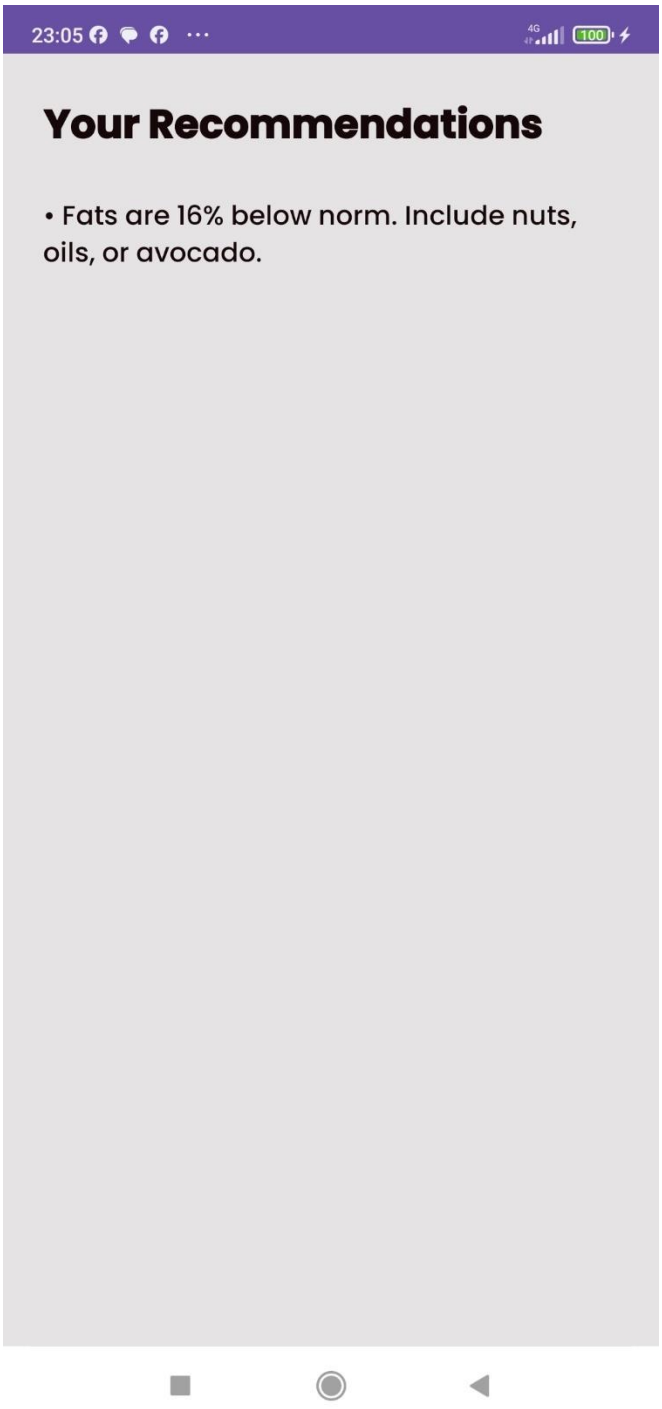


Рисунок 4.10 – Екран персоналізованих рекомендацій після зміни раціону

Також у застосунку є змога редагувати профіль користувача, змінюючи дані про вагу, зріст, вік, стать, ім’я. Це може бути корисно у разі схуднення, або зміни у віці. Зміна цих параметрів буде змінювати результат обчислення кількості необхідної калорійності та нутрієнтів, спожитих за день.

Екран зміни профілю користувача зображено на рисунку 4.11.

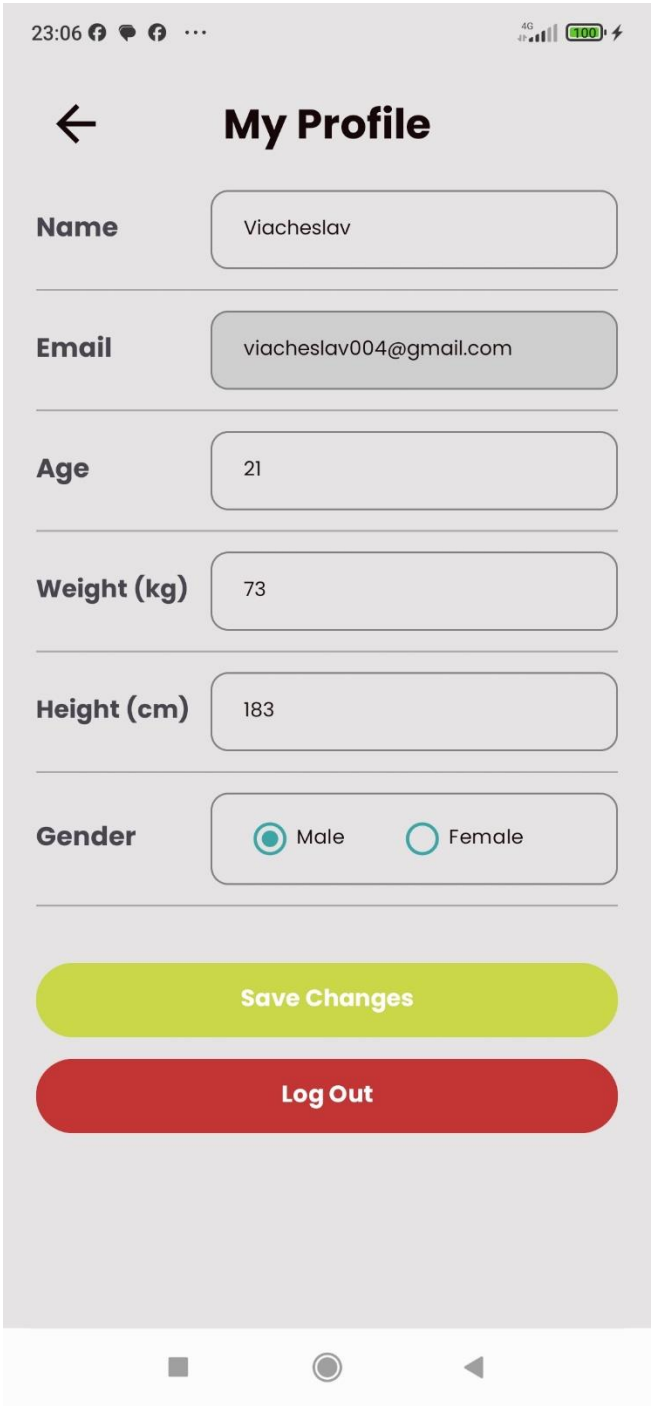


Рисунок 4.11 – Екран зміни профілю пористувача

Таким чином, було проведено тестування мобільного застосунку для моніторингу збалансованого харчування. Продемонстровано реалізований функціонал, перевірено його роботу в реальних сценаріях використання. Розроблений застосунок дозволяє вести облік спожитих продуктів, автоматично розраховувати КБЖВ, оцінювати збалансованість раціону та отримувати персоналізовані рекомендації. Отже, застосунок успішно виконує завдання, поставлені на початку розробки.

4.3 Висновки

Таким чином, у межах четвертого розділу було проведено тестування мобільного застосунку для моніторингу збалансованого харчування. Проаналізовано методи тестування, обґрунтовано вибір ручного функціонального підходу як найбільш релевантного для перевірки інтерактивних сценаріїв та логіки обчислень.

У ході тестування перевірено коректність роботи основного функціоналу: авторизації, додавання їжі, обчислення КБЖВ, візуального відображення прогресу та генерації персоналізованих рекомендацій. Продемонстровано зміну кольору індикатора збалансованості раціону залежно від даних, а також динамічне оновлення порад. Перевірено стабільність застосунку на різних екранах і за різних сценаріїв використання, включаючи зміну профілю користувача.

Результати тестування підтвердили, що реалізовані компоненти працюють стабільно, логіка функціонування відповідає поставленим вимогам, а застосунок виконує основне призначення – допомагає користувачу ефективно контролювати харчування та підтримувати збалансований раціон.

ВИСНОВКИ

У рамках кваліфікаційної роботи були розроблені засоби мобільної системи для моніторингу збалансованого харчування. Для розробки використовувалося середовище розробки Android Studio. Реалізація кваліфікаційної роботи відповідає методичним вказівкам [24, 25].

Під час виконання кваліфікаційної роботи було проведено аналіз стану питання розробки застосунків для моніторингу збалансованого харчування. Розглянуті існуючі аналоги, проведено порівняльний аналіз із власною розробкою, в результаті чого було доведено актуальність власної розробки, та її переваги над аналогами, після чого було проведено постановку задач дослідження.

Під час варіантного аналізу мов програмування було обрано мову програмування Kotlin.

У кваліфікаційній роботі було зроблено:

– розроблено модель застосунку для моніторингу харчування;

– розроблено метод оцінки харчової збалансованості раціону;

– розроблено алгоритми обліку КБЖВ та генерації рекомендацій;

– розроблено модулі застосунку для введення харчових даних та перегляду статистики;

– проведено тестування розробленого застосунку.

Було розроблено алгоритми роботи застосунку та наведено у вигляді UML діаграм.

Тестування довело працездатність розробленого застосунку. Розроблений застосунок дозволяє проводити моніторинг харчування, оцінки харчової збалансованості раціону, отримувати персоналізовані рекомендації, отже виконує поставлені на початку розробки завдання.

ЛІТЕРАТУРА

1. Трач В.О., Ткаченко О.М. Розробка мобільного застосунку для моніторингу збалансованого харчування // *Матеріали студентської науково-практичної конференції Вінницького національного технічного університету.* – Вінниця: ВНТУ, 2024. – С. 1–3.

2. Кравченко Д.О. Мобільні застосунки у сфері дієтології та фітнесу: аналітичний огляд. – Київ: НАУ, 2021. – 95 с.

3. Пономаренко М.В. Системи моніторингу фізіологічного стану людини: моделі та алгоритми. – Вінниця: ВНТУ, 2020. – 138 с.

4. YAZIO – Calorie Counter App [Електронний ресурс]. – Режим доступу: https://www.yazio.com

5. MyFitnessPal – Nutrition Tracker [Електронний ресурс]. – Режим доступу: https://www.myfitnesspal.com

6. Lifesum – Healthy Eating App [Електронний ресурс]. – Режим доступу: https://www.lifesum.com

7. Всесвітня організація охорони здоров’я. Healthy diet. [Електронний ресурс] – Режим доступу: https://www.who.int/news-room/fact-sheets/detail/healthy-diet (дата звернення: 15.05.2025).

8. Android Developers [Електронний ресурс]. – Режим доступу: https://developer.android.com

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

10. Jetpack Compose: Modern toolkit for building native Android UI [Електронний ресурс]. – Режим доступу: https://developer.android.com/jetpack/compose

11. Room Persistence Library [Електронний ресурс]. – Режим доступу: https://developer.android.com/training/data-storage/room

12. Oracle. The Java™ Tutorials: Learning the Java Language [Електронний ресурс]. – Режим доступу: https://docs.oracle.com/javase/tutorial/java/

13. Flutter documentation. Introduction to Flutter [Електронний ресурс]. – Режим доступу: https://docs.flutter.dev/get-started/install

14. Гнатюк С.П., Барабанов В.В. Основи побудови мобільних застосунків: навч. посіб. – Київ: КНУ, 2020. – 148 с.

15. Головач С.М. Програмування на Kotlin для Android. – Харків: Фактор, 2021. – 278 с.

16. Король М.В. Інтерфейси користувача: принципи побудови та тестування. – Львів: ЛНУ, 2022. – 204 с.

17. Мороз І.О. Архітектурні підходи до побудови мобільних інформаційних систем у галузі здоров'я. – Чернівці: ЧНУ, 2021. – 168 с.

18. ISO/IEC 25010:2011. Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models.

19. Литвиненко І.І., Шевченко В.А. Тестування програмного забезпечення. – Вінниця: ВНТУ, 2021. – 152 с.

20. Куліш А.І. Аналіз і обробка даних у мобільних застосунках. – Тернопіль: ТНТУ, 2022. – 164 с.

21. Шевченко В.А., Литвиненко І.І. Тестування програмного забезпечення. – Вінниця: ВНТУ, 2021. – 152 с.

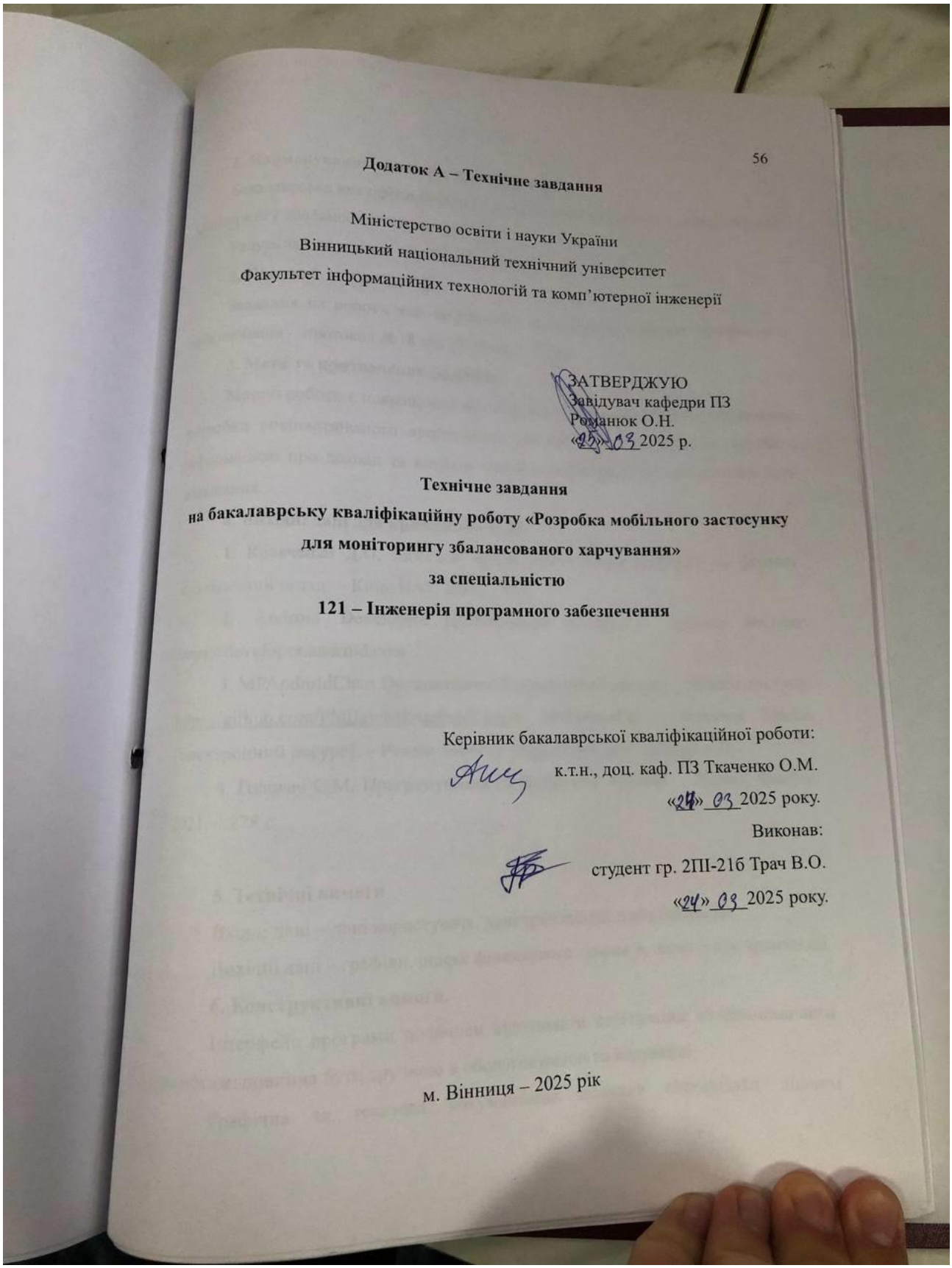
22. Nielsen J. Usability Engineering. – Academic Press, 1993. – 362 p.

23. Поляков С.В. Тестування програмного забезпечення: теорія і практика. – Харків: ХНУРЕ, 2020. – 188 с.

24. Методичні вказівки до виконання курсових проєктів з дисципліни "Проєктний практикум" для студентів спеціальності 121 "Інженерія програмного забезпечення" / Уклад. В.В. Войтко, Г. О. Черноволик – Вінниця: ВНТУ, 2022. – 44 с.

25. Лисенко Г.Л. Методичні вказівки до оформлення курсових проектів (робіт) у Вінницькому національному технічному університеті /Уклад. Г.Л. Лисенко, А.Г. Буда, Р.Р. Обертюх, – Вінниця: ВНТУ, 2006. – 60 с.

ДОДАТКИ



Додаток А – Технічне завдання

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

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О.Н.
«25» 03 2025 р.

Технічне завдання
на бакалаврську кваліфікаційну роботу «Розробка мобільного застосунку для моніторингу збалансованого харчування»
за спеціальністю
121 – Інженерія програмного забезпечення

Керівник бакалаврської кваліфікаційної роботи:
к.т.н., доц. каф. ПЗ Ткаченко О.М.
«24» 03 2025 року.

Виконав:
студент гр. 2ПІ-21б Трач В.О.
«24» 03 2025 року.

м. Вінниця – 2025 рік

57

1. Найменування та галузь застосування

Бакалаврська кваліфікаційна робота: «Розробка мобільного застосунку для моніторингу збалансованого харчування».

Галузь застосування – мобільні технології.

2. Підстава для розробки.

Завдання на роботу, яке затверджене на засіданні кафедри програмного забезпечення – протокол №18 від 18 лютого 2025 р.

3. Мета та призначення розробки.

Метою роботи є покращення процесу ведення особистих фінансів шляхом розробки спеціалізованого програмного забезпечення, що дозволяє керувати інформацією про доходи та витрати, формувати бюджет та відстежувати його виконання.

4. Вихідні дані для проведення НДР

1. Кравченко Д.О. Мобільні застосунки у сфері дієтології та фітнесу: аналітичний огляд. – Київ: НАУ, 2021. – 95 с.

2. Android Developers [Електронний ресурс]. – Режим доступу: https://developer.android.com

3. MPAndroidChart Documentation [Електронний ресурс]. – Режим доступу: https://github.com/PhilJay/MPAndroidChart4. MyFitnessPal – Nutrition Tracker [Електронний ресурс]. – Режим доступу: https://www.myfitnesspal.com

4. Головач С.М. Програмування на Kotlin для Android. – Харків: Фактор, 2021. – 278 с.

5. Технічні вимоги

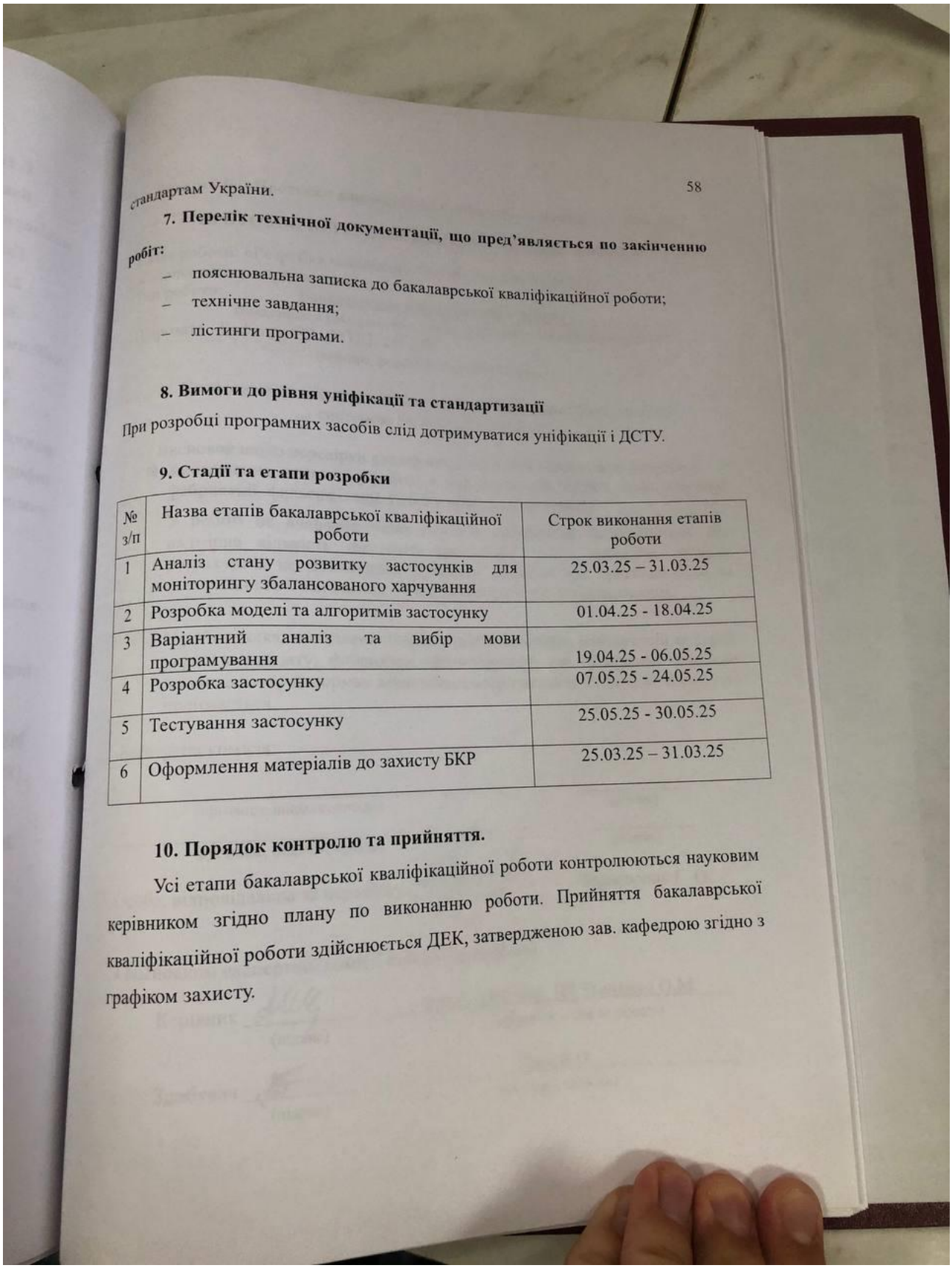
Вхідні дані – дані користувача, дані транзакцій, план бюджету.

Вихідні дані – графіки, індекс фінансового здоров'я, записи про транзакції.

6. Конструктивні вимоги.

Інтерфейс програми поовинен відповідати естетичним та ергономічним вимогам, повинна бути зручною в обслуговуванні та керуванні.

Графічна та текстова документація повинна відповідати діючим



стандартам України. 58

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

- пояснювальна записка до бакалаврської кваліфікаційної роботи;
- 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 бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи
1	Аналіз стану розвитку застосунків для моніторингу збалансованого харчування	25.03.25 – 31.03.25
2	Розробка моделі та алгоритмів застосунку	01.04.25 - 18.04.25
3	Варіантний аналіз та вибір мови програмування	19.04.25 - 06.05.25
4	Розробка застосунку	07.05.25 - 24.05.25
5	Тестування застосунку	25.05.25 - 30.05.25
6	Оформлення матеріалів до захисту БКР	25.03.25 – 31.03.25

10. Порядок контролю та прийняття.

Усі етапи бакалаврської кваліфікаційної роботи контролюються науковим керівником згідно плану по виконанню роботи. Прийняття бакалаврської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком захисту.

Додаток Б – Протокол перевірки на наявність текстових запозичень

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: «Розробка мобільного застосунку для моніторингу збалансованого харчування»

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

Підрозділ: Кафедра ПЗ, ФІТКІ, 2П-21б
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 8.1%

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 _____ к.т.н., доц. каф. ПЗ Ткаченко О.М.
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Трач В.О.
(підпис) (прізвище, ініціали)

Додаток В – Лістинг програми

Лістинг файлу MainActivity.kt

```
package com.example.calorie_diary

import android.content.Intent
import android.graphics.Color
import android.os.Bundle
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.Button
import android.widget.ImageView
import android.widget.LinearLayout
import android.widget.ProgressBar
import android.widget.TextView
import android.widget.Toast
import androidx.activity.enableEdgeToEdge
import androidx.appcompat.app.AppCompatActivity
import androidx.core.view.ViewCompat
import androidx.core.view.WindowInsetsCompat
import androidx.recyclerview.widget.RecyclerView
import androidx.recyclerview.widget.LinearLayoutManager
import com.example.calorie_diary.data.DBHelper
import com.example.calorie_diary.data.source.FoodSource
import com.example.calorie_diary.data.model.CalorieDiaries
import com.example.calorie_diary.data.model.TodayEatingHistory
import com.example.calorie_diary.data.model.User
import com.example.calorie_diary.util.BalanceEvaluator
import com.example.calorie_diary.util.Calculator
import com.example.calorie_diary.util.StringDate
import com.example.calorie_diary.util.SystemBar
import com.google.android.material.progressindicator.CircularProgressIndicator

class MainActivity : AppCompatActivity(), View.OnClickListener {
```

```
private lateinit var db: DBHelper
private lateinit var calculator: Calculator
private lateinit var stringDate: StringDate

private lateinit var profileButton: ImageView
private lateinit var addFoodButton: LinearLayout
private lateinit var caloriesProgress: BalanceRingView
private lateinit var caloriesText: TextView
private lateinit var carbohydrateProgress: ProgressBar
private lateinit var carbohydrateText: TextView
private lateinit var proteinsProgress: ProgressBar
private lateinit var proteinsText: TextView
private lateinit var fatProgress: ProgressBar
private lateinit var fatText: TextView

private lateinit var recyclerView: RecyclerView
private lateinit var todayEatingHistoryArrayList: ArrayList<TodayEatingHistory>
private lateinit var todayEatingHistoryAdapter: TodayEatingHistoryAdapter

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    enableEdgeToEdge()
    setContentView(R.layout.activity_main)

    SystemBar().setSystemBarColor(this)

    findViewById<Button>(R.id.button_recommendations).setOnClickListener {
        val intent = Intent(this, RecommendationActivity::class.java)
        startActivity(intent)
    }

    db = DBHelper(this, null)
    profileButton = findViewById(R.id.profile)
    addFoodButton = findViewById(R.id.add_food_btn)
```

```
    profileButton.setOnClickListener(this)
    addFoodButton.setOnClickListener(this)

    caloriesProgress = findViewById(R.id.progress_cals)
    carbohydrateProgress = findViewById(R.id.progress_carbs)
    proteinsProgress = findViewById(R.id.progress_protein)
    fatProgress = findViewById(R.id.progress_fat)

    caloriesText = findViewById(R.id.cals_text)
    carbohydrateText = findViewById(R.id.carbs_text)
    proteinsText = findViewById(R.id.protein_text)
    fatText = findViewById(R.id.fat_text)

    val userId: Int? = db.getCurrentUserId()
    if (userId != null) {
        startCaloriDiariesToday(userId)
        showCalorieDiariesToday(userId)
        showEatingHistoryToday(userId)
        addFoodData()
    } else {
        val intent = Intent(this@MainActivity, LoginActivity::class.java)
        startActivity(intent)
        finish()
    }
}

override fun onClick(v: View?) {
    if (v?.id == R.id.profile) {
        val intent = Intent(this@MainActivity, EditProfileActivity::class.java)
        startActivity(intent)
    } else if (v?.id == R.id.add_food_btn) {
        val intent = Intent(this@MainActivity, SearchFoodActivity::class.java)
        startActivity(intent)
    }
}
```

```
private fun startCaloriDiariesToday(userId: Int) {
    stringDate = StringDate()
    var calorieDiaries: CalorieDiaries? =
        db.getCalorieDiariesByDate(userId, stringDate.getCurrentDate())
    if (calorieDiaries == null) {
        val user: User? = db.getUserById(userId)
        if (user != null) {
            calculator = Calculator(user.gender, user.weight, user.height, user.age)
            calorieDiaries = CalorieDiaries(
                userId, stringDate.getCurrentDate(),
                0.0, calculator.maxCalories(),
                0.0, calculator.maxCarbohydrate(),
                0.0, calculator.maxProteins(),
                0.0, calculator.maxFat()
            )
            db.addCalorieDiaries(calorieDiaries)
        } else {
            val intent = Intent(this@MainActivity, LoginActivity::class.java)
            startActivity(intent)
            finish()
        }
    }
}

private fun showCalorieDiariesToday(userId: Int) {
    stringDate = StringDate()
    var calorieDiaries: CalorieDiaries? =
        db.getCalorieDiariesByDate(userId, stringDate.getCurrentDate())
    if (calorieDiaries != null) {
        // Calories
        val balance = BalanceEvaluator.evaluateBalance(calorieDiaries)
        val calorieBalance = balance.firstOrNull { it.nutrientName == "Calories" }
        caloriesProgress.balance = calorieBalance
        caloriesText.text = buildString {
```

```
    append(calorieDiaries.progressCalories.toInt().toString())
    append("/")
    append(calorieDiaries.maxCalories.toInt().toString())
    append("\ncals")
}
// Carbohydrate
carbohydrateProgress.max = calorieDiaries.maxCarbohydrate.toInt()
carbohydrateProgress.progress = calorieDiaries.progressCarbohydrate.toInt()
carbohydrateText.text = buildString {
    append(calorieDiaries.progressCarbohydrate.toInt().toString())
    append("/")
    append(calorieDiaries.maxCarbohydrate.toInt().toString())
    append(" g")
}
// Proteins
proteinsProgress.max = calorieDiaries.maxProteins.toInt()
proteinsProgress.progress = calorieDiaries.progressProteins.toInt()
proteinsText.text = buildString {
    append(calorieDiaries.progressProteins.toInt().toString())
    append("/")
    append(calorieDiaries.maxProteins.toInt().toString())
    append(" g")
}
// Fat
fatProgress.max = calorieDiaries.maxFat.toInt()
fatProgress.progress = calorieDiaries.progressFat.toInt()
fatText.text = buildString {
    append(calorieDiaries.progressFat.toInt().toString())
    append("/")
    append(calorieDiaries.maxFat.toInt().toString())
    append(" g")
}

val sharedPreferences = getSharedPreferences("calorie_data", MODE_PRIVATE)
val editor = sharedPreferences.edit()
```

```
            editor.putFloat("totalCalories", calorieDiaries.progressCalories.toFloat())
            editor.putFloat("maxCalories", calorieDiaries.maxCalories.toFloat())
            editor.apply()

        } else {
            startCaloriDiariesToday(userId)
        }
    }

    private fun showEatingHistoryToday(userId: Int) {
        stringDate = StringDate()
        recyclerView = findViewById(R.id.today_eating_history_recyclerview)
        recyclerView.setHasFixedSize(true)
        recyclerView.layoutManager = LinearLayoutManager(this)

        todayEatingHistoryArrayList = db.getTodayEatingHistoryByDate(userId,
stringDate.getCurrentDate())
        todayEatingHistoryAdapter = TodayEatingHistoryAdapter(todayEatingHistoryArrayList)
        recyclerView.adapter = todayEatingHistoryAdapter

        todayEatingHistoryAdapter.onItemClick = {
            db.deleteEatingHistoryById(it.id, it.foodId, userId)
            showCalorieDiariesToday(userId)
            showEatingHistoryToday(userId)
        }
    }

    private fun addFoodData() {
        if (db.getAllFood().size == 0) {
            val foodSource = FoodSource()
            val foodArrayList = foodSource.getFoodArrayList()
            for (food in foodArrayList) {
                db.addFood(food)
            }
        }
```

```
    }

}

class TodayEatingHistoryAdapter(
    private val todayEatingHistoryArrayList: ArrayList<TodayEatingHistory>
): RecyclerView.Adapter<TodayEatingHistoryAdapter.TodayEatingHistoryViewHolder>() {

    var onItemClick: ((TodayEatingHistory) -> Unit)? = null

    class TodayEatingHistoryViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {
        val eatingHistoryText: TextView = itemView.findViewById(R.id.today_eating_history_text)
        val delete: ImageView = itemView.findViewById(R.id.delete)
    }

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
TodayEatingHistoryViewHolder {
        val view =
LayoutInflater.from(parent.context).inflate(R.layout.recyclerview_today_eating_history,
            parent, false)
        return TodayEatingHistoryViewHolder(view)
    }

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

    override fun onBindViewHolder(holder: TodayEatingHistoryViewHolder, position: Int) {
        val todayEatingHistory = todayEatingHistoryArrayList[position]
        holder.eatingHistoryText.text = buildString {
            append(todayEatingHistory.name)
            append(" (")
            append(todayEatingHistory.foodWeight.toString())
            append(" g)")
        }
```

```
        holder.delete.setOnClickListener {
            onItemClick?.invoke(todayEatingHistory)
        }
    }
}
```

Лістинг класу AddFoodActivity

```
package com.example.calorie_diary

import android.content.Intent
import android.os.Bundle
import android.text.Editable
import android.text.TextWatcher
import android.view.View
import android.widget.Button
import android.widget.EditText
import android.widget.ImageButton
import android.widget.ProgressBar
import android.widget.TextView
import androidx.activity.enableEdgeToEdge
import androidx.appcompat.app.AppCompatActivity
import androidx.core.view.ViewCompat
import androidx.core.view.WindowInsetsCompat
import com.example.calorie_diary.util.SystemBar
import com.example.calorie_diary.data.DBHelper
import com.example.calorie_diary.data.model.CalorieDiaries
import com.example.calorie_diary.data.model.EatingHistory
import com.example.calorie_diary.util.StringDate

class AddFoodActivity : AppCompatActivity(), View.OnClickListener {

    private lateinit var db: DBHelper

    private lateinit var foodNameTextView: TextView
    private lateinit var foodCaloriesTextView: TextView
```

```
private lateinit var foodAmountEditText: EditText
private lateinit var addFoodButton: Button
private lateinit var backButton: ImageButton
private lateinit var totalCaloriesTextView: TextView
private lateinit var totalCaloriesProgressBar: ProgressBar
private lateinit var carbsTextView: TextView
private lateinit var carbsProgressBar: ProgressBar
private lateinit var proteinTextView: TextView
private lateinit var proteinProgressBar: ProgressBar
private lateinit var fatTextView: TextView
private lateinit var fatProgressBar: ProgressBar

private var foodName: String = ""
private var foodCalories: Double = 0.0
private var foodProteins: Double = 0.0
private var foodFat: Double = 0.0
private var foodCarbohydrate: Double = 0.0
private var foodAmount: Int = 0

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    enableEdgeToEdge()
    setContentView(R.layout.activity_add_food)
    ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main)) { v, insets ->
        val systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars())
        v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom)
        insets
    }

    SystemBar().setSystemBarColor(this)

    db = DBHelper(this, null)

    foodNameTextView = findViewById(R.id.foodName)
    foodCaloriesTextView = findViewById(R.id.foodCalories)
```

```
foodAmountEditText = findViewById(R.id.foodAmount)
totalCaloriesTextView = findViewById(R.id.totalCaloriesText)
totalCaloriesProgressBar = findViewById(R.id.totalCaloriesProgressBar)
carbsTextView = findViewById(R.id.CarbsText)
carbsProgressBar = findViewById(R.id.carbsProgressBar)
proteinTextView = findViewById(R.id.ProteinText)
proteinProgressBar = findViewById(R.id.proteinProgressBar)
fatTextView = findViewById(R.id.FatText)
fatProgressBar = findViewById(R.id.fatProgressBar)

backButton = findViewById(R.id.backButton)
backButton.setOnClickListener(this)

addFoodButton = findViewById(R.id.addFoodButton)
addFoodButton.setOnClickListener(this)

// Ambil data food details
val foodId = intent.getIntExtra("foodId", 0)
val food = db.getFoodById(foodId)
if (food != null) {
    foodName = food.name
    foodCalories = food.calories
    foodCarbohydrate = food.carbohydrate
    foodProteins = food.proteins
    foodFat = food.fat

    // Tampilkan food details
    foodNameTextView.text = foodName
    foodCaloriesTextView.text = buildString {
        append(foodCalories.times(100).toInt().toString())
        append(" cals / 100 g")
    }

    updateFoodDetails()
```

```
    } else {
        val intent = Intent(this@AddFoodActivity, SearchFoodActivity::class.java)
        startActivity(intent)
        finish()
    }

    foodAmountEditText.addTextChangedListener(object : TextWatcher {
        override fun beforeTextChanged(s: CharSequence?, start: Int, count: Int, after: Int) {}
        override fun onTextChanged(s: CharSequence?, start: Int, before: Int, count: Int) {}
        override fun afterTextChanged(s: Editable?) {
            updateFoodDetails()
        }
    })
}

override fun onClick(v: View?) {
    if (v?.id == R.id.backButton) {
        val intent = Intent(this@AddFoodActivity, SearchFoodActivity::class.java)
        startActivity(intent)
    } else if (v?.id == R.id.addFoodButton) {
        addFoodToEatingHistory()
        val intent = Intent(this@AddFoodActivity, MainActivity::class.java)
        startActivity(intent)
        finish()
    }
}

private fun updateFoodDetails() {
    // Get food amount (handling empty input)
    foodAmount = if (foodAmountEditText.text.toString().isNotEmpty()) {
        foodAmountEditText.text.toString().toIntOrNull() ?: 0
    } else {
        100 // Default
    }
```

```
        // Get Calorie Diaries
        val userId = db.getCurrentUserId()
        if (userId != null) {
            val calorieDiaries: CalorieDiaries? = db.getCalorieDiariesByDate(userId,
StringDate().getCurrentDate())
            if (calorieDiaries !== null) {
                // Total Calories
                totalCaloriesProgressBar.max = calorieDiaries.maxCalories.toInt()
                totalCaloriesProgressBar.progress = foodCalories.times(foodAmount).toInt()
                totalCaloriesTextView.text = buildString {
                    append(foodCalories.times(foodAmount).toInt().toString())
                    append("/")
                    append(calorieDiaries.maxCalories.toInt().toString())
                    append(" Cals")
                }
                // Carbs
                carbsProgressBar.max = calorieDiaries.maxCarbohydrate.toInt()
                carbsProgressBar.progress = foodCarbohydrate.times(foodAmount).toInt()
                carbsTextView.text = buildString {
                    append(foodCarbohydrate.times(foodAmount).toInt().toString())
                    append("/")
                    append(calorieDiaries.maxCarbohydrate.toInt().toString())
                    append(" g")
                }
                // Protein
                proteinProgressBar.max = calorieDiaries.maxProteins.toInt()
                proteinProgressBar.progress = foodProteins.times(foodAmount).toInt()
                proteinTextView.text = buildString {
                    append(foodProteins.times(foodAmount).toInt().toString())
                    append("/")
                    append(calorieDiaries.maxProteins.toInt().toString())
                    append(" g")
                }
                // Fat
                fatProgressBar.max = calorieDiaries.maxFat.toInt()
```

```
            fatProgressBar.progress = foodFat.times(foodAmount).toInt()
            fatTextView.text = buildString {
                append(foodFat.times(foodAmount).toInt().toString())
                append("/")
                append(calorieDiaries.maxFat.toInt().toString())
                append(" g")
            }
        } else {
            val intent = Intent(this@AddFoodActivity, MainActivity::class.java)
            startActivity(intent)
            finish()
        }
    } else {
        val intent = Intent(this@AddFoodActivity, LoginActivity::class.java)
        startActivity(intent)
        finish()
    }
}

private fun addFoodToEatingHistory() {
    foodAmount = if (foodAmountEditText.text.toString().isNotEmpty()) {
        foodAmountEditText.text.toString().toIntOrNull() ?: 0
    } else {
        100 // Default
    }
    val userId = db.getCurrentUserId()
    val foodId = intent.getIntExtra("foodId", 0)
    if (userId != null) {
        // Add eating history
        val eatingHistory = EatingHistory(
            null,
            userId,
            StringDate().getCurrentDate(),
            foodId,
            foodAmount
```

```
            )

            db.addEatingHistory(eatingHistory)

            // Update calorie diaries
            db.updateCalorieDiariesProgress(
                userId,
                StringDate().getCurrentDate(),
                foodCalories.times(foodAmount),
                foodCarbohydrate.times(foodAmount),
                foodProteins.times(foodAmount),
                foodFat.times(foodAmount)
            )
        } else {
            val intent = Intent(this@AddFoodActivity, LoginActivity::class.java)
            startActivity(intent)
            finish()
        }
    }
}
```

Лістинг класу BalanceEvulator

```
package com.example.calorie_diary.util

import com.example.calorie_diary.data.model.CalorieDiaries

object BalanceEvaluator {

    enum class DeviationLevel {
        NORMAL, MODERATE, CRITICAL
    }

    data class NutrientBalance(
        val nutrientName: String,
        val consumed: Double,
```

```
        val recommended: Double,
        val deviationPercent: Double,
        val level: DeviationLevel
    )

    fun evaluateBalance(diary: CalorieDiaries): List<NutrientBalance> {
        return listOf(
            evaluate("Calories", diary.progressCalories, diary.maxCalories),
            evaluate("Proteins", diary.progressProteins, diary.maxProteins),
            evaluate("Fats", diary.progressFat, diary.maxFat),
            evaluate("Carbs", diary.progressCarbohydrate, diary.maxCarbohydrate)
        )
    }

    private fun evaluate(
        nutrientName: String,
        consumed: Double,
        recommended: Double
    ): NutrientBalance {
        val deviation = if (recommended > 0) ((consumed - recommended) / recommended) * 100 else
0.0
        val level = when {
            kotlin.math.abs(deviation) <= 10 -> DeviationLevel.NORMAL
            kotlin.math.abs(deviation) <= 25 -> DeviationLevel.MODERATE
            else -> DeviationLevel.CRITICAL
        }
        return NutrientBalance(nutrientName, consumed, recommended, deviation, level)
    }
}
```

Лістинг класу RecommendationGenerator

```
package com.example.calorie_diary.util

import com.example.calorie_diary.util.BalanceEvaluator.NutrientBalance
import com.example.calorie_diary.util.BalanceEvaluator.DeviationLevel
```

```
object RecommendationGenerator {

    fun generateRecommendations(balanceList: List<NutrientBalance>): List<String> {
        val recommendations = mutableListOf<String>()

        for (balance in balanceList) {
            val name = balance.nutrientName.lowercase()
            val deviation = balance.deviationPercent
            val level = balance.level

            if (level == DeviationLevel.MODERATE || level == DeviationLevel.CRITICAL) {
                val rec = if (deviation > 0) {
                    // Перевищення
                    when (name) {
                        "калорії", "calories" ->
                            "You have exceeded your daily calorie intake by ${deviation.toInt()}%. Consider reducing snacks or portion sizes."
                        "білки", "proteins" ->
                            "Protein intake is ${deviation.toInt()}% above normal. Limit meat, dairy or supplements."
                        "жири", "fats" ->
                            "Fat intake exceeds the norm by ${deviation.toInt()}%. Try reducing fried or fatty foods."
                        "вуглеводи", "carbs" ->
                            "Carbohydrate surplus: +${deviation.toInt()}%. Cut down on sweets, bread, or cereals."
                        else -> ""
                    }
                } else {
                    // Нестача
                    when (name) {
                        "калорії", "calories" ->
                            "You consumed ${-deviation.toInt()}% fewer calories than needed. Consider adding a snack or meal."
```

```
                "білки", "proteins" ->
                    "Protein deficit: ${-deviation.toInt()}%. Add eggs, meat, legumes, or dairy."
                "жири", "fats" ->
                    "Fats are ${-deviation.toInt()}% below norm. Include nuts, oils, or avocado."
                "вуглеводи", "carbs" ->
                    "Carbohydrates are low: ${-deviation.toInt()}%. Add fruits, grains, or bread."
                else -> ""
            }
        }
        if (rec.isNotBlank()) {
            recommendations.add(rec)
        }
    }
  }
  return recommendations
 }
}
```

Лістинг файлу SearchFoodActivity.kt

```
package com.example.calorie_diary

import android.content.Intent
import android.os.Bundle
import android.view.LayoutInflater
import android.view.ViewGroup
import androidx.activity.enableEdgeToEdge
import androidx.appcompat.app.AppCompatActivity
import androidx.core.view.ViewCompat
import androidx.core.view.WindowInsetsCompat
import androidx.recyclerview.widget.LinearLayoutManager
import androidx.recyclerview.widget.RecyclerView
import android.widget.Button
import android.widget.EditText
import android.widget.ImageButton
import android.widget.TextView
```

```
import com.example.calorie_diary.data.DBHelper
import com.example.calorie_diary.data.model.Food
import com.example.calorie_diary.util.SystemBar

class SearchFoodActivity : AppCompatActivity() {
    private lateinit var db: DBHelper
    private lateinit var typeFoodInput: EditText
    private lateinit var searchButton: Button
    private lateinit var foodRecyclerView: RecyclerView
    private lateinit var foodAdapter: FoodAdapter
    private lateinit var foods: List<Food>
    private lateinit var allFoods: List<Food>
    private lateinit var displayedFoods: MutableList<Food>
    private lateinit var backButton: ImageButton

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContentView(R.layout.activity_search_food)
        ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main)) { v, insets ->
            val systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars())
            v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom)
            insets
        }

        SystemBar().setSystemBarColor(this)

        db = DBHelper(this, null)

        // Inisialisasi RecyclerView dan Adapter
        foodRecyclerView = findViewById(R.id.foodRecyclerView)
        allFoods = db.getAllFood()
        foodAdapter = FoodAdapter(this, allFoods)
        foodRecyclerView.layoutManager = LinearLayoutManager(this)
        foodRecyclerView.adapter = foodAdapter
```

```
        // Inisialisasi EditText dan Button
        typeFoodInput = findViewById(R.id.typeFoodInput)
        searchButton = findViewById(R.id.searchButton)
        backButton = findViewById(R.id.backButton)

        db = DBHelper(this, null)
        val userId = db.getCurrentUserId()
        if (userId == null) {
            val intent = Intent(this@SearchFoodActivity, LoginActivity::class.java)
            startActivity(intent)
            finish()
        }

        // Set OnClickListener untuk searchButton
        searchButton.setOnClickListener {
            val searchText = typeFoodInput.text.toString().trim()
            foodAdapter.filter(searchText)
        }

        // Set OnClickListener untuk backButton
        backButton.setOnClickListener {
            val intent = Intent(this@SearchFoodActivity, MainActivity::class.java)
            startActivity(intent)
            finish()
        }

        val sharedPreferences = getSharedPreferences("calorie_data", MODE_PRIVATE)
        val totalCalories = sharedPreferences.getFloat("totalCalories", 0f)
        val maxCalories = sharedPreferences.getFloat("maxCalories", 0f)
        val textView2 = findViewById<TextView>(R.id.textView2)
        textView2.text = "${totalCalories.toInt()}/${maxCalories.toInt()} cals"
    }

}
```

```
data class Food(val name: String, val description: String, val calories: Double)

class FoodAdapter(private val context: SearchFoodActivity, private var foodList: List<Food>) :
    RecyclerView.Adapter<FoodAdapter.FoodViewHolder>() {

    private var filteredFoodList: List<Food> = foodList.toMutableList()

    class FoodViewHolder(itemView: android.view.View) : RecyclerView.ViewHolder(itemView) {
        val foodName: TextView = itemView.findViewById(R.id.food)
        val foodDesc: TextView = itemView.findViewById(R.id.food_desc)
    }

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): FoodViewHolder {
        val itemView = LayoutInflater.from(parent.context)
            .inflate(R.layout.recyclerview, parent, false)
        return FoodViewHolder(itemView)
    }

    override fun onBindViewHolder(holder: FoodViewHolder, position: Int) {
        val currentItem = filteredFoodList[position]
        holder.foodName.text = currentItem.name
        val caloriesPer100g = currentItem.calories.times(100).toInt()
        holder.foodDesc.text = "%d Cal, (100 g)".format(caloriesPer100g)

        holder.itemView.setOnClickListener { // Add click listener to itemView
            val intent = Intent(context, AddFoodActivity::class.java) // Create Intent
            intent.putExtra("foodId", currentItem.id) // Pass food name to AddFoodActivity
            context.startActivity(intent) // Start AddFoodActivity
        }
    }

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

```
    fun filter(text: String) {
        filteredFoodList = if (text.isEmpty()) {
            foodList
        } else {
            foodList.filter { it.name.toLowerCase().contains(text.toLowerCase()) }
        }
        notifyDataSetChanged()
    }
}
```

Лістинг класу Calculator

```
package com.example.calorie_diary.util

class Calculator(
    val gender: String,
    val weight: Int,
    val height: Int,
    val age: Int
) {

    private val carbohydratePercentage = 0.5
    private val proteinsPercentage = 0.3
    private val fatPercentage = 0.2

    fun maxCalories(): Double {
        val maxCalories: Double
        if (gender == "M") {
            maxCalories = 1.2 * ((10 * weight) + (6.5 * height) - (5 * age) + 5)
        } else {
            maxCalories = 1.2 * ((10 * weight) + (6.5 * height) - (5 * age) - 161)
        }
        return maxCalories
    }
```

```
    fun maxCarbohydrate(): Double {
        return (carbohydratePercentage * maxCalories()) / 4
    }

    fun maxProteins(): Double {
        return (proteinsPercentage * maxCalories()) / 4
    }

    fun maxFat(): Double {
        return (fatPercentage * maxCalories()) / 9
    }

}
```

Лістинг класу LoginActivity

```
package com.example.calorie_diary

import android.content.Intent
import android.os.Bundle
import android.view.View
import android.widget.Button
import android.widget.EditText
import android.widget.TextView
import androidx.activity.enableEdgeToEdge
import androidx.appcompat.app.AppCompatActivity
import androidx.core.view.ViewCompat
import androidx.core.view.WindowInsetsCompat
import com.example.calorie_diary.data.DBHelper
import com.example.calorie_diary.util.SystemBar

class LoginActivity : AppCompatActivity(), View.OnClickListener {

    private lateinit var db: DBHelper
    private lateinit var emailInput: EditText
    private lateinit var passwordInput: EditText
```

```
private lateinit var logInValidationError: TextView
private lateinit var logInError: TextView
private lateinit var logInButton: Button
private lateinit var createAccoutnLink: TextView

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    enableEdgeToEdge()
    setContentView(R.layout.activity_login)
    ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main)) { v, insets ->
        val systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars())
        v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom)
        insets
    }

    SystemBar().setSystemBarColor(this)

    db = DBHelper(this, null)
    emailInput = findViewById(R.id.email_input)
    passwordInput = findViewById(R.id.password_input)
    logInValidationError = findViewById(R.id.login_validation_error)
    logInError = findViewById(R.id.login_error)
    logInButton = findViewById(R.id.login_button)
    createAccoutnLink = findViewById(R.id.create_account_link)

    createAccoutnLink.setOnClickListener(this)
    logInButton.setOnClickListener(this)
    logInValidationError.visibility = View.GONE
    logInError.visibility = View.GONE

    val userId = db.getCurrentUserId()
    if (userId != null) {
        val intent = Intent(this@LoginActivity, MainActivity::class.java)
        startActivity(intent)
        finish()
```

```
        }
    }

    override fun onClick(v: View?) {
        if (v?.id == R.id.login_button) {
            logInValidationError.visibility = View.GONE
            logInError.visibility = View.GONE

            val email = emailInput.text.toString().trim()
            val password = passwordInput.text.toString().trim()

            if (email.isEmpty() || password.isEmpty()) {
                logInValidationError.visibility = View.VISIBLE
            } else if (!email.matches(android.util.Patterns.EMAIL_ADDRESS.toRegex())) {
                logInValidationError.visibility = View.VISIBLE
            } else {
                logInValidationError.visibility = View.GONE
                if (db.logIn(email, password)) {
                    val intent = Intent(this@LoginActivity, MainActivity::class.java)
                    startActivity(intent)
                    finish()
                } else {
                    logInError.visibility = View.VISIBLE
                }
            }
        } else if (v?.id == R.id.create_account_link) {
            val intent = Intent(this@LoginActivity, SignUpActivity::class.java)
            startActivity(intent)
            finish()
        }
    }
}
```

Додаток Д – Ілюстративна частина

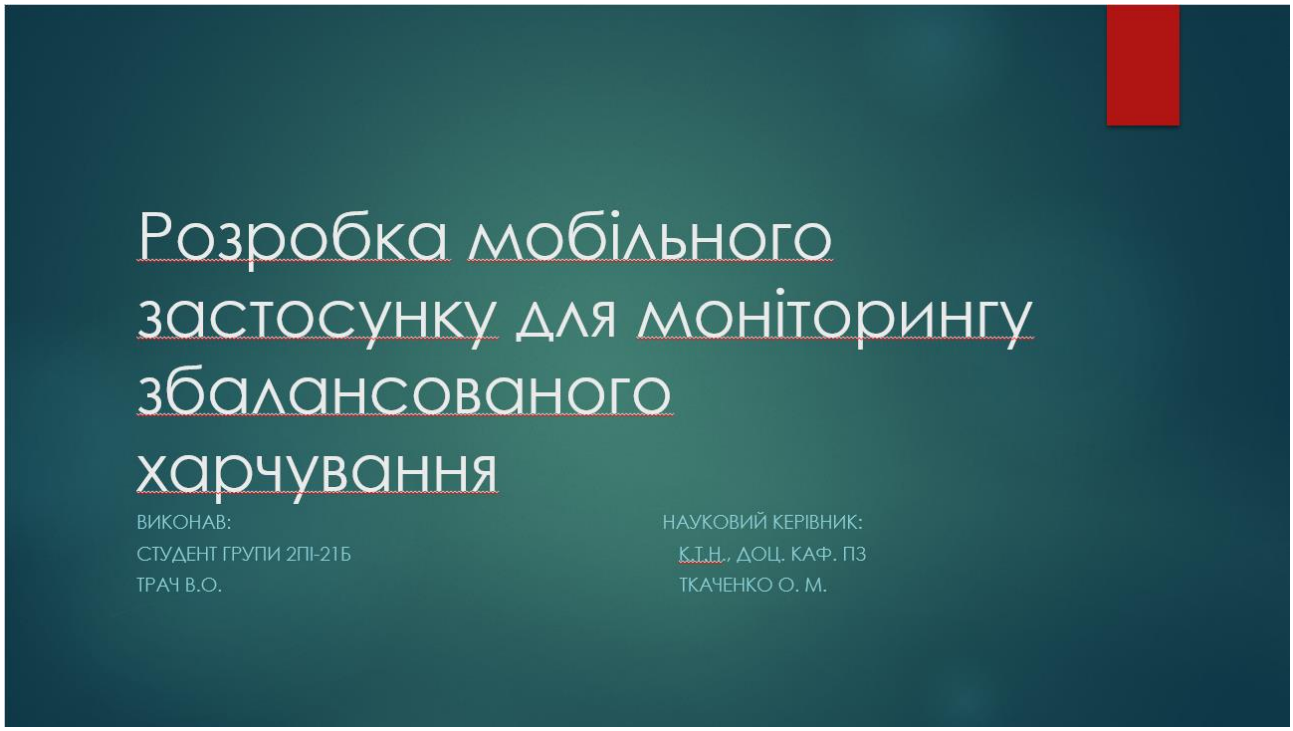


Рисунок Д.1 – Титульний слайд

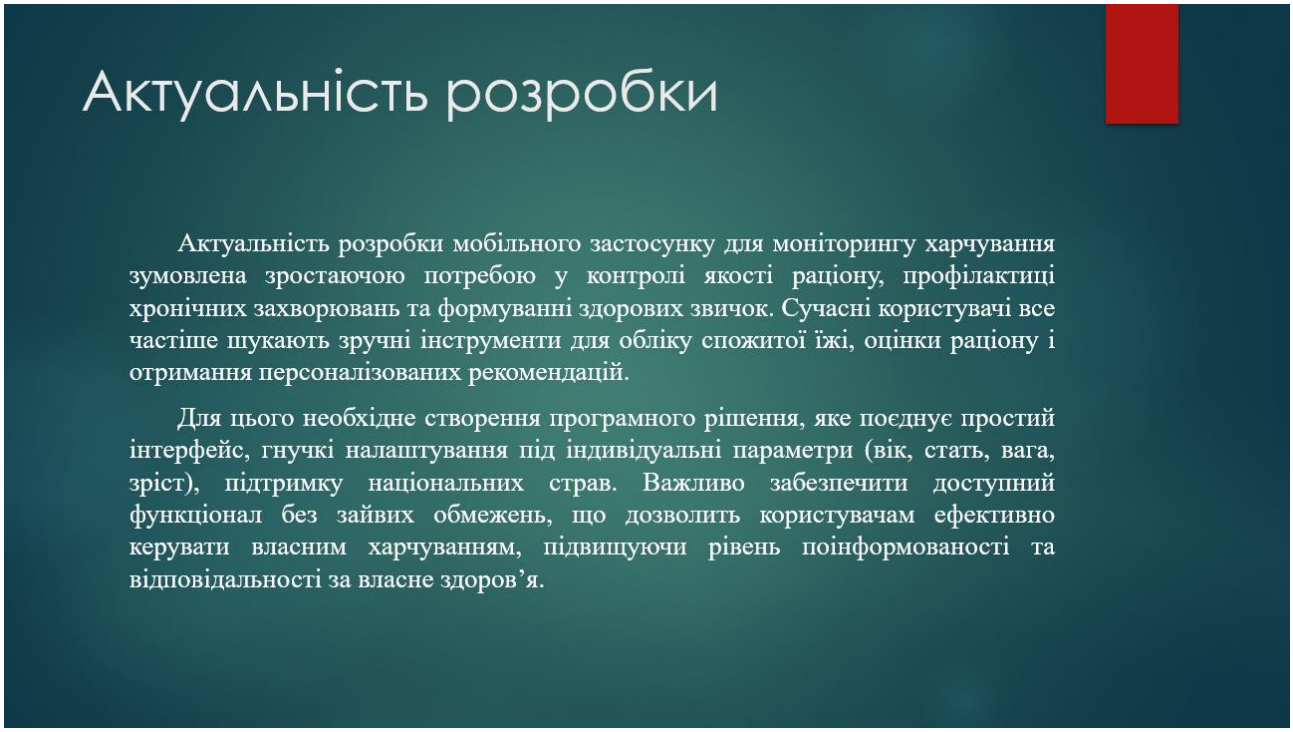


Рисунок Д.2 – Актуальність розробки

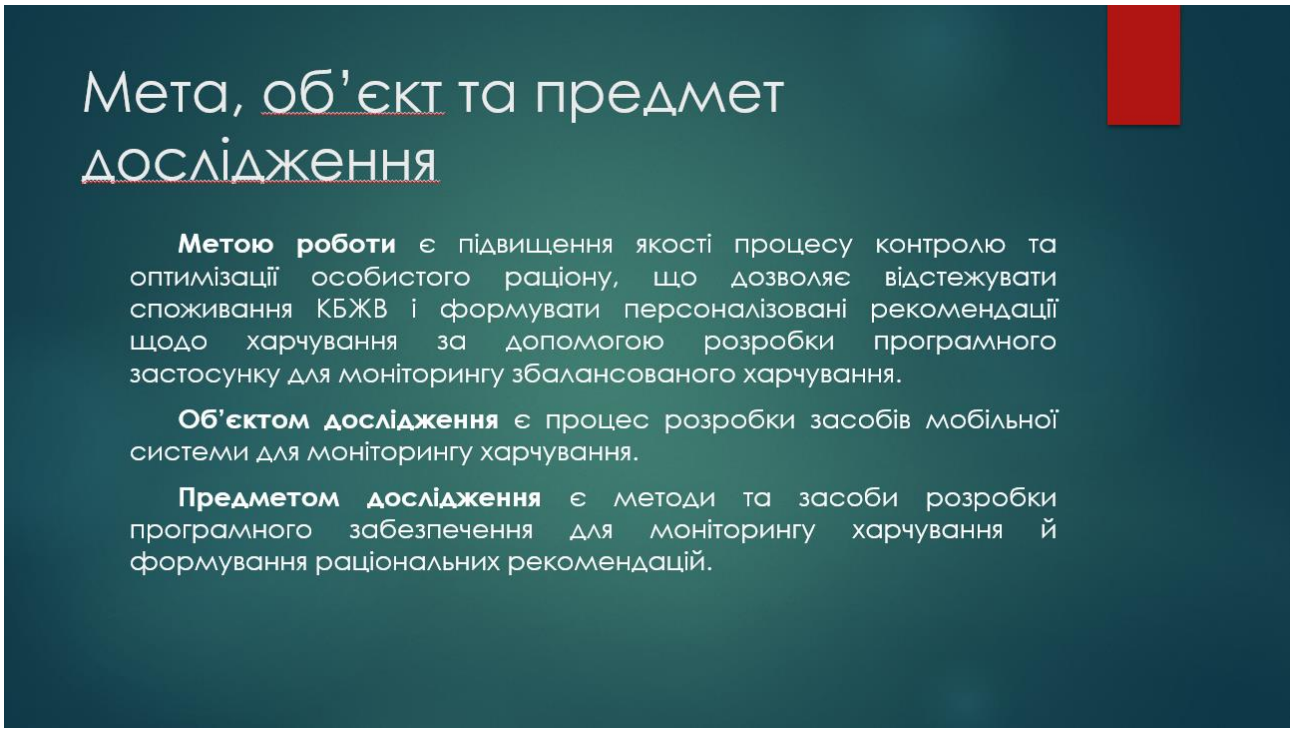


Рисунок Д.3 – Мета, об’єкт та предмет дослідження

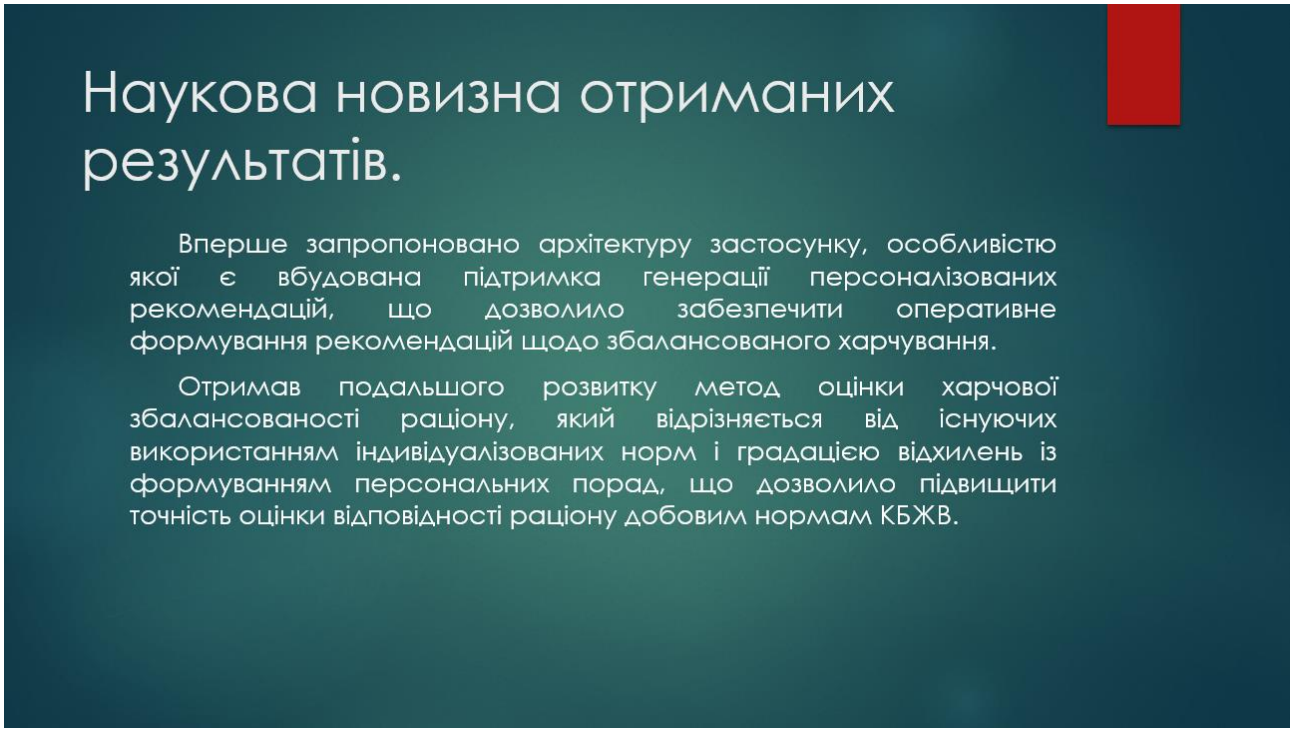


Рисунок Д.4 – Наукова новизна отриманих результатів

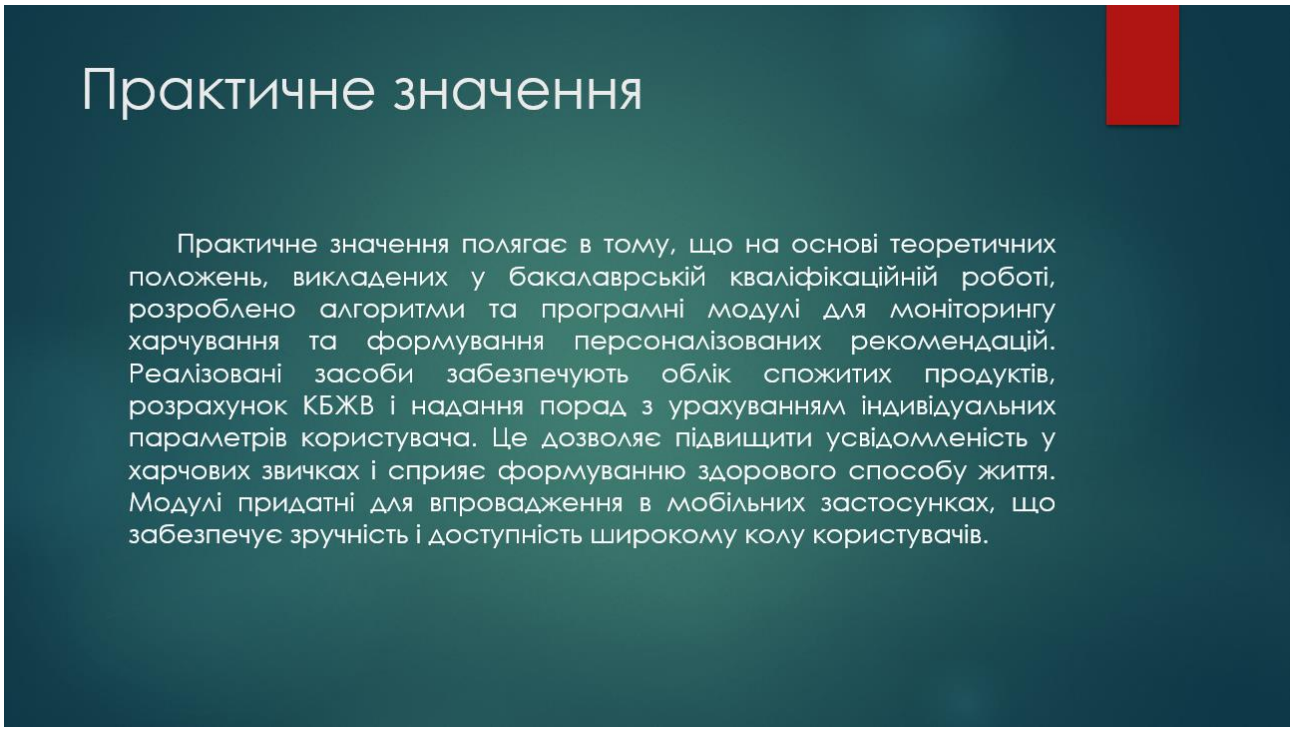


Рисунок Д.5 – Практичне значення

Рисунок Д.6 – Аналіз аналогів

Рисунок Д.7 – Блок-схема алгоритму оцінки харчової збалансованості раціону

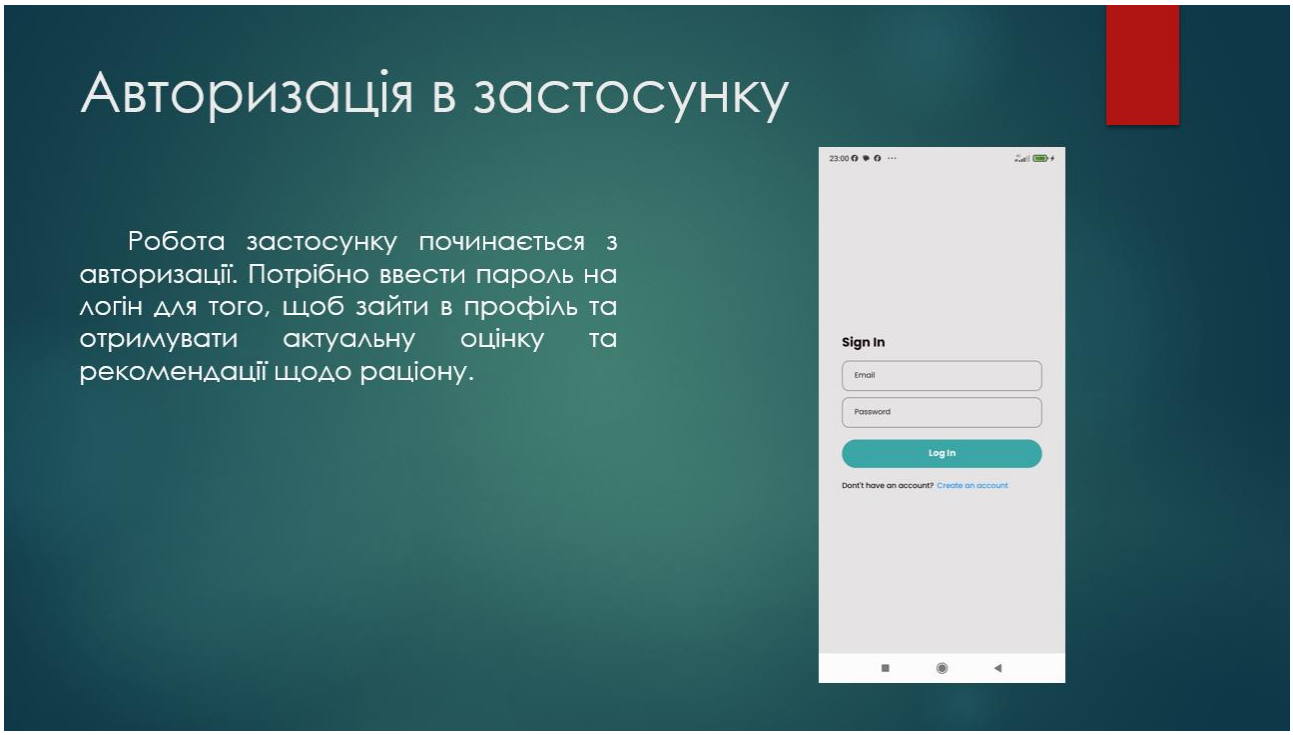


Рисунок Д.8 – Авторизація в застосунку

Головний екран застосунку

Головний екран застосунку містить всі відомості про спожиті за день калорії, білки, жири та вуглеводи. Виведено коло індикацію харчової збалансованості раціону та додано всю навігацію застосунку від змін даних про користувача до генерації персоналізованих рекомендацій.

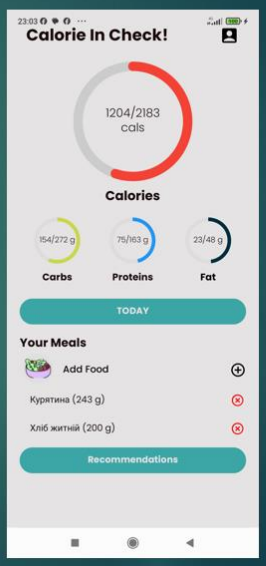


Рисунок Д.9 – Головний екран застосунку

Додавання їжі у денний раціон

Після виведення екрану додавання їжі буде надано всі відомості про калорійність та нутрієнти. А саме на шкалі зображено співвідношення калорійності продукту до потрібної калорійності на добу. Також це працює і з вуглеводами, білками та жирами. Також наявною є можливість ввести кількість спожитого продукту у грамах.

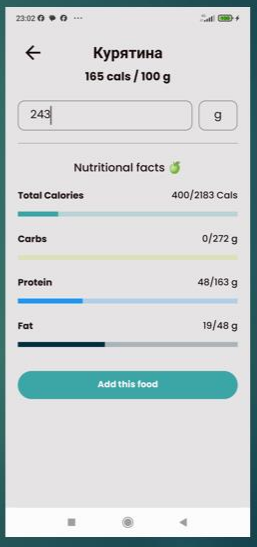


Рисунок Д.10 – Додавання їжі у денний раціон

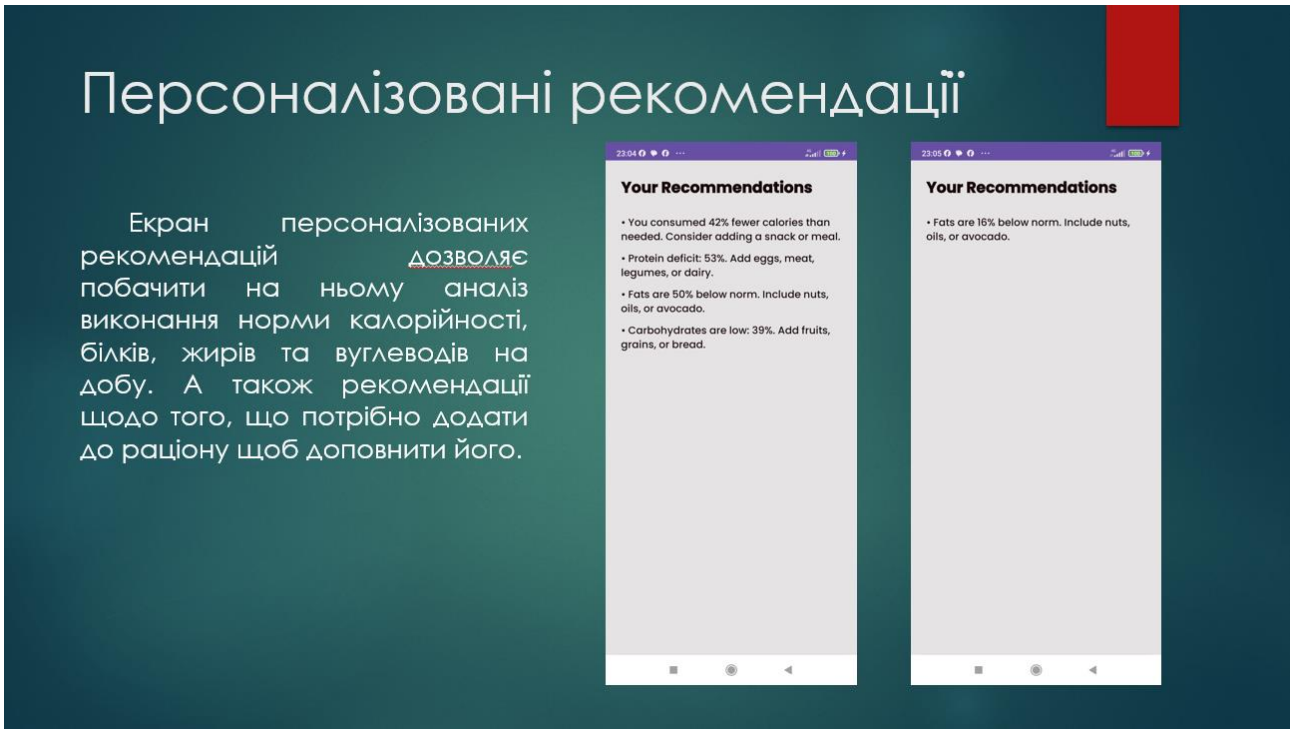


Рисунок Д.11 – Персоналізовані рекомендації

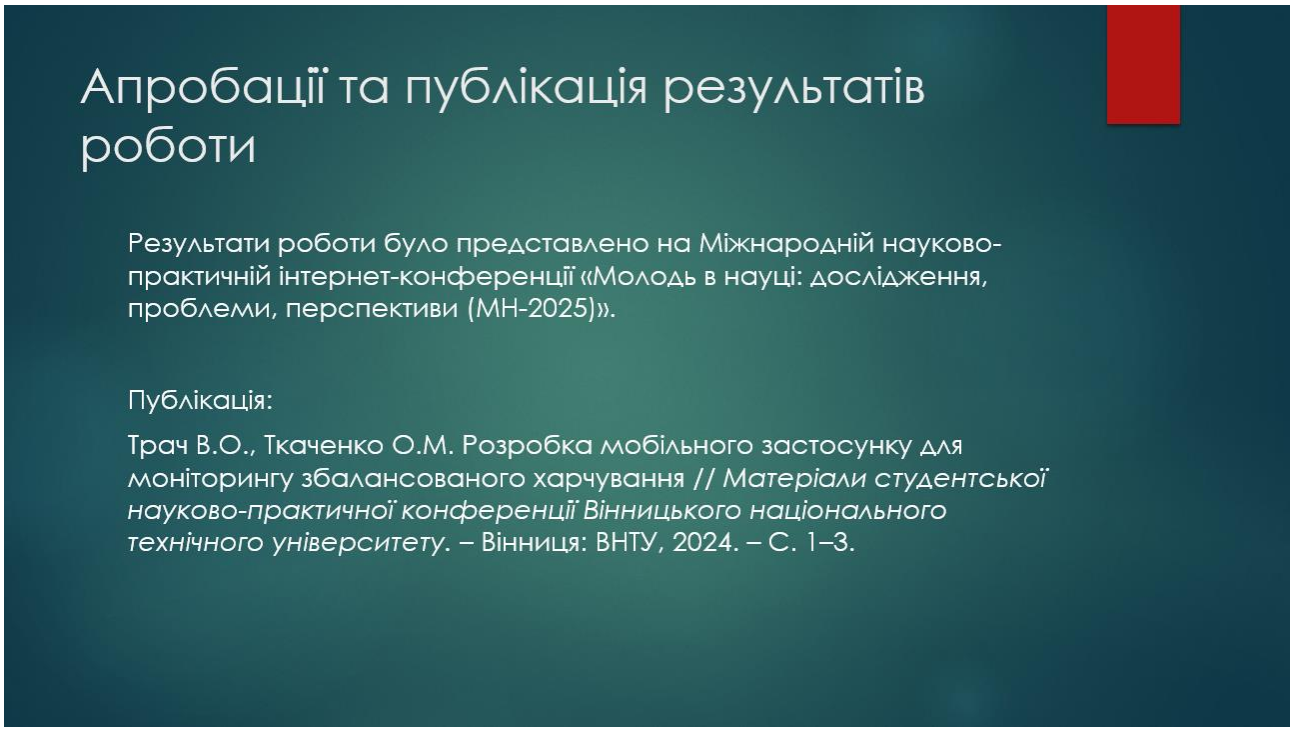


Рисунок Д.12 – Апробації та публікація результатів роботи

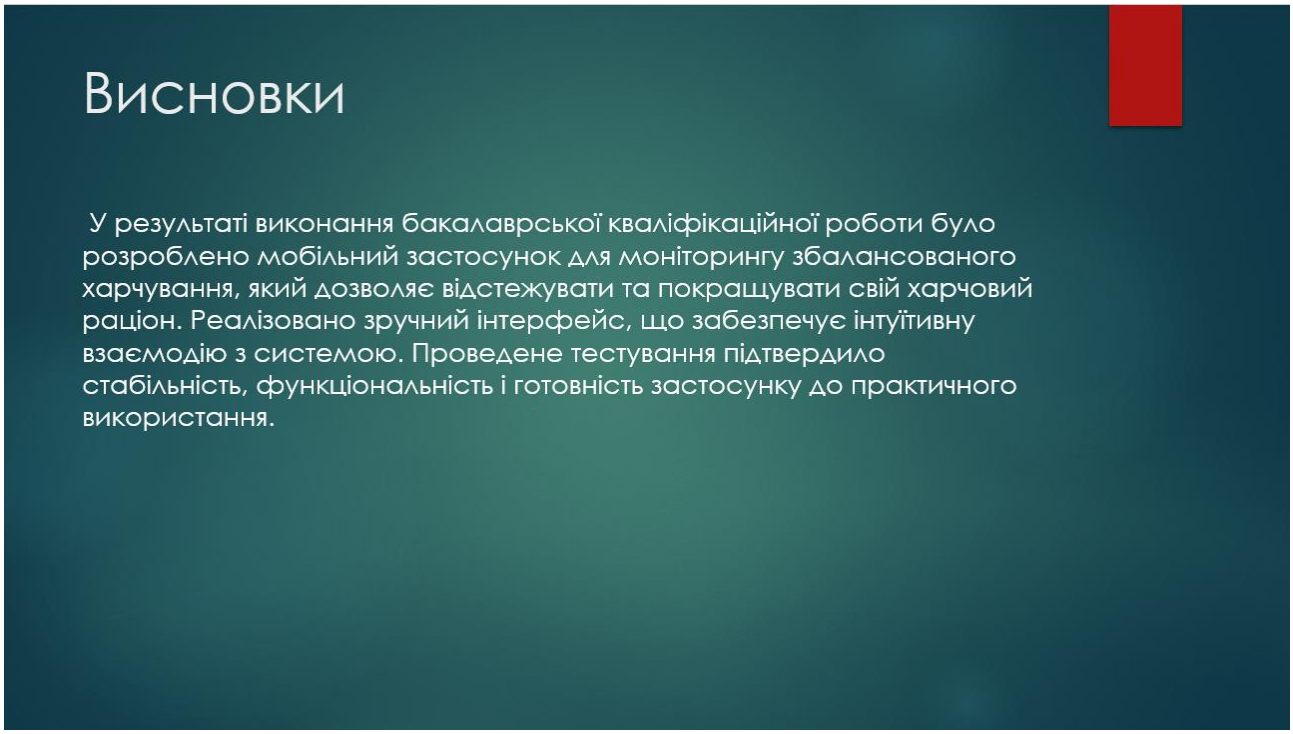


Рисунок Д.13 – Висновки

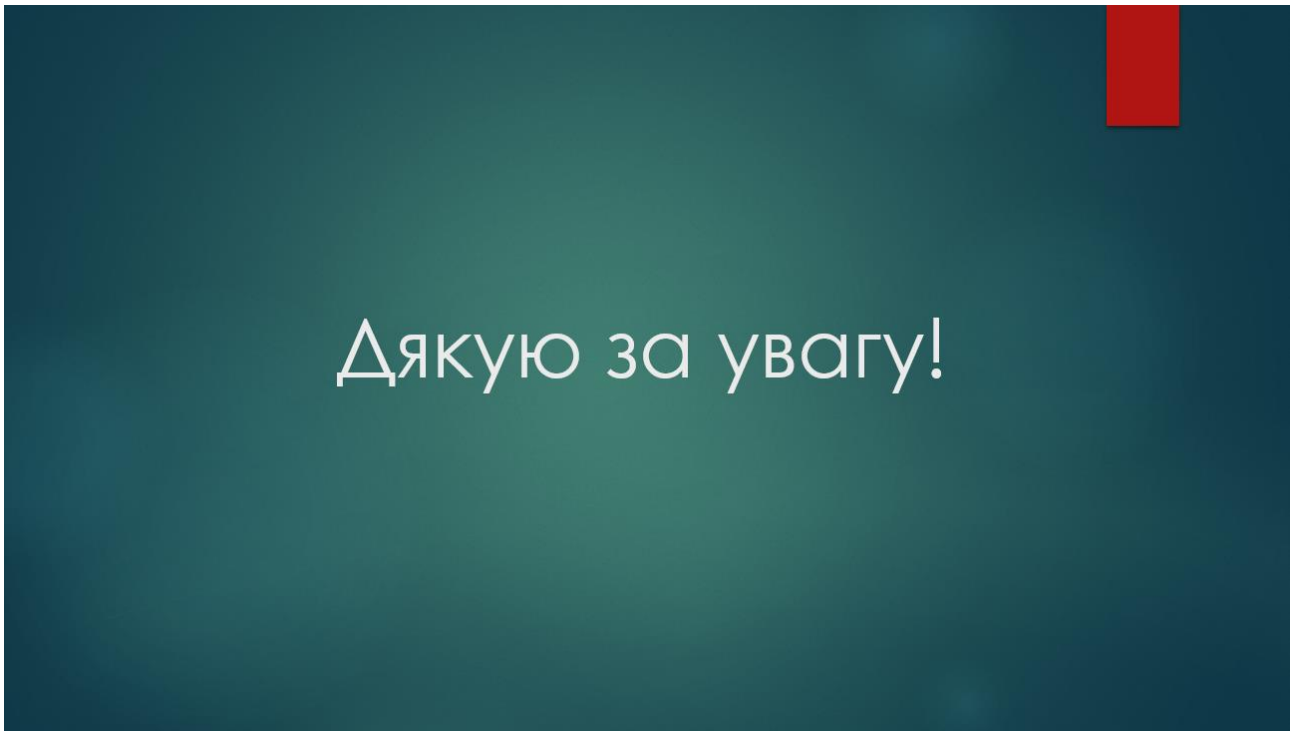


Рисунок Д.14 – Дякую за увагу