Aplikacja webowa do zarządzania codzienną organizacją życia i monitorowania well-beingu

(A web application for daily life organization and well-being monitoring)

Cezary Miłek

Praca inżynierska

Promotor: dr Marcin Młotkowski

Uniwersytet Wrocławski
Wydział Matematyki i Informatyki
Instytut Informatyki

29 stycznia 2026

Streszczenie

Celem pracy inżynierskiej było zaprojektowanie i implementacja aplikacji webowej pełniącej rolę centralnego huba do organizacji dnia i monitorowania well-beingu.

Zrealizowana aplikacja łączy w sobie funkcjonalności z zakresu dietetyki, aktywności fizycznej, finansów oraz zarządzania zadaniami w jednym, spójnym systemie. Wspiera on wszystkie urządzenia docelowe (telefon, tablet, komputer) i charakteryzuje się modularną architekturą, pozwalającą na personalizację interfejsu i wybór aktywnych funkcjonalności.

Aplikacja została wykonana w modelu SPA (*Single Page Application*) przy użyciu Reacta, frameworku Tailwind CSS, bibliotek Headless UI i Framer Motion oraz języka TypeScript. Backend w modelu BaaS (*Backend-as-a-Service*) obsługiwany jest przez platformę Supabase, zapewniającą bazę danych PostgreSQL, mechanizmy uwierzytelniania oraz funkcje bezserwerowe (służące do komunikacji z zewnętrznymi API).

Przeprowadzone testy potwierdziły poprawność działania systemu oraz realizację założonych celów projektowych.

This engineering thesis focused on the design and implementation of a web application serving as a central hub for daily organization and well-being monitoring.

Integrating functionalities from the fields of nutrition, physical activity, finance and task management, the developed solution creates a single, cohesive system. It supports all target devices (mobile, tablet, computer) and features a modular architecture, allowing for interface personalization and selection of active functionalities.

Technologically, the application was built in the SPA (*Single Page Application*) model using React, Tailwind CSS framework, Headless UI and Framer Motion libraries, and TypeScript. The backend, in the BaaS (*Backend-as-a-Service*) model, is powered by the Supabase platform, which provides a PostgreSQL database, authentication mechanisms and serverless functions (used for communication with external APIs).

Conducted tests confirmed system correctness and fulfillment of the design goals.

Spis treści

1. Wprowadzenie **7**

1.1. Analiza problemu 7

1.1.1. Geneza i motywacja podjęcia tematu 7

1.2. Przegląd i porównanie istniejących rozwiązań 8

1.3. Cel i zakres pracy 10

1.3.1. Główny cel pracy dyplomowej 10

1.3.2. Zakres funkcjonalny MVP 10

1.4. Struktura pracy dyplomowej 11

2. Założenia i wymagania **13**

2.1. Urządzenia docelowe 13

2.2. Wymagania funkcjonalne 13

2.3. Wymagania niefunkcjonalne 16

3. Architektura i implementacja **19**

3.1. Charakterystyka środowiska technologicznego 19

3.1.1. Stos technologiczny 19

3.1.2. Warstwa prezentacji 20

3.1.3. Warstwa backendowa i infrastruktura danych 20

3.1.4. Bezpieczeństwo i kontrola dostępu 21

3.1.5. Wdrożenie i hosting 21

3.2. Architektura systemu 22

3.2.1. Struktura logiczna i organizacja modułów 22

3.2.2. Model danych i schemat relacyjny 23
3.2.3. Komunikacja i interfejs API 26

4. Przegląd funkcjonalności systemu 29
4.1. Logowanie i rejestracja 29
4.2. Pulpit i system nawigacji 30
4.3. Moduły aplikacji 33
4.3.1. Moduł śledzenia kalorii 33
4.3.2. Moduł monitorowania aktywności 36
4.3.3. Moduł zarządzania finansami 38
4.3.4. Moduł zarządzania zadaniami 41

5. Dokumentacja i testy systemu 45
5.1. Dokumentacja wdrożeniowa 45
5.1.1. Instrukcja uruchomienia (dla deweloperów) 45
5.1.2. Konfiguracja środowiska 46
5.1.3. Dostępność wersji demonstracyjnej 47
5.2. Weryfikacja i walidacja systemu 47
5.2.1. Środowisko i narzędzia testowe 47
5.2.2. Testy automatyczne 47
5.2.3. Weryfikacja scenariuszy użytkownika (Testy manualne) 48
5.2.4. Weryfikacja stopnia realizacji wymagań 48
5.2.5. Weryfikacja wydajności i dostępności 50

6. Zakończenie 53
6.1. Podsumowanie i ocena realizacji celów 53
6.2. Wnioski z projektu 54
6.3. Propozycje dalszego rozwoju 55

Bibliografia 57

Rozdział 1.

Wprowadzenie

1.1. Analiza problemu

1.1.1. Geneza i motywacja podjęcia tematu

Współczesny świat charakteryzuje się dynamicznym tempem życia oraz rosnącą zależnością od technologii cyfrowych w zarządzaniu codziennymi obowiązkami. Użytkownicy komputerów, tabletów i smartfonów sięgają po liczne aplikacje, aby łatwiej było im monitorować swoją dietę, planować treningi, organizować codzienne zadania albo lepiej kontrolować wydatki.

Problem pojawia się, gdy każdy z tych obszarów życia jest obsługiwany przez osobne, często niezintegrowane narzędzia. Prowadzi to do fragmentacji informacji i konieczności przełączania się wielokrotnie między kilkoma aplikacjami, co skutkuje brakiem całościowego obrazu postępów w realizacji celów oraz produktywności.

Takie rozproszenie danych może zaćmić dostrzeganie powiązań między różnymi sferami życia. Trochę ciężej jest dostrzec łańcuch przyczynowo-skutkowy, taki jak na przykład powiązanie słabszej wydolności fizycznej z gorszą w ostatnim czasie dietą, która za to pojawiła się pod wpływem przytłaczającej ilości zadań.

Brak synchronizacji tych informacji powoduje frustrację, która potencjalnie prowadzi do demotywacji i porzucenia monitorowania swojego stylu życia, a co za tym idzie, zmniejsza szansę na jego polepszenie. Ciągła zmiana kontekstu pomiędzy programami utrudnia stworzenie spójnej i efektywnej rutyny oraz prowadzi do nieefektywnego gospodarowania czasem.

W odpowiedzi na ten problem pojawia się potrzeba stworzenia jednego, zintegrowanego rozwiązania, które pełniłoby rolę centralnego huba organizacyjnego. Motywacją do podjęcia tematu jest chęć stworzenia właśnie takiego narzędzia i dostarczenia użytkownikowi rozwiązania typu "wszystko w jednym", wspierającego budowanie zdrowych nawyków i zwiększenie kontroli nad codziennym życiem.

1.2. Przegląd i porównanie istniejących rozwiązań

Rynek oprogramowania do zarządzania produktywnością i stylem życia jest nasycony, jednak większość rozwiązań skupia się na wąskich specjalizacjach. Patrząc na zakres ich funkcjonalności, można podzielić je na dwie główne kategorie:

- **Aplikacje specjalistyczne** - oferują zaawansowane funkcje w swojej dziedzinie, jednak izolacja gromadzonych w nich danych sprawia, że użytkownik musi samodzielnie łączyć fakty z różnych źródeł. Przykładowe aplikacje tego typu to: **MyFitnessPal** [1], **Fitatu** [2] (monitorowanie diety), **Strava** [3] (śledzenie aktywności fizycznej) lub **aplikacje bankowe**.
- **Ogólne narzędzia organizacyjne** - pozwalają na zbudowanie samemu systemu takiego jak tracker nawyków/budżetu. Wadą tego rozwiązania jest nieco wyższy próg wejścia niż do specjalistycznej aplikacji (wynikający z potrzeby konfiguracji i/lub zrozumienia działania użytego szablonu, jeśli został użyty) oraz brak części przydatnych, dedykowanych funkcji, takich jak pobieranie wartości odżywczych produktów. Przykładowe aplikacje tego typu to na przykład **Notion** [4] albo **Obsidian** [5].

Tabela 1.1: Porównanie funkcjonalności wybranych aplikacji specjalistycznych

Aplikacja	**Główne funkcje żywieniowe**	**Główne funkcje aktywności i fitness**	**Główne funkcje finansowe**
Fitatu	Liczenie kalorii i makroskładników, skanowanie kodów kreskowych, baza polskich produktów, planowanie posiłków, post przerywany.	Baza treningów, synchronizacja z niektórymi aplikacjami fitness, proste raporty aktywności.	Brak funkcji finansowych i budżetowania.
MyFitnessPal	Liczenie kalorii i makroskładników, skanowanie kodów kreskowych, globalna baza produktów, monitorowanie wody.	Śledzenie aktywności, integracja z zewnętrznymi trackerami kroków, wykresy postępów.	Brak funkcji finansowych i budżetowania.
Strava	Brak zaawansowanych funkcji dietetycznych.	GPS tracking (bieg/rower), segmenty rywalizacyjne, analiza tempa, mapy tras.	Brak funkcji finansowych i budżetowania.
Aplikacje bankowe	Brak funkcji dietetycznych.	Brak funkcji fitness.	Budżetowanie, możliwość kategoryzacji, podsumowanie finansów, raporty wydatków.

Istotną barierą w aplikacjach specjalistycznych jest model monetyzacji. Wiele funkcjonalności jest blokowanych w darmowych wersjach, a pojawiające się reklamy wpływają negatywnie na doświadczenie użytkownika, co może zniechęcić do regularnego używania aplikacji - tym bardziej jeśli próbuje się wyrobić dobre nawyki od zera.

Z kolei bardziej zaawansowane funkcje wersji premium często wykraczają poza potrzeby przeciętnego użytkownika. Koniec końców większość konsumentów i tak ogranicza się jedynie do tego, do czego inicjalnie pobrali aplikację. Biorąc na przykład aplikację Fitatu - użytkownik pobiera ją, aby mieć wgląd w spożywane posiłki i ich wartości odżywcze. Wersja premium pozbywa się reklam i oferuje dodatkowe funkcjonalności, takie jak szacowanie kaloryczności ze zdjęcia za pomocą AI lub sugerowanie konkretnych posiłków, ale finalnie konsument i tak wraca do najprostszej metody, czyli ręcznego wyszukiwania/skanowania kodu i dodawania produktu.

Zaawansowane funkcje, choć brzmią imponująco w opisie produktu, w codziennym użytkowaniu są często zbędnym dodatkiem, a opłata subskrypcyjna może być postrzegana jako niewspółmierna do oferowanych korzyści. Te funkcjonalności też znajdą swoich zwolenników, ale stanowić będą oni margines w tłumie typowych konsumentów.

Tabela 1.2: Porównanie modeli finansowych i funkcji premium wybranych aplikacji specjalistycznych

Aplikacja	**Model finansowy**	**Funkcje premium**
Fitatu	Darmowy plan podstawowy. Subskrypcja rozszerzająca.	Brak reklam, sugestie posiłków (makro/kalorie), szacowanie AI ze zdjęcia, tryb postu przerywanego, synchronizacja z Apple Watch, baza przepisów, smart lista zakupów.
MyFitnessPal	Darmowa z ograniczeniami. Subskrypcja rozszerzająca.	Zaawansowana analiza makroskładników, brak reklam, skanowanie kodu kreskowego, szczegółowe raporty postępów, priorytetowe wsparcie.
Strava	Darmowa z ograniczeniami. Subskrypcja rozszerzająca.	Zaawansowane analizy, mapy tras offline, tworzenie własnych tras, tworzenie i wykonywanie wyzwań ze znajomymi, niestandardowe cele.
Aplikacje bankowe	Wliczone w koszt prowadzenia konta.	Funkcje inwestycyjne, kantor walutowy, budżetowanie.

Zupełnie inne podejście prezentują uniwersalne narzędzia do zarządzania wiedzą i zadaniami (*all-in-one workspaces*) oraz arkusze kalkulacyjne. Programy takie jak Notion czy Obsidian nie narzucają użytkownikowi gotowego procesu, oferując zamiast tego elastyczność i możliwość zbudowania własnego systemu od podstaw.

Ta swoboda wiąże się jednak z wyższą barierą wejścia i kosztem czasowym. Użytkownik, zamiast od razu przystąpić do działania (np. logowania posiłku), zmuszony jest najpierw skonfigurować narzędzie pod siebie. Musi samodzielnie zaprojektować wprowadzanie danych, stworzyć relacje oraz dostosować interfejs. Może to prowadzić do sytuacji, gdzie więcej czasu poświęca się na udoskonalanie i konfigurowanie narzędzia, niż faktyczną realizację celów.

Największą wadą tych rozwiązań w kontekście śledzenia stylu życia jest jednak brak integracji z zewnętrznymi źródłami danych. Aplikacja taka jak Fitatu posiada bazę produktów z polskich sklepów i automatyzuje proces wprowadzania makroelementów do aplikacji, natomiast ogólne narzędzia do organizacji wymagają od przeciętnego użytkownika, aby ręcznie wpisywać oszacowane wartości albo przepisywać wartości kaloryczne i makroskładniki z etykiet. Sprawia to, że utrzymanie regularności w dłuższym okresie staje się dla typowego konsumenta nużące i w efekcie prowadzi do porzucenia systemu.

Proponowana w pracy aplikacja ma na celu wypełnienie luki pomiędzy tymi dwoma kategoriami, czyli zaoferowanie wyspecjalizowanych modułów (jak w tych dedykowanych aplikacjach) w ramach jednej, spójnej platformy (jak w narzędziach ogólnych).

1.3. Cel i zakres pracy

1.3.1. Główny cel pracy dyplomowej

Celem niniejszej pracy inżynierskiej jest zaprojektowanie i implementacja aplikacji webowej, która będzie pełniła funkcję centralnego huba do organizacji codziennego życia. Ideą projektu jest zastąpienie korzystania z wielu oddzielnych programów przez zaoferowanie użytkownikowi jednego, spójnego ekosystemu do zarządzania codzienną organizacją życia i monitorowania well-beingu.

Kluczowym założeniem projektu jest modularna architektura. Użytkownik powinien mieć możliwość dostosowania interfejsu do własnych potrzeb, a przede wszystkim wyboru i aktywacji tylko tych modułów, które są dla niego ważne. Ma to na celu sprawienie, że aplikacja nie jest przeładowana zbędnymi dla odbiorcy funkcjami oraz upraszcza jej obsługę.

1.3.2. Zakres funkcjonalny MVP

Zakres funkcjonalny minimalnej wersji produktu (*Minimum Viable Product - MVP*) obejmuje implementacje panelu głównego oraz czterech kluczowych modułów:

- **Pulpit**: Centralny hub prezentujący podsumowania z aktywnych modułów.

Użytkownik posiada możliwość personalizacji widoku i włączonych funkcjonalności - sam decyduje, które dane są dla niego najważniejsze.

- **Moduł żywieniowy**: Śledzenie spożywanych na co dzień kalorii i wartości odżywczych. Moduł zintegrowany z zewnętrznym API umożliwiającym wyszukiwanie produktów w globalnej bazie danych i dodawanie ich na podstawie kodów kreskowych.
- **Moduł fitnessowy**: Zarządzanie aktywnością fizyczną poprzez bazę ćwiczeń (siłowe, cardio, rozciąganie) oraz kreator planów treningowych. Umożliwia rejestrowanie wykonanych serii, powtórzeń, obciążenia lub czasu trwania określonego ćwiczenia.
- **Moduł finansowy**: Kontrola budżetu z podziałem na kategorie. Obejmuje rejestrowanie transakcji, definiowanie budżetów i zestawianie wpisów ze sobą.
- **Moduł zadaniowy**: Narzędzie do organizacji pracy w widoku tablicy. Pozwala na priorytetyzacje, kategoryzacje i ustalanie terminów realizacji zadań.

1.4. Struktura pracy dyplomowej

Niniejsza praca składa się z sześciu rozdziałów. **Rozdział pierwszy** (obecny) stanowi wprowadzenie do tematyki, definiuje cel pracy oraz zakres realizowanych zadań. **W rozdziale drugim** opisano urządzenia docelowe oraz przeprowadzono analizę wymagań funkcjonalnych i niefunkcjonalnych. **Rozdział trzeci** przedstawia projekt aplikacji, w tym architekturę i model bazy danych. Opisuje również proces implementacji, w którym przedstawiono wybrane technologie oraz rozwiązania programistyczne zastosowane przy budowie poszczególnych modułów aplikacji. **Rozdział czwarty** stanowi przegląd funkcjonalności systemu, wraz z opisami korzystania z aplikacji. **Rozdział piąty** przedstawia dokumentację, instrukcję instalacji wytworzonego oprogramowania oraz testy systemu. Pracę zamyka **rozdział szósty**, stanowiący podsumowanie. Zawiera on wnioski z realizacji projektu oraz wskazanie potencjalnych kierunków dalszego rozwoju aplikacji.

Rozdział 2.

Założenia i wymagania

2.1. Urządzenia docelowe

Zgodnie z założeniami projektowymi aplikacja ma mieć charakter uniwersalny i powinna funkcjonować poprawnie na szerokim zakresie urządzeń końcowych. Wymagane jest więc dostosowanie interfejsu do:

- **Smartfonów** - obsługa ekranów dotykowych o małej przekątnej (widok mobilny).
- **Tabletów** - obsługa ekranów dotykowych o średniej przekątnej.
- **Komputerów osobistych** - obsługa dużych ekranów sterowanych myszą (widok desktopowy).

2.2. Wymagania funkcjonalne

Poniższa sekcja prezentuje szczegółowe wymagania funkcjonalne systemu, pogrupowane według modułów aplikacji. Każde wymaganie opatrzone jest unikalnym identyfikatorem w formacie **WF-XX**.

Moduł uwierzytelniania i zarządzania kontekstem

Moduł odpowiedzialny za bezpieczeństwo dostępu do aplikacji, realizację procesu weryfikacji tożsamości użytkownika oraz utrzymanie kontekstu jego pracy.

- **WF-01 Rejestracja i logowanie użytkowników:** System musi umożliwiać utworzenie nowego konta użytkownika oraz autoryzację dostępu dla istniejących kont przy użyciu bezpiecznego mechanizmu uwierzytelnienia.

- **WF-02 Zarządzanie sesją i wylogowanie:** System musi utrzymywać bezpieczną sesję użytkownika oraz umożliwiać jej manualne zakończenie (wylogowanie).
- **WF-03 Kontrola dostępu do zasobów:** System musi weryfikować uprawnienia przy każdej próbie dostępu do widoków wewnętrznych, przekierowując niezalogowanych użytkowników do ekranu logowania.
- **WF-04 Zarządzanie danymi profilowymi:** System umożliwia użytkownikowi edycję danych powiązanych z kontem, w tym zmianę hasła oraz danych personalnych.

Moduł żywieniowy

Moduł realizujący funkcje monitorowania diety i analizy wartości odżywczych spożywanych produktów.

- **WF-05 Wyszukiwanie produktów spożywczych:** System zapewnia wyszukiwarkę produktów zintegrowaną poprzez API z zewnętrzną bazą danych oraz wewnętrzną bazą użytkownika.
- **WF-06 Skanowanie kodów kreskowych:** System umożliwia automatyczne pobranie nazwy i informacji o makroskładnikach z kodu kreskowego produktu.
- **WF-07 Dziennik żywieniowy:** System umożliwia rejestrację spożytych posiłków w ujęciu dziennym. Automatycznie przelicza kaloryczność i makroskładniki (białko, tłuszcze, węglowodany) proporcjonalnie do wprowadzonej gramatury produktu.
- **WF-08 Definiowanie własnych produktów:** System umożliwia użytkownikowi manualne dodawanie nowych produktów do lokalnej bazy danych z określeniem wartości odżywczych i (opcjonalnym) rozmiarem porcji.
- **WF-09 Monitorowanie celów makroskładnikowych:** System umożliwia zdefiniowanie dziennych limitów kalorycznych, wyznaczeniu makroskładników oraz wizualizuje stopień realizacji celów w czasie rzeczywistym.

Moduł fitnessowy

Moduł przeznaczony do planowania i rejestrowania aktywności fizycznej.

- **WF-10 Baza ćwiczeń:** System udostępnia katalog podstawowych ćwiczeń, skategoryzowanych według typu wysiłku (siłowy, cardio, rozciąganie) oraz partii mięśniowych.

- **WF-11 Zarządzanie ćwiczeniami:** System umożliwia tworzenie nowego ćwiczenia i edycję istniejącego.
- **WF-12 Zarządzanie planami treningowymi:** System umożliwia tworzenie trwałych zestawów treningowych składających się z sekwencji wybranych ćwiczeń oraz planowanie ich na przestrzeni tygodnia.
- **WF-13 Dziennik treningowy:** System pozwala na zapisywanie wyników wykonanych ćwiczeń lub zestawów treningowych w ramach danego dnia.
- **WF-14 Analiza progresji:** System prezentuje historię wyników w formie listy, umożliwiając analizę postępów siłowych w czasie. Wyświetla również podsumowanie dnia z liczbą wykonanych ćwiczeń/serii oraz objętość/przebieg całego dnia.

Moduł finansowy

Moduł wspierający zarządzanie budżetem i monitorowanie transakcji, oparty na cyklach miesięcznych.

- **WF-15 Rejestracja operacji finansowych:** System umożliwia dodawanie transakcji finansowych, definiując ich nazwę, typ operacji (przychód/wydatek), kwotę, datę, kategorię oraz (opcjonalny) opis.
- **WF-16 Zarządzanie kategoriami transakcji:** System pozwala na przypisywanie kategorii do przychodów/wydatków oraz zestawianie ze sobą sumarycznych kwot z nimi powiązanych.
- **WF-17 Miesięczne planowanie budżetu:** System umożliwia definiowanie limitów wydatków (budżetów) dla poszczególnych kategorii w ramach wybranego miesiąca oraz oblicza globalny limit wydatków jako sumę budżetów wszystkich kategorii.
- **WF-18 Analiza i wizualizacja finansowa:** System prezentuje zestawienia dla wybranego miesiąca, oblicza bilans oraz wyświetla filtrowalną listę wszystkich zarejestrowanych transakcji.

Moduł zadaniowy

Moduł organizacji pracy i planowania zadań.

- **WF-19 Tablica zadań:** System prezentuje zadania w widoku kolumnowym, umożliwiając filtrowanie wyświetlanych kategorii oraz ukończonych i nieukończonych zadań.

- **WF-20 Definicja zadania:** System umożliwia tworzenie zadań zawierających treść, priorytet, (opcjonalny) termin realizacji oraz status dla konkretnych kategorii.
- **WF-21 Personalizacja widoku zadań:** System umożliwia użytkownikowi dodawanie i edycję nazw kolumn (kategorii) na tablicy zadań.
- **WF-22 Archiwizacja zadań:** System zapewnia mechanizm przenoszenia zrealizowanych zadań do widoku historycznego, usuwając je z głównego widoku.
- **WF-23 Usuwanie zadań:** System umożliwia usunięcie archiwizowanych wcześniej zadań.

Pulpit

Moduł integrujący i prezentujący kluczowe wskaźniki systemu.

- **WF-24 Agregacja danych:** System udostępnia panel główny zawierający podsumowania (w formie widgetów) z wybranych przez użytkownika modułów.
- **WF-25 Adaptacyjny układ interfejsu:** System dynamicznie dostosowuje układ widgetów na panelu w zależności od dostępności danych i ustawień użytkownika.

2.3. Wymagania niefunkcjonalne

Poniższa sekcja definiuje wymagania niefunkcjonalne mające na celu zapewnienie satysfkacji użytkownika, stabilności rozwiązania oraz podatności na dalszy rozwój. Każde z nich opatrzone jest unikalnym identyfikatorem w formacie **WN-XX**.

Wydajność i efektywność

System musi charakteryzować się wysoką responsywnością i optymalizacją zasobów, aby interakcja z użytkownikiem była płynna.

- **WN-01 Czas reakcji interfejsu:** Czas odpowiedzi interfejsu użytkownika na interakcje (kliknięcia, przejścia między widokami) powinien być minimalny dla operacji lokalnych, aby zapewnić wrażenie natychmiastowości działania.
- **WN-02 Płynność renderowania:** Animacje interfejsu oraz przejścia między stanami muszą być renderowane w wysokiej częstotliwości odświeżania, eliminując zjawisko zacinania się obrazu.

- **WN-03 Optymalizacja ładowania zasobów:** Czas pierwszego wyrenderowania treści dla głównego panelu powinien wynosić maksymalnie kilka sekund dla standardowego połączenia, a w podrzędnych modułach powinno być zastosowane leniwe ładowanie.

Użyteczność i estetyka (UX/UI)

System musi implementować nowoczesne wzorce projektowe maksymalizujące komfort użytkowania i estetykę stylistyczną.

- **WN-04 Spójność wizualna:** Interfejs graficzny musi utrzymywać spójność stylistyczną opartą na zdefiniowanym designie (paleta barw, odstępy).

- **WN-05 Responsywność strukturalna:** Układ graficzny musi dynamicznie adaptować się do rozdzielczości ekranu urządzenia końcowego, zachowując pełną czytelność i funkcjonalność w zakresach od urządzeń mobilnych po monitory komputerowe.

- **WN-06 Intuicyjność nawigacji:** Dostęp do kluczowych funkcjonalności (np. dodanie wpisu, wyświetlenie podsumowania) nie powinien wymagać więcej niż kilka interakcji z poziomu panelu głównego.

Bezpieczeństwo i integralność danych

Ze względu na przetwarzanie danych osobistych, system musi posiadać solidne mechanizmy ochronne.

- **WN-07 Izolacja danych:** Mechanizmy dostępu do bazy danych muszą gwarantować, że użytkownik posiada dostęp wyłącznie do rekordów, których jest właścicielem (RLS).

- **WN-08 Uwierzytelnianie i sesja:** System musi wykorzystywać bezpieczne tokeny do zarządzania sesją użytkownika z bezpiecznym przechowywaniem po stronie klienta.

- **WN-09 Walidacja danych wejściowych:** Wszystkie wprowadzane przez użytkownika dane podlegają walidacji zarówno po stronie klienta, jak i serwera, aby zapobiec błędom spójności lub atakom.

- **WN-10 Obsługa błędów:** W przypadku wystąpienia błędów po stronie serwera lub utraty połączenia, system musi w czytelny sposób informować użytkownika o stanie aplikacji.

Jakość kodu

Struktura kodu musi umożliwiać łatwą rozbudowę i modyfikacje systemu.

- **WN-11 Separacja logiczna kodu:** Kod źródłowy musi realizować zasadę separacji odpowiedzialności - logika biznesowa, warstwa komunikacji z API i prezentacji muszą być od siebie wyraźnie oddzielone.
- **WN-12 Komponentaryzacja:** Warstwa interfejsu ma być budowana w oparciu o izolowane, sparametryzowane komponenty. Powtarzalne elementy powinny być definiowane raz i wykorzystywane wielokrotnie w celu zachowania spójności i łatwości utrzymania.

Rozdział 3.

Architektura i implementacja

3.1. Charakterystyka środowiska technologicznego

W celu realizacji powyższych wymagań projekt został zrealizowany w modelu *Single Page Application* (**SPA**), opierając się na separacji frontendu od backendu realizowanego w modelu *Backend-as-a-Service* (**BaaS**).

3.1.1. Stos technologiczny

- **Frontend:** Biblioteka **React** [6] wykorzystująca silnik **Vite** [7] jako narzędzie budowania, kod źródłowy napisany w języku **TypeScript** [8]. Warstwa oparta głównie na frameworku **TailwindCSS** [9], wspieranego miejscami przez bibliotekę **HeadlessUI** [10] do obsługi modali oraz **Framer Motion** [11] w celu zapewnienia płynności interfejsu i animacji.

- **Backend:** Platforma **Supabase** [12] pełniąca rolę kompleksowego backendu. Obsługuje procesy autentykacji użytkowników (**Supabase Auth**) oraz dostarcza **Supabase Edge Functions**, pełniących rolę bezpiecznego proxy do zewnętrznych API.

- **Baza danych:** Relacyjna baza danych **PostgreSQL** [13] hostowana w **chmurze Supabase**.

- **Zewnętrzne API:** Integracja z otwartą bazą danych produktów spożywczych **OpenFoodFacts** [14] w celu dostarczenia do aplikacji informacji o makroskładnikach.

- **Narzędzia pomocnicze:** Repozytorium kodu i kontrola wersji w serwisie **GitHub** [15]. Hosting aplikacji w środowisku **GitHub Pages** [16] ze skonfigurowanym mechanizmem SPA fallback.

3.1.2. Warstwa prezentacji

Fundamentem warstwy prezentacji jest biblioteka **React 19**, pozwalająca na budowanie interfejsu użytkownika opartego na komponentach. Wykorzystanie najnowszej (w czasie pisania pracy) stabilnej wersji biblioteki pozwala na optymalizację procesu renderowania oraz efektywne zarządzanie stanem aplikacji.

Ze względu na wieloaspektowy charakter aplikacji (logika dietetyczna, treningowa, finansowa) wykorzystano język **TypeScript**. Zastosowanie typowania statycznego pozwala na eliminację błędów już na etapie kompilacji, ułatwia zrozumienie kodu oraz stanowi dobre źródło wiedzy o przepływających przez system strukturach danych.

Jako środowisko uruchomieniowe i narzędzie budowania wybrano **Vite**. Wyboru dokonano ze względu na lepszą wydajność względem poprzedników (np. *Create React App*) wynikającą z ekspresowego startu serwera i obsługi HMR (*Hot Module Replacement* - aktualizacja kodu w przeglądarce bez odświeżania całej strony).

Warstwa wizualna została zaprojektowana z naciskiem aplikacji na estetykę oraz responsywność (RWD) do różnych rozdzielczości ekranów. Do stylowania komponentów wykorzystano framework typu *utility-first* - **Tailwind CSS** w wersji v4. Podejście to zrywa ze standardowym modelem separacji struktury (HTML) i stylowania (CSS) na rzecz architektury komponentowej, w której wygląd elementu definiowany jest przez kompozycję klas pomocniczych. Zastosowanie Tailwinda pozwoliło na uproszczenie zachowania spójności wizualnej na poziomie całej aplikacji, dodatkowo nowa architektura silnika wersji v4, oparta na kompilacji JIT (*Just-in-Time*) znacząco przyspiesza proces kompilacji stylów i minimalizuje rozmiar wynikowego pliku CSS.

Za logikę bardziej dynamicznych elementów interfejsu (takich jak na przykład okna modalne) odpowiada biblioteka **Headless UI**, użyta, aby przyspieszyć prace deweloperskie. W celu wzbogacenia doświadczeń użytkownika, warstwa prezentacji została uzupełniona o bibliotekę **Framer Motion**, odpowiadającą za płynne animacje i przejścia między widokami. Całość dopełnia ikonografia oparta na zestawie *Material Symbols Sharp* [17], zapewniająca czytelność i nowoczesny wygląd.

Nawigacja wewnątrz aplikacji realizowana jest przez bibliotekę **React Router DOM** [18]. Pozwala ona na dynamiczne przełączanie widoków pomiędzy modułami bez konieczności przeładowywania strony, zachowując stan aplikacji i zapewniając płynność nawigacji.

3.1.3. Warstwa backendowa i infrastruktura danych

Zrezygnowano z budowy i utrzymania tradycyjnego serwera aplikacyjnego na rzecz architektury *Serverless* i modelu *Backend-as-a-Service* (**BaaS**). Pozwoliło to

na skupienie prac na logice biznesowej aplikacji.

Backend jest obsługiwany przez platformę **Supabase**, dostarczająca relacyjną bazę danych **PostgreSQL**. W przeciwieństwie do rozwiązań NoSQL, PostgreSQL zapewnia możliwość tworzenia zaawansowanych relacji i silną spójność danych. Platforma integruje również gotowy system uwierzytelniania użytkowników (**Supabase Auth**) zabezpieczający dostęp do danych prywatnych na poziomie wiersza w bazie danych (**RLS** - *Row Level Security*).

Do komunikacji z zewnętrznymi serwisami wykorzystano **Supabase Edge Functions**, czyli bezserwerowe funkcje uruchamiane w środowisku uruchomieniowym **Deno**. Funkcje te pełnią rolę bezpiecznego **API Proxy/Gateway**. W projekcie obsługują one zapytania do zewnętrznego API **OpenFoodFacts**, rozwiązując jednocześnie problem polityki **Same-Origin Policy (CORS)** (uniemożliwiający bezpośrednie odpytanie serwisów z poziomu przeglądarki). Dodatkowo ta architektura pozwala na transformację surowych danych z zewnątrz do jednolitego formatu otrzymywanego przez frontend, odciążając aplikację kliencką z logiki przetwarzania danych.

3.1.4. Bezpieczeństwo i kontrola dostępu

Mechanizm uwierzytelniania oparto na usłudze **Supabase Auth**, która odpowiada za bezpieczne zarządzanie tożsamością i sesjami użytkowników. Integracja z usługą realizowana jest za pośrednictwem dedykowanej biblioteki klienckiej, która pozwala na komunikację z API autoryzacyjnym. Odpowiada ona za automatyczne odświeżanie tokenów dostępowych (JWT) oraz bezpieczne przechowywanie sesji w pamięci przeglądarki.

Po stronie aplikacji klienckiej stan uwierzytelnienia jest monitorowany i udostępniany globalnie za pomocą **Context API**, co zapewnia spójność danych o użytkowniku w całym systemie.

Kluczowym elementem ochrony zasobów jest `ProtectedRoute`, pełniący rolę strażnika dostępu do prywatnych sekcji aplikacji (takich jak pulpit czy moduły funkcjonalne). Weryfikuje on aktywność sesji przy każdej próbie nawigacji, a w przypadku wykrycia braku uprawnień automatycznie przekierowuje użytkownika do panelu logowania.

3.1.5. Wdrożenie i hosting

Jako platformę hostingową dla wersji produkcyjnej wybrano **GitHub Pages**, pozwalającą na darmowe i zautomatyzowane udostępnianie aplikacji bezpośrednio z repozytorium kodu.

Proces wdrożenia został zautomatyzowany przy pomocy skryptu zdefiniowanego w `package.json`:

```
"deploy": "gh-pages -d dist"
```

Wywołanie komendy `npm run deploy` powoduje zbudowanie aplikacji, a następnie wysłanie zawartości katalogu `dist` na dedykowaną gałąź `gh-pages`, z której GitHub serwuje stronę.

Obsługa routingu

Specyficzną cechą hostingu plików statycznych (takiego jak GitHub Pages) jest brak natywnej obsługi routingu po stronie klienta, co stanowi problem dla aplikacji SPA. Standardowo, przy próbie bezpośredniego wejścia na podstronę (np. `/tasks`) lub odświeżeniu jej, serwer zwraca błąd 404, ponieważ fizyczny plik o takiej ścieżce nie istnieje.

Aby rozwiązać ten problem, zastosowano technikę **SPA Fallback**. Polega ona na utworzeniu specjalnego pliku `404.html`, będącego identyczną kopią `index.html`.

W procesie budowania aplikacji skrypt automatycznie kopiuje `index.html` do `404.html`. Aplikacja działa wtedy w następujący sposób:

1. Użytkownik próbuje uzyskać dostęp do ścieżki `/tasks`.
2. GitHub Pages nie znajduje pliku `tasks/index.html`, więc zgodnie z konfiguracją serwuje plik błędu `404.html`.
3. Ponieważ `404.html` zawiera ten sam kod co główna aplikacja, ładuje się silnik Reacta oraz router.
4. Router po stronie klienta odczytuje aktualny adres URL z paska przeglądarki i renderuje odpowiedni widok zamiast strony błędu.

Dzięki temu zabiegowi użytkownik ma wrażenie płynnego działania aplikacji, a nawigacja działa poprawnie również przy bezpośrednich odwołaniach do podstron.

3.2. Architektura systemu

3.2.1. Struktura logiczna i organizacja modułów

Architektura aplikacji klienckiej została zaprojektowana z myślą o skalowalności i łatwości utrzymania, stosując podział kodu względem funkcjonalności (ang. *Feature-Based Folder Structure*). Kod źródłowy w katalogu `src/components` podzielony jest na autonomiczne moduły:

- **Calories** - moduł dietetyczny (dziennik kalorii i spożytych makroskładników, baza produktów, cele żywieniowe).
- **Fitness** - dziennik aktywności fizycznej (zarządzanie planami i sesjami treningowymi, tworzenie zestawów treningowych, śledzenie postępów).
- **Finance** - moduł budżetowy (budżetowanie, kontrola przepływów finansowych, kategoryzacja przychodów i wydatków).
- **Tasks** - system zarządzania zadaniami (*To-Do* listy, priorytetyzacja, kategoryzacja, planowanie zadań w czasie).

Każdy z modułów posiada odizolowaną, wewnętrzną strukturę, na którą składają się foldery:

- `components/` - dedykowane komponenty i widoki specyficzne dla danego modułu.
- `hooks/` - logika biznesowa i zapytania do API, wydzielone do niestandardowych hook-ów.
- `types/` - definicje typów odzwierciedlające modele danych specyficzne dla domeny.

Elementy współdzielone, takie jak generyczne komponenty (przyciski, modale), zostały wydzielone do katalogu `src/components/UI`.

3.2.2. Model danych i schemat relacyjny

Schemat bazy danych został zaprojektowany w celu zapewnienia integralności danych oraz optymalizacji zapytań dla poszczególnych modułów. Na skutek integracji z modułem Supabase Auth dane uwierzytelniające przechowuje tabela systemowa `auth.users`.

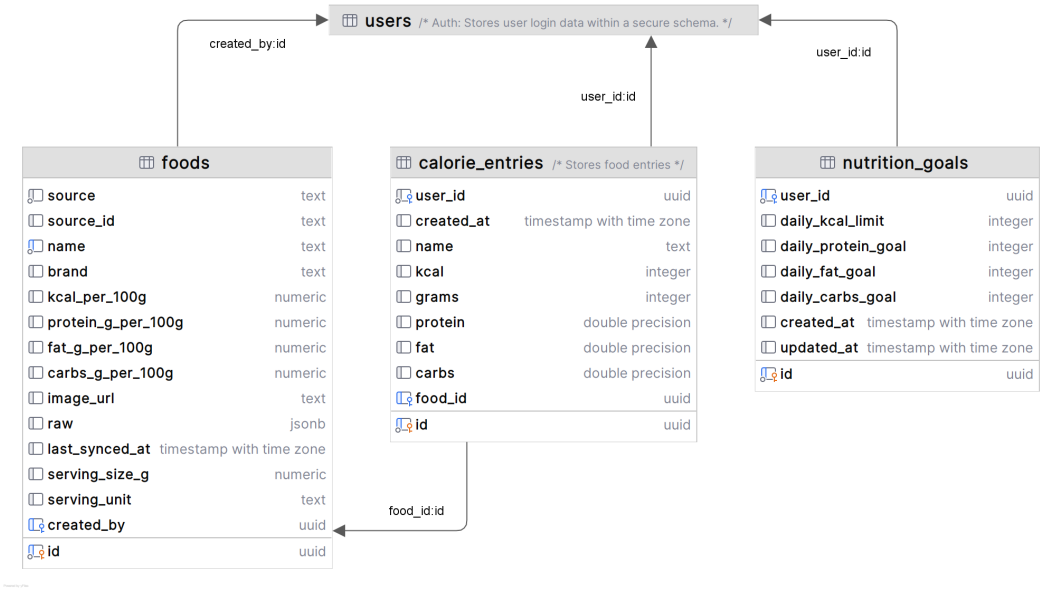
W celu zachowania przejrzystości, poniższe schematy bazy danych przedstawiają tabelę `auth.users` w uproszczonej formie, skupiając się na jej relacjach z zaprojektowanymi strukturami. Zdefiniowane tabele objęto politykami bezpieczeństwa **RLS** (*Row Level Security*), zapewniającymi dostęp użytkownika wyłącznie do własnych danych.

Domena dietetyczna

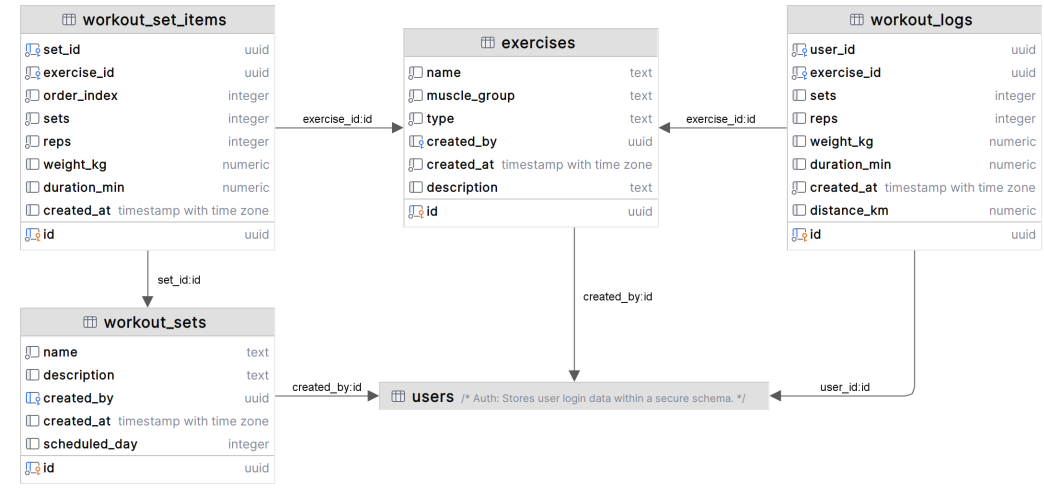
- `foods` - lokalna baza produktów spożywczych, pełniąca również rolę pamięci podręcznej (*cache*) dla produktów wyszukanych wcześniej w zewnętrznym API.

- `calorie_entries` - dziennik żywieniowy rejestrujący spożycie produktów w czasie, zawierający przeliczone wartości makroskładników w oparciu o spożytą gramaturę.

- `nutrition_goals` - tabela konfiguracyjna przechowująca spersonalizowane cele żywieniowe (limity kalorii i makroskładników).



Rysunek 3.1: Tabele powiązane z modułem dietetycznym



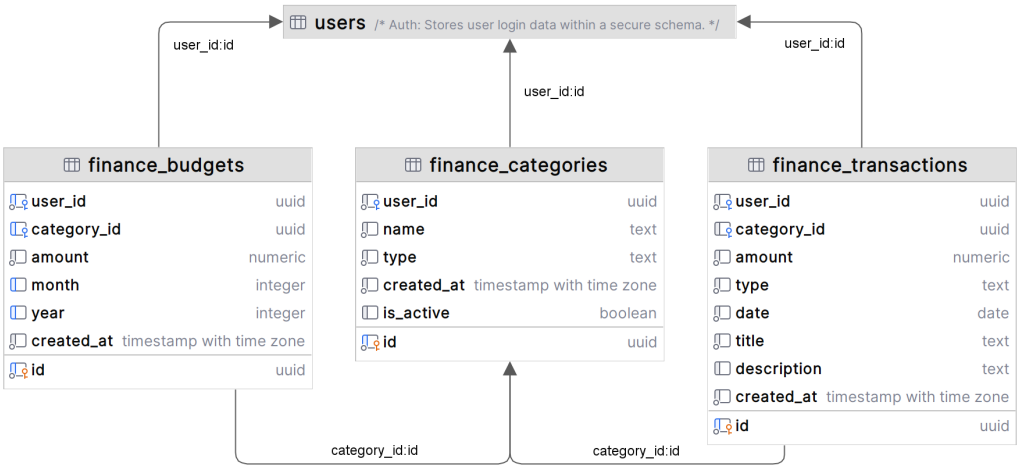
Rysunek 3.2: Tabele powiązane z modułem fitnessowym

Domena fitnessowa

- `exercises` - katalog dostępnych ćwiczeń (dostępnych domyślnie i dodanych przez użytkowników).
- `workout_logs` - rejestr pojedynczych wpisów treningowych.
- `workout_sets` - nagłówki zestawów treningowych, grupujących wybrane ćwiczenia.
- `workout_set_items` - tabela przechowująca parametry zestawów ćwiczeń (jakie i ile ćwiczeń wchodzi w skład zestawu, parametry każdego z nich).

Domena finansowa

- `finance_categories` - słownik kategorii przychodów i wydatków.
- `finance_transactions` - główny rejestr operacji finansowych przechowujący kwotę, typ operacji, datę i (opcjonalnie) opis.
- `finance_budgets` - tabela definiująca limity wydatków dla poszczególnych kategorii w cyklach miesięcznych.



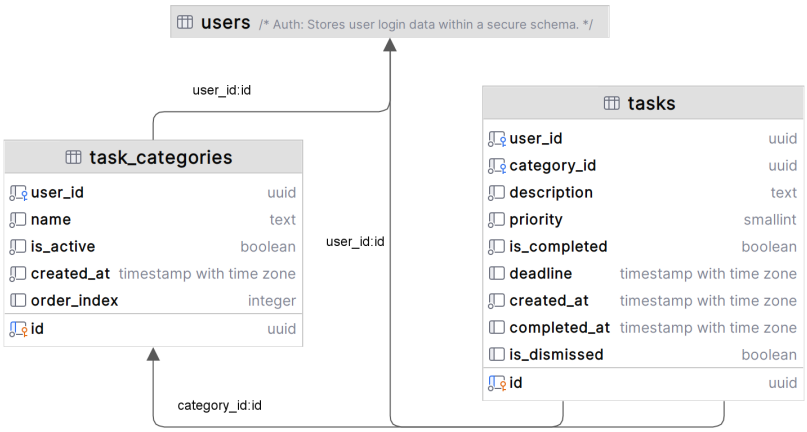
Rysunek 3.3: Tabele powiązane z modułem finansowym

Domena zadaniowa

- `tasks` - encje poszczególnych zadań zawierające priorytet, status wykonania, termin realizacji oraz klucz obcy do kategorii.
- `task_categories` - grupy zadań (listy) definiowane przez użytkownika.

Domena użytkownika

- `profiles` - dane (nazwa użytkownika) i preferencje użytkownika (moduły aktywne w aplikacji).



Rysunek 3.4: Tabele powiązane z modułem zadaniowym

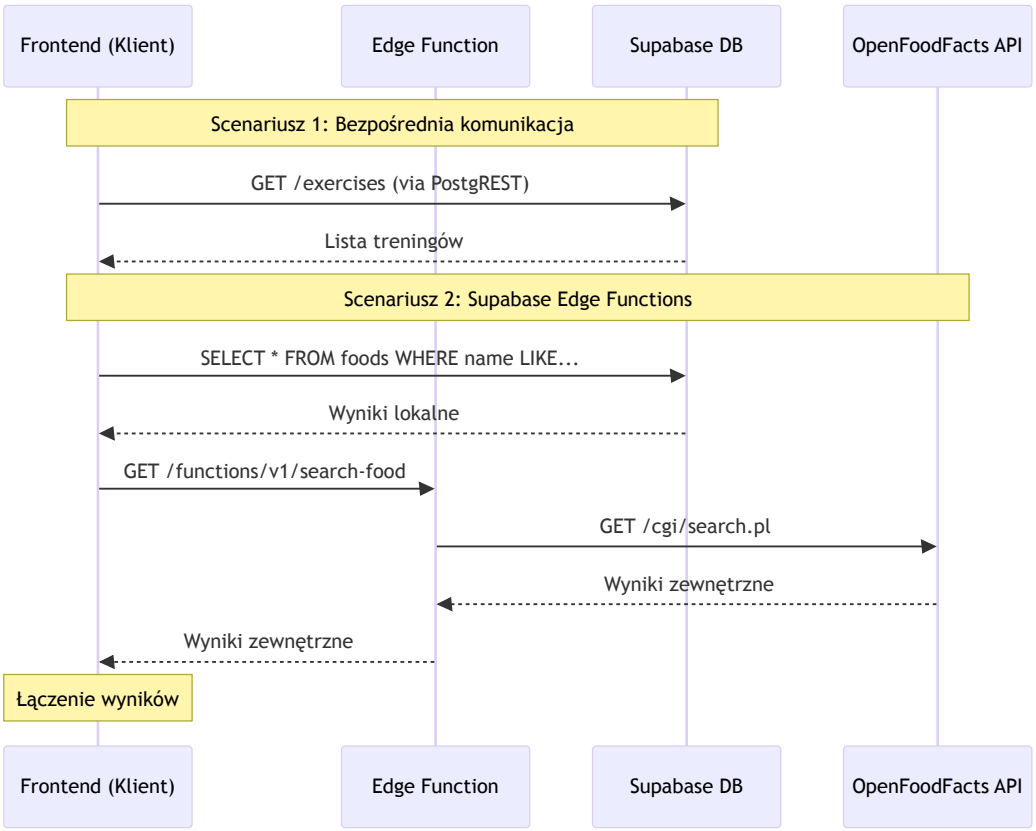
3.2.3. Komunikacja i interfejs API

Aplikacja wykorzystuje dwa główne kanały komunikacji z warstwą danych:

- **Bezpośrednia komunikacja z bazą danych (PostgREST):** Większość operacji na danych (CRUD) realizowana jest bezpośrednio z poziomu klienta przy użyciu biblioteki `@supabase/supabase-js`. Komunikuje się ona z warstwą PostgREST, która automatycznie wystawia bezpieczne API REST-owe na podstawie schematu bazy danych.
- **Supabase Edge Functions (Serverless):** Dla operacji wymagających logiki po stronie serwera lub integracji z zewnętrznymi serwisami wykorzystano funkcje bezserwerowe.

Praktycznym przykładem zastosowania architektury *Serverless* w projekcie jest funkcja `search-food` działająca jako pośrednik (*proxy*) do zewnętrznego API OpenFoodFacts. Proces ten, zilustrowany na rys. 3.5, przebiega w następujących krokach:

1. Aplikacja wysyła zapytanie do funkcji `search-food`.
2. Funkcja przeszukuje lokalną bazę danych (`foods`) w celu znalezienia zapisanych wcześniej produktów (*cache*).
3. Funkcja odpytuje zewnętrzne API.
4. Agregowane z lokalnej bazy danych i API dane są zwracane do klienta.



Rysunek 3.5: Diagram sekwencji procesu wyszukiwania produktów z wykorzystaniem Edge Functions

Rozdział 4.

Przegląd funkcjonalności systemu

4.1. Logowanie i rejestracja

Aby uzyskać dostęp do funkcjonalności systemu, konieczne jest założenie konta w aplikacji. Proces rozpoczęcia pracy z aplikacją został zaprojektowany tak, aby był jak najbardziej intuicyjny dla nowego użytkownika.

Na ekranie startowym widoczne są formularze **logowania** (rys. 4.1a) oraz **rejestracji** (rys. 4.1b). Nowy użytkownik wybiera opcję rejestracji, gdzie proszony jest o podanie adresu email oraz utworzenie bezpiecznego hasła (składającego się z minimum 6 znaków). System weryfikuje poprawność wprowadzonych danych, a przycisk zatwierdzający formularz pozostaje nieaktywny do momentu spełnienia wszystkich wymogów, aby zapobiec pomyłkom.

(a) Formularz logowania

(b) Formularz rejestracji

Rysunek 4.1: Formularze autentykacji użytkownika

Rysunek 4.2: Pulpit z aktywowanymi wszystkimi dostępnymi modułami oraz domyślnie widocznym paskiem bocznym (desktop)

Po pomyślnej rejestracji następuje automatyczne zalogowanie i przekierowanie do głównego widoku aplikacji - **pulpitu** (rys. 4.2). Manualne wylogowanie możliwe jest dzięki przyciskowi obok nazwy użytkownika na pasku bocznym. Po jego naciśnięciu, użytkownik wraca do panelu logowania, skąd w przypadku chęci powrotu do aplikacji musi wprowadzić poprawne dane uwierzytelniające do formularza i zalogować się ponownie.

4.2. Pulpit i system nawigacji

Pulpit oraz panel boczny stanowią centralne punkty interakcji, odpowiadające za aranżację widoków oraz prezentację danych w przekrojowej formie, adaptując się jednocześnie do preferencji użytkownika.

Nawigacja i dostosowanie widoku

Nawigacja w aplikacji realizowana jest głównie przez panel boczny, pełniący funkcję kontrolera dostępu do poszczególnych modułów. Na urządzeniach mobilnych jest on chowany i dostępny pod przyciskiem menu w lewym górnym rogu (rys 4.4).

System oferuje możliwość personalizacji, pozwalając użytkownikowi na dostosowanie środowiska pracy do własnych potrzeb poprzez:

- **Konfigurację widoczności modułów**

  Użytkownik może zdecydować, z których funkcjonalności aplikacji chciałby korzystać, używając **panelu konfiguracyjnego** (rys. 4.3a). Z jego poziomu możliwe jest wyłączenie lub włączenie poszczególnych modułów, a zmiana jest natychmiastowa - deaktywowany moduł znika zarówno z paska nawigacji, jak i z pulpitu głównego, co pozwala zachować przejrzystość interfejsu.

- **Ustawienie preferencji pulpitu**

  W **menu preferencji** (rys. 4.3b) użytkownik może dostosować szczegółowe zachowanie widgetów (na przykład wybierając, która lista zadań ma być domyślnie wyświetlana na pulpicie po uruchomieniu aplikacji) oraz zmienić wyświetlaną na pasku bocznym nazwę użytkownika.

Jeśli użytkownik wyłączy wszystkie moduły, pulpit pokaże komunikat z instrukcją konfiguracji (rys. 4.5), zapobiegając wyświetleniu pustego ekranu.

(a) Panel konfiguracyjny (b) Menu preferencji użytkownika

Rysunek 4.3: Okna modalne służące do personalizacji aplikacji

Widgety pulpitu

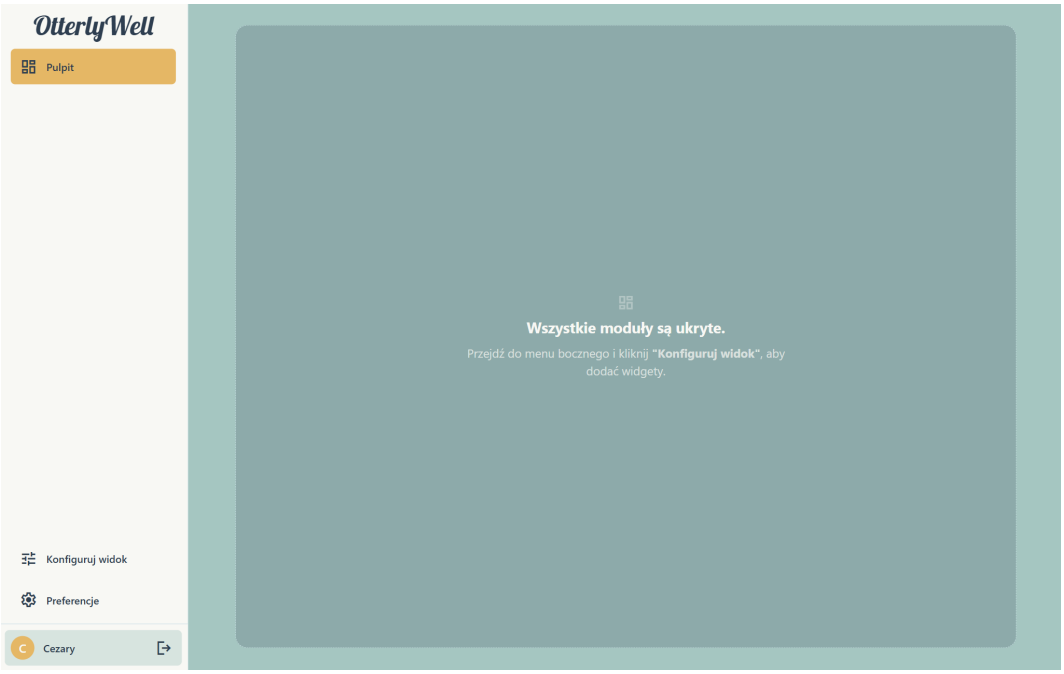
Pulpit składa się z widgetów - komponentów podsumowujących dane znajdujące się w poszczególnych modułach. Aby prezentować je w estetyczny sposób, widok został zaprojektowany w formie "układu cegłowego" (rys. 4.2). Na smartfonach elementy układają się w jedną kolumnę (rys. 4.4b), natomiast na szerszych ekranach tabletów system dynamicznie dopasowuje je do dwóch (rys. 4.4a).

Celem widgetów nie jest zastąpienie pełnej funkcjonalności danego modułu, lecz dostarczenie najważniejszych jego danych (takich jak paski postępu spożycia

makroskładników czy zaplanowany na dziś trening) na jednym ekranie. Pełnią również rolę odnośnika do podstron bezpośrednio z pulpitu, bez potrzeby otwierania paska bocznego na urządzeniach mobilnych.

(a) Ekran tabletu (b) Ekran smartfona

Rysunek 4.4: Pulpit aplikacji na urządzeniach mobilnych



Rysunek 4.5: Pulpit z wyłączonymi wszystkimi modułami (desktop)

Rysunek 4.6: Główny widok modułu śledzenia kalorii (desktop)

4.3. Moduły aplikacji

4.3.1. Moduł śledzenia kalorii

Moduł śledzenia kalorii (rys. 4.6) stanowi narzędzie do monitorowania bilansu energetycznego oraz makroskładników.

Wyszukiwanie i dodawanie produktów

Podstawową funkcją modułu jest **dziennik żywieniowy**. Aby dodać istniejący produkt do dziennika, użytkownik korzysta z paska wyszukiwania. System przeszukuje zarówno lokalną bazę produktów (rozszerzaną przez użytkowników dodających produkty z globalnej bazy), jak i zewnętrzną, publiczną bazę produktów OpenFoodFacts. Wyszukiwanie może odbywać się na dwa sposoby:

1. **Tekstowo**: Wpisując część lub całość nazwy produktu (np. ”Czekolada”).
2. **Kodami kreskowymi**: Korzystając z funkcji automatycznie rozpoznającej konkretny produkt na podstawie kodu z opakowania.

Po wybraniu produktu użytkownik określa spożytą gramaturę, a system automatycznie przelicza kalorie oraz makroskładniki (białko, tłuszcze, węglowodany) proporcjonalnie do podanej wagi. Dodany wpis pojawia się w historii posiłków, co aktualizuje informacje widoczne w panelu podsumowującym.

Zarządzanie listą produktów

Lista spożytych produktów wizualizuje nie tylko nazwę i kaloryczność każdego z nich, ale także szczegółowy rozkład makroskładników w formie miniaturowego paska oraz wartości liczbowych (B/T/W).

Przyciski akcji (edycja, usuwanie) pojawiają się dopiero po najechaniu kursorem na dany wpis na urządzeniach desktopowych, a w przypadku mobilnych widoczne są one cały czas.

System edycji działa analogicznie do systemu dodawania, automatycznie przeliczając wartości odżywcze w momencie modyfikacji gramatury spożytego produktu, a akcja usunięcia wpisu skutkuje natychmiastową aktualizacją statystyk i bilansu energetycznego.

Definiowanie własnych produktów

W przypadku, gdy szukanego produktu nie ma w bazie, użytkownik może zdefiniować go ręcznie, korzystając z formularza. Aplikacja oferuje tu elastyczność, pozwalając na wprowadzenie danych w dwóch trybach:

- **Na 100g**: Standardowy format, zgodny z tabelami wartości odżywczych na opakowaniach (rys. 4.8a).
- **Na porcję**: Użytkownik definiuje niestandardową porcję (np. "kromka") i jej wagę, co ułatwia późniejsze logowanie tego produktu bez konieczności oszacowywania wagi za każdym razem (rys. 4.8b).

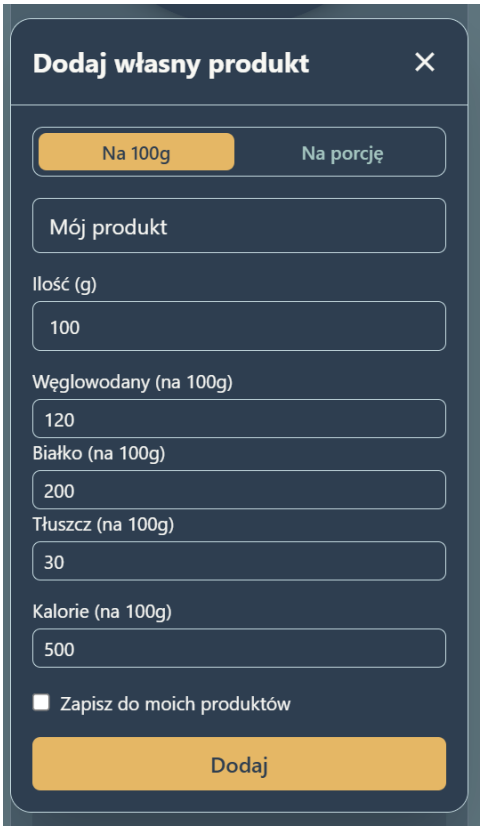
Opcja "Zapisz do moich produktów" pozwala na zachowanie definicji na stałe dla tworzącego użytkownika i używanie jej ponownie w przyszłości.

Cele żywieniowe i wizualizacja

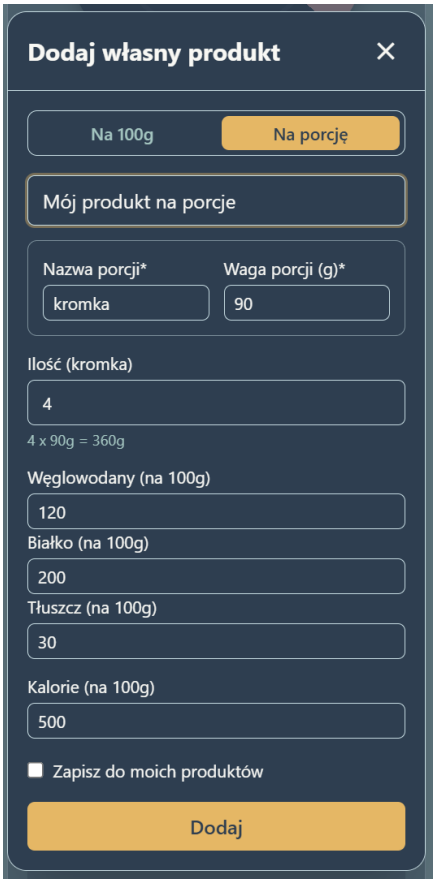
Panel podsumowujący (widoczny u góry rys. 4.6) pozwala na szybki wgląd w bilans planu dietetycznego. Użytkownik może w dowolnym momencie zmodyfikować swoje cele (limit kalorii oraz proporcje makroskładników), klikając ikonę edycji. Za pomocą selektora dnia znajdującego się nad panelem można uzyskać wgląd również w statystyki i historię posiłków z poprzednich dni.

System wizualizuje postępy za pomocą kolorowych pasków (makroskładniki) oraz licznika kołowego (kalorie). Przekroczenie ustalonych limitów jest sygnalizowane zmianą koloru na ostrzegawczy (rys. 4.7), co stanowi jasny sygnał zwrotny dla użytkownika.

Rysunek 4.7: Główny widok modułu śledzenia kalorii po przekroczeniu limitu (desktop)

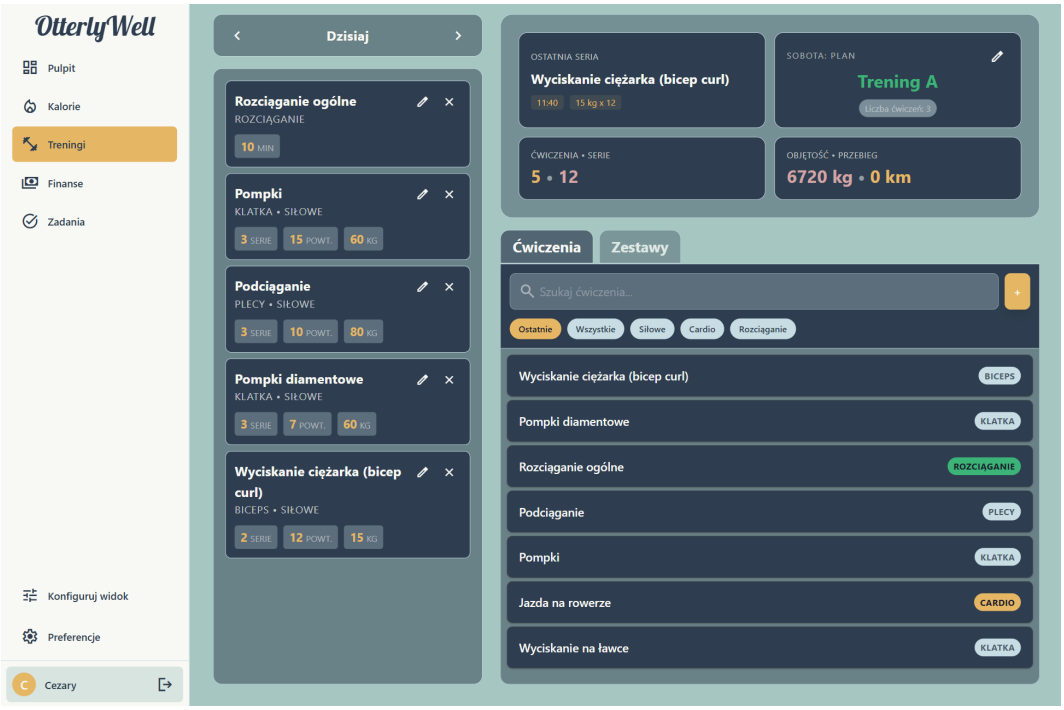


(a) Produkt na 100g



(b) Produkt na porcję

Rysunek 4.8: Formularze dodawania nowego produktu (smartfon)



Rysunek 4.9: Widok modułu fitnessowego z wyświetloną listą dostępnych ćwiczeń (desktop)

4.3.2. Moduł monitorowania aktywności

Moduł fitnessowy (rys. 4.9) umożliwia użytkownikom planowanie oraz rejestrowanie aktywności fizycznej, oferując elastyczne podejście zarówno dla pojedynczych ćwiczeń, jak i złożonych planów treningowych.

Zarządzanie bazą ćwiczeń

System oferuje dostęp do hybrydowej bazy ćwiczeń, składającej się z predefiniowanego katalogu ćwiczeń domyślnych oraz prywatnych, zdefiniowanych przez użytkownika aktywności. Można je wyszukiwać i filtrować według typów (siłowe, cardio lub rozciąganie).

W przypadku braku wprowadzonej frazy lub wpisania mniej niż 2 znaków do paska wyszukiwania, system analizuje historię treningową zalogowanego użytkownika i pobiera ostatnio wykonywane ćwiczenia. W przeciwnym wypadku przeszukuje bazę i zwraca ćwiczenie, jeśli znajdzie je wśród listy domyślnych lub zdefiniowanych przez szukającego.

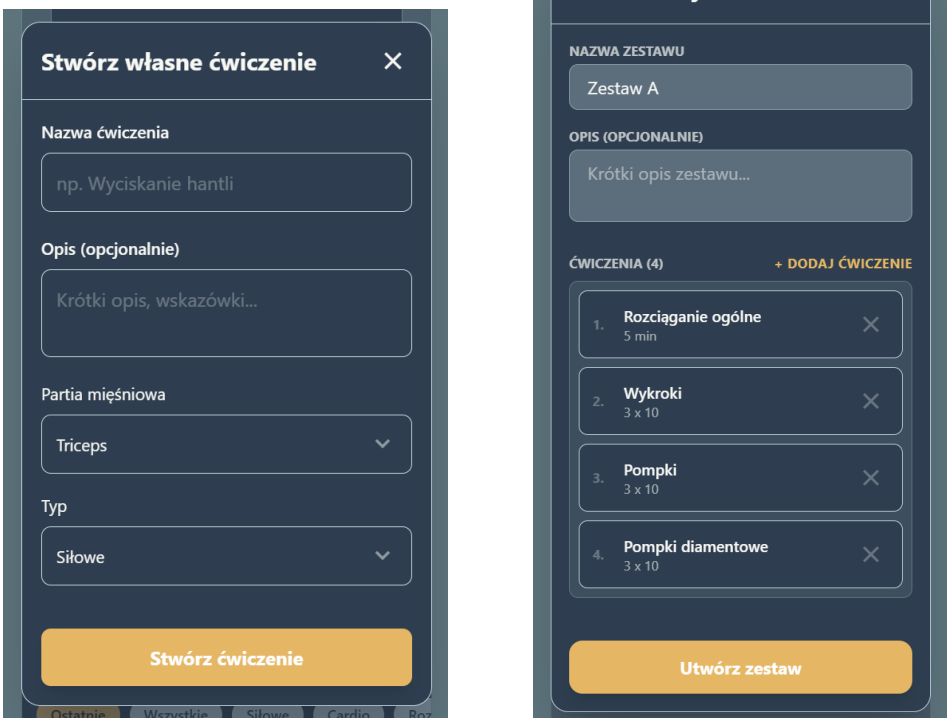
Tworząc własne ćwiczenie, użytkownik nadaje mu nazwę, (opcjonalnie) opis oraz określa typ, a w przypadku ćwiczeń siłowych musi zostać podana również partia mięśniowa, której aktywność dotyczy.

Autorskie ćwiczenia można edytować lub usunąć, natomiast domyślnie dostępne ćwiczenia nie pozwalają na ich modyfikację, ponieważ mają być prostym punktem startowym dla każdego użytkownika.

Zestawy treningowe

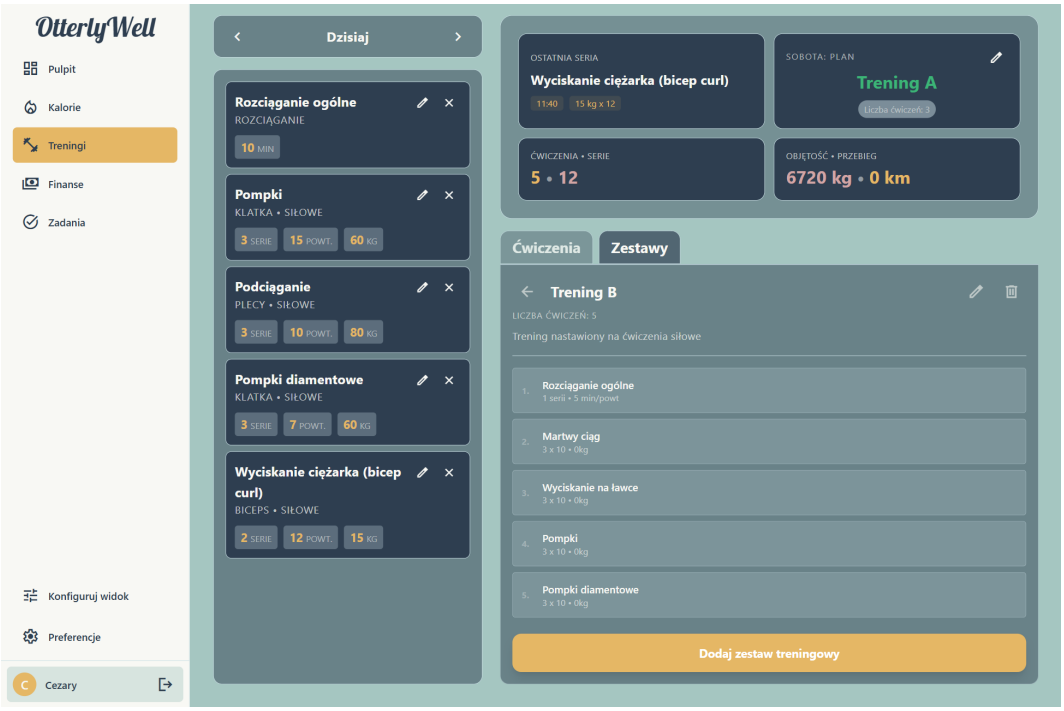
Aby usprawnić codzienne logowanie powtarzalnych treningów, system umożliwia tworzenie **zestawów treningowych**. W zakładce "Zestawy" użytkownik może skomponować własny plan (np. "Trening B"), dodając do niego dowolną listę ćwiczeń wraz z domyślnymi parametrami (liczba serii, powtórzeń, ciężar, dystans, czas) i opcjonalny opis z np. dodatkowymi instrukcjami.

Po utworzeniu, zestaw odnaleźć można na liście, gdzie po kliknięciu na niego ujawniają się zdefiniowane wcześniej szczegóły. Z tego poziomu można edytować lub usunąć zestaw, a jego logowanie odbywa się jednym kliknięciem ("Dodaj zestaw treningowy"), które automatycznie dodaje do dziennika całą listę ćwiczeń, oszczędzając czas użytkownika. Po dodaniu, każdy element może być **niezależnie edytowany** - użytkownik może zmienić ciężar albo przebiegnięty dystans w konkretnym wpisie bez wpływu na definicję całego szablonu.



(a) Dodawanie ćwiczenia (b) Dodawanie zestawu

Rysunek 4.10: Formularze dodawania nowych aktywności fizycznych (smartfon)



Rysunek 4.11: Widok modułu fitnessowego z wyświetlonymi szczegółami zestawu ćwiczeń (desktop)

Planowanie i analiza

System pozwala na utworzenie harmonogramu treningów w postaci **planera tygodniowego**. Użytkownik może przypisać zestawy treningowe do konkretnych dni tygodnia, dzięki czemu, wchodząc do aplikacji danego dnia na pulpicie oraz w module treningowym widoczna jest informacja o zaplanowanej aktywności.

Panel analityczny (widoczny u góry rys. 4.11) agreguje dane z wykonanych danego dnia ćwiczeń, prezentując kluczowe wskaźniki:

- **Objętość**: Sumaryczna objętość (serie × powtórzenia × ciężar) dla ćwiczeń siłowych.
- **Intensywność**: Liczba wykonanych ćwiczeń i serii.
- **Przebieg**: Dystans pokonany w ramach ćwiczeń cardio.

Za pomocą selektora dnia znajdującego się nad panelem można uzyskać wgląd również w statystyki i historię treningów wcześniejszych dni.

4.3.3. Moduł zarządzania finansami

Moduł finansowy (rys. 4.12) umożliwia monitorowanie przepływów pieniężnych oraz planowanie budżetu. Widok domyślnie prezentuje zestawienie bieżącego mie-

Rysunek 4.12: Widok modułu finansowego z aktywnym filtrem na historii transakcji (desktop)

siąca, ale za pomocą selektora daty użytkownik może swobodnie przeglądać historię z poprzednich okresów.

Rejestrowanie transakcji

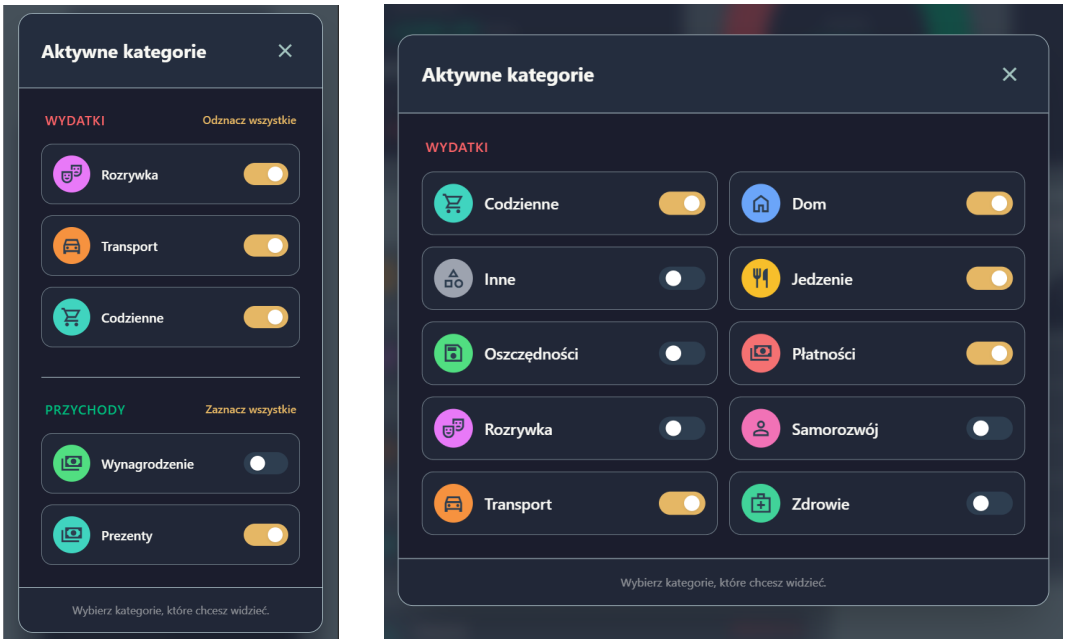
Dodawanie przychodów i wydatków dostępne jest za pomocą formularza otwieranego w górnej części ekranu (rys. 4.14a). W oknie dodawania transakcji użytkownik określa kwotę, tytuł, kategorię oraz datę i (opcjonalny) opis.

Zapisana transakcja trafia na listę **historii operacji**. Aby ułatwić analizę wydatków, lista ta wyposażona jest w mechanizm grupowania chronologicznego oraz filtry (rys. 4.13a), pozwalające na ukrycie wybranych kategorii i wyświetlenie tylko konkretnych obszarów finansowych.

Kliknięcie na transakcję rozwija jej opis i ujawnia przyciski edycji oraz usuwania. W przypadku zmiany kwoty lub kategorii wpisu podczas edycji lub usunięcia go z listy, powiązane sumy automatycznie aktualizują się, co zachowuje spójność wszystkich danych.

Planowanie budżetu

Moduł udostępnia możliwość definiowania **miesięcznych limitów** wydatków (”budżetów”). W planerze budżetów użytkownik może dla każdej kategorii ustalić



(a) Filtrowanie historii transakcji (smartfon)

(b) Dostosowywanie budżetowanych kategorii wydaktów (desktop)

Rysunek 4.13: Okna zarządzania aktywnymi kategoriami finansowymi

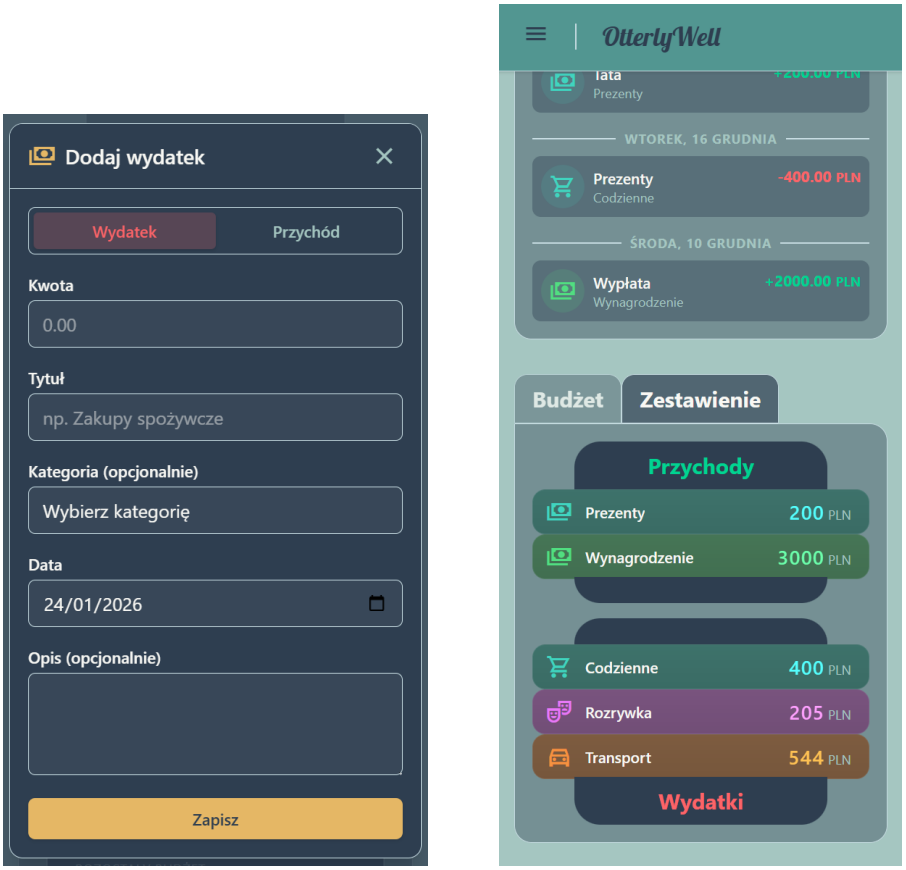
maksymalną kwotę, jaką planuje na nią wydać w danym miesiącu. Może również ukryć niechciane kategorie, aby do całkowitej sumy w tym zestawieniu podliczać tylko interesujące go obszary.

System na bieżąco monitoruje realizację tych założeń. Sumaryczny budżet wyświetlanych kategorii oraz stopień jego zapełnienia reprezentuje pasek widoczny nad listą. Kwoty w każdym wierszu listy wizualnie wskazują, ile środków z zaplanowanej puli zostało już wykorzystanych, a jeśli suma wydatków w danej kategorii przekroczy ustalony na nią limit, zmienia ona kolor na ostrzegawczy, co pozwala szybko zidentyfikować obszary wymagające oszczędności.

Analiza finansowa

Poza wspomnianym wcześniej planerem budżetów, funkcję paneli analitycznych pełnią również **panel statystyk** (widoczny w górnej części głównego widoku na rys. 4.12) oraz **struktura wydatków/przychodów** (rys. 4.14b). Kluczowe wskaźniki prezentowane użytkownikowi to:

- **Bilans miesięczny**: Różnica między sumą przychodów a sumą wydatków, przedstawiona w formie licznika kołowego.
- **Pozostałe środki**: Kwota dostępna do wydania w ramach ustalonego budżetu (sumarycznego ze wszystkich kategorii, wliczając te ukryte w planerze).



(a) Modal dodawania nowej transakcji

(b) Struktura wszystkich transakcji

Rysunek 4.14: Widoki mobilne okna modalnego i zestawienia sumarycznego

- **Struktura wydatków/przychodów**: Kwotowy udział poszczególnych kategorii w całości przychodów/wydatków.

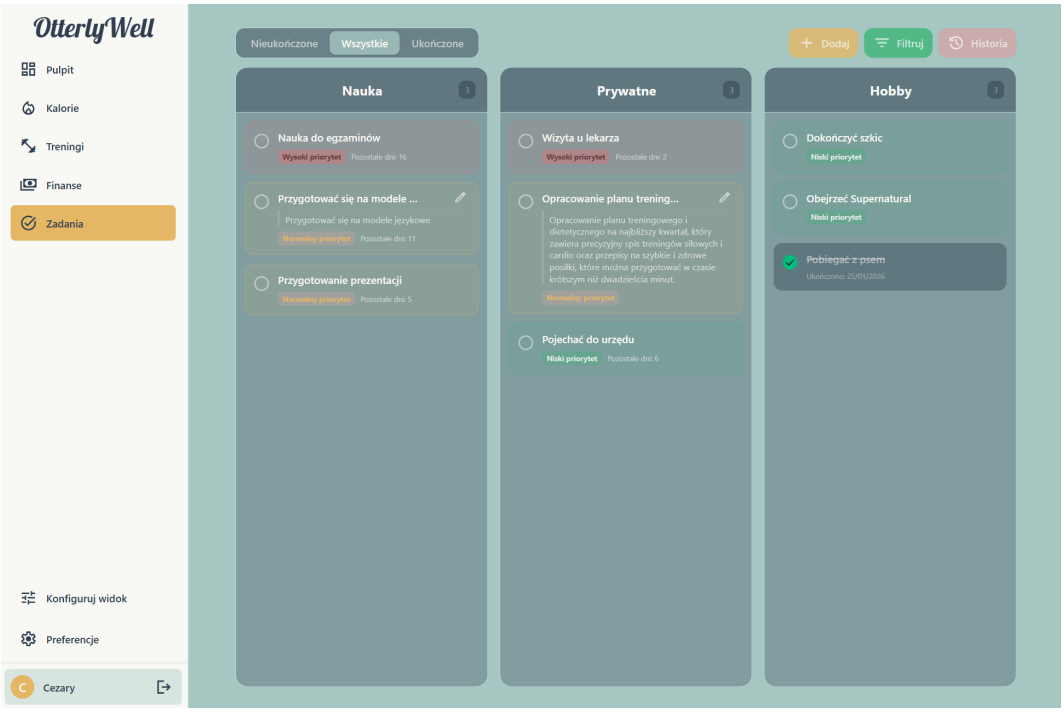
4.3.4. Moduł zarządzania zadaniami

Moduł zadań (rys. 4.15) pełni funkcję osobistego organizera, umożliwiając tworzenie, kategoryzowanie i śledzenie postępów w realizacji obowiązków.

Organizacja pracy i kategorie

Zadania grupowane są w kolumnach (kategoriach), takich jak np. ”Praca”, ”Nauka”. Można je filtrować po statusie, aby zamiast wyświetlać je wszystkie pokazać tylko te ukończone lub nieukończone, a w nagłówku każdej z kategorii znajduje się również licznik zadań, informujący użytkownika o liczbie elementów spełniających aktualne kryteria filtrowania w danej grupie.

Użytkownik ma pełną kontrolę nad strukturą tablicy - może swobodnie dodawać



Rysunek 4.15: Widok główny modułu zadań z aktywnymi trzema kategoriami (desktop)

nowe kategorie, usuwać niepotrzebne, a także zmieniać ich kolejność metodą "przeciągnij i upuść" (*drag-and-drop*) w **menu filtrowania** (rys. 4.18a). Widok tablicy jest elastyczny - na dużych ekranach (rys. 4.15) kategorie wyświetlane są obok siebie i dostosowują szerokość kolumn do ich liczby, natomiast na urządzeniach mobilnych (rys. 4.16a) widoczne są jedna pod drugą.

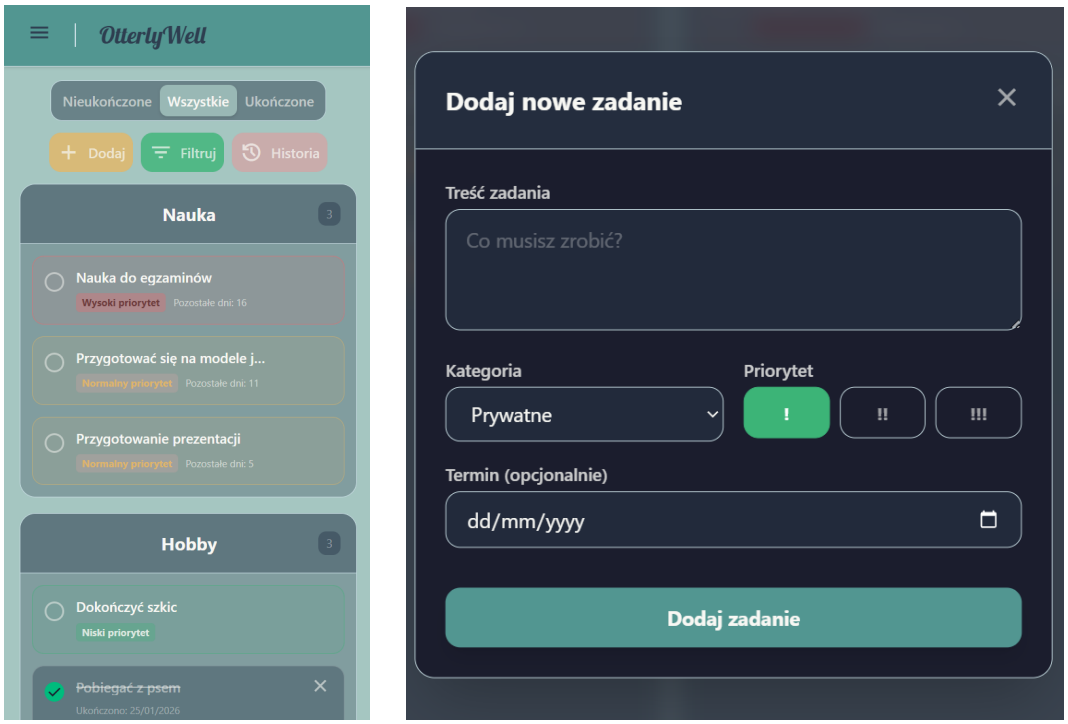
Dla nowych użytkowników przygotowano specjalny stan pusty, który zamiast pustej tablicy wyświetla instrukcje nakierowujące użytkownika na utworzenie pierwszej kategorii i zadania (rys. 4.17).

Zarządzanie zadaniami

Nowe zadanie dodaje się **formularzem** (rys. 4.16b) wywoływanym przez przycisk "Dodaj" u góry widoku, określając w nim jego treść, kategorię, priorytet (Niski/Średni/Wysoki) oraz opcjonalny termin realizacji. Zadania na liście są automatycznie sortowane - te o najwyższym priorytecie wyświetlane są zawsze na szczycie. Interakcja z zadaniem jest intuicyjna:

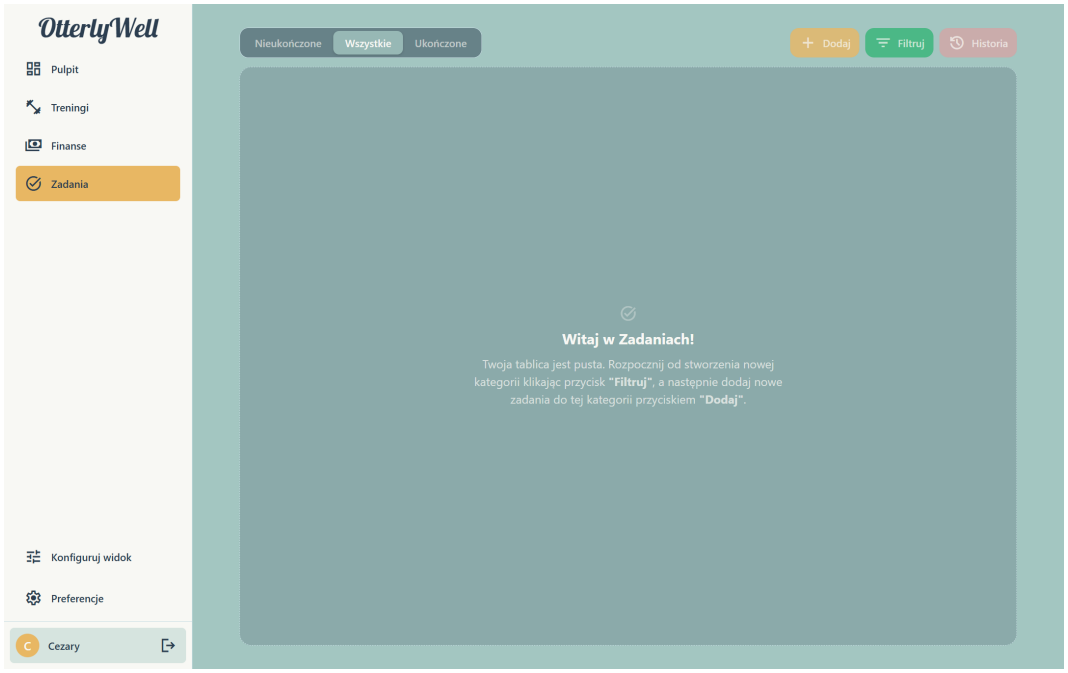
- **Kliknięcie** rozwija pełny opis zadania.
- **Przycisk edycji** pozwala zmienić szczegóły zadania.
- **Przycisk ukończenia** (niewypełnione kółko) zmienia stan między ukończonym/nieukończonym.

Zadania ukończone zostają przekreślone, wyświetlają termin ukończenia i są przeniesione na dół listy, aby nie zasłaniały bieżących obowiązków.



(a) Ekran smartfona (b) Ekran tabletu

Rysunek 4.16: Widok główny i formularz tworzenia nowego zadania na urządzeniach mobilnych



Rysunek 4.17: Widok główny pustego modułu zadań (desktop)

Archiwizacja i historia

Aby utrzymać porządek, ukończone zadania mogą zostać **zarchiwizowane** (usunięte z głównego widoku, ale zachowane w systemie). Dostęp do nich możliwy jest poprzez naciśnięcie przycisku "Historia" znajdującego się u góry ekranu. W **widoku historycznym** (rys. 4.18b) użytkownik może przeglądać swoje przeszłe zadania, wyszukiwać i filtrować je po dacie lub kategorii, usunąć na stałe, a w razie pomyłkowej archiwizacji - przywrócić zadanie z powrotem do widoku głównego.

(a) Menu zarządzania kategoriami (smartfon)

(b) Widok historyczny zadań (tablet)

Rysunek 4.18: Okna zarządzania widocznością oraz stanem kategorii i zadań

Rozdział 5.

Dokumentacja i testy systemu

5.1. Dokumentacja wdrożeniowa

Poniższa dokumentacja opisuje proces instalacji, konfiguracji środowiska uruchomieniowego oraz procedurę wdrożenia aplikacji na serwerze produkcyjnym.

5.1.1. Instrukcja uruchomienia (dla deweloperów)

Aplikacja jest projektem zbudowanym w oparciu o ekosystem Node.js [19]. Aby uruchomić projekt w środowisku lokalnym, wymagane jest posiadanie zainstalowanego środowiska Node.js w wersji 20 (LTS) lub nowszej.

Proces instalacji i uruchomienia przebiega następująco:

1. **Pobranie repozytorium**:

   Należy sklonować kod źródłowy z repozytorium Git pod adresem `https://github.com/czarekmilek/OtterlyWell`.

2. **Instalacja zależności**:

   W katalogu głównym projektu należy następnie wykonać polecenie:

   `npm install`

   Komenda ta pobiera i instaluje wszystkie kluczowe zależności, takie jak `react`, `vite` czy `@supabase/supabase-js` zdefiniowane w pliku `package.json`.

3. **Uruchomienie serwera deweloperskiego**:

   W kolejnym kroku należy uruchomić serwer Vite poleceniem:

   `npm run dev`

   Uruchomiony w ten sposób serwer powinien być domyślnie dostępny pod adresem `http://localhost:5173`.

Aby przygotować wersję zoptymalizowaną do wdrożenia na produkcję, należy wykonać polecenie:

```
npm run build
```

W rezultacie, w katalogu `dist/` wygenerowany zostanie zminifikowany kod, gotowy do umieszczenia na serwerze statycznym.

5.1.2. Konfiguracja środowiska

Aplikacja do poprawnego działania wymaga skonfigurowania zmiennych środowiskowych, pozwalających na bezpieczne połączenie z usługami zewnętrznymi bez zaszywania wrażliwych danych w kodzie źródłowym. Konfiguracja odbywa się poprzez plik `.env` w głównym katalogu projektu. Wymagane zmienne to:

- `VITE_SUPABASE_URL`: Adres URL instancji bazy danych Supabase.
- `VITE_SUPABASE_ANON_KEY`: Publiczny klucz API (anon key) pozwalający na bezpieczną komunikację z usługą Supabase z poziomu przeglądarki.
- `VITE_SUPABASE_FUNCTIONS_URL`: Adres URL zdeployowanych funkcji bezserwerowych (Edge Functions), wykorzystywany do integracji z zewnętrznym API OpenFoodFacts.

Przykładowy plik konfiguracyjny (`.env.example`):

```
VITE_SUPABASE_URL=https://instancja.supabase.co
VITE_SUPABASE_FUNCTIONS_URL=https://instancja.supabase.co/functions
VITE_SUPABASE_ANON_KEY=twoj-publiczny-klucz-api
```

Inicjalizacja bazy danych

Aby aplikacja funkcjonowała poprawnie po stronie serwera, konieczne jest odwzorowanie zaprojektowanego schematu bazy danych w Supabase. W katalogu głównym projektu znajduje się plik `schema.sql`, który zawiera kompletny zestaw instrukcji SQL. Proces inicjalizacji bazy przebiega następująco:

1. Zaloguj się do panelu administracyjnego swojego projektu w Supabase.
2. Przejdź do zakładki **SQL Editor**.
3. Utwórz nowe zapytanie i wklej zawartość pliku `schema.sql`.
4. Uruchom skrypt przyciskiem **Run**.

Skrypt ten automatycznie stworzy wymagane tabele, skonfiguruje relacje między tabelami, aktywuje rozszerzenia oraz nałoży polityki bezpieczeństwa RLS.

5.1.3. Dostępność wersji demonstracyjnej

W celu umożliwienia weryfikacji funkcjonalności systemu bez lokalnej instalacji, aplikacja hostowana jest na Github Pages. Wersja demonstracyjna dostępna jest pod adresem:

`https://czarekmilek.github.io/OtterlyWell/`.

5.2. Weryfikacja i walidacja systemu

Celem procesu testowania była weryfikacja poprawności działania aplikacji, potwierdzenie spełnienia zdefiniowanych wymagań funkcjonalnych i niefunkcjonalnych oraz ocena jakości wytworzonego kodu. Przyjęta strategia zapewnienia jakości systemu obejmuje automatyczne testy kodu źródłowego, manualną weryfikację scenariuszy użytkownika oraz audyty wydajności.

5.2.1. Środowisko i narzędzia testowe

Do realizacji testów wykorzystano następujące narzędzia:

- **Vitest** [20]: Lekki i szybki framework zintegrowany z Vite, służący do wykonywania testów jednostkowych i integracyjnych.
- **React Testing Library** [21]: Biblioteka umożliwiająca testowanie komponentów w sposób symulujący interakcję z użytkownikiem (kliknięcia, wprowadzanie tekstu).
- **Google Lighthouse** [22]: Narzędzie do audytu wydajności, dostępności i optymalizacji aplikacji webowych.

5.2.2. Testy automatyczne

Kluczowa logika aplikacji, w tym mechanizmy przeliczania kalorii, agregacji transakcji finansowych czy zarządzania stanem zadań, została wyizolowana do niestandardowych hooków (np. `useCaloriesData`, `useTransactions`). Przeprowadzone testy jednostkowe koncentrowały się na weryfikacji poprawności zwracanych danych w zależności od stanu wejściowego, zapewniając, że funkcje działają poprawnie.

Testy komponentów weryfikowały natomiast poprawność renderowania interfejsu użytkownika. Sprawdzano, czy kluczowe elementy (przyciski, formularze) pojawiają się na ekranie w odpowiednich momentach cyklu życia aplikacji. Wszystkie przeprowadzone testy zakończyły się wynikiem pozytywnym.

5.2.3. Weryfikacja scenariuszy użytkownika (Testy manualne)

W celu potwierdzenia użyteczności systemu przeprowadzono manualną weryfikację najczęstszych scenariuszy użytkowania. Tabela 5.1 prezentuje wyniki walidacji, potwierdzające, że funkcjonalności dostępne dla końcowego użytkownika działają poprawnie.

Tabela 5.1: Weryfikacja scenariuszy użytkownika

ID	**Nazwa scenariusza**	**Funkcjonalność**
1	Rejestracja i konfiguracja pulpitu	Poprawna
2	Zmiana preferencji użytkownika	Poprawna
3	Śledzenie kalorii	Poprawna
4	Dodawanie własnego produktu	Poprawna
5	Modyfikowanie celów żywieniowych	Poprawna
6	Śledzenie aktywności fizycznej	Poprawna
7	Tworzenie i edytowanie własnych ćwiczeń	Poprawna
8	Tworzenie i zarządzanie zestawami treningowymi	Poprawna
9	Planowanie treningów	Poprawna
10	Rejestracja transakcji finansowych i śledzenie ich historii	Poprawna
11	Planowanie budżetu miesięcznego	Poprawna
12	Organizacja zadań i kategorii	Poprawna
13	Trwałe usuwanie i przywracanie zadań	Poprawna

5.2.4. Weryfikacja stopnia realizacji wymagań

Podsumowaniem procesu testowego jest weryfikacja pokrycia zdefiniowanych na etapie projektowym wymagań funkcjonalnych oraz niefunkcjonalnych. Tabele 5.2 i 5.3 przedstawiają zestawienie tych wymagań wraz z potwierdzeniem ich implementacji w finalnej wersji systemu.

Tabela 5.2: Weryfikacja wymagań funkcjonalnych

ID	**Skrócona nazwa**	**Zgodność**	**Komentarz**
WF-01	Rejestracja i logowanie	Tak	Moduł Auth (Supabase)
WF-02	Zarządzanie sesją i wylogowanie	Tak	AuthContext + RLS
WF-03	Kontrola dostępu do zasobów	Tak	Przez ProtectedRoute
WF-04	Zarządzanie danymi profilowymi	Częściowa	Widok preferencji pozwala na edycję, ale brak opcji zmiany hasła z poziomu aplikacji
WF-05	Wyszukiwanie produktów spożywczych	Tak	OpenFoodFacts API oraz baza wewnętrzna
WF-06	Skanowanie kodów kreskowych	Częściowa	Pobieranie produktu na podstawie kodu działa, ale moduł kamery pozostaje do zaimplementowania
WF-07	Dziennik żywieniowy	Tak	Tabela calorie_entries
WF-08	Definiowanie własnych produktów	Tak	Formularz tworzenia niestandardowego produktu
WF-09	Monitorowanie celów żywieniowych	Tak	Formularz definiowania celów, koło i paski postępu
WF-10	Baza ćwiczeń	Tak	Tabela exercises
WF-11	Zarządzanie ćwiczeniami	Tak	CRUD ćwiczeń
WF-12	Zarządzanie zestawami ćwiczeń	Tak	CRUD zestawów, dedykowany planer tygodnia
WF-13	Dziennik treningowy	Tak	Tabela workout_logs
WF-14	Analiza progresji treningów	Tak	Podliczanie statystyk, historia treningów
WF-15	Rejestracja transakcji	Tak	Formularz dodawania transakcji
WF-16	Zarządzanie kategoriami finansowymi	Tak	Podliczanie kategoryzowanych transakcji w planerze, filtrowanie kategorii w historii
WF-17	Planowanie budżetu	Tak	Planer budżetowy
WF-18	Analiza finansowa	Tak	Podsumowanie danych i bilans
WF-19	Tablica zadań	Tak	Widok kolumnowy z filtrowaniem
WF-20	Definicja zadania	Tak	Formularz tworzenia zadań
WF-21	Personalizacja widoku zadań	Tak	Komponent filtrujący widok
WF-22	Archiwizacja zadań	Tak	Ukończone zadania można przenieść do widoku historycznego
WF-23	Usuwanie zadań	Tak	Przycisk w archiwum
WF-24	Agregacja danych na pulpicie	Tak	Widgety modułowe
WF-25	Adaptacyjny układ interfeju	Tak	Zmienna liczba kolumn w zależności od ekranu

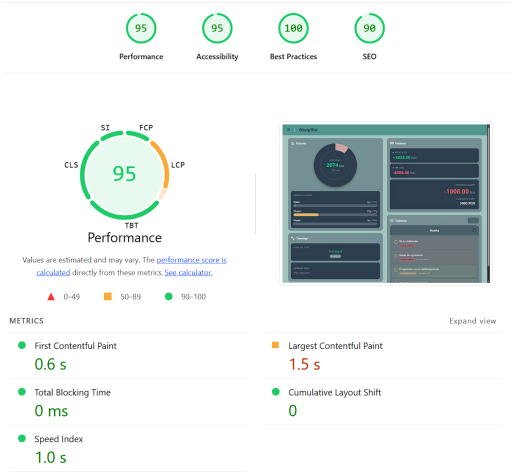
Tabela 5.3: Weryfikacja wymagań niefunkcjonalnych

ID	**Nazwa wymagania**	**Zgodność**	**Komentarz**
WN-01	Czas reakcji interfejsu	Tak	Architektura SPA eliminuje przeładowania strony, zapewniając natychmiastowe przejścia
WN-02	Płynność renderowania	Tak	Biblioteka Framer Motion
WN-03	Optymalizacja ładowania zasobów	Tak	Lazy loading pobiera moduły tylko, gdy są potrzebne, odciążając start aplikacji
WN-04	Spójność wizualna	Tak	Centralnie zdefiniowane zmienne w konfiguracji Tailwind oraz analogiczna struktura modułów
WN-05	Responsywność strukturalna	Tak	Elastyczne kontenery dostosowują układ do każdej szerokości ekranu
WN-06	Intuicyjność nawigacji	Tak	Pasek boczny/menu mobilne oraz spójny układ wewnętrzny każdego modułu zapewniają łatwą orientację i szybki dostęp do funkcji
WN-07	Izolacja danych	Tak	Silnik bazy danych wymusza polityki RLS, blokując dostęp do rekordów obcych użytkowników
WN-08	Uwierzytelnianie i sesja	Tak	Tokeny JWT (Supabase)
WN-09	Walidacja danych wejściowych	Tak	Weryfikacja w przeglądarce (UX) oraz w bazie danych
WN-10	Obsługa błędów	Tak	Komunikaty błędów typu *toast*
WN-11	Separacja logiczna kodu	Tak	Logika biznesowa (hooks) oddzielona od warstwy prezentacji (components)
WN-12	Komponentaryzacja	Tak	Reużywalne komponenty UI

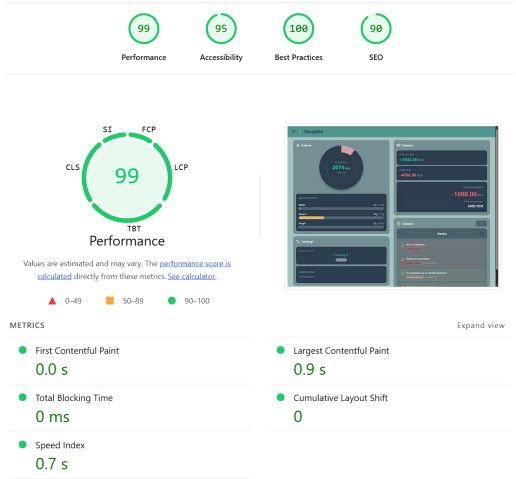
5.2.5. Weryfikacja wydajności i dostępności

Weryfikację jakości aplikacji przeprowadzono z wykorzystaniem narzędzia **Google Lighthouse** w środowisku produkcyjnym. Badanie obejmowało audyt wydajności (*Performance*), dostępności (*Accessibility*), najlepszych praktyk (*Best Practices*) oraz SEO. Testy wykonano dla dwóch scenariuszy użytkowania, zarówno na platformie desktopowej, jak i mobilnej:

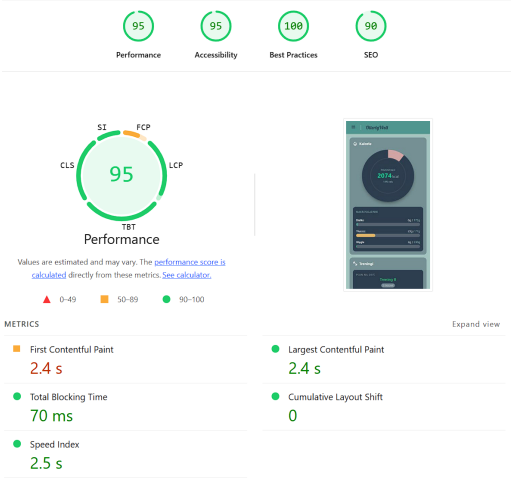
1. **Powracający użytkownik** (test bez czyszczenia pamięci podręcznej) - symuluje codzienne korzystanie z aplikacji, gdzie zasoby są już zapisane w pamięci przeglądarki ("warm start").
2. **Pierwsze uruchomienie** (test z wyczyszczeniem pamięci podręcznej) - symuluje wejście nowego użytkownika z koniecznością pobrania wszystkich zasobów z sieci ("cold start").



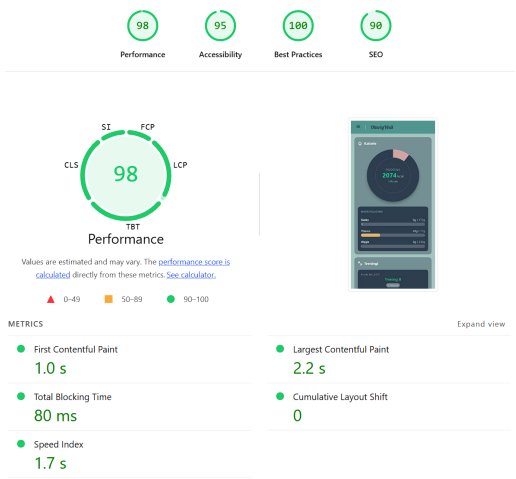
(a) Desktop - pierwsze uruchomienie



(b) Desktop - powracający użytkownik



(c) Urządzenie mobilne - pierwsze uruchomienie



(d) Urządzenie mobilne - powracający użytkownik

Rysunek 5.1: Zestawienie wyników audytu Google Lighthouse.

Wyniki audytów widoczne powyżej potwierdzają:

- **Wysoką wydajność** dla obu wersji, które uzyskały wyniki bliskie perfekcyjnym, niezależnie od stanu pamięci podręcznej. Widać jedynie niewielką różnicę między "zimnym" a "ciepłym" startem, a wskaźnik LCP (*Largest Contentful Paint*) utrzymuje się w bezpiecznym przedziale ($\leq$ 2.5 s) nawet na urządzeniach mobilnych.
- **Zgodność interfejsu** z obowiązującymi standardami, zgodnie z wysokimi wynikami w kategorii dostępności.
- Poprawną strukturę semantyczną kodu oraz zastosowanie nowoczesnych standardów webowych, w związku z wysokimi wynikami w pozostałych kategoriach.

Rozdział 6.

Zakończenie

6.1. Podsumowanie i ocena realizacji celów

Głównym celem pracy było zaprojektowanie i zaimplementowanie aplikacji webowej, pełniącej rolę centralnego huba do organizacji codziennego życia. Cel ten został osiągnięty poprzez stworzenie systemu integrującego cztery kluczowe obszary: żywienie, aktywność fizyczną, finanse oraz zarządzanie zadaniami.

Dostarczono wszystkie funkcjonalności zdefiniowane w opisanym zakresie MVP (podsekcja 1.3.2.), a analiza funkcjonalna oraz przeprowadzony proces weryfikacji wykazały, że aplikacja spełnia niemal wszystkie założenia projektowe.

Drobne odstępstwa od pierwotnego planu wystąpiły jedynie w przypadku wymagań WF-04 oraz WF-06, których pełna implementacja została celowo przesunięta na późniejszy etap na rzecz dopracowania logiki biznesowej. Wynikało to z początkowego niedoszacowania pracochłonności związanej z integracją kilku różnych domen w jednym systemie oraz konieczności poświęcenia większej ilości czasu na bardziej priorytetowe elementy.

Braki tym spowodowane nie wpływają jednak negatywnie na użyteczność końcowego produktu - wciąż działa on w pełni poprawnie, tyle że bez dwóch konkretnych drugorzędnych funkcjonalności.

Przeprowadzone testy, w tym audyty wydajności narzędziem Google Lighthouse, potwierdziły satysfakcjonującą jakość techniczną rozwiązania. Aplikacja charakteryzuje się krótkim czasem ładowania, responsywnym designem na urządzeniach mobilnych i desktopowych oraz poprawnym działaniem mechanizmów bezpieczeństwa i autoryzacji.

6.2. Wnioski z projektu

Realizacja projektu pozwoliła na wyciągnięcie kilku istotnych wniosków.

Wykorzystanie platformy Supabase okazało się kluczowym czynnikiem optymalizacji pracy. Przeniesienie odpowiedzialności za uwierzytelnianie i zarządzanie bazą danych na gotowe rozwiązanie BaaS pozwoliło skupić się maksymalnie na implementacji logiki biznesowej i dopracowaniu warstwy prezentacji.

Zastosowanie TypeScript mocno ułatwiło zarządzanie strukturą danych oraz proces refaktoryzacji kodu. Jako że projekt operował na różnorodnych rodzajach danych (żywieniowych, treningowych, finansowych, zadaniowych), typowanie pilnowało integralności systemu i weryfikowało poprawność kodu już na etapie kompilacji, co przyczyniło się do redukcji czasu potrzebnego na debugowanie.

Framework Tailwind CSS umożliwił budowę spójnego systemu wizualnego, przy jednoczesnym uniknięciu nadmiarowych arkuszy stylów, co znacząco skróciło czas pracy nad wyglądem aplikacji. Jednym z wyzwań był również sam projekt interfejsu, aby pasował do każdego rodzaju ekranu i adaptował swój układ w zależności od wyświetlanych treści, dając jednocześnie poczucie kontroli nad narzędziem.

Integracja z zewnętrznym API wymusiła zastosowanie funkcji bezserwerowych jako bezpiecznego pośrednika i stanowiła lekcję projektowania bezpiecznej komunikacji w aplikacjach webowych.

Dostosowanie interfejsu do urządzeń mobilnych ujawniło problemy niewidoczne w wersji desktopowej, takie jak wielkość obszarów touch-targetów i optymalny dla ekranów dotykowych design. Przygotowanie spójnych widoków wymagało dodatkowych prac nad warstwą wizualną oraz częstej weryfikacji, czy nowe zmiany stylowania dokonane na jednym z widoków nie wpływają negatywnie na pozostałe.

Jeśli chodzi o wdrożenie, kreatywnego podejścia wymagało wykorzystanie darmowego hostingu Github Pages. Obsługa routingu aplikacji SPA na platformie wymagała zastosowania techniki polegającej na duplikacji pliku startowego jako strony błędu, aby bezpośrednie odnośniki do podstron zaczęły działać prawidłowo.

Audyty wydajności pokazały też, że dość rozbudowana aplikacja typu "wszystko w jednym" nie musi ustępować szybkością narzędziom specjalistycznym, a ważne w tej kwestii było m.in. zastosowanie leniwego ładowania modułów.

Realizacja projektu w narzuconych ramach czasowych wymagała umiejętnej priorytetyzacji zadań, co wymusiło efektywniejsze rozkładanie pracy na różne obszary oraz potrzebę lepszego oszacowywania wymaganej przez nie pracochłonności. Poza pogłębieniem wiedzy technicznej, praca rozwinęła również moje umiejętności w zakresie zarządzania cyklem życia oprogramowania: od analizy wymagań, przez projektowanie architektury, aż po procesy wdrożeniowe i optymalizacyjne.

6.3. Propozycje dalszego rozwoju

Mimo spełnienia fundamentalnych założeń funkcjonalnych i stworzenia w pełni działającego produktu, projekt wciąż posiada duży potencjał rozwojowy. Otwarta architektura oraz zastosowane nowoczesne technologie stanowią solidny fundament pod dalszą rozbudowę, między innymi:

- **Dodanie częściowo brakujących funkcjonalności**:

  Doimplementować brakujący moduł kamery oraz logikę resetowania hasła.

- **Dalszy rozwój funkcjonalności modułów**:

  Rozwinąć bardziej istniejące już moduły, np. dodać wyświetlanie szczegółowych wykresów ze statystykami, sprawdzanie składu i jakości produktów w module żywieniowym, wizualizowanie niektórych ćwiczeń w module fitnessowym czy monitorowanie BMI współdzielące dane obu tych modułów.

- **Dodanie kolejnych modułów**:

  Dodać jeszcze więcej nowych funkcjonalności w postaci kolejnych modułów, np. moduł śledzenia jakości snu, timer pomodoro czy organizator notatek.

- **Integracja z zewnętrznymi ekosystemami**:

  Rozbudować moduły o synchronizację z powiązanymi tematycznie ekosystemami, np. Google Fit/Apple Health w module fitness (w celu pobierania danych o krokach i spalonych kaloriach) lub Kalendarz Google w module zadań (w celu synchronizacji zadań z kalendarzem).

- **Mechanizmy wspierające systematyczność i postęp**:

  Wprowadzenie mechanizmów takich jak system osiągnięć realizacji kamieni milowych czy śledzenie serii dni regularnego korzystania z aplikacji.

- **Funkcje społecznościowe**:

  Dodanie możliwości interakcji między użytkownikami, np. dzielenia się planami treningowymi czy wykonywanie wspólnych zadań.

- **Wsparcie sztucznej inteligencji**:

  Wykorzystać algorytmy uczenia maszynowego do analizy danych i zaoferować spersonalizowane sugestie, np. proponowanie zbalansowanego treningu czy generowanie nowej listy zakupów na podstawie wcześniejszej.

Bibliografia

[1] MyFitnessPal: https://www.myfitnesspal.com/pl [ostatni dostęp: 28.01.2026].

[2] Fitatu: https://www.fitatu.com/ [ostatni dostęp: 28.01.2026].

[3] Strava: https://www.strava.com/ [ostatni dostęp: 28.01.2026].

[4] Notion: https://www.notion.com/ [ostatni dostęp: 28.01.2026].

[5] Obsidian: https://obsidian.md/ [ostatni dostęp: 28.01.2026].

[6] Biblioteka React: https://react.dev/ [ostatni dostęp: 28.01.2026].

[7] Dokumentacja Vite: https://vite.dev/ [ostatni dostęp: 28.01.2026].

[8] TypeScript: https://www.typescriptlang.org/ [ostatni dostęp: 28.01.2026].

[9] Framework Tailwind CSS: https://tailwindcss.com/ [ostatni dostęp: 28.01.2026].

[10] Biblioteka Headless UI: https://headlessui.com/ [ostatni dostęp: 28.01.2026].

[11] Biblioteka Framer Motion: https://motion.dev/docs [ostatni dostęp: 28.01.2026].

[12] Dokumentacja platformy Supabase: https://supabase.com/docs [ostatni dostęp: 28.01.2026].

[13] Dokumentacja baz danych PostgreSQL: https://www.postgresql.org/docs/ [ostatni dostęp: 28.01.2026].

[14] Open Food Facts API: https://openfoodfacts.github.io/documentation/docs/Product-Opener/api/ [ostatni dostęp: 28.01.2026].

[15] Serwis GitHub: https://github.com/ [ostatni dostęp: 28.01.2026].

[16] GitHub Pages: https://docs.github.com/en/pages [ostatni dostęp: 28.01.2026].

[17] Material Symbols & Icons - Google Fonts: `https://fonts.google.com/icons` [ostatni dostęp: 28.01.2026].

[18] React Router: `https://reactrouter.com/` [ostatni dostęp: 28.01.2026].

[19] Środowisko Node.js: `https://nodejs.org/en` [ostatni dostęp: 28.01.2026].

[20] Framework Vitest: `https://vitest.dev/` [ostatni dostęp: 28.01.2026].

[21] React Testing Library: `https://testing-library.com/docs/` [ostatni dostęp: 28.01.2026].

[22] Google Lighthouse: `https://developer.chrome.com/docs/lighthouse?hl=pl` [ostatni dostęp: 28.01.2026].