Państwowa Wyższa Szkoła Zawodowa w Tarnowie

Wydział Politechniczny

Kierunek: Informatyka

Specjalność: Informatyka stosowana

Specjalizacja: Inżynieria systemów informatycznych

2021/2022

Artur Hamernik

PRACA INŻYNIERSKA

Elektroniczny trener personalny - „Iron Muscle”

Promotor pracy:

mgr inż. Tomasz Gądek

Tarnów, 2022

Spis treści

1. Wstęp

W sklepach internetowych możemy znaleźć niezliczoną liczbę aplikacji treningowych. Nie mamy jednak pewności czy treningi i ćwiczenia w nich przedstawiane, są dla nas odpowiednie. Mogę nawet stwierdzić, że w takich aplikacjach bądź na stronach z treningami możemy znaleźć wiele ćwiczeń, które nie tylko nam nie pomogą poprawić naszej sylwetki, ale mogą nam również zaszkodzić. Często szkody wyrządzone przez takie ćwiczenia muszą być naprawiane podczas specjalistycznych rehabilitacji. Pandemia COVID-19 ograniczyła nam dostęp do siłowni. To ważne jednak abyśmy pozostali aktywni. Aktywność fizyczna pozytywnie wpływa na: nastrój, odporność, podnosi samoocenę, poprawia sen, zwiększa wydolność fizyczną oraz siłę mięśni. Światowa Organizacja Zdrowia zaleca od 150 do 300 minut aktywności fizycznej o umiarkowanej intensywności oraz od 75 do 150 minut aktywności o dużej intensywności tygodniowo.

Na rynku istnieje realne zapotrzebowanie na aplikację z łatwo dostępnymi treningami, które możemy wykonywać w dowolnym miejscu. Nie musimy inwestować w specjalne przyrządy do ćwiczeń, które kosztują fortunę i zajmują dużo miejsca. Większa część społeczeństwa nie posiada wykształcenia w kierunku fitness. Pomimo tego, że znamy swoje ciało, to ułożenie planu treningowego dostosowanego do naszych potrzeb i możliwości nie jest takie proste. Aplikacja rozwiązuje ten problem, udostępniając nam treningi ułożone przez profesjonalnych trenerów personalnych. Ćwiczenia wchodzące w skład treningów zaopatrzone są w filmy instruktażowe, które pomogą użytkownikom wykonywać je we właściwy sposób. Zaimplementowany został system śledzenia naszego postępu, z możliwością sprawdzenia jak sobie radzimy na tle innych użytkowników. Widoczne postępy i duch rywalizacji zmotywują użytkowników do dalszej aktywności fizycznej.

1.1. Cel pracy

Przedmiotem niniejszej pracy inżynierskiej jest zaprojektowanie oraz implementacja systemu, którego zadaniem będzie dostarczenie użytkownikowi szerokiego zasobu treningów. Treningi zostaną podzielone na standardowe oraz dostosowane do potrzeb użytkownika. Użytkownicy w systemie powinni zostać podzieleni na odpowiednie grupy: klienci, trenerzy, administratorzy. Każda z tych grup musi posiadać inny interfejs, a co za tym idzie dostęp do innych funkcji w systemie. System powinien posiadać elementy grywalizacji, tj. ranking oraz odznaki, które użytkownicy mogą odblokować za pomocą punktów, otrzymanych po wykonaniu treningu. System jest zobowiązany do śledzenia postępu użytkowników i wyświetlania statystyk. Następnym zadaniem aplikacji jest umożliwienie trenerom tworzenie treningów ogólnodostępnych i spersonalizowanych.

1.2. Zakres pracy

W skład pracy inżynierskiej wchodzą:

- aplikacja serwerowa,
- baza danych,
- aplikacja mobilna,
- aplikacja internetowa.

2. Opis wykorzystanych technologii

2.1. Java

Java to oparty na klasach, obiektowy język programowania [3]. Pierwszy projekt tego języka został utworzony przez Jamesa Goslinga, Mike Sheridana i Patricka Naughtona w czerwcu 1991 roku. Pierwsza publiczna implementacja Javy została wydana w 1996 roku przez firmę Sun Microsystems pod nazwą Java 1.0. Główne koncepcje Javy zawierają:

- obiektowość,
- dziedziczenie,
- niezależność od architektury,
- sieciowość i obsługa programowania rozproszonego,
- niezawodność i bezpieczeństwo.

2.2. Spring Boot

Spring Boot ułatwia tworzenie samodzielnych aplikacji. Powstał na bazie kodu z książki „Design and Development" autorstwa Roda Johnsona. Pierwsze wydanie ukazało się w październiku 2002 roku. Jest to najpopularniejszy framework dla Javy. Znacznie usprawnia pracę nad projektem, ponieważ zapewnia wiele usprawnień, których implementacja zajęłaby zbyt wiele czasu. Jedną z jego zalet jest kontener aplikacji, który pozwala na łatwe uruchomienie programu [13]. Główne cechy frameworka Spring:

- możliwość tworzenia aplikacji „standalone",
- wbudowany Tomcat, Jetty lub Undertow,
- zapewnienie startowych zależności w celu uproszczenia konfiguracji,
- automatyczna konfiguracja zależności Springa oraz bibliotek firm trzecich,
- zapewnienie gotowych metryk, statusów kondycji i zewnętrznej konfiguracji,
- brak generowania kodu i potrzeby konfiguracji XML.

2.3. Spring Security

Spring Security to silny i szeroko konfigurowalny framework służący do uwierzytelniania i kontroli dostępu [15]. Jest standardem zabezpieczania aplikacji opartych o framework Spring. Zapewnia ochronę przed atakami takimi jak utrwalanie sesji, porywanie kliknięć (ang. clickjacking), fałszowanie żądań między witrynami.

2.4. JavaScript

JavaScript to skryptowy język programowania wydany przez firmę Netscape. Jego najczęściej spotykanym zastosowaniem są strony internetowe. Skrypty te najczęściej służą do zapewnienia interakcji z użytkownikiem poprzez reagowanie na zdarzenia, walidację danych wprowadzanych w formularzach oraz tworzeniu zaawansowanych efektów wizualnych. JavaScript jednak to nie tylko skrypty stosowane po stronie klienta, można go spotkać również w postaci Node.js [9]. To wieloplatformowe środowisko uruchomieniowe o otwartym kodzie pozwala programistom na pisanie aplikacji serwerowych. Początki Node.js sięgają 2009 roku, liderem projektu był Ryan Dahl. Pierwsze wersje obsługiwały tylko Linuksa oraz Mac OS X. W 2011 roku Microsoft i Joyent wdrożyły natywna wersje Node.js dla systemu Windows. Pierwotnym celem Dahla było utworzenie technologii push, którą można spotkać w aplikacjach pocztowych. Po wypróbowaniu wielu języków autor zdecydował się na JavaScript. Z powodu braku API wejścia/wyjścia Dahl zaimplementował nieblokujące sterowane zdarzeniami API wejścia/wyjścia. Specyfikacja jest nazywana ECMAScript lub ES. Szósta wersja JavaScript (ES6) wydana w 2015 roku zawiera prawie dwa tuziny nowych funkcji. JavaScript napisany w tej wersji znacznie różni się od wersji ES5 [1]. W ostatnich latach język doczekuje się corocznych stabilnych wersji. ECMAScript 1 do 6 są w pełni wspierane przez przeglądarki internetowe.

2.5. PostgreSQL

PostgreSQL to jeden z najpopularniejszych systemów zarządzania relacyjnymi bazami danych. Opracowany w 1995 roku na Uniwersytecie Kalifornijskim w Berkeley i opublikowany pod nazwą Ingres [10]. W PostgreSQL domyślnie możemy używać funkcji składowanych takich jak PL/pgSQL, PL/Python, PL/Perl, PL/Tcl oraz z rozszerzeń, które należy doinstalować. W PostgreSQL zaimplementowano wiele rodzajów indeksów m.in. B-tree, Hash, GiST, SP-GiST oraz GIN. Indeksy mogą zostać utworzone na kolumnie, kolumnach lub widoku zmaterializowanym. Zawiera również mechanizmy wyzwalaczy, które uruchamiane są przed lub po określonej operacji na tabeli. W PostgreSQL jest zaimplementowany mechanizm MVCC do zarządzania transakcjami. Posiada również reguły, czyli elementy aktywne. Za ich pomocą można np. zrealizować widoki modyfikujące. Użytkownicy mogą definiować własne typy danych.

2.6. React.js

React to biblioteka języka JavaScript wykorzystywana do tworzenia interfejsów graficznych aplikacji internetowych [1]. Głównym pomysłodawcą i projektantem był programista firmy Facebook (obecnie Meta) - Jordan Walke. Autor React.js był zainspirowany przez rozszerzenie języka PHP o nazwie XHP. Jedną z wyróżniających cech tej biblioteki jest wirtualny DOM (ang. Document Object Model). React przechowuje DOM aplikacji w swojej pamięci. Po zmianie stanu w programie znajduje różnice pomiędzy wirtualnym a rzeczywistym DOM i wprowadza zmiany. Drugą cechą jest język JSX. Jest to nakładka JavaScript pozwalająca zapisywać kod HTML i XML oraz komponenty Reacta wewnątrz kodu JavaScript.

2.7. React Native

React Native to otwarto-źródłowy zestaw narzędzi przeznaczony do tworzenia wieloplatformowych aplikacji mobilnych, internetowych oraz telewizorowych [11]. Został zaimplementowany przez firmę Facebook i opiera się o te same biblioteki co React.js. W 2020 roku React Native był najchętniej wybieranym zestawem narzędzi do tworzenia aplikacji mobilnych.

2.8. Spring Data JPA (Java Persistence API)

JPA (ang. Java Persistence API) jest standardem mapowania obiektowo-relacyjnego dla języka programowania Java [14]. Pozwala on programiście operować na obiektach zwanymi encjami. Zapisywać wyniki wykonywanych operacji do bazy danych za pomocą obiektu EntityManager. Za pomocą adnotacji lub dokumentów XML obiekty są przekładane na elementy w bazie danych. JPA definiuje również język zapytań o nazwie JPQL (ang. Java Persistence Query Language), który swoją składnią przypomina SQL.

2.9. Hibernate

Hibernate to framework służący do realizacji warstwy dostępu do danych [2]. Zapewnia translację z relacyjnej bazy danych na obiekty stosowane w Javie. Zwiększa wydajność operacji na bazie danych, dzięki minimalizacji przesyłanych zapytań i buforowaniu. Głównym projektantem jest Gavin King, który również pracuje przy JPA.

2.10. HTTP & HTTPS

HTTP (ang. Hypertext Transfer Protocol) jest to protokół przesyłania dokumentów hipertekstowych przez sieć. W znormalizowany sposób pozwala na komunikację urządzeń przez sieć. Określa formę żądań klienta oraz formę odpowiedzi serwera. W tabeli 2.2 zostały przedstawione najpopularniejsze formy żądań HTTP. Zalicza się do tzw. protokołów bezstanowych (ang. stateless), ponieważ po poprzednich transakcjach nie są przechowywane żadne dane. Takie rozwiązanie znacznie zmniejsza obciążenie serwera. Problematyczna może być sytuacja, jeżeli konkretny stan użytkownika musi być zapamiętany. Rozwiązaniem tego problemu są ciasteczka (ang. cookies), wysyłane z serwera i przechowywane po stronie klienta.

HTTPS (ang. Hypertext Transfer Protocol Secure) jest szyfrowaną wersją HTTP. Został utworzony w celu zabezpieczenia przesyłanych danych przed potencjalnymi atakami. Przykładowym atakiem, na który jest podatny HTTP to przechwycenie i podmiana danych w drodze pomiędzy klientem a serwerem. W protokole HTTPS stosuje się protokół szyfrowania TLS, który zapobiega takim atakom [8].

GET	Pobranie zasobu wskazanego przez URI.
POST	Przyjęcie danych przesyłanych od klienta do serwera.
PUT	Przyjęcie danych przesyłanych od klienta do serwera, najczęściej w celu aktualizacji zasobu.
DELETE	Żądanie usunięcia zasobu.

Tabela 2.1. Formy żądań HTTP (opracowanie własne)

2.11. JSON

JSON (JavaScript Object Notation) to luźny format wymiany danych. JSON jest formatem tekstowym bazującym na podzbiorze języka JavaScript [4]. Format został opisany w dokumencie RFC 4626. Pomimo specyficznej nazwy jest on niezależny od konkretnego języka. Wiele języków programowania obsługuje format JSON. Komunikat JSON to literał obiektu w JavaScripcie. Wszystkie dane są zmiennymi, a ich nazwy otoczone są cudzysłowami. Wartości mogą być typu string (otoczone cudzysłowem), numeryczne, logiczne lub null. Tablice i obiekty w JSON mogą być zagnieżdżone. Komunikat jest kodowany najczęściej za pomocą UTF-8.

```json
{
  "boolean": false,
  "text": "string",
  "integer": 1,
  "double": 0.5,
  "object": {
    "param1": true,
    "param2": 1,
    "param3": "object"
  },
  "arrayOfIntegers": [
    1,
    2,
    3
  ],
  "arrayOfStrings": [
    "one",
    "two",
    "three"
  ],
  "arrayOfObjects": [
    {
      "key": 1,
      "value": "one"
    },
    {
      "key": 2,
      "value": "two"
    }
  ]
}
```

Listing 2.1. Przykład obiektu JSON (opracowanie własne)

2.12. REST

REST (ang. Representational State Transfer) jest to styl architektury dla systemów rozproszonych [12]. Poniżej zostały wymienione główne reguły architektury typu REST:

- Jednolity interfejs komunikacyjny, serwer powinien udostępniać API, które będzie zrozumiałe przez wszystkie aplikacje komunikujące się z nim.
- Podział na aplikacje klient-serwer zwiększa możliwości skalowania, rozszerzalności i przenośności. Ułatwia integrację z usługami klienckimi w przyszłości i pozwala na niezależny rozwój i działanie.
- Bezstanowość oznacza, że na serwerze nie powinny znajdować się żadne mechanizmy przechowujące dane klienta, które byłoby konieczne dla poprawnej pracy systemu. Oznacza to również, że użytkownik musiałby przesyłać swoje dane w żądaniu do

serwera, np. w nagłówku jako token JWT. Na jego podstawie API może stwierdzić, czy żądanie powinno być przetworzone lub odrzucone.

- Cache danych, dzięki któremu API może obsługiwać duży ruch, pozwala na zapamiętywanie odpowiedzi w przeglądarce.
- Odseparowanie warstw. Komunikacja pomiędzy aplikacjami klienckimi a serwerem nie powinna zawierać informacji o serwisach, z jakich korzysta serwer.
- Opcjonalną regułą architektury REST jest wysłanie kodu do aplikacji klienta, np. skryptów JavaScript gotowych do przetworzenia i uruchomienia.

2.13. CRUD

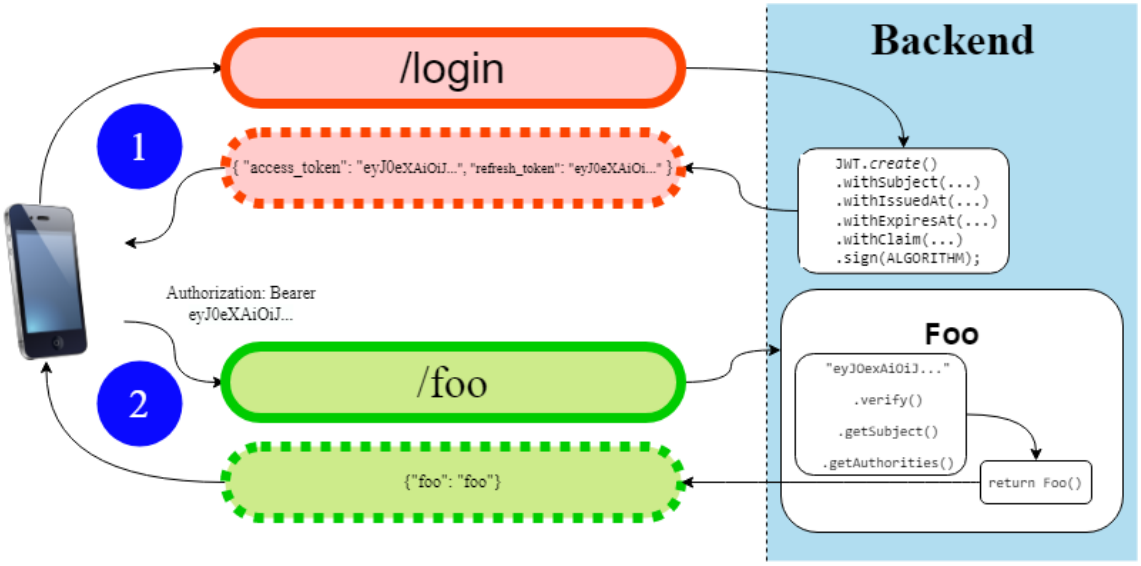
CRUD (ang. Create Read Update Delete) są to cztery podstawowe funkcje w aplikacjach korzystających z pamięci trwałej [12]. Każdy człon tego skrótu może być wykonany poprzez podstawowe operacje bazodanowe oraz polecenia HTTP.

Instrukcja SQL	**HTTP**
INSERT	POST
SELECT	GET
UPDATE	PUT
DELETE	DELETE

Tabela 2.2. Odzwierciedlenie instrukcji SQL w HTTP (opracowanie własne)

2.14. JWT

JWT (JSON Web Token) jest otwartym standardem przesyłania zabezpieczonych informacji jako obiekt JSON pomiędzy dwoma aplikacjami. Przesłana informacja może być zweryfikowana i uwierzytelniona, ponieważ jest cyfrowo podpisana. JWT może być podpisany za pomocą sekretu (algorytmu HMAC) lub pary kluczy publiczny/prywatny, używając RSA lub ECDSA [7].



Rysunek 2.1. Autoryzacja za pomocą JWT (opracowanie własne)

2.15. JUnit

JUnit jest narzędziem zaprojektowanym przez Ericha Gamma oraz Kenta Becka. Jego początki sięgają drugiej połowy 1995 roku. W obecnym czasie JUnit jest standardem testowania jednostkowego aplikacji napisanych w Javie [6]. Najnowsza wersja JUnit 5 składa się z kilku modułów pochodzących od trzech różnych projektów. W jego skład wchodzą: JUnit Platform, JUnit Jupiter i JUnit Vintage.

JUnit Platform służy jako podstawa do uruchamiania narzędzi testowych na wirtualnej maszynie Java (JVM). Posiada konsolę do uruchomienia platformy z wiersza poleceń oraz JUnit Platform Suite Engine do uruchamiania testów używając jednego lub więcej silników. Wsparcie dla JUnit Platform możemy znaleźć w popularnych IDE takich jak: Intellij IDEA, Eclipse, NetBeans i Visual Studio Code oraz w narzędziach do budowania: Gradle, Maven i Ant.

JUnit Jupiter jest kombinacją modelu programowania i rozszerzenia do pisania testów i rozszerzeń JUnit 5. Podprojekt udostępnia TestEngine do uruchamiania testów bazujących na Jupiterze na platformie.

JUnit Vintage udostępnia TestEngine do uruchamiania testów JUnit 3 i JUnit 4 na platformie. Wymaga JUnit w wersji 4.12 lub wyższej [5].

3. Dokumentacja techniczna

W poniższym rozdziale przedstawiona zostanie architektura, moduły oraz aktorzy systemu. Przeanalizowany zostanie sposób, w jaki osiągnięto bezpieczeństwo aplikacji.

3.1. Funkcje systemu

W poniższym podrozdziale zostaną omówione funkcje, które są spełniane przez poszczególne części systemu. Wszystkie poniżej opisane moduły zostały zrealizowane zgodnie ze wstępnymi założeniami.

3.1.1. Aplikacja mobilna

Głównym założeniem projektu było zaimplementowanie responsywnej aplikacji mobilnej na smartfony oraz tablety. Aplikacja umożliwia użytkownikowi niezalogowanemu podstawowe operacje takie jak utworzenie konta oraz odzyskanie hasła. Każdy użytkownik zalogowany może zmienić mail powiązany z kontem, hasło lub zdjęcie profilowe. Z uwagi na to, że użytkownicy podzieleni są na trzy grupy. Każda z tych grup ma inne możliwości.

Zwykli użytkownicy, czyli klienci, mają dostęp do wszystkich treningów standardowych oraz treningów spersonalizowanych utworzonych na ich życzenie. Użytkownicy mogą przeglądać ćwiczenia wchodzące w skład treningów oraz zmieniać ich kolejność przed rozpoczęciem sesji treningowej. Animacje instruktażowe pomogą użytkownikowi poprawnie wykonać odpowiedni zestaw ćwiczeń. Użytkownik może zatwierdzać wykonanie ćwiczenia polegającego na ilości powtórzeń oraz zatrzymać lub wznowić jego przebieg. Użytkownik może również przewijać ćwiczenia. Po ukończeniu treningu aplikacja wyświetla użytkownikowi podsumowanie. Każdy skończony trening nagradzany jest punktami. Punkty mogą być zamieniane na odznaki. Aplikacja pozwala na przeglądanie historii treningów, rankingu użytkowników oraz własnych odznak. Dodatkowo aplikacja pozwala na tworzenie zgłoszeń z prośbą o ułożenie treningu personalnego.

Trenerzy mogą przeglądać nowe zgłoszenia i je obsługiwać. Kompletują treningi ćwiczeniami z dostępnej puli. Następnie muszą określić czy ćwiczenie powinno być wykonane w określonej liczbie powtórzeń lub przez określony czas. Aplikacja pozwala na zmianę kolejności ćwiczeń oraz daje możliwość powielania ich w treningu. Trenerzy mają możliwość: przeglądać wszystkie treningi, edytować liczbę ćwiczeń w treningu, zmieniać liczbę powtórzeń oraz kontrolować czas wykonania ćwiczenia. Aplikacja daje możliwość tworzenia treningów standardowych, czyli dostępnych dla wszystkich użytkowników.

Program prezentuje administratorom następujące funkcje: przeglądanie listy użytkowników, blokowanie konkretnym usługobiorcom dostępu do serwisu oraz zakładanie kont dla użytkowników każdej roli. Głównym celem ostatniego modułu jest przygotowywanie kont dla nowych trenerów w systemie „Iron Muscle".

3.1.2. Aplikacja internetowa

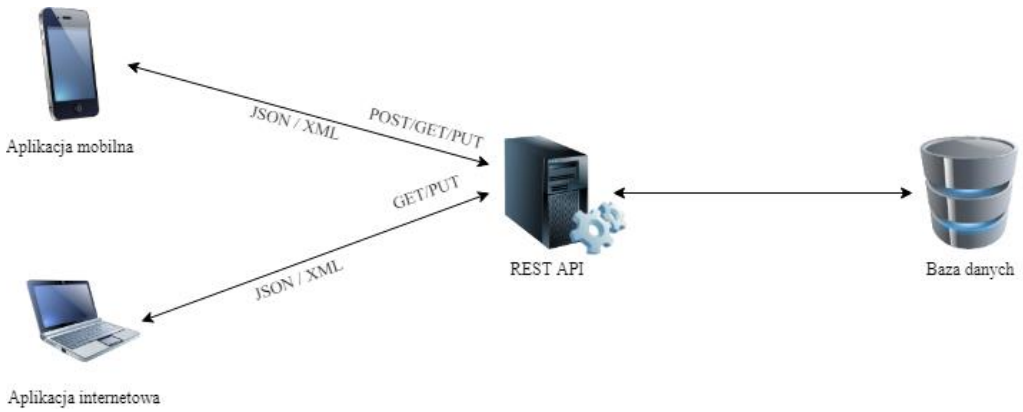
Aplikacja internetowa służy do aktywacji nowo utworzonych kont oraz zmiany zapomnianego hasła. Po udanej rejestracji serwer generuje e-mail z linkiem, który przekieruje użytkownika do strony internetowej systemu. Po otworzeniu strony przez adresata za pomocą wygenerowanego łącza zostanie wysłane zapytanie do serwera z żetonem zaszytym w odsyłaczu, które potwierdzi utworzenie konta. Aplikacja internetowa posiada również moduł umożliwiający zmianę hasła, podobnie jak przy potwierdzeniu e-maila w linku zaszyty jest token, który wysłany razem z nowym hasłem podanym w aplikacji internetowej pozwoli na zmianę hasła do konta. Ostatnią funkcją jest inicjalizacja konta założonego przez administratora. Po zatwierdzeniu formularza przez administratora na e-mail zostanie wysłany odnośnik, który pozwala na potwierdzenie rejestracji i ustawienie pierwszego hasła.

3.1.3. Aplikacja serwerowa

Aplikacja serwerowa obsługuje zapytania HTTP wysyłane przez aplikacje klienckie. Po nagłówku „Authorization" rozpoznaje jaki użytkownik wykonuje daną akcję. Służy jako medium pomiędzy aplikacjami klienckimi a bazą danych. Przechowuje konfigurację serwera SMTP, który służy do wysyłania e-maili do użytkowników. Ogranicza dostęp do zasobów zależnie od roli użytkownika wywołującego zapytanie. Przetwarza zapytania i aplikuje zmiany do bazy danych.

3.2. Architektura systemu

Na rysunku nr 3.1 została przedstawiona architektura systemu elektronicznego trenera personalnego. REST API udostępnia aplikacji mobilnej i internetowej operacje CRUD na bazie danych. W ramach komunikacji pomiędzy urządzeniami o charakterze klienta a serwerem przesyłane są dane w formacie JSON. System jest podzielony na aplikacje klient-serwer, co sprawiło, że programy wchodzące w skład tej architektury mogły być samodzielnie rozwijane i mogą samodzielnie działać. Poszczególne komponenty architektury zostanę omówione w podrozdziałach 3.2.1, 3.2.2 i 3.2.3.



Rysunek 3.1. Architektura systemu (opracowanie własne)

3.2.1. Aplikacja mobilna

Aplikacja na urządzenia mobilne jest najlepszym sposobem implementacji ogólnodostępnego systemu treningowego z kilku powodów. Po pierwsze, w dzisiejszym świecie prawie każdy posiada smartfon lub inne urządzenie przenośne z dostępem do Internetu. Po drugie, czas do włączenia aplikacji śledzącej treningi na takim urządzeniu jest znacznie mniejszy niż włączenie aplikacji internetowej na komputerze osobistym lub laptopie. Po trzecie, przy rozwoju takiej aplikacji umiejscowienie jej na urządzeniu mobilnym pozwala na połączenie z opaską śledzącą tętno itp.

Aplikację mobilną zaimplementowano przy użyciu narzędzia React Native. Pozwala ono na wdrażanie aplikacji na urządzenia zarówno z systemem Android, jak i iOS. Z powodu braku możliwości testowania aplikacji na urządzeniu Apple bądź jego emulatorze aplikacja została napisana na urządzenia z systemem Android. Aplikacja przetestowano pod względem responsywności zarówno na emulatorach smartfonów, jak i tabletów, które cechują się większą przekątną ekranu.

Do komunikacji HTTP z serwerem został wykorzystany oparty na obietnicy (ang. promise) klient Axios. Pozwala on na zaimplementowanie interceptorów zapytania i odpowiedzi, które zostaną omówione w rozdziale pt. „Bezpieczeństwo systemu".

Routing i nawigacja zostały zrealizowane przy pomocy React Native Navigation. Biblioteka pomaga zaprojektować strukturę aplikacji. Wymaga React Native w wersji 0.63.0 lub wyższej.

3.2.2. Aplikacja serwerowa (REST API)

Serwer aplikacji został zaimplementowany przy użyciu narzędzia Spring Boot. Encje reprezentują tabele przechowywane w relacyjnej bazie danych. Encje są używane w celu ułatwienia wykonywania logiki biznesowej na danych. Dzięki encjom możemy zarządzać wieloma zależnymi obiektami podczas wykonywania skomplikowanych zadań. Repozytoria pozwalają na wykonywanie zapytań do bazy danych. Spring Data bada metody zaimplementowane w interfejsie repozytorium w celu zrozumienia ich w kontekście obiektu. Przy pomocy konwencji nazewnictwa możemy obsłużyć proste zapytania, do bardziej złożonych używamy adnotacji „@Query". W serwisach znajdują się metody operujące na encjach i repozytoriach. Serwisy wstrzykiwane są do kontrolerów, gdzie ich metody są udostępnione pod odpowiednimi zakończeniami usługi sieciowej (ang. endpoints) i dostępnymi z aplikacji mobilnej lub internetowej. Serwer zapewnia również bezpieczeństwo danych, które zostanie omówione w rozdziale pt. „Bezpieczeństwo systemu". Poszczególne moduły serwera zostaną przedstawione w rozdziale pt. „Moduły systemu". Działanie serwera jest bezstanowe, co oznacza, że nie znajdziemy na serwerze mechanizmów przetrzymujących dane klienta, które byłyby potrzebne do poprawnego działania systemu. Klient zawsze będzie musiał przesłać komplet informacji razem z żądaniem w celu jego poprawnego przetworzenia.

3.2.3. Baza danych

Baza danych została zaprojektowana tak, aby była zgodna z pierwszą, drugą i trzecią postacią normalną. Baza używa silnika PostgreSQL. Jest on bardziej zaawansowany niż np. MySQL i daje więcej możliwości. Problemem może się okazać szybkość odczytu danych, która w przypadku aplikacji mobilnych powinna być priorytetem. W bazie znajdują się proste typy danych takie jak bigint, integer, varchar, boolean i timestamp.

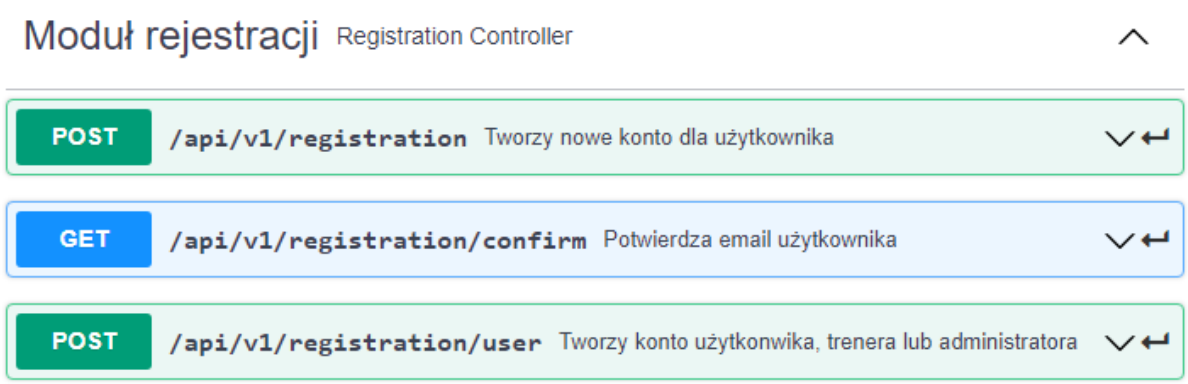
3.3. Bezpieczeństwo systemu

Przed przystąpieniem do implementacji została przeprowadzona analiza potrzeb i funkcjonalności, z której wynikało, że prawa do konkretnych funkcji i zasobów systemu będą przydzielane na podstawie roli użytkownika. W systemie znajdują się cztery role: użytkownik niezalogowany, użytkownik zalogowany, trener oraz administrator. W celu zabezpieczenia systemu i pomocy w autoryzacji określonych akcji został wykorzystany standard JWT (ang. JSON Web Token). Użytkownik po udanym uwierzytelnieniu otrzymuje token dostępu i odnowienia dostępu. Każdy z tych żetonów posiada zaszyfrowane informacje o tożsamości użytkownika, jego roli oraz dokładnej dacie uzyskania i wygaśnięcia tokenu. Żetony są szyfrowane za pomocą sekretu przechowywanego w lokalnej konfiguracji serwera, algorytmem HMAC. W konfiguracji serwera każdy punkt końcowy usługi sieci Web posiada określone role użytkowników, którzy mają do niego dostęp. Aplikacja mobilna w momencie tworzenia zapytania HTTP dołącza token dostępu do nagłówka. Filtr autoryzacji został zaimplementowany po stronie serwera. Przechwytuje każde zapytanie niebędące prośbą o uwierzytelnienie, przypomnienie hasła lub rejestrację. Weryfikuje token sekretem i algorytmem HMAC. Jeśli weryfikacja się powiedzie, określa tożsamość użytkownika proszącego o uwierzytelnienie i sprawdza czy ma odpowiednie uprawnienia. W przeciwnym razie zapytanie zostanie odrzucone. Bada również, czy token jest wciąż ważny. Po wygaśnięciu ważności zwraca kod błędu 401. Po otrzymaniu takiego kodu błędu interceptor po stronie aplikacji mobilnej wysyła zapytanie odnawiające token dostępu przy użyciu drugiego żetonu, który uzyskała podczas operacji uwierzytelnienia. Jeśli otrzyma nowy token dostępu, to ponowi zapytanie. Jeśli token odnowienia dostępu wygasł, to użytkownik zostanie wylogowany. Aplikacja zapisuje żetony za pomocą Async Storage i odczytuje je w momencie wysyłania zapytania, w celu dodania ich do nagłówka. Po wylogowaniu tokeny są usuwane z Async Storage.

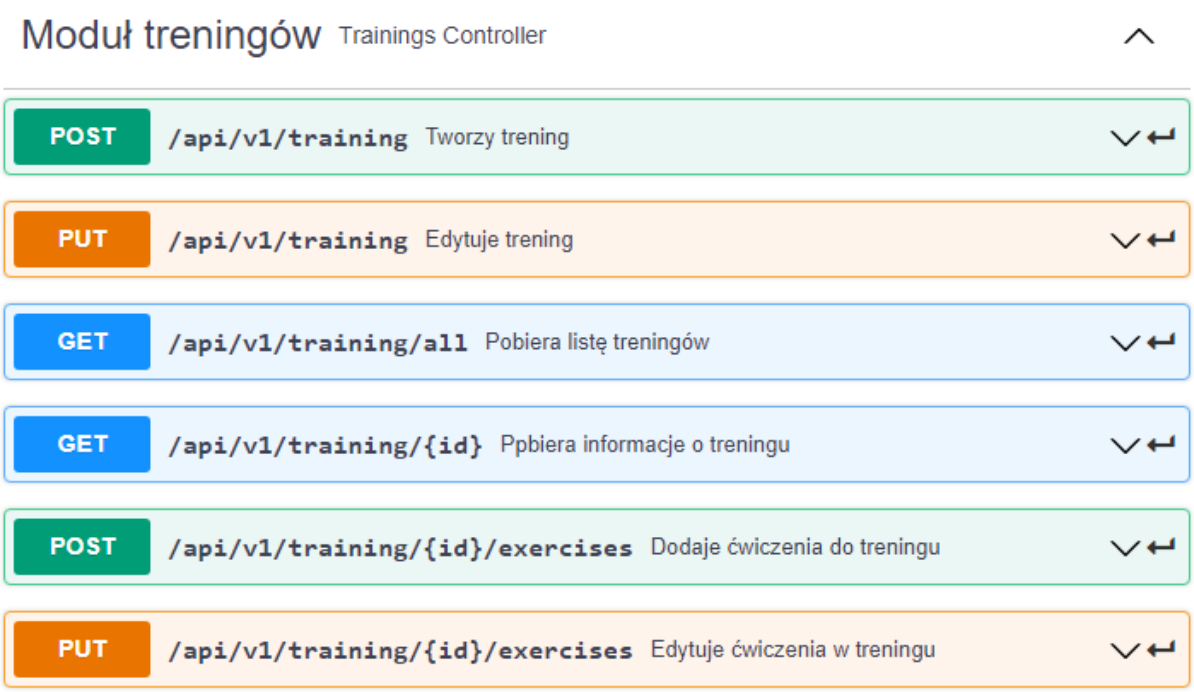
Następnymi środkami bezpieczeństwa wprowadzonymi w systemie są „confirmation_token” i „password_token”. Po rejestracji użytkownika jest tworzony specjalny token i zapisywany w bazie danych. Ten token ma określoną datę wygaśnięcia, ale również datę potwierdzenia. Podobny token jest tworzony podczas resetu hasła. Po potwierdzeniu e-maila lub zmianie hasła stają się nieaktywne. Żetony mają na celu upewnienie się, że dana akcja zakończy się sukcesem tylko raz. Zapobiega to sytuacji, w której ktoś mógłby wejść w posiadanie czyjegoś linku do zmiany hasła i użyć go ponownie pozbawiając kogoś dostępu do konta.

Ostatnim środkiem bezpieczeństwa jest szyfrowanie haseł użytkowników w bazie danych. Szyfrowanie jest wykonywane za pomocą klasy BCryptPasswordEncoder.

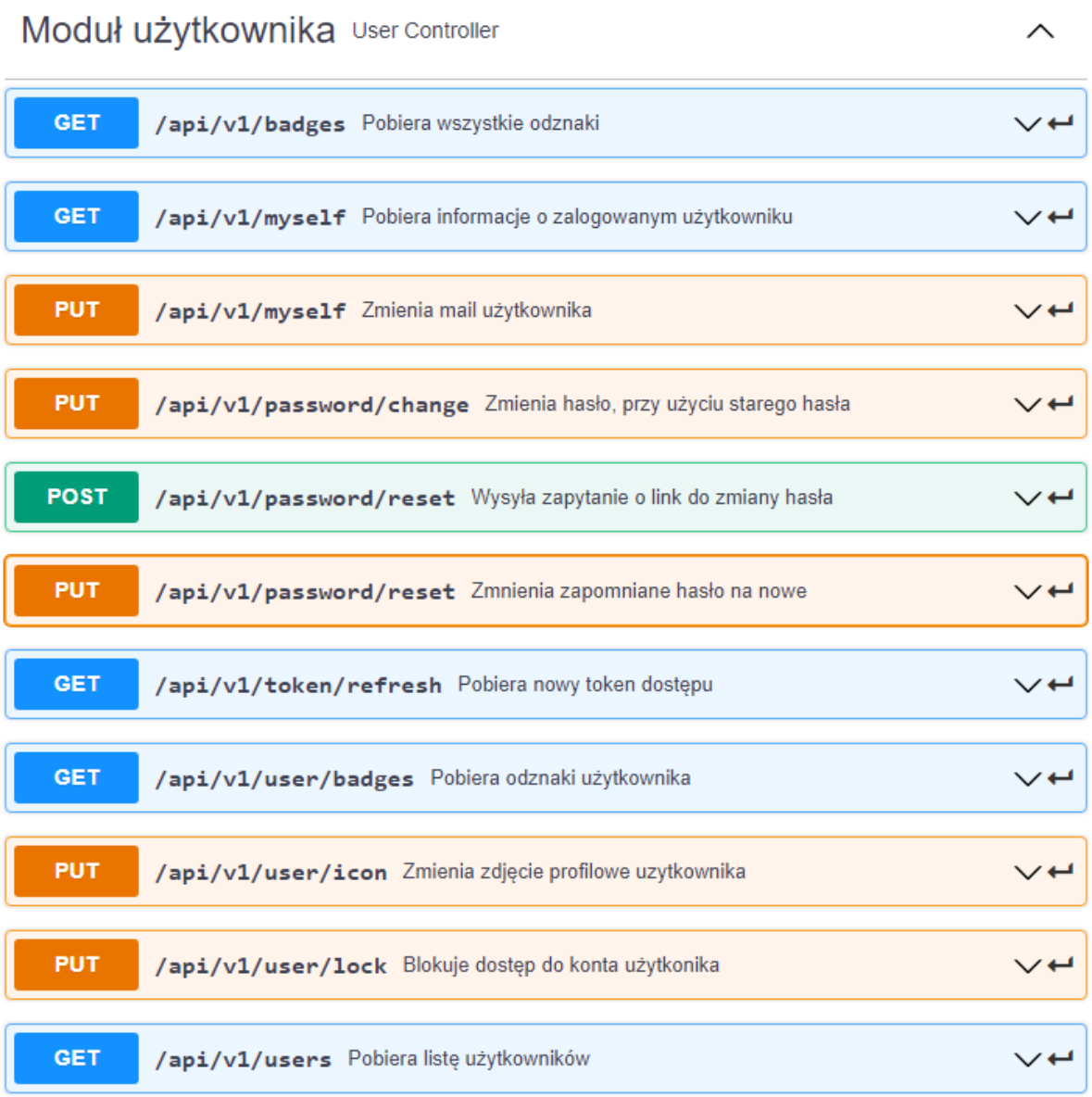
3.4. Moduły systemu



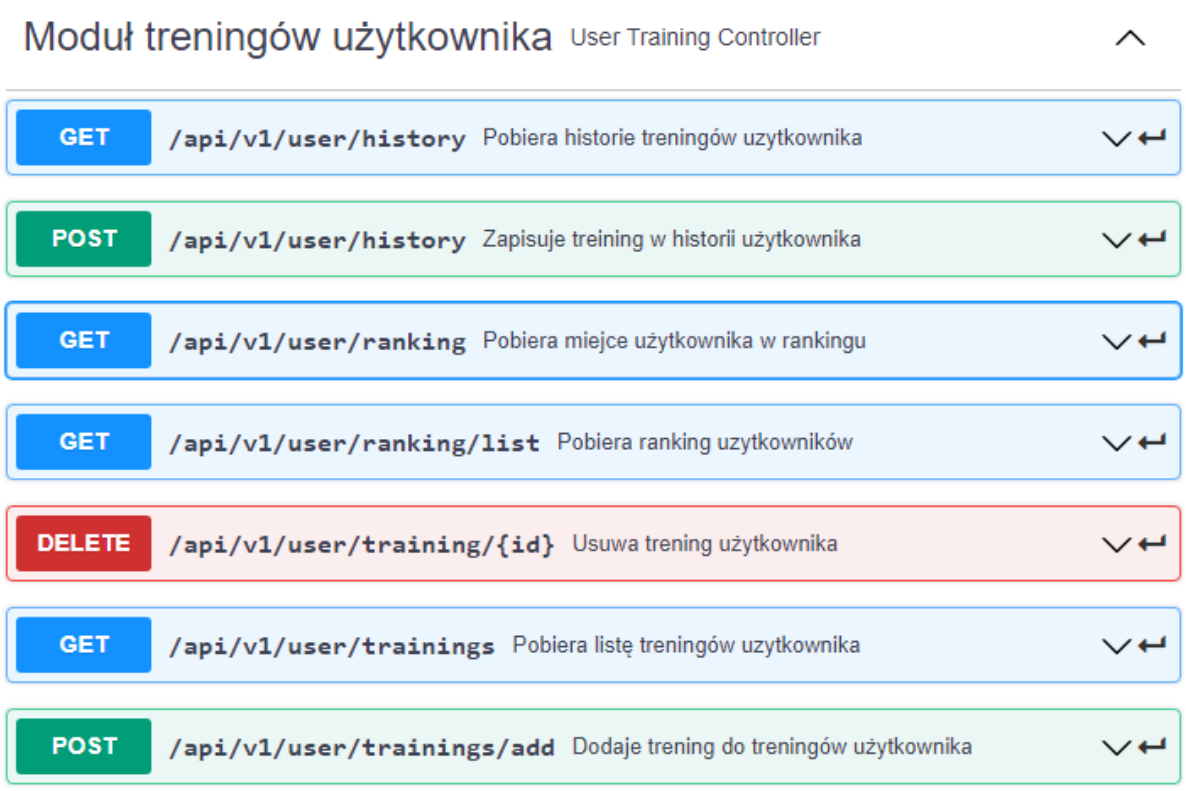
Rysunek 3.2. Moduł rejestracji (Dokumentacja SwaggerHub)



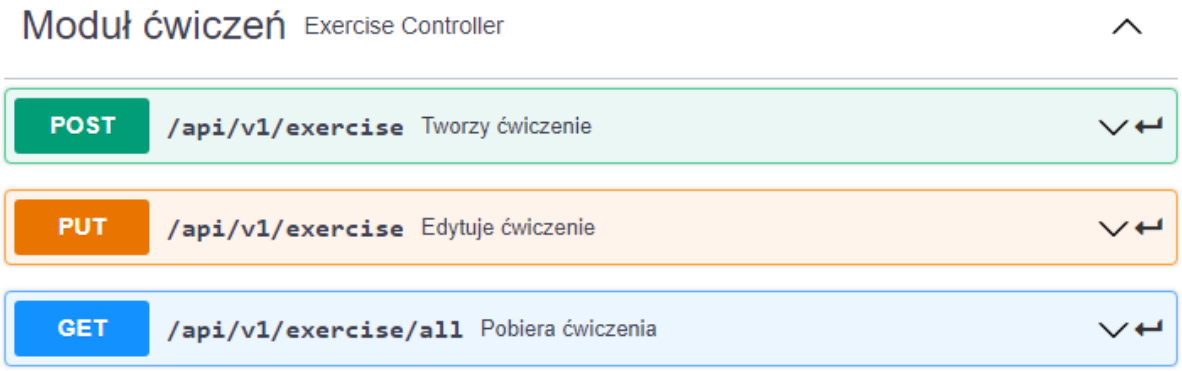
Rysunek 3.3. Moduł treningów (Dokumentacja SwaggerHub)



Rysunek 3.4. Moduł użytkownika (Dokumentacja SwaggerHub)



Rysunek 3.5. Moduł treningów użytkownika (Dokumentacja SwaggerHub)



Rysunek 3.6. Moduł ćwiczeń (Dokumentacja SwaggerHub)

Rysunek 3.7. Moduł zgłoszeń (Dokumentacja SwaggerHub)

3.5. Aktorzy systemu

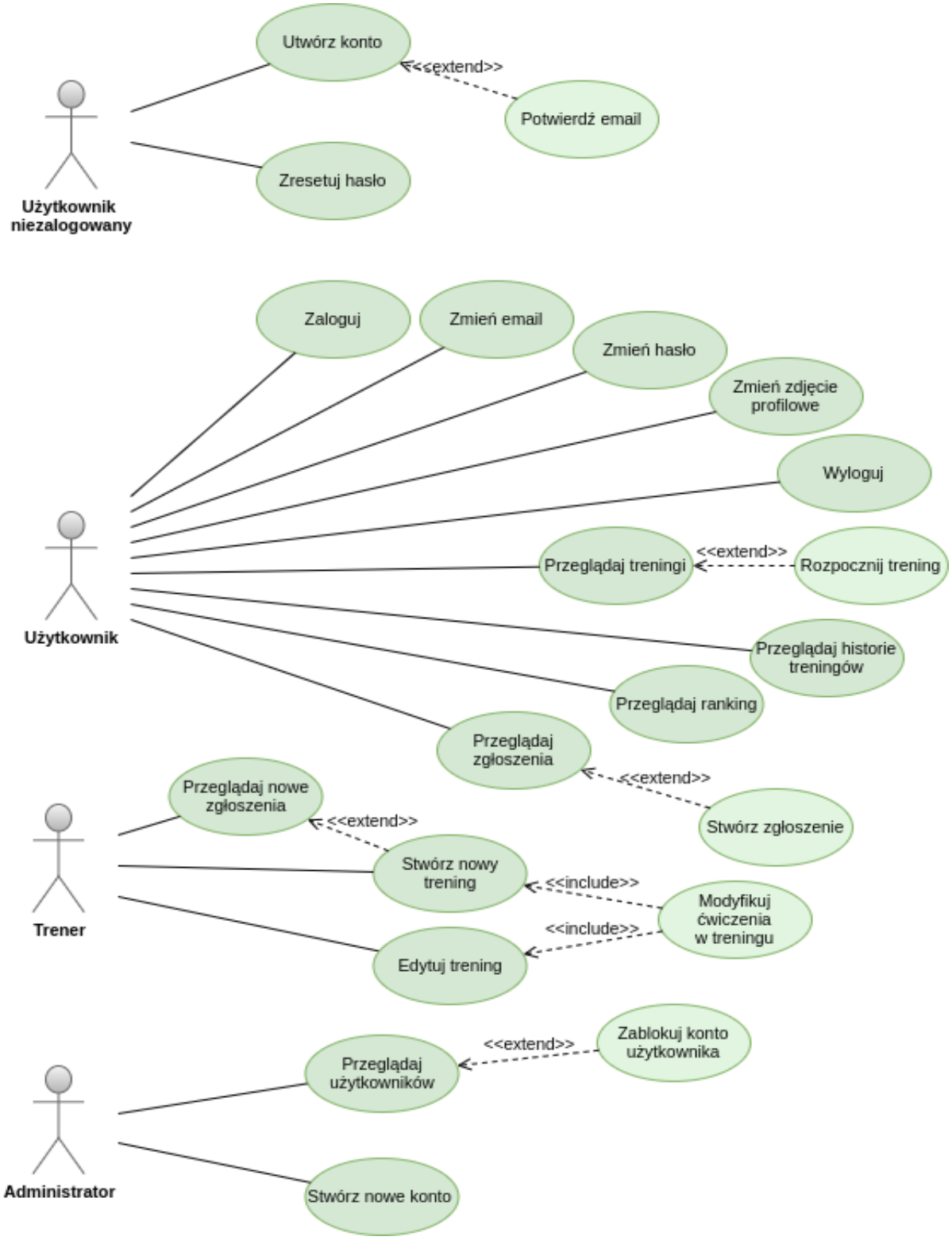
W systemie znajdują się następujący aktorzy:

- użytkownik niezalogowany,
- użytkownik zalogowany,
- trener,
- administrator.

4. Diagram ERD

Rysunek 4.1. Diagram ERD (opracowanie własne)

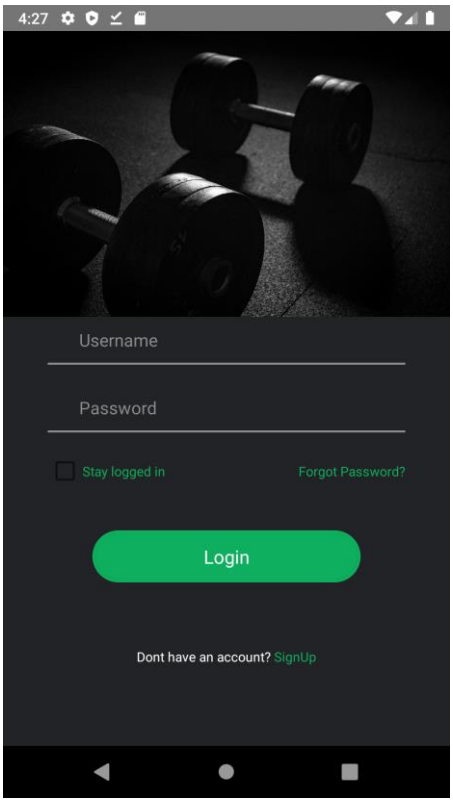
5. Diagram przypadków użycia



Rysunek 5.1. Diagram przypadków użycia (opracowanie własne)

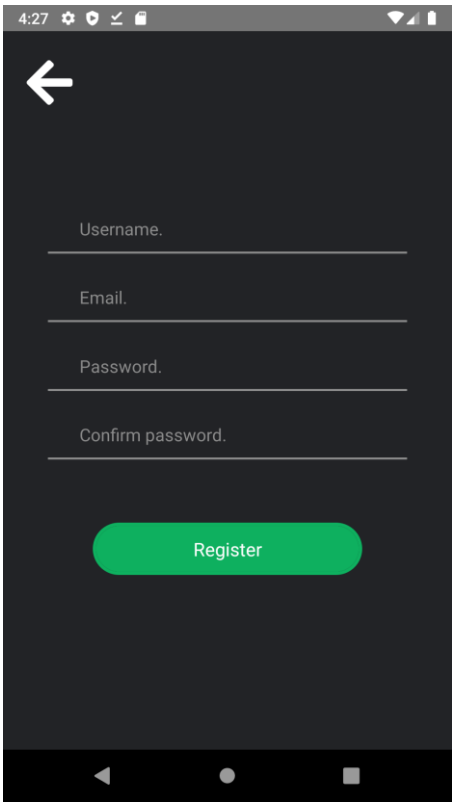
6. Interfejs użytkownika

W poniższym rozdziale omówię interfejs użytkownika. Zostaną przedstawione możliwe operacje na poszczególnych ekranach aplikacji. Zrzuty ekranu zostały wykonane na emulatorze smartfona Google Pixel 2. Emulator został zainicjowany przy pomocy Android Studio w wersji 4.2.1.

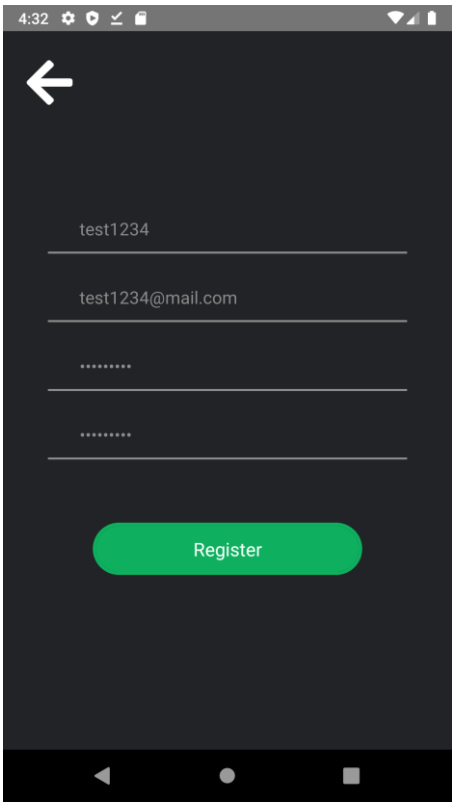


Rysunek 6.1. Ekran logowania (opracowanie własne)

Na rysunku 6.1 zaprezentowano ekran logowania, który jest również ekranem powitalnym. Jest to współdzielony widok dla wszystkich ról użytkowników. Po uzupełnieniu pól formularza i naciśnięciu przycisku „Login" użytkownik zostanie przeniesiony do ekranu głównego. Należy wziąć pod uwagę, że każda rola użytkownika (użytkownik, trener administrator) ma własny ekran główny. Aplikacja rozszyfrowuje token JWT i na podstawie zawartości pola „authorities" określa, do jakiego ekranu głównego powinna przenieść zalogowanego użytkownika. Pole do zaznaczenia (ang. checkbox) z podpisem „Stay logged in" sprawia, że dane logowania zostaną zapisane w Async Storage i po wyłączeniu aplikacji i ponownym uruchomieniu użytkownik zostanie zalogowany automatycznie. Za pomocą przycisku „Forgot password?" użytkownik zostanie przeniesiony do formularza pozwalającego zresetować hasło. Przycisk „Don't have an account? SingUp" otworzy formularz rejestracji.

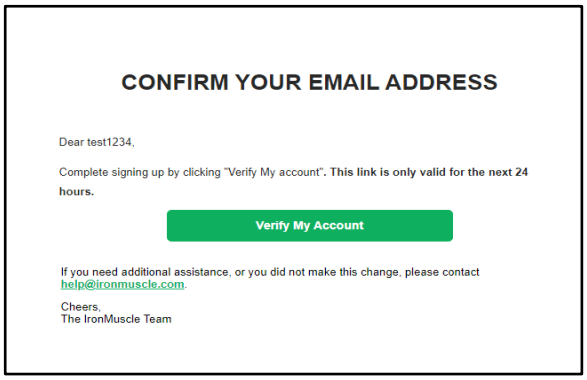


Rysunek 6.2. Ekran rejestracji (opracowanie własne)

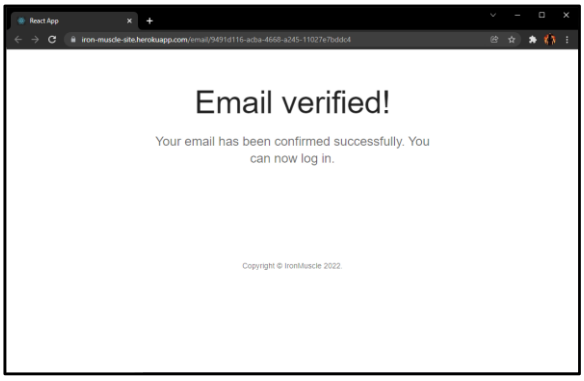


Rysunek 6.3. Ekran rejestracji uzupełniony (opracowanie własne)

Rysunki 6.2 i 6.3 przedstawiają ekran z formularzem rejestracji. W przypadku pozostawienia pustego pola lub błędnego uzupełnienia jednego z nich i zatwierdzenia formularza przyciskiem „Register” zostanie wyświetlony komunikat o błędzie. Jeśli formularz rejestracji jest poprawnie wypełniony, to konto zostanie utworzone, aplikacja wyświetli komunikat o sukcesie i przeniesie użytkownika do ekranu logowania. Przycisk „strzałki” przeniesie użytkownika do ekranu logowania bez wysyłania formularza.

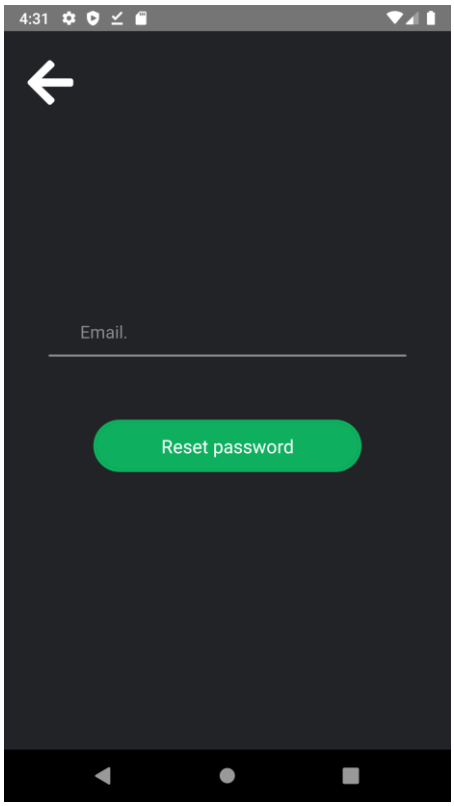


Rysunek 6.4. Wiadomość e-mail wysyłana po rejestracji (opracowanie własne)

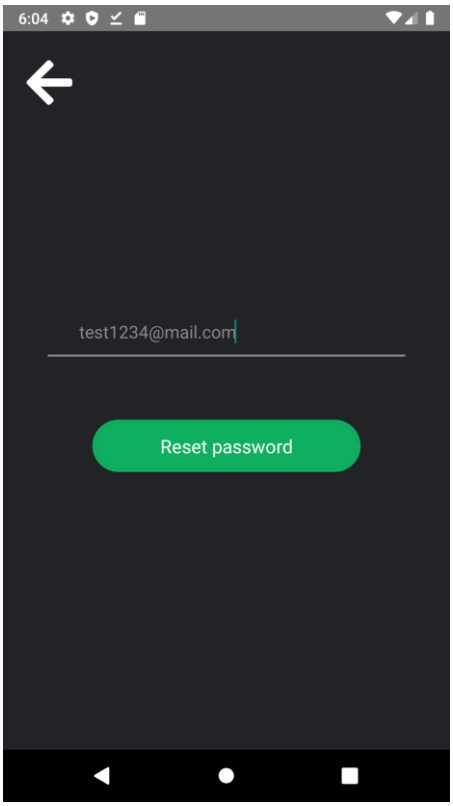


Rysunek 6.5. Widok strony potwierdzenia utworzenia konta (opracowanie własne)

Po utworzeniu konta na skrzynkę mailową adresu podanego przy rejestracji serwer wyśle e-maila z odsyłaczem w formie przycisku. Treść wiadomości jest widoczna na rysunku 6.4. Na rysunku 6.5 została przedstawiona strona aplikacji internetowej, do której odsyła link w wiadomości. Po otwarciu łącza wysyła zapytanie do serwera, które aktywuje konto użytkownika. W tym momencie użytkownik może się już zalogować i korzystać ze wszystkich oferowanych funkcji. Przeładowanie strony internetowej wywoła ponowne wysłanie zapytania, które tym razem zakończy się niepowodzeniem, ponieważ konto już zostało aktywowane. Otworzenie linku po upływie 24 godzin od wysłania wiadomości e-mail na skrzynkę pocztową również zakończy się niepowodzeniem aktywacji konta. Nieaktywne konta są systematycznie usuwane z bazy danych przez zaimplementowany na serwerze moduł.

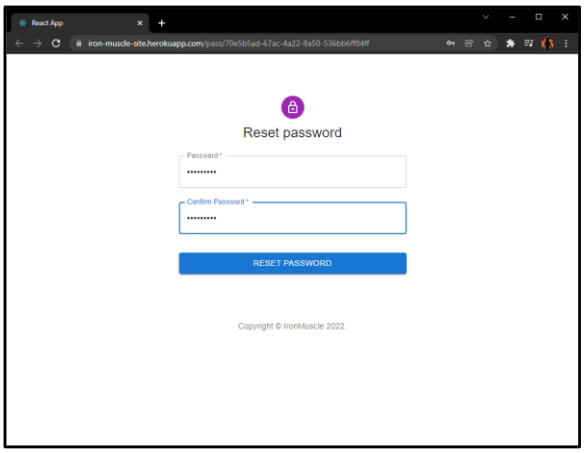


Rysunek 6.6. Ekran resetu hasła (opracowanie własne)

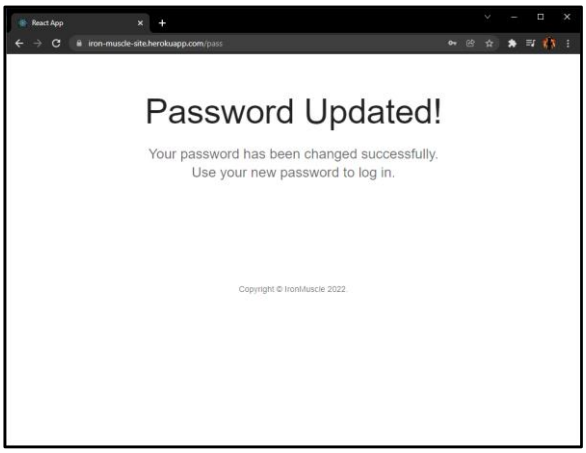


Rysunek 6.7. Ekran resetu hasła uzupełniony (opracowanie własne)

Rysunki 6.6 i 6.7 przedstawiają ekran zmiany hasła w przypadku, gdy użytkownik nie pamięta obecnego. Po wpisaniu adresu e-mail powiązanego z kontem w systemie i zatwierdzeniu przyciskiem „Reset password” zostanie wysłana wiadomość z linkiem prowadzącym do strony umożliwiającej zmianę hasła. Wiadomość nie zostanie wysłana, jeśli podany e-mail nie znajduje się w bazie. Zapobiega to niepotrzebnemu spamowi na skrzynkach pocztowych osób nieposiadających konta w systemie. Analogicznie do rejestracji „strzałka” przeniesie użytkownika do ekranu logowania bez wysyłania zapytania.

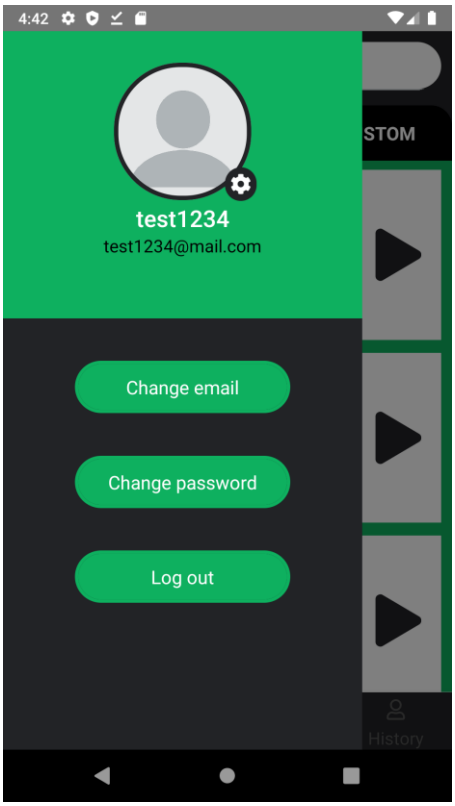


Rysunek 6.8. Widok strony resetu hasła uzupełniony (opracowanie własne)

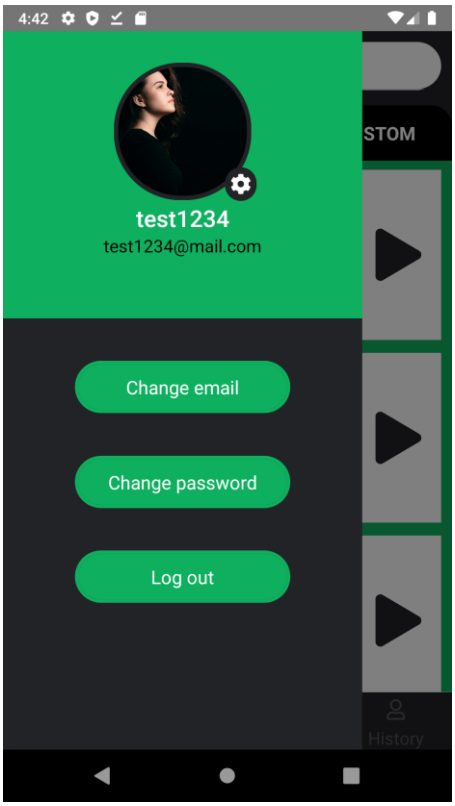


Rysunek 6.9. Widok strony potwierdzenia utworzenia konta (opracowanie własne)

Jeśli e-mail podany przez użytkownika (rysunek 6.7) istnieje w bazie, to na ten sam adres zostanie wysłana wiadomość. Po naciśnięciu przycisku w wiadomości zostanie otwarta strona zaprezentowana na rysunku 6.8. Tak jak w przypadku aktywacji konta link do zmiany hasła może zostać wykorzystany tylko raz. Komunikaty o błędach takich jak zatwierdzenie formularza bez potwierdzenia hasła lub zbyt słabe hasło zostaną wyświetlone w formie alertów. Po udanej zmianie hasła użytkownik będzie przekierowany do strony przedstawionej na rysunku 6.9. Jeśli użytkownik spróbuje ponownie przesłać formularz, to zobaczy stronę z komunikatem o błędzie.

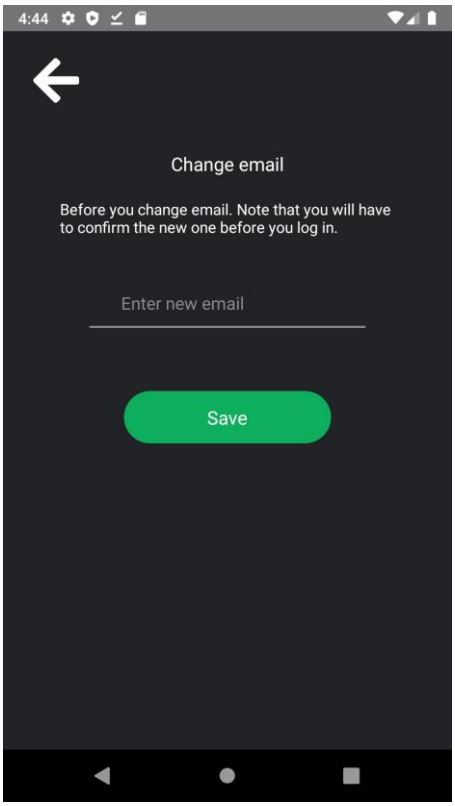


Rysunek 6.10. Ekran opcji użytkownika (opracowanie własne)

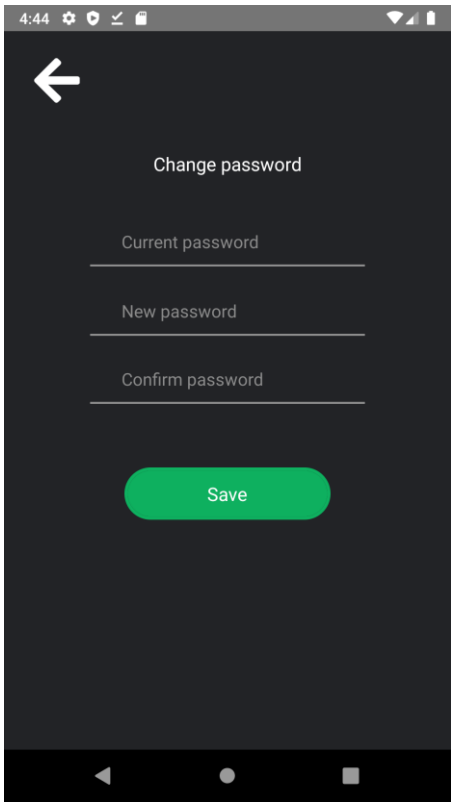


Rysunek 6.11. Ekran opcji użytkownika ze zmienionym zdjęciem profilowym (opracowanie własne)

Na rysunkach 6.10 i 6.11 został przedstawiony wysuwany ekran (ang. drawer screen), jest dostępny po zalogowaniu dla każdej roli, w każdym innym ekranie. Użytkownik musi wykonać gest przesunięcia od lewej krawędzi ekranu do prawej, aby wysunąć opcje. Naciskając ikonę „zębatki", aplikacja otworzy menadżer plików smartfona, który wyświetli zdjęcia zapisane w pamięci. Po wybraniu jednego zdjęcia aplikacja zmieni ikonę użytkownika na wybrane zdjęcie. Plik zostanie zapisany w specjalnym folderze po stronie serwera, a ścieżka do niego w encji danego użytkownika w bazie danych. Przyciski „Change email" oraz „Change password" otwierają odpowiednie formularze do zmiany właściwości profilu. Przycisk „Log out" przeniesie użytkownika do ekranu logowania oraz usunie z Async Storage token dostępu (ang. access token), odnowienia (ang. refresh token) oraz dane logowania, jeżeli użytkownik zaznaczył opcję „Keep me logged in" (pl. Pozostaw mnie zalogowanym). Przy następnym logowaniu aplikacja poprosi o login i hasło.



Rysunek 6.12. Ekran zmiany e-maila (opracowanie własne)

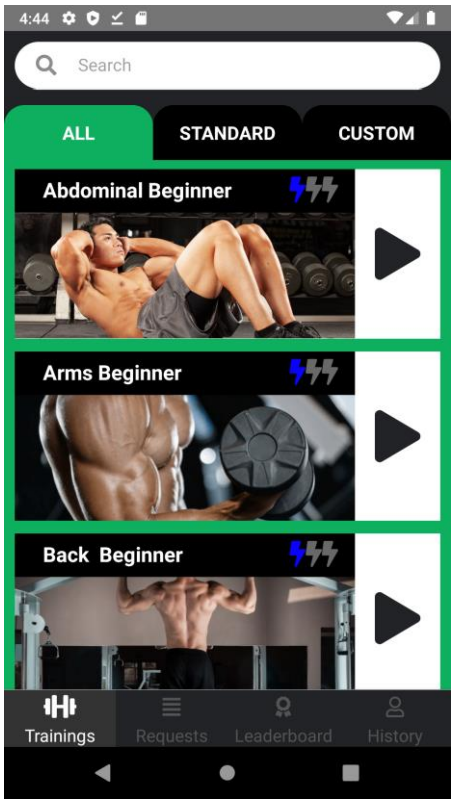


Rysunek 6.13. Ekran zmiany hasła (opracowanie własne)

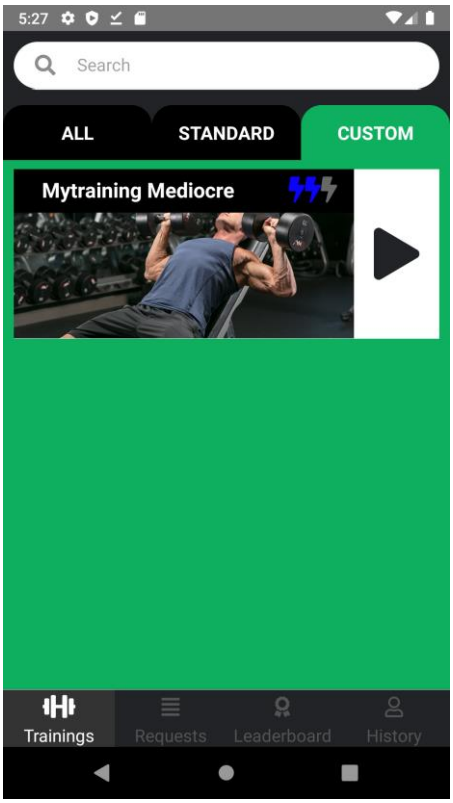
Rysunki 6.12 i 6.13 przedstawiają ekrany otwierane z menu bocznego przedstawionego na rysunku 6.10.

Ekran widoczny na rysunku 6.12 prezentuje formularz zmiany adresu e-mail. Po uzupełnieniu pustego pola i naciśnięciu przycisku „Save” adres powiązany z kontem zostanie zmieniony, o ile podany e-mail jest prawidłowy i nie należy do innego użytkownika. Następnie na ten e-mail zostanie wysłana wiadomość z prośbą aktywowania konta. Proces aktywacji wygląda tak, jak w przypadku potwierdzenia adresu po rejestracji. Z tą różnicą, że łącze pozostaje aktywne do momentu użycia. Nie posiada daty wygaśnięcia.

Na rysunku 6.13 widoczny jest ekran zmiany hasła. Użytkownik podaje stare hasło, nowe hasło i potwierdza je. Po wysłaniu formularza hasło zostanie zmienione, jeżeli następujące warunki będą spełnione: stare hasło będzie odpowiadało hasłu w bazie, nowe hasło będzie wystarczająco silne, a potwierdzenie hasła będzie prawidłowe.



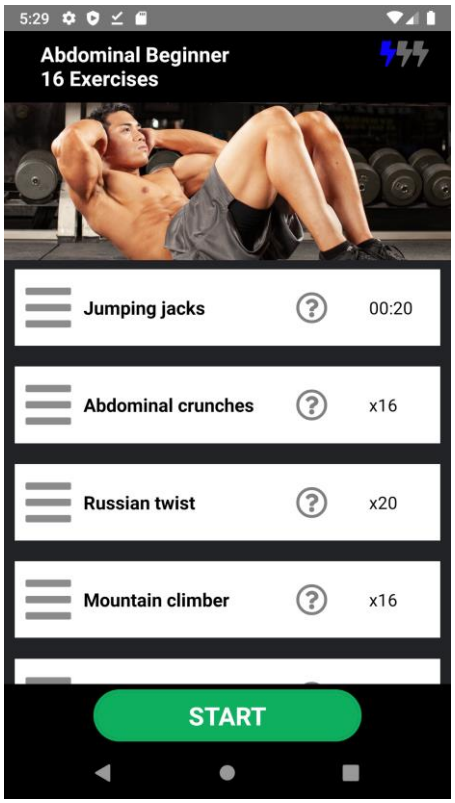
Rysunek 6.14. Ekran treningów „All” (opracowanie własne)



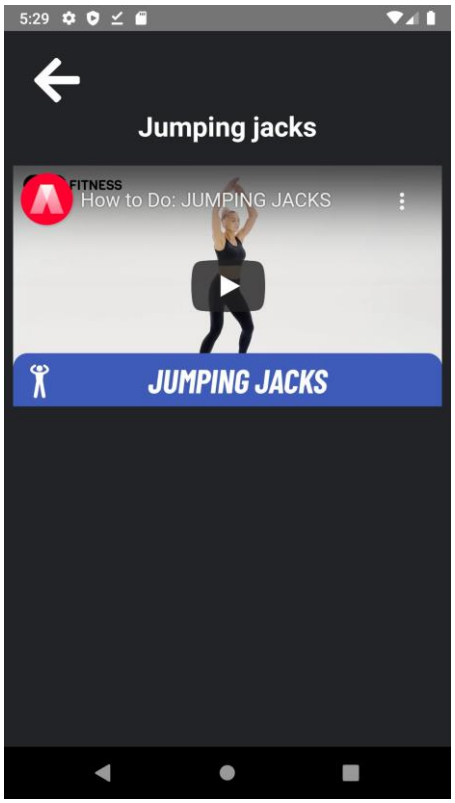
Rysunek 6.15. Ekran treningów „Custom” (opracowanie własne)

Na rysunkach 6.14 i 6.15 przedstawiono ekran treningów użytkownika. Pasek wyszukiwania służy do filtrowania treningów po nazwie i stopniu trudności. Zakładki „All”, „Standard” i „Custom” wyświetlają odpowiednio: wszystkie treningi, standardowe dostępne dla każdego użytkownika, treningi personalne danej osoby. Po naciśnięciu przycisku „Play” na karcie danego treningu aplikacja otworzy ekran z ćwiczeniami, który jest widoczny na rysunku 6.16.

Menu nawigacyjne służy do przemieszczania się po aplikacji. Analizując menu od lewej do prawej mamy przyciski do ekranu z treningami, zgłoszeniami, rankingiem oraz historią ćwiczeń.

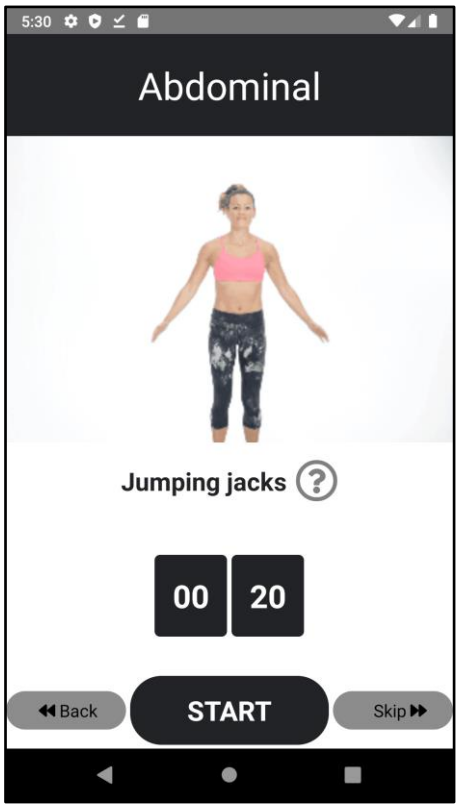


Rysunek 6.16. Ekran listy ćwiczeń (opracowanie własne)

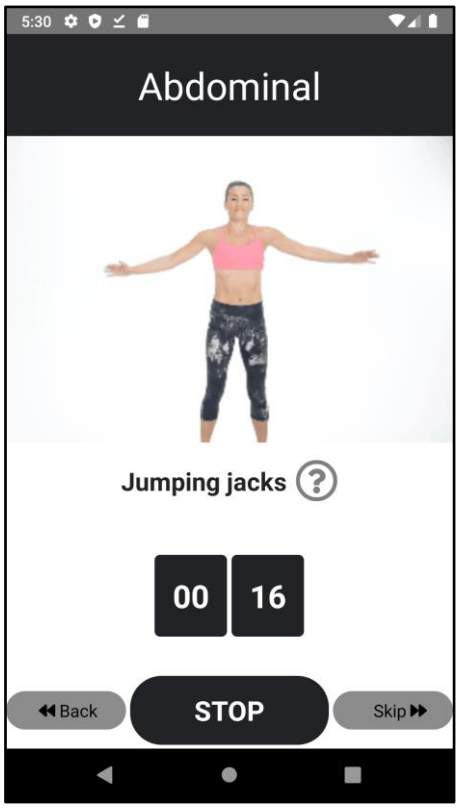


Rysunek 6.17. Ekran z odtwarzaczem YouTube (opracowanie własne)

Rysunek 6.16 przedstawia ekran ćwiczeń. Przy górnej krawędzi widzimy nazwę treningu i poziom trudności oraz liczbę ćwiczeń. W centrum znajduje się przewijana lista z ćwiczeniami. Użytkownik, przytrzymując ikonę „trzech pasków" i poruszając w górę lub w dół może zmienić kolejność ćwiczeń i na przykład przenieść „Jumping jacks" z pozycji pierwszej na trzecią. W tym momencie kolejność będzie wyglądać następująco: „Abdominal crunches", „Russian twist", „Jumping jacks" itd. Po naciśnięciu przycisku „pytajnika" aplikacja wyświetli modal pokazany na rysunku 6.17, na którym zostanie zaprezentowany filmik instruktażowy danego ćwiczenia. Po naciśnięciu „strzałki" użytkownik wróci do listy ćwiczeń. Po naciśnięciu „START" aplikacja włączy stoper liczący czas trwania gimnastyki i przeniesie użytkownika do ekranu zaprezentowanego na rysunku 6.18, który przeprowadzi go przez trening.

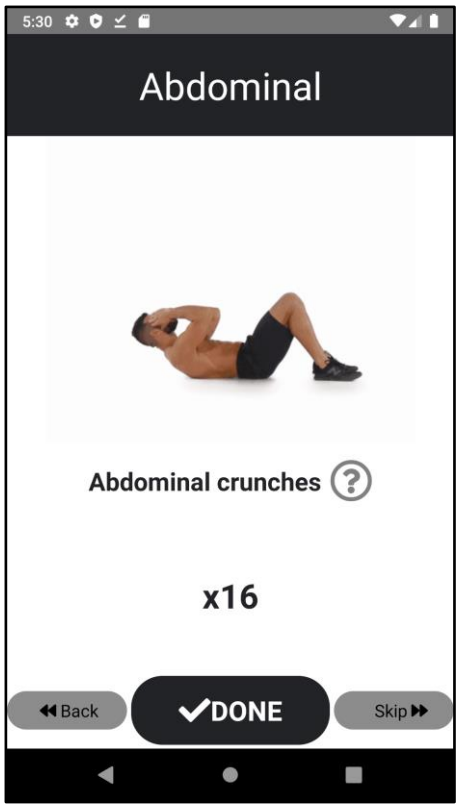


Rysunek 6.18. Ekran ćwiczenie czasowe „Start” (opracowanie własne)

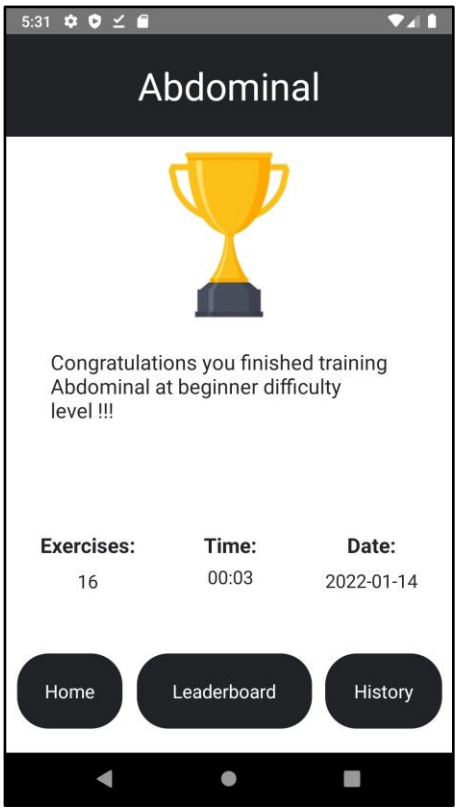


Rysunek 6.19. Ekran ćwiczenie czasowe „Stop” (opracowanie własne)

Na rysunkach 6.18, 6.19 i 6.20 został przedstawiony ekran widoczny po rozpoczęciu treningu aż do jego zakończenia. Prezentuje kolejne ćwiczenia danego treningu w kolejności ustalonej przez użytkownika na poprzednim widoku. Na środku znajduje się zapętlony gif, który ma na celu wizualizować ćwiczenie. Rysunki 6.18 i 6.19 przedstawiają ćwiczenie czasowe. Naciśnięcie „START” uruchomi odliczanie. Gdy czas dobiegnie końca, wyświetli się następne ćwiczenie. Przycisk „Pytajnika” otworzy okno modalne z odtwarzaczem. W momencie otwarcia odtwarzacza odliczanie zostanie przerwane i po jego zamknięciu wznowione w momencie przerwania. Przyciski „Back” i „Skip” służą do pomijania ćwiczeń.



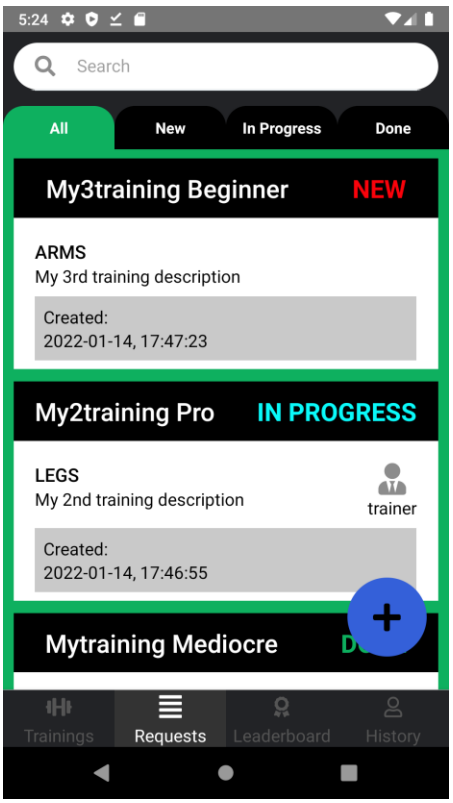
Rysunek 6.20. Ekran ćwiczenie powtórzenia (opracowanie własne)



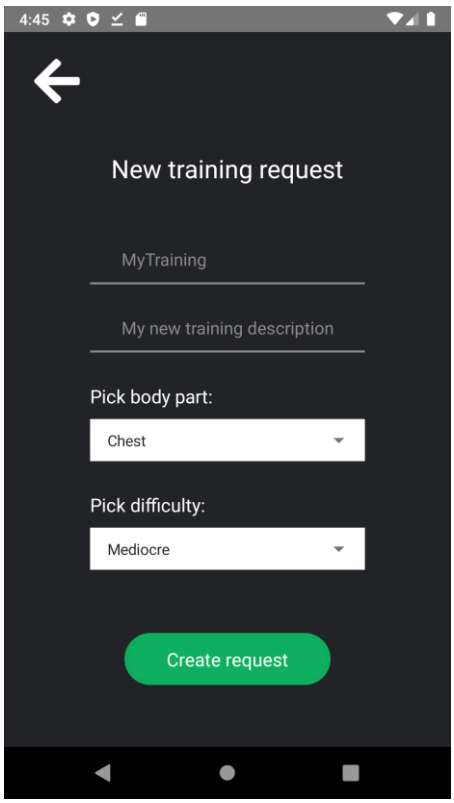
Rysunek 6.21. Ekran podsumowania (opracowanie własne)

Rysunek 6.20 przedstawia ekran wykonywania ćwiczenia. W tym przypadku użytkownik musi wykonać określoną liczbę powtórzeń. Po osiągnięciu celu powinien nacisnąć przycisk „DONE", który otworzy następne ćwiczenie. Gdyby użytkownik spróbował wyjść z aplikacji systemowym przyciskiem w kształcie trójkąta program, wyświetli okno z przyciskami „Yes" i „No", aby użytkownik potwierdził akcję, ponieważ jeśli opuści aplikację to trening zostanie przerwany i postępy nie będą zapisane.

Gdy wszystkie ćwiczenia zostaną wykonane, aplikacja wyświetli podsumowanie przedstawione na rysunku 6.21. Zostanie zapisana historia użytkownika, czas trwania treningu, data oraz nazwa i poziom trudności. Poniżej znajdują się przyciski, które po naciśnięciu przenoszą użytkownika do strony głównej, czyli listy treningów, rankingu i historii.



Rysunek 6.22. Ekran zgłoszeń (opracowanie własne)

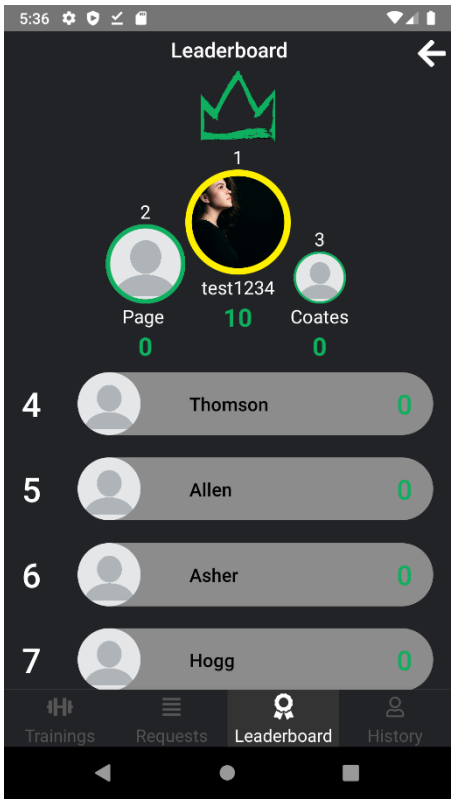


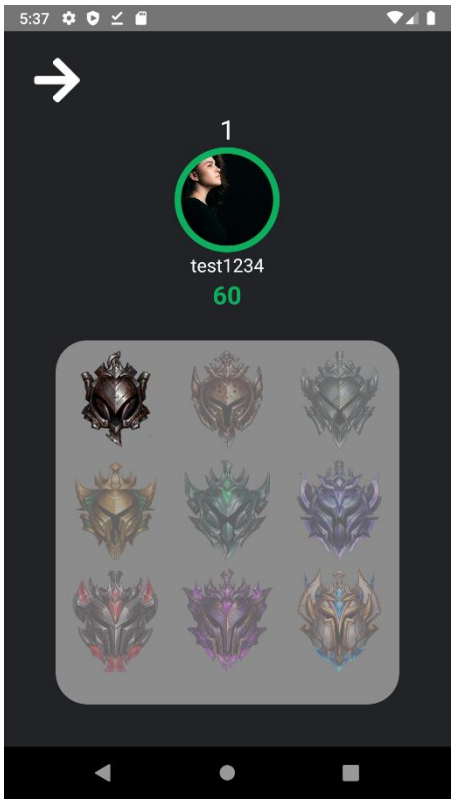
Rysunek 6.23. Ekran tworzenia zgłoszenia uzupełniony (opracowanie własne)

Ekran przedstawiony na rysunku 6.22 pokazuje listę zgłoszeń, które są swego rodzaju prośbami o utworzenie treningu personalnego. Wyżej widzimy pasek wyszukiwania służący do filtrowania zgłoszeń po tytule i trudności oraz zakładki do sortowania zgłoszeń po stanie. Występują następujące stany:

- „NEW” - zgłoszenie zostało utworzone,
- „IN PROGRESS” - trener rozpoczął przetwarzanie zgłoszenia,
- „DONE” - trener ukończył tworzenie treningu personalnego.

Na karcie zgłoszenia możemy zauważyć tytuł, poziom trudności, stan, część ciała, której dotyczyć ma trening, dodatkowe uwagi użytkownika, czas utworzenia oraz czas zamknięcia. Na zgłoszeniach o stanie innym niż „NEW” po prawej stronie widnieje nazwa trenera zajmującego się zgłoszeniem.



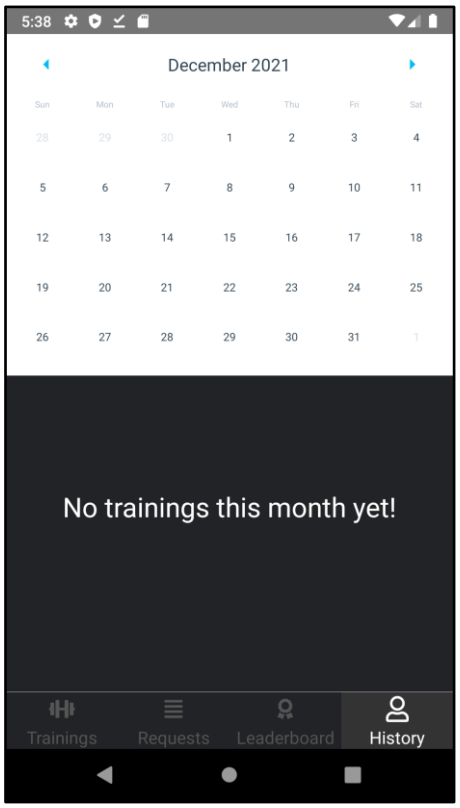


Rysunek 6.24. Ekran rankingu (opracowanie własne)

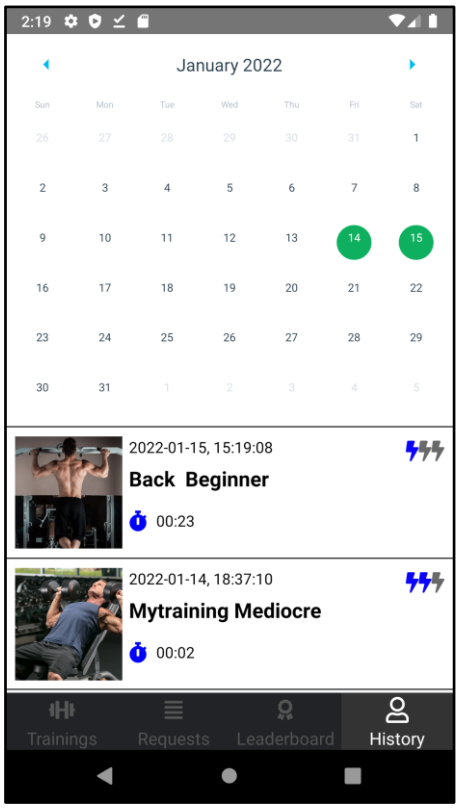
Rysunek 6.25. Ekran odznak (opracowanie własne)

Przycisk „+” otwiera okno modalne z formularzem do zgłoszenia zapotrzebowania na trening personalny, które zostało zaprezentowane na rysunku 6.23. Po uzupełnieniu pól tekstowych, wybraniu części ciała oraz poziomu trudności użytkownik zatwierdza utworzenie zgłoszenia przyciskiem „Create request”. Okno formularza się zamyka i ponownie otwiera listę zgłoszeń.

Rysunek 6.24 przedstawia ranking w formie listy. Wykonując gest przesunięcia od góry ekranu do dołu, użytkownik może odświeżyć ranking. Dla każdej pozycja w rankingu wyświetla następujące informacje: miejsce w rankingu, zdjęcie profilowe, nazwę użytkownika oraz liczbę punktów. Po naciśnięciu „strzałki” skierowanej w lewo aplikacja otworzy ekran z odznakami (rysunek 6.25). Na nim znajdują się informacje o miejscu w rankingu danego użytkownika oraz odznaki odblokowane i jeszcze niedostępne. Na rysunku 6.25 widzimy, że odblokowana jest tylko jedna odznaka. Naciskając odznakę, użytkownik wyświetli informację, ile punktów potrzebuje, żeby ja odblokować. Naciśnięcie „strzałki” skierowanej w prawo przeniesie użytkownika z powrotem do rankingu.

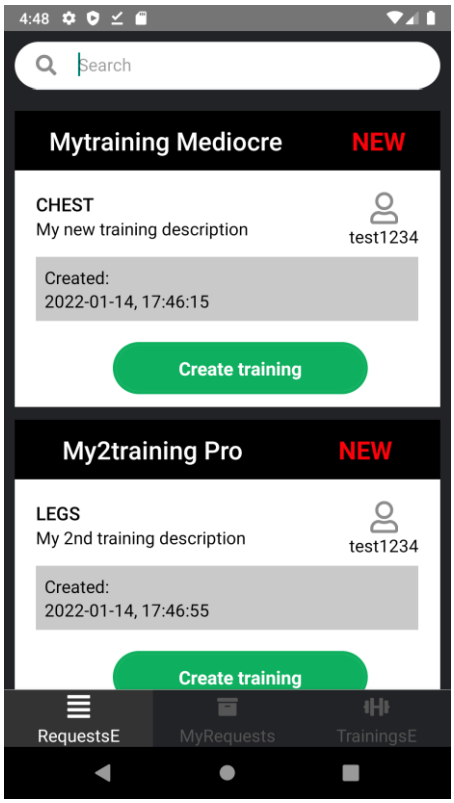


Rysunek 6.26. Ekran historii (opracowanie własne)

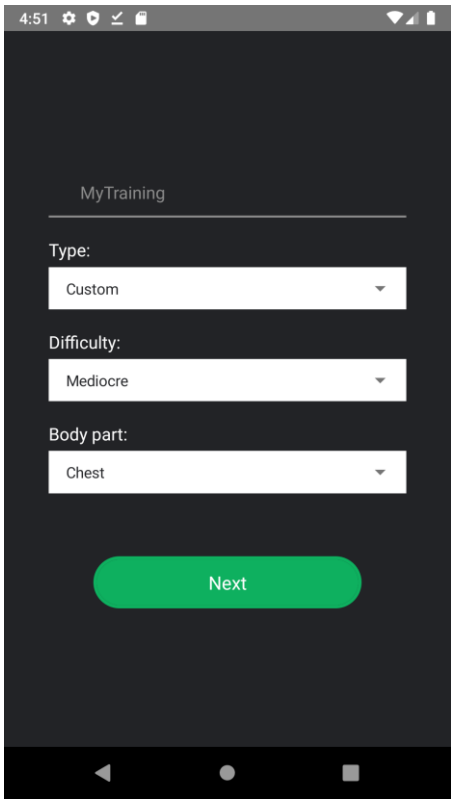


Rysunek 6.27. Ekran historii po treningach (opracowanie własne)

Na rysunkach 6.26 i 6.27 zaprezentowany został ekran historii treningów użytkownika. Kalendarz prezentuje aktualny dzień. Użytkownik może zmieniać, przeglądać ubiegłe i przyszłe miesiące. Dni, w które został wykonany trening, są zaznaczone zieloną kropką. Pod kalendarzem znajduje się przewijana lista z treningami wykonanymi w danym miesiącu. Analizując historię treningów, możemy stwierdzić, że w grudniu 2021 użytkownik nie wykonał żadnej aktywności, a w styczniu ćwiczył w dwa dni i wykonał co najmniej dwa treningi. Na karcie treningu pokazana jest data, nazwa, stopień trudności i czas gimnastyki.



Rysunek 6.28. Ekran nowych zgłoszeń (opracowanie własne)

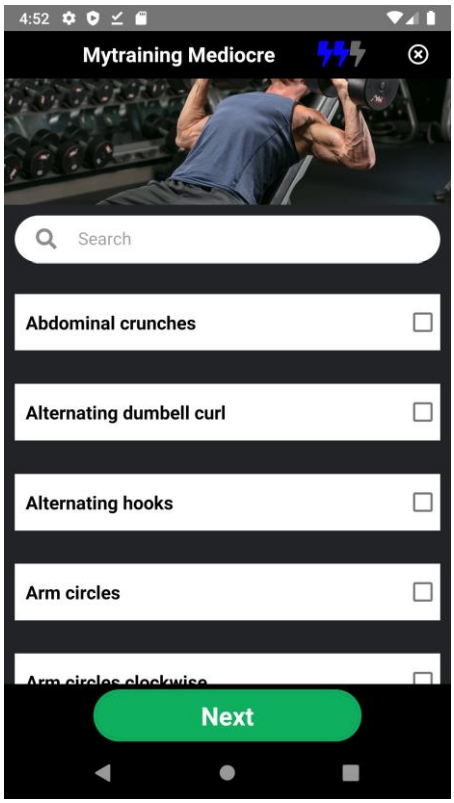


Rysunek 6.29. Ekran tworzenia treningu (opracowanie własne)

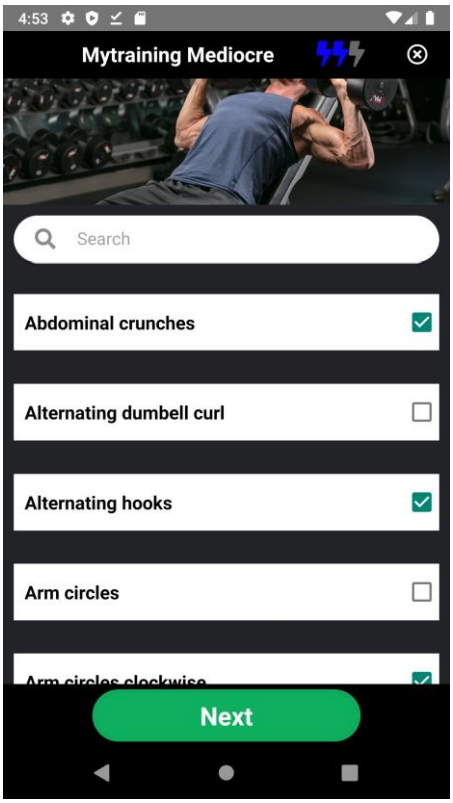
Na rysunku 6.28 przedstawiono ekran, który wyświetla się trenerom po zalogowaniu. Za pomocą paska wyszukiwania trenerzy mogą wyszukiwać zgłoszenia po nazwie lub stopniu trudności. Na karcie zgłoszenia znajduje się sugerowana nazwa treningu, poziom trudności, nazwa autora, część ciała, uwagi użytkownika i data utworzenia. Po naciśnięciu przycisku „Create training" aplikacja otworzy formularz tworzenia treningu, który jest widoczny na rysunku 6.29. Pole tekstowe służy do nadania nazwy treningu. Pola wyboru służą do określenia typu, poziomu trudności oraz części ciała. Istnieją dwa typy:

- „standard" - treningi dostępne dla wszystkich użytkowników,
- „custom" - treningi personalne.

W przypadku tworzenia treningu przy pomocy zgłoszenia musi być wybrany typ „custom". Wszystkie pola zostaną uzupełnione przez aplikacje. Trener musi zatwierdzić formularz przyciskiem „Next". Trening zostanie zapisany w bazie, a zgłoszenie przejdzie w stan „In Progress". Trening jednak nie jest jeszcze dostępny dla użytkownika. Trener musi jeszcze dodać ćwiczenia. Dlatego po zatwierdzeniu formularza aplikacja przenosi użytkownika do ekranu przedstawionego na rysunku 6.30.

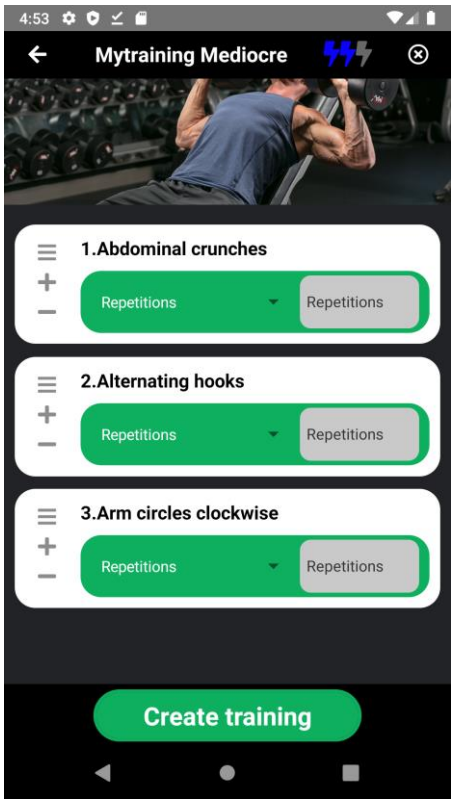


Rysunek 6.30. Ekran dodawania ćwiczeń (opracowanie własne)

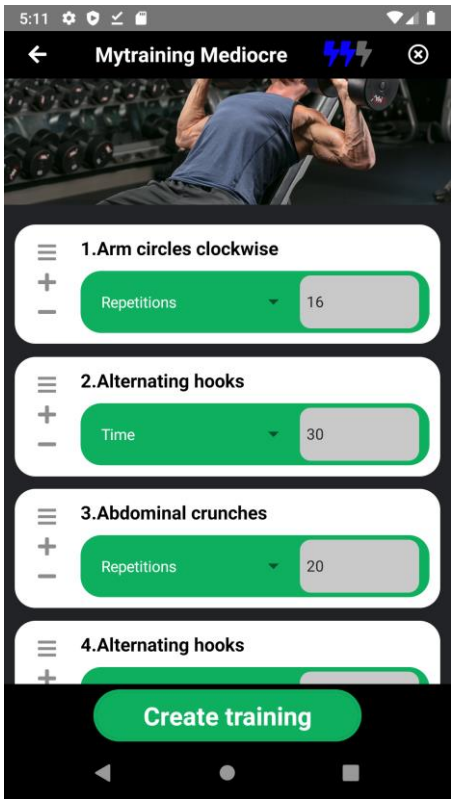


Rysunek 6.31. Ekran dodawania ćwiczeń (opracowanie własne)

Rysunki 6.30 i 6.31 prezentują ekran dodawania ćwiczeń do treningu. Trener może przewijać listę, wyszukiwać ćwiczenia po nazwie oraz zaznaczać ćwiczenia, które mają się znaleźć w treningu. Po naciśnięciu przycisku „Next” aplikacja przejdzie do następnego ekranu przedstawionego na rysunku 6.32.

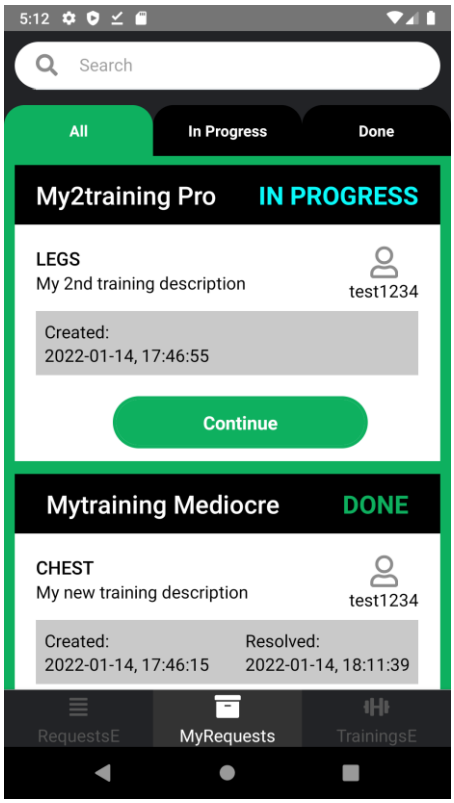


Rysunek 6.32. Ekran szczegółów ćwiczeń (opracowanie własne)

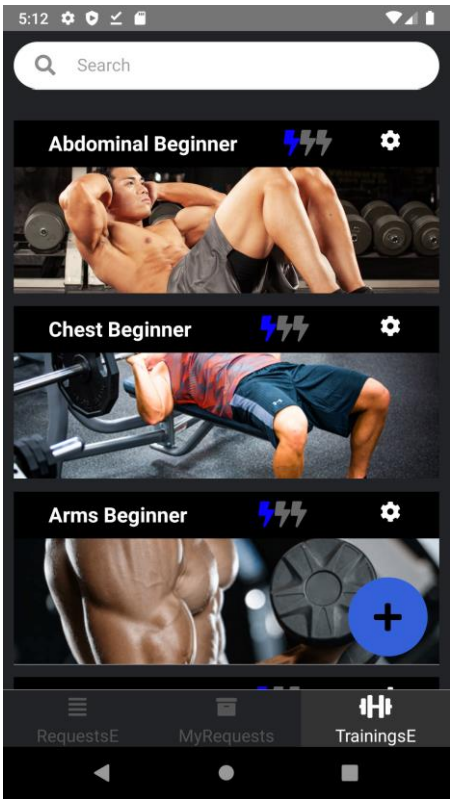


Rysunek 6.33. Ekran szczegółów ćwiczeń (opracowanie własne)

Na rysunkach 6.32 i 6.33 znajduje się ekran, na którym trener decyduje o czasie, kolejności oraz liczbie powtórzeń konkretnego ćwiczenia. Przy każdym ćwiczeniu znajdują się trzy przyciski: „trzy paski”, „+” oraz „-”. Pierwszy służy do zmiany kolejności ćwiczeń. Przycisk w kształcie plusa klonuje daną aktywność, a minus ją usuwa. Na liście rozwijanej znajdują się dwie pozycje „Repetitions” i „Time”. Lista określa, jak powinno być wykonane ćwiczenie. W polu tekstowym trener wpisuje liczbę powtórzeń lub liczbę sekund. Po naciśnięciu „strzałki” aplikacja wróci do ekranu wyboru ćwiczeń. Trener będzie mógł skorygować wybrane aktywności. Po zatwierdzeniu przyciskiem „Create training” program doda ćwiczenia do treningu. W tym momencie trening stanie się dostępny dla użytkownika. Tym samym zgłoszenie zostanie oznaczone jako ukończone.



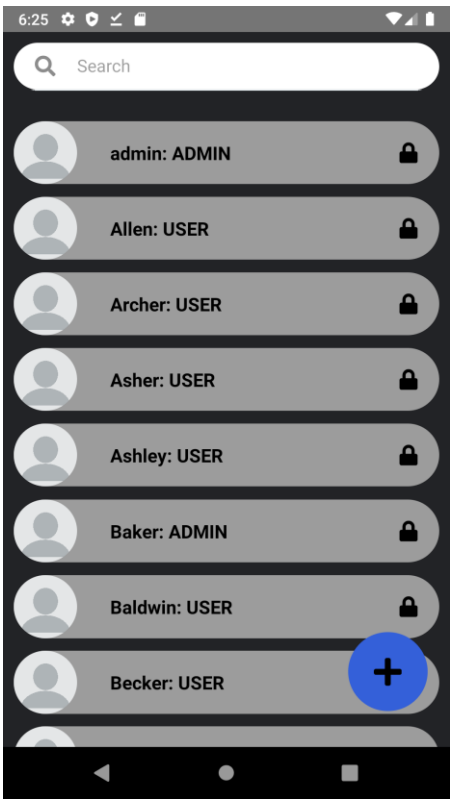
Rysunek 6.34. Ekran zgłoszeń trenera (opracowanie własne)



Rysunek 6.35. Ekran wszystkich treningów (opracowanie własne)

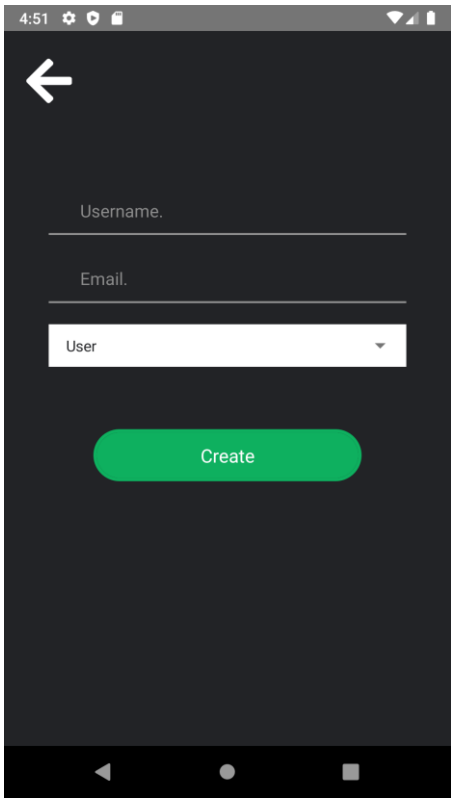
Rysunek 6.34 przedstawia ekran zgłoszeń przypisanych do trenera. Znajdują się tutaj zgłoszenia o statusie w trakcie (ang. in progress) i gotowe (ang. done). Tak jak w przypadku zgłoszeń użytkownika mamy pasek wyszukiwania oraz zakładki filtrujące po stanie zgłoszenia. Karty zgłoszeń wyświetlają te same informacje co na rysunku 6.28. Jeśli zgłoszenie nie jest ukończone, dostępny jest przycisk „Continue", który otworzy ekran wyboru ćwiczeń przedstawionych na rysunku 6.30. Gdy trener utworzy trening w procesie obsługi zgłoszenia, to zgłoszenie zostaje do niego przypisane. Gdyby na etapie dodawania ćwiczeń przerwał obsługę zgłoszenia, to będzie mógł do niej powrócić. Jeśli nie utworzy treningu, to zgłoszenie będzie wciąż dostępne w głównej puli dla wszystkich trenerów.

Na rysunku 6.35 znajduje się ekran, na którym dostępne są wszystkie treningi. Trenerzy mogą za jego pomocą modyfikować istniejące treningi lub tworzyć nowe. Po naciśnięciu ikony „zębatki" aplikacja otworzy ekran edycji ćwiczeń z rysunku 6.32. Po naciśnięciu „+" otworzy się formularza tworzenia treningu. Treningi utworzone z tego panelu będą dostępne dla wszystkich użytkowników.

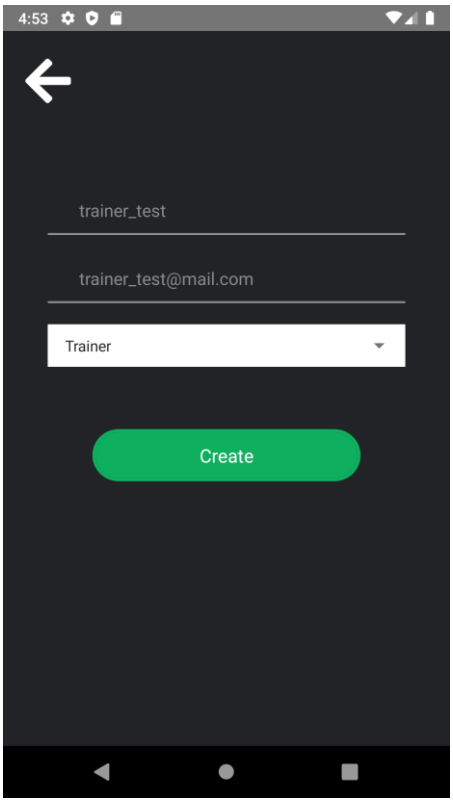


Rysunek 6.36. Ekran główny administratora (opracowanie własne)

Na rysunku 6.36 został przedstawiony ekran główny administratora. Na ekranie znajduje się przewijana lista użytkowników oraz pasek wyszukiwania. Na karcie użytkownika znajduje się zdjęcie profilowe, nazwa oraz rola. Po kliknięciu w kłódkę dostęp do konta dla danej osoby zostanie zablokowany. Ikona kłódki zmieni się na otwartą i po ponownym naciśnięciu dostęp zostanie przydzielony. Po naciśnięciu przycisku „+” aplikacja otworzy formularz tworzenia nowego konta.

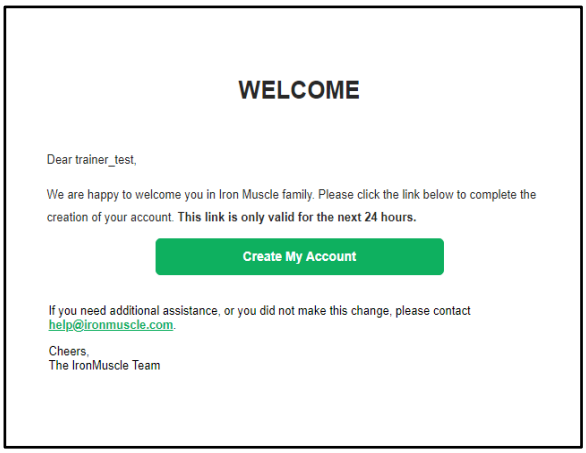


Rysunek 6.37. Ekran tworzenia konta (opracowanie własne)

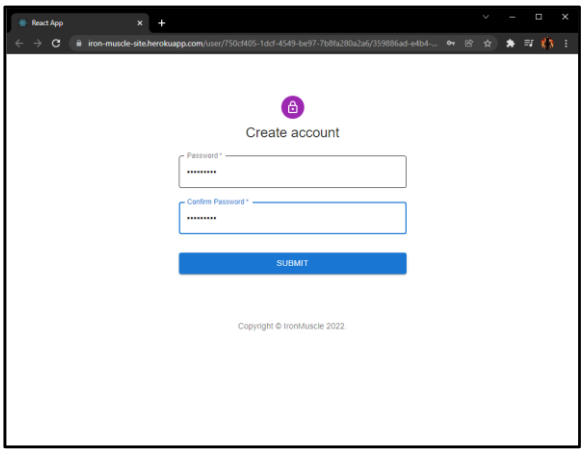


Rysunek 6.38. Ekran tworzenia konta uzupełniony (opracowanie własne)

Na rysunkach 6.37 oraz 6.38 zaprezentowano formularz do utworzenia nowego konta przez administratora. Jego zadaniem jest rejestrowanie nowych trenerów lub administratorów w systemie. Administrator wpisuje nazwę użytkownika oraz e-mail. Z listy rozwijanej może wybrać jedną z trzech ról: użytkownika (ang. user), trenera (ang. trainer), administratora (ang. admin). Po naciśnięciu przycisku „Create” konto zostaje utworzone, ale nie jest jeszcze dostępne. Na podany e-mail system wysyła wiadomość, dzięki której nowy trener może aktywować konto i nadać pierwsze hasło. Treść wiadomości jest widoczna poniżej na rysunku 6.39.



Rysunek 6.39. Wiadomość e-mail witająca nowego trenera (opracowanie własne)



Rysunek 6.40. Widok strony aktywacji i ustawienia hasła uzupełniony (opracowanie własne)

Rysunek 6.40 przedstawia stronę aplikacji internetowej, na której nowy trener może ustawić swoje pierwsze hasło. Po zatwierdzeniu formularza aplikacja przesyła żądanie do serwera, który szyfruje hasło i zapisuje w bazie danych oraz aktywuje konto użytkownika. Łącze zamieszczone w wiadomości e-mail może zostać wykorzystane tylko raz. Komunikaty o błędach we wpisanych hasłach wyświetlane są jako alerty. Aktywacja konta następuje po zapisaniu poprawnego hasła w pamięci trwałej, aby uniknąć sytuacji, w której konto byłoby dostępne bez wpisywania hasła.

7. Testy jednostkowe

Testy jednostkowe (ang. unit tests) polegają na weryfikacji działania pojedynczych jednostek aplikacji. Zaimplementowany system opiera się na komunikacji pomiędzy pamięcią trwałą, serwisami, kontrolerami oraz aplikacjami klienckimi. Główny nacisk został nałożony na napisanie testów integracyjnych, które zostaną omówione w następnym rozdziale. W pierwszym podrozdziale znajdują się testy jednostkowe wykonane przy pomocy JUnit.

7.1. Test walidacji e-mail

Każdy użytkownik w systemie posiada unikalny e-mail, który jednoznacznie określa użytkownika. Na podstawie adresu e-mail jesteśmy w stanie określić tożsamość użytkownika i nie znajdziemy dwóch użytkowników o tym samym adresie. Dodatkowo e-mail pełni funkcję zabezpieczenia w przypadku aktywacji konta i zmiany hasła. W przyszłości rozwijając aplikacje, e-mail może stanowić integralną część następujących funkcji: powiadomień o logowaniu na nowym urządzeniu oraz dwuetapowej weryfikacji. To ważne, aby e-mail podany podczas rejestracji był prawidłowy.

Walidacja adresu w systemie jest wykonywana za pomocą metody używającej wyrażenia regularnego do sprawdzenia podanego parametru tekstowego (ang. String) zwracającej wartość logiczną (ang. boolean). Metoda zwraca prawdę (ang. true), gdy e-mail jest poprawny i fałsz (ang. false), gdy e-mail zawiera błędy.

```
@Test
public void correctMail_1() {
  assertTrue(emailValidator.test("example@mail.com"));
}
@Test
public void correctMail_2() {
  assertTrue(emailValidator.test("example@mail.something.com"));}
@Test
public void correctMail_3() {
  assertTrue(emailValidator.test("example.something@mail.com"));
}
@Test
public void correctMail_4() {
  assertTrue(emailValidator.test("example@mail.pl"));
}
@Test
public void incorrectMail_1() {
  assertFalse(emailValidator.test("examplemail.com"));
}
@Test
public void incorrectMail_2() {
  assertFalse(emailValidator.test("example@mailcom"));
}
@Test
public void incorrectMail_3() {
  assertFalse(emailValidator.test("@mail.com"));
```

```
}
@Test
public void incorrectMail_4() {
  assertFalse(emailValidator.test("example@mail.com.")); }
@Test
public void emptyMail_1() {
  assertFalse(emailValidator.test(""));
}
```

Listing 7.1. Testy walidacji e-mail (opracowanie własne)

Wykonano dziewięć testów. Podano cztery poprawne e-maile, cztery niepoprawne oraz pusty ciąg znaków. W przypadku poprawnych adresów spodziewano się, że metoda zwróci prawdę, w przypadku niepoprawnych i ciągu pustego zwróci fałsz. Postawione założenia zostały spełnione, co oznacza, że wyrażenie regularne zostało dobrze zaimplementowane.

8. Testy integracyjne

Testy integracyjne są kolejnym etapem testowania oprogramowania. Mają na celu wykrycie niepożądanych efektów w interakcjach pomiędzy interfejsami lub modułami systemu. Sprawdzają one zgodność implementacji z postawionymi wymaganiami funkcjonalnymi. Podczas tego etapu zostały wykonane testy na następujących modułach aplikacji serwerowej: kontrolerach, serwisach i repozytoriach.

Testy na kontrolerach wymagały zainicjowania mockowanych serwisów, z których korzystają kontrolery oraz określania zachowania metod na konkretne dane wejściowe. Testy polegały na wywoływaniu tych metod poprzez zapytania HTTP wykonane przy pomocy klasy MockMvc, a następnie oczekiwaniu, czy odpowiedź po przetworzeniu zapytania jest taka sama jak oczekiwana.

Do przetestowania serwisów została napisana osobna konfiguracja aplikacji serwerowej, w której program używa osobnej lokalnej bazy danych opartej o H2. Dzięki takiemu podejściu można było sprawdzić metody serwisowe zapisujące i odczytujące z pamięci trwałej bez ingerowania w bazę produkcyjną.

Przy testach repozytoriów wykorzystano adnotację @DataJpaTest. Dopisanie jej przed klasą testową wyłącza pełną auto-konfiguracje. Wykorzystuje konfigurację dotyczącą testów JPA. Testy z tym przypisem są transakcyjne i cofają zmiany po zakończeniu każdego testu oraz używają tymczasowej bazy danych.

8.1. Testy kontrolera rejestracji

W każdej klasie testowej przed przystąpieniem do testów zaimplementowano metody inicjujące obiekty, które będą wykorzystywane w trakcie testów. Metody z adnotacją @Before zostaną wykonane przed każdą metodą testową. Takie rozwiązanie zapobiega redundancji kodu.

```
@Before
public void setupValidRequestAndWrongRequest() {
  validRequest = new RegistrationRequestDto(
        "alex",
        "alex@mail.com",
        "Alex#123",
        Collections.singletonList("USER"));
  wrongRequest = new RegistrationRequestDto(
        "alex",
        "alexmail.com",
        "Alex#123",
        Collections.singletonList("USER"));
}
```

Listing 8.1. Przygotowanie do testów kontrolera rejestracji (opracowanie własne)

W pierwszym teście sprawdzono odpowiedź kontrolera na poprawne dane. Po wywołaniu zapytania spodziewano się kodu odpowiedzi 200, który oznacza, że rejestracja przebiegła pomyślnie, czyli zgodnie z oczekiwaniami.

```
@Test
public void registerUser_Successfully() throws Exception {
  doNothing().when(registrationService).register(validRequest);

  mvc.perform(post("/api/v1/registration")
        .contentType(MediaType.APPLICATION_JSON)
        .content(asJsonString(validRequest)))
        .andExpect(status().isOk());
}
```

Listing 8.2. Test rejestracji z sukcesem (opracowanie własne)

Metoda w serwisie nie zwraca wartości. Jeżeli dane są niepoprawne, to metoda rzuca wyjątkiem, który zostanie wyłapany i odpowiednio obsłużony przez kontroler. W drugim teście podano niepoprawny e-mail i zmieniono zachowanie metody rejestracji, aby rzucała wyjątek. Zgodnie z oczekiwaniami kontroler wyłapał wyjątek i zwrócił kod błędu 400. Kontroler zachował się prawidłowo.

```
@Test
public void registerUser_Unsuccessfully_BadEmail() throws Exception {
  doThrow(new IllegalStateException("Email is not
valid")).when(registrationService).register(wrongRequest);

  mvc.perform(post("/api/v1/registration")
        .contentType(MediaType.APPLICATION_JSON)
        .content(asJsonString(wrongRequest)))
        .andExpect(status().is4xxClientError());
}
```

Listing 8.3. Test rejestracji z błędami (opracowanie własne)

8.2. Testy kontrolera użytkowników

```
@Before
public void setupUser() {
  user = IronUser.builder()
        .id(1L)
        .username("alex")
        .email("alex@gmail.com")
        .password("Alex#123")
        .icon("profile-picture/default/icon.png")
        .locked(false)
        .enabled(true)
        .roles(Collections.singletonList(new Role(1L, "USER")))
        .build();
}
```

Listing 8.4. Przygotowanie do testów kontrolera użytkowników (opracowanie własne)

W pierwszym teście kontrolera użytkowników sprawdzono działanie metody służącej do pobierania informacji o użytkowniku. Na podstawie wcześniej zainicjowanego obiektu powstały dwa kolejne: IronUserDetails oraz UserResponse. Pierwszy został użyty w metodzie wstrzykniętego serwisu do wygenerowanie tokenu autoryzacji. Drugi jest obiektem zwracanym przez metodę do pobierania szczegółów o osobie. Wywołane zapytanie zostało zaopatrzone w nagłówek autoryzacji z wygenerowanym tokenem. Metoda wyłuskuje nazwę użytkownika z tokena i na jej podstawie szuka go w bazie danych. Zgodnie z przewidywaniami po przyjęciu ciągu znaków w nagłówku metoda kontrolera zwraca obiekt UserResponse.

```
@Test
public void returnAuthorizedUserInformation() throws Exception {
  IronUserDetails userDetails = new IronUserDetails(user.dto());
  MockHttpServletRequest request = new MockHttpServletRequest();
  request.setRequestURI("/api/v1/login");

  String token = jwtUtil.createToken(request, userDetails, 1000 * 60);

  UserResponse response = UserResponse.builder()
        .id(user.getId())
        .username(user.getUsername())
        .email(user.getEmail())
        .build();

  when(userService.getMyself(any(String.class))).thenReturn(response);

  mvc.perform(get("/api/v1/myself")
        .contentType(MediaType.APPLICATION_JSON)
        .header(HttpHeaders.AUTHORIZATION, "Bearer " + token))
        .andDo(print())
        .andExpect(status().isOk())
        .andExpect(jsonPath("$.id", is(user.getId().intValue())))
```

```
        .andExpect(jsonPath("$.username", is(user.getUsername())))
        .andExpect(jsonPath("$.email", is(user.getEmail())));
}
```

Listing 8.5. Test pobrania informacji o autoryzowanym użytkowniku (opracowanie własne)

Drugi test polegał na zainicjowaniu obiektu strony przechowującej listę użytkowników. W liście znajdował się wcześniej zaaranżowany obiekt IronUser. Następnie wywołano metodę zwracającą paginowaną listę użytkowników z pustym ciągiem znaków jako zapytanie (ang. query). W ten sposób kontroler powinien zwrócić wszystkich użytkowników i tak się też stało, ponieważ pobrał stronę o liczbie elementów równej jeden. Tym pojedynczym elementem był zainicjowany użytkownik.

```
@Test
public void getPaginatedUsers_withoutFiltering() throws Exception {
  Page<IronUser> userPage = new PageImpl<>(Collections.singletonList(user));
  Pageable pageable = PageRequest.of(0, 100);

  when(userService.getPaginatedUsers(pageable, "")).thenReturn(userPage);

  mvc.perform(get("/api/v1/users")
        .contentType(MediaType.APPLICATION_JSON)
        .param("page", "0")
        .param("size", "100")
        .param("query", ""))
        .andExpect(status().isOk())
        .andDo(print())
        .andExpect(jsonPath("$.currentPage", is(0)))
        .andExpect(jsonPath("$.totalPages", is(1)))
        .andExpect(jsonPath("$.totalItems", is(1)))
        .andExpect(jsonPath("$.users[0].username", is(user.getUsername())));
}
```

Listing 8.6. Test pobrania stronicowanej listy użytkowników (opracowanie własne)

8.3. Testy kontrolera treningów

```
@Before
public void setupCreateTrainingAndTraining() {
  createTrainingDto = new CreateTrainingDto(
        "name",
        "standard",
        "abdominal",
        "beginner",
        10);
  training = new Training(
        1L,
        "name",
        "standard",
        "training/abdominal_beginner.png",
        "beginner", 10);
}
```

Listing 8.7 Przygotowanie do testów kontrolera treningów (opracowanie własne)

W pierwszym teście sprawdzono metodę zapisującą trening. Metoda przyjmuje obiekt CreateTrainingDto jako ciało zapytania (ang. body). Po otworzeniu treningu metoda powinna zwrócić obiekt zawierający informacje o treningu i tak też się stało.

```
@Test
public void saveTraining_Successfully() throws Exception {
  when(trainingsService.saveTraining(createTrainingDto)).thenReturn(training.response());

  mvc.perform(post("/api/v1/training")
        .contentType(MediaType.APPLICATION_JSON)
        .content(asJsonString(createTrainingDto)))
        .andExpect(status().isOk())
        .andDo(print())
        .andExpect(jsonPath("$.name", is(training.getName())))
        .andExpect(jsonPath("$.type", is(training.getType())))
        .andExpect(jsonPath("$.difficulty", is(training.getDifficulty())));
}
```

Listing 8.8. Test zapisu treningu (opracowanie własne)

W drugim teście przeprowadzono badanie zapytania pobierającego paginowaną listę treningów. Zgodnie z oczekiwaniami metoda zwróciła pierwszą stronę, a liczba elementów wynosiła jeden.

```
@Test
public void getPaginatedTrainings_withoutFiltering() throws Exception {
  List<Training> trainings = Collections.singletonList(training);

  Page<Training> trainingPage = new PageImpl<>(trainings);
  Pageable pageable = PageRequest.of(0, 100);

  when(trainingsService.getPaginatedTrainings(pageable, "")).thenReturn(trainingPage);

  mvc.perform(get("/api/v1/training/all")
        .contentType(MediaType.APPLICATION_JSON)
        .param("page", "0")
        .param("size", "100")
        .param("query", ""))
        .andExpect(status().isOk())
        .andDo(print())
        .andExpect(jsonPath("$.currentPage", is(0)))
        .andExpect(jsonPath("$.totalPages", is(1)))
        .andExpect(jsonPath("$.trainings[0].name", is(training.getName())));
}
```

Listing 8.9. Test pobrania stronicowanej listy treningów (opracowanie własne)

8.4. Testy serwisu użytkowników

```
@Before
public void setupUserAndFile() {
  user = IronUser.builder()
        .id(1L)
        .username("alex")
        .email("alex@gmail.com")
        .password("Alex#123")
        .icon("profile-picture/default/icon.png")
        .locked(false)
        .enabled(true)
        .roles(Collections.singletonList(new Role(1L, "USER")))
        .build();

  file = new MockMultipartFile("file","filename.png","image/png","example".getBytes());
}
```

Listing 8.10. Przygotowanie do testów serwisu użytkowników (opracowanie własne)

Większość metod dostępnych w aplikacji podlega mapowaniu obiektów na encje, zapisywaniu i odczytywaniu danych z pamięci trwałej. Najciekawszą metodą do przetestowania jest zmiana zdjęcia profilowego. Metoda przyjmuje obiekt użytkownika oraz tablicę bajtów nowej ikony. Jeśli użytkownik zmienia zdjęcie po raz pierwszy, system tworzy folder odpowiadający jego identyfikatorowi i w tym folderze zapisuje plik. Ścieżka do pliku jest zapisywana w bazie danych. Gdy użytkownik pobiera informacje o profilu, serwer na podstawie ścieżki odnajduje plik i odpowiednio opakowany wysyła do klienta. W teście sprawdzono, czy plik rzeczywiście jest zapisany we właściwej lokalizacji. Dodatkowo zweryfikowano, czy w encji klienta znajduje się ścieżka do nowej ikony.

```
@Test
public void whenUploadedIconSaved_thenCheckIfExists_thenCheckIfEqualsPathInUser() throws
IOException {

  userService.saveImage(user, file.getBytes());

  String MAIN_DIR = "src/main/resources/images/";
  String SUB_DIR = "/profile-picture/" + user.getId().toString() + "/";
  String FILE_LOCATION = SUB_DIR + user.getId().toString() + ".png";

  Path path = Paths.get(MAIN_DIR + FILE_LOCATION);
  assertTrue(Files.exists(path));

  IronUser found = userService.getUserByUsername(user.getUsername());

  assertEquals(FILE_LOCATION, found.getIcon());
}
```

Listing 8.11. Test zapisu zdjęcia profilowego (opracowanie własne)

8.5. Testy repozytorium użytkowników

```
@Before
public void setupUser() {
  Role role = new Role(1L, "USER");
  user = IronUser.builder()
        .username("alex")
        .email("alex@gmail.com")
        .password("Alex#123")
        .icon("profile-picture/default/icon.png")
        .locked(false)
        .enabled(true)
        .roles(Collections.singletonList(role))
        .build();
}
```

Listing 8.12. Przygotowanie do testów repozytorium użytkowników (opracowanie własne)

W pierwszym teście zapisano w bazie wcześniej przygotowanego użytkownika. Następnie przy użyciu repozytorium pobrano encję z bazy za pomocą nazwy konta. Zgodnie z założeniami obiekt przechowujący informacje o użytkowniku został zwrócony.

```
@Test
public void whenFindByUsername_thenReturnUser() {
  entityManager.persist(user);
  entityManager.flush();

  Optional<IronUser> found = userRepository.findByUsername(user.getUsername());
  assertTrue(found.isPresent());
  assertEquals(found.get().getUsername(), user.getUsername());
}
```

Listing 8.13. Test pobrania istniejącego użytkownika z bazy (opracowanie własne)

W następnym teście sprawdzono metodę do zarządzania użytkownikami. W pierwszej kolejności zapisano użytkownika w bazie danych. Następnie pobrano listę użytkowników. Zapytanie do bazy powinno zwrócić listę z jednym elementem, ponieważ osoba z taką nazwą przed chwilą została zapisana.

```
@Test
public void whenFindUsernameContains_thenReturnList_withOneElement() {
  entityManager.persist(user);
  entityManager.flush();

  Pageable pageable = PageRequest.of(0, 100);
  Page<IronUser> userPage = userRepository.findByUsernameContainsOrderByUsernameAsc(pageable,
user.getUsername());

  assertEquals(0, userPage.getNumber());
  assertEquals(1L, userPage.getTotalElements());
  assertEquals(1, userPage.getContent().size());
```

```
  assertEquals(user.getUsername(), userPage.getContent().get(0).getUsername());
}
```

Listing 8.14. Test pobrania stronicowej listy użytkowników z bazy (opracowanie własne)

Dla sprawdzenia przeprowadzono podobny test. W tym przypadku nie wykonano zapisu do bazy danych. Zwrócona lista była pusta. Efekt działania testu był zgodny z oczekiwaniami.

```
@Test
public void whenFindUsernameContains_thenReturnEmptyList() {
  Pageable pageable = PageRequest.of(0, 100);
  Page<IronUser> userPage = userRepository.findByUsernameContainsOrderByUsernameAsc(pageable,
"alex");

  assertEquals(0, userPage.getNumber());
  assertEquals(0L, userPage.getTotalElements());
  assertEquals(0, userPage.getContent().size());
}
```

Listing 8.15. Test pobrania pustej listy użytkowników z bazy (opracowanie własne)

8.6. Testy repozytorium ćwiczeń treningowych

```
@Before
public void setupTrainingAndExercises() {
  Training training = Training.builder()
.name("training_name").type("standard").difficulty("beginner").points(10).build();
  List<Exercise> exercises = new ArrayList<>();
  exercises.add(Exercise.builder()
              .name("exercise_name_1")
              .image("image")
              .video("videoId")
              .build());
  exercises.add(Exercise.builder()
        .name("exercise_name_2")
        .image("image")
        .video("videoId")
        .build());
  trainingId = (Long) entityManager.persistAndGetId(training);
  exercises.forEach(exercise -> entityManager.persist(exercise));
  exercises.forEach(exercise -> entityManager.persist(
        TrainingExercise.builder()
        .training(training)
        .exercise(exercise)
        .repetitions(20)
        .time(0).build()));
  entityManager.flush();
}
```

Listing 8.16. Przygotowanie do testów repozytorium ćwiczeń treningowych (opracowanie własne)

Przed przystąpieniem do testów w bazie zapisano trening oraz dwa ćwiczenia. W tabeli odpowiadającej za połączenie pomiędzy tymi encjami również wykonano odpowiednie wpisy. W poniższym teście sprawdzono, czy operacje wpisywania przebiegły pomyślnie.

```
@Test
public void returnTrainingExercises() {
  List<TrainingExercise> exercises = trainingExerciseRepository
        .findByTrainingId(trainingId);

  assertEquals(2, exercises.size());
  assertEquals("training_name", exercises.get(0).getTraining().getName());
  assertEquals("training_name", exercises.get(exercises.size() - 1)
      .getTraining().getName());
  assertEquals("exercise_name_1", exercises.get(0).getExercise().getName());
  assertEquals("exercise_name_2", exercises.get(exercises.size() - 1)
     .getExercise().getName());
}
```

Listing 8.17. Test pobrania listy ćwiczeń treningowych z bazy (opracowanie własne)

Gdy upewniono się, że wpisy przebiegają prawidłowo, zweryfikowane zostało działanie metody usuwającej połączenie między treningiem a ćwiczeniami, przy pomocy identyfikatora treningu.

```
@Test
public void deleteExercisesFromTraining_thenReturnEmptyList() {
  trainingExerciseRepository.deleteByTrainingId(trainingId);

  List<TrainingExercise> exercises = trainingExerciseRepository
      .findByTrainingId(trainingId);

  assertEquals(0, exercises.size());
}
```

Listing 8.18. Test usunięcia ćwiczeń treningowych z bazy (opracowanie własne)

8.7. Testy repozytorium zgłoszeń

```
@Before
public void setupUserAndTrainer() {
  IronUser user = IronUser.builder()
        .username("alex")
        .email("alex@gmail.com")
        .password("Alex#123")
        .icon("profile-picture/default/icon.png")
        .locked(false)
        .enabled(true)
        .roles(Collections.singletonList(new Role(1L, "USER")))
        .build();

  IronUser trainer = IronUser.builder()
        .username("james")
        .email("james@gmail.com")
        .password("James#123")
        .icon("profile-picture/default/icon.png")
        .locked(false)
        .enabled(true)
        .roles(Collections.singletonList(new Role(2L, "TRAINER")))
        .build();

  entityManager.persist(user);
  entityManager.persist(trainer);
  entityManager.flush();
}
```

Listing 8.19. Przygotowanie do testów repozytorium zgłoszeń (opracowanie własne)

Po przygotowaniu obiektów użytkownika i trenera zapisano zgłoszenie w bazie i sprawdzono, czy metoda wykorzystywana przy pobieraniu zgłoszeń klientów je zwraca.

```
@Test
public void createTrainingRequest_thenReturnPage() {
  Optional<IronUser> user = userRepository.findByUsername("alex");
  Optional<IronUser> trainer = userRepository.findByUsername("james");

  assertTrue(user.isPresent());
  assertTrue(trainer.isPresent());

  TrainingRequest trainingRequest = TrainingRequest.builder()
        .title("request")
        .description("description")
        .difficulty("beginner")
        .status("new")
        .bodyPart("arms")
        .created_at(LocalDateTime.now())
        .user(user.get())
        .build();

  entityManager.persist(trainingRequest);
```

```
  entityManager.flush();

  Pageable pageable = PageRequest.of(0, 100);
  Page<TrainingRequest> trainingRequestPage = trainingRequestRepository
        .findByUserIdAndStatusAndQuery(user.get().getId(), "new", "request", pageable);

  assertEquals(0, trainingRequestPage.getNumber());
  assertEquals(1, trainingRequestPage.getTotalElements());
  assertEquals(1, trainingRequestPage.getContent().size());
  assertEquals("request", trainingRequestPage.getContent().get(0).getTitle());
  assertEquals("new", trainingRequestPage.getContent().get(0).getStatus());
  assertEquals("alex", trainingRequestPage.getContent().get(0).getUser().getUsername());
}
```

Listing 8.20. Test zapisu zgłoszenia do bazy (opracowanie własne)

W drugim teście zaktualizowano istniejące zgłoszenie, zmieniając status i dołączając identyfikator trenera do klucza obcego. Następnie sprawdzono, czy dokonane zmiany poprawnie się zapisały.

```
@Test
public void createTrainingRequest_thenUpdate_thenReturnPage() {
  Optional<IronUser> user = userRepository.findByUsername("alex");
  Optional<IronUser> trainer = userRepository.findByUsername("james");

  assertTrue(user.isPresent());
  assertTrue(trainer.isPresent());

  TrainingRequest trainingRequest = TrainingRequest.builder()
        .title("request")
        .description("description")
        .difficulty("beginner")
        .status("new")
        .bodyPart("arms")
        .created_at(LocalDateTime.now())
        .user(user.get())
        .build();

  entityManager.persist(trainingRequest);
  entityManager.flush();

  trainingRequest.setTrainer(trainer.get());
  trainingRequest.setStatus("in progress");

  trainingRequestRepository.save(trainingRequest);

  Pageable pageable = PageRequest.of(0, 100);
  Page<TrainingRequest> trainingRequestPage = trainingRequestRepository
        .findByTrainerIdAndStatusAndQuery(
              trainer.get().getId(), "in progress", "request" , pageable);

  assertEquals(0, trainingRequestPage.getNumber());
```

```
  assertEquals(1, trainingRequestPage.getTotalElements());
  assertEquals(1, trainingRequestPage.getContent().size());
  assertEquals("request", trainingRequestPage.getContent().get(0).getTitle());
  assertEquals("in progress", trainingRequestPage.getContent().get(0).getStatus());
  assertEquals("alex", trainingRequestPage.getContent().get(0).getUser().getUsername());
  assertEquals("james", trainingRequestPage.getContent().get(0).getTrainer().getUsername());
}
```

Listing 8.21. Test aktualizacji zgłoszenia w bazie (opracowanie własne)

9. Testy sprzętowe

Jednym z podstawowych wymagań aplikacji mobilnych jest ich responsywność. Na rynku znajduje się niezliczona liczba modeli smartfonów oraz tabletów. W sklepach znajdziemy smartfony o przekątnej ekranu wahającej się od 4 do 7 cali oraz tablety od 7 do 13 cali. Możliwość korzystania z aplikacji na każdym z tych urządzeń zwiększa pulę potencjalnych klientów. Wygląd interfejsu to wizytówka developera startupu. Przejrzysty, skalowalny i przyjemny dla oka layout przyciągnie klientów. Źle zaimplementowany interfejs może sprawić, że program nie znajdzie wiernej grupy konsumentów pomimo innowacyjnych i potrzebnych na rynku funkcji. Podczas implementacji zadbano o ten aspekt projektu, a następnie przeprowadzono testy sprzętowe na kilku urządzeniach Android.

10. Podsumowanie i wnioski

Głównym celem niniejszej pracy dyplomowej było zaprojektowanie i zaimplementowanie systemu, którego zadaniem jest dopasowanie planu treningowego do konkretnego użytkownika. Postawione cele i założenia zostały w pełni zrealizowane. System zaplanowano i zrealizowano zgodnie z założeniami architektury REST. Do zapisu danych zaprojektowano relacyjną bazę danych. Upewniono się, że będą przechowywane tylko konieczne do poprawnego działania systemu informacje. Serwer zaimplementowano w języku Java przy użyciu frameworka Spring. Najważniejsze serwisy przetestowano za pomocą JUnit. Komunikacja pomiędzy aplikacją serwerową a klienckimi została nawiązana z wykorzystaniem protokołu HTTP. Podczas projektowania zdecydowano, że aplikacja mobilna będzie najlepszą formą implementacji części klienckiej ze względu na charakter programu i świadczone przez niego usługi. Zaprogramowano ją przy pomocy React Native, najbardziej popularnego narzędzia do pisania takiego oprogramowania.

Sprostano następującym wymaganiom funkcjonalnym. Użytkownicy aplikacji zostali podzieleni na grupy: użytkownicy zwyczajni alias klienci, trenerzy oraz administratorzy. Każda z tych grup ma dostęp do oddzielnych usług. System zapewnia konsumentom elementy grywalizacji w postaci odblokowywania odznak oraz zbierania punktów za wykonane treningi z możliwością sprawdzenia swojego miejsca w rankingu. Aplikacja umożliwia swego rodzaju komunikację pomiędzy klientami a trenerami w celu możliwości dobrania ćwiczeń do potrzeb konkretnego użytkownika. Odpowiednie moduły śledzą i udostępniają użytkownikom ich statystyki oraz postępy. Dodatkowo udostępniono treningi uniwersalne dostępne dla wszystkich użytkowników. Zaimplementowano również złożony system wsparcia dla ćwiczeń tak, aby osiągnąć, jak najlepsze rezultaty.

W procesie projektowania i implementacji systemu wyciągnięto następujące wnioski. Aplikacje treningowe są przyszłością branży fitness. Przy niewielkich kosztach utrzymania, jak i subskrypcji mogą stanowić realną alternatywę dla siłowni i trenerów personalnych. Dzięki popularności wykorzystanych technologii, a co za tym idzie dużej liczby potencjalnych członków zespołu, możliwy jest dalszy rozwój projektu, który mógłby obejmować rozszerzenie programu o inteligentne opaski mierzące tętno i spalone kalorie.

Literatura

[1] Anthony Accomazzo, Nate Murray, Ari Lerner, *Fullstack React*, Fullstack.io, 2017

[2] Christian Bauer, Gavin King, Linda DeMichiel, *Java Persistence with Hibernate*, Manning Publications, 2007

[3] Joshua Bloch, *Java. Efektywne programowanie*, Helion, 2018

[4] JSON [Online] https://www.json.org/json-en.html

[5] JUnit 5 [Online] https://junit.org/junit5/docs/current/user-guide/

[6] Catalin Tudose, *JUnit in Action, Third Edition*, Manning Publication Co., 2020

[7] JWT [Online] https://jwt.io/introduction

[8] Stephen Ludin, Javier Garza, *Learning HTTP/2*, O'Reilly Media, Inc., 2017

[9] Node.Js [Online] https://nodejs.org/en/about/

[10] PostgreSQL [Online] https://www.postgresql.org/about/

[11] React Native [Online] https://reactnative.dev/

[12] Mark Massé, *REST API Design Rulebook*, O'Reilly Media, Inc., 2012

[13] Spring Boot [Online] https://spring.io/projects/spring-boot

[14] Spring Data JPA [Online] https://spring.io/projects/spring-data-jpa

[15] Spring Security [Online] https://spring.io/projects/spring-security

Spis rysunków

Spis tabel

Spis listingów