UNIWERSYTET RZESZOWSKI

WYDZIAŁ NAUK ŚCISŁYCH I TECHNICZNYCH

INSTYTUT INFORMATYKI

Oliwier Hędrzak, Michał Janik

134913, 134915

Informatyka

Dziennik treningowy AwareFit – dokumentacja projektowa

Praca projektowa

Praca wykonana pod kierunkiem
dr inż. Piotr Grochowalski

Rzeszów 2026

Spis treści

1. Wprowadzenie

Projekt **AwareFit** powstał z przekonania, że większość dostępnych na rynku aplikacji treningowych to jedynie cyfrowe wersje papierowego notesu. Choć pozwalają one zapisać wykonane serie, rzadko dają użytkownikowi realną wartość dodaną w postaci analizy. W dzisiejszych czasach osoby trenujące siłowo wiedzą, że progres nie bierze się z przypadku, ale z precyzyjnego zarządzania objętością (tonażem), intensywnością oraz regeneracją.

Głównym celem AwareFit było stworzenie inteligentnego systemu, który nie tylko archiwizuje jednostki treningowe, ale przede wszystkim je „rozumie". Na rynku wciąż brakuje narzędzi, które potrafiłyby wyciągać wnioski z historii treningów i zamieniać surowe liczby w konkretne wskazówki. System AwareFit automatycznie przelicza progresję, analizuje tonaż i identyfikuje aktualny etap planu użytkownika. To aplikacja stworzona dla tych, którzy chcą trenować w pełni świadomie, co zresztą podkreśla sama jej nazwa. System automatyzuje kluczowe procesy np. oblicza całkowitą objętość sesji, porównuje ją z wynikami z poprzednich tygodni, a dzięki autorskim algorytmom zaimplementowanym po stronie bazy danych, samodzielnie identyfikuje rodzaj realizowanego systemu treningowego.

W warstwie technologicznej projekt skupia się na maksymalnym wykorzystaniu możliwości relacyjnej bazy danych PostgreSQL. Logika aplikacji nie spoczywa wyłącznie na barkach interfejsu, lecz jest głęboko zakorzeniona w procedurach składowanych i funkcjach bazodanowych (pgSQL). Takie podejście gwarantuje spójność danych, szybkość obliczeń oraz skalowalność systemu. AwareFit to kompletne środowisko, które łączy świat bazy danych, logiki serwerowej PHP oraz nowoczesnego interfejsu użytkownika, tworząc spójne narzędzie do świadomego kształtowania własnej formy fizycznej.

2. Specyfikacja

Projektowanie systemu **AwareFit** rozpoczęto od analizy tego, jak naprawdę wygląda trening siłowy i co tak naprawdę pomoże użytkownikowi w jego przygodzie z siłownią. Kluczowe było to, aby nie tworzyć kolejnego notesu do zapisywania treningu, lecz takiego dziennika, który faktycznie będzie wspierał użytkownika i prowadził go przez swoją drogę do upragnionej formy. Poniżej znajduje się opis, w jaki sposób rzeczywistość została przeniesiona z siłowni do logiki systemu bazodanowego. Ważnym aspektem była elastyczność danych, co pozwala na swobodny rozwój aplikacji. Na bazie własnych obserwacji oraz doświadczeniu utworzony został koncept, który zawiera odpowiednią strukturę oraz algorytmy w języku pgSQL stanowiące serce całego systemu.

2.1. Scenariusz interakcji z systemem

Aby w pełni zrozumieć strukturę projektowanej bazy danych, należy prześledzić typową ścieżkę użytkownika, która determinuje sposób przepływu informacji w systemie:

1. **Rejestracja/Logowanie:** Pierwszym krokiem interakcji z systemem jest utworzenie unikalnego konta użytkownika. Logowanie to mechanizm, który pozwala systemowi wyizolować dane konkretnej osoby. Dzięki temu, po podaniu danych uwierzytelniających, baza filtruje setki rekordów, aby wyświetlić tylko te trendy i statystyki, które należą do danego użytkownika, zapewniając mu prywatność i spersonalizowany widok dashboardu.

2. **Przygotowanie i cel:** Użytkownik rozpoczyna korzystanie z aplikacji od wprowadzenia swojego celu oraz kluczowych pomiarów ciała, co pozwala na zdefiniowanie momentu, w którym się znajduje. Dzięki zaawansowanym algorytmom po stronie bazy danych możliwe jest obliczenie zapotrzebowania kalorycznego, makroskładników, czy chociażby procenta tkanki tłuszczowej w ciele. Zarówno cel jak i pomiary mogą być aktualizowane przez użytkownika w przeznaczonym do tego panelu.

3. **Rozpoczęcie sesji i inteligentne wsparcie:** Po przyjściu na siłownię, użytkownik inicjuje nową jednostkę treningową. W tym momencie system tworzy w bazie danych unikalny rekord treningu przypisany do zalogowanej osoby i rozpoczyna pomiar czasu trwania sesji. Użytkownik wybiera partię mięśniową, a następnie konkretne ćwiczenie z nią powiązane.

   Kluczowym elementem systemu jest wsparcie w czasie rzeczywistym. Podczas wpisywania danych, użytkownik widzi podpowiedzi w polach formularza, które przypominają mu o użytym ciężarze i liczbie powtórzeń z ostatniego treningu tego samego ćwiczenia. Dzięki temu nie musi on co chwilę spoglądać w historię zapisanych treningów. Warto dodać, że system analizuje treningi wstecz i wyświetla rekomendację, czy użytkownik powinien zwiększyć obciążenie, czy pozostać przy obecnym.

   Po każdej serii użytkownik zapisuje swoje osiągi, budując strukturę treningu krok po kroku, aż do jego zakończenia. Cała ta logika jest wynikiem zaawansowanych algorytmów działających bezpośrednio po stronie silnika bazy danych.

4. **Analiza potreningowa:** Po zakończeniu i zapisaniu sesji, system natychmiastowo przetwarza zebrane informacje, aby zaktualizować centralny panel sterowania – Dashboard. W tym momencie dane z tabel dotyczących treningów są agregowane przez funkcje bazodanowe, które przeliczają całkowitą objętość (tonaż) z ostatnich 7 dni i porównują ją z analogicznym okresem w przeszłości.

   Dzięki temu, przy kolejnym uruchomieniu aplikacji, użytkownik od razu widzi dynamiczne wskaźniki postępu, takie jak procentowy wzrost objętości czy aktualną passę treningową (streak). System generuje również interaktywne wykresy progresji siłowej dla wybranych ćwiczeń, pozwalając na wizualną ocenę progresu. Dodatkowo do systemu zaimplementowany został algorytm, który samodzielnie rozpoznaje rodzaj treningu, na bazie wykonanych jednostek w skali tygodnia.

   W tym samym miejscu dane treningowe spotykają się z danymi dietetycznymi. Dashboard wyświetla podsumowanie spożytych kalorii i makroskładników w formie czytelnych kart, co pozwala użytkownikowi błyskawicznie ocenić, czy jego regeneracja i odżywianie są spójne z intensywnością wykonanej pracy na siłowni. Cały ten proces sprawia, że AwareFit przestaje być tylko dziennikiem, a staje się osobistym analitykiem sportowym dostępnym na żądanie.

5. **Panel progresu:** Kluczowym elementem AwareFit jest zaawansowany panel progresu, który służy do monitorowania rozwoju. Najważniejszym wskaźnikiem jest tutaj objętość, która pokazuje użytkownikowi jego tendencję progresu. Dzięki temu możliwe jest, aby użytkownik doszedł do wniosku, czy przyjęta przez niego metoda treningowa przynosi zamierzone efekty.

   Baza danych analizuje wszystkie treningi z ostatnich 7 dni i wylicza procentowy udział każdej partii mięśniowej w całym planie. Jeśli użytkownik zaniedba jakąś grupę (np. pominie trening nóg), system natychmiast to wyłapie i wyświetli komunikat o pominiętej partii. Ma to na celu zapobieganie dysproporcjom sylwetkowym i kontuzjom. Dodatkowo, na podstawie historii serii, aplikacja estymuje aktualny rekord maksymalny (1RM) w najważniejszych bojach, takich jak wyciskanie, przysiad czy martwy ciąg, co pozwala na bieżąco śledzić wzrost czystej siły bez konieczności ryzykownego sprawdzania jej w każdej sesji.

Podsumowując, powyższy scenariusz pokazuje, że system **AwareFit** został zaprojektowany z myślą o rzeczywistym problemie. Struktura została zaprojektowana w taki sposób, aby możliwe było prowadzenie użytkownika przez jego przygodę z siłownią. Od momentu pomiarów ciała, przez wyliczenie diety oraz zapisywanie treningu, po analizę danych i końcowe wskazówki i kluczowe wartości. Dzięki temu użytkownik otrzymuje intuicyjny w obsłudze dziennik treningowy, który dba o jego progres, zdrowie i buduje jego świadomość ciała.

2.2. Użyte technologie

Realizacja systemu **AwareFit** opiera się na sprawdzonych rozwiązaniach, które zapewniają stabilność przetwarzania danych oraz responsywność interfejsu użytkownika:

- **PostgreSQL (Silnik bazy danych):** Centralny element systemu. Wybrany ze względu na zaawansowane wsparcie dla języka **pgSQL**, który pozwolił na przeniesienie logiki analitycznej i algorytmów progresji bezpośrednio na stronę serwera bazy danych.
- **PHP (Backend):** Odpowiada za bezpieczną komunikację między interfejsem użytkownika a bazą danych, obsługę sesji oraz logikę przesyłania formularzy.
- **HTML5 / CSS3 / JavaScript (Frontend):** Wykorzystane do budowy intuicyjnego interfejsu oraz dynamicznych dashboardów. Za warstwę wizualną wykresów progresji odpowiada biblioteka **Chart.js**, która w czasie rzeczywistym renderuje dane dostarczane z bazy. Projektowanie GUI opierało się o metodę mobile-first.
- **XAMPP:** Zintegrowany pakiet oprogramowania, który posłużył jako lokalne środowisko deweloperskie. Zapewnił on niezbędne komponenty, takie jak serwer Apache do obsługi skryptów PHP.

3. Opis struktury bazy danych

3.1. Zdefiniowane struktury tabel

Tabela: users

Tabela ta stanowi centralny punkt bazy danych, przechowując dane identyfikacyjne użytkowników oraz informacje niezbędne do procesów autoryzacji i personalizacji.

Nazwa kolumny	Typ danych	Opis i ograniczenia
id (PK)	serial	Unikalny identyfikator użytkownika.
username	varchar(50)	Nazwa użytkownika wykorzystywana do logowania.
password	varchar(255)	Zahaszowane hasło użytkownika.
email	varchar(100)	Adres e-mail przypisany do konta.
first_name	varchar(50)	Imię użytkownika.
last_name	varchar(50)	Nazwisko użytkownika.
gender	text	Płeć użytkownika.

Tabela 3.1. Struktura tabeli users

Tabela: body_measurements

Przechowuje historię pomiarów antropometrycznych użytkownika oraz zdefiniowane przez niego cele i poziomy aktywności.

Nazwa kolumny	Typ danych	Opis i ograniczenia
id (PK)	serial	Unikalny identyfikator pomiaru.
user_id (FK)	integer	Powiązanie z tabelą users.
date	timestamp	Data i godzina wykonania pomiaru.
height	double precision	Wzrost użytkownika.
weight	double precision	Masa ciała.
chest	double precision	Obwód klatki piersiowej.
waist	double precision	Obwód pasa.
biceps	double precision	Obwód ramienia.
thighs	double precision	Obwód uda.
hips	double precision	Obwód bioder.
neck	numeric	Obwód szyi.
goal	varchar(50)	Cel treningowy (np. redukcja, masa).
activity_level	numeric(4,3)	Współczynnik aktywności fizycznej.

Tabela 3.2. Struktura tabeli body_measurements

Tabela: muscle_groups

Słownikowa tabela zawierająca nazwy partii mięśniowych, co pozwala na kategoryzację ćwiczeń.

Nazwa kolumny	Typ danych	Opis i ograniczenia
id (PK)	serial	Identyfikator grupy mięśniowej.
name	varchar(50)	Nazwa partii (np. klatka piersiowa, plecy).

Tabela 3.3. Struktura tabeli muscle_groups

Tabela: exercises

Katalog dostępnych w systemie ćwiczeń wraz z ich opisami i przypisaniem do konkretnych partii.

Nazwa kolumny	Typ danych	Opis i ograniczenia
id (PK)	serial	Identyfikator ćwiczenia.
name	varchar(100)	Nazwa ćwiczenia.
description	text	Instrukcja lub opis techniki ćwiczenia.
muscle_group_id (FK)	integer	Powiązanie z tabelą muscle_groups.

Tabela 3.4. Struktura tabeli exercises

Tabela: workouts

Rejestruje każdą rozpoczętą sesję treningową użytkownika wraz z czasem jej trwania.

Nazwa kolumny	Typ danych	Opis i ograniczenia
id (PK)	serial	Identyfikator sesji treningowej.
user_id (FK)	integer	Identyfikator użytkownika wykonującego trening.
date	timestamp	Data i godzina rozpoczęcia treningu.
duration	interval	Czas trwania sesji treningowej.

Tabela 3.5. Struktura tabeli workouts

Tabela: workout_exercises

Tabela pośrednicząca, która przypisuje konkretne ćwiczenia do danej jednostki treningowej.

Nazwa kolumny	Typ danych	Opis i ograniczenia
id (PK)	serial	Identyfikator wpisu ćwiczenia w treningu.
workout_id (FK)	integer	Powiązanie z konkretną sesją (workouts).
exercise_id (FK)	integer	Powiązanie z katalogiem ćwiczeń (exercises).

Tabela 3.6. Struktura tabeli workout_exercises

Tabela: workout_sets

Najbardziej szczegółowa tabela systemu, przechowująca parametry każdej wykonanej serii.

Nazwa kolumny	Typ danych	Opis i ograniczenia
id (PK)	serial	Identyfikator serii.
workout_exercise_id (FK)	integer	Powiązanie z ćwiczeniem w ramach treningu.
weight	double precision	Obciążenie użyte w serii.
reps	integer	Liczba wykonanych powtórzeń.
set_number	integer	Numer porządkowy serii w danym ćwiczeniu.

Tabela 3.7. Struktura tabeli workout_sets

3.2. Powiązania pomiędzy tabelami i ich interpretacja

Analiza diagramu ERD oraz struktur tabel pozwala na zdefiniowanie relacji, które gwarantują spójność danych oraz umożliwiają zaawansowaną analitykę treningową. Poniżej przedstawiono szczegółową interpretację kluczowych powiązań:

- **Relacja users – body_measurements (1:N):** To powiązanie pozwala na przypisanie wielu pomiarów ciała do jednego profilu użytkownika. Z punktu widzenia systemu jest to kluczowe dla monitorowania zmian w czasie (np. spadku wagi czy wzrostu obwodów). Dzięki tej relacji dashboard może generować wykresy trendów, porównując najnowszy wpis z danymi historycznymi.
- **Relacja users – workouts (1:N):** Fundament rejestracji aktywności. Każdy trening jest trwale przypisany do konkretnego użytkownika poprzez klucz obcy *user_id*. Umożliwia to izolację danych i obliczanie statystyk takich jak aktualna passa treningowa (streak) czy całkowita objętość podniesiona przez daną osobę w skali tygodnia.
- **Struktura hierarchiczna treningu (workouts – workout_exercises – workout_sets):** Zastosowano tutaj potrójne powiązanie typu jeden-do-wielu, co odzwierciedla naturalną strukturę sesji na siłowni. Jeden rekord w *workouts* (sesja) zawiera wiele wpisów w *workout_exercises* (wykonane ćwiczenia), a każde z tych ćwiczeń posiada wiele rekordów w *workout_sets* (serie). Taka dekompozycja danych pozwala na precyzyjne wyliczanie tonażu (objętości) dla każdego ćwiczenia z osobna.
- **Relacja exercises – workout_exercises (1:N):** Łączy konkretne wykonanie ćwiczenia z jego definicją w katalogu. Dzięki temu system "wie", jakie ćwiczenie wykonuje użytkownik i może pobrać z tabeli *exercises* jego opis lub przypisaną grupę mięśniową.
- **Relacja muscle_groups – exercises (1:N):** Każde ćwiczenie jest sklasyfikowane pod jedną grupą mięśniową. To powiązanie jest niezbędne do działania algorytmu balansu strukturalnego. System sumuje serie z tabeli *workout_sets*, przechodzi przez powiązania do *muscle_groups* i na tej podstawie wylicza procentowy udział treningu klatki piersiowej, pleców czy nóg w skali tygodnia.

3.2.1. Opcjonalność relacji

Dla zachowania spójności danych, w systemie zdefiniowano następujące zasady opcjonalności:

- **Relacja users – workouts (Obowiązkowa):** Każdy rekord w tabeli *workouts* musi posiadać przypisane *user_id*. Nie jest możliwe istnienie treningu w systemie bez przypisanego autora (użytkownika).
- **Relacja workouts – workout_exercises (Obowiązkowa):** Każdy wpis o wykonanym ćwiczeniu musi odwoływać się do konkretnej sesji treningowej. Usunięcie treningu powoduje kaskadowe usunięcie przypisanych do niego ćwiczeń.
- **Relacja exercises – muscle_groups (Obowiązkowa):** W systemie każde ćwiczenie musi być skategoryzowane pod konkretną partią mięśniową, aby algorytmy analityczne mogły poprawnie przeliczać objętość tygodniową.
- **Relacja users – body_measurements (Opcjonalna):** Użytkownik może, ale nie musi posiadać wpisów w tabeli pomiarów. System dopuszcza istnienie konta bez historii pomiarów, choć ogranicza to wtedy dostęp do niektórych funkcji dashboardu (np. wykresów zmiany wagi).
- **Relacja workout_exercises – workout_sets (Obowiązkowa):** Seria nie może istnieć bez powiązania z konkretnym ćwiczeniem w ramach danego treningu.

3.3. Diagram związków encji (ERD)

Poniższy diagram przedstawia graficzną reprezentację struktury bazy danych systemu. Ilustruje on wszystkie opisane wcześniej tabele, ich atrybuty oraz relacje zachodzące pomiędzy poszczególnymi encjami, z uwzględnieniem kluczy głównych (PK) oraz obcych (FK).

Rys. 3.1. Diagram związków encji (ERD) projektowanej bazy danych.

3.4. Kod implementujący strukturę bazy danych

Poniższy przedstawiono kod, który posłużył do fizycznego utworzenia struktury tabel bazy danych zgodnie z zaprojektowanym diagramem ERD.

Listing 3.1. Kod implementujący strukturę tabel bazy danych

```
-- 1. Tabela uzytkownikow
CREATE TABLE public.users (
    id SERIAL PRIMARY KEY,
    username VARCHAR(50) NOT NULL UNIQUE,
    password VARCHAR(255) NOT NULL,
    email VARCHAR(100) NOT NULL UNIQUE,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    gender TEXT
);

-- 2. Tabela pomiarow ciala
CREATE TABLE public.body_measurements (
    id SERIAL PRIMARY KEY,
    user_id INTEGER NOT NULL,
    date TIMESTAMP WITHOUT TIME ZONE NOT NULL,
    height DOUBLE PRECISION,
    weight DOUBLE PRECISION,
    chest DOUBLE PRECISION,
    waist DOUBLE PRECISION,
    biceps DOUBLE PRECISION,
    thighs DOUBLE PRECISION,
    hips DOUBLE PRECISION,
    neck NUMERIC,
    goal VARCHAR(50),
    activity_level NUMERIC(4,3),
    FOREIGN KEY (user_id) REFERENCES public.users(id)
);

-- 3. Tabela grup miesniowych
CREATE TABLE public.muscle_groups (
    id SERIAL PRIMARY KEY,
    name VARCHAR(50) NOT NULL
);

-- 4. Tabela cwiczen
CREATE TABLE public.exercises (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    description TEXT,
    muscle_group_id INTEGER NOT NULL,
    FOREIGN KEY (muscle_group_id) REFERENCES public.muscle_groups(id)
);
```

```
-- 5. Tabela treningow (naglowki)
CREATE TABLE public.workouts (
    id SERIAL PRIMARY KEY,
    user_id INTEGER NOT NULL,
    date TIMESTAMP WITHOUT TIME ZONE NOT NULL,
    duration INTERVAL,
    FOREIGN KEY (user_id) REFERENCES public.users(id)
);

-- 6. Tabela cwiczen w ramach treningu
CREATE TABLE public.workout_exercises (
    id SERIAL PRIMARY KEY,
    workout_id INTEGER NOT NULL,
    exercise_id INTEGER NOT NULL,
    FOREIGN KEY (workout_id) REFERENCES public.workouts(id),
    FOREIGN KEY (exercise_id) REFERENCES public.exercises(id)
);

-- 7. Tabela serii treningowych
CREATE TABLE public.workout_sets (
    id SERIAL PRIMARY KEY,
    workout_exercise_id INTEGER NOT NULL,
    weight DOUBLE PRECISION,
    reps INTEGER,
    set_number INTEGER,
    FOREIGN KEY (workout_exercise_id) REFERENCES public.workout_exercises(id)
);
```

4. Logika operacyjna i funkcjonalność bazy danych

Struktura bazy danych została zaprojektowana tak, aby wspierała kluczowe funkcje aplikacji AwareFit. Funkcjonalność opiera się na dwóch aspektach: standardowej obsłudze danych poprzez mechanizmy CRUD oraz zaawansowanej logice algorytmicznej, która pozwala na przekształcenie surowych wpisów treningowych w użyteczne informacje analityczne dla użytkownika.

4.1. Założenia i wymagania dla mechanizmów CRUD

Mechanizm CRUD (ang. Create, Read, Update, Delete) stanowi fundament interakcji użytkownika z systemem. W ramach aplikacji AwareFit zdefiniowano, w schemacie *crud* zaimplementowany zestaw funkcji i procedur. Poniżej przedstawiono podział na komponenty CRUD.

Zarządzanie tożsamością i dostępem (Komponent: users)

Komponent ten dotyczny zarządzeniem kontami użytkowników, zaimplementowany w procedurach schematu *crud*.

- **Operacja Create (Rejestracja):** Wykorzystywana jest procedura `insert_user`, która przyjmuje komplet danych: login, hasło, email oraz dane personalne. Zapewnia ona poprawność wpisu w tabeli głównej *public.users*.
- **Operacja Read (Logowanie i Profil):** Funkcja `get_all_users` pozwala na weryfikację uprawnień i pobieranie danych użytkowników.
- **Operacja Update i Delete:** Zaimplementowano procedury `update_user` oraz `delete_user`. Warto zaznaczyć, że choć są one gotowe w warstwie bazy danych, aktualna wersja interfejsu graficznego (GUI) nie wykorzystuje ich bezpośrednio, co stanowi przygotowanie pod przyszłą rozbudowę modułu administracyjnego.

Rejestracja i obsługa sesji treningowych (Komponent: workouts)

Jest to najbardziej rozbudowany fragment schematu *crud*, obsługujący wielopoziomową strukturę jednostki treningowej.

- **Struktura hierarchiczna:** Proces zapisu treningu jest atomowy i sekwencyjny. Najpierw wywoływana jest procedura `insert_workout`, a następnie dla każdego wybranego ćwiczenia `insert_workout_exercise`. Detale wykonania (ciężar, powtórzenia) są zapisywane przez `insert_workout_set`.
- **Zarządzanie zmianami i perspektywy rozwoju:** Procedury `update_workout_set` oraz `update_workout_exercise` zostały zaprojektowane i wdrożone w warstwie bazy danych jako fundament pod przyszłą rozbudowę funkcjonalności systemu. W obecnej iteracji aplikacji, interfejs użytkownika (GUI) nie udostępnia możliwości edycji treningów historycznych, co jest podyktowane dbałością o spójność danych i rzetelność generowanych statystyk. Niemniej jednak, pełna implementacja tych procedur w schemacie *crud* umożliwia ich natychmiastowe wykorzystanie w przyszłości bez konieczności modyfikacji logiki serwera bazy danych.

Monitoring parametrów biometrycznych (Komponent: body_measurements)

Komponent ten dedykowany jest gromadzeniu i analizie danych o stanie fizycznym użytkownika. Logika bazodanowa została zaprojektowana tak, aby wspierać rzetelne śledzenie progresji w czasie.

- **Rejestracja pomiarów:** Główną operacją w interfejsie użytkownika jest *Create*, realizowana przez zestaw procedur `insert_body_measurement`. Użytkownik ma możliwość regularnego dodawania nowych rekordów zawierających wagę oraz obwody poszczególnych partii ciała.
- **Niezmienność danych historycznych:** Przyjęto założenie projektowe, według którego raz wprowadzone dane biometryczne nie podlegają edycji z poziomu interfejsu graficznego (GUI). Ma to na celu zachowanie autentyczności historii pomiarów i uniemożliwienie manipulacji danymi archiwalnymi.
- **Zaimplementowany mechanizm Update:** Pomimo ograniczeń w interfejsie, w schemacie *crud* w pełni zaimplementowano procedurę `update_body_measurement`. Pozwala to na ewentualne przyszłe wdrożenie funkcji korekty błędnych wpisów przez użytkownika.
- **Wyświetlanie i prezentacja danych:** Za warstwę *Read* odpowiada funkcja `get_all_body_measurements` oraz dedykowane zapytania filtrujące, które zasilają interfejs graficzny. Mechanizm ten pozwala na dynamiczne prezentowanie parametrów w formie zestawień tabelarycznych oraz stanowi źródło danych dla poszczególnych funkcji prezentujących informacje na temat ciała.

Zarządzanie strukturą słownikową (Komponent: exercises & muscle_groups)

Tabele słownikowe stanowią bazę wiedzy systemu, definiując dostępne ćwiczenia oraz ich przynależność do grup mięśniowych.

- **Dostęp użytkownika:** Z poziomu interfejsu standardowego użytkownika, dostęp do danych słownikowych jest ograniczony wyłącznie do operacji *Read*. Wykorzystywane są do tego funkcje `get_all_exercises` oraz `get_all_muscle_groups`, które zasilają listy wyboru podczas kreowania nowej sesji treningowej.
- **Perspektywa administracyjna:** Mimo braku odpowiednich modułów w bieżącym GUI, w schemacie *crud* zaimplementowano pełen zestaw procedur zarządczych: `insert_exercise`, `update_exercise`, `delete_exercise` (oraz ich odpowiedniki dla grup mięśniowych).
- **Założenia projektowe:** Istnienie tych mechanizmów w warstwie bazy danych zostało przewidziane jako fundament pod dedykowany "Panel Administratora". Pozwoli on w przyszłości na dynamiczne rozbudowywanie bazy ćwiczeń, czy edycję opisów bez konieczności bezpośredniej ingerencji programisty w strukturę bazy danych.

4.2. Implementacja mechanizmów CRUD na przykładzie komponentu użytkowników

W celu zapewnienia integralności danych oraz odciążenia warstwy logicznej aplikacji (PHP), kluczowe reguły zostały zaimplementowane bezpośrednio w procedurach składowanych schematu *crud*.

Tworzenie rekordu (Create)

Za dodawanie nowych użytkowników odpowiada procedura `crud.insert_user`.

- **Założenia:** Procedura przyjmuje komplet danych profilowych i dokonuje ich ostatecznej weryfikacji przed trwałym zapisem w bazie.
- **Implementacja i walidacja:** Poza operacją `INSERT`, procedura aktywnie sprawdza poprawność formatu adresu e-mail przy użyciu wyrażeń regularnych (*regex*).
- **Obsługa błędów unikalności:** Zastosowano blok `EXCEPTION`, który przechwytuje naruszenia kluczy unikalnych (`unique_violation`). Pozwala to na precyzyjne rozróżnienie, czy błąd rejestracji wynika z duplikatu nazwy użytkownika, czy zajętego adresu e-mail, co jest kluczowe dla komunikatów zwrotnych w interfejsie użytkownika.

Odczyt danych (Read)

Funkcja `crud.get_all_users` stanowi standardowy interfejs dostępu do danych dla warstwy aplikacji.

- **Założenia:** Zapewnienie szybkiego dostępu do listy użytkowników przy zachowaniu struktury rekordowej.
- **Implementacja:** Funkcja wykorzystuje mechanizm `RETURNS SETOF users`, co pozwala traktować jej wynik jak wirtualną tabelę. Zastosowanie instrukcji `RETURN QUERY` optymalizuje proces pobierania danych przez sterownik PDO w PHP.

Aktualizacja danych (Update)

Procedura `crud.update_user` wprowadza warstwę weryfikacji przed modyfikacją istniejących zasobów.

- **Założenia:** Edycja danych jest dopuszczalna wyłącznie dla istniejącego w systemie identyfikatora użytkownika.
- **Mechanizm walidacji:** Przed wykonaniem polecenia `UPDATE`, procedura sprawdza istnienie rekordu:

```
IF NOT EXISTS (SELECT 1 FROM public.users WHERE id = p_id) THEN
   RAISE EXCEPTION 'Nie znaleziono użytkownika o ID %', p_id;
END IF;
```

Gwarantuje to spójność operacji i ułatwia debugowanie komunikacji na linii API-Baza danych.

Usuwanie danych (Delete)

Procedura `crud.delete_user` odpowiada za bezpieczne usuwanie kont przy zachowaniu integralności referencyjnej.

- **Założenia:** Blokada usunięcia użytkowników posiadających aktywne powiązania w innych modułach systemu (np. historia treningowa).
- **Obsługa wyjątków:** Wykorzystano przechwytywanie błędu `foreign_key_violation`:

```
EXCEPTION WHEN foreign_key_violation THEN
   RAISE EXCEPTION 'Nie można usunąć użytkownika - posiada powiązane dane...';
```

  Dzięki temu system chroni historię treningową przed powstaniem rekordów osieroconych (*orphaned records*).

4.3. Opis pytań algorytmicznych

W niniejszym rozdziale szczegółowo opisano logikę zaimplementowaną bezpośrednio w warstwie bazy danych. Wykorzystanie procedur składowanych (*PL/pgSQL*) pozwoliło na odciążenie warstwy aplikacji oraz zapewnienie wysokiej wydajności przy przetwarzaniu złożonych wskaźników treningowych i biometrycznych.

4.3.1. Moduł analityki treningowej i progresji siłowej

Głównym zadaniem tego modułu jest transformacja surowych danych o seriach treningowych w czytelne wskaźniki postępu.

1. Algorytm obliczania całkowitej objętości treningu

Objętość treningowa (ang. *training volume*) jest jednym z najważniejszych wskaźników progresu, pozwalającym na ocenę całkowitej pracy wykonanej podczas jednej jednostki treningowej. Za realizację tego zadania odpowiada funkcja `calculate_workout_total_volume`.

Implementacja SQL:

Listing 4.1. Procedura obliczania tonażu sesji treningowej

```
CREATE OR REPLACE FUNCTION public.calculate_workout_total_volume(p_workout_id integer)
RETURNS double precision LANGUAGE 'plpgsql' AS $BODY$
DECLARE
    v_total_volume float;
BEGIN
    SELECT SUM(ws.weight * ws.reps) INTO v_total_volume
    FROM public.workout_sets ws
    JOIN public.workout_exercises we ON ws.workout_exercise_id = we.id
    WHERE we.workout_id = p_workout_id;

    RETURN COALESCE(v_total_volume, 0);
END; $BODY$;
```

Kroki algorytmu:

1. **Przyjęcie parametrów:** Pobranie `p_workout_id` i inicjalizacja zmiennej `v_total_volume`.
2. **Złączenie relacyjne:** Operacja *JOIN* tabel *workout_sets* oraz *workout_exercises* w celu identyfikacji rekordów sesji.
3. **Przetwarzanie danych:** Matematyczne mnożenie obciążenia (*weight*) przez liczbę powtórzeń (*reps*).
4. **Agregacja:** Wykorzystanie funkcji `SUM` do uzyskania całkowitego tonażu.
5. **Walidacja końcowa:** Zastosowanie `COALESCE` w celu zamiany ewentualnej wartości `NULL` na 0.
6. **Ekspedycja wyniku:** Zwrócenie wartości typu *double precision* do aplikacji.

2. Algorytm analizy progresji objętości ćwiczenia w czasie

Funkcja `get_exercise_volume_progression` generuje zestawienie tonażu dla konkretnego ćwiczenia z okresu ostatnich trzech miesięcy, co pozwala na wizualizację trendu siłowego.

Implementacja SQL:

Listing 4.2. Generowanie szeregu czasowego objętości

```
CREATE OR REPLACE FUNCTION public.get_exercise_volume_progression(p_user_id integer, p_exercise_id integer)
RETURNS TABLE(workout_date date, total_volume numeric) LANGUAGE 'plpgsql' AS $BODY$
BEGIN
    RETURN QUERY
    SELECT w.date::DATE, SUM(COALESCE(ws.weight, 0) * COALESCE(ws.reps, 0))::NUMERIC
    FROM public.workouts w
    JOIN public.workout_exercises we ON w.id = we.workout_id
    JOIN public.workout_sets ws ON we.id = ws.workout_exercise_id
    WHERE w.user_id = p_user_id AND we.exercise_id = p_exercise_id
      AND w.date >= CURRENT_DATE - INTERVAL '3 months'
    GROUP BY w.date::DATE ORDER BY w.date::DATE ASC;
END; $BODY$;
```

Kroki algorytmu:

1. **Przyjęcie kontekstu:** Pobranie identyfikatorów `p_user_id` oraz `p_exercise_id`.
2. **Złączenie wielorelacyjne:** Połączenie tabel *workouts*, *workout_exercises* oraz *workout_sets*.
3. **Filtrowanie:** Ograniczenie danych do konkretnego użytkownika, ćwiczenia oraz okresu `INTERVAL '3 months'`.
4. **Obsługa błędów:** Użycie `COALESCE` przy wagach i powtórzeniach przed mnożeniem.
5. **Grupowanie:** Agregacja danych według daty (`GROUP BY`) w celu zsumowania serii z jednego dnia.
6. **Sortowanie:** Uporządkowanie chronologiczne (`ASC`) dla potrzeb wykresów.

3. Algorytm wyznaczania statusu progresji ćwiczenia

Funkcja `get_exercise_progression_status` porównuje tonaż z dwóch ostatnich sesji, zwracając tekstową ocenę trendu (PROGRES, STAGNACJA, REGRES).

Implementacja SQL:

Listing 4.3. Analiza trendu przy użyciu CTE i funkcji okna

```
CREATE OR REPLACE FUNCTION public.get_exercise_progression_status(
    p_user_id integer,
    p_exercise_id integer)
    RETURNS text LANGUAGE 'plpgsql' AS $BODY$
DECLARE
    v_last_total_vol float;
    v_prev_total_vol float;
    v_entry_count integer;
BEGIN
    -- 1. Pobranie objetosci z dwoch ostatnich sesji (CTE)
    WITH exercise_history AS (
        SELECT
            w.id as workout_id,
            w.date,
            SUM(ws.weight * ws.reps) as total_volume
        FROM public.workout_sets ws
        JOIN public.workout_exercises we ON ws.workout_exercise_id = we.id
        JOIN public.workouts w ON we.workout_id = w.id
        WHERE w.user_id = p_user_id
          AND we.exercise_id = p_exercise_id
        GROUP BY w.id, w.date
        ORDER BY w.date DESC
        LIMIT 2
    ),
    ranked_history AS (
        SELECT
            total_volume,
            ROW_NUMBER() OVER (ORDER BY date DESC) as rn
        FROM exercise_history
    )
    SELECT
        MAX(CASE WHEN rn = 1 THEN total_volume END),
        MAX(CASE WHEN rn = 2 THEN total_volume END),
        (SELECT COUNT(*) FROM exercise_history)
    INTO v_last_total_vol, v_prev_total_vol, v_entry_count
    FROM ranked_history;

    -- 2. Obsluga przypadku nowej aktywnosci
    IF v_entry_count < 2 THEN
        RETURN 'NEW';
    END IF;

    -- 3. Logiczne porownanie trendow
    IF v_last_total_vol > v_prev_total_vol THEN
        RETURN 'PROGRESS';
    ELSIF v_last_total_vol = v_prev_total_vol THEN
        RETURN 'STAGNATION';
    ELSE
        RETURN 'REGRESSION';
```

```
    END IF;
END; $BODY$;
```

Kroki algorytmu:

1. **Inicjalizacja:** Przyjęcie `p_user_id` i `p_exercise_id`, deklaracja zmiennych `v_last_total_vol` i `v_prev_total_vol`.
2. **Agregacja historyczna (CTE):** Wyznaczenie objętości dla dwóch ostatnich sesji z użyciem `LIMIT 2`.
3. **Rankingowanie (CTE):** Nadanie numerów porządkowych (`ROW_NUMBER()`) dla odróżnienia ostatniego treningu od przedostatniego.
4. **Ekstrakcja:** Przypisanie wartości do zmiennych za pomocą konstrukcji `CASE WHEN`.
5. **Weryfikacja bazy:** Jeśli `v_entry_count < 2`, zwrócenie statusu `'NEW'`.
6. **Logika porównawcza:** Instrukcje `IF` porównujące wartości tonażu.
7. **Zakończenie:** Zwrócenie etykiety tekstowej do warstwy prezentacji.

4. Algorytm porównawczy objętości tygodniowej

Funkcja `get_volume_comparison` służy do zestawienia aktywności użytkownika w ujęciu tydzień do tygodnia. Pozwala to na szybką ocenę, czy ogólny nakład pracy wzrasta, czy maleje.

Implementacja SQL:

Listing 4.4. Zestawienie tonażu z bieżącego i poprzedniego tygodnia

```
CREATE OR REPLACE FUNCTION public.get_volume_comparison(p_user_id integer)
RETURNS TABLE(current_volume numeric, previous_volume numeric) LANGUAGE 'plpgsql' AS $BODY$
BEGIN
    RETURN QUERY
    SELECT
        -- Tonaz z ostatnich 7 dni
        COALESCE((SELECT SUM(ws.weight * ws.reps)::numeric
         FROM public.workout_sets ws
         JOIN public.workout_exercises we ON ws.workout_exercise_id = we.id
         JOIN public.workouts w ON we.workout_id = w.id
         WHERE w.user_id = p_user_id AND w.date > NOW() - INTERVAL '7 days'), 0),

        -- Tonaz z dni 8-14
        COALESCE((SELECT SUM(ws.weight * ws.reps)::numeric
         FROM public.workout_sets ws
         JOIN public.workout_exercises we ON ws.workout_exercise_id = we.id
         JOIN public.workouts w ON we.workout_id = w.id
         WHERE w.user_id = p_user_id AND w.date <= NOW() - INTERVAL '7 days'
         AND w.date > NOW() - INTERVAL '14 days'), 0);
END; $BODY$;
```

Kroki algorytmu:

1. **Inicjalizacja zapytania:** Przyjęcie identyfikatora użytkownika `p_user_id` jako jedynego parametru wejściowego.
2. **Ekstrakcja danych bieżących:** Wykonanie podzapytania agregującego tonaż dla rekordów, których data (`w.date`) mieści się w przedziale ostatnich 7 dni od momentu wywołania (`NOW()`).
3. **Ekstrakcja danych historycznych:** Wykonanie analogicznego podzapytania dla zakresu od 8 do 14 dni wstecz, co stanowi punkt odniesienia (poprzedni tydzień).
4. **Normalizacja wyników:** Zastosowanie funkcji `COALESCE` dla obu podzapytań, co gwarantuje zwrócenie wartości 0 w przypadku braku aktywności w danym okresie.
5. **Zwrócenie rekordu:** Funkcja przesyła do aplikacji dwa wyniki numeryczne, umożliwiając bezpośrednie porównanie ich w interfejsie graficznym.

5. Algorytm pobierania historycznego kontekstu serii

Funkcja `get_last_exercise_stats` jest wykorzystywana podczas trwania treningu. Pozwala ona użytkownikowi podejrzeć, jakie obciążenie zastosował w konkretnej serii podczas ostatniego wykonywania danego ćwiczenia.

Implementacja SQL:

Listing 4.5. Pobieranie parametrów ostatniej serii o danym numerze

```
CREATE OR REPLACE FUNCTION public.get_last_exercise_stats(
    p_user_id integer, p_exercise_id integer, p_set_no integer)
RETURNS TABLE(last_weight double precision, last_reps integer) LANGUAGE 'sql' AS $BODY$
    SELECT ws.weight, ws.reps
    FROM workout_sets ws
    JOIN workout_exercises we ON ws.workout_exercise_id = we.id
    JOIN workouts w ON we.workout_id = w.id
    WHERE w.user_id = p_user_id AND we.exercise_id = p_exercise_id
      AND ws.set_number = p_set_no
    ORDER BY w.date DESC LIMIT 1;
$BODY$;
```

Kroki algorytmu:

1. **Określenie parametrów serii:** Przyjęcie ID użytkownika, ID ćwiczenia oraz konkretnego numeru serii (`p_set_no`).
2. **Złączenie tabel historycznych:** Połączenie tabel *workouts*, *workout_exercises* i *workout_sets* w celu odnalezienia danych historycznych.
3. **Filtrowanie precyzyjne:** Wybór rekordów pasujących do użytkownika i ćwiczenia, przy jednoczesnym dopasowaniu numeru serii.
4. **Sortowanie chronologiczne:** Uporządkowanie wyników od najnowszego (`ORDER BY w.date DESC`).
5. **Ograniczenie wyniku:** Pobranie wyłącznie pierwszego (najświeższego) rekordu za pomocą klauzuli `LIMIT 1`.

6. Algorytm estymacji rekordu życiowego (1RM)

Algorytm `calculate_exercise_1rm` wyznacza teoretyczny ciężar maksymalny (One Rep Max), jaki użytkownik byłby w stanie podnieść jednorazowo. Wykorzystuje on matematyczny model Brzyckiego.

Implementacja SQL:

Listing 4.6. Wykorzystanie wzoru Brzyckiego w PL/pgSQL

```
-- Fragment kluczowy algorytmu obliczeniowego
IF v_reps = 1 THEN
    v_1rm := v_weight;
ELSE
    -- Zastosowanie modelu matematycznego
    v_1rm := v_weight / (1.0278 - (0.0278 * v_reps));
END IF;
RETURN ROUND((v_1rm * 2)::numeric, 0) / 2;
```

Kroki algorytmu:

1. **Pobranie danych bazowych:** Algorytm identyfikuje najświeższą pierwszą serię (`set_number = 1`) dla danego ćwiczenia i użytkownika.
2. **Ekstrakcja parametrów:** Pobranie wartości ciężaru (`v_weight`) oraz liczby powtórzeń (`v_reps`).
3. **Walidacja danych wejściowych:** Sprawdzenie, czy dane nie są puste lub czy liczba powtórzeń nie wynosi 0. W takim przypadku zwracane jest 0.
4. **Weryfikacja przypadku szczególnego:** Jeśli użytkownik wykonał 1 powtórzenie, to podniesiony ciężar jest uznawany za aktualny rekord (1RM).
5. **Obliczenia matematyczne:** W przypadku większej liczby powtórzeń stosowany jest wzór Brzyckiego: $1RM = \frac{weight}{1.0278 - (0.0278 \times reps)}$.
6. **Zaokrąglanie inżynierskie:** Wynik jest rzutowany na typ `numeric` i zaokrąglany do najbliższego 0,5 kg (standardowy skok obciążenia na siłowni).
7. **Zwrócenie estymacji:** Przekazanie gotowego wyniku `v_1rm` do modułu statystyk.

4.3.2. Moduł dietetyki i analizy biometrycznej

Moduł ten odpowiada za przetwarzanie danych antropometrycznych użytkownika w celu wyznaczenia kluczowych wskaźników zdrowotnych oraz zapotrzebowania energetycznego.

7. Algorytm estymacji poziomu tkanki tłuszczowej (Body Fat)

Funkcja `calculate_user_bf` implementuje metodę Marynarki Wojennej USA (*US Navy Body Fat Method*) do szacowania procentowej zawartości tkanki tłuszczowej na podstawie obwodów ciała. Jest to kluczowy wskaźnik pozwalający na monitorowanie składu ciała, a nie tylko masy całkowitej.

Implementacja SQL:

Listing 4.7. Implementacja wzoru US Navy dla obu płci

```
SELECT height, waist, neck, hips INTO v_height, v_waist, v_neck, v_hips
FROM public.body_measurements
WHERE user_id = p_user_id
ORDER BY date DESC LIMIT 1;

-- Walidacja matematyczna (zabezpieczenie logarytmow)
IF v_height IS NULL OR v_waist IS NULL OR v_neck IS NULL OR (v_waist - v_neck) <= 0 THEN
    RETURN NULL;
END IF;

IF v_gender = 'male' THEN
    v_bf := 495 / (1.0324 - 0.19077 * log(v_waist - v_neck) + 0.15456 * log(v_height)) - 450;
ELSE
    IF v_hips IS NULL OR (v_waist + v_hips - v_neck) <= 0 THEN RETURN NULL; END IF;
    v_bf := 495 / (1.29579 - 0.35004 * log(v_waist + v_hips - v_neck) + 0.22100 * log(v_height)) - 450;
END IF;

IF v_bf < 2 THEN RETURN 2.0; END IF;
RETURN ROUND(v_bf::numeric, 1);
END; BODY;
```

Kroki algorytmu:

1. **Przygotowanie danych demograficznych:** Pobranie płci użytkownika z tabeli *users*. Zastosowanie funkcji `LOWER` i `TRIM` zapewnia odporność algorytmu na różną wielkość liter w bazie danych.
2. **Ekstrakcja pomiarów antropometrycznych:** Pobranie najświeższych wpisów z tabeli `body_measurements` (wzrost, obwód talii, szyi oraz bioder). Sortowanie po dacie gwarantuje aktualność wyniku końcowego.
3. **Walidacja bezpieczeństwa matematycznego:** Sprawdzenie, czy dane nie są puste (`NULL`) oraz czy różnica obwodów jest dodatnia. Jest to niezbędne, ponieważ logarytm naturalny (`log`) nie jest zdefiniowany dla liczb niedodatnich i spowodowałby błąd wykonania funkcji.
4. **Dyferencjacja płciowa (Branching):** Wybór odpowiedniego modelu matematycznego w zależności od wartości zmiennej `v_gender`:
    - **Dla mężczyzn:** Wykorzystanie logarytmicznego stosunku obwodów talii (`v_waist`) i szyi (`v_neck`).
    - **Dla kobiet:** Uwzględnienie dodatkowo obwodu bioder (`v_hips`), co jest kluczowe dla specyfiki kobiecej kompozycji ciała.

5. **Korekta błędów grubych:** Zastosowanie progu dolnego (2.0%), aby zapobiec zwracaniu nierealnych wskaźników w przypadku błędnie wprowadzonych danych przez użytkownika.

6. **Normalizacja wyniku:** Zaokrąglenie obliczonej wartości `v_bf` do jednego miejsca po przecinku za pomocą funkcji `ROUND` i zwrot wyniku do aplikacji.

8. Algorytm obliczania zapotrzebowania energetycznego (BMR/TDEE)

Funkcja `calculate_user_diet_calories` odpowiada za wyznaczenie dziennego zapotrzebowania kalorycznego użytkownika. Algorytm opiera się na uznanym wzorze Mifflina-St Jeora, a następnie modyfikuje wynik w zależności od poziomu aktywności fizycznej (*TDEE*) oraz zadeklarowanego celu treningowego.

Implementacja SQL:

Listing 4.8. Obliczanie kalorii przy użyciu wzoru Mifflina-St Jeora

```
SELECT weight, height, activity_level, goal
INTO v_weight, v_height, v_activity, v_goal
FROM public.body_measurements
WHERE user_id = p_user_id ORDER BY date DESC LIMIT 1;

-- 2. Walidacja danych (wartosci domyslne)
v_weight := COALESCE(v_weight, 70.0);
v_height := COALESCE(v_height, 175.0);
v_activity := COALESCE(v_activity, 1.2);

-- 3. Obliczenie BMR (Mifflin-St Jeor)
IF lower(v_gender) LIKE 'm%' THEN
    v_bmr := (10 * v_weight) + (6.25 * v_height) - (5 * v_age_const) + 5;
ELSE
    v_bmr := (10 * v_weight) + (6.25 * v_height) - (5 * v_age_const) - 161;
END IF;

-- 4. Wyznaczenie TDEE i korekta o cel sylwetkowy
v_tdee := round(v_bmr * v_activity);

IF v_goal = 'Zbudowanie masy miesniowej' THEN
    recommended_calories := round(v_tdee * 1.10);
    goal_label := 'Masa';
ELSIF v_goal = 'Redukcja tkanki tluszczowej' THEN
    recommended_calories := round(v_tdee * 0.80);
    goal_label := 'Redukcja';
ELSE
    recommended_calories := v_tdee;
    goal_label := 'Rekompozycja';
END IF;

difference_from_tdee := recommended_calories - v_tdee;
RETURN NEXT;
END; BODY;
```

Kroki algorytmu:

1. **Agregacja danych wejściowych:** System pobiera płeć z profilu użytkownika oraz najnowsze parametry fizyczne (masa ciała `v_weight`, wzrost `v_height`, poziom aktywności `v_activity` oraz cel `v_goal`) z tabeli pomiarów biometrycznych.

2. **Zapewnienie ciągłości obliczeń:** Poprzez funkcję `COALESCE`, algorytm podstawia wartości uśrednione w przypadku braku kompletnych pomiarów, co zapobiega błędom w warstwie prezentacji.

3. **Obliczanie podstawowej przemiany materii (BMR):** Algorytm stosuje dyferencjację ze względu na płeć (`v_gender`), wykorzystując stałą wieku `v_age_const`. Wynik określa liczbę kalorii niezbędną do podtrzymania funkcji życiowych.

4. **Wyznaczenie całkowitego zapotrzebowania (TDEE):** Wartość `v_bmr` jest mnożona przez współczynnik aktywności fizycznej (`v_activity`), odzwierciedlający styl życia użytkownika.

5. **Aplikacja strategii dietetycznej:** W zależności od wybranego celu (`v_goal`), system wyznacza końcową podaż energii:

   - **Nadwyżka (Masa):** Zwiększenie zapotrzebowania o 10% (`v_tdee * 1.10`).
   - **Deficyt (Redukcja):** Obniżenie zapotrzebowania o 20% (`v_tdee * 0.80`).
   - **Zero kaloryczne (Rekompozycja):** Utrzymanie podaży na poziomie `v_tdee`.

6. **Analiza różnicy bilansu:** Obliczana jest zmienna `difference_from_tdee`, która informuje użytkownika o wielkości zastosowanego deficytu lub nadwyżki.

7. **Zwrócenie zestawu danych:** Funkcja zwraca sformatowany rekord zawierający docelową kaloryczność, etykietę celu oraz różnicę bilansową.

9. Algorytm podziału makroskładników dietetycznych

Funkcja `get_user_macros` dokonuje podziału całkowitej puli kalorii na poszczególne makroskładniki: białka, tłuszcze i węglowodany. Algorytm stosuje podejście hybrydowe – zapotrzebowanie na białko obliczane jest na podstawie masy ciała, natomiast tłuszcze i węglowodany są wyznaczane jako procentowy udział w całkowitej podaży energii.

Implementacja SQL:

Listing 4.9. Obliczanie gramatury makroskładników w PL/pgSQL

```
-- 2. Pobranie najnowszej masy ciala
SELECT weight INTO v_weight FROM public.body_measurements
WHERE user_id = p_user_id ORDER BY date DESC LIMIT 1;

v_weight := COALESCE(v_weight, 70.0);
calories_total := v_kcal;

-- 3. Obliczenia szczegolowe
protein_g := round(v_weight * 2.0); -- 2g / kg m.c.
fat_g := round((v_kcal * 0.25) / 9); -- 25% energii
carbs_g := round((v_kcal - (protein_g * 4) - (fat_g * 9)) / 4); -- Reszta

RETURN NEXT;
END; BODY;
```

Kroki algorytmu:

1. **Integracja z modułem kaloryczności:** Algorytm wywołuje funkcję `calculate_user_diet_calories`, aby uzyskać wartość `v_kcal` (rekomendowaną podaż energii), co zapewnia spójność danych w całym systemie.

2. **Ekstrakcja masy ciała:** Z tabeli `body_measurements` pobierana jest zmienna `v_weight`. W przypadku braku danych, stosowana jest funkcja `COALESCE` z wartością domyślną 70,0 kg.

3. **Priorytetyzacja białka:** W pierwszej kolejności wyznaczana jest wartość `protein_g` według standardu sportowego: 2 g na każdy kilogram masy ciała. Wartość ta jest następnie przeliczana na kilokalorie (przy założeniu 4 kcal za 1 g).

4. **Wyznaczenie limitu tłuszczów:** Ilość tłuszczów (`fat_g`) jest ustalana jako 25% całkowitego zapotrzebowania energetycznego. Wynik jest dzielony przez 9 (gęstość energetyczna tłuszczu), zgodnie z regułą: $fat\_g = \frac{v\_kcal \times 0{,}25}{9}$.

5. **Dopełnienie węglowodanami:** Zmienna `carbs_g` jest obliczana jako „reszta" energetyczna. Od całkowitej puli `v_kcal` odejmowana jest energia pochodząca z białek i tłuszczów, a pozostała wartość jest dzielona przez 4 kcal (gęstość energetyczna węglowodanów).

6. **Finalizacja i zwrot danych:** Wyniki są zaokrąglane do pełnych gramów za pomocą funkcji `round` i przekazywane do interfejsu użytkownika jako kompletny rekord *Macros*.

10. Procedura ekstrakcji historii pomiarów biometrycznych

Funkcja `get_user_measurements` służy do pobierania pełnej, chronologicznej historii danych antropometrycznych przypisanych do konkretnego konta użytkownika. Zapewnia ona ustandaryzowany zestaw danych wynikowych, który jest niezbędny do generowania wykresów progresji sylwetkowej oraz zestawień tabelarycznych w interfejsie użytkownika.

Implementacja SQL:

Listing 4.10. Funkcja SQL do pobierania historii pomiarów ciała

```
CREATE OR REPLACE FUNCTION public.get_user_measurements(p_user_id integer)
RETURNS TABLE(
    measurement_id integer,
    date timestamp,
    height double precision,
    weight double precision,
    chest double precision,
    waist double precision,
    neck double precision,
    biceps double precision,
    thighs double precision,
    hips double precision
) LANGUAGE 'sql' AS $BODY$
    SELECT
        bm.id, bm.date, bm.height, bm.weight,
        bm.chest, bm.waist, bm.neck, bm.biceps,
        bm.thighs, bm.hips
    FROM public.body_measurements bm
    WHERE bm.user_id = p_user_id
    ORDER BY bm.date ASC;
$BODY$;
```

Kroki algorytmu:

1. **Weryfikacja kontekstu użytkownika:** Algorytm przyjmuje parametr wejściowy `p_user_id`, który stanowi klucz do filtrowania rekordów. Dzięki temu zapytanie zwraca wyłącznie dane należące do zalogowanej osoby.

2. **Agregacja kompletu danych antropometrycznych:** Funkcja wybiera z tabeli `body_measurements` wszystkie kluczowe parametry: masę ciała (`weight`), wzrost (`height`), obwód szyi (`neck`) oraz obwody klatki piersiowej (`chest`), talii (`waist`), bicepsa (`biceps`), ud (`thighs`) i bioder (`hips`).

3. **Identyfikacja rekordów:** Do strumienia danych dołączane jest pole `measurement_id` (pochodzące z kolumny `bm.id`). Pozwala to aplikacji na precyzyjne zarządzanie konkretnymi wpisami w historii.

4. **Utrzymanie porządku chronologicznego:** Kluczowym elementem algorytmu jest sortowanie według pola `date` w porządku rosnącym (`ASC`). Gwarantuje to poprawność renderowania osi czasu.

5. **Zwrócenie ustrukturyzowanej tabeli:** Wynik działania funkcji trafia do aplikacji jako obiekt typu `TABLE`, co pozwala na uniknięcie błędów typu *Undefined array key* w kodzie PHP, ponieważ każda kolumna (w tym `neck`) jest jawnie zdefiniowana.

4.3.3. Moduł Inteligencji Projektowej i Struktury

Ten rozdział opisuje algorytmy odpowiedzialne za automatyczną analizę schematów treningowych oraz weryfikację balansu strukturalnego planów realizowanych przez użytkowników.

11. Algorytm detekcji systemu treningowego (Split Detection)

Funkcja `detect_training_split` dokonuje heurystycznej analizy danych z ostatnich 30 dni aktywności w celu sklasyfikowania systemu treningowego. Jest to kluczowy element personalizacji raportów progresu.

Implementacja SQL:

Listing 4.11. Funkcja klasyfikująca system treningowy na podstawie statystyk

```
CREATE OR REPLACE FUNCTION public.detect_training_split(p_user_id integer)
RETURNS text LANGUAGE 'plpgsql' AS $BODY$
DECLARE
    v_avg_groups_per_workout FLOAT;
    v_days_active_30d INT;
    v_total_groups_30d INT;
    v_has_legs BOOLEAN; v_has_push BOOLEAN; v_has_pull BOOLEAN;
    v_result TEXT;
BEGIN
    SELECT
        COUNT(DISTINCT s.workout_id),
        COUNT(DISTINCT s.muscle_group_name),
        EXISTS (SELECT 1 FROM crud.get_user_training_stats(p_user_id) x WHERE x.muscle_group_name ILIKE
        EXISTS (SELECT 1 FROM crud.get_user_training_stats(p_user_id) x WHERE x.muscle_group_name IN ('K
        EXISTS (SELECT 1 FROM crud.get_user_training_stats(p_user_id) x WHERE x.muscle_group_name IN ('P
    INTO v_days_active_30d, v_total_groups_30d, v_has_legs, v_has_push, v_has_pull
    FROM crud.get_user_training_stats(p_user_id) s;

    SELECT AVG(daily_count) INTO v_avg_groups_per_workout
    FROM (
        SELECT COUNT(DISTINCT muscle_group_name) as daily_count
        FROM crud.get_user_training_stats(p_user_id)
        GROUP BY workout_id
    ) AS subquery;

    IF v_days_active_30d = 0 OR v_avg_groups_per_workout IS NULL THEN
        RETURN 'Brak␣aktywnosci␣(30␣dni)';
    END IF;

    -- Logika klasyfikacji
    IF v_avg_groups_per_workout >= 3.8 THEN v_result := 'Full␣Body␣Workout';
    ELSIF v_has_push AND v_has_pull AND NOT v_has_legs THEN v_result := 'Push␣Pull␣(No␣Legs)';
    ELSE v_result := 'Wlasny␣system'; END IF;

    RETURN v_result;
END; $BODY$;
```

Kroki algorytmu:

1. **Agregacja statystyk okresowych:** Funkcja korzysta z widoku `crud.get_user_training_stats` w celu wyznaczenia liczby aktywnych dni (`v_days_active_30d`) oraz unikalnych grup mięśniowych trenowanych w ciągu ostatnich 30 dni.
2. **Analiza wektorów treningowych:** Poprzez zapytania `EXISTS` system sprawdza obecność kluczowych wzorców ruchowych: dolnych partii ciała (`v_has_legs`), ruchów wyciskających (`v_has_push`) oraz przyciągających (`v_has_pull`).
3. **Obliczanie gęstości treningu:** Wyznaczana jest średnia liczba grup mięśniowych przypadająca na jedną sesję (`v_avg_groups_per_workout`).
4. **Klasyfikacja logiczna:** System przechodzi przez zestaw warunków decyzyjnych w celu dopasowania wzorca do znanych systemów (np. FBW, Split).
5. **Obsługa braku danych:** W przypadku braku wpisów w tabeli `workout_sets`, algorytm zwraca informację o braku aktywności.

12. Algorytm analizy balansu strukturalnego (Muscle Balance)

Algorytm `get_user_muscle_balance` oblicza procentowy udział objętościowy poszczególnych partii mięśniowych w skali ostatniego tygodnia.

Implementacja SQL:

Listing 4.12. Analiza procentowego rozkladu objetosci treningowej

```
CREATE OR REPLACE FUNCTION public.get_user_muscle_balance(p_user_id integer)
RETURNS TABLE(muscle_group_name text, volume_percentage numeric) AS $BODY$
DECLARE
    v_total_volume numeric;
BEGIN
    SELECT SUM(ws.weight * ws.reps)::numeric INTO v_total_volume
    FROM public.workout_sets ws
    JOIN public.workout_exercises we ON ws.workout_exercise_id = we.id
    JOIN public.workouts w ON we.workout_id = w.id
    WHERE w.user_id = p_user_id AND w.date > NOW() - INTERVAL '7 days';

    IF v_total_volume IS NULL OR v_total_volume = 0 THEN RETURN; END IF;

    RETURN QUERY
    SELECT
        mg.name::text,
        ROUND(((SUM(ws.weight * ws.reps) / v_total_volume) * 100)::numeric, 1) as percentage
    FROM public.workout_sets ws
    JOIN public.workout_exercises we ON ws.workout_exercise_id = we.id
    JOIN public.workouts w ON we.workout_id = w.id
    JOIN public.exercises e ON we.exercise_id = e.id
    JOIN public.muscle_groups mg ON e.muscle_group_id = mg.id
    WHERE w.user_id = p_user_id AND w.date > NOW() - INTERVAL '7 days'
    GROUP BY mg.name ORDER BY percentage DESC;
END; $BODY$;
```

Kroki algorytmu:

1. **Kalkulacja objętości całkowitej:** System sumuje iloczyn ciężaru i powtórzeń ze wszystkich serii wykonanych przez użytkownika (`p_user_id`). Wynik zapisywany jest w zmiennej `v_total_volume`.
2. **Złączenia relacyjne:** Funkcja łączy tabele `workout_sets`, `workout_exercises`, `exercises` oraz `muscle_groups`.
3. **Normalizacja procentowa:** Dla każdej grupy mięśniowej obliczany jest stosunek jej objętości do objętości całkowitej.
4. **Precyzja obliczeń:** Zastosowano rzutowanie na typ `numeric` oraz funkcję `ROUND` dla poprawy czytelności.
5. **Sortowanie hierarchiczne:** Rekordy są zwracane w kolejności malejącej według kolumny `percentage`.

4.3.4. Moduł Integracji i Operacji Złożonych

Ten rozdział opisuje funkcje integrujące wiele operacji bazodanowych w ramach jednej transakcji oraz zaawansowane procedury odczytu danych historycznych i autoryzacji.

13. Algorytm autoryzacji użytkowników (User Authentication)

Funkcja `login_by_username` stanowi warstwę uwierzytelniania bezpośrednio na poziomie bazy danych. Została zaprojektowana w celu szybkiej weryfikacji tożsamości przy minimalnym narzucie komunikacyjnym.

Implementacja SQL:

Listing 4.13. Procedura logowania użytkownika

```
CREATE OR REPLACE FUNCTION public.login_by_username(
    p_username character varying,
    p_password character varying)
RETURNS TABLE(user_id integer, first_name character varying)
LANGUAGE 'sql' AS $BODY$
    SELECT u.id, u.first_name
    FROM users u
    WHERE u.username = p_username
      AND u.password = p_password;
$BODY$;
```

Kroki algorytmu:

- **Walidacja parametrów:** System przyjmuje ciągi znaków `p_username` oraz `p_password`.
- **Wyszukiwanie binarne:** Dzięki indeksowi na kolumnie `username`, baza danych lokalizuje rekord użytkownika.
- **Weryfikacja tożsamości:** Porównanie haseł następuje po stronie silnika bazy danych w klauzuli `WHERE`.
- **Zwracanie kontekstu:** W przypadku sukcesu funkcja zwraca `user_id` oraz `first_name`.

14. Algorytm filtrowania zasobów ćwiczeń (Exercise Discovery)

Funkcja `get_exercises_by_muscle_group` wspiera interfejs użytkownika w procesie planowania treningu, oferując dynamiczne filtrowanie bazy ćwiczeń według przypisanych grup mięśniowych.

Implementacja SQL:

Listing 4.14. Filtrowanie bazy ćwiczeń według grup mięśniowych

```
CREATE OR REPLACE FUNCTION public.get_exercises_by_muscle_group(
    p_muscle_group_name character varying)
RETURNS TABLE(exercise_id integer, exercise_name character varying, muscle_group_name
    character varying)
LANGUAGE 'sql' AS $BODY$
    SELECT
        e.id,
        e.name,
        mg.name
    FROM exercises e
    JOIN muscle_groups mg ON e.muscle_group_id = mg.id
    WHERE mg.name ILIKE p_muscle_group_name
    ORDER BY e.name;
$BODY$;
```

Kroki algorytmu:

- **Relacyjne złączenie danych:** System wykonuje operację `JOIN` między tabelą `exercises` a `muscle_groups`.
- **Dopasowanie wzorca:** Użycie operatora `ILIKE` pozwala na elastyczne wyszukiwanie bez względu na wielkość liter.
- **Sortowanie alfabetyczne:** Zastosowanie `ORDER BY` gwarantuje spójność wyświetlanej listy ćwiczeń (`e.name`).

15. Algorytm odczytu szczegółów treningu (Workout Detailed View)

Funkcja `get_detailed_workout` służy do agregacji danych o konkretnej sesji treningowej. Algorytm łączy informacje o ćwiczeniach, seriach, ciężarze i liczbie powtórzeń w jedną strukturę wynikową.

Implementacja SQL:

Listing 4.15. Pobieranie szczegółowych danych o treningu

```
CREATE OR REPLACE FUNCTION public.get_detailed_workout(
    p_workout_id integer)
RETURNS TABLE(exercise_name character varying, set_number integer, weight numeric, reps
    integer, rpe integer)
LANGUAGE 'sql' AS $BODY$
    SELECT
        e.name,
        ws.set_number,
        ws.weight,
        ws.reps,
        ws.rpe
    FROM workout_exercises we
    JOIN exercises e ON we.exercise_id = e.id
    JOIN workout_sets ws ON we.id = ws.workout_exercise_id
    WHERE we.workout_id = p_workout_id
```

```
    ORDER BY we.id, ws.set_number;
$BODY$;
```

Kroki algorytmu:

- **Identyfikacja sesji:** Przyjęcie `p_workout_id` jako klucza głównego do filtrowania danych.
- **Relacyjne złączenie danych:** System wykonuje operacje `JOIN` między tabelami `workout_exercises`, `exercises` oraz `workout_sets`.
- **Sortowanie strukturalne:** Zastosowanie `ORDER BY` zapewnia poprawną kolejność wyświetlania ćwiczeń i serii.

16. Algorytm zestawienia historii treningowej (Workout History)

Funkcja `get_user_workout_history` odpowiada za dostarczenie danych do widoku kalendarza lub listy archiwalnej użytkownika.

Implementacja SQL:

Listing 4.16. Pobieranie historii treningów użytkownika

```
CREATE OR REPLACE FUNCTION public.get_user_workout_history(
    p_user_id integer)
RETURNS TABLE(workout_id integer, workout_date timestamp without time zone, total_volume
     numeric, exercise_count bigint)
LANGUAGE 'sql' AS $BODY$
    SELECT
        w.id,
        w.date,
        SUM(ws.weight * ws.reps) as total_volume,
        COUNT(DISTINCT we.exercise_id) as exercise_count
    FROM workouts w
    LEFT JOIN workout_exercises we ON w.id = we.workout_id
    LEFT JOIN workout_sets ws ON we.id = ws.workout_exercise_id
    WHERE w.user_id = p_user_id
    GROUP BY w.id, w.date
    ORDER BY w.date DESC;
$BODY$;
```

Kroki algorytmu:

- **Agregacja objętości:** Obliczanie sumy iloczynów ciężaru i powtórzeń (`SUM`) dla wszystkich serii w treningu.
- **Analiza różnorodności:** Wyznaczenie liczby unikalnych ćwiczeń przy użyciu funkcji `COUNT(DISTINCT)`.
- **Sortowanie chronologiczne:** Zastosowanie `ORDER BY DESC` dla zapewnienia widoczności najnowszych wpisów na górze listy.

17. Algorytm zapisu kompletnej sesji (Atomic Workout Save)

Procedura `save_complete_workout` zapewnia spójność danych poprzez zapis nagłówka treningu w ramach jednej operacji.

Implementacja SQL:

Listing 4.17. Zapis nagłówka sesji treningowej

```
CREATE OR REPLACE PROCEDURE public.save_complete_workout(
    p_user_id integer,
    p_date timestamp without time zone,
    p_notes text)
LANGUAGE 'plpgsql' AS $BODY$
BEGIN
    INSERT INTO workouts (user_id, date, notes)
    VALUES (p_user_id, p_date, p_notes);
END;
$BODY$;
```

Kroki algorytmu:

- **Inicjalizacja rekordu:** Wstawienie danych podstawowych (identyfikator użytkownika, data) do tabeli `workouts`.
- **Persystencja uwag:** Zapis opcjonalnych notatek treningowych przekazanych w parametrze `p_notes`.

4.3.5. Podsumowanie logiki bazodanowej

Przedstawione powyżej algorytmy oraz procedury składowane stanowią fundament operacyjny systemu *AwareFit*. Dzięki przeniesieniu kluczowych obliczeń (takich jak estymacja *1RM*, wyliczanie objętości treningowej czy zapotrzebowania kalorycznego) bezpośrednio do warstwy bazy danych, uzyskano wysoką wydajność oraz spójność danych niezależnie od platformy klienckiej.

Zaimplementowana logika zapewnia pełną automatyzację procesów analitycznych, co minimalizuje obciążenie urządzenia końcowego i pozwala na błyskawiczne generowanie raportów progresu użytkownika.

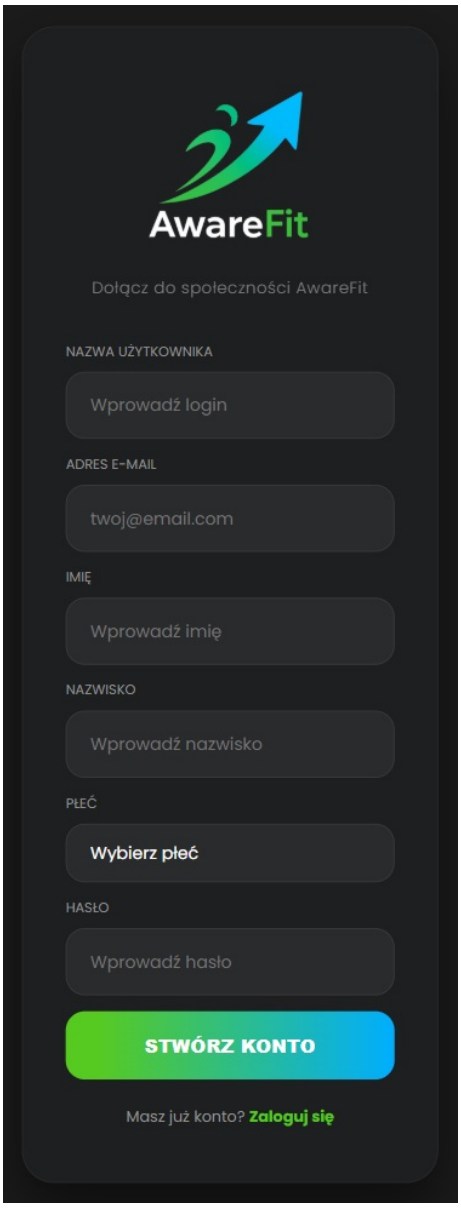
W kolejnym rozdziale dokumentacji przedstawiona zostanie implementacja warstwy wizualnej systemu. Zostanie tam zaprezentowany interfejs graficzny użytkownika (GUI), który w sposób intuicyjny wykorzystuje opisane wcześniej funkcje i procedury, przekładając surowe dane bazodanowe na czytelne widoki, wykresy oraz formularze interaktywne.

5. Prezentacja interfejsu graficznego (GUI)

W niniejszym rozdziale przedstawiono warstwę wizualną aplikacji *AwareFit*, która stanowi interfejs komunikacji użytkownika z bazą danych. Każdy z prezentowanych widoków został zaprojektowany z myślą o intuicyjności oraz responsywności. Kluczowym aspektem opisu jest wskazanie bezpośrednich powiązań pomiędzy elementami interfejsu a logiką bazodanową i algorytmami szczegółowo opisanymi w rozdziale 4.

5.1. Widok rejestracji użytkownika

Pierwszym etapem interakcji z systemem jest proces rejestracji, który pozwala na utworzenie unikalnego konta użytkownika. Interfejs gromadzi podstawowe dane uwierzytelniające oraz parametry profilowe.



Rys. 5.1. Interfejs formularza rejestracji użytkownika

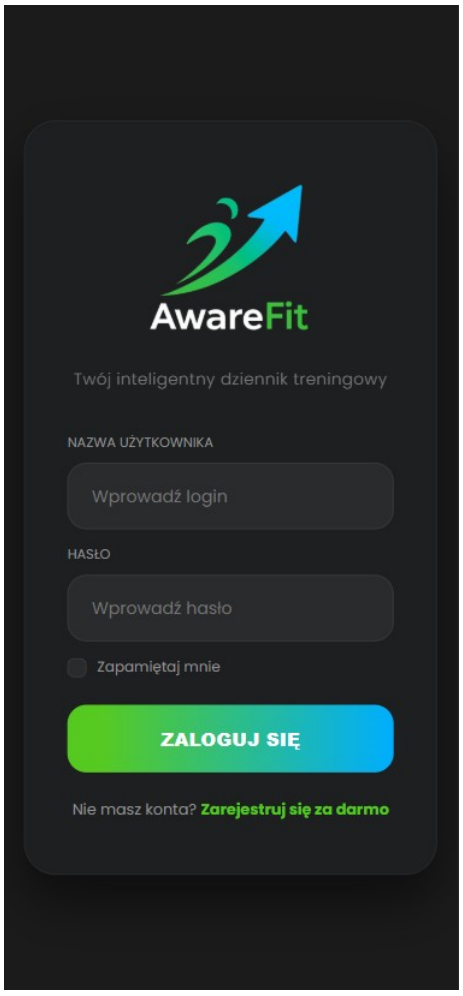
Powiązanie z logiką bazy danych: Powiązanie z logiką bazy danych:
Proces rejestracji obsługiwany jest przez skrypt backendowy `register.php`. Po stronie serwera następuje odebranie danych z formularza metodą `POST`, a następnie wywoływana jest funkcja bazodanowa `crud.insert_user`. Odpowiada ona za:

- **Walidację unikalności:** Sprawdzenie, czy podany adres e-mail lub username nie istnieje już w tabeli `users`.
- **Persystencję danych:** Trwałe zapisanie hasła oraz danych profilowych.
- **Inicjalizację konta:** Automatyczne nadanie identyfikatora użytkownika (`user_id`), który jest niezbędny do działania pozostałych modułów systemu.

Funkcjonalność GUI: Użytkownik wypełnia pola tekstowe, a system po zatwierdzeniu formularza przesyła dane do procedury bazodanowej.

5.2. Widok logowania

Widok logowania stanowi główny punkt dostępu do chronionej części aplikacji. Został zaprojektowany w sposób minimalistyczny, z wyraźnym podziałem na sekcję wizualną (branding) oraz funkcjonalną (formularz).



Rys. 5.2. Interfejs ekranu logowania systemu AwareFit

Powiązanie z logiką bazy danych:
Logika autoryzacji realizowana jest w pliku `index.php`. Po przesłaniu formularza metodą `POST`, skrypt nawiązuje połączenie z bazą danych i wykorzystuje funkcję bazodanową `public.login_by_username`. Proces ten obejmuje:

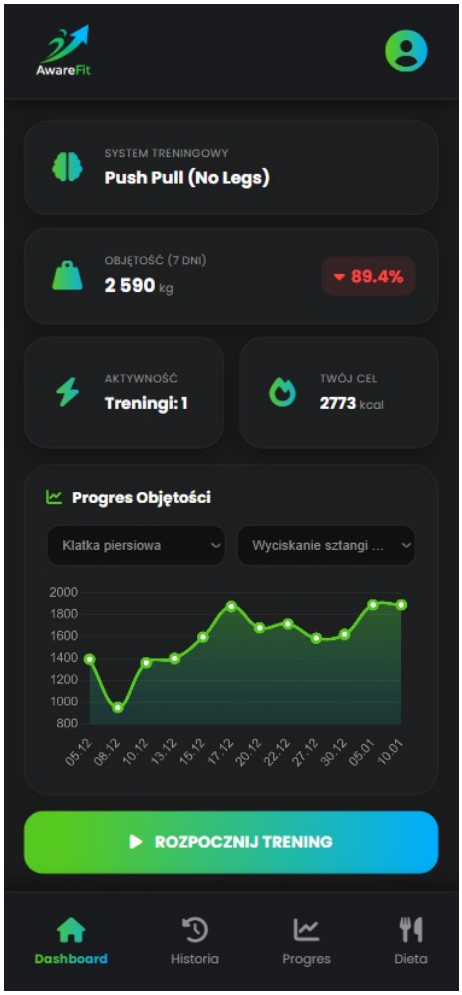
- **Weryfikację poświadczeń:** Funkcja przyjmuje login (`username`) oraz hasło, a następnie sprawdza ich poprawność w tabeli `users`.
- **Ekstrakcję danych sesyjnych:** W przypadku sukcesu, funkcja zwraca rekord zawierający `user_id` oraz `first_name`, co pozwala na zainicjowanie zmiennych sesyjnych w PHP (`$_SESSION`).
- **Zarządzanie dostępem:** Jeśli funkcja nie zwróci żadnego rekordu, skrypt generuje komunikat o błędzie (*„Nieprawidłowy login lub hasło”*), blokując dostęp do dalszych modułów systemu.

Funkcjonalność GUI:
Interfejs zawiera mechanizmy poprawiające doświadczenie użytkownika (*User Experience*), takie jak:

- Przejrzysta sekcja komunikatów o błędach (widoczna w przypadku nieudanej próby logowania).
- Zastosowanie atrybutów `autocomplete`, co zwiększa bezpieczeństwo i wygodę korzystania z menedżerów haseł.
- Bezpośredni odnośnik do formularza rejestracji (`register.php`) dla nowych użytkowników.

5.3. Panel główny (Dashboard)

Panel główny (`dashboard.php`) pełni funkcję centrum analitycznego dla użytkownika. Agreguje on surowe dane treningowe i dietetyczne, prezentując je w formie czytelnych kart informacyjnych (widżetów) oraz interaktywnego wykresu progresji.



Rys. 5.3. Panel główny aplikacji z systemem widżetów i wykresem progresu

Powiązanie z logiką bazy danych:
Widok ten jest jednym z najbardziej zaawansowanych elementów systemu pod kątem integracji z bazą danych. Wykorzystuje on szereg funkcji opisanych w poprzednich rozdziałach:

- **System treningowy i aktywność:** Skrypt wywołuje funkcję `public.detect_training_split` w celu identyfikacji aktualnego schematu ćwiczeń oraz wykonuje zliczanie rekordów z tabeli `workouts` z ostatnich 7 dni.
- **Analiza objętości i trendów:** Poprzez funkcję `public.get_volume_comparison` system pobiera dane o sumarycznym ciężarze podniesionym w obecnym i poprzednim tygodniu, wyliczając procentowy trend (`$diff_percent`).
- **Moduł żywieniowy:** Wykorzystywane są funkcje `calculate_user_diet_calories` oraz `get_user_macros`, które na podstawie profilu użytkownika zwracają dobowe zapotrzebowanie na kalorie i makroskładniki.

- **Progresja ćwiczeń (Wykres):** Dynamiczny wykres generowany jest dzięki funkcji `get_exercise_volume_progression`, która dostarcza historyczne dane o objętości dla konkretnego, wybranego przez użytkownika ćwiczenia.

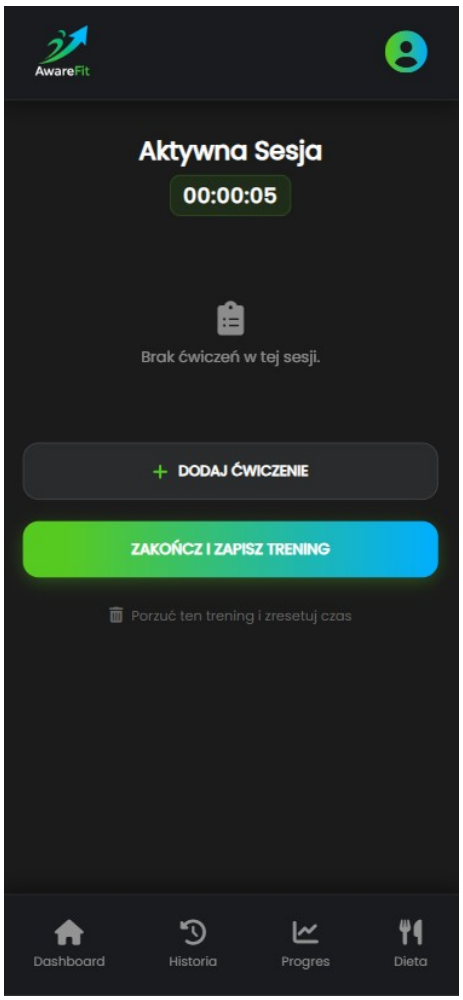
Funkcjonalność GUI:

Interfejs oferuje wysoką interaktywność dzięki zastosowaniu technologii *Chart.js* oraz asynchronicznej komunikacji:

- **Karty Progress Card:** Wykorzystują system ikon (*FontAwesome*) do szybkiej wizualizacji statusu treningowego i trendów objętości.
- **Interaktywny Wykres:** Użytkownik może filtrować dane za pomocą list rozwijanych (*Select*), co powoduje przeładowanie widoku z nowymi parametrami dla wybranego ćwiczenia.
- **Modal Makroskładników:** Kliknięcie w kartę kalorii wywołuje okno modalne (`macroModal`), prezentujące szczegółowy rozkład białek, tłuszczy i węglowodanów w formie graficznej.

5.4. Widok aktywnej sesji treningowej (Workout Session)

Moduł `workout.php` stanowi najbardziej interaktywną część aplikacji, pełniąc rolę cyfrowego asystenta treningowego. Wykorzystuje on architekturę *Single Page Application* (SPA) wewnątrz skryptu PHP – widoki przełączane są dynamicznie przez JavaScript (`workout.js`), co zapewnia płynność pracy bez przeładowywania strony.



Rys. 5.4. Główny ekran aktywnej sesji ze stoperem i listą ćwiczeń

Powiązanie z logiką bazy danych:
Mimo że sesja przechowywana jest tymczasowo w pamięci lokalnej przeglądarki, system stale komunikuje się z bazą danych:

- **Pobieranie struktur:** Wykorzystanie funkcji `crud.get_all_muscle_groups` do wygenerowania kafelków partii mięśniowych.
- **Inteligentne podpowiedzi (Coach Advice):** Skrypt `get_coach_advice.php` analizuje historyczne wyniki (algorytm *PROGRESSION*) i wyświetla sugestie w czasie rzeczywistym.
- **Dynamiczne placeholdery:** Dzięki `get_last_stats.php`, pola formularza podpowiadają ciężar i powtórzenia z poprzedniej sesji.
- **Persystencja danych:** Zakończenie treningu wywołuje skrypt `save_workout.php`, który przesyła obiekt JSON do procedury `save_complete_workout`.

Struktura procesu i widoki interfejsu:

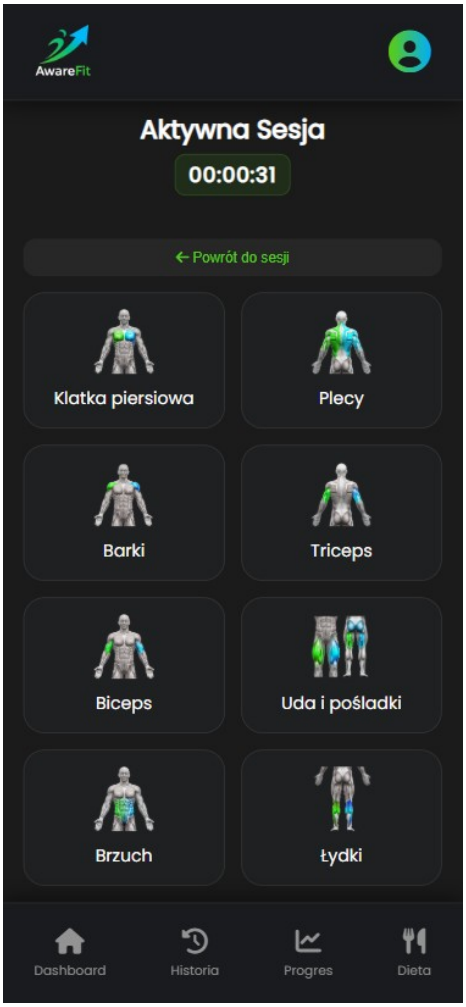
1. **Wybór partii mięśniowej:** Użytkownik widzi siatkę kafelków z ikonami reprezentującymi grupy mięśniowe (Rys. 5.5).

2. **Lista ćwiczeń:** Po wybraniu partii, skrypt `get_exercises.php` zwraca listę ćwiczeń przypisanych do danej grupy (Rys. 5.6).

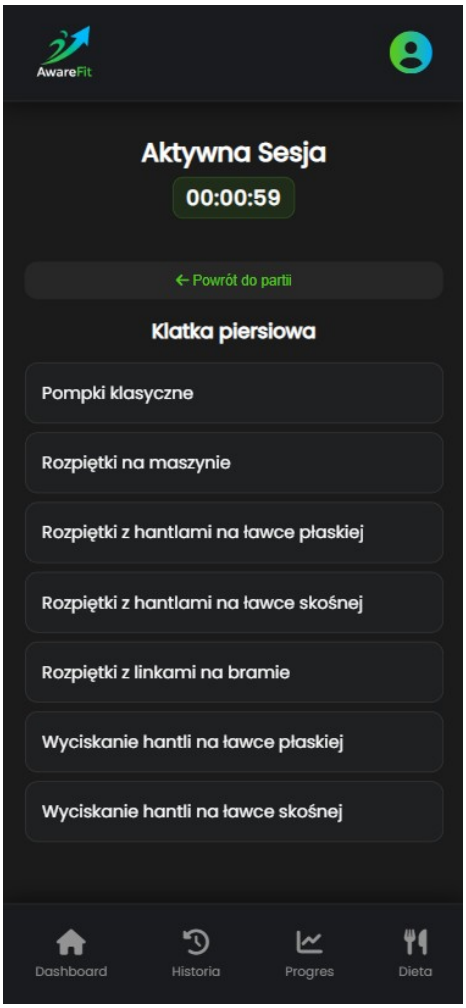
3. **Logowanie serii:** Ekran `log-set` umożliwia wprowadzanie danych. Wykorzystuje on system kolorowych ramek „Coach Advice" zależnych od statusu progresji (Rys. 5.7).

4. **Podsumowanie sesji:** Widok prezentuje wszystkie wykonane ćwiczenia wraz z ich seriami, pozwalając na edycję sesji przed jej finalnym zapisem.

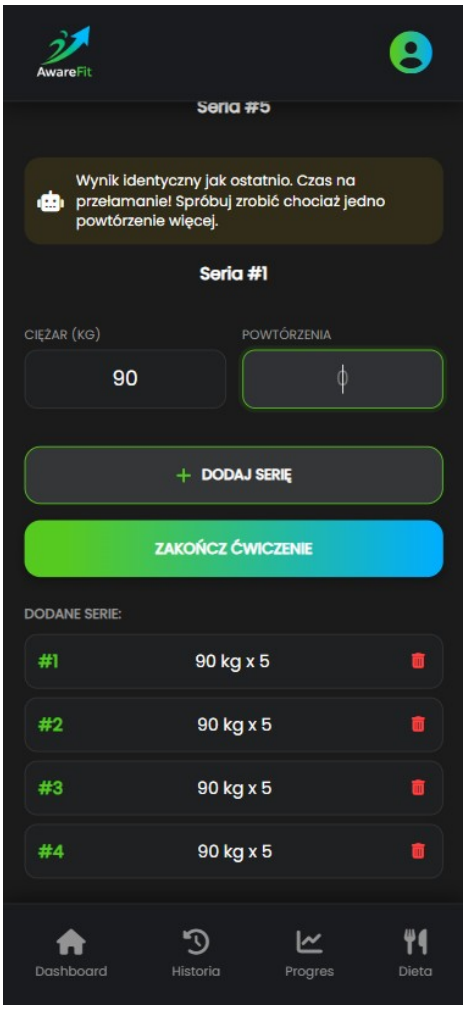
5. **Zakończenie i zapis:** System przelicza czas trwania sesji (`durationSec`) i wysyła kompletny obiekt `currentWorkout` do bazy danych (Rys. 5.8).



Rys. 5.5. Menu wyboru partii mięśniowej



Rys. 5.6. Dynamiczna lista ćwiczeń



Rys. 5.7. Ekran wprowadzania danych serii

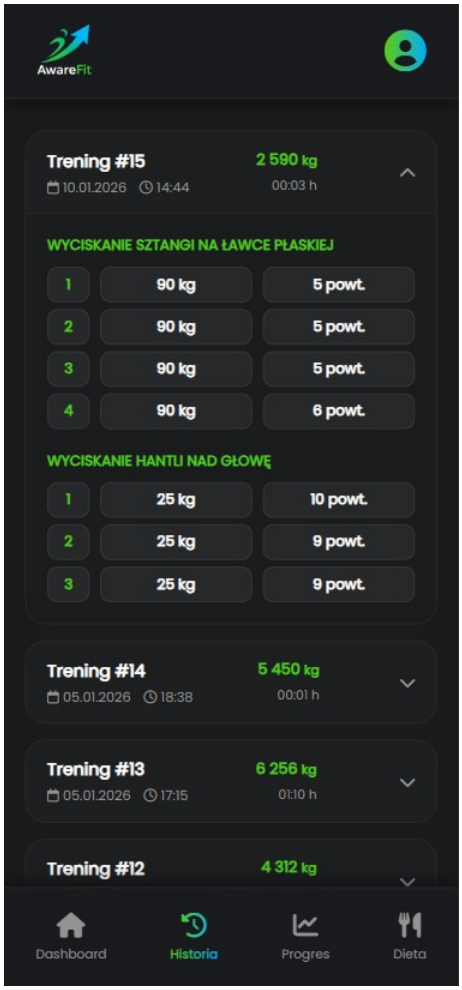
Rys. 5.8. Podsumowanie i finalizacja sesji

Zarządzanie stanem (JavaScript):

Kluczowym rozwiązaniem jest użycie `localStorage` z unikalnym kluczem `activeWorkout_${currentUserId}`. Dzięki temu, w przypadku awarii przeglądarki, dane treningu oraz czas rozpoczęcia sesji (`timerKey`) nie zostają utracone.

5.5. Widok historii treningów

Widok `history.php` służy do analizy aktywności użytkownika. Został on zaprojektowany w formie interaktywnej listy (akordeonu), która pozwala na szybki przegląd ogólnych statystyk sesji oraz ich szczegółowe rozwinięcie.



Rys. 5.9. Interfejs historii treningów z wykorzystaniem systemu akordeonów

Powiązanie z logiką bazy danych:
Skrypt `history.php` wykonuje złożone zapytanie do bazy danych, które łączy dane z funkcji tabelarycznej oraz funkcji skalarnej:

- **Agregacja sesji:** System wykorzystuje funkcję `public.get_user_workout_history`, która zwraca pełną historię ćwiczeń i serii przypisanych do danego `user_id`.
- **Dynamiczne obliczanie objętości:** Wewnątrz zapytania SQL wywoływana jest funkcja `public.calculate_workout_total_volume(workout_id)`. Pozwala to na wyświetlenie sumarycznego tonażu treningu (`db_total_volume`) bez konieczności kosztownych obliczeń po stronie PHP.
- **Mapowanie danych:** Ponieważ wynik zapytania jest „płaską” listą serii, skrypt PHP dokonuje transformacji danych do wielowymiarowej tablicy `$history`, grupując wpisy według identyfikatora treningu (`workout_id`) oraz nazw ćwiczeń (`exercise_name`).

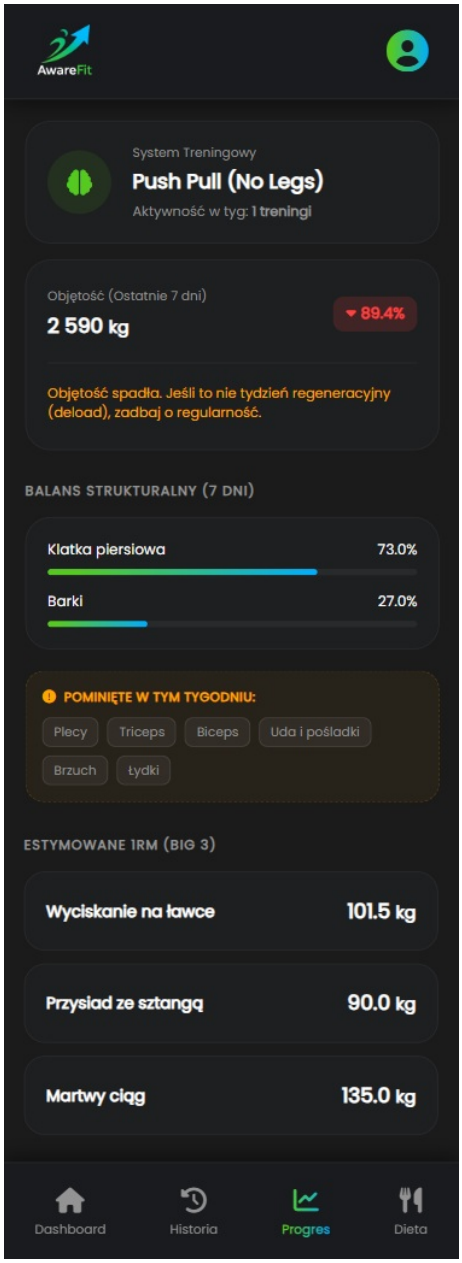
Funkcjonalność interfejsu:
Za warstwę prezentacji odpowiadają dedykowane style `history.css` oraz logika zawarta w `history.js`:

- **System Akordeonów:** Nagłówek każdego treningu wyświetla kluczowe metadane: numer treningu (`user_workout_no`), datę, czas trwania oraz łączną objętość. Dzięki funkcji `toggleAccordion`, użytkownik może dynamicznie rozwijać szczegóły konkretnej sesji.
- **Szczegółowy wykaz serii:** Po rozwinięciu sekcji, wyświetlana jest lista ćwiczeń wraz z tabelarycznym zestawieniem serii (numer serii, ciężar, liczba powtórzeń), co pozwala na precyzyjną analizę wykonanej pracy.
- **Wizualizacja tonażu:** Całkowita objętość sesji jest wyróżniona za pomocą klasy `volume-tag`, co ułatwia śledzenie progresji siłowej na przestrzeni czasu.

Obsługa skryptowa (`history.js`):
Interakcja z dokumentem realizowana jest przez prosty mechanizm manipulacji modelem DOM. Funkcja `toggleAccordion` zarządza stanem widoczności bloku `accordion-content` oraz animacją ikony nawigacyjnej (`chevron`), co znacząco poprawia czytelność interfejsu przy dużej liczbie zarejestrowanych treningów.

5.6. Widok analityczny postępów (Progress)

Moduł `progress.php` stanowi centrum analityczne aplikacji AwareFit. Jego zadaniem jest przetworzenie surowych danych treningowych na informacje o charakterze motywacyjnym i diagnostycznym. Widok ten w stopniu najwyższym wykorzystuje logikę zaszytą w warstwie bazy danych (PostgreSQL).



Rys. 5.10. Panel analityczny z zestawieniem objętości, balansu i rekordów

Integracja z zaawansowanymi funkcjami bazy danych:
W przeciwieństwie do prostych list, widok progresu opiera się na wynikach skomplikowanych obliczeń wykonywanych po stronie serwera bazy danych:

- **Detekcja systemu treningowego:** Wykorzystanie autorskiej funkcji `public.detect_training_split` pozwala systemowi automatycznie zdiagnozować, czy użytkownik trenuje systemem np. „Full Body Workout” czy „Split”, bazując na częstotliwości i doborze ćwiczeń.

- **Estymacja 1RM (One Rep Max):** Dla „Wielkiej Trójki" bojów siłowych aplikacja wywołuje funkcję `public.calculate_exercise_1rm`. Wynik obliczany jest na podstawie najlepszych serii użytkownika przy użyciu matematycznych wzorów progresji (np. Brzyckiego lub Epleya) zaimplementowanych w pgSQL.
- **Analiza porównawcza objętości:** Funkcja `public.get_volume_comparison` zwraca tonaż z obecnego i poprzedniego tygodnia, co pozwala skryptowi PHP wyliczyć procentowy trend (`diff_percent`) i dobrać odpowiedni komunikat motywacyjny (trend pozytywny, stagnacja lub regresja).
- **Balans strukturalny:** Dzięki funkcji `public.get_user_muscle_balance`, aplikacja generuje wykresy słupkowe pokazujące procentowe zaangażowanie poszczególnych partii mięśniowych w skali tygodnia.

Kluczowe elementy interfejsu:

1. **Karta Systemu i Streaku:** Wyświetla aktualną metodę treningową oraz liczbę jednostek treningowych w ostatnich 7 dniach, co pozwala monitorować systematyczność.
2. **Analizator Objętości:** Sekcja ta dynamicznie zmienia kolorystykę (`trend_class`) w zależności od wyników. Wykorzystuje ikony `fa-caret-up/down` do wizualizacji kierunku zmian tonażu.
3. **Wizualizacja Balansu:** Interaktywne paski postępu (`balance-bar-fill`) pokazują, które partie dominują w planie, a sekcja „Pominięte" (wyliczana przez `array_diff` w PHP) ostrzega o zaniedbanych grupach mięśniowych.
4. **Sekcja Rekordów:** Wyświetla estymowaną siłę maksymalną, co pozwala użytkownikowi śledzić progres bez konieczności wykonywania ryzykownych prób maksymalnych.

Logika prezentacji danych:

Aplikacja dba o czytelność poprzez zaokrąglanie wyników siłowych do jednego miejsca po przecinku oraz formatowanie dużych liczb objętości (np. `number_format` dla tonażu w kg). W przypadku braku danych za dany okres, system wyświetla komunikaty pomocnicze, zachęcając do rejestracji pierwszego treningu.

5.7. Widok diety i zarządzania pomiarami ciała

Moduł `diet.php` integruje dane antropometryczne użytkownika z automatycznym systemem wyliczania zapotrzebowania kalorycznego. Jest to kluczowy element personalizacji planu treningowego, pozwalający na dostosowanie diety do aktualnych celów sylwetkowych.

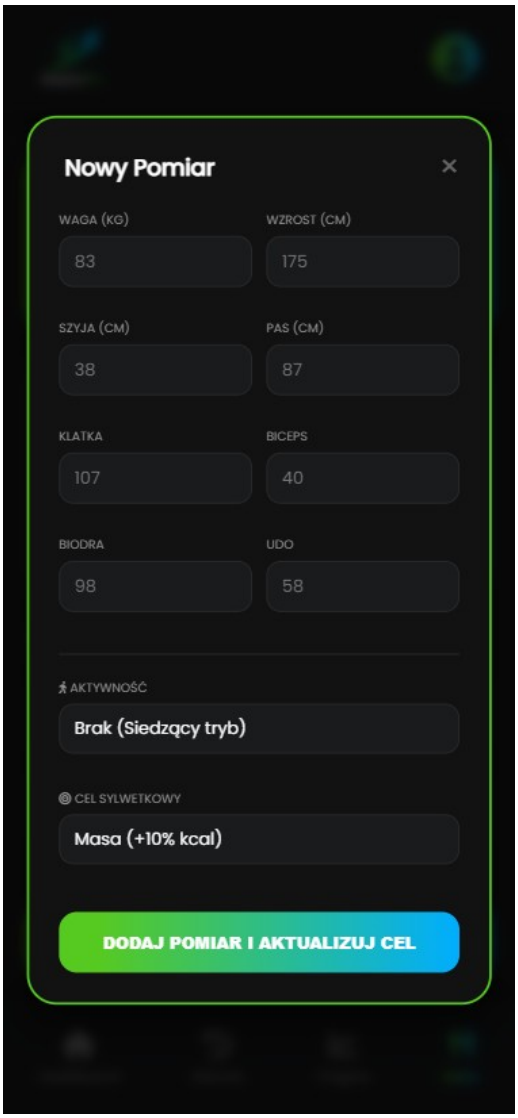
Rys. 5.11. Panel diety z wyliczonymi makroskładnikami i historią pomiarów

Zaawansowana analityka i funkcje SQL:
Strona ta w dużej mierze polega na logice obliczeniowej zaimplementowanej w PostgreSQL, co odciąża warstwę aplikacji:

- **Estymacja Body Fat:** Funkcja `public.calculate_user_bf` implementuje algorytm *US Navy Body Fat*, wykorzystując pomiary wzrostu, szyi, pasa i bioder.
- **Obliczanie zapotrzebowania (TDEE):** Skrypt wywołuje funkcję `public.calculate_user_diet_calories`, która na podstawie wzoru Mifflina-St Jeora oraz zadeklarowanego poziomu aktywności (`activity_level`), wylicza całkowite zapotrzebowanie kaloryczne.
- **Podział Makroskładników:** Funkcja `public.get_user_macros` dokonuje procentowego podziału kalorii na białka, tłuszcze i węglowodany, biorąc pod uwagę wybrany cel (masa, redukcja lub rekompozycja).

Zarządzanie danymi (Modal i Procedury):
Aplikacja umożliwia szybką aktualizację danych bez przeładowywania strony głównej dzięki zastosowaniu interaktywnego modala (`measurementModal`):

- **Zapisywanie danych:** Formularz przesyła dane do `add_measurement_action.php`, który zamiast prostego zapytania `INSERT`, wywołuje procedurę składowaną `crud.insert_body_measurement`. Gwarantuje to spójność danych i poprawność rzutowania typów (np. `double precision`, `numeric`).
- **Interaktywność (JS):** Skrypt `add_measurements.js` zarządza stanem modala oraz zapewnia odpowiednie *User Experience* poprzez wizualne potwierdzenie zapisu (animacja spinnera).



Rys. 5.12. Interfejs wprowadzania nowych pomiarów ciała i wyboru celu treningowego

Prezentacja wyników:
Interfejs użytkownika został podzielony na trzy sekcje priorytetowe:

1. **Highlight Card:** Wyświetla aktualny poziom tkanki tłuszczowej, co jest najważniejszym wskaźnikiem kompozycji ciała.

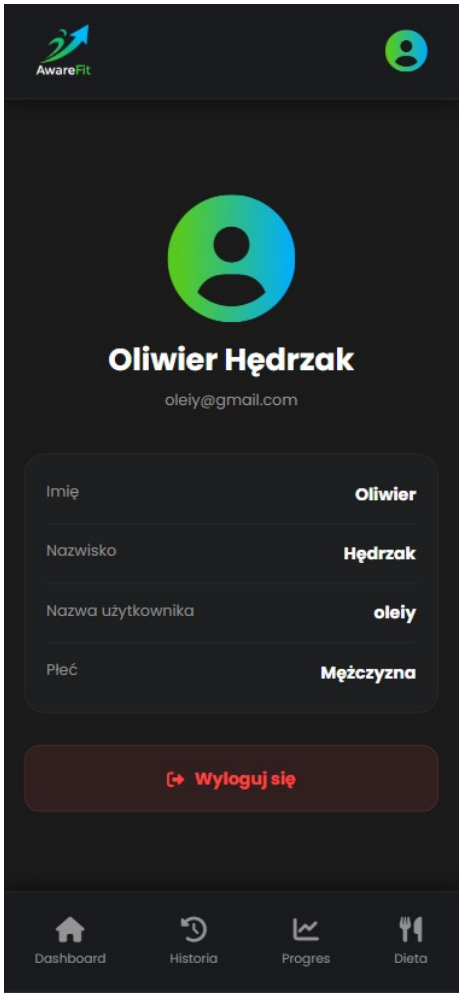
2. **Diet Summary:** Prezentuje docelową kaloryczność oraz gramaturę makroskładników w formie czytelnych kafelków.

3. **Measurements Grid:** Zestawienie wszystkich obwodów ciała w kompaktowej formie, pozwalające na szybką weryfikację ostatniej aktualizacji danych.

W przypadku braku danych w bazie, system dynamicznie wyświetla widok *empty state*, prowadząc użytkownika za rękę do pierwszego wpisu, co jest istotnym elementem *onboardingu* w aplikacji.

5.8. Widok profilu użytkownika

Widok `account.php` jest dedykowany zarządzaniu tożsamością użytkownika w aplikacji. Pozwala on na weryfikację danych konta oraz bezpieczne zakończenie sesji. Dostęp do tego panelu uzyskiwany jest poprzez interaktywną ikonę w nagłówku aplikacji (*header*).



Rys. 5.13. Podgląd profilu użytkownika z danymi osobowymi

Integracja z bazą danych:
Aplikacja wykorzystuje relacyjną strukturę tabeli `public.users` do dynamicznego generowania treści profilu:

- **Personalizacja nagłówka:** Skrypt PHP implementuje logikę priorytetyzacji wyświetlania nazw. Jeśli w bazie uzupełnione są pola `first_name` oraz `last_name`, system wyświetla pełne imię i nazwisko. W przeciwnym razie, jako identyfikator główny, prezentowany jest `username`.
- **Bezpieczeństwo sesji:** Każde żądanie do pliku `account.php` jest poprzedzone inkluzją `auth.php`, co gwarantuje, że dane z tabeli `users` są pobierane wyłącznie dla zweryfikowanego `user_id` zapisanego w sesji.
- **Lokalizacja danych:** System dokonuje mapowania wartości z bazy danych na język polski (np. konwersja wartości pola `gender` z formatu *Male/Female* na *Mężczyzna/Kobieta*).

Komponenty interfejsu:

1. **Sekcja identyfikacji (Avatar):** Centralny punkt widoku z ikoną systemową oraz adresem e-mail, co pozwala użytkownikowi szybko potwierdzić, na jakie konto jest zalogowany.
2. **Lista szczegółów (*Profile Details*):** Zestawienie kluczowych atrybutów konta w formie czytelnych par etykieta-wartość.
3. **Zarządzanie sesją:** Przycisk wylogowania (`logout-btn`) odsyła do `index.php`.

Warstwa wizualna:

Podobnie jak reszta aplikacji, widok profilu korzysta z arkusza `global.css` oraz dedykowanego pliku `account.css`. Zastosowano czcionkę *Poppins* oraz bibliotekę *Font Awesome*, aby zachować spójność wizualną z systemem Android/iOS, co nadaje aplikacji charakter natywnego narzędzia mobilnego.

6. Dostęp zdalny do bazy danych

W niniejszym rozdziale przedstawiono koncepcję oraz techniczną realizację komunikacji pomiędzy aplikacją AwareFit a systemem zarządzania bazą danych PostgreSQL.

6.1. Koncepcja dostępu zdalnego

Model architektury systemowej:
Aplikacja AwareFit została zaprojektowana jako progresywna platforma webowa działająca w modelu *Client-Server*. Koncepcja dostępu zdalnego opiera się na separacji interfejsu użytkownika od fizycznej lokalizacji danych.

- **Dostęp przez protokół HTTP/HTTPS:** Użytkownik uzyskuje dostęp do aplikacji poprzez przeglądarkę internetową. Eliminuje to konieczność instalacji bazy danych na urządzeniu końcowym – wszystkie operacje są procesowane zdalnie na serwerze.
- **Rola serwera pośredniczącego:** Serwer Apache (XAMPP) pełni rolę bramy (Gateway). Przyjmuje on żądania od użytkownika, a następnie nawiązuje zdalne połączenie z silnikiem PostgreSQL, aby pobrać lub zapisać dane.
- **Niezależność lokalizacji:** Dzięki zastosowaniu standardu TCP/IP oraz sterowników PDO, system jest przygotowany do pracy w rozproszonym środowisku sieciowym. Baza danych może znajdować się na dedykowanym hostingu, podczas gdy pliki strony serwowane są z innej lokalizacji.

Zalety koncepcji webowej:
Dzięki takiemu podejściu, dostęp zdalny zapewnia:

- **Mobilność:** Użytkownik może edytować swój trening na telefonie w siłowni, a następnie analizować postępy na komputerze domowym – dane są zawsze zsynchronizowane w centralnej bazie.
- **Bezpieczeństwo:** Kluczowe dane (hasła, wyniki) nigdy nie są przechowywane bezpośrednio w przeglądarce, a jedynie przesyłane bezpiecznym kanałem do bazy danych.

6.2. Opis realizacji dostępu zdalnego

Realizacja połączenia opiera się na rozszerzeniu `PDO` (PHP Data Objects). Środowisko uruchomieniowe zostało oparte na pakiecie **XAMPP** (serwer Apache) oraz systemie **PostgreSQL** zarządzanym przez narzędzie **pgAdmin 4**.

Implementacja techniczna połączenia:
Kluczowym elementem jest plik `includes/db.php`, który inicjuje sesję komunikacyjną. Poniżej przedstawiono kod źródłowy realizujący to zadanie:

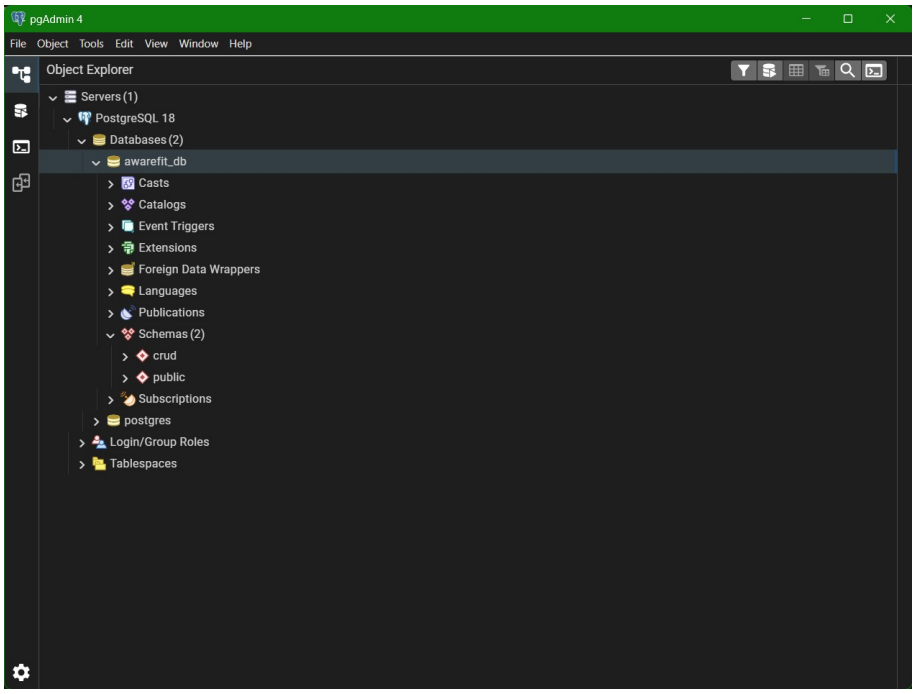
```
<?php
$host = "localhost";
$port = "5432";
$dbname = "awarefit_db";
$user = "postgres";
$password = "psql";

try {
    $dsn = "pgsql:host=$host;port=$port;dbname=$dbname";
    $pdo = new PDO($dsn, $user, $password, [
        PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION
    ]);
} catch (PDOException $e) {
    die("Błąd połączenia: " . $e->getMessage());
}
?>
```

Autoryzacja dostępu w warstwie aplikacji:
Aby zapewnić, że zdalny dostęp do danych jest bezpieczny, każdy skrypt wymagający połączenia z bazą chroniony jest przez mechanizm sesji zawarty w pliku `auth.php`:

```
<?php
session_start();
if (!isset($_SESSION['user\_id'])) {
    header("Location: index.php");
    exit();
}
?>
```

Prezentacja i monitoring (pgAdmin 4):
Do monitorowania stanu połączeń oraz weryfikacji poprawności zapisu danych wykorzystywane jest narzędzie **pgAdmin 4**. Pozwala ono na podgląd aktywnych procesów serwera oraz zarządzanie strukturą tabel (Rys. 6.1).



Rys. 6.1. Struktura bazy danych awarefit_db widoczna w narzędziu pgAdmin 4

Weryfikacja działania:

Poprawność realizacji dostępu zdalnego została potwierdzona poprzez:

1. **Testy połączenia:** Weryfikacja reakcji aplikacji na błędne dane logowania (poprawne przechwycenie wyjątku `PDOException`).
2. **Integracja z Apache:** Poprawne procesowanie żądań przez serwer Apache wewnątrz środowiska XAMPP i stabilne utrzymywanie połączenia z procesem `postgres.exe`.
3. **Zdalne zarządzanie:** Możliwość edycji i przeglądu danych w czasie rzeczywistym bezpośrednio w interfejsie graficznym pgAdmin 4.

7. Podsumowanie techniczne projektu AwareFit

Niniejszy rozdział stanowi syntetyczne zestawienie informacji o architekturze, technologiach oraz przeznaczeniu aplikacji AwareFit, bazujące na zaimplementowanych rozwiązaniach programistycznych.

7.1. Przeznaczenie i cele aplikacji

Głównym celem aplikacji AwareFit jest dostarczenie użytkownikowi kompleksowego narzędzia do monitorowania progresji siłowej oraz zarządzania dietą. System został zaprojektowany z myślą o:

- **Efektywnym zapisywaniu jednostek treningowych:** Intuicyjny system rejestrowania sesji treningowych, pozwalający na precyzyjne dokumentowanie każdego ćwiczenia, serii, obciążenia oraz liczby powtórzeń, co stanowi fundament rzetelnej analizy postępów.
- **Automatyzacji obliczeń parametrów treningowych:** Wykorzystanie mechanizmów bazodanowych do bieżącego wyliczania tonażu treningowego oraz ciągłości aktywności użytkownika.
- **Wizualizacji progresu:** Dynamiczne generowanie zestawień graficznych obrazujących wzrost siły i objętości w czasie.
- **Monitorowaniu kompozycji ciała:** Automatyczne dostosowywanie wyliczeń makroskładników i zapotrzebowania kalorycznego na podstawie wprowadzanych pomiarów antropometrycznych.

7.2. Struktura bazy danych i logiki serwerowej

Architektura bazy danych PostgreSQL opiera się na dwóch głównych schematach, co zapewnia separację logiki operacyjnej od fizycznej struktury przechowywania danych:

- **Schemat (`public`):** Przechowuje strukturę tabel bazy danych oraz zaawansowane algorytmy.
- **Schemat (`crud`):** Zawiera wszystkie procedury oraz funkcje CRUD.

7.3. Interfejs użytkownika (GUI) i warstwa frontendowa

Warstwa wizualna została zbudowana w oparciu o podejście *Mobile-First* oraz nowoczesne standardy webowe, zapewniając pełną responsywność na urządzeniach mobilnych:

- **Technologie:** HTML5, CSS3 (z wykorzystaniem zmiennych CSS dla ujednoliconego motywu graficznego) oraz nowoczesny JavaScript (ES6+).
- **Wizualizacja danych:** Wykorzystanie profesjonalnych bibliotek do tworzenia interaktywnych wykresów liniowych, które pozwalają użytkownikowi na intuicyjną analizę historii treningowej.
- **Interaktywność:** Zastosowanie systemowych okien modalnych do prezentacji danych szczegółowych oraz mechanizmów pamięci lokalnej przeglądarki, które umożliwiają kontynuację sesji treningowej nawet po przypadkowym zamknięciu aplikacji.
- **Estetyka:** Wykorzystanie nowoczesnej typografii oraz płynnych animacji interfejsu, które podnoszą komfort użytkowania i czytelność prezentowanych statystyk.

7.4. Integracja technologii (Tech Stack)

System stanowi spójne połączenie warstwy prezentacji, logiki biznesowej oraz trwałego składowania danych:

- **Bezpieczeństwo i komunikacja:** Wykorzystanie języka PHP jako pośrednika pomiędzy interfejsem a bazą danych. Zastosowano bezpieczne połączenia poprzez sterowniki bazodanowe.
- **Asynchroniczność:** Skrypty klienckie dynamicznie przetwarzają dane, co pozwala na aktualizację kluczowych elementów panelu sterowania (*Dashboard*) bez konieczności przeładowywania całej strony.
- **Środowisko uruchomieniowe:** Serwer Apache oraz silnik PostgreSQL, zintegrowane w ramach lokalnego środowiska deweloperskiego, zapewniające wysoką stabilność podczas prac nad projektem.

7.5. Dostępność projektu i kontrola wersji

Projekt AwareFit jest rozwijany w oparciu o nowoczesne standardy zarządzania kodem źródłowym, co zapewnia bezpieczeństwo danych oraz pełną przejrzystość historii zmian:

- **Repozytorium zdalne:** Pełny kod źródłowy aplikacji, wraz z dokumentacją bazy danych oraz plikami konfiguracyjnymi środowiska, jest dostępny pod adresem: *https://github.com/MichalJanikUR/ProjektBazyDanych.git*.
- **System kontroli wersji:** Wykorzystanie systemu Git umożliwiło precyzyjne śledzenie postępów prac, zarządzanie równoległymi zmianami w kodzie oraz bezpieczne wdrażanie poprawek bez ryzyka utraty stabilności głównej wersji systemu.
- **Struktura plików i dokumentacja:** Kod w repozytorium został zorganizowany w sposób modularny, z wyraźnym podziałem na warstwy aplikacji (logika PHP, interfejs użytkownika, skrypty bazy danych). Pozwala to na łatwą weryfikację implementacji oraz ewentualną rozbudowę systemu w przyszłości.

Bibliografia

[1] Chart.js Contributors. Chart.js - open source html5 charts for wordpress, drupal, react and more, 2025. Biblioteka wykorzystana do generowania interaktywnych wykresów progresji treningowej.

[2] Google Fonts. Poppins font family, 2026. Źródło rodziny czcionek wykorzystanej w typografii systemu.

[3] Freepik. Zestaw ikon oraz grafik wykorzystanych w interfejsie aplikacji awarefit, 2025. Zasoby poddane edycji w programie GIMP w celu zapewnienia spójności wizualnej.

[4] PostgreSQL Global Development Group. Postgresql 16 documentation, 2024.

[5] M. Hędrzak, O. oraz Janik. Projektbazydanych - dokumentacja techniczna i kod źródłowy aplikacji awarefit, 2026. Główne repozytorium projektu w serwisie GitHub.

Spis rysunków

Załącznik nr 2 do Zarządzenia nr 228/2021 Rektora Uniwersytetu Rzeszowskiego z dnia 1 grudnia 2021 roku w sprawie ustalenia procedury antyplagiatowej w Uniwersytecie Rzeszowskim

OŚWIADCZENIE STUDENTA O SAMODZIELNOŚCI PRACY

……Oliwier Hędrzak, Michał Janik……
Imię (imiona) i nazwisko studenta

Wydział Nauk Ścisłych i Technicznych

……Informatyka……
Nazwa kierunku

……134913, 134915……
Numer albumu

1. Oświadczam, że moja praca projektowa pt.: Dziennik treningowy AwareFit – dokumentacja projektowa
   1) została przygotowana przeze mnie samodzielnie*,
   2) nie narusza praw autorskich w rozumieniu ustawy z dnia 4 lutego 1994 roku o prawie autorskim i prawach pokrewnych (t.j. Dz.U. z 2021 r., poz. 1062) oraz dóbr osobistych chronionych prawem cywilnym,
   3) nie zawiera danych i informacji, które uzyskałem/am w sposób niedozwolony,
   4) nie była podstawą otrzymania oceny z innego przedmiotu na uczelni wyższej ani mnie, ani innej osobie.
2. Jednocześnie wyrażam zgodę/nie wyrażam zgody** na udostępnienie mojej pracy projektowej do celów naukowo–badawczych z poszanowaniem przepisów ustawy o prawie autorskim i prawach pokrewnych.

(miejscowość, data) (czytelny podpis studenta)

* Uwzględniając merytoryczny wkład prowadzącego przedmiot

** – niepotrzebne skreślić