

FILIA
UNIwersytetu w Białymstoku
w WILNIE

INFORMATYKA

Modestas Januškevič

Nr albumu: 88360

PROTOTYP APLIKACJI MOBILNEJ DO
MONITOROWANIA ILOŚCI PRZYJMOWANYCH
KALORII I DO REKOMENDACJI W ZAKRESIE
POSIŁKÓW

Praca dyplomowa napisana

pod kierunkiem

prof. dr hab. inż. Mieczysława Muraszkiewicza

Wilno, lipiec 2026

STRESZCZENIE

W pracy opisano projekt i wykonanie aplikacji mobilnej *NutritionTracker* przeznaczonej do rejestrowania posiłków oraz kontrolowania kalorii i makroskładników. Program powstał dla systemu Android z użyciem React Native i Expo. Jednym z głównych założeń projektu było ograniczenie zależności od serwera zewnętrznego. Profil, historia posiłków i pamięć zeskanowanych produktów są zapisywane w lokalnej bazie SQLite, dlatego podstawowe funkcje dziennika pozostają dostępne również bez połączenia z Internetem.

Na podstawie parametrów użytkownika aplikacja wyznacza początkowy cel energetyczny i podział makroskładników. Osoba korzystająca z programu może zamiast tego wpisać własne wartości. Produkty można dodawać ręcznie, wyszukiwać w katalogu lokalnym albo pobierać po zeskanowaniu kodu kreskowego. Przed zapisem użytkownik sprawdza gramaturę oraz wartości na 100 g, a każdy produkt jest zapisywany osobno i może być później edytowany. Propozycje posiłków są porządkowane według celu sylwetkowego, pory dnia oraz wartości pozostałych do realizacji w danym dniu. Dane produktów oznakowanych pochodzą z Open Food Facts i są zapamiętywane na urządzeniu.

Dodatkową funkcją jest lokalny asystent korzystający z modelu Qwen3 1.7B lub Qwen3 4B w formacie GGUF. Wybrany model pobiera się po instalacji, a obliczenia są wykonywane bezpośrednio na telefonie przez bibliotekę `llama.rn`. Aplikacja ocenia możliwości urządzenia, sprawdza wolne miejsce, pokazuje postęp i pozwala zatrzymać generowanie. Model nie oblicza celów żywieniowych ani wartości produktów. Te zadania pozostają po stronie zwykłego, powtarzalnego kodu.

Rezultatem pracy jest samodzielna aplikacja mobilna, która łączy dziennik żywieniowy, lokalny katalog produktów, skanowanie kodów kreskowych, rekomendacje posiłków oraz opcjonalnego asystenta działającego na urządzeniu użytkownika.

Słowa kluczowe: Android, monitorowanie kalorii, skanowanie kodów kreskowych, lokalny model językowy, prywatność.

ABSTRACT

This thesis describes the design and implementation of *NutritionTracker*, an Android application for logging meals and tracking calories and macronutrients. It was built with React Native and Expo. One of the main design assumptions was to reduce dependence on an external server. The user profile, meal history, and cached barcode products are stored in a local SQLite database, so the core diary features remain available offline.

The application estimates an initial energy target from body measurements, activity, and physique goal, while still allowing the user to enter custom targets. Products can be entered manually, selected from a local catalogue, or added through barcode scanning. Before saving, the user can review the serving size and nutrition per 100 g. Every product is stored separately and can later be edited. Meal ideas are ranked using the user's goal, the selected meal type, food exclusions, and the nutrition still remaining for the day. Packaged food data are retrieved from Open Food Facts and cached on the device.

The project also includes an optional on-device Qwen3 1.7B or Qwen3 4B model in GGUF format. The selected model is downloaded after installation and run directly on the phone through `llama.rn`. The application checks device capability and free storage, streams responses, and allows generation to be stopped. Nutrition targets and product values are calculated by deterministic code rather than delegated to the language model.

The result of the work is a standalone mobile application that combines a food diary, a local product catalogue, barcode scanning, meal recommendations, and an optional assistant running on the user's device.

Keywords: Android, calorie tracking, barcode scanning, local language model, privacy.

SPIS TREŚCI

Streszczenie	1
Abstract	2
1 Wstęp	5
1.1 Znaczenie codziennego monitorowania żywienia	5
1.2 Prywatność w czasach powszechnego korzystania z AI	5
1.3 Cel, zadanie i struktura pracy	6
2 Podstawy monitorowania żywienia	8
2.1 Bilans energetyczny	8
2.2 Podstawowa przemiana materii	8
2.3 Całkowite zapotrzebowanie energetyczne	9
2.4 Makroskładniki	9
2.5 Dokładność danych	10
2.6 Codzienne trudności i istniejące rozwiązania	10
2.6.1 Dlaczego zapisywanie posiłków bywa trudne	10
2.6.2 Klasy istniejących rozwiązań	11
2.6.3 Porównanie z MyFitnessPal i Cronometer	11
2.6.4 Uzasadnienie podejścia offline-first	13
3 Realizacja aplikacji NutritionTracker	15
3.1 Wymagania funkcjonalne i нефункционалне	16
3.1.1 Wymagania funkcjonalne	16
3.1.2 Wymagania нефункционалне	19
3.1.3 Powiązanie wymagań z celami	21
3.1.4 Przypadki użycia	21
3.2 Architektura i diagramy systemu	23
3.2.1 Architektura logiczna	23
3.2.2 Przepływ danych	24
3.2.3 Model danych	25
3.2.4 Nawigacja	26
3.2.5 Projekt interfejsu	27
3.2.6 Struktura modułów projektu	29
3.3 Narzędzia i biblioteki	30
3.3.1 Android i Android Studio	30
3.3.2 React Native	31
3.3.3 Expo	31
3.3.4 Zustand i React Navigation	32
3.3.5 SQLite	32
3.3.6 Open Food Facts	32
3.3.7 Qwen3, GGUF oraz silnik llama.rn	33
3.4 Interfejs i implementacja aplikacji	35
3.4.1 Inicjalizacja	35
3.4.2 Ekran dzienny	35
3.4.3 Formularz dodawania posiłku	35

3.4.4	Skaner	36
3.4.5	Kalendarz	36
3.4.6	Profil	36
3.4.7	Konfiguracja AI	37
3.4.8	Rozmowa	37
3.4.9	Budowanie pakietu APK	37
3.5	Algorytmy i procesy działania aplikacji	38
3.5.1	Wyznaczanie celu energetycznego i tryb pracy	38
3.5.2	Rejestrowanie i przetwarzanie żywności	38
3.5.3	Lokalny asystent AI	42
4	Weryfikacja i ocena rozwiązania	46
4.1	Realizacja założeń prywatności	46
4.1.1	Minimalizacja danych	46
4.1.2	Lokalizacja i przepływ danych	46
4.1.3	Uprawnienia	48
4.1.4	Bezpieczeństwo lokalnych danych	48
4.1.5	Kontrola użytkownika i cykl życia danych	49
4.1.6	Granice odpowiedzialności systemu	49
4.2	Testowanie systemu	49
4.2.1	Metodyka	49
4.2.2	Testy funkcji obliczeniowych	50
4.2.3	Testy scenariuszy użytkownika	52
4.2.4	Ocena zgodności z wymaganiami	52
4.2.5	Test na urządzeniu Samsung Galaxy A23	53
4.2.6	Walidacja techniczna	53
4.3	Ocena rozwiązania	55
	Zakończenie	57
	Spis tabel	58
	Spis rysunków	59
	Bibliografia	60
	Słownik pojęć	63

1. WSTĘP

1.1. Znaczenie codziennego monitorowania żywienia

Monitorowanie sposobu odżywiania może być przydatne osobom realizującym różne cele: utrzymanie masy ciała, dopasowanie diety do stylu życia, stopniową zmianę masy lub budowanie mięśni. Jego zadaniem jest uporządkowanie informacji o posiłkach i ułatwienie porównania codziennych wyborów z celem określonym przez samego użytkownika.

W praktyce regularne prowadzenie takich zapisów wymaga sprawdzania wartości odżywczych, przeliczania ich na wielkość porcji oraz sumowania produktów w obrębie dnia. Dane na etykietach są podawane w różnych formach, a posiłek często składa się z kilku składników. Dlatego potrzebne jest proste narzędzie, które ogranicza liczbę ręcznych obliczeń i przedstawia wynik w zrozumiały sposób.

1.2. Prywatność w czasach powszechnego korzystania z AI

Systemy sztucznej inteligencji stają się dla wielu osób codziennym narzędziem. Użytkownicy pytają je nie tylko o fakty, ale również o własne problemy, ciało, zdrowie, sposób odżywiania, relacje i plany na przyszłość. Rozmowa z modelem jest prosta i przypomina rozmowę z człowiekiem. Ta naturalność może sprawić, że łatwo zapomnieć, iż tekst wpisany do usługi chmurowej opuszcza urządzenie i jest przetwarzany przez cudzą infrastrukturę.

Problem nie polega na tym, że każda usługa internetowa niewłaściwie wykorzystuje dane. Polega raczej na tym, że użytkownik często nie wie, gdzie dokładnie trafia jego wiadomość, jak długo jest przechowywana i kto może mieć do niej dostęp. Wygoda rozmowy zachęca do szczerości, natomiast świadomość technicznego przepływu danych pozostaje w tle. Im bardziej osobisty staje się asystent, tym większe znaczenie ma pytanie, czy prywatna rozmowa nadal jest prywatna.

W aplikacji żywieniowej ten problem jest szczególnie widoczny. Pojedyncza informacja o zjedzonym produkcie może wydawać się nieistotna. Cała historia, połączona z masą ciała, celem sylwetkowym, alergiami i pytaniami kierowanymi do AI, tworzy jednak dokładny obraz codziennych zachowań człowieka. Są to dane, którymi użytkownik może chcieć podzielić się z trenerem lub lekarzem, ale niekoniecznie z operatorem każdej używanej aplikacji.

Prywatność nie oznacza posiadania czegoś do ukrycia. Oznacza możliwość decydowania, które informacje pozostają osobiste, komu zostaną przekazane i w jakim celu. Człowiek zachowuje się swobodniej, gdy wie, że jego pytania nie tworzą profilu na odległym serwerze. W przypadku narzędzia dotyczącego ciała i odżywiania takie poczucie kontroli jest częścią jakości aplikacji, a nie jedynie dodatkową opcją techniczną.

Z tego powodu w projekcie *NutritionTracker* przyjęto lokalność jako główne założenie. Profil i historia posiłków pozostają w telefonie, a model językowy może działać bez wysyłania

rozmowy do dostawcy chmurowej AI. Rozwiązanie lokalne może być wolniejsze, zajmować więcej pamięci i oferować niższą jakość odpowiedzi niż największe modele internetowe. W zamian użytkownik otrzymuje coś innego: możliwość korzystania z asystenta bez oddawania treści rozmowy operatorowi usługi.

Projekt nie zakłada, że technologia lokalna rozwiązuje wszystkie problemy bezpieczeństwa. Telefon może zostać zgubiony, a dane lokalne wymagają ochrony. Najważniejsza jest jednak zmiana punktu wyjścia. Dane nie są wysyłane domyślnie, a ich pozostawienie na urządzeniu nie wymaga dodatkowej decyzji użytkownika. To aplikacja ma uzasadnić potrzebę transmisji, zamiast zakładać, że wszystko może zostać przesłane do chmury.

Adiustacja językowa tekstu została przeprowadzona z wykorzystaniem narzędzia ChatGPT 5.5 firmy OpenAI.

1.3. Cel, zadanie i struktura pracy

Temat wybrano ze względu na praktyczną potrzebę połączenia prostego dziennika żywieniowego z większą kontrolą nad danymi użytkownika. Istniejące aplikacje oferują rozbudowane katalogi i synchronizację, lecz zwykle opierają podstawowe funkcje na koncie oraz infrastrukturze usługodawcy. Dodatkową motywacją była możliwość sprawdzenia, na ile lokalny model językowy może być użyteczny w codziennej aplikacji działającej na zwykłym telefonie.

Celem pracy było zaprojektowanie i wykonanie aplikacji, która łączy prostotę licznika kalorii z prywatnością przetwarzania lokalnego. Najważniejsze dane mają pozostawać na urządzeniu, a działanie systemu nie powinno zależeć od utrzymywanego przez autora backendu. Rozwiązanie powinno udostępniać kilka metod dodawania produktów, automatyczne i ręczne cele żywieniowe, kalendarz historii oraz opcjonalnego lokalnego asystenta. Asystent ma wspierać rozmowę, natomiast wartości liczbowe powinny być wyznaczone przez jawne algorytmy i rekordy katalogu produktów.

Zakres pracy obejmuje:

- analizę problemu monitorowania energii i makroskładników;
- sformułowanie wymagań funkcjonalnych i нефunkcjonalnych;
- wybór technologii dla samodzielnej aplikacji Android;
- projekt lokalnej bazy danych i architektury warstwowej;
- implementację rejestrowania posiłków, profilu, kalendarza i skanera;
- integrację lokalnych modeli Qwen3 z mechanizmem ochronnym;
- weryfikację kodu oraz przygotowanie instalacyjnego pakietu APK.

W pracy zastosowano analizę literatury i oficjalnej dokumentacji technicznej, analizę porównawczą istniejących aplikacji, projektowanie i implementację oprogramowania oraz testy

techniczne. Poprawność obliczeń sprawdzano za pomocą deterministycznych przypadków testowych. Działanie interfejsu i lokalnego modelu oceniano również na fizycznym telefonie z systemem Android. Końcową wersję zweryfikowano przez analizę składni kodu, kontrolę konfiguracji Expo i produkcyjny eksport Hermes oraz kompilację natywnych zadań Androida.

Materiał źródłowy obejmuje publikacje naukowe dotyczące zapotrzebowania energetycznego i podaży białka, raporty instytucji publicznych, dokumentację React Native, Expo, SQLite, Open Food Facts i Qwen3, a także kod źródłowy aplikacji oraz wyniki własnych testów. Dokumentacja techniczna służy do opisu zachowania użytych narzędzi, natomiast publikacje i raporty stanowią podstawę części żywieniowej oraz porównawczej.

Po wstępie praca została podzielona na trzy rozdziały merytoryczne. Rozdział pierwszy zawiera podstawy obliczeń żywieniowych, analizę problemu i porównanie istniejących rozwiązań. Rozdział drugi opisuje realizację aplikacji *NutritionTracker*: wymagania, architekturę, narzędzia, interfejs i algorytmy. Rozdział trzeci przedstawia weryfikację rozwiązania, założenia prywatności, testy oraz ocenę końcowego rezultatu. Całość zamyka podsumowanie wyników pracy.

2. PODSTAWY MONITOROWANIA ŻYWIENIA

2.1. Bilans energetyczny

Energia dostarczana przez pożywienie jest zwykle podawana w kilokaloriach. Zmiana masy ciała w dłuższym okresie zależy między innymi od relacji między energią dostarczoną i wydatkowaną. Aplikacja do monitorowania diety nie mierzy bezpośrednio rzeczywistego wydatku energetycznego. Może jedynie wyznaczyć wartość początkową na podstawie modelu matematycznego, a następnie pomóc użytkownikowi konsekwentnie rejestrować spożycie.

Wartości energetyczne produktów są obliczane na podstawie zawartości składników dostarczających energię. Ogólne współczynniki Atwatera przypisują białku i dostępnym węglowodonom około 4 kcal na gram, a tłuszczom około 9 kcal na gram [11]. W uproszczonej postaci energia posiłku może być zatem wyrażona wzorem:

$$E = 4P + 4C + 9F, \quad (2.1)$$

gdzie:

- E oznacza energię w kilokaloriach,
- P oznacza ilość białka w gramach,
- C oznacza ilość węglowodanów w gramach,
- F oznacza ilość tłuszczów w gramach.

Wzór 2.1 jest wystarczający do prezentowania orientacyjnej sumy w interfejsie, ale nie zastępuje danych producenta. Różnice mogą wynikać między innymi z błonnika, alkoholi wielowodorotlenowych, zaokrągleń etykiety i szczegółowych współczynników właściwych dla konkretnych produktów. Z tego powodu aplikacja zachowuje możliwość ręcznego podania kalorii niezależnie od makroskładników.

2.2. Podstawowa przemiana materii

Podstawowa przemiana materii opisuje minimalny wydatek energetyczny organizmu w warunkach spoczynku. Do obliczeń wybrano równanie Mifflina–St Jeora, opracowane na podstawie danych 498 zdrowych osób [22]. Dla mężczyzn przyjęto:

$$BMR_m = 10W + 6,25H - 5A + 5, \quad (2.2)$$

a dla kobiet:

$$BMR_k = 10W + 6,25H - 5A - 161, \quad (2.3)$$

gdzie W jest masą ciała w kilogramach, H wzrostem w centymetrach, a A wiekiem w latach.

Równanie jest estymatorem populacyjnym. Nie uwzględnia bezpośrednio składu ciała, stanu zdrowia, adaptacji metabolicznej ani dokładnego wydatku wynikającego z treningu. Wynik nie powinien być przedstawiany jako pewna diagnoza. W aplikacji jest on określony jako wartość startowa, którą użytkownik może zastąpić celem ręcznym.

2.3. Całkowite zapotrzebowanie energetyczne

Dzienne zapotrzebowanie jest szacowane przez pomnożenie podstawowej przemiany materii przez współczynnik aktywności. W aplikacji dostępnych jest pięć poziomów opisanych w tabeli 1.

Tabela 1: Współczynniki aktywności zastosowane w aplikacji

Poziom aktywności	Współczynnik
Siedzący tryb życia	1,200
Lekka aktywność	1,375
Umiarkowana aktywność	1,550
Wysoka aktywność	1,725
Bardzo wysoka aktywność	1,900

Źródło: opracowanie własne.

Wartość po uwzględnieniu aktywności jest następnie modyfikowana przez cel użytkownika. Utrzymanie masy ma współczynnik 1,00, redukcja 0,85, stopniowe zwiększanie masy 1,08, a cel budowania masy mięśniowej 1,12. Ostateczna wartość jest zaokrąglana do 10 kcal:

$$E_d = \text{round}_{10}(BMR \cdot a \cdot g), \quad (2.4)$$

gdzie a oznacza współczynnik aktywności, a g współczynnik celu. Współczynniki celu są decyzyją projektową, a nie zaleceniem medycznym. Użytkownik otrzymuje możliwość ich pośredniego zastąpienia przez tryb ręczny.

2.4. Makroskładniki

Aplikacja przechowuje cele makroskładników jako procent dziennej energii. Jeżeli udziały białka, węglowodanów i tłuszczów oznaczono odpowiednio przez p_P , p_C i p_F , ich suma musi wynosić 100%. Docelową liczbę gramów wyznaczają zależności:

$$P_g = \frac{E_d \cdot p_P / 100}{4}, \quad (2.5)$$

$$C_g = \frac{E_d \cdot p_C / 100}{4}, \quad (2.6)$$

$$F_g = \frac{E_d \cdot p_F / 100}{9}. \quad (2.7)$$

Dla osób aktywnych fizycznie literatura sportowa często podaje zwiększone zapotrzebowanie na białko. Stanowisko International Society of Sports Nutrition wskazuje orientacyjny zakres 1,4–2,0 g/kg masy ciała na dobę dla osób ćwiczących, zależnie od rodzaju i intensywności aktywności [15]. Nowsze stanowiska podkreślają również znaczenie całkowitej energii, trwałości sposobu żywienia i indywidualnej reakcji organizmu [1]. W aplikacji konkretne pytanie o białko może zostać obsłużone przez deterministyczny mechanizm zwracający zachowawczy zakres 1,6–2,0 g/kg, zamiast przez swobodne generowanie liczb przez model językowy.

2.5. Dokładność danych

Wynik licznika zależy przede wszystkim od poprawnej gramatury i wartości odżywczych wybranego produktu. Różnice między producentami oraz sposobem przygotowania żywności sprawiają, że wynik ma charakter użytkowy, a nie laboratoryjny. Aplikacja pozwala dlatego zmieniać porcję i cele ręcznie, a dla produktów paczkowanych użytkownik może porównać dane z etykietą.

2.6. Codzienne trudności i istniejące rozwiązania

2.6.1 Dlaczego zapisywanie posiłków bywa trudne

Najwięcej czasu zajmuje wpisywanie posiłków, a nie samo sumowanie kalorii. Typowy posiłek może zawierać kilka produktów, na przykład ryż, kurczaka i brokuły. Jeżeli każdy składnik trzeba osobno znaleźć, wpisać jego masę i zatwierdzić, dodanie jednego posiłku staje się uciążliwe. Użytkownik może wtedy pomijać część wpisów, przez co dziennik przestaje dokładnie przedstawiać jego codzienne odżywianie.

Kolejna trudność polega na znalezieniu właściwego produktu. Niewielki lokalny katalog wystarcza dla podstawowych składników, ale produkty paczkowane występują w wielu odmianach i stale pojawiają się nowe. Z tego powodu aplikacja łączy lokalną listę najczęstszych produktów z otwartą bazą internetową, z której korzysta podczas skanowania kodu kreskowego.

Użytkownicy różnią się również doświadczeniem. Osoba początkująca może potrzebować automatycznie wyliczonego celu, natomiast bardziej doświadczona może chcieć samodzielnie ustawić kalorie i makroskładniki. Aplikacja powinna pozwalać na oba sposoby pracy.

Osobną decyzją jest sposób działania asystenta AI. Model chmurowy zwykle odpowiada szybciej, ale wymaga przesyłania danych oraz korzystania z serwera lub klucza API. Model lokalny lepiej chroni prywatność, lecz działa wolniej i ma mniejsze możliwości. W projekcie

wybrano więc opcjonalny model lokalny, który użytkownik może pobrać wtedy, gdy chce z niego korzystać.

2.6.2 Klasy istniejących rozwiązań

Na rynku można wyróżnić trzy podstawowe klasy narzędzi:

1. **Proste dzienniki lokalne:** pozwalają wpisać nazwę i kalorie, ale nie oferują szerokiej bazy produktów ani zaawansowanej personalizacji.
2. **Platformy chmurowe:** oferują duże katalogi, synchronizację i rozbudowane raporty, lecz zwykle wymagają konta i przekazywania danych na serwer.
3. **Asystenci generatywni:** pozwalają opisać posiłek językiem naturalnym, ale mogą wymyślać wartości odżywcze i nie gwarantują powtarzalności wyniku.

Projekt *NutritionTracker* łączy wybrane zalety tych podejść. Dane podstawowe są lokalne, skanowanie korzysta z otwartego katalogu produktów, natomiast asystent działa na urządzeniu. Model generatywny nie jest źródłem danych odżywczych. Podział ról pomiędzy kod deterministyczny i AI przedstawiono w tabeli 2.

Tabela 2: Podział odpowiedzialności między kod aplikacji i model językowy

Obszar	Kod deterministyczny	Model językowy
Kalorie i makroskładniki	Oblicza sumy, cele, porcje i przeliczenia	Nie powinien tworzyć wartości liczbowych
Dane produktu	Katalog lokalny lub Open Food Facts	Nie stanowi źródła danych produktu
Alergie i produkty unikane	Filtruje katalog przed pokazaniem wyników	Otrzymuje instrukcję, aby ich nie polecać
Rozmowa i wskazówki	Obsługuje pytania wysokiego ryzyka oraz kontekst	Formułuje ogólne wskazówki treningowe i żywieniowe
Historia	SQLite i AsyncStorage	Otrzymuje tylko ograniczony kontekst bieżącej rozmowy

Źródło: opracowanie własne.

2.6.3 Porównanie z MyFitnessPal i Cronometer

Do porównania wybrano MyFitnessPal oraz Cronometer, ponieważ są to dojrzałe aplikacje realizujące podobny podstawowy proces: utworzenie profilu, wyznaczenie celu, wyszukanie

żywności i prowadzenie dziennika. Porównanie dotyczy stanu oficjalnie opisanego przez producentów w czerwcu 2026 r. Nie ma ono wykazać przewagi *NutritionTracker* pod względem liczby funkcji. Celem jest wskazanie innego zestawu priorytetów projektowych.

MyFitnessPal udostępnia bezpłatne rejestrowanie żywności i ćwiczeń, makroskładniki, tworzenie własnych produktów, posiłków oraz przepisów, a także integracje z aplikacjami partnerskimi. Funkcje takie jak skaner kodów kreskowych, cele makroskładników w gramach, eksport danych, analiza żywności i brak reklam należą do płatnych poziomów Premium [26, 23]. Oficjalna dokumentacja wskazuje również, że bez Internetu użytkownik nie ma dostępu do bazy żywności, a zmiany wykonane offline muszą zostać później zsynchronizowane [25]. Usługa zbiera dane podczas tworzenia konta, korzystania z funkcji oraz z wybranych źródeł zewnętrznych [24].

Cronometer kładzie większy nacisk na szczegółowe składniki odżywcze. Bezpłatne konto obejmuje źródła analizowane laboratoryjnie, bazę społecznościową, własne produkty i przepisy, wybrane integracje oraz eksport surowych danych. Subskrypcja Gold dodaje między innymi rozbudowane wykresy, sugestie produktów, post, harmonogram makroskładników i brak reklam [6]. Aplikacja pozwala wyszukiwać produkty i skanować ich kody, a nowe etykiety mogą zostać przesłane do sprawdzenia przez zespół kuratorski [3]. Pierwsze uruchomienie wymaga utworzenia konta, podania adresu e-mail i jego potwierdzenia [5]. Producent deklaruje możliwość eksportu lub usunięcia danych oraz zabezpieczenia infrastruktury chmurowej [4]. Funkcja rozpoznawania posiłku ze zdjęcia jest opisana jako funkcja konta Gold [2].

Tabela 3: Porównanie założeń aplikacji *NutritionTracker*, *MyFitnessPal* i *Cronometer*

Kryterium	<i>NutritionTracker</i>	<i>MyFitnessPal</i>	<i>Cronometer</i>
Główny cel	Prosty dziennik kalorii i makro z lokalnym asystentem.	Rozbudowana platforma żywienia, aktywności i planów.	Szczegółowa analiza energii oraz składników odżywczych.
Konto i dane	Brak konta; profil i historia są lokalne.	Konto i synchronizacja z usługą.	Konto, potwierdzenie e-mail i infrastruktura chmurowa.
Praca offline	Dziennik, cele i katalog podstawowy działają bez sieci.	Ograniczona praca offline, bez dostępu do bazy żywności.	Usługa kontowa; pełna praca offline nie jest głównym założeniem.
Produkty i skaner	Lokalny katalog oraz Open Food Facts dla kodów.	Duża baza; skaner w płatnych poziomach Premium.	Bazy kuratorskie, skaner i zgłaszanie nowych etykiet.
AI i prywatność	Opcjonalny model Qwen3 działa lokalnie na urządzeniu.	Funkcje zależą od aktualnej oferty usługi.	Wybrane funkcje AI, np. zdjęcie posiłku, są częścią Gold.
Najważniejsza cecha	Minimalizacja danych i prosty przepływ.	Skala bazy oraz ekosystem integracji.	Dokładność i szerokość analizy składników.

Źródło: opracowanie własne.

Zestawienie pokazuje, że *MyFitnessPal* i *Cronometer* oferują szersze bazy, integracje, raporty i synchronizację. Są lepszym wyborem dla użytkownika, który potrzebuje kompletnego ekosystemu albo szczegółowej analizy mikroskładników. *NutritionTracker* odpowiada na ważniejszy problem: umożliwia szybkie prowadzenie podstawowego dziennika bez obowiązkowego konta, reklam, subskrypcji i centralnego przechowywania historii. Prostota nie oznacza tu braku funkcji, lecz świadome ograniczenie ich liczby do najczęstszych zadań.

2.6.4 Uzasadnienie podejścia *offline-first*

Architektura *offline-first* oznacza, że podstawowy stan systemu jest przechowywany i przetwarzany na urządzeniu. Wybór ten wynika z charakteru aplikacji. Profil i dziennik są odczytywane wielokrotnie każdego dnia, a większość operacji dotyczy jednego użytkownika i nie wymaga współdzielenia z innymi urządzeniami. Wysyłanie każdego wpisu do serwera zwiększałoby liczbę elementów systemu bez konieczności wynikającej z podstawowego celu projektu.

Takie podejście ma następujące zalety:

- działanie bez rejestracji i logowania;

- brak kosztu utrzymywania serwera aplikacyjnego;
- mniejsze ryzyko masowego wycieku centralnej bazy;
- natychmiastowy dostęp do lokalnych danych;
- możliwość korzystania z licznika podczas braku sieci;
- prostsza dystrybucja w formie samodzielnego pliku APK.

Internet jest potrzebny jedynie podczas pierwszego pobrania modelu oraz wyszukiwania nieznanego kodu kreskowego. Pozostałe podstawowe funkcje korzystają z danych zapisanych na urządzeniu.

2.6.4.1 Motywacja wyboru lokalnej bazy SQLite

Do przechowywania profilu, posiłków i pamięci produktów wybrano SQLite [10]. Dane mają strukturę tabelaryczną, wymagają filtrowania po dacie, sumowania wartości oraz trwałego zachowania relacji między rekordem posiłku i dniem. Relacyjna baza pozwala wykonywać te operacje za pomocą zapytań, indeksów i transakcji, zamiast za każdym razem odczytywać i przepisywać jeden duży dokument JSON.

W tabeli `meal_s` utworzono indeks daty, ponieważ ekran dnia i kalendarz regularnie pobierają rekordy z wybranego okresu. Tryb WAL zmniejsza ryzyko blokowania odczytu podczas zapisu i zwiększa odporność operacji [34]. Migracje sprawdzają istnienie kolumn profilu i dodają nowe pola bez usuwania wcześniejszych danych. SQLite pełni zatem nie tylko rolę magazynu, ale również zapewnia spójny model trwałości aplikacji.

Prostszy magazyn klucz–wartość `AsyncStorage` pozostawiono dla danych, które naturalnie mają formę ustawienia lub niewielkiego dokumentu: konfiguracji AI oraz historii rozmowy zapisanej jako JSON. Rozdzielenie magazynów wynika z rodzaju danych, a nie z założenia, że jeden mechanizm jest odpowiedni do wszystkich zastosowań.

Wariant lokalny wybrano ze względu na prostotę, działanie offline i ograniczenie centralnego gromadzenia danych. Serwer byłby potrzebny przede wszystkim do synchronizacji wielu urządzeń lub współdzielenia historii. Funkcje te nie należą do przyjętego zakresu projektu.

3. REALIZACJA APLIKACJI NUTRITIONTRACKER

Cel, zakres i wkład własny

Cel przedstawiony we wstępie przełożono na pięć założeń, które kierowały implementacją. Projekt ma być prostym dziennikiem dla osoby monitorującej kalorie i makroskładniki, a nie systemem klinicznym ani platformą społecznościową.

Tabela 4: Główne założenia projektowe aplikacji

Założenie	Znaczenie w projekcie
Prostota	Jeden główny ekran dnia, cztery podstawowe zakładki i brak funkcji niezwiązanych bezpośrednio z żywieniem.
Wygoda	Kilka metod dodawania: gotowe propozycje, wyszukiwarka, kod kreskowy i wpis ręczny, połączone jednym edytorem produktu.
Prywatność	Profil, posiłki i rozmowa są przechowywane lokalnie; konto sieciowe nie jest wymagane.
Niezależność	Podstawowe funkcje nie zależą od backendu autora, klucza API ani aktywnej subskrypcji.
Przewidywalność	Cele i wartości odżywcze oblicza kod deterministyczny, a nie model generatywny.

Źródło: opracowanie własne.

Użytkownik docelowy

Podstawowym użytkownikiem jest pełnoletnia lub starsza nastoletnia osoba aktywna fizycznie, która chce obserwować dzienne spożycie i realizować cel redukcji, utrzymania albo zwiększania masy. Aplikacja nie jest przeznaczona do diagnozowania chorób, leczenia zaburzeń odżywiania ani układania diet klinicznych. W przypadku chorób, ciąży, przyjmowania leków lub reakcji alergicznej użytkownik powinien skorzystać z pomocy wykwalifikowanego specjalisty.

Wkład własny autora

Rezultatem pracy nie jest wyłącznie warstwa interfejsu, lecz kompletny przepływ od wprowadzenia danych do ich trwałego zapisu, przeliczenia i prezentacji. Wkład własny autora obejmuje zaprojektowanie architektury, implementację logiki żywieniowej, integrację źródeł danych oraz przygotowanie natywnej wersji Android. Najważniejsze elementy zestawiono w tabeli 5.

Tabela 5: Najważniejsze elementy wkładu własnego autora

Obszar	Wykonane rozwiązanie	Znaczenie inżynierskie
Architektura	Podział na ekrany, stan, usługi i lokalny dostęp do danych.	Oddzielenie interfejsu od logiki domenowej i infrastruktury.
Logika żywieniowa	Obliczanie celu, makroskładników, porcji i sum dnia oraz walidacja zakresów.	Powtarzalne wyniki niezależne od modelu generatywnego.
Rejestrowanie żywności	Katalog lokalny, rekomendacje, wpis ręczny, skaner kodów oraz edycja zapisanych produktów.	Obsługa kilku ścieżek użytkownika przy wspólnym edytorze i modelu danych.
Dane zewnętrzne	Integracja Open Food Facts, normalizacja odpowiedzi i lokalna pamięć zeskanowanych produktów.	Połączenie usługi sieciowej z działaniem offline po pierwszym odczycie.
Lokalna AI	Pobieranie, ładowanie, przełączanie i usuwanie modeli GGUF oraz strumieniowanie i zatrzymywanie odpowiedzi.	Integracja natywnego silnika wnioskowania z aplikacją React Native.
Adaptacja do urządzenia	Ocena pamięci i klasy telefonu, rekomendacja modelu, kontrola miejsca i komunikaty ostrzegawcze.	Ograniczenie ryzyka awarii na urządzeniach o różnych parametrach.
Dystrybucja	Konfiguracja projektu Android i proces budowania samodzielnego APK.	Uruchomienie bez Expo Go, serwera Metro i backendu autora.

Źródło: opracowanie własne.

Istotną decyzją projektową było rozdzielenie odpowiedzialności. Model językowy odpowiada za formę dialogu i ogólne wskazówki, natomiast liczby żywieniowe pochodzą z algorytmów oraz rekordów produktów. AI rozszerza funkcjonalność, ale nie stanowi fundamentu poprawności podstawowego dziennika.

3.1. Wymagania funkcjonalne i нефункционалне

3.1.1 Wymagania funkcjonalne

Wymagania funkcjonalne opisują czynności, które system ma umożliwiać użytkownikowi. Zostały wyprowadzone z typowego przebiegu korzystania: konfiguracja profilu, ustalenie celu, dodanie żywności, obserwacja postępu i opcjonalna rozmowa z lokalnym asystentem. Funkcje podstawowego dziennika nie mogą zależeć od aktywacji AI.

Tabela 6: Wymagania funkcjonalne systemu

Identyfikator	Wymaganie
F01	System tworzy lokalny profil bez wymagania konta sieciowego.
F02	Użytkownik może podać nazwę, wzrost, masę, wiek, płeć, aktywność i cel.
F03	System automatycznie wyznacza startowy cel kalorii i proporcje makroskładników.
F04	Użytkownik może przełączyć profil na tryb ręczny.
F05	Procenty białka, węglowodanów i tłuszczów muszą sumować się do 100%.
F06	Użytkownik może zapisać alergie oraz produkty, których chce unikać.
F07	Użytkownik może dodać posiłek ręcznie.
F08	Kalorie mogą być obliczone z gramów makroskładników.
F09	Użytkownik może wyszukiwać najczęstsze produkty w lokalnym katalogu.
F10	Zeskanowany produkt jest przekazywany bezpośrednio do formularza kontroli gramatury i wartości odżywczych.
F11	Każdy wybrany produkt jest zapisywany jako oddzielny wpis.
F12	Użytkownik może zmienić gramaturę i wartości na 100 g przed zapisem oraz edytować je później.
F13	Użytkownik może zeskanować kod EAN lub UPC aparatem.
F14	System pobiera wartości produktu z Open Food Facts i zapisuje je lokalnie.
F15	System prezentuje dzienne sumy kalorii, białka, węglowodanów i tłuszczów.
F16	Użytkownik może usuwać wpisy posiłków.
F17	Kalendarz przedstawia sumę kalorii w poszczególnych dniach.
F18	Użytkownik może później wybrać model Qwen3 1.7B albo Qwen3 4B w ustawieniach profilu.
F19	System sugeruje model na podstawie pamięci i klasy urządzenia oraz blokuje wariant przekraczający bezpieczne minimum pamięci.

Identyfikator	Wymaganie
F20	System wyświetla postęp pobierania i ładowania modelu w zakresie od 0 do 100%.
F21	Użytkownik może wyłączyć AI lub usunąć model z pamięci telefonu.
F22	Asystent otrzymuje profil, sumy wszystkich wpisów oraz szczegóły mieszczące się w oknie kontekstu bieżącego albo wskazanego miesiąca.
F23	Odpowiedź modelu jest wyświetlana strumieniowo.
F24	Użytkownik może przerwać generowanie odpowiedzi.
F25	Historia rozmowy jest zachowywana lokalnie i może zostać wyczyszczona.
F26	Użytkownik może korzystać z pełnego dziennika bez pobierania modelu AI.
F27	Użytkownik może dodać często spotykany produkt bez ręcznego wpisywania wszystkich wartości odżywczych.
F28	Użytkownik może przeglądać wcześniejsze dni bez połączenia z serwerem aplikacyjnym.
F29	Po uruchomieniu system pokazuje logo, a następnie bezpośrednio ekran bieżącego dnia.
F30	Użytkownik może anulować trwające pobieranie pliku modelu.
F31	System porządkuje propozycje posiłków według celu sylwetkowego, pory dnia, bieżącego spożycia, porcji białka, udziału produktów roślinnych i wykluczeń żywieniowych.
F32	Użytkownik może przełączyć interfejs oraz język odpowiedzi asystenta między językiem polskim i angielskim.
F33	Użytkownik może jednym potwierdzonym poleceniem usunąć profil, posiłki, pamięć produktów, rozmowę, ustawienia oraz wszystkie pobrane modele.
F34	Pytania o kalendarz, kalorie i makroskładniki są odpowiadane bezpośrednio na podstawie danych SQLite, bez powierzania wartości liczbowych modelowi językowemu.

Źródło: opracowanie własne.

3.1.2 Wymagania niefunkcjonalne

Wymagania niefunkcjonalne określają sposób działania systemu. W tym projekcie szczególne znaczenie mają prywatność, łatwość rozpoczęcia pracy, przewidywalność obliczeń i niezależność od infrastruktury. Wymaganie „wygodny dla każdego” byłoby niemożliwe do obiektywnego potwierdzenia, dlatego zastąpiono je mierzalnymi warunkami: brakiem obowiązkowej rejestracji, ograniczoną liczbą głównych ekranów, kilkoma metodami dodawania i możliwością wyłączenia funkcji wymagających dużych zasobów.

Tabela 7: Wymagania niefunkcjonalne systemu

Identyfikator	Wymaganie
N01	Podstawowe funkcje działają bez własnego serwera i bez połączenia z Internetem.
N02	Dane profilu i posiłków są przechowywane lokalnie w SQLite.
N03	Interfejs jest czytelny na ekranie telefonu w orientacji pionowej.
N04	Wartości liczbowe są wyznaczane przez jawne i powtarzalne algorytmy.
N05	Model AI nie blokuje możliwości korzystania z funkcji żywieniowych.
N06	Przed pobraniem modelu sprawdzana jest dostępna przestrzeń dyskowa.
N07	Niepełny plik modelu nie może zostać uznany za poprawnie pobrany.
N08	System ostrzega o możliwej niskiej wydajności i błędach na słabym urządzeniu.
N09	Aplikacja prosi o dostęp do aparatu dopiero przy skanowaniu kodu.
N10	Kod jest podzielony na warstwy interfejsu, stanu, usług i dostępu do danych.
N11	Pakiet instalacyjny działa bez uruchomionego środowiska deweloperskiego.
N12	Użytkownik może całkowicie zrezygnować z pobierania lokalnego modelu.
N13	Korzystanie z podstawowego dziennika nie wymaga subskrypcji ani oglądania reklam.
N14	Po pierwszym uruchomieniu użytkownik może przejść do aplikacji bez zakładania konta.
N15	Główne zadania są dostępne z czterech stałych zakładek i ekranów modalnych o małej głębokości nawigacji.

Identyfikator	Wymaganie
N16	Brak Internetu nie uniemożliwia odczytu profilu, historii, sum dnia ani lokalnego katalogu.
N17	Funkcje opcjonalne nie mogą uniemożliwiać korzystania z podstawowych funkcji po wystąpieniu błędu.
N18	Przed pobraniem modelu system ostrzega o wymaganym połączeniu i konieczności pozostawienia aplikacji otwartej.
N19	Plik modelu znajduje się w prywatnym katalogu aplikacji i jest usuwany razem z jej danymi podczas standardowego odinstalowania.
N20	Wersja instalacyjna blokuje systemową kopię danych aplikacji i nie zezwala na nieszyfrowany ruch HTTP.
N21	Rekomendacje posiłków są wyznaczone lokalnie przez powtarzalny algorytm i nie wymagają modelu językowego.
N22	Wybrany język jest zapisywany lokalnie i stosowany po ponownym uruchomieniu aplikacji.
N23	System udostępnia jawny i potwierdzany mechanizm trwałego usunięcia wszystkich danych lokalnych bez kontaktu z administratorem usługi.
N24	Kryteria rankingu posiłków są jawne, możliwe do przetestowania i odwołują się do opisanych źródeł naukowych, bez deklarowania diagnozy ani indywidualnej porady medycznej.

Źródło: opracowanie własne.

3.1.3 Powiązanie wymagań z celami

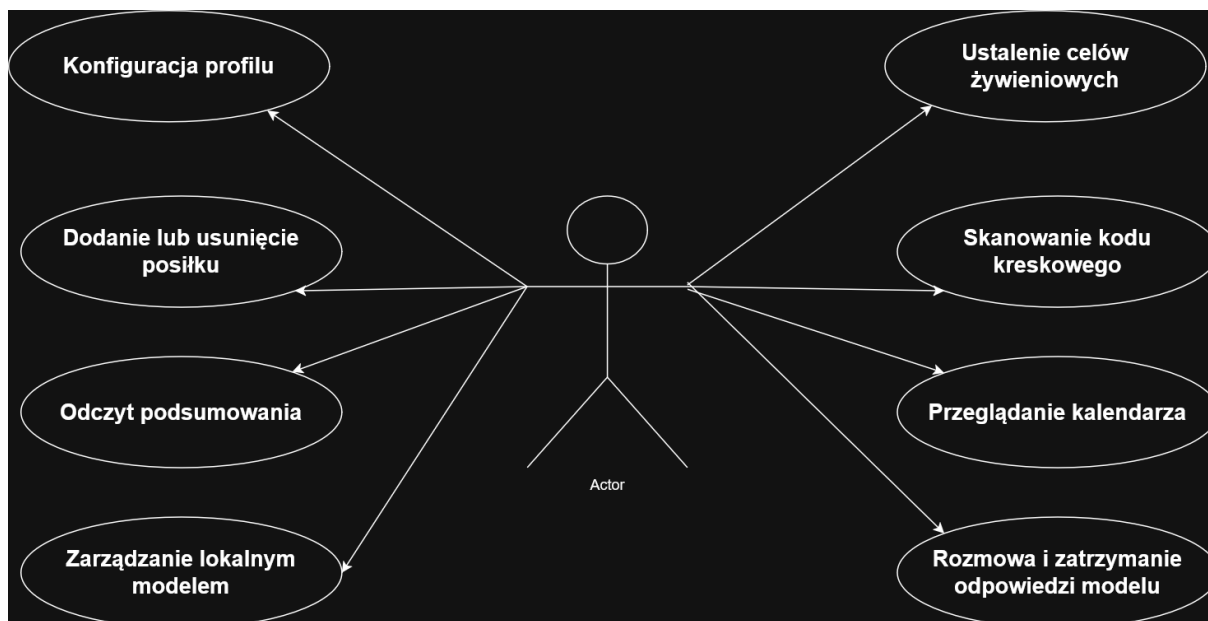
Tabela 8: Powiązanie głównych celów z wymaganiami systemu

Cel	Wymagania	Sposób realizacji
Prostota	F01, F07–F12, F27, F32, N14–N15, N22	Lokalny profil, jeden formularz posiłku i kilka krótkich metod dodawania.
Prywatność	F01, F25, F33–F34, N01–N02, N14, N16, N19–N20, N23	Prywatne magazyny urządzenia, zablokowana kopia systemowa i pełne czyszczenie danych kontrolowane przez użytkownika.
Niezależność	F21, F26, F28, N01, N11–N13, N17	Samodzielny APK, opcjonalne AI i brak wymaganego backendu.
Wygoda	F09–F17, F27, F31–F32, F34, N03, N15, N22	Katalog, rekomendacje, skaner, wspólny edytor produktów, podsumowanie oraz kalendarz.
Poprawność	F03–F06, F12, F31, F34, N04, N07–N08, N21, N24	Walidacja profilu, jawne wzory, kontrola procentów, rekomendacji i plików modelu.

Źródło: opracowanie własne.

3.1.4 Przypadki użycia

Główne interakcje użytkownika z systemem przedstawia rysunek 1. Funkcje AI są częścią opcjonalną: pominięcie pobierania modelu nie ogranicza konfiguracji profilu, rejestrowania posiłków ani przeglądania historii.



Rysunek 1: Diagram głównych przypadków użycia aplikacji *NutritionTracker*

Źródło: opracowanie własne.

3.1.4.1 Konfiguracja profilu

Użytkownik uruchamia aplikację, widzi znak NutritionTracker, a następnie ekran bieżącego dnia. Konfiguracja AI nie przerywa pierwszego uruchomienia. W profilu użytkownik podaje parametry ciała i aktywności. System oblicza wartość startową. Użytkownik zatwierdza ją albo wybiera tryb ręczny i wpisuje własne cele. Zapis powoduje natychmiastową aktualizację ekranu głównego.

3.1.4.2 Dodanie kilku produktów

Użytkownik otwiera formularz posiłku i wyszukuje kolejno ryż oraz pierś kurczaka albo wybiera gotową propozycję. System dodaje oba produkty do listy roboczej jako oddzielne pozycje. Dla każdej z nich użytkownik może sprawdzić nazwę, gramaturę, wartości na 100 g oraz wynik dla wybranej porcji. Po zatwierdzeniu powstają dwa niezależne rekordy, które można później otworzyć i edytować.

3.1.4.3 Wybór rekomendowanego posiłku

Po otwarciu formularza aplikacja odczytuje cel użytkownika i podsumowanie wybranego dnia. Dla celu budowy masy wyżej umieszcza zestawy bogate w białko. Podczas redukcji większe znaczenie ma stosunek białka do kalorii. Z listy są wcześniej usuwane produkty zgodne z alergiami i wykluczeniami zapisanymi w profilu. Użytkownik widzi proponowany zestaw, jego kaloryczność, ilość białka oraz krótkie uzasadnienie, po czym może zmienić porcje przed zapisem.

3.1.4.4 Skanowanie produktu

Użytkownik uruchamia skaner, udziela jednorazowej zgody na użycie aparatu i umieszcza kod w ramce. Jeżeli produkt znajduje się w pamięci podręcznej, wynik jest zwracany bez połączenia sieciowego. W przeciwnym razie aplikacja wysyła kod do API Open Food Facts, pobiera wybrane pola i zapisuje znormalizowany rekord w SQLite. Następnie wraca do rozwiniętego formularza produktu, w którym użytkownik kontroluje porcję i wartości przed zapisem.

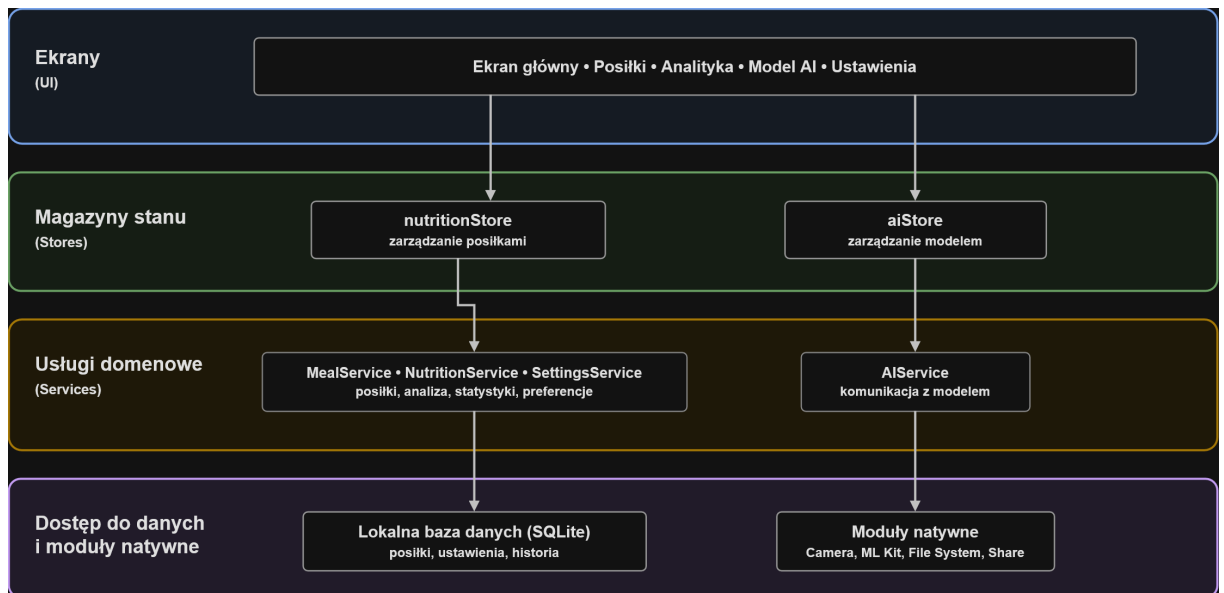
3.1.4.5 Rozmowa z lokalnym asystentem

Użytkownik pobiera jeden z dwóch modeli i otwiera zakładkę rozmowy. Model jest ładowany do pamięci przy pierwszym użyciu. Wiadomość, ostatnie elementy historii, profil oraz dane posiłków są przekazywane do lokalnego silnika. Domyślnie pobierane są wszystkie wpisy z bieżącego miesiąca. Jeżeli pytanie zawiera nazwę innego miesiąca albo zapis roku i miesiąca, aplikacja jednorazowo buduje kontekst z tego okresu. Sumy miesiąca i dni obejmują wszystkie rekordy. Szczegóły wpisów są dodawane tylko do granicy okna kontekstu modelu. Tokeny odpowiedzi pojawiają się stopniowo, a znaczniki wewnętrznego rozumowania nie są prezentowane. Użytkownik może przerwać obliczenia. Pytania rozpoznane jako medyczne albo dotyczące prostego celu białka są obsługiwane przed wywołaniem modelu. Tak samo obsługiwane są pytania o kalendarz, kalorie i makroskładniki. Aplikacja buduje wtedy odpowiedź bezpośrednio z wyniku zapytania SQLite. Dzięki temu lokalny model nie może błędnie stwierdzić, że nie ma dostępu do danych, ani zmienić wartości zapisanych w dzienniku.

3.2. Architektura i diagramy systemu

3.2.1 Architektura logiczna

Aplikacja ma architekturę warstwową. Kierunek zależności prowadzi od ekranów przez magazyny stanu i usługi do lokalnego dostępu do danych lub natywnych modułów. Schemat przedstawiono na rysunku 2.



Rysunek 2: Architektura logiczna aplikacji *NutritionTracker*

Źródło: opracowanie własne.

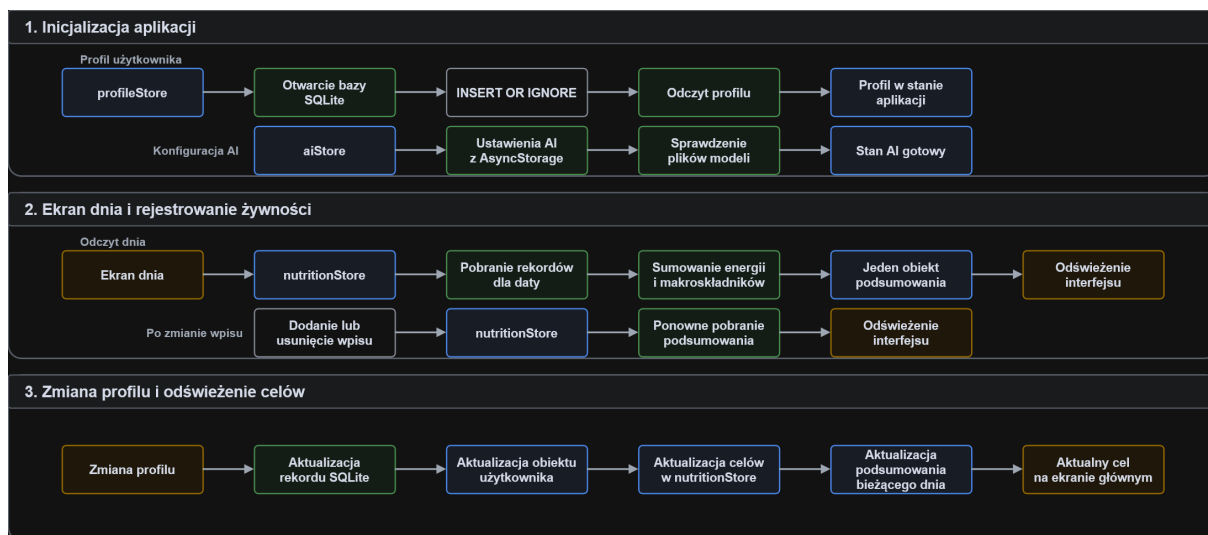
Ekrany nie zawierają bezpośrednio definicji bazy ani szczegółów inicjalizacji modelu. Operacje na posiłkach są wykonywane przez `nutritionStore`, a operacje modelu przez `aiStore`. Usługi domenowe pozostają niewielkie i odpowiadają za pojedyncze obszary. Struktura katalogu `src` zawiera 41 plików JavaScript i 5394 niepuste wiersze kodu źródłowego [17].

3.2.2 Przepływ danych

Po uruchomieniu aplikacji `profileStore` otwiera bazę SQLite. Jeżeli tabela profilu nie ma rekordu, wykonywany jest `INSERT OR IGNORE` dla lokalnego użytkownika. Następnie profil jest odczytywany i umieszczany w stanie aplikacji. Równolegle `aiStore` odczytuje ustawienia AI z `AsyncStorage` i sprawdza obecność plików modeli.

Ekran dnia pobiera rekordy dla wskazanej daty. Warstwa danych sumuje energię i makroskładniki, po czym zwraca jeden obiekt podsumowania. Po dodaniu, edycji lub usunięciu wpisu magazyn ponownie pobiera ten obiekt. Interfejs nie utrzymuje więc kilku niezależnych kopii tych samych sum.

Zmiana profilu aktualizuje rekord SQLite, obiekt użytkownika, aktywne cele w magazynie żywieniowym i podsumowanie bieżącego dnia. Rozwiązuje to problem, w którym ekran główny mógłby wyświetlać poprzedni cel po zmianie aktywności. Pełny przepływ od wprowadzenia danych do odświeżenia interfejsu przedstawiono na rysunku 3.



Rysunek 3: Przepływ danych podczas rejestrowania żywności

Źródło: opracowanie własne.

3.2.3 Model danych

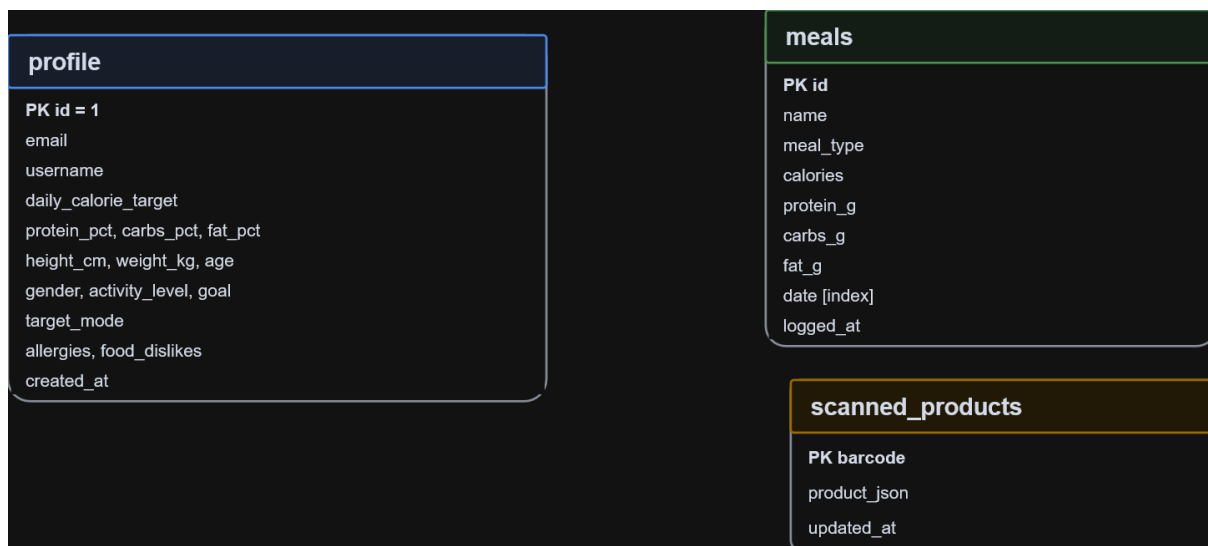
Lokalna baza `nutritiontracker.db` zawiera trzy tabele przedstawione w tabeli 9.

Tabela 9: Tabele lokalnej bazy danych

Tabela	Najważniejsze pola	Przeznaczenie
profile	cel kalorii, udziały makro, parametry ciała, aktywność, cel, alergię	Jeden lokalny profil użytkownika o identyfikatorze 1
meals	nazwa, typ posiłku, kalorie, makroskładniki, gramatura, klucz katalogu, kod kreskowy, źródło, data	Historia osobno zapisanych i edytowalnych produktów
scanned_products	kod kreskowy, dokument JSON, data aktualizacji	Pamięć podręczna produktów pobranych z Open Food Facts

Źródło: opracowanie własne.

Graficzny schemat pól oraz zależności realizowanych przez aplikację przedstawiono na rysunku 4.



Rysunek 4: Schemat głównych pól lokalnej bazy danych nutritiontracker.db

Źródło: opracowanie własne.

Na kolumnie `meals.date` utworzono indeks. Jest to pole używane przy każdym odczycie dnia oraz podczas agregacji miesiąca. Profil ma ograniczenie CHECK (`id = 1`), ponieważ aplikacja nie obsługuje wielu kont na jednym urządzeniu. Klucze obce nie są tu potrzebne: wpisy nie należą do osobnej tabeli kont, a pamięć zeskanowanych produktów jest niezależnym cache'em identyfikowanym przez kod kreskowy.

3.2.3.1 Migracje

Podczas otwarcia bazy wykonywane jest `CREATE TABLE IF NOT EXISTS`. Następnie aplikacja pobiera wynik `PRAGMA table_info(profile)` i dodaje brakujące kolumny przez `ALTER TABLE`. Starszą bazę można w ten sposób otworzyć bez kasowania historii.

3.2.4 Nawigacja

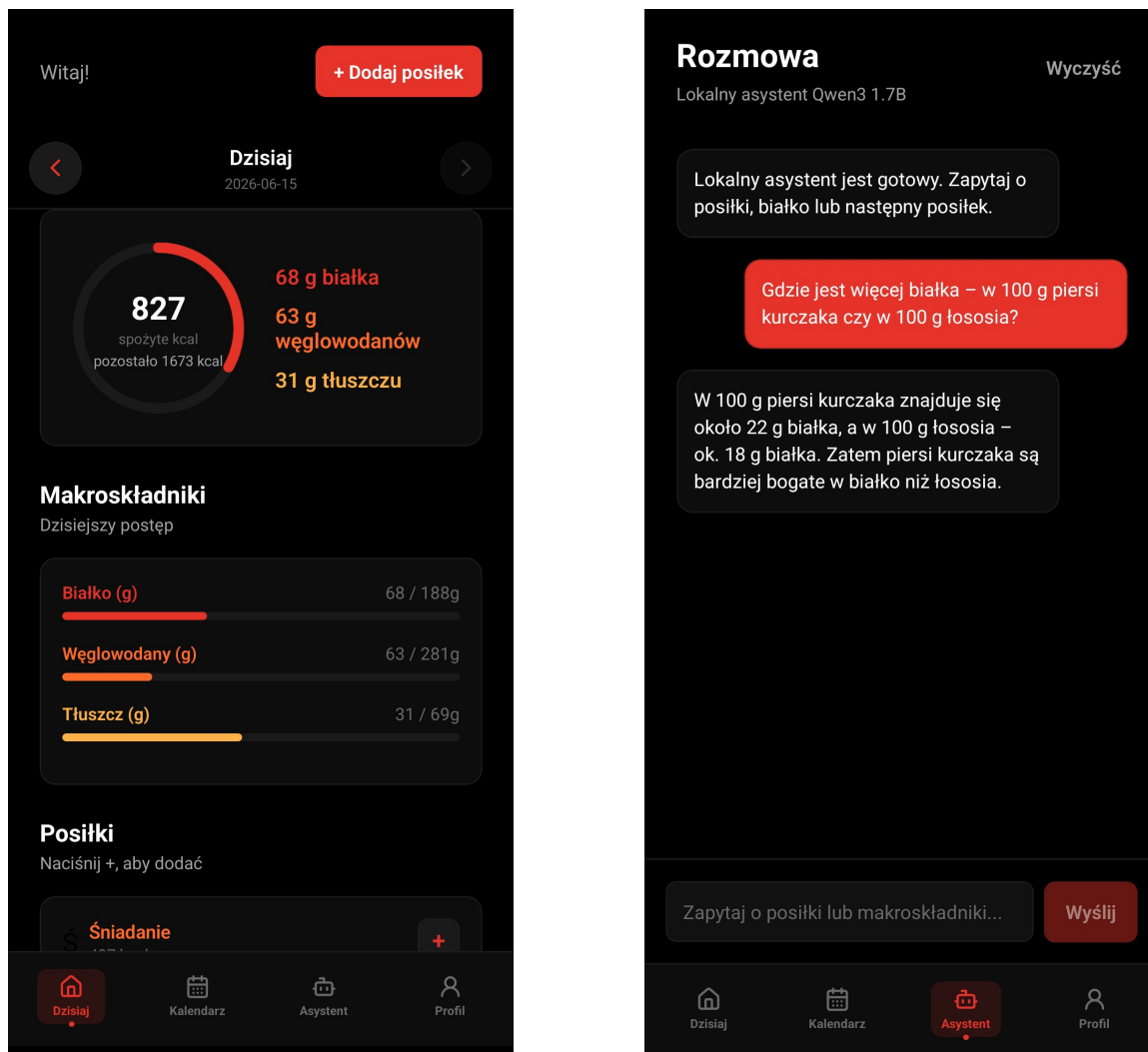
Dolna nawigacja zawiera cztery stałe cele:

- **Today:** bieżące podsumowanie i lista posiłków;
- **Calendar:** historia miesięczna;
- **Coach:** rozmowa z lokalnym modelem;
- **Profile:** parametry ciała, cele, alergie i ustawienia AI.

Dodawanie posiłku i skanowanie są operacjami krótkimi, dlatego zastosowano prezentację modalną od dołu. Natywny ekran startowy i krótki ekran inicjalizacji pokazują logo aplikacji. Po otwarciu lokalnej bazy oraz ustawień program przechodzi zawsze do zakładki **Today**. Opcjonalny model językowy nie stanowi więc bramki wejściowej do dziennika.

3.2.5 Projekt interfejsu

Interfejs wykorzystuje ciemne tło, jasną typografię i czerwono-pomarańczowe akcenty. Czerwony wskazuje element główny i białko, pomarańczowy węglowodany, a żółty tłuszcze. Stałe odstępy i promienie obramowania są zebrane w module motywu.



Rysunek 5: Ekran dziennego podsumowania oraz rozmowa z lokalnym asystentem

Źródło: zrzut ekranu aplikacji, opracowanie własne.

Rysunek 5 pokazuje dwa główne przepływy aplikacji: kontrolę dziennej energii oraz rozmowę z lokalnym modelem Qwen3 1.7B. Ekran główny eksponuje kalorie, makroskładniki i posiłki, a zakładka rozmowy pozwala zadać pytanie bez wysyłania historii żywienia do zewnętrznego asystenta AI.

Profil

Cele, bezpieczeństwo żywienia i AI

Język

English **Polski**

Dane i cel

Imię lub pseudonim

Wzrost (cm) Masa (kg) Wiek

180 80 20

Płeć

Mężczyzna Kobieta

Aktywność

Siedząca Lekka **Umiarkowana**

Wysoka Bardzo wysoka

Cel

Redukcja Utrzymanie

Kontrolowany przyrost **Budowa masy**

Dzisiaj

Kalendarz

Asystent

Profil

Profil

Cele, bezpieczeństwo żywienia i AI

Bezpieczeństwo żywienia

Alergie

Owsianka

Produkty do unikania

Grzyby, wieprzowina

Pasujące produkty są ukrywane, ale przy ciężkiej alergii zawsze sprawdź etykietę.

Tryb AI

Tryb

Lokalny **Wyłączony**

Dane i prywatność

Instalacja nowszego APK jako aktualizacji zachowuje profil, posiłki, rozmowę, ustawienia i pobrane modele.

Usuń wszystkie dane aplikacji

Dzisiaj

Kalendarz

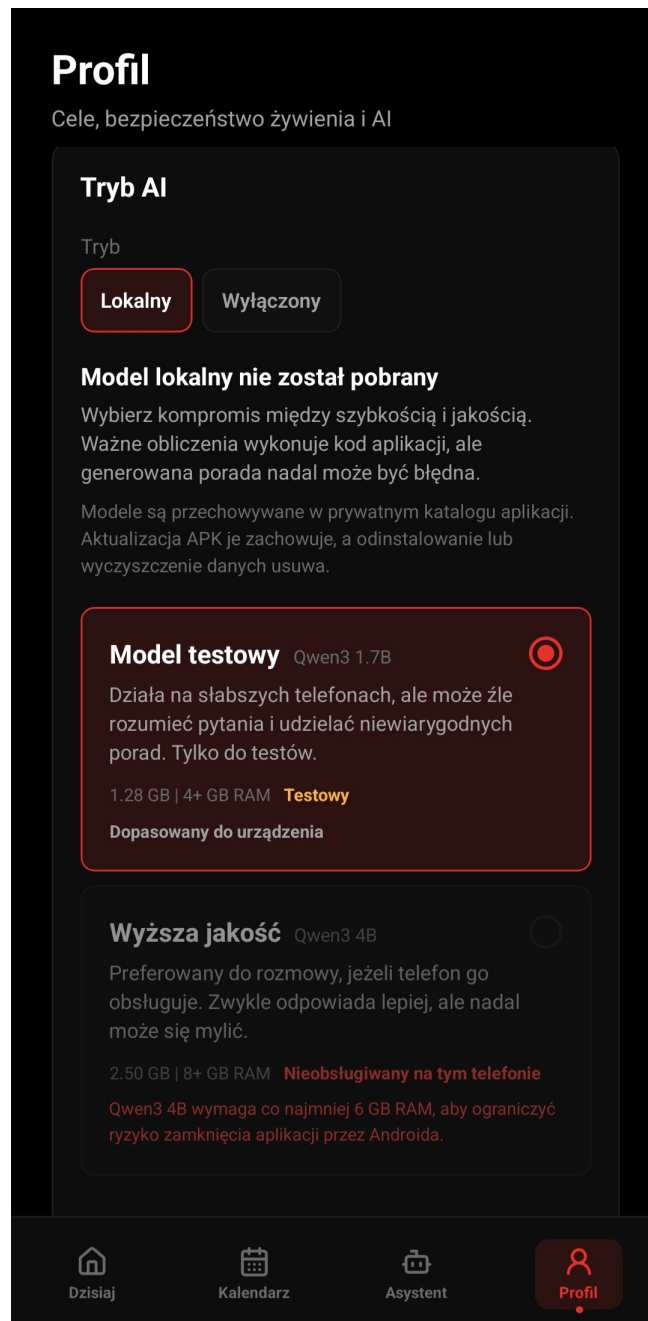
Asystent

Profil

Rysunek 6: Ustawienia danych, celu i bezpieczeństwa żywienia

Źródło: zrzut ekranu aplikacji, opracowanie własne.

Profil z rysunku 6 łączy dane potrzebne do obliczeń, preferencje językowe, wybór celu, alergię oraz produkty wykluczone. W tej samej zakładce użytkownik może wyłączyć asystenta albo trwale usunąć dane aplikacji.



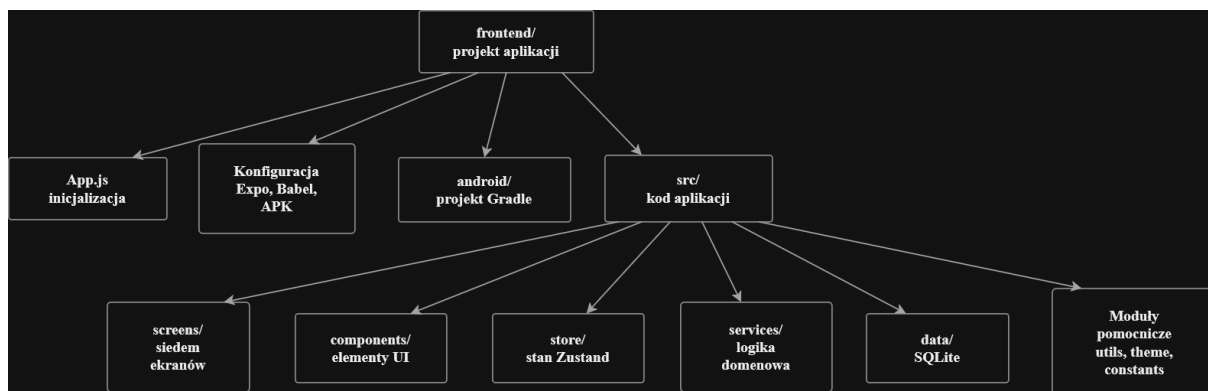
Rysunek 7: Konfiguracja lokalnego modelu AI

Źródło: zrzut ekranu aplikacji, opracowanie własne.

Widok z rysunku 7 pokazuje rozmiar pliku, zalecaną pamięć RAM i komunikat blokujący wariant zbyt ciężki dla danego telefonu.

3.2.6 Struktura modułów projektu

Najważniejsze katalogi i ich odpowiedzialności przedstawiono graficznie na rysunku 8.



Rysunek 8: Podział kodu źródłowego aplikacji na moduły

Źródło: opracowanie własne.

Warstwa ekranów korzysta z komponentów, magazynów stanu i usług. Usługi realizują obliczenia oraz integracje, natomiast moduł data ukrywa szczegóły lokalnej bazy danych. Część android zawiera kod i konfigurację wygenerowaną dla natywnego pakietu Android.

Rozkład niepustych wierszy pomiędzy główne warstwy przedstawia rysunek 9. Największa część przypada na ekrany, ponieważ obejmuje formularze, obsługę stanów interakcji i style React Native. Logika domenowa pozostaje wydzielona w usługach, magazynach i warstwie dostępu do danych.



Rysunek 9: Rozkład niepustych wierszy kodu pomiędzy warstwy aplikacji

Źródło: opracowanie własne.

3.3. Narzędzia i biblioteki

3.3.1 Android i Android Studio

Android jest docelową platformą projektu. Minimalna wersja API została ustawiona na 24, a wersja docelowa na API 35. Aplikacja korzysta z aparatu, lokalnego systemu plików, SQLite oraz natywnych bibliotek C/C++ wymaganych przez silnik modelu językowego. Pakiet wynikowy

jest budowany przez Gradle. Android Studio wykorzystano do zarządzania zestawem Android SDK, uruchamiania emulatora, analizy komunikatów Logcat oraz kontroli wygenerowanego projektu Gradle.

System Android zarządza cyklem życia aktywności i może zakończyć proces działający w tle, zwłaszcza przy presji na pamięć [14]. Jest to szczególnie ważne dla lokalnego modelu zajmującego kilka gigabajtów. Obecna wersja pozwala przełączyć się do innej aplikacji, ale nie może zagwarantować, że długie generowanie będzie kontynuowane. Pełna obsługa takiego scenariusza wymagałaby natywnej usługi pierwszoplanowej i trwałego mechanizmu wznawiania.

3.3.2 React Native

React Native umożliwia tworzenie natywnego interfejsu mobilnego przy użyciu JavaScript i komponentowego modelu React [21]. Projekt korzysta z React Native 0.79.6 i React 19.0.0. React odpowiada za model komponentów, stanu i aktualizacji widoku, natomiast React Native odwzorowuje komponenty na elementy interfejsu platformy mobilnej. Aplikacja nie jest stroną internetową uruchomioną w komponencie WebView.

Technologię wybrano przede wszystkim dlatego, że pozwala używać jednego języka do interfejsu, zarządzania stanem i logiki żywieniowej, a jednocześnie udostępnia dostęp do natywnych funkcji Androida. Było to istotne, ponieważ aplikacja łączy wiele formularzy i dynamicznych ekranów z aparatem, SQLite, systemem plików oraz silnikiem `llama.cpp`. W porównaniu z implementacją całości w Kotlinie ograniczono ilość kodu platformowego, nie rezygnując z możliwości zbudowania zwykłego pakietu APK.

Kosztom tego wyboru jest dodatkowa warstwa pośrednia między JavaScript a Androidem oraz konieczność pilnowania zgodności wersji modułów natywnych. Włączono nową architekturę React Native, obejmującą Fabric i TurboModules [20], ponieważ wymaga jej używana wersja biblioteki `llama.rn` [18]. React Native nie usuwa więc kodu natywnego; porządkuje sposób, w jaki większość logiki aplikacji współpracuje z tym kodem.

3.3.3 Expo

Expo SDK 53 dostarcza spójne moduły dostępu do funkcji platformy i narzędzia budowania [9]. Wybrano je jako warstwę nad React Native, ponieważ projekt potrzebował kilku typowych funkcji urządzenia, które Expo udostępnia przez interfejsy o zgodnych wersjach. Pozwoliło to skupić własny kod na logice dziennika i obsłudze modelu zamiast osobno konfigurować aparat, bazę, system plików i informacje o urządzeniu.

W projekcie wykorzystano:

- `expo-sqlite` do lokalnej bazy danych;
- `expo-camera` do podglądu aparatu i rozpoznawania kodów;
- `expo-file-system` do pobierania i usuwania modeli;
- `expo-device` do odczytu informacji o urządzeniu;

- `expo-build-properties` do konfiguracji części natywnej;
- `expo-keep-awake` do utrzymania aktywnego ekranu podczas pobierania modelu.

Dokumentacja Expo wskazuje, że baza utworzona przez `expo-sqlite` jest trwała pomiędzy ponownymi uruchomieniami aplikacji [10]. Komponent `CameraView` udostępnia natywne wykrywanie kodów w obrazie aparatu [7]. Moduł `expo-device` dostarcza między innymi nazwę modelu, pamięć urządzenia i klasę rocznikową [8].

Expo Go nie mogło być środowiskiem docelowym, ponieważ nie zawiera natywnego modułu RNLLama. Do testów zastosowano natywny wariant debug tworzony poleceniem `npx expo run:android` oraz wygenerowany projekt Android. Expo pełni w tej architekturze rolę zestawu modułów i narzędzi konfiguracyjnych, ale końcowy APK jest pełną aplikacją natywną i nie wymaga uruchomionego klienta Expo ani serwera Metro. Ograniczeniem jest konieczność przebudowania aplikacji po zmianie zależności natywnej.

3.3.4 Zustand i React Navigation

Stan aplikacji został podzielony na trzy magazyny Zustand:

- `profileStore`: inicjalizacja lokalnego profilu;
- `nutritionStore`: dzienne podsumowanie i operacje na posiłkach;
- `aiStore`: wybór, pobieranie, ładowanie i usuwanie modelu.

React Navigation odpowiada za dolny pasek czterech zakładek oraz stos ekranów modalnych. Główne zakładki to dzień, kalendarz, asystent i profil. Formularz posiłku oraz skaner są otwierane nad głównym interfejsem. Taki podział odpowiada częstotliwości wykonywanych operacji i ogranicza głębokość nawigacji.

3.3.5 SQLite

Uzasadnienie wyboru SQLite przedstawiono w punkcie dotyczącym architektury *offline-first*. W implementacji baza jest otwierana przez `expo-sqlite`; włączono klucze obce i tryb Write-Ahead Logging, a dla dat posiłków utworzono indeks [34]. Szczegółowy schemat opisano w sekcji modelu danych.

3.3.6 Open Food Facts

Open Food Facts jest otwartą, współtworzoną bazą produktów spożywczych, zawierającą między innymi nazwy, marki, składniki, alergenów i wartości odżywcze [28]. API pozwala pobierać dane produktu na podstawie kodu kreskowego [27]. Integracja korzysta z wersji 3.6 i odczytuje znormalizowany zestaw wartości `nutrition.aggregated_set`. Ponieważ dane są wprowadzane przez wielu uczestników, aplikacja sprawdza obecność nazwy i energii na 100 g, ale nie może zagwarantować poprawności źródłowego rekordu.

3.3.7 Qwen3, GGUF oraz silnik llama.rn

Qwen3 jest rodziną otwartych modeli językowych obejmującą warianty o różnej liczbie parametrów. Raport techniczny opisuje tryb rozumowania i tryb bez rozbudowanego toku rozumowania oraz wsparcie wielu języków [33]. Model Qwen3 1.7B posiada 1,7 miliarda parametrów i jest udostępniony na licencji Apache 2.0 [31]. Wariant 4B oferuje większą pojemność kosztem pamięci i szybkości [32].

Modele zastosowane w projekcie mają format GGUF i kwantyzację Q4_K_M. Plik wariantu 1.7B pochodzi z repozytorium QuantFactory i jest kwantyzacją oficjalnego modelu Qwen3-1.7B [30]. Plik wariantu 4B pochodzi z oficjalnego repozytorium Qwen [32]. GGUF jest binarnym formatem zaprojektowanym do szybkiego ładowania modeli w narzędziach opartych na GGML; może przechowywać model wraz z metadanymi w jednym pliku [12]. Kwantyzacja zmniejsza rozmiar wag i zapotrzebowanie na pamięć kosztem części precyzji.

Biblioteka llama.rn udostępnia wiązanie React Native do llama.cpp. Obsługuje inicjalizację kontekstu, generowanie tokenów, zatrzymanie odpowiedzi i zwolnienie zasobów [18]. W aplikacji obliczenia są wykonywane na CPU z czterema wątkami i kontekstem 4096 tokenów. Backendy OpenCL i Hexagon nie są deklarowane przez aplikację. Taki wariant wybrano dla przewidywalności na różnych układach Android oraz zgodności z użytymi plikami Q4_K_M, kosztem mniejszej szybkości niż w rozwiązaniu dostrojonym do jednego procesora.

Nazwa biblioteki może prowadzić do nieporozumienia. W projekcie nie następuje zamiana modelu Qwen na model Llama. Qwen3 jest właściwym modelem, czyli zbiorem wag i architekturą odpowiedzialną za generowanie tekstu. llama.cpp jest natomiast silnikiem wykonawczym, który potrafi uruchamiać różne modele zapisane w obsługiwanym formacie GGUF. Biblioteka llama.rn udostępnia ten silnik aplikacji React Native. Słowo „llama” w nazwie narzędzia opisuje więc pochodzenie projektu, a nie model wybrany przez użytkownika.

3.3.7.1 Wiedza modelu i brak bazy wektorowej

Plik modelu Qwen3 nie jest relacyjną bazą produktów ani tabelą gotowych odpowiedzi. Wiedza nabyta podczas uczenia jest zapisana pośrednio w parametrach sieci neuronowej. Nie można jej przeglądać jak rekordów SQLite ani wskazać jednego wiersza, z którego model utworzył odpowiedź. Pozwala to swobodnie generować tekst, ale nie gwarantuje poprawności każdego zdania.

W obecnej wersji historia posiłków nie jest wektoryzowana. Aplikacja pobiera z SQLite wszystkie wpisy wybranego miesiąca i oblicza ich sumy za pomocą zwykłych operacji na danych liczbowych. Następnie tworzy tekst zawierający podsumowanie miesiąca oraz szczegóły posiłków mieszczące się w limicie kontekstu modelu. Tekst ten jest dołączany do komunikatu systemowego.

Takie podejście wybrano dlatego, że datę, kalorie i makroskładniki można odczytać dokładnie za pomocą SQL. Wyszukiwanie wektorowe opiera się na podobieństwie znaczenia tekstu, dlatego nie zwiększyłoby dokładności sumowania wartości liczbowych ani wyboru wpisów z konkretnego miesiąca. Dodałoby natomiast kolejny model, indeks i obciążenie pamięci urzą-

dzenia.

Rozwiązanie ma również ograniczenie. Miesięczne sumy obejmują wszystkie wpisy, lecz przy bardzo długiej historii część opisów pojedynczych produktów może nie zmieścić się w kontekście. Ponadto Qwen3 może błędnie zinterpretować nawet poprawnie przekazane dane. Z tego powodu obliczenia pozostają po stronie aplikacji, a odpowiedź modelu ma charakter pomocniczy.

Baza wektorowa mogłaby być przydatna w przyszłości do przeszukiwania dużego zbioru zatwierdzonych artykułów i zaleceń według znaczenia pytania. Nie jest jednak potrzebna do odczytu uporządkowanego dziennika posiłków i nie stanowi części obecnej implementacji.

Tabela 10: Źródła danych używane przez aplikację i lokalnego asystenta

Element	Sposób przechowywania	Rola
Profil i posiłki	SQLite	Dane użytkownika i historia przekazywana do kontekstu.
Zeskanowane produkty	SQLite, rekord JSON	Pamięć podręczna danych Open Food Facts.
Katalog podstawowy	Moduł JavaScript	Wyszukiwanie, porcje i lokalne rekomendacje.
Rozmowa i ustawienia	AsyncStorage	Historia czatu, wybrany model, tryb AI i język.
Wiedza Qwen3	Parametry w pliku GGUF	Generowanie języka na podstawie uczenia modelu.
Baza wektorowa	Brak	Aplikacja nie wykonuje wyszukiwania embeddingów ani RAG.

Źródło: opracowanie własne.

3.3.7.2 Powstawanie odpowiedzi

Przed generowaniem tekst pytania, instrukcja systemowa, wybrane wiadomości rozmowy i kontekst z SQLite są dzielone przez tokenizer na tokeny. Model przetwarza całą sekwencję i wielokrotnie przewiduje następny token. Każdy wygenerowany token staje się częścią wejścia dla kolejnego kroku, aż do osiągnięcia znacznika końca, limitu 420 tokenów albo przerwania przez użytkownika. Interfejs odbiera tokeny na bieżąco i pokazuje je jako strumieniowaną odpowiedź. Model nie wykonuje zapytania do Internetu i nie przeszukuje ukrytej bazy danych.

3.4. Interfejs i implementacja aplikacji

3.4.1 Inicjalizacja

Komponent główny uruchamia inicjalizację profilu i ustawień AI. W czasie odczytu wyświetlany jest ekran startowy. Moduł `data/database.js` udostępnia operacje `getProfile`, `addMeal` i `getDailySummary`. Ekran nie zna poleceń SQL ani szczegółów migracji schematu.

Aplikacja nie ma logowania, tokenu dostępowego ani konta serwerowego. `profileStore` inicjalizuje lokalną bazę i odczytuje jeden profil urządzenia. Usunięcie dawnej warstwy zgodności z klientem HTTP uprościło przepływ startowy i sprawiło, że nazwy modułów odpowiadają ich rzeczywistej roli.

3.4.2 Ekran dzienny

Ekran dzienny pobiera podsumowanie wskazanej daty. Użytkownik może przechodzić między dniami. Dla bieżącej daty widoczny jest główny przycisk rejestrowania. Posiłki są grupowane jako śniadanie, obiad, kolacja i przekąski.

Pierścień kalorii ogranicza wizualny postęp do 100%, ale po przekroczeniu celu zmienia kolor i pokazuje wartość nadwyżki. Paski makroskładników porównują bieżące sumy z gramami wyznaczonymi z aktualnego profilu.

3.4.3 Formularz dodawania posiłku

Formularz łączy cztery sposoby wprowadzania:

1. wybór propozycji dopasowanej do celu i bieżącego spożycia;
2. wybór pojedynczego produktu z katalogu;
3. zeskanowanie kodu produktu;
4. pełny wpis ręczny.

Wszystkie metody prowadzą do tego samego edytora. Wybrane produkty trafiają do listy roboczej, ale nie są zapisywane automatycznie. Każdą pozycję można rozwinąć, sprawdzić nazwę, gramaturę, źródło oraz wartości energii, białka, węglowodanów i tłuszczu na 100 g. Zmiana gramatury natychmiast przelicza wartości dla porcji. Zatwierdzenie listy tworzy oddzielny rekord dla każdego produktu.

W trybie ręcznym kalorie początkowo aktualizują się na podstawie makroskładników. Jeżeli użytkownik bezpośrednio edytuje pole kalorii, automatyczne nadpisywanie zostaje wyłączone.

Na ekranie dnia każdy zapisany produkt pokazuje gramaturę, kalorie i trzy makroskładniki. Dotknięcie wiersza otwiera ten sam formularz w trybie edycji. Użytkownik może zmienić porcję, wartości na 100 g, nazwę lub porę posiłku. Usuwanie jest oddzielną akcją i wymaga potwierdzenia.

3.4.4 Skaner

Ekran skanera sprawdza pozwolenie aparatu. Jeżeli nie zostało przyznane, wyświetla wyjaśnienie i przycisk zgody. Po zeskanowaniu czasowo wyłącza callback, aby jeden kod nie utworzył kilku równoległych zapytań. Podczas wyszukiwania widoczny jest wskaźnik. Błąd można zamknąć i ponowić skanowanie. Poprawnie odczytany produkt nie jest jedynie sugestią. Skaner wraca do formularza, dodaje produkt do listy i automatycznie rozwija jego pola, aby użytkownik mógł porównać dane z etykietą.

3.4.5 Kalendarz

Kalendarz pobiera agregację dla roku i miesiąca. Zapytanie SQL grupuje rekordy według daty i sumuje energię. Dni bez wpisów otrzymują wartość zero. Dzień bieżący ma wyróżnione obramowanie, dni przyszłe są nieaktywne, a wynik przekraczający 110% celu otrzymuje kolor ostrzegawczy. Dotknięcie dnia otwiera jego szczegóły na ekranie głównym.

3.4.6 Profil

Profil zawiera pięć obszarów:

- wybór języka polskiego albo angielskiego;
- parametry ciała, aktywność, cel i tryb obliczeń;
- alergie i produkty unikane;
- tryb AI, wybór modelu, stan pobrania i możliwości urządzenia;
- dane i prywatność, w tym pełne wyczyszczenie stanu aplikacji.

Wynik celu jest prezentowany na żywo. W trybie ręcznym widoczna jest suma procentów i błąd, gdy nie wynosi 100%. Po zapisaniu aplikacja aktualizuje stan globalny i odświeża podsumowanie. Pola liczbowe przyjmują zarówno przecinek, jak i kropkę dziesiętną, co jest ważne na polskiej klawiaturze systemowej.

Przy pierwszym uruchomieniu wybierany jest polski, jeżeli ustawienia regionalne telefonu wskazują język polski, w przeciwnym razie angielski. Zmiana jest zapisywana w AsyncStorage i od razu aktualizuje główne ekrany, skaner, komunikaty błędów oraz instrukcję językową wysyłaną do modelu. Pole imienia jest opcjonalne. Gdy pozostaje puste, ekran dzienny pokazuje neutralne powitanie.

Sekcja danych i prywatności wyjaśnia, że zwykła instalacja nowszego APK jest aktualizacją i zachowuje dane. Przycisk usunięcia całego stanu ma styl ostrzegawczy i wymaga dodatkowego potwierdzenia. Po jego użyciu aplikacja usuwa bazę, rozmowę, ustawienia i wszystkie warianty modelu, a następnie uruchamia się ponownie.

3.4.7 Konfiguracja AI

Konfiguracja AI znajduje się wyłącznie w profilu. Sekcja przedstawia dwa modele wraz z rozmiarem, minimalną pamięcią i krótkim opisem. Pole rekomendacji zawiera model telefonu, przybliżoną ilość RAM i klasę urządzenia, jeśli system udostępnił te dane.

Pobieranie nie jest obowiązkowe. Przed rozpoczęciem aplikacja informuje, że wymagane jest połączenie z Internetem i pozostawienie programu otwartego do osiągnięcia 100%. Na czas pobierania ekran nie jest automatycznie wygaszany. Obecna wersja nie używa Android DownloadManager ani WorkManager, dlatego nie gwarantuje kontynuacji po ręcznym zamknięciu lub uśpieniu procesu. Użytkownik może anulować aktywne pobieranie, przełączać wariant i usuwać wybrany plik. Ustawienia są przechowywane w AsyncStorage.

3.4.8 Rozmowa

Historia rozmowy jest zapisywana jako JSON. Ekran odczytuje ją przy otwarciu i aktualizuje po zmianie wiadomości. Dla ochrony pamięci przechowywanych jest maksymalnie 40 ostatnich komunikatów. Nagłówek pokazuje aktywny wariant oraz stan ładowania.

Jeżeli model nie został jeszcze załadowany, pierwsze użycie rozpoczyna inicjalizację. W trakcie generowania pole tekstowe jest zablokowane, a użytkownik może zatrzymać odpowiedź. Czyszczenie rozmowy wymaga potwierdzenia.

3.4.9 Budowanie pakietu APK

Wersję instalacyjną przygotowano jako samodzielny pakiet APK dla systemu Android. Proces budowania obejmuje kontrolę testów, sprawdzenie konfiguracji Expo oraz kompilację natywnej części projektu. Główna sekwencja poleceń ma postać:

Listing 3.1: Polecenie budowania wersji instalacyjnej

```
1 npm.cmd test
2 npx.cmd expo config --type public
3 gradlew.bat app:lintRelease app:testReleaseUnitTest app:assembleRelease \
4   --configure-on-demand \
5   --no-build-cache \
6   -PreactNativeArchitectures=arm64-v8a,x86_64
```

Gradle osadza pakiet JavaScript i biblioteki natywne. W badanej wersji testowej plik APK miał rozmiar około 156 MB. Modele AI nie są umieszczane w APK, dlatego plik instalacyjny pozostaje znacznie mniejszy niż pełny zestaw z modelami. Użytkownik osobno decyduje, czy pobrać model 1,7B, model 4B, oba pliki czy żaden.

Wersja przygotowana do testów jest podpisana kluczem deweloperskim. Publikacja w Google Play wymaga utworzenia chronionego klucza produkcyjnego, zmiany konfiguracji podpisu i przygotowania pakietu Android App Bundle.

3.5. Algorytmy i procesy działania aplikacji

3.5.1 Wyznaczanie celu energetycznego i tryb pracy

Funkcja `calculateProfileTargets` przyjmuje sześć parametrów. Przed obliczeniem sprawdza zakresy: masa 30–300 kg, wzrost 120–230 cm i wiek 16–100 lat. Brak płci albo wartość poza zakresem powoduje zwrócenie `null`. Fragment implementacji przedstawiono na listingu 3.2.

Listing 3.2: Obliczenie celu energetycznego

```
1 const genderOffset = gender === 'male' ? 5 : -161;  
2 const bmr = 10 * weight + 6.25 * height - 5 * years + genderOffset;  
3 const calories = Math.round(  
4   (bmr * activity.multiplier * goalConfig.calorieFactor) / 10  
5 ) * 10;
```

Dla mężczyzny o masie 80 kg, wzroście 180 cm i wieku 20 lat wartość BMR wynosi:

$$10 \cdot 80 + 6,25 \cdot 180 - 5 \cdot 20 + 5 = 1830 \text{ kcal.} \quad (3.1)$$

Przy umiarkowanej aktywności i celu utrzymania otrzymuje się $1830 \cdot 1,55 = 2836,5$ kcal, czyli po zaokrągleniu 2840 kcal.

3.5.1.1 Tryb automatyczny i ręczny

W trybie automatycznym każda zmiana parametrów ciała, aktywności lub celu ponownie wywołuje obliczenie. Wynik aktualizuje pola kalorii i procentów. W trybie ręcznym ten efekt jest wyłączony. Przed zapisem wykonywane są następujące sprawdzenia:

- cel energetyczny mieści się w zakresie 800–10000 kcal;
- każdy udział makroskładnika mieści się w zakresie 0–100%;
- suma udziałów jest równa 100% z tolerancją 0,01.

Po zapisie cele są natychmiast przekazywane do ekranu głównego, a podsumowanie bieżącego dnia jest pobierane ponownie. Nowe wartości są widoczne bez restartowania aplikacji.

3.5.2 Rejestrowanie i przetwarzanie żywności

3.5.2.1 Przeliczanie produktu na porcję

Każdy rekord katalogu zawiera wartości na 100 g. Dla gramatury G skala wynosi:

$$s = \frac{G}{100}. \quad (3.2)$$

Kalorie są zaokrąglane do liczby całkowitej, a makroskładniki do jednego miejsca po przecinku. Dla 200 g piersi kurczaka o wartości 165 kcal i 31 g białka na 100 g wynik wynosi 330 kcal oraz 62 g białka.

3.5.2.2 Lokalny katalog produktów

Katalog zawiera 90 popularnych produktów. Obejmuje mięso, ryby, nabiał, produkty roślinne, zboża, kasze, owoce, warzywa, rośliny strączkowe, orzechy, nasiona i podstawowe dodatki. Wartości mają charakter referencyjny dla ogólnych produktów, a punktem odniesienia są publiczne dane składu żywności, takie jak USDA FoodData Central [35]. Nie opisują konkretnej marki ani partii produktu. Dla żywności paczkowanej pierwszeństwo ma etykieta producenta, a aplikacja udostępnia skaner i ręczną korektę gramatury. Wyszukiwanie rozpoznaje aliasy angielskie oraz najczęstsze nazwy polskie. Każdy rekord ma:

- stabilny klucz;
- nazwę prezentowaną;
- listę aliasów używanych podczas wyszukiwania;
- tagi, w tym potencjalne alergeny;
- domyślną gramaturę;
- wartości na 100 g.

Wyszukiwanie rozpoczyna się po wpisaniu dwóch znaków. Najpierw pokazywane są dopasowania dokładne, potem nazwy zaczynające się od zapytania, a na końcu wyniki zawierające wpisany fragment. Lista ma najwyżej osiem pozycji, więc nie przysłania pozostałych elementów formularza.

3.5.2.3 Rekomendowanie posiłków

Rekomendacje nie są generowane przez model językowy. Aplikacja korzysta z gotowych zestawów opartych na danych katalogowych, a następnie nadaje im wynik punktowy. Wejściami algorytmu są: typ posiłku, cel sylwetkowy, cel kalorii i makroskładników, dotychczasowe spożycie oraz lista produktów wykluczonych.

Na początku odrzucane są wszystkie zestawy zawierające składnik zgodny z alergią lub listą unikania. Dla pozostałych aplikacja sumuje energię i makroskładniki. Następnie oblicza pozostałą energię dnia i ocenia strukturę zestawu. Sprawdzane są: obecność źródła białka, produktu skrobiowego, pełnego ziarna, źródła tłuszczów nienasyconych oraz łączna gramatura owoców i warzyw.

Kryteria te wynikają z przeglądu zaleceń naukowych, ale nie stanowią próby układania diety leczniczej. Stanowisko ISSN dla zdrowych osób ćwiczących wskazuje orientacyjnie 1,4–2,0 g białka/kg masy ciała na dobę oraz ogólną porcję około 0,25 g/kg albo 20–40 g wysokiej jakości

białka [15]. W aplikacji zakres pojedynczego głównego posiłku jest dopasowywany do masy ciała, a jego górna granica została poszerzona do 50 g, ponieważ gotowe danie mieszane może przekraczać zakres samego źródła białka. Nie jest to limit fizjologicznego wchłaniania.

ISSN podkreśla również znaczenie całkowitej podaży dobowej i względnie równomiernego rozłożenia porcji w ciągu dnia [16]. Badanie Mamerow i współautorów wykazało większą całodobową syntezę białek mięśniowych przy bardziej równomiernym rozkładzie białka między posiłkami, choć dotyczyło małej grupy zdrowych dorosłych i nie uzasadnia jednego obowiązkowego jadłospisu [19]. Z kolei WHO zaleca różnorodną dietę opartą głównie na produktach mało przetworzonych, z udziałem pełnych ziaren, warzyw, owoców i roślin strączkowych [36]. Dlatego algorytm premiuje strukturę posiłku, a nie tylko możliwie dużą liczbę gramów białka.

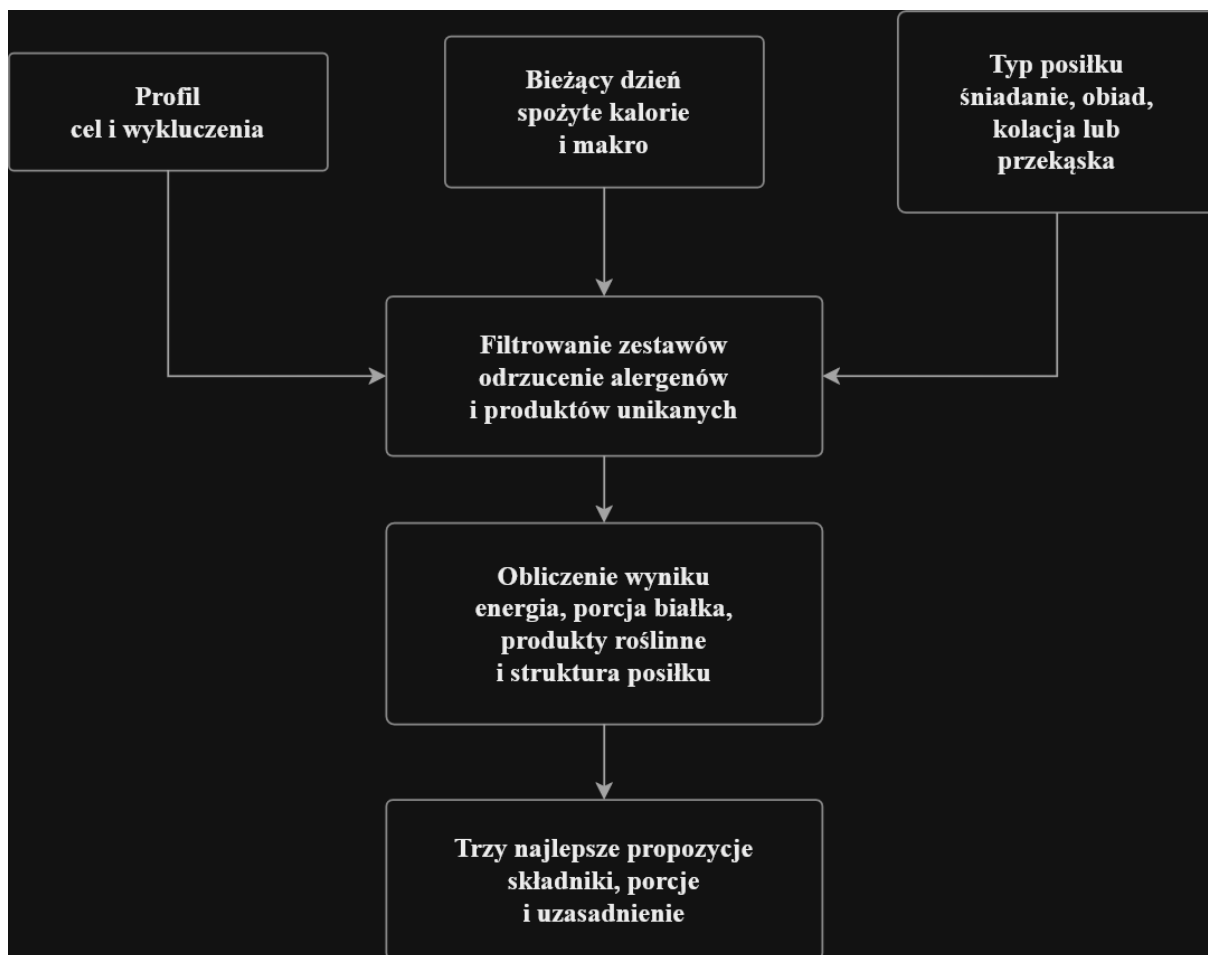
Przypisanie zestawów do pory dnia jest decyzją użytkową. Owsianka ze skyrem i owocami jest wygodnym przykładem śniadania, a kurczak z ryżem i brokułem przykładem obiadu, lecz badania nie wskazują, że te konkretne produkty muszą być jedzone o danej godzinie. Użytkownik może wybrać inny zestaw, zmienić gramaturę albo dodać własny posiłek. Sposób punktowania zależy od celu i został przedstawiony w tabeli 11.

Tabela 11: Priorytety algorytmu rekomendowania posiłków

Cel	Najważniejsze kryteria kolejności
Redukcja	Dopasowanie do pozostałej energii, porcja białka, wysoki stosunek białka do kalorii, udział owoców lub warzyw i pełnych ziaren.
Utrzymanie	Równowaga między brakującą energią, porcją białka i pełną strukturą posiłku, bez faworyzowania bardzo dużych porcji.
Kontrolowany przyrost	Dopasowanie energii, porcja białka, źródło węglowodanów skrobiowych oraz owoce lub warzywa.
Budowa masy	Największa waga dopasowanej porcji białka, kompletnej struktury posiłku i źródła węglowodanów, przy zachowaniu rozsądnej kaloryczności.

Źródło: opracowanie własne.

Trzy zestawy z najwyższym wynikiem są prezentowane wraz z nazwami składników, ich gramaturą, kalorycznością, ilością białka i krótkim uzasadnieniem. Po wybraniu zestawu użytkownik może zmienić porcje, usunąć składnik, a następnie zapisać wszystkie produkty jako osobne wpisy jednego posiłku. Przykładowa propozycja poranna zawiera płatki owsiane, skyr, maliny i nasiona chia. Przykładowy obiad składa się z piersi kurczaka, ryżu basmati i brokułów. Interfejs pokazuje także gramaturę owoców lub warzyw oraz informuje, że na pozycję wpływają pozostała energia i struktura posiłku. Cały proces odbywa się lokalnie i daje ten sam wynik dla tych samych danych. Schemat przedstawiono na rysunku 10.



Rysunek 10: Przebieg lokalnego rekomendowania posiłków

Źródło: opracowanie własne.

3.5.2.4 Wspólny edytor produktów

Wyszukiwarka, rekomendacje, skaner i wpis ręczny tworzą ten sam rodzaj pozycji roboczej. Zawiera ona nazwę produktu, gramaturę, wartości na 100 g, źródło danych oraz opcjonalny klucz katalogu lub kod kreskowy. Dzięki temu logika przeliczenia i walidacji nie zależy od sposobu znalezienia produktu.

Świadomie zrezygnowano z automatycznego dzielenia dowolnego zdania na składniki. Krótki tekst mógł być niejednoznaczny, pomijać sposób przygotowania albo błędnie przypisywać gramaturę. Jawne dodawanie każdego produktu wymaga jednego dodatkowego dotknięcia, ale pozwala użytkownikowi sprawdzić dane przed zapisem i upraszcza późniejszą korektę dziennika.

3.5.2.5 Filtrowanie alergii i produktów unikanych

Tekst wpisany w profilu jest dzielony według przecinków, średników i nowych linii. Każdy termin jest porównywany z nazwą, aliasami i tagami produktu. Pasujące produkty są ukrywane w wyszukiwarce, gotowych pomysłach i podpowiedziach. Zeskanowany produkt jest sprawdzany przed dodaniem.

Mechanizm ma charakter pomocniczy. Proste dopasowanie tekstowe może nie rozpoznać wszystkich synonimów, śladowych ilości lub złożonych składników. Nie powinien być jedyną ochroną osoby z ciężką alergią. Model językowy otrzymuje oddzielną instrukcję zakazu polecania alergenu, ale jest to druga warstwa ostrożności, a nie gwarancja medyczna.

3.5.2.6 Obsługa kodu kreskowego

Skaner rozpoznaje typy EAN-13, EAN-8, UPC-A i UPC-E. Po usunięciu znaków innych niż cyfry aplikacja sprawdza, czy kod ma od 8 do 14 znaków. Następnie:

1. odczyt z tabeli `scanned_products`;
2. w razie braku wysłanie zapytania do API v3.6;
3. pobranie tylko wymaganych pól;
4. odczyt zestawu `nutrition.aggregated_set`;
5. sprawdzenie nazwy, energii i zakresów makroskładników na 100 g;
6. normalizacja pól liczbowych oraz odrzucenie wartości nierealnych;
7. zapis całego znormalizowanego obiektu jako JSON;
8. przekazanie produktu do rozwiniętego edytora gramatury i wartości na 100 g.

Zapytanie ma limit 12 sekund i pobiera tylko potrzebne pola. Pamięć podręczna przyspiesza kolejne skanowanie tego samego produktu i pozwala użyć go później bez sieci. Uszkodzony rekord JSON jest usuwany zamiast powodować awarię formularza.

3.5.3 Lokalny asystent AI

3.5.3.1 Dobór lokalnego modelu

Aplikacja oferuje dwa modele przedstawione w tabeli 12.

Tabela 12: Lokalne modele dostępne w aplikacji

Nazwa w aplikacji	Model	Rozmiar	Zalecana RAM	Opis
Testowy	Qwen3 1.7B Q4_K_M	1,28 GB	co najmniej 4 GB	Wariant dla słabszych telefonów; może źle rozumieć pytania i udzielać niewiarygodnych porad
Zalecany jakościowo	Qwen3 4B Q4_K_M	2,50 GB	co najmniej 8 GB	Preferowany do rozmowy na obsługiwany urządzeniu; zwykle odpowiada lepiej, lecz nadal nie gwarantuje poprawności

Źródło: opracowanie własne.

Moduł urządzenia próbuje odczytać całkowitą pamięć, maksymalną stertę, klasę rocznikową i informację, czy program działa na urządzeniu fizycznym. W interfejsie rozdzielono ocenę jakości od dopasowania sprzętowego. Model 4B jest oznaczony jako zalecany jakościowo, o ile urządzenie może go uruchomić. Model 1.7B ma oznaczenie testowe. Jeżeli telefon ma mniej niż 8 GB RAM, aplikacja może wskazać 1.7B jako wariant dopasowany do urządzenia, ale nie przedstawia go jako równoważnego jakościowo. Wariant 4B nie może zostać pobrany ani załadowany, gdy wykryta pamięć jest mniejsza niż 6 GB. Próg wprowadzono po teście na urządzeniu z 4 GB RAM, na którym proces Android został zakończony podczas ładowania większego modelu. Zablokowany model pozostaje widoczny w ustawieniach, dzięki czemu użytkownik rozumie przyczynę niedostępności i może usunąć wcześniej pobrany plik.

3.5.3.2 Pobieranie i weryfikacja modelu

Przed pobraniem sprawdzana jest wolna przestrzeń. Wymagana wartość to rozmiar modelu powiększony o 500 MB marginesu. Istniejący niepełny plik docelowy jest usuwany. Po zakończeniu aplikacja sprawdza kod odpowiedzi i wielkość pliku. Plik jest uznawany za poprawny, jeśli ma co najmniej 98% oczekiwanego rozmiaru. Takie kryterium chroni przed próbą uruchomienia wyrażnie niekompletnego pobrania. Biblioteka natywna zwraca postęp ładowania w skali od 0 do 100, natomiast część interfejsu używa wartości od 0 do 1. Funkcja normalizująca rozpoznaje obie skale, a następnie ogranicza wynik do przedziału od 0 do 100% widocznego

dla użytkownika.

Plik jest zapisywany w podkatalogu `models` prywatnego katalogu dokumentów aplikacji. Przed pobraniem użytkownik potwierdza komunikat o wymaganym Internecie i pozostawieniu programu otwartego. Moduł `expo-keep-awake` zapobiega automatycznemu wygaszeniu ekranu do zakończenia lub błędu operacji. Użytkownik może anulować transfer. Aplikacja wstrzymuje aktywne pobieranie i usuwa niepełny plik, aby nie uznać go później za gotowy model.

3.5.3.3 Generowanie odpowiedzi

Model jest inicjalizowany z mapowaniem pamięci, bez blokowania całego pliku w RAM, z czterema wątkami CPU i kontekstem 4096 tokenów. Ładowanie ma limit pięciu minut. Przed każdym generowaniem czyszczona jest pamięć podręczna kontekstu.

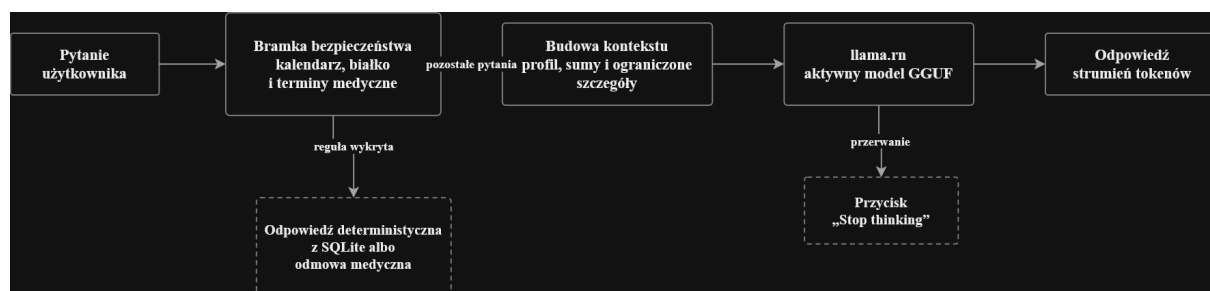
Do modelu trafia komunikat systemowy, profil użytkownika, alergie, produkty unikane, suma całego wybranego miesiąca, sumy poszczególnych dni i szczegóły posiłków mieszczące się w limicie 9000 znaków. Jeżeli szczegółów jest więcej, model otrzymuje jawny komunikat o ich skróceniu, natomiast miesięczne sumy nadal obejmują wszystkie rekordy. Bieżący miesiąc jest wybierany domyślnie. Parser pytania rozpoznaje nazwę miesiąca po angielsku lub polsku, określenie poprzedniego miesiąca oraz zapis `RRRR-MM`. Przy dłuższym kontekście liczba dołączonych wiadomości rozmowy jest również zmniejszana, aby pozostawić miejsce w oknie 4096 tokenów.

Pytania typu „zobacz mój kalendarz i moje kalorie” nie są przekazywane do modelu. Aplikacja tworzy deterministyczne podsumowanie bezpośrednio z wyniku SQLite: podaje sumę miesiąca, liczbę dni z wpisami, średnią na zapisany dzień, różnicę względem celu i wartości dla każdego dnia. Prośba o ocenę sposobu żywienia nadal trafia do modelu, ale wraz z opisanym wyżej zrzutem danych. Pytania o konkretny dzień rozpoznają zwroty „dzisiaj”, „wczoraj”, datę `RRRR-MM-DD` oraz zapis `DD.MM.RRRR`. Odpowiedź podaje dokładną sumę tego dnia i nazwy zapisanych posiłków. Przy podsumowaniu miesiąca aplikacja może dołączyć maksymalnie 30 szczegółowych wpisów, a o pominięciu starszych pozycji informuje wprost. Rozdzielenie odpowiedzi liczbowej od komentarza językowego usuwa przypadek, w którym model twierdził, że nie ma dostępu do kalendarza.

Komunikat systemowy zawiera również regułę języka. Po wybraniu polskiego model ma odpowiadać naturalnie po polsku nawet wtedy, gdy część nazw produktów lub starszych wiadomości jest angielska. Deterministyczne odpowiedzi ochronne, na przykład podsumowanie kalendarza, zakres białka i odmowa porady medycznej, mają osobne wersje polskie i angielskie, dlatego nie zależą od tego, czy model poprawnie wykona instrukcję językową.

Maksymalna odpowiedź ma 420 tokenów. Parametry próbkowania odpowiadają zaleceniom trybu bez rozbudowanego rozumowania Qwen3: temperatura 0,7, `top_k=20` i `top_p=0.8` [31]. Oprócz polecenia `/no_think` do biblioteki przekazywane są parametry `enable_thinking=false` i `reasoning_format=none`. Jako zabezpieczenie interfejs usuwa również kompletne, otwarte i częściowo przesłane znaczniki `think` oraz `analysis`.

Callback tokenów aktualizuje tekst w interfejsie. Przycisk zatrzymania wywołuje `stopCompletion`. Przełączenie modelu zwalnia istniejący kontekst przed załadowaniem drugiego pliku. Przebieg obsługi pytania pokazano na rysunku 11.



Rysunek 11: Przepływ obsługi pytania do lokalnego asystenta

Źródło: opracowanie własne.

3.5.3.4 Ochrona przed błędami modelu

Przed wywołaniem LLM tekst analizują deterministyczne funkcje ochronne i funkcje historii. Jeżeli użytkownik pyta o ilość białka, wartość jest liczona z masy podanej w pytaniu lub profilu. Jeżeli pyta o kalendarz, kalorie lub makroskładniki, odpowiedź jest budowana z danych SQLite. Pytania zawierające wybrane terminy medyczne, na przykład chorobę nerek, ciążę albo reakcję alergiczną, nie są przekazywane do modelu. Użytkownik otrzymuje komunikat o konieczności kontaktu ze specjalistą.

Polecenie systemowe zakazuje:

- obliczania celów kalorycznych, makro, BMI i dawek leków;
- wymyślania wartości żywieniowych;
- polecania alergii i produktów oznaczonych do unikania;
- porad dotyczących sterydów anabolicznych, leków i leczenia urazów;
- tworzenia twierdzeń medycznych.

Historia posiłków przekazana z SQLite jest oznaczona jako jedyne źródło prawdy o tym, co użytkownik zjadł. Jeżeli w wybranym miesiącu nie ma wpisów, model ma poinformować o braku danych, zamiast uzupełniać je na podstawie domysłów.

Ograniczenia polecenia nie są formalną gwarancją. Mały model może zignorować instrukcję. Z tego powodu jego rola jest celowo ograniczona, a krytyczne dane pozostają poza generowaniem.

4. WERYFIKACJA I OCENA ROZWIĄZANIA

4.1. Realizacja założeń prywatności

Motywację prywatności przedstawioną we wstępie przełożono na konkretne decyzje: brak obowiązkowego konta, lokalną bazę, model uruchamiany na urządzeniu, ograniczony kontekst rozmowy i brak chmurowego klucza API. Odpowiada to idei ochrony danych w fazie projektowania i ochrony domyślnej opisanej w art. 25 RODO [29]. Poniższe punkty opisują zakres tej ochrony oraz jej ograniczenia.

4.1.1 Minimalizacja danych

Profil zawiera dane mogące ujawniać informacje o budowie ciała, zwyczajach i ograniczeniach żywieniowych. Artykuł 5 RODO wskazuje między innymi zasadę minimalizacji, zgodnie z którą dane powinny być adekwatne i ograniczone do tego, co niezbędne dla celu przetwarzania [29].

Projekt ogranicza zakres danych przez:

- brak obowiązkowego konta, adresu e-mail i danych płatniczych;
- brak centralnego serwera z historią posiłków;
- przechowywanie profilu i rozmowy na urządzeniu;
- przekazywanie do modelu tylko bieżącego kontekstu potrzebnego do odpowiedzi;
- pobieranie z Open Food Facts wyłącznie pól produktu wymaganych przez aplikację;
- możliwość działania z całkowicie wyłączonym AI.

4.1.2 Lokalizacja i przepływ danych

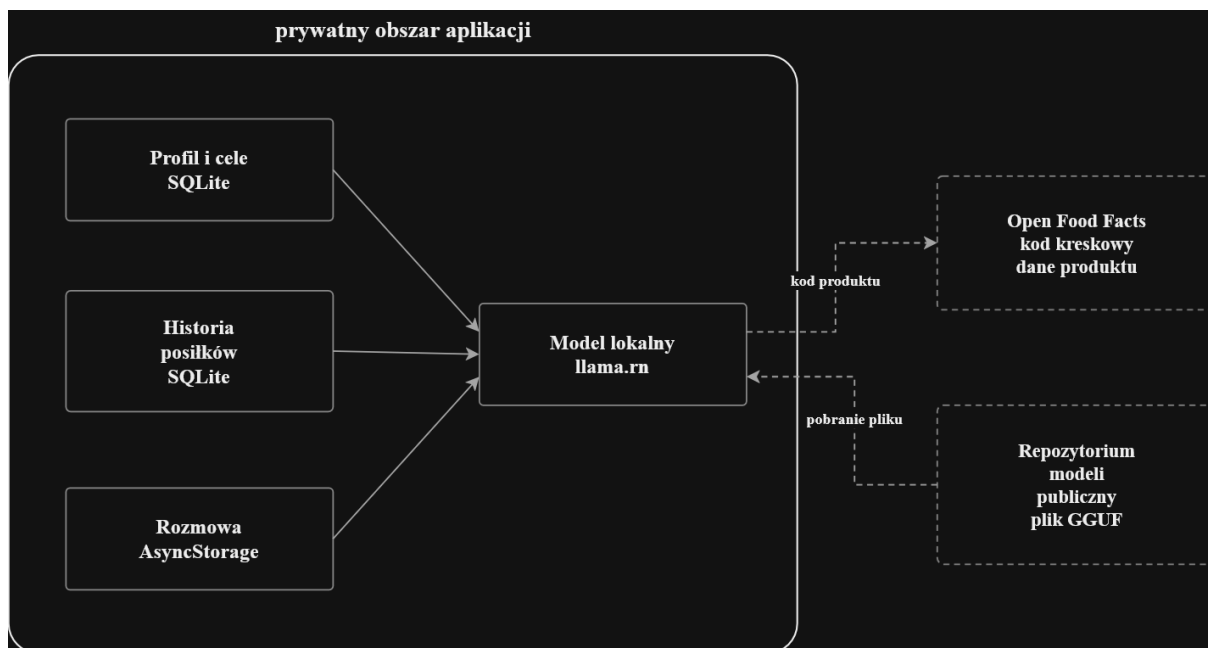
Architektura nie eliminuje wszystkich połączeń sieciowych, ale ogranicza je do funkcji, które rzeczywiście wymagają zasobu zewnętrznego. Tabela 13 rozróżnia dane trwałe, dane przekazywane do lokalnego modelu i informacje wysyłane poza urządzenie.

Tabela 13: Lokalizacja i przepływ danych w aplikacji

Rodzaj danych	Miejsce przechowywania	Transmisja poza urządzenie
Profil i cele	Tabela <code>profile</code> w SQLite.	Brak transmisji do backendu autora; wybrane pola są używane lokalnie do obliczeń i tworzenia kontekstu modelu.
Historia posiłków	Tabela <code>meals</code> w SQLite.	Brak transmisji poza urządzenie; wpisy wybranego miesiąca są przekazywane wyłącznie do modelu działającego na telefonie.
Zeskanowany kod	Po pobraniu produkt jest zapisywany w tabeli <code>scanned_products</code> .	Sam kod jest wysyłany do Open Food Facts tylko wtedy, gdy produktu nie ma w lokalnej pamięci. Obraz z aparatu nie jest wysyłany.
Historia rozmowy	Dokument JSON w <code>AsyncStorage</code> .	Wiadomości są przetwarzane przez lokalny silnik; aplikacja nie przesyła ich do chmurowego dostawcy modelu.
Ustawienia AI	<code>AsyncStorage</code> .	Brak transmisji ustawień; wybór określa jedynie lokalny plik modelu.
Plik modelu	Prywatny katalog dokumentów aplikacji.	Pobierany z publicznego repozytorium. Dostawca pliku i operator sieci mogą zaobserwować typowe metadane połączenia, lecz nie otrzymują historii żywieniowej. Standardowe odinstalowanie usuwa plik razem z danymi aplikacji.

Źródło: opracowanie własne.

Granice przetwarzania przedstawiono dodatkowo na rysunku 12. Profil, historia i treść rozmowy pozostają wewnątrz obszaru aplikacji. Na zewnątrz wychodzi jedynie kod kreskowy podczas wyszukania produktu, natomiast repozytorium modelu przekazuje do telefonu publiczny plik GGUF.



Rysunek 12: Granica prywatności i komunikacja z usługami zewnętrznymi

Źródło: opracowanie własne.

Po pobraniu modelu rozmowa może odbywać się bez Internetu. Open Food Facts jest potrzebne tylko dla nieznanego kodu kreskowego, natomiast lokalny katalog produktów, obliczenia, profil, kalendarz i wcześniej zapisane produkty nie wymagają sieci. Takie rozdzielenie pozwala użytkownikowi przewidzieć, które działanie powoduje komunikację z usługą zewnętrzną.

4.1.3 Uprawnienia

Jedynym szczególnym uprawnieniem używanym przez interfejs jest aparat. Prośba pojawia się dopiero po otwarciu skanera. Obraz nie jest zapisywany ani wysyłany przez aplikację; komponent natywny odczytuje kod z podglądu. Zapytanie sieciowe zawiera kod kreskowy, a nie zdjęcie.

Końcowy manifest usuwa uprawnienia do nagrywania dźwięku, nakładania okien systemowych i dostępu do pamięci zewnętrznej. Pozostają aparat, dostęp do Internetu oraz zwykła kontrola stanu sieci. W wersji instalacyjnej ustawiono również `allowBackup=false` i `usesCleartextTraffic=false`.

4.1.4 Bezpieczeństwo lokalnych danych

SQLite znajduje się w prywatnym obszarze aplikacji. Ogranicza to dostęp zwykłych innych aplikacji, ale nie chroni przed wszystkimi scenariuszami, takimi jak odblokowane urządzenie, złośliwe oprogramowanie, kopia systemowa albo urządzenie z uprawnieniami administratora. Obecna wersja nie szyfruje bazy osobnym kluczem.

Systemowa kopia zapasowa danych aplikacji została wyłączona. Zmniejsza to ryzyko automatycznego przeniesienia profilu, historii i rozmowy do magazynu kopii powiązanego z urządzeniem. Ceną jest brak automatycznego odtworzenia danych po reinstalacji lub zmianie telefonu.

Możliwym rozszerzeniem jest SQLCipher oraz przechowywanie klucza w Android Keystore. Wprowadzenie szyfrowania zwiększa jednak złożoność migracji i odzyskiwania danych. Dla prototypu dyplomowego priorytetem było uniknięcie transmisji danych do własnego serwera.

Również AsyncStorage nie powinno być traktowane jako magazyn kryptograficzny. Przechowuje historię rozmowy i ustawienia dla wygody użytkownika, ale obecna wersja nie szyfruje ich osobnym kluczem. Dlatego architektura zapewnia przede wszystkim prywatność względem zewnętrznego operatora, nie zaś pełną poufność wobec osoby posiadającej szeroki dostęp do urządzenia.

4.1.5 Kontrola użytkownika i cykl życia danych

Lokalne przechowywanie daje użytkownikowi bezpośrednią kontrolę nad danymi. Może on usuwać pojedyncze wpisy, wyczyścić rozmowę, wyłączyć AI, usunąć model albo użyć funkcji „Usuń wszystkie dane aplikacji”. Ostatnia operacja wymaga potwierdzenia, zatrzymuje aktywne zadania AI, usuwa wszystkie pliki modeli, bazę `nutritiontracker.db` oraz dane AsyncStorage, a następnie uruchamia aplikację ponownie w stanie początkowym.

Ważne jest rozróżnienie aktualizacji i odinstalowania. Instalacja nowszego APK z tym samym identyfikatorem `com.nutritiontracker.app` jest aktualizacją, dlatego Android zachowuje prywatne dane aplikacji. Nie jest to błąd mechanizmu lokalnego zapisu. Standardowe odinstalowanie usuwa prywatny katalog aplikacji. Dodatkowo manifest ustawia `allowBackup=false`, a reguły ekstrakcji wykluczają bazę, pliki, preferencje i pamięć zewnętrzną z kopii chmurowej oraz transferu między urządzeniami [13].

Podczas testów rozróżniano zwykłą aktualizację oraz czystą instalację. Czysta instalacja usuwa wcześniejszą wersję aplikacji i jej dane przed wgraniem nowego APK, natomiast aktualizacja zachowuje dane użytkownika, tak jak robi to standardowy mechanizm Androida.

4.1.6 Granice odpowiedzialności systemu

Dane pobrane z Open Food Facts są walidowane przed zapisem, a wartości produktów paczkowanych można porównać z etykietą. Model językowy pełni funkcję pomocniczą: nie oblicza celów ani wartości odżywczych, a pytania wymagające dokładnych danych są obsługiwane bezpośrednio przez kod aplikacji. Wariant 1.7B jest przeznaczony dla słabszych urządzeń, natomiast model 4B jest udostępniany tylko przy odpowiedniej ilości pamięci.

4.2. Testowanie systemu

4.2.1 Metodyka

Weryfikację podzielono na cztery poziomy:

1. testy deterministycznych funkcji obliczeniowych;
2. sprawdzenie przepływów interfejsu;

3. walidację konfiguracji Expo i składni źródeł;
4. pełne zbudowanie wersji Android bez serwera Metro.

Największe ryzyko dotyczy natywnych zależności AI, ponieważ aplikacja uruchomiona w Expo Go nie zawiera modułu RNLIama. Test lokalnego modelu wymaga własnego development build albo samodzielnego APK.

4.2.2 Testy funkcji obliczeniowych

Przykładowe przypadki przedstawiono w tabeli 14.

Tabela 14: Przypadki testowe obliczeń

Id	Dane wejściowe	Oczekiwany rezultat	Wynik
T01	80 kg, 180 cm, 20 lat, mężczyzna, aktywność umiarkowana, utrzymanie	2840 kcal	Poprawny
T02	2500 kcal; makro 30/45/25	188 g białka, 281 g węglowodanów, 69 g tłuszczu	Poprawny
T03	200 g piersi kurczaka	330 kcal, 62 g białka	Poprawny
T04	200 g ugotowanego ryżu	260 kcal, 5,4 g białka, 56,4 g węglowodanów	Poprawny
T05	Dwa produkty dodane ręcznie przed zapisem posiłku	Dwa osobne wpisy widoczne w dzienniku i w sumie dnia	Poprawny
T06	Suma makro 30/40/20	Odrzucenie zapisu, suma 90%	Poprawny
T07	Masa 20 kg dla dorosłego profilu	Brak automatycznego wyniku	Poprawny
T08	Cel budowy masy i nieuzupełnione dzienne białko	Pierwsza propozycja lunchu zawiera źródło białka, produkt skrobiowy, co najmniej 100 g warzyw lub owoców i co najmniej 35 g białka	Poprawny
T09	Polski interfejs i pytanie o białko dla masy 80 kg	Polskie etykiety oraz odpowiedź zawierająca zakres 128–160 g	Poprawny
T10	Wykluczenia „mleko, soja, jajka”	Brak nabiału, produktów sojowych i jaj w rekomendacjach	Poprawny

Id	Dane wejściowe	Oczekiwany rezultat	Wynik
T11	Słowniki języka polskiego i angielskiego	Identyczny zestaw kluczy oraz polskie nazwy wszystkich 90 produktów	Poprawny
T12	Pytanie „zobacz mój kalendarz i moje kalorie”	Polskie podsumowanie powstaje bezpośrednio z danych miesiąca, bez odpowiedzi modelu językowego	Poprawny

Źródło: opracowanie własne.

4.2.3 Testy scenariuszy użytkownika

Tabela 15: Główne scenariusze testowe interfejsu

Id	Scenariusz	Kryterium akceptacji
U01	Profil, aktywność i tryb ręczny	Cel odświeża się na ekranie głównym, a ręczne kalorie i makro można edytować.
U02	Katalog, skaner i ręczny produkt	Produkt otwiera wspólny edytor z gramaturą oraz wartościami na 100 g.
U03	Dodanie, edycja i usunięcie wpisu	Każdy produkt jest osobnym rekordem, a sumy dnia są przeliczane po zmianie.
U04	Alergie i produkty unikane	Pasujące produkty i zestawy nie są automatycznie proponowane.
U05	Pierwsze uruchomienie	Po logo otwiera się ekran dzienny, bez wymuszania wyboru modelu AI.
U06	Pobieranie i anulowanie modelu	Użytkownik widzi postęp, ostrzeżenie o Internecie i może zatrzymać transfer.
U07	Słabe urządzenie i model 4B	Pobieranie lub ładowanie jest blokowane czytelnym komunikatem.
U08	Generowanie i przerwanie odpowiedzi	Przycisk zatrzymuje generowanie, a znaczki think nie są pokazywane.
U09	Pytania o kalendarz i kalorie	Odpowiedź korzysta z danych miesiąca i nie twierdzi, że kalendarz jest niedostępny.
U10	Rekomendacje posiłków	Ranking uwzględnia porę dnia, cel, białko, pozostałą energię i wykluczenia.
U11	Zmiana języka	Ekrany, katalog i odpowiedzi lokalnego asystenta pozostają w wybranym języku.
U12	Usunięcie danych i czysta instalacja	Profil, posiłki, rozmowa, ustawienia i pobrane modele zostają usunięte.

Źródło: opracowanie własne.

4.2.4 Ocena zgodności z wymaganiami

Końcowa weryfikacja sprawdza 58 punktów odpowiadających wymaganiom F01–F34 i N01–N24. Dla każdego punktu wskazano konkretny element kodu, konfiguracji Android albo scenariusza testowego. Wszystkie kontrole zakończyły się powodzeniem, ale to nie oznacza, że model będzie identycznie wydajny na każdym telefonie.

Tabela 16: Końcowa zgodność grup wymagań z implementacją

Grupa	Stan	Sposób potwierdzenia
Profil, cele i język	Spełniona	Testy obliczeń, walidacji, zapisu i zgodności słowników.
Dziennik, katalog i kalendarz	Spełniona	Testy SQLite, agregacji, produktów, gramatur i aktualizacji stanu.
Rekomendacje i wykluczenia	Spełniona	Powtarzalne testy rankingu oraz polskich i angielskich nazw alergenów.
Skaner produktów	Spełniona	Kontrola konfiguracji aparatu, walidacji odpowiedzi i lokalnej pamięci.
Lokalne AI	Spełniona warunkowo	Testy kodu i urządzenia; szybkość zależy od RAM, procesora i systemu.
Prywatność i usuwanie danych	Spełniona	Reset bazy, modeli i ustawień oraz kontrola manifestu i reguł kopii.
Samodzielny APK	Spełniona technicznie	Proces wydania zweryfikowano, a po zmianie nazwy zbudowano końcowy, samodzielny APK podpisany kluczem testowym.

Źródło: opracowanie własne.

4.2.5 Test na urządzeniu Samsung Galaxy A23

Test praktyczny wykonano na telefonie Samsung Galaxy A23 z 4 GB RAM. Model Qwen3 1.7B został pobrany, załadowany i generował odpowiedzi, chociaż przed poprawką wyświetlał przy każdym pytaniu znaczniki `<think>` i `</think>`. Model Qwen3 4B został pobrany, lecz podczas ładowania system Android zakończył proces aplikacji. Zaobserwowano również błędny postęp dochodzący do 10 000%.

Na podstawie tego testu ujednolicono skalę postępu, wyłączono tryb rozbudowanego rozumowania, dodano filtrowanie jego znaczników i ustawiono twardy próg 6 GB RAM dla modelu 4B. Poprawiono także budowanie kontekstu wybranego miesiąca. Testy automatyczne obejmują obie spotykane skale postępu, usuwanie znaczników, wybór okresu, bezpośrednie odpowiedzi o kalendarzu, limit kontekstu oraz rekomendacje zależne od pory posiłku, celu i struktury zestawu.

4.2.6 Walidacja techniczna

Parser Babel przeanalizował 42 pliki źródłowe JavaScript, czyli 41 modułów z katalogu `src` oraz plik `App.js`, bez błędów składni. Polecenie `npm test` wykonało 58 kontroli dowodów realizacji wymagań oraz testy deterministyczne obliczeń profilu, rekomendacji, katalogu, polskich tłumaczeń, przecinka dziesiątego, wykluczeń żywieniowych, kontekstu AI i ustawień

prywatności. Expo Doctor zakończył 17 z 17 kontroli bez błędów, a `expo install -check` potwierdził zgodność zależności z Expo SDK 53. Audyt npm nie wykazał podatności wysokich ani krytycznych. Zgłosił 22 ostrzeżenia umiarkowane w łańcuchu narzędzi Expo. Dostępna automatyczna poprawka wymaga przejścia na inną główną wersję Expo, dlatego nie została wprowadzona do stabilnej wersji przygotowanej do prezentacji.

Proces budowania wersji `release` zweryfikowano pełnym przejściem zadań Gradle. Po zmianie nazwy aplikacji na NutritionTracker ponownie wykonano testy kodu źródłowego, kontrolę konfiguracji Expo, kompilację Kotlin, przetwarzanie zasobów Androida i Android Lint. Produkcyjny eksport Metro i kompilacja bundla Hermes objęły 2678 modułów. Końcowy wynik stanowi samodzielny pakiet APK podpisany kluczem testowym.

APK zawiera natywny moduł `llama.rn`, aparat, SQLite i osadzony pakiet JavaScript. Nie wymaga uruchomionego Metro Bundlera. Modele pozostają osobnymi plikami pobieranymi po instalacji.

Tabela 17: Wyniki końcowej walidacji technicznej

Kontrola	Wynik	Znaczenie
Parser Babel	42/42 pliki	Brak błędów składni kodu aplikacji.
<code>npm test</code>	58 kontroli wymagań	Powtarzalna kontrola profilu, rekomendacji, tłumaczeń, żywienia, AI i prywatności.
Expo Doctor	17/17 kontroli	Poprawna konfiguracja projektu i zgodność zależności.
Audyt npm	0 wysokich i krytycznych	Pozostały 22 umiarkowane ostrzeżenia w zależnościach narzędziowych Expo.
Konfiguracja Expo	Poprawna	Nazwa, ikona, slug i pakiet <code>com.nutritiontracker.app</code> są spójne.
Kotlin i zasoby Androida	Sukces	Klasy aplikacji, manifest, zasoby i Android Lint zostały przetworzone.
Manifest Androida	2 główne uprawnienia	Aparat i Internet; brak dźwięku, nakładek i pamięci zewnętrznej.
Ustawienia prywatności	Aktywne	Kopia systemowa i nieszyfrowany ruch HTTP są wyłączone.
Końcowy APK i podpis	Gotowy proces	Utworzono samodzielny pakiet APK podpisany kluczem testowym.

Źródło: opracowanie własne.

Mierzalny zakres implementacji Zestawienie w tabeli 18 opisuje stan kodu i artefaktów projektu z 15 czerwca 2026 r. Liczba wierszy została wyznaczona dla plików źródłowych w

katalogu `src`, bez zależności, kodu wygenerowanego i projektu Android tworzonego przez Expo.

Tabela 18: Mierzalny zakres implementacji aplikacji

Element	Wartość	Zakres
Moduły w katalogu <code>src</code>	41	Ekrany, komponenty, usługi, stan, nawigacja, dane i narzędzia pomocnicze.
Kod w katalogu <code>src</code>	5761 wierszy	5394 wiersze niepuste; bez <code>App.js</code> , zależności i kodu wygenerowanego.
Ekrany aplikacji	7	Dzień, posiłek, skaner, kalendarz, profil z ustawieniami AI, rozmowa oraz ekran błędu inicjalizacji.
Główna nawigacja	4 zakładki	Dzień, kalendarz, asystent oraz profil; formularz i skaner jako ekrany modalne.
Języki interfejsu	2	Polski i angielski, z lokalnym zapisem wyboru i instrukcją języka dla AI.
Tabele SQLite	3	Profil, posiłki i pamięć zeskanowanych produktów.
Produkty lokalne	90	Produkty z polskimi i angielskimi aliasami, porcjami i wartościami na 100 g.
Kontrole wymagań	58	Dowody implementacji funkcji i cech jakościowych oraz dodatkowe testy domenowe.
Lokalne modele	2	Qwen3 1.7B o rozmiarze 1,28 GB oraz Qwen3 4B o rozmiarze 2,50 GB.
Pakiet instalacyjny	Samodzielny APK	Proces budowania tworzy pakiet o rozmiarze około 156 MB, zawierający kod JavaScript i wymagane moduły natywne; modele AI pozostają osobnymi plikami pobieranymi po instalacji.
Własny backend	0 usług	Podstawowe funkcje i lokalne AI nie wymagają serwera utrzymywanego przez autora.

Źródło: opracowanie własne.

4.3. Ocena rozwiązania

Testy potwierdziły działanie najważniejszych scenariuszy dziennika, rekomendacji i lokalnego asystenta. Zakres obecnej wersji celowo koncentruje się na jednym użytkowniku i urządzeniu.

Katalog lokalny obejmuje najczęstsze produkty, skaner zależy od jakości wpisu w Open Food Facts, a szybkość asystenta zależy od możliwości telefonu. Ograniczenia te nie blokują podstawowego dziennika, ponieważ zapis, obliczenia, historia i rekomendacje działają niezależnie od modelu językowego.

ZAKOŃCZENIE

W pracy powstała aplikacja *NutritionTracker* do monitorowania kalorii i makroskładników. Profil oraz historia posiłków są przechowywane na telefonie, a podstawowy dziennik nie potrzebuje konta ani serwera autora. Taka architektura upraszcza utrzymanie projektu i ogranicza liczbę miejsc, do których trafiają dane użytkownika.

Produkt można dodać ręcznie, wybrać z katalogu albo pobrać po zeskanowaniu kodu. Wszystkie drogi prowadzą do jednego edytora, a zapisane pozycje można później otworzyć i poprawić. Lokalne rekomendacje uwzględniają cel, bieżące spożycie i produkty wykluczone. Obliczenia celu korzystają z równania Mifflina–St Jeora oraz jawnych przeliczeń makroskładników. Tryb ręczny pozostawia użytkownikowi możliwość ustawienia własnych wartości.

Najbardziej eksperymentalną częścią jest lokalny asystent Qwen3. Test na telefonie z 4 GB RAM pokazał zarówno zaletę tego podejścia, czyli brak konieczności wysyłania rozmowy do chmury, jak i jego koszt. Mały model działa, lecz odpowiada wolniej i dysponuje ograniczoną wiedzą. Wariant 4B przekracza możliwości słabszego urządzenia. Z tego powodu model zajmuje się rozmową, a cele, wartości produktów i rekomendacje są obliczane poza nim.

Aplikację można zbudować jako samodzielny APK działający bez Metro Bundlera i bez środowiska deweloperskiego. Założony cel pracy został osiągnięty. Powstał spójny, lokalny system łączący dziennik żywieniowy, rekomendacje posiłków i opcjonalnego asystenta językowego.

SPIS TABEL

1	Współczynniki aktywności zastosowane w aplikacji	9
2	Podział odpowiedzialności między kod aplikacji i model językowy	11
3	Porównanie założeń aplikacji <i>NutritionTracker</i> , <i>MyFitnessPal</i> i <i>Cronometer</i> . .	13
4	Główne założenia projektowe aplikacji	15
5	Najważniejsze elementy wkładu własnego autora	16
6	Wymagania funkcjonalne systemu	17
7	Wymagania нефункционалне systemu	19
8	Powiązanie głównych celów z wymaganiami systemu	21
9	Tabele lokalnej bazy danych	25
10	Źródła danych używane przez aplikację i lokalnego asystenta	34
11	Priorytety algorytmu rekomendowania posiłków	40
12	Lokalne modele dostępne w aplikacji	43
13	Lokalizacja i przepływ danych w aplikacji	47
14	Przypadki testowe obliczeń	50
15	Główne scenariusze testowe interfejsu	52
16	Końcowa zgodność grup wymagań z implementacją	53
17	Wyniki końcowej walidacji technicznej	54
18	Mierzalny zakres implementacji aplikacji	55

SPIS RYSUNKÓW

1	Diagram głównych przypadków użycia aplikacji <i>NutritionTracker</i>	22
2	Architektura logiczna aplikacji <i>NutritionTracker</i>	24
3	Przepływ danych podczas rejestrowania żywności	25
4	Schemat głównych pól lokalnej bazy danych <i>nutritiontracker.db</i>	26
5	Ekran dziennego podsumowania oraz rozmowa z lokalnym asystentem	27
6	Ustawienia danych, celu i bezpieczeństwa żywienia	28
7	Konfiguracja lokalnego modelu AI	29
8	Podział kodu źródłowego aplikacji na moduły	30
9	Rozkład niepustych wierszy kodu pomiędzy warstwy aplikacji	30
10	Przebieg lokalnego rekomendowania posiłków	41
11	Przepływ obsługi pytania do lokalnego asystenta	45
12	Granica prywatności i komunikacja z usługami zewnętrznymi	48

BIBLIOGRAFIA

- [1] A. A. Aragon i in., *International Society of Sports Nutrition position stand: diets and body composition*, Journal of the International Society of Sports Nutrition, 14:16, 2017, DOI: 10.1186/s12970-017-0174-y, <https://link.springer.com/article/10.1186/s12970-017-0174-y>, dostęp: 12 stycznia 2026 r.
- [2] Cronometer, *Mobile - Photo-Logging*, aktualizacja: 24 czerwca 2026 r., <https://support.cronometer.com/hc/en-us/articles/39013533811092>, dostęp: 25 czerwca 2026 r.
- [3] Cronometer, *Mobile - Scan Food*, aktualizacja: 24 czerwca 2026 r., <https://support.cronometer.com/hc/en-us/articles/360020441392>, dostęp: 25 czerwca 2026 r.
- [4] Cronometer, *Data security at the forefront*, <https://cronometer.com/features/data-security.html>, dostęp: 21 marca 2026 r.
- [5] Cronometer, *Mobile - Quick Start Guide*, aktualizacja: 24 czerwca 2026 r., <https://support.cronometer.com/hc/en-us/articles/360021677792>, dostęp: 25 czerwca 2026 r.
- [6] Cronometer, *Subscription Types*, aktualizacja: 15 listopada 2025 r., <https://support.cronometer.com/hc/en-us/articles/360028026971>, dostęp: 26 marca 2026 r.
- [7] Expo, *Expo Camera*, <https://docs.expo.dev/versions/latest/sdk/camera/>, dostęp: 3 lutego 2026 r.
- [8] Expo, *Expo Device*, <https://docs.expo.dev/versions/latest/sdk/device/>, dostęp: 4 lutego 2026 r.
- [9] Expo, *Expo SDK reference*, <https://docs.expo.dev/versions/latest/>, dostęp: 5 lutego 2026 r.
- [10] Expo, *Expo SQLite*, <https://docs.expo.dev/versions/latest/sdk/sqlite/>, dostęp: 6 lutego 2026 r.
- [11] Food and Agriculture Organization of the United Nations, *Food energy: methods of analysis and conversion factors*, FAO Food and Nutrition Paper 77, Rzym 2003, <https://www.fao.org/4/y5022e/y5022e00.htm>, dostęp: 15 stycznia 2026 r.
- [12] GGML contributors, *GGUF specification*, <https://github.com/ggml-org/ggml/blob/master/docs/gguf.md>, dostęp: 10 lutego 2026 r.
- [13] Google, *Back up user data with Auto Backup*, <https://developer.android.com/identity/data/autobackup>, dostęp: 12 lutego 2026 r.
- [14] Google, *The activity lifecycle*, <https://developer.android.com/guide/components/activities/activity-lifecycle>, dostęp: 13 lutego 2026 r.
- [15] R. Jäger i in., *International Society of Sports Nutrition position stand: protein and exercise*, Journal of the International Society of Sports Nutrition, 14:20, 2017, DOI: 10.1186/s12970-017-0177-8, <https://link.springer.com/article/10.1186/s12970-017-0177-8>, dostęp: 16 stycznia 2026 r.

- [16] C. M. Kerksick i in., *International Society of Sports Nutrition position stand: nutrient timing*, Journal of the International Society of Sports Nutrition, 14:33, 2017, DOI: 10.1186/s12970-017-0189-4, <https://link.springer.com/article/10.1186/s12970-017-0189-4>, dostęp: 19 stycznia 2026 r.
- [17] Kod źródłowy projektu *NutritionTracker*, wersja 1.0.4, stan na dzień 15 czerwca 2026 r., materiał własny autora.
- [18] llama.rn contributors, *React Native binding of llama.cpp*, <https://github.com/myb igday/llama.rn>, dostęp: 17 lutego 2026 r.
- [19] M. M. Mamerow i in., *Dietary protein distribution positively influences 24-h muscle protein synthesis in healthy adults*, The Journal of Nutrition, 144(6), s. 876–880, 2014, DOI: 10.3945/jn.113.185280, <https://doi.org/10.3945/jn.113.185280>, dostęp: 22 stycznia 2026 r.
- [20] Meta Platforms, *React Native Architecture Overview*, <https://reactnative.dev/architecture/overview>, dostęp: 19 lutego 2026 r.
- [21] Meta Platforms, *React Native Documentation*, <https://reactnative.dev/docs/getting-started>, dostęp: 20 lutego 2026 r.
- [22] M. D. Mifflin, S. T. St Jeor, L. A. Hill, B. J. Scott, S. A. Daugherty, Y. O. Koh, *A new predictive equation for resting energy expenditure in healthy individuals*, The American Journal of Clinical Nutrition, 51(2), 1990, s. 241–247, DOI: 10.1093/ajcn/51.2.241, <https://doi.org/10.1093/ajcn/51.2.241>, dostęp: 27 stycznia 2026 r.
- [23] MyFitnessPal, *How do I use the barcode scanner to log foods?*, aktualizacja: 30 września 2024 r., <https://support.myfitnesspal.com/hc/en-us/articles/360032624771>, dostęp: 8 kwietnia 2026 r.
- [24] MyFitnessPal, *What information does MyFitnessPal collect?*, aktualizacja: 18 marca 2025 r., <https://support.myfitnesspal.com/hc/en-us/articles/34887587961997>, dostęp: 10 kwietnia 2026 r.
- [25] MyFitnessPal, *Czy mogę uzyskać dostęp do MyFitnessPal, gdy nie mam połączenia z Internetem?*, aktualizacja: 2 sierpnia 2023 r., <https://support.myfitnesspal.com/hc/pl/articles/360032622851>, dostęp: 11 kwietnia 2026 r.
- [26] MyFitnessPal, *What's the difference between Free, Premium, and Premium +?*, aktualizacja: 26 maja 2026 r., <https://support.myfitnesspal.com/hc/en-us/articles/34889191368077>, dostęp: 27 maja 2026 r.
- [27] Open Food Facts, *Introduction to Open Food Facts API documentation*, <https://openfoodfacts.github.io/openfoodfacts-server/api/>, dostęp: 16 kwietnia 2026 r.
- [28] Open Food Facts, *Open Food Facts Documentation: Overview*, <https://openfoodfacts.github.io/documentation/docs/>, dostęp: 17 kwietnia 2026 r.
- [29] Parlament Europejski i Rada Unii Europejskiej, *Rozporządzenie (UE) 2016/679, art. 5 i 25*, <https://eur-lex.europa.eu/eli/reg/2016/679/oj/pol>, dostęp: 24 kwietnia 2026 r.
- [30] QuantFactory, *Qwen3-1.7B-GGUF: Q4_K_M quantization*, <https://huggingface.co/QuantFactory/Qwen3-1.7B-GGUF>, dostęp: 6 maja 2026 r.

- [31] Qwen Team, *Qwen3-1.7B: model card*, <https://huggingface.co/Qwen/Qwen3-1.7B>, dostęp: 5 maja 2026 r.
- [32] Qwen Team, *Qwen3-4B-GGUF: model card and license*, <https://huggingface.co/Qwen/Qwen3-4B-GGUF>, dostęp: 7 maja 2026 r.
- [33] Qwen Team, *Qwen3 Technical Report*, arXiv:2505.09388, 2025, <https://arxiv.org/abs/2505.09388>, dostęp: 8 maja 2026 r.
- [34] SQLite Consortium, *Write-Ahead Logging*, <https://www.sqlite.org/wal.html>, dostęp: 29 kwietnia 2026 r.
- [35] U.S. Department of Agriculture, Agricultural Research Service, *FoodData Central*, <https://fdc.nal.usda.gov/>, dostęp: 22 kwietnia 2026 r.
- [36] World Health Organization, *Healthy diet*, aktualizacja: 26 stycznia 2026 r., <https://www.who.int/news-room/fact-sheets/detail/healthy-diet>, dostęp: 28 stycznia 2026 r.

SŁOWNIK POJĘĆ

AI	sztuczna inteligencja (<i>Artificial Intelligence</i>).
API	interfejs programistyczny umożliwiający komunikację między aplikacją a zewnętrzną usługą.
APK	pakiet instalacyjny aplikacji przeznaczonej dla systemu Android.
BMR	podstawowa przemiana materii (<i>Basal Metabolic Rate</i>).
Embedding	wektor liczbowy reprezentujący znaczenie tekstu; nie jest używany w obecnej wersji aplikacji.
GGUF	format pliku przeznaczony do przechowywania modeli językowych i ich metadanych.
LLM	duży model językowy (<i>Large Language Model</i>).
Offline-first	podejście projektowe, w którym podstawowe funkcje aplikacji pozostają dostępne bez połączenia z Internetem.
RAG	metoda łączenia modelu językowego z wyszukiwaniem w zewnętrznym zbiorze wiedzy (<i>Retrieval-Augmented Generation</i>); nie jest używana w obecnej wersji.
SQLite	lekki, osadzony system relacyjnej bazy danych działający bez osobnego serwera.
Tokenizer	moduł dzielący tekst na tokeny przetwarzane przez model.
TDEE	całkowity dzienny wydatek energetyczny (<i>Total Daily Energy Expenditure</i>).