МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Державне некомерційне підприємство

«Державний університет» Київський авіаційний інститут»

Факультет комп’ютерних наук та технологій
Кафедра інженерії програмного забезпечення

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри

Олена ГРІНЕНКО
“____”_____2025р.

КВАЛІФІКАЦІЙНА РОБОТА
(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ЗДОБУВАЧА ОСВІТНЬОГО СТУПЕНЯ «МАГІСТР»

Тема: «Адаптивний алгоритм та застосунок для персоніфікованого підбору дієтчних продуктів»

Виконавець: Сутир Кирило Андрійович

Керівник: к.т.н., доцент кафедри ІПЗ, доцент Шибицька Наталія Миколаївна

Нормоконтролер: к.т.н., доцент кафедри ІПЗ, доцент Шибицька Наталія Миколаївна

Київ 2025

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет комп'ютерних наук та технологій
Кафедра інженерії програмного забезпечення
Освітній ступінь магістр
Спеціальність 121 «Інженерія програмного забезпечення»
Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Олена Гріненко
«___» _____ 2025 р

ЗАВДАННЯ
на виконання кваліфікаційної роботи студента
Сутира Кирила Андрійовича

1. Тема кваліфікаційної роботи: «Адаптивний алгоритм та застосунок для персоніфікованого підбору дієтчних продуктів»
   затверджена наказом ректора від 17.11.2025 р. № 2450/ст

2. Термін виконання проекту: з 05.09.2025 р. до 31.12.2025 р.

3. Вихідні данні до проекту: мобільний iOS-застосунок для персональних харчових рекомендацій і контролю алергенів; наявність модулів автентифікації, сканування етикеток та інтеграції з відкритими каталогами; наявність локальної бази даних і шару взаємодії з нею для збереження профілю, продуктів та історії;

4. Зміст пояснювальної записки:
   1. Аналіз існуючих інформаційних технологій персоналізованого добору харчових продуктів.
   2. Теоретичні та методичні засади розробки персоналізованого добору харчових продуктів
   3. Програмна реалізація тестового програмного застосунку для аналізу харчових продуктів та рекомендацій

5. Перелік обов'язкових слайдів презентації:
   1. Існуюче програмне забезпечення
   2. Функціональні та нефункціональні вимоги
   3. Структура системи

6. Календарний план-графік

№ пор.	Завдання	Термін виконання	Відмітка про виконання
1.	Вибір та узгодження теми з керівником	05.09.25-07.09.25	+
2.	Аналіз актуальності теми та наявних готових рішень. Написання першого розділу роботи та представлення керівнику	07.09.25-25.09.25	+
3.	Аналіз та вибір інструментів для реалізації застосунку	26.09.25-03.10.25	+
4.	Написання другого розділу роботи та представлення керівнику	06.10.25-17.10.25	+
5.	Програмна реалізація алгоритму та застосунку	20.10.25-07.11.25	+
6.	Написання третього розділу роботи та узгодження з керівником	10.11.25-21.11.25	+
7.	Редагування роботи, проходження перевірки на антиплагіат	24.11.25-05.12.25	+
8.	Проходження нормоконтролю та передзахист	08.12.25-12.12.25	+
9.	Отримання відгуку керівника та рецензії. Розробка доповіді та презентації	15.12.25-26.12.25	
10.	Захист роботи	29.12.25	

7. Дата видачі завдання 05.09.2025

Керівник: доцент Наталія Шибицька

Завдання прийняв до виконання: Кирило СУТИР

РЕФЕРАТ

Пояснювальна записка до кваліфікаційного проєкту «Адаптивний алгоритм та застосунок для персоніфікованого підбору дієтичних продуктів»: 64 сторінки, 17 рисунків, 6 таблиці, 33 використані джерела і додаток.

Об'єкт дослідження – методи та засоби персоніфікованого підбору харчових продуктів на основі контекстних бандитів і автоматизованого зчитування етикеток.

Мета кваліфікаційної роботи – дослідити й розробити адаптивний алгоритм LinUCB та iOS-застосунок, що в режимі реального часу рекомендують продукти з урахуванням індивідуальних цілей Б/Ж/В, рівня активності та алергенів, використовуючи дані зі штрихкодів та OCR.

Методи дослідження – аналіз предметної області та наявних БД продуктів; огляд і формалізація методів контекстних бандитів; побудова ознак (protein, fat, carbs, fiber, sugars, salt, bias) з нормуванням; розробка робастного парсера нутрієнтів (OCR + OFF); on-device навчання LinUCB; експериментальна оцінка точності/затримок; UX-тестування інтерфейсів.

Результати роботи можуть бути використані під час розроблення програмних засобів для персоналізованого харчування та безпеки споживача: мобільні додатки зі сканером штрихкоду/OCR для миттєвого аналізу складу, виявлення алергенів та оцінки поживної цінності; рекомендаційні системи на основі контекстних бандитів (LinUCB), що адаптуються до цілей користувача

Розробка та дослідження проводилися під управлінням ОС MacOS Sequoia версії 15.6.1. Розробка програми проводилася у середовищі Xcode 16.2, на мові програмування Swift.

КОРИСТУВАЦЬКИЙ ІНТЕРФЕЙС, МОБІЛЬНИЙ ЗАСТОСУНОК, LINUCB, КОНТЕКСТНІ БАНДИТИ, OCR, ШТРИХКОД, НУТРІЄНТИ, ПЕРСОНАЛІЗАЦІЯ РЕКОМЕНДАЦІЙ

ABSTRACT

Explanatory note to the qualification project **“**Adaptive Algorithm and Application for Personalized Selection of Dietary Products”: 64 pages, 17 figures, 6 tables, 33 references, and an appendix.

Object of research – methods and tools for personalized selection of food products based on contextual bandits and automated label reading.

Aim of the qualification work – to research and develop an adaptive LinUCB algorithm and an iOS application that recommend products in real time, taking into account individual protein/fat/carbohydrate goals, activity level, and allergens, using data from barcodes and OCR.

Research methods – analysis of the subject area and existing product databases; review and formalization of contextual bandit methods; feature construction (protein, fat, carbs, fiber, sugars, salt, bias) with normalization; development of a robust nutrient parser (OCR + OFF); on-device LinUCB training; experimental evaluation of accuracy/latency; UX testing of interfaces.

Results of the work can be used in the development of software for personalized nutrition and consumer safety: mobile applications with a barcode/OCR scanner for instant composition analysis, allergen detection, and nutritional assessment; recommendation systems based on contextual bandits (LinUCB) that adapt to user goals

Development and experiments were carried out under **macOS Sequoia 15.6.1**. The application was developed in **Xcode 16.2** using the **Swift** programming language.

KEYWORDS: USER INTERFACE, MOBILE APPLICATION, LINUCB, CONTEXTUAL BANDITS, OCR, BARCODE, NUTRIENTS, PERSONALIZATION OF RECOMMENDATIONS.

Зміст

ВСТУП

Актуальність теми. Персоналізований вибір харчових продуктів стає ключовим елементом профілактики неінфекційних захворювань і контролю маси тіла [1]. Більшість популярних мобільних застосунків зосереджені або на скануванні штрихкодів і «універсальних» індексах на кшталт Nutri-Score/NOVA (загальна оцінка поживної якості та рівня переробки), або на підрахунку калорій/макроелементів без глибокої адаптації під конкретні цілі користувача [2]. Це створює розрив між науково обґрунтованими рекомендаціями та щоденними рішеннями людини в магазині: обрати саме той продукт, що найкраще підходить під її індивідуальні обмеження макронутрієнтів (Б/Ж/В), стан здоров'я й вподобання [3].

Об'єкт дослідження – процес прийняття рішень щодо вибору харчових продуктів у мобільному середовищі.

Передмет дослідження - методи та засоби адаптивної персоналізації вибору продуктів на основі аналізу Б/Ж/В, даних етикеток і зворотного зв'язку користувача на iOS.

Мета роботи – обґрунтувати та розробити адаптивний алгоритм і мобільний застосунок iOS для персоналізованого підбору харчових продуктів, який поєднує on-device аналіз етикеток (OCR/штрихкоди), науково вивірені нутріційні норми та онлайн-навчання з поведінкового фідбеку користувача.

Завдання дослідження:

1. Вивчити підходи конкурентних систем до аналізу Б/Ж/В і загальних оцінок поживної якості.
2. Сформувати вимоги до персоналізації (цілі й обмеження користувача; метрики відповідності).
3. Обґрунтувати вибір нутріційних нормативів та формул енергетичних потреб (BMR/TDEE) для розрахунку індивідуальних цілей.

4. Розробити модель скорингу продуктів під конкретні цілі користувача (відхилення Б/Ж/В від таргету, штрафи/бонуси).
5. Запропонувати та формально описати адаптивний алгоритм (контекстні бандити: LinUCB) для швидкого «підлаштування» під користувача.
6. Спроєктувати архітектуру застосунку (iOS, on-device OCR/база, приватність за замовчуванням).
7. Провести експерименти: точність розбору етикеток, відповідність Б/Ж/В цілям, показники навчання алгоритму.

Методи дослідження. Порівняльний аналіз програмних аналогів; теоретичний аналіз нутріційних рекомендацій формул енергетичних потреб; методи машинного навчання для онлайн-персоналізації (контекстні бандити: LinUCB); експериментальна перевірка на тестових наборах продуктів та зворотному зв’язку користувачів.

Технічні та програмні засоби. iOS (UIKit/Swift), Vision (OCR/штрихкоди), Core ML/локальна БД, офлайн-субсет Open Food Facts/Nutri-Score/NOVA як довідники.

Наукова новизна. Запропоновано адаптивний on-device підхід: поєднання персонального скорингу Б/Ж/В (відносно індивідуальних цілей) із контекстно-бандитним онлайн-навчанням за взаємодіями користувача, що дає змогу обходитись без серверної персоналізації, зберігаючи приватність і підвищуючи релевантність рекомендацій у реальному часі.

Практична значимість. Застосунок допоможе швидко обирати продукти, які відповідають особистим цілям Б/Ж/В, нормам жирів/солі/цукрів і/або алергенним обмеженням, із поясненнями «чому саме цей продукт» — корисно для користувачів і рітейлу (персоналізовані полиці).

Особистий внесок автора – аналіз аналогів; формалізація цілей і обмежень; проєктування скорингу; розробка адаптивного алгоритму; реалізація iOS-прототипу; план та проведення експериментів.

РОЗДІЛ 1.
АНАЛІЗ ІСНУЮЧИХ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ПЕРСОНАЛІЗОВАНОГО ДОБОРУ ХАРЧОВИХ ПРОДУКТІВ

1.1 Актуальність розробки програмного забезпечення персоналізованого добору харчових продуктів

Цифровізація щоденних рутин змінила спосіб, у який люди обирають продукти харчування. У супермаркеті або в онлайн-магазині користувач часто має лише кілька секунд на рішення, які спираються на етикетку, штрихкод та короткі індикатори якості [4]. Популярні фронт-оф-пак етикетки (типу Nutri-Score) і класифікації за ступенем переробки (NOVA) допомагають зорієнтуватися, але вони відображають «середню корисність», а не індивідуальні цілі й обмеження конкретної людини [5-6].

У той же час клас трекерів харчування надає механізми підрахунку калорій і Б/Ж/В на основі формул енергетичних потреб (BMR→TDEE) та дієтичних планів. Проте ці інструменти працюють переважно у форматі післяфактум-обліку та майже не підтримують моментальний персональний добір «прямо з полиці». Це створює розрив між теоретично правильними нормами і практикою щоденного вибору.

Актуальність полягає у розробці рішення, яке поєднає:

1. Локальне розпізнавання етикетки (OCR/штрихкоди) з підтримкою україномовних та англомовних позначень.

2. Персональний скоринг відповідно до індивідуальних добових цілей Б/Ж/В і лімітів (сіль, насичені жири, додані цукри).

3. Адаптивне онлайн-навчання з поведінкового фідбеку користувача без передачі персональних даних на сервер.

На перетині цих трьох складових лежить підвищення якості харчування, зручності та довіри до цифрових інструментів.

Мета роботи — дослідити й реалізувати мобільний застосунок, який локально розпізнає етикетки, агрегує дані OFF та формує персональні рекомендації в реальному часі [7].

Додатковий мотив — приватність і доступність. Перевага on-device підходу полягає в тому, що фото та розпізнаний текст не покидають пристрій, а застосунок залишається працездатним офлайн. Це особливо важливо для користувачів, які не готові ділитися даними про харчові вподобання й здоров'я з віддаленими сервісами.

1.2 Огляд ринку та аналогів

Аналоги можна умовно розділити на три групи:

1. Сканери етикеток із «загальним» індексом якості.
2. Трекери БЖВ з калорійними планами.
3. Системи візуального аналізу готових страв.

Нижче подано короткий аналіз підходів і обмежень.

Yuka: використовує інтегральний бал (0–100), що складається з Nutri-Score, оцінки харчових добавок та ознаки органічності. Методика добре підходить для швидкої інтерпретації, однак персональні добові цілі не враховуються; відтак рейтинг є універсальним і не може підказати, чи відповідає продукт саме вашому залишку Б/Ж/В на день. Нижче, на рис. 1.1 наведено скріншот інтерфейсу застосунку Yuka:

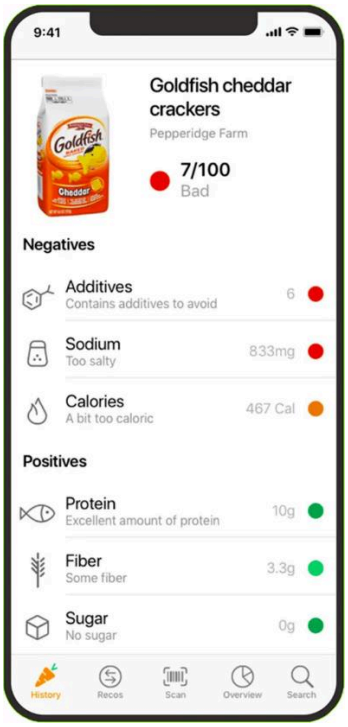

Рис. 1.1. Зовнішній вигляд мобільного застосунку Yuka

Open Food Facts: відкрита база, що надає харчову цінність на 100 г/мл, Nutri-Score, NOVA та довідкові атрибути за штрихкодом. OFF корисна як джерело даних, але сама по собі не виконує персоналізованого добору й не пропонує алгоритмів адаптації. Нижче, на рис. 1.2 наведено скріншот інтерфейсу веб-сайту Open Food Facts:

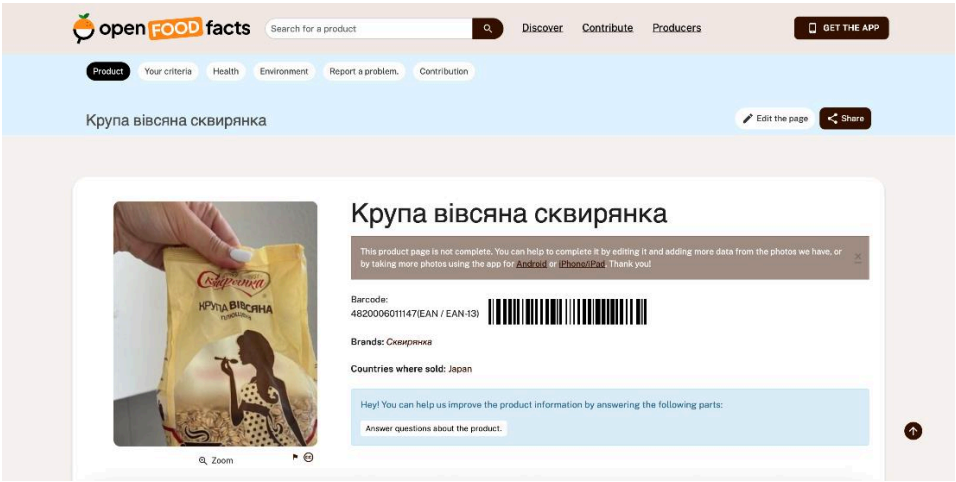

Рис. 1.2. Зовнішній вигляд веб-сайту Open Food Facts

MyFitnessPal, Lifesum: дозволяють встановити добові цілі за BMR/TDEE і відстежувати БЖВ. Сильна сторона — контроль денного раціону; слабка — статичність планів та фокус на обліку, а не на миттєвому персональному порівнянні конкретних товарів у магазині. Нижче, на рис. 1.3 наведено скріншот інтерфейсу застосунку MyFitnessPal:

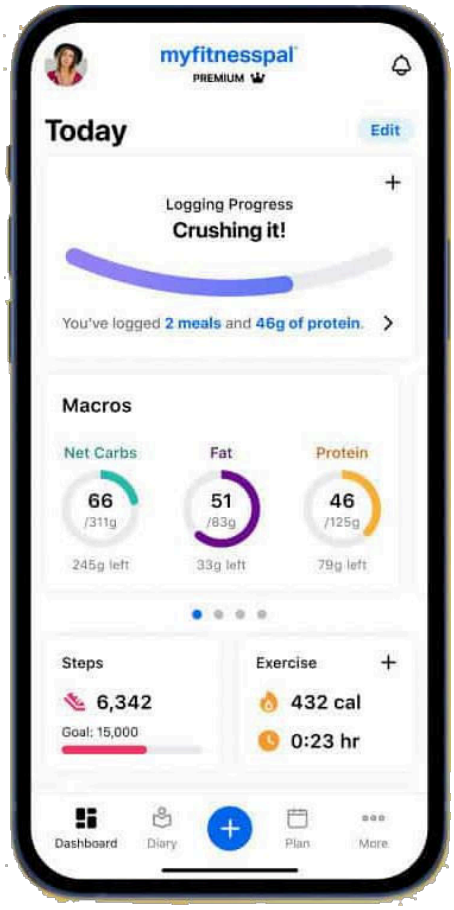

Рис. 1.3. Зовнішній вигляд мобільного застосунку MyFitnessPal

Cronometer: робить акцент на точності мікронутрієнтів і авторитетних джерелах (USDA тощо), проте підбір альтернатив під моментну потребу користувача не є центральною функцією. Foodvisor: аналізує зображення готових страв за допомогою CV/ML, що корисно для щоденника харчування, але не допомагає з упакованими товарами за етикеткою. Нижче, на рис. 1.4 наведено скріншот інтерфейсу застосунку Cronometer:

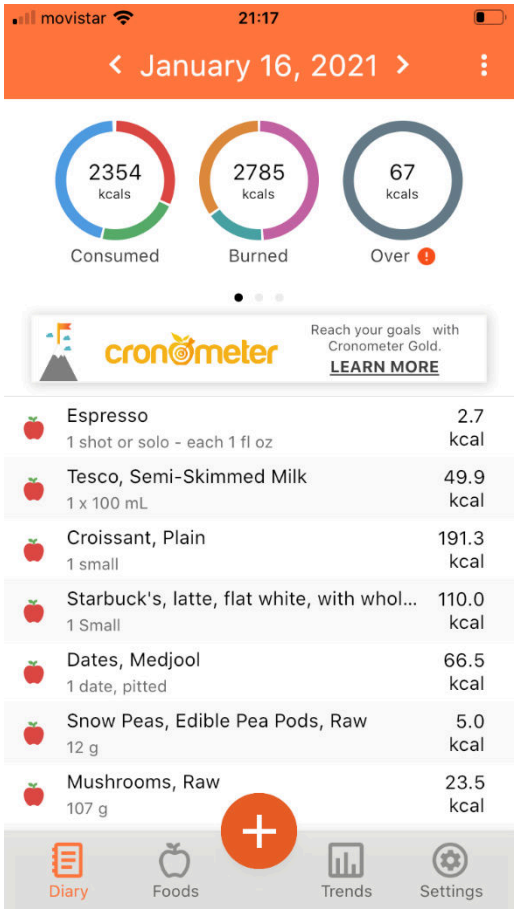

Рис. 1.4. Зовнішній вигляд мобільного застосунку Cronometer

Висновок підрозділу: наявні продукти або дають узагальнену «якість», або забезпечують підрахунок. Персоналізоване ранжування упакованих продуктів під поточні індивідуальні цілі в момент вибору практично відсутнє. Саме цю нішу має заповнити запропоноване рішення.

Нижче наведено порівняльну табл. 1.1 наявних рішень.

Таблиця 1.1 – Порівняння наявних рішень за підходом до аналізу та персоналізацією

Сервіс / продукт	**Підхід до аналізу БЖВ/індикаторів**	**Джерела даних**	**Персоналізація**	**Переваги**	**Обмеження**
Yuka	Інтегральний бал (0–100): Nutri-Score + добавки + органічність	Власна БД + OFF	Немає індивідуальних БЖВ-цілей	Швидке рішення «світлофором»	Універсальний індекс, без урахування особистих цілей
Open Food Facts	Поживність на 100 г/мл, Nutri-Score, NOVA, добавки	Відкрита БД OFF (ODbL)	Не персоналізує	Відкритість, широке покриття штрихкодів	Якість записів варіюється; відсутність персоналізації
MyFitnessPal	Підрахунок калорій і БЖВ відносно BMR/TDEE	Користувацькі та довідкові записи	Статичні цілі/плани	Гнучкий контроль денних макросів	Пост-фактум облік, слабка підтримка моментального вибору
Lifesum	Макро-плани, розподіл БЖВ за TDEE	Власні довідники	Статичні дієтичні плани	Зручні пресети дієт	Без онлайн-адаптації під поведінку
Cronometer	Точний облік макро-/мікронутрієнтів	USDA/NCCDB тощо	Переважно ручне налаштування	Висока точність мікронутрієнтів	Не орієнтовано на вибір «з полиці»
Foodvisor	CV-аналіз фото страв (калорійність, макроси)	Власні ML-моделі	Обмежена	Зручно для приготованих страв	Не працює з упакованими товарами за етикеткою

1.3 Визначення проблематики та формалізація задачі

Проблема полягає в тому, що «універсальні» індекси якості та традиційний облік БЖВ не відповідають на персональне питання: який із доступних на полиці продуктів найкраще наближає мене до моїх добових цілей тут і зараз. Формально завдання пропонується розглядати як ранжування множини кандидатів за персональною цільовою функцією [8-9].

Вхідні дані: профіль користувача (стать, вік, зріст, вага, активність, цілі — дефіцит/баланс/набір); поточні «залишки» денних Б/Ж/В та ліміти (сіль, насичені жири, додані цукри); ознаки продукту з етикетки (поживність на 100 г/мл і на порцію, білок, клітковина, сіль, цукри, насичені жири, енергетична щільність, можливі алергени); додаткові індикатори (Nutri-Score/NOVA); поведінковий фідбек (вибір/відхилення, позначка «подобається»).

Вихідні дані: упорядкований список альтернатив із поясненням («+18 г білка до цілі; −6 г від допустимого жиру; сіль у межах; Nutri-Score B»). Обмеження: час відповіді до сотень мілісекунд; робота без мережі; конфіденційність; підтримка української/англійської етикетки; прозорість рішення.

Дослідницькі питання:

1. Як стабільно витягувати Б/Ж/В і ключові показники з «шумних» етикеток.
2. Як сконструювати цільову функцію, що відображає індивідуальні пріоритети.
3. Який онлайн-алгоритм адаптації краще збігається з поведінкою користувача.
4. Як валідувати якість рекомендацій офлайн-методами.

1.4 Методи та підходи до оцінювання й персоналізації

Парсинг етикетки. Застосовується on-device OCR для виділення блоку «Склад/Ingredients» і таблиці поживності. Далі — нормалізація одиниць (г/мл/порція), узгодження значень «на 100 г/мл» із порцією, корекція заокруглень. Для інгредієнтів використовується словниковий шар (синоніми/алергени) з

можливістю доповнення користувачем. Також надається альтернативний спосіб отрмання інформації про продукт за рахунок сканування баркоду. За допомогою відкритої бази Open Food Facts можна отримати дані про товар, який відсканував користувач. Після чого розпарсити інформацію для подальшого аналізу. В обох вибадках користувач зможе вручну замінити помилково опрацьовані дані продукту. Також для зручності кожний товар перед збереженням в локальну БД має назву та фото, яке може бути отримано або з результату аналізу, або напряму прописані користувачем.

Персональний скоринг. Оцінювання ведеться для 100 г продукту: білок *p*, жири *f*, вуглеводи *c*, клітковина *fi*, цукри *su*, сіль *sa.* Нехай добові цілі користувача — T_P, T_F, T_C (білок/жири/вуглеводи), а ліміти — L_sat, L_Na, L_sugar. Для порції продукту обчислюємо внесок у досягнення цілей і відхилення від «залишків». Цільова функція поєднує: бонус за наближення до таргетів білка/клітковини; штраф за перевищення насичених жирів/солі/доданих цукрів; регуляризацію за калорійну щільність; (опційно) ваги для Nutri-Score та NOVA як додаткових ознак. Ваги ініціалізуються нормативами (EFSA/WHO) та налаштовуються користувачем.

Адаптивність. Для підлаштування ваг під конкретні вподобання застосовуються контекстні бандити (LinUCB). Алгоритм одержує контекст (профіль + залишки + ознаки продукту) і вибирає дію (продукт) з урахуванням невизначеності; фідбек (вибір/відмова) оновлює параметри на пристрої. Це забезпечує баланс дослідження/експлуатації та швидку персоналізацію без сервера.

Словниковий шар алергенів і нормалізація. Перед пошуком виконуємо нормалізацію (зниження регістру, усунення діакритики, згортання пробілів, компенсація OCR-помилок 0↔o, 1↔l, 5↔s). Словник охоплює укр/рос/eng варіанти та синоніми; пошук — регулярними виразами по токенах. Збіги агрегуються по категоріях для UI та штрафів.

Технічна реалізація. iOS (UIKit/Swift): Vision для OCR і штрихкодів; локальна БД (SwiftData/Core Data) для кешу продуктів, словників і профілів; скоринг і адаптація — у Core ML/Swift без мережевих викликів. Для довідника можливий офлайн-зріз відкритої бази (наприклад, Open Food Facts) з атрибуцією.

Валідація. Для OCR — CER/WER; для парсингу — частка коректно витягнутих Б/Ж/В/сіль/цукри на 100 г; для персоналізації — офлайн-replay із кумулятивним регретом, частка прийнятих рекомендацій, середнє відхилення від цілей БЖВ за день.

1.5 Обґрунтування вибору технологій та архітектурних підходів

У центрі рішення — мобільний застосунок під iOS, написаний на Swift. Вибір обумовлений поєднанням продуктивності, зрілої екосистеми та прямого доступу до фреймворків комп'ютерного зору й безпеки на пристрої [10]. Використання UIKit забезпечує контрольований мінімалістичний інтерфейс із швидкою реакцією, що важливо для камерних сценаріїв та оверлеїв розпізнавання. Фреймворк Vision надає високоточний OCR і детекцію штрихкодів без відправки фото на сервер, тож обробка відбувається локально, з низькою затримкою і кращим контролем приватності.

Джерелом довідкових даних про продукти обрано OpenFoodFacts (OFF) як відкритий, глобальний реєстр з публічним API [7]. Для українського ринку покриття нерівномірне, однак це компенсується комбінованою стратегією: спочатку запит за EAN до OFF, далі локальний кеш, і якщо даних бракує — OCR етикетки з евристиками парсингу нутрієнтів. Такий «каскад» дає стійкість до пропусків у базі й покращує повноту заповнення, а також зменшує час очікування користувача за наявності даних у кеші.

Парсер нутрієнтів проектовано OCR-дружнім: ми нормалізуємо текст (усуваємо діакритики, уніфікуємо пробіли, виправляємо схожі символи 0↔O, 1↔l, 5↔S), шукаємо значення по кількох варіантах полів OFF (наприклад, proteins_100g,

protein_100g, proteins), відсікаємо аномалії саніті-чеками і пріоритезуємо одиниці «на 100 г». Визначення алергенів здійснюється словниковим підходом із лексиконом українською/російською/англійською, розширеним синонімами та морфологічними варіантами. Збіги, що перетинаються з персональним профілем, виділяються як критично важливі — це знижує шум і підсилює практичну цінність підказок.

За персоналізацію відповідає рекомендатор на базі контекстної бандитної моделі LinUCB. Модель використовує компактний, але інформативний вектор ознак: нормовані нутрієнти за 100 г (білки, жири, вуглеводи, клітковина, цукри, сіль), похідні показники (умовний “псевдоскор”), а також параметри профілю (вага, зріст, вік, рівень активності) і ціль (дефіцит/баланс/надлишок). У векторі присутній бінарний індикатор перетину алергенів з профілем, що знижує привабливість небезпечних товарів. LinUCB балансує між експлуатацією та експлорацією завдяки доданку невизначеності: товари з малою історією отримують «бонус за дослідження», а відгуки користувача (implicit/explicit) поступово переналаштовують ваги.

Архітектурно застосунок «офлайн-перший»: усе чутливе — на пристрої; у мережу йде лише EAN і запит до OFF по HTTPS. Збережені дані (профіль, улюблені товари, фото) лишаються в iOS sandbox; логування обмежено технічним debug-режимом. Така конструкція одночасно підвищує приватність, швидкодію і живучість при нестабільному інтернеті. З погляду UX ми залишили можливість ручного редагування нутрієнтів і алергенів через нижнє напівекранне вікно: це швидкий спосіб підчистити OCR або доповнити бідні офіційні дані, не виходячи з потоку користувача.

Сукупно обраний стек — Swift + Vision + OFF + локальна модель LinUCB — мінімізує зовнішні залежності, дає контроль над якістю розпізнавання, дозволяє коректно працювати з алергенами та створює прозору основу для подальшого

масштабування (наприклад, підключення FDA/EFSA-джерел або серверної нормалізації довідників у майбутньому).

1.6 Ризики проєкту та способи їх мінімізації

У проєкті поєднано локальну обробку зображень (Swift + Vision/OCR), відкриті довідники (OpenFoodFacts) та персональні рекомендації на базі LinUCB. Такий стек дає хорошу швидкодію і приватність, але має характерні ризики — від якості даних до UX та регуляторних нюансів. Нижче — ключові групи ризиків і наші підходи до їх стримування; детальна пріоритизація наводиться у матриці ризиків.

Якість та повнота зовнішніх даних. Покриття OFF для українського ринку нерівномірне: частина EAN повертає порожні або фрагментарні записи. Це загрожує «німими» картками продуктів і зниженням довіри. Ми пом'якшуємо це каскадом джерел: спершу швидкий запит до OFF, далі — локальний кеш (щоб не питати повторно), і як резерв — OCR етикетки з парсером нутрієнтів, заточеним під помилки розпізнавання. Додатково передбачено ручне редагування нутрієнтів/алергенів у мінімалістичному нижньому вікні — користувач може довнести дані за кілька дотиків.

Помилки OCR та варіативність макетів етикеток. Різні шрифти, верстка, засвіт кадру і друкарські артефакти породжують шум. Щоби тримати стабільність, ми обмежуємо область інтересу, нормалізуємо текст (діакритики, пробіли, "0↔O", "1↔l", "5↔S"), шукаємо значення за кількома ключами (наприклад, proteins_100g / protein_100g / proteins) і застосовуємо саніті-чеки (недопускаємо від'ємні або нереалістично великі значення, фіксуємо «на 100 г» як дефолт). Якщо витягнуто менше трьох полів — вважаємо результат ненадійним і продовжуємо скан.

Невизначеність у визначенні алергенів. Назви інгредієнтів подаються різними мовами, з варіаціями та синонімами; помилки OCR посилюють ризик хибних спрацювань або пропусків. Ми використовуємо багатомовний лексикон із

морфологічними варіантами й очищенням тексту; індикатори, що збігаються з персональним профілем користувача, виділяємо як пріоритетні (⚠), а нейтральні — знижуємо у важливості. У разі сумнівів користувач має швидкий селектор алергенів із каталогу — редагування зберігається у картці продукту.

UX-ризики у камерних сценаріях. Проблеми доступу до камери, занадто темні сцени, нестабільність фокусу та відблиски можуть робити скан довгим і фруструючим. Ми застосовуємо прозорі підказки (режим «Штрихкод» із затемненням поза рамкою та «OCR» без обмежень), тротлінг обробки кадрів, а також чіткий стан «аналізуємо/знайдено/повторити». Навіть якщо скан не вдався, користувач отримує можливість зберегти мінімальну картку й доповнити її вручну.

Холодний старт рекомендацій. На початку модель LinUCB має мало відгуків і може ранжувати «дивно», підвищуючи елементи з високою невизначеністю. Ми компенсуємо це фіксованими ознаками з профілю (ціль: дефіцит/баланс/надлишок, активність, антропометрія), регуляризованим “псевдоскором” із нутрієнтів і стартовими примерами для кожної цілі. Явний фідбек або фактичні взаємодії поступово зміщують ваги; кнопка «Оновити рекомендації» дозволяє швидко пересортувати список після правок профілю або карток.

Модельні упередження та небажана експлуатація. Бандит за означенням віддає перевагу комбінації корисності й невизначеності — іноді це «випадкові фаворити». Ми обмежуємо простір ознак, нормуємо їх до [0,1], включаємо штраф за перетин алергенів із профілем і контролюємо параметр α (баланс експлоатації/експлорації). Для звітності зберігаємо прозорі індикатори (пояснювальний текст і бейдж оцінки).

Приватність і безпека. Фото не залишають пристрій; у мережу йде лише EAN і запит HTTPS до OFF. Дані профілю та збережені продукти — у sandbox iOS; користувач може стерти їх одним натисканням. Юридично продукт не надає

медичних рекомендацій — лише довідкову інформацію; відповідне попередження на екранах унеможливлює помилки трактування [11].

Технічна відмовостійкість. Відсутній інтернет, таймаути OFF або зміни у форматі API можуть призводити до «тихих» збоїв. Ми використовуємо таймаути з fallback-запитами, кешування відповідей, обмежене діагностичне логування (видно, де саме зірвався ланцюжок: штрихкод/мережа/парсер). Це суттєво прискорює відлов точкових проблем у полі.

Дизайн-узгодженість і доступність. Тримання єдиної стилістики (зеленувата тема, фіолетовий індикатор високої оцінки, помаранчеві CTA) важливе для довіри та читабельності в умовах різних освітлень. Ми використовуємо контрастні шрифти, мінімум візуального шуму і підказки у критичних точках (редагування нутрієнтів/алергенів). Це знижує ризик помилок взаємодії.

Етичні аспекти та неправильна інтерпретація. Оцінка — не абсолютна «корисність», а індикатор відповідності профілю та нутрієнтів. Ми уникаємо категоричних висновків, не «клеймимо» продукти як «погані/хороші», а підсвічуємо фактори (білок, цукри, сіль, алергени) й даємо короткий коментар. Це допомагає приймати поінформовані рішення без стигматизації.

У підсумку, ризики у нашому випадку зосереджені навколо якості джерел, помилок розпізнавання та стартової невизначеності моделі. Каскадний збір даних, OCR-дружній парсер, ручні правки «в один дотик», суворі саніті-чеки, локальна обробка й прозора персоналізація істотно зменшують їхній вплив. Матриця ризиків у додатку деталізує кожен пункт із оцінкою й пріоритетами пом’якшення. Нижче наведена матриця ризиків (табл 1.2), яка відображає ризики, степінь їх впливу та методи вирішення:

Таблиця 1.2

Порівняннння ризиків та їх впливу

№	Ризик	Ймовірність	Вплив	Виявлення	Пом’якшення / Контрзаходи
1.	Відсутні дані в OFF	Середня	Середній	Моніторинг кодів у логах	Каскад OFF→кеш→OCR
2.	Неповні нутрієнти	Середня	Середній	Перевірки completeness	Синоніми ключів; конвертації до 100 г; валідаційні межі
3.	OCR-помилки	Висока	Низький	Звірка сум/форматів	Акумуляція кадрів; нормалізація; регулярки; ручне редагування
4.	Неправильне визначення алергенів	Середня	Високий	Перехресна перевірка з профілем	Розширений лексикон (синоніми/мови), ваги збігів, ручний вибір
5.	«Фальшиві» штрихкоди	Низька	Низький	Поріг конфіденсу, гістерезис	Вузький ROI, обмеження символогій, підсвічування рамки
6.	Витік персональних даних	Низка	Високий	Code review, стат. аналіз	Локальна обробка, без відправки зображень; HTTPS; мінімізація даних
7.	Зміна схеми OFF/API	Середня	Середній	Сентрі/логи помилок	Фолбек на інші ключі, таймаути/ретраї, версіонування клієнта

Висновки

У розділі обґрунтовано суспільну та технологічну актуальність персоналізованого добору харчових продуктів у мобільному середовищі. Показано, що існуючі рішення або спираються на універсальні індикатори якості (Nutri-Score/NOVA), або орієнтовані на післяфактум-облік БЖВ та статичні плани. Таким чином, вони не дають відповіді на ключове практичне питання користувача в магазині: «який із доступних продуктів найкраще підходить саме мені зараз?».

Запропоновано концепцію on-device системи, яка поєднує локальне розпізнавання етикетки, персональний скоринг відносно індивідуальних добових цілей і лімітів, а також адаптивне онлайн-навчання з поведінкового фідбеку. Очікувані переваги:

1. Релевантність рекомендацій у момент вибору.

2. Приватність і робота офлайн.

3. Прозорі пояснення («чому саме цей продукт»).

4. Можливість гнучкого налаштування під різні дієтичні сценарії (дефіцит / баланс/набір, контроль солі/цукрів/насичених жирів).

Водночас окреслено ризики та виклики: якість OCR на «шумних» етикетках; різноманіття форматів поживних таблиць; узгодження значень «на 100 г/мл» із порцією; змішані мовні позначення інгредієнтів; калібрування ваг у скорингу. Для їх пом’якшення передбачено словниковий шар, нормалізацію одиниць, офлайн-валідацію та адаптивні алгоритми з контрольованою експлорацією.

Сформована теоретична та методична база стане підґрунтям для наступних розділів, де буде представлено архітектуру застосунку, деталізацію алгоритмів парсингу та персонального скорингу, а також дизайн експериментів і результати оцінювання якості та корисності системи.

РОЗДІЛ 2.
ТЕОРЕТИЧНІ ТА МЕТОДИЧНІ ЗАСАДИ РОЗРОБКИ ПЕРСОНАЛІЗОВАНОГО ДОБОРУ ХАРЧОВИХ ПРОДУКТІВ

2.1 Нормативна база, припущення та обмеження

Розробка персоналізатора ґрунтується на загальноприйнятих у дієтології підходах до оцінювання енергетичних потреб людини та діапазонів споживання макронутрієнтів [12]. На першому кроці оцінюється базальна швидкість метаболізму (BMR), після чого використовується коефіцієнт фізичної активності для отримання загальних добових енерговитрат (TDEE) [13].

Ці значення є основою для індивідуальних добових цілей за білками, жирами та вуглеводами (Б/Ж/В), а також обмежень щодо насичених жирів, доданих цукрів і натрію. Для користувача в застосунку це проявляється у заповненні короткого профілю та виборі цілі (дефіцит/баланс/набір).

Обрана методика не надає медичних рекомендацій і не замінює консультацій лікаря. Вона формалізує алгоритм добору, який узгоджується із загальновідомими нормами та дозволяє користувачеві самостійно налаштовувати цілі. Серед припущень — адекватність самооцінки рівня активності, допустимість використання узагальнених коефіцієнтів для перерахунку енергії, а також відповідність маркування на етикетці чинним нормам. Обмеження: вагітність, певні захворювання, дитячий вік тощо — потребують окремих рекомендацій і виходять за межі автоматизованого добору.

Практичний аспект — перетворення відсоткових цілей на абсолютні грами. Якщо користувач має TDEE = 2200 ккал і обрав, наприклад, 25% білка, то добова ціль білка становить близько 138 г (2200×0.25/4). Аналогічно розраховуються жири (9 ккал/г) і вуглеводи (4 ккал/г). Застосунок виконує ці перетворення автоматично, пропонуючи заокруглення до зручних значень (табл. 2.1).

Таблиця 2.1

Ілюстративні діапазони макронутрієнтів для різних сценаріїв

Сценарій	**Білок (P)**	**Жири (F)**	**Вуглеводи (C)**	**Примітки**
Підтримка ваги	15–25% калорій	20–35% калорій	45–55% калорій	Збалансоване харчування
Помірний дефіцит	20–30% калорій	20–30% калорій	40–50% калорій	Підвищений білок для ситості
Набір м’язової маси	25–35% калорій	20–30% калорій	40–50% калорій	Високий білок

2.2 Дані етикетки та нормалізація позначень

Етикетка зазвичай містить список інгредієнтів, таблицю харчової цінності (на 100 г/мл та/або на порцію), позначення алергенів, а також допоміжні атрибути (типи жирів, наявність доданих цукрів, тощо). Паралельно до зчитування з етикетки застосунок використовує відкритий довідник **Open Food Facts (OFF)** за штрихкодом, якщо користувач відсканував EAN/UPC. Ми поєднуємо обидва джерела:

OFF → базові поля: nutriments.{proteins_100g, fat_100g, carbohydrates_100g, sugars_100g, fiber_100g, salt_100g}; локалізовані назви (product_name_uk/ru/en) та інгредієнти (ingredients_text_uk/ru/en), зображення.

OCR → уточнення: коли OFF неповний або відсутній, локальний OCR/парсер добирає макро/інгредієнти з етикетки.

Злиття: якщо значення дублюються, пріоритет має *_100g, далі — «на порцію», далі — евристика перерахунку. Всі величини приводяться до формату на 100 г/мл, а також (за потреби) — до порції.

Для локальних маркувань підтримуємо синоніми та морфологію (укр/рос/англ), що підвищує стійкість до варіацій написання: *молоко* ↔ *молочні; глютен* ↔ *пшениця; арахіс* ↔ *арахісова олія* тощо. У разі нестачі полів ми явно зберігаємо «неповні записи» і відмічаємо знижений рівень упевненості.

Таблиця 2.2

Відображення полів етикетки у нормалізовані ознаки

Поле етикетки	**Приклади позначень**	**Нормалізація/одиниця**	**Примітки**
Енергійність	kcal, kJ	ккал на 100 г та на порцію	kJ→ккал (1 ккал ≈ 4.184 кДж)
Білки	protein, білок	г/100 г; г/порцію	округлення до 0.1 г
Жири (всього)	fat	г/100 г; г/порцію	окремо насичені жири
Насичені жири	saturated fat	г/100 г; г/порцію	обмежуються в скорингу
Вуглеводи	carbohydrate	г/100 г; г/порцію	окремо цукри/додані цукри
Цукри	sugars	г/100 г; г/порцію	можлива різниця «всього» vs «доданих»
Сіль/Натрій	salt/sodium	г солі або мг натрію	Na→сіль: г·2.5; мг Na→г солі: мг·2.5/1000
Клітковина	fiber	г/100 г; г/порцію	бонус у скорингу

Для несуперечності порівнянь усі значення приводяться до представлень «на 100 г/мл» і «на порцію». За відсутності поля система може вивести оцінку із наявних даних (напр., енергетичність з макросів за факторами Атвотера).

2.3 On-device OCR та виділення структур етикетки

Пайплайн OCR працює на пристрої, що гарантує приватність і низьку затримку. Ключові етапи:

1. Захоплення кадру з камери.

2. Попередня обробка (вибір експозиції, подавлення шуму, геометрична корекція за необхідності).

3. Багатомовне розпізнавання тексту.

4. Евристичне виділення блоку «Склад/Ingredients» і таблиці харчової цінності

5. Повернення структурованого тексту з координатами елементів та оцінкою впевненості.

Для виділення інгредієнтів застосовується набір тригерів-заголовків («Склад», «Ingredients», «Состав») і стоп-слів («Поживна/Пищевая цінність», «Nutrition facts»). Таблиця харчової цінності визначається за маркерами «на 100 г/мл»/«на порцію», наявністю числових стовпців і одиниць вимірювання. Якщо структура не знайдена або впевненість низька, система переходить у режим аналізу всього тексту з відповідним маркуванням результатів.

Додаткові прийоми включають тротлінг обчислень (не кожен кадр), відкидання занадто косих або розмитих зображень та підказки для користувача («наблизьтесь», «покращіть освітлення»). Це зменшує кількість помилок і покращує стабільність даних.

2.4 Парсинг і нормалізація тексту етикетки

Після OCR виконується нормалізація тексту: зниження регістру, усунення діакритики, згортання пробілів і уніфікація роздільників («,», «;», «•»). Числові значення вилучаються регулярними виразами, що враховують локальні десяткові віддільники (кома/крапка), варіанти одиниць (г/мг/мл/ккал) і позначення діапазонів («<», «≤», «до»). Для інгредієнтів використовується словниковий шар із синонімами та морфологічними варіаціями, який можна поповнювати з UI.

На цьому ж кроці виконується узгодження неоднозначностей: перетворення натрію в еквівалент солі, розділення «вуглеводів» на «всього» та «цукри» (коли доступно), узгодження значень на 100 г/мл з даними на порцію. Якщо деякі поля відсутні, алгоритм пом'якшено реагує, зменшуючи їній внесок у підсумкову оцінку та відображаючи нижчу впевненість.

Для підвищення якості застосовуються перевірки цілісності: сума макросів співвідноситься з енергетичністю (за відомими коефіцієнтами), а невідповідності відображаються як попередження. Це дозволяє уникнути помилкових рішень через округлення на етикетках або помилки OCR.

2.5 Обчислення похідних показників і приведення до порції

Значення перераховуються як у стандарт «на 100 г/мл», так і в «на порцію», яку обирає користувач (або задає виробник). Енергетичність може бути розрахована з макросів (фактори Атвотера: білки та вуглеводи 4 ккал/г, жири 9 ккал/г), натрій перетворюється в еквівалент солі (×2.5). Для рідких продуктів у разі потреби використовують густину для перерахунку мл→г.

Якщо порція не вказана, застосунок пропонує дефолтні величини (наприклад, 30 г для пластівців, 250 мл для напоїв), які користувач може відкоригувати. Усі обчислення супроводжуються позначенням точності та заокругленням до однакової точності відображення (наприклад, 0.1 г).

2.6 Модель персонального скорингу продуктів

Модель скорингу поєднує бонуси за наближення до індивідуальних цілей Б/Ж/В та штрафи за наближення до лімітів солі, насичених жирів і доданих цукрів. Для кандидата і формується вектор ознак x_i на порцію: внесок у білок/жири/вуглеводи, клітковина, насичені жири, цукри, сіль (екв.), енергетична щільність, а також (опціонально) індикатори Nutri-Score та NOVA.

Цільова функція — зважена сума, нормована в діапазон [0;100] для інтерпретованості: високі значення відповідають кращій узгодженості з поточними цілями користувача. Ваги ініціалізуються нормативно) і можуть бути змінені користувачем. Пояснюваність забезпечується через розклад оцінки на внески ознак («+18 г білка → +15 балів; 0.9 г насичених жирів → −6 балів» (табл. 2.3)).

Для стабільності запроваджується регуляризація (щоб штрафи не «перебивали» бонуси повністю), а також механізм тай-брейків (коли бали однакові, перевага надається продуктам із вищою клітковиною або нижчою енергетичною щільністю).

Таблиця 2.3

Приклад ознак і ваг у базовій ініціалізації

Ознака (порція)	**Сенс**	**Початкова вага**
Δ білка до цілі	покриває дефіцит	+
Δ жирів до ліміту	наближення до ліміту — штраф	−
Δ вуглеводів до цілі	балансує енергію	±
Клітковина	ситість/ШКТ	+
Насичені жири	обмежуємо	−
Додані цукри	обмежуємо	−
Сіль (екв.)	обмежуємо	−
Енергетична щільність	регуляризація	−

Nutri-Score/NOVA	додаткова ознака	±

2.7 Адаптивний алгоритм персоналізації (контекстні бандити)

Теоретичні відомості про контекстні бандити. Контекстні бандити — це спрощена постановка підкріплювального навчання, де на кожному кроці система спостерігає контекст $\boldsymbol{x} \in \mathbb{R}^{\boldsymbol{d}}$(у нас — ознаки продукту та стан дня: «залишки Б/Ж/В» до добової цілі, час доби тощо), має множину кандидатів (товарів/дій), обирає одного кандидата, отримує винагороду $\boldsymbol{r}$(прийнято/відхилено або безперервна оцінка) і оновлює внутрішній стан [15]. На відміну від повного RL, тут немає довгих епізодів і складної динаміки станів — лише миттєвий контекст і миттєва винагорода. Така постановка ідеально підходить до персональних рекомендацій «тут-і-зараз» для харчових продуктів: вона забезпечує швидке on-device навчання та прозору пояснюваність.

LinUCB: лінійна модель з оптимістичною межею. Ми припускаємо, що очікувана винагорода від кандидата з ознаками $\boldsymbol{x}$ лінійна:

$$\mathbb{E}[\boldsymbol{r} \mid \boldsymbol{x}] = \boldsymbol{\theta}^{\top}\boldsymbol{x},$$

де $\boldsymbol{\theta}$— вектор «ваг вподобань» користувача. Для кожного кандидата обчислюємо оптимістичний скор:

$$\boldsymbol{p}(\boldsymbol{x}) = \widehat{\boldsymbol{\theta}}^{\top}\boldsymbol{x} \; + \; \boldsymbol{\alpha}\sqrt{\boldsymbol{x}^{\top}\boldsymbol{A}^{-\mathbf{1}}\boldsymbol{x}},$$

де $\widehat{\boldsymbol{\theta}} = \boldsymbol{A}^{-\mathbf{1}}\boldsymbol{b}$— оцінка ваг (рідж-регресія у реальному часі), $\boldsymbol{A} = \boldsymbol{\lambda I} + \sum \boldsymbol{x}\boldsymbol{x}^{\top}$— «дизайн-матриця» з регуляризацією $\boldsymbol{\lambda} > \mathbf{0}$, $\boldsymbol{b} = \sum \boldsymbol{r}\boldsymbol{x}$, а $\boldsymbol{\alpha} > \mathbf{0}$ балансує експлуатацію і дослідження. Перший доданок — прогноз користі, другий — бонус за невизначеність: що менше ми знаємо про «подібні» товари, то більший шанс дати їм спробу.

Мета та постановка. На кожному кроці система бачить контекст дня (залишки добрати Б/Ж/В, активні ліміти, час доби) і набір кандидатів. Завдання — миттєво запропонувати той варіант, що найкраще наближає користувача до індивідуальних цілей саме зараз, і паралельно навчатися на його виборах повністю локально.

Джерела даних: OFF та OCR — злитий пайплайн. Якщо доступний штрих-код, ми звертаємося до Open Food Facts (OFF) і витягуємо нормалізовані поля на 100 г: proteins_100g, fat_100g, carbohydrates_100g, sugars_100g, fiber_100g, salt_100g, а також текст інгредієнтів (ingredients_text_{uk,ru,en}) та зображення. Якщо OFF відсутній або незаповнений — вмикаємо OCR етикетки, витягуємо таблицю харчової цінності та «склад», нормалізуємо одиниці, чистимо артефакти. Коли є обидва джерела, зливаємо їх за пріоритетом OFF→OCR (OFF як «джерело істини», OCR — доповнення/резерв).

На рисунку 2.1 зображено перехід від джерел даних до алгоритму LinUCB.

Рис. 2.1. Перехід від джерел даних до алгоритму LinUCB

Ознаки продукту та контексту. Для кожного кандидата формуємо вектор ознак $\boldsymbol{x}$: бонуси за protein, fiber; штрафи за sugars та salt; за потреби — енергетична щільність; контекстні компоненти («скільки білка/клітковини лишилося добрати сьогодні», «наскільки перевищена сіль»), а також ціль користувача (дефіцит/баланс/профіцит) у вигляді числового індикатора. Всі ознаки масштабуються, щоб довжина вектора не «вибухала»; це стабілізує навчання та робить порівняння товарів коректним.
Алергени обробляються на двох рівнях: продукти, що містять алергени з профілю, взагалі не подаються у ранжування; додатково можемо тримати бінарний індикатор «ризик» в ознаках, аби модель «запам'ятовувала» різницю між схожими товарами з/без ризику.

Принцип вибору. Кожному кандидату обчислюємо $\boldsymbol{p}(\boldsymbol{x})$. Якщо про варіант знаємо мало — бонус невизначеності великий, і він отримує шанс «проявитися»; якщо впевнені у його користі — обираємо його. Єдиний керуючий параметр $\boldsymbol{\alpha}$задає рівень «сміливості»: більше $\boldsymbol{\alpha}$— активніше дослідження нового.

Фідбек і оновлення. Мінімальний фідбек — бінарний (1 — прийнято, 0 — відхилено), можливий і безперервний [0..1]. Після кожної взаємодії оновлюємо достатні статистики:

$$\boldsymbol{A \leftarrow A + xx^{\top}, b \leftarrow b + rx}.$$

Наївне перевирахування $\boldsymbol{A^{-1}}$коштує $\boldsymbol{O(d^3)}$, що повільно для мобільного. Тому застосовуємо формулу Шермана–Моррісона [16] — класичний прийом інкрементального оновлення оберненої матриці при ранґ-1 зміні:

$$(A + u\,v^T)^{-1} = A^{-1} - \frac{A^{-1}u\,v^T A^{-1}}{1 + v^T A^{-1} u}$$

У нашому випадку $\boldsymbol{u = v = x}$, тож:

$$(A + x\,x^T)^{-1} = A^{-1} - \frac{A^{-1}x\,x^T A^{-1}}{1 + x^T A^{-1} x}$$

Це знижує складність оновлення до $\boldsymbol{O(d^2)}$, не потребує сторонніх бібліотек, добре працює on-device і стабілізується простою L2-регуляризацією $\boldsymbol{\lambda I}$(а також перевіркою малого знаменника). Інтуїтивно ми «уточнюємо» впевненість моделі саме в напрямку спостереженого вектора ознак.

Старт і безпека. На початку задаємо «здоровий» пріор: білок/клітковина у плюс; цукри/сіль — у мінус. Перші 50–100 кроків доцільно додати невеликий $\boldsymbol{\varepsilon}$-жадібний шум, щоб прискорити навчання. Продукти з алергенами йдуть у фільтр до ранжування, а не «покладаються» на покарання в скорі — це політика безпеки.

Адаптація й «забування». Щоб реагувати на зміну дієти/цілей, використовуємо ковзне вікно останніх взаємодій або експоненційне затухання впливу старих подій. Так модель швидко переорієнтовується без повного «перенавчання».

Пояснюваність та інтерфейс. Користувач бачить розклад скорингу: де саме бонус (білок, клітковина), де штраф (сіль, цукри). Пояснення просте: «Прогноз + Бонус за невпевненість = Підстава рекомендації». При близьких балах застосовуємо зрозумілі тай-брейки (більше клітковини, менше солі/цукру).

Складність і зберігання. Оцінка $\boldsymbol{p(x)}$та оновлення — $\boldsymbol{O(d^2)}$на крок; для $\boldsymbol{d \approx 10}$–$\boldsymbol{20}$ **-** це мілісекунди на iPhone. Стан моделі — кілька кілобайт, зберігається у UserDefaults/локальному сховищі.

2.8 Архітектура застосунку iOS та модель даних

Система складається з модулів: Камера/OCR; Виділення структур; Нормалізація; Обчислення похідних; Персональний скоринг; Адаптивний модуль; Локальна БД; Інтерфейс користувача. Потік даних: кадр → текст → структуровані

поля → нормалізовані ознаки → скоринг → рекомендація з поясненням. Для підвищення продуктивності застосовано кешування результатів за штрихкодом.

Модель даних включає сутності Product (GTIN, назва, поживність на 100 г/порцію, атрибути), Scan (журнал сканувань), Profile (цілі та ліміти), Lexicon (словник інгредієнтів/алергенів), BanditState (параметри адаптивного модуля). Передбачено механізми міграції схеми БД при оновленнях та інкрементального імпорту локального зрізу довідника (табл. 2.4).

Неперервність роботи забезпечується обробниками помилок (OCR-невдачі, нестача даних, конфлікти одиниць). Для користувача система надає зрозумілі підказки, а також мінімалістичні елементи UI для контролю порції та ваг ознак у скорингу.

Таблиця 2.4

Основні сутності локальної БД

Сутність	**Ключові поля**	**Призначення**	**Примітки**
Product	gtin, name, brand, nutrition100, perServing	Картка продукту	OCR/штрихкод/довідник
Scan	timestamp, imageHash, ocrText, confidence	Журнал сканувань	Валідація/відлагодження
Profile	targets (T_P,T_F,T_C), limits	Профіль користувача	UI-налаштування
Lexicon	token, canonical, tags	Словник	синоніми/алергени/E-номери
BanditState	A, b, counts, alpha	Стан адаптивного модуля	локальні оновлення

2.9 Приватність, безпека та етичні аспекти

Принцип «приватність за замовчуванням»: усі обчислення виконуються локально, зображення та текст OCR не передаються на сервери. Дані зберігаються мінімально необхідний час; резервне копіювання покладається на механізми системи із шифруванням. Анонімізовані журнали можуть бути відключені або очищені користувачем у будь-який момент.

Юридичні аспекти використання відкритих довідників (наприклад, Open Food Facts) вирішуються через атрибуцію та дотримання умов ліцензії. Етичні застереження щодо немедичного характеру рекомендацій відображаються в інтерфейсі; алгоритм пояснює, які фактори вплинули на оцінку, що підвищує довіру й керованість.

2.10 Метрики якості та методика експериментів

Якість OCR оцінюється метриками CER/WER на локальній вибірці етикеток (різні бренди, умови освітлення, варіанти шрифтів). Якість парсингу — часткою коректно витягнутих полів (Б/Ж/В/сіль/цукри/клітковина) по відношенню до ручної розмітки; додатково — точність перетворень одиниць і перевірок цілісності.

Персоналізацію перевіряємо простими показниками: яку частку рекомендацій користувач приймає; наскільки в середньому зменшується відхилення від денних цілей (після вибору продукту); а також за офлайн-replay — проганяємо історичні сесії через алгоритм і дивимося, наскільки рішення близькі до фактичних. Додатково стежимо, щоб ваги вподобань лишалися стабільними під час зміни смаків. Щоб результати були переконливими, будуємо довірчі інтервали простим методом бутстрепу: багаторазово пере-вибірковуємо дані й дивимося на розкид метрик.

Висновки

У розділі вибудувано повний теоретико-методичний фундамент персоналізованого добору харчових продуктів у мобільному середовищі. Сформульовано вимоги до даних етикетки, описано on-device пайплайн OCR, алгоритми парсингу та нормалізації, а також процедури приведення величин до порції та стандартизованих одиниць. Запропонована модель персонального скорингу поєднує наближення до індивідуальних цілей Б/Ж/В із контролем обмежень (сіль, насичені жири, додані цукри) і забезпечує пояснюваність через декомпозицію внесків ознак.

Показано, як адаптивний модуль на базі контекстних бандитів (LinUCB) дозволяє пристроєві навчатися з поведінки користувача без відправки даних назовні, підтримуючи баланс між дослідженням нових альтернатив і експлуатацією вже перевірених. Розглянуто архітектуру iOS-застосунку, локальну модель даних та механізми кешування/міграцій, що забезпечують масштабованість і надійність рішення.

Окремо проаналізовано питання приватності, безпеки та етики: алгоритм працює локально, дані зберігаються мінімально, а пояснюваний інтерфейс дає користувачеві контроль над вагами, порціями та цілями. Запропоновано чіткий набір метрик і план експериментів, який дозволяє не лише оцінити точність парсингу та продуктивність, а й кількісно показати користь персоналізації через зменшення відхилення від денних цілей та підвищення прийнятності рекомендацій.

Таким чином, друга глава формує методичну базу, достатню для реалізації прототипу й подальшої експериментальної валідації. На її основі в наступних розділах буде представлено конкретні рішення щодо архітектури коду iOS-застосунку, вибору форматів зберігання та інтерфейсів, а також результати порівняльних експериментів із різними конфігураціями скорингу та адаптивного модуля.

РОЗДІЛ 3.
ПРОГРАМНА РЕАЛІЗАЦІЯ ЗАСТОСУНКУ ДЛЯ АНАЛІЗУ ХАРЧОВИХ ПРОДУКТІВ ТА РЕКОМЕНДАЦІЙ

3.1 Опис програмної реалізації та етап реєстрації користувача

Мобільний застосунок реалізовано нативно під iOS (Swift, UIKit, AVFoundation, Vision) [16]. Дані про продукти додатково збагачуються через Open Food Facts (OFF) API. Персональні дані (профіль, обрані продукти, стан рекомендацій) зберігаються локально на пристрої з де-дуплікацією за штрих-кодом.

Архітектура модулів:

Сканер і OCR (AVFoundation + Vision): камера формує потік кадрів; Vision виконує розпізнавання тексту [17-19] з акумуляцією кількох кадрів, що підвищує стійкість до тремтіння; паралельно працює детектор штрих-кодів (EAN/UPC/QR).

Парсер нутрієнтів та словник алергенів: виділення Б/Ж/В, клітковини, цукрів, солі з толерантністю до OCR-помилок; пошук токенів алергенів та їх агрегація за категоріями; коментар і «оцінка продукту».

Модуль штрих-коду (OFF API): отримання назви (product_name|product_name_uk), зображення (selected_images.front.display) та нутрієнтів (*_100g|*_serving), із валідацією значень (діапазон 0–100 г).

Локальне зберігання: профіль (мета, алергії, щоденні цілі БЖВ, аватар), продукти, стан алгоритму рекомендацій; унікальність запису за штрих-кодом (оновлення замість дублю).

Рекомендації LinUCB: контекстний бандит для персонального ранжування (баланс «дослідження–експлуатація»), навчання на пристрої.

Реєстрація (онбординг):

1) Введення профілю: ім’я, активність, ціль харчування (дефіцит / баланс / набір).

2) Вибір алергенів/непереносимостей зі словника (використовується для фільтрації та попереджень).

3) Автоматичний розрахунок добових цілей БЖВ і збереження в профілі.

Гігієна даних:

1) Макро-значення нормалізуються до [0; 100] г (на 100 г або на порцію).

2) Якщо штрих-код уже є — запис оновлюється, а не дублюється.

3) Назва/фото беруться з надійних OFF-полів (fallback відключається, якщо поле порожнє/аномальне).

3.2 Опис програмної реалізації мобільного застосунку

Екран реєстрації: на екрані знаходяться поля вводу імені, ваги, зросту, ваги, статі, цілі (дефіцит, баланс чи набір), та рівень активності (низька, легка, середня чи висока) користувача.

Екран вибору алергенів: на екрані знадятиться набір з кнопок з можливістю множинного вибору, для визначення алергенів.

Головний екран: у верхній частині — картка цільових БЖВ з профілю (білки, жири, вуглеводи). Нижче — стрічка персональних рекомендацій (результат LinUCB) із коротким поясненням та оцінкою. Підказка під карткою заохочує додати кілька «улюблених» товарів зі штрих-кодом для покращення персоналізації.

Екран сканера: одночасно працюють два канали:

OCR-канал: розпізнає таблицю харчової цінності та/або склад. Текст накопичується з кількох кадрів; «м’якші» правила зупинки — успіх, якщо знайдено ≥3 поля з множини {білки, жири, вуглеводи, клітковина, цукри, сіль}.

Штрих-код (OFF): при впевненому зчитуванні EAN/UPC отримуємо назву, фото та нутрієнти. Невалідні/аномальні значення відкидаються.

Паралельні канали обробки. Сканер запускає два взаємодоповнювальні конвеєри. Канал OFF активується за впевненого зчитування EAN/UPC у прямокутній ROI; через OFF API отримуємо назву, фото та мапу нутрієнтів із пріоритетом *_100g і валідацією діапазонів (0–100 г/100 мл). Канал OCR паралельно накопичує текст з кількох кадрів (стабілізація), виділяє таблицю

поживності/склад, нормалізує одиниці та парсить основні поля Б/Ж/В/клітковина/цукри/сіль. Першим «лочиться» той канал, який дає валідні ≥3 поля нутрієнтів або інгредієнти з відпрацьованим словником алергенів. Це зменшує латентність і підвищує надійність за «шумних» етикеток.

Картка результатів: короткий коментар, оцінка, БЖВ (на 100 г чи на порцію), список алергенів із позначкою для персональних ризиків. Кнопки «Зберегти» / «Новий пошук». На екрані деталей збереженого продукту кнопки «Зберегти» немає.

3.3 Розробка інтерфейсу та сценаріїв використання

Основні сценарії:

Реєстрація: мета, алергени → розрахунок щоденних БЖВ. Екрани зображені на рискунках 3.1 та 3.2.

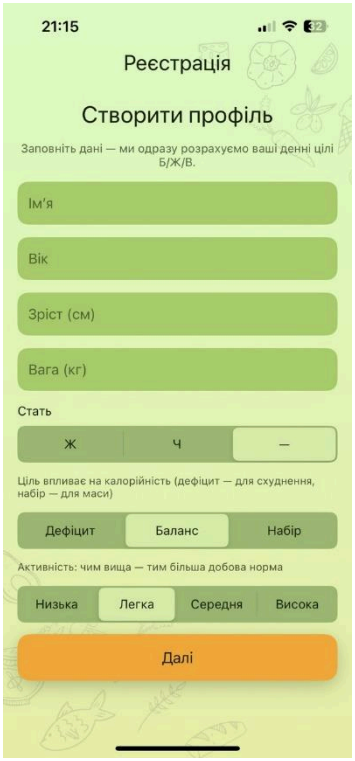

Рис. 3.1. Екран реєстрації програмного застосунку

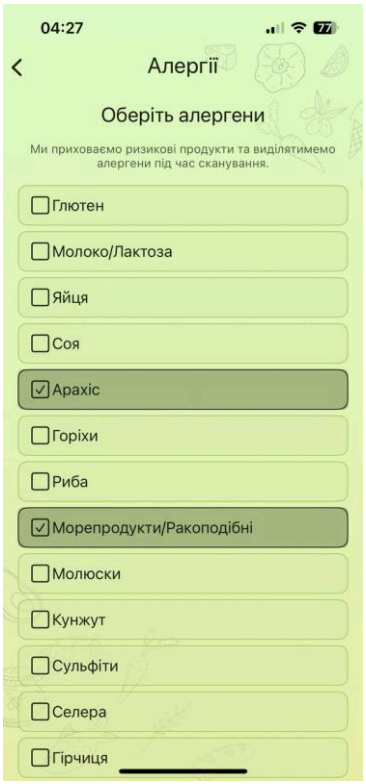

Рис. 3.2. Екран вибору алергенів програмного застосунку

Сканування: наведення на таблицю/склад або штрих-код; паралельний OCR+OFF. Екран зображено на рискунку 3.3.

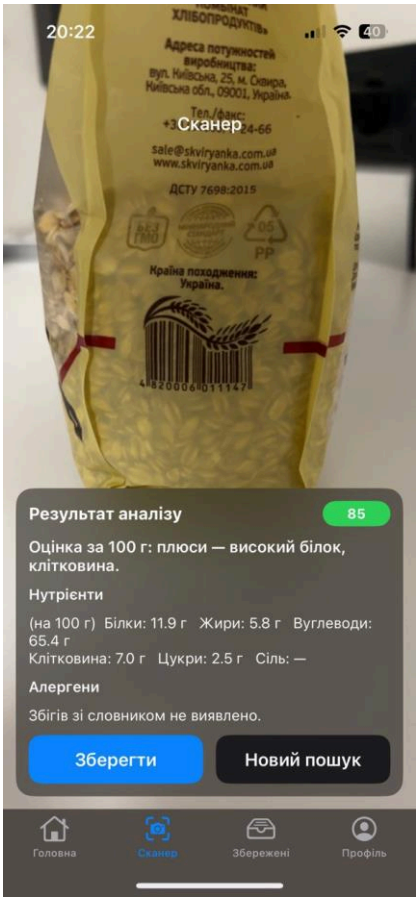

Рис. 3.3. Екран сканера продукта та виведення результату аналізу

Інтерактивні елементи. Сканер має перемикач режимів (Штрих-код/OCR) та напівпрозорий прямокутник ROI для фокусування на штрих-коді. Картка результату пропонує «Зберегти» або «Новий пошук». На головному екрані додано кнопку «Оновити рекомендації», що перераховує бали LinUCB для поточних товарів профілю. У деталях товару доступне редагування нутрієнтів (на 100 г) і алергенів через мінімалістичний аркуш із прокруткою.

Збереження: створення/оновлення (унікальність за штрих-кодом), додавання фото. Екран зображено на рискунку 3.4.

Рис. 3.4. Екран збереження просканованого продукту

Перегляд: детальна сторінка у стилі профілю: БЖВ, оцінка, алергени, назва, зображення. Також екран редагування даних користувача. Екрани зображено на рисунках 3.5 та 3.6.

Рис. 3.5. Екран перегляду профіля

Рис. 3.6. Екран редагування профіля

Екран перегляду внутрішньої бази збережених товарів, з індивідульною оцінкою, з можливістю видалення непотрібних товарів. Екран зображено на рисунку 3.7.

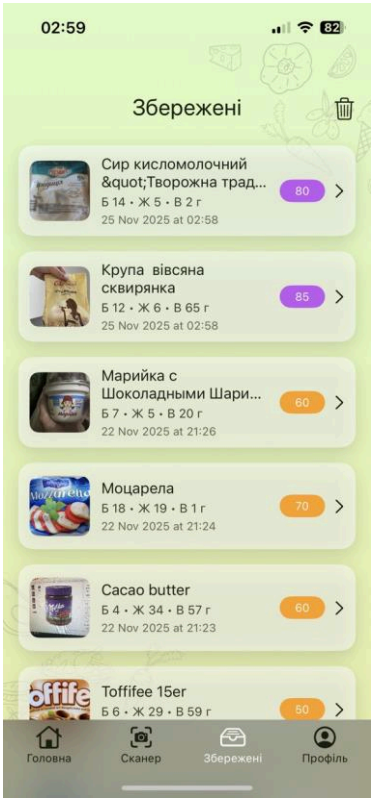

Рис. 3.7. Екран перегляду збережених товарів

Головний екран, на якому таблицей відображено результат роботи алгоритму, оцінка товару, можливість оновити дані зображено на рисунку 3.8.

Рис. 3.8. Екран перегляду збережених товарів

Початковий запуск (cold-start): пропонуємо додати кілька відомих товарів (по 3–4 штрих-коди під кожну мету), щоб одразу «навчити» модель і покращити персоналізацію з перших хвилин.

На рисунку 3.9 зображено повний флоу роботи застосунку.

Рис. 3.8. Флоу роботи застосунку

3.4 Особливості локальної БД та алгоритм рекомендацій LinUCB

Моделі даних:

1. UserProfile — мета, алергени, щоденні цілі БЖВ, фото/аватар.

2. SavedProduct — barcode?, назва, зображення, нутрієнти, відбиток алергенів, дата створення/оновлення.

3. LinUCB — параметри та статистики LinUCB (матриці/вектори для кожного «рукава»).

Графічне зображення наведених вище моделей зображено на рисунках 3.9-3.11.

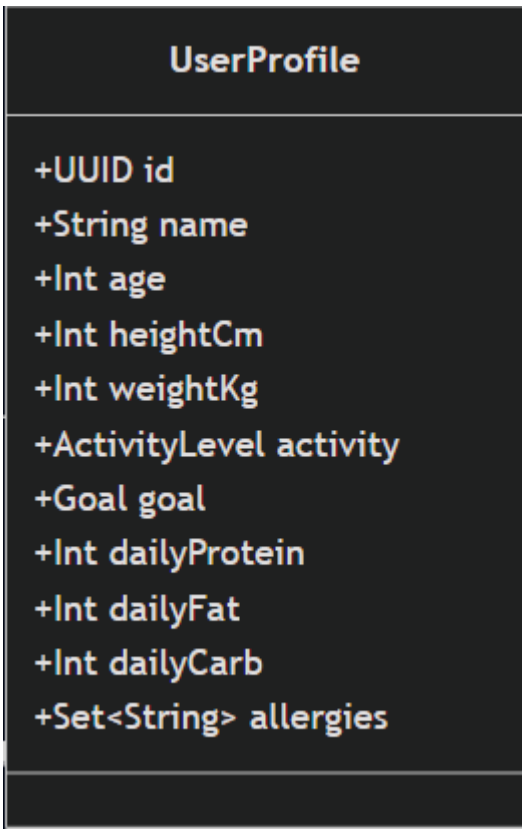

Рис. 3.9. Графічне зображення моделі UserProfile

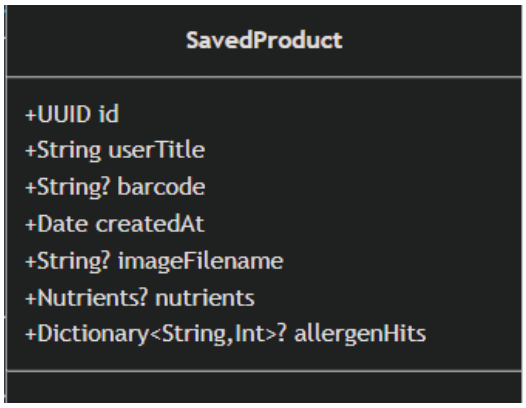

Рис. 3.10. Графічне зображення моделі SavedProduct

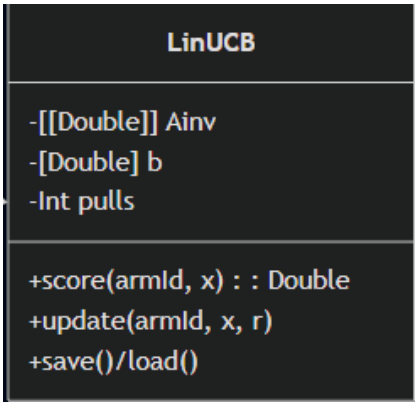

Рис. 3.11. Графічне зображення моделі LinUCB

Зберігання і унікальність:

Серіалізація у файлову систему. При додаванні товару з уже існуючим штрих-кодом — оновлення запису, а не дублювання (це запобігає крашам і роздуттю списку). Можлива подальша міграція на Core Data/SQLite без зміни публічного API.

Алгоритм LinUCB (контекстний бандит):

Кожен продукт — «рукав», контекст — вектор ознак (відхилення від цілей БЖВ, наявність алергенів, базові смакові/категорійні ознаки). Для рукава підтримуються матриця A, вектор b, оцінка θ. Вибір здійснюється за UCB: score(x) = $\theta^T x + \alpha * \text{sqrt}(x^T A^{-1} x)$, де α — параметр дослідження. Після взаємодії/збережень/виборів користувача виконується оновлення параметрів рукава. Навчання виконується локально (дані не покидають пристрій), що підвищує приватність. Нижче наведено код на мові програмування Swift, який реалізує даний алгоритм:

```
final class LinUCB: Codable {
   let d: Int; let alpha: Double
   private var arms: [String: ArmState] = [:]
   init(dimension: Int, alpha: Double = 0.8) { self.d = dimension; self.alpha = alpha }
   private func ensure(_ id: String) {
      if arms[id] != nil { return }
```

```
        let I = (0..<d).map { i in (0..<d).map { j in i==j ? 1.0 : 0.0 } }
        arms[id] = ArmState(Ainv: I, b: Array(repeating: 0.0, count: d))
    }
    func score(arm id: String, context x: [Double]) -> Double {
        ensure(id); guard let st = arms[id] else { return 0 }
        let theta = matVec(st.Ainv, st.b)
        let mean  = dot(theta, x)
        let Ax    = matVec(st.Ainv, x)
        let ucb   = sqrt(max(0, dot(x, Ax)))
        return mean + alpha * ucb
    }
    func update(arm id: String, context x: [Double], reward r: Double) {
        ensure(id); guard var st = arms[id] else { return }
        rankOneInverseUpdate(Ainv: &st.Ainv, with: x)
        for i in 0..<d { st.b[i] += r * x[i] }          // XTy ← XTy + r·x
        arms[id] = st
    }
}
```

Саніти-перевірки OFF:

Макро-поля обрізаються до [0; 100] г; аномалії (NaN, ∞, від'ємні) ігноруються. Назва бере значення з product_name|product_name_uk; зображення — з selected_images.front.display. Якщо OFF не надіслав корисних даних — пріоритет за OCR-результатом.

3.5 Продуктивність і відмовостійкість

Система проєктувалася з акцентом на швидку реакцію інтерфейсу та стабільність в умовах «польового» використання — різне освітлення, хитання камери, нестабільний інтернет, часткова відсутність даних у відкритих каталогах [20]. На рівні відеопотоку застосовано пресет 720p, що мінімізує навантаження на CPU/GPU й енергоспоживання, але забезпечує достатню роздільну здатність для

OCR та детекції EAN/UPC. Кадри обробляються з дроселюванням (≈0.8 с між аналізами) і накопиченням до 250 рядків тексту з кількох послідовних кадрів. Акумуляція тексту робить розпізнавання стійким до розфокусування й “motion blur”: навіть якщо окремі символи в різних кадрах розпізнаються по-різному, сукупна вибірка дає стабільний результат для таблиці харчової цінності та списку інгредієнтів.

Паралельні канали отримання даних — OFF (через штрихкод) та OCR — працюють незалежно і конкурують за право «зафіксувати» результат. Канал OFF активується в межах прямокутної ROI, що зменшує хибні спрацьовування та скорочує час до першої валідної відповіді; OCR продовжує фонове накопичення й парсинг, підхоплюючи випадки, коли у відкритій базі немає запису або мережа нестабільна. Механіка «перший, хто дав валідні ≥3 нутрієнти чи осмислений список інгредієнтів — лочиться» гарантує мінімальну латентність і прогнозованість UI. Якщо один канал не спрацював (наприклад, OFF повернув «product not found»), інший продовжує роботу без пауз і перезапусків, тож користувач не «втрачає момент».

З боку мережі застосовуються тайм-аути, повтор із альтернативного домену OFF, а також легкий кеш за штрихкодом, що скорочує час другої та наступних перевірок одного й того самого продукту. Дані з OFF проходять валідаційний шар: нормалізація десяткового розділювача, відсікання нечислових і негативних значень, а також обрізання очевидних аномалій за верхніми межами (для показників “на 100 г/мл”). Це захищає інтерфейс від «викидів» та неадекватних карток.

Парсер нутрієнтів і детектор алергенів оптимізовано на векторних перетвореннях рядків і попередньо скомпільованих регулярних виразах. Словник алергенів плоский, але групується в категорії лише під час відображення; такий підхід дає швидке обчислення, а структура категорій використовується повторно в декількох екранах. Для збереження акуратної пам’яті прибираються проміжні

масиви (наприклад, очищення акумулятора OCR після фіксації результату), а великі об’єкти (зображення) зберігаються як файли з посиланнями в локальній БД. Вміст користувача та стан моделі LinUCB серіалізуються після змін маленькими порціями, що мінімізує ризик втрати прогресу й не блокує головний потік.

Оновлення моделі виконується повністю на пристрої за формулою Шермана–Моррісона. Це ранґ-одиничне оновлення оберненої матриці знижує обчислювальну складність із кубічної до квадратичної від розмірності ознак і дозволяє оновлювати переваги практично миттєво після кожної взаємодії. Безпека користувача гарантується подвійним бар’єром: по-перше, жорстка фільтрація кандидатів за алергенами профілю (ризикові товари взагалі не подаються в ранжування), по-друге, на рівні ознак присутній явний індикатор «ризику алергену», що підсилює штрафи навіть при часткових збігах у складі.

Відмовостійкість підтримується через явні гілки деградації. За відсутності мережі OFF-канал пропускається, а OCR продовжує роботу. Якщо освітлення або фокус перешкоджають стабільному OCR, користувач отримує підказку фокусуватися на штрихкоді (режим «Штрихкод» з ROI). Усі помилки зовнішніх сервісів обробляються без «крашів» та без перешкод для жестів/навігації. Додатково в інтерфейсі передбачено ручне редагування нутрієнтів “на 100 г” і ручний вибір алергенів зі словника — це дає змогу коригувати неточності сканування та доповнювати дані, які не знайшлися у відкритих джерелах.

Для контролю якості передбачено прості метрики експлуатації, що не розкривають персональних даних: частка «успішних» локацій за OCR/OFF, середній час до валідного результату, частка ручних редагувань після сканування, коефіцієнт «not found» у OFF за регіональними штрихкодами. Ці метрики використовуються виключно локально або в анонімізованому вигляді — вони допомагають пріоритезувати подальші покращення парсера, словника та UX-підказок, не впливаючи на приватність [22-23].

3.6 Відповідність стандартам розробки мобільного програмного застосунку

У процесі проєктування та реалізації застосунку дотримано низки галузевих стандартів і настанов, що забезпечують якість, безпеку та зручність використання.

Інтерфейс та UX. Орієнтація на Apple Human Interface Guidelines та Swift API Design Guidelines [24-25] гарантує консистентність навігації, типографіки, жестів (tap/drag/dismiss), а також зрозумілі патерни редагування (аркуші з підтвердженням, інлайнові підказки). Для доступності застосовано принципи WCAG 2.1 AA та iOS Accessibility: контраст, VoiceOver, великі цілі натискання, динамічні шрифти.

Безпека й приватність. Для мобільної безпеки орієнтиром слугує OWASP MASVS (вибірково L1/L2): мінімізація даних, захист локального сховища, відсутність чутливих логів, валідація вхідних даних, коректні таймаути мережі. Принципи GDPR/App Store Privacy — локальна обробка скану, відсутність передачі персональних харчових вподобань у сторонні сервіси; зовнішні запити (OpenFoodFacts) не містять персональних ідентифікаторів.

Якість ПЗ та продуктивність. У відповідності до ISO/IEC 25010 забезпечено цільові атрибути: функціональна придатність (сталі пайплайни «Barcode/OCR→Парсер→Скоринг»), надійність (деґрадаційні сценарії: OFF↔OCR; кеш), продуктивність (обмеження латентності, інкрементні оновлення LinUCB), зручність супроводу (модульна архітектура, окремі сервіси). Дані в обміні — JSON, узгоджено з RFC 8259; версіонування вимог і API — SemVer; перевірки збірок — CI-скрипти/локальні тести.

Трасованість та пояснюваність. Ключові рішення в інтерфейсі пояснюються: «прогноз + бонус невпевненості» у LinUCB, розклад внесків за ознаками (protein/fiber/sugars/salt), маркування ризикових інгредієнтів і збігів зі словником алергенів.

Мапа відповідності. Відповідність стандартів конкретним модулям відображено на рисунку 3.12.

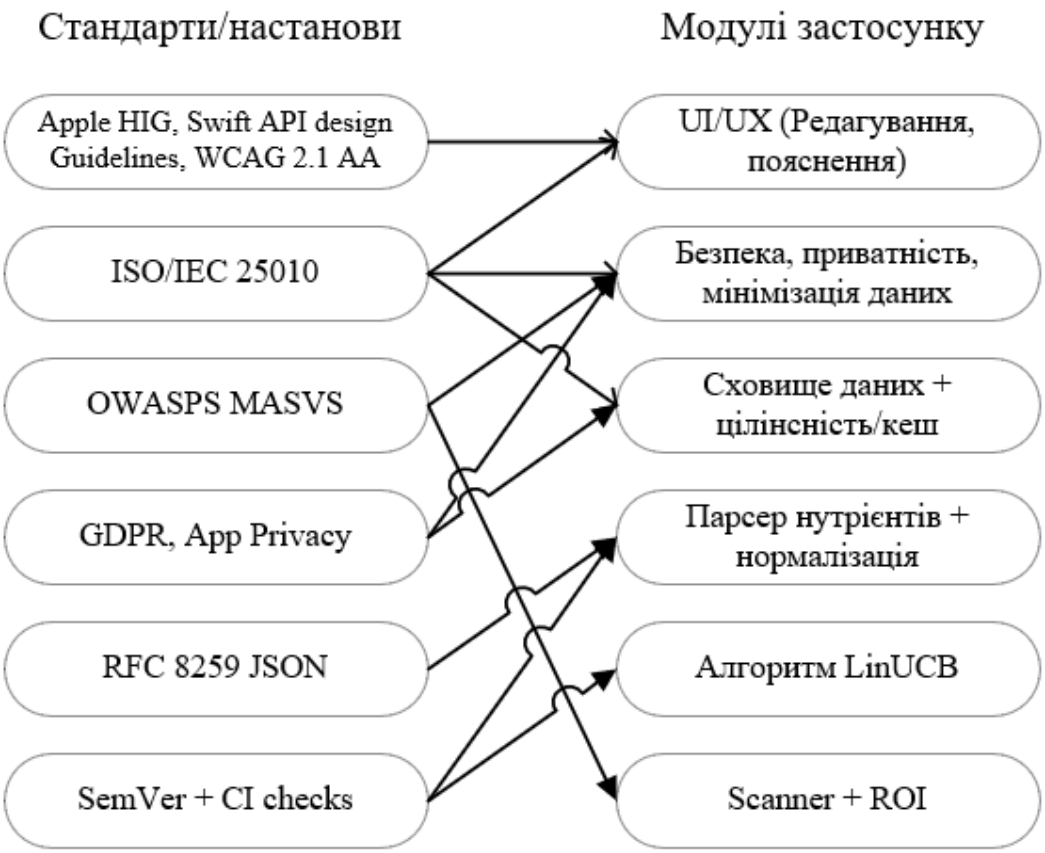

Рис. 3.12. Відповідність стандартів конкретним модулям

Висновки

У межах розділу реалізовано та інтегровано повний технологічний ланцюжок від зчитування етикетки до персональної рекомендації з поясненням. Паралельні канали — штрихкод із запитом до OpenFoodFacts та OCR-аналіз етикетки — взаємно страхують один одного: якщо відкритий каталог не має запису або мережа обмежена, систему підхоплює локальний парсер; якщо умови зйомки ускладнюють OCR, домінує OFF. Валідаційний шар і нормалізація даних роблять картки стабільними, а ручне редагування дає користувачу контроль і можливість коригувати нестандартні випадки.

Алгоритм LinUCB забезпечує швидке on-device-навчання й баланс «дослідження–експлуатація» без складних серверних компонентів. Застосування формули Шермана–Моррісона для інкрементного оновлення інверсії робить адаптацію миттєвою навіть на мобільному процесорі, а інженерні рішення щодо ознак (поживні властивості на 100 г, індикатори алергенів, профіль користувача) гарантують осмислену динаміку балів. Додаткові UX-деталі — ROI для штрихкоду, перемикач режимів, аркуші редагування, кнопка «Оновити рекомендації» — перетворюють алгоритм на зрозумілий користувачеві продукт: система не тільки підказує, а й пояснює, що саме вплинуло на оцінку.

З погляду надійності, ланцюжок має чіткі сценарії деградації та відновлення: OFF і OCR працюють незалежно; падіння одного каналу не блокує інший; усі помилки зовнішніх джерел перехоплюються без втрати чутливості інтерфейсу. Локальне кешування, безпечні межі для числових полів і контролі цілісності в БД роблять поведінку передбачуваною навіть за неякісних етикеток і нестабільної мережі. У підсумку розділ демонструє, що обрана архітектура та стек (Swift + AVFoundation/Vision + локальна БД + OFF як довідник) дозволяють створити практичний, енергоощадний і масштабований прототип, готовий до подальшого

розширення словників, підключення регіональних джерел та тонкого налаштування моделей ознак під різні категорії продуктів.

Висновок

У кваліфікаційній роботі розв’язано комплексну задачу персоніфікованого добору харчових продуктів у мобільному середовищі шляхом поєднання трьох взаємодоповнювальних компонентів:

1. Локального витягу даних з етикеток двома паралельними каналами (штрихкод/OpenFoodFacts та OCR з робастним парсером нутрієнтів).

2. Прозорої моделі персонального скорингу, яка співвідносить профіль користувача з «поживними» ознаками продуктів.

3. Адаптивного модуля LinUCB (контекстні бандити) для on-device навчання на поведінковому фідбеку.

Розроблений iOS-застосунок забезпечує «миттєву» рекомендацію в місці прийняття рішення (біля полиці/у магазині), зберігаючи приватність, надійність і пояснюваність.

Досягнуті результати та їх значущість

Джерела даних і робастність вводу. Запропоновано і реалізовано подвійний вхідний ланцюжок:

– Штрихкод → OFF: коли продукт наявний у відкритому каталозі, миттєво отримуємо назву, базові нутрієнти *_100g*, зображення та (за наявності) інгредієнти;

– Камера → OCR → Парсер: коли каталожного запису немає або мережа нестабільна, локальний алгоритм витягує таблицю харчової цінності, нормалізує позначення і приводить значення до базового масштабу «на 100 г/мл». Така редундантність каналів забезпечує стійкість системи: відмова одного способу не блокує інший. Додано нормалізацію одиниць, обробку OCR-помилок і словникові правила, що покращують якість парсингу на «шумних» етикетках.

Персональний скоринг та пояснюваність «чому цей продукт». Побудовано інтерпретовану модель балу, що акумулює внески ознак (protein, fiber

— «плюс»; sugars, salt — «мінус»; тощо) відносно індивідуальних цілей користувача (дефіцит/баланс/профіцит, активність). У застосунку користувач отримує текстові пояснення та може редагувати як нутрієнти (за потреби), так і ознаки ризику/алергени, що підсилює довіру і контроль.

On-device адаптація через LinUCB. Ключовим внеском є інтеграція LinUCB методу контекстних бандитів, який у реальному часі балансує між експлуатацією «відомо корисних» варіантів і дослідженням «нових/невпевнених». Модель працює локально, зберігаючи лише компактні матриці/вектори (кБ), без відправки персональних даних. Інкрементне оновлення параметрів реалізовано через формулу Шермана–Моррісона, що робить інверсійні перерахунки мілісекундними на мобільному процесорі. Практична користь: система швидко підлаштовується під реальні вибори користувача, а не нав'язує «середні» рейтинги.

Архітектура, UX та надійність. У межах iOS-екосистеми (Swift, AVFoundation, Vision, локальна БД) спроєктовано повний технологічний ланцюжок:

1. Модуль сканера з ROI для штрихкоду та перемикачем «Barcode/OCR».

2. «Аркуші» редагування нутрієнтів і алергенів з адаптацією під клавіатуру, свайп-жестами закриття, підказками.

3. Кнопка оновлення рекомендацій та таблиця ранжування із «бейджем» балу.

Система має чіткі сценарії деградації (відмова OFF не впливає на OCR і навпаки), безпечні межі для числових полів та контроль цілісності, що гарантує стабільність інтерфейсу.

Валідація та метрики. Запропоновано набір метрик для трьох рівнів:

1. Якість парсингу: частка правильного витягу ключових полів (*protein_100g*, *fat_100g*, *carbohydrates_100g*), точність розпізнавання «на 100 г»;

2. Продуктивність: затримка «скан → рекомендація», споживання пам'яті й енергії.

3. Користь персоналізації: зменшення відхилення від денних цілей Б/Ж/В, зростання частки прийняття рекомендацій, вплив на різні цілі (дефіцит/баланс/профіцит).

Наукова новизна полягає у поєднанні контекстних бандитів LinUCB з on-device витягом харчових ознак (OFF/OCR) у режимі реального часу. Робота демонструє, що лінійна модель із оптимістичною границею (UCB) достатньо потужна для персоналізації «тут-і-зараз», а її простота дає прозорі пояснення і стабільну роботу на мобільному пристрої без серверної складності.

Практична цінність — у створенні життєздатного прототипу iOS-застосунку, який розв'язує реальну прикладну проблему: вибір кращого продукту під конкретні цілі та обмеження користувача в магазині, з урахуванням алергенів і точного нормування «на 100 г».

Обмеження, ризики та способи їх пом'якшення

1. Неповнота/якість каталогу OFF. Не всі локальні продукти присутні в БД; поля можуть бути частково заповнені. Пом'якшення: OCR-канал і можливість ручного редагування; локальне кешування; готовність до підключення регіональних джерел та crowdsourcing-оновлень.

2. Чутливість OCR до умов зйомки. Бликіт, малий шрифт, викривлення впливають на точність. Пом'якшення: ROI, адаптивна обробка, словниковий шар, фільтри «явного сміття», накопичення тексту з кількох кадрів, чіткі правила зупинки аналізу.

3. Нормування та інтерпретація «на 100 г/порцію». Некоректне трактування порцій може спотворювати скоринг. Пом'якшення: у прототипі уніфікація до «на 100 г/мл»; у дорожній карті — безпечне додавання «порцій» з підказками.

4. Баланс explore/exploit. Занадто мала/велика αзмінює поведінку системи. Пом'якшення: калібрування αна валідаційних сценаріях, опційна ε-жадібність на старті, контрольні «стоп-правила» (заборона алергенів до ранжування).

UX-ризики довіри. Користувач має розуміти «чому». Пом’якшення: пояснювані тексти, видимість внесків ознак, можливість редагування і підтвердження змін.

Розроблений прототип iOS-застосунку є технічно завершеним демонстратором вирішення задачі з чітким потенціалом масштабування: підключення нових джерел, розширення словників і поглиблення моделей ознак. Отримані результати підтверджують доцільність обраної архітектури та методів і слугують ґрунтовною базою для подальшого прикладного та наукового розгортання систем персоніфікованих харчових рекомендацій.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Global action plan for the prevention and control of noncommunicable diseases 2013-2020. Geneva : World Health Organization, 2013. 103 p.
2. Захаріна Є. А., Мазін В. В. Розробка мобільного застосунку для оптимізації харчування в силовому фітнесі. *Спортивний вісник Придніпров'я.* 2023. № 1. С. 134–141.
3. Попова Т. Персоналізоване харчування як тренд сучасної нутриціології. *Харчові технології та інженерія.* 2024. Т. 6, № 1. С. 45–52.
4. Husted S., et al. Digital food environments: a systematic review of the apps and their impact on consumer behavior. *Current Developments in Nutrition.* 2020. Vol. 4, Issue 11. Art. nzaa163.
5. Julia C., Hercberg S. Development of a new front-of-pack nutrition label in France: the five-colour Nutri-Score. *Public Health Panorama.* 2017. Vol. 3, No. 4. P. 712–725.
6. Monteiro C. A., et al. NOVA. The star shines bright. *World Nutrition.* 2016. Vol. 7, No. 1–3. P. 28–38.
7. Open Food Facts. The open database of food products. URL: **https://world.openfoodfacts.org/** (дата звернення: 11.12.2025).
8. Січко Т. В., Нескородєва Т. В., Федоров Є. Є. Експертні та рекомендаційні системи : навч. посіб. Вінниця : ДонНУ імені Василя Стуса, 2023. 240 с.
9. Кошляк А. С., Бричковський О. Д. Рекомендаційна система для персоналізованого підбору книг. *Наукові інновації та передові технології.* 2025. № 5 (45). С. 421–430. DOI: 10.62731/mcnd-30.05.2025.011.
10. Висоцька В. А., Чирун Л. В. Технології проєктування мобільних застосунків : навч. посіб. Львів : Видавництво Львівської політехніки, 2020. 320 с.
11. Про захист персональних даних : Закон України від 01.06.2010 р. № 2297-VI. URL: **https://zakon.rada.gov.ua/laws/show/2297-17** (дата звернення: 12.12.2025).

12. Про затвердження Норм фізіологічних потреб населення України в основних харчових речовинах і енергії : Наказ М-ва охорони здоров'я України від 03.09.2017 р. № 1073. *Офіційний вісник України.* 2017. № 99. Ст. 3041.

13. Шибицька Н. М. Інтелектуальні інформаційні системи підтримки прийняття рішень в управлінні проектами. *Управління проектами та розвиток виробництва.* 2011. № 3 (39). С. 132–138.

14. Sutton R. S., Barto A. G. Reinforcement Learning: An Introduction. 2nd ed. Cambridge : MIT Press, 2018. 552 p.

15. Sherman J., Morrison W. J. Adjustment of an Inverse Matrix Corresponding to a Change in One Element of a Given Matrix. *The Annals of Mathematical Statistics.* 1950. Vol. 21, No. 1. P. 124–127.

16. Sahar A. iOS 17 Programming for Beginners: Unlock the world of iOS development with Swift 5.9, Xcode 15, and iOS 17. 8th ed. Birmingham : Packt Publishing, 2023. 604p.

17. Терещенко В. М. Розпізнавання образів та комп’ютерний зір : конспект лекцій. Київ : НУБіП України, 2020. 115 с.

18. Apple Inc. Vision Framework Documentation. URL: **https://developer.apple.com/documentation/vision** (дата звернення: 11.12.2024).

19. Шибицька Н.М., Громяк О.А. Засоби створення інтерактивного інтерфейсу систем автоматичного управління //Збірник тез науково-практичної конференції "Мультимедійні технологій в освіті та інших сферах діяльності". - К.: НАУ, 2021. – С.128-130.

20. Зуб К. В. Методи та засоби побудови рекомендаційних систем для вибору освітньої траєкторії : дисертація директора філософії : 122 Комп'ютерні науки. Львів, 2024. 210с.

21. Цехмейструк Р. Рекомендаційні системи новинних платформ: типологія, принципи роботи та вплив на контент. *Молодий вчений.* 2024. № 4 (116). С. 12–18.

22. Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data (GDPR). *Official Journal of the European Union.* 2016. L 119. P. 1–88.
23. Шибицька Н.М., Кочеткова О.В. Інтеграційне тестування та контроль якості програмного забезпечення// Proceedings 1st international scientific and practical conference «Information Systems and Technology: Results and Prospects» (IST 2024)", March 6, 2024 - K.: FIT TSNUK, 2024.- С.366-369. http://kist.ntu.edu.ua/konferencii/46_konf_2024.pdf
24. Apple Human Interface Guidelines (developer.apple.com/design/human-interface-guidelines)
25. Swift API Design Guidelines (swift.org/documentation/api-design-guidelines)
26. OWASP MASVS (owasp.org/www-project-mobile-app-security)

ДОДАТКИ

Додаток А. Частина коду програми

ScannerViewController.swift

```
import UIKit
import AVFoundation
import Vision

final class ScannerViewController: UIViewController {

    private enum ScanState { case idle, analyzing, locked }
    private enum ScanMode { case barcode, ocr }

    private let session = AVCaptureSession()
    private let videoOutput = AVCaptureVideoDataOutput()
    private var previewLayer: AVCaptureVideoPreviewLayer!
    private let videoQueue = DispatchQueue(label: "camera.video.queue", qos: .userInitiated)
    private let sessionQueue = DispatchQueue(label: "camera.session.queue")

    private var isProcessing = false
    private var lastAnalysisTime: TimeInterval = 0
    private let analysisInterval: TimeInterval = 0.8

    private var textRequest: VNRecognizeTextRequest!
    private lazy var barcodeRequest = VNDetectBarcodesRequest(completionHandler: self.handleBarcodes)
    private let visionQueue = DispatchQueue(label: "vision.queue", qos: .userInitiated)
```

```
private let resultView = ResultView()
private let dimLayer = CAShapeLayer()
private let boxLayer = CAShapeLayer()
private let modeControl = UISegmentedControl(items: ["Штрихкод", "OCR"])
private let hint = UILabel()

private var scanState: ScanState = .idle {
    didSet { if scanState != .locked { resultView.showAnalyzing() } }
}
private var scanMode: ScanMode = .barcode { didSet { updateModeUI() } }
private var lastPayload: ResultView.ResultPayload?

private var lastBarcode: String?
private var lastProductName: String?
private var lastProductImageURL: URL?

private var isSessionConfigured = false
private var didShowDeniedOnce = false
private var alreadyFetchingBarcode = false

private let allergenLexicon: [String] = AllergenCatalog.flatLexicon
private var accumulatedLines = Set<String>()

override func viewDidLoad() {
    super.viewDidLoad()
    view.backgroundColor = .black
```

```
    title = "Сканер"
    configurePreview()
    configureOverlays()
    configureResultView()
    configureVision()
    configureModeUI()
}

override func viewWillAppear(_ animated: Bool) {
    super.viewWillAppear(animated)
    if scanState == .locked { resumeScanning() }
    if scanState != .locked { startIfAuthorized(); scanState = .analyzing }
}

override func viewWillDisappear(_ animated: Bool) {
    super.viewWillDisappear(animated)
    stopSessionAsync()
}

override func viewDidLayoutSubviews() {
    super.viewDidLayoutSubviews()
    previewLayer.frame = view.bounds
    dimLayer.frame = view.bounds
    boxLayer.frame = view.bounds
    layoutModeUI()
    updateModeUI()
}
```

```
private func configurePreview() {
    previewLayer = AVCaptureVideoPreviewLayer(session: session)
    previewLayer.videoGravity = .resizeAspectFill
    view.layer.insertSublayer(previewLayer, at: 0)
}

private func configureOverlays() {
    dimLayer.fillRule = .evenOdd
    dimLayer.fillColor = UIColor.black.withAlphaComponent(0.55).cgColor
    view.layer.addSublayer(dimLayer)

    boxLayer.fillColor = UIColor.clear.cgColor
    boxLayer.strokeColor = UIColor.systemTeal.withAlphaComponent(0.9).cgColor
    boxLayer.lineWidth = 2
    view.layer.addSublayer(boxLayer)
}

private func configureModeUI() {
    modeControl.selectedSegmentIndex = 0
    modeControl.addTarget(self, action: #selector(modeChanged), for: .valueChanged)
    modeControl.translatesAutoresizingMaskIntoConstraints = false
    view.addSubview(modeControl)

    hint.text = "Наведіть штрихкод у рамку"
    hint.textColor = .white
    hint.textAlignment = .center
```

```
        hint.font = .systemFont(ofSize: 13, weight: .medium)
        hint.alpha = 0.85
        hint.translatesAutoresizingMaskIntoConstraints = false
        view.addSubview(hint)
        layoutModeUI()
    }

    private func layoutModeUI() {
        NSLayoutConstraint.deactivate(view.constraints.filter {
            ($0.firstItem as? UIView) == modeControl || ($0.firstItem as? UIView) == hint
        })
        NSLayoutConstraint.activate([
            modeControl.centerXAnchor.constraint(equalTo: view.centerXAnchor),
            modeControl.topAnchor.constraint(equalTo:
view.safeAreaLayoutGuide.topAnchor, constant: 8),
            modeControl.widthAnchor.constraint(equalTo:  view.widthAnchor,  multiplier:
0.64),
            hint.centerXAnchor.constraint(equalTo: view.centerXAnchor),
            hint.topAnchor.constraint(equalTo: modeControl.bottomAnchor, constant: 8),
        ])
    }

    @objc private func modeChanged() {
        scanMode = modeControl.selectedSegmentIndex == 0 ? .barcode : .ocr
    }

    private func updateModeUI() {
```

```
        let full = UIBezierPath(rect: view.bounds)
        if scanMode == .barcode {
            let roiViewRect = CGRect(
                x: view.bounds.width * 0.08,
                y: view.bounds.midY - (view.bounds.height * 0.10),
                width: view.bounds.width * 0.84,
                height: view.bounds.height * 0.20
            )
            let cut = UIBezierPath(roundedRect: roiViewRect, cornerRadius: 12)
            full.append(cut)
            dimLayer.path = full.cgPath
            boxLayer.path = cut.cgPath
            dimLayer.isHidden = false
            hint.isHidden = false
        } else {
            dimLayer.isHidden = true
            boxLayer.path = nil
            hint.isHidden = true
        }
    }

    private func configureResultView() {
        view.addSubview(resultView)
        resultView.translatesAutoresizingMaskIntoConstraints = false
        let safe = view.safeAreaLayoutGuide
        NSLayoutConstraint.activate([
            resultView.leadingAnchor.constraint(equalTo: safe.leadingAnchor, constant: 12),
```

```
            resultView.trailingAnchor.constraint(equalTo: safe.trailingAnchor, constant: -
12),
            resultView.bottomAnchor.constraint(equalTo: safe.bottomAnchor, constant: -12),
            resultView.heightAnchor.constraint(greaterThanOrEqualToConstant: 160)
        ])
        resultView.delegate = self
    }

    private func configureVision() {
        textRequest = VNRecognizeTextRequest(completionHandler: handleText)
        textRequest.recognitionLevel = .accurate
        textRequest.usesLanguageCorrection = true
        textRequest.minimumTextHeight = 0.008
        textRequest.recognitionLanguages = ["uk-UA","ru-RU","en-US"]
        barcodeRequest.symbologies = [.EAN13, .EAN8, .UPCE, .Code128, .Code39]
    }

    private func startIfAuthorized() {
        let status = AVCaptureDevice.authorizationStatus(for: .video)
        switch status {
        case .authorized: configureSessionIfNeededAndStart()
        case .notDetermined:
            AVCaptureDevice.requestAccess(for: .video) { granted in
                if granted { self.configureSessionIfNeededAndStart() }
                else { DispatchQueue.main.async { self.showDeniedOnce() } }
            }
        default: showDeniedOnce()
```

```
        }
    }

    private func configureSessionIfNeededAndStart() {
        sessionQueue.async {
            if !self.isSessionConfigured {
                self.session.beginConfiguration()
                self.session.sessionPreset = .hd1280x720
                let discovery = AVCaptureDevice.DiscoverySession(
                    deviceTypes: [.builtInWideAngleCamera, .builtInDualWideCamera, .builtInTripleCamera, .builtInDualCamera],
                    mediaType: .video, position: .back
                )
                let camera = discovery.devices.first
                    ?? AVCaptureDevice.default(.builtInWideAngleCamera, for: .video, position: .back)
                guard let cam = camera,
                      let input = try? AVCaptureDeviceInput(device: cam),
                      self.session.canAddInput(input) else {
                    self.session.commitConfiguration()
                    DispatchQueue.main.async { self.showDeniedOnce() }
                    return
                }
                self.session.addInput(input)
                self.videoOutput.alwaysDiscardsLateVideoFrames = true
                self.videoOutput.videoSettings = [ kCVPixelBufferPixelFormatTypeKey as String: kCVPixelFormatType_32BGRA ]
```

```
            self.videoOutput.setSampleBufferDelegate(self, queue: self.videoQueue)
            if self.session.canAddOutput(self.videoOutput) { self.session.addOutput(self.videoOutput) }
            if let conn = self.videoOutput.connection(with: .video), conn.isVideoOrientationSupported {
                conn.videoOrientation = .portrait
            }
            self.session.commitConfiguration()
            self.isSessionConfigured = true
        }
        if !self.session.isRunning { self.session.startRunning() }
    }
  }

  private func stopSessionAsync() {
    sessionQueue.async {
      if self.session.isRunning { self.session.stopRunning() }
    }
  }

  private func showDeniedOnce() {
    guard !didShowDeniedOnce, view.window != nil else { return }
    didShowDeniedOnce = true
    let a = UIAlertController(title: "Немає доступу до камери", message: "Дозвольте доступ у Налаштуваннях → Приватність → Камера.", preferredStyle: .alert)
    a.addAction(.init(title: "OK", style: .default))
    present(a, animated: true)
```

```
    }

    private func processFrame(_ pixelBuffer: CVPixelBuffer) {
        guard scanState != .locked else { return }
        let now = CACurrentMediaTime()
        guard !isProcessing, now - lastAnalysisTime > analysisInterval else { return }
        isProcessing = true
        lastAnalysisTime = now

        let handler = VNImageRequestHandler(cvPixelBuffer: pixelBuffer, orientation: .right, options: [:])
        visionQueue.async {
            do {
                switch self.scanMode {
                case .barcode:
                    let roiViewRect = CGRect(
                        x: self.view.bounds.width * 0.08,
                        y: self.view.bounds.midY - (self.view.bounds.height * 0.10),
                        width: self.view.bounds.width * 0.84,
                        height: self.view.bounds.height * 0.20
                    )
                    let normalized = self.previewLayer.metadataOutputRectConverted(fromLayerRect: roiViewRect)
                    self.barcodeRequest.regionOfInterest = normalized
                    try handler.perform([self.barcodeRequest])
                case .ocr:
                    try handler.perform([self.textRequest])
```

```
            }
        } catch {
            DispatchQueue.main.async { self.isProcessing = false }
        }
    }
}

private func handleText(request: VNRequest, error: Error?) {
    defer { DispatchQueue.main.async { self.isProcessing = false } }
    guard scanState != .locked else { return }
    guard let observations = request.results as? [VNRecognizedTextObservation],
!observations.isEmpty else {
        DispatchQueue.main.async { self.resultView.showAnalyzing() }
        return
    }
    let lines: [String] = observations.compactMap { $0.topCandidates(1).first?.string }
    for l in lines {
        let s = l.trimmingCharacters(in: .whitespacesAndNewlines)
        if s.count >= 3 { accumulatedLines.insert(s) }
    }
    let fullRaw = accumulatedLines.prefix(250).joined(separator: "\n")
    let extr = NutritionOrIngredientsExtractor.extract(from: fullRaw)
    let textToAnalyze = extr.text
    let assumedPer100g = extr.per100g
    let normalized = normalize(textToAnalyze)
    let tokenHits = findAllergenHits(in: normalized, lexicon: allergenLexicon)
    let catCounts = AllergenCatalog.groupHitsByCategory(tokenHits)
```

```
        let nutrients = NutrientParserV2.parse(from: textToAnalyze, assumePer100g: assumedPer100g)
        let score: Int = {
            guard let n = nutrients else { return tokenHits.isEmpty ? 70 : 55 }
            var s = 70
            if let p = n.protein, p >= 10 { s += 10 }
            if let f = n.fiber,  f >= 3  { s += 5  }
            if let su = n.sugars, su > 8 { s -= 10 }
            if let sl = n.salt,   sl > 1 { s -= 10 }
            if !tokenHits.isEmpty { s -= 10 }
            return max(0, min(100, s))
        }()
        let fieldsFound: Int = {
            guard let n = nutrients else { return 0 }
            return [n.protein, n.fat, n.carbs, n.fiber, n.sugars, n.salt].compactMap { $0 }.count
        }()
        if fieldsFound >= 3 {
            let payload = ResultView.ResultPayload(
                nutrients: nutrients,
                tokenHits: tokenHits,
                categoryCounts: catCounts,
                score: score,
                comment: makeComment(nutrients: nutrients, tokenHits: tokenHits)
            )
            DispatchQueue.main.async {
                self.lastBarcode = nil
                self.lastProductName = nil
```

```
            self.lastProductImageURL = nil
            self.lock(with: payload)
        }
    } else {
        DispatchQueue.main.async { self.resultView.showAnalyzing() }
    }
}

private func handleBarcodes(request: VNRequest, error: Error?) {
    guard scanState != .locked else { return }
    guard let results = request.results as? [VNBarcodeObservation], !results.isEmpty
else {
        DispatchQueue.main.async { self.boxLayer.path = nil; self.isProcessing = false }
        return
    }
    let best = results.max(by: { $0.confidence < $1.confidence })
    guard let code = best?.payloadStringValue, !alreadyFetchingBarcode else {
        DispatchQueue.main.async { self.isProcessing = false }
        return
    }
    if let box = best?.boundingBox {
        let rect = previewLayer.layerRectConverted(fromMetadataOutputRect: box)
        let p = UIBezierPath(roundedRect: rect, cornerRadius: 6)
        DispatchQueue.main.async { self.boxLayer.path = p.cgPath }
    }
    alreadyFetchingBarcode = true
    BarcodeService.fetchProduct(by: code) { name, nutrients, ingredients, imgURL in
```

```
        self.alreadyFetchingBarcode = false
        self.isProcessing = false
        guard self.scanState != .locked else { return }
        if nutrients == nil, (ingredients ?? "").trimmingCharacters(in: .whitespacesAndNewlines).isEmpty { return }
        let normalized = self.normalize(ingredients ?? "")
        let tokenHits = self.findAllergenHits(in: normalized, lexicon: self.allergenLexicon)
        let catCounts = AllergenCatalog.groupHitsByCategory(tokenHits)
        let score: Int = {
            guard let n = nutrients else { return tokenHits.isEmpty ? 70 : 50 }
            var s = 70
            if let p = n.protein, p >= 10 { s += 10 }
            if let f = n.fiber,  f >= 3  { s += 5  }
            if let su = n.sugars, su > 8 { s -= 10 }
            if let sl = n.salt,   sl > 1 { s -= 10 }
            if !tokenHits.isEmpty { s -= 10 }
            return max(0, min(100, s))
        }()
        let payload = ResultView.ResultPayload(
            nutrients: nutrients,
            tokenHits: tokenHits,
            categoryCounts: catCounts,
            score: score,
            comment: self.makeComment(nutrients: nutrients, tokenHits: tokenHits)
        )
        print("[BARCODE] found code:", code, "confidence:", best?.confidence ?? -1)
```

```
            self.lastBarcode = code
            self.lastProductName = name
            self.lastProductImageURL = imgURL
            self.lock(with: payload)
            print("[BARCODE] OFF:", "name=\(name ?? "nil")",
                  "hasNutrients=\(nutrients != nil)",
                  "hasIngredients=\(!(ingredients ?? "").isEmpty))")
        }
    }

    private func lock(with payload: ResultView.ResultPayload) {
        guard scanState != .locked else { return }
        stopSessionAsync()
        lastPayload = payload
        scanState = .locked
        resultView.lock(with: payload)
    }

    private func resumeScanning() {
        lastPayload = nil
        lastBarcode = nil
        lastProductName = nil
        lastProductImageURL = nil
        accumulatedLines.removeAll()
        scanState = .analyzing
        boxLayer.path = nil
        resultView.showAnalyzing()
```

```
        configureSessionIfNeededAndStart()
    }

    private func makeComment(nutrients: Nutrients?, tokenHits: [String:Int]) -> String {
        if let n = nutrients {
            var pros: [String] = []; var cons: [String] = []
            if let p = n.protein, p >= 10 { pros.append("високий білок") }
            if let f = n.fiber,   f >= 3  { pros.append("клітковина") }
            if let s = n.sugars,  s > 8   { cons.append("багато цукрів") }
            if let sl = n.salt,   sl > 1  { cons.append("багато солі") }
            var base = n.per100g ? "Оцінка за 100 г: " : "Оцінка за порцію: "
            if pros.isEmpty && cons.isEmpty { base += "збалансовано" }
            else {
                if !pros.isEmpty { base += "плюси — " + pros.joined(separator: ", ") + ". " }
                if !cons.isEmpty { base += "мінуси — " + cons.joined(separator: ", ") + "." }
            }
            if !tokenHits.isEmpty { base += " Увага: знайдено потенційні алергени." }
            return base
        } else {
            return tokenHits.isEmpty ? "Дані за штрихкодом/сканом обмежені; нутрієнти не знайдено." : "Дані обмежені; знайдено потенційні алергени."
        }
    }

    private func normalize(_ s: String) -> String {
        var t = s.lowercased()
        t = t.folding(options: .diacriticInsensitive, locale: .current)
```

```
            .replacingOccurrences(of: "\r", with: "")
            .replacingOccurrences(of: "\\s+", with: " ", options: .regularExpression)
        t = t.replacingOccurrences(of: "0", with: "o")
            .replacingOccurrences(of: "1", with: "l")
            .replacingOccurrences(of: "5", with: "s")
        return t
    }

    private func findAllergenHits(in text: String, lexicon: [String]) -> [String: Int] {
        var counters: [String: Int] = [:]
        for token in lexicon {
            let pattern = "\\b\(NSRegularExpression.escapedPattern(for: token))\\w*\\b"
            if let regex = try? NSRegularExpression(pattern: pattern, options: [.caseInsensitive]) {
                let n = regex.numberOfMatches(in: text, options: [], range: NSRange(location: 0, length: (text as NSString).length))
                if n > 0 { counters[token] = n }
            }
        }
        return counters
    }
}

extension ScannerViewController: AVCaptureVideoDataOutputSampleBufferDelegate {
    func captureOutput(_ output: AVCaptureOutput, didOutput sampleBuffer: CMSampleBuffer, from connection: AVCaptureConnection) {
        guard scanState != .locked else { return }
```

```
        guard let pb = CMSampleBufferGetImageBuffer(sampleBuffer) else { return }
        processFrame(pb)
    }
}

extension ScannerViewController: ResultViewDelegate {
    func resultViewDidTapNewScan(_ view: ResultView) { resumeScanning() }
    func resultViewDidTapSave(_ view: ResultView) {
        guard let payload = lastPayload else { return }
        let vc = SaveProductViewController(
            defaultTitle: lastProductName ?? "",
            defaultBarcode: lastBarcode,
            defaultNutrients: payload.nutrients,
            defaultAllergenHits: payload.tokenHits,
            defaultImageURL: lastProductImageURL
        )
        vc.onSaved = { _ in
            if self.view.window != nil {
                let a = UIAlertController(title: "Збережено", message: "Продукт додано до списку.", preferredStyle: .alert)
                a.addAction(.init(title: "OK", style: .default))
                self.present(a, animated: true)
            }
        }
        navigationController?.pushViewController(vc, animated: true)
    }
}
```

HomeViewController.swift

```
import UIKit

final class HomeViewController: UIViewController, UITableViewDataSource,
UITableViewDelegate {

    private let scroll = UIScrollView()
    private let content = UIStackView()

    private let macrosCard = UIStackView()
    private let proteinLabel = UILabel()
    private let fatLabel = UILabel()
    private let carbLabel = UILabel()

    private let explLabel = UILabel()
    private let refreshButton = UIButton(type: .system)

    private let table = UITableView(frame: .zero, style: .plain)
    private var ranked: [(SavedProduct, Double)] = []

    private let recommender = Recommender()

    override func viewDidLoad() {
        super.viewDidLoad()
        title = "Головна"
```

```
        addBackground()
        setupUI()
        primeDefaultsIfNeeded()
    }

    private func addBackground() {
        let bg = UIImageView(frame: view.bounds)
        bg.image = UIImage(named: "bg")
        bg.contentMode = .scaleAspectFill
        view.addSubview(bg)
        view.sendSubviewToBack(bg)
    }

    override func viewWillAppear(_ animated: Bool) {
        super.viewWillAppear(animated)
        reloadData()
    }

    private func setupUI() {
        scroll.translatesAutoresizingMaskIntoConstraints = false
        view.addSubview(scroll)
        content.axis = .vertical
        content.spacing = 16
        content.translatesAutoresizingMaskIntoConstraints = false
        scroll.addSubview(content)

        NSLayoutConstraint.activate([
```

```
            scroll.leadingAnchor.constraint(equalTo:
view.safeAreaLayoutGuide.leadingAnchor),
            scroll.trailingAnchor.constraint(equalTo:
view.safeAreaLayoutGuide.trailingAnchor),
            scroll.topAnchor.constraint(equalTo: view.safeAreaLayoutGuide.topAnchor),
            scroll.bottomAnchor.constraint(equalTo: view.bottomAnchor),

            content.leadingAnchor.constraint(equalTo:
scroll.contentLayoutGuide.leadingAnchor, constant: 16),
            content.trailingAnchor.constraint(equalTo:
scroll.contentLayoutGuide.trailingAnchor, constant: -16),
            content.topAnchor.constraint(equalTo: scroll.contentLayoutGuide.topAnchor,
constant: 16),
            content.bottomAnchor.constraint(equalTo:
scroll.contentLayoutGuide.bottomAnchor, constant: -24),
            content.widthAnchor.constraint(equalTo:
scroll.frameLayoutGuide.widthAnchor, constant: -32)
        ])

        macrosCard.axis = .horizontal
        macrosCard.distribution = .fillEqually
        macrosCard.spacing = 12
        [metricBox(title: "Білки", label: proteinLabel),
         metricBox(title: "Жири", label: fatLabel),
         metricBox(title: "Вуглеводи", label: carbLabel)
        ].forEach { macrosCard.addArrangedSubview($0) }
```

```
        explLabel.text = "Нижче — персональні рекомендації. Порядок визначає LinUCB: прогноз корисності + бонус за невпевненість. Обирайте — модель навчається."
        explLabel.font = .mont(size: 14, weight: 400)
        explLabel.textColor = .darkGreen.withAlphaComponent(0.8)
        explLabel.numberOfLines = 0

        refreshButton.setTitle("Оновити рекомендації", for: .normal)
        refreshButton.titleLabel?.font = .mont(size: 16, weight: 600)
        refreshButton.setTitleColor(.darkGreen, for: .normal)
        refreshButton.backgroundColor = .customOrange
        refreshButton.layer.cornerRadius = 14
        refreshButton.heightAnchor.constraint(equalToConstant: 44).isActive = true
        refreshButton.addTarget(self, action: #selector(forceRefresh), for: .touchUpInside)

        table.backgroundColor = .clear
        table.separatorStyle = .none
        table.dataSource = self
        table.delegate = self
        table.register(HomeRecCell.self, forCellReuseIdentifier: "rec")
        table.heightAnchor.constraint(equalToConstant: 640).isActive = true

        [macrosCard, explLabel, refreshButton, table].forEach { content.addArrangedSubview($0) }
    }

    private func metricBox(title: String, label: UILabel) -> UIView {
```

```
    let box = UIView()
    box.backgroundColor = .viewBG
    box.layer.cornerRadius = 16

    let t = UILabel()
    t.text = title
    t.font = .mont(size: 13, weight: 500)
    t.textColor = .darkGreen.withAlphaComponent(0.75)

    label.font = .mont(size: 28, weight: 600)
    label.textColor = .darkGreen
    label.textAlignment = .center

    let stack = UIStackView(arrangedSubviews: [t, label])
    stack.axis = .vertical
    stack.alignment = .center
    stack.spacing = 6
    stack.translatesAutoresizingMaskIntoConstraints = false
    box.addSubview(stack)

    NSLayoutConstraint.activate([
        stack.centerXAnchor.constraint(equalTo: box.centerXAnchor),
        stack.centerYAnchor.constraint(equalTo: box.centerYAnchor),
        box.heightAnchor.constraint(equalToConstant: 110)
    ])
    return box
}
```

```
    @objc private func forceRefresh() { reloadData() }

    private func reloadData() {
        let prof = StorageService.shared.loadProfile()
        proteinLabel.text = "\(prof?.dailyProtein ?? 0) г"
        fatLabel.text     = "\(prof?.dailyFat ?? 0) г"
        carbLabel.text    = "\(prof?.dailyCarb ?? 0) г"

        let items = StorageService.shared.loadProducts()
        if let p = prof {
            ranked = recommender.rank(profile: p, products: items)
        } else {
            ranked = items.map { ($0, 0) }
        }
        table.reloadData()
    }

    func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) ->
Int { ranked.count }

    func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) ->
UITableViewCell {
        let (item, score) = ranked[indexPath.row]
        let cell = tableView.dequeueReusableCell(withIdentifier: "rec", for: indexPath) as!
HomeRecCell
        cell.apply(product: item, score: score)
```

```
        return cell
    }

    func tableView(_ tableView: UITableView, didSelectRowAt indexPath: IndexPath) {
        tableView.deselectRow(at: indexPath, animated: true)

navigationController?.pushViewController(SavedProductDetailViewController(product:
ranked[indexPath.row].0), animated: true)
    }

    private func primeDefaultsIfNeeded() {
        guard let profile = StorageService.shared.loadProfile() else { return }
        StarterPrimer.primeIfNeeded(
            profile: profile,
            barcodesByGoal: [
                .deficit:
["4820006011147","4820224233765","4823065722320","4820156150369"],
                .balance:
["4820006011147","4820224233765","4823065722320","4820156150369"],
                .surplus:
["4820006011147","4820224233765","4823065722320","4820156150369"]
            ],
            completion: { [weak self] in self?.reloadData() }
        )
    }
}
```

```
final class HomeRecCell: UITableViewCell {
    private let container = UIView()
    private let thumb = UIImageView()
    private let titleL = UILabel()
    private let scoreBadge = UILabel()

    override init(style: UITableViewCell.CellStyle, reuseIdentifier: String?) {
        super.init(style: style, reuseIdentifier: reuseIdentifier)

        backgroundColor = .clear
        contentView.backgroundColor = .clear
        selectionStyle = .none

        container.backgroundColor = .viewBG
        container.layer.cornerRadius = 14
        container.layer.shadowColor = UIColor.black.cgColor
        container.layer.shadowOpacity = 0.12
        container.layer.shadowRadius = 10
        container.layer.shadowOffset = .init(width: 0, height: 6)
        container.layer.masksToBounds = false
        container.translatesAutoresizingMaskIntoConstraints = false
        contentView.addSubview(container)

        thumb.contentMode = .scaleAspectFill
        thumb.clipsToBounds = true
        thumb.layer.cornerRadius = 10
        thumb.backgroundColor = .secondarySystemBackground
```

```
thumb.translatesAutoresizingMaskIntoConstraints = false
thumb.widthAnchor.constraint(equalToConstant: 60).isActive = true
thumb.heightAnchor.constraint(equalToConstant: 60).isActive = true

titleL.font = .mont(size: 16, weight: 600)
titleL.textColor = .darkGreen
titleL.numberOfLines = 2

scoreBadge.font = .mont(size: 13, weight: 600)
scoreBadge.textAlignment = .center
scoreBadge.textColor = .white
scoreBadge.backgroundColor = .systemPurple
scoreBadge.layer.cornerRadius = 12
scoreBadge.layer.masksToBounds = true
scoreBadge.widthAnchor.constraint(equalToConstant: 54).isActive = true
scoreBadge.heightAnchor.constraint(equalToConstant: 24).isActive = true

let v = UIStackView(arrangedSubviews: [titleL])
v.axis = .vertical
v.spacing = 2

let h = UIStackView(arrangedSubviews: [thumb, v, scoreBadge])
h.axis = .horizontal
h.alignment = .center
h.spacing = 12
h.translatesAutoresizingMaskIntoConstraints = false
```

```
        container.addSubview(h)
        NSLayoutConstraint.activate([
            container.leadingAnchor.constraint(equalTo: contentView.leadingAnchor,
constant: 16),
            container.trailingAnchor.constraint(equalTo: contentView.trailingAnchor,
constant: -16),
            container.topAnchor.constraint(equalTo: contentView.topAnchor, constant: 6),
            container.bottomAnchor.constraint(equalTo: contentView.bottomAnchor,
constant: -6),

            h.leadingAnchor.constraint(equalTo: container.leadingAnchor, constant: 12),
            h.trailingAnchor.constraint(equalTo: container.trailingAnchor, constant: -12),
            h.topAnchor.constraint(equalTo: container.topAnchor, constant: 10),
            h.bottomAnchor.constraint(equalTo: container.bottomAnchor, constant: -10),
        ])
    }

    required init?(coder: NSCoder) { fatalError("init(coder:) has not been implemented")
}

    func apply(product: SavedProduct, score: Double) {
        titleL.text = product.userTitle
        scoreBadge.text = String(format: "%.1f", score)
        if let fn = product.imageFilename, let img =
StorageService.shared.loadImage(named: fn) {
            thumb.image = img
            thumb.contentMode = .scaleAspectFill
```

```
            thumb.backgroundColor = .secondarySystemBackground
        } else {
            thumb.image = UIImage(systemName: "photo")
            thumb.tintColor = .systemGray3
            thumb.contentMode = .scaleAspectFit
            thumb.backgroundColor = .clear
        }
    }
}
```

LinUCB.swift

```
import Foundation
import CoreGraphics

private struct LA {
    static func dot(_ a: [Double], _ b: [Double]) -> Double {
        precondition(a.count == b.count)
        var s = 0.0
        for i in 0..<a.count { s += a[i]*b[i] }
        return s
    }
    static func outer(_ x: [Double]) -> [[Double]] {
        let d = x.count
        var M = Array(repeating: Array(repeating: 0.0, count: d), count: d)
        for i in 0..<d { for j in 0..<d { M[i][j] = x[i]*x[j] } }
        return M
    }
    static func matVec(_ A: [[Double]], _ x: [Double]) -> [Double] {
```

```
        precondition(!A.isEmpty && A[0].count == x.count)
        var y = Array(repeating: 0.0, count: A.count)
        for i in 0..<A.count { y[i] = dot(A[i], x) }
        return y
    }
    static func rankOneInverseUpdate(Ainv: inout [[Double]], with x: [Double]) {
        let Ainvx = matVec(Ainv, x)
        let denom = 1.0 + dot(x, Ainvx)
        if denom <= 1e-9 { return }
        let d = x.count
        for i in 0..<d {
            for j in 0..<d {
                Ainv[i][j] -= (Ainvx[i] * Ainvx[j]) / denom
            }
        }
    }
}

private struct ArmState: Codable {
    var Ainv: [[Double]]
    var b: [Double]
    var pulls: Int
}

final class LinUCB: Codable {
    typealias ArmID = String
```

```
private let d: Int
private let alpha: Double
private var arms: [ArmID: ArmState] = [:]

init(dimension: Int, alpha: Double = 0.8) {
    self.d = dimension
    self.alpha = alpha
}

private func ensureArm(_ id: ArmID) {
    if arms[id] != nil { return }
    let I = (0..<d).map { i in (0..<d).map { j in i==j ? 1.0 : 0.0 } }
    arms[id] = ArmState(Ainv: I, b: Array(repeating: 0.0, count: d), pulls: 0)
}

private func theta(for id: ArmID) -> [Double] {
    guard let st = arms[id] else { return Array(repeating: 0, count: d) }
    return LA.matVec(st.Ainv, st.b)
}

func score(arm id: ArmID, context x: [Double]) -> Double {
    precondition(x.count == d)
    ensureArm(id)
    guard let st = arms[id] else { return 0 }
    let th = LA.matVec(st.Ainv, st.b)
    let mean = LA.dot(th, x)
    let Ax = LA.matVec(st.Ainv, x)
```

```
        let ucb = sqrt(max(0, LA.dot(x, Ax)))
        return mean + alpha * ucb
    }

    func selectBest(arms candidates: [(ArmID, [Double])]) -> (ArmID?, [(ArmID,
Double)]) {
        guard !candidates.isEmpty else { return (nil, []) }
        var scored: [(ArmID, Double)] = []
        scored.reserveCapacity(candidates.count)
        for (id, x) in candidates { scored.append((id, score(arm: id, context: x))) }
        scored.sort { $0.1 > $1.1 }
        return (scored.first?.0, scored)
    }

    func update(arm id: ArmID, context x: [Double], reward r: Double) {
        precondition(x.count == d)
        ensureArm(id)
        guard var st = arms[id] else { return }
        LA.rankOneInverseUpdate(Ainv: &st.Ainv, with: x)
        for i in 0..<d { st.b[i] += r * x[i] }
        st.pulls += 1
        arms[id] = st
    }

    func save(to key: String = "linucb_state") {
        if let data = try? JSONEncoder().encode(self) {
            UserDefaults.standard.set(data, forKey: key)
```

```
        }
    }
    static func load(from key: String = "linucb_state") -> LinUCB? {
        guard let data = UserDefaults.standard.data(forKey: key),
              let model = try? JSONDecoder().decode(LinUCB.self, from: data) else { return
nil }
        return model
    }
}

struct LinUCBFeatureBuilder {
    static let dimension = 12

    static func features(profile: UserProfile, product: SavedProduct) -> [Double] {
        let wt = Double(max(40, min(140, profile.weightKg)))
        let h  = Double(max(140, min(210, profile.heightCm)))
        let age = Double(max(14, min(80, profile.age)))
        let act = activityFactor(profile.activity)
        let goal = goalSign(profile.goal)
        let userAll = profile.allergies.map { $0.lowercased() }
        let n = product.nutrients
        let p = clamp01((n?.protein ?? 0) / 40.0)
        let f = clamp01((n?.fat ?? 0)     / 60.0)
        let c = clamp01((n?.carbs ?? 0)   / 100.0)
        let fi = clamp01((n?.fiber ?? 0)  / 30.0)
        let su = clamp01((n?.sugars ?? 0) / 40.0)
        let sa = clamp01((n?.salt ?? 0)   / 6.0)
```

```
    let allergenHit: Double = {
        guard let hits = product.allergenHits, !hits.isEmpty else { return 0 }
        let joined = hits.keys.joined(separator: " ").lowercased()
        return userAll.contains(where: { joined.contains($0) }) ? 1 : 0
    }()
    let pseudoScore = clamp01((p + fi) - 0.5 * (su + sa))
    let wN = (wt - 40) / 100.0
    let hN = (h - 140) / 70.0
    let aN = (age - 14) / 66.0
    let actN = (act - 1.2) / 0.7
    var x: [Double] = [
        wN, hN, aN, actN,
        goal,
        p, f, c, fi, su, sa,
        pseudoScore
    ]
    for i in 0..<x.count { if !x[i].isFinite { x[i] = 0 } }
    return x
}

private static func clamp01(_ v: Double) -> Double { max(0, min(1, v)) }
private static func activityFactor(_ a: UserProfile.ActivityLevel) -> Double {
    switch a {
    case .sedentary: return 1.2
    case .light:     return 1.375
    case .moderate:  return 1.55
    case .high:      return 1.725
```

```
        }
    }
    private static func goalSign(_ g: UserProfile.Goal) -> Double {
        switch g {
        case .deficit: return -1
        case .balance: return 0
        case .surplus: return 1
        }
    }
}

final class Recommender {
    private let model: LinUCB

    init(alpha: Double = 0.8) {
        if let saved = LinUCB.load() { self.model = saved }
        else { self.model = LinUCB(dimension: LinUCBFeatureBuilder.dimension, alpha:
alpha) }
    }

    func rank(profile: UserProfile, products: [SavedProduct]) -> [(SavedProduct,
Double)] {
        guard !products.isEmpty else { return [] }
        var scored: [(SavedProduct, Double)] = []
        scored.reserveCapacity(products.count)
        for p in products {
            let id = p.barcode ?? p.id.uuidString
```

```
            let x  = LinUCBFeatureBuilder.features(profile: profile, product: p)
            let s  = model.score(arm: id, context: x)
            scored.append((p, s))
        }
        scored.sort { $0.1 > $1.1 }
        return scored
    }

    func feedback(profile: UserProfile, product: SavedProduct, reward r: Double) {
        let id = product.barcode ?? product.id.uuidString
        let x  = LinUCBFeatureBuilder.features(profile: profile, product: product)
        model.update(arm: id, context: x, reward: max(0, min(1, r)))
        model.save()
    }
}
```