Kierunek: ***Informatyka***

2025/2026

Mikołaj Niewola

PRACA INŻYNIERSKA

Projekt i implementacja aplikacji mobilnej wspomagającej planowanie treningu siłowego

Promotor pracy:

mgr inż. Tomasz Gądek

Tarnów, *2026*

Spis treści

1. Wstęp

We współczesnym społeczeństwie ludzie przykładają coraz większą uwagę do zdrowego trybu życia, aktywności fizycznej oraz dbania o kondycję. Wzrost świadomości zdrowotnej oraz popularyzacja mediów społecznościowych, w których coraz częściej promowany jest sportowy styl życia, sprawiają, że coraz więcej osób decyduje się na wprowadzenie aktywności fizycznej do swojej codzienności. Na zdrowy tryb życia składa się wiele czynników takich jak: zbilansowana dieta, odpowiednia ilość snu oraz ograniczanie stresu, jednak najważniejszą z nich jest aktywność fizyczna. Jedną z najpopularniejszych form aktywności fizycznej w ostatnich latach jest uczęszczanie na siłownię.

Wiele siłowni oferuje szeroki zakres możliwości treningowych, pozwalając na rozwój całego ciała oraz poprawę samopoczucia. Jednak dla wielu początkujących, rozpoczęcie treningu może być dużym wyzwaniem. Brak wiedzy na temat prawidłowego doboru ćwiczeń, techniki ich wykonania oraz zasad tworzenia zbilansowanego planu może prowadzić do braku postępów, przeciążeń, a nawet kontuzji. W rezultacie wiele osób rezygnuje z dalszych ćwiczeń już na początkowym etapie swojej przygody z treningiem na siłowni.

W ostatnich latach dużą rolę w kształtowaniu nawyków zaczęły odgrywać aplikacje mobilne. Powszechny dostęp do smartfonów i Internetu sprawił że użytkownicy coraz częściej korzystają z aplikacji do monitorowania zdrowia. Od aplikacji liczących kalorie i makroskładniki, po aplikacje do monitorowania postępów czy planowania treningów. Chociaż na rynku jest wiele aplikacji związanych z aktywnością fizyczną, zdecydowana większość z nich jest skierowana do osób posiadających już pewną wiedzę na temat treningów. Takie aplikacje często opisują ćwiczenia w sposób zbyt ogólny co może być skomplikowane dla początkujących.

Projektowana aplikacja ma na celu wypełnienie tej luki, oferując rozwiązanie, które pomoże osobom początkującym w tworzeniu swojego własnego, zbilansowanego planu treningowego. Kluczowym elementem będzie interaktywna mapa mięśni, pozwalająca użytkownikowi na wizualne sprawdzenie wpływu ćwiczeń na jego mięśnie. Dodatkowo aplikacja zostanie wyposażona w bazę filmów instruktażowych prezentujących użytkownikowi poprawną technikę wykonania ćwiczeń co zminimalizuje ryzyko błędów i kontuzji.

1.1. Cel pracy

Celem niniejszej pracy będzie zaprojektowanie i implementacja aplikacji mobilnej, która posłuży jako pomoc w ułożeniu indywidualnego planu treningowego. Aplikacja ma na celu umożliwić użytkownikowi w intuicyjny i przejrzysty sposób dobrać ćwiczenia do poszczególnych partii mięśniowych, a także zaprezentować ich prawidłowe wykonanie. Dzięki zastosowaniu interaktywnej mapy mięśni oraz bazy z filmami, użytkownik będzie mógł świadomie budować swój indywidualny plan treningowy, minimalizując ryzyko kontuzji i zwiększając efektywność i przyjemność z treningów.

1.2. Zakres pracy

Niniejsza praca inżynierska obejmuje:

- aplikacje mobilną,
- bazę danych,
- warstwę serwerową.

Zakres niniejszej pracy inżynierskiej obejmuje zaprojektowanie i implementację mobilnej aplikacji wspierającej użytkowników w tworzeniu indywidualnych planów treningowych. W ramach projektu zostaną opracowane:

- Interfejs użytkownika umożliwiający intuicyjny dobór ćwiczeń,
- Interaktywna mapa mięśni prezentująca zaangażowane partie ciała,
- System uzupełniający plan na podstawie dodanych ćwiczeń,
- Baza danych zawierająca ćwiczenia oraz filmy instruktażowe,
- Moduł tworzenia i zapisywania planów treningowych oraz śledzenia postępów,
- Prosty backend do obsługi zapytań.

2. Analiza biznesowa

W tym rozdziale zostały przedstawione główne funkcjonalności aplikacji. Analiza ma na celu przedstawić oczekiwania wobec działania systemu oraz wymagania funkcjonalne i niefunkcjonalne.

2.1. Analiza rynku

Na rynku jest obecnie dostępnych wiele aplikacji o podobnej tematyce i funkcjonalności. Zdecydowana większość z nich jest jednak skierowana do użytkowników, którzy posiadają już wiedzę lub doświadczenie na ten temat. Dla początkujących takie rozwiązania bywają często mało intuicyjne lub są wyjaśnione zbyt ogólnie.

Projektowana aplikacja będzie miała na celu pomóc początkującym w prawidłowym wykonaniu ćwiczeń za pomocą zintegrowanych filmików instruktażowych. Dodatkowo mapa mięśni oraz inteligentne polecanie ćwiczeń pomoże użytkownikom w tworzeniu swojego własnego zestawu ćwiczeń, gdzie w innych aplikacjach najczęstszym rozwiązaniem są predefiniowane plany treningowe.

Pomoc w planowaniu treningu oraz funkcje umożliwiające monitorowanie postępu pomogą aplikacji dotrzeć do szerszego grona odbiorców, zarówno początkujących, jak i bardziej doświadczonych użytkowników.

2.2. Wymagania funkcjonalne

Aplikacja ma na celu pomóc użytkownikowi w zaplanowaniu treningu na siłownię. Konieczne będzie zapewnienie funkcjonalności, które pomogą w złożeniu zbilansowanego i bezpiecznego planu osobom, które nie mają wiedzy na ten temat. Funkcjonalności powinny być intuicyjne i łatwe w obsłudze.

- **Interaktywna mapa mięśni człowieka**

Główną funkcją aplikacji będzie mapa mięśni człowieka, która podświetla partie mięśniowe, na które wykonujemy ćwiczenia w danym planie treningowym.

Gdy użytkownik doda dużo ćwiczeń na jedną partię, zostanie ona podświetlona kolorem czerwonym, aby poinformować go o możliwym przeciążeniu mięśniowym, które może z tego wynikać. Rozwiązanie to ma być prostym i szybkim sposobem przekazania użytkownikowi informacji na temat ilości ćwiczeń i stanie przetrenowania mięśni.

- **Baza ćwiczeń z filmami**

Drugą istotną funkcjonalnością aplikacji będzie baza ćwiczeń. Baza będzie zawierać informacje na temat partii mięśniowej, na którą wpływa dane ćwiczenie oraz filmik, który użytkownik może oglądnąć by dowiedzieć się jak prawidłowo powinno zostać wykonane.

- **Tworzenie planu treningowego**

Użytkownik będzie mieć możliwość stworzenia swojego własnego planu treningowego. Dzięki bazie ćwiczeń oraz mapie mięśni będzie on w stanie przygotować trening bez potrzeby wiedzy na temat ćwiczeń, oraz proporcji, w jakiej powinny zostać wykonywane na poszczególne partie mięśniowe.

- **Automatyczne wskazówki oraz pomoc w doborze ćwiczeń**

Gdy użytkownik przygotuje swój plan, aplikacja przeanalizuje go i zaproponuje ćwiczenia uzupełniające w celu zbalansowania obciążenia poszczególnych partii mięśniowych. (np. mała liczba ćwiczeń na nogi sugeruje dodanie przysiadów).

2.3. Wymagania niefunkcjonalne

Wymagania niefunkcjonalne to kryteria, które określają właściwości aplikacji, które mają kluczowe znaczenia dla jakości, komfortu i bezpieczeństwa jej użytkowania. Zaprojektowanie i spełnienie wymagań niefunkcjonalnych jest konieczne, aby aplikacja była stabilna, intuicyjna i przyjazna dla użytkownika. Dodatkowo musi ona spełniać standardy techniczne obowiązujące we współczesnych czasach.

- **Wydajność**

Jednym z najważniejszych aspektów aplikacji jest jej wydajność. Aplikacja powinna działać płynnie i bez zauważalnych opóźnień. Dlatego powinna uruchamiać się w czasie nie

dłuższym niż 3 sekundy, aby zapewnić szybki dostęp do jej funkcjonalności. Akcje wykonane przez użytkownika, takie jak przełączanie ekranów, obsługa mapy mięśniowej, czy dodawanie nowych ćwiczeń, powinny być przetwarzane w czasie nie dłuższym niż sekunda [6].

- **Bezpieczeństwo**

Kolejnym aspektem są wymagania dotyczące bezpieczeństwa. Aplikacja powinna łączyć się z bazą danych za pomocą protokołu HTTPS, który gwarantuje szyfrowane połączenie i chroni przesyłane informacje przed nieupoważnionym dostępem.

- **Użyteczność**

Ważnym wymaganiem jest również użyteczność, która wpływa na odbiór aplikacji przez użytkowników. Interfejs graficzny powinien być intuicyjny, czytelny oraz umożliwiać obsługę aplikacji bez potrzeby korzystania z instrukcji. Kluczową częścią aplikacji jest mapa mięśniowa, która powinna w prosty i szybki sposób dostarczać informację na temat wpływu ćwiczeń na poszczególne partie mięśniowe.

- **Dostępność aplikacji**

Ostatnim, ale istotnym elementem jest dostępność aplikacji. System powinien działać przez 90% czasu w skali roku [12]. Dodatkowo aplikacja powinna umożliwiać dostęp do zapisanych planów treningowych bez dostępu do Internetu. Ta funkcjonalność będzie bardzo ważna dla użytkowników trenujących w miejscach bez zasięgu sieci. Dodatkowo w ramach możliwości zachowania swoich planów i rekordów, aplikacja powinna umożliwiać przenoszenie danych na inne urządzenie, co pozwoli użytkownikowi kontynuować używanie aplikacji bez utraty wcześniejszych postępów.

3. Opis wykorzystanych technologii

W tym rozdziale zostanie omówiony stos technologiczny, który zostanie wykorzystany do implementacji nowoczesnej aplikacji treningowej dla użytkownika.

3.1. JavaScript

Stworzony w 1995 roku przez Brendana Eicha w firmie Netscape Communications. Jako prosty język skryptowy mający na celu wzbogacić strony internetowe o elementy interaktywne. Wraz z pojawieniem się standardu ECMAScript, JavaScript przeszedł wiele zmian, stając się pełnoprawnym językiem programowania [1].

Język ten jest interpretowany i dynamicznie typowany, co oznacza, że nie wymaga wcześniejszej kompilacji kodu, a jego struktura jest elastyczna i przystępna [2]. Dzięki temu JavaScript stał się podstawowym narzędziem w tworzeniu interaktywnych aplikacji webowych, mobilnych i hybrydowych. Obecnie znajduje on zastosowanie w licznych środowiskach, takich jak Node.js (wykorzystanie serwerowe), oraz w nowoczesnych bibliotekach i frameworkach frontendowych takich jak: React, Angular lub Vue.js [3].

Jedną z głównych zalet JavaScriptu jest jego uniwersalność oraz ogromna społeczność, która nieustannie rozwija i wspiera język. Liczba dostępnych bibliotek oraz narzędzi sprawia, że jest on jednym z najbardziej przyjaznych języków dla początkujących. Dodatkowo, dzięki rozwojowi technologii mobilnych, JavaScript stał się fundamentem rozwiązań wieloplatformowych, umożliwiających tworzenie kodu zarówno na systemie Android, jak i iOS przy użyciu jednego wspólnego kodu.

Według raportu Stack Overflow z 2024 roku JavaScript jest jednym z najczęściej używanych języków programowania na świecie, co potwierdza jego dominującą pozycję i wszechstronność zastosowań [10].

3.2. React Native

React Native to kolejny fundament stosu technologicznego. Framework stworzony przez firmę Facebook (obecnie Meta) w 2015 roku. React Native umożliwia tworzenia natywnych aplikacji mobilnych przy użyciu języka JavaScript i biblioteki React. Głównym założeniem tej technologii jest możliwość wykorzystania jednego wspólnego kodu do napisania aplikacji działającej na wielu platformach, przy jednoczesnym zachowaniu natywnego wyglądu i wydajności [4].

React Native opiera się na komponentach, które reprezentują poszczególne elementy interfejsu użytkownika. Każdy komponent może być ponownie wykorzystany, co znacznie skraca czas tworzenia i testowania aplikacji. Framework łączy w sobie zalety rozwoju webowego (szybkość i elastyczność) z natywnym działaniem aplikacji mobilnych, dzięki czemu zyskał dużą popularność wśród firm i programistów na całym świecie.

Jedną z największych zalet React Native jest jego wydajność. Komponenty w React Native są renderowane przy użyciu natywnych elementów systemu operacyjnego, co pozwala osiągnąć bardzo dobrą płynność działania [8].

3.3. Express.js

Express.js to lekki i elastyczny framework przygotowany dla środowiska Node.js. Został opracowany w 2010 roku i służy do tworzenia aplikacji webowych oraz interfejsów API. W 2014 roku prawa do projektu zostały przejęte przez firmę StrongLoop, firma ta natomiast została wykupiona przez firmę IBM we wrześniu 2015 roku. W 2016 firma IBM ogłosiła, że projekt trafi pod opiekę inkubatora Node.js Foundation.

Express jest jednym z najpopularniejszych narzędzi do budowy warstwy serwerowej w aplikacjach opartych na języku JavaScript. Wykorzystuje się go w wielu stosach deweloperskich takich jak: MEAN, MERN lub MEVN razem z MongoDB oraz frontendowym frameworkiem korzystającym z JavaScript. Cały framework jest relatywnie niewielki i większość dostępnych funkcjonalności jest dostępna jako pluginy.

Express ułatwia proces obsługi żądań HTTP oraz routingu, co pozwala na tworzenie przejrzystych i skalowalnych aplikacji backendowych. Dzięki dużej ilości pluginów i braku narzuconej struktury łatwo jest dopasować aplikację do potrzeb systemu.

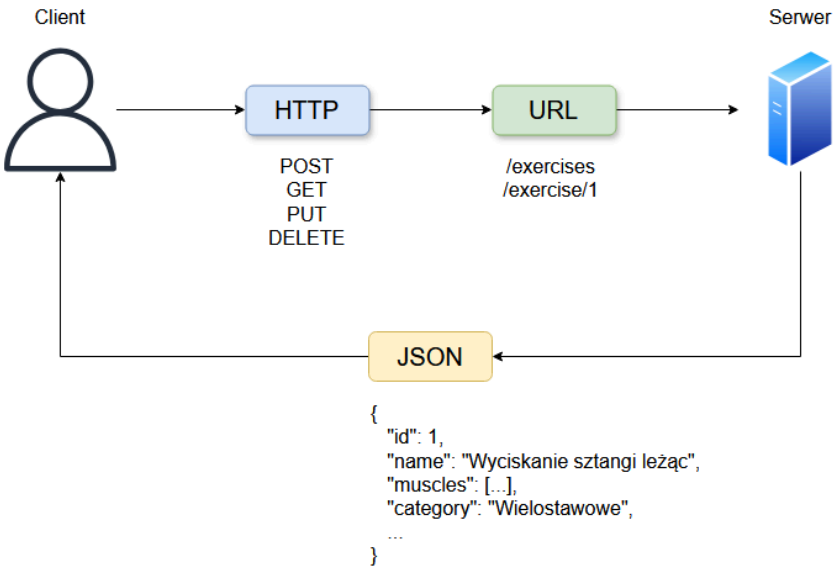
3.4. Protokół HTTP

HTTP (ang. *Hypertext Transfer Protocol*) to podstawowy protokół sieciowy wykorzystywany do komunikacji między klientem a serwerem w sieci internetowej. Został opracowany na początku lat 90. Przez Timothy'ego Bernersa-Lee w ramach projektu World Wide Web i do dziś stanowi fundament funkcjonowania stron internetowych oraz aplikacji webowych.

3.5. REST API

REST API (Representational State Transfer Application Programming Interface) jest standardem komunikacji między systemami informatycznymi, który opiera się na protokole HTTP. Architektura REST została opracowana przez Roya Fieldinga w 2000 roku jako sposób tworzenia niezależnych od platformy usług sieciowych. REST API umożliwia wymianę danych pomiędzy różnymi elementami systemu, takimi jak frontend (aplikacja) i backend (serwer lub baza danych) w prosty i szybki sposób.

Wymiana danych odbywa się przy użyciu metod HTTP, takich jak POST, GET, PUT czy DELETE, które odpowiadają kolejno za tworzenie, pobieranie, aktualizację oraz usuwanie danych. Dane przesyłane przez API są zazwyczaj w formacie JSON, co sprawia, że są czytelne w łatwy sposób dla aplikacji oraz człowieka. Zaletą REST API jest jego uniwersalność, dzięki standaryzowanej strukturze zapytań i odpowiedzi możliwa jest integracja z wieloma różnymi systemami bez konieczności dostosowywania kodu do konkretnej platformy. Przykład komunikacji klienta z serwerem za pomocą REST API został przedstawiony na rysunku 3.1.



Rys. 3.1 *REST API [opracowanie własne]*

3.6. JSON

JSON (JavaScript Object Notation) to prosty format wymiany danych oparty na strukturze obiektów z języka JavaScript. Został opracowany przez Douglasa Crockforda na początku lat 2000 i szybko zyskał popularność jako prosty i uniwersalny sposób reprezentacji danych. JSON jest łatwy do odczytania zarówno przez człowieka, jak i maszynę, a jego struktura opiera się na parach: klucz - wartość, listach i obiektach zagnieżdżonych [7]. Największą zaletą formatu JSON jest jego uniwersalność, można go wykorzystać w niemal każdym języku programowania i systemie. W porównaniu do starszego formatu XML, JSON jest bardziej zwięzły, prostszy w składni oraz wydajniejszy.

W projektowanej aplikacji JSON posłuży do przechowywania lokalnych danych takich jak zapisane plany treningowe. Do tego umieszczony na serwerze plik JSON posłuży jako prosta wersja bazy danych, która będzie zawierać wszystkie informacje na temat ćwiczeń.

3.7. Biblioteki React Native

3.7.1. **Axios**

Biblioteka Axios służy do komunikacji między zewnętrznym API poprzez protokół HTTP. Umożliwia wysyłanie zapytań do serwera i odbieranie odpowiedzi w formacie JSON. W porównaniu ze wbudowaną funkcją fetch(), Axios jest bardziej rozbudowany. Umożliwia on łatwiejsze zarządzanie błędami oraz konfigurację zapytań.

3.7.2. **Zustand**

Zustand to lekka biblioteka służąca do zarządzania stanem aplikacji. Pozwala na przechowywanie danych w jednym miejscu oraz ich udostępnianie między komponentami bez konieczności przekazywania ich jako propsów.

3.7.3. **React Navigation**

Jest to biblioteka używana przez React Native, która służy do nawigacji między ekranami. Jedna z najpopularniejszych bibliotek do tego typu zadań. Umożliwia tworzenie różnych rodzajów nawigacji i pozwala w prosty sposób kontrolować przepływ użytkownika w aplikacji [9].

4. Implementacja systemu

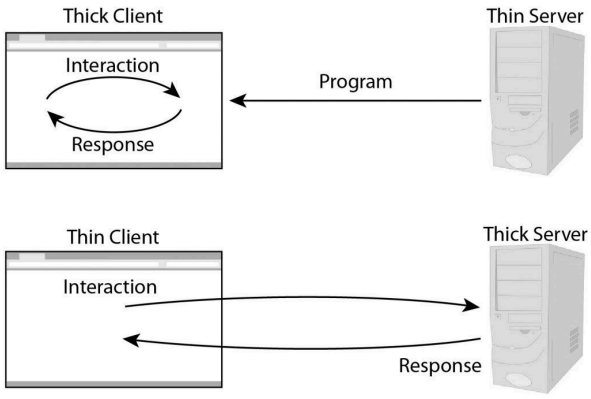
W tej części pracy zostanie przedstawiony proces implementacji zaprojektowanej aplikacji mobilnej. Celem tego rozdziału będzie przedstawienie sposobu implementacji głównych funkcji systemu. Do tego za pomocą diagramów zostaną przedstawione: Baza danych przy użyciu diagramu ERD oraz interakcje użytkownika z systemem za pomocą diagramu przypadków użycia. Diagramy te pozwolą na lepsze zrozumienie systemu oraz zależności między jego komponentami. Dodatkowo przedstawiona zostanie specyfikacja API, która zaprezentuje wszystkie endpointy potrzebne do komunikacji między aplikacją a bazą danych.

4.1. Architektura systemu

Architektura aplikacji opiera się na dwuwarstwowym modelu klient-serwer, a dokładniej Thick-Client. Największą zaletą tej architektury jest możliwość napisania aplikacji działającej w trybie offline.

Aplikacja mobilna pełni rolę klienta, odpowiedzialnego za prezentację interfejsu użytkownika, jak i za realizację logiki systemu. Druga warstwa danych, w której przechowywane są zasoby aplikacji - w tym przypadku plik JSON umieszczono na zewnętrznym serwerze.

Serwer pełni wyłącznie funkcję hostingu plików i przetwarza zapytania z prośbą o dostęp. Komunikacja między warstwami odbywa się za pomocą protokołu HTTP, a dane są w formacie JSON.



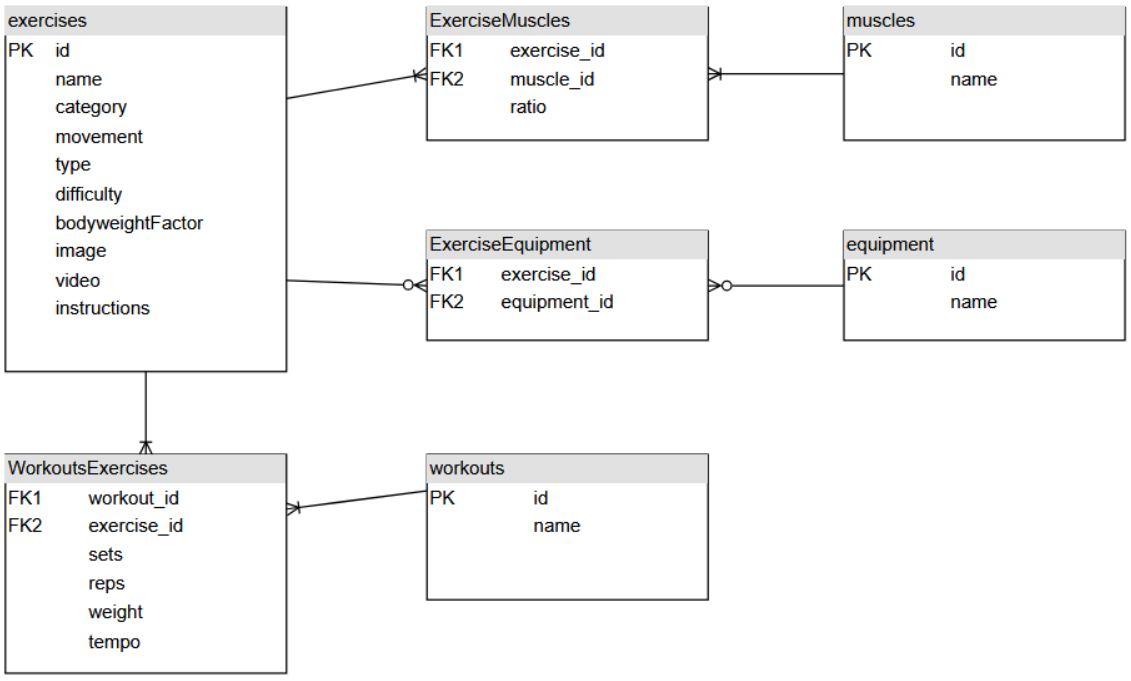
Rys. 4.1 *Architektura Thick-Client [11]*

4.2. Diagram ERD

W projekcie została wykorzystana prosta baza danych w formacie JSON, w której przechowywane są informacje o ćwiczeniach. Ten sposób działania będzie dobry dla aplikacji, która ma działać w trybie offline. Struktura bazy została przedstawiona za pomocą diagramu ERD na rysunku 4.2.

Tabela exercises przechowuje szczegółowe informacje o wszystkich ćwiczeniach, w tym ich nazwę, zaangażowanie mięśni, poziom trudności itp. Do tego posiada ona odnośniki do plików multimedialnych.

Tabela workouts (lokalna) zawiera informacje na temat zapisanych planów użytkownika. Zawarte w niej są: nazwa planu oraz tablica, która przechowuje id ćwiczenia, ilość powtórzeń, serii, użyty ciężar oraz tempo.

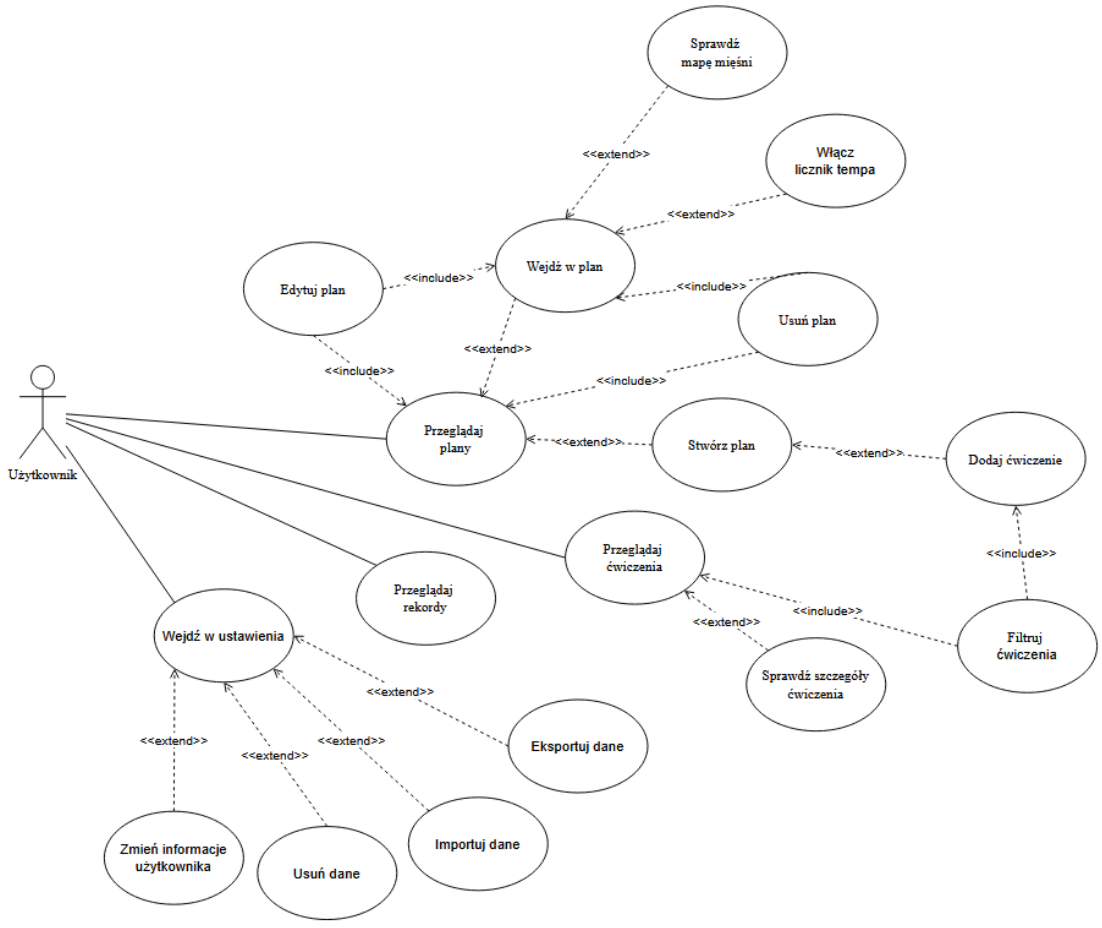


Rys. 4.2 *Diagram ERD [opracowanie własne]*

4.3. Diagram przypadków użycia

Na rysunku 4.3 przedstawiono diagram przypadków użycia, który pokazuje interakcje pomiędzy użytkownikiem aplikacji, a jej głównymi funkcjami. Diagram Przypadków użycia przedstawia system z perspektywy użytkownika końcowego. Pokazuje on jakie czynności może wykonać użytkownik w systemie oraz jakie zależności zachodzą pomiędzy poszczególnymi przypadkami użycia.

Na diagramie przedstawiony jest jeden użytkownik, który reprezentuje osobę korzystającą z aplikacji. Użytkownik może wykonać różne czynności związane z obsługą aplikacji, takie jak przeglądanie ćwiczeń czy tworzenie planów treningowych.



Rys. 4.3 *Diagram przypadków użycia [opracowanie własne]*

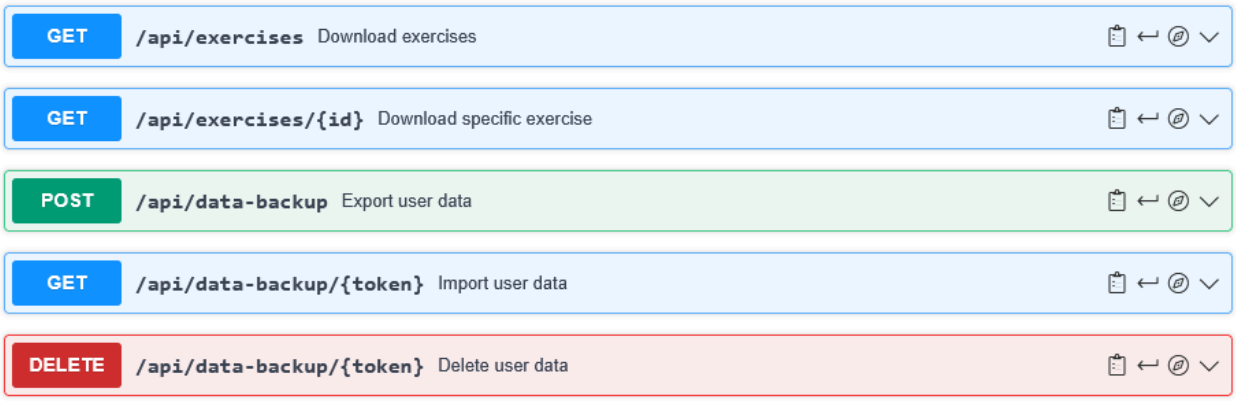
4.4. Endpointy API

Komunikacja pomiędzy aplikacją a zdalnym serwerem odbywa się za pomocą interfejsu REST API. Zastosowanie tego rozwiązania pozwala na wymianę danych w prosty i wydajny sposób niezależny od technologii serwera. W prezentowanej aplikacji dane o ćwiczeniach przechowywane są w pliku JSON, umieszczonym na zewnętrznym serwerze (prostym backendzie). Dzięki temu możliwe jest odczytywanie informacji o ćwiczeniach.

Aplikacja komunikuje się z serwerem za pomocą żądań HTTP, korzystając z biblioteki Axios do obsługi zapytań. Wszystkie przesyłane dane mają format JSON, co pozwala na ich łatwe przetwarzanie w środowisku JavaScript.

Operacje na bazie danych będą wykonywane przy pomocy endpointów przedstawionych na rysunku 4.4. Endpointy zostały przedstawione za pomocą narzędzia Swagger.

Dwa pierwsze endpointy posłużą do pobierania i aktualizowania listy ćwiczeń, która będzie zapisana lokalnie w pamięci urządzenia. Natomiast pozostałe endpointy posłużą do eksportowania i importowania danych użytkownika.



Rys. 4.4 *Endpointy API [opracowanie własne]*

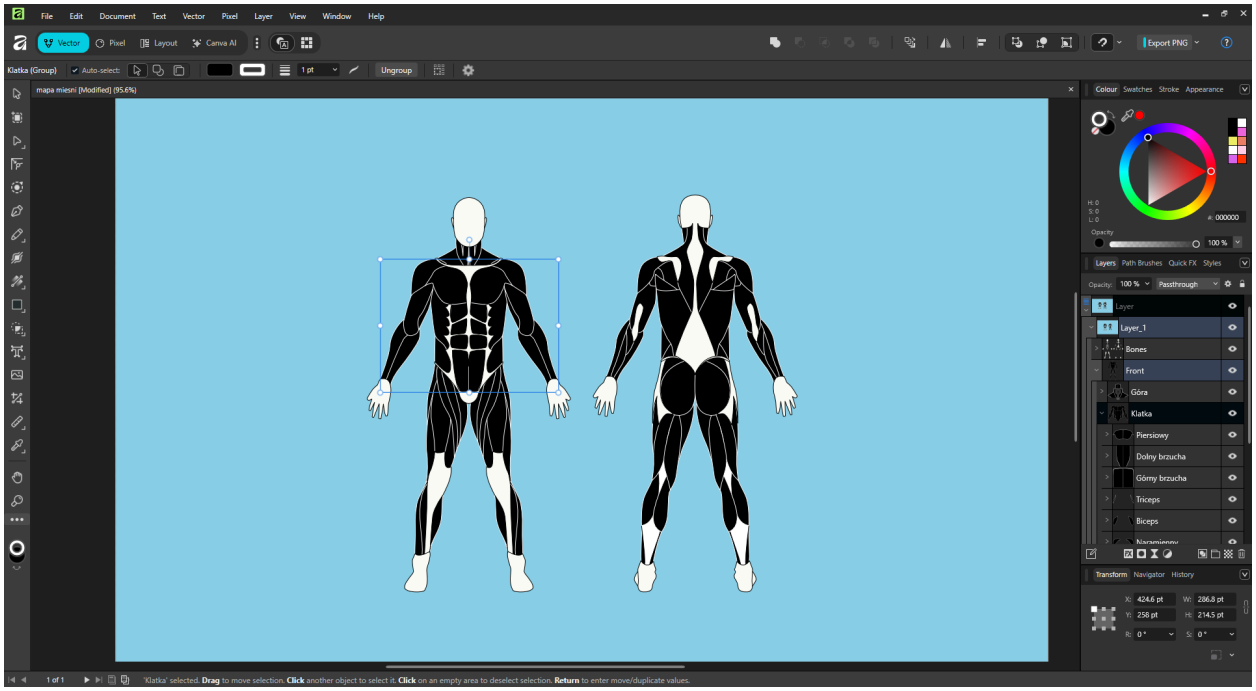
Wszystkie operacje będą wykonywane przy użyciu protokołu HTTPS, który zapewni szyfrowanie danych przesyłanych pomiędzy aplikacją a serwerem.

Zastosowanie architektury REST umożliwia też łatwe rozbudowanie systemu w przyszłości. Można bez problemu dodać logowanie użytkowników, synchronizację planów treningowych w chmurze, integrację z zewnętrznymi serwisami fitness lub połączenie z innym API.

4.5. Implementacja mapy mięśni

Mapa mięśniowa to kluczowy element aplikacji ma ona na celu pokazać użytkownikowi to, jak poszczególne ćwiczenia wpływają na jego mięśnie. Rozwiązanie zostało oparte na warstwach obrazów SVG (*ang. Scalable Vector Graphics*), które są na siebie nakładane i można im zmienić dynamicznie kolor przy użyciu stylów CSS (*ang. Cascading Style Sheets*). Jest to proste rozwiązanie, które nie wymaga przygotowywania paru wersji jednego elementu na potrzeby zaprezentowania różnych wersji natężenia mięśni, dodatkowo zdjęcia jest łatwo pozycjonować. Wadą tego rozwiązania jest potrzeba przygotowania dużej liczby pojedynczych obrazów SVG dla każdej grupy mięśniowej.

Warstwy mięśni w plikach SVG zostały przygotowane w darmowym programie Canva Affinity. Proces przedstawiono na rysunku 4.5.



Rys. 4.5 *Przygotowanie warstw mięśni w Affinity [opracowanie własne]*

W aplikacji zdjęcia są ustawione warstwami co ułatwia pozycjonowanie i wyświetlanie ich. Funkcja zarządzająca warstwami przyjmuje listę warstw, obliczoną objętość treningową oraz cel treningowy użytkownika. Mapa mięśni przedstawia mięśnie człowieka z widoku od przodu lub od tyłu (użytkownik wybiera, na którą stronę patrzy za pomocą przycisku na stronie mapy mięśni). Implementację funkcji zaprezentowano na listingu 4.1

Listing 4.1 *Implementacja funkcji zarządzającej warstwami SVG [opracowanie własne]*

```
const calculateVolumeFills = (layers, muscleVolumes, goal) => {
    const fills = {};
    layers.forEach(layer => (fills[layer.name] = 'black'));

    if (!muscleVolumes) return fills;

    Object.entries(muscleVolumes).forEach(([name, volume]) => {
        if (fills[name] === undefined) return;
        const ratio = volumeToRatio(volume, goal);
        fills[name] = ratioToColor(ratio);
    });

    return fills;
};
```

Funkcja odpowiada za wyznaczenie kolorów wypełnienia warstw mapy mięśni na podstawie objętości treningowej poszczególnych partii mięśniowych. Dla każdej warstwy przypisywany jest domyślny kolor (czarny), który następnie zostaje zmieniony w zależności od poziomu zaangażowania mięśnia oraz aktualnego celu treningowego.

Funkcja odpowiadająca za obliczenie objętości treningowej została przedstawiona na listingu 4.2. Służy ona do obliczenia wartości zapisanej pod zmienną muscleVolumes.

Listing 4.2 *Implementacja funkcji obliczającej objętość treningową [opracowanie własne]*

```
export const calculateMuscleVolume = (planExercises = [], exercisesDB = []) => {
    const { bodyweight } = useUserProfileStore();

    const muscleVolumes = {};

    if (!Array.isArray(planExercises) || !Array.isArray(exercisesDB)) {
        return muscleVolumes;
    }

    planExercises.forEach(planExercise => {
        const fullExercise = exercisesDB.find(e => e.id === planExercise.id);

        const sets = Number(planExercise.sets);
        const reps = Number(planExercise.reps);
        const weight = Number(planExercise.weight) || 0;

        if (!Number.isFinite(sets) || !Number.isFinite(reps)) return;

        let effectiveWeight = 0;
        const hasBodyweight = Number.isFinite(fullExercise.bodyweightFactor);
        const hasExternalWeight = Number.isFinite(weight) && weight > 0;

        if (hasBodyweight) {
            effectiveWeight = bodyweight * fullExercise.bodyweightFactor;
        }

        if (hasExternalWeight) {
            effectiveWeight += weight;
        }

        if (!hasBodyweight && hasExternalWeight) {
            effectiveWeight = weight;
        }

        const volume = sets * reps * effectiveWeight;
        if (!Number.isFinite(volume) || volume <= 0) return;

        fullExercise.muscles.forEach(({ name, ratio }) => {
            if (!name || !Number.isFinite(ratio)) return;
            muscleVolumes[name] = (muscleVolumes[name] || 0) + volume * ratio;
        });
    });

    return muscleVolumes;
};
```

Funkcja oblicza całkowitą objętość treningową dla poszczególnych partii mięśniowych na podstawie podanego planu treningowego oraz informacji zawartych w bazie ćwiczeń. Objętość treningowa jest wyznaczana jako iloczyn liczby serii, powtórzeń oraz efektywnego obciążenia, które uwzględnia ciężar oraz ciężar masy ciała użytkownika i procentowy udział masy własnego ciała w zależności od rodzaju ćwiczenia. Następnie uzyskana wartość jest rozdzielana pomiędzy zaangażowane mięśnie zgodnie z określonymi współczynnikami.

4.6. Implementacja bazy ćwiczeń

Baza ćwiczeń jest przechowywana w pliku JSON. Dostęp do serwera odbywa się poprzez protokół HTTP i zapytania REST. W bazie znajduje się lista ćwiczeń, zawierająca informacje o nazwie ćwiczenia, wymaganym sprzęcie, typie oraz poziomie trudności. Dodatkowo każde ćwiczenie posiada kategorię oraz rodzaj wykonywanego ruchu. Głównym elementem bazy jest opis zaangażowania poszczególnych partii mięśniowych wraz ze stopniem ich aktywacji. Baza do tego zawiera pisemne instrukcje wykonania ćwiczenia oraz obrazek gif prezentujący poprawną technikę wykonania wraz z wyróżniającym obrazkiem.

Gdy użytkownik pobierze aplikację i połączy się z serwerem lista ćwiczeń zostanie pobrana i zapisana lokalnie w pamięci. To rozwiązanie pozwala na korzystanie z aplikacji bez dostępu do Internetu, a gdy dostęp do sieci zostanie wznowiony można wtedy pobrać aktualizacje do listy ćwiczeń. Alternatywą tego rozwiązania byłoby przechowywanie listy ćwiczeń wyłącznie w pamięci lokalnej pobranej razem z aplikacją. Wadą tego rozwiązania byłaby potrzeba aktualizacji aplikacji za każdym razem gdy aktualizowana jest baza ćwiczeń. Innym sposobem realizacji byłoby przechowywanie bazy wyłącznie na serwerze co zmniejszyłoby rozmiar zajętej pamięci. To rozwiązanie sprawiłoby, że korzystanie z aplikacji możliwe jest wyłącznie podczas połączenia z Internetem. W aktualnych czasach taki sposób implementacji nie jest do końca zły z powodu powszechnego dostępu do Internetu oraz sieci komórkowej, jednakże trzeba pamiętać o użytkownikach bez możliwości połączenia. Na tej podstawie aktualna implementacja jest najlepsza, ponieważ mimo zajmowania nieznacznie większej ilości miejsca pozwala na korzystanie z aplikacji w trybie offline oraz pozwala na pozostawienie aktualizacji aplikacji na realne zmiany takie jak rozbudowa funkcji czy zmiany działania systemu.

4.7. Tworzenie planów treningowych

Użytkownicy aplikacji mogą przygotować swój własny plan treningowy. Tworzenie planu przebiega w ten sposób:

1. Użytkownik tworzy nowy plan;
2. Wybiera nazwę dla planu;
3. Przegląda listę ćwiczeń i wybiera ćwiczenia, które go interesują;
4. Po dodaniu ćwiczenia wybiera ile serii i powtórzeń należy wykonać;
5. W trakcie dodawania może sprawdzić wpływ ćwiczeń na partie mięśniowe za pomocą mapy mięśni;
6. Przed zapisaniem może poprosić o sugestie ćwiczeń, które uzupełnią trening o ćwiczenia na brakujące partie;

Przeglądanie ćwiczeń wyświetla tylko nazwę ćwiczenia, użytkownik ma możliwość rozwinięcia kafelka z danym ćwiczeniem w celu uzyskania dokładniejszych informacji, takich jak: wpływ na mięśnie, ekwipunek, poziom trudności itp.

Użytkownik ma możliwość zapisania ciężaru na jakim ćwiczył w danym ćwiczeniu. Ilość serii oraz powtórzeń jest indywidualna dla każdego z ćwiczeń. Gdy użytkownik poprosi o uzupełnienie ćwiczeń, aplikacja sprawdzi pominięte mięśnie oraz zasugeruje ćwiczenia, które na nie wpływają (np. gdy użytkownik nie ma ćwiczeń wpływających na nogi, aplikacja może zasugerować jakąś formę przysiadów).

4.8. Wybór oraz edycja planów treningowych

Użytkownik ma możliwość tworzenia i zarządzania wieloma planami treningowymi. Wszystkie zapisane plany dostępne są z menu planów gdzie prezentowane są w formie listy. Po wybraniu planu użytkownik może przeglądać przypisane do niego ćwiczenia oraz sprawdzać liczbę serii i powtórzeń. Edytowanie planu pozwala użytkownikowi na zmianę jego nazwy, dodawanie nowych ćwiczeń, modyfikację istniejących już ćwiczeń (zmiana liczby serii i powtórzeń) lub usuwanie elementów, które nie są już potrzebne. Edytowanie planu pozwala też na jego całkowite usunięcie w ramach utrzymania porządku i pozbycia się starych treningów.

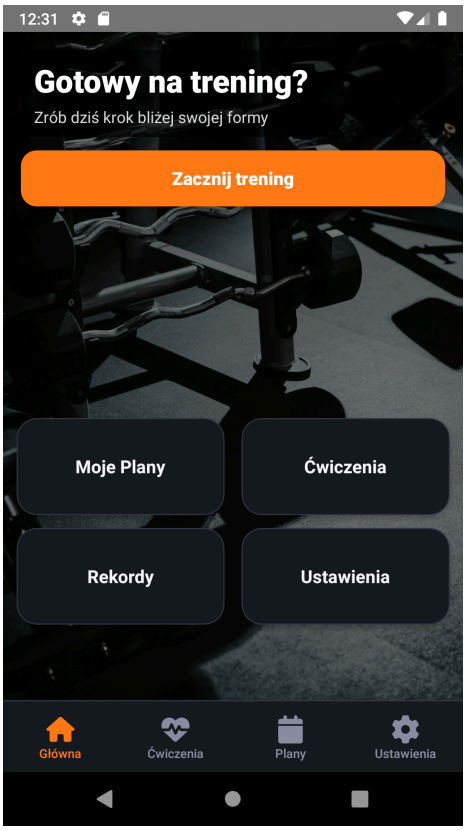
5. Interfejs użytkownika

W tej części pracy zostanie przedstawiona implementacja interfejsu użytkownika. Celem tego rozdziału będzie przedstawienie i omówienie zaprojektowanego systemu oraz interakcji użytkownika z aplikacją. Przedstawione zostaną główne ekrany aplikacji, elementy znajdujące się na nich oraz ich układ.

Zanim użytkownik dostanie się do menu głównego, poproszony zostanie o wypełnienie, krótkiego formularza przedstawionego na rysunku 5.1. Formularz ten ma na celu zapisać masę ciała użytkownika, która będzie potrzebna do obliczania wpływu ćwiczeń na mięśnie. Drugą informacją będzie cel treningowy, który posłuży do obliczenia progu ostrzegania przed przetrenowaniem. Formularz wyświetli się tylko przy pierwszym uruchomieniu aplikacji. Swój wybór lub zmianę masy ciała użytkownik będzie mógł później zmienić z poziomu ustawień.

Rys. 5.1 *Modal startowy [opracowanie własne]*

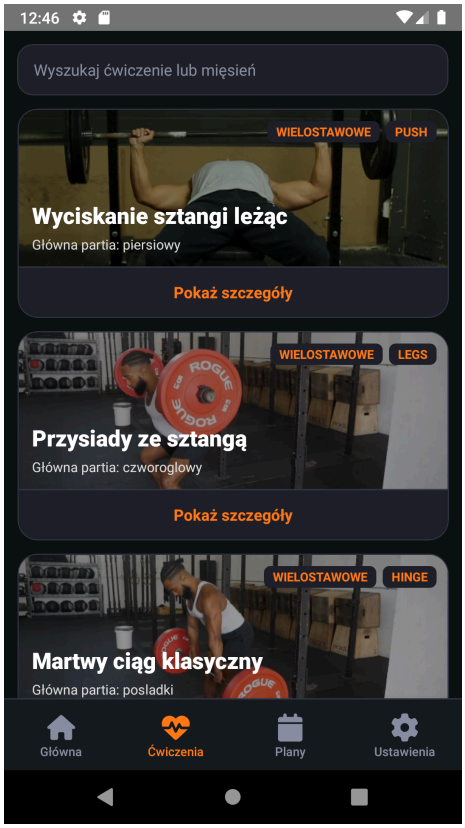
Po wpisaniu swojej wagi oraz wyborze celu treningowego użytkownik trafi do menu głównego, które zostało przedstawione na rysunku 5.2. Menu główne jest ekranem powitalnym dla użytkownika przy każdym kolejnym uruchomieniu aplikacji.



Rys. 5.2 *Menu główne [opracowanie własne]*

Menu główne to pierwsza strona jaką widzi użytkownik. Z niego ma dostęp do pozostałych części aplikacji, poprzez nawigację za pomocą kafelków na stronie głównej lub poprzez zakładki na dole ekranu. Z menu głównego użytkownik może przejść do bazy ćwiczeń, zapisanych planów, rekordów lub ustawień.

Kolejnym ekranem jest baza ćwiczeń dostępna pod przyciskiem “Ćwiczenia”. Widok bazy został przedstawiony na rysunku 5.3.

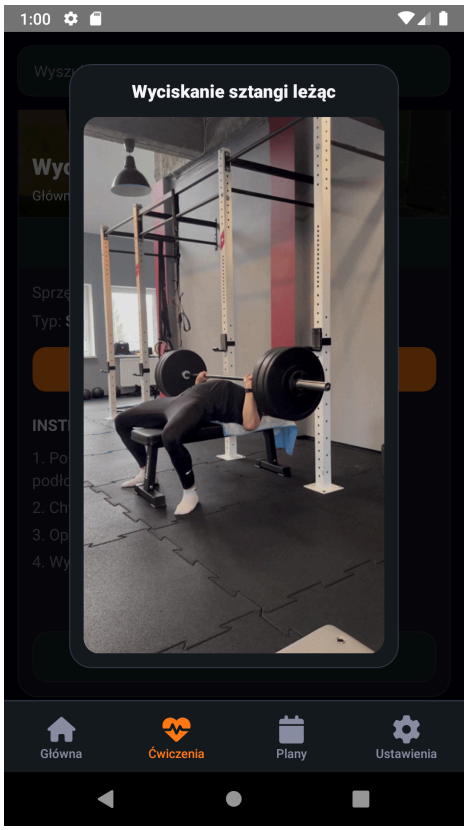


Rys. 5.3 *Baza ćwiczeń [opracowanie własne]*

Ekran “Ćwiczenia” służy jako baza ćwiczeń, które są udostępniane przez aplikację. Ćwiczenia są wyświetlone w postaci listy i każde z nich może zostać rozwinięte w celu uzyskania większej ilości informacji na temat danego ćwiczenia. Jest też dostępna na tej stronie wyszukiwarka, która pozwala użytkownikowi szukać i filtrować ćwiczenia na liście poprzez dowolne kryterium (np. Nazwa, mięsień, na który chcemy ćwiczyć, sprzęt czy nawet rodzaj ćwiczenia). W swojej podstawowej, zwiniętej formie, karta ćwiczenia udostępnia informacje takie jak: nazwa, główna partia mięśniowa, rodzaj ćwiczenia oraz ruch ćwiczenia. Gdy użytkownik naciśnie przycisk “Pokaż szczegóły”, karta zostanie rozwinięta i aplikacja wyświetli więcej informacji na temat ćwiczenia. Na rysunku 5.4 przedstawiony został widok rozwiniętej karty, która dodatkowo pokazuje informacje na temat wymaganego sprzętu, typu oraz poziomu trudności ćwiczenia. Dodatkowo użytkownik ma możliwość sprawdzenia wpływu ćwiczenia na jego ciało poprzez mapę mięśni. W celu prawidłowego wykonania ćwiczenia została przygotowana instrukcja tekstowa oraz przycisk “Sprawdź technikę”, który uruchomi modal

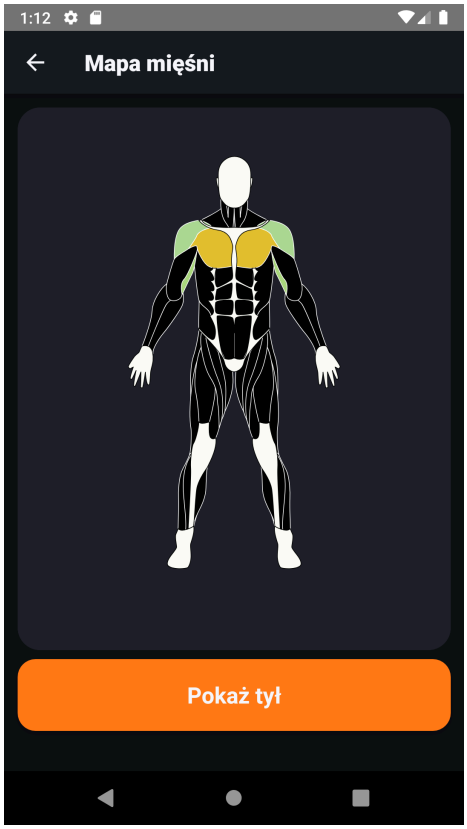
z obrazkiem gif przedstawiającym poprawne wykonanie ćwiczenia. Modal z gifem został przedstawiony na rysunku 5.5.

Rys. 5.4 *Rozwinięta karta [opracowanie własne]*

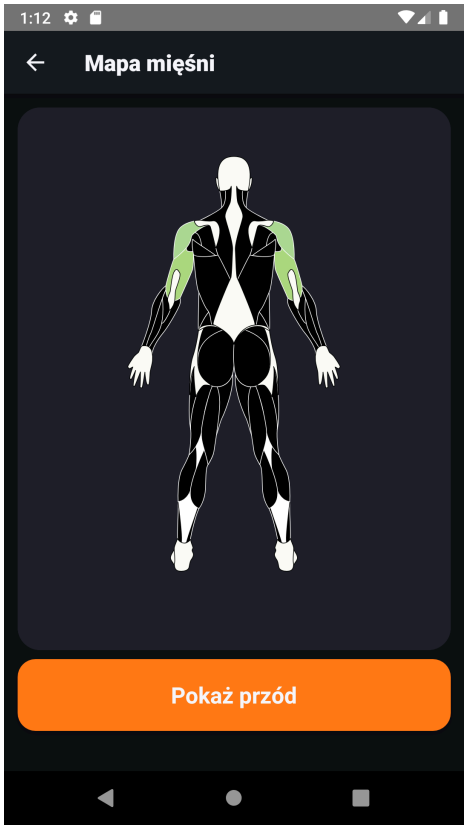


Rys. 5.5 *Modal z techniką wykonania [opracowanie własne]*

Mapa mięśni ma dwa różne tryby, w zależności od tego z jakiego ekranu ją uruchomimy. Gdy zostanie uruchomiona z poziomu bazy ćwiczeń, posłuży jako narzędzie edukacyjne, które ma na celu pokazać wpływ ćwiczenia na mięśnie. Im bardziej ciemny i czerwony kolor tym większy wpływ na daną partię mięśniową ma dane ćwiczenie. Skala wpływu ćwiczeń na mięśnie idzie od bladego zielonego przez żółty i pomarańczowy, kończąc na ciemnym czerwonym. Efekt działania dla wyciskania sztangi leżąc został przedstawiony na rysunku 5.6 oraz na rysunku 5.7.

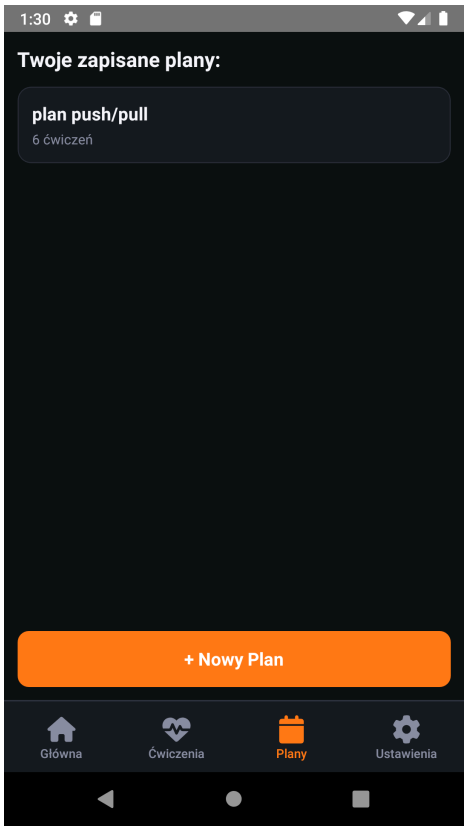


Rys. 5.6 *Mapa mięśni, wersja edukacyjna (front) [opracowanie własne]*

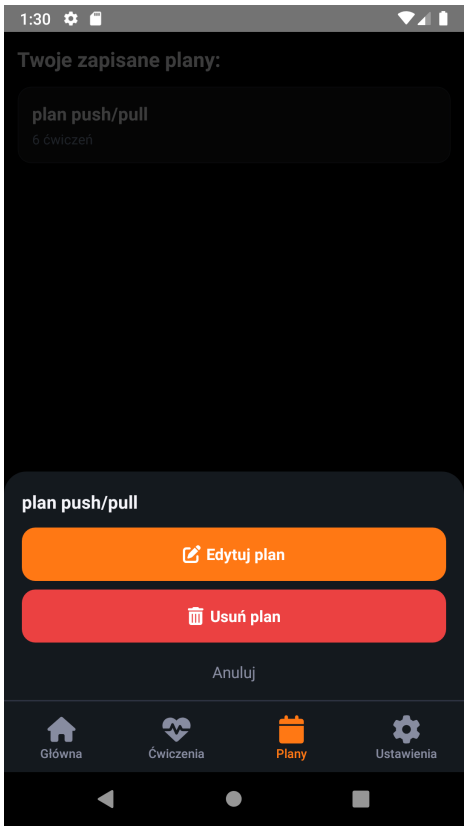


Rys. 5.7 *Mapa mięśni, wersja edukacyjna (tył) [opracowanie własne]*

Następnym ekranem będzie ekran planów. Ekran z stworzonym jednym planem został przedstawiony na rysunku 5.8. Na liście planów mamy możliwość przejścia do wybranego planu, stworzenia nowego, lub przytrzymanie planu w ramach edycji lub usunięcia co zostało przedstawione na rysunku 5.9.

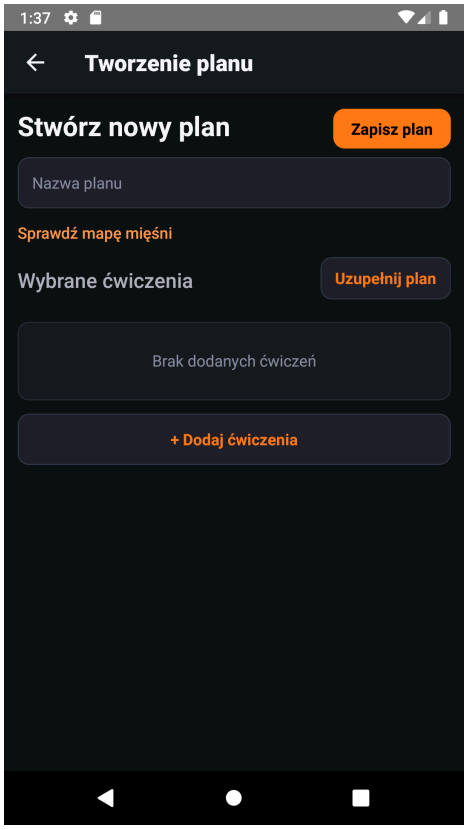


Rys. 5.8 *Ekran planów*
[opracowanie własne]

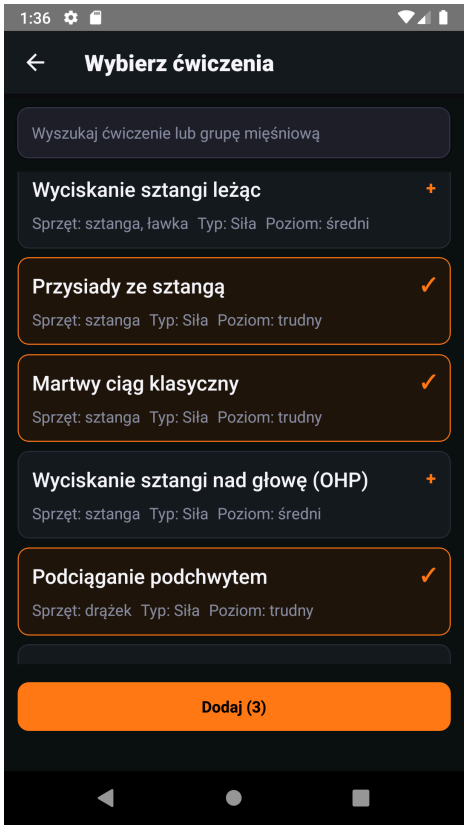


Rys. 5.9 *Menu kontekstowe*
[opracowanie własne]

Pusty ekran tworzenia nowego planu został przedstawiony na rysunku 5.10. Podczas tworzenia planu możemy nadać mu nazwę, poprosić o analizę planu i sugerowanie ćwiczeń, oraz dodanie ćwiczeń ręcznie. Dodawanie ćwiczeń zostało przedstawione na rysunku 5.11.

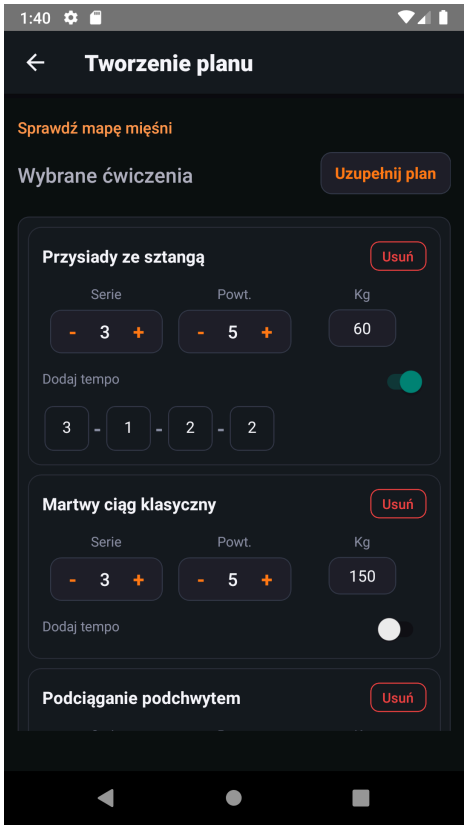


Rys. 5.10 *Pusty plan*
[opracowanie własne]

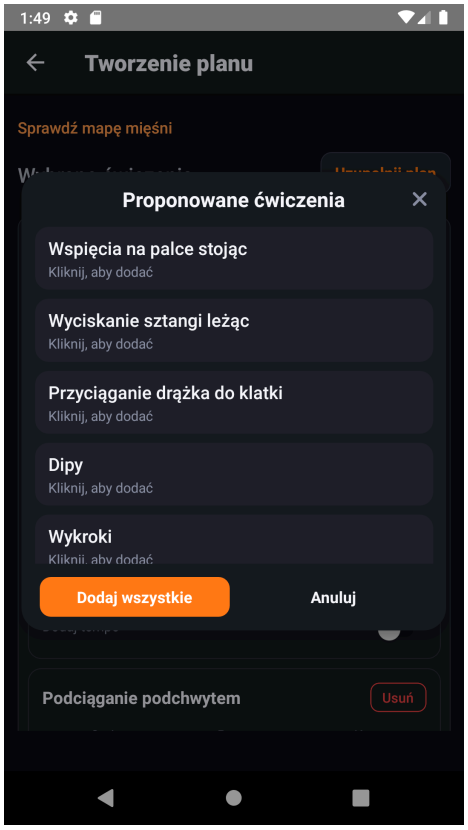


Rys. 5.11 *dodawanie ćwiczeń do planu*
[opracowanie własne]

Po dodaniu ćwiczeń użytkownik ma możliwość ustawienia ilości serii, powtórzeń, ciężaru oraz tempa dla każdego ćwiczenia. Przykład został pokazany na rysunku 5.12. Prośba o sugestie ćwiczeń została przedstawiona na rysunku 5.13

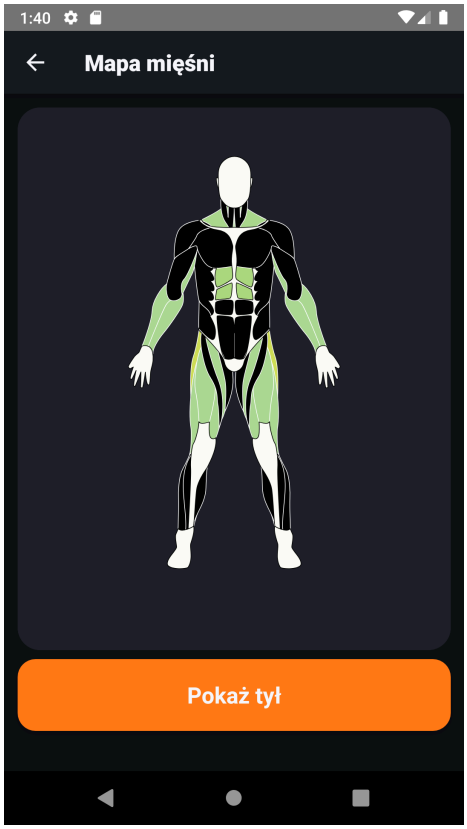


Rys. 5.12 *Plan wypełniony ćwiczeniami [opracowanie własne]*

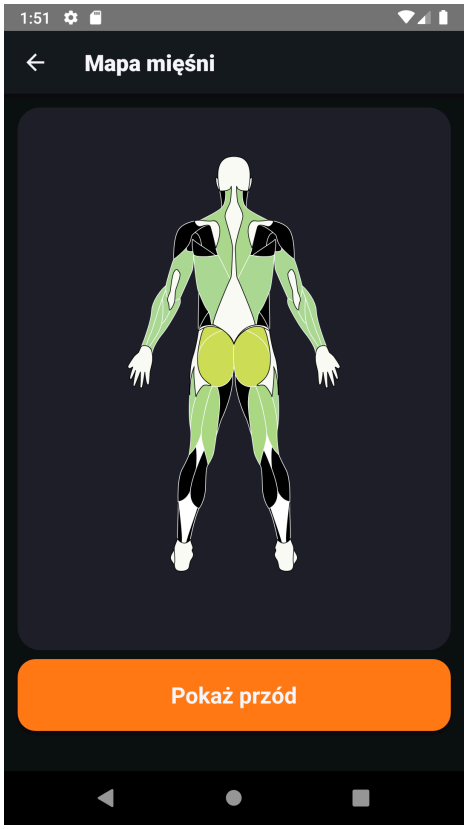


Rys. 5.13 *Proponowane ćwiczenia [opracowanie własne]*

Z poziomu planu użytkownik może też sprawdzić wpływ pracy całego planu na mięśnie. Ta funkcjonalność jest drugim trybem mapy mięśniowej, który pozwala na obliczanie objętości treningowej jednego planu. Wpływ przygotowanego planu bez dodanych ćwiczeń z sugestii został przedstawiony na rysunkach 5.14 i 5.15.



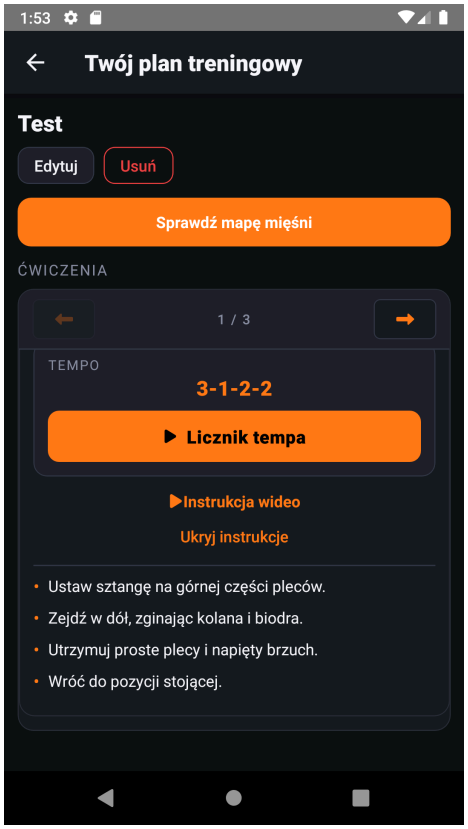
Rys. 5.14 *Mapa mięśni, wersja obliczeniowa (front) [opracowanie własne]*



Rys. 5.15 *Mapa mięśni, wersja obliczeniowa (tył) [opracowanie własne]*

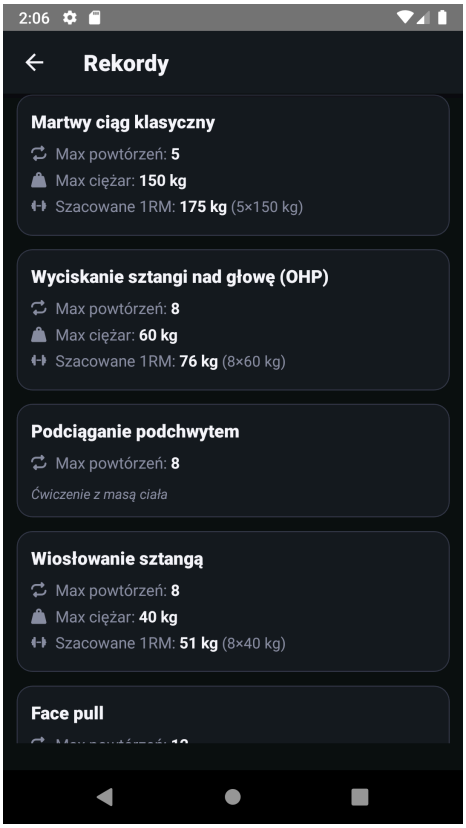
Po wejściu na utworzony plan, dodane ćwiczenia wyświetlają się w formie karuzeli. Jeżeli ćwiczenie ma dodane tempo pojawi się dodatkowo opcja uruchomienia licznika. Uruchomienia go sprawi, że po 5 sekundach zacznie odliczać podane przez nas czasy. Tempo zapętli się tyle razy ile powtórzeń w serii. Domyślny widok uruchomionego planu został przedstawiony na rysunku 5.16, a widok z rozwiniętymi instrukcjami został przedstawiony na rysunku 5.17. Naciśnięcie przycisku “Instrukcja wideo” uruchomi ten sam modal, który był przedstawiony na rysunku 5.5.

Rys. 5.16 *Widok planu*
[opracowanie własne]



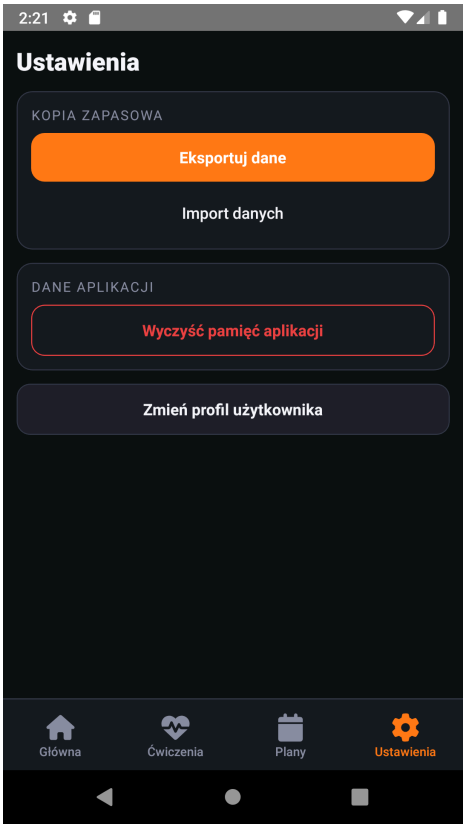
Rys. 5.17 *Plan z instrukcją*
[opracowanie własne]

Przedostatnim ekranem jest ekran rekordów, przedstawiony na rysunku 5.18. Użytkownik może w nim sprawdzać swoje rekordy wykonanych ćwiczeń. Rekordy są zapisywane automatycznie na podstawie tego co jest zawarte w planach. Więc ekran służy tylko jako narzędzie do sprawdzania postępów oraz jak fajny dodatek, który cieszy wielu użytkowników siłowni. Każde ćwiczenie siłowe posiada maksymalną ilość powtórzeń, oraz maksymalny ciężar. Do tego do każdego ćwiczenia siłowego obliczany jest 1RM za pomocą wzoru Epleya. Ćwiczenia z masą ciała natomiast mają tylko maksymalną liczbę powtórzeń, chyba że są wykonywane z dodatkowym ciężarem, który wtedy też jest zapisywany.



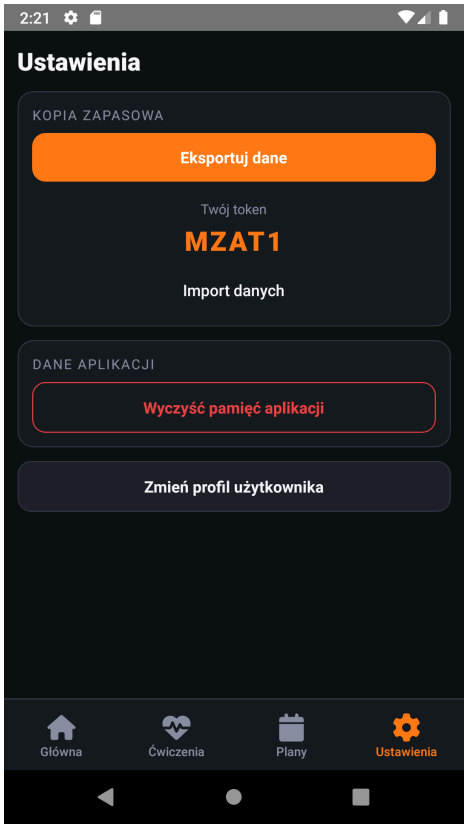
Rys. 5.18 *Ekran rekordów [opracowanie własne]*

Ostatnim ekranem jest ekran ustawień przedstawiony na rysunku 5.19. Z poziomu ustawień użytkownik ma dostęp do eksportu i importu danych. Wyczyszczenia pamięci podręcznej oraz zmianu ustawień profilu użytkownika. Zmiana ustawień profilu spowoduje otwarcie modalu z rysunku 5.1, który wyświetlił się przy pierwszy uruchomieniu aplikacji.



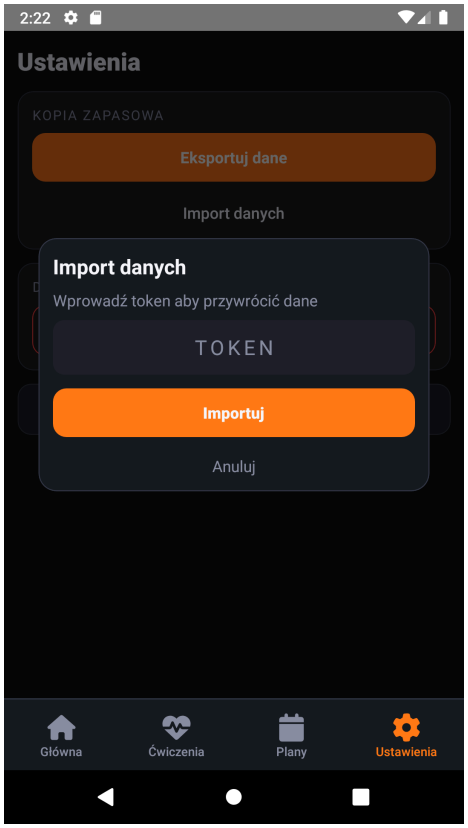
Rys. 5.19 *Ekran ustawień [opracowanie własne]*

Po kliknięciu eksport danych, dane użytkownika zostaną przesłane do bazy. Gdy dane zostaną zapisane w bazie, użytkownik otrzyma 5 znakowy token, dzięki któremu będzie mógł zaimportować swoje dane do aplikacji. Przykład eksportu danych został przedstawiony na rysunku 5.20, a modal importowania na rysunku 5.21.



Rys. 5.20 *Eksport danych*

[opracowanie własne]



Rys. 5.21 *Import danych*

[opracowanie własne]

6. Testy jednostkowe

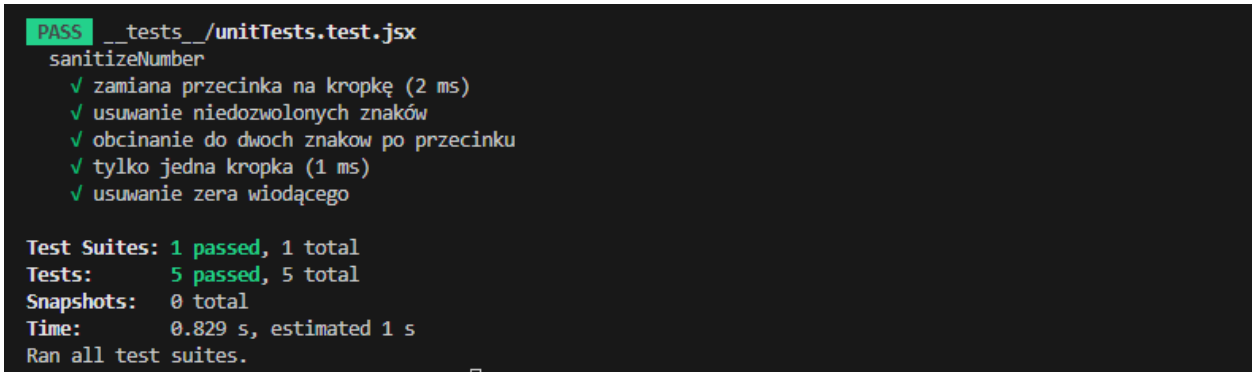
W tym rozdziale zostaną przedstawione testy jednostkowe (*ang. Unit Tests*) stosowane w celu weryfikacji działania kluczowych funkcji aplikacji. Testy jednostkowe stanowią podstawową formę testowania oprogramowania i polegają na sprawdzaniu pojedynczych elementów systemu, takich jak funkcje lub moduły, w izolacji od pozostałych części aplikacji. Ich głównym celem jest upewnienie się, że dana jednostka programu działa zgodnie z założeniami oraz poprawnie reaguje na różne dane wejściowe.

Zastosowanie testów jednostkowych pozwala na wczesne wykrycie błędów, ograniczenie ryzyka wystąpienia regresji oraz zwiększenie ogólnej stabilności całego systemu. Przeprowadzone testy potwierdzają poprawność implementacji oraz zgodność działania aplikacji z wymaganiami projektowymi [5].

Przykłady testów jednostkowych możemy zobaczyć na listingu 6.1, który pokazuje 5 testów sprawdzających funkcję oczyszczającą dane wejściowe przy wpisywaniu wagi. Funkcja przyjmuje dane w postaci wartości tekstowej i zamienia je na, poprawny format wpisywania wagi, czyli kropka jako separator wartości dziesiętnych oraz tylko dwie cyfry dziesiętne. Testy zostały napisane przy użyciu podejścia Arrange-Act-Assert (AAA), polegającego na przygotowaniu danych testowych, wykonaniu testowanej operacji oraz weryfikacji otrzymanego wyniku. Wynik testów jednostkowych został przedstawiony na rysunku 6.1.

Listing 6.1 *Testy funkcji oczyszczającej input wagi [opracowanie własne]*

```
describe('sanitizeNumber', () => {
    test('zamiana przecinka na kropkę', () => {
        const testValue = '12,34';
        const finalValue = '12.34';

        const result = sanitizeNumber(testValue);

        expect(result).toBe(finalValue);
    })
    test('usuwanie niedozwolonych znaków', () => {
        const testValue = 'a1b2c.3d4';
        const finalValue = '12.34';

        const result = sanitizeNumber(testValue);

        expect(result).toBe(finalValue);
    });
    test('obcinanie do dwóch znaków po przecinku', () => {
        const testValue = '12.342323';
        const finalValue = '12.34';

        const result = sanitizeNumber(testValue);

        expect(result).toBe(finalValue);
    });
    test('tylko jedna kropka', () => {
        const testValue = '1.2.3.4.';
        const finalValue = '1.23';

        const result = sanitizeNumber(testValue);

        expect(result).toBe(finalValue);
    });
    test('usuwanie zera wiodącego', () => {
        const testValue = '012.34';
        const finalValue = '12.34';

        const result = sanitizeNumber(testValue);

        expect(result).toBe(finalValue);
    });
})
```

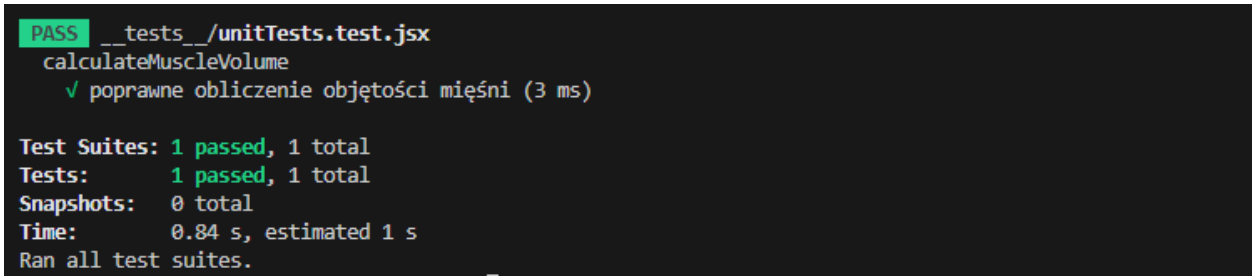


Rys. 6.1 - *Wynik testów jednostkowych [opracowanie własne]*

Drugi przykład testu jednostkowego to test funkcji odpowiedzialnej za obliczanie objętości treningowej dla poszczególnych mięśni. Test został przedstawiony na listingu 6.2, a wynik na rysunku 6.2. Ten test sprawdza poprawność działania funkcji, która jest bardziej złożona. Oprócz napisania samego testu wymagane było zamockowanie działania store'a biblioteki Zustand, który odpowiada za przechowywanie ustawień profilu użytkownika, w tym przypadku jego wagi. Fragment kodu *beforeEach()* służy do wykonania jakiejś instrukcji przed każdym testem. Do testowania tej funkcji potrzebny jest dostęp do wagi użytkownika, więc została tam wywołana instrukcja ustawiająca ją na 80 przed każdym testem.

Listing 6.2 *Test jednostkowy funkcji obliczającej objętość treningową [opracowanie własne]*

```
jest.mock('../ZustandStores/UserProfileStore', () => ({
    useUserProfileStore: jest.fn()
}));

describe('calculateMuscleVolume', () => {
    beforeEach(() => {
        useUserProfileStore.mockReturnValue({
            bodyweight: 80
        });
    });

    test('poprawne obliczenie objętości mięśni', () => {
        const planExercises = [
            {id: 1, sets: 3, reps: 10, weight: 20 }
        ];

        const exercisesDB = [
            {
                id: 1,
                bodyweightFactor: 0.5,
                muscles: [
                    { name: 'chest', ratio: 0.7 },
                    { name: 'triceps', ratio: 0.3 }
                ]
            }
        ];

        const result = calculateMuscleVolume(planExercises, exercisesDB);

        expect(result).toEqual({
            chest: 1800 * 0.7,
            triceps: 1800 * 0.3
        });
    });
});
```



Rys. 6.2 - *Wynik testu jednostkowego funkcji liczącej objętość mięśniową [opracowanie własne]*

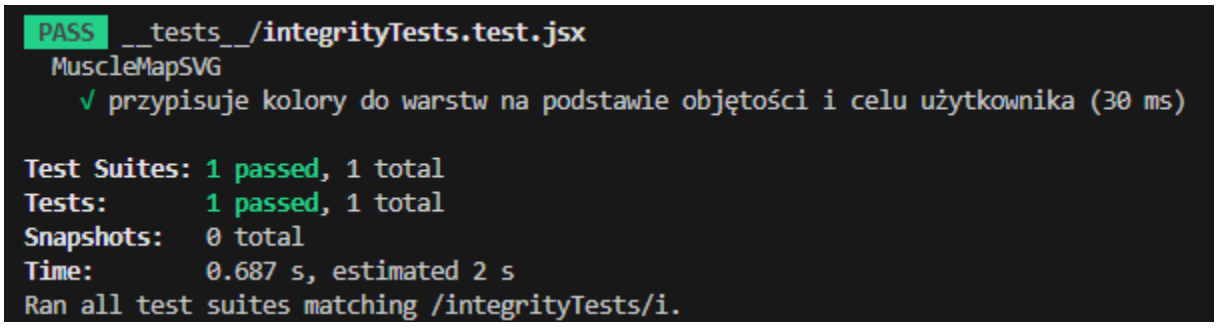
7. Testy integracyjne

W tym rozdziale zostaną przedstawione testy integracyjne (*ang. Integrity tests*), których celem było sprawdzenie poprawności współdziałania poszczególnych modułów aplikacji. Testy integracyjne polegają na testowaniu interakcji pomiędzy niezależnie zaimplementowanymi komponentami systemu, takimi jak warstwa logiki aplikacji, zarządzanie stanem oraz komunikacja z danymi. Ich zastosowanie pozwala ocenić, czy połączenie poszczególnych elementów systemu działa zgodnie z założeniami projektowymi.

Przeprowadzenie testów integracyjnych umożliwiło wykrycie potencjalnych problemów wynikających z integracji różnych części aplikacji oraz potwierdzenia poprawności przepływu danych pomiędzy nimi.

Przykład testów integracyjnych można zobaczyć na listingu 7.1, który przedstawia test sprawdzający poprawność działania komponentu MuscleMapSVG. Test sprawdza czy komponent poprawnie przyjmuje dane wejściowe oraz przypisuje kolor do mięśni, które mają podaną objętość. Sprawdzane jest również czy pod uwagę brany jest cel treningowy.

W przypadku gdy użytkownik ma określony cel treningowy i dostępne są dane o objętości danego mięśnia, komponent powinien przypisać odpowiedni kolor do wizualnej reprezentacji. Test nie weryfikuje konkretnego koloru, a jedynie fakt, że warstwa nie pozostaje w stanie domyślnym. Pozytywny wynik testu został przedstawiony na rysunku 7.1.



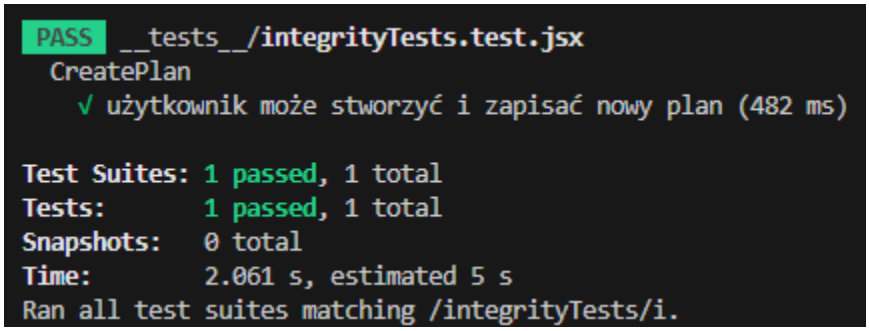
Rys. 7.1 - *Wynik testu integracyjnego komponentu MuscleMapSVG [opracowanie własne]*

Listing 7.1 *Test integracyjny sprawdzający działanie komponentu MuscleMapSVG [opracowanie własne]*

```
jest.mock('../ZustandStores/UserProfileStore', () => ({
    useUserProfileStore: jest.fn(),
}));
jest.mock('react-native-svg', () => {
    return {
        Svg: ({ children }) => <>{children}</>,
        G: ({ children }) => <>{children}</>,
    };
});

describe('MuscleMapSVG', () => {
    test('przypisuje kolory na podstawie objętości i celu użytkownika', () => {
        useUserProfileStore.mockReturnValue({
            goal: 'hypertrophy',
        });

        const layers = [
            {
                name: 'najszerszy',
                Component: ({ fill }) => <layer fill={fill} />,
            },
            {
                name: 'triceps',
                Component: ({ fill }) => <layer fill={fill} />,
            },
        ];

        const muscleVolumes = {
            najszerszy: 1500,
            triceps: 700
        };

        const { UNSAFE_queryAllByType } = render(
            <MuscleMapSVG
                layers={layers}
                muscleVolumes={muscleVolumes}
            />
        );

        const [layer] = UNSAFE_queryAllByType('layer');
        expect(layer.props.fill).not.toBe('black');
    });
});
```

Kolejnym testem integracyjnym będzie test komponentu odpowiedzialnego za tworzenie planu treningowego użytkownika. Test został przedstawiony na listingu 7.2, a wynik tego testu na rysunku 7.2.

Test sprawdza poprawność zapisu planu do pamięci urządzenia. Przed rozpoczęciem testu mockowany jest wybór ćwiczeń użytkownika. Po przyjęciu wybranych ćwiczeń test wpisuje nazwę planu oraz zapisuje go. Formą weryfikacji poprawności testu jest sprawdzenie czy plan użytkownika został zapisany w pamięci.



Rys. 7.2 - *Test integracyjny sprawdzający zapis planu treningowego [opracowanie własne]*

Listing 7.2 *Test integracyjny sprawdzający poprawność zapisania planu treningowego [opracowanie własne]*

```
describe('CreatePlan', () => {
    beforeEach(() => {
        useExerciseStore.mockReturnValue({
            fetchExercises: jest.fn(),
            pickedExercisesTemp: [
                { id: 1, name: 'Przysiad', reps: 5, sets: 3, weight: '' },
            ],
            setPickedExercisesTemp: jest.fn(),
        });

        AsyncStorage.getItem.mockResolvedValue(JSON.stringify([]));
    });

    test('użytkownik może stworzyć i zapisać nowy plan', async () => {
        const { getByPlaceholderText, getByText } = render(<CreatePlan />);

        fireEvent.changeText( getByPlaceholderText('Nazwa planu'), 'Przysiad');

        fireEvent.press(getByText('Zapisz plan'));

        await waitFor(() => {
            expect(AsyncStorage.setItem).toHaveBeenCalled();
        });
    });
});
```

8. Testy sprzętowe

Testy sprzętowe zostały przeprowadzone w celu sprawdzenia poprawności działania aplikacji mobilnej w określonym środowisku sprzętowym oraz oceny jej stabilności i wydajności. Ich głównym celem było zweryfikowanie, czy aplikacja działa poprawnie na urządzeniu mobilnym o określonych parametrach technicznych oraz czy interfejs użytkownika zachowuje czytelność i responsywność podczas codziennego użytkowania.

Testy zostały wykonane z wykorzystaniem emulatora urządzenia mobilnego Nexus 5X, dostępnego w środowisku Android Studio. Emulator ten reprezentuje urządzenie ze średniej półki sprzętowej, co pozwala na realistyczne odwzorowanie warunków pracy aplikacji na typowym telefonie użytkownika. Zastosowanie emulatora pozwoliło na dostęp do kontrolowanego środowiska testowego oraz łatwe powtarzanie scenariuszy testowych bez konieczności użycia urządzenia fizycznego.

Zakres przeprowadzonych testów obejmował:

- Uruchamianie i zamykanie aplikacji,
- Nawigacje między ekranami,
- Korzystanie z mapy mięśni,
- Tworzenie planów treningowych,
- Przeglądanie bazy ćwiczeń,
- Eksportowanie oraz importowanie danych.

Dodatkowo zweryfikowano poprawność wyświetlania interfejsu oraz czytelność tekstu na ekranie emulatora.

Przeprowadzone testy potwierdziły poprawne działanie aplikacji w środowisku testowym. Aplikacja uruchamiała się bez błędów, a wszystkie funkcjonalności działały zgodnie z założeniami. Nawigacja między ekranami była płynna, a interfejs użytkownika reagował na interakcje szybko i płynnie.

9. Podsumowanie i wnioski

W ramach pracy inżynierskiej zrealizowano mobilną aplikację umożliwiającą użytkownikowi tworzenie, edytowanie oraz zarządzanie własnymi planami treningowymi. Zgodnie z założeniami przedstawionymi w rozdziale drugim, wszystkie wymagania funkcjonalne i niefunkcjonalne zostały spełnione.

Implementację wykonano przy użyciu stosu technologicznego omówionego w rozdziale trzecim. Aplikacja została napisana z wykorzystaniem języka JavaScript oraz frameworku React Native, natomiast logika przechowywania danych opiera się na formacie JSON oraz komunikacji realizowanej przez REST API i bibliotekę Axios. Wszystkie podstawowe elementy systemu działają zgodnie z założeniami i zostały pomyślnie przetestowane.

W ramach realizacji pracy skupiono się na implementacji kluczowych funkcjonalności oraz spełnienia wszystkich wymagań funkcjonalnych. Część bardziej zaawansowanych funkcjonalności została celowo poza zakresem niniejszej pracy jako potencjalny kierunek dalszego rozwoju.

W przyszłości projekt może zostać rozwinięty o te funkcjonalności:

- Bardziej rozbudowana baza ćwiczeń;
- Zaawansowana wersja modelu wizualizacji mięśni (bardziej szczegółowa wersja lub model 3D);
- Większa liczba opcji podczas tworzenia planu treningowego;
- Połączenie z zewnętrznym API;
- Napisanie aplikacji serwerowej oraz przeniesienie bazy danych z formatu JSON;
- Implementacja kont użytkowników;

Podsumowując, przygotowana aplikacja w pełni realizuje założone cele oraz spełnia wszystkie wymagania i jest gotowa do wprowadzenia na rynek dla użytkowników.

Literatura

[1] Kirupa Chinnathambi, *"JavaScript. Przewodnik dla absolutnie początkujących"*

[2] Douglas Crockford, *"JavaScript - mocne strony"*

[3] Javier Cuello, José Vittone *"Designing Mobile Apps"*

[4] Bonnie Eisenman, *"React Native. Tworzenie aplikacji mobilnych w języku JavaScript"*

[5] Vladimir Khorikov, *"Unit Testing: Principles, Practices and Patterns"*

[6] Jakob Nielsen *"Usability Engineering"*

[7] JSON [online] - Dostęp 05.11.2025

https://www.json.org/json-en.html

[8] React Native [online] - Dostęp 05.11.2025

https://reactnative.dev/

[9] React Navigation [online] - Dostęp 05.11.2025

https://reactnavigation.org/

[10] Stack Overflow, Developer Survey Results 2024 [online] - Dostęp 05.11.2025

https://survey.stackoverflow.co/2024

[11] Saurabh Kumar, "Difference Between Thick and Thin Client" [Online] - Dostęp 26.11.2025

https://itjunction.org/2019/10/05/what-is-the-difference-between-thick-client-and-thin-client/

[12] IBM - "Deciding what level of availability you need" [online] - Dostęp 09.01.2026

https://www.ibm.com/docs/en/i/7.6.0?topic=roadmap-deciding-what-level-availability-you-need

Spis rysunków

Spis listingów