Akademia Finansów i Biznesu Vistula

Wydział Informatyki, Grafiki i Architektury

Kierunek studiów Informatyka

Mateusz Poleski

Numer albumu: *****

ZARZĄDZANIE DRUŻYNĄ PIŁKARSKĄ - APLIKACJA DLA TRENERÓW

Praca inżynierska

napisana pod kierunkiem

Wybierz element.

Warszawa, 2025

Spis treści

Rozdział 1 Wprowadzenie

Rozwój technologii informatycznych oraz rosnące znaczenie danych i ich analiza w zarządzaniu zespołami sportowymi sprawiają, że coraz więcej trenerów i menadżerów drużyn piłkarskich poszukuje narzędzi, które pozwolą na skuteczniejsze zarządzanie ich zespołami. Planowanie treningów, zarządzanie kadrą trenerską oraz kadrą zawodników, planowanie meczów wymagają zastosowania dedykowanych aplikacji, które nie tylko ułatwią podejmowanie decyzji, ale także ułatwiają codzienną pracę trenera oraz jego sztabu.

1.1 Cel pracy

Celem niniejszej pracy inżynierskiej jest stworzenie aplikacji webowej „Zarządzanie Drużyną Piłkarską – Aplikacja dla Trenerów", która umożliwi kompleksowe zarządzanie zespołem piłkarskim. Aplikacja została wykonana przy użyciu frameworka Angular, ma ona udzielać wsparcia w takich rzeczach jak:

-zarządzanie harmonogramem treningów i meczów

-zarządzanie kadrą trenerską oraz kadrą zawodników

-organizowanie kadry meczowej oraz treningów

1.2 Zakres pracy i wybór technologii

Framework Angular został wybrany jako główne narzędzie do budowy aplikacji ze względu na swoje szerokie możliwości tj. tworzenie dynamicznych i interaktywnych aplikacji typu Single Page Application (SPA). Angular, stworzony i rozwijany przez Google, jest jednym z najpopularniejszych narzędzi wykorzystywanych w tworzeniu nowoczesnych aplikacji webowych. Zalety Angulara to m.in. łatwość integracji z backendem, modularność, szeroka społeczność, jak również wsparcie dla rozwiązań mobilnych.

JSON Server wykorzystany w projekcie do implementacji bazy danych i backendu. Umożliwia szybkie prototypowanie aplikacji przy użyciu plików JSON jako bazy danych.

Niniejsza praca nie tylko opisuje proces tworzenia aplikacji, ale także zawiera szczegółową analizę wymagań funkcjonalnych, projekt systemu, wybór technologii oraz omówienie testów i wdrożenia. Dodatkowo w ramach pracy przedstawiono także modele UML, które ilustrują architekturę i przepływ informacji w systemie, co stanowi istotny element dokumentacji technicznej projektu.

Zaprezentowana aplikacja ma na celu zautomatyzowanie wielu procesów, które są wykonywane w tradycyjny ręczny sposób oraz ułatwienie i wsparcie w podejmowaniu decyzji przez trenerów. Aplikacja usprawni zarządzanie na różnych poziomach rozgrywkowych, zarówno amatorskich, jak i profesjonalnych.

1.3 Istniejące rozwiązania

1.3.1 ProTrainUp

Na obecnym rynku IT istnieje kilka rozwiązań, które również pomagają trenerom w jak najlepszym zarządzaniu ich drużyną. Jedną z kilku aplikacji, która to umożliwia jest ProTrainUp. Dzięki niej menedżer zespołu również może dodawać nowych zawodników do kadry drużyny, tworzyć nowe wydarzenia np. treningi, mecze oraz komunikować się z zespołem poprzez wyświetlanie nadchodzących wydarzeń z harmonogramu. Jest to jednak aplikacja o wysokim poziomie zaawansowania a jej duża złożoność może być problematyczna, jeżeli chodzi o kwestię obsługi. Z tego powodu trenerzy z niższych poziomów rozrywkowych oraz osoby starsze mogą mieć problem z wykorzystaniem jej pełnego potencjału.

1.3.2 Coach ISM

Innym popularnym rozwiązaniem jest aplikacja Coach ISM od TrainerPro. Oferująca trenerom możliwość planowania treningów, meczy oraz organizowania kadry zespołu jak również kadry współpracowników. Jest ona rozbudowana o archiwum gier treningowych, w którym można znaleźć konkretne ćwiczenia z dokładnym ich omówieniem. Dodatkowo zawiera możliwość obserwacji zawodników spoza drużyny. W celu oceny, czy dany zawodnik przed transferem do zespołu będzie dla niego wartością dodaną.

Podobnie jak w ProTrainUp wadą jest szeroko rozbudowany interfejs, z którym mniej doświadczeni szkoleniowcy mogą mieć problem. Złożoność systemu może utrudniać szybkie i intuicyjne korzystanie z jego funkcji, zwłaszcza wśród osób starszych lub preferujących prostsze rozwiązania.

1.4 Podsumowanie

Tworzona aplikacja ma na celu usprawnienie pracy trenerów oraz zapewnienie centralnego punktu dostępu do danych o drużynie. Położenie nacisku na intuicyjność i prostotę w obsłudze, ułatwia nawigację nawet użytkownikom o ograniczonych umiejętnościach technicznych. Wszystkie funkcjonalności są łatwo dostępne, a funkcje takiej jak dodawanie zawodników uproszczone. Dzięki temu użytkownicy aplikacji będą mogli efektywnie planować treningi oraz mecze, zarządzać kadrą zespołu i trenerów. Jest to alternatywa dla istniejących już aplikacji charakteryzujących się wysokim stopniem złożoności. Rozwiązanie to ma szansę znaleźć zastosowanie na różnych poziomach rozgrywkowych, co dodatkowo zwiększa jego uniwersalność i wartość praktyczną.

Rozdział 2 Analiza Wymagań

W celu prawidłowego działania aplikacji „Zarządzanie Drużyną Piłkarską – Aplikacja dla Trenerów” niezbędne jest przeprowadzenie dokładnej analizy wymagań funkcjonalnych i niefunkcjonalnych. W tym rozdziale pracy przedstawione zostaną kluczowe aspekty, które mają na celu zagwarantowanie, intuicyjności i funkcjonalności systemu oraz dostosowanie do rzeczywistych potrzeb trenerów.

2.1 Wymagania funkcjonalne

Wymagania funkcjonalne, które aplikacja ma realizować, zapewniają spełnienie głównych celów systemu. Na podstawie analizy potrzeb przyszłych użytkowników, wyodrębniono następujące wymagania funkcjonalne.

- Zarządzanie zawodnikami

  Aplikacja ma za zadanie udostępnić użytkownikowi możliwość dodawania, edytowania i usuwania danych zawodnika na które składają się:

  -Imię i Nazwisko,

  -Pozycja na boisku,

  -Statystyki (zdobyte bramki/czyste konta.),

- Zarządzanie trenerami

  Aplikacja umożliwia dodawanie, edycje i usuwanie trenerów należących do drużyny. Dane które będzie można konfigurować to:

  -Imię i Nazwisko,

  -Rola trenera

- Tworzenie harmonogramu treningów i meczów

  System powinien umożliwiać planowanie wydarzeń takich jak treningi i mecze. Każde wydarzenie musi zawierać następujące informację:

  -Data i godzina,

  -Typ wydarzenia (mecz/trening),

  -Liczba uczestników/ drużyny biorące udział

- Wyświetlanie informacji użytkownikom

  Przejrzystość aplikacji, łatwość zrozumienia systemu i tego do czego ma służyć powinna składać się z danych wyświetlanych w czysty i klarowny sposób. Użytkownik ma możliwość zdobycia informacji na temat:

  -Zaplanowanych treningów

  -Zaplanowanych meczów

  -Kadry zawodników

  -Kadry trenerów

2.2 Wymagania niefunkcjonalne

Wymagania niefunkcjonalne dotyczą cech jakościowych systemu takich jak wydajność, bezpieczeństwo i użyteczność

- Wydajność

  Aplikacja powinna obsługiwać do 200 użytkowników jednocześnie, zachowując przy tym płynność działania.

- Dostępność i niezawodność

  Działanie w trybie 24/7 i dostępność z urządzeń takich jak laptop/komputer stacjonarny.

- Bezpieczeństwo

  Zapewnienie bezpieczeństwa danych użytkownika, w szczególności danych osobowych oraz informacji o drużynach.

- Skalowalność

  System musi być skalowalny, aby w przyszłości mógł obsługiwać więcej użytkowników, drużyn bez utraty wydajności. Architektura systemu powinna umożliwiać łatwą rozbudowę o nowe moduły i funkcjonalności.

2.3 Analiza interesariuszy

Interesariusze to osoby lub grupy, które będą miały wpływ na rozwój aplikacji lub będą jej bezpośrednimi użytkownikami. W tym projekcie wyróżniają się następujące grupy interesariuszy:

- Trenerzy drużyn piłkarskich

  Głowni użytkownicy aplikacji, którzy będą korzystać z niej do zarządzania harmonogramem oraz zarządzania drużyną. To dla tej grupy aplikacja musi być intuicyjna o prosta w obsłudze, ale jednocześnie funkcjonalna.

- Użytkownicy

  Pośredni użytkownicy aplikacji, którzy mogą otrzymywać informację dotyczące kadry zawodników, trenerów, harmonogramu treningów oraz planowanych meczów.

- Administratorzy systemu

  Osoby odpowiedzialne za utrzymanie i monitorowanie aplikacji, zarządzanie danymi oraz bezpieczeństwem systemu. Ich zadaniem będzie zapewnienie stabilności działania aplikacji i dbanie o bezpieczeństwo danych.

2.4 Przypadki użycia (Use-cases)

Na podstawie wymagań funkcjonalnych i analizy interesariuszy opracowano następujące przypadki użycia, które ilustrują, jak różne grupy użytkowników będą korzystać z aplikacji:

1. Rejestracja nowego użytkownika
    a) Cel: Umożliwienie nowemu użytkownikowi założenia konta w aplikacji.
    b) Aktorzy:
        a. Użytkownik
        b. System
        c. Administrator (aktywuje konto)
    c) Warunki początkowe:
        a. Użytkownik ma dostęp do formularza rejestracyjnego.
        b. System jest dostępny i działa poprawnie.
    d) Warunki końcowe:
        a. Konto użytkownika zostaje pomyślnie zarejestrowane i czeka na aktywację przez administratora.
        b. W przypadku błędów system wyświetla komunikat o problemie.
    e) Scenariusz główny:
        a. Użytkownik otwiera ekran logowania.
        b. Przechodzi do formularza rejestracyjnego.
        c. Użytkownik wprowadza wymagane dane:
            i. Login.
            ii. Adres e-mail.
            iii. Hasło.
        d. Użytkownik zatwierdza formularz przyciskiem „Zarejestruj”
        e. System weryfikuje poprawność wprowadzonych danych.
        f. Pozytywna weryfikacja:
            i. System zapisuje dane nowego użytkownika w bazie danych.
            ii. Konto użytkownika zostaje oznaczone jako „oczekujące na aktywację”.
            iii. System wyświetla komunikat: „Konto zarejestrowane, skontaktuj się z administratorem”
        g. Administrator loguje się do panelu administracyjnego, aktywuje konto użytkownika i informuje go o zmianie statusu konta.

f) Alternatywne scenariusze:
    a. Brak wymaganych danych:
        i. System wyświetla komunikat o brakujących danych i zaznacza brakujące pola w formularzu.
        ii. Użytkownik uzupełnia dane i ponownie klika „Zarejestruj”.
    b. Problemy techniczne:
        i. System nie może zapisać danych (np. brak połączenia z bazą danych).

2. Logowanie użytkownika
    g) Cel: Umożliwienie użytkownikowi dostępu do aplikacji po poprawnej walidacji danych.
    h) Aktorzy: Użytkownik
    i) Warunki początkowe:
        a. Użytkownik ma aktywne konto w systemie.
        b. Aplikacja jest dostępna
    j) Warunki końcowe:
        a. Użytkownik zostaje zalogowany, następuje przekierowanie do odpowiedniego panelu aplikacji.
        b. W przypadku błędnego logowania użytkownik otrzymuje komunikat o błędzie.
    k) Scenariusz główny:
        a. Użytkownik otwiera ekran logowania.
        b. Wprowadza login i hasło.
        c. System weryfikuje dane:
            i. Sprawdza, czy konto istnieje.
            ii. Sprawdza poprawność loginu i hasła.
            iii. Sprawdza, czy konto jest aktywne.
        d. Walidacja przebiegła pomyślnie:
            i. System identyfikuje role użytkownik/administrator.
            ii. Wyświetlenie odpowiedniego panelu
        e. Walidacja zakończona niepowodzeniem:
            i. System przekazuje użytkownikowi informacje o błędnym loginie lub haśle
            ii. Użytkownik ma możliwość ponownej próby logowania
    l) Alternatywne scenariusze:

a. Nieudane logowanie: System wyświetla komunikat o błędzie i daje możliwość ponownego logowania.
b. Nowy użytkownik: Użytkownik ma możliwość wyboru opcji rejestracji która kieruje do formularza rejestracyjnego.
c. Nieaktywne konto: System informuje użytkownika, że konto nie zostało aktywowane.
d. Problemy techniczne: Jeżeli system nie może nawiązać połączenia z serwerem, wyświetla stosowny komunikat.

3. Dodawanie nowego zawodnika
   a) Cel: Dodanie nowego zawodnika do kadry zespołu.
   b) Aktorzy:
      a. Użytkownik (z uprawnieniami administratora)
      b. System zarządzania drużyną
   c) Warunki początkowe:
      a. Administrator jest zalogowany do aplikacji.
      b. System jest dostępny i działa poprawnie.
   d) Warunki końcowe:
      a. Nowy zawodnik zostaje dodany do kadry drużyny, a jego dane są dostępne w sekcji zarządzania zespołem. Dostępne opcje w tej sekcji to edycja lub usuwanie danego zawodnika.
      b. System nie zapisuje danych zawodnika, zwraca komunikat o błędzie np. braku wymaganych danych.
   e) Scenariusz główny:
      a. Użytkownik loguje się do aplikacji.
      b. Przechodzi do sekcji zarządzania zawodnikami.
      c. System wyświetla aktualną kadrę zespołu oraz formularz umożliwiający dodanie nowego zawodnika.
      d. Użytkownik wprowadza niezbędne dane do formularza takie jak:
         i. Imię i nazwisko.
         ii. Pozycja na boisku.
      e. Zatwierdzenie formularza przyciskiem „Dodaj zawodnika”.
      f. System weryfikuje poprawność wprowadzonych danych.
      g. Pozytywna weryfikacja:
         i. System zapisuje dane zawodnika w bazie danych.
         ii. Zawodnik pojawia się w kadrze drużyny.
      h. Negatywna weryfikacja:

i. System wyświetla komunikat o błędzie

f) Alternatywne scenariusze:
   a. Brak wymaganych danych:
      i. System wyświetla komunikat o brakujących danych, zaznacza odpowiednie pola w formularzu.
      ii. Użytkownik uzupełnia brakujące dane i ponownie klika „Dodaj zawodnika”.
   b. Problemy techniczne:
      i. System nie może zapisać danych nowego zawodnika (brak połączenia z bazą danych).

4. Dodanie nowego trenera
   a) Cel: Dodanie nowego trenera do kadry trenerskiej.
   b) Aktorzy:
      a. Użytkownik (z uprawnieniami administratora)
      b. System zarządzania drużyną
   c) Warunki początkowe:
      a. Administrator jest zalogowany do aplikacji.
      b. System jest dostępny i działa poprawnie.
   d) Warunki końcowe:
      a. Nowy trener zostaje dodany do kadry trenerskiej, a jego dane są dostępne w sekcji zarządzania trenerami. Dostępne opcje w tej sekcji to edycja lub usuwanie danego trenera.
      b. System nie zapisuje danych trenera, zwraca komunikat o błędzie np. braku wymaganych danych.
   e) Scenariusz główny:
      a. Użytkownik loguje się do aplikacji.
      b. Przechodzi do sekcji zarządzania trenerami.
      c. System wyświetla aktualną kadrę trenerską oraz formularz umożliwiający dodanie nowego trenera.
      d. Użytkownik wprowadza niezbędne dane do formularza takie jak:
         i. Imię i nazwisko.
         ii. Rola.
      e. Zatwierdzenie formularza przyciskiem „Dodaj trenera”.
      f. System weryfikuje poprawność wprowadzonych danych.

g. Pozytywna weryfikacja:
    i. System zapisuje dane trenera w bazie danych.
    ii. Trener pojawia się w kadrze trenerskiej.
h. Negatywna weryfikacja:
    i. System wyświetla komunikat o błędzie.

f) Alternatywne scenariusze:
a. Brak wymaganych danych:
    i. System wyświetla komunikat o brakujących danych, zaznacza odpowiednie pola w formularzu.
    ii. Użytkownik uzupełnia brakujące dane i ponownie klika „Dodaj trenera”.
b. Problemy techniczne:
    i. System nie może zapisać danych nowego trenera (brak połączenia z bazą danych).

5. Planowanie meczu
a) Cel: Zaplanowanie nowego meczu i dodanie go do harmonogramu drużyny
b) Aktorzy:
a. Użytkownik z uprawnieniami administratora (trener)
b. System zarządzania drużyną
c) Warunki początkowe:
a. Administrator jest zalogowany do aplikacji.
b. System jest dostępny i działa poprawnie.
d) Warunki końcowe:
a. Nowy mecz zostaje dodany do harmonogramu drużyny, tam można zobaczyć datę i czas zaplanowanego spotkania oraz zespoły biorące udział.
b. System nie zapisuje danych i zwraca komunikat o błędzie np. braku wymaganych danych.
e) Scenariusz główny:
a. Użytkownik loguje się do aplikacji.
b. Przechodzi do sekcji mecze.
c. System wyświetla formularz dodawania meczu.

d. Użytkownik wprowadza niezbędne dane do formularza takie jak:
    i. Drużyna gospodarzy.
    ii. Drużyna gości.
    iii. Data i czas
e. Zatwierdzenie formularza przyciskiem „Dodaj mecz”.
f. System weryfikuje poprawność wprowadzonych danych.
g. Pozytywna weryfikacja:
    i. System zapisuje dane meczu w bazie danych.
    ii. Mecz pojawia się w harmonogramie.
    iii. System wyświetla komunikat „Mecz został utworzony”
h. Negatywna weryfikacja:
    i. System wyświetla komunikat o błędzie

f) Alternatywne scenariusze:
    a. Brak wymaganych danych:
        i. System wyświetla komunikat o brakujących danych, zaznacza odpowiednie pola w formularzu.
        ii. Użytkownik uzupełnia brakujące dane i ponownie klika „Dodaj mecz”.
    b. Problemy techniczne:
        i. System nie może zapisać danych nowego meczu (brak połączenia z bazą danych).

6. Planowanie treningu
    a) Cel: Zaplanowanie nowego treningu i dodanie go do harmonogramu drużyny
    b) Aktorzy:
        a. Użytkownik z uprawnieniami administratora (trener)
        b. System zarządzania drużyną
    c) Warunki początkowe:
        a. Administrator jest zalogowany do aplikacji.
        b. System jest dostępny i działa poprawnie.
    d) Warunki końcowe:
        a. Nowy trening zostaje dodany do harmonogramu drużyny, tam można zobaczyć datę i czas zaplanowanego wydarzenia oraz zawodników biorących udział.
        b. System nie zapisuje danych i zwraca komunikat o błędzie np. braku wymaganych danych.

e) Scenariusz główny:
    a. Użytkownik loguje się do aplikacji.
    b. Przechodzi do sekcji treningi.
    c. System wyświetla formularz dodawania treningu.
    d. Użytkownik wprowadza niezbędne dane do formularza takie jak:
        i. Data i czas.
        ii. Ćwiczenie.
        iii. Zawodnicy, którzy wezmą udział.
    e. Zatwierdzenie formularza przyciskiem „Dodaj trening”.
    f. System weryfikuje poprawność wprowadzonych danych.
    g. Pozytywna weryfikacja:
        i. System zapisuje dane treningu w bazie danych.
        ii. Trening pojawia się w harmonogramie.
        iii. System wyświetla komunikat „Trening został utworzony”
    h. Negatywna weryfikacja:
        i. System wyświetla komunikat o błędzie.
f) Alternatywne scenariusze:
    a. Brak wymaganych danych:
        i. System wyświetla komunikat o brakujących danych, zaznacza odpowiednie pola w formularzu.
        ii. Użytkownik uzupełnia brakujące dane i ponownie klika „Dodaj trening”.
    b. Problemy techniczne:
        i. System nie może zapisać danych nowego treningu (brak połączenia z bazą danych).

7. Powiadamianie użytkowników o nadchodzących wydarzeniach
    a) Cel: Zapewnienie użytkownikowi szybkiego dostępu do informacji o nadchodzących wydarzeniach.
    b) Aktorzy:
        a. Użytkownik
        b. System zarządzania drużyną
    c) Warunki początkowe:
        a. Użytkownik jest zalogowany do aplikacji.
        b. Harmonogram zawiera zaplanowane wydarzenia.
        c. System jest dostępny i działa poprawnie.

d) Warunki końcowe:
    a. Użytkownik widzi listę najbliższych nadchodzących wydarzeń z dokładną ich datą.
    b. W przypadku braku wydarzeń system wyświetla komunikat o braku zaplanowanych wydarzeń.
e) Scenariusz główny:
    a. Użytkownik loguje się do aplikacji.
    b. System przekierowuje użytkownika w zależności od jego roli.
    c. System pobiera z harmonogramu dane najbliższych wydarzeń.
    d. System wyświetla najbliższy trening i mecz zawierając:
        i. Data i czas.
        ii. Typ (trening/mecz).
        iii. Szczegóły wydarzenia (np. drużyny biorące udział).
    e. Użytkownik może zapoznać się z listą nadchodzących wydarzeń.
f) Alternatywne scenariusze:
    a. Brak nadchodzących wydarzeń:
        i. System informuje użytkownika o braku zaplanowanych wydarzeń.
    b. Problemy techniczne:
        i. System nie może pobrać danych z harmonogramu (brak połączenia z bazą danych).

Każdy przypadek użycia zostanie przedstawiony za pomocą diagramu, aby dokładnie zobrazować interakcje użytkownika z systemem.

2.5 Diagramy poszczególnych przypadków użycia

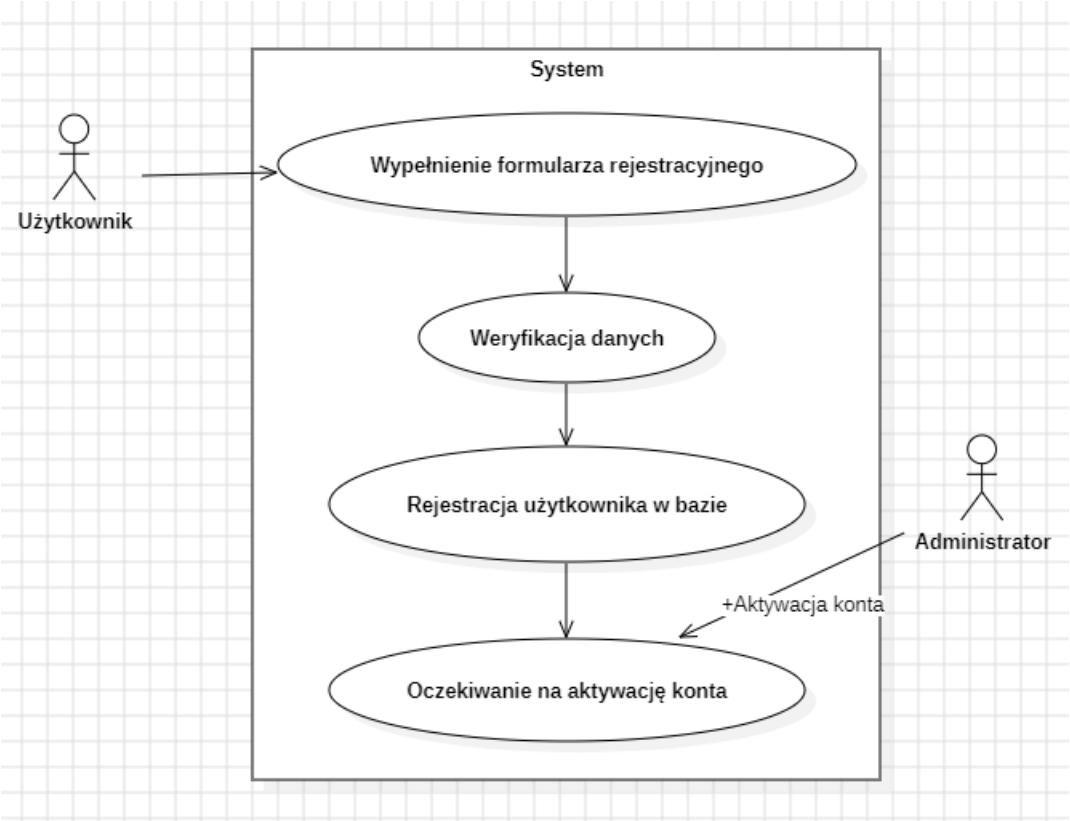


Diagram 1 Rejestracja nowego użytkownika - opracowanie własne

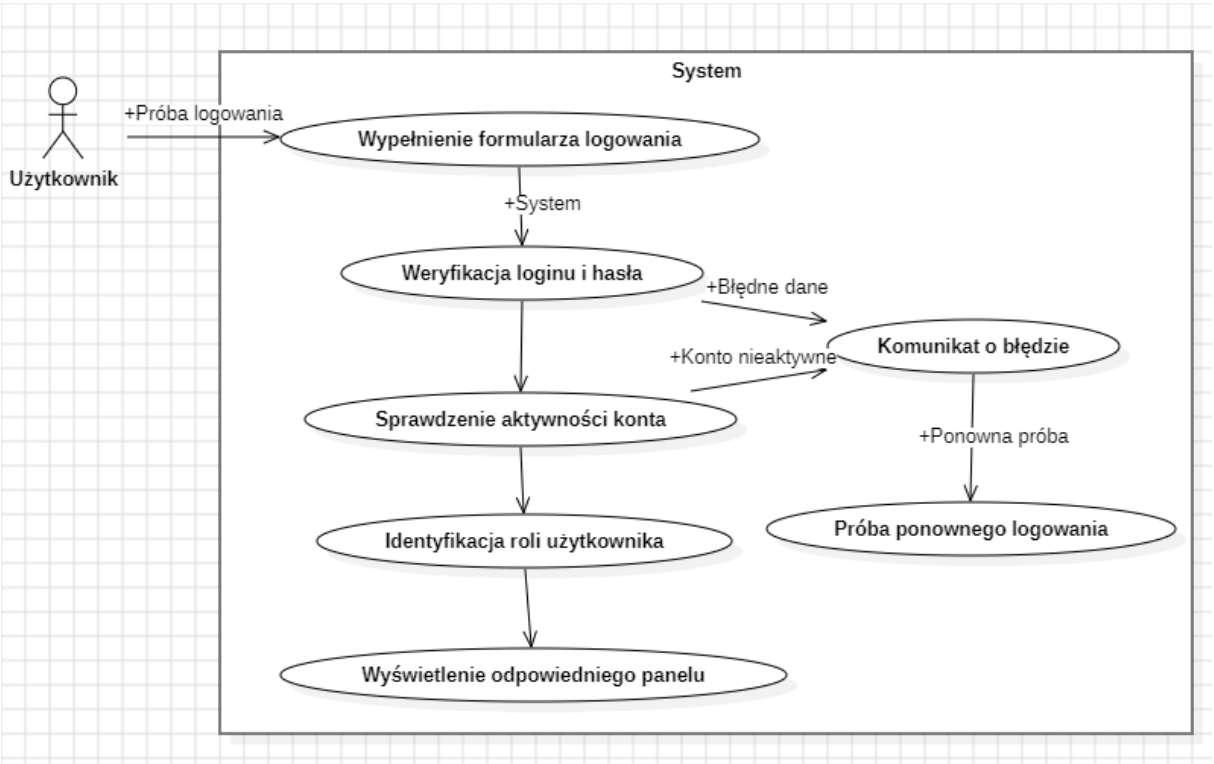


Diagram 2 Logowanie - opracowanie własne

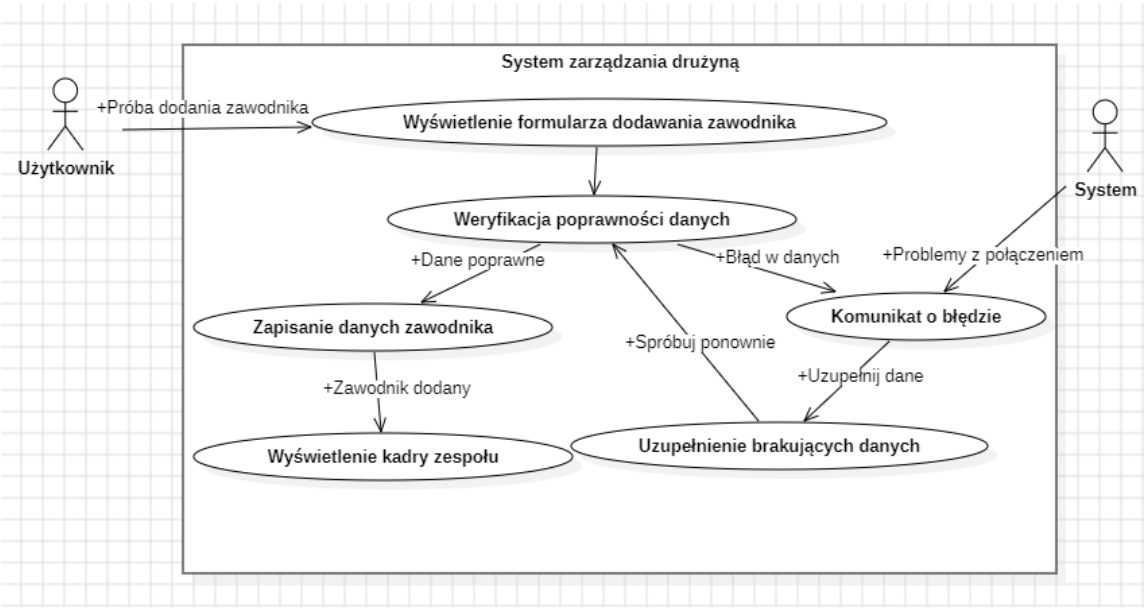


Diagram 3 Dodanie nowego zawodnika - opracowanie własne

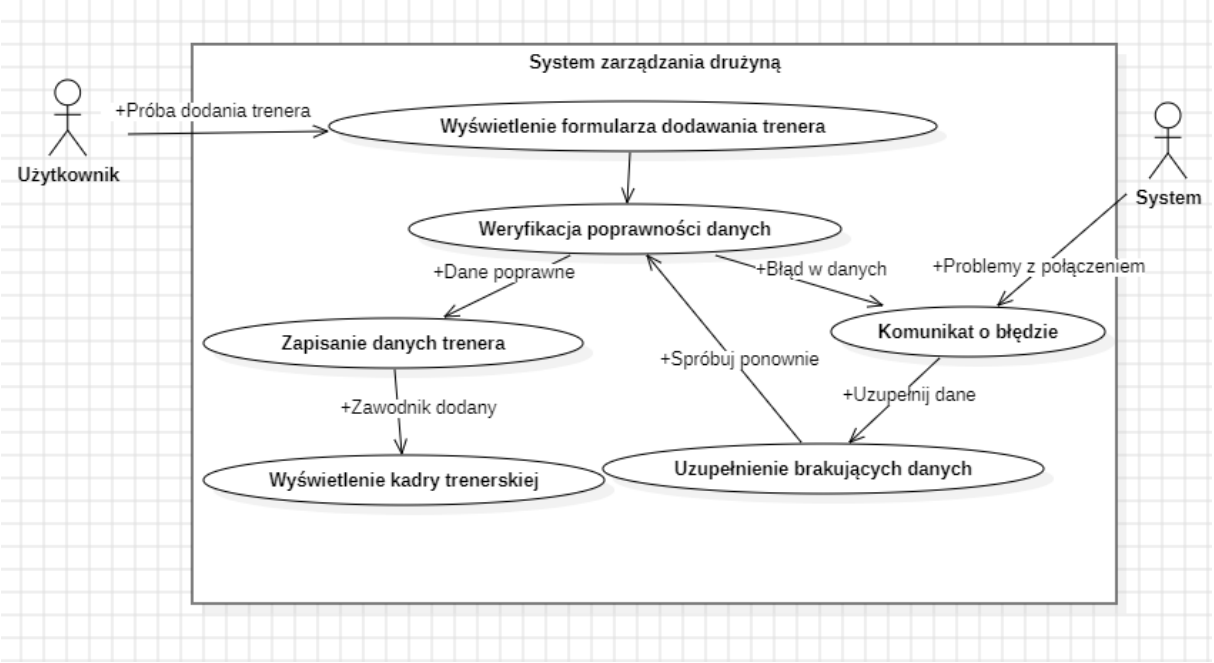


Diagram 4 Dodanie nowego trenera - opracowanie własne

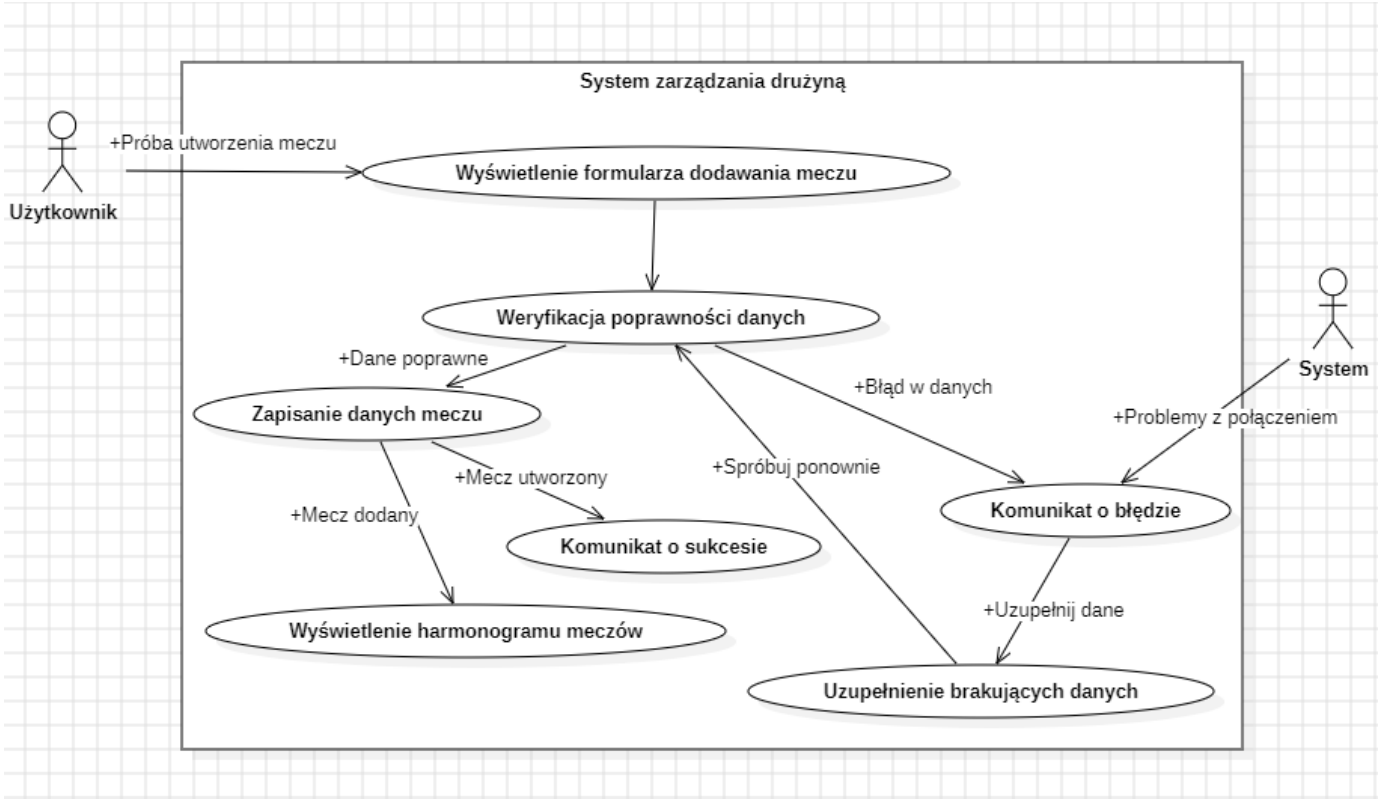


Diagram 5 Dodawanie meczu - opracowanie własne

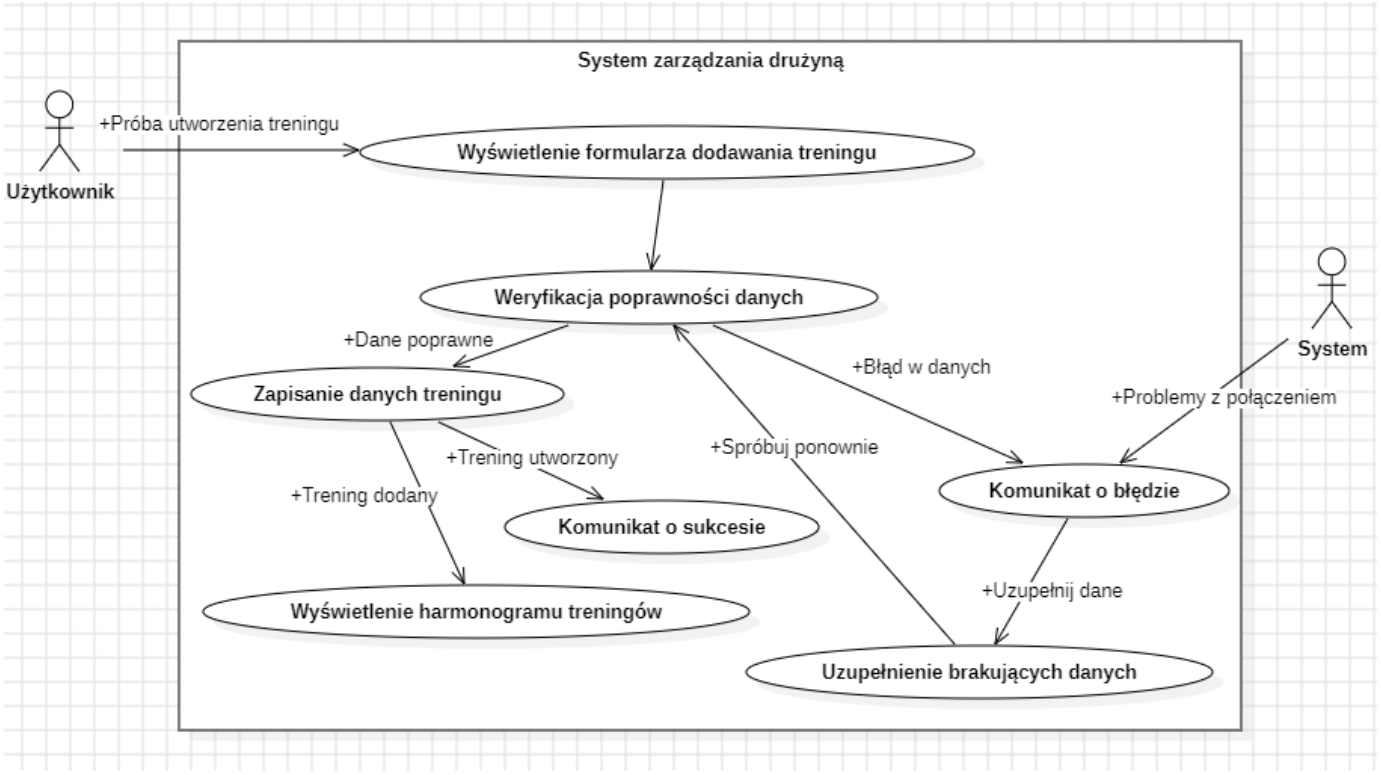


Diagram 6 Dodawanie treningu - opracowanie własne

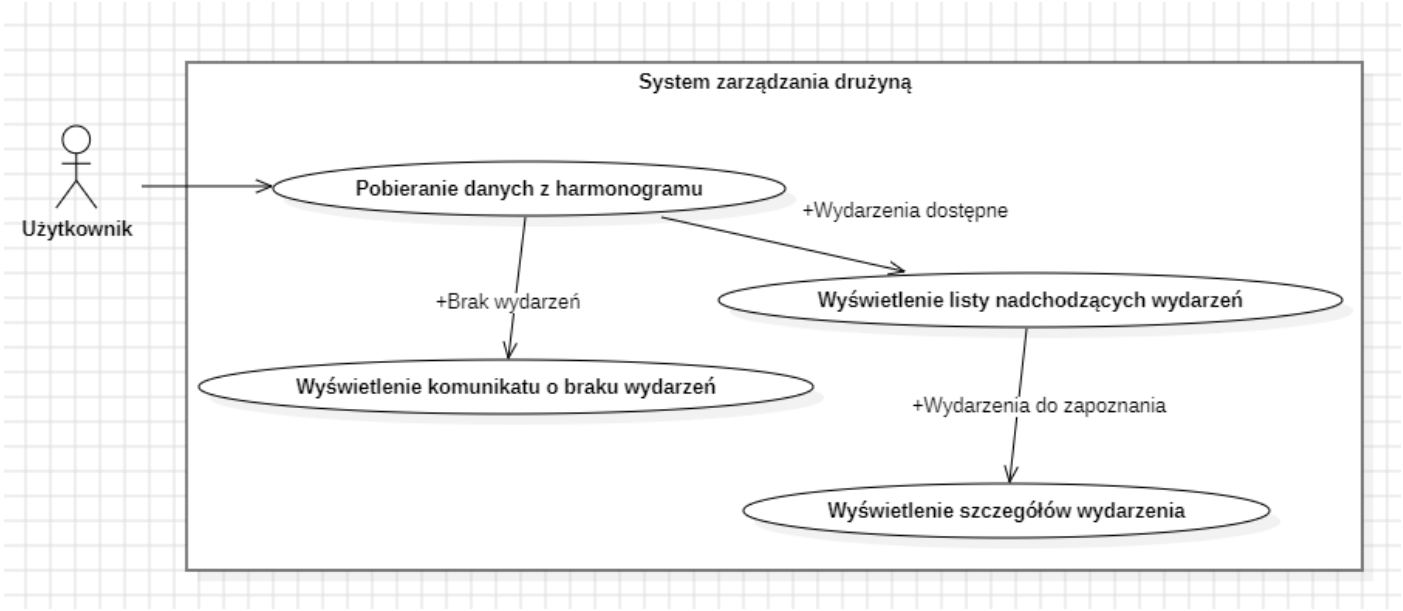


Diagram 7 Powiadomienia o wydarzeniach - opracowanie własne

Rozdział 3 Projektowanie systemu

3.1 Architektura systemu

Aplikacja została zaprojektowana w oparciu o architekturę klient-serwer. Pozwala to na rozdzielenie logiki biznesowej od interfejsu użytkownika. Główne komponenty systemu to:

Frontend, zbudowany przy pomocy frameworka Angular 17, który jest odpowiedzialny za obsługę interfejsu użytkownika i komunikację z backendem.

Backend lokalny serwer plików obsługujący plik db.json, pełni role bazy danych. Komunikacja z bazą odbywa się przy pomocy protokołu HTTP.

Baza danych, przechowywana w formacie JSON w pliku db.json. Jego struktura danych umożliwia łatwe przechowywanie i manipulowanie w małych i średnich aplikacjach.

Architektura systemu składa się z trzech głównych warstw:

1. Warstwa prezentacji – frontend
2. Warstwa logiki aplikacji – komunikacja z bazą danych.
3. Warstwa przechowywania danych – plik JSON.

3.2 Projekt bazy danych

Dane aplikacji przechowywane są w pliku db.json. Struktura bazy danych została zaprojektowana w sposób przejrzysty i łatwy do modyfikacji. Główne kolekcje w pliku JSON to:

users: przechowuje dane użytkownika takie jak : id, email , hasło, rola oraz status aktywności.

role: zawiera listę dostępnych ról użytkowników.

coaches: przechowuje dane trenerów tj. imię, nazwisko, rola.

players: przechowuje dane zawodników w tym ich pozycję oraz liczbę goli/czystych kont.

training: zawiera informację na temat treningów, datę, ćwiczenie i przypisanych zawodników.

game: zawiera informację o meczach, drużyny biorące udział i datę.

3.3 Diagramy UML

3.3.1 Diagram klas

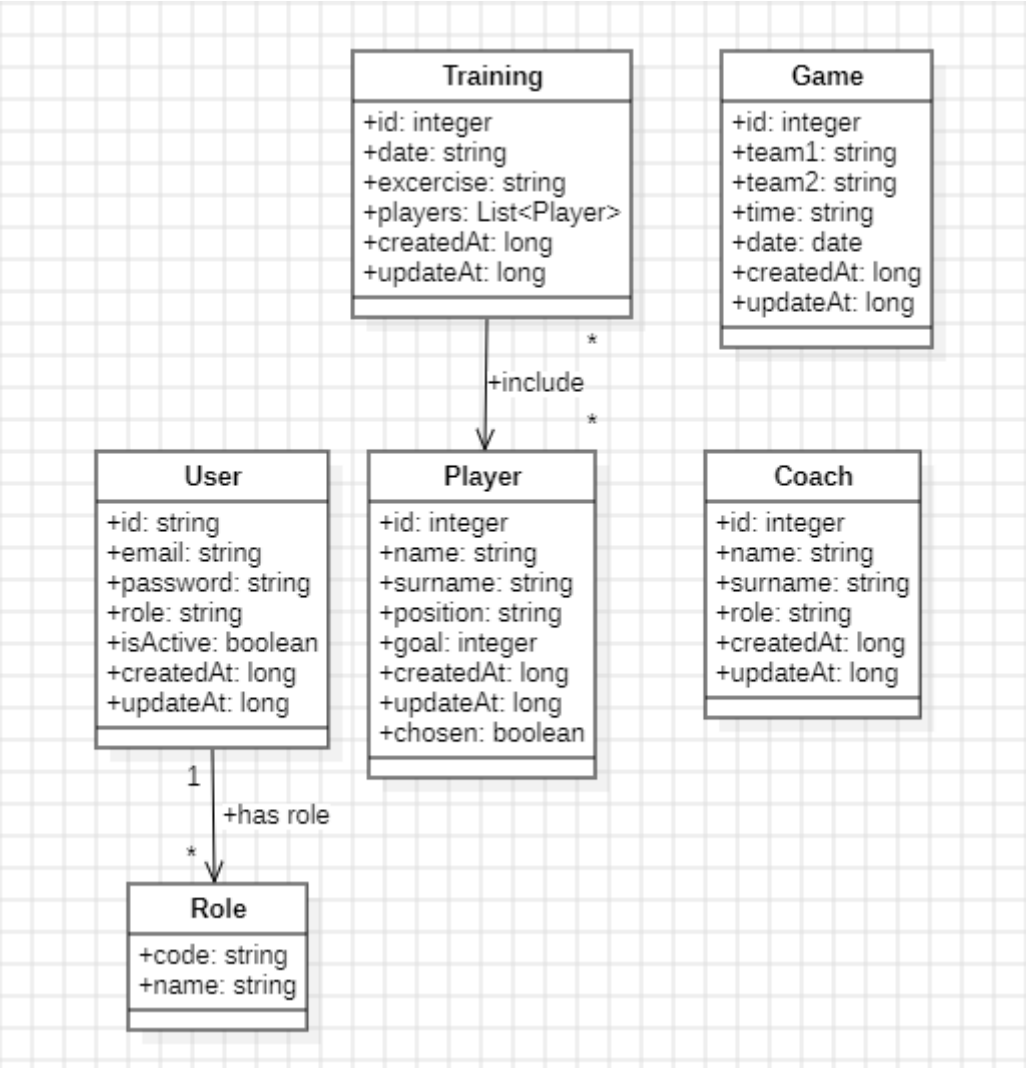


Diagram 8 Diagram klas - opracowanie własne

3.3.2 Diagram komponentów

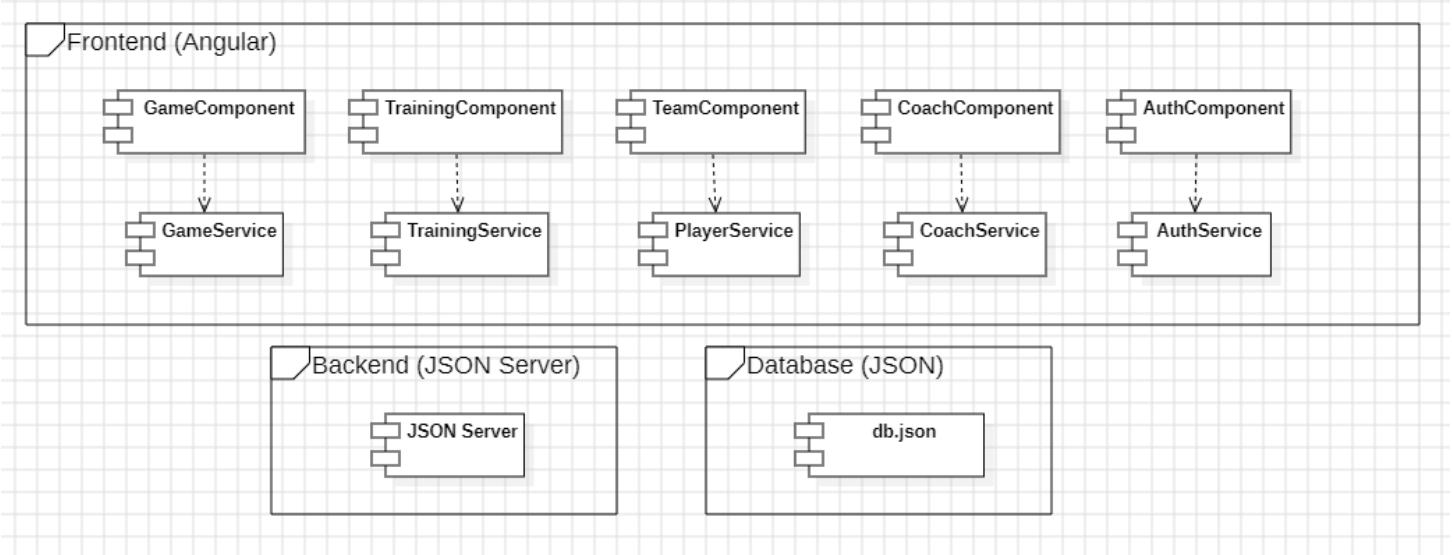


Diagram 9 Diagram komponentów - opracowanie własne

3.3.3 Diagram sekwencji – Proces dodawania nowego trenera

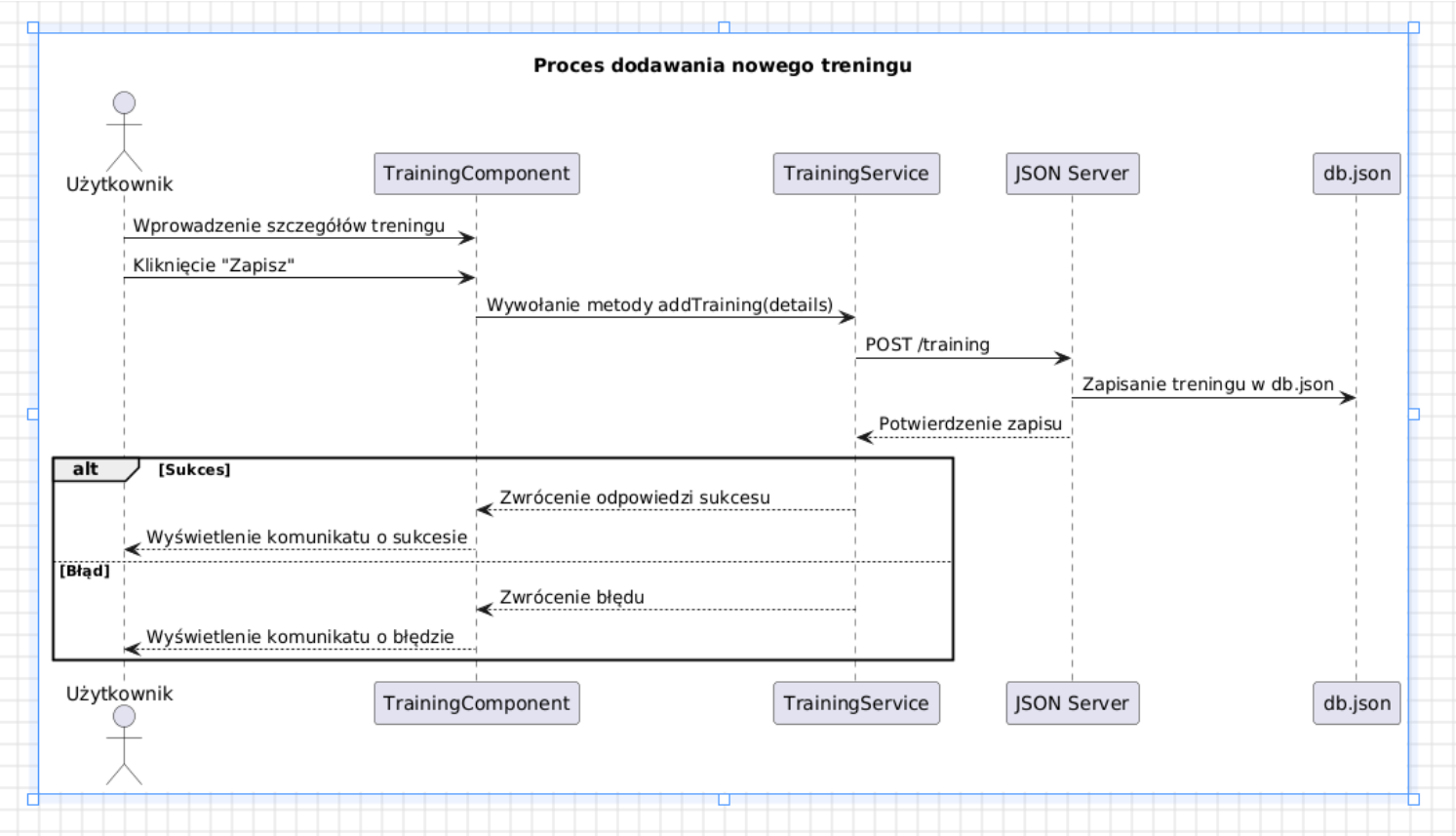


Diagram 10 Diagram sekwencji proces dodawania nowego trenera - opracowanie własne

Rozdział 4 Opis funkcjonalności i procesów biznesowych

4.1 Rejestracja nowego użytkownika

Funkcjonalność rejestracji nowego użytkownika obejmuje:

1. Formularz na froncie w komponencie o nazwie RegistrationComponent.
    - Wprowadzenie danych (e-mail, login, hasło)
    - Walidacja formularza (np. minimalne wymagania dla hasła)
2. Przesyłanie danych do authService, który komunikuje się z JSON Server przez REST API (POST/users)
3. Zapis danych w pliku db.json przez JSON Server

4.2 Zarządzanie treningami

Funkcjonalność zarządzania treningami została zaimplementowana jako:

1. Komponent TreningCopomonent umożliwiający:
    - Dodawanie nowych treningów.
    - Wyświetlanie nowych treningów.
2. TreningService obsługuję komunikację z JSON Server i wystawia metody takie jak GetAll, createTraining.
3. Dane treningów są przechowywane w sekcji training w pliku db.json. Każdy trening zawiera datę, nazwę ćwiczenia oraz listę uczestników.

4.3 Zarządzanie meczami

1. Formularz w komponencie GameGeneratorComponent umożliwia dodawanie meczów wraz z informacjami o drużynach pomiędzy którymi dojdzie do konfrontacji, datą i dokładną godziną.
2. GameService obsługuję operację takie jak GetAllGame, AddGame.

4.4 Dodanie nowego zawodnika

1. Komponent który zawiera formularz umożliwiający dodawanie nowego zawodnika to add-playerComponent. Pozwala on na:
    - Wprowadzenie imienia i nazwiska zawodnika.
    - Przypisanie pozycji na boisku
2. Przesyła on dane do komponentu rodzica team-list.page.component który zawiera serwis o nazwie PlayerService obsługujący operacje CRUD (od ang. create, read, update, delete) na danym zawodniku w pliku db.json.
3. Po wyświetleniu wszystkich zawodników dzięki komponentowi team-list-filters istnieje możliwość jego filtrowania z listy po imieniu bądź nazwisku a także pozycji na boisku.

4.5 Dodanie nowego trenera

Komponent który odpowiada za logikę dodawania nowego trenera do kadry trenerskiej zawiera funkcjonalności:

1. Formularz umożliwiający wprowadzenie:
    - Imienia i nazwiska nowego trenera.
    - Roli trenera w drużynie (np. główny trener).
2. Wysyła on dane pobrane od użytkownika przy użyciu formularza do komponentu rodzica (list-coaches.component), który przy użyciu serwisu CoachesService dodaje nowego trenera do listy. Serwis ten zawiera również funkcje pozwalające na edytowanie bądź usuwanie istniejących już obiektów w bazie danych.
3. Dane trenerów zapisane są w pliku db.json w sekcji coaches. Za pomocą komponentu coach-list-filters możliwa jest wcześniejsza selekcja danych przed ich wyświetleniem za pomocą dostępnych w nim funkcji filtrujących dane.

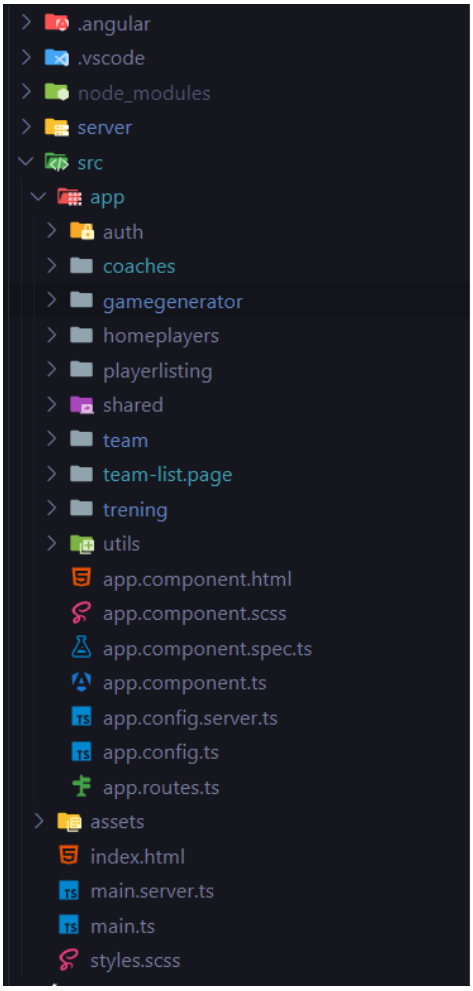
Rozdział 5 Architektura i szczegóły implementacji

5.1 Struktura projektu

5.1.1 Frontend

Frontend aplikacji został przygotowany przy użyciu frameworka Angular 17, wykorzystując podejście standalone, które eliminuje konieczność używania tradycyjnych modułów (NgModule). Podejście to znacznie uprościło strukturę projektu co znacząco zwiększyło jego przejrzystość i modularność.

Struktura projektu frontendowego została przedstawiona na poniższym obrazku.



Obraz 1 Struktura plików aplikacji - opracowanie własne

Każda funkcjonalność w aplikacji została zaprojektowana jako niezależne komponenty standalone. Dzięki temu są one samowystarczalne i nie wymagają deklaracji w modułach. Każdy komponent który jest standalone zarządza własnymi zależnościami poprzez sekcję imports.

Głównym punktem startowym aplikacji jest plik main.ts, w którym aplikacja jest inicjalizowana za pomocą metody bootstrapAplication.

Kod inicjalizacji aplikacji :

main.ts

```ts
import { bootstrapApplication } from '@angular/platform-browser';
import { appConfig } from './app/app.config';
import { AppComponent } from './app/app.component';


bootstrapApplication(AppComponent, appConfig)
  .catch((err) => console.error(err));

```

Obraz 2 Plik main.ts - opracowanie własne

Główne komponenty frontendowe:

- Strona główna (AppComponent)– wyświetla podstawowe informacje oraz umożliwia dostęp do pozostałych sekcji aplikacji.
- Zarządzanie zawodnikami (teamComponent, team-list-pageComponent) – odpowiada za funkcje dodawania, edycji i usuwania zawodników.
- Treningi (treninglistingComponent) – zawiera widoki do zarządzania treningami.
- Mecze (gamelistingComponent) – wyświetla harmonogram meczów.

5.1.2 Backend

Backend aplikacji został zrealizowany za pomocą JSON Server, który pełni funkcję lokalnego serwera REST API. Dane są przechowywane w pliku db.json. Struktura danych obejmuje:

- Tabele w db.json
  - users – przechowuje dane użytkowników (e-mail,login,hasło).
  - training – lista treningów z datą, opisem i uczestnikami.
  - game – harmonogram meczów z informacjami o drużynach
  - role – role użytkowników.
  - players – lista zawodników z imieniem, nazwiskiem, pozycją na boisku i liczbą bramek/czystych kont.
  - coaches – lista trenerów drużyny

5.2 Kluczowe komponenty aplikacji

Aplikacja została podzielona na modułowe sekcje, które realizują różne funkcjonalności. Każda sekcja składa się z komponentów standalone, które są odpowiedzialne za odrębne funkcjonalności. W dalszej części zostaną pokazane główne komponenty odpowiedzialne za prawidłowe funkcjonowanie aplikacji.

5.2.1 Strona główna

Strona główna aplikacji wyświetla podstawowe informacje i umożliwia dostęp do innych sekcji przy wykorzystaniu RouterModule.

```
export class AppComponent implements DoCheck {
    private router = inject(Router);
    private serviceAuth = inject(AuthService);

    ismenurequired = true;
    isadminuser = false;
    isplayeruser = false;
    ngDoCheck(): void {
        let currenturl = this.router.url;
        if (currenturl == '/login' || currenturl == '/registration') {
            this.ismenurequired = false;
        } else {
            this.ismenurequired = true;
        }
        if (this.serviceAuth.GetUserrole() === 'admin') {
            this.isadminuser = true;
            this.isplayeruser = false;
        } else {
            this.isadminuser = false;
            this.isplayeruser = true;
        }
    }
}
```

Obraz 3 Fragment pliku app.component.ts - opracowanie własne

app.component.html M

Go to component

```
<main class="main">
    <div class="toolbar" *ngIf="ismenurequired" role="banner">
        <button [routerLink]="['home-players']">Home</button>
        <button *ngIf="isadminuser" [routerLink]="['coach']">Trenerzy</button>
        <button *ngIf="isplayeruser" [routerLink]="['coaches-list']">Lista Trenerów</button>
        <button *ngIf="isadminuser" [routerLink]="['team']">Zawodnicy</button>
        <button *ngIf="isplayeruser" [routerLink]="['players-list']">Lista Zawodników</button>
        <button *ngIf="isadminuser" [routerLink]="['trening']">Treningi</button>
        <button *ngIf="isplayeruser" [routerLink]="['treining-list']">Lista Treningów</button>
        <button *ngIf="isadminuser" [routerLink]="['game-generator']">Mecze</button>
        <button *ngIf="isplayeruser" [routerLink]="['game-list']">Lista meczów</button>
        <button *ngIf="isadminuser" [routerLink]="['/user']">User</button>
        <button [routerLink]="['/login']">Wyloguj</button>
    </div>
    <router-outlet></router-outlet>
</main>
```

Obraz 4 Plik app.component.html – opracowanie własne

5.2.2 Zarządzanie zespołem

Zarządzanie zespołem pozwala na dodawanie, edytowanie i usuwanie zawodników.

```
export class TeamComponent {
    @Input({ required: true }) players: Player[] = [];
    visiblePlayers: Player[] = [];
    pageSize = 3;
    currentPage = 0;
    private playersService = inject(PlayersService);

    selectPlayer: Player | undefined;

    ngOnInit() {
        this.updateVisiblePlayers();
    }

    playerSelected(player: Player, event: Event) {
        event.stopPropagation();
        player.chosen = !player.chosen;
    }

    updatePlayer(playerId: number, updatePlayer: PlayerUpdatePayload) {
        this.playersService.update(playerId, updatePlayer).subscribe({
            next: (res) => {
                this.players = this.players.map((player) =>
                    player.id === res.id ? res : player
                );
                this.updateVisiblePlayers();
            },
            error: (res) => {
                alert(res.message);
            },
        });
    }
```

Obraz 5 Fragment pliku team.component.ts - opracowanie własne

```
<h1>Kadra zespołu</h1>
<ul>
    <li *ngFor="let player of visiblePlayers">
        <app-player-card [player]="player" (update)="updatePlayer(player.id, $event)"
            (delete)="deletePlayer(player.id)">
        </app-player-card>
    </li>
</ul>
<mat-paginator [length]="players.length" [pageSize]="pageSize" (page)="onPageChange($event)">
</mat-paginator>
```

Obraz 6 Fragment pliku team.component.html

5.2.3 Zarządzanie kadrą trenerską

Zarządzanie kadrą trenerską pozwala na dodawanie, edytowanie oraz usuwanie trenerów do drużyny.

```
export class CoachesComponent {
    @Input({ required: true }) coaches: Coach[] = [];
    visibleCoaches: Coach[]=[];
    pageSize = 3;
    currentPage = 0;

    private coachesService = inject(CoachesService);

    ngOnInit() {
        this.updateVisibleCoaches();
    }

    updateCoach(coachId: number, updateCoach: CoachUpdatePayload) {
        this.coachesService.update(coachId, updateCoach).subscribe({
            next: (res) => {
                this.coaches = this.coaches.map((coach) => {
                    if (coach.id === res.id) {
                        return res;
                    } else {
                        return coach;
                    }
                });
                this.updateVisibleCoaches();
            },
            error: (res) => {
                alert(res.message);
            },
        });
    }

    deleteCoach(coachId: number) {
        this.coachesService.deleteCoach(coachId).subscribe({
            next: () => {
                this.coaches = this.coaches.filter((coach) => coach.id !== coachId);
                this.updateVisibleCoaches();
            },
            error: (res) => {
                alert(res.message);
```

Obraz 7 Fragment pliku coaches.component.ts - opracowanie własne

```
Go to component
<h1>Sztab trenerski</h1>
<ul>
    <li *ngFor="let coach of visibleCoaches">
        <app-card-coaches [coach]="coach" (updateCoaches)="updateCoach(coach.id , $event)"
            (deleteCoaches)="deleteCoach(coach.id)"></app-card-coaches>
    </li>
</ul>

<mat-paginator [length]="coaches.length" [pageSize]="pageSize" (page)="onPageChange($event)">
</mat-paginator>
```

Obraz 8 Fragment pliku coaches.component.html - opracowanie własne

5.2.4 Zarządzanie treningami

Komponent umożliwia dodanie nowego treningu do harmonogramu. Użytkownik może wybrać datę, ćwiczenie oraz zawodników biorących udział.

```
70  export class TreningComponent {
71      @Input({ required: true }) players: Player[] = [];
72      selectedTime: string = '';
73
74      selectedDate: Date | null = null;
75      minDate: Date = new Date();
76
77  >   newTraining: { date: Date; exercise: string; players: Player[] } = { ···
81      };
82      dataSource: any;
83      private playersServie = inject(PlayersService);
84      private trainingService = inject(TreningServiceService);
85      private toastr = inject(ToastrService);
86      private cdr = inject(ChangeDetectorRef);
87      listState: ComponentListState<Player> = { state: LIST_STATE_VALUE.IDLE };
88      listStateValue = LIST_STATE_VALUE;
89
90      readonly selectedPlayers = signal<Player[]>([]);
91
92  >   ngOnInit(): void {···
98      }
99
100 >   getAllPlayers(searchParams: GetAllPlayersSearchParams): void {···
120     }
121 >   createNewTraining(): void {···
164     }
165
166 >   openTimepicker(timepicker: any) {···
168     }
169
170 >   togglePlayerSelection(player: Player, event: MatCheckboxChange): void {···
179     }
180
181 >   assignTaskToSelectedPlayers(): void {···
187     }
188 }
```

Obraz 9 Fragment pliku trening.component.ts - opracowanie własne

```html
<div class="treining">
    <h1>Wybór treningu</h1>
    <mat-form-field>
        <mat-label>Wybierz datę</mat-label>
        <input matInput [matDatepicker]="picker" [(ngModel)]="selectedDate" [min]="minDate">
        <mat-datepicker-toggle matIconSuffix [for]="picker"></mat-datepicker-toggle>
        <mat-datepicker #picker></mat-datepicker>
    </mat-form-field>

    <mat-form-field>
        <mat-label>Wybierz godzinę</mat-label>
        <input matInput aria-label="24hr format" [ngxTimepicker]="fullTime" [format]="24" readonly
            [(ngModel)]="selectedTime">
        <mat-icon matSuffix (click)="openTimepicker(fullTime)">access_time</mat-icon>
        <ngx-material-timepicker #fullTime></ngx-material-timepicker>
    </mat-form-field>

    <mat-form-field>
        <mat-label>Wybierz ćwiczenie</mat-label>
        <mat-select [(ngModel)]="newTraining.exercise">
            <mat-option value="Trening taktyczny">Trening taktyczny</mat-option>
            <mat-option value="Trening strzelecki">Trening strzelecki</mat-option>
            <mat-option value="Trening wydolnościowy">Trening wydolnościowy</mat-option>
        </mat-select>
    </mat-form-field>
```

Obraz 10 Fragment pliku trening.component.html – opracowanie własne

5.2.5 Zarządzanie meczami

Za pomocą formularza użytkownik może dodawać nowe mecze do harmonogramu.

```typescript
    ngOnInit(): void {
        this.gameForm = this.gameAddFormBuilder.group({
            team1: this.gameAddFormBuilder.control('', Validators.required),
            team2: this.gameAddFormBuilder.control('', Validators.required),
            time: this.gameAddFormBuilder.control('', Validators.required),
            date: this.gameAddFormBuilder.control('', Validators.required),
        });
    }

    gameAdd() {
        if (this.gameForm.valid) {
            const formValue = this.gameForm.value;

            const localDate = new Date(formValue.date);
            localDate.setHours(0, 0, 0, 0);

            const gameData = {
                ...formValue,
                date: localDate.toISOString(),
            };

            this.gameService.AddGame(gameData).subscribe({
                next: (res) => {
                    this.toastr.success('Mecz został utworzony');
                },
            });
        } else {
            this.toastr.warning('Coś poszło nie tak spróbuj jeszcze raz');
        }
    }
```

Obraz 11 Fragment pliku gamegenerator.component.ts – opracowanie własne

```
            <mat-form-field>
                <mat-label>Drużyna gości</mat-label>
                <mat-select formControlName="team2" [(value)]="selectedTeam2">
                    <mat-option *ngFor="let team of filteredTeams" [value]="team.value"
                        [disabled]="team.value === selectedTeam1">
                        {{ team.viewValue }}
                    </mat-option>
                </mat-select>
            </mat-form-field>
        </div>

        <mat-form-field>
            <mat-label>Wybierz datę</mat-label>
            <input formControlName="date" matInput [matDatepicker]="picker" [min]="minDate">
            <mat-datepicker-toggle matIconSuffix [for]="picker"></mat-datepicker-toggle>
            <mat-datepicker #picker></mat-datepicker>
        </mat-form-field>

        <mat-form-field>
            <mat-label>Wybierz godzinę</mat-label>
            <input formControlName="time" matInput aria-label="24hr format" [ngxTimepicker]="fullTime" [format]="24"
                readonly>
            <mat-icon matSuffix (click)="openTimepicker(fullTime)">access_time</mat-icon>
            <ngx-material-timepicker #fullTime></ngx-material-timepicker>
        </mat-form-field>
```

Obraz 12 Fragment pliku gamegenerator.component.html - opracowanie własne

5.3 Integracja z backendem

5.3.1 Endpointy API

Komunikacja z backendem odbywa się poprzez JSON Server z użyciem REST API. Główne endpointy :

- GET / users – Pobranie użytkowników.
- POST / training – Tworzenie nowego treningu.
- POST / game – Tworzenie meczu.
- GET / players – Pobieranie listy zawodników.

5.3.2 Bezpieczeństwo i autoryzacja

Dane użytkowników (login, e-mail, hasło) są przechowywane w pliku db.json. W przyszłości można wprowadzić JWT lub mechanizmy szyfrowania haseł. Aktualnie zaimplementowano autoryzację polegającą na uprawnieniach nadawanych przez system na podstawie roli użytkownika.

Rozdział 6 Testowanie

6.1 Plan testów

Testowanie aplikacji miało na celu zapewnienie poprawności działania głównych funkcjonalności, stabilności oraz zgodności z wymaganiami funkcjonalnymi i niefunkcjonalnymi. W procesie testowania wyróżniono następujące etapy:

1. Testowanie jednostkowe – weryfikacja działania pojedynczych funkcji, komponentów oraz serwisów aplikacji.
2. Testowanie integracyjne – sprawdzenie poprawności komunikacji między komponentami frontendowymi o backendem (JSON Server).
3. Testowanie systemowe – przetestowanie aplikacji jako całości pod kątem spełnienia wymagań użytkownika końcowego.

6.2 Cel testów

- Weryfikacja poprawności działania funkcjonalności takich jak rejestracja użytkownika, zarządzanie zespołem, harmonogram treningów i meczów.
- Wykrycie błędów związanych z walidacją formularzy, przetwarzaniem danych w JSON Server oraz ich wyświetlenie na froncie.
- Sprawdzenie interakcji użytkownika z aplikacją.

6.3 Narzędzia testowe

- Karma + Jasmine – do testów jednostkowych w Angularze.
- Postman – do testowania endpointów REST API.
- Cypress – do testów systemowych.
- JSON Server – jako backend do obsługi testowych danych.

6.4 Testowanie jednostkowe

Testy jednostkowe zostały przeprowadzone dla kluczowych funkcji aplikacji. W Angularze wykorzystano framework Jasmine oraz Karmę jako narzędzie uruchamiające testy.

Zakres testów jednostykowych :

1. Komponenty :
    - RegistrationComponent – testy walidacji formularza (np. minimalna długość hasła, poprawność adresu e-mail).
    - TeamComponent – weryfikacja poprawnego renderowania listy zawodników.
    - TreningComponent – testy formularza dodawania nowego trenera.

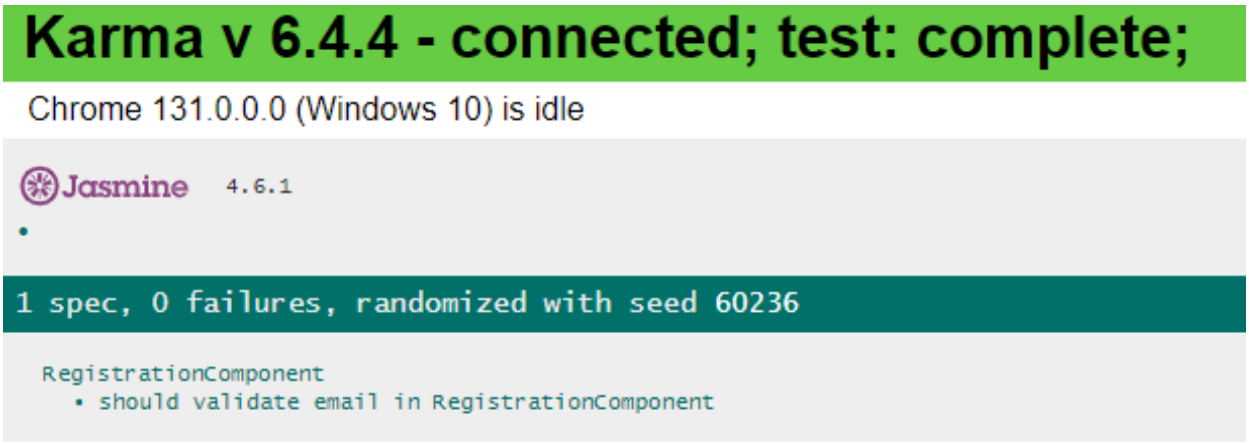
2. Serwisy :
    - AuthService – testy poprawności logowania oraz rejestracji użytkowników.
    - GameService – test metody addGame() i jej interreakcji z backendem.

6.4.1 Test formularza rejestracyjnego

Test jednostkowy na podstawie formularza rejestracyjnego ma na celu sprawdzić walidację pola email w formularzu rejestracyjnym (registrationForm) w komponencie (RegistrationComponent). Sprawdza, czy pole email jest niepoprawne (czy jest uznane za nieważne w formularzu) oraz czy dla tego pola został wygenerowany błąd walidacji o nazwie email, który wskazuje, że wprowadzony tekst nie jest prawidłowym adresem e-mail. Daje to zapewnienie, że formularz poprawnie identyfikuje błędy związane z niepoprawnym formatem adresu mailowego.

```
describe('RegistrationComponent', () => {
  let component: RegistrationComponent;
  let fixture: ComponentFixture<RegistrationComponent>;

  beforeEach(async () => {
    await TestBed.configureTestingModule({
      imports: [
        ReactiveFormsModule,
        ToastrModule.forRoot(),
        NoopAnimationsModule,
        HttpClientModule,
        RegistrationComponent
      ],
      providers: [
        { provide: ActivatedRoute, useValue: { snapshot: { queryParams: {} } } }
      ]
    }).compileComponents();

    fixture = TestBed.createComponent(RegistrationComponent);
    component = fixture.componentInstance;
    fixture.detectChanges();
  });

  it('should validate email in RegistrationComponent', () => {
    component.registrationForm.controls['email'].setValue('invalid-email');
    expect(component.registrationForm.controls['email'].valid).toBeFalse();
    expect(component.registrationForm.controls['email'].hasError('email')).toBeTrue();
  });
});
```

Obraz 13 Fragment pliku registration.component.spec.ts - opracowanie własne



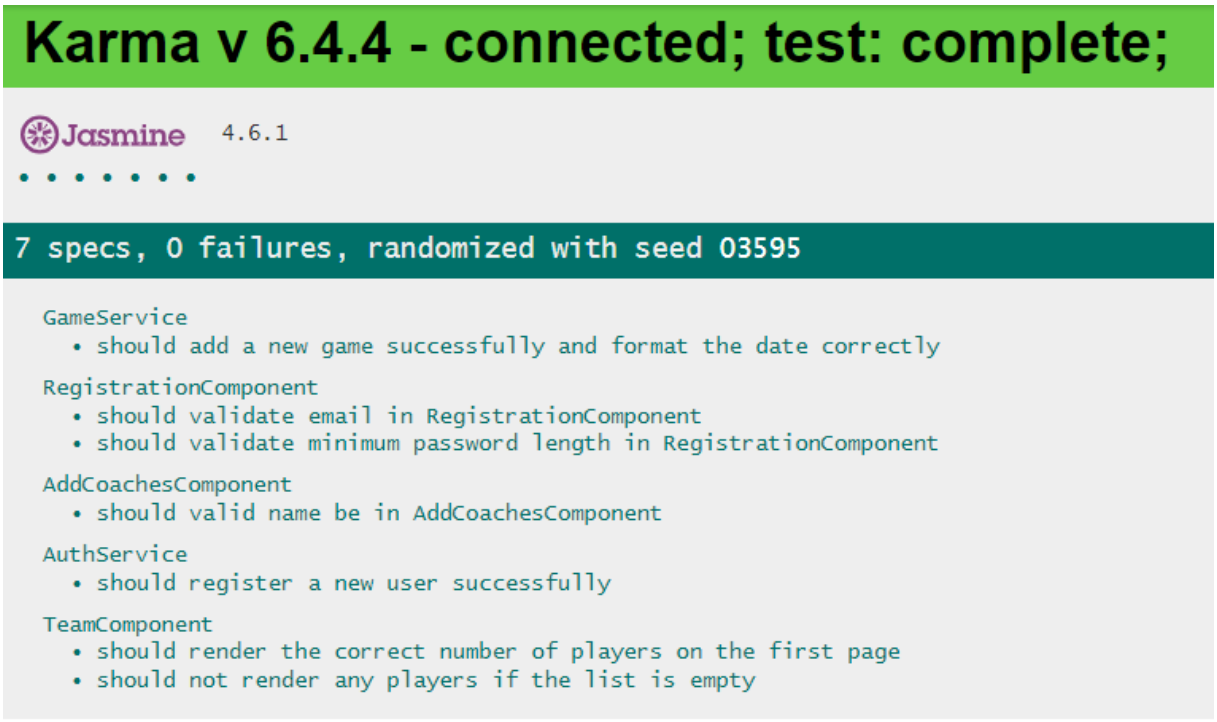
Obraz 14 Wynik testu formularza rejestracyjnego - opracowanie własne

6.4.2 Test poprawności rejestracji użytkowników

Test ma na celu sprawdzenie poprawności działania funkcjonalności rejestracji nowych użytkowników. W ramach testu, symulujemy wysłanie danych nowego użytkownika za pomocą metody Registration() z serwisu AuthService. Test sprawdza, czy metoda poprawnie wysyła żądanie HTTP POST do serwera oraz czy dane użytkownika są prawidłowo dodawane do bazy danych. Celem jest upewnienie się, że dane użytkownika zostały przesłane i poprawnie zapisane.

```
describe('AuthService', () => {
    let service: AuthService;
    let httpMock: HttpTestingController;

    beforeEach(async () => {
        await TestBed.configureTestingModule({
            imports: [NoopAnimationsModule, ReactiveFormsModule, HttpClientTestingModule],
            providers: [AuthService],
        }).compileComponents();

        service = TestBed.inject(AuthService);
        httpMock = TestBed.inject(HttpTestingController);
    });

    it('should register a new user successfully', (done) => {
        const newUser = {
            username: 'newUser',
            useremail: 'newUser@mail.pl',
            password: 'Newuserpassword123',
        };

        service.Registration(newUser).subscribe((result) => {
            expect(result).toBeTruthy();
            done();
        });
```

Obraz 15 Fragment pliku auth.service.spec.ts - opracowanie własne



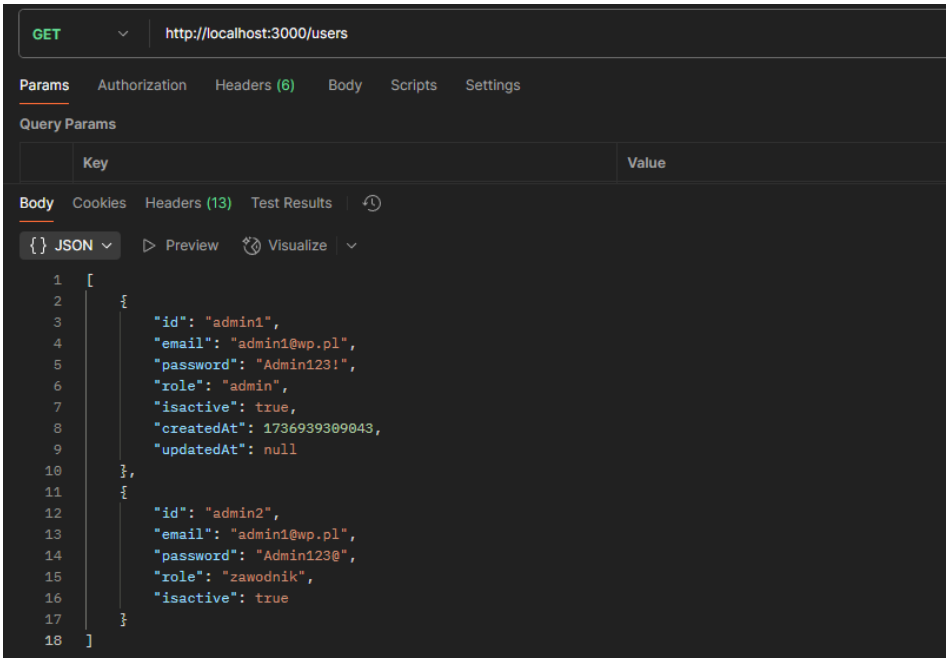
Obraz 16 Wyniki testów jednostkowych - opracowanie własne

6.5 Testowanie integracyjne

Testy integracyjne skupiały się na weryfikacji poprawnej współpracy między komponentami frontendowymi a backendem opartym na JSON Server.

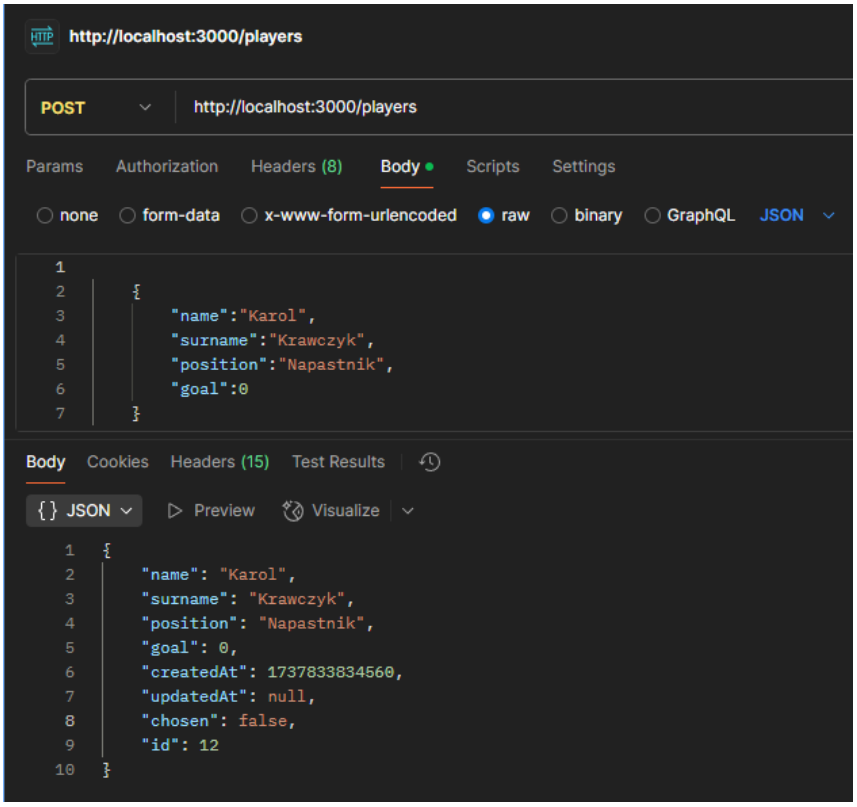
Zakres testów integracyjnych:

1. Autoryzacja użytkownika:
    - Sprawdzenie poprawnego przesyłania danych z formularza rejestracji i logowania do backendu (POST/users).
    - Weryfikacja odpowiedzi serwera (np. status 201 dla poprawnej rejestracji).

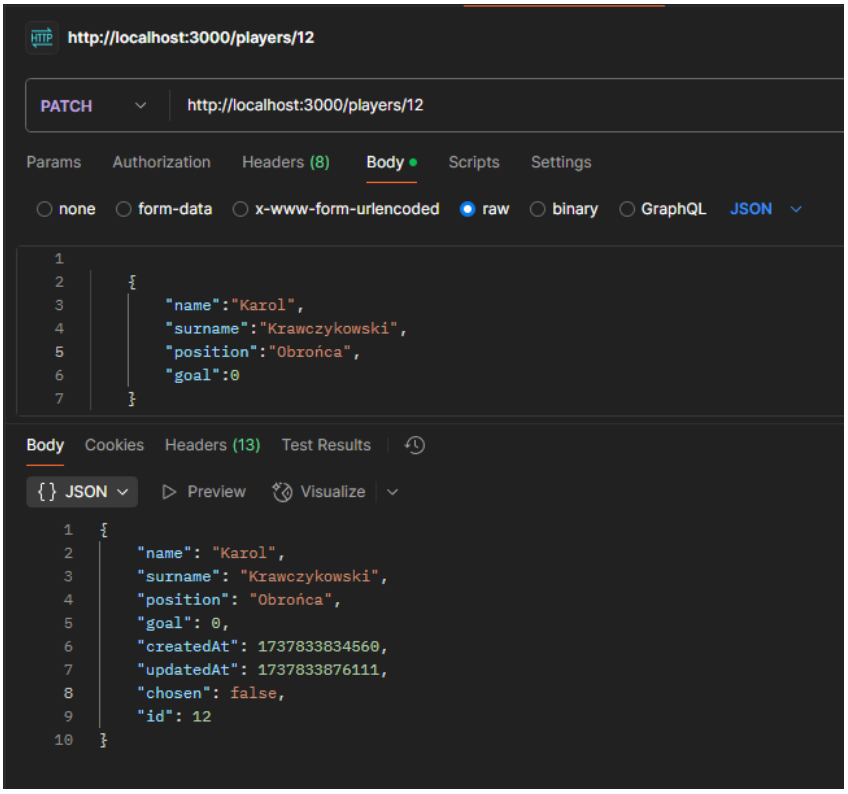


Obraz 17Postman GET users - opracowanie własne

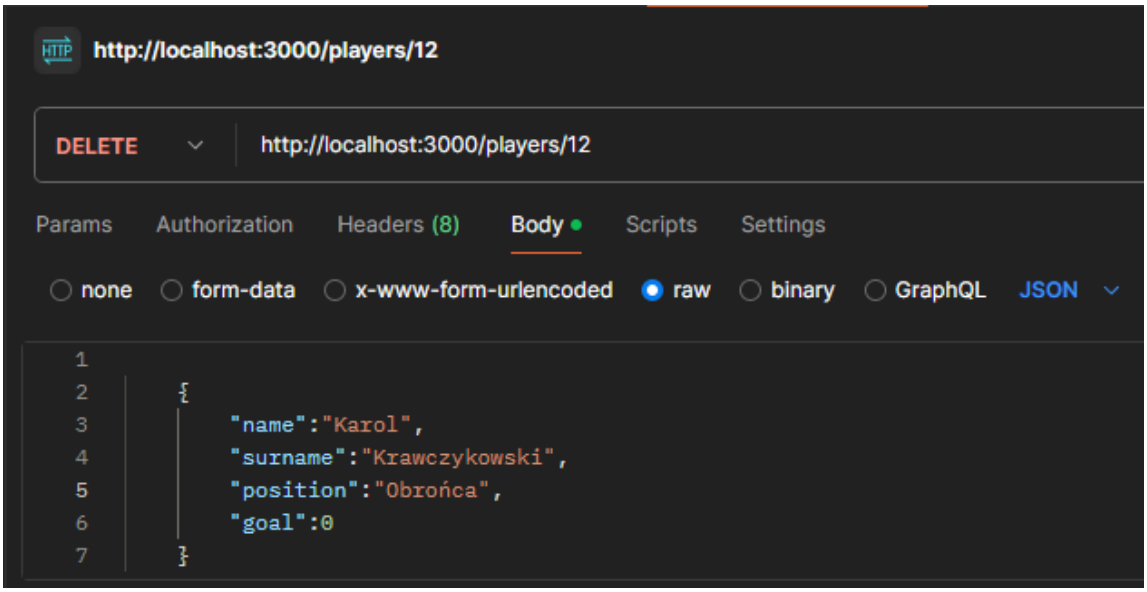
2. Zarządzanie zawodnikami:
    - Testy komunikacji między TeamComponent a PlayerService przy dodawaniu, edytowaniu i usuwaniu zawodnika.



Obraz 18 Postman POST players /dodanie zawodnika- opracowanie własne

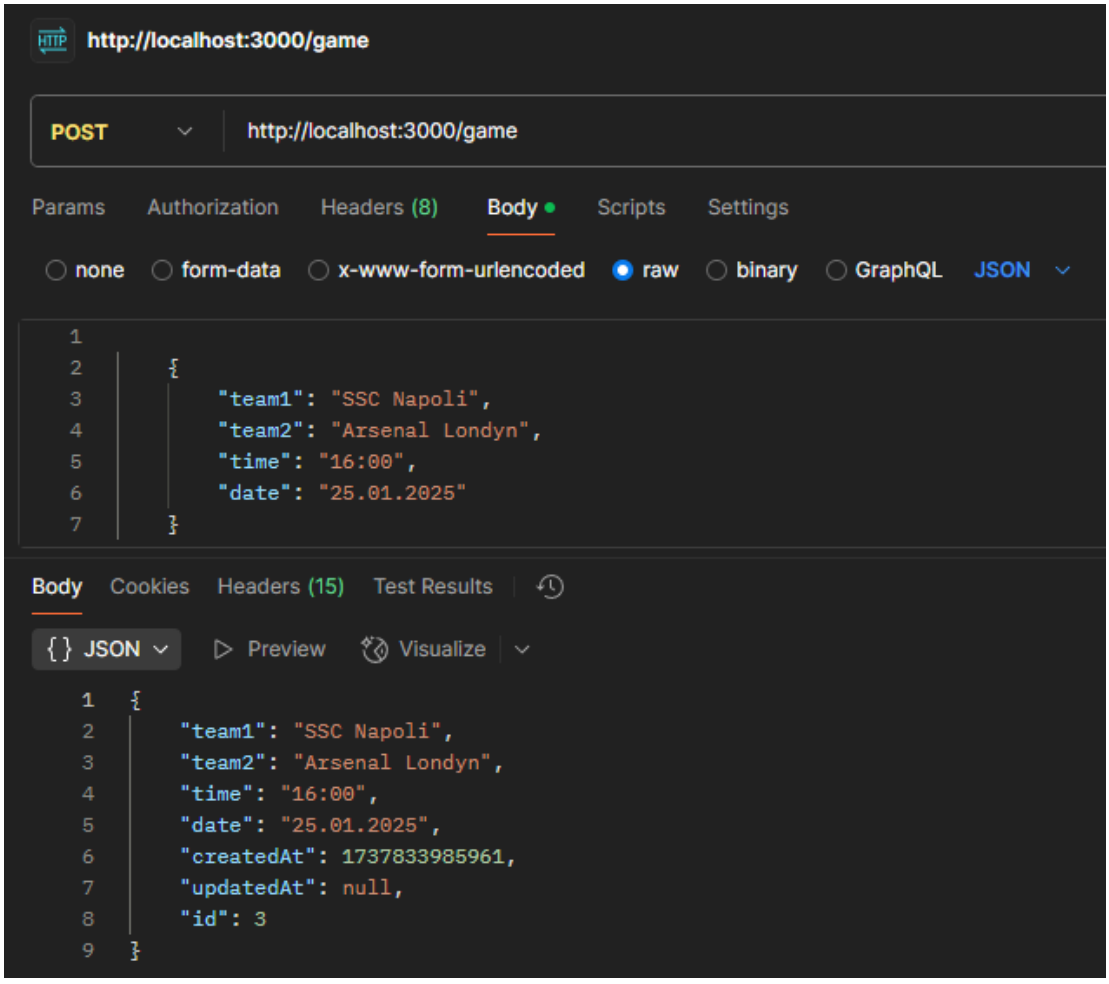


Obraz 19 Postman PATCH players / edycja zawodnika - opracowanie własne



Obraz 20 Postman DELETE players / usuwanie zawodnika - opracowanie własne

3. Harmonogram meczów:
    - Sprawdzenie poprawności przesyłania danych o meczach JSON Server do GameGeneratorComponent.
    - Weryfikacja, czy dane z backendu są poprawnie wyświetlone na froncie.



Obraz 21 Postman POST game / dodawanie nowego meczu - opracowanie własne

6.6 Testowanie systemowe

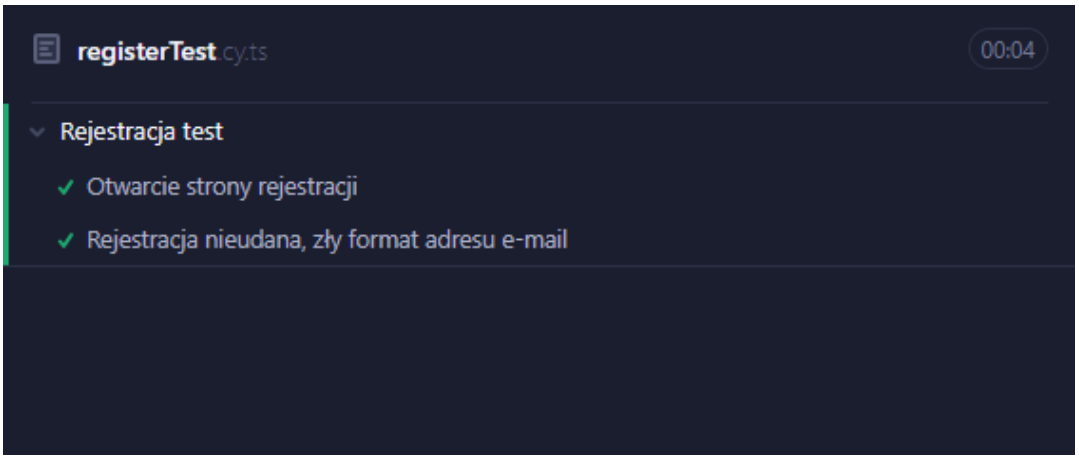
Testy systemowe obejmowały sprawdzenie działania aplikacji jako całości w warunkach maksymalnie zbliżonych do rzeczywistych. Zastosowano scenariusze testowe uwzgledniające wszystkie kluczowe funkcjonalności.

Scenariusze testowe:

1. Rejestracja użytkownika:
    - Użytkownik wprowadza poprawne dane w formularzu i otrzymuje komunikat o sukcesie.
    - Użytkownik podaje niepoprawne dane i otrzymuje odpowiedni komunikat błędu.

```
describe('Rejestracja test', () => {
    it('Otwarcie strony rejestracji', () => {
        cy.visit('/registration');

        cy.get('input[formControlName="id"]').type('Johny3');
        cy.get('input[formControlName="email"]').type('John@john.pl');
        cy.get('input[formControlName="password"]').type('Johny123!');
        cy.get('button[type="submit"]').click();

        cy.url().should('include', '/login');

        cy.contains('Rejestracja udana!').should('be.visible');
    });

    it('Rejestracja nieudana, zły format adresu e-mail', () => {
        cy.visit('/registration');

        cy.get('input[formControlName="id"]').type('Johny2');
        cy.get('input[formControlName="email"]').type('johny2@');
        cy.get('input[formControlName="password"]').type('Johny123@');
        cy.get('button[type="submit"]').click();

        cy.contains('Sprawdź poprawność danych').should('be.visible');
    });
});
```

Obraz 22 Cypress kod testowy funkcjonalność rejestracji - opracowanie własne



Obraz 23 Cypress wyniki testu funkcjonalności rejestracji - opracowanie własne

2. Dodanie nowego zawodnika:
    - Dodanie nowego zawodnika powoduje aktualizację listy z kadrą zespołu.

```
describe('Dodawanie zawodnika', () => {
    it('Poprawne dodanie zawodnika', () => {
        cy.visit('/login');

        cy.get('input[formControlName="name"]').type('admin1');
        cy.get('input[formControlName="password"]').type('Admin123!');
        cy.get('button[type="submit"]').click();

        cy.get('button[name="team"]').click();

        cy.get('input#name').type('Jan');
        cy.get('input#surname').type('Kowalski');
        cy.get('mat-select#position').click();
        cy.get('mat-option').contains('Napastnik').click(); //


        cy.get('button').contains('Dodaj zawodnika').click();
    });
});

```

Obraz 24 Cypress kod testowy funkcjonalności dodanie nowego zawodnika - opracowanie własne

addPlayer cy.ts

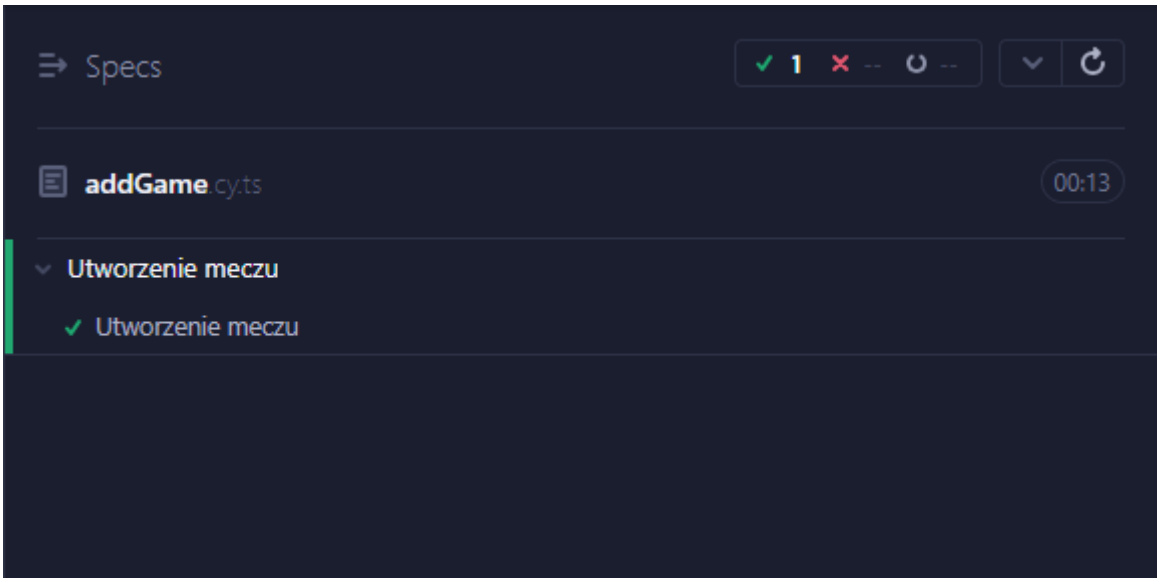
```
-type admin1
get input[formControlName="password"]
-type Admin123!
get button[type="submit"]
-click
(fetch) GET 200 http://localhost:3000/users/admin1
(new url) http://localhost:4200/home-players
(fetch) GET 200 http://localhost:3000/users/admin1
(fetch) GET 200 http://localhost:3000/training
get button[name="team"]
-click
(fetch) GET 200 http://localhost:3000/game
(new url) http://localhost:4200/team
get input#name
-type Jan
(fetch) GET 200 http://localhost:3000/players?q=&position_like=
get input#surname
-type Kowalski
get mat-select#position
-click
get mat-option
-contains Napastnik
-click
get button
-contains Dodaj zawodnika
-click
(fetch) POST 201 http://localhost:3000/players
```

Obraz 25 Cypress wynik testu funkcjonalności dodanie nowego zawodnika - opracowanie własne

3. Dodanie meczu:
    - Użytkownik dodaje nowy mecz, który jest dodawany do harmonogramu i otrzymuje komunikat o sukcesie.

```
describe('Utworzenie meczu', () => {
    it('Utworzenie meczu', () => {
        cy.visit('/login');

        cy.get('input[formControlName="name"]').type('admin1');
        cy.get('input[formControlName="password"]').type('Admin123!');
        cy.get('button[type="submit"]').click();

        cy.get('button[name="game"]').click();

        cy.get('mat-select[formControlName="team1"]').click();
        cy.get('mat-option').contains('Fc Barcelona').click();

        cy.get('mat-select[formControlName="team2"]').click();
        cy.get('mat-option').contains('Real Madryt').click();

        cy.get('mat-datepicker-toggle').click();
        cy.get('.mat-calendar-body-today').click();


        cy.get('input[formControlName="time"]').click({ force: true });
        cy.wait(1000);
        cy.contains('14').click();
        cy.contains('00').click({ force: true });
        cy.get('button').contains('Ok').click({ force: true });
        cy.get('button').contains('Zaplanuj mecz').click();

        cy.visit('/game-list');
    });
});

```

Obraz 26 Cypress kod testowy funkcjonalności dodawanie meczu - opracowanie własne

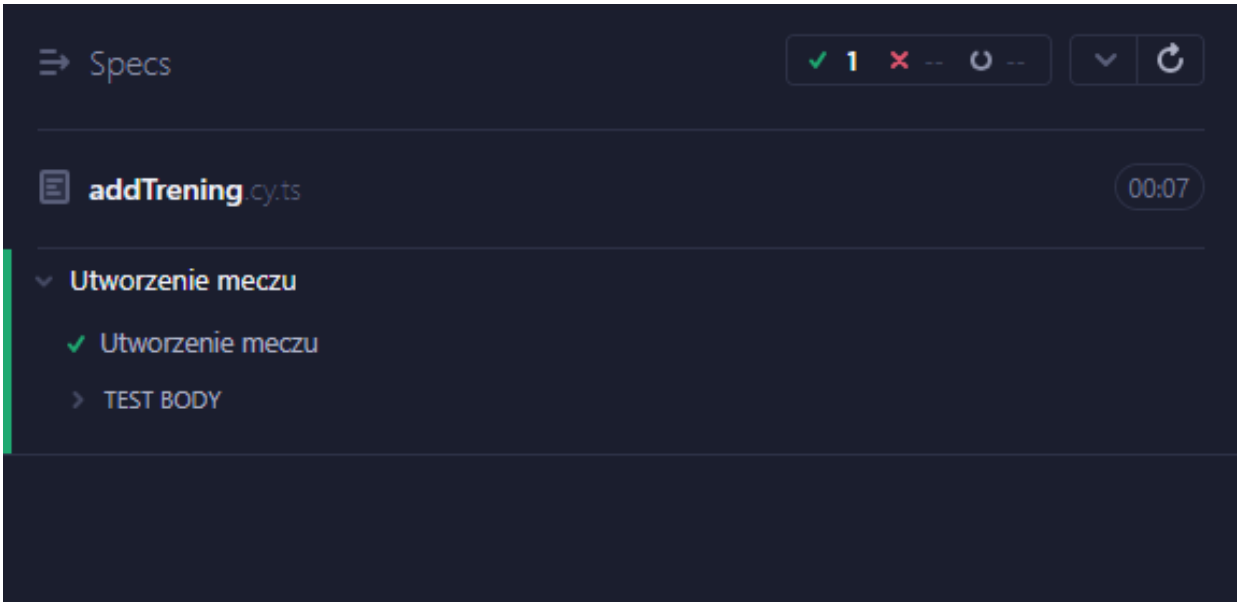


Obraz 27 Cypress wynik testu funkcjonalności dodawanie meczu - opracowanie własne

4. Dodanie treningu:
    - Wypełnienie formularza odpowiadającego za dodanie treningu, powoduje utworzenie nowego treningu, widocznego na liście.

```
describe('Utworzenie meczu', () => {
    it('Utworzenie meczu', () => {
        cy.visit('/login');

        cy.get('input[formControlName="name"]').type('admin1');
        cy.get('input[formControlName="password"]').type('Admin123!');
        cy.get('button[type="submit"]').click();

        cy.get('button[name="training"]').click();

        cy.get('mat-datepicker-toggle').click();
        cy.get('.mat-calendar-body-today').click({ force: true });
        cy.get('.mat-calendar-body-today').click({ force: true });

        cy.get('input[aria-label="24hr format"]').click({ force: true });
        cy.wait(1000);
        cy.contains('14').click();
        cy.contains('00').click({ force: true });
        cy.get('button').contains('Ok').click({ force: true });
        cy.get('button').contains('Ok').click({ force: true });
        cy.get('mat-select').click();
        cy.get('mat-option').contains('Trening strzelecki').click();
        cy.get('mat-checkbox').contains('Tadeusz Norek').click();

        cy.get('button').contains('Przypisz zadania do wybranych zawodników').click();

        cy.visit('/treining-list');
    });
});
```

Obraz 28 Cypress kod testowy funkcjonalności dodanie nowego treningu - opracowanie własne



Obraz 29 Cypress wynik testu funkcjonalności dodanie nowego treningu - opracowanie własne

6.7 Podsumowanie

Przeprowadzono testy jednostkowe, integracyjne i systemowe, które pokryły wszystkie kluczowe funkcjonalności systemu. Dzięki wykorzystaniu narzędzi takich jak Karma, Jasmine, Postman czy Cypress zapewniono wysoką jakość aplikacji oraz zgodność z wymaganiami użytkowników końcowych.

Rozdział 7 Dyskusja i wnioski

W tym rozdziale znajduję się podsumowanie wyników przeprowadzonej pracy oraz omówienie napotkanych trudności i wyzwań, które pojawiły się podczas implementacji aplikacji. Dodatkowo zaprezentowano rozwiązania wprowadzonych problemów oraz wnioski dotyczące przyszłych działań.

7.1 Podsumowanie wyników

Celem pracy było stworzenie wydajnej i skalowalnej aplikacji webowej dla trenerów drużyn piłkarskich przy użyciu frameworka Angular. Ma ona za zadanie ułatwienie im codziennej pracy oraz poprawę jej jakości. Po przeprowadzeniu analizy wymagań, implementacji oraz testów uzyskano następujące wyniki:

1. Zrealizowanie podstawowych funkcji aplikacji

   Aplikacja umożliwia użytkownikom rejestrację, logowanie, zarządzanie kadrą drużyny oraz trenerów, dodawanie meczów i treningów do harmonogramu jak i ich wyświetlenie. Został wdrożony mechanizm autoryzacji oparty na roli użytkownika w systemie (administrator/użytkownik).

2. Sukcesy w zakresie UX/UI

   Aplikacja zapewnia użytkownikom pozytywne doświadczenia z interfejsu dzięki zastosowaniu prostego, ale efektywnego designu jaki oferuje biblioteka Angular Material. Interfejs jest przejrzysty, a nawigacja między sekcjami aplikacji jest intuicyjna.

3. Wydajność aplikacji

   Zoptymalizowano czas ładowania aplikacji poprzez implementację lazy loading oraz mechanizmów cache`owania.

7.2 Napotkane problemy i ich rozwiązania

Podczas realizacji projektu napotkano kilka wyzwań, które zostały skutecznie rozwiązane. Oto niektóre z nich:

1. Problemy z integracją i backendem

   Początkowo wystąpiły trudności z komunikacją frontendem (Angular) a backendem. Okazało się, że różnice w wersjach API oraz niepoprawnie skonfigurowane nagłówki CORS powodowały problem z autoryzacją użytkowników i dostępem do zasobów.

   Rozwiązanie: Problem rozwiązano poprzez aktualizację API. Dodatkowo wdrożono mechanizm obsługi błędów w aplikacji front-endowej, aby użytkownik otrzymał jasne komunikaty o nieudanych próbach logowania.

2. Optymalizacja wydajności

   Aplikacja początkowo miała problemy z wydajnością przy dużej liczbie danych, zwłaszcza w przypadku wyświetlania list i tabel w aplikacji.

   Rozwiązanie: Zastosowano lazy loading, oraz wprowadzenie stronnictwa w widokach tabel. Zoptymalizowano również zapytania do API, co znacząco poprawiło czas odpowiedzi aplikacji.

3. Implementacja Angular Material

   Przy próbie implementacji biblioteki Angular Material, aplikacja napotkała trudności związane z integracją niektórych komponentów, takich jak mat-datepicker (kalendarz wyboru daty) oraz mat-select (lista rozwijana). Problem pojawił się w momencie, gdy po zaimplementowaniu odpowiednich modułów w aplikacji, komponent nie był wyświetlany poprawnie, a ich interakcje z użytkownikiem były niestabilne (np. brak reakcji na kliknięcie)

   Rozwiązanie: W celu rozwiązania problemu zaktualizowano bibliotekę Angular Material. Sprawdzono poprawność importów oraz kolejność modułów Poprawiono inicjalizację komponentów. Zainstalowano dodatkowe zależności, takie jak MatNativeDateModule dla komponentu mat-datepicker. Przeprowadzono testy integracyjne.

7.3 Wnioski

Projekt udało się zrealizować pomimo napotkanych trudności, a ostateczna wersja aplikacji jest bezpieczna, wydajna i spełnia wymagania użytkowników. Kluczowe wnioski, które można wyciągnąć po realizacji projektu to:

1. Znaczenie testów bezpieczeństwa i audytów – regularne testowanie aplikacji pod kątem potencjalnych luk w prawidłowym działaniu funkcjonalności pozwala uniknąć poważnych problemów w przyszłości.
2. Optymalizacja wydajności – wdrażanie rozwiązań takich jak lazy loading czy stronicowanie już w początkowej fazie projektu pozwala uniknąć problemów związanych z wydajnością w późniejszych etapach pracy.
3. Elastyczność i rozwiązywanie problemów – proces tworzenia aplikacji nie jest wolny od wyzwań, ale każe z nich stanowi okazję do nauki, samodoskonalenia i wprowadzania ulepszeń.

Na podstawie doświadczeń i nauki z realizacji tego projektu, w przyszłości warto skupić się na jeszcze większej automatyzacji procesów testowania, zabezpieczeniu aplikacji przed atakami hakerskimi, procesie uwierzytelniania użytkowników za pomocą np. JWT Token oraz monitorowaniu aplikacji w czasie rzeczywistym.

Rozdział 8 Załączniki

8.1 Słownik terminów

Słownik terminów zawiera listę kluczowych pojęć, które są używane w dokumentacji aplikacji lub są istotne dla zrozumienia jej działania. Każdy termin jest jasno zdefiniowany, aby uniknąć nieporozumień, szczególnie w przypadku osób, które mogą nie być zaznajomione z technicznym językiem projektu.

- Framework – Zbiór narzędzi i zasad, które ułatwiają tworzenie aplikacji.
- Angular – Framework JavaScript stworzony przez Google, wykorzystywany do tworzenia interaktywnych aplikacji webowych typu Single Page Application.
- Backend – Część aplikacji, która odpowiada za logikę biznesową, przechowywanie danych oraz interakcję z bazą danych.
- Frontend – Część aplikacji, która jest widoczna od strony użytkownika. Pobiera informację od użytkownika i przekazuje je do backendu.
- Baza danych – Zbiór zorganizowanych informacji, które aplikacja wykorzystuje do przechowywania i pobierania danych.
- JSON Server – Narzędzie do szybkiego prototypowania backendu aplikacji. Umożliwia tworzenie prostych baz danych opartych na plikach JSON.
- Single Page Application (SPA) – Typ aplikacji webowej, w której cała aplikacja ładuję się na jedne stronie, a dane są dynamicznie ładowane w odpowiedzi na interakcje użytkownika bez przeładowania całej strony.
- UML (Unified Modeling Language) – Język do tworzenia wizualnych modeli, który jest wykorzystywany do przedstawiania architektury systemów.
- Modularność – Cecha aplikacji, która pozwala na dzielenie je na mniejsze, niezależne moduły.
- Moduł – W kontekście aplikacji, moduł to część aplikacji odpowiadająca za określoną funkcjonalność.
- Wymagania funkcjonalne – Szczegółowy opis funkcji, które aplikacja musi spełniać, aby mogła realizować potrzeby użytkownika.
- Wymagania niefunkcjonalne – Cechy systemu, które nie są związane bezpośrednio z funkcjonalnością, ale są ważne dla jakości.
- Przypadki użycia (Use-Cases) – konkretne scenariusze pokazujące, jak różne grupy użytkowników będą korzystać z aplikacji.

- Architektura klient-serwer – Model architektury systemu, w którym aplikacja jest podzielona na dwie główne części: klient (część odpowiedzialna za interakcję z użytkownikiem) oraz serwer: (część odpowiedzialna za przechowywanie danych i logikę biznesową).
- HTTP (HyperText Transfer Protocol) – Protokół komunikacyjny, który umożliwia przesyłanie danych między klientem a serwerem w aplikacji internetowych.
- Baza danych w formacie JSON – plik (db.json) w którym przechowywane są dane aplikacji JSON (JavaScript Object Notation) jest formatem danych, który jest łatwy do odczytu przez człowieka oraz maszynę.
- Warstwa logiki aplikacji – Warstwa, odpowiadająca za przetwarzanie danych i logikę działania systemu.
- Warstwa prezentacji – Część systemu odpowiedzialna za interakcję z użytkownikami.
- Warstwa przechowywania danych – Część systemu, która odpowiada za przechowywanie danych, w tym plik db.json, w którym zapisywane są dane aplikacji.
- Diagram klas – Diagram UML przedstawiający struktury klas w systemie oraz ich wzajemne relacje.
- Diagram komponentów – Diagram UML przedstawiający główne komponenty systemu oraz ich zależności i interakcje.
- Diagram sekwencji – Diagram UML przedstawiający kolejność zdarzeń i interakcji między obiektami w systemie w kontekście konkretnego procesu.
- REST API – Zestaw reguł i narzędzi umożliwiających komunikację pomiędzy klientem a serwerem.
- Struktura projektu – Organizacja plików oraz komponentów w aplikacji.
- Standalone – Podejście, w którym komponenty w Angularze są samodzielne i nie wymagają deklaracji w tradycyjnych modułach.
- NgModule – Tradycyjny sposób organizowania aplikacji Angulara, który wykorzystuje moduły do zarządzania komponentami, serwisami i innymi elementami aplikacji.
- Endpoint – Adres w API, do którego wysyłane są zapytania HTTP.
- JWT (JSON Web Token) – Standard tokenów wykorzystywanych do autoryzacji użytkowników w aplikacjach webowych.

- Testowanie jednostkowe – Proces weryfikacji działania pojedynczych funkcji, komponentów lub serwisów aplikacji, aby upewnić się, że działają one zgodnie z założeniami.
- Testowanie integracyjne – Testowanie współpracy pomiędzy różnymi komponentami systemu, zwłaszcza sprawdzenie poprawności komunikacji między frontendem a backendem.
- Testowanie systemowe – Testowanie aplikacji jako całości w warunkach przypominających rzeczywiste użycie, aby sprawdzić czy wszystkie funkcjonalności działają zgodnie z wymaganiami użytkownika końcowego.
- Karma + Jasmine – Narzędzie do testów jednostowych w Angularze. Karma uruchamia testy , a Jasmine jest frameworkiem testowym do pisania testów.
- Postman – Narzędzie do testowania API, które umożliwia wysyłania zapytań http do serwera i analizowanie odpowiedzi.
- Cypress – Narzędzie do testów systemowych, które pozwala na automatyczne testowanie aplikacji webowych w rzeczywistych warunkach przeglądarki.
- UX/UI – UX (User Experience) oznacza doświadczenie użytkownika, czyli ogólne wrażenia z korzystania z aplikacji, a UI (User Interface) to projektowanie interfejsu użytkownika, czyli widoczna część aplikacji.
- Lazy loading – Technika optymalizacji aplikacji, polegająca na ładowaniu zasobów (np. komponentów, modułów) tylko wtedy, gdy są one faktycznie potrzebne, co zmniejsza czas początkowego ładowania aplikacji.
- Cache`owanie – Mechanizm przechowywania często wykorzystywanych danych w pamięci podręcznej, co przyspiesza dostęp do nich i zwiększa wydajność aplikacji.
- CORS (Cross-Origin Resource Sharing) – Mechanizm, który umożliwia dostęp do zasobów na serwerze z różnych domen (np. frontend z jednej domeny, backend z innej).
- Angular Material – Biblioteka komponentów UI dla aplikacji Angularowych, która oferuje gotowe komponenty oparte na zasadach projektowania Material Design.
- Audyt aplikacji – Proces oceny aplikacji w celu identyfikacji problemów z jej bezpieczeństwem, wydajnością, użytecznością i innymi aspektami działania.

8.2 Lista skrótów

- API - Application Programming Interface
- CORS – Cross-Origin Resource Sharing
- CSS – Cascading Style Sheets
- DB – Database
- DOM – Document Object Model
- HTTP – HyperText Transfer Protocol
- JSON – JavaScript Object Notation
- JWT – JSON Web Token
- Karma – Narzędzie uruchamiające testy jednostkowe w Angularze
- JSON Server – Mockowy backend do testowania aplikacji
- REST – Representational State Transfer
- UI – User Interface
- UX – User Experience
- VCS – Version Control System
- IDE – Integrated Development Environment
- HTML – HyperText Markup Language
- CRUD – Create, Read, Update, Delete
- CLI – Command Line Interface
- SPA – Single Page Application
- TS – TypeScript
- JS – JavaScript

Rozdział 9 Bibliografia

9.1 Książki i podręczniki

- Wróblewski, P. (2018). Angular. Profesjonalne techniki. Helion
- Aristeidis B, Deeleman P . (2023). Poznaj Angular. Rzeczowy przewodnik po tworzeniu aplikacji webowych z użyciem frameworku Angular 15. Wydanie IV. Helion

9.2 Dokumentacje i materiały techniczne

- Angular Dev. Dokumentacja Angular. https://angular.dev/
- Angular Material Biblioteka Material https://material.angular.io/components/categories
- Node.js Dokumentacja Node.js. https://nodejs.org/docs/latest/api/
- JSON Server Dokumentacja json-server https://www.npmjs.com/package/json-server

9.3 Strony internetowe i blogi

- Wprowadzenie do Angulara. https://javappa.com/kurs-aplikacji-web/aplikacja-webowa-pakiet/wprowadzenie-angular
- Pierwsze kroki w Node.js https://justjoin.it/blog/pierwsze-kroki-w-node-js

Rozdział 10 Załączniki i wykorzystane narzędzia

10.1 Narzędzia do programowania i testowania

1. Visual Studio Code – edytor kodu źródłowego.
2. Angular CLI – narzędzie do zarządzania projektem Angular.
3. Node.js – środowisko uruchomieniowe JavaScript.
4. Postman – narzędzie do testowania API.
5. Karma + Jasmine – framework do testowania jednostkowego w Angularze.
6. Cypress – narzędzie do testowania systemowego i funkcjonalnego.
7. JSON Server – backend do obsługi danych,

10.2 Narzędzia do projektowania

1. Angular Material – biblioteka komponentów UI.

10.3 Narzędzia wspierające zarządzanie projektem

1. Git – system kontroli wersji.
2. GitHub – platforma do hostowania repozytoriów kodu.

10.4 Inne narzędzia

1. Microsoft Word – do pisania i edycji dokumentacji
2. StarUML – do tworzenia diagramów UML.

10.5 Załączniki

- Plik Projektu Angular – struktura aplikacji w formie kodu źródłowego. Dostępna w repozytorium GitHuB. Link : https://github.com/mateuszpoolski/app-inz
- Plik PDF o nazwie READ.ME – instrukcja do uruchomienia aplikacji.