



POLSKO-JAPOŃSKA AKADEMIA TECHNIK KOMPUTEROWYCH

Wydział Informatyki

Katedra Inżynierii Oprogramowania

Inżynieria Oprogramowania i Baz Danych

Mikołaj Rafalski, s26654

Natalia Tylek, s28827

Dawid Kołodziej, s26623

Coachify

**Platforma wspierająca współpracę trenerów
personalnych z klientami**

Praca inżynierska
napisana pod kierunkiem:

dr inż. Mariusza Trzaski, prof.
PJATK

Warszawa, lipiec, 2026

Streszczenie

Niniejsza praca inżynierska dotyczy projektu oraz implementacji systemu Coachify - platformy webowej wspierającej współpracę trenerów personalnych z klientami. Celem pracy było połączenie w jednym środowisku aplikacyjnym czynności, które w praktyce branżowej pozostają rozproszone pomiędzy komunikatory, arkusze kalkulacyjne i osobne aplikacje treningowe: wyszukiwania trenera, zebrania informacji o kliencie, tworzenia planów treningowych i dietetycznych, raportowania postępów, wymiany materiałów multimedialnych oraz komunikacji w czasie rzeczywistym.

System obejmuje trzy role użytkowników - klienta, trenera oraz administratora - a także publiczny katalog trenerów (*marketplace*) dostępny bez logowania. Aplikację serwową zrealizowano w języku Java 21 z wykorzystaniem frameworka Spring Boot, aplikację kliencką w bibliotece React (TypeScript, Vite), natomiast dane przechowywane są w relacyjnej bazie PostgreSQL oraz w magazynie obiektowym MinIO. Uwierzytelnianie oparto na tokenach JWT, komunikację modułu czatu na protokole WebSocket/STOMP, a całość przygotowano do uruchomienia w środowisku kontenerowym (Docker, Kubernetes, Helm) z wykorzystaniem procesu *CI/CD*.

Rezultatem pracy jest działający prototyp realizujący pełny przepływ współpracy - od założeń domenowych po wdrożenie. Pracę zakończono omówieniem osiągnięć, ograniczeń prototypu oraz kierunków dalszego rozwoju systemu.

Słowa kluczowe:

platforma fitness, trener personalny, aplikacja webowa, *marketplace*, plany treningowe, Spring Boot, React, PostgreSQL, REST API, konteneryzacja

Spis treści

1. WSTĘP	6
1.1. Cel pracy	6
1.2. Zakres pracy	6
1.3. Motywacja	7
1.4. Rezultaty pracy	7
1.5. Organizacja pracy	7
1.6. Podział prac w zespole	8
2. ISTNIEJĄCE ROZWIĄZANIA	9
2.1. Kontekst	9
2.2. Przegląd wybranych platform	10
2.2.1 Platformy wyszukiwania trenerów	10
2.2.2 Narzędzia do prowadzenia klientów	11
2.2.3 Społecznościowe aplikacje treningowe	12
2.2.4 Stan obecny - narzędzia ogólnego przeznaczenia	13
2.3. Porównanie funkcjonalności	14
2.4. Wnioski	17
3. MOTYWACJA I ZAŁOŻENIA ROZWIĄZANIA	18
3.1. Opis systemu z perspektywy użytkownika	18
3.2. Role w systemie	18
3.3. Główne moduły funkcjonalne	19
3.3.1 Rejestracja, logowanie i profile użytkowników	19
3.3.2 Kwestionariusz podopiecznego	20
3.3.3 Plany treningowe - szablony i przypisania	20
3.3.4 Plany dietetyczne - szablony, posiłki i produkty	20
3.3.5 Raporty z treningów i śledzenie postępów	20
3.3.6 Multimedia - zdjęcia postępów, awatary i materiały wideo ćwiczeń	21
3.3.7 Czat	21
3.3.8 Marketplace	21
3.3.9 Panel administratora	22
3.4. Wymagania funkcjonalne	22
3.5. Wymagania нефункционалне	25
3.6. Aktorzy i wybrane przypadki użycia	26
3.7. Model danych	30
3.8. Słownik pojęć	35
4. TECHNOLOGIE I NARZĘDZIA	37
4.1. Java 21 i Spring Boot	37
4.2. PostgreSQL i Hibernate	38
4.3. JSON Web Token (JWT)	39
4.4. OpenAPI / Swagger UI	41
4.5. MinIO	41
4.6. React, TypeScript, Vite	42
4.7. Tailwind CSS, Radix UI	43
4.8. Komunikacja czasu rzeczywistego	43
4.9. Docker i BuildKit	44
4.10. Kubernetes i Helm	44
4.11. Jenkins	45
4.12. Prometheus i Grafana	45
4.13. Trivy	46
4.14. Narzędzia wspierające	46

5.	IMPLEMENTACJA SYSTEMU.....	47
5.1.	Architektura systemu	47
5.1.1	Architektura logiczna	48
5.1.2	Podział warstwowy warstwy serwerowej	49
5.2.	Uwierzytelnianie i autoryzacja.....	50
5.3.	Moduł trenera.....	55
5.4.	Moduł podopiecznego	57
5.5.	Czat	59
5.6.	Marketplace	63
5.7.	Obsługa plików	64
5.8.	Panel administratora.....	67
6.	INTERFEJS UŻYTKOWNIKA	70
6.1.	Ogólna nawigacja i layout aplikacji	70
6.2.	Widoki trenera	71
6.2.1	Pulpit trenera i obsługa podopiecznych.....	71
6.2.2	Profil, usługi i ustawienia trenera.....	73
6.2.3	Katalog ćwiczeń trenera.....	74
6.2.4	Szablony planów treningowych	77
6.2.5	Plany treningowe przypisane klientom.....	81
6.2.6	Szablony diet, szablony posiłków i plany dietetyczne klientów	83
6.2.7	Raporty treningowe klientów	86
6.3.	Widoki podopiecznego	88
6.3.1	Onboarding i konfiguracja profilu klienta	88
6.3.2	Pulpit podopiecznego	96
6.3.3	Plany treningowe i kalendarz treningowy.....	97
6.3.4	Diety i szczegóły planu dietetycznego	98
6.3.5	Dziennik treningowy i raportowanie postępów	100
6.4.	Widoki administratora.....	102
6.4.1	Pulpit administratora	102
6.4.2	Zarządzanie ćwiczeniami globalnymi	103
6.4.3	Zarządzanie trenerami	103
6.4.4	Zarządzanie klientami	104
6.5.	Widoki publiczne i marketplace	104
6.5.1	Strona główna, rejestracja i logowanie.....	104
6.5.2	Lista trenerów w marketplace	106
6.5.3	Profil trenera, prośba o współpracę i status zgłoszenia	107
6.6.	Czat	108
6.6.1	Lista konwersacji i aktywna rozmowa.....	109
6.6.2	Prośby o wiadomość.....	110
6.6.3	Zarządzanie zablokowanymi klientami.....	112
6.7.	Multimedia	112
6.8.	Responsywność i dostępność	112
7.	TESTOWANIE APLIKACJI	114
7.1.	Podejście do testowania w projekcie.....	114
7.2.	Testy jednostkowe	114
7.3.	Testy warstwy webowej i kontraktu API	115
7.4.	Testy integracyjne	115
7.5.	Testy manualne	116
7.6.	Testowanie w procesie budowania.....	116
7.7.	Skanowanie bezpieczeństwa obrazów kontenerowych	118
7.8.	Health check i weryfikacja wdrożenia	118
7.9.	Pokrycie kodu	119
7.10.	Znane ograniczenia i wnioski z testowania.....	120
8.	DEVOPS I WDROŻENIE.....	121

8.1.	Konteneryzacja	121
8.2.	Orkiestracja	122
8.3.	<i>Pipeline CI/CD</i>	126
8.4.	Konfiguracja środowisk	129
8.5.	Monitoring	130
8.6.	Ograniczenia	132
9.	PODSUMOWANIE	133
9.1.	Osiągnięcia.....	133
9.2.	Problemy napotkane podczas realizacji	134
9.3.	Ograniczenia prototypu.....	135
9.4.	Niezrealizowane funkcje i kierunki rozwoju	135
9.5.	Wnioski końcowe.....	136
10.	BIBLIOGRAFIA.....	137
11.	SPIS RYSUNKÓW	140
12.	SPIS TABEL.....	143
13.	SPIS LISTINGÓW	144
14.	DODATKI	145
	Dodatek A: Słownik skrótów i terminologii	145
	Dodatek B: Zawartość nośnika.....	147
	Dodatek C: Przykładowe elementy konfiguracji.....	147

1. Wstęp

W niniejszym rozdziale przedstawiono cel, zakres, motywację oraz rezultaty niniejszej pracy inżynierskiej dotyczącej systemu Coachify. System został zaprojektowany jako platforma wspierająca współpracę trenerów personalnych z klientami. Zakres pracy obejmuje zarówno warstwę aplikacyjną, jak i wybrane elementy infrastruktury potrzebne do uruchomienia prototypu w środowisku kontenerowym.

Współpraca trenera personalnego z podopiecznym obejmuje wiele powiązanych czynności: zebranie informacji o kliencie, przygotowanie planu treningowego, przygotowanie planu dietetycznego, raportowanie wykonanych treningów, wymianę materiałów multimedialnych oraz komunikację bieżącą. Rozproszenie tych czynności między komunikator, arkusz kalkulacyjny, osobne aplikacje treningowe i ręcznie prowadzone notatki utrudnia utrzymanie spójnego obrazu współpracy. Coachify odpowiada na ten problem przez połączenie kluczowych przepływów w jednej aplikacji webowej.

1.1. Cel pracy

Celem niniejszej pracy inżynierskiej było zaprojektowanie i zaimplementowanie prototypu systemu, który umożliwia trenerom personalnym i klientom prowadzenie współpracy w jednym środowisku aplikacyjnym. System obsługuje rolę klienta, trenera oraz administratora, udostępnia publiczny katalog trenerów, wspiera proces nawiązywania współpracy, pozwala na tworzenie i udostępnianie planów treningowych oraz dietetycznych, umożliwia raportowanie wykonanych treningów i przechowywanie multimediów oraz zapewnia komunikację w czasie rzeczywistym.

Celem technicznym pracy było przygotowanie aplikacji serwerowej, aplikacji klienckiej oraz konfiguracji wdrożeniowej umożliwiającej uruchomienie rozwiązania w środowisku kontenerowym. Istotnym elementem celu było także zastosowanie odpowiednich mechanizmów bezpieczeństwa, walidacji danych, dokumentacji API adekwatnych do prototypu inżynierskiego.

1.2. Zakres pracy

Niniejsza aplikacja webowa składa się z warstwy serwerowej, warstwy klienckiej oraz elementów infrastruktury wdrożeniowej. Serwer systemu został oparty na architekturze warstwowej i udostępnia REST API oraz kanał WebSocket/STOMP dla modułu czatu.

Aplikacja kliencka została przygotowana jako aplikacja typu *Single-Page Application*, z osobnymi widokami dla trenera, klienta i administratora. Dane transakcyjne są przechowywane w relacyjnej bazie PostgreSQL, natomiast pliki binarne, takie jak awatary, zdjęcia postępów i materiały wideo ćwiczeń, są przechowywane w MinIO.

Zakres funkcjonalny systemu obejmuje rejestrację i logowanie użytkowników, profile trenera i klienta, kwestionariusz podopiecznego, *marketplace* trenerów, zgłoszenia o współpracę, plany treningowe, plany dietetyczne, szablony planów, raporty z treningów, zdjęcia postępów, bibliotekę ćwiczeń, czat w czasie rzeczywistym oraz panel administratora. Zakres infrastrukturalny obejmuje konteneryzację aplikacji, przygotowanie wartości konfiguracyjnych dla środowisk *dev* i *prod*, integrację z Jenkinsem oraz wdrożenie w Kubernetes z użyciem Helm.

Niniejsza aplikacja może zostać rozszerzona o funkcje płatności, automatyczne dopasowanie trenera do klienta, generowanie planów z użyciem sztucznej inteligencji oraz o aplikację mobilną.

1.3. Motywacja

Motywacją do realizacji systemu była potrzeba uporządkowania procesu współpracy między trenerem personalnym a klientem. W typowym przebiegu współpracy trener potrzebuje informacji o celu treningowym, poziomie doświadczenia, ograniczeniach zdrowotnych, dostępności sprzętu oraz preferencjach klienta. Następnie musi przygotować plan, przekazać go podopiecznemu, odbierać raporty z wykonania oraz reagować na postępy. Jeżeli każdy z tych etapów jest obsługiwany w innym narzędziu, rośnie ryzyko utraty kontekstu, nieaktualnych danych oraz trudności w śledzeniu historii współpracy.

Motywacja inżynierska wynikała z możliwości opracowania pełnego systemu obejmującego kilka istotnych obszarów wytwarzania oprogramowania: model domenowy, bezpieczeństwo, komunikację w czasie rzeczywistym, obsługę plików, interfejs użytkownika, testowanie oraz wdrożenie kontenerowe.

1.4. Rezultaty pracy

Rezultatem pracy jest prototyp systemu Coachify obejmujący trzy główne role użytkowników. Klient może utworzyć konto, uzupełnić profil, przejść kwestionariusz preferencji, przeglądać *marketplace* trenerów, wysłać zgłoszenie o współpracę, korzystać z przypisanych planów, raportować treningi oraz komunikować się z trenerem.

Trener może zarządzać profilem i ofertą usług, przyjmować lub odrzucać zgłoszenia, tworzyć plany treningowe i dietetyczne, korzystać z szablonów, zarządzać ćwiczeniami, analizować raporty klientów oraz prowadzić rozmowy.

Administrator ma dostęp do panelu umożliwiającego przegląd trenerów i klientów oraz zarządzanie biblioteką ćwiczeń globalnych.

W warstwie technicznej przygotowano aplikacje serwerową w technologii Java 21 i Spring Boot, aplikacje kliencką w React, TypeScript i Vite, bazę PostgreSQL, obiektowe przechowywanie plików w MinIO, dokumentację API w Swagger UI, mechanizmy uwierzytelniania oparte na JWT i tokenach odświeżających, czat w czasie rzeczywistym, testy jednostkowe i wycinkowe oraz konfigurację uruchomieniową dla środowiska Kubernetes. Prototyp nie jest prezentowany jako gotowy produkt komercyjny, lecz jako działające rozwiązanie inżynierskie pokazujące pełny przepływ od założeń domenowych do wdrożenia.

1.5. Organizacja pracy

Niniejsza praca została podzielona na dziewięć rozdziałów oraz część załącznikową.

Rozdział pierwszy wprowadza w tematykę pracy i przedstawia jej cel, zakres, motywację, rezultaty, a także organizację dokumentu oraz podział prac w zespole.

Rozdział drugi omawia istniejące rozwiązania wspierające współpracę trenerów z klientami, klasyfikuje je w trzech kategoriach produktowych, zestawia ich funkcjonalności i wskazuje lukę rynkową uzasadniając powstanie systemu Coachify.

Rozdział trzeci przedstawia motywację oraz założenia rozwiązania: opis systemu z perspektywy użytkownika, role w systemie, główne moduły funkcjonalne, wymagania funkcjonalne i niefunkcjonalne, aktorów i wybrane przypadki użycia, model danych oraz słownik pojęć.

Rozdział czwarty opisuje technologie i narzędzia zastosowane w projekcie, w oderwaniu od samej implementacji.

Rozdział piąty dotyczy implementacji systemu - architektury, mechanizmów uwierzytelniania i autoryzacji oraz poszczególnych modułów warstwy serwerowej.

Rozdział szósty prezentuje interfejs użytkownika w podziale na widoki trenera, podopiecznego, administratora oraz widoki publiczne.

Rozdział siódmy omawia przyjęte podejście do testowania oraz poszczególne rodzaje testów zastosowane w projekcie.

Rozdział ósmy opisuje obszar DevOps i wdrożenia: konteneryzację, orkiestrację, proces *CI/CD*, konfigurację środowisk oraz monitoring.

Rozdział dziewiąty stanowi podsumowanie - przedstawia osiągnięcia, napotkane problemy, ograniczenia prototypu, niezrealizowane funkcje i kierunki rozwoju oraz wnioski końcowe.

Pracę uzupełniają bibliografia oraz dodatki zawierające słownik skrótów i terminologii, opis zawartości nośnika oraz przykładowe elementy konfiguracji.

1.6. Podział prac w zespole

Pracę zrealizował trzyosobowy zespół. Każdy z autorów wniósł porównywalny wkład w realizację prototypu oraz tekstu pracy, przy czym poszczególne obszary zostały rozdzielone zgodnie z kompetencjami i zainteresowaniami członków zespołu.

Mikołaj Rafalski odpowiadał za obszar DevOps oraz przygotowanie aplikacji do działania w środowisku zbliżonym do produkcyjnego. Do jego zadań należała konteneryzacja aplikacji serwerowej i klienckiej, dostosowanie konfiguracji wdrożeniowej oraz uruchomienie systemu na środowisku serwerowym. Brał również udział w poprawkach aplikacyjnych oraz w usprawnieniach warstwy interfejsu użytkownika.

Natalia Tylek pełniła rolę głównej osoby odpowiedzialnej za rdzeń aplikacji. Od początku projektu rozwijała warstwę serwerową, odpowiadając za implementację funkcjonalności oraz logiki biznesowej, a także za spięcie głównych elementów systemu w działającą całość. W największym stopniu odpowiadała również za projekt bazy danych.

Dawid Kołodziej odpowiadał za bardziej złożone elementy aplikacji. Brał aktywny udział w implementacji widoków warstwy klienckiej oraz rozwijał warstwę serwerową, koncentrując się na tworzeniu i wdrażaniu wymagających funkcjonalności i rozwiązań technicznych.

2. Istniejące rozwiązania

W niniejszym rozdziale przedstawiono przegląd istniejących rozwiązań wspierających współpracę trenerów personalnych z klientami. Omówiono kontekst rynkowy oraz rzeczywiste praktyki branży, scharakteryzowano reprezentatywne rozwiązania w trzech kategoriach produktowych, zestawiono ich funkcjonalności oraz sformułowano wnioski wpływające na założenia systemu Coachify. Analizę przeprowadzono przede wszystkim z perspektywy rynku polskiego, na który kierowana jest początkowa wersja platformy, przy uwzględnieniu rozwiązań zagranicznych jako punktu odniesienia ze względu na planowaną w przyszłości internacjonalizację.

2.1. Kontekst

Rynek oprogramowania wspierającego pracę trenerów personalnych rozwija się dynamicznie od około dekady, napędzany rosnącą popularnością *coachingu*. Pomimo dostępności licznych rozwiązań rzeczywista praktyka prowadzenia podopiecznych często odbiega od modelu zakładanego przez producentów oprogramowania. Znaczna część trenerów prowadzi działalność w sposób rozproszony: ofertę publikuje na własnej stronie lub w mediach społecznościowych, komunikację prowadzi przez komunikatory (Instagram, WhatsApp), plany przekazuje w postaci dokumentów lub wiadomości, a płatności obsługuje poza jakimkolwiek zintegrowanym systemem. To rozproszone podejście, oparte na narzędziach ogólnego przeznaczenia należy uznać za faktyczny stan obecny branży i realny punkt odniesienia dla projektowanego systemu.

Problem dotyczy obu stron współpracy. Z perspektywy klienta barierą stanowi trudność w znalezieniu i porównaniu ofert trenerów, brak jednego miejsca do śledzenia własnych postępów oraz częsty brak dalszego zaangażowania trenera po przekazaniu planu. Z perspektywy trenera problemem jest konieczność łączenia wielu narzędzi oraz, w przypadku rozwiązań komercyjnych w modelu oprogramowania jako usługi (ang. *Software as a Service, SaaS*), ponoszenie opłat abonamentowych jeszcze przed pozyskaniem pierwszego podopiecznego.

Wśród przeanalizowanych rozwiązań nie zidentyfikowano systemu oferującego zakres tożsamy z systemem Coachify. Dostępne narzędzia zajmują natomiast wyspecjalizowane nisze, które można pogrupować w trzy kategorie produktowe:

- platformy wyszukiwania trenerów (ang. *marketplace*) - umożliwiają klientowi znalezienie i wybór trenera, lecz nie zapewniają rozbudowanego środowiska prowadzenia współpracy;
- narzędzia do prowadzenia klientów (ang. *coaching, management*) - udostępniają trenerowi zaawansowane funkcje tworzenia planów i raportowania, lecz nie pełnią funkcji otwartego katalogu i kierują pozyskiwanie klientów poza platformę;
- społecznościowe aplikacje treningowe - oferują śledzenie postępów oraz mechanizmy społecznościowe i motywacyjne, lecz nie obsługują relacji trener-klient.

System Coachify plasuje się na przecięciu tych trzech kategorii, łącząc otwartą platformę wyszukiwania trenerów, pełne środowisko prowadzenia podopiecznego oraz podejście zorientowane na klienta. Z tego powodu w dalszej części rozdziału rozwiązania porównano w obrębie każdej kategorii osobno, z zastosowaniem kryteriów istotnych dla danej grupy.

Analizę oparto na publicznie dostępnych informacjach producentów, dokumentacji oraz niezależnych recenzjach i porównaniach branżowych. Funkcjonalności poszczególnych platform zweryfikowano bezpośrednio tam, gdzie umożliwiała to bezpłatna rejestracja.

2.2. Przegląd wybranych platform

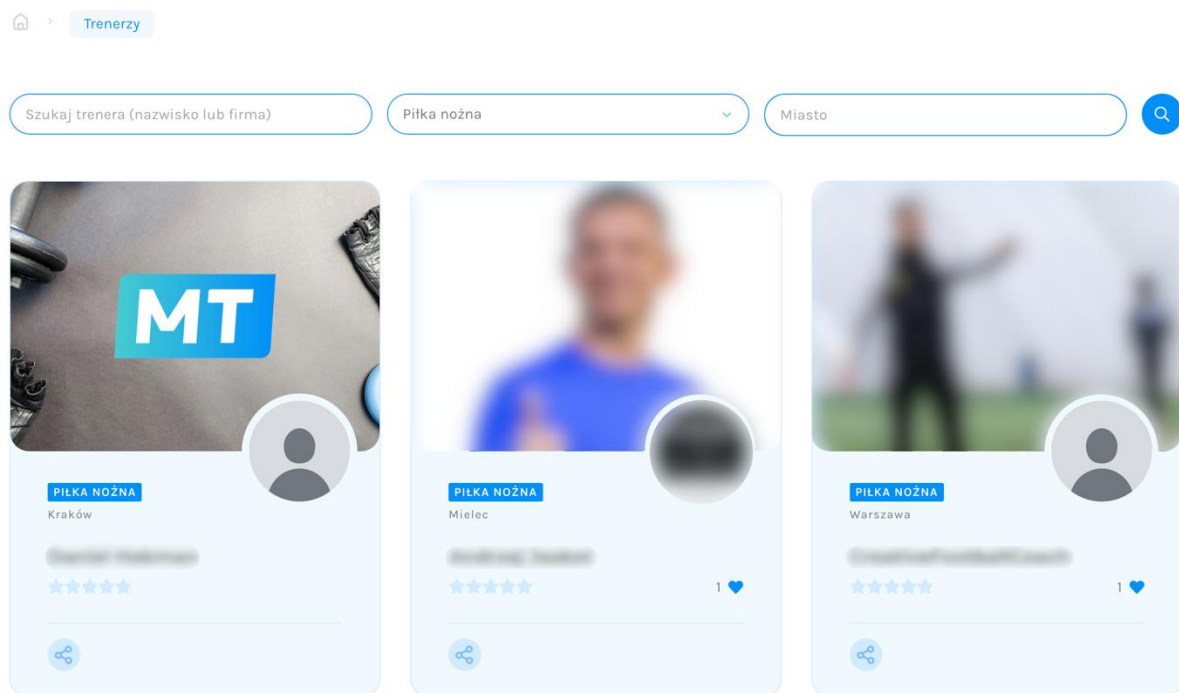
Kolejne podrozdziały omawiają szczegółowo trzy wyróżnione kategorie rozwiązań oraz stan obecny oparty na narzędziach ogólnego przeznaczenia.

2.2.1 Platformy wyszukiwania trenerów

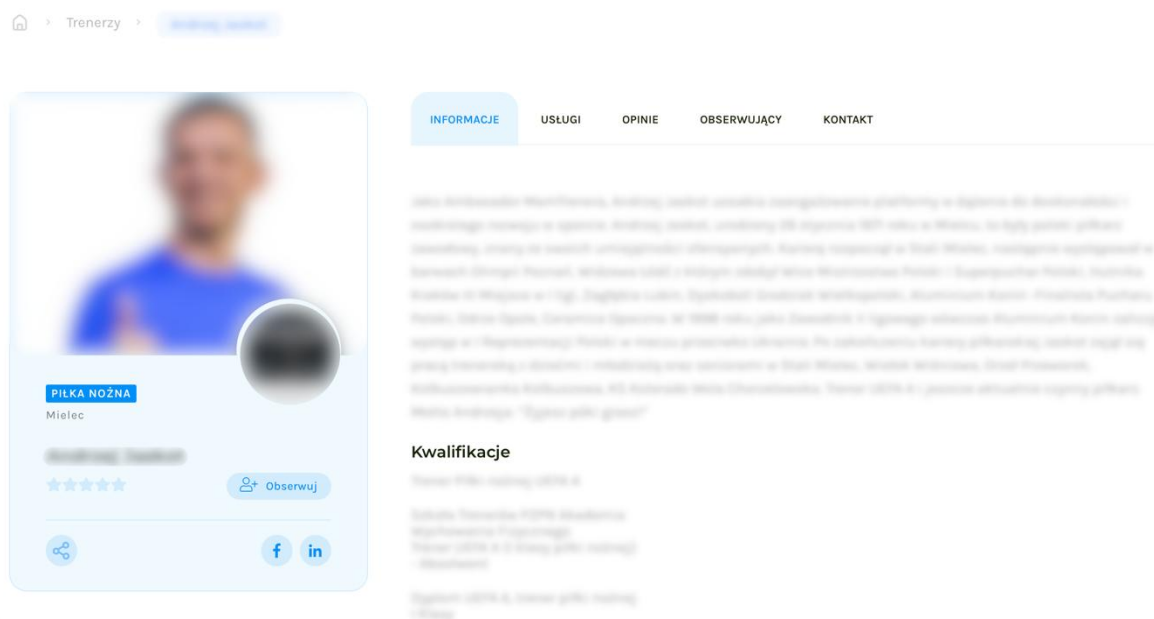
Kategoria ta jest najbliższa modelowi Coachify w wymiarze udostępniania oferty trenerów potencjalnym klientom.

Platformy wyszukiwania trenerów są najbliższe modelowi Coachify w wymiarze udostępniania oferty trenerów potencjalnym klientom. Na rynku polskim funkcjonuje kilka takich serwisów. Serwis TrenerHUB zrzesza ponad 200 trenerów [1] i działa w modelu pośrednictwa typu *marketplace*: klient przegląda profile trenerów i kontaktuje się z wybraną osobą, najczęściej rezerwując bezpłatną konsultację, a serwis deklaruje weryfikację wykształcenia, doświadczenia i kursów trenerów [1]. Serwis nie udostępnia jednak wbudowanego czatu, kwestionariusza wstępnego ani modułu prowadzenia treningu. Serwis MamTrenera.pl umożliwia wyszukiwanie trenerów według dyscypliny i lokalizacji, rezerwację sesji oraz wystawianie ocen; kontakt z trenerem odbywa się jednak poza serwisem (np. drogą mailową), bez wbudowanego komunikatora, a korzystanie z platformy jest bezpłatne dla użytkowników [2]. Serwis TwójTrener.pl udostępnia rozbudowane profile trenerów zawierające cennik, pakiety usług, lokalizację oraz oceny, pełniąc funkcję otwartego katalogu ofert [3].

Przykładowy widok wyszukiwania trenerów w serwisie tego typu przedstawiono na rysunku 1, a przykładowy profil trenera na rysunku 2.



Rysunek 1. Widok wyszukiwania trenerów w serwisie MamTrenera. Źródło: [2].



Rysunek 2. Profil trenera w serwisie wyszukiwania. Źródło: [2].

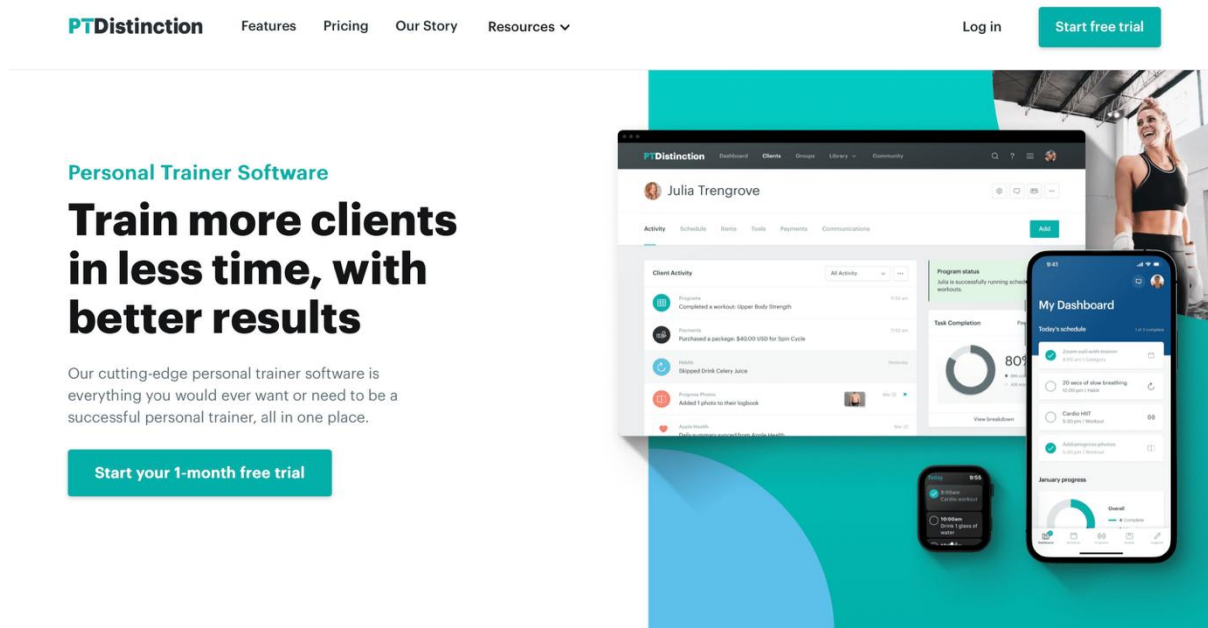
Na rynku zagranicznym analogiczny model reprezentuje serwis FindYourTrainer) [4], kojarzący klientów z trenerami na podstawie krótkiego formularza dopasowującego i umożliwiający bezpośrednią rezerwację sesji w kalendarzu trenera.

Wspólną cechą tej kategorii jest skupienie na etapie wyszukiwania, rezerwacji i pierwszego kontaktu. Rozwiązania te nie udostępniają rozbudowanego środowiska prowadzenia współpracy - tworzenia ustrukturyzowanych planów treningowych i dietetycznych, kwestionariusza wstępnego warunkującego współpracę ani dziennika treningowego z poziomu pojedynczych serii i powtórzeń.

2.2.2 Narzędzia do prowadzenia klientów

Dojrzałe narzędzia w modelu *SaaS* wyznaczają standard funkcjonalny w zakresie prowadzenia podopiecznych. Platforma ABC Trainerize to jedno z najbardziej rozpoznawalnych rozwiązań, według danych producenta używane przez ponad 400 000 trenerów [5]. Udostępnia kreator planów treningowych z biblioteką ćwiczeń, planowanie żywienia, śledzenie nawyków i postępów, komunikację, obsługę płatności, integracje z urządzeniami ubieralnymi oraz aplikację mobilną pod marką trenera [5]. Platforma TrueCoach jest skierowana wyłącznie do trenerów i koncentruje się na prowadzeniu współpracy jeden-do-jednego; umożliwia tworzenie planów żywieniowych oraz integrację z aplikacją MyFitnessPal [6]. Platforma PT Distinction udostępnia wszystkie funkcje w każdym planie cenowym, w tym planer posiłków wspomagany sztuczną inteligencją i automatyczne przepływy pracy [7]. Na rynku globalnym działają również platforma My PT Hub, stosująca model stałej opłaty za Nielimitowaną liczbę klientów [8], oraz platforma Everfit, oferująca zaawansowany kreator treningów wraz z modułem żywieniowym [9]. Na rynku polskim analogiczne narzędzia oferują serwis CoachPro, udostępniający system zarządzania relacjami z klientem, kalendarz, raporty PDF oraz bazę ponad 2500 ćwiczeń [10], serwis TreningLab z aplikacją mobilną dla klientów [11] oraz serwis Fitebo z personalizacją aplikacji klienckiej i bazą ponad 2000 ćwiczeń [12].

Stronę powitalną przykładowej platformy komercyjnej przedstawiono na rysunku 3.



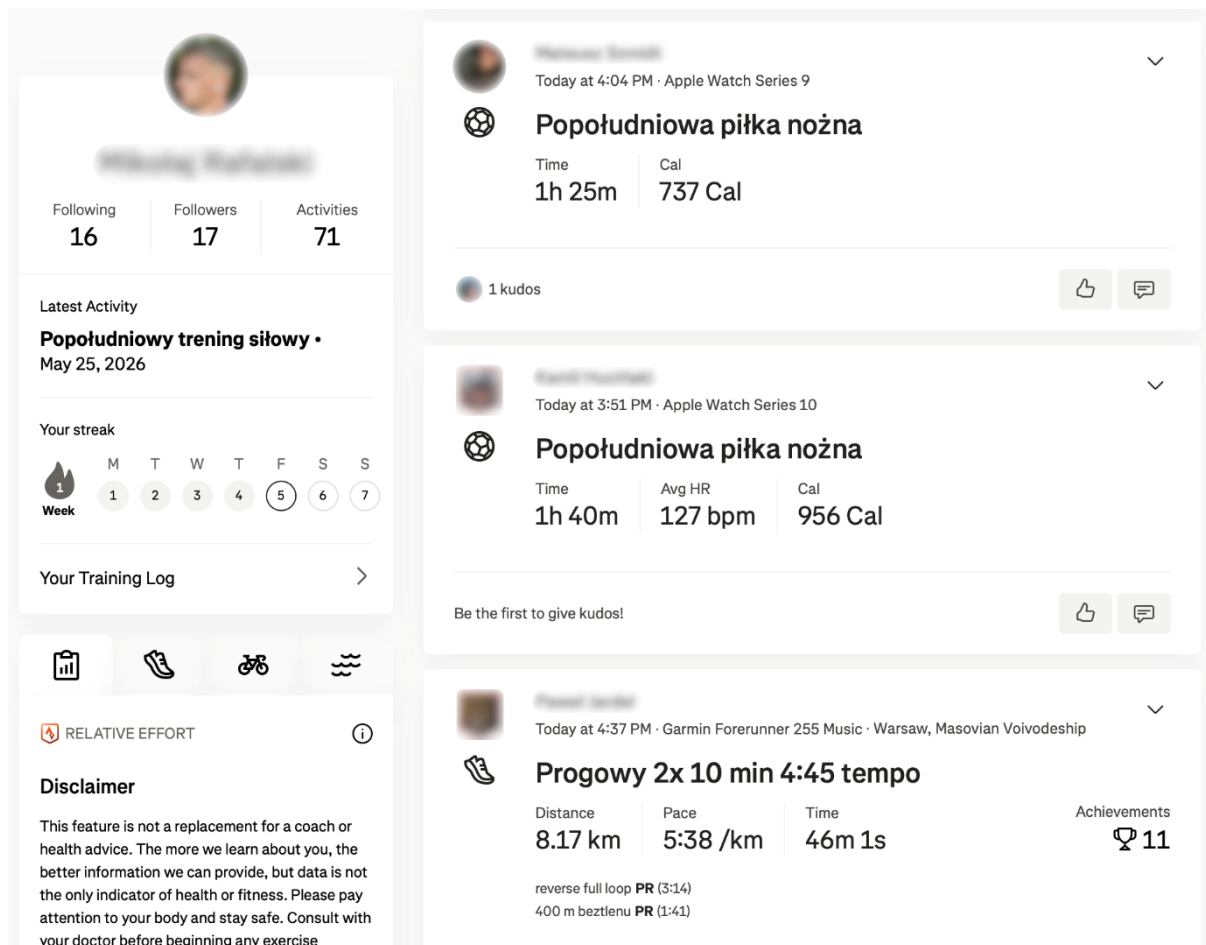
Rysunek 3. Strona powitalna komercyjnej platformy *SaaS* z wezwaniem do rejestracji - bariera przed dostępem do interfejsu. Źródło: [7].

Wspólną cechą tej kategorii jest model biznesowy obciążający trenera opłatą abonamentową, zależną najczęściej od liczby klientów, oraz brak otwartej platformy wyszukiwania - klient trafia do systemu z zaproszenia trenera, a nie w wyniku samodzielnego wyszukiwania. Dostęp do rzeczywistego interfejsu wymaga rejestracji, a często podania danych płatniczych w ramach okresu próbnego. Obecność panelu administratora w tych rozwiązaniach nie jest zwykle dokumentowana publicznie. Ograniczenia panelu administratora w systemie Coachify omówiono w podrozdziale 9.3.

2.2.3 Społecznościowe aplikacje treningowe

Społecznościowe aplikacje treningowe są istotne w kontekście planowanych kierunków rozwoju Coachify w obszarze zaangażowania użytkownika. Aplikacja Hevy służy do rejestrowania treningów siłowych i udostępnia rozbudowaną warstwę społecznościową: strumień aktywności (ang. *feed*), obserwowanie innych użytkowników, profile, rankingi najlepszych wyników w poszczególnych ćwiczeniach (ang. *leaderboard*), śledzenie postępów oraz zdjęcia postępów [13]. Aplikacja Strava, skupiona głównie na aktywnościach wytrzymałościowych, pełni funkcję sieci społecznościowej z wyzwaniem, rankingami i klubami, gromadząc ponad 100 milionów użytkowników [14]. Mechanizmy gamifikacji (poziomy, odznaki, wyzwania) spopularyzował również nieistniejący już serwis Fitocracy.

Przykładowy strumień aktywności oraz ranking wyników przedstawiono na rysunku 4.



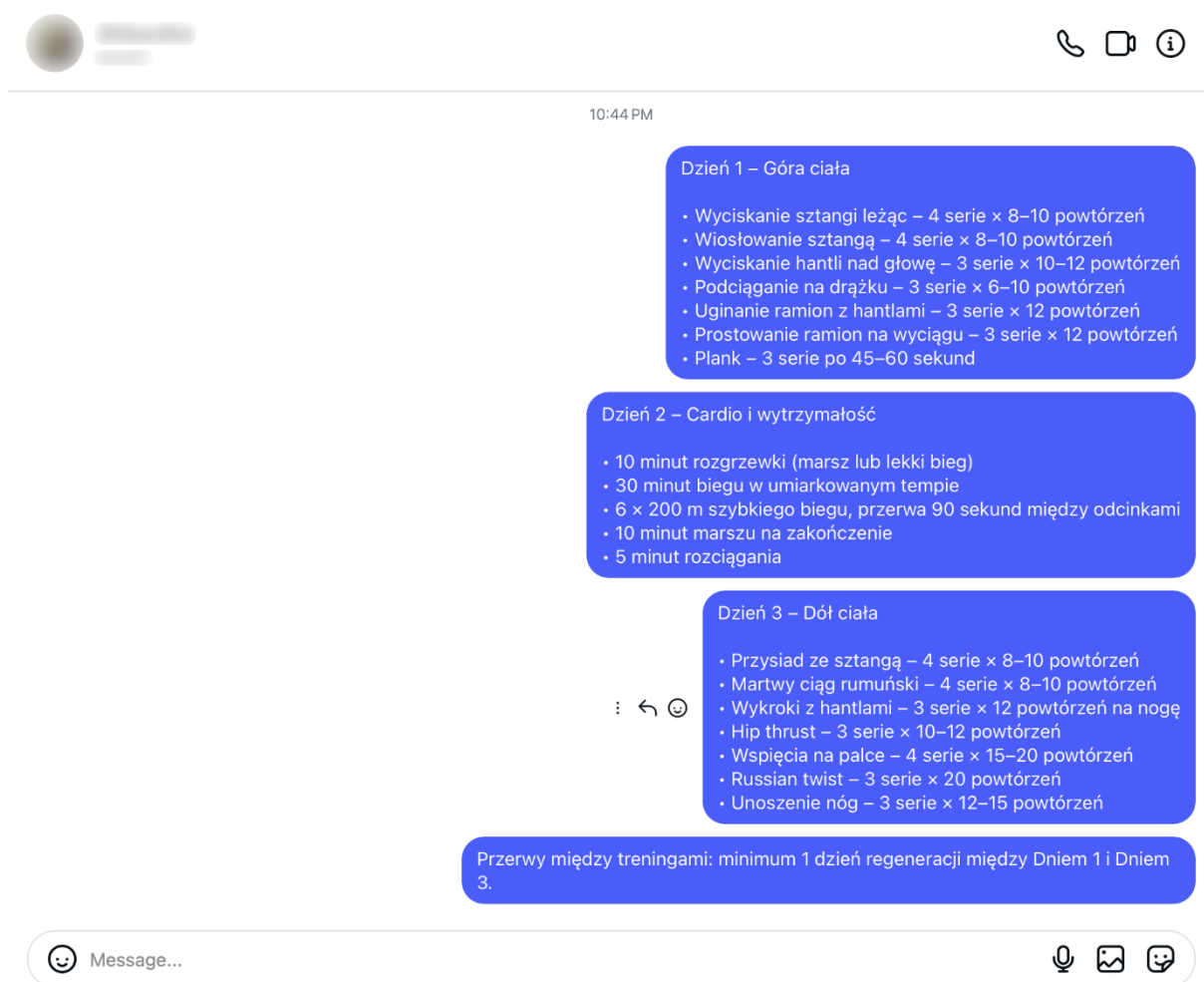
Rysunek 4. Strumień aktywności w aplikacji Strava - przykład warstwy społecznościowej i mechanizmów społecznościowych. Źródło: [14].

Wspólną cechą tej kategorii jest nacisk na samodzielne rejestrowanie aktywności oraz motywację społecznościową. Rozwiązania te nie obsługują jednak relacji trener-klient: nie umożliwiają przypisywania klientowi planu treningowego ani dietetycznego przez trenera, nie zawierają kwestionariusza wstępnego ani platformy wyszukiwania trenerów. Planowane w systemie Coachify elementy społecznościowe oraz gamifikację opisano w podrozdziale 9.4.

2.2.4 Stan obecny - narzędzia ogólnego przeznaczenia

Najczęściej stosowanym w praktyce rozwiązaniem pozostaje zestaw narzędzi ogólnego przeznaczenia: komunikatory (Instagram, WhatsApp) do kontaktu i przesyłania planów oraz arkusze kalkulacyjne i dokumenty do ich tworzenia. Narzędzia te są darmowe i powszechnie znane oraz nie wymagają od klienta zakładania dodatkowego konta, jednak nie zapewniają struktury danych, śledzenia postępów, archiwizacji historii współpracy ani możliwości porównania ofert trenerów. To właśnie ten rozproszony zestaw narzędzi, a nie wyspecjalizowane platformy, stanowi w praktyce główny punkt odniesienia dla większości trenerów rozpoczynających prowadzenie podopiecznych.

Przykład planu treningowego prowadzonego w narzędziach ogólnego przeznaczenia przedstawiono na rysunku 5.



Rysunek 5. Plan treningowy przesłany przez komunikator - zanonimizowany przykład rozproszonego prowadzenia podopiecznego. Źródło: opracowanie własne.

2.3. Porównanie funkcjonalności

Ze względu na odmienny charakter trzech kategorii produktowych porównanie przeprowadzono w trzech osobnych tabelach, z zastosowaniem kryteriów właściwych dla danej grupy. We wszystkich zestawieniach jako kolumnę wspólną umieszczono system Coachify. Zestawienia odzwierciedlają stan faktyczny prototypu - funkcje niezaimplementowane oznaczono symbolem „X”, niezależnie od tego, czy są przewidziane w dalszych pracach; kierunki rozwoju opisano w podrozdziale 9.4. Przyjęto trójstopniową skalę oceny: „✓” oznacza pełne wsparcie funkcji, „częściowo” - wsparcie ograniczone lub realizowane w odmiennym zakresie niż w systemie Coachify, a „X” - brak danej funkcji.

W tabeli 1 zestawiono porównanie z platformami wyszukiwania trenerów.

Tabela 1. Porównanie z platformami wyszukiwania trenerów. Źródło: opracowanie własne.

Kryterium	TrenerHUB	MamTrenera	TwójTrener	FindYourTrainer	Coachify
Przeglądanie bez oferty logowania	✓	✓	✓	X	✓

Brak opłat dla trenera	X	Częściowo	✓	✓	✓
Weryfikacja kompetencji trenera	✓	X	X	X	X
Wbudowany czat	X	X	X	X	✓
Kwestionariusz przekazywany trenerowi	X	X	X	X	✓
Kreator planów treningowych	X	X	X	X	✓
Plany dietetyczne	X	X	X	X	✓
Dziennik treningowy	X	X	X	X	✓

Serwis MamTrenera funkcjonuje w modelu freemium: konto standardowe jest bezpłatne, natomiast konto Pro zwiększa dostępne limity, m.in. liczbę dyscyplin, linków profilowych oraz pojemność galerii zdjęć. Z tego względu kryterium braku opłat po stronie trenera oznaczono dla tego serwisu jako „częściowo”.

W tabeli 2 zestawiono porównanie z narzędziami do prowadzenia klientów.

Tabela 2. Porównanie z narzędziami do prowadzenia klientów. Źródło: opracowanie własne.

Kryterium	Trainerize	TrueCoach	PT Distinction	CoachPro	Coachify
Otwarta forma wyszukiwania trenerów	✓	X	X	X	✓
Brak opłat po stronie trenera	Częściowo	X	X	X	✓
Kreator planów treningowych i szablony	✓	✓	✓	✓	✓
Plany dietetyczne	✓	✓	✓	X	✓
Dziennik treningowy	✓	✓	✓	X	✓
Kwestionariusz wstępny	X	X	X	X	✓

Natywna aplikacja mobilna	✓	✓	✓	✓	✗
Integracja z urządzeniami ubieralnymi	✓	✓	✓	✓	✗
Generator planów treningowych oparty na sztucznej inteligencji	✓	✗	✓	✗	✗

Platforma ABC Trainerize również działa w modelu *freemium* z wieloma wariantami cenowymi: plan podstawowy umożliwia tworzenie treningów dla klientów, jednak funkcje takie jak integracje z aplikacjami zdrowotnymi (np. Apple Health) czy generowanie szablonów wymagają wykupienia droższego planu. Z tego względu kryterium braku opłat po stronie trenera oznaczono dla tej platformy jako „częściowo”.

W tabeli 3 zestawiono porównanie ze społecznościowymi aplikacjami treningowymi.

Tabela 3. Porównanie ze społecznościowymi aplikacjami treningowymi. Źródło: opracowanie własne.

Kryterium	Hevy	Strava	Coachify
Strumień aktywności	✓	✓	✗
Gamifikacja	✓	✓	✓
Śledzenie postępów	✓	✓	✓
Relacja trener-klient	✗	✗	✓
Plan treningowy przypisywany przez trenera	✗	✗	✓
Plan dietetyczny przypisywany przez trenera	✗	✗	✓
Platforma wyszukiwania trenerów	✗	✗	✓

Zestawienia ujawniają spójny wzorzec: w każdej z trzech kategorii istnieją rozwiązania przewyższające Coachify w obrębie własnej specjalizacji, jednak żadne z analizowanych nie obejmuje równocześnie wszystkich trzech obszarów. Platformy wyszukiwania zapewniają etap znalezienia trenera, lecz nie prowadzenie; narzędzia *SaaS* zapewniają prowadzenie, lecz nie otwarty katalog; aplikacje społecznościowe zapewniają zaangażowanie, lecz nie obsługują relacji trener-klient. Spośród analizowanych rozwiązań Coachify wyróżnia się połączeniem otwartego katalogu trenerów z pełnym

środowiskiem prowadzenia podopiecznego. Funkcje oznaczone jako niezaimplementowane, przewidziane do dalszego rozwoju, opisano w podrozdziale 9.4.

2.4. Wnioski

Z przeprowadzonej analizy wynikają następujące wnioski dla projektu Coachify.

Zidentyfikowana luka rynkowa. Wśród przeanalizowanych rozwiązań nie zidentyfikowano systemu integrującego w jednym środowisku otwartą platformę wyszukiwania trenerów, pełne narzędzie prowadzenia podopiecznego (plany treningowe i dietetyczne, kwestionariusz wstępny, raporty) oraz podejście zorientowane na klienta. Analizowane produkty zajmują pojedyncze, wyspecjalizowane nisze. Wypełnienie tej luki - poprzez połączenie etapu wyszukiwania trenera z pełnym środowiskiem współpracy - stanowi podstawowy cel systemu Coachify. Wyróżnikiem nie jest przy tym sama bezpłatność (część platform wyszukiwania, np. TrenerHUB czy MamTrenera, jest darmowa dla użytkowników), lecz integracja funkcji rozproszonych obecnie pomiędzy trzy odrębne kategorie produktów.

Odmienny model biznesowy. W przeciwieństwie do narzędzi *SaaS*, w których koszty ponosi trener, Coachify przyjmuje model platformy pośredniczącej z monetyzacją po stronie transakcji oraz opcjonalnych funkcji premium. Korzystanie z podstawowych funkcji platformy pozostaje bezpłatne dla obu stron; docelowo planowana jest prowizja od rozliczeń realizowanych bezpośrednio między klientem a trenerem (trener samodzielnie ustala cenę usług), a nie opłata za samo korzystanie z platformy. Możliwość rozszerzenia systemu o obsługę płatności wskazano w podrozdziale 1.2, a planowaną funkcję premium opartą na sztucznej inteligencji opisano w podrozdziale 9.4.

Podejście zorientowane na klienta. Część rozwiązań traktuje klienta jako odbiorcę planu bez dalszego zaangażowania. Coachify udostępnia klientowi dedykowane środowisko obejmujące przypisane plany, dziennik treningowy, śledzenie postępów oraz zdjęcia, dzięki czemu klient nie musi prowadzić odrębnych notatek, a trener uzyskuje bieżący wgląd w realizację planu i może go modyfikować. Poszczególne moduły klienta i trenera opisano w podrozdziałach 3.3.1-3.3.6.

Strukturalizacja informacji wstępnych. W praktyce trener często musi samodzielnie dopytywać klienta o szczegóły, a klient nie zawsze wie, jakie informacje są istotne. Coachify wprowadza wieloetapowy kwestionariusz wstępny, którego ukończenie warunkuje wysłanie zgłoszenia o współpracę, dzięki czemu trener otrzymuje komplet danych przed podjęciem decyzji o przyjęciu klienta. Szczegółowy opis kwestionariusza zamieszczono w podrozdziale 3.3.1, a definicje pojęć związanych z procesem *onboardingu* - w słowniku pojęć w podrozdziale 3.8.

Świadome ograniczenia zakresu prototypu. Analiza wykazała, że rozwiązania komercyjne zachowują przewagę w obszarach wymagających długiego czasu rozwoju: natywnych aplikacji mobilnych, integracji z urządzeniami ubieralnymi, generatorów opartych na sztucznej inteligencji oraz obsługi płatności. Zakres prototypu został świadomie skoncentrowany na obszarach stanowiących o jego oryginalności - integracji platformy wyszukiwania i środowiska prowadzenia oraz podejściu zorientowanym na klienta. Brak aplikacji mobilnej i wynikające z niego konsekwencje (aplikacja jednostronicowa, (ang. *Single-Page Application*, *SPA*); uruchomienie we własnej infrastrukturze, (ang. *on-premise*) omówiono w kontekście ograniczeń infrastrukturalnych w podrozdziale 8.6, a pełny wykaz ograniczeń prototypu - w podrozdziale 9.3.

Kierunki rozwoju. Zidentyfikowane luki funkcjonalne wyznaczają kierunki dalszego rozwoju systemu - między innymi generator planów oparty na sztucznej inteligencji, automatyczne dopasowanie trenera do podopiecznego, aplikację mobilną, rozbudowę panelu administratora, warstwę społecznościową oraz weryfikację kompetencji trenerów. Niezrealizowane funkcje i kierunki rozwoju opisano szczegółowo w podrozdziale 9.4.

3. Motywacja i założenia rozwiązania

W niniejszym rozdziale przedstawiono motywację opracowania systemu Coachify oraz założenia funkcjonalne i organizacyjne przyjęte dla projektowanej platformy. Opisano system z perspektywy użytkownika, role występujące w aplikacji, główne moduły funkcjonalne, wymagania funkcjonalne i niefunkcjonalne, aktorów oraz wybrane przypadki użycia. Rozdział obejmuje również model danych i słownik pojęć stosowanych w dalszych częściach pracy.

System Coachify został zaprojektowany jako platforma wspierająca współpracę trenerów personalnych z klientami. Głównym założeniem było połączenie w jednym środowisku funkcji, które w typowym procesie współpracy są rozproszone między komunikatorami, arkuszami kalkulacyjnymi, dokumentami tekstowymi, aplikacjami do przechowywania zdjęć oraz ręcznie prowadzonymi dziennikami treningowymi. W rezultacie system koncentruje się na relacji trener-klient: od znalezienia trenera, przez wypełnienie kwestionariusza, utworzenie planu treningowego lub dietetycznego, aż po raportowanie wykonanych jednostek treningowych i bieżącą komunikację.

3.1. Opis systemu z perspektywy użytkownika

Z perspektywy użytkownika Coachify jest aplikacją webową dostępną przez przeglądarkę internetową. Użytkownik niezarejestrowany może przejść do publicznego *marketplace'u* trenerów, zapoznać się z ofertami oraz utworzyć konto. Po zalogowaniu zakres dostępnych funkcji zależy od roli przypisanej do konta.

Klient korzysta z aplikacji przede wszystkim jako z panelu współpracy z trenerem. Po rejestracji uzupełnia dane osobowe oraz kwestionariusz treningowy obejmujący m.in. cel treningowy, lokalizację ćwiczeń, dostępny sprzęt, poziom doświadczenia, preferowaną częstotliwość treningów, kontuzje, ulubione i niechciane ćwiczenia oraz dodatkowe uwagi. Następnie może przeglądać *marketplace*, wysłać zgłoszenie o współpracę do wybranego trenera, odczytywać przypisane plany treningowe i dietetyczne, raportować wykonane treningi, dołączać zdjęcia postępów oraz komunikować się z trenerem przez czat.

Trener korzysta z systemu jako z narzędzia do obsługi podopiecznych. Może konfigurować profil i ofertę usług, decydować o widoczności w *marketplace*, akceptować lub odrzucać zgłoszenia, przeglądać kwestionariusze klientów, przygotowywać plany treningowe i dietetyczne, tworzyć szablony planów oraz posiłków, zarządzać biblioteką ćwiczeń i analizować raporty treningowe. Dodatkowo trener obsługuje prośby o wiadomość od klientów, którzy nie są jeszcze jego podopiecznymi.

Administrator ma dostęp do panelu administracyjnego, w którym przegląda listy trenerów i klientów oraz zarządza globalną biblioteką ćwiczeń dostępną dla wszystkich trenerów.

3.2. Role w systemie

W systemie zdefiniowano trzy role użytkowników zalogowanych oraz jednego aktora niezalogowanego. Role określają dostęp do widoków, operacji biznesowych oraz danych. Zestawienie ról przedstawiono w tabeli 4.

Tabela 4. Role użytkowników w systemie Coachify. Źródło: opracowanie własne.

Rola	Charakterystyka	Główne uprawnienia
Użytkownik niezarejestrowany	Osoba odwiedzająca aplikację bez aktywnej sesji.	Rejestracja, logowanie, przeglądanie publicznego <i>marketplace'u</i> trenerów.
Klient	Podopieczny korzystający z usług trenera personalnego.	Uzupełnienie profilu i kwestionariusza, wysłanie zgłoszenia o współpracę, podgląd planów, raportowanie treningów, czat z trenerem.
Trener	Użytkownik świadczący usługi treningowe lub dietetyczne.	Zarządzanie profilem i ofertą, obsługa zgłoszeń, tworzenie planów i szablonów, przegląd raportów, zarządzanie ćwiczeniami prywatnymi, obsługa próśb o wiadomość.
Administrator	Użytkownik nadzorujący wybrane zasoby wspólne systemu.	Przegląd kont trenerów i klientów, zarządzanie globalną biblioteką ćwiczeń, dostęp do panelu administracyjnego.

Podział ról jest istotny z punktu widzenia bezpieczeństwa i przejrzystości interfejsu. Klient nie ma dostępu do narzędzi tworzenia planów, trener nie ma uprawnień administracyjnych, a administrator nie uczestniczy w prywatnej komunikacji trener-klient. Każdy panel prezentuje wyłącznie funkcje właściwe dla danej odpowiedzialności.

3.3. Główne moduły funkcjonalne

Zakres systemu Coachify podzielono na dziewięć głównych modułów funkcjonalnych. Moduły nie są rozumiane jako pojedyncze ekrany aplikacji, lecz jako obszary odpowiedzialności systemu powiązane przez role użytkowników, relację trener-klient oraz obiekty przypisywane do kont. Podział ten odpowiada podstawowym etapom korzystania z platformy: utworzeniu konta, zebraniu danych klienta, przygotowaniu planów, raportowaniu realizacji treningów, obsłudze multimedialnych, komunikacji, wyszukiwaniu trenerów oraz administracji danymi wspólnymi.

3.3.1 Rejestracja, logowanie i profile użytkowników

Moduł rejestracji, logowania i profili odpowiada za identyfikację użytkowników oraz przypisanie im właściwego zakresu uprawnień. System umożliwia utworzenie konta klienta albo trenera, natomiast rola administratora nie jest dostępna przez publiczny proces rejestracji. Takie rozdzielenie ogranicza ryzyko nieuprawnionego dostępu do funkcji nadzorczych.

Kluczowymi obiektami modułu są konto użytkownika, dane uwierzytelniające, rola systemowa oraz profil. Rola określa, które operacje mogą zostać wykonane przez użytkownika, a profil przechowuje dane potrzebne w dalszych procesach systemu. Profil klienta zawiera informacje podstawowe wykorzystywane w toku współpracy, natomiast profil trenera obejmuje dane prezentujące jego doświadczenie, zakres usług oraz dostępność oferty.

Moduł rozwiązuje problem kontrolowanego dostępu do funkcji zależnych od roli. Klient korzysta z planów, raportów, zgłoszeń i komunikacji, trener zarządza współpracą z klientami, planami oraz szablonami, a administrator odpowiada za wybrane zasoby wspólne. Dostęp do danych jest ograniczany zgodnie z rolą oraz powiązaniem wynikającymi z relacji trener-klient.

3.3.2 Kwestionariusz podopiecznego

Moduł kwestionariusza podopiecznego służy do zebrania uporządkowanych danych wejściowych przed rozpoczęciem współpracy z trenerem. Rozwiązuje problem braku spójnych informacji o celach, ograniczeniach, doświadczeniu i preferencjach klienta, które są potrzebne do oceny zgłoszenia oraz przygotowania planu.

Kluczowymi obiektami modułu są kwestionariusz, odpowiedzi klienta, cel treningowy, preferencje, ograniczenia zdrowotne, dostępny sprzęt, miejsce treningu oraz status ukończenia. Dane obejmują między innymi parametry fizyczne, poziom doświadczenia, aktualną i planowaną częstotliwość treningów, preferowane partie mięśniowe, kontuzje, ćwiczenia preferowane i niechciane oraz dodatkowe uwagi.

System przechowuje wersję roboczą kwestionariusza, dzięki czemu uzupełnianie danych może być kontynuowane po przerwaniu procesu. Ukończenie kwestionariusza stanowi warunek wysłania zgłoszenia o współpracę. Dzięki temu trener otrzymuje podstawowe informacje przed podjęciem decyzji o przyjęciu klienta oraz przed przygotowaniem zaleceń treningowych lub dietetycznych.

3.3.3 Plany treningowe - szablony i przypisania

Moduł planów treningowych umożliwia trenerowi tworzenie programów treningowych dla konkretnych klientów oraz przygotowywanie szablonów wielokrotnego użytku. Rozwiązuje problem powtarzalnego układania podobnych struktur treningowych, przy jednoczesnej potrzebie dostosowania planu do indywidualnych możliwości podopiecznego.

Najważniejszymi obiektami modułu są plan treningowy, szablon planu, przypisanie planu, dzień planu, ćwiczenie oraz parametry wykonania. Plan ma strukturę tygodniową, w której poszczególne dni mogą zawierać jednostki treningowe albo pełnić funkcję dni odpoczynku. Ćwiczenie w planie może mieć określoną liczbę serii, powtórzeń, obciążenie, czas, przerwy oraz notatki trenera.

Ćwiczenia wykorzystywane w planach mogą pochodzić z biblioteki globalnej albo z prywatnej biblioteki trenera. Szablon planu nie jest bezpośrednim planem klienta, lecz wzorcem używanym jako punkt wyjścia. Po przypisaniu planu klientowi powstaje niezależny obiekt, który może zostać zmodyfikowany bez wpływu na szablon bazowy. Klient ma dostęp do przypisanych planów, ale nie modyfikuje ich struktury.

3.3.4 Plany dietetyczne - szablony, posiłki i produkty

Moduł planów dietetycznych odpowiada za przygotowywanie zaleceń żywieniowych dla klientów. Jego celem jest uporządkowanie diety w strukturze możliwej do wielokrotnego wykorzystania, a jednocześnie pozwalającej na indywidualne przypisanie planu konkretnej osobie.

Kluczowymi obiektami modułu są plan dietetyczny, szablon diety, przypisanie planu, dzień diety, posiłek, szablon posiłku, produkt lub składnik oraz wartości odżywcze. Plan dietetyczny jest dzielony na dni, a dni zawierają posiłki składające się z produktów, ilości, jednostek miary oraz informacji o kaloryczności i makroskładnikach.

Szablony diet i posiłków pełnią funkcję wzorców. Trener może wykorzystać je podczas tworzenia planu dla klienta, a następnie dostosować wynikowy plan do potrzeb konkretnego podopiecznego. Edycja przypisanego planu nie narusza szablonu, dzięki czemu te same wzorce mogą być używane w pracy z wieloma klientami.

3.3.5 Raporty z treningów i śledzenie postępów

Moduł raportowania służy do dokumentowania wykonania treningów oraz monitorowania postępów klienta. Rozwiązuje problem braku informacji zwrotnej po realizacji planu, ponieważ pozwala zestawić założenia przygotowane przez trenera z rzeczywistym przebiegiem jednostki treningowej.

Podstawowymi obiektami modułu są raport treningowy, dzień planu, wykonane ćwiczenie, seria raportowana, status wykonania, notatka oraz zdjęcie postępów. Raport jest powiązany z konkretnym planem i dniem treningowym, co zachowuje kontekst wykonania. Klient może zapisać rzeczywiste wartości, takie jak liczba powtórzeń, obciążenie, czas wykonania albo liczba serii.

System umożliwia oznaczenie ćwiczeń lub serii jako pominiętych oraz dodanie notatek dotyczących całego treningu albo pojedynczych ćwiczeń. Trener może analizować raporty swoich klientów i na tej podstawie oceniać regularność, objętość oraz jakość realizacji planu. Do raportu można dołączyć ograniczoną liczbę zdjęć postępów, które uzupełniają dane liczbowe o materiał porównawczy.

3.3.6 Multimedia - zdjęcia postępów, awatary i materiały wideo ćwiczeń

Moduł multimediów odpowiada za obsługę plików wykorzystywanych w innych częściach systemu. Obejmuje awatary użytkowników, zdjęcia postępów oraz materiały wideo powiązane z ćwiczeniami. Jego celem jest uporządkowanie plików przez przypisanie ich do konkretnych obiektów domenowych.

Kluczowymi obiektami modułu są plik multimedialny, kategoria pliku, właściciel, metadane oraz obiekt nadrzędny. Awatar jest powiązany z profilem użytkownika, zdjęcie postępów z raportem klienta, a materiał wideo z ćwiczeniem. Każda kategoria plików ma odmienne przeznaczenie oraz odmienne ograniczenia funkcjonalne.

Reguły modułu obejmują kontrolę typu, rozmiaru i poprawności przesyłanych materiałów. Dostęp do pliku wynika z kontekstu biznesowego: klient korzysta z własnych materiałów, trener z plików powiązanych z jego ćwiczeniami i klientami, a administrator z zasobów wspólnych w zakresie funkcji administracyjnych.

3.3.7 Czat

Moduł czatu umożliwia komunikację tekstową jeden-do-jednego między trenerem i klientem. Rozwiązuje problem rozproszonego kontaktu poza systemem, zapewniając miejsce do omawiania planów, raportów, pytań organizacyjnych oraz korekt w toku współpracy.

Podstawowymi obiektami modułu są konwersacja, wiadomość, uczestnik rozmowy, status odczytu, licznik nieprzeczytanych wiadomości, prośba o kontakt oraz blokada użytkownika. Konwersacja jest przypisana do jednej pary użytkowników. Standardowo rozmowa jest dostępna po nawiązaniu relacji trener-klient.

System obsługuje również pierwszy kontakt przed rozpoczęciem współpracy. Klient może wysłać prośbę o wiadomość, a trener może ją zaakceptować, odrzucić albo zablokować użytkownika. Do momentu akceptacji prośby pełna rozmowa nie jest prowadzona. Zakres czatu obejmuje wiadomości tekstowe i potwierdzenia odczytu, bez załączników, edycji wysłanych wiadomości oraz wskaźnika pisanie.

3.3.8 Marketplace

Marketplace jest publiczną częścią systemu służącą do prezentowania trenerów oraz inicjowania współpracy. Moduł rozwiązuje problem odnalezienia trenera przez klienta i uporządkowania procesu przejścia od zapoznania się z ofertą do utworzenia relacji trener-klient.

Kluczowymi obiektami modułu są profil trenera, oferta, typ usługi, cena, opis usługi, widoczność profilu, zgłoszenie o współpracę, status zgłoszenia oraz relacja trener-klient. Trener jest dostępny w *marketplace* tylko wtedy, gdy ma włączoną widoczność profilu i posiada co najmniej jedną aktywną ofertę. Oferta może dotyczyć planu treningowego, planu dietetycznego albo pełnego prowadzenia.

Marketplace nie pełni funkcji sklepu internetowego i nie obsługuje płatności. Jego zadaniem jest prezentowanie ofert, umożliwienie filtrowania trenerów według typu usługi oraz rozpoczęcie kontaktu lub zgłoszenia o współpracę. Akceptacja zgłoszenia przez trenera tworzy aktywną relację trener-klient,

która pozwala korzystać z modułów zależnych od współpracy, takich jak plany, raporty i standardowa komunikacja.

3.3.9 Panel administratora

Moduł administratora służy do nadzoru nad wybranymi danymi wspólnymi oraz podstawowymi informacjami o użytkownikach systemu. Rozwiązuje problem utrzymania zasobów, które nie należą do pojedynczego trenera, lecz powinny być dostępne w całej platformie w spójnej postaci.

Kluczowymi obiektami modułu są administrator, użytkownik, trener, klient, ćwiczenie globalne oraz relacja trener-klient. Najważniejszym zasobem zarządzanym przez administratora jest globalna biblioteka ćwiczeń. Ćwiczenia globalne mogą być wykorzystywane przez wszystkich trenerów podczas tworzenia planów treningowych i szablonów.

Zakres administracji ma charakter nadzorczy i porządkujący. Administrator może zarządzać ćwiczeniami globalnymi oraz przeglądać podstawowe dane o trenerach i klientach, jednak nie przejmuje obowiązków trenera, nie prowadzi prywatnej komunikacji z klientami, nie modyfikuje planów podopiecznych i nie obsługuje płatności.

3.4. Wymagania funkcjonalne

Wymagania funkcjonalne opisują zachowania, które system udostępnia użytkownikom. Zostały pogrupowane według obszaru odpowiedzialności i powiązane z rolami korzystającymi z danej funkcji.

Tabela 5 zawiera zestawienie wymagań funkcjonalnych systemu Coachify.

Tabela 5. Wymagania funkcjonalne systemu Coachify. Źródło: opracowanie własne.

Identyfikator	Nazwa wymagania	Opis	Rola główna	Priorytet
WF-01	Rejestracja konta	System umożliwia utworzenie konta klienta albo trenera z użyciem formularza rejestracji.	Użytkownik niezarejestrowany	Wysoki
WF-02	Logowanie i wylogowanie	System umożliwia rozpoczęcie i zakończenie sesji użytkownika.	Klient, trener, administrator	Wysoki
WF-03	Zarządzanie profilem klienta	System umożliwia klientowi uzupełnienie danych osobowych i sprawdzenie statusu <i>onboarding</i> .	Klient	Wysoki
WF-04	Zarządzanie profilem trenera	System umożliwia trenerowi edycję danych profilu, opisu, widoczności w <i>marketplace</i> oraz oferty usług.	Trener	Wysoki

WF-05	Wypełnienie kwestionariusza	System umożliwia klientowi wypełnienie wieloetapowego kwestionariusza i zapis wersji roboczej.	Klient	Wysoki
WF-06	Przeglądanie <i>marketplace'u</i>	System umożliwia przeglądanie publicznej listy trenerów oraz szczegółów ich ofert.	Użytkownik niezarejestrowany, klient	Wysoki
WF-07	Filtrowanie ofert	System umożliwia filtrowanie trenerów według typu oferowanej usługi.	Użytkownik niezarejestrowany, klient	Średni
WF-08	Wysłanie zgłoszenia o współpracę	System umożliwia klientowi wysłanie zgłoszenia do trenera po ukończeniu wymaganych danych i kwestionariusza.	Klient	Wysoki
WF-09	Obsługa zgłoszeń przez trenera	System umożliwia trenerowi zaakceptowanie albo odrzucenie zgłoszenia klienta.	Trener	Wysoki
WF-10	Utworzenie relacji trener-klient	System tworzy aktywne przypisanie klienta do trenera po zaakceptowaniu zgłoszenia.	Trener, klient	Wysoki
WF-11	Tworzenie planów treningowych	System umożliwia trenerowi przygotowanie planu treningowego dla konkretnego klienta.	Trener	Wysoki
WF-12	Szablony planów treningowych	System umożliwia trenerowi tworzenie i ponowne wykorzystywanie szablonów planów treningowych.	Trener	Średni
WF-13	Podgląd planu treningowego	System umożliwia klientowi odczytanie przypisanego planu treningowego.	Klient	Wysoki
WF-14	Tworzenie planów dietetycznych	System umożliwia trenerowi przygotowanie planu dietetycznego dla konkretnego klienta.	Trener	Wysoki

WF-15	Szablony diet i posiłków	System umożliwia trenerowi tworzenie szablonów planów dietetycznych oraz szablonów posiłków.	Trener	Średni
WF-16	Podgląd planu dietetycznego	System umożliwia klientowi odczytanie przypisanego planu dietetycznego.	Klient	Wysoki
WF-17	Raportowanie treningu	System umożliwia klientowi utworzenie raportu z wykonanego treningu na podstawie dnia planu.	Klient	Wysoki
WF-18	Zdjęcia postępów	System umożliwia klientowi dołączenie zdjęć postępów do raportu treningowego z zachowaniem limitu liczby plików.	Klient	Średni
WF-19	Przegląd raportów klienta	System umożliwia trenerowi przeglądanie raportów treningowych swoich klientów.	Trener	Wysoki
WF-20	Biblioteka ćwiczeń trenera	System umożliwia trenerowi zarządzanie prywatnymi ćwiczeniami i korzystanie z ćwiczeń globalnych.	Trener	Wysoki
WF-21	Multimedia ćwiczeń	System umożliwia dodawanie materiałów wideo i miniatur powiązanych z ćwiczeniami.	Trener, administrator	Średni
WF-22	Czat trener-klient	System umożliwia prowadzenie konwersacji tekstowej między trenerem i klientem.	Trener, klient	Wysoki
WF-23	Prośby o wiadomość	System umożliwia klientowi wysłanie prośby o wiadomość do trenera oraz trenerowi zaakceptowanie, zignorowanie albo zablokowanie takiej prośby.	Klient, trener	Średni
WF-24	Panel administracyjny	System umożliwia administratorowi przegląd trenerów i klientów oraz zarządzanie ćwiczeniami globalnymi.	Administrator	Wysoki

Najwyższy priorytet przypisano funkcjom, które były niezbędne do zrealizowania głównego procesu współpracy: rejestracji, *onboardingu*, zgłoszenia, planów, raportów oraz komunikacji.

3.5. Wymagania niefunkcjonalne

Wymagania niefunkcjonalne określają oczekiwane cechy jakościowe systemu. Obejmują bezpieczeństwo, integralność danych, wydajność, obserwowalność, utrzymywalność, dostępność interfejsu oraz ograniczenia dotyczące plików. Zestawienie przedstawiono w tabelach 6-9.

Tabela 6. Wymagania niefunkcjonalne systemu Coachify - bezpieczeństwo i kontrola dostępu. Źródło: opracowanie własne.

Identyfikator	Obszar	Wymaganie	Uzasadnienie
WN-01	Bezpieczeństwo sesji	System powinien uwierzytelniać użytkowników bez utrzymywania sesji serwerowej, z wykorzystaniem krótkotrwałego tokenu dostępowego i tokenu odświeżającego.	Ogranicza zależność od stanu serwera i ułatwia wdrożenie w środowisku kontenerowym.
WN-02	Ochrona haseł	System powinien przechowywać hasła wyłącznie w postaci skrótów kryptograficznych.	Ogranicza skutki ewentualnego wycieku danych uwierzytelniających.
WN-03	Ograniczenie prób logowania	System powinien limitować liczbę prób logowania z jednego źródła.	Zmniejsza ryzyko ataków siłowych na formularz logowania.
WN-04	Kontrola dostępu	System powinien egzekwować dostęp do funkcji na podstawie roli użytkownika.	Zapobiega dostępowi klienta do funkcji trenera, trenera do funkcji administratora oraz administratora do prywatnych procesów współpracy.

Tabela 7. Wymagania niefunkcjonalne systemu Coachify - dane i współbieżność. Źródło: opracowanie własne.

Identyfikator	Obszar	Wymaganie	Uzasadnienie
WN-05	Spójność danych	System powinien utrzymywać spójny schemat bazy danych przez migracje i walidację zgodności modelu z bazą.	Ułatwia powtarzalne wdrożenia i ogranicza ryzyko rozjazdu schematów między środowiskami.
WN-06	Współbieżność	System powinien wykrywać konflikty współbieżnych modyfikacji kluczowych encji.	Chroni przed niezamierzonym nadpisaniem zmian w zgłoszeniach, planach, raportach i konwersacjach.

Tabela 8. Wymagania niefunkcjonalne systemu Coachify - wydajność, pliki i infrastruktura. Źródło: opracowanie własne.

Identyfikator	Obszar	Wymaganie	Uzasadnienie
WN-07	Wydajność odczytu	System powinien stosować stronicowanie oraz ograniczenia rozmiaru odpowiedzi listowych.	Utrzymuje przewidywalny koszt zapytań dla <i>marketplace'u</i> , list konwersacji, list klientów i panelu administratora.
WN-08	Obsługa plików	System powinien walidować typ, rozmiar i zawartość przesyłanych plików.	Ogranicza ryzyko przesłania niedozwolonych lub złośliwych plików.
WN-09	Przechowywanie multimediów	System powinien oddzielać pliki binarne od danych transakcyjnych.	Pozwala przechowywać multimedia w magazynie obiektowym, a metadane w bazie relacyjnej.
WN-10	Obserwowalność	System powinien udostępniać informacje o stanie aplikacji oraz metryki operacyjne.	Ułatwia diagnozowanie problemów w środowisku wdrożeniowym.
WN-11	Spójny format błędów	System powinien zwracać błędy w ujednoliconym formacie.	Ułatwia obsługę błędów po stronie aplikacji klienckiej i powiązanie zgłoszenia użytkownika z logami.

Tabela 9. Wymagania niefunkcjonalne systemu Coachify - interfejs użytkownika. Źródło: opracowanie własne.

Identyfikator	Obszar	Wymaganie	Uzasadnienie
WN-12	Responsywność interfejsu	Interfejs użytkownika powinien działać na urządzeniach mobilnych i desktopowych.	Użytkownicy mogą korzystać z systemu zarówno przy biurku, jak i bezpośrednio po treningu.
WN-13	Dostępność interfejsu	Komponenty interfejsu powinny wspierać semantykę WAI-ARIA i obsługę klawiatury tam, gdzie jest to wymagane.	Zwiększa dostępność aplikacji i zmniejsza ryzyko błędów w komponentach modalnych, zakładkach i formularzach.

Wymagania niefunkcjonalne wynikają z charakteru systemu, który przechowuje dane osobowe, informacje o aktywności treningowej, treść prywatnej komunikacji oraz pliki użytkowników. Z tego względu szczególny nacisk położono na kontrolę dostępu, walidację danych, odporność na błędy współbieżności i przewidywalność działania w środowisku wdrożeniowym.

3.6. Aktorzy i wybrane przypadki użycia

Aktorzy systemu odpowiadają rolom opisanym w podrozdziale 3.2, jednak w analizie przypadków użycia uwzględniono również kontekst działania: użytkownik niezarejestrowany rozpoczyna kontakt z systemem, klient inicjuje współpracę, trener prowadzi proces obsługi, a administrator utrzymuje wspólne zasoby. Tabela 10 zawiera syntetyczne zestawienie aktorów.

Tabela 10. Aktorzy systemu Coachify. Źródło: opracowanie własne.

Aktor	Cel w systemie	Przykładowe czynności
Użytkownik niezarejestrowany	Zapoznanie się z platformą i rozpoczęcie korzystania.	Przegląd <i>marketplace'u</i> , rejestracja, logowanie.
Klient	Uzyskanie wsparcia trenera i realizacja planu.	Wypełnienie kwestionariusza, wysłanie zgłoszenia, przegląd planów, raportowanie treningów, rozmowa z trenerem.
Trener	Prowadzenie klientów i zarządzanie ofertą.	Konfiguracja profilu, obsługa zgłoszeń, tworzenie planów, analiza raportów, czat.
Administrator	Utrzymanie zasobów wspólnych i wgląd operacyjny.	Przegląd użytkowników, zarządzanie ćwiczeniami globalnymi.

Wybrane przypadki użycia obejmują procesy kluczowe dla działania platformy. Nie przedstawiają wszystkich operacji CRUD, lecz te scenariusze, które najlepiej opisują przepływ wartości między użytkownikami.

Tabele 11-19 prezentują scenariusze przypadków użycia.

Tabela 11. Wybrane przypadki użycia systemu Coachify - rejestracja konta. Źródło: opracowanie własne.

Pole	Opis
Aktor główny	Użytkownik niezarejestrowany
Warunki początkowe	Użytkownik nie ma aktywnej sesji.
Przebieg główny	Użytkownik otwiera formularz, wybiera rolę klienta albo trenera, podaje wymagane dane i zatwierdza formularz.
Przebieg alternatywny	Jeżeli dane są niepoprawne lub e-mail jest zajęty, system zwraca komunikat walidacyjny.
Rezultat	Konto użytkownika zostaje utworzone.

Tabela 12. Wybrane przypadki użycia systemu Coachify - wypełnianie kwestionariusza. Źródło: opracowanie własne.

Pole	Opis
Aktor główny	Klient
Warunki początkowe	Klient jest zalogowany i posiada konto w roli klienta.
Przebieg główny	Klient przechodzi przez kolejne kroki kreatora, zapisuje odpowiedzi i kończy kwestionariusz.
Przebieg alternatywny	Jeżeli klient przerwie pracę, system zachowuje wersję roboczą i pozwala wznowić niedokończony krok.
Rezultat	Kwestionariusz otrzymuje status ukończony.

Tabela 13. Wybrane przypadki użycia systemu Coachify - nawiązanie współpracy. Źródło: opracowanie własne.

Pole	Opis
Aktor główny	Klient
Warunki początkowe	Klient ukończył wymagane dane i kwestionariusz; trener jest widoczny w <i>marketplace</i> i ma odpowiednią ofertę.
Przebieg główny	Klient wybiera trenera i typ usługi, a następnie wysyła zgłoszenie.
Przebieg alternatywny	Jeżeli klient ma aktywną współpracę albo oczekujące zgłoszenie, system odrzuca operację.
Rezultat	Trener otrzymuje zgłoszenie w stanie oczekującym.

Tabela 14. Wybrane przypadki użycia systemu Coachify - akceptacja zgłoszenia. Źródło: opracowanie własne.

Pole	Opis
Aktor główny	Trener
Warunki początkowe	Trener jest zalogowany i posiada oczekujące zgłoszenie od klienta.
Przebieg główny	Trener otwiera listę zgłoszeń, przegląda dane klienta i akceptuje zgłoszenie.
Przebieg alternatywny	Trener może odrzucić zgłoszenie, co kończy proces bez utworzenia współpracy.
Rezultat	System tworzy aktywne przypisanie klienta do trenera oraz konwersację.

Tabela 15. Wybrane przypadki użycia systemu Coachify - plany i raporty treningowe. Źródło: opracowanie własne.

Pole	Opis
Aktor główny	Trener
Warunki początkowe	Istnieje aktywna relacja trener-klient.
Przebieg główny	Trener wybiera klienta, definiuje strukturę dni i ćwiczeń, opcjonalnie korzysta z szablonu i zapisuje plan.
Przebieg alternatywny	Jeżeli plan zawiera niedostępne ćwiczenie, system odrzuca zapis.
Rezultat	Klient może przeglądać przypisany plan treningowy.

Tabela 16. Wybrane przypadki użycia systemu Coachify - utworzenie raportu. Źródło: opracowanie własne.

Pole	Opis
Aktor główny	Klient
Warunki początkowe	Klient posiada przypisany plan treningowy.
Przebieg główny	Klient wybiera dzień planu, wprowadza wykonane serie, oznacza pominięcia i zapisuje raport.
Przebieg alternatywny	Klient może dołączyć zdjęcia postępów; przekroczenie limitu plików powoduje odrzucenie załączania.

Rezultat	Raport jest dostępny dla klienta i trenera.
----------	---

Tabela 17. Wybrane przypadki użycia systemu Coachify - wysłanie wiadomości. Źródło: opracowanie własne.

Pole	Opis
Aktor główny	Klient albo trener
Warunki początkowe	Istnieje konwersacja między klientem i trenerem.
Przebieg główny	Użytkownik otwiera czat, wpisuje wiadomość i wysyła ją.
Przebieg alternatywny	Jeżeli połączenie czasu rzeczywistego jest niedostępne, system może użyć ścieżki REST.
Rezultat	Wiadomość zostaje zapisana i dostarczona drugiej stronie.

Tabela 18. Wybrane przypadki użycia systemu Coachify - obsługa prosby wiadomości. Źródło: opracowanie własne.

Pole	Opis
Aktor główny	Trener
Warunki początkowe	Klient wysłał prośbę o wiadomość z <i>marketplace'u</i> .
Przebieg główny	Trener otwiera zakładkę próśb i akceptuje prośbę.
Przebieg alternatywny	Trener może zignorować albo zablokować nadawcę.
Rezultat	Konwersacja przechodzi w stan normalny albo kontakt zostaje ograniczony.

Tabela 19. Wybrane przypadki użycia systemu Coachify - zarządzanie ćwiczeniami globalnymi. Źródło: opracowanie własne.

Pole	Opis
Aktor główny	Administrator
Warunki początkowe	Administrator jest zalogowany.
Przebieg główny	Administrator otwiera panel ćwiczeń, dodaje lub edytuje ćwiczenie globalne i zapisuje zmiany.
Przebieg alternatywny	Jeżeli plik wideo nie spełnia ograniczeń, system odrzuca przesłanie pliku.
Rezultat	Ćwiczenie globalne jest dostępne dla trenerów.

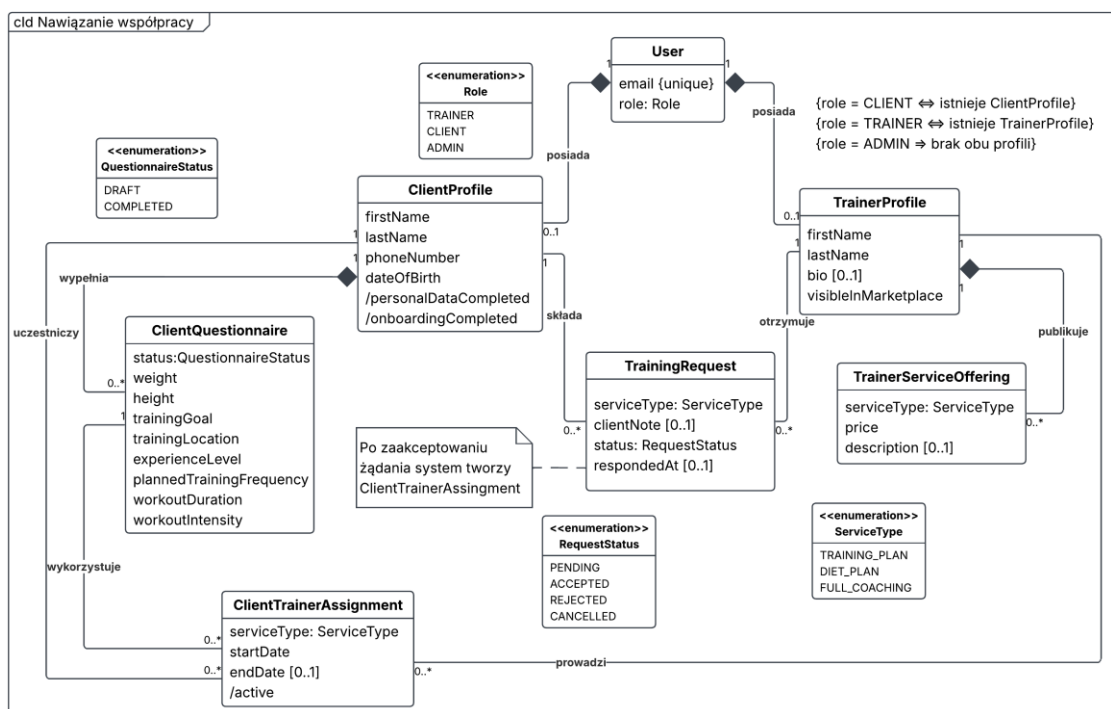
Przypadki użycia pokazują, że rdzeń systemu nie ogranicza się do pojedynczej roli. Najważniejsze procesy wymagają współdziałania klienta i trenera, a panel administratora zapewnia jedynie wsparcie zasobów wspólnych.

3.7. Model danych

W miejsce relacyjnego modelu danych przedstawiono projektowy model klas dziedziny. Model ten opisuje najważniejsze obiekty występujące w systemie oraz zależności między nimi z perspektywy analizy i projektu. Nie jest to opis fizycznego schematu bazy danych ani szczegółów implementacji utrwalania danych. Szczegóły związane z technologiami, strukturą bazy oraz mechanizmami dostępu do danych zostały omówione w rozdziałach implementacyjnych.

Na rysunku 6 przedstawiono kompletny model klas znajdujących się w aplikacji Coachify, natomiast w dalszej części modele zostały podzielone na tematyczne części oraz kolejno omówione.

Centralnym pojęciem modelu jest użytkownik systemu, z którym powiązana jest rola oraz profil odpowiadający sposobowi korzystania z aplikacji. Klient posiada profil klienta oraz może uzupełnić kwestionariusz podopiecznego. Trener posiada profil trenera, ofertę usług oraz może być widoczny w *marketplace*. Zgłoszenie o współpracę łączy klienta, trenera oraz wybrany typ usługi. Po zaakceptowaniu zgłoszenia powstaje relacja trener-klient, która stanowi podstawę dalszego korzystania z planów, raportów i komunikacji. Model klas obejmujący użytkowników, profile oraz współpracę trenera z klientem przedstawiono na rysunku 7. Najważniejsze elementy modelu danych przedstawiono w tabeli 20.



Rysunek 7. Model klas użytkowników, profili i współpracy. Źródło: opracowanie własne.

Tabela 20. Główne elementy modelu danych systemu Coachify. Źródło: opracowanie własne.

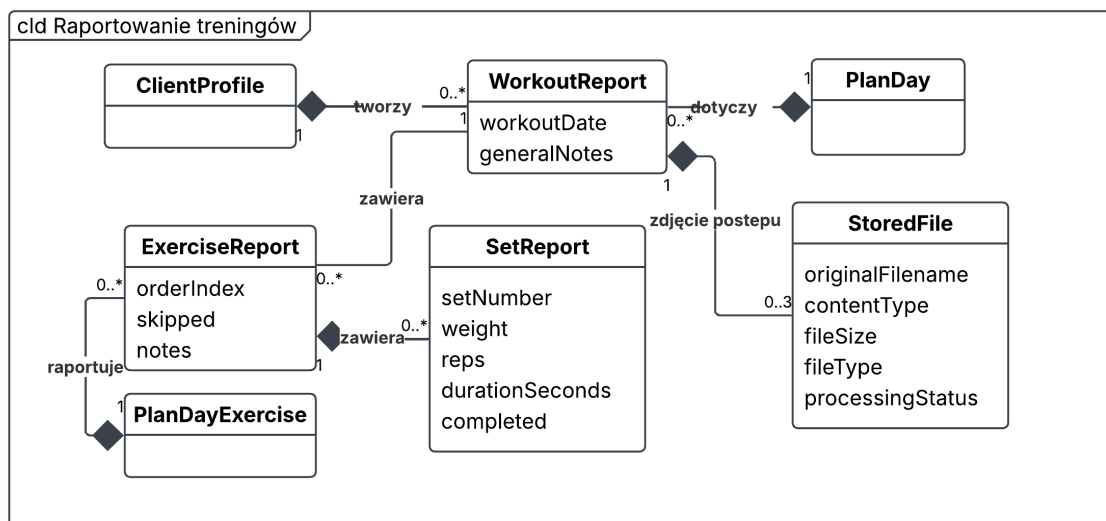
Obszar modelu	Przykładowe encje	Rola w systemie
Tożsamość i profile	User, ClientProfile, TrainerProfile	Reprezentują konto użytkownika, rolę oraz dane specyficzne dla klienta albo trenera.
Marketplace i współpraca	TrainerServiceOffering, TrainingRequest, ClientTrainerAssignment	Opisują ofertę trenera, zgłoszenia klientów oraz aktywne relacje współpracy.
Onboarding klienta	ClientQuestionnaire	Przechowuje odpowiedzi klienta potrzebne do oceny zgłoszenia i przygotowania planu.

Plany treningowe	TrainingPlan, PlanDay, PlanDayExercise, Exercise	Reprezentują strukturę planu treningowego, dni planu oraz ćwiczenia.
Szablony treningowe	TrainingPlanTemplate, TemplateDay, TemplateDayExercise	Pozwalają trenerowi przygotowywać powtarzalne struktury planów treningowych.
Plany dietetyczne	DietPlan, DietDay, DietMeal, DietMealProduct	Reprezentują strukturę planu dietetycznego, dni, posiłki i produkty.
Szablony diet i posiłków	DietPlanTemplate, DietTemplateDay, DietTemplateMeal, MealTemplate	Pozwalają wielokrotnie wykorzystywać przygotowane wzorce diet i posiłków.
Raportowanie treningu	WorkoutReport, ExerciseReport, SetReport	Przechowują informacje o wykonanym treningu, ćwiczeniach i seriach.
Komunikacja	Conversation, ChatMessage, TrainerClientBlock	Obsługują rozmowy, wiadomości, prośby o kontakt i blokady użytkowników.
Multimedia	StoredFile	Reprezentuje pliki powiązane z profilami, raportami i ćwiczeniami.

Relacje między klasami odzwierciedlają główne procesy biznesowe systemu. Profil trenera może posiadać wiele ofert usług, a profil klienta może wysłać zgłoszenie do wybranego trenera. Zaakceptowane zgłoszenie prowadzi do utworzenia przypisania klienta do trenera. Na podstawie tej relacji trener może przygotowywać plany treningowe i dietetyczne, a klient może przeglądać przypisane plany oraz raportować wykonanie treningów.

Model planów treningowych i dietetycznych został oparty na strukturze hierarchicznej. Plan treningowy składa się z dni, a dni zawierają ćwiczenia wraz z parametrami wykonania. Plan dietetyczny składa się z dni, dni zawierają posiłki, a posiłki składają się z produktów lub składników. W obu przypadkach zastosowano szablony, które pozwalają trenerowi przygotowywać powtarzalne wzorce i wykorzystywać je przy pracy z wieloma klientami. Strukturę klas planów i szablonów treningowych przedstawiono na rysunku 8, a planów i szablonów dietetycznych na rysunku 9.

Raport treningowy jest powiązany z klientem, planem oraz konkretnym dniem planu. Dzięki temu dane raportowane przez klienta zachowują kontekst pierwotnych założeń treningowych. Raport może zawierać informacje o wykonanych ćwiczeniach, seriach, notatkach oraz zdjęciach postępów. Model klas raportów treningowych oraz plików multimedialnych przedstawiono na rysunku 10.



Rysunek 10. Model klas raportów treningowych. Źródło: opracowanie własne.

Komunikacja w systemie jest reprezentowana przez konwersację i wiadomości. Konwersacja łączy jednego trenera i jednego klienta. Dodatkowym elementem modelu jest prośba o kontakt, która pozwala rozpocząć komunikację jeszcze przed nawiązaniem pełnej współpracy. Trener może taką prośbę zaakceptować, odrzucić albo zablokować użytkownika.

Klasa pliku multimedialnego pełni rolę wspólnej reprezentacji materiałów wykorzystywanych w różnych modułach systemu. Plik może być powiązany z awatarem użytkownika, zdjęciem postępów albo materiałem ćwiczenia. Dzięki temu obsługa multimediiów pozostaje spójna mimo różnych zastosowań biznesowych.

3.8. Słownik pojęć

W rozdziale zastosowano pojęcia domenowe i techniczne, które powracają w kolejnych częściach pracy. Tabela 21 zawiera ich definicje przyjęte na potrzeby opisu systemu Coachify.

Tabela 21. Słownik pojęć stosowanych w opisie systemu Coachify. Źródło: opracowanie własne.

Pojęcie	Definicja
Administrator	Użytkownik z dostępem do panelu administracyjnego i zarządzania ćwiczeniami globalnymi.
Coachify	Platforma webowa wspierająca współpracę trenerów personalnych z klientami.
Czat	Moduł komunikacji tekstowej jeden-do-jednego między trenerem i klientem.
Ćwiczenie globalne	Ćwiczenie zarządzane przez administratora i dostępne dla wszystkich trenerów.

Ćwiczenie prywatne	Ćwiczenie utworzone przez trenera i dostępne w jego bibliotece.
Klient	Użytkownik systemu korzystający z usług trenera i realizujący przypisane plany.
Kwestionariusz podopiecznego	Wieloetapowy formularz zbierający dane potrzebne trenerowi do przygotowania planu.
<i>Marketplace</i>	Publiczny katalog trenerów wraz z ofertami usług.
Miękkie usuwanie	Oznaczenie rekordu jako usuniętego bez fizycznego kasowania go z bazy danych.
<i>Onboarding</i>	Proces uzupełnienia danych klienta i przejścia przez kwestionariusz przed rozpoczęciem współpracy.
Optymistyczne blokowanie	Mechanizm wykrywania konfliktów współbieżnej edycji danych przez wersjonowanie rekordu.
Plan dietetyczny	Struktura dni, posiłków i produktów przypisana przez trenera konkretnemu klientowi.
Plan treningowy	Struktura dni i ćwiczeń przypisana przez trenera konkretnemu klientowi.
Prośba o wiadomość	Konwersacja oczekująca na decyzję trenera, inicjowana przez klienta z poziomu <i>marketplace'u</i> .
Przypisanie klient-trener	Aktywna relacja utworzona po zaakceptowaniu zgłoszenia przez trenera.
Raport treningowy	Zapis wykonania treningu utworzony przez klienta na podstawie dnia planu.
STOMP	Protokół komunikacyjny wykorzystywany nad WebSocket do obsługi subskrypcji i kolejek wiadomości.
Szablon	Reużywalna struktura planu albo posiłku przygotowana przez trenera.
Trener	Użytkownik systemu przygotowujący plany, obsługujący zgłoszenia i prowadzący klientów.

4. Technologie i narzędzia

Przy wyborze technologii oraz narzędzi kierowano się przede wszystkim dojrzałością i stabilnością rozwiązań, dostępnością dokumentacji oraz wsparciem społeczności. Główny stos aplikacyjny oparto przede wszystkim na rozwiązaniach o otwartym kodzie źródłowym (ang. *open source*), co eliminuje koszty licencyjne i ułatwia niezależny rozwój systemu. W rozdziale opisano technologie wykorzystane w części aplikacyjnej. Całość infrastruktury uruchomiono na maszynie wirtualnej z systemem Linux, co warunkowało wybór narzędzi kompatybilnych z tym środowiskiem.

4.1. Java 21 i Spring Boot

Do implementacji warstwy serwerowej wybrano język Java 21 wraz z *frameworkiem* Spring Boot. Decyzja ta była podyktowana zarówno stabilnością i dojrzałością ekosystemu Java, jak również doświadczeniem autorów zdobytym podczas realizacji zajęć na uczelni. Spring Boot umożliwia szybkie tworzenie nowoczesnych aplikacji webowych dzięki bogatemu zestawowi gotowych rozwiązań oraz integracji z licznymi bibliotekami. Framework zapewnia wsparcie dla mechanizmów bezpieczeństwa, dostępu do danych oraz komunikacji w czasie rzeczywistym, realizowanych odpowiednio przy użyciu modułów Spring Security, Spring Data JPA oraz Spring WebSocket. Pozwala to ograniczyć ilość kodu konfiguracyjnego i skoncentrować się na implementacji logiki biznesowej systemu.

Wybór języka Java można uzasadnić przede wszystkim jego ugruntowaną pozycją w tworzeniu aplikacji webowych i serwerowych. Herbert Schildt podkreśla znaczenie tego języka, pisząc: „Java is one of the world’s most important and widely used computer languages. [15]” Stwierdzenie to wskazuje, że Java nie jest technologią niszową, lecz jednym z najczęściej wykorzystywanych języków programowania na świecie. W praktyce przekłada się to na szeroką dostępność dokumentacji, materiałów edukacyjnych, bibliotek oraz rozwiązań sprawdzonych przez społeczność programistyczną. Z punktu widzenia niniejszego projektu oznacza to większą niezawodność procesu wytwarzania oprogramowania oraz łatwiejsze rozwiązywanie problemów pojawiających się podczas implementacji.

Istotną cechą Javy jest również jej przenośność, wynikająca z działania na maszynie wirtualnej JVM [16]. Oznacza to, że raz skompilowana aplikacja może zostać uruchomiona na różnych systemach operacyjnych bez konieczności modyfikacji kodu źródłowego. Cecha ta ma szczególne znaczenie we współczesnych środowiskach wdrożeniowych, gdzie aplikacje są uruchamiane w kontenerach Docker, środowiskach chmurowych lub na serwerach pracujących pod kontrolą różnych systemów operacyjnych. Przenośność upraszcza proces wdrażania oraz zwiększa elastyczność infrastruktury systemu.

Kolejnym argumentem przemawiającym za wyborem Javy jest jej wieloletnie zastosowanie w systemach sieciowych i rozproszonych. Pat Niemeyer i Jonathan Knudsen zauważają, że Java zyskała istotne znaczenie jako platforma dla aplikacji serwerowych [17]. Serwer projektowanej aplikacji odpowiada za realizację logiki biznesowej, komunikację z bazą danych, autoryzację użytkowników, zarządzanie sesjami, obsługę tokenów dostępowych JWT i tokenów odświeżających oraz obsługę żądań HTTP i komunikacji w czasie rzeczywistym. Są to obszary, w których Java od wielu lat znajduje szerokie zastosowanie zarówno w projektach komercyjnych, jak i systemach korporacyjnych. Wysoka wydajność środowiska uruchomieniowego JVM oraz możliwość obsługi dużej liczby równoczesnych połączeń sprawiają, że język ten dobrze nadaje się do tworzenia skalowalnych aplikacji serwerowych.

Dodatkowym argumentem jest obiektowy charakter języka [18]. Programowanie obiektowe umożliwia odwzorowanie elementów rzeczywistego systemu w postaci klas i obiektów, co ułatwia organizację kodu oraz jego dalszy rozwój. Dzięki zastosowaniu zasad takich jak enkapsulacja, dziedziczenie czy polimorfizm możliwe jest tworzenie modułowych i łatwych w utrzymaniu aplikacji.

Podjęcie to dobrze współgra z architekturą *frameworka* Spring Boot, w której występuje wyraźny podział na encje, repozytoria, serwisy oraz kontrolery. Taka struktura zwiększa czytelność kodu, ułatwia testowanie poszczególnych komponentów oraz sprzyja rozbudowie systemu w przyszłości. Zastosowany podział na warstwy oraz wzorzec organizacji kodu w aplikacjach opartych na *frameworku* Spring opisano szerzej w [19].

W projekcie wykorzystano Java 21 LTS, czyli wersję długoterminowo wspieraną. Wersja ta wprowadza usprawnienia związane z wydajnością, bezpieczeństwem oraz produktywnością programistów, zachowując jednocześnie zgodność z ekosystemem bibliotek i *frameworków* wykorzystywanych w projekcie. Wybór wersji LTS zwiększa przewidywalność utrzymania aplikacji, co jest istotne w przypadku systemu przeznaczonego do dalszego rozwoju.

4.2. PostgreSQL i Hibernate

Jako system zarządzania bazą danych w projekcie wybrano PostgreSQL. Decyzja ta została podjęta ze względu na wysoką niezawodność, zgodność ze standardem SQL, szerokie możliwości konfiguracji oraz bardzo dobrą integrację z technologiami wykorzystywanymi w ekosystemie Java. PostgreSQL należy do najpopularniejszych systemów relacyjnych baz danych typu *open source* i od wielu lat znajduje zastosowanie zarówno w projektach komercyjnych, jak i systemach korporacyjnych. Dodatkowym argumentem przemawiającym za jego wyborem była pełna kompatybilność z biblioteką Hibernate oraz *frameworkiem* Spring Data JPA, które zostały wykorzystane do realizacji warstwy dostępu do danych.

W ramach architektury wdrożeniowej baza danych została uruchomiona poza klastrem Kubernetes, na dedykowanej instancji serwerowej. Takie rozwiązanie pozwala oddzielić warstwę przechowywania danych od warstwy aplikacyjnej, co zwiększa bezpieczeństwo i niezawodność całego systemu. Ewentualny restart, aktualizacja lub rekonfiguracja klastra nie wpływa bezpośrednio na dane zgromadzone w bazie. Ponadto dla każdego środowiska (deweloperskiego, testowego i produkcyjnego) utworzono odrębne bazy danych, co zapobiega przypadkowemu mieszanii danych oraz umożliwia bezpieczne przeprowadzanie testów nowych funkcjonalności.

PostgreSQL jest systemem bazodanowym rozwijanym od ponad trzydziestu lat, wywodzącym się z projektu POSTGRES stworzonego na Uniwersytecie Kalifornijskim w Berkeley. Oficjalna dokumentacja opisuje go następująco: „PostgreSQL is an object-relational database management system. [20]” Określenie PostgreSQL jako systemu obiektowo-relacyjnego wskazuje, że łączy on zalety klasycznych relacyjnych baz danych z rozwiązaniami umożliwiającymi odwzorowywanie bardziej złożonych struktur danych. Oprócz standardowych typów danych PostgreSQL oferuje między innymi obsługę typów JSON i JSONB, tablic, typów geometrycznych oraz możliwość definiowania własnych typów danych. Funkcjonalności te pozwalają tworzyć bardziej elastyczne modele danych i dostosowywać strukturę bazy do potrzeb rozwijającej się aplikacji. W przypadku projektowanego systemu umożliwia to łatwiejsze rozszerzanie funkcjonalności bez konieczności przebudowy całej warstwy danych.

Istotną zaletą PostgreSQL jest również zgodność z powszechnie stosowanym standardem SQL. W dokumentacji podkreślono, że „It supports a large part of the SQL standard. [20]” Zgodność ze standardem SQL ma duże znaczenie z punktu widzenia utrzymywalności i dalszego rozwoju projektu. Dzięki temu struktura bazy danych oraz stosowane zapytania są bardziej czytelne, przewidywalne i łatwiejsze do przeniesienia pomiędzy różnymi środowiskami. Ogranicza to również uzależnienie projektu od rozwiązań charakterystycznych wyłącznie dla jednego producenta systemu bazodanowego. Jednocześnie PostgreSQL oferuje liczne rozszerzenia wykraczające poza standard SQL, co pozwala zachować dużą elastyczność podczas projektowania i optymalizacji warstwy danych.

PostgreSQL wspiera również szereg zaawansowanych mechanizmów charakterystycznych dla nowoczesnych systemów bazodanowych, takich jak klucze główne i obce, ograniczenia integralności,

wyzwalacze, procedury składowane, widoki, indeksy czy transakcje. Szczególnie istotny jest mechanizm wielowersyjnej kontroli współbieżności (MVCC - Multi-Version Concurrency Control), który umożliwia jednoczesną pracę wielu użytkowników bez blokowania operacji odczytu danych [20]. Ma to duże znaczenie w aplikacjach webowych obsługujących wielu użytkowników równocześnie, ponieważ pozwala utrzymać wysoką wydajność systemu nawet przy zwiększonym obciążeniu.

W literaturze PostgreSQL jest często przedstawiany jako rozwiązanie stabilne i niezawodne. Autorzy książki *Beginning Databases with PostgreSQL* wskazują, że PostgreSQL jest bardzo niezawodny [21]. Niezawodność systemu zarządzania bazą danych jest jednym z najważniejszych kryteriów wyboru technologii dla warstwy serwerowej. Dane przechowywane przez aplikację stanowią kluczowy zasób systemu i często mają większą wartość niż sama warstwa programistyczna. Kod aplikacji można odtworzyć z repozytorium źródłowego, natomiast utrata danych użytkowników może prowadzić do nieodwracalnych konsekwencji. Z tego względu wybór sprawdzonego i stabilnego systemu bazodanowego był istotnym elementem procesu projektowego.

Kolejną cechą PostgreSQL jest architektura klient-serwer, umożliwiająca oddzielenie aplikacji od fizycznego sposobu przechowywania danych [21]. Model klient-serwer bardzo dobrze współgra z nowoczesnymi architekturami opartymi na kontenerach oraz rozwiązaniach chmurowych. W projektowanym systemie serwer działa jako zestaw usług uruchamianych w środowisku Kubernetes, natomiast PostgreSQL funkcjonuje jako niezależna usługa odpowiedzialna za trwałe przechowywanie danych. Rozdzielenie tych komponentów zwiększa odporność systemu na awarie, upraszcza proces tworzenia kopii zapasowych oraz umożliwia niezależne skalowanie warstwy aplikacyjnej i warstwy danych.

W celu uproszczenia komunikacji pomiędzy aplikacją a bazą danych wykorzystano bibliotekę Hibernate, będącą najpopularniejszą implementacją technologii ORM (Object Relational Mapping) w ekosystemie Java. Jej zadaniem jest mapowanie klas języka Java na tabele relacyjnej bazy danych oraz automatyczne tłumaczenie operacji wykonywanych na obiektach na odpowiednie zapytania SQL. Dzięki temu programista może operować na encjach i obiektach biznesowych zamiast bezpośrednio zarządzać komunikacją z bazą danych.

Zastosowanie Hibernate przynosi szereg korzyści. Przede wszystkim znacząco ogranicza ilość kodu technicznego odpowiedzialnego za obsługę operacji CRUD (Create, Read, Update, Delete) [22]. Biblioteka wspiera automatyczne generowanie zapytań, zarządzanie relacjami między encjami, obsługę transakcji oraz mechanizmy buforowania danych. Pozwala to skupić się na implementacji logiki biznesowej aplikacji zamiast na ręcznym tworzeniu i utrzymywaniu dużej liczby zapytań SQL.

W projekcie Hibernate został wykorzystany za pośrednictwem Spring Data JPA. Takie rozwiązanie umożliwia definiowanie repozytoriów w postaci interfejsów oraz automatyczne generowanie podstawowych operacji dostępu do danych. Podejście to jest zgodne z architekturą warstwową zastosowaną w aplikacji i pozwala zachować wyraźny podział odpowiedzialności pomiędzy poszczególnymi komponentami systemu. Migracje schematu bazy danych należy przypisać mechanizmowi Flyway, natomiast Hibernate odpowiada za mapowanie encji oraz kontrolę zgodności modelu obiektowego ze schematem w podstawowej konfiguracji aplikacji.

4.3. JSON Web Token (JWT)

W aplikacji zastosowano mechanizm JSON Web Token (JWT) jako podstawowy sposób realizacji procesu uwierzytelniania i autoryzacji użytkowników. JWT jest standardem umożliwiającym bezpieczne przekazywanie informacji pomiędzy stronami komunikacji w postaci samodzielnego tokenu zawierającego zestaw danych opisujących użytkownika. Token dostępowy JWT jest wystawiany po poprawnym logowaniu i przechowywany w ciasteczku `HttpOnly`. Serwer obsługuje również nagłówek `HTTP Authorization` jako mechanizm pomocniczy, przydatny między innymi

w testowaniu API. Główny przepływ aplikacji przeglądarkowej pozostaje jednak oparty na ciasteczkach przesyłanych automatycznie przez przeglądarkę.

W dokumencie opracowanym przez OAuth Working Group JWT opisano jako: JSON Web Token (JWT) is a compact claims representation format intended for space constrained environments such as HTTP Authorization headers and URI query parameters. [23]” Powyższa definicja wskazuje na jedną z najważniejszych cech JWT, jaką jest kompaktowość. Token zajmuje niewiele miejsca i może być wygodnie przesyłany wraz z każdym żądaniem HTTP bez znaczącego zwiększania ruchu sieciowego. Cecha ta ma szczególne znaczenie w aplikacjach internetowych opartych na architekturze REST, gdzie każde żądanie powinno zawierać wszystkie informacje niezbędne do jego obsługi. Dzięki temu JWT bardzo dobrze współpracuje z *frameworkiem* Spring Security, który umożliwia automatyczną weryfikację tokenów oraz kontrolę dostępu do chronionych zasobów aplikacji.

Głównym zadaniem JWT jest przechowywanie i przekazywanie tzw. *claims*, czyli zestawu informacji opisujących użytkownika lub inny podmiot uczestniczący w komunikacji. Mogą to być między innymi identyfikator użytkownika, adres e-mail, role systemowe, czas wygaśnięcia tokenu czy dodatkowe dane wykorzystywane podczas procesu autoryzacji. Sebastián Peyrott definiuje JWT następująco: „JWT [...] is a standard for safely passing claims in space constrained environments. [24]” W praktyce oznacza to, że po poprawnym zalogowaniu użytkownik otrzymuje token zawierający podstawowe informacje niezbędne do jego identyfikacji. Podczas kolejnych żądań serwer odczytuje dane zapisane w tokenie i na ich podstawie określa uprawnienia użytkownika. Takie podejście pozwala przyspieszyć proces autoryzacji a jednocześnie umożliwia zachowanie bezstanowego charakteru aplikacji, co stanowi jedną z głównych zalet architektury REST.

Istotnym elementem JWT jest jego wewnętrzna struktura. Token składa się z trzech części: nagłówka (*header*), ładunku danych (*payload*) oraz podpisu (*signature*). Nagłówek zawiera informacje o typie tokenu i zastosowanym algorytmie kryptograficznym, *payload* przechowuje *claims*, natomiast podpis służy do weryfikacji integralności danych. JWT nie jest przypadkowym ciągiem znaków, lecz uporządkowaną strukturą danych zakodowaną w formacie Base64URL. Należy podkreślić, że samo zakodowanie danych nie jest równoznaczne z ich zaszyfrowaniem. Każda osoba posiadająca token może odczytać informacje zapisane w jego części *payload*. Z tego względu w tokenach JWT nie powinny być przechowywane dane poufne, takie jak hasła, numery kart płatniczych czy inne informacje wymagające zachowania tajemnicy. Bezpieczeństwo JWT opiera się przede wszystkim na ochronie integralności danych, a nie na ich ukrywaniu.

Z punktu widzenia bezpieczeństwa szczególnie istotne jest rozróżnienie pomiędzy podpisywaniem a szyfrowaniem tokenów. W specyfikacji JWT wskazano, że: „The claims in a JWT are encoded as a JSON object that is used as the payload of a JSONmWeb Signature (JWS) structure or as the plaintext of a JSON Web Encryption (JWE) structure, enabling the claims to be digitally signed or integrity protected with a Message Authentication Code (MAC) and/or encrypted. [23]” W większości współczesnych aplikacji webowych wykorzystywane są tokeny podpisane przy użyciu standardu JWS. Podpis umożliwia serwerowi sprawdzenie, czy token został wystawiony przez zaufane źródło oraz czy jego zawartość nie została zmodyfikowana przez osobę nieuprawnioną. W projektowanej aplikacji zastosowano właśnie takie rozwiązanie. Serwer podczas każdego żądania weryfikuje podpis tokenu oraz sprawdza jego ważność czasową przed udzieleniem dostępu do chronionych zasobów systemu.

Jedną z najważniejszych zalet JWT jest możliwość realizacji bezstanowego modelu uwierzytelniania. W tradycyjnym podejściu opartym na sesjach serwer musi przechowywać informacje o każdej aktywnej sesji użytkownika. W przypadku JWT wszystkie niezbędne informacje znajdują się bezpośrednio w tokenie. Dzięki temu serwer nie musi utrzymywać dodatkowego stanu pomiędzy kolejnymi żądaniami, co upraszcza architekturę systemu oraz ułatwia jego skalowanie. Cecha ta jest szczególnie istotna w środowiskach opartych na kontenerach i orkiestracji Kubernetes, gdzie wiele instancji aplikacji może równocześnie obsługiwać żądania użytkowników.

Popularność JWT wynika również z jego szerokiego zastosowania w nowoczesnych standardach bezpieczeństwa. Jak wskazuje Hafizur Rahman Anik: „JWTs [...] can be used in authentication, authorization and for information exchange. [25]” JWT znajduje zastosowanie między innymi w protokołach OAuth 2.0 oraz OpenID Connect, które są obecnie powszechnie wykorzystywane do realizacji procesów logowania i zarządzania tożsamością użytkowników. W projektowanym systemie token pełni przede wszystkim funkcję nośnika informacji identyfikujących użytkownika oraz jego uprawnienia. Po zalogowaniu użytkownik otrzymuje token, który następnie dołącza do wszystkich kolejnych żądań kierowanych do serwera.

Warto również zauważyć, że wykorzystanie JWT dobrze współgra z architekturą mikroservisową oraz rozwiązaniami chmurowymi. Ponieważ token zawiera komplet informacji potrzebnych do identyfikacji użytkownika, może być weryfikowany przez różne komponenty systemu bez konieczności współdzielenia pamięci sesji. Ułatwia to skalowanie aplikacji, równoważenie obciążenia pomiędzy instancjami oraz wdrażanie kolejnych usług wykorzystujących wspólny mechanizm uwierzytelniania.

4.4. OpenAPI / Swagger UI

Do udokumentowania oraz testowania interfejsu programistycznego aplikacji zastosowano specyfikację OpenAPI wraz z narzędziem Swagger UI. OpenAPI jest powszechnie przyjętym, niezależnym od języka programowania standardem opisu interfejsów typu REST w postaci czytelnej zarówno dla człowieka, jak i dla maszyny [26]. Na podstawie takiego opisu Swagger UI generuje interaktywną dokumentację, umożliwiającą przeglądanie dostępnych zasobów oraz wywoływanie poszczególnych punktów końcowych (ang. *endpoint*) bezpośrednio z poziomu przeglądarki.

Wybór tego rozwiązania był podyktowany wcześniejszym doświadczeniem zespołu, zdobytym podczas zajęć z technologii .NET, oraz pozycją OpenAPI jako faktycznego standardu dokumentowania interfejsów REST [26]. Istotną zaletą jest spójność dokumentacji z rzeczywistą implementacją - opis powstaje na podstawie kodu warstwy serwerowej, co ogranicza ryzyko rozbieżności pomiędzy dokumentacją a zachowaniem systemu i ułatwia współpracę osób rozwijających warstwę serwerową i kliencką.

W projektowanym systemie konfiguracja OpenAPI opisuje uwierzytelnianie typu *Bearer*, obsługiwane przez serwer jako mechanizm pomocniczy, przydatny przede wszystkim w testowaniu interfejsu. Główny przepływ aplikacji klienckiej pozostaje natomiast oparty na ciasteczkach *HttpOnly*. Wykorzystanie Swagger UI w procesie weryfikacji interfejsu opisano w rozdziale 7.

4.5. MinIO

Do przechowywania plików, takich jak awatary, zdjęcia postępów oraz materiały wideo, zastosowano MinIO - system obiektowego przechowywania danych (ang. *object storage*) w pełni zgodny z interfejsem API Amazon S3 [27]. W modelu obiektowym dane przechowywane są jako obiekty grupowane w zasobnikach (ang. *bucket*) i identyfikowane przez klucze, w odróżnieniu od klasycznych systemów plikowych czy blokowych. Podejście to dobrze odpowiada charakterystyce danych nieustrukturyzowanych, których ilość i rozmiar mogą dynamicznie rosnąć wraz z rozwojem systemu.

Wybór MinIO był podyktowany doświadczeniem autorów z interfejsem S3 oraz większą stabilnością tego rozwiązania w porównaniu z alternatywnym podejściem opartym na wolumenach trwałych Kubernetes (ang. *Persistent Volume Claim, PVC*). Zgodność z interfejsem S3 [27] ma dodatkową zaletę praktyczną - pozwala w przyszłości zastąpić MinIO usługą chmurową zgodną

z tym samym standardem bez konieczności modyfikacji kodu aplikacji, co zwiększa elastyczność architektury.

W projekcie uruchomiono pojedynczą instancję MinIO współdzieloną pomiędzy środowiskami, z rozdzielaniem danych poszczególnych środowisk za pomocą prefiksów zasobników. Szczegóły integracji warstwy serwerowej z MinIO opisano w podrozdziale 5.7.

4.6. React, TypeScript, Vite

Warstwę kliencką systemu zrealizowano jako aplikację typu *Single-Page Application* z wykorzystaniem biblioteki React 19.2.0, języka TypeScript 5.9.3 oraz narzędzia budującego Vite 7.1.11. React odpowiada za komponentowy model interfejsu użytkownika, TypeScript zapewnia statyczne typowanie kodu, natomiast Vite pełni funkcję środowiska deweloperskiego oraz narzędzia do budowy wersji produkcyjnej aplikacji.

React jest biblioteką JavaScript przeznaczoną do tworzenia interaktywnych interfejsów użytkownika w oparciu o architekturę komponentową. Został opracowany przez Facebook w 2011 roku, a w 2013 roku udostępniono go jako projekt *open source* [28]. Według A. Banksa i E. Porcello React powstał jako odpowiedź na problemy związane z tworzeniem dużych, opartych na danych aplikacji internetowych, w których zarządzanie interfejsem użytkownika staje się szczególnie złożone [29].

Wybór Reacta w projekcie wynikał z możliwości budowy interfejsu jako zbioru niezależnych i wielokrotnie wykorzystywanych komponentów. Podejście to ułatwia organizację kodu oraz rozwój systemu obsługującego różne role użytkowników: administratora, trenera i podopiecznego. React wykorzystuje programowanie deklaratywne, w którym programista opisuje oczekiwany stan interfejsu, a biblioteka odpowiada za jego aktualizację. Chris Minnick wskazuje, że nazwa React odnosi się do reagowania interfejsu na zmiany danych oraz aktualizowania tylko tych elementów, których stan faktycznie uległ zmianie [28]. Dodatkową zaletą Reacta jest fakt, że koncentruje się on na warstwie prezentacji, pozostawiając możliwość doboru narzędzi odpowiedzialnych za komunikację z warstwą serwerową, routing czy zarządzanie danymi [29].

W projekcie zastosowano mechanizmy typowe dla współczesnego Reacta, takie jak hooki stanu i efektów, konteksty dla danych współdzielonych oraz komponenty przekazujące referencje. Dane pochodzące z warstwy serwerowej obsługiwane są przez bibliotekę TanStack Query, która odpowiada za pobieranie danych, ich buforowanie, ponawianie zapytań oraz unieważnianie wyników po wykonaniu mutacji. Dzięki temu stan serwerowy został oddzielony od lokalnego stanu interfejsu, a w projekcie nie było potrzeby wprowadzania globalnego magazynu stanu, takiego jak Redux.

TypeScript jest rozszerzeniem języka JavaScript opracowanym przez firmę Microsoft. Wprowadza opcjonalny system typów i pozostaje nadzbiorem JavaScriptu, co oznacza, że poprawny kod JavaScript jest również poprawnym kodem TypeScript [30]. Wykorzystanie TypeScripta w projekcie zwiększa niezawodność warstwy klienckiej oraz ułatwia integrację z warstwą serwerową. Ma to znaczenie w systemie wielomodułowym, w którym warstwa kliencka obsługuje różne struktury DTO zależne od roli użytkownika i realizowanego procesu.

Boris Cherny definiuje bezpieczeństwo typów jako wykorzystanie typów do zapobiegania wykonywaniu przez program operacji niezgodnych z jego założeniami [31]. W praktyce pozwala to ograniczyć błędy związane między innymi z odwołaniami do nieistniejących właściwości, przekazywaniem nieprawidłowych argumentów oraz niezgodnością struktur danych otrzymywanych z API. TypeScript wspiera również refaktoryzację, ponieważ potencjalne błędy mogą zostać wykryte przez kompilator przed uruchomieniem programu [30].

Vite zastosowano jako narzędzie uruchomieniowe i budujące dla aplikacji klienckiej. Wykorzystuje ono natywne moduły ECMAScript, co skraca czas startu środowiska deweloperskiego i przyspiesza pracę podczas wprowadzania zmian w kodzie. W projekcie skonfigurowano Vite

z wtyczką `@vitejs/plugin-react-swc`, która wykorzystuje SWC do szybkiej transformacji kodu React i TypeScript. Serwer deweloperski przekierowuje żądania `/api` oraz `/ws` do lokalnej warstwy serwerowej, dzięki czemu warstwa kliencka może korzystać ze względnych ścieżek API. Ułatwia to wdrożenie aplikacji za tym samym *originem* i *reverse proxy*, bez konieczności modyfikowania kodu źródłowego. W konfiguracji produkcyjnej usuwane są również wywołania `console` oraz instrukcje `debugger`, co ogranicza ilość kodu diagnostycznego trafiającego do paczki końcowej.

4.7. Tailwind CSS, Radix UI

Warstwę wizualną interfejsu zbudowano z wykorzystaniem Tailwind CSS 4.1.15 oraz komponentów opartych na Radix UI. Tailwind CSS zastosowano jako narzędzie do stylowania komponentów, natomiast Radix UI wykorzystano jako zestaw prymitywów odpowiedzialnych za logikę, semantykę i dostępność bardziej złożonych elementów interfejsu.

Tailwind CSS reprezentuje podejście *utility-first*, w którym wygląd elementów określa się przez łączenie małych klas narzędziowych bezpośrednio w kodzie widoku. Oficjalna dokumentacja opisuje ten model jako stylowanie za pomocą jednofunkcyjnych klas prezentacyjnych komponowanych w znaczniku komponentu [32]. Noel Rappin wskazuje, że w odróżnieniu od *frameworków* takich jak Bootstrap, klasy Tailwinda zwykle odpowiadają pojedynczym właściwościom CSS, na przykład marginesowi, rozmiarowi tekstu, kolorowi tła lub odstępowi wewnętrznemu [33]. Takie podejście ułatwia szybkie prototypowanie, ponieważ zakres zmiany jest widoczny bezpośrednio w miejscu jej wprowadzenia.

W projekcie zastosowano Tailwind CSS 4 z wtyczką dla Vite oraz tokenami motywu definiowanymi w pliku CSS. Dokumentacja Tailwinda wskazuje, że integracja z Vite odbywa się przez dodanie wtyczki `@tailwindcss/vite` oraz import Tailwinda w głównym arkuszu CSS [34]. Zmienne motywu pozwalają natomiast definiować wartości projektowe, takie jak kolory, typografia, cienie czy *breakpointy* [35]. Paleta kolorów aplikacji została oparta na przestrzeni OKLCH i obejmuje wariant jasny oraz ciemny, co jest zgodne ze zmianami w Tailwind CSS 4, gdzie domyślną paletę kolorów przeniesiono z RGB do OKLCH [36].

Radix UI wykorzystano do budowy komponentów takich jak dialogi, zakładki, selekty, akordeony, menu rozwijane, *tooltipy* i przełączniki. Radix Primitives są niskopoziomową biblioteką komponentów React skoncentrowaną na dostępności, możliwości dostosowania oraz wygodzie tworzenia interfejsów [37]. Są to komponenty typu *headless*, dlatego zapewniają zachowanie i strukturę, ale pozostawiają pełną kontrolę nad wyglądem. Dokumentacja Radix UI wskazuje, że biblioteka obsługuje między innymi atrybuty ARIA, zarządzanie fokusem oraz nawigację klawiaturą [38]. Dzięki temu ograniczono ryzyko błędów przy samodzielnej implementacji komponentów interaktywnych.

W projekcie prymitywy Radix UI opakowano we własne komponenty w katalogu UI. Do definiowania wariantów komponentów, takich jak rozmiar lub stan wizualny, wykorzystano `class-variance-authority`, a do bezpiecznego łączenia klas Tailwinda zastosowano `tailwind-merge`. Pozwoliło to zachować spójność komponentów przy jednoczesnej elastyczności ich stylowania.

4.8. Komunikacja czasu rzeczywistego

Komunikację czasu rzeczywistego zrealizowano z wykorzystaniem Spring WebSocket oraz subprotokołu STOMP. WebSocket zapewnia dwukierunkowe połączenie między klientem a serwerem, natomiast STOMP definiuje wyższy poziom komunikacji oparty na ramkach, nagłówkach, komendach `SEND` i `SUBSCRIBE` oraz adresach docelowych wiadomości. Dzięki temu obsługa czatu

i powiadomień jest czytelniejsza niż w przypadku bezpośredniego przesyłania surowych wiadomości WebSocket.

Po stronie klienckiej zastosowano biblioteki `@stomp/stompjs` oraz `sockjs-client`, a warstwa serwerowa udostępnia *endpoint* WebSocket pod ścieżką `/ws`. Spring wskazuje `stompjs/stompjs` jako aktywnie utrzymywaną bibliotekę kliencką do obsługi STOMP w przeglądarce [39]. SockJS umożliwia natomiast korzystanie z interfejsu podobnego do WebSocket oraz zastosowanie mechanizmów zastępczych, gdy natywne WebSockets nie są dostępne [40].

W projekcie wykorzystano wbudowany broker pamięciowy Springa, czyli `SimpleBroker`. Odpowiada on za przechowywanie subskrypcji i przekazywanie wiadomości do klientów korzystających z kanałów takich jak `/topic` i `/queue`. Dodatkowo zastosowano kolejki użytkownika, które pozwalają kierować wiadomości do konkretnego odbiorcy, co jest szczególnie istotne w module czatu. Rozwiązanie to jest proste operacyjnie i wystarczające dla wdrożenia jednoprosesowego, jednak ogranicza skalowanie horyzontalne, ponieważ subskrypcje są utrzymywane w pamięci procesu aplikacji.

Uwierzytelnianie połączenia WebSocket przebiega dwuetapowo. W fazie uzgadniania połączenia HTTP warstwa serwerowa odczytuje token JWT z ciasteczki i odrzuca połączenie, jeżeli token jest niepoprawny lub wygasł. Następnie token jest ponownie walidowany w ramce STOMP CONNECT, a do kontekstu wiadomości przypisywany jest użytkownik wraz z rolą. Pozwala to kierować wiadomości do kolejek przypisanych do konkretnego użytkownika oraz ograniczać dostęp do kanałów zgodnie z tożsamością zalogowanej osoby. Spring Security wskazuje, że informacje uwierzytelniające z żądania HTTP mogą zostać przekazane do kontekstu WebSocket jako `Principal`.

4.9. Docker i BuildKit

Do konteneryzacji aplikacji zastosowano technologię Docker. Konteneryzacja polega na uruchamianiu aplikacji w odizolowanych, lekkich środowiskach współdzielących jądro systemu operacyjnego, co odróżnia ją od pełnej wirtualizacji opartej na maszynach wirtualnych [41]. Obraz kontenera zawiera aplikację wraz z jej zależnościami, dzięki czemu zachowuje się identycznie niezależnie od środowiska uruchomieniowego.

Wykorzystanie kontenerów pozwoliło ujednolicić środowisko uruchomieniowe pomiędzy maszynami lokalnymi członków zespołu, eliminując problemy wynikające z różnic w konfiguracji systemów. Ponadto konteneryzacja umożliwiła przeniesienie procesu budowania i uruchamiania aplikacji do środowiska serwerowego bez konieczności dodatkowej konfiguracji [41] co bezpośrednio wspiera proces automatycznego wdrażania.

Do budowania obrazów kontenerowych wewnątrz klastra Kubernetes zastosowano BuildKit - mechanizm budowania rozwijany jako rdzeń platformy Docker, zapewniający między innymi równoległe wykonywanie etapów oraz efektywne buforowanie warstw [42]. Pierwotnie wykorzystywano w tym celu narzędzie Kaniko [43], które zastąpiono ze względu na ograniczoną aktywność rozwojową projektu oraz niższą wydajność. Szczegóły konfiguracji procesu budowania opisano w podrozdziale 8.3.

4.10. Kubernetes i Helm

Do orkiestracji kontenerów wybrano Kubernetes - system zarządzania uruchamianiem i cyklem życia aplikacji kontenerowych, działający w oparciu o model deklaracyjny, w którym administrator definiuje pożądany stan systemu, a platforma samodzielnie dąży do jego utrzymania [44]. Kubernetes

odpowiada między innymi za rozmieszczanie kontenerów, automatyczne odtwarzanie uszkodzonych instancji oraz skalowanie aplikacji.

W projekcie zastosowano lekką dystrybucję k3s, zawierającą wbudowaną i uproszczoną warstwę sterującą (ang. *control plane*), co ułatwia instalację i administrację [45]. Wybór ten był podyktowany doświadczeniem autorów z platformą Kubernetes oraz niskimi wymaganiami zasobowymi dystrybucji k3s, co miało kluczowe znaczenie przy ograniczonej infrastrukturze sprzętowej. Celem było przygotowanie środowiska wdrożeniowego zbliżonego do warunków produkcyjnych - obejmującego separację środowisk, zarządzanie wdrożeniami, kontrolę dostępu oraz automatyzację. Z tego względu odrzucono narzędzie Docker Compose, ograniczone do lokalnego uruchamiania kontenerów, na rzecz samodzielnie hostowanego klastra.

Do zarządzania wdrożeniami zastosowano menedżer pakietów Helm, który umożliwia opisanie aplikacji w postaci tzw. wykresów (ang. *chart*) oraz parametryzację konfiguracji, dzięki czemu tę samą aplikację można wdrażać w różnych środowiskach z odmiennymi ustawieniami [46]. Szczegóły konfiguracji klastra oraz wykresów Helm opisano w podrozdziale 8.2.

4.11.Jenkins

Jako platformę ciągłej integracji i ciągłego dostarczania (ang. *Continuous Integration / Continuous Delivery, CI/CD*) wybrano Jenkins [47]. Idea ciągłego dostarczania zakłada automatyzację procesu budowania, testowania i wdrażania oprogramowania w sposób powtarzalny i niezawodny, co skraca czas dostarczania zmian oraz ogranicza ryzyko błędów wynikających z operacji wykonywanych ręcznie [48].

Kluczowym czynnikiem przemawiającym za wyborem Jenkinsa była możliwość samodzielnego hostowania narzędzia wewnątrz klastra Kubernetes, bez konieczności wystawiania usług na zewnątrz zamkniętego środowiska serwerowego. Rozważano wykorzystanie usług GitLab oraz Circle CI, jednak wiązałyby się one z udostępnieniem infrastruktury poza środowisko lokalne albo z poniesieniem dodatkowych kosztów usługi zewnętrznej.

Procesy w Jenkinsie zdefiniowano zgodnie z podejściem Pipeline as Code, w którym przebieg potoku zapisywany jest w plikach przechowywanych w repozytorium kodu, w deklaratywnej składni opartej na języku Groovy [47]. Dzięki temu konfiguracja procesów wdrożeniowych podlega kontroli wersji na równi z kodem aplikacji. Szczegóły potoków *CI/CD* opisano w podrozdziale 8.3.

4.12.Prometheus i Grafana

Do monitorowania infrastruktury oraz aplikacji zastosowano Prometheus [49] oraz Grafanę [50]. Prometheus jest systemem zbierania i przechowywania metryk w postaci szeregów czasowych, działającym w modelu cyklicznego pobierania danych (ang. *pull*) z monitorowanych komponentów i udostępniającym własny język zapytań do ich analizy [49]. W projektowanym systemie Prometheus odpowiada za gromadzenie metryk pochodzących z klastra Kubernetes oraz z aplikacji serwerowej.

Grafana pełni rolę warstwy wizualizacji, umożliwiając prezentację zgromadzonych metryk w formie konfigurowalnych pulpitów (ang. *dashboard*) oraz analizę zużycia zasobów i zachowania aplikacji w czasie [50]. Połączenie obu narzędzi tworzy spójne środowisko obserwowalności (ang. *observability*) systemu.

Wybór tych narzędzi był podyktowany wcześniejszym doświadczeniem autorów z oboma rozwiązaniami oraz ich pozycją jako standardu branżowego w obszarze monitorowania systemów kontenerowych. Szczegóły konfiguracji monitoringu opisano w podrozdziale 8.5.

4.13.Trivy

Do skanowania obrazów kontenerowych pod kątem znanych podatności (ang. *CVE - Common Vulnerabilities and Exposures*) zastosowano Trivy [51]. Narzędzie analizuje zawartość obrazu, w tym pakiety systemu operacyjnego oraz zależności aplikacji, i porównuje je z bazami znanych podatności, raportując wykryte zagrożenia wraz z ich poziomem istotności [51].

Trivy uruchamiane jest w ramach potoku *CI/CD* po zbudowaniu obrazu, a przed jego wdrożeniem na środowisko. Takie umiejscowienie realizuje zasadę wczesnego wykrywania problemów bezpieczeństwa (ang. *shift-left security*), dzięki czemu podatne obrazy mogą zostać zatrzymane jeszcze przed wdrożeniem. Wybór narzędzia był podyktowany jego popularnością w ekosystemie Kubernetes oraz prostotą integracji z potokiem wdrożeniowym. Szczegóły integracji opisano w podrozdziale 8.3.

4.14.Narzędzia wspierające

W ramach infrastruktury projektu uruchomiono prywatny rejestr obrazów Docker (ang. *Docker Registry*) wewnątrz klastra Kubernetes, służący do przechowywania obrazów budowanych w potoku *CI/CD*. Zdecydowano się na rejestr prywatny zamiast publicznych usług (np. Docker Hub, GitHub Container Registry), dzięki czemu obrazy pozostają w zamkniętym środowisku serwerowym, a rozwiązanie nie generuje dodatkowych kosztów ani ograniczeń związanych z limitami pobierania.

Do wystawiania usług aplikacyjnych zastosowano serwer Nginx, pełniący funkcję serwera treści statycznych oraz odwrotnego pośrednika (ang. *reverse proxy*). Wybór był podyktowany prostotą konfiguracji oraz stabilnością tego rozwiązania [52]. W warstwie klienckiej Nginx odpowiada za serwowanie zbudowanej aplikacji jednostronicowej oraz kierowanie żądań do warstwy serwerowej.

Do kontroli wersji kodu wykorzystano system Git [53], z repozytoriami hostowanymi na platformie GitHub. Wydzielono trzy repozytoria: aplikacji serwerowej, aplikacji klienckiej oraz konfiguracji infrastrukturalnej. System kontroli wersji umożliwił równoległą pracę zespołu oraz śledzenie historii zmian, a strukturę repozytorium infrastrukturalnego opisano w podrozdziale 8.3.

Na etapie projektowania interfejsu użytkownika posłużono się narzędziem Figma [54], w którym opracowano prototypy widoków aplikacji przed ich implementacją. Pozwoliło to uzgodnić układ i przepływ ekranów w zespole, zanim rozpoczęto prace nad warstwą kliencką.

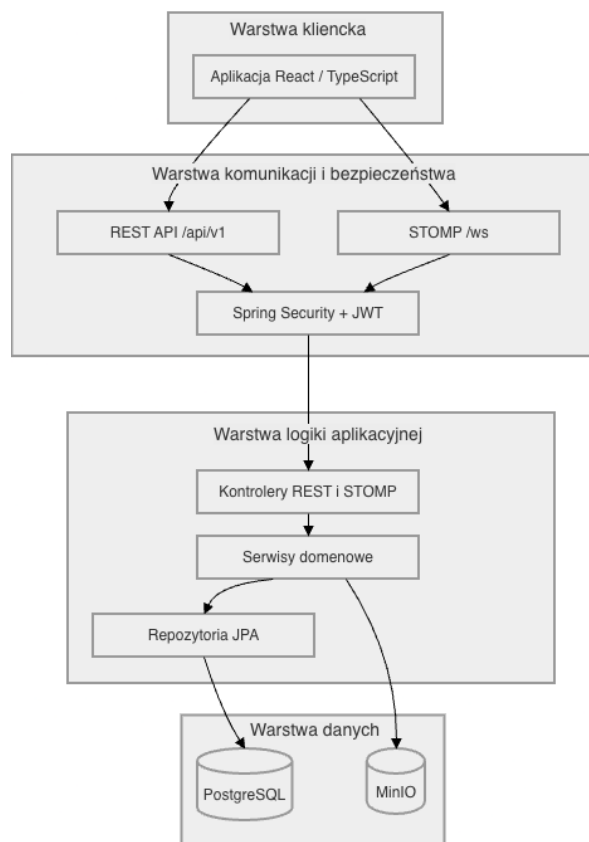
5. Implementacja systemu

W niniejszym rozdziale przedstawiono implementację systemu Coachify. Opis obejmuje architekturę logiczną, podział warstwowy serwera, mechanizmy bezpieczeństwa, moduły trenera i podopiecznego, czat, *marketplace*, obsługę plików oraz panel administratora. W opisie skoncentrowano się na mechanizmach aplikacyjnych, przepływach danych, regułach domenowych oraz integracjach technicznych wykorzystywanych przez poszczególne moduły systemu.

5.1. Architektura systemu

W podrozdziale przedstawiono ogólną organizację systemu Coachify oraz sposób podziału odpowiedzialności pomiędzy aplikację kliencką, aplikację serwerową, bazę danych i magazyn obiektowy. Zastosowana architektura została oparta na wydzieleniu warstwy komunikacji HTTP, zabezpieczeń, kontrolerów, serwisów domenowych oraz repozytoriów. Dzięki temu logika biznesowa została odseparowana od transportu danych, a poszczególne moduły mogą być rozwijane bez naruszania wspólnego modelu bezpieczeństwa i dostępu do danych.

Rysunek 11 przedstawia logiczną architekturę systemu Coachify wraz z głównymi zależnościami pomiędzy komponentami.



Rysunek 11. Logiczna architektura warstwowa systemu Coachify. Źródło: opracowanie własne.

Na rysunku 11 wyróżniono dwie główne części systemu: aplikację kliencką oraz serwerową. Komunikacja synchroniczna została zrealizowana przez REST API, natomiast komunikacja czasu rzeczywistego przez STOMP. Warstwa serwisów domenowych korzysta z repozytoriów JPA oraz z magazynu MinIO, co pozwala oddzielić operacje biznesowe od szczegółów trwałego zapisu danych i plików.

5.1.1 Architektura logiczna

Architektura logiczna systemu została oparta na centralizacji komunikacji z API po stronie aplikacji klienckiej oraz na przekazywaniu żądań do kontrolerów REST po stronie serwera. W takim układzie komponenty aplikacji klienckiej nie wykonują bezpośrednio operacji transportowych, lecz korzystają ze wspólnego klienta HTTP. Rozwiązanie to ogranicza duplikację obsługi nagłówków, danych wejściowych, ciasteczek oraz interceptorów odpowiedzialnych za reakcję na odpowiedzi serwera.

Listing 1. Centralizacja ruchu HTTP w kliencie API.

```
private async request<T>(endpoint: string, options: RequestOptions): Promise<T> {
  const url = `${this.baseUrl}${endpoint}`;
  const performFetch = (): Promise<Response> => fetch(url,
    this.buildInit(options));

  let response = await performFetch();

  for (const interceptor of this.interceptors) {
    response = await interceptor(response, performFetch, options);
  }
  return this.handleResponse<T>(response);
}

private buildInit(options: RequestOptions): RequestInit {
  const headers: Record<string, string> = {};

  if (!options.isFormData) {
    headers['Content-Type'] = 'application/json';
  }

  let body: BodyInit | undefined;

  if (options.body !== undefined) {
    body = options.isFormData ? (options.body as BodyInit) :
    JSON.stringify(options.body);
  }

  return {
    method: options.method ?? 'GET',
    credentials: 'include',
    headers,
    body,
    signal: options.signal,
  };
}
```

Listing 1 przedstawia centralny klient HTTP aplikacji klienckiej odpowiedzialny za składanie żądań i obsługę wspólnych ustawień transportu. Przedstawiony fragment znajduje się w warstwie komunikacji i odpowiada za jednolite przygotowanie żądań HTTP. Szczególne znaczenie ma ustawienie `credentials: 'include'`, ponieważ umożliwia przesyłanie ciasteczek wykorzystywanych przez

mechanizm uwierzytelniania. Zastosowanie interceptorów pozwala rozszerzać zachowanie klienta bez modyfikowania kodu wywołującego poszczególne *endpointy*.

5.1.2 Podział warstwowy warstwy serwerowej

Aplikacja serwerowa systemu została podzielona na warstwę kontrolerów, serwisów domenowych oraz repozytoriów. Kontrolery odpowiadają za przyjęcie żądania i przekazanie kontekstu bezpieczeństwa, natomiast serwisy realizują logikę biznesową oraz pobierają właściwe profile domenowe. Dzięki temu informacje o aktualnym użytkowniku nie są rozpraszane w wielu klasach kontrolerów.

Listing 2 pokazuje serwis kontekstu użytkownika, który udostępnia serwisom domenowym aktualnego użytkownika, profil trenera oraz profil podopiecznego.

Listing 2. Wspólny kontekst użytkownika dla serwisów domenowych.

```
@Service
public class CurrentUserService {
    private final UserRepository userRepository;
    private final TrainerProfileRepository trainerProfileRepository;
    private final ClientProfileRepository clientProfileRepository;

    public User getCurrentUser(Authentication authentication) {
        return userRepository.findByEmail(authentication.getName())
            .orElseThrow(() -> new
ResourceNotFoundException("Użytkownik nie został znaleziony"));
    }

    public TrainerProfile getCurrentTrainerProfile(Authentication
authentication) {
        User user = getCurrentUser(authentication);
        return trainerProfileRepository.findById(user.getId())
            .orElseThrow(() -> new ResourceNotFoundException("Profil
trenera nie został znaleziony."));
    }

    public ClientProfile getCurrentClientProfile(Authentication
authentication) {
        User user = getCurrentUser(authentication);
        return clientProfileRepository.findById(user.getId())
            .orElseThrow(() -> new ResourceNotFoundException("Profil
klienta nie został znaleziony."));
    }
    public User getUserByEmail(String email) {
        return userRepository.findByEmail(email)
            .orElseThrow(() -> new
ResourceNotFoundException("Użytkownik nie został znaleziony"));
    }

    public TrainerProfile getTrainerByEmail(String email) {
        return trainerProfileRepository.findByUserEmail(email)
            .orElseThrow(() -> new ResourceNotFoundException("Profil
trenera nie został znaleziony."));
    }
    public ClientProfile getClientByEmail(String email) {
        return clientProfileRepository.findByUserEmail(email)
    }
```

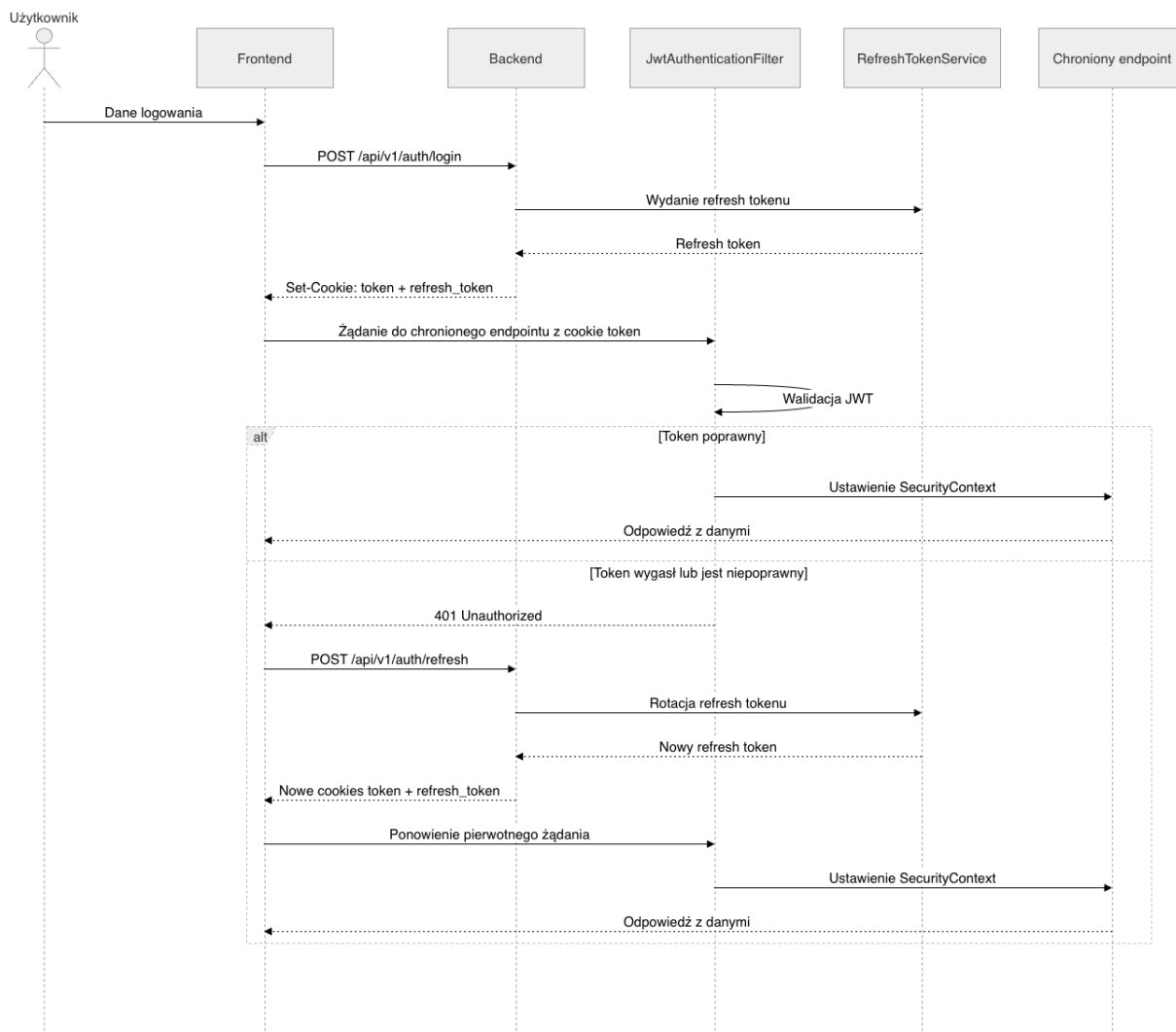
```
        .orElseThrow(() -> new ResourceNotFoundException("Profil  
klienta nie został znaleziony"));  
    }  
}
```

Najważniejszym elementem listingu 2 jest oddzielenie technicznej tożsamości *Authentication* od profili domenowych wykorzystywanych w logice aplikacji. Serwis pobiera dane przez repozytoria, a w przypadku braku wymaganej encji zgłasza wyjątek domenowy. Takie rozwiązanie wzmacnia podział warstwowy aplikacji serwerowej, ponieważ serwisy biznesowe otrzymują spójny sposób dostępu do aktualnego użytkownika bez powielania zapytań.

5.2. Uwierzytelnianie i autoryzacja

Mechanizm bezpieczeństwa został oparty na krótkotrwałym tokenie JWT oraz rotowanym tokenie odświeżającym. Token dostępowy jest przechowywany w ciasteczku *HttpOnly* i odczytywany przez filtr uwierzytelniający przy żądaniach do chronionych *endpointów*. W konfiguracji pozostawiono również możliwość obsługi tokenu przekazanego w nagłówku *Authorization*. Autoryzacja została powiązana z regułami tras w Spring Security oraz z rolą użytkownika zapisaną w tokenie JWT.

Rysunek 12 przedstawia sekwencję logowania, użycia chronionego *endpointu* oraz odświeżania sesji po wygaśnięciu tokenu dostępowego.



Rysunek 12. Sekwencja logowania i rotacji tokenu. Źródło: opracowanie własne.

Na rysunku 12 pokazano, że uwierzytelnienie nie kończy się wyłącznie na wydaniu tokenu dostępowego. Po wygaśnięciu tokenu JWT aplikacja kliencka otrzymuje odpowiedź 401 Unauthorized, a następnie wywołuje *endpoint* odświeżania sesji. Serwer weryfikuje token odświeżający, rotuje go i zwraca nową parę ciasteczek. Dzięki temu czas życia tokenu JWT może być krótki, a ponowne użycie unieważnionego tokenu odświeżającego może zostać wykryte i obsłużone.

Listing 3 przedstawia konfigurację ochrony tras i umieszczenie filtra JWT w łańcuchu Spring Security.

Listing 3. Konfiguracja *stateless security* i publicznych *endpointów*.

```
@Bean
public SecurityFilterChain securityFilterChain(HttpSecurity http) throws
Exception {
    http
        .cors(cors -> cors.configurationSource(corsConfigurationSource()))
        .csrf(csrf -> csrf.disable())
        .authorizeHttpRequests(auth -> auth
            .requestMatchers("/health").permitAll()
            .requestMatchers("/actuator/health/**",
                "/actuator/info").permitAll()
            .requestMatchers(
                "/api/v1/auth/login",
                "/api/v1/auth/register",
                "/api/v1/auth/logout",
                "/api/v1/auth/refresh"
            ).permitAll()
            .requestMatchers("/api/v1/marketplace/**").permitAll()
            .requestMatchers("/api/v1/trainers/*/avatar").permitAll()
            .requestMatchers("/api/v1/exercises//media",
                "/api/v1/exercises//media/status").permitAll()
            .requestMatchers("/api/v1/exercises/media/batch").permitAll()
            .requestMatchers("/api/v1/exercises/enums/**").permitAll()
            .requestMatchers("/ws/**").permitAll()
            .anyRequest().authenticated()
        )
        .formLogin(form -> form.disable())
        .httpBasic(basic -> basic.disable())
        .sessionManagement(session ->
            session.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
        .exceptionHandling(ex -> ex
            .authenticationEntryPoint(jwtAuthenticationEntryPoint)
            .accessDeniedHandler(jwtAccessDeniedHandler)
        )
        .addFilterBefore(jwtAuthenticationFilter,
            UsernamePasswordAuthenticationFilter.class);
    return http.build();
}
```

Konfiguracja wskazuje na bezstanowy model bezpieczeństwa, w którym sesja serwerowa nie jest utrzymywana przez Spring Security. Publiczny dostęp dopuszczono tylko dla wybranych *endpointów*, takich jak logowanie, rejestracja, odświeżenie tokenu, *marketplace* oraz *handshake* WebSocket. Wszystkie pozostałe żądania wymagają uwierzytelnienia, a filtr JWT zostaje wykonany przed standardowym filtrem logowania.

Listing 4 prezentuje mechanizm odczytu JWT, walidacji tokenu oraz ustawienia `SecurityContext`.

Listing 4. Odtwarzanie kontekstu bezpieczeństwa z JWT.

```
if (request.getCookies() != null) {
    for (Cookie cookie : request.getCookies()) {
        if ("token".equals(cookie.getName())) {
            token = cookie.getValue();
            break;
        }
    }
}
if (token == null) {
    String authHeader = request.getHeader("Authorization");
    if (authHeader != null && authHeader.startsWith("Bearer ")) {
        token = authHeader.substring(7);
    }
}
try {
    if (token != null && jwtUtil.validateToken(token) &&
    !jwtUtil.isTokenExpired(token)) {
        String email = jwtUtil.extractEmail(token);
        String role = jwtUtil.extractRole(token);
        Optional<User> user = userRepository.findByEmail(email).map(User::getId);
        if (user.isEmpty()) {
            filterChain.doFilter(request, response);
            return;
        }
        UsernamePasswordAuthenticationToken authentication = new
        UsernamePasswordAuthenticationToken(
            new AuthenticatedUser(email, user.getId()),
            null,
            Collections.singletonList(new SimpleGrantedAuthority("ROLE_" +
        role)))
        );
        authentication.setDetails(new
        WebAuthenticationDetailsSource().buildDetails(request));
        SecurityContextHolder.getContext().setAuthentication(authentication);
        MDC.put(MDC_USER_ID, email);
    }
    filterChain.doFilter(request, response);
} finally {
    MDC.remove(MDC_USER_ID);
}
```

W listingu 4 pokazano, że kontekst bezpieczeństwa jest odtwarzany tylko po poprawnej walidacji tokenu i potwierdzeniu istnienia użytkownika w bazie danych. Rola wyodrębniona z JWT zostaje przekształcona na `SimpleGrantedAuthority`, co umożliwia późniejszą autoryzację w mechanizmach Spring Security. Dodatkowe użycie MDC pozwala powiązać wpisy diagnostyczne z aktualnym użytkownikiem w trakcie obsługi żądania.

Listing 5 pokazuje rotację tokenu odświeżającego oraz reakcję systemu na ponowne użycie starego tokenu.

Listing 5. Rotacja *refresh* tokenu i wykrywanie *replay attack*.

```
@Transactional(noRollbackFor = Exception.class)
public RotationResult rotateRefreshToken(String rawToken) {
    String hash = sha256Hex(rawToken);
    RefreshToken existing =
        refreshTokenRepository
            .findByTokenHash(hash)
            .orElseThrow(() -> new UnauthorizedException("Token odświeżający jest
nieprawidłowy"));
    if (existing.isExpired()) {
        throw new UnauthorizedException("Token odświeżający wygasł");
    }

    User owner = existing.getUser();
    String newRawToken = generateRawToken();
    String newHash = sha256Hex(newRawToken);
    LocalDateTime newExpiresAt =
        LocalDateTime.now().plusSeconds(jwtProperties.refreshExpiration() / 1000);

    int rowsUpdated = refreshTokenRepository.revokeIfActive(hash, newHash);
    if (rowsUpdated == 0) {
        log.warn(
            "Refresh-token rotation lost the race or replayed for user {} - revoking
all active tokens",
            owner.getId());
        refreshTokenRepository.revokeAllByUserId(owner.getId());
        throw new UnauthorizedException("Token odświeżający został unieważniony");
    }
    refreshTokenRepository.save(new RefreshToken(owner, newHash, newExpiresAt));
    return new RotationResult(owner, newRawToken);
}
```

Przedstawiony mechanizm realizuje zasadę jednorazowości tokenu odświeżającego. W bazie danych zapisywany jest skrót tokenu, a nie jego surowa wartość. Atomowa operacja `revokeIfActive` ogranicza ryzyko równoległego albo powtórnego użycia tego samego tokenu, ponieważ w przypadku niepowodzenia rotacji aktywne tokeny użytkownika zostają unieważnione.

Listing 6 przedstawia kliencką obsługę odpowiedzi 401, odświeżenie sesji i ponowienie pierwotnego żądania.

Listing 6. Interceptor odświeżania sesji po stronie aplikacji klienckiej.

```
installAuthRefresh(opts: {
    refresh: () => Promise;
    onRefreshFailure: () => void;
}): void {
    this.interceptors = this.interceptors.filter(i => !(i as { authRefresh?:
boolean }).authRefresh);
    const interceptor: ResponseInterceptor & { __authRefresh?: boolean } = async (
        response,
        retry,
        options,
    ) => {
        if (response.status !== 401 || options.skipAuthRefresh) {
            return response;
        }
        try {
            await this.singleFlightRefresh(opts.refresh);
        } catch {
            opts.onRefreshFailure();
        }
    }
}
```

```

        return response;
    }
    return retry();
};
interceptor.__authRefresh = true;
this.interceptors.push(interceptor);
}
private singleFlightRefresh(refresh: () => Promise): Promise {
    if (!this.refreshPromise) {
        this.refreshPromise = refresh().finally(() => {
            this.refreshPromise = null;
        });
    }
    return this.refreshPromise;
}
}

```

Listing 6 uzupełnia serwerowy mechanizm rotacji tokenu o obsługę po stronie klienta HTTP. W przypadku odpowiedzi 401 wykonywana jest próba odświeżenia sesji, a następnie ponawiane jest pierwotne żądanie. Mechanizm `singleFlightRefresh` zapobiega wysłaniu wielu równoległych żądań odświeżenia, co stabilizuje zachowanie aplikacji przy jednoczesnym wygaśnięciu tokenu dla kilku aktywnych zapytań.

5.3. Moduł trenera

Moduł trenera został zaimplementowany wokół operacji przyjęcia zgłoszenia współpracy, utworzenia relacji trener-podopieczny oraz przygotowania planu zgodnego z zakresem wykupionej usługi. Reguły domenowe zostały umieszczone w warstwie serwisowej, ponieważ w tej warstwie wymagane jest połączenie danych zgłoszenia, profilu podopiecznego, przypisania trenerskiego, kwestionariusza oraz mechanizmów powiązanych z komunikacją.

W ramach przyjęcia zgłoszenia konieczne jest sprawdzenie, czy zgłoszenie nadal oczekuje na decyzję oraz czy podopieczny nie posiada już aktywnej współpracy z innym trenerem. Listing 7 przedstawia akceptację zgłoszenia i utworzenie relacji biznesowej.

Listing 7. Akceptacja zgłoszenia i utworzenie relacji trener-klient.

```

if (request.getStatus() != RequestStatus.PENDING) {
    throw new BadRequestException("To zgłoszenie zostało już rozpatrzone.");
}
ClientProfile client = request.getClient();
if (assignmentRepository.existsByClientAndActiveTrue(client)) {
    request.setStatus(RequestStatus.REJECTED);
    request.setRespondedAt(LocalDateTime.now());
    trainingRequestRepository.save(request);
    throw new BadRequestException("Klient ma już aktywną współpracę z innym trenerem.");
}
ClientQuestionnaire questionnaire = questionnaireRepository
    .findFirstByClientProfileAndStatusOrderByCompletedAtDesc(client,
        QuestionnaireStatus.COMPLETED)
    .orElse(null);
request.setStatus(RequestStatus.ACCEPTED);
request.setRespondedAt(LocalDateTime.now());
trainingRequestRepository.save(request);
ClientTrainerAssignment assignment = new ClientTrainerAssignment(
    client,
    trainer,
    request.getServiceType(),
    request,

```

```

        questionnaire
    );
    client.setTrainer(trainer);
    clientProfileRepository.save(client);
    ClientTrainerAssignment savedAssignment = assignmentRepository.save(assignment);
    autoUnblockIfBlocked(trainer, client);
    createOrUpdateConversation(trainer, client);
    return savedAssignment;

```

W przedstawionym fragmencie najpierw weryfikowany jest status zgłoszenia, a następnie sprawdzane jest istnienie aktywnego przypisania podopiecznego do trenera. W przypadku wykrycia aktywnej współpracy zgłoszenie zostaje oznaczone jako odrzucone, co zabezpiecza system przed równoległym prowadzeniem tego samego podopiecznego przez wielu trenerów. Po pozytywnej weryfikacji pobierany jest ostatni ukończony kwestionariusz, który zostaje powiązany z tworzącym przypisaniem `ClientTrainerAssignment`. Dodatkowo aktualizowany jest profil podopiecznego oraz inicjowana jest konwersacja, dzięki czemu relacja domenowa zostaje zsynchronizowana z modulem komunikacji.

Drugim kluczowym obszarem modułu trenera jest tworzenie planu treningowego. W tym przypadku walidacja uprawnień nie została ograniczona do sprawdzenia roli użytkownika, lecz została powiązana z aktywną relacją trener-podopieczny oraz typem wykupionej usługi. Listing 8 pokazuje tworzenie planu treningowego z kontrolą zakresu usługi.

Listing 8. Tworzenie planu z walidacją aktywnej współpracy.

```

@Transactional
public PlanResponse createPlan(String trainerEmail, Long clientId,
    CreatePlanRequest request) {
    TrainerProfile trainer = currentUser.getTrainerByEmail(trainerEmail);
    ClientProfile client = clientProfileRepository.findById(clientId)
        .orElseThrow(() -> new ResourceNotFoundException("Klient nie został
znaleziony"));
    validateClientScope(trainer, client);
    TrainingPlan plan = new TrainingPlan(trainer, client, request.name());
    plan.setDescription(request.description());
    plan = planRepository.save(plan);
    if (request.days() != null && !request.days().isEmpty()) {
        updatePlanDays(plan, request.days(), trainer.getId());
        plan = planRepository.save(plan);
    }
    return PlanResponse.fromEntity(plan);
}

private void validateClientScope(TrainerProfile trainer, ClientProfile client) {
    ClientTrainerAssignment assignment = assignmentRepository
        .findByTrainerAndClientIdAndActiveTrue(trainer, client.getId())
        .orElseThrow(() -> new BadRequestException("Nie masz aktywnego
przypisania do tego klienta"));
    ServiceType serviceType = assignment.getServiceType();
    if (serviceType == ServiceType.DIET_PLAN) {
        throw new BadRequestException(
            "Klient ma wykupioną tylko usługę dietetyczną. " +
            "Nie możesz tworzyć dla niego planów treningowych."
        );
    }
}

```

Metoda została oznaczona adnotacją `@Transactional`, dlatego utworzenie planu oraz ewentualna aktualizacja dni planu są wykonywane w ramach jednej operacji transakcyjnej. Przed zapisem planu pobierany jest profil trenera oraz profil podopiecznego, a następnie wykonywana jest

walidacja `validateClientScope`. Najważniejszym elementem tej walidacji jest sprawdzenie aktywnego przypisania oraz typu usługi zapisanej w relacji `ClientTrainerAssignment`. Jeżeli podopieczny posiada wyłącznie usługę dietetyczną, zgłaszany jest wyjątek, przez co uniemożliwione zostaje utworzenie planu treningowego poza zakresem wykupionej współpracy.

5.4. Moduł podopiecznego

Moduł podopiecznego został oparty na procesie *onboardingu*, obsłudze kwestionariusza oraz raportowaniu wykonanych treningów. W warstwie serwisowej skoncentrowano walidację cyklu życia kwestionariusza, kontrolę kompletności danych oraz mapowanie raportów treningowych na strukturę domenową powiązaną z planem treningowym.

W procesie *onboardingu* dane podopiecznego są zapisywane etapami, a zakończenie kwestionariusza wymaga przejścia pełnej walidacji. Listing 9 prezentuje zapis kroków kwestionariusza i zakończenie *onboardingu*.

Listing 9. Aktualizacja i zakończenie kwestionariusza podopiecznego.

```
@Transactional
public QuestionnaireResponse updateStep(String email, Long questionnaireId, int
stepNumber, StepUpdateRequest request) {
    ClientProfile profile = getProfileByEmail(email);
    ClientQuestionnaire questionnaire =
getQuestionnaireForClient(questionnaireId, profile);
    if (questionnaire.getStatus() == QuestionnaireStatus.COMPLETED) {
        throw new BadRequestException("Nie można edytować ukończonego
kwestionariusza");
    }
    if (stepNumber < 1 || stepNumber > TOTAL_STEPS) {
        throw new BadRequestException("Nieprawidłowy numer kroku: " +
stepNumber);
    }
    applyStepUpdate(questionnaire, stepNumber, request);
    questionnaire.markStepCompleted(stepNumber);
    questionnaire = questionnaireRepository.save(questionnaire);
    return QuestionnaireResponse.fromEntity(questionnaire);
}

@Transactional
public QuestionnaireResponse complete(String email, Long questionnaireId) {
    ClientProfile profile = getProfileByEmail(email);
    ClientQuestionnaire questionnaire =
getQuestionnaireForClient(questionnaireId, profile);
    if (questionnaire.getStatus() == QuestionnaireStatus.COMPLETED) {
        throw new BadRequestException("Kwestionariusz został już ukończony");
    }
    validateCompletion(questionnaire);
    questionnaire.complete();
    questionnaire = questionnaireRepository.save(questionnaire);
    if (!profile.isOnboardingCompleted()) {
        onboardingService.markOnboardingCompleted(profile);
    }
    return QuestionnaireResponse.fromEntity(questionnaire);
}
```

Obie operacje zostały objęte transakcją, co zapewnia spójność zapisu stanu kwestionariusza i profilu podopiecznego. Przed aktualizacją pobierany jest profil powiązany z adresem e-mail oraz kwestionariusz należący do tego profilu. Edycja ukończonego kwestionariusza jest blokowana, a numer kroku jest sprawdzany względem dozwolonego zakresu. Zakończenie ankiety wymaga walidacji

kompletności, zmiany statusu kwestionariusza oraz oznaczenia *onboardingu* jako ukończonego, jeżeli profil nie posiadał jeszcze takiego stanu.

Po przygotowaniu planu treningowego przez trenera podopieczny może raportować wykonanie treningu. Raport nie jest zapisywany jako niezależny formularz, lecz zostaje powiązany z konkretnym planem, dniem planu oraz ćwiczeniami należącymi do tego dnia. Listing 10 pokazuje tworzenie raportu treningowego.

Listing 10. Mapowanie raportu treningowego na strukturę domenową.

```
@Transactional
public WorkoutReportResponse createReport(String clientEmail,
CreateWorkoutReportRequest request) {
    ClientProfile client = currentUser.getClientByEmail(clientEmail);
    TrainingPlan trainingPlan = trainingPlanRepository.findByIdAndClientId(
        request.trainingPlanId(), client.getId())
        .orElseThrow(() -> new ResourceNotFoundException("Plan treningowy nie
został znaleziony"));
    PlanDay planDay = trainingPlan.getDays().stream()
        .filter(d -> d.getId().equals(request.planDayId()))
        .findFirst()
        .orElseThrow(() -> new ResourceNotFoundException("Dzień treningowy
nie został znaleziony"));
    WorkoutReport workoutReport = new WorkoutReport(client, trainingPlan,
planDay, request.workoutDate());
    workoutReport.setGeneralNotes(request.generalNotes());
    if (request.exercises() != null && !request.exercises().isEmpty()) {
        Map<Long, PlanDayExercise> exerciseMap = planDay.getExercises().stream()
            .collect(Collectors.toMap(PlanDayExercise::getId, e -> e));
        int orderIndex = 0;
        for (ExerciseReportRequest exerciseRequest : request.exercises()) {
            PlanDayExercise planDayExercise =
exerciseMap.get(exerciseRequest.planDayExerciseId());
            if (planDayExercise == null) {
                throw new ResourceNotFoundException("Ćwiczenie nie istnieje w tym
dniu treningowym");
            }
            ExerciseReport exerciseReport = new ExerciseReport(workoutReport,
planDayExercise, orderIndex++);
            exerciseReport.setSkipped(exerciseRequest.skipped() != null &&
exerciseRequest.skipped());
            exerciseReport.setNotes(exerciseRequest.notes());
            // ... pominięto przepisywanie raportów serii ...
            workoutReport.addExerciseReport(exerciseReport);
        }
    }
    WorkoutReport saved = workoutReportRepository.save(workoutReport);
    return WorkoutReportResponse.fromEntity(saved);
}
```

Operacja tworzenia raportu również została oznaczona adnotacją `@Transactional`, dlatego zapis raportu głównego oraz powiązanych raportów ćwiczeń odbywa się spójnie. Najpierw wyszukiwany jest plan treningowy przypisany do zalogowanego podopiecznego, a następnie weryfikowane jest istnienie wskazanego dnia planu. Dla ćwiczeń tworzona jest mapa identyfikatorów, co pozwala sprawdzić, czy każde raportowane ćwiczenie rzeczywiście należy do wybranego dnia treningowego. Żądanie zostaje przekształcone w agregat `WorkoutReport -> ExerciseReport -> SetReport`, dzięki czemu historia wykonanych treningów pozostaje zgodna z planem przypisanym podopiecznemu.

5.5. Czat

Moduł czatu został zrealizowany jako mechanizm komunikacji w czasie rzeczywistym oparty na WebSocket oraz protokole STOMP. W implementacji połączono warstwę transportową, kontroler komunikatów, serwis domenowy oraz subskrypcje zdarzeń po stronie klienta. Dzięki temu wysyłanie wiadomości, obsługa potwierdzeń odczytu oraz przekazywanie błędów zostały rozdzielone na jasno określone odpowiedzialności.

Konfiguracja komunikacji w czasie rzeczywistym została umieszczona w warstwie konfiguracyjnej serwera. Listing 11 przedstawia konfigurację brokera STOMP oraz *endpointu* WebSocket.

Listing 11. Konfiguracja brokera STOMP i *endpointu* WebSocket.

```
@Configuration
@EnableWebSocketMessageBroker
public class WebSocketConfig implements WebSocketMessageBrokerConfigurer {
    private final WebSocketAuthInterceptor webSocketAuthInterceptor;
    private final WebSocketHandshakeInterceptor webSocketHandshakeInterceptor;
    @Override
    public void configureMessageBroker(MessageBrokerRegistry config) {
        config.enableSimpleBroker("/topic", "/queue");
        config.setApplicationDestinationPrefixes("/app");
        config.setUserDestinationPrefix("/user");
    }
    @Override
    public void registerStompEndpoints(StompEndpointRegistry registry) {
        registry.addEndpoint("/ws")
            .setAllowedOriginPatterns(
                "http://localhost:5173",
                "http://localhost:*",
                "http://172.21.40.173:*",
                "172.21.40.173"
            )
            .addInterceptors(webSocketHandshakeInterceptor)
            .withSockJS()
            .setSessionCookieNeeded(true);
    }
    @Override
    public void configureClientInboundChannel(ChannelRegistration registration) {
        registration.interceptors(webSocketAuthInterceptor);
    }
}
```

Przedstawiony fragment odpowiada za uruchomienie kanału komunikacji WebSocket oraz zdefiniowanie konwencji adresowania komunikatów STOMP. Prefiks `/app` został przeznaczony dla komunikatów obsługiwanych przez metody oznaczone adnotacją `@MessageMapping`, natomiast prefiks `/user` umożliwia kierowanie zdarzeń do prywatnych kolejek użytkowników. Istotnym elementem konfiguracji jest zastosowanie interceptorów, ponieważ *handshake* oraz kanał wejściowy zostały objęte kontrolą uwierzytelnienia. W efekcie połączenia *real-time* korzystają z mechanizmu identyfikacji użytkownika, a nie wyłącznie z samego faktu nawiązania sesji WebSocket.

Obsługa komunikatów przesyłanych przez STOMP została wydzielona do kontrolera WebSocket. Listing 12 pokazuje sposób przetwarzania wysłania wiadomości oraz potwierdzenia odczytu.

Listing 12. Obsługa komunikatów STOMP w kontrolerze czatu.

```
@PostMapping("/chat.send")
public void sendMessage(@Payload SendMessageRequest request, Principal principal)
{
    if (principal == null) {
        log.warn("Received message from unauthenticated user");
        return;
    }
    String senderEmail = principal.getName();
    String validationError = validate(request);
    if (validationError != null) {
        chatBroadcastService.sendError(senderEmail, validationError);
        return;
    }
    try {
        ChatMessageDTO message = chatService.sendMessage(senderEmail, request);
        chatBroadcastService.broadcastNewMessage(request.conversationId(),
senderEmail, message);
    } catch (Exception e) {
        log.error("Error processing message from {}: {}", senderEmail,
e.getMessage(), e);
        chatBroadcastService.sendError(senderEmail, "Nie udało się wysłać wiadomości:
" + e.getMessage());
    }
}
@PostMapping("/chat.read")
public void markAsRead(@Payload MarkReadRequest request, Principal principal) {
    if (principal == null || request == null || request.conversationId() == null) {
        log.warn("Invalid read receipt request");
        return;
    }
    ReadReceiptDTO readReceipt =
        chatService.markMessagesAsRead(request.conversationId(),
principal.getName());
    chatBroadcastService.broadcastReadReceipt(request.conversationId(),
principal.getName(), readReceipt);
}
```

Kontroler pełni funkcję warstwy protokołu i nie przejmuje odpowiedzialności za reguły biznesowe czatu. Najpierw weryfikowana jest obecność obiektu `Principal`, który reprezentuje uwierzytelnionego użytkownika połączenia `WebSocket`. Następnie wykonywana jest walidacja żądania, a właściwa operacja zapisu wiadomości zostaje przekazana do `ChatService`. Takie rozdzielenie odpowiedzialności pozwala utrzymać logikę domenową poza kontrolerem STOMP oraz umożliwia jednolite wykorzystanie mechanizmów czatu przez różne kanały komunikacji.

Reguły biznesowe związane z wysyłaniem wiadomości zostały zaimplementowane w serwisie domenowym. Listing 13 prezentuje zapis wiadomości wraz z kontrolą dostępu, blokad oraz statusu konwersacji.

Listing 13. Zapis wiadomości z walidacją dostępu i blokad.

```
@Transactional

public ChatMessageDTO sendMessage(String userEmail, SendMessageRequest
request) {

    User sender = currentUser.getUserByEmail(userEmail);
```

```

        Conversation conversation =
findConversationWithAccess(request.conversationId(), sender.getId());

        if (conversation.isBlocked()) {

            throw new BadRequestException("Nie możesz wysłać wiadomości -
konwersacja została zablokowana");

        }

        if (sender.getRole() == Role.TRAINER &&
conversation.isPendingRequest()) {

            throw new BadRequestException("Musisz najpierw zaakceptować
prośbę o wiadomość");

        }

        if (sender.getRole() == Role.CLIENT) {

            Long trainerId = conversation.getTrainer().getId();

            Long clientId = sender.getId();

            if (blockRepository.existsByTrainerIdAndClientId(trainerId,
clientId)) {

                throw new BadRequestException("Nie możesz wysłać wiadomości
do tego trenera");

            }

        }

        ChatMessage message = new ChatMessage(conversation, sender,
request.content());

        message = chatMessageRepository.save(message);

        conversation.updateLastMessageAt();

        conversationRepository.save(conversation);

        return ChatMessageDTO.fromEntity(message);
    }

```

Najważniejszym elementem listingu 13 jest użycie metody `findConversationWithAccess`, ponieważ zapis wiadomości zostaje poprzedzony sprawdzeniem, czy nadawca ma dostęp do wskazanej konwersacji. Dodatkowo uwzględniono blokadę konwersacji, oczekującą prośbę o kontakt oraz blokadę relacji pomiędzy trenerem i podopiecznym. Operacja została oznaczona jako transakcyjna, dzięki czemu zapis wiadomości i aktualizacja czasu ostatniej aktywności konwersacji są traktowane jako jedna spójna zmiana stanu. W efekcie reguły dostępu pozostają wspólne dla komunikacji WebSocket oraz pozostałych ścieżek korzystających z serwisu czatu.

Po stronie klienta zastosowano klienta STOMP, który subskrybuje kolejki użytkownika skonfigurowane w aplikacji serwerowej. Listing 14 pokazuje subskrypcje zdarzeń wiadomości, potwierdzeń odczytu oraz błędów.

Listing 14. Subskrypcje STOMP po stronie aplikacji klienckiej.

```
const socket = new SockJS(WS_URL, null, {
  transports: ['websocket', 'xhr-streaming', 'xhr-polling'],
});
const client = new Client({
  websocketFactory: () => socket,
  reconnectDelay: 5000,
  heartbeatIncoming: 4000,
  heartbeatOutgoing: 4000,
  onConnect: () => {
    setIsConnected(true);
    setIsConnecting(false);
    client.subscribe('/user/queue/messages', (message: IMessage) => {
      const chatMessage = JSON.parse(message.body) as ChatMessage;
      onMessageRef.current?.(chatMessage);
    });
    client.subscribe('/user/queue/read-receipts', (message: IMessage) => {
      const receipt = JSON.parse(message.body) as ReadReceipt;
      onReadReceiptRef.current?.(receipt);
    });
    client.subscribe('/user/queue/errors', (message: IMessage) => {
      const error = JSON.parse(message.body) as { message: string };
      onErrorRef.current?.(error);
    });
    onConnectedRef.current?.();
  },
  onDisconnect: () => {
    setIsConnected(false);
    onDisconnectedRef.current?.();
  },
});
client.activate();
clientRef.current = client;
```

Przedstawiony fragment potwierdza zgodność adresowania zdarzeń z konfiguracją serwerową `/user/queue`. Klient odbiera trzy typy komunikatów: nowe wiadomości, potwierdzenia odczytu oraz błędy zwracane przez serwer. Zastosowanie mechanizmów `reconnectDelay` oraz `heartbeat` zwiększa odporność połączenia na chwilowe przerwy komunikacyjne. W rezultacie moduł czatu realizuje komunikację zdarzeniową bez konieczności cyklicznego odpytywania *endpoints* REST dla każdej zmiany stanu konwersacji.

5.6. Marketplace

Moduł *marketplace* odpowiada za udostępnienie katalogu trenerów spełniających warunki widoczności oraz posiadających skonfigurowaną ofertę usług. Filtrowanie zostało zrealizowane po stronie serwerowej z użyciem JPA Specification, dzięki czemu kryteria wyszukiwania mogą być składane deklaratywnie i przekazywane do repozytorium jako jeden warunek zapytania. Listing 15 przedstawia specyfikację wykorzystywaną do filtrowania trenerów w *marketplace*.

Listing 15. Specyfikacja JPA filtrująca trenerów *marketplace*.

```
public static Specification withFilters(TrainerFilterCriteria criteria) {
    return Specification.allOf(
        isVisibleInMarketplace(),
        hasServiceOfferings(),
        hasId(criteria.id()),
        hasServiceType(criteria.serviceType())
    );
}

private static Specification isVisibleInMarketplace() {
    return (root, query, cb) -> cb.equal(root.get("visibleInMarketplace"), true);
}

private static Specification hasServiceOfferings() {
    return (root, query, cb) -> {
        query.distinct(true);
        return cb.isNotEmpty(root.get("serviceOfferings"));
    };
}

private static Specification hasId(Long id) {
    return (root, query, cb) -> {
        if (id == null) {
            return cb.conjunction();
        }
        return cb.equal(root.get("id"), id);
    };
}

private static Specification hasServiceType(ServiceType serviceType) {
    return (root, query, cb) -> {
        if (serviceType == null) {
            return cb.conjunction();
        }
        query.distinct(true);
        Join<TrainerProfile, TrainerServiceOffering> offerings =
root.join("serviceOfferings", JoinType.INNER);
        return cb.equal(offerings.get("serviceType"), serviceType);
    };
}
```

Przedstawiony fragment znajduje się w warstwie dostępu do danych i odpowiada za ograniczenie wyników do profili spełniających reguły biznesowe *marketplace*. Warunek `isVisibleInMarketplace()` eliminuje profile ukryte, natomiast `hasServiceOfferings()` wymusza istnienie co najmniej jednej oferty usług. Dodatkowo zastosowano opcjonalne kryteria identyfikatora oraz typu usługi. Użycie `query.distinct(true)` ogranicza ryzyko zwrócenia zduplikowanych trenerów po dołączeniu kolekcji ofert.

Po stronie klienta kryteria filtrowania oraz paginacji zostały powiązane z kluczem zapytania TanStack Query. Dzięki temu zmiana numeru strony albo typu usługi powoduje odświeżenie danych zgodnie z aktualnym stanem filtrów. Listing 16 prezentuje obsługę zależności zapytania od paginacji i wybranego typu usługi.

Listing 16. Filtrowanie *marketplace* w TanStack Query.

```
const [page, setPage] = useState(0);
const [selectedServiceType, setSelectedServiceType] = useState<ServiceType |
undefined>();
const pageSize = 9;
const { data: serviceTypes } = useQuery({
  queryKey: ['serviceTypes'],
  queryFn: () => marketplaceService.getServiceTypes(),
});
const { data: trainersResponse, isLoading } = useQuery({
  queryKey: ['marketplace-trainers', page, selectedServiceType],
  queryFn: () => marketplaceService.getTrainers({
    page,
    size: pageSize,
    serviceType: selectedServiceType,
  }),
});
const clearFilters = () => {
  setSelectedServiceType(undefined);
  setPage(0);
};
// ... pominięto pozostałą część komponentu ...
{serviceTypes?.map((st: ServiceTypeOption) => (
  <Button
    key={st.value}
    variant={selectedServiceType === st.value ? 'default' : 'outline'}
    onClick={() => {
      setSelectedServiceType(selectedServiceType === st.value ? undefined :
st.value);
      setPage(0);
    }}
  >
    {SERVICE_TYPE_LABELS[st.value]}
  </Button>
))}
```

Najważniejszym elementem listingu 16 jest `queryKey`, w którym uwzględniono numer strony oraz wybrany typ usługi. Takie rozwiązanie powoduje, że pobieranie danych jest zależne od rzeczywistych parametrów zapytania, a zmiana filtra resetuje paginację do pierwszej strony. W rezultacie warstwa kliencka pozostaje spójna z mechanizmem filtrowania zaimplementowanym w serwerowej specyfikacji JPA.

5.7. Obsługa plików

Obsługa plików została oparta na magazynie obiektowym MinIO, metadanych przechowywanych w bazie danych oraz walidacji przesyłanej zawartości. Pliki nie są udostępniane bezpośrednio przez serwer jako zasoby binarne, lecz za pomocą czasowych adresów podpisanych. Listing 17 przedstawia adapter odpowiedzialny za komunikację z MinIO oraz generowanie podpisanych adresów URL.

Listing 17. Adapter MinIO i generowanie podpisanych URL.

```
@Component
public class MinioGateway {
  private final MinioClient minioClient;
```

```

public void putObject(String bucket, String objectKey, InputStream stream, long
size, String contentType) {
    try {
        minioClient.putObject(
            PutObjectArgs.builder()
                .bucket(bucket)
                .object(objectKey)
                .stream(stream, size, -1)
                .contentType(contentType)
                .build());
    } catch (Exception e) {
        log.error("Failed to upload to {}/{}", bucket, objectKey, e);
        throw new StorageException("Failed to upload file", e);
    }
}
// ... pominięto analogiczne pobieranie obiektu ...
public String getPresignedUrl(String bucket, String objectKey, int expiry,
TimeUnit unit) {
    try {
        return minioClient.getPresignedObjectUrl(
            GetPresignedObjectUrlArgs.builder()
                .method(Method.GET)
                .bucket(bucket)
                .object(objectKey)
                .expiry(expiry, unit)
                .build());
    } catch (Exception e) {
        log.error("Failed to generate presigned URL for {}/{}", bucket, objectKey,
e);
        throw new StorageException("Failed to generate file URL", e);
    }
}
}

```

Przedstawiony adapter izoluje szczegóły SDK MinIO od pozostałych warstw systemu. Warstwa aplikacyjna może zapisywać obiekty przez metodę `putObject`, natomiast dostęp do plików jest realizowany przez czasowe adresy generowane w `getPresignedUrl`. Takie rozdzielanie odpowiedzialności ogranicza zależność logiki biznesowej od konkretnego klienta magazynu obiektowego.

Bezpieczeństwo przesyłania plików wymaga weryfikacji nie tylko rozmiaru i deklarowanego nagłówka `Content-Type`, lecz także rzeczywistej zawartości pliku. W tym celu wykorzystano bibliotekę Apache Tika, która wykrywa typ MIME na podstawie strumienia danych. Listing 18 przedstawia defensywną walidację pliku przed jego zapisaniem.

Listing 18. Defensywna walidacja typu i rozmiaru pliku.

```

@Component
public class FileValidator {
    private static final Tika TIKA = new Tika();
    public void validate(MultipartFile file, List allowedTypes, long maxSize,
String fileDescription) {
        if (file == null || file.isEmpty()) {
            throw new StorageException("File is required");
        }
        if (file.getSize() > maxSize) {
            throw new StorageException(
                String.format("File size exceeds maximum allowed size of %d MB for %s",
                    maxSize / (1024 * 1024), fileDescription));
        }
    }
}

```

```

    }
    String declaredType = file.getContentType();
    if (declaredType == null || !allowedTypes.contains(declaredType)) {
        throw new StorageException(
            String.format("Invalid file type '%s' for %s. Allowed types: %s",
                declaredType, fileDescription, String.join(", ", allowedTypes)));
    }
    String detectedType;
    try (InputStream is = file.getInputStream()) {
        detectedType = TIKA.detect(is, file.getOriginalFilename());
    } catch (IOException e) {
        throw new StorageException("Failed to read file for content validation",
            e);
    }
    if (!allowedTypes.contains(detectedType)) {
        log.warn("Rejected upload: declared {} but content matches {}",
            declaredType, detectedType);
        throw new StorageException("File content does not match a permitted type
            for " + fileDescription);
    }
}
}

```

Kod potwierdza, że kontrola pliku została wykonana wieloetapowo. Najpierw sprawdzana jest obecność pliku, jego rozmiar oraz zadeklarowany typ MIME, a następnie typ wykryty na podstawie zawartości. Dzięki temu ograniczono możliwość przesłania pliku, którego nagłówek wskazuje dozwolony format, ale rzeczywista zawartość nie spełnia wymagań.

Dla filmów ćwiczeń zastosowano osobny przepływ przetwarzania. Załadowanie pliku kończy się zapisaniem obiektu i ustawieniem statusu PROCESSING, natomiast generowanie miniatury jest wykonywane asynchronicznie. Listing 19 pokazuje przyjęcie filmu ćwiczenia oraz uruchomienie procesu tworzenia miniatury.

Listing 19. Załadowanie filmu i uruchomienie asynchronicznej miniatury.

```

public ExerciseMediaResponse uploadExerciseVideo(String userEmail, Long
exerciseId,
                                                MultipartFile videoFile, boolean
isAdmin) {
    Exercise exercise = exerciseRepository.findById(exerciseId)
        .orElseThrow(() -> new ResourceNotFoundException("Exercise not
found"));
    if (!isAdmin) {
        validateTrainerAccess(userEmail, exercise);
    }
    try {
        byte[] videoBytes = videoFile.getBytes();
        String contentType = videoFile.getContentType();
        String originalFilename = videoFile.getOriginalFilename();
        validateVideoFile(videoBytes, contentType);
        VideoProcessingService.VideoMetadata metadata =
            videoProcessingService.getVideoMetadataFromBytes(videoBytes,
originalFilename);
        int maxDuration =
storageProperties.getExerciseMedia().getMaxDurationSeconds();
        if (metadata.durationSeconds() > maxDuration) {
            throw new StorageException("Video duration exceeds maximum allowed
duration");
        }
    }
}

```

```

    }
    StoredFile videoStoredFile =
fileStorageService.uploadExerciseVideoFromBytes(
        videoBytes, contentType, originalFilename, exerciseId);
    storedFileWriter.markProcessingStatus(videoStoredFile.getId(),
ProcessingStatus.PROCESSING);
    asyncThumbnailProcessor.processThumbnail(videoStoredFile.getId(),
exerciseId, originalFilename);
    return ExerciseMediaResponse.processing(
        exerciseId,
        fileStorageService.getPresignedUrl(videoStoredFile),
        videoStoredFile.getId());
} catch (IOException e) {
    throw new StorageException("Failed to process video file", e);
}
}
}

```

Najważniejszym elementem przedstawionego fragmentu jest rozdzielenie operacji synchronicznych od kosztownego przetwarzania multimediów. Przed zapisaniem pliku weryfikowany jest dostęp do ćwiczenia, poprawność filmu oraz maksymalny czas trwania. Po zapisaniu obiektu status metadanych zostaje ustawiony na PROCESSING, a generowanie miniatury jest przekazywane do asynchronicznego procesora. Dzięki temu żądanie HTTP nie jest blokowane przez operacje związane z analizą i przetwarzaniem filmu.

5.8. Panel administratora

Panel administratora został zabezpieczony na poziomie serwera przez rolę ADMIN. Dostęp do zasobów administracyjnych nie wynika wyłącznie z odrębnej ścieżki API, lecz także z adnotacji autoryzacyjnej wymuszającej odpowiednie uprawnienie. Listing 20 prezentuje kontroler udostępniający operacje administracyjne dotyczące trenerów.

Listing 20. Chronione *endpointy* panelu administratora dla trenerów.

```

@RestController
@RequestMapping("/api/v1/admin/trainers")
@PreAuthorize("hasRole('ADMIN')")
@Tag(name = "Admin - Trainers", description = "Zarządzanie trenerami")
public class AdminTrainerController {
    private final AdminTrainerService adminTrainerService;

    @GetMapping
    @Operation(summary = "Pobierz listę wszystkich trenerów")
    public ResponseEntity<PagedResponse<AdminTrainerListResponse>>
getAllTrainers(
        @RequestParam(required = false) String search,
        @PageableDefault(size = 10, sort = "id", direction =
Sort.Direction.DESC) Pageable pageable
    ) {
        return
ResponseEntity.ok(adminTrainerService.getAllTrainers(search, pageable));
    }

    @GetMapping("/{id}")
    @Operation(summary = "Pobierz szczegóły trenera")
    public ResponseEntity<AdminTrainerDetailResponse>
getTrainerById(@PathVariable Long id) {

```

```

        return ResponseEntity.ok(adminTrainerService.getTrainerById(id));
    }

    @GetMapping("/count")
    @Operation(summary = "Pobierz liczbę trenerów")
    public ResponseEntity<Long> countTrainers() {
        return ResponseEntity.ok(adminTrainerService.countTrainers());
    }
}

```

Fragment potwierdza wydzielenie administracyjnej części API przez prefiks `/api/v1/admin/trainers` oraz zabezpieczenie całego kontrolera adnotacją `@PreAuthorize("hasRole('ADMIN')")`. Oznacza to, że wszystkie metody kontrolera wymagają roli administratora niezależnie od konkretnego *endpointu*. Kontroler udostępnia operacje pobrania listy trenerów, szczegółów pojedynczego trenera oraz liczby trenerów, natomiast szczegółowa logika agregacji danych została przeniesiona do warstwy serwisowej.

Lista trenerów w panelu administratora wymaga pobrania danych podstawowych oraz liczników opisujących aktywność powiązaną z trenerami. Zamiast wykonywać osobne zapytania dla każdego rekordu, zastosowano ładowanie agregatów dla identyfikatorów znajdujących się na aktualnej stronie wyników. Listing 21 pokazuje sposób pobierania strony trenerów oraz danych zagregowanych.

Listing 21. Agregaty listy trenerów ładowane zapytaniami grupującymi.

```

@Transactional(readOnly = true)
public PagedResponse<AdminTrainerListResponse> getAllTrainers(String search,
    Pageable pageable) {

    Page<TrainerProfile> page = trainerProfileRepository.findAll(pageable);

    List<Long> trainerIds =
    page.getContent().stream().map(TrainerProfile::getId).toList();

    TrainerListAggregates aggregates = loadAggregates(trainerIds);

    List<AdminTrainerListResponse> content = page.getContent().stream()
        .map(t -> mapToListResponse(t, aggregates))
        .toList();

    return new PagedResponse<>(
        content,
        page.getNumber(),
        page.getSize(),
        page.getTotalElements(),
        page.getTotalPages(),
        page.isFirst(),
        page.isLast()
    );
}

private TrainerListAggregates loadAggregates(List<Long> trainerIds) {
    if (trainerIds.isEmpty()) {
        return TrainerListAggregates.empty();
    }

    return new TrainerListAggregates(
        toCountMap(assignmentRepository.countActiveClientsByTrainerIds(trainerIds)),
        groupServiceTypeCounts(assignmentRepository.countClientsByServiceTypeForTrainerIds(trainerIds)),
    );
}

```

```
toCountMap(trainingTemplateRepository.countByTrainerIds(trainerIds)),  
toCountMap(dietTemplateRepository.countByTrainerIds(trainerIds)),  
toCountMap(exerciseRepository.countByTrainerIds(trainerIds))  
);  
}
```

Kod pokazuje, że lista administracyjna nie została zaimplementowana jako prosty odczyt encji. Najpierw pobierana jest strona profili trenerów, a następnie na podstawie ich identyfikatorów ładowane są agregaty z repozytoriów powiązanych z klientami, szablonami treningów, szablonami diet oraz ćwiczeniami. Takie podejście ogranicza liczbę zapytań wykonywanych dla pojedynczych rekordów i pozwala przygotować odpowiedź stronicowaną zawierającą dane potrzebne do administracyjnej analizy trenerów.

Podsumowując, implementacja systemu została zorganizowana wokół separacji warstw, centralizacji mechanizmów bezpieczeństwa oraz umieszczania reguł domenowych w serwisach. Moduły funkcjonalne korzystają ze wspólnych mechanizmów kontekstu użytkownika, autoryzacji, komunikacji HTTP, komunikacji STOMP, trwałego zapisu danych i obsługi plików. Taki układ stanowi podstawę do dalszej walidacji działania systemu w rozdziale poświęconym testom.

6. Interfejs użytkownika

W niniejszym rozdziale przedstawiono warstwę interfejsu użytkownika systemu Coachify. Opis obejmuje organizację nawigacji, układ widoków zależny od roli użytkownika, funkcje dostępne dla trenera, podopiecznego i administratora, a także publiczną część *marketplace'u* oraz globalny komponent czatu. Rozdział kończy omówienie responsywności i dostępności, z uwzględnieniem potwierdzonych mechanizmów zaimplementowanych w aplikacji klienckiej.

Interfejs użytkownika został przygotowany jako aplikacja typu *Single-Page Application*. Podstawowa nawigacja odbywa się po stronie przeglądarki, a komunikacja z serwerem realizowana jest przez REST API oraz w przypadku czatu, przez WebSocket-STOMP. Warstwa widoków jest powiązana z rolami systemowymi: TRAINER, CLIENT i ADMIN. Użytkownik niezalogowany otrzymuje dostęp do widoków publicznych, w szczególności formularzy rejestracji i logowania oraz publicznego *marketplace'u* trenerów.

6.1. Ogólna nawigacja i layout aplikacji

Nawigacja aplikacji została oparta na dwóch niezależnych układach widoków. Pierwszy układ obsługuje część publiczną oraz panele trenera i podopiecznego. Składa się z górnego paska nawigacji, obszaru treści oraz globalnego widżetu czatu. Drugi układ został przygotowany dla administratora i wykorzystuje boczny pasek nawigacji, co odpowiada narzędziowemu charakterowi panelu administracyjnego.

Routing aplikacji wykorzystuje polskie nazwy ścieżek, między innymi `/trener/pulpit`, `/klient/plany`, `/rynek/trener/:id`, `/admin/pulpit`, `/admin/cwiczenia`. Takie nazewnictwo zachowuje spójność z językiem interfejsu i zakresem aplikacji przeznaczonej dla polskojęzycznych użytkowników. Dostęp do tras chronionych jest kontrolowany przez komponenty ochrony tras. Ochrona roli sprawdza stan uwierzytelnienia oraz dopasowanie roli użytkownika do wymaganej roli widoku. W przypadku braku dostępu użytkownik jest kierowany do właściwego panelu albo do ekranu logowania. Osobny mechanizm obsługuje widoki publiczne dostępne wyłącznie dla użytkowników niezalogowanych, takie jak logowanie i rejestracja.

Górny pasek nawigacji buduje menu dynamicznie na podstawie roli zalogowanego użytkownika. Dla trenera dostępne są skróty do panelu, klientów, szablonów dietetycznych i ustawień. Dla podopiecznego dostępne są skróty do panelu, planów, dziennika, diet, rynku oraz ustawień. Administrator korzysta z odrębnego układu, dlatego nie jest prowadzony przez standardowy pasek górny. Aktywna trasa jest wyróżniana na podstawie aktualnej ścieżki, co wskazuje aktualnie wybraną sekcję aplikacji.

Tabela 22 przedstawia podział głównych obszarów interfejsu według ról systemowych oraz wskazuje typ układu stosowany w każdym panelu aplikacji.

Tabela 22. Role. Źródło: opracowanie własne.

Rola	Główne obszary interfejsu	Charakter układu
Użytkownik niezalogowany	Strona główna, logowanie, rejestracja, publiczny <i>marketplace</i>	Układ publiczny z górną nawigacją
Podopieczny	Pulpit, plany treningowe, diety, dziennik treningowy, <i>marketplace</i> , ustawienia, czat	Układ użytkownika z górną nawigacją i globalnym widżetem czatu

Trener	Pulpit, klienci, plany i szablony, diety i szablony posiłków, ćwiczenia, raporty, ustawienia, czat	Układ użytkownika z górną nawigacją i globalnym widżetem czatu
Administrator	Pulpit, ćwiczenia globalne, trenerzy, klienci	Dedykowany układ z bocznym panelem nawigacji

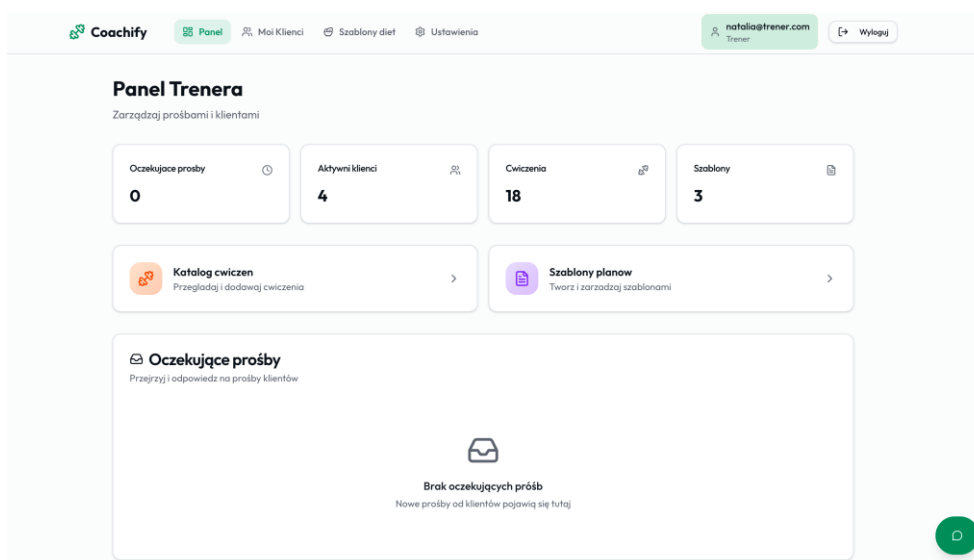
Podział na dwa układy ogranicza liczbę warunkowo renderowanych elementów w warstwie prezentacji. Panel administratora nie musi ukrywać elementów przeznaczonych dla trenera lub podopiecznego, a część użytkownika końcowego wykorzystuje odrębny zestaw elementów nawigacyjnych.

6.2. Widoki trenera

Panel trenera jest częścią interfejsu obejmującą największą liczbę widoków. Obejmuje widoki obsługujące przyjmowanie zgłoszeń od klientów, zarządzanie podopiecznymi, tworzenie planów treningowych i dietetycznych, pracę na szablonach oraz analizę raportów treningowych.

6.2.1 Pulpit trenera i obsługa podopiecznych

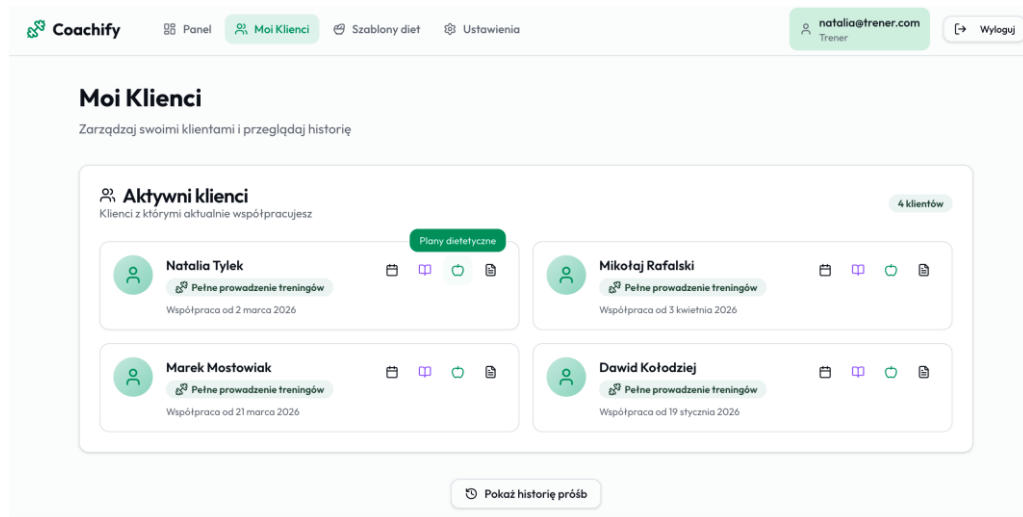
Centralnym widokiem jest pulpit trenera. Prezentuje on oczekujące prośby od klientów, listę aktywnych podopiecznych oraz liczniki zasobów istotnych dla pracy trenera, takich jak szablony i ćwiczenia. Trener może zaakceptować albo odrzucić zgłoszenie klienta, a po wykonaniu akcji dane widoku są odświeżane przez mechanizm unieważniania zapytań. Takie rozwiązanie ogranicza ryzyko prezentowania nieaktualnych informacji po zmianie stanu współpracy.



Rysunek 13. Panel trenera. Źródło: opracowanie własne.

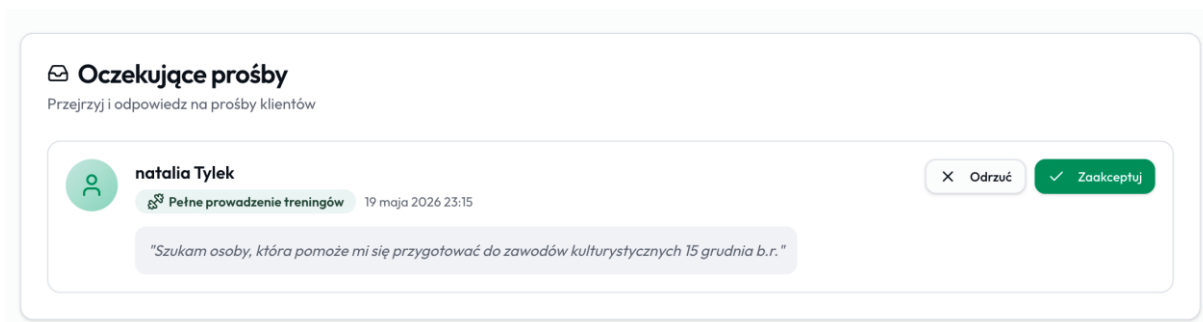
Z poziomu panelu trener może przejść do katalogu ćwiczeń oraz modułu zarządzania szablonami planów (rysunek 13).

Widok listy podopiecznych ma charakter operacyjny. Każdy podopieczny jest prezentowany w postaci karty z informacją o aktywnych przypisaniach i skrótami do działań zależnych od typu usługi. Dla usługi planu treningowego dostępne są skróty do planów i dziennika, dla planu dietetycznego skróty do planów dietetycznych, a dla pełnego coachingu dostępne są funkcje obu obszarów. Jeżeli klient wypełnił kwestionariusz, trener może przejść do jego podglądu w trybie tylko do odczytu. Historia próśb została ukryta pod przełącznikiem i jest stronicowana, co ogranicza liczbę danych archiwalnych prezentowanych jednocześnie.



Rysunek 14. Widok klientów trenera. Źródło: opracowanie własne.

Na rysunku 14 przedstawiono listę aktywnych klientów trenera. Dla każdego klienta dostępne są skróty prowadzące do najważniejszych modułów współpracy, w tym planów treningowych, planów dietetycznych (w zależności od wykupionej usługi).



Rysunek 15. Zarządzanie próśbami o współpracę. Źródło: opracowanie własne.

Na rysunku 15 przedstawiono moduł obsługi oczekujących próśb o współpracę dostępny dla trenera. System wyświetla informacje o kliencie, wybranej usłudze, czasie wysłania zgłoszenia oraz treści wiadomości dołączonej przez użytkownika.

Trener może zaakceptować lub odrzucić zgłoszenie. Akceptacja próśby powoduje utworzenie aktywnej relacji trener-klient, dzięki czemu możliwe staje się korzystanie z pozostałych modułów systemu, takich jak przypisywanie planów treningowych, komunikacja przez chat czy monitorowanie postępów klienta.

6.2.2 Profil, usługi i ustawienia trenera

W przypadku trenera nie występuje żaden wstępny formularz rejestracyjny. Trener uzupełnia dane o sobie w zakładce „ustawienia”.

The screenshot shows the 'Ustawienia' (Settings) page for a trainer. The main heading is 'Ustawienia' with a subtitle 'Zarządzaj swoim profilem i ofertą usług'. Below this is the 'Profil' section, which includes a circular profile picture placeholder, a text input for 'Imię' (Name) with the value 'Jan', and another for 'Nazwisko' (Surname) with the value 'Kowalski'. A note states 'Te informacje będą widoczne dla potencjalnych klientów'. Below the name fields is a section for 'O mnie' (About me) with a text area for 'Opisz swoje doświadczenie, specjalizacje i podejście do treningów...' and a character limit of 'Maksymalnie 1000 znaków'. At the bottom of the profile section is a toggle for 'Widoczność w marketplace' (Visibility in marketplace), which is currently turned on, with the text 'Twój profil jest widoczny dla klientów'. A green button labeled 'Zapisz profil' (Save profile) is at the bottom.

Rysunek 16. Ustawienia trenera cz. 1. Źródło: opracowanie własne.

Widok ustawień profilu trenera, przedstawiony na rysunku 16 umożliwia zarządzanie informacjami widocznymi dla potencjalnych klientów w *marketplace*. Trener może uzupełnić dane osobowe, dodać zdjęcie profilowe, opisać swoje doświadczenie oraz określić widoczność swojego profilu w systemie.

The screenshot shows the 'Oferowane usługi' (Offered services) section of the 'Ustawienia' (Settings) page. It includes a subtitle 'Wybierz usługi które oferujesz i ustaw ich ceny. Musisz mieć co najmniej jedną aktywną usługę, aby być widocznym w marketplace.' Below this are three toggle switches for 'Plan Treningowy', 'Pełne prowadzenie treningów', and 'Dieta', all of which are currently turned off. A yellow warning banner states 'Włącz co najmniej jedną usługę, aby być widocznym w marketplace'. A green button labeled 'Zapisz usługi' (Save services) is below the toggles. Below this is a section for 'Zablokowani klienci' (Blocked clients) with a subtitle 'Zarządzaj listą zablokowanych klientów, którzy nie mogą wysyłać Ci próśb o wiadomość'. It features a button labeled 'Zarządzaj zablokowanymi' with a right-pointing arrow.

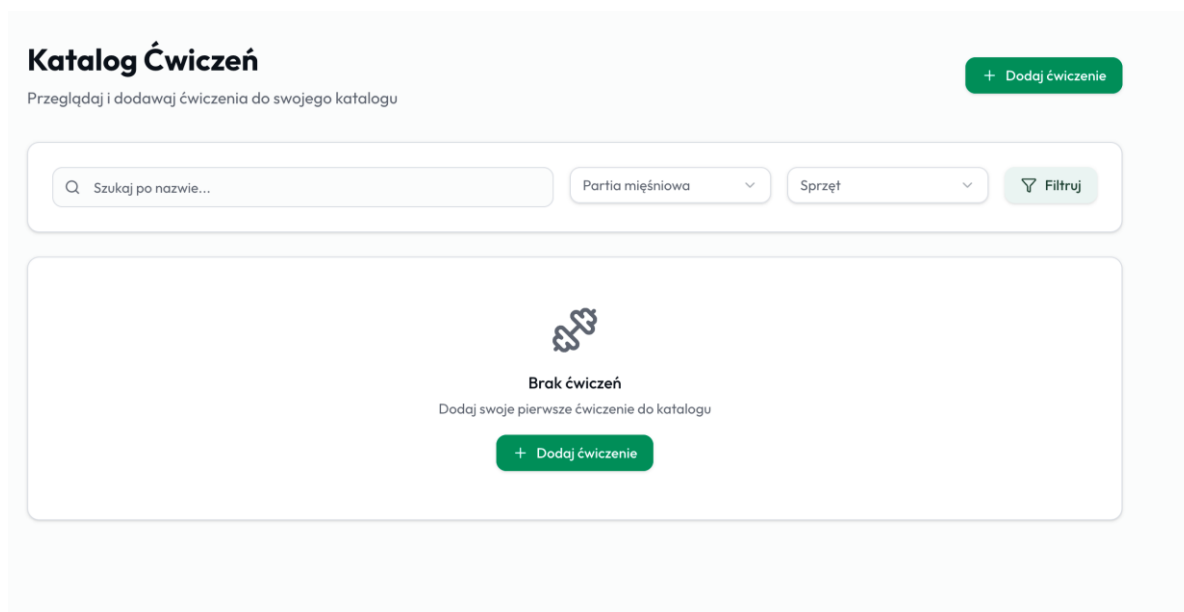
Rysunek 17. Ustawienia trenera cz. 2. Źródło: opracowanie własne.

W zakładce ustawienia znajduje się także sekcja zarządzania usługami oraz zablokowanymi klientami, przedstawiona na rysunku 17, pozwala trenerowi określić rodzaje oferowanych usług, takich

jak plan treningowy, pełne prowadzenie treningów czy dieta. Trener może aktywować wybrane usługi oraz zarządzać listą zablokowanych klientów, którzy nie mają możliwości wysyłania wiadomości. System wymaga aktywowania przynajmniej jednej usługi, aby profil był widoczny w *marketplace*.

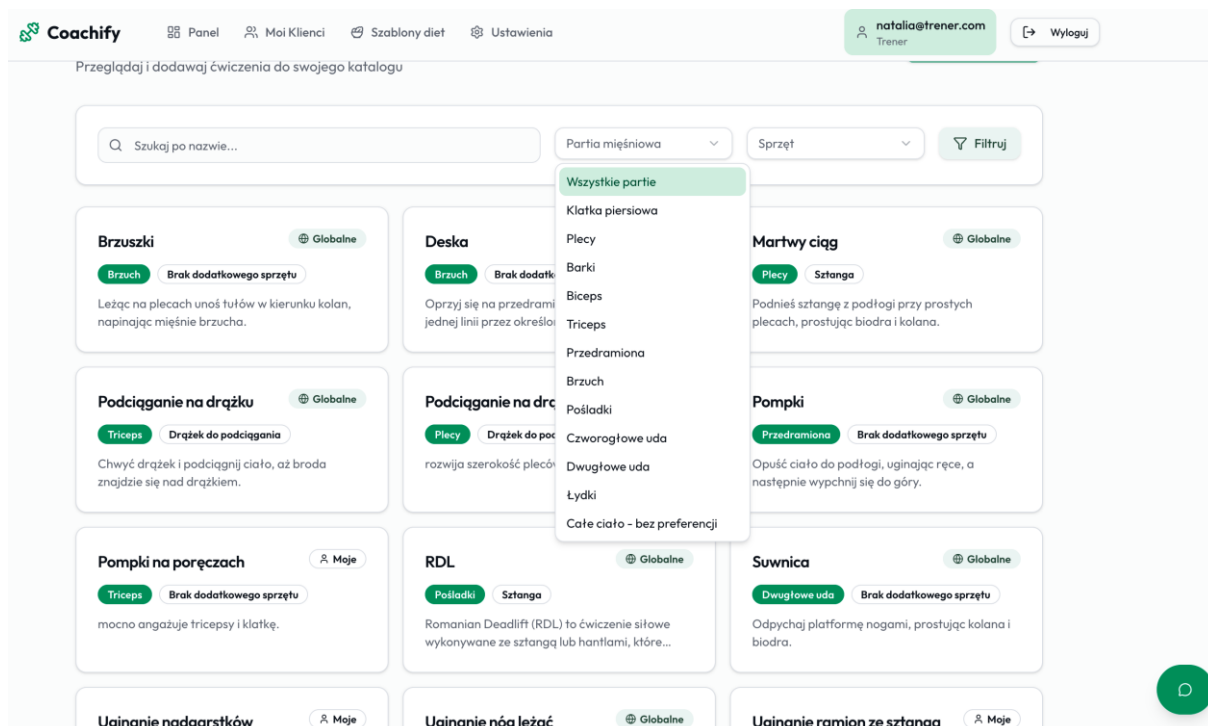
6.2.3 Katalog ćwiczeń trenera

Widok katalogu ćwiczeń przedstawiony na rysunku 18 umożliwia przeglądanie oraz dodawanie ćwiczeń do prywatnej biblioteki trenera. Interfejs zawiera wyszukiwarkę, filtry partii mięśniowych oraz sprzętu, a także przycisk dodawania nowego ćwiczenia. W przypadku braku dostępnych ćwiczeń system wyświetla komunikat informujący o pustym katalogu.

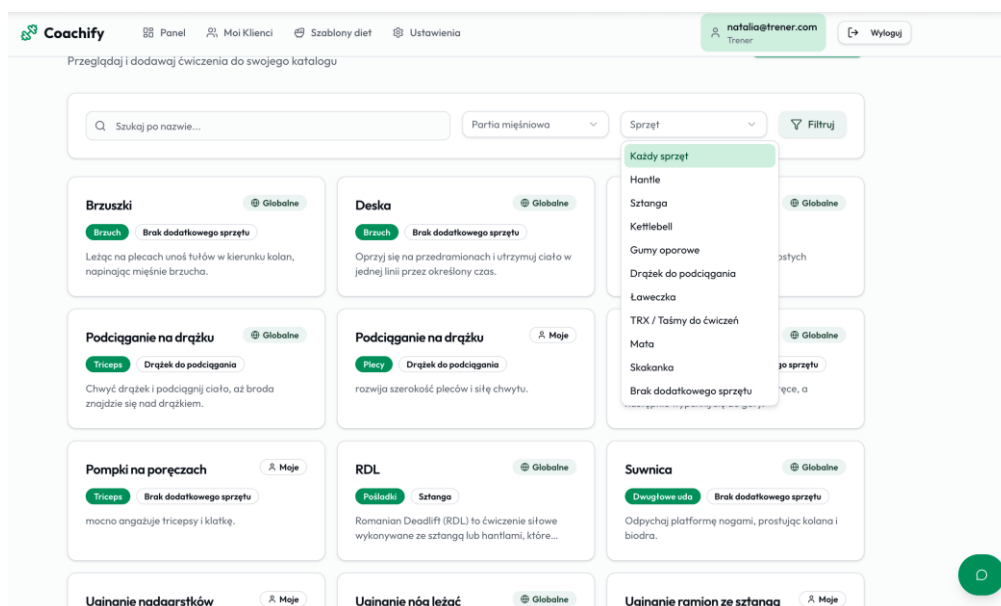


Rysunek 18. Katalog ćwiczeń - pusty stan. Źródło: opracowanie własne.

Na rysunku 19 przedstawiono mechanizm filtrowania ćwiczeń według wybranej partii mięśniowej. Po rozwinięciu listy trener może wybrać konkretną grupę mięśniową, co pozwala ograniczyć widok katalogu wyłącznie do ćwiczeń spełniających określone kryterium.



Rysunek 19. Filtrowanie ćwiczeń według partii mięśniowej. Źródło: opracowanie własne.



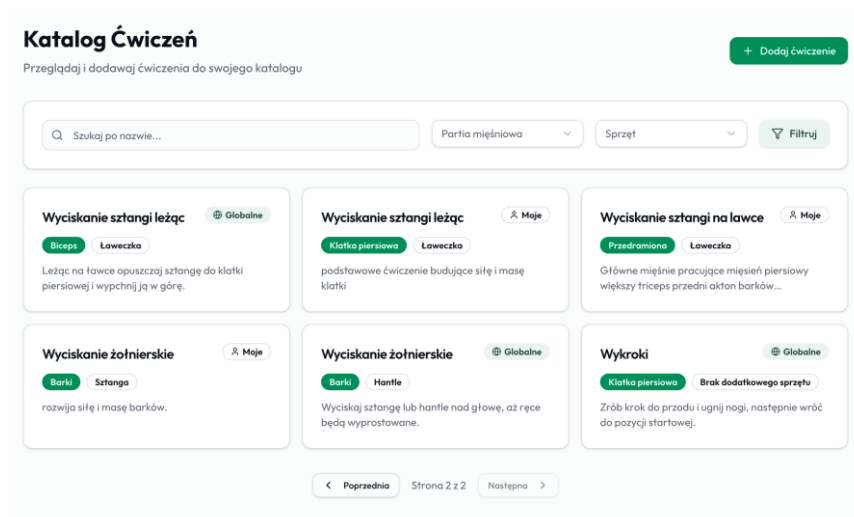
Rysunek 20. Filtrowanie ćwiczeń według sprzętu. Źródło: opracowanie własne.

Widok przedstawiony na rysunku 20 prezentuje możliwość filtrowania ćwiczeń według wymaganego sprzętu treningowego. Trener może wybrać jeden z dostępnych typów wyposażenia, takich jak hantle, sztanga czy gumy oporowe.

Rysunek 21. Wypełniony formularz dodawania ćwiczenia. Źródło: opracowanie własne.

Widok przedstawiony na rysunku 21 prezentuje przykładowo uzupełniony formularz dodawania nowego ćwiczenia. Trener wprowadza szczegółowy opis techniki wykonania ćwiczenia oraz przypisuje odpowiednią partię mięśniową i wymagany sprzęt. Po zatwierdzeniu formularza ćwiczenie zostaje zapisane w prywatnym katalogu trenera. Możliwe jest dodanie filmu instruktażowego.

Rysunek 22. Katalog ćwiczeń po dodaniu pozycji cz. 1. Źródło: opracowanie własne.



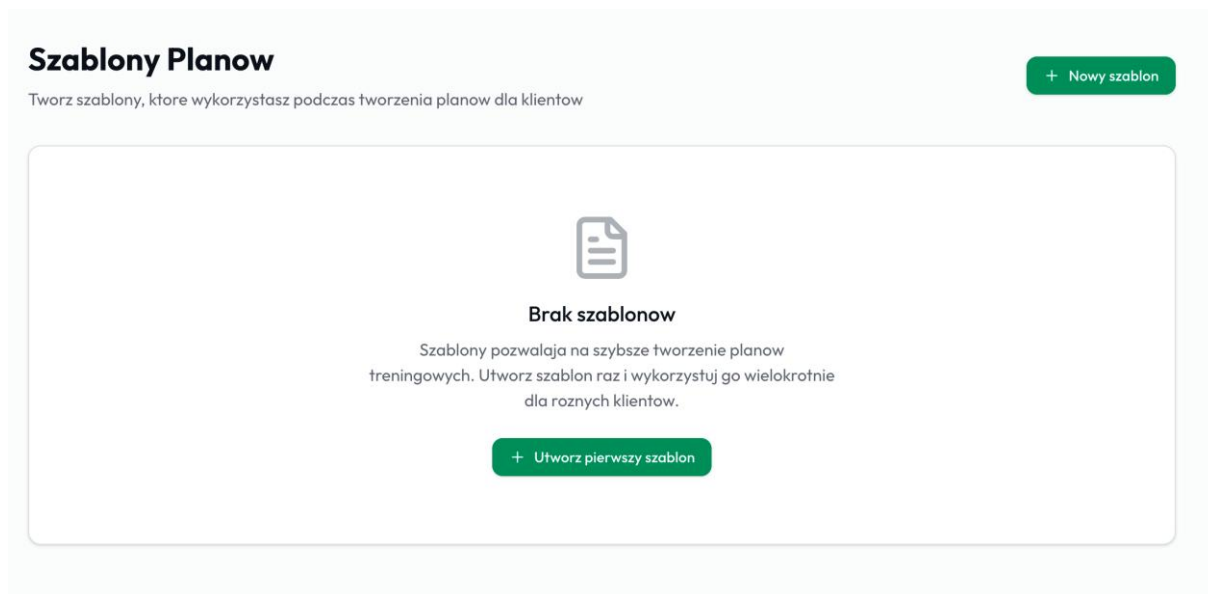
Rysunek 23. Katalog ćwiczeń po dodaniu pozycji cz. 2. Źródło: opracowanie własne.

Na rysunku 22 i 23 przedstawiono widok katalogu ćwiczeń po dodaniu nowych ćwiczeń do systemu. Dodane ćwiczenie jest wyświetlane w postaci karty zawierającej nazwę, przypisaną partię mięśniową, wymagany sprzęt oraz skrócony opis techniki wykonania. Ćwiczenia oznaczone jako „Moje” należą do prywatnej biblioteki trenera i mogą być wykorzystywane podczas tworzenia planów treningowych oraz szablonów. Dzięki temu trener nie musi każdorazowo definiować ćwiczeń od początku, a proces tworzenia planów dla klientów jest znacznie szybszy i bardziej uporządkowany.

6.2.4 Szablony planów treningowych

Moduł planów treningowych umożliwia trenerowi tworzenie planów dla konkretnych klientów oraz przygotowywanie szablonów wielokrotnego użytku. Plan treningowy jest zorganizowany jako struktura tygodniowa, w której dni zawierają ćwiczenia wraz z parametrami wykonania. Ćwiczenia mogą pochodzić z biblioteki globalnej zarządzanej przez administratora albo z prywatnej biblioteki trenera.

Szablony planów treningowych są niezależne od przypisań do klientów. Pozwala to trenerowi przygotować powtarzalne schematy, a następnie użyć ich jako podstawy do planu konkretnego podopiecznego. Klient ma dostęp wyłącznie do planów przypisanych do jego konta i nie może modyfikować ich struktury.



Rysunek 24. Widok modułu szablonów planów treningowy. Źródło: opracowanie własne.

Na rysunku 24 pokazano moduł zarządzania szablonami planów treningowych. Widok umożliwia tworzenie wielokrotnego użytku schematów treningowych, które mogą być później wykorzystane podczas przygotowywania planów dla klientów. W przypadku braku zapisanych szablonów system wyświetla komunikat informujący o pustej liście oraz przycisk umożliwiający utworzenie pierwszego szablonu.

Rysunek 25. Formularz tworzenia nowego szablonu treningowego. Źródło: opracowanie własne.

Widok przedstawiony na rysunku 25 prezentuje formularz tworzenia nowego szablonu planu treningowego. Trener może określić nazwę szablonu oraz opcjonalny opis zawierający założenia treningowe. Poniżej znajduje się tygodniowy harmonogram składający się z siedmiu dni, które mogą zostać oznaczone jako dni treningowe lub dni odpoczynku. Domyślnie dzień jest ustawiony jako dzień odpoczynku i nie zawiera zaplanowanych ćwiczeń.

Rysunek 26. Uzupełnianie informacji o szablonie. Źródło: opracowanie własne.

Na rysunku 26 przedstawiono formularz po wprowadzeniu podstawowych danych szablonu. Trener definiuje nazwę planu oraz szczegółowy opis zawierający informacje dotyczące zasad treningowych.

Rysunek 27. Wybór ćwiczenia do dnia treningowego. Źródło: opracowanie własne.

Widok przedstawiony na rysunku 27 prezentuje okno wyboru ćwiczeń przypisywanych do konkretnego dnia treningowego. System umożliwia wyszukiwanie ćwiczeń po nazwie oraz dodawanie

ich do planu za pomocą dedykowanego przycisku. Lista zawiera zarówno ćwiczenia globalne dostępne w systemie, zarządzane przez administratora, jak i ćwiczenia prywatne dodane wcześniej przez trenera do własnego katalogu.

Plan tygodniowy

Dzien 1 **Dzien 2** Dzien 3 Dzien 4 Dzien 5 Dzien 6 Dzien 7

Trening

4 cwiczen

+ Dodaj cwiczenie

1

Wyciskanie sztangi na ławce

Serie	Powtorzenia	Przerwa (s)	Uwagi
5	10	60	...

2

Podciąganie na drążku

Serie	Powtorzenia	Przerwa (s)	Uwagi
2	15	60	...

3

Pompki na poręczach

Serie	Powtorzenia	Przerwa (s)	Uwagi
4	30	120	do pełnego zgięcia

4

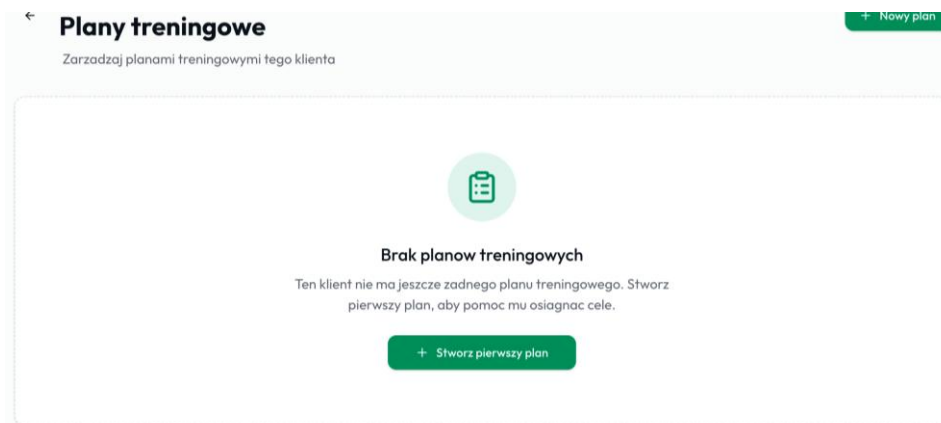
Uginanie ramion ze sztangą

Serie	Powtorzenia	Przerwa (s)	Uwagi
3	8	60	progresujące obciąż

Rysunek 28. Gotowy szablon treningowy z przypisanym ćwiczeniem. Źródło: opracowanie własne.

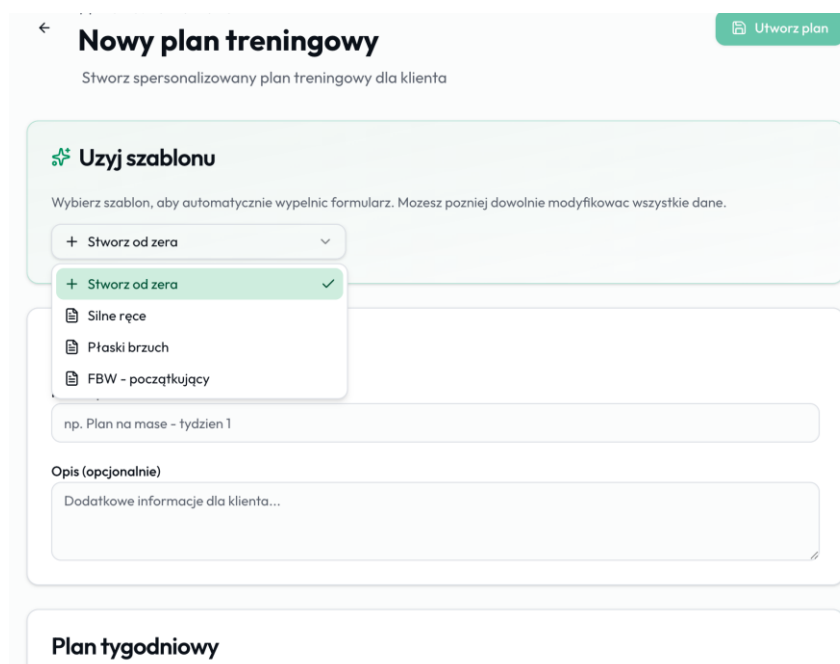
Na rysunku 28 zaprezentowano przykładowo skonfigurowany szablon treningowy zawierający przypisane ćwiczenie wraz z parametrami wykonania. Dla każdego ćwiczenia trener może określić liczbę serii, powtórzeń, czas przerwy oraz dodatkowe uwagi. W harmonogramie tygodniowym dni treningowe oznaczone są zielonymi kropkami znajdującymi się nad numerami dni, co pozwala szybko odróżnić dni aktywne od dni przeznaczonych na regenerację.

6.2.5 Plany treningowe przypisane klientom



Rysunek 29. Widok modułu planów treningowych klienta. Źródło: opracowanie własne.

Na rysunku 29 przedstawiono moduł planów treningowych przypisanych do konkretnego klienta, z którym trener nawiązał współpracę. Widok umożliwia zarządzanie planami treningowymi podopiecznego oraz tworzenie nowych planów dostosowanych do jego celów i możliwości. W przypadku braku przypisanych planów system wyświetla komunikat informujący o pustej liście oraz przycisk pozwalający na utworzenie pierwszego planu treningowego.



Rysunek 30. Tworzenie nowego planu treningowego z wykorzystaniem szablonu. Źródło: opracowanie własne.

Na rysunku 30 zaprezentowano formularz tworzenia nowego planu treningowego dla klienta. System umożliwia rozpoczęcie pracy od pustego formularza lub wykorzystanie wcześniej przygotowanego szablonu treningowego. Dzięki temu trener może znacznie przyspieszyć proces przygotowywania planów treningowych oraz zachować spójność stosowanych schematów treningowych.

Informacje podstawowe

Nazwa planu *

Silne ręce

Opis (opcjonalnie)

trening wzmacniający barki, przedramiona i klatkę.

Plan tygodniowy

Dzień 1
Dzień 2
Dzień 3
Dzień 4
Dzień 5
Dzień 6
Dzień 7

Trening
3 ćwiczenia
Dodaj ćwiczenie

1
Podciąganie na drążku

Serie: 5
Powtórzenia: 3
Przerwa (s): 26
Uwagi: ...

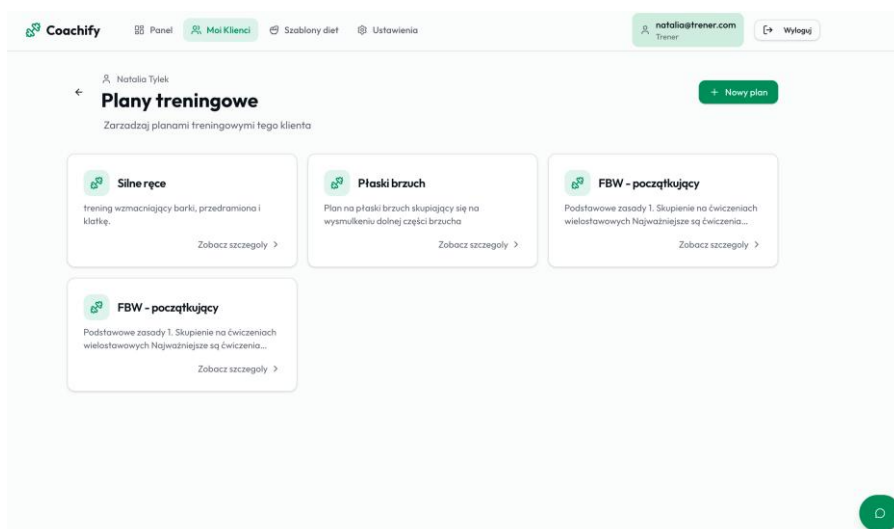
2
Uginanie nadgarstków

Serie: 3
Powtórzenia: 10
Przerwa (s): 60
Uwagi: ...

Rysunek 31. Zmodyfikowany plan treningowy utworzony na podstawie szablonu. Źródło: opracowanie własne.

Widok przedstawiony na rysunku 31 prezentuje gotowy plan treningowy utworzony na podstawie wcześniej przygotowanego szablonu „Silne ręce”. Po załadowaniu szablonu trener może dowolnie modyfikować jego zawartość poprzez zmianę parametrów ćwiczeń, usuwanie istniejących elementów lub dodawanie nowych ćwiczeń dopasowanych do potrzeb konkretnego klienta.

Na przedstawionym przykładzie trener wykorzystał wcześniej przygotowany szablon, a następnie rozszerzył go o dodatkowe ćwiczenia, dostosowując plan do indywidualnych potrzeb podopiecznego.



Rysunek 32. Lista planów treningowych przypisanych do klienta z perspektywy trenera. Źródło: opracowanie własne.

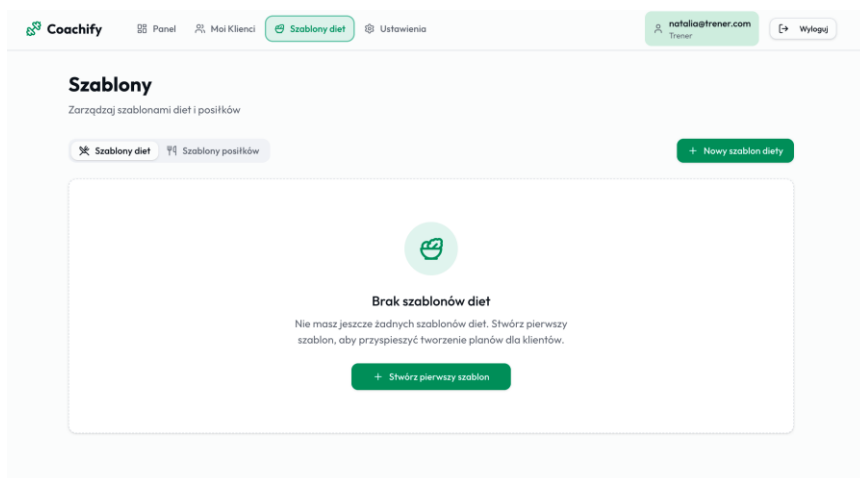
Na rysunku 32 przedstawiono widok planów treningowych przypisanych do konkretnego klienta z poziomu konta trenera. Trener ma dostęp do wszystkich planów utworzonych dla podopiecznego,

zarówno aktualnych, jak i wcześniejszych. Każdy plan prezentowany jest w postaci osobnej karty zawierającej nazwę planu, krótki opis oraz możliwość przejścia do szczegółów.

6.2.6 Szablony diet, szablony posiłków i plany dietetyczne klientów

Moduł dietetyczny odzwierciedla sposób pracy z planami treningowymi, ale dotyczy żywienia. Trener może przygotować plan dietetyczny dla klienta, podzielić go na dni, posiłki i produkty oraz tworzyć szablony planów dietetycznych. Dodatkowo przewidziano szablony posiłków, które pełnią funkcję mniejszych, wielokrotnie wykorzystywanych fragmentów planu.

System oddziela plan przypisany do klienta od szablonu trenera, dzięki czemu późniejsza edycja planu klienta nie narusza bazowego szablonu wykorzystywanego dla innych podopiecznych.



Rysunek 33. Widok modułu szablonów diet i posiłków. Źródło: opracowanie własne.

Na rysunku 33 zaprezentowano moduł zarządzania szablonami diet oraz posiłków dostępny dla trenera. W module dostępny jest podział na szablony diet oraz szablony pojedynczych posiłków.

Rysunek 34. Formularz tworzenia szablonu posiłku. Źródło: opracowanie własne.

Na rysunku 34 pokazano formularz tworzenia nowego szablonu posiłku. Trener może określić nazwę posiłku, opis, wartości odżywcze oraz instrukcję przygotowania.

Dla każdego posiłku możliwe jest przypisanie typu, np. śniadanie, obiad, kolacja, deser lub przekąska. Informacja ta jest później wykorzystywana podczas budowania planów dietetycznych, co ułatwia filtrowanie oraz dobieranie odpowiednich posiłków do konkretnej pory dnia.

Składniki + Dodaj składnik

1.	jajko	3	g	×
2.	mleko	200	ml	×
3.	sól	1	łyżka	×
4.	woda	2	szklanka	×

Utwórz szablon

Rysunek 35. Definiowanie składników posiłku. Źródło: opracowanie własne.

Na rysunku 35 przedstawiono sekcję odpowiedzialną za definiowanie składników szablonu posiłku. Trener może dodawać dowolną liczbę składników wraz z ilością oraz jednostką miary.

System umożliwia dynamiczne zarządzanie listą składników, w tym dodawanie nowych pozycji oraz usuwanie istniejących.

Szablony
Zarządzaj szablunami diet i posiłków

Szablony diet Szablony posiłków + Nowy szablon posiłku

Sernik na zimno Deser na bazie wysokobiałkowego sera 213 kcal B: 45g W: 82g T: 39g 3 składniki Edytuj >	Owsianka Kolacja pieczona w piekarniku 560 kcal B: 34g W: 65g T: 87g 3 składniki Edytuj >	Kurczak w sosie serowym Obiad Gotowany na parze 345 kcal B: 23g W: 24g T: 12g 3 składniki Edytuj >
Chleb w jajku Drugie śniadanie wersja wysokobiałkowa 200 kcal B: 20g W: 45g T: 63g 4 składniki Edytuj >	Omlet z warzywami Śniadanie wersja bez laktozy 276 kcal B: 16g W: 43g T: 24g 4 składniki Edytuj >	

Rysunek 36. Lista utworzonych szablonów posiłków. Źródło: opracowanie własne.

Na rysunku 36 zaprezentowano widok listy zapisanych szablonów posiłków. Każda karta zawiera podstawowe informacje o posiłku, takie jak nazwa, typ posiłku, liczba składników oraz wartości odżywcze.

Z poziomu listy trener może przejść do edycji wybranego szablonu. Utworzone wcześniej szablony posiłków mogą być następnie wykorzystywane podczas budowania planów dietetycznych, a po dodaniu do diety mogą być dalej dowolnie modyfikowane bez wpływu na oryginalny szablon.

Rysunek 37. Tworzenie szablonu planu dietetycznego. Źródło: opracowanie własne.

Na rysunku 37 przedstawiono formularz tworzenia nowego szablonu planu dietetycznego. Trener definiuje podstawowe informacje, takie jak nazwa planu, opis, a docelowe wartości kaloryczne i makroskładniki są obliczane automatycznie na podstawie posiłków.

Plan dietetyczny tworzony jest w układzie tygodniowym z podziałem na dni tygodnia. Dla każdego dnia możliwe jest dodawanie posiłków oraz notatek dla klienta. System wizualnie oznacza dni, które zostały już uzupełnione posiłkami.

Rysunek 38. Tworzenie planu dietetycznego dla klienta z wykorzystaniem szablonu. Źródło: opracowanie własne.

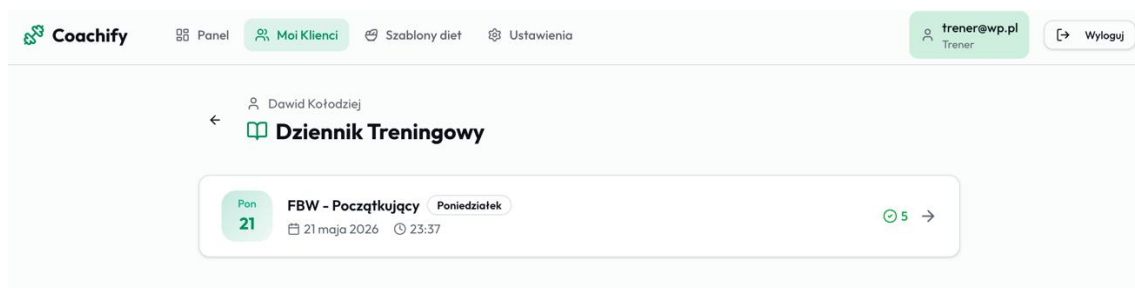
Na rysunku 38 zaprezentowano formularz tworzenia nowego planu dietetycznego przypisywanego klientowi. Podczas tworzenia planu trener może rozpocząć pracę od pustego formularza lub wykorzystać wcześniej przygotowany szablon diety.

Wybranie szablonu automatycznie uzupełnia plan gotowymi posiłkami, strukturą tygodnia oraz wartościami odżywczymi. Po załadowaniu szablonu trener może dowolnie modyfikować wszystkie elementy planu bez wpływu na oryginalny szablon zapisany w systemie.

6.2.7 Raporty treningowe klientów

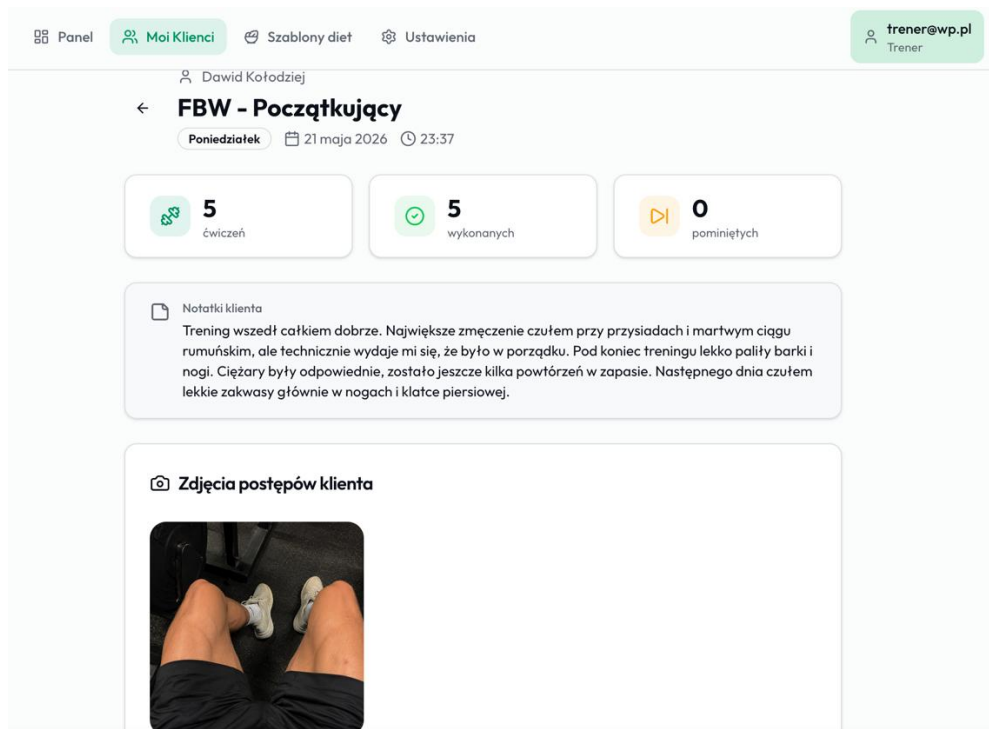
Panel trenera obejmuje również widoki raportów treningowych. Trener może przeglądać listę raportów klienta oraz szczegóły pojedynczego raportu, w tym informacje o wykonanych seriach, pominiętych ćwiczeniach i załączonych zdjęciach postępów, jeżeli zostały przesłane przez podopiecznego.

Listę zaraportowanych treningów wybranego podopiecznego przedstawiono na rysunku 39. Każdy wpis zawiera nazwę dnia planu, etykietę dnia tygodnia, datę i godzinę wykonania oraz liczbę zaraportowanych ćwiczeń.



Rysunek 39. Lista zaraportowanych treningów podopiecznego w dzienniku treningowym. Źródło: opracowanie własne.

Szczegóły pojedynczego raportu zaprezentowano na rysunku 40. W górnej części znajduje się podsumowanie liczby ćwiczeń zaplanowanych, wykonanych oraz pominiętych, a w dolnej notatki klienta opisujące przebieg treningu oraz sekcja zdjęć postępów, do której można dołączyć maksymalnie trzy zdjęcia.



Rysunek 40. Widok szczegółowy raportu treningowego - podsumowanie, notatki klienta i zdjęcia postępów.
Źródło: opracowanie własne.

Zestawienie wykonanych ćwiczeń ukazano na rysunku 41. Dla każdego ćwiczenia wyświetlana jest przypisana partia mięśniowa oraz tabela serii zawierająca obciążenie, liczbę powtórzeń i status wykonania, wraz z porównaniem realizacji z założeniami planu.

Wykonane ćwiczenia			
1	Przysiad goblet		Wykonane
	Czworogłowe uda		
	Seria	Obciążenie	Powtórzenia
	Status		
1	30 kg	12	✓
	2	30 kg	10
	3	30 kg	8
Plan: 3 serii x 8-12 powt. Wykonano: 3 serii			
2	Wyciskanie hantli na ławce		Wykonane
	Klatka piersiowa		
	Seria	Obciążenie	Powtórzenia
	Status		
1	30 kg	10	✓
	2	25 kg	8
	3	25 kg	8

Rysunek 41. Zestawienie wykonanych ćwiczeń wraz z seriami i porównaniem z planem. Źródło: opracowanie własne.

6.3. Widoki podopiecznego

Panel podopiecznego koncentruje się na rozpoczęciu współpracy z trenerem, dostępie do przypisanych planów oraz raportowaniu wykonanych treningów. Głównym punktem wejścia jest pulpit klienta, który agreguje status użytkownika, status *onboardingu*, najnowszy plan treningowy, najnowszy plan dietetyczny oraz raporty treningowe z trzymiesięcznego okna.

6.3.1 Onboarding i konfiguracja profilu klienta

Na rysunkach 42-56 przedstawiony został kwestionariusz podopiecznego. Kwestionariusz podopiecznego służy do zebrania informacji niezbędnych trenerowi do przygotowania planu. Formularz ma charakter wieloetapowego kreatora i obejmuje czternaście kroków merytorycznych, dotyczących m.in. celu treningowego, miejsca ćwiczeń, dostępnego sprzętu, doświadczenia, aktualnej i planowanej częstotliwości treningów, preferowanych grup mięśniowych, kontuzji, ćwiczeń preferowanych oraz ćwiczeń niechcianych.

System przechowuje stan wersji roboczej kwestionariusza, dzięki czemu klient może przerwać wypełnianie i wrócić do kolejnego niedokończonego kroku. Ukończenie kwestionariusza jest warunkiem wysłania zgłoszenia o współpracę. Dzięki temu trener otrzymuje dane wejściowe przed podjęciem decyzji o przyjęciu klienta oraz przed przygotowaniem planu.

Pierwszy etap konfiguracji profilu klienta (rysunek 42) jest obligatoryjny, jest to krok, w którym użytkownik uzupełnia podstawowe dane osobowe, takie jak imię, nazwisko, numer telefonu oraz data urodzenia. Dane te umożliwiają trenerowi łatwiejszy kontakt z klientem.

The screenshot shows a mobile application interface for a personal data form. At the top, it says 'Dane osobowe' and 'Krok 1 z 14'. Below this is a progress bar. In the center, there is a green circular icon with a person silhouette and the text 'Uzupełnij swoje dane osobowe, aby trenerzy mogli się z Tobą skontaktować.' Below this are four input fields: 'Imię *' (Name) with the value 'Jan', 'Nazwisko *' (Surname) with the value 'Kowalski', 'Numer telefonu *' (Phone number) with the value '+48 123 456 789', and 'Data urodzenia *' (Date of birth) with a placeholder 'dd/mm/yyyy' and a calendar icon. At the bottom right, there is a green button with a right arrow and the text 'Dalej'.

Rysunek 42. Formularz danych osobowych (klient) - krok 1. Źródło: opracowanie własne.

Drugi etap formularza profilu klienta (rysunek 43) służący do wprowadzenia podstawowych parametrów fizycznych użytkownika, takich jak masa ciała oraz wzrost. Informacje te są wykorzystywane do lepszego dopasowania planu treningowego.

Podaj swoje podstawowe dane fizyczne, które pomogą trenerowi dopasować plan treningowy.

Waga (kg) * 75 Podaj aktualną wagę	Wzrost (cm) * 180 Podaj wzrost w centymetrach
--	---

Dokończ później [Dalej](#)

Rysunek 43. Formularz danych osobowych (klient) - krok 2. Źródło: opracowanie własne.

Trzeci etap konfiguracji profilu klienta, przedstawiony na rysunku 44, umożliwia określenie głównego celu treningowego użytkownika. Klient może wybrać jedną z dostępnych opcji, takich jak budowa masy mięśniowej, redukcja tkanki tłuszczowej, zwiększenie siły, poprawa wytrzymałości, ogólna poprawa kondycji, rehabilitacja lub przygotowanie do sportu. Wybrany cel jest wykorzystywany przez system do personalizacji planu treningowego oraz dopasowania odpowiednich ćwiczeń do potrzeb użytkownika.

Wybierz swój główny cel treningowy.

Budowa masy mięśniowej	Redukcja tkanki tłuszczowej
Zwiększenie siły	Poprawa wytrzymałości
Ogólna poprawa kondycji	Rehabilitacja
Przygotowanie do sportu	

[Wstecz](#) Dokończ później [Dalej](#)

Rysunek 44. Formularz danych osobowych (klient) - krok 3. Źródło: opracowanie własne.

Czwarty etap konfiguracji profilu klienta, przedstawiony na rysunku 45, umożliwia określenie preferowanego miejsca wykonywania treningów. Użytkownik może wybrać jedną z dostępnych opcji, takich jak siłownia, dom, trening na zewnątrz lub mała siłownia/studio. Informacja ta pozwala systemowi lepiej dopasować plan treningowy do warunków, w jakich klient będzie ćwiczył.

Gdzie planujesz ćwiczyć?

 Siłownia	 Dom
 Na zewnątrz	 Mała siłownia / Studio

[< Wstecz](#)
[Dokończ później](#)
[> Dalej](#)

Rysunek 45. Formularz danych osobowych (klient) - krok 4. Źródło: opracowanie własne.

Piąty etap formularza, zaprezentowany na rysunku 46, dotyczy poziomu doświadczenia treningowego użytkownika w przypadku wcześniejszego wyboru opcji „siłownia” lub „na zewnątrz”. Klient określa swój stopień zaawansowania poprzez wybór jednej z trzech kategorii: początkujący, średniozaawansowany lub zaawansowany. Dane te umożliwiają dostosowanie intensywności oraz trudności planu treningowego.

Jakie jest Twoje doświadczenie treningowe?

 Początkujący - do 6 miesięcy
 Średniozaawansowany - 6 miesięcy do 2 lat
 Zaawansowany - ponad 2 lata

[< Wstecz](#)
[Dokończ później](#)
[> Dalej](#)

Rysunek 46. Formularz danych osobowych (klient) - krok 5 dla wcześniej wybranej opcji „siłownia” lub „na zewnątrz”. Źródło: opracowanie własne.

Alternatywna wersja piątego etapu formularza została przedstawiona na rysunku 47 jest wyświetlana użytkownikowi po wybraniu opcji „dom” lub „mała siłownia”. Formularz umożliwia wskazanie dostępnego sprzętu treningowego, takiego jak hantle, sztanga, gumy oporowe czy mata. Użytkownik może zaznaczyć wiele opcji jednocześnie lub wskazać brak dodatkowego sprzętu. Informacje te pozwalają trenerowi przygotować plan ćwiczeń dostosowany do dostępnego wyposażenia.

Jakim sprzętem dysponujesz w domu lub małej siłowni? Możesz wybrać wiele opcji.

☐ Hantle	☐ Sztanga
☐ Kettlebell	☐ Gumy oporowe
☐ Drążek do podciągania	☐ Ławeczka
☐ TRX / Taśmy do ćwiczeń	☐ Mata
☐ Skakanka	
☐ Brak dodatkowego sprzętu	

< Wstecz Dokończ później > Dalej

Rysunek 47. Formularz danych osobowych (klient) - krok 5 dla wcześniej wybranej opcji „dom” lub „mała siłownia”. Źródło: opracowanie własne.

Szósty etap konfiguracji profilu klienta, pokazany na rysunku 48, służy do określenia aktualnej częstotliwości treningów użytkownika. Klient wybiera jedną z dostępnych odpowiedzi, określających liczbę treningów wykonywanych w ciągu tygodnia. Dane te są wykorzystywane do dopasowania objętości oraz intensywności planu treningowego do aktualnej aktywności fizycznej użytkownika.

Jak często aktualnie trenujesz?

☐ Nie trenuję
☐ 1-2 razy w tygodniu
☐ 3-4 razy w tygodniu
☐ 5-6 razy w tygodniu
☐ Codziennie

< Wstecz Dokończ później > Dalej

Rysunek 48. Formularz danych osobowych (klient) - krok 6. Źródło: opracowanie własne.

Siódmy etap konfiguracji profilu klienta, przedstawiony na rysunku 49, umożliwia określenie preferowanych partii mięśniowych, na których użytkownik chce się skupić podczas treningów. Klient może zaznaczyć wiele grup mięśniowych jednocześnie lub wybrać opcję „Całe ciało - bez preferencji”.

Na jakie partie mięśniowe chcesz się skupić? Możesz wybrać wiele opcji lub "Całe ciało".

☐ Klatka piersiowa	☐ Plecy	☐ Barki
☐ Biceps	☐ Triceps	☐ Przedramiona
☐ Brzuch	☐ Pośladki	☐ Czworogłowe uda
☐ Dwugłowe uda	☐ Łydki	
☐ Całe ciało - bez preferencji		

< Wstecz Dokończ później > Dalej

Rysunek 49. Formularz danych osobowych (klient) - krok 7. Źródło: opracowanie własne.

Ósmy etap formularza, pokazany na rysunku 50, służy do przekazania informacji dotyczących kontuzji lub ograniczeń zdrowotnych użytkownika. Klient może wskazać brak przeciwwskazań lub opisać występujące problemy zdrowotne w dodatkowym polu tekstowym. Dane te umożliwiają trenerowi przygotowanie bezpiecznego planu treningowego, uwzględniającego stan zdrowia użytkownika.



Czy masz jakiegokolwiek kontuzje lub ograniczenia zdrowotne, o których trener powinien wiedzieć?

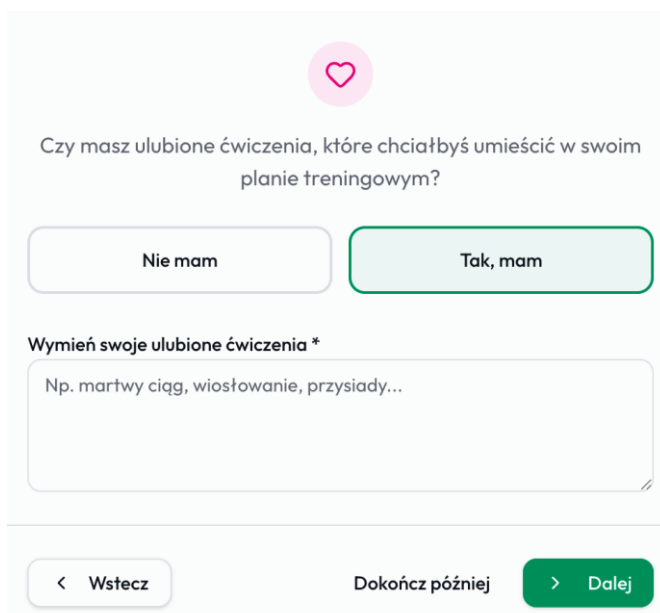
Opisz swoje kontuzje lub ograniczenia *

Np. kontuzja kolana z 2020 roku, problemy z kręgosłupem...

< Wstecz Dokończ później > Dalej

Rysunek 50. Formularz danych osobowych (klient) - krok 8. Źródło: opracowanie własne.

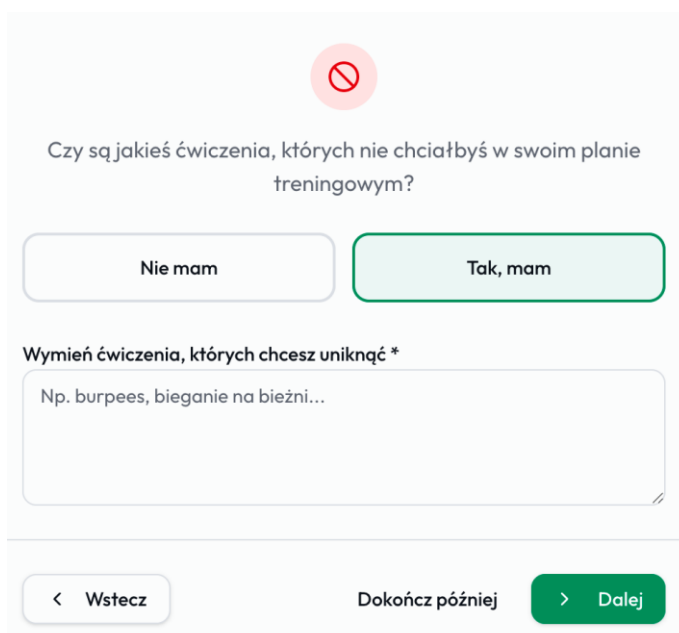
Dziewiąty etap konfiguracji profilu klienta, przedstawiony na rysunku 51, pozwala użytkownikowi określić ulubione ćwiczenia, które chciałby uwzględnić w swoim planie treningowym. W przypadku wyboru odpowiedzi twierdzącej użytkownik może wpisać preferowane ćwiczenia w dedykowanym polu tekstowym.



Formularz konfiguracji profilu klienta - krok 9. Na górze znajduje się ikona serca w czerwonym okręgu. Poniżej jest pytanie: "Czy masz ulubione ćwiczenia, które chciałbyś umieścić w swoim planie treningowym?". Dwa przyciski: "Nie mam" i "Tak, mam" (z zieloną ramką). Poniżej jest sekcja "Wymień swoje ulubione ćwiczenia *" z polem tekstowym zawierającym przykład: "Np. martwy ciąg, wiosłowanie, przysiady...". Na dole znajdują się przyciski nawigacyjne: "< Wstecz", "Dokończ później" i "> Dalej" (z zieloną ramką).

Rysunek 51. Formularz danych osobowych (klient) - krok 9. Źródło: opracowanie własne.

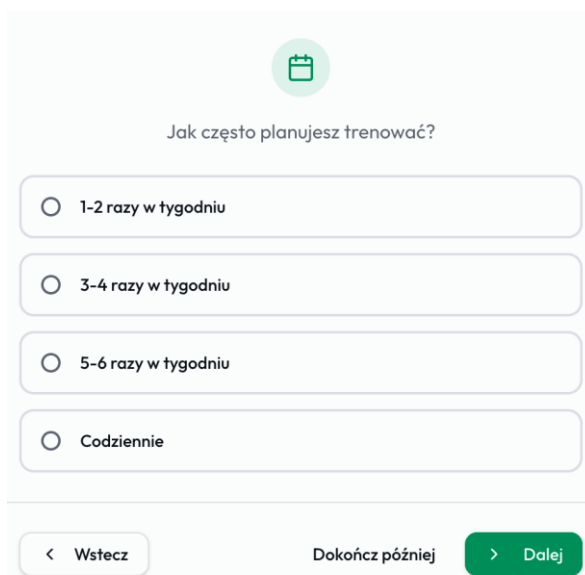
Dziesiąty etap formularza, zaprezentowany na rysunku 52, umożliwia wskazanie ćwiczeń, których użytkownik nie chce wykonywać w swoim planie treningowym. Po zaznaczeniu odpowiedniej opcji klient może podać listę niepożądanych ćwiczeń w polu tekstowym.



Formularz konfiguracji profilu klienta - krok 10. Na górze znajduje się ikona zakazania (czerwony okrąg z przekreśleniem). Poniżej jest pytanie: "Czy są jakieś ćwiczenia, których nie chciałbyś w swoim planie treningowym?". Dwa przyciski: "Nie mam" i "Tak, mam" (z zieloną ramką). Poniżej jest sekcja "Wymień ćwiczenia, których chcesz uniknąć *" z polem tekstowym zawierającym przykład: "Np. burpees, bieganie na bieżni...". Na dole znajdują się przyciski nawigacyjne: "< Wstecz", "Dokończ później" i "> Dalej" (z zieloną ramką).

Rysunek 52. Formularz danych osobowych (klient) - krok 10. Źródło: opracowanie własne.

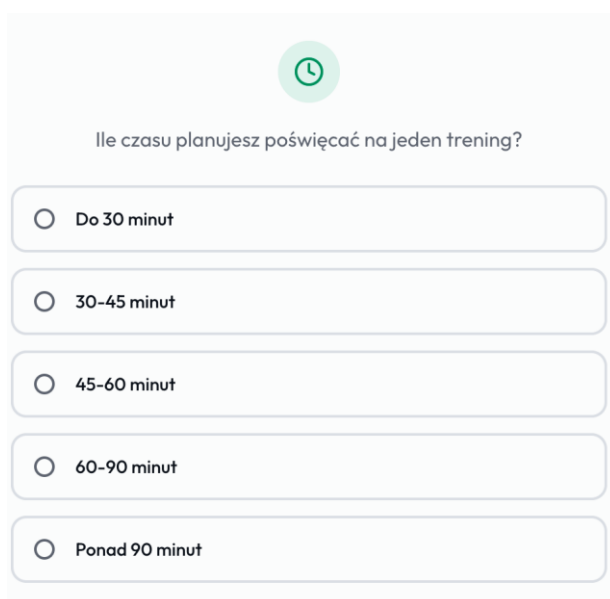
Jedenasty etap konfiguracji profilu klienta, przedstawiony na rysunku 53, służy do określenia planowanej częstotliwości treningów. Użytkownik wybiera jedną z dostępnych opcji określających liczbę treningów wykonywanych w ciągu tygodnia. Informacja ta jest wykorzystywana podczas tworzenia harmonogramu oraz planowania objętości treningowej.



Formularz z tytułem "Jak często planujesz trenować?". Zawiera cztery opcje wyboru: "1-2 razy w tygodniu", "3-4 razy w tygodniu", "5-6 razy w tygodniu" oraz "Codziennie". Na dole znajdują się przyciski nawigacyjne: "Wstecz", "Dokończ później" oraz "Dalej".

Rysunek 53. Formularz danych osobowych (klient) - krok 11. Źródło: opracowanie własne.

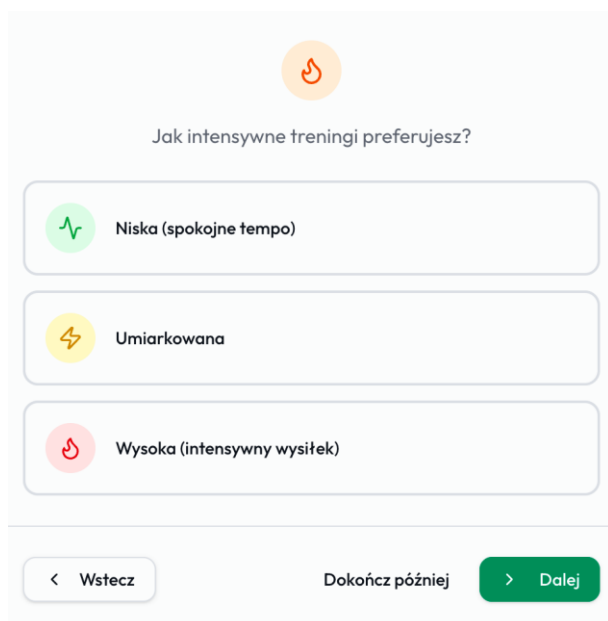
Dwunasty etap formularza, pokazany na rysunku 54, umożliwia określenie planowanego czasu trwania pojedynczej jednostki treningowej. Klient wybiera przedział czasowy najlepiej odpowiadający jego możliwościom oraz preferencjom.



Formularz z tytułem "Ile czasu planujesz poświęcać na jeden trening?". Zawiera pięć opcji wyboru: "Do 30 minut", "30-45 minut", "45-60 minut", "60-90 minut" oraz "Ponad 90 minut".

Rysunek 54. Formularz danych osobowych (klient) - krok 12. Źródło: opracowanie własne.

Trzynasty etap konfiguracji profilu klienta, przedstawiony na rysunku 55, pozwala określić preferowaną intensywność treningów. Użytkownik może wybrać niski, umiarkowany lub wysoki poziom intensywności, co umożliwia dostosowanie obciążenia treningowego do indywidualnych możliwości i oczekiwań klienta.



Jak intensywne treningi preferujesz?

Niska (spokojne tempo)

Umiarkowana

Wysoka (intensywny wysiłek)

< Wstecz Dokończ później > Dalej

Rysunek 55. Formularz danych osobowych (klient) - krok 13. Źródło: opracowanie własne.

Czternasty, ostatni etap formularza danych osobowych klienta, przedstawiony na rysunku 56, umożliwia przekazanie dodatkowych informacji dla trenera. Użytkownik może wpisać opcjonalne uwagi dotyczące m.in. preferowanych godzin treningów, dodatkowych celów czy alergii pokarmowych. Po zakończeniu tego etapu profil klienta zostaje w pełni skonfigurowany.

Ostatni krok!

Czy jest coś jeszcze, o czym Twój trener powinien wiedzieć?
Możesz podzielić się dodatkowymi uwagami.

Dodatkowe uwagi (opcjonalne)

Np. preferowane godziny treningów, dodatkowe cele, alergie pokarmowe...

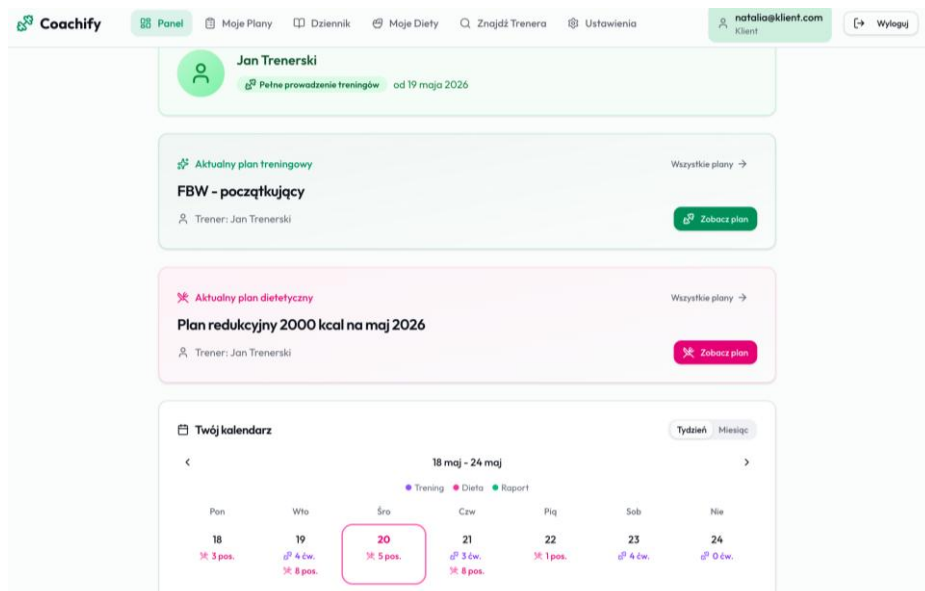
To pole jest opcjonalne. Możesz je zostawić puste.

[< Wstecz](#) [✓ Zakończ](#)

Rysunek 56. Formularz danych osobowych (klient) - krok 14. Źródło: opracowanie własne.

6.3.2 Pulpit podopiecznego

Pulpit klienta jest widokiem agregującym informacje o współpracy, planach i raportach. Poza informacją o aktywnej współpracy prezentuje tygodniowy kalendarz łączący dane treningów, posiłków i raportów. Taki układ ogranicza konieczność przechodzenia między wieloma widokami w celu sprawdzenia bieżących zadań. Jeżeli *onboarding* nie został ukończony lub został pominięty, aplikacja prezentuje baner z komunikatem o możliwości dokończenia kwestionariusza.

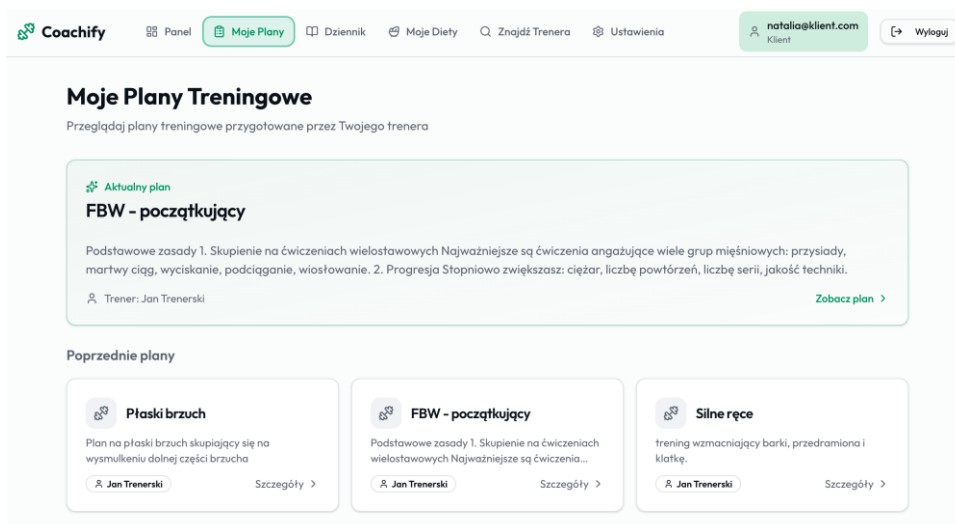


Rysunek 57. Widok panelu klienta z aktywnym planem dietetycznym. Źródło: opracowanie własne.

Na rysunku 57 przedstawiono panel główny klienta po przypisaniu planu dietetycznego przez trenera. Aktualny plan dietetyczny jest prezentowany bezpośrednio na stronie głównej obok aktywnego planu treningowego.

Dodatkowo plan dietetyczny jest integrowany z kalendarzem klienta. W widoku tygodniowym użytkownik widzi liczbę zaplanowanych posiłków dla danego dnia, co pozwala szybko sprawdzić harmonogram diety oraz połączyć go z planem treningowym.

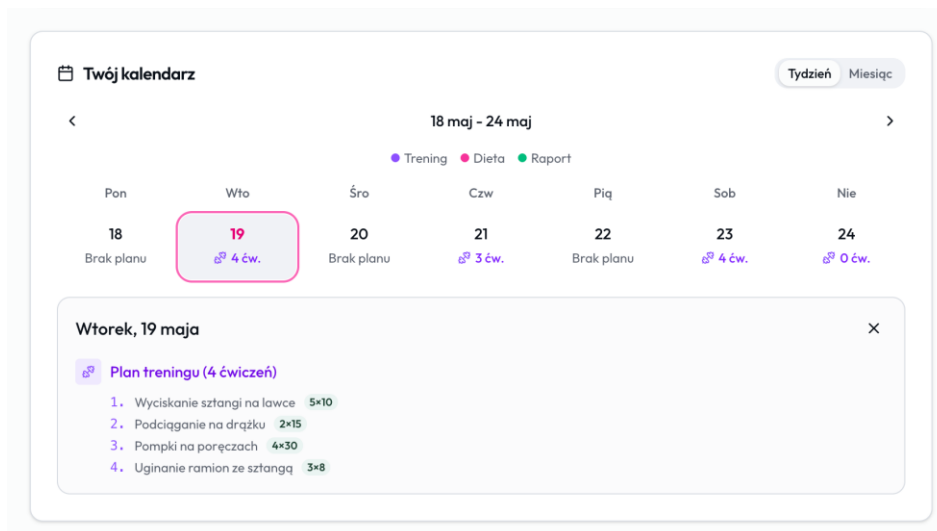
6.3.3 Plany treningowe i kalendarz treningowy



Rysunek 58. Widok planów treningowych z perspektywy klienta. Źródło: opracowanie własne.

Na rysunku 58 przedstawiono moduł „Moje Plany” dostępny dla klienta. System wyświetla aktualnie aktywny plan treningowy w wyróżnionej sekcji u góry ekranu. Klient widzi nazwę planu, jego opis oraz informacje o trenerze odpowiedzialnym za przygotowanie programu.

Poniżej prezentowana jest lista poprzednich planów treningowych, do których klient nadal może uzyskać dostęp w celu przeglądania wcześniejszych treningów.

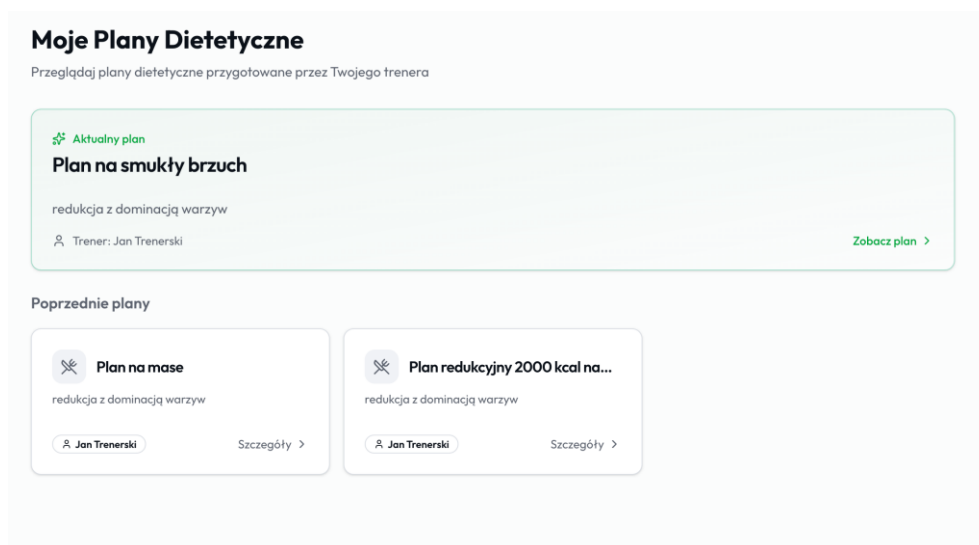


Rysunek 59. Kalendarz treningowy klienta. Źródło: opracowanie własne.

Na rysunku 59 przedstawiono widok kalendarza treningowego dostępnego dla klienta. Kalendarz prezentuje tygodniowy harmonogram aktywności wraz z oznaczeniem dni, w których zaplanowano treningi. Dla każdego dnia system wyświetla liczbę ćwiczeń przypisanych do wykonania.

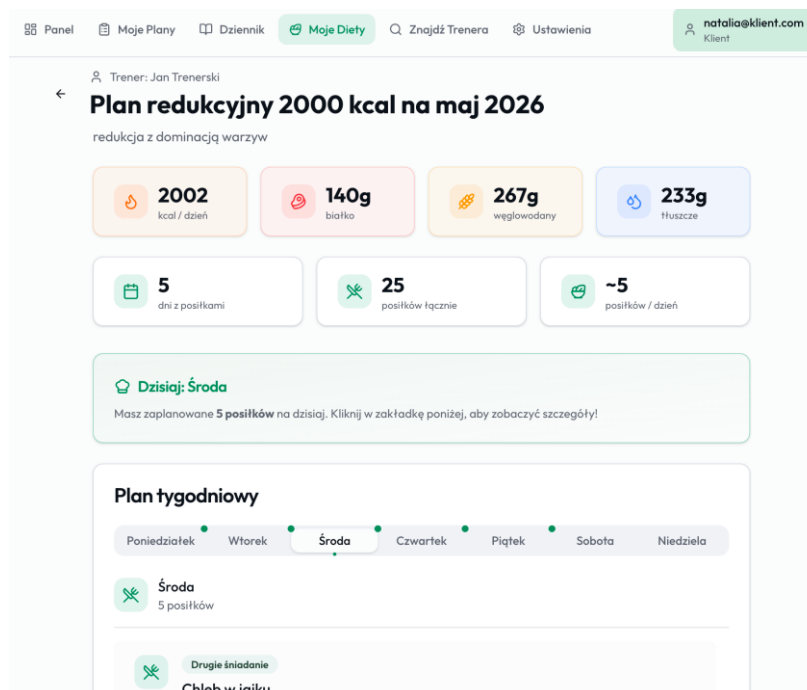
Po wybraniu konkretnego dnia użytkownik otrzymuje podgląd szczegółowego planu treningowego obejmującego listę ćwiczeń wraz z liczbą serii i powtórzeń.

6.3.4 Diety i szczegóły planu dietetycznego

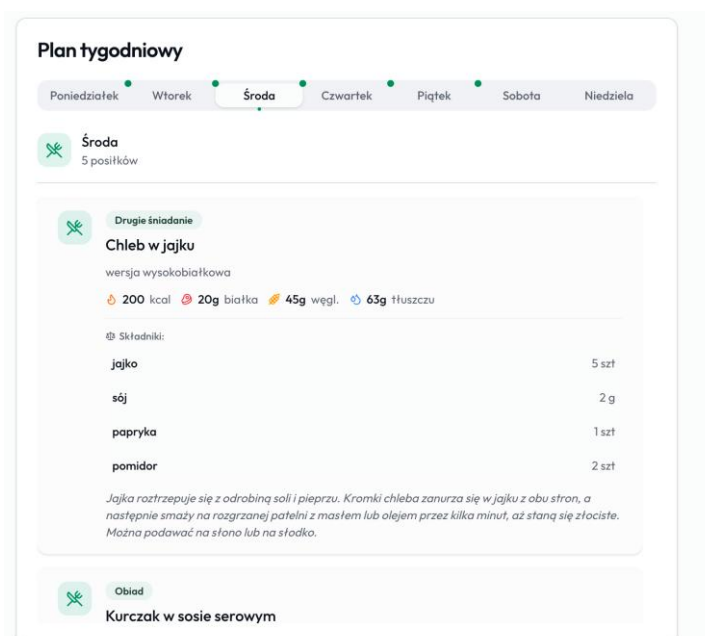


Rysunek 60. Zakładka „Moje Diety”. Źródło: opracowanie własne.

Na rysunku 60 przedstawiono widok listy planów dietetycznych dostępnych dla klienta. System wyświetla aktualnie aktywny plan wraz z podstawowymi informacjami, takimi jak nazwa, opis oraz trener odpowiedzialny za przygotowanie diety.



Rysunek 61. Szczegóły planu dietetycznego z perspektywy klienta cz. 1. Źródło: opracowanie własne.



Rysunek 62. Szczegóły planu dietetycznego z perspektywy klienta cz. 2. Źródło: opracowanie własne.

Na rysunkach 61 i 62 przedstawiono szczegółowy widok planu dietetycznego dostępny dla klienta. System prezentuje podsumowanie wartości odżywczych, w tym dzienną kaloryczność oraz ilość białka, węglowodanów i tłuszczów.

Widoczne są również dodatkowe informacje, takie jak liczba dni z przypisanymi posiłkami, całkowita liczba posiłków oraz średnia liczba posiłków na dzień.

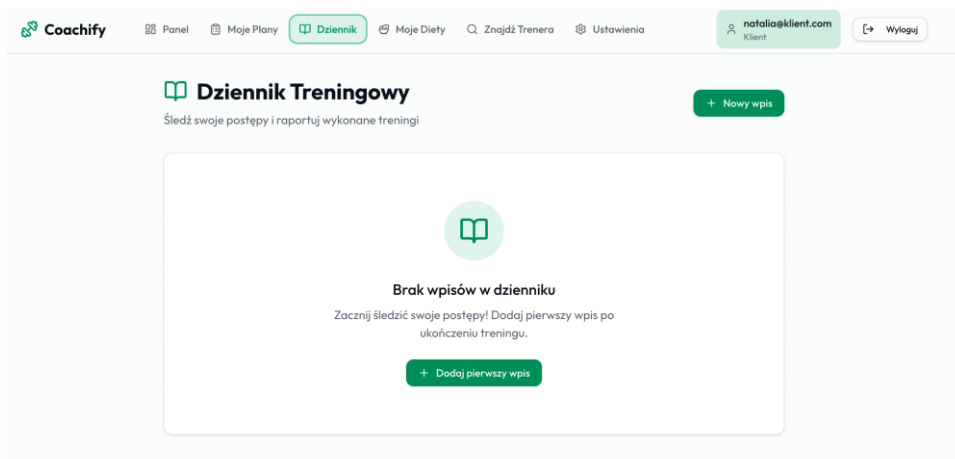
Każdy posiłek zawiera informacje o typie posiłku, np. śniadanie, drugie śniadanie, obiad lub kolacja, co ułatwia organizację diety i szybkie odnalezienie odpowiednich pozycji w planie. Widoczne

są również podstawowe wartości odżywcze posiłku, takie jak kalorie, ilość białka, węglowodanów oraz tłuszczów.

6.3.5 Dziennik treningowy i raportowanie postępów

Raporty treningowe służą do dokumentowania wykonania jednostek treningowych. Klient tworzy raport na podstawie wybranego dnia planu, wprowadza dane dotyczące serii, takie jak obciążenie, liczba powtórzeń lub czas, oraz oznacza ewentualnie pominięte ćwiczenia. Raport jest powiązany z planem treningowym i konkretnym dniem planu, co pozwala zachować kontekst wykonania.

Trener przegląda raporty swoich klientów i może oceniać postęp na podstawie danych liczbowych oraz zdjęć. System obsługuje dołączenie maksymalnie trzech zdjęć postępów do raportu.



Rysunek 63. Moduł raportowania treningów i śledzenia postępów klienta cz. 1. Źródło: opracowanie własne.

Rysunek 64. Moduł raportowania treningów i śledzenia postępów klienta cz. 2. Źródło: opracowanie własne.

2 Uginanie nadgarstków Plan: 3 serii 10 powt. Pominięte

Notatki do ćwiczenia (opcjonalne)...

3 Pompki na poręczach Plan: 3 serii 10 powt. Pomiń

Powtórzenia Czas (sek)

Seria	Obciążenie (kg)	Powtórzenia	
1	—	—	🔄 🗑️
2	—	—	🔄 🗑️
3	—	—	✅ 🗑️

+ Dodaj serię

Nie lubię tego ćwiczenia

Rysunek 65. Moduł raportowania treningów i śledzenia postępów klienta cz. 3. Źródło: opracowanie własne.

Ogólne notatki

Trening za ciężki na moją kondycję.

Zdjęcia postępów Opcjonalne

📷

⬆️

Przeciągnij lub kliknij

JPG, PNG, WebP • max 5.0 MB

📷

Dodaj zdjęcia swoich postępów

Maksymalnie 3 zdjęcia, do 5MB każde

Anuluj Zapisz wpis

Rysunek 66. Moduł raportowania treningów i śledzenia postępów klienta cz. 4. Źródło: opracowanie własne.

Na rysunkach 63-66 przedstawiono moduł dziennika treningowego.

Klient może utworzyć nowy wpis w dzienniku po zakończonym treningu. Podczas dodawania wpisu wybierany jest dzień treningowy przypisany wcześniej przez trenera. Na tej podstawie system automatycznie wyświetla listę ćwiczeń zaplanowanych dla wybranego dnia wraz z docelową liczbą serii oraz powtórzeń.

Dla każdego ćwiczenia użytkownik może uzupełnić rzeczywiste wyniki treningowe, takie jak liczba wykonanych powtórzeń, zastosowane obciążenie lub czas wykonywania ćwiczenia. System

umożliwia również oznaczanie wykonanych serii oraz dodawanie kolejnych serii w przypadku realizacji większej objętości treningowej niż zakładał plan.

Klient może oznaczyć wybrane ćwiczenie jako pominięte, jeśli nie zostało wykonane podczas treningu. Dodatkowo dostępne są pola notatek zarówno dla pojedynczych ćwiczeń, jak i całego treningu. Pozwala to przekazać trenerowi informacje dotyczące trudności treningu, poziomu zmęczenia, bólu lub konieczności modyfikacji planu.

Moduł wspiera również dodawanie zdjęć postępów.

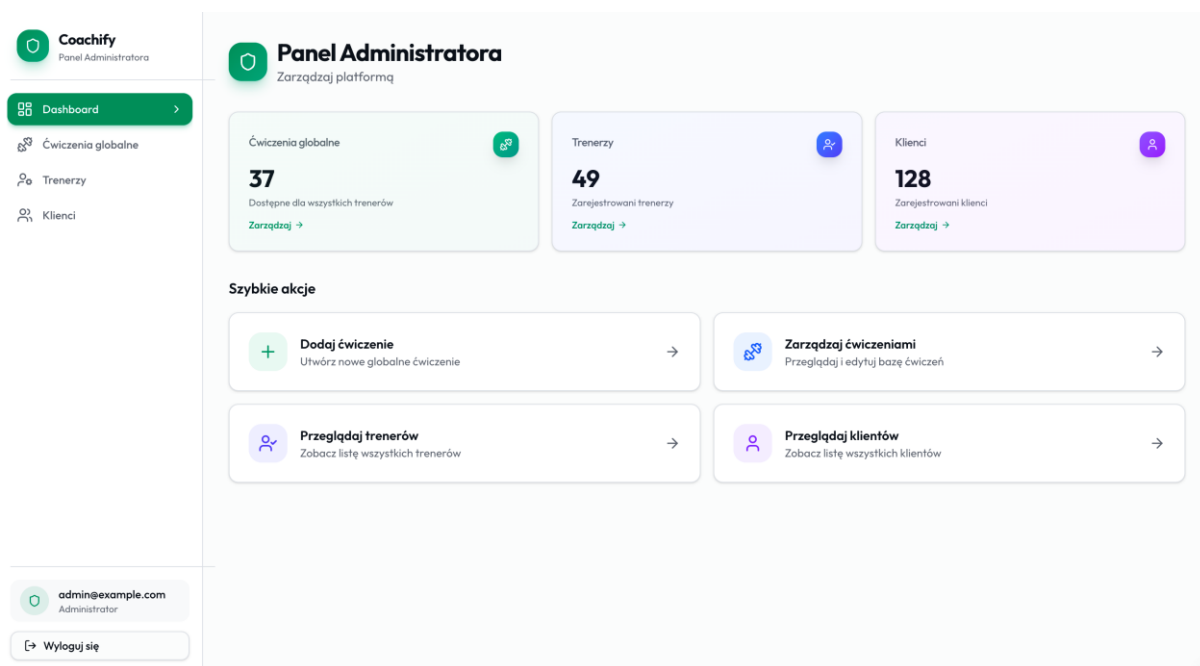
Wszystkie wpisy w dzienniku są zapisywane w systemie i mogą być później analizowane zarówno przez klienta, jak i trenera w celu monitorowania regularności treningów oraz skuteczności realizowanego planu.

6.4. Widoki administratora

Panel administratora został wydzielony do osobnego układu z bocznym paskiem nawigacji. Obejmuje cztery główne widoki: pulpit, ćwiczenia globalne, trenerów oraz klientów. Wszystkie trasy administracyjne są chronione mechanizmem wymagającym odpowiedniej roli.

6.4.1 Pulpit administratora

Administrator ma dostęp do list trenerów i klientów oraz do zarządzania globalną biblioteką ćwiczeń. Ćwiczenia globalne są dostępne dla wszystkich trenerów i mogą być wykorzystywane w planach treningowych.

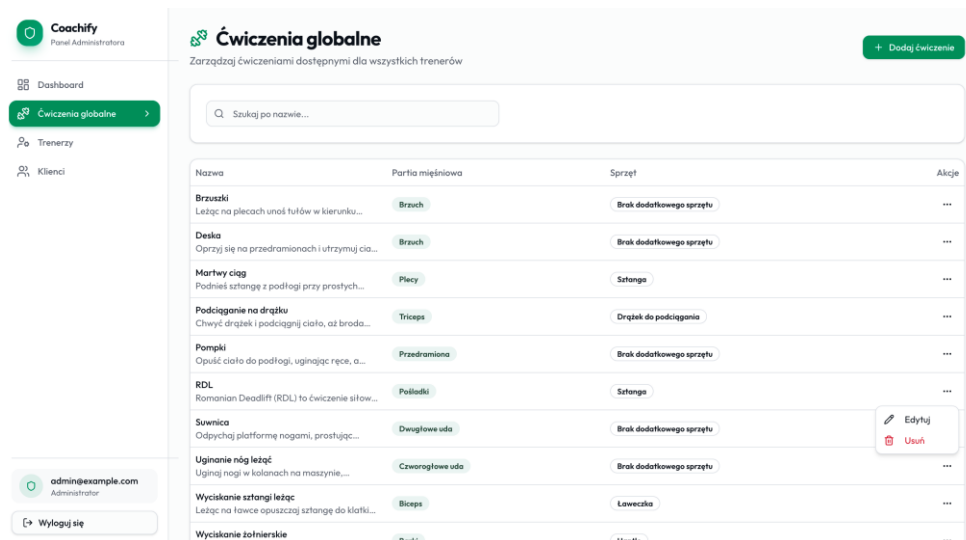


Rysunek 67. Panel główny administratora. Źródło: opracowanie własne.

Na rysunku 67 przedstawiono główny panel administratora. Widok ten umożliwia szybki dostęp do najważniejszych funkcji administracyjnych platformy. Administrator może monitorować podstawowe statystyki systemu, takie jak liczba ćwiczeń globalnych, liczba trenerów oraz liczba

klientów korzystających z aplikacji. Panel zawiera również sekcję szybkich akcji pozwalającą na natychmiastowe przejście do zarządzania ćwiczeniami, trenerami lub klientami.

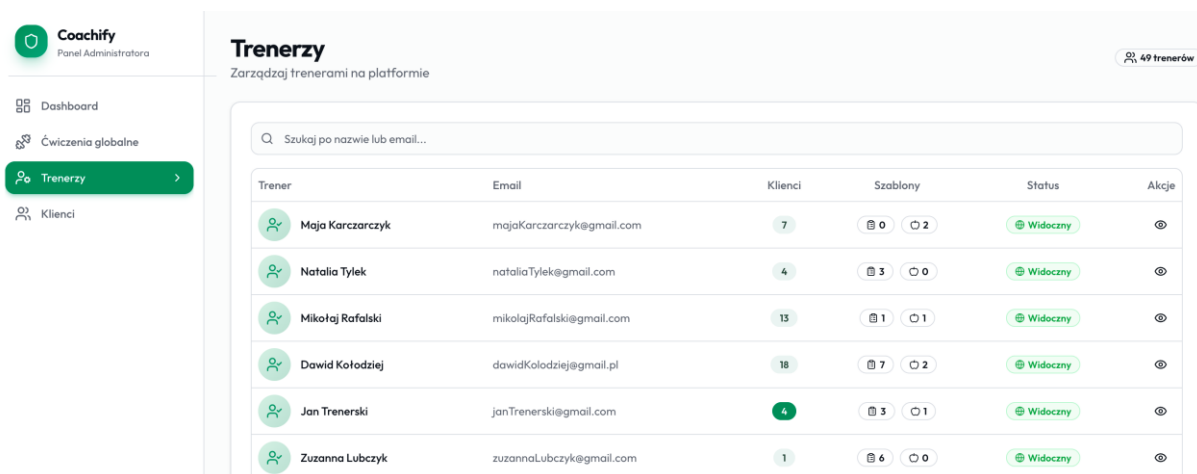
6.4.2 Zarządzanie ćwiczeniami globalnymi



Rysunek 68. Zarządzanie ćwiczeniami globalnymi. Źródło: opracowanie własne.

Na rysunku 68 przedstawiono moduł zarządzania ćwiczeniami globalnymi dostępnymi dla wszystkich trenerów korzystających z platformy. Administrator może przeglądać listę ćwiczeń wraz z podstawowymi informacjami, takimi jak nazwa ćwiczenia, opis, trenowana partia mięśniowa oraz wymagany sprzęt. System umożliwia wyszukiwanie ćwiczeń, dodawanie nowych pozycji, a także edycję lub usuwanie istniejących elementów bazy.

6.4.3 Zarządzanie trenerami

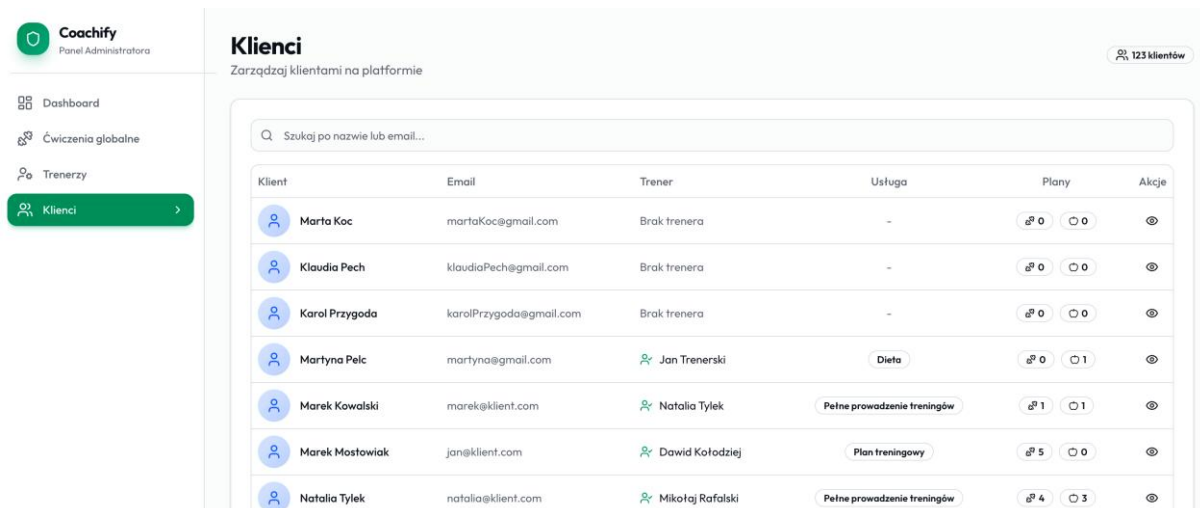


Rysunek 69. Zarządzanie trenerami w systemie. Źródło: opracowanie własne.

Na rysunku 69 zaprezentowano moduł zarządzania trenerami zarejestrowanymi na platformie. Administrator może przeglądać listę trenerów wraz z informacjami dotyczącymi adresu e-mail, liczby

przypisanych klientów oraz liczby utworzonych planów treningowych i dietetycznych. Widoczny jest również status konta trenera, umożliwiając kontrolowanie dostępności profilu w systemie.

6.4.4 Zarządzanie klientami



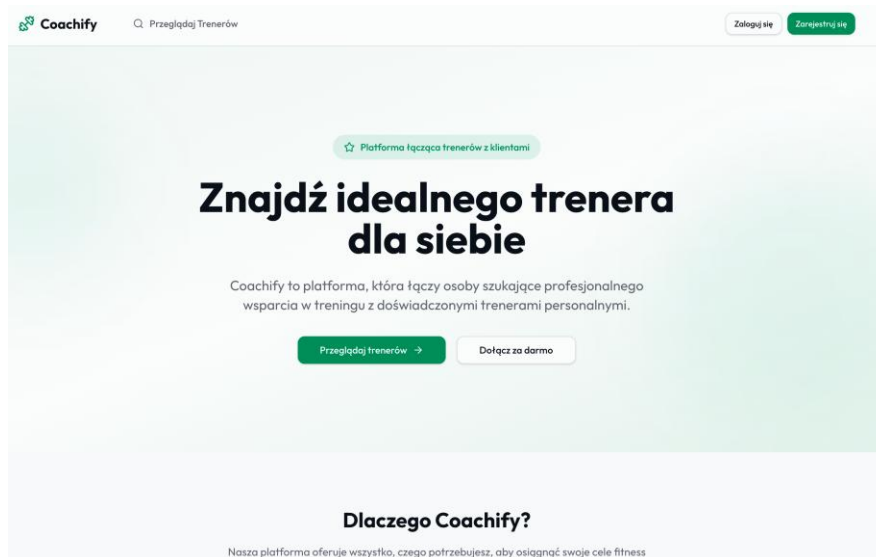
Rysunek 70. Zarządzanie klientami platformy. Źródło: opracowanie własne.

Na rysunku 70 przedstawiono moduł zarządzania klientami korzystającymi z systemu Coachify. Administrator ma możliwość przeglądania wszystkich klientów wraz z informacjami dotyczącymi przypisanego trenera, rodzaju aktywnej usługi oraz liczby planów treningowych i dietetycznych przypisanych do danego użytkownika. System umożliwia szybkie wyszukiwanie klientów oraz kontrolowanie relacji pomiędzy trenerami i podopiecznymi.

6.5. Widoki publiczne i *marketplace*

6.5.1 Strona główna, rejestracja i logowanie

Wchodząc na stronę wyświetla się widok pokazany na rysunku 71. W prawym górnym rogu widoczna jest możliwość rejestracji oraz logowania.



Rysunek 71. Widok główny bez logowania. Źródło: opracowanie własne.

Klikając w pole „Zarejestruj się” pokazuje się formularz rejestracji (rysunek 72) umożliwiający utworzenie nowego konta w aplikacji. Użytkownik wybiera rolę klienta lub trenera, a następnie podaje adres e-mail oraz hasło potrzebne do założenia konta.

Rysunek 72. Panel rejestracji. Źródło: opracowanie własne.

Wybierając opcję „Zaloguj się” z poziomu widoku startowego, ukazuje się ekran logowania (rysunek 73) pozwalający użytkownikowi uzyskać dostęp do wcześniej utworzonego konta poprzez podanie adresu e-mail oraz hasła. Formularz zawiera również odnośnik do procesu rejestracji nowego użytkownika.

Witaj ponownie

Wprowadź swoje dane, aby uzyskać dostęp do konta

Email

jan@example.com

Hasło

Zaloguj się

Nie masz konta? [Zarejestruj się](#)

Rysunek 73. Panel logowania. Źródło: opracowanie własne.

6.5.2 Lista trenerów w *marketplace*

Marketplace jest publiczną częścią systemu prezentującą trenerów widocznych dla użytkowników. Trener pojawia się w *marketplace*, jeżeli ma włączoną widoczność i posiada co najmniej jedną zdefiniowaną ofertę. Oferta może dotyczyć planu treningowego, planu dietetycznego albo pełnego prowadzenia. Każda oferta zawiera typ usługi, cenę oraz opcjonalny opis.

Marketplace nie pełni funkcji sklepu internetowego i nie obsługuje płatności. Jego zadaniem jest znalezienie trenera oraz rozpoczęcie kontaktu lub procesu zgłoszenia. Lista trenerów jest stronicowana i może być filtrowana według typu usługi. Po wejściu w szczegóły trenera klient może podjąć dalszą akcję, zależnie od stanu zalogowania i istniejącej relacji z trenerem.

Coachify

Panel Moje Plany Dziennik Moje Diety **Znajdź Trenera** Ustawienia

natalia@klient.com Klient Wyloguj

Znajdź swojego trenera

Przeglądaj naszych trenerów i znajdź idealnego dla siebie

Filtruj według usługi:

Plan Treningowy **Pełne prowadzenie treningów** Dieta

Jan Treński
1 usługa

Cześć! Nazywam się Janek i pomagam ludziom budować lepszą sylwetkę, siłę oraz pewność siebie dzięki prostym i skutecznym planom treningowym...

249,96 zł

Zobacz profil

Dawid Kołodziej
1 usługa

Jestem trenerem personalnym z kilkuletnim doświadczeniem w pracy zarówno z osobami początkującymi, jak i zaawansowanymi. Specjalizuję...

480,00 zł

Zobacz profil

Mikołaj Rafałski
1 usługa

Od lat pomagam ludziom osiągać ich cele sylwetkowe i zdrowotne. Jako trener personalny skupiam się nie tylko na treningu, ale również na budowaniu dobrych...

229,99 zł

Zobacz profil

Natalia Tylek
1 usługa

Sport to moja pasja, którą od kilku lat przekuwam w pracę jako trener personalny. Pomagam klientom poprawić kondycję, zwiększyć siłę oraz zadbaj o...

234,00 zł

Zobacz profil

Maja Karczarzyk
1 usługa

Jako trener personalny stawiam na indywidualne podejście i pełne wsparcie w drodze do celu. Mam doświadczenie w pracy z osobami chcącymi schudnąć...

276,96 zł

Zobacz profil

Rysunek 74. *Marketplace* trenerów personalnych. Źródło: opracowanie własne.

Na rysunku 74 przedstawiono moduł *marketplace'u* umożliwiający klientom wyszukiwanie trenerów personalnych dostępnych w systemie. Widok prezentuje listę trenerów wraz z podstawowymi informacjami, takimi jak nazwa profilu, krótki opis działalności oraz ceny oferowanych usług.

Marketplace umożliwia filtrowanie trenerów według rodzaju świadczonych usług, np. planów treningowych, pełnego prowadzenia treningowego lub diety. Każdy trener posiada własny profil, do którego klient może przejść w celu zapoznania się ze szczegółami oferty.

6.5.3 Profil trenera, prośba o współpracę i status zgłoszenia

The screenshot shows a user interface for a trainer's profile and a request form. At the top left, there is a back arrow and the text "Powrót do marketplace". The profile section on the left includes a green circular avatar icon, the name "Dawid Kołodziej", and "1 usługa w ofercie". Below this is a button labeled "Wyślij wiadomość". The "O trenerze" section contains a paragraph about the trainer's experience. The "Oferowane usługi" section shows a list of services, with "Pełne prowadzenie treningów" selected and priced at "480,00 zł". On the right, the "Zaaplikuj do trenera" section has a heading "Wyślij prośbę o współpracę". It features a green box for "Wybrana usługa: Pełne prowadzenie treningów". Below is a text area for "Wiadomość do trenera (opcjonalnie)" with a placeholder text. At the bottom right are two buttons: "Anuluj" and "Wyślij prośbę".

Rysunek 75. Profil trenera oraz formularz wysłania prośby o współpracę. Źródło: opracowanie własne.

Na rysunku 75 przedstawiono szczegółowy widok profilu trenera. Klient może zapoznać się z opisem doświadczenia trenera, zakresem oferowanych usług oraz cenami współpracy.

Po wybraniu konkretnej usługi użytkownik ma możliwość wysłania prośby o współpracę. Formularz pozwala dodatkowo przekazać trenerowi wiadomość opisującą cele treningowe, doświadczenie lub oczekiwania klienta. Wysłanie prośby inicjuje proces nawiązywania współpracy pomiędzy klientem a trenerem.



Rysunek 76. Status oczekiwania na odpowiedź trenera. Źródło: opracowanie własne.

Na rysunku 76 przedstawiono widok po poprawnym wysłaniu prośby o współpracę. System informuje klienta, że zgłoszenie zostało przekazane trenerowi i oczekuje na jego decyzję.

6.6. Czat

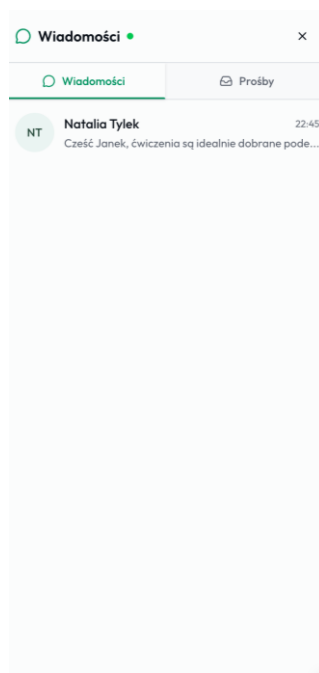
Czat jest dostępny dla zalogowanych trenerów i podopiecznych jako globalny widżet umieszczony w prawym dolnym rogu aplikacji. Dzięki temu użytkownik może prowadzić rozmowę niezależnie od aktualnie otwartego widoku, bez konieczności opuszczania panelu, w którym pracuje. Administrator nie korzysta z modułu czatu.

Moduł umożliwia komunikację jeden-do-jednego między trenerem a klientem. Rozmowa jest zwykle dostępna po rozpoczęciu współpracy, jednak klient może także wysłać prośbę o wiadomość z poziomu profilu trenera w *marketplace*. Taka prośba trafia do trenera, który może ją zaakceptować albo odrzucić.

Przycisk czatu informuje użytkownika o liczbie nieprzeczytanych wiadomości. W przypadku trenera licznik obejmuje również oczekujące prośby o kontakt. Po otwarciu widżetu użytkownik widzi listę konwersacji, a następnie może przejść do wybranej rozmowy. Przy wiadomościach prezentowane są oznaczenia czasu oraz status odczytu. Pojedyncze potwierdzenie oznacza wysłanie wiadomości, natomiast podwójne potwierdzenie informuje, że wiadomość została odczytana przez drugą stronę.

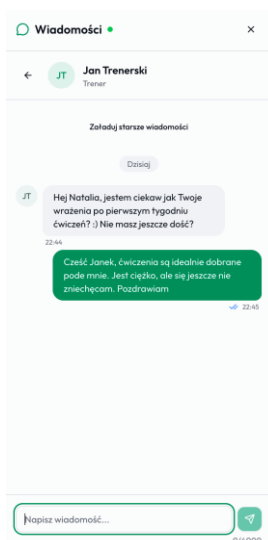
Interfejs trenera zawiera dwie sekcje: konwersacje oraz prośby o wiadomość. Podopieczny widzi wyłącznie listę swoich rozmów. Czat obsługuje wiadomości tekstowe i nie obejmuje funkcji załączników, edycji wiadomości ani wskaźnika pisania.

6.6.1 Lista konwersacji i aktywna rozmowa



Rysunek 77. Lista konwersacji w module wiadomości. Źródło: opracowanie własne.

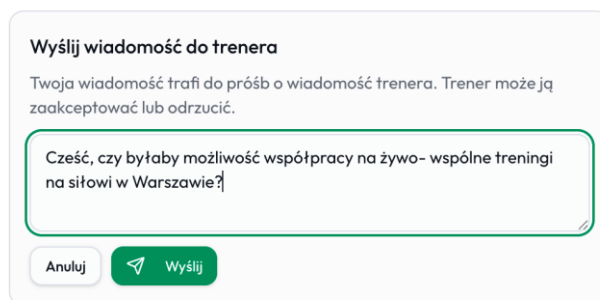
Na rysunku 77 przedstawiono widok listy konwersacji dostępnych w module wiadomości. System wyświetla aktywne rozmowy prowadzone pomiędzy trenerem a klientem wraz z podglądem ostatniej wiadomości, godziną jej wysłania oraz oznaczeniem nowych, nieprzeczytanych wiadomości.



Rysunek 78. Widok aktywnej konwersacji między trenerem a klientem. Źródło: opracowanie własne.

Na rysunku 78 przedstawiono szczegółowy widok aktywnej rozmowy pomiędzy trenerem a klientem. Moduł obsługuje potwierdzenia odczytu wiadomości oraz oznaczenia czasu wysłania.

6.6.2 Prośby o wiadomość



Wyślij wiadomość do trenera

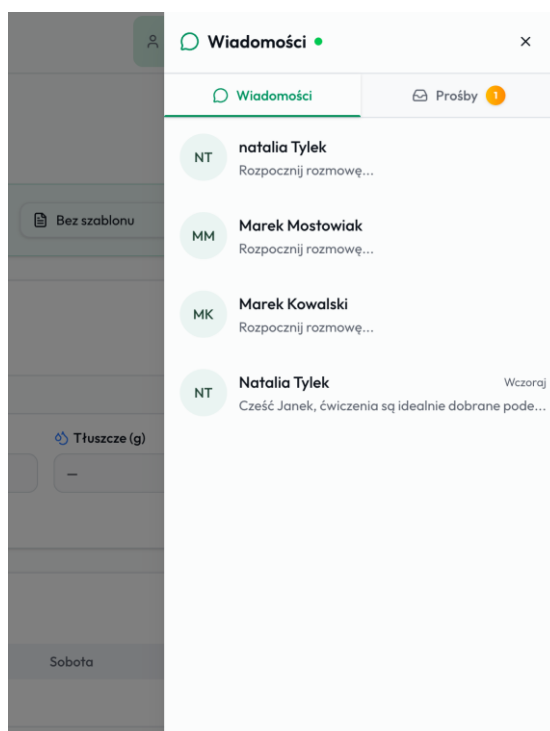
Twoja wiadomość trafi do próśb o wiadomość trenera. Trener może ją zaakceptować lub odrzucić.

Cześć, czy byłaby możliwość współpracy na żywo- wspólne treningi na siłowni w Warszawie?

Anuluj Wyślij

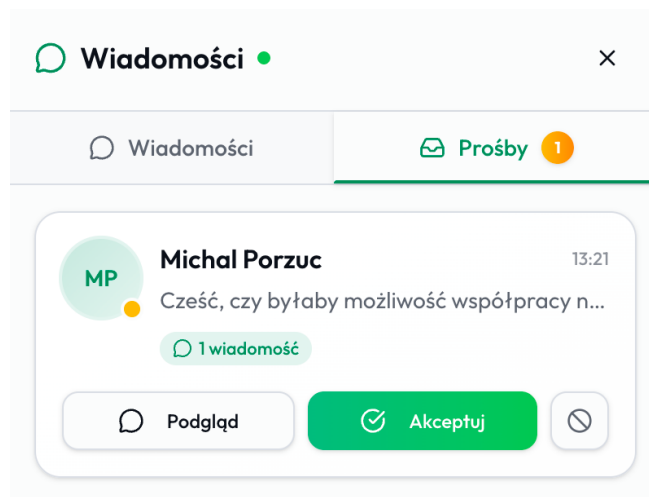
Rysunek 79. Wysyłanie wiadomości do trenera. Źródło: opracowanie własne.

Na rysunku 79 przedstawiono formularz umożliwiający klientowi wysłanie wiadomości do wybranego trenera bez konieczności wcześniejszego nawiązania współpracy. Formularz pozwala klientowi wysłać zapytanie do trenera jeszcze przed rozpoczęciem współpracy. Wiadomość trafia do sekcji próśb o kontakt, gdzie trener może ją zaakceptować albo odrzucić. Dzięki temu użytkownik może rozpocząć kontakt bez wcześniejszego przypisywania usług lub planów treningowych.



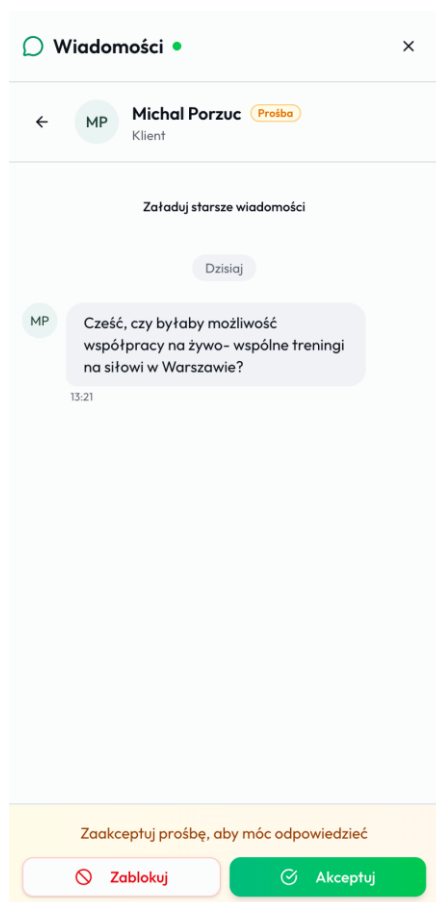
Rysunek 80. Panel wiadomości użytkownika. Źródło: opracowanie własne.

Na rysunku 80 przedstawiono panel wiadomości trenera, w którym widoczna jest prośba (w osobnej sekcji) o kontakt za pośrednictwem wyżej pokazanej prośby.



Rysunek 81. Lista próśb o wiadomość w widoku trenera. Źródło: opracowanie własne.

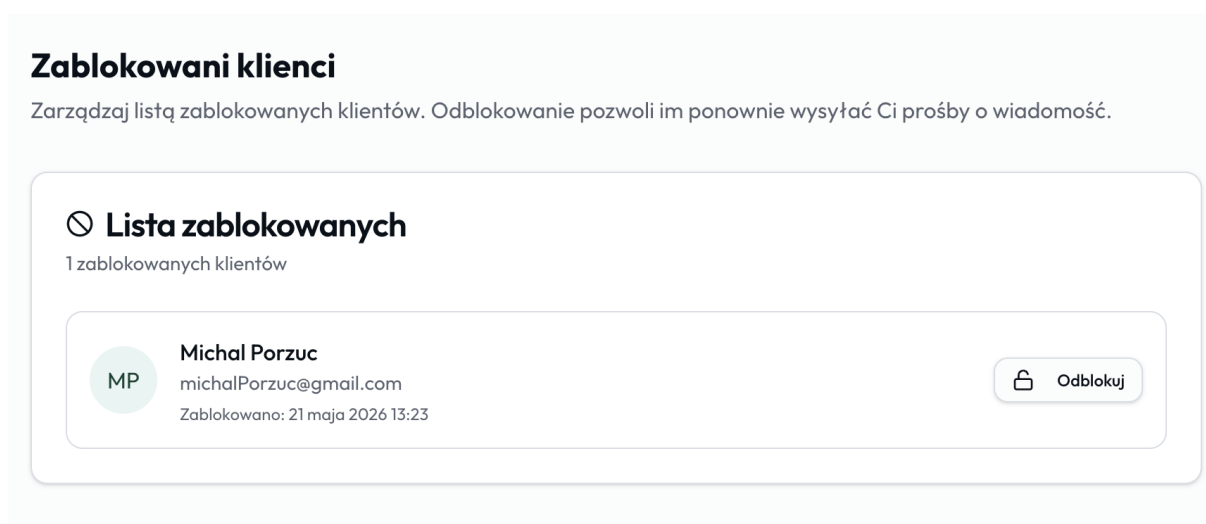
Na rysunku 81 przedstawiono widok próśby o rozpoczęcie rozmowy. Każda wiadomość wysłana przez klienta trafia najpierw do sekcji próśb, gdzie trener może zapoznać się z jej treścią, zaakceptować kontakt lub odrzucić zgłoszenie.



Rysunek 82. Podgląd próśby o kontakt. Źródło: opracowanie własne.

Na rysunku 82 zaprezentowano szczegółowy widok pojedynczej prośby o wiadomość. Trener może odczytać pełną treść wiadomości przesłanej przez klienta jeszcze przed zaakceptowaniem rozmowy. System umożliwia zaakceptowanie prośby i rozpoczęcie dalszej komunikacji lub zablokowanie użytkownika. Do momentu zaakceptowania trener nie może odpowiadać na wiadomość klienta.

6.6.3 Zarządzanie zablokowanymi klientami



Rysunek 83. Zarządzanie zablokowanymi klientami. Źródło: opracowanie własne.

Na rysunku 83 przedstawiono panel ustawień umożliwiający zarządzanie listą zablokowanych klientów. Trener może przeglądać użytkowników, którzy zostali wcześniej zablokowani, wraz z informacją o dacie blokady oraz może ich w dowolnym momencie odblokować. Zablokowany klient nie może ponownie wysłać próśb o kontakt ani rozpocząć nowych rozmów.

6.7. Multimedia

W warstwie interfejsu multimedia pojawiają się w ustawieniach profilu użytkownika, formularzu dodawania ćwiczenia oraz formularzu raportowania treningu. Trener może dodać zdjęcie profilowe i materiał wideo ćwiczenia, natomiast podopieczny może dołączyć zdjęcia postępów do raportu treningowego.

6.8. Responsywność i dostępność

Mobilny układ marketplace'u pokazano na rysunku 84. *Marketplace* prezentuje trenerów w responsywnej siatce kart. Na urządzeniach mobilnych stosowana jest jedna kolumna, na tabletach dwie, a na większych ekranach trzy. Nad listą znajduje się filtr typów usług. Typy usług obejmują plan treningowy, plan dietetyczny oraz pełny coaching. Filtr jest prezentowany jako rząd przełączników generowanych na podstawie danych z API. Jeżeli dane są ładowane, interfejs wykorzystuje szkielety ładowania. Jeżeli wynik jest pusty, prezentowany jest stan pusty z komunikatem zależnym od użytego filtra.



Rysunek 84. Widok *marketplace'u* w układzie mobilnym. Źródło: opracowanie własne.

Responsywność aplikacji wynika również z podziału widoków na układy dostosowane do charakteru pracy użytkownika. Panele trenera i podopiecznego korzystają z górnej nawigacji, natomiast panel administratora wykorzystuje boczny pasek nawigacyjny. Aktywna sekcja jest wyróżniana na podstawie aktualnej ścieżki, co ułatwia orientację w aplikacji.

7. Testowanie aplikacji

W niniejszym rozdziale przedstawiono sposób weryfikacji działania systemu Coachify. Opis obejmuje strategię testowania przyjętą w projekcie, testy automatyczne, weryfikację interfejsu API, testy manualne, testowanie w trakcie budowania obrazów kontenerowych, skanowanie bezpieczeństwa oraz mechanizmy *health check*.

7.1. Podejście do testowania w projekcie

Testowanie systemu Coachify miało charakter iteracyjny i było prowadzone równolegle z implementacją kolejnych modułów aplikacji. W początkowych etapach rozwoju najważniejszą rolę odgrywała weryfikacja manualna pełnych scenariuszy użytkownika, ponieważ pozwalała szybko sprawdzić, czy część serwerowa, aplikacja kliencka oraz mechanizmy uwierzytelniania działają razem w rzeczywistym przepływie. Po ustabilizowaniu głównych modułów systemu zakres weryfikacji uzupełniono o testy automatyczne, których zadaniem było zabezpieczenie najważniejszych reguł biznesowych oraz kontraktów interfejsu API przed regresją.

W części serwerowej zastosowano testy oparte na JUnit, Jupiter, Mockito, AssertJ, `MockMvc` oraz testach wycinkowych Springa. Przyjęto podejście, w którym testy automatyczne koncentrują się przede wszystkim na miejscach istotnych z punktu widzenia działania systemu: uwierzytelnianiu, rotacji tokenów, kontroli dostępu, obsłudze zgłoszeń trenerskich, komunikacji, raportach treningowych, obsłudze plików oraz mapowaniu błędów HTTP. Dzięki temu testy nie obejmują wyłącznie prostych ścieżek sukcesu, lecz również sytuacje wyjątkowe i warunki brzegowe.

W aplikacji klienckiej podstawową formą automatycznej weryfikacji jest kompilacja TypeScript oraz budowanie paczki produkcyjnej przez Vite. Pozwala to wykrywać błędy typowania, niespójności importów oraz problemy uniemożliwiające przygotowanie aplikacji do uruchomienia w środowisku produkcyjnym. Uzupełnieniem tej kontroli była manualna weryfikacja widoków w przeglądarce, obejmująca role klienta, trenera i administratora.

7.2. Testy jednostkowe

Testy jednostkowe zastosowano głównie w warstwie serwisów, czyli tam, gdzie znajduje się zasadnicza logika biznesowa aplikacji. Testowane klasy uruchamiane są w izolacji, a ich zależności, takie jak repozytoria lub inne serwisy, są zastępowane obiektami pozorowanymi tworzonymi za pomocą Mockito. Takie podejście pozwala sprawdzić konkretne reguły domenowe bez konieczności uruchamiania pełnej aplikacji i środowiska bazodanowego.

W module uwierzytelniania testy obejmują rejestrację użytkownika, tworzenie profilu klienta lub trenera, blokadę rejestracji konta administratora, wykrywanie duplikatu adresu e-mail, kodowanie hasła oraz obsługę błędnych danych logowania. Osobno zweryfikowano generowanie i walidację tokenów JWT, w tym odrzucanie tokenów uszkodzonych, wygasłych, podpisanych innym kluczem lub wystawionych dla niewłaściwego odbiorcy.

Istotną część testów stanowi weryfikacja procesu nawiązywania współpracy między klientem i trenerem. Testy `TrainingRequestServiceTest` sprawdzają m.in. utworzenie zgłoszenia, odrzucenie zgłoszenia przy nieukończonym *onboardingu*, brak ukończonego kwestionariusza, aktywną współpracę z innym trenerem, istniejące zgłoszenie oczekujące, ukryty profil trenera oraz brak

wymaganej usługi w ofercie. Dzięki temu utrwalono reguły, które zabezpieczają system przed niespójnymi relacjami pomiędzy użytkownikami.

Testami objęto również wybrane elementy modułu czatu, raportowania treningów, szablonów posiłków oraz obsługi multimediów ćwiczeń. W przypadku plików sprawdzono m.in. odrzucenie pustego pliku, niepoprawnego typu MIME, przekroczenia rozmiaru oraz próbę modyfikacji ćwiczenia bez odpowiednich uprawnień. Testy asynchronicznego przetwarzania miniatur wideo potwierdzają zarówno poprawne zakończenie operacji, jak i oznaczanie przetwarzania jako nieudanego w przypadku błędu.

7.3. Testy warstwy webowej i kontraktu API

Warstwę webową zweryfikowano przy użyciu testów `@WebMvcTest` oraz narzędzia `MockMvc`. Testy tego typu uruchamiają wybrany fragment aplikacji odpowiedzialny za obsługę żądań HTTP, dzięki czemu można sprawdzić kontrolery, walidację danych wejściowych, kody odpowiedzi, strukturę odpowiedzi JSON oraz działanie zabezpieczeń na poziomie ról użytkowników.

Sz szczególnie istotne znaczenie mają testy kontrolera uwierzytelniania. Sprawdzają one rejestrację, logowanie, pobranie danych aktualnego użytkownika, odświeżenie sesji oraz wylogowanie. Weryfikowane są również nagłówki `Set-Cookie`, w tym atrybuty `HttpOnly`, `Secure`, `SameSite=Strict`, `Path` oraz `Max-Age`. Jest to ważne, ponieważ konfiguracja ciasteczek ma bezpośredni wpływ na bezpieczeństwo sesji użytkownika w aplikacji przeglądarkowej.

Drugim ważnym obszarem są testy kontroli dostępu w module próśb o wiadomość. Sprawdzono, czy klient może wysłać prośbę o kontakt do trenera, natomiast operacje listowania, akceptowania, blokowania i odblokowywania są dostępne wyłącznie dla roli trenera. Próba wykonania operacji przez niewłaściwą rolę kończy się odpowiedzią `403 Forbidden`, co potwierdza poprawne działanie adnotacji autoryzacyjnych.

Osobno przetestowano globalną obsługę wyjątków. Testy `GlobalExceptionHandlerTest` mapują najważniejsze typy błędów na odpowiednie kody HTTP, m.in. `400 Bad Request`, `401 Unauthorized`, `404 Not Found`, `405 Method Not Allowed`, `409 Conflict` oraz `500 Internal Server Error`. Dzięki temu aplikacja kliencka otrzymuje przewidywalny format odpowiedzi błędów, a zmiany w kontrakcie API mogą zostać wykryte automatycznie.

7.4. Testy integracyjne

Wybrane scenariusze wymagały testów integracyjnych, ponieważ nie dało się ich wiarygodnie sprawdzić wyłącznie przy użyciu obiektów pozorowanych. Dotyczyło to przede wszystkim operacji zależnych od rzeczywistych transakcji bazodanowych oraz liczby wykonywanych zapytań SQL. W tym celu zastosowano testy `@DataJpaTest` uruchamiane na bazie H2 w trybie zgodności z PostgreSQL.

Pierwszym przykładem jest test rotacji tokenów odświeżających. Mechanizm ten musi poprawnie obsługiwać sytuację, w której ten sam refresh token zostanie użyty ponownie albo dwie operacje rotacji zostaną wykonane niemal równocześnie. Test `RefreshTokenServiceRaceTest` wykorzystuje rzeczywiste transakcje oraz `CountDownLatch`, aby zasymulować równoległe próby rotacji tego samego tokenu. Oczekiwany rezultat zakłada, że tylko jedna operacja może zakończyć się powodzeniem, natomiast druga zostaje potraktowana jako próba ponownego użycia tokenu.

Drugim przykładem jest test liczby zapytań w module obsługi mediów ćwiczeń. Test `ExerciseMediaServiceQueryCountTest` sprawdza, czy pobieranie informacji o plikach wideo i miniaturach dla wielu ćwiczeń odbywa się w sposób wsadowy, a nie przez wykonywanie

osobnych zapytań dla każdego ćwiczenia. Dzięki temu test chroni przed regresją typu N+1, która nie zawsze jest widoczna w prostym scenariuszu funkcjonalnym, ale mogłaby pogorszyć wydajność systemu przy większej liczbie danych.

7.5. Testy manualne

Testy manualne były podstawową formą weryfikacji pełnych przepływów użytkownika w systemie Coachify. Wynikało to z charakteru aplikacji, która łączy wiele warstw: interfejs użytkownika, komunikację HTTP, mechanizmy uwierzytelniania, role użytkowników, bazę danych, obsługę plików oraz komunikację w czasie rzeczywistym. Manualne przechodzenie scenariuszy pozwalało sprawdzić nie tylko pojedyncze *endpointy*, lecz cały proces z perspektywy użytkownika końcowego.

Weryfikacja obejmowała trzy główne role systemowe. Dla klienta sprawdzano rejestrację, logowanie, uzupełnienie profilu, przejście formularza *onboardingowego*, przeglądanie *marketplace'u* trenerów, wysłanie prośby o współpracę, dostęp do planów treningowych i dietetycznych oraz raportowanie wykonanych treningów. Dla trenera testowano obsługę zgłoszeń, zarządzanie klientami, tworzenie ćwiczeń, szablonów treningowych, planów treningowych, szablonów posiłków i planów dietetycznych. Dla administratora sprawdzano dostęp do panelu administracyjnego, list trenerów, list klientów oraz zarządzania ćwiczeniami globalnymi.

Osobną część testów manualnych stanowiła komunikacja. Sprawdzano wysyłanie próśb o wiadomość, akceptację kontaktu, widok listy konwersacji, wymianę wiadomości między trenerem i klientem oraz oznaczanie wiadomości jako odczytanych. W przypadku plików weryfikowano dodawanie awatarów, przesyłanie zdjęć postępów oraz obsługę materiałów wideo ćwiczeń wraz z późniejszym przetwarzaniem miniatur.

Po wdrożeniach wykonywano również testy dymne związane z bezpieczeństwem i konfiguracją aplikacji. Obejmowały one logowanie i ustawienie ciasteczek sesyjnych, odrzucenie rejestracji ze zbyt słabym hasłem, weryfikację konfiguracji CORS oraz sprawdzenie, czy *endpointy* przeznaczone dla określonej roli nie są dostępne dla innych użytkowników. Dzięki temu testy manualne uzupełniały testy automatyczne w obszarach, które najlepiej ocenia się przez rzeczywiste użycie aplikacji w przeglądarce.

7.6. Testowanie w procesie budowania

Testy automatyczne uruchamiane są jako integralna część procesu budowania obrazów kontenerowych, a nie jako osobny etap *pipeline CI/CD*. Taka decyzja wynikała z kilku czynników. Po pierwsze, konfiguracja osobnego etapu testowego w *pipeline* wymagałaby uruchomienia dodatkowego kontenera z JDK i Maven (serwer) lub Node.js (klient) w ramach poda Jenkins, co przy ograniczonych zasobach środowiska (12 GB RAM współdzielone pomiędzy klaster, Jenkins, budowanie obrazów i działające aplikacje) stanowiło istotne ryzyko przekroczenia limitów pamięci.

Po drugie, w projekcie nie wdrożono platformy analizy statycznej typu SonarQube, która uzasadniałaby wydzielenie osobnego etapu jakościowego - ograniczenia zasobowe uniemożliwiły uruchomienie dodatkowej usługi w klastrze (opisano w podrozdziale 8.6). Po trzecie, uruchomienie testów wewnątrz procesu budowania obrazu zapewnia zasadę, że do rejestru trafiają wyłącznie obrazy zbudowane z przetestowanego kodu -- obraz nie może istnieć bez pomyślnego przejścia testów.

W przypadku serwera *Dockerfile* wykonuje polecenie `./mvnw package`, które w ramach cyklu życia Maven obejmuje fazy kompilacji, uruchomienia testów oraz pakowania artefaktu. Faza testów uruchamia wszystkie klasy testowe opisane w podrozdziałach 7.2-7.4: testy jednostkowe z Mockito, testy wycinkowe warstwy webowej (@WebMvcTest) i persystencji (@DataJpaTest)

oraz testy integracyjne na bazie H2. Niepowodzenie któregośkolwiek testu przerywa budowanie obrazu. Fragment etapu budowania przedstawiono w listingu 22.

Listing 22. Etap budowania w Dockerfile aplikacji serwerowej z uruchomieniem testów.

```
FROM eclipse-temurin:21-jdk AS builder
WORKDIR /app
COPY mvnw .
COPY .mvn .mvn
COPY pom.xml .
RUN ./mvnw dependency:go-offline -B

COPY src src
RUN ./mvnw package -B \
    && java -Djarmode=layertools -jar target/*.jar extract --destination
/app/extracted
```

W przypadku aplikacji klienckiej proces budowania obejmuje dwa etapy weryfikacji: lintowanie kodu przez ESLint (`npm run lint`) oraz kompilację TypeScript z budowaniem paczki produkcyjnej (`npm run build`, obejmujące `tsc -b` i Vite). Błąd lintowania lub błąd typowania przerywa budowanie obrazu analogicznie do niepowodzenia testu w przypadku budowy aplikacji serwerowej. Fragment Dockerfile przedstawiono w listingu 23.

Listing 23. Etap budowania w Dockerfile aplikacji klienckiej z lintowaniem.

```
FROM node:18-alpine AS build
WORKDIR /app
COPY package*.json ./
RUN npm install

COPY . .
RUN npm run lint
RUN npm run build
```

Przyjęte podejście okazało się wystarczające dla potrzeb prototypu. *Pipeline CI/CD* przebiega według schematu: *checkout* kodu, budowanie obrazu przez BuildKit (z uruchomieniem testów i lintowania), skanowanie bezpieczeństwa przez Trivy, wdrożenie na środowisko deweloperskie, ręczna akceptacja, wdrożenie produkcyjne. Brak osobnego etapu testowego upraszcza konfigurację *pipeline*, zmniejsza zużycie zasobów i eliminuje konieczność zarządzania *cache'em* zależności Maven lub NPM na poziomie Jenkins. Jednocześnie podejście to ma ograniczenie: wyniki testów nie są publikowane jako osobny artefakt w interfejsie Jenkinsa, a raport pokrycia JaCoCo (generowany wyłącznie w fazie *verify*, nie *package*) dostępny jest tylko lokalnie na maszynach deweloperskich. Tabela 23 podsumowuje bramki jakościowe na poszczególnych etapach.

Tabela 23. Bramki jakościowe w procesie budowania i wdrażania. Źródło: opracowanie własne.

Etap	Warstwa serwerowa	Warstwa kliencka
Budowanie obrazu Docker	Testy JUnit (jednostkowe, wycinkowe, integracyjne)	ESLint + kompilacja TypeScript
<i>Pipeline</i> - po budowaniu	Skanowanie Trivy (HIGH, CRITICAL)	Skanowanie Trivy (HIGH, CRITICAL)
<i>Pipeline</i> - przed produkcją	Ręczna akceptacja po weryfikacji na środowisku deweloperskim	Ręczna akceptacja po weryfikacji na środowisku deweloperskim

Po wdrożeniu	Sondy Kubernetes (<i>startup, readiness, liveness</i>)	Sondy Kubernetes (<i>startup, readiness, liveness</i>)
--------------	--	--

7.7. Skanowanie bezpieczeństwa obrazów kontenerowych

Elementem weryfikacji jakości wdrożenia jest skanowanie obrazów kontenerowych pod kątem znanych podatności. W *pipeline CI/CD* zintegrowano narzędzie Trivy, uruchamiane po zbudowaniu obrazu, a przed jego wdrożeniem na środowisko. Skan obejmuje podatności o poziomie istotności HIGH oraz CRITICAL, co pozwala wykryć krytyczne luki bezpieczeństwa w zależnościach systemowych i bibliotekach zawartych w obrazie.

Trivy uruchamiany jest jako kontener w ramach poda Jenkins, korzystając z obrazu aquasec/trivy:latest. Baza podatności przechowywana jest na wolumenie PVC (trivy-cache), co eliminuje konieczność pobierania pełnej bazy przy każdym uruchomieniu i skraca czas skanowania. Konfigurację etapu skanowania przedstawiono w listingu 24.

Listing 24. Etap skanowania bezpieczeństwa w *pipeline CI/CD* (security-scan.groovy).

```
def call(Map args) {
    container('trivy') {
        sh """
            trivy image \
                --severity HIGH,CRITICAL \
                --scanners vuln \
                --timeout 20m \
                --no-progress \
                --insecure \
                ${args.registryUrl}/${args.appName}:${args.imageTag}
        """
    }
}
```

Skanowanie dotyczy obrazu opublikowanego w prywatnym rejestrze, co oznacza, że weryfikowana jest dokładnie ta wersja obrazu, która zostanie wdrożona na środowisko. Flaga `--insecure` wynika z wykorzystania wewnętrznego rejestru bez certyfikatu TLS. Flaga `--timeout 20m` uwzględnia scenariusz pierwszego uruchomienia, kiedy baza podatności musi zostać pobrana w całości.

Należy zaznaczyć, że skanowanie Trivy w obecnej konfiguracji nie blokuje wdrożenia automatycznie - wynik skanu prezentowany jest w logach *pipeline*, a decyzja o kontynuowaniu należy do osoby zatwierdzającej wdrożenie na etapie ręcznej akceptacji. W przyszłości możliwe byłoby wprowadzenie polityki automatycznie blokującej wdrożenie w przypadku wykrycia podatności krytycznych poprzez dodanie flagi `--exit-code 1`.

7.8. Health check i weryfikacja wdrożenia

Weryfikacja działania wdrożonej aplikacji opiera się na mechanizmie sond Kubernetes w połączeniu ze Spring Boot Actuator. W konfiguracji aplikacji włączono sondy Actuatora (`management.endpoint.health.probes.enabled: true`), dzięki czemu udostępniane są endpointy `/actuator/health/liveness` oraz `/actuator/health/readiness`.

W praktyce zdecydowano się na uproszczone podejście: wszystkie trzy sondy Kubernetes: `startupProbe`, `readinessProbe` i `livenessProbe` skierowano na tę samą ścieżkę `/actuator/health`. Decyzja podyktowana była charakterem prototypu: aplikacja nie posiada zewnętrznych zależności wymagających osobnej weryfikacji gotowości, a baza danych jest uruchomiona poza klastrem. W takim modelu rozróżnienie pomiędzy sondą żywotności a sondą gotowości nie wnosi dodatkowej wartości diagnostycznej. Konfigurację sond przedstawiono w listingu 25.

Listing 25. Konfiguracja sond.

```
startupProbe:
  httpGet:
    path: {{ .Values.ingress.health }}
    port: {{ .Values.containerPort }}
  initialDelaySeconds: 30
  periodSeconds: 10
  failureThreshold: 30

readinessProbe:
  httpGet:
    path: {{ .Values.ingress.health }}
    port: {{ .Values.containerPort }}
  periodSeconds: 10
  failureThreshold: 3

livenessProbe:
  httpGet:
    path: {{ .Values.ingress.health }}
    port: {{ .Values.containerPort }}
  periodSeconds: 15
  failureThreshold: 3
```

Wartość `ingress.health` definiowana jest w plikach `values-dev.yaml` i `values-prod.yaml` jako `/actuator/health/liveness`. Sonda startowa (`startupProbe`) dopuszcza do 30 prób z interwałem 10 sekund, co daje aplikacji serwerowej do 5 minut na inicjalizację kontekstu Spring. Po pomyślnym uruchomieniu sonda żywotności (`livenessProbe`) odpytuje aplikację co 15 sekund - trzy kolejne niepowodzenia skutkują restartem kontenera przez Kubernetes

Aplikacja kliencka nie posiada skonfigurowanych sond Kubernetes. Obraz Nginx serwujący pliki statyczne nie udostępnia dedykowanego *endpointu health check*, a weryfikacja dostępności odbywa się manualnie po wdrożeniu.

7.9. Pokrycie kodu

Pokrycie kodu serwera mierzone jest przez JaCoCo w wersji 0.8.13. Raport generowany jest w fazie `verify` cyklu Maven i zawiera dane o pokryciu instrukcji, gałęzi, linii, metod i klas. Z pomiaru wyłączono elementy o niskiej wartości diagnostycznej: klasę startową aplikacji (`CoachifyApplication`), obiekty DTO, encje modelu oraz wybrane klasy konfiguracyjne. Dodatkowo skonfigurowano Lomboka do generowania adnotacji `@lombok.Generated`, dzięki czemu kod wygenerowany przez Lomboka jest automatycznie pomijany w raporcie pokrycia.

Minimalny próg pokrycia nie został wymuszony. Decyzję tę uzasadniono etapowym charakterem prac - wprowadzenie progu na zbyt wczesnym etapie mogłoby blokować buildy przed uzupełnieniem testów dla wszystkich modułów. Raport JaCoCo generowany jest lokalnie na maszynach deweloperskich, lecz nie jest publikowany ani weryfikowany przez *pipeline CI/CD*.

Po stronie serwera skonfigurowano również Spotless z formatem Google Java Format, umożliwiający ujednolicenie stylu kodu Java. Narzędzie działa jako profil uruchamiany opcjonalnie, co pozwala stopniowo poprawiać styl bez blokowania głównej ścieżki budowy.

Po stronie aplikacji klienckiej skonfigurowano ESLint z regułami dla TypeScript, React Hooks i React Refresh. Standardowy proces budowania obejmuje sprawdzenie typów TypeScript (`tsc -b`), które pełni rolę statycznej analizy typów. Błąd typowania przerywa budowanie obrazu Docker, co zapewnia, że do rejestru nie trafia obraz ze skompilowaną, ale niepoprawnie typowaną aplikacją.

7.10. Znane ograniczenia i wnioski z testowania

Najważniejszym ograniczeniem strategii testowej jest brak pełnych testów *end-to-end* na poziomie HTTP. Żaden test nie uruchamia całej aplikacji jako serwera i nie wysyła do niej rzeczywistych żądań HTTP. Integracja warstwy filtrów bezpieczeństwa, konfiguracji Spring Security, serializacji, obsługi wyjątków i routingu weryfikowana jest przez połączenie testów wycinkowych oraz testów manualnych.

Drugim ograniczeniem jest brak automatycznych testów WebSocket. Moduł czatu stanowi hybryda REST i WebSocket, jednak autoryzacja połączenia, subskrypcje kolejek użytkownika oraz potwierdzenia odczytu weryfikowane są wyłącznie manualnie.

Trzecim ograniczeniem jest brak automatyzacji przy *pull requestach*. Projekt nie wykorzystuje mechanizmu automatycznego uruchamiania testów przy otwieraniu *pull requestów* na platformie GitHub. *Buildy* inicjowane są ręcznie w interfejsie Jenkinsa. Wynika to z architektury środowiska, Jenkins działa wewnątrz zamkniętego klastra bez bezpośredniego dostępu do *webhooków* GitHub.

Czwartym ograniczeniem jest brak wymuszonego progu pokrycia JaCoCo. Spadek pokrycia testów nie blokuje budowania ani wdrożenia.

8. DevOps i wdrożenie

Niniejszy rozdział opisuje infrastrukturę wdrożeniową systemu Coachify. Elementy odpowiedzialne za uruchamianie i utrzymanie aplikacji, takie jak szablony Helm, *pipeline'y CI/CD*, manifesty Kubernetes, konfiguracje narzędzi monitorujących czy mechanizmy zarządzania sekretami wydzielono do osobnego repozytorium niezależnego od repozytoriów aplikacyjnych. W kolejnych podrozdziałach omówiono konteneryzację aplikacji, orkiestrację, automatyzację wdrożeń, konfigurację środowisk, monitoring oraz napotkane ograniczenia podczas implementacji.

8.1. Konteneryzacja

Obie aplikacje wchodzące w skład systemu: serwer (Spring Boot) oraz klient (React/Vite), zostały skonteneryzowane przy użyciu technologii Docker. Każda z nich zawiera w swoim repozytorium plik `Dockerfile` definiujący proces budowania obrazu. W obu przypadkach zastosowano budowanie wieloetapowe (ang. *multi-stage build*), polegające na rozdzieleniu etapu kompilacji od etapu uruchomienia. W pierwszym etapie kod źródłowy jest kompilowany z użyciem pełnego środowiska deweloperskiego (JDK 21 dla serwera, Node.js 18 dla klienta), a w drugim do wynikowego obrazu kopiowane są jedynie artefakty niezbędne do uruchomienia aplikacji. Rozdzielenie to pozwala ograniczyć obraz uruchomieniowy do artefaktów niezbędnych do działania aplikacji, co stanowi zalecaną praktykę budowania obrazów kontenerowych [41].

W obrazie aplikacji serwerowej zastosowano mechanizm warstw Spring Boot (ang. *layered JAR*). Mechanizm ten dzieli archiwum JAR na cztery osobne warstwy: zależności zewnętrzne, loader Spring Boot, zależności typu `SNAPSHOT` oraz kod aplikacji. Każda warstwa kopiowana jest do obrazu w osobnej instrukcji `COPY`, dzięki czemu Docker może *cache'ować* je niezależnie. Listing 26 przedstawia fragment `Dockerfile` aplikacji serwerowej ilustrujący ekstrakcję warstw oraz konfigurację obrazu uruchomieniowego.

Listing 26. `Dockerfile` aplikacji serwerowej.

```
# BUILD
FROM eclipse-temurin:21-jdk AS builder
WORKDIR /app

COPY mvnw .
COPY .mvn .mvn
COPY pom.xml .
RUN ./mvnw dependency:go-offline -B

COPY src src
RUN ./mvnw package -B \
  && mkdir -p /app/extracted \
  && java -Djarmode=layertools -jar target/*.jar extract --destination
  /app/extracted

# RUN
FROM eclipse-temurin:21-jre-jammy
WORKDIR /app

RUN groupadd --system --gid 1000 app \
  && useradd --system --uid 1000 --gid app --no-create-home --shell
  /usr/sbin/nologin app
USER app

COPY --from=builder --chown=app:app /app/extracted/dependencies/.
COPY --from=builder --chown=app:app /app/extracted/spring-boot-loader/.

```

```
COPY --from=builder --chown=app:app /app/extracted/snapshot-dependencies/./
COPY --from=builder --chown=app:app /app/extracted/application/./

EXPOSE 8080

ENV JAVA_TOOL_OPTIONS="-XX:MaxRAMPercentage=75.0 -XX:+ExitOnOutOfMemoryError"

ENTRYPOINT ["java", "org.springframework.boot.loader.launch.JarLauncher"]
```

Obraz aplikacji klienckiej oparto o serwer Nginx. Etap budowania kompiluje aplikację React z użyciem narzędzia Vite, a wynikowe pliki statyczne kopiowane są do obrazu Nginx. Konfigurację Dockerfile zamieszczono w Dodatku C.

Kluczowym wyzwaniem przy wdrażaniu aplikacji klienckiej był wymóg dynamicznej konfiguracji ścieżek. Aplikacja musiała działać na wielu środowiskach, w tym pod różnymi prefiksami ścieżek, bez konieczności ponownego budowania obrazu dla każdej z nich. Szczegółowe problemy napotkane przy wdrażaniu tego mechanizmu opisano w podrozdziale 9.2. W celu rozwiązania tego problemu zastosowano mechanizm konfiguracji uruchomieniowej (ang. *runtime configuration*). Przy starcie kontenera plik szablonowy `env-config.js.template` jest przetwarzany przez wbudowany w obraz Nginx mechanizm `envsubst`, który podstawia wartości zmiennych środowiskowych (`BASENAME`, `API_URL`, `WS_URL`) do szablonu. Listing 27 przedstawia plik szablonowy konfiguracji uruchomieniowej:

Listing 27. Szablon konfiguracji uruchomieniowej aplikacji klienckiej - `env-config.js.template`.

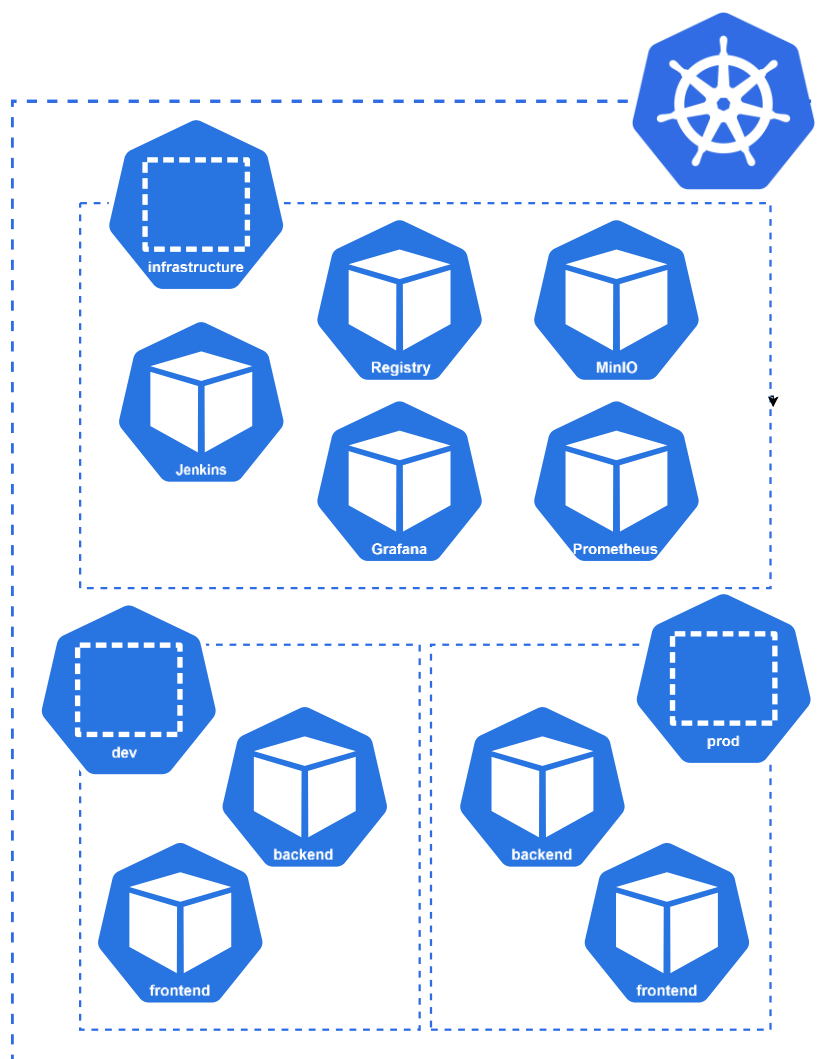
```
window.APP_CONFIG = {
  BASENAME: "${BASENAME}",
  API_URL: "${API_URL}",
  WS_URL: "${WS_URL}"
};
```

Wynikowy plik `env-config.js` ładowany jest przez przeglądarkę przed głównym skryptem aplikacji i udostępnia konfigurację poprzez obiekt `window.APP_CONFIG`. Router aplikacji (React Router) konfigurowany jest dynamicznie z użyciem wartości `BASENAME`, a klient API wykorzystuje `API_URL` do budowania adresów żądań. Konfiguracja Vite definiuje ścieżkę bazową jako względną dla buildu produkcyjnego, co pozwala na poprawne ładowanie zasobów statycznych niezależnie od ścieżki, pod którą aplikacja jest wystawiona. Dzięki temu rozwiązanie realizuje zasadę *build once, deploy many* [48] - jeden obraz budowany jest jednokrotnie, a następnie wdrażany na dowolną liczbę środowisk z odmienną konfiguracją przekazywaną poprzez zmienne środowiskowe.

8.2. Orkiestracja

Środowisko uruchomieniowe oparto na maszynie wirtualnej z systemem Linux, wyposażonej w 4 rdzenie CPU, 12 GB pamięci RAM oraz 86 GB przestrzeni dyskowej. Na maszynie uruchomiono klaster Kubernetes w dystrybucji k3s, działający w trybie jednego węzła (ang. *single-node*).

Infrastrukturę podzielono na trzy przestrzenie nazw (ang. *namespaces*): `dev` (środowisko deweloperskie), `prod` (środowisko produkcyjne) oraz `infrastructure` (narzędzia wspierające). Rysunek 85 przedstawia podział komponentów pomiędzy przestrzeniami nazw.



Rysunek 85. Podział komponentów infrastruktury pomiędzy przestrzeniami nazw Kubernetes. Źródło: opracowanie własne.

W przestrzeni nazw *infrastructure* uruchomiono następujące komponenty:

- Jenkins - platforma *CI/CD* odpowiedzialna za automatyzację wdrożeń (podrozdział 8.3),
- Docker Registry - prywatny rejestr obrazów kontenerowych przechowujący obrazy budowane w *pipeline*'ach,
- MinIO - obiektowe przechowywanie plików, współdzielone między środowiskami *dev* i *prod* z separacją danych poprzez prefiksy zasobników (podrozdział 8.4),
- Prometheus i Grafana - stos monitorujący (rozdział 8.5)

Do automatyzacji tworzenia przestrzeni nazw oraz wolumenów trwałych przygotowano dedykowane *pipeline*'y (*jenkinsfile.namespaces*, *jenkinsfile.pvc*), umożliwiające tworzenie nowych zasobów bez ręcznej interwencji na serwerze. *Pipeline* tworzenia wolumenów przyjmuje nazwę, pojemność, klasę *storage* oraz tryb dostępu jako parametry wejściowe.

Dla zapewnienia trwałości danych oraz optymalizacji wydajności *pipeline*'ów zdefiniowano wolumeny trwałe. Tabela 24 przedstawia zestawienie wolumenów wykorzystywanych w infrastrukturze.

Tabela 24. Wolumeny trwałe w infrastrukturze. Źródło: opracowanie własne.

Nazwa	Pojemność	Przestrzeń nazw	Przeznaczenie
registry-data	10 Gi	infrastructure	Przechowywanie obrazów kontenerowych w prywatnym rejestrze Docker
buildkit-cache	5 Gi	infrastructure	Cache warstw BuildKit montowany w podach pipeline CI/CD
trivy-cache	5 Gi	infrastructure	Baza danych CVE narzędzia Trivy
Jenkins (Helm)	5 Gi	infrastructure	Dane konfiguracyjne Jenkinsa (<i>joby</i> , <i>credentiale</i> , historia)
MinIO (Helm)	5 Gi	infrastructure	Pliki użytkowników (awatary, zdjęcia postępów, materiały wideo)

Dla konta serwisowego Jenkins skonfigurowano mechanizm RBAC (ang. *Role-Based Access Control*), umożliwiający *pipeline*'om wykonywanie operacji w poszczególnych przestrzeniach nazw. Zdefiniowano oddzielne role w przestrzeniach nazw dev, prod oraz infrastructure, przyznając uprawnienia do zarządzania wdrożeniami, sekretami i usługami sieciowymi. Rola w przestrzeni infrastructure ma szerszy zakres, ponieważ *pipeline*'y odpowiedzialne za instalację komponentów infrastrukturalnych wymagają zarządzania zasobami z grup batch oraz rbac.authorization.k8s.io. W tym celu zdefiniowano również rolę klastrową (ang. *ClusterRole*) jenkins-cluster-deploy.

Do zarządzania wdrożeniami aplikacyjnymi zastosowano Helm. W repozytorium infrastrukturalnym zdefiniowano uniwersalny *chart* (app-template), wykorzystywany zarówno przez aplikacje serwerową, jak i kliencką. *Chart* generuje manifesty Deployment, Service, Ingress oraz opcjonalnie Secret na podstawie parametrów przekazywanych w plikach values. Listing 28 przedstawia fragment szablonu Deployment z *chartu* app-template.

Listing 28. Fragment szablonu Deployment - konfiguracja zmiennych środowiskowych.

```
env:
  {{- range .Values.env }}
  - name: {{ .name }}
    value: "{{ .value }}"
  {{- end }}

  {{- range .Values.secretEnv }}
  - name: {{ . }}
    valueFrom:
      secretKeyRef:
        name: {{ $.Values.name }}
        key: {{ . }}
```

```

{{- end }}

{{- if .Values.ingress.health }}
startupProbe:
  httpGet:
    path: {{ .Values.ingress.health }}
    port: {{ .Values.containerPort }}
  initialDelaySeconds: 30
  periodSeconds: 10
  failureThreshold: 30
readinessProbe:
  httpGet:
    path: {{ .Values.ingress.health }}
    port: {{ .Values.containerPort }}
  periodSeconds: 10
  failureThreshold: 3
livenessProbe:
  httpGet:
    path: {{ .Values.ingress.health }}
    port: {{ .Values.containerPort }}
  periodSeconds: 15
  failureThreshold: 3
{{- end }}

```

Szablon konfiguruje zmienne środowiskowe z dwóch źródeł: sekcja `env` zawiera wartości jawne (np. adres bazy danych, profil Spring Boot), natomiast sekcja `secretEnv` generuje referencje do obiektu Kubernetes Secret (np. hasło do bazy danych, klucz JWT). Sondy żywotności i gotowości (ang. *liveness/readiness probes*) konfigurowane są na podstawie ścieżki zdefiniowanej w konfiguracji Ingress. Każdy deployment oznaczany jest etykietą `commit` z SHA wdrażanego *commity*, umożliwiającą identyfikację wdrożonej wersji kodu. Listing 29 przedstawia fragment szablonu Ingress realizującego routing oparty na ścieżkach.

Listing 29. Fragment szablonu Ingress - routing z `pathPrefix`.

```

apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: {{ .Values.name }}-ingress
  annotations:
    {{- if .Values.ingress.path }}
    nginx.ingress.kubernetes.io/rewrite-target: /{{ .Values.ingress.path }}/$2
    {{- else }}
    nginx.ingress.kubernetes.io/rewrite-target: /$2
    {{- end }}
    nginx.ingress.kubernetes.io/use-regex: "true"
spec:
  ingressClassName: {{ .Values.ingress.className }}
  rules:
    - http:
        paths:
          {{- if $segments }}
          - path: "/{{ join "/" $segments }}(/|$)(.*)"
            pathType: ImplementationSpecific
          {{- else }}
          - path: /
            pathType: Prefix
          {{- end }}
        backend:
          service:
            name: {{ .Values.name }}

```

```
port:
  number: {{ .Values.containerPort }}
```

Pole `pathPrefix` definiuje prefiks środowiskowy (np. `dev`), usuwany przez regułę `rewrite`, natomiast pole `path` definiuje prefiks usługi (np. `api`), zachowywany w przekazywanej ścieżce. Dzięki temu aplikacja serwerowa zawsze otrzymuje żądania pod ścieżką `/api`, niezależnie od tego czy dostęp odbywa się przez `/api` (produkcja) czy `/dev/api` (środowisko deweloperskie). Mechanizm ten umożliwia wystawianie wielu środowisk na jednym adresie IP bez modyfikacji kodu aplikacji.

Każda aplikacja przechowuje w swoim repozytorium katalog `devops/` zawierający plik `Jenkinsfile` oraz pliki `values-dev.yaml` oraz `values-prod.yaml` z konfiguracją środowiskową. Tabela 25 przedstawia porównanie wybranych parametrów konfiguracji dla aplikacji serwerowej.

Tabela 25. Porównanie konfiguracji środowisk DEV i PROD dla aplikacji serwerowej. Źródło: opracowanie własne.

Parametr	DEV	PROD
CPU (request / limit)	100m / 500m	200m / 1000m
Pamięć RAM (request / limit)	512 Mi / 768 Mi	512 Mi / 1 Gi
CORS	http://172.21.40.173, http://localhost:5173	http://172.21.40.173
Prefiks zasobników MinIO	dev-coachify-*	prod-coachify-*
Prefiks ścieżki Ingress	/dev/api	/api
Profil Spring Boot	k8s	k8s, prod
Cookie Secure	false	false

8.3. Pipeline CI/CD

Automatyzację wdrożeń (*pipeline*) zrealizowano z użyciem Jenkinsa, uruchomionego wewnątrz klastra Kubernetes w przestrzeni nazw `infrastructure`. Architektura *pipeline*'ów zaprojektowano modularnie - główny plik orkiestrujący (`Jenkinsfile.app`) definiuje przepływ wdrożenia, a poszczególne etapy wydzielono do osobnych plików Groovy, ładowanych dynamicznie w trakcie wykonywania *pipeline*'u. Zapisanie definicji procesu wdrożeniowego w formie wersjonowanego kodu (ang. *Pipeline as Code*) umożliwia ponowne wykorzystanie logiki pomiędzy *pipeline*'ami serwowymi i klienckimi bez duplikowania kodu [47]. Listing 30 przedstawia strukturę katalogów *pipeline*'ów w repozytorium infrastrukturalnym.

Listing 30. Struktura *pipeline*'ów w repozytorium infrastrukturalnym.

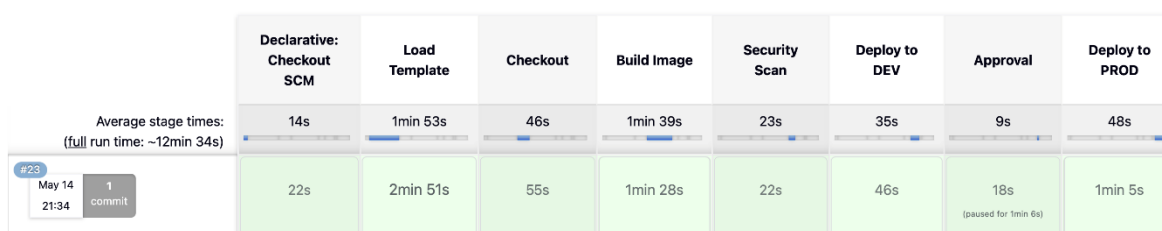
```
infrastructure_coachify/
├── charts/
│   └── app-template/
│       └── templates/
│           ├── deployment.yaml
│           ├── ingress.yaml
│           ├── service.yaml
│           └── secret.yaml
└── pipelines/
```



Każda aplikacja posiada w swoim repozytorium plik Jenkinsfile, który klonuje repozytorium infrastrukturalne i wywołuje szablon *pipeline*'u z odpowiednimi parametrami: nazwą aplikacji, nazwą repozytorium, opcjonalnym tagiem obrazu (na potrzeby *rollbacku*) oraz flagą określającą, czy aplikacja wymaga wstrzykiwania sekretów.

Jenkins wykorzystuje Kubernetes Plugin do dynamicznego tworzenia agentów na żądanie. Każdy agent uruchamiany jest jako *pod* z trzema kontenerami: BuildKit (budowanie obrazów), Trivy (skanowanie CVE) oraz Helm (wdrażanie). *Pod* montuje wolumeny PVC z *cache* BuildKit oraz bazą CVE Trivy. Po zakończeniu *pipeline*'u *pod* jest automatycznie usuwany, co pozwala ograniczyć stałe zużycie zasobów przez agenty.

Pipeline aplikacyjny składa się z sześciu etapów. Rysunek 86 przedstawia przepływ *pipeline*'u od momentu uruchomienia do wdrożenia na środowisko produkcyjne.



Rysunek 86. Przepływ *pipeline*'u CI/CD. Źródło: opracowanie własne.

W etapie Checkout *pipeline* klonuje repozytorium aplikacji i pobiera skrócony SHA *commita* (7 znaków), wykorzystywany jako tag obrazu. Połączenie z GitHub realizowane jest wyłącznie w trybie odczytu za pomocą kluczy SSH przechowywanych w Jenkins Credentials. W etapie Build Image budowany jest obraz kontenerowy z użyciem BuildKit. Pierwotnie do budowania obrazów wykorzystywano narzędzie Kaniko, które zostało zastąpione ze względu na ograniczoną aktywność rozwojową projektu w okresie realizacji pracy oraz niższą wydajność obserwowaną podczas budowania obrazów. BuildKit, wykorzystywany jako nowoczesny mechanizm budowania obrazów kontenerowych, umożliwia efektywniejsze wykorzystanie *cache* oraz równoległe wykonywanie części operacji budowania. *Cache* warstw przechowywany jest pod tagiem *buildcache* i importowany przy kolejnych budowaniach. Zbudowany obraz wypychany jest do prywatnego rejestru Docker działającego wewnątrz klastra.

W etapie *Security Scan* narzędzie Trivy skanuje zbudowany obraz pod kątem znanych podatności o krytyczności HIGH oraz CRITICAL. Baza danych CVE przechowywana jest na wolumenie PVC (2 Gi), co pozwala uniknąć ponownego jej pobierania przy każdym uruchomieniu. *Timeout* skanowania ustawiono na 20 minut - wartość dobrana po napotkaniu problemów z przekroczeniem domyślnego limitu czasu podczas początkowego pobierania bazy.

Etap *Deploy* realizuje wdrożenie za pomocą polecenia `helm upgrade --install` z plikiem `values` specyficznym dla danego środowiska. Listing 31 przedstawia fragment etapu wdrożenia z mechanizmem wstrzykiwania sekretów.

Listing 31. Fragment etapu wdrożenia - wstrzykiwanie sekretów z Jenkins Credentials.

```
def call(Map args) {
    container('helm') {
        def helmCmd = "helm upgrade --install ${args.appName}" +
            " ./infra/charts/app-template" +
            "-n ${args.namespace} --create-namespace" +
            "-f ${args.repoName}/devops/${args.valuesFile}" +
            "--set image.repository=${args.registryUrl}/${args.appName}" +
            "--set image.tag=${args.imageTag}" +
            "--set commitSha=${args.imageTag}"

        if (args.pathPrefix) {
            helmCmd += " --set ingress.pathPrefix=${args.pathPrefix}"
        }

        def credId = "${args.appName}-${args.namespace}"
        withCredentials([file(credentialsId: credId, variable: 'SECRETS')]) {
            sh "${helmCmd} -f \"\$SECRETS\""
        }
    }
}
```

Sekrety (hasła do bazy danych, klucze MinIO, JWT) przechowywane są w Jenkins Credentials jako pliki YAML (typ *Secret file*). Każde środowisko posiada odrębny *credential* identyfikowany konwencją nazewniczą: nazwa aplikacji i nazwa środowiska (np. `backend-dev`, `backend-prod`). Przy wdrożeniu *pipeline* łąduje odpowiedni plik i przekazuje go Helmowi jako dodatkowy plik `values` (`-f $SECRETS`). Helm *chart* na podstawie sekcji `secrets` w pliku `values` tworzy obiekt Kubernetes *Secret*, a *Deployment* referuje poszczególne klucze poprzez `secretKeyRef`. Dzięki temu sekrety nie są przechowywane bezpośrednio w kodzie, a ich aktualizacja nie wymaga logowania na serwer, co jest zgodne z zasadą oddzielania konfiguracji wrażliwej od kodu aplikacji [48].

Pomiędzy wdrożeniem na środowisko deweloperskie a produkcyjne *pipeline* realizuje step *Manual Approval* - manualną akceptację wymagającą zatwierdzenia przez członka zespołu. Etap ten wprowadzono ze względu na brak dedykowanego zespołu testowego - przed zatwierdzeniem wdrożenia na produkcję weryfikowana jest poprawność działania aplikacji na środowisku deweloperskim.

Tagowanie obrazów skróconym SHA *commita* umożliwia deterministyczne wersjonowanie oraz mechanizm *rollback*. Podanie SHA wcześniejszego *commita* jako parametru `IMAGE_TAG` przy uruchomieniu *pipeline*'u powoduje pominięcie etapów budowania i skanowania - *pipeline* wdraża istniejący obraz z rejestru bez ponownego budowania.

Oprócz *pipeline*'ów aplikacyjnych zdefiniowano *pipeline*'y infrastrukturalne, odpowiedzialne za instalację i aktualizację narzędzi wspierających. Każdy komponent infrastruktury posiada dedykowany *pipeline* z własnym plikiem konfiguracyjnym przechowywany w katalogu `pipelines/apps/`. Tabela 26 przedstawia zestawienie *pipeline*'ów infrastrukturalnych.

Tabela 26. *Pipeline'y* infrastrukturalne. Źródło: opracowanie własne.

Komponent	Metoda wdrożenia	Parametry wrażliwe
Docker Registry	kubectl apply (manifest YAML)	Brak
Jenkins	Helm <i>chart</i> jenkins/jenkins	Brak (konfiguracja w values.yaml)
Prometheus	Helm <i>chart</i> prometheus-community/prometheus	Brak
Grafana	Helm <i>chart</i> grafana/grafana	ADMIN_PASSWORD (parametr pipeline)
MinIO	Helm <i>chart</i> minio/minio	ROOT_USER, ROOT_PASSWORD (parametry pipeline)

Pipeline'y MinIO i Grafany przyjmują parametry wrażliwe (hasła administracyjne) przekazywane przy uruchomieniu, dzięki czemu nie są one widoczne w konfiguracji przechowywanej w repozytorium. Przykładowy fragment pliku Jenkinsfile komponentu infrastrukturalnego MinIO umieszczono w Dodatku C.

Wprowadzone optymalizacje - *cache* BuildKit i Trivy na wolumenach PVC, zamiana Kaniko na BuildKit, zmniejszenie limitu pamięci Jenkins z 4 Gi do 2 Gi po analizie rzeczywistego zużycia zasobów pozwoliły skrócić czas pełnego wdrożenia z początkowych 20 minut do około 10 minut.

8.4. Konfiguracja środowisk

System Coachify wdrożono na dwóch środowiskach: deweloperskim (dev) oraz produkcyjnym (prod). Początkowo infrastruktura obejmowała wyłącznie jedno środowisko deweloperskie - w fazie koncepcyjnej projektu aplikacja była rozwijana głównie lokalnie, a infrastruktura serwerowa służyła przede wszystkim do weryfikacji komunikacji pomiędzy komponentami w środowisku kontenerowym.

Separację środowisk wprowadzono w momencie, gdy aplikacja osiągnęła stabilność wystarczającą do utrzymywania wersji produkcyjnej - pojawiła się potrzeba zachowania stabilnej wersji przy jednoczesnej możliwości testowania nowych zmian na dedykowanym środowisku. Oba środowiska działają na tym samym klastrze i tym samym adresie IP. Ich rozdzielenie zrealizowano za pomocą routingu opartego na ścieżkach (ang. *path-based routing*) w kontrolerze Nginx Ingress. Tabela 27 przedstawia mapowanie ścieżek na poszczególne usługi.

Tabela 27. Routing ścieżek w kontrolerze Nginx Ingress. Źródło: opracowanie własne.

Ścieżka	Typ	Usługa docelowa	Przestrzeń nazw
/	Prefiks (<i>fallback</i>)	frontend	prod
/dev(/ \$)(.*)	Regeks	frontend	dev
/api(/ \$)(.*)	Regeks	backend	prod
/dev/api(/ \$)(.*)	Regeks	backend	dev

/infrastructure/jenkins	Prefiks	jenkins	infrastructure
/infrastructure/prometheus	Prefiks	prometheus-server	infrastructure
/infrastructure/grafana	Prefiks	grafana	infrastructure
/infrastructure/minio	Prefiks	minio	infrastructure
/infrastructure/minio-console	Prefiks	minio-console	infrastructure

Kontroler Nginx Ingress łączy reguły ze wszystkich przestrzeni nazw klastra. Ścieżki definiowane wyrażeniami regularnymi mają wyższy priorytet niż ścieżki prefiksowe. Środowisko produkcyjne klienckie celowo skonfigurowano jako Prefix (nie regex), dzięki czemu pełni rolę *fallbacku* - obsługuje żądania, które nie zostały przechwycone przez żadną inną regułę. Rozwiązanie to wdrożono po napotkaniu problemu, w którym produkcyjna reguła regex aplikacji klienckiej przechwytywała żądania przeznaczone dla środowiska deweloperskiego.

Dane w MinIO rozdzielono za pomocą prefiksów w nazwach zasobników: `dev-coachify-avatars`, `dev-coachify-progress`, `dev-coachify-exercises` dla środowiska deweloperskiego oraz odpowiadające im zasobniki z prefiksem `prod` dla produkcyjnego. Sama instancja MinIO jest jedna, uruchomiona w przestrzeni nazw `infrastructure` i współdzielona pomiędzy środowiskami. Analogicznie, serwer PostgreSQL (uruchomiony poza klastrem) zawiera oddzielne bazy danych dla każdego środowiska.

ustawienia produkcyjne nadpisujące konfigurację bazową wartościami o zastrzonym poziomie bezpieczeństwa i wydajności

Konfiguracja aplikacji serwerowej zarządzana jest za pomocą profili Spring Boot:

- `local` - praca deweloperska na maszynach lokalnych, z bezpośrednim połączeniem do lokalnej bazy danych i instancji MinIO;
- `k8s` - wdrożenia na klastrze (oba środowiska), z konfiguracją pobieraną ze zmiennych środowiskowych wstrzykiwanych przez Helm;
- `prod` - ustawienia produkcyjne nadpisujące konfigurację bazową wartościami o zastrzonym poziomie bezpieczeństwa i wydajności.

8.5. Monitoring

W celu zapewnienia obserwowalności infrastruktury i aplikacji wdrożono stos monitorujący oparty o Prometheus i Grafanę. Prometheus gromadzi metryki w modelu cyklicznego pobierania ich z monitorowanych komponentów [49]. Decyzja o wdrożeniu monitoringu była inicjatywą zespołu, wynikającą z potrzeby bieżącej analizy zużycia zasobów klastra oraz diagnozowania problemów wydajnościowych. Przy ograniczonych zasobach maszyny wirtualnej istotna była możliwość szybkiego zidentyfikowania, który komponent zużywa nadmierną ilość pamięci lub CPU.

Prometheus zbiera metryki na dwóch poziomach. Na poziomie infrastrukturalnym - za pośrednictwem komponentów `kube-state-metrics` (stan obiektów Kubernetes: *pod*ów, *deployment*ów, węzłów) oraz `node-exporter` (metryki hosta: CPU, pamięć RAM, dysk) działającego jako `DaemonSet` na każdym węźle klastra. Na poziomie aplikacyjnym za pośrednictwem `endpointu` `/actuator/prometheus` udostępnianego przez aplikację serwerową.

Integrację monitoringu z aplikacją zrealizowano za pomocą biblioteki Micrometer z rejestrem Prometheus oraz modułu Spring Boot Actuator. W konfiguracji Spring Security `endpointy`

/actuator/health i /actuator/info udostępniono bez uwierzytelnienia, natomiast /actuator/prometheus pozostaje dostępny wyłącznie wewnątrz klastra - Prometheus *scrape'uje* go bezpośrednio po nazwie serwisu Kubernetes, bez przechodzenia przez Ingress. Zdefiniowano oddzielne zadania *scrape'ujące* dla środowiska deweloperskiego i produkcyjnego, z etykietą `environment` umożliwiającą filtrowanie w Grafanie.

Tabela 28 przedstawia wybrane metryki aplikacyjne i ich wartość diagnostyczną.

Tabela 28. Wybrane metryki aplikacyjne Spring Boot. Źródło: opracowanie własne.

Metryka	Źródło	Wartość diagnostyczna
http_server_requests_seconds	Micrometer	Czas odpowiedzi HTTP per <i>endpoint</i> , wykrywanie degradacji wydajności
jvm_memory_used_bytes	JVM	Zużycie pamięci heap i non-heap, wykrywanie wycieków pamięci
hikaricp_connections_active	HikariCP	Liczba aktywnych połączeń z bazą danych
hikaricp_connections_pending	HikariCP	Żądania oczekujące na połączenie, sygnał przeciążenia bazy
process_cpu_usage	JVM	Zużycie CPU przez proces aplikacji
logback_events_total	Logback	Liczba logów per poziom, wykrywanie anomalii

Metryki infrastrukturalne informują o tym, że problem występuje (np. wysoki CPU, restart *pod*), natomiast metryki aplikacyjne pozwalają wskazać jego przyczynę i lokalizację.

Retencję danych Prometheus skonfigurowano na 7 dni bez wolumenu trwałego - przy dwóch aplikacjach i ograniczonych zasobach dłuższa retencja nie była uzasadniona. Alertmanager wyłączono z tego samego powodu.

Grafanę skonfigurowano z predefiniowanym źródłem danych wskazującym na instancję Prometheus oraz trzema *dashboardami* ładowanymi automatycznie przy wdrożeniu z repozytorium Grafana. Tabela 29 przedstawia zestawienie *dashboardów*.

Tabela 29. *Dashboardsy* Grafana. Źródło: opracowanie własne.

<i>Dashboard</i>	ID (Grafana)	Zakres
Kubernetes Cluster	6417	Przegląd klastra: zużycie CPU, RAM i dysku per węzeł, status <i>podów</i>
Kubernetes Pods	6336	Szczegóły <i>podów</i> : zużycie zasobów, restarty, status kontenerów

Spring Boot	17175	Metryki aplikacyjne: HTTP, JVM, HikariCP, Tomcat
-------------	-------	--

Dashboardy ładowane są automatycznie za pomocą mechanizmu `dashboardProviders` w konfiguracji Helm Grafany.

Łączny koszt zasobowy stosu monitorującego wynosi około 700 Mi pamięci RAM.

8.6. Ograniczenia

Głównym ograniczeniem infrastruktury okazały się zasoby sprzętowe maszyny wirtualnej. Początkowa konfiguracja maszyny obejmowała 6 GB pamięci RAM, co okazało się niewystarczające - *pipeline'y CI/CD* z narzędziem Kaniko (wykorzystywanym przed migracją na BuildKit) w połączeniu z Jenkinsem kończyły się błędami wynikającymi z braku możliwości alokacji pamięci. Podjęto decyzję o jednorazowym zwiększeniu pamięci do 12 GB, jednocześnie przyjmując strategię optymalizacji zużycia zasobów zamiast dalszego skalowania maszyny.

W ramach optymalizacji przeprowadzono następujące działania:

- Zmniejszono limit pamięci Jenkins z 4 Gi do 2 Gi po analizie rzeczywistego zużycia zasobów;
- Zastąpiono Kaniko narzędziem BuildKit, które efektywniej wykorzystuje pamięć;
- Wdrożono wolumeny PVC dla *cache* BuildKit i bazy CVE Trivy;
- Zrezygnowano z wielu środowisk deweloperskich na rzecz jednego per aplikacja;
- Zoptymalizowano obrazy Docker tak, aby zawierały minimum niezbędnych zależności.

Po wdrożeniu pełnego stosu infrastrukturalnego oraz obu środowisk aplikacyjnych łączne zużycie pamięci RAM wynosi około 8 GB z dostępnych 12 GB. Pozostałe 4 GB stanowią rezerwę na szczyty obciążenia podczas budowania obrazów.

Klaster działa w trybie jednego węzła, co wyklucza autoskalowanie horyzontalne (HPA) oraz zapewnienie wysokiej dostępności (HA). W przypadku awarii węzła cała infrastruktura staje się niedostępna. Baza danych PostgreSQL jest uruchomiona poza klastrem, dlatego nie jest bezpośrednio dotknięta taką awarią, co pozwala zabezpieczyć dane.

Środowisko nie wykorzystuje protokołu TLS. Konsekwencją jest konieczność ustawienia flagi `APP_COOKIE_SECURE=false` - przeglądarki odrzucają ciasteczka z atrybutem `Secure` przesyłane przez nieszyfrowane połączenie.

Opisana infrastruktura wdrożeniowa ma charakter prototypu produkcyjnie zorientowanego - realizuje powtarzalność wdrożeń, podstawową obserwowalność czy kontrolę bezpieczeństwa obrazów, jednak nie zapewnia wysokiej dostępności, izolacji środowisk ani kompletnego modelu bezpieczeństwa sieciowego.

9. Podsumowanie

W niniejszym rozdziale podsumowano rezultaty pracy inżynierskiej nad systemem Coachify. Przedstawiono osiągnięcia projektu, problemy napotkane podczas realizacji, ograniczenia prototypu, funkcje pozostające poza zakresem bieżącej wersji, kierunki dalszego rozwoju, a także wnioski.

9.1. Osiągnięcia

Rezultatem pracy jest prototyp platformy wspierającej współpracę trenerów personalnych z klientami. System obejmuje trzy główne role: klienta, trenera oraz administratora. Każda z ról otrzymała odrębny zakres funkcjonalny, własne widoki interfejsu oraz mechanizmy autoryzacji.

W efekcie przygotowano aplikację, która nie ogranicza się do pojedynczego procesu rejestracji lub komunikacji, lecz obejmuje pełniejszy cykl współpracy: od znalezienia trenera, przez nawiązanie kontaktu, po plany treningowe, plany dietetyczne, raportowanie aktywności i wymianę wiadomości.

Po stronie serwera zrealizowano aplikację Spring Boot opartą na klasycznym podziale warstwowym. Kontrolery REST delegują logikę do serwisów, serwisy zarządzają transakcjami, a repozytoria Spring Data JPA odpowiadają za dostęp do PostgreSQL. Model danych obejmuje użytkowników, profile trenerów i klientów, oferty usług, zgłoszenia współpracy, przypisania klient-trener, plany treningowe i dietetyczne, raporty treningowe, konwersacje czatu, wiadomości oraz pliki multimedialne. Schemat bazy utrzymywany jest migracjami Flyway, a walidacja schematu w trybie produkcyjnym ogranicza ryzyko niekontrolowanego dryfu między encjami a rzeczywistą bazą danych.

Zaimplementowano rozbudowaną warstwę bezpieczeństwa. Uwierzytelnianie oparto na JWT przekazywanym w ciasteczkach `HttpOnly`, `Secure` i `SameSite=Strict`, z osobnym rotowanym tokenem odświeżającym. Hasła są przechowywane z użyciem BCrypt, a *endpoint* logowania chroniony jest limitem prób. Dostęp do funkcji systemu egzekwowany jest przez role oraz adnotacje bezpieczeństwa na poziomie kontrolerów. Dodatkowo wprowadzono jednolity format błędów RFC 7807 `ProblemDetail`, co poprawia spójność komunikacji pomiędzy serwerem a klientem.

Istotnym osiągnięciem jest obsługa współpracy trenera z klientem jako procesu domenowego. Klient może przejść kwestionariusz *onboardingowy*, przeglądać *marketplace* trenerów, wysłać zgłoszenie albo prośbę o wiadomość, a po akceptacji korzystać z przypisanych planów. Trener może zarządzać profilem, ofertą, klientami, planami, szablonami oraz raportami. Administrator otrzymał osobny panel do przeglądania użytkowników i zarządzania globalną biblioteką ćwiczeń.

W aplikacji zaimplementowano także komunikację w czasie rzeczywistym. Czat działa w modelu jeden do jednego pomiędzy trenerem i klientem, korzystając z WebSocket, STOMP oraz mechanizmu kolejek użytkownika. Historia wiadomości i lista konwersacji są dostępne przez REST, natomiast nowe wiadomości oraz potwierdzenia odczytu dostarczane są przez kanał WebSocket. Dzięki temu zachowano prostotę paginowanego pobierania historii oraz responsywność komunikacji na żywo.

Po stronie klienckiej przygotowano aplikację React z TypeScript i Vite. Interfejs obejmuje widoki klienta, trenera, administratora, *marketplace*, kreator *onboardingu*, edytory planów, dziennik treningowy oraz pływający widżet czatu. Zastosowano TanStack Query do obsługi stanu zdalnego, Tailwind CSS do stylowania oraz Radix UI jako bazę dostępnych komponentów interfejsu.

W obszarze plików wykorzystano MinIO jako magazyn obiektowy zgodny z API S3. System obsługuje awatary, zdjęcia postępów i materiały wideo ćwiczeń. Walidacja plików obejmuje typ, rozmiar oraz analizę zawartości, a dostęp do zasobów odbywa się przez czasowo ograniczone adresy *pre-signed* URL. Asynchroniczna obróbka wideo ogranicza czas oczekiwania użytkownika przy operacjach kosztownych obliczeniowo.

Z perspektywy DevOps przygotowano konteneryzację aplikacji serwerowej i klienckiej, konfigurację środowisk, integrację z Kubernetes przez Helm oraz *pipeline* Jenkins oparty na wspólnym szablonie infrastrukturalnym. System udostępnia sondy zdrowia przez Spring Boot Actuator, *endpoint* metryk Prometheus oraz logi strukturalne w środowisku Kubernetes i produkcyjnym. Oznacza to, że prototyp został przygotowany nie tylko jako aplikacja lokalna, lecz także jako system możliwy do uruchomienia w środowisku serwerowym.

9.2. Problemy napotkane podczas realizacji

W trakcie realizacji projektu napotkano szereg problemów technicznych, zarówno po stronie infrastruktury, jak i aplikacji.

Najistotniejszym ograniczeniem okazały się zasoby sprzętowe maszyny wirtualnej. Początkowa konfiguracja (6 GB RAM) nie pozwalała na stabilne wykonywanie *pipeline*ów CI/CD - *builder* obrazów Kaniko w szczytowym obciążeniu (ang. *peak*) przekraczał dostępne zasoby, co skutkowało błędami alokacji pamięci i przerywaniem procesu budowania. Podjęto decyzję o zwiększeniu pamięci do 12 GB oraz migracji na BuildKit, co rozwiązało problem. Dodatkowym wyzwaniem było utrzymywanie obrazów w Docker Registry - przy częstych wdrożeniach rozmiar zgromadzonych obrazów mógłby wyczerpać dostępną przestrzeń dyskową.

Istotnym problemem było wdrożenie routingu klienckiego na wielu środowiskach. Aplikacja typu SPA wymaga, aby serwer przekierowywał wszystkie żądania do pliku `index.html`, natomiast przy różnych prefiksach środowiskowych (`/dev` vs `/`) część ścieżek zwracała błąd 404. Przyczyną okazało się nieprawidłowe indeksowanie ścieżek w konfiguracji Ingress - reguła regeks produkcyjnej aplikacji klienckiej przechwytywała żądania przeznaczone dla środowiska deweloperskiego. Problem rozwiązano zmianą typu reguły z regeks na prefiks, co nadało jej niższy priorytet.

Analogiczny problem wystąpił po stronie serwerowej - reguła `rewrite` w konfiguracji Ingress usuwała prefiks `/api` ze ścieżki, przez co serwer otrzymywał żądania bez wymaganego prefiksu i zwracał błąd 401. Rozwiązaniem było zmodyfikowanie reguły `rewrite` tak, aby zachowywała prefiks usługi (`/api/$2` zamiast `/$2`).

Podczas wdrażania sond żywotności (ang. *liveness probe*) w Kubernetes wystąpił problem z pętlą restartów *pod*ów. Sonda skonfigurowana na ścieżkę `/api/health` była blokowana przez mechanizm Spring Security, który wymagał uwierzytelnienia. *Pod* nie był w stanie potwierdzić swojej żywotności, co powodowało ciągłe restarty. Problem rozwiązano użyciem *endpoint*ów z modułu Spring Boot Actuator.

Problemem było również początkowe podejście do zarządzania schematem bazy danych. W pierwszych iteracjach każde wdrożenie przy większych zmianach wymagało ręcznego przywrócenia bazy danych do stanu początkowego, bez danych testowych (ang. *seed data*). Problem rozwiązano wdrożeniem narzędzia Flyway do zarządzania migracjami schematu.

Wyzwaniem okazała się konfiguracja MinIO do pracy na środowiskach lokalnych członków zespołu. Nie chciano wymagać od każdego członka zespołu instalowania MinIO i konfigurowania połączenia z bazą danych. Zdecydowano się na uruchomienie jednej centralnej instancji MinIO na serwerze, dostępnej ze środowisk lokalnych za pomocą VPN uczelnianego i port forwardingu.

Po stronie aplikacji klienckiej głównym wyzwaniem był brak doświadczenia komercyjnego zespołu w tej dziedzinie przy projekcie o tej skali. Przekładało się to na trudności z utrzymaniem spójności interfejsu oraz na problemy z konfiguracją ciasteczek - atrybut `Secure: true` w połączeniu z protokołem http powodował odrzucanie ciasteczek przez przeglądarkę. Rozwiązaniem było ustawienia flagi `APP_COOKIE_SECURE=false` dla środowisk bez TLS.

W konfiguracji Helm napotkano problem z tagami obrazów składającymi się wyłącznie z cyfr - parser YAML interpretował je jako liczby całkowite, co powodowało błąd przy generowaniu etykiet Kubernetes. Rozwiązaniem było zastosowanie funkcji `quote` w szablonie Helm, wymuszającej traktowanie wartości jako ciągu znaków.

9.3. Ograniczenia prototypu

Prototyp systemu Coachify posiada szereg ograniczeń wynikających z przyjętych kompromisów oraz zakresu pracy. Środowisko wdrożeniowe ograniczone jest do jednej maszyny wirtualnej, co uniemożliwia zapewnienie wysokiej dostępności. Wdrożenia realizowane są na żądanie, poprzez ręczne uruchomienie *pipeline'u*. Zdecydowano się nie konfigurować automatycznego wyzwalania *buildów* z gałęzi domyślnej (ang. *trigger*), aby nie obciążać maszyny ciągłymi *buildami*. Przy uruchomieniu *pipeline'u* pobierany jest kod z wybranego *commity* i wdrażana jest wersja z tego *commity*.

Aplikacja nie jest wystawiona na zewnętrzny serwer ani nie posiada domeny publicznej, jest dostępna wyłącznie pod adresem IP maszyny wirtualnej. Nie skonfigurowano żadnych mechanizmów ochrony ruchu sieciowego (np. TLS, CDN). Środowisko deweloperskie nie jest chronione mechanizmem kontroli dostępu - ograniczono się do wystawienia go pod ścieżką `/dev`, co nie stanowi zabezpieczenia przed nieautoryzowanym dostępem. Kierunki rozwiązania tego ograniczenia opisano w podrozdziale 9.4.

Ograniczeniem jest również funkcjonalność panelu administratora. Rozwój panelu administratora rozpoczęto po zaimplementowaniu modułów trenera i podopiecznego, w związku z czym administrator korzysta z tego samego interfejsu logowania co pozostali użytkownicy. Nie wydzielono osobnego serwera ani panelu logowania dla administratora, co stanowi kompromis z perspektywy bezpieczeństwa.

9.4. Niezrealizowane funkcje i kierunki rozwoju

W trakcie realizacji projektu zidentyfikowano funkcjonalności, których nie udało się wdrożyć w ramach zaplanowanego zakresu pracy. Stanowią one potencjalne kierunki dalszego rozwoju systemu.

Planowano wdrożenie modelu sztucznej inteligencji do generowania propozycji planów treningowych jako funkcjonalności *premium* dla trenerów. W widoku panelu klienta, podczas tworzenia planu treningowego, trener miałby możliwość skorzystania z propozycji wygenerowanej przez AI na podstawie formularza uzupełnionego przez podopiecznego. Rozważano wykorzystanie lokalnego modelu językowego, jednak ograniczyła to waga modelu oraz wymagania zasobowe, przekraczające możliwości dostępnej infrastruktury.

Nie zrealizowano również funkcji automatycznego dopasowania trenera do podopiecznego. Na podstawie uzupełnionego formularza oraz opisu umiejętności i specjalizacji trenerów system miałby sugerować najlepsze dopasowanie. W przyszłości rozważane jest wykorzystanie modelu językowego (LLM) lub dedykowanego modułu dopasowywania opartego na regułach.

Nie wdrożono narzędzia SonarQube do statycznej analizy kodu ze względu na wysokie wymagania zasobowe, które przekraczałyby dostępną rezerwę.

Z perspektywy infrastruktury planowane kierunki rozwoju obejmują:

- Wydzielenie środowiska deweloperskiego na osobny serwer, niedostępny dla użytkowników końcowych;
- Wdrożenie statycznej analizy kodu z użyciem narzędzia SonarQube, zintegrowanego z *pipeline'em CI/CD*;

- Wdrożenie protokołu TLS oraz domeny publicznej;
- Rozbudowę panelu administratora o dodatkowe funkcje zarządzania użytkownikami i grupami oraz wydzielenie go na osobną instancję z odrębnym mechanizmem uwierzytelniania;
- Wdrożenie aplikacji mobilnej;
- Wdrożenie polityki rotacji sekretów oraz wymuszenie minimalnej złożoności haseł serwisowych.

9.5. Wnioski końcowe

W ramach pracy zaprojektowano i zaimplementowano prototyp systemu Coachify, który odpowiada na rzeczywisty problem organizacji współpracy pomiędzy trenerami personalnymi a klientami. Projekt łączy funkcje komunikacyjne, planistyczne, raportowe i administracyjne, a jednocześnie wykorzystuje współczesny stos technologiczny po stronie serwera, klienta oraz infrastruktury.

Najważniejszą wartością projektu jest spójność domenowa. System nie jest zbiorem niezależnych formularzy, lecz odwzorowuje proces współpracy: klient przedstawia cele i ograniczenia, trener przygotowuje plany, klient raportuje wykonanie, a obie strony utrzymują kontakt przez czat. *Marketplace* pełni funkcję punktu wejścia do relacji, natomiast panel administratora wspiera utrzymanie danych wspólnych.

Drugą istotną wartością jest uwzględnienie właściwości niefunkcjonalnych. W projekcie zaimplementowano mechanizmy bezpieczeństwa, walidacji plików, migracji schematu, obsługi błędów, obserwowalności, konteneryzacji i wdrażania.

Podsumowując, cel pracy został zrealizowany. Opracowano platformę, która wspiera kluczowe czynności trenera i klienta, zapewnia podstawy bezpiecznej komunikacji oraz może być uruchamiana w środowisku kontenerowym. Coachify stanowi kompletny prototyp inżynierski oraz punkt wyjścia do dalszego rozwoju produktu.

10. Bibliografia

- [1]. Serwis TrenerHUB, <https://trenerhub.pl>, dostęp: 03.06.2026.
- [2]. Serwis MamTrenera, <https://mamtrenera.pl>, dostęp: 04.06.2026.
- [3]. Serwis TwójTrener, <https://twojtrener.pl>, dostęp: 03.06.2026.
- [4]. FindYourTrainer, <https://www.findyourtrainer.com>, dostęp: 05.06.2026.
- [5]. Trainerize, <https://www.trainerize.com>, dostęp: 04.06.2026.
- [6]. TrueCoach, <https://truecoach.co>, dostęp: 05.06.2026.
- [7]. PT Distinction, <https://ptdistinction.com>, dostęp: 04.06.2026.
- [8]. My PT Hub, <https://www.mypthub.net>, dostęp: 04.06.2026.
- [9]. Everfit, <https://everfit.io>, dostęp: 04.06.2026.
- [10]. CoachPro, <https://systemtreningowy.pl>, dostęp: 05.06.2026.
- [11]. TreningLab, <https://treninglab.pl>, dostęp: 05.06.2026.
- [12]. Fitebo, <https://fitebo.com>, dostęp: 05.06.2026.
- [13]. Hevy, <https://www.hevy.com>, dostęp: 04.06.2026.
- [14]. Strava, <https://www.strava.com>, dostęp: 04.06.2026.
- [15]. Herbert Schildt, Java: The Complete Reference, Ninth Edition, Oracle Press, 2014, ISBN 9780071808569.
- [16]. David J. Eck, Introduction to Programming Using Java, Orange Grove Text Plus, 2006, ISBN 9781616100995.
- [17]. Pat Niemeyer, Jonathan Knudsen, Learning Java, O'Reilly, 2000, ISBN 1565927184.
- [18]. Barry Burd, Java For Dummies, 7th Edition, 2017, ISBN 9781119235552.
- [19]. Mariusz Trzaska, Modelowanie i implementacja systemów informatycznych 2.0, 2025, ISBN 9788397644205.
- [20]. PostgreSQL Documentation, The PostgreSQL Global Development Group, 2026, <https://www.postgresql.org/docs/18/index.html>, dostęp: 10.06.2026.
- [21]. Neil Matthew, Richard Stones, Beginning Databases with PostgreSQL: From Novice to Professional, Apress, 2005, ISBN 9781590594780.
- [22]. Relational Database Concepts, <https://www.egyankosh.ac.in/bitstream/123456789/78764/1/Unit-11.pdf>, dostęp: 09.06.2026.

- [23]. M. Jones, J. Bradley, N. Sakimura, JSON Web Token (JWT), RFC 7519, OAuth Working Group, <https://datatracker.ietf.org/doc/html/rfc7519>, dostęp: 11.06.2026.
- [24]. Sebastián E. Peyrott, The JWT Handbook, Auth0 Inc., 2016-2018, <https://auth0.com/resources/ebooks/jwt-handbook>, dostęp: 11.06.2026.
- [25]. Hafizur Rahman Anik, Security Analysis of JSON Web Tokens - Attack Scenarios and Countermeasures, Aalborg University, 2024, <https://projekter.aau.dk/projekter/files/633758915/JWT.pdf>, dostęp: 11.06.2026.
- [26]. OpenAPI Initiative, OpenAPI Specification, <https://spec.openapis.org/oas/latest.html>, dostęp: 10.05.2026.
- [27]. MinIO Documentation, MinIO Inc., <https://min.io/docs/minio/linux/index.html>, dostęp: 10.05.2026.
- [28]. Chris Minnick, Beginning ReactJS Foundations: Building User Interfaces with ReactJS, Wiley, 2022, ISBN 9781119685548.
- [29]. Alex Banks, Eve Porcello, Learning React, O'Reilly Media, 2017, ISBN 9781491954614.
- [30]. Basarat Ali Syed, TypeScript Deep Dive, Samurai Media Limited, 2017, ISBN 9789888407125.
- [31]. Boris Cherny, Programming TypeScript: Making Your JavaScript Applications Scale, O'Reilly Media, 2019, ISBN 9781492037606.
- [32]. Tailwind CSS Documentation - Styling with utility classes, <https://tailwindcss.com/docs/styling-with-utility-classes>, dostęp: 10.06.2026.
- [33]. Noel Rappin, Modern CSS with Tailwind, Second Edition, Pragmatic Bookshelf, 2022, ISBN 9781680509403.
- [34]. Tailwind CSS Documentation - Installation using Vite, <https://tailwindcss.com/docs/installation/using-vite>, dostęp: 10.06.2026.
- [35]. Tailwind CSS Documentation - Theme, <https://tailwindcss.com/docs/theme>, dostęp: 10.06.2026.
- [36]. Tailwind CSS v4, <https://tailwindcss.com/blog/tailwindcss-v4>, dostęp: 10.06.2026.
- [37]. Radix UI Primitives - Introduction, <https://www.radix-ui.com/primitives/docs/overview/introduction>, dostęp: 10.06.2026.
- [38]. Radix UI Primitives - Accessibility, <https://www.radix-ui.com/primitives/docs/overview/accessibility>, dostęp: 10.06.2026.
- [39]. Spring Framework Documentation - STOMP, <https://docs.spring.io/spring-framework/reference/web/websocket/stomp/enable.html>, dostęp: 10.06.2026.
- [40]. Spring Framework Documentation - Fallback, <https://docs.spring.io/spring-framework/reference/web/websocket/fallback.html>, dostęp: 10.06.2026.
- [41]. Nigel Poulton, Docker Deep Dive, Packt Publishing, 2020, ISBN 9781837028344.

- [42]. BuildKit Documentation, Docker Inc., <https://docs.docker.com/build/buildkit/>, dostęp: 18.05.2026.
- [43]. Kaniko - Build Container Images In Kubernetes, Google Container Tools, <https://github.com/GoogleContainerTools/kaniko>, dostęp: 18.05.2026.
- [44]. Brendan Burns, Joe Beda, Kelsey Hightower, Kubernetes: Up and Running, 2nd Edition, O'Reilly Media, 2019, ISBN 9781492046523.
- [45]. K3s - Lightweight Kubernetes Documentation, <https://docs.k3s.io/>, dostęp: 15.05.2026.
- [46]. Matt Butcher, Matt Farina, Josh Dolitsky, Learning Helm, O'Reilly Media, 2021, ISBN 9781492083603.
- [47]. Brent Laster, Jenkins 2: Up and Running, O'Reilly Media, 2018, ISBN 9781491979549.
- [48]. Jez Humble, David Farley, Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation, Addison-Wesley, 2010, ISBN 9780321601919.
- [49]. Brian Brazil, Prometheus: Up & Running, O'Reilly Media, 2018. ISBN 9781492034148.
- [50]. Grafana Documentation, Grafana Labs, <https://grafana.com/docs/>, dostęp: 14.05.2026.
- [51]. Trivy Documentation, Aqua Security, <https://trivy.dev/latest/docs/>, dostęp: 15.05.2026.
- [52]. Nginx documentation, <https://nginx.org/en/docs/>, dostęp: 20.05.2026.
- [53]. Scott Chacon, Ben Straub, Pro Git, 2nd Edition, Apress, 2014, ISBN 9788328387324.
- [54]. Figma, <https://www.figma.com>, dostęp: 05.06.2026.

11. Spis rysunków

Rysunek 1. Widok wyszukiwania trenerów w serwisie MamTrenera. Źródło: [2].	10
Rysunek 2. Profil trenera w serwisie wyszukiwania. Źródło: [2].	11
Rysunek 3. Strona powitalna komercyjnej platformy <i>SaaS</i> z wezwaniem do rejestracji - bariera przed dostępem do interfejsu. Źródło: [7].	12
Rysunek 4. Strumień aktywności w aplikacji Strava - przykład warstwy społecznościowej i mechanizmów społecznościowych. Źródło: [14].	13
Rysunek 5. Plan treningowy przesłany przez komunikator - zanonimizowany przykład rozproszonego prowadzenia podopiecznego. Źródło: opracowanie własne.	14
Rysunek 6. Model klas aplikacji Coachify. Źródło: opracowanie własne.	31
Rysunek 7. Model klas użytkowników, profili i współpracy. Źródło: opracowanie własne.	32
Rysunek 8. Model klas planów i szablonów treningowych. Źródło: opracowanie własne.	34
Rysunek 9. Model klas planów i szablonów dietetycznych. Źródło: opracowanie własne.	34
Rysunek 10. Model klas raportów treningowych. Źródło: opracowanie własne.	35
Rysunek 11. Logiczna architektura warstwowa systemu Coachify. Źródło: opracowanie własne.	47
Rysunek 12. Sekwencja logowania i rotacji tokenu. Źródło: opracowanie własne.	51
Rysunek 13. Panel trenera. Źródło: opracowanie własne.	71
Rysunek 14. Widok klientów trenera. Źródło: opracowanie własne.	72
Rysunek 15. Zarządzanie prośbami o współpracę. Źródło: opracowanie własne.	72
Rysunek 16. Ustawienia trenera cz. 1. Źródło: opracowanie własne.	73
Rysunek 17. Ustawienia trenera cz. 2. Źródło: opracowanie własne.	73
Rysunek 18. Katalog ćwiczeń - pusty stan. Źródło: opracowanie własne.	74
Rysunek 19. Filtrowanie ćwiczeń według partii mięśniowej. Źródło: opracowanie własne.	75
Rysunek 20. Filtrowanie ćwiczeń według sprzętu. Źródło: opracowanie własne.	75
Rysunek 21. Wypełniony formularz dodawania ćwiczenia. Źródło: opracowanie własne.	76
Rysunek 22. Katalog ćwiczeń po dodaniu pozycji cz. 1. Źródło: opracowanie własne.	76
Rysunek 23. Katalog ćwiczeń po dodaniu pozycji cz. 2. Źródło: opracowanie własne.	77
Rysunek 24. Widok modułu szablonów planów treningowy. Źródło: opracowanie własne.	78
Rysunek 25. Formularz tworzenia nowego szablonu treningowego. Źródło: opracowanie własne.	78
Rysunek 26. Uzupełnianie informacji o szablonie. Źródło: opracowanie własne.	79
Rysunek 27. Wybór ćwiczenia do dnia treningowego. Źródło: opracowanie własne.	79
Rysunek 28. Gotowy szablon treningowy z przypisanym ćwiczeniem. Źródło: opracowanie własne.	80
Rysunek 29. Widok modułu planów treningowych klienta. Źródło: opracowanie własne.	81
Rysunek 30. Tworzenie nowego planu treningowego z wykorzystaniem szablonu. Źródło: opracowanie własne.	81
Rysunek 31. Zmodyfikowany plan treningowy utworzony na podstawie szablonu. Źródło: opracowanie własne.	82
Rysunek 32. Lista planów treningowych przypisanych do klienta z perspektywy trenera. Źródło: opracowanie własne.	82
Rysunek 33. Widok modułu szablonów diet i posiłków. Źródło: opracowanie własne.	83
Rysunek 34. Formularz tworzenia szablonu posiłku. Źródło: opracowanie własne.	83
Rysunek 35. Definiowanie składników posiłku. Źródło: opracowanie własne.	84
Rysunek 36. Lista utworzonych szablonów posiłków. Źródło: opracowanie własne.	84
Rysunek 37. Tworzenie szablonu planu dietetycznego. Źródło: opracowanie własne.	85
Rysunek 38. Tworzenie planu dietetycznego dla klienta z wykorzystaniem szablonu. Źródło: opracowanie własne.	85
Rysunek 39. Lista zaraportowanych treningów podopiecznego w dzienniku treningowym. Źródło: opracowanie własne.	86
Rysunek 40. Widok szczegółowy raportu treningowego - podsumowanie, notatki klienta i zdjęcia postępów. Źródło: opracowanie własne.	87

Rysunek 41. Zestawienie wykonanych ćwiczeń wraz z seriami i porównaniem z planem. Źródło: opracowanie własne.	87
Rysunek 42. Formularz danych osobowych (klient) - krok 1. Źródło: opracowanie własne.	88
Rysunek 43. Formularz danych osobowych (klient) - krok 2. Źródło: opracowanie własne.	89
Rysunek 44. Formularz danych osobowych (klient) - krok 3. Źródło: opracowanie własne.	89
Rysunek 45. Formularz danych osobowych (klient) - krok 4. Źródło: opracowanie własne.	90
Rysunek 46. Formularz danych osobowych (klient) - krok 5 dla wcześniej wybranej opcji „siłownia” lub „na zewnątrz”. Źródło: opracowanie własne.	90
Rysunek 47. Formularz danych osobowych (klient) - krok 5 dla wcześniej wybranej opcji „dom” lub „mała siłownia”. Źródło: opracowanie własne.	91
Rysunek 48. Formularz danych osobowych (klient) - krok 6. Źródło: opracowanie własne.	91
Rysunek 49. Formularz danych osobowych (klient) - krok 7. Źródło: opracowanie własne.	92
Rysunek 50. Formularz danych osobowych (klient) - krok 8. Źródło: opracowanie własne.	92
Rysunek 51. Formularz danych osobowych (klient) - krok 9. Źródło: opracowanie własne.	93
Rysunek 52. Formularz danych osobowych (klient) - krok 10. Źródło: opracowanie własne.	93
Rysunek 53. Formularz danych osobowych (klient) - krok 11. Źródło: opracowanie własne.	94
Rysunek 54. Formularz danych osobowych (klient) - krok 12. Źródło: opracowanie własne.	94
Rysunek 55. Formularz danych osobowych (klient) - krok 13. Źródło: opracowanie własne.	95
Rysunek 56. Formularz danych osobowych (klient) - krok 14. Źródło: opracowanie własne.	96
Rysunek 57. Widok panelu klienta z aktywnym planem dietetycznym. Źródło: opracowanie własne.	96
Rysunek 58. Widok planów treningowych z perspektywy klienta. Źródło: opracowanie własne.	97
Rysunek 59. Kalendarz treningowy klienta. Źródło: opracowanie własne.	97
Rysunek 60. Zakładka „Moje Diety”. Źródło: opracowanie własne.	98
Rysunek 61. Szczegóły planu dietetycznego z perspektywy klienta cz. 1. Źródło: opracowanie własne.	99
Rysunek 62. Szczegóły planu dietetycznego z perspektywy klienta cz. 2. Źródło: opracowanie własne.	99
Rysunek 63. Moduł raportowania treningów i śledzenia postępów klienta cz. 1. Źródło: opracowanie własne.	100
Rysunek 64. Moduł raportowania treningów i śledzenia postępów klienta cz. 2. Źródło: opracowanie własne.	100
Rysunek 65. Moduł raportowania treningów i śledzenia postępów klienta cz. 3. Źródło: opracowanie własne.	101
Rysunek 66. Moduł raportowania treningów i śledzenia postępów klienta cz. 4. Źródło: opracowanie własne.	101
Rysunek 67. Panel główny administratora. Źródło: opracowanie własne.	102
Rysunek 68. Zarządzanie ćwiczeniami globalnymi. Źródło: opracowanie własne.	103
Rysunek 69. Zarządzanie trenerami w systemie. Źródło: opracowanie własne.	103
Rysunek 70. Zarządzanie klientami platformy. Źródło: opracowanie własne.	104
Rysunek 71. Widok główny bez logowania. Źródło: opracowanie własne.	105
Rysunek 72. Panel rejestracji. Źródło: opracowanie własne.	105
Rysunek 73. Panel logowania. Źródło: opracowanie własne.	106
Rysunek 74. <i>Marketplace</i> trenerów personalnych. Źródło: opracowanie własne.	106
Rysunek 75. Profil trenera oraz formularz wysłania prośby o współpracę. Źródło: opracowanie własne.	107
Rysunek 76. Status oczekiwania na odpowiedź trenera. Źródło: opracowanie własne.	108
Rysunek 77. Lista konwersacji w module wiadomości. Źródło: opracowanie własne.	109
Rysunek 78. Widok aktywnej konwersacji między trenerem a klientem. Źródło: opracowanie własne.	109
Rysunek 79. Wysyłanie wiadomości do trenera. Źródło: opracowanie własne.	110
Rysunek 80. Panel wiadomości użytkownika. Źródło: opracowanie własne.	110
Rysunek 81. Lista próśb o wiadomość w widoku trenera. Źródło: opracowanie własne.	111
Rysunek 82. Podgląd prośby o kontakt. Źródło: opracowanie własne.	111

Rysunek 83. Zarządzanie zablokowanymi klientami. Źródło: opracowanie własne.....	112
Rysunek 84. Widok <i>marketplace'u</i> w układzie mobilnym. Źródło: opracowanie własne.	113
Rysunek 85. Podział komponentów infrastruktury pomiędzy przestrzeniami nazw Kubernetes. Źródło: opracowanie własne.	123
Rysunek 86. Przepływ <i>pipeline'u CI/CD</i> . Źródło: opracowanie własne.....	127

12. Spis tabel

Tabela 1. Porównanie z platformami wyszukiwania trenerów. Źródło: opracowanie własne.	14
Tabela 2. Porównanie z narzędziami do prowadzenia klientów. Źródło: opracowanie własne.	15
Tabela 3. Porównanie ze społecznościowymi aplikacjami treningowymi. Źródło: opracowanie własne.	16
Tabela 4. Role użytkowników w systemie Coachify. Źródło: opracowanie własne.	19
Tabela 5. Wymagania funkcjonalne systemu Coachify. Źródło: opracowanie własne.	22
Tabela 6. Wymagania нефункционалне systemu Coachify - bezpieczeństwo i kontrola dostępu. Źródło: opracowanie własne.	25
Tabela 7. Wymagania нефункционалне systemu Coachify - dane i współbieżność. Źródło: opracowanie własne.	25
Tabela 8. Wymagania нефункционалне systemu Coachify - wydajność, pliki i infrastruktura. Źródło: opracowanie własne.	26
Tabela 9. Wymagania нефункционалне systemu Coachify - interfejs użytkownika. Źródło: opracowanie własne.	26
Tabela 10. Aktorzy systemu Coachify. Źródło: opracowanie własne.	27
Tabela 11. Wybrane przypadki użycia systemu Coachify - rejestracja konta. Źródło: opracowanie własne.	27
Tabela 12. Wybrane przypadki użycia systemu Coachify - wypełnianie kwestionariusza. Źródło: opracowanie własne.	27
Tabela 13. Wybrane przypadki użycia systemu Coachify - nawiązanie współpracy. Źródło: opracowanie własne.	28
Tabela 14. Wybrane przypadki użycia systemu Coachify - akceptacja zgłoszenia. Źródło: opracowanie własne.	28
Tabela 15. Wybrane przypadki użycia systemu Coachify - plany i raporty treningowe. Źródło: opracowanie własne.	28
Tabela 16. Wybrane przypadki użycia systemu Coachify - utworzenie raportu. Źródło: opracowanie własne.	28
Tabela 17. Wybrane przypadki użycia systemu Coachify - wysłanie wiadomości. Źródło: opracowanie własne.	29
Tabela 18. Wybrane przypadki użycia systemu Coachify - obsługa prosby wiadomości. Źródło: opracowanie własne.	29
Tabela 19. Wybrane przypadki użycia systemu Coachify - zarządzanie ćwiczeniami globalnymi. Źródło: opracowanie własne.	29
Tabela 20. Główne elementy modelu danych systemu Coachify. Źródło: opracowanie własne.	32
Tabela 21. Słownik pojęć stosowanych w opisie systemu Coachify. Źródło: opracowanie własne.	35
Tabela 22. Role. Źródło: opracowanie własne.	70
Tabela 23. Bramki jakościowe w procesie budowania i wdrażania. Źródło: opracowanie własne.	117
Tabela 24. Wolumeny trwałe w infrastrukturze. Źródło: opracowanie własne.	124
Tabela 25. Porównanie konfiguracji środowisk DEV i PROD dla aplikacji serwerowej. Źródło: opracowanie własne.	126
Tabela 26. <i>Pipeline</i> 'y infrastrukturalne. Źródło: opracowanie własne.	129
Tabela 27. Routing ścieżek w kontrolerze Nginx Ingress. Źródło: opracowanie własne.	129
Tabela 28. Wybrane metryki aplikacyjne Spring Boot. Źródło: opracowanie własne.	131
Tabela 29. <i>Dashboardy</i> Grafana. Źródło: opracowanie własne.	131

13. Spis listingów

Listing 1. Centralizacja ruchu HTTP w kliencie API.....	48
Listing 2. Wspólny kontekst użytkownika dla serwisów domenowych.....	49
Listing 3. Konfiguracja <i>stateless security</i> i publicznych <i>endpointów</i>	52
Listing 4. Odtwarzanie kontekstu bezpieczeństwa z JWT.	53
Listing 5. Rotacja <i>refresh</i> tokenu i wykrywanie <i>replay attack</i>	54
Listing 6. Interceptor odświeżania sesji po stronie aplikacji klienckiej.	54
Listing 7. Akceptacja zgłoszenia i utworzenie relacji trener-klient.	55
Listing 8. Tworzenie planu z walidacją aktywnej współpracy.....	56
Listing 9. Aktualizacja i zakończenie kwestionariusza podopiecznego.....	57
Listing 10. Mapowanie raportu treningowego na strukturę domenową.....	58
Listing 11. Konfiguracja brokera STOMP i <i>endpointu</i> WebSocket.....	59
Listing 12. Obsługa komunikatów STOMP w kontrolerze czatu.....	60
Listing 13. Zapis wiadomości z walidacją dostępu i blokad.	60
Listing 14. Subskrypcje STOMP po stronie aplikacji klienckiej.	62
Listing 15. Specyfikacja JPA filtrująca trenerów <i>marketplace</i>	63
Listing 16. Filtrowanie <i>marketplace</i> w TanStack Query.....	64
Listing 17. Adapter MinIO i generowanie podpisanych URL.	64
Listing 18. Defensywna walidacja typu i rozmiaru pliku.....	65
Listing 19. Załadowanie filmu i uruchomienie asynchronicznej miniatury.	66
Listing 20. Chronione <i>endpointy</i> panelu administratora dla trenerów.	67
Listing 21. Agregaty listy trenerów ładowane zapytaniami grupującymi.....	68
Listing 22. Etap budowania w <i>Dockerfile</i> aplikacji serwerowej z uruchomieniem testów.....	117
Listing 23. Etap budowania w <i>Dockerfile</i> aplikacji klienckiej z lintowaniem.....	117
Listing 24. Etap skanowania bezpieczeństwa w <i>pipeline CI/CD</i> (<i>security-scan.groovy</i>).....	118
Listing 25. Konfiguracja sond.	119
Listing 26. <i>Dockerfile</i> aplikacji serwerowej.....	121
Listing 27. Szablon konfiguracji uruchomieniowej aplikacji klienckiej - <i>env-config.js.template</i>	122
Listing 28. Fragment szablonu <i>Deployment</i> - konfiguracja zmiennych środowiskowych.	124
Listing 29. Fragment szablonu <i>Ingress</i> - routing z <i>pathPrefix</i>	125
Listing 30. Struktura <i>pipeline'ów</i> w repozytorium infrastrukturalnym.....	126
Listing 31. Fragment etapu wdrożenia - wstrzykiwanie sekretów z Jenkins Credentials.	128
Listing 32. Plik <i>Dockerfile</i> aplikacji klienckiej.....	147
Listing 33. Fragment pliku <i>Jenkinsfile</i> komponentu infrastrukturalnego (MinIO).....	148

14. Dodatki

Dodatek A: Słownik skrótów i terminologii

Skrót	Rozwinięcie	Znaczenie
AI	Artificial Intelligence	sztuczna inteligencja
API	Application Programming Interface	interfejs programistyczny aplikacji
CI/CD	Continuous Integration / Continuous Delivery	ciągła integracja i ciągłe dostarczanie
CDN	Content Delivery Network	sieć dostarczania treści
CORS	Cross-Origin Resource Sharing	współdzielenie zasobów między źródłami
CPU	Central Processing Unit	procesor
CRUD	Create, Read, Update, Delete	operacje tworzenia, odczytu, aktualizacji i usuwania danych
CSS	Cascading Style Sheets	kaskadowe arkusze stylów
CVE	Common Vulnerabilities and Exposures	wykaz znanych podatności i zagrożeń
DTO	Data Transfer Object	obiekt transferu danych
HA	High Availability	wysoka dostępność
HPA	Horizontal Pod Autoscaler	poziome autoskalowanie <i>podów</i>
HTTP	Hypertext Transfer Protocol	protokół przesyłania dokumentów hipertekstowych
JDK	Java Development Kit	zestaw narzędzi programistycznych Javy
JPA	Jakarta Persistence API	standard mapowania obiektowo-relacyjnego w Javie
JVM	Java Virtual Machine	wirtualna maszyna Javy
JWE	JSON Web Encryption	standard szyfrowania tokenów JSON
JWS	JSON Web Signature	standard podpisywania tokenów JSON
JSON	JavaScript Object Notation	tekstowy format wymiany danych
JWT	JSON Web Token	token sieciowy w formacie JSON

LLM	Large Language Model	duży model językowy
LTS	Long-Term Support	wsparcie długoterminowe
MDC	Mapped Diagnostic Context	mapowany kontekst diagnostyczny (logowanie)
MVCC	Multi-Version Concurrency Control	wielowersyjna kontrola współbieżności
OAuth	Open Authorization	otwarty standard autoryzacji
ORM	Object-Relational Mapping	mapowanie obiektowo-relacyjne
PVC	Persistent Volume Claim	żądanie woluminu trwałego (Kubernetes)
RAM	Random Access Memory	pamięć operacyjna
RBAC	Role-Based Access Control	kontrola dostępu oparta na rolach
REST	Representational State Transfer	styl architektury usług sieciowych
RFC	Request for Comments	dokument standaryzacyjny
S3	Simple Storage Service	usługa obiektowego przechowywania danych (Amazon)
SaaS	Software as a Service	oprogramowanie jako usługa
SHA	Secure Hash Algorithm	algorytm skrótu kryptograficznego
SPA	Single-Page Application	aplikacja jednostronicowa
STOMP	Simple Text Oriented Messaging Protocol	prosty tekstowy protokół przesyłania wiadomości
SQL	Structured Query Language	strukturalny język zapytań
SSH	Secure Shell	protokół bezpiecznego dostępu zdalnego
SWC	Speedy Web Compiler	szybki kompilator kodu
TLS	Transport Layer Security	protokół zabezpieczania warstwy transportowej
UI	User Interface	interfejs użytkownika
URL	Uniform Resource Locator	ujednolicony adres zasobu
UX	User Experience	doświadczenie użytkownika

WAI-ARIA	Web Accessibility Initiative - Accessible Rich Internet Applications	standard dostępności interfejsów internetowych
YAML	YAML Ain't Markup Language	format serializacji danych konfiguracyjnych

Dodatek B: Zawartość nośnika

System wdrożono na maszynie wirtualnej i udostępniono pod adresem <http://172.21.40.173/>. Dostęp do działającej aplikacji jest możliwy wyłącznie z sieci uczelnianej PJATK.

Na nośniku przekazano kod źródłowy systemu w postaci trzech katalogów:

- katalog infrastruktury - pliki Dockerfile, wykresy Helm, manifesty Kubernetes oraz definicje procesu *CI/CD* (Jenkinsfile);
- katalog aplikacji serwerowej - kod źródłowy w języku Java wraz z konfiguracją *frameworka* Spring Boot, migracjami schematu bazy danych oraz testami;
- katalog aplikacji klienckiej - kod źródłowy w języku TypeScript, komponenty React oraz konfigurację narzędzia Vite.

Dodatek C: Przykładowe elementy konfiguracji

W dodatku zamieszczono wybrane, reprezentatywne fragmenty konfiguracji systemu.

Listing 32 przedstawia plik Dockerfile aplikacji klienckiej.

Listing 32. Plik Dockerfile aplikacji klienckiej.

```
# Build
FROM node:18-alpine AS build
WORKDIR /app

COPY package*.json ./
RUN npm install

COPY . .
RUN npm run build

# Run
FROM nginx:alpine

COPY --from=build /app/dist /usr/share/nginx/html
COPY nginx.conf /etc/nginx/conf.d/default.conf
COPY env-config.js.template /etc/nginx/templates/env-config.js.template

ENV NGINX_ENVSUBST_OUTPUT_DIR=/usr/share/nginx/html
ENV NGINX_ENVSUBST_FILTER="^(BASENAME|API_URL|WS_URL) $"
ENV BASENAME=""
ENV API_URL=""
ENV WS_URL="/ws"

EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

Listing 33 przedstawia fragment pliku Jenkinsfile dla komponentu infrastrukturalnego MinIO

Listing 33. Fragment pliku Jenkinsfile komponentu infrastrukturalnego (MinIO).

```
pipeline {
  agent {
    kubernetes {
      yaml """
apiVersion: v1
kind: Pod
spec:
  containers:
  - name: helm
    image: alpine/helm:3.14.0
    command: ['cat']
    tty: true
      """
    }
  }

  parameters {
    string(name: 'ROOT_USER', defaultValue: '', description: 'MinIO root
username')
    password(name: 'ROOT_PASSWORD', defaultValue: '', description: 'MinIO
root password')
  }

  stages {
    stage('Deploy MinIO') {
      steps {
        container('helm') {
          sh """
            helm upgrade --install minio minio/minio \\\
            -n infrastructure \\\
            --create-namespace \\\
            -f pipelines/apps/minio/values.yaml \\\
            --set rootUser=${params.ROOT_USER} \\\
            --set rootPassword=${params.ROOT_PASSWORD}
          """
        }
      }
    }
  }
}
```